```
import numpy as np
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv("ucl_stats.csv")
```

UEFA Champions League Statistics(1993-2020) Analysis

Preprocessing

```
df.sample(10)
```

	year	team	match_played	wins	draws	losts	goals_scored	goals_conceded	g
703	2020	Borussia Dortmund	8	4	1	3	10	11	
583	2016	Gent	8	3	3	2	10	11	
613	2017	Club Brugge	6	0	0	6	2	14	-
178	2004	Lokomotiv Moscow	8	3	2	3	9	9	
368	2010	Maccabi Haifa	6	0	0	6	0	8	
554	2015	BATE Borisov	6	1	0	5	2	24	-
462	2013	Dynamo Kyiv	6	1	2	3	6	10	

```
df.head()
```

	year	team	match_played	wins	draws	losts	goals_scored	goals_conceded	gd	gr
0	1993	Marseille	6	3	3	0	14	4	10	
1	1993	Milan	7	6	0	1	11	2	9	
2	1993	Rangers	6	2	4	0	7	5	2	
3	1993	Club Brugge	6	2	1	3	5	8	-3	

```
df.tail()
```

	year	team	match_played	wins	draws	losts	goals_scored	goals_conceded	gd
709	2020	Zenit Saint Petersburg	6	2	1	3	7	9	-2
710	2020	Valencia	8	3	2	3	13	15	-2
711	2020	Chelsea	8	3	2	3	12	16	-4
712	2020	Ajax	6	3	1	2	12	6	6

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 714 entries, 0 to 713
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   year            714 non-null    int64
```

```
 1   team            714 non-null    object
 2   match_played    714 non-null    int64
 3   wins            714 non-null    int64
 4   draws           714 non-null    int64
 5   losts           714 non-null    int64
 6   goals_scored    714 non-null    int64
 7   goals_conceded  714 non-null    int64
 8   gd              714 non-null    int64
 9   group_point     714 non-null    int64
 10  champions       714 non-null    int64
dtypes: int64(10), object(1)
memory usage: 61.5+ KB
```

```
df.dtypes
```

```
year               int64
team              object
match_played       int64
wins               int64
draws              int64
losts              int64
goals_scored       int64
goals_conceded     int64
gd                 int64
group_point        int64
champions          int64
dtype: object
```

```
df.shape
```

```
(714, 11)
```

```
print(df.isnull().sum())
```

```
year              0
team              0
match_played      0
wins              0
draws             0
losts             0
goals_scored      0
goals_conceded    0
gd                0
group_point       0
champions         0
dtype: int64
```

```
print(f"\nNumber of duplicates = {df.duplicated().sum()}")
```

```

Number of duplicates = 0
```

data statistics

```
df.describe()
```

	year	match_played	wins	draws	losts	goals_scored	goals_
count	714.000000	714.000000	714.000000	714.000000	714.000000	714.000000	7
mean	2008.711485	7.673669	2.939776	1.844538	2.892157	10.582633	
std	7.340121	2.100880	2.316600	1.273169	1.234552	6.979194	
min	1993.000000	6.000000	0.000000	0.000000	0.000000	0.000000	
25%	2004.000000	6.000000	1.000000	1.000000	2.000000	5.000000	
50%	2009.000000	6.000000	2.000000	2.000000	3.000000	9.000000	
75%	2015.000000	8.000000	4.000000	3.000000	4.000000	14.000000	
max	2020.000000	13.000000	11.000000	7.000000	6.000000	43.000000	

variance

```
df.var()
```

```
<ipython-input-18-28ded241fd7c>:1: FutureWarning: The default value of numeric_only in DataFrame.var is deprecated. In a future version,
  df.var()
year              53.877371
match_played       4.413696
wins               5.366634
draws              1.620959
losts              1.524118
goals_scored      48.709151
goals_conceded    13.206535
gd                57.360111
group_point       17.580596
champions          0.037731
dtype: float64
```

standard deviation

```
df.std()
```

```
<ipython-input-19-ce97bb7eaef8>:1: FutureWarning: The default value of numeric_only in DataFrame.std is deprecated. In a future version,
  df.std()
year              7.340121
match_played      2.100880
wins              2.316600
draws             1.273169
losts             1.234552
goals_scored      6.979194
goals_conceded    3.634080
gd                7.573646
group_point       4.192922
champions         0.194244
dtype: float64
```

skewness

```
df.skew()
```

```
<ipython-input-20-9e0b1e29546f>:1: FutureWarning: The default value of numeric_only in DataFrame.skew is deprecated. In a future version
  df.skew()
year             -0.292574
match_played      1.067313
wins              0.864137
draws             0.690167
losts             0.192457
goals_scored      1.250151
goals_conceded    0.462985
gd                0.572069
group_point      -0.083664
champions         4.757718
dtype: float64
```

kurtosis

```
df.kurt()
```

```
<ipython-input-21-8bd0d54cd88d>:1: FutureWarning: The default value of numeric_only in DataFrame.kurt is deprecated. In a future version
  df.kurt()
year              -0.916990
match_played       0.055671
wins               0.285389
draws              0.780566
losts             -0.125295
goals_scored       1.879041
goals_conceded     0.242339
gd                 1.069469
group_point       -0.570018
```

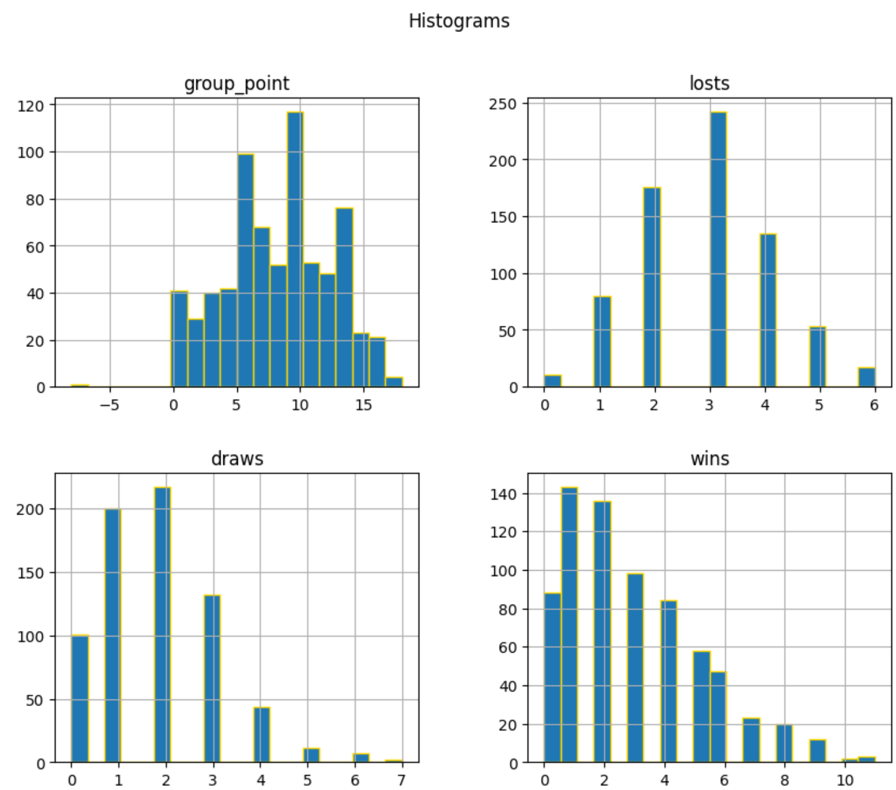
```
champions        20.693839
dtype: float64
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

Data Visualization

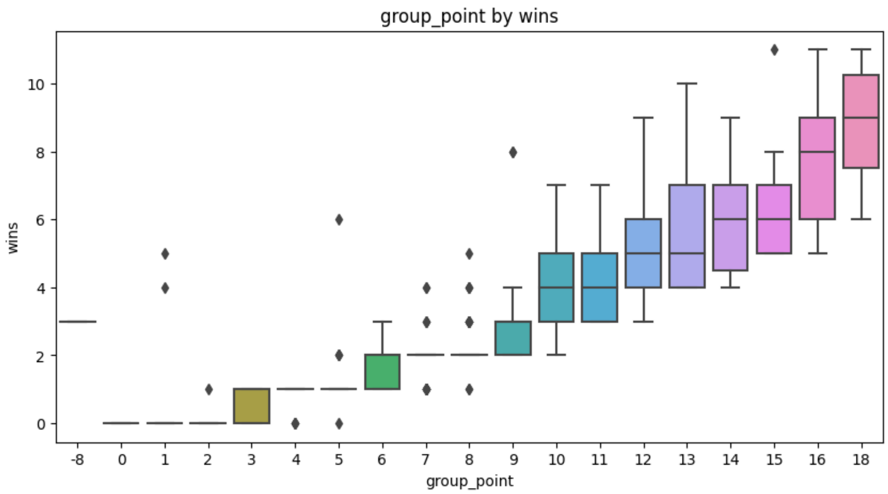
histogram for some columns

```
df[['group_point','losts', 'draws', 'wins']].hist(bins=20, figsize=(10, 8), edgecolor='gold')
plt.suptitle('Histograms')
plt.show()
```

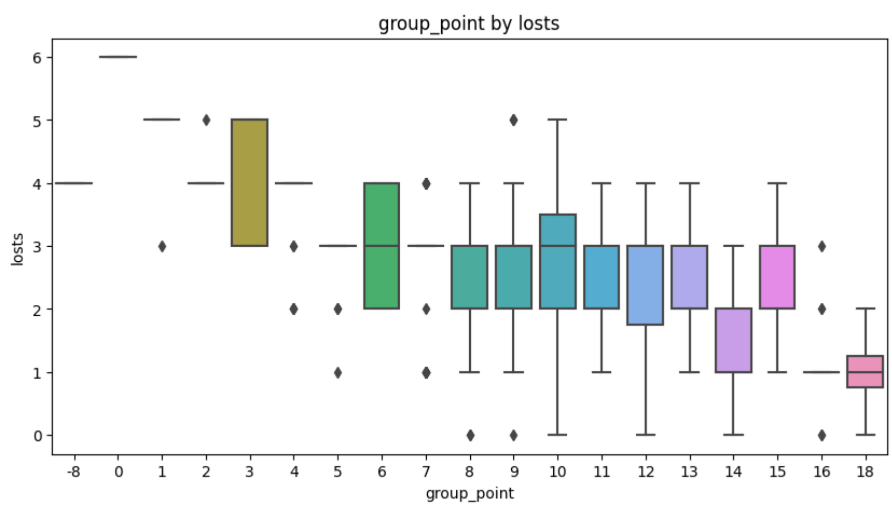


Boxplot for group_point by wins

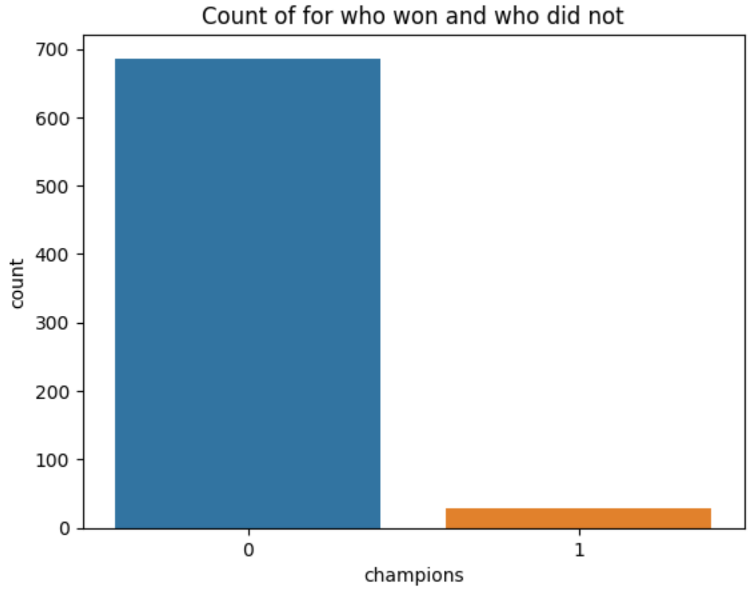
```
plt.figure(figsize=(10, 5))
sns.boxplot(x='group_point', y='wins', data=df)
plt.title('group_point by wins')
plt.show()
```



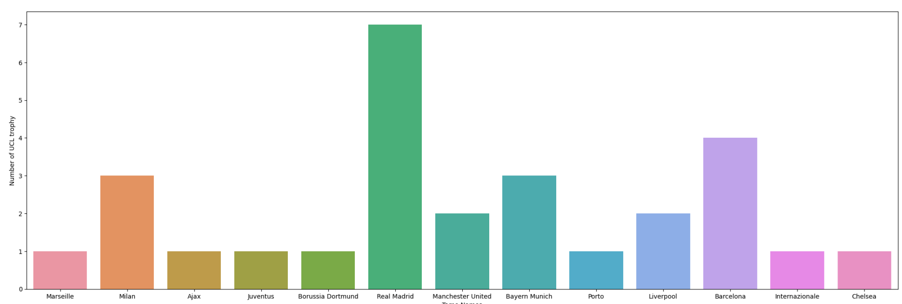

```
plt.figure(figsize=(10, 5))
sns.boxplot(x='group_point', y='losts', data=df)
plt.title('group_point by losts')
plt.show()
```




```
# Countplot of who won and who did not
sns.countplot(x='champions', data=df)
plt.title('Count of for who won and who did not')
plt.show()
```

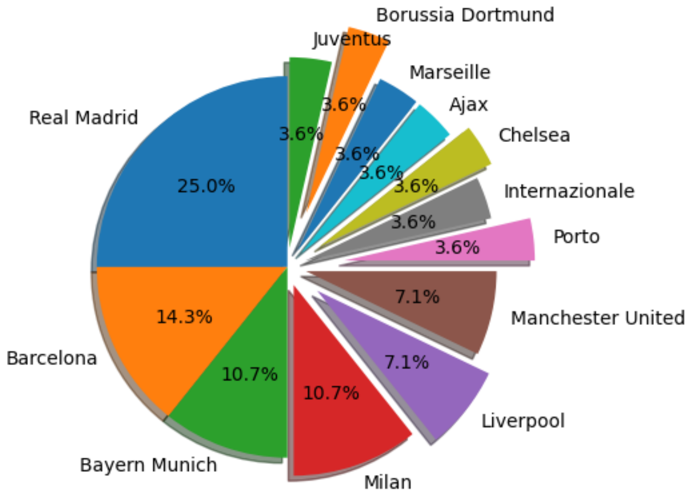



```
plt.figure(figsize=(25,8))
sns.countplot(x = df.team[df.champions == 1], data = df)
plt.xlabel('Tame Names')
plt.ylabel('Number of UCL trophy');
```

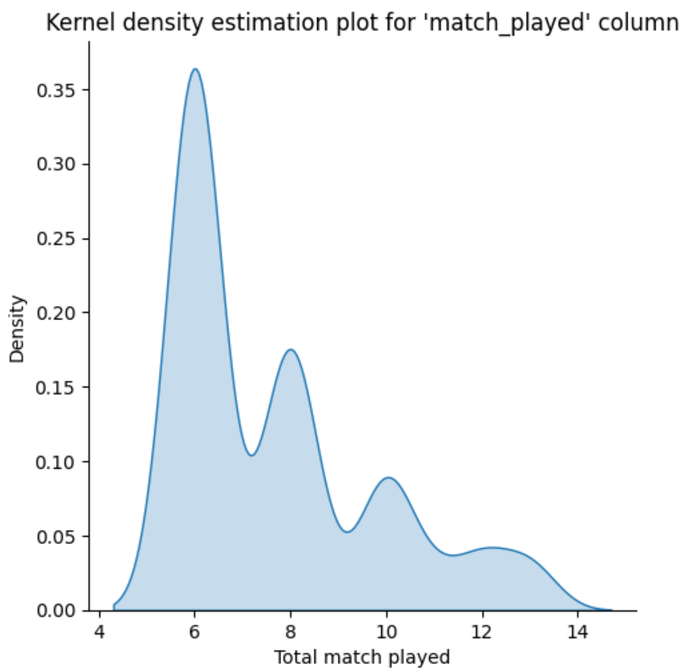


visualizing in Pie chart

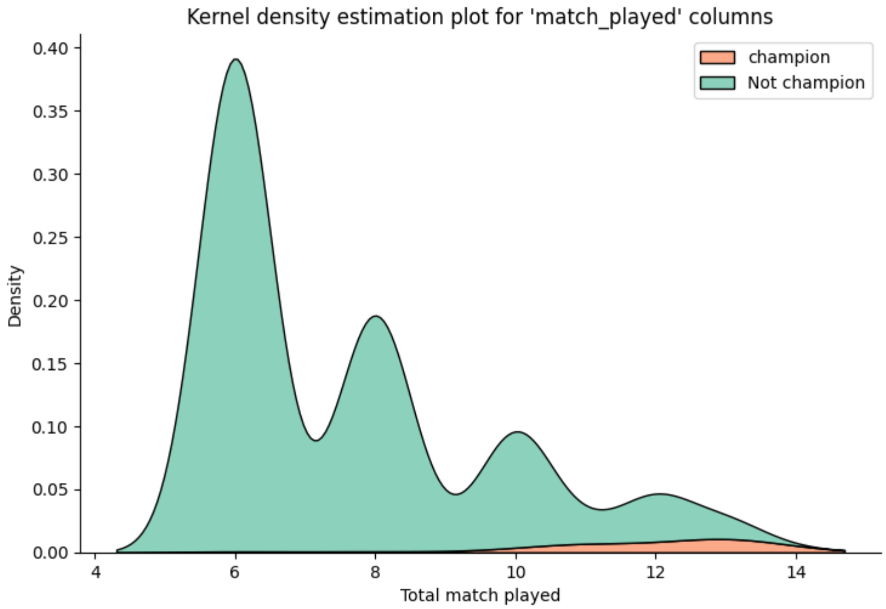
```
labels = 'Real Madrid', 'Barcelona', 'Bayern Munich', 'Milan', 'Liverpool', 'Manchester United', 'Porto', 'Internazionale', 'Chelsea', 'Ajax
explode = (0, 0, 0, 0.1, 0.2, 0.1, 0.3, 0.1, 0.2, 0.1, 0.1, 0.3, 0.1)

fig1, ax1 = plt.subplots()
ax1.pie(df.team[df.champions == 1].value_counts(), explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
```



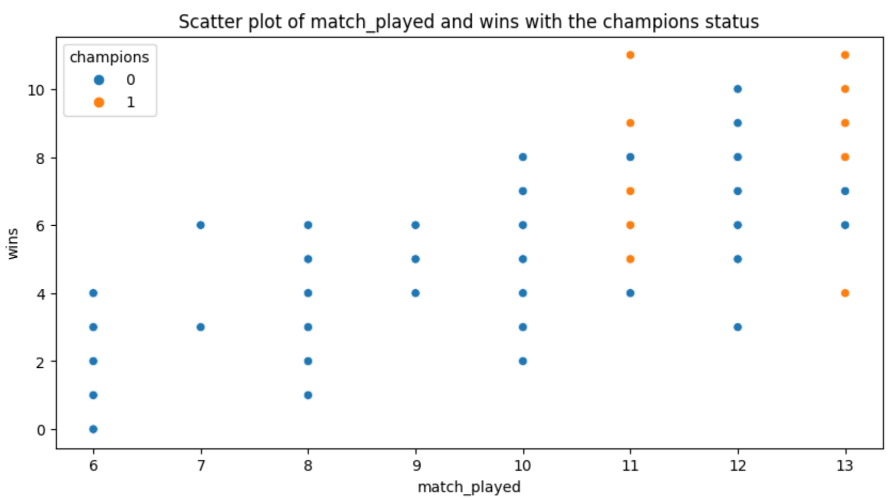

```
#Kernel density estimation plot for 'match_played' column
sns.displot(x = 'match_played', kind = 'kde', data = df, fill=True)
plt.title("Kernel density estimation plot for 'match_played' column")
plt.xlabel("Total match played");
```



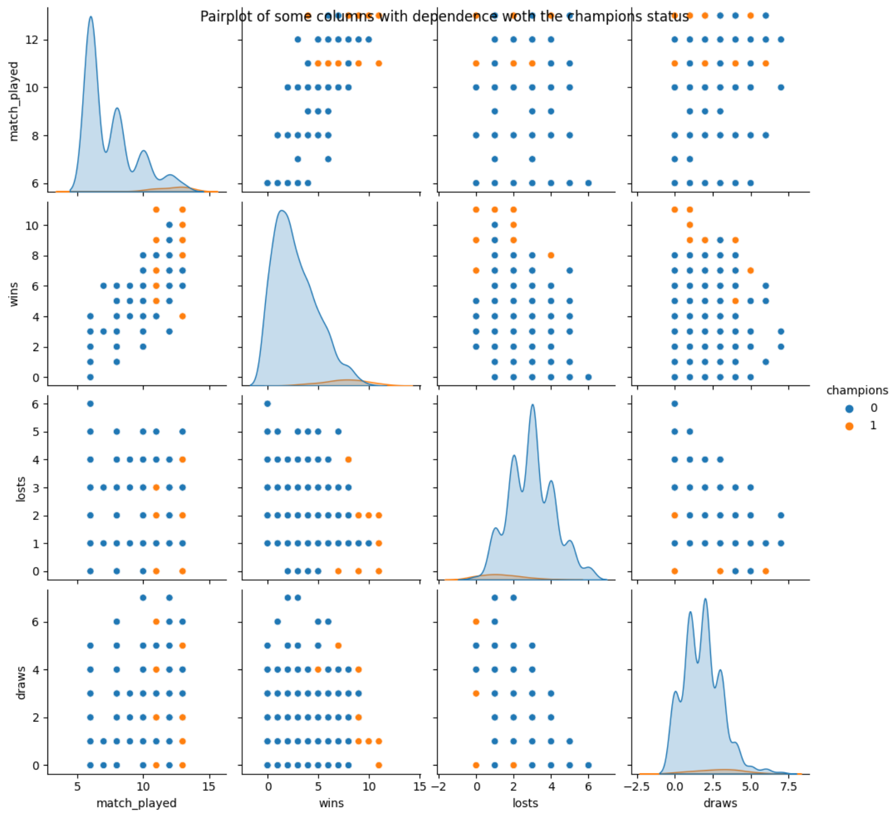

```
#Kernel density estimation plot for 'match_played' column w.r.t champions and not champions
sns.displot(x = 'match_played', hue = 'champions', kind = 'kde', data = df, legend = False, multiple="stack", aspect=1.5, palette = 'Set2')
plt.title("Kernel density estimation plot for 'match_played' columns")
plt.xlabel("Total match played")
plt.legend(['champion', 'Not champion']);
```




```
# Scatter plot for Age and Fare
plt.figure(figsize=(10, 5))
sns.scatterplot(x='match_played', y='wins', hue='champions', data=df)
plt.title('Scatter plot of match_played and wins with the champions status')
plt.show()
```




```
# Pairplot for number of columns
sns.pairplot(df[['match_played', 'wins', 'losts', 'draws', 'champions']], hue='champions')
plt.suptitle('Pairplot of some columns with dependence woth the champions status')
plt.show()
```



covariance matrix

```
df.cov()
```

```
<ipython-input-31-6f98a29763d5>:1: FutureWarning: The default value of numeric_only in D
  df.cov()
```

	year	match_played	wins	draws	losts	goals_scored	go
year	53.877371	0.923949	0.835335	-0.743373	0.772489	6.074365	
match_played	0.923949	4.413696	4.291680	0.949197	-0.861329	12.203020	
wins	0.835335	4.291680	5.366634	0.328633	-1.404821	14.252529	
draws	-0.743373	0.949197	0.328633	1.620959	-0.983128	1.891546	
losts	0.772489	-0.861329	-1.404821	-0.983128	1.524118	-3.994582	
goals_scored	6.074365	12.203020	14.252529	1.891546	-3.994582	48.709151	
goals_conceded	5.901065	0.354063	-1.165370	-0.849535	2.360710	1.921887	
gd	0.194908	11.746618	15.265851	2.719601	-6.287078	46.149041	
group_point	1.435898	6.177769	8.400967	1.132350	-3.364478	22.367006	
champions	-0.086847	0.162885	0.181888	0.038363	-0.061684	0.538129	

correlation_matrix

```
df.corr()
```

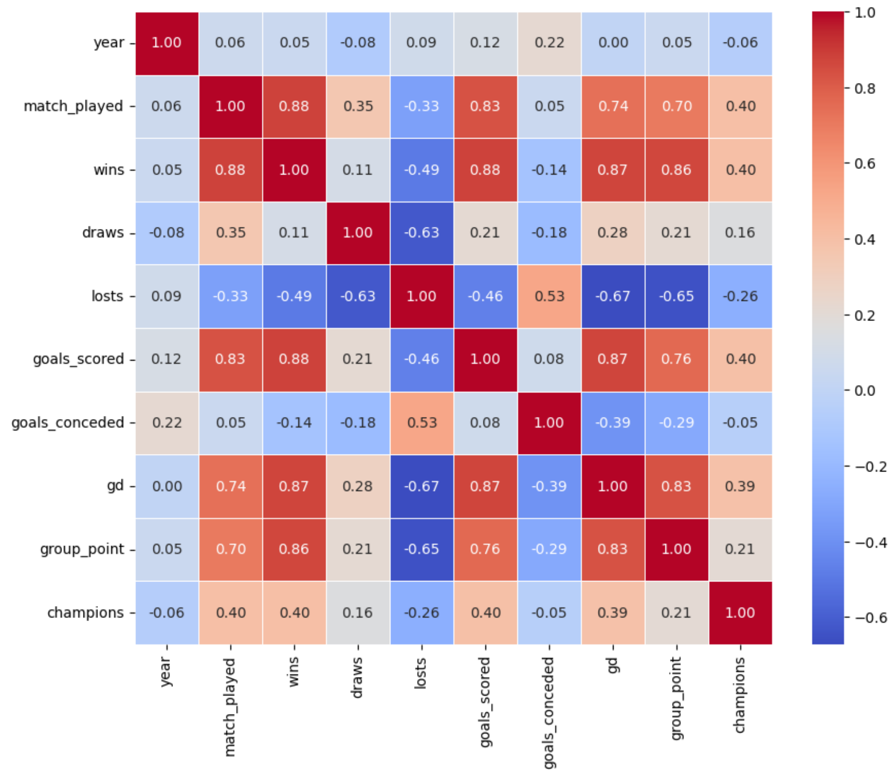
```
<ipython-input-32-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in D
  df.corr()
```

	year	match_played	wins	draws	losts	goals_scored	goa
year	1.000000	0.059916	0.049125	-0.079546	0.085247	0.118575	
match_played	0.059916	1.000000	0.881810	0.354870	-0.332092	0.832264	
wins	0.049125	0.881810	1.000000	0.111423	-0.491203	0.881527	
draws	-0.079546	0.354870	0.111423	1.000000	-0.625482	0.212875	
losts	0.085247	-0.332092	-0.491203	-0.625482	1.000000	-0.463614	
goals_scored	0.118575	0.832264	0.881527	0.212875	-0.463614	1.000000	
goals_conceded	0.221224	0.046375	-0.138426	-0.183612	0.526186	0.075775	
gd	0.003506	0.738255	0.870092	0.282042	-0.672411	0.873077	
group_point	0.046656	0.701316	0.864891	0.212118	-0.649967	0.764339	
champions	-0.060912	0.399148	0.404209	0.155125	-0.257225	0.396949	

A Heatmap for the correlation

```
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.show()
```

```
<ipython-input-33-a433e2e6d85e>:1: FutureWarning: The default value of numeric_only in D
  correlation_matrix = df.corr()
```



Chi-square Test

```
from scipy.stats import chi2_contingency
chi2, p_value, _, _ = chi2_contingency(pd.crosstab(df['champions'], df['wins']))
print(f"Chi-square Value: {chi2}")
print(f"P-value: {p_value}")
```

```
Chi-square Value: 246.23571266552426
P-value: 1.714813021055084e-46
```

Z-test

```
from statsmodels.stats.weightstats import ztest
z_stat, p_value = ztest(df['wins'][df['champions'] == 1],
df['wins'][df['group_point'] > 6])
print(f"Z-Test score = {z_stat}")
print(f"P-value = {p_value}")
```

```
Z-Test score = 8.742899014388236
P-value = 2.272037546972689e-18
```

```
for c in df.columns:
    print(f"unique values in the {c} column: {df[c].unique()} \n")
```

```
unique values in the year column: [1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020]

unique values in the team column: ['Marseille' 'Milan' 'Rangers' 'Club Brugge' 'CSKA Moscow' 'IFK Goteborg'
 ' Porto' 'PSV Eindhoven' ' Spartak Moscow' ' Galatasaray' 'Werder Bremen'
```

```
 'Anderlecht' 'Porto' 'Monaco' 'Barcelona' 'Manchester United'
 'Galatasaray' 'Spartak Moscow' 'Dynamo Kyiv' 'Steaua Bucure?ti'
 'Casino Salzburg' 'AEK Athens' 'Paris Saint-Germain' 'Bayern Munich'
 ' Benfica' 'Hajduk Split' 'Ajax' 'Panathinaikos' 'Nantes' 'Aalborg BK'
 'Legia Warsaw' 'Rosenborg' 'Blackburn Rovers' 'Juventus'
 'Borussia Dortmund' 'Real Madrid' 'Ferencvaros' 'Grasshopper' 'Auxerre'
 'Atletico Madrid' 'Rapid Wien' 'Parma' 'Sparta Prague' 'Feyenoord'
 'Newcastle United' 'Olympiacos' 'Besiktas' 'Bayer Leverkusen'
 'Sporting CP' 'Lierse' 'Croatia Zagreb' ' Juventus' 'Athletic Bilbao'
 'Internazionale' 'Sturm Graz' 'Arsenal' 'Lens' 'Kaiserslautern' 'Benfica'
 'HJK' 'Hertha BSC' 'Valencia' 'Fiorentina' 'Bordeaux' 'Lazio' 'Chelsea'
 'Deportivo La Coruna' 'Lyon' 'Leeds United' 'Boavista' 'Liverpool' 'Roma'
 'Lokomotiv Moscow' 'Basel' 'Celtic' 'Real Sociedad' 'Stuttgart'
 'Partizan' 'Celta Vigo' 'Maccabi Tel Aviv' 'Fenerbahce'
 'Shakhtar Donetsk' 'Thun' 'Udinese' 'Villarreal' 'Lille' 'Schalke 04'
 'Fenerbahae' 'Real Betis' 'Artmedia' 'Levski Sofia' 'Copenhagen'
 'Hamburg' 'Betiktas' 'Sevilla' 'Slavia Prague' 'CFR Cluj' 'Anorthosis'
 'PSV' 'Zenit Saint Petersburg' 'BATE Borisov' 'Maccabi Haifa' 'Wolfsburg'
 'Zurich' 'APOEL' 'Debrecen' 'Rubin Kazan' 'Unirea Urziceni'
 'Standard Liege' 'AZ' 'Tottenham Hotspur' 'Twente' 'Hapoel Tel Aviv'
 'Bursaspor' 'Braga' 'Napoli' 'Manchester City' 'Trabzonspor'
 'O?elul Gala?i' 'Dinamo Zagreb' 'Genk' 'Viktoria Plze?' 'Montpellier'
 'Malaga' 'Nordsjaelland' 'Austria Wien' 'Ludogorets Razgrad' 'Maribor'
 'Astana' 'Borussia Monchengladbach' 'Gent' 'Be?ikta?' 'Rostov'
 'Leicester City' 'Qaraba?' 'RB Leipzig' 'Inter Milan' 'Red Star Belgrade'
 '1899 Hoffenheim' 'Young Boys' 'Atalanta' 'Red Bull Salzburg']

unique values in the match_played column: [ 6  7  8 10 11 13 12  9]

unique values in the wins column: [ 3  6  2  0  1  5  4  7  8  9 10 11]

unique values in the draws column: [3 0 4 1 2 5 7 6]

unique values in the losts column: [0 1 3 4 5 2 6]

unique values in the goals_scored column: [14 11  7  5  2  4  6  1 10  9 16 13  3  8 12 15 18 17 22 23 21 20 29 26
 19 27  0 24 32 25 30 35 31 41 33 28 36 43]

unique values in the goals_conceded column: [ 4  2  5  8 11 13 12 10 15  9  7  6 14 16  3 19 17 20 21 22 18 24]

unique values in the gd column: [ 10   9   2  -3  -9  -1   0  -6  -4   1  11  -2   3  -5  14   5  -7  13
  -8  19 -10   7   4   8  17   6 -14  12 -12 -11 -13 -16 -15  15  21  24
 -19  25 -18  20  31  16  18 -22  22  35 -17]

unique values in the group_point column: [ 9 12  8  5  2  6  1  4  7 10  3 11 18 13 16 15  0 14 -8]

unique values in the champions column: [1 0]
```

ANOVA

```
from scipy.stats import f_oneway
anova = f_oneway(df['wins'][df['losts'] == 0],
df['wins'][df['losts'] == 1],
df['wins'][df['losts'] == 2],
df['wins'][df['losts'] == 3],
df['wins'][df['losts'] == 4],
df['wins'][df['losts'] == 5],
df['wins'][df['losts'] == 6],)
print(anova)
```

```
F_onewayResult(statistic=39.53230520881937, pvalue=1.55760090585177e-41)
```

Feature Reduction

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
X = df[['wins', 'losts', 'draws', 'match_played', 'group_point', 'goals_scored', 'goals_conceded']]
y = df['champions']
lda = LinearDiscriminantAnalysis(n_components = 1)
X_lda = lda.fit_transform(X, y)
print("LDA features: ")
print(X_lda)
```

```
 [-4.87485562e-01]
 [-6.87039734e-01]
 [ 4.03658437e+00]
 [-3.37194150e-01]
 [-4.98380160e-01]
 [ 1.09114810e+00]
 [ 1.94876419e+00]
 [-3.30793621e-01]
 [-5.61678038e-01]
 [ 2.03471737e-01]
 [ 1.46473690e-02]
 [-3.20822426e-01]
 [-7.04871690e-01]
 [ 1.42402380e+00]
 [-1.02952632e+00]
 [-7.74882480e-01]
 [-4.76561137e-01]
 [ 3.10190926e-02]
 [-1.16356246e-01]
 [-1.27474163e-01]
 [-3.14079688e-01]
 [ 2.43652938e+00]
 [-1.90493080e-02]
 [-2.28039384e-01]
 [-5.57496352e-01]
 [-7.13656704e-01]
 [-3.67599453e-01]
 [-6.27507142e-01]
 [-2.86783540e-01]
 [ 2.09323304e+00]
 [ 1.19364450e-01]
 [-5.43343471e-01]
 [-4.92932861e-01]
 [-4.48011452e+00]
 [ 2.07042402e-01]
 [-9.47307159e-02]
 [-3.01672250e-02]
 [ 2.12632192e+00]
 [ 8.24684014e-03]
 [-6.38431566e-01]
 [-5.95716925e-01]
 [-2.06193346e-01]
 [-1.42354678e+00]
 [-4.05939327e-01]
 [-4.49607199e-01]
 [-5.79150581e-01]
 [ 3.18839190e-01]
 [-4.00462201e-01]
 [-7.39577981e-01]
 [-1.47353946e+00]
 [-3.91860638e-01]
 [-9.47307159e-02]
 [-3.25004113e-01]
 [ 5.33398678e-01]
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
X = df[['wins', 'losts', 'draws', 'match_played', 'group_point', 'goals_scored', 'goals_conceded']]
X_standardized = StandardScaler().fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_standardized)
print(pca.explained_variance_ratio_)
print(X_pca)
```

```
[0.56872352 0.22055848]
[[ 1.23535449 -2.6931855 ]
 [ 1.44566503 -1.31961561]
 [ 0.61641105 -3.24101755]
 ...
 [ 0.28100677  1.04329647]
 [ 0.22424873 -0.96298721]
 [-3.01674461  0.94946519]]
```

Singular Value Decomposition (SVD)

```
from sklearn.decomposition import TruncatedSVD
# Perform Singular Value Decomposition (SVD)
svd = TruncatedSVD(n_components=2)
X_svd = svd.fit_transform(X)

# Display the reduced feature sets
print("Singular Value Decomposition (SVD):")
print(X_svd)
```

```
Singular Value Decomposition (SVD):
[[ 17.41761474   6.24879473]
 [ 16.66604294   7.48663721]
 [ 13.18217412   1.66498246]
 ...
 [ 24.11139917  -4.18152939]
 [ 17.71411056   3.54868532]
 [ 12.846651   -10.26297877]]
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD

# Split the data into training and testing sets
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2, random_state=42)
X_train_lda, X_test_lda, y_train_lda, y_test_lda = train_test_split(X_lda, y, test_size=0.2, random_state=42)
X_train_svd, X_test_svd, y_train_svd, y_test_svd = train_test_split(X_svd, y, test_size=0.2, random_state=42)

# Train and predict using Logistic Regression on PCA reduced features
pca_model = LogisticRegression()
pca_model.fit(X_train_pca, y_train_pca)
pca_predictions = pca_model.predict(X_test_pca)
pca_accuracy = accuracy_score(y_test_pca, pca_predictions)

# Train and predict using Logistic Regression on LDA reduced features
lda_model = LogisticRegression()
lda_model.fit(X_train_lda, y_train_lda)
lda_predictions = lda_model.predict(X_test_lda)
lda_accuracy = accuracy_score(y_test_lda, lda_predictions)

# Train and predict using Logistic Regression on SVD reduced features
svd_model = LogisticRegression()
svd_model.fit(X_train_svd, y_train_svd)
svd_predictions = svd_model.predict(X_test_svd)
svd_accuracy = accuracy_score(y_test_svd, svd_predictions)

# Display accuracies
print("Accuracy of PCA: {:.2f}%".format(pca_accuracy * 100))
print("Accuracy of LDA: {:.2f}%".format(lda_accuracy * 100))
print("Accuracy of SVD: {:.2f}%".format(svd_accuracy * 100))
```

```
Accuracy of PCA: 95.80%
Accuracy of LDA: 96.50%
Accuracy of SVD: 94.41%
```

Naive Bayesian

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
X = df[['wins', 'losts', 'draws', 'match_played', 'group_point', 'goals_scored', 'goals_conceded']]
y = df['champions']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4)
NBmodel = GaussianNB()
NBmodel.fit(X_train, y_train)
NBy_pred = NBmodel.predict(X_test)
print("accuracy: ", accuracy_score(y_test, NBy_pred))
print("classification report:", classification_report(y_test, NBy_pred))
```

```
accuracy:  0.9055944055944056
classification report:               precision    recall  f1-score   support
```

```
           0       0.99      0.91      0.95       275
           1       0.26      0.82      0.40        11

    accuracy                           0.91       286
   macro avg       0.63      0.86      0.67       286
weighted avg       0.96      0.91      0.93       286
```

```
!pip install pgmpy
```

```
Collecting pgmpy
  Downloading pgmpy-0.1.24-py3-none-any.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 6.4 MB/s eta 0:00:00
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.2.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.5.3)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.1.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.1.0+cu121)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.14.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pgmpy) (4.66.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.3.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2023.3.post1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pgmpy) (3.2.0)
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (0.5.4)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (23.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.4->statsmodels->pgmpy) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->pgmpy) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->pgmpy) (1.3.0)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.24
```

```
df.dtypes
```

```
year               int64
team              object
match_played       int64
wins               int64
draws              int64
losts              int64
goals_scored       int64
goals_conceded     int64
gd                 int64
group_point        int64
champions          int64
dtype: object
```

Bayesian Belief Network

```
pip install pgmpy
```

```
Collecting pgmpy
  Downloading pgmpy-0.1.24-py3-none-any.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 15.2 MB/s eta 0:00:00
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.2.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.5.3)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.1.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.1.0+cu121)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.14.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pgmpy) (4.66.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.3.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2023.3.post1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pgmpy) (3.2.0)
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (0.5.4)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (23.2)
```

```
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.4->statsmodels->pgmpy) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->pgmpy) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->pgmpy) (1.3.0)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.24
```

Decision Tree

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
X = df[['wins', 'losts', 'draws', 'match_played', 'group_point', 'goals_scored', 'goals_conceded']]
y = df['champions']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
DTmodel = DecisionTreeClassifier(criterion='entropy')
DTmodel.fit(X_train, y_train)
DTy_pred = DTmodel.predict(X_test)
print("DT accuracy = ", accuracy_score(y_test, DTy_pred))
print("\nthe classification report:\n", classification_report(y_test, DTy_pred))
```

```
DT accuracy =  0.9475524475524476

the classification report:
               precision    recall  f1-score   support

           0       0.99      0.96      0.97       276
           1       0.37      0.70      0.48        10

    accuracy                           0.95       286
   macro avg       0.68      0.83      0.73       286
weighted avg       0.97      0.95      0.96       286
```

LDA accuracy

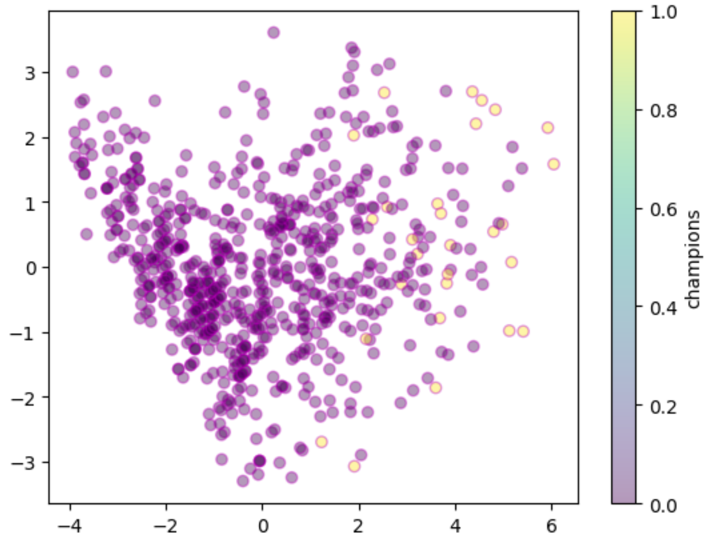
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
LDAy_pred = lda.predict(X_test)
print("LDA accuracy:", accuracy_score(y_test, LDAy_pred))
print("\nthe classification report:\n", classification_report(y_test, LDAy_pred))
```

```
LDA accuracy: 0.965034965034965

the classification report:
               precision    recall  f1-score   support

           0       0.99      0.97      0.98       276
           1       0.50      0.80      0.62        10

    accuracy                           0.97       286
   macro avg       0.75      0.89      0.80       286
weighted avg       0.98      0.97      0.97       286
```

variance ratio

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)
pca = PCA(n_components=X_standardized.shape[1])
X_pca = pca.fit_transform(X_standardized)
print("variance ratio:", pca.explained_variance_ratio_)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['champions'], cmap = 'viridis', edgecolor = 'm', alpha = 0.4)
plt.colorbar(label='champions')
plt.show()
```

```
variance ratio: [5.68723519e-01 2.20558478e-01 1.29781273e-01 4.78700186e-02
 2.14258318e-02 1.11173206e-02 5.23558972e-04]
```



KNN with manhattan

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 4, metric = 'manhattan')
knn.fit(X_train, y_train)
KNNy_pred = knn.predict(X_test)
print("KNN accuracy = ", accuracy_score(y_test, KNNy_pred))
print("\nthe classification report:\n", classification_report(y_test, KNNy_pred))
```

```
KNN accuracy =  0.9615384615384616

the classification report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98       276
           1       0.00      0.00      0.00        10

    accuracy                           0.96       286
   macro avg       0.48      0.50      0.49       286
weighted avg       0.93      0.96      0.95       286
```

KNN with euclidean

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2, metric = 'euclidean')
knn.fit(X_train, y_train)
KNNy_pred = knn.predict(X_test)
print("KNN accuracy = ", accuracy_score(y_test, KNNy_pred))
print("\nthe classification report:\n", classification_report(y_test, KNNy_pred))
```

```
KNN accuracy =  0.9615384615384616

the classification report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98       276
           1       0.00      0.00      0.00        10

    accuracy                           0.96       286
   macro avg       0.48      0.50      0.49       286
weighted avg       0.93      0.96      0.95       286
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
F1_Score = f1_score(y_test, y_pred)
conMat = confusion_matrix(y_test, y_pred)
print("accuracy = ", accuracy)
print("\nprecision = ", precision)
print("\nrecall = ", recall)
print("\nF1 Score = ", F1_Score)
print("\nconfusion matrix:\n", conMat)
```

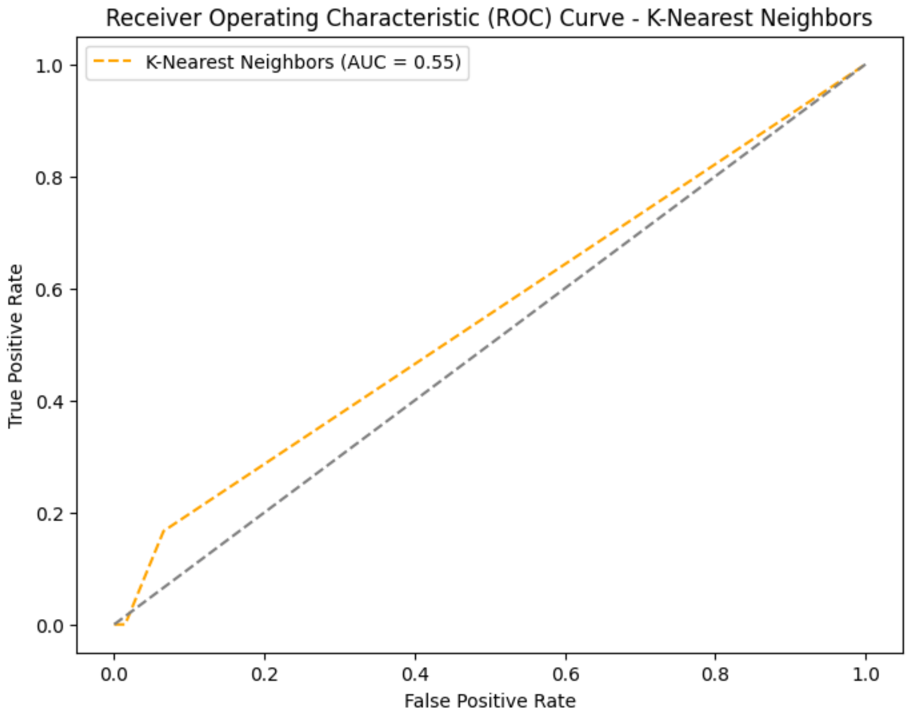
```
accuracy =  0.965034965034965

precision =  1.0

recall =  0.23076923076923078

F1 Score =  0.375

confusion matrix:
 [[273   0]
 [ 10   3]]
```

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
kfold = StratifiedKFold(n_splits=5, shuffle=True)
cvScores = cross_val_score(knn, X, y, cv = kfold, scoring = 'accuracy')
avg_Accuracy = cvScores.mean()
print("Avg accuracy = ", avg_Accuracy)
```

```
Avg accuracy =  0.9565842608096128
```

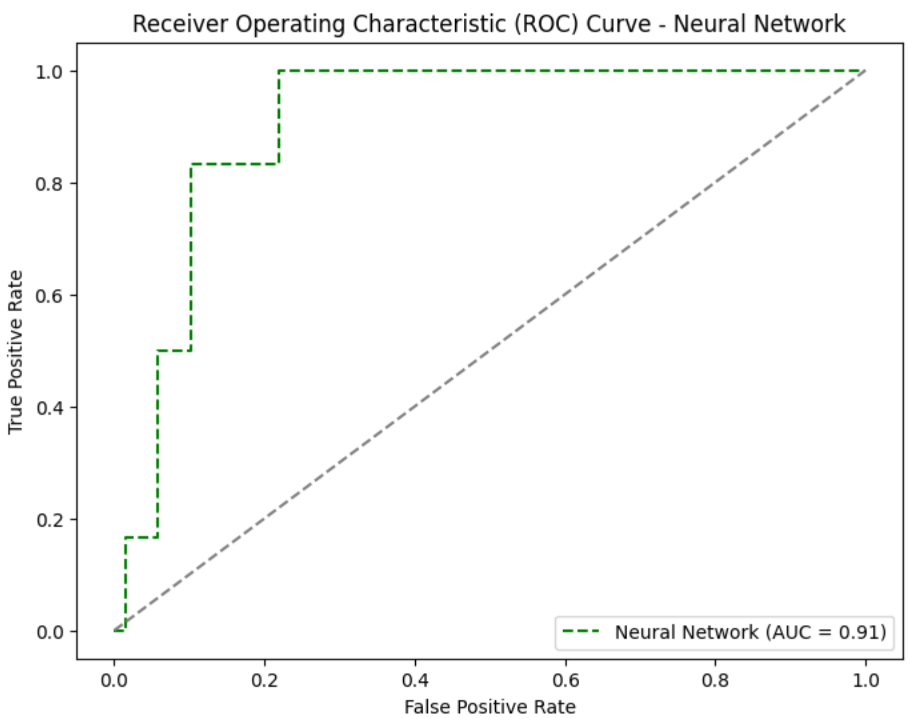
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Assuming 'X_train_scaled' and 'X_test_scaled' are already defined
# You can replace the 'n_neighbors' parameter with your preferred value
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train_scaled, y_train)

# Make predictions
knn_predictions = knn_model.predict(X_test_scaled)

# Calculate metrics
knn_accuracy = accuracy_score(y_test, knn_predictions)
knn_precision = precision_score(y_test, knn_predictions)
knn_recall = recall_score(y_test, knn_predictions)
knn_f1 = f1_score(y_test, knn_predictions)
knn_confusion_matrix = confusion_matrix(y_test, knn_predictions)

# Display the metrics
print("K-Nearest Neighbors:")
print("Confusion Matrix:")
print(knn_confusion_matrix)
print("Accuracy:", knn_accuracy)
print("Precision:", knn_precision)
print("Recall:", knn_recall)
print("F-measure:", knn_f1)

# Calculate ROC and AUC for K-Nearest Neighbors
knn_proba = knn_model.predict_proba(X_test_scaled)[:, 1]
knn_fpr, knn_tpr, _ = roc_curve(y_test, knn_proba)
knn_auc = roc_auc_score(y_test, knn_proba)

# Display the metrics and ROC curve for K-Nearest Neighbors
print(f"K-Nearest Neighbors ROC AUC: {knn_auc:.2f}")

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(knn_fpr, knn_tpr, label=f'K-Nearest Neighbors (AUC = {knn_auc:.2f})', linestyle='--', color='orange')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title('Receiver Operating Characteristic (ROC) Curve - K-Nearest Neighbors')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

```
K-Nearest Neighbors:
Confusion Matrix:
[[135   2]
 [  6   0]]
Accuracy: 0.9440559440559441
Precision: 0.0
Recall: 0.0
F-measure: 0.0
K-Nearest Neighbors ROC AUC: 0.55
```



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Assuming you have a DataFrame 'df' with features and target variable
# Replace 'champions' with the actual name of your target variable
X = df.drop('champions', axis=1)
y = df['champions']

# Perform one-hot encoding for categorical columns
X_encoded = pd.get_dummies(X, columns=['team'])

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Apply StandardScaler to the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build a simple neural network model
model = Sequential()
model.add(Dense(128, input_dim=X_train_scaled.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, verbose=1)

# Make predictions
nn_predictions_proba = model.predict(X_test_scaled)
nn_predictions = (nn_predictions_proba > 0.5).astype(int)

# Calculate metrics
nn_accuracy = accuracy_score(y_test, nn_predictions)
nn_precision = precision_score(y_test, nn_predictions)
nn_recall = recall_score(y_test, nn_predictions)
nn_f1 = f1_score(y_test, nn_predictions)
nn_confusion_matrix = confusion_matrix(y_test, nn_predictions)

# Calculate ROC and AUC for Neural Network
nn_fpr, nn_tpr, _ = roc_curve(y_test, nn_predictions_proba)
nn_auc = roc_auc_score(y_test, nn_predictions_proba)

# Display the metrics and ROC curve
print("Neural Network:")
print("Confusion Matrix:")
print(nn_confusion_matrix)
print("Accuracy:", nn_accuracy)
print("Precision:", nn_precision)
print("Recall:", nn_recall)
print("F-measure:", nn_f1)
print(f"Neural Network ROC AUC: {nn_auc:.2f}")

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(nn_fpr, nn_tpr, label=f'Neural Network (AUC = {nn_auc:.2f})', linestyle='--', color='green')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title('Receiver Operating Characteristic (ROC) Curve - Neural Network')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

```
Epoch 1/10
18/18 [==============================] - 2s 7ms/step - loss: 0.3218 - accuracy: 0.8862
Epoch 2/10
18/18 [==============================] - 0s 10ms/step - loss: 0.1311 - accuracy: 0.9615
Epoch 3/10
18/18 [==============================] - 0s 9ms/step - loss: 0.1021 - accuracy: 0.9615
Epoch 4/10
18/18 [==============================] - 0s 5ms/step - loss: 0.0883 - accuracy: 0.9597
Epoch 5/10
18/18 [==============================] - 0s 5ms/step - loss: 0.0765 - accuracy: 0.9667
Epoch 6/10
18/18 [==============================] - 0s 5ms/step - loss: 0.0682 - accuracy: 0.9667
Epoch 7/10
18/18 [==============================] - 0s 6ms/step - loss: 0.0630 - accuracy: 0.9685
Epoch 8/10
18/18 [==============================] - 0s 7ms/step - loss: 0.0568 - accuracy: 0.9755
Epoch 9/10
18/18 [==============================] - 0s 5ms/step - loss: 0.0515 - accuracy: 0.9772
Epoch 10/10
18/18 [==============================] - 0s 5ms/step - loss: 0.0497 - accuracy: 0.9702
5/5 [==============================] - 0s 4ms/step
Neural Network:
Confusion Matrix:
[[136   1]
 [  6   0]]
Accuracy: 0.951048951048951
Precision: 0.0
Recall: 0.0
F-measure: 0.0
Neural Network ROC AUC: 0.91
```



```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Assuming 'X_train_scaled' and 'X_test_scaled' are already defined
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)

# Make predictions
nb_predictions = nb_model.predict(X_test_scaled)

# Calculate metrics
nb_accuracy = accuracy_score(y_test, nb_predictions)
nb_precision = precision_score(y_test, nb_predictions)
nb_recall = recall_score(y_test, nb_predictions)
nb_f1 = f1_score(y_test, nb_predictions)
nb_confusion_matrix = confusion_matrix(y_test, nb_predictions)

# Display the metrics
print("Naive Bayesian:")
print("Confusion Matrix:")
print(nb_confusion_matrix)
print("Accuracy:", nb_accuracy)
print("Precision:", nb_precision)
print("Recall:", nb_recall)
print("F-measure:", nb_f1)
```