Predicting the best Fantasy premier league lineup for the upcoming gameweek

March 2022

1 Introduction

Although I don't really like to play football, I love watching them with my friends. They introduced me to this mini-game called "Fantasy premier league", which is based on the highest level English football league - The premier league. The player has to calculate and predict which 11 footballers will do the best in the next gameweek. The total points of the player will be the sum of all the points of 11 footballers they chose in that gameweek. After that, the player can compare their points to other players to know how well they did in that gameweek. The problem is I'm not very good at the game and my lineup rarely does well. For that reason, I want to predict the lineup that will perform the best for the next gameweek by predicting the earned points of each footballer and sorting them to see which ones are likely to get high points.

Section 2 discusses the problem formulation. Then **section 3** examines the methods chosen for this machine learning problem. After that, the result is presented in **section 4**. Based on the found results, **section 5** gives the conclusions of this report. Finally, a bibliography and an appendix are given at the end of the report - **section 6** and **7**.

2 Problem formulation

Each individual data point is the statistics of a footballer in a gameweek that he participated in. The number of data points is currently 18737 which is the combined number of all the data entries of 713 footballers registered in the league since the start of the season. This machine learning problem has a single feature which is the **Total Minutes**. It is basically the number of minutes a footballer has played since the start of the season and the data type of it is integer. The label or the quantity of interest of this project is the predicted **Total Points** of a footballer after the next gameweek ends. The data type of the label is decimal. After finding that value, we can easily find the earned points by doing a simple subtraction:

$$\text{Points earned} = \text{New Total points value} - \text{Old total points value} \tag{1}$$

Basically, we apply the same procedure to all the footballers in the league to find the predicted points for each of them. Then we sort them in descending order to find out the best options for the next gameweek.

3 Methods

3.1 Data set

The dataset is extracted from this github repository: **https://github.com/vaastav/Fantasy-Premier-League**. As written in the previous section, the data set is the combination of all the data entries of 713 footballers registered in the league since the start of the season. The number of data points is currently 18737. At the previous 2 stages, I intended to have multiple features like

Goals, **Assists**, **Clean sheets**, etc. However, I realized that those features are also related to another subsystem of the game - the bonus points system. Consequently, they greatly increase the complexity of this machine learning problem and that is definitely not what I aim for in this course. For that reason, I decided to have a single feature for this project which is the **Total minutes** - the number of minutes a footballer has played since the start of the season.

The motivation behind choosing it as the feature is simple. Similar to my old features (Goals, Assists, etc.), the total number of minutes played correlates to the total points that a footballer has gathered since the start of the season. A footballer who has a high number of minutes played has a vastly greater chance of earning higher points than those who has fewer to no minutes played on the field. Figure 2 shows the relationship between the total minutes played and the total points earned. The figure is a scatter plot: each point is an individual footballer plotted in the graph with total minutes as the x-axis and total points as the y-axis. As we can see, there is a clear trend in the scatter plot: the higher the amount of minutes a footballer has, the higher the points he has.

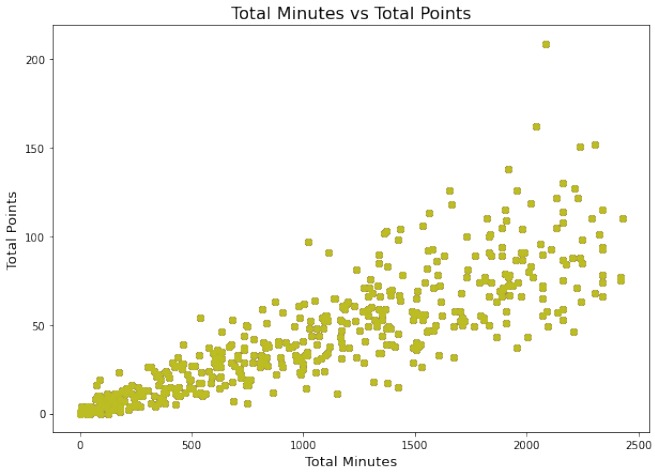


Figure 1: Data visualization of the dataset

Since I want to apply machine learning models to each footballer, data cleaning is required. Firstly, the data set is divided into 4 data sets based on the position of the footballers: GK (goalkeeper), FWD (forward), MID (midfielder) and DEF (defender). Secondly, I converted those 4 data sets into 4 dictionaries (a python data structure) for better navigation and observation. The keys of those dictionaries are the names of the footballers and the value are their stats. Finally, since a lot of footballers in the league are actually just Substitutes for the main line-up of their team and we wouldn't want to pick them in our lineup anyways, I removed all those who have less than 1350 total played minutes since the start of the season (equivalent to playing full duration in at least 15 gameweek). This report presents two machine learning models: linear regression and polynomial regression.

3.2 Data splitting

Unfortunately, I realized that my data set is smaller than I expected since I only took into account this current season. For that reason, each footballer only has between 15 to 30 data entries. After the cleaning, the amounts of footballers left are 202 (20 goalkeepers, 79 defenders, 80 midfielders, and 23 forwards) which results in 3030 to 6060 data points in total. As recommended by Ajitesh[1], I decided to do a double split for my data set and the ratio I chose to go with is 70:20:10. Using that ratio, each footballer would have around 10 to 21 training data points, 3 to 6 validation data points, and 2 to 3 testing data points.

3.3 Linear regression method

The first model I chose for my machine learning problem is the Linear regression model. The reason is due to its simplicity and this is also the machine learning method I'm most comfortable with. Also according to Figure (2), this ML method looks like it would provide a good fit for my data due to its linearity.

When I observed the fit model, I realized that there are a large number of outliers. As a result, I chose the Huber loss function due to its robustness against outliers [2]. The overall loss of the model is calculated by taking the average of all the losses of 202 footballers.

$$Loss = \frac{1}{N}\sum_{i=1}^{N} L_{Huber}((x_i, y_i), h)[3] \qquad \text{where N} = 202 \tag{2}$$

The linear regression is done by using the HuberRegressor class of the sklearn library [2].

3.4 Polynomial regression method

After I applied the Linear regression model, I realized that some of the scatter plots of the footballers show a non-linear relationship between the feature and the label. Because of that, I don't think that the linear regression model in the previous section would fit the data like I expected. For that reason, I decided to choose Polynomial regression as my second method for this project.

I chose MSE (mean squared error loss) as the loss function for this model since according to Alexander[3], Polynomial regression learns the hypothesis h by minimizing the average squared error loss. Similar to the Huber loss above, the overall loss of the model is also calculated by taking the average of all the MSE loss of 202 footballers:

$$Loss = \frac{1}{N}\sum_{i=1}^{N} L_{MSE}((x_i, y_i), h)[3] \qquad \text{where N} = 202 \tag{3}$$

The Polynomial regression is done by using the LinearRegression and PolynomialFeatures classes of the sklearn library [4][5].

4 Results

After trying to fit different orders of polynomials from 2 to 7, I decided to choose 5 as the degree of the polynomial regression model. The reason is that according to figure 2, degree of 5 provides the lowest training and validation errors out of the tested set of orders.

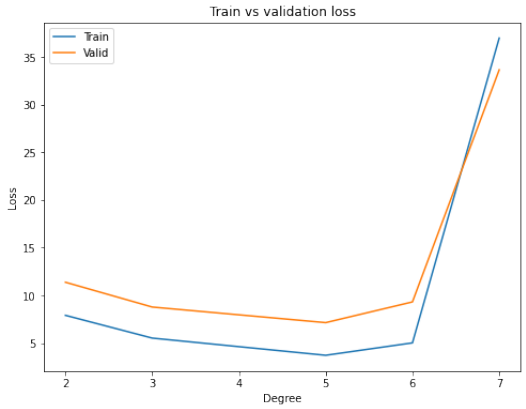


Figure 2: The train vs validation loss of each degree from 2 to 7

The training errors and validation errors of the 2 chosen machine learning models are presented in Table 1. Although the training errors are less than the validation errors in both models, they are relatively close to each other so there are not any signs of over-fitting. Based on the results, the Polynomial Regression model seems to be the better choice between the 2 chosen models. After all considerations, I decided to choose the Polynomial Regression model as the final model for my machine learning problem.

As mentioned in section 3.2, the ratio I chose is 70:20:10. After calculating the test errors for both models, we can clearly see that the test error for the polynomial regression, which is 6.53, is significantly lower than that of the Linear Regression model, which is 15.27.

Table 1: Comparison between 2 models

Method	Training Error	Validation Error	Test Error
Linear Regression	12.58	15.74	15.27
Polynomial Regression	3.33	7.75	6.53

Figures 3 and 4 provide the top 5 recommended footballers for each position. After observing both results, I saw that the result of the Polynomial Regression is much more believable than the result of the Linear Regression since the points of each footballer are generally between 2 to 10 points. Based on the 2 figures and table 1 above, not only does the Polynomial Regression model provide better errors but it also gives vastly superior recommendation, which is the ultimate aim for my project.

```
Recommended FWD:
Emmanuel Dennis  - Predicted points:  16.59  - Cost:  61
Armando Broja  - Predicted points:  12.99  - Cost:  55
Neal Maupay  - Predicted points:  10.72  - Cost:  65
Ollie Watkins  - Predicted points:  7.45  - Cost:  75
Joshua King  - Predicted points:  7.08  - Cost:  58
Richarlison de Andrade  - Predicted points:  6.51  - Cost:  75
Recommended MID:
Phil Foden  - Predicted points:  12.17  - Cost:  79
Raphael Dias Belloli  - Predicted points:  11.35  - Cost:  65
Bernardo Mota Veiga de Carvalho e Silva  - Predicted points:  10.56  - Cost:  72
Leandro Trossard  - Predicted points:  10.52  - Cost:  59
Conor Gallagher  - Predicted points:  9.35  - Cost:  61
Youri Tielemans  - Predicted points:  8.59  - Cost:  64
Recommended DEF:
Marc Cucurella  - Predicted points:  11.91  - Cost:  50
Jan Bednarek  - Predicted points:  11.53  - Cost:  44
Kieran Tierney  - Predicted points:  10.83  - Cost:  51
Romain Saiss  - Predicted points:  8.19  - Cost:  51
Grant Hanley  - Predicted points:  8.05  - Cost:  44
Charlie Taylor  - Predicted points:  8.02  - Cost:  44
Recommended GK:
Robert Sánchez  - Predicted points:  8.0  - Cost:  46
José Malheiro de Sá  - Predicted points:  7.19  - Cost:  53
Aaron Ramsdale  - Predicted points:  5.93  - Cost:  51
Illan Meslier  - Predicted points:  5.24  - Cost:  48
David de Gea  - Predicted points:  5.22  - Cost:  52
Lukasz Fabianski  - Predicted points:  5.09  - Cost:  50
```

Figure 3: Results of the Linear Regression model

```
Recommended FWD:
Cristiano Ronaldo dos Santos Aveiro  - Predicted points:  8.15  - Cost:  123
Jamie Vardy  - Predicted points:  8.09  - Cost:  103
Ivan Toney  - Predicted points:  7.17  - Cost:  66
Danny Ings  - Predicted points:  7.08  - Cost:  77
Emmanuel Dennis  - Predicted points:  6.36  - Cost:  61
Harry Kane  - Predicted points:  6.26  - Cost:  124
Recommended MID:
Diogo Jota  - Predicted points:  12.59  - Cost:  83
Mohamed Salah  - Predicted points:  11.86  - Cost:  133
Raheem Sterling  - Predicted points:  7.59  - Cost:  106
James Maddison  - Predicted points:  7.35  - Cost:  67
Nathan Redmond  - Predicted points:  6.93  - Cost:  59
Mason Mount  - Predicted points:  5.99  - Cost:  75
Recommended DEF:
Tino Livramento  - Predicted points:  7.19  - Cost:  44
Joel Matip  - Predicted points:  6.81  - Cost:  49
Trent Alexander-Arnold  - Predicted points:  6.54  - Cost:  85
Luke Ayling  - Predicted points:  6.02  - Cost:  43
Kieran Tierney  - Predicted points:  5.38  - Cost:  51
Jamaal Lascelles  - Predicted points:  4.95  - Cost:  44
Recommended GK:
Hugo Lloris  - Predicted points:  6.28  - Cost:  54
Vicente Guaita  - Predicted points:  4.96  - Cost:  46
Alisson Ramses Becker  - Predicted points:  4.07  - Cost:  60
Aaron Ramsdale  - Predicted points:  3.15  - Cost:  51
Edouard Mendy  - Predicted points:  3.08  - Cost:  61
Kasper Schmeichel  - Predicted points:  2.49  - Cost:  48
```

Figure 4: Results of the Polynomial Regression with degree 5 model

5 Conclusions

This report discussed 2 machine learning methods Linear Regression and Polynomial Regression of degree 5. After applying the 2 methods to predict the best Fantasy Premier League lineup for the upcoming gameweek, the result indicates that the Polynomial Regression model provides significantly better performance than the Linear Regression model. The average test error for the Polynomial Regression is 6.53, which is a fair result since the data set is not really large. There doesn't seem to be any signs of over-fitting or under-fitting because those values are fairly close to each other. However, improvement would be achieved by taking into account data from previous seasons instead of just this currently running season. This would increase the quantity of the data points significantly and thus provide much better results than what we have right now. For future directions, I consider using more features and by that, applying the bonus point system we discussed above in section 3.1 into the project. Combined with a larger data set, it would be interesting to see how well the same 2 models perform.

6 Bibliography

References

[1] Kumar Ajitesh. Machine learning – training, validation test data set. https://vitalflux.com/machine-learning-training-validation-test-data-set/, 2021. Accessed: 20122-03-10.

[2] sklearn. sklearn.linearmodel.huberregressor. https://scikit-learn.org/stable/modules/generated/sklearn.linear$_m$odel.$HuberRegressor.html, n.d. Accessed: 20122 - 03 - 30.$

[3] Jung Alexander. *Machine Learning: The Basics*. Springer, Singapore, 2022.

[4] sklearn. sklearn.linearmodel.linearregression. https://scikit-learn.org/stable/modules/generated/sklearn.linear$_m$odel.$LinearRegression.html, n.d. Accessed: 20122 - 03 - 30.$

[5] sklearn. sklearn.preprocessing.polynomialfeatures. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html, n.d. Accessed: 20122-03-30.

7 Appendix

The codes are provided below

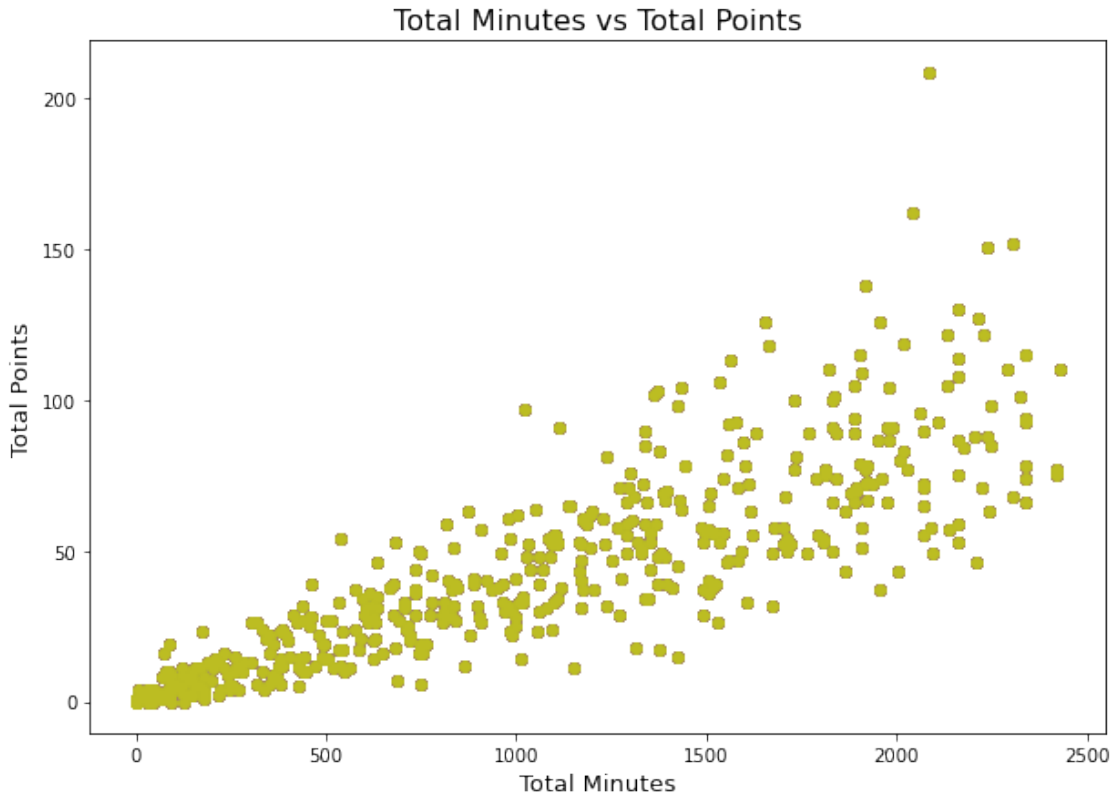
project

April 3, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, HuberRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

[2]:
```
raw_data = pd.read_csv("merged_gw.csv")
data = raw_data.drop(['team','xP', 'bonus','bps', 'bps', 'creativity',␣
 ↪'element', 'fixture','ict_index', 'influence',
                      'kickoff_time', 'opponent_team', 'own_goals',␣
 ↪'penalties_missed', 'penalties_saved', 'red_cards',
                      'round', 'saves', 'selected', 'team_a_score',␣
 ↪'team_h_score', 'threat', 'transfers_balance',
                      'transfers_in', 'transfers_out', 'was_home',␣
 ↪'yellow_cards'],axis=1)
data.columns=['Name','Position','Assists','Clean Sheets','Goals␣
 ↪conceded','Goals Scored','Minutes','Points','Costs','GW']
data = data[['Name','Position','Goals Scored', 'Assists', 'Clean Sheets',␣
 ↪'Goals conceded','Points','Minutes','GW','Costs']]
```

[3]:
```
viz_data = pd.read_csv("cleaned_players.csv")
plt.figure(figsize=(10,7))
plt.xlabel('Total Minutes',size=13)
plt.ylabel('Total Points',size=13)
plt.title("Total Minutes vs Total Points",size=16)
for i in viz_data:
    plt.scatter(viz_data['minutes'],viz_data['total_points'])
plt.show()
```



[4]:
```
#Splitting the original data set into 4 based on position
GK_data = data[data['Position'] == 'GK']
DEF_data = data[data['Position'] == 'DEF']
MID_data = data[data['Position'] == 'MID']
FWD_data = data[data['Position'] == 'FWD']
```

[5]:
```
#Credits to Woody Pride - stackoverflow
#Convert those 4 data sets to dictionaries
name_list = FWD_data['Name'].unique()
FWD_dict = {name : pd.DataFrame for name in name_list}
for key in FWD_dict.keys():
    FWD_dict[key] = data[:][data['Name'] == key]

name_list = DEF_data['Name'].unique()
DEF_dict = {name : pd.DataFrame for name in name_list}
for key in DEF_dict.keys():
    DEF_dict[key] = data[:][data['Name'] == key]

name_list = GK_data['Name'].unique()
GK_dict = {name : pd.DataFrame for name in name_list}
for key in GK_dict.keys():
```

```
    GK_dict[key] = data[:][data['Name'] == key]

name_list = MID_data['Name'].unique()
MID_dict = {name : pd.DataFrame for name in name_list}
for key in MID_dict.keys():
    MID_dict[key] = data[:][data['Name'] == key]

dictionary_list = [FWD_dict,MID_dict,DEF_dict,GK_dict]
tr_error_list = [] #Collecting the training error
val_error_list = [] #Collecting the validation error
test_error_list = [] #Collecting the test error
```

[6]:
```
#Linear regression
FWD_pred_dict = {}
for player in FWD_dict:
    FWD_dict[player].insert(0,'Total Minutes',FWD_dict[player]['Minutes'].
↪cumsum())
    FWD_dict[player].insert(0,'Total Points',FWD_dict[player]['Points'].
↪cumsum())

    # Create the feature and label vectors
    X = FWD_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = FWD_dict[player]['Total Points'].to_numpy()

    #Splitting
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
↪33, random_state=42)

    if FWD_dict[player]['Total Minutes'].iloc[-1] >= 1350: #play less then 1350␣
↪minutes this season
        lin_regr = LinearRegression().fit(X, y)
        y_pred_train = lin_regr.predict(X_train)
        tr_error = mean_squared_error(y_train, y_pred_train)
        tr_error_list.append(tr_error)
        y_pred_val = lin_regr.predict(X_val)
        val_error = mean_squared_error(y_val, y_pred_val)
        val_error_list.append(val_error)
        y_pred_test = lin_regr.predict(X_test)
        test_error = mean_squared_error(y_test, y_pred_test)
        test_error_list.append(test_error)
        predicted_points = lin_regr.intercept_ + (lin_regr.
↪coef_*(FWD_dict[player]['Total Minutes'].iloc[-1]+45)
                                                  - FWD_dict[player]['Total␣
↪Points'].iloc[-1])
```

```
        FWD_pred_dict[player] = round(predicted_points[0],2)
```

[7]:
```
#Linear regression
DEF_pred_dict = {}
for player in DEF_dict:
    DEF_dict[player].insert(0,'Total Minutes',DEF_dict[player]['Minutes'].
 ↪cumsum())
    DEF_dict[player].insert(0,'Total Points',DEF_dict[player]['Points'].
 ↪cumsum())

    # Create the feature and label vectors
    X = DEF_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = DEF_dict[player]['Total Points'].to_numpy()

    #Splitting
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
 ↪33, random_state=42)

    if DEF_dict[player]['Total Minutes'].iloc[-1] >= 1350: #play less then 1350␣
 ↪minutes this season
        lin_regr = LinearRegression().fit(X, y)
        y_pred_train = lin_regr.predict(X_train)
        tr_error = mean_squared_error(y_train, y_pred_train)
        tr_error_list.append(tr_error)
        y_pred_val = lin_regr.predict(X_val)
        val_error = mean_squared_error(y_val, y_pred_val)
        val_error_list.append(val_error)
        y_pred_test = lin_regr.predict(X_test)
        test_error = mean_squared_error(y_test, y_pred_test)
        test_error_list.append(test_error)
        predicted_points = lin_regr.intercept_ + (lin_regr.
 ↪coef_*(DEF_dict[player]['Total Minutes'].iloc[-1]+45)
                                                  - DEF_dict[player]['Total␣
 ↪Points'].iloc[-1])
        DEF_pred_dict[player] = round(predicted_points[0],2)
```

[8]:
```
#Linear regression
MID_pred_dict = {}
for player in MID_dict:
    MID_dict[player].insert(0,'Total Minutes',MID_dict[player]['Minutes'].
 ↪cumsum())
    MID_dict[player].insert(0,'Total Points',MID_dict[player]['Points'].
 ↪cumsum())
```

```
    # Create the feature and label vectors
    X = MID_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = MID_dict[player]['Total Points'].to_numpy()

    #Splitting
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
↪33, random_state=42)

    if MID_dict[player]['Total Minutes'].iloc[-1] >= 1350: #play less then 1350␣
↪minutes this season
        lin_regr = LinearRegression().fit(X, y)
        y_pred_train = lin_regr.predict(X_train)
        tr_error = mean_squared_error(y_train, y_pred_train)
        tr_error_list.append(tr_error)
        y_pred_val = lin_regr.predict(X_val)
        val_error = mean_squared_error(y_val, y_pred_val)
        val_error_list.append(val_error)
        y_pred_test = lin_regr.predict(X_test)
        test_error = mean_squared_error(y_test, y_pred_test)
        test_error_list.append(test_error)
        predicted_points = lin_regr.intercept_ + (lin_regr.
↪coef_*(MID_dict[player]['Total Minutes'].iloc[-1]+45)
                                                  - MID_dict[player]['Total␣
↪Points'].iloc[-1])
        MID_pred_dict[player] = round(predicted_points[0],2)
```

[9]:
```
#Linear regression
GK_pred_dict = {}
for player in GK_dict:
    GK_dict[player].insert(0,'Total Minutes',GK_dict[player]['Minutes'].
↪cumsum())
    GK_dict[player].insert(0,'Total Points',GK_dict[player]['Points'].cumsum())

    # Create the feature and label vectors
    X = GK_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = GK_dict[player]['Total Points'].to_numpy()

    #Splitting
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
↪33, random_state=42)
```

```
    if GK_dict[player]['Total Minutes'].iloc[-1] >= 1350: #play less then 1350␣
↪minutes this season
        lin_regr = LinearRegression().fit(X, y)
        y_pred_train = lin_regr.predict(X_train)
        tr_error = mean_squared_error(y_train, y_pred_train)
        tr_error_list.append(tr_error)
        y_pred_val = lin_regr.predict(X_val)
        val_error = mean_squared_error(y_val, y_pred_val)
        val_error_list.append(val_error)
        y_pred_test = lin_regr.predict(X_test)
        test_error = mean_squared_error(y_test, y_pred_test)
        test_error_list.append(test_error)
        predicted_points = lin_regr.intercept_ + (lin_regr.
↪coef_*(GK_dict[player]['Total Minutes'].iloc[-1]+45)
                                                  - GK_dict[player]['Total␣
↪Points'].iloc[-1])
        GK_pred_dict[player] = round(predicted_points[0],2)
```

[10]:
```
#Credited to Devin Jeanpierre - stackoverflow
#List the top 5 footballers in each position

#FWD
dictsorted = dict(sorted(FWD_pred_dict.items(), key=lambda item: item[1],␣
↪reverse=True))
count = 0
print("Recommended FWD:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
↪FWD_dict[i]['Costs'].iloc[-1] )
        count = count + 1

#MID
dictsorted = dict(sorted(MID_pred_dict.items(), key=lambda item: item[1],␣
↪reverse=True))
count = 0
print("Recommended MID:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
↪MID_dict[i]['Costs'].iloc[-1] )
        count = count + 1
```

```
#DEF
dictsorted = dict(sorted(DEF_pred_dict.items(), key=lambda item: item[1],␣
 ↪reverse=True))
count = 0
print("Recommended DEF:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
 ↪DEF_dict[i]['Costs'].iloc[-1] )
        count = count + 1

#GK
dictsorted = dict(sorted(GK_pred_dict.items(), key=lambda item: item[1],␣
 ↪reverse=True))
count = 0
print("Recommended GK:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
 ↪GK_dict[i]['Costs'].iloc[-1] )
        count = count + 1
```

```
Recommended FWD:
Emmanuel Dennis  - Predicted points:  16.59  - Cost:  61
Armando Broja  - Predicted points:  12.99  - Cost:  55
Neal Maupay  - Predicted points:  10.72  - Cost:  65
Ollie Watkins  - Predicted points:  7.45  - Cost:  75
Joshua King  - Predicted points:  7.08  - Cost:  58
Richarlison de Andrade  - Predicted points:  6.51  - Cost:  75
Recommended MID:
Phil Foden  - Predicted points:  12.17  - Cost:  79
Raphael Dias Belloli  - Predicted points:  11.35  - Cost:  65
Bernardo Mota Veiga de Carvalho e Silva  - Predicted points:  10.56  - Cost:  72
Leandro Trossard  - Predicted points:  10.52  - Cost:  59
Conor Gallagher  - Predicted points:  9.35  - Cost:  61
Youri Tielemans  - Predicted points:  8.59  - Cost:  64
Recommended DEF:
Marc Cucurella  - Predicted points:  11.91  - Cost:  50
Jan Bednarek  - Predicted points:  11.53  - Cost:  44
Kieran Tierney  - Predicted points:  10.83  - Cost:  51
Romain Saïss  - Predicted points:  8.19  - Cost:  51
Grant Hanley  - Predicted points:  8.05  - Cost:  44
```

```
Charlie Taylor  - Predicted points:  8.02  - Cost:  44
Recommended GK:
Robert Sánchez  - Predicted points:  8.0  - Cost:  46
José Malheiro de Sá  - Predicted points:  7.19  - Cost:  53
Aaron Ramsdale  - Predicted points:  5.93  - Cost:  51
Illan Meslier  - Predicted points:  5.24  - Cost:  48
David de Gea  - Predicted points:  5.22  - Cost:  52
Lukasz Fabianski  - Predicted points:  5.09  - Cost:  50
```

[11]:
```python
#Calculating the average errors
print("The average errors of the Linear regression model are:")
print("  *The training error: ", sum(tr_error_list)/len(tr_error_list))
print("  *The validation error: ", sum(val_error_list)/len(val_error_list))
print("  *The test error: ", sum(test_error_list)/len(test_error_list))
```

```
The average errors of the Linear regression model are:
  *The training error:  13.323362201095522
  *The validation error:  14.999252546069942
  *The test error:  15.273670656282041
```

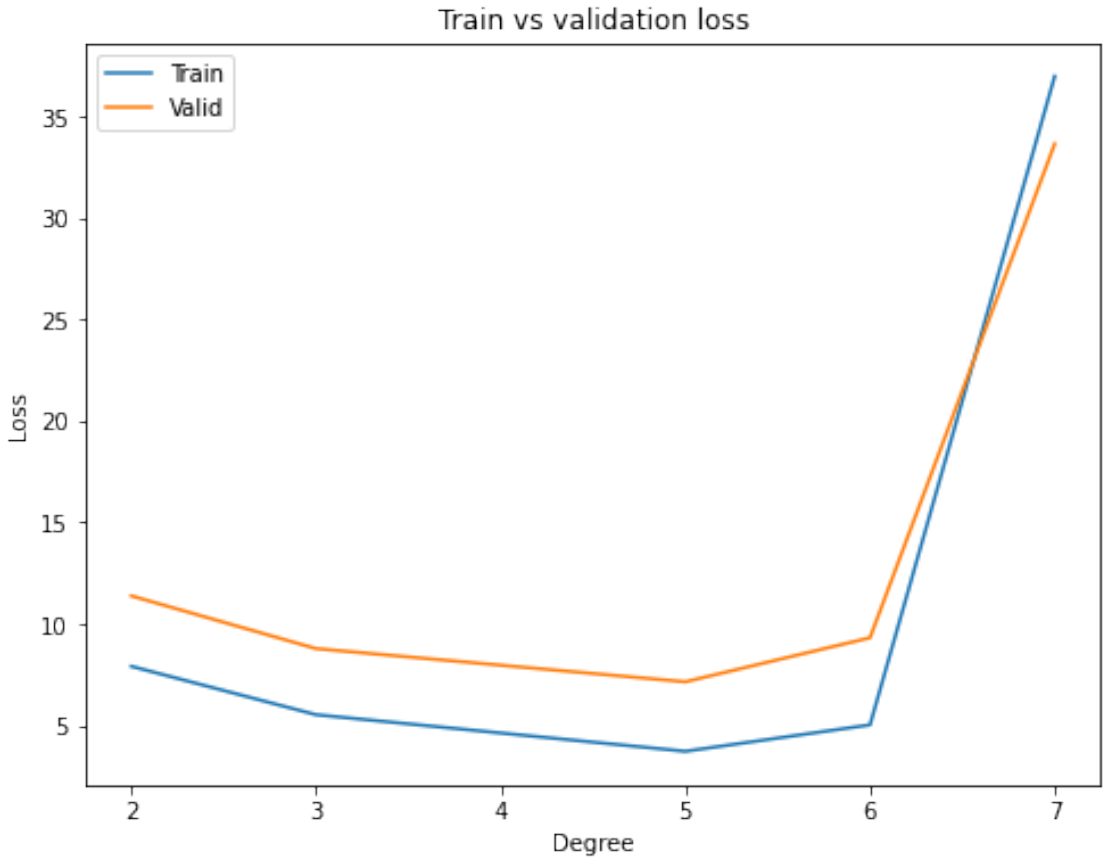
[12]:
```python
tr_error_list = [] #Collecting the training error
val_error_list = [] #Collecting the validation error
test_error_list = [] #Collecting the test error
```

[13]:
```python
#Polynomial regression
degrees = [2,3,5,6,7]
GK_pred_dict = {}
DEF_pred_dict = {}
MID_pred_dict = {}
FWD_pred_dict = {}
for i, degree in enumerate(degrees):
    tr_errors, val_errors, test_errors = [], [], []
    lin_regr = LinearRegression(fit_intercept=False)
    poly = PolynomialFeatures(degree=degree)
    for player in GK_dict:
        X = GK_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
        y = GK_dict[player]['Total Points'].to_numpy()
        X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
        X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem,␣
↪test_size=0.33, random_state=42)
        if GK_dict[player]['Total Minutes'].iloc[-1] >= 1350:
            X_train_poly = poly.fit_transform(X_train)
            lin_regr.fit(X_train_poly, y_train)
            y_pred_train = lin_regr.predict(X_train_poly)
            tr_error = mean_squared_error(y_train, y_pred_train)
            X_val_poly = poly.fit_transform(X_val)
```

```
            y_pred_val = lin_regr.predict(X_val_poly)
            val_error = mean_squared_error(y_val, y_pred_val)
            X_test_poly = poly.fit_transform(X_test)
            y_pred_test = lin_regr.predict(X_test_poly)
            test_error = mean_squared_error(y_test, y_pred_test)
            tr_errors.append(tr_error)
            val_errors.append(val_error)
            test_errors.append(test_error)

    for player in DEF_dict:
        X = DEF_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
        y = DEF_dict[player]['Total Points'].to_numpy()
        X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
        X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem,␣
↪test_size=0.33, random_state=42)
        if DEF_dict[player]['Total Minutes'].iloc[-1] >= 1350:
            X_train_poly = poly.fit_transform(X_train)
            lin_regr.fit(X_train_poly, y_train)
            y_pred_train = lin_regr.predict(X_train_poly)
            tr_error = mean_squared_error(y_train, y_pred_train)
            X_val_poly = poly.fit_transform(X_val)
            y_pred_val = lin_regr.predict(X_val_poly)
            val_error = mean_squared_error(y_val, y_pred_val)
            X_test_poly = poly.fit_transform(X_test)
            y_pred_test = lin_regr.predict(X_test_poly)
            test_error = mean_squared_error(y_test, y_pred_test)
            tr_errors.append(tr_error)
            val_errors.append(val_error)
            test_errors.append(test_error)

    for player in MID_dict:
        X = MID_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
        y = MID_dict[player]['Total Points'].to_numpy()
        X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
        X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem,␣
↪test_size=0.33, random_state=42)
        if MID_dict[player]['Total Minutes'].iloc[-1] >= 1350:
            X_train_poly = poly.fit_transform(X_train)
            lin_regr.fit(X_train_poly, y_train)
            y_pred_train = lin_regr.predict(X_train_poly)
            tr_error = mean_squared_error(y_train, y_pred_train)
            X_val_poly = poly.fit_transform(X_val)
            y_pred_val = lin_regr.predict(X_val_poly)
            val_error = mean_squared_error(y_val, y_pred_val)
            X_test_poly = poly.fit_transform(X_test)
```

```
            y_pred_test = lin_regr.predict(X_test_poly)
            test_error = mean_squared_error(y_test, y_pred_test)
            tr_errors.append(tr_error)
            val_errors.append(val_error)
            test_errors.append(test_error)

    for player in FWD_dict:
        X = FWD_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
        y = FWD_dict[player]['Total Points'].to_numpy()
        X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
        X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem,␣
↪test_size=0.33, random_state=42)
        if FWD_dict[player]['Total Minutes'].iloc[-1] >= 1350:
            X_train_poly = poly.fit_transform(X_train)
            lin_regr.fit(X_train_poly, y_train)
            y_pred_train = lin_regr.predict(X_train_poly)
            tr_error = mean_squared_error(y_train, y_pred_train)
            X_val_poly = poly.fit_transform(X_val)
            y_pred_val = lin_regr.predict(X_val_poly)
            val_error = mean_squared_error(y_val, y_pred_val)
            X_test_poly = poly.fit_transform(X_test)
            y_pred_test = lin_regr.predict(X_test_poly)
            test_error = mean_squared_error(y_test, y_pred_test)
            tr_errors.append(tr_error)
            val_errors.append(val_error)
            test_errors.append(test_error)

    tr_error_list.append(sum(tr_errors)/len(tr_errors))
    val_error_list.append(sum(val_errors)/len(val_errors))
    test_error_list.append(sum(test_errors)/len(test_errors))
```

[14]:
```
plt.figure(figsize=(8, 6))

plt.plot(degrees, tr_error_list, label = 'Train')
plt.plot(degrees, val_error_list,label = 'Valid')
plt.legend(loc = 'upper left')

plt.xlabel('Degree')
plt.ylabel('Loss')
plt.title('Train vs validation loss')
plt.show()
```



[15]:
```
#Degree = 5
tr_errors = []
val_errors = []
test_errors = []
GK_pred_dict = {}
DEF_pred_dict = {}
MID_pred_dict = {}
FWD_pred_dict = {}

for player in GK_dict:
    lin_regrGK = LinearRegression(fit_intercept=False)
    polyGK = PolynomialFeatures(degree=5)
    X = GK_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = GK_dict[player]['Total Points'].to_numpy()
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
 ↪33, random_state=42)
    if GK_dict[player]['Total Minutes'].iloc[-1] >= 1350:
```

```
        X_train_poly = polyGK.fit_transform(X_train)
        lin_regrGK.fit(X_train_poly, y_train)
        y_pred_train = lin_regrGK.predict(X_train_poly)
        tr_error = mean_squared_error(y_train, y_pred_train)
        X_val_poly = polyGK.fit_transform(X_val)
        y_pred_val = lin_regrGK.predict(X_val_poly)
        val_error = mean_squared_error(y_val, y_pred_val)
        X_test_poly = polyGK.fit_transform(X_test)
        y_pred_test = lin_regrGK.predict(X_test_poly)
        test_error = mean_squared_error(y_test, y_pred_test)
        tr_errors.append(tr_error)
        val_errors.append(val_error)
        test_errors.append(test_error)
        coE = lin_regrGK.coef_
        total = GK_dict[player]['Total Minutes'].iloc[-1]+45
        predicted_points =␣
↪coE[0]*pow(total,0)+coE[1]*pow(total,1)+coE[2]*pow(total,2)+coE[3]*pow(total,3)+coE[4]*pow(
        GK_pred_dict[player] = round(predicted_points - GK_dict[player]['Total␣
↪Points'].iloc[-1],2)

for player in DEF_dict:
    lin_regrDEF = LinearRegression(fit_intercept=False)
    polyDEF = PolynomialFeatures(degree=5)
    X = DEF_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = DEF_dict[player]['Total Points'].to_numpy()
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
↪33, random_state=42)
    if DEF_dict[player]['Total Minutes'].iloc[-1] >= 1350:
        X_train_poly = polyDEF.fit_transform(X_train)
        lin_regrDEF.fit(X_train_poly, y_train)
        y_pred_train = lin_regrDEF.predict(X_train_poly)
        tr_error = mean_squared_error(y_train, y_pred_train)
        X_val_poly = polyDEF.fit_transform(X_val)
        y_pred_val = lin_regrDEF.predict(X_val_poly)
        val_error = mean_squared_error(y_val, y_pred_val)
        X_test_poly = polyDEF.fit_transform(X_test)
        y_pred_test = lin_regrDEF.predict(X_test_poly)
        test_error = mean_squared_error(y_test, y_pred_test)
        tr_errors.append(tr_error)
        val_errors.append(val_error)
        test_errors.append(test_error)
        coE = lin_regrDEF.coef_
        total = DEF_dict[player]['Total Minutes'].iloc[-1]+45
        predicted_points =␣
↪coE[0]*pow(total,0)+coE[1]*pow(total,1)+coE[2]*pow(total,2)+coE[3]*pow(total,3)+coE[4]*pow(
```

```
        DEF_pred_dict[player] = round(predicted_points -␣
↪DEF_dict[player]['Total Points'].iloc[-1],2)

for player in MID_dict:
    lin_regrMID = LinearRegression(fit_intercept=False)
    polyMID = PolynomialFeatures(degree=5)
    X = MID_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = MID_dict[player]['Total Points'].to_numpy()
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
↪33, random_state=42)
    if MID_dict[player]['Total Minutes'].iloc[-1] >= 1350:
        X_train_poly = polyMID.fit_transform(X_train)
        lin_regrMID.fit(X_train_poly, y_train)
        y_pred_train = lin_regrMID.predict(X_train_poly)
        tr_error = mean_squared_error(y_train, y_pred_train)
        X_val_poly = polyMID.fit_transform(X_val)
        y_pred_val = lin_regrMID.predict(X_val_poly)
        val_error = mean_squared_error(y_val, y_pred_val)
        X_test_poly = polyMID.fit_transform(X_test)
        y_pred_test = lin_regrMID.predict(X_test_poly)
        test_error = mean_squared_error(y_test, y_pred_test)
        tr_errors.append(tr_error)
        val_errors.append(val_error)
        test_errors.append(test_error)
        coE = lin_regrMID.coef_
        total = MID_dict[player]['Total Minutes'].iloc[-1]+45
        predicted_points =␣
↪coE[0]*pow(total,0)+coE[1]*pow(total,1)+coE[2]*pow(total,2)+coE[3]*pow(total,3)+coE[4]*pow(
        MID_pred_dict[player] = round(predicted_points -␣
↪MID_dict[player]['Total Points'].iloc[-1],2)

for player in FWD_dict:
    lin_regrFWD = LinearRegression(fit_intercept=False)
    polyFWD = PolynomialFeatures(degree=5)
    X = FWD_dict[player]['Total Minutes'].to_numpy().reshape(-1,1)
    y = FWD_dict[player]['Total Points'].to_numpy()
    X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.
↪33, random_state=42)
    if FWD_dict[player]['Total Minutes'].iloc[-1] >= 1350:
        X_train_poly = polyFWD.fit_transform(X_train)
        lin_regrFWD.fit(X_train_poly, y_train)
        y_pred_train = lin_regrFWD.predict(X_train_poly)
```

```
        tr_error = mean_squared_error(y_train, y_pred_train)
        X_val_poly = polyFWD.fit_transform(X_val)
        y_pred_val = lin_regrFWD.predict(X_val_poly)
        val_error = mean_squared_error(y_val, y_pred_val)
        X_test_poly = polyFWD.fit_transform(X_test)
        y_pred_test = lin_regrFWD.predict(X_test_poly)
        test_error = mean_squared_error(y_test, y_pred_test)
        tr_errors.append(tr_error)
        val_errors.append(val_error)
        test_errors.append(test_error)
        coE = lin_regrFWD.coef_
        total = FWD_dict[player]['Total Minutes'].iloc[-1]+45
        predicted_points =␣
↪coE[0]*pow(total,0)+coE[1]*pow(total,1)+coE[2]*pow(total,2)+coE[3]*pow(total,3)+coE[4]*pow(
        FWD_pred_dict[player] = round(predicted_points -␣
↪FWD_dict[player]['Total Points'].iloc[-1],2)
```

[16]:
```
#Credited to Devin Jeanpierre - stackoverflow
#List the top 5 footballers in each position

#FWD
dictsorted = dict(sorted(FWD_pred_dict.items(), key=lambda item: item[1],␣
↪reverse=True))
count = 0
print("Recommended FWD:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
↪FWD_dict[i]['Costs'].iloc[-1] )
        count = count + 1

#MID
dictsorted = dict(sorted(MID_pred_dict.items(), key=lambda item: item[1],␣
↪reverse=True))
count = 0
print("Recommended MID:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
↪MID_dict[i]['Costs'].iloc[-1] )
        count = count + 1
```

```
#DEF
dictsorted = dict(sorted(DEF_pred_dict.items(), key=lambda item: item[1],␣
 ↪reverse=True))
count = 0
print("Recommended DEF:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
 ↪DEF_dict[i]['Costs'].iloc[-1] )
        count = count + 1

#GK
dictsorted = dict(sorted(GK_pred_dict.items(), key=lambda item: item[1],␣
 ↪reverse=True))
count = 0
print("Recommended GK:")
for i in dictsorted:
    if count > 5:
        break
    else:
        print(i," - Predicted points: ", dictsorted[i], " - Cost: ",␣
 ↪GK_dict[i]['Costs'].iloc[-1] )
        count = count + 1
```

```
Recommended FWD:
Jamie Vardy  - Predicted points:  7.84  - Cost:  103
Cristiano Ronaldo dos Santos Aveiro  - Predicted points:  7.16  - Cost:  123
Ivan Toney  - Predicted points:  7.02  - Cost:  66
Danny Ings  - Predicted points:  6.97  - Cost:  77
Emmanuel Dennis  - Predicted points:  6.25  - Cost:  61
Harry Kane  - Predicted points:  6.18  - Cost:  124
Recommended MID:
Diogo Jota  - Predicted points:  14.76  - Cost:  83
Mohamed Salah  - Predicted points:  12.74  - Cost:  133
Mason Mount  - Predicted points:  9.0  - Cost:  75
Raheem Sterling  - Predicted points:  7.04  - Cost:  106
James Maddison  - Predicted points:  6.44  - Cost:  67
Nathan Redmond  - Predicted points:  6.44  - Cost:  59
Recommended DEF:
Tino Livramento  - Predicted points:  8.13  - Cost:  44
Trent Alexander-Arnold  - Predicted points:  8.02  - Cost:  85
Joel Matip  - Predicted points:  7.6  - Cost:  49
Antonio Rüdiger  - Predicted points:  6.08  - Cost:  61
Luke Ayling  - Predicted points:  5.85  - Cost:  43
```

```
Virgil van Dijk  - Predicted points:  5.8  - Cost:  68
Recommended GK:
Hugo Lloris  - Predicted points:  7.26  - Cost:  54
Vicente Guaita  - Predicted points:  5.68  - Cost:  46
Alisson Ramses Becker  - Predicted points:  5.48  - Cost:  60
Kasper Schmeichel  - Predicted points:  5.34  - Cost:  48
Edouard Mendy  - Predicted points:  4.2  - Cost:  61
Aaron Ramsdale  - Predicted points:  3.82  - Cost:  51
```

[17]:
```python
#Calculating the average errors
print("The average errors of the Linear regression model are:")
print("  *The training error: ", sum(tr_errors)/len(tr_errors))
print("  *The validation error: ", sum(val_errors)/len(val_errors))
print("  *The test error: ", sum(test_errors)/len(test_errors))
```

```
The average errors of the Linear regression model are:
  *The training error:  3.7343609502079897
  *The validation error:  7.15471104467227
  *The test error:  6.527039648431503
```