```
import warnings
warnings.filterwarnings('ignore')
import csv
from plotnine import *
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import mean_squared_error, r2_score,
accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import plot_confusion_matrix
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as sch
from matplotlib import pyplot as plt
import numpy as np
import seaborn as sb

pathDF = "/content/FinalProjectDatasetCSV.csv"

prem = pd.read_csv(pathDF)
dummies = pd.get_dummies(prem["position"])
prem = pd.concat([prem,dummies], axis = 1)
prem = prem.drop(columns="position")

prem.dropna(inplace = True)
prem.head()

         full_name  age   birthday birthday_GMT          league
season  \
0  Aaron Cresswell   32  629683200   12/15/1989  Premier League
2018/2019
1     Aaron Lennon   34  545529600    4/16/1987  Premier League
2018/2019
2       Aaron Mooy   31  653356800    9/15/1990  Premier League
2018/2019
3     Aaron Ramsey   31  662169600   12/26/1990  Premier League
2018/2019
4       Aaron Rowe   21  968284800     9/7/2000  Premier League
2018/2019

        Current Club  minutes_played_overall  minutes_played_home  \
0    West Ham United                    1589                  888
1            Burnley                    1217                  487
2  Huddersfield Town                    2327                 1190
3            Arsenal                    1327                  689
```

```
4  Huddersfield Town                      69                   14

   minutes_played_away  ... min_per_assist_overall
cards_per_90_overall  \
0                  701  ...                   1589
0.06
1                  730  ...                   1217
0.07
2                 1137  ...                   2327
0.15
3                  638  ...                    221
0.00
4                   55  ...                      0
0.00

   rank_in_league_top_attackers  rank_in_league_top_midfielders  \
0                           290                             191
1                           196                             187
2                           144                             233
3                            69                               8
4                            -1                              -1

   rank_in_league_top_defenders  rank_in_club_top_scorer  Defender
Forward  \
0                            80                       20         1
0
1                            -1                       10         0
0
2                            -1                        3         0
0
3                            -1                        5         0
0
4                            -1                       31         0
1

   Goalkeeper  Midfielder
0           0           0
1           0           1
2           0           1
3           0           1
4           0           0

[5 rows x 50 columns]
```

```
GoaliePredictors = ['goals_overall', 'minutes_played_overall',
'assists_overall', 'clean_sheets_overall', 'conceded_overall',
'appearances_overall', 'red_cards_overall',
'yellow_cards_overall','age','Defender',
       'Forward', 'Midfielder']
MidPredictors = ['goals_overall', 'minutes_played_overall',
```

```
'assists_overall', 'clean_sheets_overall', 'conceded_overall',
'appearances_overall', 'red_cards_overall',
'yellow_cards_overall','age','Defender',
       'Forward', 'Goalkeeper']
DefPredictors = ['goals_overall', 'minutes_played_overall',
'assists_overall', 'clean_sheets_overall', 'conceded_overall',
'appearances_overall', 'red_cards_overall',
'yellow_cards_overall','age','Midfielder',
       'Forward', 'Goalkeeper']
AttPredictors = ['goals_overall', 'minutes_played_overall',
'assists_overall', 'clean_sheets_overall', 'conceded_overall',
'appearances_overall', 'red_cards_overall',
'yellow_cards_overall','age','Defender',
       'Midfielder', 'Goalkeeper']
contin = ['goals_overall', 'minutes_played_overall',
'assists_overall', 'clean_sheets_overall', 'conceded_overall',
'appearances_overall', 'red_cards_overall', 'yellow_cards_overall']

Goalie_train, Goalie_test, Goalie_Y, Goalie_pred =
train_test_split(prem[GoaliePredictors], prem["Goalkeeper"], test_size
= 0.2)

GoalieZ = StandardScaler()
GoalieZ.fit(Goalie_train[contin])
Goalie_trainZ = GoalieZ.transform(Goalie_train[contin])
Goalie_testZ = GoalieZ.transform(Goalie_test[contin])

Goalie = LogisticRegression()
GoalieModel = Goalie.fit(Goalie_trainZ, Goalie_Y)

Def_train, Def_test, Def_Y, Def_pred =
train_test_split(prem[DefPredictors], prem["Defender"], test_size =
0.2)

DefZ = StandardScaler()
DefZ.fit(Def_train[contin])
Def_trainZ = DefZ.transform(Def_train[contin])
Def_testZ = DefZ.transform(Def_test[contin])

Def = LogisticRegression()
DefModel = Def.fit(Def_trainZ, Def_Y)

Mid_train, Mid_test, Mid_Y, Mid_pred =
train_test_split(prem[MidPredictors], prem["Midfielder"], test_size =
0.2)

MidZ = StandardScaler()
MidZ.fit(Mid_train[contin])
Mid_trainZ = MidZ.transform(Mid_train[contin])
Mid_testZ = MidZ.transform(Mid_test[contin])
```

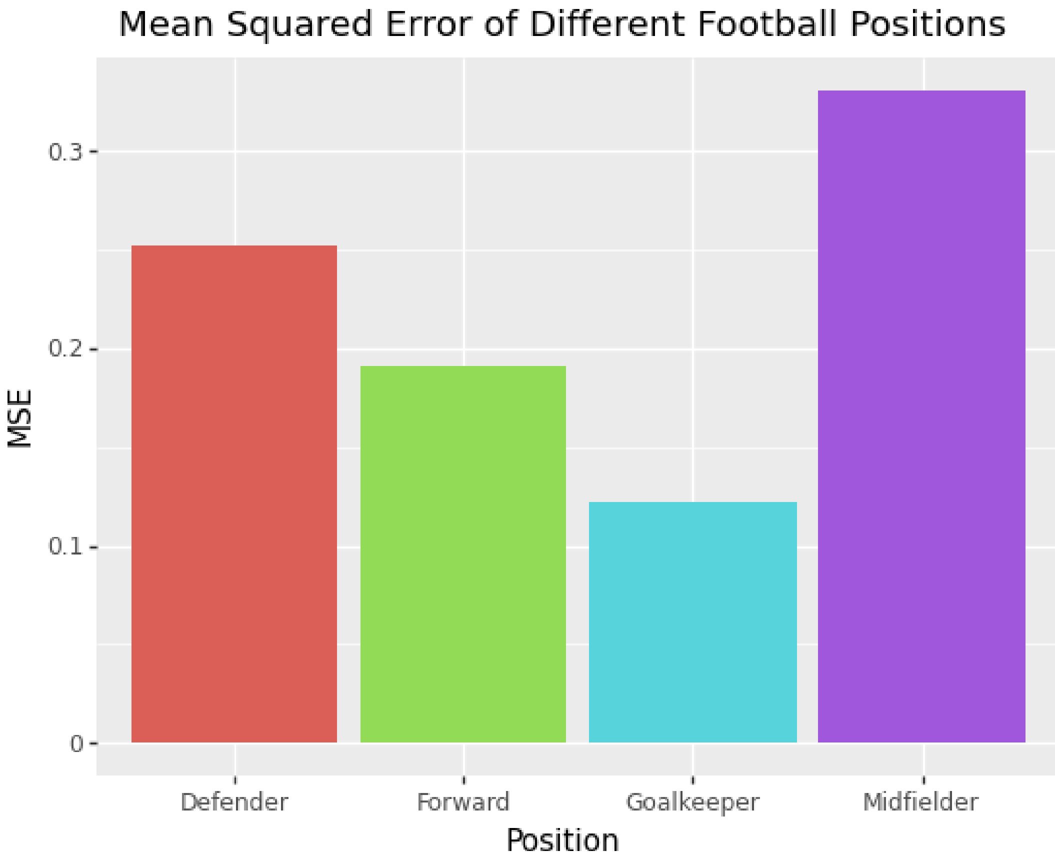
```
Mid = LogisticRegression()
MidModel = Mid.fit(Mid_trainZ, Mid_Y)

Att_train, Att_test, Att_Y, Att_pred =
train_test_split(prem[AttPredictors], prem["Forward"], test_size =
0.2)

AttZ = StandardScaler()
AttZ.fit(Att_train[contin])
Att_trainZ = AttZ.transform(Att_train[contin])
Att_testZ = AttZ.transform(Att_test[contin])

Att = LogisticRegression()
AttModel = Att.fit(Att_trainZ, Att_Y)

GoaliePred = GoalieModel.predict(Goalie_testZ)
GoalieScore = accuracy_score(Goalie_pred, GoaliePred)
GoalieMSE = mean_squared_error(Goalie_pred, GoaliePred)

DefPred = DefModel.predict(Def_testZ)
DefScore = accuracy_score(Def_pred, DefPred)
DefMSE = mean_squared_error(Def_pred, DefPred)

MidPred = MidModel.predict(Mid_testZ)
MidScore = accuracy_score(Mid_pred, MidPred)
MidMSE = mean_squared_error(Mid_pred, MidPred)

AttPred = AttModel.predict(Att_testZ)
AttScore = accuracy_score(Att_pred, AttPred)
AttMSE = mean_squared_error(Att_pred, AttPred)
```

Q1

To create each of the different Logistic Regression models, I made four different train test splits, each using the same predictor variables. However, I swapped out the dummy variable being used as predictor in their respective models. Before doing this I dropped any null values and z-scored all continuous variables (the general football statistic variables). Z-scoring allows the model to be able to compare variables better since they become the same level of unit (distance to mean). These models were now ready to predict if a new set of these predictor variables was a player in one of the positions or not. I calculated all the accuracy scores and mean squared error for each model and placed them into a dataframe so I could present them in ggplots.

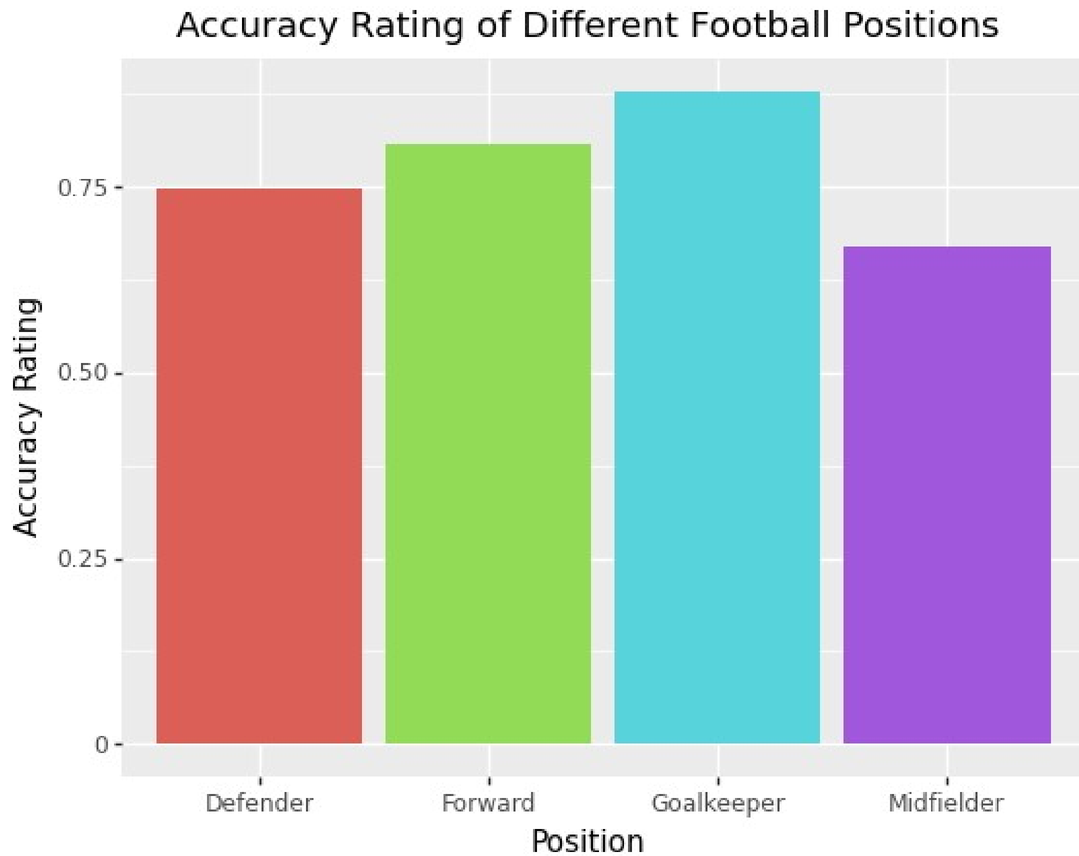
```
PositionMSE = pd.DataFrame({"Position": ["Goalkeeper", "Defender",
"Midfielder", "Forward"],
      "MSE" : [GoalieMSE, DefMSE, MidMSE, AttMSE]})


(ggplot(PositionMSE, aes(x = "Position", y = "MSE", fill =
"Position" ))
```

```
+ ggtitle("Mean Squared Error of Different Football Positions")
+ geom_bar(stat = "identity", show_legend=False))
```




```
<ggplot: (8731155789293)>
```

```
PositionScores = pd.DataFrame({"Position": ["Goalkeeper", "Defender",
"Midfielder", "Forward"],
      "Accuracy Rating" : [GoalieScore, DefScore, MidScore,
AttScore]})


(ggplot(PositionScores, aes(x = "Position", y = "Accuracy Rating",
fill = "Position" ))
+ ggtitle("Accuracy Rating of Different Football Positions")
+ geom_bar(stat = "identity", show_legend=False))
```




```
<ggplot: (8731155795833)>
```

Q1

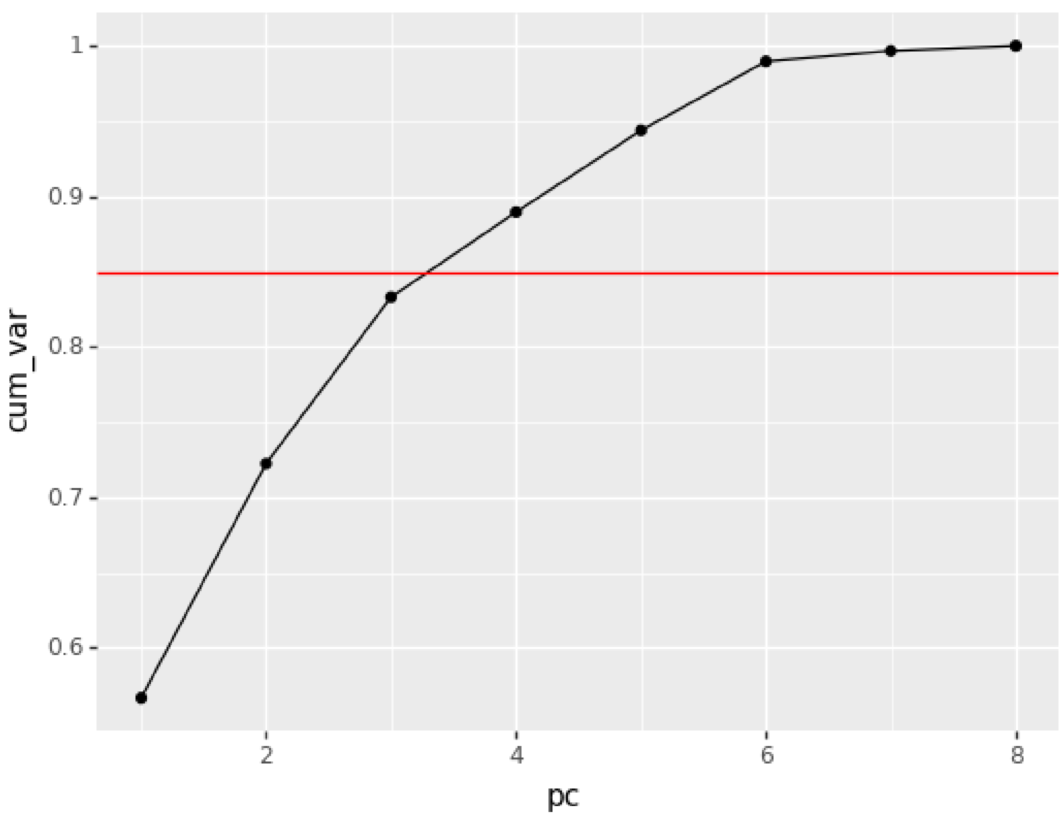
I was confused at first about the results, as there was a pretty large range in accuracy scores between each of the positions. Thinking about it more, it began to make sense. The set of variables I was able to use is not extensive enough to the point where defenders and goalkeepers are taken into account. Thus, since goalkeepers would obviously have far less in terms of goals, assists, etc. it would be easy to predict that a player would be in that position. Following that, forwards have the most connection to the stats used, then midfielders than defenders. Because of this, forwards had the second-highest accuracy score followed by the other two in that order. Perhaps with more statistics (such as successful tackles, pass percentages, etc.) the defender and midfielder models would have higher scores. These assumptions are highlighted by the MSE ggplot. The defender and midfielder models had significantly higher mean squared errors (the models were being wrong) as clearly their models were having trouble predicting them. This model could be a good tool as managers could look at other team's players stats and if they're closer to another position it might reveal clues on how that player is being used, tactics wise.

```
Goalie = PCA()
Def = PCA()
Mid = PCA()
Att = PCA()
```

```
Goalie.fit(Goalie_trainZ)
Def.fit(Def_trainZ)
Mid.fit(Mid_trainZ)
Att.fit(Att_trainZ)
```

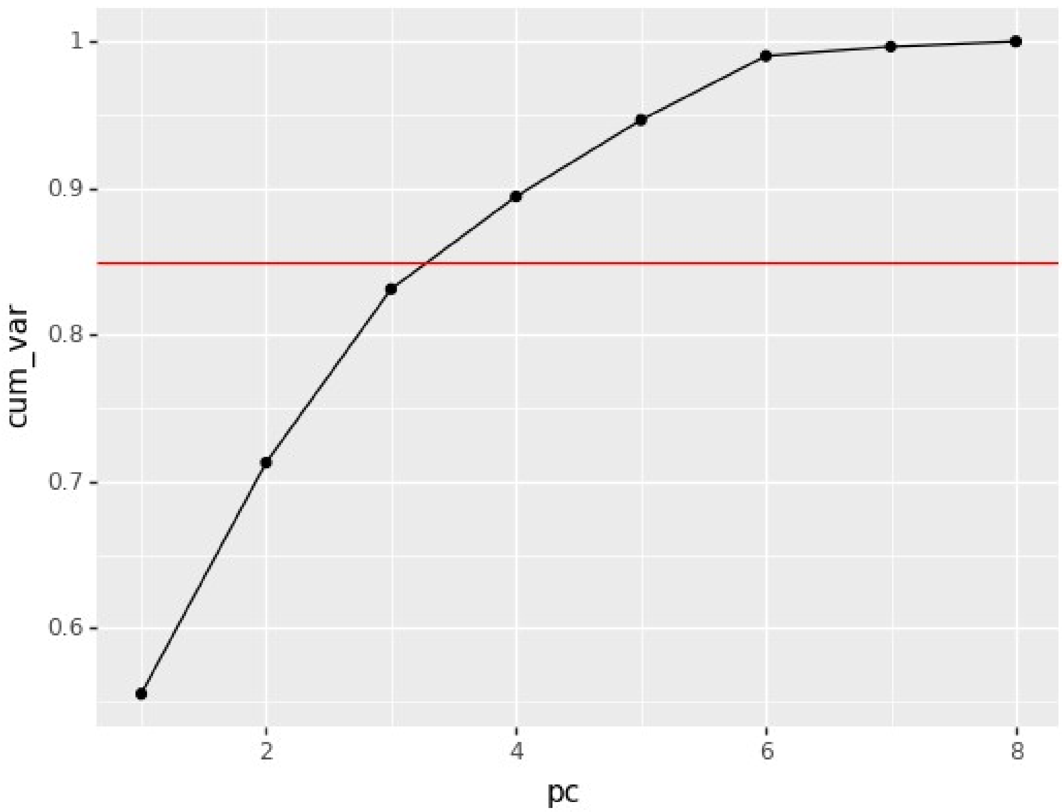
```
PCA()
```

```
GoaliePCA = pd.DataFrame({"expl_var" :
                      Goalie.explained_variance_ratio_,
                      "pc": range(1,9),
                      "cum_var":
                      Goalie.explained_variance_ratio_.cumsum()})
(ggplot(GoaliePCA, aes(x = "pc", y = "cum_var")) + geom_line() +
geom_point()+ geom_hline(yintercept=0.85, color = "red"))
```



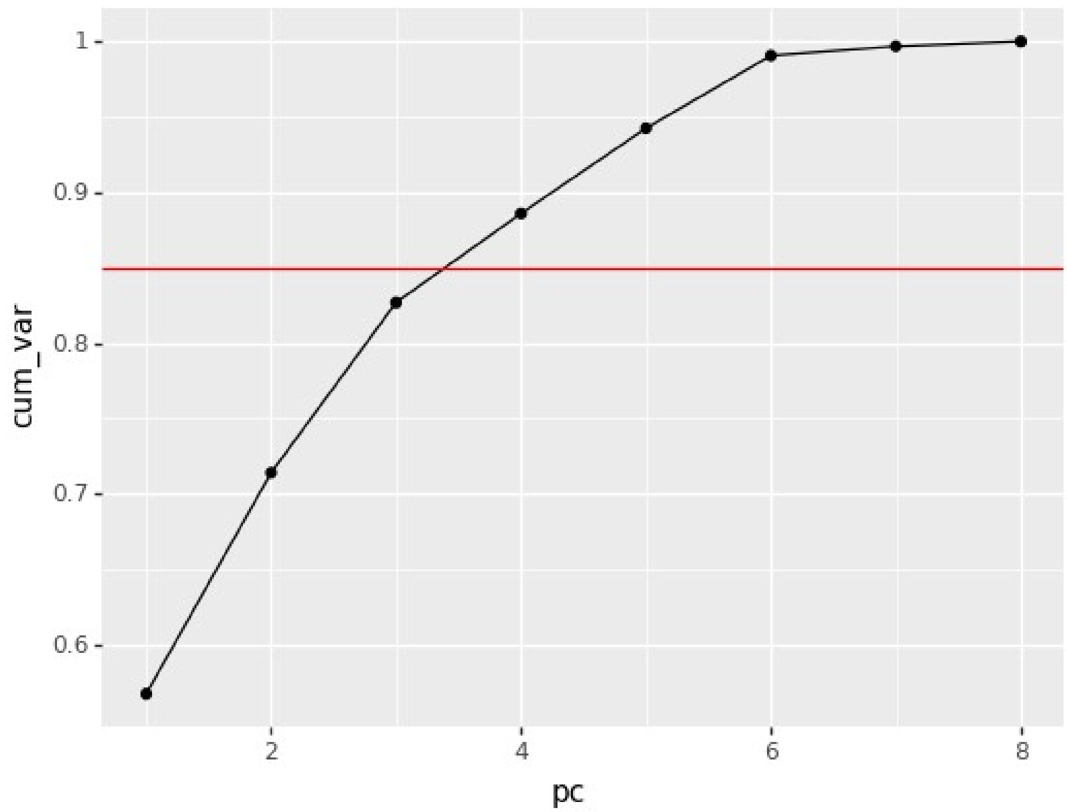

```
<ggplot: (8731155917305)>
```

```
DefPCA = pd.DataFrame({"expl_var" :
                      Def.explained_variance_ratio_,
                      "pc": range(1,9),
                      "cum_var":
                      Def.explained_variance_ratio_.cumsum()})
(ggplot(DefPCA, aes(x = "pc", y = "cum_var")) + geom_line() +
geom_point()+ geom_hline(yintercept=0.85, color = "red"))
```

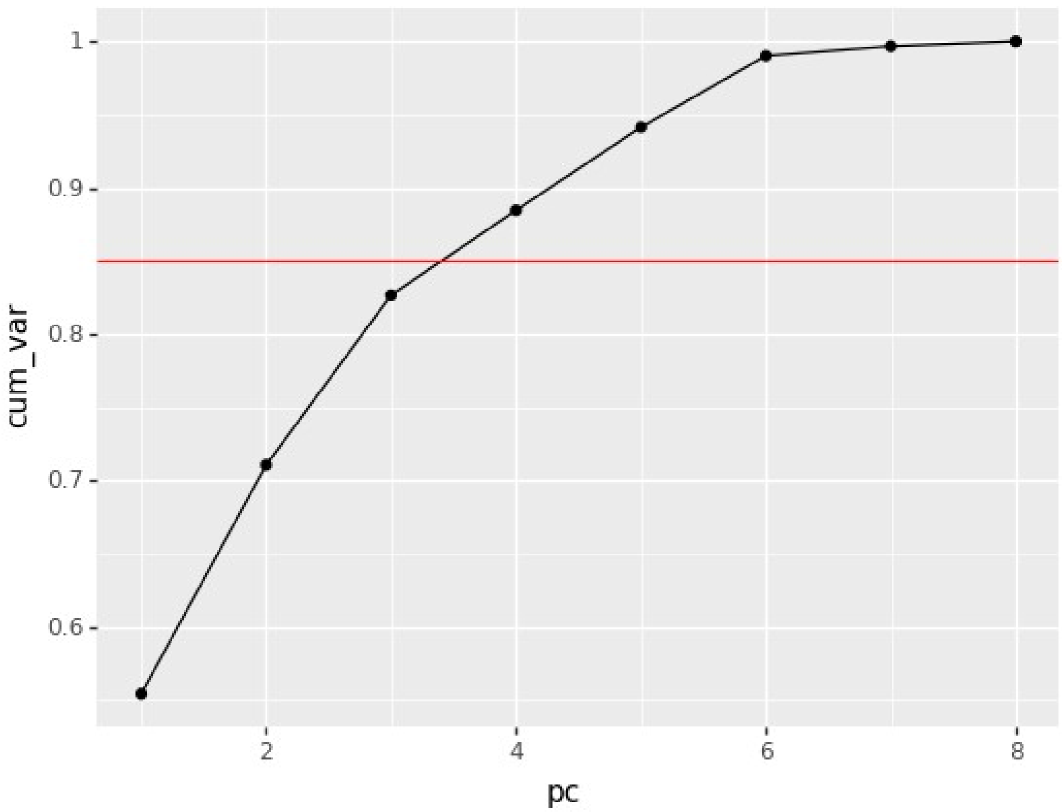



```
<ggplot: (8731155937389)>
```

```
MidPCA = pd.DataFrame({"expl_var" :
                      Mid.explained_variance_ratio_,
                      "pc": range(1,9),
                      "cum_var":
                      Mid.explained_variance_ratio_.cumsum()})
(ggplot(MidPCA, aes(x = "pc", y = "cum_var")) + geom_line() +
geom_point()+ geom_hline(yintercept=0.85, color = "red"))
```




```
<ggplot: (8731155971497)>

AttPCA = pd.DataFrame({"expl_var" :
                       Att.explained_variance_ratio_,
                       "pc": range(1,9),
                       "cum_var":
                       Att.explained_variance_ratio_.cumsum()})
(ggplot(AttPCA, aes(x = "pc", y = "cum_var")) + geom_line() +
geom_point()+ geom_hline(yintercept=0.85, color = "red"))
```



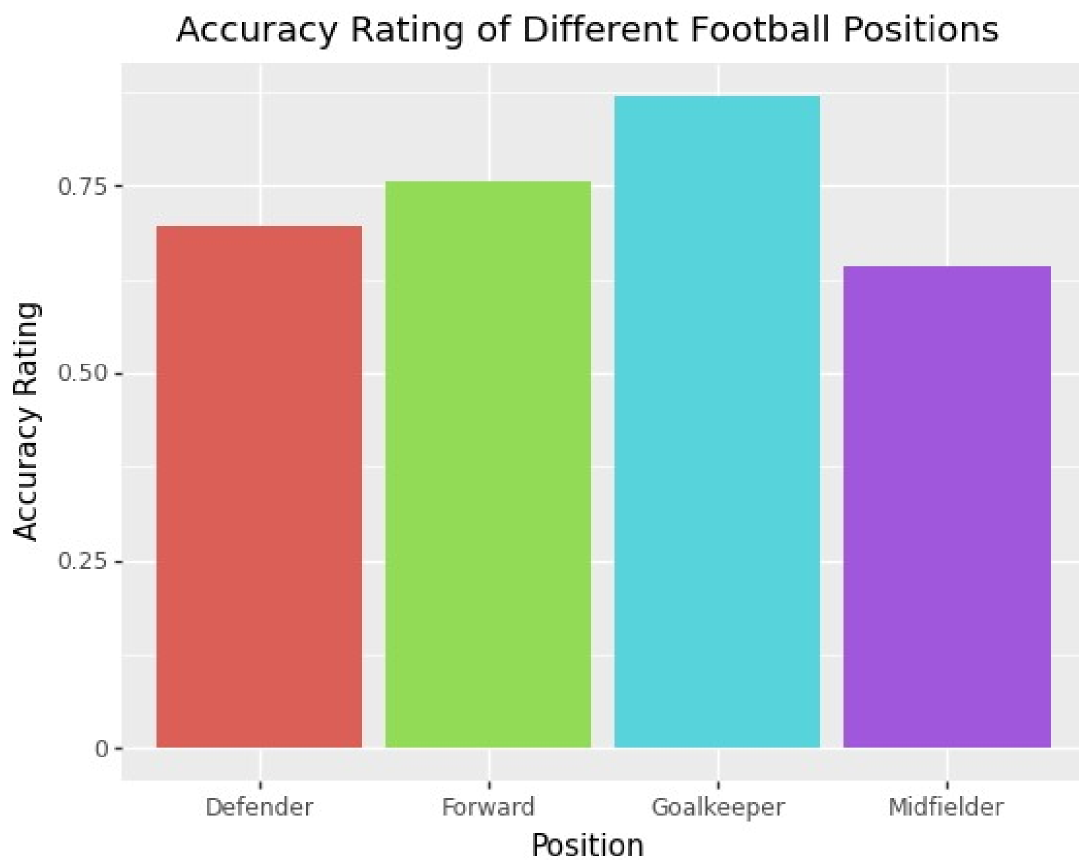

```
<ggplot: (8731156018717)>
```

Q2

To choose the amount of raw variables adequate enough to retain 85% of the original model's information, I printed scree plots for each of the models (after making four different PCA transformation code blocks for them). PCA takes all the variables in a model and puts them on the same axes so as to be able to compare how much of the previous model's information each variable adds. Through this, we can determine how many variables would be required to attain a certain amount of percentage to cut down how many we could use instead of the full amount. Since all the variables were the same in each model, each ended up requiring four raw variables to attain the information. Following this setup, I created four new models each using PCA to transform the variables to use the first four (most relevant) components.

```
GoaliePCATrain = Goalie.transform(Goalie_trainZ)
GoaliePCATest = Goalie.transform(Goalie_testZ)

train_pca = pd.DataFrame(GoaliePCATrain[:,0:4])
test_pca = pd.DataFrame(GoaliePCATest[:,0:4])

GoalieModel = LogisticRegression()
GoalieModel.fit(train_pca, Goalie_Y)
```

```
GoaliePred = GoalieModel.predict(test_pca)
GoalieScore2 = accuracy_score(Goalie_pred, GoaliePred)

DefPCATrain = Def.transform(Def_trainZ)
DefPCATest = Def.transform(Def_testZ)

train_pca = pd.DataFrame(DefPCATrain[:,0:4])
test_pca = pd.DataFrame(DefPCATest[:,0:4])

DefModel = LogisticRegression()
DefModel.fit(train_pca, Def_Y)

DefPred = DefModel.predict(test_pca)
DefScore2 = accuracy_score(Def_pred, DefPred)

MidPCATrain = Mid.transform(Mid_trainZ)
MidPCATest = Mid.transform(Mid_testZ)

train_pca = pd.DataFrame(MidPCATrain[:,0:4])
test_pca = pd.DataFrame(MidPCATest[:,0:4])

MidModel = LogisticRegression()
MidModel.fit(train_pca, Mid_Y)

MidPred = MidModel.predict(test_pca)
MidScore2 = accuracy_score(Mid_pred, MidPred)

AttPCATrain = Att.transform(Att_trainZ)
AttPCATest = Att.transform(Att_testZ)

train_pca = pd.DataFrame(AttPCATrain[:,0:4])
test_pca = pd.DataFrame(AttPCATest[:,0:4])

AttModel = LogisticRegression()
AttModel.fit(train_pca, Att_Y)

AttPred = AttModel.predict(test_pca)
AttScore2 = accuracy_score(Att_pred, AttPred)

PositionScores2 = pd.DataFrame({"Position": ["Goalkeeper", "Defender",
"Midfielder", "Forward"],
      "Accuracy Rating" : [GoalieScore2, DefScore2, MidScore2,
AttScore2]})


(ggplot(PositionScores2, aes(x = "Position", y = "Accuracy Rating",
fill = "Position" ))
+ ggtitle("Accuracy Rating of Different Football Positions")
+ geom_bar(stat = "identity", show_legend=False))
```




```
<ggplot: (8731158508637)>
```

Q2

The new model created actually resulted in slightly lower scores for each variable used. We've discussed before how PCA is better used on models with a very large amount of statistics. These models did not each reach 10 variables, thus it makes sense the PCA effect would not be as evident. Additionally, football statistics are hard enough to use for these predictions (as I discovered previously) so I would definitely go with the previous model containing more variables. PCA is heavily concerned with simplifying computation for computers when running models on data, so the original number of values was already low enough to result in quick computation. As I mentioned in Question 1, if the model's were more accurate with a much higher amount of variables, then this PCA process could be extremely helpful with easing computer efficiency.

```
print(prem['Current Club'].unique())
```

```
['West Ham United' 'Burnley' 'Huddersfield Town' 'Arsenal'
 'Crystal Palace' 'Watford' 'Fulham' 'Liverpool' 'AFC Bournemouth'
 'Wolverhampton Wanderers' 'Everton' 'Leicester City' 'Southampton'
 'Cardiff City' 'Manchester United' 'Tottenham Hotspur'
 'Brighton & Hove Albion' 'Chelsea' 'Newcastle United' 'Manchester
City']
```

```
PositionScores2 = pd.DataFrame({"Position": ["Goalkeeper", "Defender",
"Midfielder", "Forward"],
      "Accuracy Rating" : [GoalieScore2, DefScore2, MidScore2,
AttScore2]})

PremTeam = pd.DataFrame({"Club": ['West Ham United', 'Burnley',
'Huddersfield Town', 'Arsenal',
 'Crystal Palace', 'Watford', 'Fulham', 'Liverpool', 'AFC
Bournemouth',
 'Wolverhampton Wanderers', 'Everton', 'Leicester City',
'Southampton',
 'Cardiff City', 'Manchester United', 'Tottenham Hotspur',
 'Brighton & Hove Albion', 'Chelsea', 'Newcastle United', 'Manchester
City'],
 "Goals" : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], "Average Minutes Played" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 "Assists" : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], "Clean Sheets" : [7, 4, 5, 7, 5, 7, 5, 21, 5, 8, 14, 10, 7, 10, 7,
12, 6, 16, 11, 20],
 "Average Appearances" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 "Red Cards" : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0], "Yellow Cards" : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0],
 "Average Age" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 "Average Defender Rank" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 "Average Midfielder Rank" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 "Average Forwards Rank" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 "Average Ranking" : [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
 })

Club = ['West Ham United', 'Burnley', 'Huddersfield Town', 'Arsenal',
 'Crystal Palace', 'Watford', 'Fulham', 'Liverpool', 'AFC
Bournemouth',
 'Wolverhampton Wanderers', 'Everton', 'Leicester City',
'Southampton',
 'Cardiff City', 'Manchester United', 'Tottenham Hotspur',
 'Brighton & Hove Albion', 'Chelsea', 'Newcastle United', 'Manchester
City']
for club in Club:
  PremTeam.loc[PremTeam["Club"] == club, "Average Minutes Played"] =
round(prem.loc[prem["Current Club"] == club]
["minutes_played_overall"].mean())
  PremTeam.loc[PremTeam["Club"] == club, "Average Appearances"] =
round(prem.loc[prem["Current Club"] == club]
```

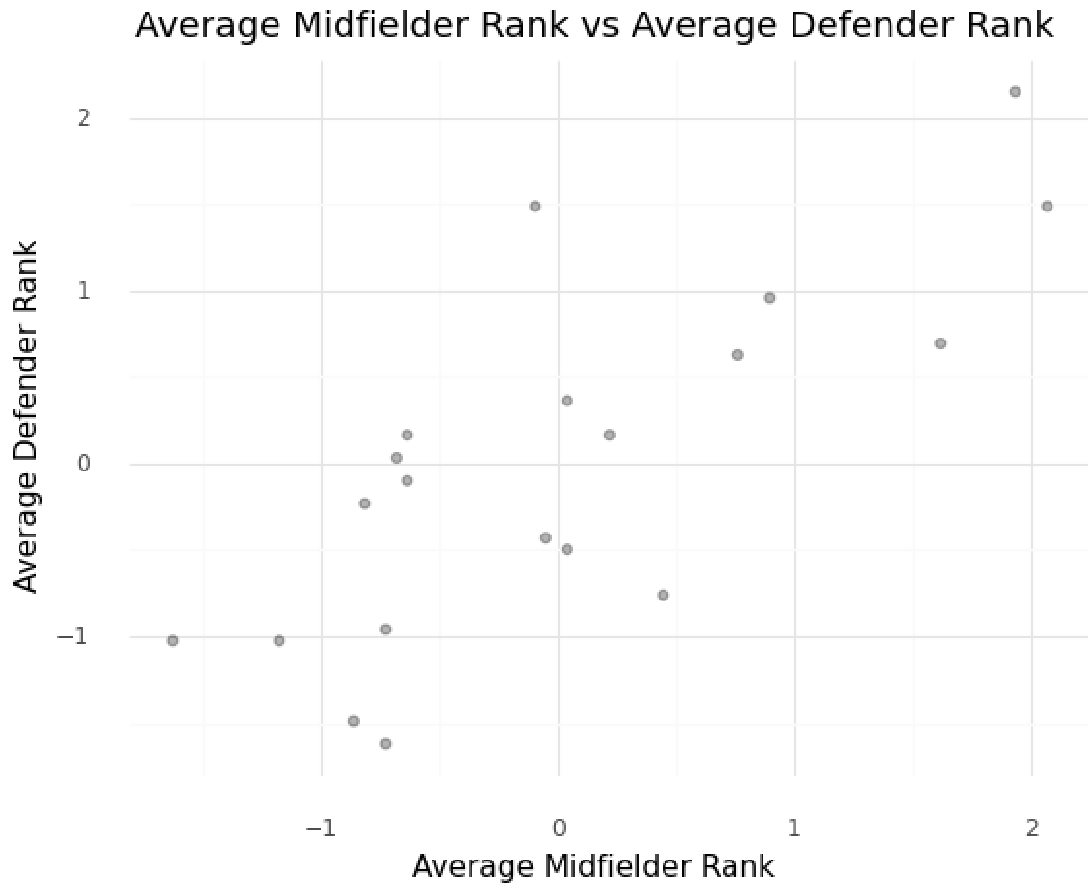
```
["appearances_overall"].mean())
  PremTeam.loc[PremTeam["Club"] == club, "Average Age"] =
round(prem.loc[prem["Current Club"] == club]["age"].mean())
  PremTeam.loc[PremTeam["Club"] == club, "Average Defender Rank"] =
round(prem.loc[prem["Current Club"] == club]
["rank_in_league_top_defenders"].mean())
  PremTeam.loc[PremTeam["Club"] == club, "Average Midfielder Rank"] =
round(prem.loc[prem["Current Club"] == club]
["rank_in_league_top_midfielders"].mean())
  PremTeam.loc[PremTeam["Club"] == club, "Average Forwards Rank"] =
round(prem.loc[prem["Current Club"] == club]
["rank_in_league_top_attackers"].mean())
  PremTeam.loc[PremTeam["Club"] == club, "Average Ranking"] =
round((PremTeam.loc[PremTeam["Club"] == club, "Average Defender Rank"]
+ PremTeam.loc[PremTeam["Club"] == club, "Average Midfielder Rank"] +
PremTeam.loc[PremTeam["Club"] == club, "Average Forwards Rank"]) /
3.0)
  PremTeam.loc[PremTeam["Club"] == club, "Goals"] =
prem.loc[prem["Current Club"] == club]["goals_overall"].sum()
  PremTeam.loc[PremTeam["Club"] == club, "Assists"] =
prem.loc[prem["Current Club"] == club]["assists_overall"].sum()
  PremTeam.loc[PremTeam["Club"] == club, "Red Cards"] =
prem.loc[prem["Current Club"] == club]["red_cards_overall"].sum()
  PremTeam.loc[PremTeam["Club"] == club, "Yellow Cards"] =
prem.loc[prem["Current Club"] == club]["yellow_cards_overall"].sum()

PremTeam.head()

                Club  Goals  Average Minutes Played  Assists  Clean
Sheets  \
0    West Ham United     51                  1252.0       33
7
1            Burnley     43                  1393.0       32
4
2  Huddersfield Town     21                  1170.0       13
5
3            Arsenal     69                  1212.0       52
7
4     Crystal Palace     48                  1253.0       33
5

   Average Appearances  Red Cards  Yellow Cards  Average Age  \
0                 18.0          1            60         30.0
1                 19.0          1            76         31.0
2                 17.0          4            60         28.0
3                 17.0          2            75         28.0
4                 17.0          2            61         30.0

   Average Defender Rank  Average Midfielder Rank  Average Forwards
Rank  \
```

```
0                    19.0                    152.0
149.0
1                    31.0                    154.0
166.0
2                    48.0                    199.0
179.0
3                    24.0                    139.0
147.0
4                    22.0                    135.0
126.0

   Average Ranking
0            107.0
1            117.0
2            142.0
3            103.0
4             94.0
```

Q3

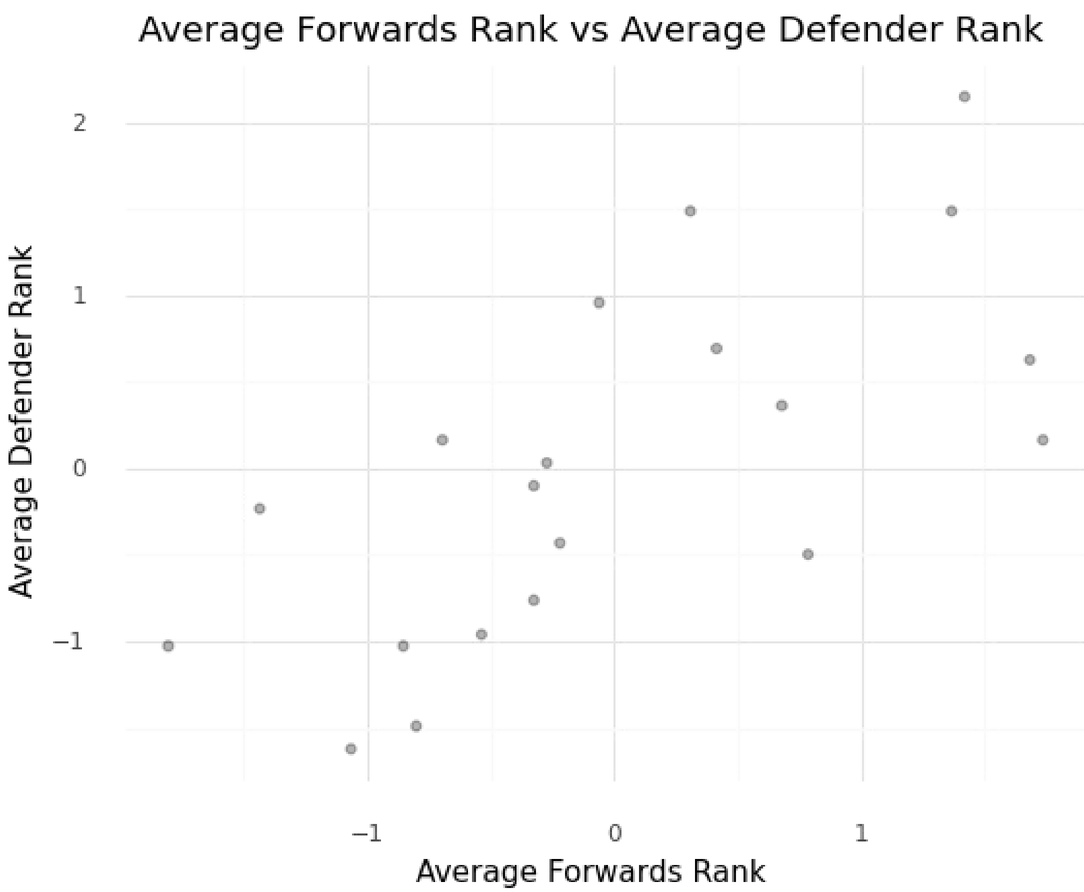
For this question, I wanted to do something more interesting to me personally, so I calculated either sums or averages for the continous variables used in previous models above and inputted them in for each team. Using these values, I created relationship comparison charts to compare average rank in league for defenders, midfielders, and forwards.

```
features = ["Average Defender Rank", "Average Midfielder Rank",
"Average Forwards Rank"]

PremTeamRanks = PremTeam[features]

TeamZ = StandardScaler()

PremTeamRanks[features] = TeamZ.fit_transform(PremTeamRanks[features])

(ggplot(PremTeamRanks, aes(y = "Average Defender Rank", x = "Average
Midfielder Rank"))
+ geom_point(alpha = 0.3) + ggtitle("Average Midfielder Rank vs
Average Defender Rank") +
 labs(y = "Average Defender Rank", x = "Average Midfielder Rank") +
theme_minimal())
```



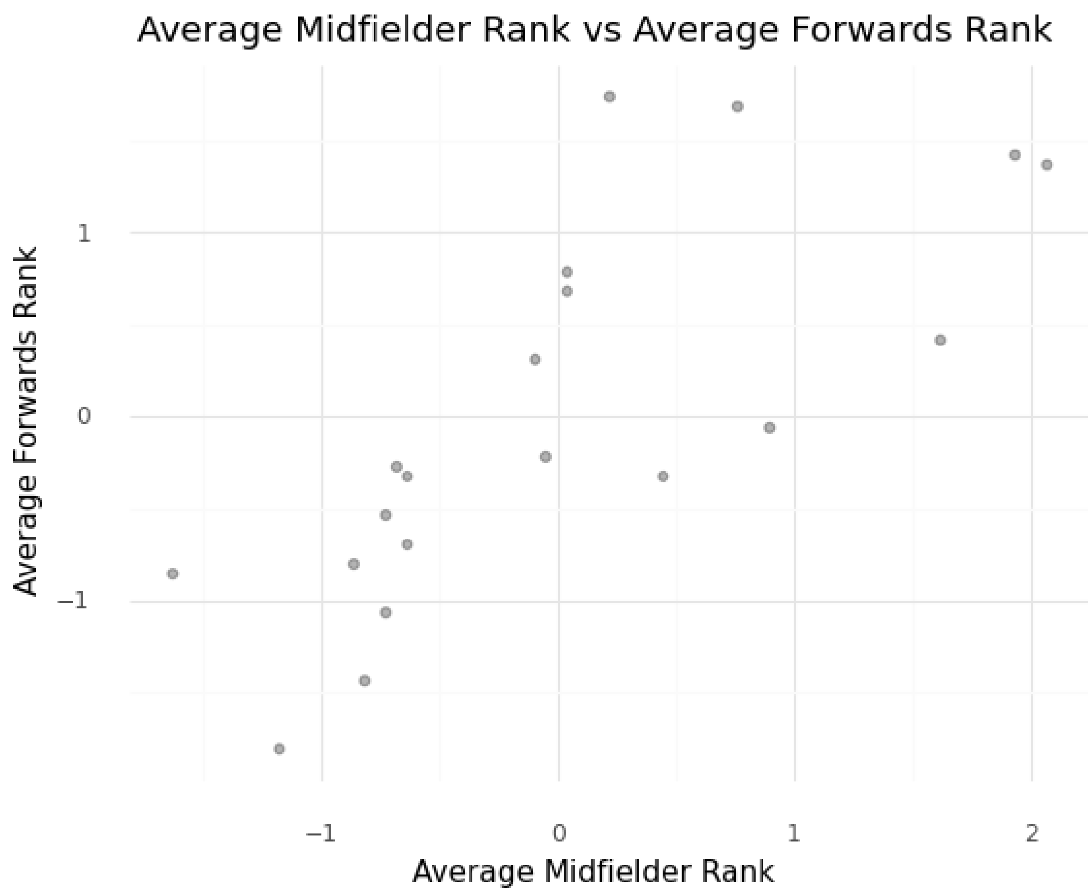

```
<ggplot: (8731155960625)>
```

```
(ggplot(PremTeamRanks, aes(y = "Average Defender Rank", x = "Average
Forwards Rank"))
+ geom_point(alpha = 0.3) + ggtitle("Average Forwards Rank vs Average
Defender Rank") +
 labs(y = "Average Defender Rank", x = "Average Forwards Rank") +
theme_minimal())
```



<ggplot: (8731155852009)>

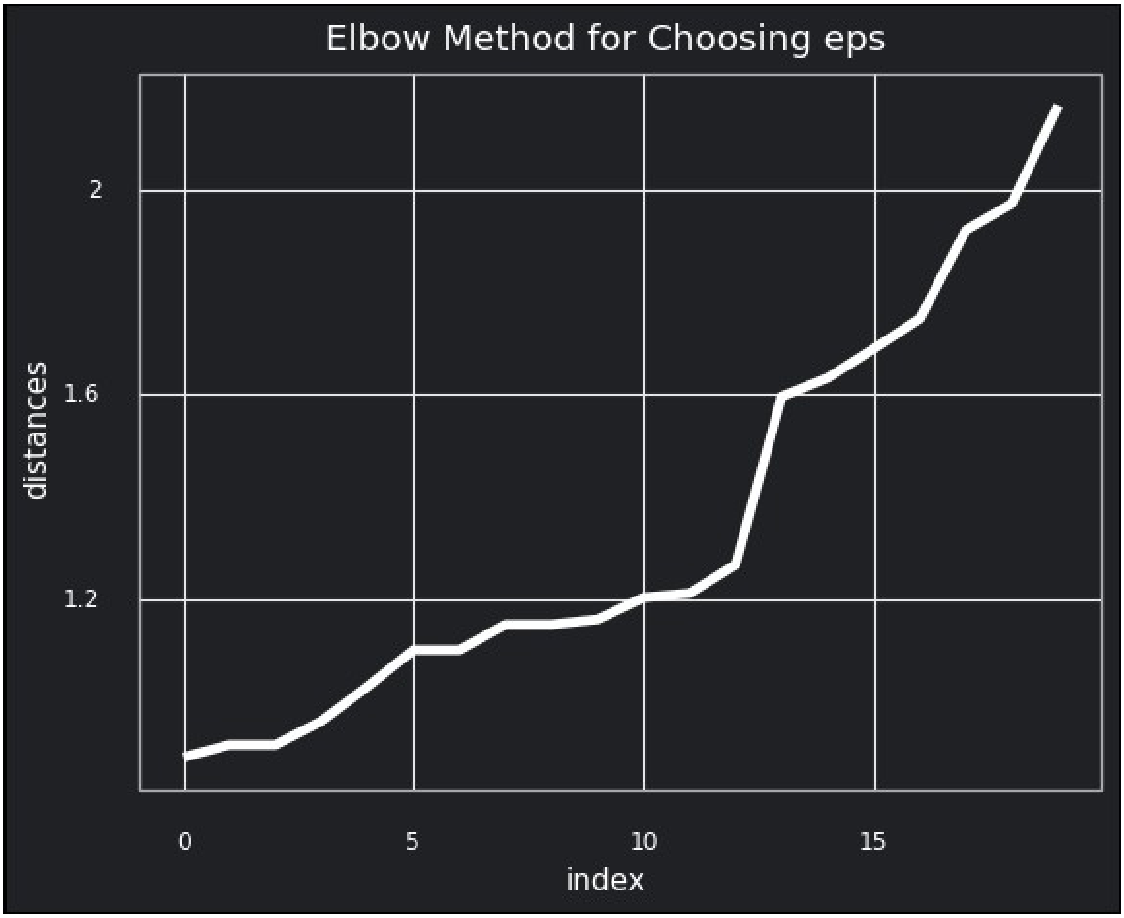
```
(ggplot(PremTeamRanks, aes(y = "Average Forwards Rank", x = "Average
Midfielder Rank"))
+ geom_point(alpha = 0.3) + ggtitle("Average Midfielder Rank vs
Average Forwards Rank") +
 labs(y = "Average Forwards Rank", x = "Average Midfielder Rank") +
theme_minimal())
```




```
<ggplot: (8731155852741)>
```

```
mins = 4
nn = NearestNeighbors(n_neighbors = mins + 1)

nn.fit(PremTeamRanks[features])

distances, neighbors = nn.kneighbors(PremTeamRanks[features])

distances = np.sort(distances[:, mins], axis = 0)

distances_df = pd.DataFrame({"distances": distances,
                             "index": list(range(0,len(distances)))})
plt = (ggplot(distances_df, aes(x = "index", y = "distances")) +
 geom_line(color = "white", size = 2) + theme_minimal() +
 labs(title = "Elbow Method for Choosing eps") +
 theme(panel_grid_minor = element_blank(),
      rect = element_rect(fill = "#202124ff"),
      axis_text = element_text(color = "white"),
      axis_title = element_text(color = "white"),
      plot_title = element_text(color = "white"),
      panel_border = element_line(color = "darkgray"),
      plot_background = element_rect(fill = "#202124ff")
```

```
       ))
ggsave(plot=plt, filename='elbow.png', dpi=300)

plt
```



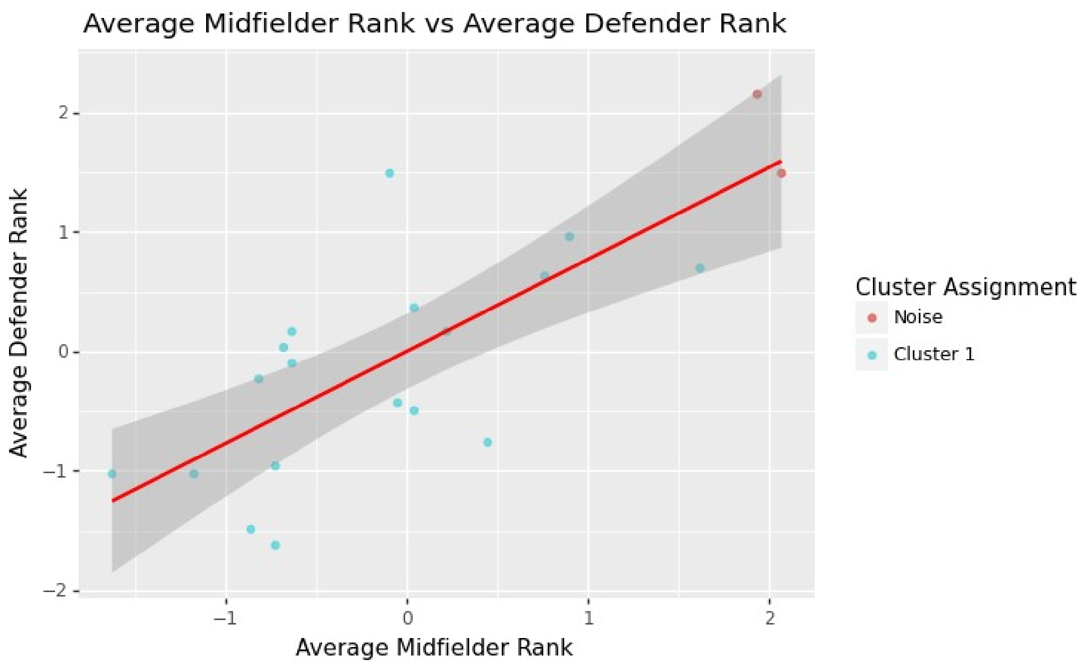

```
<ggplot: (8731155952185)>
```

Q3

This is the process by which we choose the distance required for a point to be considered a neighboring point to a cluster point. The elbow here determined the eps score to be 1.3, while I just used the default min points score of 4 since I did not know any better about the model.

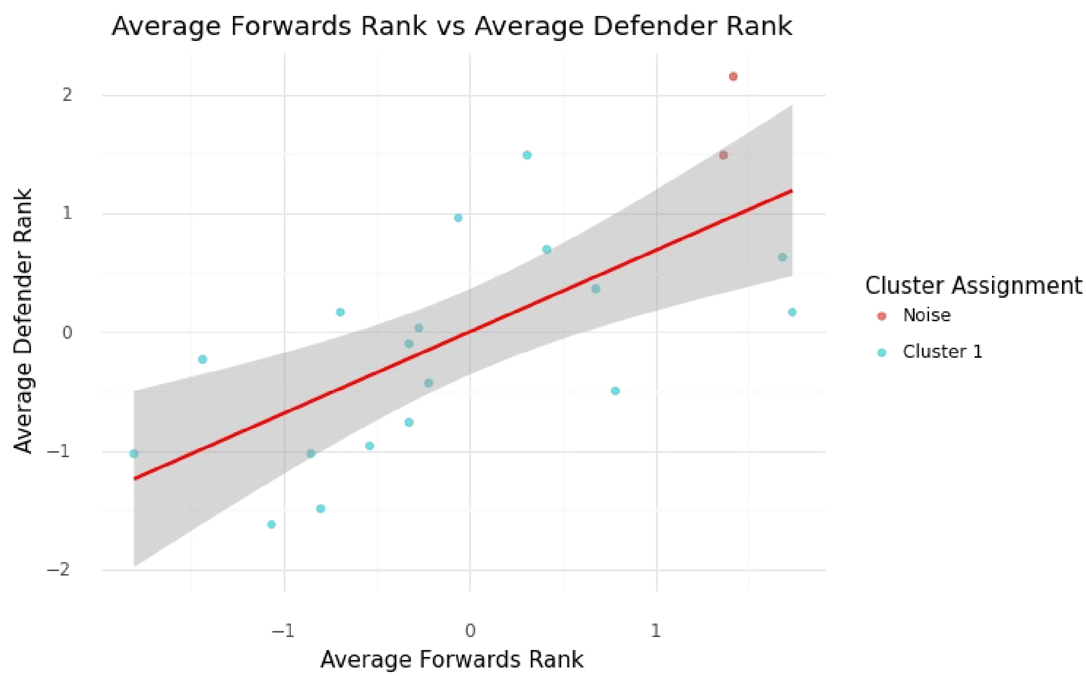
```
DBScan = DBSCAN(eps = 1.3, min_samples = 
4).fit(PremTeamRanks[features])

labsList = ["Noise"]
labsList = labsList  + ["Cluster " + str(i) for i in 
range(1,len(set(DBScan.labels_)))]
PremTeamRanks["assignments"] = DBScan.labels_
```

```
(ggplot(PremTeamRanks, aes(y = "Average Defender Rank", x = "Average
Midfielder Rank", color = "factor(assignments)"))
+ geom_point(alpha = 0.8) + scale_color_discrete(name = "Cluster
Assignment", labels = labsList) +
  ggtitle("Average Midfielder Rank vs Average Defender Rank") + labs(y
= "Average Defender Rank", x = "Average Midfielder Rank") +
stat_smooth(method = "lm", color = "red"))
```

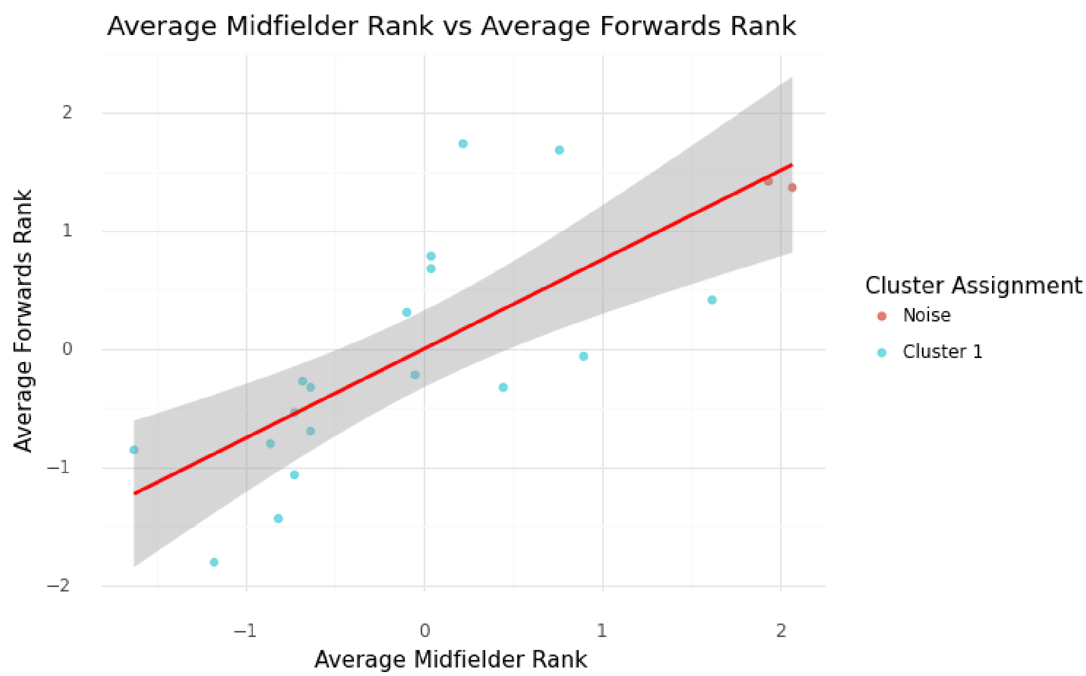



```
<ggplot: (8731155821221)>
```

```
(ggplot(PremTeamRanks, aes(y = "Average Defender Rank", x = "Average
Forwards Rank", color = "factor(assignments)"))
 + geom_point(alpha = 0.8) + scale_color_discrete(name = "Cluster
Assignment", labels = labsList) +
 ggtitle("Average Forwards Rank vs Average Defender Rank") +
 labs(y = "Average Defender Rank", x = "Average Forwards Rank") +
theme_minimal() + stat_smooth(method = "lm", color = "red"))
```




```
<ggplot: (8731155752605)>

(ggplot(PremTeamRanks, aes(y = "Average Forwards Rank", x = "Average
Midfielder Rank", color = "factor(assignments)"))
+ geom_point(alpha = 0.8) + scale_color_discrete(name = "Cluster
Assignment", labels = labsList) +
 ggtitle("Average Midfielder Rank vs Average Forwards Rank") +
 labs(y = "Average Forwards Rank", x = "Average Midfielder Rank") +
theme_minimal() + stat_smooth(method = "lm", color = "red"))
```



```
<ggplot: (8731155821297)>
```

Q3

I wanted to see if, using relationship graphs, I would be able to see different clusters that represented the gulf in quality between sets of team in the premier league. However, the amount of points being 20 and the similarity to each other prevented relevant clustering to take place. DBSCAN, with its epsilon chosen using the elbow method, resulted in only one cluster being created. Looking back on the creation of the question, I should've realized this would occur. We can discuss the "cluster" in terms of a linear relationship, however, as teams rise in quality the average ranking of their positions rises as well. I thought it was interesting how the noise points, points which are not similar enough to be included in a cluster are at the top right. I would assume these are Manchester City and Liverpool, who were significantly better than every other team in the Premier League in 2018/19 (98 and 97 points, while third place was in the 70s). Perhaps DBSCAN, with its usage of point density, was not the best option to use for clustering this data.

```
PremTeamRanks = PremTeam[["Average Ranking", "Club"]]
PremTeamRanks.sort_values(by=["Average Ranking"])
```

	Average Ranking	Club
17	85.0	Chelsea
15	88.0	Tottenham Hotspur
7	90.0	Liverpool
19	92.0	Manchester City
4	94.0	Crystal Palace
9	97.0	Wolverhampton Wanderers
5	102.0	Watford
3	103.0	Arsenal
14	104.0	Manchester United
0	107.0	West Ham United
11	108.0	Leicester City
18	113.0	Newcastle United
1	117.0	Burnley
8	119.0	AFC Bournemouth
12	122.0	Southampton
10	124.0	Everton
13	129.0	Cardiff City
16	130.0	Brighton & Hove Albion
2	142.0	Huddersfield Town
6	145.0	Fulham

Club		MP	W	D	L	GF	GA	GD	Pts	Last 5
1	Man. City	38	32	2	4	95	23	72	98	
2	Liverpool	38	30	7	1	89	22	67	97	
3	Chelsea	38	21	9	8	63	39	24	72	
4	Tottenham	38	23	2	13	67	39	28	71	
5	Arsenal	38	21	7	10	73	51	22	70	
6	Man United	38	19	9	10	65	54	11	66	
7	Wolves	38	16	9	13	47	46	1	57	
8	Everton	38	15	9	14	54	46	8	54	
9	Leicester City	38	15	7	16	51	48	3	52	
10	West Ham	38	15	7	16	52	55	-3	52	
11	Watford	38	14	8	16	52	59	-7	50	
12	Crystal Palace	38	14	7	17	51	53	-2	49	
13	Newcastle	38	12	9	17	42	48	-6	45	
14	Bournemouth	38	13	6	19	56	70	-14	45	
15	Burnley FC	38	11	7	20	45	68	-23	40	
16	Southampton	38	9	12	17	45	65	-20	39	
17	Brighton	38	9	9	20	35	60	-25	36	
18	Cardiff City	38	10	4	24	34	69	-35	34	
19	Fulham	38	7	5	26	34	81	-47	26	
20	Huddersfield	38	3	7	28	22	76	-54	16	

Q3

As more of personal interest, I sorted all the teams in ascending order by their average position ranking. I wanted to see if the order created here was similar to what their order was in the actual premier league table. The order created was completely different, which is something I also should have expected. While it is obvious the teams with generally higher rankings are higher on the actual table, you can't use something like this to predict league tables. So many factors go into actual soccer games including things like fan presence, coaches, and the fact that football is a very team-oriented sport. Sometimes, the eye test is just as important as the statistics that might back it up.