Configuration Manual: Early Diagnosis of Parkinson’s Disease Progression

MSc Research Project
Data Analytics

Dharesh Vadalia
Student ID: x18192076

School of Computing
National College of Ireland

Supervisor: Prof. Christian Horn

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Dharesh Vadalia		
Student ID:	X18192076		
Programme:	Master of Science - Data Analytics	**Year:**	2020
Module:	M.Sc. Research Project		
Supervisor:	Prof. Christian Horn		
Submission Due Date:	17/8/2020		
Project Title:	Configuration Manual: Early diagnosis of Parkinson's Disease Progression		
Word Count:	2006	**Page Count:**	14

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:	Dharesh Vadalia
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):	

Configuration Manual

Dharesh Vadalia
x18192076

1 Introduction

The configuration manual outlines the description of step by step process involved in environmental setup for this research project and the implementation of this project, which is motivated to identify the rate of progression in Parkinson's Disease patients based on their baseline assessment characteristic. Configuration manual includes the information about the programming language used, necessary library packages and system configuration details. Document also discusses the results obtained from different tests scenarios, plotted info graphics and outputs of evaluation metrics used in this research.

2 Specification for Environment Setup

2.1 System Specifications

Implementation of this project is carried out on Google Collaboratory (Colab). It is an open source online platform build on top of Jupiter Notebook, which allows users to run python programs on Google servers and leverages high end GPU's and TPU's free of cost for implementing machine learning model.

2.2 Technical Specifications

For implementation of this project Python 3[1] programming language is used. Following packages of Python are used in execution of project:

- Pandas 1.1.0[2]
- Numpy 1.19.1[3]
- Scikit-learn 0.23.2[4]
- Keras 2.3.0[5]
- Matplotlib 3.3.1[6]

[1] https://www.python.org
[2] https://pandas.pydata.org/
[3] https://numpy.org/
[4] https://scikit-learn.org/stable/index.html
[5] https://keras.io/
[6] https://matplotlib.org/

- Tensor flow 2.3.0[7]
- pypmi[8]

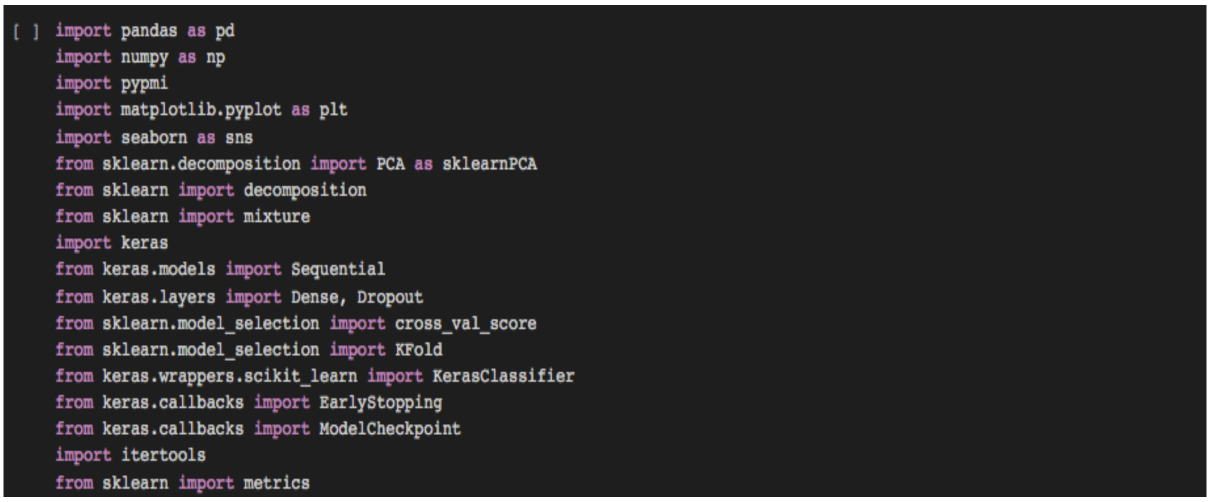
2.3 Data Source

The data for implementation of the project is collected from the Parkinson's Progression Markers Initiative (PPMI)[9] organisation funded by Michael J. Fox Foundation. PPMI pioneers in the collection and management of clinical data of PD the patients for the purpose of research in area PD diagnosis. This comprehensive set of clinical data is maintained in a public repository of PPMI which is made available to researchers on request for study purposes. Data is fetched from the repository via API call using fetch methods defined in '*pypmi'* library to communicate with PPMI data repository.

3 Implementation

This section elaborates the steps performed for end to end implementation of proposed project. Providing understanding over the techniques employed for data preparation and feature engineering, feature extraction and modelling of classification algorithms.

3.1 Necessary Library Imports



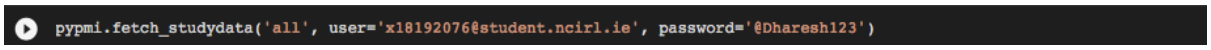
```
[ ] import pandas as pd
    import numpy as np
    import pypmi
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.decomposition import PCA as sklearnPCA
    from sklearn import decomposition
    from sklearn import mixture
    import keras
    from keras.models import Sequential
    from keras.layers import Dense, Dropout
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import KFold
    from keras.wrappers.scikit_learn import KerasClassifier
    from keras.callbacks import EarlyStopping
    from keras.callbacks import ModelCheckpoint
    import itertools
    from sklearn import metrics
```

Figure 1: Necessary Libraries

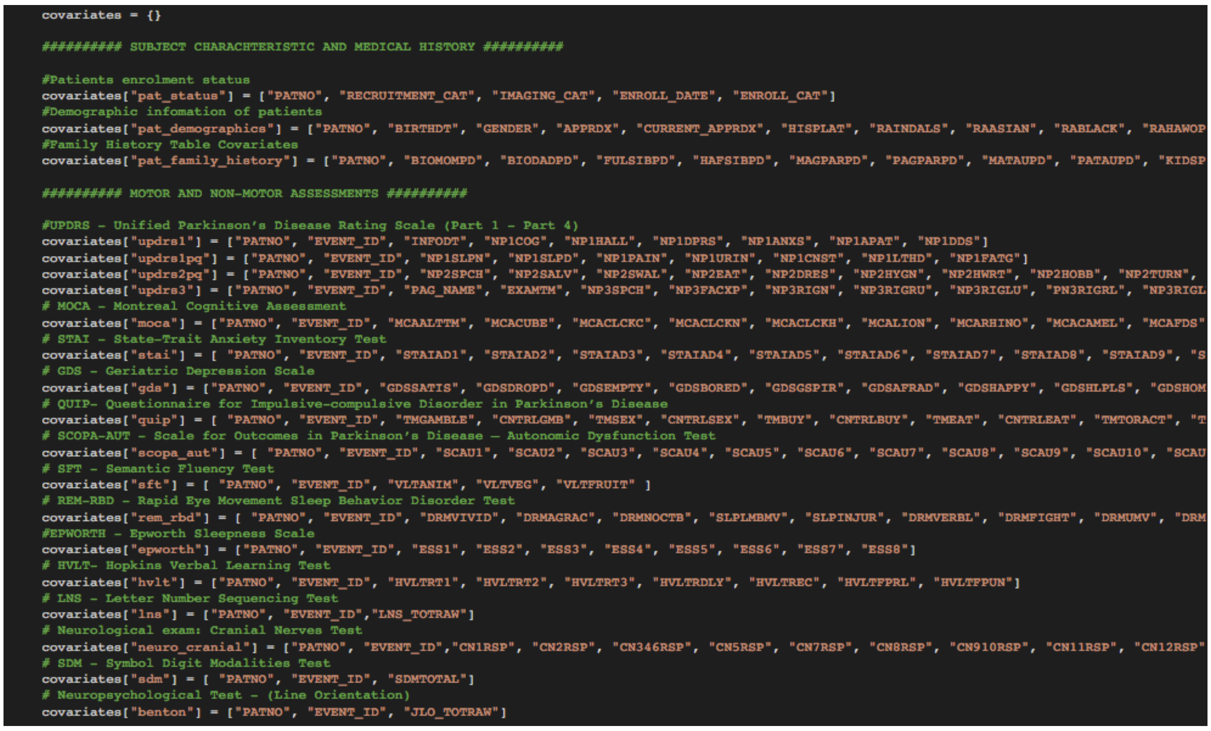
[7] https://www.tensorow.org/install/
[8] https://pypmi.readthedocs.io/en/latest/index.html
[9] http://www.ppmi-info.org

3.2 Data Loading and Pre-processing

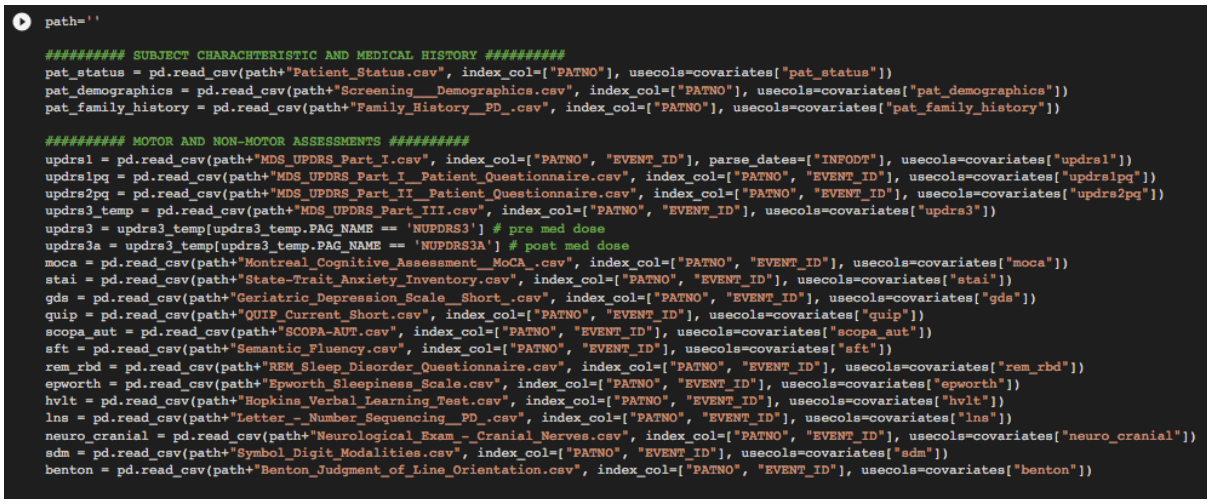
Download data from PPMI repository via **fetch_studydata()** API function of *pypmi* library. **Note: In case of poor network connection API connection may timeout while downloading the dataset. In such case re-run the line of code. Download file size is approx. 20 MB
A **Dictionary** is created with important covariates which are to be extracted from downloaded .csv files. Each .csv file represents different pre-clinical assessment test conducted to access patient's condition against biomarkers of Parkinson's Disease. Data is loaded into separate data frame for each selected clinical assessment.



```
pypmi.fetch_studydata('all', user='x18192076@student.ncirl.ie', password='@Dharesh123')
```



```
covariates = {}

########## SUBJECT CHARACHTERISTIC AND MEDICAL HISTORY ##########

#Patients enrolment status
covariates["pat_status"] = ["PATNO", "RECRUITMENT_CAT", "IMAGING_CAT", "ENROLL_DATE", "ENROLL_CAT"]
#Demographic infomation of patients
covariates["pat_demographics"] = ["PATNO", "BIRTHDT", "GENDER", "APPRDX", "CURRENT_APPRDX", "HISPLAT", "RAINDALS", "RAASIAN", "RABLACK", "RAHAWOP
#Family History Table Covariates
covariates["pat_family_history"] = ["PATNO", "BIOMOMPD", "BIODADPD", "FULSIBPD", "HAFSIBPD", "MAGPARPD", "PAGPARPD", "MATAUPD", "PATAUPD", "KIDSP

########## MOTOR AND NON-MOTOR ASSESSMENTS ##########

#UPDRS - Unified Parkinson's Disease Rating Scale (Part 1 - Part 4)
covariates["updrs1"] = ["PATNO", "EVENT_ID", "INFODT", "NP1COG", "NP1HALL", "NP1DPRS", "NP1ANXS", "NP1APAT", "NP1DDS"]
covariates["updrs1pq"] = ["PATNO", "EVENT_ID", "NP1SLPN", "NP1SLPD", "NP1PAIN", "NP1URIN", "NP1CNST", "NP1LTHD", "NP1FATG"]
covariates["updrs2pq"] = ["PATNO", "EVENT_ID", "NP2SPCH", "NP2SALV", "NP2SWAL", "NP2EAT", "NP2DRES", "NP2HYGN", "NP2HWRT", "NP2HOBB", "NP2TURN",
covariates["updrs3"] = ["PATNO", "EVENT_ID", "PAG_NAME", "EXAMTM", "NP3SPCH", "NP3FACXP", "NP3RIGN", "NP3RIGRU", "NP3RIGLU", "PN3RIGRL", "NP3RIGL
# MOCA - Montreal Cognitive Assessment
covariates["moca"] = ["PATNO", "EVENT_ID", "MCAALTTM", "MCACUBE", "MCACLCKC", "MCACLCKN", "MCACLCKH", "MCALION", "MCARHINO", "MCACAMEL", "MCAFDS"
# STAI - State-Trait Anxiety Inventory Test
covariates["stai"] = [ "PATNO", "EVENT_ID", "STAIAD1", "STAIAD2", "STAIAD3", "STAIAD4", "STAIAD5", "STAIAD6", "STAIAD7", "STAIAD8", "STAIAD9", "S
# GDS - Geriatric Depression Scale
covariates["gds"] = ["PATNO", "EVENT_ID", "GDSSATIS", "GDSDROPD", "GDSEMPTY", "GDSBORED", "GDSGSPIR", "GDSAFRAD", "GDSHAPPY", "GDSHLPLS", "GDSHOM
# QUIP- Questionnaire for Impulsive-compulsive Disorder in Parkinson's Disease
covariates["quip"] = [ "PATNO", "EVENT_ID", "TMGAMBLE", "CNTRLGMB", "TMSEX", "CNTRLSEX", "TMBUY", "CNTRLBUY", "TMEAT", "CNTRLEAT", "TMTORACT", "T
# SCOPA-AUT - Scale for Outcomes in Parkinson's Disease — Autonomic Dysfunction Test
covariates["scopa_aut"] = [ "PATNO", "EVENT_ID", "SCAU1", "SCAU2", "SCAU3", "SCAU4", "SCAU5", "SCAU6", "SCAU7", "SCAU8", "SCAU9", "SCAU10", "SCAU
# SFT - Semantic Fluency Test
covariates["sft"] = [ "PATNO", "EVENT_ID", "VLTANIM", "VLTVEG", "VLTFRUIT" ]
# REM-RBD - Rapid Eye Movement Sleep Behavior Disorder Test
covariates["rem_rbd"] = [ "PATNO", "EVENT_ID", "DRMVIVID", "DRMAGRAC", "DRMNOCTB", "SLPLMBMV", "SLPINJUR", "DRMVERBL", "DRMFIGHT", "DRMUMV", "DRM
#EPWORTH - Epworth Sleepness Scale
covariates["epworth"] = ["PATNO", "EVENT_ID", "ESS1", "ESS2", "ESS3", "ESS4", "ESS5", "ESS6", "ESS7", "ESS8"]
# HVLT- Hopkins Verbal Learning Test
covariates["hvlt"] = ["PATNO", "EVENT_ID", "HVLTRT1", "HVLTRT2", "HVLTRT3", "HVLTRDLY", "HVLTREC", "HVLTFPRL", "HVLTFPUN"]
# LNS - Letter Number Sequencing Test
covariates["lns"] = ["PATNO", "EVENT_ID","LNS_TOTRAW"]
# Neurological exam: Cranial Nerves Test
covariates["neuro_cranial"] = ["PATNO", "EVENT_ID","CN1RSP", "CN2RSP", "CN346RSP", "CN5RSP", "CN7RSP", "CN8RSP", "CN910RSP", "CN11RSP", "CN12RSP"
# SDM - Symbol Digit Modalities Test
covariates["sdm"] = [ "PATNO", "EVENT_ID", "SDMTOTAL"]
# Neuropsychological Test - (Line Orientation)
covariates["benton"] = ["PATNO", "EVENT_ID", "JLO_TOTRAW"]
```

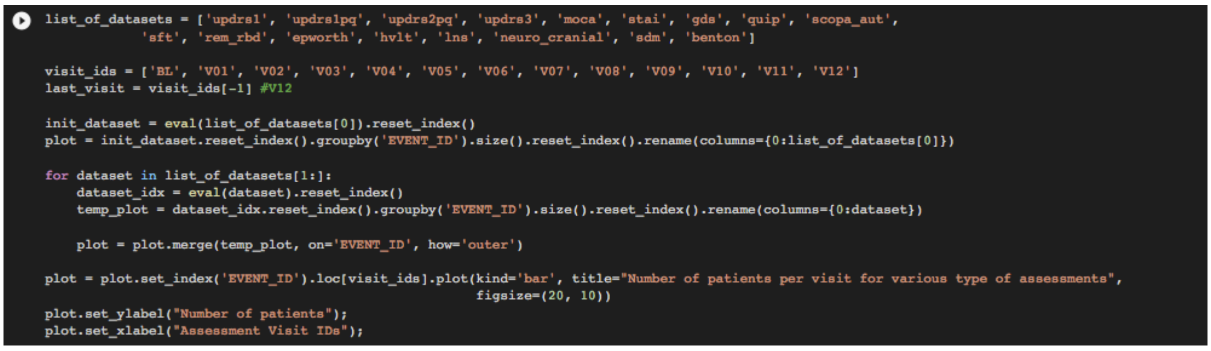


```
path=''

########## SUBJECT CHARACHTERISTIC AND MEDICAL HISTORY ##########
pat_status = pd.read_csv(path+"Patient_Status.csv", index_col=["PATNO"], usecols=covariates["pat_status"])
pat_demographics = pd.read_csv(path+"Screening___Demographics.csv", index_col=["PATNO"], usecols=covariates["pat_demographics"])
pat_family_history = pd.read_csv(path+"Family_History__PD_.csv", index_col=["PATNO"], usecols=covariates["pat_family_history"])

########## MOTOR AND NON-MOTOR ASSESSMENTS ##########
updrs1 = pd.read_csv(path+"MDS_UPDRS_Part_I.csv", index_col=["PATNO", "EVENT_ID"], parse_dates=["INFODT"], usecols=covariates["updrs1"])
updrs1pq = pd.read_csv(path+"MDS_UPDRS_Part_I__Patient_Questionnaire.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["updrs1pq"])
updrs2pq = pd.read_csv(path+"MDS_UPDRS_Part_II__Patient_Questionnaire.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["updrs2pq"])
updrs3_temp = pd.read_csv(path+"MDS_UPDRS_Part_III.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["updrs3"])
updrs3 = updrs3_temp[updrs3_temp.PAG_NAME == 'NUPDRS3'] # pre med dose
updrs3a = updrs3_temp[updrs3_temp.PAG_NAME == 'NUPDRS3A'] # post med dose
moca = pd.read_csv(path+"Montreal_Cognitive_Assessment__MoCA_.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["moca"])
stai = pd.read_csv(path+"State-Trait_Anxiety_Inventory.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["stai"])
gds = pd.read_csv(path+"Geriatric_Depression_Scale__Short_.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["gds"])
quip = pd.read_csv(path+"QUIP_Current_Short.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["quip"])
scopa_aut = pd.read_csv(path+"SCOPA-AUT.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["scopa_aut"])
sft = pd.read_csv(path+"Semantic_Fluency.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["sft"])
rem_rbd = pd.read_csv(path+"REM_Sleep_Disorder_Questionnaire.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["rem_rbd"])
epworth = pd.read_csv(path+"Epworth_Sleepiness_Scale.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["epworth"])
hvlt = pd.read_csv(path+"Hopkins_Verbal_Learning_Test.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["hvlt"])
lns = pd.read_csv(path+"Letter_-_Number_Sequencing__PD_.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["lns"])
neuro_cranial = pd.read_csv(path+"Neurological_Exam_-_Cranial_Nerves.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["neuro_cranial"])
sdm = pd.read_csv(path+"Symbol_Digit_Modalities.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["sdm"])
benton = pd.read_csv(path+"Benton_Judgment_of_Line_Orientation.csv", index_col=["PATNO", "EVENT_ID"], usecols=covariates["benton"])
```

Figure 2: Data loading and Pre-processing

3.3 Exploratory Data Analysis

Detailed analysis of fetched data is conducted to obtain better understanding on characteristic and quality of fetched data.



```python
list_of_datasets = ['updrs1', 'updrs1pq', 'updrs2pq', 'updrs3', 'moca', 'stai', 'gds', 'quip', 'scopa_aut',
                    'sft', 'rem_rbd', 'epworth', 'hvlt', 'lns', 'neuro_cranial', 'sdm', 'benton']

visit_ids = ['BL', 'V01', 'V02', 'V03', 'V04', 'V05', 'V06', 'V07', 'V08', 'V09', 'V10', 'V11', 'V12']
last_visit = visit_ids[-1] #V12

init_dataset = eval(list_of_datasets[0]).reset_index()
plot = init_dataset.reset_index().groupby('EVENT_ID').size().reset_index().rename(columns={0:list_of_datasets[0]})

for dataset in list_of_datasets[1:]:
    dataset_idx = eval(dataset).reset_index()
    temp_plot = dataset_idx.reset_index().groupby('EVENT_ID').size().reset_index().rename(columns={0:dataset})

    plot = plot.merge(temp_plot, on='EVENT_ID', how='outer')

plot = plot.set_index('EVENT_ID').loc[visit_ids].plot(kind='bar', title="Number of patients per visit for various type of assessments",
                                                        figsize=(20, 10))
plot.set_ylabel("Number of patients");
plot.set_xlabel("Assessment Visit IDs");
```

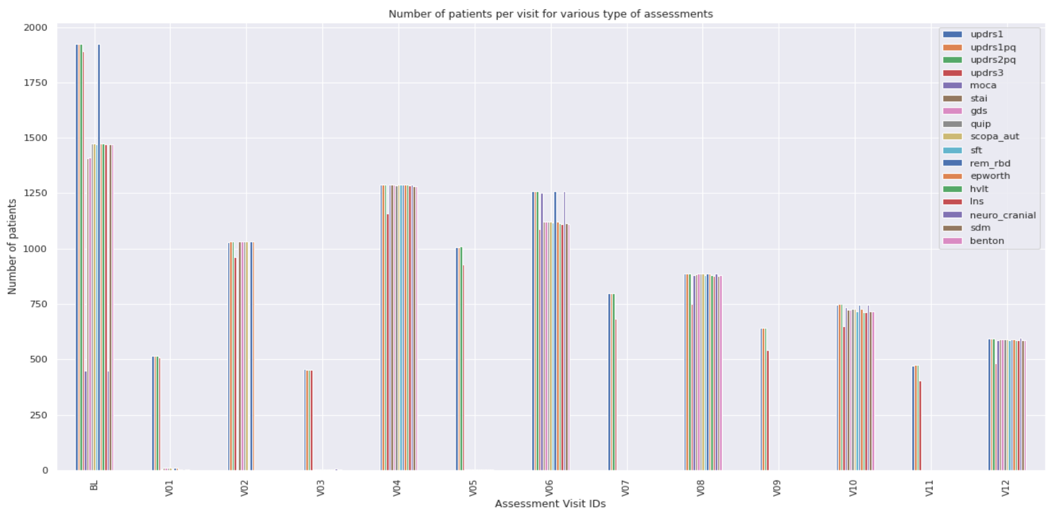
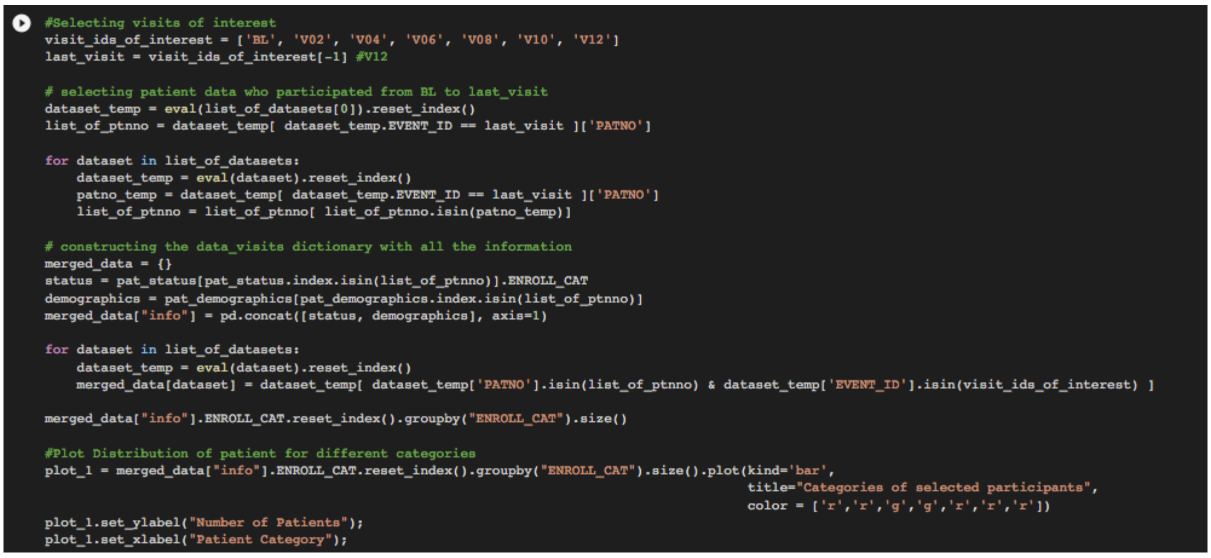


Figure 3: Process of clustering PD cases based on risk of progression

From above visualisation, it can be inferred that as we move towards last visit of assessment, steep decline in number of participants can be seen. Also, for various tests are not mandatory to be conducted during each visit. Therefore, based presented data instead of selecting each visit, specific visits with most test data is selected. That is BL, V02, V04, V06, V08, V10, V12. Records of data for all the essential covariates identified in selected list of clinical assessment is merged over unique index key defined on patient id (PATNO) and visit id (EVENT_ID) for the filtered list of patients.



```
#Selecting visits of interest
visit_ids_of_interest = ['BL', 'V02', 'V04', 'V06', 'V08', 'V10', 'V12']
last_visit = visit_ids_of_interest[-1] #V12

# selecting patient data who participated from BL to last_visit
dataset_temp = eval(list_of_datasets[0]).reset_index()
list_of_ptnno = dataset_temp[ dataset_temp.EVENT_ID == last_visit ]['PATNO']

for dataset in list_of_datasets:
    dataset_temp = eval(dataset).reset_index()
    patno_temp = dataset_temp[ dataset_temp.EVENT_ID == last_visit ]['PATNO']
    list_of_ptnno = list_of_ptnno[ list_of_ptnno.isin(patno_temp)]

# constructing the data_visits dictionary with all the information
merged_data = {}
status = pat_status[pat_status.index.isin(list_of_ptnno)].ENROLL_CAT
demographics = pat_demographics[pat_demographics.index.isin(list_of_ptnno)]
merged_data["info"] = pd.concat([status, demographics], axis=1)

for dataset in list_of_datasets:
    dataset_temp = eval(dataset).reset_index()
    merged_data[dataset] = dataset_temp[ dataset_temp['PATNO'].isin(list_of_ptnno) & dataset_temp['EVENT_ID'].isin(visit_ids_of_interest) ]

merged_data["info"].ENROLL_CAT.reset_index().groupby("ENROLL_CAT").size()

#Plot Distribution of patient for different categories
plot_1 = merged_data["info"].ENROLL_CAT.reset_index().groupby("ENROLL_CAT").size().plot(kind='bar',
                                                                                         title="Categories of selected participants",
                                                                                         color = ['r','r','g','g','r','r','r'])

plot_1.set_ylabel("Number of Patients");
plot_1.set_xlabel("Patient Category");
```

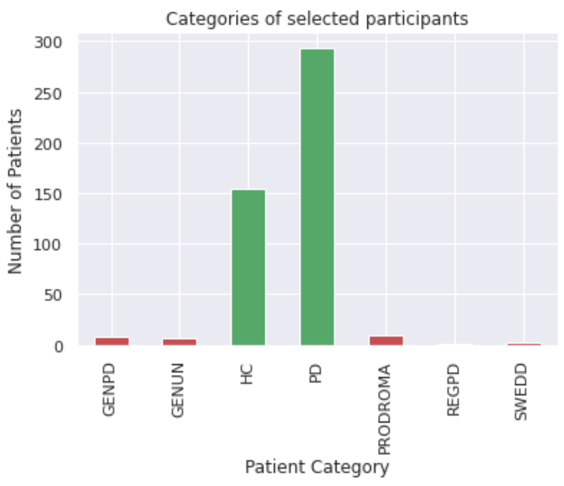


Figure 4: Process of clustering PD cases based on risk of progression

Above bar graph represents the category of participating patients. It can be inferred that most of the participants fall under Healthy Control (HC) and Parkinson's Disease (PD) category. Thus, we will consider records from these two categories for training model.

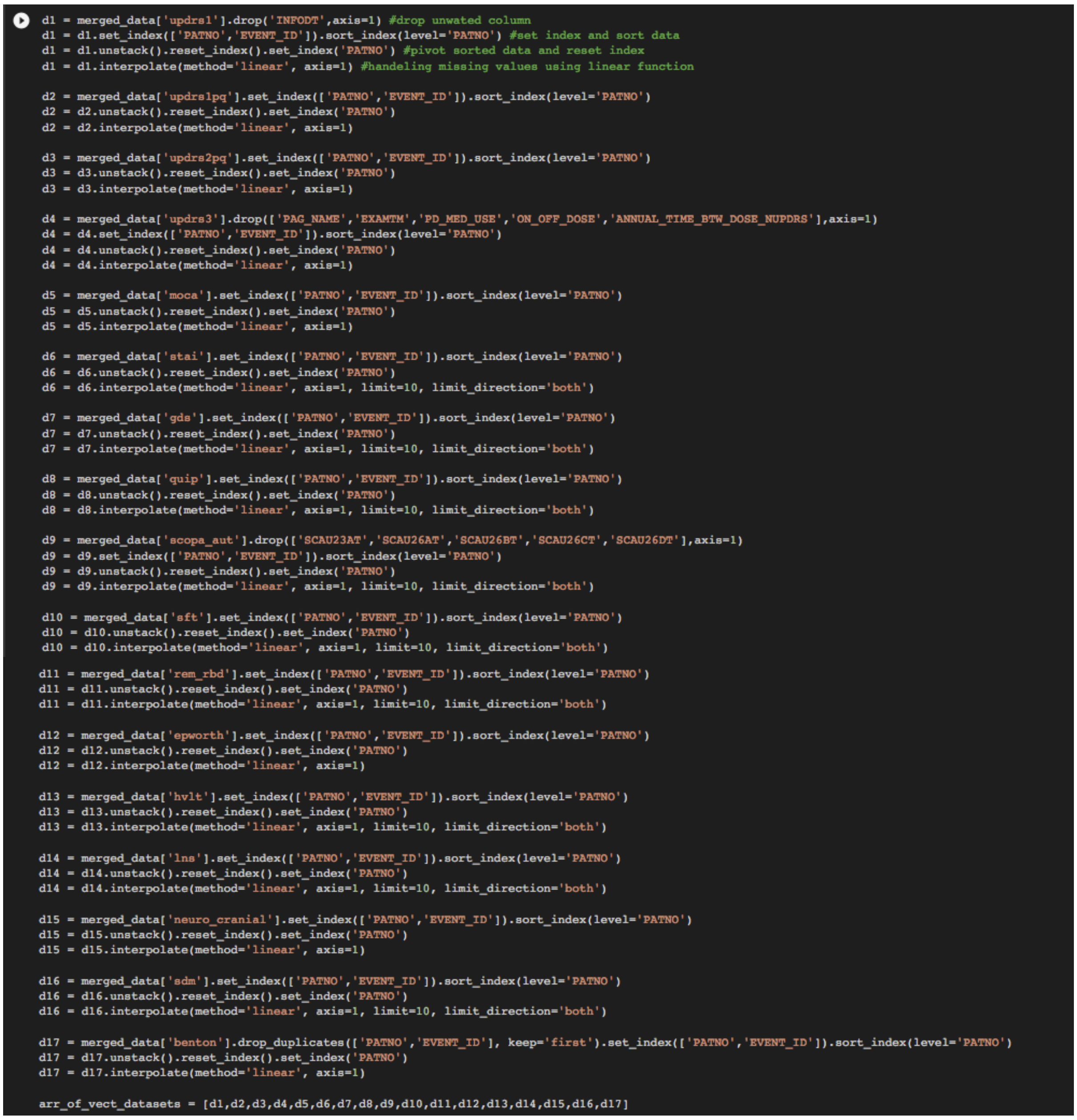
3.4 Feature Engineering

STEP 1: Time Series Vectorisation & Imputation

Below steps are performed to Vectorise time series data into one series, that is patient's data from each visit and handle missing values using ***interpolate()*** function[10]. Vectorisation technique eliminates looping hops, generating better performing model and reduces the

[10] https://scikit-learn.org/stable/modules/impute.html

computation load by 20 to 30%. In this technique data points of each covariate from every visit of patient is pivoted against its unique patient ID, creating a single row of record for each patient ID.



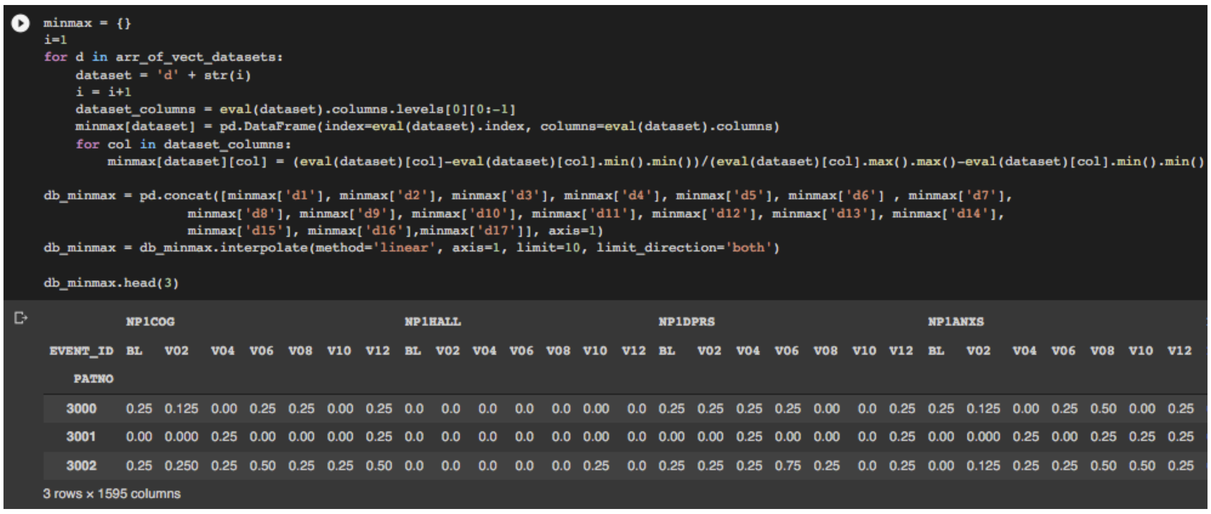
```
d1 = merged_data['updrs1'].drop('INFODT',axis=1) #drop unwated column
d1 = d1.set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO') #set index and sort data
d1 = d1.unstack().reset_index().set_index('PATNO') #pivot sorted data and reset index
d1 = d1.interpolate(method='linear', axis=1) #handeling missing values using linear function

d2 = merged_data['updrs1pq'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d2 = d2.unstack().reset_index().set_index('PATNO')
d2 = d2.interpolate(method='linear', axis=1)

d3 = merged_data['updrs2pq'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d3 = d3.unstack().reset_index().set_index('PATNO')
d3 = d3.interpolate(method='linear', axis=1)

d4 = merged_data['updrs3'].drop(['PAG_NAME','EXAMTM','PD_MED_USE','ON_OFF_DOSE','ANNUAL_TIME_BTW_DOSE_NUPDRS'],axis=1)
d4 = d4.set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d4 = d4.unstack().reset_index().set_index('PATNO')
d4 = d4.interpolate(method='linear', axis=1)

d5 = merged_data['moca'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d5 = d5.unstack().reset_index().set_index('PATNO')
d5 = d5.interpolate(method='linear', axis=1)

d6 = merged_data['stai'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d6 = d6.unstack().reset_index().set_index('PATNO')
d6 = d6.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d7 = merged_data['gds'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d7 = d7.unstack().reset_index().set_index('PATNO')
d7 = d7.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d8 = merged_data['quip'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d8 = d8.unstack().reset_index().set_index('PATNO')
d8 = d8.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d9 = merged_data['scopa_aut'].drop(['SCAU23AT','SCAU26AT','SCAU26BT','SCAU26CT','SCAU26DT'],axis=1)
d9 = d9.set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d9 = d9.unstack().reset_index().set_index('PATNO')
d9 = d9.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d10 = merged_data['sft'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d10 = d10.unstack().reset_index().set_index('PATNO')
d10 = d10.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d11 = merged_data['rem_rbd'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d11 = d11.unstack().reset_index().set_index('PATNO')
d11 = d11.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d12 = merged_data['epworth'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d12 = d12.unstack().reset_index().set_index('PATNO')
d12 = d12.interpolate(method='linear', axis=1)

d13 = merged_data['hvlt'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d13 = d13.unstack().reset_index().set_index('PATNO')
d13 = d13.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d14 = merged_data['lns'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d14 = d14.unstack().reset_index().set_index('PATNO')
d14 = d14.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d15 = merged_data['neuro_cranial'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d15 = d15.unstack().reset_index().set_index('PATNO')
d15 = d15.interpolate(method='linear', axis=1)

d16 = merged_data['sdm'].set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d16 = d16.unstack().reset_index().set_index('PATNO')
d16 = d16.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

d17 = merged_data['benton'].drop_duplicates(['PATNO','EVENT_ID'], keep='first').set_index(['PATNO','EVENT_ID']).sort_index(level='PATNO')
d17 = d17.unstack().reset_index().set_index('PATNO')
d17 = d17.interpolate(method='linear', axis=1)

arr_of_vect_datasets = [d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15,d16,d17]
```

Figure 5: Code base for vectorizing time series data and imputation

STEP 2: Normalisation\Standardisation

Normalisation\Standardisation[11] techniques are adopted to uniformly scale data points. As different covariates in dataset are measured on different scales and thus do not contribute equally during training of machine learning model, which might end up creating a bias.

[11] https://scikit-learn.org/stable/modules/preprocessing.html



```python
minmax = {}
i=1
for d in arr_of_vect_datasets:
    dataset = 'd' + str(i)
    i = i+1
    dataset_columns = eval(dataset).columns.levels[0][0:-1]
    minmax[dataset] = pd.DataFrame(index=eval(dataset).index, columns=eval(dataset).columns)
    for col in dataset_columns:
        minmax[dataset][col] = (eval(dataset)[col]-eval(dataset)[col].min().min())/(eval(dataset)[col].max().max()-eval(dataset)[col].min().min()

db_minmax = pd.concat([minmax['d1'], minmax['d2'], minmax['d3'], minmax['d4'], minmax['d5'], minmax['d6'] , minmax['d7'],
                minmax['d8'], minmax['d9'], minmax['d10'], minmax['d11'], minmax['d12'], minmax['d13'], minmax['d14'],
                minmax['d15'], minmax['d16'],minmax['d17']], axis=1)
db_minmax = db_minmax.interpolate(method='linear', axis=1, limit=10, limit_direction='both')

db_minmax.head(3)
```

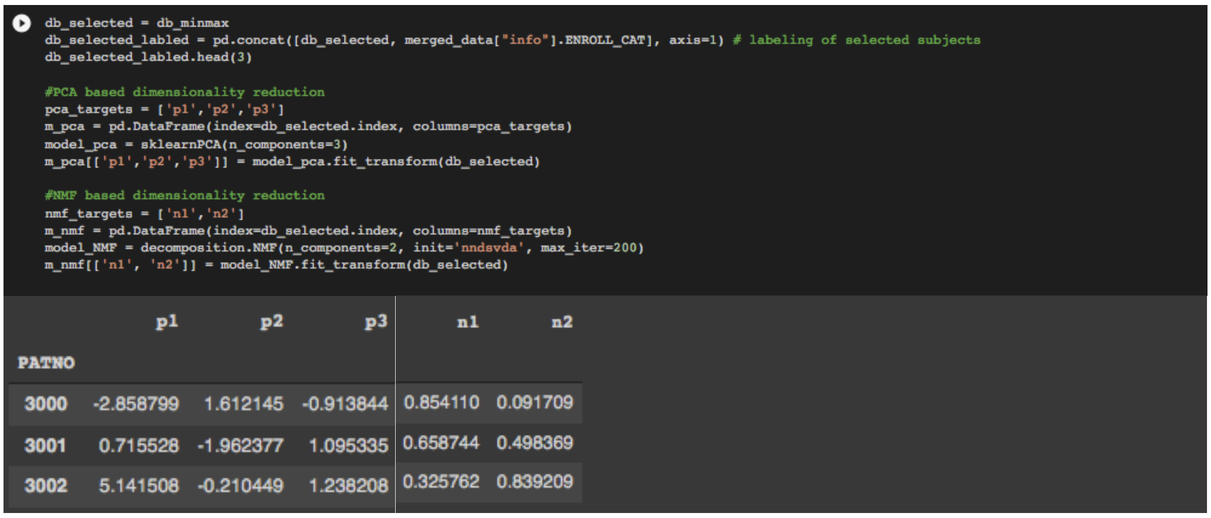
```
         NP1COG                               NP1HALL                       NP1DPRS                              NP1ANXS
EVENT_ID BL   V02   V04  V06  V08  V10  V12  BL  V02 V04 V06 V08 V10  V12  BL  V02  V04  V06  V08  V10  V12  BL   V02   V04  V06  V08  V10  V12
PATNO
3000     0.25 0.125 0.00 0.25 0.25 0.00 0.25 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.25 0.25 0.25 0.25 0.00 0.0 0.25 0.25 0.125 0.00 0.25 0.50 0.00 0.25
3001     0.00 0.000 0.25 0.00 0.00 0.00 0.25 0.0 0.0 0.0 0.0 0.0 0.00 0.0 0.00 0.00 0.25 0.00 0.00 0.0 0.25 0.00 0.000 0.25 0.00 0.25 0.25 0.25
3002     0.25 0.250 0.25 0.50 0.25 0.25 0.50 0.0 0.0 0.0 0.0 0.0 0.25 0.0 0.25 0.25 0.25 0.75 0.25 0.0 0.25 0.00 0.125 0.25 0.25 0.50 0.50 0.25
3 rows × 1595 columns
```

Figure 6: Output from Min-Max Normalization

Min-Max normalisation technique is performed to scale value of data points between 0 and 1.

3.5 Dimensionality Reduction

Dimensionality reduction[12] techniques are adopted to scale down the number of covariates, dimensionality reduction technique is applied to group correlated features into new component vectors and prevent loss of information. Two techniques are compared for the purpose of study, Principle Component Analysis (PCA) and Nonnegative Matrix Factorization (NMF). PCA which tends to group both positively as well as negatively correlated components together (Ian Goodfellow, Yoshua Bengio, 2016). Whereas, NMF tends to find patterns among variable with the same direction of correlation.

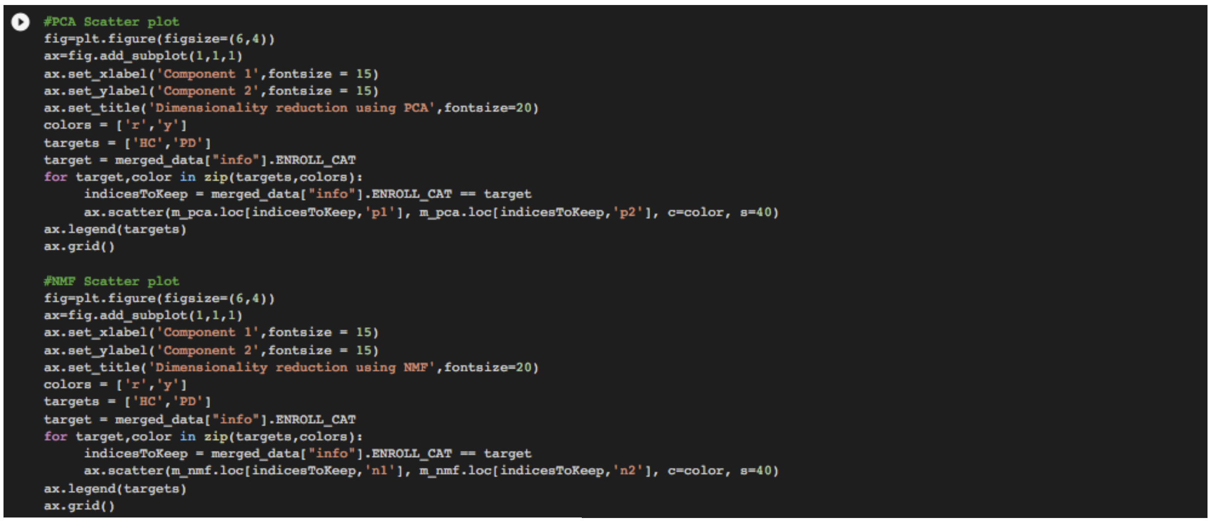


```python
db_selected = db_minmax
db_selected_labled = pd.concat([db_selected, merged_data["info"].ENROLL_CAT], axis=1) # labeling of selected subjects
db_selected_labled.head(3)

#PCA based dimensionality reduction
pca_targets = ['p1','p2','p3']
m_pca = pd.DataFrame(index=db_selected.index, columns=pca_targets)
model_pca = sklearnPCA(n_components=3)
m_pca[['p1','p2','p3']] = model_pca.fit_transform(db_selected)

#NMF based dimensionality reduction
nmf_targets = ['n1','n2']
m_nmf = pd.DataFrame(index=db_selected.index, columns=nmf_targets)
model_NMF = decomposition.NMF(n_components=2, init='nndsvda', max_iter=200)
m_nmf[['n1', 'n2']] = model_NMF.fit_transform(db_selected)
```

```
        p1        p2         p3        n1        n2
PATNO
3000   -2.858799  1.612145  -0.913844  0.854110  0.091709
3001    0.715528 -1.962377   1.095335  0.658744  0.498369
3002    5.141508 -0.210449   1.238208  0.325762  0.839209
```

Figure 7: Output from PCA and NMF Dimensionality Reduction technique

[12] https://scikit-learn.org/stable/modules/decomposition.html



```
#PCA Scatter plot
fig=plt.figure(figsize=(6,4))
ax=fig.add_subplot(1,1,1)
ax.set_xlabel('Component 1',fontsize = 15)
ax.set_ylabel('Component 2',fontsize = 15)
ax.set_title('Dimensionality reduction using PCA',fontsize=20)
colors = ['r','y']
targets = ['HC','PD']
target = merged_data["info"].ENROLL_CAT
for target,color in zip(targets,colors):
     indicesToKeep = merged_data["info"].ENROLL_CAT == target
     ax.scatter(m_pca.loc[indicesToKeep,'p1'], m_pca.loc[indicesToKeep,'p2'], c=color, s=40)
ax.legend(targets)
ax.grid()

#NMF Scatter plot
fig=plt.figure(figsize=(6,4))
ax=fig.add_subplot(1,1,1)
ax.set_xlabel('Component 1',fontsize = 15)
ax.set_ylabel('Component 2',fontsize = 15)
ax.set_title('Dimensionality reduction using NMF',fontsize=20)
colors = ['r','y']
targets = ['HC','PD']
target = merged_data["info"].ENROLL_CAT
for target,color in zip(targets,colors):
     indicesToKeep = merged_data["info"].ENROLL_CAT == target
     ax.scatter(m_nmf.loc[indicesToKeep,'n1'], m_nmf.loc[indicesToKeep,'n2'], c=color, s=40)
ax.legend(targets)
ax.grid()
```

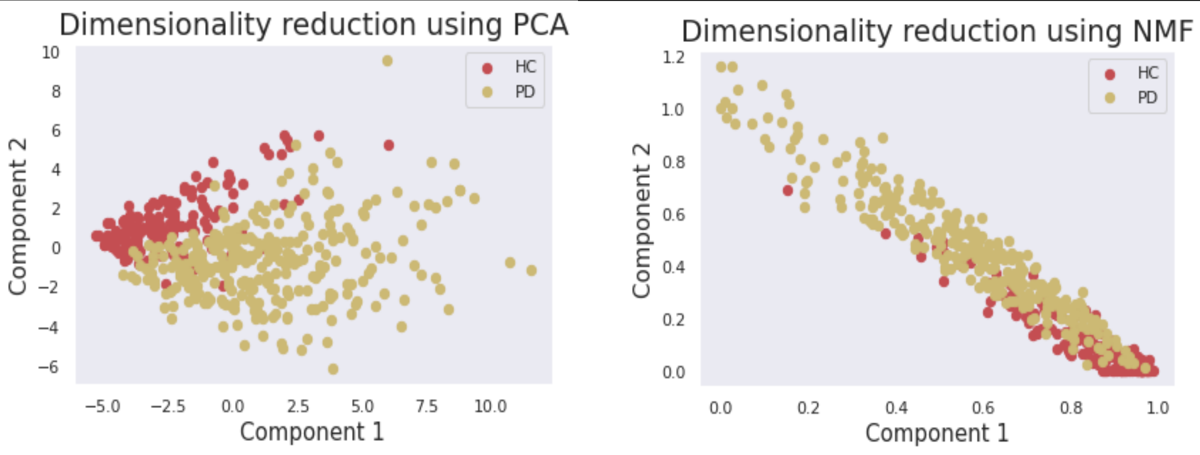
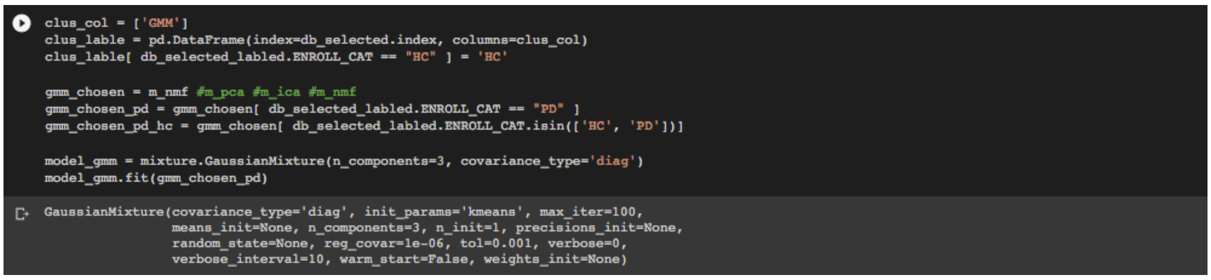


Figure 8: Scatter plot from PCA and NMF vector components

3.6 Unsupervised Clustering

Clustering based on **Gaussian Mixture Modelling (GMM)**[13] techniques generates data clusters by grouping similar data points based on their feature and correlation between the points. In proposed design data points are grouped into 3 clusters, where each cluster defines the risk of PD progression among the patient. These clusters are marked as Low, Medium and High referencing to progression rate.

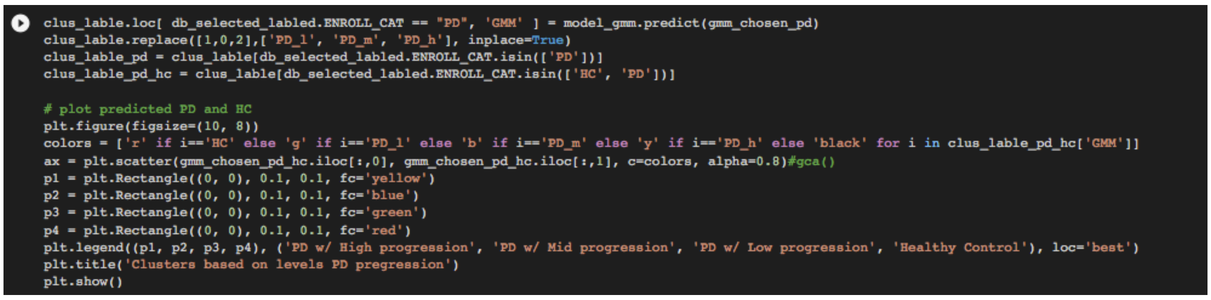


```
clus_col = ['GMM']
clus_lable = pd.DataFrame(index=db_selected.index, columns=clus_col)
clus_lable[ db_selected_labled.ENROLL_CAT == "HC" ] = 'HC'

gmm_chosen = m_nmf #m_pca #m_ica #m_nmf
gmm_chosen_pd = gmm_chosen[ db_selected_labled.ENROLL_CAT == "PD" ]
gmm_chosen_pd_hc = gmm_chosen[ db_selected_labled.ENROLL_CAT.isin(['HC', 'PD'])]

model_gmm = mixture.GaussianMixture(n_components=3, covariance_type='diag')
model_gmm.fit(gmm_chosen_pd)
```

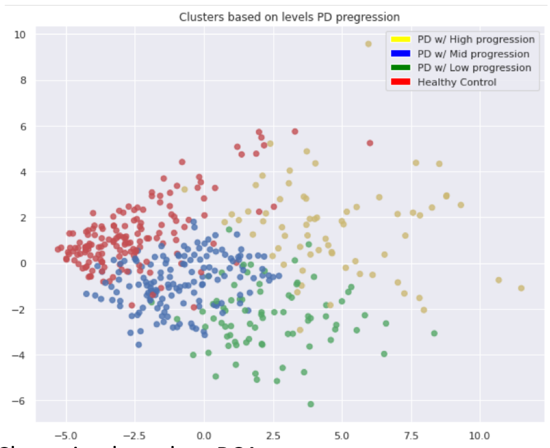
```
GaussianMixture(covariance_type='diag', init_params='kmeans', max_iter=100,
                means_init=None, n_components=3, n_init=1, precisions_init=None,
                random_state=None, reg_covar=1e-06, tol=0.001, verbose=0,
                verbose_interval=10, warm_start=False, weights_init=None)
```

Figure 9: Process of clustering PD cases based on risk of progression

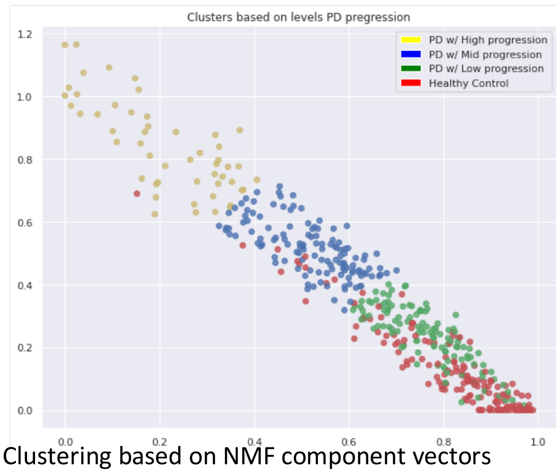
[13] https://scikit-learn.org/stable/modules/preprocessing.html



```
clus_lable.loc[ db_selected_labled.ENROLL_CAT == "PD", 'GMM' ] = model_gmm.predict(gmm_chosen_pd)
clus_lable.replace([1,0,2],['PD_l', 'PD_m', 'PD_h'], inplace=True)
clus_lable_pd = clus_lable[db_selected_labled.ENROLL_CAT.isin(['PD'])]
clus_lable_pd_hc = clus_lable[db_selected_labled.ENROLL_CAT.isin(['HC', 'PD'])]

# plot predicted PD and HC
plt.figure(figsize=(10, 8))
colors = ['r' if i=='HC' else 'g' if i=='PD_l' else 'b' if i=='PD_m' else 'y' if i=='PD_h' else 'black' for i in clus_lable_pd_hc['GMM']]
ax = plt.scatter(gmm_chosen_pd_hc.iloc[:,0], gmm_chosen_pd_hc.iloc[:,1], c=colors, alpha=0.8)#gca()
p1 = plt.Rectangle((0, 0), 0.1, 0.1, fc='yellow')
p2 = plt.Rectangle((0, 0), 0.1, 0.1, fc='blue')
p3 = plt.Rectangle((0, 0), 0.1, 0.1, fc='green')
p4 = plt.Rectangle((0, 0), 0.1, 0.1, fc='red')
plt.legend((p1, p2, p3, p4), ('PD w/ High progression', 'PD w/ Mid progression', 'PD w/ Low progression', 'Healthy Control'), loc='best')
plt.title('Clusters based on levels PD progression')
plt.show()
```



Clustering based on PCA component vectors



Clustering based on NMF component vectors

Figure 10: Process of clustering PD cases based on risk of progression

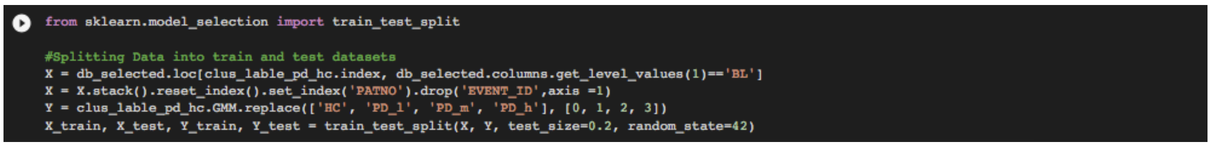
Comparative representation of resultant clusters generated by GMM model for both dimensionality reduction technique. NMF is best suited for clustering with non-negative gaussian data. PCA it is ideal for pattern recognition and dimension reduction.

3.7 Modelling and Training

For the purpose of study, a comparative study on various classification algorithm is conducted to determine the classification accuracy on this dataset. List of selected models are: Nearest Neighbours, Support Vector Machine (SVM), Decision Tree, Random Forest Classifier, AdaBoost, XGBoost, Multilayer Perceptron learning (MLP)

- **Step 1:** Split into Train/Test data

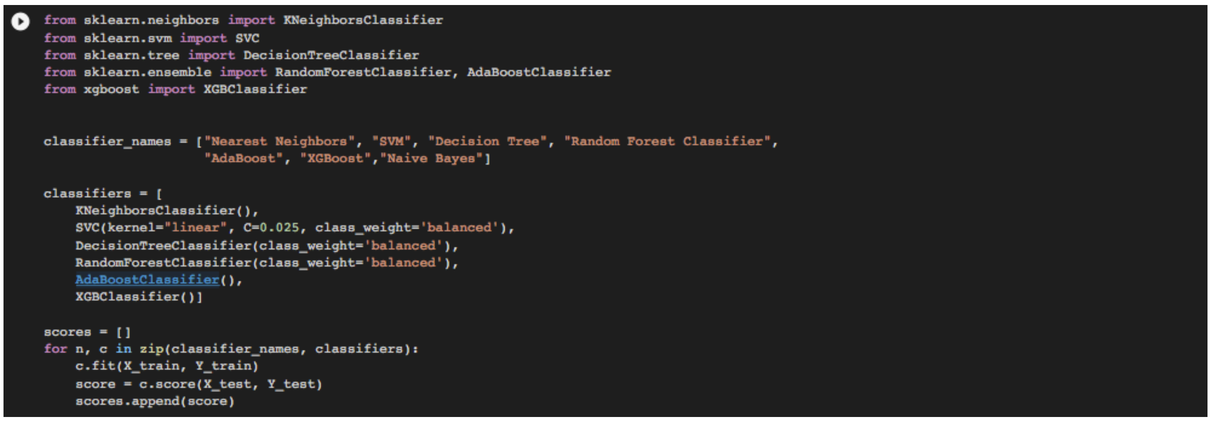
Labelled baseline characteristic dataset is split into test and train set with split ratio of 0.2, for training and validation of built models.



```
from sklearn.model_selection import train_test_split

#Splitting Data into train and test datasets
X = db_selected.loc[clus_lable_pd_hc.index, db_selected.columns.get_level_values(1)=='BL']
X = X.stack().reset_index().set_index('PATNO').drop('EVENT_ID',axis =1)
Y = clus_lable_pd_hc.GMM.replace(['HC', 'PD_l', 'PD_m', 'PD_h'], [0, 1, 2, 3])
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Figure 11: Code logic to slit into train and test set

- **Step 2:** Training Tradition Classification Models

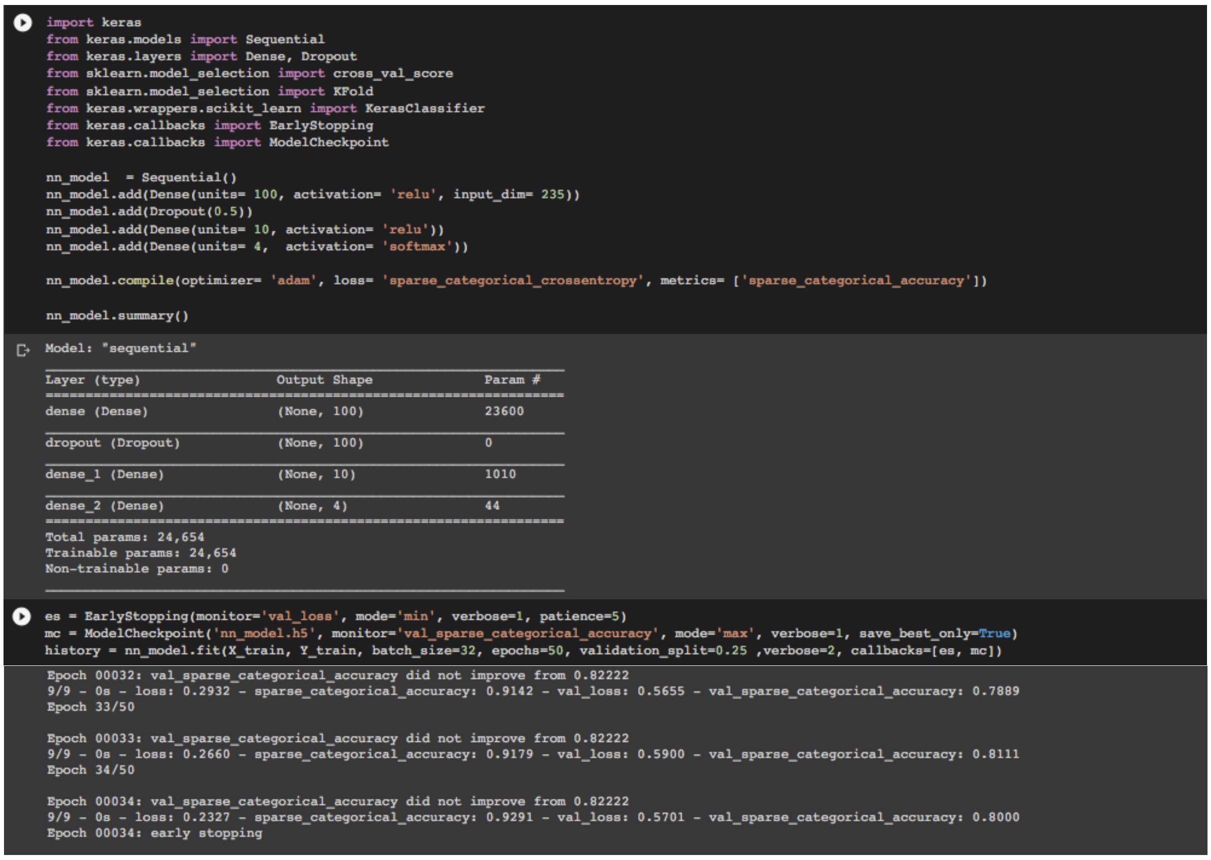


```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from xgboost import XGBClassifier


classifier_names = ["Nearest Neighbors", "SVM", "Decision Tree", "Random Forest Classifier",
                    "AdaBoost", "XGBoost","Naive Bayes"]

classifiers = [
    KNeighborsClassifier(),
    SVC(kernel="linear", C=0.025, class_weight='balanced'),
    DecisionTreeClassifier(class_weight='balanced'),
    RandomForestClassifier(class_weight='balanced'),
    AdaBoostClassifier(),
    XGBClassifier()]

scores = []
for n, c in zip(classifier_names, classifiers):
    c.fit(X_train, Y_train)
    score = c.score(X_test, Y_test)
    scores.append(score)
```

Figure 12: Code logic for training classification

- **Step 3:** Multilayer Perceptron Learning Model

After testing various structural combinations of layers and hyper-parameters, adopted state of art delivers model with best performance. In order to improve performance regularisation using Dropout layers are added and activation function ReLu and SoftMax are used in the structured model.



```python
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from keras.wrappers.scikit_learn import KerasClassifier
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

nn_model  = Sequential()
nn_model.add(Dense(units= 100, activation= 'relu', input_dim= 235))
nn_model.add(Dropout(0.5))
nn_model.add(Dense(units= 10, activation= 'relu'))
nn_model.add(Dense(units= 4,  activation= 'softmax'))

nn_model.compile(optimizer= 'adam', loss= 'sparse_categorical_crossentropy', metrics= ['sparse_categorical_accuracy'])

nn_model.summary()
```

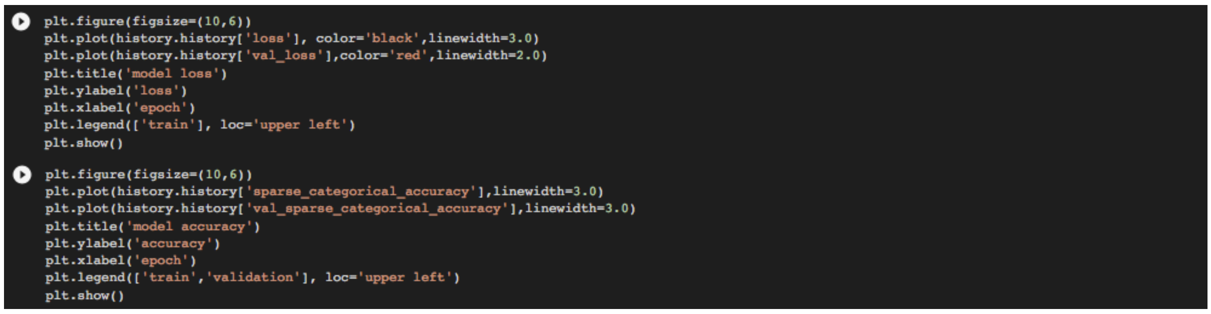
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               23600
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 44
=================================================================
Total params: 24,654
Trainable params: 24,654
Non-trainable params: 0
_________________________________________________________________
```

```python
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
mc = ModelCheckpoint('nn_model.h5', monitor='val_sparse_categorical_accuracy', mode='max', verbose=1, save_best_only=True)
history = nn_model.fit(X_train, Y_train, batch_size=32, epochs=50, validation_split=0.25 ,verbose=2, callbacks=[es, mc])
```

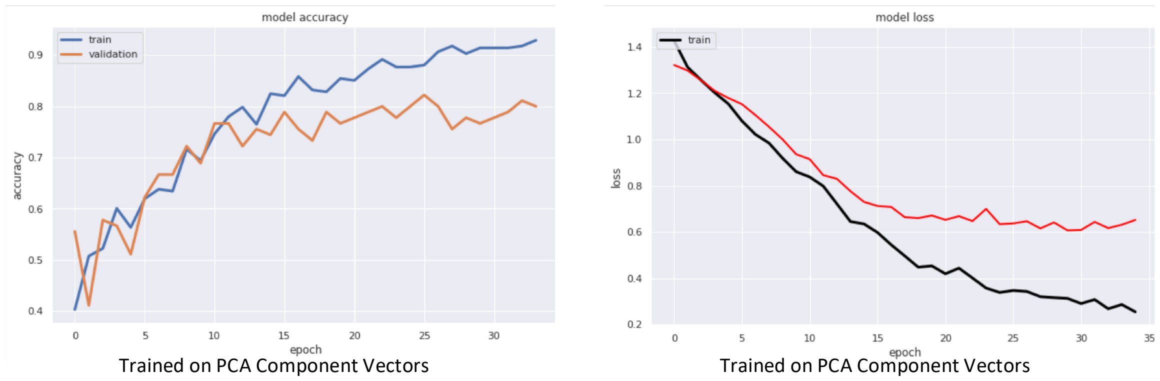
```
Epoch 00032: val_sparse_categorical_accuracy did not improve from 0.82222
9/9 - 0s - loss: 0.2932 - sparse_categorical_accuracy: 0.9142 - val_loss: 0.5655 - val_sparse_categorical_accuracy: 0.7889
Epoch 33/50

Epoch 00033: val_sparse_categorical_accuracy did not improve from 0.82222
9/9 - 0s - loss: 0.2660 - sparse_categorical_accuracy: 0.9179 - val_loss: 0.5900 - val_sparse_categorical_accuracy: 0.8111
Epoch 34/50

Epoch 00034: val_sparse_categorical_accuracy did not improve from 0.82222
9/9 - 0s - loss: 0.2327 - sparse_categorical_accuracy: 0.9291 - val_loss: 0.5701 - val_sparse_categorical_accuracy: 0.8000
Epoch 00034: early stopping
```

Figure 13: Output for deigned ML model architecture

Early stop function is assigned to training function to monitor the training rate and prevent overfitting. Model checkpoint saves model at every stage of improvement to deliver best performing model as resultant.



```
plt.figure(figsize=(10,6))
plt.plot(history.history['loss'], color='black',linewidth=3.0)
plt.plot(history.history['val_loss'],color='red',linewidth=2.0)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()

plt.figure(figsize=(10,6))
plt.plot(history.history['sparse_categorical_accuracy'],linewidth=3.0)
plt.plot(history.history['val_sparse_categorical_accuracy'],linewidth=3.0)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='upper left')
plt.show()
```



Trained on PCA Component Vectors   Trained on PCA Component Vectors

Figure 14: Learning curves for MLP model

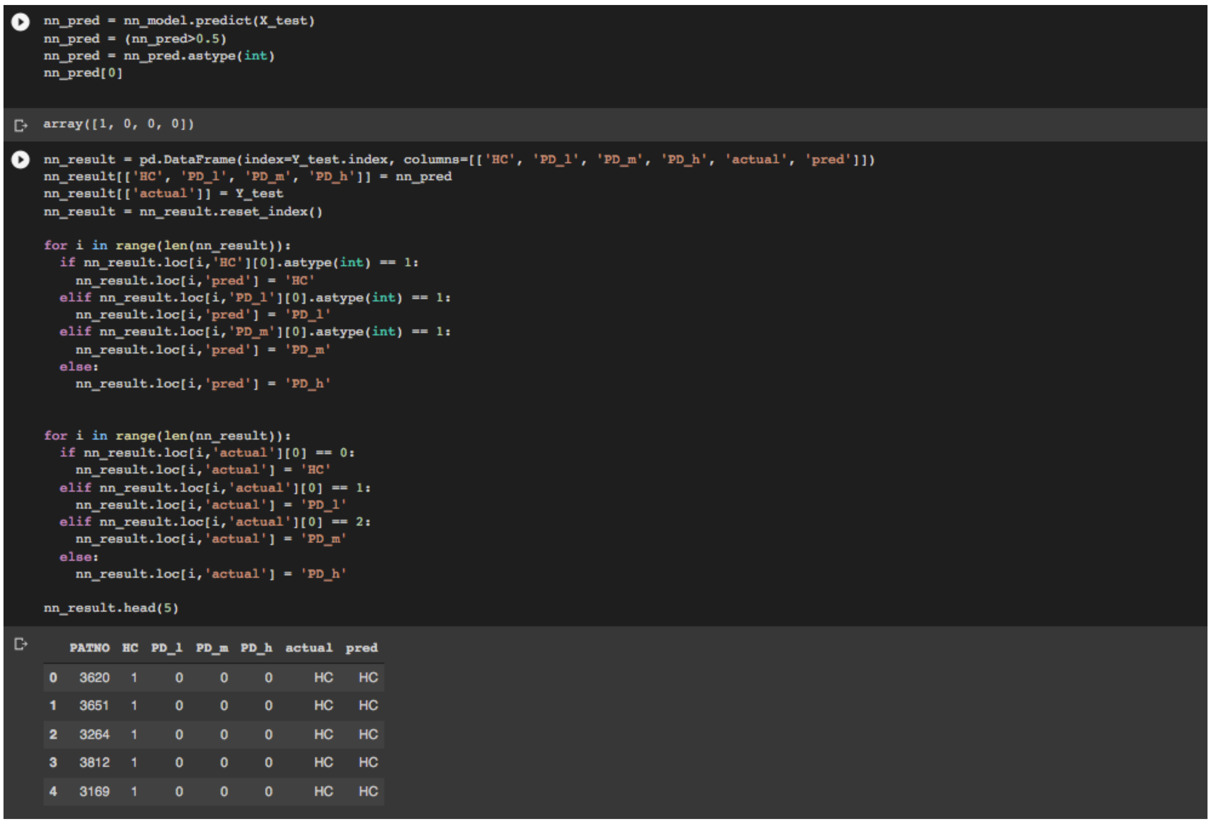
From train and validation accuracy plot it can visualise that both curves show improvement with each epochs and progress close to each other this shows model is a good fit and we are not overfitting the data. Also, with increasing number of epochs accuracy of model is also improving steadily. From train and validation losses curve, it can be visualised that training and validation loss is decreasing with each epoch.

4 Evaluation

Models are evaluated over various evaluation metrics studied from (Kuhn and Johnson, 2013):

1. **Matthews Correlation Coefficient:** MCC score measures prediction of accuracy for each class in multi class classification model.
2. **Precision:** Precision is the measure to calculate ratio of true positive to the number of predicated positive results.
3. **Recall:** Recall is the measure to calculate proportion of true positive with respect to all possible relevant data.
4. **F1-Score:** F1-Score gives the harmonic mean from precision and recall values and is also called as f1 measure. Higher the F1-score better is the model performance.

5. **Confusion Metrix:** Confusion matrix results in number true positive prediction and miss classifications made by trained model for each class in multi class classification scenario.



```
nn_pred = nn_model.predict(X_test)
nn_pred = (nn_pred>0.5)
nn_pred = nn_pred.astype(int)
nn_pred[0]
```

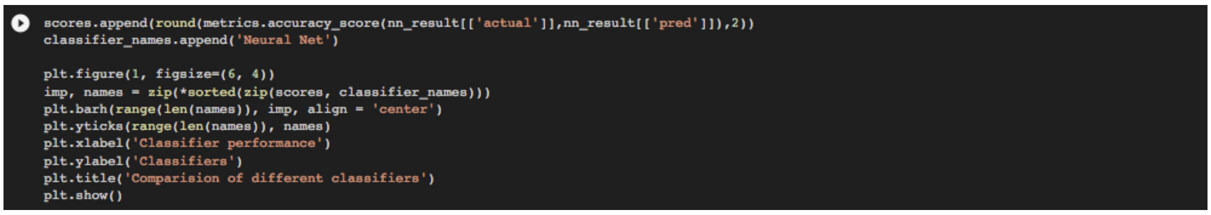
```
array([1, 0, 0, 0])
```

```
nn_result = pd.DataFrame(index=Y_test.index, columns=[['HC', 'PD_l', 'PD_m', 'PD_h', 'actual', 'pred']])
nn_result[['HC', 'PD_l', 'PD_m', 'PD_h']] = nn_pred
nn_result[['actual']] = Y_test
nn_result = nn_result.reset_index()

for i in range(len(nn_result)):
  if nn_result.loc[i,'HC'][0].astype(int) == 1:
    nn_result.loc[i,'pred'] = 'HC'
  elif nn_result.loc[i,'PD_l'][0].astype(int) == 1:
    nn_result.loc[i,'pred'] = 'PD_l'
  elif nn_result.loc[i,'PD_m'][0].astype(int) == 1:
    nn_result.loc[i,'pred'] = 'PD_m'
  else:
    nn_result.loc[i,'pred'] = 'PD_h'


for i in range(len(nn_result)):
  if nn_result.loc[i,'actual'][0] == 0:
    nn_result.loc[i,'actual'] = 'HC'
  elif nn_result.loc[i,'actual'][0] == 1:
    nn_result.loc[i,'actual'] = 'PD_l'
  elif nn_result.loc[i,'actual'][0] == 2:
    nn_result.loc[i,'actual'] = 'PD_m'
  else:
    nn_result.loc[i,'actual'] = 'PD_h'

nn_result.head(5)
```

```
   PATNO  HC  PD_l  PD_m  PD_h  actual  pred
0   3620   1     0     0     0      HC    HC
1   3651   1     0     0     0      HC    HC
2   3264   1     0     0     0      HC    HC
3   3812   1     0     0     0      HC    HC
4   3169   1     0     0     0      HC    HC
```

Figure 15: Output from prediction of Test set data using MLP model

Above code block labels predicted values against actual values for the purpose of comparison and applying evaluation metrics over MLP the model.



```
scores.append(round(metrics.accuracy_score(nn_result[['actual']],nn_result[['pred']]),2))
classifier_names.append('Neural Net')

plt.figure(1, figsize=(6, 4))
imp, names = zip(*sorted(zip(scores, classifier_names)))
plt.barh(range(len(names)), imp, align = 'center')
plt.yticks(range(len(names)), names)
plt.xlabel('Classifier performance')
plt.ylabel('Classifiers')
plt.title('Comparision of different classifiers')
plt.show()
```

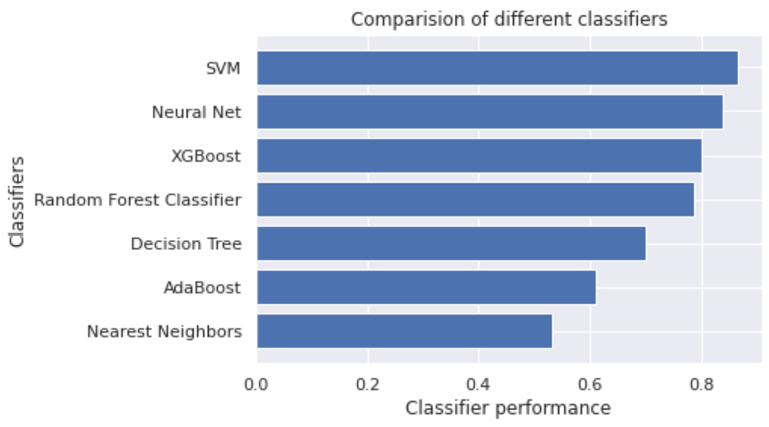
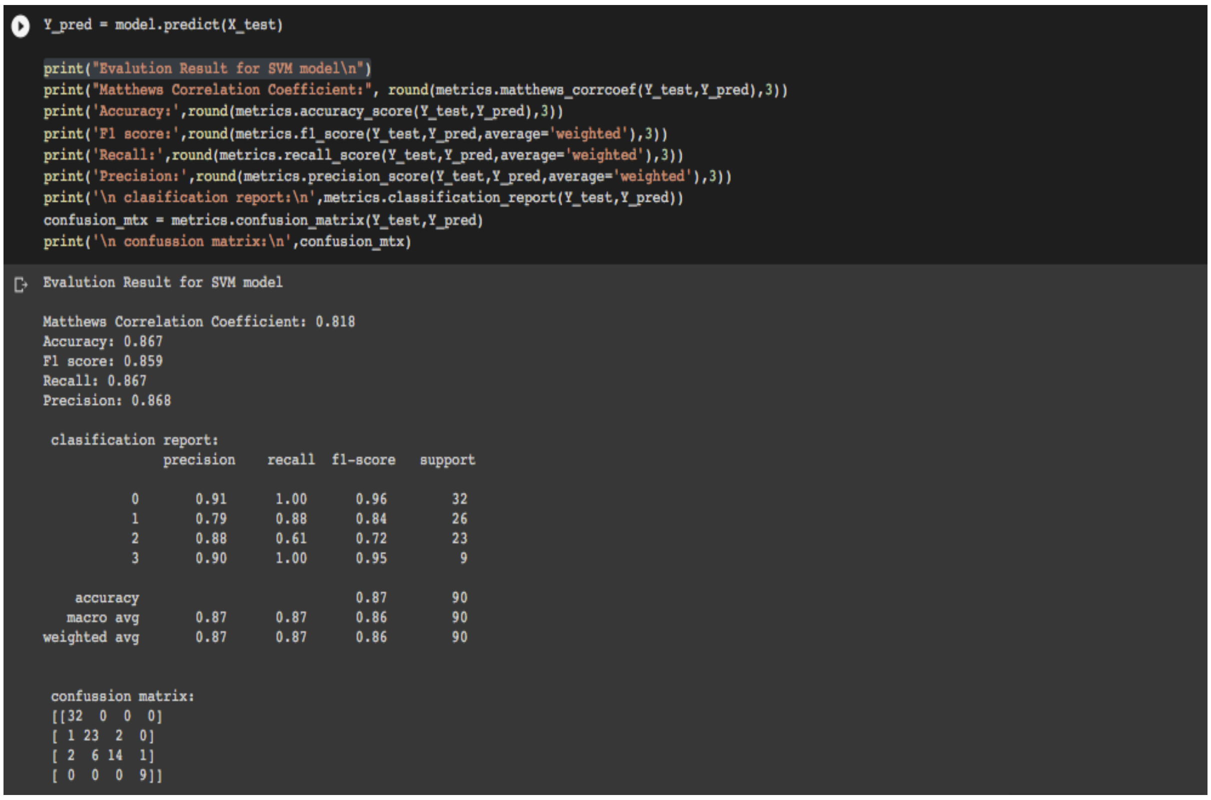


Figure 16: Performance comparison of all models

In this research, performance of 7 different models are compared. Based on thorough evaluation of model with optimal selection of hyper parameters. Best performing model identified is SVM model with classification prediction accuracy of 87%.

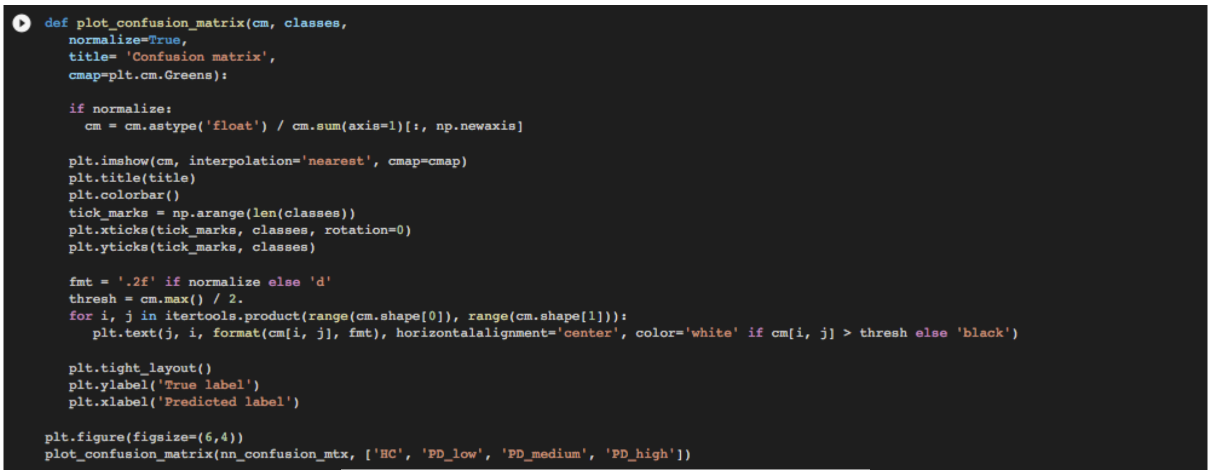


```
Y_pred = model.predict(X_test)

print("Evalution Result for SVM model\n")
print("Matthews Correlation Coefficient:", round(metrics.matthews_corrcoef(Y_test,Y_pred),3))
print('Accuracy:',round(metrics.accuracy_score(Y_test,Y_pred),3))
print('F1 score:',round(metrics.f1_score(Y_test,Y_pred,average='weighted'),3))
print('Recall:',round(metrics.recall_score(Y_test,Y_pred,average='weighted'),3))
print('Precision:',round(metrics.precision_score(Y_test,Y_pred,average='weighted'),3))
print('\n clasification report:\n',metrics.classification_report(Y_test,Y_pred))
confusion_mtx = metrics.confusion_matrix(Y_test,Y_pred)
print('\n confussion matrix:\n',confusion_mtx)
```

```
Evalution Result for SVM model

Matthews Correlation Coefficient: 0.818
Accuracy: 0.867
F1 score: 0.859
Recall: 0.867
Precision: 0.868

 clasification report:
               precision    recall  f1-score   support

           0       0.91      1.00      0.96        32
           1       0.79      0.88      0.84        26
           2       0.88      0.61      0.72        23
           3       0.90      1.00      0.95         9

    accuracy                           0.87        90
   macro avg       0.87      0.87      0.86        90
weighted avg       0.87      0.87      0.86        90


 confussion matrix:
 [[32  0  0  0]
 [ 1 23  2  0]
 [ 2  6 14  1]
 [ 0  0  0  9]]
```

Figure 17: Result of various evaluation metrics from SVM model

Results from various evaluation metrics states that SVM model is capable to classify class of PD progression with 87% accuracy. Also, with good precision and recall score, MCC score of 0.82 state that model has good prediction accuracy for each individual class.



```
def plot_confusion_matrix(cm, classes,
   normalize=True,
   title= 'Confusion matrix',
   cmap=plt.cm.Greens):

   if normalize:
     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

   plt.imshow(cm, interpolation='nearest', cmap=cmap)
   plt.title(title)
   plt.colorbar()
   tick_marks = np.arange(len(classes))
   plt.xticks(tick_marks, classes, rotation=0)
   plt.yticks(tick_marks, classes)

   fmt = '.2f' if normalize else 'd'
   thresh = cm.max() / 2.
   for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
      plt.text(j, i, format(cm[i, j], fmt), horizontalalignment='center', color='white' if cm[i, j] > thresh else 'black')

   plt.tight_layout()
   plt.ylabel('True label')
   plt.xlabel('Predicted label')

plt.figure(figsize=(6,4))
plot_confusion_matrix(nn_confusion_mtx, ['HC', 'PD_low', 'PD_medium', 'PD_high'])
```

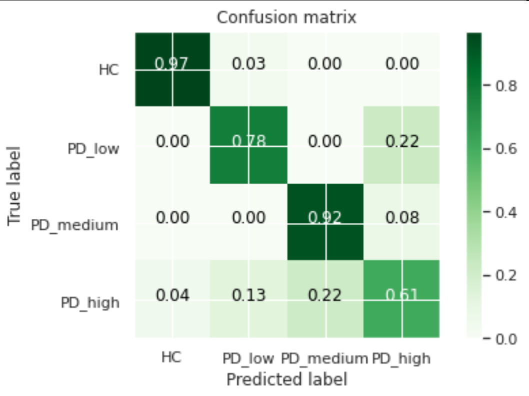


Figure 18: Confusion Matrix plot for test set prediction by SVM model

Above matrix plot is a confusion matrix. Plotted to visually understand the true positive classification and miss classifications in each class.

5 References

Ian Goodfellow, Yoshua Bengio, A. C. (2016) *Deep Learning - Ian Goodfellow, Yoshua Bengio, Aaron Courville - Google Books*, *MIT Press*. Available at: https://books.google.com/books?hl=fa&lr=&id=omivDQAAQBAJ&oi=fnd&pg=PR5&dq=deep+learning&ots=MMV1gumDRV&sig=28lNwSUYLNWXXQhzxqzjPZOZg3s#v=onepage&q&f=true%0Ahttps://books.google.co.uk/books?hl=en&lr=&id=omivDQAAQBAJ&oi=fnd&pg=PR5&dq=deep+learning+goodfello.

Kuhn, M. and Johnson, K. (2013) *Applied predictive modeling*, *Applied Predictive Modeling*. doi: 10.1007/978-1-4614-6849-3.