Predicting Flight Delays

BA-810 Team - Jordan Grose, Maraline Torres, Barbara Liang, Yi Ming, Cheng Chen, Jingjing Lu

3/2/2021

Introduction

Flying doesn't always go smoothly. Many people have horror stories of extremely long delays, but others haven't experienced this experience (Lucky them!). This brings us to the following question: Wouldn't it be nice to know how much your flight will likely be delayed?

This project will attempt to predict flight delays in minutes, specifically for all airlines in the Los Angeles, Boston and Atlanta airports. We will employ a number of different models to try to solve this, including regressions, ridge & lasso, decision trees and random forest.

A real world application of this could be a potential flight tracker application. Based on flight metadata, weather data, and possibly other sources of data in the future, predicting departure delay and notifying a flier prior to their arrival at the airport of their expected delay could be a very useful feature of a smartphone application.

Getting the data

We gathered our flight data from the US Department of Transportation (https://www.bts.gov/) for March 2019 and 2020. We chose the following features:

- DAY_OF_MONTH
- YEAR
- DAY_OF_WEEK
- DEP_DELAY: Departure Delay in minutes
- CRS_ARR_TIME: Scheduled Arrival time for flight (not actual arrival time)
- CRS_ELAPSED_TIME: Scheduled/Expected flight duration
- CRS_DEP_HOUR: Scheduled hour of flight departure
- AIRLINE: Airline maker
- humidity: Humidity for the day and location
- precipMM: Precipitation for the day and location
- pressure: Air pressure for the day and location
- tempC: Average temperature for the day and location
- visibility: Visibility index for the day and location
- windspeedKmph: Wind speed for the day and location

The weather variables were retrieved using the World Weather API. We created a new column called "Total_Cancellations" which means the number of cancellations the day before in the specific airport and airline. However, we decided to remove the variable because it didn't have predictive value.

Load Dataset

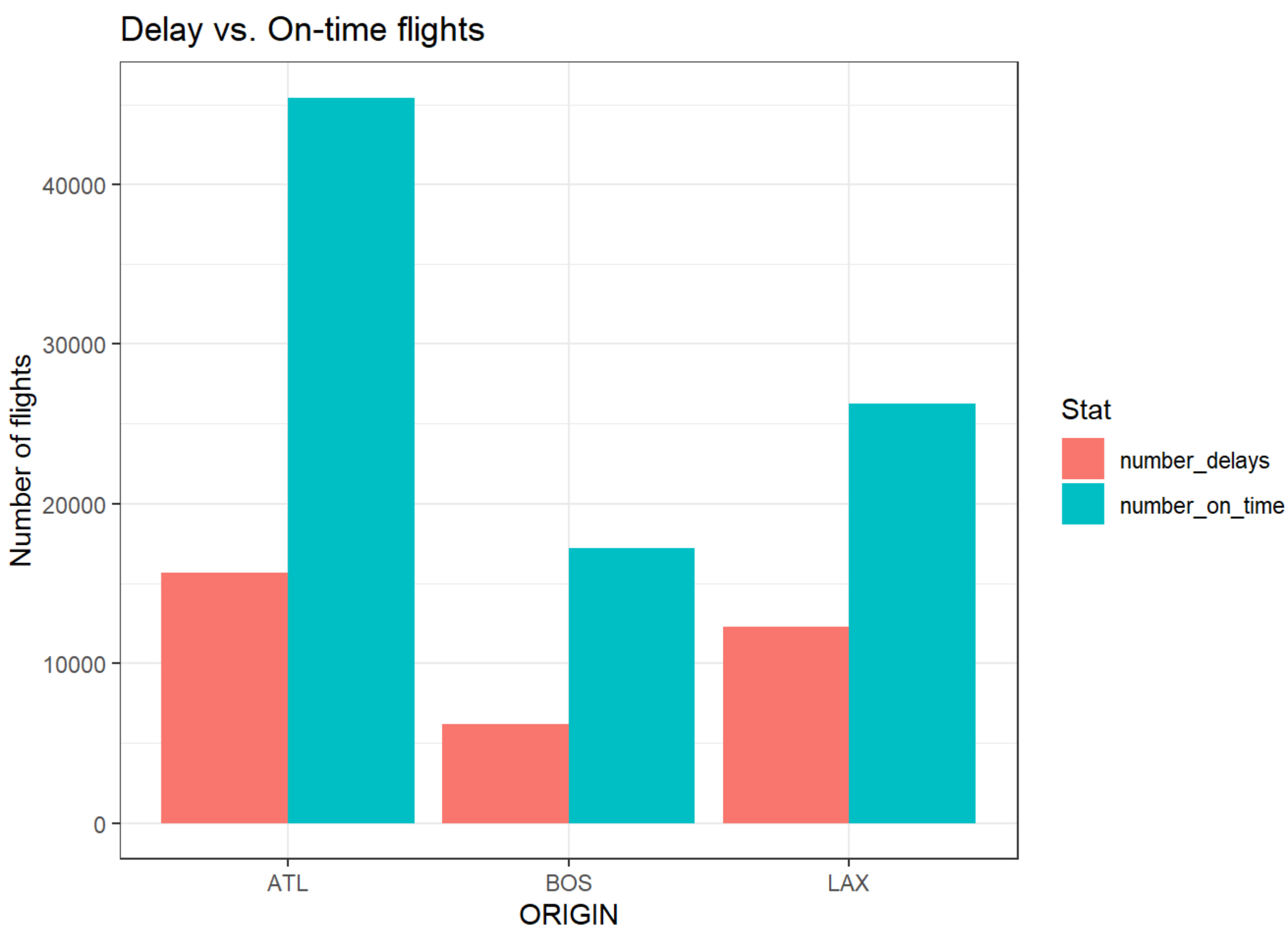
The "df" contains observations for the three airports. The other three are subset data tables for each airport respectively.

```
df <- fread("C:/Users/jorda/Documents/BU/flights.csv")
df.la <- df[ORIGIN == 'LAX', ]
df.bos <- df[ORIGIN == 'BOS', ]
df.atl <- df[ORIGIN == 'ATL', ]
summary(df)
```

```
##   DAY_OF_MONTH        YEAR       DAY_OF_WEEK       ORIGIN
##  Min.   : 1.00   Min.   :2019   Min.   :1.000   Length:122983
##  1st Qu.: 8.00   1st Qu.:2019   1st Qu.:2.000   Class :character
##  Median :15.00   Median :2019   Median :4.000   Mode  :character
##  Mean   :15.05   Mean   :2019   Mean   :4.013
##  3rd Qu.:22.00   3rd Qu.:2020   3rd Qu.:6.000
##  Max.   :31.00   Max.   :2020   Max.   :7.000
##
##      DEST            CRS_DEP_TIME    DEP_DELAY         CRS_ARR_TIME
##  Length:122983      Min.   :   5   Min.   : -68.000   Min.   :   1
##  Class :character   1st Qu.: 951   1st Qu.:  -5.000   1st Qu.:1127
##  Mode  :character   Median :1420   Median :  -3.000   Median :1550
##                     Mean   :1411   Mean   :   5.557   Mean   :1523
##                     3rd Qu.:1840   3rd Qu.:   2.000   3rd Qu.:2004
##                     Max.   :2359   Max.   :1434.000   Max.   :2400
##
##  CRS_ELAPSED_TIME    AIR_TIME        DISTANCE        AIRLINE
##  Min.   : 41.0    Min.   : 14.0   Min.   :  83.0   Length:122983
##  1st Qu.: 90.0    1st Qu.: 59.0   1st Qu.: 369.0   Class :character
##  Median :122.0    Median : 92.0   Median : 612.0   Mode  :character
##  Mean   :152.9    Mean   :121.6   Mean   : 882.2
##  3rd Qu.:193.0    3rd Qu.:158.0   3rd Qu.:1211.0
##  Max.   :690.0    Max.   :694.0   Max.   :5095.0
##                   NA's   :234
##   EARLY_AM           AM               PM            LATE_PM
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:120669    FALSE:79113     FALSE:67244     FALSE:101923
##  TRUE :2314      TRUE :43870     TRUE :55739     TRUE :21060
##
##
##
##
##  CRS_DEP_TIME_Formatted  CRS_DEP_HOUR      humidity        precipMM
##  Length:122983          Min.   : 0.00   Min.   :15.00   Min.   : 0.000
##  Class :character       1st Qu.: 9.00   1st Qu.:44.00   1st Qu.: 0.000
##  Mode  :character       Median :14.00   Median :57.00   Median : 0.100
##                         Mean   :13.84   Mean   :56.02   Mean   : 2.717
##                         3rd Qu.:18.00   3rd Qu.:70.00   3rd Qu.: 2.400
##                         Max.   :23.00   Max.   :92.00   Max.   :43.600
##
##     pressure        tempC         visibility    windspeedKmph
##  Min.   : 995   Min.   :-5.00   Min.   : 4.00   Min.   : 2.00
##  1st Qu.:1017   1st Qu.:13.00   1st Qu.: 9.00   1st Qu.: 8.00
##  Median :1019   Median :17.00   Median :10.00   Median :11.00
##  Mean   :1020   Mean   :16.29   Mean   :10.42   Mean   :11.49
##  3rd Qu.:1024   3rd Qu.:21.00   3rd Qu.:10.00   3rd Qu.:14.00
##  Max.   :1038   Max.   :30.00   Max.   :20.00   Max.   :26.00
##
```

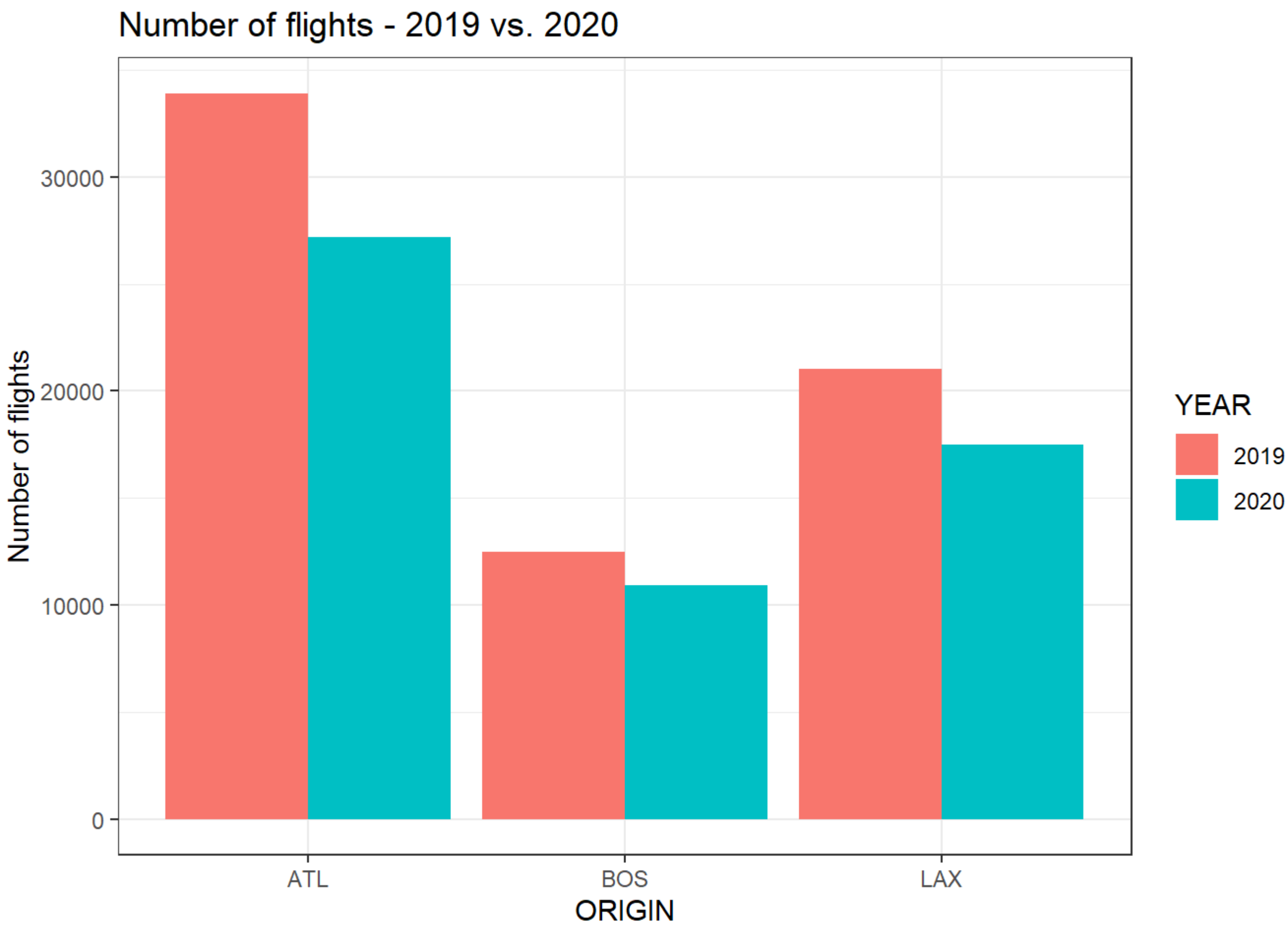
Exploratory data analysis

```
delayed <- df[DEP_DELAY > 0, .(number_delays = .N), by = ORIGIN]
not_delayed <- df[DEP_DELAY <= 0,.(number_on_time = .N), by= ORIGIN]
total <- merge(delayed, not_delayed, by='ORIGIN')
total$ORIGIN <- as.factor(total$ORIGIN)
dat_long <- total %>%
  gather("Stat", "Value", -ORIGIN)
setDT(dat_long)
ggplot(dat_long, aes(x = ORIGIN, y = Value, fill = Stat)) +
  geom_col(position = "dodge") + ylab('Number of flights')  + ggtitle("Delay vs. On-time flight
s")
```

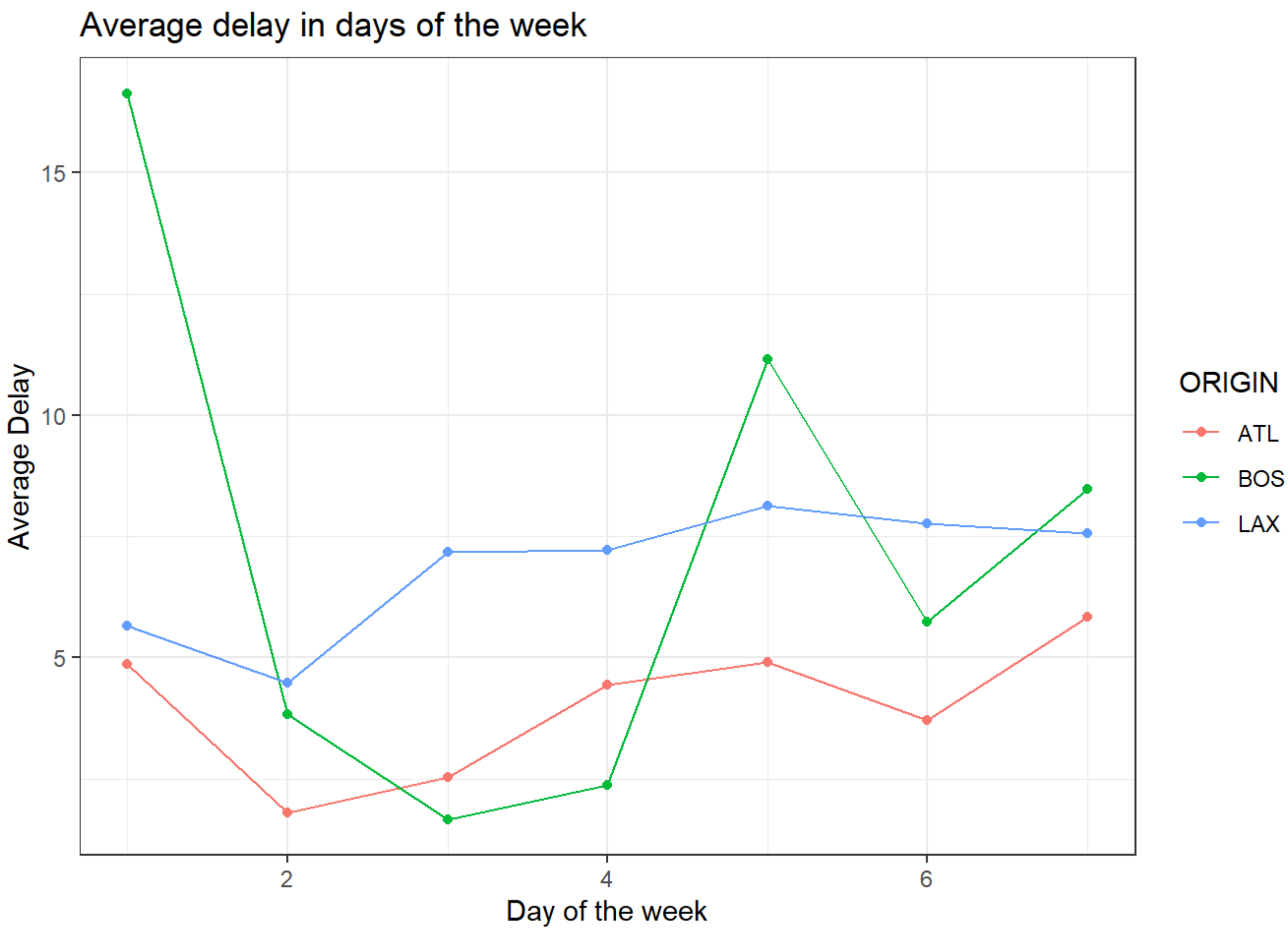


Across all three locations, the majority of flights are on time. The trick is being able to predict which flights will be delayed.

```
flights_by_year <-df[,.(Number_flights = .N), by =.(ORIGIN, YEAR)]
flights_by_year$YEAR <- as.factor(flights_by_year$YEAR)

ggplot(flights_by_year, aes(x = ORIGIN, y = Number_flights, fill =YEAR)) +
  geom_col(position = "dodge") + ylab('Number of flights') + ggtitle("Number of flights - 2019 v
s. 2020")
```

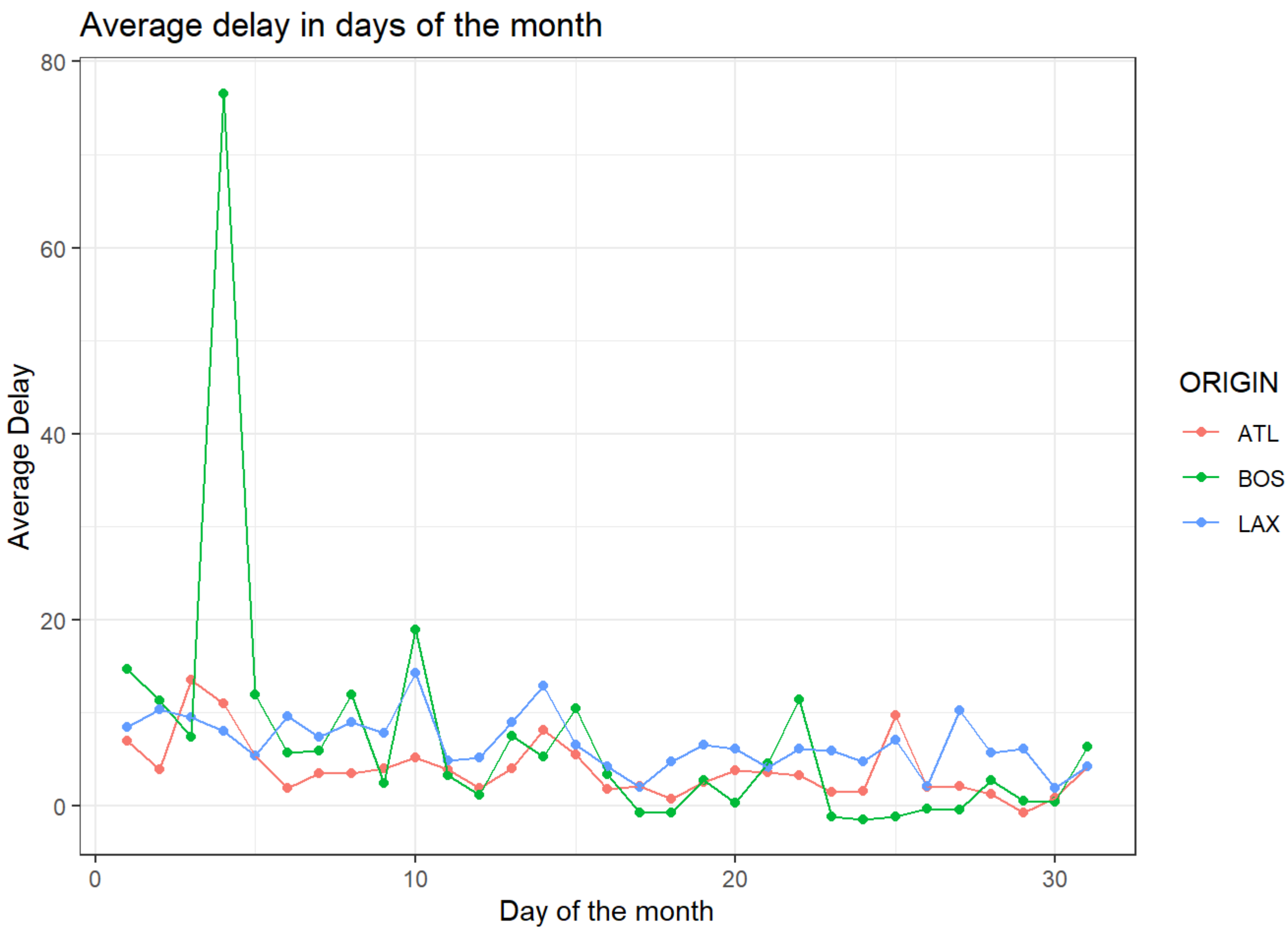



```
weekly_delay <-df[,.(mean_delay = mean(DEP_DELAY)), by =.(DAY_OF_WEEK = df$DAY_OF_WEEK, ORIGIN =
df$ORIGIN)]
ggplot(weekly_delay,aes(x = DAY_OF_WEEK, y = mean_delay, color=ORIGIN)) + geom_point() + geom_li
ne()  + ylab('Average Delay') + xlab('Day of the week') + ggtitle("Average delay in days of the
 week")
```



Delays tend to increase later in the week. This could be because airports are busier on the weekend. We will likely want to keep Day of Week as a predictor in our models.

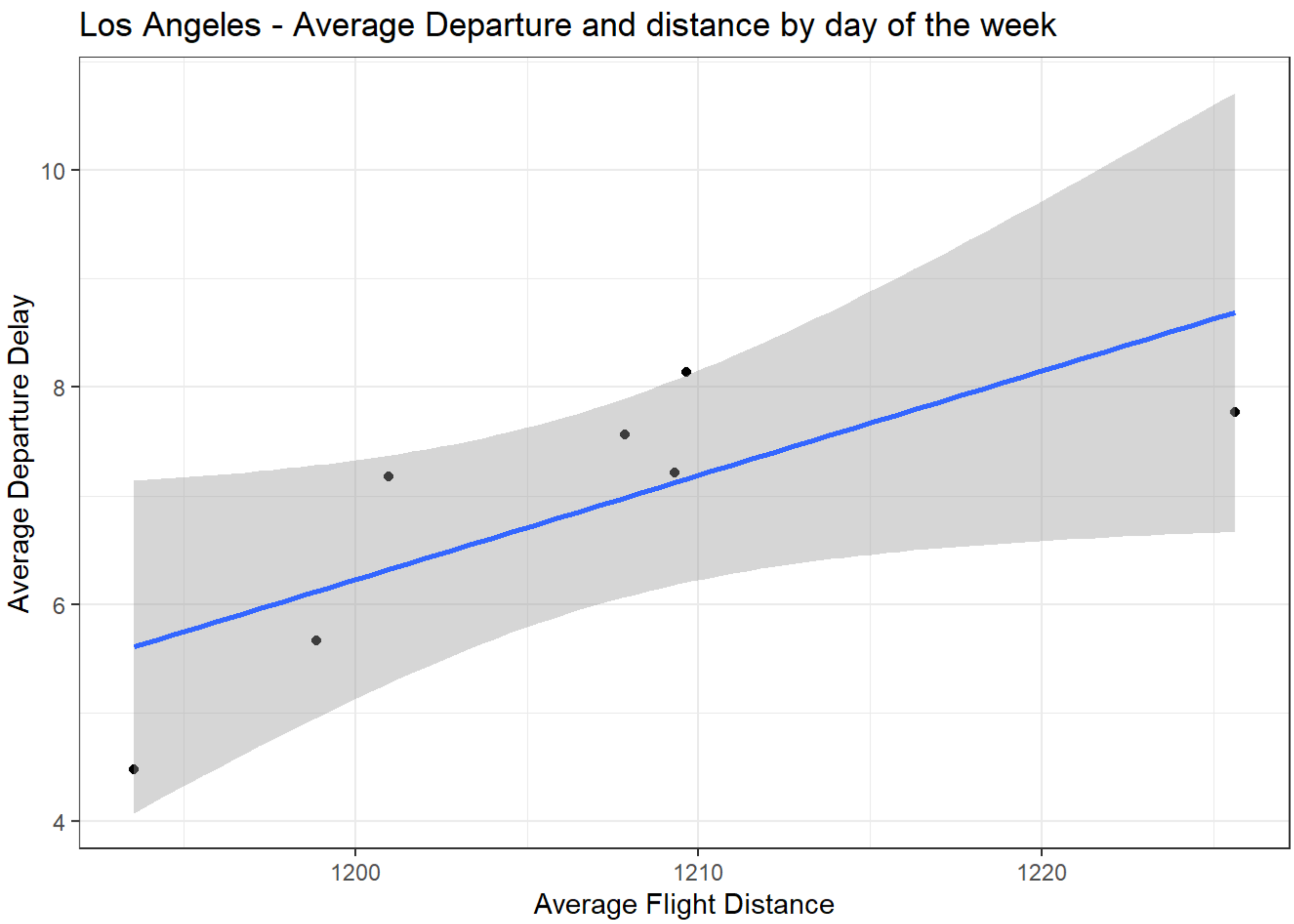
```
day_month_delay <-  df[,.(mean_delay = mean(DEP_DELAY)), by =.(DAY_OF_MONTH = DAY_OF_MONTH, ORIG
IN = ORIGIN)]
ggplot(day_month_delay,aes(x = DAY_OF_MONTH, y = mean_delay, color = ORIGIN)) + geom_point() + g
eom_line() + ylab('Average Delay') + xlab('Day of the month') + ggtitle("Average delay in days o
f the month")
```



Boston has some outliers skewing the avg delay. Otherwise, all three locations share similar trends across the month.

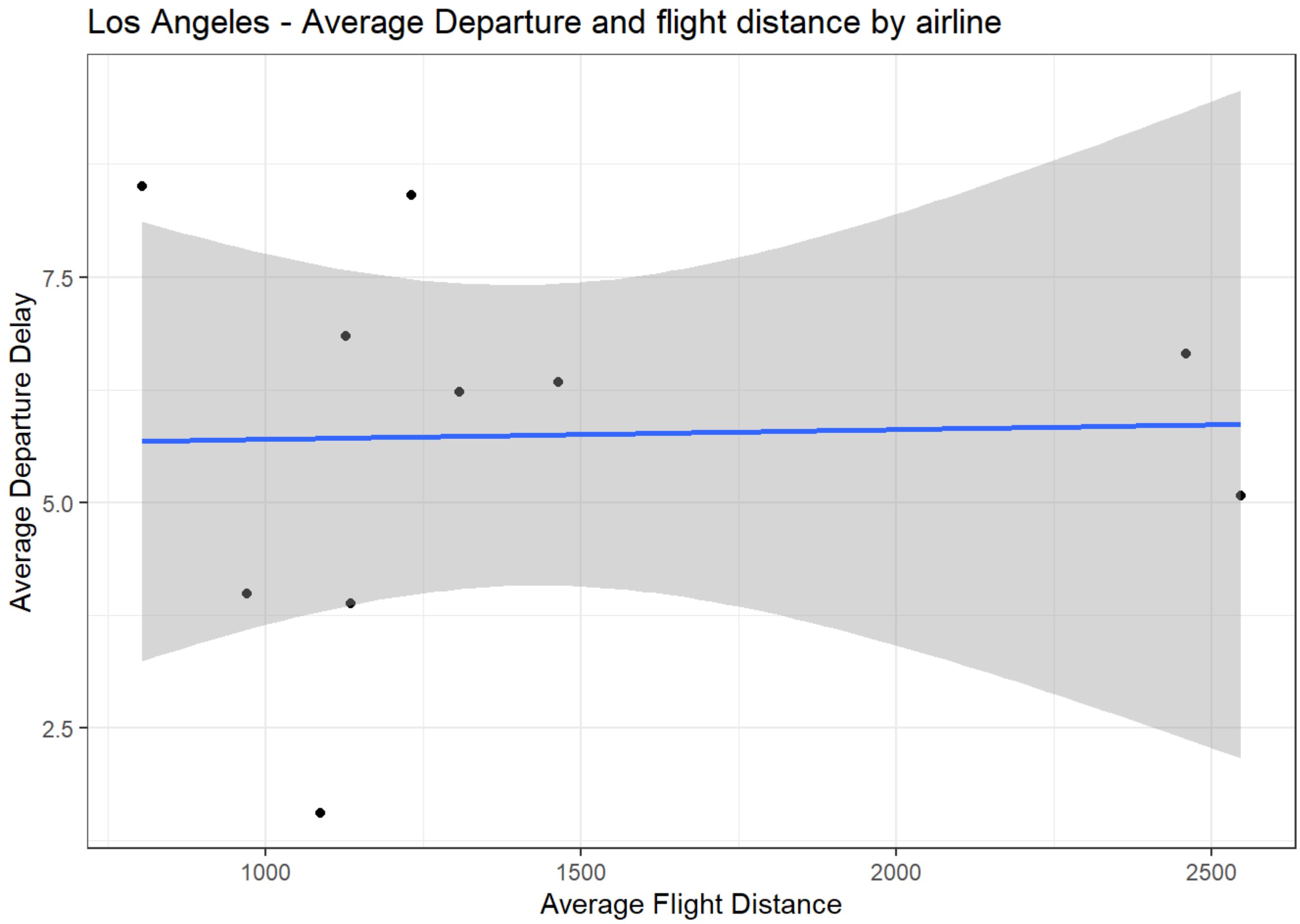
```
distance_delay_weekly <- df.la[,.(delay = mean(DEP_DELAY), dist = mean(DISTANCE)), by ='DAY_OF_W
EEK']
ggplot(distance_delay_weekly,aes(x = dist, y = delay)) + geom_point() + geom_smooth(method = "l
m") + ggtitle("Los Angeles - Average Departure and distance by day of the week") + ylab("Average
Departure Delay") + xlab("Average Flight Distance")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
distance_delay_airline <- df.la[,.( delay = mean(DEP_DELAY), dist = mean(DISTANCE)), by ='AIRLIN
E']
ggplot(distance_delay_airline,aes(x = dist, y = delay)) + geom_point() + geom_smooth(method = "l
m") + ggtitle("Los Angeles - Average Departure and flight distance by airline") + ylab("Average
 Departure Delay") + xlab("Average Flight Distance")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



While Distance seems to have a positive relationship with delays, it is not very strong.

Models for each airport

In this section, we present many of our models for each airport location and highlight which perform best (some models' code may be excluded in order to keep the length of the document down!)

Los Angeles Airport

Naive Regression

Started doing the Naive Regression (Baseline) to have an MSE to compare to.

```
set.seed(123)
y.test.b <- df.test$DEP_DELAY
df_la_y <- mean(df.la$DEP_DELAY)

mse_baseline <- mean((y.test.b - df_la_y)^2)
rmse_baseline <- sqrt(mse_baseline)
mse_baseline
```

```
## [1] 1685.469
```

```
rmse_baseline
```

```
## [1] 41.05446
```

Our goal is to apply more advanced ML methods to obtain a lower RMSE.

Linear Regression

Validation set approach

Fit the model and make predictions using the train & test data sets.

```
y.train.lm <- df.train$DEP_DELAY
y.test.lm <- df.test$DEP_DELAY

f1 <- as.formula(DEP_DELAY ~ DAY_OF_MONTH + YEAR + DAY_OF_WEEK + AIRLINE
                 + CRS_DEP_HOUR  + CRS_ARR_TIME + humidity + precipMM + pressure
                 + tempC + visibility + windspeedKmph)

fit.lm1 <- lm(f1,df.train)

#Let's compute an MSE on the training data
yhat.train.lm1 <- predict(fit.lm1)
mse.train.lm1 <- mean((y.train.lm - yhat.train.lm1)^2)
mse.train.lm1
```

```
## [1] 1370.635
```

```
#Let's compute an MSE on the test data
yhat.test.lm1 <- predict(fit.lm1, df.test)
mse.test.lm1 <- mean((y.test.lm - yhat.test.lm1)^2)
mse.test.lm1
```

```
## [1] 1646.063
```

```
rmse_lm1 <- sqrt(mse.test.lm1)
rmse_lm1
```

```
## [1] 40.5717
```

K-folds cross validation

Considered the LOOCV method but it was too much computational cost. We decided to do the K-folds approach.

```
train.control <- trainControl(method = "cv", number = 10)
#train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
model <- train(f1, data = df.test, method = "lm", trControl = train.control) # Train the model
print(model) # Summarize the results
```

```
## Linear Regression
##
## 7699 samples
##   12 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 6929, 6930, 6928, 6929, 6929, 6930, ...
## Resampling results:
##
##   RMSE      Rsquared    MAE
##   39.11249  0.02924188  16.82624
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Got a lower RMSE using K-means fold cross validation. However, we decided that we need a more flexible model to yield a better MSE.

Ridge Regression

Ridge regression shrinks coefficients towards zero. We used glmnet command to shrink coefficients towards zero. We passed the predictors matrix as parameters and used alpha=0 to invoke ridge regression.

```
dd <- copy(df.la)
dd[, test:=0] #Adds a new column with value 0
dd[sample(nrow(dd),5000), test:=1] #Take 5k random rows and assign it to test
dd.test <- dd[test==1]
dd.train <- dd[test==0] #Around 33K for training

#assign our response variable (target)
y.train.r <- dd.train$DEP_DELAY
y.test.r <- dd.test$DEP_DELAY

x1.train <- model.matrix(f1, dd.train)[,-1]
x1.test <- model.matrix(f1,dd.test)[,-1]

fit.ridge <- cv.glmnet(x1.train, y.train.r, alpha = 0, nfolds = 10)

##Test the MSE in training data
yhat.train.ridge <- predict(fit.ridge, x1.train, s = fit.ridge$lambda.min)
mse.train.ridge <- mean((y.train.r - yhat.train.ridge)^2)
mse.train.ridge
```

```
## [1] 1446.723
```

```
##Test the MSE test
yhat.test.ridge <- predict(fit.ridge, x1.test, alpha = 0, s = fit.ridge$lambda.min)
mse.test.ridge <-  mean((y.test.r - yhat.test.ridge)^2)
mse.test.ridge
```

```
## [1] 1284.646
```

```
rsme_test_ridge <- sqrt(mse.test.ridge)
rsme_test_ridge
```

```
## [1] 35.84196
```

Ridge Regression gives the lowest MSE for Los Angeles airport with +- 35.8419583. This make sense because we have high variance in the data set mainly because of multiple outliers representing longer/weird delays.

This is our best performance yet. Let’s see the optimal lambda and the coefficients.

```
optimal_lambda <- fit.ridge$lambda.min
coef(fit.ridge, s = optimal_lambda)
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                                           1
## (Intercept)                    2.038667e+04
## DAY_OF_MONTH                  -1.286260e-01
## YEAR                          -1.013453e+01
## DAY_OF_WEEK                    3.655191e-01
## AIRLINEAllegiant Air          -1.271729e+00
## AIRLINEAmerican Airlines Inc.  2.153899e+00
## AIRLINEDelta Air Lines Inc.    4.006637e+00
## AIRLINEFrontier Airlines Inc.  3.413081e+00
## AIRLINEHawaiian Airlines Inc.  2.517392e+00
## AIRLINEJetBlue Airways         3.190563e+00
## AIRLINESouthwest Airlines Co.  4.361005e+00
## AIRLINESpirit Air Lines        2.392173e+00
## AIRLINEUnited Air Lines Inc.   2.383629e+00
## CRS_DEP_HOUR                   3.780013e-01
## CRS_ARR_TIME                   1.696494e-03
## humidity                      -9.425759e-03
## precipMM                       1.502217e-01
## pressure                       8.673078e-02
## tempC                         -4.046194e-01
## visibility                    -1.657314e-01
## windspeedKmph                 -2.066544e-01
```

The optimal lambda is 0.4477809.

Lasso Regression

The lasso is like ridge regression – but instead of shrinking coefficients towards zero,it tries to set as many as it can to zero.

```
dd.lasso <- copy(df.la)
dd.lasso[, test:=0]
dd.lasso[sample(nrow(dd.lasso),5000), test:=1]
dd.test.l <- dd.lasso[test==1]
dd.train.l <- dd.lasso[test==0] #Around 33K for training

y.train.lasso <- dd.train.l$DEP_DELAY
y.test.lasso <- dd.test.l$DEP_DELAY

x2.train <- model.matrix(f1, dd.train.l)[,-1]
x2.test <- model.matrix(f1,dd.test.l)[,-1]

fit.lasso <- cv.glmnet(x2.train, y.train.lasso, alpha = 1, nfolds = 10)

##Test the MSE in training data
yhat.train.lasso <- predict(fit.lasso, x2.train, s = fit.lasso$lambda.min)
mse.train.lasso <- mean((y.train.lasso - yhat.train.lasso)^2)
mse.train.lasso
```

```
## [1] 1412.715
```

```
##Test the MSE test
yhat.test.lasso <- predict(fit.lasso, x2.test, alpha = 0, s = fit.lasso$lambda.min)
mse.test.lasso <-  mean((y.test.lasso - yhat.test.lasso)^2)
mse.test.lasso
```

```
## [1] 1512.52
```

```
rmse_lasso <- sqrt(mse.test.lasso)
rmse_lasso
```

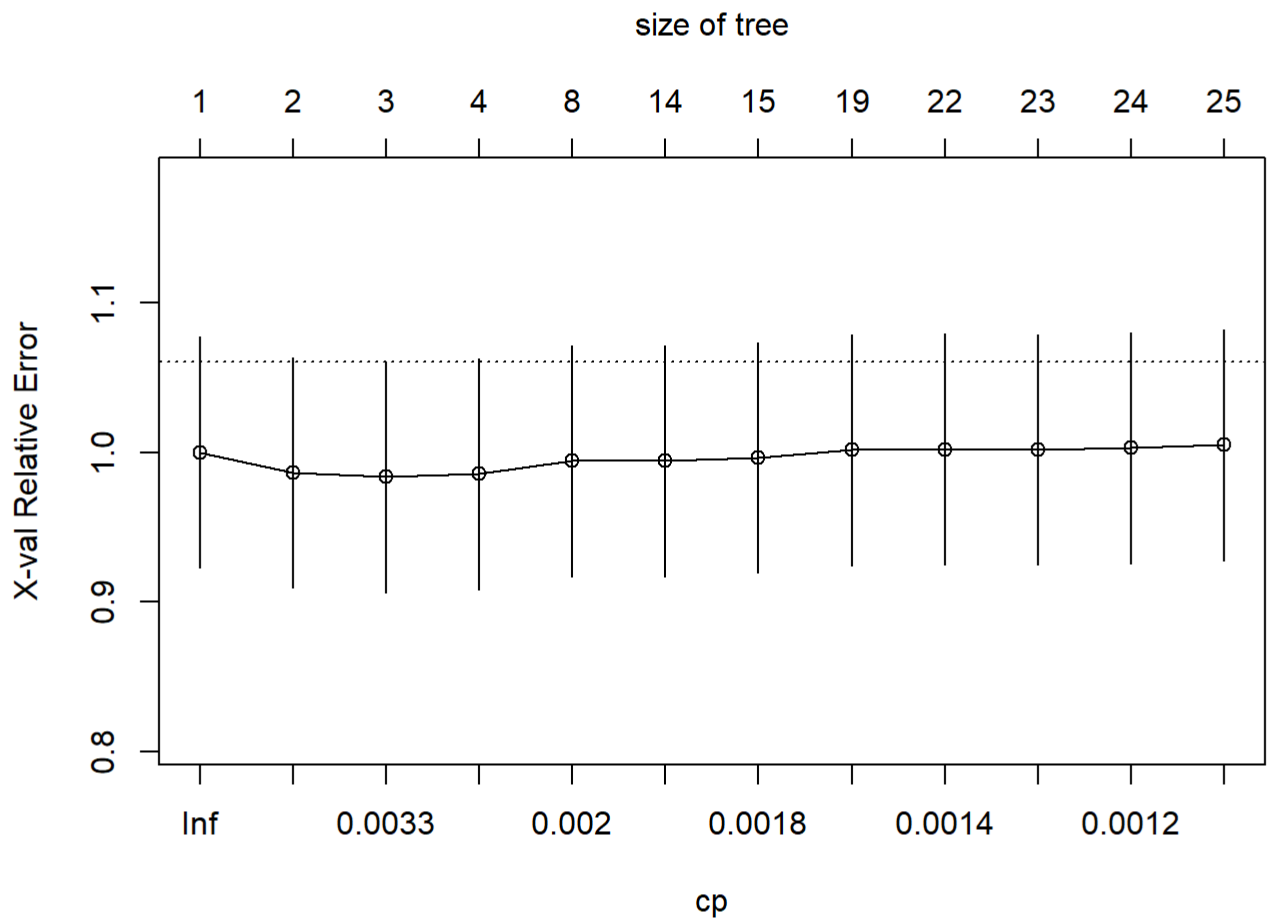
```
## [1] 38.89112
```

Decision Tree Model

```
smp_size <- floor(.75 * nrow(df.la))
train_index <- sample(nrow(df.la), smp_size)
df.test.dt <- df.la[-train_index,] ##not the one in train_index
df.train.dt <-df.la[train_index,]

y.train.dt <- df.train.dt$DEP_DELAY
y.test.dt <- df.test.dt$DEP_DELAY

f5 <- as.formula(DEP_DELAY ~ DAY_OF_MONTH + YEAR + CRS_DEP_HOUR  + CRS_ARR_TIME
                 + CRS_ELAPSED_TIME + humidity + precipMM)

#grow tree
fit.tree <- rpart(f1, data = df.train.dt,control = rpart.control(cp = 0.001), method = "anova")
printcp(fit.tree) # display the results
```

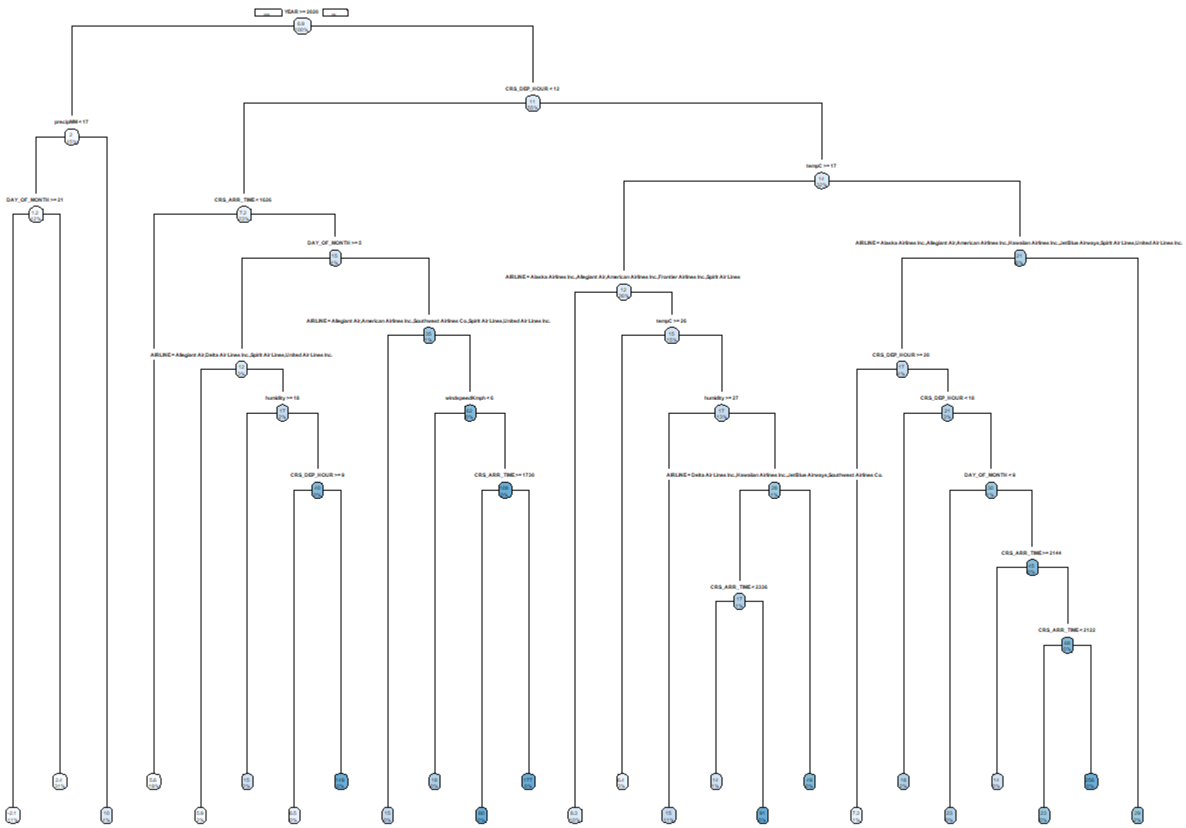
```
##
## Regression tree:
## rpart(formula = f1, data = df.train.dt, method = "anova", control = rpart.control(cp = 0.00
1))
##
## Variables actually used in tree construction:
## [1] AIRLINE       CRS_ARR_TIME  CRS_DEP_HOUR  DAY_OF_MONTH  humidity
## [6] precipMM      tempC         windspeedKmph YEAR
##
## Root node error: 43446471/28871 = 1504.8
##
## n= 28871
##
##           CP nsplit rel error  xerror     xstd
## 1  0.0135507      0   1.00000 1.00002 0.077369
## 2  0.0039872      1   0.98645 0.98651 0.077028
## 3  0.0027799      2   0.98246 0.98361 0.077167
## 4  0.0020612      3   0.97968 0.98552 0.077117
## 5  0.0019006      7   0.97144 0.99436 0.077250
## 6  0.0018665     13   0.96003 0.99423 0.077248
## 7  0.0016538     14   0.95817 0.99641 0.077132
## 8  0.0014568     18   0.95155 1.00182 0.077217
## 9  0.0013858     21   0.94718 1.00210 0.077196
## 10 0.0013052     22   0.94580 1.00190 0.077194
## 11 0.0011144     23   0.94449 1.00292 0.077170
## 12 0.0010000     24   0.94338 1.00504 0.077119
```

```
plotcp(fit.tree) # visualize cross-validation results
```

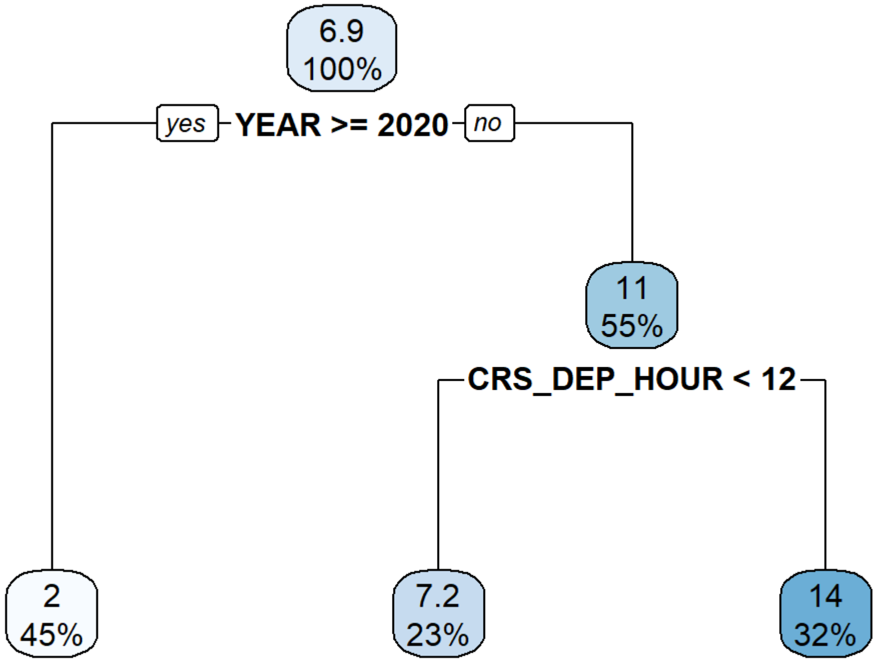


A good choice of cp for pruning is often the leftmost value for which the mean lies below the horizontal line.

```
# plot tree
rpart.plot(fit.tree, type = 1)
```




```r
# prune the tree
optimal_cp <- fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"]
pfit<- prune(fit.tree, cp=optimal_cp) # from cptable
rpart.plot(pfit)
```




```
summary(pfit)
```

```
## Call:
## rpart(formula = f1, data = df.train.dt, method = "anova", control = rpart.control(cp = 0.00
1))
##   n= 28871
##
##            CP nsplit rel error    xerror       xstd
## 1 0.013550730      0 1.0000000 1.0000246 0.07736902
## 2 0.003987160      1 0.9864493 0.9865121 0.07702775
## 3 0.002779899      2 0.9824621 0.9836148 0.07716723
##
## Variable importance
##          YEAR windspeedKmph      precipMM         tempC  CRS_DEP_HOUR
##            36            13            13            11            11
##      humidity      pressure  CRS_ARR_TIME
##             6             5             5
##
## Node number 1: 28871 observations,    complexity param=0.01355073
##   mean=6.919019, MSE=1504.848
##   left son=2 (13082 obs) right son=3 (15789 obs)
##   Primary splits:
##       YEAR          < 2019.5 to the right, improve=0.013550730, (0 missing)
##       humidity      < 69     to the left,  improve=0.005362918, (0 missing)
##       CRS_DEP_HOUR  < 10.5   to the left,  improve=0.003887227, (0 missing)
##       CRS_ARR_TIME  < 1615.5 to the left,  improve=0.003391522, (0 missing)
##       windspeedKmph < 8.5    to the right, improve=0.002808920, (0 missing)
##   Surrogate splits:
##       windspeedKmph < 8.5    to the right, agree=0.712, adj=0.365, (0 split)
##       precipMM      < 0.15   to the right, agree=0.711, adj=0.361, (0 split)
##       tempC         < 18.5   to the left,  agree=0.684, adj=0.303, (0 split)
##       humidity      < 50.5   to the right, agree=0.623, adj=0.168, (0 split)
##       pressure      < 1016.5 to the left,  agree=0.614, adj=0.149, (0 split)
##
## Node number 2: 13082 observations
##   mean=1.958034, MSE=979.9985
##
## Node number 3: 15789 observations,    complexity param=0.00398716
##   mean=11.02945, MSE=1902.426
##   left son=6 (6684 obs) right son=7 (9105 obs)
##   Primary splits:
##       CRS_DEP_HOUR < 11.5   to the left,  improve=0.005767079, (0 missing)
##       CRS_ARR_TIME < 1544.5 to the left,  improve=0.005622706, (0 missing)
##       DAY_OF_MONTH < 10.5   to the right, improve=0.004612928, (0 missing)
##       tempC        < 16.5   to the right, improve=0.004081528, (0 missing)
##       precipMM     < 0.25   to the left,  improve=0.003100703, (0 missing)
##   Surrogate splits:
##       CRS_ARR_TIME < 1611   to the left,  agree=0.760, adj=0.432, (0 split)
##       AIRLINE      splits as  RLRRLLRRRR, agree=0.586, adj=0.021, (0 split)
##
## Node number 6: 6684 observations
##   mean=7.163525, MSE=1499.279
##
## Node number 7: 9105 observations
##   mean=13.86744, MSE=2179.351
```

```
yhat.train.tree <- predict(pfit, df.train.dt)
mse.train.tree <- mean((y.train.dt - yhat.train.tree) ^ 2)
mse.train.tree
```

```
## [1] 1478.456
```

```
yhat.test.tree <- predict(pfit, df.test.dt)
mse.test.tree <- mean((y.test.dt - yhat.test.tree) ^ 2)
mse.test.tree
```

```
## [1] 1308.146
```

```
rmse_tree_test <- sqrt(mse.test.tree)
rmse_tree_test
```

```
## [1] 36.1683
```

The RSME test using Decision Tree is 36.1683035. We were unable to yield a lower RMSE than the one using Ridge Regression.

Random Forrest

For Random Forrest, we tried different formulas but decided ‘f6’ yield the lower MSE.

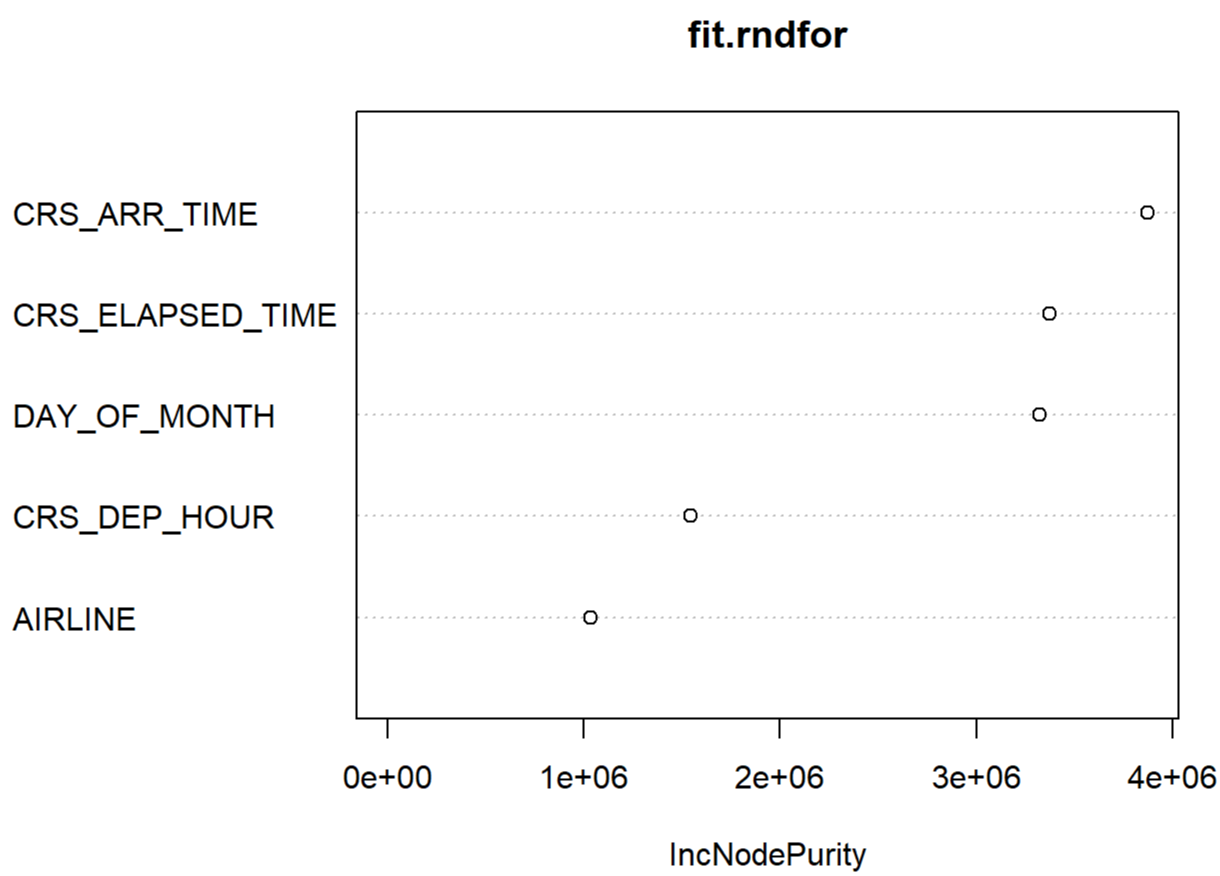
```
smp_size <- floor(.75 * nrow(df.la))
train_index <- sample(nrow(df.la), smp_size)
df.test.rf <- df.la[-train_index,] ##not the one in train_index
df.train.rf <-df.la[train_index,]

y.train.rf <- df.train.rf$DEP_DELAY
y.test.rf <- df.test.rf$DEP_DELAY

f6 <- as.formula(DEP_DELAY ~ DAY_OF_MONTH + AIRLINE + CRS_DEP_HOUR  + CRS_ARR_TIME
                 + CRS_ELAPSED_TIME)
fit.rndfor <- randomForest(f6, df.train.rf,ntree=500, do.trace=F)

print(fit.rndfor) #View results
```

```
##
## Call:
##  randomForest(formula = f6, data = df.train.rf, ntree = 500, do.trace = F)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 1450.421
##                     % Var explained: 1.87
```

```
importance(fit.rndfor)
```

```
##                  IncNodePurity
## DAY_OF_MONTH           3323448
## AIRLINE                1036792
## CRS_DEP_HOUR           1546266
## CRS_ARR_TIME           3875334
## CRS_ELAPSED_TIME       3376062
```

```
#We can check which variables are most predictive using a variable importance plot
varImpPlot(fit.rndfor) #Arrival time, CRS Elapsed time, Depature hour, Airline and Day of Month
```



For Random Forrest, the top three predictors are Arrival Time, Elapsed Time and Day of Month.

```
#Calculate the Train MSE
yhat.train.rndfor <- predict(fit.rndfor, df.train.rf)
mse.train.rndfor <- mean((y.train.rf - yhat.train.rndfor) ^ 2)
mse.train.rndfor
```

```
## [1] 1021.747
```

```
#Calculate the Test MSE
yhat.test.rndfor <- predict(fit.rndfor, df.test.rf)
mse.test.rndfor <- mean((y.test.rf - yhat.test.rndfor) ^ 2)
mse.test.rndfor
```

```
## [1] 1398.961
```

```
rmse.test.rndfor <- sqrt(mse.test.rndfor)
rmse.test.rndfor
```

```
## [1] 37.40269
```

Segmenting LA flights

Since Ridge regression yielded our best results, let's look back at how this LA model performs on various segments of our data set (remember, our Ridge Regression had a Test MSE of 1284.64, or 35 mins, for the test set of LA flights):

```
# apply Ridge model to the entire LA data set, not just the test group
dd[, test := NULL]
LA_model <- dd
X_Ridge <- model.matrix(f1, LA_model)[, -1]
predicted <- predict(fit.ridge, X_Ridge, s=fit.ridge$lambda.min)
mse.predicted <- mean((LA_model$DEP_DELAY - predicted)^2)
mse.predicted # this is our MSE with the trained LA Ridge regression on the entire LA dataset
```

```
## [1] 1425.671
```

```
LA_model[, PREDICTED := predicted]
```

This is our MSE from our best LA model (ridge) on the entire LA data set. Let's see how the model predicts big delays as opposed to small ones:

```
# Big Delays (larger than 90 minutes)
la_grouping1 <- LA_model[DEP_DELAY > 90]
mse.grouping1 <- mean((la_grouping1$DEP_DELAY - la_grouping1$PREDICTED)^2)
mse.grouping1
```

```
## [1] 43325.29
```

```
# Smaller Delays (less than 90 minutes)
la_grouping2 <- LA_model[DEP_DELAY < 90]
mse.grouping2 <- mean((la_grouping2$DEP_DELAY - la_grouping2$PREDICTED)^2)
mse.grouping2
```

```
## [1] 269.2043
```

We can see that our Ridge model has a very high MSE (43,325.29), way higher than our Naive baseline, when we segment the flights to only include those that have a delay greater than 90 minutes. On the other hand, our model has an extremely low MSE when we segment the flights to only include those that have a delay of less than 90 minutes. We expect this to be similar across all models and can conclude that our model does a very good job of predicting smaller delays, but a poor job of predicting large delays. This could be because our feature set may not capture delay variance for large delays very well. Perhaps it is due to randomness, or perhaps there are key events occurring that lead to larger delays that we cannot incorporate into our model (lateness of previous flight, finding a new airplane, luggage or airport issues, etc…)

Boston Models

Our first model is a Naive Linear Regression to get a baseline of MSE performance for future models.

```
### NAIVE LINEAR REGRESSION ###

yhat <- mean(boston$DEP_DELAY)
boston_NLR <- boston
boston_NLR[, yhat := yhat]

mse.NLR <- mean((boston_NLR$DEP_DELAY - boston_NLR$yhat)^2)
mse.NLR
```

```
## [1] 1894.266
```

Next we'll try three linear regression models with a different set of predictors in each to get a sense of which predictors lead to better performance.

```
### LINEAR REGRESSION ###

set.seed(810)
boston_LM1 <- boston

# train/test split
test_index <- sample(nrow(boston_LM1), 4678) # this represents an 80/20 train/test split on the
 entire boston dataset
# now split
dd.test <- boston_LM1[test_index,]
dd.train <- boston_LM1[-test_index,]


# LM Formulas
f1 <- as.formula(DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK + DEST + DISTANCE + AIRLINE + CRS_DEP_HO
UR + humidity + precipMM + pressure + tempC + visibility + windspeedKmph)

f2 <- as.formula(DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK + DEST + DISTANCE + AIRLINE + CRS_DEP_HO
UR + visibility + windspeedKmph)

f3 <- as.formula(DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK + DISTANCE + CRS_DEP_HOUR + humidity + p
recipMM + pressure + tempC + visibility + windspeedKmph)



y.train <- dd.train$DEP_DELAY
y.test <- dd.test$DEP_DELAY


# Fitting the LM model
fit.lm1 <- lm(f1, dd.train)   # all predictors
fit.lm2 <- lm(f2, dd.train)   # removing humidity, precip, pressure, temp
fit.lm3 <- lm(f3, dd.train)   # removing airline and destination


# compute MSEs for training LMs
# LM1
yhat.train.lm1 <- predict(fit.lm1)
mse.train.lm1 <- mean((y.train - yhat.train.lm1)^2)


# LM2
yhat.train.lm2 <- predict(fit.lm2)
mse.train.lm2 <- mean((y.train - yhat.train.lm2)^2)


# LM3
yhat.train.lm3 <- predict(fit.lm3)
mse.train.lm3 <- mean((y.train - yhat.train.lm3)^2)


# Test MSE
# LM1
```

```
yhat.test.lm1 <- predict(fit.lm1, dd.test)
mse.test.lm1 <- mean((y.test - yhat.test.lm1)^2)


# LM2
yhat.test.lm2 <- predict(fit.lm2, dd.test)
mse.test.lm2 <- mean((y.test - yhat.test.lm2)^2)


# LM3
yhat.test.lm3 <- predict(fit.lm3, dd.test)
mse.test.lm3 <- mean((y.test - yhat.test.lm3)^2)
```

Results:
Train MSE LM1: 1483.9359012
Train MSE LM2: 1547.8021063
Train MSE LM3: 1497.8951033
Test MSE LM1: 1803.1110583
Test MSE LM2: 1891.5591969
Test MSE LM3: 1806.8919894

While the Train MSE LM1 was our lowest train MSE, the test MSE for this model was still pretty high with MSE of 1803.11. We expect this might be due to multicolinearity among predictors. Let’s try using Ridge, Lasso and Elastic Net Regressions to control for multicolinearity as we saw our best results in the LA data set using these models. We will use all relevant predictors for these regressions because LM1 (all predictors) performed best for linear regression.

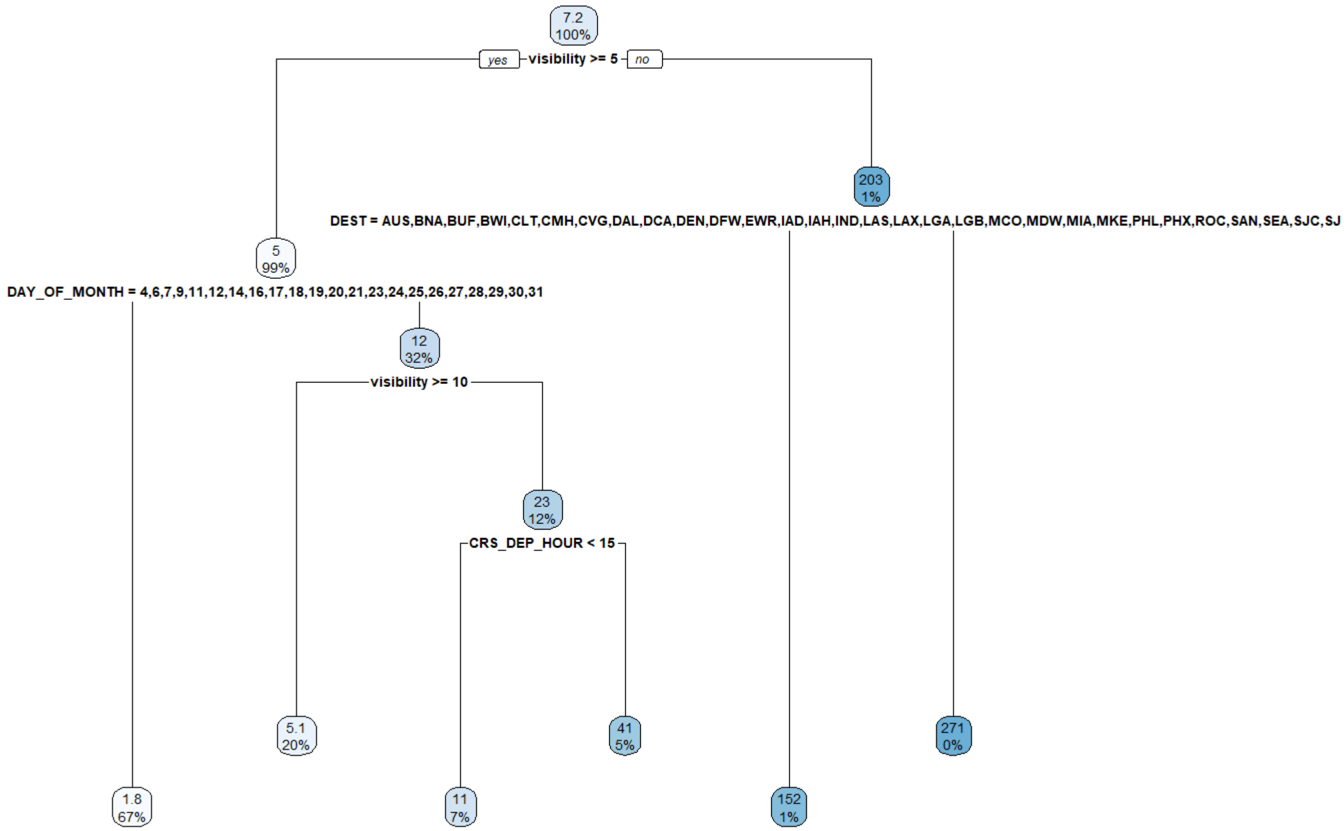
```
#### RIDGE REGRESSION ####

boston <- boston[, yhat := NULL]
boston_RR <- boston

# Random Train/Test split
boston_RR[, test:=0]
boston_RR[sample(nrow(boston_RR), 4678), test:=1]

RR.test <- boston_RR[test==1]
RR.train <- boston_RR[test==0]


x1.train <- model.matrix(f1, RR.train)[, -1]
y.train <- RR.train$DEP_DELAY


x1.test <- model.matrix(f1, RR.test)[, -1]
y.test <- RR.test$DEP_DELAY


fit.ridge <- cv.glmnet(x1.train, y.train, alpha = 0, nfolds = 10)


# Ridge Train MSE
yhat.train.ridge <- predict(fit.ridge, x1.train, s = fit.ridge$lambda.min)
mse.train.ridge <- mean((y.train - yhat.train.ridge)^2)


# Ridge Test MSE
yhat.test.ridge <- predict(fit.ridge, x1.test, s = fit.ridge$lambda.min)
mse.test.ridge <- mean((y.test - yhat.test.ridge)^2)



#### LASSO REGRESSION ####


fit.lasso <- cv.glmnet(x1.train, y.train, alpha = 1, nfolds = 10)


# Lasso Train MSE
yhat.train.lasso <- predict(fit.lasso, x1.train, s = fit.lasso$lambda.min)
mse.train.lasso <- mean((y.train - yhat.train.lasso)^2)


# Lasso Test MSE
yhat.test.lasso <- predict(fit.lasso, x1.test, s = fit.lasso$lambda.min)
mse.test.lasso <- mean((y.test - yhat.test.lasso)^2)



#### ELASTIC NET ####
```

```
fit.net <- cv.glmnet(x1.train, y.train, alpha = 0.5, nfolds = 10)


# Elastic Net Train MSE
yhat.train.net <- predict(fit.net, x1.train, s = fit.net$lambda.min)
mse.train.net <- mean((y.train - yhat.train.net)^2)


# Elastic Net Test MSE
yhat.test.net <- predict(fit.net, x1.test, s = fit.net$lambda.min)
mse.test.net <- mean((y.test - yhat.test.net)^2)
```

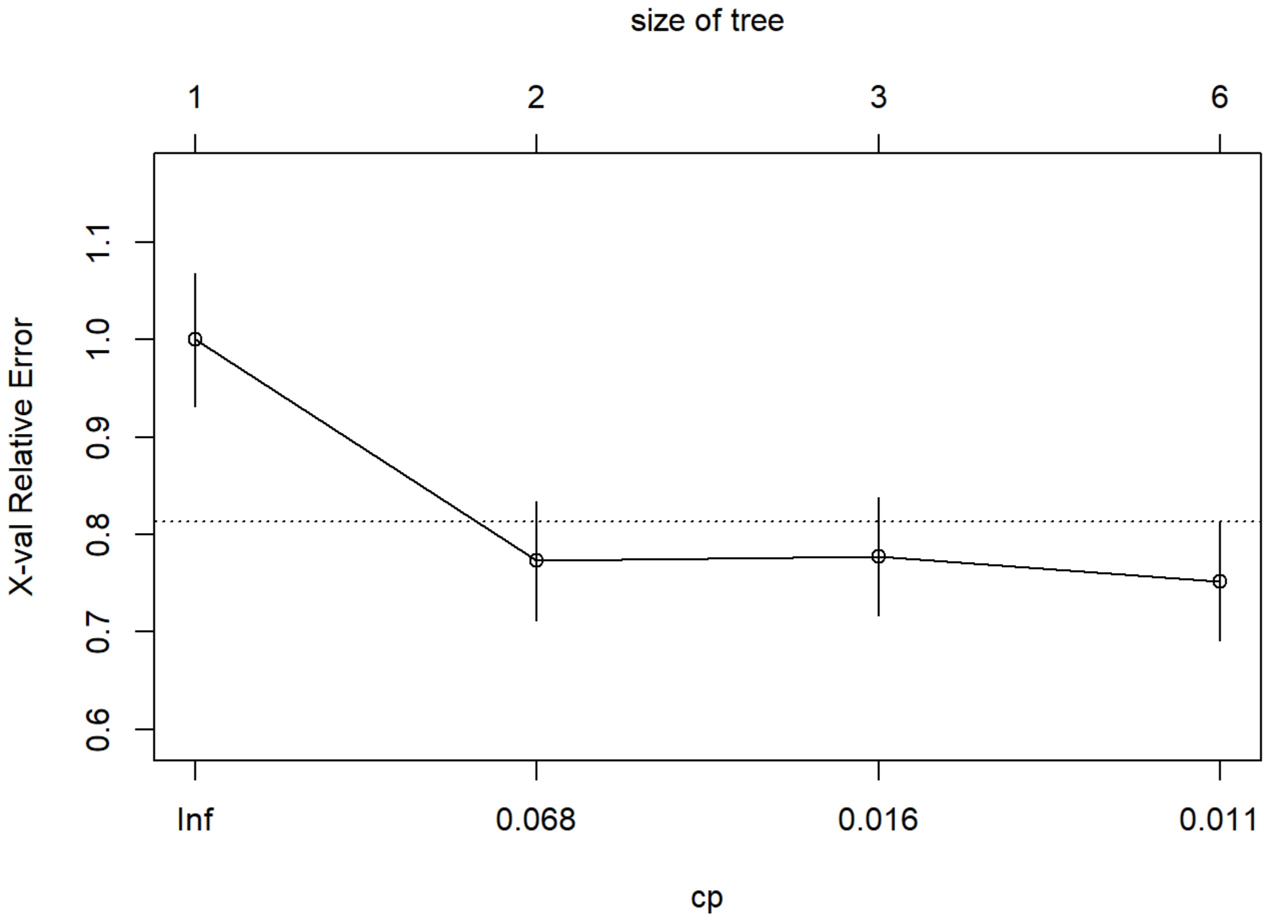
Results:
Ridge Train MSE: 1588.2738784
Ridge Test MSE: 1389.7162676
Lasso Train MSE: 1587.7343442
Lasso Test MSE: 1386.1906389
Net Train MSE: 1587.8484514
Net Test MSE: 1386.2950717

The Lasso Regression with Test MSE of 1386.19 (37.2 minutes) was the best performing model. This is consistent with our LA Ridge regressions being our best performing models. Let's now try running Decision Tree and Random Forest models and compare these results with our Lasso Regression MSE.

```
### DECISION TREE ###

# split the data set into train/test
set.seed(2021)
index=sample(2,nrow(boston),replace = TRUE,prob=c(0.8,0.2))
trainData<-boston[index==1,]
testData<-boston[index==2,]

# fitting the decision tree
tree <- rpart(DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK + DEST + DISTANCE + AIRLINE + CRS_DEP_HOUR
 + humidity + precipMM + pressure + tempC + visibility + windspeedKmph, data = trainData, method
= "anova")
rpart.plot(tree)
```


7.2
100%
yes
visibility >= 5
no
203
1%
DEST = AUS,BNA,BUF,BWI,CLT,CMH,CVG,DAL,DCA,DEN,DFW,EWR,IAD,IAH,IND,LAS,LAX,LGA,LGB,MCO,MDW,MIA,MKE,PHL,PHX,ROC,SAN,SEA,SJC,SJ
5
99%
DAY_OF_MONTH = 4,6,7,9,11,12,14,16,17,18,19,20,21,23,24,25,26,27,28,29,30,31
12
32%
visibility >= 10
23
12%
CRS_DEP_HOUR < 15
5.1
20%
41
5%
271
0%
1.8
67%
11
7%
152
1%

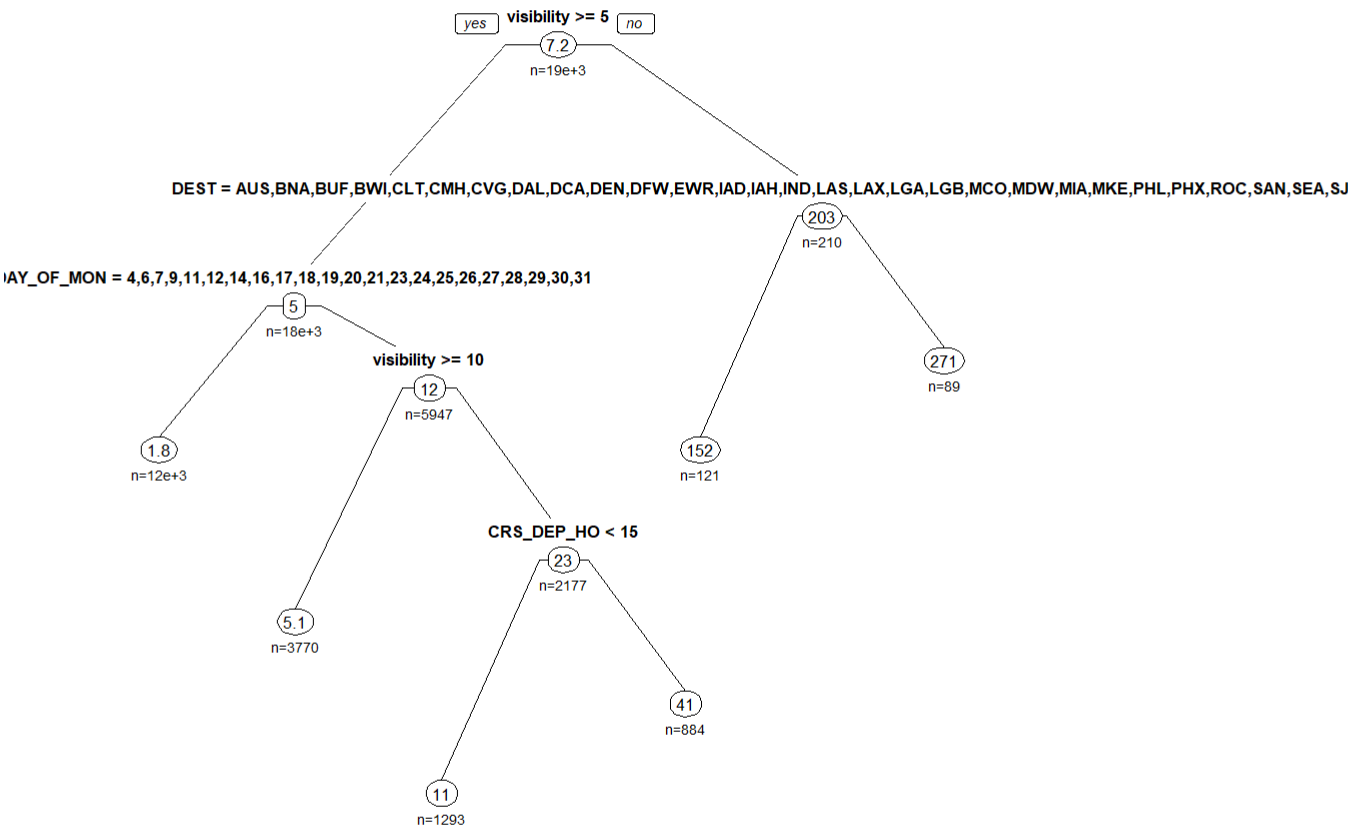
```
plotcp(tree)
```




```
tree$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.22760026      0 1.0000000 1.0000410 0.06815659
## 2 0.02029548      1 0.7723997 0.7728615 0.06084120
## 3 0.01208766      2 0.7521043 0.7773146 0.06079881
## 4 0.01000000      5 0.7158413 0.7520202 0.06075622
```

```
# prune the tree with cp = 0.01
prune.tree <- prune(tree, cp = 0.01)
prp(prune.tree, type = 1, extra = 1, under = TRUE, split.font = 2, varlen = -10)
```




```
# evaluate tree performance
pred <- predict(prune.tree, newdata = testData)

mse.tree <- mean((pred - testData$DEP_DELAY ) ^ 2)
print(mse.tree)
```

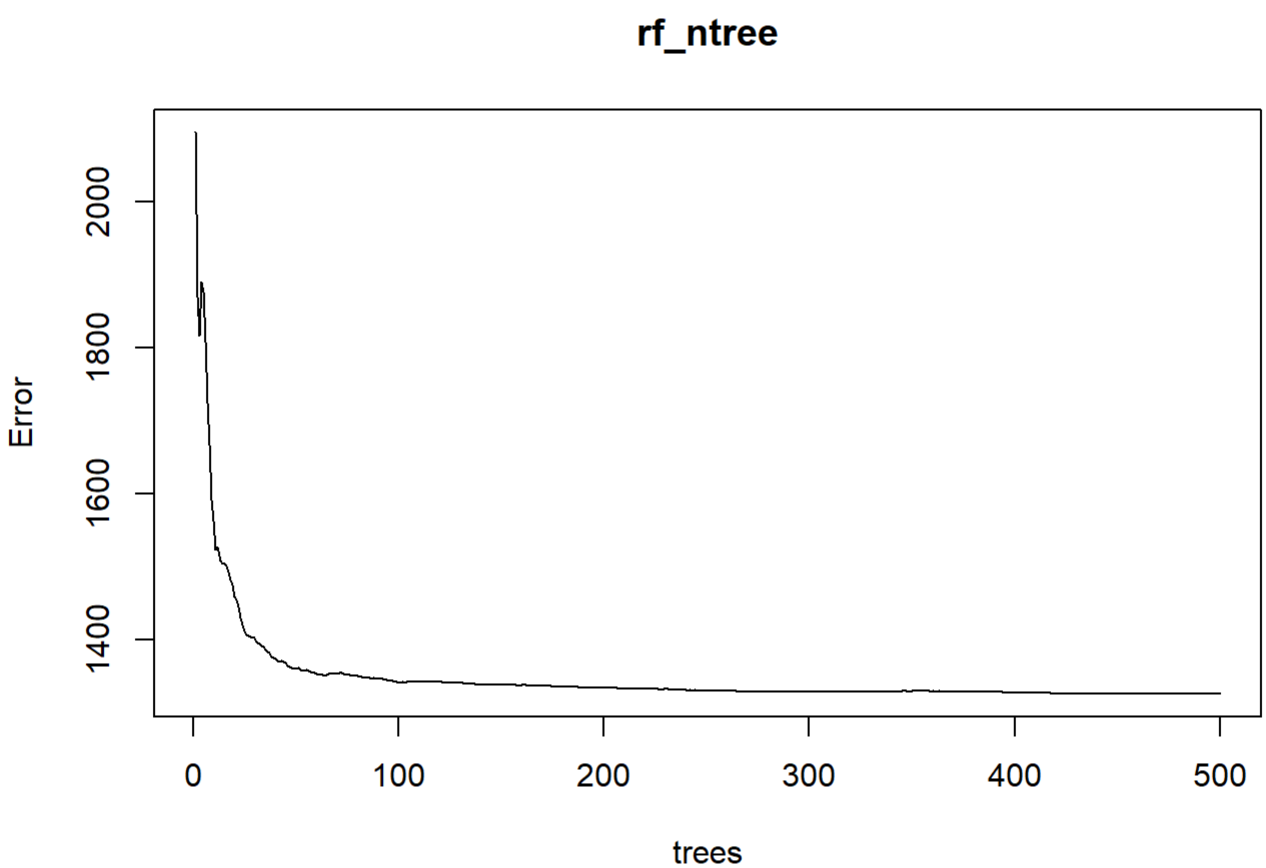
```
## [1] 1339.099
```

```
Values1 <- data.frame(obs = testData$DEP_DELAY, pred = pred)
defaultSummary(Values1)
```

```
##       RMSE   Rsquared        MAE
## 36.5937084  0.2619616 17.4161321
```

The Decision Tree yields an MSE of 1339.09 for the Boston dataset, which is our lowest MSE yet! According to the tree, visibility and the time of the flight are the most significant predictors of departure delay. Let's see how three Random Forest models compares.

```
trainData <- na.omit(trainData)

rf_ntree <- randomForest(DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK + DEST + DISTANCE + AIRLINE + CR
S_DEP_HOUR + humidity + precipMM + pressure + tempC + visibility + windspeedKmph,data = trainDat
a,ntree=500, proximity=TRUE)
plot(rf_ntree)
```




```
rsample.rf=randomForest(DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK + DEST + DISTANCE + AIRLINE + CRS
_DEP_HOUR + humidity + precipMM + pressure + tempC + visibility + windspeedKmph,data = trainDat
a,ntree=60,mtry=2, proximity=TRUE)
print(rsample.rf)
```

```
##
## Call:
##  randomForest(formula = DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK +      DEST + DISTANCE + AIRLI
NE + CRS_DEP_HOUR + humidity + precipMM +      pressure + tempC + visibility + windspeedKmph, da
ta = trainData,      ntree = 60, mtry = 2, proximity = TRUE)
##                Type of random forest: regression
##                      Number of trees: 60
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 1347.342
##                     % Var explained: 29.63
```

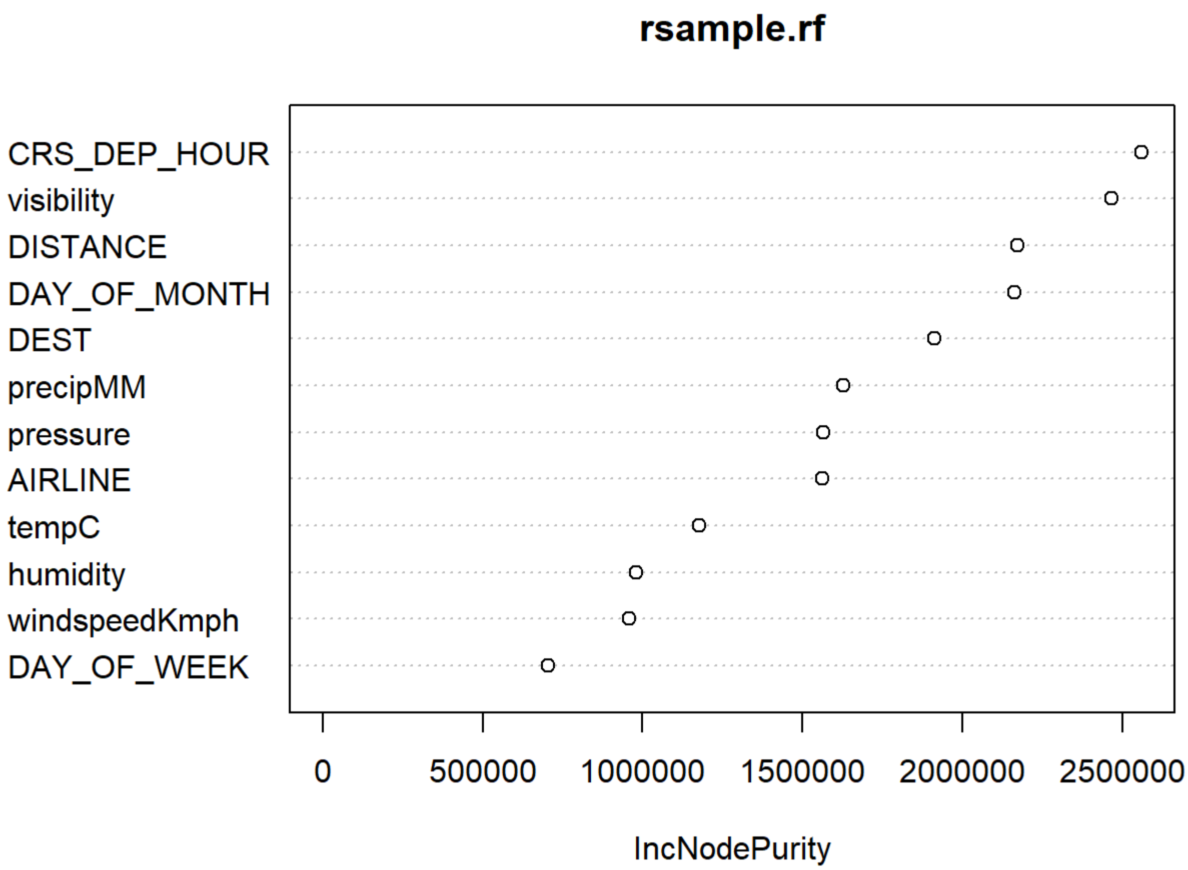
```
rsample.rf=randomForest(DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK + DEST + DISTANCE + AIRLINE + CRS
_DEP_HOUR + humidity + precipMM + pressure + tempC + visibility + windspeedKmph,data = trainDat
a,ntree=100,mtry=3, proximity=TRUE)
print(rsample.rf)
```

```
##
## Call:
##  randomForest(formula = DEP_DELAY ~ DAY_OF_MONTH + DAY_OF_WEEK +      DEST + DISTANCE + AIRLI
NE + CRS_DEP_HOUR + humidity + precipMM +      pressure + tempC + visibility + windspeedKmph, da
ta = trainData,      ntree = 100, mtry = 3, proximity = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 1335.875
##                     % Var explained: 30.23
```

```
importance(rsample.rf, type=2)
```

```
##                IncNodePurity
## DAY_OF_MONTH      2163940.1
## DAY_OF_WEEK        703726.5
## DEST              1911774.6
## DISTANCE          2171780.8
## AIRLINE           1560803.9
## CRS_DEP_HOUR      2560232.0
## humidity           979814.4
## precipMM          1627005.2
## pressure          1565713.3
## tempC             1176699.3
## visibility        2465131.6
## windspeedKmph      958866.1
```

```
varImpPlot(rsample.rf)
```




```
rsample_pred=predict(rsample.rf,testData)
```

Our last Random forest model returns an MSE of 1335.87, our lowest MSE yet. We can see that the optimal number of trees is around 100. We are conscious that increasing the number of trees will result in higher variance in our results going forward, so we want to select a number of trees that both minimizes MSE and keeps Variance as low as possible. We believe this will be 100 trees, with an MSE of 1335.87 (36.5 minutes) based on the Boston data. Again, the random forest model places visibility and departure time as being the most significant predictors for departure delay.

Segmenting Boston flights

While Random Forest yielded our best results, let’s look back at our Lasso Regression to test various segments of our data set, while keeping our Lasso predicted values to ultimately calculate MSE for various Boston flights (remember, our Lasso Regression had a Test MSE of 1386.19, or 37 mins, for the test set of boston flights):

```
# apply Lasso model to the entire Boston data set, not just the test group
boston[, test := NULL]
boston_model <- boston
X_lasso <- model.matrix(f1, boston_model)[, -1]
predicted <- predict(fit.lasso, X_lasso, s=fit.lasso$lambda.min)
mse.predicted <- mean((boston_model$DEP_DELAY - predicted)^2)
mse.predicted # this is our MSE with the trained Boston Lasso regression on the entire Boston da
taset
```

```
## [1] 1547.424
```

```
boston_model[, PREDICTED := predicted]
```

Let’s see how the model predicts big delays as opposed to small ones

```
# Big Delays (larger than 90 minutes)
bos_grouping1 <- boston_model[DEP_DELAY > 90]
mse.grouping1 <- mean((bos_grouping1$DEP_DELAY - bos_grouping1$PREDICTED)^2)
mse.grouping1
```

```
## [1] 30408.22
```

```
# Smaller Delays (less than 90 minutes)
bos_grouping2 <- boston_model[DEP_DELAY < 90]
mse.grouping2 <- mean((bos_grouping2$DEP_DELAY - bos_grouping2$PREDICTED)^2)
mse.grouping2
```

```
## [1] 404.5207
```

We can see that our Lasso model has a very high MSE, way higher than our Naive baseline, when we segment the flights to only include those that have a delay greater than 90 minutes. On the other hand, our model has an extremely low MSE when we segment the flights to only include those that have a delay of less than 90 minutes. We saw a similar result while segmenting our LA data set with predicted values from its Ridge Regression.

Atlanta Flights

Our last location is the Atlanta airport. We used similar code and formulas as the other two locations. For Linear Regression, we yield a RSME of 28-29 minutes.

Below is our code for the Atlanta Ridge Regression, which yielded our best MSEs for this data set. Also, we compare this MSE with the Naive Regression.

Naive Regression

Started doing the Naive Regression (Baseline) to have an MSE to compare to.

```
set.seed(123)
y.test.b <- df.test$DEP_DELAY
df_atl_y <- mean(df.train$DEP_DELAY)
mse_baseline <- mean((y.test.b - df_atl_y)^2)
rmse_baseline <- sqrt(mse_baseline)
mse_baseline
```

```
## [1] 860.1254
```

```
rmse_baseline
```

```
## [1] 29.32789
```

We want to apply better ML techniques to yield a lower MSE.

Ridge Regression

Let’s see the RSME using Ridge Regression. Remember, this shrinks coefficients towards zero and reduces variance in our prediction.

```
set.seed(12345)
dd <- copy(df.atl)
dd[, test:=0] #Adds a new column with value 0
dd[sample(nrow(dd),5000), test:=1] #Take 5k random rows and assign it to test
dd.test <- dd[test==1]
dd.train <- dd[test==0] #Around 33K for training

#assign our response variable (target)
y.train.r <- dd.train$DEP_DELAY
y.test.r <- dd.test$DEP_DELAY

f1 <- as.formula(DEP_DELAY ~ DAY_OF_MONTH + YEAR + DAY_OF_WEEK + AIRLINE
                 + CRS_DEP_HOUR  + CRS_ARR_TIME + humidity + precipMM + pressure
                 + tempC + visibility + windspeedKmph)

x1.train <- model.matrix(f1, dd.train)[,-1]
x1.test <- model.matrix(f1,dd.test)[,-1]
fit.ridge <- cv.glmnet(x1.train, y.train.r, alpha = 0, nfolds = 10)
##Test the MSE in training data
yhat.train.ridge <- predict(fit.ridge, x1.train, s = fit.ridge$lambda.min)
mse.train.ridge <- mean((y.train.r - yhat.train.ridge)^2)
mse.train.ridge
```

```
## [1] 795.5863
```

```
##Test the MSE test
yhat.test.ridge <- predict(fit.ridge, x1.test, alpha = 0, s = fit.ridge$lambda.min)
mse.test.ridge <-  mean((y.test.r - yhat.test.ridge)^2)
mse.test.ridge
```

```
## [1] 612.341
```

```
rsme_test_ridge <- sqrt(mse.test.ridge)
rsme_test_ridge
```

```
## [1] 24.74552
```

```
optimal_lambda <- fit.ridge$lambda.min
coef(fit.ridge, s = optimal_lambda)
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                                          1
## (Intercept)                   1.622627e+04
## DAY_OF_MONTH                 -2.057584e-01
## YEAR                         -8.010303e+00
## DAY_OF_WEEK                   3.224824e-01
## AIRLINEAmerican Airlines Inc.  1.989824e+00
## AIRLINEDelta Air Lines Inc.   -1.838202e+00
## AIRLINEFrontier Airlines Inc. -7.034998e-01
## AIRLINEJetBlue Airways         8.088111e+00
## AIRLINESouthwest Airlines Co.  2.626730e-01
## AIRLINESpirit Air Lines        3.931301e+00
## AIRLINEUnited Air Lines Inc.   2.897993e+00
## CRS_DEP_HOUR                   2.724168e-01
## CRS_ARR_TIME                   1.029388e-03
## humidity                       5.028872e-02
## precipMM                       4.850696e-01
## pressure                      -5.331905e-02
## tempC                          1.203806e-01
## visibility                    -1.831202e-01
## windspeedKmph                  1.530610e-01
```

The optimal lambda is 0.273277 and we also printed the coefficients using Ridge Regression. We also ran Lasso Regression and it resulted in a similar MSE.

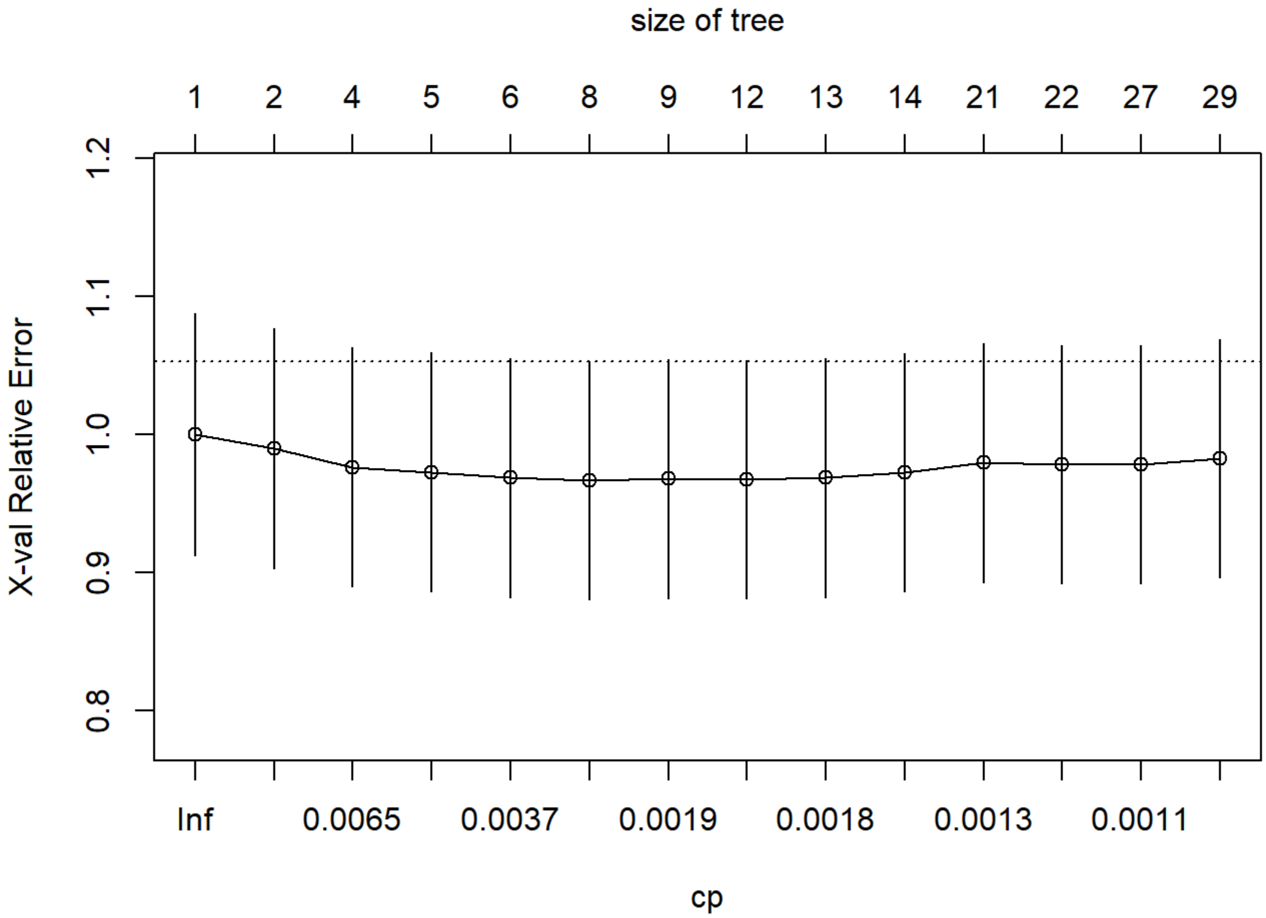
Decision Tree Model

Let’s see if Decision Tree Model yields a lower MSE.

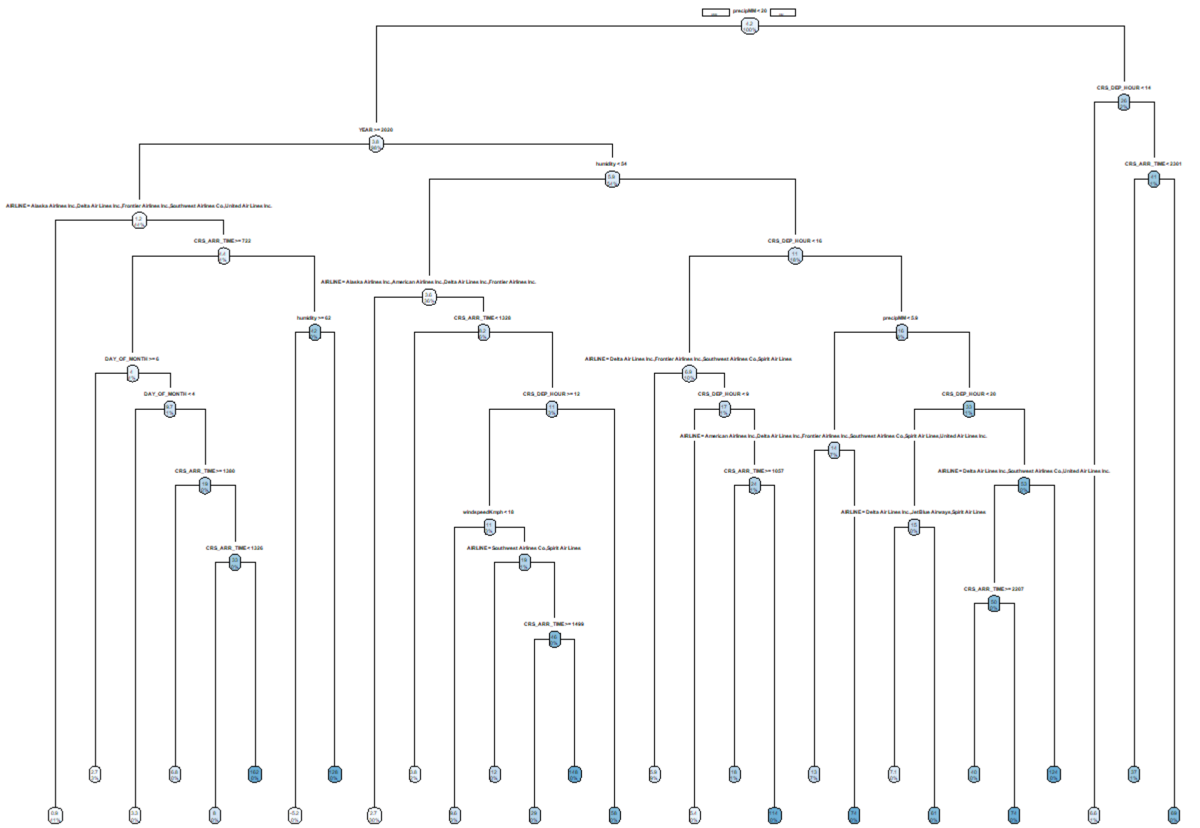
```
set.seed(12345)
smp_size <- floor(.80 * nrow(df.atl))
train_index <- sample(nrow(df.atl), smp_size)
df.test.dt <- df.atl[-train_index,] ##not the one in train_index
df.train.dt <-df.atl[train_index,]
y.train.dt <- df.train.dt$DEP_DELAY
y.test.dt <- df.test.dt$DEP_DELAY

#grow tree
fit.tree <- rpart(f1, data = df.train.dt,control = rpart.control(cp = 0.001), method = "anova")
printcp(fit.tree) # display the results
```

```
##
## Regression tree:
## rpart(formula = f1, data = df.train.dt, method = "anova", control = rpart.control(cp = 0.00
1))
##
## Variables actually used in tree construction:
## [1] AIRLINE       CRS_ARR_TIME  CRS_DEP_HOUR  DAY_OF_MONTH  humidity
## [6] precipMM      windspeedKmph YEAR
##
## Root node error: 40791237/48879 = 834.54
##
## n= 48879
##
##           CP nsplit rel error  xerror     xstd
## 1  0.0104040      0   1.00000 1.00005 0.087308
## 2  0.0069009      1   0.98960 0.98999 0.087043
## 3  0.0061812      3   0.97579 0.97622 0.086608
## 4  0.0040718      4   0.96961 0.97285 0.086554
## 5  0.0034466      5   0.96554 0.96862 0.086476
## 6  0.0019804      7   0.95865 0.96679 0.086332
## 7  0.0018188      8   0.95667 0.96797 0.086358
## 8  0.0017964     11   0.95121 0.96727 0.086357
## 9  0.0017770     12   0.94941 0.96865 0.086363
## 10 0.0013439     13   0.94764 0.97244 0.086335
## 11 0.0012440     20   0.93823 0.97948 0.086329
## 12 0.0011407     21   0.93699 0.97855 0.086282
## 13 0.0010559     26   0.93128 0.97837 0.086248
## 14 0.0010000     28   0.92917 0.98265 0.086283
```

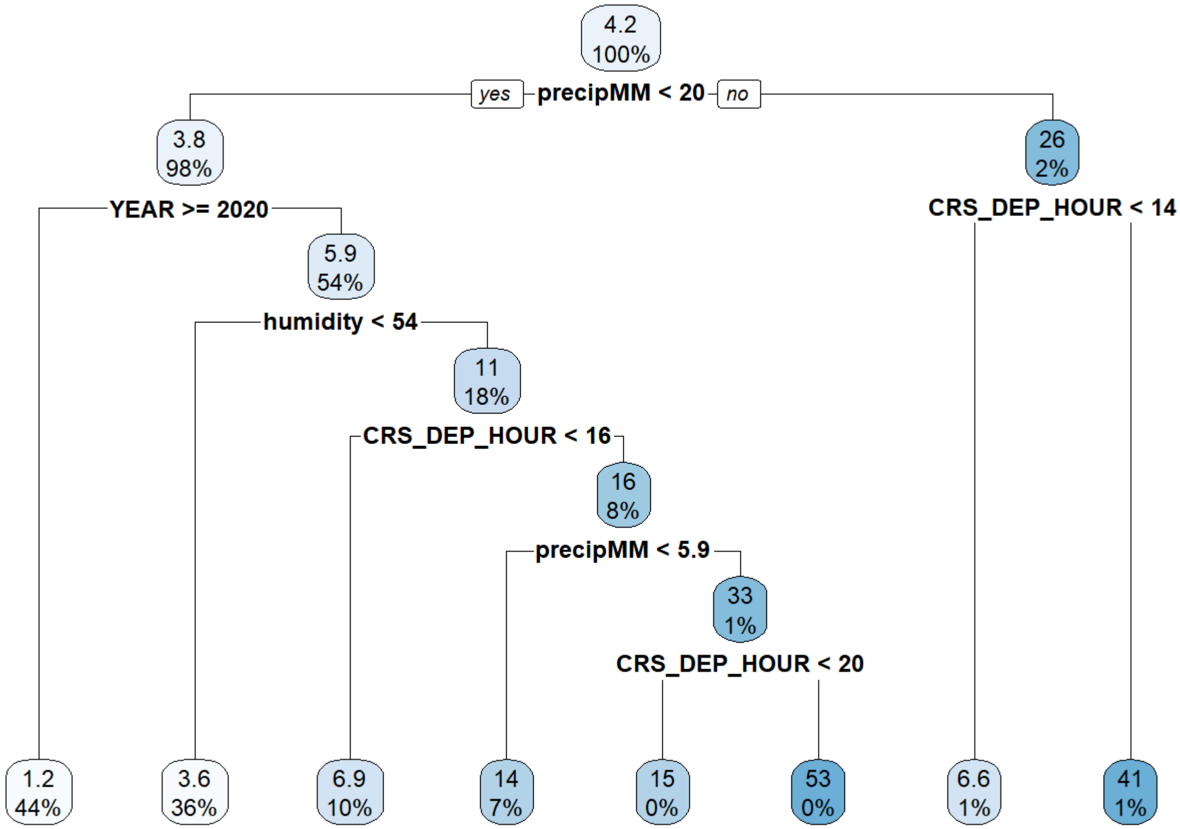
```
plotcp(fit.tree) # visualize cross-validation results
```




```
# plot tree
rpart.plot(fit.tree, type = 1)
```



```
optimal_cp <- fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"]
pfit<- prune(fit.tree, cp=optimal_cp) # from cptable
rpart.plot(pfit)
```




```
#Train
yhat.train.tree.atl <- predict(pfit, df.train.dt)
mse.train.tree.atl <- mean((y.train.dt - yhat.train.tree.atl) ^ 2)
mse.train.tree.atl
```

```
## [1] 800.0253
```

```
#Test
yhat.test.tree.atl <- predict(pfit, df.test.dt)
mse.test.tree.atl <- mean((y.test.dt - yhat.test.tree.atl) ^ 2)
mse.test.tree.atl
```

```
## [1] 648.2747
```

```
rmse_tree_test.atl <- sqrt(mse.test.tree.atl)
rmse_tree_test.atl
```

```
## [1] 25.46124
```

The RSME test using Decision Tree is 25.4612391. We were unable to yield a lower MSE using Decision Tree model.

Segmenting Atlanta Flights

```r
# apply Ridge model to the entire ATL data set, not just the test group
dd[, test := NULL]
ATL_model <- dd
X_ridge_ATL <- model.matrix(f1, ATL_model)[, -1]
predicted_atl <- predict(fit.ridge, X_ridge_ATL, s=fit.ridge$lambda.min)
mse.predicted_atl <- mean((ATL_model$DEP_DELAY - predicted_atl)^2)
mse.predicted_atl # this is our MSE with the trained ATL Ridge regression on the entire ATL data set
```

```
## [1] 780.5905
```

```r
ATL_model[, PREDICTED := predicted_atl]
```

Let’s see how the model predicts big delays as opposed to small ones

```r
# Big Delays (larger than 90 minutes)
atl_grouping1 <- ATL_model[DEP_DELAY > 90]
mse.grouping1 <- mean((atl_grouping1$DEP_DELAY - atl_grouping1$PREDICTED)^2)
mse.grouping1
```

```
## [1] 40777.38
```

```r
# Smaller Delays (less than 90 minutes)
atl_grouping2 <- ATL_model[DEP_DELAY < 90]
mse.grouping2 <- mean((atl_grouping2$DEP_DELAY - atl_grouping2$PREDICTED)^2)
mse.grouping2
```

```
## [1] 183.4946
```

As we saw before, our Ridge model results has a very high MSE, way higher than our Naive baseline, when we segment the flights to only include those that have a delay greater than 90 minutes. On the other hand, our model has an extremely low MSE when we segment the flights to only include those that have a delay of less than 90 minutes. We saw a similar result while segmenting our LA & Boston data sets with predicted values and actual departure delays. This is likely due to our models being able to predict small delays better than large delays, most likely because large delays are related to either randomness or events that we are unable to capture in our current feature set.

Brief summary of all models

TABLE of MSE

Airport	Linear regression	Ridge regression	Lasso Regression	Decision Tree	Random Forest
Los Angeles	1646.06	1284.65	1512.52	1308.15	1398.96
Boston	1803.11	1389.72	1386.19	1339.09	1335.86

Airport	Linear regression	Ridge regression	Lasso Regression	Decision Tree	Random Forest
Atlanta	832.6257	612.341	612.3239	648.2747	N/A

Conclusion

Challenges

- High variance - Most of our models would result in different MSEs without setting a random sampling seed. We suppose that this is why Ridge and Lasso regression tended to show better results. Lasso and Ridge regressions control best for multicolinearity and reduce variance across different samples. However, for the purposes of this report, we have set multiple seeds to maintain consistent results.
- Initially we started out with data at every single airport, amounting to over 2 Million rows. We found that running models and generating EDA plots was a very slow process with this much data. Of the plots we were able to generate with all airports, we were not able to find any discernible trends within the data. Too much data lead to us not being able to identify good models to build, and running models lead took too long and lead to inconsistent results. We decided to start binning our data into three popular locations: LA, BOS, ATL. This method was much more feasible and became the basis of our report.
- While we had most of the actual flight metadata, and weather data, we believe that there is still a large element of randomness at play here. Our feature set and subsequent models did a very good job at predicting small delays, but did not have enough information to predict large delays (> 90 minutes) well at all. We expect these delays are more random or would need more data in order to accurately predict.

Looking Forward

- We looked at this problem mainly from an airport perspective. You could also generate models that look at different airlines and try to predict delays based on which airline is being used.
- If we had more powerful computers, we would likely want to look data from other months and years. With more flights in our models, we would likely be able to reduce the error for predicting both small and larger delays. With more year-round data, we could incorporate different monthly trends as well as holiday trends to capture delays as a function of seasonality or big travel weeks. This would likely help our models in explaining more of the variance we saw in just the month of March.
- We also think it would be interesting and very feasible to predict arrival delay. Obviously a predictor of arrival delay would be departure delay and this would likely give very accurate results. This could be used by people who are picking up friends/family at the airport: Once the flight takes off, the arrival delay would be predicted and the appriopriate parties would be notified of the expected delay.

Final Thoughts

Overall, Atlanta models had the least amount of error. We expect this is because the number of observations was higher than in the other two cities, so the models had more data to work with. We also expect that there was less variance in Atlanta data overall, and possibly less large delays. While our Random Forrest or Decision Tree models did beat out Lasso & Regression for Los Angeles, these models experienced high variance. In Boston

Ridge and Lasso performed best because they were able to control for variance best. Overall, we believe that for this particular dataset, Ridge and Lasso regressions are the most appropriate models to use because they control best for high variance, irrelevent features, and multicolinearity.