Import the Libraries

```
import numpy as np

# dataframe format options
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('max_colwidth', 50)

import re

import requests
from bs4 import BeautifulSoup
import time

from google.colab import drive

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

import matplotlib.pyplot as plt

import seaborn as sns
sns.set_style("whitegrid")

import sklearn as sk

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.multioutput import MultiOutputClassifier
from xgboost import XGBClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.multioutput import MultiOutputRegressor

from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import r2_score

# To store a model
import joblib

# for google credentials
#from google.colab import auth
#auth.authenticate_user()
#from oauth2client.client import GoogleCredentials

# to import google sheets
#import googleapiclient.discovery
#import gspread
```

```
# Mount your Google Drive
drive.mount('/content/drive')

# Create a directory in Google Drive where you want to save your file
#!mkdir -p '/content/drive/My Drive/My Folder'
gfolder = '/content/drive/My Drive/Colab Notebooks'
gmodels = gfolder + "/models"
#print(gmodels)
```

```
Mounted at /content/drive
```

Get the Cricket datasource from Google Drive

Add Moneyline Caculations

https://www.vegasinsider.com/moneyline-calculator/#:~:text=CALCULATING%20POSITIVE%20AND%20NEGATIVE%20MONEYLINE%20ODDS&text=For%20negative%20odds%2C%20you%20divide,number%20by%20the%20wager%20amount.&text=To%20calculate%20positive%20odds%2C%20you,that%20number%20by%20your%20wager.

Publish to Web

https://stackoverflow.com/questions/19611729/getting-google-spreadsheet-csv-into-a-pandas-dataframe

File -> Share -> Publish to WEB -> Entire doc as CSV

TEAM

https://docs.google.com/spreadsheets/d/e/2PACX-1vQQ1h8ZpfXcXb8ISOzghHgfkKE5zvP2Crb9fCHzoQAvPbbaEorkKixQYZiBkq1Hi8JUpjE5fJhT4dTG/pub?gid=303052024&single=true&output=csv

```
# copy of cricket_datasource
#team_url_csv = "https://docs.google.com/spreadsheets/d/e/2PACX-1vTxuBSHzKjVH-4FBSGi3eubMO9yzTM0WJWfWDGu4CWMgbVEobUkkw0z4SoYeSwUDDAV2tygxVCC
# prednov10
team_url_csv = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSJYmiFqa4zPem1fORoEQ7DLY7MtxvDBe17ZqjEsy3XQntLjKulCbGL-TSft5lnAg/pub?output
df = pd.read_csv(team_url_csv)
df_bk = df
#print(df)
#df.sample(10)
df.head(10)
```

	Team1	Team2	Date_of_Game	Month	Location	Location_Country	Team1_Win_Lo
0	Australia	South Africa	11/16/2023	Nov	Lucknow	India	
1	India	New Zealand	11/15/20023	Nov	Mumbai	India	
2	England	Pakistan	11/11/2023	Nov	Kolkata	India	
3	Australia	Bangladesh	11/10/2023	Nov	Mumbai	India	
4	New Zealand	Sri Lanka	11/9/2023	Nov	Bangalore	India	
5	Australia	New Zealand	10/27/2023	Oct	Dharamshala	India	
6	Pakistan	South Africa	10/27/2023	Oct	Chennai	India	
7	England	Sri Lanka	10/26/2023	Oct	Bangalore	India	
8	South Africa	Bangladesh	10/24/2023	Oct	Mumbai	India	
9	India	New Zealand	10/22/2023	Oct	Dharamshala	India	

10 rows × 47 columns

```
# Team pair for prediction
# ind = 1 eng = 2 aus = 3 saf = 4 nzd = 5 pak = 6 bgl = 7 slk = 8

team1_pred = 'India'
team2_pred = 'Australia'

#team1_pred = 'Australia'
#team2_pred = 'South Africa'

#team1_pred = 'India'
#team2_pred = 'New Zealand'

#team1_pred = 'England'
#team2_pred = 'Pakistan'

#team1_pred = 'Australia'
#team2_pred = 'Bangladesh'

#team1_pred = 'New Zealand'
#team2_pred = 'Sri Lanka'

df_team1_2 = df[(df['Team1'].isin([team1_pred, team2_pred])) & (df['Team2'].isin([team1_pred, team2_pred]))]
#print(df_team1_2)
df_team1_2 = pd.concat([df_team1_2, df_team1_2, df_team1_2, df_team1_2, df_team1_2, df_team1_2, df_team1_2, df_team1_2, df_team1_2, df_team1
df = df_team1_2
#df.columns
#df.info
#df.sample(10)

#df_ip = df.loc[0]
#df_ip = (df.iloc[0]).to_dict()

#df_ip['Team1_Encoded'] = 7
#df_ip['Team2_Encoded'] = 3

#print(df_ip)
#df_ip['Team2_Encoded']
```

```
#print(df)
#print(df.sample(10))
df.describe()
```

	Team1_Win_Loss_Ratio	Team2_Win_Loss_Ratio	Team1_Score	Team2_Score	Standardized_Wins_by_
count	9.000000e+01	9.000000e+01	90.000000	90.000000	
mean	7.780000e-01	2.220000e-01	173.000000	172.111111	
std	2.232886e-16	2.791107e-17	33.459686	34.904185	
min	7.780000e-01	2.220000e-01	92.000000	90.000000	
25%	7.780000e-01	2.220000e-01	161.000000	152.000000	
50%	7.780000e-01	2.220000e-01	186.000000	186.000000	
75%	7.780000e-01	2.220000e-01	195.000000	195.000000	
max	7.780000e-01	2.220000e-01	208.000000	211.000000	

8 rows × 37 columns

Add Moneyline metrics

```
df['Team1_Moneyline_Oddsportal'] = np.where(df['Team1_Odd_Oddsportal'] < 0, 100/-df['Team1_Odd_Oddsportal'], df['Team1_Odd_Oddsportal']/100)
df['Team2_Moneyline_Oddsportal'] = np.where(df['Team2_Odd_Oddsportal'] < 0, 100/-df['Team2_Odd_Oddsportal'], df['Team2_Odd_Oddsportal']/100)
df.sample(10)
```

	Team1	Team2	Date_of_Game	Month	Location	Location_Country	Team1_Win_Loss_Status	Team2_Win_Loss_Status	Team1_Win_Loss_Ratio	_Wi
13	India	Australia	9/20/2022	Sep	Mohali	India	L	W	0.778	
7	India	Australia	12/6/2020	Dec	Sydney	Australia	W	L	0.778	
44	India	Australia	12/4/2020	Dec	Griffith	Australia	W	L	0.778	
86	India	Australia	10/20/2021	Oct	Dubai	United Arab Emirates	W	L	0.778	
50	India	Australia	10/20/2021	Oct	Dubai	United Arab Emirates	W	L	0.778	
24	India	Australia	12/8/2020	Dec	Sydney	Australia	L	W	0.778	
36	India	Australia	10/08/2023	Oct	Chennai	India	W	L	0.778	
60	India	Australia	12/8/2020	Dec	Sydney	Australia	L	W	0.778	

Spread Calculation

Sources:

1. BARD
2. https://help.smarkets.com/hc/en-gb/articles/214058369-How-to-calculate-implied-probability-in-betting

```
df['Team1_Spread_Oddsportal'] = np.where(df['Team1_Odd_Oddsportal'] < 0,
 (abs(df['Team1_Odd_Oddsportal'])/(abs(df['Team1_Odd_Oddsportal']) + 100))*100, 100/(df['Team1_Odd_Oddsportal'] + 100) * 100)
df['Team2_Spread_Oddsportal'] = np.where(df['Team2_Odd_Oddsportal'] < 0,
 (abs(df['Team2_Odd_Oddsportal'])/(abs(df['Team2_Odd_Oddsportal']) + 100))*100, 100/(df['Team2_Odd_Oddsportal'] + 100) * 100)

#df['Spread'] = 100 * ((1/df['Team1_Odd_Oddsportal']) * 100 - (1/df['Team2_Odd_Oddsportal']) * 100)
#df.sample(10)
#print(df)
df.head(10)
```

	Team1	Team2	Date_of_Game	Month	Location	Location_Country	Team1_Win_Loss_Stat
0	India	Australia	10/08/2023	Oct	Chennai	India	
1	India	Australia	10/16/2022	Oct	Perth	Australia	
2	India	Australia	9/25/2022	Sep	Hyderabad	India	
3	India	Australia	9/23/2022	Sep	Nagpur	India	
4	India	Australia	9/20/2022	Sep	Mohali	India	
5	India	Australia	10/20/2021	Oct	Dubai	United Arab Emirates	
6	India	Australia	12/8/2020	Dec	Sydney	Australia	
7	India	Australia	12/6/2020	Dec	Sydney	Australia	
8	India	Australia	12/4/2020	Dec	Griffith	Australia	
9	India	Australia	10/08/2023	Oct	Chennai	India	

10 rows × 51 columns

A Build the Model

> Random Forest Classifier
>
> Logistic Regression
>
> Decision Tree Classifier
>
> Gradient Boosting Classifier
>
> XGBoost Classifier

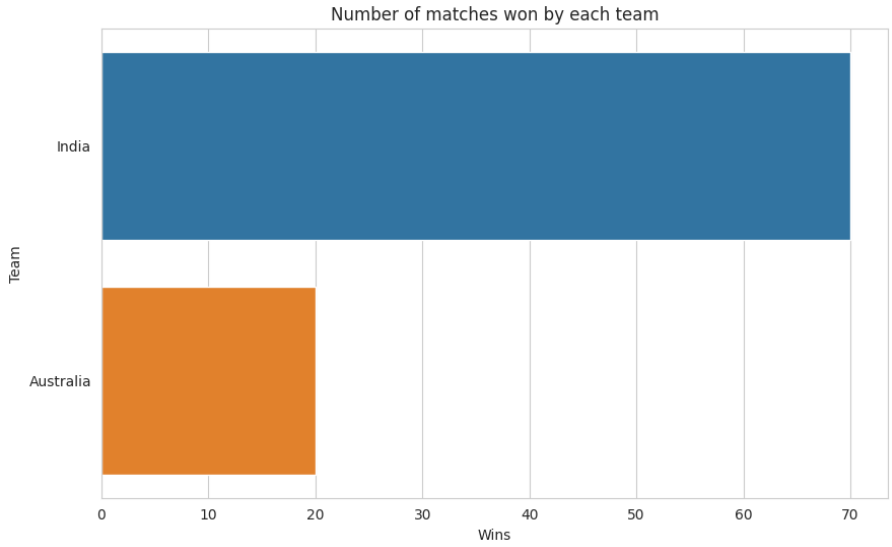
A1 Data visualization

```
# save original data to recover, for debugging reasons
  # uncomment the following and comment back for recovery
#df = df_before_encoding
df_before_encoding = df

df['Date_of_Game'] = pd.to_datetime(df['Date_of_Game'])
#df.info()
```

```
plt.figure(figsize=(10, 6))
sns.countplot(y='Winner', data=df, order=df['Winner'].value_counts().index)
plt.xlabel('Wins')
plt.ylabel('Team')
plt.title('Number of matches won by each team')
```

```
Text(0.5, 1.0, 'Number of matches won by each team')
```



Data Analysis

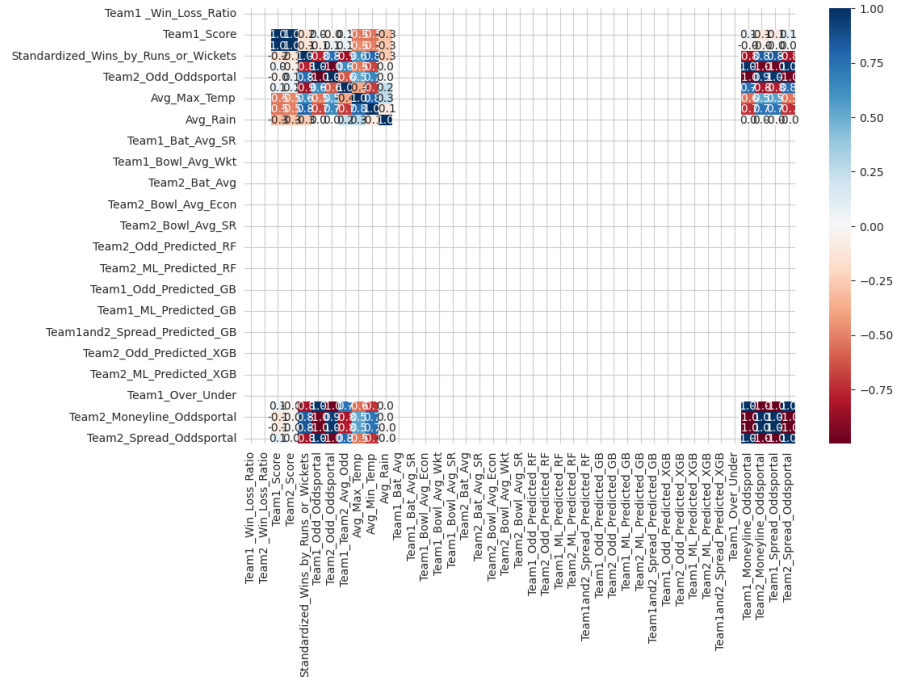
CORR HeatMap

```
# simple corr, w/o values
corr = df.corr()
#sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns)

# colormap to a divergent
#sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, vmin=-1, vmax=1, cmap="RdBu")

# include values and plot
fig, ax = plt.subplots()
fig.set_size_inches(11, 7)
sns.heatmap(corr, annot=True, fmt=".1f", cmap="RdBu", center=0, ax=ax)
```

```
<ipython-input-10-509a86891e8a>:2: FutureWarning: The default value of numeric_only in D
  corr = df.corr()
<Axes: >
```



PCA to analze for dimentionality

```
df.columns
```

```
Index(['Team1', 'Team2', 'Date_of_Game', 'Month', 'Location',
       'Location_Country', 'Team1_Win_Loss_Status', 'Team2_Win_Loss_Status',
       'Team1 _Win_Loss_Ratio', 'Team2 _Win_Loss_Ratio', 'Format',
       'Team1_Score', 'Team2_Score', 'Winner',
       'Standardized_Wins_by_Runs_or_Wickets', 'Team1_Odd_Oddsportal',
       'Team2_Odd_Oddsportal', 'Team1_Team2_Avg_Odd', 'Avg_Max_Temp',
       'Avg_Min_Temp', 'Avg_Rain', 'Team1_Bat_Avg', 'Team1_Bat_Avg_SR',
       'Team1_Bowl_Avg_Econ', 'Team1_Bowl_Avg_Wkt', 'Team1_Bowl_Avg_SR',
       'Team2_Bat_Avg', 'Team2_Bat_Avg_SR', 'Team2_Bowl_Avg_Econ',
       'Team2_Bowl_Avg_Wkt', 'Team2_Bowl_Avg_SR', 'Team1_Odd_Predicted_RF',
       'Team2_Odd_Predicted_RF', 'Team1_ML_Predicted_RF',
       'Team2_ML_Predicted_RF', 'Team1and2_Spread_Predicted_RF',
       'Team1_Odd_Predicted_GB', 'Team2_Odd_Predicted_GB',
       'Team1_ML_Predicted_GB', 'Team2_ML_Predicted_GB',
       'Team1and2_Spread_Predicted_GB', 'Team1_Odd_Predicted_XGB',
       'Team2_Odd_Predicted_XGB', 'Team1_ML_Predicted_XGB',
       'Team2_ML_Predicted_XGB', 'Team1and2_Spread_Predicted_XGB',
       'Team1_Over_Under', 'Team1_Moneyline_Oddsportal',
       'Team2_Moneyline_Oddsportal', 'Team1_Spread_Oddsportal',
       'Team2_Spread_Oddsportal'],
      dtype='object')
```

A2 Data Preparation

Encoding may be necessary for all Object type features of the dataframe

```
encoder = LabelEncoder()

# Date_of_Game
df['Date_of_Game_Encoded'] = encoder.fit_transform(df['Date_of_Game'])
#df[['Date_of_Game', "Date_of_Game_Encoded"]].sample(5)

# Month
df['Month_Encoded'] = encoder.fit_transform(df['Month'])

# Location
df['Location_Encoded'] = encoder.fit_transform(df['Location'])

# Location_Country
#df['Location_Country_Encoded'] = encoder.fit_transform(df['Location_Country'])

# Team1_Win_Loss_Status
df['Team1_Win_Loss_Status_Encoded'] = encoder.fit_transform(df['Team1_Win_Loss_Status'])

# Team2_Win_Loss_Status
df['Team2_Win_Loss_Status_Encoded'] = encoder.fit_transform(df['Team2_Win_Loss_Status'])

# Format
df['Format_Encoded'] = encoder.fit_transform(df['Format'])

# Winner
df['Winner_Encoded'] = encoder.fit_transform(df['Winner'])
```

```
# Team1 & Team2, manual encoding
# India, England, Australia, South Africa, New Zealand, Pakistan, Bangladesh, Sri Lanka as 1 through 8, in order
#df = pd.get_dummies(df, columns=["Team1", "Team2"], drop_first=True)

ind = 1
eng = 2
aus = 3
saf = 4
nzd = 5
pak = 6
bgl = 7
slk = 8

df.loc[df['Team1'] == "India", 'Team1_Encoded'] = ind
df.loc[df['Team1'] == "England", 'Team1_Encoded'] = eng
df.loc[df['Team1'] == "Australia", 'Team1_Encoded'] = aus
df.loc[df['Team1'] == "South Africa", 'Team1_Encoded'] = saf
df.loc[df['Team1'] == "New Zealand", 'Team1_Encoded'] = nzd
df.loc[df['Team1'] == "Pakistan", 'Team1_Encoded'] = pak
df.loc[df['Team1'] == "Bangladesh", 'Team1_Encoded'] = bgl
df.loc[df['Team1'] == "Sri Lanka", 'Team1_Encoded'] = slk

df.loc[df['Team2'] == "India", 'Team2_Encoded'] = ind
df.loc[df['Team2'] == "England", 'Team2_Encoded'] = eng
df.loc[df['Team2'] == "Australia", 'Team2_Encoded'] = aus
df.loc[df['Team2'] == "South Africa", 'Team2_Encoded'] = saf
df.loc[df['Team2'] == "New Zealand", 'Team2_Encoded'] = nzd
df.loc[df['Team2'] == "Pakistan", 'Team2_Encoded'] = pak
df.loc[df['Team2'] == "Bangladesh", 'Team2_Encoded'] = bgl
df.loc[df['Team2'] == "Sri Lanka", 'Team2_Encoded'] = slk

df.loc[df['Winner'] == "India", 'Winner_Encoded'] = ind
df.loc[df['Winner'] == "England", 'Winner_Encoded'] = eng
df.loc[df['Winner'] == "Australia", 'Winner_Encoded'] = aus
df.loc[df['Winner'] == "South Africa", 'Winner_Encoded'] = saf
df.loc[df['Winner'] == "New Zealand", 'Winner_Encoded'] = nzd
df.loc[df['Winner'] == "Pakistan", 'Winner_Encoded'] = pak
df.loc[df['Winner'] == "Bangladesh", 'Winner_Encoded'] = bgl
df.loc[df['Winner'] == "Sri Lanka", 'Winner_Encoded'] = slk

df.loc[df['Location_Country'] == "India", 'Location_Country_Encoded'] = ind
df.loc[df['Location_Country'] == "England", 'Location_Country_Encoded'] = eng
df.loc[df['Location_Country'] == "Australia", 'Location_Country_Encoded'] = aus
df.loc[df['Location_Country'] == "South Africa", 'Location_Country_Encoded'] = saf
df.loc[df['Location_Country'] == "New Zealand", 'Location_Country_Encoded'] = nzd
df.loc[df['Location_Country'] == "Pakistan", 'Location_Country_Encoded'] = pak
df.loc[df['Location_Country'] == "Bangladesh", 'Location_Country_Encoded'] = bgl
df.loc[df['Location_Country'] == "Sri Lanka", 'Location_Country_Encoded'] = slk

df['Team1_Encoded'] = df['Team1_Encoded'].astype(int)
df['Team2_Encoded'] = df['Team2_Encoded'].astype(int)
df['Winner_Encoded'] = df['Winner_Encoded'].astype(int)
df['Location_Country_Encoded'] = df['Winner_Encoded'].astype(int)
```

```
## Backup and drop the non int/float fields
 # restore if needed for debugging
#df = df_before_modeling
print(df.columns)
df_before_modeling = df
```

```
Index(['Team1', 'Team2', 'Date_of_Game', 'Month', 'Location',
       'Location_Country', 'Team1_Win_Loss_Status', 'Team2_Win_Loss_Status',
       'Team1 _Win_Loss_Ratio', 'Team2 _Win_Loss_Ratio', 'Format',
       'Team1_Score', 'Team2_Score', 'Winner',
       'Standardized_Wins_by_Runs_or_Wickets', 'Team1_Odd_Oddsportal',
       'Team2_Odd_Oddsportal', 'Team1_Team2_Avg_Odd', 'Avg_Max_Temp',
       'Avg_Min_Temp', 'Avg_Rain', 'Team1_Bat_Avg', 'Team1_Bat_Avg_SR',
       'Team1_Bowl_Avg_Econ', 'Team1_Bowl_Avg_Wkt', 'Team1_Bowl_Avg_SR',
       'Team2_Bat_Avg', 'Team2_Bat_Avg_SR', 'Team2_Bowl_Avg_Econ',
       'Team2_Bowl_Avg_Wkt', 'Team2_Bowl_Avg_SR', 'Team1_Odd_Predicted_RF',
       'Team2_Odd_Predicted_RF', 'Team1_ML_Predicted_RF',
       'Team2_ML_Predicted_RF', 'Team1and2_Spread_Predicted_RF',
       'Team1_Odd_Predicted_GB', 'Team2_Odd_Predicted_GB',
       'Team1_ML_Predicted_GB', 'Team2_ML_Predicted_GB',
       'Team1and2_Spread_Predicted_GB', 'Team1_Odd_Predicted_XGB',
       'Team2_Odd_Predicted_XGB', 'Team1_ML_Predicted_XGB',
       'Team2_ML_Predicted_XGB', 'Team1and2_Spread_Predicted_XGB',
       'Team1_Over_Under', 'Team1_Moneyline_Oddsportal',
```

```
             'Team2_Moneyline_Oddsportal', 'Team1_Spread_Oddsportal',
             'Team2_Spread_Oddsportal', 'Date_of_Game_Encoded', 'Month_Encoded',
             'Location_Encoded', 'Team1_Win_Loss_Status_Encoded',
             'Team2_Win_Loss_Status_Encoded', 'Format_Encoded', 'Winner_Encoded',
             'Team1_Encoded', 'Team2_Encoded', 'Location_Country_Encoded'],
            dtype='object')
```

```
# 'Team1', 'Team2', 'Date_of_Game', 'Month', 'Location', 'Location_Country', 'Team1_Win_Loss_Status', 'Team2_Win_Loss_Status',
# 'Team1 _Win_Loss_Ratio', 'Team2 _Win_Loss_Ratio', 'Format', 'Team1_Score', 'Team2_Score', 'Winner', 'Standardized_Wins_by_Runs_or_Wickets'
# 'Team1_Odd_Oddsportal', 'Team2_Odd_Oddsportal', 'Team1_Team2_Avg_Odd', 'Avg_Max_Temp', 'Avg_Min_Temp', 'Avg_Rain',
# 'Team1_Moneyline_Oddsportal', 'Team2_Moneyline_Oddsportal', 'Team1_Spread_Oddsportal', 'Team2_Spread_Oddsportal'

#drop = ["Team1", "Team2", "Date_of_Game", "Month", "Location", "Location_Country", "Team1_Win_Loss_Status", "Team2_Win_Loss_Status", "Forma
#drop_pca1 = ["Team1", "Team2", 'Date_of_Game', 'Month', 'Location', 'Location_Country', 'Team1_Win_Loss_Status', 'Team2_Win_Loss_Status', '
#drop_pca1 = ["Team1", "Team2", 'Date_of_Game', 'Month', 'Location', 'Location_Country', 'Team1_Win_Loss_Status', 'Team2_Win_Loss_Status']
#drop_pca2 = ['Format', 'Winner', 'Team1_Score', 'Team2_Score', 'Standardized_Wins_by_Runs_or_Wickets', 'Team1_Odd_Oddsportal', 'Team2_Odd_O
#drop_pca3 = ['Team1_Team2_Avg_Odd', 'Team1_Moneyline_Oddsportal', 'Team2_Moneyline_Oddsportal', 'Team1_Spread_Oddsportal', 'Team2_Spread_Od
#drop_encoded = ['Date_of_Game_Encoded', 'Team1_Win_Loss_Status_Encoded', 'Team2_Win_Loss_Status_Encoded', 'Format_Encoded', 'Location_Count

#drop_pca1 = ["Team1", "Team2", 'Date_of_Game', 'Month', 'Location', 'Location_Country', 'Team1_Win_Loss_Status', 'Team2_Win_Loss_Status', '
drop_pca1 = ["Team1", "Team2", 'Date_of_Game', 'Month', 'Location', 'Location_Country', 'Team1_Win_Loss_Status', 'Team2_Win_Loss_Status']

drop_pca2 = ['Format', 'Team1_Odd_Oddsportal', 'Team2_Odd_Oddsportal']
#drop_pca2 = ['Format', 'Team1_Score', 'Team2_Score', 'Standardized_Wins_by_Runs_or_Wickets', 'Team1_Odd_Oddsportal', 'Team2_Odd_Oddsportal'
#drop_pca2 = ['Format', 'Winner', 'Team1_Score', 'Team2_Score', 'Standardized_Wins_by_Runs_or_Wickets', 'Team1_Odd_Oddsportal', 'Team2_Odd_O

drop_pca3 = ['Team1_Team2_Avg_Odd', 'Team1_Moneyline_Oddsportal', 'Team2_Moneyline_Oddsportal', 'Team1_Spread_Oddsportal', 'Team2_Spread_Odd

drop_encoded = ['Date_of_Game_Encoded', 'Month_Encoded', 'Team1_Win_Loss_Status_Encoded', 'Team2_Win_Loss_Status_Encoded', 'Format_Encoded',
                'Location_Encoded', 'Location_Country_Encoded', 'Winner_Encoded']
#drop_encoded = ['Date_of_Game_Encoded', 'Team1_Win_Loss_Status_Encoded', 'Team2_Win_Loss_Status_Encoded', 'Format_Encoded', 'Location_Count

#drop_pred = ['Team1_Odd_Predicted_RF', 'Team2_Odd_Predicted_RF', 'Team1_Odd_Predicted_GB', 'Team2_Odd_Predicted_GB', 'Team1_Odd_Predicted_X
#       'Team2_Odd_Predicted_XGB', 'Team1_ML_Predicted_RF', 'Team2_ML_Predicted_RF', 'Team1_ML_Predicted_GB', 'Team2_ML_Predicted_GB',
#       'Team1_ML_Predicted_XGB', 'Team2_ML_Predicted_XGB', 'Team1_Spread_Predicted_RF', 'Team2_Spread_Predicted_RF',
#      'Team1_Spread_Predicted_GB', 'Team2_Spread_Predicted_GB', 'Team1_Spread_Predicted_XGB', 'Team2_Spread_Predicted_XGB']
drop_pred = ['Team1_Odd_Predicted_RF', 'Team2_Odd_Predicted_RF', 'Team1_ML_Predicted_RF', 'Team2_ML_Predicted_RF',
       'Team1and2_Spread_Predicted_RF', 'Team1_Odd_Predicted_GB', 'Team2_Odd_Predicted_GB', 'Team1_ML_Predicted_GB',
       'Team2_ML_Predicted_GB', 'Team1and2_Spread_Predicted_GB', 'Team1_Odd_Predicted_XGB', 'Team2_Odd_Predicted_XGB',
       'Team1_ML_Predicted_XGB', 'Team2_ML_Predicted_XGB', 'Team1and2_Spread_Predicted_XGB', 'Team1_Over_Under']

#drop_tmp = ['Team1_Win_Loss_Ratio', 'Team2_Win_Loss_Ratio',  'Team1_Score', 'Team2_Score', 'Standardized_Wins_by_Runs_or_Wickets']
drop_tmp = ['Team1_Score', 'Team2_Score', 'Standardized_Wins_by_Runs_or_Wickets']

#df = df.drop(drop, axis=1)
df = df.drop(drop_pca1, axis=1)
df = df.drop(drop_pca2, axis=1)
df = df.drop(drop_pca3, axis=1)
df = df.drop(drop_encoded, axis=1)
df = df.drop(drop_pred, axis=1)
df = df.drop(drop_tmp, axis=1)

y = df["Winner"]
drop = ['Winner']
df = df.drop(drop, axis=1)
```

```
#print(df)
#print(y)
#df.isnull()
```

```
# Standardize the data
#X = df.values
#print(len(X))
#X_std = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Perform PCA
#pca = PCA(n_components=15)
pcs = PCA()
pcs.fit(df)
pcs_sum = pd.DataFrame({'SD': np.sqrt(pcs.explained_variance_), 'Var Proportion': pcs.explained_variance_ratio_, 'Cumu Proportion': np.cumsu
pcs_sum = pcs_sum.transpose()
pcs_sum.columns = ['PC{}'.format(i) for i in range(1, len(pcs_sum.columns) + 1)]
pcs_sum.round(2)
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	I
SD	91.01	5.37	1.51	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Var Proportion	1.00	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

```
pcs_comp = pd.DataFrame(pcs.components_.transpose(), columns=pcs_sum.columns, index=df.columns)
#pcs_comp.iloc[:, :10]
pcs_comp
```

	PC1	PC2	PC3	PC4	
Team1 _Win_Loss_Ratio	4.336726e-19	4.519107e-20	-1.125107e-19	-6.183004e-03	-2.78443
Team2 _Win_Loss_Ratio	-3.330669e-16	1.734723e-16	-1.401657e-15	-1.995482e-03	9.99610
Avg_Max_Temp	1.147865e-02	6.841068e-01	7.292915e-01	5.542442e-16	8.12717
Avg_Min_Temp	-4.093821e-03	7.293656e-01	-6.841119e-01	-6.175616e-16	-1.07552
Avg_Rain	9.999257e-01	-4.867090e-03	-1.117274e-02	1.665335e-16	-1.11022
Team1_Bat_Avg	-0.000000e+00	8.673617e-18	-7.372575e-17	9.947085e-02	1.81297
Team1_Bat_Avg_SR	-0.000000e+00	6.938894e-17	-5.898060e-16	7.957668e-01	1.45038
Team1_Bowl_Avg_Econ	-0.000000e+00	-0.000000e+00	-0.000000e+00	-0.000000e+00	0.00000(
Team1_Bowl_Avg_Wkt	-0.000000e+00	3.469447e-17	-2.949030e-16	3.978834e-01	7.25190
Team1_Bowl_Avg_SR	-0.000000e+00	-0.000000e+00	-0.000000e+00	-0.000000e+00	0.00000(
Team2_Bat_Avg	-0.000000e+00	-0.000000e+00	-0.000000e+00	-0.000000e+00	0.00000(
Team2_Bat_Avg_SR	-0.000000e+00	-3.469447e-17	2.949030e-16	-3.978834e-01	-7.25190
Team2_Bowl_Avg_Econ	-0.000000e+00	2.168404e-18	-1.843144e-17	2.486771e-02	4.53244
Team2_Bowl_Avg_Wkt	-0.000000e+00	-0.000000e+00	-0.000000e+00	-0.000000e+00	0.00000(
Team2_Bowl_Avg_SR	-1.387779e-17	-2.688821e-17	1.986258e-16	-1.989417e-01	-3.62595
Team1_Encoded	-0.000000e+00	-0.000000e+00	0.000000e+00	-0.000000e+00	-0.00000(
Team2_Encoded	-0.000000e+00	-0.000000e+00	0.000000e+00	-0.000000e+00	-0.00000(

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Team1 _Win_Loss_Ratio  90 non-null     float64
 1   Team2 _Win_Loss_Ratio  90 non-null     float64
 2   Avg_Max_Temp           90 non-null     float64
 3   Avg_Min_Temp           90 non-null     float64
 4   Avg_Rain               90 non-null     float64
 5   Team1_Bat_Avg          90 non-null     float64
 6   Team1_Bat_Avg_SR       90 non-null     float64
 7   Team1_Bowl_Avg_Econ    90 non-null     float64
 8   Team1_Bowl_Avg_Wkt     90 non-null     float64
 9   Team1_Bowl_Avg_SR      90 non-null     float64
 10  Team2_Bat_Avg          90 non-null     float64
 11  Team2_Bat_Avg_SR       90 non-null     float64
 12  Team2_Bowl_Avg_Econ    90 non-null     float64
 13  Team2_Bowl_Avg_Wkt     90 non-null     float64
 14  Team2_Bowl_Avg_SR      90 non-null     float64
 15  Team1_Encoded          90 non-null     int64
 16  Team2_Encoded          90 non-null     int64
dtypes: float64(15), int64(2)
memory usage: 12.1 KB
```

```
df.sample(10)
#df.size
```

	Team1 _Win_Loss_Ratio	Team2 _Win_Loss_Ratio	Avg_Max_Temp	Avg_Min_Temp	Avg_Rain	Team1_Bat_Av
67	0.778	0.222	32.8	20.6	258.8	29.2162
23	0.778	0.222	36.9	27.0	4.6	29.2162
78	0.778	0.222	28.1	18.6	124.8	29.2162
45	0.778	0.222	28.4	24.6	10.6	29.2162
27	0.778	0.222	28.4	24.6	10.6	29.2162
35	0.778	0.222	28.1	18.6	124.8	29.2162
33	0.778	0.222	28.1	18.6	124.8	29.2162
24	0.778	0.222	28.1	18.6	124.8	29.2162
18	0.778	0.222	28.4	24.6	10.6	29.2162
75	0.778	0.222	34.8	27.6	249.6	29.2162

```
feat_arrange = ['Team1_Encoded', 'Team2_Encoded', 'Team1 _Win_Loss_Ratio',  'Team2 _Win_Loss_Ratio',  'Team1_Bat_Avg',  'Team1_Bat_Avg_SR',
  'Team1_Bowl_Avg_Econ',  'Team1_Bowl_Avg_Wkt',  'Team1_Bowl_Avg_SR',  'Team2_Bat_Avg',  'Team2_Bat_Avg_SR',  'Team2_Bowl_Avg_Econ',
  'Team2_Bowl_Avg_Wkt',  'Team2_Bowl_Avg_SR',  'Avg_Max_Temp',  'Avg_Min_Temp',  'Avg_Rain',]

# redo the columns
df = df.reindex(columns=feat_arrange)
df.columns
```

```
Index(['Team1_Encoded', 'Team2_Encoded', 'Team1 _Win_Loss_Ratio',
       'Team2 _Win_Loss_Ratio', 'Team1_Bat_Avg', 'Team1_Bat_Avg_SR',
       'Team1_Bowl_Avg_Econ', 'Team1_Bowl_Avg_Wkt', 'Team1_Bowl_Avg_SR',
       'Team2_Bat_Avg', 'Team2_Bat_Avg_SR', 'Team2_Bowl_Avg_Econ',
       'Team2_Bowl_Avg_Wkt', 'Team2_Bowl_Avg_SR', 'Avg_Max_Temp',
       'Avg_Min_Temp', 'Avg_Rain'],
      dtype='object')
```

A3 Data Modeling

A3i Team winning Odds

A3i.a Team winning Odds - Setup DV & IDV

to remove: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names

X = X.values y = y.values

After this do the train test split,

from sklearn.model_selection import train_test_split X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)

```
#print(y)
```

```
X = df
X = X.values
y = y.values

# encode y
y = encoder.fit_transform(y)
# didnt work got error: Invalid classes inferred from unique values of `y`.  Expected: [0 1 2 3 4 5 6 7], got [1 2 3 4 5 6 7 8]
# use this
#y_train_xgb = y_train.map({"1": 0, "2": 1, "3": 2, "4": 3, "5": 4, "6": 5, "7": 6, "8": 7})

#y = y.assign(**df[['Team1_Encoded', 'Team2_Encoded']])
# or use the following

#y1 = pd.DataFrame()
#y1["Team1_Odd_Oddsportal"] = df["Team1_Odd_Oddsportal"]
#drop = ['Team1_Odd_Oddsportal']
#df = df.drop(drop, axis=1)

#y2 = pd.DataFrame()
#y2["Team2_Odd_Oddsportal"] = df["Team2_Odd_Oddsportal"]
#drop = ['Team2_Odd_Oddsportal']
#df = df.drop(drop, axis=1)

# Split the data into training and testing sets
#x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.1,random_state=0)
#x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0)

# Scale the data using StandardScaler
scaler = StandardScaler()
#y_train_scaled = scaler.fit_transform(y_train)
#y_test_Scaled = scaler.fit_transform(y_test)
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
print(y)
```

```
[1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1
 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1
 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1]
```

```
#x_train
#y_train

#print(x_test)
#x_test.sample(5)
#x_test.columns
#x_test.info()

#print(y_test)
#y_test.info()
```

A3i.b Team winning Odds - Create Models

```
# Random Forest Classifier
#######################################################
#model_rf = RandomForestClassifier(n_estimators=100, min_samples_split=3, max_features="auto")
model_rf = RandomForestClassifier(n_estimators=100, min_samples_split=3, max_features=100)
#model_rf.fit(x_train, y_train.values)
model_rf.fit(x_train, y_train)
y_pred_rf = model_rf.predict(x_test)
# Predict
predictions = model_rf.predict(x_test)
probabilities = model_rf.predict_proba(x_test)[:, 1]
#ac_rf = accuracy_score(y_pred_rf, y_test)
#print("Random Forest Accuracy:", ac_rf)
# Evaluate
accuracy = accuracy_score(y_test, predictions)

# Output results
print(f"Model Accuracy: {accuracy}")
print("Winning Probabilities:", probabilities)

# Logistic Regression
#######################################################
model_lr = LogisticRegression(max_iter=1000, random_state=42)
model_lr.fit(x_train_scaled, y_train)
#model_lr.fit(x_train, y_train)
y_pred_lr = model_lr.predict(x_test_scaled)
#y_pred_lr = model_lr.predict(x_test)
ac_lr = accuracy_score(y_pred_lr, y_test)
print("Logistic Regression Accuracy:", ac_lr)

# Decision Tree Classifier
#######################################################
model_dt = DecisionTreeClassifier(random_state=42)
model_dt.fit(x_train, y_train)
y_pred_dt = model_dt.predict(x_test)
ac_dt = accuracy_score(y_pred_dt, y_test)
print("Decision Tree Accuracy:", ac_dt)

# Gradient Boosting Classifier
#######################################################
model_gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
model_gb.fit(x_train, y_train)
y_pred_gb = model_gb.predict(x_test)
ac_gb = accuracy_score(y_pred_gb, y_test)
print("Gradient Boosting Accuracy:", ac_gb)

# XGBoost Classifier
#######################################################
#start_time = time.time()
#model_xgb = XGBClassifier(random_state=42)
model_xgb = XGBClassifier(n_estimators = 400, learning_rate = 0.1, max_depth = 3)
model_xgb.fit(x_train, y_train)
#print('Fit time : ', time.time() - start_time)
y_pred_xgb = model_xgb.predict(x_test)
ac_xgb = accuracy_score(y_pred_xgb, y_test)
print("XGBoost Accuracy:", ac_xgb)

# Gradient Boosting Regressor
#######################################################
# Initialize and train the regressor
model_gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=0)
model_gbr.fit(x_train_scaled, y_train)
#model_gbr.fit(x_train, y_train.values.ravel())

# Predict probabilities on the test set
y_pred_gbr = model_gbr.predict(x_test_scaled)

# Calculate the R² score
r2 = r2_score(y_test, y_pred_gbr)

# Print the R² score
print(f'Gradient Boosting Regressor r-square: {r2:.2f}')

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred_gbr)
print(f"Gradient Boosting Regressor mse of test set: {mse}")
#print(y_pred)
#print(len(y_pred))
```

```
# Define the LSTM model
#####################################################
model_lstm = tf.keras.Sequential([
    layers.LSTM(64, return_sequences=True, input_shape=(x_train_scaled.shape[1], 1)),
    layers.LSTM(32),
    layers.Dense(1)
])

# Compile the model
#model_lstm.compile(loss='mean_squared_error', optimizer='adam')
model_lstm.compile('adam', 'binary_crossentropy', metrics='accuracy')

# Train the model
model_lstm.fit(x_train_scaled, y_train, epochs=10, batch_size=16)

# Evaluate the model on the test set
loss, mse = model_lstm.evaluate(x_test_scaled, y_test)
print('Mean Squared Error:', mse)

# Make predictions on the test set
y_pred_lstm = model_lstm.predict(x_test_scaled)

# Define the encoder and decoder
#####################################################
# Define the dimension of the input features and the encoding
input_dim = x_train_scaled.shape[1]  # Number of features
encoding_dim = 32  # This is the size of our encoded representations

# Input layer
input_layer = Input(shape=(input_dim,))

# Encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)

# Regression prediction layer based on encoded features
regression_output = Dense(1, activation='linear')(encoded)  # Output layer for regression

# Complete model
autoencoder = Model(input_layer, regression_output)

# Compile the model
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Summary of the model
autoencoder.summary()

# Training the model
history = autoencoder.fit(x_train_scaled, y_train,
                          epochs=50,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test_scaled, y_test))

# Make predictions on the test set
y_pred_enc = autoencoder.predict(x_test_scaled)

# Calculate the mean squared error (MSE)
mse = tf.keras.losses.MeanSquaredError()
mse_value = mse(y_test, y_pred_enc).numpy()

# Calculate the mean absolute error (MAE)
mae = tf.keras.losses.MeanAbsoluteError()
mae_value = mae(y_test, y_pred_enc).numpy()

# Calculate the root mean squared error (RMSE)
rmse_value = np.sqrt(mse_value)

# For R-squared, we can use the r2_score function from sklearn.metrics
#from sklearn.metrics import r2_score
r2_value = r2_score(y_test, y_pred_enc)

# Print the metrics
print(f'Auto Encoder Mean Squared Error (MSE): {mse_value}')
print(f'Auto Encoder Mean Absolute Error (MAE): {mae_value}')
print(f'Auto Encoder Root Mean Squared Error (RMSE): {rmse_value}')
```

```
print(f'Auto Encoder R-squared : {r2_value}')
```

```
Model Accuracy: 0.8148148148148148
Winning Probabilities: [1.        0.        0.6885144 1.        1.        1.        1.
 0.6885144 0.6885144 0.6885144 0.6885144 0.6885144 0.6885144 1.
 0.6885144 0.6885144 0.6885144 1.        1.        0.6885144 0.
 0.6885144 0.6885144 1.        0.6885144 1.        1.       ]
Logistic Regression Accuracy: 0.8148148148148148
Decision Tree Accuracy: 0.8148148148148148
Gradient Boosting Accuracy: 0.8148148148148148
XGBoost Accuracy: 0.8148148148148148
Gradient Boosting Regressor r-square: 0.37
Gradient Boosting Regressor mse of test set: 0.12008114910083255
Epoch 1/10
4/4 [==============================] - 8s 34ms/step - loss: 4.2365 - accuracy: 0.2063
Epoch 2/10
4/4 [==============================] - 0s 27ms/step - loss: 1.9851 - accuracy: 0.2063
Epoch 3/10
4/4 [==============================] - 0s 26ms/step - loss: 1.6803 - accuracy: 0.2063
Epoch 4/10
4/4 [==============================] - 0s 30ms/step - loss: 1.4837 - accuracy: 0.2063
Epoch 5/10
4/4 [==============================] - 0s 27ms/step - loss: 1.3291 - accuracy: 0.2063
Epoch 6/10
4/4 [==============================] - 0s 28ms/step - loss: 1.1800 - accuracy: 0.2063
Epoch 7/10
4/4 [==============================] - 0s 29ms/step - loss: 1.0467 - accuracy: 0.2063
Epoch 8/10
4/4 [==============================] - 0s 27ms/step - loss: 0.9019 - accuracy: 0.2063
Epoch 9/10
4/4 [==============================] - 0s 27ms/step - loss: 0.7537 - accuracy: 0.2540
Epoch 10/10
4/4 [==============================] - 0s 27ms/step - loss: 0.6135 - accuracy: 0.7937
1/1 [==============================] - 1s 1s/step - loss: 0.5734 - accuracy: 0.7407
Mean Squared Error: 0.7407407164573669
1/1 [==============================] - 2s 2s/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 17)]              0

 dense_1 (Dense)             (None, 32)                576

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 609 (2.38 KB)
Trainable params: 609 (2.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/50
1/1 [==============================] - 1s 723ms/step - loss: 0.5781 - val_loss: 0.5295
Epoch 2/50
1/1 [==============================] - 0s 36ms/step - loss: 0.5603 - val_loss: 0.5157
Epoch 3/50
1/1 [==============================] - 0s 41ms/step - loss: 0.5430 - val_loss: 0.5021
Epoch 4/50
1/1 [==============================] - 0s 41ms/step - loss: 0.5259 - val_loss: 0.4888
Epoch 5/50
```

A3i.c Team winning Odds - Caculate Odds & Moneyline:

```
pred_url_csv = "https://docs.google.com/spreadsheets/d/e/2PACX-1vRhaWdmjg3BKWlDKMgN2M4QrzGM20PGZ2_z0norCz4-LOhnLaN-OivLNjNYmmarcMmhhPFCiASin
x_team1_2 = pd.read_csv(pred_url_csv)
x_team1_2 = x_team1_2.values
print(x_team1_2)
```

```
[[ 0.778       0.222       24.8        17.2         0.3        29.21625
  82.43625     5.4225      40.535      43.995      27.72733333 92.17733333
   5.77230769 39.89076923 41.56384615  1.          3.        ]]
```

```
# American Odds
def probability_to_american_odds(probability):
    if probability < 0 or probability > 1:
        raise ValueError("Probability must be between 0 and 1.")

    if probability == 0.5:  # Edge case where the odds would be infinite
        return "EVEN"

    if probability > 0.5:
        odds = -100 / (probability - 1)
    else:
        odds = 100 / (1 - probability)

    # Formatting to return a nice string representation
    if odds > 0:
        return f'+{int(odds)}'
```

```
# Moneyline
def american_odds_to_payout(american_odds, bet_amount):
    if american_odds > 0:
        # Positive odds (e.g., +150)
        profit = (american_odds / 100) * bet_amount
    else:
        # Negative odds (e.g., -200)
        profit = bet_amount / (-american_odds / 100)

    # The total payout is the initial bet amount plus the profit
    #total_payout = bet_amount + profit
    total_payout = profit
    return total_payout
```

```
#x_team1_2 = np.array([x_team1_2])
#x_team1_2 = np.array([[18.3, 9.6, 61, 29.453333, 88.704667, 5.574545, 33.232727, 35.726364, 24.878750, 76.751250, 5.679167, 34.493333, 38.2
#x_team1_2 = np.array([[18.3, 9.6, 61, 29.453333, 88.704667, 5.574545, 33.232727, 35.726364, 24.878750, 76.751250, 5.679167, 34.493333, 38.2
#y_pred_rf1 = model_rf.predict([[1.125, 0.542, 78, 70, 10.3, 33.8, 28.4, 17.8, 27.72733333, 92.17733333, 5.772307692, 39.89076923, 41.563846
y_pred_rf1 = model_rf.predict_proba(x_team1_2)
y_pred_rf1
```

```
array([[0.0359206, 0.9640794]])
```

```
prob1_rf = y_pred_rf1[0][0]
#american_odds1_rf = probability_to_american_odds(prob1_rf)

prob2_rf = y_pred_rf1[0][1]
#american_odds2_rf = probability_to_american_odds(prob2_rf)

print(f"Random Forest - The American odds : {prob1_rf} | {prob2_rf}")
#print(f"Random Forest - The American odds {prob1_rf} | {prob2_rf} : {american_odds1_rf} | {american_odds2_rf}")
```

```
Random Forest - The American odds : 0.03592059750534982 | 0.9640794024946502
```

```
# probability to American Odds
american_odds1_rf = (100 * prob1_rf) / (1 - prob1_rf)
american_odds2_rf = (100 * prob2_rf) / (1 - prob2_rf)

# bookmakers odd
if (american_odds1_rf > american_odds2_rf):
  american_odds1_rf = -(american_odds1_rf)
  american_odds2_rf = american_odds2_rf + 200
else:
  american_odds2_rf = -(american_odds2_rf)
  american_odds1_rf = american_odds1_rf + 200

#print(f"Random Forest - The American odds  {prob1} | {prob2} : {american_odds1} | {american_odds2}")
print(f"Random Forest - The American odds  {prob1_rf} | {prob2_rf} : {round(int(american_odds1_rf))} | {round(int(american_odds2_rf))}")

# Moneyline
bet_amount = 100  # The amount you are betting
team1_payout_rf = american_odds_to_payout(american_odds1_rf, bet_amount)
team2_payout_rf = american_odds_to_payout(american_odds2_rf, bet_amount)
```