STAT 652 PROJECT - FLIGHT DELAY PREDICTION

Krishna Chaitanya Gopaluni

Student No: 301377313

1. INTRODUCTION

The nycflights13 data used in this project has departure delays of the flights in 2013 departing from three NYC airports (EWR, JFK and LGA). This project is mainly focused on the regression analysis approaches to predict the departure delays. Departure delay prediction involves identifying important explanatory variables contributing to delay.

2. DATA

The nycflights13 dataset is maintained by CRAN which has year 2013 all flight (336,776 fights) information that departs from the airports EWR, JFK and LGA in Newyork. The 'fltrain' dataset used in this project has 43 explanatory variables out of which 11 are categorical and **'dep_delay'** as the target. Derived features from preprocessing were also used for the analysis.

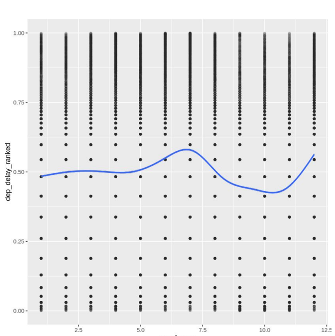
3. METHODS

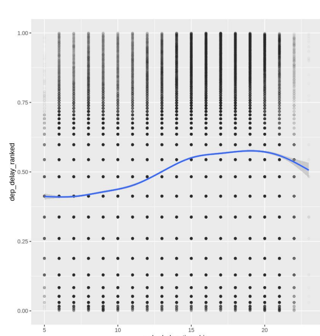
3.1 Preprocessing, Feature Engineering

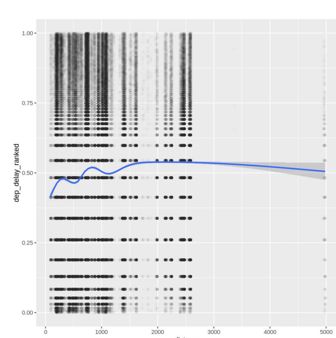
Columns with more than 12.5% of NA values were removed. FeatureHashing is used (murmur3.32 function), which is best practice to deal with categorical columns having a large number of unique values. Because it always returns the same output for a given string which is helpful to handle categorical features and murmur hash has less collision rate. Also, hashing can deal with any new categorical values in the new holdout test set. The rows which had NA values were dropped. Binning technique is used on the schedule departure and scheduled arrival times which maps to any of the 24 bins (one for each hour of a whole day). Day of year, day of the week and month of year from `time_hour` are calculated. The intuition is, the day of the year, month or week indirectly may tell us the seasonal and weather impacts on the flight delays. Other columns with constant values like **tzone**, **dst**, **tz**, **year.x** were removed. **dep_tim**, **arr_time**, **arr_delay**, **air_time** will not have any effect on departure delay as they are the variables to be considered when the flight took off and not before taking off. Dropping of other columns is assured by feature importance calculation using GBM and XGBoost. Assumptions to drop other columns are in the '**##5'** section of the code.

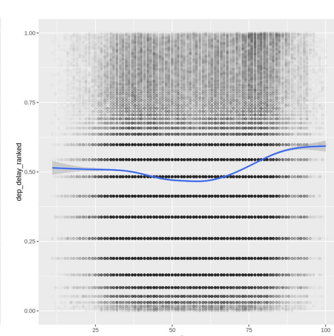
3.2 Exploratory Analysis

The departures are high in the months of summer. There's also an increase in the departure delays in the months of winter. As the day passes (which part of the day the scheduled departure time belongs to) there is an increase in the departure delays. Departure delays are increasing and slightly fluctuating with distance to travel. The departure delays are large when the temperatures are too high or too low.

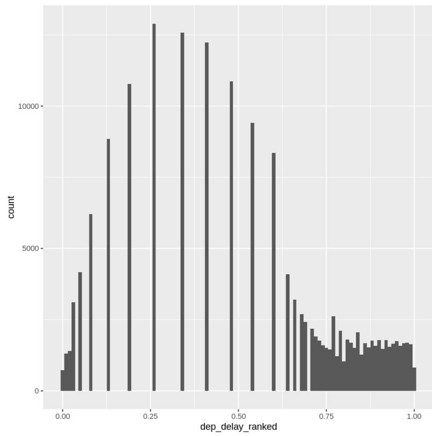








3.2 Model Evaluation Criteria



Mean Absolute Error(MAE) changes with the variance of prediction errors. It cannot identify the high variance in the frequency distribution of the errors, and remains constant if error variance is constant. If model predictions has high variance in the frequency distribution of errors, RMSE is good measure. The RMSE gives a relatively high weight to large errors, because errors are squared before they are averaged.

In current scenario, the response variable 'dep_delay' has its values with huge magnitude in fourth quartile (Image to the left). So, RMSE should be more useful if there is a case of large errors because if high variance in the frequency distribution of the data.

With the split of 80% training and 20% of test, 5 fold CV is used on the training set for best model selection. After selecting the best model configuration, entire dataset is used to build the model. After that, this model is used to test the predictions on the released hold-out set.

3.3 Modelling

Although, models like GAM, Ridge regression and Lasso regression were performing better compared to the baseline model, RandomForests and XGBoost were giving best results.

XGBoost is an optimized and distributed gradient boosting library used for both regression as well as classification. Apart from being a boosting model, XgBoost implements features sub-sampling to further reduce the chances of overfitting. XGBoost borrows the feature sub-sampling concept from the world of bagging. In addition to implementing the feature subsampling feature, XGBoost is an extremely fast, distributed and parallel implementation of boosting technique specifically designed to reduce training times. All these features in addition to the advantages that tree based models provide inherently, XGBoost has quickly become a favorite in Kaggle competitions.

Gbtree is chosen as the boosting parameter for xgboost. Gbtree is an additive model in a forward stage-wise fashion. it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function (RMSE is used with 5 fold CV). So, XGBoost in parallel builds many such GBtrees with different weighted combinations of the data points & feature sampling then finds the best model out of ensemble of models. Slightly similar results were obtained both for bagging technique (using RandomForests) and XGBoost.

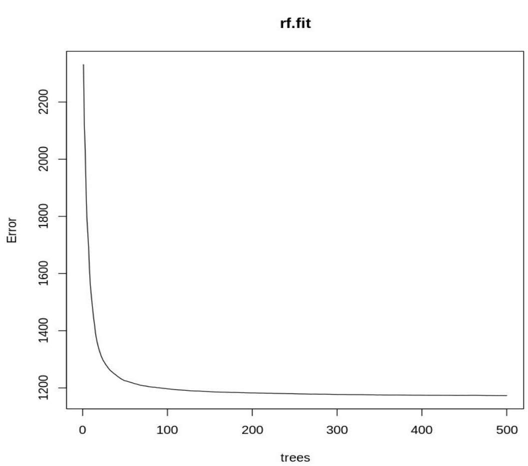
The algorithm has been trained using the following parameters turns out to have good results: eta = 0.01, max_depth = 5, min_child_weight = 5, subsample = 0.65, colsample_bytree = 1, 5 fold CV with 1000 rounds. GBM has also slightly similar important features on the training data. Please find the XGB implementation in '**##16'** section of code.

4. RESULTS

These are the results for the regression analysis using different approaches with only 20 important features and scaling. The baseline model simply returns the average delay from the test dataset.

Model	MAE	RMSE
Basemodel	Test - 21.577 Hold-Out - 21.2	Test -37.79 Hold-Out - 37.4
GAM	Test - 20.31 Hold-Out - 20.17	Test -36.56 Hold-Out - 36.27
Lasso	Test - 20.30 Hold-Out - 20.16	Test - 36.57 Hold-Out - 36.2
Ridge	Test - 20.31 Hold-Out - 20.17	Test - 36.56 Hold-Out - 36.27
GBM	Test - 19.95 Hold-Out - 19.77	Test - 35.99 Hold-Out - 35.7
RandomFrst	Test - 19 Hold-Out - 18.8	Test - 34.87 Hold-Out - 34.37
XGBoost	Test - 18.39 Hold-Out - 18.1	Test - 34.2 Hold-Out - 33.9

Each approach has some improvement from the baseline model. Because response variable is skewed towards 4th quadrant, high variance of frequency distribution of the errors might possible. So, we need to look for the model which has relatively less RMSE.



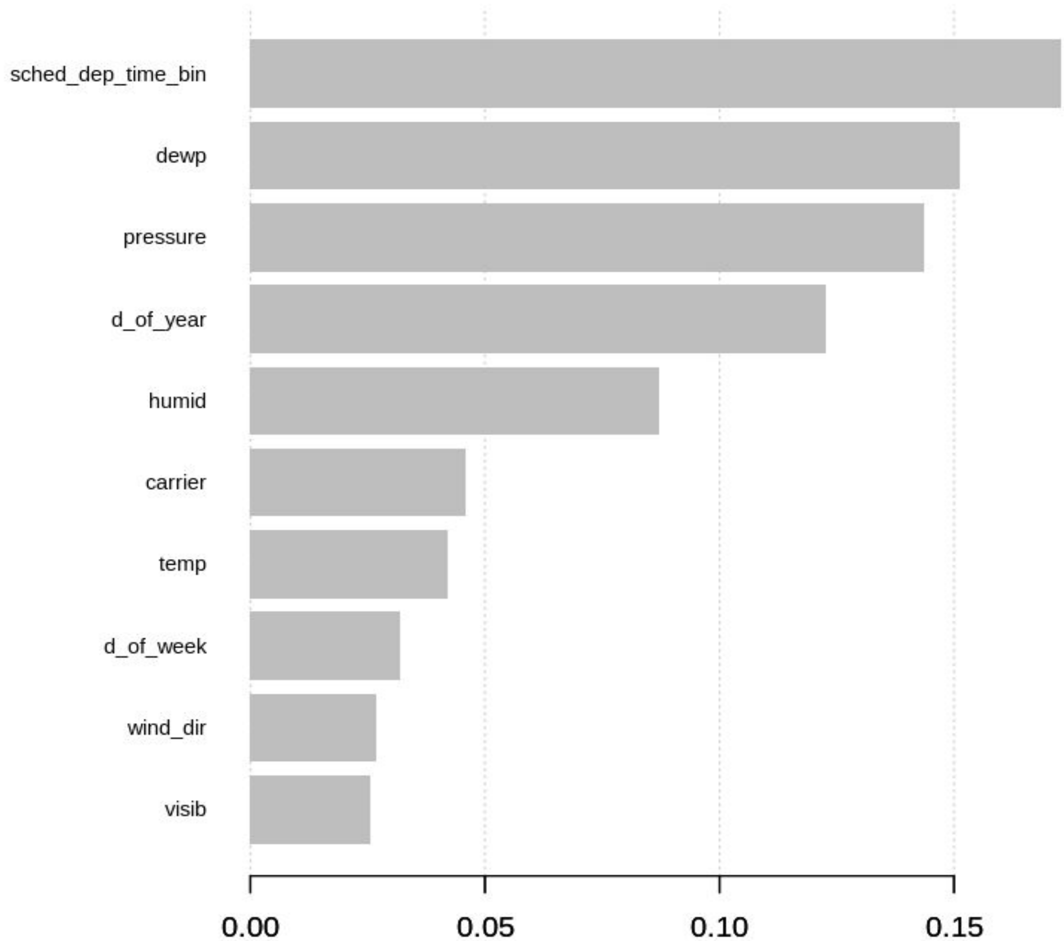
The bagging technique is used in RF. RF performed better in reducing the variance of errors and there is min over fit. We can see that by comparing both test and hold-out scores. The below image represents the training error is going down as the number of trees increasing during training. The number of trees with min mse are 498 in random forests.

Furthermore, RF could not perform better than XGBoost. Further tuning the parameters of Random Forest led to overfitting. Cross validation using 10 fold to find the best parameters for RF took a training time of took almost 10 hours. Other model plots can be seen in code section '**##12'** to '**##16**'.

However, XGBoost is best compared to all other models. The tuned parameters with 5 fold cv specified in the 3.3 section did a better job.

5. CONCLUSION AND SUMMARY

Regression is one of the chosen methods to solve the flight delay prediction problem. We can also go for classification as"delay" or "not delay" by choosing a threshold delay time. Or as multiclass classification with bin of delay values. However, for regression approach XGBoost has very less RMSE and it is the best model. XGB also performed better on the hold-out set. The following features are important according to the XGB. This can be seen in '##16' section of the code.



Scheduled departure time, dew, pressure, dayofyear, humidity and wind direction all seem to have a significant effect on flight delay prediction.

We also improved our prediction scores over the baseline model using the XGBoost model. Although bagging techniques like Random Forest came close, GBM and XGBoost improved the baseline model the most.

6. REFERENCES

MAE vs RMSE

Lasso Regression - R Statistics Blog

Ridge Regression - R Statistics Blog

Beginners Tutorial on XGBoost and Parameter Tuning in R Tutorials & Notes | Machine Learning | HackerEarth

Random Forests · UC Business Analytics R Programming Guide

7. APPENDIX

- Note: Each code section is numbered as ##1, ##2 … ## 23.
- There are a total of 23 coding sections.
- Appendix has all the code, the output of the code and all other plots.

krishna

December 6, 2019

#Flight delay prediction >(STAT652) Statistical Learning and Prediction Final Project

Krishna Chaitanya Gopaluni

R version 3.6.1 (2019-07-05) x86_64-pc-linux-gnu

List of packages to be installed are in section 1

##1. Install the packages and load all the libraries

[0]:
```
list.of.packages <- c("ggplot2", "tidyverse", "nycflights13", "FeatureHashing",␣
↪"mltools", "rsample", "xgboost", "gbm", "caret", "randomForest", "lars",␣
↪"vtreat", "Metrics", "gam", "hashFunction", "glmnet", "vtreat", "xgboost")
new.packages <- list.of.packages[!(list.of.packages %in% installed.
↪packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)
```

[0]:
```
library(tidyverse)
library(nycflights13)
library(rsample)
library('Metrics')
library(gam)
library(hashFunction)
library(randomForest)
library(glmnet)
library(vtreat)
library(xgboost)
library(gbm)
```

##2. Read the flight dataset and check the dimentions

[0]:
```
flight_train <- read_csv("https://github.com/SFUStatgen/SFUStat452/raw/master/
↪Project652/fltrain.csv.gz")
```

[0]:
```
dim(flight_train)# rows  and cols
str(flight_train) # get the column names and type
```

##3.Checking the amount of missing values in each column and drop the rows/cols which has more missing values

[0]:
```
colMeans(is.na(flight_train))
```

[0]:
```
fl <- flight_train
#Drop columns where there are more than 12.5% missing values
fl <- fl[, -which(colMeans(is.na(flight_train)) > 0.125)]
#Next, Drop the rows which has the NA entries
fl <- na.omit(fl)
```

[0]:
```
#dimentions and the summary of the  dataset
dim(fl)
summary(fl)
```

1. 165021 2. 34

```
     year.x         month             day           dep_time     sched_dep_time
 Min.   :2013   Min.   : 1.000   Min.   : 1.00   Min.   :   1   Min.   : 500
 1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00   1st Qu.: 917   1st Qu.: 915
 Median :2013   Median : 7.000   Median :16.00   Median :1421   Median :1411
 Mean   :2013   Mean   : 6.557   Mean   :15.74   Mean   :1359   Mean   :1349
 3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:1750   3rd Qu.:1730
 Max.   :2013   Max.   :12.000   Max.   :31.00   Max.   :2400   Max.   :2339
   dep_delay          arr_time    sched_arr_time   arr_delay
 Min.   : -43.00   Min.   :   1   Min.   :   1   Min.   : -75.000
 1st Qu.:  -5.00   1st Qu.:1110   1st Qu.:1128   1st Qu.: -17.000
 Median :  -2.00   Median :1557   Median :1612   Median :  -6.000
 Mean   :  11.19   Mean   :1521   Mean   :1551   Mean   :   4.878
 3rd Qu.:   9.00   3rd Qu.:1949   3rd Qu.:1953   3rd Qu.:  12.000
 Max.   :1301.00   Max.   :2400   Max.   :2359   Max.   :1272.000
   carrier              flight       tailnum             origin
 Length:165021      Min.   :   1   Length:165021      Length:165021
 Class :character   1st Qu.: 545   Class :character   Class :character
 Mode  :character   Median :1506   Mode  :character   Mode  :character
                    Mean   :1971
                    3rd Qu.:3459
                    Max.   :6181
     dest              air_time      distance         hour
 Length:165021      Min.   : 20   Min.   :  80   Min.   : 5.00
 Class :character   1st Qu.: 81   1st Qu.: 502   1st Qu.: 9.00
 Mode  :character   Median :127   Median : 828   Median :14.00
                    Mean   :149   Mean   :1032   Mean   :13.23
                    3rd Qu.:183   3rd Qu.:1372   3rd Qu.:17.00
                    Max.   :695   Max.   :4983   Max.   :23.00
     minute        time_hour                          temp             dewp
 Min.   : 0.00   Min.   :2013-01-01 10:00:00   Min.   : 10.94   Min.   :-9.94
 1st Qu.: 8.00   1st Qu.:2013-04-04 14:00:00   1st Qu.: 42.08   1st Qu.:24.98
 Median :29.00   Median :2013-07-06 14:00:00   Median : 57.02   Median :41.00
 Mean   :26.09   Mean   :2013-07-03 16:35:37   Mean   : 56.94   Mean   :40.22
 3rd Qu.:43.00   3rd Qu.:2013-10-01 14:00:00   3rd Qu.: 71.96   3rd Qu.:57.02
```

```
 Max.   :59.00   Max.   :2013-12-30 23:00:00   Max.   :100.04   Max.   :78.08
     humid           wind_dir     wind_speed         precip
 Min.   : 13.00   Min.   :  0   Min.   : 0.000   Min.   :0.000000
 1st Qu.: 42.30   1st Qu.:140   1st Qu.: 6.905   1st Qu.:0.000000
 Median : 54.22   Median :230   Median :10.357   Median :0.000000
 Mean   : 56.08   Mean   :205   Mean   :11.093   Mean   :0.001391
 3rd Qu.: 69.28   3rd Qu.:290   3rd Qu.:14.960   3rd Qu.:0.000000
 Max.   :100.00   Max.   :360   Max.   :39.127   Max.   :0.530000
    pressure         visib            name                lat
 Min.   : 985   Min.   : 0.000   Length:165021      Min.   :21.32
 1st Qu.:1013   1st Qu.:10.000   Class :character   1st Qu.:32.90
 Median :1018   Median :10.000   Mode  :character   Median :36.08
 Mean   :1018   Mean   : 9.587                      Mean   :35.98
 3rd Qu.:1023   3rd Qu.:10.000                      3rd Qu.:41.41
 Max.   :1042   Max.   :10.000                      Max.   :61.17
      lon               alt               tz              dst
 Min.   :-157.92   Min.   :   3.0   Min.   :-10.000   Length:165021
 1st Qu.: -95.34   1st Qu.:  26.0   1st Qu.: -6.000   Class :character
 Median : -83.35   Median : 433.0   Median : -5.000   Mode  :character
 Mean   : -89.54   Mean   : 581.8   Mean   : -5.754
 3rd Qu.: -80.15   3rd Qu.: 748.0   3rd Qu.: -5.000
 Max.   : -68.83   Max.   :6602.0   Max.   : -5.000
    tzone
 Length:165021
 Class :character
 Mode  :character
```

##4.FetureHashing the categorical columns

[0]:
```
for(i in 1:ncol(fl)) {
  if(typeof(fl[[i]]) == "character") {
    fl[[i]] <- as.numeric(unlist(lapply(fl[[i]], murmur3.32)))

  }
}
```

[0]:
```
unique(flight_train$carrier)
unique(fl$carrier)
```

##5. Assumptions to drop other columns

Following columns were removed based on these assumtions

- year.x is constant and all the datapoints are from year 2013.
- month, day, hour, minute were removed because this information is present in time_hour

column.
- tzone, dst, tz were removed because these are the constant values for all the observations.
- name full name of destination airport which we have already in dst column
- tailnum, flight are just numeric representation and we can drop them.
- dep_tim, arr_time, arr_delay, air_time will not be known in advance and not needed for the models.

[0]:
```
fl <- as.data.frame(subset(fl, select=-c(year.x, month, day, hour, minute,␣
↪tzone, dst, tz, name, tailnum, flight, dep_time, arr_time, arr_delay,␣
↪air_time)))
```

[0]:
```
fl %>% head(10)
```

##6. Summarizing the target variable 'dep_delay' using quantiles.

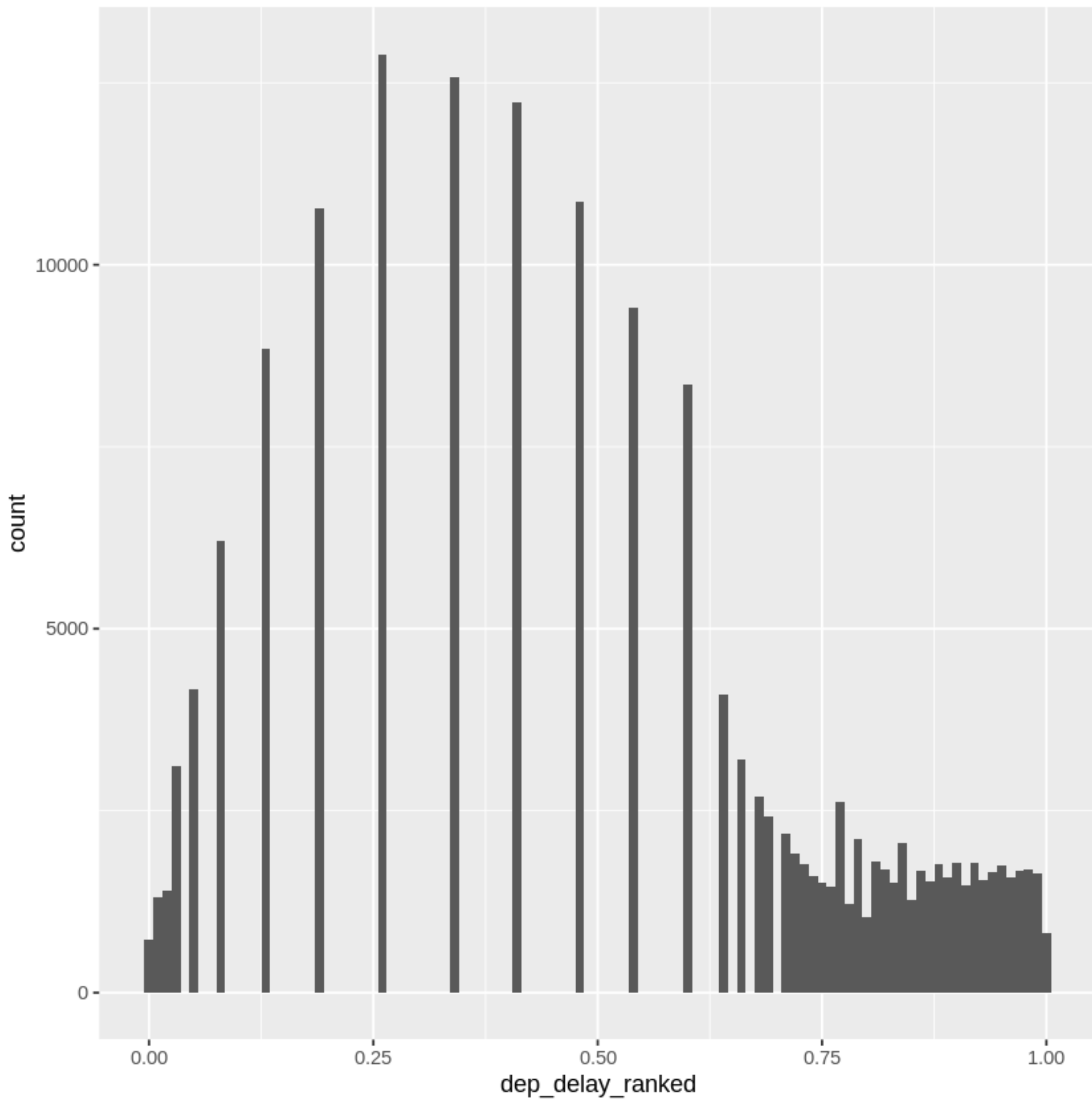
[0]:
```
quantile(fl$dep_delay, probs = c(0.01,0.05,0.1,0.25,.5,.75,.90,.95,.99))
```

Top 10 delayed flightdelays

[0]:
```
fl%>% arrange(desc(dep_delay)) %>% head(10)
```

##7. Density plot of response variable

[0]:
```
den <- nrow(fl)+1 # to avoid truncate of last value during ranking
fl <- fl %>% mutate(dep_delay_ranked = rank(dep_delay)/den) # as the dep_delay␣
↪values are skewed, create new column with scaled and ranked dep_delay col
ggplot(fl,aes(x=dep_delay_ranked)) + geom_histogram(binwidth=.01)
```



##8. Other preprocessing and feature engineering techniques

- Changing ‘pres’ to binary to show the presence of precipitation

```
[0]: fl <- fl %>% mutate(precip = as.numeric(fl$precip > 0))
```

- Bining the `sched_dep_time` and `sched_arr_time` in buckets of 23

```
[0]: fl <- fl %>% mutate(sched_dep_time_bin = as.numeric(floor(fl$sched_dep_time/
     ↪100))) %>% select(-sched_dep_time)

     fl <- fl %>% mutate(sched_arr_time_bin = as.numeric(floor(fl$sched_arr_time/
     ↪100))) %>% select(-sched_arr_time)
```

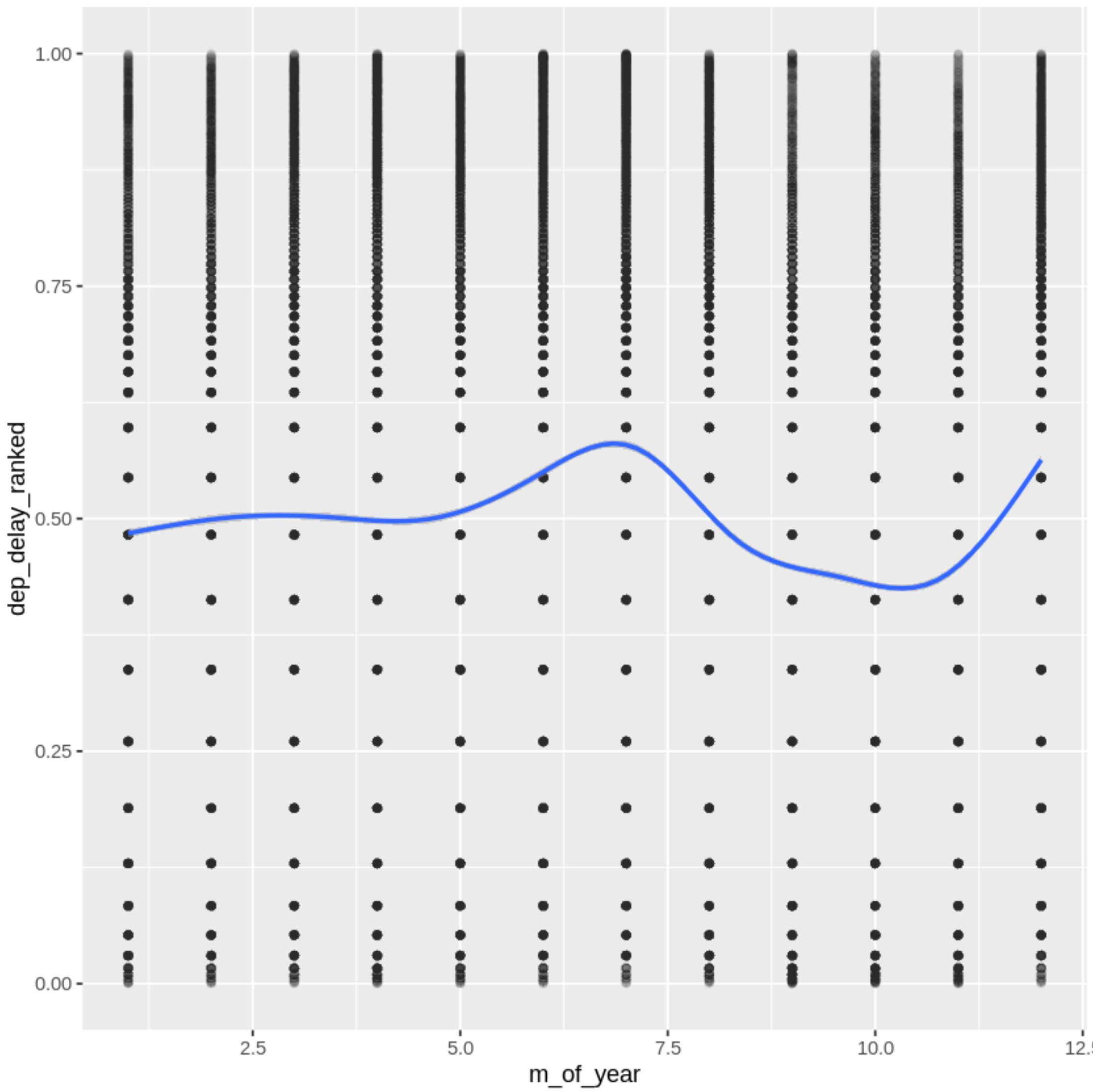
- Calculating the day of year, day of week and month of year from `time_hour`

[0]:
```
fl <- fl %>% mutate(d_of_year = as.numeric(strftime(fl$time_hour, format =␣
↪"%j"))) %>% mutate(d_of_week = as.numeric(strftime(fl$time_hour, format =␣
↪"%w"))) %>% mutate(m_of_year = as.numeric(strftime(fl$time_hour, format =␣
↪"%m"))) %>% select(-time_hour)
```

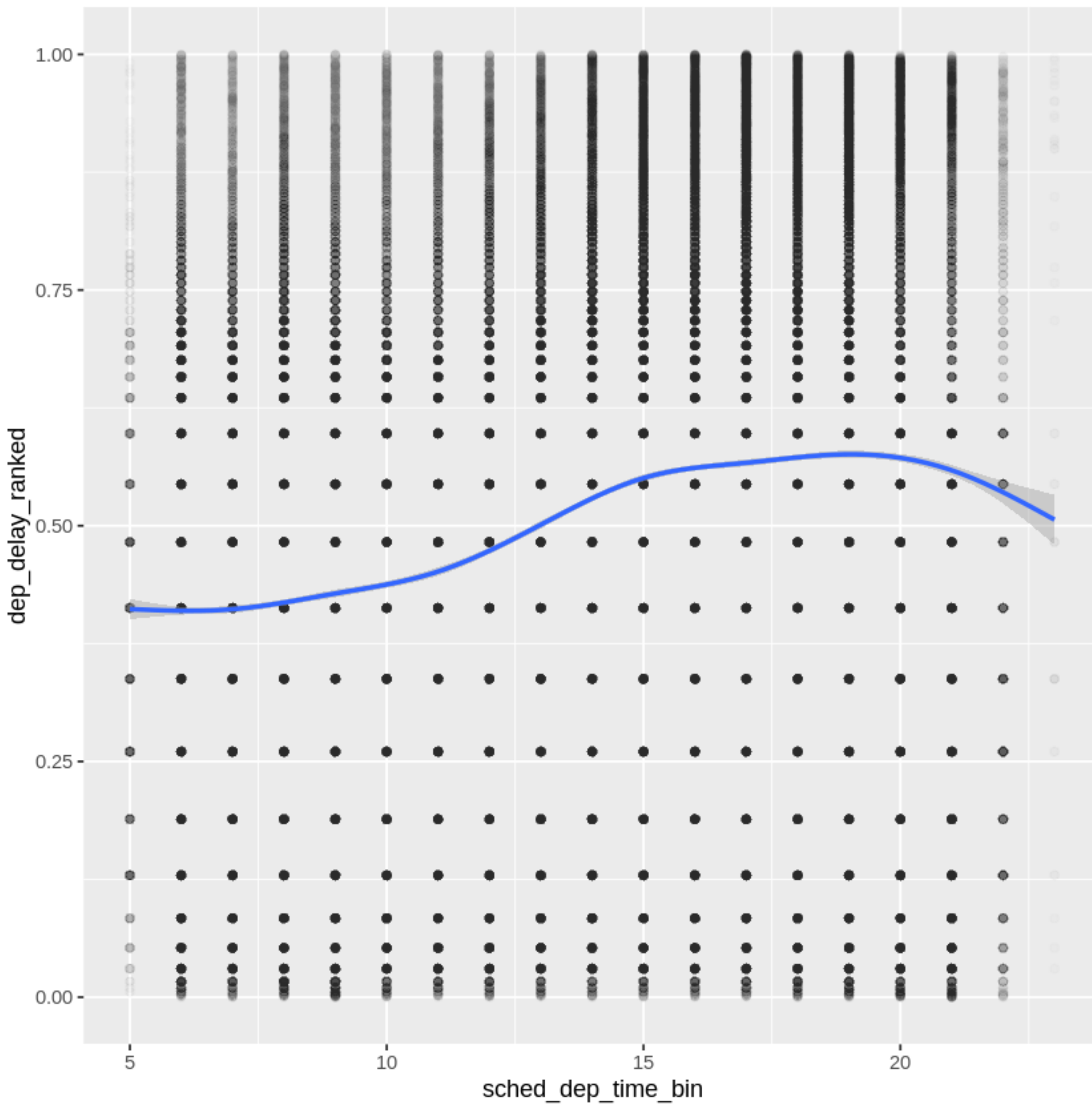
##9. Associations of `dep_delay_ranked` and quantitative predictors

[0]:
```
ggplot(fl,aes(x=m_of_year,y=dep_delay_ranked)) + geom_point(alpha=.01) +␣
↪geom_smooth()
# The relationship shows the the depatures are high in the months of summer
# There's also increase in the departure delays in the month of winter.

ggplot(fl,aes(x=sched_dep_time_bin,y=dep_delay_ranked)) + geom_point(alpha=0.
↪01) + geom_smooth()
# With the increase in the time there is increase in the departure delays.

ggplot(fl,aes(x=distance,y=dep_delay_ranked)) + geom_point(alpha=0.01) +␣
↪geom_smooth()
ggplot(fl,aes(x=log(distance),y=dep_delay_ranked)) + geom_point(alpha=0.01) +␣
↪geom_smooth()
# Departure delays are increasing with distance hence we transform it to log␣
↪distance below
fl <- mutate(fl,logdistance = log(distance)) %>% select(-distance)

ggplot(fl,aes(x=temp,y=dep_delay_ranked)) + geom_point(alpha=0.01) +␣
↪geom_smooth()
# Bigger departure delays when the temperatures are too high or too low
ggplot(fl,aes(x=dewp,y=dep_delay_ranked)) + geom_point(alpha=0.01) +␣
↪geom_smooth()



ggplot(fl,aes(x=alt,y=dep_delay_ranked)) + geom_point(alpha=0.01) +␣
↪geom_smooth()
ggplot(fl,aes(x=log(alt),y=dep_delay_ranked)) + geom_point(alpha=0.01) +␣
↪geom_smooth()
# Similar to distance, replace alt with log(alt)
fl <- mutate(fl,logalt = log(alt)) %>% select(-alt, -dep_delay_ranked)
```

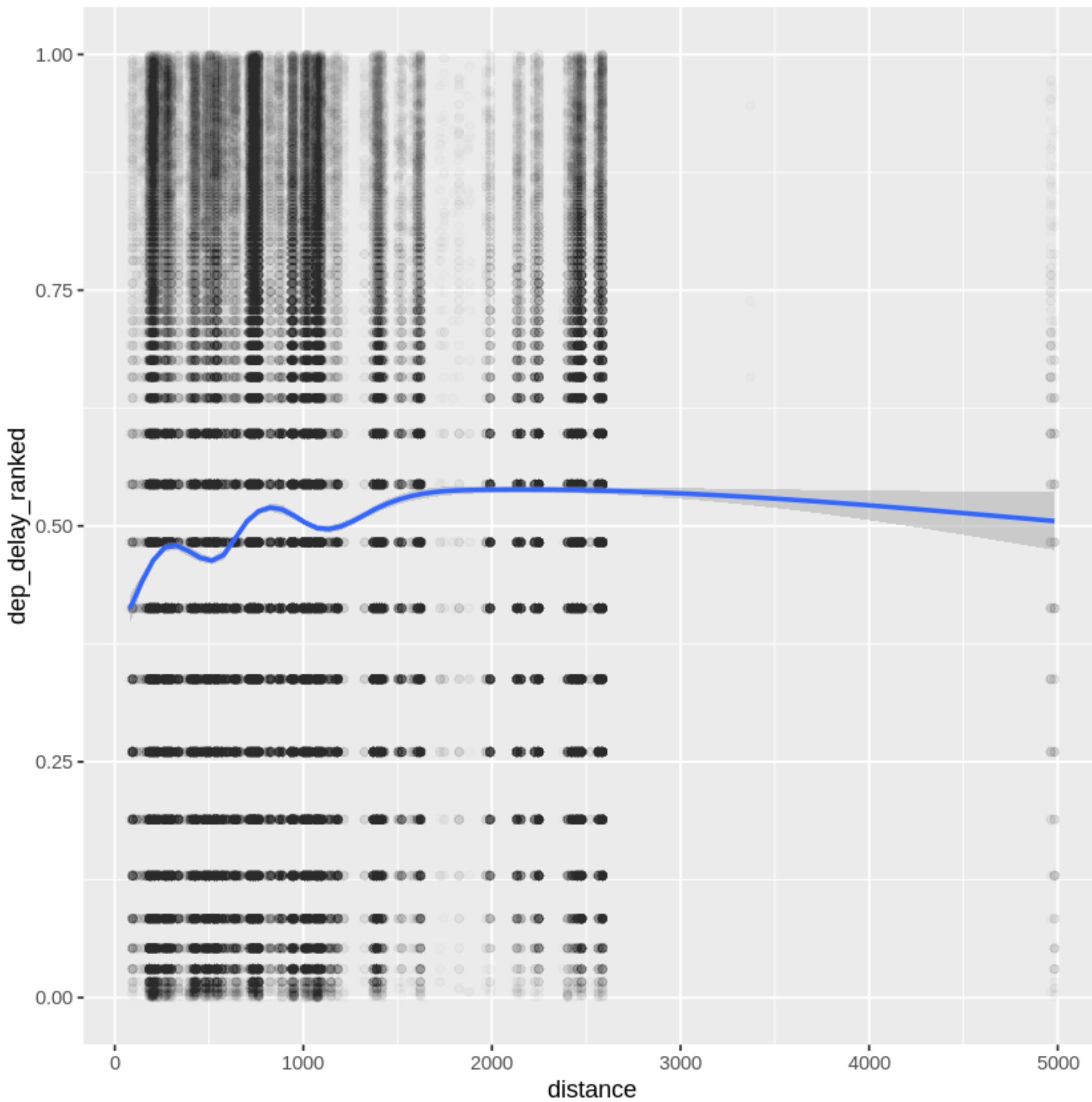
```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



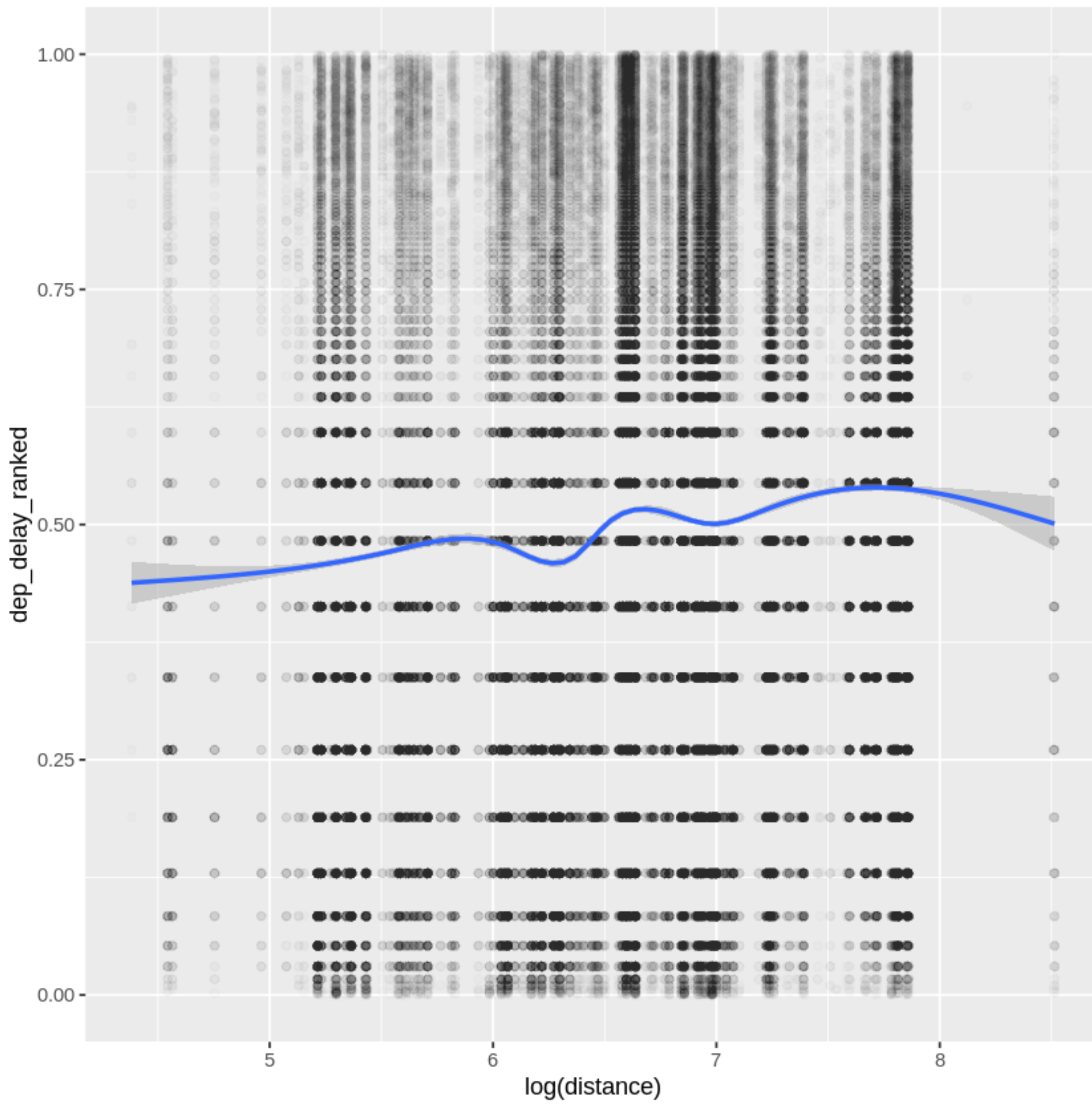

```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



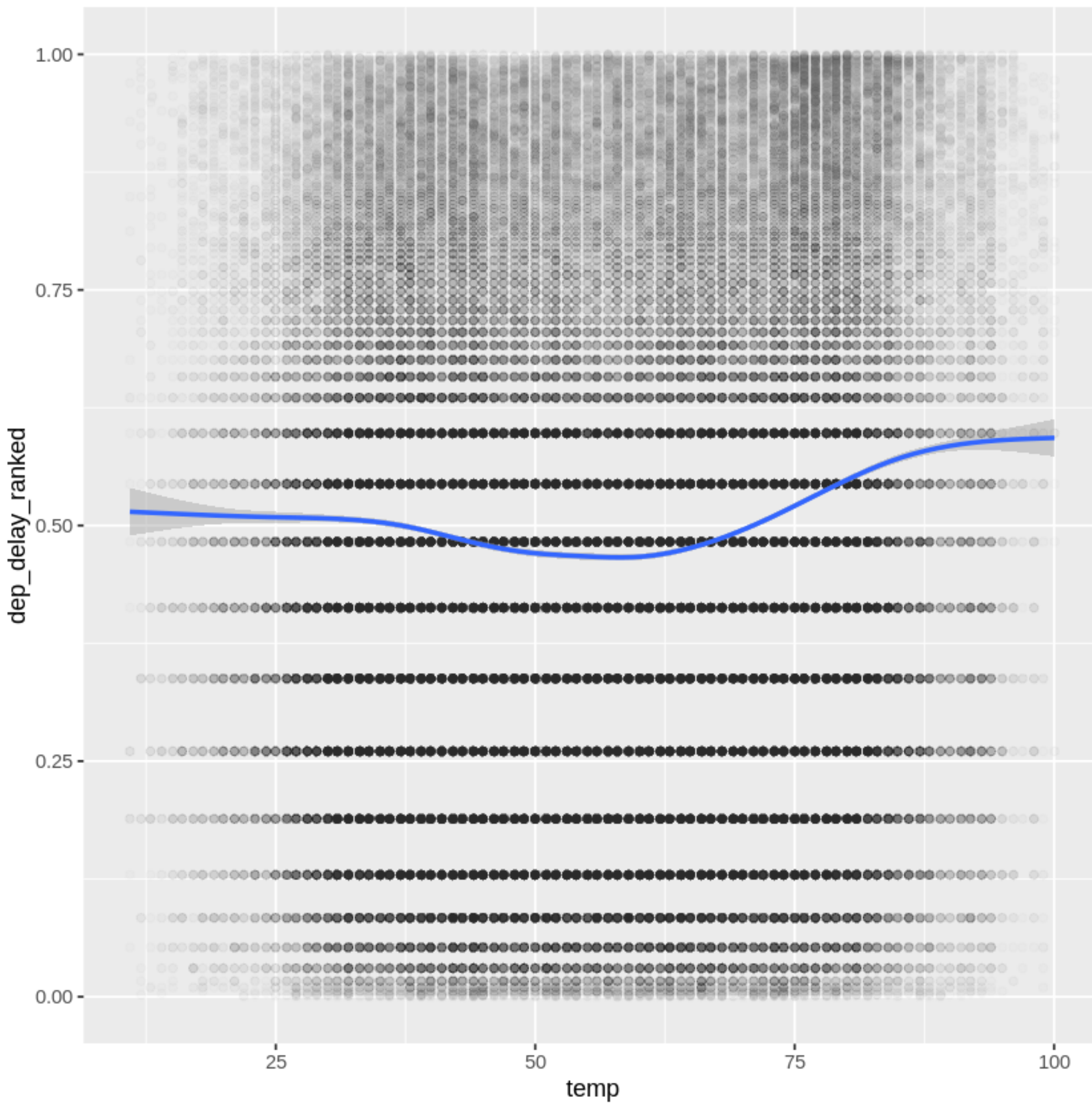

```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



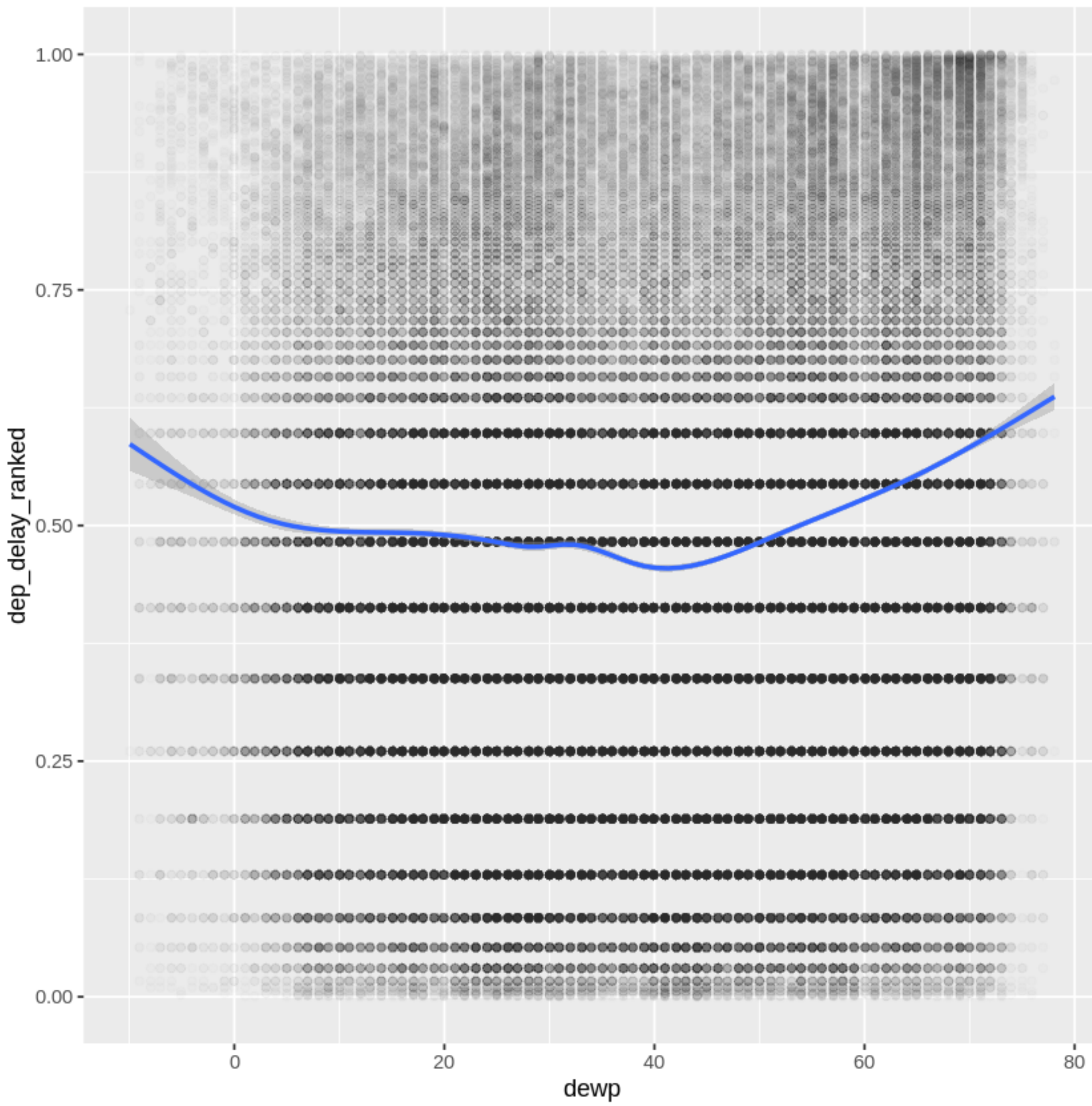

```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



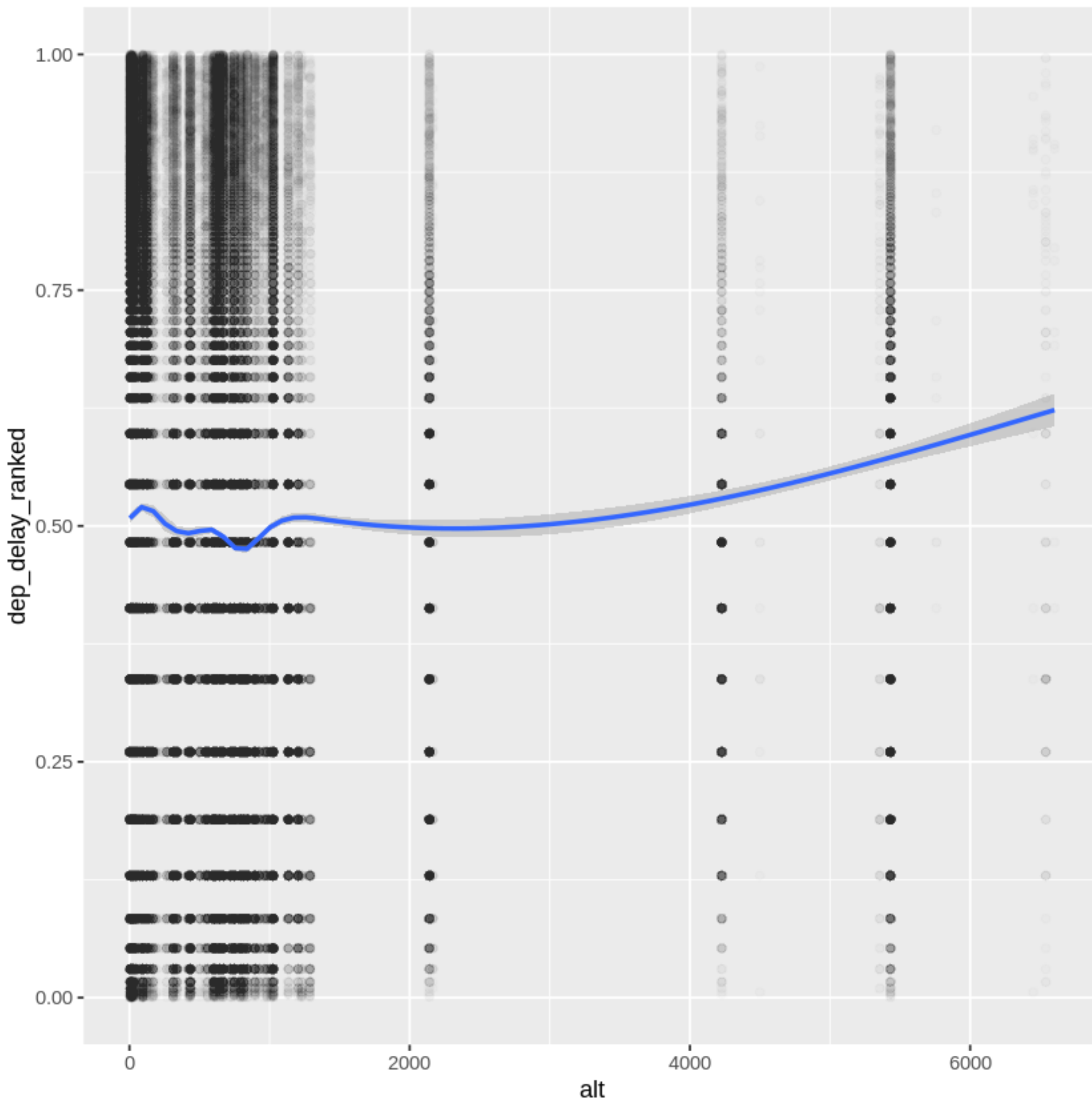

```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

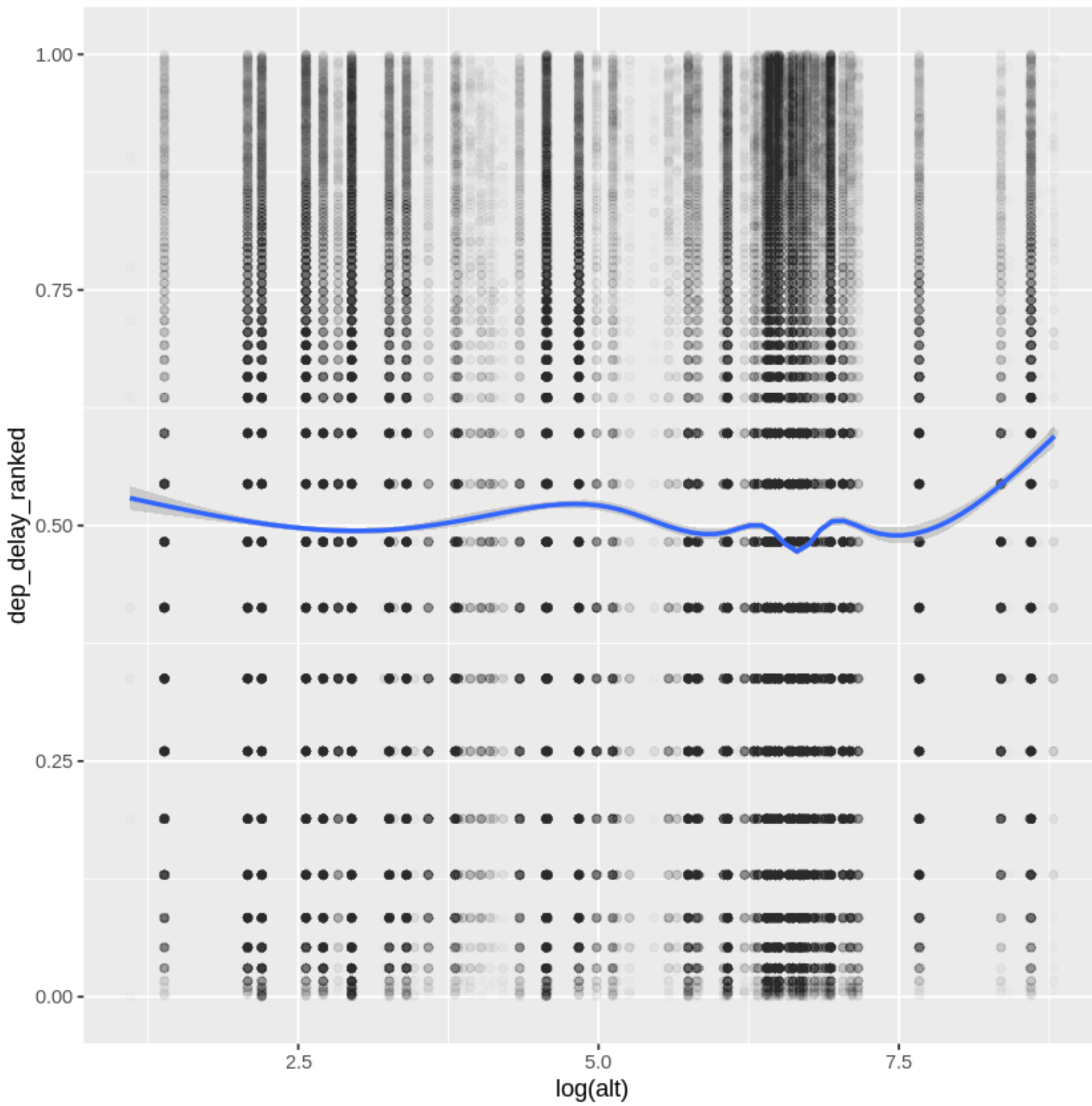



```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```




```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```


1.00
0.75
0.50
0.25
0.00
dep_delay_ranked
0
2000
4000
6000
alt



[0]:
```
summary(fl)
```

```
  dep_delay          carrier              origin
 Min.   : -43.00   Min.   :-2.060e+09   Min.   :-1.124e+09
 1st Qu.:  -5.00   1st Qu.:-6.295e+08   1st Qu.:-1.124e+09
 Median :  -2.00   Median : 7.488e+08   Median : 1.791e+09
 Mean   :  11.19   Mean   : 1.270e+08   Mean   : 9.536e+08
 3rd Qu.:   9.00   3rd Qu.: 8.602e+08   3rd Qu.: 2.135e+09
 Max.   :1301.00   Max.   : 1.771e+09   Max.   : 2.135e+09
      dest                 temp              dewp           humid
 Min.   :-2.070e+09   Min.   : 10.94   Min.   :-9.94   Min.   : 13.00
 1st Qu.:-9.143e+08   1st Qu.: 42.08   1st Qu.:24.98   1st Qu.: 42.30
 Median : 3.952e+08   Median : 57.02   Median :41.00   Median : 54.22
```

```
 Mean   : 9.066e+07   Mean   : 56.94   Mean   :40.22   Mean   : 56.08
 3rd Qu.: 9.192e+08   3rd Qu.: 71.96   3rd Qu.:57.02   3rd Qu.: 69.28
 Max.   : 2.091e+09   Max.   :100.04   Max.   :78.08   Max.   :100.00
    wind_dir      wind_speed         precip           pressure
 Min.   :  0   Min.   : 0.000   Min.   :0.00000   Min.   : 985
 1st Qu.:140   1st Qu.: 6.905   1st Qu.:0.00000   1st Qu.:1013
 Median :230   Median :10.357   Median :0.00000   Median :1018
 Mean   :205   Mean   :11.093   Mean   :0.03158   Mean   :1018
 3rd Qu.:290   3rd Qu.:14.960   3rd Qu.:0.00000   3rd Qu.:1023
 Max.   :360   Max.   :39.127   Max.   :1.00000   Max.   :1042
     visib             lat              lon           sched_dep_time_bin
 Min.   : 0.000   Min.   :21.32   Min.   :-157.92   Min.   : 5.00
 1st Qu.:10.000   1st Qu.:32.90   1st Qu.: -95.34   1st Qu.: 9.00
 Median :10.000   Median :36.08   Median : -83.35   Median :14.00
 Mean   : 9.587   Mean   :35.98   Mean   : -89.54   Mean   :13.23
 3rd Qu.:10.000   3rd Qu.:41.41   3rd Qu.: -80.15   3rd Qu.:17.00
 Max.   :10.000   Max.   :61.17   Max.   : -68.83   Max.   :23.00
 sched_arr_time_bin   d_of_year        d_of_week        m_of_year
 Min.   : 0.00      Min.   :  1.0   Min.   :0.000   Min.   : 1.00
 1st Qu.:11.00      1st Qu.: 94.0   1st Qu.:1.000   1st Qu.: 4.00
 Median :16.00      Median :187.0   Median :3.000   Median : 7.00
 Mean   :15.22      Mean   :184.1   Mean   :2.981   Mean   : 6.56
 3rd Qu.:19.00      3rd Qu.:274.0   3rd Qu.:5.000   3rd Qu.:10.00
 Max.   :23.00      Max.   :364.0   Max.   :6.000   Max.   :12.00
  logdistance         logalt
 Min.   :4.382   Min.   :1.099
 1st Qu.:6.219   1st Qu.:3.258
 Median :6.719   Median :6.071
 Mean   :6.671   Mean   :5.163
 3rd Qu.:7.224   3rd Qu.:6.617
 Max.   :8.514   Max.   :8.795
```

##10. Split the test train set using `rsample` library

[0]:
```
set.seed(123)
fl_split <- initial_split(fl, prop = .8)
fl_train <- training(fl_split)
fl_test  <- testing(fl_split)
```

##11. Baseline model and the metrics

[0]:
```
##mean absolute error
mean <- abs(mean(fl_test$dep_delay))
fl_test_dummy <- fl_test %>% mutate(dep_delay_calc = abs(fl_test$dep_delay -
 ↪mean)) %>% select(dep_delay_calc)
mae_dummy <- mean(fl_test_dummy$dep_delay_calc)
mae_dummy
```

```
mse_dummy <- var(fl_test$dep_delay)
mse_dummy#Mean Squared Error
rmse_dummy <- sqrt(var(fl_test$dep_delay))
rmse_dummy#Root Mean Squared Error
```

21.5771679801823

1428.55857201025

37.7962772242221

##12. GAM and its metrics on training and test splits

[0]:
```
gam.fit <- gam(
  formula = dep_delay ~ .,
  family = gaussian,
  data = fl_train
  )
pred_gam <- predict(gam.fit, fl_test)

mae_gam <- mae(fl_test$dep_delay, pred_gam)#mean absolute error for the gam
mae_gam
mse_gam <- mse(fl_test$dep_delay, pred_gam)#Calculate Mean Squared Error (MAE)␣
 ↪for test data
mse_gam
rmse_gam <- rmse(fl_test$dep_delay, pred_gam)# Calculate Root Mean Squared␣
 ↪Error for test data
rmse_gam
```

```
Warning message in model.matrix.default(mt, mf, contrasts):
"non-list contrasts argument ignored"
```

20.3150343558084

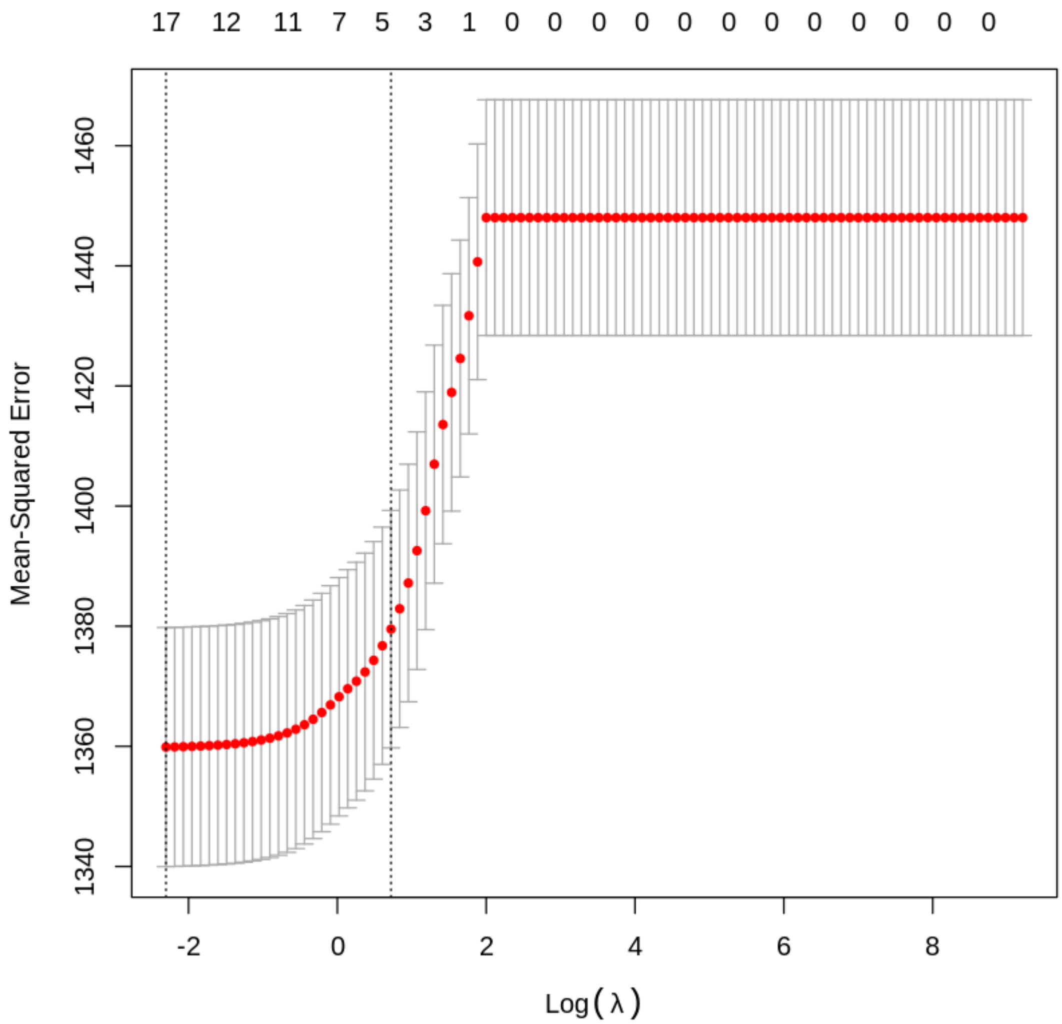
1336.67101795096

36.5605117298837

##13. Implementing Lasso regression and metrics on training and test splits

[0]:
```
X_train <- model.matrix(dep_delay ~., data=fl_train)
```

[0]:
```
lambdas <- 10^{seq(from=-1,to=4,length=100)}
cv.lafit <- cv.glmnet(X_train,fl_train$dep_delay,alpha=1,lambda=lambdas)
plot(cv.lafit)
```



- **Finding the best value of the lambda for lasso regression and calculating metrics**

[0]:
```
best_lam <- cv.lafit$lambda.min
```

[0]:
```
best_lam
```

0.1

[0]:
```
X_test <- model.matrix(dep_delay ~., data=fl_test)
```

[0]:
```
lasso_best <- glmnet(X_train,fl_train$dep_delay, alpha = 1, lambda = best_lam)
pred <- predict(lasso_best, s = best_lam, newx =X_test)
```

[0]:
```
#Calculate Mean Absolute Error (MAE) for test data
mae_lasso <- mae(fl_test$dep_delay, pred)
mae_lasso
#Calculate Mean Squared Error (MAE) for test data
mse_lasso <- mse(fl_test$dep_delay, pred)
mse_lasso
#Calculate Root Mean Squared Error for test data
rmse_lasso <- rmse(fl_test$dep_delay, pred)
rmse_lasso
```

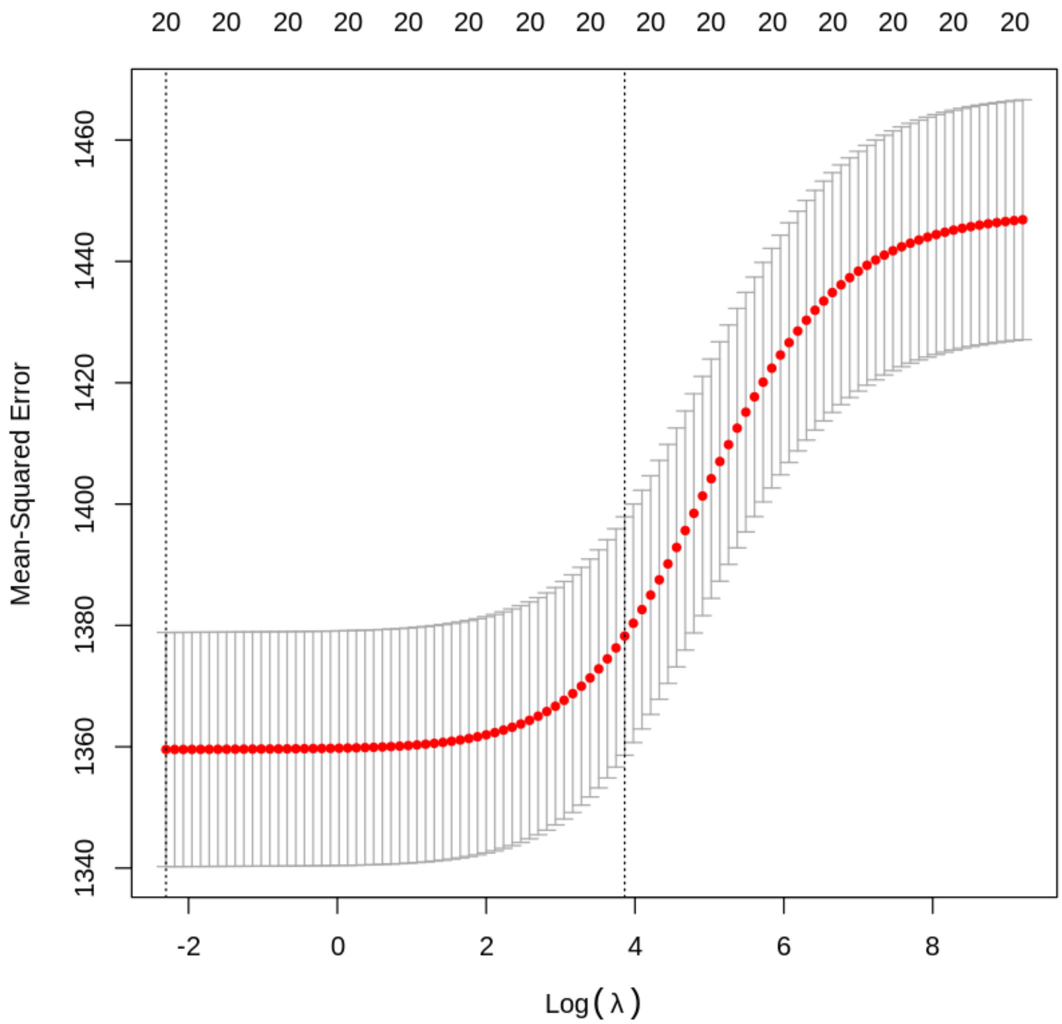
20.3054527014934

1337.78519732319

36.5757460255179

##14. Implementing Ridge regression and metrics on training and test splits

[0]:
```
lambda_seq <- 10^seq(2, -2, by = -.1)
cv.rifit <- cv.glmnet(X_train,fl_train$dep_delay,alpha=0,lambda=lambdas)
best_lam_rid <- cv.rifit$lambda.min
plot(cv.rifit)
```



[0]:
```
best_lam_rid
```

0.1

[0]:
```
ridge_best <- glmnet(X_train,fl_train$dep_delay, alpha = 0, lambda =
    best_lam_rid)
pred_ridge <- predict(ridge_best, s = ridge_best, newx =X_test)
```

[0]:
```
#Calculate Mean Absolute Error (MAE) for test data
mae_ridge <- mae(fl_test$dep_delay, pred_ridge)
mae_ridge
#Calculate Mean Squared Error (MAE) for test data
mse_ridge <- mse(fl_test$dep_delay, pred_ridge)
```

```
mse_ridge
#Calculate Root Mean Squared Error for test data
rmse_ridge <- rmse(fl_test$dep_delay, pred_ridge)
rmse_ridge
```

20.3144807460818

1336.95058308644

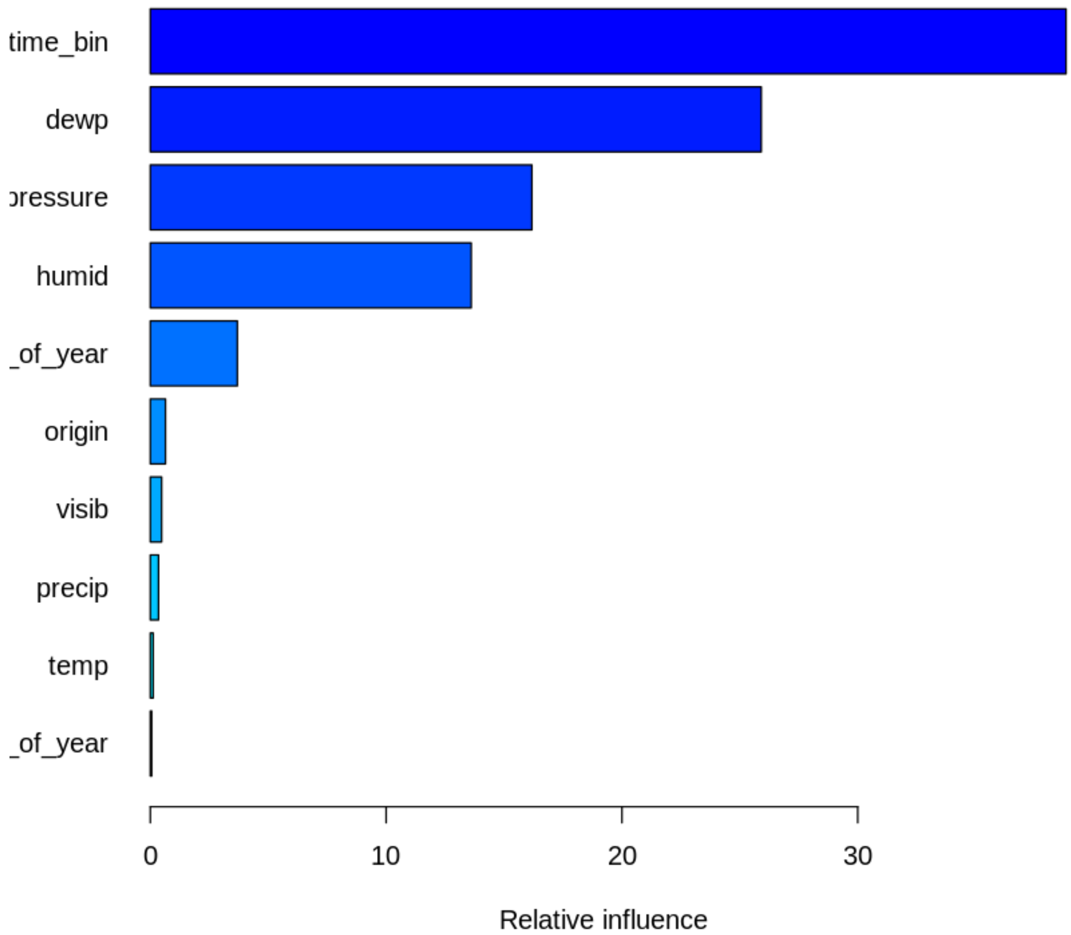
36.5643348508686

##15. Generalized Boosted Regression Model (GBM) and metrics on training and test splits. Feature importance plot included.

[0]:
```
##Fit a GBM model on the train data `fl_train` using 5 fold cross validation
gbm.fit <- gbm(
  formula = dep_delay ~ .,
  distribution = "gaussian",
  data = fl_train,
  n.trees = 2500,
  interaction.depth = 3,
  shrinkage = 0.001,
  cv.folds = 5,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
)

summary.gbm(gbm.fit, n.trees = 2000, cBars=10, plotit=TRUE, normalize=TRUE,
 ↪order=TRUE, las = 1)#Print the feature importance of the fitted GBM model
# sched_dep_time_bin, dewp, pressure, humid are important features with more
 ↪than 10%
# There were 20 predictors of which 15 had non-zero influence.
```

A data.frame: 20 × 2

	var	rel.inf
	<fct>	<dbl>
sched_dep_time_bin	sched_dep_time_bin	38.834055234
dewp	dewp	25.908216630
pressure	pressure	16.182219023
humid	humid	13.608225825
d_of_year	d_of_year	3.691197376
origin	origin	0.643414692
visib	visib	0.479724168
precip	precip	0.348151254
temp	temp	0.123262539
m_of_year	m_of_year	0.057549679
carrier	carrier	0.051628517
wind_dir	wind_dir	0.038027947
d_of_week	d_of_week	0.018250607
sched_arr_time_bin	sched_arr_time_bin	0.012070120
logalt	logalt	0.004006387
dest	dest	0.000000000
wind_speed	wind_speed	0.000000000
lat	lat	0.000000000
lon	lon	0.000000000
logdistance	logdistance	0.000000000



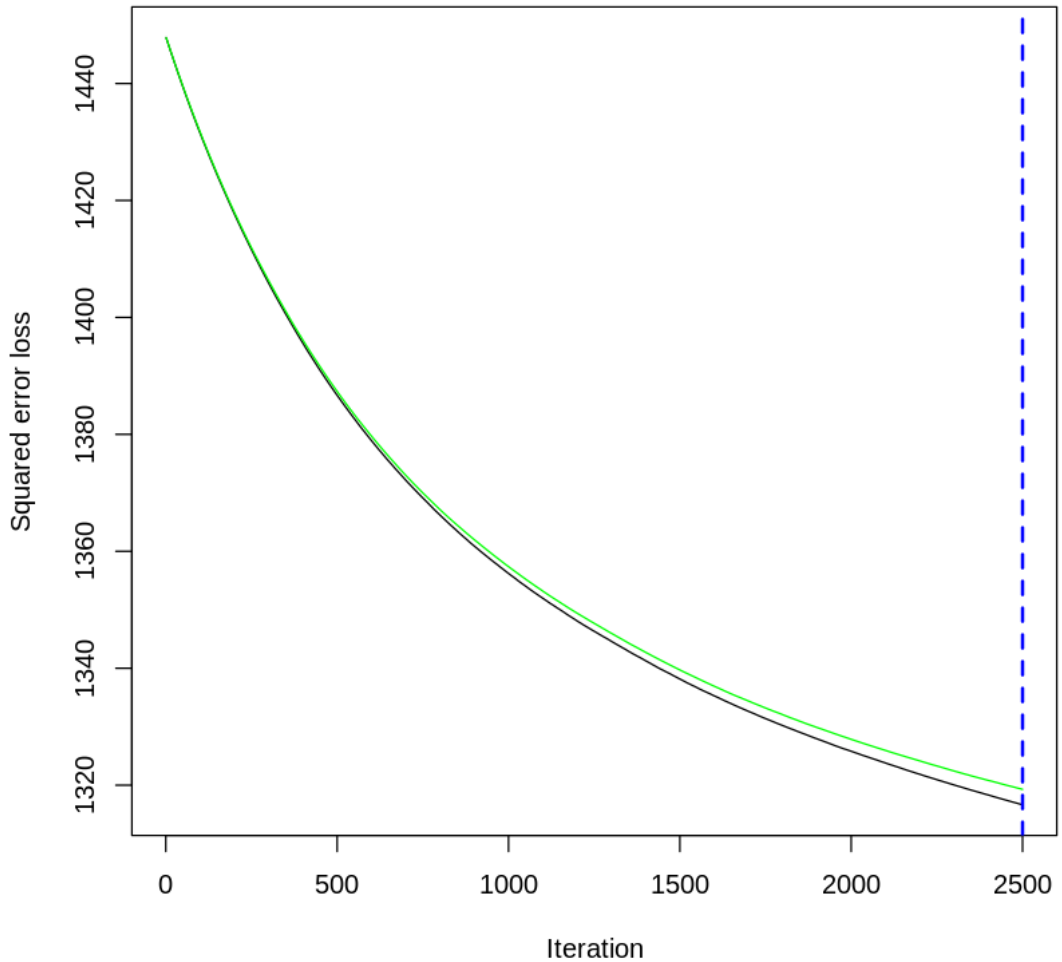
- Calculate RMSE of CV Error

[0]:
```
sqrt(min(gbm.fit$cv.error))
```

36.3221498319001

[0]:
```
gbm.perf(gbm.fit, method = "cv")
```

2500



- **GBM Matrices on predictions**

[0]:
```
pred_gbm <- predict(gbm.fit, n.trees = 2500, fl_test)#Predict the target␣
↪`dep_delay` on `fl_test`
mae_gbm <- mae(fl_test$dep_delay, pred_gbm) #Calculate Mean Absolute Error␣
↪(MAE) for test data
mae_gbm
mse_gbm <- mse(fl_test$dep_delay, pred_gbm)#Calculate Mean Squared Error (MAE)␣
↪for test data
mse_gbm
rmse_gbm <- rmse(fl_test$dep_delay, pred_gbm)#Calculate Root Mean Squared Error␣
↪for test data
```

```
rmse_gbm
```

19.9555912883

1295.98946271989

35.9998536485899

##16. EXtreme Gradient Boosting Training (XGBoost) and metrics on training and test splits. Feature importance graphs included.

- **Prepare the data for XGBoost algorithm**

[0]:
```
# variable names
features <- setdiff(names(fl_train), "dep_delay")

# Create the treatment plan from the training data
treatplan <- vtreat::designTreatmentsZ(fl_train, features, verbose = FALSE)
# Prepare the training data
features_train <- vtreat::prepare(treatplan, fl_train) %>% as.matrix()
response_train <- fl_train$dep_delay

# Prepare the test data
features_test <- vtreat::prepare(treatplan, fl_test) %>% as.matrix()
response_test <- fl_test$dep_delay
```

- **Fit the XGB on the train data**

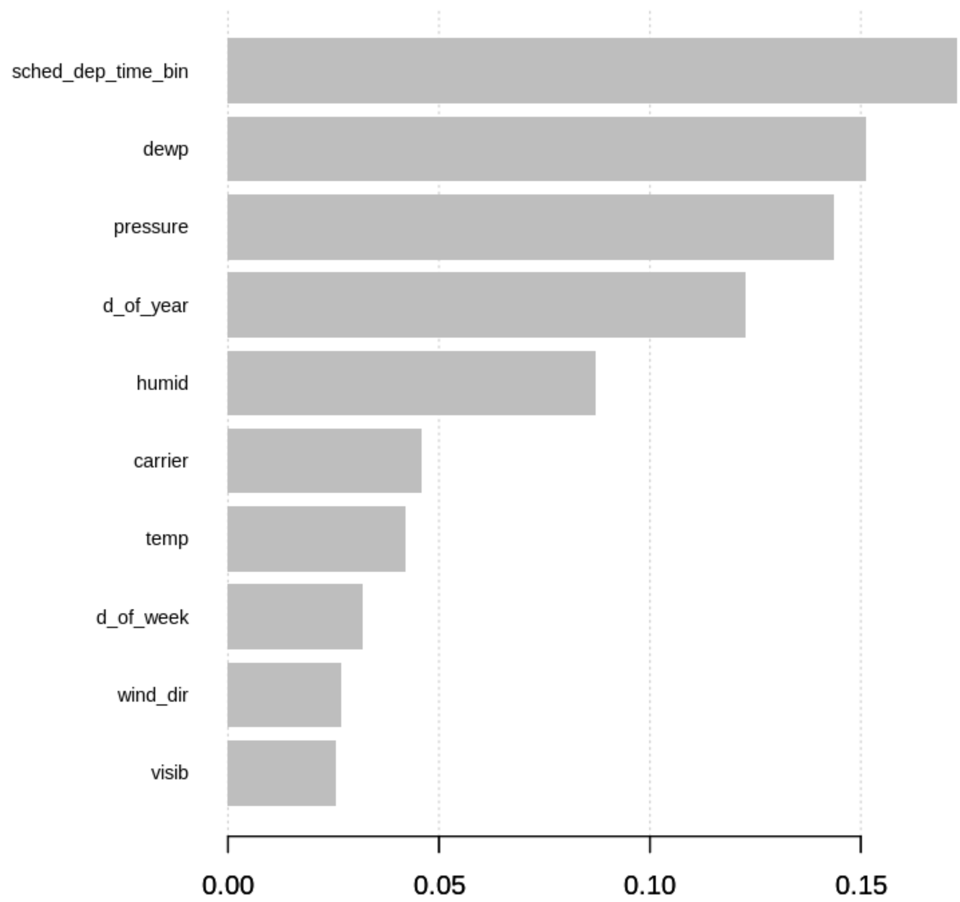
[0]:
```
set.seed(124)
params <- list(
  eta = 0.01,
  max_depth = 5,
  min_child_weight = 5,
  subsample = 0.65,
  colsample_bytree = 1
)
xgb.fit <- xgboost(
  params = params,
  data = features_train,
  label = response_train,
  nrounds = 1000,
  nfold = 5,
  objective = "reg:linear",  # for regression models
  verbose = 0,               # silent
  early_stopping_rounds = 10 # tells XgBoost to stop if there's no improvement␣
↪for 10 consecutive trees
)
```

- **Importance matrix**

[0]:
```
importance_matrix <- xgb.importance(model = xgb.fit)
importance_matrix
```

- **Featire importances plots**

[0]:
```
xgb.plot.importance(importance_matrix, top_n = 10, measure = "Gain") # using
↪information gain
```



[0]:
```
pred_xgb <- predict(xgb.fit, features_test)
```

- XGBoost Matrices

[0]:
```
mae_xgb <- mae(fl_test$dep_delay, pred_xgb)
mae_xgb

mse_xgb <- mse(fl_test$dep_delay, pred_xgb)#Calculate Mean Squared Error (MAE)␣
↪for test data
mse_xgb

rmse_xgb <- rmse(fl_test$dep_delay, pred_xgb)#Calculate Root Mean Squared Error␣
↪for test data
rmse_xgb
```

18.3803578341618

1169.54803738023

34.1986554908265

##17. XGB metrics on the new "fltest.csv.gz" test data

- **Preprocess the new test data**

[0]:
```
flight_Newtest <- read_csv("https://github.com/SFUStatgen/SFUStat452/raw/master/
↪Project652/fltest.csv.gz")

flTest <- flight_Newtest
#Drop columns where there are more than 12.5% missing values
flTest <- flTest[, -which(colMeans(is.na(flight_Newtest)) > 0.125)]
#Next, Drop the rows which has the NA entries
flTest <- na.omit(flTest)

for(i in 1:ncol(flTest)) {
  if(typeof(flTest[[i]]) == "character") {
    flTest[[i]] <- as.numeric(unlist(lapply(flTest[[i]], murmur3.32)))

  }
}

unique(flight_Newtest$carrier)
unique(flTest$carrier)

flTest <- as.data.frame(subset(flTest, select=-c(year.x, month, day, hour,␣
↪minute, tzone, dst, tz, name, tailnum, flight, dep_time, arr_time,␣
↪arr_delay, air_time)))
#quantile(fl$dep_delay, probs = c(0.01,0.05,0.1,0.25,.5,.75,.90,.95,.99))
den <- nrow(flTest)+1 # to avoid truncate of last value during ranking
flTest <- flTest %>% mutate(dep_delay_ranked = rank(dep_delay)/den)
flTest <- flTest %>% mutate(precip = as.numeric(flTest$precip > 0))
```

```
flTest <- flTest %>% mutate(sched_dep_time_bin = as.
↪numeric(floor(flTest$sched_dep_time/100))) %>% select(-sched_dep_time)

flTest <- flTest %>% mutate(sched_arr_time_bin = as.
↪numeric(floor(flTest$sched_arr_time/100))) %>% select(-sched_arr_time)

flTest <- flTest %>% mutate(d_of_year = as.numeric(strftime(flTest$time_hour,␣
↪format = "%j"))) %>% mutate(d_of_week = as.
↪numeric(strftime(flTest$time_hour, format = "%w"))) %>% mutate(m_of_year =␣
↪as.numeric(strftime(flTest$time_hour, format = "%m"))) %>% select(-time_hour)

flTest <- mutate(flTest,logdistance = log(distance)) %>% select(-distance)
flTest <- mutate(flTest,logalt = log(alt)) %>% select(-alt, -dep_delay_ranked)
```

- **Load the entire training data to retrain the models**

[0]:
```
flight_train <- read_csv("https://github.com/SFUStatgen/SFUStat452/raw/master/
↪Project652/fltrain.csv.gz")

fl <- flight_train
#Drop columns where there are more than 12.5% missing values
fl <- fl[, -which(colMeans(is.na(flight_train)) > 0.125)]
#Next, Drop the rows which has the NA entries
fl <- na.omit(fl)

for(i in 1:ncol(fl)) {
  if(typeof(fl[[i]]) == "character") {
    fl[[i]] <- as.numeric(unlist(lapply(fl[[i]], murmur3.32)))

  }
}



unique(flight_train$carrier)
unique(fl$carrier)

fl <- as.data.frame(subset(fl, select=-c(year.x, month, day, hour, minute,␣
↪tzone, dst, tz, name, tailnum, flight, dep_time, arr_time, arr_delay,␣
↪air_time)))
#quantile(fl$dep_delay, probs = c(0.01,0.05,0.1,0.25,.5,.75,.90,.95,.99))
den <- nrow(fl)+1 # to avoid truncate of last value during ranking
fl <- fl %>% mutate(dep_delay_ranked = rank(dep_delay)/den)
fl <- fl %>% mutate(precip = as.numeric(fl$precip > 0))

fl <- fl %>% mutate(sched_dep_time_bin = as.numeric(floor(fl$sched_dep_time/
↪100))) %>% select(-sched_dep_time)
```

```
fl <- fl %>% mutate(sched_arr_time_bin = as.numeric(floor(fl$sched_arr_time/
↪100))) %>% select(-sched_arr_time)

fl <- fl %>% mutate(d_of_year = as.numeric(strftime(fl$time_hour, format =␣
↪"%j"))) %>% mutate(d_of_week = as.numeric(strftime(fl$time_hour, format =␣
↪"%w"))) %>% mutate(m_of_year = as.numeric(strftime(fl$time_hour, format =␣
↪"%m"))) %>% select(-time_hour)

fl <- mutate(fl,logdistance = log(distance)) %>% select(-distance)
fl <- mutate(fl,logalt = log(alt)) %>% select(-alt, -dep_delay_ranked)
```

[0]:
```
fl_train <- fl
dim(fl_train)
```

[0]:
```
# variable names
features <- setdiff(names(fl_train), "dep_delay")

# Create the treatment plan from the training data
treatplan <- vtreat::designTreatmentsZ(fl_train, features, verbose = FALSE)
# Prepare the training data
features_train <- vtreat::prepare(treatplan, fl_train) %>% as.matrix()
response_train <- fl_train$dep_delay

# Prepare the test data
# features_test <- vtreat::prepare(treatplan, fl_test) %>% as.matrix()
# response_test <- fl_test$dep_delay

set.seed(124)
params <- list(
  eta = 0.01,
  max_depth = 5,
  min_child_weight = 5,
  subsample = 0.65,
  colsample_bytree = 1
)
xgb.fit <- xgboost(
  params = params,
  data = features_train,
  label = response_train,
  nrounds = 1000,
  nfold = 5,
  objective = "reg:linear",  # for regression models
  verbose = 0,               # silent
  early_stopping_rounds = 10 # tells XgBoost to stop if there's no improvement␣
↪for 10 consecutive trees
```

```
)
```

- **metrics on new 'fltest' using XGB**

```
[0]: # variable names
features_flTest <- setdiff(names(flTest), "dep_delay")

# Create the treatment plan from the training data
treatplan_flTest <- vtreat::designTreatmentsZ(flTest, features_flTest, verbose␣
↪= FALSE)
# Prepare the training data
features_flTest <- vtreat::prepare(treatplan_flTest, flTest) %>% as.matrix()
response_flTest <- flTest$dep_delay

pred_xgb_flTest <- predict(xgb.fit, features_flTest)

mae_xgb_flTest <- mae(response_flTest, pred_xgb_flTest)
mae_xgb_flTest
mse_xgb_flTest <- mse(response_flTest, pred_xgb_flTest)#Calculate Mean Squared␣
↪Error (MAE) for test data
mse_xgb_flTest
rmse_xgb_flTest <- rmse(response_flTest, pred_xgb_flTest)#Calculate Root Mean␣
↪Squared Error for test data
rmse_xgb_flTest
```

18.170247407622

1149.24307917743

33.9004878899616

##18. Baseline metrics on 'fltest.csv.gz'

```
[0]: mean_test <- abs(mean(flTest$dep_delay))
fl_test_dummy <- flTest %>% mutate(dep_delay_calc = abs(flTest$dep_delay -␣
↪mean_test)) %>% select(dep_delay_calc)
mae_dummy_test <- mean(fl_test_dummy$dep_delay_calc)
mae_dummy_test

mse_dummy_test <- var(flTest$dep_delay)
mse_dummy_test#Mean Squared Error
rmse_dummy_test <- sqrt(var(flTest$dep_delay))
rmse_dummy_test#Root Mean Squared Error
```

21.242544419518

1402.44585700022

37.449243744036

##19. GAM metrics on 'fltest.csv.gz'

[0]:
```
dim(fl_train)
```

1. 165021 2. 21

[0]:
```
gam.fit <- gam(
  formula = dep_delay ~ .,
  family = gaussian,
  data = fl_train
  )
pred_gam_flTest <- predict(gam.fit, flTest)

mae_gam_flTest <- mae(flTest$dep_delay, pred_gam_flTest)#mean absolute error
 ↪for the gam
mae_gam_flTest
mse_gam_flTest <- mse(flTest$dep_delay, pred_gam_flTest)#Calculate Mean Squared
 ↪Error (MAE) for test data
mse_gam_flTest
rmse_gam_flTest <- rmse(flTest$dep_delay, pred_gam_flTest)# Calculate Root Mean
 ↪Squared Error for test data
rmse_gam_flTest
```

```
Warning message in model.matrix.default(mt, mf, contrasts):
"non-list contrasts argument ignored"
```

20.176621642404

1315.89848566412

36.2753151008247

##20. Lasso regression metrics on 'fltest.csv.gz'

[0]:
```
X_train_flTest <- model.matrix(dep_delay ~., data=fl_train)

lambdas <- 10^{seq(from=-1,to=4,length=100)}
cv.lafit <- cv.glmnet(X_train_flTest,fl_train$dep_delay,alpha=1,lambda=lambdas)
# plot(cv.lafit)
best_lam <- cv.lafit$lambda.min

X_test_flTest <- model.matrix(dep_delay ~., data=flTest)

#retrain with best lambda
lasso_best <- glmnet(X_train_flTest,fl_train$dep_delay, alpha = 1, lambda =
 ↪best_lam)
pred <- predict(lasso_best, s = best_lam, newx =X_test_flTest)
```

```
#Calculate Mean Absolute Error (MAE) for test data
mae_lasso <- mae(flTest$dep_delay, pred)
mae_lasso
#Calculate Mean Squared Error (MAE) for test data
mse_lasso <- mse(flTest$dep_delay, pred)
mse_lasso
#Calculate Root Mean Squared Error for test data
rmse_lasso <- rmse(flTest$dep_delay, pred)
rmse_lasso
```

20.1603602978611

1316.30770042231

36.2809550649141

##21. Ridge regression metrics on 'fltest.csv.gz'

[0]:
```
# X_train_flTest <- model.matrix(dep_delay ~., data=fl_train)

lambdas <- 10^seq(2, -2, by = -.1)
cv.rifit <- cv.glmnet(X_train_flTest,fl_train$dep_delay,alpha=0,lambda=lambdas)
# plot(cv.lafit)
best_lam <- cv.rifit$lambda.min

# X_test_flTest <- model.matrix(dep_delay ~., data=flTest)

#retrain with best lambda
ridge_best <- glmnet(X_train_flTest,fl_train$dep_delay, alpha = 0, lambda =
 ↪best_lam)
pred <- predict(ridge_best, s = best_lam, newx =X_test_flTest)


#Calculate Mean Absolute Error (MAE) for test data
mae_ridge <- mae(flTest$dep_delay, pred)
mae_ridge
#Calculate Mean Squared Error (MAE) for test data
mse_ridge <- mse(flTest$dep_delay, pred)
mse_ridge
#Calculate Root Mean Squared Error for test data
rmse_ridge <- rmse(flTest$dep_delay, pred)
rmse_ridge
```

20.1755791792041

1315.94382592453

36.2759400419138

##22. GBM metrics on 'fltest.csv.gz'

[0]:
```
gbm.fit <- gbm(
  formula = dep_delay ~ .,
  distribution = "gaussian",
  data = fl_train,
  n.trees = 2500,
  interaction.depth = 3,
  shrinkage = 0.001,
  cv.folds = 5,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
)

# summary.gbm(gbm.fit, n.trees = 2000, cBars=10, plotit=TRUE, normalize=TRUE,
 ↪order=TRUE, las = 1)#Print the feature importance of the fitted GBM model
# sched_dep_time_bin, dewp, pressure, humid are important features with more
 ↪than 10%
# There were 20 predictors of which 15 had non-zero influence.




sqrt(min(gbm.fit$cv.error))# rmse of cv error
gbm.perf(gbm.fit, method = "cv")



pred_gbm <- predict(gbm.fit, n.trees = 2500, flTest)#Predict the target
 ↪`dep_delay` on `fl_test`
mae_gbm <- mae(flTest$dep_delay, pred_gbm) #Calculate Mean Absolute Error (MAE)
 ↪for test data
mae_gbm
mse_gbm <- mse(flTest$dep_delay, pred_gbm)#Calculate Mean Squared Error (MAE)
 ↪for test data
mse_gbm
rmse_gbm <- rmse(flTest$dep_delay, pred_gbm)#Calculate Root Mean Squared Error
 ↪for test data
rmse_gbm
```

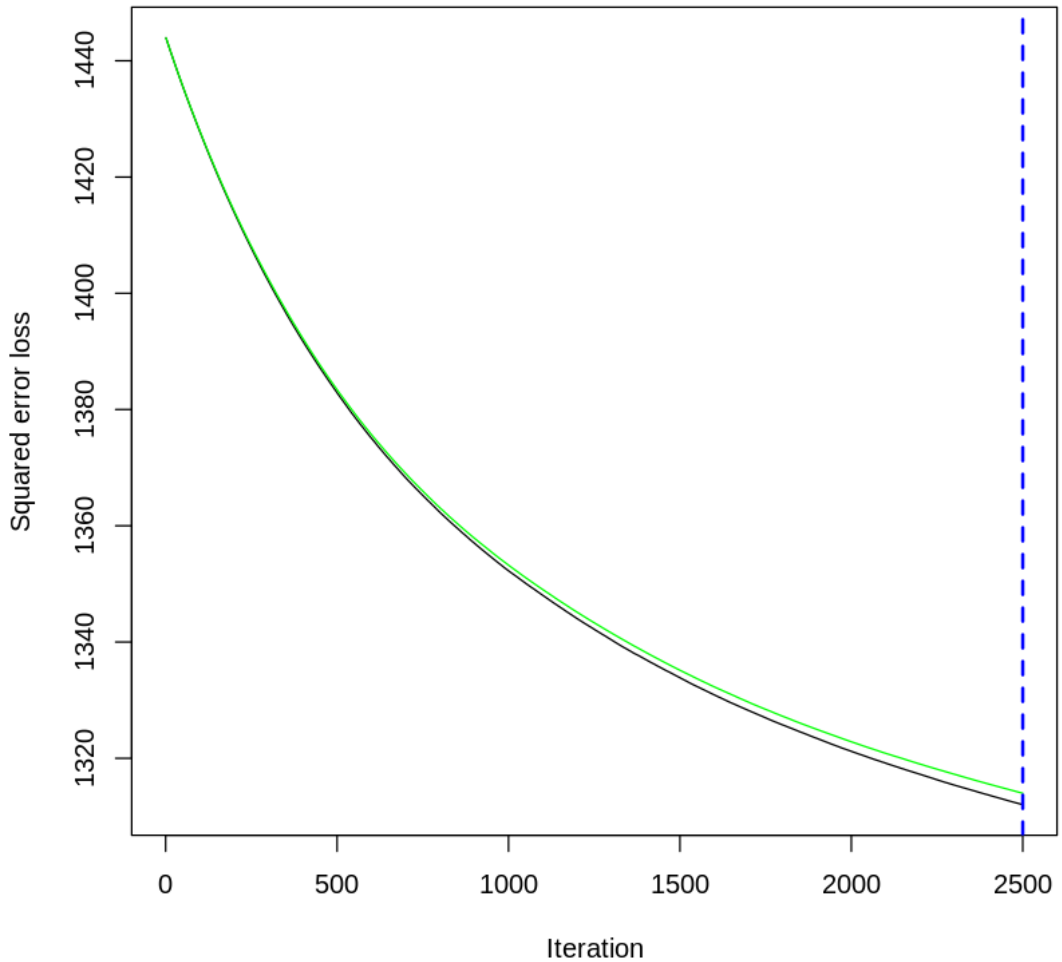
36.2488938792788

2500

19.7773265539083

1274.884529184

35.7055251912641



##23. RF on train/test split. RF on 'fltest.csv' holdout test set

[0]:
```
#flTest2  <- flTest # storing for RF holdout set metrics
# flTest  <- fl_test

#set.seed(222)
#   default RF model
#   rf.fit <- randomForest(
#   formula = dep_delay ~ .,
#   data    = fl_train
# )
```

```
# pred_rf <- predict(rf.fit, flTest)

# mae_rf_flTest <- mae(flTest$dep_delay, pred_rf)
# mae_rf_flTest.
# mse_rf_flTest <- mse(flTest$dep_delay, pred_rf)#Calculate Mean Squared Error␣
↪(MAE) for hold out data
# mse_rf_flTest
# rmse_rf_flTest <- rmse(flTest$dep_delay, pred_rf)#Calculate Root Mean Squared␣
↪Error for holout data
# rmse_rf_flTest
# plot(rf.fit)
```

[0]:
```
#set.seed(222)
# # default RF model
# rf.fit <- randomForest(
#   formula = dep_delay ~ .,
#   data    = fl_train # whole training data
# )



# pred_rf <- predict(rf.fit, flTest2)

# mae_rf_flTest <- mae(flTest2$dep_delay, pred_rf)
# mae_rf_flTest.
# mse_rf_flTest <- mse(flTest2$dep_delay, pred_rf)#Calculate Mean Squared Error␣
↪(MAE) for hold out data
# mse_rf_flTest
# rmse_rf_flTest <- rmse(flTest2$dep_delay, pred_rf)#Calculate Root Mean␣
↪Squared Error for holout data
# rmse_rf_flTest
# plot(rf.fit)
```