National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Saurav Jaglan
Student ID: 22105433

School of Computing
National College of Ireland

Supervisor: Dr.Athanasios Staikopoulos

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saurav Jaglan
Student ID:	22105433
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr.Athanasios Staikopoulos
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	729
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:	Saurav Jaglan
Date:	13th December 2023

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

Saurav Jaglan
22105433

1 Introduction

In this configuration manual the building of simulation model for simulating end season ranking of football leagues and model to predict players value is described. The hardware and software required along with the description on where and how the softwares were used is discussed.

2 System Configuration

This section of the report describe the system configurations and requirements.

2.1 Hardware Requirements

Table 1: Hardware requirements

Device	Lenovo Ideapad 5
Operating System	Windows 10 Home
RAM	16 gb
Storage	512 gb SSD
Processor	AMD Ryzen 5 7530U with Radeon Graphics 2.00 GHz
System Type	64-bit operating system, x64-based processor

2.2 Software Requirements

Table 2: Hardware requirements

Language	Python 3.11.3
IDE	Jupyter Notebook
Initial Processing	Excel (Power Query)
Web Browser	Google Chrome
Environment	Anaconda
Other Softwares	Microsoft Excel & Overleaf

3 Project Development

This section provide information about configuring the environment, collection of data, and information about different libraries required is discussed in this section.

3.1 Installing Python

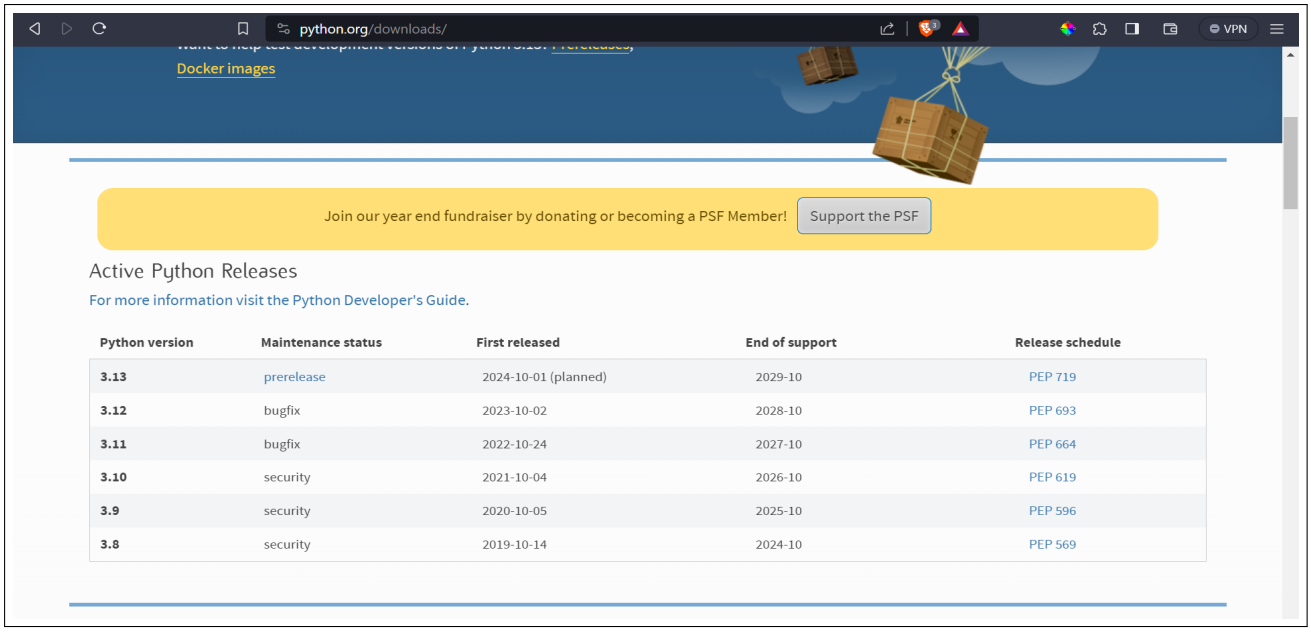

Figure 1: Website for downloading python

The figure 1 displays the website to download python on the device and the website[1] also contains steps to installing it.

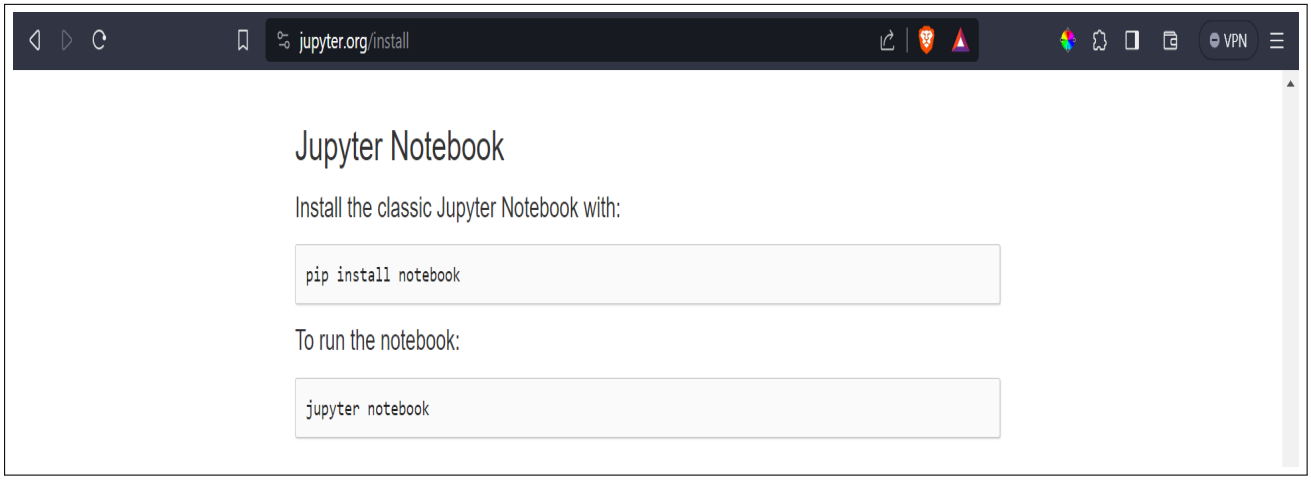

Figure 2: Website for installing Jupyter Notebook

[1] https://www.python.org/downloads/

In the figure 2, installation command of Jupyter Notebook is displayed, and is available on the website[2] .

3.2 Data Acquisition

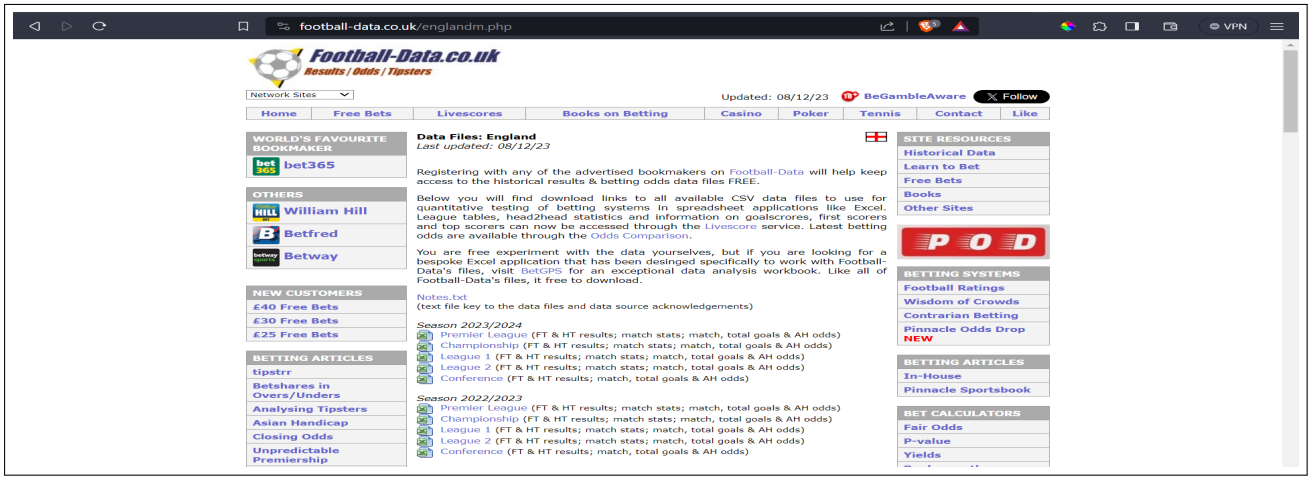

Figure 3: Teams past matches result data

The data set require to build the simulation model was collected from the Football-data [3]website as shown in figure3.

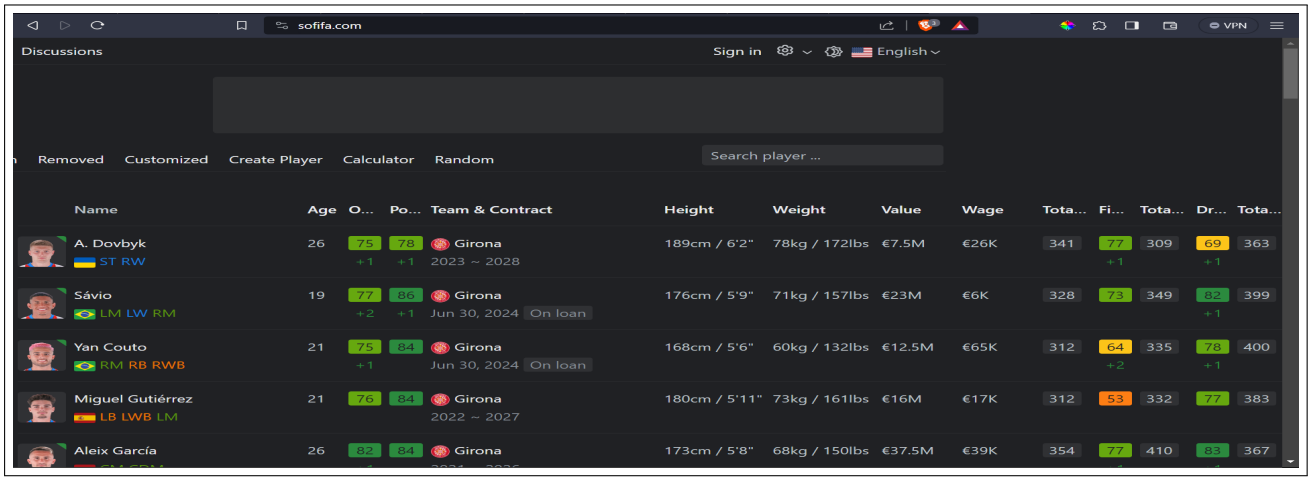

Figure 4: Players attributes data

To estimate players value in different leagues the data was collected from the webiste named Sofifa [4] as shown in figure 4.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

Figure 5: Libraries required for simulation model

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt
```

Figure 6: Libraries Required for players value forecasting model

3.3 Importing Libraries

Figure 5 and 6 represnts the figures required for the two models respectively. Libraries such as Pandas and Numpy were mainly used for descriptive analysis and pandas was mainly used to read the data in a frame (Tabular format). For visualisation and to illustrate findings graphically, Seaborn and Matplotlib were used. The library Sklearn contains all the functions rquired for scaling data, splitting data into training and testing, encoding data, and building machine learning models.

3.4 Reading Datasets

```
# Load the dataset
file_path = 'Bundesliga.xlsx'
bundesliga_data = pd.read_excel(file_path)

# Display the first few rows of the dataset to understand its structure and contents
bundesliga_data.head()
```

	Date	HomeTeam	AwayTeam	FullTime	Halftime	HomeGoals	HomeGoalsHalftime	HomeShots	HomeShotsOnTarget	HomeCorners	...	HomeYellowCard
0	05/08/2022	Ein Frankfurt	Bayern Munich	A	A	1	0	8	2	5	...	
1	06/08/2022	Augsburg	Freiburg	A	D	0	0	10	2	5	...	
2	06/08/2022	Bochum	Mainz	A	D	1	1	16	3	3	...	
3	06/08/2022	M'gladbach	Hoffenheim	H	D	3	1	18	8	4	...	
4	06/08/2022	Union Berlin	Hertha	H	H	3	1	18	8	5	...	

Figure 7: Reading Dataset 1

```
# Load the Bundesliga dataset
bundesliga_df = pd.read_excel('Bundesilga Player Data.xlsx')

# Display the first few rows and some general information about the dataset
bundesliga_info = bundesliga_df.info()
bundesliga_head = bundesliga_df.head()

bundesliga_info, bundesliga_head
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 360 entries, 0 to 359
Data columns (total 30 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Name                360 non-null    object
 1   Age                 360 non-null    int64
 2   Overall rating      360 non-null    int64
 3   Potential           360 non-null    int64
 4   Value               360 non-null    object
 5   Wage                360 non-null    object
 6   Total attacking     360 non-null    int64
 7   Finishing           360 non-null    int64
 8   Total skill         360 non-null    int64
 9   Dribbling           360 non-null    int64
 10  Ball control        360 non-null    int64
 11  Total movement      360 non-null    int64
 12  Acceleration        360 non-null    int64
 13  Total power         360 non-null    int64
 14  Stamina             360 non-null    int64
 15  Strength            360 non-null    int64
 16  Total mentality     360 non-null    int64
 17  Aggression          360 non-null    int64
 18  Total defending     360 non-null    int64
 19  Marking             360 non-null    int64
 20  Total goalkeeping   360 non-null    int64
```

Figure 8: Reading Dataset 2

[2] https://jupyter.org/install
[3] https://www.football-data.co.uk/englandm.php
[4] https://sofifa.com/

4 Model Building

4.1 Simulation Model to forecast season end ranking

```
# Setting up visualisation styles
sns.set(style="whitegrid")

# 1. Team Performance Analysis
# Counting the number of wins, draws, and losses for each team
home_wins = bundesliga_data[bundesliga_data['FullTime'] == 'H'].groupby('HomeTeam').size()
away_wins = bundesliga_data[bundesliga_data['FullTime'] == 'A'].groupby('AwayTeam').size()
draws = bundesliga_data[bundesliga_data['FullTime'] == 'D'].groupby('HomeTeam').size()  # Draw can be counted for either team

# Merging the counts into a single DataFrame
team_performance = pd.DataFrame({
    'HomeWins': home_wins,
    'AwayWins': away_wins,
    'Draws': draws
}).fillna(0)  # Fill NaN values with 0

# Total wins, losses, and draws for each team
team_performance['TotalWins'] = team_performance['HomeWins'] + team_performance['AwayWins']
team_performance['TotalMatches'] = team_performance.sum(axis=1)
team_performance['WinRate'] = team_performance['TotalWins'] / team_performance['TotalMatches']

# 2. Home vs. Away Performance Analysis
# Calculating the total number of goals scored at home and away
home_goals = bundesliga_data.groupby('HomeTeam')['HomeGoals'].sum()
away_goals = bundesliga_data.groupby('AwayTeam')['AwayGoals'].sum()

# Merging the goals into the team_performance DataFrame
team_performance['HomeGoals'] = home_goals
team_performance['AwayGoals'] = away_goals
team_performance['TotalGoals'] = team_performance['HomeGoals'] + team_performance['AwayGoals']

# Plotting the data
fig, ax = plt.subplots(1, 2, figsize=(18, 6))

# Team Win Rate
sns.barplot(x=team_performance['WinRate'], y=team_performance.index, ax=ax[0])
ax[0].set_title('Team Win Rate')
ax[0].set_xlabel('Win Rate')
ax[0].set_ylabel('Teams')

# Home vs Away Goals
team_performance[['HomeGoals', 'AwayGoals']].plot(kind='bar', stacked=True, ax=ax[1])
ax[1].set_title('Home vs Away Goals')
ax[1].set_xlabel('Teams')
ax[1].set_ylabel('Goals')

plt.tight_layout()
plt.show()

# Displaying the first few rows of the team performance DataFrame
team_performance.head()
```

Figure 9: Descriptive Analysis

```
# Simulating a single round of matches (each team plays once at home and once away)
def simulate_round(teams, avg_goals):
    results = []

    for home_team in teams:
        for away_team in teams:
            if home_team != away_team:
                # Simulating goals scored using Poisson distribution
                home_goals = np.random.poisson(avg_goals.loc[home_team, 'AvgHomeGoals'])
                away_goals = np.random.poisson(avg_goals.loc[away_team, 'AvgAwayGoals'])
                results.append((home_team, away_team, home_goals, away_goals))
    return results

# List of teams
teams = team_goals_avg.index.tolist()

# Simulating one round of matches
round_results = simulate_round(teams, team_goals_avg)

# Displaying the results of the first few simulated matches
round_results[:5]
```

Figure 10: Simulating Matches using Random Poisson

```python
# Converting the final standings to a DataFrame for better readability
final_standings_df = pd.DataFrame({
    'Team': [team for team, data in final_standings],
    'Points': [data['Points'] for team, data in final_standings],
    'Goals For': [data['GoalsFor'] for team, data in final_standings],
    'Goals Against': [data['GoalsAgainst'] for team, data in final_standings],
    'Goal Difference': [data['GoalsFor'] - data['GoalsAgainst'] for team, data in final_standings]
})

# Displaying the final standings table
final_standings_df.head()  # Displaying the top 5 teams in the standings
```

Figure 11: Converting Results into Dataframe

```python
# Assuming bundesliga_data is your DataFrame containing the dataset

# Define the features and the target
features = ['HomeTeam', 'AwayTeam', 'HomeGoals', 'AwayGoals', 'HomeShots', 'AwayShots',
            'HomeShotsOnTarget', 'AwayShotsOnTarget', 'HomeCorners', 'AwayCorners',
            'HomeFouls', 'AwayFouls', 'HomeYellowCards', 'AwayYellowCards', 'HomeRedCards', 'AwayRedCards']
X = bundesliga_data[features]
y = bundesliga_data['FullTime']

# Encoding categorical data and normalizing
categorical_features = ['HomeTeam', 'AwayTeam']
numeric_features = X.columns.drop(categorical_features)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Creating the KNN model pipeline
knn_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('knn', KNeighborsClassifier(n_neighbors=5))  # Using 5 neighbors for KNN
])

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Training the KNN model
knn_pipeline.fit(X_train, y_train)

# Evaluating the model
y_pred = knn_pipeline.predict(X_test)
classification_report_result = classification_report(y_test, y_pred)

print(classification_report_result)
```

Figure 12: Simulating Matches using KNN

```python
# Slicing the DataFrame to include only the first 38 matches
# Creating a DataFrame for comparison
comparison_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred
})

# Resetting the index to get a common index for plotting
comparison_df.reset_index(drop=True, inplace=True)

comparison_38_matches = comparison_df.head(38)

# Plotting
plt.figure(figsize=(15, 6))
plt.plot(comparison_38_matches['Actual'], label='Actual Results', marker='o', linestyle='-')
plt.plot(comparison_38_matches['Predicted'], label='Predicted Results', marker='x', linestyle='--')
plt.title('Comparison of Actual vs Predicted Match Outcomes (First 38 Matches)')
plt.xlabel('Matches')
plt.ylabel('Outcomes (H/D/A)')
plt.xticks(range(38))  # to show each match as a tick
plt.legend()
plt.show()
```

Figure 13: Acutal match result vs predicted match result

4.2 Model to estimate players value in different leagues

```python
def convert_currency_to_numeric(df, columns):
    """
    Convert currency columns (e.g., '€119.5M', '€170K') to numeric values in millions.
    """
    for column in columns:
        # Remove currency symbol and convert K and M to their numeric equivalents
        df[column] = df[column].replace(r'[€K]', '', regex=True).replace(r'M', 'e6', regex=True).astype(float)
        # Convert values from euros to millions
        df[column] = df[column] / 1e6

    return df

# Convert currency columns for the Bundesliga dataset
bundesliga_df = convert_currency_to_numeric(bundesliga_df, ['Value', 'Wage'])

# Check the first few rows to confirm the conversion
bundesliga_df[['Value', 'Wage']].head()
```

Figure 14: Converting value and wage format to numeric

```python
# Function to train and evaluate models
def train_and_evaluate(X_train, X_test, y_train, y_test):
    # Dictionary to store models and their names
    models = {
        'Random Forest': RandomForestRegressor(),
        'Gradient Boosting': GradientBoostingRegressor(),
        'Decision Tree': DecisionTreeRegressor()
    }

    results = {}

    # Training and evaluating each model
    for name, model in models.items():
        # Train the model
        model.fit(X_train, y_train)

        # Predict on the test set
        y_pred = model.predict(X_test)

        # Calculate evaluation metrics
        mae = mean_absolute_error(y_test, y_pred)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_test, y_pred)

        # Store results
        results[name] = {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R2': r2}

    return results

# Prepare Bundesliga dataset
X_bundesliga = bundesliga_df.drop('Value', axis=1)  # Features
y_bundesliga = bundesliga_df['Value']  # Target variable
```

Figure 15: Model building to predict players value

```python
# Removing non-numeric columns from the datasets
X_bundesliga = bundesliga_df.select_dtypes(include=['int64', 'float64']).drop('Value', axis=1)
X_premier_league = premier_league_df.select_dtypes(include=['int64', 'float64']).drop('Value', axis=1)
X_serie_a = serie_a_df.select_dtypes(include=['int64', 'float64']).drop('Value', axis=1)

# Re-splitting the Bundesliga dataset into training and testing sets
X_train_bundesliga, X_test_bundesliga, y_train_bundesliga, y_test_bundesliga = train_test_split(
    X_bundesliga, y_bundesliga, test_size=0.2, random_state=42
)

# Train and evaluate models on the corrected Bundesliga dataset
results_bundesliga = train_and_evaluate(X_train_bundesliga, X_test_bundesliga, y_train_bundesliga, y_test_bundesliga)
results_bundesliga
```

Figure 16: Evaluating Model

```python
# Data for plotting
leagues = ['Bundesliga', 'Premier League', 'Serie A']
predicted_values_kane = [
    kane_prediction['Predicted Value by Bundesliga Model (in Millions)'],
    kane_prediction['Predicted Value by Premier League Model (in Millions)'],
    kane_prediction['Predicted Value by Serie A Model (in Millions)']
]

# Creating the bar chart
plt.figure(figsize=(10, 6))
plt.bar(leagues, predicted_values_kane, color=['blue', 'green', 'red'])
plt.xlabel('League')
plt.ylabel('Predicted Value (in Millions)')
plt.title('Predicted Market Value of H. Kane ST Across Different Leagues')
plt.ylim(0, max(predicted_values_kane) + 10)  # Adding some space above the highest bar for clarity

# Adding the value labels on top of each bar
for i in range(len(leagues)):
    plt.text(i, predicted_values_kane[i], f'€{predicted_values_kane[i]:.2f}M', ha='center', va='bottom')

# Showing the plot
plt.show()
```

Figure 17: Displaying Results