National College of Ireland

Configuration Manual

MSc Research Project
MSCDAD*A*

Taiwo Mubarak Oladapo
StudentID:22107312

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Taiwo Mubarak Oladapo
Student ID:	22107312
Programme:	MSCDAD$_A$
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr Catherine Mulwa
Submission Due Date:	31/01/2024
Project Title:	Configuration Manual
Word Count:	1671
Page Count:	10

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:	TAIWO M. OLADAPO
Date:	31st January 2024

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

Taiwo Mubarak Oladapo
x22107312

1 Introduction

The Configuration manual serves as a guidance document that provides information on the project development, installation, deployment, and implementation of the" Exploring the Impact of Artificial Intelligence in Predicting English Premier League Football Matches" project, as described in the technical report. This report serves as a support and guidance document for the technical report, helping to accomplish the intended output and outcomes at each step. Many hardware, software, libraries, and technology combinations are used in the completion of the entire project.

2 System Requirements

This section discusses the system requirements for the project; it is generally helpful to know the prerequisites before performing computer experiments. It is divided into 2 parts: hardware and software requirements.

2.1 Hardware Requirements

The Hardware requirements comprise the laptop specification used to execute this research project.

Figure 1: Hardware Requirements

2.2 Software Requirements

- Python 3.11.5: Python is a popular and flexible programming language that forms the basis of this project. The project code uses Python version 3.7.

- Jupyter Notebook: For interactive development, data exploration, and documentation, the Jupyter Notebook is essential to this project. With the help of this open-source web application, documents with live code, visualizations, and descriptive text can be created and shared.

3 Environment setup

3.1 Jupyter Notebook

Depending on the operating system, Python must first be installed; installing the most recent version is advised. For MacOS, Python 3.11.5 was downloaded and set up. A development environment is necessary for writing, running, and assessing programs after Python has been installed. The most widely used and convenient platform is Jupyter Notebook. It is part of the Anaconda Python distribution, for which the system type will determine which installation is suitable. The Anaconda interface is seen in Figure 2 together with additional applications like Jupyter Notebook. We must launch the Jupyter Notebook and create a new Python file before we can start writing Python code.

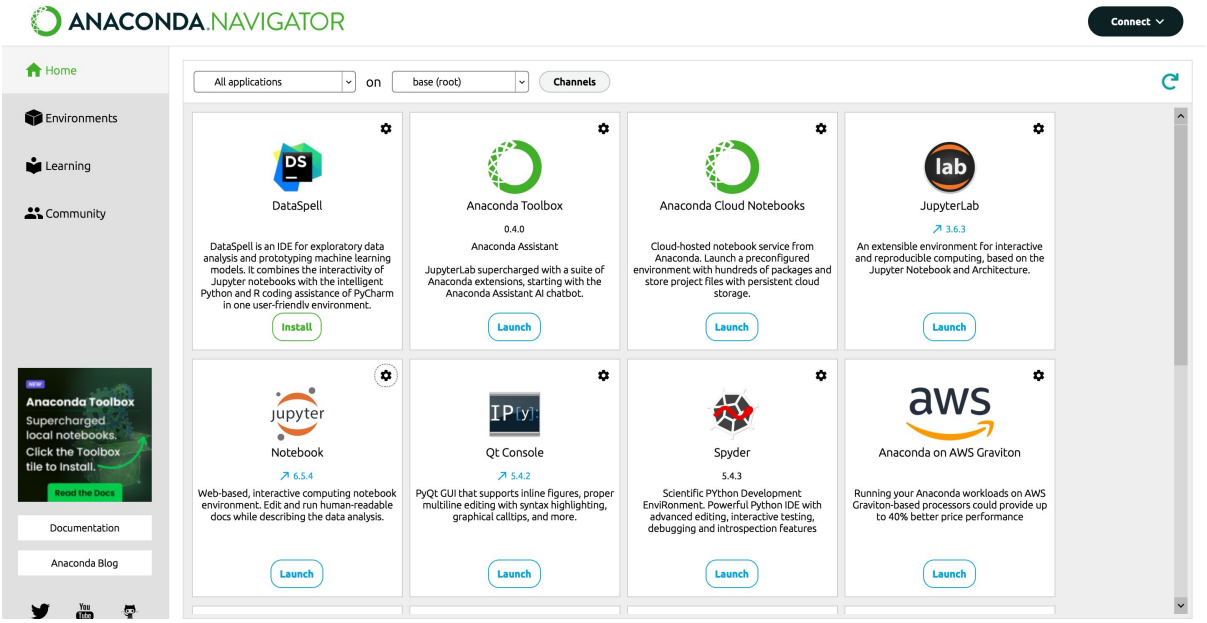

Figure 2: Hardware Requirements

3.2 Installation of Packages and Importing Raw Data

Premier League matches 1993–2022 is the dataset that was obtained from Kaggle. The dataset is acquired from its source and stored on the desktop in the.csv format. Importing and preprocessing data cannot be done without first installing packages.

```
import pandas as pd
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, precision_score
from xgboost import XGBClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

%matplotlib inline
```

Figure 3: Importing Libraries

3.3 Importing of the Datasets

The dataset is imported from the desktop after being saved there using the Panda library, which is loaded as pd. As seen in Figure 4, the dataset name is premier-league-matches 3.csv, and EPL df is the variable in which the dataset is loaded.

```
EPL_df = pd.read_csv("premier-league-matches 3.csv")
EPL_df.head()
```

Figure 4: Importing data into Jupyter Notebook

3.4 Data Preprocessing

All values in the dataset were intact. No missing values were found. Figure 5 shows the exploratory data analysis to plot the continuous distribution and categorical distribution of the dataset.

```
def plot_continuous_distribution(data: pd.DataFrame = None, column: str = None, height: int = 8):
  _ = sns.displot(data, x=column, kde=True, height=height, aspect=height/5).set(title=f'Distribution of {column}');

def get_unique_values(data, column):
  num_unique_values = len(data[column].unique())
  value_counts = data[column].value_counts()
  print(f"Column: {column} has {num_unique_values} unique values\n")
  print(value_counts)

def plot_categorical_distribution(data: pd.DataFrame = None, column: str = None, height: int = 8, aspect: int = 2):
  _ = sns.catplot(data=data, x=column, kind='count', height=height, aspect=aspect).set(title=f'Distribution of {colu
  _.set_xticklabels(rotation=60)
  plt.show()

def barplot(data: pd.DataFrame = None, column1: str = None, column2: str = None, ylab_str: str = None):
  plt.figure(figsize=(10, 6))
  plt.bar(data[column1], data[column2])
  plt.title(f'Total {column2} by Team')
  plt.xlabel('Team')
  plt.xticks(rotation=90)
  plt.ylabel(f'Total {column1} {ylab_str}')
  plt.show()
```

Figure 5: Exploratory Data Analysis

Feature Engineering was also performed to make the dataset more robust. date was converted to date time, daycode was generated from date and monthcode was generated from date. The process is illustrated in Figure 6.

```
#EPL_df = convert_to_datetime(EPL_df, 'Date')

EPL_df["Date"] = pd.to_datetime(EPL_df["Date"])

EPL_df["DayCode"] = EPL_df["Date"].dt.dayofweek
EPL_df['MonthCode'] = EPL_df['Date'].dt.month
EPL_df.info()
```

Figure 6: Conversion to datetime

Label encoding was also denoted on the" FTR" variable. The FTR was also transformed into a new column" Target". The process is illustrated in Figure 7.

```
encoder = LabelEncoder()

EPL_df['Target'] = encoder.fit_transform(EPL_df['FTR'])
EPL_df.head()
```

Figure 7: FTR Label encoding to Target.

Home and Away Categorical variables were converted to numerical variables as Homecode and Awaycode. Figure 8 shows the illustration.

```
# Convert Categorical data to numerical
EPL_df["HomeCode"] = EPL_df["Home"].astype("category").cat.codes
EPL_df['AwayCode'] = EPL_df['Away'].astype("category").cat.codes

EPL_df.info()
```

Figure 8: Categorical variable to numerical variable

4 Modelling

The function rolling averages is designed to compute rolling averages for specific columns in the data frame grouped by some criteria. The group parameter represents a group of data whereas the col parameter consists of column names that need to be calculated with a rolling average. The new cols parameter saves the new column names for the computed rolling averages. The columns that were calculated with the rolling average were the Home Goals and Away Goals and a new data frame was created for the new rolling averages which is EPL rolling as shown in Figures 9 and 10.

```
def rolling_averages(group, cols, new_cols):
    group = group.sort_values("Date")
    rolling_stats = group[cols].rolling(3, closed='left').mean()
    group[new_cols] = rolling_stats
    group = group.dropna(subset=new_cols)
    return group
```

Figure 9: Rolling average model.

```
cols = ['HomeGoals', 'AwayGoals']
new_cols = [f"{c}_rolling" for c in cols]

EPL_rolling = EPL_df.groupby("Home").apply(lambda x: rolling_averages(x, cols, new_cols))
```

Figure 10: New data frame for the rolling averages

Based on the values in the Date column, two subsets of the EPL rolling data frame are generated. There are training and test sets from the two groups which is shown in Figure 11. The test set produces results where the Date is greater than 2017-04-01, while the training set produces results where the Date is less than or equal to 2017-04-01. The
models are trained using the training set, and their performance is evaluated using the test set.

```
train = EPL_rolling[EPL_rolling['Date'] <= '2017-04-01']
test = EPL_rolling[EPL_rolling['Date'] > '2017-04-01']
```

Figure 11: EPL rolling data frame.

A list called predictor was also created. The predictors initially contain the home code and Away code columns. These columns are used as predictors for this project. new col which is

generated from the rolling average is added as a predictor to predict the outcome of matches. Figure 12 illustrates the code.

```
predictors = ['HomeCode', 'AwayCode']
predictors = predictors + new_cols
```

Figure 12: Predictors for the model

The code preparation for the training and validation sets of a machine learning model is shown in Figure 13, where predictor variables are assigned, the data is standardised using StandardScaler, and the scaled arrays are converted back to Pandas Data Frames. Standardization is used to make the features similar, and the use of StandardScaler raises the possibility that the model may be sensitive to the feature scale.

```
'''
X_train[predictors] = train[predictors]
y_train = train['Target']
X_val[predictors] = test[predictors]
y_val = test['Target']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

X_train = pd.DataFrame(X_scaled)
X_val = pd.DataFrame(X_val_scaled)

X_train.head()
'''
```

Figure 13: Training and Test Predictors

The training and validation set's components are shown in Figure 14. For each training set, the feature matrix is represented by X train, related target values by y train, the feature matrix for the testing set by X val, and the corresponding target values by y val for the set of validations.

```
X_train = train[predictors]
y_train = train['Target']
X_val = test[predictors]
y_val = test['Target']
```

Figure 14: Training and Test Predictors Variables

5 Evaluation of the Implemented Model

Five machine-learning model code snippets used for the implementation are shown in Figure 15. The Random Forest Classifier uses the Scikit-learn library and 3 hyperparameters. 100 n

estimators: is used which is the number of trees in the forest. 2 minimum samples split was used to split the internal node. The random state was split into 42 seeds to ensure the reproducibility of the model. The Logistic Regression uses the Scikit-learn library with no default parameters used. The Decision Tree Classifier uses the Scikit-learn library. No default parameters are used. There is a comment indicating some parameters (min samples split, random state, n estimators, and max depth), but these are typically associated with the Random Forest model, not a single Decision Tree. The Linear Support Vector Classifier (svc) uses the Scikit-learn library. It uses just a single parameter which is" dual". This parameter is used to solve the primal optimization problem. Setting it to False is recommended when the number of samples is greater than the number of features. The XGBoost Classifier (xgb) uses an XGBoost library. No default parameters are used in this case. There is a comment indicating some parameters (n estimators, max depth, and random state), but these are typically associated with XGBoost's hyperparameters.

```
rf = RandomForestClassifier(n_estimators=100, min_samples_split=10, random_state=42)
lr = LogisticRegression()
dt = DecisionTreeClassifier() # min_samples_split=10, random_state=1, n_estimators=100, max_depth=7
svc = LinearSVC(dual=False)
xgb = XGBClassifier() # n_estimators=500, max_depth=7, random_state=42
```

Figure 15: Implemented Models

The machine learning model is being trained on the dataset. The line fits train each model using the training features (train[predictors]) and the corresponding target variable (train['Target']). The code snippet is shown in Figure 16.

```
rf.fit(train[predictors], train['Target'])
lr.fit(train[predictors], train['Target'])
dt.fit(train[predictors], train['Target'])
svc.fit(train[predictors], train['Target'])
xgb.fit(train[predictors], train['Target'])
```

Figure 16: Model Training

6 Analysis of Results

The classification report of each model was evaluated. Figure 17 shows the code snippet for the classification report of the Random Forest. The classification report uses the sklearn.metrics library.

```
print(f"\nForest: {classification_report(test['Target'], rfpred)}")
```

```
Forest:               precision    recall  f1-score   support

           0       0.47      0.33      0.39       769
           1       0.24      0.12      0.16       535
           2       0.50      0.73      0.59      1056

    accuracy                           0.46      2360
   macro avg       0.40      0.39      0.38      2360
weighted avg       0.43      0.46      0.43      2360
```

Figure 17: Random Forest Result

Figure 18 shows the code snippet for the classification report of the Logistic Regression.

```
print(f"Logistic: {classification_report(test['Target'], lrpred)}")
```

```
Logistic:               precision    recall  f1-score   support

           0       0.41      0.14      0.21       769
           1       0.00      0.00      0.00       535
           2       0.46      0.92      0.61      1056

    accuracy                           0.46      2360
   macro avg       0.29      0.35      0.27      2360
weighted avg       0.34      0.46      0.34      2360
```

Figure 18: Logistic Regression Result

Figure 19 shows the code snippet classification report of the Decision Tree.

```
print(f"\nDecision Tree: {classification_report(test['Target'], dtpred)}")
```

```
Decision Tree:               precision    recall  f1-score   support

           0       0.38      0.35      0.36       769
           1       0.24      0.26      0.25       535
           2       0.49      0.49      0.49      1056

    accuracy                           0.39      2360
   macro avg       0.37      0.37      0.37      2360
weighted avg       0.40      0.39      0.39      2360
```

Figure 19: Decision Tree Result

Figure 20 shows the code snippet classification report of the Support Vector Machine.

```
print(f"\nSVC: {classification_report(test['Target'], svcpred)}")
```

```
SVC:              precision    recall  f1-score   support

           0       0.41      0.13      0.20       769
           1       0.00      0.00      0.00       535
           2       0.46      0.92      0.61      1056

    accuracy                           0.45      2360
   macro avg       0.29      0.35      0.27      2360
weighted avg       0.34      0.45      0.34      2360
```

Figure 20: Support Vector Machine Result

Figure 21 shows the code snippet classification report of the Extreme Gradient Boosting.

```
print(f"\nXGB: {classification_report(test['Target'], xgbpred)}")
```

```
XGB:              precision    recall  f1-score   support

           0       0.48      0.38      0.42       769
           1       0.24      0.13      0.17       535
           2       0.51      0.71      0.60      1056

    accuracy                           0.47      2360
   macro avg       0.41      0.41      0.40      2360
weighted avg       0.44      0.47      0.44      2360
```

Figure 21: Extreme Gradient Boosting Result

6.1 Hyperparameter Tuning on the Best Model

Figure 22 shows the code snippet of the hyperparameter tuning performed on the best model which is the Extreme Gradient Boosting.

```
from sklearn.model_selection import RandomizedSearchCV, RepeatedKFold
from sklearn.metrics import make_scorer

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
parameters = {'learning_rate':[0.0001,0.001,0.005,0.5],
              'subsample': [0.9,0.5,0.2,0.1],
              'n_estimators':[100,400,800,1000],
              'max_depth':[4,6,8,10]}

random_model = RandomizedSearchCV(estimator=xgb, param_distributions= parameters,
                                  scoring='precision_macro', n_iter=100, n_jobs=-1, cv=cv)
```

Figure 22: Extreme Gradient Boosting Hyperparameter Tuning

Figure 23 shows the code snippet for the best hyperparameter tuning for the Extreme Gradient Boosting.

```
print(f"Best Hyperparameters: {model.best_params_}")
best_est = model.best_estimator_
```

```
Best Hyperparameters: {'subsample': 0.2, 'n_estimators': 100, 'max_depth': 6, 'learning_rate': 0.005}
```

Figure 23: Best hyperparameter Selector

Figure 24 shows the code snippet for the accuracy and precision of the hyperparameter tuning performed on extreme gradient boosting.

```
precision = precision_score(test["Target"], search_pred, average='macro')
accuracy = accuracy_score(test["Target"], search_pred)

print(f"After optimization:\nprecision = {precision}\naccuracy = {accuracy}")
```

Figure 24: Accuracy of the Extreme Gradient Boosting Hyper Parameter Tuning