

REPORT FOR FINALS_FIFA22 PREDICTION WITH MACHINE LEARNING

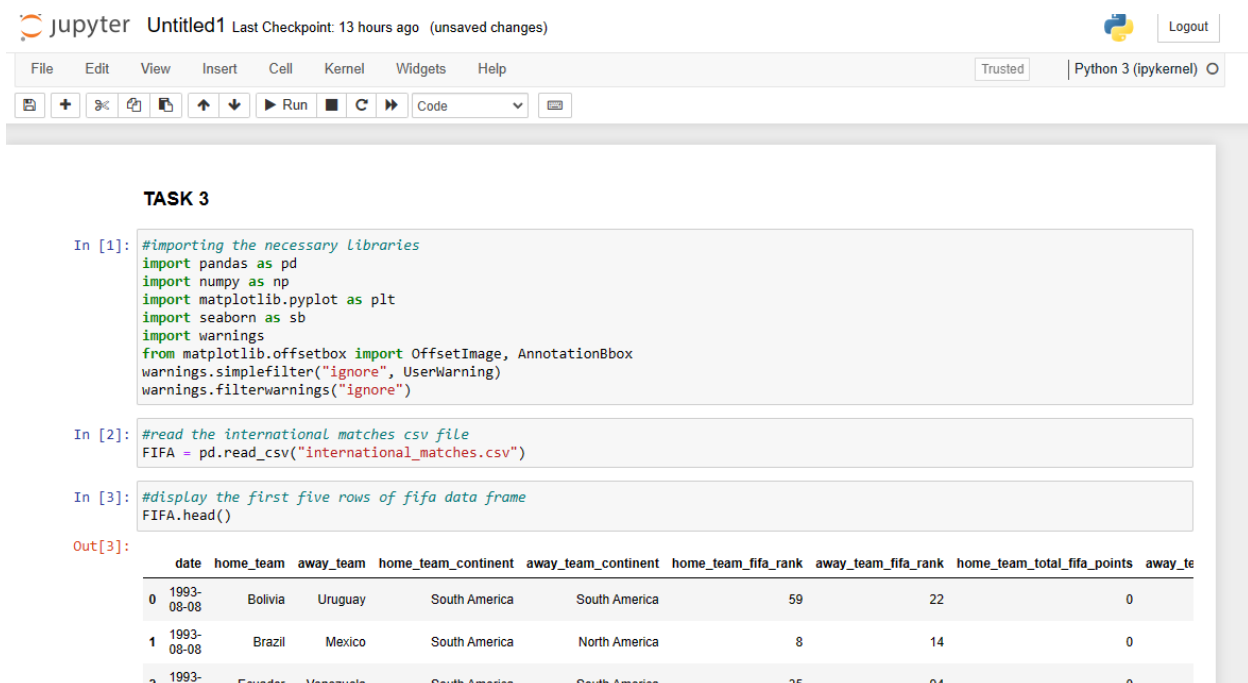
Babatunde Racheal

5/03/2023

DATA PRE-PROCESSING

Data pre-processing is a data mining technique that involves transforming raw data into a clean, consistent, and usable format. The main goal of data pre-processing is to ensure that the data is ready for analysis and modelling by reducing noise, inconsistencies, and other types of errors that might affect the performance of the machine learning algorithms.

Cleaning, normalizing, transforming, and extracting features from data are all part of the pre-processing phase. Data cleaning entails discarding unnecessary information, filling in missing values, and handling outliers. Normalization scales the data to a common range to enhance the efficiency of machine learning algorithms. Data transformation is the process of changing one data format into another for the purpose of analysis. Selecting the most pertinent variables from the data that can be used to construct a model is what feature extraction is all about.



The image shows a Jupyter Notebook interface with the following content:

TASK 3

```
In [1]: #importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import warnings
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
warnings.simplefilter("ignore", UserWarning)
warnings.filterwarnings("ignore")

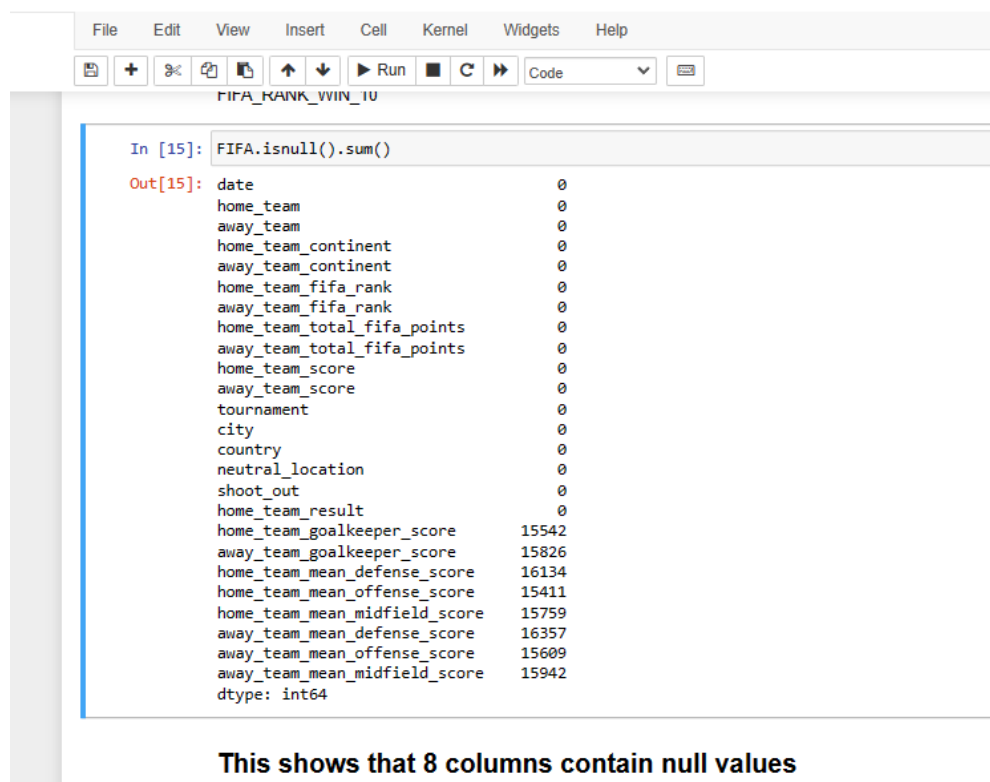
In [2]: #read the international matches csv file
FIFA = pd.read_csv("international_matches.csv")

In [3]: #display the first five rows of fifa data frame
FIFA.head()
```

Out[3]:

	date	home_team	away_team	home_team_continent	away_team_continent	home_team_fifa_rank	away_team_fifa_rank	home_team_total_fifa_points	away_te
0	1993-08-08	Bolivia	Uruguay	South America	South America	59	22	0	
1	1993-08-08	Brazil	Mexico	South America	North America	8	14	0	
2	1993-08-08	Ecuador	Venezuela	South America	South America	35	94	0	

NECESSARY PRE-PROCESSING STEPS



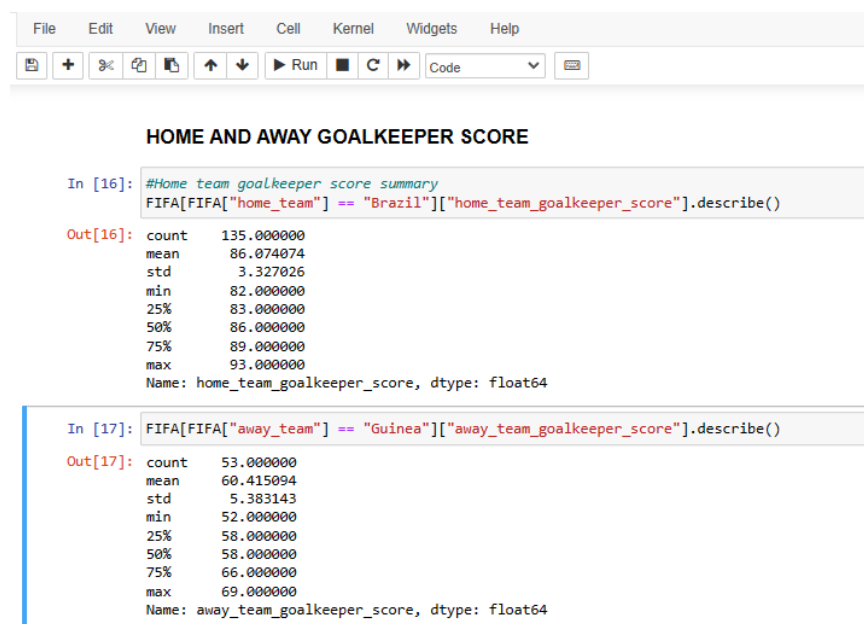
The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook title is "FIFA_RANK_WIN_TU". The code cell contains the command `FIFA.isnull().sum()`. The output shows a list of 20 columns with their respective null counts. Eight columns have a count of 0, while the other 12 have non-zero counts.

```
In [15]: FIFA.isnull().sum()

Out[15]: date                                0
home_team                                    0
away_team                                    0
home_team_continent                         0
away_team_continent                        0
home_team_fifa_rank                         0
away_team_fifa_rank                         0
home_team_total_fifa_points                 0
away_team_total_fifa_points                0
home_team_score                             0
away_team_score                             0
tournament                                  0
city                                         0
country                                     0
neutral_location                           0
shoot_out                                   0
home_team_result                           0
home_team_goalkeeper_score                 15542
away_team_goalkeeper_score                 15826
home_team_mean_defense_score               16134
home_team_mean_offense_score               15411
home_team_mean_midfield_score              15759
away_team_mean_defense_score               16357
away_team_mean_offense_score               15609
away_team_mean_midfield_score              15942
dtype: int64
```

This shows that 8 columns contain null values

pandas describe() function is used on selected column to determine if the data contains outliers by generating summary statistics



The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar. The notebook title is "HOME AND AWAY GOALKEEPER SCORE". The code cell contains two commands: `FIFA[FIFA["home_team"] == "Brazil"]["home_team_goalkeeper_score"].describe()` and `FIFA[FIFA["away_team"] == "Guinea"]["away_team_goalkeeper_score"].describe()`. The output shows summary statistics for the home team goalkeepers of Brazil and the away team goalkeepers of Guinea.

```
HOME AND AWAY GOALKEEPER SCORE

In [16]: #Home team goalkeeper score summary
FIFA[FIFA["home_team"] == "Brazil"]["home_team_goalkeeper_score"].describe()

Out[16]: count    135.000000
mean         86.074074
std           3.327026
min           82.000000
25%           83.000000
50%           86.000000
75%           89.000000
max           93.000000
Name: home_team_goalkeeper_score, dtype: float64

In [17]: FIFA[FIFA["away_team"] == "Guinea"]["away_team_goalkeeper_score"].describe()

Out[17]: count     53.000000
mean         60.415094
std           5.383143
min           52.000000
25%           58.000000
50%           58.000000
75%           66.000000
max           69.000000
Name: away_team_goalkeeper_score, dtype: float64
```

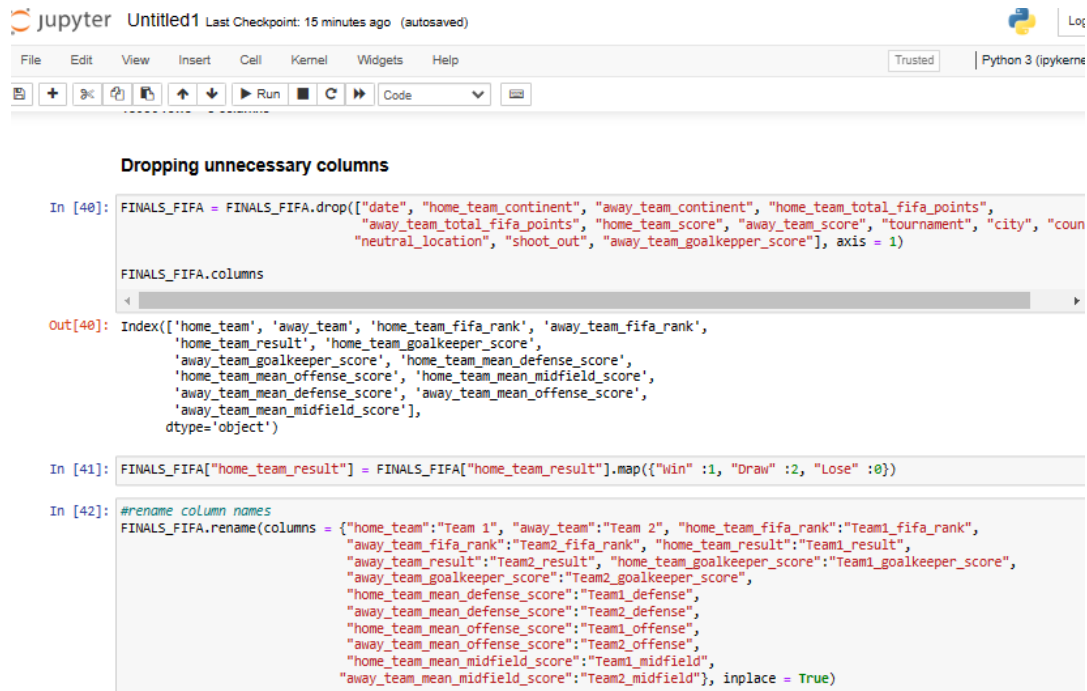
Data transformation

This refers to the process of converting raw data into a more suitable format for analysis. This involves various techniques to prepare and clean data, such as removing duplicates, dealing with missing values, and converting data types. Data transformation is an essential step in data pre-processing, as it can significantly affect the results of the analysis.

Some common techniques for data transformation include:

1. **Data cleaning:** Removing duplicates, dealing with missing values, correcting inconsistent values, and handling outliers.
2. **Data normalization:** Scaling the data to a common range to avoid bias due to differences in data magnitudes.
3. **Data aggregation:** Combining multiple data points to create a summary or composite data point.
4. **Data discretization:** Dividing continuous data into discrete intervals or categories to simplify analysis.
5. **Feature extraction:** Reducing the number of features in the dataset by selecting the most relevant or significant ones.

By applying appropriate data transformation techniques, data can be made more meaningful, interpretable, and useful for analysis.



The image shows a Jupyter Notebook interface with the following content:

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [40]: FINALS_FIFA = FINALS_FIFA.drop(["date", "home_team_continent", "away_team_continent", "home_team_total_fifa_points",
    "away_team_total_fifa_points", "home_team_score", "away_team_score", "tournament", "city", "country",
    "neutral_location", "shoot_out", "away_team_goalkeeper_score"], axis = 1)

FINALS_FIFA.columns

Out[40]: Index(['home_team', 'away_team', 'home_team_fifa_rank', 'away_team_fifa_rank',
    'home_team_result', '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'],
    dtype='object')

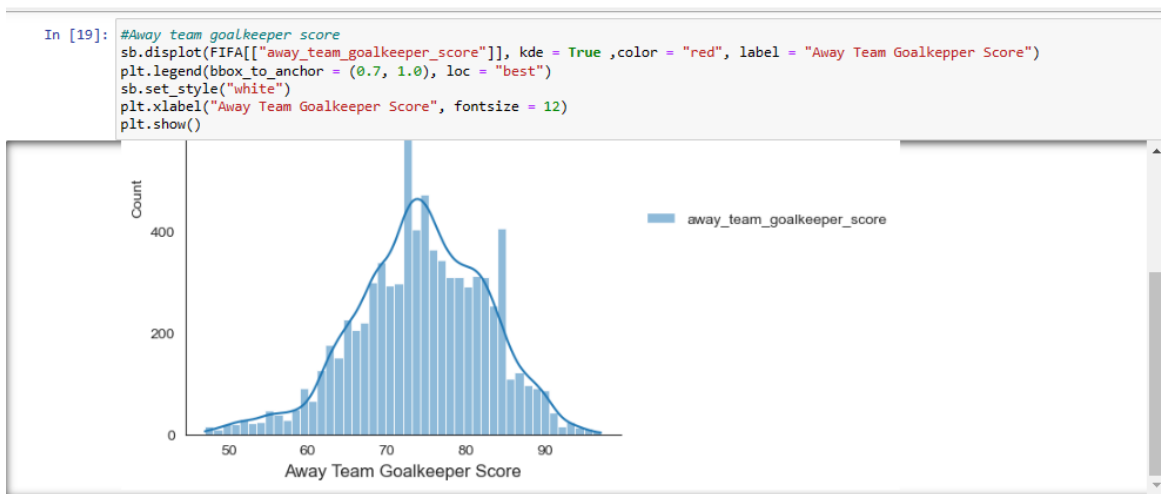
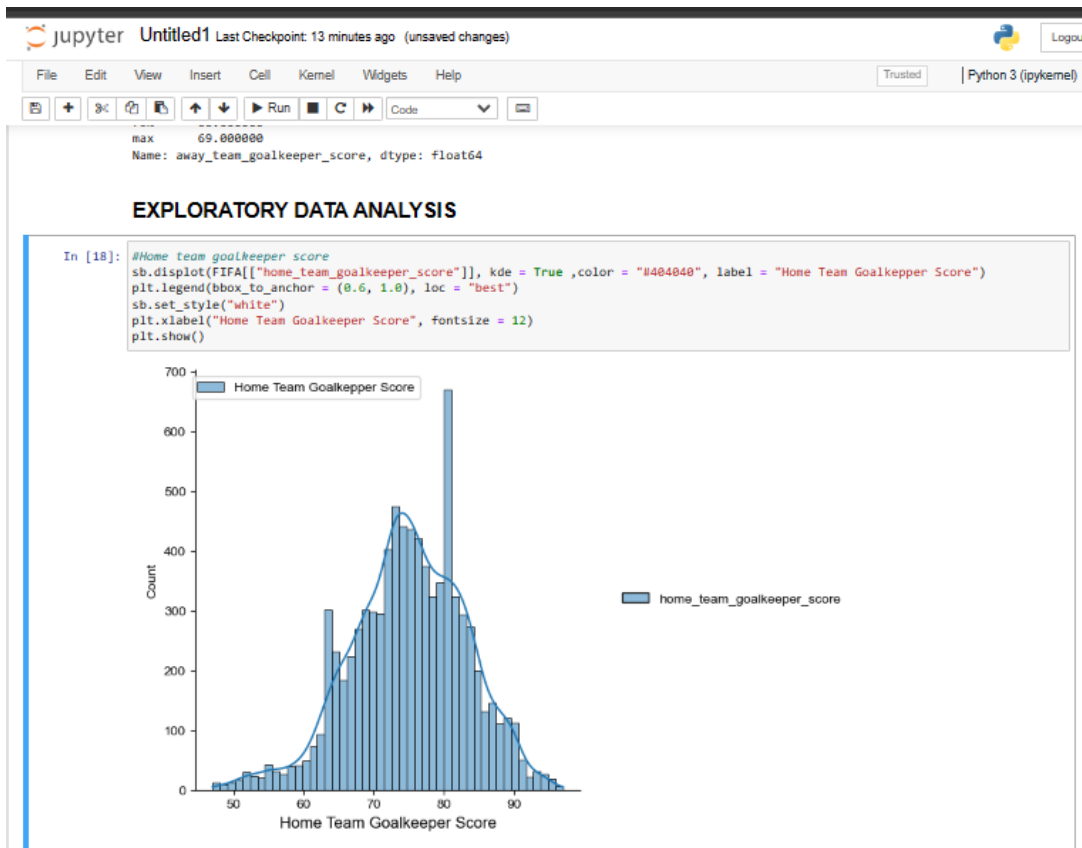
In [41]: FINALS_FIFA["home_team_result"] = FINALS_FIFA["home_team_result"].map({"win" :1, "Draw" :2, "Lose" :0})

In [42]: #rename column names
FINALS_FIFA.rename(columns = {"home_team":"Team 1", "away_team":"Team 2", "home_team_fifa_rank":"Team1_fifa_rank",
    "away_team_fifa_rank":"Team2_fifa_rank", "home_team_result":"Team1_result",
    "away_team_result":"Team2_result", "home_team_goalkeeper_score":"Team1_goalkeeper_score",
    "away_team_goalkeeper_score":"Team2_goalkeeper_score",
    "home_team_mean_defense_score":"Team1_defense",
    "away_team_mean_defense_score":"Team2_defense",
    "home_team_mean_offense_score":"Team1_offense",
    "away_team_mean_offense_score":"Team2_offense",
    "home_team_mean_midfield_score":"Team1_midfield",
    "away_team_mean_midfield_score":"Team2_midfield"}, inplace = True)
```

As shown in the picture above, we got rid of the columns that were not very important. The most important thing, no matter the city or country, is to know if the place is neutral or not. We only want to know about the first round of the World Cup, so the "shootout" doesn't matter.

We also renamed vital columns to provide more meaningful names to the columns and to make the column names more consistent with the naming conventions used in the field. This makes it easier to understand the contents of the dataset, especially when working with large datasets with many columns.

This shows that the difference between max and min values is little, Hence missing values can be replaced with the mean value



```
In [20]: #replace na's with mean in goal keeper score
FIFA["home_team_goalkeeper_score"] = round(FIFA.groupby("home_team")["home_team_goalkeeper_score"]
                                             .transform(lambda x: x.fillna(x.mean()))
                                             .round())

FIFA["away_team_goalkeeper_score"] = round(FIFA.groupby("away_team")["away_team_goalkeeper_score"]
                                             .transform(lambda x: x.fillna(x.mean()))
                                             .round())
```

EXTRACTING IMPORTANT FEATURES OF THE DATASET

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [7]: Portugal 2022-06-12 8
In [8]: Mexico 2022-06-14 9
In [9]: Netherlands 2022-06-14 10

In [11]: def get_flag(name):
          path = "C:\\Users\\Rhoda\\flags.jpg".format(name.title())
          im = plt.imread(path)
          return im

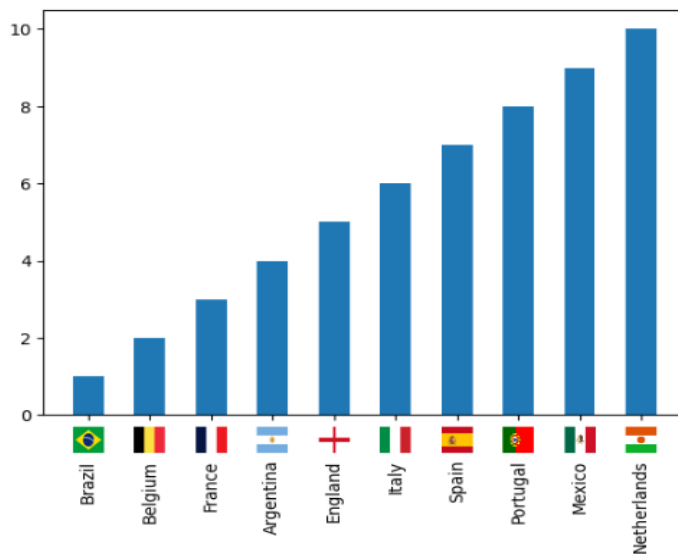
In [12]: def offset_image(coord, name, ax):
          img = get_flag(name)
          im = OffsetImage(img, zoom = 0.05)
          im.image.axes = ax
          ab = AnnotationBbox(im, (coord, 0), xybox = (0., -16.), frameon = False, xycoords = "data",
                              boxcoords = "offset points", pad = 0)
          ax.add_artist(ab)

In [ ]: Countries = FIFA_RANK_10["TEAM"]
          values = FIFA_RANK_10["RANK"]

          fig, ax = plt.subplots()
          ax.bar(range(len(Countries)), values, width = 0.5, align = "center")
          ax.set_xticks(range(len(Countries)))
          ax.set_xticklabels(Countries)
          ax.tick_params(axis = "x", which = "major", pad = 26)

          for i, c in enumerate(Countries):
              offset_image(i, c, ax)
          plt.xticks(rotation = 90, fontsize = 13)
          plt.show
```

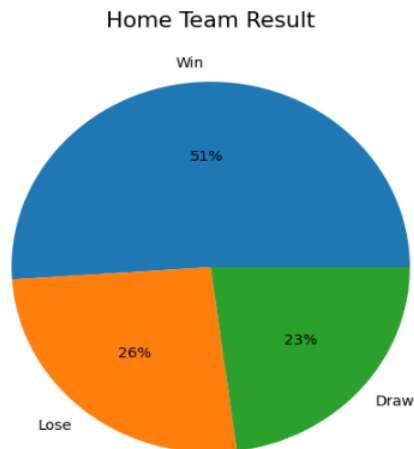
TOP TEN TEAMS AND RANKS



ADVANTAGES OF HOME TEAM OVER AWAY TEAM

```
In [17]: home_team_advantage = original_data[original_data['neutral_location']!=False]['home_team_result'].value_counts(normalize=True)
home_team_advantage_df = pd.DataFrame(columns=["Result", "Percentage"])
home_team_advantage_df["Result"] = ["Win", "Lose", "Draw"]
home_team_advantage_df["Percentage"] = home_team_advantage.values

plt.figure(figsize=(6, 6))
plt.pie(home_team_advantage, labels=home_team_advantage_df["Result"], autopct="%.0f%%");
plt.title("Home Team Result", fontsize=15)
plt.show()
```



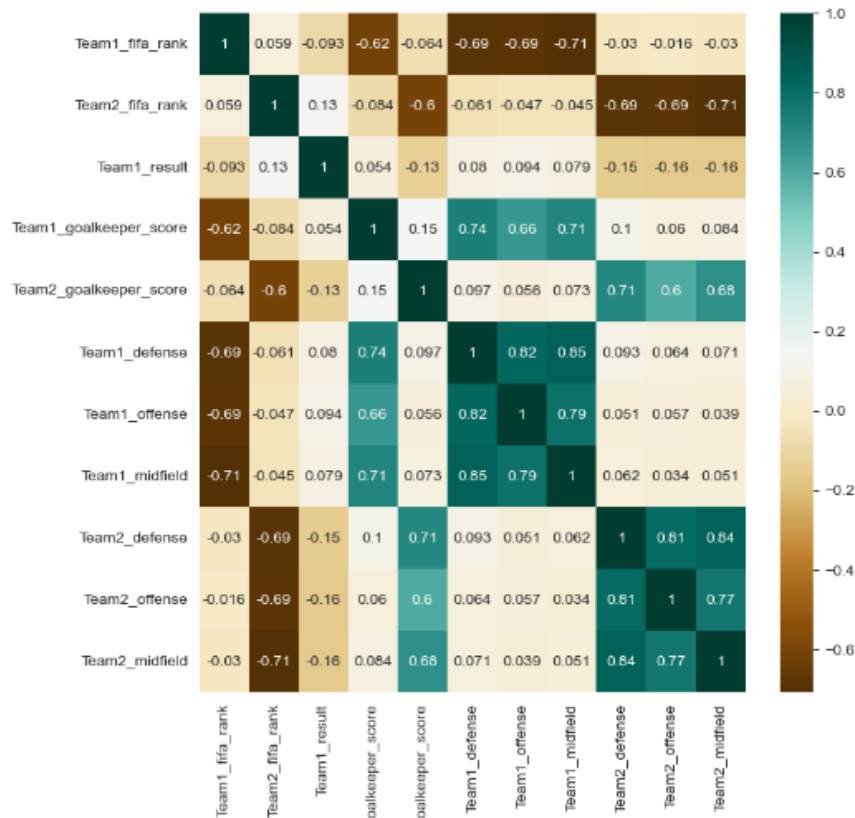
The pie chart shows that the home team is more likely to win than the team that is playing away. So, teams that host matches during the World Cup are likely to have a better chance of winning.

The heat map displays the correlation between the variables of the FIFA finals data frame. The legend displays the degree of correlation between the variables. The major purpose of this heat map is to visually represent the distribution and relationships between variables in a dataset using colours. Heat maps are particularly useful for exploring large datasets with many variables as they allow us to quickly identify patterns and correlations.

Overall, the purpose of a heat map is to provide an intuitive, visual representation of complex data that can be easily understood and interpreted by decision-makers.

```
In [44]: plt.figure(figsize = (8, 8), dpi = 100)
sb.heatmap(FINALS_FIFA.cor(), annot = True, cmap = "BrBG")

Out[44]: <Axes: >
```



DATA INGESTION

This is the process of obtaining and importing data from various sources into a data storage system or software application for further processing and analysis. It is a crucial step in the data management and analysis pipeline, as it lays the foundation for all subsequent tasks.

The goal of data ingestion is to ensure that data is efficiently and accurately collected, organized, and made easily available.

Data ingestion can involve several steps, such as data extraction, data transformation, and data loading. The data extraction process involves retrieving data from a source system or application. The data transformation process involves cleaning, filtering, and structuring the data to make it suitable for analysis.


```
In [45]: FINALS_FIFA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3248 entries, 8264 to 23918
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Team 1                 3248 non-null   object
1   Team 2                 3248 non-null   object
2   Team1_fifa_rank        3248 non-null   int64
3   Team2_fifa_rank        3248 non-null   int64
4   Team1_result           3248 non-null   int64
5   Team1_goalkeeper_score 3248 non-null   float64
6   Team2_goalkeeper_score 3248 non-null   float64
7   Team1_defense           3248 non-null   float64
8   Team1_offense           3248 non-null   float64
9   Team1_midfield          3248 non-null   float64
10  Team2_defense           3248 non-null   float64
11  Team2_offense           3248 non-null   float64
12  Team2_midfield          3248 non-null   float64
dtypes: float64(8), int64(3), object(2)
memory usage: 355.2+ KB
```

```
In [46]: FINALS_FIFA.to_csv("Data.csv")
```

```
In [47]: FINALS_FIFA = pd.read_csv("Data.csv")
FINALS_FIFA.tail()
```

Out[47]:

	Unnamed: 0	Team 1	Team 2	Team1_fifa_rank	Team2_fifa_rank	Team1_result	Team1_goalkeeper_score	Team2_goalkeeper_score	Team1_defense	Team2_defense
3243	23908	England	Hungary	5	40	0	83.0	85.0	85.0	85.0
3244	23907	Germany	Italy	12	6	1	90.0	89.0	84.0	84.0
3245	23908	Netherlands	Wales	10	18	1	81.0	74.0	85.2	85.2
3246	23909	Poland	Belgium	26	2	0	87.0	89.0	75.2	75.2
3247	23918	Chile	Ghana	28	60	0	79.0	74.0	75.5	75.5

```
In [48]: FINALS_FIFA = FINALS_FIFA.drop("Unnamed: 0", axis = 1)
```

```
In [49]: #data frame for pipeline
PIPELINE_FIFA = FINALS_FIFA
```

Specific reasons for the choice of the pre-processing methods

Data wrangling: Python was chosen for this task since it is an efficient programming language. It's also flexible enough to deal with data sets of any size. Hadoop, SQL databases, and Spark are just some of the data tools and technologies that can integrate with it thanks to its interoperability and adaptability. Numerous libraries are available there as well.

Eliminating unnecessary columns helped simplify the analysis by clearing the way for a clearer focus on key relationships in the data. This was done to increase performance by using less memory while storing the dataset. It takes more time to process a dataset with more columns, thus removing unnecessary columns can speed things up. In addition removal of some columns may become necessary if such columns contain sensitive information so as not to risk data breach

Data Ingestion: The goal of data ingestion is the gathering of information from different places and mediums such as databases and files. In addition, pandas and Numpy can be used to do data transformations. This can also facilitate data validation, which is essential for ensuring data quality before data analysis and the system's distributed learning mechanism makes it simple to manage massive datasets by facilitating data integration because users can easily manipulate and analyze data, analysis is also possible.

Data Transformation: When the amount of data available is too great, data transformation enables its decrease. In addition to this, it makes it possible to translate data into something that can be visualized. It also enables feature engineering, which consists of the creation of new features by changing current ones and their corresponding data.

TASK 4

MODEL SELECTION AND TRAINING

Here are the basic steps to use the SageMaker Python SDK to train and deploy a machine learning model:

1. Prepare your training data: SageMaker Python SDK can read data from a local file system. You need to load your data into a Pandas data frame or Numpy array.
2. Define your model: SageMaker Python SDK supports popular machine learning frameworks which include Scikit-learn. We will need to define your model using the appropriate framework.
3. Configure your training job: You need to specify the training job parameters, such as the number and type of instances, the input and output channels, and the hyperparameters.
4. Train your model: SageMaker Python SDK can start a training job on a GPU or CPU
5. Evaluate your model: After training your model, you need to evaluate its performance on a validation data set.

6. Deploy your model: Once you are satisfied with your model performance, you can deploy it to an Amazon SageMaker endpoint. SageMaker Python SDK provides an easy way to deploy a model on a scalable and fully managed infrastructure

A mixed set of attributes, including both numerical and category values is contained in the dataset. In order to make efficient use of those categorical values in the programming, dummy variables that can take the values 0 and 1 have been constructed. Categorical characteristics In order to achieve the highest level of precision, both Team 1 and Team 2 were modelled with dummy variables, one for each category.

TASK 4 MODEL SELECTION AND TRAINING

MACHINE LEARNING MODEL

LEAGUE MODEL

```
In [50]: #create dummies for categorical columns to derive the best model accuracy
FINALFIFA = pd.get_dummies(FINALFIFA)
```

TRAIN AND TEST SPLIT OF DATA

```
In [51]: x = FINALFIFA.drop("Team1_result", axis = 1)
y = FINALFIFA["Team1_result"]
```

```
In [52]: from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.1, random_state = 42)

#validation and test set
x_hold_test, x_test, y_hold_test, y_test = train_test_split(x_val, y_val, test_size = 0.5, random_state = 42)
```

```
In [53]: #Normalizing
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_hold_test = scaler.transform(x_hold_test)
```

NORMALIZING

When the features are nearly the same size and somewhat close to the normal distribution, the machine learning algorithm functions at its optimal level. Because the data follow a normal distribution, the StandardScaler() function is the one to use in this situation. A feature is standardized via the standardscaler function, which does so by first subtracting the mean and then scaling the feature to unit variance. The standard deviation is a statistical measure that may be used to calculate the unit variance of all of the values. The result follows a normal distribution with a standard deviation equal to one.

When it comes to selecting a model between decision tree, random forest, XGBoost, and AdaBoost classifiers, there are several factors to consider. Here are some key points to keep in mind:

1. In terms of their performance when applied to the situation at hand, many models can either perform better or worse depending on the particulars of the issue that need to be resolved. It is recommended that you train each of the models on your data and then evaluate their overall performance using the metrics that are most applicable to determine which model performs the best.
2. In general, more straightforward models are easier to understand and train, but more complicated models have the potential to produce better outcomes. If we are working with a problem that needs you to work under time limitations or if we have limited data to work with, it is possible that a simpler model, such as a decision tree or AdaBoost, will be the best option. On the other hand, if there is access to a significant amount of data and enough time to train a more complex model, you should give some consideration to employing random forest or XGBoost. These are two machine learning techniques.
3. It is much simpler to perceive and comprehend certain models, such as decision trees and AdaBoost, in comparison to other models. These models are possibly a better option to go with if you need to be able to explain how your model is making its predictions.
4. Ensemble methods: Random forest, XGBoost, and AdaBoost are all examples of ensemble methods. These methods integrate the results of numerous models to get more accurate predictions. If a significant amount of data is available and are searching for the highest potential level of performance, one of these models might be the best option.
5. Training time: In general, decision tree and AdaBoost are faster to train than random forest or XGBoost, but this will depend on the specifics of the problem.

Ultimately, the model that will work best for the problem at hand will be determined by a wide range of factors, such as the quantity and quality of the data available, the level of difficulty in the problem, and the particular objectives established for the model. It's a smart move to train more than one model at the same time and compare how well each one does its job so we can pick the one that works best.

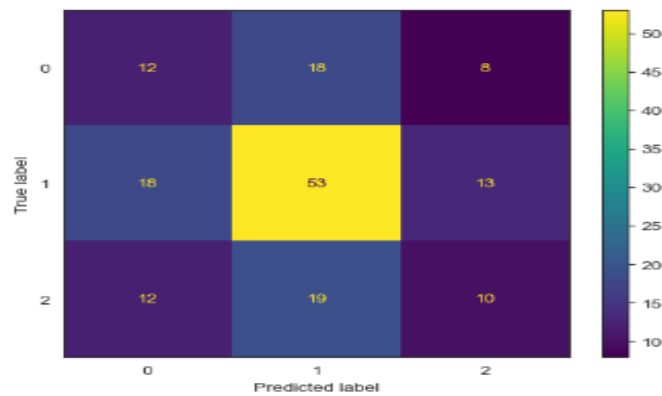
DECISION TREE

MODELLING

```
In [54]: from sklearn.metrics import classification_report, ConfusionMatrixDisplay
def metrics_display(model):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    print(classification_report(y_test, y_pred))
    ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```

```
In [55]: #DECISION TREE CLASSIFIER
from sklearn.tree import DecisionTreeClassifier
metrics_display(DecisionTreeClassifier())
```

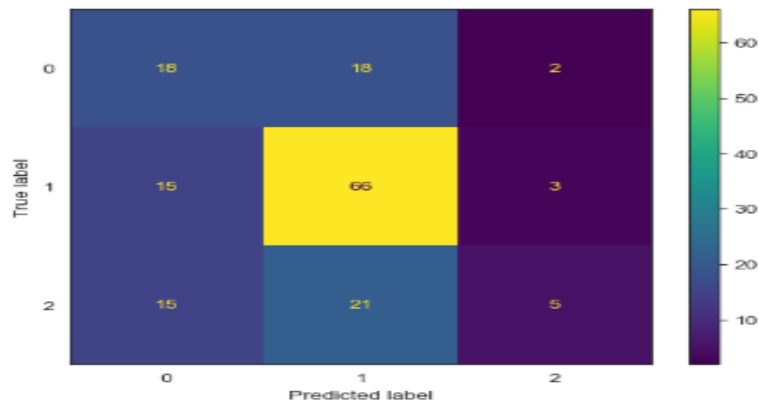
	precision	recall	f1-score	support
0	0.29	0.32	0.30	38
1	0.59	0.63	0.61	84
2	0.32	0.24	0.28	41
accuracy			0.46	163
macro avg	0.40	0.40	0.40	163
weighted avg	0.45	0.46	0.45	163



RANDOM FOREST

```
In [56]: ##Random Forest
from sklearn.ensemble import RandomForestClassifier
metrics_display(RandomForestClassifier())
```

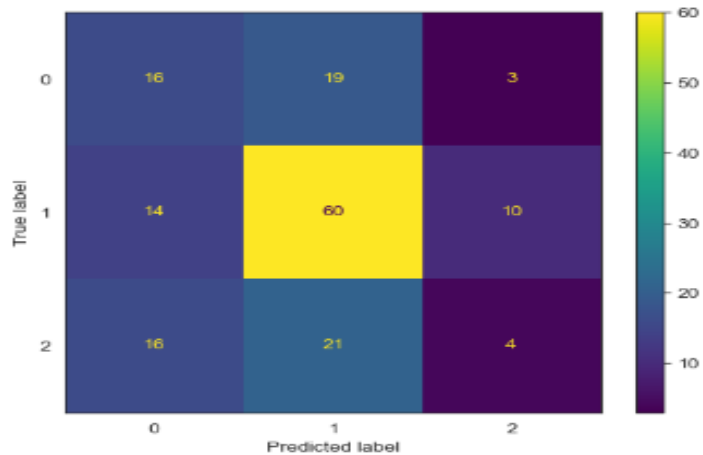
	precision	recall	f1-score	support
0	0.38	0.47	0.42	38
1	0.63	0.79	0.70	84
2	0.50	0.12	0.20	41
accuracy			0.55	163
macro avg	0.50	0.46	0.44	163
weighted avg	0.54	0.55	0.51	163



XGBCLASSIFIER

```
In [58]: #XGB Boost
from xgboost import XGBClassifier
metrics_display(XGBClassifier(use_label_encoder = False))
```

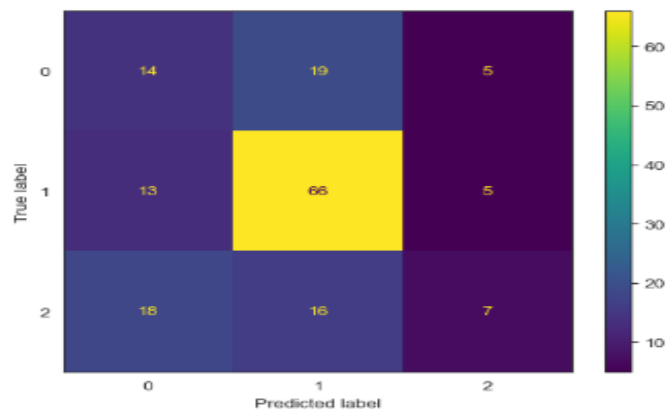
	precision	recall	f1-score	support
0	0.35	0.42	0.38	38
1	0.60	0.71	0.65	84
2	0.24	0.10	0.14	41
accuracy			0.49	163
macro avg	0.39	0.41	0.39	163
weighted avg	0.45	0.49	0.46	163



ADABOOST CLASSIFIER

```
In [57]: #Ada Boost Classifier
from sklearn.ensemble import AdaBoostClassifier
metrics_display(AdaBoostClassifier())
```

	precision	recall	f1-score	support
0	0.31	0.37	0.34	38
1	0.65	0.79	0.71	84
2	0.41	0.17	0.24	41
accuracy			0.53	163
macro avg	0.46	0.44	0.43	163
weighted avg	0.51	0.53	0.51	163



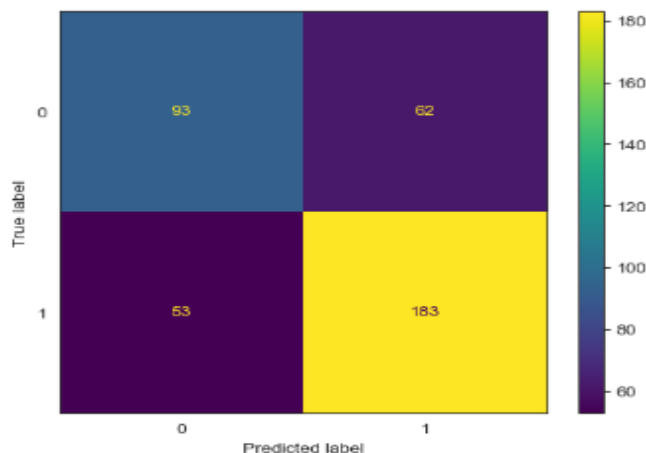
XG Boost, Ada Boot, Decision Tree, Random Forest classifier were used and evaluation metrics were calculated.

Random Forest classifier outperformed the other classifier with an accuracy of 55% A machine learning model is tested using the hold-out method to see how well it will perform on new data. So, Random Forest's accuracy was checked using the hold-out test dataset. The result of the Random Forest classifier on hold out test dataset displays an accuracy of 71%

```
In [120]: model = RandomForestClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_hold_test)
print(classification_report(y_hold_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_hold_test, y_pred)
```

	precision	recall	f1-score	support
0	0.64	0.68	0.62	155
1	0.75	0.78	0.76	236
accuracy			0.71	391
macro avg	0.69	0.69	0.69	391
weighted avg	0.70	0.71	0.70	391

```
Out[120]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c2d231b100>
```



HYPERPARAMETER TUNING

It is absolutely necessary, in order to get acceptable performance, to select the appropriate hyper-parameters for each model. However, the optimal hyper-parameters might look different depending on the data set in question and the challenge being tackled. As a result, it is essential to carry out hyper-parameter tuning utilizing methods such as grid search or random search in order to locate the optimal collection of hyper-parameters for each individual model.

Predicted label

HYPERPARAMETER TUNING

```
In [60]: from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
#creating a dictionary of hyperparameter values to search
search_space = {
    "n_estimators" : [300, 330],
    "max_depth" : [4],
    "gamma" : [0.01],
    "learning_rate" : [0.01]
}
#make a gridsearchcv obj
grid_search = GridSearchCV(estimator = XGBClassifier(use_label_encoder = False),
    param_grid = search_space,
    scoring = "accuracy",
    cv = 5,
    verbose = 4)
```

```
In [61]: grid_search.fit(x_train, y_train)
```

```
print(grid_search.best_params_)
```

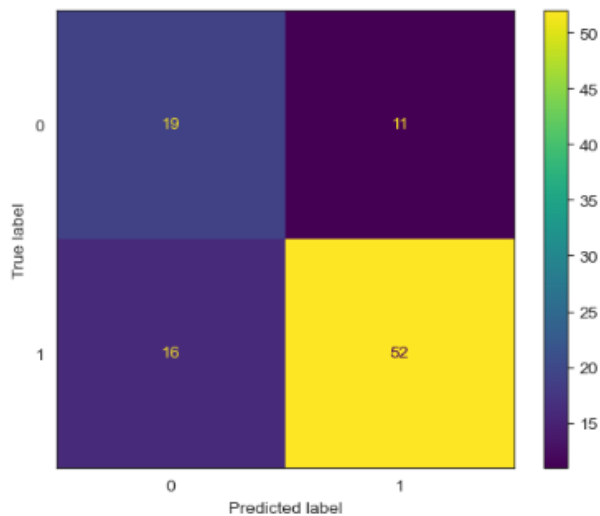
```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
[CV 1/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=300; score=0.530 total time= 10.1s
[CV 2/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=300; score=0.545 total time= 5.8s
[CV 3/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=300; score=0.549 total time= 7.0s
[CV 4/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=300; score=0.546 total time= 9.6s
[CV 5/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=300; score=0.517 total time= 7.8s
[CV 1/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=330; score=0.530 total time= 6.2s
[CV 2/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=330; score=0.547 total time= 6.9s
[CV 3/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=330; score=0.554 total time= 7.3s
[CV 4/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=330; score=0.546 total time= 5.5s
[CV 5/5] END gamma=0.01, learning_rate=0.01, max_depth=4, n_estimators=330; score=0.512 total time= 8.6s
{'gamma': 0.01, 'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 330}
```

```
In [62]: #XGBoost Classifier with hyperparameters on test data set
model = XGBClassifier(use_label_encoder = False, gamma = 0.01, learning_rate = 0.01,
```

XGBClassifier using hyperparameter

```
In [127]: model = XGBClassifier(use_label_encoder = False, gamma = 0.01, learning_rate = 0.01,
    n_estimators = 300, max_depth = 4)
metrics_display(XGBClassifier(use_label_encoder = False))
```

	precision	recall	f1-score	support
0	0.54	0.63	0.58	30
1	0.83	0.76	0.79	68
accuracy			0.72	98
macro avg	0.68	0.70	0.69	98
weighted avg	0.74	0.72	0.73	98



PIPELINE

The scikit-learn class called Column transformer is used to develop and use distinct transformer for categorical and numerical data. Most Machine learning algorithm requires the conversion of categorical data into numerical data to function. The dataset contains categorical features (Team1 and Team2). These labels do not have a particular preference, and because the data is string labels, machine learning models interpret them as having a hierarchy using a hot coding technique.

```
PIPELINE
```

```
In [64]: ##create pipeline for League matches
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
column_trans = make_column_transformer(
    (OneHotEncoder(), ["Team 1", "Team 2"]), remainder = "passthrough")
```

```
In [65]: FIFA_X = PIPELINE_FIFA.drop("Team1_result", axis = 1)
FIFA_Y = PIPELINE_FIFA["Team1_result"]
```

```
In [66]: FIFA_X
```

```
Out[66]:
```

	Team 1	Team 2	Team1_fifa_rank	Team2_fifa_rank	Team1_goalkeeper_score	Team2_goalkeeper_score	Team1_defense	Team1_offense	Team1_midfield
0	Spain	Scotland	3	87	94.0	84.0	88.5	89.3	90.
1	Austria	England	90	7	83.0	88.0	78.2	73.0	74.
2	Croatia	Hungary	25	78	77.0	74.0	80.5	78.7	79.
3	Morocco	Tunisia	33	38	88.0	51.0	75.2	73.7	71.
4	Northern Ireland	Poland	109	29	84.0	83.0	71.8	65.7	75.
...
3243	England	Hungary	5	40	83.0	85.0	85.0	88.0	84.
3244	Germany	Italy	12	6	90.0	89.0	84.0	82.7	88.
3245	Netherlands	Wales	10	18	81.0	74.0	85.2	83.0	84.
3246	Poland	Belgium	26	2	87.0	89.0	75.2	84.7	78.
3247	Chile	Ghana	28	80	79.0	74.0	75.5	78.7	78.

3248 rows x 12 columns

```
In [67]: FIFA_Y
```

```
Out[67]:
```

0	2
1	2
2	1

LEAGUE MODEL AND KNOCKOUT MODEL

LEAGUE MODEL

```
In [54]: from sklearn.pipeline import make_pipeline
pipe_league = make_pipeline(column_trans, StandardScaler(with_mean = False),
                             XGBClassifier(use_label_encoder = False, gamma = 0.01, learning_rate = 0.01,
                                             n_estimators = 300, max_depth = 4))
pipe_league.fit(FIFA_X, FIFA_Y)
```

```
Out[54]: Pipeline
> columntransformer: ColumnTransformer
  > onehotencoder > remainder
    > OneHotEncoder > passthrough
  > StandardScaler
  > XGBClassifier
```

```
In [55]: ##SAVE THE LEAGUE MODEL
import joblib
joblib.dump(pipe_league, "League_model.pkl")
```

```
Out[55]: ['League_model.pkl']
```

KNOCKOUT MODEL

```
In [56]: ##predictions for knockout stage
KNOCK_FIFA = PIPELINE_FIFA[PIPELINE_FIFA["Team1_result"] != 2]
PIPE_KNOCK_DF = KNOCK_FIFA
KNOCK_FIFA = pd.get_dummies(KNOCK_FIFA)
X = KNOCK_FIFA.drop("Team1_result", axis = 1)
Y = KNOCK_FIFA["Team1_result"]
```

```
In [66]: FIFA_KNOCK_X = PIPE_KNOCK_DF.drop("Team1_result", axis = 1)
FIFA_KNOCK_Y = PIPE_KNOCK_DF["Team1_result"]
pipe_knock = make_pipeline(column_trans, StandardScaler(with_mean = False),
                             RandomForestClassifier())
pipe_knock.fit(FIFA_KNOCK_X, FIFA_KNOCK_Y)
```

```
Out[66]: Pipeline
> columntransformer: ColumnTransformer
  > onehotencoder > remainder
    > OneHotEncoder > passthrough
  > StandardScaler
  > RandomForestClassifier
```

```
In [67]: pipe_knock.predict_proba(FIFA_KNOCK_X)
```

```
Out[67]: array([[0.02, 0.98],
                [0.96, 0.04],
                [0.1 , 0.9 ],
                ...,
                [0.04, 0.96],
                [0.89, 0.11],
                [0.68, 0.32]])
```

```
In [68]: joblib.dump(pipe_knock, "knockout_model.pkl")
```

```
Out[68]: ['knockout_model.pkl']
```

```
In [69]: joblib.dump(FIFA_KNOCK_X.columns, "col_names.pkl")
```

```
Out[69]: ['col_names.pkl']
```

```
In [70]: league_model = joblib.load("league_model.pkl")
knockout_model = joblib.load("knockout_model.pkl")
colnames = joblib.load("col_names.pkl")
```

TASK 5

MODEL EVALUATION AND VISUALIZATION

Model Evaluation

The following metrics were taken into consideration during the selection process for the models: Accuracy can be defined as the percentage of instances that are correctly classified. Precision is the ratio of actual positive results to the sum of all expected positive results; remember that this is the proportion of genuine positives to the overall number of positive tests.

The F1 score is the score that represents the harmonic mean of precision and recall.

THE LEARNING CURVE OF THE MODEL

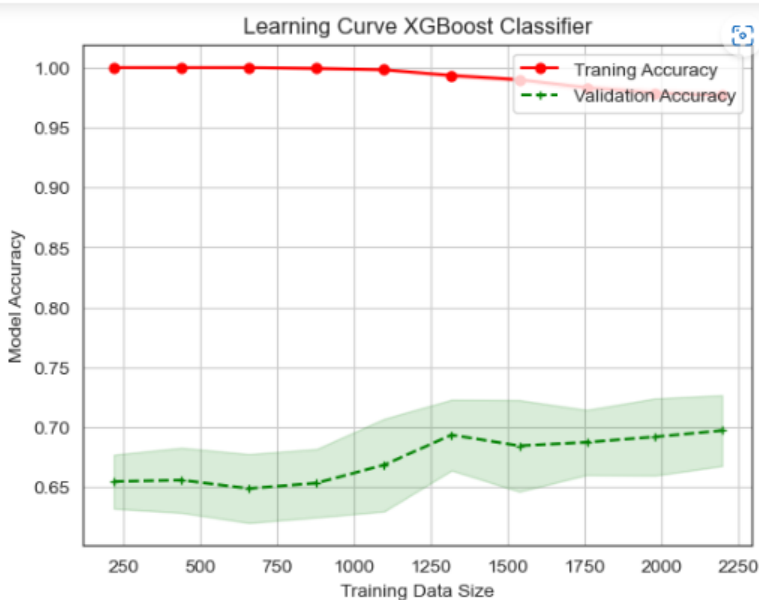
THE LEARNING CURVE FOR LEAGUE MODEL

```
In [71]: #learning curve league model XGBoost Classifier
from sklearn.model_selection import learning_curve
train_sizes, train_scores, test_scores = learning_curve(estimator = XGBClassifier(), X = X, y = Y,
                                                         cv = 10, train_sizes = np.linspace(0.1, 1.0, 10, 50),
                                                         n_jobs = 1)

##calculated training and test mean and std
train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
test_mean = np.mean(test_scores, axis = 1)
test_std = np.std(test_scores, axis = 1)

#plot the Learning curve
plt.plot(train_sizes, train_mean, color = "red", marker = "o", markersize = 5, label = "Traning Accuracy")
plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alpha = 0.15, color = "red")

plt.plot(train_sizes, test_mean, color = "green", marker = "+", markersize = 5, linestyle = "--", label = "Validation Accuracy")
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha = 0.15, color = "green")
plt.title("Learning Curve XGBoost Classifier")
plt.xlabel("Training Data Size")
plt.ylabel("Model Accuracy")
plt.grid()
plt.legend(loc = "upper right")
plt.show()
```



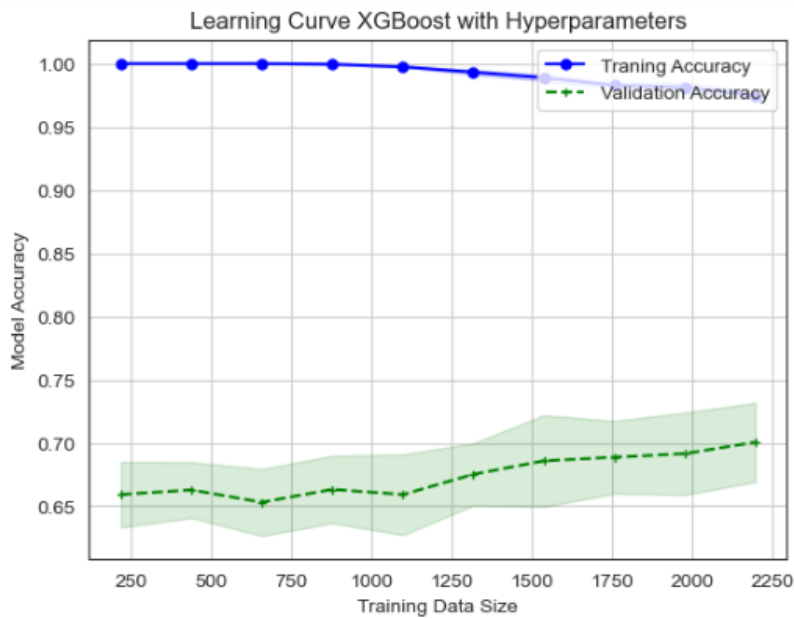
A good learning curve shows a gradual increase in performance for both the training and validation data as the number of training examples increases, with a small gap between the two curves. If the gap between the two curves is large, regularization techniques or more data should be considered to reduce overfitting.

```
In [72]: #learning curve league model XGBoost Classifier with hyperparameter
train_sizes, train_scores, test_scores = learning_curve(estimator = XGBClassifier(use_label_encoder = False, gamma = 0.01),
                                                         X = X, y = Y, cv = 10, train_sizes = np.linspace(0.1, 1.0, 10, 10),
                                                         n_jobs = 1)

##calculated training and test mean and std
train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
test_mean = np.mean(test_scores, axis = 1)
test_std = np.std(test_scores, axis = 1)

#plot the learning curve
plt.plot(train_sizes, train_mean, color = "blue", marker = "o", markersize = 5, label = "Traning Accuracy")
plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alpha = 0.15, color = "blue")

plt.plot(train_sizes, test_mean, color = "green", marker = "+", markersize = 5, linestyle = "--", label = "Validation Accuracy")
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha = 0.15, color = "green")
plt.title("Learning Curve XGBoost with Hyperparameters")
plt.xlabel("Training Data Size")
plt.ylabel("Model Accuracy")
plt.grid()
plt.legend(loc = "upper right")
plt.show()
```

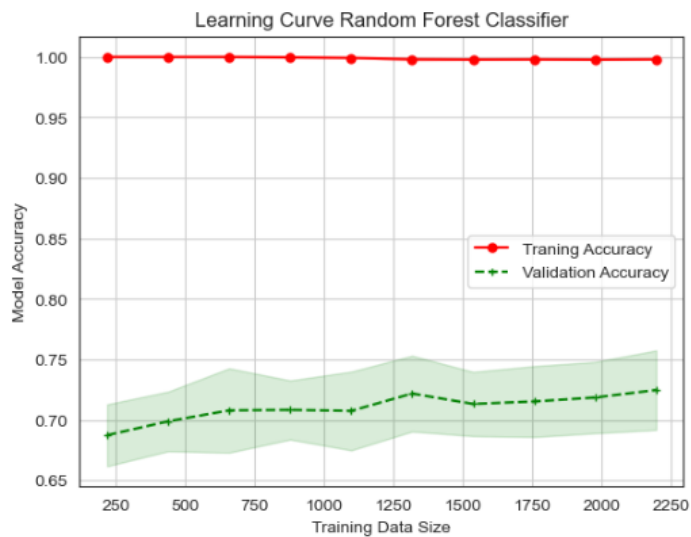


```
In [73]: #learning curve League model RandomForest Classifier
train_sizes, train_scores, test_scores = learning_curve(estimator = RandomForestClassifier(), X = X, y = Y,
                                                         cv = 10, train_sizes = np.linspace(0.1, 1.0, 10, 50),
                                                         n_jobs = 1)

##calculated training and test mean and std
train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
test_mean = np.mean(test_scores, axis = 1)
test_std = np.std(test_scores, axis = 1)

#plot the learning curve
plt.plot(train_sizes, train_mean, color = "red", marker = "o", markersize = 5, label = "Traning Accuracy")
plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alpha = 0.15, color = "red")

plt.plot(train_sizes, test_mean, color = "green", marker = "+", markersize = 5, linestyle = "--", label = "Validation Accuracy")
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha = 0.15, color = "green")
plt.title("Learning Curve Random Forest Classifier")
plt.xlabel("Training Data Size")
plt.ylabel("Model Accuracy")
plt.grid()
plt.legend(loc = "center right")
plt.show()
```



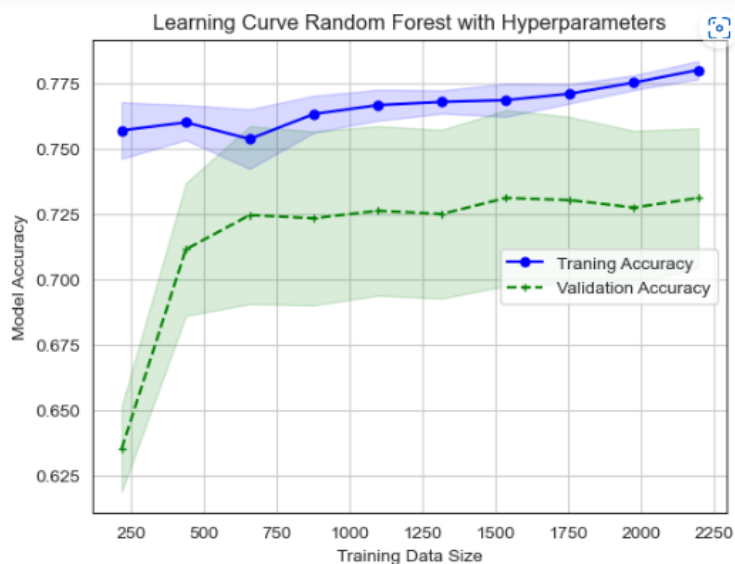
RANDOM FOREST CLASSIFIER WITH HYPER-PARAMETER

```
In [74]: #Learning curve league model Random Forest Classifier with hyperparameter
train_sizes, train_scores, test_scores = learning_curve(estimator = RandomForestClassifier(max_depth = 15,
                                                                                          n_estimators = 320,
                                                                                          max_leaf_nodes = 190,
                                                                                          min_samples_leaf = 5),
                                                         X = X, y = Y, cv = 10,
                                                         train_sizes = np.linspace(0.1, 1.0, 10), n_jobs = 1)

##calculated training and test mean and std
train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
test_mean = np.mean(test_scores, axis = 1)
test_std = np.std(test_scores, axis = 1)

#plot the learning curve
plt.plot(train_sizes, train_mean, color = "blue", marker = "o", markersize = 5, label = "Traning Accuracy")
plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alpha = 0.15, color = "blue")

plt.plot(train_sizes, test_mean, color = "green", marker = "+", markersize = 5, linestyle = "--", label = "Validation Accuracy")
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha = 0.15, color = "green")
plt.title("Learning Curve Random Forest with Hyperparameters")
plt.xlabel("Training Data Size")
plt.ylabel("Model Accuracy")
plt.grid()
plt.legend(loc = "center right")
plt.show()
```



The learning curve exhibits both high variation and overfitting in its characteristics. The utilization of hyper-parameters allows for prevention of over-fitting. GridSearchCV() was used so that the optimal hyper-parameter could be located. It examines each value that is entered into the parameter grid and then generates the best possible parameter combination based on the accuracy of the scoring metric that was selected. The value of max depth, which was discovered to be the optimal hyper-parameter was 15, max leaf nodes is 190 and min sample leaf as 5. The learning curve display the training and validation accuracy of a random forest classifier with the

use of different sizes of the training dataset. This model performs better than the XGBoost classifier and it shows a well executed bias variance trade off.

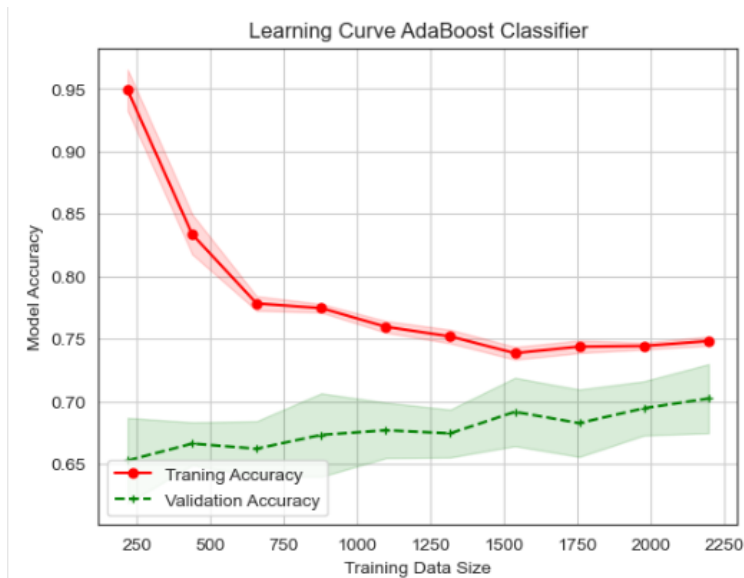
LEARNING CURVE OF ADABOOST CLASSIFIER

```
In [75]: #Learning curve League model AdaBoost Classifier
train_sizes, train_scores, test_scores = learning_curve(estimator = AdaBoostClassifier(), X = X, y = Y,
                                                         cv = 10, train_sizes = np.linspace(0.1, 1.0, 10, 10),
                                                         n_jobs = 1)

##calculated training and test mean and std
train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
test_mean = np.mean(test_scores, axis = 1)
test_std = np.std(test_scores, axis = 1)

#plot the Learning curve
plt.plot(train_sizes, train_mean, color = "red", marker = "o", markersize = 5, label = "Traning Accuracy")
plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alpha = 0.15, color = "red")

plt.plot(train_sizes, test_mean, color = "green", marker = "+", markersize = 5, linestyle = "--", label = "Validation Accuracy")
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha = 0.15, color = "green")
plt.title("Learning Curve AdaBoost Classifier")
plt.xlabel("Training Data Size")
plt.ylabel("Model Accuracy")
plt.grid()
plt.legend(loc = "lower left")
plt.show()
```



SIMULATION OF MODELS

Simulation models try to copy the way a real system works and how it makes decisions by using statistical explanations of the things that happen. For example, a line might move at a rate of 1000 units per hour on average. Simulation is a model that shows how a system works or how it could work. It helps people make decisions by letting them try out different scenarios or process changes.

The quality of the data used to make a model and the assumptions it is based on determine how accurate its simulations are. The model should look good at first glance, and the assumptions should be checked. The input-output transformations of the model should be compared to the real system's input-output transformations.

To make a simulation more accurate, the model can be run for a long enough time and either the relative tolerance or the absolute tolerance can be lowered. Then run the model simulation again and compare the results of the two runs.

SIMULATIONS TO PREDICT THE WINNER OF FIFA 2022

```
In [76]: FIFA = FIFA.dropna()
```

```
In [78]: #Generating a dataset to predict winner
FIFA_RANKINGS22 = FIFA[['date', 'home_team', 'away_team', 'home_team_fifa_rank',
                        'away_team_fifa_rank', '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']]
```

```
In [79]: HOME = FIFA_RANKINGS22[['date', 'home_team', 'home_team_fifa_rank', 'home_team_goalkeeper_score',
                                'home_team_mean_defense_score', 'home_team_mean_offense_score', 'home_team_mean_midfield_score'
                                ]].rename(columns = {'home_team': 'TEAM', 'home_team_fifa_rank': 'RANK',
                                'home_team_goalkeeper_score': 'GOALKEEPER_SCORE',
                                'home_team_mean_defense_score': 'DEFENSE_SCORE',
                                'home_team_mean_offense_score': 'OFFENSE_SCORE',
                                'home_team_mean_midfield_score': 'MIDFIELD_SCORE' })
```

```
In [80]: AWAY = FIFA_RANKINGS22[['date', 'away_team', 'away_team_fifa_rank', 'away_team_goalkeeper_score',
                                'away_team_mean_defense_score', 'away_team_mean_offense_score', 'away_team_mean_midfield_score',
                                ]].rename(columns = {'away_team': 'TEAM', 'away_team_fifa_rank': 'RANK',
                                'away_team_goalkeeper_score': 'GOALKEEPER_SCORE',
                                'away_team_mean_defense_score': 'DEFENSE_SCORE',
                                'away_team_mean_offense_score': 'OFFENSE_SCORE',
                                'away_team_mean_midfield_score': 'MIDFIELD_SCORE' })
```

```
In [81]: FIFA_RANKINGS22 = pd.concat([HOME, AWAY])
FIFA_RANKINGS22.head()
```

Out[81]:

```
Out[81]:
```

	date	TEAM	RANK	GOALKEEPER_SCORE	DEFENSE_SCORE	OFFENSE_SCORE	MIDFIELD_SCORE
8264	2004-09-03	Spain	3	94.0	86.5	89.3	90.0
8266	2004-09-04	Austria	90	83.0	76.2	73.0	74.0
8271	2004-09-04	Croatia	25	77.0	80.5	78.7	79.0
8276	2004-09-04	Iceland	80	78.0	68.8	77.0	69.0
8278	2004-09-04	Italy	9	97.0	91.8	92.3	88.0

```
In [82]: FIFA_RANKINGS22 = FIFA_RANKINGS22.sort_values(['TEAM', 'date'], ascending = [True, False])
FIFA_RANKINGS22 = FIFA_RANKINGS22.groupby('TEAM').first()
FIFA_RANKINGS22 = FIFA_RANKINGS22.reset_index()
FIFA_RANKINGS22.head()
```

```
Out[82]:
```

	TEAM	date	RANK	GOALKEEPER_SCORE	DEFENSE_SCORE	OFFENSE_SCORE	MIDFIELD_SCORE
0	Albania	2022-06-06	66	80.0	76.2	70.0	73.0
1	Algeria	2022-06-12	44	78.0	78.0	81.0	78.0
2	Angola	2021-03-25	125	68.0	75.5	70.7	68.0
3	Argentina	2022-06-01	4	84.0	82.2	89.0	84.0
4	Australia	2022-06-13	42	77.0	72.0	72.3	74.0

```
In [83]: list_2022 = ['Finland', 'Germany', 'Denmark', 'Brazil', 'France', 'Belgium', 'Croatia', 'Spain',
                    'England', 'Serbia', 'Switzerland', 'Netherlands', 'Argentina', 'IR Iran', 'Korea Republic',
                    'Japan', 'Saudi Arabia', 'Ecuador', 'Uruguay', 'Canada', 'Ghana', 'Senegal', 'Portugal', 'Poland',
                    'Tunisia', 'Morocco', 'Cameroon', 'USA', 'Mexico', 'Wales', 'Australia', 'Costa Rica']
```

```
In [84]: FINALS_FIFA22 = FIFA_RANKINGS22[FIFA_RANKINGS22["TEAM"].apply(lambda x: x in list_2022)]
FINALS_FIFA22.isnull().sum()
```



```
'Japan', 'Saudi Arabia', 'Ecuador', 'Uruguay', 'Canada', 'Ghana', 'Senegal', 'Portugal', 'Tunisia', 'Morocco', 'Cameroon', 'USA', 'Mexico', 'Wales', 'Australia', 'Costa Rica']
```

```
In [84]: FINALS_FIFA22 = FIFA_RANKINGS22[(FIFA_RANKINGS22["TEAM"].apply(lambda x: x in list_2022))]  
FINALS_FIFA22.isnull().sum()
```

```
Out[84]: TEAM          0  
date              0  
RANK              0  
GOALKEEPER_SCORE  0  
DEFENSE_SCORE     0  
OFFENSE_SCORE     0  
MIDFIELD_SCORE    0  
dtype: int64
```

```
In [85]: FINALS_FIFA22.sort_values('RANK', inplace = True)  
FINALS_FIFA22.drop('date', axis = 1, inplace = True)
```

```
In [86]: FINALS_FIFA22.to_csv('FIFA.csv', index = False)
```

```
In [87]: RANKINGS22 = pd.read_csv("FIFA.csv")  
RANKINGS22.head()
```

```
Out[87]:
```

	TEAM	RANK	GOALKEEPER_SCORE	DEFENSE_SCORE	OFFENSE_SCORE	MIDFIELD_SCORE
0	Brazil	1	89.0	84.8	86.3	86.0
1	Belgium	2	89.0	80.8	85.7	86.0
2	France	3	87.0	84.2	88.3	87.0
3	Argentina	4	84.0	82.2	89.0	84.0
4	England	5	83.0	85.0	88.0	84.0

```
In [88]: def Teamlist(Team1, Team2):  
    Team1_fifa_rank = RANKINGS22[RANKINGS22["TEAM"] == Team1]["RANK"].to_list()[0]  
    Team2_fifa_rank = RANKINGS22[RANKINGS22["TEAM"] == Team2]["RANK"].to_list()[0]  
    Team1_goalkeeper_score = RANKINGS22[RANKINGS22["TEAM"] == Team1]["GOALKEEPER_SCORE"].to_list()[0]  
    Team2_goalkeeper_score = RANKINGS22[RANKINGS22["TEAM"] == Team2]["GOALKEEPER_SCORE"].to_list()[0]  
    Team1_defense = RANKINGS22[RANKINGS22["TEAM"] == Team1]["DEFENSE_SCORE"].to_list()[0]  
    Team2_defense = RANKINGS22[RANKINGS22["TEAM"] == Team2]["DEFENSE_SCORE"].to_list()[0]  
    Team1_offense = RANKINGS22[RANKINGS22["TEAM"] == Team1]["OFFENSE_SCORE"].to_list()[0]  
    Team2_offense = RANKINGS22[RANKINGS22["TEAM"] == Team2]["OFFENSE_SCORE"].to_list()[0]  
    Team1_midfield = RANKINGS22[RANKINGS22["TEAM"] == Team1]["MIDFIELD_SCORE"].to_list()[0]  
    Team2_midfield = RANKINGS22[RANKINGS22["TEAM"] == Team2]["MIDFIELD_SCORE"].to_list()[0]  
    list_values = [[Team1, Team2, Team1_fifa_rank, Team2_fifa_rank, Team1_goalkeeper_score,  
                    Team2_goalkeeper_score, Team1_defense, Team2_defense, Team1_offense,  
                    Team2_offense, Team1_midfield, Team2_midfield]]  
    df = pd.DataFrame(data = list_values, columns = colnames)  
    return df
```

```
In [89]: #function for league results  
def league_model_result(Team1, Team2):  
    result = league_model.predict(Teamlist(Team1, Team2))  
    proba = league_model.predict_proba(Teamlist(Team1, Team2))  
    if result == 0:  
        return Team2, proba[0][0]  
    if result == 1:  
        return Team1, proba[0][1]  
    if result == 2:  
        return "Draw", proba[0][2]  
    league_model_result
```

```
In [106]: #Team1 = "Win":1, "Lose":0
def knockout_result(Team1, Team2):
    result = knockout_model.predict(Teamlist(Team1, Team2))
    proba = knockout_model.predict_proba(Teamlist(Team1, Team2))
    if result ==0:
        return Team2, proba[0][0]
    if result ==1:
        return Team1, proba[0][1]
```

Simulation Of League Matches

```
In [90]: def league_round(Team1, Team2, Team3, Team4):
    group_winners = []
    match1 = league_model_result(Team1, Team2)
    match2 = league_model_result(Team1, Team3)
    match3 = league_model_result(Team1, Team4)
    match4 = league_model_result(Team2, Team3)
    match5 = league_model_result(Team2, Team4)
    match6 = league_model_result(Team3, Team4)
    points = [match1[0], match2[0], match3[0], match4[0], match5[0], match6[0]]
    Team1_points = points.count(Team1) * 3
    Team2_points = points.count(Team2) * 3
    Team3_points = points.count(Team3) * 3
    Team4_points = points.count(Team4) * 3
    if match1 == "Draw":
        Team1_points += 1
        Team2_points += 1
    if match2 == "Draw":
        Team1_points += 1
        Team3_points += 1
    if match3 == "Draw":
        Team1_points += 1
        Team4_points += 1
```

Simulation Of League Matches

```
In [90]: def league_round(Team1, Team2, Team3, Team4):
    group_winners = []
    match1 = league_model_result(Team1, Team2)
    match2 = league_model_result(Team1, Team3)
    match3 = league_model_result(Team1, Team4)
    match4 = league_model_result(Team2, Team3)
    match5 = league_model_result(Team2, Team4)
    match6 = league_model_result(Team3, Team4)
    points = [match1[0], match2[0], match3[0], match4[0], match5[0], match6[0]]
    Team1_points = points.count(Team1) * 3
    Team2_points = points.count(Team2) * 3
    Team3_points = points.count(Team3) * 3
    Team4_points = points.count(Team4) * 3
    if match1 == "Draw":
        Team1_points += 1
        Team2_points += 1
    if match2 == "Draw":
        Team1_points += 1
        Team3_points += 1
    if match3 == "Draw":
        Team1_points += 1
        Team4_points += 1
    if match4 == "Draw":
        Team2_points += 1
        Team3_points += 1
    if match5 == "Draw":
        Team2_points += 1
        Team4_points += 1
    if match6 == "Draw":
        Team3_points += 1
        Team4_points += 1
    dict = {Team1:Team1_points, Team2:Team2_points, Team3:Team3_points, Team4:Team4_points}
    group_winners = pd.DataFrame(list(dict.items()),
                                  columns = ["Team", "Points"]).sort_values("Points",
                                                                              ascending = False)[0:2]

    return group_winners
```

FUNCTIONS DEFINED FOR THE SIMULATIONS

TEAM LIST

LEAGUE MODEL RESULT

KNOCKOUT RESULTS

LEAGUE ROUNDS

This following functions takes inputs such as the team names, the league model and knockout model previously created, the probability of each outcomes and to also make a predictions, one can obtain the probability of each outcome or the expected winner probabilities.

GROUP STAGE OUTCOME

```
In [91]: #Group A matches
Group_A_winners = league_round("Finland", "Ecuador", "Senegal", "Netherlands")
Group_A_winners
```

```
Out[91]:
```

	Team	Points
2	Senegal	6
3	Netherlands	6

```
In [93]: #Group B matches
Group_B_winners = league_round("England", "IR Iran", "USA", "Wales")
Group_B_winners
```

```
Out[93]:
```

	Team	Points
0	England	9
1	IR Iran	6

```
In [94]: #Group C matches
Group_C_winners = league_round("Argentina", "Saudi Arabia", "Mexico", "Poland")
Group_C_winners
```

```
Out[94]:
```

	Team	Points
0	Argentina	9
2	Mexico	6

```
In [95]: #Group D matches
Group_D_winners = league_round("France", "Australia", "Denmark", "Tunisia")
Group_D_winners
```

```
Out[95]:
```

	Team	Points
--	------	--------

```
In [95]: #Group D matches
Group_D_winners = league_round("France", "Australia", "Denmark", "Tunisia")
Group_D_winners
```

Out[95]:

	Team	Points
0	France	9
2	Denmark	6

```
In [96]: #Group E matches
Group_E_winners = league_round("Spain", "Costa Rica", "Germany", "Japan")
Group_E_winners
```

Out[96]:

	Team	Points
0	Spain	9
2	Germany	6

```
In [97]: #Group F matches
Group_F_winners = league_round("Belgium", "Canada", "Morocco", "Croatia")
Group_F_winners
```

Out[97]:

	Team	Points
0	Belgium	6
2	Morocco	6

```
In [98]: #Group G matches
Group_G_winners = league_round("Brazil", "Serbia", "Switzerland", "Cameroon")
Group_G_winners
```

Out[98]:

	Team	Points
0	Brazil	9

```
In [98]: #Group G matches
Group_G_winners = league_round("Brazil", "Serbia", "Switzerland", "Cameroon")
Group_G_winners
```

Out[98]:

	Team	Points
0	Brazil	9
2	Switzerland	6

```
In [99]: #Group H matches
Group_H_winners = league_round("Portugal", "Ghana", "Uruguay", "Korea Republic")
Group_H_winners
```

Out[99]:

	Team	Points
0	Portugal	9
2	Uruguay	6

ROUND OF 16

SIMULATIONS FOR KNOCK OUT LEAGUES ROUND OF 16

```
In [107]: #Group1A and Group2B
MatchAB = knockout_result(Group_A_winners["Team"].iloc[0], Group_B_winners["Team"].iloc[1])
MatchAB
```

```
Out[107]: ('Senegal', 0.78)
```

```
In [108]: #Group1C and Group2D
MatchCD = knockout_result(Group_C_winners["Team"].iloc[0], Group_D_winners["Team"].iloc[1])
MatchCD
```

```
Out[108]: ('Argentina', 0.6275)
```

```
In [109]: #Group1B and Group2A
MatchBA = knockout_result(Group_B_winners["Team"].iloc[0], Group_A_winners["Team"].iloc[1])
MatchBA
```

```
Out[109]: ('England', 0.55)
```

```
In [110]: #Group1D and Group2C
MatchDC = knockout_result(Group_D_winners["Team"].iloc[0], Group_C_winners["Team"].iloc[1])
MatchDC
```

```
Out[110]: ('France', 0.71)
```

```
In [111]: #Group1E and Group2F
MatchEF = knockout_result(Group_E_winners["Team"].iloc[0], Group_F_winners["Team"].iloc[1])
MatchEF
```

```
Out[111]: ('Spain', 0.79)
```

```
In [112]: #Group1G and Group2H
MatchGH = knockout_result(Group_G_winners["Team"].iloc[0], Group_H_winners["Team"].iloc[1])
MatchGH
```

```
Out[112]: ('Brazil', 0.81)
```

```
In [113]: #Group1F and Group2E
MatchFE = knockout_result(Group_F_winners["Team"].iloc[0], Group_E_winners["Team"].iloc[1])
MatchFE
```

```
Out[113]: ('Belgium', 0.51)
```

```
In [114]: #Group1H and Group2G
MatchHG = knockout_result(Group_H_winners["Team"].iloc[0], Group_G_winners["Team"].iloc[1])
MatchHG
```

```
Out[114]: ('Portugal', 0.71)
```

ROUND OF 16 WINNERS

```
In [116]: print("", MatchAB, "\n", MatchCD, "\n", MatchBA, "\n", MatchDC, "\n", MatchEF, "\n",
              MatchGH, "\n", MatchFE, "\n", MatchHG)
```

```
('Senegal', 0.78)
('Argentina', 0.6275)
('England', 0.55)
('France', 0.71)
('Spain', 0.79)
('Brazil', 0.81)
('Belgium', 0.51)
('Portugal', 0.71)
```

Quarterfinals

```
In [117]: Match1 = knockout_result(MatchAB[0], MatchCD[0])  
Match1
```

```
Out[117]: ('Argentina', 0.62)
```

```
In [118]: Match2 = knockout_result(MatchBA[0], MatchDC[0])  
Match2
```

```
Out[118]: ('France', 0.55)
```

```
In [119]: Match3 = knockout_result(MatchEF[0], MatchGH[0])  
Match3
```

```
Out[119]: ('Brazil', 0.62)
```

```
In [122]: Match4 = knockout_result(MatchFE[0], MatchHG[0])  
Match4
```

```
Out[122]: ('Belgium', 0.66)
```

Quarterfinal Winners

```
In [123]: print("", Match1, "\n", Match2, "\n", Match3, "\n", Match4)  
  
('Argentina', 0.62)  
('France', 0.55)  
('Brazil', 0.62)  
('Belgium', 0.66)
```

Semi-finals

```
In [127]: Semi_match1 = knockout_result(Match1[0], Match2[0])  
Semi_match1
```

```
Out[127]: ('France', 0.55)
```

```
In [128]: Semi_match2 = knockout_result(Match3[0], Match4[0])  
Semi_match2
```

```
Out[128]: ('Brazil', 0.51)
```

Semi-finals Winners

```
In [129]: print("", Semi_match1, "\n", Semi_match2)  
  
('France', 0.55)  
('Brazil', 0.51)
```

Third place

```
In [130]: Team1 = list([Match1[0], Match2[0]])
Team1.remove(Semi_match1[0])
Team1 = Team1[0]
Team1
```

```
Out[130]: 'Argentina'
```

```
In [131]: Team2 = list([Match3[0], Match4[0]])
Team2.remove(Semi_match2[0])
Team2 = Team2[0]
Team2
```

```
Out[131]: 'Belgium'
```

```
In [132]: Third_place = knockout_result(Team1, Team2)
Third_place
```

```
Out[132]: ('Belgium', 0.57)
```

FINALS OF FIFA 2022

```
In [133]: Finals = knockout_result(Semi_match1[0], Semi_match2[0])
Finals
```

```
Out[133]: ('Brazil', 0.53)
```

```
In [134]: Runner = list([Semi_match1[0], Semi_match2[0]])
Runner.remove(Finals[0])
Runner = Runner[0]
Runner
```

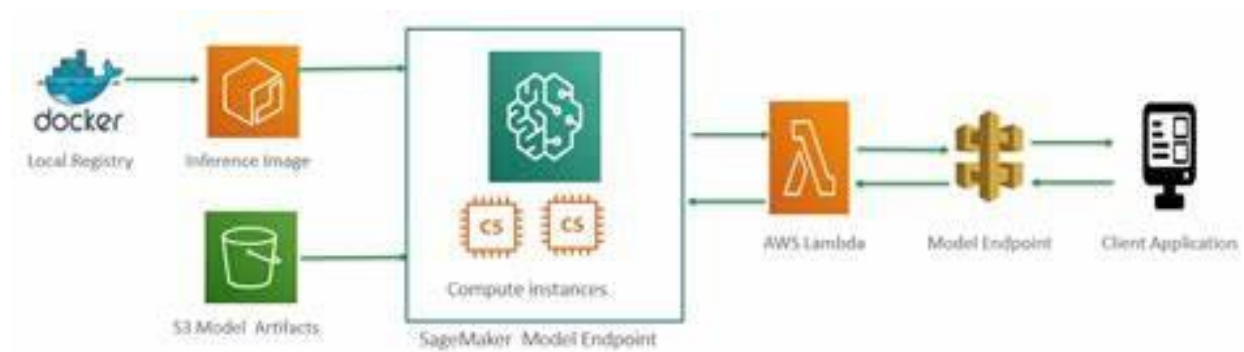
```
Out[134]: 'France'
```

```
In [136]: print("WINNER IS {}".format(Finals[0]))
print("2nd PLACE IS {}".format(Runner))
print("3rd PLACE IS {}".format(Third_place[0]))
```

```
WINNER IS Brazil
2nd PLACE IS France
3rd PLACE IS Belgium
```

The winner of FIFA 2022 will be Brazil with a winning probability of 53%

TASK 6: MODEL DEPLOYMENT



Model deployment is a key step in the machine learning process. It includes putting the trained model into a production environment where it can be used to make predictions on new data. In this task, we'll talk about how to use SageMaker Hosting Services to install the model and what the best professional practices are to follow when it comes to ethics and privacy.

The following steps explain how to use SageMaker hosting services to put the model learned in Task 4 into use:

Create a model artifact: Before deploying the model, we need to create a model artifact that includes the trained model parameters and any other metadata needed for prediction.

Deploy the model: Once the model artifact has been made, it can be sent to a SageMaker API.

Predict: Once the model is up and running, we can use the SageMaker SDK to use new data to make predictions.

When putting a model into production, it's important to follow rules about ethics and privacy to make sure the model is used in a responsible way and doesn't hurt people or groups. This is done by making sure that:

- **Fairness:** Make sure that the model doesn't favor any one group or person over another. This can be done by carefully choosing the data used for training, checking the model for bias and fairness, and fixing any problems that are found.
- **Protect the privacy of individuals and their data** by utilizing suitable data anonymization and encryption techniques, and by ensuring that sensitive data is only accessible by

authorized staff. This can be accomplished by ensuring that sensitive data is only accessed by authorized personnel.

- **Transparency:** Please provide detailed explanations and documentation regarding the operation of the model, as well as its training and the process through which it generates predictions. This can assist users and stakeholders create trust in one another and be accountable for their actions.
- **Safety:** Check to see that the model and all of the data linked with it are safe from intrusion by unauthorized users or assaults. This is something that can be accomplished by putting in place suitable security measures like as encryption, access controls, and monitoring.
- **Establish transparent policies and procedures** for utilizing and deploying machine learning models, and make sure that these policies and procedures are in line with the values and ethical standards of the firm. This can assist in ensuring that the model is utilized in a responsible manner and in a way that is to the benefit of all parties involved.

Best privacy practice to predict winner

1. **Raising awareness;** it is vital to stay up to date on the rules and regulations that are relevant to this field. This can be accomplished by participating in appropriate training seminars and workshops.
2. **Develop an acceptable privacy policy:** this would be essential in order to decide how personal information is gathered and utilized. The policy must to be straightforward and easily accessible.
3. **Get people's permission** before utilizing their personal information, and make sure that data is acquired, handled, and stored ethically.

