Final

Nikhil Gopal

12/19/2021

Provide a clear, detailed description of your overall pipeline sufficient to reproduce your exact pipeline.

In this project, I first loaded the following libraries, and cleaned the dataset by removing NA values. And changing the flight dates to a date object. Afterwards, I split the data into a testing set and a training set to build some starting regression models with.

Finally, I ended up deciding on a Random Forest model, using the variables: year, carrier, month, departure delay, air time, distance, origin airport and destination airport. These variables were found through trial and error. I also tried building an OLS model with a 10 fold cross validation split, but the Random Forest gave me a lower RMSE so I decided to choose that. You can find below my code to clean the data, and for the OLS and the Random Forest model along with RMSE calculation below:

Prepare work space (I hid warning messages):

```
rm(list = ls())
library(randomForest)
require(caTools)
library(readr)
library(tidyverse)
library(caret)
library(Metrics)
library(MASS)
library(glmnet)
library(party)
library(rfviz)
```

Load/clean data:

```
flights <- read.csv("pnwflights14.csv")

# Remove NAs
flights <- na.omit(flights)

# Remove nonsense entries
flights <- flights[-which(flights$dep_time < 0),]
flights <- flights[-which(flights$air_time < 0),]
flights <- flights[-which(flights$distance < 0),]
```

```r
#convert the timing into a date time object
flights$date <- paste(flights$year, "-", flights$month, "-", flights$day, " ", flights$hour, ":", flight

flights$dep_date <- strptime(flights$date, format="%Y-%m-%d %H:%M")

flights2 <- flights


flights2$arr_time2 <- substr(as.POSIXct(sprintf("%04.0f", flights$arr_time), format='%H%M'), 12, 16)

flights2$arr_date <- paste(flights$year, "-", flights$month, "-", flights$day, " ", flights2$arr_time2,

flights2$arr_date <- strptime(flights2$arr_date, format="%Y-%m-%d %H:%M")

flights$arr_date <- flights2$arr_date

#flights2 <- select(flights, -c(month, day, year, date))

flights2 <- flights

flights2$dep_date <- as.Date(flights2$dep_date)
flights2$arr_date <- as.Date(flights2$arr_date)
```

Test/Train Split:

```r
#Test/Train Split
trainIndex <- createDataPartition(flights2$arr_delay, p = .8,
                                  list = FALSE,
                                  times = 1)
train <- flights2[trainIndex,]
test <- flights2[-trainIndex,]
```

Simple Regression Model

```r
# Define training control
set.seed(123)
train.control <- trainControl(method = "cv", number = 10)

# Train the model
model <- train(arr_delay ~ air_time+carrier+origin+distance+dest+dep_date,data = flights2, method = "lm

#RMSE = 29.88867
print(model)
```

Random Forest (untuned):

```r
rf <- randomForest(
  arr_delay ~ air_time+carrier+origin+distance+dest+dep_delay+month,
  data = train
)
```

RMSE Calculation of RF Model:

```
test_pred_rf <- predict(rf, test)
train_pred_rf <- predict(rf, train)
full_set_rf <- predict(rf, flights2)

#RMSE of Test Set = 8.677777
test_rmse_untuned <- rmse(test$arr_delay, test_pred_rf)
#RMSE of Train Set = 8
train_rmse_untuned <- rmse(train$arr_delay, train_pred_rf)
#RMSE of Full Data = 7.382817
full_rmse_untuned <- rmse(flights2$arr_delay, full_set_rf)
```

```
print(paste("The RMSE of the untuned RF model is:", 7.382817))
```

```
## [1] "The RMSE of the untuned RF model is: 7.382817"
```

Describe the process you used to select your pipeline and improve it. This should include summaries of experiments you performed to evaluate potential improvements as well as a detailed description of your hyperparameter selection process.

The first Random Forest model left me with a pretty good RMSE of about 8.677 on the test data, but it requires lots of computational power, and I had to let it run overnight to complete the model. Thus, I decided to focus on optimizing run time, and to see if I could create a model with less prediction error.

I ended up building another OLS model using my test/train split, which achieved my initial goal of reducing run time and providing a slightly lower RMSE (8.03). Even though I didn't use a lot of features in my model, I wanted to try Stepwise regression, which provided the same model as my simple OLS model. I also tried Ridge and Lasso to see if I could identify unimportant features or to see if regularization would improve results, however they provided even worse RMSE values.

I did not decide to perform much exploratory analysis or assumptions testing on either the regression based or the random forest model. As the goal was simply maximizing predicitve accuracy, we do not care if the models meet the assumptions as long as they predict well.

Afterwards I ended up tuning the hyperparameters in my original Random Forest model, but I will describe that in further detail below after the code for the other regression based methods:

Simple Regression Model w test/train split

```
full_model <- lm(arr_delay ~ year+carrier+month+dep_delay + air_time + distance + origin + dest, data =

#RMSE

test_pred_ols <- predict(full_model, test)
train_pred_ols <- predict(full_model, train)
full_set_ols <- predict(full_model, flights2)

#RMSE of Test Set = 8.035658
rmse(test$arr_delay, test_pred_ols)
#RMSE of Train Set = 8.095134
rmse(train$arr_delay, train_pred_ols)
#RMSE of Full Data = 8.083275
rmse(flights2$arr_delay, full_set_ols)

plot(full_model$residuals)
```

Stepwise Model + RMSE:

```
step_model <- stepAIC(full_model, direction = "both", trace = FALSE)

summary(step_model)


#RMSE
test_pred_sw <- predict(step_model, test)
train_pred_sw <- predict(step_model, train)
full_set_sw <- predict(step_model, flights2)

#RMSE of Test Set = 8.035658
rmse(test$arr_delay, test_pred_sw)
#RMSE of Train Set = 8.095134
rmse(train$arr_delay, train_pred_sw)
#RMSE of Full Data = 8.083275
rmse(flights2$arr_delay, full_set_sw)
```

Ridge:

```
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 0)
# Display the best lambda value
cv$lambda.min

ridge <- glmnet(x, y, alpha = 0, lambda = cv$lambda.min)

# Make predictions on the test data
x.test <- model.matrix(arr_delay~air_time+carrier+origin+distance+dest+dep_delay+month, train)[,-1]
predictions <- ridge %>% predict(x.test) %>% as.vector()

#RMSE = 42.85495
RMSE(predictions, test$arr_delay)
```

Lasso:

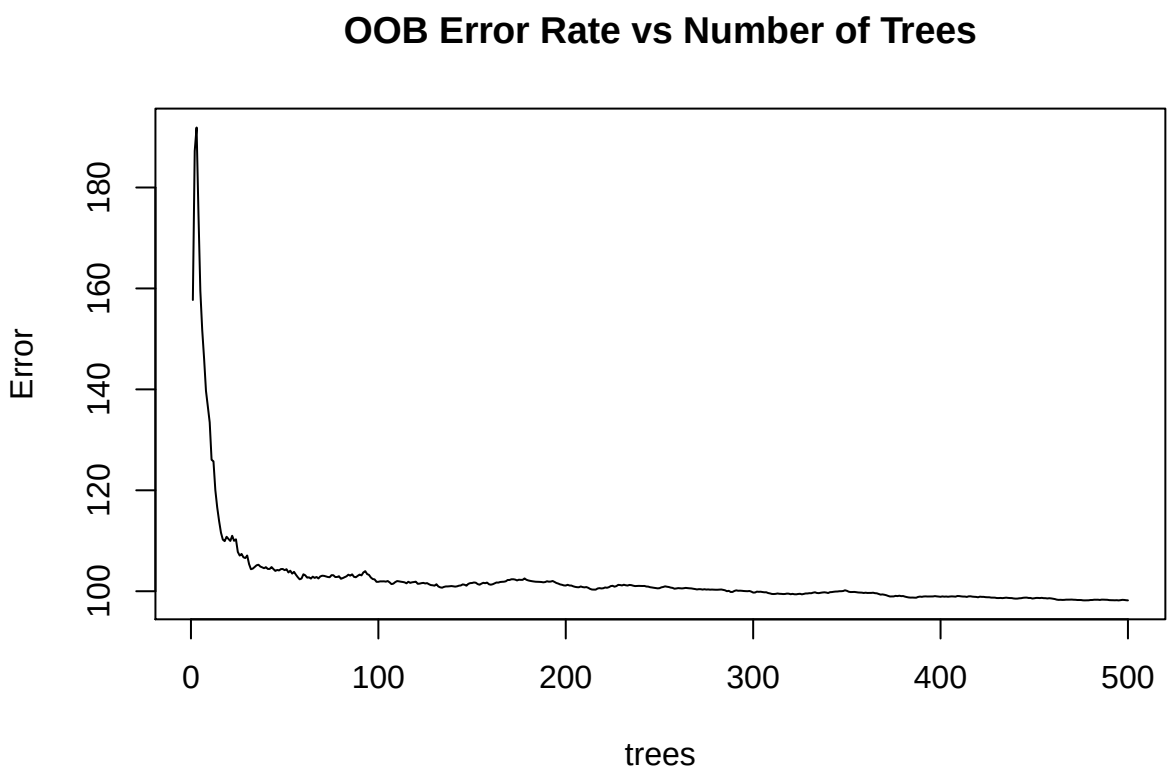
```
# Find the best lambda using cross-validation
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 1)
# Display the best lambda value
cv$lambda.min

# Fit the final model on the training data
lasso <- glmnet(x, y, alpha = 1, lambda = cv$lambda.min)

# Make predictions on the test data
x.test <- model.matrix(arr_delay~air_time+carrier+origin+distance+dest+dep_delay+month, train)[,-1]
predictions <- lasso %>% predict(x.test) %>% as.vector()

#RMSE = 42.86947
RMSE(predictions, test$arr_delay)
```

Hyperparameter selection

Tune the RF model:

Plot the original model:

```
load(file = "rfworkspace.RData")

plot(rf, main = "OOB Error Rate vs Number of Trees")
```



I wanted to plot the original model to see if error would be reduced after a specific number of trees. By setting a maximum number of trees, we would be able to reduce run time.

Next, I used the tuneRF function in the caret package to find the optimal MTRY value for my Random Forest. Mtry corresponds to the number of variables that can be sampled as candidates during each split of the tree. Obviously, this would change the way the forest is made, and choosing the optimal value will result in the best predictive accuracy of the random forest. I also realized that Random Forest is not subject to over-fitting like regression is, so there is no need to run the model only on training data like I did when making my previous model. I then reran my model using the whole data set, and was able to reduce the RMSE to 4.399224, almost half of the original Random Forest's RMSE.

```
#use parallel processing
doParallel::registerDoParallel()


#tuning parameters
set.seed(1)
```

```
y <- flights2$arr_delay
x <- model.matrix(arr_delay~air_time+carrier+origin+distance+dest+dep_delay+month, flights2)[,-1]

bestMtry <- tuneRF(x, y,
                   plot = TRUE, improve = 1e-5,
                   trace = TRUE, ntreeTry = 50)

rf_tuned <- randomForest(
  arr_delay ~ air_time+carrier+origin+distance+dest+dep_delay+month,
  data = flights2,
  ntree = 50,
  mtry = bestMtry
)
```

Tuned RF RMSE calculation:

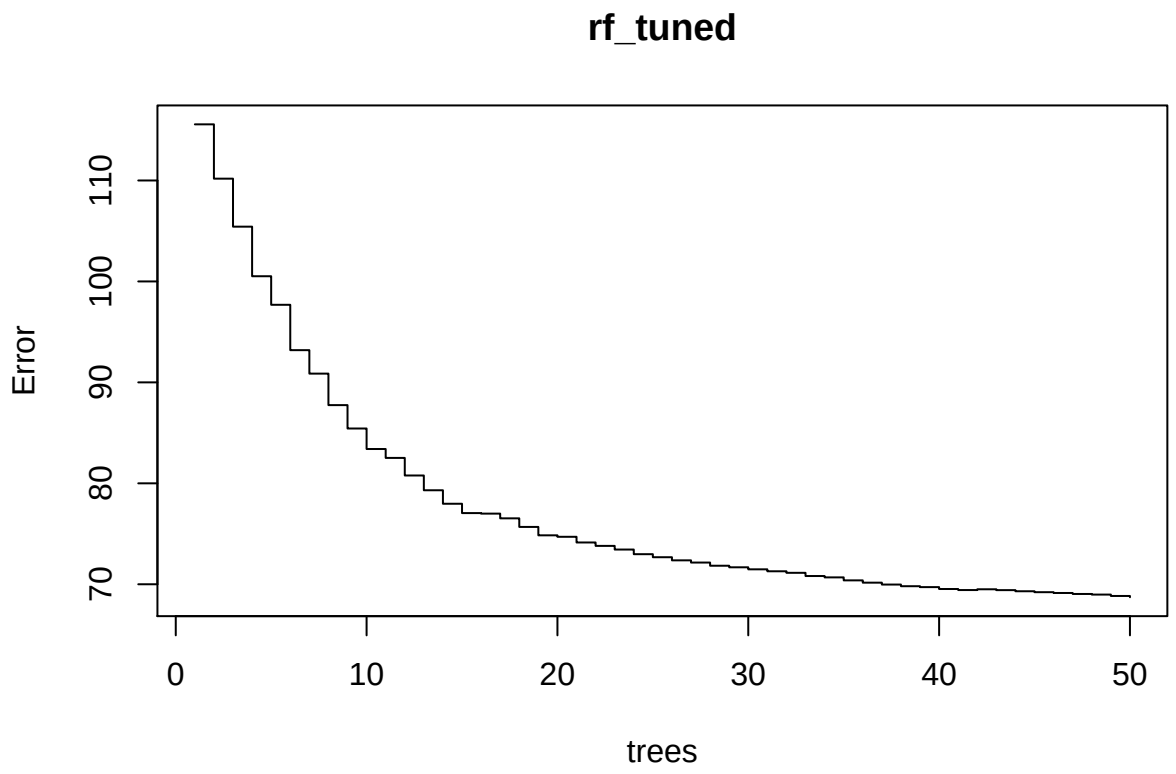
```
test_pred_rf_tuned <- predict(rf_tuned, test)
train_pred_rf_tuned <- predict(rf_tuned, train)
full_set_rf_tuned <- predict(rf_tuned, flights2)

#RMSE of Test Set = 4.639182
test_rmse <- rmse(test$arr_delay, test_pred_rf_tuned)
#RMSE of Train Set = 4.391177
train_rmse <- rmse(train$arr_delay, train_pred_rf_tuned)
#RMSE of Full Data = 4.385753
full_RMSE <- rmse(flights2$arr_delay, full_set_rf_tuned)

#RMSE when model run on just training data:

#RMSE of Test Set = 4.412082
#RMSE of Train Set = 4.396004
#RMSE of Full Data = 4.399224
```

```
print(paste("The RMSE of the tuned RF model is:", 4.385753))
```

```
## [1] "The RMSE of the tuned RF model is: 4.385753"
```

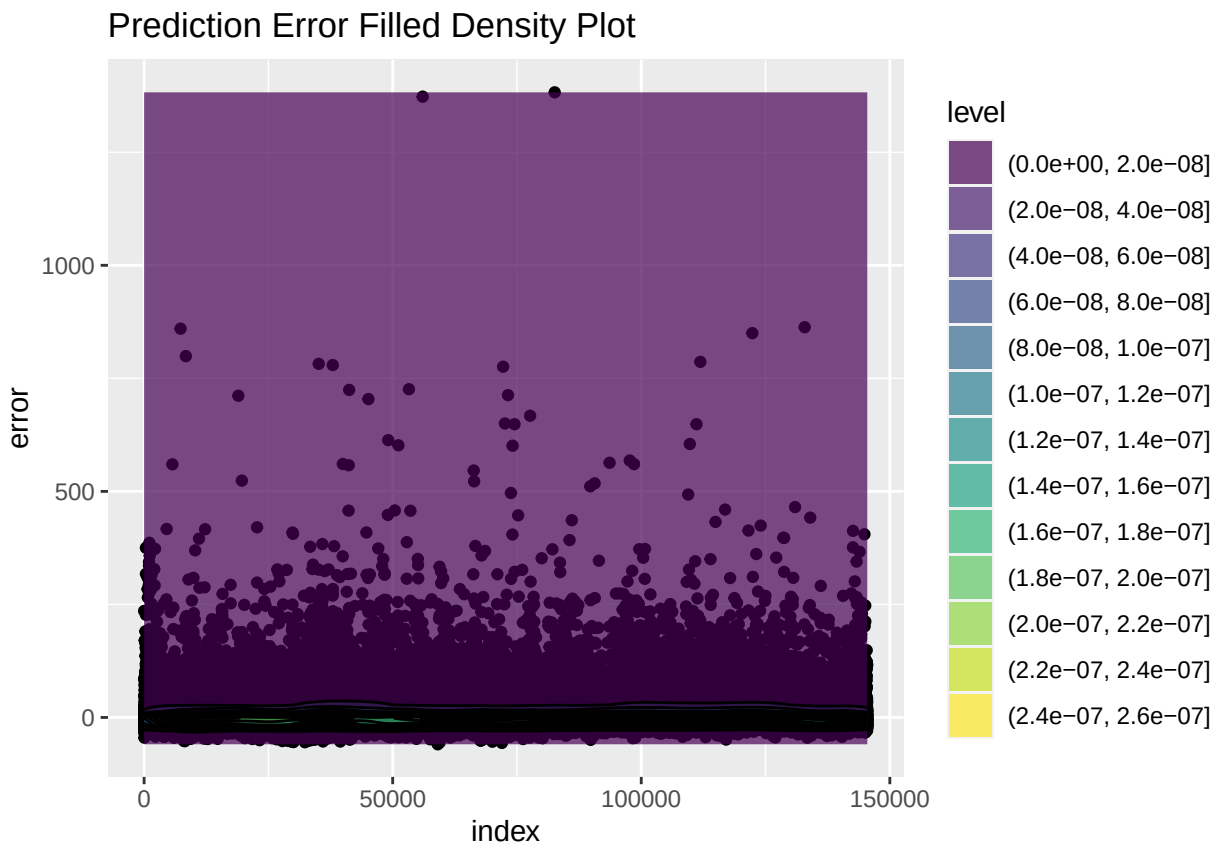
Plot of tuned model,

```
load(file = "tunedworkspace.RData")


plot(rf_tuned, type = "simple")
```



Prediction Error Plot, :

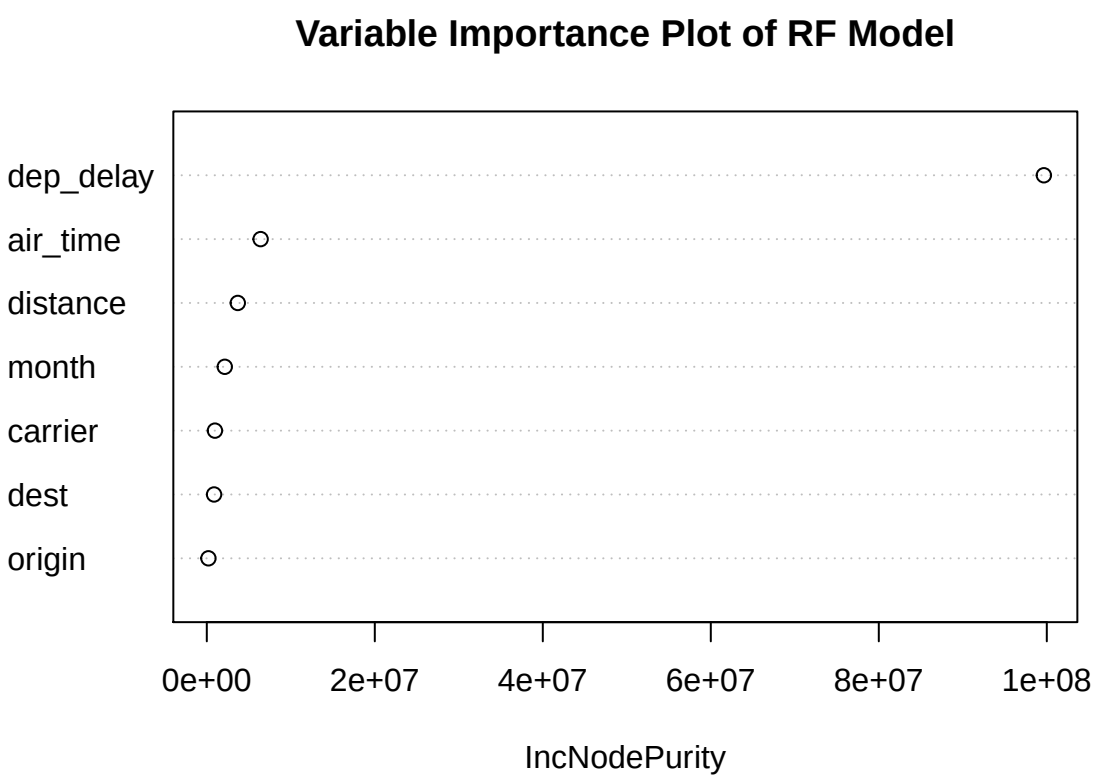
```
prediction_error <- data.frame(full_set_rf_tuned, rep(1:length(full_set_rf_tuned)))
names(prediction_error) <- c("error", "index")

prediction_error_plot <- ggplot(prediction_error, aes(x = index, y = error)) +
  geom_point() + geom_density2d_filled(alpha = 0.7) +
  geom_density2d(size = 0.5, color = "black") + ggtitle("Prediction Error Filled Density Plot")

prediction_error_plot
```



Above is a filled density plot of the error values of my Random Forest model, with the green lines corresponding to density bands. Most of the error values in the model are at the bottom, in the more black parts. This is good because it means that error is on average extremely low.

Variable Importance Plot:

```
varImpPlot(rf_tuned, main = "Variable Importance Plot of RF Model")
```



Above is a variable importance plot. It shows departure delay as the most important factor in prediction arrival delay, with air time being the second most important and origin being the least. This intuitively would make sense. This plot would be more useful in a situation where I had lots of variables and I wanted to make a new forest with a lower runtime, while still retaining high predictive accuracy. Since there were not that many variables, I did not think it was necessary to reduce the number of variables used in my model.

Describe the additional data you used, how you gathered it, and what modifications you made to your pipeline to use it.

I decided to use some weather data, as weather is often one of the biggest factors in causing flight delays. I obtained data on weather in Portland and Seattle from the National Center for Environmental Information, and this data includes min and max temperature, snow status and wind.

Link to data:

https://www.ncdc.noaa.gov/cdo-web/search

Add external data to dataframe:

```
seattle_weather <- read.csv("seattle.csv")
portland_weater <- read.csv("portland.csv")

weather <- rbind(portland_weater, seattle_weather)

weather$DATE <- as.Date(weather$DATE, format="%m/%d/%Y")

weather$origin <- ifelse(weather$STATION == "USW00024229", "PDX", "SEA")

colnames(weather)[which(names(weather) == "DATE")] <- "departure_date"
```

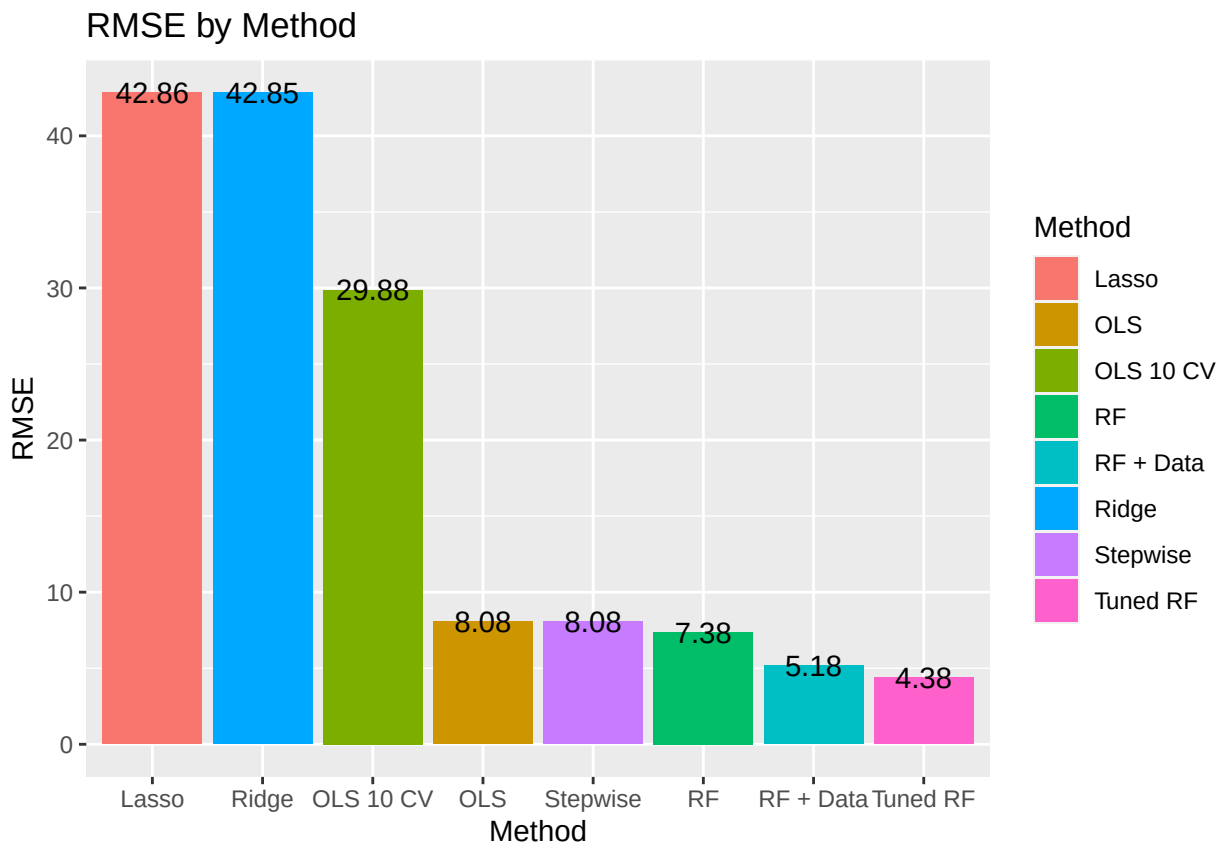
```r
flights2$departure_date <- as.Date(flights2$departure_date)

#merge data
flights2 <- merge(flights2, weather, by=c("origin", "departure_date"), all.x = T)

#make Snow a factor so model knows it is not an integer
flights2$SNOW <- as.factor(flights2$SNOW)
```

Rerun model with extra data added:

```r
rf_tuned_extra <- randomForest(
  arr_delay ~ air_time+carrier+origin+distance+dest+dep_delay+month+TMAX+TMIN+SNOW,
  data = flights2,
  ntree = 50,
)

full_set_rf_tuned_extra <- predict(rf_tuned_extra, flights2)
full_RMSE_extra <- rmse(flights2$arr_delay, full_set_rf_tuned_extra)


print(paste("The RMSE of the tuned RF model with extra data is:", full_RMSE_extra))
```

Perform a basic ablation analysis. At the very least, compare your model with "pipeline improvements and external data" to "pipeline improvements only," "external data only," and "no improvements and no external data."

```r
ablation_df <- data.frame(
  method <- c("OLS 10 CV", "RF", "OLS", "Stepwise", "Ridge", "Lasso", "Tuned RF", "RF + Data"),
  rmse <- c(29.88, 7.38, 8.08, 8.08, 42.85, 42.86, 4.38, 5.18)
)

names(ablation_df) <- c("Method", "RMSE")


ablation_df %>%
  arrange(desc(rmse)) %>%
  ggplot(aes(x = reorder(Method, -RMSE), y = RMSE, fill = Method)) +
  geom_bar(stat = "identity") + ggtitle("RMSE by Method") +
  geom_text(aes(label = RMSE)) + xlab("Method")
```



The bar graph above lists the RMSE values calculated for every model that I made. All of the regressions (except the OLS 10 fold CV) calculated the RMSE using the testing set. Since Random Forest is not as prone to over fitting as regression, I used the whole data set when training my model, and calculated the RMSE using the entire model.

The two best models were my Random Forest models after tuning the hyper parameters. My model without the extra data had the lowest RMSE at 4.38, compared to my Lasso model which was the worst at 42.86. Since my random forest model had the lowest prediction error, it has the best predictive accuracy. Since adding the weather data caused the RMSE to increase, I did not include the extra data in my final model, as I would not want to increase error.

Justify your choice of overall pipeline and external data so that a (non-expert) practitioner could understand why you made these choices. This justification could reference the outcomes of the experiments in point 'b' and ablation analysis in point 'd'.

I choose my random forest model because it had the highest prediction accuracy, and after tuning the hyperparameters had among the lowest run times. Simple OLS had the lowest run time at around ten minutes, but it had almost 9 times the error.

Random Forest is a useful algorithm because it can be run on both categorical and nominal data, and generates random pairings (trees) to predict the outcome variable. The best trees are selected to put into the final model. Tuning the mtry gave a slight drop in RMSE from the first random forest model to the second (~7 vs ~4). Additionally, limiting the number of trees to 50 decreased the run time of the model from 10 hours to about 30 minutes, a significant improvement. I also chose to omit the extra weather data from my final model since it increased prediction error.

Propose concrete and meaningful modifications or extensions to your solution.

Random Forest's run time is significantly limited by the number of variables, and I noticed that trying to run the model on the whole data set gave me lots of errors. Usually, using more data allows one to fit a

better model, so I would like to experiment with other models that allow one to use more variables. Perhaps using variables like tail number might allow us to factor on delays associated with a single plane that tends to need repair a lot. Since my Stepwise regression was able to achieve an impressive RMSE of around 8 with a smaller run time than my tuned random forest, it might be worth exploring adding extra variables to another regression based method to see if I can further improve the model.

Additionally, I would like to add more external data that happens to have a decreasing effect on RMSE. Maybe there are other variables associated with flight delays that are not in this data set, that would help us better predict flight delay.

Add Kaggle Data and generate kaggle submission:

```
test_kaggle <- read.csv("G:/My Drive/STAT 3105/HW4/test_student.csv")


#add back dep_date col
test_kaggle$date <- paste(test_kaggle$year, "-", test_kaggle$month, "-", test_kaggle$day, " ", test_kagg
test_kaggle$dep_date <- strptime(test_kaggle$date, format="%Y-%m-%d %H:%M")
test_kaggle$dep_date <- as.Date(test_kaggle$dep_date)
```

Generate RF Kaggle submission:

```
submission <- predict(rf_tuned, test_kaggle)



submission_df <- data.frame(test_kaggle$Id)
submission_df$arr_delay <- submission
names(submission_df) <- c("Id", "arr_delay")

write.csv(submission_df, "G:/My Drive/STAT 3105/final/submission.csv", row.names = FALSE)
```