

In [23]:

```
import pandas as pd
import matplotlib.pyplot as plt
```

In [12]:

```
#Classe degli indicatori che ci permette di inizializzare il dataframe data una squadra i
n input

## Classe Indicators utilizzata per contener einformazioni su Team (= squadra scelta) #
#
## __init__: inizializza le varie istanze dell'oggetto, che poi saranno gli indicatori di
Team ##

class Indicators:

    def __init__(self, DataFrame, TeamChosen):
        self.Team = TeamChosen
        self.TeamMatches = DataFrame.loc[((DataFrame.home_team == self.Team) | (DataFram
e.away_team == self.Team)) & (DataFrame.tournament == "FIFA World Cup")].copy()
        #inizializziamo gli indicatori di vittorie, sconfitte e pareggi
        self.wins = len(self.TeamMatches[((self.TeamMatches.home_team == self.Team) & (s
elf.TeamMatches.home_score > self.TeamMatches.away_score)) | ((self.TeamMatches.away_tea
m == self.Team) & (self.TeamMatches.home_score < self.TeamMatches.away_score))].index)
        self.loses = len(self.TeamMatches[((self.TeamMatches.home_team == self.Team) & (
self.TeamMatches.home_score < self.TeamMatches.away_score)) | ((self.TeamMatches.away_te
am == self.Team) & (self.TeamMatches.home_score > self.TeamMatches.away_score))].index)
        self.drafts = len(self.TeamMatches[self.TeamMatches.home_score == self.TeamMatch
es.away_score].index)
        goals_home = self.TeamMatches[['date', 'home_score', 'away_score']][self.TeamMatc
hes.home_team == self.Team]
        goals_away = self.TeamMatches[['date', 'away_score', 'home_score']][self.TeamMatc
hes.away_team == self.Team]

        goals_home.columns = ['date', 'goals_scored', 'goals_conceded']
        goals_away.columns = ['date', 'goals_scored', 'goals_conceded']

        frames_goals = [goals_home, goals_away]
        self.goals = pd.concat(frames_goals).sort_index()
        #Convertiamo da obj a datetime per poi poter andare a fare il groupby sulla data
nel metodo plotData()
        self.goals['date'] = pd.to_datetime(self.goals['date'])
        #Calcoliamo media dei goal fatti e subiti
        self.meanScores = self.goals['goals_scored'].mean()
        self.meanConceded = self.goals['goals_conceded'].mean()

    def printData(self):
        print("Wins : " + str(self.wins))
        print("Loses : " + str(self.loses))
        print("Drafts : " + str(self.drafts))
        print("Sum of W+L+D : " + str(self.wins+self.loses+self.drafts))
        print("The mean of the scores made by the team : " + str(self.meanScores))
        print("The mean of the scores conceded by the team : " + str(self.meanConceded))

    ## plotData(): metodo utilizzato per graficare il numero dei gol fatti e subiti di Te
am per anno
    def plotData(self):
        self.goals['goals_difference'] = self.goals['goals_scored'] - self.goals['goals_
conceded']
        perYearPlot = self.goals.groupby(self.goals.date.dt.year).sum()
        perYearPlot.reset_index().plot(kind = "line", x="date", y="goals_scored", label=
"Goal Scored")
        perYearPlot.reset_index().plot(kind = "line", x="date", y="goals_conceded", labe
l="Goal Conceded")
        perYearPlot.reset_index().plot(kind = "line", x="date", y="goals_difference", la
bel="Goal Difference")
```

```

plt.show()

## metodi getter ##
def getData(self):
    return self.wins, self.loses, self.drafts, self.meanScores, self.meanConceded
def getMeanScores(self):
    return self.meanScores
def getMeanConceded(self):
    return self.meanConceded
def getWins(self):
    return self.wins
def getLoses(self):
    return self.loses
def getNumMatchesPlayed(self):
    return self.wins + self.drafts + self.loses
def getTeamMatches(self):
    return self.TeamMatches

## isBetter(): metodo utilizzato per paragonare Team con gli indicatori di altre squadre (indicatori in formato tupla) ##
## paragone che viene eseguito per singolo indicatore:
# - wins: numero vittorie
# - loses: numero sconfitte
# - attack: media gol fatti per partita
# - defense: media gol subiti per partita

def isBetter(self, otherTeamIndicators, parameter="wins"):
    if (parameter == "wins"):
        return self.wins >= otherTeamIndicators[0]

    if (parameter == "loses"):
        return self.wins <= otherTeamIndicators[1]

    if (parameter == "attack"):
        return self.meanScores >= otherTeamIndicators[3]

    if (parameter == "defense"):
        return self.meanConceded <= otherTeamIndicators[4]
pass

```

In [13]:

```

#####
## Analisi statistiche su <TeamName> nella storia della FIFA World Cup ##
#####

# Dataframe con tutte le partite di tutte le nazionali #
worldFootball = pd.read_csv("results.csv")

# Vengono filtrate tutte e solo quelle della FIFA World Cup #
worldFootball = worldFootball[worldFootball.tournament == "FIFA World Cup"]

# Selezione team su cui eseguire analisi statistiche #
TeamName = "Italy"

# Vengono reperiti gli indicatori di Team e successivamente visualizzati e graficati #
Team = Indicators(worldFootball, TeamName)

Team.printData()

Team.plotData()

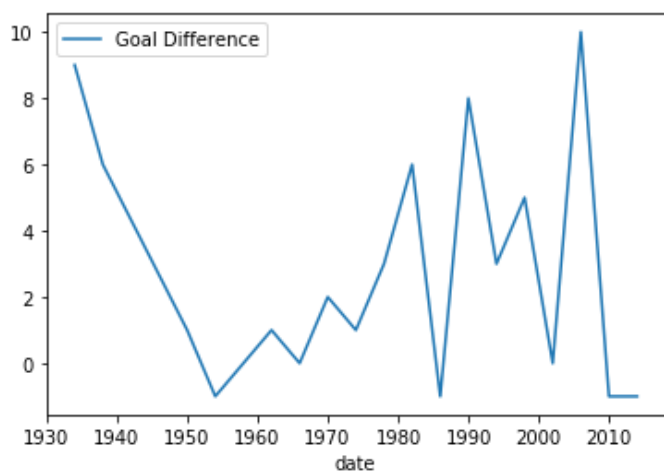
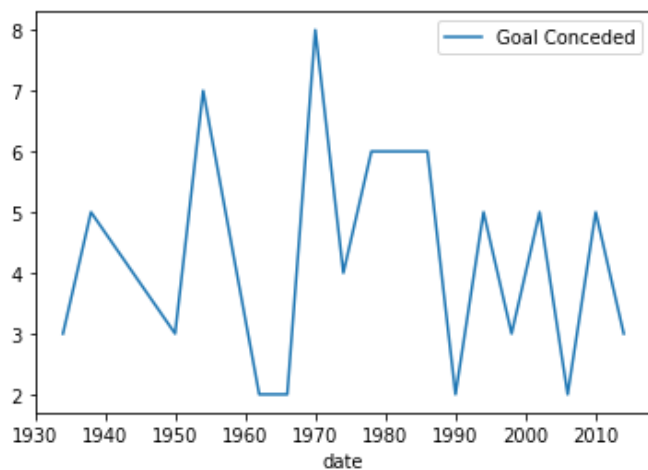
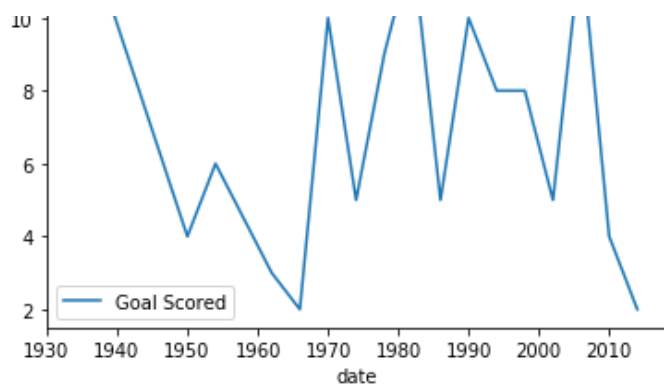
```

```

Wins : 45
Loses : 17
Drafts : 21
Sum of W+L+D : 83
The mean of the scores made by the team : 1.5421686746987953
The mean of the scores conceded by the team : 0.927710843373494

```





In [15]:

```
#####
## Confronto di Team con le altre squadre ##
#####
# Metodi di confronto:
# - per trovare le squadre migliori in assoluto viene stilata una classifica
# di tutte le squadre che hanno partecipato alla FWC prendendo in considerazione
# tutte le partite di tutta la storia (3 punti per vittoria, 1 punto per pareggio e 0
per sconfitta)
# [Metodo ispirato dallo stesso metodo di confronto utilizzato anche sulla pagina dei
mondiali di calcio di Wikipedia]
#
# - gli indicatori delle squadre vengono confrontati singolarmente, visualizzando
# tutte le squadre che hanno l'indicatore x migliore di Team, con annessi anche dati
che contestualizzano
# l'indicatore. (Ad esempio nelle medie gol fatti/subiti è giusto tenere conto anche
delle partite giocate
# in quanto il confronto con le squadre che hanno giocato un numero di partite molto
lontano da quello di Team
# potrebbe risultare fazioso)#

# Generazione dataframe con tutte le squadre che hanno partecipato a FWC #
# preleva sia home_team che away_team perché potrebbero esserci squadre che han giocat
```

*o solo una volta #*

```
allTeams_home = worldFootball[['home_team']].drop_duplicates()
allTeams_away = worldFootball[['away_team']].drop_duplicates()

allTeams_away.columns = ['team']
allTeams_home.columns = ['team']

allTeams = pd.concat([allTeams_away, allTeams_home]).drop_duplicates()
```

In [16]:

```
# In tale dataframe associo ad ogni team le proprie statistiche (indicatori, punti) #
# gli indicatori delle squadre vengono ottenuti creando un oggetto della squadre e utiliz
zando il getter che restituisce la tupla degli indicatori #
# Tupla indicatori = (W, L, D, MediaGolFatti, MediaGolSubiti) #

allTeams['indicators'] = (allTeams['team'].map(lambda x: Indicators(worldFootball, x).ge
tData()))

allTeams['Pts'] = (allTeams['indicators'].map(lambda x: x[0]*3 + x[2]))

# ottengo i punti di Team #

TeamPts = Team.getData()
TeamPts = TeamPts[0]*3 + TeamPts[2]

# filtro tutte le squadre che hanno ottenuto un punteggio migliore di quello di Team #
BetterTeamsByPts = allTeams[['team', 'Pts']][allTeams.Pts > TeamPts]

# visualizzo le squadre migliori #
print(f"Teams that have gained more points than {TeamName} ({TeamPts})")
print("-----")
print(BetterTeamsByPts)
print("")
```

Teams that have gained more points than Italy (156)

```
-----
      team  Pts
1327  Brazil  237
1690  Germany  221
```

In [17]:

```
#GET BETTER TEAMS BY WINNING TIMES:
# - isBetter è una series con True/False a seconda se è vero che la squadra ha più vitt
orie di Team
# - won/played contiene una stringa che evidenzia quante vittorie ogni squadra ha otten
uto e su quante partite
# - winning rate contiene il rateo vittore (vittorie/partite_giocate)

allTeams['isBetter'] = (allTeams['indicators'].map(lambda x: not Team.isBetter(x)))
allTeams['won/played'] = (allTeams['indicators'].map(lambda x: str(x[0]) + '/' + str(x[0]
+x[1]+x[2])))
allTeams['winning rate'] = (allTeams['indicators'].map(lambda x: x[0] / (x[0]+x[1]+x[2])
))

betterTeams = allTeams[['team', 'won/played', 'winning rate']][allTeams.isBetter == True]

# Visualizzo squadre che hanno vinto più di Team#
print(f"Teams that have won more than {TeamName} ({Team.getWins()}/{Team.getNumMatchesPla
yed()}) with a winning rate of {Team.getWins()/Team.getNumMatchesPlayed()})")
print("-----")
print(betterTeams)
print("")
```

Teams that have won more than Italy (45/83 with a winning rate of 0.5421686746987951)

	team	won/played	winning rate
1327	Brazil	73/109	0.669725
1690	Germany	67/109	0.614679

In [18]:

```
#GET BETTER TEAMS BY LOSING TIMES
# - isBetter è una series con True/False a seconda se è vero che la squadra ha meno sconfitte di Team
# - lost/played contiene una stringa che evidenzia quante sconfitte ogni squadra ha ottenuto e su quante partite
# - losing rate rate contiene il rateo sconfitta (sconfitte/partite_giocate)
allTeams['isBetter'] = (allTeams['indicators'].map(lambda x: not Team.isBetter(x, "loses")))
allTeams['lost/played'] = (allTeams['indicators'].map(lambda x: str(x[1]) + '/' + str(x[0]+x[1]+x[2])))
allTeams['losing rate'] = (allTeams['indicators'].map(lambda x: x[1] / (x[0]+x[1]+x[2])))

betterTeams = allTeams[['team', 'lost/played', 'losing rate']][allTeams.isBetter == True]

# Visualizzo squadre che hanno perso meno di Team #
print(f"Teams that have lost less than {TeamName} ({Team.getLoses()}/{Team.getNumMatchesPlayed()}) with a losing rate of {Team.getLoses()/Team.getNumMatchesPlayed()}")
print("-----")
print(betterTeams)
print("")
```

Teams that have lost less than Italy (17/83 with a losing rate of 0.20481927710843373)

	team	lost/played	losing rate
1314	United States	19/33	0.575758
1315	Mexico	27/57	0.473684
1316	Yugoslavia	12/33	0.363636
1317	Romania	8/21	0.380952
1318	France	19/66	0.287879
...	...	...	...
36107	Bosnia and Herzegovina	2/3	0.666667
39901	Iceland	2/3	0.666667
39908	Panama	3/3	1.000000
2102	Cuba	1/3	0.333333
7457	Israel	1/3	0.333333

[81 rows x 3 columns]

In [19]:

```
#GET BETTER TEAMS BY BEST ATTACKS
# - isBetter è una series con True/False a seconda se è vero che la squadra ha un attacco migliore di quello di Team (media gol fatti per partita)
# - goals scored rate contiene la media gol fatti
# - Matches played contiene il numero di partite giocate
allTeams['isBetter'] = (allTeams['indicators'].map(lambda x: not Team.isBetter(x, "attack")))
allTeams['goals scored rate'] = (allTeams['indicators'].map(lambda x: x[3]))
allTeams['Matches played'] = (allTeams['indicators'].map(lambda x: x[0] + x[1] + x[2]))

betterTeams = allTeams[['team', 'goals scored rate', 'Matches played']][allTeams.isBetter == True]

# visualizzo square con un attacco migliore di Team #
print(f"Teams that have a better mean goals scored for match than {TeamName} ({Team.getMeanScores()} in {Team.getNumMatchesPlayed()} matches played)")
print("-----")
print(betterTeams)
```

```
print("")
```

Teams that have a better mean goals scored for match than Italy (1.5421686746987953 in 83 matches played)

```
-----
      team  goals scored rate  Matches played
1316  Yugoslavia          1.666667           33
1318    France          1.818182           66
1327    Brazil          2.100917          109
1333  Argentina          1.691358           81
1688    Sweden          1.568627           51
1690    Germany          2.073394          109
1691    Spain          1.571429           63
1693    Hungary          2.718750           32
2103  Netherlands          1.720000           50
3136    Uruguay          1.553571           56
3658    Turkey          2.000000           10
4366    Russia          1.711111           45
6218    Portugal          1.633333           30
2102     Cuba          1.666667            3
```

In [20]:

```
#GET BETTER TEAMS BY BEST DEFENSE
# - isBetter è una series con True/False a seconda se è vero che la squadra ha una dife
sa migliore di Team (media gol subiti per partita)
# - goals conceded rate contiene la media di gol subiti per partita
# - Matches played contiene il numero di partite giocate
allTeams['isBetter'] = (allTeams['indicators'].map(lambda x: not Team.isBetter(x, "defen
se")))
allTeams['goals conceded rate'] = (allTeams['indicators'].map(lambda x: x[4]))
allTeams['Matches played'] = (allTeams['indicators'].map(lambda x: x[0] + x[1] + x[2]))

betterTeams = allTeams[['team', 'goals conceded rate', 'Matches played']][allTeams.isBetter
== True]

# Visualizzo squadre che hanno una difesa migliore di Team #
print(f"Teams that have a better mean goals conceded for maatch than {TeamName} ({Team.ge
tMeanConceded()} in {Team.getNumMatchesPlayed()} matches played)")
print("-----
----")
print(betterTeams)
print("")
```

Teams that have a better mean goals conceded for maatch than Italy (0.927710843373494 in 83 matches played)

```
-----
      team  goals conceded rate  Matches played
3124   England          0.927536            69
4368    Wales          0.800000             5
9032  German DR          0.833333             6
16110 Republic of Ireland          0.769231           13
28397    Angola          0.666667             3
```

In [21]:

```
#####
##                               ##
#####

## Funzione ausiliaria utilizzata per determinare se Team ha vinto la partita associata #
#
def getResult(row):
    if(row['home_score'] > row['away_score']):
        if(row['home_team'] == TeamName):
            return "W"
        else:
            return "L"
```

```

elif(row['home_score'] < row['away_score']):
    if(row['away_team'] == TeamName):
        return "W"
    else:
        return "L"
else:
    return "D"

```

In [22]:

```

# Ottengo tutte le partite giocate da Team #
TeamMatches = Team.getTeamMatches()

# Aggiungo la colonna/series dell'esito della partite dal punto di vista di Team (W,L,D)
#
TeamMatches['match_result'] = (TeamMatches.apply(getResult, axis=1))

# Creo colonna consecutive che conta quante W, D e L consecutive ci sono
g = TeamMatches.match_result.__eq__("W").astype(int).diff().fillna(0).abs().cumsum()
TeamMatches['consecutive'] = TeamMatches.groupby(g).match_result.cumcount().add(1)

# Nella colonna consecutive prendo solo i numeri associati alle vittorie, poi prendo il
numero massimo #
MaxStreak = TeamMatches[['consecutive']][TeamMatches.match_result == "W"].max()

# Visualizzo serie di vittorie più lunga per Team #
print(f"Longest winning streak for {TeamName}: {MaxStreak}")

```

```

Longest winning streak for Italy: consecutive      7
dtype: int64

```

In [ ]:

```


```