Machine Learning Report:
Sport Predict - Predicting Premier League Wins

Class 2 - Group 5

Team Members:

22BI13321 - Lê Hoài Nam

22BI13323 - Nguyễn Hoàng Phương Nam

22BI13325 - Nguyễn Thành Nam

22BI13327 - Tạ Hiếu Nam

22BI13328 - Trần Thành Nam

Introduction:

The objective of this project is to develop machine learning models to predict the number of wins for football teams in the English Premier League (EPL). The prediction of wins can provide valuable insights into team performance and success in the league. In this report, we will outline the data analysis, model development, evaluation, and insights derived from the project.

Table of Contents:

I. Introduction to the Dataset:

1. Purpose:

The dataset aims to analyze and predict outcomes in the English Premier League (EPL). Specifically, it seeks to understand patterns in team performance, goals, and standings to predict future wins.

2. Source:

The dataset used is a combination of custom-built data and publicly available data. It includes match results, player statistics, and season standings.

3. Size:

- The dataset consists of several files:
    + `results.csv`: Match outcomes
    + `stats.csv`: Team statistics
    + `EPL Standings 2000-2022.csv`: Historical standings
    + `with_goalscorers.csv`: Top scorer’s information

4. Data Split:

The data is divided into training, testing, and validation sets. Typically, 80% of the data is used for training, and 20% for testing. If applicable, a further split might be used to create a validation set.

II. Data Overview:

1. Dataset:

The dataset comprises several CSV files containing EPL match results, team statistics, league standings, and top goalscorers from the seasons 2000-2022.

2. Data Processing:

Initial data processing involved loading the datasets using pandas and numpy libraries. This included handling missing values, extracting relevant information, and merging datasets for further analysis.

League data:

	home_team	away_team	...	result	season
0	Sheffield United	Liverpool	...	D	2006-2007
1	Arsenal	Aston Villa	...	D	2006-2007
2	Everton	Watford	...	H	2006-2007
3	Newcastle United	Wigan Athletic	...	H	2006-2007
4	Portsmouth	Blackburn Rovers	...	H	2006-2007
...	...	...	...	...	...
4555	Newcastle United	Chelsea	...	H	2017-2018
4556	Southampton	Manchester City	...	A	2017-2018
4557	Swansea City	Stoke City	...	A	2017-2018
4558	Tottenham Hotspur	Leicester City	...	H	2017-2018
4559	West Ham United	Everton	...	H	2017-2018

Standing 1 in each season:

	Season	Pos	...	Pts	Qualification or relegation
120	2006-07	1	...	89	Qualification for the Champions League group s...
140	2007-08	1	...	87	Qualification for the Champions League group s...
160	2008-09	1	...	90	Qualification for the Champions League group s...
180	2009-10	1	...	86	Qualification for the Champions League group s...
200	2010-11	1	...	80	Qualification for the Champions League group s...
220	2011-12	1	...	89	Qualification for the Champions League group s...
240	2012-13	1	...	89	Qualification for the Champions League group s...
260	2013-14	1	...	86	Qualification for the Champions League group s...
280	2014-15	1	...	87	Qualification for the Champions League group s...
300	2015-16	1	...	81	Qualification for the Champions League group s...
320	2016-17	1	...	93	Qualification for the Champions League group s...
340	2017-18	1	...	100	Qualification for the Champions League group s...

[12 rows x 12 columns]

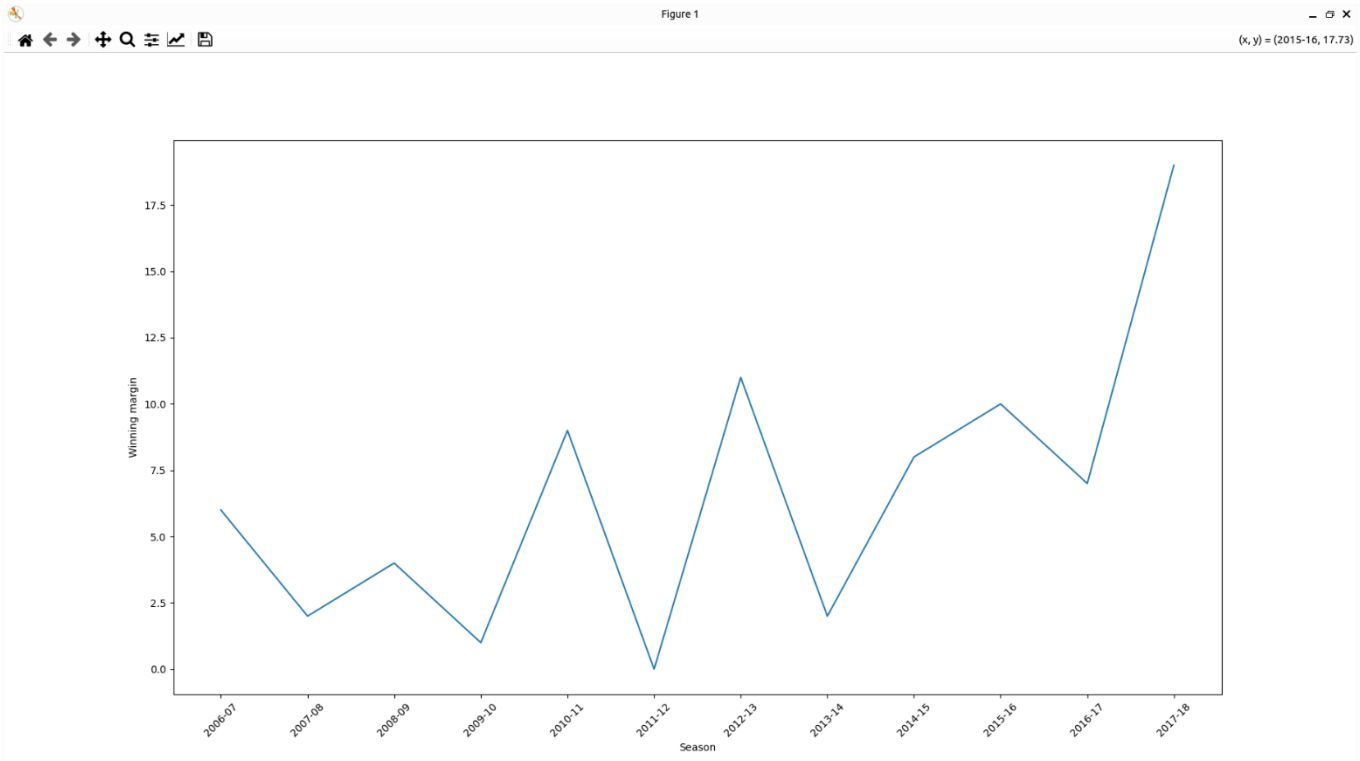
Ranking details(position,team, number of matchs, number of wins,..):

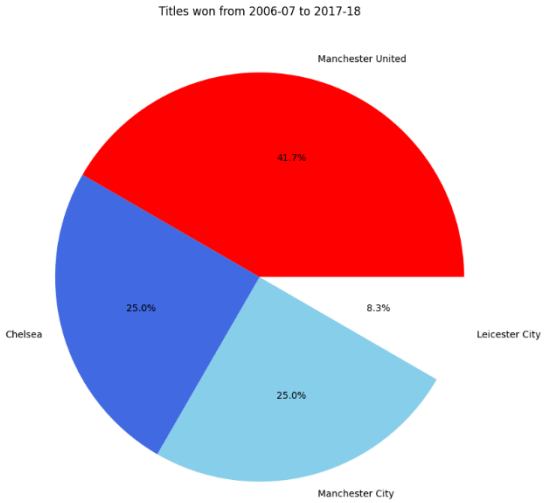
	Season	Pos	Team	Pld	W	D	L	GF	GA	GD	Pts
0	2006-07	1	Manchester United	38	28	5	5	83	27	56	89
1	2006-07	2	Chelsea	38	24	11	3	64	24	40	83
2	2006-07	3	Liverpool	38	20	8	10	57	27	30	68
3	2006-07	4	Arsenal	38	19	11	8	63	35	28	68
4	2006-07	5	Tottenham Hotspur	38	17	9	12	57	54	3	60
..	...	...	...	...	..	..	..	..	..	..	...
235	2017-18	16	Huddersfield Town	38	9	10	19	28	58	-30	37
236	2017-18	17	Southampton	38	7	15	16	37	56	-19	36
237	2017-18	18	Swansea City	38	8	9	21	28	56	-28	33
238	2017-18	19	Stoke City	38	7	12	19	35	68	-33	33
239	2017-18	20	West Bromwich Albion	38	6	13	19	31	56	-25	31

[240 rows x 11 columns]

3. Data Analysis:

Descriptive statistics, visualizations, and exploratory data analysis (EDA) were performed to gain insights into team performances, top goalscorers, and league standings.





III. AI-ML Models Used:

1. Models:

The primary models used are Linear Regression, Logistic Regression, Random Forest Regressor . These models are built using the `scikit-learn` library.

2. Construction:

- Linear Regression: Used to predict the number of wins based on various team statistics.

- Logistic Regression: Used to classify outcomes, such as whether a team will win a championship or not.

- Random Forest Regressor: Utilized to predict continuous outcomes, such as the expected points scored by a team in a season, based on a combination of input features. This model aggregates the results from multiple decision trees to improve predictive accuracy and control over-fitting.

3. Implementation:

The code to implement these models uses `scikit-learn`, and the models are trained and evaluated using standard procedures from this library. The implementation is custom-written, with methodologies informed by online tutorials and documentation from `scikit-learn` and other reputable sources.

IV. Model Development:

1. Models Used:

Machine learning models were employed: Linear Regression, Logistic Regression and Random Forest Regressor

2. Feature Engineering:

Features such as goals scored, goals conceded, points and based on the number of wins, position were used to predict the number of wins.

3. Data Splitting:

The dataset was split into training and testing sets using the `train_test_split` function from `scikit-learn`.

4. Model Training:

Linear Regression, Logistic Regression, Random Forest Regressor models were trained on the training data using `scikit-learn's` Linear Regression, Logistic Regression, Random Forest Regressor classes.

V. Model Evaluation:

1. Evaluation Metrics:

The performance of the models was assessed using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Accuracy, Precision, Recall, and F1 Score.

2. Results:

Evaluation results indicated the effectiveness of the models in predicting wins, with linear regression achieving lower error rates compared to and random forest regression.

VI. Results Analysis:

1. Evaluation Metrics:

- Linear Regression: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) are used to evaluate performance.

- Logistic Regression: Accuracy, Precision, Recall, and F1 Score are used for classification performance.

- Random Forest Regressor: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) are used to evaluate performance.

2. Results:

- Linear Regression: Provides a quantitative prediction of wins. Metrics like MAE, MSE, and RMSE indicate how close the predicted values are to the actual values.

- Logistic Regression: Provides a classification of outcomes. Metrics like accuracy and F1 Score indicate the balance between precision and recall in predictions.

- Random Forest Regressor: Delivers a quantitative prediction for continuous outcomes, such as the expected points scored by a team. Metrics like MAE, MSE, and RMSE help in understanding the accuracy and reliability of these predictions.

3. Comparison and Analysis:

By building and comparing multiple models, we can analyze which model provides better performance for the given task. For example, comparing the RMSE of Linear Regression and Random Forest Regressor helps in understanding which model better predicts quantitative outcomes, while comparing the accuracy of Logistic Regression helps in understanding its classification performance. This comprehensive comparison allows us to determine the most suitable model for our specific data and prediction goals.

Result of logistic and linear regression:

```
Logistic Regression Evaluation Metrics:
Accuracy: 0.10
Precision: 0.53
Recall: 0.10
F1 Score: 0.36

Linear Regression Evaluation Metrics:
Mean absolute error: 1.58
Mean squared error: 3.42
Root mean squared error: 1.85


0.8417016599784696
[Elapsed time: 6.272038459777832 seconds]
```

```
Random Forest Regressor Evaluation Metrics:
Mean absolute error: 1.44
Mean squared error: 3.40
Root mean squared error: 1.84
```

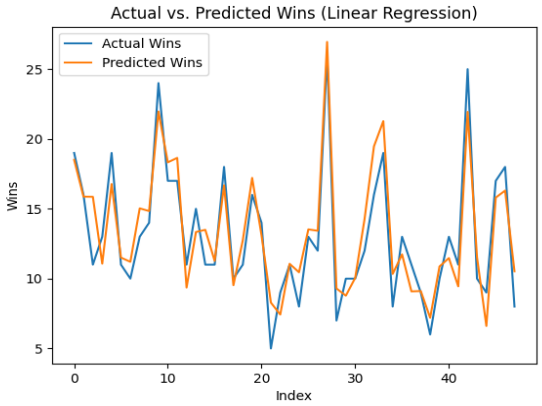
- The data after calculation of the linear function compared to the random function are almost the same, with very little difference:

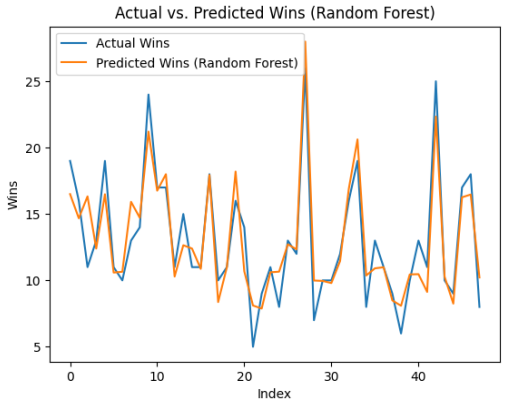
+ MAE: 1.58 >< 1.44

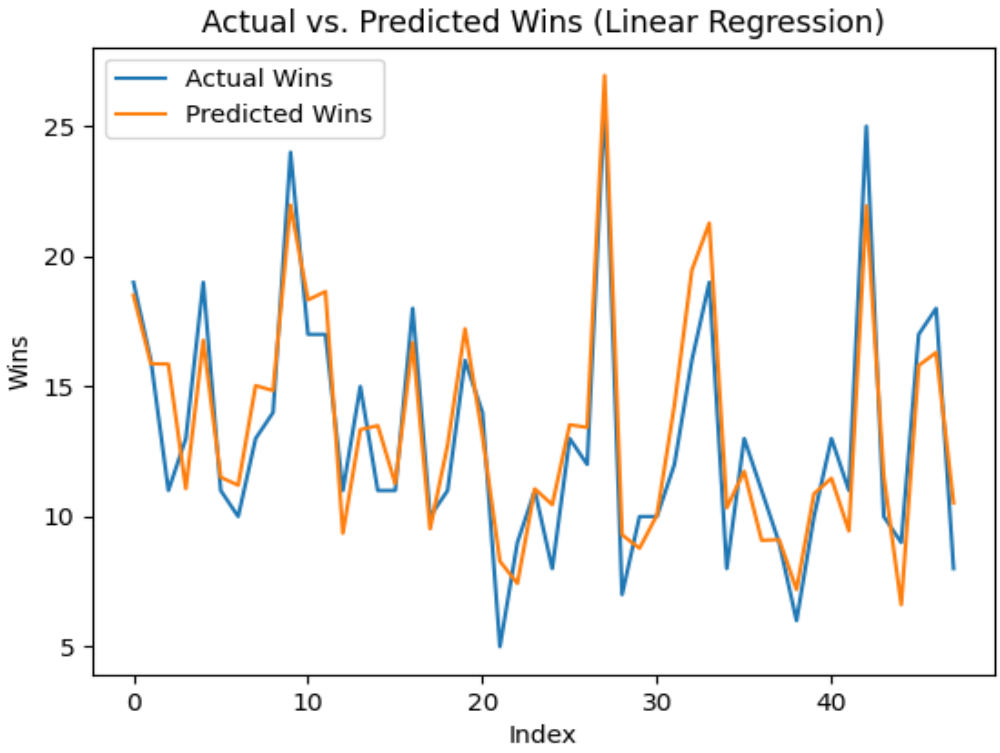
+ MSE: 3.42 >< 3.40

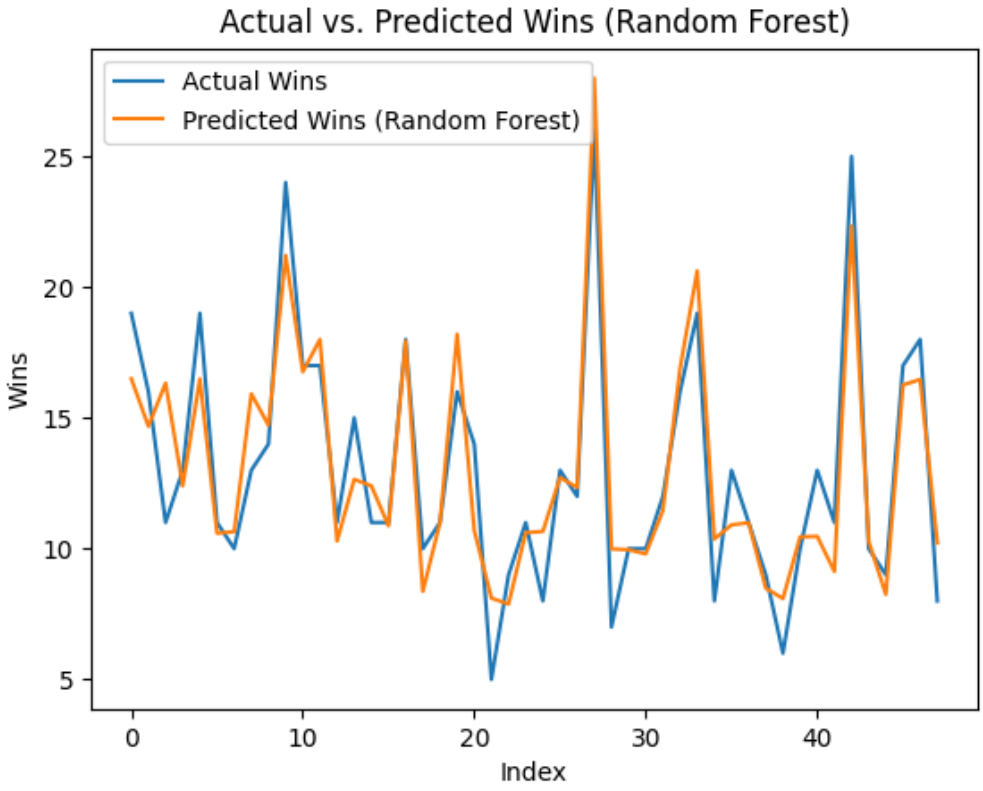
+ RMSE: 1.85 >< 1.84

About difference between 2 function in chart, we can easily see it with our eyes:

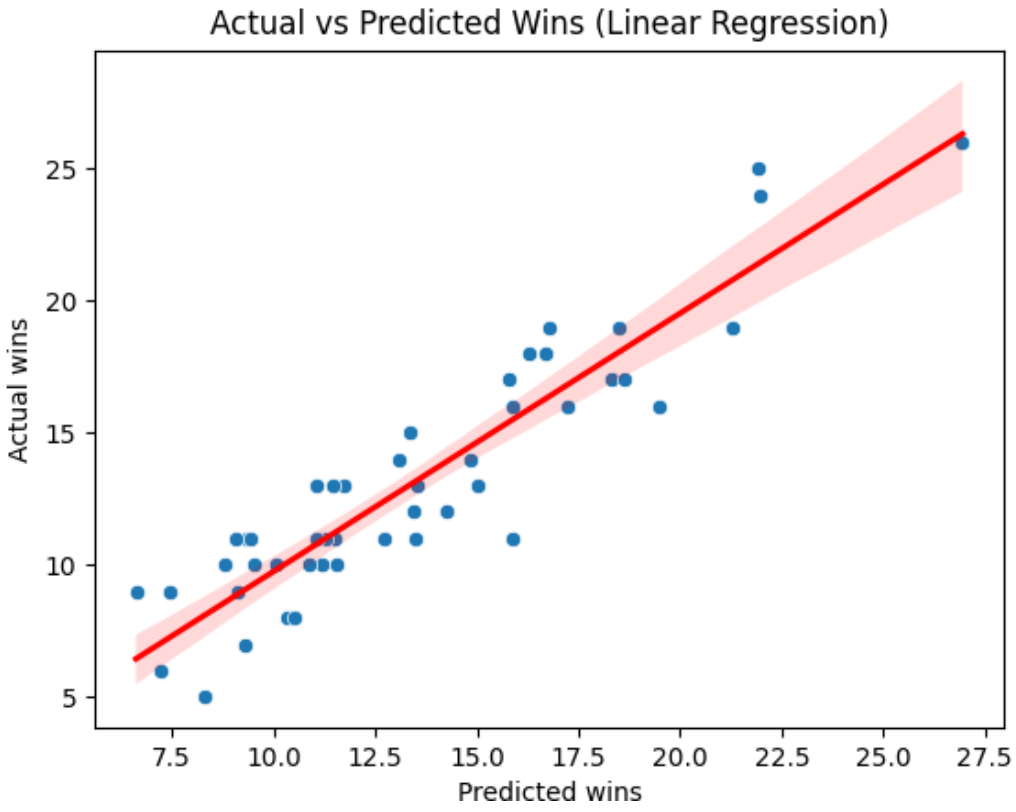


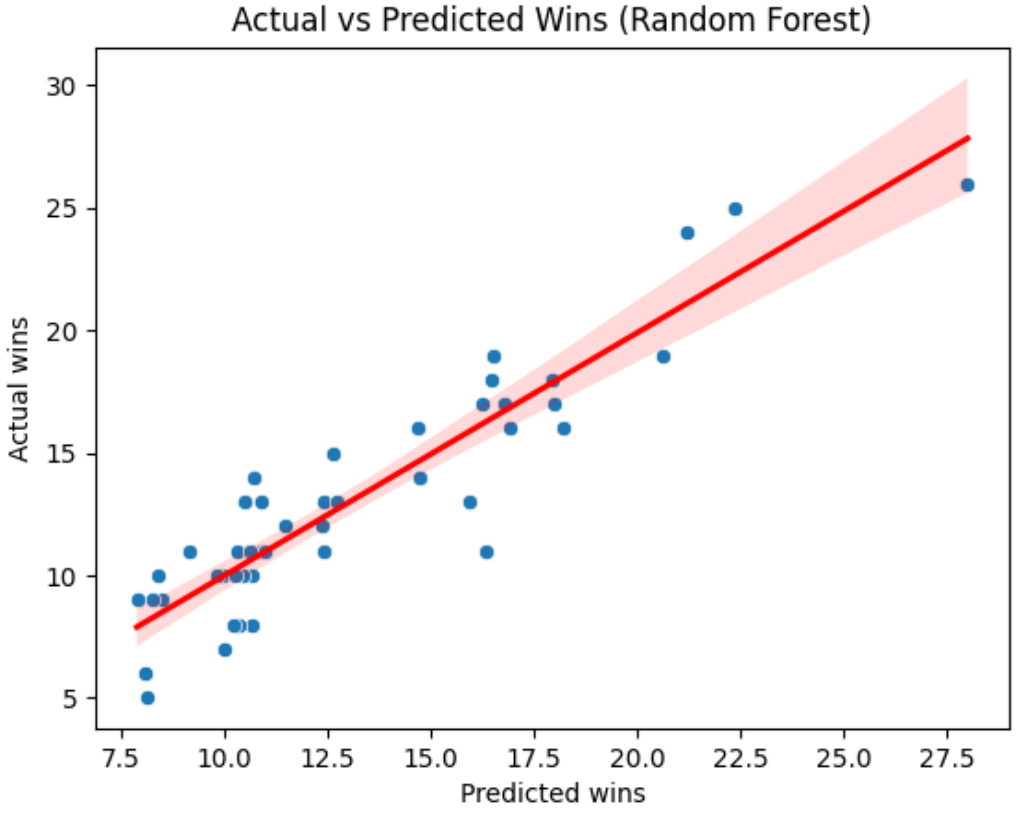



Actual vs. Predicted Wins (Linear Regression)
Actual Wins
Predicted Wins
Wins
Index
0
10
20
30
40
5
10
15
20
25


Actual vs. Predicted Wins (Random Forest)
Actual Wins
Predicted Wins (Random Forest)
Wins
Index
0
10
20
30
40
5
10
15
20
25

- Scatter plot:





VII. Conclusion:

1. Key Findings:

The machine learning models demonstrated effectiveness in predicting EPL wins, with linear regression offering superior performance.

2. Recommendations:

Further refinement of features and exploration of advanced modeling techniques could enhance predictive accuracy.

3. Future Work:

Future research could focus on incorporating additional data sources and refining the models to predict other performance metrics.

VIII. Code Implementation:

The provided code includes data loading, preprocessing, model training, evaluation, and visualization using Python libraries such as pandas, scikit-learn, seaborn, and matplotlib. Detailed comments within the code explain each step and its relevance to the machine learning project.

Code:

```
import numpy as np

import pandas as pd

from sklearn.linear_model import LogisticRegression, LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, mean_squared_error, accuracy_score, precision_score, recall_score, f1_score

from sklearn.impute import SimpleImputer

import seaborn as sns

import time

import matplotlib.pyplot as plt


start_time = time.time()


# Custom assert functions
```

```
def assert_is_not_none(obj):
    assert obj is not None, "Object is None"

def assert_equal(a, b):
    assert a == b, f"{a} != {b}"

# Load data
league_data = pd.read_csv("data/results.csv")
stats = pd.read_csv('data/stats.csv')
standings = pd.read_csv('data/EPL Standings 2000-2022.csv')
with_goalscorers = pd.read_csv('data/with_goalscorers.csv')

# Assertions
assert_is_not_none(league_data)
assert_is_not_none(stats)
assert_is_not_none(standings)
assert_is_not_none(with_goalscorers)

# Export league_data to Excel
league_data.to_excel("league_data.xlsx", index=False)
print(league_data)
```

```
# Check data integrity
expected_columns = ['home_team', 'away_team', 'home_goals', 'away_goals', 'result', 'season']
expected_shape = (4560, 6)
assert_equal(list(league_data.columns), expected_columns)
assert_equal(league_data.shape, expected_shape)

# Filter standings data
new_standings = standings[standings['Season'].between('2006-07', '2017-18')]
new_standings.to_excel("new_standings.xlsx", index=False)
print(new_standings[new_standings.Pos == 1])

# Clean new_standings
new_standings.reset_index(drop=True, inplace=True)
new_standings = new_standings.drop('Qualification or relegation', axis=1)
print(new_standings)
assert_equal(new_standings.shape, (240, 11))

# Process with_goalscorers
with_goalscorers[['Top Scorer Goals', 'Top Scorer Team']] = with_goalscorers['Top Scorer'].str.extract(r'(\d+)\((.*?)\)$')
```

```
with_goalscorers.drop('Top Scorer', axis=1, inplace=True)
with_goalscorers['Top Scorer Goals'] = with_goalscorers['Top Scorer Goals'].astype(int)
with_goalscorers.drop('# Squads', axis=1, inplace=True)
with_goalscorers_sorted = with_goalscorers[with_goalscorers.Season.between('2006-2007', '2017-2018')]
with_goalscorers.to_excel("with_goalscorers.xlsx", index=False)
assert_equal(with_goalscorers_sorted.shape, (12, 5))

# Data Analysis
champions = new_standings[new_standings.Pos == 1]
champions.reset_index(drop=True, inplace=True)
assert_equal(champions.shape, (12, 11))

most_goals_season = stats.loc[stats.goals.idxmax()][['team', 'goals', 'season']]
print(most_goals_season)

non_champions = new_standings[new_standings['Pos'] != 1]
non_champions.reset_index(drop=True, inplace=True)
most_goals_non_champion = non_champions.nlargest(1, 'GF')
print(most_goals_non_champion[['Team', 'GF', 'Season', 'Pts']])
```

```
champions_scored_less_1 = champions[champions['GF'] <
non_champions['GF'].max()]

var = champions_scored_less_1.shape[0]

champions_scored_more = champions[champions.GF >
non_champions.GF.max()]

print(champions_scored_more)


with_goalscorers_sorted.sort_values(by='Season', inplace=True)

with_goalscorers_sorted.reset_index(drop=True, inplace=True)

print(with_goalscorers_sorted)


goalscorer_champions =
with_goalscorers_sorted[with_goalscorers_sorted.apply(lambda row:
str(row['Champion']) in str(row['Top Scorer Team']), axis=1)]


# Verify the length of the intended slice

slice_length = len(with_goalscorers_sorted[14:26])

if slice_length == 0:

    percentage_tg_ch = 0

else:

    percentage_tg_ch = len(goalscorer_champions) / slice_length * 100

percentage_formatted = f"{percentage_tg_ch:.1f}%"
```

```
print(f"The percentage of the champions have had the single top goalscorer: {percentage_formatted}")


second_placed = new_standings[new_standings.Pos == 2]
second_placed.reset_index(drop=True, inplace=True)
winning_margin = champions.Pts - second_placed.Pts
champions = champions.assign(Winning_margin=winning_margin.values)
print(champions)


# Plots
plt.plot(champions.Season, champions.Winning_margin)
plt.xlabel('Season')
plt.ylabel('Winning margin')
plt.xticks(rotation=45)
plt.show()


times_champion = champions.Team.value_counts()
teams = times_champion.index.to_list()
plt.pie(times_champion, labels=teams, autopct='%1.1f%%', colors=['red', 'royalblue', 'skyblue', 'white'])
plt.title('Titles won from 2006-07 to 2017-18')
plt.show()
```

```
# Modeling
y = stats['wins']
X = stats.drop(['wins', 'season', 'team'], axis=1)

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

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

# Linear Regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

actual_predicted = pd.DataFrame({'Actual wins': y_test.squeeze(),
'Predicted wins': y_pred.squeeze()}).reset_index(drop=True)
print(actual_predicted.head())

# Performance metrics
```

```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f'Mean absolute error: {mae:.2f}')
print(f'Mean squared error: {mse:.2f}')
print(f'Root mean squared error: {rmse:.2f}\n\n')

# Logistic Regression model
logistic_regressor = LogisticRegression(max_iter=2000, solver='saga')
logistic_regressor.fit(X_train, y_train)
y_pred_logistic = logistic_regressor.predict(X_test)

# Performance metrics for Logistic Regression
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
precision_logistic = precision_score(y_test, y_pred_logistic,
average='weighted', zero_division=1)
recall_logistic = recall_score(y_test, y_pred_logistic,
average='weighted', zero_division=1)
f1_logistic = f1_score(y_test, y_pred_logistic, average='weighted',
zero_division=1)
print("Logistic Regression Evaluation Metrics:")
print(f"Accuracy: {accuracy_logistic:.2f}")
print(f"Precision: {precision_logistic:.2f}")
```

```
print(f"Recall: {recall_logistic:.2f}")
print(f"F1 Score: {f1_logistic:.2f}")
print()

# Print evaluation metrics for Linear Regression
print("Linear Regression Evaluation Metrics:")
print(f'Mean absolute error: {mae:.2f}')
print(f'Mean squared error: {mse:.2f}')
print(f'Root mean squared error: {rmse:.2f}\n\n')
print(regressor.score(X_test, y_test))

# Elapsed time
end_time = time.time()
elapsed_time = end_time - start_time
print(f"[Elapsed time: {elapsed_time} seconds]\n")

# Create a line plot for actual wins vs predicted wins
plt.plot(actual_predicted.index, actual_predicted['Actual wins'],
label='Actual Wins')
plt.plot(actual_predicted.index, actual_predicted['Predicted wins'],
label='Predicted Wins')
plt.xlabel('Index')
plt.ylabel('Wins')
```

```
plt.title('Actual vs. Predicted Wins (Linear Regression)')
plt.legend()
plt.show()

# Scatter plot for Linear Regression
sns.scatterplot(x='Predicted wins', y='Actual wins',
data=actual_predicted)
sns.regplot(x='Predicted wins', y='Actual wins', data=actual_predicted,
scatter=False, color='red')
plt.xlabel('Predicted wins')
plt.ylabel('Actual wins')
plt.title('Actual vs Predicted Wins (Linear Regression)')
plt.show()

# Random Forest Regressor model
random_forest_regressor = RandomForestRegressor(random_state=42)
random_forest_regressor.fit(X_train, y_train)
y_pred_rf = random_forest_regressor.predict(X_test)

# Performance metrics for Random Forest Regressor
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
```

```
print("Random Forest Regressor Evaluation Metrics:")
print(f'Mean absolute error: {mae_rf:.2f}')
print(f'Mean squared error: {mse_rf:.2f}')
print(f'Root mean squared error: {rmse_rf:.2f}\n\n')

# Create a DataFrame with actual and predicted values for Random Forest
actual_predicted_rf = pd.DataFrame({'Actual wins': y_test.squeeze(), 'Predicted wins': y_pred_rf.squeeze()}).reset_index(drop=True)

# Create a line plot for actual wins vs predicted wins (Random Forest)
plt.plot(actual_predicted_rf.index, actual_predicted_rf['Actual wins'], label='Actual Wins')
plt.plot(actual_predicted_rf.index, actual_predicted_rf['Predicted wins'], label='Predicted Wins (Random Forest)')
plt.xlabel('Index')
plt.ylabel('Wins')
plt.title('Actual vs. Predicted Wins (Random Forest)')
plt.legend()
plt.show()

# Scatter plot for Random Forest
```

```
sns.scatterplot(x='Predicted wins', y='Actual wins',
data=actual_predicted_rf)

sns.regplot(x='Predicted wins', y='Actual wins',
data=actual_predicted_rf, scatter=False, color='red')

plt.xlabel('Predicted wins')

plt.ylabel('Actual wins')

plt.title('Actual vs Predicted Wins (Random Forest)')

plt.show()


# Final elapsed time

end_time = time.time()

elapsed_time = end_time - start_time

print(f"[Elapsed time: {elapsed_time} seconds]\n")


# Feature importance

feature_importance =
pd.Series(random_forest_regressor.feature_importances_,
index=X.columns).sort_values(ascending=False)

print(feature_importance)
```

IX. References:

- Datasets: "https://github.com/RobertRusev/ML-Premier-League-Wins-Predictor/tree/main/data"

- Libraries: `pandas`, `numpy`, `scikit-learn`, `seaborn`, `matplotlib`, `Pylance`.

- Documentation:

+ Official documentation of Python libraries and online tutorials for machine learning practices.

+ Rusev, R. (2023). ML Premier League Wins Predictor. GitHub repository. Retrieved from https://github.com/RobertRusev/ML-Premier-League-Wins-Predictor