National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shveta Srivastava
Student ID: 18194851

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shveta Srivastava
Student ID:	18194851
Programme:	Data Analytics
Year:	2020-2021
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	16-08-2021
Project Title:	Configuration Manual
Word Count:	1189
Page Count:	20

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:	16th August 2021

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):	

Genetic Algorithm Optimized Deep Learning Model for Parkinson's Disease Severity Detection

Shveta Srivastava
18194851

1 Introduction

This configuration manual contains the information for complete implementation of the research project including hardware and software used, so as to replicate the work done at any time. This step-by-step manual will aid the users to understand the code and its implementation.

2 Hardware Specification

The project was carried out on a Dell G3 15 laptop using the configuration shown in Figure 1 and Figure 2.

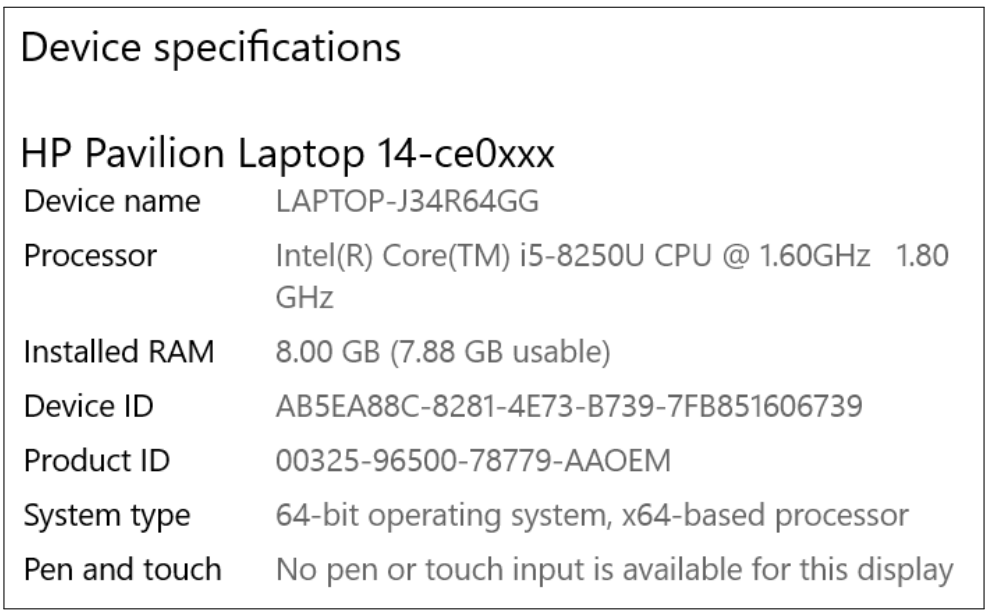

Figure 1: Device Specifications

3 Software Requirement and Environment Setup

The research work was implemented using R and Python. The process of setting up R environment and Python environment are shown below.

OS Name	Microsoft Windows 10 Home
Version	10.0.19043 Build 19043
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	LAPTOP-J34R64GG

Figure 2: System Specifications

3.1 R

R is a software environment that is free for use for building machine learning tools, graphical tools and statistical computing (Matloff; 2011). R (Version 4.1.1) can be installed using Cran R website,[1].

3.2 Anaconda for Python

Anaconda is the data science toolkit mainly for python programming (Van Rossum et al.; 2000). For python, Anaconda was installed from the website for Windows[2]. Once Anaconda is installed successfully, Jupyter Notebook can be launched and used for Python Programming as shown in Figure 3.

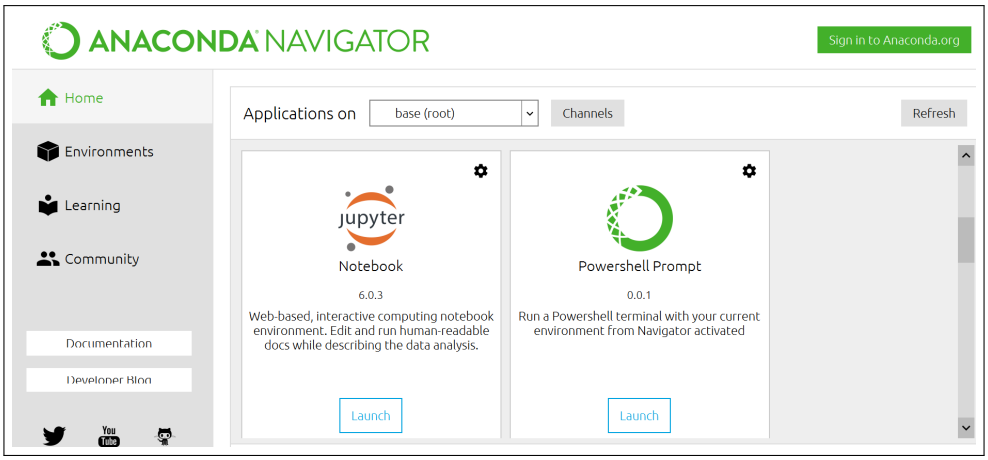

Figure 3: Launch Jupyter Notebook from Anaconda Navigator

3.3 Tensor Flow

Tensorflow environment can be installed by using the CMD.EXE prompt following the instructions given at Anaconda online user guide [3]. Abadi et al. (2016) describes several tensorflow applications for machine learning algorithm in detail.

[1] https://cran.r-project.org/bin/windows/base/
[2] https://www.anaconda.com/products/individual
[3] https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/

3.4 Google Colaboratory

Google Colaboratory, also called 'Google Colab' is a free cloud service which offers Jupyter Notebook for running python by selecting GPU and TPU setup within Colab (Carneiro et al.; 2018). Dataset can be retrieved from Google Drive by mounting it in Colab Notebook.

4 Data Collection and Merging

Parkinson's Disease patient's motor and non-motor assessment data has been collected from PPMI (Parkinson's Progressive Markers Initiative) on approval of request from the PPMI website [4].

The merging and data transformation has been done using R programming. Downloaded all the motor and non-motor assessments data and uploaded on R environment.

4.1 Read and Understand Datasets

The following datasets have been downloaded from the website. The structure of the datasets have been visualized and understood the important features from each of the dataset, using code dictionary given at PPMI.

```
#read the datasets

UPDRS1 <- read.csv("MDS_UPDRS_Part_I.csv", header = T)
head(UPDRS1)
str(UPDRS1)

UPDRSII <- read.csv("MDS_UPDRS_Part_II.csv", header = T)
str(UPDRSII)

UPDRSIII <- read.csv("MDS_UPDRS_Part_III.csv", header = T)
str(UPDRSIII)
head(UPDRSIII)

SCOPA <- read.csv("SCOPA-AUT.csv", header = T)
str(SCOPA)

MOCA <- read.csv("Montreal_Cognitive_Assessment__MoCA.csv", header = T)
str(MOCA)

BOSTON <- read.csv("Modified_Boston_Naming_Test.csv", header = T)
str(BOSTON)

STAI <- read.csv("State-Trait_Anxiety_Inventory.csv", header= T)
str(STAI)

QUIP <- read.csv("QUIP_Impulsive Compulsive Disorder.csv", header= T)
str(QUIP)

GDS <- read.csv("Geriatric_Depression_Scale.csv", header= T)
str(GDS)

```

[4] https://www.ppmi-info.org/

```
HVLT <- read.csv("Hopkins_Verbal_Learning_Test.csv", header = T)
str(HVLT)

LNS <- read.csv("Letter-Number_Sequencing.csv", header = T)
str(LNS)

SFT <- read.csv("Semantic_Fluency.csv", header = T)
str(SFT)

SDM <- read.csv("Symbol_Digit_Modalities.csv", header = T)
str(SDM)

CLOCK <- read.csv("Clock_Drawing.csv", header = T)
str(CLOCK)

COGNITIVE <- read.csv("Cognitive_Categorization.csv", header = T)
str(COGNITIVE)

LFT <- read.csv("Lexical_Fluency.csv", header = T)
str(LFT)

TRAIL <- read.csv("Trail_Making_A_and_B.csv", header = T)
str(TRAIL)

EPWORTH <- read.csv("Epworth_Sleepiness_Scale.csv", header = T)
str(EPWORTH)

REM <- read.csv("REM_Sleep_Disorder_Questionnaire.csv", header = T)
str(REM)
```

Listing 1: Reading and Understanding Datasets

The list of several motor and non-motor assessment datasets used in this study can be seen in R environment as shown in Figure 4 with the number of variables and the number of patient records in the PPMI database. It should be noted that LFT, BOSTON, CLOCK and TRAIL have been excluded from study due to very few number of patient records which will not add any value to the analysis and might hinder the model building process.

The target variable 'HY' severity scale scores are contained in 'UPDRS III' dataset.

4.2 Select important variables

Next, the important variables from all the datasets have been selected using 'Dplyr' function.

```

library(dplyr)
N_UPDRS1 <- UPDRS1 %>%
  select(PATNO, EVENT_ID, NP1COG, NP1HALL,NP1DPRS, NP1ANXS, NP1APAT,
   NP1DDS)
head(N_UPDRS1)

N_UPDRSII <- UPDRSII %>%
  select(PATNO, EVENT_ID, NP2SPCH, NP2SALV, NP2SWAL, NP2EAT, NP2DRES,
   NP2HYGN, NP2HWRT, NP2HOBB,
         NP2TURN, NP2TRMR, NP2RISE, NP2WALK, NP2FREZ)

```

BOSTON	438 obs. of 13 variables
CLOCK	332 obs. of 18 variables
COGNITIVE	7030 obs. of 18 variables
EPWORTH	8227 obs. of 19 variables
GDS	8546 obs. of 25 variables
HVLT	6999 obs. of 23 variables
LFT	443 obs. of 13 variables
LNS	6969 obs. of 34 variables
MOCA	7860 obs. of 38 variables
QUIP	8229 obs. of 24 variables
REM	8836 obs. of 32 variables
SCOPA	8229 obs. of 46 variables
SDM	6986 obs. of 15 variables
SFT	6995 obs. of 16 variables
STAI	8535 obs. of 50 variables
TRAIL	436 obs. of 16 variables
UPDRS1	13904 obs. of 17 variables
UPDRSII	13908 obs. of 24 variables
UPDRSIII	15843 obs. of 54 variables

Figure 4: List of Datasets from PPMI

```
N_UPDRSIII <- UPDRSIII %>%
  select(PATNO, EVENT_ID, NP3SPCH, NP3FACXP, NP3RIGN, NP3RIGRU,
    NP3RIGLU, PN3RIGRL,NP3RIGLL, NP3FTAPR,
          NP3FTAPL,NP3HMOVR, NP3HMOVL, NP3PRSPR,NP3PRSPL,NP3TTAPR,
    NP3TTAPL,NP3LGAGL,
          NP3LGAGR,NP3RISNG,
          NP3GAIT, NP3FRZGT, NP3PSTBL, NP3POSTR, NP3BRADY, NP3PTRMR,
    NP3PTRML, NP3KTRMR, NP3KTRML, NP3RTARU,
          NP3RTALU, NP3RTARL, NP3RTALL,NP3RTALJ, NP3RTCON, DYSKPRES, NHY
    )

N_SCOPA <- SCOPA %>%
  select(PATNO, EVENT_ID, SCAU1, SCAU2, SCAU3, SCAU4, SCAU5, SCAU6,
    SCAU7, SCAU8,
          SCAU9, SCAU10, SCAU11, SCAU12, SCAU13, SCAU14, SCAU15, SCAU16,
     SCAU17,
          SCAU18, SCAU19, SCAU20, SCAU21, SCAU24, SCAU26A, SCAU26B,
    SCAU26C, SCAU26D)
str(N_SCOPA)

N_MOCA <- MOCA %>%
  select(PATNO, EVENT_ID, MCAALTTM, MCACUBE, MCACLCKC, MCACLCKN,
    MCACLCKH, MCALION, MCARHINO,
MCACAMEL, MCAFDS, MCABDS, MCAVIGIL, MCASER7, MCASNTNC, MCAVFNUM, MCAVF,
     MCAABSTR, MCAREC1,
```

```
MCAREC2, MCAREC3, MCAREC4, MCAREC5, MCADATE, MCAMONTH, MCAYR, MCADAY,
    MCAPLACE, MCACITY, MCATOT)

N_REM <- REM %>%
  select(PATNO, EVENT_ID, DRMVIVID, DRMAGRAC, DRMNOCTB, SLPLMBMV,
    SLPINJUR, DRMVERBL, DRMFIGHT,
         DRMUMV, DRMOBJFL, MVAWAKEN, DRMREMEM, SLPDSTRB, STROKE, HETRA,
     RLS, NARCLPSY, DEPRS,
         EPILEPSY, BRNINFM)

head(N_REM)

N_GDS <- GDS %>%
  select(PATNO, EVENT_ID, GDSSATIS, GDSDROPD, GDSEMPTY, GDSBORED,
    GDSGSPIR, GDSAFRAD, GDSHAPPY,
GDSHLPLS, GDSHOME, GDSMEMRY, GDSALIVE, GDSWRTLS, GDSENRGY, GDSHOPLS,
    GDSBETER)

N_STAI <- STAI %>%
  select(PATNO, EVENT_ID, STAIAD1, STAIAD2, STAIAD3, STAIAD4, STAIAD5,
    STAIAD6, STAIAD7, STAIAD8,
         STAIAD9, STAIAD10, STAIAD11, STAIAD12, STAIAD13, STAIAD14,
    STAIAD15, STAIAD16, STAIAD17,
         STAIAD18, STAIAD19, STAIAD20, STAIAD21, STAIAD22, STAIAD23,
    STAIAD24, STAIAD25, STAIAD26,
         STAIAD27, STAIAD28, STAIAD29, STAIAD30, STAIAD31, STAIAD32,
    STAIAD33, STAIAD34, STAIAD35,
         STAIAD36, STAIAD37, STAIAD38, STAIAD39, STAIAD40)

N_QUIP <- QUIP %>%
  select(PATNO, EVENT_ID,TMGAMBLE, CNTRLGMB, TMSEX, CNTRLSEX, TMBUY,
    CNTRLBUY, TMEAT, CNTRLEAT,
         TMTORACT, TMTMTACT, TMTRWD)

N_EPWORTH <- EPWORTH %>%
  select(PATNO, EVENT_ID, ESS1, ESS2, ESS3, ESS4, ESS5, ESS6, ESS7,
    ESS8)

N_SFT <- SFT %>%
  select(PATNO, EVENT_ID,  VLTANIM, VLTVEG, VLTFRUIT)
str(N_SFT)
library(Amelia)
missmap(N_SFT)

N_HVLT <- HVLT %>%
  select(PATNO, EVENT_ID, HVLTRT1, HVLTRT2, HVLTRT3, HVLTRDLY, HVLTREC,
    HVLTFPRL, HVLTFPUN)
missmap(N_HVLT)

N_LNS <- LNS %>%
  select(PATNO, EVENT_ID, LNS_TOTRAW)

N_SDM <- SDM%>%
  select(PATNO, EVENT_ID, SDMTOTAL)
```

Listing 2: Selecting Variables from each dataset

4.3 Merge datasets

The datasets have been merged together by unique patient number 'PATNO' and their visit ID ' EVENT_ID'. The Datasets with very few records have been avoided. The final merged dataset has been saved in the google drive for further retrieval in Colab notebook for further analysis.

```
#merge
D1 <- merge(N_UPDRS1,N_UPDRSIII, by= c("PATNO", "EVENT_ID"))
str(D1)

D2 <- merge(D1,N_UPDRSII, by= c("PATNO", "EVENT_ID"))
str(D2)

D3 <- merge(D2,N_REM, by= c("PATNO", "EVENT_ID"))
str(D3)

D4 <- merge(D3,N_GDS, by= c("PATNO", "EVENT_ID"))
str(D4)

D5 <- merge(D4,N_STAI, by= c("PATNO", "EVENT_ID"))
str(D5)

D7 <- merge(D5,N_QUIP, by= c("PATNO", "EVENT_ID"))

D8 <- merge(D7,N_EPWORTH, by= c("PATNO", "EVENT_ID"))
str(D8)

D9 <- merge(D8,N_SCOPA, by= c("PATNO", "EVENT_ID"))


#exclude MOCA, LNS, HVLT, SDM, SFT
#as the number of rows reduced drastically

#D9 <- merge(D8,N_MOCA, by= c("PATNO", "EVENT_ID"))
#str(D9)

#D10 <- merge(D9,N_LNS, by= c("PATNO", "EVENT_ID"))
```

Listing 3: Merging datasets

4.4 Coding the Events

The table of the number of total visits from participants can be visualized as shown in figure 5. 'Event_ID' of 'PW', 'ST', 'U01', 'V01', 'V03', 'V05' have been excluded from study due to insufficient data. The rest are coded numerical values for analysis.

BL	PW	ST	U01	V01	V02	V03	V04	V05	V06	V08	V10	V12	V13	V14	V15	V16
1558	10	201	2	12	1029	2	1535	2	1399	1135	985	818	489	524	262	42

Figure 5: List of Datasets from PPMI

```
Final_Data <- D9
str(Final_Data)

```

```
#FIND DEPENDENT VARIABLE 'NHY'

table(Final_Data$NHY)
table(Final_Data$EVENT_ID)

# select visits that are of interest for modeling

#encoding of visits from baseline till visit 14
Final_Data$EVENT_ID <- ifelse(Final_Data$EVENT_ID == 'BL',1,
                              ifelse(Final_Data$EVENT_ID == 'V02', 2,
                                     ifelse(Final_Data$EVENT_ID == 'V04',3,
                                            ifelse(Final_Data$EVENT_ID == 'V06',4,
                                                   ifelse(Final_Data$EVENT_ID == 'V08',5,
                                                          ifelse(Final_Data$EVENT_ID == 'V10',6,
                                                                 ifelse(Final_Data$EVENT_ID == 'V12',7,
                                                                        ifelse(Final_Data$EVENT_ID == 'V13',8, 9)
                                                                 ))))))))


write.csv(Final_Data, file = "Final_Data.csv")
```

Listing 4: Coding Visits of Interest

5 EDA and Pre-processing

5.1 Load the data in Google Colab

Rest of the analysis is done using Python on Jupyter Notebook in Google Colab. The data is retrieved from Google Drive which has to be mounted on Colab using authentication code. Then the libraries are imported followed by uploading the 'Final Data' from the data path.

```
from google.colab import drive
drive.mount('/content/drive/')

import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split,cross_val_score

data_path = '/content/drive/MyDrive/Final_Data.csv'
data = pd.read_csv(data_path)
```

Listing 5: Read the Data

5.2 Exploratory Data Analysis

The dataset is visualized initially using the following steps.

```
data.head()
data.shape # Dataset has 9472 rows and 175 features
data.dtypes # all the variables have either integer or float values
data.describe() # describes the count of variables, mean, std.
    deviation, range of values with min, max values.
sns.catplot(x='NHY',kind='count',data=data) # The severity scale based
    on 'HY' index is from '1' to '5' indicating various stages of
    Parkinson's disease.
sns.catplot(x='EVENT_ID',kind='count',data=data) # Number of patients
    participating by visits. number '9' is the final visit. It can be
    observed that there is a decline in visits.
print(data['PATNO'].value_counts())
n = len(pd.unique(data['PATNO']))
print("Number of Patients participating in assessments :",
      n) #Number of Patients participating in assessments : 1478
```

Listing 6: Exploratory Data Analysis

The following figure 6 and figure 7 shows the plot of the target variable 'NHY' and 'EVENTS' respectively.

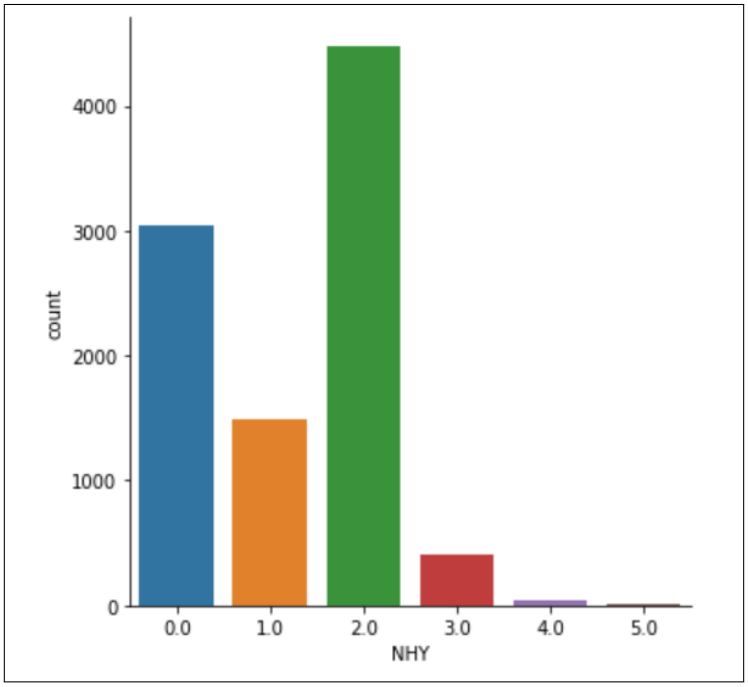

Figure 6: Different stages of Disease Severity

5.3 Handling Missing Records

Missing variables can be visualized using heatmap. Missing records has been replaced with previous assessment score for that patient's visits.

```
# plot the missmap to visualize the columns with missing records
sns.heatmap(data.isnull(), cbar=False, yticklabels=False, cmap='viridis
    ')
```

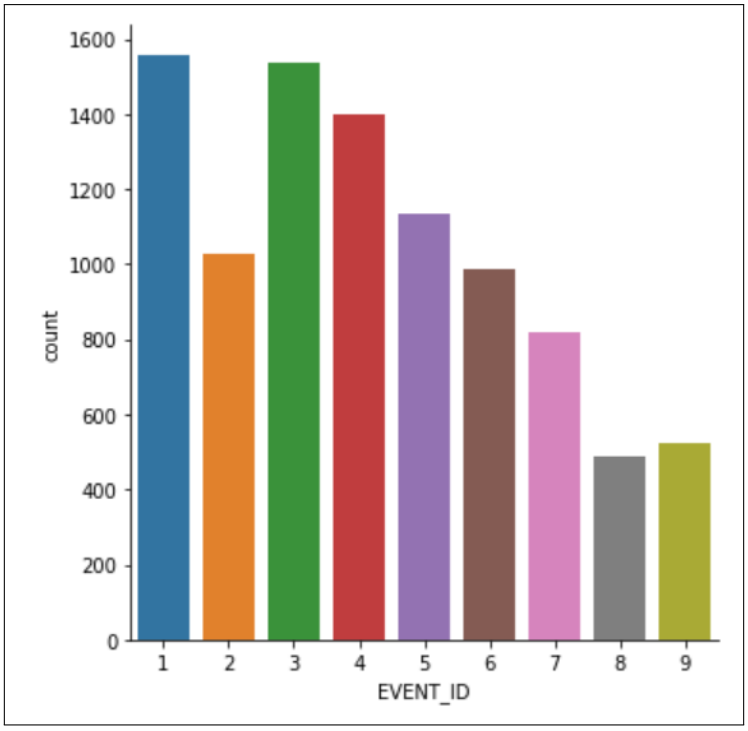

Figure 7: Different visits for PD Assessments

```
data= data.drop(['SCAU20','SCAU21','SCAU24'],axis=1) # These column has
    mostly missing values, so its better to remove them

data.isna().sum().sum() # 827 missing records in dataset
data['NHY'].isna().sum()

# fill NA with interpolate function after sorting each patient with
   their visits.
data = data.set_index(['PATNO']).sort_index(level='PATNO') #set index
   and sort data
data = data.interpolate(method='linear', axis=1)

# replace the late stages into moderate stages due to insufficient data
   , and as we want to predict if the patients are 'severe' and need
   medical attention or 'not-severe'
data['NHY'] = data['NHY'].replace([5.0, 4.0], 3.0)
PD_data = data
print(PD_data) # data is now ready for analysis
```

Listing 7: Handling Missing Records

The heat map for missing variables can be seen in figure 8.

6 Modelling

The data is split into features and labels where 'NHY' is label and rest all are features. The features are further standardized using min-max normalisation. The data is then split into train and test in the ratio of 80-20. The steps are shown below.

```
#The dataset is split into features(the independent variables) and
   labels(the dependent or target variable):
```

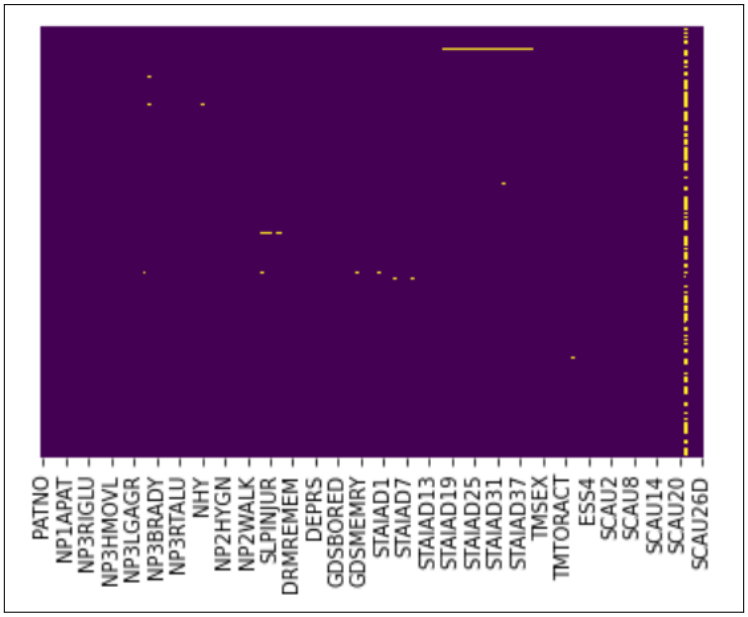

Figure 8: Heat Map for Missing Variables

```
features=PD_data.drop(['NHY'],axis=1)
labels=PD_data['NHY']
#normalize the data using the minmax scaler to bring the feature
   variables within the range -1 to 1:
scaler=MinMaxScaler((-1,1))
X=scaler.fit_transform(features)
y=labels

#split for training data is 80% and testing is 20%
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.2)
print('X_train num',X_train.shape[0])
print('X_test num',X_test.shape[0])
```

Listing 8: Standardization and Train Test split

6.1 LSTM

Long Short Term Memory Model is built for predicting Parkinson's Disease Severity. LSTM model deals well with the complexities of longitudinal data of PD patients, learning from long-term dependencies. The code is given below. Keras has been utilized to import models and layers.

```
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
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
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import confusion_matrix, classification_report,
    roc_auc_score, roc_curve
from keras.layers.core import Activation
```

Listing 9: import required libraries

The standardized data is reshaped into Time-series with time-steps equal to number of average visits for each patient, i.e. 5.

```
# reshape input to be 3D [samples, timesteps, features]
timesteps = 5
X_train = X_train.reshape((X_train.shape[0], timesteps, int(X_train.
    shape[1]/timesteps)))
X_test = X_test.reshape((X_test.shape[0], timesteps, int(X_test.shape
    [1]/timesteps)))
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
#(7577, 5, 34) (7577,) (1895, 5, 34) (1895,)
```

Listing 10: Reshape according to time-steps

The LSTM model is built with 10% dropouts to avoid over-fitting after each dense layer. Softmax activation is used for outer layer to make sure the probability outcomes totals to 1. Optimizer used in compiling the LSTM architecture is 'adam' optimizer and metrics is 'sparse categorical accuracy'.

```
model = Sequential()
model.add(LSTM(44, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(20))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Dense(10))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Dense(4))
model.add(Activation('softmax'))

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

model.summary()
```

Listing 11: LSTM Model Architecture

Model is implemented with early stopping for model loss and model checkpoint for validation accuracy.

```
early_stopping = keras.callbacks.EarlyStopping(monitor= 'val_loss',
    mode= 'min',  patience= 10)

model_checkpoint = ModelCheckpoint ('model.h5', monitor= '
    val_sparse_categorical_accuracy', mode = 'max', verbose= 1,
    save_best_only= True)

history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
    batch_size=32, epochs=100, shuffle= False,
                    callbacks=[early_stopping, model_checkpoint],
    verbose=2)
```

Listing 12: Fit the Model

Plot the loss and accuracy for comparing validation sample results.

```
# plot the loss
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot(title="LSTM Model Loss")
plt.ylabel('loss')
plt.xlabel('epoch')

#plot model accuracy
plt.plot(history.history['sparse_categorical_accuracy'])
plt.plot(history.history['val_sparse_categorical_accuracy'])
plt.title('LSTM model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='upper left')
plt.show()

# Make Predictions
y_score = model.predict(X_test)
y_pred = y_score

y_pred = (y_pred>0.5)
y_pred = y_pred.astype(int)
y_pred[0] #array([0, 1, 0, 0])
```

Listing 13: Plots for Model Loss and Model Accuracy

Accuracy for LSTM model is 88%. Model doesn't seem to overfit as the curves of train and validation data are not spaced apart. The LSTM model plots for loss function and model accuracy is shown in figure 9 and figure 10 respectively.

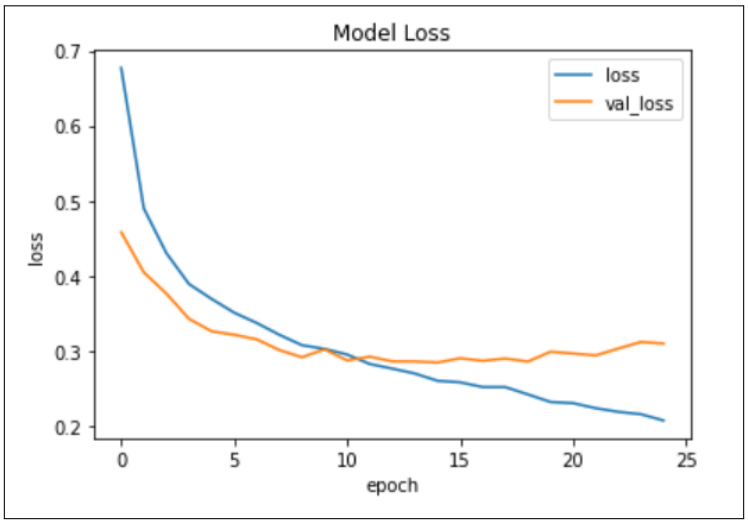

Figure 9: LSTM Model Loss Plot

6.2 DEEP NEURAL NETWORK- Multi-Layer Perceptron

The feed forward network has been built using similar steps as before for binary classification.

```
#Patients with 0.0 and 1.0 score on HY scale are considered 'unilateral
    normal' patients,
#whereas patients with score of 2 or more needs medical attention and
   are considered 'cognitively impaired' in this project.
```

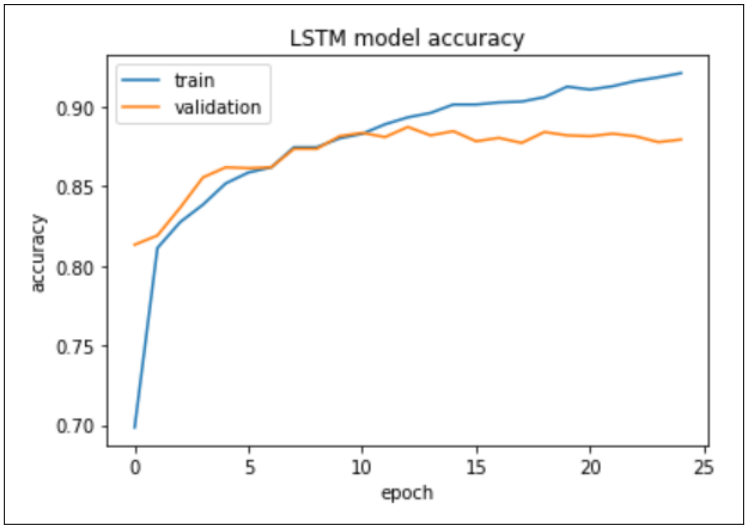

Figure 10: LSTM Model Accuracy Plot

```
PD_data['NHY'] = PD_data['NHY'].replace([0.0],1.0)
PD_data['NHY'] = PD_data['NHY'].replace([3.0], 2.0)

#split for training data is 80% and testing is 20%
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.2)
print(X_train.shape)
input_shape = [X_train.shape[1]] #(7577, 170)

#Model architecture
mlp_model = Sequential()
mlp_model.add(layers.Dense(170, activation='relu', input_dim=
    input_shape))
mlp_model.add(Dropout(0.1))
mlp_model.add(layers.Dense(70, activation='relu'))
mlp_model.add(Dropout(0.1))
mlp_model.add(layers.Dense(16, activation='relu'))
mlp_model.add(Dropout(0.1))
mlp_model.add(layers.Dense(1, activation='sigmoid'))

# compile model with optimizing algorithm, loss function and accuracy
    function
mlp_model.compile(optimizer="adam", loss="binary_crossentropy", metrics
    =["binary_accuracy"])

early_stopping = keras.callbacks.EarlyStopping(monitor= 'val_loss',
    mode= 'min',  patience= 10)

history = mlp_model.fit(X_train, y_train, validation_data=(X_test,
    y_test),batch_size=32, epochs=100, shuffle= False,
                    callbacks=[early_stopping],verbose=2)
#Model Plots
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot(title="Cross-entropy")
history_df.loc[:, ['binary_accuracy', 'val_binary_accuracy']].plot(
    title="Accuracy")
#Make predictions
mlp_preds = mlp_model.predict(X_test)
#Evaluate
```

```
accuracy = mlp_model.evaluate (X_test, y_test)[1]
print ('Accuracy:', accuracy)
#60/60 [==============================] - 0s 1ms/step - loss:
    -2475099947008.0000 - binary_accuracy: 0.4739
#Accuracy: 0.4738786220550537
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
    mlp_preds))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
    mlp_preds))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(
    y_test, mlp_preds)))
#Mean Absolute Error: 0.5261213720316623
#Mean Squared Error: 0.5261213720316623
#Root Mean Squared Error: 0.72534224475875
```

Listing 14: Multi Layer Perceptron Model

7 Hybrid Model- GA-LSTM

Next, built LSTM model within genetic algorithm heuristic search engine to find the best hyperparameters of the model.

```
# import the required libraries. The model was built using bitstring
    and Deap framework.
!pip install bitstring
!pip install deap
from keras.models import Model
from deap import base, creator, tools, algorithms
from scipy.stats import bernoulli
from bitstring import BitArray
np.random.seed(1120)
```

Listing 15: install depedencies and libraries

```
Collecting bitstring
  Downloading bitstring-3.1.9-py3-none-any.whl (38 kB)
Installing collected packages: bitstring
Successfully installed bitstring-3.1.9
```

Figure 11: BitString Installation

```
Collecting deap
  Downloading deap-1.3.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (160 kB)
     |████████████████████████████████| 160 kB 12.1 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from deap) (1.19.5)
Installing collected packages: deap
Successfully installed deap-1.3.1
```

Figure 12: DEAP Installation

The successful installation message will appear as shown in figure 11 and 12.

```
# reshape data into 1-dimensional array
data = np.reshape(np.array(PD_data['NHY']),(len(PD_data['NHY']),1))
data.shape

```

```
# Use first 8525 points as training/validation and rest of the points as test set.
train_data = data[0:8525]
test_data = data[8525:]

# Prepare the dataset with X,Y pair where for a given time t, X is the previous values and Y is the future values
def prep_dataset(data, window_size):
    X, Y = np.empty((0,window_size)), np.empty((0))
    for i in range(len(data)-window_size-1):
        X = np.vstack([X,data[i:(i + window_size),0]])
        Y = np.append([Y,data[i + window_size,0]])
    X = np.reshape(X,(len(X),window_size,1))
    Y = np.reshape(Y,(len(Y),1))
    return X, Y
```

Listing 16: Prepare dataset for genetic algorithm search

Next, Define a function to get window size and number of units by decoding GA solution. Then,prepare the dataset using GA searched window size, further divide the dataset into train and validation set. Finally, train the LSTM model, calculate fitness score of the current solution by Genetic Algorithm.

```

def train_evaluate(ga_individual_solution):
    # Decode GA solution to integer for window_size and num_units
    window_size_bits = BitArray(ga_individual_solution[0:6])
    num_units_bits = BitArray(ga_individual_solution[6:])
    window_size = window_size_bits.uint
    num_units = num_units_bits.uint
    print('\nWindow Size: ', window_size, ', Num of Units: ', num_units)
        # Return fitness score of 100 if window_size or num_unit is zero
    if window_size == 0 or num_units == 0:
        return 100,

    # Segment the train_data based on new window_size; split into train and test (80/20)
    X,Y = prep_dataset(train_data,window_size)
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.20, random_state = 1120)

    # Train LSTM model and predict on validation set
    inputs = Input(shape=(window_size,1))
    lstm = LSTM(num_units, input_shape=(window_size,1))(inputs)
    predictions = Dense(1, activation='linear')(lstm)
    gamodel = Model(inputs=inputs, outputs=predictions)
    gamodel.compile(optimizer='adam',loss='binary_crossentropy')
    gamodel.fit(X_train, y_train, epochs=10, batch_size=10,shuffle=True)
    y_pred = gamodel.predict(X_test)
    y_pred = y_pred.astype(int)

    # Calculate the RMSE score as fitness score for GA
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    print('Validation RMSE: ', rmse,'\n')

```

```
    return rmse,
```

Listing 17: Define Functions for Generating Genetic Algorithm Search

Using DEAP package, defined parameters for implementing GA. Here, the solution is represented by a binary representation '0's and '1's. Bernoulli distribution will randomly initialize the generation. Crossover and mutation are randomly selected by roulette wheel. Finally, the GA parameter values are arbitrarily initialized.

```
population_size = 4
num_generations = 4
gene_length = 10

# to minimize the RMSE score, weights are defined as -1.0.
creator.create('FitnessMax', base.Fitness, weights = (-1.0,))
creator.create('Individual', list , fitness = creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register('binary', bernoulli.rvs, 0.5)
toolbox.register('individual', tools.initRepeat, creator.Individual,
    toolbox.binary,
n = gene_length)
toolbox.register('population', tools.initRepeat, list , toolbox.
    individual)

toolbox.register('mate', tools.cxOrdered)
toolbox.register('mutate', tools.mutShuffleIndexes, indpb = 0.6)
toolbox.register('select', tools.selRoulette)
toolbox.register('evaluate', train_evaluate)

population = toolbox.population(n = population_size)
r = algorithms.eaSimple(population, toolbox, cxpb = 0.4, mutpb = 0.1,
ngen = num_generations, verbose = False)
```

Listing 18: Define Genetic Algorithm parameters

```
# Print top N solutions - (1st only)
best_individuals = tools.selBest(population,k = 1)
best_window_size = None
best_num_units = None

for bi in best_individuals:
    window_size_bits = BitArray(bi[0:6])
    num_units_bits = BitArray(bi[6:])
    best_window_size = window_size_bits.uint
    best_num_units = num_units_bits.uint
    print('\nTop Solution for Window Size: ', best_window_size, ', Num
    of Units: ', best_num_units)

##### Result for top solution
#Top Solution for Window Size:  12 , Num of Units:  8
```

Listing 19: Print the top solutions

```
# Train the model using best configuration on complete training set
#and make predictions on the test set
X_train,y_train = prep_dataset(train_data,best_window_size)
X_test, y_test = prep_dataset(test_data,best_window_size)

```

```
inputs = Input(shape=(best_window_size,1))
GA_LSTM = LSTM(best_num_units, input_shape=(best_window_size,1))(inputs)
predictions = Dense(1, activation='linear')(GA_LSTM)
gamodel = Model(inputs = inputs, outputs = predictions)
gamodel.compile(optimizer='adam',loss='binary_crossentropy',  metrics=['mean_squared_error'])
gamodel.fit(X_train, y_train, epochs=50, batch_size=32,shuffle=True)
y_pred = gamodel.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print('Test RMSE: ', rmse)

#Epoch 50/50
#266/266 [==============================] - 1s 4ms/step - loss: 0.1349 - mean_squared_error: 0.1349
#Test RMSE:  0.32939433126774104

#y_pred = gamodel.predict(X_test)
#print ('Accuracy:', metrics.accuracy_score(y_test, y_pred)) ##

############ Not working on GA-LSTM model as the classification metrics can't handle a mix of binary and continuous targets.
```

Listing 20: Train the LSTM model using the top solutions

Hence, Test Root Mean Square Error (RMSE) was significantly reduced from 0.72 of baseline approach to 0.33 in GA-LSTM model

8 Multiple Baseline Machine Learning Algorithms

Multiple baseline classification models have been trained and evaluated to monitor the classification performance on Parkinson's disease data.

```
#Importing various classification algorithm to find which algorithm suits the best for the dataset:
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import SGDClassifier
```

Listing 21: Importing Classifiers

```
names = ["Nearest_Neighbors", "Linear_SVM", "Polynomial_SVM", "RBF_SVM", "Gaussian_Process",
         "Gradient_Boosting", "Decision_Tree", "Extra_Trees", "Random_Forest", "Neural_Net", "AdaBoost",
         "Naive_Bayes", "QDA", "SGD"]

classifiers = [
```

```
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(kernel="poly", degree=3, C=0.025),
    SVC(kernel="rbf", C=1, gamma=2),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    GradientBoostingClassifier(n_estimators=100, learning_rate=1.0),
    DecisionTreeClassifier(max_depth=5),
    ExtraTreesClassifier(n_estimators=10, min_samples_split=2),
    RandomForestClassifier(max_depth=5, n_estimators=100),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(n_estimators=100),
    GaussianNB(),
    QuadraticDiscriminantAnalysis(),
    SGDClassifier(loss="hinge", penalty="l2")]
```

Listing 22: Fitting the classification models

```
scores = []
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    scores.append(score)

df = pd.DataFrame()
df['name'] = names
df['score'] = scores
df

#Plot the results for comparison
sns.set(style="whitegrid")
ax = sns.barplot(y="name", x="score", data=df)
```

Listing 23: Evaluate and compare the classifiers

The plot has been shown in figure 13. It can be seen that XG Boost model shows the best performance for classification.

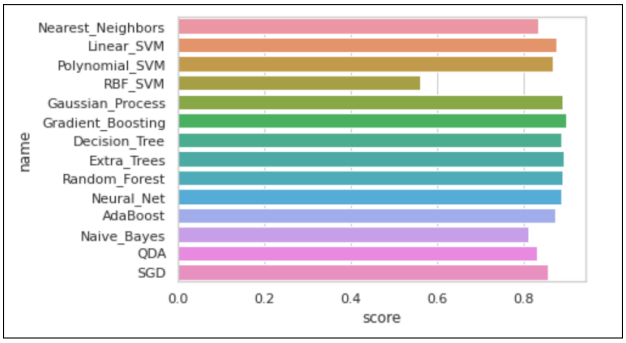

Figure 13: Comparison Plot

Further the classification scores can be seen in figure 14 which shows XG Boost algorithm with accuracy of 89.76% whereas deep learning neural network shows the accuracy of 88.75%. Even though the score for most of the machine learning algorithms are high, they do not consider the multiple time steps for multiple visits of patients.

	name	score
0	Nearest_Neighbors	0.833245
1	Linear_SVM	0.874934
2	Polynomial_SVM	0.868074
3	RBF_SVM	0.559894
4	Gaussian_Process	0.890237
5	Gradient_Boosting	0.897625
6	Decision_Tree	0.887071
7	Extra_Trees	0.891821
8	Random_Forest	0.889182
9	Neural_Net	0.887599
10	AdaBoost	0.874406
11	Naive_Bayes	0.810026
12	QDA	0.829551
13	SGD	0.857520

Figure 14: Classification Scores for different Models

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. et al. (2016). Tensorflow: A system for large-scale machine learning, *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283.

Carneiro, T., Da Nóbrega, R. V. M., Nepomuceno, T., Bian, G.-B., De Albuquerque, V. H. C. and Reboucas Filho, P. P. (2018). Performance analysis of google colaboratory as a tool for accelerating deep learning applications, *IEEE Access* **6**: 61677–61685.

Matloff, N. (2011). *The art of R programming: A tour of statistical software design*, No Starch Press.

Van Rossum, G., Drake, F. L. et al. (2000). *Python reference manual*, iUniverse Indiana.