# Predicting goals scored based on teams' season performances using machine learning

## 1. Introduction

Football is a game full of surprises, but in the last few years statistics and data analysis have become an important part of understanding team performances. With so much detailed match data available today, it is possible to use machine learning to look for patterns and even make predictions about future games. In the real world, machine learning methods are used by betting companies to calculate coefficients for events (Intellias, 2025)[5], but with this project we want to test and apply ML methods learnt during lectures to our passion: football.

## 2. Problem formulation

The aim of this project is to predict the number of goals a football team will score during a season using a dataset consisting of league results of different teams in the last 5 years. Football is changing every 5-6 years with new trends coming out like new schemes and tactics or rules changed by FIFA. Hence, we decided to cover 5 consecutive seasons.

In this project, the objective is to train and compare two different regression models; Polynomial Regression and Fully Connected Neural Networks (MLP).

**Polynomial Regression**

Polynomial regression is used to capture simple non-linear relationships between match statistics and goals in an interpretable way, possibly showing the correlation between the two. Football is also an emotion driven, and unpredictable sport, meaning the relationships between the two variables is more likely to be non-linear, possibly hinting at the issues that this model will deal with. Given that we will use one feature at a time, our hypothesis space is given by **Eq. 1** where **x** is our feature input values.

**Eq. 1 (Salami, Lecture 2) [1] -**

$$h(x, w) = w_0 + w_1 x + w^2{}_2 x + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

**Multilayer perceptron**

A multilayer perceptron (MLP) is a type of neural network which uses multiple layers of neurons. The neurons in a MLP typically use nonlinear activation functions, allowing the network to learn more complex patterns within data (Jaiswal, 2021) [4].

This method is used to model more complex patterns and interactions that polynomial regression potentially misses. Its hypothesis space is given by **Eq. 2** where **x** is our feature input and the **w** are the weights across the network.

**Eq. 2 (Salami, Lecture 6) [2] -**

$$h_k(x, w) = f^{(4)}{}_{act}\left(\sum_{k=1}^{D_3} w_{k1} f^{(3)}{}_{act}\left(\sum_{j=1}^{D_2} w_{jk}^{(2)} f^{(2)}{}_{act}\left(\sum_{i=1}^{D_1} w^{(1)}{}_{ij} x_i + w^{(1)}{}_{0j}\right) + w^{(2)}{}_{0k}\right) + w^{(3)}{}_{01}\right)$$

By comparing these two approaches, the project aims to evaluate which method gives better accuracy for goal prediction over a season and to analyze which match statistics have the strongest impact on goal scoring.

**Loss Function**

Given that one of our regression methods is Polynomial Regression, our loss function will be the mean squared error. The mean squared error loss function is optimal for penalizing larger errors and identifying which features are mainly impacting the model's accuracy.

Our second regression method (MLP) does not typically have an arbitrary loss function, but rather is dependent on the topic and case. Thus we have initially decided to use the mean squared error as our primary loss function. We will aim to minimize our mean squared error loss in order to maximize the functionality and accuracy of the model. Though not a loss function, we will also use $R^2$ score to determine the variance between data and the models predicted labels.

## 3. Model & Data

Due to the fact that our target is discrete (goals scored over a season), our model will be based on supervised learning. The model will try to learn the relationship between actual goals scored and the features, and then attempts to generalize it accordingly onto our test set.

We have been able to collect a dataset from Kaggle, by the user Sergi Lehkyi [3], which has football data from 2014 to 2019. The data consists primarily of league finishes each season for each team in the English, German, Russian, French, Spanish, and Italian leagues. Our dataset consists of 685 datapoints, and it was split into a **training set (80%)** and a **testing set (20%)** using train test split functions. The 80% split in favour of the training set was chosen so that the model can have enough data to learn the relationship between our features and label. The remaining 20% of the dataset is used as a test set to ensure that there is enough unseen data for evaluation of model performance.

**Feature Engineering & Selection**

Given that our dataset consisted of numerous features (21), we had to carefully select which ones to use for this project. We plotted scatter plots (Figure 3.3) in order to find potential relationships between the label (goals scored) and the features. The 'wins', 'pts', and 'deep' features hinted at potential positive relationships in the scatter plots, and hence we ended up choosing them as our features. Additionally, we added xGA as a feature, as a team with higher xGA is less likely to score because they must defend more. Because the data consists of matches played across different leagues, the amount of matches played by a team in a season varies. As a result, we engineered the win ratio feature in order to standardize the amount of wins, and avoid inconsistencies in the data.

Figure 3.1: Table of labels

| Label | Description | Type of Data |
|---|---|---|
| Goals scored across a season | Number of goals scored by a team during the season. | Discrete |

Figure 3.2: Table of features

| Feature | Description | Type of Data |
|---|---|---|
| Win ratio (engineered, Appendix B) | Ratio between the teams total wins and the total matches played by the team during the season. | Continuous |
| Deep | Passes completed within an estimated 20 yards of opponents goal (crosses excluded) | Discrete |
| xGA (expected conceded goals) | Like xG, xGA measures quality and quantity of scoring opportunities created against the team during the season. Higher xGA suggests a team was dominated more, having less opportunities to score. It should negatively affect the number of goals scored by a team. | Continuous |
| Pts (Points) | The measure of points obtained by a team during the season. | Discrete |

## 4. Results

**Polynomial regression**

Based on figures 4.1 and 4.2, polynomial degree 2 minimised the test mean squared error and calculated the greatest $R^2$ score relative to the other degrees. This indicates that the best trade-off between underfitting and overfitting was reached at degree 2. At higher degrees, the model became overfitted and inaccurate, as demonstrated by the simultaneous increase in test error and decrease in training error. Furthermore, the simplicity of the second degree meant that the model was much more generalizable to unseen data.

The model's accuracy is reflected in figure 4.3 by the $R^2$ score at 0.836, meaning the model was able to relatively accurately predict the amount of goals the team would score over a season, given the features. Despite the few anomalies in the data that could've stemmed from overperformances, 'lucky' goals, or underperformances, the mean squared error was effectively able to penalize it by not favouring these outliers during optimization.

**MLP Regression**

After running the MLP model with different layers and comparing them, it was apparent that a four-layer-architecture with 15 neurons each was the most optimal, as seen in figure 4.4. Using four layers made it possible for the MLP model to achieve the greatest $R^2$ score and the lowest testing error, as shown in the figures 4.4 and 4.5. Increasing the number of hidden layers evidently made the model become overfitted.

Figure 4.5 illustrates that some models with more layers resulted in worse results (lower $R^2$, greater testing error). This can be attributed to overfitting and the datasets size, containing only 685 datapoints. As a result, our MLP model's best values were $R^2 = 0.847$ and $MSE_{test} = 47.104$.

**Comparing results**

Next, the aim was to compare $R^2$ and testing error scores between the polynomial regression and MLP models.

$$R^2_{Polynomial} < R^2_{MLP} \Rightarrow 0.836 < 0.847$$
$$Test\ MSE_{Polynomial} > Test\ MSE_{MLP} \Rightarrow 50.75 > 47.104$$

As demonstrated, the MLP model had greater $R^2$ score, which means it understands better variance in the amount of actual goals scored. The MLP model captured the nonlinear relationships between the features (such as win_ratio, deep, pts, xGA) and the target variable (predicted goals scored) more effectively, resulting in better generalization performance. Despite being a more interpretable model, the polynomial regression model failed to capture the complex patterns in footballing data to the same extent as MLP. Given that football is not exactly a non-linear sport, the MLP model had a higher chance of learning the relationship between variables because of the various types of patterns. However, because of our small dataset, it is plausible that the MLP memorized the patterns in our data rather than investigated the relationship between the features and labels. Although the MLP had slightly better values regarding the errors and $R^2$ losses, both models had their respective downsides to them.

## 5. Conclusion

Both **Polynomial Regression** and **Multilayer Perceptron (MLP) Regressor** were used to predict the number of goals a team scores based on key performance metrics such as number of win ratio, points, deep, and xGA. The Multilayer Perceptron method showed greater $R^2$ score , and lower testing MSE. However, it was essential to use a suitable amount of layers to avoid overfitting. The results showed that while Polynomial Regression offered a simpler relationship between features and the target variable, it was limited in capturing complex, non-linear dependencies that appear in football. The MLP Regressor, on the other hand, achieved higher predictive accuracy regarding the goals scored, as reflected by its lower error values. This demonstrates that the MLP model is more capable of modeling the intricate relationships within football performance data, better suiting our project. However, the polynomial regression model was still able to predict the goals scored by a team during a season relatively accurately.

## 6. Evaluation

Despite the fact that our MLP and Polynomial regression programmes worked, our project could still be developed in certain ways. Firstly, the dataset size could be massively increased. Because one of our models was the MLP, which typically requires larger datasets to be trained and tested, having 685 datapoints limited the model. Additionally, to split our data we only used the train_test_split functions, meaning that the splits were done at random and differed for each degree and layer. The combination of a small dataset and random splitting could potentially have prevented our test sets from representing the underlying distribution in the data, possibly limiting our models. In this case, the K-cross fold validation could've suited our data.

This project can also be expanded to other models, such as the poisson regression, which better handles counts of data, similar to goals in football. All in all, our models were relatively successful at learning from the data, with the MLP model holding a slight edge in the results.

References


[1] - Salami, Dariush. *Lecture 2: Regression.* Aalto University, 2025. https://mycourses.aalto.fi/pluginfile.php/2537089/mod_resource/content/5/Slides_MachineLearningD_2025_Regression.pdf

[2] - Salami, Dariush. *Lecture 6: DeepLearning.* Aalto University, 2025. https://mycourses.aalto.fi/pluginfile.php/2547532/mod_resource/content/10/MachineLearningD_2025_DeepLearning.pdf

[3] - Lekhyi, Sergi. *Football Data: Expected Goals and Other Metrics.* Kaggle, 2023. https://www.kaggle.com/datasets/slehkyi/extended-football-stats-for-european-leagues-xg

[4] - Jaiswal, Sejal. *Multilayer Perceptrons in Machine Learning: A Comprehensive Guide.* Datacamp, 2021, updated 5 April 2025. https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning

[5] - Intellias. *Raising the Stakes: Top 5 Use Cases for Machine Learning in Sports Betting.* April 4, 2025. Updated July 10, 2025. https://intellias.com/machine-learning-for-sports-betting/

[6] - OpenAI. *ChatGPT*. 07 October, 2025. Used for code. https://chat.openai.com/

Appendix A

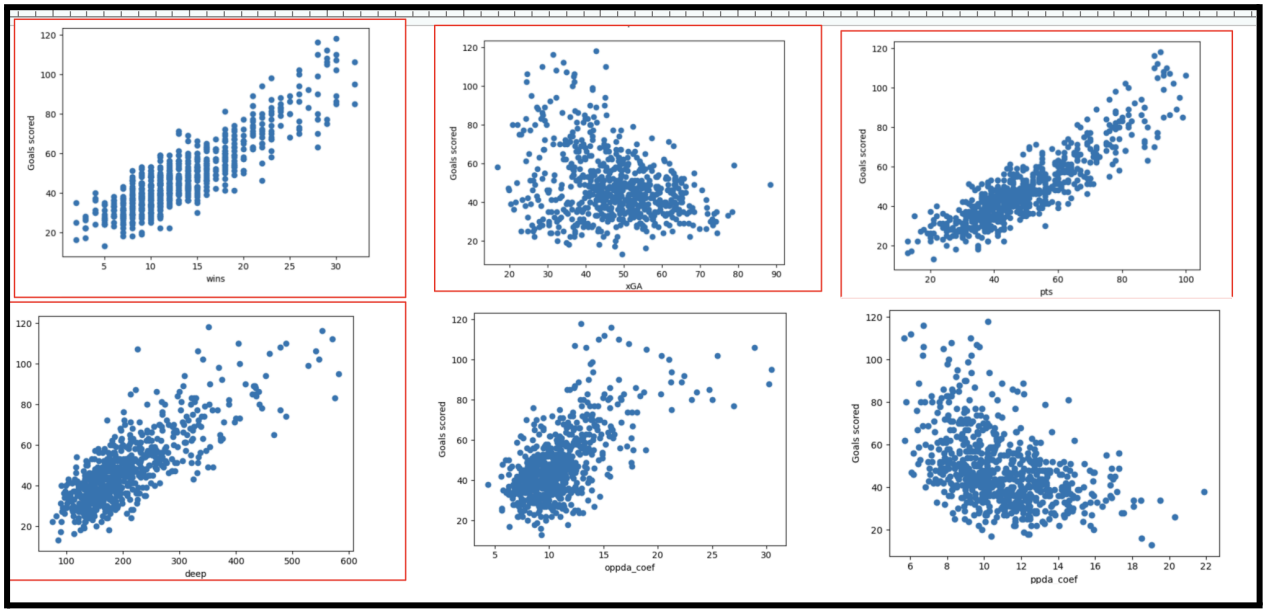


Figure 3.3: Scatter plots

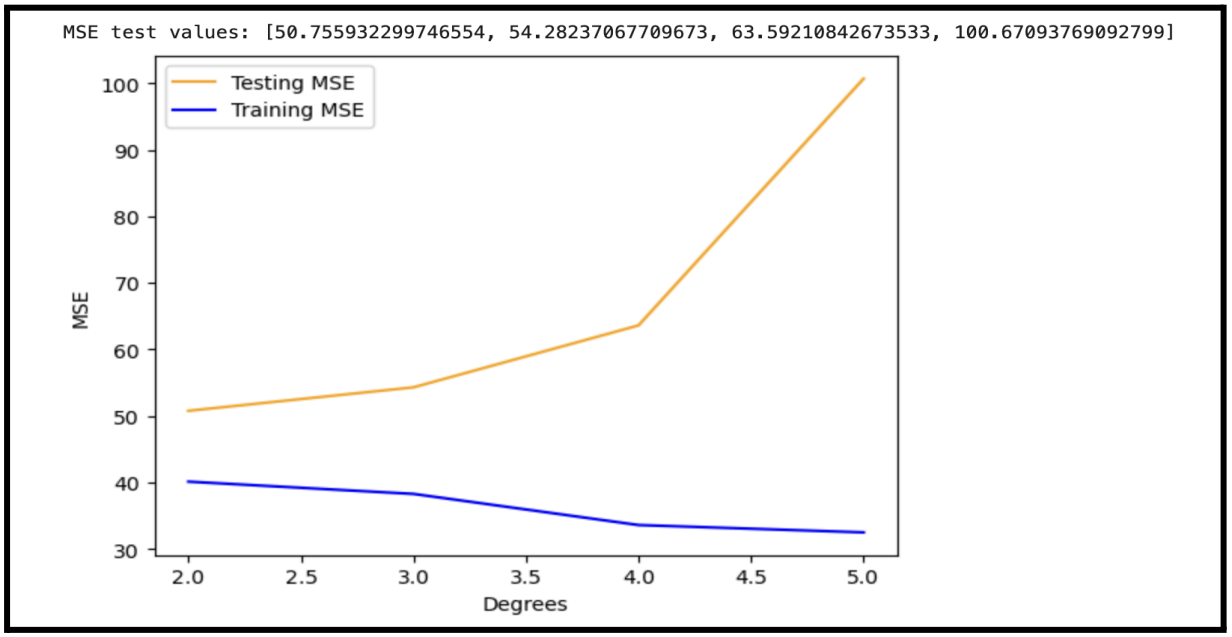


Figure 4.1: Plot of polynomial MSE vs degrees

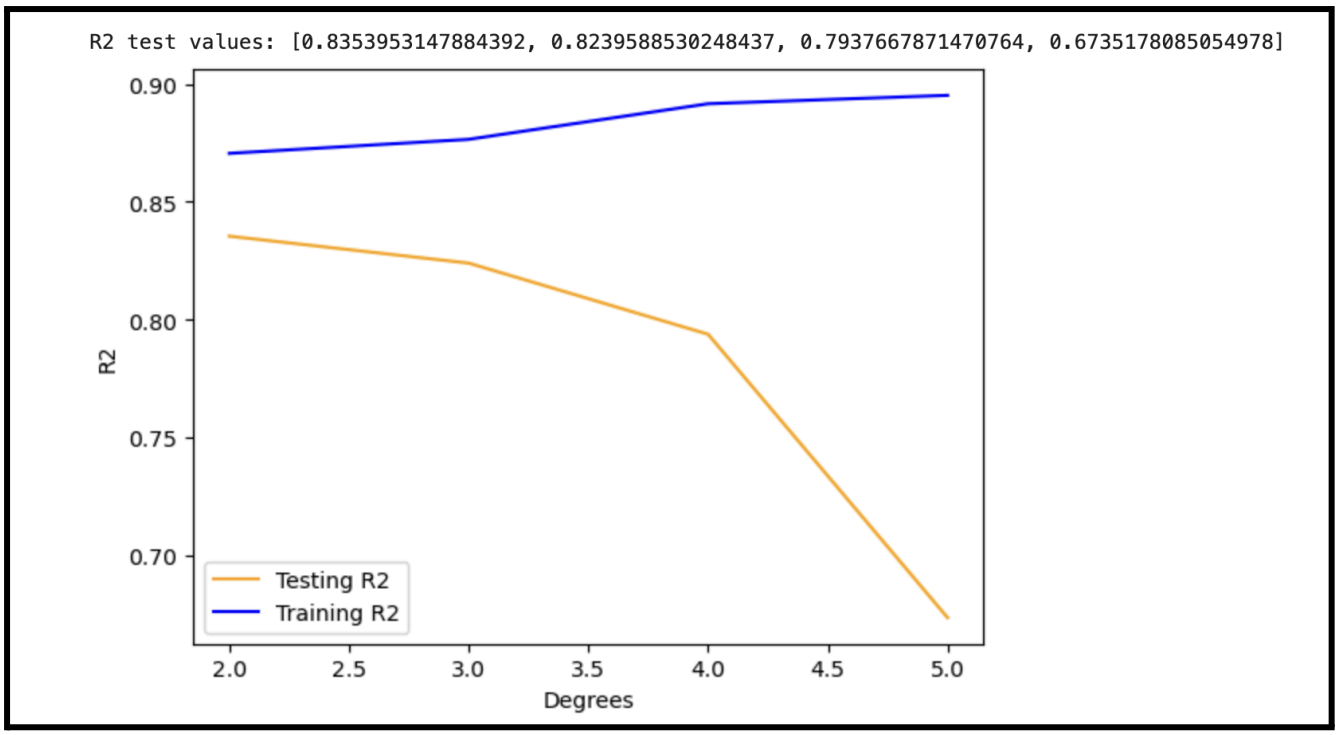


Figure 4.2: Plot of polynomial $R^2$ vs degrees

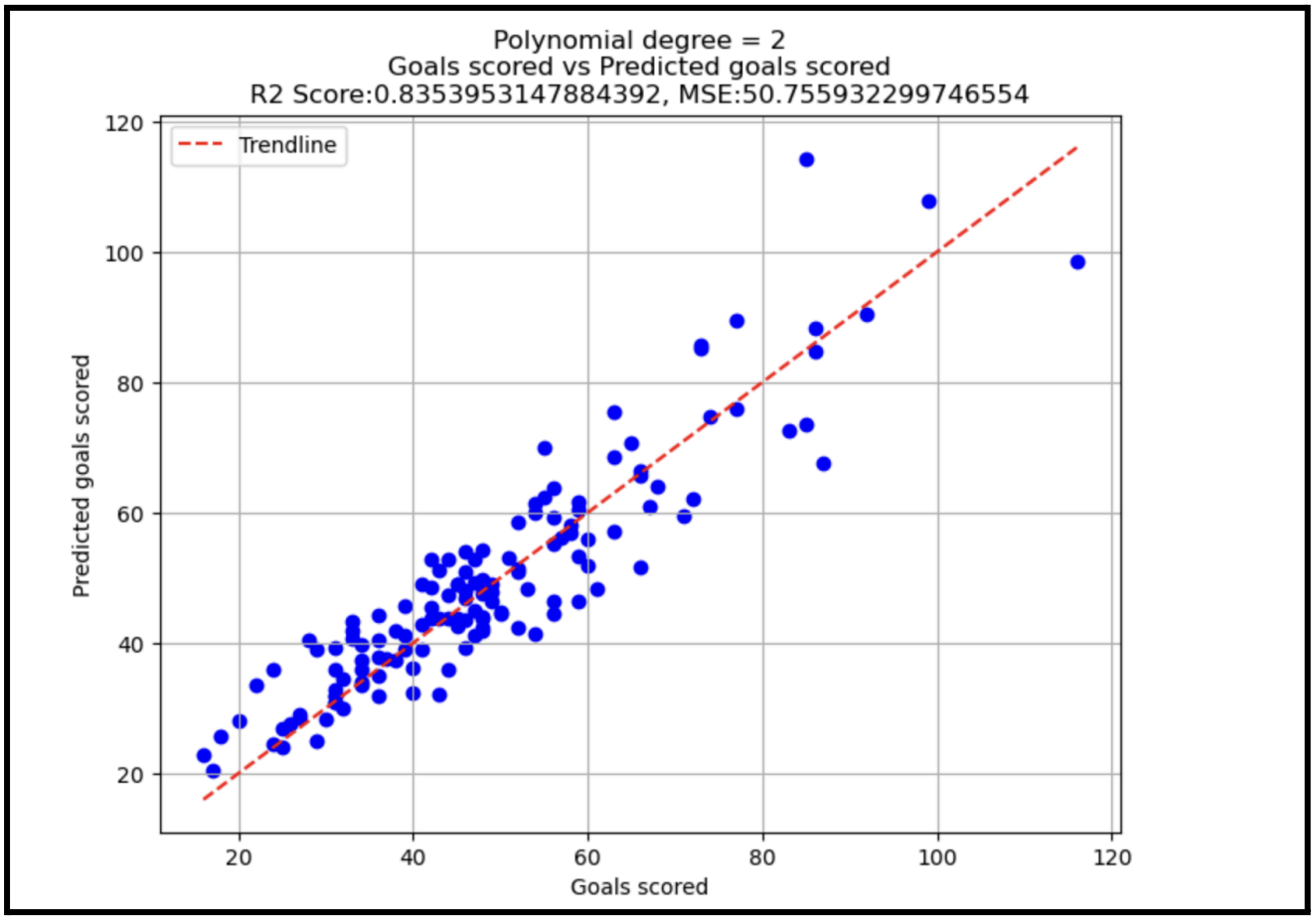


Figure 4.3: Plot of polynomial models predicted goals scored over the season against the actual goals scored

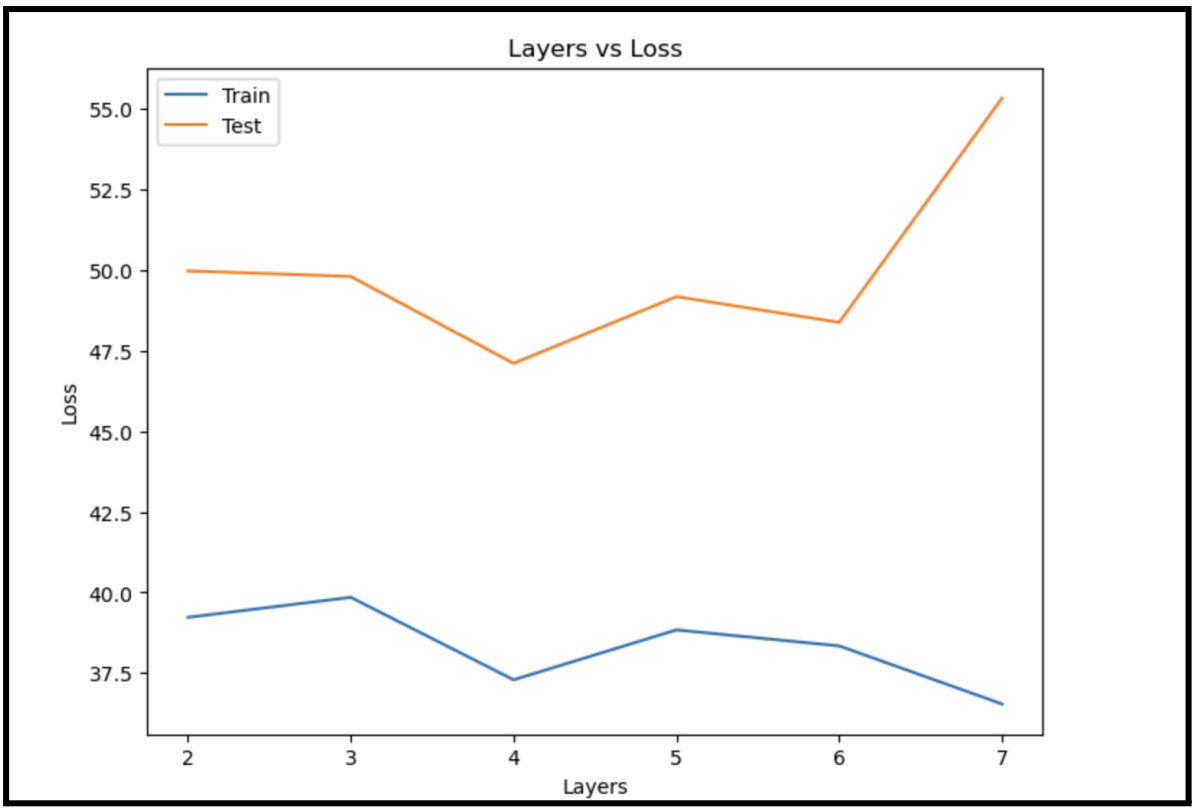


Figure 4.4: Plots of MLP test vs training loss

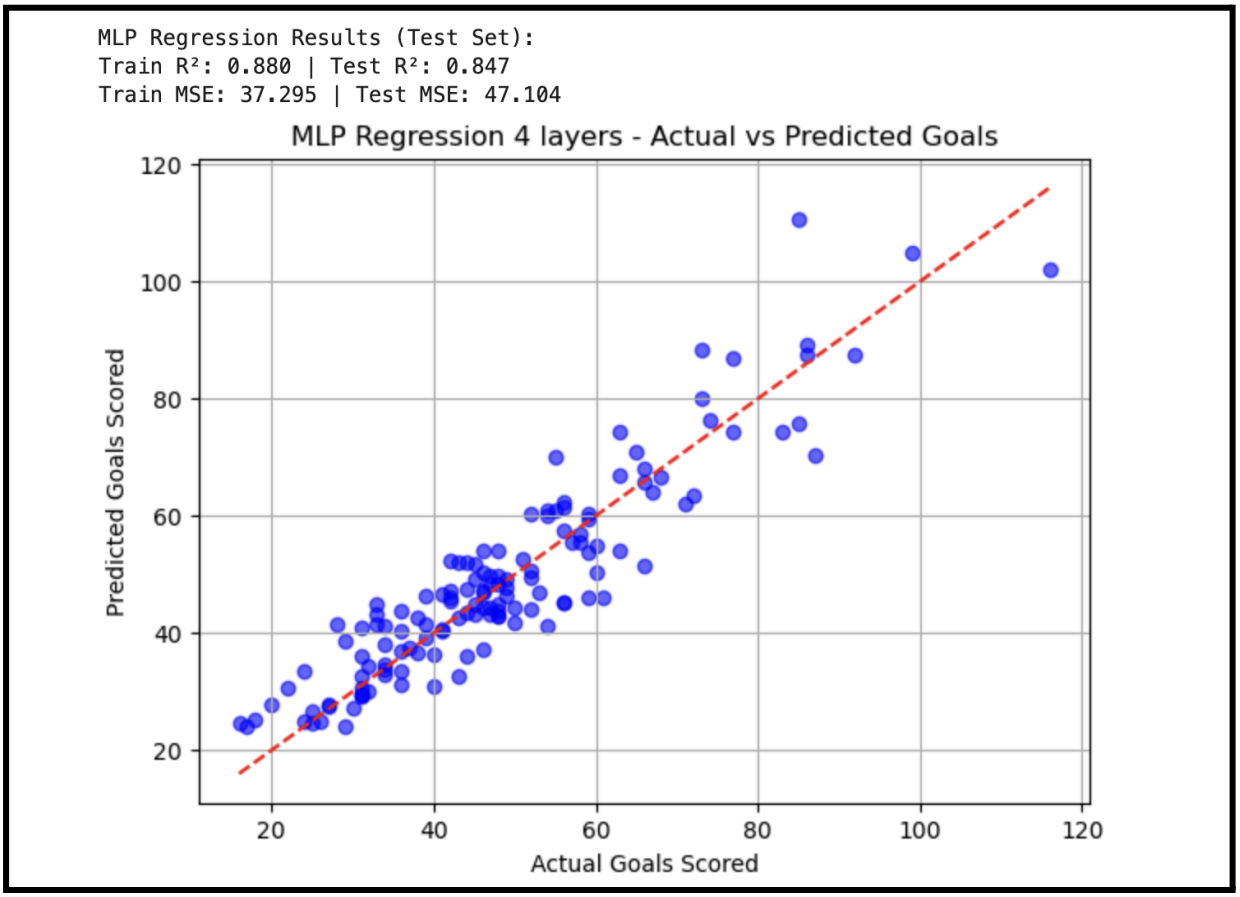


Figure 4.5: the MLP models predicted goals against the actual goals scored over the season by a team.

Appendix B

```
#Imported all required libraries and functions for our project at once
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split


dataframe= pd.read_csv("understat.csv")

dataframe.head(685)
#Dropping some features we knew we would not use
dataframe = dataframe.drop(['position','team','Unnamed: 0',
               'Unnamed: 1', 'loses',
               'missed', 'xG', 'xG_diff', 'npxG',],axis=1)

#Scatter plotting for the rest of the features to potentially find hints at relationships
for column in dataframe.columns:
    if column != 'scored':
        X = dataframe[column].to_numpy()
        y = dataframe['scored'].to_numpy()
        plt.scatter(X,y)
        plt.ylabel("Goals scored")
        plt.xlabel(f"{column}")
        plt.show()
```

```
#Engineered our 4th feature: win ratio
dataframe['win_ratio'] = dataframe['wins'] / dataframe ['matches']

dataframe.head()
```

| | matches | wins | draws | scored | pts | xGA | xGA_diff | npxGA | npxGD | ppda_coef | oppda_coef | deep | deep_allowed | xpts | xpts_diff | win_ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 38 | 30 | 4 | 110 | 94 | 28.444293 | 7.444293 | 24.727907 | 73.049305 | 5.683535 | 16.367593 | 489 | 114 | 94.0813 | 0.0813 | 0.789474 |
| **1** | 38 | 30 | 2 | 118 | 92 | 42.607198 | 4.607198 | 38.890805 | 47.213090 | 10.209085 | 12.929510 | 351 | 153 | 81.7489 | -10.2511 | 0.789474 |
| **2** | 38 | 23 | 9 | 67 | 78 | 29.069107 | 0.069107 | 26.839271 | 25.748737 | 8.982028 | 9.237091 | 197 | 123 | 73.1353 | -4.8647 | 0.605263 |
| **3** | 38 | 22 | 11 | 70 | 77 | 39.392572 | 7.392572 | 33.446477 | 16.257501 | 8.709827 | 7.870225 | 203 | 172 | 63.7068 | -13.2932 | 0.578947 |
| **4** | 38 | 23 | 7 | 71 | 76 | 47.862742 | 2.862742 | 41.916529 | 20.178070 | 8.276148 | 9.477805 | 305 | 168 | 67.3867 | -8.6133 | 0.605263 |

```python
#Creating the polynomial regression model
#Testing for degrees
degrees = [2,3,4,5]
y = dataframe['scored'].to_numpy()
X = dataframe[['win_ratio', 'xGA', 'pts', 'deep']].to_numpy()

#To gather all the testing & training errors, in order to graph later
MSE_test_list = []
MSE_train_list = []
R2_test_list = []
R2_train_list = []

for degree in degrees:
    #Splitting data, splits the data for each degree
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

    poly = PolynomialFeatures(degree)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.fit_transform(X_test)

    regr = LinearRegression()
    regr.fit(X_train_poly,y_train)

    #Calculating predicted values for both the training & testing set
    #In this way we can
    y_pred_train = regr.predict(X_train_poly)
    y_pred_test = regr.predict(X_test_poly)

    #Calculating the mean_squared_error & r2_score for each set
    MSE_test = mean_squared_error(y_test, y_pred_test)
    MSE_train = mean_squared_error(y_train, y_pred_train)
    r2_train = r2_score(y_train,y_pred_train)
    r2_test = r2_score(y_test, y_pred_test)

    #Appending it to the list, with which it will be graphed later
    MSE_test_list.append(MSE_test)
    MSE_train_list.append(MSE_train)
    R2_test_list.append(r2_test)
    R2_train_list.append(r2_train)
```

```python
    #Plotting figures for polynomial model
    plt.figure(figsize=(8,6))
    plt.scatter(y_test,y_pred_test, color ='b')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
             'r--', label='Trendline') #(OpenAI, 2025) [6] - used for this specific line, as
                                       #we could not get the trendline to fit the whole data
    plt.xlabel('Goals scored')
    plt.ylabel('Predicted goals scored')
    plt.legend(loc="best")
    plt.title(f"Polynomial degree = {degree}\nGoals scored vs Predicted goals scored\nR2 Score:{r2_test}, MSE:{MSE_test}")
    plt.grid(True)
    plt.show()

#Plotting the MSE & R2 values in order to choose the best degree for our model
print(f'MSE test values: {MSE_test_list}')
plt.plot(degrees, MSE_test_list, color='orange', label='Testing MSE')
plt.plot(degrees, MSE_train_list, color='b', label='Training MSE')
plt.ylabel("MSE")
plt.xlabel("Degrees")
plt.legend()
plt.show()

print(f'R2 test values: {R2_test_list}')
plt.plot(degrees, R2_test_list, color ='orange', label='Testing R2')
plt.plot(degrees,R2_train_list, color='b', label='Training R2')
plt.ylabel("R2")
plt.xlabel("Degrees")
plt.legend()
plt.show()
```

```python
#Creating the MLP model
num_layers = [2,3,4,5,6,7]
#Number of neurons was tested with different values, but eventually, with recommendation from OpenAI, 15 was
#the value we settled on. (OpenAI, 2025)[6].
num_neurons = 15
list_mse_train = []
list_mse_test = []
r2_error_list = []

for i, num in enumerate(num_layers):
    #Tested with different number of layers, in order to find optimal amount
    hidden_layer_sizes = tuple([num_neurons]*num)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

    #Scaled this set because of its proneness to differences in range. Made the model operate more swiftly
    #and sped up learning.
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    #Used standard activation functions, as well as solvers
    #Max iter was 5000 to ensure convergence
    mlp = MLPRegressor(hidden_layer_sizes, activation='relu', solver='adam', max_iter= 5000, random_state=42)

    mlp.fit(X_train_scaled, y_train)

    y_train_pred = mlp.predict(X_train_scaled)
    y_test_pred = mlp.predict(X_test_scaled)

    #Calculated r2 and MSE for both training and test sets
    train_mse = mean_squared_error(y_train,y_train_pred)
    train_r2 = r2_score(y_train,y_train_pred)

    test_mse = mean_squared_error(y_test,y_test_pred)
    test_r2 = r2_score(y_test,y_test_pred)

    list_mse_train.append(train_mse)
    list_mse_test.append(test_mse)
    r2_error_list.append(test_r2)
```

```python
    print("MLP Regression Results (Test Set):")
    print(f"Train R²: {train_r2:.3f} | Test R²: {test_r2:.3f}")
    print(f"Train MSE: {train_mse:.3f} | Test MSE: {test_mse:.3f}")

    #Plotted the results of the model
    plt.figure(figsize=(7,5))
    plt.scatter(y_test, y_test_pred, color='blue', alpha=0.6)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--') #(OpenAI, 2025) [6] - used for this specific line, as
                                                                                #we could not get the trendline to fit the whole data
    plt.xlabel("Actual Goals Scored")
    plt.ylabel("Predicted Goals Scored")
    plt.title(f"MLP Regression {len(hidden_layer_sizes)} layers - Actual vs Predicted Goals")
    plt.grid(True)
    plt.show()

#Here we plotted the test vs train MSE, in order to distinguish the most optimal number
#of layers, from the rest.
plt.figure(figsize=(8,6))
plt.plot(num_layers,list_mse_train,label = "Train")
plt.plot(num_layers,list_mse_test,label = "Test")
plt.xticks(num_layers)
plt.legend(loc="upper left")
plt.xlabel('Layers')
plt.ylabel('Loss')
plt.title(f'Layers vs Loss')
plt.show()
```