finalproject

December 17, 2017

1 Final Project

1.1 Author: Matt Hixon

1.2 Title: Model College Football Games

1.3 Abstract:

The goal of this project is to create a statistical model of a college football game. A markov network model is constructed using data from the 2013 college football season. In order to simulate a game, multiple random walks are performed on the markov network model which will create a sequence of events that determine the outcome of the game. A monte carlo simulation is then performed on the games to aquire a simulated season. The simulated season is then compared to the actual 2013 season.

1.4 Data

The data was retieved from http://www.cfbstats.com/blog/college-football-data/. The dataset consists of games, players, teams, and statistics for a large majority of games between 2005-2013. For this project, I used 2013 data only.

1.4.1 Plays Data

The plays data file consists of every play for a game and includes the teams playing, score, down, distance to go, and distance to score.

```
In [12]: # import libraries
         import numpy as np
         import pandas as pd
         import random as rnd
         import matplotlib.pyplot as plt
         import seaborn as sns
         from collections import Counter
         import operator
         import networkx as nx
```

```
In [13]: plays = pd.read_csv('data/play.csv')
         print(plays.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160697 entries, 0 to 160696
Data columns (total 14 columns):
Game Code             160697 non-null int64
Play Number           160697 non-null int64
Period Number         160697 non-null int64
Clock                 49767 non-null float64
Offense Team Code     160697 non-null int64
Defense Team Code     160697 non-null int64
Offense Points        160697 non-null int64
Defense Points        160697 non-null int64
Down                  144883 non-null float64
Distance              144883 non-null float64
Spot                  160697 non-null int64
Play Type             160697 non-null object
Drive Number          130817 non-null float64
Drive Play            130817 non-null float64
dtypes: float64(5), int64(8), object(1)
memory usage: 17.2+ MB
None
```

It contains 160697 different plays. There a alot of null values for Clock, Down, and Distance. Upon further inspection, it looks like the Clock was only recorded at a stoppage of play. I decided to ignore the Clock as the lack of data would make the network model inaccurate. Instead, I opted for the average number of plays in a game to serve as the limit to how long a game would occur (more on this later). The Down and Distance contain many null values because of teams scoring. When a team scores, they must kickoff. The kickoff play type does not have a Down or Distance associated with it. This is used to figure out when a scoring play occured when determining probabilities.

```
In [16]: # How many games are there?
         print('Number of unique games {}'.format(plays['Game Code'].nunique()))
         # How many teams are there games for?
         print('Number of unique teams {}'.format(plays['Offense Team Code'].nunique()))
```

```
Number of unique games 848
Number of unique teams 207
```

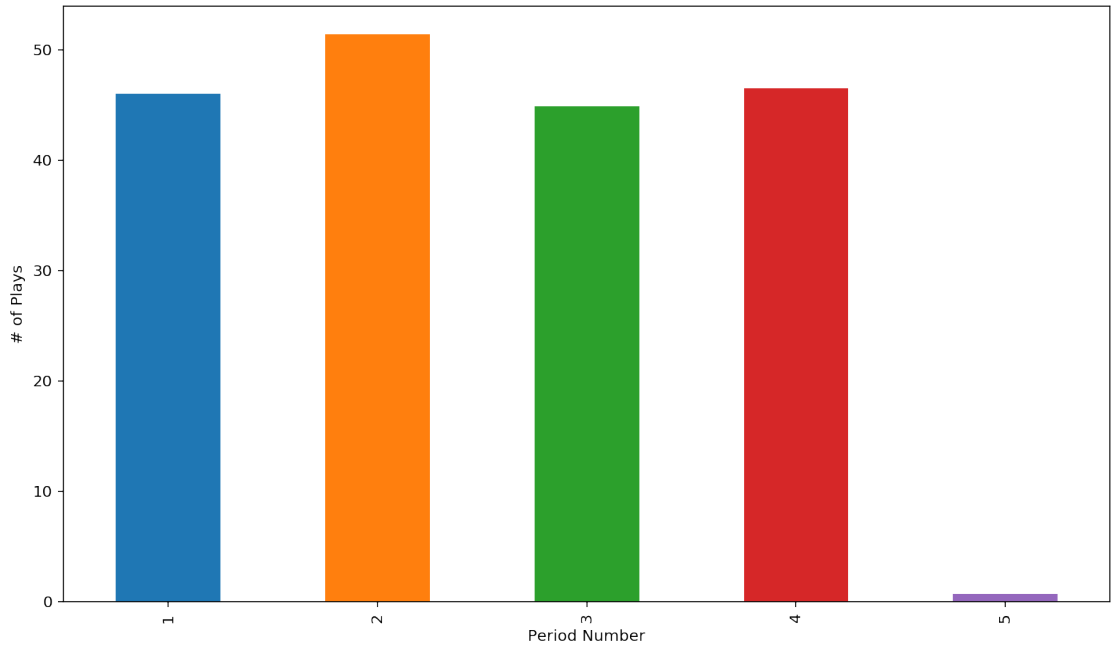
```
In [14]: # Average number of plays per period
         average_plays_per_period = plays.groupby(['Period Number']).size()/(plays['Game Code'].
         average_plays_per_period.plot(kind='bar')
         print(average_plays_per_period)
         plt.xlabel('Period Number')
         plt.ylabel('# of Plays')
```

```
Period Number
1    45.985849
```

```
2    51.408019
3    44.871462
4    46.549528
5     0.686321
dtype: float64
```

Out[14]: Text(0,0.5,'# of Plays')

Out[14]:



After looking at the average number of plays per Period, we determine that the average number of plays in a college football game for 2013 is ~198. We will use this number the number of plays in our simulated games.

1.4.2 Team Game Statistics Data

The team game statistics data file consists of most statistics for a game, which includes touchdowns, punts, red zone attempts, and more.

```
In [17]: team_game_stats = pd.read_csv("data/team-game-statistics.csv")
         print(team_game_stats.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1696 entries, 0 to 1695
Data columns (total 68 columns):
Team Code                1696 non-null int64
```

```
Game Code                1696 non-null int64
Rush Att                 1696 non-null int64
Rush Yard                1696 non-null int64
Rush TD                  1696 non-null int64
Pass Att                 1696 non-null int64
Pass Comp                1696 non-null int64
Pass Yard                1696 non-null int64
Pass TD                  1696 non-null int64
Pass Int                 1696 non-null int64
Pass Conv                1696 non-null int64
Kickoff Ret              1696 non-null int64
Kickoff Ret Yard         1696 non-null int64
Kickoff Ret TD           1696 non-null int64
Punt Ret                 1696 non-null int64
Punt Ret Yard            1696 non-null int64
Punt Ret TD              1696 non-null int64
Fum Ret                  1696 non-null int64
Fum Ret Yard             1696 non-null int64
Fum Ret TD               1696 non-null int64
Int Ret                  1696 non-null int64
Int Ret Yard             1696 non-null int64
Int Ret TD               1696 non-null int64
Misc Ret                 1696 non-null int64
Misc Ret Yard            1696 non-null int64
Misc Ret TD              1696 non-null int64
Field Goal Att           1696 non-null int64
Field Goal Made          1696 non-null int64
Off XP Kick Att          1696 non-null int64
Off XP Kick Made         1696 non-null int64
Off 2XP Att              1696 non-null int64
Off 2XP Made             1696 non-null int64
Def 2XP Att              1696 non-null int64
Def 2XP Made             1696 non-null int64
Safety                   1696 non-null int64
Points                   1696 non-null int64
Punt                     1696 non-null int64
Punt Yard                1696 non-null int64
Kickoff                  1696 non-null int64
Kickoff Yard             1696 non-null int64
Kickoff Touchback        1696 non-null int64
Kickoff Out-Of-Bounds    1696 non-null int64
Kickoff Onside           1696 non-null int64
Fumble                   1696 non-null int64
Fumble Lost              1696 non-null int64
Tackle Solo              1696 non-null int64
Tackle Assist            1696 non-null int64
Tackle For Loss          1696 non-null float64
Tackle For Loss Yard     1696 non-null int64
```

```
Sack                    1696 non-null float64
Sack Yard               1696 non-null int64
QB Hurry                1696 non-null int64
Fumble Forced           1696 non-null int64
Pass Broken Up          1696 non-null int64
Kick/Punt Blocked       1696 non-null int64
1st Down Rush           1696 non-null int64
1st Down Pass           1696 non-null int64
1st Down Penalty        1696 non-null int64
Time Of Possession      1696 non-null int64
Penalty                 1696 non-null int64
Penalty Yard            1696 non-null int64
Third Down Att          1696 non-null int64
Third Down Conv         1696 non-null int64
Fourth Down Att         1696 non-null int64
Fourth Down Conv        1696 non-null int64
Red Zone Att            1696 non-null int64
Red Zone TD             1696 non-null int64
Red Zone Field Goal     1696 non-null int64
dtypes: float64(2), int64(66)
memory usage: 901.1 KB
None
```

Given that there are 848 total games record in the plays dataset, we see that there are two entries for every game with no null values! This means we will have a good comparison for our simulated games.

1.5 Model Design

1.5.1 Markov Chain

A simulated game will be inspired from a markov chain. At any point in the game, a team will be in a state. There will be a transition matrix of probabilities to determine where the state will advance to next, which is based on historical occurances in 2013. At the conclusion of the game, the randomized states will be used to calculate the statistics for that game and then they will be stored for comparison with the statistics from actual games.

1.5.2 The States

In order to simplify the college football game, seven different states of various sets of characteristics were constructed. The seven states are:

Name	Distance to score
Far Field (FF)	FF >= 80
Mid Field (MF)	80 > MF >= 50
Field Goal Range (FGR)	50 > FGR >= 20
Red Zone (RZ)	20 > RZ

Name	Distance to score
Touchdown (TD)	Touchdown
Field Goal (FG)	Field Goal
Safety (S)	Safety

1.5.3 Creating Transition Matrix

In order to create an accurate model, we need to look at the historical transitions between states for the 2013 season and apply those to our network.

Transition Probabilities from FF

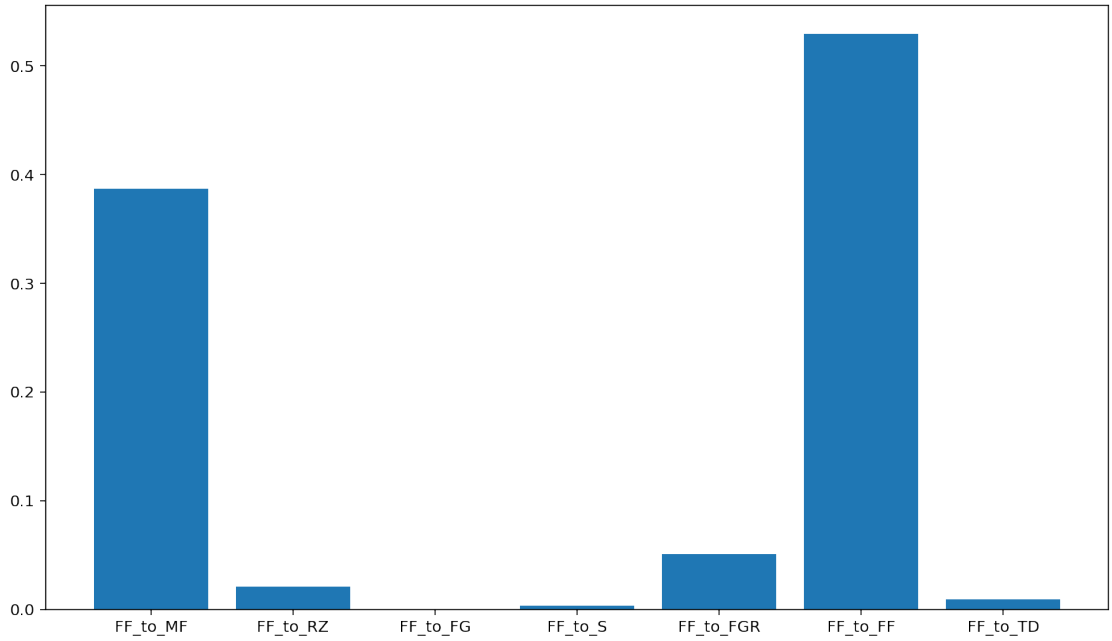
In [20]:
```
# Transition Probability for plays in Far Field (FF)
FF_plays_start = plays.loc[plays['Spot'] >= 80]
FF_plays_start_count = len(plays.loc[plays['Spot'] >= 80].index)
print('Total plays FF: {}'.format(FF_plays_start_count))

# plays start in FF and end in an S
FF_to_S_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] == 80)].in
FF_to_S_index_offset = plays.iloc[list(FF_to_S_index.values.flatten())]
FF_to_S_count =len(FF_to_S_index_offset.loc[FF_to_S_index_offset['Spot'] >= 80].index)
print('FF to S: {}'.format(FF_to_S_count))
#FF_to_S = plays.iloc(i-1) for i in plays.loc[(plays['Play Type'] == 'KICKOFF') & (play

# plays start in FF and end in a TD
FF_to_TD_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].i
FF_to_TD_index_offset_is_TD = plays.iloc[list(FF_to_TD_index.values.flatten())]
FF_to_TD_index_is_TD = FF_to_TD_index_offset_is_TD.loc[(plays['Play Type'] == 'ATTEMPT'
FF_to_TD_index_offset_zone = plays.iloc[list(FF_to_TD_index_is_TD.values.flatten())]
FF_to_TD_count =len(FF_to_TD_index_offset_zone.loc[(plays['Spot'] >= 80)].index)
print('FF to TD: {}'.format(FF_to_TD_count))

# plays start in FF and end in a FG
FF_to_FG_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].i
FF_to_FG_index_offset_is_FG = plays.iloc[list(FF_to_FG_index.values.flatten())]
FF_to_FG_index_is_FG = FF_to_FG_index_offset_is_FG.loc[(plays['Play Type'] == 'FIELD_GO
FF_to_FG_index_offset_zone = plays.iloc[list(FF_to_FG_index_is_FG.values.flatten())]
FF_to_FG_count =len(FF_to_FG_index_offset_zone.loc[(plays['Spot'] >= 80)].index)
print('FF to FG: {}'.format(FF_to_FG_count))

# Plays start in FF and end in MF
FF_to_MF_index = plays.loc[(plays['Spot'] < 80) & (plays['Spot'] >= 50)].index.to_frame
FF_to_MF_index_offset = plays.iloc[list(FF_to_MF_index.values.flatten())]
FF_to_MF_count =len(FF_to_MF_index_offset.loc[FF_to_MF_index_offset['Spot'] >= 80].inde
print('FF to MF: {}'.format(FF_to_MF_count))

# Plays start in FF and end in FGR
```

```
        FF_to_FGR_index = plays.loc[(plays['Spot'] < 50) & (plays['Spot'] >= 20)].index.to_fram
        FF_to_FGR_index_offset = plays.iloc[list(FF_to_FGR_index.values.flatten())]
        FF_to_FGR_count =len(FF_to_FGR_index_offset.loc[FF_to_FGR_index_offset['Spot'] >= 80].i
        print('FF to FGR: {}'.format(FF_to_FGR_count))

        # Plays start in FF and end in RZ
        FF_to_RZ_index = plays.loc[(plays['Spot'] < 20) & (plays['Spot'] >= 0)].index.to_frame(
        FF_to_RZ_index_offset = plays.iloc[list(FF_to_RZ_index.values.flatten())]
        FF_to_RZ_count =len(FF_to_RZ_index_offset.loc[FF_to_RZ_index_offset['Spot'] >= 80].inde
        print('FF to RZ: {}'.format(FF_to_RZ_count))

        # max number of plays from FF
        # (total yards/10)*4
        FF_max_plays = 20/10*4

        FF_to_S_ratio = FF_to_S_count/FF_plays_start_count
        FF_to_TD_ratio = FF_to_TD_count/FF_plays_start_count
        FF_to_FG_ratio = FF_to_FG_count/FF_plays_start_count
        FF_to_MF_ratio = FF_to_MF_count/FF_plays_start_count
        FF_to_FGR_ratio = FF_to_FGR_count/FF_plays_start_count
        FF_to_RZ_ratio = FF_to_RZ_count/FF_plays_start_count

        FF_ratio_dict = {
            'FF_to_S': FF_to_S_ratio,
            'FF_to_TD':FF_to_TD_ratio,
            'FF_to_FG':FF_to_FG_ratio,
            'FF_to_MF':FF_to_MF_ratio,
            'FF_to_FGR':FF_to_FGR_ratio,
            'FF_to_RZ':FF_to_RZ_ratio,
            }
        FF_ratio_dict.update({'FF_to_FF': 1-sum(FF_ratio_dict.values())})
        print(FF_ratio_dict)

        plt.bar(range(len(FF_ratio_dict)), list(FF_ratio_dict.values()), align='center')
        plt.xticks(range(len(FF_ratio_dict)), list(FF_ratio_dict.keys()))
        # # for python 2.x:
        # plt.bar(range(len(D)), D.values(), align='center')  # python 2.x
        # plt.xticks(range(len(D)), D.keys())  # in python 2.x

        plt.show()
```

```
Total plays FF: 16224
FF to S: 50
FF to TD: 153
FF to FG: 1
FF to MF: 6277
FF to FGR: 824
FF to RZ: 334
```

```
{'FF_to_MF': 0.3868959566074951, 'FF_to_RZ': 0.020586785009861934, 'FF_to_FG': 6.16370808678501e
```

Out[20]:



$$\Pr(MF \mid FF) = \frac{6277}{16224} \approx 0.38689$$

$$\Pr(FGR \mid FF) = \frac{824}{16224} \approx 0.05078$$

$$\Pr(RZ \mid FF) = \frac{334}{16224} \approx 0.02058$$

$$\Pr(S \mid FF) = \frac{50}{16224} \approx 0.00308$$

$$\Pr(TD \mid FF) = \frac{153}{16224} \approx 0.00943$$

$$\Pr(FG \mid FF) = \frac{1}{16224} \approx 0.00001$$

$$\Pr(FF \mid FF) = \frac{8585}{16224} \approx 0.52915$$

This makes intuitive sense. It is rare for a touchdown to occur from this far away from the goal. It is more probable that a safety will occur and almost impossible to kick a field goal.

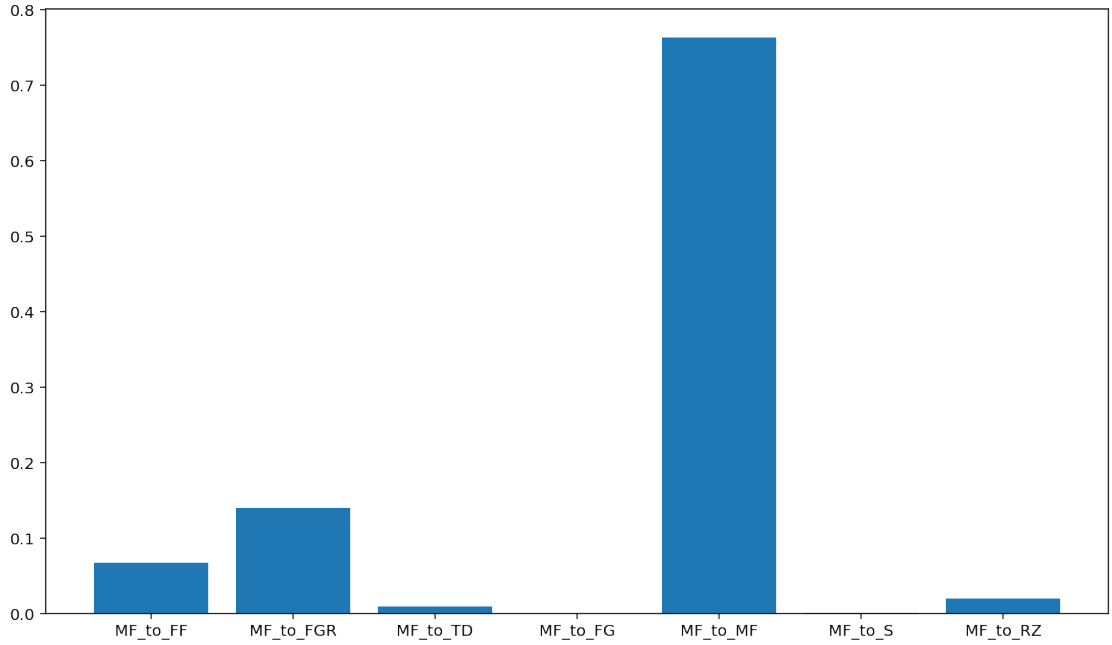
Transition Probabilities from MF

In [23]:
```
# Transition Probability for plays in Mid Field (MF)
MF_plays_start = plays.loc[(plays['Spot'] < 80) & (plays['Spot'] >= 50)]
MF_plays_start_count = len(plays.loc[(plays['Spot'] < 80) & (plays['Spot'] >= 50)].inde
print('Total plays MF: {}'.format(MF_plays_start_count))

# plays start in MF and end in an S
MF_to_S_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] == 80)].in
MF_to_S_index_offset = plays.iloc[list(MF_to_S_index.values.flatten())]
MF_to_S_count =len(MF_to_S_index_offset.loc[(MF_to_S_index_offset['Spot'] < 80) & (MF_t
print('MF to S: {}'.format(MF_to_S_count))
#FF_to_S = plays.iloc(i-1) for i in plays.loc[(plays['Play Type'] == 'KICKOFF') & (play

# plays start in MF and end in a TD
MF_to_TD_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].i
MF_to_TD_index_offset_is_TD = plays.iloc[list(MF_to_TD_index.values.flatten())]
MF_to_TD_index_is_TD = MF_to_TD_index_offset_is_TD.loc[(plays['Play Type'] == 'ATTEMPT'
MF_to_TD_index_offset_zone = plays.iloc[list(MF_to_TD_index_is_TD.values.flatten())]
MF_to_TD_count =len(MF_to_TD_index_offset_zone.loc[(plays['Spot'] < 80) & (plays['Spot'
print('MF to TD: {}'.format(MF_to_TD_count))

# plays start in MF and end in a FG
MF_to_FG_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].i
MF_to_FG_index_offset_is_FG = plays.iloc[list(MF_to_FG_index.values.flatten())]
MF_to_FG_index_is_FG = MF_to_FG_index_offset_is_FG.loc[(plays['Play Type'] == 'FIELD_GO
MF_to_FG_index_offset_zone = plays.iloc[list(MF_to_FG_index_is_FG.values.flatten())]
MF_to_FG_count =len(MF_to_FG_index_offset_zone.loc[(plays['Spot'] < 80) & (plays['Spot'
print('MF to FG: {}'.format(MF_to_FG_count))

# Plays start in MF and end in FF
MF_to_FF_index = plays.loc[(plays['Spot'] >= 80)].index.to_frame().apply(lambda x: x-1)
MF_to_FF_index_offset = plays.iloc[list(MF_to_FF_index.values.flatten())]
MF_to_FF_count =len(MF_to_FF_index_offset.loc[(MF_to_FF_index_offset['Spot'] < 80) & (M
print('MF to FF: {}'.format(MF_to_FF_count))

# Plays start in MF and end in FGR
MF_to_FGR_index = plays.loc[(plays['Spot'] < 50) & (plays['Spot'] >= 20)].index.to_fram
MF_to_FGR_index_offset = plays.iloc[list(MF_to_FGR_index.values.flatten())]
MF_to_FGR_count =len(MF_to_FGR_index_offset.loc[(MF_to_FGR_index_offset['Spot'] < 80) &
print('MF to FGR: {}'.format(MF_to_FGR_count))

# Plays start in FF and end in RZ
MF_to_RZ_index = plays.loc[(plays['Spot'] < 20) & (plays['Spot'] >= 0)].index.to_frame(
MF_to_RZ_index_offset = plays.iloc[list(MF_to_RZ_index.values.flatten())]
MF_to_RZ_count =len(MF_to_RZ_index_offset.loc[(MF_to_RZ_index_offset['Spot'] < 80) & (M
print('MF to RZ: {}'.format(MF_to_RZ_count))
```

```
        # max number of plays from FF
        # (total yards/10)*4
        MF_max_plays = 30/10*4

        MF_to_S_ratio = MF_to_S_count/MF_plays_start_count
        MF_to_TD_ratio = MF_to_TD_count/MF_plays_start_count
        MF_to_FG_ratio = MF_to_FG_count/MF_plays_start_count
        MF_to_FF_ratio = MF_to_FF_count/MF_plays_start_count
        MF_to_FGR_ratio = MF_to_FGR_count/MF_plays_start_count
        MF_to_RZ_ratio = MF_to_RZ_count/MF_plays_start_count

        MF_ratio_dict = {
            'MF_to_S': MF_to_S_ratio,
            'MF_to_TD':MF_to_TD_ratio,
            'MF_to_FG':MF_to_FG_ratio,
            'MF_to_FF':MF_to_FF_ratio,
            'MF_to_FGR':MF_to_FGR_ratio,
            'MF_to_RZ':MF_to_RZ_ratio,
            }
        MF_ratio_dict.update({'MF_to_MF': 1-sum(MF_ratio_dict.values())})
        print(MF_ratio_dict)

        plt.bar(range(len(MF_ratio_dict)), list(MF_ratio_dict.values()), align='center')
        plt.xticks(range(len(MF_ratio_dict)), list(MF_ratio_dict.keys()))
        # # for python 2.x:
        # plt.bar(range(len(D)), D.values(), align='center')  # python 2.x
        # plt.xticks(range(len(D)), D.keys())  # in python 2.x

        plt.show()
```

```
Total plays MF: 75529
MF to S: 58
MF to TD: 669
MF to FG: 3
MF to FF: 5086
MF to FGR: 10575
MF to RZ: 1482
{'MF_to_FF': 0.06733837334004157, 'MF_to_FGR': 0.14001244555071563, 'MF_to_TD': 0.00885752492420
```

Out[23]:



$$\Pr(FF \mid MF) = \frac{5086}{75529} \approx 0.06733$$

$$\Pr(FGR \mid MF) = \frac{10575}{75529} \approx 0.14001$$

$$\Pr(RZ \mid MF) = \frac{1482}{75529} \approx 0.01962$$

$$\Pr(S \mid MF) = \frac{58}{75529} \approx 0.00076$$

$$\Pr(TD \mid MF) = \frac{669}{75529} \approx 0.00885$$

$$\Pr(FG \mid MF) = \frac{3}{75529} \approx 0.00004$$

$$\Pr(MF \mid MF) = \frac{57656}{75529} \approx 0.76336$$

Transition Probabilities from FGR

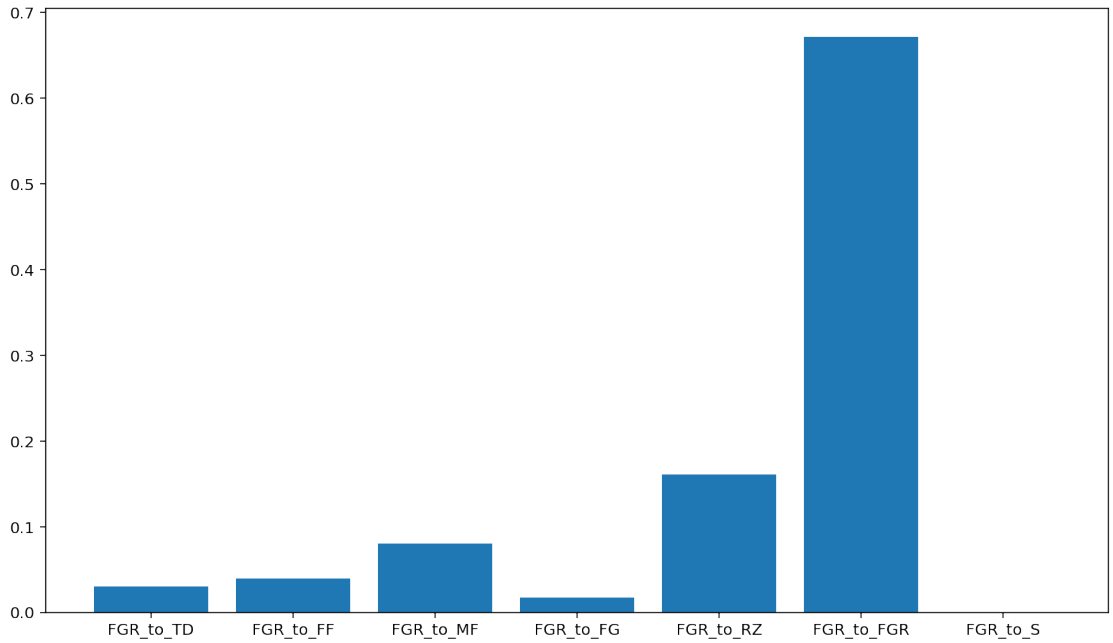
```
In [28]: # Transition Probability for plays in Field Goal Range (FGR)
         FGR_plays_start = plays.loc[(plays['Spot'] < 50) & (plays['Spot'] >= 20)]
         FGR_plays_start_count = len(plays.loc[(plays['Spot'] < 50) & (plays['Spot'] >= 20)].ind
         print('Total plays FGR: {}'.format(FGR_plays_start_count))

         # plays start in FGR and end in an S
         FGR_to_S_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] == 80)].i
         FGR_to_S_index_offset = plays.iloc[list(FGR_to_S_index.values.flatten())]
```

```
FGR_to_S_count =len(FGR_to_S_index_offset.loc[(FGR_to_S_index_offset['Spot'] < 50) & (M
print('FGR to S: {}'.format(FGR_to_S_count))
#FF_to_S = plays.iloc(i-1) for i in plays.loc[(plays['Play Type'] == 'KICKOFF') & (play

# plays start in FGR and end in a TD
FGR_to_TD_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].
FGR_to_TD_index_offset_is_TD = plays.iloc[list(FGR_to_TD_index.values.flatten())]
FGR_to_TD_index_is_TD = FGR_to_TD_index_offset_is_TD.loc[(plays['Play Type'] == 'ATTEMP
FGR_to_TD_index_offset_zone = plays.iloc[list(FGR_to_TD_index_is_TD.values.flatten())]
FGR_to_TD_count =len(FGR_to_TD_index_offset_zone.loc[(plays['Spot'] < 50) & (plays['Spo
print('FGR to TD: {}'.format(FGR_to_TD_count))

# plays start in FGR and end in a FG
FGR_to_FG_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].
FGR_to_FG_index_offset_is_FG = plays.iloc[list(FGR_to_FG_index.values.flatten())]
FGR_to_FG_index_is_FG = FGR_to_FG_index_offset_is_FG.loc[(plays['Play Type'] == 'FIELD_
FGR_to_FG_index_offset_zone = plays.iloc[list(FGR_to_FG_index_is_FG.values.flatten())]
FGR_to_FG_count =len(FGR_to_FG_index_offset_zone.loc[(plays['Spot'] < 50) & (plays['Spo
print('FGR to FG: {}'.format(FGR_to_FG_count))

# Plays start in FGR and end in FF
FGR_to_FF_index = plays.loc[(plays['Spot'] >= 80)].index.to_frame().apply(lambda x: x-1
FGR_to_FF_index_offset = plays.iloc[list(FGR_to_FF_index.values.flatten())]
FGR_to_FF_count =len(FGR_to_FF_index_offset.loc[(FGR_to_FF_index_offset['Spot'] < 50) &
print('FGR to FF: {}'.format(FGR_to_FF_count))

# Plays start in FGR and end in MF
FGR_to_MF_index = plays.loc[(plays['Spot'] < 80) & (plays['Spot'] >= 50)].index.to_fram
FGR_to_MF_index_offset = plays.iloc[list(FGR_to_MF_index.values.flatten())]
FGR_to_MF_count =len(FGR_to_MF_index_offset.loc[(FGR_to_MF_index_offset['Spot'] < 50) &
print('FGR to MF: {}'.format(FGR_to_MF_count))

# Plays start in FGR and end in RZ
FGR_to_RZ_index = plays.loc[(plays['Spot'] < 20) & (plays['Spot'] >= 0)].index.to_frame
FGR_to_RZ_index_offset = plays.iloc[list(FGR_to_RZ_index.values.flatten())]
FGR_to_RZ_count =len(FGR_to_RZ_index_offset.loc[(FGR_to_RZ_index_offset['Spot'] < 50) &
print('FGR to RZ: {}'.format(FGR_to_RZ_count))

# max number of plays from FF
# (total yards/10)*4
FGR_max_plays = 30/10*4

FGR_to_S_ratio = FGR_to_S_count/FGR_plays_start_count
FGR_to_TD_ratio = FGR_to_TD_count/FGR_plays_start_count
FGR_to_FG_ratio = FGR_to_FG_count/FGR_plays_start_count
FGR_to_FF_ratio = FGR_to_FF_count/FGR_plays_start_count
FGR_to_MF_ratio = FGR_to_MF_count/FGR_plays_start_count
FGR_to_RZ_ratio = FGR_to_RZ_count/FGR_plays_start_count
```

```
        FGR_ratio_dict = {
            'FGR_to_S': FGR_to_S_ratio,
            'FGR_to_TD':FGR_to_TD_ratio,
            'FGR_to_FG':FGR_to_FG_ratio,
            'FGR_to_FF':FGR_to_FF_ratio,
            'FGR_to_MF':FGR_to_MF_ratio,
            'FGR_to_RZ':FGR_to_RZ_ratio,
            }
        FGR_ratio_dict.update({'FGR_to_FGR': 1-sum(FGR_ratio_dict.values())})
        print(FGR_ratio_dict)

        plt.bar(range(len(FGR_ratio_dict)), list(FGR_ratio_dict.values()), align='center')
        plt.xticks(range(len(FGR_ratio_dict)), list(FGR_ratio_dict.keys()))

        plt.show()
```

```
Total plays FGR: 42911
FGR to S: 0
FGR to TD: 1314
FGR to FG: 728
FGR to FF: 1690
FGR to MF: 3442
FGR to RZ: 6909
{'FGR_to_TD': 0.03062151895784298, 'FGR_to_FF': 0.03938384097317704, 'FGR_to_MF': 0.080212532916
```

Out[28]:



$$\Pr(FF \mid FGR) = \frac{1690}{42911} \approx 0.03938$$

$$\Pr(MF \mid FGR) = \frac{3442}{42911} \approx 0.08021$$

$$\Pr(RZ \mid FGR) = \frac{6909}{42911} \approx 0.16100$$

$$\Pr(S \mid FGR) = \frac{0}{42911} \approx 0$$

$$\Pr(TD \mid FGR) = \frac{1314}{42911} \approx 0.03062$$

$$\Pr(FG \mid FGR) = \frac{728}{42911} \approx 0.01696$$

$$\Pr(FGR \mid FGR) = \frac{28828}{42911} \approx 0.67180$$

Transition Probabilities from RZ

In [29]:
```
# Transition Probability for plays in Red Zone (RZ)
RZ_plays_start = plays.loc[(plays['Spot'] < 20) & (plays['Spot'] >= 0)]
RZ_plays_start_count = len(plays.loc[(plays['Spot'] < 20) & (plays['Spot'] >= 0)].index
print('Total plays RZ: {}'.format(RZ_plays_start_count))

# plays start in RZ and end in an S
RZ_to_S_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] == 80)].in
RZ_to_S_index_offset = plays.iloc[list(RZ_to_S_index.values.flatten())]
RZ_to_S_count =len(RZ_to_S_index_offset.loc[(RZ_to_S_index_offset['Spot'] < 20) & (MF_t
print('RZ to S: {}'.format(RZ_to_S_count))
#FF_to_S = plays.iloc(i-1) for i in plays.loc[(plays['Play Type'] == 'KICKOFF') & (play

# plays start in RZ and end in a TD
RZ_to_TD_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].i
RZ_to_TD_index_offset_is_TD = plays.iloc[list(RZ_to_TD_index.values.flatten())]
RZ_to_TD_index_is_TD = RZ_to_TD_index_offset_is_TD.loc[(plays['Play Type'] == 'ATTEMPT'
RZ_to_TD_index_offset_zone = plays.iloc[list(RZ_to_TD_index_is_TD.values.flatten())]
RZ_to_TD_count =len(RZ_to_TD_index_offset_zone.loc[(plays['Spot'] < 20) & (plays['Spot'
print('RZ to TD: {}'.format(RZ_to_TD_count))

# plays start in RZ and end in a FG
RZ_to_FG_index = plays.loc[(plays['Play Type'] == 'KICKOFF') & (plays['Spot'] != 80)].i
RZ_to_FG_index_offset_is_FG = plays.iloc[list(RZ_to_FG_index.values.flatten())]
RZ_to_FG_index_is_FG = RZ_to_FG_index_offset_is_FG.loc[(plays['Play Type'] == 'FIELD_GO
RZ_to_FG_index_offset_zone = plays.iloc[list(RZ_to_FG_index_is_FG.values.flatten())]
RZ_to_FG_count =len(RZ_to_FG_index_offset_zone.loc[(plays['Spot'] < 20) & (plays['Spot'
print('RZ to FG: {}'.format(RZ_to_FG_count))
```

```
        # Plays start in MF and end in FF
        RZ_to_FF_index = plays.loc[(plays['Spot'] >= 80)].index.to_frame().apply(lambda x: x-1)
        RZ_to_FF_index_offset = plays.iloc[list(RZ_to_FF_index.values.flatten())]
        RZ_to_FF_count =len(RZ_to_FF_index_offset.loc[(RZ_to_FF_index_offset['Spot'] < 20) & (R
        print('RZ to FF: {}'.format(RZ_to_FF_count))

        # Plays start in RZ and end in FGR
        RZ_to_FGR_index = plays.loc[(plays['Spot'] < 50) & (plays['Spot'] >= 20)].index.to_fram
        RZ_to_FGR_index_offset = plays.iloc[list(RZ_to_FGR_index.values.flatten())]
        RZ_to_FGR_count =len(RZ_to_FGR_index_offset.loc[(RZ_to_FGR_index_offset['Spot'] < 20) &
        print('RZ to FGR: {}'.format(RZ_to_FGR_count))

        # Plays start in RZ and end in MF
        RZ_to_MF_index = plays.loc[(plays['Spot'] < 80) & (plays['Spot'] >= 50)].index.to_frame
        RZ_to_MF_index_offset = plays.iloc[list(RZ_to_MF_index.values.flatten())]
        RZ_to_MF_count =len(RZ_to_MF_index_offset.loc[(RZ_to_MF_index_offset['Spot'] < 20) & (R
        print('RZ to MF: {}'.format(RZ_to_MF_count))

        # max number of plays from FF
        # (total yards/10)*4
        RZ_max_plays = 20/10*4

        RZ_to_S_ratio = RZ_to_S_count/RZ_plays_start_count
        RZ_to_TD_ratio = RZ_to_TD_count/RZ_plays_start_count
        RZ_to_FG_ratio = RZ_to_FG_count/RZ_plays_start_count
        RZ_to_FF_ratio = RZ_to_FF_count/RZ_plays_start_count
        RZ_to_FGR_ratio = RZ_to_FGR_count/RZ_plays_start_count
        RZ_to_MF_ratio = RZ_to_MF_count/RZ_plays_start_count

        RZ_ratio_dict = {
            'RZ_to_S': RZ_to_S_ratio,
            'RZ_to_TD':RZ_to_TD_ratio,
            'RZ_to_FG':RZ_to_FG_ratio,
            'RZ_to_FF':RZ_to_FF_ratio,
            'RZ_to_FGR':RZ_to_FGR_ratio,
            'RZ_to_MF':RZ_to_MF_ratio,
            }
        RZ_ratio_dict.update({'RZ_to_RZ': 1-sum(RZ_ratio_dict.values())})
        print(RZ_ratio_dict)

        # plt.bar(range(len(RZ_ratio_dict)), list(RZ_ratio_dict.values()), align='center')
        # plt.xticks(range(len(RZ_ratio_dict)), list(RZ_ratio_dict.keys()))

        # plt.show()

Total plays RZ: 26033
RZ to S: 0
```

```
RZ to TD: 3762
RZ to FG: 1113
RZ to FF: 659
RZ to FGR: 642
RZ to MF: 7424
{'RZ_to_RZ': 0.4775861406676142, 'RZ_to_FGR': 0.024661007183190566, 'RZ_to_S': 0.0, 'RZ_to_FF':
```

$$\Pr(FF \mid RZ) = \frac{659}{26033} \approx 0.02531$$
$$\Pr(MF \mid RZ) = \frac{7424}{26033} \approx 0.28517$$
$$\Pr(FGR \mid RZ) = \frac{642}{26033} \approx 0.02466$$
$$\Pr(S \mid RZ) = \frac{0}{26033} \approx 0$$
$$\Pr(TD \mid RZ) = \frac{3762}{26033} \approx 0.14450$$
$$\Pr(FG \mid RZ) = \frac{1113}{26033} \approx 0.04275$$
$$\Pr(RZ \mid RZ) = \frac{12433}{26033} \approx 0.47758$$

Transition Probabilities from S, TD, FG After scoring plays or a punt, the state is reset by a constant state to represent a kickoff or punt.

FF	MF	FGR	RZ	TD
.5	.35	.1	.035	.015

```
In [31]: states = ['FF', 'MF', 'FGR', 'RZ', 'S', 'FG', 'TD']
         pi = [0.5, 0.35, 0.1, .035, 0, 0, .015]
         state_space = pd.Series(pi, index=states, name='states')
         print(state_space)
         print(state_space.sum())
```

```
FF     0.500
MF     0.350
FGR    0.100
RZ     0.035
S      0.000
FG     0.000
TD     0.015
Name: states, dtype: float64
1.0
```

1.6 Model Implementation

1.6.1 Create Graph

We construct a graph with nodes that represent states and edges with weights that represent the probability of transitioning to a different state.

```
In [32]: q_df = pd.DataFrame(columns=states, index=states)
         q_df.loc[states[0]] = [
             FF_ratio_dict['FF_to_FF'],
             FF_ratio_dict['FF_to_MF'],
             FF_ratio_dict['FF_to_FGR'],
             FF_ratio_dict['FF_to_RZ'],
             FF_ratio_dict['FF_to_S'],
             FF_ratio_dict['FF_to_FG'],
             FF_ratio_dict['FF_to_TD']
             ]
         q_df.loc[states[1]] = [
             MF_ratio_dict['MF_to_FF'],
             MF_ratio_dict['MF_to_MF'],
             MF_ratio_dict['MF_to_FGR'],
             MF_ratio_dict['MF_to_RZ'],
             MF_ratio_dict['MF_to_S'],
             MF_ratio_dict['MF_to_FG'],
             MF_ratio_dict['MF_to_TD']
             ]
         q_df.loc[states[2]] = [
             FGR_ratio_dict['FGR_to_FF'],
             FGR_ratio_dict['FGR_to_MF'],
             FGR_ratio_dict['FGR_to_FGR'],
             FGR_ratio_dict['FGR_to_RZ'],
             FGR_ratio_dict['FGR_to_S'],
             FGR_ratio_dict['FGR_to_FG'],
             FGR_ratio_dict['FGR_to_TD']
             ]
         q_df.loc[states[3]] = [
             RZ_ratio_dict['RZ_to_FF'],
             RZ_ratio_dict['RZ_to_MF'],
             RZ_ratio_dict['RZ_to_FGR'],
             RZ_ratio_dict['RZ_to_RZ'],
             RZ_ratio_dict['RZ_to_S'],
             RZ_ratio_dict['RZ_to_FG'],
             RZ_ratio_dict['RZ_to_TD']
             ]
         q_df.loc[states[4]] = pi
         q_df.loc[states[5]] = pi
         q_df.loc[states[6]] = pi

         print(q_df)
```

```
        q = q_df.values
        print('\n', q, q.shape, '\n')
        print(q_df.sum(axis=1))
```

```
            FF         MF       FGR         RZ            S           FG  \
FF    0.529154   0.386896  0.050789  0.0205868   0.00308185  6.16371e-05
MF   0.0673384   0.763362  0.140012  0.0196216  0.000767917  3.97198e-05
FGR  0.0393838  0.0802125  0.671809   0.161008            0    0.0169653
RZ    0.025314   0.285177  0.024661   0.477586            0    0.0427534
S          0.5       0.35       0.1      0.035            0            0
FG         0.5       0.35       0.1      0.035            0            0
TD         0.5       0.35       0.1      0.035            0            0

             TD
FF   0.00943047
MF   0.00885752
FGR   0.0306215
RZ     0.144509
S          0.015
FG         0.015
TD         0.015

 [[0.529154339250493 0.3868959566074951 0.050788954635108484
  0.020586785009861934 0.003081854043392505 6.16370808678501e-05
  0.009430473372781065]
 [0.06733837334004157 0.7633624170848283 0.14001244555071563
  0.019621602298454898 0.0007679169590488422 3.971984270942287e-05
  0.0088575249242013]
 [0.03938384097317704 0.08021253291696767 0.6718090932394957
  0.16100766703176342 0.0 0.01696534688075319 0.03062151895784298]
 [0.025314024507356047 0.28517650674144357 0.024661007183190566
  0.4775861406676142 0.0 0.04275342834095187 0.1445088925594438]
 [0.5 0.35 0.1 0.035 0 0 0.015]
 [0.5 0.35 0.1 0.035 0 0 0.015]
 [0.5 0.35 0.1 0.035 0 0 0.015]] (7, 7)

FF     1.0
MF     1.0
FGR    1.0
RZ     1.0
S      1.0
FG     1.0
TD     1.0
dtype: float64
```

```
In [35]: from pprint import pprint
```

```
        # create a function that maps transition probability dataframe
        # to markov edges and weights

        def _get_markov_edges(Q):
            edges = []
            for col in Q.columns:
                for idx in Q.index:
                    edges.append((idx,col,str(round(Q.loc[idx,col], 4))))
            return edges

        edges_wts = _get_markov_edges(q_df)
        pprint(edges_wts)
```

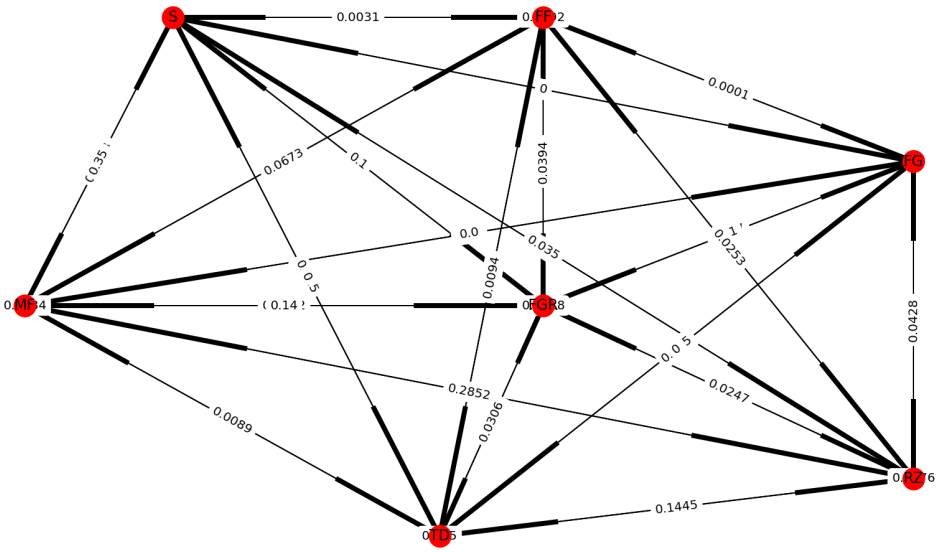
```
[('FF', 'FF', '0.5292'),
 ('MF', 'FF', '0.0673'),
 ('FGR', 'FF', '0.0394'),
 ('RZ', 'FF', '0.0253'),
 ('S', 'FF', '0.5'),
 ('FG', 'FF', '0.5'),
 ('TD', 'FF', '0.5'),
 ('FF', 'MF', '0.3869'),
 ('MF', 'MF', '0.7634'),
 ('FGR', 'MF', '0.0802'),
 ('RZ', 'MF', '0.2852'),
 ('S', 'MF', '0.35'),
 ('FG', 'MF', '0.35'),
 ('TD', 'MF', '0.35'),
 ('FF', 'FGR', '0.0508'),
 ('MF', 'FGR', '0.14'),
 ('FGR', 'FGR', '0.6718'),
 ('RZ', 'FGR', '0.0247'),
 ('S', 'FGR', '0.1'),
 ('FG', 'FGR', '0.1'),
 ('TD', 'FGR', '0.1'),
 ('FF', 'RZ', '0.0206'),
 ('MF', 'RZ', '0.0196'),
 ('FGR', 'RZ', '0.161'),
 ('RZ', 'RZ', '0.4776'),
 ('S', 'RZ', '0.035'),
 ('FG', 'RZ', '0.035'),
 ('TD', 'RZ', '0.035'),
 ('FF', 'S', '0.0031'),
 ('MF', 'S', '0.0008'),
 ('FGR', 'S', '0.0'),
 ('RZ', 'S', '0.0'),
 ('S', 'S', '0'),
 ('FG', 'S', '0'),
 ('TD', 'S', '0'),
```

```
 ('FF', 'FG', '0.0001'),
 ('MF', 'FG', '0.0'),
 ('FGR', 'FG', '0.017'),
 ('RZ', 'FG', '0.0428'),
 ('S', 'FG', '0'),
 ('FG', 'FG', '0'),
 ('TD', 'FG', '0'),
 ('FF', 'TD', '0.0094'),
 ('MF', 'TD', '0.0089'),
 ('FGR', 'TD', '0.0306'),
 ('RZ', 'TD', '0.1445'),
 ('S', 'TD', '0.015'),
 ('FG', 'TD', '0.015'),
 ('TD', 'TD', '0.015')]
```

```
In [36]: # create graph object
         DG = nx.DiGraph()
         DG.add_weighted_edges_from(edges_wts)
         # print(DG.nodes())
         pos = {'FF':(0.5,1), 'MF':(0.15,.5), 'FGR':(0.5,0.5), 'RZ':(0.75,.2), 'S':(0.25,1), 'FG
         nx.draw(DG, pos, with_labels=True)
         labels = nx.get_edge_attributes(DG,'weight')
         # print(labels)
         nx.draw_networkx_edge_labels(DG,pos,edge_labels=labels)
         # labels = nx.get_edge_attributes(G,'weight')
         # nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)

Out[36]: {('FF', 'FF'): Text(0.5,1,'0.5292'),
          ('FF', 'FG'): Text(0.625,0.875,'0.0001'),
          ('FF', 'FGR'): Text(0.5,0.75,'0.0508'),
          ('FF', 'MF'): Text(0.325,0.75,'0.3869'),
          ('FF', 'RZ'): Text(0.625,0.6,'0.0206'),
          ('FF', 'S'): Text(0.375,1,'0.0031'),
          ('FF', 'TD'): Text(0.465,0.55,'0.0094'),
          ('FG', 'FF'): Text(0.625,0.875,'0.5'),
          ('FG', 'FG'): Text(0.75,0.75,'0'),
          ('FG', 'FGR'): Text(0.625,0.625,'0.1'),
          ('FG', 'MF'): Text(0.45,0.625,'0.35'),
          ('FG', 'RZ'): Text(0.75,0.475,'0.035'),
          ('FG', 'S'): Text(0.5,0.875,'0'),
          ('FG', 'TD'): Text(0.59,0.425,'0.015'),
          ('FGR', 'FF'): Text(0.5,0.75,'0.0394'),
          ('FGR', 'FG'): Text(0.625,0.625,'0.017'),
          ('FGR', 'FGR'): Text(0.5,0.5,'0.6718'),
          ('FGR', 'MF'): Text(0.325,0.5,'0.0802'),
          ('FGR', 'RZ'): Text(0.625,0.35,'0.161'),
          ('FGR', 'S'): Text(0.375,0.75,'0.0'),
```

```
('FGR', 'TD'): Text(0.465,0.3,'0.0306'),
('MF', 'FF'): Text(0.325,0.75,'0.0673'),
('MF', 'FG'): Text(0.45,0.625,'0.0'),
('MF', 'FGR'): Text(0.325,0.5,'0.14'),
('MF', 'MF'): Text(0.15,0.5,'0.7634'),
('MF', 'RZ'): Text(0.45,0.35,'0.0196'),
('MF', 'S'): Text(0.2,0.75,'0.0008'),
('MF', 'TD'): Text(0.29,0.3,'0.0089'),
('RZ', 'FF'): Text(0.625,0.6,'0.0253'),
('RZ', 'FG'): Text(0.75,0.475,'0.0428'),
('RZ', 'FGR'): Text(0.625,0.35,'0.0247'),
('RZ', 'MF'): Text(0.45,0.35,'0.2852'),
('RZ', 'RZ'): Text(0.75,0.2,'0.4776'),
('RZ', 'S'): Text(0.5,0.6,'0.0'),
('RZ', 'TD'): Text(0.59,0.15,'0.1445'),
('S', 'FF'): Text(0.375,1,'0.5'),
('S', 'FG'): Text(0.5,0.875,'0'),
('S', 'FGR'): Text(0.375,0.75,'0.1'),
('S', 'MF'): Text(0.2,0.75,'0.35'),
('S', 'RZ'): Text(0.5,0.6,'0.035'),
('S', 'S'): Text(0.25,1,'0'),
('S', 'TD'): Text(0.34,0.55,'0.015'),
('TD', 'FF'): Text(0.465,0.55,'0.5'),
('TD', 'FG'): Text(0.59,0.425,'0'),
('TD', 'FGR'): Text(0.465,0.3,'0.1'),
('TD', 'MF'): Text(0.29,0.3,'0.35'),
('TD', 'RZ'): Text(0.59,0.15,'0.035'),
('TD', 'S'): Text(0.34,0.55,'0'),
('TD', 'TD'): Text(0.43,0.1,'0.015')}
```

Out[36]:



1.6.2 Manage State Transitions

In order to transition from one state to another, we need to randomly choose a node to go to next. If the node randomly chooses the same node, it represents that the ball did not move far enough to change states and needs to check again. This helps simulate the 1st, 2nd, 3rd, and 4th down. If a different state is chosen, then a random number of plays is chosen between 1 and max_steps and returned as the number of plays it took to change states.

```
In [46]: def return_time(G,start,max_steps=8):
             '''
             Given a graph G and a starting node 'start', simulates a
             random walk for a football game drive. The function will return
             the total number of steps taken.

             If the simulation requires more than max_steps iterations,
             the team turns the ball over and the ball switches possesion.
             '''

             import numpy as np

             # initialize the starting node
             cur = start

             # don't do more than max_steps
             for i in range(max_steps):
```

```
            # take a step
            trasition_mat = [(k[1],v) for k,v in nx.get_edge_attributes(G,'weight').items()
            names = [i[0] for i in trasition_mat]
            probs = [float(i[1]) for i in trasition_mat]
            out = np.random.choice(names,1, probs)

            # stop if we've returned
            if str(out[0]) == 'S' or str(out[0]) == 'FG' or str(out[0]) == 'TD':
                if out != start and str(out[0]) != 'S' and str(out[0]) != 'FG':
                    return out, rnd.randint(1, max_steps)
            else:
                return out, rnd.randint(1, max_steps)
#           elif str(out[0]) != start:


        # if we got here, it took too many steps
        return out,rnd.randint(1, max_steps)
```

1.6.3 Simulate Game Method

The simulate game method creates a sequence of state transitions (football drives) within the alloted plays. It keeps track of the statistics that correspond to what happens in the random state transitions. For simplicity, 188 plays is considered the amount of plays in a game and no two point conversions or penalties are considered.

```
In [47]: # print(len((DG.neighbors('TD'))))
         # print(nx.get_edge_attributes(DG,'weight'))
         def simulate_game():
             stats = {
                 't1TD': 0,
                 't2TD': 0,
                 't1S': 0,
                 't2S': 0,
                 't1FG': 0,
                 't2FG': 0,
                 't1TO': 0,
                 't2TO': 0,
                 'FF': 0,
                 'MF': 0,
                 'FGR': 0,
                 'RZ': 0,
                 'TD': 0,
                 'S': 0,
                 'FG': 0
             }
             avg_plays_in_game = 188 # via cell 27
             team_bool = True # True means 1 has ball, False means 2 has ball
             team1_score = 0
```

```
team2_score = 0
old_state = 'TD'
while avg_plays_in_game > 0:
    next_state, plays_taken = return_time(DG, old_state)
    stats[str(next_state[0])] = stats[str(next_state[0])] + 1
    if next_state == 'TD':
        avg_plays_in_game = avg_plays_in_game - plays_taken
        if team_bool:
            team1_score = team1_score + 7
            old_state = 'TD'
            team_bool = False
            stats['t1TD'] = stats['t1TD'] + 1
        else:
            team2_score = team2_score + 7
            old_state = 'TD'
            team_bool = True
            stats['t2TD'] = stats['t2TD'] + 1
    elif next_state == 'S':
        avg_plays_in_game = avg_plays_in_game - plays_taken
        if team_bool:
            team1_score = team1_score + 2
            old_state = 'S'
            team_bool = True
            stats['t1S'] = stats['t1S'] + 1
        else:
            team2_score = team2_score + 2
            old_state = 'S'
            team_bool = False
            stats['t2S'] = stats['t2S'] + 1
    elif next_state == 'FG':
        avg_plays_in_game = avg_plays_in_game - plays_taken
        if team_bool:
            team1_score = team1_score + 3
            old_state = 'FG'
            team_bool = False
            stats['t1FG'] = stats['t1FG'] + 1
        else:
            team2_score = team2_score + 3
            old_state = 'FG'
            team_bool = True
            stats['t2FG'] = stats['t2FG'] + 1
    elif next_state == old_state:
        avg_plays_in_game = avg_plays_in_game - plays_taken
        if team_bool:
            stats['t1TO'] = stats['t1TO'] + 1
        else:
            stats['t2TO'] = stats['t2TO'] + 1
        team_bool = ~team_bool
```

```
                old_state = 'TD'
            else:
                avg_plays_in_game = avg_plays_in_game - plays_taken
                old_state = next_state

        return (team1_score,team2_score,stats)
```

1.7 Results

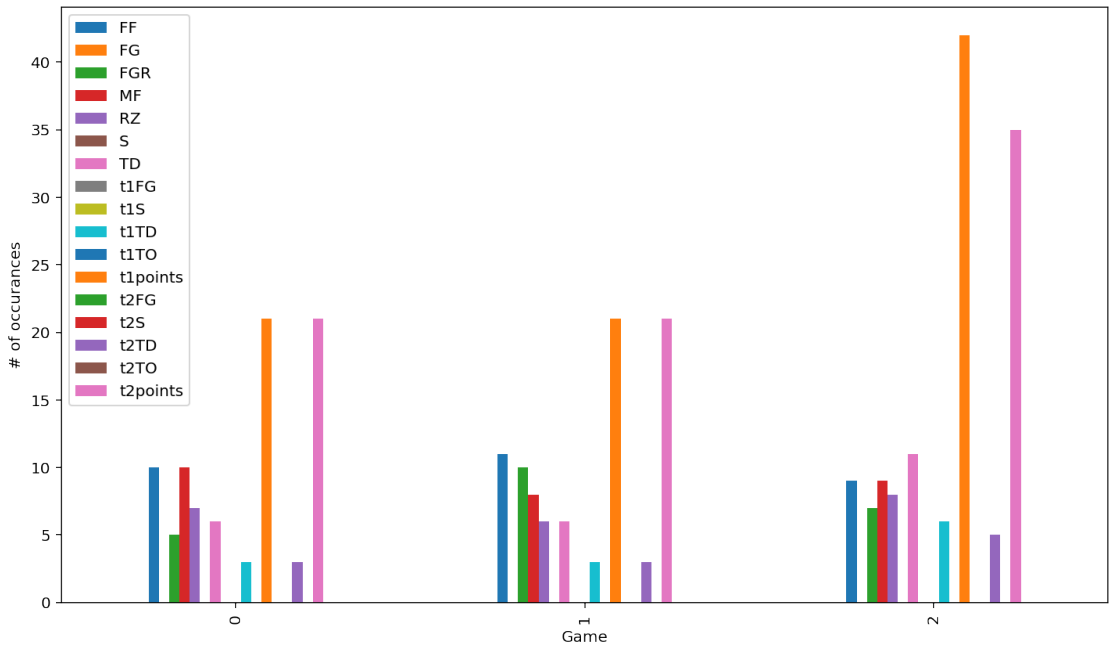
We simulate a 848 game season using our model and store all the statistics.

```
In [48]: stats = []
         for i in range(848):
             game_data = simulate_game()
             game_data[2].update({'t1points': game_data[0],'t2points':game_data[1]})
             stats.append(game_data[2])
         simulated_stats = pd.DataFrame(stats)
```

```
In [44]: simulated_stats.head(3).plot(kind='bar')
         plt.xlabel('Game')
         plt.ylabel('# of occurances')
```

```
Out[44]: Text(0,0.5,'# of occurances')
```

```
Out[44]:
```



```
In [49]: # Comparison
         print('Actual TD per game: {}'.format(team_game_stats['Rush TD'].mean() + team_game_sta
         print('Simulated TD per game: {}'.format((simulated_stats['t1TD'].mean() + simulated_st
         print('Actual FG per game: {}'.format(team_game_stats['Field Goal Made'].mean()))
         print('Simulated FG per game: {}'.format((simulated_stats['t1FG'].mean() + simulated_st
         print('Actual TO per game: {}'.format(team_game_stats['Punt'].mean()))
         print('Simulated TO per game: {}'.format((simulated_stats['t1TO'].mean() + simulated_st
         print('Actual RZ per game: {}'.format(team_game_stats['Red Zone Att'].mean()))
         print('Simulated RZ per game: {}'.format((simulated_stats['RZ'].mean() + simulated_stat
         print('Actual Points per game: {}'.format(team_game_stats['Points'].mean()))
         print('Simulated Points per game: {}'.format((simulated_stats['t1points'].mean() + simu
```

```
Actual TD per game: 3.389740566037736
Simulated TD per game: 2.992924528301887
Actual FG per game: 1.0731132075471699
Simulated FG per game: 0.001768867924528302
Actual TO per game: 5.105542452830188
Simulated TO per game: 3.0064858490566038
Actual RZ per game: 3.7570754716981134
Simulated RZ per game: 9.247641509433961
Actual Points per game: 28.54304245283019
Simulated Points per game: 20.961674528301884
```

1.8 Conclusion

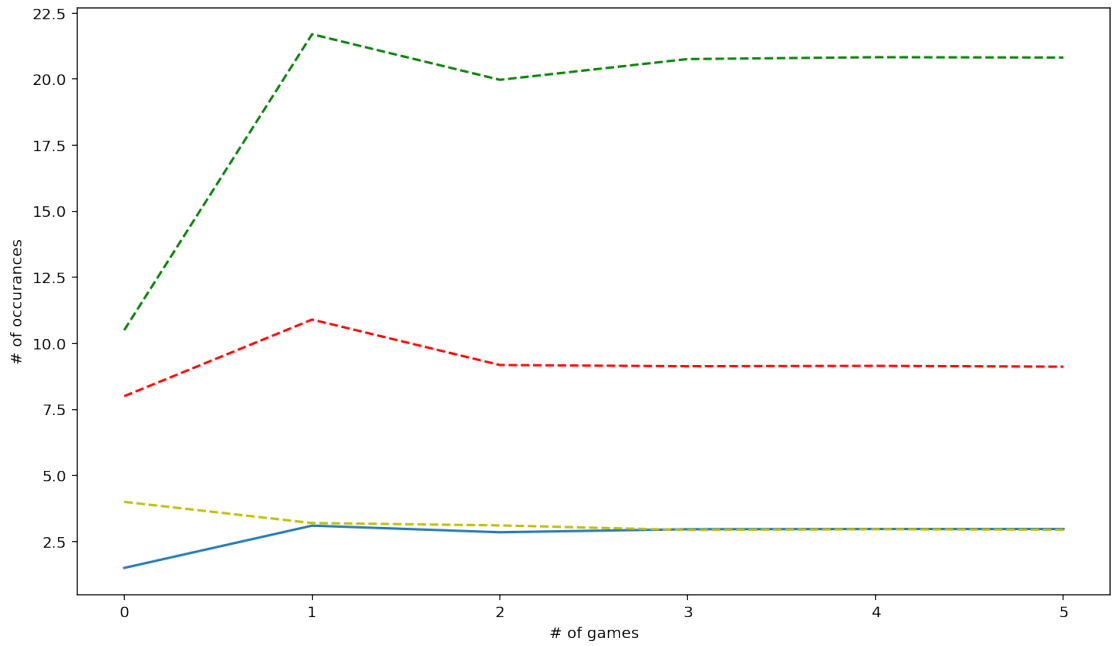
The model actually does a decent job of predicting the scores of games. Considering that different years have different averages, the TD per game is reasonable and Points per game is slightly low, but still reasonable. The number of FG per game is significantly lower than in actual games. FG are usually a given action on 4th down within the 50 yard line, but they can only occur on one play. There may be a discrepency in these numbers because there can be three plays in FGR to every one in FG. Since the model just picks an output and a number of plays it took, it does not have a frame of reference of a 4th down situation. The TO per game is also slightly lower. The number of RZ attempts is 3x higher in the simulated games than in actual games. The RZ would be about correct if there were about two more TO and one FG and one TD outside of the RZ.

```
In [60]: def sim(games):
             stats = []
             for i in range(games):
                 game_data = simulate_game()
                 game_data[2].update({'t1points': game_data[0],'t2points':game_data[1]})
                 stats.append(game_data[2])
             return pd.DataFrame(stats)
         converge = [sim(1), sim(10), sim(100), sim(500), sim(848), sim(1500)]

         td_conv = [((i['t1TD'].mean() + i['t2TD'].mean())/2) for i in converge]
         points_conv = [((i['t1points'].mean() + i['t2points'].mean())/2) for i in converge]
         to_conv = [((i['t1TO'].mean() + i['t2TO'].mean())/2) for i in converge]
```

```
rz_conv = [((i['RZ'].mean() + i['RZ'].mean())/2) for i in converge]

plt.plot([0,1,2,3,4,5], td_conv, [0,1,2,3,4,5], points_conv, 'g--', [0,1,2,3,4,5], to_c
plt.xlabel('# of games')
plt.ylabel('# of occurances')
# print('Actual TD per game: {}'.format(team_game_stats['Rush TD'].mean() + team_game_s
# print('Simulated TD per game: {}'.format((simulated_stats['t1TD'].mean() + simulated_
# print('Actual FG per game: {}'.format(team_game_stats['Field Goal Made'].mean()))
# print('Simulated FG per game: {}'.format((simulated_stats['t1FG'].mean() + simulated_
# print('Actual TO per game: {}'.format(team_game_stats['Punt'].mean()))
# print('Simulated TO per game: {}'.format((simulated_stats['t1TO'].mean() + simulated_
# print('Actual RZ per game: {}'.format(team_game_stats['Red Zone Att'].mean()))
# print('Simulated RZ per game: {}'.format((simulated_stats['RZ'].mean() + simulated_st
# print('Actual Points per game: {}'.format(team_game_stats['Points'].mean()))
# print('Simulated Points per game: {}'.format((simulated_stats['t1points'].mean() + si
```

Out[60]: Text(0,0.5,'# of occurances')

Out[60]:



The results usually converge after approximately 500 games for most statistics. Can a college football game be predicted using probability and and networks? It can with some degree of accuracy. The markov chain prediction method may not be the most effective model as it assumes that the next play has no reliance on the previous play. In football, there is momemtum, confidence, and nerves that all effect plays. There is also wind and precipitation which play a role in the type of plays of a game. Our model is simple in that it assumes state to state transitions always have the same probability.

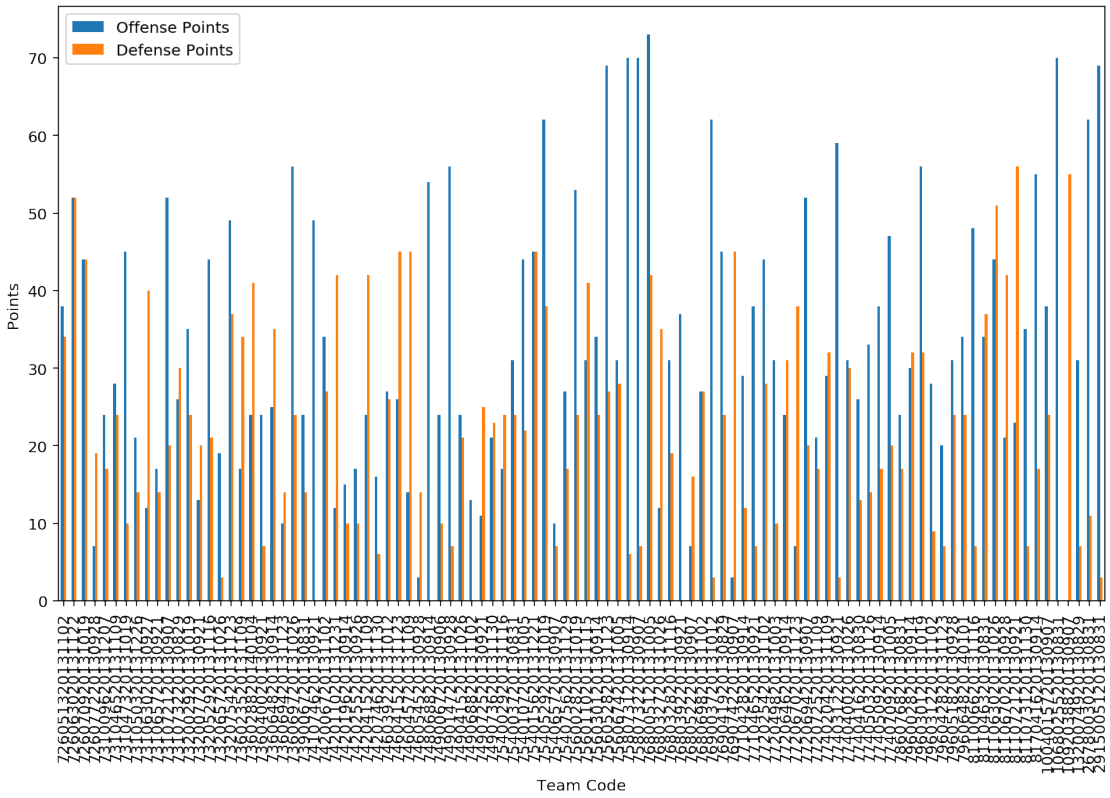
We see that by using a markov network with random walks and monte carlo simulations that we can create a model that somewhat resembles college football games.

1.9 Extra

In [10]:
```
# Look at the W/L ratio of 125 (good) teams vs. 82 (bad) teams
games_with_teams = plays.groupby(['Game Code']).last()[['Offense Team Code', 'Offense P
games_with_teams[['Offense Points', 'Defense Points']].tail(100).plot(kind='bar')
plt.xlabel('Team Code')
plt.ylabel('Points')
#print(games_with_teams)
```

Out[10]:
```
Text(0,0.5,'Points')
```

Out[10]:



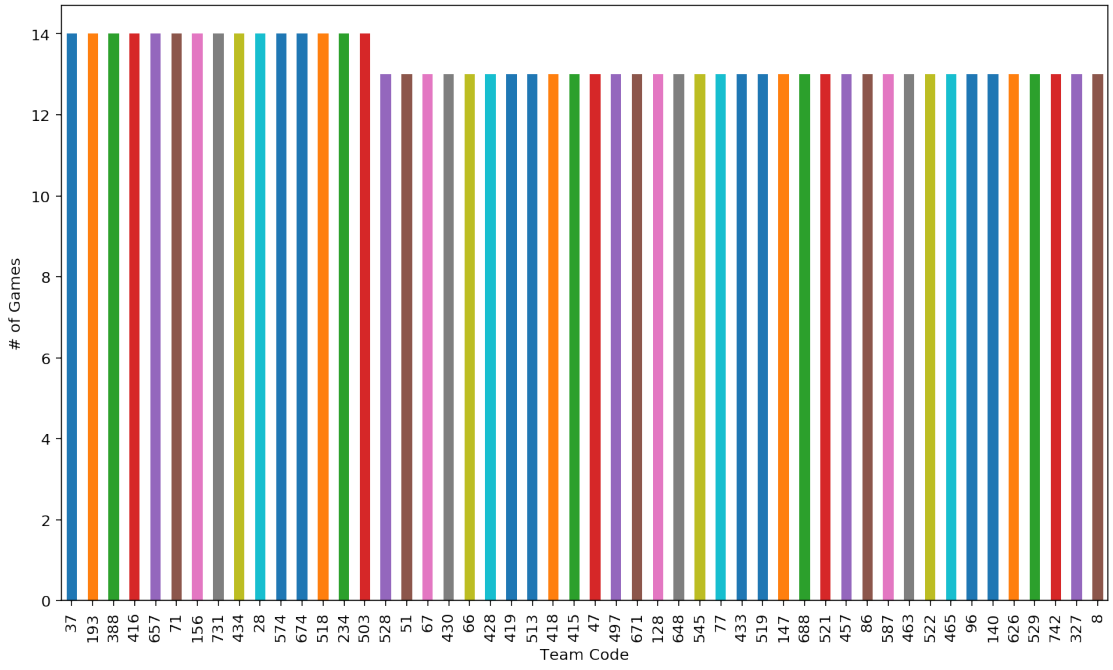
In [6]:
```
# How many games are there?
print('Number of unique games {}'.format(plays['Game Code'].nunique()))
# How many teams are there games for?
print('Number of unique teams {}'.format(plays['Offense Team Code'].nunique()))
# Which teams are the most frequent games for?
games_with_teams = plays.loc[plays['Play Number'] == 1,['Offense Team Code', 'Defense Te
```

```
        team_code_with_game_count = pd.value_counts(games_with_teams.values.flatten())
        team_code_with_game_count.head(50).plot(kind='bar')
        plt.xlabel('Team Code')                    # label = name of label
        plt.ylabel('# of Games')
```

```
Number of unique games 848
Number of unique teams 207
```

Out[6]: Text(0,0.5,'# of Games')

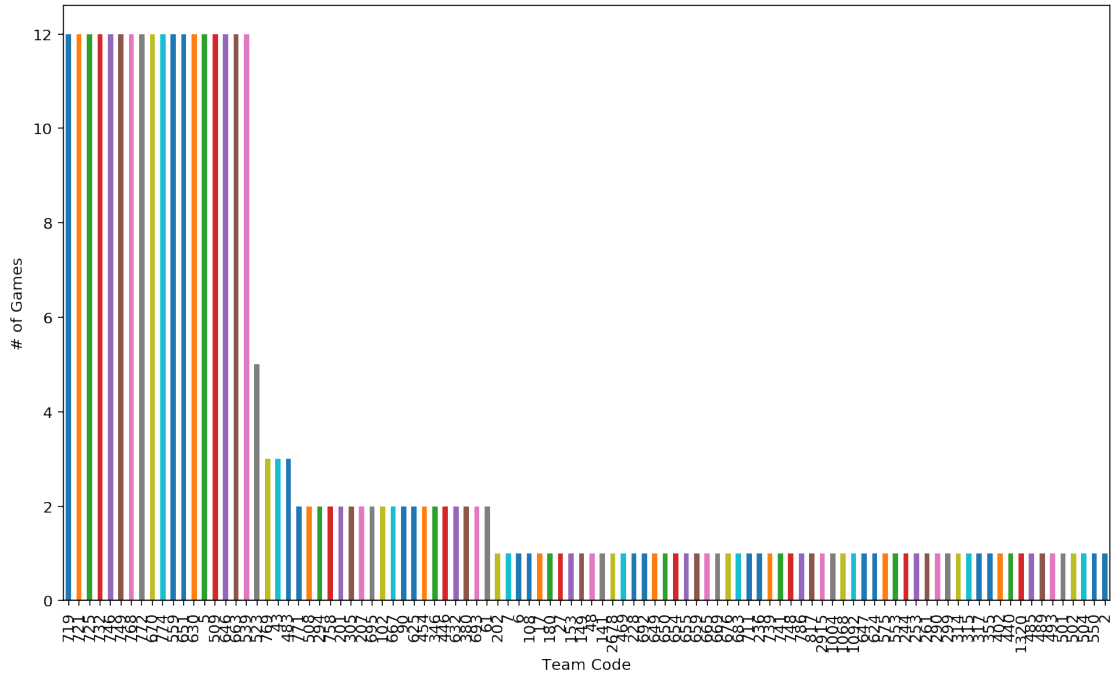
Out[6]:



In [7]:
```
# Is there a dropoff in games per team?
team_code_with_game_count.tail(100).plot(kind='bar')
plt.xlabel('Team Code')
plt.ylabel('# of Games')
```

Out[7]: Text(0,0.5,'# of Games')

Out[7]:




```
In [8]: # How many teams have less than 12 games recorded?
        dict_team_code_count = team_code_with_game_count.to_dict()
        print(sum(v < 12 for k,v in dict_team_code_count.items()))
        # How many teams have 12 or more games recorded?
        print(sum(v >= 12 for k,v in dict_team_code_count.items()))
        # Notice the two add up to the total teams!
```

```
82
125
```

```
In [9]: # Keep the good data to include in out network model
        good_data_dict = {
            k : v
            for k,v in dict_team_code_count.items()
            if v >= 12
        }
        print('# of good teams remaining {}'.format(len(good_data_dict)))
        # How many games did this eliminate?
        bad_data_list = [
            k
            for k,v in dict_team_code_count.items()
            if v < 12
        ]
        games_with_teams = games_with_teams[games_with_teams['Offense Team Code'].isin(bad_data_
```

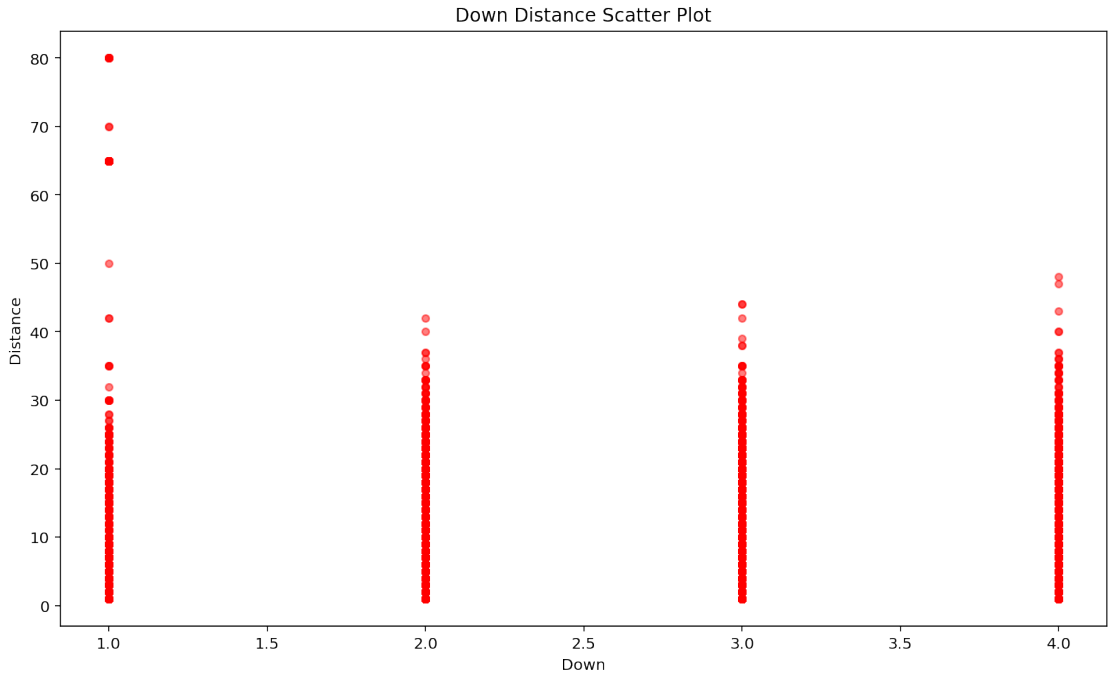
```
        games_with_teams = games_with_teams[games_with_teams['Defense Team Code'].isin(bad_data_
        print('# of games elimatinated by bad teams {}'.format(plays['Game Code'].nunique() - le
```

```
# of good teams remaining 125
# of games elimatinated by bad teams 111
```

In [4]:
```
        # Scatter Plot
        plays.plot(kind='scatter', x='Down', y='Distance',alpha = 0.5,color = 'red')
        plt.xlabel('Down')              # label = name of label
        plt.ylabel('Distance')
        plt.title('Down Distance Scatter Plot')            # title = title of plot
```

Out[4]: Text(0.5,1,'Down Distance Scatter Plot')

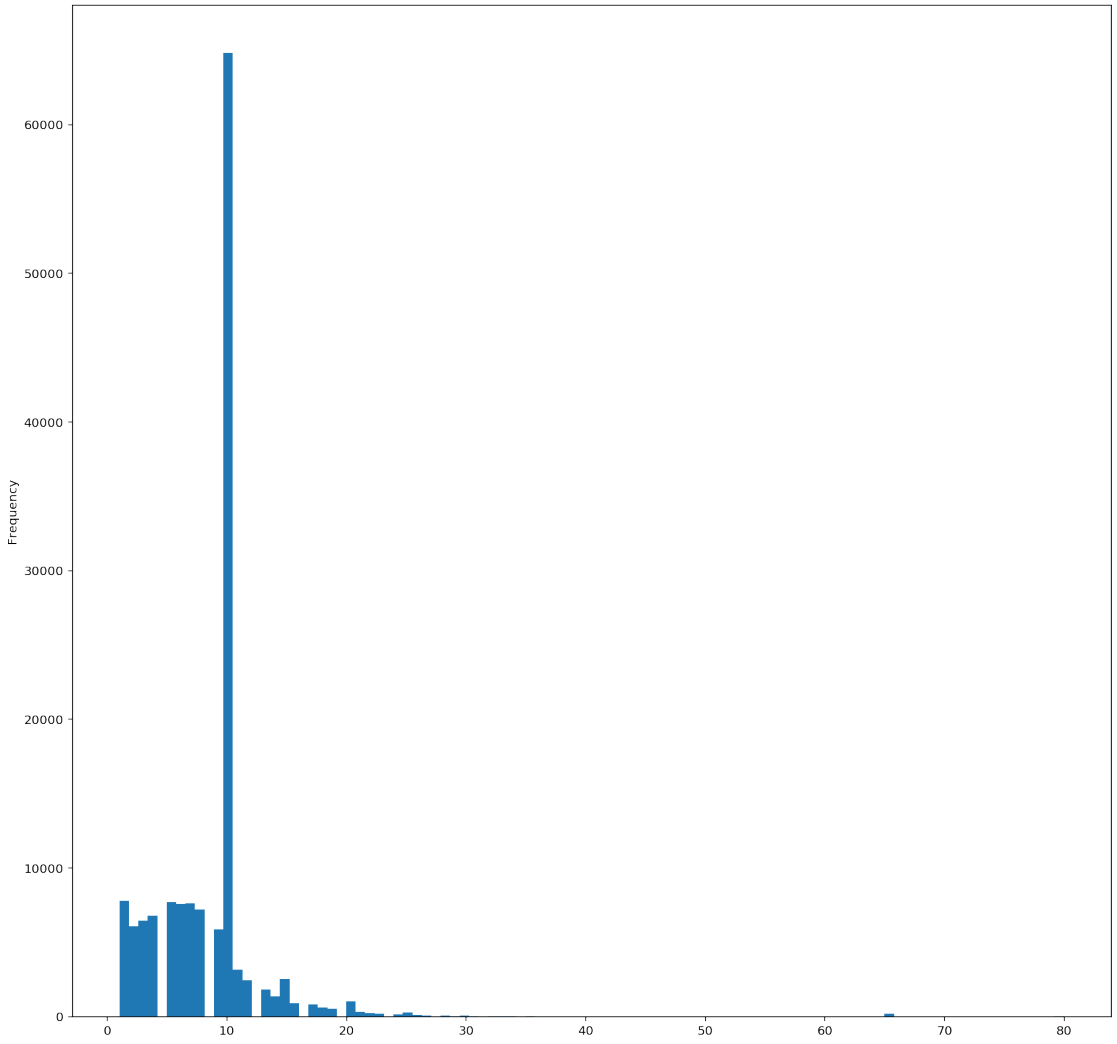
Out[4]:



In [5]:
```
        # Histogram
        # bins = number of bar in figure
        plays.Distance.plot(kind = 'hist',bins = 100,figsize = (15,15))
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc0a5c556a0>

Out[5]:



Outline

train a model to predict outcome First down, second down, third down, punt, field goal, touchdown, safety run a prediction that feeds output into input end quarters and halftime see the outcome of games