```
#!/usr/bin/env python
# coding: utf-8

# In[714]:


import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# In[715]:


get_ipython().run_line_magic('matplotlib', 'inline')


# In[716]:


df13 = pd.read_csv('FB_Data/cfb13.csv')


# In[717]:


df14 = pd.read_csv('FB_Data/cfb14.csv')


# In[718]:


df15 = pd.read_csv('FB_Data/cfb15.csv')


# In[719]:


df16 = pd.read_csv('FB_Data/cfb16.csv')


# In[720]:


df17 = pd.read_csv('FB_Data/cfb17.csv')


# In[721]:


df18 = pd.read_csv('FB_Data/cfb18.csv')


# In[722]:


df19 = pd.read_csv('FB_Data/cfb19.csv')


# In[723]:


df20 = pd.read_csv('FB_Data/cfb20.csv')


# In[724]:


df21 = pd.read_csv('FB_Data/cfb21.csv')


# In[725]:


df22 = pd.read_csv('FB_Data/cfb22.csv')


# In[726]:


df_combined = pd.concat([df13, df14, df15, df16, df17, df18, df19, df20, df21, df22], axis=0)


# In[727]:


df_combined


# In[728]:


x = df_combined[['Games',
 'Win',
 'Loss',
 'Off.Rank',
 'Off.Plays',
 'Off.Yards',
 'Off.Yards.Play',
 'Off.TDs',
 'Total.TDs',
 'Off.Yards.per.Game',
```

```
'Def.Rank',
'Def.Plays',
'Yards.Allowed',
'Yards.Play.Allowed',
'Off.TDs.Allowed',
'Total.TDs.Allowed',
'Yards.Per.Game.Allowed',
'First.Down.Rank',
'First.Down.Runs',
'First.Down.Passes',
'First.Down.Penalties',
'First.Downs',
'First.Down.Def.Rank',
'Opp.First.Down.Runs',
'Opp.First.Down.Passes',
'Opp.First.Down.Penalties',
'Opp.First.Downs',
'X4th.Down.Rank',
'X4th.Attempts',
'X4th.Conversions',
'X4th.Percent',
'X4rd.Down.Def.Rank',
'Opp.4th.Conversion',
'Opp.4th.Attempt',
'Opponent.4th.Percent',
'Kickoff.Return.Rank',
'Kickoffs.Returned',
'Kickoff.Return.Yards',
'Kickoff.Return.Touchdowns',
'Avg.Yard.per.Kickoff.Return',
'Passing.Off.Rank',
'Pass.Attempts',
'Pass.Completions',
'Interceptions.Thrown.x',
'Pass.Yards',
'Pass.Yards.Attempt',
'Yards.Completion',
'Pass.Touchdowns',
'Pass.Yards.Per.Game',
'Pass.Def.Rank',
'Opp.Completions.Allowed',
'Opp.Pass.Attempts',
'Opp.Pass.Yds.Allowed',
'Opp.Pass.TDs.Allowed',
'Yards.Attempt.Allowed',
'Yards.Completion.Allowed',
'Pass.Yards.Per.Game.Allowed',
'Penalty.Rank',
'Penalties',
'Penalty.Yards',
'Penalty.Yards.Per.Game',
'Punt.Return.Rank',
'Punt.Returns',
'Net.Punt.Return.Yards',
'Punt.Return.Touchdowns',
'Avg.Yards.Per.Punt.Return',
'Punt.Return.Def.Rank',
'Opp.Punt.Returns',
'Opp.Net.Punt.Return.Yards',
'Opp.Punt.Return.Touchdowns.Allowed',
'Avg.Yards.Allowed.per.Punt.Return',
'Redzone.Off.Rank',
'Redzone.Attempts',
'Redzone.Rush.TD',
'Redzone.Pass.TD',
'Redzone.Field.Goals.Made',
'Redzone.Scores',
'Redzone.Points',
'Redzone.Def.Rank',
'Opp.Redzone.Attempts',
'Opp.Redzone.Rush.TD.Allowed',
'Opp.Redzone.Pass.Touchdowns.Allowed',
'Opp.Redzone.Field.Goals.Made',
'Opp.Redzone.Scores',
'Redzone.Points.Allowed',
'Rushing.Off.Rank',
'Rush.Attempts',
'Rush.Yds',
'Yards.Rush',
'Rushing.TD',
'Rushing.Yards.per.Game',
'Rushing.Def.Rank',
'Opp.Rush.Attempts',
'Opp.Rush.Yards.Alloweed',
'Yds.Rush.Allowed',
'Opp.Rush.Touchdowns.Allowed',
'Rush.Yards.Per.Game.Allowed',
'Sack.Rank',
'Sacks',
'Sack.Yards',
'Average.Sacks.per.Game',
'Scoring.Def.Rank',
'Touchdowns.Allowed',
'Opponent.Extra.Points',
'X2.Point.Conversions.Allowed',
'Opp.Deflected.Extra.Points',
'Opp.Feild.Goals.Made',
'Opp.Safety',
'Points.Allowed',
'Avg.Points.per.Game.Allowed',
'Scoring.Off.Rank',
'Touchdowns',
'PAT',
```

```
 'X2.Point.Conversions',
 'Defensive.Points',
 'Feild.Goals',
 'Safety',
 'Total.Points',
 'Points.Per.Game',
 'Tackle.for.Loss.Rank',
 'Solo.Tackle.For.Loss',
 'Assist.Tackle.For.Loss',
 'Tackle.for.Loss.Yards',
 'Total.Tackle.For.Loss',
 'Tackle.For.Loss.Per.Game',
 'X3rd.Down.Rank',
 'X3rd.Attempts',
 'X3rd.Conversions',
 'X3rd.Percent',
 'X3rd.Down.Def.Rank',
 'Opp.3rd.Conversion',
 'Opp.3rd.Attempt',
 'Opponent.3rd.Percent',
 'Time.of.Possession.Rank',
 'Turnover.Rank',
 'Fumbles.Recovered',
 'Opponents.Intercepted',
 'Turnovers.Gain',
 'Fumbles.Lost',
 'Interceptions.Thrown.y',
 'Turnovers.Lost',
 'Turnover.Margin',
 'Avg.Turnover.Margin.per.Game']]


# In[729]:


df_combined


# In[730]:


x


# In[731]:


y = df_combined['Win']


# In[732]:


df_combined


# In[733]:


for i in range(0,len(x.columns)):
    x.iloc[:,i] = x.iloc[:,i].fillna(x.iloc[:,i].mean())



# In[734]:


y = y.fillna(y.mean())


# In[735]:


x.isnull().sum()


# In[736]:


x.shape


# In[737]:


x.shape


# In[738]:


y.shape


# In[739]:


from sklearn.model_selection import train_test_split


# In[740]:
```

```
from sklearn.linear_model import LinearRegression

# In[741]:

lm = LinearRegression()

# In[742]:

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=101)

# In[743]:

lm.fit(x_train, y_train)

# In[744]:

lm.coef_

# In[745]:

lm.intercept_

# In[746]:

cdf = pd.DataFrame(lm.coef_,x.columns, columns=['Coefficients'])

# In[747]:

cdf

# In[748]:

sns.displot(y, stat='count', kde=True, discrete=True)

# In[749]:

predictions = lm.predict(x_test)

# In[750]:

predictions

# In[751]:

sns.displot(y_test-predictions)

# In[752]:

plt.scatter(y_test, predictions)

# In[753]:

predictComp = pd.DataFrame(zip(df_combined['Team'],y_test, predictions), columns=['Team','real value', 'predicted value'])

# In[754]:

predictComp

# In[755]:

predictComp.to_csv('FB_predictions/combinedFootballPrediction.csv')

# In[756]:

from sklearn.metrics import confusion_matrix, classification_report

# In[757]:

dfFb2 = pd.read_csv('FB_data/collegefootballbowl.csv')
```

```
# In[758]:


pd.get_dummies(dfFb2['winner_tie'])


# In[759]:


pd.get_dummies(df_combined['Team'])


# In[760]:


shorter_team_list = []
for i in pd.get_dummies(df_combined['Team']).columns:
    team = ''
    for j in i:
        if j != '(':
            team += j
        else:
            break
    shorter_team_list.append(team)

shorter_team_list


# In[761]:


same_team_list = []
for x in pd.get_dummies(dfFb2['winner_tie']).columns:
    for y in shorter_team_list:
        if x.replace(" ", "") == y.replace(" ", ""):
            same_team_list.append(x)



# In[762]:


pd.get_dummies(dfFb2['winner_tie']).columns


# In[763]:


filtered_teams = pd.DataFrame(columns=df_combined.columns)
no_paren_teams = []
for k in df_combined['Team']:
    team = ''
    for n in k:
        if n != '(':
            team+=n
        else:
            break
    no_paren_teams.append(team)


# In[764]:


no_paren_teams


# In[765]:


for i in range(len(no_paren_teams)):
    for j in range(len(same_team_list)):
        if no_paren_teams[i].replace(" ", "") == same_team_list[j].replace(" ", ""):
            filtered_teams.loc[len(filtered_teams.index)]= df_combined.iloc[i, :]


# In[766]:


filtered_teams
teamList = filtered_teams['Team']


# In[767]:


filtered_teams.isnull().count()


# In[768]:


df_combined.isnull().count()


# In[769]:


team_split = pd.get_dummies(filtered_teams['Team'], drop_first=True)


# In[770]:
```

```
#filtered_teams = pd.concat([team_split, filtered_teams], axis=1)

# In[771]:

filtered_teams

# In[772]:

filtered_teams.drop('Team', inplace=True, axis=1)

# In[773]:

filtered_teams

# In[774]:

filtered_teams[['Time.of.Possession','Average.Time.of.Possession.per.Game']].dropna()

# In[775]:

filtered_teams['Average.Time.of.Possession.per.Game'][0:111].apply(lambda cols: int(str(cols).split(':')[0])).mean()

# In[776]:

def convert_time(cols):
    try:
        dateComponents1 = cols[0].split(':')
        dateComponents2 = cols[1].split(':')

        index1 = 0
        for j in dateComponents1[0]:
            if j == 0:
                index1 = dateComponents1[0].index(j)
        index2 = 0
        for h in dateComponents1[1]:
            if h == 0:
                index2 = dateComponents1[1].index(h)

        index3 = 0
        for k in dateComponents2[0]:
            if k == 0:
                index3 = dateComponents2[0].index(k)

        index4 = 0
        for g in dateComponents2[1]:
            if g == 0:
                index3 = dateComponents2[1].index(g)

        return (int(dateComponents1[0][index1:])*60+int(dateComponents1[1][index2:])), (int(dateComponents2[0][index3:])*60+int(dateComponents2[1][index4:]))
    except:
        return 381.3783783783784*60, 29.486486486486488*60

convTimes = filtered_teams[['Time.of.Possession','Average.Time.of.Possession.per.Game']].apply(convert_time, axis=1, result_type='expand')

# In[777]:

convTimes

# In[778]:

convTimes.rename(columns={0:'Time.of.Possession_sec', 1:'Average.Time.of.Possession.per.Game_sec'}, inplace=True)

# In[779]:

convTimes

# In[780]:

filtered_teams = pd.concat([filtered_teams, convTimes], axis=1)

# In[781]:

filtered_teams.drop(['Time.of.Possession','Average.Time.of.Possession.per.Game'], axis=1, inplace=True)

# In[782]:

sns.heatmap(filtered_teams.isnull(),yticklabels=False, cbar=False, cmap='viridis')
```

```
# In[783]:


filtered_teams_y = filtered_teams['Win']
filtered_teams_y = filtered_teams_y.fillna(filtered_teams_y.mean())


# In[784]:


filtered_teams.drop(['Kickoff.Return.Def.Rank'], axis=1, inplace=True)


# In[785]:


filtered_teams['Avg.Turnover.Margin.per.Game']


# In[786]:


filtered_teams_x = filtered_teams.drop('Win', axis=1)
#filtered_teams_x = filtered_teams_x.iloc[:,:285]
dropIndex = []
for i in range(0,len(filtered_teams_x.columns)):
    try:
        filtered_teams_x.iloc[:,i].fillna(filtered_teams_x.iloc[:,i].mean(), inplace=True)
    except:
        dropIndex.append(filtered_teams_x.columns[i])


# In[787]:


for j in dropIndex:
    filtered_teams_x.drop(j, inplace=True, axis=1)


# In[788]:


dropIndex


# In[789]:


len(filtered_teams_x.columns)


# In[790]:


sns.heatmap(filtered_teams_x.isnull(),yticklabels=False, cbar=False, cmap='viridis')


# In[791]:


def convert_time(cols):
    try:
        dateComponents1 = cols[0].split(':')
        dateComponents2 = cols[1].split(':')

        index1 = 0
        for j in dateComponents1[0]:
            if j == 0:
                index1 = dateComponents1[0].index(j)
        index2 = 0
        for h in dateComponents1[1]:
            if h == 0:
                index2 = dateComponents1[1].index(h)

        index3 = 0
        for k in dateComponents2[0]:
            if k == 0:
                index3 = dateComponents2[0].index(k)

        index4 = 0
        for g in dateComponents2[1]:
            if g == 0:
                index3 = dateComponents2[1].index(g)

        return (int(dateComponents1[0][index1:])*60+int(dateComponents1[1][index2:])), (int(dateComponents2[0][index3:])*60+int(dateComponents2[1][index4:]))
    except:
        return 381.3783783783784*60, 29.486486486486488*60

convTimes = df_combined[['Time.of.Possession','Average.Time.of.Possession.per.Game']].apply(convert_time, axis=1, result_type='expand')
convTimes.rename(columns={0:'Time.of.Possession_sec', 1:'Average.Time.of.Possession.per.Game_sec'}, inplace=True)
df_combined = pd.concat([df_combined, convTimes], axis=1)
df_combined.drop(['Time.of.Possession','Average.Time.of.Possession.per.Game'], axis=1, inplace=True)


# In[792]:


df_combined.drop('Team', axis=1, inplace=True)


# In[793]:
```

```
dropIndex = []
for i in range(0,len(df_combined.columns)):
    try:
        df_combined.iloc[:,i].fillna(df_combined.iloc[:,i].mean(), inplace=True)
    except:
        dropIndex.append(df_combined.columns[i])
for j in dropIndex:
    df_combined.drop(j, inplace=True, axis=1)


# In[794]:


sns.heatmap(df_combined.isnull(),yticklabels=False, cbar=False, cmap='viridis')


# In[795]:


x2 = df_combined.drop(['Win','Kickoff.Return.Def.Rank'] , axis=1).iloc[:,:280]
y2 = df_combined['Win']
x_train2, x_test2, y_train2, y_test2 = train_test_split(x2, y2, test_size=0.01, random_state=101)


# In[796]:


linmo = LinearRegression()


# In[797]:


linmo.fit(x_train2, y_train2)


# In[798]:


filtered_teams_x


# In[799]:


x2


# In[800]:


prediction2 = linmo.predict(filtered_teams_x.iloc[:,:280])


# In[801]:


list(set(x2.columns) - set(filtered_teams_x.columns))


# In[802]:


predictComp2 = pd.DataFrame(zip(teamList,filtered_teams_y, prediction2), columns=['Team','real value', 'predicted value'])


# In[803]:


predictComp2


# In[804]:


predictComp2.to_csv('FB_predictions/linearMergedPrediction', index=True)


# In[805]:


import math
math.sqrt(24*0.76)


# In[806]:


len(teamList)


# In[807]:


len(no_paren_teams)


# In[808]:


len(prediction2)
```

```
# In[809]:


no_paren_TeamList = []
for k in teamList:
    team = ''
    for n in k:
        if n != '(':
            team+=n
        else:
            break
    no_paren_TeamList.append(team)


predictComp2 = pd.DataFrame(zip(no_paren_TeamList,filtered_teams_y, prediction2), columns=['Team','#_of_wins_real value', '#_of_wins_predicted value'])


# In[810]:


predictComp2.to_csv('FB_predictions/linearMergedPrediction.csv', index=False)


# In[811]:


predictComp2


# In[812]:


filtered_teams_y2 = filtered_teams['Loss']
filtered_teams_y2 = filtered_teams_y2.fillna(filtered_teams_y2.mean())

filtered_teams_x2 = filtered_teams.drop('Loss', axis=1)

dropIndex = []
for i in range(0,len(filtered_teams_x2.columns)):
    try:
        filtered_teams_x2.iloc[:,i].fillna(filtered_teams_x2.iloc[:,i].mean(), inplace=True)
    except:
        dropIndex.append(filtered_teams_x2.columns[i])

for j in dropIndex:
    filtered_teams_x2.drop(j, inplace=True, axis=1)

sns.heatmap(filtered_teams_x2.isnull(),yticklabels=False, cbar=False, cmap='viridis')


# In[813]:


x3 = df_combined.drop(['Loss','Kickoff.Return.Def.Rank'] , axis=1).iloc[:,:280]
y3 = df_combined['Loss']
x_train3, x_test3, y_train3, y_test3 = train_test_split(x3, y3, test_size=0.01, random_state=101)
linmo2 = LinearRegression()
linmo2.fit(x_train3, y_train3)
prediction3 = linmo2.predict(filtered_teams_x2.iloc[:,:280])
predictComp3 = pd.DataFrame(zip(filtered_teams_y2, prediction3), columns=['#_of_losses_real value', '#_of_losses_predicted value'])


# In[814]:


df_combined


# In[815]:


predictComp2 = pd.concat([predictComp2, predictComp3],axis=1)


# In[816]:


predictComp3


# In[817]:


predictComp2.to_csv('FB_predictions/linearMergedPrediction.csv', index=False)


# In[ ]:




# In[ ]:
```