An Investigation of Sports Betting

Research Question:

To what extent can supervised machine learning techniques make NFL sports betting a profitable endeavor?

IB Mathematics Analysis & Approaches HL

Word Count: 3287

May 2024

Table of Contents

Introduction

In this essay, I will answer the question, ‘To what extent can supervised machine learning techniques make NFL sports betting a profitable endeavor’. I will first introduce the ideas behind sports betting and analytics. Subsequently, I will outline the processes used to gather and clean data. Then I will go over the methods to select the best features (independent variables that show correlation with the predicted target) and to train a set of data with these features. Finally, I will explore the predictions acquired from testing a new set of data, and I will conclude by examining the accuracy of these predictions.

While scouting for worthy sports bets and examining the analytics for sports is mostly recreational and guided by one's intuition, bettors and companies have found success by using past data to predict future statistics. I will attempt to replicate this idea, improving it by using better feature selection techniques and more sophisticated models to ultimately improve the chances of making a profit. For my project, I will be using Python to gather, clean and train and test the data. Python is useful for its ability to process a lot of data and the many models it has available. I will be using NFL statistics from 1999-2022, specifically those pertaining to starting quarterbacks in an attempt to look at their seasons statistics and use these statistics to predict next year's total passing touchdowns for each starting quarterback.

Sports Betting

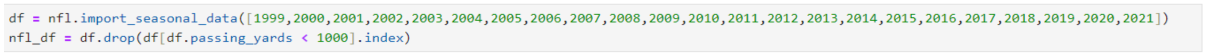
In the United States, Sports Betting odds are given with a plus or minus and then a number attached. For example, on August 3, 2023, the odds in the Seattle Mariners vs Los Angeles

Angels MLB game is -135 for the Mariners winning and + 110 for betting on the Angels to win. In American odds, the team associated with the plus is the underdog and for every $100 bet on that team, the profit gained is the number following the plus sign (in this case a profit of 110 dollars for every 100 dollars bet on the Angles if they win). Betting on the favorite, the team with negative odds yields a profit of 100 dollars for a bet with the amount to the left of the negative side. In this case, correctly betting 135 dollars on the Mariners to win would yield a 100-dollar profit.

The odds given by a sportsbook also give the bookmakers predictions on the probability of a bet. The implied probability for negative is $\frac{100(Odds)}{Odds + 100}$ (Ortega-Araiza). The implied probability for the underdog = $\frac{100}{100(Odds + 100)}$ (Ortega-Araiza). In this case, the implied probability of the Mariners winning is 57.45%, while the implied probability of the Angels winning is 47.62%. The sum of these probabilities is 105.07%, above the typical sum of probabilities (100%) meaning that these odds are designed for sports bookkeepers to make a profit. For example, one would need to be more than 47.62% confident that the Angels are going to win or more than 57.45% that the Mariners will win to make a profit on this bet, even though the actual probabilities of each team winning (which must be a sum of 100%) may be 55 and 45 percent. Vigorish is the term that represents the additional amount that sportsbooks charge for taking a bet. Mathematically it can be understood as the difference in the amount that the bettor has to bet and the amount they can win. For example, a set of -110 bets on both sides mean that a successful bet of 110 dollars will yield 100 dollars of profit, whereas an unsuccessful bet will yield 110 dollars of loss. Thus, the expected loss on a 110-dollar bet is 10 dollars, meaning the Vigorish or amount the bookmakers are expected to gain is 10/110 = 0.09090 dollars for every dollar betted. The vigorish is calculated by 1/ the sum of implied odds, meaning that in the Angels and Mariners game, the vigorish is 1/1.0507 = 0.9517.

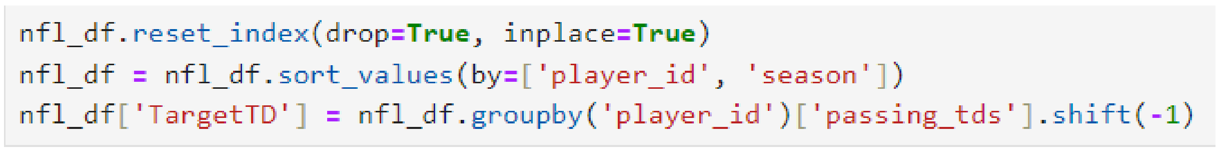
Data Gathering

To gather the data, I used Python because it has many libraries that can help with data science related problems. I began by using a library called nfl_data_py to import player data from the 1999-2021 season. I subsequently filtered this data to include only the statistics of quarterbacks with more than 1000 yards.



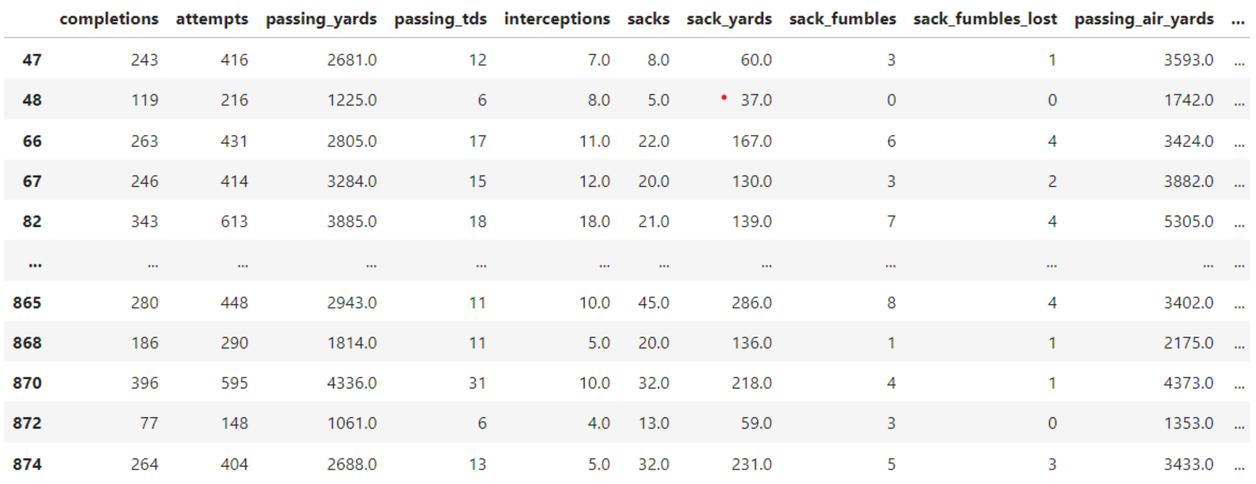
```python
df = nfl.import_seasonal_data([1999,2000,2001,2002,2003,2004,2005,2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018,2019,2020,2021])
nfl_df = df.drop(df[df.passing_yards < 1000].index)
```

Afterwards, I shifted the number of touchdowns passed in every year, the target backwards 1 year, so each data frame would show the current year's stats and next year's touchdowns, ensuring the model only is using the data from the current season to predict a statistic for the next season.



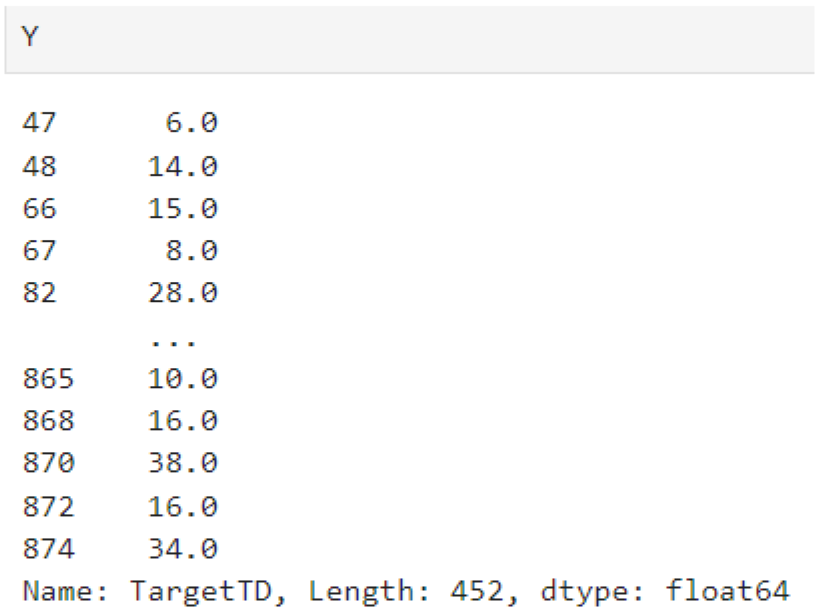
```python
nfl_df.reset_index(drop=True, inplace=True)
nfl_df = nfl_df.sort_values(by=['player_id', 'season'])
nfl_df['TargetTD'] = nfl_df.groupby('player_id')['passing_tds'].shift(-1)
```

This data set would represent the training set, a group of data in which the model fits toward, and then uses this fit to make predictions on a different set of data, the testing set. This training set includes the data of quarterbacks from 1999-2020, and their touchdowns passed for in the next year (2000 to 2021). I then separated the training set into X, an assortment of features designed to predict the number of passing touchdowns thrown and Y, the passing touchdowns thrown for the next year.



	completions	attempts	passing_yards	passing_tds	interceptions	sacks	sack_yards	sack_fumbles	sack_fumbles_lost	passing_air_yards	...
47	243	416	2681.0	12	7.0	8.0	60.0	3	1	3593.0	...
48	119	216	1225.0	6	8.0	5.0	37.0	0	0	1742.0	...
66	263	431	2805.0	17	11.0	22.0	167.0	6	4	3424.0	...
67	246	414	3284.0	15	12.0	20.0	130.0	3	2	3882.0	...
82	343	613	3885.0	18	18.0	21.0	139.0	7	4	5305.0	...
...	...	...	...	...	...	...	...	...	...	...	...
865	280	448	2943.0	11	10.0	45.0	286.0	8	4	3402.0	...
868	186	290	1814.0	11	5.0	20.0	136.0	1	1	2175.0	...
870	396	595	4336.0	31	10.0	32.0	218.0	4	1	4373.0	...
872	77	148	1061.0	6	4.0	13.0	59.0	3	0	1353.0	...
874	264	404	2688.0	13	5.0	32.0	231.0	5	3	3433.0	...

Figure 1: A sample of data frame X, in total 452 rows and 66 columns



```
Y
```

```
47       6.0
48      14.0
66      15.0
67       8.0
82      28.0
        ...
865     10.0
868     16.0
870     38.0
872     16.0
874     34.0
Name: TargetTD, Length: 452, dtype: float64
```

Figure 2: A sample of data frame y, which contains the next years passing touchdowns for each Quarterback in X

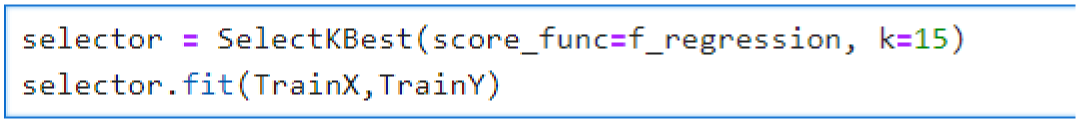
From there, I used the same nfl.import_seasonal_data method to accumulate a testing set of player statistics from 2021 and their corresponding next year passing touchdowns in 2022.

Feature Selection

In terms of machine learning, bias is the inability for a machine learning model to capture the true relationship. Variance is the difference in fits between the training and testing data set. The

Bias -Variance Tradeoff is central to Machine Learning: How can one obtain the lowest bias and variance? (“What Is the Difference Between Bias and Variance?”) However, when the bias is decreased, variance is increased and vice versa. One of the problems with trying to obtain such a low bias is overfitting. If we were to use all 60 features, or independent variables, just to predict the number of passing touchdowns thrown, some of the variables like rushing yards, that have very minimal effect on passing touchdowns, could be assigned a coefficient or used in a decision tree to decrease the sum of squared residuals in the training set. While this may seem beneficial, in reality, we are assigning value to a parameter that has no meaning, and the value is accounting for the luck and randomness of sports. Then, we get a model with low bias, but when we start making predictions on the testing set with the model, all these meaningless features suddenly have illegitimate value which in turn diminishes the accuracy of these predictions resulting in high variance (the training data is fit so well, but the testing data is not). This is called overfitting. Another issue that can occur when doing regression is multicollinearity, which occurs when one or more of the features are highly correlated with each other. For example, if a player's fantasy points, and fantasy points ppr(fantasy points just measured slightly differently) are highly correlated, then we would be unable to distinguish the effect of fantasy points on passing touchdowns because as fantasy points increased, the fantasy points ppr would also experience a similar increase.

To reduce the chance of overfitting and multicollinearity and to optimize the Bias-Variance tradeoff, feature selection is used. In my model I used the SelectKBest feature selector from a Python module called sklearn (SelectKBest).



```
selector = SelectKBest(score_func=f_regression, k=15)
selector.fit(TrainX,TrainY)
```

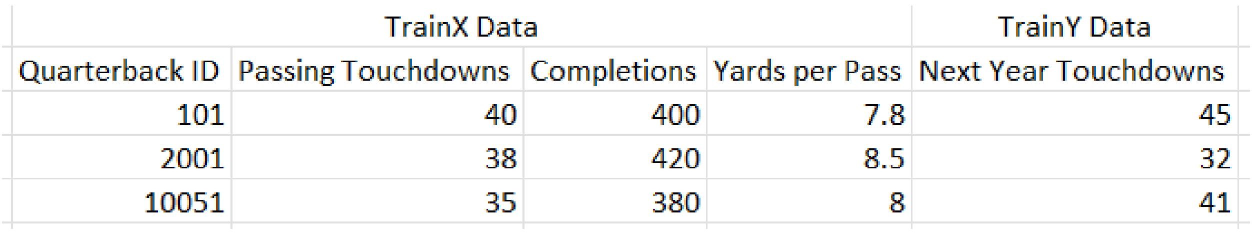
This feature selector takes two parameters, or inputs into the function that affects the output. These features are score_func and k. The score_function can be any metric that can identify the strength of a relationship between each feature and the target variable. In this case, scoring is based on the f_regression function metric and the selector will find the 15 features with the highest f_regression score to fit and train the model.

F_Regression

The f_regression function returns a F Statistic that is highly correlated with the Pearson correlation coefficient (r) and uses the F Statistic to return a p-value.

(““Sklearn.Feature_selection.F_regression.”)

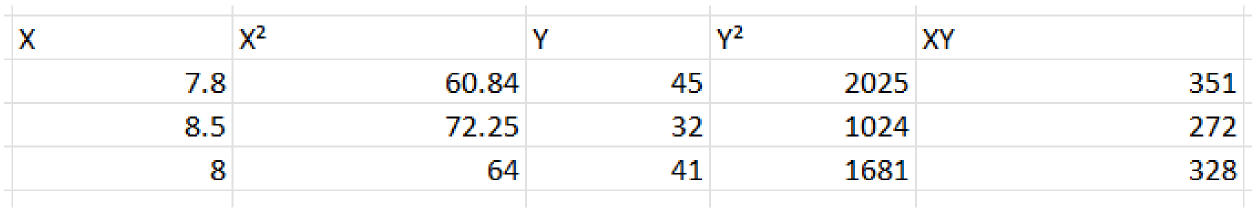
To explain the F Statistic calculation, I will use a sample subset of the training X and training Y data sets to calculate these.



TrainX Data				TrainY Data
Quarterback ID	Passing Touchdowns	Completions	Yards per Pass	Next Year Touchdowns
101	40	400	7.8	45
2001	38	420	8.5	32
10051	35	380	8	41

When looking at this data we can infer that the feature Yards Per Pass and the target Next Year Touchdowns have somewhat of a strong negative correlation. Based on intuition we can infer that the Quarterback ID would not affect the target variable. The f_regression test can confirm this, and give us more information, such as is this feature statistically significant, and does any subset of these features demonstrate multicollinearity?

First, we will look at the Yards per Pass columns (X) and the Next Year Touchdowns (Y) columns. From there we add three columns X^2, and Y^2, squaring each value and a XY column, multiplying x and y for each row.



X	X^2	Y	Y^2	XY
7.8	60.84	45	2025	351
8.5	72.25	32	1024	272
8	64	41	1681	328

From here we calculate $\Sigma X, \Sigma Y, \Sigma X^2, \Sigma Y^2$, and ΣXY.

$\Sigma X = 7.8 + 8.5 + 8 = 24.3$

$\Sigma Y = 45 + 32 + 41 = 118$

$\Sigma X^2 = 60.84 + 72.25 + 64.00 = 197.09$

$\Sigma Y^2 = 2025 + 1024 + 1681 = 4730$

$\Sigma XY = 351 + 272 + 328 = 951$

We know that n, the number of samples is 3.

To calculate Pearson's correlation coefficient , we use the formula:

$$r = \frac{n\sum xy - (\sum x)(\sum y)}{\sqrt{(n\sum x^2 - (\sum x)^2)(n\sum y^2 - (\sum y)^2)}}$$

Which after simplification yields us $\frac{-14.4}{\sqrt{207.48}}$ which equals to a r value of -0.99971077. This R-value can be confirmed by running a r_regression test in Python to calculation Pearson's correlation coefficient. In this case, the newxdf represents the subset of TrainX data used above, and the y represents the subset of TrainY data used above.

```
from sklearn.feature_selection import r_regression
rreg1 = r_regression(newxdf, y)
```

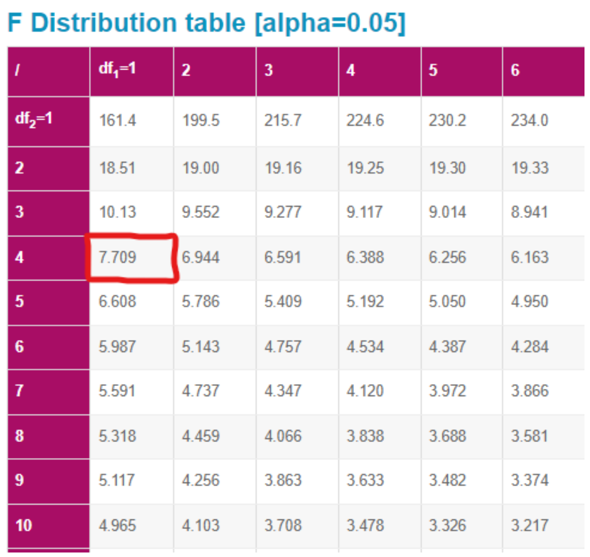
```
rreg1
```

```
array([-0.99971077])
```

Then, we convert the R-value to its coefficient of determination (R^2) value. Squaring (-0.9971077) gives us a R-squared value of 0.999421637. We can convert R^2 to the F statistic with the formula:

$$F_c = R^2 * \frac{n-2}{1-R^2}$$

Evaluating this formula gives us an F-statistic of 1727.98. This F-statistic can be proven as statistically significant using the F-distribution at a significance level of 0.05.



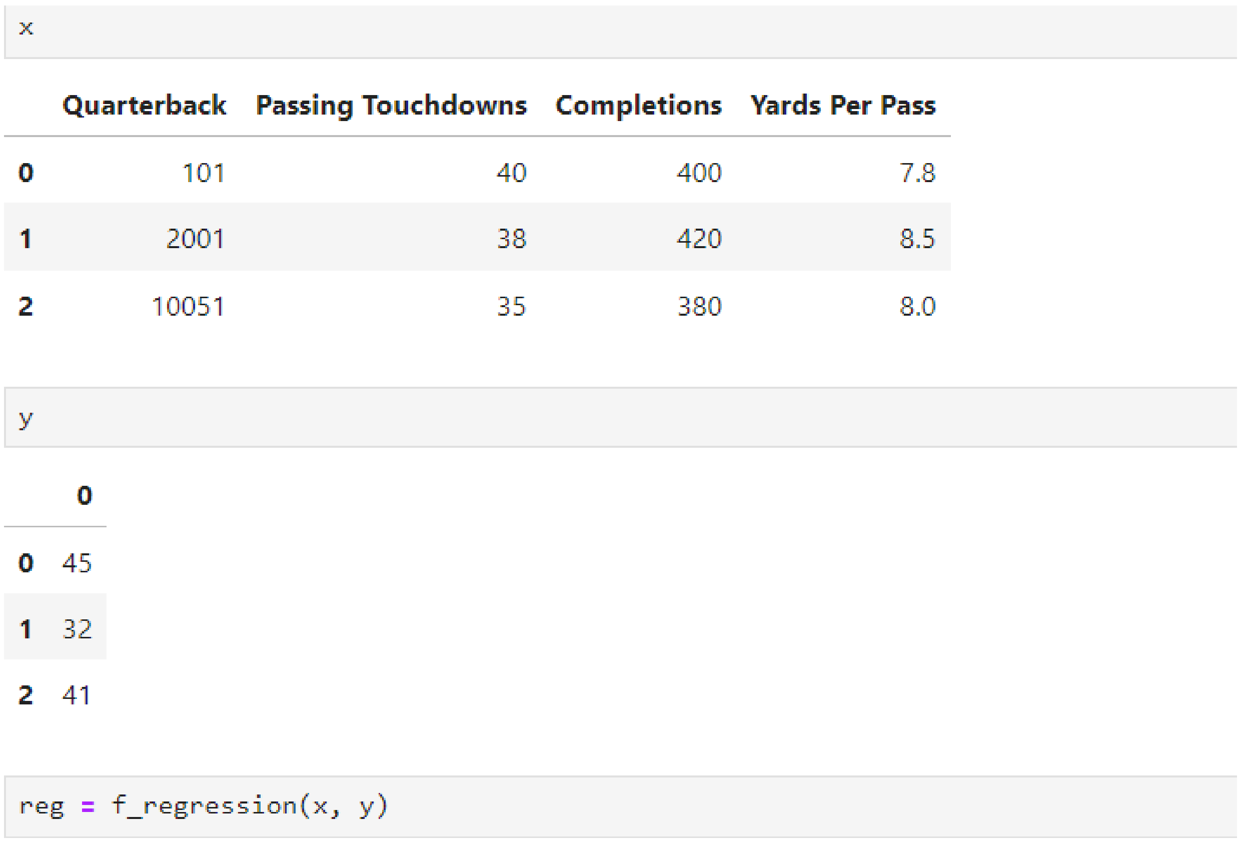
F Distribution table [alpha=0.05]

/	df_1=1	2	3	4	5	6
df_2=1	161.4	199.5	215.7	224.6	230.2	234.0
2	18.51	19.00	19.16	19.25	19.30	19.33
3	10.13	9.552	9.277	9.117	9.014	8.941
4	7.709	6.944	6.591	6.388	6.256	6.163
5	6.608	5.786	5.409	5.192	5.050	4.950
6	5.987	5.143	4.757	4.534	4.387	4.284
7	5.591	4.737	4.347	4.120	3.972	3.866
8	5.318	4.459	4.066	3.838	3.688	3.581
9	5.117	4.256	3.863	3.633	3.482	3.374
10	4.965	4.103	3.708	3.478	3.326	3.217

The df_1 represents the degrees of freedom in the numerator, the degrees of freedom when calculated something known as the Sum of Squares Between (SSB), and df2 represents the degrees of freedom in the denominator, when calculating the Sum of Squares Within (SSW).

If m = the number of columns and n = the total number of samples, df1 = m -1 and df2 = m(n-1). In our case, m = 2 and n = 3, so df1 =1 and df2 = 4. Because our F - statistic is larger than 7.709, we can conclude that the relationship between the Yards per Pass and the Next Year Touchdowns is statistically significant.

Using python and sklearn, a library that runs almost everything related to machine learning, we can easily get the exact p-value for this data point and the F-statistics and the p-values for the three other columns (Quarterback ID, Passing Touchdowns and Completions)



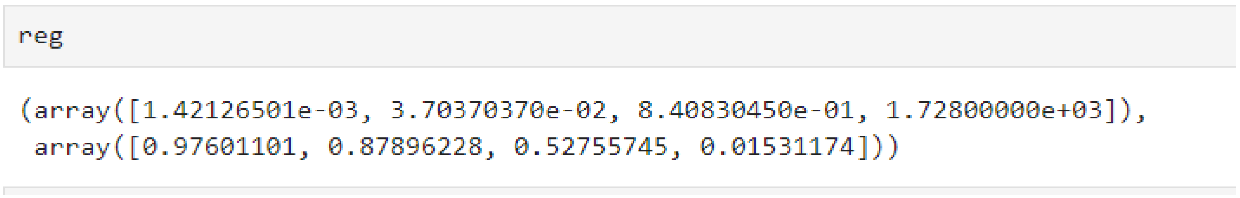
```
x
```

	Quarterback	Passing Touchdowns	Completions	Yards Per Pass
0	101	40	400	7.8
1	2001	38	420	8.5
2	10051	35	380	8.0

```
y
```

	0
0	45
1	32
2	41

```
reg = f_regression(x, y)
```

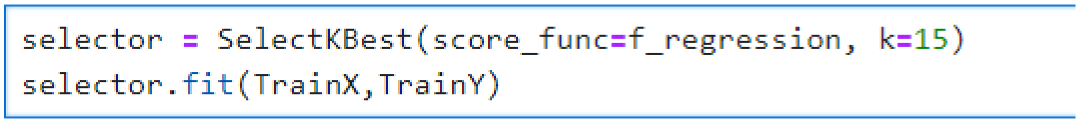


```
reg
```

```
(array([1.42126501e-03, 3.70370370e-02, 8.40830450e-01, 1.72800000e+03]),
 array([0.97601101, 0.87896228, 0.52755745, 0.01531174]))
```

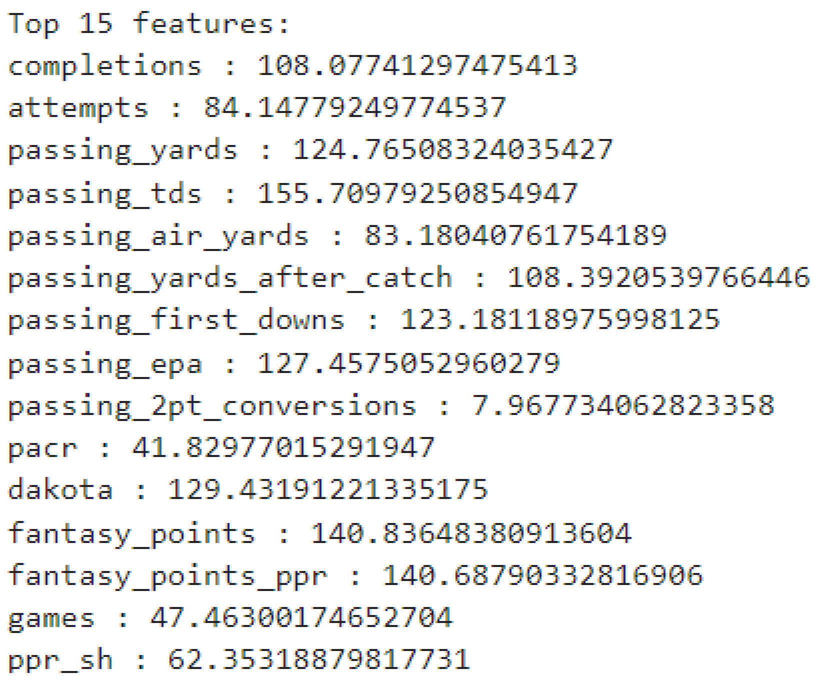
The output, reg, is presented as a collection of two arrays that work from left to write. The first array displays the F statistic and the second displays the P-value.

The feature selector, referenced earlier, and shown below will look at the entirety of the TrainX and Train Y data, and for every feature in the TrainX data set will calculate the F-statistic with respect to the TrainY values. Then, the selector will find the 15 features with the highest F-statistic and output them as a list.



```
selector = SelectKBest(score_func=f_regression, k=15)
selector.fit(TrainX,TrainY)
```

After running this selector and a for loop to get both the 15 selected columns, and their corresponding F-statistics, we get an output.



```
Top 15 features:
completions : 108.07741297475413
attempts : 84.14779249774537
passing_yards : 124.76508324035427
passing_tds : 155.70979250854947
passing_air_yards : 83.18040761754189
passing_yards_after_catch : 108.3920539766446
passing_first_downs : 123.18118975998125
passing_epa : 127.4575052960279
passing_2pt_conversions : 7.967734062823358
pacr : 41.82977015291947
dakota : 129.43191221335175
fantasy_points : 140.83648380913604
fantasy_points_ppr : 140.68790332816906
games : 47.46300174652704
ppr_sh : 62.35318879817731
```

Gradient Boosting Regression

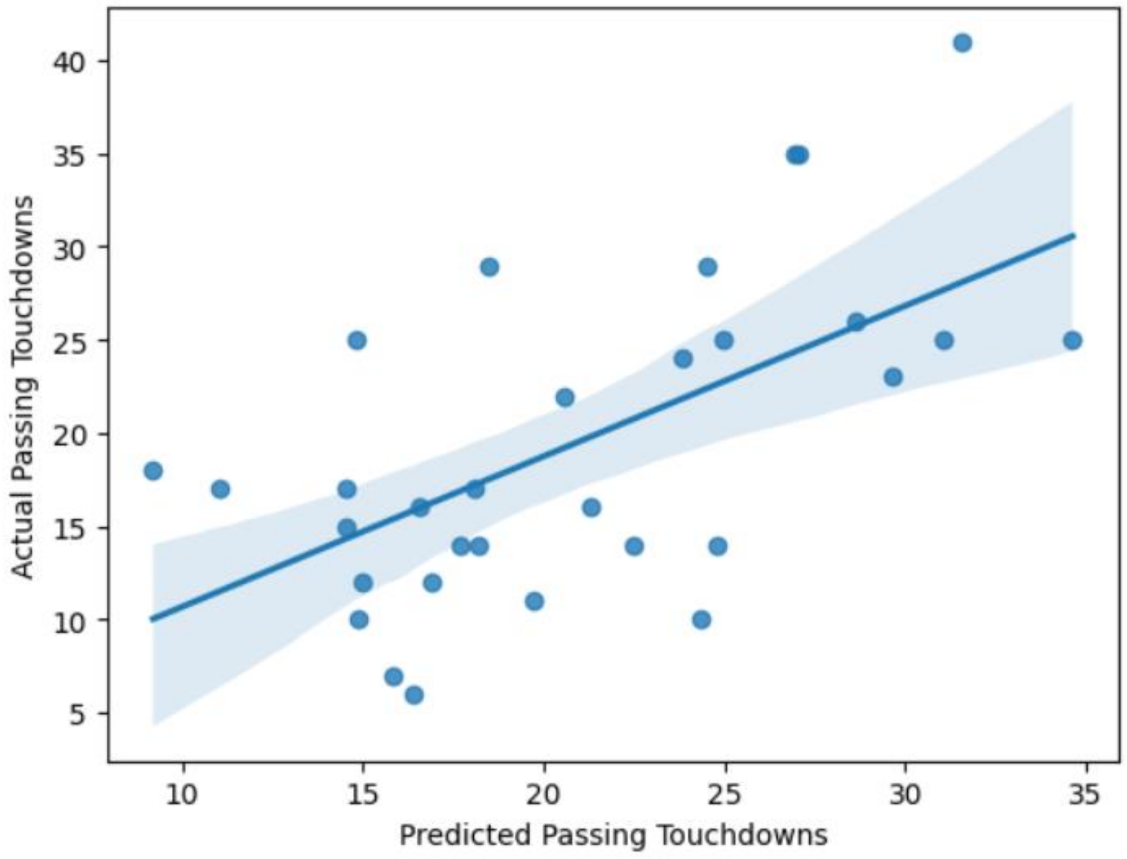
There are many different regression tests within the supervised machine learning category. To find the optimal test, I tested almost every available regression model, from more common models like Logistic and Linear Regression to regularization regression models like Lasso and Ridge Regression. I also tried decision tree regression models like Random Forest Regression and Gradient Boosting regression.

To test each model, I created a testing set, with statistics from quarterbacks who threw for more than 1000 passing yards in the 2021 NFL season and then made a list with the touchdowns they threw in the following season. For every instance, I would fit the model to the training data. Fitting the model, means that for every model, for example linear regression, coefficients are added to each of the selected features to minimize the loss function, which in this case was the mean squared error. In the case of Random Forest Regression, a bunch of decision trees would be made to ensure that the mean squared error was minimized. From there each model retained the coefficients or decision trees used and then predicted next year's passing touchdowns from the testing data. When I compared each model's predictions to the actual 2022 passing touchdowns thrown, and calculated the mean squared error, the model using Gradient Boosting Regression (GBR) has the lowest mean squared error.

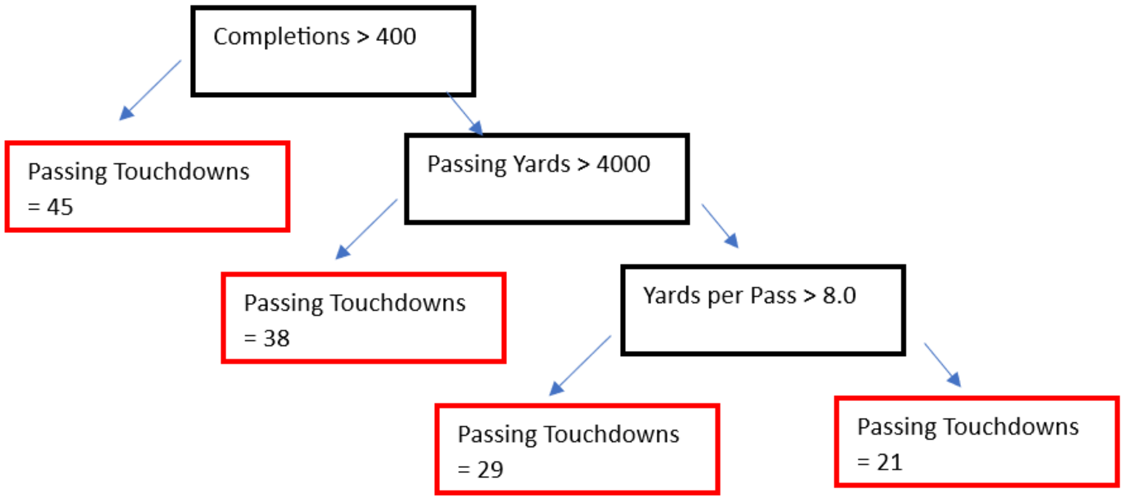
```
#Gradient Boosting Regression
GBReg = GradientBoostingRegressor()
GBReg.fit(TrainX,y)
preds = GBReg.predict(Real_Test_X)
mse = mean_squared_error(Test_Y, preds)
mse
```

```
49.253578862714804
```

To look at the results we obtained more holistically we can reference a graph comparing the predicted passing touchdowns to the actual passing touchdowns.



Gradient Boosting Regression works by using decision trees, specifically regression trees, decision trees that deal with numeric data and give out puts that are numeric rather than classification trees which output True or False, or Yes or No. A simplified regression tree is presented below. To use the regression tree, one begins with the top statement (the root) and looks at the question it poses. If the answer to the question is true, they continue leftwards, and if the answer is false one continues to the right. The process is repeated until someone ends at the leaf or leaf node, which instead of posing a question, gives a numeric answer (Starmer).



For example, a quarterback with 300 completions, 3000 passing yards, and 10 yards per pass, would be false in the first two decision nodes, but true in the third node, meaning their predicted passing touchdowns would be 29.

The decision nodes and roots are calculated based on the split that minimizes the sum of residuals.

However, gradient boost regression is slightly different from other types of regression with decision trees.

The two requirements for Gradient Boosting Regression are input data where x and y have the same length n, and a loss function that the algorithm attempts to minimize.

$$L\big(y, f(x)\big)$$

$$\{(x_i, y_i)\}$$

The most popular loss function for GBR is ½(observed - predicted)^2. This is because if we take the derivative with respect to predicted, we get (predicted - observed) which is just the negative of the residual, which makes the derivative easy to be calculated.

GBR starts with a single leaf that represents an initial guess.

$$F_0(x) = argmin\sum_{i=1}^{n} L(y_i, \gamma)$$

The initial guess, F_0 , attempts to minimize the sum from i = 1, to n, the number of samples of the Loss function between γ (the predicted values) and y_i, the observed values. After expanding $F_0(x)$, we can differentiate the equation with respect to the predicted values to prove that for numeric values $F_0(x)$ is equal to the average of y_i.

$$\frac{d}{d\ Predicted}\ \frac{1}{2}(y_1 - predicted)^2 + \frac{1}{2}(y_2 - predicted)^2 + \frac{1}{2}(y_3 - predicted)^2 \ldots + \frac{1}{2}(y_n - predicted)^2$$

$$predicted - y_1 + predicted - y_2 + predicted - y_3 \ldots + predicted - y_n$$

$$n(predicted) = y_1 + y_2 + y_3 \ldots + y_n$$

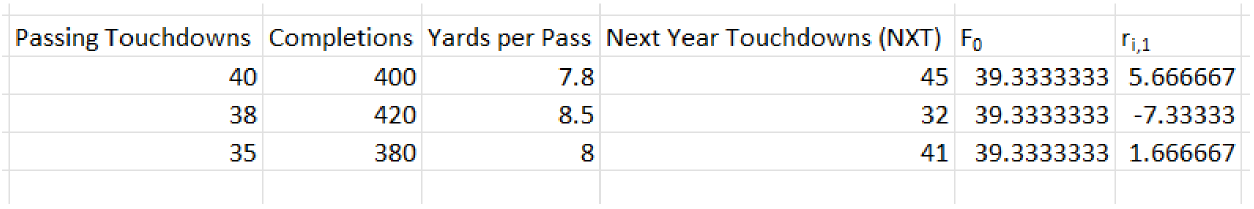
$$predicted = \frac{y_1 + y_2 + y_3 \ldots + y_n}{n}$$

Next, GBR computes the pseudo residual r_{im} for i values (from i = 1 to n) and m trees (from m=1 to M, the maximum number of trees specified within the parameters of GBR)

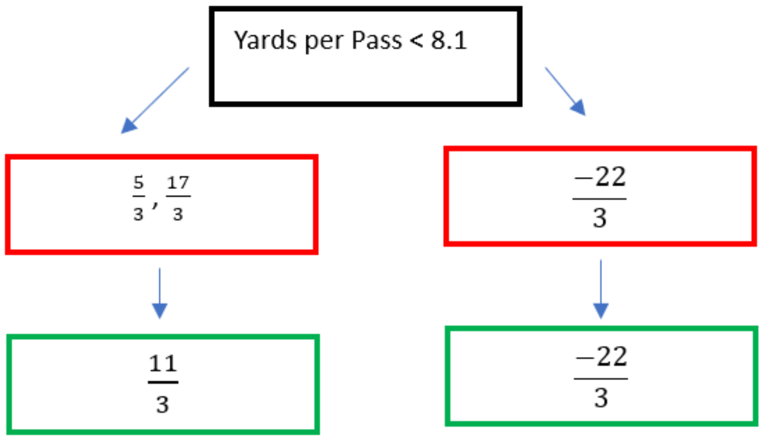
$F_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]$ for i = 1,2,…,n

Essentially, GBR just takes the negative of the derivative of the loss function, which in this case, ends up being (observed - predicted) which is the formula for a residual. Once the residuals for every i value is calculated, GBR begins to build a regression tree to predict these residuals.

We can use the same data table used to illustrate feature selection to demonstrate a regression tree used to predict residuals.



Passing Touchdowns	Completions	Yards per Pass	Next Year Touchdowns (NXT)	F_0	$r_{i,1}$
40	400	7.8	45	39.3333333	5.666667
38	420	8.5	32	39.3333333	-7.33333
35	380	8	41	39.3333333	1.666667



Note: This regression tree is a simplified version of an actual tree used in GBR. Typically, there will be more leaves and each leaf will have multiple values attached to it.

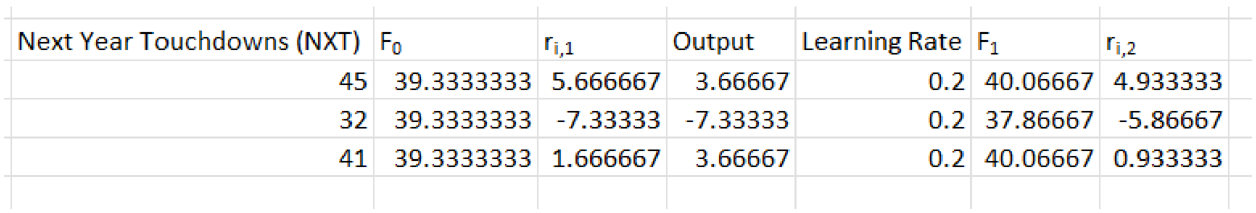
In the case of $r_{1,1}$ and any other case where two or more values are included in a leaf, the output value for the leaf is calculated using this function:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} R_{jm}(x), \gamma_{jm} = argmin \sum^{L} (y_i, F_{m-1}(x_i) + \gamma)$$

This formula is almost identical to the formula used to find our initial guess. The only differences are that instead of including all values from i to n, this summation only includes x_i values such that x_i is a member of $r_{j,m}$. Likewise, we can prove that the output value is just the average of any number of values in the leaf (Starmer).

Next, we update our pseudo residual column. We use a learning rate (γ) between 0 and 1 to scale the output value for $r_{1,m}$ to $r_{n,m}$, and then add this to $F_{m-1}(x)$.

Using a learning rate of 0.2, we can update our predictions, and their pseudo residuals.



Next Year Touchdowns (NXT)	F_0	$r_{i,1}$	Output	Learning Rate	F_1	$r_{i,2}$
45	39.3333333	5.666667	3.66667	0.2	40.06667	4.933333
32	39.3333333	-7.33333	-7.33333	0.2	37.86667	-5.86667
41	39.3333333	1.666667	3.66667	0.2	40.06667	0.933333

Now with the improved $r_{i,2}$ residuals, we can continue this process until m = M, the maximum number of trees specified as a parameter.

In the eyes of Jerome Friedman, the pioneer behind gradient boost, all these small improvements would lead to a model with relatively low bias and variance.

Within GBR, a couple other parameters help reduce overfitting. A maximum number of trees can be set, to prevent the model from excessively fitting to irrelevant patterns in the training data.

Additionally, a maximum depth of trees restricts the number of decision nodes per each tree to limit the variance of the model (Starmer).

Hyperparameter tuning is the idea of optimizing the model, by changing the parameters within a model. For example, the mean squared error between the actual results and the predictions was 49.254 when using the default parameters of GBR. However, we can run a nested loop to generate thousands of mean squared errors by changing the learning rate and M (maximum number of trees). By running this, we can obtain an even lower mean squared error of 43.323, by using a M of 53, instead of the default of 100 and a learning rate of 0.13, rather than the default of 0.1. Of course, a M value of 53 won't always obtain the best results when testing on data in the future, but we can plot the relationships between the value of parameters and the MSE.

Conclusion

In the future, to improve the results of the model, more data could be used, and more parameters could be considered. Additionally, some data points are not accounted for due to lack of features. For example, new player additions can improve or weaken a team. By looking at the residuals in the testing sets and analyzing less numeric values like coaching changes, trades, ownership, fan bases and home field advantages, I can begin to create numeric representations of these unexplored values to possibly decrease the mean squared error.

However, the inconsistency in sports, means that a mean squared error of 43.323 for measuring total passing touchdowns is pretty accurate. To fully answer my question on whether supervised machine learning techniques make NFL sports betting a profitable endeavor, I compared the predictions from my model with the Vegas Insider 2022 projected touchdowns. Ultimately, my model correctly predicted the over/under for total passing touchdowns 20 out of 26 times, resulting in correct bets 76.92 percent of the time. While these predictions are a small sample size, it proves that these supervised machine learning techniques can make sports betting profitable.

Placing 50 dollars, on every single one of these bets with -110 odds, would lead to a profit of 20 winning bets(50 dollars per bet x $\frac{100\ dollars\ profit}{110\ dollars\ spent}$)- 6 losing bets(50 dollars) = \$609.09.

The idea that machine learning techniques can optimize decision based on sports, a field based on so many variables including luck, means that machine learning techniques can excel no matter the field, whether it is sports or stocks or any other analysis.

Works Cited

Bhandari, Aniruddha. "Multicollinearity: Causes, Effects and Detection Using VIF (Updated 2023)." *Analytics Vidhya,* 17 July 2023, www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/.

Dominguez, B. (2021, February 4). Modeling NFL game outcomes using python and scikit-learn. Open Source Football. Retrieved March 23, 2023, from https://www.opensourcefootball.com/posts/2021-01-21-nfl-game-prediction-using-logistic-regression/

Gupta, Sparsh. "The 7 Most Common Machine Learning Loss Functions." *Built In,* builtin.com/machine-learning/common-loss-functions. Accessed 4 Aug. 2023.

Monahan, A. (2022, October 6). Math Behind Sports Betting. OddsJam. Retrieved March 29, 2023, from https://oddsjam.com/betting-education/math-behind-sports-betting

Nourse, G. (2022, June 23). A machine learning algorithm for predicting outcomes of MLB games. Medium. Retrieved March 27, 2023, from https://towardsdatascience.com/a-machine-learning-algorithm-for-predicting-outcomes-of-mlb-games-fa17710f3c04

Ortega-Araiza, Javier. "How to Calculate and Read Sports Betting Odds." *LINES,* 6 Aug. 2023, www.lines.com/guides/how-to-read-and-calculate-sports-odds/1267.

Predictive modeling - time-series regression, linear regression models. MathWorks. (n.d.). Retrieved March 31, 2023, from https://www.mathworks.com/discovery/predictive-modeling.html

Shanks, Payton. "Seattle Mariners at Los Angeles Angels Odds, Picks and Predictions." *USA Today,* 5 Aug. 2023, sportsbookwire.usatoday.com/2023/08/05/seattle-mariners-at-los-angeles-angels-odds-picks-and-predictions-11/.

"Sklearn.Feature_selection.F_regression." *Scikit-Learn,* scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html#sklearn.feature_selection.f_regression. Accessed 15 July 2023.

"Sklearn.Feature_selection.SelectKbest." *Scikit-Learn,* scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html. Accessed 1 Aug. 2023.

Staff, VI. "2022-23 NFL Player Prop Betting Odds." *VegasInsider,* 2022, www.vegasinsider.com/nfl/odds/player-prop-betting-odds/.

Starmer, Josh."Gradient Boost Part 1 (of 4): Regression Main Ideas." *YouTube*, uploaded by StatQuest with Josh Starmer, 25 . 2019, https://www.youtube.com/watch?v=2xudPOBz-vs&t=1261s.

Starmer, Josh."Gradient Boost Part 2 (of 4): Regression Details." *YouTube*, uploaded by StatQuest with Josh Starmer, 01 Apr. 2019, https://www.youtube.com/watch?v=2xudPOBz-vs&t=1261s.

"What Is the Difference Between Bias and Variance?" *Master's in Data Science*, 19 July 2022, www.mastersindatascience.org/learning/difference-between-bias-and-variance/#:~:text=A%20model%20that%20exhibits%20small,bias%20will%20overfit%20the%20target.

6 types of regression models in Machine Learning You should know about. upGrad blog. (2022, December 1). Retrieved April 18, 2023, from https://www.upgrad.com/blog/types-of-regression-models-in-machine-learning/