Configuration Manual

MSc Research Project
Data Analytics

Ayushi Bajaj
Student ID: x20242638

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ayushi Bajaj
Student ID:	x20242638
Programme:	Data Analytics
Year:	2018
Module:	MSc Research Project
Supervisor:	Aaloka Anant
Submission Due Date:	20/12/2018
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	6

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:	15th December 2022

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

Ayushi Bajaj
x20242638

1 Introduction

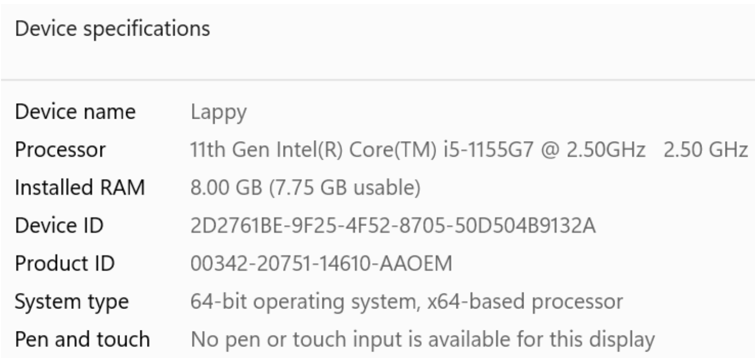
This paper's goal is to describe the coding procedure for the project. The hardware and software setups required to duplicate the research in the future are outlined. This section describes the steps required to execute the script, as well as the design and implementation procedures required for effective executable code.

2 System Configuration

This segment will discuss the system configuration of the project.

2.1 Hardware Configuration

The hardware configuration of the device used is as follows.



Device specifications

Device name	Lappy
Processor	11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz 2.50 GHz
Installed RAM	8.00 GB (7.75 GB usable)
Device ID	2D2761BE-9F25-4F52-8705-50D504B9132A
Product ID	00342-20751-14610-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: Device configuration

2.2 Software Configuration

The software used for this project is Jupyter Notebook. This software was launched using Anaconda Navigator. This coding platform is open source, easy to use and web interactive. Figure 2 shows the Anaconda navigator.

3 Data Preparation

Following steps will show the code that was sequenced and run in Jupyter Notebook

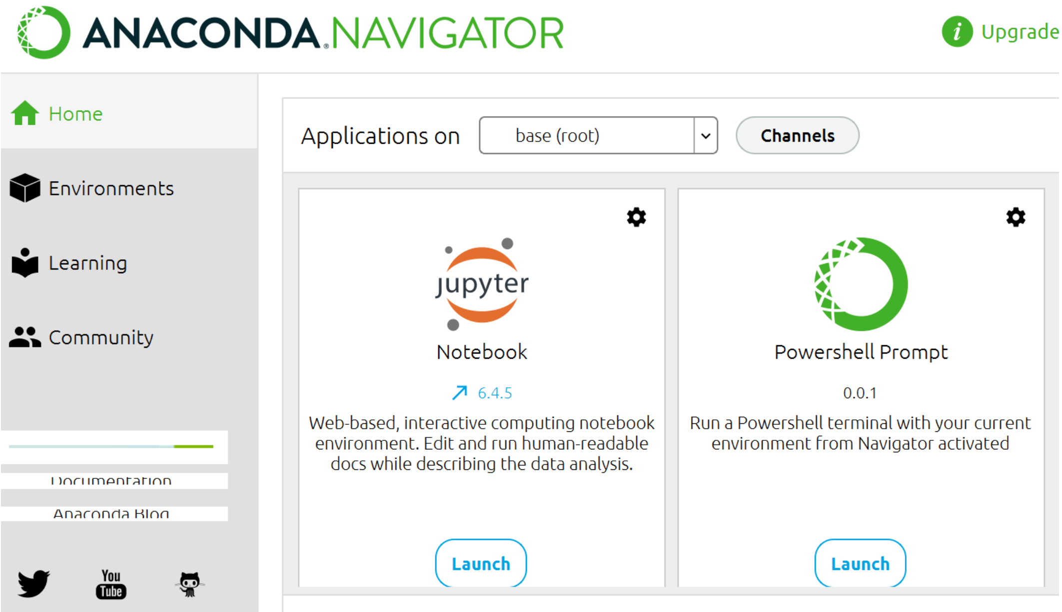


Figure 2: Software Required

3.1 Data Selection

Six data files are used in this project they all are in CSV format and are been downloaded from a open source site, Kaggle.

3.2 Importing Libraries

As shown in figure 3 Following libraries were imported initially in the project. SHAP library requires older version of numpy, Hence numpy is later imported in the project.

```
In [1]: #importing libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# explainability
import shap
# print the JS visualization code to the notebook to check for any missing library
shap.initjs()
import os
import warnings
warnings.filterwarnings('ignore')
```

Figure 3: libraries imported

3.3 Importing Data

As shown in figure 4, The data is imported to different data frames.

3.4 Data Pre-processing

For ptr processing the data, As shown in figure 5 and 6, the null value of data is checked and a primary is form to merge the data.

```
In [76]: #importing data
         train_df = pd.read_csv('Training.csv')
         submission = pd.read_csv('score.csv')
         matches=pd.read_csv('Matches IPL.csv')
         pre_matches= pd.read_csv('pre Matches IPL.csv')
         squads = pd.read_csv('IPL Squads.csv',encoding= 'unicode_escape')
```

Figure 4: Importing CSV files

```
In [3]: train_df['player'] = train_df['Id']
        train_df['number'] =  train_df['Id']
        for i in range(0, len( train_df)):
            train_df['player'][i] =  train_df['Id'][i].split("_")[-1]
            train_df['number'][i] = int( train_df['Id'][i].split('_')[:1][0])
```

```
In [4]: submission['player'] = submission['Id']
        submission['match_number'] =  submission['Id']
        for i in range(0, len( submission)):
            submission['player'][i] =  submission['Id'][i].split("_")[-1]
            submission['match_number'][i] = int( submission['Id'][i].split('_')[:1][0])
            submission['season'] = 2020
```

Figure 5: creating primary key

```
In [24]: #checking for missing values
         df.isnull().sum().sum()

Out[24]: 0
```

Figure 6: checking for null value in final data frame.

4 Data Mining

This part will show the data modelling part of the project. Figure 7 shows the library used for data mining. Figure 8 shows how data was transformed and divided into training

Data Modelling

```
In [29]: #Libraries for data modelling are imported
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import mean_squared_error
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.svm import SVR
         from sklearn import metrics
         from sklearn.preprocessing import LabelEncoder
         import xgboost as xgb
         import optuna
         import numpy as np
```

Figure 7: Libraries used

and testing.

4.1 Random Forest Regression Model

Figure 9 shows how Random Forest Regression Model was implemented and RMSE value was checked.

```
In [31]: X =df.drop(['Id','match_number','team','total_score'],axis=1)
         le =LabelEncoder()
         X.player = le.fit_transform(df.player)
         X.team1 = le.fit_transform(df.team1)
         X.team2 = le.fit_transform(df.team2)
         X.venue = le.fit_transform(df.venue)
         X.season = le.fit_transform(df.season)
         X = X[:15916]
         y = df['total_score']
         X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2, random_state=10)
```

Figure 8: Data used for training and testing

```
RFR= RandomForestRegressor()
RFR.fit(X_train,y_train)

RFR_predictiontrain=RFR.predict(X_train)
RFR_predictiontest=RFR.predict(X_test)
```

```
dfRFR = pd.DataFrame({'Actual': y_train, 'Predicted': RFR_predictiontrain})
print(dfRFR.head())
print('RMSE train:', np.sqrt(metrics.mean_squared_error(y_train, RFR_predictiontrain)))
print('MSE train:', metrics.mean_squared_error(y_train, RFR_predictiontrain))
```

```
print('RMSE test:', np.sqrt(metrics.mean_squared_error(y_test, RFR_predictiontest)))
print('MSE test:', metrics.mean_squared_error(y_test, RFR_predictiontest))
```

Figure 9: Random Forest Regression Model

4.2 Decision Tree Regression Model

Figure 10 shows how Random Forest Support Regression Model vector machine (SVM) was implemented and RMSE value was checked.

```
DTR = DecisionTreeRegressor()

DTR.fit(X_train,y_train)

DTR_predictiontrain=DTR.predict(X_train)
DTR_predictiontest=DTR.predict(X_test)
```

```
dfDTR = pd.DataFrame({'Actual': y_train, 'Predicted': DTR_predictiontrain})
print(dfDTR.head())
print('RMSE train:', np.sqrt(metrics.mean_squared_error(y_train, DTR_predictiontrain)))
print('MSE train:', metrics.mean_squared_error(y_train, DTR_predictiontrain))
```

```
print('RMSE test:', np.sqrt(metrics.mean_squared_error(y_test, DTR_predictiontest)))
print('MSE test:', metrics.mean_squared_error(y_test, DTR_predictiontest))
```

Figure 10: Decision Tree Regression Model

4.3 Support vector machine (SVM)

Figure 12 shows how Support vector machine (SVM) Regression Model was implemented and RMSE value was checked.

```
SVRmodel = SVR()
SVRmodel.fit(X_train,y_train)

SVR_predictiontrain=SVRmodel.predict(X_train)
SVR_predictiontest=SVRmodel.predict(X_test)
```

```
dfSVR = pd.DataFrame({'Actual': y_train, 'Predicted': SVR_predictiontrain})
print(dfSVR.head())
print('RMSE train:', np.sqrt(metrics.mean_squared_error(y_train, SVR_predictiontrain)))
print('MSE train:', metrics.mean_squared_error(y_train, SVR_predictiontrain))
```

```
print('RMSE test:', np.sqrt(metrics.mean_squared_error(y_test, SVR_predictiontest)))
print('MSE test:', metrics.mean_squared_error(y_test, SVR_predictiontest))
```

Figure 11: Support vector machine (SVM)

4.4 XGBoost Regression Model

Figure 12,13 and 14 shows how XGBoost Regression Model was implemented, hyperparametric tuning applied and RMSE value was checked.

```
In [41]: def objective(trial):
             params = {
                 'random_state': 0,
                 'n_estimators': trial.suggest_categorical('n_estimators', [1000]),
                 'max_depth': trial.suggest_int('max_depth', 3, 8),
                 'learning_rate': trial.suggest_float('learning_rate', 0.001, 1.0),
                 'reg_lambda': trial.suggest_float('reg_lambda', 0.0, 10),
                 'reg_alpha': trial.suggest_float('reg_alpha', 0.0, 10),
                 'gamma': trial.suggest_float('gamma', 0.0, 10),
                 'subsample': trial.suggest_categorical('subsample', [0.8, 0.9, 1.0]),
                 'colsample_bytree': trial.suggest_categorical('colsample_bytree', [0.1, 0.2, 0.3, 0.4, 0.5]),
             }

             model = xgb.XGBRegressor(**params)
             model.fit(X_train, y_train, eval_set=[(X_test,y_test)], early_stopping_rounds=1000, verbose=0,)
             y_pred = model.predict(X_test)
             rmse = mean_squared_error(y_test, y_pred, squared=False)
             return rmse

In [42]: %%time
         study = optuna.create_study(direction='minimize',sampler=optuna.samplers.TPESampler(seed=0))
         study.optimize(objective, n_trials=100)
         print('Number of finished trials:', len(study.trials))
         print('Best parameters:', study.best_trial.params)
         print('Best RMSE:', study.best_trial.value)
```

Figure 12: Applying XGBoost

```
In [43]: params = study.best_params
         params['random_state'] = 0
         params['n_estimators'] = 10000
         #params['tree_method'] = 'gpu_hist'

         finalmodel = xgb.XGBRegressor(**params)
         finalmodel.fit(X_train,y_train,eval_set=[(X_test, y_test)],early_stopping_rounds=1000,verbose=2)
```

Figure 13: Hyperparametric tuning

```
In [45]: print('min',y_pred.min())
         print('max',y_pred.max())

         print('RMSE',np.sqrt(mean_squared_error(y_test,y_pred)))
         print("MSE :",mean_squared_error(y_test,y_pred))

         min 14.670556
         max 52.735462
         RMSE 28.97214917658526
         MSE : 839.3854279103098
```

Figure 14: Checking RMSE

4.5 Prediction

In fig 15 roles and price is assign to players and mean of theire points is taken to score them for overall season.

5 Explainable AI

In this segment explainable AI part of the code is discussed.

5.1 Random Forest

Figure 16 shows the implementation of explainable AI in Random Forest Regression Model. Here, heat map is produced along with beeswarm and bar graph.

```
In [ ]: submission['Total Points'] = ys_pred

In [ ]: Role =pd.read_csv('Player role.csv')

In [ ]: Role= Role.drop(['country','batting_style','bowling_style','Price in Rupees(lakh)'],axis=1)

In [ ]: Role.head()

In [ ]: listRole = pd.merge(submission,Role, right_index=False, left_index=False)

In [ ]: listRole.head()

In [ ]: test2 = listRole.groupby(['player','role','Price in Dollar'])['Total Points'].mean()
        test2.head()
```

Figure 15: Final Prediction

```
In [ ]: explainerRn = shap.Explainer(RFR)
        shap_valuesRn = explainerRn(X)

In [ ]: shap.plots.bar(shap_valuesRn)

In [ ]: shap.plots.beeswarm(shap_valuesRn)

In [ ]: shap.plots.heatmap(shap_valuesRn[:1000])
```

Figure 16: Random Forest

5.2 Decision Tree

Figure 17 shows the implementation of explainable AI in Decision Tree Regression Model. Here, beeswarm and bar graph are produced.

```
In [ ]: explainerDt = shap.Explainer(DTR)
        shap_valuesDt = explainerDt(X)

In [ ]: shap.plots.bar(shap_valuesDt)

In [ ]: shap.plots.beeswarm(shap_valuesDt)
```

Figure 17: Decision Tree

5.3 XGBoost

Figure 16 shows the implementation of explainable AI in XGBoost Regression Model. Here, waterfall, beeswarm and bar graph are produced.

Explainable AI

```
In [ ]: explainer = shap.Explainer(finalmodel)
        shap_values = explainer(total_df)
        shap.plots.bar(shap_values)

In [ ]: shap.plots.beeswarm(shap_values)

In [ ]: shap.plots.waterfall(shap_values[0])
```

Figure 18: XGBoost