The Identification of Foot-Strike Patterns and Prediction of Running Related Injuries - Configuration Manual

MSc Research Project
Data Analytics

Shane Gore
Student ID: x18174175

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shane Gore
Student ID:	x18174175
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr Catherine Mulwa
Submission Due Date:	17/08/2020
Project Title:	The Identification of Foot-Strike Patterns and Prediction of Running Related Injuries - Configuration Manual
Word Count:	15530
Page Count:	136

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

The Identification of Foot-Strike Patterns and Prediction of Running Related Injuries - Configuration Manual

Shane Gore
x18174175

1 Introduction

This configuration manual contains the detailed steps required to undertake the project entitled; "The Identification of Foot-Strike Patterns and Prediction of Running Related Injuries". The remaining document is structured as follows:

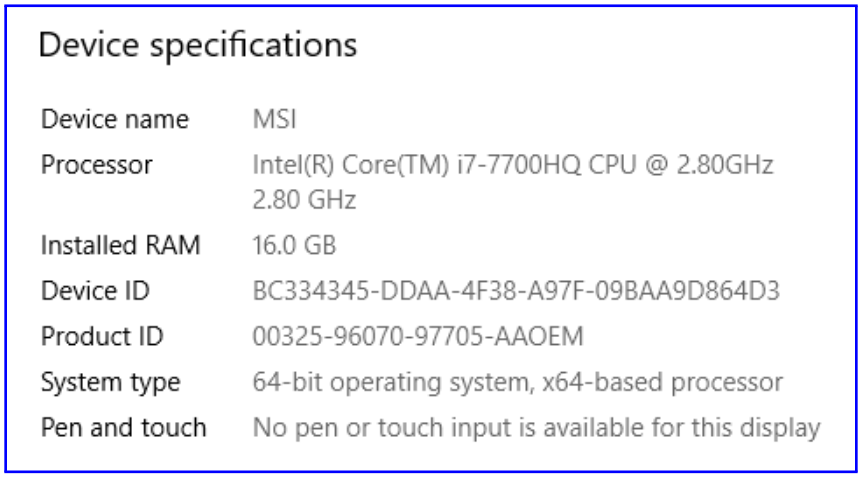
- Section 2 gives an overview of the enviromental configuration. This includes the hardware which was utilised for this project along with the software configurations for the primary tools utilised. These include; Vicon Nexus, MATLAB and Python.
- Section 3 presents the ICT implementation for the pre-processing steps. This includes the motion capture pre-processing, the general pre-processing and waveform screening.
- Section 4 presents the ICT implementation for the clustering solutions.
- Section 5 presents the ICT implementation for the predictive classification solutions.
- Section 6 presents addtional materal for the technical documentation. This includes additional literature review, methodologies and results.

2 Environmental Configuration

This section includes the description and setup required for the hardware and software utilised in this project.

2.1 Hardware Configuration

All data processing was tested on a Window 10 PC (Figure 1).



Device specifications

Device name	MSI
Processor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz
Installed RAM	16.0 GB
Device ID	BC334345-DDAA-4F38-A97F-09BAA9D864D3
Product ID	00325-96070-97705-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Windows PC device specification

2.2 Software Configurations

The primary softwares utilised in this project include Vicon Nexus (2.10, UK), MATLAB (R2018B, USA) and Python (3.7). Additionally, Microsoft Office applications (16.01); Word, Excel and PowerPoint were utilised along with TexWork 0.6.3 to produce the project documentation and Mendeley (1.19.5) was utilised as a reference manager.

2.2.1 Vicon Nexus

Vicon Nexus is the software utilised to capture and process the motion capture data. The following steps are required to install Nexus:

1. Navigate to the Vicon Website and locate the download section for Vicon Nexus (https://www.vicon.com/software/nexus/?section=downloads). Enter your email address and download the software (Figure 2).

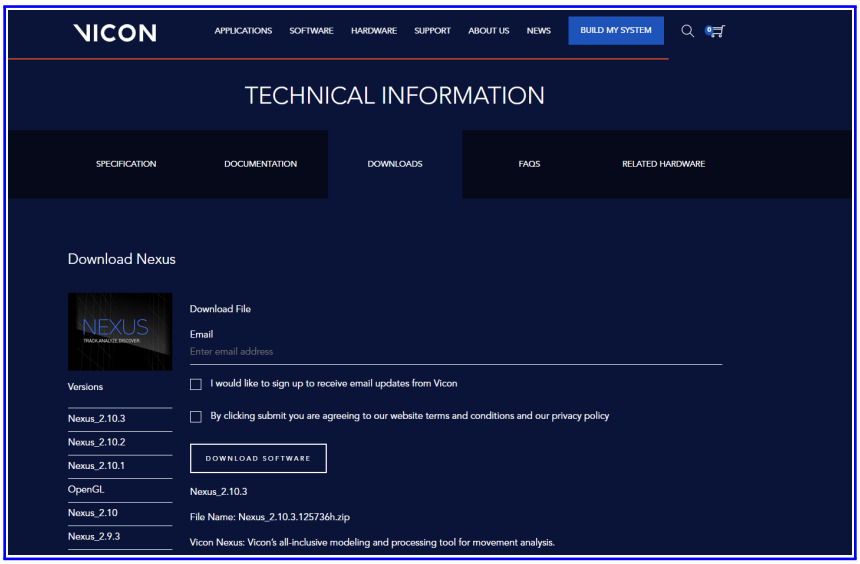
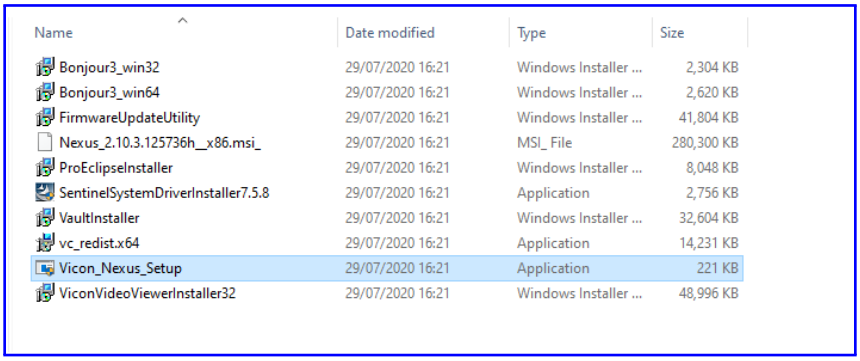


Figure 2: Vicon Nexus download location

2. After downloading and unzipping the files, double click on the set up application (Figure 3).



Name	Date modified	Type	Size
Bonjour3_win32	29/07/2020 16:21	Windows Installer ...	2,304 KB
Bonjour3_win64	29/07/2020 16:21	Windows Installer ...	2,620 KB
FirmwareUpdateUtility	29/07/2020 16:21	Windows Installer ...	41,804 KB
Nexus_2.10.3.125736h__x86.msi_	29/07/2020 16:21	MSI_ File	280,300 KB
ProEclipseInstaller	29/07/2020 16:21	Windows Installer ...	8,048 KB
SentinelSystemDriverInstaller7.5.8	29/07/2020 16:21	Application	2,756 KB
VaultInstaller	29/07/2020 16:21	Windows Installer ...	32,604 KB
vc_redist.x64	29/07/2020 16:21	Application	14,231 KB
Vicon_Nexus_Setup	29/07/2020 16:21	Application	221 KB
ViconVideoViewerInstaller32	29/07/2020 16:21	Windows Installer ...	48,996 KB

Figure 3: Launch set up wizard

3. This will launch the set-up wizard. Press Next (Figure 4).

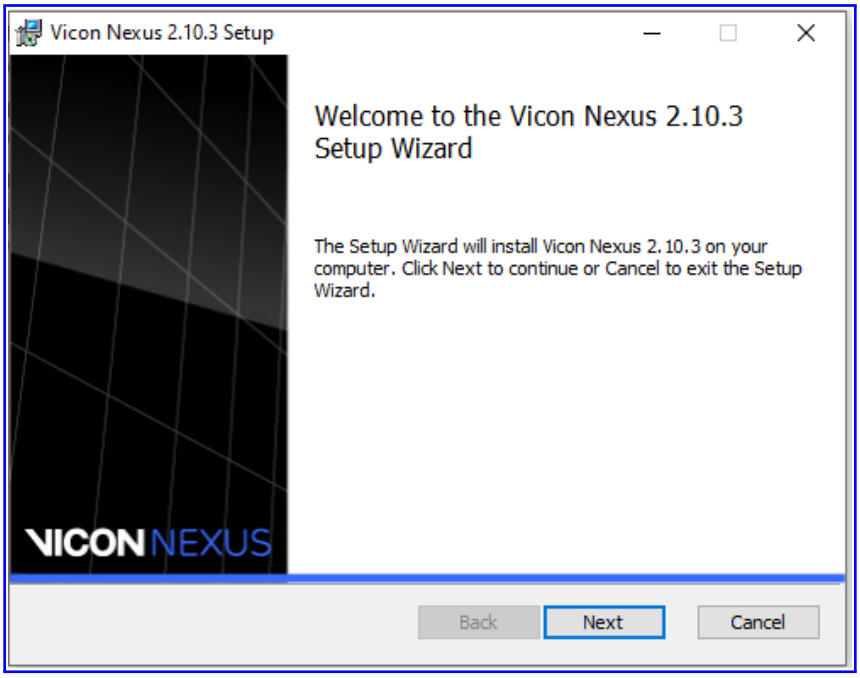


Figure 4: First step of set up

4. Agree to the terms of the licence to continue and press Next (Figure 5).

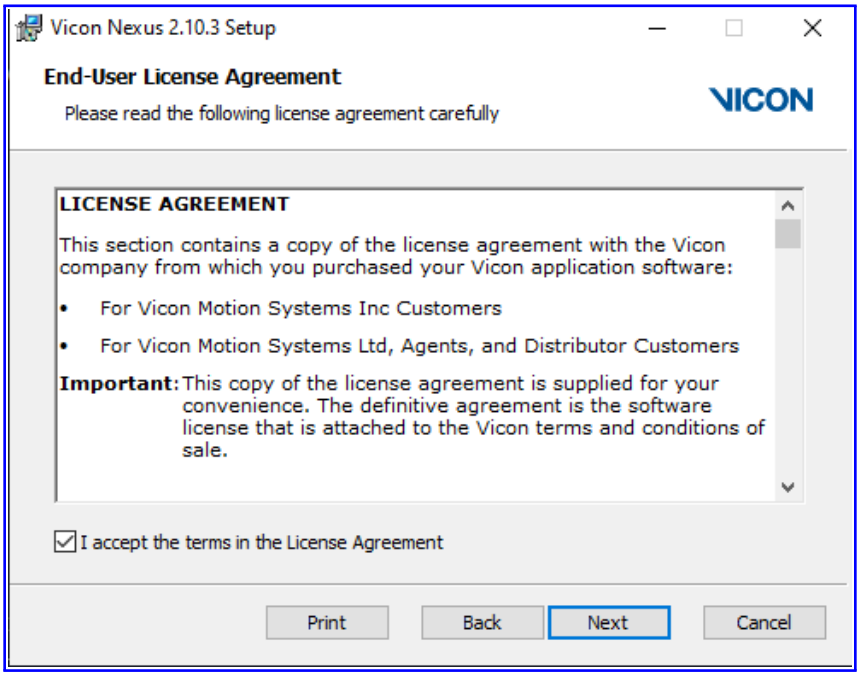


Figure 5: Licence agreement

5. Press Install to finish the installation process (Figure 6).

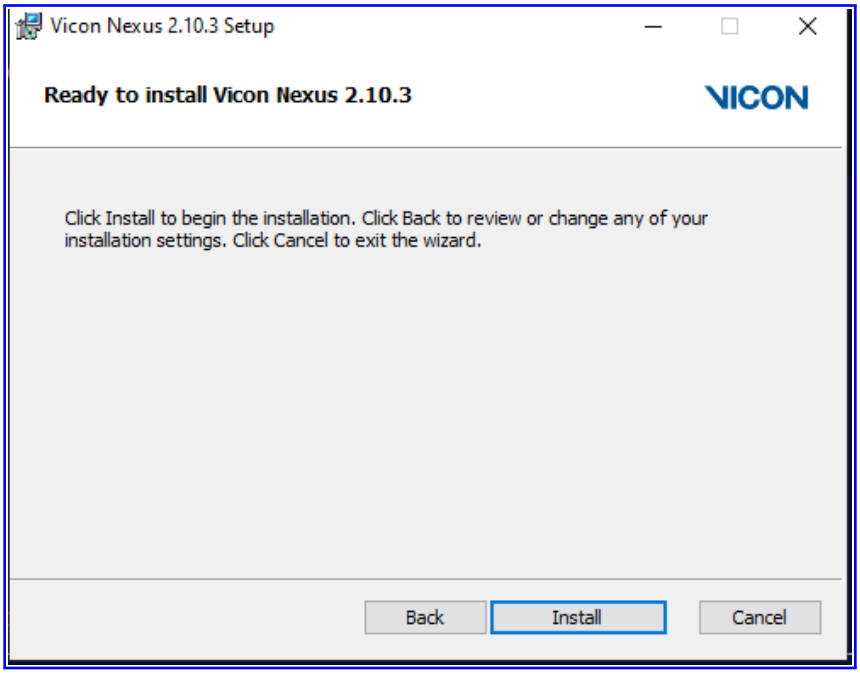


Figure 6: Final installation step

2.2.2 MATLAB

Within this project, MATLAB was utilised in extraction of the biomechanical data from the C3D files created by the Vicon Nexus Software, event detection and stride segmentation. To install MATLAB, the following steps are required.

1. Navigate to the website (https://uk.mathworks.com/downloads/) and download the installation of MATLAB. In this project, MATLAB (2018B) was utilised (Figure 7).

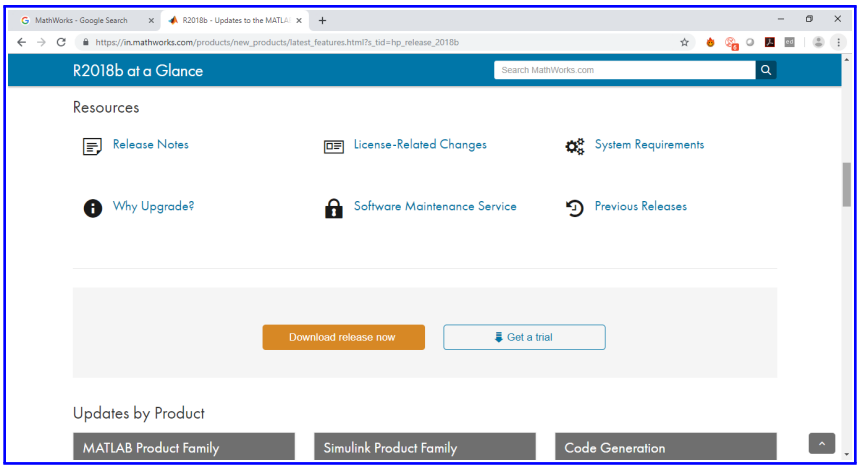


Figure 7: MATLAB download location

2. Once downloaded and unzipped, double click the setup and login with your MathWorks Account or use an Installation Key (Figure 8).

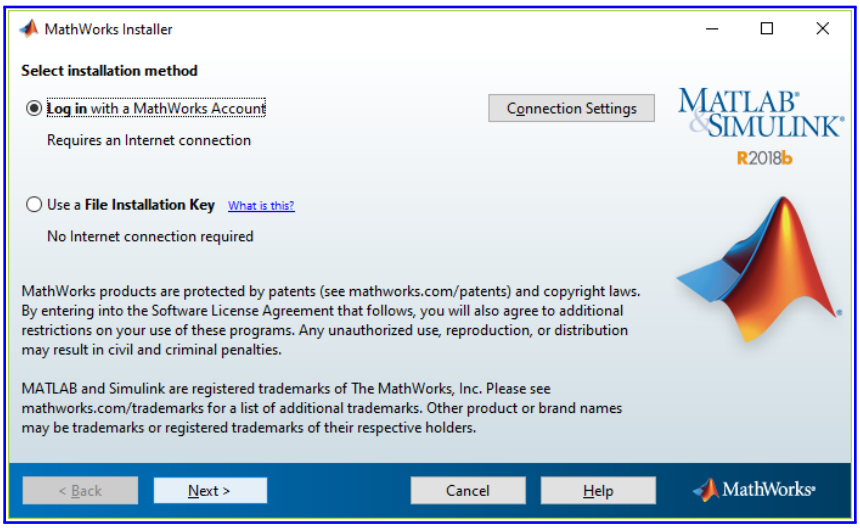


Figure 8: First step of installation

3. To continue with the installation, you will be required to accept the licence agreement (Figure 9).

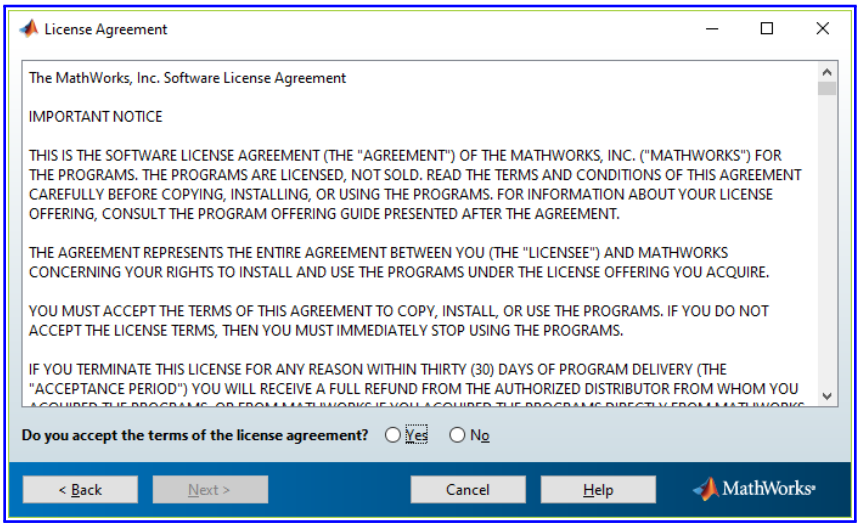


Figure 9: Licence agreement

4. Choose the installation location and press next to install (Figure 10).

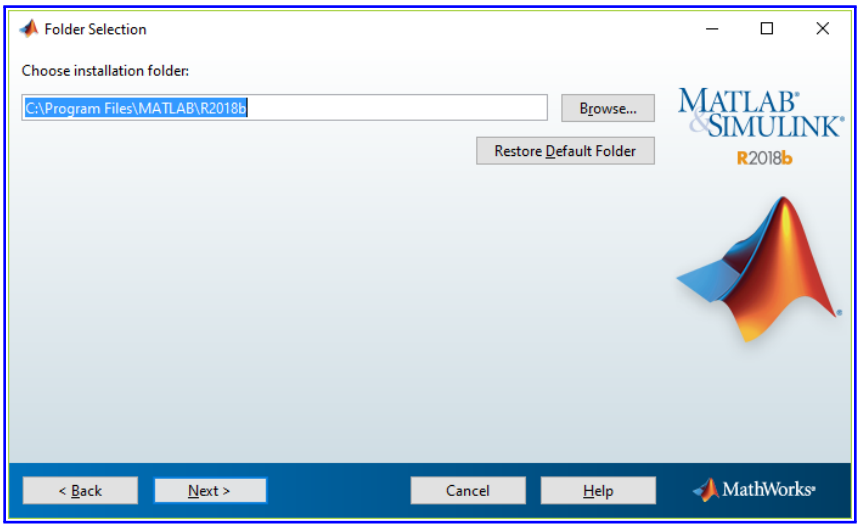


Figure 10: InstallationlLocation

2.2.3 Python

Within this project, Python was utilised for most of the ICT solution. This included, data pre-processing, feature engineering, feature selection, data modelling and visualisation. To install python, the following steps are required.

1. Python was installed with the Anaconda Distribution. Navigate to the Anaconda website (https://www.anaconda.com/products/individual) and download the 3.6 Distribution for the OS required. Within this current project, python was downloaded Windows (Figure 11).

Figure 11: Python download location

2. Once downloaded, double click on the file to launch to the set up (Figure 12).

Figure 12: First step of installation

3. Agree to the licence agreement to proceed (Figure 13).

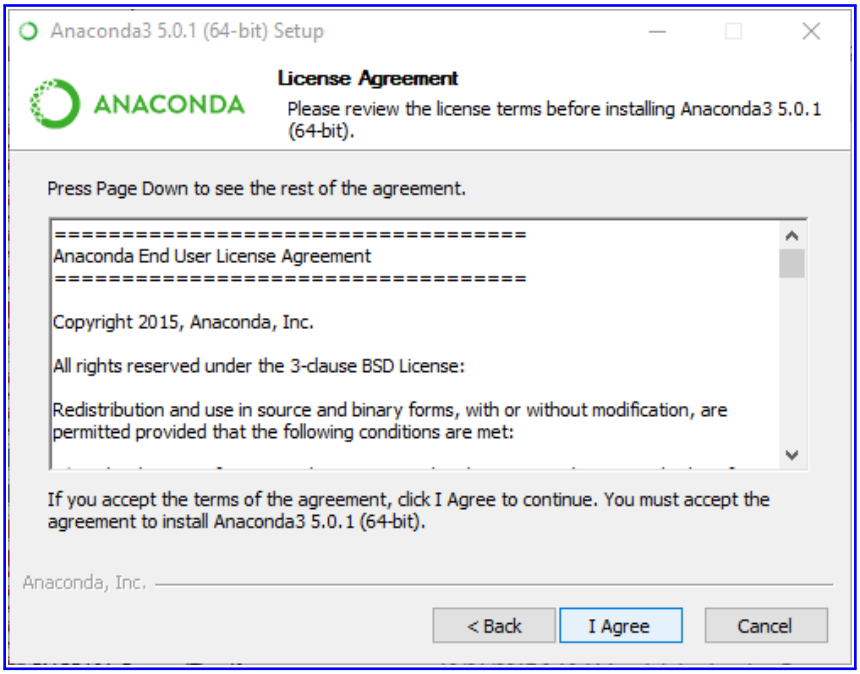


Figure 13: Licence agreement

4. To finish installation, register Anaconda as the default Python (Figure 14).

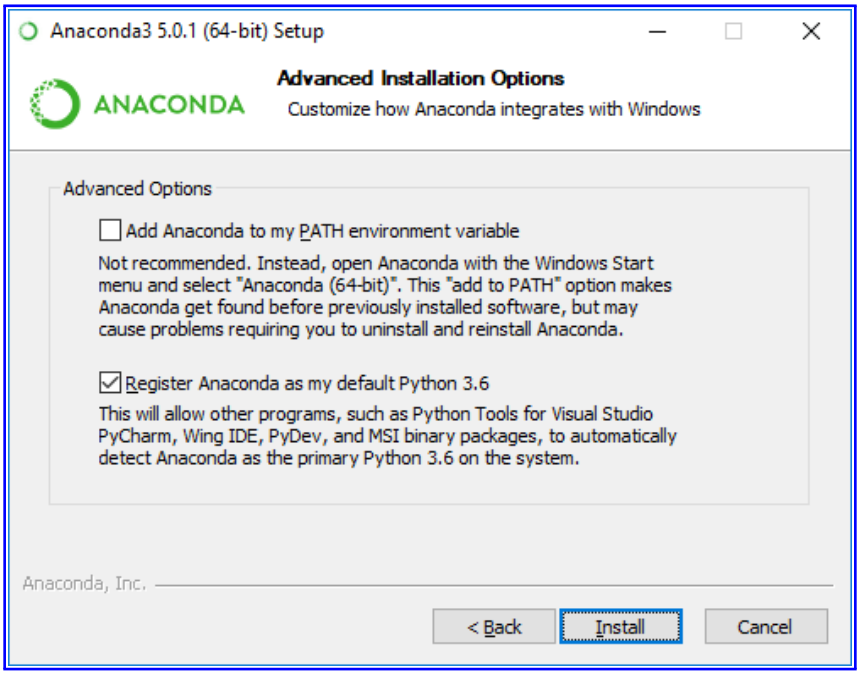


Figure 14: Final step of installation

3 ICT Implementation (Preprocesing)

This section details the preprocessing steps that were conducted as part of this project. This includes:

- Motion capture pre-processing which involves the tracking, modelling and gap filling of motion capture data using Vicon Nexus software.
- Data extraction and segmentation into stance phases.
- Waveform screening using a custom application.
- General data pre-processing which includes landmark registration, feature generation, screening for outliers and imputation.

3.1 Motion Capture Preprocessing

Motion capture data was preprocessed using Vicon Nexus software. After data capture, marker trajectories taken from 16 cameras in a calibrated space are used to create a three-dimensional representation of the marker position. A biomechanical model called 'PlugIn Gait' is then applied to this raw trajectory data, allowing the calculation of joint centres and axes of rotation. However, before applying this model, it is essential to track and fill gaps in the maker trajectories.

A screenshot of the main interface is provided below with some key buttons indicated (Figure 15). For a full description on the use of Vicon NEXUS software, please see the Vicon NEXUS manual available online [1].

[1] https://docs.vicon.com/display/Nexus29/Vicon+Nexus+User+Guide

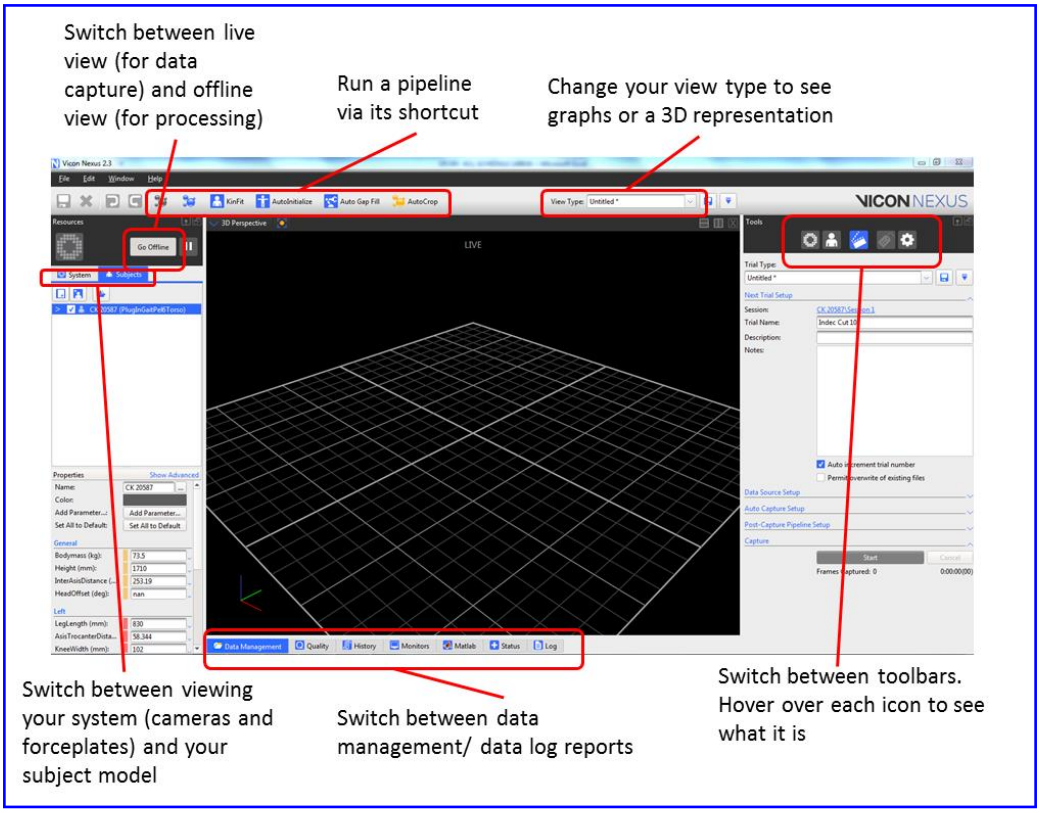


Figure 15: General layout of Vicon Nexus Software

3.1.1 Static trial processing

The first step involves processing the static trial. To do so, press F2 and double click the static trial in the data managament tab (Figure 16).

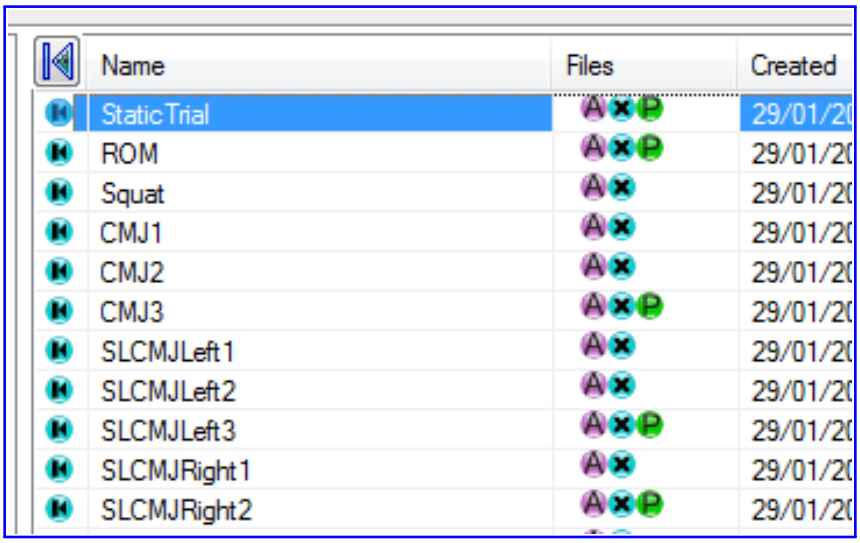
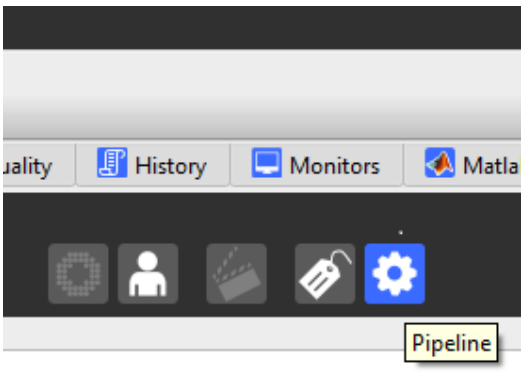
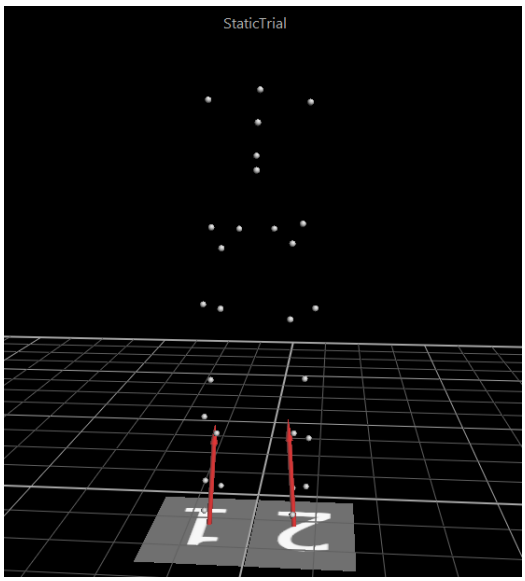


Figure 16: Data managment tab

Next, select the pipeline tab on the tools bar and run the reconstruct pipeline. This will reconstruct the marker trajectories in 3D space (Figure 17).



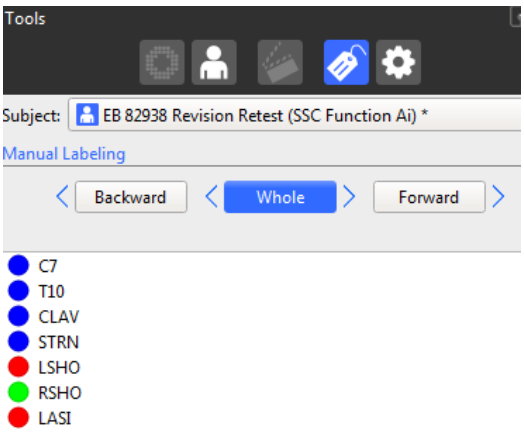
(a) Pipeline tab where the reconstruct pipeline will be found



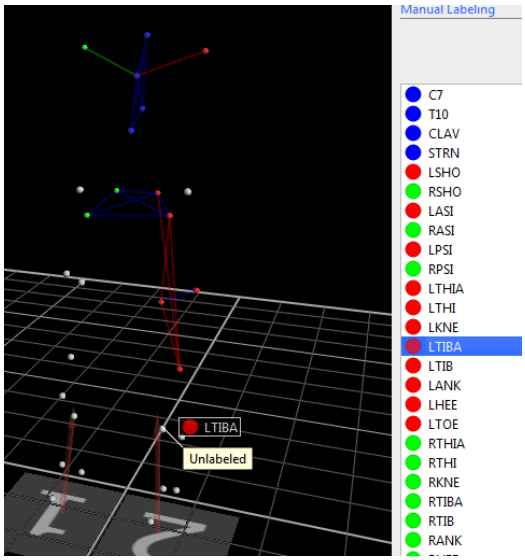
(b) Reconstructed markers in 3D space

Figure 17: Procedure to reconstruct the unlabelled maker trajectories

Using the predefined markers associated with the 'Plugin Gait' model, label the the figure (Figure 18). This involves, right clicking on an available marker and subsequently right clicking on the unlabelled trajectory to assign it.



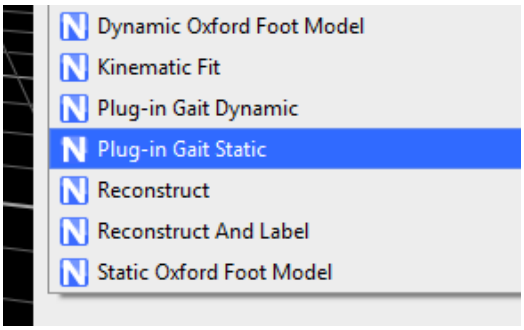
(a) Marker labels associated with the biomechanical model



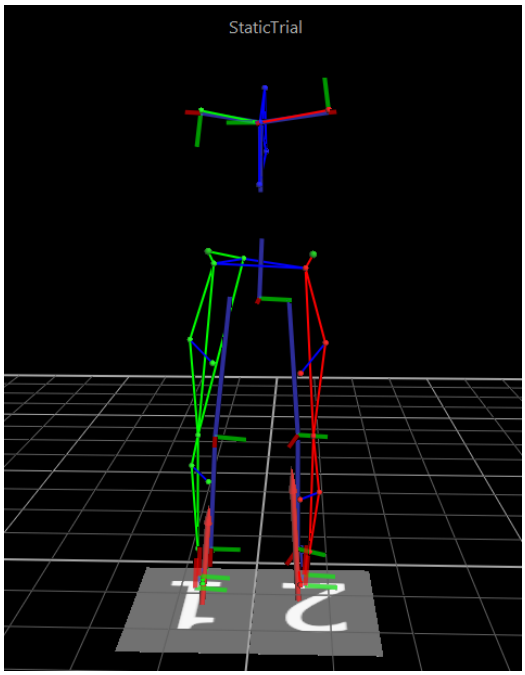
(b) Static trial in the process of being labelled

Figure 18: Procedure to label the unlabelled maker trajectories

Once the trajectories have been labelled, run the 'Plug-in Gait Static' pipeline in the pipelines tab on the tools bar. This will model the data, and inform Nexus of the marker positions (Figure 19).



(a) Static PugIn Gait pipeline



(b) Modelled Static Trial

Figure 19: Procedure to model the reconstructed and labelled maker trajectories

3.1.2 Dynamic trial processing

After modelling the static trial, the dynamic trials can be processed. This involves using pipelines to automaticallly reconstruct and label the marker trajectories, after which, the data is screened and gaps are filled. To view a specific dynamic trial, press F2 and double click on the trial to view the data. To produce tracker marker trajectories, press the 'Reconstruct and Label' icon. This will reconstruct the positions of the markers in 3D space from the 2D camera images and apply the model marker template to the marker positions (Figure 20).

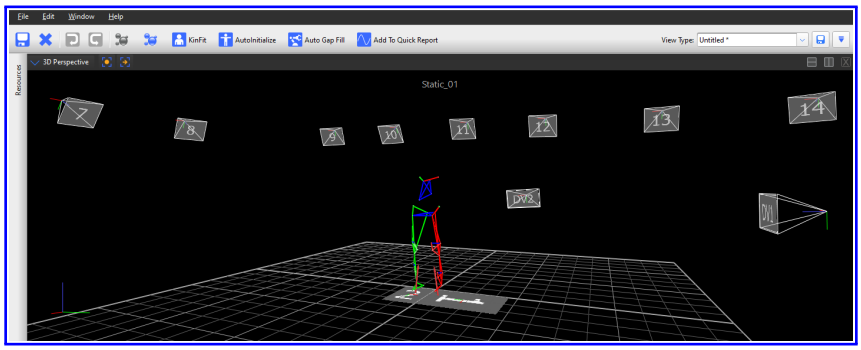


Figure 20: Reconstructed and labelled makers in 3D space

After reconstructing the marker trajectories, it is useful to be able to view the data from multiple perspectives. Key controls for the 3D space are provided in Table 1.

Table 1: Key controls to manipulate the 3D perspective.

Outcome	**Required Action**
Dolly/Zoom:	Right-click and drag forward or backward.
Orbit:	Left-click and drag left, right, forward, or backward.
Truck/Translate:	Click right and left simultaneously and drag.

To correct a mislabelled trajectory, select the trajectory at the first instant it becomes mislabelled or begins an incorrect trajectory and manually re-label it by selecting the appropriate marker label from the tools pane (Figure 21).

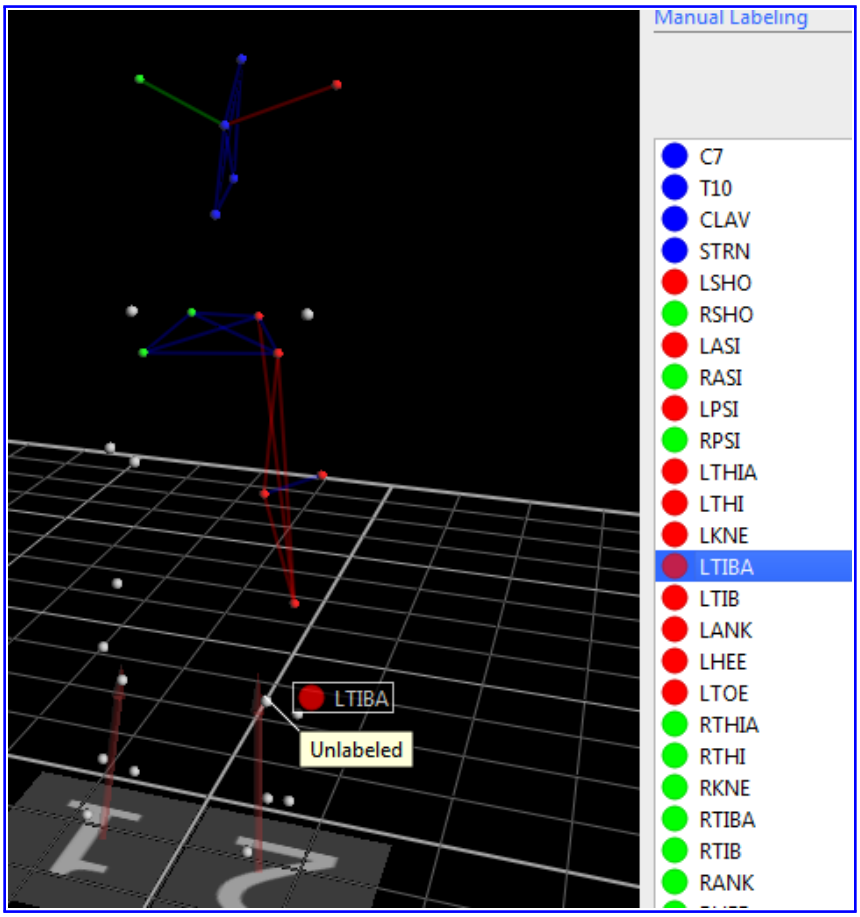


Figure 21: Manually relabel any mislabelled trajectories

If there are any gaps in a labelled trajectory, these will be listed by the marker name the gap occurs in (LASI in this example) (Figure 22).

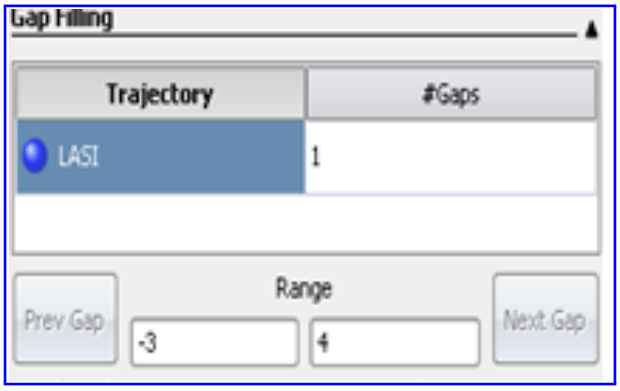


Figure 22: Indication of gaps in labelled trajectories

Selecting the name of the trajectory in the list causes the workspace to zoom to the area of the trajectory containing the gap (Figure 23).

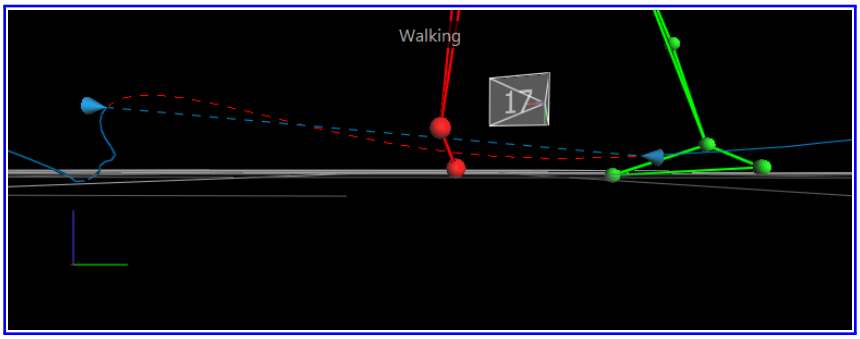


Figure 23: Graphical representation of the gap in the labelled trajectory

There are five potential methods of gap-filling a trajectory in Vicon Nexus which are presented in the labels tab of the tools bar (Figure 24). These are:

- Spline Fill: Extrapolates the missing trajectory based on the last known and first reappearing coordinates.
- Pattern Fill: Uses the trajectory of another manually selected marker and fills the gap on the assumption that the missing marker follows the same pattern of movement as the selected source marker.
- Rigid Body Fill: Uses the trajectory of three other manually selected markers and fills the gap on the assumption that the three source markers and the missing marker are located on the same rigid body.
- Kinematic Fill: Approximates where a missing trajectory is on the assumption that it is located on a particular segment selected from the Resources menu on the left and defined sufficiently by other markers.
- Cyclic Fill: Used for cyclic actions such as running and fills the gap as the likely position of the marker based on previous repeating cycles.

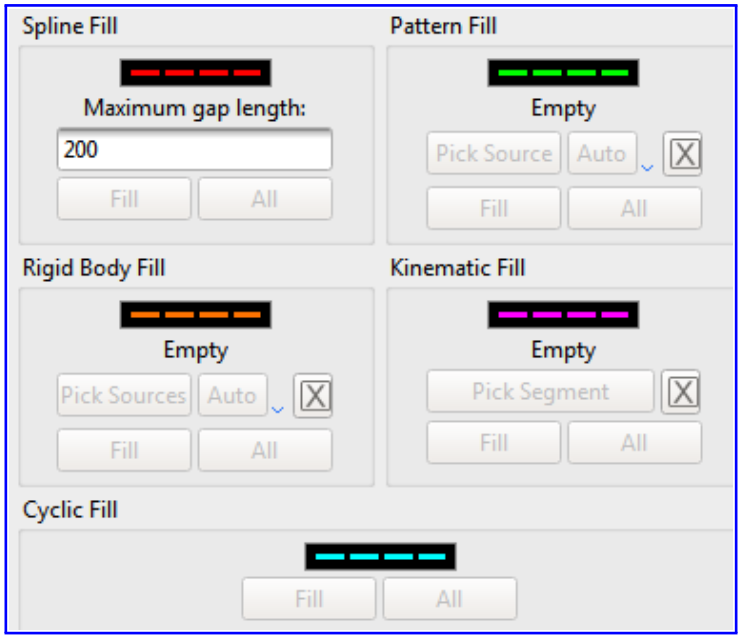
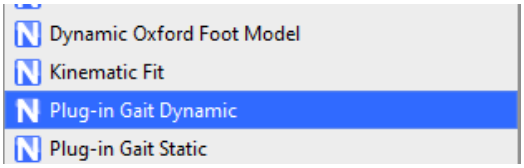
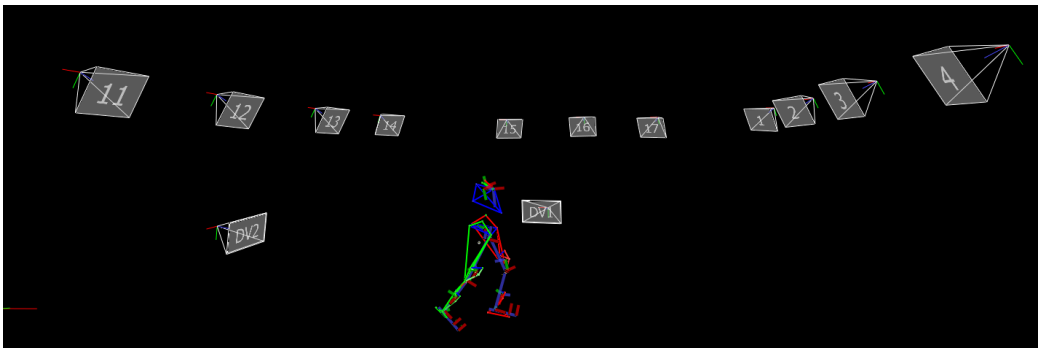


Figure 24: Gap fill functions available in Nexus

Once the marker trajectories have been tracked and gap filled, the dynamic trials can be modellled using the 'PlugIn Gait' model and exported to C3d file format for further analysis.



(a) Dynamic PugIn Gait pipeline



(b) Modelled Dynamic Trial

Figure 25: Procedure to model the dynamic trials

3.2 Data extraction and segmentation into stance phases

The data preprocessing begins from a script. After moving C3d files into a single folder, and defining the raw file locations, the various properties of the primary class to extract the data is defined in the configeration file (Figure 26). This latter process is usefull for rapid testing of the code.

```
% This is a script for extracting, processing and segmenting of motion capture
% data for the MSc project entitled 'The identification of foot-strike patterns
%using unsupervised learning and their association with injury'

% Written by Shane Gore 2020 contact Shane.Gore2@gmail.com

% Extract C3D files to a folder
extract =  ExtractC3D2folder_RISC;
extract.origin = 'F:\DCU RISC Study\Mocap';
extract.exercise = {'Vue','static'};
extract.output = 'F:\DCU RISC Study\Extracted C3D Vue';
extract.Extractdata

%%---------------------Define File locations------------------------------%%
cd 'C:\Users\Shane Gore\Desktop\DCU RISC Study'
InputFolders = {'F:\RISC'};
SubjectIdx   = {'C:\Users\Shane Gore\Desktop\DCU RISC Study\Codes\Subject_Index'};
ExcludeIdx   = {'C:\Users\Shane Gore\Desktop\DCU RISC Study\Codes\exclude_index'};

%%-------------------Set Config options---------------------------------%%
%Define variables to extract in config file:
open('get_biomech_config_RISC')
```

Figure 26: Script to set up data and prepare for further analysis

An example excerpt for the config file is presented in Figure (27).

```
function config = get_biomech_config_RISC(caseString)
%GET_BIOMECH_CONFIG This function updates object properties
%   For each of the sections below change the number from off (0) to on (1)
%   where the parameter is to be examined.

% Written by Shane Gore

switch (caseString)

%%-----------------------------------------------------------------------%%
    case 'Exercise'
        % Define exercises to be examined
        Exercise.HH  = 0;
        Exercise.Cut = 0;
        Exercise.SLDL = 0;
        Exercise.SLDJ = 0;
        Exercise.Run  = 1;
        Exercise.Hopping = 0;
        Exercise.SLCMJ = 0;
        Exercise.Ex45BE = 0;
        Exercise.CMJ = 0;
        Exercise.DLDJ = 0;          %#ok<*STRNU>

%%-----------------------------------------------------------------------%%

    case 'Variables'
        %Define Joints to be extracted
        Joints.Thorax = 1;
        Joints.Pelvis = 1;
        Joints.Hip = 1;
        Joints.Knee = 1;
        Joints.Ankle = 1;
        Joints.Foot = 1;
        %Define variable type to be extracted
        Extract.marker  = 0;
        Extract.Angles  = 1;
        Extract.Velocity  = 1;
        Extract.Acceleration  = 1;
        Extract.AbsAngles  = 0;
        Extract.Moment = 0;
        Extract.Power  = 0;
        Extract.Work    = 0;
        Extract.GRF     = 0;
        Extract.COMpower = 0;
```

Figure 27: Excerpt from the config file defining the test, segments and metrics to extract

The data is then extracted in the class `DataExtract_3DMOCAP_MSc` with the properties as defined by the config file (Figure 27) and exported for futher analysis in python (Figure 28).

```
%%------------------Extract the data from C3d files-----------------------%%
Extract = DataExtract_3DMOCAP_MSc;
Objproperties = properties(Extract);
for i = 1:size(Objproperties,1) %#ok<FORPF>
    Extract.(Objproperties{i}) = get_biomech_config_RISC(Objproperties{i});
end
Extract.Inputorigin = {'F:\DCU RISC Study\Extracted C3D Vue\Baseline'};
Extract.SubjectIdx  = SubjectIdx;
Extract.ExcludeIdx  = ExcludeIdx;
[ProblemData,Data]  = Extract.ExtractdatafromC3D;

%%------------------Join files and export for analysis-----------------------%%
%join conditions tables
names = fieldnamesr(Data);
conditions_t = cell(0,5);
for i =1:size(names,1)
    conditions = split(names{i},'.');
    conditions = repmat(conditions',size(eval(['Data.',names{i}]),2),1);
    conditions = [conditions,(eval(['Data.',names{i},'.Properties.VariableNames']))'];
    conditions_t = [conditions_t;conditions];
end

%join data tables
cdata = [];
for i =1:size(names,1)
    cdata =  [cdata;table2array(eval(['Data.',names{i}]))'];
end

%export data
csvwrite('Vuedata_stance_28_06.csv',cdata)
T = cell2table(conditions_t);
writetable(T,'Vuedata_index_stance_28_06.csv')
```

Figure 28: Data is extracted from the C3d files and exported for further analysis

The following sections will detail the ‘`DataExtract_3DMOCAP_MSc`’ class utilsied to extract the biomechanics data (Figure 29).

```
classdef DataExtract_3DMOCAP_MSc
    %DATAEXTRACT_3DMOCAP Extracts 3D mocap data from C3D file.
    %This class was written to extract data from the mocap data based
    %on the variables indicated in a config file 'get_biomech_config_RISC'.
    %Data is extracted and normalised to 101 data points using a parallel
    %computing paradigm.

    %Written by Shane Gore 2020. Contact Shane.Gore2@gmail.com

    properties
        Groups          = '' % List of groups to be examined
        Exercise        = '' % List of exercises to be examined
        Inputorigin     = 'none' % Location of folder containing data
        SubjectIdx      = 'none' % Location of subject index
        Events          = '' % List of events to examine the data within
        Variables       = '' % List of variables to be examined
        FunctJoints     = true % Determines if functional joints are used.
        Combined        = true % Determines if force plare are combined.
        SetUp           = 'RISC' % Define some default setup values
        ExcludeIdx      = '' % Location of list of trials to exclude.
    end

    methods
```

Figure 29: Class for extracting biomechanical data

Before extracting data from the running trials, the static trials were analysed (see section 3.2.1 for details on the function). When exploring the running triials, the first steps essentially involve extracting information from the file names and controlling for variations in the mocap format (Figure 30 and 31)

```
%% -------- Dynamic Trials ------- %%

files = getAllFiles(foldername);
files = files(~cellfun('isempty',regexp(files,'.c3d'))); % Delimit to .c3d files
files = files(cellfun('isempty',regexpi(files,'static'))); %remove static trials

%open up workers for parrell loop
if isempty(gcp('nocreate'))
    parpool(3)
end

%Preallocate persistant arrays for indexing when storing data.
variables = control.Variables;
idx1 = zeros(size(variables,2),3);
idx2 = zeros(size(variables,2),3);

AllExercises = cell(''); %Create Empty Cell to hold Exercies list.

%Loop through relivent dynamic trial files and extract data.
n = 0; h = waitbar(0,['Processing Dynamic Trials...',num2str(n),' of ',num2str(size(files,1))]);
for n = 1:size(files,1)
    waitbar(n/size(files,1),h,['Processing Dynamic Trials...',num2str(n),' of ',num2str(size(files,1))])
    file    = files(n);

    % set up current subject string
    idx = strfind(file{1},filesep);
    trial_id = file{1}(idx(size(idx,2))+1:end);
    sub_idx = strfind(trial_id,'_');
    Csub = trial_id(1:sub_idx(2)-1);

    %Exclude subjects not in subject index
    IDX = readtable(control.SubjectIdx{1});
    if sum(strcmp(Csub,IDX.SubName)) ~= 1
        continue
    end

    %Exclude data if in the Exculde index
    exIDX = readtable(control.ExcludeIdx{1});
    if sum(strcmp(file,exIDX.Trial_Name)) ~= 0
        continue
    end

    % Add Direction to variable labels where required
    Varidx = (strcmp (variables,'GroundReactionForce')|....
        strcmp(variables,'CentreofMass')|....
        strcmp(variables,'CentreOfMassPower')|....
        strcmp(variables,'STRN')|....
        strcmp(variables,'CLAV')|....
        strcmp(variables,'T10')|....
        strcmp(variables,'C7')|....
        strcmp(variables,'LASI')|....
        strcmp(variables,'RASI')|....
        strcmp(variables,'LSHO')|....
        strcmp(variables,'RSHO')|....
        strcmp(variables,'RPSI')|....
        strcmp(variables,'LPSI'));
```

Figure 30: Extracting file information and controlling for format variations (1 of 2)

```
% Identify tested limb from file name.
if contains(trial_id,'Right')
    Dir = {'R'};
elseif contains(trial_id,'Left')
    Dir = {'L'};
else
    Dir = {'R','L'};
end

for nn = 1:size(Dir,2)
    c_Dir = Dir{nn};
    NewVariables = variables;
    DirIDX = find(Varidx==0);

    % Control for Combined Forceplates
    if strcmp(control.Combined,'true')
        NewVariables(strcmp (NewVariables,'GroundReactionForce')) = {[c_Dir,'GroundReactionForce_M']};
    end

    %Control for functional joints
    if strcmp(control.FunctJoints,'true')
        for f = 1:size(DirIDX,2)
            NewVariables(1,DirIDX(f)) = {[c_Dir,variables{DirIDX(f)},'_M']};
        end
    else
        for f = 1:size(DirIDX,2)
            NewVariables(1,DirIDX(f)) = {[c_Dir,variables{DirIDX(f)}]};
        end
    end
```

Figure 31: Extracting file information and controlling for format variations (2 of 2)

The data was then extracted from the C3d file using the biomechanics toolkit (Barre and Armand; 2014) (Figure 32).

```
%Extract data from c3d file and down sample force plate to mocap
file     = files(n);
c3dfile     = btkReadAcquisition(file{1});
c3d.points      = btkGetPoints(c3dfile);
c3d.plates      = btkGetAnalogs(c3dfile);
c3d.ratio       = btkGetAnalogSampleNumberPerFrame(c3dfile);
c3d.labels      = fieldnames(c3d.plates);
c3d.events      = btkGetEvents(c3dfile);
c3d.frequency   = btkGetPointFrequency(c3dfile);
c3d.firstframe = btkGetFirstFrame(c3dfile);
for is  = 1:btkGetAnalogNumber(c3dfile)
    c3d.plates.(c3d.labels{is}) = c3d.plates.(c3d.labels{is})(1:c3d.ratio:end);
end
btkCloseAcquisition(c3dfile);

%Control for data not processed
if  ~isfield(c3d.points,'LHEE')
    warning([trial_id ,'_not_reconstructed'])
    ProblemData(find(cellfun('isempty',(ProblemData)),1,'first'),1)    = {[file{1},'_','Reconstruct']};
    continue
end
```

Figure 32: Extracting data from the C3d file

Afterwhich, the extracted data was then segmented by identifying the events of foot-strike and toe-off (Figure 33). For full details on the custom class utilised to identify these events, please see section 3.2.2.

```
% Set up new events
if any(strcmpi(control.Exercise,'run'))
    control.Exercise = {'run','_9k','_Self'};
    if sum(~cellfun('isempty', regexpi(trial_id,control.Exercise))) >0
        CurrentExercise = 'Run';
    end
else
    CurrentExercise = control.Exercise(~cellfun('isempty', regexpi(trial_id,control.Exercise)));
end


%Define events allowing for multiple events.
DefineEvent                 = EVENTS_3DMOCAP_RISC;
DefineEvent.CurrentEvent    = control.Events;
DefineEvent.c3d             = c3d;
DefineEvent.file            = file;
DefineEvent.SubjectIdx      = control.SubjectIdx;
DefineEvent.Combined        = control.Combined;
DefineEvent.Exercise        = CurrentExercise;
DefineEvent.SubQS           = SubQS;
DefineEvent.side            = c_Dir;
DefineEvent.StaticFoot      = StaticFoot;

try
    [Event1,Event2,~]     = DefineEvent.Define3DEvents;
catch
    warning([file{1} ,'_Events','_can not process'])
    ProblemData(find(cellfun('isempty',(ProblemData)),1,'first'),1)    = {[file{1},'_','Events']};
    continue
end
```

Figure 33: Identifying events for segementing into stance phases

Given the variations possible in the naming of the motion capture data, the files were classified using a combination of exact and fuzzy matching using Levenshtein distance (Figure 34)

```
% Define Type of run with exact and fuzzy matching critera.
teststring =  trial_id(1:strfind(trial_id,'.c3d')-1);
teststring = strsplit(teststring,{'_', ' '}); % Split strings on delimiter
if contains(trial_id,'9km','IgnoreCase',true) && contains(trial_id,'Vue','IgnoreCase',true)...
        && sum(cellfun(@(x) strdist(x,'Baseline'),teststring) < 3) == 1
    CurrentExercise = ['Baseline_Run_9kmhr_Vue']; %#ok<*NBRAK>
elseif contains(trial_id,'9km','IgnoreCase',true) && ~contains(trial_id,'Vue','IgnoreCase',true)...
        && sum(cellfun(@(x) strdist(x,'Baseline'),teststring) < 3) == 1
    CurrentExercise = ['Baseline_Run_9kmhr'];
elseif  ~contains(trial_id,'9km','IgnoreCase',true) && ~contains(trial_id,'Vue','IgnoreCase',true)...
        && sum(cellfun(@(x) strdist(x,'Post'),teststring) < 2) == 1 && sum(cellfun(@(x) strdist(x,'Self')...
        ,teststring) < 2) == 1
    CurrentExercise = ['Post_Run_SelfSelected'];
elseif  contains(trial_id,'9km','IgnoreCase',true) && ~contains(trial_id,'Vue','IgnoreCase',true)...
        && sum(cellfun(@(x) strdist(x,'Post'),teststring) < 2) == 1
    CurrentExercise = ['Post_Run_9kmhr'];
elseif  ~contains(trial_id,'9km','IgnoreCase',true) && ~contains(trial_id,'Vue','IgnoreCase',true)...
        &&  sum(cellfun(@(x) strdist(x,'Baseline'),teststring) < 3) == 1 && sum(cellfun(@(x) strdist(x,'Post')...
        ,teststring) > 1) ~= 0
    CurrentExercise = ['Baseline_Run_SelfSelected'];
elseif  ~contains(trial_id,'9km','IgnoreCase',true) && ~contains(trial_id,'Vue','IgnoreCase',true)...
        &&  sum(cellfun(@(x) strdist(x,'Baseline'),teststring) < 3) == 1 && sum(cellfun(@(x) strdist(x,'Post')...
        ,teststring) < 2) == 1
    CurrentExercise = ['Post_Run_9kmhr'];
elseif  ~contains(trial_id,'9km','IgnoreCase',true) &&  sum(cellfun(@(x) strdist(x,'Post'),teststring) < 2) == 1
    CurrentExercise = ['Post_Run_SelfSelected'];
else
    warning([file{1} ,'_Name','_can not process'])
    ProblemData(find(cellfun('isempty',(ProblemData)),1,'first'),1)   = {[file{1},'_','NodefName']};
    continue
end

if isempty(AllExercises)
    AllExercises{1,1} = CurrentExercise;  %#ok<*AGROW>
elseif sum(strcmp(AllExercises,CurrentExercise)) == 0
    AllExercises = [AllExercises;CurrentExercise];
end

%Specify Events depending on type
if strcmp(c_type,'stance')
    c_Event1 = Event1;
    c_Event2 = Event2;
else
    c_Event1 = Event1(1:end-1);
    c_Event2 = Event1(2:end);
end
```

Figure 34: Classifying motion capture files using exact and fuzzy name matching

Additional biomechanical metrics were then derived from the raw motion capture data for the foot (Figure 35).

```
%Derive metrics and normalise data using parrell computing
temp_norm_cell = cell(size(NewVariables,2),3);
c_points = struct;
try
    pointnames = fieldnames(points);
    for x = 1:3 %Seperate data into planes
        for v = 1:size(pointnames,1)
            if sum(~cellfun('isempty',regexp(pointnames{v},{'HEE','TOE'}))) > 0
                c_points.([pointnames{v}])(:,:) =  points.([pointnames{v}])(:,:);
            else
                c_points.([pointnames{v}])(:,1) =  points.([pointnames{v}])(:,x);
            end
        end

        parfor v = 1:size(NewVariables,2)
            if ~isempty(regexp(NewVariables{v},'Foot', 'once'))
                if x ~= 1 %planes not required.
                    continue
                else
                    %Extract Footstrike angle
                    if strcmp(c_Dir,'L')
                        StaticCalc = table2array(StaticFoot.Left(~cellfun('isempty',...
                            regexp(file,table2cell(StaticFoot.Left(:,1)))),2)); %#ok<*PFBNS>
                    else
                        StaticCalc = table2array(StaticFoot.Right(~cellfun('isempty',...
                            regexp(file,table2cell(StaticFoot.Right(:,1)))),2));
                    end
                    hoz = (c_points.([c_Dir,'TOE'])(:,1) -  c_points.([c_Dir,'HEE'])(:,1));
                    vert = (c_points.([c_Dir,'TOE'])(:,3) - c_points.([c_Dir,'HEE'])(:,3));
                    angle = atan2d(vert,hoz);
                    if ~isempty(regexp(NewVariables{v},'FootVelocity', 'once'))
                        angle   = angle - StaticCalc;
                        paddata = angle(1,1)- (angle(2,1)- angle(1,1)); % Pad data to retain signal length
                        cdata = ([paddata;diff(angle(:,1))])./(1/200);
                    elseif ~isempty(regexp(NewVariables{v},'FootAcceleration', 'once'))
                        angle   = angle - StaticCalc;
                        paddata = angle(1,1)- (angle(2,1)- angle(1,1)); % Pad data to retain signal length
                        cdata = ([paddata;diff(angle(:,1))])./(1/200);
                        paddata = cdata(1,1)- (cdata(2,1)- cdata(1,1)); % Pad data to retain signal length
                        cdata = ([paddata;diff(cdata(:,1))])./(1/200);
                    else
                        cdata =  angle - StaticCalc;
                    end
                end
```

Figure 35: Deriving foot metrics

This was also conducted for the remaining extracted segments which were then normalised to 101 datapoints using a cubic spline to represent 100% of the stance phase between the foot-strike and toe-off events (Figure 36).

```
    elseif contains(NewVariables{v},'Work','IgnoreCase',true) %Calculate joint work done.
        angle  = c_points.([NewVariables{v}(1:strfind(NewVariables{v},'Work')-1),'Angles']);
        moment = c_points.([NewVariables{v}(1:strfind(NewVariables{v},'Work')-1),'Moment']);
        paddata = c_points.(NewVariables{v})(1,1)- ...
            (c_points.(NewVariables{v})(2,1)- c_points.(NewVariables{v})(1,1)); % Pad data to retain signal length
        deltaAngle = [paddata;diff(angle(:,1))];
        cdata = (moment(:,1)./1000) .* deltaAngle;
    elseif contains(NewVariables{v},'Moment','IgnoreCase',true)
        cdata  = c_points.([NewVariables{v}])(:,1)./1000; %Convert to Newton Meters
    elseif  contains(NewVariables{v},'Velocity','IgnoreCase',true)
        angle  = c_points.([NewVariables{v}(1:strfind(NewVariables{v},'Velocity')-1),'Angles']);
        paddata = angle(1,1)- (angle(2,1)- angle(1,1)); % Pad data to retain signal length
        cdata = ([paddata;diff(angle(:,1))])./(1/200);
    elseif  contains(NewVariables{v},'Acceleration','IgnoreCase',true)
        angle  = c_points.([NewVariables{v}(1:strfind(NewVariables{v},'Acceleration')-1),'Angles']);
        paddata = angle(1,1)- (angle(2,1)- angle(1,1)); % Pad data to retain signal length
        cdata = ([paddata;diff(angle(:,1))])./(1/200);
        paddata = cdata(1,1)- (cdata(2,1)- cdata(1,1)); % Pad data to retain signal length
        cdata = ([paddata;diff(cdata(:,1))])./(1/200);
    elseif ~isempty(regexp(NewVariables{v},'Foot', 'once'))
        continue
    else
        cdata  = c_points.([NewVariables{v}])(:,1);
    end

    %Split norm data into trials
    c_norm = zeros(101,size(c_Event1,1));%preallocate with zeros
    for c = 1:size(c_Event1,1)
        c_norm(:,c) = norm2frame(cdata(c_Event1(c):c_Event2(c),:),101);
    end
    temp_norm_cell{v,x} = c_norm;
end
```

Figure 36: Deriving additional biomechanical metrics and normalising to 101 datapoints

The final steps involved storing the data to export. The size of the data being examined in this project excluded the use heterogeneous data structures due increasing time complexity. As such the numeric data and their string identifiers were stored separately in homogeneous arrays (Figure 37) .

```
%set up column names.
a = repmat({[Csub,'_',c_Dir,'_',c_type]},1,size(c_Event1,2));
b = num2cell(1:size(c_Event1,1));
b = cellfun(@num2str,b,'un',0);
colnames = strcat(a,b);
for v =1:size(NewVariables,2)
    %Set up axis labels
    if ~isempty(regexp(NewVariables{v},'GroundReactionForce', 'once')) || ...
            size(NewVariables{v},2) < 5 || ...
            strcmp(NewVariables{v},'CentreOfMass')
        planes = {'_x','_y','_z'};
    else
        planes = {'_fle','_abd','_rot'};

    end

    %Store data in a structure
    for x = 1:3
        %Account for if there are duplicates in the names
        try
            prefix = {'B_','C_','D_','E_','F_'};
            while any(~cellfun('isempty',regexp(colnames(1),...
                    (Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x},'_colnames']))))))
                a = repmat({[Csub,prefix{1},c_Dir,'_',c_type]},1,size(c_Event1,2));
                colnames = strcat(a,b);
                prefix(1) =[];
            end
        catch
            %do nothing
        end

        if size(temp_norm_cell{v,x},2) > 0
            try
                if isfield(Data.Norm.(CurrentExercise).(c_type),([variables{v},planes{x}]))
                    idx1(v,x) = idx2(v,x) + 1; idx2(v,x) = idx2(v,x) + size(temp_norm_cell{v,x},2);
                    Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x},'_colnames'])...
                        (:,idx1(v,x):idx2(v,x)) = colnames;
                    Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x}])...
                        (:,idx1(v,x):idx2(v,x)) = temp_norm_cell{v,x};
                else
                    idx1(v,x) = 1; idx2(v,x) = size(temp_norm_cell{v,x},2);
                    Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x}]) = NaN(101,1e5);
                    Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x}])...
                        (:,idx1(v,x):idx2(v,x)) = temp_norm_cell{v,x};
                    Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x},'_colnames']) =  cell(1,1e5);
                    Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x},'_colnames'])...
                        (:,idx1(v,x):idx2(v,x)) = colnames;
                end
            catch
                idx1(v,x) = 1; idx2(v,x) = size(temp_norm_cell{v,x},2);
                Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x}]) = NaN(101,1e5);
                Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x}])...
                    (:,idx1(v,x):idx2(v,x)) = temp_norm_cell{v,x};
                Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x},'_colnames']) =  cell(1,1e5);
                Data.Norm.(CurrentExercise).(c_type).([variables{v},planes{x},'_colnames'])...
                    (:,idx1(v,x):idx2(v,x)) = colnames;
            end
        end
    end

end
```

Figure 37: Storing data to export

Similarly, given the time and memory cost of dynamically growing arrays, they were excessively pre-allocated with NaNs. After the data had been stored in the structures, the excess NaNs were removed and the structures were exported for further analysis (Figure 38) .

```
% remove NaNs
for c = 1:size(AllExercises,1)
    for v =1:size(NewVariables,2)
        for x = 1:size(planes,2)
            if  isfield(Data.Norm.(AllExercises{c}).(c_type),([variables{v},planes{x}]))
                Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x}])...
                    = Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x}])...
                    (:,~arrayfun(@isnan, Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x}])(1,:)));
                Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x},'_colnames'])...
                    = Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x},'_colnames'])...
                    (:,~cellfun('isempty', Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x},'_colnames']))
            end
        end
    end
end

% Convert numeric array to table for
% convience later.
for c = 1:size(AllExercises,1)
    for v =1:size(NewVariables,2)
        for x = 1:size(planes,2)
            if isfield(Data.Norm.(AllExercises{c}).(c_type),([variables{v},planes{x}]))
                Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x}])...
                    = array2table( Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x}]),'VariableNames',...
                    Data.Norm.(AllExercises{c}).(c_type).([variables{v},planes{x},'_colnames']));
                Data.Norm.(AllExercises{c}).(c_type)...
                    = rmfield(Data.Norm.(AllExercises{c}).(c_type),([variables{v},planes{x},'_colnames']));
            end
        end
    end
end
```

Figure 38: Removing excess rows in the data and exporting for analysis

3.2.1 Static Trial Metrics

This section depicts the function to extract useful metrics from the static trial when the subject is standing still (Figure 39 and 40). Theses include quiet standing force and angle of the foot when in a neutral position. The later point is particularly important to this project, as it is essential for calculating the angle of the foot while running (Altman and Davis; 2012).

```
function  [ProblemData ,StaticFoot ,SubQS] = extract_static(foldername,ProblemData)
%EXTRACT_STATIC: This function extracts useful metrics from the static
%trial while the subject is  standing still.
folder = dir(foldername);
folder = folder(arrayfun(@(x) x.name(1), folder) ~= '.');
folder = folder(~cellfun('isempty',regexp({folder.name},'Static')));
h = waitbar(0,'Processing Static Trials...');
for n = 1:size(folder,1)
    waitbar(n/size(folder,1),h)
    try
        %Extract data from C3D
        file    = folder(n).name;
        c3dfile    = btkReadAcquisition([foldername,filesep,file]);
        plates     = btkGetAnalogs(c3dfile);
        points     = btkGetPoints(c3dfile);
        btkCloseAcquisition(c3dfile);

        %Calculate Quiet Standing Force
        QS = mean(plates.Force_Fz1 + plates.Force_Fz2);
        QS_res = QS + 1.7 * max(abs(mean(plates.Force_Fz1) - plates.Force_Fz1));

        %Calculate the foot anlge during static
        hoz = median(points.('LTOE')(:,1)) - median(points.('LHEE')(:,1));
        vert = median(points.('LTOE')(:,3)) - median(points.('LHEE')(:,3));
        staticAngleL = (atan(vert/hoz))* 360 /(2 * pi);
        hoz = median(points.('RTOE')(:,1)) - median(points.('RHEE')(:,1));
        vert = median(points.('RTOE')(:,3)) - median(points.('RHEE')(:,3));
        staticAngleR = (atan(vert/hoz))* 360 /(2 * pi);
        StanddataL = table;
        StanddataR = table;
```

Figure 39: Function to extract useful metrics from the static trial (1 of 2)

```
        if ~exist('StaticFoot','var')
            idx = strfind(file,'_');
            StanddataL.Name = file(1:idx(2)-1);
            StanddataL.staticAngle =  staticAngleL;
            StaticFoot.('Left') = StanddataL;
            StanddataR.Name = file(1:idx(2)-1);
            StanddataR.staticAngle =  staticAngleR;
            StaticFoot.('Right') = StanddataR;
            StaticFoot.('RightToe') = [table(file(1:idx(2)-1),median(points.('RTOE')(:,3)))] ;
            StaticFoot.('LeftToe') =[table(file(1:idx(2)-1),median(points.('LTOE')(:,3)))] ;
            StaticFoot.('RightHee') = [table(file(1:idx(2)-1),median(points.('RHEE')(:,3)))];
            StaticFoot.('LeftHee') = [table(file(1:idx(2)-1),median(points.('LHEE')(:,3)))] ;
        else
            idx = strfind(file,'_');
            StanddataL.Name = file(1:idx(2)-1);
            StanddataL.staticAngle =  staticAngleL;
            StaticFoot.('Left') =  [StaticFoot.('Left');StanddataL];
            StanddataR.Name = file(1:idx(2)-1);
            StanddataR.staticAngle =  staticAngleR;
            StaticFoot.('Right')    = [StaticFoot.('Right');StanddataR];
            StaticFoot.('RightToe') = [StaticFoot.('RightToe');...
                [table(file(1:idx(2)-1),median(points.('RTOE')(:,3)))]] ;
            StaticFoot.('LeftToe') =  [StaticFoot.('LeftToe');...
                [table(file(1:idx(2)-1),median(points.('LTOE')(:,3)))]] ;
            StaticFoot.('RightHee') = [StaticFoot.('RightHee');...
                [table(file(1:idx(2)-1),median(points.('RHEE')(:,3)))]] ;
            StaticFoot.('LeftHee') =  [StaticFoot.('LeftHee');...
                [table(file(1:idx(2)-1),median(points.('LHEE')(:,3)))]] ;
        end
        Subdata = table;
        if ~exist('SubQS','var')
            idx = strfind(file,'_');
            Subdata.Name = file(1:idx(2)-1);
            Subdata.QS =  QS;
            Subdata.QS_res = QS_res;
            SubQS = Subdata;
        else
            idx = strfind(file,'_');
            Subdata.Name = file(1:idx(2)-1);
            Subdata.QS =  QS;
            Subdata.QS_res = QS_res;
            SubQS =  [SubQS;Subdata];
        end
    catch
        warning([file ,'_Static','_can not process'])
        ProblemData(find(cellfun('isempty',(ProblemData)),1,'first'),1)  = {[file,'_','Static']};
    end
end
delete(h) % delete static waitbar
end
```

Figure 40: Function to extract useful metrics from the static trial (2 of 2)

3.2.2 Event Detection

Within this project, data was collected, continiously over a one minute period. In order to segment the biomechanical data into stance phases, the events of inital contact and toe-off had to be detected. Event detection was coded as a self contained class (Figure 41).

```
classdef EVENTS_3DMOCAP_RISC
    %EVENTS_3DMOCAP Defines events in motion capture data, to allow the
    %segmentation of key movements and/or tasks.

    %Written by Shane Gore. Contact Shane.Gore2@gmsil.com

    properties
        CurrentEvent    = '' % List of groups to be examined;
        Exercise        = '' % List of exercises to be examined
        file            = '' % List of events to examine the data within;
        SubjectIdx      = '' % The subject index (data dictionary)
        Combined        = '' % A boolean determining if the plates have been combined.
        SubQS           = {} % A cell array of subject quiet standing.
        side            = '' % A string indicating the side currently being examined.
        c3d             = '' % A structure containing C3d data
        StaticFoot      = '' % A cell array containing info from the static trial

    end

    methods
        function [Event1,Event2,plate ]= Define3DEvents(control)
```

Figure 41: Event detection class

Initial foot contact was defined by firstly identifying a window in which the ankle marker was within 10cm of its local minima (Figure 42).

```
%% -- Events for running -- %%
plate = 1; %arbitary value (must be defined)

% Find where Ankle is at its minimum height
[Peakvals,~,~,proms]  = findpeaks(((control.c3d.points.([control.side,'ANK'])(:,3)) .*-1),...
    'MinPeakDistance',10); % Find where marker is at its lowest.
Peakvals = sort(Peakvals);
Peakvals = Peakvals(round(size(Peakvals,1)/4):round(size(Peakvals,1)/2));
[~,WindowEnd_marker]  = findpeaks(((control.c3d.points.([control.side,'ANK'])(:,3)) .*-1),...
    'MinPeakDistance',50,'MinPeakHeight',(median(Peakvals) *1.12),...
    'MinPeakHeight',(median(Peakvals) *1.10),'MinPeakProminence',(median(proms) *0.80));
```

Figure 42: Identifying a search window defined by the height of the ankle marker

Heel contact was then subjsequently defined using the peak negative horizontal velocity of heel (Figure 43).

```
%------ Calculate usefull marker metrics for heel -------%
marker_m = (control.c3d.points.([control.side,'HEE'])(:,:))./1000;
paddata = marker_m(1,:)-(marker_m(2,:)-marker_m(1,:)); % Pad data to retain signal length
marker_vel =  (diff([paddata;marker_m]))./(1/control.c3d.frequency);

% this section checks for double peaks within 60 frames. if there
% is more than one peak within 60 frames, it checks for
% peak height and if there is a difference takes the
% largest, otherwise takes the second peak.
idx = zeros(size(WindowEnd_marker,1),1);
for ii = 1:size(WindowEnd_marker,1) -1
    if WindowEnd_marker(ii + 1) - WindowEnd_marker(ii) < 60 %events within 60 frames
        thresh = max(marker_m(WindowEnd_marker(ii + 1)),marker_m(WindowEnd_marker(ii))) *.05; %within 5%
        if   abs(marker_m(WindowEnd_marker(ii + 1))- marker_m(WindowEnd_marker(ii))) > abs(thresh)
            if marker_m(WindowEnd_marker(ii + 1)) > marker_m(WindowEnd_marker(ii))
                idx(ii) =1;
            else
                idx(ii + 1) =1;
            end
        else
            idx(ii + 1) =1;
        end
    end
end
WindowEnd_marker(logical(idx)) = [];

% Remove early and late peaks - within 40 frames of start and end
WindowEnd_marker(WindowEnd_marker < (control.c3d.frequency *0.2)) =[];
WindowEnd_marker(WindowEnd_marker + (control.c3d.frequency *0.2) > length(marker_m)) = [];

%Find first negative velocity of heel
markerstrike_vel =[];
for w = 1:size(WindowEnd_marker,1)
    if WindowEnd_marker(w) < double(control.c3d.frequency) *0.15
        offset =  WindowEnd_marker(w) - (WindowEnd_marker(w) - 1);
    else
        offset = (double(control.c3d.frequency) *0.15);
    end
    vel_idx = find(marker_vel((WindowEnd_marker(w) - offset):WindowEnd_marker(w),1) < 0,1) ;
    markerstrike_vel =  [markerstrike_vel;((WindowEnd_marker(w) - offset) + vel_idx -1)]; %#ok<AGROW>
end
HEE_markerstrike_vel = round(markerstrike_vel);
```

Figure 43: Identifying the first negative heel velocity

This was then repeated for the toe, and the first occuring event was defined as inital contact (Figure 44).

```
%------ Calculate usefull marker metrics for Toe -------%
marker_m = (control.c3d.points.([control.side,'TOE'])(:,:))./1000;
paddata = marker_m(1,:)-(marker_m(2,:)-marker_m(1,:)); % Pad data to retain signal length
marker_vel =  (diff([paddata;marker_m]))./(1/control.c3d.frequency);

% Find where Ankle is at its minimum height
[Peakvals,~,~,proms]  = findpeaks(((control.c3d.points.([control.side,'ANK'])(:,3)) .*-1),...
    'MinPeakDistance',10); % Find where marker is at its lowest.
Peakvals = sort(Peakvals);
Peakvals = Peakvals(round(size(Peakvals,1)/4):round(size(Peakvals,1)/2));
[~,WindowEnd_marker]  = findpeaks(((control.c3d.points.([control.side,'ANK'])(:,3)) .*-1),'MinPeakDistance',50,...
    'MinPeakHeight',(median(Peakvals) *1.13),'MinPeakProminence',(median(proms) *0.80));

% this section checks for double peaks within 60 frames.
idx = zeros(size(WindowEnd_marker,1),1);
for ii = 1:size(WindowEnd_marker,1) -1
    if WindowEnd_marker(ii + 1) - WindowEnd_marker(ii) < 60 %events within 60 frames
        thresh = max(marker_m(WindowEnd_marker(ii + 1)),marker_m(WindowEnd_marker(ii))) *.05; %within 5%
        if   abs(marker_m(WindowEnd_marker(ii + 1))- marker_m(WindowEnd_marker(ii))) > abs(thresh)
            if marker_m(WindowEnd_marker(ii + 1)) > marker_m(WindowEnd_marker(ii))
                idx(ii) =1;
            else
                idx(ii + 1) =1;
            end
        else
            idx(ii + 1) =1;
        end
    end
end
WindowEnd_marker(logical(idx)) = [];

% Remove early and late peaks - within 40 frames of start and end
WindowEnd_marker(WindowEnd_marker < (control.c3d.frequency *0.2)) =[];
WindowEnd_marker(WindowEnd_marker + (control.c3d.frequency *0.2) > length(marker_m)) = [];

markerstrike_vel = zeros(size(WindowEnd_marker));
for w = 1:size(WindowEnd_marker,1)
    if WindowEnd_marker(w) < double(control.c3d.frequency) *0.15
        offset =  WindowEnd_marker(w) - (WindowEnd_marker(w) - 1);
    else
        offset = (double(control.c3d.frequency) *0.15);
    end
    vel_idx = find(marker_vel((WindowEnd_marker(w) - offset):WindowEnd_marker(w),1) < 0,1) ;
    markerstrike_vel(w) = ((WindowEnd_marker(w) - offset) + vel_idx -1);
end
TOE_markerstrike_vel = round(markerstrike_vel);

% Pick Touchdown Event
Event1 = zeros(size(TOE_markerstrike_vel));
h_idx =  HEE_markerstrike_vel < TOE_markerstrike_vel;
Event1(h_idx) = HEE_markerstrike_vel(h_idx);
Event1(~h_idx) = TOE_markerstrike_vel(~h_idx);
```

Figure 44: Identifying the first negative toe velocity, and definfing initial contact

To identify toe off, a search window was again specified when the ankle marker was within 10cm of its local minima (Figure 45).

```
%----------------Calculate toe off -------------%
% usefull vertical movement metrics
marker_m = (control.c3d.points.([control.side,'TOE'])(:,3))./1000;
paddata = marker_m(1,:)-(marker_m(2,:)-marker_m(1,:)); % Pad data to retain signal length
marker_vel =  (diff([paddata;marker_m]))./(1/control.c3d.frequency);
paddata = marker_vel(1,:) - mean(diff(marker_vel(1:3,:)));
marker_acc =  (diff([paddata;marker_vel]))./(1/control.c3d.frequency);
paddata = marker_acc(1,:) - mean(diff(marker_acc(1:3,:)));% Pad data to retain signal length
marker_jerk =  (diff([paddata;marker_acc]))./(1/control.c3d.frequency);

[Peakvals,~,~,proms]  = findpeaks((((control.c3d.points.([control.side,'ANK'])(:,3))./1000) .*-1)...
    ,'MinPeakDistance',10); % Find where marker is at its lowest.
[~,WindowEnd_marker]  = findpeaks((((control.c3d.points.([control.side,'ANK'])(:,3))./1000) .*-1)...
    ,'MinPeakDistance',50,...
    'MinPeakHeight',(median(Peakvals) *1.12),'MinPeakProminence',(median(proms) *0.80));

% this section checks for double peaks within 60 frames.
idx = zeros(size(WindowEnd_marker,1),1);
for ii = 1:size(WindowEnd_marker,1) -1
    if WindowEnd_marker(ii + 1) - WindowEnd_marker(ii) < 60 %events within 60 frames.
        thresh = max(marker_m(WindowEnd_marker(ii + 1)),marker_m(WindowEnd_marker(ii))) *.05; %within 5%
        if   abs(marker_m(WindowEnd_marker(ii + 1))- marker_m(WindowEnd_marker(ii))) > abs(thresh)
            if marker_m(WindowEnd_marker(ii + 1)) > marker_m(WindowEnd_marker(ii))
                idx(ii) =1;
            else
                idx(ii + 1) =1;
            end
        else
            idx(ii + 1) =1;
        end
    end
end
WindowEnd_marker(logical(idx)) = [];

% Remove early and late peaks - within 40 frames of start and end
WindowEnd_marker(WindowEnd_marker < (control.c3d.frequency *0.2)) =[];
WindowEnd_marker(WindowEnd_marker + (control.c3d.frequency *0.2) > length(marker_m)) = [];
```

Figure 45: Identifying a search window defined by the height of the ankle marker

Toe off was defined using the toe jerk maxima (3rd derivative of toe marker position) following peak knee extension (Figure 46) as a combination of two previously published algorithms (Handsaker et al.; 2016; Dingwell et al.; 2001).

```
Event2 = zeros(size(WindowEnd_marker)); %preallocation
for w = 1:size(WindowEnd_marker,1)
    if w == size(WindowEnd_marker,1)
        min_toe_frames =   (WindowEnd_marker(w)-1): ((WindowEnd_marker(w)-1)...
            + find((marker_m(WindowEnd_marker(w):end)) > ...
            (min(marker_m(WindowEnd_marker(w):end)) + 0.05), 1 ));
        if isempty(min_toe_frames)
            % there is no toe off before the end so last footstrike is void.
            Event1(size(Event1,1)) = [];
            continue  % there is no toe off before the end.
        end
    else
        [~,min_idx] = min(marker_m(WindowEnd_marker(w):Event1(w+1)));
        min_toe_frames = (WindowEnd_marker(w)-1): (((WindowEnd_marker(w)+ min_idx)-1)...
            + find((marker_m((WindowEnd_marker(w)+ min_idx):Event1(w+1))) > ...
            (min(marker_m(WindowEnd_marker(w):Event1(w+1))) + 0.05), 1 ));
    end

    % find peak knee extension
    [~,min_idx] = min(control.c3d.points.([control.side,'KneeAngles'])(min_toe_frames));
    % find next max acceleration
    [~,acc_idx] = max(marker_acc(min_toe_frames(min_idx):min_toe_frames(end)));
    % Find maximum jerk between peak knee extension and acceleration
    [~,jerkidx] =  max(marker_jerk(min_toe_frames(min_idx):min_toe_frames(min_idx + acc_idx -1)));
    %if there are multiple peaks, take the earlest.
    maxjerk = min(jerkidx + min_idx -1 );
    Event2(w,1) = min_toe_frames(maxjerk);

end

%Ensure we have complete footstrikes.
if Event1(1) > Event2(1)
    Event2(1) = [];
end
if Event1(end) > Event2(end)
    Event1(end) = [];
end
```

Figure 46: Identifying toe off event

3.3 Waveform Screening

In order to assist with the initial screening of the biomechanical waveform data, a custom written application was developed in MATLAB (R2018B). After collecting biomechanical motion capture data, it can be challenging to identify when motion capture data has been modelled inappropriately. To overcome this challenge, this application was developed which enables the end user to rapidly visualise, interpret and delimit biomechanical waveforms for further investigation or correction via a graphical user interface (Figure 47).
Amongst its functionalities:

- It allows the user to read in motion capture time series data.
- Plot a user defined number of curves at a time.
- Zoom in, zoom out and pan on plots.
- Cycle forward or backward in the plotting of the data.
- Select and remove user identified inappropriate waveforms for further investigation.
- Use statistical measures to delimit the data being screened.

Within this current project, the latter point was particularly important given the large number of waveforms which had to be analysed.

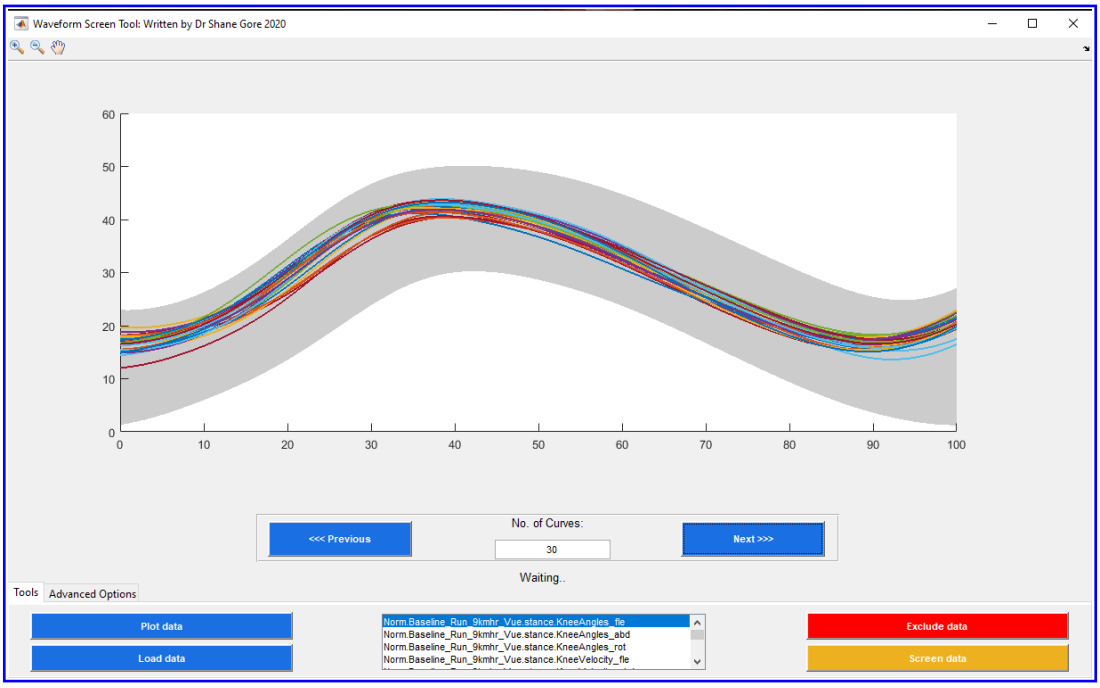


Figure 47: Custom application designed to assist in the screening of biomechanical data

3.3.1 Method: LaunchGui

This is a function which initialises the application. The appearance of the application is set and the various interactive features (e.g. push buttons) are created with callback functions to the object's other methods (Figure 48, Figure 49 and Figure 50).

```
function LaunchGui(control)

    %%%------Set up gui display-----%%%
    %create a figure to house the GUI
    Fig = figure('toolbar','none');
    set(Fig,'Name','Waveform Screen Tool: Written by Dr Shane Gore 2020',...
        'NumberTitle','off');

    %Utilise standard Matlab plot tools
    H = uitoolbar('parent',Fig);
    uitoolfactory(H,'Exploration.ZoomIn');
    uitoolfactory(H,'Exploration.ZoomOut');
    uitoolfactory(H,'Exploration.Pan');
    set(Fig ,'Menubar','none');

    %Set default size of figure window
    Fig.Position = [754.5,231,1051,618];

    %Set figure window position
    set(Fig,'Units', 'pixels' );
    %get your display size
    screenSize = get(0, 'ScreenSize');
    %calculate the center of the display
    position = get(Fig,'Position' );
    position(1) = (screenSize(3)-position(3))/2;
    position(2) = (screenSize(4)-position(4))/2;
    %center the window
    set(Fig,'Position', position );

    % Create a panel for gui items
    pnl = uitabgroup('Parent', Fig,'Units' ,'Normalized',...
        'Position',[0 0 1.0 0.16]);
    tab1 = uitab('Parent', pnl, 'Title', 'Tools');
    tab2 = uitab('Parent', pnl, 'Title', 'Advanced Options');

    % Create text to display code status
    Status_h =     uicontrol('Style','text',...,...
        'String','Current Status:Waiting',...
        'fontsize',10,'Units' ,'Normalized',...
        'Position',[0.244,0.141,0.5,0.035],...
        'Tag','UpdateGUI');

    % Create checkbox to select automatic screening techniques
    Auto_type_h = uipanel('Parent',tab2,'visible','on','units','normalized','Position',[0.02, 0.040 , 0.243, 0.43]);
    items = size(control.auto_types,2); %Number of items
    itemspace = (1/(items*1.1));
    for e = 1:size(control.auto_types,2)
        ExEv{e}= uicontrol('Style','Checkbox','String',control.auto_types{e},...
            'units','normalized','pos',[((itemspace*e)- 0.3), .3,  1, 0.5],...
            'parent',Auto_type_h,'HandleVisibility','on'); %#ok<AGROW>
    end

    % Create checkbox to determine how to load the data
    Load_type_h = uibuttongroup('Parent',tab2,'visible','on','units'...
        ,'normalized','Position',[0.3, 0.040 , 0.4,    0.43]);
    items = size(control.load_types,2); %Number of items
    itemspace = (1/(items*1.1));
    for e = 1:size(control.load_types,2)
        Lt_h{e}= uicontrol('Style','Radio','String',control.load_types{e},...
            'units','normalized','pos',[((itemspace*e)- 0.25), .3,  1, 0.5],...
            'parent',Load_type_h,'HandleVisibility','on'); %#ok<AGROW>
    end
    uicontrol('Style', 'text', 'String','Load Options','fontsize',10,'units','normalized',...
        'position', [0.37, 0.5, 0.243, 0.395],'parent',tab2)
```

Figure 48: Initialising the application, setting appearance and creating interactive features (1 of 3)

```
%Create a button to delimit data based on statistical measures.
Auto_exlude_h =   uicontrol('Parent',tab2,'Style' , 'pushbutton',...
    'String','Auto Screen' ,...
    'Units' ,'Normalized',...
    'Position', [0.020 0.5 0.243 0.395],...
    'BackgroundColor',[.1 .44 .89],...
    'ForegroundColor', [1 1 1],...
    'FontWeight','bold',...
    'Tag','PlotData',...
    'Callback',{@GUI_Auto_screen,Status_h,ExEv,control}); %#ok<NASGU>

% Create list box to select sensor
datalist_h = uicontrol('Parent',tab1,'Style', 'list',...
    'String', {control.datalist},...
    'max',1,'min',1,...
    'Units' ,'Normalized',...
    'Position', [0.345,0.08,0.30,0.79]);


%Set up controls for plotting data.
plot_control_h = uipanel('visible','on','units','normalized','Position',[0.23,0.189,0.537,0.079]);
no_curves_h = uicontrol('Style', 'edit', 'units','normalized',...
    'position', [0.408,0.010,0.2,0.45],'parent',plot_control_h);% Number of curves to plot at a time.
curves_text_h = uicontrol('Style', 'text', 'String','No. of Curves:','fontsize',10,'units','normalized',...
    'position', [0.10,0.469,0.80,0.55],'parent',plot_control_h);
uistack(curves_text_h,'top');

%Create load data button
load_data_h =   uicontrol('Parent',tab1,'Style' , 'pushbutton',...
    'String','Load data' ,...
    'Units' ,'Normalized',...
    'Position', [0.020 0.043 0.243 0.395],...
    'BackgroundColor',[.1 .44 .89],...
    'ForegroundColor', [1 1 1],...
    'FontWeight','bold',...
    'Tag','PlotData',...
    'Callback',{@GUI_load_data,Status_h,datalist_h,Lt_h,curves_text_h,no_curves_h,control}); %#ok<NASGU>

%Create plot forwards button
plot_next_h =   uicontrol('Parent',plot_control_h,'Style' , 'pushbutton',...
    'String','Next >>>' ,...
    'Units' ,'Normalized',...
    'Position', [0.729,0.07,0.249,0.807],...
    'BackgroundColor',[.1 .44 .89],...
    'ForegroundColor', [1 1 1],...
    'FontWeight','bold',...
    'Tag','PlotData',...
    'Callback',{@GUI_next_data,Status_h,curves_text_h,no_curves_h,control}); %#ok<NASGU>
```

Figure 49: Initialising the application, setting appearance and creating interactive features (2 of 3)

```
%Create plot backwards button
plot_prev_h =   uicontrol('Parent',plot_control_h,'Style' , 'pushbutton',...
    'String','<<< Previous' ,...
    'Units' ,'Normalized',...
    'Position', [ 0.018,0.07,0.249,0.807],...
    'BackgroundColor',[.1 .44 .89],...
    'ForegroundColor', [1 1 1],...
    'FontWeight','bold',...
    'Tag','PlotData',...
    'Callback',{@GUI_prev_data,Status_h,curves_text_h,no_curves_h,control}); %#ok<NASGU>

%Create plot data button
plot_data_h =   uicontrol('Parent',tab1,'Style' , 'pushbutton',...
    'String','Plot data' ,...
    'Units' ,'Normalized',...
    'Position', [0.020 0.5 0.243 0.395],...
    'BackgroundColor',[.1 .44 .89],...
    'ForegroundColor', [1 1 1],...
    'FontWeight','bold',...
    'Tag','PlotData',...
    'Callback',{@GUI_plotdata,Status_h,datalist_h,curves_text_h,no_curves_h,control}); %#ok<NASGU>

%Create exclude data button
exclude_data_h =   uicontrol('Parent',tab1,'Style' , 'pushbutton',...
    'String','Exclude data' ,...
    'Units' ,'Normalized',...
    'Position', [0.737 0.5 0.243 0.395],...
    'BackgroundColor',[1, 0, 0],...
    'ForegroundColor', [1 1 1],...
    'FontWeight','bold',...
    'Callback',{@GUI_excludedata,Status_h,datalist_h,control}); %#ok<NASGU>

%Create screen data button
screen_data_h =   uicontrol('Parent',tab1,'Style' , 'pushbutton',...
    'String','Screen data' ,...
    'Units' ,'Normalized',...
    'Position', [0.737 0.043 0.243 0.395],...
    'BackgroundColor',  [0.9290, 0.6940, 0.1250],...
    'ForegroundColor', [1 1 1],...
    'FontWeight','bold',...
    'Callback',{@GUI_screendata,Status_h,datalist_h,control}); %#ok<NASGU>
```

Figure 50: Initialising the application, setting appearance and creating interactive features (3 of 3)

3.3.2 Method: LoadData

This method launches a dialogue box to allow the user to locate the data, which is then loaded into the application. It allows three forms of loading, as stored (default), randomised and ordered by the base identifier. This functionality was useful to visualise the data in different ways. After reading in the data, it writes the file categories (as determined by the MATLAB data structure) to a selection box (Figure 51 and Figure 52).

```
function GUI_load_data(~,~,Status_h,datalist_h,Lt_h,curves_text_h,no_curves_h,control)
    %GUI_LOAD_DATA: This function opens a dialouge box to allow
    %the user select the MATLAB structure where the data is
    %stored.

    [struct_name, control.pathname, ~] = uigetfile('Select Data Structure');
    set(Status_h,'String','Loading data')
    drawnow

    % read in structure and rename appropiately
    try
        control.my_struct = load([control.pathname,struct_name]);
        struct_name = fieldnames([control.my_struct]);
        control.my_struct = control.my_struct.(struct_name{1});
    catch
        set(Status_h,'String','Error loading data')
        drawnow
        return
    end

    %read the fieldnames of the structure and store in property
    control.datalist = fieldnamesr(control.my_struct);

    if Lt_h{2}.Value == 1
        %randmise columns so more varition in sequenctially plotted
        %curves.
        for i = 1:size(control.datalist,1)
            rng(42)
            rand_idx = randperm(size(eval(['control.my_struct.',control.datalist{i}]),2));
            currrent_data =  eval(['control.my_struct.',control.datalist{i}]);
            Structstrings  = strsplit(control.datalist{i},'.');
            control.my_struct = Dynamicstruct(control.my_struct,Structstrings,currrent_data(:,rand_idx));
        end
    elseif Lt_h{3}.Value == 1
        %order by participant.
        if ~istable(eval(['control.my_struct.',control.datalist{1}]))
            errordlg('Participant load only works with tables for now','Error Loading');
        else
            for i = 1:size(control.datalist,1)
                currrent_data =  eval(['control.my_struct.',control.datalist{i}]);
                [~,sort_idx] = sort(currrent_data.Properties.VariableNames);
                Structstrings  = strsplit(control.datalist{i},'.');
                control.my_struct = Dynamicstruct(control.my_struct,Structstrings,currrent_data(:,sort_idx));
                set(curves_text_h, 'String','Screening Participants')
                set(no_curves_h,'visible','off');
            end
        end
        set(curves_text_h, 'String','Screening Participants')
        set(no_curves_h,'visible','off');
        control.c_load_type = 'Participant load';
    end
```

Figure 51: Method to load data (1 of 2)

```
    % list structure field names in gui box.
    set(datalist_h,'String',control.datalist)
    drawnow
    set(Status_h,'String','Finished importing data')

    %If there is not a screening index in the file location,
    %create it.
    try
        if ~exist([control.pathname,'ScreenIndex.xlsx'],'file')
            T = cell2table(cell(1,2),'VariableNames',{'Observation','Condition'});
            writetable(T,[control.pathname,'ScreenIndex.xlsx'],'Sheet','ScreenData')
            writetable(T,[control.pathname,'ScreenIndex.xlsx'],'Sheet','ExcludeData')
            control.ExcludeIndex = readtable([control.pathname,'ScreenIndex.xlsx'],'Sheet','ExcludeData');
            control.ScreenIndex = readtable([control.pathname,'ScreenIndex.xlsx'],'Sheet','ScreenData');

        else
            control.ExcludeIndex = readtable([control.pathname,'ScreenIndex.xlsx'],'Sheet','ExcludeData');
            control.ScreenIndex = readtable([control.pathname,'ScreenIndex.xlsx'],'Sheet','ScreenData');
        end
    catch
        set(Status_h,'String','Error writing files')
        drawnow
    end
end
```

Figure 52: Method to load data (2 of 2)

3.3.3 Method: PlotData

This method plots user selected data stored in long or wide format (Figure 53). After consulting the screen index file to determine if any of trials should be excluded, this method plots user selected data. In order to provide some context to the data, a shaded region is also plotted to represent the mean ± 2 * standard deviations of the whole dataset. If the user has loaded the data by base identifier (participant), the number of plots displayed will equal the all the trials by the first participant. If the user has loaded the data as default or by randomised load, the number of trials displayed will be determined by a user defined number entered in an interactive text box (or defaults to 20) (Figure 54).

```
function GUI_plotdata(~,~,Status_h,datalist_h,curves_text_h,no_curves_h,control)
    %GUI_PLOTDATA: Function to plot data from the first
    %observation.#
    %reset properties
    control.selected_curve = []; control.ob_ident = []; control.num_curve = [];
    control.num_curve = 0;

    %set up enviroment for plotting.
    set(Status_h,'String','Plotting data')
    drawnow
    cfield = datalist_h.String(datalist_h.Value);
    cdata = eval(['control.my_struct.',cfield{1}]) ;
    Plot_h = subplot(3,1,1:2);
    Plot_h.Position = [0.104,0.401,0.77,0.515];

    %check to see if data is in wide or long format and format
    %approporiatly.
    if istable(cdata)
        cdata = [cdata.Properties.VariableNames; table2cell(cdata)];
    end
    if iscell(cdata)
        if sum(cellfun(@ischar,cdata(:,1)))/size(cdata(:,1),1) == 1 %Long data
            %Gather oberservation identifiers:
            subdata = cdata(:,(cellfun(@ischar,cdata(1,:))));
            control.ob_ident = cell(size(subdata,1),1);
            control.c_data = cell2mat(cdata(:,~cellfun(@ischar,cdata(1,:))));
            for i = 1:size(subdata,2)
                if i > 1
                    control.ob_ident = strcat(control.ob_ident,'_',subdata(:,i));
                else
                    control.ob_ident = strcat(control.ob_ident,subdata(:,i));
                end
            end
        else % wide data
            subdata = cdata((cellfun(@ischar,cdata(:,1))),:);
            control.ob_ident = cell(size(subdata,2),1);
            control.c_data = cell2mat(cdata(~cellfun(@ischar,cdata(:,1)),:));
            subdata = subdata';
            for i = 1:size(subdata,2)
                if i > 1
                    control.ob_ident = strcat(control.ob_ident,'_',subdata(:,i));
                else
                    control.ob_ident = strcat(control.ob_ident,subdata(:,i));
                end
            end
        end
    else
        set(Status_h,'String','This app can currently only import cell or table data')
    end
```

Figure 53: Method to plot data (1 of 2)

```
%exclude all of a participants data based on 'all_data' keyword.
if ~isempty(control.ExcludeIndex.Observation)
    exclude_idx =   control.ExcludeIndex.Observation(contains(control.ExcludeIndex.Observation,'all_data'));
    exlude_prefix = cellfun(@(x){x(1:strfind(x,'all_data')-1)}, exclude_idx);
    control.c_data   = control.c_data(:,~contains(control.ob_ident,exlude_prefix));
    control.ob_ident  = control.ob_ident(~contains(control.ob_ident,exlude_prefix));
end

%exclude data in exclude and screen index.
if ~isempty(control.ScreenIndex.Observation)
    control.c_data   = control.c_data(:,~contains(control.ob_ident,control.ScreenIndex.Observation));
    control.ob_ident  = control.ob_ident(~contains(control.ob_ident,control.ScreenIndex.Observation));
end
if ~isempty(control.ExcludeIndex.Observation)
    control.c_data   = control.c_data(:,~contains(control.ob_ident,control.ExcludeIndex.Observation));
    control.ob_ident  = control.ob_ident(~contains(control.ob_ident,control.ExcludeIndex.Observation));
end

% store error bands
control.bands = [(mean(control.c_data,2) + (2* std(control.c_data,1,2)));...
    flipud((mean(control.c_data,2) - (2* std(control.c_data,1,2))))];

% plot error bands
xframes = [(0:size(control.c_data,1)-1),fliplr(0:size(control.c_data,1)-1)]';
hPatch = patch(xframes, control.bands,...
    'k','facealpha',0.2,'edgecolor','none');
set(hPatch,'HitTest','off');
hold on

%Plot by participant
if strcmp(control.c_load_type,'Participant load')
    participants =  cellfun(@(x){x(1:max(strfind(x,'_')))}, control.ob_ident);
    control.c_particpant = participants{1};
    end_idx = find(contains(participants,control.c_particpant), 1, 'last' );
    control.current_range = [1,end_idx];
    set(curves_text_h, 'String',['Screening Participant ',control.c_particpant])
    cdata = control.c_data(:,1:control.current_range(2));
    participant_bands = [(mean(cdata,2) + (2* std(cdata,1,2))); flipud((mean(cdata,2) - (2* std(cdata,1,2))))];

    % plot error bands
    xframes = [(0:size(control.c_data,1)-1),fliplr(0:size(control.c_data,1)-1)]';
    control.hPatch_p = patch(xframes, participant_bands,...
        'b','facealpha',0.2,'edgecolor','none');
    set(control.hPatch_p,'HitTest','off');
    hold on
else
    %Plot by curves - Set current plot range
    if isempty(no_curves_h.String)
        no_curves_h.String = '20';
        control.current_range = [1,str2double(no_curves_h.String)];
    else
        control.current_range = [1,str2double(no_curves_h.String)];

    end
    %control for less data than range
    if size(control.c_data,2) <= control.current_range(2)
        control.current_range(2) = size(control.c_data,2);
        no_curves_h.String = num2str(size(control.c_data,2));
        set(Status_h,'String','Plotting all data')
    end

end
%plot data with button down callback function
control.linelist = plot3(0:100,control.c_data(:,1:control.current_range(2)),ones(1,101),...
    'ButtonDownFcn',{@StoreCurve,Status_h,control},'LineWidth',1.5);
```

Figure 54: Method to plot data (2 of 2)

The data is plotted with a button-down call back function, which allows the user to select any of the curves after they are plotted by clicking on a curve and excluding it from the current session (Figure 55).

```
function StoreCurve(LineH, ~,Status_h, control)
    %STORECURVE: This function stores the handle of selected curves

    if sum(control.selected_curve == LineH) > 0
        set(Status_h,'String',[num2str(control.num_curve),' Lines currently selected - (Line already selected)'])
    else
        control.num_curve = control.num_curve + 1;
        set(Status_h,'String',[num2str(control.num_curve),' Lines currently selected'])
        control.selected_curve = [control.selected_curve;LineH];
        set(LineH,'LineWidth', 2.5)
    end
end
```

Figure 55: Callback function to store clicked curves

3.3.4 Method: Next data and Previous data

These methods allow the user to cycle forward or backwards through the data. Similar to the plot data method (see section 3.3.3), the number of plots displayed depend on the method of loading. As both the next data and previous data methods are very similar, only the next data method is shown here (Figure 56 and Figure 57).

```
function GUI_next_data(~,~,Status_h,curves_text_h,no_curves_h,control)
    %GUI_NEXT_DATA: button donwn function to plot next n curves.

     set(Status_h,'String','Plotting data')
     drawnow

    %rest properties
    control.selected_curve = []; control.num_curve = 0;

    %delete all lines on figure
    hLine = findall(gcf,'Type','line') ;
    delete(hLine)
    delete(control.hPatch_p)

    %exclude data in exclude and screen index.
    if ~isempty(control.ScreenIndex.Observation)
        control.c_data    = control.c_data(:,~contains(control.ob_ident,control.ScreenIndex.Observation));
        control.ob_ident  = control.ob_ident(~contains(control.ob_ident,control.ScreenIndex.Observation));
    end
    if ~isempty(control.ExcludeIndex.Observation)
        control.c_data    = control.c_data(:,~contains(control.ob_ident,control.ExcludeIndex.Observation));
        control.ob_ident  = control.ob_ident(~contains(control.ob_ident,control.ExcludeIndex.Observation));
    end

    %plot by participant
    if strcmp(control.c_load_type,'Participant load')
        participants =  cellfun(@(x){x(1:max(strfind(x,'_')))}, control.ob_ident);
        start_idx = find(contains(participants,control.c_particpant), 1, 'last' ) + 1;
        try
            control.c_particpant = participants(start_idx);
            end_idx =  find(contains(participants,control.c_particpant), 1, 'last' );
            control.current_range = [start_idx,end_idx];
            set(curves_text_h, 'String',['Screening Participant ',control.c_particpant])

            cdata = control.c_data(:,control.current_range(1):control.current_range(2));
            participant_bands = [(mean(cdata,2) + (2* std(cdata,1,2)));...
                flipud((mean(cdata,2) - (2* std(cdata,1,2))))];
```

Figure 56: Method to cycle through data and plot curves (1 or 2)

```
        cdata = control.c_data(:,control.current_range(1):control.current_range(2));
        participant_bands = [(mean(cdata,2) + (2* std(cdata,1,2)));...
            flipud((mean(cdata,2) - (2* std(cdata,1,2))))];

        % plot error bands
        xframes = [(0:size(control.c_data,1)-1),fliplr(0:size(control.c_data,1)-1)]';
        control.hPatch_p = patch(xframes, participant_bands,...
            'b','facealpha',0.2,'edgecolor','none');
        set(control.hPatch_p,'HitTest','off');
        hold on
    catch
        set(Status_h,'String','No more data to plot')
    end
else

    %determine how many lines to plot
    if isempty(no_curves_h.String)
        no_curves_h.String = '20';
        inc = 20;
    else
        inc = str2double(no_curves_h.String);
    end

    if  size(control.c_data,2) >  control.current_range(2) + inc
        control.current_range = [control.current_range(2) + 1, control.current_range(2) + inc];
    elseif (size(control.c_data,2) - control.current_range(2)) == 0
        set(Status_h,'String','No more data to plot')
    else
        control.current_range = [size(control.c_data,2) - (inc -1) ,size(control.c_data,2)];
    end
end
   set(Status_h,'String','Waiting..')
    %plot curves.
    control.linelist = plot3(0:size(control.c_data,1)-1,...
        control.c_data(:,control.current_range(1):control.current_range(2)),ones(1,101),...
        'ButtonDownFcn',{@StoreCurve,Status_h,control},'LineWidth',1.5);
end
```

Figure 57: Method to cycle through data and plot curves (2 or 2)

3.3.5 Method: Exclude data and Screen data

These methods are called by pressing the exclude data or screen data buttons. These methods write the identifiers associated with the selected curves to a file. As both methods are very similar only the exclude data method will be presented here (Figure 58). An example of the output is also presented (Figure 59).

```
function GUI_excludedata(~,~,Status_h,datalist_h, control)
    %GUI_EXCLUDEDATA: This button function writes currently
    %selected curve indentifiers to a screening list.

    set(Status_h,'String','Excluding data')
    cfield = datalist_h.String(datalist_h.Value);

    %find data belonging to currently selected lines.
    for i = 1:size(control.selected_curve,1)
        idx = control.selected_curve(i) == control.linelist;
        c_obs = control.ob_ident(control.current_range(1):control.current_range(2));
        control.ExcludeIndex =  [control.ExcludeIndex ;[c_obs(idx),cfield(1)]];
    end

    try
        %Write ExcludeIndex to file.
        writetable(control.ExcludeIndex,[control.pathname,'ScreenIndex.xlsx'],'Sheet','ExcludeData')
    catch
        set(Status_h,'String','Excluded data not stored')
        errordlg('Make sure target excel file is closed and try again','Error Exporting');
        return
    end

    %delete currently select line objects.
    for i = 1:size(control.selected_curve,1)
        delete(control.selected_curve(i))
    end

    %reset curves
    control.num_curve = 0; control.selected_curve = [];

    set(Status_h,'String','Ready..')

end
```

Figure 58: Method to write selected curve identifiers to file

Observation	Condition
P_4315_L_stance24	Norm.Baseline_Run_9kmhr_Vue.stance.KneeVelocity_rot_Auto_Entropy
P_4315_L_stance35	Norm.Baseline_Run_9kmhr_Vue.stance.KneeVelocity_rot_Auto_Entropy
P_4315_L_stance75	Norm.Baseline_Run_9kmhr_Vue.stance.KneeVelocity_rot_Auto_Entropy
P_4056_R_stance24	Norm.Baseline_Run_9kmhr_Vue.stance.KneeAcceleration_fle_Auto_Amplitude
P_4057_R_stance33	Norm.Baseline_Run_9kmhr_Vue.stance.KneeAcceleration_fle_Auto_Amplitude
P_4057_L_stance26	Norm.Baseline_Run_9kmhr_Vue.stance.KneeAcceleration_fle_Auto_Amplitude

Figure 59: Example outputed file after excluding curves

3.3.6 Method: Auto Screen

This final method provides the option to automatically delimit the data based on statistical tests, to identify extreme outliers in terms of amplitude and entropy (Figure 60 and Figure 61).

```
function GUI_Auto_screen(~,~,Status_h,ExEv,control)
    %This function implements some statistical tests to automatically delimit the data

    %read the fieldnames of the structure and store in property
    control.datalist = fieldnamesr(control.my_struct);
    cdata =  eval(['control.my_struct.',control.datalist{1}]);

        if istable(cdata)
            control.ob_ident = cdata.Properties.VariableNames;
            control.c_data =  table2array(cdata);
        else
            errordlg('Auto Screen only works with tables for now','Error Screening');
        end

        %exclude all of a participants data based on 'all_data' keyword.
        if ~isempty(control.ExcludeIndex.Observation)
          exclude_idx =   control.ExcludeIndex.Observation(contains(control.ExcludeIndex.Observation,'all_data'));
          exlude_prefix = cellfun(@(x){x(1:strfind(x,'all_data')-1)}, exclude_idx);
          control.c_data   = control.c_data(:,~contains(control.ob_ident,exlude_prefix));
          control.ob_ident  = control.ob_ident(~contains(control.ob_ident,exlude_prefix));
        end

        %exclude data in exclude and screen index.
        if ~isempty(control.ScreenIndex.Observation)
            control.c_data   = control.c_data(:,~contains(control.ob_ident,control.ScreenIndex.Observation));
            control.ob_ident  = control.ob_ident(~contains(control.ob_ident,control.ScreenIndex.Observation));
        end
        if ~isempty(control.ExcludeIndex.Observation)
            control.c_data   = control.c_data(:,~contains(control.ob_ident,control.ExcludeIndex.Observation));
            control.ob_ident  = control.ob_ident(~contains(control.ob_ident,control.ExcludeIndex.Observation));
        end

        for i = 1:size(control.datalist,1)
            %update info provided to user.
            set(Status_h,'String',['Auto Screening ', num2str(i),' out of ',num2str(size(control.datalist,1))])
            drawnow
            cdata =  eval(['control.my_struct.',control.datalist{i}]);

            % Auto screen based on Amplitude
            if ExEv{1}.Value == 1
                idx1 =  control.c_data' > (mean(control.c_data') + (std(control.c_data') * 4.0)); %#ok<UDIM>
                idx2 =   control.c_data' <( mean(control.c_data') - (std(control.c_data') * 4.0));%#ok<UDIM>
                idx = sum((idx1 == 1|idx2 == 1),2)> 5 ;
                outliers = control.ob_ident(idx);
                control.ScreenIndex =  [control.ScreenIndex ;...
                    [outliers',repmat({[control.datalist{i},'_Auto_Amplitude']},size(outliers'))]];
            end
```

Figure 60: Method to auto screen data (1 of 2)

```
    % Auto screen based on Entropy
    if ExEv{2}.Value == 1
        data_std = std(control.c_data);
        ent_data = zeros(size(data_std));
        for x = 1:size(data_std,2)
            [se,~,~] = sampenc(control.c_data(:,x),2,(0.2*data_std(x)));
            ent_data(1,x) =  se(2,1);
        end
        idx1 =  ent_data > (mean(ent_data,2,'omitnan') + (std(ent_data,0,2,'omitnan') * 4.5));
        idx2 =      ent_data < (mean(ent_data,2,'omitnan') - (std(ent_data,0,2,'omitnan') * 4.5));
        idx = (idx1 == 1|idx2 == 1)~=0 ;
        outliers = control.ob_ident(idx);
        control.ScreenIndex =  [control.ScreenIndex ;...
            [outliers',repmat({[control.datalist{i},'_Auto_Entropy']},size(outliers'))]];
    end
end
if ExEv{1}.Value == 0 && ExEv{2}.Value == 0
    set(Status_h,'String','Error... no metric selected to auto screen')
end

% Save data to file
try
    writetable( control.ScreenIndex,[control.pathname,'ScreenIndex.xlsx'],'Sheet','ScreenData')
catch
    set(Status_h,'String','Data to screen not stored')
    errordlg('Make sure target excel file is closed and try again','Error Exporting');
    return
end
set(Status_h,'String','Finished Auto Screening')
```

Figure 61: Method to auto screen data (2 of 2)

3.4 General Preprocessing

After extracting the data from the biomechanical files and normalising the stance phase data to 101 data points in MATLAB, the data was ready for the general preprocessing phase in python. Firstly, the required packages were loaded including the custom modules (LandMarkReg and ACP). The data was read in, along with the injury status of the participants. The data was then delimited to those participants who completed the prospective arm of the study (Figure 62).

```python
"""
This script was written for the MSc project entitled 'The identification of foot-strike patterns
using unsupervised learning and their association with injury'. This script represents the general
preprocessing of the data.

An overview of the preprocessing steps:

    The biomechanical data is aligned using dynamic time warping
    Features are generated using ACP and TSFresh.
    The data is scaled to zero mean, unit variance.
    The data is screened for outliers using LocalOutlierFactor and IsolatedForests.
    Missing data is imputed using MICE and a baysian ridge regression.
    Features with with near zero variance are removed.
    Highly correlated features are removed.

#Written by Shane Gore Cotact: Shane.Gore2@gmail.com
"""
##############################################################################
#                            Import Packages and data                        #
##############################################################################
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from scipy import interpolate
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import copy
from sklearn.feature_selection import VarianceThreshold
from tsfresh import extract_features
from tsfresh.feature_extraction import MinimalFCParameters
from sklearn.impute import SimpleImputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest

os.chdir('D:\MSc Thesis\MSc Files')

#custom modules
from LandMarkReg import LandMarkReg #custom module
from ACP import ACP #custom module

#Read data in
data_conditions = pd.read_csv('Vuedata_index_stance_28_06.csv')
data = pd.read_csv('Vuedata_stance_28_06.csv', header=None)

#View data
data.head()
data_conditions.head()

#Read in injury status.
injury_status = pd.read_excel('Injury Status_06_2020.xlsx')
injury_status.columns = ['Participant_ID','Injured_Prospectively']
injury_status['Injured_Prospectively'][injury_status['Injured_Prospectively']
.str.contains("y",na=False,case=False)]=1
injury_status['Injured_Prospectively'][injury_status['Injured_Prospectively']
.str.contains("n",na=False,case=False)]=0

#Delimit data if no injury status (due to study withdrawl, non RRI injury etc)
participants = data_conditions.conditions_t5.str[0:6]
len(set(participants))
idx  =  np.in1d(participants,injury_status.Participant_ID)
data_conditions = data_conditions.iloc[idx,:]
participants =  participants.iloc[idx]
data  = data.iloc[idx,:]
idx  =  np.in1d(injury_status.Participant_ID,participants)
injury_status = injury_status.iloc[idx,:]
```

Figure 62: Loading packages and data

The biomechanical time series data was then aligned using landmark registration Moudy et al. (2018) using the LandMarkReg class (Figure 63). Details of the LandMarkReg are presented in section 3.4.1.

```
##############################################################################
#                            General Preprocessing                           #
##############################################################################

#Landmark registration using DTW.
#define global landmark to align all other data
global_data = (data.loc[data_conditions.conditions_t4 == 'KneeAngles_fle',:]).transpose()
global_data_mean = np.array(global_data.mean(axis = 1))
global_landmark = np.argmax(global_data_mean)
x_array = np.arange(len(global_data_mean)).reshape(len(global_data_mean),1) #array 0:100


#create a list of the features to be registered.
registered_data_conditions = pd.DataFrame(columns=data_conditions.columns)
variables = list(data_conditions.conditions_t4.unique()) #list of different biomech variables

#carry out landmark registration
registered_data = np.zeros_like(data)
counter = 0
for variable in variables:
    print(variable)
    current_data = data.loc[data_conditions.conditions_t4 == variable,:].transpose()
    for i in tqdm(range(np.size(current_data,axis = 1))):
        current_landmark = np.argmax(np.array(global_data.iloc[0:80,i]))
        Landmark = LandMarkReg(101,current_landmark,global_landmark)
        reg = Landmark.DynamicTimeWarp()
        tck = interpolate.splrep(x_array, current_data.iloc[:,i])
        registered_data[counter,:] = (interpolate.splev(reg, tck)).transpose()
        counter += 1
    current_condit = data_conditions.loc[data_conditions.conditions_t4 == variable,:]
    registered_data_conditions = registered_data_conditions.append(current_condit)
participants_delim =  pd.Series(participants
                                [data_conditions.conditions_t4 == variable],name="Participant_ID")

#Save files incase of error.
np.savetxt("registered_data.csv", registered_data, delimiter=",")
registered_data_conditions.to_csv('registered_data_conditions.csv', index=False)
```

Figure 63: Implementation of Landmark Registration

Using the time aligned data, 'Analysis of Characterising Phases' (ACP) is conducted 100 times on random 70% subsamples (Figure 64). Only robust phases were then retained defined as being identified more than 80% of the time (Richter et al.; 2019). For details on the ACP class, please see section 3.4.2.

```
#replace zero variance time series with nans
registered_data[np.std(registered_data[:,:],axis = 1) == 0,:]  = np.nan

#Calculate Robust ACP features
#Excessuvly preallocate.
Features = np.zeros([registered_data
                    [registered_data_conditions.conditions_t4 == variable,:].shape[0],1000],np.float64)
Feature_names = pd.DataFrame()
counter = 0
for variable in variables:
    print(variable)
    reg_data = copy.copy(registered_data[registered_data_conditions.conditions_t4 == variable ,:])
    ACP_idx   = np.zeros([1,101],int)
    phase_idx = np.zeros([100,101],int)
    for x in range(0,100):
        np.random.shuffle(reg_data)
        c_reg_data = reg_data[0:round(len(reg_data)*0.70),:]
        ACP_phases = ACP(c_reg_data,90)
        phase_start,phase_end = ACP_phases.identify_phases()
        for i in range(len(phase_start)):
            phase_idx[x,phase_start[i]:phase_end[i] +1] = 1
    ACP_idx[0,np.sum(phase_idx,0) > 80] = 1
    cp = np.diff(ACP_idx)
    phase_end = np.where(cp == -1)[1] + 1
    phase_start = np.where(cp == 1)[1] + 1
    if phase_start[0] > phase_end[0]:
        phase_start = np.hstack(([0],phase_start))
    if phase_start[-1] > phase_end[-1]:
        phase_end =  np.hstack((phase_end,[101]))
    reg_data = registered_data[registered_data_conditions.conditions_t4 == variable ,:]
    for i in range(len(phase_start)):
        Features[:,counter] = np.nanmean(reg_data[:,phase_start[i]:phase_end[i]],axis = 1)
        Feature_names = Feature_names.append([variable + '_' +str(phase_start[i]) + '_' + str(phase_end[i])])
        counter += 1

ACP_Features  = Features[:,np.sum(Features,axis = 0) != 0] #remove empty rows  from preallocation
ACP_Features = pd.DataFrame(ACP_Features,columns = Feature_names[0].tolist())
ACP_Features = ACP_Features.set_index(registered_data_conditions['conditions_t5']
[registered_data_conditions['conditions_t4'] == variable])
ACP_Features.to_csv('Features.csv')
```

Figure 64: Implementation of Analysis of Characterising Phases

Additional time series metrics are then calculated using the TSFresh python package (Christ et al.; 2018) (Figure 65).

```
#Calculate TS FRESH on time series
registered_data_conditions = registered_data_conditions.reset_index()
ts_data = pd.concat([registered_data_conditions[['conditions_t4','conditions_t5']],
                     pd.DataFrame(registered_data)], axis =1)
ts_data = pd.melt(ts_data, id_vars =['conditions_t5','conditions_t4'])
ts_data = ts_data.pivot_table(index=['conditions_t5', 'variable'],
                              columns='conditions_t4', values='value').reset_index()
ts_data.to_csv('ts_data_0507.csv') #Save down in case of error. Pivoting data resource intensive.
ts_data = pd.read_csv('ts_data_0507.csv')
ts_data = ts_data.iloc[:,1:]

#Temp Input with median on missing data so TsFresh can run.
imp = SimpleImputer(missing_values=np.nan, strategy='median')
imp.fit(ts_data.iloc[:,2:])
TS_temp = imp.transform(ts_data.iloc[:,2:])

ts_data_c = copy.copy(ts_data)
ts_data_c.iloc[:,2:] = TS_temp
TSF_features = extract_features(ts_data_c, column_id="conditions_t5", column_sort="variable",
                                default_fc_parameters= MinimalFCParameters(), n_jobs = 0)
TSF_features.to_csv('TSF_features_0507.csv')

#Merge all features into single file.
All_Features = pd.merge(ACP_Features,TSF_features,left_index=True, right_index=True)
All_Features.to_csv('All_Features_0507.csv')
```

Figure 65: Engineering additional feature with TSFresh

Data was scaled to zero mean and unit variance. Outliers in the generated features were detected using isolated forests and local outlier factor. Missing data was then imputed using multivariate imputation by chained equations (MICE) and a Bayesian ridge regression approach based on the twenty nearest features (Figure 66).

```
#Scale data to zero mean, unit variance
Features_scaled = StandardScaler().fit_transform(All_Features)
Feature_names = list(All_Features)

#Temp imput data with median so outlier detection can be used.
imp = SimpleImputer(missing_values=np.nan, strategy='median')
imp.fit(Features_scaled)
Features_scaled_temp = imp.transform(Features_scaled)

#Screen outliers
clf = LocalOutlierFactor(n_neighbors=20, contamination='auto', n_jobs = -1)
y_pred = clf.fit_predict(Features_scaled_temp[:,:-1])
clf = IsolationForest(max_samples = 'auto', contamination='auto', n_jobs = -1)
clf = clf.fit(Features_scaled_temp[:,:-1])
y_pred2 = clf.predict(Features_scaled_temp[:,:-1])
Features_scaled[(y_pred == -1)|(y_pred2 == -1),:] = np.nan

#Impute misssing values using MICE
MICE = IterativeImputer(random_state=0, estimator = BayesianRidge(verbose = 2), n_nearest_features = 20)
MICE.fit(Features_scaled)
Features_scaled = MICE.transform(Features_scaled)
```

Figure 66: Scaling the data, identifying outliers and imputation with MICE

In the final stage of the general preprocessing, near zero variance and highly correlated features are removed. Injury status is added to the feature matrix and the data is visualised as its first two PCs (Figure 67).

```
#Remove features with near zero variance (e.g column lenght)
from itertools import compress
selector = VarianceThreshold(0.01)
selector.fit(Features_scaled)
Features_scaled = Features_scaled[:,selector.get_support()]
Feature_names = list(compress(Feature_names, selector.get_support()))

#Remove highly correlated features (r > 0.95)
Features_scaled = pd.DataFrame(Features_scaled,columns = Feature_names)
corr_matrix = Features_scaled.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]
Features_scaled = Features_scaled.drop(Features_scaled[to_drop], axis=1)
[Feature_names.remove(drop) for drop in to_drop]

#Add in injury status to Feature space
paticipant_class = pd.merge(participants_delim,injury_status)
Features_scaled = pd.concat([Features_scaled,paticipant_class[['Participant_ID','Injured_Prospectively']]
.reset_index(drop=True)], axis=1)
Features_scaled = Features_scaled.set_index(registered_data_conditions['conditions_t5']
[registered_data_conditions['conditions_t4'] == variable])
Feature_names = list(Features_scaled)
Features_scaled.to_csv('Features_scaled_preprocessed.csv', index=False)

#Visualises features as first 2 PCs
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(Features_scaled.iloc[:,:-2])
fig, ax = plt.subplots(figsize=(10, 5))
ax.scatter(principalComponents[:,0], principalComponents[:,1],alpha=0.70,c= Features_scaled.Injured_Prospectively)

#Percentage of uninjuried.
np.sum(paticipant_class.Injured_Prospectively)/ len(paticipant_class.Injured_Prospectively)
```

Figure 67: Removing near zero and high correlated feature before adding injury status

3.4.1 LandMark Registration

In order to remove unwanted temporal variations from the biomechanical waveforms, a landmark registration algorithm as previously described (Moudy et al.; 2018), was employed using a custom writen python class (Figure 68) and a sub method to dynamically warp the timing of the signals (Figure 69 and Figure 70). In comparison to the algorithm proposed by Moudy et al. (2018), this current project used an akima spline rather than a cubic spline and implemented a binary search approach to speed up convergence. This latter point was important given the size of the dataset.

```python
class LandMarkReg(object):
    """This is a class to warp the velocity of signals so that the user
    identified landmarks align. This class is based on the MATLAB class
    written by Dr Chris Richter and described in the paper by Moudy et al 2018.

    Modifications:
    - Uses divide and conquer, binary search algorithm to speed up convergence.
    - Uses Akima spline rather than cubic spline to reduce overshooting between
      landmarks.
    - Smaller error tolerance resutls in better alignment.

    Attributes:
        nrows: An int representing the number of rows in the signal.
        current_pos: A int representing the position of the ladmark in the
        current signal.
        landmark_pos: An int representing the position of the registration
        landmark.


    Moudy, S., Richter, C., & Strike, S. (2018). Landmark registering waveform
    data improves the ability to predict performance measures.
    ournal of biomechanics, 78, 109-117.

    Writen by Shane Gore 2020. Contact Shane.Gore2@Gmail.com"""

    def __init__(self, nrows =101, current_pos ='', landmark_pos=''):

        self.nrows = nrows
        self.current_pos  = current_pos
        self.landmark_pos = landmark_pos

    def DynamicTimeWarp(self):

        #Preallocate array for speed and set up array for warping.
        speedMAT = np.empty((self.nrows,1))
        speedMAT.fill(np.nan)
        speedMAT[np.array([0, self.landmark_pos, self.nrows -1])] = 1

        #landmarkss
        mpos = [0,self.landmark_pos, self.nrows]
        cpos = [0,self.current_pos, self.nrows]

        #Loop through the number of landmarks and warp velocity as required.
        warp = np.empty([0,1],np.float64)
        for n in range(len(mpos)-1):
            c_len = cpos[n+1] - cpos[n]

            if n == 0:
                cutting = False
            else:
                cutting = True

            warp = np.append(warp,self.warpfnc(speedMAT[mpos[n]:mpos[n+1]+1],c_len,cutting))

        #convert to frames, zero index
        warp = np.cumsum(warp) -1

        return(warp)
```

Figure 68: Class to landmark register the biomechanical time series data

```python
def warpfnc(self,raw_sig,c_len,cutting):

   #create a copy of raw signal.
    c_sig = np.empty_like(raw_sig)
    c_sig[:] = raw_sig


    #Initiate function search with excessive threshold.
    c_thresh = 10
    thresh = 10
    midpoint = round(len(c_sig)*0.5)
    c_sig[midpoint] = c_thresh

    #interpolate between start mid and end of signal and calculate
    #the sum of the signal.
    x1 = np.where(~np.isnan(c_sig))
    x2 = np.where(np.isnan(c_sig))

    f = interpolate.Akima1DInterpolator(x1[0], c_sig[~np.isnan(c_sig)])
    c_sig[np.isnan(c_sig)] =  f(x2[0])

    #Binary search algorithm to find the midpoint value which will
    # provide an appropriate time warping function
    while abs(sum(c_sig) - c_len) >  0.001:
        if sum(c_sig)  > c_len:# clandmark occures early, must be slowed down.

            thresh = thresh/2
            c_thresh  = (c_thresh - thresh)

            c_sig = np.empty_like(raw_sig)
            c_sig[:] = raw_sig
            c_sig[midpoint] = c_thresh

            #interpolate between start mid and end of signal and calculate
            #the sum of the signal.
            f = interpolate.Akima1DInterpolator(x1[0], c_sig[~np.isnan(c_sig)])
            c_sig[np.isnan(c_sig)] =  f(x2[0])
```

Figure 69: Method of the LandMarkReg class to dynamically warp the timing of the signals (1 of 2)

```
        # in case there are minus values make them 0
        c_sig[c_sig<0] = 0;

        # remove first data point if requested
        if cutting == True:
            c_sig = c_sig[1:]

    elif sum(c_sig)  < c_len: # clandmark occures late, must be sped up.
        thresh = thresh/2
        c_thresh  = (c_thresh + thresh)
        c_sig = np.empty_like(raw_sig)
        c_sig[:] = raw_sig
        c_sig[midpoint] = c_thresh

        #interpolate between start mid and end of signal and calculate
        #the sum of the signal.
        f = interpolate.Akima1DInterpolator(x1[0], c_sig[~np.isnan(c_sig)])
        c_sig[np.isnan(c_sig)] =  f(x2[0])

        # remove first data point if requested
        if cutting == True:
            c_sig = c_sig[1:]

return(c_sig)
```

Figure 70: Method of the LandMarkReg class to dynamically warp the timing of the signals (2 of 2)

3.4.2 Analysis of Characterising Phases

In order to reduce the dimensionality of the data and extract key features, the concept of 'Analysis of Characterizing Phases' (ACP) was used to generate participant scores that represent the movement of each participant within key phases of variation using VARIMAX rotated principal components (Richter et al.; 2014). Each score captures the samples movement for each identified phase (k) as the summed difference between a participant' s waveform (p) and the mean waveform (q) for each time point (i) between the start (n) and end (m) of a phase. This was completed for each biomechanical waveform (j) (Equation 1):

$$feature_{j,k} = \sum_{i=n}^{m} p(i) - q(i) \tag{1}$$

The following figures (Figure 71 and Figure 72) depecit the class written based on the paper by Richter et al. (2014).

```python
import numpy as np

class ACP(object):
    """This is a class used to extract key phases from time series data
    based on the method 'Analysis of Characterising Phases' as described
    in the paper by Richter et al. (2014).

    Attributes:
        data: An n*m array where each row represents a case and each column
        represents a time point.
        threshold: An int (10:90) representing the percentage of the peak
        vector loading that should be extracted. Default = 90.


    Richter, C., O'Connor, N. E., Marshall, B., & Moran, K. (2014).
    Analysis of characterizing phases on waveforms: an application to vertical
    jumps. Journal of applied biomechanics, 30(2), 316-321.

    Writen by Shane Gore. Contact: Shane.Gore2@Gmail.com
    """

    def __init__(self,data = '',threshold = 90):

        self.data = data
        self.threshold  = threshold

    def identify_phases(self):

        #remove signals with nans. Fast check first.
        if np.isnan(np.min(self.data)):
            self.data = self.data[~np.isnan(self.data).any(axis=1)]

        #zero mean data
        data_mean = np.mean(self.data,axis=0)  #Compute mean of each time point
        data_centered = self.data - np.tile(data_mean,(self.data.shape[0],1))

        #Calculate the covariance matrix
        cov_data = np.cov(data_centered.T)

        #Compute Eigen decomosition and order by eigen values
        eigen_values, eigen_vectors = np.linalg.eig(cov_data)
        desc_order = np.flip(np.argsort(eigen_values))
        eigen_values = eigen_values[desc_order]
        eigen_vectors = np.real(eigen_vectors[:, desc_order])

        #Calculate variance explained
        var_explained = (eigen_values / np.sum(eigen_values)) *100
```

Figure 71: Class to calculate ACP phases (1 of 2)

```
#Retain eigen vectors that explain at least 1% of the variance
princ_comp = eigen_vectors[:,var_explained > 1]

#Apply Varimax Rotation to minimize the variance of the squared
#components thereby maximising individual component loading.
rot_comps = self.varimax(princ_comp)

#Extract key phases
phase_start, phase_end  = self.PC2_keyphase(rot_comps)

#Arrange key phases in descending order
desc_order   =  np.argsort(phase_start, axis=0)
phase_start  = phase_start[desc_order]
phase_end   = phase_end[desc_order]

#Merge overlapping phases if present.
c_phase_end = np.empty(0,int)
c_phase_start = np.empty(0,int)
while len(phase_end) > 0:
        mergeidx = (abs(phase_end[0] - phase_end)< 2)
        if  (np.sum(mergeidx) > 1):
            c_phase_end = np.append(c_phase_end,max(phase_end[mergeidx]))
            c_phase_start = np.append(c_phase_start,min(phase_start[mergeidx]))
            phase_end = np.delete(phase_end,np.where(mergeidx))
            phase_start = np.delete(phase_start,np.where(mergeidx))
        else:
            c_phase_end   = np.append(c_phase_end,phase_end[0])
            c_phase_start = np.append(c_phase_start,phase_start[0])
            phase_end = np.delete(phase_end,0)
            phase_start  = np.delete(phase_start,0)


return(c_phase_start.astype('int'), c_phase_end.astype('int'))
```

Figure 72: Class to calculate ACP phases (2 of 2)

The figure below depicts the method to extract the key phases from the principle component waveforms (Figure 73).

```
def PC2_keyphase(self,rot_comps):
    #Function to identify key phases of the rotated
    #principle compnonents.

    threshold = round((self.threshold /100),2)

    #convert rotated PC vectors to absolute values
    rot_comps = abs(rot_comps)

    #identiy peak of PC vectors
    peak_pos = np.argmax(rot_comps,axis=0)
    peak  = np.max(rot_comps,axis=0)

    phase_start = np.zeros((rot_comps.shape[1],1))
    phase_end = np.zeros((rot_comps.shape[1],1))

    for n in range(rot_comps.shape[1]):
        try:
            phase_start[n] = np.max(np.where(rot_comps[0:peak_pos[n]+1,n]
            < peak[n] * threshold))
        except:
            phase_start[n] = 0 #if threshold not found
        try:
            phase_end[n] =  peak_pos[n] + np.min(np.where
            (rot_comps[peak_pos[n]:rot_comps.shape[0]+1,n]
            < peak[n] * threshold)) + 1
        except:
            phase_end[n] = rot_comps.shape[0] #if threshold not found

    return (phase_start,phase_end)
```

Figure 73: Method of the ACP class to extract phases from the principle component waveforms

The identified principle component waveforms are then varimax rotated (Figure 74).

```
def varimax(self,Phi, gamma = 1, q = 20, tol = 1e-6):
# Function to rotate prinicple components to minimize the variance
#of the squared components.

#Adapted from https://en.wikipedia.org/wiki/Talk:Varimax_rotation

    p,k = Phi.shape
    R = np.eye(k)
    d=0
    for i in range(q):
        d_old = d
        Lambda = np.dot(Phi, R)
        u,s,vh = np.linalg.svd(np.dot(Phi.T,np.asarray(Lambda)**3 -
                        (gamma/p) * np.dot(Lambda,
                        np.diag(np.diag(np.dot(Lambda.T,Lambda))))))
        R = np.dot(u,vh)
        d = np.sum(s)
        try:
            if d/d_old  < tol: break
        except:
            continue #account for division by zero.
        return np.dot(Phi, R)
```

Figure 74: Method of the ACP class to varimax rotate principle components

4 Implementation: Clustering

In order to identify the presence of naturally occurring foot-strike patterns, six clustering algorithms (K-means, Hierarchical, Spectral, OPTICS, HDBSCAN, Mean Shift) were implemented and assessed. The required modules were firstly loaded (Figure 75).

```
'''This script was written for the MSc project entitled 'The identification of foot-strike patterns
using unsupervised learning and their association with injury'. This script represents the implementation
of the clustering solutions.
An overview of the steps:
    Minority class over-sampled.
    Feature selection conducted with Spectral Feature Selection.
    Data tested with Hopkins statistic.
    Six algorithmns applyed to data(Kmeans, Hierarchial, Spectral, Optics, HDBSCAN, Mean Shift)
    Clustering solutions evaluated using bootstrapped Adjusted rand index.
    Clustering Solution statistically tested with Welch ANOVA and Games-Howell post hoc tests.
    Clustering solutions vizualised.

    #Written by Shane Gore 2020 Cotact: Shane.Gore2@gmail.com

'''
##############################################################################
#                            Import Packages and data                        #
##############################################################################
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import sklearn.cluster as cluster
from imblearn.over_sampling import SMOTE
import seaborn as sns
import hdbscan
from sklearn.utils import resample
from sklearn.metrics import adjusted_rand_score
import pingouin as pg
from itertools import compress
import ptitprince as pt
from sklearn.decomposition import PCA

os.chdir('D:\\MSc Thesis\\MSc Files2')

#custom modules
from cluster_validation import cluster_validation
```

Figure 75: Loading the required modules

The required data generated in the general implementation phase was then read in, the data class was rebalanced with SMOTE and the data was visualised (Figure 76).

```
##############################################################################
#                            Footstrike based Clustering                     #
##############################################################################

#Read in data for clusering
Features_scaled = pd.read_csv('Features_scaled_preprocessed_ds.csv')
registered_data_conditions = pd.read_csv('registered_data_conditions.csv')
registered_data = pd.read_csv("registered_data.csv")
paticipant_class = pd.read_csv("paticipant_class.csv")
data = pd.read_csv("data.csv")
data_conditions = pd.read_csv("data_conditions.csv")

data.head()
data_conditions.head()
Features_scaled.to_csv('Features_scaled_preprocessed_ds.csv', index=False)

# Split on foot data
Feature_names = list(Features_scaled)
Foot_Features = Features_scaled.iloc[:,['foot' in var.lower() for var in Feature_names]]
Foot_names = list(Foot_Features)

#Conduct SMOTE to rebalance the minority class
oversample = SMOTE(random_state=42,k_neighbors=5)
X_res, y_res = oversample.fit_resample(Foot_Features.iloc[:,:-1],
                                       Features_scaled.iloc[:,-1].astype('int'))
#Plot oversampled data
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X_res)
fig, ax = plt.subplots(figsize=(10, 5))
ax.scatter(principalComponents[:,0], principalComponents[:,1],alpha=0.70, c= y_res)
```

Figure 76: Loading the data, rebalancing the classes and visualisation

Using the concept of spectral feature selection, the number of features to retain for the clustering solution was determined by visual inspection of the spec scores (Zhao and Liu; 2007), and the five retained features produced a Hopkins statistic of 0.96, suggesting high clusterability of the data (Figure 77).

```
#Five features selected based on Generic Spectural Feature Selection
os.chdir('D:\\MSc Thesis\\MSc Files2\\fsfc-master')
from fsfc.generic.SPEC import GenericSPEC
select2 = GenericSPEC(5)
Gspec = select2.fit(X_res[:,:])
scores_idx = np.argsort(Gspec.scores*-1)

#plot spec scores
fig, ax = plt.subplots(figsize=(10, 5))
plt.plot(Gspec.scores[scores_idx])
plt.xlabel('Features', fontsize=14)
plt.ylabel('Score', fontsize=14)
plt.suptitle('Spectral Feature Selection', fontsize= 14)
fig.savefig('FS_cluster_SPEC')

#Data Mining the Textbook (Hopkins Statistic)
os.chdir('D:\\MSc Thesis\\MSc Files2')
from cluster_feature_selection import cluster_feature_selection
cfs = cluster_feature_selection(X_res[::5,Gspec.get_support()])
best_score,best_indices = cfs.hopkins_feature_selection()

#Delimit sample size for testing
X_res = X_res[:,Gspec.get_support()]
Foot_names = list(compress(Foot_names, Gspec.get_support()))
```

Figure 77: Spectral feature selection and statistical test for clusterability

To test the clustering algorithms in a repeatable manner, the various clustering parameters were saved as dictionary structure in a dataframe which would then be passed to a class for evaluation (Figure 78).

```
#Creata a dataframe containing all the clustering algorithms to be tested
cluster_types = pd.DataFrame(columns=['algorithm', 'args', 'kwds','model',
                                      'prediction_strength','adjusted_rand_index',
                                      'VIC_stability','silhouette_coefficient','density_coefficient'])
#Kmeans
for i in range(2,6):
    cluster_types = cluster_types.append(pd.DataFrame ({'algorithm': [cluster.KMeans],'args':[()],
                    'kwds': [{'init': 'k-means++','max_iter':200,'n_clusters':i}],
                    'model':[()],'prediction_strength':[0],'adjusted_rand_index':[0],
                    'VIC_stability':[0],'silhouette_coefficient':[0],
                    'density_coefficient':[0]}))
#Hierachial clustering
linkages = {'ward', 'complete', 'average', 'single'}
for link in linkages:
    for i in range(2,6):
        cluster_types = cluster_types.append(pd.DataFrame({'algorithm':[cluster.AgglomerativeClustering],
                                'args':[()],'kwds': [{'linkage':link, 'n_clusters': i}],'model':[()],
                                'prediction_strength':[0],'adjusted_rand_index':[0],
                                'VIC_stability':[0],'silhouette_coefficient':[0],
                                'density_coefficient':[0]}))
#HBDSCAN
for i in [round(len(X_res) * 0.02),round(len(X_res) * 0.04),round(len(X_res) * 0.06)]: # min_cluster_size
    for x in range(1,int(round(np.log(len(X_res))))+1,2):
        cluster_types = cluster_types.append(pd.DataFrame ({'algorithm': [hdbscan.HDBSCAN],'args':[()],
                                'kwds': [{'min_samples':x,'min_cluster_size':i}],'model':[()],
                                'prediction_strength':[0],'adjusted_rand_index':[0],
                                'VIC_stability':[0],'silhouette_coefficient':[0],
                                'density_coefficient':[0]}))
#MeanShift
cluster_types = cluster_types.append(pd.DataFrame ({'algorithm': [cluster.MeanShift],'args':[()],
                                'kwds': [{ 'bin_seeding': False}],'model':[()],
                                'prediction_strength':[0],'adjusted_rand_index':[0],
                                'VIC_stability':[0],'silhouette_coefficient':[0],
                                'density_coefficient':[0]}))

#OPTICS (Ordering Points To Identify the Clustering Structure)
for i in range(5,25,5): # min_cluster_size
    cluster_types = cluster_types.append(pd.DataFrame ({'algorithm': [cluster.OPTICS],'args':[()],
                            'kwds': [{'min_samples':i,}],'model':[()],
                            'prediction_strength':[0],'adjusted_rand_index':[0],
                            'VIC_stability':[0],'silhouette_coefficient':[0],
                            'density_coefficient':[0]}))
 #Spectural clustering
assignmnets = {'discretize','kmeans'}
for assignmnet in assignmnets:
    for i in range(2,6):
        cluster_types = cluster_types.append(pd.DataFrame ({'algorithm': [cluster.SpectralClustering],
                        'args':[()],'kwds': [{'n_clusters':i,'assign_labels': "discretize"}],
                        'prediction_strength':[0],'adjusted_rand_index':[0],
                        'VIC_stability':[0],'silhouette_coefficient':[0], 'density_coefficient':[0]}))
```

Figure 78: Feature Selection with spectral feature selection

The clustering algorithms were then evaluated using the custom module (cluster validation) and the traditional foot-strike classifications (Altman and Davis; 2012) were calculated (Figure 79). For further details on the cluster validation module, please see section 4.1

```
#Cluster data
for x in range(len(cluster_types)):
    print(x)
    cluster_val = cluster_validation(X_res[:,:],cluster_types.algorithm.iloc[x],
                                     cluster_types.args.iloc[x],cluster_types.kwds.iloc[x],
                                     ground_truth = y_res[:],repeats = 2)
    methods =  cluster_val.call_methods({'adjusted_rand_index': [],'VIC_stability': [],
                                         'silhouette_coefficient': [],
                                         'density_coefficient': [],
                                         'plot_clusters': 'foot_cluster_'+ str(x) +'.png'})
    cluster_types.loc[x ,'adjusted_rand_index'] = methods['adjusted_rand_index']
    cluster_types.loc[x ,'VIC_stability'] = methods['VIC_stability']
    cluster_types.loc[x ,'silhouette_coefficient'] = methods['silhouette_coefficient']
    cluster_types.loc[x ,'density_coefficient'] = methods['density_coefficient']
    cluster_types.loc[x ,'model'] = methods['model']

#Traditional Footstrike classification
FA = data_conditions[data_conditions.conditions_t4 == 'FootAngles_fle']
Foot_Angle_IC = FA.reset_index(drop=True)
Foot_Angle_IC = Foot_Angle_IC.join(paticipant_class.reset_index())
FA_data = pd.DataFrame(data[data_conditions.conditions_t4 == 'FootAngles_fle'][['0']]).reset_index()
Foot_Angle_IC = Foot_Angle_IC.join(FA_data[['0']])
Foot_Angle_IC = Foot_Angle_IC.rename(columns={'0': 'Foot_Angle'})
#Classify footstrikes
Foot_Angle_IC['FootStrike_Class'] = 'x'
Foot_Angle_IC.loc[Foot_Angle_IC.Foot_Angle > 8,'FootStrike_Class'] = 0 #'RFS'
Foot_Angle_IC.loc[Foot_Angle_IC.Foot_Angle < -1.6,'FootStrike_Class'] = 1 #'FFS'
Foot_Angle_IC.loc[(Foot_Angle_IC.Foot_Angle > -1.6) &
                  (Foot_Angle_IC.Foot_Angle < 8),'FootStrike_Class'] = 2 #'MFS'

(sum(Foot_Angle_IC['FootStrike_Class'] == 0))/len(Foot_Angle_IC) # 'RFS' 87%
(sum(Foot_Angle_IC['FootStrike_Class'] == 1))/len(Foot_Angle_IC) # FFS'1%
(sum(Foot_Angle_IC['FootStrike_Class'] == 2)) /len(Foot_Angle_IC) # 'MFS' 11.3%
```

Figure 79: Inital Evaluation of the clustering approaches and classification of foot-strike angle

In the final evaluation of the clustering algorithms, 100 bootstrapped Adjusted Rand Index (ARI) scores were calculated the four best clustering solutions along with the traditional classification approach (Figure 80).

```
#Clustering Method 1
ARI_KM = bootstrap_ARI(X_res[:,:],cluster_types.algorithm.iloc[1],
                                    cluster_types.args.iloc[1],
                                    cluster_types.kwds.iloc[1],
                                    ground_truth = y_res[:],
                                    cluster_labels = None)
 #Clustering Method 2
ARI_H = bootstrap_ARI(X_res[:,:],cluster_types.algorithm.iloc[8],
                                    cluster_types.args.iloc[8],
                                    cluster_types.kwds.iloc[8],
                                    ground_truth = y_res[:],
                                    cluster_labels = None)
 #Clustering Method 2
ARI_O = bootstrap_ARI(X_res[:,:],cluster_types.algorithm.iloc[39],
                                    cluster_types.args.iloc[39],
                                    cluster_types.kwds.iloc[39],
                                    ground_truth = y_res[:],
                                    cluster_labels = None)
 #Clustering Method 2
ARI_S = bootstrap_ARI(X_res[:,:],cluster_types.algorithm.iloc[47],
                                    cluster_types.args.iloc[47],
                                    cluster_types.kwds.iloc[47],
                                    ground_truth = y_res[:],
                                    cluster_labels = None)
#Traditional Approach
ARI_trad = bootstrap_ARI(X_res[:,:],cluster_types.algorithm.iloc[2],
                                    cluster_types.args.iloc[2],
                                    cluster_types.kwds.iloc[2],
                                    ground_truth = y_res[:],
                                    cluster_labels = np.array(Foot_Angle_IC['FootStrike_Class']))
```

Figure 80: Evaluation of the best clustering solutions with boostrapped ARI

The functions used to carry out this bootstrapped ARI testing are presented below (Figure 81).

```
#Calculate Bootstrapped Adjusted Rand Index for best performing Methods.
def bootstrap_ARI (data,algorithm,args,kwds,ground_truth,cluster_labels):
    #This function generates bootstrapped ARI scores.
    n_iterations = 100
    n_size = round(len(data) * 0.70)

    data = np.column_stack([data,ground_truth])

    if any(cluster_labels == None):

        cluster_info, cluster_labels = find_clusters(data[:,:-1],algorithm,args,kwds)

        stats = list()
        for i in range(n_iterations):
            print(i)
            c_data = resample(data, n_samples = n_size, random_state = i)
            if 'kmeans' in str(algorithm).lower():
                cluster_labels = cluster_info.predict(c_data[:,:-1])
            else:
                try:
                    cluster_info = find_clusters(c_data[:,:-1],algorithm,args,kwds)
                    cluster_labels = cluster_info.labels_
                except:
                    cluster_info, cluster_labels = find_clusters(c_data[:,:-1],algorithm,args,kwds)
            ARI = adjusted_rand_score(c_data[:,-1], cluster_labels)
            stats.append(ARI)
    else:
        #This is used for the predefined classification
        stats = list()
        for i in range(n_iterations):
            print(i)
            c_data = resample(data, n_samples = n_size,random_state = i)
            c_cluster_labels = resample(cluster_labels,n_samples = n_size,random_state = i)
            ARI = adjusted_rand_score(c_data[:,-1], c_cluster_labels)
            stats.append(ARI)
    return(stats)

def find_clusters(data,algorithm,args,kwds):
    #function to find clusters in data.
    print('clustering data')

    clusters = algorithm(*args, **kwds).fit(data)
    cluster_labels = algorithm(*args, **kwds).fit_predict(data)
    return(clusters, cluster_labels)
```

Figure 81: Functions used to calculate bootstrapped ARI scores

The ARI scores were then visualised using rain cloud plots and statistically tested using a one-way Welch's ANOVA with Games-Howell post hoc follow up test. Finally, to compare the ARI scores for each model with zero (random assignment) a series of one sample welch t-tests with holm's correction for multi-comparisons were conducted (Figure 82).

```
#Plot ARI Resutls as rain clouds
ARI_results = pd.DataFrame([ARI_KM, ARI_H, ARI_O, ARI_S, ARI_trad])
ARI_results = ARI_results.T
ARI_results.columns = ['K-means', 'Hierarchical', 'OPTICS', 'Spectral', 'Taditional']
ARI_results = pd.melt(ARI_results)
pal = sns.color_palette(n_colors=1)
dx = "variable"; dy = "value"; ort = "v"; pal = "Set2"; sigma = .2
fig, ax = plt.subplots(figsize=(7, 5))
pt.RainCloud(x = dx, y = dy, data = ARI_results, palette = pal, bw = sigma,
                  width_box = 0.3, width_viol = .6, ax = ax, orient = ort)
ax.set_xlabel('Model',fontsize=14);
ax.set_ylabel('Bootstrapped Adjusted Rand Index',fontsize=14);
plt.xticks(rotation=45)
fig.savefig('Footstrike_Bootstrapped_ARI.png',dpi=300, transparent=False, bbox_inches='tight')

#Test for homogenity of variance
pg.homoscedasticity(dv='value', group='variable', data=ARI_results)

#Statistically compare approaches with one way Welch ANOVA
table = pg.welch_anova(data=ARI_results, dv='value', between='variable')

# post hoc gameshowell test with
ph = pg.pairwise_gameshowell(dv='value', between='variable', data=ARI_results,effsize ='cohen')
ph.to_csv('Foot_post_hoc_clusters.csv', index=False)
table.to_csv('Foot_Anova_clusters.csv')

# One sample welch t-test with holm correction.
owt = pd.DataFrame()
for i in range(len(ARI_results.T)):
    owt = owt.append(pg.ttest(ARI_results.iloc[:,i],0, tail = 'two-sided' ,correction='auto').round(2))
pg.multicomp(np.array(owt[['p-val']]),method = 'holm')
```

Figure 82: Visualisation and statistical testing of the bootstrapped ARI scores

4.1 Cluster Validation

In order to validate the clustering solutions, a custom class was written to validate each clustering algorithm in a repeatable manner (Figure 84 , Figure 85, Figure 86). The class contains several methods which will be detailed in turn. The prediction strength method (Tibshirani and Walther; 2005) will not be detailed here as within this current project, it was superseded by the cluster validation approach (Rodríguez et al.; 2018) which is based on the same concept of cluster stability. However, unlike the prediction strength method, the cluster validation index is suitable for all clustering types, not just does based on a distance metric (Rodríguez et al.; 2018).

```python
import os
import pandas as pd
import numpy as np
from sklearn.metrics import adjusted_rand_score
from sklearn.metrics import silhouette_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from hdbscan.validity import validity_index

class cluster_validation(object):
    """This is a class used to validate clusters using unsupervised methods. It
       is written as a wrapper for Scikit-learn Clustering algorithms and should
       support any Scikit-learn compatible algorithms (untested).

       Methods:
       adjusted_rand_index

           This is a wrapper method for the Scikit-learn evaluation metric
           adjusted_rand_score. This approach uses the adjusted rand index
           (Hubert and Arabi 1985), to detmerine the agreement between the
           know class label and the clustering solution.
           For more info,see: https://scikit-learn.org/stable/modules/generated
           /sklearn.metrics.adjusted_rand_score.html

       prediction_strength

           This method uses predictive strength to assess clustering performance
           as described by Tibshirani and Walther (2005). As per this approach
           euclaidian distance is used to determine cluster membership.
           This method is currently only supported for K-means clustering.
           Outcome: mean percentage of data points with stable cluster
           memebership. Experiments for selecting k, suggests taking
           the highest k for which prediction stregth is above 80
           (Tibshirani and Walther 2005).

      silhouette_coefficient

           This is a wrapper method for the Scikit-learn evaluation metric
           Silhouette Coefficient (Rousseeuw, 1987). This approach evaluates
           clustering solutions by its within cluster distance to between
           cluster distance.
           For more info,see: https://scikit-learn.org/stable/modules/
           clustering.html#silhouette-coefficient
```

Figure 83: Class to evaluate clustering solutions (1 of 3)

```
density_coefficient

     This approach is based on the method outlined in Moulavi et al
     (2014). This approach evaluates clustering solutions by its within
     cluster density to between cluster density.

VIC_stability

     This approach is based on the cluster validation index method
     outline in Rodríguez (2018).This method uses an ensemble of supervised
     learners and 10 fold cross validation to evalutate the clustering
     solution. The basic premise is that a good clustering solution should
     invoke a good classifier.

call_methods:

     A method to call several of the above methods on a single clustering
     solution.


ttributes:
  data: An n x m array, where each row represents a case and each column
      represents a feature.
  algorithm: Name of the clustering algorithm to be tested. Currently
      supports Scikit-learn clustering methods and should support any
      approach that is Scikit-learn compatible.

      Additionaly algorithms tested: hdbscan (McInnes et al 2017)
      https://github.com/scikit-learn-contrib/hdbscan

  args: A variable number of arguments to be passed to the cluster
      algorithm function.

  kwds: A keyworded, variable-length argument list to be passed to the
      cluster algorithm function.

  repeats: An int representing the number of times the validation should
      be tested on random splits.

  ground_truth: An optional argument used when the class labels are known.

  classifiers: An optional argument of what classifiers to use in the
      VIC_stability method. If not provided, default classifiers used
      similar to Rodríguez (2018).
```

Figure 84: Class to evaluate clustering solutions (2 of 3)

```
    cluster_labels: An optional argument which is assigned if the call_methods
        method is utilised.

    cluster_info: An optional argument which is assigned if the call_methods
        method is utilised.

 References:

 L. McInnes, J. Healy, S. Astels, "hdbscan: Hierarchical density based
 clustering", Journal of Open Source Software,
 The Open Journal, volume 2, number 11. 2017

 R. Tibshirani, G. Walther, "Cluster Validation by Prediction Strength",
 American Statistical Association, Institute of Mathematical Statistics
 and Interface Foundation of North America 2005, Journal of Computational
 and Graphical Statistics, Volume 14 Number 3, Pages 511-528.
 Source: http://pubs.amstat.org/doi/abs/10.1198/106186005X59243

 Peter J. Rousseeuw (1987). “Silhouettes:
 a Graphical Aid to the Interpretation and Validation of Cluster Analysis”.
 Computational and Applied Mathematics 20: 53–65.
 doi:10.1016/0377-0427(87)90125-7.

 Hubert, Lawrence, and Phipps Arabie. "Comparing partitions."
 Journal of classification 2, no. 1 (1985): 193-218.

 Moulavi, D., Jaskowiak, P.A., Campello, R.J., Zimek, A. and Sander, J.,
 2014, April. Density-based clustering validation. In Proceedings of
 the 2014 SIAM international conference on data mining (pp. 839-847).
 Society for Industrial and Applied Mathematics.

 Rodríguez, J., Medina-Pérez, M.A., Gutierrez-Rodríguez, A.E.,
 Monroy, R. and Terashima-Marín, H., 2018. Cluster validation
 using an ensemble of supervised classifiers. Knowledge-Based Systems,
 145, pp.134-144.

Written by Dr Shane Gore 2020, Contact Shane.Gore2@Gmail.com

"""

def __init__(self, data, algorithm, args, kwds,repeats = None,ground_truth = None,
             classifiers = None, cluster_labels = None, cluster_info = None):

    self.data = data
    self.algorithm = algorithm
    self.args = args
    self.kwds = kwds
    self.repeats = repeats - 1 #zero indexing
    self.ground_truth = ground_truth
    self.classifiers = classifiers
    self.cluster_labels = cluster_labels
    self.cluster_info   = cluster_info
```

Figure 85: Class to evaluate clustering solutions (3 of 3)

4.1.1 Method: Cluster Validation Index

This method is based on the cluster validation index outlined in Rodríguez (2018). This method uses an ensemble of supervised learners and 5 fold cross validation to evaluate the clustering solution (Figure 86). The basic premise is that a good clustering solution should invoke a good classifier.

```
def VIC_stability(self):
    #Based on the paper by Rodriguez et al 2018

    print('calculating VIC')

    c_data = self.data
    c_cluster_labels = self.cluster_labels

        #Cluster data
    if all(self.cluster_labels == None):
        self.cluster_info, self.cluster_labels = self._find_clusters(self.data,
                                                                     self.algorithm,self.args,self.kwds)

    if (self.classifiers == None):
        print("No classifiers provided, using journal default")
         #This approach uses the classifier proposed in the paper by Rodriguez et al 2018
        try:

            clf1 = LogisticRegression(max_iter = 200)
            clf2 = RandomForestClassifier(n_estimators=100, random_state=0)
            clf3 = GaussianNB()
            clf4 = SVC(kernel='rbf',C = 1, gamma ='auto', probability=True)
            clf5 = KNeighborsClassifier(n_neighbors=1, metric = 'euclidean' )
            clf6 = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
            eclf = VotingClassifier(estimators=[('lr',clf1),('rf',clf2),('gnb',clf3),('svc',clf4),
                                                ('knn',clf5),('lda',clf6)],voting='soft')

            #calculate the cross validated auc score
            aucs = []
            unique, counts = np.unique(c_cluster_labels, return_counts=True)
            if len(set(c_cluster_labels)) < 3:
                if min(counts) < 2:
                    return('Na')
                else:
                    aucs  = cross_val_score(eclf,c_data, c_cluster_labels, cv=5,scoring='roc_auc')
            else:
                if min(counts) < 2:
                    return('Na')
                elif min(counts) < 5:
                    k = min(counts)
                else:
                    k = 5
                cv = StratifiedKFold(n_splits=k)
                for train, test in cv.split(c_data, c_cluster_labels):
                    prediction = eclf.fit(c_data.iloc[train,:],
                                          c_cluster_labels[train]).predict_proba(c_data.iloc[test,:])
                    aucs.append(roc_auc_score(c_cluster_labels[test],
                                              prediction, multi_class = 'ovo',average="macro"))

            return (sum(aucs)/len(aucs))
        except:
            return ('Na')
```

Figure 86: Method to calculate the cluster validation index

4.1.2 Method: Silhouette Coefficient

This is a wrapper method for the Scikit-learn evaluation metric Silhouette Coefficient (Rousseeuw; 1987). This approach evaluates clustering solutions by its within cluster distance to between cluster distance (Figure 87).

```
def silhouette_coefficient(self):

    print('calculating silhouette_score')

    if all(self.cluster_labels == None):
        self.cluster_info, self.cluster_labels = self._find_clusters(self.data,
                                                                     self.algorithm,self.args,self.kwds)
    try:
        # Calculate adjusted rand score
        SS = silhouette_score(self.data, self.cluster_labels, metric='euclidean')

        return(SS)
    except:
        return('Na')
```

Figure 87: Method to calculate the silhouette coefficient

4.1.3 Method: Density Coefficient

This approach is based on the method outlined in Moulavi et al. (2014). This approach evaluates clustering solutions by its within cluster density to between cluster density (Figure 88).

```
def density_coefficient(self):
    #implement density based cluster validation.
    print('calculating density_coefficient')

    #Cluster data
    if all(self.cluster_labels == None):
        self.cluster_info, self.cluster_labels = self._find_clusters(self.data,
                                                                     self.algorithm,self.args,self.kwds)
    try:
        DB_CV = validity_index(self.data, self.cluster_labels)
    except:
        DB_CV = 'Na'

    return(DB_CV)
```

Figure 88: Method to calculate the density based validation index

4.1.4 Method: Adjusted Rand Index

This is a wrapper method for the Scikit-learn Adjusted rand score evaluation metric. This approach uses the adjusted rand index (Hubert and Arabie; 1985), to determine the agreement between the know class label and the clustering solution (Figure 89).

```
def adjusted_rand_index(self):
    #Funcation to calculate adjusted rand index.
    assert any(self.ground_truth != None), "Please provide the argument ground_truth"

    print('calculating ARI')

    #Cluster data
    if all(self.cluster_labels == None):
        self.cluster_info, self.cluster_labels = self._find_clusters(self.data,
                                                                     self.algorithm,self.args,self.kwds)

    # Calculate adjusted rand score
    ARI = adjusted_rand_score(self.ground_truth, self.cluster_labels)

    return(ARI)
```

Figure 89: Method to calculate the adjusted rand index

4.1.5 Method: Plot Clusters

This method plots the clustering solution labels over the first two prinicple components of the data with the calculated metric overlayed (Figure 90 and Figure 91).

```
def plot_clusters(self,filename,cluster_types):

    print('Plotting clusters')

    #Cluster data
    if all(self.cluster_labels == None):
        self.cluster_info, self.cluster_labels = self._find_clusters(self.data,
                                                                     self.algorithm,self.args,self.kwds)

    #Read in the clustering metrics.
    if 'kmeans' in str(cluster_types.algorithm).lower():
        n_clusters = [d.get('n_clusters') for d in cluster_types.kwds]
        Algorithm  = 'Kmeans (k = '+ str(n_clusters[0]) + ')'
    elif 'fastcluster' in str(cluster_types.algorithm).lower():
        link = [d.get('link') for d in cluster_types.kwds]
        Algorithm  = 'Hierarchical (Linkage = '+ str(link[0]) + ')'
    elif 'agglomerative' in str(cluster_types.algorithm).lower():
        link = [d.get('linkage') for d in cluster_types.kwds]
        n_clusters = [d.get('n_clusters') for d in cluster_types.kwds]
        Algorithm  = 'Hierarchical (Linkage = '+ str(link[0]) +'(k = '+ str(n_clusters[0]) + ')' ')'
    elif 'meanshift' in str(cluster_types.algorithm).lower():
        Algorithm  = 'Mean Shift '
    elif 'optics' in str(cluster_types.algorithm).lower():
        min_samples = [d.get('min_samples') for d in cluster_types.kwds]
        Algorithm  = 'OPTICS (min samples = '+ str(min_samples[0]) + ')'
    elif 'hdbscan' in str(cluster_types.algorithm).lower():
         min_samples = [d.get('min_samples') for d in cluster_types.kwds]
         min_cluster_size = [d.get('min_cluster_size') for d in cluster_types.kwds]
         Algorithm  ='HDBSCAN (min size = '+str(min_cluster_size[0])+', min samples = '+str(min_samples[0])+')'
    elif 'spectral'  in str(cluster_types.algorithm).lower():
         n_clusters = [d.get('n_clusters') for d in cluster_types.kwds]
         assign_labels = [d.get('assign_labels') for d in cluster_types.kwds]
         Algorithm  = 'Spectral (k = '+ str(n_clusters[0]) +', Assignment = ' + str(assign_labels[0]) + ')'
```

Figure 90: Method to plot the clustering solution (1 of 2)

```
#Extract metrics
#ARI
ARI       = str(round(cluster_types['adjusted_rand_index'].iloc[0],3))
#VIC
if  (cluster_types['VIC_stability'].iloc[0] != 'Na'):
    VIC       = str(round(cluster_types['VIC_stability'].iloc[0],3))
else:
    VIC  = cluster_types['VIC_stability'].iloc[0]
#SS
if  (cluster_types['silhouette_coefficient'].iloc[0] != 'Na'):
    SS   = str(round(cluster_types['silhouette_coefficient'].iloc[0],3))
else:
    SS  = cluster_types['silhouette_coefficient'].iloc[0]
#DBCV
if (cluster_types['density_coefficient'].iloc[0] != 'Na'):
    DBCV      = str(round(cluster_types['density_coefficient'].iloc[0],3))
else:
    DBCV  = cluster_types['density_coefficient'].iloc[0]


pca = PCA(n_components=2)
principalComponents = pca.fit_transform(self.data)
fig, ax = plt.subplots(figsize=(10, 5))
ax.scatter(principalComponents[:,0], principalComponents[:,1],
           alpha=0.70, c= self.cluster_labels, zorder = 1)
plt.xlabel('PC1', fontsize=14)
plt.ylabel('PC2', fontsize=14)
plt.suptitle(Algorithm, fontsize= 14)
textstr = ('ARI:' + ARI + ' VIC:' + VIC + ' SS:' + SS + ' DBCV:' + DBCV )
props = dict(boxstyle='round', facecolor='wheat', alpha=0.2, zorder = 2)

ylim = list(ax.get_ylim())
ylim[1] +=  1
ax.set_ylim(ylim[0],ylim[1])
ax.text(0.5, .95, textstr, transform=ax.transAxes, fontsize=14,
horizontalalignment = 'center', verticalalignment='top', bbox=props,zorder = 10)

directory = os.getcwd() + '/cluster_figures'
if not os.path.exists(directory):
    os.makedirs(directory)

fig.savefig('cluster_figures/' + filename)

plt.close(fig)
```

Figure 91: Method to plot the clustering solution (2 of 2)

An example clustering plot is provided in Figure 92.

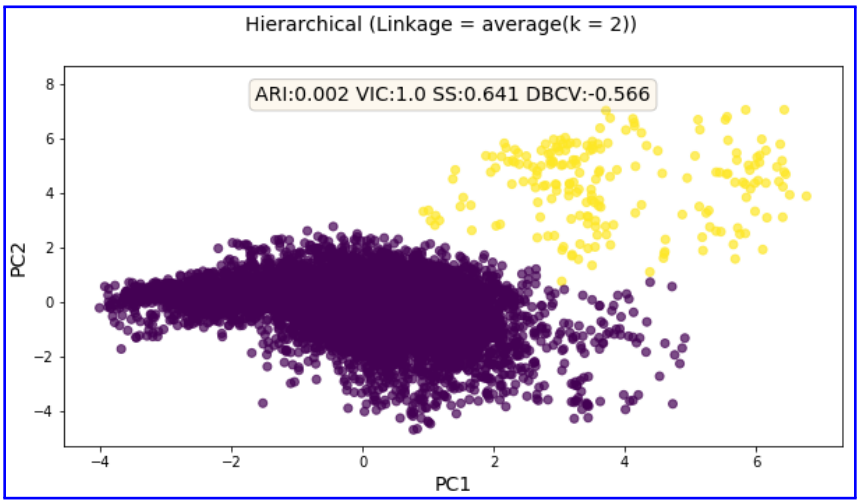


Figure 92: Example plot of a clustering solution

4.1.6 Method: Call Methods

The final method was utilised as a means of calling several of the above methods to evaluate a single clustering solution simultaneously and plotting the final solution (Figure 93).

```
def call_methods(self,methods):
    #This is a method which allows to calculate several metrics on the one
    #clustering solution at once.

        self.cluster_info, self.cluster_labels = self._find_clusters(self.data,
                                                                     self.algorithm,self.args,self.kwds)

        cluster_types = pd.DataFrame(columns=['algorithm', 'kwds',
                                              'adjusted_rand_index',
                                              'VIC_stability',
                                              'silhouette_coefficient',
                                              'density_coefficient'])
        algorithm = self.algorithm
        kwds      = self.kwds

        if ('adjusted_rand_index' in methods):
            adjusted_rand_index   = self.adjusted_rand_index()
        else:
            adjusted_rand_index = ''

        if ('VIC_stability' in methods):
            VIC_stability   = self.VIC_stability()
        else:
            VIC_stability = ''

        if ('silhouette_coefficient' in methods):
            silhouette_coefficient    = self.silhouette_coefficient()
        else:
            silhouette_coefficient = ''

        if ('density_coefficient' in methods):
            density_coefficient = self.density_coefficient()
        else:
            density_coefficient = ''

        if ('plot_clusters' in methods):
            cluster_types = cluster_types.append(pd.DataFrame ({'algorithm': [algorithm],'kwds': [kwds],
                                  'adjusted_rand_index':[adjusted_rand_index],
                                  'VIC_stability':[VIC_stability],
                                  'silhouette_coefficient':[silhouette_coefficient],
                                  'density_coefficient':[density_coefficient]}))

            self.plot_clusters(methods['plot_clusters'],cluster_types)

        for method in  methods:
            if method != 'plot_clusters':
                methods[method] = eval(method)

        methods = {**methods, **{'model': [self.cluster_info]}}

        return(methods)
```

Figure 93: Example plot of a clustering solution

5 Implementation: Classification

In order to determine if any of the biomechanics of the lower limb and trunk could predict those who would go on to become injured, six classification models (Naive Bayes, Elastic Net Logistic Regression, Bagged SVM, Random Forest, Adaboost and a weighted Stacked Ensemble) were implemented and assessed. The required packages were firstly loaded (Figure 94).

```
'''This script was written for the MSc project entitled 'The identification of foot-strike patterns
using unsupervised learning and their association with injury'. This script represents the implementation
of the classification solutions.

An overview of the steps:
   - Data split into train and test folds
   - Five classification algorithms trained (Naive Bayes, Elastic Net Logistic Regression,
                                             Bagged SVM, Random Forest, Adaboost):
        Initiate models with all features and random grid search of hyperparameters.
        Feature selection with a genetic search algorithm and recursive feature elimination.
        Tune models with either a greedy grid search or Baysian optimisation.
    -Train weighted stacked ensemble of the above algorithms.
    -Evaluate and visualise algorithms with 100 bootstrapped measures of Accuracy,
        Sensitivity and Specifity on the hold out test set.
    -Statistically test with Welch ANOVA and Games-Howell post hoc tests.

    Written by Shane Gore 2020 Cotact: Shane.Gore2@gmail.com
"""
##############################################################################
#                              Import Packages                               #
##############################################################################
import os
import pandas as pd
import numpy as np
from sklearn.feature_selection import VarianceThreshold
from imblearn.over_sampling import SMOTE
from itertools import compress
import itertools
from tqdm import tqdm
from colorama import Fore, Style
from copy import copy
from time import time
#Machine Learning and Statistics.
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from itertools import combinations
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import BaggingClassifier
import pingouin as pg
#Evaluation
from imblearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.utils import resample
from skopt import BayesSearchCV
from genetic_selection import GeneticSelectionCV
from sklearn.metrics import roc_curve
from scipy import stats
from sklearn.metrics import roc_auc_score
from scipy.stats import sem
#Visualisations.
import matplotlib.pyplot as plt
import seaborn as sns
import ptitprince as pt
```

Figure 94: Loading the required modules

The required data generated in the general implementation phase was then read in and split into train and test sets (Figure 95).

```
##############################################################################
#                    Import data and prep for analysis                       #
##############################################################################

os.chdir('D:\MSc Thesis\MSc Files2')

#Read in participant data first
Features_scaled_p = pd.read_csv('Participant_Features.csv')
#Split data into test and train datasets based on participants.
X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(Features_scaled_p.iloc[:,:-1],
                                                            Features_scaled_p.iloc[:,-1],
                                                   train_size=0.70,
                                                   random_state=42,
                                                   stratify= Features_scaled_p.iloc[:,-1])
#Read in Footstrike data
Features_scaled = pd.read_csv('Features_scaled_preprocessed_ds.csv')
Features_scaled_t = Features_scaled.drop(['Participant_ID'], axis = 1)

#Screen for low variance & high correlation
Feature_names = list(Features_scaled_t)
selector = VarianceThreshold(0.02)
selector.fit(Features_scaled_t)
Features_scaled_t = Features_scaled_t.iloc[:,selector.get_support()]
Feature_names = list(compress(Feature_names, selector.get_support()))

corr_matrix = Features_scaled_t.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]
Features_scaled_t = Features_scaled_t.drop(Features_scaled_t[to_drop], axis=1)
[Feature_names.remove(drop) for drop in to_drop]
Features_scaled = pd.concat([Features_scaled['Participant_ID'],Features_scaled_t],axis = 1)

#Split data
X_train = Features_scaled.iloc[Features_scaled['Participant_ID'].isin(X_train_p['Participant_ID']).values,:-1]
y_train = Features_scaled.iloc[Features_scaled['Participant_ID'].isin(X_train_p['Participant_ID']).values,-1]
X_train = X_train.drop(['Participant_ID'], axis = 1)

#point serial correaltion
stats_corr = []
for i in range(X_train.shape[1]):
    stats_corr.append(stats.pointbiserialr(X_train.iloc[:,i].values, y_train.values)[0])
Feature_idx  = abs(np.array(stats_corr)) >  0.05
Feature_idx = np.append(Feature_idx,True)
Features_scaled_t = Features_scaled_t.iloc[:,Feature_idx]
Features_scaled = pd.concat([Features_scaled['Participant_ID'],Features_scaled_t],axis = 1)

#Split data into test and train datasets.
X_train_og = Features_scaled.iloc[Features_scaled['Participant_ID'].isin(X_train_p['Participant_ID']).values,:-1]
y_train_og = Features_scaled.iloc[Features_scaled['Participant_ID'].isin(X_train_p['Participant_ID']).values,-1]
X_test_og = Features_scaled.iloc[~Features_scaled['Participant_ID'].isin(X_train_p['Participant_ID']).values,:-1]
y_test_og = Features_scaled.iloc[~Features_scaled['Participant_ID'].isin(X_train_p['Participant_ID']).values,-1]
X_train_og = X_train_og.drop(['Participant_ID'], axis = 1)
X_test_og = X_test_og.drop(['Participant_ID'], axis = 1)
```

Figure 95: Loading the data and spliting into train and test sets

A set of dictionaries were created to store the hyperparameter tuning grids for each of the classification models and to store useful metrics when evaluating the algorithms (Figure 96).

```
#Set up hyperparameter grid
parameters = {}

#Naive Bayes
parameters.update({"naive_bayes": {'gaussiannb__var_smoothing': np.logspace(0,-9, num=100)}})

#Elastic logistic regression
parameters.update({"ElasticNet": {"sgdclassifier__loss": ['log'],
                            "sgdclassifier__alpha": [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1,
                                                     0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 1.0],
                            "sgdclassifier__penalty": ["elasticnet"],
                            "sgdclassifier__l1_ratio":[0.10,0.20,0.30,0.40,0.50,0.60,0.70,0.80,0.90,1.0],
                            "sgdclassifier__n_jobs": [6]
                             }})
#Random Forest
parameters.update({"RandomForest": {
                                    "randomforestclassifier__n_estimators": [300, 500, 800, 1000],
                                    "randomforestclassifier__max_features": ["sqrt", "log2"],
                                    "randomforestclassifier__max_depth" : [3, 4, 5, 6, 7, 8],
                                    "randomforestclassifier__min_samples_split": [0.005, 0.01, 0.05, 0.10],
                                    "randomforestclassifier__min_samples_leaf": [0.005, 0.01, 0.05, 0.10],
                                    "randomforestclassifier__criterion" :["gini", "entropy"],
                                    "randomforestclassifier__n_jobs": [6]
                                     }})
#Adaboost
parameters.update({"AdaBoost": {
        "adaboostclassifier__base_estimator":[DecisionTreeClassifier(max_depth= ii) for ii in range(1,6)],
        "adaboostclassifier__n_estimators": [300, 500, 800, 1000],
        "adaboostclassifier__learning_rate": [0.001, 0.01, 0.05, 0.1, 0.25, 0.50, 0.75, 1.0]
                                }})

#Bagged support vector Machine
keys, values = zip(*{'C': [0.1,1,10,100],
                           'gamma': [0.1,1,10, 100],
                           'kernel': ["rbf", "poly", "sigmoid"]}.items())
permutations_dicts = [dict(zip(keys, v)) for v in itertools.product(*values)]
parameters.update({"SVC_bag": {
                                "baggingclassifier__base_estimator": [SVC(**params) for params in permutations_dicts],
                                 'baggingclassifier__max_samples': [0.30]}})

classifiers = {}
classifiers.update({"naive_bayes": GaussianNB})
classifiers.update({"ElasticNet": SGDClassifier})
classifiers.update({"RandomForest": RandomForestClassifier})
classifiers.update({"SVC_bag": BaggingClassifier})
classifiers.update({"AdaBoost": AdaBoostClassifier})

classifier_prefix = {}
classifier_prefix.update({"RandomForest":'randomforestclassifier__'})
classifier_prefix.update({"SVC_bag":'baggingclassifier__'})
classifier_prefix.update({"ElasticNet":'sgdclassifier__'})
classifier_prefix.update({"AdaBoost":'adaboostclassifier__'})
classifier_prefix.update({"naive_bayes":'gaussiannb__'})
classifier_results = {}
```

Figure 96: Storing the hyperparameters to be tested

After initialising the models with an appropriate hyperparameter solution using a random search of the hyperparameter grid, feature selection was conducted with a genetic search algorithm followed by recursive feature elimination. In order to enhance generalisability, the model with the smallest number of features within one standard error of the best solution (maximised area under the receiver operator curve) was chosen (Figure 97). For the cross-validation procedures, the training folds were rebalanced using synthetic minority oversampling technique.

```
for classifier in classifiers:

    # Reset splits
    X_train = copy(X_train_og)
    X_test =  copy(X_test_og)
    y_train = copy(y_train_og)
    y_test =  copy(y_test_og)

    clf  = classifiers[classifier]
    params = parameters[classifier]

    #Initiate classifier with broad search of hyperparameters using random search.
    try:
        imba_pipeline = make_pipeline(SMOTE(random_state=42),
                                      clf(random_state=42))
    except:
        imba_pipeline = make_pipeline(SMOTE(random_state=42),
                                      clf())
    random_search = RandomizedSearchCV(imba_pipeline, param_distributions= params, cv=3, scoring='roc_auc',
                            return_train_score=True, n_iter=35, verbose=7, random_state=42 , n_jobs = 6)
    random_search.fit(X_train, y_train)

    #Wrapper based Feature Selection  with random search hyper parameters.
    best_params ={key.replace(classifier_prefix[classifier],''):
        random_search.best_params_[key] for key in random_search.best_params_}

    #Genetic Algorithim
    c_pipeline = make_pipeline(SMOTE(random_state=42),
                                      clf(**best_params))
    selector = GeneticSelectionCV(c_pipeline,
                                      cv=5,
                                      verbose=1,
                                      scoring="accuracy",
                                      max_features= round(X_train.shape[1]/4),
                                      n_population=50,
                                      crossover_proba=0.8,
                                      mutation_proba=0.2,
                                      n_generations=200,
                                      crossover_independent_proba=0.5,
                                      mutation_independent_proba=0.05,
                                      tournament_size= 3,
                                      n_gen_no_change=35,
                                      caching=True,
                                      n_jobs= 6)
    selector = selector.fit(X_train, y_train)
    X_train_delim = X_train.iloc[:,selector.support_]
    X_test_delim = X_test.iloc[:,selector.support_]

    #Recursive Feature Elimention on remaining features
    best_score,best_indices = recursive_feature_selection(X_train_delim, y_train,c_pipeline, 1, None)
    suitable_scores = best_score > (max(best_score) - sem(best_score))
    n_features_to_select = np.where(np.flipud(suitable_scores))[0][0]
    best_subset = np.flipud(best_indices)[n_features_to_select]
    X_train_delim = X_train_delim.iloc[:,list(best_subset)]
    X_test_delim = X_test_delim.iloc[:,list(best_subset)]
```

Figure 97: Feature selection implementation

Given that the scikit-learn recursive feature elimination approach cannot accommodate every algorithm, a custom function was written (Figure 98).

```
def recursive_feature_selection(train, label,c_pipeline, min_k_features,max_k_features):
        #This is a custom function used when the sklearn is non compatiable
        if isinstance(train, pd.DataFrame):
            #temp conversion to np array
            train = np.array(train)

        if  max_k_features == None:
            dim = train.shape[1]
        else:
            dim = max_k_features

        c_indices = tuple(range(train.shape[1]))
        data = pd.DataFrame(train, columns = c_indices[:])

        best_score = []
        best_indices = []
        while dim > min_k_features:
            print(Fore.GREEN + ('selecting best features of size ' + str(dim)))
            print(Style.RESET_ALL)
            scores = []
            subsets = []
            i = 0
            for p in tqdm(combinations(c_indices, r= dim - 1)):
                i +=1
                print(i)
                c_data =  data.iloc[:,list(p[:])]
                score = cross_val_score(c_pipeline, c_data, label, cv=3, scoring='roc_auc', n_jobs = -1).mean()
                scores.append(score)
                subsets.append(p)
            best  =  np.argmax(scores)
            best_score.append(scores[best])
            best_indices.append(subsets[best])
            c_indices = subsets[best]
            dim -= 1
        return (best_score,best_indices)
```

Figure 98: Recursive feature elimination function

The final models were then tuned with greedy grid search and Bayesian optimisation (Figure 99).

```
# Fine tune model using bayes search
if classifier ==  'AdaBoost':
    #Minimis grid, to overcome bug in baysian search
    params = {'adaboostclassifier__n_estimators': [300, 500, 800, 1000],
            'adaboostclassifier__learning_rate': [0.001, 0.01, 0.05,0.1,0.25,0.5,0.75,1.0]}
    #Nested cross validation with baysian search of hyperparameters.
    def on_step(optim_result):
        score = bayes_grid.best_score_
        print("best score: %s" % score)
        if score >= 0.90:
            print('Interrupting!')
            return True

    bayes_grid = BayesSearchCV(imba_pipeline, search_spaces=params, cv = 3,
                               scoring='roc_auc',n_iter=20, verbose = 7, n_jobs = 6, random_state=42)
    bayes_grid.fit(X_train_delim, y_train, callback=on_step)
    cv_score = cross_val_score(bayes_grid, X_train_delim, y_train, cv=2, verbose = 7).mean()
    c_class = classifier_prefix[classifier].replace('__','')
    classifier_results.update({classifier:{
                                            "train_result": [cv_score],
                                            "model_params": [bayes_grid.best_estimator_[c_class]],
                                            "features":[list(X_train_delim)]
                                            }})

elif (classifier == 'SVC_bag' or classifier == 'naive_bayes' or classifier == 'ElasticNet'):
     # perform extensive grid search with best features.
    grid_search = GridSearchCV(imba_pipeline, param_grid= params, cv=3, scoring='roc_auc',
                        verbose=7, n_jobs = 6)
    grid_search.fit(X_train_delim, y_train)
    cv_score = np.nanmean(cross_val_score(grid_search, X_train_delim, y_train, cv=2, verbose = 7))
    c_class = classifier_prefix[classifier].replace('__','')
    classifier_results.update({classifier:{
                                            "train_result": [cv_score],
                                            "model_params": [grid_search.best_estimator_[c_class]],
                                            "features":[list(X_train_delim)]
                                            }})
else:
    #Nested cross validation with best subset & baysian search of hyperparameters.
    def on_step(optim_result):
        score = bayes_grid.best_score_
        print("best score: %s" % score)
        if score >= 0.90:
            print('Interrupting!')
            return True

    bayes_grid = BayesSearchCV(imba_pipeline, search_spaces=params, cv = 3,
                               scoring='roc_auc',n_iter=50, verbose = 7, n_jobs = 1, random_state=42)
    bayes_grid.fit(X_train_delim, y_train, callback=on_step)
    cv_score = cross_val_score(bayes_grid, X_train_delim, y_train, cv=2, verbose = 7).mean()
    c_class = classifier_prefix[classifier].replace('__','')
    classifier_results.update({classifier:{
                                            "train_result": [bayes_grid.best_score_],
                                            "model_params": [bayes_grid.best_estimator_[c_class]],
                                            "features":[list(X_train_delim)]
                                            }})
```

Figure 99: Hyperparameter tuning

After training the models, they were evaluated using 100 bootstrapped resamples of the hold out validation set for accuracy, sensitivity and specificity (Figure 100). The function used to calculate the evaluation metrics is presented in figure 101 while the function used to carry out the bootstrap resampling is presented in figure 102 - 103.

```
##############################################################################
#                              Model Evaluation                              #
##############################################################################

# Test final models on hold out validation set
class_vote = []
AROC = []
classifier_accuracy = []
class_conf =[]
for classifier in classifiers:

    X_train, X_test, y_train, y_test = train_test_split(Features_scaled.iloc[:,:-1], Features_scaled.iloc[:,-1],
                                                        train_size=0.70,
                                                        random_state=42,
                                                        stratify= Features_scaled.iloc[:,-1])

    X_test_delim = X_test[[*classifier_results[classifier]["features"][0]]]

    clf = classifier_results[classifier]['model_params'][0]


    accuracy,sensitivity,specificity,class_vote,AROC = bootstrap_evaluation(X_test_delim,
                                                                             clf,y_test,False,
                                                                             class_vote,False,AROC,class_conf)
    classifier_results[classifier]["test_acc"] =  accuracy
    classifier_results[classifier]["test_sens"] =  sensitivity
    classifier_results[classifier]["test_spec"] =  specificity
    classifier_accuracy.append(accuracy)
```

Figure 100: Bootstrapped evaluation implementation

```
def calculate_sensitivity_specificity(y_test, y_pred_test):

    actual_pos = y_test == 1
    actual_neg = y_test == 0

    # Get true and false test
    true_pos = (y_pred_test == 1) & (actual_pos)
    #false_pos = (y_pred_test == 1) & (actual_neg)
    true_neg = (y_pred_test == 0) & (actual_neg)
    #false_neg = (y_pred_test == 0) & (actual_pos)

    # Calculate accuracy
    accuracy = np.mean(y_pred_test == y_test)

    # Calculate sensitivity and specificity
    sensitivity = np.sum(true_pos) / np.sum(actual_pos)
    specificity = np.sum(true_neg) / np.sum(actual_neg)

    return sensitivity, specificity, accuracy
```

Figure 101: Function to calculate evaluation metrics

```python
def bootstrap_evaluation (data,algorithm,ground_truth,naive_model,class_vote,ensemble,AROC,class_conf):
    #This function generates bootstrapped resamples for evaluating the classification models.
    n_iterations = 100
    n_size = round(len(data) * 0.70)

    data = np.column_stack([data,ground_truth])
    accuracy =[]
    sensitivity =[]
    specificity =[]
    if naive_model == True:
        #Always predict majority class.
        for i in range(n_iterations):
            print(i)
            c_data = resample(data, n_samples = n_size, random_state = i)
            y_test = c_data[:,-1]
            yhat =  np.ones_like(y_test)

            sens, spec, acc = calculate_sensitivity_specificity(y_test, yhat)
            accuracy.append(acc)
            sensitivity.append(sens)
            specificity.append(spec)
            yhat_total =[]
            auc_total = []
            conf_total =[]
    elif ensemble == True:
          # Ensemble model
          for i in range(n_iterations):
            print(i)
            c_data = resample(data, n_samples = n_size, random_state = i)
            yhat =  class_vote.iloc[:,i]
            y_test = c_data[:,-1]

            sens, spec, acc = calculate_sensitivity_specificity(y_test, yhat)
            accuracy.append(acc)
            sensitivity.append(sens)
            specificity.append(spec)
            yhat_total =[]
            auc_total = []
            conf_total= []
```

Figure 102: Function to carry out bootstrapped resampling (1 of 2)

```
else:
    for i in range(n_iterations):
        print(i)
        c_data = resample(data, n_samples = n_size, random_state = i)
        #Calculate optimal cut threshold based on J statistic
        yhat = algorithm.predict_proba(c_data[:,:-1])
        yhat = yhat[:, 1] #keep only positive probabilities
        y_test = c_data[:,-1]
        #Calculate roc curves
        fpr, tpr, thresholds = roc_curve(y_test, yhat)
        auc = roc_auc_score(y_test, yhat)

        if auc > 0.53: # if area under curve is very bad, J optimisation appears to make the model worse
            # get the best threshold based on j statistic
            J = tpr - fpr
            best_thresh = thresholds[np.argmax(J)]
            #set best classification threshold
            yhat = (algorithm.predict_proba(c_data[:,:-1])[:,1] >= best_thresh).astype(bool)
            conf = abs(algorithm.predict_proba(c_data[:,:-1])[:,1] - best_thresh)
        else:
            yhat = (algorithm.predict_proba(c_data[:,:-1])[:,1] >= 0.5).astype(bool)  #default threshold
            conf = abs(algorithm.predict_proba(c_data[:,:-1])[:,1] - 0.5)

        #calcuate  performance metrics based on best threshold.
        sens, spec, acc = calculate_sensitivity_specificity(y_test, yhat)

        accuracy.append(acc)
        sensitivity.append(sens)
        specificity.append(spec)

        try:
            yhat_total = pd.concat([yhat_total,pd.DataFrame(yhat)], axis = 1)
        except:
            yhat_total = pd.DataFrame(yhat)
        try:
            conf_total = pd.concat([conf_total,pd.DataFrame(conf)], axis = 1)
        except:
            conf_total = pd.DataFrame(conf)
        try:
            auc_total = pd.concat([auc_total,pd.DataFrame(pd.Series(auc)) ], axis = 1)
        except:
            auc_total = pd.DataFrame(pd.Series(auc))

AROC.append(auc_total)
class_vote.append(yhat_total)
class_conf.append(conf_total)
return(accuracy,sensitivity,specificity,class_vote,AROC)
```

Figure 103: Function to carry out bootstrapped resampling (2 of 2)

The weighted stacked ensemble model was then calculated in a manner that took into account the classifier vote, the average classifier performance and the classifier confidence to optimise Youden's J statistic (Figure 104). Full details on this algorithm are presented in section 6.5.

```
#Weighted voting ensemble
class_vote2 = copy(class_vote)
class_conf2 = copy(class_conf)
class_vote =   copy(class_vote2)
tot_accuracy =[]
j_stat = []
for x in range(1,100,5):
    class_vote =   copy(class_vote2)
    class_conf = copy(class_conf2)
    for i in range(len(class_vote)):
        c_class_vote = class_vote[i].astype(int)
        class_vote[i] = c_class_vote.mask(class_vote[i] == False,-1)
    class_vote_weighted = (((class_vote[0] * class_conf[0]) * np.tile(np.array(AROC[0]**x),(len(class_vote[0]),1))
    + (class_vote[1] * class_conf[1]) * np.tile(np.array(AROC[1]**x),(len(class_vote[1]),1))
    + (class_vote[2] * class_conf[2]) * np.tile(np.array(AROC[2]**x),(len(class_vote[2]),1))
    + (class_vote[3] * class_conf[3]) * np.tile(np.array(AROC[3]**x),(len(class_vote[3]),1))
    + (class_vote[4] * class_conf[4]) * np.tile(np.array(AROC[4]**x),(len(class_vote[4]),1))) > 0).astype(bool)
    accuracy,sensitivity,specificity,class_vote,AROC  = bootstrap_evaluation(X_test_delim,clf,y_test,
                                                                              False,class_vote_weighted,
                                                                              True,AROC,class_conf)
    tot_accuracy.append(np.mean(accuracy))
    j_stat.append(np.mean((np.array(sensitivity) + np.array(specificity))  -1))
np.argmax(j_stat)
np.max(j_stat)
np.argmax(tot_accuracy)

# Best Stacked Model based on Youden's statistic
tot_accuracy =[]
j_stat = []
class_vote =   copy(class_vote2)
class_conf = copy(class_conf2)
for i in range(len(class_vote)):
    c_class_vote = class_vote[i].astype(int)
    class_vote[i] = c_class_vote.mask(class_vote[i] == False,-1)
class_vote_weighted = (((class_vote[0] * class_conf[0]) * np.tile(np.array(AROC[0]**15),(len(class_vote[0]),1))
+ (class_vote[1] * class_conf[1]) * np.tile(np.array(AROC[1]**15),(len(class_vote[1]),1))
+ (class_vote[2] * class_conf[2]) * np.tile(np.array(AROC[2]**15),(len(class_vote[2]),1))
+ (class_vote[3] * class_conf[3]) * np.tile(np.array(AROC[3]**15),(len(class_vote[3]),1))
+ (class_vote[4] * class_conf[4]) * np.tile(np.array(AROC[4]**15),(len(class_vote[4]),1))) > 0).astype(bool)
accuracy,sensitivity,specificity,class_vote,AROC  =  bootstrap_evaluation(X_test_delim,clf,y_test,
                                                                           False,class_vote_weighted,
                                                                           True,AROC,class_conf)
tot_accuracy.append(np.mean(accuracy))
j_stat.append(np.mean((np.array(sensitivity) + np.array(specificity))  -1))

# Save final model results
classifier = 'Stacked Ensemble'
classifier_results.update({classifier:{"test_acc": ['test_acc']}})
classifier_results[classifier]["test_acc"] = accuracy
classifier_results[classifier]["test_sens"] =  sensitivity
classifier_results[classifier]["test_spec"] =  specificity
```

Figure 104: Creating a weigthed stacked ensemble model

The naïve classifier was then assessed (Figure 105).

```
#Evaluate Naive model (always majority class)
accuracy,sensitivity,specificity,class_vote,AROC =  bootstrap_evaluation(X_test,clf,y_test,True,class_vote,False,AROC)
classifier = 'Naive_majority'
classifier_results.update({classifier:{"test_acc": ['test_acc']}})
classifier_results[classifier]["test_acc"] = accuracy
classifier_results[classifier]["test_sens"] =  sensitivity
classifier_results[classifier]["test_spec"] =  specificity

for classifier in classifiers:
    clf = classifier_results[classifier]['model_params'][0]
    # Remove zero coefficents
    if hasattr(clf, 'coef_'):
        classifier_results[classifier]["features"]  = list(X_test_delim.iloc[:,np.where(clf.coef_ > 0)[1]])
```

Figure 105: Evaluating a naive majority classifier

The performance of the models over the 100 bootstrapped resamples were then visualised using raincloud plots (Figure 106)

```
#Plot Accuracy Resutls as rain cloudds
classifiers.update({"Stacked Ensemble": []})
classifiers.update({"Naive_majority": []})
classifiers_names = list(['Naive Bayes', 'Elastic Net','Random Forest', 'SVM Bag',
                          'AdaBoost',"Stacked Ensemble",'Naive Majority'])
ACC_results = pd.DataFrame()
for i, classifier in enumerate(classifiers):
    df2 =  pd.DataFrame(classifier_results[classifier]["test_acc"], columns = [classifiers_names[i]])
    ACC_results = pd.concat([ACC_results,df2], axis=1)
ACC_results = pd.melt(ACC_results)
pal = sns.color_palette(n_colors=1)
dx = "variable"; dy = "value"; ort = "v"; pal = "Set2"; sigma = .2
fig, ax = plt.subplots(figsize=(7, 5))
pt.RainCloud(x = dx, y = dy, data = ACC_results, palette = pal, bw = sigma,
                  width_box = 0.3, width_viol = .6, ax = ax, orient = ort)
ax.set_xlabel('Model',fontsize=14);
ax.set_ylabel('Bootstrapped Accuracy',fontsize=14);
plt.xticks(rotation=45)
fig.savefig('Foot_Boot_Acc.png',dpi=300, transparent=False, bbox_inches='tight')
ACC_results.to_csv('ACC_results_foot_boot.csv')

#Sensitivity
classifiers.pop('Naive_majority', None) #Remove Naive majority model
classifiers_names = list(['Naive Bayes', 'Elastic Net','Random Forest', 'SVM Bag',
                          'AdaBoost',"Stacked Ensemble"])
SENS_results = pd.DataFrame()
for i, classifier in enumerate(classifiers):
    df2 =  pd.DataFrame(classifier_results[classifier]["test_sens"], columns = [classifiers_names[i]])
    SENS_results = pd.concat([SENS_results,df2], axis=1)
SENS_results = pd.melt(SENS_results)
pal = sns.color_palette(n_colors=1)
dx = "variable"; dy = "value"; ort = "v"; pal = "Set2"; sigma = .2
fig, ax = plt.subplots(figsize=(7, 5))
pt.RainCloud(x = dx, y = dy, data = SENS_results, palette = pal, bw = sigma,
                  width_box = 0.3,width_viol = .6, ax = ax, orient = ort)
ax.set_xlabel('Model',fontsize=14);
ax.set_ylabel('Bootstrapped Sensitivity',fontsize=14);
plt.xticks(rotation=45)
fig.savefig('Foot_Boot_Sens.png',dpi=300, transparent=False, bbox_inches='tight')
SENS_results.to_csv('SENS_results_foot_boot.csv')


#Specificity
SPEC_results = pd.DataFrame()
for i, classifier in enumerate(classifiers):
    df2 =  pd.DataFrame(classifier_results[classifier]["test_spec"], columns = [classifiers_names[i]])
    SPEC_results = pd.concat([SPEC_results,df2], axis=1)
SPEC_results = pd.melt(SPEC_results)
pal = sns.color_palette(n_colors=1)
dx = "variable"; dy = "value"; ort = "v"; pal = "Set2"; sigma = .2
fig, ax = plt.subplots(figsize=(7, 5))
pt.RainCloud(x = dx, y = dy, data = SPEC_results, palette = pal, bw = sigma,
                 width_box = 0.3, width_viol = 0.6, ax = ax, orient = ort)
ax.set_xlabel('Model',fontsize=14);
ax.set_ylabel('Bootstrapped Specificity',fontsize=14);
plt.xticks(rotation=45)
fig.savefig('Foot_Boot_Spec.png',dpi=300, transparent=False, bbox_inches='tight')
SPEC_results.to_csv('SPEC_results_foot_boot.csv')
```

Figure 106: Visualisation of the bootstrapped evaluation metrics.

The results were then statistically tested using Welch's one-way ANOVA followed by Games Howell post hoc tests. For the accuracy results, a series of one sample welch t-tests with holm's correction were used to compare against a value of 50, representing random classification (Figure 107).

```
#Test Homogenity of Variance
pg.homoscedasticity(dv='value', group ='variable', data=ACC_results)
pg.homoscedasticity(dv='value', group ='variable', data=SPEC_results)
pg.homoscedasticity(dv='value', group ='variable', data=SENS_results)

#Accuracy Welch ANOVA
table = pg.welch_anova(dv='value', between ='variable', data=ACC_results)
print(table)
#Posthoc tests
ph = pg.pairwise_gameshowell(dv='value', between='variable', data=ACC_results, effsize='cohen')
ph.to_csv('acc_posthoc_foot.csv', index=False)
table.to_csv('acc_anova_foot.csv')

#Accuracy one sample Welch t-test
acc_index = np.tile(np.transpose(np.array(range(100))),6)
ACC_results = pd.concat([ACC_results,pd.DataFrame(acc_index)], axis = 1)
ACC_unmelted = ACC_results.pivot_table( index = 0, columns='variable', values = 'value')
owt = pd.DataFrame()
for i in range(len(ACC_unmelted.T)):
    owt = owt.append(pg.ttest(ACC_unmelted.iloc[:,i],0.5, tail = 'greater' ,correction='auto').round(2))
pg.multicomp(np.array(owt[['p-val']]),method = 'holm')

#Specificity Welch Anova and post hoc
table = pg.welch_anova(dv='value', between ='variable', data=SPEC_results)
print(table)
ph = pg.pairwise_gameshowell(dv='value', between='variable', data=SPEC_results, effsize='cohen')
ph.to_csv('spec_posthoc_foot.csv', index=False)
table.to_csv('spec_anova_foot.csv')

#Sensitivity Welch Anova and post hoc
table = pg.welch_anova(dv='value', between ='variable', data=SENS_results)
print(table)
ph = pg.pairwise_gameshowell(dv='value', between='variable', data=SENS_results, effsize='cohen')
ph.to_csv('sens_posthoc_foot.csv', index=False)
table.to_csv('sens_anova_foot.csv')
```

Figure 107: Statistical testing of the bootstrapped evaluation metrics

Finally, feature importance was determine using permutation tests and partial dependency analysis (Figure 108).

```
#Determine feature importance with  permutation of feature inputs
X_test_delim = X_test[[*classifier_results['RandomForest']["features"][0]]]
clf = classifier_results['RandomForest']['model_params'][0]
forest_perm = permutation_importance(clf, X_test_delim, y_test, n_repeats=10,
                                     random_state=42, n_jobs=6)
sorted_idx = forest_perm.importances_mean.argsort()
new_features_tot =  ['Knee flexion Velocity (1-7%)',
 'Knee rotation velocity (90-100%)',
 'Thorax frontal plane angle (90-10%)',
 'Thorax ipsilateral tilt velocity (90-100%)',
 'Pelvis sagittal plane tilt (4 -24%)',
 'Hip extension acceleration (66-82%)',
 'Ankle plantar flexion velocity (61-71%)',
 'Ankle frontal plane ROM',
 'Hip sagittal plane acceleration (mean)',
 'Knee transverse plane acceleration (mean)']
fig, ax = plt.subplots()
ax.boxplot(forest_perm.importances[sorted_idx].T,
           vert=False, labels=np.array(new_features_tot)[sorted_idx])
ax.set_title("Permutation Importances")
fig.tight_layout()
fig.savefig('Foot_RF_Perm_import.png',dpi=300, transparent=False, bbox_inches='tight')

#Partial dependency plots
X_test_delim.columns = new_features_tot
new_features = ['Knee flexion Velocity (1-7%)', 'Thorax ipsilateral tilt velocity (90-100%)',
  'Pelvis sagittal plane tilt (4-24%)','Hip extension acceleration (66-82%)']
pdp = plot_partial_dependence(clf, X_test_delim, X_test_delim.columns[sorted_idx[-4:]],
                        n_jobs=6, n_cols = 2, grid_resolution=20)
pdp.axes_[0,0].xlabel = new_features[0]
pdp.axes_[0,1].xlabel = new_features[1]
pdp.axes_[1,0].xlabel = new_features[2]
pdp.axes_[1,1].xlabel = new_features[3]
fig = plt.gcf()
#fig.suptitle('Partial dependence plots')
fig.subplots_adjust(wspace=0.6, hspace=0.4)
fig.tight_layout()
fig.savefig('foot_partical_depnency_plot.png',dpi=300, transparent=False, bbox_inches='tight')
```

Figure 108: Investigating feature importance for the random forest model

6 Additional Material for the Technical Report

6.1 Additional Related Work

6.1.1 Search criteria for foot-strike and injury

Sports Discus and Web of Science databases were searched to identify studies investigating foot strike and running injuries from January 1960 to January 2020. The search was restricted to studies that were in the English language and conducted with human subjects. To avoid including potentially confounding factors, studies which included cohorts from other sports were excluded. Finally reviews, commentaries, opinion articles, case studies and conference proceedings were excluded from the primary review. The following search terms were utilised: ‘’running” OR “runners” AND “injury” OR “injuries” AND “rearfoot” OR “rear-foot” OR “midfoot” OR “mid-foot” OR “forefoot” OR “fore-foot” OR “foot contact angle” OR “foot angle” OR “foot strike pattern” OR “foot strike angle” OR “strike index”.

6.1.2 Prospective risk factors for running related injury

Within the literature there has been considerable interest in the biomechanical risk factors for running related injuries (Pohl et al.; 2008; Taunton et al.; 2002). While the majority of research to date has been retrospective in nature, this form of research is limited as it is unclear if any biomechanical factors identified are causative in nature or a result of the injury itself (Bahr; 2016). A more robust research design is prospective in nature, where uninjured participants are tested and the biomechanical factors that were associated with them becoming injured are assessed.

A total of 16 prospective cohort studies were identified in a recent systematic review of risk factors for running injury (Ceyssens et al.; 2019). Overall, the risk factors for running related injury appear to be inconsistent and may be related to the heterogeneity in study populations and the injuries being studied. When synthesising the findings of the research, it appears that the majority of risk factors are kinetic in nature (loading related) (Stefanyshyn et al.; 2006; Dudley et al.; 2017; Brund et al.; 2017; Van Ginckel et al.; 2009; Thijs et al.; 2008; Napier et al.; 2018; Bredeweg, Buist and Kluitenberg; 2013; Davis et al.; 2016; Bredeweg, Kluitenberg, Bessem and Buist; 2013). This would make sense since injuries are caused by relative excessive loading. Of the kinematic (movement) features identified, the foot was most commonly identified as a risk factor for injury (Dudley et al.; 2017; Kuhman et al.; 2016; Hein et al.; 2014). Interestingly, despite this, conflicting evidence was observed in ankle eversion velocity (Dudley et al.; 2017; Kuhman et al.; 2016) with inconsistent evidence that peak ankle, rearfoot eversion and ankle eversion range of motion were related increased risk of running related injury (Dudley et al.; 2017; Kuhman et al.; 2016; Noehren et al.; 2007; Messier et al.; 2018). Similarly, limited evidence was presented for smaller ankle dorsiflexion in runners who go on to develop Achilles tendinopathy (Hein et al.; 2014). Again, conflicting evidence exists for the knee with one study identifying a smaller peak knee flexion angle as a risk factor for injury (Hein et al.; 2014) while others reported no significant difference (Messier et al.; 2018). At the hip, limited evidence was identified for peak hip adduction in female recreational runners as a risk factor for injury (Noehren et al.; 2007, 2013) Interestingly, the one study that found opposing evidence was in a mixed sex population (Dudley et al.;

2017) suggesting that there may be sex related risk factors for certain injuries. Of particular note is the fact that none of the research considered the trunk as a risk factor for injury. This is surprising given that the thorax (including the arms and head) accounts for up to 68% of the body mass (Winter; 2009) and can have considerable influence on the loading experienced by the lower limbs (Blackburn and Padua; 2008).

Interestingly, when exploring the methodology of the 16 studies included in the systematic review, several methodological weaknesses were identified which were not highlighted by the author of the review (Ceyssens et al.; 2019). Firstly, six of the studies investigated univariate risk factors for running injuries with no control for multiple comparisons which can lead to inflation of type 1 errors and ignores multivariate relationships (Noehren et al.; 2007, 2013; Stefanyshyn et al.; 2006; Kuhman et al.; 2016; Hein et al.; 2014; Dudley et al.; 2017). Of the remaining 10 studies, which used logistic regression (Luedke et al.; 2016; Ghani Zadeh Hesar et al.; 2009; Davis et al.; 2016; Messier et al.; 2018; Van Ginckel et al.; 2009; Thijs et al.; 2008), linear regression (Brund et al.; 2017), and Cox proportional hazard models (Napier et al.; 2018; Bredeweg, Buist and Kluitenberg; 2013; Bredeweg, Kluitenberg, Bessem and Buist; 2013), 5 studies (Messier et al.; 2018; Van Ginckel et al.; 2009; Thijs et al.; 2008; Bredeweg, Buist and Kluitenberg; 2013; Bredeweg, Kluitenberg, Bessem and Buist; 2013) exclusively utilised univariate feature selection, which risks excluding potentially important features that can act as covariates in the final model. In addition, all 10 studies that conducted some form of multivariate modelling, failed to use any form of out of sample testing and only explored a single model. This can lead to poor generalisability of the studies' findings, and as per the no free lunch theorem (Wolpert; 1996), runs the risk of utilising a non-optimal model for the data being examined. Finally, all sixteen studies explored in the systematic review considered discrete biomechanical features which can lead to discarding potentially important features contained in the whole waveform (Pataky; 2012).

In summary, several prospective risk factors have been identified in the literature, but with inconsistent findings. While the cause of this inconsistency is unclear, it may be related to some of the statistical limitations identified in this review.

6.2 Methodology

6.2.1 Biomechanical Waveforms

All biomechanical waveform plots extracted from the motion capture data are presented below. The plots present the mean ± standard deviations for the nomalised stance phase. The robust key phases identified using the concept of 'Analysis of Characterising Phases' are represented by the grey shaded regions (Figure 109 - Figure 114).

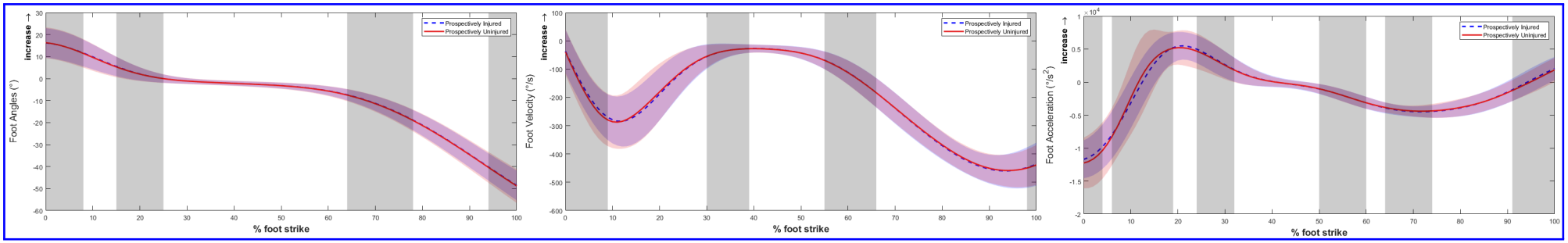


Figure 109: Biomechanical waveforms for the foot

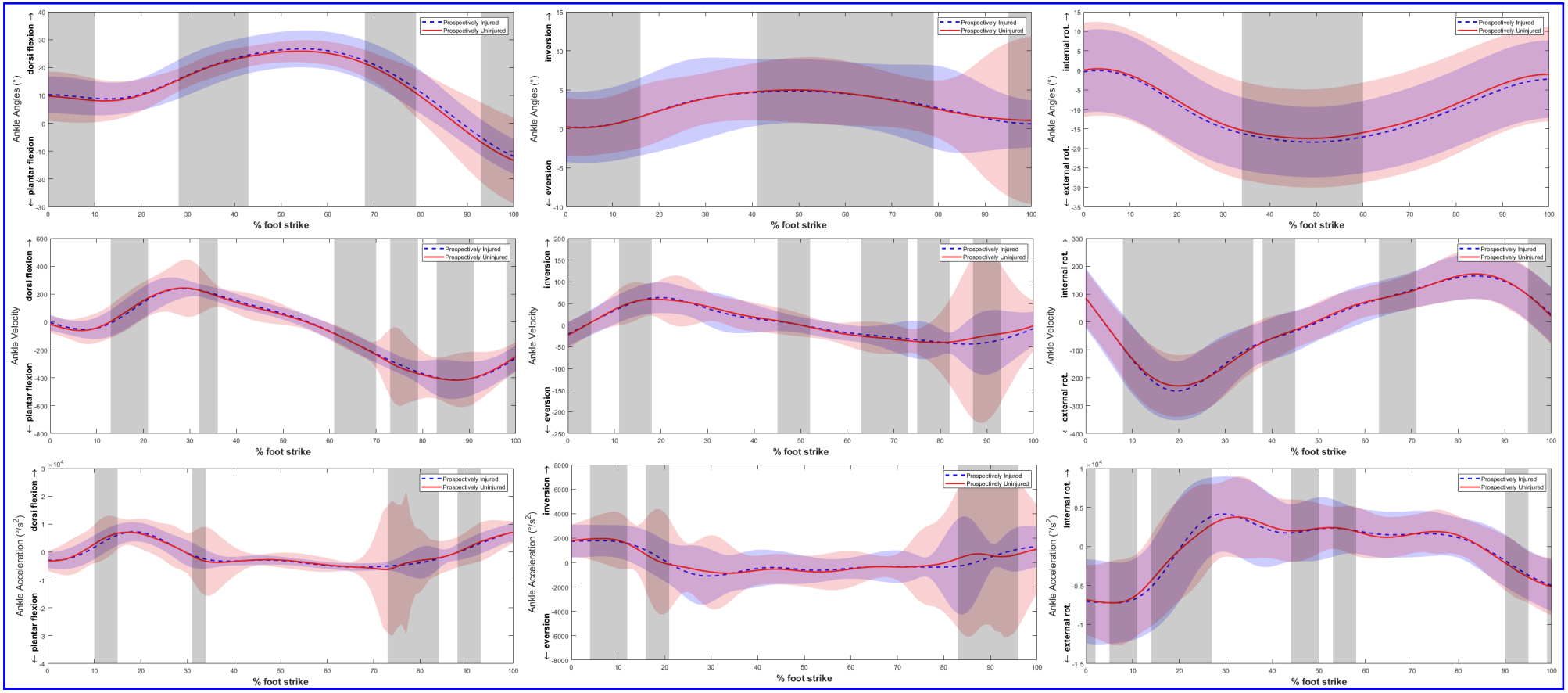


Figure 110: Biomechanical waveforms for the ankle

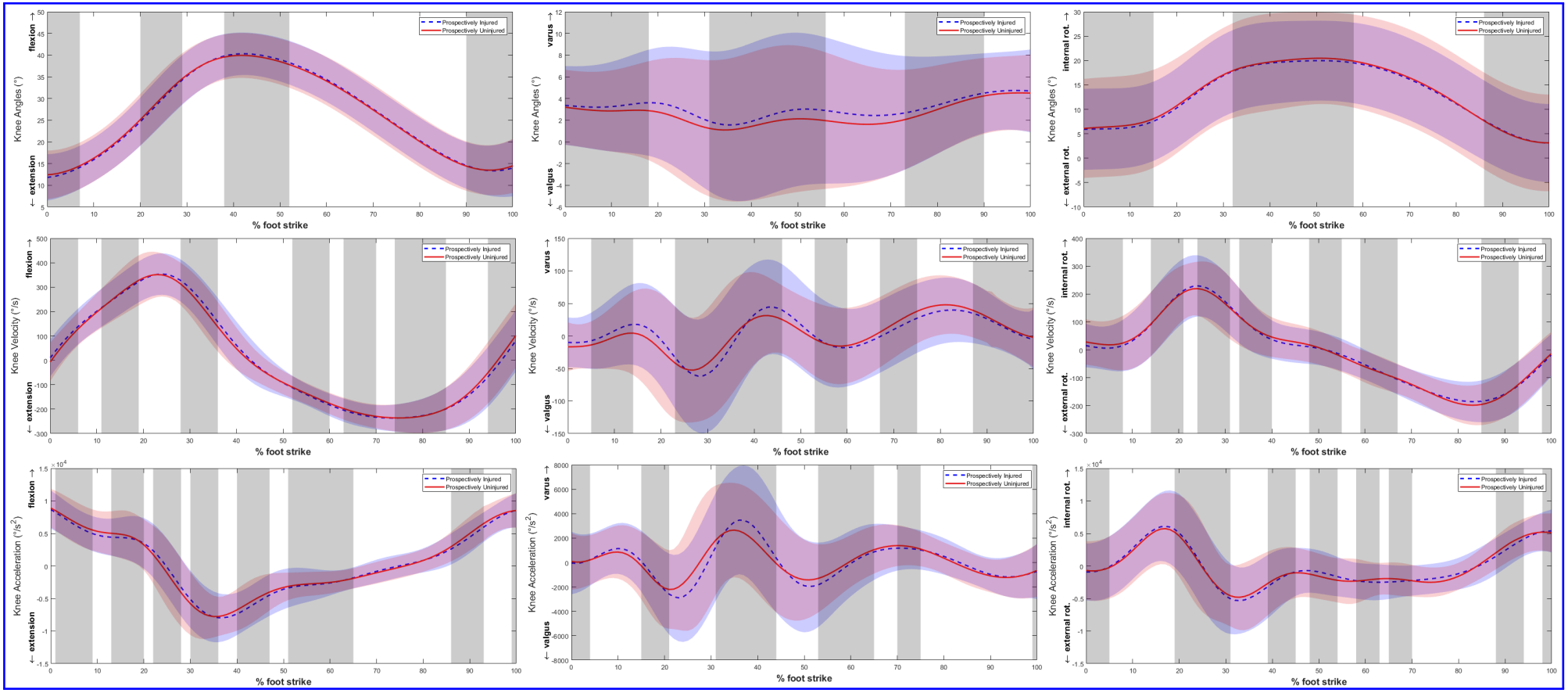


Figure 111: Biomechanical waveforms for the knee

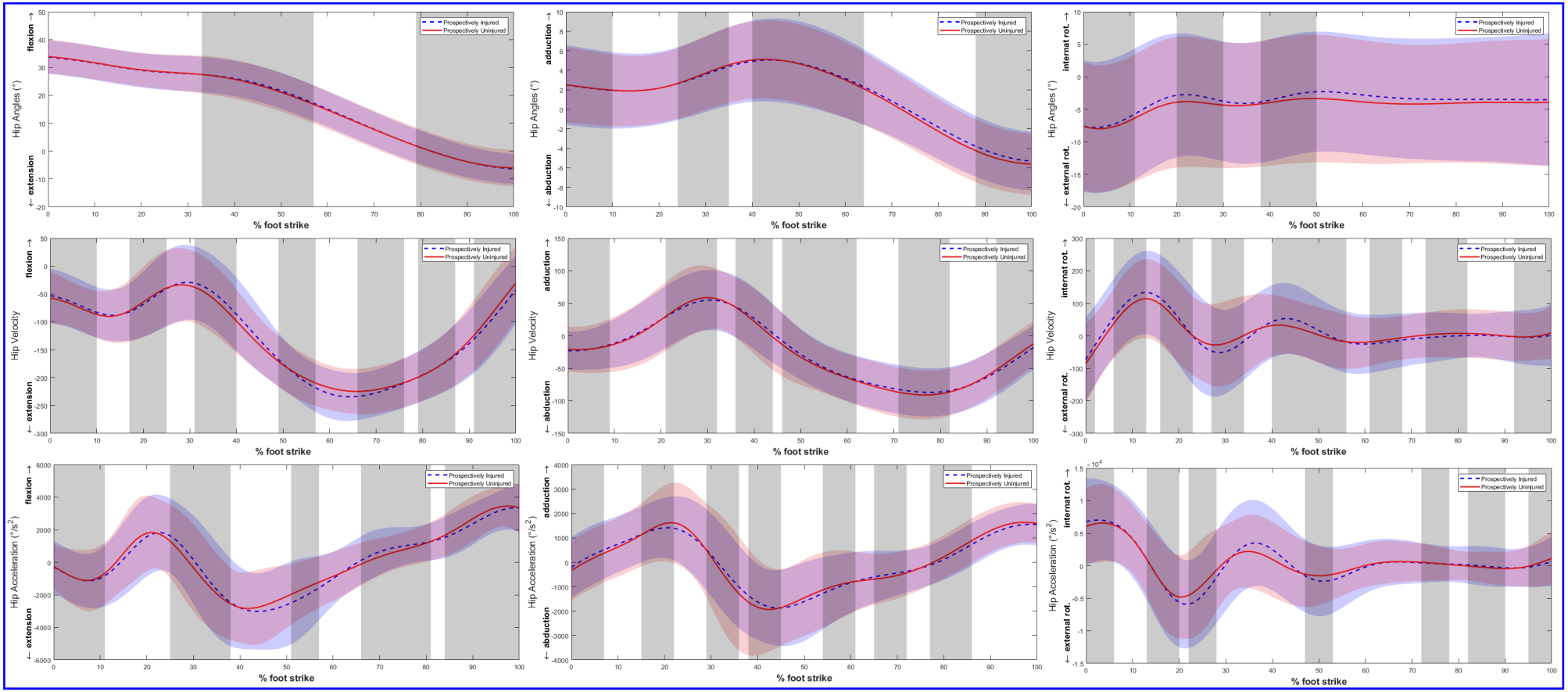


Figure 112: Biomechanical waveforms for the hip

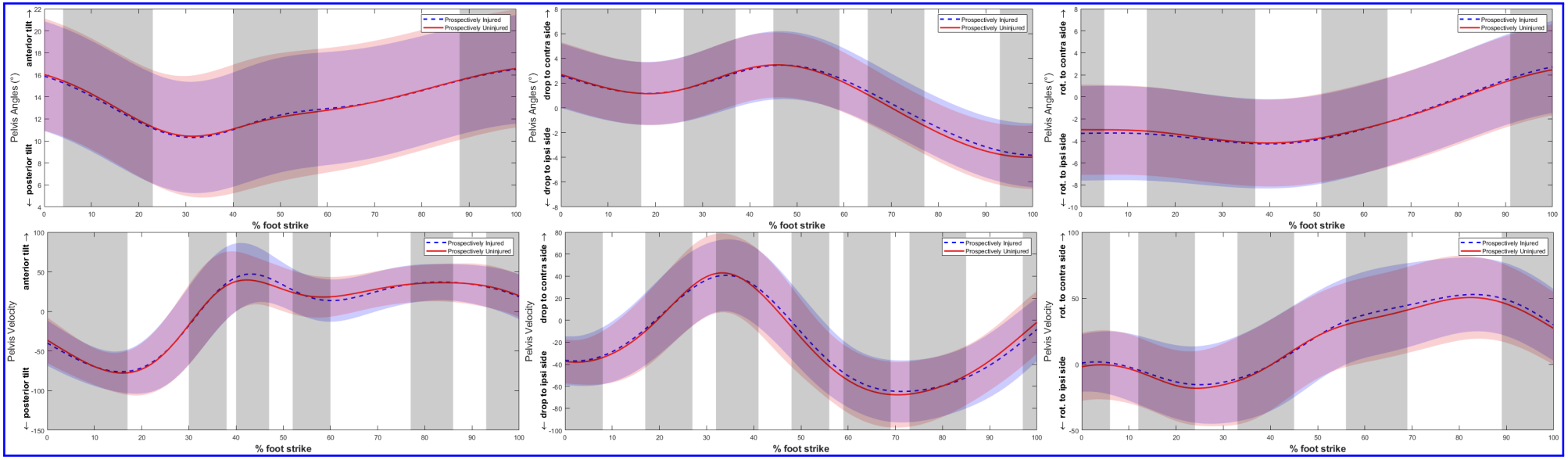


Figure 113: Biomechanical waveforms for the pelvis

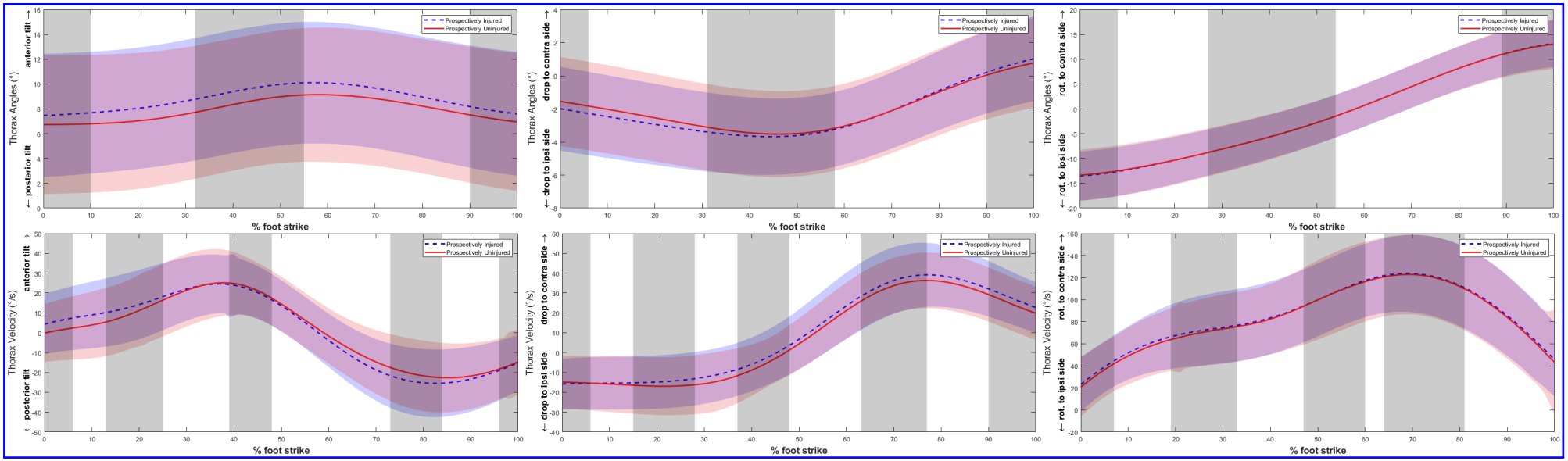


Figure 114: Biomechanical waveforms for the thorax

6.2.2 Participant Demographics

Basic subject demogrpahics of the participants who partook in this project are presented in the table below (Table 2).

Table 2: Subject Demographics

Sex	N	Age (yrs)	Height (cm)	Mass (kg)
Female	113	42.4 ± 8.6	164.5 ± 7.6	61.5 ± 8.2
Male	169	44.7 ± 9.4	177.9 ± 6.6	79.4 ± 10.6

Subject demographics presented as mean ± standard deviation.
N = number of participants, yrs. = years, cm = centimetre, kg = kilogram

6.3 Implementation

6.3.1 Data description

The following table provides a list and description of the features that were considered in this project. It is worth noting that for the clustering implementation, the data was delimited to the foot movement features.

Table 3: Feature list and description

Feature Name	Description
'AnkleAcceleration_abd__maximum'	Maximum Ankle acceleration in the frontal plane for the whole stance phase
'AnkleAcceleration_abd__mean'	Mean Ankle acceleration in the frontal plane for the whole stance phase
'AnkleAcceleration_abd__median'	Median Ankle acceleration in the frontal plane for the whole stance phase
'AnkleAcceleration_abd__minimum'	Minimum Ankle acceleration in the frontal plane for the whole stance phase
'AnkleAcceleration_abd__standard_deviation'	Ankle acceleration standard deviation in the frontal plane for the whole stance phase
'AnkleAcceleration_abd_16_22'	Mean Ankle acceleration in the frontal plane over 16-21% of the stance phase
'AnkleAcceleration_abd_4_13'	Mean Ankle acceleration in the frontal plane over 4-12% of the stance phase
'AnkleAcceleration_abd_83_97'	Mean Ankle acceleration in the frontal plane over 83-96% of the stance phase
'AnkleAcceleration_fle__maximum'	Maximum Ankle acceleration in the transverse plane for the whole stance phase
'AnkleAcceleration_fle__mean'	Mean Ankle acceleration in the transverse plane for the whole stance phase
'AnkleAcceleration_fle__median'	Median Ankle acceleration in the transverse plane for the whole stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'AnkleAcceleration_fle__standard_deviation'	Ankle acceleration standard deviation in the transverse plane for the whole stance phase
'AnkleAcceleration_fle_10_16'	Mean Ankle acceleration in the transverse plane over 10-15% of the stance phase
'AnkleAcceleration_fle_31_35'	Mean Ankle acceleration in the transverse plane over 31-34% of the stance phase
'AnkleAcceleration_fle_73_85'	Mean Ankle acceleration in the transverse plane over 73-84% of the stance phase
'AnkleAcceleration_fle_88_94'	Mean Ankle acceleration in the transverse plane over 88-93% of the stance phase
'AnkleAcceleration_rot__maximum'	Maximum Ankle acceleration in the transverse plane for the whole stance phase
'AnkleAcceleration_rot__minimum'	Minimum Ankle acceleration in the transverse plane for the whole stance phase
'AnkleAcceleration_rot__standard_deviation'	Ankle acceleration standard deviation in the transverse plane for the whole stance phase
'AnkleAcceleration_rot_0_3'	Mean Ankle acceleration in the transverse plane over 0-2% of the stance phase
'AnkleAcceleration_rot_14_28'	Mean Ankle acceleration in the transverse plane over 14-27% of the stance phase
'AnkleAcceleration_rot_44_51'	Mean Ankle acceleration in the transverse plane over 44-50% of the stance phase
'AnkleAcceleration_rot_5_12'	Mean Ankle acceleration in the transverse plane over 5-11% of the stance phase
'AnkleAcceleration_rot_53_59'	Mean Ankle acceleration in the transverse plane over 53-58% of the stance phase
'AnkleAcceleration_rot_79_80'	Mean Ankle acceleration in the transverse plane over 79-79% of the stance phase
'AnkleAcceleration_rot_90_96'	Mean Ankle acceleration in the transverse plane over 90-95% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'AnkleAcceleration_rot_99_101'	Mean Ankle acceleration in the transverse plane over 99-100% of the stance phase
'AnkleAngles_abd__minimum'	Minimum Ankle angle in the frontal plane for the whole stance phase
'AnkleAngles_abd__standard_deviation'	Ankle angle standard deviation in the frontal plane for the whole stance phase
'AnkleAngles_abd_0_17'	Mean Ankle angle in the frontal plane over 0-16% of the stance phase
'AnkleAngles_abd_41_80'	Mean Ankle angle in the frontal plane over 41-79% of the stance phase
'AnkleAngles_abd_95_101'	Mean Ankle angle in the frontal plane over 95-100% of the stance phase
'AnkleAngles_fle__maximum'	Maximum Ankle angle in the transverse plane for the whole stance phase
'AnkleAngles_fle__mean'	Mean Ankle angle in the transverse plane for the whole stance phase
'AnkleAngles_fle__standard_deviation'	Ankle angle standard deviation in the transverse plane for the whole stance phase
'AnkleAngles_fle_0_11'	Mean Ankle angle in the transverse plane over 0-10% of the stance phase
'AnkleAngles_fle_28_44'	Mean Ankle angle in the transverse plane over 28-43% of the stance phase
'AnkleAngles_fle_68_80'	Mean Ankle angle in the transverse plane over 68-79% of the stance phase
'AnkleAngles_fle_93_101'	Mean Ankle angle in the transverse plane over 93-100% of the stance phase
'AnkleAngles_rot__standard_deviation'	Ankle angle standard deviation in the transverse plane for the whole stance phase
'AnkleAngles_rot_34_61'	Mean Ankle angle in the transverse plane over 34-60% of the stance phase
'AnkleVelocity_abd__maximum'	Maximum Ankle velocity in the frontal plane for the whole stance phase
'AnkleVelocity_abd__mean'	Mean Ankle velocity in the frontal plane for the whole stance phase
'AnkleVelocity_abd__median'	Median Ankle velocity in the frontal plane for the whole stance phase
'AnkleVelocity_abd__minimum'	Minimum Ankle velocity in the frontal plane for the whole stance phase
'AnkleVelocity_abd__standard_deviation'	Ankle velocity standard deviation in the frontal plane for the whole stance phase
'AnkleVelocity_abd_0_6'	Mean Ankle velocity in the frontal plane over 0-5% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'AnkleVelocity_abd_11_19'	Mean Ankle velocity in the frontal plane over 11-18% of the stance phase
'AnkleVelocity_abd_45_53'	Mean Ankle velocity in the frontal plane over 45-52% of the stance phase
'AnkleVelocity_abd_63_74'	Mean Ankle velocity in the frontal plane over 63-73% of the stance phase
'AnkleVelocity_abd_75_83'	Mean Ankle velocity in the frontal plane over 75-82% of the stance phase
'AnkleVelocity_abd_87_94'	Mean Ankle velocity in the frontal plane over 87-93% of the stance phase
'AnkleVelocity_fle__maximum'	Maximum Ankle velocity in the transverse plane for the whole stance phase
'AnkleVelocity_fle__mean'	Mean Ankle velocity in the transverse plane for the whole stance phase
'AnkleVelocity_fle__median'	Median Ankle velocity in the transverse plane for the whole stance phase
'AnkleVelocity_fle__standard_deviation'	Ankle velocity standard deviation in the transverse plane for the whole stance phase
'AnkleVelocity_fle_13_22'	Mean Ankle velocity in the transverse plane over 13-21% of the stance phase
'AnkleVelocity_fle_32_37'	Mean Ankle velocity in the transverse plane over 32-36% of the stance phase
'AnkleVelocity_fle_61_71'	Mean Ankle velocity in the transverse plane over 61-70% of the stance phase
'AnkleVelocity_fle_73_80'	Mean Ankle velocity in the transverse plane over 73-79% of the stance phase
'AnkleVelocity_fle_83_92'	Mean Ankle velocity in the transverse plane over 83-91% of the stance phase
'AnkleVelocity_fle_98_101'	Mean Ankle velocity in the transverse plane over 98-100% of the stance phase
'AnkleVelocity_rot__maximum'	Maximum Ankle velocity in the transverse plane for the whole stance phase
'AnkleVelocity_rot__minimum'	Minimum Ankle velocity in the transverse plane for the whole stance phase
'AnkleVelocity_rot__standard_deviation'	Ankle velocity standard deviation in the transverse plane for the whole stance phase
'AnkleVelocity_rot_38_46'	Mean Ankle velocity in the transverse plane over 38-45% of the stance phase
'AnkleVelocity_rot_63_72'	Mean Ankle velocity in the transverse plane over 63-71% of the stance phase
'AnkleVelocity_rot_8_37'	Mean Ankle velocity in the transverse plane over 8-36% of the stance phase
'AnkleVelocity_rot_95_101'	Mean Ankle velocity in the transverse plane over 95-100% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'FootAcceleration_fle__maximum'	Maximum Foot acceleration in the transverse plane for the whole stance phase
'FootAcceleration_fle__mean'	Mean Foot acceleration in the transverse plane for the whole stance phase
'FootAcceleration_fle__median'	Median Foot acceleration in the transverse plane for the whole stance phase
'FootAcceleration_fle__standard_deviation'	Foot acceleration standard deviation in the transverse plane for the whole stance phase
'FootAcceleration_fle_0_5'	Mean Foot acceleration in the transverse plane over 0-4% of the stance phase
'FootAcceleration_fle_24_33'	Mean Foot acceleration in the transverse plane over 24-32% of the stance phase
'FootAcceleration_fle_50_61'	Mean Foot acceleration in the transverse plane over 50-60% of the stance phase
'FootAcceleration_fle_6_20'	Mean Foot acceleration in the transverse plane over 6-19% of the stance phase
'FootAcceleration_fle_64_75'	Mean Foot acceleration in the transverse plane over 64-74% of the stance phase
'FootAcceleration_fle_91_101'	Mean Foot acceleration in the transverse plane over 91-100% of the stance phase
'FootAngles_fle__mean'	Mean Foot angle in the transverse plane for the whole stance phase
'FootAngles_fle__median'	Median Foot angle in the transverse plane for the whole stance phase
'FootAngles_fle__standard_deviation'	Foot angle standard deviation in the transverse plane for the whole stance phase
'FootAngles_fle_0_9'	Mean Foot angle in the transverse plane over 0-8% of the stance phase
'FootAngles_fle_15_26'	Mean Foot angle in the transverse plane over 15-25% of the stance phase
'FootAngles_fle_64_79'	Mean Foot angle in the transverse plane over 64-78% of the stance phase
'FootAngles_fle_94_101'	Mean Foot angle in the transverse plane over 94-100% of the stance phase
'FootVelocity_fle__maximum'	Maximum Foot velocity in the transverse plane for the whole stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'FootVelocity_fle__mean'	Mean Foot velocity in the transverse plane for the whole stance phase
'FootVelocity_fle__median'	Median Foot velocity in the transverse plane for the whole stance phase
'FootVelocity_fle__minimum'	Minimum Foot velocity in the transverse plane for the whole stance phase
'FootVelocity_fle__standard_deviation'	Foot velocity standard deviation in the transverse plane for the whole stance phase
'FootVelocity_fle_0_10'	Mean Foot velocity in the transverse plane over 0-9% of the stance phase
'FootVelocity_fle_30_40'	Mean Foot velocity in the transverse plane over 30-39% of the stance phase
'FootVelocity_fle_55_67'	Mean Foot velocity in the transverse plane over 55-66% of the stance phase
'FootVelocity_fle_98_101'	Mean Foot velocity in the transverse plane over 98-100% of the stance phase
'HipAcceleration_abd__maximum'	Maximum Hip acceleration in the frontal plane for the whole stance phase
'HipAcceleration_abd__mean'	Mean Hip acceleration in the frontal plane for the whole stance phase
'HipAcceleration_abd__median'	Median Hip acceleration in the frontal plane for the whole stance phase
'HipAcceleration_abd__minimum'	Minimum Hip acceleration in the frontal plane for the whole stance phase
'HipAcceleration_abd__standard_deviation'	Hip acceleration standard deviation in the frontal plane for the whole stance phase
'HipAcceleration_abd_0_8'	Mean Hip acceleration in the frontal plane over 0-7% of the stance phase
'HipAcceleration_abd_15_23'	Mean Hip acceleration in the frontal plane over 15-22% of the stance phase
'HipAcceleration_abd_29_37'	Mean Hip acceleration in the frontal plane over 29-36% of the stance phase
'HipAcceleration_abd_38_46'	Mean Hip acceleration in the frontal plane over 38-45% of the stance phase
'HipAcceleration_abd_54_62'	Mean Hip acceleration in the frontal plane over 54-61% of the stance phase
'HipAcceleration_abd_65_73'	Mean Hip acceleration in the frontal plane over 65-72% of the stance phase
'HipAcceleration_abd_77_87'	Mean Hip acceleration in the frontal plane over 77-86% of the stance phase
'HipAcceleration_fle__maximum'	Maximum Hip acceleration in the transverse plane for the whole stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'HipAcceleration_fle__mean'	Mean Hip acceleration in the transverse plane for the whole stance phase
'HipAcceleration_fle__median'	Median Hip acceleration in the transverse plane for the whole stance phase
'HipAcceleration_fle__minimum'	Minimum Hip acceleration in the transverse plane for the whole stance phase
'HipAcceleration_fle__standard_deviation'	Hip acceleration standard deviation in the transverse plane for the whole stance phase
'HipAcceleration_fle_0_12'	Mean Hip acceleration in the transverse plane over 0-11% of the stance phase
'HipAcceleration_fle_25_39'	Mean Hip acceleration in the transverse plane over 25-38% of the stance phase
'HipAcceleration_fle_51_58'	Mean Hip acceleration in the transverse plane over 51-57% of the stance phase
'HipAcceleration_fle_66_82'	Mean Hip acceleration in the transverse plane over 66-81% of the stance phase
'HipAcceleration_fle_84_101'	Mean Hip acceleration in the transverse plane over 84-100% of the stance phase
'HipAcceleration_rot__maximum'	Maximum Hip acceleration in the transverse plane for the whole stance phase
'HipAcceleration_rot__mean'	Mean Hip acceleration in the transverse plane for the whole stance phase
'HipAcceleration_rot__median'	Median Hip acceleration in the transverse plane for the whole stance phase
'HipAcceleration_rot__minimum'	Minimum Hip acceleration in the transverse plane for the whole stance phase
'HipAcceleration_rot_0_7'	Mean Hip acceleration in the transverse plane over 0-6% of the stance phase
'HipAcceleration_rot_13_21'	Mean Hip acceleration in the transverse plane over 13-20% of the stance phase
'HipAcceleration_rot_22_29'	Mean Hip acceleration in the transverse plane over 22-28% of the stance phase
'HipAcceleration_rot_47_54'	Mean Hip acceleration in the transverse plane over 47-53% of the stance phase
'HipAcceleration_rot_72_79'	Mean Hip acceleration in the transverse plane over 72-78% of the stance phase
'HipAcceleration_rot_82_91'	Mean Hip acceleration in the transverse plane over 82-90% of the stance phase
'HipAcceleration_rot_93_94'	Mean Hip acceleration in the transverse plane over 93-93% of the stance phase
'HipAcceleration_rot_95_101'	Mean Hip acceleration in the transverse plane over 95-100% of the stance phase
'HipAngles_abd__mean'	Mean Hip angle in the frontal plane for the whole stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'HipAngles_abd__standard_deviation'	Hip angle standard deviation in the frontal plane for the whole stance phase
'HipAngles_abd_0_11'	Mean Hip angle in the frontal plane over 0-10% of the stance phase
'HipAngles_abd_24_36'	Mean Hip angle in the frontal plane over 24-35% of the stance phase
'HipAngles_abd_40_65'	Mean Hip angle in the frontal plane over 40-64% of the stance phase
'HipAngles_abd_88_101'	Mean Hip angle in the frontal plane over 88-100% of the stance phase
'HipAngles_fle__maximum'	Maximum Hip angle in the transverse plane for the whole stance phase
'HipAngles_fle__standard_deviation'	Hip angle standard deviation in the transverse plane for the whole stance phase
'HipAngles_fle_33_58'	Mean Hip angle in the transverse plane over 33-57% of the stance phase
'HipAngles_fle_79_101'	Mean Hip angle in the transverse plane over 79-100% of the stance phase
'HipAngles_rot__maximum'	Maximum Hip angle in the transverse plane for the whole stance phase
'HipAngles_rot__minimum'	Minimum Hip angle in the transverse plane for the whole stance phase
'HipAngles_rot__standard_deviation'	Hip angle standard deviation in the transverse plane for the whole stance phase
'HipAngles_rot__variance'	Mean Hip angle in the transverse plane over -% of the stance phase
'HipAngles_rot_0_12'	Mean Hip angle in the transverse plane over 0-11% of the stance phase
'HipAngles_rot_20_31'	Mean Hip angle in the transverse plane over 20-30% of the stance phase
'HipAngles_rot_38_51'	Mean Hip angle in the transverse plane over 38-50% of the stance phase
'HipVelocity_abd__maximum'	Maximum Hip velocity in the frontal plane for the whole stance phase
'HipVelocity_abd__mean'	Mean Hip velocity in the frontal plane for the whole stance phase
'HipVelocity_abd__median'	Median Hip velocity in the frontal plane for the whole stance phase
'HipVelocity_abd__minimum'	Minimum Hip velocity in the frontal plane for the whole stance phase
'HipVelocity_abd__standard_deviation'	Hip velocity standard deviation in the frontal plane for the whole stance phase
'HipVelocity_abd_0_10'	Mean Hip velocity in the frontal plane over 0-9% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'HipVelocity_abd_21_33'	Mean Hip velocity in the frontal plane over 21-32% of the stance phase
'HipVelocity_abd_36_45'	Mean Hip velocity in the frontal plane over 36-44% of the stance phase
'HipVelocity_abd_46_63'	Mean Hip velocity in the frontal plane over 46-62% of the stance phase
'HipVelocity_abd_71_83'	Mean Hip velocity in the frontal plane over 71-82% of the stance phase
'HipVelocity_abd_92_101'	Mean Hip velocity in the frontal plane over 92-100% of the stance phase
'HipVelocity_fle__maximum'	Maximum Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_fle__mean'	Mean Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_fle__median'	Median Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_fle__minimum'	Minimum Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_fle__standard_deviation'	Hip velocity standard deviation in the transverse plane for the whole stance phase
'HipVelocity_fle_0_11'	Mean Hip velocity in the transverse plane over 0-10% of the stance phase
'HipVelocity_fle_17_26'	Mean Hip velocity in the transverse plane over 17-25% of the stance phase
'HipVelocity_fle_31_41'	Mean Hip velocity in the transverse plane over 31-40% of the stance phase
'HipVelocity_fle_49_58'	Mean Hip velocity in the transverse plane over 49-57% of the stance phase
'HipVelocity_fle_66_77'	Mean Hip velocity in the transverse plane over 66-76% of the stance phase
'HipVelocity_fle_79_88'	Mean Hip velocity in the transverse plane over 79-87% of the stance phase
'HipVelocity_fle_91_101'	Mean Hip velocity in the transverse plane over 91-100% of the stance phase
'HipVelocity_rot__maximum'	Maximum Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_rot__mean'	Mean Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_rot__median'	Median Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_rot__minimum'	Minimum Hip velocity in the transverse plane for the whole stance phase
'HipVelocity_rot__standard_deviation'	Hip velocity standard deviation in the transverse plane for the whole stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'HipVelocity_rot_0_3'	Mean Hip velocity in the transverse plane over 0-2% of the stance phase
'HipVelocity_rot_16_24'	Mean Hip velocity in the transverse plane over 16-23% of the stance phase
'HipVelocity_rot_27_35'	Mean Hip velocity in the transverse plane over 27-34% of the stance phase
'HipVelocity_rot_40_57'	Mean Hip velocity in the transverse plane over 40-56% of the stance phase
'HipVelocity_rot_6_14'	Mean Hip velocity in the transverse plane over 6-13% of the stance phase
'HipVelocity_rot_61_69'	Mean Hip velocity in the transverse plane over 61-68% of the stance phase
'HipVelocity_rot_73_83'	Mean Hip velocity in the transverse plane over 73-82% of the stance phase
'HipVelocity_rot_92_101'	Mean Hip velocity in the transverse plane over 92-100% of the stance phase
'KneeAcceleration_abd__maximum'	Maximum Knee acceleration in the frontal plane for the whole stance phase
'KneeAcceleration_abd__mean'	Mean Knee acceleration in the frontal plane for the whole stance phase
'KneeAcceleration_abd__median'	Median Knee acceleration in the frontal plane for the whole stance phase
'KneeAcceleration_abd__minimum'	Minimum Knee acceleration in the frontal plane for the whole stance phase
'KneeAcceleration_abd_0_5'	Mean Knee acceleration in the frontal plane over 0-4% of the stance phase
'KneeAcceleration_abd_15_22'	Mean Knee acceleration in the frontal plane over 15-21% of the stance phase
'KneeAcceleration_abd_31_45'	Mean Knee acceleration in the frontal plane over 31-44% of the stance phase
'KneeAcceleration_abd_53_66'	Mean Knee acceleration in the frontal plane over 53-65% of the stance phase
'KneeAcceleration_abd_70_76'	Mean Knee acceleration in the frontal plane over 70-75% of the stance phase
'KneeAcceleration_abd_99_101'	Mean Knee acceleration in the frontal plane over 99-100% of the stance phase
'KneeAcceleration_fle__maximum'	Maximum Knee acceleration in the transverse plane for the whole stance phase
'KneeAcceleration_fle__mean'	Mean Knee acceleration in the transverse plane for the whole stance phase
'KneeAcceleration_fle__median'	Median Knee acceleration in the transverse plane for the whole stance phase
'KneeAcceleration_fle__minimum'	Minimum Knee acceleration in the transverse plane for the whole stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'KneeAcceleration_fle__standard_deviation'	Knee acceleration standard deviation in the transverse plane for the whole stance phase
'KneeAcceleration_fle_1_10'	Mean Knee acceleration in the transverse plane over 1-9% of the stance phase
'KneeAcceleration_fle_13_21'	Mean Knee acceleration in the transverse plane over 13-20% of the stance phase
'KneeAcceleration_fle_22_29'	Mean Knee acceleration in the transverse plane over 22-28% of the stance phase
'KneeAcceleration_fle_30_37'	Mean Knee acceleration in the transverse plane over 30-36% of the stance phase
'KneeAcceleration_fle_40_48'	Mean Knee acceleration in the transverse plane over 40-47% of the stance phase
'KneeAcceleration_fle_51_66'	Mean Knee acceleration in the transverse plane over 51-65% of the stance phase
'KneeAcceleration_fle_86_94'	Mean Knee acceleration in the transverse plane over 86-93% of the stance phase
'KneeAcceleration_fle_99_101'	Mean Knee acceleration in the transverse plane over 99-100% of the stance phase
'KneeAcceleration_rot__maximum'	Maximum Knee acceleration in the transverse plane for the whole stance phase
'KneeAcceleration_rot__mean'	Mean Knee acceleration in the transverse plane for the whole stance phase
'KneeAcceleration_rot__median'	Median Knee acceleration in the transverse plane for the whole stance phase
'KneeAcceleration_rot__minimum'	Minimum Knee acceleration in the transverse plane for the whole stance phase
'KneeAcceleration_rot_0_6'	Mean Knee acceleration in the transverse plane over 0-5% of the stance phase
'KneeAcceleration_rot_19_32'	Mean Knee acceleration in the transverse plane over 19-31% of the stance phase
'KneeAcceleration_rot_39_46'	Mean Knee acceleration in the transverse plane over 39-45% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'KneeAcceleration_rot_48_55'	Mean Knee acceleration in the transverse plane over 48-54% of the stance phase
'KneeAcceleration_rot_58_64'	Mean Knee acceleration in the transverse plane over 58-63% of the stance phase
'KneeAcceleration_rot_65_71'	Mean Knee acceleration in the transverse plane over 65-70% of the stance phase
'KneeAcceleration_rot_88_95'	Mean Knee acceleration in the transverse plane over 88-94% of the stance phase
'KneeAcceleration_rot_98_101'	Mean Knee acceleration in the transverse plane over 98-100% of the stance phase
'KneeAngles_abd__maximum'	Maximum Knee angle in the frontal plane for the whole stance phase
'KneeAngles_abd__standard_deviation'	Knee angle standard deviation in the frontal plane for the whole stance phase
'KneeAngles_abd_0_19'	Mean Knee angle in the frontal plane over 0-18% of the stance phase
'KneeAngles_abd_31_57'	Mean Knee angle in the frontal plane over 31-56% of the stance phase
'KneeAngles_abd_73_91'	Mean Knee angle in the frontal plane over 73-90% of the stance phase
'KneeAngles_fle__mean'	Mean Knee angle in the transverse plane for the whole stance phase
'KneeAngles_fle__minimum'	Minimum Knee angle in the transverse plane for the whole stance phase
'KneeAngles_fle__standard_deviation'	Knee angle standard deviation in the transverse plane for the whole stance phase
'KneeAngles_fle_0_8'	Mean Knee angle in the transverse plane over 0-7% of the stance phase
'KneeAngles_fle_20_30'	Mean Knee angle in the transverse plane over 20-29% of the stance phase
'KneeAngles_fle_38_53'	Mean Knee angle in the transverse plane over 38-52% of the stance phase
'KneeAngles_fle_90_101'	Mean Knee angle in the transverse plane over 90-100% of the stance phase
'KneeAngles_rot__standard_deviation'	Knee angle standard deviation in the transverse plane for the whole stance phase
'KneeAngles_rot_0_16'	Mean Knee angle in the transverse plane over 0-15% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'KneeAngles_rot_32_59'	Mean Knee angle in the transverse plane over 32-58% of the stance phase
'KneeAngles_rot_86_101'	Mean Knee angle in the transverse plane over 86-100% of the stance phase
'KneeVelocity_abd__maximum'	Maximum Knee velocity in the frontal plane for the whole stance phase
'KneeVelocity_abd__mean'	Mean Knee velocity in the frontal plane for the whole stance phase
'KneeVelocity_abd__median'	Median Knee velocity in the frontal plane for the whole stance phase
'KneeVelocity_abd__minimum'	Minimum Knee velocity in the frontal plane for the whole stance phase
'KneeVelocity_abd__standard_deviation'	Knee velocity standard deviation in the frontal plane for the whole stance phase
'KneeVelocity_abd_23_47'	Mean Knee velocity in the frontal plane over 23-46% of the stance phase
'KneeVelocity_abd_5_15'	Mean Knee velocity in the frontal plane over 5-14% of the stance phase
'KneeVelocity_abd_53_61'	Mean Knee velocity in the frontal plane over 53-60% of the stance phase
'KneeVelocity_abd_67_76'	Mean Knee velocity in the frontal plane over 67-75% of the stance phase
'KneeVelocity_abd_87_101'	Mean Knee velocity in the frontal plane over 87-100% of the stance phase
'KneeVelocity_fle__maximum'	Maximum Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_fle__mean'	Mean Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_fle__median'	Median Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_fle__minimum'	Minimum Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_fle__standard_deviation'	Knee velocity standard deviation in the transverse plane for the whole stance phase
'KneeVelocity_fle_0_7'	Mean Knee velocity in the transverse plane over 0-6% of the stance phase
'KneeVelocity_fle_11_20'	Mean Knee velocity in the transverse plane over 11-19% of the stance phase
'KneeVelocity_fle_28_37'	Mean Knee velocity in the transverse plane over 28-36% of the stance phase
'KneeVelocity_fle_52_61'	Mean Knee velocity in the transverse plane over 52-60% of the stance phase
'KneeVelocity_fle_63_71'	Mean Knee velocity in the transverse plane over 63-70% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'KneeVelocity_fle_74_86'	Mean Knee velocity in the transverse plane over 74-85% of the stance phase
'KneeVelocity_fle_94_101'	Mean Knee velocity in the transverse plane over 94-100% of the stance phase
'KneeVelocity_rot__maximum'	Maximum Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_rot__mean'	Mean Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_rot__median'	Median Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_rot__minimum'	Minimum Knee velocity in the transverse plane for the whole stance phase
'KneeVelocity_rot__standard_deviation'	Knee velocity standard deviation in the transverse plane for the whole stance phase
'KneeVelocity_rot_0_9'	Mean Knee velocity in the transverse plane over 0-8% of the stance phase
'KneeVelocity_rot_14_22'	Mean Knee velocity in the transverse plane over 14-21% of the stance phase
'KneeVelocity_rot_24_32'	Mean Knee velocity in the transverse plane over 24-31% of the stance phase
'KneeVelocity_rot_33_41'	Mean Knee velocity in the transverse plane over 33-40% of the stance phase
'KneeVelocity_rot_48_56'	Mean Knee velocity in the transverse plane over 48-55% of the stance phase
'KneeVelocity_rot_59_68'	Mean Knee velocity in the transverse plane over 59-67% of the stance phase
'KneeVelocity_rot_85_94'	Mean Knee velocity in the transverse plane over 85-93% of the stance phase
'KneeVelocity_rot_98_101'	Mean Knee velocity in the transverse plane over 98-100% of the stance phase
'PelvisAngles_abd__maximum'	Maximum Pelvis angle in the frontal plane for the whole stance phase
'PelvisAngles_abd__mean'	Mean Pelvis angle in the frontal plane for the whole stance phase
'PelvisAngles_abd__standard_deviation'	Pelvis angle standard deviation in the frontal plane for the whole stance phase
'PelvisAngles_abd_0_18'	Mean Pelvis angle in the frontal plane over 0-17% of the stance phase
'PelvisAngles_abd_26_38'	Mean Pelvis angle in the frontal plane over 26-37% of the stance phase
'PelvisAngles_abd_45_60'	Mean Pelvis angle in the frontal plane over 45-59% of the stance phase
'PelvisAngles_abd_65_78'	Mean Pelvis angle in the frontal plane over 65-77% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'PelvisAngles_abd_93_101'	Mean Pelvis angle in the frontal plane over 93-100% of the stance phase
'PelvisAngles_fle__standard_deviation'	Pelvis angle standard deviation in the transverse plane for the whole stance phase
'PelvisAngles_fle_4_24'	Mean Pelvis angle in the transverse plane over 4-23% of the stance phase
'PelvisAngles_fle_40_59'	Mean Pelvis angle in the transverse plane over 40-58% of the stance phase
'PelvisAngles_fle_88_101'	Mean Pelvis angle in the transverse plane over 88-100% of the stance phase
'PelvisAngles_rot__minimum'	Minimum Pelvis angle in the transverse plane for the whole stance phase
'PelvisAngles_rot__standard_deviation'	Pelvis angle standard deviation in the transverse plane for the whole stance phase
'PelvisAngles_rot_0_6'	Mean Pelvis angle in the transverse plane over 0-5% of the stance phase
'PelvisAngles_rot_14_38'	Mean Pelvis angle in the transverse plane over 14-37% of the stance phase
'PelvisAngles_rot_51_66'	Mean Pelvis angle in the transverse plane over 51-65% of the stance phase
'PelvisAngles_rot_91_101'	Mean Pelvis angle in the transverse plane over 91-100% of the stance phase
'PelvisVelocity_abd__maximum'	Maximum Pelvis velocity in the frontal plane for the whole stance phase
'PelvisVelocity_abd__mean'	Mean Pelvis velocity in the frontal plane for the whole stance phase
'PelvisVelocity_abd__median'	Median Pelvis velocity in the frontal plane for the whole stance phase
'PelvisVelocity_abd__minimum'	Minimum Pelvis velocity in the frontal plane for the whole stance phase
'PelvisVelocity_abd__standard_deviation'	Pelvis velocity standard deviation in the frontal plane for the whole stance phase
'PelvisVelocity_abd_0_9'	Mean Pelvis velocity in the frontal plane over 0-8% of the stance phase
'PelvisVelocity_abd_17_28'	Mean Pelvis velocity in the frontal plane over 17-27% of the stance phase
'PelvisVelocity_abd_32_42'	Mean Pelvis velocity in the frontal plane over 32-41% of the stance phase
'PelvisVelocity_abd_48_57'	Mean Pelvis velocity in the frontal plane over 48-56% of the stance phase
'PelvisVelocity_abd_59_70'	Mean Pelvis velocity in the frontal plane over 59-69% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'PelvisVelocity_abd_73_86'	Mean Pelvis velocity in the frontal plane over 73-85% of the stance phase
'PelvisVelocity_abd_97_101'	Mean Pelvis velocity in the frontal plane over 97-100% of the stance phase
'PelvisVelocity_fle__maximum'	Maximum Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_fle__mean'	Mean Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_fle__median'	Median Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_fle__minimum'	Minimum Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_fle__standard_deviation'	Pelvis velocity standard deviation in the transverse plane for the whole stance phase
'PelvisVelocity_fle_0_18'	Mean Pelvis velocity in the transverse plane over 0-17% of the stance phase
'PelvisVelocity_fle_30_39'	Mean Pelvis velocity in the transverse plane over 30-38% of the stance phase
'PelvisVelocity_fle_40_48'	Mean Pelvis velocity in the transverse plane over 40-47% of the stance phase
'PelvisVelocity_fle_52_61'	Mean Pelvis velocity in the transverse plane over 52-60% of the stance phase
'PelvisVelocity_fle_77_87'	Mean Pelvis velocity in the transverse plane over 77-86% of the stance phase
'PelvisVelocity_fle_93_101'	Mean Pelvis velocity in the transverse plane over 93-100% of the stance phase
'PelvisVelocity_rot__maximum'	Maximum Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_rot__mean'	Mean Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_rot__median'	Median Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_rot__minimum'	Minimum Pelvis velocity in the transverse plane for the whole stance phase
'PelvisVelocity_rot__standard_deviation'	Pelvis velocity standard deviation in the transverse plane for the whole stance phase
'PelvisVelocity_rot_0_7'	Mean Pelvis velocity in the transverse plane over 0-6% of the stance phase
'PelvisVelocity_rot_12_25'	Mean Pelvis velocity in the transverse plane over 12-24% of the stance phase
'PelvisVelocity_rot_33_46'	Mean Pelvis velocity in the transverse plane over 33-45% of the stance phase
'PelvisVelocity_rot_56_70'	Mean Pelvis velocity in the transverse plane over 56-69% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'PelvisVelocity_rot_89_101'	Mean Pelvis velocity in the transverse plane over 89-100% of the stance phase
'ThoraxAngles_abd__standard_deviation'	Thorax angle standard deviation in the frontal plane for the whole stance phase
'ThoraxAngles_abd_0_7'	Mean Thorax angle in the frontal plane over 0-6% of the stance phase
'ThoraxAngles_abd_31_59'	Mean Thorax angle in the frontal plane over 31-58% of the stance phase
'ThoraxAngles_abd_90_101'	Mean Thorax angle in the frontal plane over 90-100% of the stance phase
'ThoraxAngles_fle__standard_deviation'	Thorax angle standard deviation in the transverse plane for the whole stance phase
'ThoraxAngles_fle_0_11'	Mean Thorax angle in the transverse plane over 0-10% of the stance phase
'ThoraxAngles_fle_32_56'	Mean Thorax angle in the transverse plane over 32-55% of the stance phase
'ThoraxAngles_fle_90_101'	Mean Thorax angle in the transverse plane over 90-100% of the stance phase
'ThoraxAngles_rot__standard_deviation'	Thorax angle standard deviation in the transverse plane for the whole stance phase
'ThoraxAngles_rot_0_9'	Mean Thorax angle in the transverse plane over 0-8% of the stance phase
'ThoraxAngles_rot_27_55'	Mean Thorax angle in the transverse plane over 27-54% of the stance phase
'ThoraxAngles_rot_89_101'	Mean Thorax angle in the transverse plane over 89-100% of the stance phase
'ThoraxVelocity_abd__maximum'	Maximum Thorax velocity in the frontal plane for the whole stance phase
'ThoraxVelocity_abd__mean'	Mean Thorax velocity in the frontal plane for the whole stance phase
'ThoraxVelocity_abd__median'	Median Thorax velocity in the frontal plane for the whole stance phase
'ThoraxVelocity_abd__minimum'	Minimum Thorax velocity in the frontal plane for the whole stance phase
'ThoraxVelocity_abd__standard_deviation'	Thorax velocity standard deviation in the frontal plane for the whole stance phase
'ThoraxVelocity_abd_0_7'	Mean Thorax velocity in the frontal plane over 0-6% of the stance phase
'ThoraxVelocity_abd_15_29'	Mean Thorax velocity in the frontal plane over 15-28% of the stance phase
'ThoraxVelocity_abd_37_49'	Mean Thorax velocity in the frontal plane over 37-48% of the stance phase

Table 3 Feature list and description (*continued from previous page*)

Feature Name	Description
'ThoraxVelocity_abd_63_78'	Mean Thorax velocity in the frontal plane over 63-77% of the stance phase
'ThoraxVelocity_abd_90_101'	Mean Thorax velocity in the frontal plane over 90-100% of the stance phase
'ThoraxVelocity_fle__maximum'	Maximum Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_fle__mean'	Mean Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_fle__median'	Median Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_fle__minimum'	Minimum Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_fle__standard_deviation'	Thorax velocity standard deviation in the transverse plane for the whole stance phase
'ThoraxVelocity_fle_0_7'	Mean Thorax velocity in the transverse plane over 0-6% of the stance phase
'ThoraxVelocity_fle_13_26'	Mean Thorax velocity in the transverse plane over 13-25% of the stance phase
'ThoraxVelocity_fle_39_49'	Mean Thorax velocity in the transverse plane over 39-48% of the stance phase
'ThoraxVelocity_fle_73_85'	Mean Thorax velocity in the transverse plane over 73-84% of the stance phase
'ThoraxVelocity_fle_96_101'	Mean Thorax velocity in the transverse plane over 96-100% of the stance phase
'ThoraxVelocity_rot__maximum'	Maximum Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_rot__mean'	Mean Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_rot__median'	Median Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_rot__minimum'	Minimum Thorax velocity in the transverse plane for the whole stance phase
'ThoraxVelocity_rot__standard_deviation'	Thorax velocity standard deviation in the transverse plane for the whole stance phase
'ThoraxVelocity_rot_0_8'	Mean Thorax velocity in the transverse plane over 0-7% of the stance phase
'ThoraxVelocity_rot_19_34'	Mean Thorax velocity in the transverse plane over 19-33% of the stance phase
'ThoraxVelocity_rot_47_61'	Mean Thorax velocity in the transverse plane over 47-60% of the stance phase
'ThoraxVelocity_rot_64_82'	Mean Thorax velocity in the transverse plane over 64-81% of the stance phase

6.4 Clustering Models

In line with the no free lunch theorem (Wolpert; 1996), a wide range of clustering models were explored in this project in order to identify a suitable clustering solution in the foot-strike patterns. All clustering models with the exception of HDBSCAN were implemented using scikit-learn (Pedregosa et al.; 2011).
The two most widely studied clustering models are partitional and hierarchical clustering (Aggarwal and Reddy; 2013). Similarly, within the biomechanics domain, it would appear the k-means and Hierarchical clustering are the most widely utilised algorithms. As such, these were the first two models implemented in this project.

K-means: K-means is the most widely used partitional clustering model (Aggarwal and Reddy; 2013), which aims to minimize the within sum of squares of the clusters. K-means tends to perform very quickly however it susceptible to noise, it has an assumption of convex clusters and requires the number of clusters to be defined in advance (VanderPlas; 2016). Within this current project, K-means was implemented using the default scikit learn algorithm, 'elkan' with 300 iterations. To initialise the centroids, the k-means++ algorithm was implemented as this improves both the speed and the accuracy of k-means in comparison to random initialisation (Arthur and Vassilvitskii; 2007). In order to account for the variation in the initialisation, this model was implemented 10 times and the solution with the best inertia was retained. K-means was implemented using k of size 2-5.

Hierarchical clustering: Hierarchical clustering is stable clustering model that does not necessarily require the number of clusters to be defined in advance and does not have an assumption of globular or convex clusters (VanderPlas; 2016). However, it can also be influenced by noise. Within this project, Hierarchical clustering was implemented using the four linkage options available in the scikit-learn package ('ward', 'complete', 'average' and 'single'). Rather than subjectively interpreting a dendrogram, hierarchical clustering was implement using predefined number of clusters to form using k of size 2-5.

Mean-Shift: Mean-shift is a popular nonparametric clustering technique that is both density based and seen as a variation of the K-means model (Aggarwal and Reddy; 2013). It aims to determine local maxima present in the density of the data through an iterative convergence routine. It does not cluster every data point and as such, it is less susceptible to noise, however, it does aim for globular clusters. Mean-shift has a single key parameter to select;' bandwidth', which dictates the size of the region to search through in order to identify groups of high density. Within this project, the bandwidth was estimated using the scikit-learn function 'estimate bandwidith', which identifies the best bandwidth given the statistical properties of the dataset.

Spectral Clustering: Spectral clustering is a graph-based method which learns the clusters in the data by following the underlying manifolds (Aggarwal and Reddy; 2013). It also does not assume globular clusters, but it is sensitive to noise in the data. Within this current project Spectral clustering was performed by constructing an affinity matrix using a radial basis function kernel. The kernel coefficient was set to one, and the number of eigen vectors was set to the number of clusters as per default in the scikit-learn package. Cluster labels were assigned using discretization, which is less sensitive to initialization

in comparison to k-means. Finally, spectral clustering was assessed using a predefined number of clusters to form with k of size 2-5.

HDBSCAN: HDBSCAN is a density-based method that is an evolution of the popular DBSCAN model. Like its predecessor, HDBSCAN does not require clusters to be globular and is not largely affected by noise (Aggarwal and Reddy; 2013). However, in comparison to the DBSCAN model, HDBSCAN can handle clusters of varying densities. Within this project, HDBSCAN was implemented using the HDBSCAN package (McInnes et al.; 2017). The two main parameters to select in this algorithm are min cluster size and min samples which control the minimum cluster size you wish to consider a cluster and how conservative you want you clustering to be respectively. Within this project, min cluster size was assessed for 2,4 and 6% of the total sample size (n), while min samples was assessed from 1 to log(n).

OPTICS: Similar to HDBCSAN, OPTICS is a density-based method that is an evolution of the popular DBSCAN model. However, in comparison to the DBSCAN model, OPTICS relaxes the requirements to specify a single distance value in which two samples datapoints are considered neighbours (Aggarwal and Reddy; 2013). OPTICS does not require clusters to be globular and is not largely affected by noise. Within this current project, a single clustering parameter (min cluster size), was evaluated for five values (5,10,15,20,25) which controls the number of samples required for a point to considered a core point.

Traditional Approach: The final grouping was conducted using the traditional approach of identifying foot-strike types by the angle of the foot at the instance of initial contact (Altman and Davis; 2012). Using this approach a forefoot strike would be classified when the foot angle was <- 1.6°, a rearfoot strike would be classified with a foot angle of >8.0°, while a mid-foot strike is defined when the foot angle was between these two thresholds.

6.5 Classification Models

Given that there is no such thing as a universally best machine learning algorithm (Wolpert; 1996), six classification models were assessed in this project. The justification for there choice and hyperparameters are presented below.

Naïve Bayes: Firstly, Naïve Bayes was implemented as parametric generative classifier (Kelleher et al.; 2015), that tends to produce fast and simple classifications with reasonable accuracy. In addition, it provides probabilistic predictions that are often easily interpretable. As such Naïve Bayes is commonly implemented as the first initial classifier, however it is recognised that given the stringent assumptions Naïve Bayes makes about data, other more complicated models may outperform it (VanderPlas; 2016).
Within this project, a single tuning parameter ‘var smoothing ‘was evaluated over 100 log spaced values between 0 and -9. This controls the portion of the largest variance of all features that is added to variances for calculation stability.

Elastic Net Logistic Regression: Logistic Regression was implemented as parametric discriminative classifier (Kelleher et al.; 2015), that like the Naïve Bayes model, tends to results in a highly interpretable model that is fast to train. However, with increasing dimensionality of data, there is increased risk of overfitting. To increase the generalisability of a logistic regression model and reduce its variance, Elastic Net regularisation was implemented which combines both lasso (L1) and Ridge (L2) regularisations (Zou and Hastie; 2005).
Elastic net regularization is a linear combination of the lasso (L1) and Ridge (L2) regularization (Zou and Hastie; 2005). Within this current project, 15 regularization values were tested from 1e-6 to 1 along with 10 values of the l1 ratio tuning parameters from 0.1 to 1.

Bagged SVM: Support Vector Machines were implemented as a non-parametric discriminative and non-linear classifier (Kelleher et al.; 2015). As they are only affected by points near the margin of their boundary, they tend to work well with high dimensional data (VanderPlas; 2016). Despite this, scaling to large number of samples can be computationally prohibitive. As such, within this current project, Support Vector Machines was implemented using bootstrap aggregated ensembles. The advantage of this approach is that it speeds up convergence to a suitable solution, and theoretically several weak learners tend to outperform one strong learner (VanderPlas; 2016).
For the bagged SVM, the hyperparameters were focused on the base classifier. As such, four values of C and gamma were evaluated (0.1,1,10,100) which control the size of the hyperplane margin and the level of curvature in the hyperplane respectively. In addition, three nonlinear kernels were tested [gaussian kernel (rbf), polynomial kernel (ploy) and sigmoid kernel (sigmoid)].

Random Forest and AdaBoost: Random Forest and Adaboost were implemented as nonparametric, discriminative ensemble methods (Kelleher et al.; 2015). Both approaches are ensembles of decision trees. The disadvantage of decision trees is that they are prone to overfitting the training data. Ensemble learning reduce this bias, by training multiple weak learners combining the findings for a better classification. Random Forest is an ensemble of randomised decision trees that are trained in parallel and the results are aggregated. In contrast Adaboost is an ensemble of decision trees that are trained sequentially with each subsequent tree increasing the weight of misclassified data points (Geéron; 2017).

The Random Forest model was tuned with a combination of six hyperparameters. Four values were evaluated for the min samples per split and min samples per leaf parameters (0.005, 0.01, 0.05 and 0.10 multiplied by the number of samples). These control the minimum required number of observations in a node in order to split it and the minimum number of observations in a node after splitting it respectively. Six values of max dept were considered (3, 4, 5, 6, 7, 8) which control the longest path between the root node and the leaf node in a tree. Four values for the number of estimators were tested (300, 500, 800, 1000) which control the number of base trees in the random forest. The maximum number of features to consider when looking for the best split was set at either the $\sqrt{(}$number of features) or log2(number of features). Finally, the measure used to determine the quality of each split in a tree was evaluated over two metrics (gini and entropy).

For the Adaboost algorithm, the base estimator was set as a decision tree with variations in its max depth ranging from 1 to 6. Similar to the Random Forest model, the number of estimators was evaluated over four values (300, 500, 800, 1000). Finally, eight values controlling the learning rate of subsequent tree iterations were evaluated (0.001, 0.01, 0.05, 0.1, 0.25, 0.50, 0.75, 1.0).

Stacked Ensemble: Finally, a weighted, stacked ensemble model was implemented which performs a weighted majority classification from the aforementioned models (Geéron; 2017). Given that Scikit-Learn does not directly support weighted stacking, a pragmatic equation was proposed and utilised in this current project. After firstly transforming predicted negative classes from zero to minus one, the aggregative classification was calculated as follows:

$$obv(x) \begin{cases} 0 \left(\sum_{c=1}^{n} v_{xc} * \overline{\int ROC_{\mathrm{c}}} * \left| \left[\hat{P}_{xc}(1) - j_c \right] \right| * w \right) < 0 \\ 1 \left(\sum_{c=1}^{n} v_{xc} * \overline{\int ROC_{\mathrm{c}}} * \left| \left[\hat{P}_{xc}(1) - j_c \right] \right| * w \right) >= 0 \end{cases}$$

Each observation (x), is classified as a 0 should the sum of the weighted voting be less than zero, otherwise it is a 1. Where v_{xc} is the class vote for the xth observation by the cth classifier. $\overline{\int ROC_{\mathrm{c}}}$ is the average area under the receiver operator curve for c^{th} classifier across the one hundred bootstrapped samples. $\left| \left[\hat{P}_{xc}(1) - j_c \right] \right|$ is the classifier confidence for the class vote for the xth observation, where $\hat{P}_{xc}(1)$ is the estimated probability by the cth classifier of the xth observation being a positive class (1) and j_c is the classification cut off probability as defined by Youden's J statistic. Finally, w is an exponent weighting bias, that would increase the weighting placed on the most confident and best predictor as w increases. Within this current project, w was incremented from 0 -100 in increments of five, with the aim of optimising the overall Youden's J statistic.

6.6 Clustering Evaluation

All plots from the evaluated clustering solutions are presented below (Figure 115 - Figure 120).

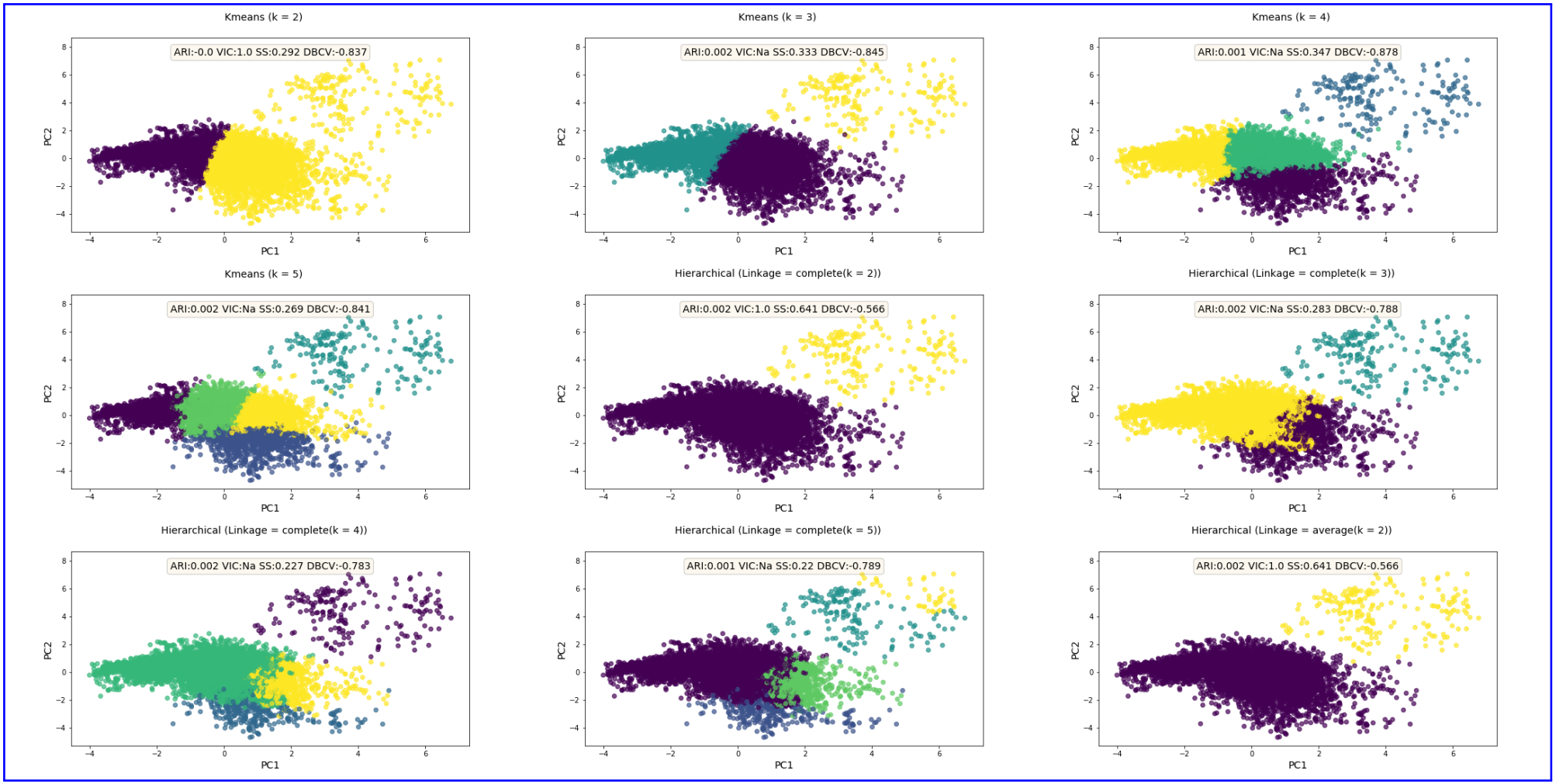


Figure 115: Visualising the cluster solution labels overlayed on the first two principle components of the data (1 of 6)

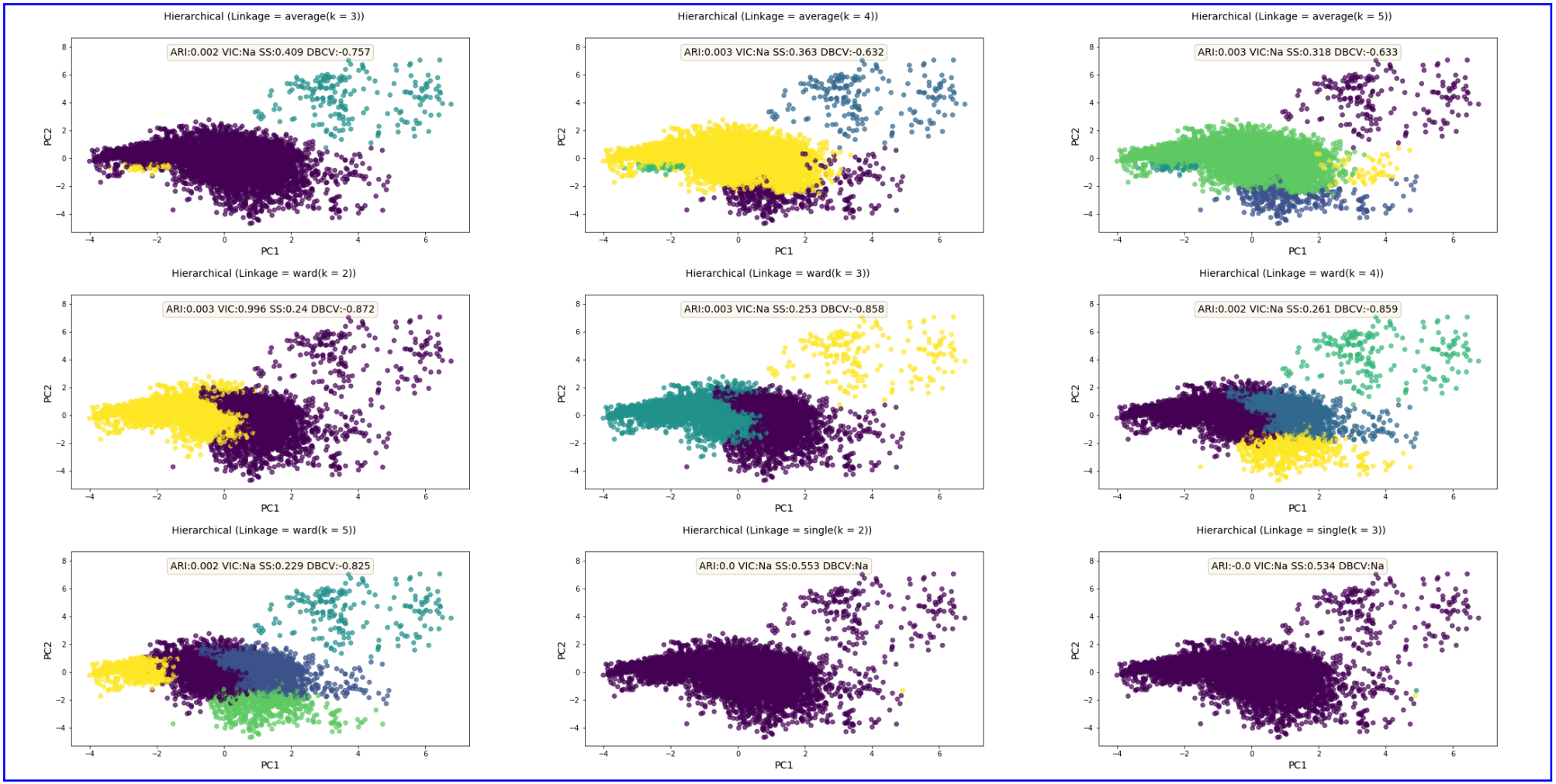


Figure 116: Visualising the cluster solution labels overlayed on the first two principle components of the data (2 of 6)

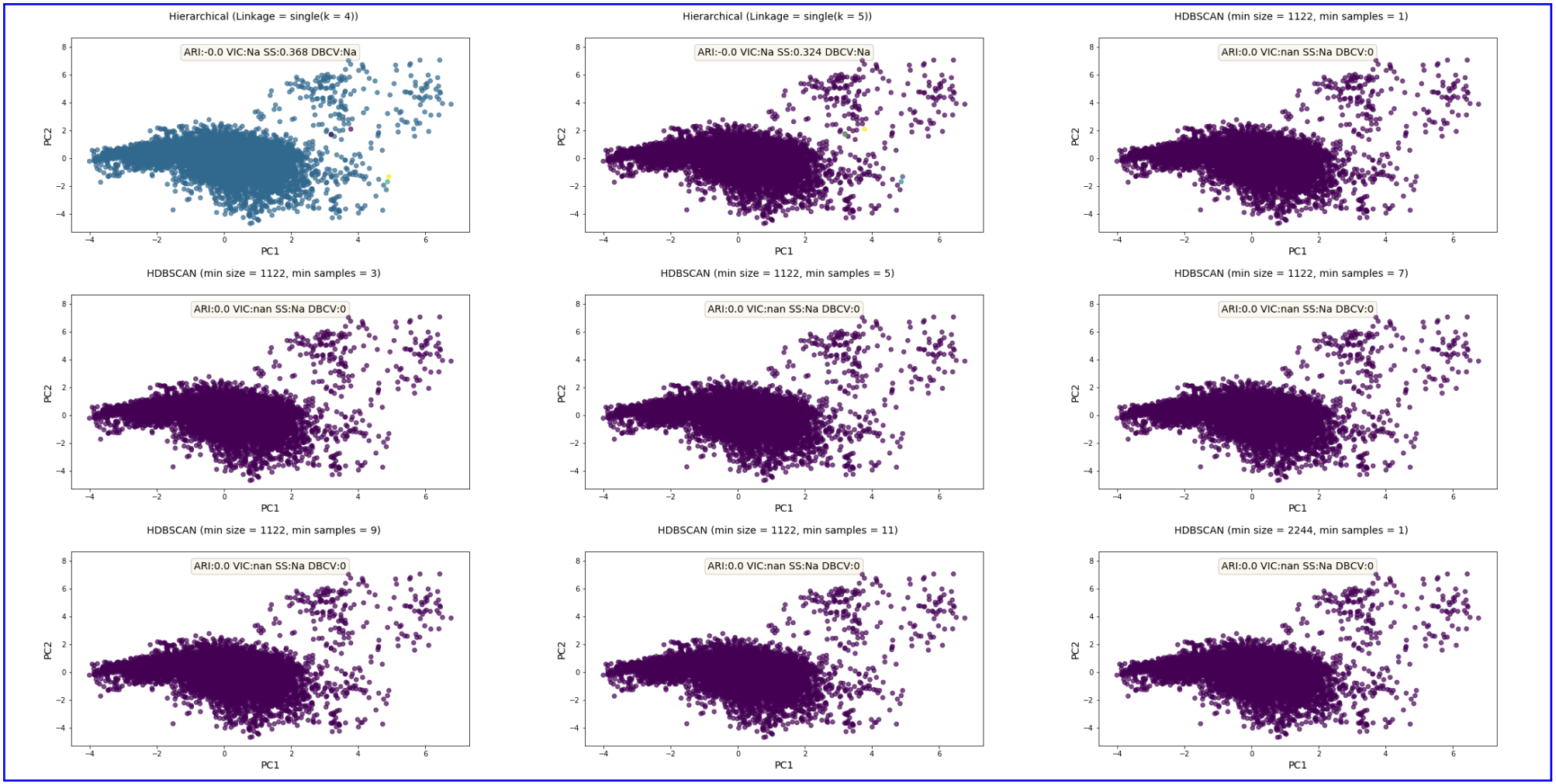


Figure 117: Visualising the cluster solution labels overlayed on the first two principle components of the data (3 of 6)

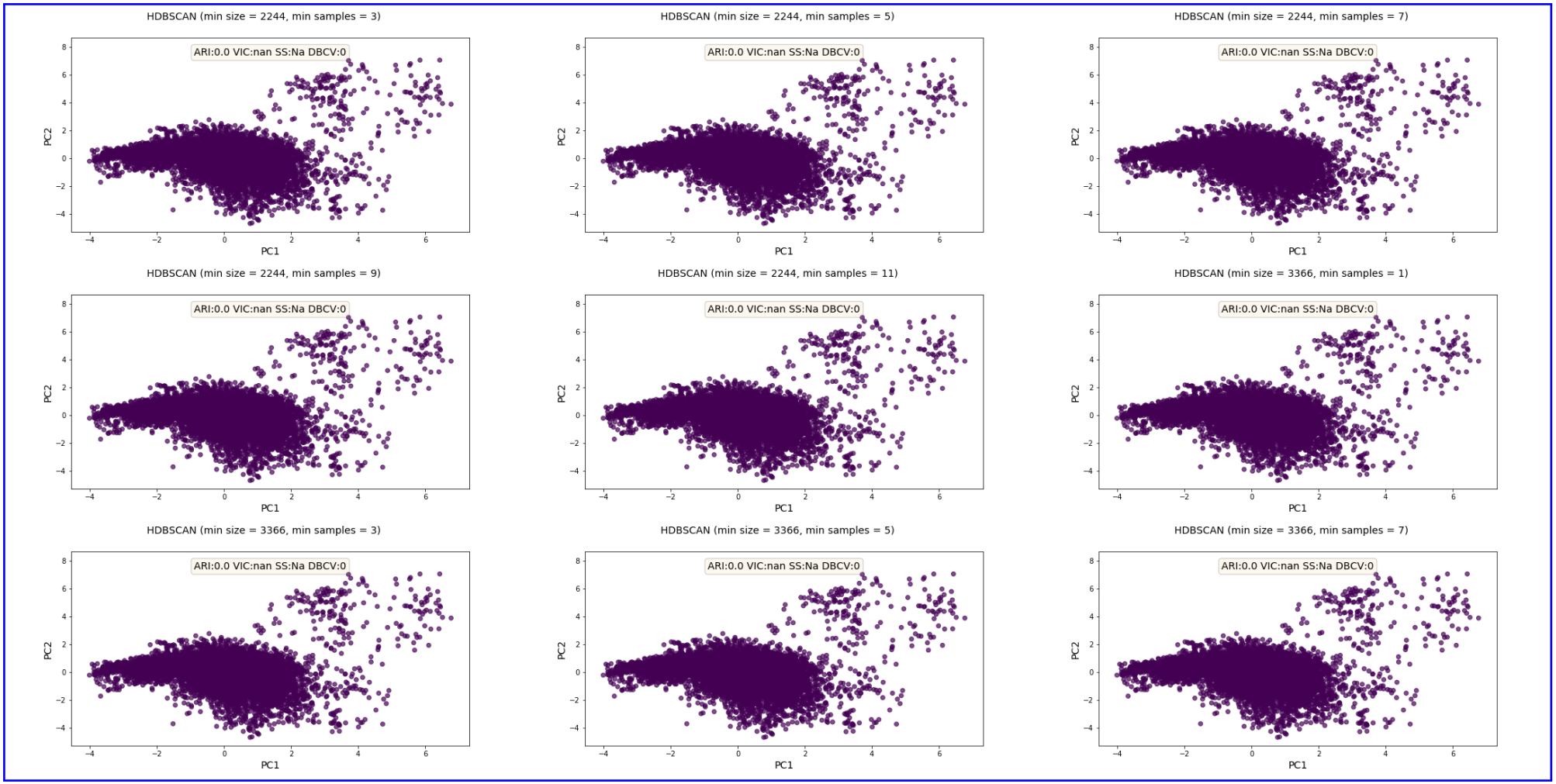


Figure 118: Visualising the cluster solution labels overlayed on the first two principle components of the data (4 of 6)

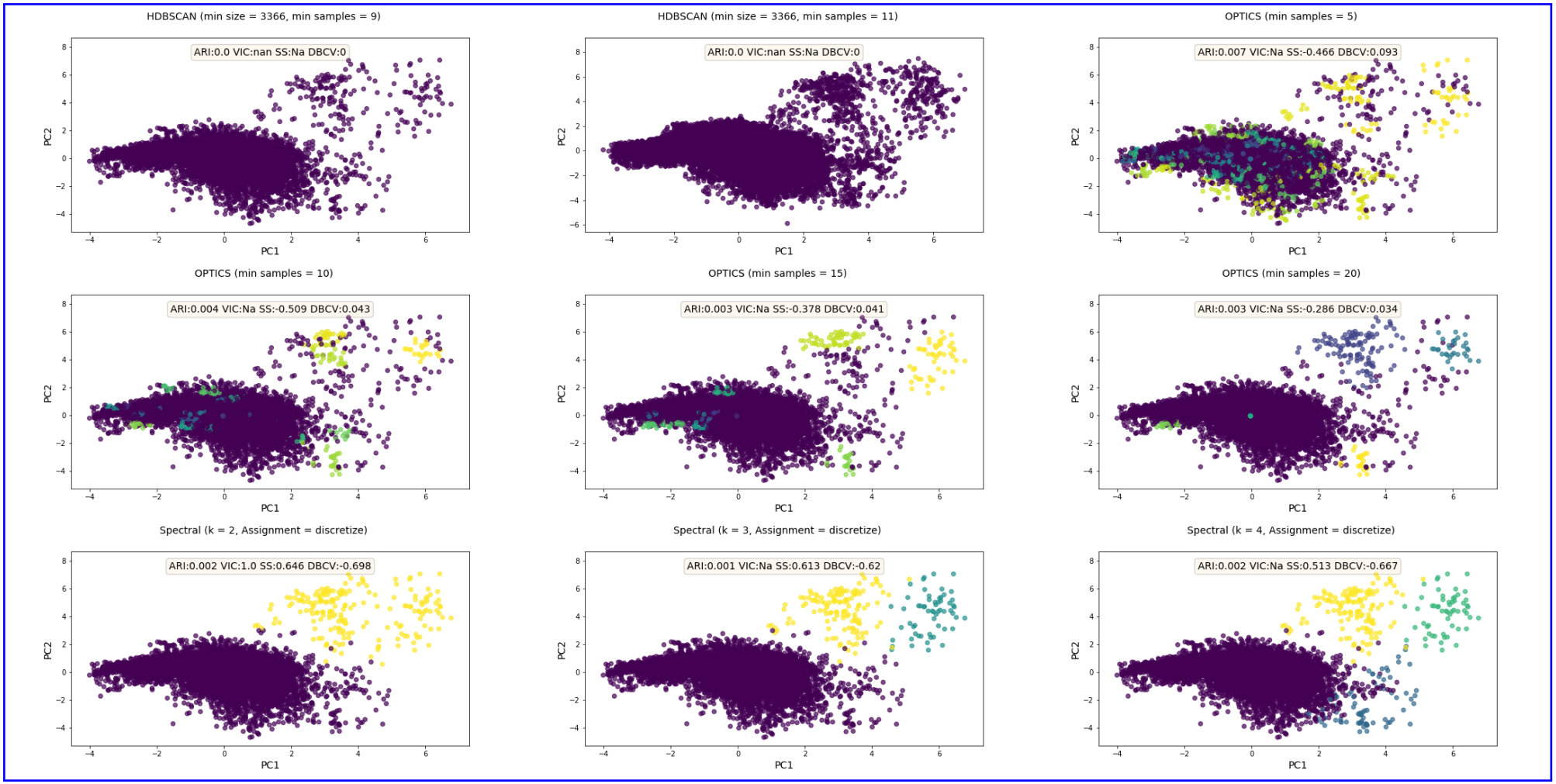


Figure 119: Visualising the cluster solution labels overlayed on the first two principle components of the data (5 of 6)

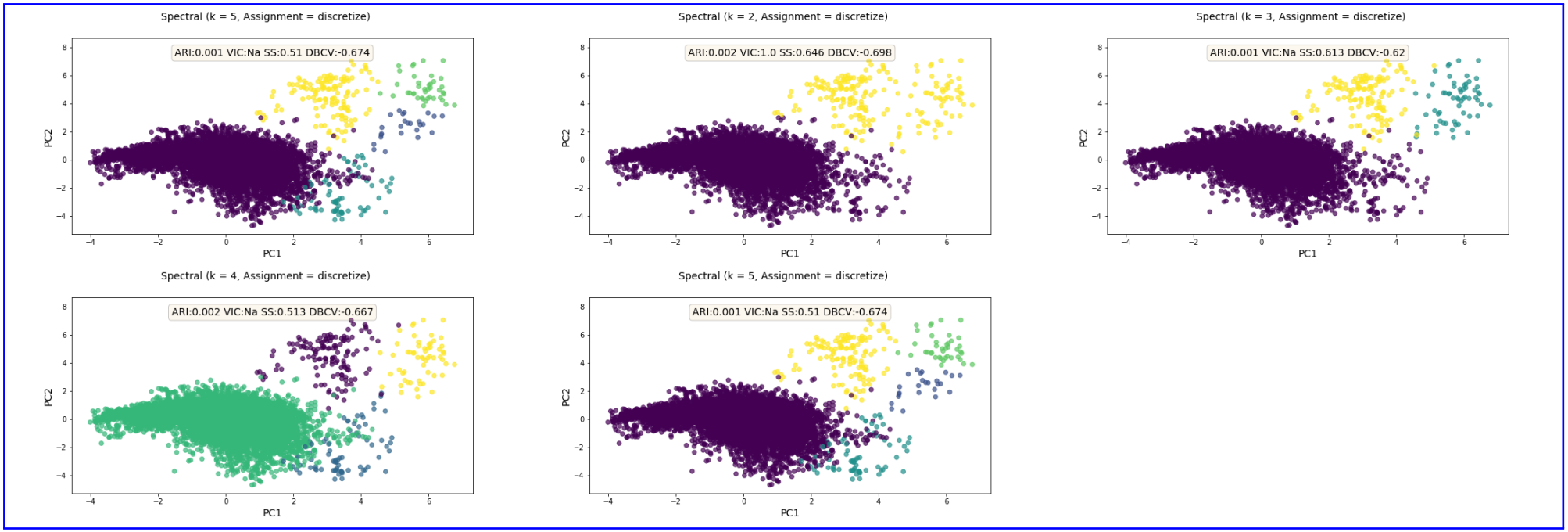


Figure 120: Visualising the cluster solution labels overlayed on the first two principle components of the data (6 of 6)

6.6.1 Post hoc tests for Adjusted Rand Index

Full post hoc findings for the bootstrapped Adjusted Rand Index (ARI) scores are presented in table 4. A Games-Howell post hoc test indicated the only approaches that was not statistically significantly different from another was the Hierarchical vs Spectral comparison (p = 0.35, D = 0.26). All other pairwise comparisons were statistically different with effect sizes ranging from medium to large (p <0.05, D = 0.42 – 4.17).

Table 4: Post hoc pair wise comparison for the bootstrapped ARI results

Comparison	**Mean Difference**	**p-val**	**Cohen's D**
Hierarchical vs. K-means	-0.001	<0.01	-0.91
Hierarchical vs. OPTICS	-0.015	<0.01	-3.67
Hierarchical vs. Spectral	0.001	0.35	-0.26
Hierarchical vs. Taditional	0.002	<0.01	1.66
K-means vs. OPTICS	-0.013	<0.01	-3.28
K-means vs. Spectral	0.001	0.02	0.42
K-means vs. Taditional	0.003	<0.01	2.47
OPTICS vs. Spectral	0.014	<0.01	3.35
OPTICS vs. Taditional	0.016	<0.01	4.17
Spectral vs. Taditional	0.002	<0.01	1.24

6.7 Classification Evaluation

This section presents the full posthoc comparisons for the bootstrap comparisons of accuracy, specificity and sensistivity. Post hoc comparisons were conducted using Games-Howell tests given the hetrogenity in the variances across the groups. The Games-Howell test is similar to the Tukey HSD test, in that it uses Tukey's studentized range distribution but is based on Welch's degrees of freedom correction. It is robust to both unequal variances and non-normality. Standardised effect sizes were reported using Cohen's D.

6.7.1 Accuracy

Full post hoc findings for the bootstrapped accuracy analysis are presented in table 5. A Games-Howell post hoc test indicated the only approaches that were not statistically significantly different from another were AdaBoost vs Naïve Majority comparison (p = 0.29, D = 0.31) along with Random Forest vs Stacked Ensemble comparison (p = 0.89, D = 0.17). All other pairwise comparisons were statistically different with large effect sizes (p <0.01, D = 1.42 – 27.95)

Table 5: Post hoc pair wise comparison for the bootstrapped accuracy results

Comparison	**Mean Difference**	**p-val**	**Cohen's D**
AdaBoost vs. Elastic Net	0.06	<0.01	6.86
AdaBoost vs. Naive Bayes	0.03	<0.01	1.73
AdaBoost vs. Naive Majority	0.00	0.29	0.31
AdaBoost vs. Random Forest	-0.06	<0.01	-3.51
AdaBoost vs. SVM Bag	0.14	<0.01	19.30
AdaBoost vs. Stacked Ensemble	-0.05	<0.01	-3.49
Elastic Net vs. Naive Bayes	-0.03	<0.01	-1.42
Elastic Net vs. Naive Majority	-0.06	<0.01	-8.50
Elastic Net vs. Random Forest	-0.12	<0.01	-7.17
Elastic Net vs. SVM Bag	0.08	<0.01	11.84
Elastic Net vs. Stacked Ensemble	-0.11	<0.01	-7.38
Naive Bayes vs. Naive Majority	-0.03	<0.01	-1.68
Naive Bayes vs. Random Forest	-0.09	<0.01	-3.86
Naive Bayes vs. SVM Bag	0.11	<0.01	6.06
Naive Bayes vs. Stacked Ensemble	-0.09	<0.01	-3.82
Naive Majority vs. Random Forest	-0.06	<0.01	-3.86
Naive Majority vs. SVM Bag	0.14	<0.01	27.95
Naive Majority vs. Stacked Ensemble	-0.06	<0.01	-3.88
Random Forest vs. SVM Bag	0.20	<0.01	12.77
Random Forest vs. Stacked Ensemble	0.00	0.89	0.17
SVM Bag vs. Stacked Ensemble	-0.20	<0.01	-13.39

6.7.2 Specificity

Full post hoc findings for the bootstrapped specificity analysis are presented in table 6. Games-Howell post hoc tests indicated that all methods were statistically significantly different from one another (p <0.01), with the exception of the Random Forest model vs Stacked Ensemble comparison (p = 0.71, D = 0.2) and the Naive Bayes vs Random Forest comparison (p = 0.32, D = 0.3).

Table 6: Post hoc pair wise comparison for the bootstrapped specificity results

Comparison	**Mean Difference**	**p-val**	**Cohen's D**
AdaBoost vs. Elastic Net	-0.03	<0.01	-1.1
AdaBoost vs. Naive Bayes	0.08	<0.01	1.3
AdaBoost vs. Random Forest	0.06	<0.01	1.1
AdaBoost vs. SVM Bag	0.23	<0.01	10.7
AdaBoost vs. Stacked Ensemble	0.04	<0.01	0.9
Elastic Net vs. Naive Bayes	0.11	<0.01	1.8
Elastic Net vs. Random Forest	0.09	<0.01	1.7
Elastic Net vs. SVM Bag	0.26	<0.01	16.8
Elastic Net vs. Stacked Ensemble	0.07	<0.01	1.6
Naive Bayes vs. Random Forest	-0.02	0.32	-0.3
Naive Bayes vs. SVM Bag	0.16	<0.01	2.7
Naive Bayes vs. Stacked Ensemble	-0.04	<0.01	-0.5
Random Forest vs. SVM Bag	0.18	<0.01	3.7
Random Forest vs. Stacked Ensemble	-0.01	0.71	-0.2
SVM Bag vs. Stacked Ensemble	-0.19	<0.01	-4.4

6.7.3 Sensitivity

Full post hoc findings for the bootstrapped specificity analysis are presented in table 7. Games-Howell post hoc tests indicated that all methods were statistically significantly different from one another (p <0.01), with the exception of the Random Forest model vs Stacked Ensemble comparison (p = 0.78, D = 0.20) and the AdaBoost vs Naive Bayes comparison (p = 0.9, D = 0.12).

Table 7: Post hoc pair wise comparison for the bootstrapped sensitivity results

Comparison	**Mean Difference**	**p-val**	**Cohen's D**
AdaBoost vs. Elastic Net	0.11	<0.01	4.38
AdaBoost vs. Naive Bayes	0.01	0.9	0.12
AdaBoost vs. Random Forest	-0.12	<0.01	-2.30
AdaBoost vs. SVM Bag	0.09	<0.01	4.65
AdaBoost vs. Stacked Ensemble	-0.11	<0.01	-2.19
Elastic Net vs. Naive Bayes	-0.10	<0.01	-1.65
Elastic Net vs. Random Forest	-0.23	<0.01	-4.49
Elastic Net vs. SVM Bag	-0.01	<0.01	-0.89
Elastic Net vs. Stacked Ensemble	-0.21	<0.01	-4.54
Naive Bayes vs. Random Forest	-0.13	<0.01	-1.68
Naive Bayes vs. SVM Bag	0.09	<0.01	1.47
Naive Bayes vs. Stacked Ensemble	-0.11	<0.01	-1.56
Random Forest vs. SVM Bag	0.21	<0.01	4.39
Random Forest vs. Stacked Ensemble	0.01	0.78	0.20
SVM Bag vs. Stacked Ensemble	-0.20	<0.01	-4.45

References

Aggarwal, C. C. and Reddy, C. K. (2013). *Data Clustering Algorithms and Applications*, Chapman and Hall.

Altman, A. R. and Davis, I. S. (2012). A kinematic method for footstrike pattern detection in barefoot and shod runners, *Gait Posture* **35**(2): 298–300.

Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding, *Proc. Annu. ACM-SIAM Symp. Discret. Algorithms*, Vol. 07-09-Janu, pp. 1027–1035.

Bahr, R. (2016). Why screening tests to predict injury do not work-and probably never will.: A critical review, *Br. J. Sports Med.* **50**(13): 776–780.

Barre, A. and Armand, S. (2014). Biomechanical ToolKit: Open-source framework to visualize and process biomechanical data, *Comput. Methods Programs Biomed.* **114**(1): 80–87.

Blackburn, J. T. and Padua, D. A. (2008). Influence of trunk flexion on hip and knee joint kinematics during a controlled drop landing, *Clin. Biomech.* **23**(3): 313–319.

Bredeweg, S. W., Buist, I. and Kluitenberg, B. (2013). Differences in kinetic asymmetry between injured and noninjured novice runners: A prospective cohort study, *Gait Posture* **38**(4): 847–852.

Bredeweg, S. W., Kluitenberg, B., Bessem, B. and Buist, I. (2013). Differences in kinetic variables between injured and noninjured novice runners: A prospective cohort study, *J. Sci. Med. Sport* **16**(3): 205–210.

Brund, R. B., Rasmussen, S., Nielsen, R. O., Kersting, U. G., Laessoe, U. and Voigt, M. (2017). Medial shoe-ground pressure and specific running injuries: A 1-year prospective cohort study, *J. Sci. Med. Sport* **20**(9): 830–834.

Ceyssens, L., Vanelderen, R., Barton, C., Malliaras, P. and Dingenen, B. (2019). Biomechanical Risk Factors Associated with Running-Related Injuries: A Systematic Review, *Sport. Med.* **49**(7): 1095–1115.

Christ, M., Braun, N., Neuffer, J. and Kempa-Liehr, A. W. (2018). Time Series FeaTuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package), *Neurocomputing* **307**: 72–77.

Davis, I. S., Bowser, B. J. and Mullineaux, D. R. (2016). Greater vertical impact loading in female runners with medically diagnosed injuries: A prospective investigation, *Br. J. Sports Med.* **50**(14): 887–892.

Dingwell, J. B., Cusumano, J. P., Cavanagh, P. R. and Sternad, D. (2001). Local dynamic stability versus kinematic variability of continuous overground and treadmill walking, *J. Biomech. Eng.* **123**(1): 27–32.

Dudley, R. I., Pamukoff, D. N., Lynn, S. K., Kersey, R. D. and Noffal, G. J. (2017). A prospective comparison of lower extremity kinematics and kinetics between injured and non-injured collegiate cross country runners, *Hum. Mov. Sci.* **52**: 197–202.

Geéron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.*

Ghani Zadeh Hesar, N., Van Ginckel, A., Cools, A., Peersman, W., Roosen, P., De Clercq, D. and Witvrouw, E. (2009). A prospective study on gait-related intrinsic risk factors for lower leg overuse injuries, *Br. J. Sports Med.* **43**(13): 1057–1061.

Handsaker, J. C., Forrester, S. E., Folland, J. P., Black, M. I. and Allen, S. J. (2016). A kinematic algorithm to identify gait events during running at different speeds and with different footstrike types, *J. Biomech.* **49**(16): 4128–4133.

Hein, T., Janssen, P., Wagner-Fritz, U., Haupt, G. and Grau, S. (2014). Prospective analysis of intrinsic and extrinsic risk factors on the development of Achilles tendon pain in runners, *Scand. J. Med. Sci. Sport.* **24**(3).

Hubert, L. and Arabie, P. (1985). Comparing partitions, *J. Classif.* **2**(1): 193–218.

Kelleher, J. D., Namee, B. M. and D'Arcy, A. (2015). *Fundamentals of Machine Learning for Predictive Data Analytics*, The MIT Press.

Kuhman, D. J., Paquette, M. R., Peel, S. A. and Melcher, D. A. (2016). Comparison of ankle kinematics and ground reaction forces between prospectively injured and uninjured collegiate cross country runners, *Hum. Mov. Sci.* **47**: 9–15.

Luedke, L. E., Heiderscheit, B. C., Williams, D. S. and Rauh, M. J. (2016). Influence of Step Rate on Shin Injury and Anterior Knee Pain in High School Runners, *Med. Sci. Sports Exerc.* **48**(7): 1244–1250.

McInnes, L., Healy, J. and Astels, S. (2017). hdbscan: Hierarchical density based clustering, *J. Open Source Softw.* **2**(11): 205.

Messier, S. P., Martin, D. F., Mihalko, S. L., Ip, E., DeVita, P., Cannon, D. W., Love, M., Beringer, D., Saldana, S., Fellin, R. E. and Seay, J. F. (2018). A 2-Year Prospective Cohort Study of Overuse Running Injuries: The Runners and Injury Longitudinal Study (TRAILS), *Am. J. Sports Med.* **46**(9): 2211–2221.

Moudy, S., Richter, C. and Strike, S. (2018). Landmark registering waveform data improves the ability to predict performance measures, *J. Biomech.* **78**: 109–117.

Moulavi, D., Jaskowiak, P. A., Campello, R. J., Zimek, A. and Sander, J. (2014). Density-based clustering validation, *SIAM Int. Conf. Data Min. 2014, SDM 2014*, Vol. 2, pp. 839–847.

Napier, C., MacLean, C. L., Maurer, J., Taunton, J. E. and Hunt, M. A. (2018). Kinetic risk factors of running-related injuries in female recreational runners, *Scand. J. Med. Sci. Sport.* **28**(10): 2164–2172.

Noehren, B., Davis, I. and Hamill, J. (2007). Prospective study of the biomechanical factors associated with iliotibial band syndrome, *Clin. Biomech.* **22**(9): 951–956.

Noehren, B., Hamill, J. and Davis, I. (2013). Prospective evidence for a hip etiology in patellofemoral pain, *Med. Sci. Sports Exerc.* **45**(6): 1120–1124.

Pataky, T. C. (2012). One-dimensional statistical parametric mapping in Python, *Comput. Methods Biomech. Biomed. Engin.* **15**(3): 295–301.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É. (2011). Scikit-learn: Machine learning in Python, *J. Mach. Learn. Res.* **12**: 2825–2830.

Pohl, M. B., Mullineaux, D. R., Milner, C. E., Hamill, J. and Davis, I. S. (2008). Biomechanical predictors of retrospective tibial stress fractures in runners, *J. Biomech.* **41**(6): 1160–1165.

Richter, C., King, E., Strike, S. and Franklyn-Miller, A. (2019). Objective classification and scoring of movement deficiencies in patients with anterior cruciate ligament reconstruction, *PLoS One* **14**(7): e0206024.

Richter, C., O'Connor, N. E., Marshall, B. and Moran, K. (2014). Analysis of characterizing phases on waveforms: An application to vertical jumps, *J. Appl. Biomech.* **30**(2): 316–321.

Rodríguez, J., Medina-Pérez, M. A., Gutierrez-Rodríguez, A. E., Monroy, R. and Terashima-Marín, H. (2018). Cluster validation using an ensemble of supervised classifiers, *Knowledge-Based Syst.* **145**: 134–144.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, *J. Comput. Appl. Math.* **20**(C): 53–65.

Stefanyshyn, D. J., Stergiou, P., Lun, V. M., Meeuwisse, W. H. and Worobets, J. T. (2006). Knee angular impulse as a predictor of patellofemoral pain in runners, *Am. J. Sports Med.* **34**(11): 1844–1851.

Taunton, J. E., Ryan, M. B., Clement, D. B., McKenzie, D. C., Lloyd-Smith, D. R. and Zumbo, B. D. (2002). A retrospective case-control analysis of 2002 running injuries, *Br. J. Sports Med.* **36**(2): 95–101.

Thijs, Y., De Clercq, D., Roosen, P. and Witvrouw, E. (2008). Gait-related intrinsic risk factors for patellofemoral pain in novice recreational runners, *Br. J. Sports Med.* **42**(6): 466–471.

Tibshirani, R. and Walther, G. (2005). Cluster validation by prediction strength, *J. Comput. Graph. Stat.* **14**(3): 511–528.

Van Ginckel, A., Thijs, Y., Hesar, N. G. Z., Mahieu, N., De Clercq, D., Roosen, P. and Witvrouw, E. (2009). Intrinsic gait-related risk factors for Achilles tendinopathy in novice runners: A prospective study, *Gait Posture* **29**(3): 387–391.

VanderPlas, J. (2016). *Python Data Science Handbook*, O'Reilly Media.

Winter, D. A. (2009). *Biomechanics and Motor Control of Human Movement: Fourth Edition*, John Wiley & Sons.

Wolpert, D. H. (1996). The Lack of a Priori Distinctions between Learning Algorithms, *Neural Comput.* **8**(7): 1341–1390.

Zhao, Z. and Liu, H. (2007). Spectral feature selection for supervised and unsupervised learning, *ACM Int. Conf. Proceeding Ser.*

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net, *J. R. Stat. Soc. Ser. B Stat. Methodol.* **67**(2): 301–320.