Kevin Mathews
CSC461
Fall 2018

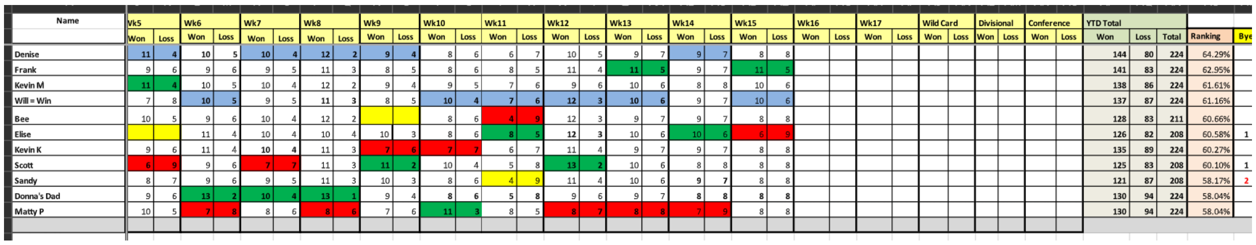
American Football Prediction Model

Introduction:

With the recent legalization of sports-betting in Rhode Island, the ability to predict the outcome of games in the NHL, NBA, NFL, or World Series begins to become a more lucrative opportunity for domain experts. According to an article from the Boston Globe, over 100 billion dollars is currently in play in the illegal betting market [1]. Although betting is currently restricted to casinos only, the ability to bet online or via smartphone may eventually allow widespread regular play across the state.

[1] https://www.bostonglobe.com/business/2018/11/11/legal-sports-betting-will-begin-casinos-later-this-month/YQqckMi6OwDBcqcpLhcCvI/story.html

While I personally don't gamble, I do participate in a fantasy football league at work with several co-workers. The purpose of this project is to try and train an algorithm which can predict the outcomes of NFL games better than my baseline performance so far (61.61% correct).



Name	Vk5		Wk6		Wk7		Wk8		Wk9		Wk10		Wk11	
	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss
Denise	11	4	10	5	10	4	12	2	9	4	8	6	6	7
Frank	9	6	9	6	9	5	11	3	8	5	8	6	8	5
Kevin M	11	4	10	5	10	4	12	2	9	4	9	5	7	6
Will = Win	7	8	10	5	9	5	11	3	8	5	10	4	7	6
Bee	10	5	9	6	10	4	12	2			8	6	4	9
Elise			11	4	10	4	10	4	10	3	8	6	8	5
Kevin K	9	6	11	4	10	4	11	3	7	6	7	7	6	7
Scott	6	9	9	6	7	7	11	3	11	2	10	4	5	8
Sandy	8	7	9	6	9	5	11	3	10	3	8	6	4	9
Donna's Dad	9	6	13	2	10	4	13	1	9	4	8	6	5	8
Matty P	10	5	7	8	8	6	8	6	7	6	11	3	8	5

Name	Wk12		Wk13		Wk14		Wk15		Wk16		Wk17		Wild Card		Divisional		Conference		YTD Total				
	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Won	Loss	Total	Ranking	Bye
Denise	10	5	9	7	9	7	8	8											144	80	224	64.29%	
Frank	11	4	11	5	9	7	11	5											141	83	224	62.95%	
Kevin M	9	6	10	6	8	8	10	6											138	86	224	61.61%	
Will = Win	12	3	10	6	9	7	10	6											137	87	224	61.16%	
Bee	12	3	9	7	9	7	8	8											128	83	211	60.66%	
Elise	12	3	10	6	10	6	6	9											126	82	208	60.58%	1
Kevin K	11	4	9	7	9	7	8	8											135	89	224	60.27%	
Scott	13	2	10	6	8	8	8	8											125	83	208	60.10%	1
Sandy	11	4	10	6	9	7	8	8											121	87	208	58.17%	2
Donna's Dad	9	6	9	7	8	8	8	8											130	94	224	58.04%	
Matty P	8	7	8	8	7	9	8	8											130	94	224	58.04%	

Methodology:

The following Kaggle Dataset was used for training.

https://www.kaggle.com/kendallgillies/nflstatistics

The webpage includes multiple files on NFL performance. The data available is as follows.

<u>Basic Stats File:</u> This file includes data on individual players. Stats include Height, Weight, College, Experience (in seasons), and other items specific to each player. 17172 Players are included.

<u>Career Stats Files:</u> These files focus on the career performance of individual players. Included are specific items on each player's performance. Number of Assisted Tackles, Number of Touchdown Passes, Number of Returns Longer than 40 Yards, and other features comprise this data.

<u>Game Logs Files:</u> Files on player specific performance during individual games. The files are organized based on team role. Game date, season, scores, and opposing team names are listed. Features include, Net Punting Yards per game, Receptions per game, and more.

During the normal Football season, games are organized into weeks. Each week normally has games on Thursdays, Sundays, and Mondays, in that order.

When it's time for me to pick scores before the Thursday game, the only information I will have available will be the teams playing, time information (game date/week #/season), & home/away status.

The problem is phrased in terms of Binary Classification (Win or Lose). Information on ties does exist in the data, but after the feature engineering process, only 14 tied games out of 19249 were left. Tie prediction was dropped to keep the problem simple.

Feature Engineering Process:
In order to use the Kaggle datasets for prediction, I needed to combine them into a form which a predictive algorithm could accept (labels followed vectors of numbers). All the code used to accomplish this is included with the source code.

Each datapoint in the Kaggle datasets corresponds to a specific player. To use this data for my problem, the training set needed to be structured such that each row describes a single game. The information needed to do this is split between the Game Logs and Career Stats files. Using Python's Pandas module and Excel, each player was connected to a particular team (via Career Stats), and both teams for each game were determined by combining the previous information with the Game Logs files. The abbreviations for each opponent were replaced with the full team names in the Game Logs.

This process was not perfect, as some players moved between teams during specific years. The method connecting players to teams led to some instances where teams played themselves during some games. These games were dropped.

There are currently only 32 teams in the NFL. Since this data includes information back to the 1970s, several teams no longer exist. Games where these teams played were dropped. This data also lists the Los Angeles Chargers as the San Diego Chargers, which was corrected via Python. A feature which contains the Win/Lose label for Team_1 was also generated for each datapoint.

After initially generating 37779 possible games, we are left with 19235 games to train with.

In order to capture some of the statistical data, the game-specific averages for multiple stats corresponding to each team were also added. Only data from the Game Logs was used for this. The Game Logs were filtered to each team, and averages for each feature were collected using excel. This resulted in a table of 32 rows (one for each team) filled with averages. The statistical game data for Team_1 in the training data was added (using VLOOKUP) next to each game instance. An extra feature, Team_Score, was added using the method above to capture the average score for all time for a particular team. All averages were calculated using the entire time range available (1970s-2016).

Model Selection/Training:
The code used for testing is included in the source code section.

Multiple models were experimented with and 5 Fold Stratified Cross Validation was attempted on each to gauge performance. In addition, data for the 2018 NFL season was also collected, and used as a final test set.

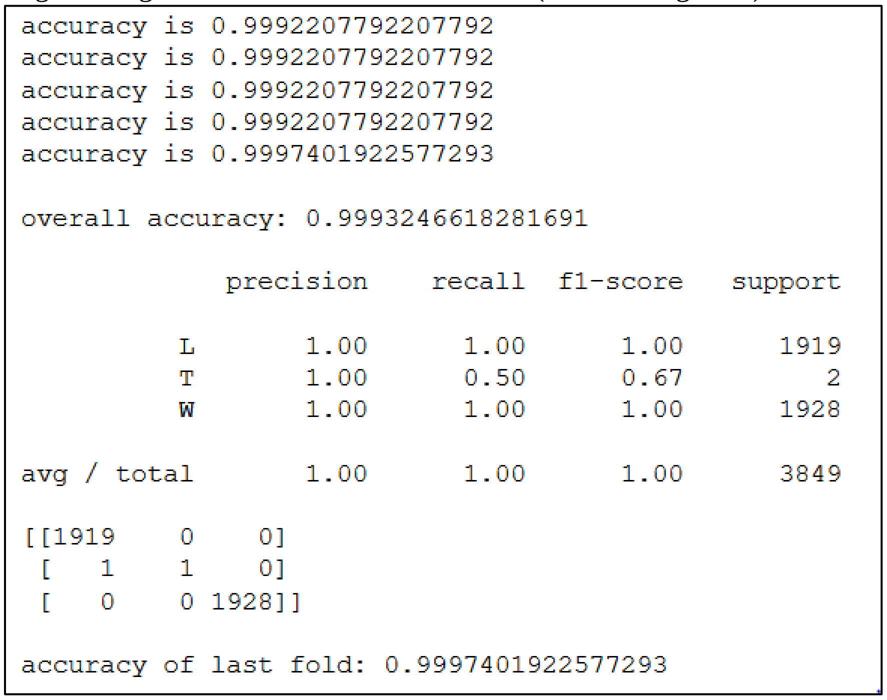
The following SKLearn algorithms for classification were tested.

- Logistic Regression
- Regression Trees
- Support Vector Machines
- Naïve Bayes

Results & Analysis:

Initial testing resulted in clear overfitting of the data (Logistic Regression), and high performance from the other algorithms. (Ties were only dropped after initial testing)

Logistic Regression 5-Fold Cross Validation (All Training Data)



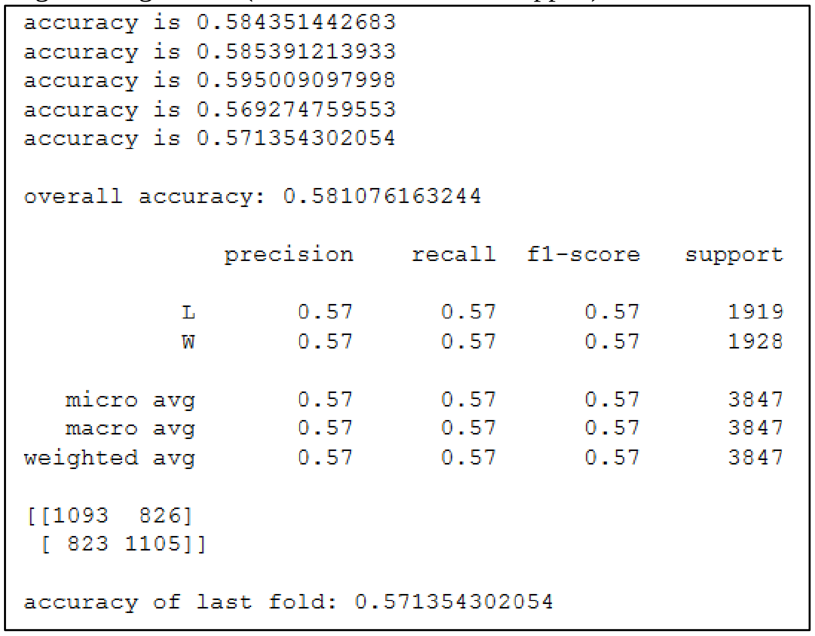
```
accuracy is 0.9992207792207792
accuracy is 0.9992207792207792
accuracy is 0.9992207792207792
accuracy is 0.9992207792207792
accuracy is 0.9997401922577293

overall accuracy: 0.9993246618281691

             precision    recall  f1-score   support

          L       1.00      1.00      1.00      1919
          T       1.00      0.50      0.67         2
          W       1.00      1.00      1.00      1928

avg / total       1.00      1.00      1.00      3849

[[1919    0    0]
 [   1    1    0]
 [   0    0 1928]]

accuracy of last fold: 0.9997401922577293
```

Gaussian Naïve Bayes: 73% Accuracy
Decision Tree: 98% Accuracy

The confusion matrix and classification report correspond to the last fold.

This was likely due to the fact that the actual scores for each game were initially included in the dataset. This makes it too easy to determine the winner and loser of a particular game (and I wouldn't have this information anyway for new predictions…). Scores were replaced by average score by team/opponent matchup. The average parameters (cols F-EC) are also calculated using the entire training set. This may give the algorithm an indirect view into what the hidden fold looks like during Cross Validation. This extra data was therefore dropped, and cross validation was attempted again. The algorithms tested appeared to all have equal accuracy using the default hyperparameters. (Ties dropped)

Logistic Regression (Scores & cols F-EC dropped)



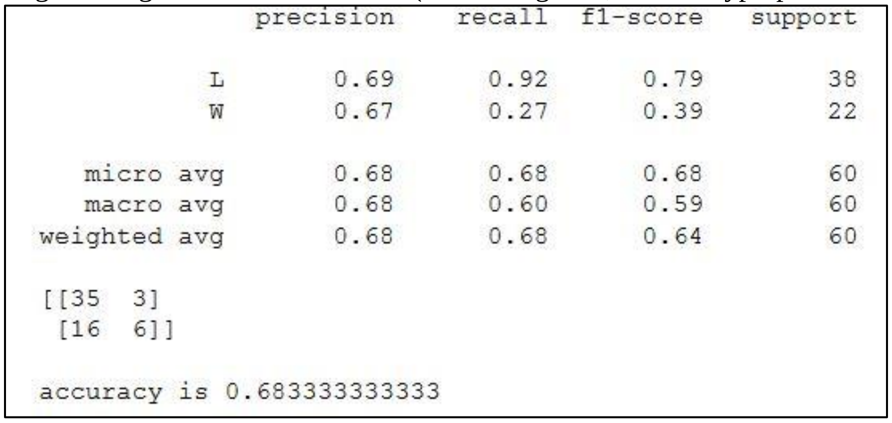
```
accuracy is 0.584351442683
accuracy is 0.585391213933
accuracy is 0.595009097998
accuracy is 0.569274759553
accuracy is 0.571354302054

overall accuracy: 0.581076163244

             precision    recall  f1-score   support

          L       0.57      0.57      0.57      1919
          W       0.57      0.57      0.57      1928

  micro avg       0.57      0.57      0.57      3847
  macro avg       0.57      0.57      0.57      3847
weighted avg      0.57      0.57      0.57      3847

[[1093  826]
 [ 823 1105]]

accuracy of last fold: 0.571354302054
```

Gaussian Naïve Bayes: 57% Accuracy
Decision Trees: 49% Accuracy – Mostly Wins
Support Vector Machine: 49% Accuracy – Mostly Losses

In order to perform true Cross Validation on the entire training set, I would need to recalculate all of the averages using only the training folds during each iteration. This requires more code to accomplish and would be an area for improvement. The ability to do proper cross validation would make the process of choosing hyperparameters much more effective.

Logistic Regression appeared to be a natural choice to concentrate on for improvement. The performance was generally high, and this was a binary classification problem. I tested the Logistic Regression using data from games during 2018. Since these games were never used for training at any point during the project, I was able to confidently use the set of team averages I generated for training. Using the default hyperparameters for Logistic Regression, 68% accuracy was achieved when predicting September 2018.

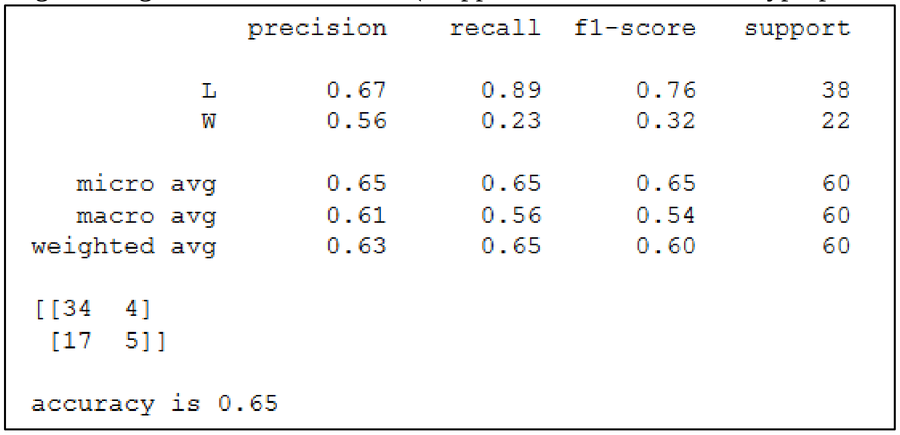
Logistic Regression 2018 Test Set (all training data, default hyperparameters):



```
             precision    recall  f1-score   support

          L       0.69      0.92      0.79        38
          W       0.67      0.27      0.39        22

  micro avg       0.68      0.68      0.68        60
  macro avg       0.68      0.60      0.59        60
weighted avg      0.68      0.68      0.64        60

[[35  3]
 [16  6]]

accuracy is 0.683333333333
```

Because columns G-EC are the same for each of the 32 possible values of Team_1, they really don't add much value in their current form. Columns G-EC were dropped and the performance was measured.

Logistic Regression 2018 Test Set (dropped cols G-EC, default hyperparameters):



```
              precision    recall  f1-score   support

           L       0.67      0.89      0.76        38
           W       0.56      0.23      0.32        22

   micro avg       0.65      0.65      0.65        60
   macro avg       0.61      0.56      0.54        60
weighted avg       0.63      0.65      0.60        60

[[34  4]
 [17  5]]

accuracy is 0.65
```

Final Remarks:
According to the final results, the potential to beat my personal predictions does appear possible. There is currently a lot of room for improvement. The dataset could be better constructed in order to better capture the differences between each team. Instead of averaging each stat over the entirety of time, I could try averaging over the past year before a particular game, or I could try adding more features for particular players instead of for the entire team. The Basic Stats files was also not used during the project, which could possibly improve prediction. In order to properly tune hyperparameters, the averaging would have to be dynamic, or the data for particular games would need be unique.

Appendix Source Code:
Source code below.

In [1]:

```
# Kevin Mathews
# Final Project CSC 461
# Fall 2018

# The following notebook contains the code used for training and testing a football prediction alg
orithm
# A clean dataset is required

# Datasets Location
# https://www.kaggle.com/kendallgillies/nflstatistics
```

In [7]:

```
# Importing the libraries
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

In [8]:

```
# Importing the training dataset
dataset = pd.read_csv('training.csv')
dataset.head()
```

Out[8]:

	Outcome Team_1	Game Date	Team_1	Team_2	Home or Away Team_1	Score Ratio	Rushing Attempts	Rushing Attempts Per Game	Rushing Yards	Yards Per Carry	...	Solo Tackles	Assiste Tackle
0	L	25571	28	9	0	1.089779	41.153285	2.195378	175.676399	3.920488	...	18.102828	5.13624
1	L	25571	25	3	1	0.755784	50.574713	2.683721	215.417625	3.988417	...	18.226601	4.85468
2	L	25585	9	3	0	1.180412	41.723270	1.969853	174.355346	3.946541	...	19.051282	6.13053
3	L	25830	23	6	0	1.043038	50.827160	2.886711	190.991770	3.472632	...	18.726131	6.07286
4	W	25830	6	23	1	1.010959	49.049908	2.843561	201.369686	3.452000	...	18.428916	4.88915

5 rows × 133 columns

In [9]:

```
# Importing the testing dataset
test_set = pd.read_csv('test.csv')
test_set.head()
```

Out[9]:

	Label	Game Date	Team_1	Team_2	Home or Away Team_1	Score_Ratio	Rushing Attempts	Rushing Attempts Per Game	Rushing Yards	Yards Per Carry	...	Solo Tackles	Assiste Tackle
0	L	43349	2	26	1	0.856908	49.075269	2.525287	201.157706	3.339928	...	17.678133	4.98525
1	W	43352	7	14	1	0.959565	44.440000	2.282759	182.633333	4.007718	...	17.386667	8.05333
2	L	43352	31	19	1	1.000000	51.787879	2.110000	212.747475	4.320408	...	19.513158	6.79736
3	L	43352	28	20	1	1.187029	41.153285	2.195378	175.676399	3.920488	...	18.102828	5.13624
4	L	43352	13	21	1	0.528814	48.102041	2.726923	198.489796	3.680612	...	17.570776	6.04794

5 rows × 133 columns

In [27]:

```
# Choose Classifier

# LogisticRegression
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()

# Naive Bayes
#from sklearn.naive_bayes import GaussianNB
#classifier = GaussianNB()

# Decision Tree
#from sklearn.tree import DecisionTreeClassifier
#classifier = DecisionTreeClassifier()

# Support Vector Machine
#from sklearn.svm import SVC
#classifier = SVC()
```

In [28]:

```
# Import Training Set for Cross Validation

import scipy.stats
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold

# Importing metrics for evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Seperating the data into dependent and independent variables
#X = dataset.iloc[:, 1:].values
#y = dataset.iloc[:, 0].values

# Extra Statistical Data Dropped
X = dataset.iloc[:, 1:5].values
y = dataset.iloc[:, 0].values
```

In [29]:

```
# In order to perform true cross-validation on this dataset, I would need to recalculate the avera
ge team-stats (cols F-EC) for each fold tested.
# Otherwise the algorithm has an indirect view into the test fold.
# Cross Validation Cell

skf = StratifiedKFold(n_splits=5, random_state=None)

#print('data shape:',X_train.shape)

accuracies = []
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    print('accuracy is',accuracy_score(y_pred,y_test))
    #print('shape is',X_test.shape)
    accuracies.append(accuracy_score(y_pred,y_test))

accuracies = np.array(accuracies)
print('\noverall accuracy:',accuracies.mean(),'\n')

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
```

```
print('\naccuracy of last fold:',accuracy_score(y_pred,y_test))
#print('\naccuracy is '+ accuracy_score(y_pred,y_test) + ' test_shape: ' + str(y_test.shape))
```

```
accuracy is 0.584351442683
accuracy is 0.585391213933
accuracy is 0.595009097998
accuracy is 0.569274759553
accuracy is 0.571354302054

overall accuracy: 0.581076163244

              precision    recall  f1-score   support

           L       0.57      0.57      0.57      1919
           W       0.57      0.57      0.57      1928

   micro avg       0.57      0.57      0.57      3847
   macro avg       0.57      0.57      0.57      3847
weighted avg       0.57      0.57      0.57      3847

[[1093  826]
 [ 823 1105]]

accuracy of last fold: 0.571354302054
```

In [30]:

```
# Importing Data for Testing
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Seperating the data into dependent and independent variables
X_train = dataset.iloc[:, 1:].values
y_train = dataset.iloc[:, 0].values
X_test = test_set.iloc[:, 1:].values
y_test = test_set.iloc[:, 0].values

# Train/Test without Game Logs parameters
#X_train = dataset.iloc[:, 1:6].values
#y_train = dataset.iloc[:, 0].values
#X_test = test_set.iloc[:, 1:6].values
#y_test = test_set.iloc[:, 0].values
```

In [20]:

```
#test_set.to_clipboard(sep='\t',index=None)
```

In [31]:

```
# Testing on hand-entered data from 2018, included nowhere in the training dataset from Kaggle.

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
from sklearn.metrics import accuracy_score
print('\naccuracy is',accuracy_score(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           L       0.69      0.92      0.79        38
           W       0.67      0.27      0.39        22

   micro avg       0.68      0.68      0.68        60
   macro avg       0.68      0.60      0.59        60
weighted avg       0.68      0.68      0.64        60

[[35  3]
```

```
 [16  6]]

accuracy is 0.683333333333
```

In [1]:

```
# Kevin Mathews
# Final Project CSC 461
# Fall 2018

# The following notebook contains the code used for converting the following football datasets
# from Kaggle into a dataset usable by the algorithms from sklearn.

# Datasets Location
# https://www.kaggle.com/kendallgillies/nflstatistics
```

In [ ]:

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

In [2]:

```
#Next three cells load all the data from Kaggle
basic_stats = pd.read_csv('Basic_Stats.csv')
```

In [3]:

```
career_stats_defensive = pd.read_csv('Career_Stats_Defensive.csv')
career_stats_field_goal_kickers = pd.read_csv('Career_Stats_Field_Goal_Kickers.csv')
career_stats_fumbles = pd.read_csv('Career_Stats_Fumbles.csv')
career_stats_kick_return = pd.read_csv('Career_Stats_Kick_Return.csv')
career_stats_kickoff = pd.read_csv('Career_Stats_Kickoff.csv')
career_stats_offensive_line = pd.read_csv('Career_Stats_Offensive_Line.csv')
career_stats_passing = pd.read_csv('Career_Stats_Passing.csv')
career_stats_punt_return = pd.read_csv('Career_Stats_Punt_Return.csv')
career_stats_punting = pd.read_csv('Career_Stats_Punting.csv')
career_stats_receiving = pd.read_csv('Career_Stats_Receiving.csv')
career_stats_rushing = pd.read_csv('Career_Stats_Rushing.csv')
```

In [4]:

```
game_logs_defensive_lineman = pd.read_csv('Game_Logs_Defensive_Lineman.csv')
game_logs_kickers = pd.read_csv('Game_Logs_Kickers.csv')
game_logs_offensive_line = pd.read_csv('Game_Logs_Offensive_Line.csv')
game_logs_punters = pd.read_csv('Game_Logs_Punters.csv')
game_logs_quarter = pd.read_csv('Game_Logs_Quarterback.csv')
game_logs_runningback = pd.read_csv('Game_Logs_Runningback.csv')
game_logs_wide_reciever_and_tight_end = pd.read_csv('Game_Logs_Wide_Receiver_and_Tight_End.csv')
```

In [5]:

```
career_stats_defensive.columns.values
```

Out[5]:

```
array(['Player Id', 'Name', 'Position', 'Year', 'Team', 'Games Played',
       'Total Tackles', 'Solo Tackles', 'Assisted Tackles', 'Sacks',
       'Safties', 'Passes Defended', 'Ints', 'Ints for TDs', 'Int Yards',
       'Yards Per Int', 'Longest Int Return'], dtype=object)
```

In [6]:

```
game_logs_defensive_lineman.columns.values
```

Out[6]:

```
array(['Player Id', 'Name', 'Position', 'Year', 'Season', 'Week',
       'Game Date', 'Home or Away', 'Opponent', 'Outcome', 'Score',
       'Games Played', 'Games Started', 'Total Tackles', 'Solo Tackles',
       'Assisted Tackles', 'Sacks', 'Safties', 'Passes Defended', 'Ints',
       'Int Yards', 'Yards Per Int', 'Longest Int Return', 'Ints for TDs',
       'Forced Fumbles'], dtype=object)
```

In [7]:

```
#The team names for each player are in the career stats files. We will use these to find the missi
ng teams in the game logs
#Preparation for connecting player ID to team
selections_career_stats = ['Player Id','Year','Team']
cs_defensive = career_stats_defensive[selections_career_stats]
cs_field_goal_kickers = career_stats_field_goal_kickers[selections_career_stats]
cs_fumbles = career_stats_fumbles[selections_career_stats]
cs_kick_return = career_stats_kick_return[selections_career_stats]
cs_kickoff = career_stats_kickoff[selections_career_stats]
cs_offensive_line = career_stats_offensive_line[selections_career_stats]
cs_passing = career_stats_passing[selections_career_stats]
cs_punt_return = career_stats_punt_return[selections_career_stats]
cs_punting = career_stats_punting[selections_career_stats]
cs_receiving = career_stats_receiving[selections_career_stats]
cs_rushing = career_stats_rushing[selections_career_stats]
```

In [8]:

```
# The game logs contain only the abbreviated name of the opponent
selections_game_logs = ['Player Id','Year','Season','Game Date','Week','Home or Away','Opponent','O
utcome','Score']
gl_defensive_lineman = game_logs_defensive_lineman[selections_game_logs]
gl_kickers = game_logs_kickers[selections_game_logs]
gl_offensive_line = game_logs_offensive_line[selections_game_logs]
gl_punters = game_logs_punters[selections_game_logs]
gl_quarter = game_logs_quarter[selections_game_logs]
gl_runningback = game_logs_runningback[selections_game_logs]
gl_wide_reciever_and_tight_end = game_logs_wide_reciever_and_tight_end[selections_game_logs]
```

In [9]:

```
#combine all datasets into master sets
frames_career_stats = [cs_defensive, cs_field_goal_kickers, cs_fumbles, cs_kick_return, cs_kickoff,
cs_offensive_line, cs_offensive_line, cs_passing, cs_punt_return, cs_punting, cs_receiving,
cs_rushing]
frames_game_logs =
[gl_defensive_lineman,gl_kickers,gl_offensive_line,gl_punters,gl_quarter,gl_runningback,gl_wide_rec
iever_and_tight_end]
```

In [10]:

```
result_career_stats = pd.concat(frames_career_stats)
result_career_stats = result_career_stats.drop_duplicates()
result_career_stats = result_career_stats.sort_values(by=['Player Id','Year'])
```

In [11]:

```
result_game_logs = pd.concat(frames_game_logs)
result_game_logs = result_game_logs.drop_duplicates()
result_game_logs = result_game_logs.sort_values(by=['Player Id','Year'])
```

In [12]:

```
# Career Stats Important data
print(result_career_stats.shape)
result_career_stats.head()
```

```
(54846, 3)
```

Out[12]:

	Player Id	Year	Team
2523	'omarellison/2500540	1995	San Diego Chargers

	Player Id	Year	Team
2522	omarellison/2500540	1996	San Diego Chargers
10315	a'shawnrobinson/2555265	2016	Detroit Lions
6013	a.c.bauer/2509176	1923	Racine Legion
13470	a.j.bouye/2541162	2013	Houston Texans

In [13]:

```
# Game Logs important data
print(result_game_logs.shape)
result_game_logs.head()
```

```
(697918, 9)
```

Out[13]:

	Player Id	Year	Season	Game Date	Week	Home or Away	Opponent	Outcome	Score
15986	'omarellison/2500540	1995	Regular Season	09/03	1	Away	OAK	L	7 to 17
15987	'omarellison/2500540	1995	Regular Season	09/10	2	Home	SEA	W	14 to 10
15988	'omarellison/2500540	1995	Regular Season	09/17	3	Away	PHI	W	27 to 21
15989	'omarellison/2500540	1995	Regular Season	09/24	4	Home	DEN	W	17 to 6
15990	'omarellison/2500540	1995	Regular Season	10/01	5	Away	PIT	L	16 to 31

In [14]:

```
# Combine to get them names of both teams for each game into one dataset
data_full = pd.merge(result_game_logs, result_career_stats)
print(data_full.shape)
data_full.head()
```

```
(664325, 10)
```

Out[14]:

	Player Id	Year	Season	Game Date	Week	Home or Away	Opponent	Outcome	Score	Team
0	'omarellison/2500540	1995	Regular Season	09/03	1	Away	OAK	L	7 to 17	San Diego Chargers
1	'omarellison/2500540	1995	Regular Season	09/10	2	Home	SEA	W	14 to 10	San Diego Chargers
2	'omarellison/2500540	1995	Regular Season	09/17	3	Away	PHI	W	27 to 21	San Diego Chargers
3	'omarellison/2500540	1995	Regular Season	09/24	4	Home	DEN	W	17 to 6	San Diego Chargers
4	'omarellison/2500540	1995	Regular Season	10/01	5	Away	PIT	L	16 to 31	San Diego Chargers

In [15]:

```
#data_full[data_full['Year']>2000]['Opponent'].drop_duplicates().to_clipboard(sep='\t',
index=None)
career_stats_field_goal_kickers.columns.values
```

Out[15]:

```
array(['Player Id', 'Name', 'Position', 'Year', 'Team', 'Games Played',
       'Kicks Blocked', 'Longest FG Made', 'FGs Made', 'FGs Attempted',
       'FG Percentage', 'FGs Made 20-29 Yards',
       'FGs Attempted 20-29 Yards', 'FG Percentage 20-29 Yards',
       'FGs Made 30-39 Yards', 'FGs Attempted 30-39 Yards',
       'FG Percentage 30-39 Yards', 'FGs Made 40-49 Yards',
       'FGs Attempted 40-49 Yards', 'FG Percentage 40-49 Yards',
       'FGs Made 50+ Yards', 'FGs Attempted 50+ Yards',
       'FG Percentage 50+ Yards', 'Extra Points Attempted',
       'Extra Points Made', 'Percentage of Extra Points Made',
       'Extra Points Blocked'], dtype=object)
```

In [16]:

```
# Function to replace opponent abbreviations from game logs datasets
def Opponent_Name(row):
    Opponent_Col = row['Opponent']
    Opponent_Out = "-"
    TeamDict = {'ARI':'Arizona Cardinals',
                'ATL':'Atlanta Falcons',
                'BAL':'Baltimore Ravens',
                'BUF':'Buffalo Bills',
                'CAR':'Carolina Panthers',
                'CHI':'Chicago Bears',
                'CIN':'Cincinnati Bengals',
                'CLE':'Cleveland Browns',
                'DAL':'Dallas Cowboys',
                'DEN':'Denver Broncos',
                'DET':'Detroit Lions',
                'GB':'Green Bay Packers',
                'HOU':'Houston Texans',
                'IND':'Indianapolis Colts',
                'JAX':'Jacksonville Jaguars',
                'KC':'Kansas City Chiefs',
                'LA':'Los Angeles Rams',
                'MIA':'Miami Dolphins',
                'MIN':'Minnesota Vikings',
                'NE':'New England Patriots',
                'NO':'New Orleans Saints',
                'NYG':'New York Giants',
                'NYJ':'New York Jets',
                'OAK':'Oakland Raiders',
                'PHI':'Philadelphia Eagles',
                'PIT':'Pittsburgh Steelers',
                'SAN':'San Francisco 49ers',
                'SEA':'Seattle Seahawks',
                'TB':'Tampa Bay Buccaneers',
                'TEN':'Tennessee Titans',
                'WAS':'Washington Redskins'}

    for key in TeamDict: #Main Loop
        if key in Opponent_Col:
            Opponent_Out = TeamDict.get(key)
            break
        else:
            continue

    return Opponent_Out
```

In [17]:

```
# function to check if a team is playing itself in the data
def equal_teams(row):
    Team_1_Col = row['Team_1']
    Team_2_Col = row['Team_2']

    if Team_1_Col == Team_2_Col:
        output = "Bad"
    elif Team_2_Col == '-':
        output = "Bad"
    else:
        output = "Good"

    return output
```

In [18]:

```
# Functions to assign scores to winners and losers
def score_assort_1(row):
    score = row['Score']
    team_1_outcome = row['Outcome Team_1']
    scores_list = [int(s) for s in score.split() if s.isdigit()]

    win_score = max(scores_list[0], scores_list[1])
    lose_score = min(scores_list[0], scores_list[1])

    if team_1_outcome == 'W':
        output = win_score
    elif team_1_outcome == 'L':
        output = lose_score
    else:
        output = win_score

    return output

def score_assort_2(row):
    score = row['Score']
    team_1_outcome = row['Outcome Team_1']
    scores_list = [int(s) for s in score.split() if s.isdigit()]

    win_score = max(scores_list[0], scores_list[1])
    lose_score = min(scores_list[0], scores_list[1])

    if team_1_outcome == 'W':
        output = lose_score
    elif team_1_outcome == 'L':
        output = win_score
    else:
        output = lose_score

    return output
```

In [19]:

```
# apply Opponent_Name functio to team 2 abbreviations
data_full['Team_2'] = data_full.apply(Opponent_Name, axis=1)
data_full = data_full.rename(columns = {'Team':'Team_1',
                                        "Home or Away": "Home or Away Team_1",
                                        "Outcome":"Outcome Team_1"}) # Rename Columns
data_full.Team_1 = data_full.Team_1.replace({"San Diego Chargers": "Los Angeles Chargers"}) # Add
correct city for Chargers
```

In [20]:

```
# Find errors in team 1 assignment method (by player), add scores for each team
data_full['Error'] = data_full.apply(equal_teams, axis=1)
data_full['Score Team_1'] = data_full.apply(score_assort_1, axis=1)
data_full['Score Team_2'] = data_full.apply(score_assort_2, axis=1)
```

In [22]:

```
# sort to basic data only, averages were then collected using VLOOKUP and AVERAGEIF in excel,
# due to strings in data columns from career stats.
# Year & Game Date combined into one column using excel
out = data_full[data_full['Error']=='Good'][['Year','Game Date','Team_1','Team_2','Home or Away
Team_1','Outcome Team_1','Score Team_1','Score Team_2','Error']]
out = out.sort_values(by=['Year','Game Date']).drop_duplicates()
out.head()
```

Out[22]:

	Year	Game Date	Team_1	Team_2	Home or Away Team_1	Outcome Team_1	Score Team_1	Score Team_2	Error
16159	1970	01/03	San Francisco 49ers	Dallas Cowboys	Home	L	10	17	Good
35209	1970	01/03	Oakland Raiders	Baltimore Ravens	Away	L	17	27	Good
50787	1970	01/03	Baltimore Colts	Oakland Raiders	Home	W	27	17	Good

	Year	Game Date	Team_1	Team_2	Home or Away Team_1	Outcome Team_1	Score Team_1	Score Team_2	Error
50788	1970	01/17	Baltimore Colts	Dallas Cowboys	Away	W	16	13	Good
51970	1970	01/17	Dallas Cowboys	Baltimore Ravens	Home	L	13	16	Good

In [ ]:

```
# CSV Mode
output_name = 'output-test.csv'
out.to_csv(output_name, index=False)

# XLSX Mode
#writer = pd.ExcelWriter(output_name, engine='xlsxwriter')
#out.to_excel(writer, sheet_name='output')
#writer.save()
```