CSI4106 - Introduction to Artificial Intelligence

Project 1 - Classification Empirical Study

Fall 2022

School of Electrical Engineering and Computer Science

Professor : Caroline Barrière

Group : 9

Student 1: Alexander Onofrei, #300089694

Student 2: Anton Pellerin-Petrov, #300092454

Submission Date: November 1st, 2022

The goal of this project is to complete both the classification study and it's documentation. This project will detail the steps taken in order to set-up the this classification. Furthermore, 3 different classification alogirthm will be used:

- Naïve Bayes
- Logisitc Regression
- Multi-Layer Perceptron

For each model, there will be data preparation, model training, testing using cross-validation, evaluation using precision/recall measures, parameter modification and finally result analysis.

1. Understanding the classification task for our dataset

In order to complete this project, we'll be using a dataset containing an overview of all internation men's team soccer matches played since the 90's. With this data, we'll be able to train our model in order to predict the winner and loser of past and future FIFA matches. On top of doing the aforementioned tasks for each model, we will chose the most precise model and predict the winner of the FIFA WORLD CUP 2022. This means we will predict the winners of each group stage until the very end: the winner of the world's cup final. Hence, this prediction can be used for betting across multiple sport betting plateforms or to simply pique one's curiosity.

In order to do so, we've chosen the following dataset from Brenda Loznik which can be found here. This dataset is a binary classification because for each football match registered, the teams are classified as winner or loosers. The given data can only be classified into two classes.

In [ ]:
```
# general imports
```

```python
import sklearn
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
from statistics import mean
from tabulate import tabulate
```

2. Analyzing our dataset

The following section will analyze our data by providing training examples, features and missing data of the selected dataset.

Some data representation regarding the FIFA ranking of the top 8 FIFA men's teams is also shown in order to represent the variation of their ranking across the years.

2.1 Training Examples

Some training examples that can be made for this dataset set include:

- Does home team advantage truely exist?
- Longest winning streak by any team.
- Does a stronger offense rating means more scoring goals? Does a stronger defense means receiving more goals?
- What are the chances of the strongest rated team winning the match?

2.2 Number of Features

Below is a list of all the features for each world cup game and their data type. The strength of each team is based on the actual FIFA rankings and the players strength, for example the home_team_mean_defense_score, is based on the EA Sport FIFA video game ratings.

In [ ]:

```python
# Importing the data from our github repository
df_url = 'https://raw.githubusercontent.com/antonpp11/CSI4106-Project1-DB/master/inte
df = pd.read_csv(df_url)
df.dtypes
```

```
Out[ ]: date                              object
home_team                         object
away_team                         object
home_team_continent               object
away_team_continent               object
home_team_fifa_rank                int64
away_team_fifa_rank                int64
home_team_total_fifa_points        int64
away_team_total_fifa_points        int64
home_team_score                    int64
away_team_score                    int64
tournament                        object
city                              object
country                           object
neutral_location                    bool
shoot_out                         object
home_team_result                  object
home_team_goalkeeper_score       float64
away_team_goalkeeper_score       float64
home_team_mean_defense_score     float64
home_team_mean_offense_score     float64
home_team_mean_midfield_score    float64
away_team_mean_defense_score     float64
away_team_mean_offense_score     float64
away_team_mean_midfield_score    float64
dtype: object
```

2.3 Missing Data

As our dataset contains only the internation men's football games since summer of 2022, it does not contain the games that will be played at the FIFA World Cup. This means these games and all of their associated information will need to be added at the very end of the dataset. Therefore, the selected prediction model will be in charge of completing it by determinating the winner and looser of each game.

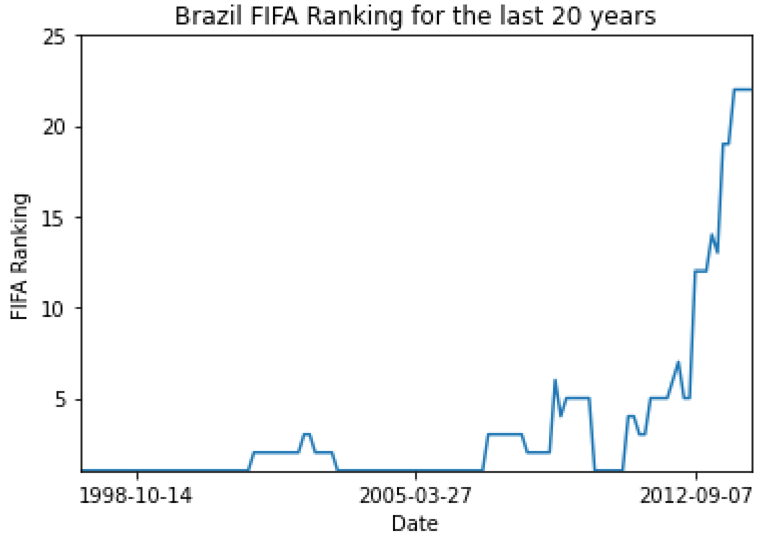
2.4 Dataset Data representation

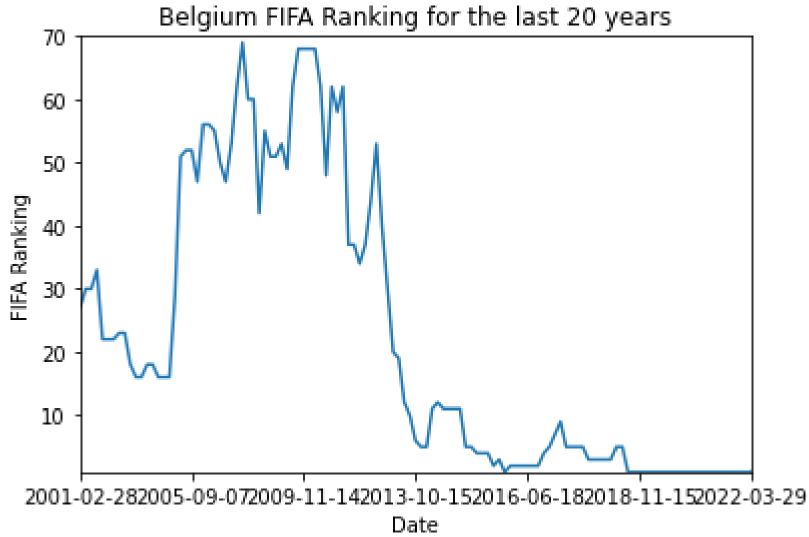
This section will seek to represent the current top 8 FIFA men's FIFA teams and their FIFA ranking from the past 10 years. This is only to show how much the FIFA ranking of a team can change over the years. This results to the fact that most recent years are more important to the model than older years.

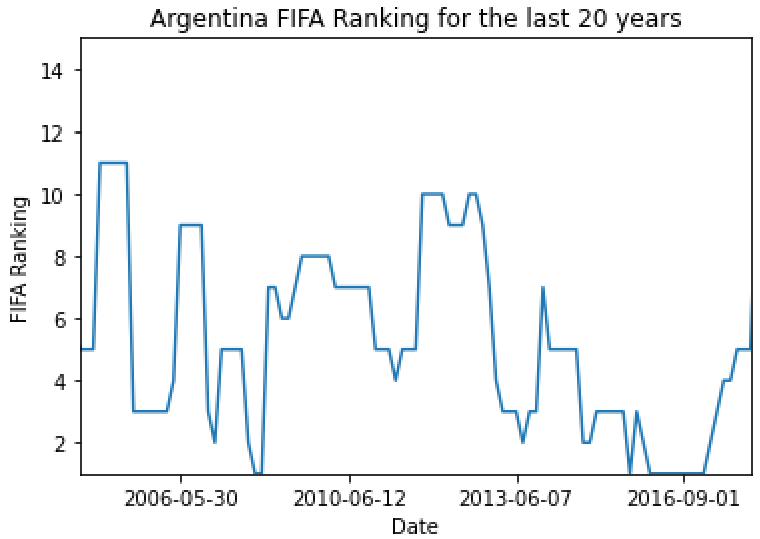
It is important to know that the lower the FIFA Ranking is, the better the team was that year. For example, a FIFA ranking of 2 is better than a FIFA ranking of 5. Also, it is impossible for 2 teams to have the same FIFA ranking for a specific year because each team has to have different FIFA ranking scores.

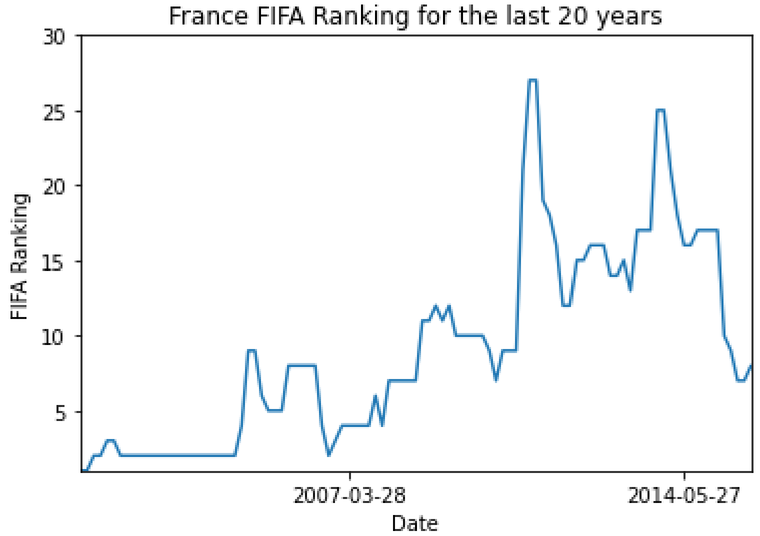
```
In [ ]: filtered_team = df.loc[((df['home_team']=='Brazil'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
plt.set_title('Brazil FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(40,160)
plt.set_ylim(1,25)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()


filtered_team = df.loc[((df['home_team']=='Belgium'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
```

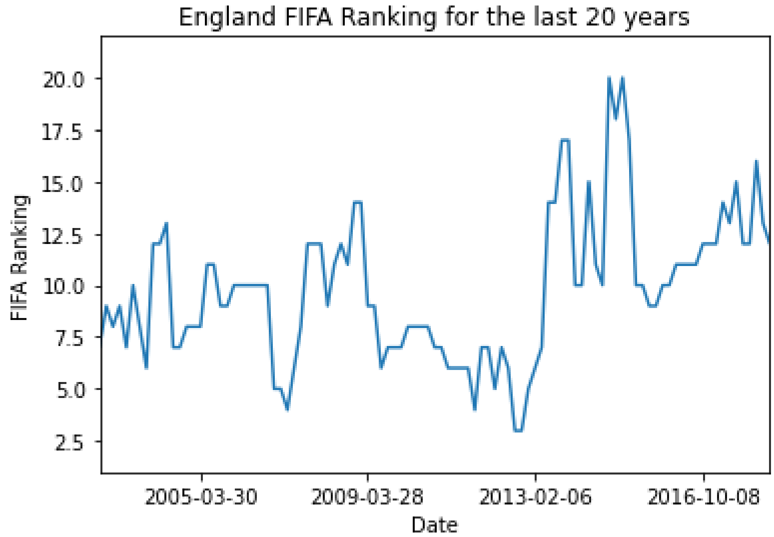
```
plt.set_title('Belgium FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(40,160)
plt.set_ylim(1,70)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()

filtered_team = df.loc[((df['home_team']=='Argentina'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
plt.set_title('Argentina FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(60,160)
plt.set_ylim(1,15)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()


filtered_team = df.loc[((df['home_team']=='France'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
plt.set_title('France FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(60,160)
plt.set_ylim(1,30)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()

filtered_team = df.loc[((df['home_team']=='England'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
plt.set_title('England FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(60,160)
plt.set_ylim(1,22)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()

filtered_team = df.loc[((df['home_team']=='Italy'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
plt.set_title('Italy FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(60,160)
plt.set_ylim(1,22)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()

filtered_team = df.loc[((df['home_team']=='Spain'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
plt.set_title('Spain FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(60,160)
plt.set_ylim(1,13)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()

filtered_team = df.loc[((df['home_team']=='Netherlands'))]
plt = filtered_team.plot(x="date", y = "home_team_fifa_rank")
plt.set_title('Netherlands FIFA Ranking for the last 20 years')
plt.set_xlabel("Date")
plt.set_xlim(60,160)
plt.set_ylim(1,35)
plt.set_ylabel("FIFA Ranking")
plt.get_legend().remove()
```


Brazil FIFA Ranking for the last 20 years
25
20
15
10
5
FIFA Ranking
1998-10-14
2005-03-27
2012-09-07
Date


Belgium FIFA Ranking for the last 20 years
70
60
50
40
30
20
10
FIFA Ranking
2001-02-28
2005-09-07
2009-11-14
2013-10-15
2016-06-18
2018-11-15
2022-03-29
Date


Argentina FIFA Ranking for the last 20 years
14
12
10
8
6
4
2
FIFA Ranking
2006-05-30
2010-06-12
2013-06-07
2016-09-01
Date


France FIFA Ranking for the last 20 years
30
25
20
15
10
5
FIFA Ranking
2007-03-28
2014-05-27
Date


England FIFA Ranking for the last 20 years
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
FIFA Ranking
2005-03-30
2009-03-28
2013-02-06
2016-10-08
Date


Italy FIFA Ranking for the last 20 years
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
FIFA Ranking
2006-06-02
2009-09-09
2013-03-21
2016-05-29
Date

3. Brainstorming about the attributes (Feature Engineering)

As seen in the above table, our dataset contains a multiple number of useless features in predicting the winner of this year's FIFA World Cup. As all teams will be playing in the same continent and the same country, all related information to this will be removed. Therefore, the neutral_location variable will be kept because the country Qatar will be playing all home games. Hence, the value of this attribute for Qatar will be set to false but for all other countries it will be true. Also, as we seek only to predict the winner of each game, we will remove the home team and away team score. If we were to keep it, we would need to predict the score for each match of the world cup in order to predict its winner. It will then become a multi-class classification instead of a binary cllassification. Also, everything related to the rating of the players will be removed as it is not usefull for our prediction. Finally, the total fifa points for each team is directly related to their fifa ranking meaning that if we use the fifa ranking attribute, we do not need the total fifa points.

In [ ]:
```
# we can remove multiple features listed below
features_to_remove = ['city',
                      'home_team_continent',
                      'away_team_continent',
                      'home_team_score',
                      'away_team_score',
                      'tournament',
                      'country',
                      'shoot_out',
                      'home_team_total_fifa_points',
```

```
                                  'away_team_total_fifa_points',
                                  'home_team_goalkeeper_score',
                                  'away_team_goalkeeper_score',
                                  'home_team_mean_defense_score',
                                  'home_team_mean_offense_score',
                                  'home_team_mean_midfield_score',
                                  'away_team_mean_defense_score',
                                  'away_team_mean_offense_score',
                                  'away_team_mean_midfield_score']
df = df.drop(features_to_remove, axis=1)
```

Also, as we will be having a binary classification of winner or looser, we will ignore all matches that ended in a draw. Also, all the matches during the World Cup after the group stages must always end up with a winner: making the draw not possible.

In [ ]:
```
# As explained above, we will drop the draws from our training data because we
# to try to predict winners only
df = df[df['home_team_result'] != "Draw"]
```

Hence, the remaining attributes are:

In [ ]:
```
df.dtypes
```

Out[ ]:
```
date                       object
home_team                  object
away_team                  object
home_team_fifa_rank         int64
away_team_fifa_rank         int64
neutral_location             bool
home_team_result           object
dtype: object
```

The date attribute was kept because older matches have a less significant weight on the prediction. The most recent matches have a bigger weight because the players and staff of those teams will likely be the same during the world cup.

We also kept the home_team and away_team attributes because it is possible that a home team is more likely of winning the match. This is based on the home team advantage principle. More information about this advantage can be found this article from (Hegde, 2022).

The fifa ranks of each teams (home team and away team) are the most crucial attributes in predicting the winner of a match. In fact, a team with a higher fifa rank would be more prone in winning a match versus a lower rank fifa team. Also, the neutral_location attribute is kept because the country Qatar will be technicaly playing all home games. Therefore, the neutral_location will be set to true for the country Qatar but not for the other countries.

Finally, the home_team_result is the attribute that will be predicted by our models.It is a boolean type value, meaning that if it is true, the home team won and if it is false, it lost the game. Hence, if the home team won, the away team lost. This is why the away_team_result is not predicted by our models.

In the table below, you can see all the games of the 2022 FIFA World Cup in a different dataset, but using the same structure. As previously mentionned, the dataset only contained played games until summer 2022. As the World Cup is in November 2022, these games weren't part of our dataset.

In [ ]:
```
# we import the group stage file which contains all the games
gs_df_url = 'https://raw.githubusercontent.com/antonpp11/CSI4106-Project1-DB/master/g
gs_df = pd.read_csv(gs_df_url)
gs_df
```

Out[ ]:

	date	home_team	away_team	home_team_continent	away_team_continent	home_team_fifa_rank
0	2022-11-20	Qatar	Ecuador	NaN	NaN	50
1	2022-11-21	England	Iran	NaN	NaN	5
2	2022-11-21	Senegal	Netherlands	NaN	NaN	18
3	2022-11-21	USA	Wales	NaN	NaN	16
4	2022-11-22	Argentina	Saudi Arabia	NaN	NaN	3
5	2022-11-22	Denmark	Tunisia	NaN	NaN	10
6	2022-11-22	Mexico	Poland	NaN	NaN	13
7	2022-11-22	France	Australia	NaN	NaN	4
8	2022-11-23	Morocco	Croatia	NaN	NaN	22
9	2022-11-23	Germany	Japan	NaN	NaN	11
10	2022-11-23	Spain	Costa Rica	NaN	NaN	7
11	2022-11-23	Belgium	Canada	NaN	NaN	2
12	2022-11-24	Switzerland	Cameroon	NaN	NaN	15
13	2022-11-24	Uruguay	Korea Republic	NaN	NaN	14
14	2022-11-24	Portugal	Ghana	NaN	NaN	9
15	2022-11-24	Brazil	Serbia	NaN	NaN	1
16	2022-11-25	Wales	Iran	NaN	NaN	19
17	2022-11-25	Qatar	Senegal	NaN	NaN	50
18	2022-11-25	Netherlands	Ecuador	NaN	NaN	8
19	2022-11-25	England	USA	NaN	NaN	5
20	2022-11-26	Tunisia	Australia	NaN	NaN	30
21	2022-11-26	Poland	Saudi Arabia	NaN	NaN	26
22	2022-11-26	France	Denmark	NaN	NaN	4
23	2022-11-26	Argentina	Mexico	NaN	NaN	3
24	2022-11-27	Japan	Costa Rica	NaN	NaN	24
25	2022-11-27	Belgium	Morocco	NaN	NaN	2

	date	home_team	away_team	home_team_continent	away_team_continent	home_team_fifa_rank
26	2022-11-27	Croatia	Canada	NaN	NaN	12
27	2022-11-27	Spain	Germany	NaN	NaN	7
28	2022-11-28	Cameroon	Serbia	NaN	NaN	43
29	2022-11-28	Korea Republic	Ghana	NaN	NaN	28
30	2022-11-28	Brazil	Switzerland	NaN	NaN	1
31	2022-11-28	Portugal	Uruguay	NaN	NaN	9
32	2022-11-29	Ecuador	Senegal	NaN	NaN	44
33	2022-11-29	Netherlands	Qatar	NaN	NaN	8
34	2022-11-29	Wales	England	NaN	NaN	19
35	2022-11-29	Iran	USA	NaN	NaN	20
36	2022-11-30	Australia	Denmark	NaN	NaN	38
37	2022-11-30	Tunisia	France	NaN	NaN	30
38	2022-11-30	Poland	Argentina	NaN	NaN	26
39	2022-11-30	Saudi Arabia	Mexico	NaN	NaN	51
40	2022-12-01	Croatia	Belgium	NaN	NaN	12
41	2022-12-01	Canada	Morocco	NaN	NaN	41
42	2022-12-01	Japan	Spain	NaN	NaN	24
43	2022-12-01	Costa Rica	Germany	NaN	NaN	31
44	2022-12-02	Ghana	Uruguay	NaN	NaN	61
45	2022-12-02	Korea Republic	Portugal	NaN	NaN	28
46	2022-12-02	Serbia	Switzerland	NaN	NaN	21
47	2022-12-02	Cameroon	Brazil	NaN	NaN	43

48 rows × 25 columns

4. Encoding the features

Our models will use discrete data because each value in the model contains clear spaces between values. For example, the fifa ranking is of integer value between 1 and 60 meaning it is impossible to have a rank of 2.56. Same principle applies to goals: it is impossible for a team to score 4.5 goals.

In order to facilitate our model prediction and in order to have better results. We changed the date time format to timedelta integer. In this way, the model can properly predict values. If more than one game has been played during a same day, then we add the index to the timedelta value in order to separate properly each game played.

In [ ]:

```
# change dates to datetime format
df['date'] = pd.to_datetime(df['date'])

# first game from our dataset
first_game_date = df['date'][0]

# create timestamps by substracting dates using the first game of our dataset
df['timedelta_int'] = ((df['date'] - first_game_date).dt.total_seconds() + df.index).
df = df.drop('date', axis=1)
```

For data encoding, we decided to use one-hot encoding. Each team that has been part internation men's FIFA games since the 90s will be one-hot encoded. Each home_team will have the prefix "h_" added to it's name and each away team will have the "a_" prefix added. As each team played a home and away game it will contain an encoding for each of the 2 prefixes. For example, "canada" will become "h_canada" and "a_canada" with a one-hot encoding.

In [ ]:

```
# perform one hot encoding for home and away teams
df = pd.get_dummies(
    data=df,
    columns=["home_team", "away_team"],
    prefix=["h", "a"])
```

In [ ]:

```
# we can now take a look at our data, it looks ready for use
df.dtypes
```

Out[ ]:

```
home_team_fifa_rank     int64
away_team_fifa_rank     int64
neutral_location         bool
home_team_result       object
timedelta_int           int64
                        ...
a_Vietnam               uint8
a_Wales                 uint8
a_Yemen                 uint8
a_Zambia                uint8
a_Zimbabwe              uint8
Length: 427, dtype: object
```

5. Preparing the data for the experiment using cross-validation

In order to prepare the data from the dataset for the experiment, we must follow 2 steps:

1. Extract the attribute associated to the class we will predict. In our case that attribute is home_team_result. This means the feature will contain every attribute except the one mentionned above. In our design, we also keep a copy of the feature for further use.

2. Separate the dataset into a test group and a train group. Our train group size represents 75% of the dataset and the test group is composed of the remaining 25%. It is important to note that we use all the games from the dataset from the first one recorded to the latest (Except for games that finished as a draw as mentionned in previous sections)

It's important to know that for our FIFA World Cup prediction section, we also created a extra dataset used to predict the future games of the World Cup that aren't in the main dataset extracted from Kaggle.

3. In order to have a better use for our data, we must scale it using the StandardScaler. To do so, we fit tranform the X_train value and we transform the the X_test value.

We also create a table in order to store the resulting game results. Therefore, this can be optional.

In [ ]:
```
## prepare dataset by splitting training and test data
from sklearn.model_selection import train_test_split

# get the features columns
features = list(df.columns)
features.remove('home_team_result')

# keep a copy of the features and all the columns
all_teams_columns = list(df.columns).copy()
all_teams_features = features.copy()

X = df.loc[:, features]
y = df.loc[:, ['home_team_result']]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, train_size
y_train = y_train.values.ravel()
```

In [ ]:
```
# Scaling the data for better use
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [ ]:
```
# We prepare a score list to in which we will add all the testing scores of all model
scores = []
```

6. Naïve Bayes

6.1. Naïve Bayes with default parameters

6.1.1. Naïve Bayes training with default parameters

As shown in the code below, we start be creating a Naïve Bayes classifier with default parameters.

We will use the article from (scikit-learn developers, 2011) to tweak our parameters.

In [ ]:
```
# Import the Naive Bayes model from sklearn
from sklearn.naive_bayes import GaussianNB

# Create an instance of the Naive Bayes model with default parameters
nb_model = GaussianNB()

# Train the model using the training sets
nb_model.fit(X_train,y_train)
```

```
# Get the default parameters for later use
nb_default_params = nb_model.get_params().copy()
```

6.1.2. Naïve Bayes testing with default parameters

Here, we seek to predict the values based on our X_test values.

In [ ]:
```
# Do a prediction on the test data
y_pred = nb_model.predict(X_test)
```

6.1.3. Naïve Bayes evaluation with default parameters

The results below show that the precision is high, but the recall, accuracy and F1 are extremely below average.

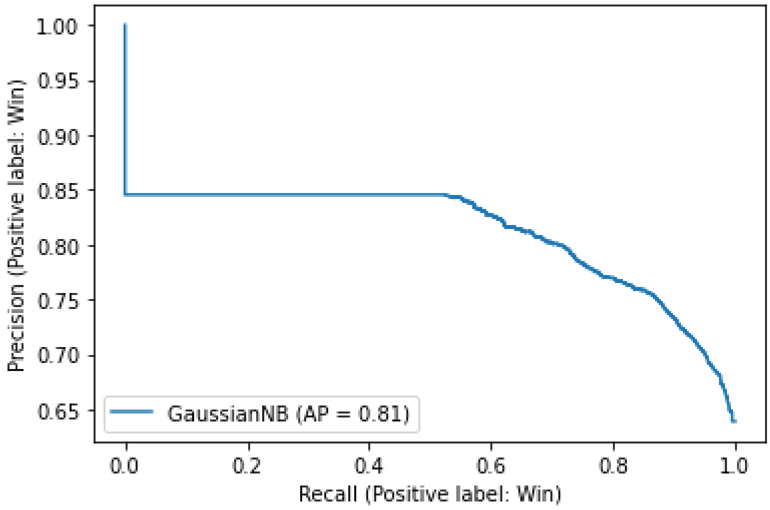
To evaluate our model, we will use the article from (scikit-learn developers, 2010) to find what measures we want to use. In our case, we want to use the following:

- Accuracy
- Precision
- Recall
- F1

Later, when trying to find the best model, we will determine which measures to use for a final decision.

In [ ]:
```
# Ignore warnings for graph creation
import warnings
warnings.filterwarnings('ignore')

# Import scikit-learn metrics module for evaluation
import sklearn.metrics

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(nb_model).__name__,
    "params": dict(nb_model.get_params().items() - nb_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(nb_model, X_test, y_test)
```

```
{'precision': 0.8075060532687651, 'recall': 0.6757852077001013, 'accuracy': 0.6898338
009928772, 'f1': 0.7357970215113072, 'average': 0.7272305208682627, 'model': 'Gaussia
nNB', 'params': {}}
```



6.2. Naïve Bayes with 1st iteration of parameters

6.2.1. Naïve Bayes training with 1st iteration of parameters

We inspire ourselves from (Sagir, 2019)'s article which shows how we can change Naïve Bayes parameters.

The Naïve Bayes model has very few parameters we can tweak:

- priors : Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.
- var_smoothing : Portion of the largest variance of all features that is added to variances for calculation stability.

As we can see in the above definitions, we shouldn't touch the 'priors' parameter because it will not adjust the model.

The only parameter left to change is 'var_smoothing'. By default, it is set to '1e-9'.

We will try to make it's value a lot bigger by changing it to '1e0'.

In [ ]:
```
# Create an instance of the Naive Bayes model with 1st iteration of parameters
nb_model = GaussianNB(var_smoothing=1e0)

# Train the model using the training sets
nb_model.fit(X_train,y_train)
```

Out[ ]:
```
GaussianNB(var_smoothing=1.0)
```

6.2.2. Naïve Bayes testing with 1st iteration of parameters

In [ ]:
```
# Do a prediction on the test data
y_pred = nb_model.predict(X_test)
```

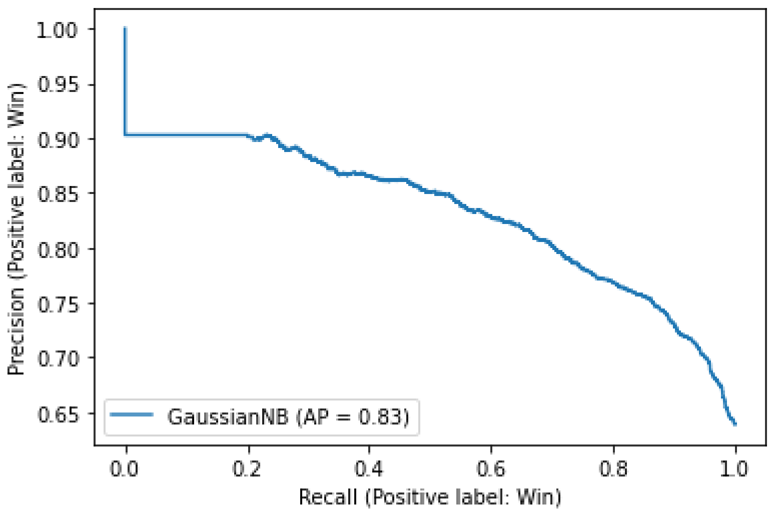
6.2.3. Naïve Bayes evaluation with 1st iteration of parameters

The results below show that the precision has dropped a little bit, and that the other values are also not very high.

In [ ]:
```
# Ignore warnings for graph creation
warnings.filterwarnings('ignore')
```

```
# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(nb_model).__name__,
    "params": dict(nb_model.get_params().items() - nb_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(nb_model, X_test, y_test)
```

```
{'precision': 0.7968097227497152, 'recall': 0.7085444106720703, 'accuracy': 0.6982516
727822146, 'f1': 0.7500893814801572, 'average': 0.7384237969210393, 'model': 'Gaussia
nNB', 'params': {'var_smoothing': 1.0}}
```



6.3. Naïve Bayes with 2nd iteration of parameters

6.3.1. Naïve Bayes training with 2nd iteration of parameters

We tried to make 'var_smoothing' a lot bigger, let's try to make it a lot smaller than the default.

We will change it to '1e-15'.

In [ ]:
```
# Create an instance of the Naive Bayes model with 2nd iteration of parameters
nb_model = GaussianNB(var_smoothing=1e-15)

# Train the model using the training sets
nb_model.fit(X_train,y_train)
```

Out[ ]:
```
GaussianNB(var_smoothing=1e-15)
```

6.3.2. Naïve Bayes testing with 2nd iteration of parameters

In [ ]:
```
# Do a prediction on the test data
```

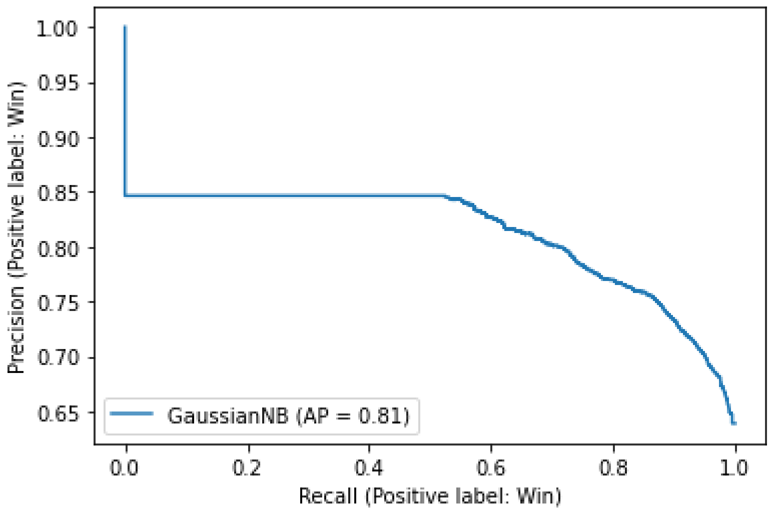
```
y_pred = nb_model.predict(X_test)
```

6.3.3. Naïve Bayes evaluation with 2nd iteration of parameters

The results below show that there isn't much difference between this variation and the default version. Overall, the scores of the Naïve Bayes classifier are not very strong.

In [ ]:
```
# Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(nb_model).__name__,
    "params": dict(nb_model.get_params().items() - nb_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(nb_model, X_test, y_test)
```

```
{'precision': 0.8075060532687651, 'recall': 0.6757852077001013, 'accuracy': 0.6898338
009928772, 'f1': 0.7357970215113072, 'average': 0.7272305208682627, 'model': 'Gaussia
nNB', 'params': {'var_smoothing': 1e-15}}
```



7. Logistic Regression

7.1. Logistic Regression with default parameters

7.1.1. Logistic Regression training with default parameters

We create a Logistic Regression model using the default parameters.

We will use the article from (scikit-learn developers, 2014) to tweak our parameters.

```
In [ ]: # Import the Logistic Regression model from sklearn
from sklearn.linear_model import LogisticRegression

# Create an instance of the Naive Bayes model with default parameters
lr_model = LogisticRegression()

# Train the model using the training sets
lr_model.fit(X_train,y_train)

# Get the default parameters for later use
lr_default_params = lr_model.get_params().copy()
```

7.1.2. Logistic Regression testing with default parameters

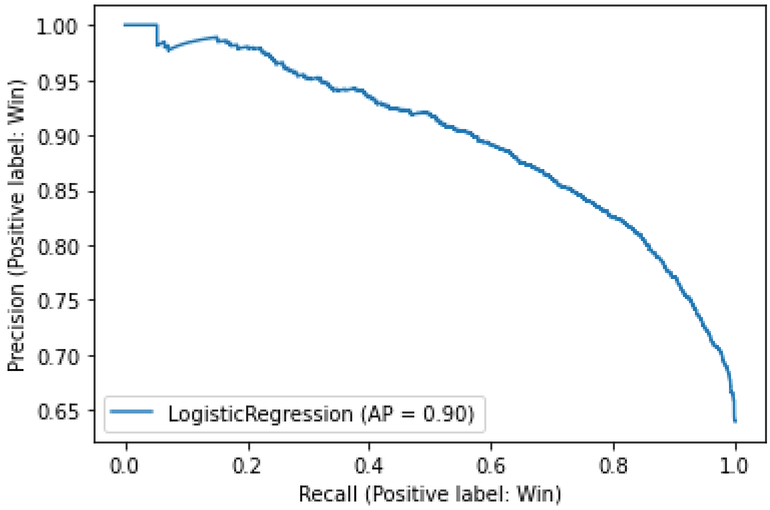
```
In [ ]: # Do a prediction on the test data
y_pred = lr_model.predict(X_test)
```

7.1.3. Logistic Regression evaluation with default parameters

The results show an improved score in every category compared to the Naïve Bayes model.

```
In [ ]: # Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(lr_model).__name__,
    "params": dict(lr_model.get_params().items() - lr_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(lr_model, X_test, y_test)
```

```
{'precision': 0.8035714285714286, 'recall': 0.851063829787234, 'accuracy': 0.77185409
02223181, 'f1': 0.8266360505166476, 'average': 0.8132813497744071, 'model': 'Logistic
Regression', 'params': {}}
```



7.2. Logistic Regression with 1st iteration of parameters

7.2.1. Logistic Regression training with 1st iteration of parameters

We use the article from (Stojiljković, 2019) to inspire ourselves for parameters modifications for the Logistic Regression model.

Online research shows that a lot of tweaks can be made to the parameters. In our case, we will modify the following:

- 'solver': Algorithm to use in the optimization problem. Default is 'lbfgs'. We will change it to 'newton-cg'
- 'C': Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization. Default is 1.0. We will change it to 0.01
- 'max_iter': Maximum number of iterations taken for the solvers to converge. Default is 100. We will augment the number of iterations to 1000.

In [ ]:
```
# Create an instance of the Naive Bayes model with 1st iteration of parameters
lr_model = LogisticRegression(solver='newton-cg', C=0.01, max_iter=1000)

# Train the model using the training sets
lr_model.fit(X_train,y_train)
```

Out[ ]:
```
LogisticRegression(C=0.01, max_iter=1000, solver='newton-cg')
```

7.2.2. Logistic Regression testing with 1st iteration of parameters

In [ ]:
```
# Do a prediction on the test data
y_pred = lr_model.predict(X_test)
```

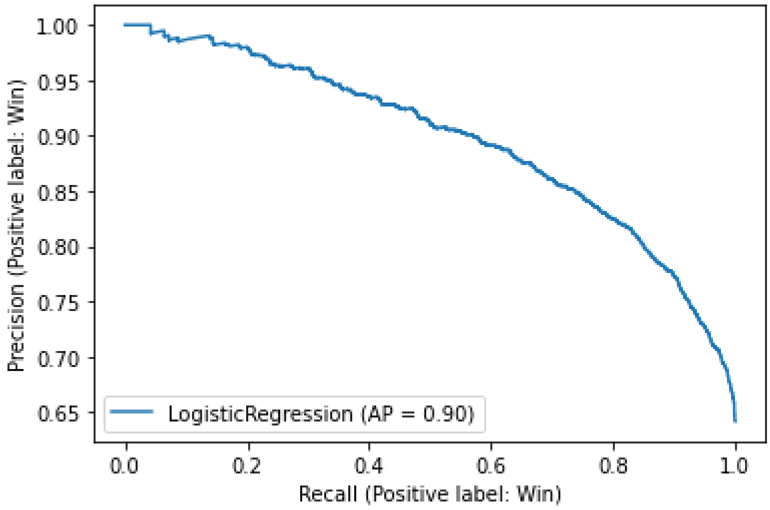
7.2.3. Logistic Regression evaluation with 1st iteration of parameters

As we can see, we have a very high recall rate, so the changes we applied are very interesting. However, precision was lost.

In [ ]:
```
# Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
```

```
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(lr_model).__name__,
    "params": dict(lr_model.get_params().items() - lr_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(lr_model, X_test, y_test)
```

```
{'precision': 0.7959824231010671, 'recall': 0.856467409658899, 'accuracy': 0.76796891
86272394, 'f1': 0.82511794371238, 'average': 0.8113841737748964, 'model': 'LogisticRe
gression', 'params': {'max_iter': 1000, 'solver': 'newton-cg', 'C': 0.01}}
```



7.3. Logistic Regression with 2nd iteration of parameters

7.3.1. Logistic Regression training with 2nd iteration of parameters

To cope with the changes of the last iteration, we will try to augment 'C' to 0.1. We will also upgrade the number of iterations to 10 000.

In [ ]:
```
# Create an instance of the Naive Bayes model with 2nd iteration of parameters
lr_model = LogisticRegression(solver='newton-cg', C=0.1, max_iter=10000)

# Train the model using the training sets
lr_model.fit(X_train,y_train)
```

Out[ ]:
```
LogisticRegression(C=0.1, max_iter=10000, solver='newton-cg')
```

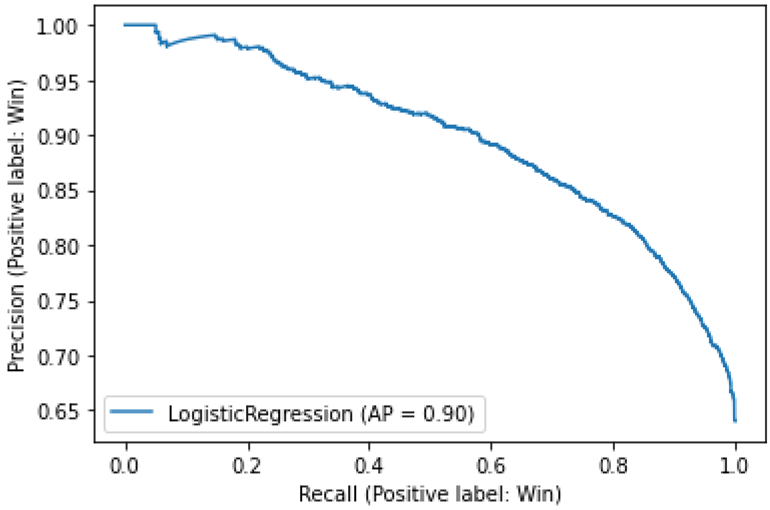
7.3.2. Logistic Regression testing with 2nd iteration of parameters

In [ ]:
```
# Do a prediction on the test data
y_pred = lr_model.predict(X_test)
```

7.3.3. Logistic Regression evaluation with 2nd iteration of parameters

In [ ]:
```
# Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(lr_model).__name__,
    "params": dict(lr_model.get_params().items() - lr_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(lr_model, X_test, y_test)
```

```
{'precision': 0.8028662420382165, 'recall': 0.8514015535292131, 'accuracy': 0.7714224
044895316, 'f1': 0.8264218980495002, 'average': 0.8130280245266154, 'model': 'Logisti
cRegression', 'params': {'C': 0.1, 'max_iter': 10000, 'solver': 'newton-cg'}}
```



8. Multi-Layer Perceptron

8.1. Multi-Layer Perceptron with default parameters

8.1.1. Multi-Layer Perceptron training with default parameters

We create our default MLP model.

We will use this article from (scikit-learn developers, 2010) to tweak our parameters.

In [ ]:
```
# Import the Multi-Layer Perceptron model from sklearn
from sklearn.neural_network import MLPClassifier

# Create an instance of the MLP model with default parameters
mlp_model = MLPClassifier()
```

```
# Train the model using the training sets
mlp_model.fit(X_train,y_train)

# Get the default parameters for later use
mlp_default_params = mlp_model.get_params().copy()
```

8.1.2. Multi-Layer Perceptron testing with default parameters

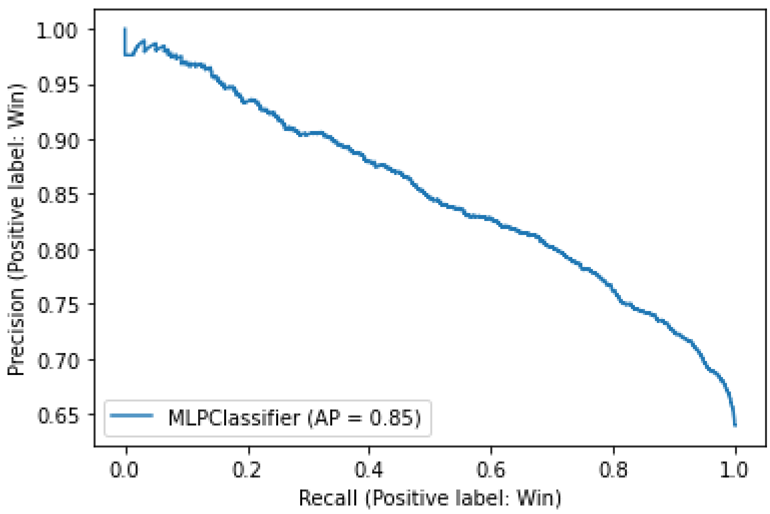
In [ ]:
```
# Do a prediction on the test data
y_pred = mlp_model.predict(X_test)
```

8.1.3. Multi-Layer Perceptron evaluation with default parameters

We get pretty average results with the default parameters.

In [ ]:
```
# Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(mlp_model).__name__,
    "params": dict(mlp_model.get_params().items() - mlp_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(mlp_model, X_test, y_test)
```

```
{'precision': 0.7683448502796972, 'recall': 0.7885849375211077, 'accuracy': 0.7129289
876969566, 'f1': 0.7783333333333334, 'average': 0.7620480272077738, 'model': 'MLPClas
sifier', 'params': {}}
```



8.2. Multi-Layer Perceptron with 1st iteration of parameters

8.2.1. Multi-Layer Perceptron training with 1st iteration of parameters

To change our parameters for the MLP Classifier, we will use the article from (Panjeh, 2020) for inspiration.

We will change the following parameters:

- hidden_layer_sizes: The ith element represents the number of neurons in the ith hidden layer. Default is (100,). We will change it to (50,).
- activation: Activation function for the hidden layer. Default is 'relu'. We will change it to the hyperbolic tan function (f(x) = tanh(x)), so it will be 'tanh'
- alpha: Strength of the L2 regularization term. The L2 regularization term is divided by the sample size when added to the loss. Default is 0.0001. We will change it to a much bigger number; 1.

In [ ]:

```
# Create an instance of the MLP model with 1st iteration of parameters
mlp_model = MLPClassifier(hidden_layer_sizes=(50,), activation='tanh', alpha=1)
# Train the model using the training sets
mlp_model.fit(X_train,y_train)
```

Out[ ]:

```
MLPClassifier(activation='tanh', alpha=1, hidden_layer_sizes=(50,))
```

8.2.2. Multi-Layer Perceptron testing with 1st iteration of parameters

In [ ]:

```
# Do a prediction on the test data
y_pred = mlp_model.predict(X_test)
```

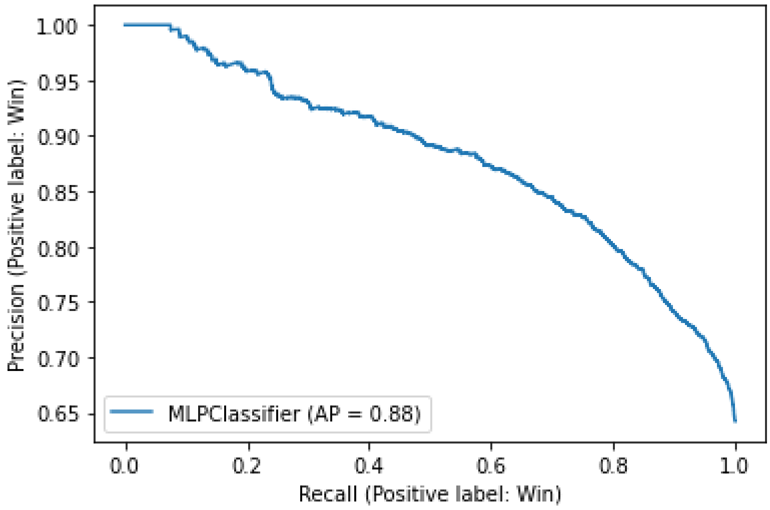
8.2.3. Multi-Layer Perceptron evaluation with 1st iteration of parameters

The results shown below show a lot of improvement over the default parameters.

In [ ]:

```
# Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(mlp_model).__name__,
    "params": dict(mlp_model.get_params().items() - mlp_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(mlp_model, X_test, y_test)
```

```
{'precision': 0.7915713819013395, 'recall': 0.8183046268152652, 'accuracy': 0.7461687
891215195, 'f1': 0.8047160411823315, 'average': 0.7901902097551139, 'model': 'MLPClas
sifier', 'params': {'hidden_layer_sizes': (50,), 'activation': 'tanh', 'alpha': 1}}
```



8.3. Multi-Layer Perceptron with 2nd iteration of parameters

8.3.1. Multi-Layer Perceptron training with 2nd iteration of parameters

In [ ]:
```
# Create an instance of the MLP model with 2nd iteration of parameters
mlp_model = MLPClassifier(hidden_layer_sizes=(150), max_iter=300, alpha=1, learning_r
# Train the model using the training sets
mlp_model.fit(X_train,y_train)
```

Out[ ]:
```
MLPClassifier(alpha=1, hidden_layer_sizes=150, learning_rate='adaptive',
              max_iter=300)
```

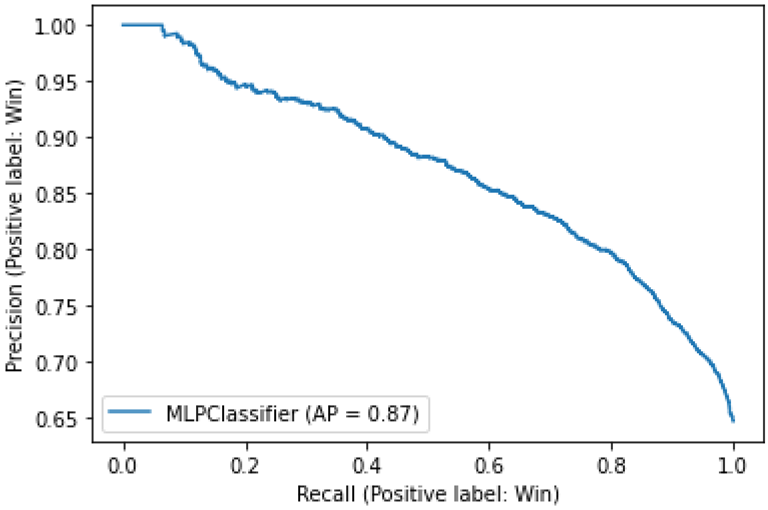
8.3.2. Multi-Layer Perceptron testing with 2nd iteration of parameters

In [ ]:
```
# Do a prediction on the test data
y_pred = mlp_model.predict(X_test)
```

8.3.3. Multi-Layer Perceptron evaluation with 2nd iteration of parameters

In [ ]:
```
# Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(mlp_model).__name__,
    "params": dict(mlp_model.get_params().items() - mlp_default_params.items())
}

scores.append(score)

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(mlp_model, X_test, y_test)
```

```
{'precision': 0.788567716791166, 'recall': 0.8199932455251604, 'accuracy': 0.74444204
61903735, 'f1': 0.8039735099337748, 'average': 0.7892441296101187, 'model': 'MLPClass
ifier', 'params': {'max_iter': 300, 'learning_rate': 'adaptive', 'alpha': 1, 'hidden_
layer_sizes': 150}}
```



9. Analyzing the obtained results

Now that we tested and evaluated all the different models, we will try to decide what model with which parameters we should use.

We will get the highest scores using the precision, recall, accuraccy and f1 scores.

In [ ]:

```
# Most Precise Model
most_precise = max(scores, key=lambda x:x['precision'])

# Most Recall Model
most_recall = max(scores, key=lambda x:x['recall'])

# Most Accurate Model
most_accurate = max(scores, key=lambda x:x['accuracy'])

# Most F1 Model
most_f1 = max(scores, key=lambda x:x['f1'])
```

In [ ]:

```
# Show a table containing the outputted results
data = [
    ["Highest Attribute", "Model Name", "Parameters", "Value"],
    ["Precision", most_precise['model'], most_precise['params'], most_precise['precis
    ["Recall", most_recall['model'], most_recall['params'], most_recall['recall']],
    ["Accuracy", most_accurate['model'], most_accurate['params'], most_accurate['accu
    ["F1", most_f1['model'], most_f1['params'], most_f1['f1']]
]

import tabulate
table = tabulate.tabulate(data, tablefmt='html')

from IPython.display import HTML, display
display(HTML(table))
```

Highest Attribute	Model Name	Parameters	Value
Precision	GaussianNB	{}	0.8075060532687651
Recall	LogisticRegression	{'max_iter': 1000, 'solver': 'newton-cg', 'C': 0.01}	0.856467409658899
Accuracy	LogisticRegression	{}	0.7718540902223181
F1	LogisticRegression	{}	0.8266360505166476

The table above shows all the most important scores to analyse our results. We used the article from (Vadakattu, 2021) to format our table for better displaying of the results.

First, lets define the scores and how we could interpret them in a 'soccer game' way.

The article from (Afonja, 2017) will help us understand better the way the different scores work. We will use the example from the article and adapt it to our problem/dataset.

Lets take an example where we have a dataset of 100 games. The games are categorized as a home team win or a home team loss.

We will give the following values:

- We have 40 home team wins.
- We have 60 home teams losses.

```
games = 100
wins = 40
losses = 60
```

After training our algorithm and testing it, we get the following results:

- 35 predicted wins
- 65 predicted losses

```
predicted_wins = 35
predicted_losses = 65
```

However, not all the predicted wins and predicted losses have the same origin.

Some wins were not detected, and some losses were not detected. We can break this down in 4 categories:

- 30/40 wins were detected --> We have 30 True Positives (TP)
- 10/40 wins were not detected --> We have 10 False Negatives (FN)
- 55/60 losses were detected --> We have 55 True Negatives (TN)
- 5/60 losses were not detected --> We have 5 False positives (FP)

```
TP = 30
FN = 10
TN = 55
FP = 5
```

We have the following definitions for all the important scores:

```
Accuracy = (TN + TP)/(TN + FP + TP + FN)

Precision = TP / TP + FP
```

```
Recall = TP / TP + FN

F1 = 2 * Precision * Recall / (Precision + Recall)
```

We can plug in our values from the example to get the desired scores.

```
Accuracy = (55 + 30) / (55 + 5 + 30 + 10) = 85%
Precision =  30 / (30 + 5) =  6/7 = 85.7%
Recall = 30 / (30 + 10) = 3/4 = 75%
F1 = 2 * 6/7 * 3/4 / (6/7 + 3/4) = 0.7999999999999999 ≈ 0.8
```

Now that we understand the measures and their meaning, we can apply it better to our example.

We now know that the most important measures to take into account are accuracy, because we want to predict the biggest amount of correct results, not caring about wether the result is a win or a loss.

Also, we want to use F1 because it is a combination of precision and recall, and it can help us to have an algorithm that offers high level of detail when trying to predict either a win or a loss.

Luckily, in our case, the best algorithm was the same for both measures, so there was no question about which one we should use.

In [ ]:
```
data = [
    ["Highest Attribute", "Model Name", "Parameters", "Value"],
    ["Accuracy", most_accurate['model'], most_accurate['params'], most_accurate['accu
    ["F1", most_f1['model'], most_f1['params'], most_f1['f1']]
]

table = tabulate.tabulate(data, tablefmt='html')

display(HTML(table))
```

Highest Attribute	Model Name	Parameters	Value
Accuracy	LogisticRegression	{}	0.7718540902223181
F1	LogisticRegression	{}	0.8266360505166476

The table above shows that the algorithm we will be using to predict the World Cup is the `LogisticRegression` with default parameters.

10. Predicting the 2022 World Cup winner

Having tested all the models, we will now decide to use only one model to predict all the games of the world cup.

The model we will use is LogisticRegression because using default parameters, it has the best accuracy and F1 score.

We will re train the model using 99% of our past data so that it has an even better percentage, and once it is trained we will use it to predict all the future games.

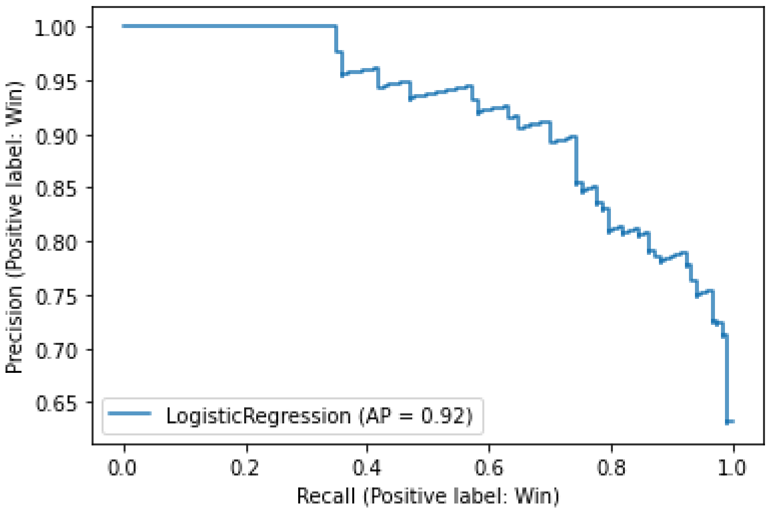
In [ ]:
```
# change original training dataset to use 99% of it's data for training, for a
# better final model accuracy
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,
```

```
                                    train_size = .99)
y_train = y_train.values.ravel()
```

In [ ]:
```
# Re Scale the data for the final model
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [ ]:
```
# Create an instance of the Naive Bayes model with default parameters
final_model = LogisticRegression()

# Train the model using the training sets
final_model.fit(X_train,y_train)

# Do a prediction on the test data
y_pred = final_model.predict(X_test)

# Ignore warnings for graph creation
warnings.filterwarnings('ignore')

# Evaluate the metrics
precision = sklearn.metrics.precision_score(y_test, y_pred, pos_label='Win')
recall = sklearn.metrics.recall_score(y_test, y_pred, pos_label='Win')
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
f1 = sklearn.metrics.f1_score(y_test, y_pred, pos_label = 'Win')

score = {
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1" : f1,
    "average" : mean([precision, recall, accuracy, f1]),
    "model": type(final_model).__name__,
    "params": final_model.get_params().items()
}

print(score)

disp = sklearn.metrics.plot_precision_recall_curve(final_model, X_test, y_test)
```

```
{'precision': 0.8015873015873016, 'recall': 0.8632478632478633, 'accuracy': 0.7795698
924731183, 'f1': 0.831275720164609, 'average': 0.818920194368223, 'model': 'LogisticR
egression', 'params': dict_items([('C', 1.0), ('class_weight', None), ('dual', Fals
e), ('fit_intercept', True), ('intercept_scaling', 1), ('l1_ratio', None), ('max_ite
r', 100), ('multi_class', 'auto'), ('n_jobs', None), ('penalty', 'l2'), ('random_stat
e', None), ('solver', 'lbfgs'), ('tol', 0.0001), ('verbose', 0), ('warm_start', Fals
e)])}
```



From the results above, we can see that our final model has the following scores:

```
Accuracy = 0.7795698924731183
F1 = 0.831275720164609
```

So, we have slight improvements over both scores by using more training data. We are now ready to use the model to do all of our predictions.

10.1. Predicting the Group Stages

In [ ]:
```
# we import the group stage file which contains all the games
gs_df_url = 'https://raw.githubusercontent.com/antonpp11/CSI4106-Project1-DB/master/g
gs_df = pd.read_csv(gs_df_url)
```

In [ ]:
```
# drop the features to remove, as we have the same file structure
gs_df = gs_df.drop(features_to_remove, axis=1)
```

In [ ]:
```
# apply the date modifications to our group stage dataframe
# change dates to datetime format
gs_df['date'] = pd.to_datetime(gs_df['date'])

# create timestamps by substracting dates using the first game of our dataset
gs_df['timedelta_int'] = ((gs_df['date'] - first_game_date).dt.total_seconds() + gs_d
gs_df = gs_df.drop('date', axis=1)
```

In [ ]:
```
# perform one hot encoding for home and away teams
gs_df = pd.get_dummies(
    data=gs_df,
    columns=["home_team", "away_team"],
    prefix=["h", "a"])
```

In [ ]:
```
# prepare group stage dataset by splitting for train and test

# add missing features (because some teams arent qualified to the world cup,
# but we still need their columns, albeit empty)
for missing_feature in list(set(all_teams_columns) - set(list(gs_df.columns))):
  gs_df[missing_feature] = 0
# reorder the columns using the original list
gs_df = gs_df.reindex(columns=all_teams_columns)

# add features for group stage
gs_features = list(gs_df.columns)
gs_features.remove('home_team_result')


# We don't need train, because we only want to predict, hence only testing
gs_X_test = gs_df.loc[:, gs_features]
gs_y_test = gs_df.loc[:, ['home_team_result']]
```

In [ ]:
```
# We use our scaler used for scaling our prediction data
gs_X_test = sc.transform(gs_X_test)
```

In [ ]:
```
# use the final model to predict all the future games
gs_y_pred = final_model.predict(gs_X_test)
gs_df.insert(len(gs_df.columns), "home_result_pred", list(gs_y_pred))
```

```
['Lose' 'Win' 'Lose' 'Win' 'Win' 'Win' 'Win' 'Win' 'Lose' 'Win' 'Win'
 'Win' 'Win' 'Win' 'Win' 'Win' 'Lose' 'Lose' 'Win' 'Win' 'Lose' 'Win'
 'Win' 'Win' 'Win' 'Win' 'Win' 'Win' 'Lose' 'Win' 'Win' 'Win' 'Win' 'Win'
 'Lose' 'Lose' 'Lose' 'Lose' 'Lose' 'Lose' 'Lose' 'Lose' 'Lose' 'Lose'
 'Lose' 'Lose' 'Lose' 'Lose']
```

In [ ]:
```
# export the groups stage df to csv
```

```
gs_df.to_csv('gs_predictions.csv')
```

We will now calculate the winners or every group using this tool from ("2022 FIFA World Cup Group Stage Points Simulator"). Every win will be 3-0, to simplify the calculations.

You can see the link to the complete result sheet here.

Here are the results of every group:

- Group A :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	Netherlands	9	3	3	0	0	9	0	9
QUALIFY	Ecuador	6	3	2	0	1	6	3	3
	Senegal	3	3	1	0	2	3	6	-3
	Qatar	0	3	0	0	3	0	9	-9

local time | your time zone | AUTO | CLEAR

1	2022-11-21 06:00	SENEGAL	0 : 3	NETHERLANDS
	2022-11-20 12:00	QATAR	0 : 3	ECUADOR
2	2022-11-25 09:00	QATAR	0 : 3	SENEGAL
	2022-11-25 12:00	NETHERLANDS	3 : 0	ECUADOR
3	2022-11-29 11:00	NETHERLANDS	3 : 0	QATAR
	2022-11-29 11:00	ECUADOR	3 : 0	SENEGAL

- Group B :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	England	9	3	3	0	0	9	0	9
QUALIFY	USA	6	3	2	0	1	6	3	3
	Iran	3	3	1	0	2	3	6	-3
	Wales	0	3	0	0	3	0	9	-9

local time | your time zone | AUTO | CLEAR

1	2022-11-21 09:00	ENGLAND	3 : 0	IRAN
	2022-11-21 15:00	USA	3 : 0	WALES
2	2022-11-25 06:00	WALES	0 : 3	IRAN
	2022-11-25 15:00	ENGLAND	3 : 0	USA
3	2022-11-29 15:00	WALES	0 : 3	ENGLAND
	2022-11-29 15:00	IRAN	0 : 3	USA

- Group C :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	Argentina	9	3	3	0	0	9	0	9
QUALIFY	Mexico	6	3	2	0	1	6	3	3
	Poland	3	3	1	0	2	3	6	-3
	Saudi Arabia	0	3	0	0	3	0	9	-9

local time your time zone AUTO CLEAR

1	2022-11-22 06:00	ARGENTINA	3 : 0	SAUDI ARABIA
	2022-11-22 12:00	MEXICO	3 : 0	POLAND
2	2022-11-26 09:00	POLAND	3 : 0	SAUDI ARABIA
	2022-11-26 15:00	ARGENTINA	3 : 0	MEXICO
3	2022-11-30 15:00	POLAND	0 : 3	ARGENTINA
	2022-11-30 15:00	SAUDI ARABIA	0 : 3	MEXICO

- Group D :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	France	9	3	3	0	0	9	0	9
QUALIFY	Denmark	6	3	2	0	1	6	3	3
	Australia	3	3	1	0	2	3	6	-3
	Tunisia	0	3	0	0	3	0	9	-9

local time your time zone AUTO CLEAR

1	2022-11-22 09:00	DENMARK	3 : 0	TUNISIA
	2022-11-22 15:00	FRANCE	3 : 0	AUSTRALIA
2	2022-11-26 06:00	TUNISIA	0 : 3	AUSTRALIA
	2022-11-26 12:00	FRANCE	3 : 0	DENMARK
3	2022-11-30 11:00	TUNISIA	0 : 3	FRANCE
	2022-11-30 11:00	AUSTRALIA	0 : 3	DENMARK

- Group E :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	Spain	9	3	3	0	0	9	0	9
QUALIFY	Germany	6	3	2	0	1	6	3	3
	Japan	3	3	1	0	2	3	6	-3
	Costa Rica	0	3	0	0	3	0	9	-9

local time your time zone AUTO CLEAR

1	2022-11-23 09:00	GERMANY	3 : 0	JAPAN
	2022-11-23 12:00	SPAIN	3 : 0	COSTA RICA
2	2022-11-27 06:00	JAPAN	3 : 0	COSTA RICA
	2022-11-27 15:00	SPAIN	3 : 0	GERMANY
3	2022-12-01 15:00	JAPAN	0 : 3	SPAIN
	2022-12-01 15:00	COSTA RICA	0 : 3	GERMANY

- Group F :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	Belgium	9	3	3	0	0	9	0	9
QUALIFY	Croatia	6	3	2	0	1	6	3	3
	Morocco	3	3	1	0	2	3	6	-3
	Canada	0	3	0	0	3	0	9	-9

local time your time zone AUTO CLEAR

1	2022-11-23 06:00	MOROCCO	0 : 3	CROATIA
	2022-11-23 15:00	BELGIUM	3 : 0	CANADA
2	2022-11-27 09:00	BELGIUM	3 : 0	MOROCCO
	2022-11-27 12:00	CROATIA	3 : 0	CANADA
3	2022-12-01 11:00	CROATIA	0 : 3	BELGIUM
	2022-12-01 11:00	CANADA	0 : 3	MOROCCO

- Group G :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	Brazil	9	3	3	0	0	9	0	9
QUALIFY	Switzerland	6	3	2	0	1	6	3	3
	Serbia	3	3	1	0	2	3	6	-3
	Cameroon	0	3	0	0	3	0	9	-9

local time your time zone AUTO CLEAR

1	2022-11-24 06:00	SWITZERLAND	3 : 0	CAMEROON
	2022-11-24 15:00	BRAZIL	3 : 0	SERBIA
2	2022-11-28 06:00	CAMEROON	0 : 3	SERBIA
	2022-11-28 12:00	BRAZIL	3 : 0	SWITZERLAND
3	2022-12-02 15:00	CAMEROON	0 : 3	BRAZIL
	2022-12-02 15:00	SERBIA	0 : 3	SWITZERLAND

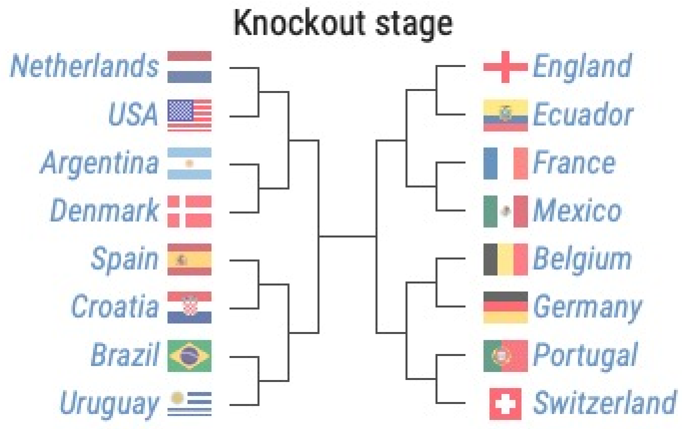
- Group H :

		Pts	MP	W	D	L	GF	GA	+/-
QUALIFY	Portugal	9	3	3	0	0	9	0	9
QUALIFY	Uruguay	6	3	2	0	1	6	3	3
	Korea Republic	3	3	1	0	2	3	6	-3
	Ghana	0	3	0	0	3	0	9	-9

local time your time zone AUTO CLEAR

1	2022-11-24 09:00	URUGUAY	3 : 0	KOREA REP.
	2022-11-24 12:00	PORTUGAL	3 : 0	GHANA
2	2022-11-28 09:00	KOREA REP.	3 : 0	GHANA
	2022-11-28 15:00	PORTUGAL	3 : 0	URUGUAY
3	2022-12-02 11:00	KOREA REP.	0 : 3	PORTUGAL
	2022-12-02 11:00	GHANA	0 : 3	URUGUAY

Here is what the knockout stages will look like:



10.2. Predicting the Round of 16

In [ ]:
```
# we import the file which contains all the future games
ks_df_url = 'https://raw.githubusercontent.com/antonpp11/CSI4106-Project1-DB/master/r
ks_df = pd.read_csv(ks_df_url)
```

In [ ]:
```
# drop the features to remove, as we have the same file structure
ks_df = ks_df.drop(features_to_remove, axis=1)
```

In [ ]:
```
# apply the date modifications to our dataframe
# change dates to datetime format
ks_df['date'] = pd.to_datetime(ks_df['date'])

# create timestamps by substracting dates using the first game of our dataset
ks_df['timedelta_int'] = ((ks_df['date'] - first_game_date).dt.total_seconds() + ks_d
ks_df = ks_df.drop('date', axis=1)
```

In [ ]:
```
# perform one hot encoding for home and away teams
ks_df = pd.get_dummies(
    data=ks_df,
    columns=["home_team", "away_team"],
    prefix=["h", "a"])
```

In [ ]:
```
# prepare the dataset by splitting for train and test

# add missing features (because some teams arent qualified to the world cup,
# but we still need their columns, albeit empty)
for missing_feature in list(set(all_teams_columns) - set(list(ks_df.columns))):
  ks_df[missing_feature] = 0
# reorder the columns using the original list
ks_df = ks_df.reindex(columns=all_teams_columns)

# add features for group stage
ks_features = list(ks_df.columns)
ks_features.remove('home_team_result')


# We don't need train, because we only want to predict, hence only testing
ks_X_test = ks_df.loc[:, ks_features]
ks_y_test = ks_df.loc[:, ['home_team_result']]
```

In [ ]:
```
# We use our scaler used for scaling our prediction data
ks_X_test = sc.transform(ks_X_test)
```

In [ ]:
```
# use the final model to predict all the future games
ks_y_pred = final_model.predict(ks_X_test)
```

```
ks_df.insert(len(ks_df.columns), "home_result_pred", list(ks_y_pred))
print(ks_y_pred)
```

```
['Win' 'Win' 'Win' 'Win' 'Win' 'Lose' 'Win' 'Win']
```

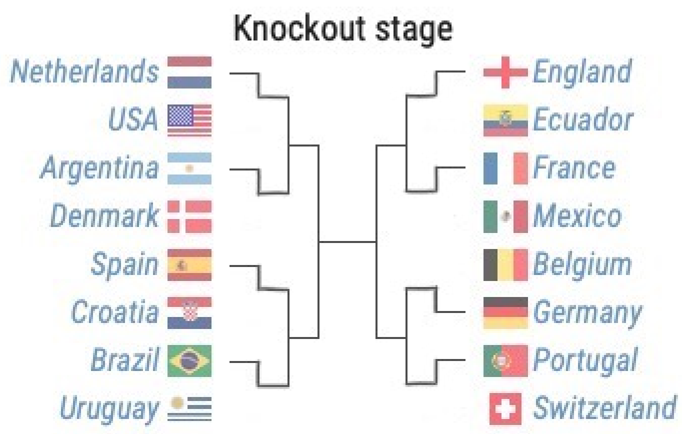
In [ ]:
```
# export the df to csv
ks_df.to_csv('ks_predictions.csv')
```

You can see the link to the complete result sheet here.

The following lines will resume the results of every game.

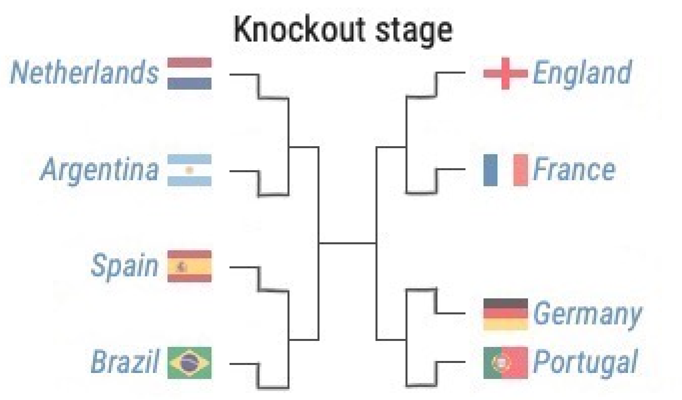
- Netherlands 3 : 0 USA
- England 3 : 0 Ecuador
- Argentina 3 : 0 Denmark
- France 3 : 0 Mexico
- Spain 3 : 0 Croatia
- Belgium 0 : 3 Germany
- Brazil 3 : 0 Uruguay
- Portugal 3 : 0 Switzerland

Here are the results of the Round of 16:



10.3. Predicting the Quarter Finals

Here are the games we want to predict:



In [ ]:
```
# we import the file which contains all the future games
qf_df_url = 'https://raw.githubusercontent.com/antonpp11/CSI4106-Project1-DB/master/q
qf_df = pd.read_csv(qf_df_url)
```

In [ ]:
```
# drop the features to remove, as we have the same file structure
```

```python
qf_df = qf_df.drop(features_to_remove, axis=1)
```

In [ ]:
```python
# apply the date modifications to our dataframe
# change dates to datetime format
qf_df['date'] = pd.to_datetime(qf_df['date'])

# create timestamps by substracting dates using the first game of our dataset
qf_df['timedelta_int'] = ((qf_df['date'] - first_game_date).dt.total_seconds() + qf_d
qf_df = qf_df.drop('date', axis=1)
```

In [ ]:
```python
# perform one hot encoding for home and away teams
qf_df = pd.get_dummies(
    data=qf_df,
    columns=["home_team", "away_team"],
    prefix=["h", "a"])
```

In [ ]:
```python
# prepare the dataset by splitting for train and test

# add missing features (because some teams arent qualified to the world cup,
# but we still need their columns, albeit empty)
for missing_feature in list(set(all_teams_columns) - set(list(qf_df.columns))):
  qf_df[missing_feature] = 0
# reorder the columns using the original list
qf_df = qf_df.reindex(columns=all_teams_columns)

# add features for group stage
qf_features = list(qf_df.columns)
qf_features.remove('home_team_result')


# We don't need train, because we only want to predict, hence only testing
qf_X_test = qf_df.loc[:, qf_features]
qf_y_test = qf_df.loc[:, ['home_team_result']]
```

In [ ]:
```python
# We use our scaler used for scaling our prediction data
qf_X_test = sc.transform(qf_X_test)
```

In [ ]:
```python
# use the final model to predict all the future games
qf_y_pred = final_model.predict(qf_X_test)
qf_df.insert(len(qf_df.columns), "home_result_pred", list(qf_y_pred))
print(qf_y_pred)
```

```
['Lose' 'Lose' 'Lose' 'Lose']
```

In [ ]:
```python
# export the df to csv
qf_df.to_csv('qf_predictions.csv')
```

You can see the link to the complete result sheet here.

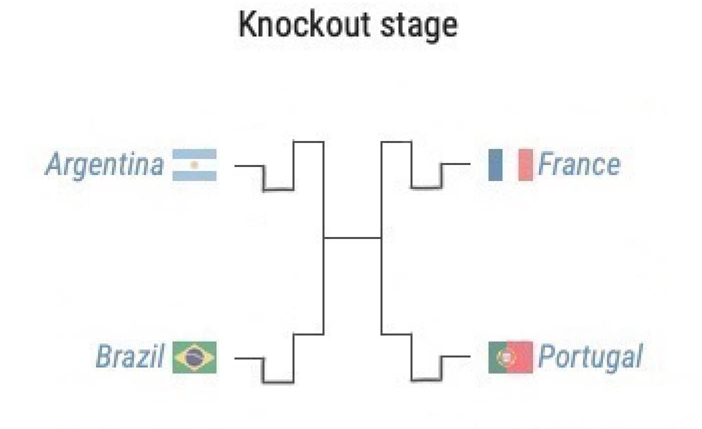
The following lines will resume the results of every game.

- Netherlands 0 : 3 Argentina
- England 0 : 3 France
- Spain 0 : 3 Brazil
- Germany 0 : 3 Portugal

Here are the results of the Quarter Finals:

10.4. Predicting the Semi Finals

Here are the games we want to predict:



In [ ]:
```
# we import the file which contains all the future games
sf_df_url = 'https://raw.githubusercontent.com/antonpp11/CSI4106-Project1-DB/master/s
sf_df = pd.read_csv(sf_df_url)
```

In [ ]:
```
# drop the features to remove, as we have the same file structure
sf_df = sf_df.drop(features_to_remove, axis=1)
```

In [ ]:
```
# apply the date modifications to our dataframe
# change dates to datetime format
sf_df['date'] = pd.to_datetime(sf_df['date'])

# create timestamps by substracting dates using the first game of our dataset
sf_df['timedelta_int'] = ((sf_df['date'] - first_game_date).dt.total_seconds() + sf_d
sf_df = sf_df.drop('date', axis=1)
```

In [ ]:
```
# perform one hot encoding for home and away teams
sf_df = pd.get_dummies(
    data=sf_df,
    columns=["home_team", "away_team"],
    prefix=["h", "a"])
```

In [ ]:
```
# prepare the dataset by splitting for train and test

# add missing features (because some teams arent qualified to the world cup,
# but we still need their columns, albeit empty)
for missing_feature in list(set(all_teams_columns) - set(list(sf_df.columns))):
  sf_df[missing_feature] = 0
# reorder the columns using the original list
sf_df = sf_df.reindex(columns=all_teams_columns)
```

```python
# add features for group stage
sf_features = list(sf_df.columns)
sf_features.remove('home_team_result')


# We don't need train, because we only want to predict, hence only testing
sf_X_test = sf_df.loc[:, sf_features]
sf_y_test = sf_df.loc[:, ['home_team_result']]
```

In [ ]:
```python
# We use our scaler used for scaling our prediction data
sf_X_test = sc.transform(sf_X_test)
```

In [ ]:
```python
# use the final model to predict all the future games
sf_y_pred = final_model.predict(sf_X_test)
sf_df.insert(len(sf_df.columns), "home_result_pred", list(sf_y_pred))
print(sf_y_pred)
```

```
['Lose' 'Win']
```

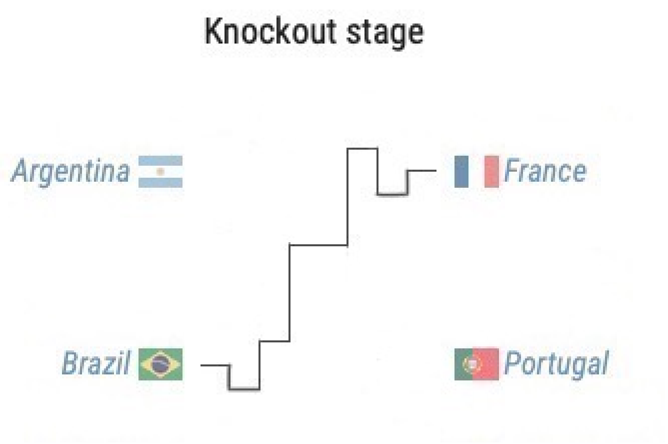
In [ ]:
```python
# export the df to csv
sf_df.to_csv('sf_predictions.csv')
```

You can see the link to the complete result sheet here.

The following lines will resume the results of every game.

- Argentina 0 : 3 Brazil
- France 3 : 0 Portugal

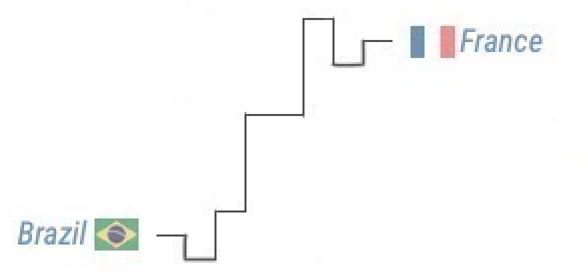
Here are the results of the Semi Finals:



10.5. Predicting the Finals

Here are the games we want to predict:

Knockout stage




```
In [ ]: # we import the file which contains all the future games
        f_df_url = 'https://raw.githubusercontent.com/antonpp11/CSI4106-Project1-DB/master/fi
        f_df = pd.read_csv(f_df_url)
```

```
In [ ]: # drop the features to remove, as we have the same file structure
        f_df = f_df.drop(features_to_remove, axis=1)
```

```
In [ ]: # apply the date modifications to our dataframe
        # change dates to datetime format
        f_df['date'] = pd.to_datetime(f_df['date'])

        # create timestamps by substracting dates using the first game of our dataset
        f_df['timedelta_int'] = ((f_df['date'] - first_game_date).dt.total_seconds() + f_df.i
        f_df = f_df.drop('date', axis=1)
```

```
In [ ]: # perform one hot encoding for home and away teams
        f_df = pd.get_dummies(
            data=f_df,
            columns=["home_team", "away_team"],
            prefix=["h", "a"])
```

```
In [ ]: # prepare the dataset by splitting for train and test

        # add missing features (because some teams arent qualified to the world cup,
        # but we still need their columns, albeit empty)
        for missing_feature in list(set(all_teams_columns) - set(list(f_df.columns))):
          f_df[missing_feature] = 0
        # reorder the columns using the original list
        f_df = f_df.reindex(columns=all_teams_columns)

        # add features for group stage
        f_features = list(f_df.columns)
        f_features.remove('home_team_result')


        # We don't need train, because we only want to predict, hence only testing
        f_X_test = f_df.loc[:, f_features]
        f_y_test = f_df.loc[:, ['home_team_result']]
```

```
In [ ]: # We use our scaler used for scaling our prediction data
        f_X_test = sc.transform(f_X_test)
```

```
In [ ]: # use the final model to predict all the future games
        f_y_pred = final_model.predict(f_X_test)
        f_df.insert(len(f_df.columns), "home_result_pred", list(f_y_pred))
        print(f_y_pred)

        ['Lose']
```

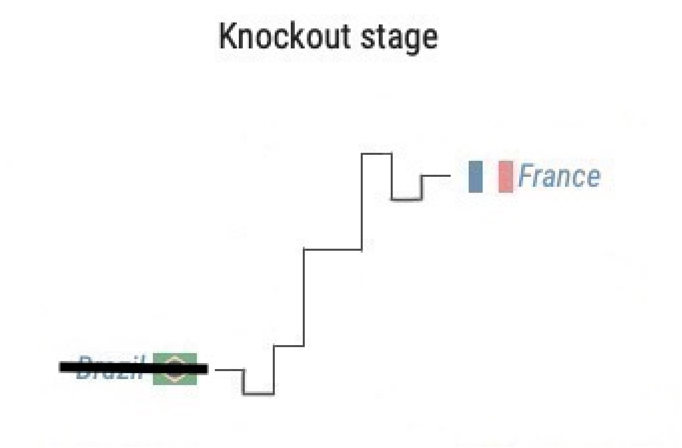
```
In [ ]: # export the df to csv
        f_df.to_csv('f_predictions.csv')
```

You can see the link to the complete result sheet here.

The following lines will resume the results of every game.

- Brazil 0 : 3 France

Here are the results of the Final:



11. The 2022 World Cup Winner

12. Conclusion

In conclusion, this project helped us define, analyse and differentiate 3 different type of classification algorithms: Naïve Bayes, Logisitc Regression and Multi-Layer Perceptron. We also studied the effects of the parameters for each algorithm by changing them and comparing the results. We noticed that a single dimensional algorithm doesn't require as much processing power and ressources as a multi-layer algorithm. However, there is a downside to it: it is less accurate.

To top it off, we also predicted the winner of the men's 2022 FIFA World Cup that will be happening in Qatar this November. Hopefully, our most effective algorithm got it right.

13. Bibliography

- “2022 FIFA World Cup Group Stage Points Simulator.” ULTRAZONE, ultra.zone/2022-FIFA-World-Cup-Group-Stage. Accessed 29 Oct. 2022.

- Afonja, Tejumade. “Model Evaluation I: Precision and Recall.” Medium, 11 Dec. 2017, towardsdatascience.com/model-evaluation-i-precision-and-recall-166ddb257c7b. Accessed 29 Oct. 2022.

- Hegde, Sushmitha. "Is there An Actual Home Field Advantage When A Sports Team Plays In Their Home Stadium?" Science ABC, 22 January 2022, https://www.scienceabc.com/social-science/is-there-an-actual-home-field-advantage-when-a-sports-team-plays-in-their-home-stadium.html. Accessed 29 Oct. 2022.

- Huilgol, Purva. “Precision vs Recall | Precision and Recall Machine Learning.” Analytics Vidhya, 3 Sept. 2020, www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/. Accessed 29 Oct. 2022.

- Loznik, Brenda. “FIFA World Cup 2022.” Www.kaggle.com, 2022, www.kaggle.com/datasets/brenda89/fifa-world-cup-2022. Accessed 29 Oct. 2022.

- Panjeh. “Scikit Learn Hyperparameter Optimization for MLPClassifier.” Medium, 29 June 2020, panjeh.medium.com/scikit-learn-hyperparameter-optimization-for-mlpclassifier-4d670413042b. Accessed 29 Oct. 2022.

- Pellerin-Petrov, Anton, and Alexander Onofrei. “CSI4106-Project1-DB.” GitHub, 13 Oct. 2022, github.com/antonpp11/CSI4106-Project1-DB. Accessed 30 Oct. 2022.

- Sagir, Umut. “Naive Bayes Classifier Optimization & Parameters.” HolyPython.com, 2019, holypython.com/nbc/naive-bayes-classifier-optimization-parameters/. Accessed 29 Oct. 2022.

- scikit-learn developers. “3.3. Metrics and Scoring: Quantifying the Quality of Predictions — Scikit-Learn 0.22.1 Documentation.” Scikit-Learn.org, 2010, scikit-learn.org/stable/modules/model_evaluation.html. Accessed 29 Oct. 2022.

- scikit-learn developers. “Sklearn.linear_model.LogisticRegression — Scikit-Learn 0.21.2 Documentation.” Scikit-Learn.org, 2014, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed 29 Oct. 2022.

- scikit-learn developers. “Sklearn.naive_bayes.GaussianNB — Scikit-Learn 0.22.1 Documentation.” Scikit-Learn.org, 2011, scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. Accessed 29 Oct. 2022.

- scikit-learn developers. “Sklearn.neural_network.MLPClassifier — Scikit-Learn 0.20.3 Documentation.” Scikit-Learn.org, 2010, scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. Accessed 29 Oct. 2022.

- Stojiljković, Mirko. “Logistic Regression in Python – Real Python.” Realpython.com, 2019, realpython.com/logistic-regression-python/. Accessed 29 Oct. 2022.

- Vadakattu, Aadarsh. “Pretty Displaying Tricks for Columnar Data in Python.” Medium, 30 May 2021, towardsdatascience.com/pretty-displaying-tricks-for-columnar-data-in-python-2fe3b3ed9b83. Accessed 29 Oct. 2022.