Predicting Atlanta Falcons NFL Touchdowns with Regression Modelling

Bhargav Ashok

2025-05-19

0. Setup - Install Packages: tidyr, readxl, dplyr, car, glmnet, Metrics, 3dscatterplot

Loading our Atlanta Falcons Data

```
Atlanta_Falcons_data <- read_excel("Atlanta_Falcons_data.xlsx")
head(Atlanta_Falcons_data)
```

```
## # A tibble: 6 x 18
##      Rk Player      From    To     G Pos      AV   Tgt   Rec `Ctch%`   Yds `Y/R`
##   <dbl> <chr>      <dbl> <dbl> <dbl> <chr> <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>
## 1     1 Julio Jon~  2011  2020   135 WR      119  1320   848   0.642 12896  15.2
## 2     2 Roddy Whi~  2005  2015   171 WR      107  1377   808   0.587 10863  13.4
## 3     3 Terance M~  1994  2001   126 WR       67   989   573   0.579  7349  12.8
## 4     4 Alfred Je~  1975  1983   110 WR       61    NA   360  NA      6267  17.4
## 5     5 Andre Ris~  1990  1994    78 WR       53   463   423  NA      5633  13.3
## 6     6 Jim Mitch~  1969  1979   155 TE       47    NA   305  NA      4358  14.3
## # i 6 more variables: TD <dbl>, Lng <dbl>, `Y/Tgt` <dbl>, `R/G` <dbl>,
## #   `Y/G` <dbl>, Fmb <dbl>
```

Preparing our data, cleaning extensively and training

Here, we are cleaning our data with the tidyr package, filtering by specific offensive positions and then splitting our data into a trained and tested set.

```
vars_needed <- c("TD", "Tgt", "Rec", "Ctch%", "Yds")

offensive_positions <- c("QB", "RB", "WR", "TE")

clean_df <- Atlanta_Falcons_data %>%
  drop_na(all_of(vars_needed)) %>%
  filter(Pos %in% offensive_positions)

set.seed(123)
n       <- nrow(clean_df)
train_i <- sample(n, 0.8*n)
train_df <- clean_df[ train_i, ]
test_df  <- clean_df[-train_i, ]
```

Fitting our Model (regular MLR)

Next, we fit our model using the equation: Y = (Beta)0 + (Beta)1X1 + (Beta)2X2 + ... + (Beta)nXn + error (assuming normal distribution)

```
mlr_fit <- lm((TD) ~ Tgt + Rec + `Ctch%` + Yds, data = train_df)
summary(mlr_fit)
```

```
##
## Call:
## lm(formula = (TD) ~ Tgt + Rec + `Ctch%` + Yds, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6585 -0.6978  0.1129  0.5897  7.8258
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.530806   0.722703  -0.734   0.4641
## Tgt         -0.028128   0.015728  -1.788   0.0763 .
## Rec          0.012559   0.015881   0.791   0.4306
## `Ctch%`      0.415199   1.030521   0.403   0.6877
## Yds          0.009374   0.001537   6.100 1.36e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.09 on 119 degrees of freedom
## Multiple R-squared:  0.8945, Adjusted R-squared:  0.891
## F-statistic: 252.3 on 4 and 119 DF,  p-value: < 2.2e-16
```

The summary statistics are displayed above. Note the coefficients for the MLR equation and the adjusted R-squared Value.

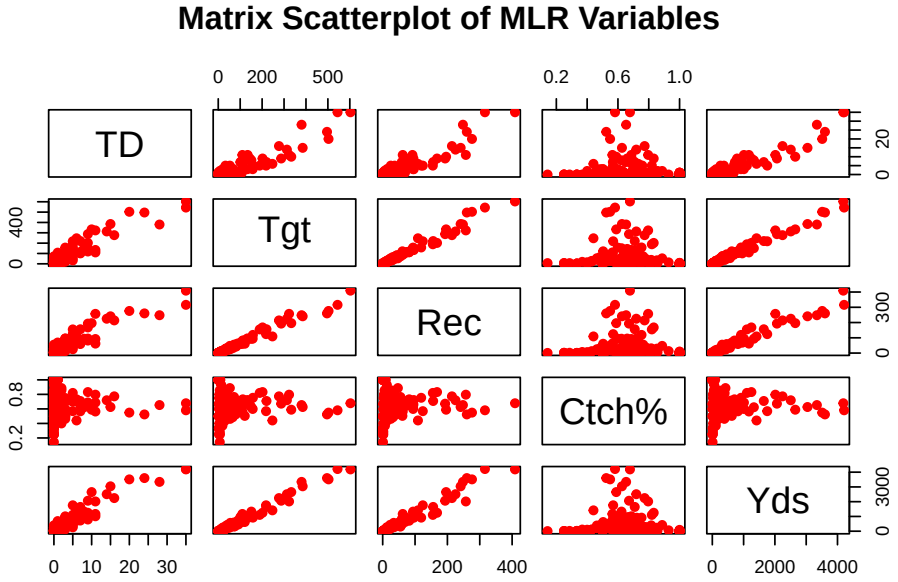
The MLR equation can be identified and written as: "TD = -0.531 + 0.028*Tgt + 0.013*Rec + 0.415*Ctch% + 0.009*Yds".

This equation suggests that Catch percentage has the largest positive effect on Touchdowns, with Receptions being second, followed by Yards.

Data Visualization of the Matrix Scatterplot

Here we see that the scatter plot compares all the different variables together to check for linearity and how each variable influences each other. We may see non-linearity between some variables but that will be accounted for later on.

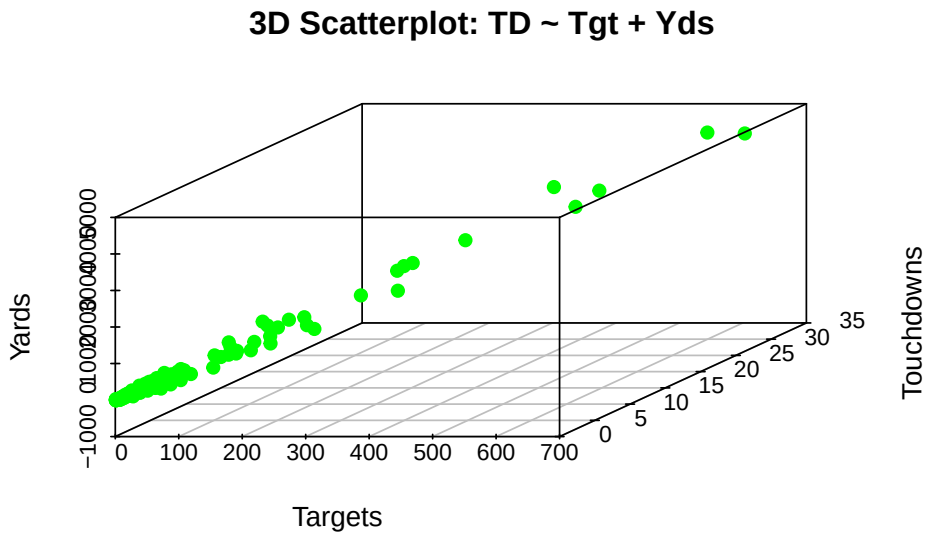
```
pairs(
  ~ TD + Tgt + Rec + `Ctch%` + Yds,
  data = train_df,
  main = "Matrix Scatterplot of MLR Variables",
  col = "red",
  pch = 19
)
```



Data visualization of the 3d scatterplot

Here we are taking the top two linearly correlated variables (not largest positive effect) and plotting them on a 3d scatterplot using the R 3d scatterplot package.

```
scatterplot3d(
  x = train_df$Tgt,
  y = train_df$TD,
  z = train_df$Yds,
  main = "3D Scatterplot: TD ~ Tgt + Yds",
  xlab = "Targets",
  ylab = "Touchdowns",
  zlab = "Yards",
  color = "green",
  pch = 19
)
```

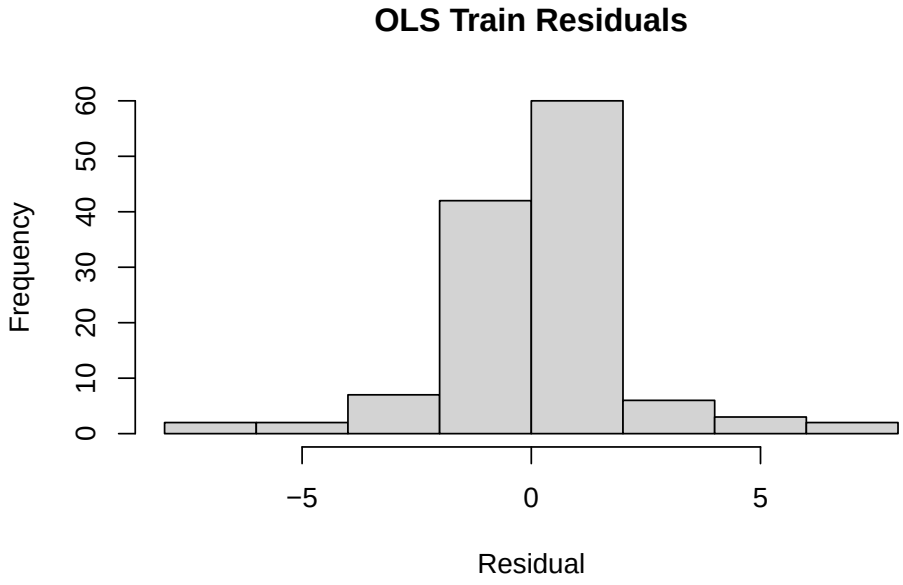


We can see that there may be some slight over fitting in the 3d plot, indicating too strong of a relationship between these variables (Tgt, Yds vs TDs).

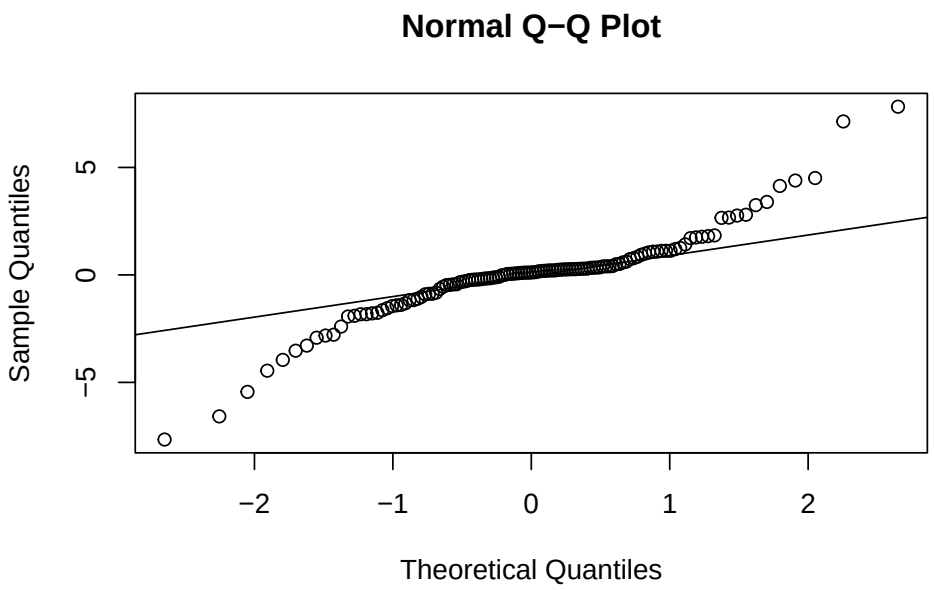
MLR fit data display (looking at model data)

Then, we display plots for our fitted data (qqnorm and histogram of residuals)

```
hist(resid(mlr_fit), main="OLS Train Residuals", xlab="Residual")
```



```r
qqnorm(resid(mlr_fit)); qqline(resid(mlr_fit))
```

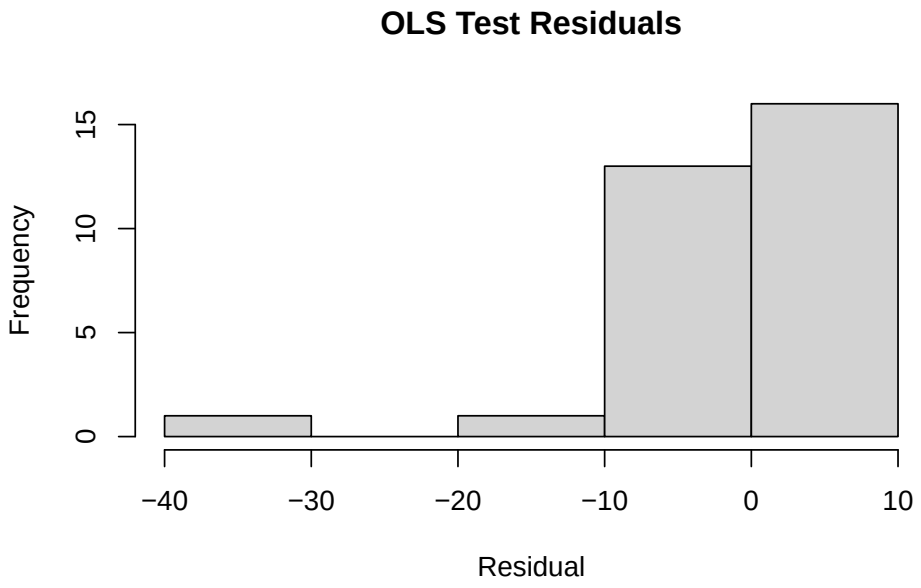


From the plots, we see that our trained residual data is roughly normal along with the qqplot (shows how well the dataset follows a normal distribution) being roughly normal (with deviation in the extreme values or outliers).

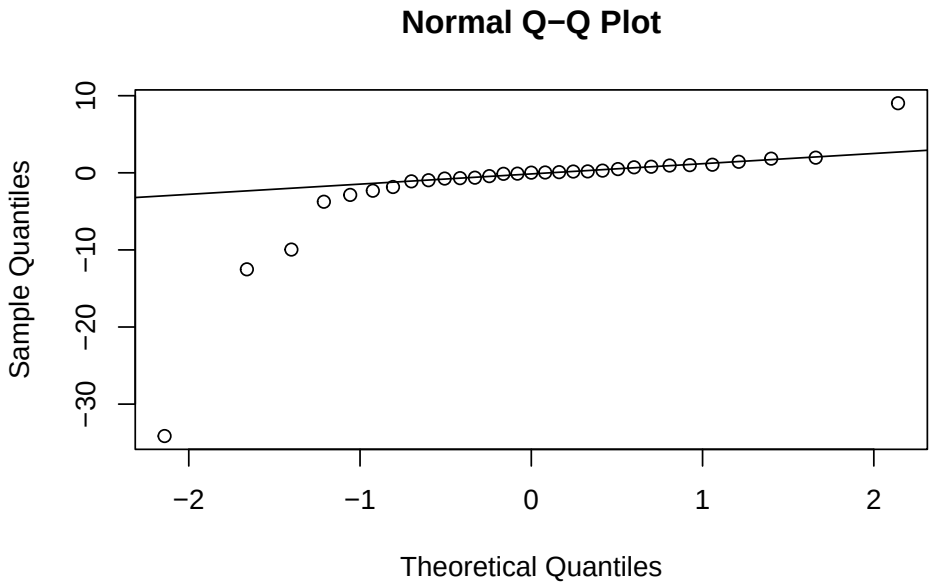
test data display (looking at unseen test data)

After that, we look at our unseen test data plots.

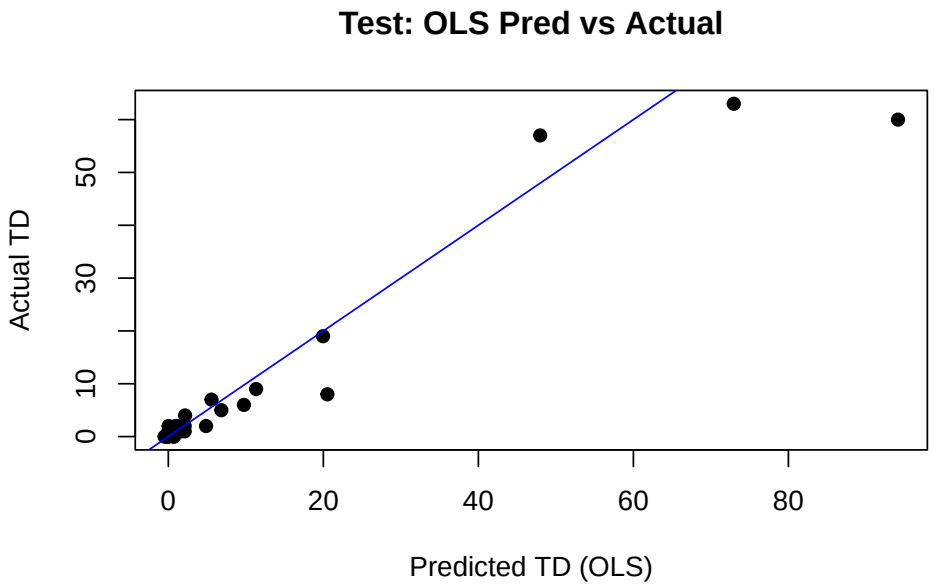
```r
test_df$pred_ols  <- predict(mlr_fit, newdata = test_df)
resid_test_ols    <- test_df$TD - test_df$pred_ols

# Plots
hist(resid_test_ols, main="OLS Test Residuals", xlab="Residual")
```




```
qqnorm(resid_test_ols); qqline(resid_test_ols)
```




```
plot(
  test_df$pred_ols, test_df$TD,
  xlab="Predicted TD (OLS)", ylab="Actual TD",
  main="Test: OLS Pred vs Actual", pch=19
)
abline(0,1,col="blue")
```



We see the linear regression line on our predicted test data above along with the qqnorm and residual plots. While normality and non-skewed data is important in general, the trained data is a more accurate representation of it compared to the tested data (as the model uses the trained data for statistical analysis) so the plots here don't matter as much for inference (but it can help us check how our ML model is performing). However, it does suggest more over fitting within the graph as it is performing like the trained data and does not show a linear correlation due to the clustering of lower data points.

Using Cross validation + Ridge/Lasso Regression

```
vif(mlr_fit)
```

```
##         Tgt         Rec    `Ctch%`         Yds
## 100.409011  41.539915   1.140006  53.222809
```

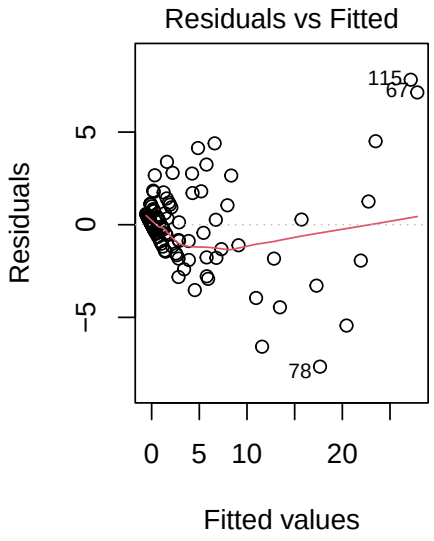
Our VIF (Variance Inflation Factor, found using car package in R) is displayed above. An accurate explanation from Statsmodels can be found here:

"The variance inflation factor is a measure for the increase of the variance of the parameter estimates if an additional variable is added to linear regression. It is a measure for multicollinearity of the design matrix".

We see that the VIF > 10 (a rule of thumb is that if the VF is greater than 5-10, it indicates high multicollinearity). This is not ideal for this scenario because the variables need to be standardized to account for the number inflation in multicollinearity (since they are supposed to be statistically significant, which is not reflected in the regular MLR).

There is also some concern of Heteroscedasticity (not a contsant variance across residuals)

```
par(mfrow = c(1, 2))

plot(mlr_fit, which = 1)
```



The plot above shows some deviation to the constant horizontal line (indicating non constant variance).

Due to all of these issues (over fitting, multicollinearity and Heteroscedasticity), we need to switch to a new type of regression model for more accurate results. We use a technique called K-folds cross validation, where the data is split into multiple subsets and is iterated more than once in order to account for the multicollinearity inflation + Heteroscedasticity, as well improving the model to see how accurately it can predict unseen data points. This involves putting our train/test data into matrices, and then running regularized models called Ridge and Lasso regression respectively.

Lasso regression accounts for the absolute value of the important coefficients and shrinks them using a penalty factor. Ridge regression accounts for the squared value of the coefficients and shrinks them with a similar penalty factor (all in the means of regularizing our data (also called hyper parameter tuning)). We use the glmnet package for this.

```
# Prepare matrices
x_train <- model.matrix(TD~Tgt+Rec+`Ctch%`+Yds, train_df)[,-1]
y_train <- train_df$TD
x_test  <- model.matrix(TD~Tgt+Rec+`Ctch%`+Yds, test_df)[,-1]
y_test  <- test_df$TD

# Ridge
cv_ridge  <- cv.glmnet(x_train,y_train,alpha=0)
best_ridge<- cv_ridge$lambda.1se #updated for fitting purposes, more on this later
ridge_mod <- glmnet(x_train,y_train,alpha=0,lambda=best_ridge)
test_df$pred_ridge <- as.numeric(predict(ridge_mod,x_test))
summary(cv_ridge)
```

```
##            Length Class  Mode
## lambda     100    -none- numeric
## cvm        100    -none- numeric
## cvsd       100    -none- numeric
## cvup       100    -none- numeric
```

```
## cvlo       100    -none- numeric
## nzero      100    -none- numeric
## call         4    -none- call
## name         1    -none- character
## glmnet.fit  12    elnet  list
## lambda.min   1    -none- numeric
## lambda.1se   1    -none- numeric
## index        2    -none- numeric
```

```
# Lasso
cv_lasso  <- cv.glmnet(x_train,y_train,alpha=1)
best_lasso<- cv_lasso$lambda.min
lasso_mod <- glmnet(x_train,y_train,alpha=1,lambda=best_lasso)
test_df$pred_lasso <- as.numeric(predict(lasso_mod,x_test))
summary(cv_lasso)
```

```
##            Length Class  Mode
## lambda     59     -none- numeric
## cvm        59     -none- numeric
## cvsd       59     -none- numeric
## cvup       59     -none- numeric
## cvlo       59     -none- numeric
## nzero      59     -none- numeric
## call        4     -none- call
## name        1     -none- character
## glmnet.fit 12     elnet  list
## lambda.min  1     -none- numeric
## lambda.1se  1     -none- numeric
## index       2     -none- numeric
```

```
# RMSE
cat("Ridge RMSE:",  rmse(y_test, test_df$pred_ridge), "\n")
```

```
## Ridge RMSE: 3.97041
```

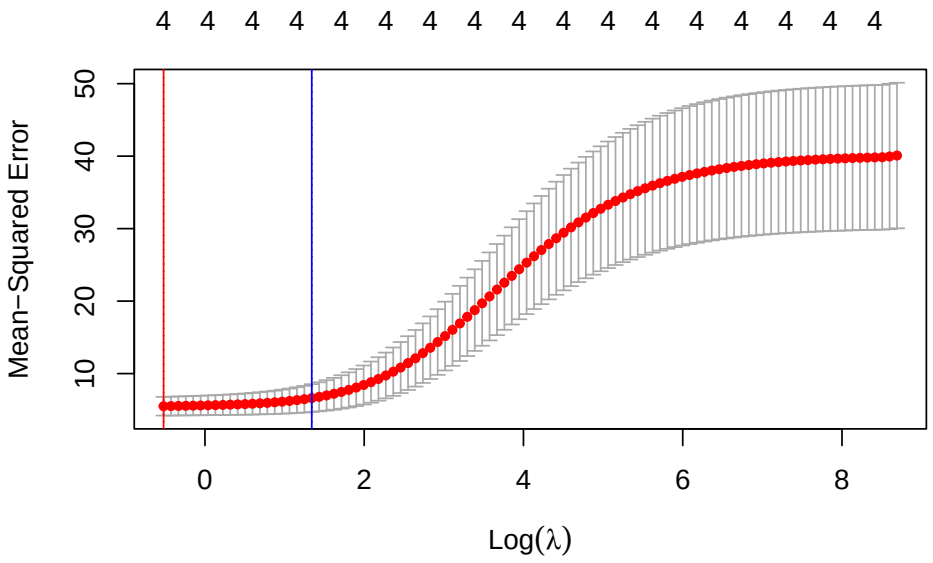
```
cat("Lasso RMSE:",  rmse(y_test, test_df$pred_lasso), "\n")
```

```
## Lasso RMSE: 5.685224
```

We do a Cross Validation of Ridge and Lasso regression to see which one is more accurate. As we can see, Ridge regression has a lower RMSE which is more accurate for our model, so we will plot the Cross validation curve. We get the RSME score from the Metrics Package.

Plotting Ridge regression CV plot

```
plot(cv_ridge)
abline(v=log(cv_ridge$lambda.min), col="red")
abline(v=log(cv_ridge$lambda.1se), col="blue")
```



The cross validation ridge plot is shown above. According to "https://bookdown.org/ssjackson300/Machine-Learning-Lecture-Notes/choosing-lambda.html":

"What is plotted is the estimated CV MSE for each value of (log)lambda on the x-axis. The dotted line on the far left indicates the value of lambda which minimizes CV error. The dotted line roughly in the middle of the x-axis indicates the 1-standard-error lambda- recall that this is the maximum value that lambda can take while still falling within the on standard error interval of the minimum-CV lambda. The second line of code has manually added a dot-dash horizontal line at the upper end of the 1-standard deviation interval of the MSE at the minimum-CV lambda to illustrate this point further". These plots can change with randomization according to our seed number.

We have two lines (red and blue) for our Lambda.min value vs our Lambda.1se value. Deciding which one to use in our predictions comes with trade offs. I elect to go for the lambda.1se value which prevents over/under fitting, though the RMSE is roughly around the same (as explained later).

```
coef(cv_ridge)
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) 0.171892809
## Tgt         0.013327969
## Rec         0.020222572
## `Ctch%`     0.068007282
## Yds         0.002003957
```

Here we see the L2 Squared coefficients, within the loss function (check out the article by Amit Yadav on the Medium in the references below for more detail!) applied and how it shifts the coefficients by accounting for large weights (coefficents in our MLR equation) or overfitting (also mentioned in the article).

The model we've built includes coordinate descent being done under the hood in the glmnet package (important for multi linear regression models), along with the loss function squared coefficients being applied in ridge regression.

An explanation of coordinate descent can be found online by author Xavier Bourret Sicotte:

"Coordinate descent successively minimizes along coordinate directions to find the minimum of a function. At each iteration, the algorithm determines a coordinate, then minimizes over the corresponding hyperplane while fixing all other coordinates.

It is based on the idea that minimization can be achieved by minimizing along one direction at a time, i.e. by solving a much simpler univariate problem in a loop.

In its simplest case of cyclic coordinate descent, we cycle through all variables one at a time, minimizing the cost function with respect to each coordinate.

This computation is all done through the glmnet package automatically, along with cross-validation and our Ridge regression model (As the package is derived from C or another lower level language).

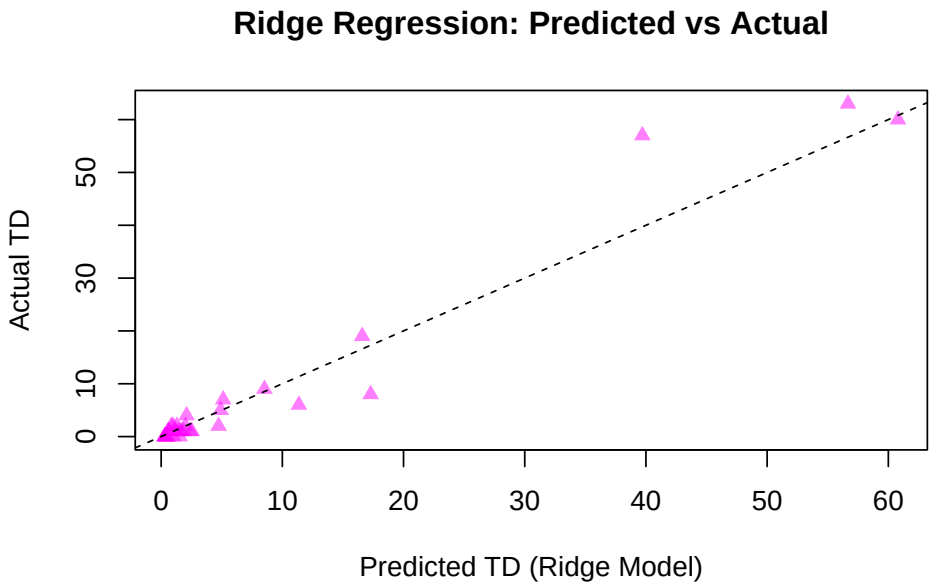
This can be different to gradient descent which looks at all points in a model and tries to minimize their weights (factors/coefficients) instead of going one by one (which is where the loop comes in for coordinate descent)".

I asked a GPT model (o4-mini-high) to breakdown what glmnet does under the hood with coordinate descent in Cross-Validation, and it answered directly with explanations from the glmnet package source (Stanford) and The Statistical Software Journal:

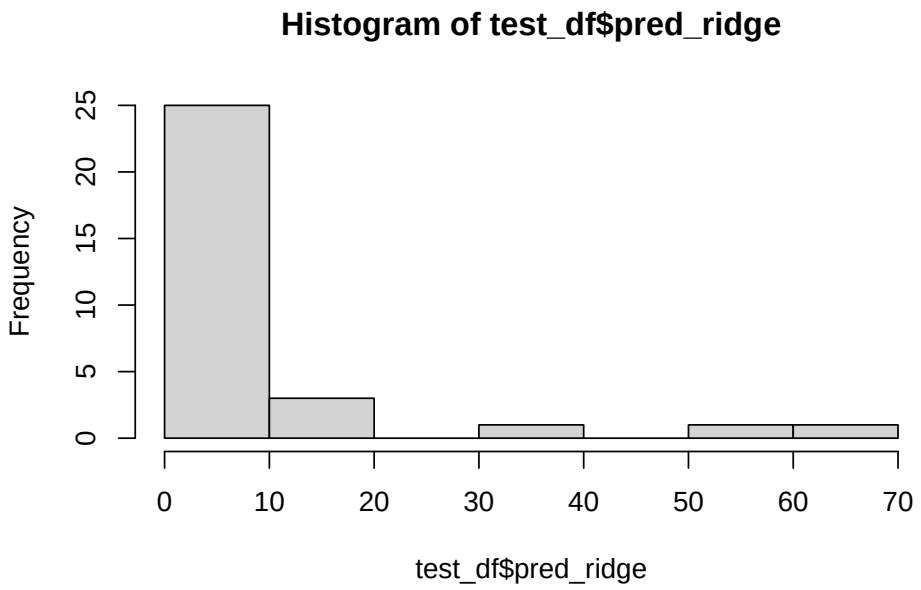
1. Standardize predictors By default, each column of X is centered to mean 0 and scaled to unit variance. This makes the penalty act equally on every coefficient and simplifies the coordinate-wise updates.
2. Choose a lambda-grid It computes a maximum lambda value (the smallest penalty that sets all slopes to zero) and then generates a decreasing geometric sequence of lambdas (e.g. 100 values). You get the full regularization path from very large penalty (all beta=0) down to your specified minimum
3. Warm starts For each lambda in the path (from largest to smallest), the solution at the previous lambda is used as the starting point. Since nearby lambda have similar solutions, this dramatically cuts the number of coordinate-descent passes needed.
4. Cyclical coordinate descent function At a given lambda, it repeatedly loops over each coefficient Beta(j) and does a one-dimensional update while holding the others fixed.
5. Strong-rule screening Before each lambda, glmnet applies a quick test ("strong rules") to drop predictors unlikely to enter the model, solving a smaller problem and then checking KKT conditions to add any wrongly excluded variables back. This can cut CPU time by orders of magnitude on high-dimensional data.
6. Convergence criteria & path output It repeats the coordinate passes until the maximum change in any coefficient is below a tolerance, then moves to the next lambda. The output is a matrix of coefficients (one column per lambda) plus cross-validation helpers if you used cv.glmnet().

Plotting our Comparison graph between MLR and Ridge Regression MLR

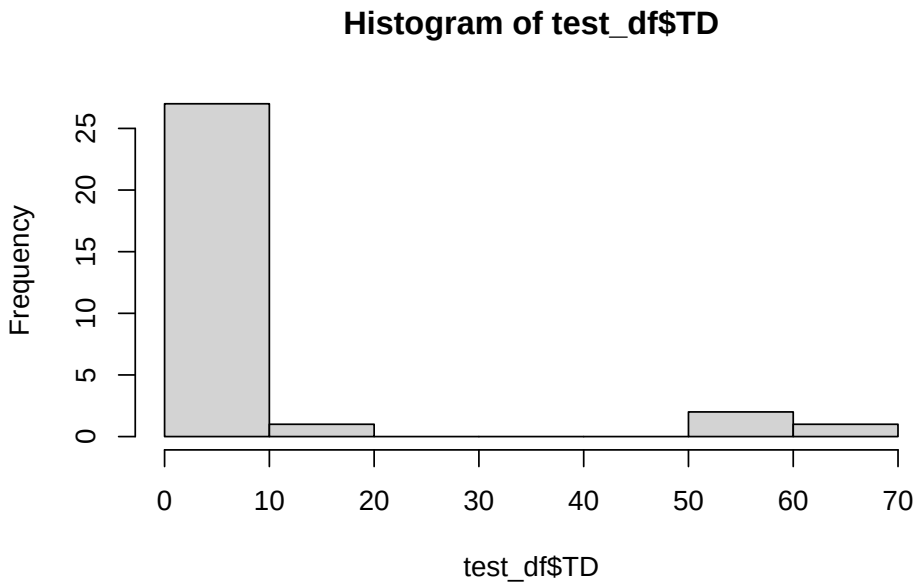
```
plot(test_df$pred_ridge, test_df$TD,
     xlim = range(c(test_df$pred_ridge)),
     ylim = range(c(test_df$TD)),
     xlab="Predicted TD (Ridge Model)", ylab="Actual TD",
     main="Ridge Regression: Predicted vs Actual", pch=17, col=rgb(1,0,1,0.5))
abline(0,1,lty=2)
```




```
hist(test_df$pred_ridge)
```




```
hist(test_df$TD)
```



We can see our Ridge regression graph along with our residuals above.Both predicted TDs and predicted ridge values have similar histograms, which indicates that the data is performing as intended with observed vs predicted values. This is miles better than our obviously over fitted original MLR graph.

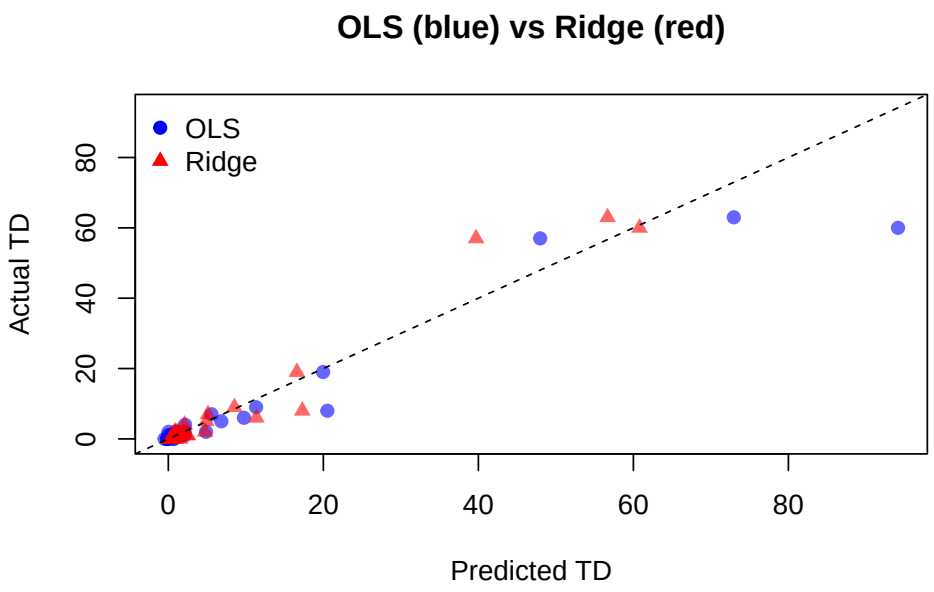
In order to strike a "balanced" data set, we have to tune parameters in order to get a predicted test data set that is neither over fitting or under fitting data points. This is where the lambda 1se function comes in. We used the lambda 1 standard error rule instead of the minimum lambda value used for the RMSE.

The lambda 1se function can be described as:

"The maximum value that lambda can take while still falling within the one standard error interval of the minimum_CV lambda".

While one function is not necessarily better than the other, it's important to note that we are finding the best fit for the Atlanta Falcons. Prioritizing large TDs (lambda.min) would only cloud the data set with noise and lead to potentially inaccurate predictions. Now, everyone wants a superstar on their roster, however in terms of more accurate data, we must pick our trade off.

```
plot(test_df$pred_ols, test_df$TD,
     xlim = range(c(test_df$pred_ols,test_df$pred_ridge)),
     ylim = range(c(test_df$pred_ols,test_df$pred_ridge)),
     xlab="Predicted TD", ylab="Actual TD",
     main="OLS (blue) vs Ridge (red)", pch=19, col=rgb(0,0,1,0.6))
points(test_df$pred_ridge, test_df$TD, pch=17, col=rgb(1,0,0,0.6))
abline(0,1,lty=2)
legend("topleft", legend=c("OLS","Ridge"), pch=c(19,17),
       col=c("blue","red"), bty="n")
```



We can compare our MLR Ordinary Least Squares Regression Model with our Cross-Validated, Ridge Regression Model visually as shown above.

Summary of trained and tested data metrics

```
summary(train_df)
```

```
##        Rk            Player              From            To
##  Min.   :  8.00   Length:124         Min.   :1992   Min.   :1993
##  1st Qu.: 78.75   Class :character   1st Qu.:2001   1st Qu.:2002
##  Median :155.50   Mode  :character   Median :2010   Median :2013
##  Mean   :163.00                      Mean   :2010   Mean   :2011
##  3rd Qu.:253.75                      3rd Qu.:2019   3rd Qu.:2021
##  Max.   :326.00                      Max.   :2024   Max.   :2024
##        G              Pos                 AV               Tgt
##  Min.   :  1.00   Length:124         Min.   :  0.000   Min.   :  1.00
##  1st Qu.: 12.00   Class :character   1st Qu.:  0.000   1st Qu.:  5.00
##  Median : 22.00   Mode  :character   Median :  2.000   Median : 26.50
##  Mean   : 32.19                      Mean   :  9.911   Mean   : 76.27
##  3rd Qu.: 49.00                      3rd Qu.: 11.000   3rd Qu.: 94.25
##  Max.   :222.00                      Max.   :203.000   Max.   :603.00
##       Rec             Ctch%              Yds               Y/R
##  Min.   :  1.00   Min.   :0.1430   Min.   :  -7.00   Min.   :-7.00
##  1st Qu.:  3.00   1st Qu.:0.5248   1st Qu.:  24.75   1st Qu.: 7.70
##  Median : 15.50   Median :0.6565   Median : 182.00   Median :10.15
##  Mean   : 48.98   Mean   :0.6622   Mean   : 544.14   Mean   :10.14
##  3rd Qu.: 59.50   3rd Qu.:0.7732   3rd Qu.: 622.25   3rd Qu.:13.07
##  Max.   :409.00   Max.   :1.0000   Max.   :4212.00   Max.   :26.00
##        TD              Lng              Y/Tgt             R/G
##  Min.   : 0.000   Min.   :-5.00   Min.   :-7.000   Min.   :0.000
```

```
##  1st Qu.: 0.000   1st Qu.:14.00   1st Qu.: 5.000   1st Qu.:0.300
##  Median : 1.000   Median :27.00   Median : 6.350   Median :0.800
##  Mean   : 3.315   Mean   :33.48   Mean   : 6.275   Mean   :1.280
##  3rd Qu.: 3.000   3rd Qu.:52.25   3rd Qu.: 8.000   3rd Qu.:1.825
##  Max.   :35.000   Max.   :94.00   Max.   :26.000   Max.   :5.100
##       Y/G              Fmb
##  Min.   :-0.800   Min.   : 0.00
##  1st Qu.: 2.875   1st Qu.: 0.00
##  Median : 8.400   Median : 0.00
##  Mean   :14.139   Mean   : 3.04
##  3rd Qu.:20.075   3rd Qu.: 2.00
##  Max.   :68.200   Max.   :89.00
```

```
summary(test_df)
```

```
##        Rk           Player               From            To
##  Min.   :  1.0   Length:31          Min.   :1992   Min.   :1992
##  1st Qu.: 53.5   Class :character   1st Qu.:1999   1st Qu.:2002
##  Median :140.0   Mode  :character   Median :2006   Median :2013
##  Mean   :135.9                      Mean   :2008   Mean   :2011
##  3rd Qu.:206.5                      3rd Qu.:2020   3rd Qu.:2020
##  Max.   :304.0                      Max.   :2024   Max.   :2024
##        G              Pos                 AV              Tgt
##  Min.   :  6.00   Length:31          Min.   :  0.00   Min.   :   1.0
##  1st Qu.: 15.00   Class :character   1st Qu.:  1.00   1st Qu.:  14.5
##  Median : 27.00   Mode  :character   Median :  4.00   Median :  39.0
##  Mean   : 46.94                      Mean   : 16.23   Mean   : 189.0
##  3rd Qu.: 84.00                      3rd Qu.:  8.00   3rd Qu.: 118.0
##  Max.   :171.00                      Max.   :119.00   Max.   :1377.0
##       Rec            Ctch%             Yds              Y/R
##  Min.   :  1.0   Min.   :0.2000   Min.   :    5.0   Min.   : 3.70
##  1st Qu.:  9.0   1st Qu.:0.5470   1st Qu.:   66.0   1st Qu.: 9.05
##  Median : 22.0   Median :0.6160   Median :  235.0   Median :11.40
##  Mean   :114.7   Mean   :0.6389   Mean   : 1502.5   Mean   :10.90
##  3rd Qu.: 72.0   3rd Qu.:0.7200   3rd Qu.:  941.5   3rd Qu.:13.20
##  Max.   :848.0   Max.   :1.0000   Max.   :12896.0   Max.   :16.80
##        TD              Lng            Y/Tgt             R/G
##  Min.   : 0.000   Min.   : 5.00   Min.   : 2.000   Min.   :0.100
##  1st Qu.: 0.500   1st Qu.:18.50   1st Qu.: 4.850   1st Qu.:0.450
##  Median : 1.000   Median :40.00   Median : 7.300   Median :0.900
##  Mean   : 8.258   Mean   :41.42   Mean   : 6.732   Mean   :1.613
##  3rd Qu.: 5.500   3rd Qu.:58.50   3rd Qu.: 8.050   3rd Qu.:2.250
##  Max.   :63.000   Max.   :90.00   Max.   :12.000   Max.   :6.300
##       Y/G              Fmb           pred_ols         pred_ridge
##  Min.   : 0.30   Min.   : 0.000   Min.   :-0.4821   Min.   : 0.2838
##  1st Qu.: 3.85   1st Qu.: 0.000   1st Qu.: 0.1247   1st Qu.: 0.7440
##  Median :10.20   Median : 1.000   Median : 1.1333   Median : 1.6456
##  Mean   :20.02   Mean   : 2.742   Mean   : 9.9430   Mean   : 8.0637
##  3rd Qu.:26.45   3rd Qu.: 4.000   3rd Qu.: 6.2060   3rd Qu.: 5.0145
##  Max.   :95.50   Max.   :15.000   Max.   :94.1407   Max.   :60.7987
##    pred_lasso
##  Min.   :-0.4926
##  1st Qu.: 0.1622
##  Median : 1.2909
```

```
##  Mean   : 9.6650
##  3rd Qu.: 5.9212
##  Max.   :85.3247
```

The summary shown by the trained/tested data are regularized and explain the scale of the variables within the ridge regression. We can use the test dataframe metrics to find the ideal candidate for the Atlanta Falcons on the offensive side of the ball.

Conclusion of Findings

We can now safely say that the Atlanta Falcons TDs can be predicted by multiple factors within a game such as catch percentage, receptions, yards, and other numerical factors.

We see that an ideal candidate for the Atlanta Falcons on the offensive side of the ball (particularly WRs, TEs and RBs) would have the optimal stats of:

Approximately 39 Receiving targets (Based on Median) (Tgt)

Approximately 63 Receptions (Based on IQR) (Rec)

Approximately 62% catch percentage (based on Median) (Ctch%)

Approximately 876 - 1500** yards (IQR and mean) (Yds)

** **(note that the mean is not used as a measure of spread here, but rather a range of indication for players with the IQR fitting the offensive scheme of the falcons)** **.

All of these stats would be preferred over a 69 game span (IQR) according to our model, which would be about 4 full seasons in the NFL (69 games from IQR divided by a 17 game NFL season in the modern era equals about 4 full seasons).This does not mean that the player should get these stats within 4 seasons, but rather, the player must **consistently** achieve these ranges of metrics for 4 **consecutive** seasons in order to be a good fit.

This means the player would have to be 25-26 years old to be considered for the Atlanta Falcons.

We can assume that a combination of these stats (with slight variability based on outliers with superstar potential) will lead to a productive increase (or stability in case of outliers) in Touchdowns for the Atlanta Falcons in the case of picking up free agents, resigning players, or trading for talent.

Keep in mind that these stats are based on my personal interpretation and can vary from person to person. I have used data online and interpreted the Falcons offensive scheme from Zac Robinson's (Falcons Offensive Coordinator) Air-Raid philosophy (based on what I've found online).

Future improvements

1. Automating roster/free agent/trade/recruiting data in future findings
2. Creating a classification model detailing other external factors (behavior, team chemistry, etc. modeled for binary classification (0 or 1) or other techniques) can also be used in tandem with this model in order to make an even more accurate decision.
3. Expanding the model to look at more advanced offensive stats/metrics like Y/G, Y/Tgt, etc.
4. Making an extensive ML regression workflow to determine team-fit with Free Agent data, NFL Trade data, or College NCAA data for drafts (NCAA data would have to be adjusted to NFL standards for accurate comparison specifically for the Falcons).
5. Creating multiple models and compare test statistics to figure out which results are more tangible to use based on directions from team scouts, front offices, coaches, etc.

References Used:

https://bookdown.org/ssjackson300/Machine-Learning-Lecture-Notes/choosing-lambda.html

https://www.pro-football-reference.com/teams/atl/career-receiving.htm

https://online.stat.psu.edu/stat462/node/180/

https://www.datacamp.com/tutorial/tutorial-lasso-ridge-regression

https://online.stat.psu.edu/stat462/node/131/

https://www.reddit.com/r/AskStatistics/comments/ycjoy4/what_threshold_is_used_to_assess_multicollinearity/

https://www.datacamp.com/doc/r/regression

https://www.geo.fu-berlin.de/en/v/soga-r/Basics-of-statistics/Linear-Regression/Polynomial-Regression/Polynomial-Regression---An-example/index.html

https://online.stat.psu.edu/stat462/node/177/

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

https://www.datacamp.com/tutorial/multicollinearity

https://www.rdocumentation.org/packages/car/versions/3.1-3

https://www.rdocumentation.org/packages/Metrics/versions/0.1.4

https://www.r-bloggers.com/2021/10/lambda-min-lambda-1se-and-cross-validation-in-lasso-binomial-response/

https://www.techtarget.com/searchenterpriseai/definition/data-splitting#:~:text=Training%20sets%20are%20commonly%20used,the%20final%20model%20works%20correctly.

https://www.datacamp.com/doc/r/scatterplot-in-r

https://www.educba.com/multiple-linear-regression-in-r/

https://www.si.com/nfl/falcons/inside-zac-robinson-s-falcons-offense-speed-shifts-controlled-chaos-01hxvw1caw08?

https://apexfantasyleagues.com/peak-age-for-nfl-wide-receiver/

https://www.statsmodels.org/dev/generated/statsmodels.stats.outliers_influence.variance_inflation_factor.html#:~:text=variance_inflation_factor-,statsmodels.stats.outliers_influence.,wikipedia.org/wiki/Variance_inflation_factor

https://medium.com/biased-algorithms/understanding-l1-and-l2-regularization-in-machine-learning-3d0d09409520

https://glmnet.stanford.edu/articles/glmnet.html

https://www.jstatsoft.org/article/view/v033i01

https://openai.com/index/introducing-o3-and-o4-mini/ (Used for explanations of functions and partial model evaluation only)

https://xavierbourretsicotte.github.io/coordinate_descent.html