NFL Go For It!

by Mike Ghirardo and Thomas McCann

In football there are many decisions a team needs to make in order win the game. In this project we focus on the decision that needs to be made on the 4th down of any given play. There are three decisions to be made on the fourth down.

1. Punt the ball
2. Kick a field goal
3. Go for a first down
   In this project we try to determine which decision should be made under certain conditions. The following are the conditions which we take into account in determining the decision.

4. Offensive and defensive rank of the offensive team.
5. Offensive and defensive rank of the defensive team.
6. The number of yards to convert for a first down.
7. The field position started from.
   With this information from the data we were able to estimate the expected points scored for each of the of three decisions.
   Finally, with this information a decision can be made.

It is fourth and one from the 40 yard line. Should I try a deep 57 yard-field goal? Should I punt or maybe even go for it? Today NFL Coaches are dealt with this type of decision on a game-by-game basis. The conventional wisdom throughout the NFL is to punt on most fourth down situations not in field goal range. However, a coach should wonder if this decision is optimal given the situation. Does my defense really benefit much from just an average of an extra 38 yards (net punt league average) to cover the field? Is that really worth giving up possession of the ball? These are questions that all NFL coaches should think about when making a critical fourth down decision in a game.

This paper investigates a mathematically rigorous way to determine whether a coach should go for it, kick a field goal or punt. The optimal decisions obviously change depending on your yard line, and how many yards you must go to convert the first down. Just these factors however, are clearly not enough to give an accurate representation of whether a team should go for it or not. One also has to consider the offensive and defensive prowess of both the team making the decision and the opposing team. These are the variables considered in our analysis.

The dataset used in our project was rich in nature. It had the play by play information for every game from the 2002 to 2012 seasons. To evaluate our data, we decided to exclude all playoff, pro-bowl, and preseason games. Our project is strictly analyzed from regular season data in the NFL from the 2002-2012 seasons.

The optimal decision is determined from an expected value calculation, given a certain decision was chosen. The average NFL coach or fan might not see the relevance. The purpose of the optimal decision is to win the game, not just increase the expected point margin. But, the graph below might give you pause. This plot shows the very strong, positive linear relationship between expected point margin and winning percentage. These figures are calculated per team, per season over the seasons 2002 – 2012 in the NFL. Expected point margin is an important variable to examine when making a fourth down decision in the early to mid-stages of the game.

Expected point margin is an important factor for determining a decision, but there are situations when this obviously should not be the case. The first oddity and solvable problem are decisions in the fourth quarter. Obviously, it would not be wise to go for a fourth down only because the expected point margin is greater for that particular decision. In the fourth quarter, coaches and management must be playing the score, as well as the clock. Because our model does not account for the clock, all fourth quarter observations were thrown out as well as all observations with less than 4 minutes to go in the first half. If you are interested about decisions in the fourth quarter, you can read about a win percentage model. This model is usually just relevant for fourth quarter situations, because not enough teams go for fourth downs earlier in the game. Our model is purely constructed to give optimal decisions in the first three quarters (excluding the last four minutes before half), where your team most likely has a greater chance of winning if the expected point margin increases.

The expected values are calculated in a complex way, because the observations are strange for usual football analysis. If a team decides to punt, then that team would obtain possession of the ball after the opposing team drives the ball down the field and either scores or does not score. If a team goes for it and converts, they give the ball to the other team after their possession. Obviously, one cannot just calculate what happens in the possession after a decision has been made because they don't measure the same thing. Depending on the choice, at the end of the observation either the team making the decision or the opposing team would have possession of the ball. Therefore, the observations in our study measure what happens regardless of decision until the team making the decision gets the ball again on offense.

So, if the decision is to punt for example, then the team making that decision would punt the ball and evaluate the expected number of points the other team would score until they get the ball back. The expected number of points the other team would score given you punted from a certain yard line would be the negative expected point margin. So, for example, if the team making the decision decided to punt from their 30-yard line, and given that decision the other team was expected to score 1.7 points after that punt, then the point margin differential would be -1.7.

The expected point margin given that you go for it on fourth down is calculated in a similar fashion as the punt, except it is a little more complex because a greater number of things can happen. If you convert, then you must calculate the expected number of points your team will score after that decision. Then, after this drive you must calculate the expected number of points the other team will score against your defense, in order to create an observation in which you get the ball when the observation is over. If you convert the expected point marginal difference will be estimated by subtracting these two numbers. If you do not convert, the calculation is the same calculation as the punt. You simply calculate the expected number of points the opposing team will score given they take over where you attempted the conversion. Then to calculate the full expected value of going for it, you must combine the probability of conversion and non-conversion, with the point marginal difference of conversion and non-conversion respectively.

If you decide to kick a field goal, the calculation is similar to the calculation of going for it because you can either convert or not convert the field goal. If you converted the field goal you scored three points, you must subtract this number from the expected number of points the other team will score once they get the ball. If you do not convert the field goal, you must take the negative expected value of the other team's drive given their new yard line.

One can see that now our observations are all measuring the same thing: the expected point marginal difference given a decision. This is pivotal because it allows us to compare the three decisions directly. It is hard to think about the possibilities of the outcomes of these decisions in an NFL game because they are complex and multi-dimensional, but our model creates a way to make comparisons that are valid and direct.

Because of the nature of our observations a coach must be wary not only for fourth quarter situations, but also decisions towards the end of the first half. If there is not much time left our observations are irrelevant because some of the calculations are based on the other team getting the ball after your possession. This may not happen if there is very little

time left in the half. Once again, our model is not without its flaws. If clock management is an important factor in the decision at hand, the fourth down decision should be evaluated by the coach and/or upper management.

In [1]:
```
import os
import csv
import math
import numpy as np
import pandas as pd
import statsmodels.formula.api as sm
import matplotlib.pyplot as plt
from sklearn import neighbors
import seaborn as sns
import pylab as pl
from mpl_toolkits.mplot3d.axes3d import Axes3D
from matplotlib.backends.backend_pdf import PdfPages
from matplotlib.lines import Line2D
import Image, ImageDraw, ImageFont, ImageOps
import IPython.core.display as icd
import warnings
import matplotlib.colors as mcolors
%load_ext rmagic
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Importing NFL play-by-play data from years 2002 to 2012.

In [3]:
```
os.system('curl -L -o 2002_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2002_nfl_pbp_data.csv"')
os.system('curl -L -o 2003_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2003_nfl_pbp_data.csv"')
os.system('curl -L -o 2004_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2004_nfl_pbp_data.csv"')
os.system('curl -L -o 2005_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2005_nfl_pbp_data.csv"')
os.system('curl -L -o 2006_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2006_nfl_pbp_data.csv"')
os.system('curl -L -o 2007_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2007_nfl_pbp_data.csv"')
os.system('curl -L -o 2008_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2008_nfl_pbp_data.csv"')
os.system('curl -L -o 2009_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2009_nfl_pbp_data.csv"')
os.system('curl -L -o 2010_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2010_nfl_pbp_data.csv"')
os.system('curl -L -o 2011_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2011_nfl_pbp_data.csv"')
```

```
os.system('curl -L -o 2012_nfl_pbp_data.csv "https://raw.githubusercontent.co
m/tomdawg86/Stat_222_NFL_Project/master/2012_nfl_pbp_data.csv"')
```

Out[3]: 0

In [2]:
```
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Joining data sets together to get one long dataset of play-by-play data accross all 11 years.

In [3]:
```
pbp_data = pd.read_csv('2002_nfl_pbp_data.csv')
yard = range(0,100)
teams = list(set(pbp_data.off))[1:]
seasons = range(2002,2013)
for i in seasons[1:]:
    pbp_data = pbp_data.append(pd.read_csv('%d_nfl_pbp_data.csv' % i), ignore
_index = True)
```

Taking out post-season games to get a more accurate ranking of teams.

In [4]:
```
pbp_dataTF = pbp_data['gameid'].map(lambda x: str(x).endswith(('01', '02'), 4
, 6)) == False
pbp_dataTF[pbp_data['gameid'].map(lambda x: str(x).endswith(('20050102', '200
60101', '20100103', '20110102', '20120101'), 0, 8)) == True] = True
pbp_dataT = pbp_dataTF[pbp_dataTF == True]
pbp_data = pbp_data.loc[pbp_dataT.index]
pbp_data = pbp_data.reset_index(range(len(pbp_data)))
pbp_data = pbp_data.drop(['index'], 1)
```

The following retrieves the first and last plays of each game. This will be useful in helping us determine team ranks by how many points scored per game and how many points let go per game.

In [5]:
```
shifted_data1 = (pbp_data.shift(1)).shift(-1)
shifted_data2 = pbp_data.shift(1)
shifted_data2.rename(columns=lambda x: 'last' + x, inplace=True)
shifted_data = shifted_data1.join(shifted_data2, how = 'outer')
offTF = shifted_data['off'] == shifted_data['lastoff']
defTF = shifted_data['def'] == shifted_data['lastdef']
gameIDTF = shifted_data['gameid'] == shifted_data['lastgameid']
first_last_play = shifted_data[offTF == False]
first_last_play = shifted_data[gameIDTF == False]

first_last_play['win'] = first_last_play['lastoff'][(first_last_play['lastoff
score'] > first_last_play['lastdefscore']) == True]
first_last_play['win'].fillna(first_last_play['lastdef'], inplace = True)
first_last_play['loss'] = first_last_play['lastdef'][(first_last_play['lastof
fscore'] > first_last_play['lastdefscore']) == True]
```

```
first_last_play['loss'].fillna(first_last_play['lastoff'], inplace = True)
wins = pd.DataFrame(first_last_play.groupby(['lastseason', 'win'])['win'].cou
nt())
losses = pd.DataFrame(first_last_play.groupby(['lastseason', 'loss'])['loss']
.count())
winPercentage = pd.DataFrame(wins/(wins + losses))
```

The following sums points gained and points let go per game per team per season. The means by which the teams are ranked offensively and defensively is taking the total number of points scored and total number of points let go and adding them.

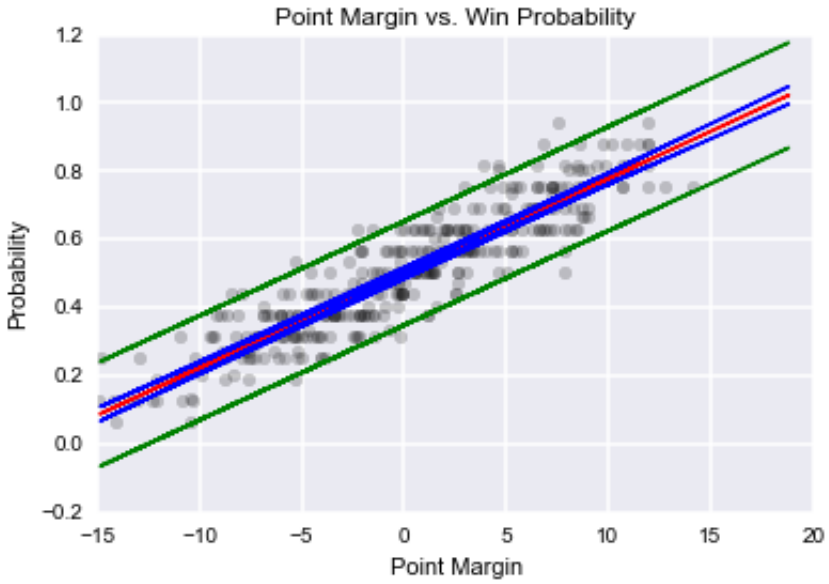
In [6]:
```
rank_off_off_score = pd.DataFrame(first_last_play.groupby(['lastseason', 'las
toff'])['lastoffscore'].sum())
rank_off_def_score = pd.DataFrame(first_last_play.groupby(['lastseason', 'las
tdef'])['lastdefscore'].sum())
rank_def_def_score = pd.DataFrame(first_last_play.groupby(['lastseason', 'las
toff'])['lastdefscore'].sum())
rank_def_off_score = pd.DataFrame(first_last_play.groupby(['lastseason', 'las
tdef'])['lastoffscore'].sum())
team_score_off_rank = rank_off_off_score.join(rank_off_def_score)
team_score_off_rank['total'] = team_score_off_rank['lastoffscore'] + team_sco
re_off_rank['lastdefscore']
team_score_def_rank = rank_def_def_score.join(rank_def_off_score)
team_score_def_rank['total'] = team_score_def_rank['lastoffscore'] + team_sco
re_def_rank['lastdefscore']
```

In [7]:
```
teamMargin = pd.DataFrame((team_score_off_rank['total'] - team_score_def_rank
['total'])/16)
teamMargVsWP = teamMargin.join(winPercentage)
teamMargVsWP.columns = ['teamMargin', 'winningPercentage']
```

In [8]:
```
def olsCIandPI(x, y, title, xaxis, yaxis, saveas):
    main = pd.DataFrame({'x' : x, 'y' : y})
    equation = sm.ols(formula="y ~ x", data=main)
    mainRes = equation.fit()
    MSERes = mainRes.mse_resid
    n = mainRes.nobs
    x = main['x']
    y = main['y']
    alpha = mainRes.params[0]
    beta = mainRes.params[1]
    y_est = alpha + beta*x
    y_u = alpha + beta*x  + 1.96*sqrt(((1.0/(n-2.0))*(n*MSERes))*(1.0/(n) + (
((x - mean(x))**2)/(sum((x - mean(x))**2)))))
    y_l = alpha + beta*x  - 1.96*sqrt(((1.0/(n-2.0))*(n*MSERes))*(1.0/(n) + (
((x - mean(x))**2)/(sum((x - mean(x))**2)))))
    y_u_pred = alpha + beta*x  + 1.96*sqrt(((1.0/(n-2.0))*(n*MSERes))*(1 + 1.
```

```
0/(n) + (((x - mean(x))**2)/(sum((x - mean(x))**2)))))
    y_l_pred = alpha + beta*x  -1.96*sqrt(((1.0/(n-2.0))*(n*MSERes))*(1 + 1.0
/(n) + (((x - mean(x))**2)/(sum((x - mean(x))**2)))))
    plt.plot(x, y, 'bo', alpha = 0.2, color = "black")
    plt.plot(x, y_est, 'r-')
    plt.plot(x, y_u, 'b-')
    plt.plot(x, y_l, 'b-')
    plt.plot(x, y_u_pred, 'g-')
    plt.plot(x, y_l_pred, 'g-')
    plt.xlabel(xaxis)
    plt.ylabel(yaxis)
    plt.title(title)
    savefig(saveas)
    plt.show()
```

In [9]:
```
olsCIandPI(x = teamMargVsWP['teamMargin'], y = teamMargVsWP['winningPercentag
e'], xaxis = 'Point Margin', yaxis = 'Probability', title = 'Point Margin vs.
 Win Probability', saveas = "MarginWinProb.pdf")
```



Creating a two matrices of the total points scored and total points let go with the season as the column and the team as the row, and then ranking them to get the offensive and defensive ranks.

In [10]:
```
off_rank_points = [[0]*len(seasons) for x in range(len(teams))]
def_rank_points = [[0]*len(seasons) for x in range(len(teams))]

for i in range(len(teams)):
    for j in range(len(seasons)):
        off_rank_points[i][j] = team_score_off_rank['total'].loc[seasons[j],
teams[i]]
        def_rank_points[i][j] = team_score_def_rank['total'].loc[seasons[j],
teams[i]]
```

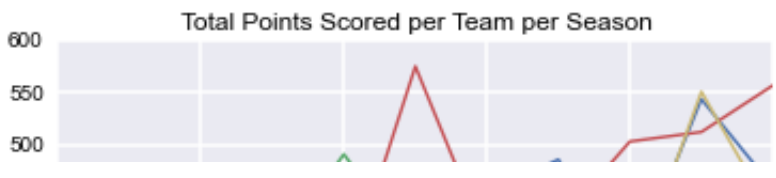
```
off_rank_points, def_rank_points = pd.DataFrame(off_rank_points, index = team
s), pd.DataFrame(def_rank_points, index = teams)
off_rank_points.rename(columns=lambda x: x + 2002, inplace=True)
def_rank_points.rename(columns=lambda x: x + 2002, inplace=True)
off_rank = off_rank_points.rank(axis = 0, method = 'max', ascending = False)
def_rank = def_rank_points.rank(axis = 0, method = 'max', ascending = True)

offrank_pps = pd.DataFrame(index = seasons)
for i in range(len(teams)):
    offrank_pps[i + 1] = pd.DataFrame(amax(off_rank_points[off_rank == amin(o
ff_rank + i, axis = 0)], axis = 0))

defrank_pps = pd.DataFrame(index = seasons)
for i in range(len(teams)):
    defrank_pps[i + 1] = pd.DataFrame(amax(def_rank_points[def_rank == amin(d
ef_rank + i, axis = 0)], axis = 0))

off_rank_points = off_rank_points.T
def_rank_points = def_rank_points.T
```

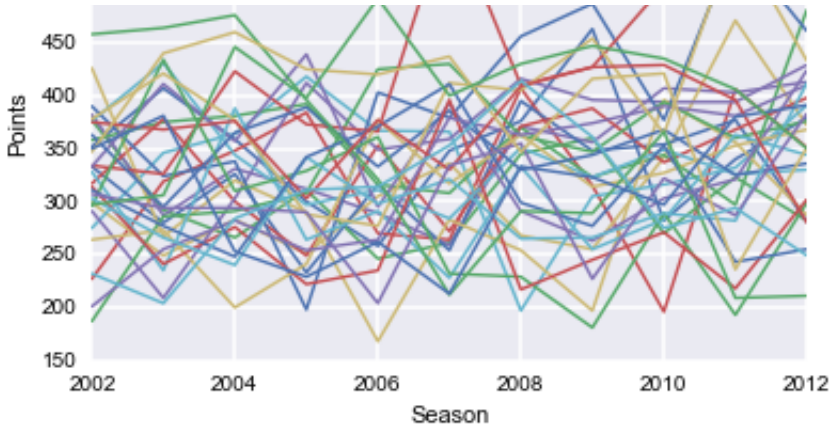
After researching and deciding on our implementation method , we needed to try and figure out a way to rank the teams 1 to 32 on offense and defense (1 = best to 32 = worst). In order to rank teams first we calculated the average number of points scored and given up per game per season. The best offensive team for a particular season would be the team that scored the most points per game. The best defensive team for a particular season would be the team that gave up the fewest number of points per game. Then, we plotted the number of points scored and given up for the 32 teams per season. One could see that because our time frame is rather long (11 years), the teams average number of points scored and given up fluctuated greatly from season to season. For example, the San Francisco 49ers went from one of the worst defenses in the league in 2007 to one of the best in 2012.

Because fluctuations were so dramatic for the teams, we chose to rank the teams 1 to 32 per season. So, the number one ranked offensive team for example could change from the Patriots in 2002 to the Packers in 2003. After, we made this correction our plots of ranking and years look much better for both offense and defense.

The following is a plot of total points scored per season per team. Each color represents a different team.

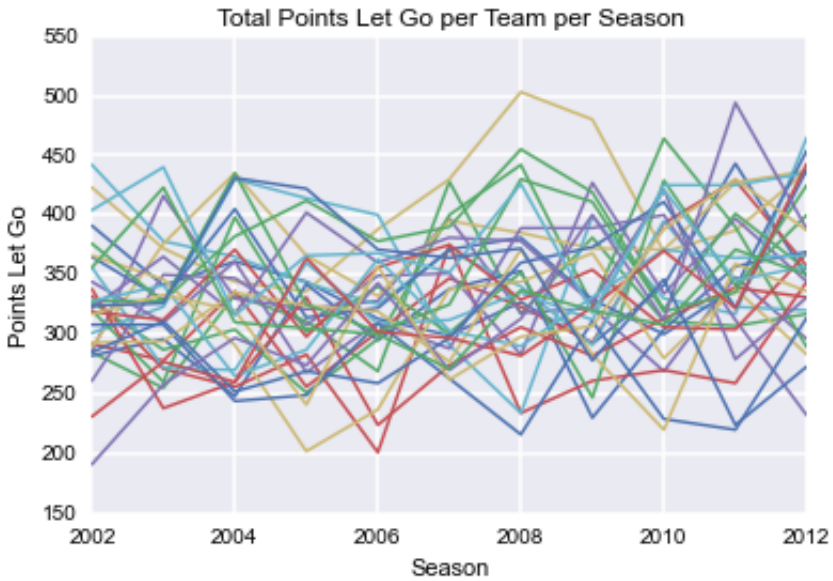
In [11]:
```
fig, ax = plt.subplots()
for i in teams:
    ax.plot(offrank_pps.index, off_rank_points[i])

xlabel("Season")
ylabel("Points")
title("Total Points Scored per Team per Season")
savefig('DefTeamperSeason.pdf')
plt.show()
```





The following is a plot of total points let go per season per team. Each color represents a different team.

In [12]:
```
fig, ax = plt.subplots()
for i in teams:
    ax.plot(defrank_pps.index, def_rank_points[i])

xlabel("Season")
ylabel("Points Let Go")
title("Total Points Let Go per Team per Season")
savefig('DefTeamperSeason.pdf')
plt.show()
```

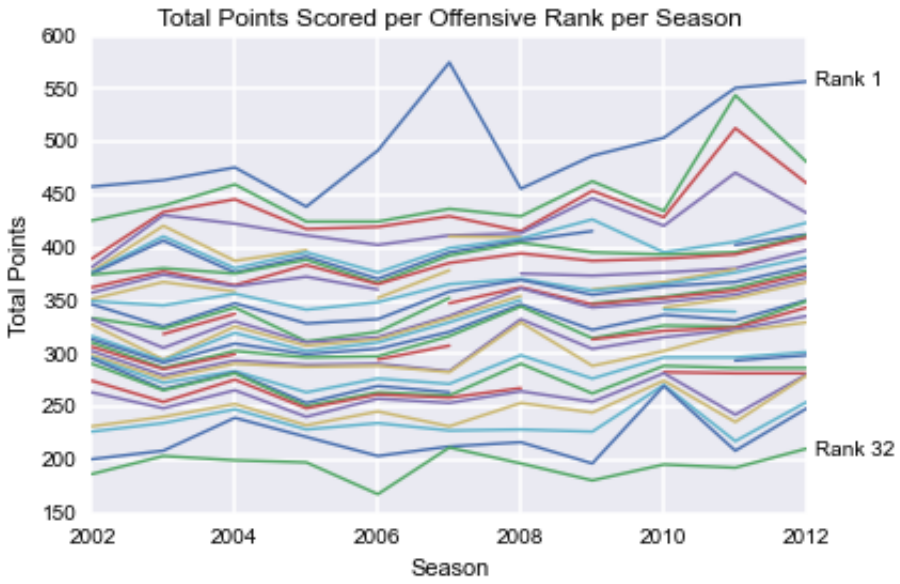


The graphs below show the top four and bottom two teams are many times outliers in particular seasons. Many times the difference between these outlying teams expected points is greater than the difference for the teams in the middle. Because of this we created dummy variables, one for those teams ranked 31 and 32, and one for those teams ranked 5 – 30 (for both offense and defense). The dummy variable between 5 and 30 are linearly related so we multiply this linear factor by the 5 – 30 dummy. However, teams 31 and 32 are close enough that we simply create a 0-1 dummy variable. Because of these dummies, our models are estimated in comparison to offenses and defenses ranked 1-4.

The following is a plot of total points scored per season per offensive rank. Each color represents a different rank.

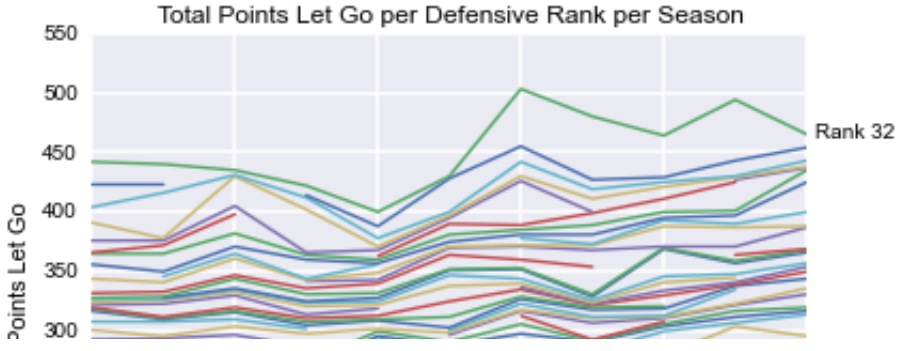
In [13]:
```
fig, ax = plt.subplots()
```

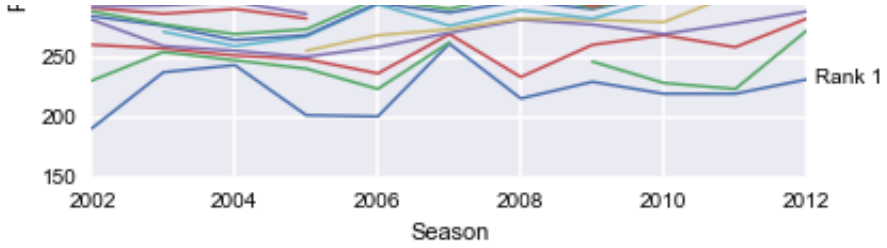
```
for i in range(1, 33):
    ax.plot(offrank_pps.index, offrank_pps[i])

xlabel("Season")
ylabel("Total Points")
title("Total Points Scored per Offensive Rank per Season")
figtext(.91,.82,"Rank 1")
figtext(.91,.22,"Rank 32")
savefig('OffRankperSeason.pdf')
plt.show()
```



The following is a plot of total points scored per season per defensive rank. Each color represents a different rank.

In [14]:
```
fig, ax = plt.subplots()
for i in range(1, 33):
    ax.plot(defrank_pps.index, defrank_pps[i])

xlabel("Season")
ylabel("Points Let Go")
title("Total Points Let Go per Defensive Rank per Season")
figtext(.91,.73,"Rank 32")
figtext(.91,.28,"Rank 1")
savefig('DefRankperSeason.pdf')
plt.show()
```





Here we bring in the team rankings into the main data frame.

In [15]:
```
off_rank = pd.DataFrame(off_rank.stack())
def_rank = pd.DataFrame(def_rank.stack())
off_rank.columns = ['offrank']
def_rank.columns = ['defrank']
pbp_data = pbp_data.join(off_rank, on = ['off', 'season'], how = 'left')
pbp_data = pbp_data.join(def_rank, on = ['def', 'season'], how = 'left')
```

Here we create dummy variables and factor variables concerning the ranking of teams. This will help us run a logistic regression using team ranking as covariates.

In [16]:
```
pbp_data['offrankbucket'] = 'NA'
pbp_data['defrankbucket'] = 'NA'
pbp_data['offrank1t4'] = 0
pbp_data['offrank5t30'] = 0
pbp_data['offrank31t32'] = 0
pbp_data['defrank1t4'] = 0
pbp_data['defrank5t30'] = 0
pbp_data['defrank31t32'] = 0
pbp_data.offrankbucket[pbp_data.offrank >= 1], pbp_data.defrankbucket[pbp_dat
a.defrank >= 1] = '(1-4)', '(1-4)'
pbp_data.offrankbucket[pbp_data.offrank >= 5], pbp_data.defrankbucket[pbp_dat
a.defrank >= 5] = '(5-30)', '(5-30)'
pbp_data.offrankbucket[pbp_data.offrank >= 31], pbp_data.defrankbucket[pbp_da
ta.defrank >= 31] = '(31-32)', '(31-32)'
pbp_data.offrank1t4[pbp_data.offrankbucket == '(1-4)'], pbp_data.defrank1t4[p
bp_data.defrankbucket == '(1-4)'] = 1, 1
pbp_data.offrank5t30[pbp_data.offrankbucket == '(5-30)'], pbp_data.defrank5t3
0[pbp_data.defrankbucket == '(5-30)'] = 1, 1
pbp_data.offrank31t32[pbp_data.offrankbucket == '(31-32)'], pbp_data.defrank3
1t32[pbp_data.defrankbucket == '(31-32)'] = 1, 1
pbp_data['offrankMid'] = pbp_data['offrank5t30']*pbp_data['offrank']
pbp_data['defrankMid'] = pbp_data['defrank5t30']*pbp_data['defrank']
offrankbucket = pd.Categorical.from_array(pbp_data['offrankbucket'])
defrankbucket = pd.Categorical.from_array(pbp_data['defrankbucket'])
pbp_data['offrankbucket'] = offrankbucket.labels
pbp_data['defrankbucket'] = defrankbucket.labels
```

In [17]:
```
pbp_data_new = pbp_data[pbp_data.description.str.contains('kicks') == False]
```

```
pbp_data_new.index = arange(len(pbp_data_new))
shifted_data1 = (pbp_data_new.shift(1)).shift(-1)
shifted_data2 = pbp_data_new.shift(1)
shifted_data2.rename(columns=lambda x: 'last' + x, inplace=True)
shifted_data = shifted_data1.join(shifted_data2, how = 'outer')
```

In [18]:
```
offTF = shifted_data['off'] == shifted_data['lastoff']
defTF = shifted_data['def'] == shifted_data['lastdef']
qtrTF = shifted_data['qtr'].isin([3]) & shifted_data['lastqtr'].isin([2]) ==
True
shifted_data['condition'] = 'NA'
shifted_data.condition[qtrTF == True] = 1
shifted_data.condition[offTF == False] = 2
shifted_data.condition[defTF == False] = 3
shifted_data.condition[0:30]
down1 = shifted_data[shifted_data['condition'].isin([1, 2, 3])]
```

The following gives drive by drive information. This information is useful in helping us know the expected number of points the offensive team will score given they started the first play of the drive on a specific yard line.

In [19]:
```
firstPlay = down1.index.values
firstPlay = np.array(firstPlay)
lastPlay = firstPlay - 1
lastPlay1 = lastPlay[1:len(lastPlay)]
lastPlay2 = np.append(lastPlay1, (len(pbp_data)-1))
lastPlays = pbp_data_new.ix[lastPlay2]

lastPlays.rename(columns=lambda x: 'last_' + x, inplace=True)
down1['mergeVals'] = np.arange(len(firstPlay))
lastPlays['mergeVals'] = np.arange(len(firstPlay))
driveByDrive = down1.merge(lastPlays, on = 'mergeVals')
driveByDrive['offrankMid'] = driveByDrive['offrank5t30']*driveByDrive['offran
k']
driveByDrive['defrankMid'] = driveByDrive['defrank5t30']*driveByDrive['defran
k']
```

The following code quantifies the consequences of certain events occurring. We create a vector of the number of points scored at the end of a teams drive. We assume that touchdown plays automatically get seven points, which means we assume the team gets the extra point given they score a touchdown. Then we run a multinomial logistic regression to determine the likelihood of these events based on certain factors, such as team rank and field position. With both pieces of information we determine the expected number points of a team given they convert a first down, their ranking and their field position.

In [20]:
```
driveByDrive['pointsScored'] = 0
madeFG = driveByDrive.last_description.str.contains('GOOD') == True
driveByDrive.pointsScored[madeFG == True] = 3
madeTD = driveByDrive.last_description.str.contains('extra point' or 'Extra P
```

```
oint' or 'Extra point') == True
madeTD2 = driveByDrive.last_description.str.contains('TWO-POINT CONVERSION')
== True
driveByDrive.pointsScored[madeTD == True] = 7
driveByDrive.pointsScored[madeTD2 == True] = 7
safety = driveByDrive.last_description.str.contains('SAFETY') == True
driveByDrive.pointsScored[safety == True] = -2
defTD = driveByDrive.last_description.str.contains('TOUCHDOWN') == True
driveByDrive.pointsScored[defTD == True] = 7
driveByDrive.pointsScored.mean()
driveByDrive['puntReturnForTD'] = 0
puntReturnTD = (driveByDrive.last_description.str.contains('punt') & driveByD
rive.last_description.str.contains('TOUCHDOWN')) == True
driveByDrive.puntReturnForTD[puntReturnTD == True] = 1
driveByDrive.pointsScored[puntReturnTD == True] = -7

driveByDrive['interceptionForTD'] = 0
interceptAndTD = driveByDrive.last_description.str.contains('INTERCEPTION') &
 driveByDrive.last_description.str.contains('TOUCHDOWN') == True
driveByDrive.interceptionForTD[interceptAndTD == True] = 1
driveByDrive.pointsScored[puntReturnTD == True] = -7

driveByDrive['fumbleForTD'] = 0
fumbleAndTD = driveByDrive.last_description.str.contains('FUMBLE') & driveByD
rive.last_description.str.contains('TOUCHDOWN') == True
driveByDrive.fumbleForTD[fumbleAndTD == True] = 1
driveByDrive.pointsScored[fumbleAndTD == True] = -7

driveByDrive['blockedPuntForTD'] = 0
blockedPuntAndTD = driveByDrive.last_description.str.contains('BLOCK') & driv
eByDrive.last_description.str.contains('TOUCHDOWN') == True
driveByDrive.blockedPuntForTD[blockedPuntAndTD == True] = 1
driveByDrive.pointsScored[blockedPuntAndTD == True] = -7

driveByDrive['intercept'] = [1]*len(driveByDrive)
driveByDrive['score'] = [0]*len(driveByDrive)
driveByDrive = pd.DataFrame(driveByDrive)

driveByDrive.score[driveByDrive['pointsScored'] == -7] = 'DefTD'
driveByDrive.score[driveByDrive['pointsScored'] == -2] = 'DefSafety'
driveByDrive.score[driveByDrive['pointsScored'] == 0] = 'NoPoints'
driveByDrive.score[driveByDrive['pointsScored'] == 3] = 'FG'
driveByDrive.score[driveByDrive['pointsScored'] == 7] = 'TD'

driveByDriveNew = (driveByDrive.shift(1)).shift(-1)
driveByDriveShift = driveByDrive[['puntReturnForTD','interceptionForTD','fumb
leForTD','blockedPuntForTD']].shift(1)
```

```
driveByDriveShift.rename(columns=lambda x: 'remove' + x, inplace=True)
driveByDrive2 = driveByDriveNew.join(driveByDriveShift, how = 'outer')
driveByDrive3 = driveByDrive2[driveByDrive2['removepuntReturnForTD'] != 1]
driveByDrive4 = driveByDrive3[driveByDrive3['removeinterceptionForTD'] != 1]
driveByDrive5 = driveByDrive4[driveByDrive4['removefumbleForTD'] != 1]
driveByDrive6 = driveByDrive5[driveByDrive5['removeblockedPuntForTD'] != 1]
driveByDriveN = driveByDrive6[['score', 'intercept', 'ydline', 'offrank31t32'
, 'offrankMid', 'defrank31t32', 'defrankMid']]
driveByDriveN = driveByDriveN.dropna(axis = 0, how = "any")
score_logit = sm.MNLogit(driveByDriveN[['score']], driveByDriveN[['intercept'
, 'ydline', 'offrankMid', 'offrank31t32', 'defrankMid', 'defrank31t32']])

score_logitResult = score_logit.fit()
score_logitResult.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.872453
         Iterations 12
```

Out[20]:

MNLogit Regression Results

Dep. Variable:	score	**No. Observations:**	65105
Model:	MNLogit	**Df Residuals:**	65081
Method:	MLE	**Df Model:**	20
Date:	Wed, 30 Apr 2014	**Pseudo R-squ.:**	0.05169
Time:	08:34:43	**Log-Likelihood:**	-56801.
converged:	True	**LL-Null:**	-59897.
		LLR p-value:	0.000

score=DefTD	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	17.1021	1.159	14.759	0.000	14.831 19.373
ydline	-0.1892	0.012	-15.208	0.000	-0.214 -0.165
offrankMid	0.0065	0.010	0.620	0.535	-0.014 0.027
offrank31t32	0.0238	0.347	0.069	0.945	-0.656 0.704
defrankMid	0.0103	0.010	1.018	0.309	-0.010 0.030
defrank31t32	0.4581	0.395	1.161	0.246	-0.315 1.231
score=FG	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	23.3837	1.125	20.785	0.000	21.179 25.589
ydline	-0.2381	0.012	-19.834	0.000	-0.262 -0.215
offrankMid	-0.0052	0.009	-0.565	0.572	-0.023 0.013

offrank31t32	-0.7413	0.308	-2.410	0.016	-1.344 -0.139
defrankMid	0.0217	0.009	2.428	0.015	0.004 0.039
defrank31t32	0.3214	0.357	0.901	0.368	-0.378 1.021
score=NoPoints	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	22.7030	1.124	20.200	0.000	20.500 24.906
ydline	-0.2050	0.012	-17.097	0.000	-0.228 -0.181
offrankMid	0.0080	0.009	0.870	0.384	-0.010 0.026
offrank31t32	-0.2494	0.303	-0.823	0.411	-0.844 0.345
defrankMid	0.0104	0.009	1.174	0.241	-0.007 0.028
defrank31t32	0.0488	0.354	0.138	0.890	-0.644 0.742
score=TD	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	23.6977	1.125	21.070	0.000	21.493 25.902
ydline	-0.2343	0.012	-19.524	0.000	-0.258 -0.211
offrankMid	-0.0209	0.009	-2.270	0.023	-0.039 -0.003
offrank31t32	-1.3770	0.307	-4.487	0.000	-1.979 -0.775
defrankMid	0.0302	0.009	3.393	0.001	0.013 0.048
defrank31t32	0.7186	0.355	2.023	0.043	0.023 1.415

The following is code concerning the decision to punt. Here, we find all punt plays and using logistic regression we determine the probability of events happening given that the team making the decision punted the ball at a certain yard line. The other team receiveing the punt can then score an offensive touchdown or field goal, get no points or give up a defensive touchdown or safety.

In [21]:
```
driveByDriveNew2 = (driveByDrive.shift(1)).shift(-1)
driveByDriveShift2 = driveByDrive[['qtr', 'ydline', 'last_description','last_
ydline', 'score', 'offrankMid', 'offrank31t32', 'defrankMid', 'defrank31t32',
 'off', 'def']].shift(1)
driveByDriveShift2.rename(columns=lambda x: 'remove' + x, inplace=True)
driveByDriveNew3 = driveByDriveNew2.join(driveByDriveShift2, how = 'outer')
diffHalf = driveByDriveNew3.removeqtr.isin([2]) & driveByDriveNew3.qtr.isin([
3]) == True
diffGame = driveByDriveNew3.removeqtr.isin([4]) & driveByDriveNew3.qtr.isin([
1]) == True
driveByDriveNew4 = driveByDriveNew3[(diffHalf & diffGame) == False]
prevNoScore = driveByDriveNew4.removescore.str.contains('NoPoints')
driveByDriveNew5 = driveByDriveNew4[prevNoScore == True]
#driveByDriveNew5[['description', 'last_description','removelast_description'
, 'removeoff', 'removeoffrankMid', 'removeoffrank31t32', 'removedef','defrank
Mid', 'defrank31t32', 'removeydline', 'ydline']].head()
```

```
### regression with y = ydline, x = removeydline, removeoff, removedef
ydsDrivenGivenNoPoints = sm.ols(formula="ydline ~ removeydline + removeoffran
kMid + removeoffrank31t32 + removedefrankMid + removedefrank31t32", data=driv
eByDriveNew5)
ydsDrivenGivenNoPointsResult = ydsDrivenGivenNoPoints.fit()
ydsDrivenGivenNoPointsResult.summary()
ydsDriven = ydsDrivenGivenNoPointsResult.params

prevPunt = driveByDriveNew4.removelast_description.str.contains('punts')
driveByDrivePunt = driveByDriveNew4[prevPunt == True]
driveByDrivePunt2 = driveByDrivePunt[['score', 'intercept', 'removelast_ydlin
e', 'offrankMid','offrank31t32', 'defrankMid', 'defrank31t32']]
driveByDrivePunt3 = driveByDrivePunt2.dropna(axis = 0, how = "any")
score_logit_punt = sm.MNLogit(driveByDrivePunt3[['score']], driveByDrivePunt3
[['intercept', 'removelast_ydline', 'offrankMid','offrank31t32', 'defrankMid'
, 'defrank31t32']])

score_logit_puntResult = score_logit_punt.fit()
score_logit_puntResult.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.881161
         Iterations 9
```

Out[21]:

MNLogit Regression Results

Dep. Variable:	score	**No. Observations:**	26271
Model:	MNLogit	**Df Residuals:**	26247
Method:	MLE	**Df Model:**	20
Date:	Wed, 30 Apr 2014	**Pseudo R-squ.:**	0.03018
Time:	08:34:46	**Log-Likelihood:**	-23149.
converged:	True	**LL-Null:**	-23869.
		LLR p-value:	2.247e-293

score=DefTD	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	-2.5052	0.567	-4.415	0.000	-3.617 -1.393
removelast_ydline	0.0502	0.009	5.741	0.000	0.033 0.067
offrankMid	0.0073	0.014	0.520	0.603	-0.020 0.035
offrank31t32	-0.1668	0.431	-0.387	0.699	-1.011 0.678
defrankMid	0.0189	0.013	1.431	0.153	-0.007 0.045
defrank31t32	0.2559	0.592	0.432	0.666	-0.905 1.417
score=FG	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**

intercept	-2.0875	0.461	-4.532	0.000	-2.990 -1.185
removelast_ydline	0.0874	0.007	11.719	0.000	0.073 0.102
offrankMid	-0.0018	0.012	-0.154	0.878	-0.024 0.021
offrank31t32	-0.9428	0.348	-2.712	0.007	-1.624 -0.262
defrankMid	0.0284	0.011	2.621	0.009	0.007 0.050
defrank31t32	0.6024	0.488	1.235	0.217	-0.354 1.558
score=NoPoints	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	1.3069	0.451	2.897	0.004	0.423 2.191
removelast_ydline	0.0607	0.007	8.239	0.000	0.046 0.075
offrankMid	0.0132	0.011	1.160	0.246	-0.009 0.036
offrank31t32	-0.3978	0.338	-1.176	0.240	-1.061 0.265
defrankMid	0.0149	0.011	1.395	0.163	-0.006 0.036
defrank31t32	0.2583	0.482	0.536	0.592	-0.687 1.204
score=TD	**coef**	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	-1.1274	0.457	-2.469	0.014	-2.022 -0.233
removelast_ydline	0.0802	0.007	10.814	0.000	0.066 0.095
offrankMid	-0.0164	0.011	-1.428	0.153	-0.039 0.006
offrank31t32	-1.4860	0.346	-4.295	0.000	-2.164 -0.808
defrankMid	0.0381	0.011	3.529	0.000	0.017 0.059
defrank31t32	1.1201	0.485	2.311	0.021	0.170 2.070

The following code is used to pull out fourth down plays from the data. This is important since we'll use plays from these downs to find the expected number of points given field goal attempt, punt attempt, or go for it attempt.

In [22]:
```
fourthDown = pbp_data[pbp_data['down'] == 4]
fourthPlayNumbers = fourthDown.index
fourthPlayNumbers2 = fourthPlayNumbers[1:len(fourthPlayNumbers)]
fourthPlayNumbers2 = np.append(fourthPlayNumbers2, 0)
nonPenaltyFourthDown = fourthPlayNumbers[fourthPlayNumbers2 - fourthPlayNumbe
rs != 1]
fourthDownPlays = pbp_data.ix[nonPenaltyFourthDown, :]
```

Pulling out field goal attempt data and running a logistic regression to determine the likelihood of converting depending on the yardline the field goal is attempted from.

In [23]:
```
fourth = fourthDownPlays
cond = fourthDownPlays['description'].str.contains('field goal' or 'Field Goa
l' or 'field Goal' or 'Field goal' or 'FIELD GOAL')
```

```
cond2 = fourthDownPlays['description'].str.contains('punt' or 'PUNT' or 'Punt
')
field_goal_data = fourthDownPlays[cond == True]
field_goal_data['converted'] = field_goal_data['description'].str.contains('G
OOD')
field_goal_data['distance'] = field_goal_data['ydline'] + 18
field_goal = field_goal_data[['ydline', 'distance', 'description', 'converted
']]
field_goal['intercept'] = [1]*len(field_goal)
field_goal_logit = sm.Logit(field_goal[['converted']], field_goal[['intercept
','ydline']])
field_goal_result = field_goal_logit.fit()
field_goal_result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.410096
         Iterations 7
```

Out[23]:

Logit Regression Results

Dep. Variable:	converted	**No. Observations:**	9650
Model:	Logit	**Df Residuals:**	9648
Method:	MLE	**Df Model:**	1
Date:	Wed, 30 Apr 2014	**Pseudo R-squ.:**	0.1206
Time:	08:34:46	**Log-Likelihood:**	-3957.4
converged:	True	**LL-Null:**	-4500.1
		LLR p-value:	5.246e-238

	coef	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	3.6030	0.083	43.532	0.000	3.441 3.765
ydline	-0.0981	0.003	-29.699	0.000	-0.105 -0.092

In [24]:
```
fg_vec = np.array(field_goal_result.params)
fg_vec_upper = np.array(field_goal_result.conf_int()[1])
fg_vec_lower = np.array(field_goal_result.conf_int()[0])
```

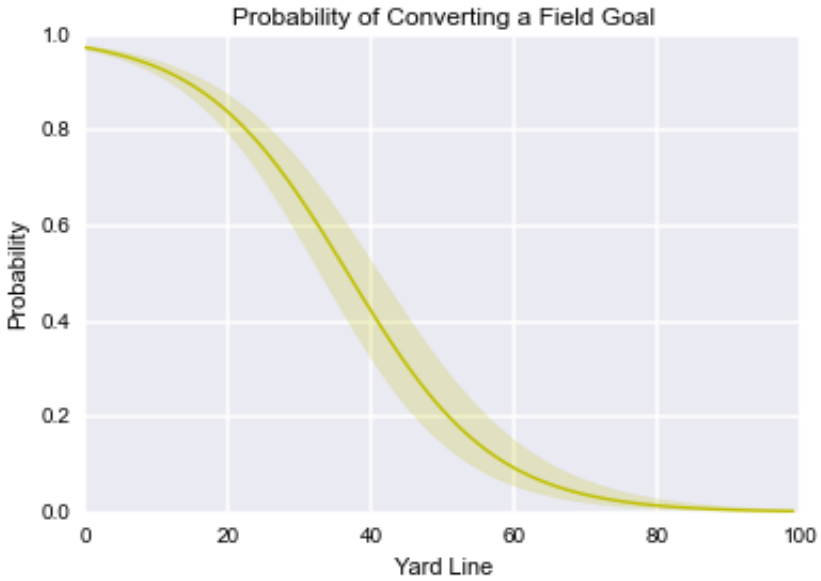
After evaluating and changing team ranks, a valid estimation of the probabilities of conversion for field goals as well as fourth down conversions must be performed. For each of these probabilities the outcome is binary, either the kick or fourth down attempt was converted or not. Therefore, we must implement a model that is used for a binary response. The types of models that are most commonly performed in these cases are logistic regression models or probit regression models. In our particular case we will use the logistic regression model because our model may not be normal, which is an assumption of the probit regression. The assumptions of the logistic regression model are that the response is binary, and the explanatory variables are independent. Obviously, the response is binary, and we found no

reason that in either equation our explanatory variables were mutually related. Therefore, we ran our models and computed 95 percent confidence intervals for given situations.

From the graph below, it is obvious that the probability of a conversion attempt is very close to 1 when it is a short kick (a kick with 0 yards to your goal line, or a 17 yard field goal). This is not surprising at all, because extra point attempts are from 20 yards away and kickers make about 98 percent of extra point attempts. However, as one gets further and further away it may surprise some people that the percentage decrease for very long field goal attempts isn't more drastic. When you have 50 yards to go (a 67-yard field goal) our model predicts about a 20 percent conversion rate. This obviously is too high; no one has ever converted a field goal of longer than 63 yards. There are three major reasons why these probabilities are clearly inflated. Firstly, only the kickers with huge legs like Sebastian Janikowski kick field goals greater than 60 yards. Secondly, they usually kick these field goals in nice weather, possibly in Denver, where the altitude might give a kicker 5 or so yards of added range. And lastly, 67 yard field goals are not in our data set. The logistic regression model does not accurately predict response values when the explanatory variables are outside of the range in the dataset. For these reasons, deep field goal probability calculations are too high. From about the 40 yard line and in, however, our results are more than believable. But, field goal success rate also depends on your kicker and on the weather. These results average over all kickers and different weather. If you are kicking in snow, or have a terrible or great kicker, the coach should adjust the expected value of kicking a field goal accordingly.

The following graph has probabilities of converting field goals on the y-axis and the yardline the field goal is attempted from. In the midterm presentation the shading on the following graph was purely aesthetic. After creating vectors of the standard errors the following graph now contains the actual confidence interval.

In [25]:
```
fg_prob = field_goal_result.predict([[1, x] for x in yard], True)
prob_upper = [exp(fg_vec_upper[0] + fg_vec_upper[1]*g)/(1 + exp(fg_vec_upper[
0] + fg_vec_upper[1]*g)) for g in yard]
prob_lower = [exp(fg_vec_lower[0] + fg_vec_lower[1]*k)/(1 + exp(fg_vec_lower[
0] + fg_vec_lower[1]*k)) for k in yard]
fig = plt.gcf()
plt.plot(yard, np.array(fg_prob), linestyle = '-', color = 'y')
plt.fill_between(yard, prob_lower, prob_upper, color='y', alpha=.2)
xlabel('Yard Line')
ylabel('Probability')
title('Probability of Converting a Field Goal')
savefig('fieldgoalprob.pdf')
```



In the following cell we create binary columns in the fourth down play data set to indicate what decision was made on the fourth down.

In [26]:
```
fourthDownPlays["Punt"] = [1]*cond2
fourthDownPlays["Field Goal"] = [1]*cond
fourthDownPlays["Go For It"] = 1 - ([1]*cond2 + [1]*cond)
```

First off, we must explain fourth downs in our data set. Because all fourth quarter plays were extracted from our data set, and the prevailing conventional wisdom is to not attempt fourth downs unless it is a late game situation, there weren't enough fourth down plays to make accurate predictions. Therefore, we accounted for this by calculating the probability of third down conversion. This seems odd, but in today's NFL, the third down is of a similar mind set to fourth down if you went for it. Teams aggressively try and convert the first because they nearly always punt if they do not convert. There are however, some third down situations that are not similar to fourth down situations. For instance, many times on third and long, teams will run the ball and concede punting just to not throw an interception or create more room for their punter. In these situations, if it was fourth down they would obviously be going more aggressively for the first. Because of this, all third down plays of greater than 10 yards which were runs were taken out of the model. We felt these observations would bias our results.

After using these particular plays to model fourth down plays, we know that the probability of a fourth down conversion should change depending on the strength of the offense and opposing teams' defense, the number of yards to go for a first down, and the field position. From our model, you can see that as the yards to go increases the probability decreases. Also, the better the offense is in comparison to the defense increases the probability of conversion. And the yard line we turned into a categorical variable. This categorical variable indicates whether the fourth down conversion is from the other teams' 0 – 10, 10 – 30, or 30-50 yard line, or your own 0-10, 10-30, or 30-50 yard line. You can see from these probabilities that the conversion rate from the other teams' 0-10 is lowest (it is the one left out of the equation) because all the numbers are positive. Then, as one might expect the conversion rate if you have greater than 90 yards to go is lower in relation to the other yard lines as well because teams are nervous about safety possibilities and other factors.

The following pulls out the rankings of teams and yards to go to convert on the fourth down. We also perform a logistic regression to determine the likelihood of converting given the rankings and yards to go. Third down plays are used in the logistic regression since using the fourth down plays gives a bias in the estimate.

In [27]:
```
thirdDown = pbp_data[pbp_data['down'] == 3]
thirdPlayNumbers = thirdDown.index
thirdPlayNumbers2 = thirdPlayNumbers[1:len(thirdPlayNumbers)]
thirdPlayNumbers2 = np.append(thirdPlayNumbers2, 0)
nonPenaltyThirdDown = thirdPlayNumbers[thirdPlayNumbers2 - thirdPlayNumbers !
= 1]
thirdDownPlays = pbp_data.ix[nonPenaltyThirdDown, :]
thirdDownPlays.ix[:,0:15].head()

FGcond = thirdDownPlays['description'].str.contains('field goal' or 'Field Go
al' or 'field Goal' or 'Field goal' or 'FIELD GOAL')
go_for_it_third = thirdDownPlays[FGcond == False]
passingCond = thirdDownPlays['description'].str.contains('pass' or 'Pass' or
'PASS' or 'sacked' or 'scramble' or 'Scramble')
```

```
greaterthan10Cond = thirdDownPlays['togo'] > 10
nonBiasedPlays = logical_or(thirdDownPlays['togo'] <= 10, logical_and(passing
Cond == True, greaterthan10Cond == True))
go_for_it_third_2 = go_for_it_third[nonBiasedPlays == True]
play_after_going_for_it_third = pbp_data.ix[np.array(go_for_it_third_2.index)
 + 1, :]
play_after_gfi_important_vars_third = play_after_going_for_it_third[['ydline'
, 'description', 'down', 'offrank', 'defrank', 'offrankMid', 'offrank31t32']]
play_after_gfi_important_vars_third.rename(columns=lambda x: 'next_' + x, inp
lace=True)
play_after_gfi_important_vars_third.index = go_for_it_third_2.index
go_for_it_third_3 = go_for_it_third_2.join(play_after_gfi_important_vars_thir
d)
go_for_it_third_3['converted'] = logical_and(go_for_it_third_3['offrank'] ==
go_for_it_third_3['next_offrank'], logical_or(go_for_it_third_3['next_down']
== 1, go_for_it_third_3.next_down.isin([1,2,3,4]) == False))
go_for_it_third_final = go_for_it_third_3[['offrank', 'defrank', 'ydline', 't
ogo', 'offrankMid', 'offrank31t32', 'defrankMid', 'defrank31t32', 'converted'
]]
go_for_it_third_final['intercept'] = [1]*len(go_for_it_third_final)
go_for_it_third_final = go_for_it_third_final.dropna(axis = 0, how = "any")
go_for_it_third_final[['ydline90t100']] = go_for_it_third_final[['ydline']] >
= 90
go_for_it_third_final[['ydline70t90']] = logical_and(go_for_it_third_final[['
ydline']] >= 70, go_for_it_third_final[['ydline']] < 90)
go_for_it_third_final[['ydline50t70']] = logical_and(go_for_it_third_final[['
ydline']] >= 50, go_for_it_third_final[['ydline']] < 70)
go_for_it_third_final[['ydline30t50']] = logical_and(go_for_it_third_final[['
ydline']] >= 30, go_for_it_third_final[['ydline']] < 50)
go_for_it_third_final[['ydline10t30']] = logical_and(go_for_it_third_final[['
ydline']] >= 10, go_for_it_third_final[['ydline']] < 30)
go_for_it_third_final[['ydline90t100']] = [1]*go_for_it_third_final[['ydline9
0t100']]
go_for_it_third_final[['ydline70t90']] = [1]*go_for_it_third_final[['ydline70
t90']]
go_for_it_third_final[['ydline50t70']] = [1]*go_for_it_third_final[['ydline50
t70']]
go_for_it_third_final[['ydline30t50']] = [1]*go_for_it_third_final[['ydline30
t50']]
go_for_it_third_final[['ydline10t30']] = [1]*go_for_it_third_final[['ydline10
t30']]

go_for_it_logit = sm.Logit(go_for_it_third_final[['converted']], go_for_it_th
ird_final[['intercept', 'togo', 'ydline10t30', 'ydline30t50', 'ydline50t70',
'ydline70t90', 'ydline90t100', 'offrankMid', 'offrank31t32', 'defrankMid', 'd
efrank31t32']])
go_for_it_result = go_for_it_logit.fit()
```

In [27]:
```
go_for_it_result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.639043
         Iterations 5
```

Out[27]:

Logit Regression Results

Dep. Variable:	converted	**No. Observations:**	72787
Model:	Logit	**Df Residuals:**	72776
Method:	MLE	**Df Model:**	10
Date:	Wed, 30 Apr 2014	**Pseudo R-squ.:**	0.06244
Time:	08:34:49	**Log-Likelihood:**	-46514.
converged:	True	**LL-Null:**	-49612.
		LLR p-value:	0.000

	coef	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
intercept	0.3322	0.035	9.441	0.000	0.263 0.401
togo	-0.1417	0.002	-68.676	0.000	-0.146 -0.138
ydline10t30	0.2740	0.035	7.912	0.000	0.206 0.342
ydline30t50	0.4415	0.033	13.261	0.000	0.376 0.507
ydline50t70	0.4682	0.032	14.554	0.000	0.405 0.531
ydline70t90	0.4394	0.033	13.149	0.000	0.374 0.505
ydline90t100	0.3284	0.063	5.242	0.000	0.206 0.451
offrankMid	-0.0136	0.001	-15.401	0.000	-0.015 -0.012
offrank31t32	-0.4547	0.036	-12.807	0.000	-0.524 -0.385
defrankMid	0.0062	0.001	7.073	0.000	0.005 0.008
defrank31t32	0.2090	0.035	5.922	0.000	0.140 0.278

In [28]:
```
yards_to_go = np.array(range(0, 21))
gfi_vec = np.array(go_for_it_result.params)
gfi_vec_upper = np.array(go_for_it_result.conf_int()[1])
gfi_vec_lower = np.array(go_for_it_result.conf_int()[0])
```

convert_prob is a three dimensional matrix that gives the probability of the offensive team converting a first down given how many yards to go for the first down, the offensive rank of the offense and the defensive rank of the defense.

In [29]:
```
convert_prob = np.ndarray(shape = (10, 32, 32)) #i -> yards to go, j -> offen
sive rank, k -> defensive rank
```

```
convert_prob[:][:][:] = 0
convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
for i in range(0, 10):
    for j in range(0, 4):
        for k in range(0, 4):
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)


convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
for i in range(0, 10):
    for j in range(0, 4):
        for k in range(4, 30):
            convert_vec[9] = k + 1
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)


convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]
for i in range(0, 10):
    for j in range(0, 4):
        for k in range(30, 32):
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)


convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
for i in range(0, 10):
    for j in range(4, 30):
        for k in range(0, 4):
            convert_vec[7] = j + 1
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)

convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
for i in range(0, 10):
    for j in range(4, 30):
        for k in range(4, 30):
            convert_vec[7] = j + 1
            convert_vec[9] = k + 1
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
```

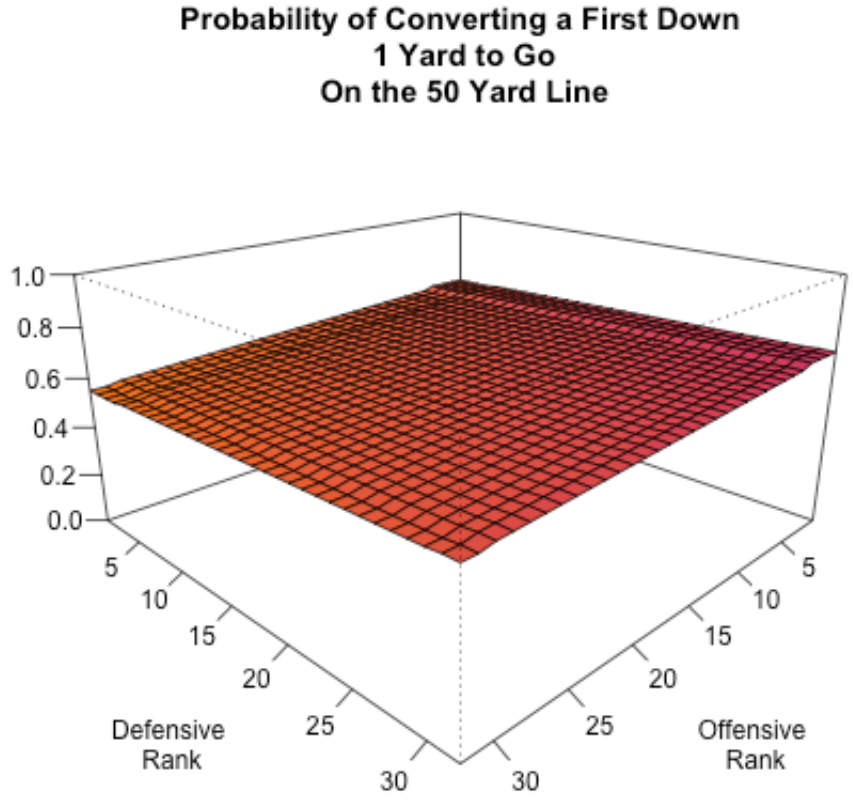
```
e)

convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]
for i in range(0, 10):
    for j in range(4, 30):
        for k in range(30, 32):
            convert_vec[7] = j + 1
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)


convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]
for i in range(0, 10):
    for j in range(30, 32):
        for k in range(0, 4):
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)

convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]
for i in range(0, 10):
    for j in range(30, 32):
        for k in range(4, 30):
            convert_vec[9] = k + 1
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)

convert_vec = [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1]
for i in range(0, 10):
    for j in range(30, 32):
        for k in range(30, 32):
            convert_vec[1] = i
            convert_prob[i][j][k] = go_for_it_result.predict(convert_vec, Tru
e)
```

In [30]:

```
z1 = np.array(convert_prob[1])
z2 = np.array(convert_prob[3])
z3 = np.array(convert_prob[6])
z4 = np.array(convert_prob[9])
np.savetxt('z1.txt', z1, delimiter = ',')
np.savetxt('z2.txt', z2, delimiter = ',')
np.savetxt('z3.txt', z3, delimiter = ',')
np.savetxt('z4.txt', z4, delimiter = ',')
```

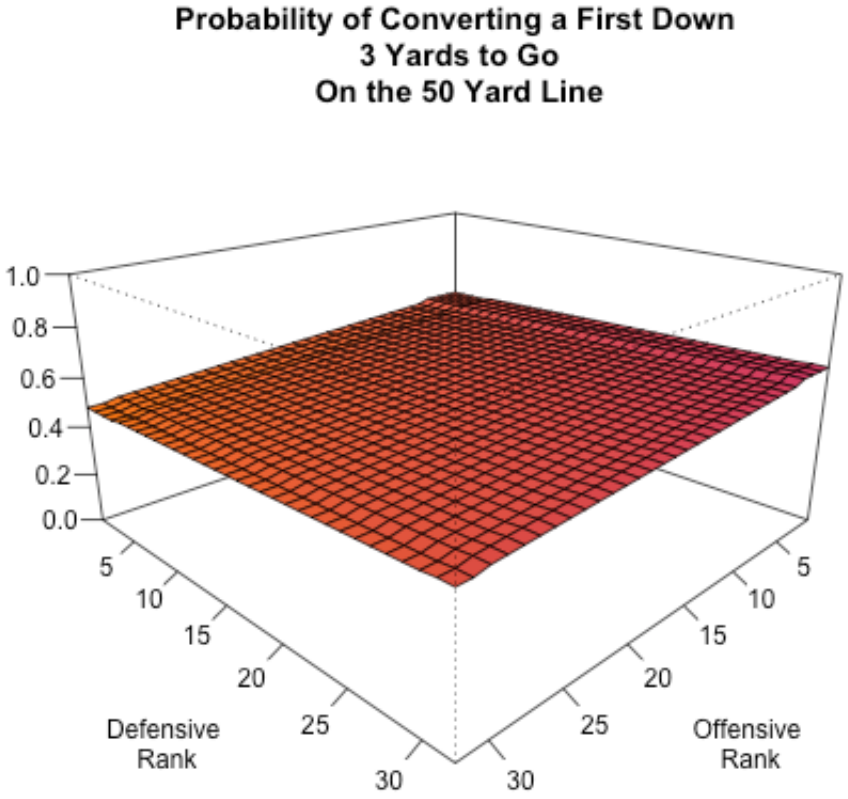
The following shows the probability of conversion of first down per offensive and defensive rankings with 1 yard to go.

In [31]:
```
%%R
z1 <- read.csv("z1.txt", header = F)
z1 <- as.matrix(z1)
x <- c(1:32); y <- c(1:32)
png("z1plot.pdf")
persp(x, y, z1, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 1 Yard to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
dev.off()
persp(x, y, z1, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 1 Yard to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
```



The following shows the probability of conversion of first down per offensive and defensive rankings with 3 yard to go.

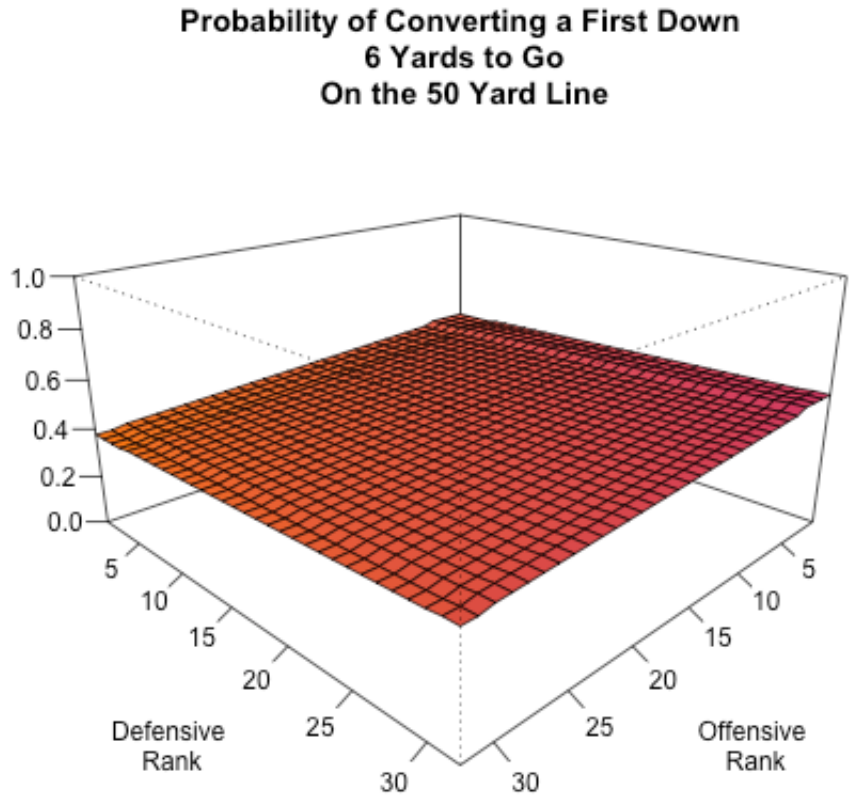
In [32]:
```
%%R
z2 <- read.csv("z2.txt", header = F)
z2 <- as.matrix(z2)
x <- c(1:32); y <- c(1:32)
png("z2plot.png")
persp(x, y, z2, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 3 Yards to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
dev.off()
persp(x, y, z2, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 3 Yards to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
```



The following shows the probability of conversion of first down per offensive and defensive rankings with 6 yard to go.

In [33]:
```
%%R
```

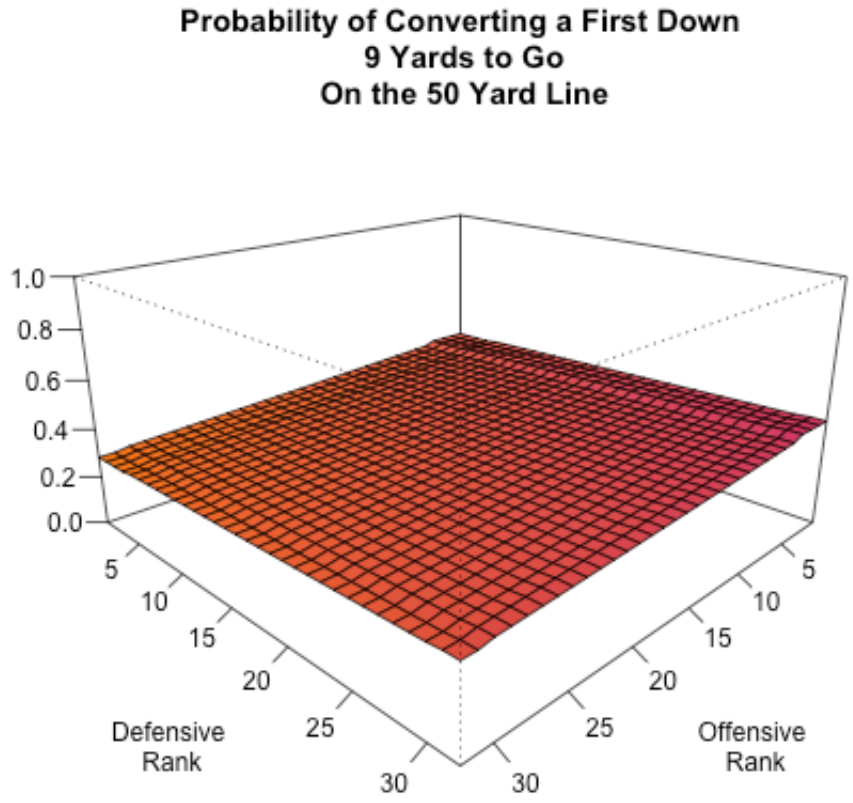
```
z3 <- read.csv("z3.txt", header = F)
z3 <- as.matrix(z3)
x <- c(1:32); y <- c(1:32)
png("z3plot.png")
persp(x, y, z3, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 6 Yards to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
dev.off()
persp(x, y, z3, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 6 Yards to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
```



The following shows the probability of conversion of first down per offensive and defensive rankings with 9 yard to go.

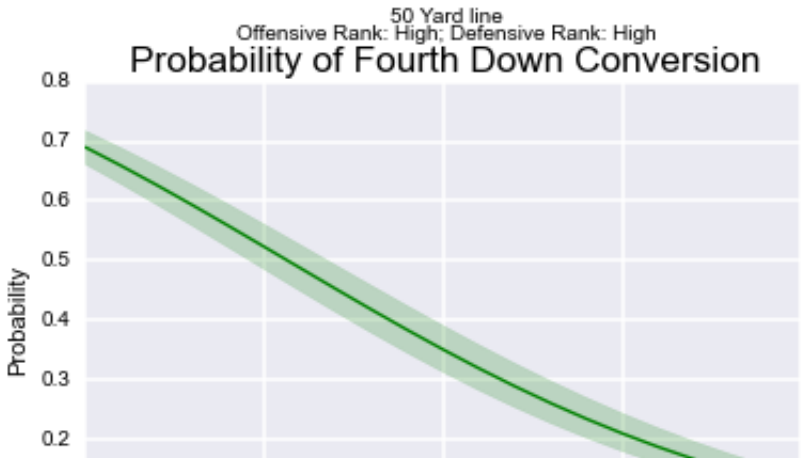
In [34]:
```
%%R
z4 <- read.csv("z4.txt", header = F)
z4 <- as.matrix(z4)
```

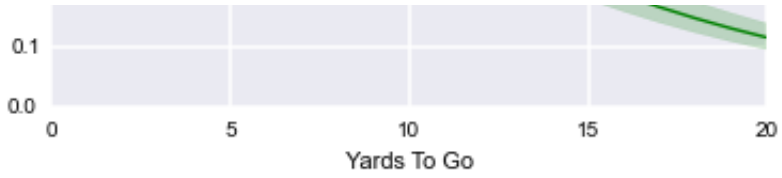
```
x <- c(1:32); y <- c(1:32)
png("z4plot.png")
persp(x, y, z4, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 9 Yards to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
dev.off()
persp(x, y, z4, expand = 0.5, zlim = range(0, 1), col = rgb(t(z1[-32, -32]) +
 0.2, (t(z1[-32, -32]))^2, z1[-32, -32] - 0.3), box = T, theta = 135, phi = 2
0, zlab = "", xlab = "", ylab = "", main = "Probability of Converting a First
 Down\n 9 Yards to Go\n On the 50 Yard Line", ticktype = "detailed")
text(-0.3,-0.35, "Defensive\n Rank")
text(0.3,-0.35, "Offensive\n Rank")
```



In [35]:
```
gfi_1_1 = [0]*21
gfi_1_1_upper = [0]*21
gfi_1_1_lower = [0]*21
for i in range(21):
    vec = [1, i, 0, 0, 1, 0, 0, 0, 0, 0, 0]
    gfi_1_1[i] = exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec[1]) + vec[2]*(gfi_ve
```

```
c[2]) * vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[5]*(gfi_vec[5]) + vec
[6]*(gfi_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec[8]) + vec[9]*(gfi_ve
c[9]) + vec[10]*(gfi_vec[10]))/(1 + exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec[1
]) + vec[2]*(gfi_vec[2]) * vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[5]
*(gfi_vec[5]) + vec[6]*(gfi_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec[8
]) + vec[9]*(gfi_vec[9]) + vec[10]*(gfi_vec[10])))
    gfi_1_1_lower[i] = exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_vec_lower[1]
) + vec[2]*(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*(gfi_vec_l
ower[4]) + vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) + vec[7]*(gf
i_vec_lower[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower[9]) + vec
[10]*(gfi_vec_lower[10]))/(1 + exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_vec_
lower[1]) + vec[2]*(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*(g
fi_vec_lower[4]) + vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) + ve
c[7]*(gfi_vec_lower[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower[9
]) + vec[10]*(gfi_vec_lower[10])))
    gfi_1_1_upper[i] = exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_vec_upper[1]
) + vec[2]*(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*(gfi_vec_u
pper[4]) + vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) + vec[7]*(gf
i_vec_upper[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper[9]) + vec
[10]*(gfi_vec_upper[10]))/(1 + exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_vec_
upper[1]) + vec[2]*(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*(g
fi_vec_upper[4]) + vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) + ve
c[7]*(gfi_vec_upper[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper[9
]) + vec[10]*(gfi_vec_upper[10])))

fig = plt.gcf()
plt.plot(yards_to_go, gfi_1_1, linestyle = '-', color="g")
plt.fill_between(yards_to_go, gfi_1_1_lower, gfi_1_1_upper, color="g", alpha=
.2)
xlabel('Yards To Go')
ylabel('Probability')
xlim(xmax = 20)
text(4.2,.87,"Offensive Rank: High; Defensive Rank: High")
text(8.5,.899,"50 Yard line")
title('Probability of Fourth Down Conversion', size = "xx-large")
savefig('gfi_1_1_prob.pdf')
```

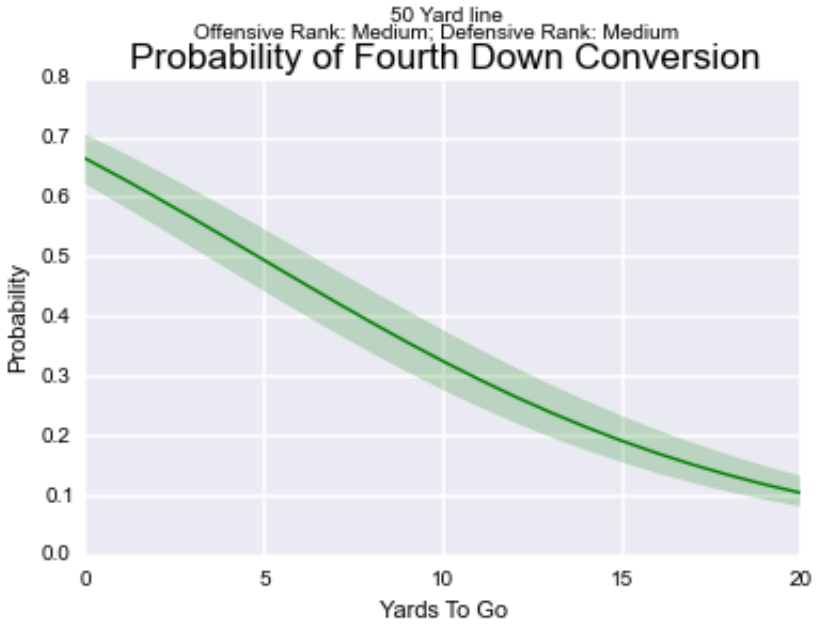




The following graph gives one particular example, of the effect of the number of yards to go on the probability conversion. This graph assumes that the offense and opposing teams defense are both near average (15th and 15th ranked), and the team making the decision is on the 50 yard line. You can clearly see that the probability of conversion is about .7 with one or fewer yards to go, and steadily goes down to about .1 with 20 yards to go. These numbers are pretty consistent with the numbers published in a few papers about league average (fourth and less than 1 is about .68) (according to one paper I read – need to find it!).

In [36]:
```
gfi_15_15 = [0]*21
gfi_15_15_upper = [0]*21
gfi_15_15_lower = [0]*21
for i in range(0, 21):
    vec = [1, i, 0, 0, 1, 0, 0, 15, 0, 15, 0]
    gfi_15_15[i] = exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec[1]) + vec[2]*(gfi_
vec[2]) * vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[5]*(gfi_vec[5]) + v
ec[6]*(gfi_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec[8]) + vec[9]*(gfi_
vec[9]) + vec[10]*(gfi_vec[10]))/(1 + exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec
[1]) + vec[2]*(gfi_vec[2]) * vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[
5]*(gfi_vec[5]) + vec[6]*(gfi_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec
[8]) + vec[9]*(gfi_vec[9]) + vec[10]*(gfi_vec[10])))
    gfi_15_15_lower[i] = exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_vec_lower[
1]) + vec[2]*(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*(gfi_vec
_lower[4]) + vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) + vec[7]*(
gfi_vec_lower[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower[9]) + v
ec[10]*(gfi_vec_lower[10]))/(1 + exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_ve
c_lower[1]) + vec[2]*(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*
(gfi_vec_lower[4]) + vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) +
vec[7]*(gfi_vec_lower[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower
[9]) + vec[10]*(gfi_vec_lower[10])))
    gfi_15_15_upper[i] = exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_vec_upper[
1]) + vec[2]*(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*(gfi_vec
_upper[4]) + vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) + vec[7]*(
gfi_vec_upper[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper[9]) + v
ec[10]*(gfi_vec_upper[10]))/(1 + exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_ve
c_upper[1]) + vec[2]*(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*
(gfi_vec_upper[4]) + vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) +
vec[7]*(gfi_vec_upper[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper
[9]) + vec[10]*(gfi_vec_upper[10])))

fig = plt.gcf()
plt.plot(yards_to_go, gfi_15_15, linestyle = '-', color="green")
plt.fill_between(yards_to_go, gfi_15_15_lower, gfi_15_15_upper, color="green"
, alpha=.2)
```

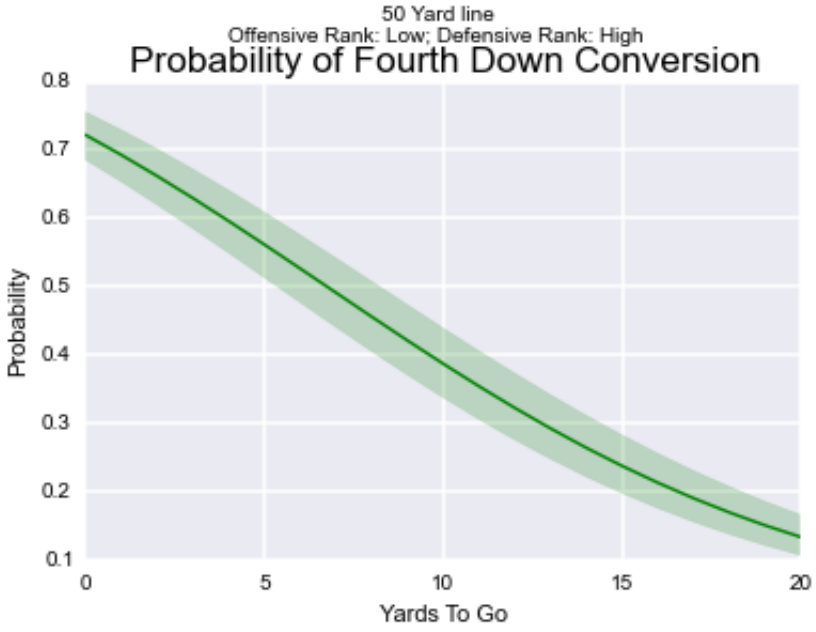
```
xlabel('Yards To Go')
ylabel('Probability')
xlim(xmax = 20)
text(3,.87,"Offensive Rank: Medium; Defensive Rank: Medium")
text(8.5,.895,"50 Yard line")
title('Probability of Fourth Down Conversion', size = "xx-large")
savefig('gfi_15_15_prob.pdf')
```



If the offensive team's rank is high (5th) and the defensive rank is low (25th) one can see that the probability increases slightly when comparing across yards to go from the previous plot. Instead of the graph steadily decreasing from about .68 to .1 (0 to 20 yards away), the probabilities range from .72 to .14.

In [37]:
```
gfi_1_25 = [0]*21
gfi_1_25_upper = [0]*21
gfi_1_25_lower = [0]*21
for i in range(0, 21):
    vec = [1, i, 0, 0, 1, 0, 0, 0, 0, 25, 0]
    gfi_1_25[i] = exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec[1]) + vec[2]*(gfi_v
ec[2]) * vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[5]*(gfi_vec[5]) + ve
c[6]*(gfi_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec[8]) + vec[9]*(gfi_v
ec[9]) + vec[10]*(gfi_vec[10]))/(1 + exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec[
1]) + vec[2]*(gfi_vec[2]) * vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[5
]*(gfi_vec[5]) + vec[6]*(gfi_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec[
8]) + vec[9]*(gfi_vec[9]) + vec[10]*(gfi_vec[10])))
    gfi_1_25_lower[i] = exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_vec_lower[1
]) + vec[2]*(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*(gfi_vec_
lower[4]) + vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) + vec[7]*(g
fi_vec_lower[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower[9]) + ve
c[10]*(gfi_vec_lower[10]))/(1 + exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_vec
_lower[1]) + vec[2]*(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*(
gfi_vec_lower[4]) + vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) + v
```

```
ec[7]*(gfi_vec_lower[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower[
9]) + vec[10]*(gfi_vec_lower[10])))
    gfi_1_25_upper[i] = exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_vec_upper[1
]) + vec[2]*(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*(gfi_vec_
upper[4]) + vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) + vec[7]*(g
fi_vec_upper[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper[9]) + ve
c[10]*(gfi_vec_upper[10]))/(1 + exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_vec
_upper[1]) + vec[2]*(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*(
gfi_vec_upper[4]) + vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) + v
ec[7]*(gfi_vec_upper[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper[
9]) + vec[10]*(gfi_vec_upper[10])))

fig = plt.gcf()
plt.plot(yards_to_go, gfi_1_25, linestyle = '-',color= "g")
plt.fill_between(yards_to_go, gfi_1_25_lower, gfi_1_25_upper, color="g", alph
a=.2)
xlabel('Yards To Go')
xlim(xmax = 20)
ylabel('Probability')
text(4,.86,"Offensive Rank: High; Defensive Rank: Low")
text(8.3,.89,"50 Yard line")
title('Probability of Fourth Down Conversion', size = "xx-large")
savefig('gfi_1_25_prob.pdf')
```



In [38]:
```
score = score_logitResult.params
score.columns = ["DefTD","FG","NoPoints","TD"]

score_lower = pd.DataFrame(np.transpose([[14.831, -.214, -.014, -.656, -.010,
 -.315], [21.179, -.262, -.023, -1.344, 0.004, -.378], [20.5, -.228, -.01, -.
844, -.007, -.644], [21.493, -.258, -.039, -1.979, 0.013, 0.023]]))
score_lower.columns = ["DefTD","FG","NoPoints","TD"]
```

```
score_upper = pd.DataFrame(np.transpose([[19.373, -.165, 0.027, .704, 0.03, 1
.231], [25.589, -.215, 0.013, -.139, 0.039, 1.021], [24.906, -.181, 0.026, 0.
345, 0.028, 0.742], [25.902, -.211, -0.003, -.775, 0.048, 1.415]]))
score_upper.columns = ["DefTD","FG","NoPoints","TD"]

Punt = pd.DataFrame(score_logit_puntResult.params)
Punt.columns = ["DefTD","FG","NoPoints","TD"]

Punt_upper = pd.DataFrame(np.transpose([[-1.393, 0.067, 0.035, 0.678, 0.045,
1.417],[-1.185, 0.102, 0.021, -0.262, 0.050, 1.558],[2.191, 0.075, 0.036, 0.2
65, 0.036, 1.204],[-0.233, 0.095, 0.006, -0.808, 0.059, 2.070]]))
Punt_upper.columns = ["DefTD","FG","NoPoints","TD"]

Punt_lower = pd.DataFrame(np.transpose([[-3.617, 0.033, -0.020, -1.011, -0.00
7, -0.905],[-2.99, 0.073, -0.024, -1.624, 0.007, -.354],[0.423, 0.046, -.009,
 -1.061, -0.006, -.687],[-2.022, 0.066, -0.039, -2.164, 0.017, 0.170]]))
Punt_lower.columns = ["DefTD","FG","NoPoints","TD"]
```

The expected number of points is calculated by multiplying the possible outcomes of a drive by the point values given those outcomes. In each drive there are five possible outcomes: a touchdown or field goal scored by the offense, a safety or touchdown scored by the defense, and no points. These numbers are multiplied by the points of each outcome: 7, 3, -2, -7, and 0 respectively. For the model, we assume every touchdown is 7 points (not 6 or 8) which is true the vast majority of the time. In order to estimate the probabilities of these outcomes we decided to use a multinomial logistic regression model. A multinomial model is used for a categorical response variable (here five categories) when the explanatory variables are independent. Here, the explanatory variables for our model are the offensive rank and the defensive rank of the two respective teams as well as the starting field position. The assumption of independence seems reasonable.
If you notice, the calculations can get complicated if for instance the team who decides to punt gets a defensive touchdown on the other teams next drive. Or, if the opposing team to the one making the decision scores a safety because they get the ball back on offense after this play. These two situations were accounted for and implemented by simply multiplying the probability of the safety or defensive touchdown by the expected values of the next drive. By accounting for these possibilities we ensure that the observations are consistent.

From the multinomial and logistic calculations, we can calculate the expected point values given each decision. Then a confidence interval is computed for each of the three scenarios by taking each of the lower bounds and upper bounds of the combined intervals and adding them together. Because of the complexities in this calculation the estimated line is not in the middle of the confidence interval. Also, this interval is technically greater than 95 percent because we are combining other 95 percent confidence intervals.

The following functions will be used to determine the choice to be made given yard line, yards to go, and both offensive and defensive rankings of the team making the decision as well as the other team.

In [39]:
```
def init(ydline, ydtogo, oorank, odrank, ddrank, dorank):
    """This function initiates vectors that will later be used to calculate p
robabilities and expections based on the information provided"""
    oorank1 = 2 if 5 <= oorank <= 30 else 3 if 31 <= oorank <= 32 else 0
    Ioorank = 1 if oorank1 == 0 else 1 if oorank1 == 3 else oorank
```

```
    ddrank1 = 4 if 5 <= ddrank <= 30 else 5 if 31 <= ddrank <= 32 else 0
    Iddrank = 1 if ddrank1 == 0 else 1 if ddrank1 == 5 else ddrank
    odrank1 = 4 if 5 <= odrank <= 30 else 5 if 31 <= odrank <= 32 else 0
    Iodrank = 1 if odrank1 == 0 else 1 if odrank1 == 5 else odrank
    dorank1 = 2 if 5 <= dorank <= 30 else 3 if 31 <= dorank <= 32 else 0
    Idorank = 1 if dorank1 == 0 else 1 if dorank1 == 3 else dorank
    ooMidrank = oorank if 5 <= oorank <= 30 else 0
    ddMidrank = ddrank if 5 <= ddrank <= 30 else 0
    oo31t32rank = 1 if 31 <= oorank <= 32 else 0
    dd31t32rank = 1 if 31 <= ddrank <= 32 else 0
    NoPointsYdline = ydsDriven[0] + ydline*ydsDriven[1] + ooMidrank*ydsDriven
[2] + oo31t32rank*ydsDriven[3] + ddMidrank*ydsDriven[4] + dd31t32rank*ydsDriv
en[5]
    X_off = [1, ydline - ydtogo, 0, 0, 0, 0]; X_off[oorank1] = Ioorank; X_off
[ddrank1] = Iddrank
    X_def_score = [1, NoPointsYdline, 0, 0, 0, 0]; X_def_score[dorank1] = Ido
rank; X_def_score[odrank1] = Iodrank
    X_def_gfi_fail = [1, 100 - ydline, 0, 0, 0, 0]; X_def_gfi_fail[dorank1] =
 Idorank; X_def_gfi_fail[odrank1] = Iodrank
    X_def_20 = [1, 80, 0, 0, 0, 0]; X_def_20[dorank1] = Idorank; X_def_20[odr
ank1] = Iodrank
    X_def_fg_fail = [1, 93 - ydline, 0, 0, 0, 0]; X_def_fg_fail[dorank1] = Id
orank; X_def_fg_fail[odrank1] = Iodrank
    X_punt = [1, ydline, 0, 0, 0, 0]; X_punt[dorank1] = Idorank; X_punt[odran
k1] = Iodrank
    X = {'off': X_off, 'def_score': X_def_score, 'gfi_fail': X_def_gfi_fail,
'20': X_def_20, 'fg_fail': X_def_fg_fail, 'punt': X_punt}
    return X
```

In [40]:

```
def prob(ydline, ydtogo, oorank, odrank, ddrank, dorank):
    """This function finds the probability of converting a first down and con
verting a field goal under certain conditions.
       which = ("lower", "upper", "actual")."""
    oorank1 = 2 if 5 <= oorank <= 30 else 3 if 31 <= oorank <= 32 else 0
    Ioorank = 1 if oorank1 == 0 else 1 if oorank1 == 3 else oorank
    ddrank1 = 4 if 5 <= ddrank <= 30 else 5 if 31 <= ddrank <= 32 else 0
    Iddrank = 1 if ddrank1 == 0 else 1 if ddrank1 == 5 else ddrank
    oorank_gfi = 7 if 5 <= oorank <= 30 else 8 if 31 <= oorank <= 32 else 0
    ddrank_gfi = 9 if 5 <= ddrank <= 30 else 10 if 31 <= ddrank <= 32 else 0
    y_bucket = 2 if 10 <= ydline <= 29 else 3 if 30 <= ydline <= 49 else 4 if
 50 <= ydline <= 69 else 5 if 70 <= ydline <= 89 else 6 if 90 <= ydline <= 10
0 else 0
    vec = [1, ydtogo, 0, 0, 0, 0, 0, 0, 0, 0, 0]; vec[oorank_gfi] = Ioorank;
vec[ddrank_gfi] = Iddrank; vec[y_bucket] = 1

    fg = exp((fg_vec[0] + ydline*(fg_vec[1])))/(1 + exp((fg_vec[0] + ydline*(
fg_vec[1]))))
```

```
    gfi = exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec[1]) + vec[2]*(gfi_vec[2]) *
 vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[5]*(gfi_vec[5]) + vec[6]*(gf
i_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec[8]) + vec[9]*(gfi_vec[9]) +
 vec[10]*(gfi_vec[10]))/(1 + exp(vec[0]*gfi_vec[0] + vec[1]*(gfi_vec[1]) + ve
c[2]*(gfi_vec[2]) * vec[3]*(gfi_vec[3]) + vec[4]*(gfi_vec[4]) + vec[5]*(gfi_v
ec[5]) + vec[6]*(gfi_vec[6]) + vec[7]*(gfi_vec[7]) + vec[8]*(gfi_vec[8]) + ve
c[9]*(gfi_vec[9]) + vec[10]*(gfi_vec[10])))

    fg_l = exp((fg_vec_lower[0] + ydline*(fg_vec_lower[1])))/(1 + exp((fg_vec
_lower[0] + ydline*(fg_vec_lower[1]))))
    gfi_l = exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_vec_lower[1]) + vec[2]*
(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*(gfi_vec_lower[4]) +
vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) + vec[7]*(gfi_vec_lower
[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower[9]) + vec[10]*(gfi_v
ec_lower[10]))/(1 + exp(vec[0]*gfi_vec_lower[0] + vec[1]*(gfi_vec_lower[1]) +
 vec[2]*(gfi_vec_lower[2]) * vec[3]*(gfi_vec_lower[3]) + vec[4]*(gfi_vec_lowe
r[4]) + vec[5]*(gfi_vec_lower[5]) + vec[6]*(gfi_vec_lower[6]) + vec[7]*(gfi_v
ec_lower[7]) + vec[8]*(gfi_vec_lower[8]) + vec[9]*(gfi_vec_lower[9]) + vec[10
]*(gfi_vec_lower[10])))

    fg_u = exp((fg_vec_upper[0] + ydline*(fg_vec_upper[1])))/(1 + exp((fg_vec
_upper[0] + ydline*(fg_vec_upper[1]))))
    gfi_u = exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_vec_upper[1]) + vec[2]*
(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*(gfi_vec_upper[4]) +
vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) + vec[7]*(gfi_vec_upper
[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper[9]) + vec[10]*(gfi_v
ec_upper[10]))/(1 + exp(vec[0]*gfi_vec_upper[0] + vec[1]*(gfi_vec_upper[1]) +
 vec[2]*(gfi_vec_upper[2]) * vec[3]*(gfi_vec_upper[3]) + vec[4]*(gfi_vec_uppe
r[4]) + vec[5]*(gfi_vec_upper[5]) + vec[6]*(gfi_vec_upper[6]) + vec[7]*(gfi_v
ec_upper[7]) + vec[8]*(gfi_vec_upper[8]) + vec[9]*(gfi_vec_upper[9]) + vec[10
]*(gfi_vec_upper[10])))

    X = {'fg': [fg, fg_u, fg_l], 'gfi': [gfi, gfi_u, gfi_l]}
    return X
```

In [41]:

```
def log_score(ydline, ydtogo, oorank, odrank, ddrank, dorank, vec):
    """Using the results of one of the logistic regressions from above, value
s are calculated that will be used later to help in determining the expectati
on of points given go for it, punt or field goal decision."""

    Xsum = (1 + exp(sum(vec*score.ix[:,0])) + exp(sum(vec*score.ix[:,1])) + e
xp(sum(vec*score.ix[:,2])) + exp(sum(vec*score.ix[:,3])))
    DefTD = exp(sum(vec*score.ix[:,0]))/Xsum
    FG = exp(sum(vec*score.ix[:,1]))/Xsum
    NoPoints = exp(sum(vec*score.ix[:,2]))/Xsum
    TD = exp(sum(vec*score.ix[:,3]))/Xsum
    DefSafety = 1/Xsum
```

```
    Xsum_l = (1 + exp(sum(vec*score_lower.ix[:,0])) + exp(sum(vec*score_lower
.ix[:,1])) + exp(sum(vec*score_lower.ix[:,2])) + exp(sum(vec*score_lower.ix[:
,3])))
    DefTD_l = exp(sum(vec*score_lower.ix[:,0]))/Xsum_l
    FG_l = exp(sum(vec*score_lower.ix[:,1]))/Xsum_l
    NoPoints_l = exp(sum(vec*score_lower.ix[:,2]))/Xsum_l
    TD_l = exp(sum(vec*score_lower.ix[:,3]))/Xsum_l
    DefSafety_l = 1/Xsum_l

    Xsum_u = (1 + exp(sum(vec*score_upper.ix[:,0])) + exp(sum(vec*score_upper
.ix[:,1])) + exp(sum(vec*score_upper.ix[:,2])) + exp(sum(vec*score_upper.ix[:
,3])))
    DefTD_u = exp(sum(vec*score_upper.ix[:,0]))/Xsum_u
    FG_u = exp(sum(vec*score_upper.ix[:,1]))/Xsum_u
    NoPoints_u = exp(sum(vec*score_upper.ix[:,2]))/Xsum_u
    TD_u = exp(sum(vec*score_upper.ix[:,3]))/Xsum_u
    DefSafety_u = 1/Xsum_u

    X = {'DefTD': [DefTD, DefTD_u, DefTD_l], 'FG': [FG, FG_u, FG_l], 'NoPoint
s': [NoPoints, NoPoints_u, NoPoints_l], 'TD': [TD, TD_u, TD_l], 'DefSafety':
[DefSafety, DefSafety_u, DefSafety_l]}
    return X
```

In [42]:

```
def log_punt(ydline, ydtogo, oorank, odrank, ddrank, dorank, vec):
    """Using the results of one of the logistic regressions from above, value
s are calculated that will be used later to help in determining the expectati
on of points given go for it, punt or field goal decision."""
    Xsum = (1 + exp(sum(vec*Punt.ix[:,0])) + exp(sum(vec*Punt.ix[:,1])) + exp
(sum(vec*Punt.ix[:,2])) + exp(sum(vec*Punt.ix[:,3])))
    DefTD = exp(sum(vec*Punt.ix[:,0]))/Xsum
    FG = exp(sum(vec*Punt.ix[:,1]))/Xsum
    NoPoints = exp(sum(vec*Punt.ix[:,2]))/Xsum
    TD = exp(sum(vec*Punt.ix[:,3]))/Xsum
    DefSafety = 1/Xsum

    Xsum_l = (1 + exp(sum(vec*Punt_lower.ix[:,0])) + exp(sum(vec*Punt_lower.i
x[:,1])) + exp(sum(vec*Punt_lower.ix[:,2])) + exp(sum(vec*Punt_lower.ix[:,3])
))
    DefTD_l = exp(sum(vec*Punt_lower.ix[:,0]))/Xsum_l
    FG_l = exp(sum(vec*Punt_lower.ix[:,1]))/Xsum_l
    NoPoints_l = exp(sum(vec*Punt_lower.ix[:,2]))/Xsum_l
    TD_l = exp(sum(vec*Punt_lower.ix[:,3]))/Xsum_l
    DefSafety_l = 1/Xsum_l

    Xsum_u = (1 + exp(sum(vec*Punt_upper.ix[:,0])) + exp(sum(vec*Punt_upper.i
x[:,1])) + exp(sum(vec*Punt_upper.ix[:,2])) + exp(sum(vec*Punt_upper.ix[:,3])
```

```
))
    DefTD_u = exp(sum(vec*Punt_upper.ix[:,0]))/Xsum_u
    FG_u = exp(sum(vec*Punt_upper.ix[:,1]))/Xsum_u
    NoPoints_u = exp(sum(vec*Punt_upper.ix[:,2]))/Xsum_u
    TD_u = exp(sum(vec*Punt_upper.ix[:,3]))/Xsum_u
    DefSafety_u = 1/Xsum_u

    X = {'DefTD': [DefTD, DefTD_u, DefTD_l], 'FG': [FG, FG_u, FG_l], 'NoPoint
s': [NoPoints, NoPoints_u, NoPoints_l], 'TD': [TD, TD_u, TD_l], 'DefSafety':
[DefSafety, DefSafety_u, DefSafety_l]}
    return X
```

In [43]:

```
def gfi_expect(ydline, ydtogo, oorank, odrank, ddrank, dorank, which):
    """Calculates the point expectation given the offensive team goes for the
 first down."""
    x = init(ydline, ydtogo, oorank, odrank, ddrank, dorank)
    y = prob(ydline, ydtogo, oorank, odrank, ddrank, dorank)
    X20 = log_score(ydline, ydtogo, oorank, odrank, ddrank, dorank, x['20'])
    XDS = log_score(ydline, ydtogo, oorank, odrank, ddrank, dorank, x['def_sc
ore'])
    XGFI = log_score(ydline, ydtogo, oorank, odrank, ddrank, dorank, x['gfi_f
ail'])
    XOFF = log_score(ydline, ydtogo, oorank, odrank, ddrank, dorank, x['off']
)
    if which == "actual":
        if ydline <= ydtogo:
            E_gfi = y['gfi'][0]*(7 - (X20['DefTD'][0]*(-7) + X20['FG'][0]*(3)
 + X20['TD'][0]*(7) + X20['DefSafety'][0]*(-2))) - (1 - y['gfi'][0])*(XGFI['D
efTD'][0]*(-7) + XGFI['FG'][0]*(3) + XGFI['TD'][0]*(7) + XGFI['DefSafety'][0]
*(-2))
        else:
            E_gfi = y['gfi'][0]*((XOFF['FG'][0]*(3) + XOFF['TD'][0]*(7) + XOF
F['DefSafety'][0]*(-2) + XOFF['DefTD'][0]*(-7)) - (XOFF['FG'][0] + XOFF['TD']
[0] + XOFF['DefSafety'][0])*(X20['DefTD'][0]*(-7) + X20['FG'][0]*(3) + X20['T
D'][0]*(7) + X20['DefSafety'][0]*(-2)) - XOFF['NoPoints'][0]*(XDS['FG'][0]*(3
) + XDS['TD'][0]*(7) + XDS['DefSafety'][0]*(-2) + XDS['DefTD'][0]*(-7))) - (1
 - y['gfi'][0])*(XGFI['DefTD'][0]*(-7) + XGFI['FG'][0]*(3) + XGFI['TD'][0]*(7
) + XGFI['DefSafety'][0]*(-2))

    if which == "lower":
        if ydline <= ydtogo:
            E_gfi = min(y['gfi'])*(7 + (min(X20['DefTD'])*(7) + min(X20['FG']
)*(3) + min(X20['TD'])*(7) + min(X20['DefSafety'])*(-2))) - (1 - min(y['gfi']
))*(min(XGFI['DefTD'])*(-7) + min(XGFI['FG'])*(3) + min(XGFI['TD'])*(7) + min
(XGFI['DefSafety'])*(-2))
        else:
            E_gfi = min(y['gfi'])*((min(XOFF['FG'])*(3) + min(XOFF['TD'])*(7)
```

```
 + max(XOFF['DefSafety'])*(-2) + max(XOFF['DefTD'])*(-7)) - (max(XOFF['FG'])
+ max(XOFF['TD']) + max(XOFF['DefSafety']))*(min(X20['DefTD'])*(-7) + max(X20
['FG'])*(3) + max(X20['TD'])*(7) + min(X20['DefSafety'])*(-2)) - max(XOFF['No
Points'])*(max(XDS['FG'])*(3) + max(XDS['TD'])*(7) + min(XDS['DefSafety'])*(-
2) + min(XDS['DefTD'])*(-7))) - (1 - min(y['gfi']))*(min(XGFI['DefTD'])*(-7)
+ max(XGFI['FG'])*(3) + max(XGFI['TD'])*(7) + min(XGFI['DefSafety'])*(-2))

    if which == "upper":
        if ydline <= ydtogo:
            E_gfi = max(y['gfi'])*(7 + (max(X20['DefTD'])*(7) + max(X20['FG']
)*(3) + max(X20['TD'])*(7) + min(X20['DefSafety'])*(-2))) - (1 - max(y['gfi']
))*(max(XGFI['DefTD'])*(-7) + max(XGFI['FG'])*(3) + max(XGFI['TD'])*(7) + max
(XGFI['DefSafety'])*(-2))
        else:
            E_gfi = max(y['gfi'])*((max(XOFF['FG'])*(3) + max(XOFF['TD'])*(7)
 + min(XOFF['DefSafety'])*(-2) + min(XOFF['DefTD'])*(-7)) - (min(XOFF['FG'])
+ min(XOFF['TD']) + min(XOFF['DefSafety']))*(max(X20['DefTD'])*(-7) + min(X20
['FG'])*(3) + min(X20['TD'])*(7) + max(X20['DefSafety'])*(-2)) - min(XOFF['No
Points'])*(min(XDS['FG'])*(3) + min(XDS['TD'])*(7) + max(XDS['DefSafety'])*(-
2) + max(XDS['DefTD'])*(-7))) - (1 - max(y['gfi']))*(max(XGFI['DefTD'])*(-7)
+ min(XGFI['FG'])*(3) + min(XGFI['TD'])*(7) + max(XGFI['DefSafety'])*(-2))

    return E_gfi
```

In [44]:

```
def fg_expect(ydline, ydtogo, oorank, odrank, ddrank, dorank, which):
    """Calculates the point expectation given the offensive team goes for a f
ield goal."""
    x = init(ydline, ydtogo, oorank, odrank, ddrank, dorank)
    y = prob(ydline, ydtogo, oorank, odrank, ddrank, dorank)
    X20 = log_score(ydline, ydtogo, oorank, odrank, ddrank, dorank, x['20'])
    XFG = log_score(ydline, ydtogo, oorank, odrank, ddrank, dorank, x['fg_fai
l'])
    if which == "actual":
        E_fg = y['fg'][0]*(3 - (X20['DefTD'][0]*(-7) + X20['FG'][0]*(3) + X20
['TD'][0]*(7) + X20['DefSafety'][0]*(-2))) - (1 - y['fg'][0])*(XFG['DefTD'][0
]*(-7) + XFG['FG'][0]*(3) + XFG['TD'][0]*(7) + XFG['DefSafety'][0]*(-2))

    if which == "lower":
        E_fg = min(y['fg'])*(3 - (min(X20['DefTD'])*(-7) + max(X20['FG'])*(3)
 + max(X20['TD'])*(7) + min(X20['DefSafety'])*(-2))) - (1 - min(y['fg']))*(mi
n(XFG['DefTD'])*(-7) + max(XFG['FG'])*(3) + max(XFG['TD'])*(7) + min(XFG['Def
Safety'])*(-2))

    if which == "upper":
        E_fg = max(y['fg'])*(3 - (max(X20['DefTD'])*(-7) + min(X20['FG'])*(3)
 + min(X20['TD'])*(7) + max(X20['DefSafety'])*(-2))) - (1 - max(y['fg']))*(ma
x(XFG['DefTD'])*(-7) + min(XFG['FG'])*(3) + min(XFG['TD'])*(7) + max(XFG['Def
```

```
Safety'])*(-2))

    return E_fg
```

In [45]:
```
def punt_expect(ydline, ydtogo, oorank, odrank, ddrank, dorank, which):
    """Calculates the point expectation given the offensive team punts."""
    x = init(ydline, ydtogo, oorank, odrank, ddrank, dorank)
    XP = log_punt(ydline, ydtogo, oorank, odrank, ddrank, dorank, x['punt'])
    if which == "actual":
        E_punt = - (XP['DefTD'][0]*(-7) + XP['FG'][0]*(3) + XP['TD'][0]*(7) +
 XP['DefSafety'][0]*(-2))

    if which == "lower":
        E_punt = - (min(XP['DefTD'])*(-7) + max(XP['FG'])*(3) + max(XP['TD'])
*(7) + min(XP['DefSafety'])*(-2))

    if which == "upper":
        E_punt = - (max(XP['DefTD'])*(-7) + min(XP['FG'])*(3) + min(XP['TD'])
*(7) + max(XP['DefSafety'])*(-2))

    return E_punt
```

In [46]:
```
def decision(ydline, ydtogo, oorank, odrank, ddrank, dorank, which):
    """Returns the three expectations calculated above to help the user deter
mine which decision is best."""

    gfi = gfi_expect(ydline, ydtogo, oorank, odrank, ddrank, dorank, which)
    fg = fg_expect(ydline, ydtogo, oorank, odrank, ddrank, dorank, which)
    punt = punt_expect(ydline, ydtogo, oorank, odrank, ddrank, dorank, which)

    dec = {'Go For It': gfi, 'Field Goal': fg, 'Punt': punt}
    return dec
```

The next plots illustrate how the yard-line affects the optimal decision, with given offensive and defensive ranks of both the team making the decision and the opposing team (all ranks around 15). In the first plot the optimal decision is to kick a field goal regardless of the number of yards to go. However, with less than one yard to go the confidence intervals overlap which means it may be optimal to go for the first down. The coach should make the final decision if the intervals overlap. With 50 yards to go, and the same rankings, the optimal decision changes in a pretty major way. Now, the graph indicates that a team should go for the first down on fourth and less than 6, between 6 and 12 yards either punt or go for it, and past 12 yards punt. From a team's own 20 yard line (80 yards to the goal line), a team's optimal decision is to go for it on fourth and less than 3, either punt or go for it between 3 and 9 yards to go, and punt the ball if there are more than 9 yards to go for a first.

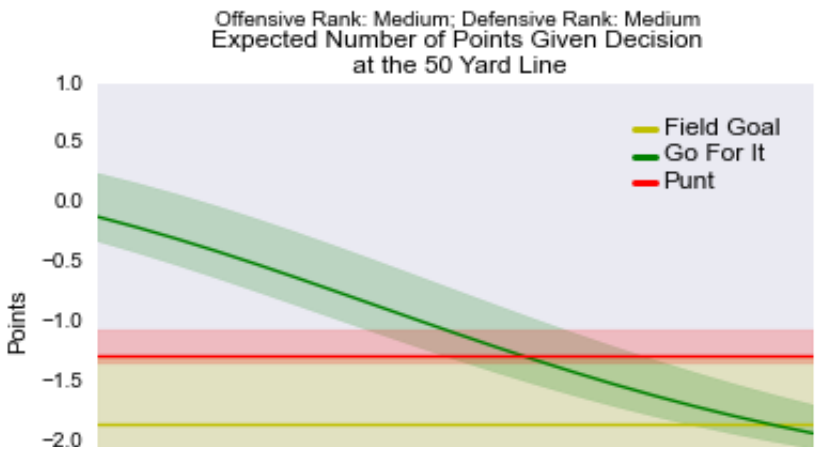
The following is the plot of the decision that should be made from the 50 yard line, where the offensive team has an mediocre ranking and the defensive team is ranked mediocre.

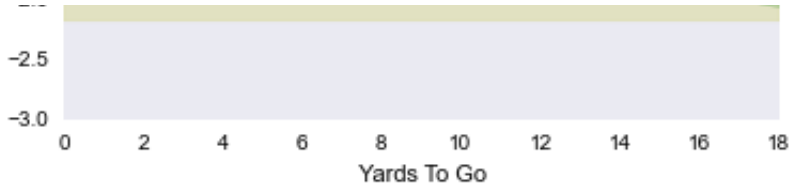
In [47]:
```
x = range(19)
```

```
d2 = [decision(50, k, 15, 16, 15, 16, "actual") for k in x]
d2 = pd.DataFrame(d2)
d2_lower = [decision(50, k, 15, 16, 15, 16, "lower") for k in x]
d2_lower = pd.DataFrame(d2_lower)
d2_upper = [decision(50, k, 15, 16, 15, 16, "upper") for k in x]
d2_upper = pd.DataFrame(d2_upper)
```

In [48]:

```
fig, ax = plt.subplots()
ax.plot(d2.index, d2['Field Goal'], color = 'y', label = "Field Goal", linest
yle = '-')
plt.fill_between(d2.index, d2_lower['Field Goal'], d2_upper['Field Goal'], co
lor='y', alpha=.2)
ax.plot(d2.index, d2['Go For It'], color = 'g', label = "Go For It", linestyl
e = '-')
plt.fill_between(d2.index, d2_lower['Go For It'], d2_upper['Go For It'], colo
r='g', alpha=.2)
ax.plot(d2.index, d2['Punt'], color = 'r', label = "Punt", linestyle = '-')
plt.fill_between(d2.index, d2_lower['Punt'], d2_upper['Punt'], color='r', alp
ha=.2)
xlabel("Yards To Go")
ylabel("Points")
ax.grid(False)
title("Expected Number of Points Given Decision \n at the 50 Yard Line")
text(2.95,1.5,"Offensive Rank: Medium; Defensive Rank: Medium")
legend = ax.legend(loc=(0.6, 0.55), fontsize = 5, frameon = False, borderpad
= 10)
xlim(xmax = 18)
ylim(ymax = 1, ymin = -3)
for label in legend.get_texts():
    label.set_fontsize('large')

for label in legend.get_lines():
    label.set_linewidth(3)

savefig('Decision5015161516.png')
plt.show()
```

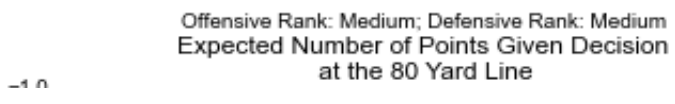


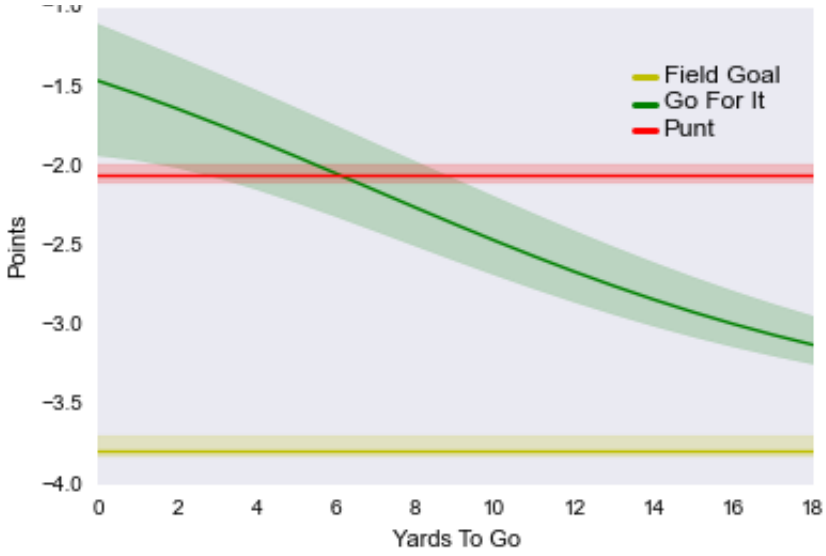


The following is the plot of the decision that should be made from the 80 yard line, where the offensive team has an mediocre ranking and the defensive team is ranked mediocre.

In [49]:
```
x = range(19)
d3 = [decision(80, k, 15, 16, 15, 16, "actual") for k in x]
d3 = pd.DataFrame(d3)
d3_lower = [decision(80, k, 15, 16, 15, 16, "lower") for k in x]
d3_lower = pd.DataFrame(d3_lower)
d3_upper = [decision(80, k, 15, 16, 15, 16, "upper") for k in x]
d3_upper = pd.DataFrame(d3_upper)
```

In [50]:
```
fig, ax = plt.subplots()
ax.plot(d3.index, d3['Field Goal'], color = 'y', label = "Field Goal", linest
yle = '-')
plt.fill_between(d3.index, d3_lower['Field Goal'], d3_upper['Field Goal'], co
lor='y', alpha=.2)
ax.plot(d3.index, d3['Go For It'], color = 'g', label = "Go For It", linestyl
e = '-')
plt.fill_between(d3.index, d3_lower['Go For It'], d3_upper['Go For It'], colo
r='g', alpha=.2)
ax.plot(d3.index, d3['Punt'], color = 'r', label = "Punt", linestyle = '-')
plt.fill_between(d3.index, d3_lower['Punt'], d3_upper['Punt'], color='r', alp
ha=.2)
xlabel("Yards To Go")
ylabel("Points")
xlim(xmax = 18)
ax.grid(False)
title("Expected Number of Points Given Decision \n at the 80 Yard Line")
text(2.95,-.6,"Offensive Rank: Medium; Defensive Rank: Medium")
legend = ax.legend(loc=(.6, .5), fontsize = 5, frameon = False, borderpad = 1
0)
for label in legend.get_texts():
    label.set_fontsize('large')

for label in legend.get_lines():
    label.set_linewidth(3)  # the legend line width

savefig('Decision8015161516.png')
plt.show()
```



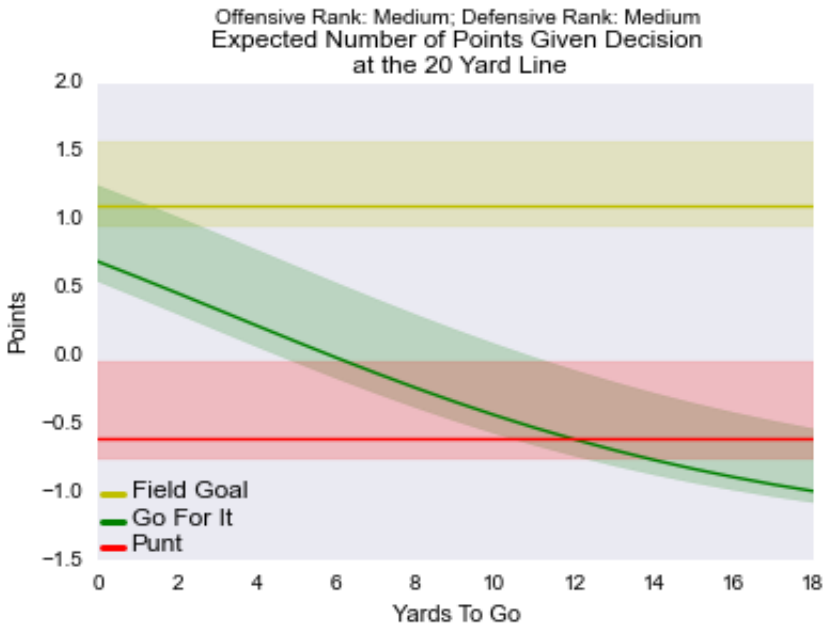


The following is the plot of the decision that should be made from the 20 yard line, where the offensive team has a mediocre ranking and the defensive team is ranked mediocre.

In [51]:
```
x = range(19)
d4 = [decision(20, k, 15, 16, 15, 16, "actual") for k in x]
d4 = pd.DataFrame(d4)
d4_lower = [decision(20, k, 15, 16, 15, 16, "lower") for k in x]
d4_lower = pd.DataFrame(d4_lower)
d4_upper = [decision(20, k, 15, 16, 15, 16, "upper") for k in x]
d4_upper = pd.DataFrame(d4_upper)
```

In [52]:
```
fig, ax = plt.subplots()
ax.plot(d4.index, d4['Field Goal'], color = 'y', label = "Field Goal", linest
yle = '-')
plt.fill_between(d4.index, d4_lower['Field Goal'], d4_upper['Field Goal'], co
lor='y', alpha=.2)
ax.plot(d4.index, d4['Go For It'], color = 'g', label = "Go For It", linestyl
e = '-')
plt.fill_between(d4.index, d4_lower['Go For It'], d4_upper['Go For It'], colo
r='g', alpha=.2)
ax.plot(d4.index, d4['Punt'], color = 'r', label = "Punt", linestyle = '-')
plt.fill_between(d4.index, d4_lower['Punt'], d4_upper['Punt'], color='r', alp
ha=.2)
xlabel("Yards To Go")
ylabel("Points")
ax.grid(False)
xlim(xmax = 18)
title("Expected Number of Points Given Decision \n at the 20 Yard Line")
text(2.95,2.45,"Offensive Rank: Medium; Defensive Rank: Medium")
legend = ax.legend(loc=(0, 0), fontsize = 5, frameon = False)
for label in legend.get_texts():
    label.set_fontsize('large')
```

```
for label in legend.get_lines():
    label.set_linewidth(3)  # the legend line width

savefig('Decision2015161516.png')
plt.show()
```



In [53]:
```
def f(x, y, z):
    return z.loc[x, y]
```

In [54]:
```
def proportion(offrank = None, defrank = None):
    """This function takes in an array of offensive rankings and defensive ra
nkings, with its default set to all rankings and creates
    a matrix of the actual decision being made currently in the NFL to plot i
n a decision map."""

    if offrank is None:
        offrank = np.array(range(1, 32))

    if defrank is None:
        defrank = np.array(range(1, 32))

    highoffrank = fourthDownPlays['offrank'].isin(offrank)
    medoffrank = fourthDownPlays['offrank'].isin(range(12, 22)).index
    pooroffrank = fourthDownPlays['offrank'].isin(range(22, 33)).index
    highdefrank = fourthDownPlays['offrank'].isin(range(1, 12)).index
    meddefrank = fourthDownPlays['offrank'].isin(range(12, 22)).index
    poordefrank = fourthDownPlays['offrank'].isin(range(22, 33)).index

    x1 = pd.DataFrame(index = range(1, 100), columns = range(1, 20))
    punt = pd.DataFrame(index = range(1, 100), columns = range(1, 20))
    go = pd.DataFrame(index = range(1, 100), columns = range(1, 20))
```

```
    field = pd.DataFrame(index = range(1, 100), columns = range(1, 20))

    x = logical_and(fourthDownPlays['offrank'].isin(offrank), fourthDownPlays
['defrank'].isin(defrank))

    for i in range(20):
        for j in range(100):

            z = logical_and(fourthDownPlays['ydline'] == j, logical_and(fourt
hDownPlays['togo'] == i, x == True))
            punt.loc[j, i] = sum(fourthDownPlays['Punt'][z])
            field.loc[j, i] = sum(fourthDownPlays['Field Goal'][z])
            go.loc[j, i] = sum(fourthDownPlays['Go For It'][z])
            x1.loc[j, i] = -1 if max(punt.loc[j, i], field.loc[j, i], go.loc[
j, i]) == 0 else 0 if max(punt.loc[j, i], field.loc[j, i], go.loc[j, i]) == g
o.loc[j, i] else 1 if max(punt.loc[j, i], field.loc[j, i], go.loc[j, i]) == p
unt.loc[j, i] else 2

    mesh1 = np.meshgrid(np.array(x1.index), np.array(x1.columns))
    Z1 = f(np.array(x1.index), np.array(x1.columns), x1)
    Z1 = np.array(Z1, dtype = float)
    Z1 = Z1.transpose()
    for i in range(20):
        for j in range(i + 1):
            Z1[i, j] = -1

    for j in range(10):
        for i in range(j):
            Z1[9 - j, 99 - i] = -1

    return Z1
```

In [55]:

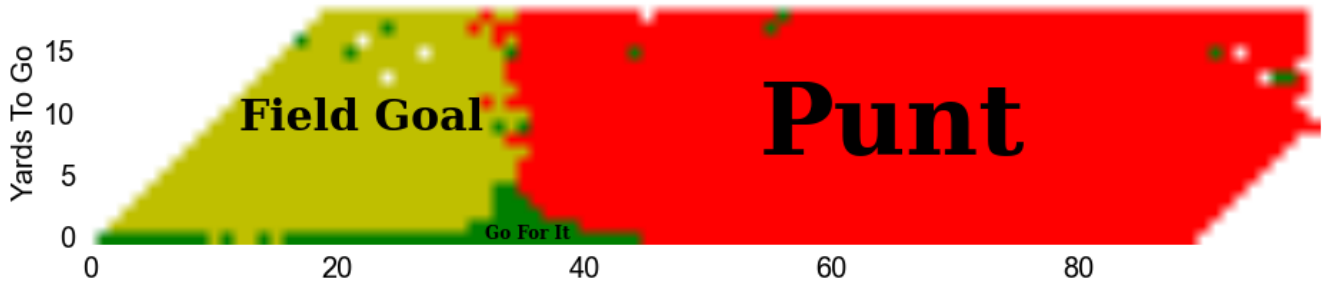
```
def optim_map(oorank, odrank, dorank, ddrank):
    """This function takes in team rankings and creates a matrix used to plot
 an optimal decision map based on the analysis."""
    yard = range(100)
    final_decision1 = pd.DataFrame(index = yard)

    for j in range(20):
        dec1 = [decision(k, j, oorank, odrank, ddrank, dorank, "actual") for
k in yard]
        final_decision1[j] = pd.DataFrame([max(g, key = g.get) for g in dec1]
)

    x1 = pd.DataFrame(index = range(100), columns = range(20))
    for i in range(20):
        for j in range(100):
```

```
            if final_decision1.loc[j,i] == 'Go For It':
                x1.loc[j, i] = 0
            elif final_decision1.loc[j,i] == 'Punt':
                x1.loc[j, i] = 1
            else:
                x1.loc[j, i] = 2

    mesh1 = np.meshgrid(np.array(x1.index), np.array(x1.columns))
    Z1 = f(np.array(x1.index), np.array(x1.columns), x1)
    Z1 = np.array(Z1, dtype = float)
    Z1 = Z1.transpose()
    for i in range(20):
        for j in range(i + 1):
            Z1[i, j] = -1

    for j in range(10):
        for i in range(j):
            Z1[9 - j, 99 - i] = -1

    return Z1
```

In [56]:
```
cmap = matplotlib.colors.ListedColormap(["white", "g", "r", "y"])
```

In [61]:
```
Z = proportion()
#Z33 = proportion(range(1,5),range(1,5))
```

In [63]:
```
plt.figure()
plt.axes([1, 1, 2, 2])
plt.imshow(Z, cmap=cmap, origin='lower')
plt.yticks(size = 20)
plt.xticks(size = 20)
plt.grid(False)
plt.ylabel("Yards To Go", size = 20)
plt.figtext(1.8,1.4, "Yard Line", family='serif', size=20)
plt.figtext(1.55,2.65, "Current NFL Decision Map", family='serif', size=25, w
eight='bold')
plt.figtext(2.1,1.92, "Punt", family='serif', size=70, color='black', weight
= 'bold')
plt.figtext(1.25,1.98, "Field Goal", family='serif', size=30, color = 'black'
, weight = 'bold')
plt.figtext(1.65,1.715, "Go For It", family='serif', size=12, color = 'black'
, weight = 'bold')

plt.show()
```

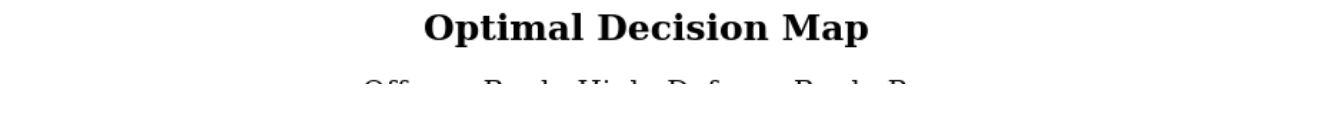


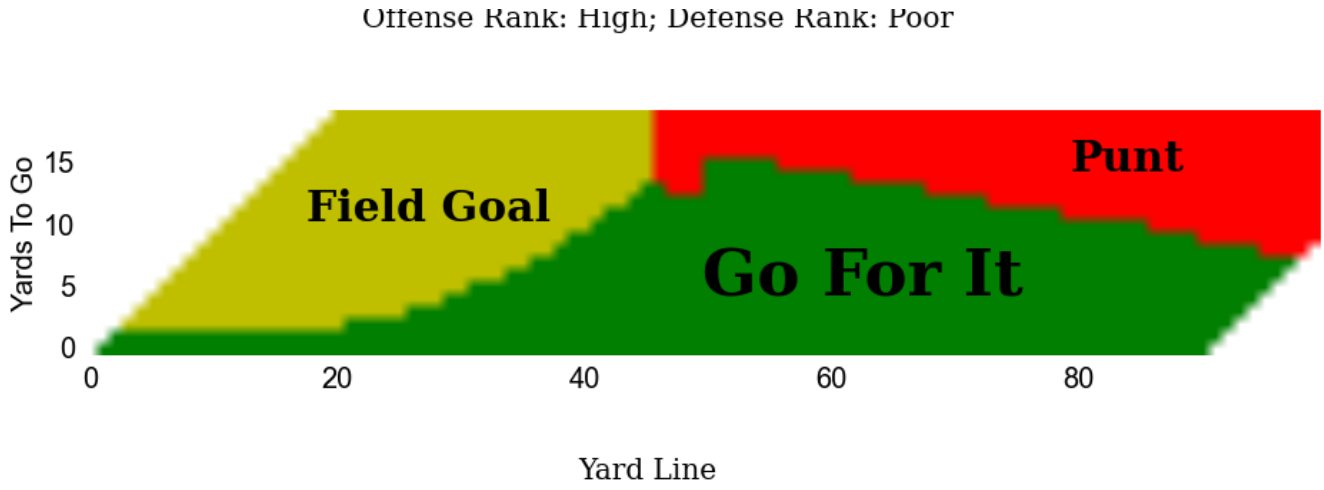
This progression is common to all offensive and defensive ranks. The 3 charts below paint the picture for the optimal decision for every position on the field and any fourth down situation less than 20 yards to go for certain rankings of the offensive and defensive units of both teams. If the team making the decision has a high ranking offense and defense, and the opposing team has a low ranking offense and defense, the optimal decision in many circumstances is to go for it. Our calculations project that it is optimal to attempt the first down anywhere on the field which is fourth and less than 2. The optimal decision map also indicates that you should go for it from about fourth and 15 from the 50, as well as fourth and 7 from your own 3. These numbers are obviously very surprising, but they are partially high because of the unusually large difference between the teams' rankings.

The following gives the decision to be made given high ranking of the offensive team and poor ranking of the defensive team.

In [57]:
```
Z1 = optim_map(1, 1, 25, 25)
```

In [58]:
```
plt.figure()
plt.axes([1, 1, 2, 2])
plt.imshow(Z1, cmap=cmap, origin='lower')
plt.yticks(size = 20)
plt.xticks(size = 20)
plt.grid(False)
plt.ylabel("Yards To Go", size = 20)
plt.figtext(1.8,1.4, "Yard Line", family='serif', size=20)
plt.figtext(1.55,2.65, "Optimal Decision Map", family='serif', size=25, weigh
t='bold')
plt.figtext(1.45,2.5, "Offense Rank: High; Defense Rank: Poor", family='serif
', size=20)
plt.figtext(2.6,2.15, "Punt", family='serif', size=30, color='black', weight
= 'bold')
plt.figtext(1.36,2.03, "Field Goal", family='serif', size=30, color = 'black'
, weight = 'bold')
plt.figtext(2,1.85, "Go For It", family='serif', size=45, color = 'black', we
ight = 'bold')
plt.show()
```





If the teams are ranked evenly for both offense and defense then our map become more conservative. Now, the map indicates that a team should attempt the first down conversion from fourth and 8 with 50 yards to go and fourth and 4 with 96 yards to go to the goal line.

The following gives the decision to be made given mediocre ranking of the offensive team and mediocre ranking of the defensive team.

In [59]:
```
Z2 = optim_map(15, 16, 16, 15)
```

In [60]:
```
plt.axes([1, 1, 2, 2])
plt.imshow(Z2, cmap= cmap, origin='lower')
yticks(size = 20)
xticks(size = 20)
grid(False)
ylabel("Yards To Go", size = 20)
figtext(1.8,1.4, "Yard Line", family='serif', size=20)
figtext(1.55,2.65, "Optimal Decision Map", family='serif', size=25, weight='b
old')
figtext(1.35,2.5, "Offense Rank: Medium; Defense Rank: Medium", family='serif
', size=20)
figtext(2.4,2.1, "Punt", family='serif', size=40, color='black', weight = 'bo
ld')
figtext(1.35,2, "Field Goal", family='serif', size=37, color = 'black', weigh
t = 'bold')
figtext(2,1.8, "Go For It", family='serif', size=35, color = 'black', weight
= 'bold')
plt.show()
```

Yard Line

The optimal decision map becomes even more conservative when the team making the decision is weak on both offense and defense, and the opposing team is strong offensively and defensively. Now, the decision map's optimal values are to go for the first down on fourth and four from the 50, and go for it on fourth and one with 91 yards to go to the goal line.

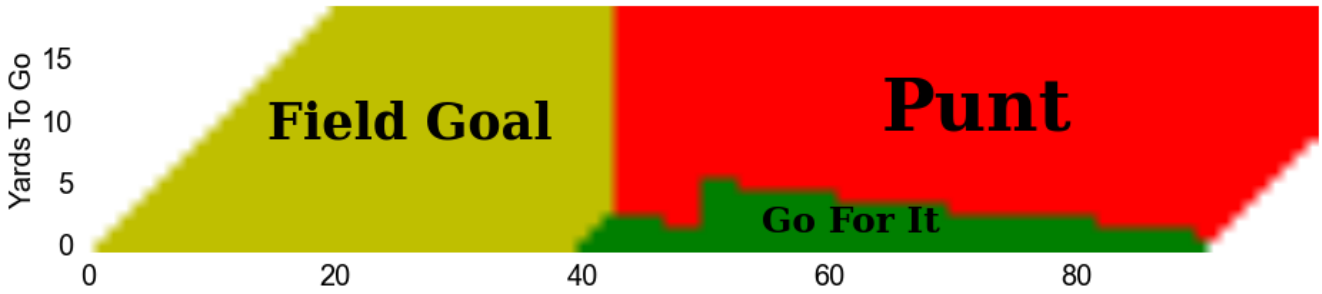
The following gives the decision to be made given poor ranking of the offensive team and high ranking of the defensive team.

In [61]:
```
Z3 = optim_map(25, 25, 2, 2)
```

In [62]:
```
plt.axes([1, 1, 2, 2])
plt.imshow(Z3, cmap= cmap, origin='lower')
yticks(size = 20)
xticks(size = 20)
grid(False)
ylabel("Yards To Go", size = 20)
figtext(1.8,1.4, "Yard Line", family='serif', size=20)
figtext(1.55,2.65, "Optimal Decision Map", family='serif', size=25, weight='b
old')
figtext(1.42,2.5, "Offense Rank: Poor; Defense Rank: High", family='serif', s
ize=20)
figtext(2.3,2, "Punt", family='serif', size=50, color='black', weight = 'bold
')
figtext(1.3,1.98, "Field Goal", family='serif', size=35, color = 'black', wei
ght = 'bold')
figtext(2.1,1.75, "Go For It", family='serif', size=25, color = 'black', weig
ht = 'bold')
plt.show()
```

Optimal Decision Map

Offense Rank: Poor; Defense Rank: High



Yard Line

Obviously, the ranking of both teams' offense and defense have an enormous impact on the optimal decision. For a high ranking offense, a coach should go for it much more often given other rankings remain constant. The dramatic changes in these plots purely due to teams' offensive and defensive ranks are interesting, but how does the overall trend compare to current NFL fourth down decision making may be a better question? The plot below shows the decisions coaches and upper management have made over these 11 seasons. When comparing the maps, there are stark differences from our previous three optimal decision maps. The first major difference is that teams rarely go for it. In fact anywhere past the 45 yard line the decision most often made in every scenario (combination of yard line and yards to go) is to punt. One other noticeable feature is that our earlier plots had kickers going for a field goal attempt from much larger distances. Once again, this value is heavily inflated because of the higher than should be conversion rate of deep attempts.
Our analysis should not convince you that the results we obtained are perfect. The data used in our analysis is observational, and there is no way to prove causality. Therefore, coaches' deep level of knowledge of the game should still play a major role in determining the fourth down decision. But, their football expertise along with an understanding of the estimated expected values could help guide their decision under uncertainty. The stark difference between our logical model and the current NFL's usual decision is obvious. We still would not recommend going for a first down on fourth and seven from your own 3 yard line. However, our process makes intuitive and mathematical sense. And the optimal decision is most likely somewhere in the middle. Obviously, giving up possession of the ball by punting has a much greater negative effect than teams currently realize.

The model could be improved, and the assumptions fortified with the help of other statistician gurus. But, this is a good start in showing that something is off. The way current NFL coaches and upper management evaluate fourth down decisions is seriously flawed. In the next decade, statistics will probably play a more integral role in fourth down decision making because the first few teams that makes sense of the data will have a distinct advantage.