Predicting Indian Airline Fares

Nazhah Mir

Introduction

Imagine wanting to book a flight and knowing how much the ticket will cost without even checking the website. The purpose of this project is to create a successful machine learning model that can accurately predict the price of a plane ticket.

Why?

The aviation industry, and India's in particular, has been booming in recent years. However, with every growing industry comes the issue of fluctuating prices. Why is it that when you look for a flight today, the price is different than it was yesterday? What factors go into determining the price of an airplane ticket? Using machine learning, we can figure out what exactly is the most influential in dictating price. With more accurate price predictions, we will know exactly how much we should be paying for a flight.

Reading in the Data

```r
airplane_data <- read_excel('IndianAirfareData.xlsx')
airplane <- as_tibble(airplane_data)
```

The data was taken from Kaggle Set “Flight Fare Prediction MH” and it was posted by user Nikhil Mittal, who got it from the Indian Data Science website MachineHack.

```r
dim(airplane)
```

```
## [1] 10683    11
```

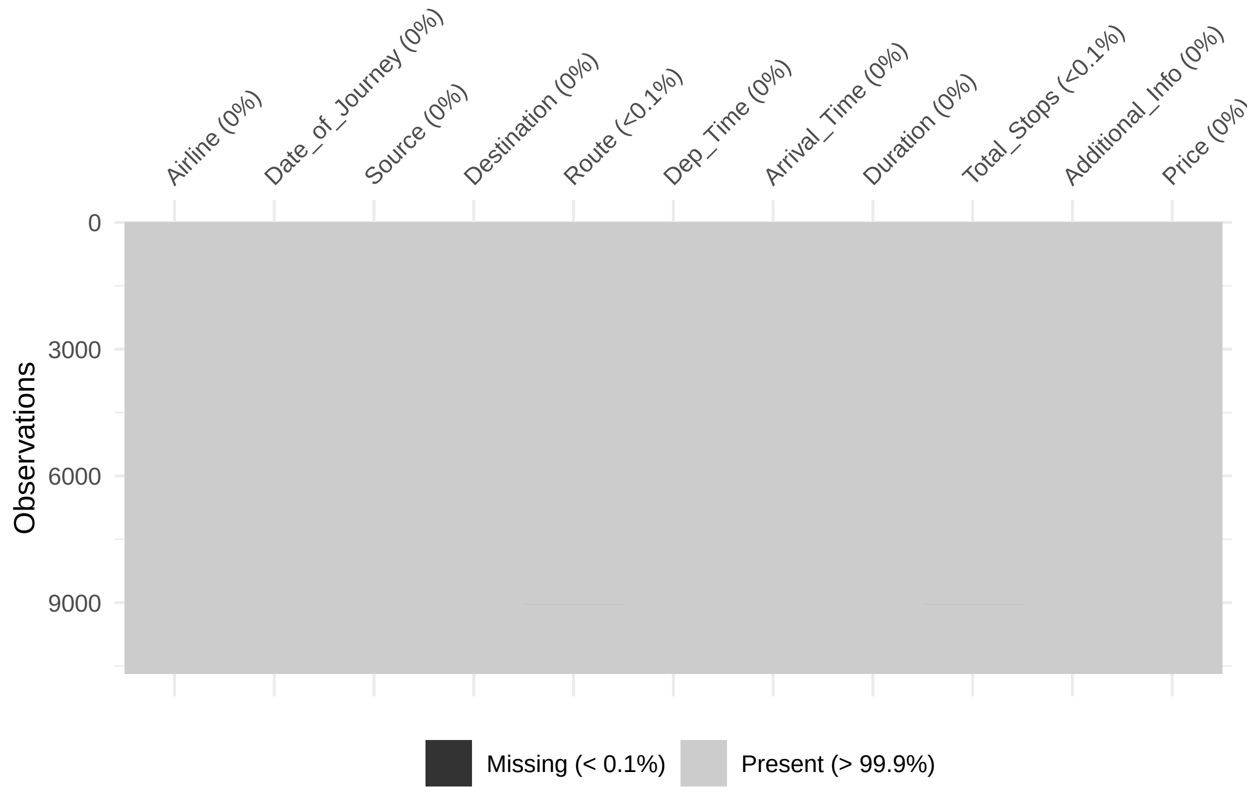
There are 11 variables, including the outcome variable airfare, and 10,683 total observations.

Cleaning the Data

Missing observations

Before we begin doing anything with the data, we must visualize if there are observations that are missing values for certain variables.

```r
vis_miss(airplane)
```



Since less than 0.1% of the data is missing, it would just be easier to drop those observations.

```
airplane <- airplane %>%
  drop_na()
```

Only one observation had missing data, so we were able to easily drop that without affecting the overall dataset, still leaving us with a whopping 10,682 obeservations!

Data Cleaning

Let us take a look at what type of variables, numerical or categorical, are being used to predict airfare.

```
str(airplane)
```

```
## tibble [10,682 x 11] (S3: tbl_df/tbl/data.frame)
##  $ Airline        : chr [1:10682] "IndiGo" "Air India" "Jet Airways" "IndiGo" ...
##  $ Date_of_Journey: chr [1:10682] "24/03/2019" "1/05/2019" "9/06/2019" "12/05/2019" ...
##  $ Source         : chr [1:10682] "Banglore" "Kolkata" "Delhi" "Kolkata" ...
##  $ Destination    : chr [1:10682] "New Delhi" "Banglore" "Cochin" "Banglore" ...
##  $ Route          : chr [1:10682] "BLR → DEL" "CCU → IXR → BBI → BLR" "DEL → LKO → BOM → COK" "CCU →
##  $ Dep_Time       : chr [1:10682] "22:20" "05:50" "09:25" "18:05" ...
##  $ Arrival_Time   : chr [1:10682] "01:10 22 Mar" "13:15" "04:25 10 Jun" "23:30" ...
##  $ Duration       : chr [1:10682] "2h 50m" "7h 25m" "19h" "5h 25m" ...
##  $ Total_Stops    : chr [1:10682] "non-stop" "2 stops" "2 stops" "1 stop" ...
##  $ Additional_Info: chr [1:10682] "No info" "No info" "No info" "No info" ...
##  $ Price          : num [1:10682] 3897 7662 13882 6218 13302 ...
```

All variables, with the exception of `Price`, are character variables, even the ones with numerical values. That means that we are going to have to perform some data manipulation so that the data is easier to visualize and use.

Data Manipulation

Let us start with the `Date_of_Journey`. To be able to use this variable in our model, we must separate the month from the day that the flight took place. As this dataset only utilizes data from 2019, we can omit the year, as that will have no effect on the airfare. We can then drop the date of journey column, as this will now serve us no purpose. For the following code chunks, we will be using the POSIX function for date-time conversions.

```
airplane$Journey_day <- as.POSIXlt(airplane$Date_of_Journey,
    format = "%d/%m/%Y")$mday
airplane$Journey_month <- as.POSIXlt(airplane$Date_of_Journey,
    format = "%d/%m/%Y")$mon + 1  #to account for r indexing starting at 0
airplane <- subset(airplane, select = -c(Date_of_Journey))
```

Now, let us extract the hour and minute times from both the departure and arrival times of the flights.

```
dep_time_posix <- as.POSIXlt(airplane$Dep_Time, format = "%H:%M")

# Extract Hour and Minute
airplane$Dep_hour <- dep_time_posix$hour
airplane$Dep_min <- dep_time_posix$min
airplane <- subset(airplane, select = -c(Dep_Time))

arrival_time_posix <- as.POSIXlt(airplane$Arrival_Time, format = "%H:%M")

# Extract Hour and Minute
airplane$Arrival_hour <- arrival_time_posix$hour
airplane$Arrival_min <- arrival_time_posix$min
airplane <- subset(airplane, select = -c(Arrival_Time))
```

The flight duration is also a character string, written in the form __h __m. We must now subset the hours and minutes and convert them to numeric values in order to ensure that we can incorporate them in our model.

```
duration <- airplane$Duration

for (i in seq_along(duration)) {
  if (!grepl("m", duration[i])) {
    duration[i] <- paste0(duration[i], " 0m")
  }
}

duration_hours <- numeric(length(duration))
duration_mins <- numeric(length(duration))

for (i in seq_along(duration)) {
  duration_split <- strsplit(duration[[i]], " ")[[1]]
  duration_hours[i] <- as.numeric(sub("h", "", duration_split[1]))
  duration_mins[i] <- as.numeric(sub("m", "", duration_split[2]))
}
```

```
## Warning: NAs introduced by coercion
```

```
airplane$Duration_hours <- duration_hours
airplane$Duration_mins <- duration_mins
airplane <- subset(airplane, select = -c(Duration)) %>%
  drop_na()
```

Encoding the Variables

Lastly, we will use different types of encoding so that we can easily analyze the effects of explicitly categorical variables, such as `Total_Stops`, `Airline`, `Source`, and `Destination`. We will additionally be categorizing the hour of the day that the flight took place as time of day so that it will be a more generalized way of viewing the relationship between the time of day that the flight took off and the price of the ticket.

First, we will start with the airlines. Some of the airlines have a premium/business classification, which means that they will be quite a bit more expensive. Let us see how many of each airline there are.

```
table(airplane$Airline)
```

```
##
##                          Air Asia                          Air India
##                               319                               1750
##                             GoAir                             IndiGo
##                               194                               2053
##                       Jet Airways               Jet Airways Business
##                              3849                                  6
##                 Multiple carriers Multiple carriers Premium economy
##                              1196                                 13
##                          SpiceJet                             Trujet
##                               818                                  1
##                           Vistara           Vistara Premium economy
##                               479                                  3
```

Here, we can see that Jet Airways Business has 6 observations, Multiple Carriers Premium economy has 13 observations, and Vistara Premium economy has 3 observations, and that Business and Premium class tickets of certain airlines have been categorized into separate airlines. Since there are not that many observations but we still want to observe the impact that a premium classification makes, we will create a new column, airline_category, to specify whether a ticket is premium or regular.

```
airline_category <- ifelse(grepl("business", airplane$Airline, ignore.case = TRUE) |
                             grepl("premium", airplane$Airline, ignore.case = TRUE),
                            "Premium", "Regular")

airplane$Airline_category <- airline_category
```

We will then classify the Premium and Business airline categories into their respective airlines.

```
premium_pattern <- "Premium economy"
business_pattern <- "Business"

# Replace premium and business patterns with the main airline name
airplane$Airline <- gsub(premium_pattern, "", airplane$Airline)
airplane$Airline <- gsub(business_pattern, "", airplane$Airline)
airplane$Airline <- trimws(airplane$Airline)
table(airplane$Airline)
```

```
##
##          Air Asia         Air India             GoAir            IndiGo
##               319              1750               194              2053
##       Jet Airways Multiple carriers          SpiceJet            Trujet
##              3855              1209               818                 1
##           Vistara
##               482
```

Addionally, since Trujet only has 1 observation, removing it will not make that much of a difference to the predictive power and might actually make our model more stable.

```r
airplane <- airplane[airplane$Airline != "Trujet", ]
```

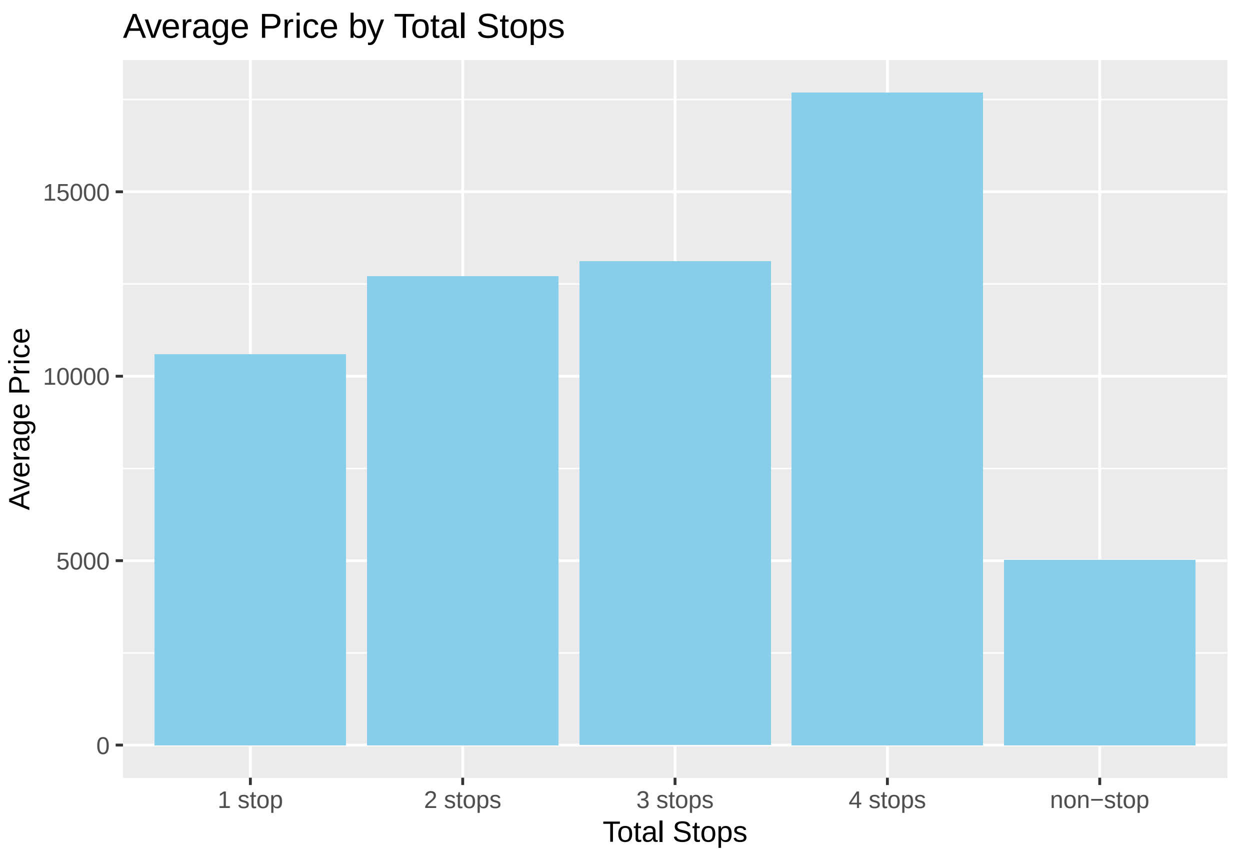
Next, we will start with number of stops, or the number of layovers in a certain flight.

```r
table(airplane$Total_Stops)
```

```
##
##   1 stop  2 stops  3 stops  4 stops non-stop
##     5624     1519       45        1     3491
```

Here, we can see that we have 5 unique values of stops, ranging from 0 stops (non-stop flight) to 4 stops. Since these are technically numerical values, we will be using encoding so that we can incorporate the number of stops into our model and visualize the relationship. Let us create a plot to see if price increases as the number of stops increases to see whether label or one-hot-encoding should be used.

```r
average_price <- aggregate(Price ~ Total_Stops, data = airplane, FUN = mean)

ggplot(average_price, aes(x = Total_Stops, y = Price)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(x = "Total Stops", y = "Average Price") +
  ggtitle("Average Price by Total Stops")
```



Through this bar graph, we can see that as the number of stops increases, so does the average price of a ticket. This is an indication that the number of stops during a flight route is influential in determining the price of that ticket, and that we should be using label encoding, as that would preserve the ordinal relationship exhibited by the graph. Furthermore, since `Route` and `Total_Stops` essentially encompass the same data, we can drop `Route`, since it is hard to convert that variable into data that the model can understand. We can also drop the `Additional_Info` column, as most of the observations for this column do not contain any data.

```
airplane$Total_Stops <- gsub("non-stop", 0, airplane$Total_Stops)
airplane$Total_Stops <- gsub("1 stop", 1, airplane$Total_Stops)
airplane$Total_Stops <- gsub("2 stops", 2, airplane$Total_Stops)
airplane$Total_Stops <- gsub("3 stops", 3, airplane$Total_Stops)
airplane$Total_Stops <- gsub("4 stops", 4, airplane$Total_Stops)

airplane$Total_Stops <- as.numeric(airplane$Total_Stops)

airplane <- subset(airplane, select = -c(Route, Additional_Info))
```

Lastly, we will use one-hot encoding to categorize the departure and arrival times of flights. For this model, I am choosing to omit the arrival times, as it would be too closely related to both the departure time and the duration time and would lead to redundancy and multicollinearity in the model. We can then drop the variables for arrival hours, departure hours, arrival minutes, and departure minutes, as leaving these would also introduce multicollinearity, leading to a less reliable model.

```
airplane$Departure_time <- ifelse(airplane$Dep_hour >= 5 & airplane$Dep_hour < 12, "Morning",
ifelse(airplane$Dep_hour >= 12 & airplane$Dep_hour < 17, "Afternoon",
airplane <- subset(airplane, select = -c(Dep_hour, Dep_min, Arrival_hour, Arrival_min))
```

Now that the data is cleaned up, we can finally convert all of the categorical variables into factors that will be used in the recipe later on.

```
airplane <- airplane %>%
    mutate(Airline = factor(Airline), Source = factor(Source),
        Destination = factor(Destination), Departure_time = factor(Departure_time,
            levels = c("Morning", "Afternoon", "Evening", "Night")),
        Airline_category = factor(Airline_category, levels = c("Regular",
            "Premium")))
```

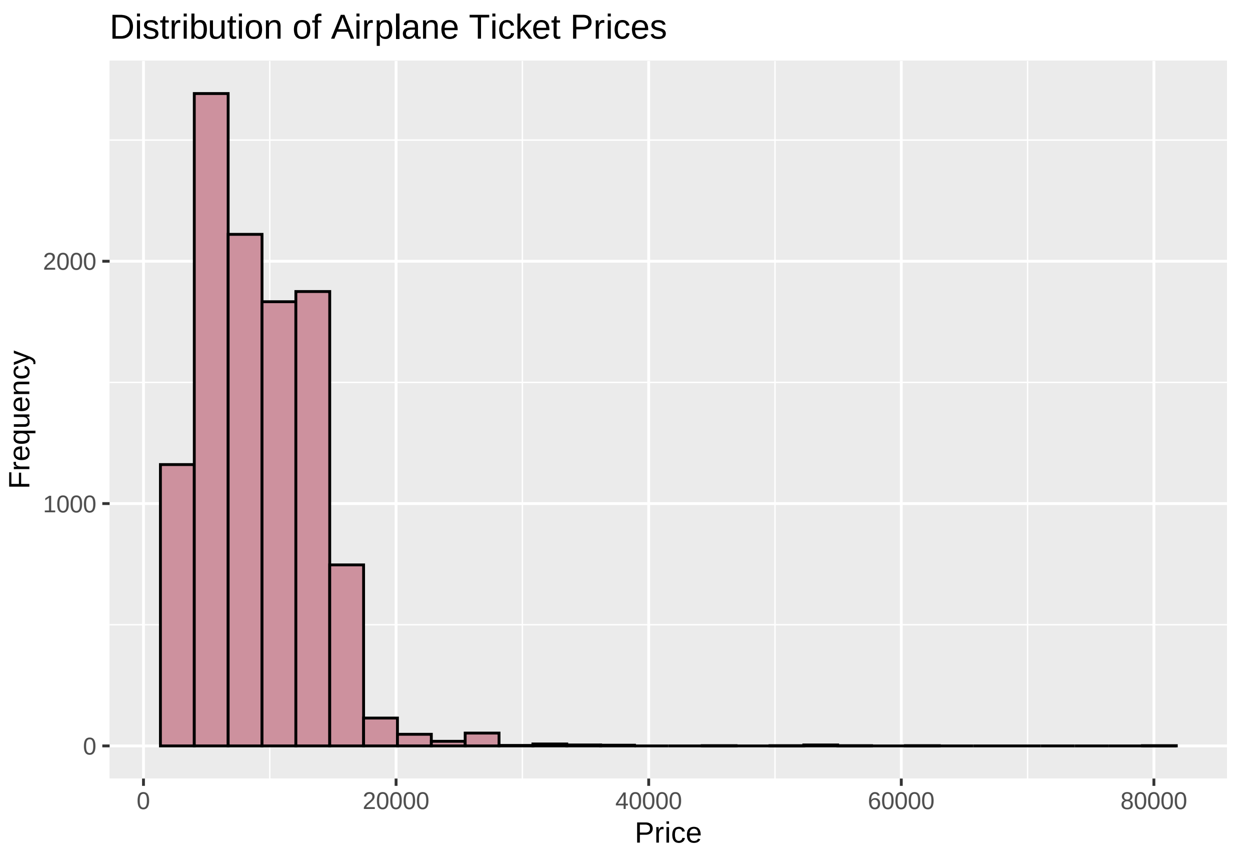
Visual EDA

Now, we will explore the relationship between select predictor variables with each other and with the response variable.

Price

First, let us analyze the distribution of our outcome variable, price.

```
price_hist <- ggplot(airplane, aes(x = Price)) +
  geom_histogram(fill = 'pink3', color = 'black', bins = 30) +
  labs(
    title = "Distribution of Airplane Ticket Prices",
    x = "Price",
    y = "Frequency"
  )
price_hist
```

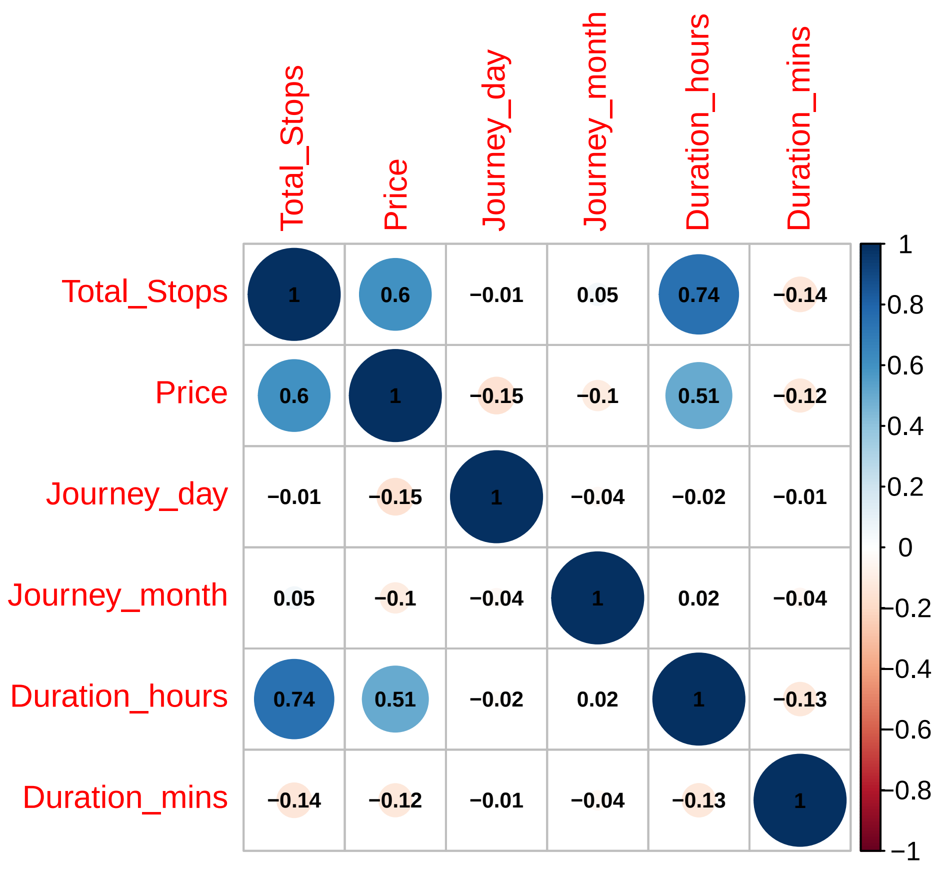


The minimum price of an airplane ticket was 1,759 rupees, and the maximum price was 79,512 rupees. The price of most of the airplane tickets fell between 2681.14 and 5362.28 rupees.

Numerical Variable Analysis

To depict the relationship between all of the numerical variables, we can construct a heatmap.

```r
airplane %>%
  select(where(is.numeric)) %>%
  cor(use="pairwise.complete.obs") %>%
  corrplot(addCoef.col = 1, number.cex = 0.7)
```



From analyzing this correlation plot, it seems as if most of the predictor variables do not have much of a correlation with each other or the price. The only clear relationship between predictor variables exhibited by the plot are between the total number of stops and the flight duration in hours. This would make sense, as the more stops you have, the more time you would spend waiting for your connecting flight, increasing the total flight time. Additionally, these two variables are the only ones that have any sort of correlation with price. Total stops has a fairly moderate correlation with price at 0.51, and the duration of the flight has the strongest correlation at 0.74. This gives a hint that a linear regression model might not be the best fit for this data.

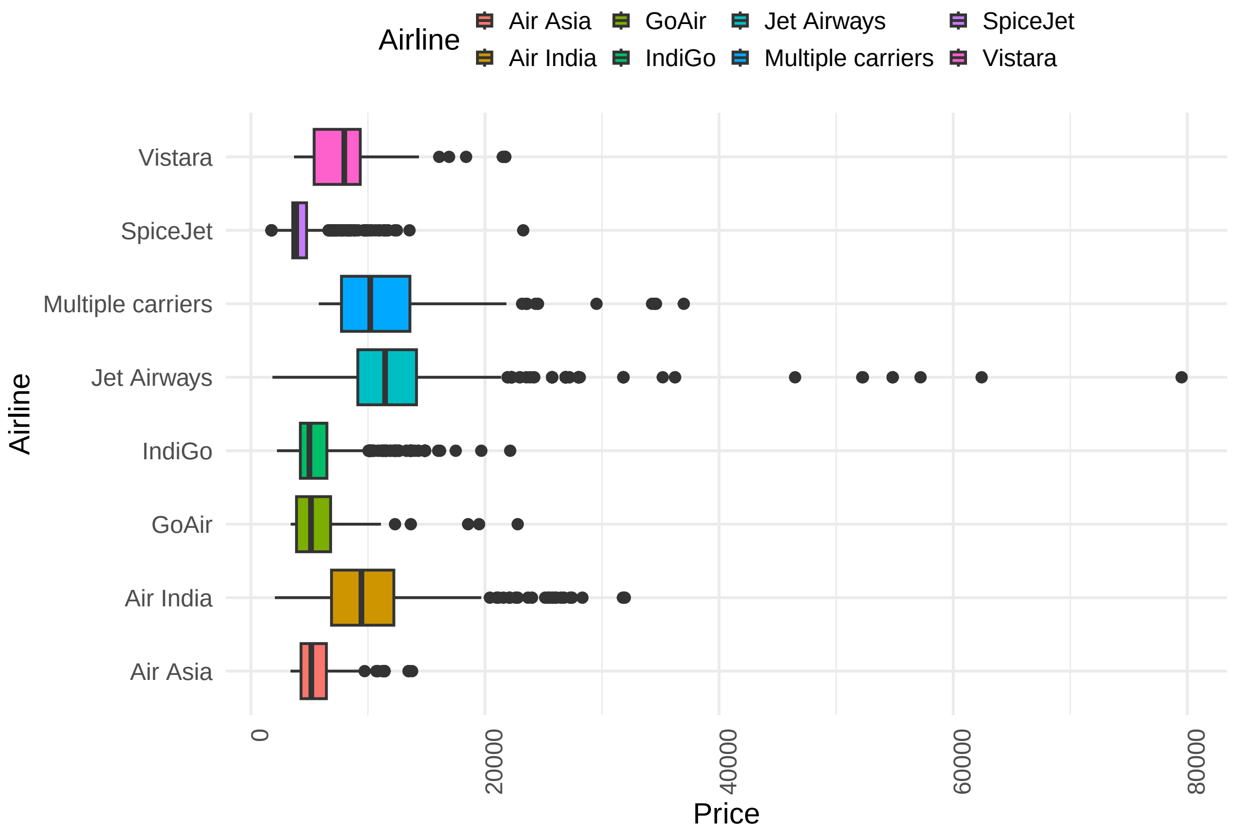
Select Categorical Variables

Now, we will visualize the data for the categorical variables using a boxplot to see if any of these factors might individually have an impact on the price that we weren't able to visualize through the heatmap.

Airline

There are 12 unique categories of airlines that have data available. This includes a single ticket using multiple airlines and a premium/business class specification for two airlines and for the ticket using multiple airlines. Let's visualize the relationship between Airline and the price of a ticket.

```
sorted_data <- airplane[order(airplane$Price, decreasing = TRUE), ]
ggplot(sorted_data, aes(x = Airline, y = Price, fill = Airline)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "Airline", y = "Price") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        legend.position = "top",
        legend.key.size = unit(0.5, "lines"))
```

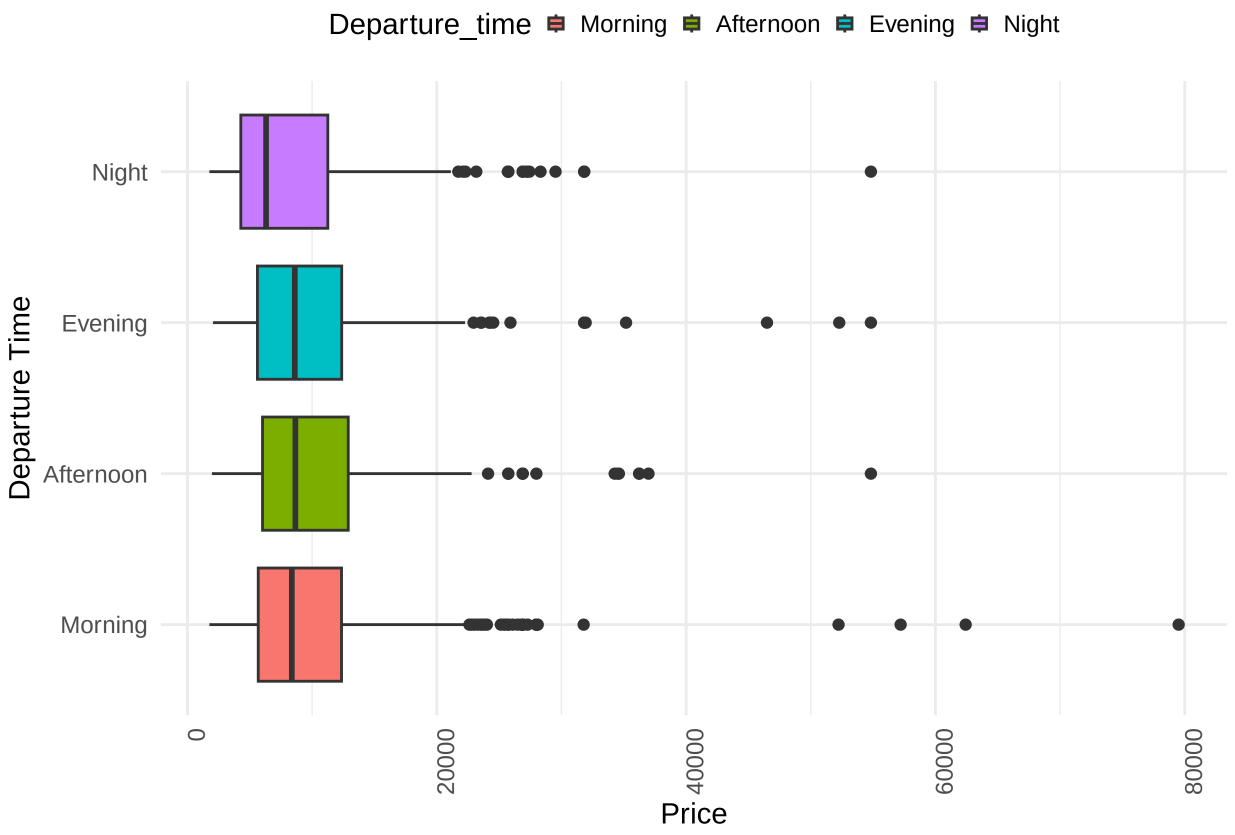


Here, we can see that there is a clear variation in the distribution of the prices depending on the airline, which indicates a clear correlation between airline and price.

Time of Day

Next, let us determine whether or not the time of day the flight takes off has any noticeable influence on the ticket price.

```
ggplot(sorted_data, aes(x = Departure_time, y = Price, fill = Departure_time)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "Departure Time", y = "Price") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        legend.position = "top",
        legend.key.size = unit(0.5, "lines"))
```

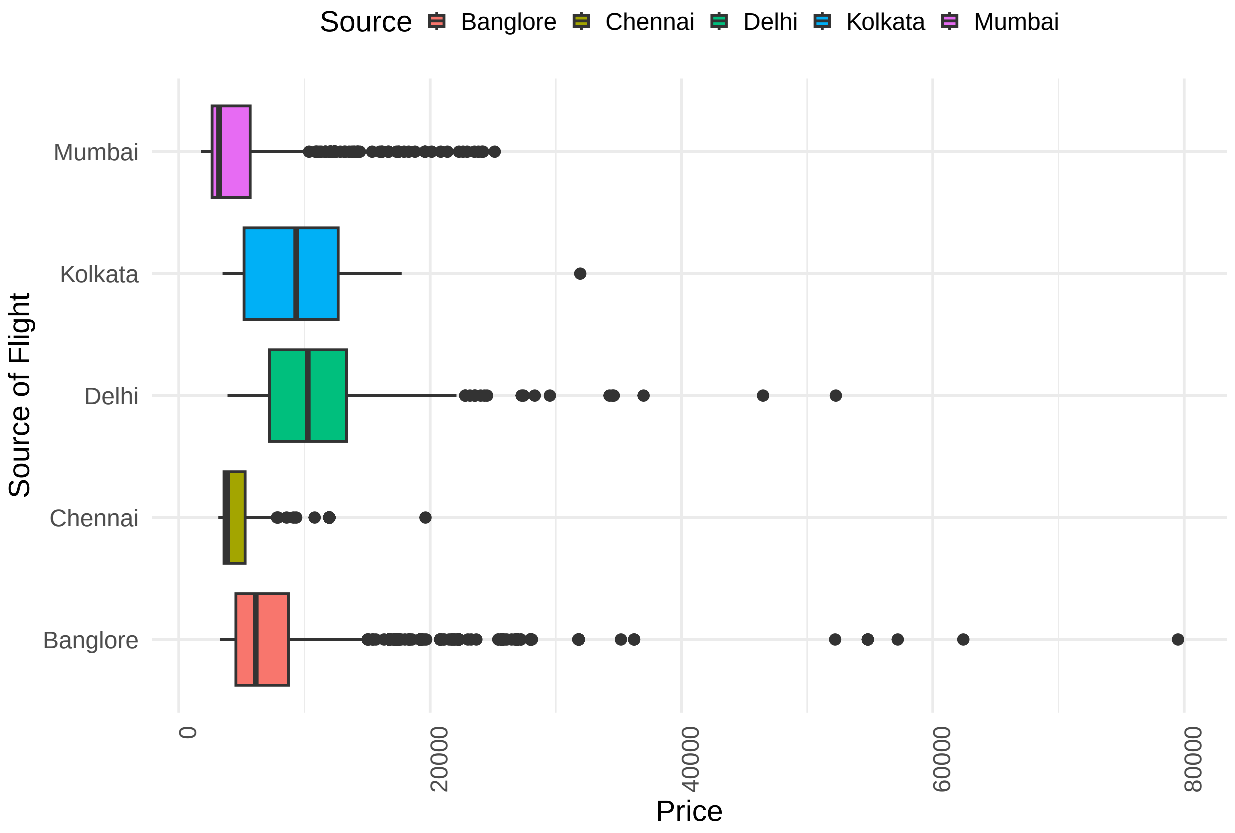


Looking at the box plot, we can see that the median prices of morning, evening, and afternoon are all fairly equal, with the prices of flights departing at night being slightly less than the others. This means that, with the exception of nighttime, the time of day does not have much influence on the price.

Source and Destination

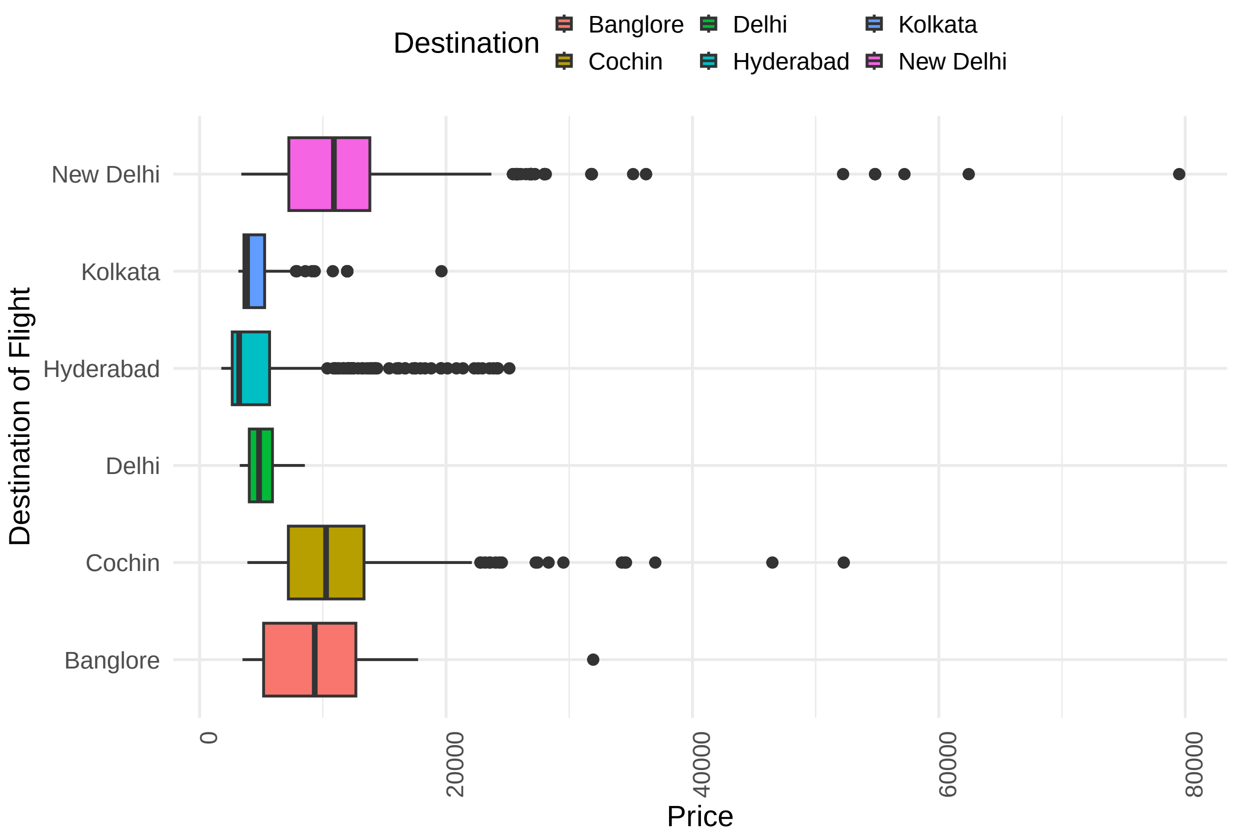
It is also important to see whether the intial source of a flight and its final destination have any clear effect on the price. We can visualize this using 2 box plots for each variable.

```
ggplot(sorted_data, aes(x = Source, y = Price, fill = Source)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "Source of Flight", y = "Price") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        legend.position = "top",
        legend.key.size = unit(0.5, "lines"))
```



Here, the median prices for all of the flights are fairly close together, with Chennai and Mumbai having a lower median than the other cities. Additionally, Banglore seems to have the most spread out outliers, followed by Delhi, indicating that there are other factors besides the source location of the flight that affect the ticket price.

```
ggplot(sorted_data, aes(x = Destination, y = Price, fill = Destination)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "Destination of Flight", y = "Price") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        legend.position = "top",
        legend.key.size = unit(0.5, "lines"))
```



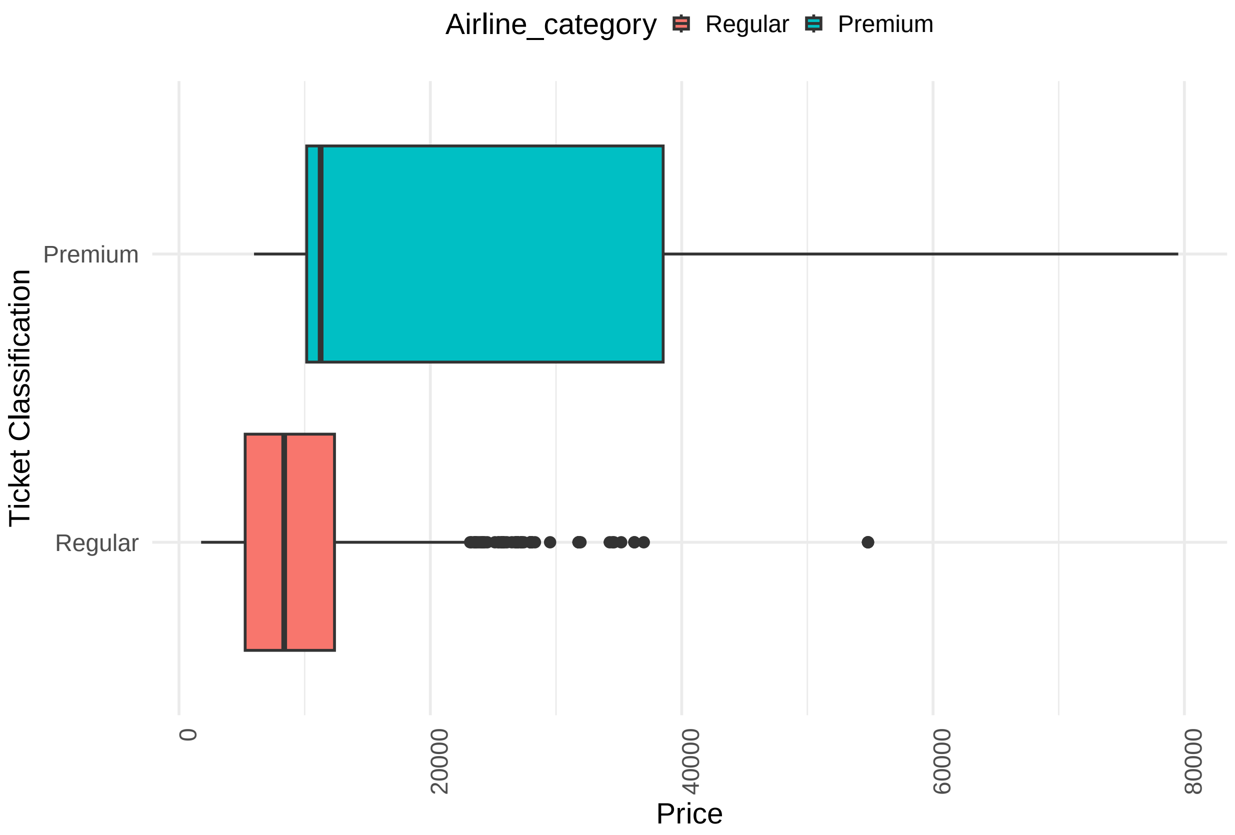
Through this plot, it is easy to visualize that flights to Banglore, Cochin, and New Delhi all have higher median prices than to Kolkata, Hyderabad, and Delhi. Additionally, the outliers for New Delhi are a lot more spread out than those for the other cities.

For both the Source and Destination variables, the median price of the plane ticket stays relatively the same regardless of differences in the city, so it is highly likely these variables do not have as much influence on the ticket price as the others. However, since some cities are more expensive to fly out of and to than others, we will still be keeping these variables, as they do still have some control over the price.

Premium tickets

Lastly, we want to visualize the potential impact that a Premium/Business class ticket has on the price. We can do this using one more box plot.

```r
ggplot(sorted_data, aes(x = Airline_category, y = Price, fill = Airline_category)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "Ticket Classification", y = "Price") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        legend.position = "top",
        legend.key.size = unit(0.5, "lines"))
```



Here, we can see that buying a premium or a business class ticket does make a significant impact on the price of the airline ticket. The median of the Premium category is slightly higher than that of the Regular category, and the Premium box plot is spread out amongst larger values than the Regular box plot. However, the Regular category does have a lot more outliers on the higher end of the price, which are not present in the Premium category, indicating that there are other, more important factors that play a role in influencing price.

Setting Up the Models

Now that we have the data cleaned up and a slight idea of what variables are better predictors of ticket price, we are almost ready to create our models. But first, we have a couple steps we need to take in order to ensure our model is the best it can be.

Splitting the Data

First, we will be splitting the data into separate datasets, a training test and a testing set. The purpose of this is to train the model on a dataset separate from the one you are actually trying to predict in order to be able to assess how well the model performs on new, unseen data. Using the same training set for different models provides us with an objective ground to determine which model performs the best.

```r
set.seed(1596)
airplane_split <- initial_split(airplane, strata = Price, prop =.7)
airplane_train <- training(airplane_split)
airplane_test <- testing(airplane_split)
```

Let us check the dimensions of both the training set and the testing set to ensure that the data was split correctly.

```
nrow(airplane_train)/nrow(airplane)
```

```
## [1] 0.6999064
```

```
nrow(airplane_test)/nrow(airplane)
```

```
## [1] 0.3000936
```

And it was! Thet training set has 70% of the data and the training set has 30%.

Creating the Recipe

To create the models, we will be using a universal recipe that utilizes all of the predictors and the response variables. This will make it a lot easier to reproduce the data workflow and make sure that the same data transformations are applied. We're using the same 11 variables that we solidified during the exploratory data analysis: `Airline`, `Source`, `Destination`, `Total_Stops`, `Journey_day`, `Journey_month`, `Duration_hours`, `Duration_mins`, `Airline_category`, `Departure_time`, and of course `Price`. First, we will start off by imputing the variables `Duration_hours` and `Duration_mins`. Then, we will dummy code the categorical variables to put them into a numerical format that the machine learning models can understand. We will then normalize and scale all of the predictors to ensure that factors with a higher magnitude do not lead to more biased parameter estimates than ones with lower magnitude.

```
airplane_recipe <- recipe(Price ~ Airline + Source + Destination +
    Total_Stops + Journey_day + Journey_month + Duration_hours +
    Duration_mins + Departure_time + Airline_category, data = airplane_train) %>%
    # imputing Duration_hours
step_impute_linear(Duration_hours, impute_with = imp_vars(Price,
    Airline, Source, Journey_day, Journey_month, Duration_mins,
    Departure_time, Airline_category)) %>%
    # imputing Duration_mins
step_impute_linear(Duration_mins, impute_with = imp_vars(Price,
    Airline, Source, Journey_day, Journey_month, Duration_hours,
    Departure_time, Airline_category)) %>%
    # dummy coding nominal variables
step_dummy(Airline, Source, Destination, Airline_category, Departure_time) %>%
    # normalizing
step_center(all_predictors()) %>%
    step_scale(all_predictors())
prep(airplane_recipe) %>%
    bake(airplane_train)
```

```
## # A tibble: 7,475 x 26
##    Total_Stops Journey_day Journey_month Duration_hours Duration_mins Price
##          <dbl>       <dbl>         <dbl>          <dbl>         <dbl> <dbl>
##  1      -1.22        1.23         -1.46          -0.967         1.28   3897
##  2      -1.22        1.23          1.11          -0.967        -0.199  3873
##  3      -1.22        0.518        -0.600         -0.967         0.391  4174
##  4      -1.22        1.23          1.11          -0.967        -0.790  4667
##  5      -1.22       -1.49          0.256         -0.967         1.57   4823
##  6      -1.22       -0.898        -0.600         -0.967         1.28   4423
##  7       0.258       0.872         0.256         -0.616        -0.790  4649
##  8      -1.22        1.23         -1.46          -0.967         1.28   3527
##  9      -1.22        1.58          0.256         -0.967         1.28   3943
## 10      -1.22        1.23         -1.46          -0.967         1.28   4377
## # i 7,465 more rows
## # i 20 more variables: Airline_Air.India <dbl>, Airline_GoAir <dbl>,
## #   Airline_IndiGo <dbl>, Airline_Jet.Airways <dbl>,
## #   Airline_Multiple.carriers <dbl>, Airline_SpiceJet <dbl>,
## #   Airline_Vistara <dbl>, Source_Chennai <dbl>, Source_Delhi <dbl>,
## #   Source_Kolkata <dbl>, Source_Mumbai <dbl>, Destination_Cochin <dbl>,
## #   Destination_Delhi <dbl>, Destination_Hyderabad <dbl>, ...
```

K-Fold Cross-Validation

We will now be conducting 10-Fold Stratified Cross-Validation on our dataset during the training process of our models. First, the dataset is divided into k folds, where k in this case is 10. We will stratify on the response variable `Price` in order to ensure an even distribution of the response variable across the folds. In each iteration of cross-validation, each fold is held out as the validation set, or the set that the models are tested on, and the rest of the folds are used to train the models. This process is repeated until each fold is used as the validation set at least once. K-fold cross validation is incredibly useful because it helps to estimate how well the models will be able to predict new, unseen data.

```
airplane_folds <- vfold_cv(airplane_train, v = 10, strata = Price)
```

Fitting the Models

We will be fitting 4 models: ridge regression, k-neighbors, random forest model, and boosted trees. We will now create the model structures, workflows, grids and finally tune the created models using our grids.

1. Specifying What Models to Run

```
#ridge regression
ridge_model <- linear_reg(mixture = 0,
                         penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

#K nearest neighbors
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")

#Random forest
rf_model <- rand_forest(mtry = tune(),
                       trees = tune(),
                       min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

bt_model <- boost_tree(mtry = tune(),
                           trees = tune(),
                           learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")
```

2. Setting up the Workflow

```
#ridge workflow
ridge_wf <- workflow() %>%
```

```r
  add_recipe(airplane_recipe) %>%
  add_model(ridge_model)

#knn workflow
knn_wf <- workflow() %>%
  add_recipe(airplane_recipe) %>%
  add_model(knn_model)

#random forest workflow
rf_wf <- workflow() %>%
  add_recipe(airplane_recipe) %>%
  add_model(rf_model)

bt_wf <- workflow() %>%
  add_recipe(airplane_recipe) %>%
  add_model(bt_model)
```

3. Creating and Tuning the Grids

```r
# lasso & ridge regression
lr_range <- penalty(range = c(-5, 5))
lr_penalty_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 50)

# k nearest neighbors
knn_grid <- grid_regular(neighbors(range = c(1, 10)), levels = 10)

# random forest
rf_grid <- grid_regular(mtry(range = c(1, 12)), trees(range = c(200,
    800)), min_n(range = c(5, 25)), levels = 5)

# boosted trees
bt_grid <- grid_regular(mtry(range = c(1, 6)), trees(range = c(200,
    600)), learn_rate(range = c(-10, -1)), levels = 10)
```

```r
# RIDGE REGRESSION
ridge_tune <- tune_grid(
  ridge_wf,
  resamples = airplane_folds,
  grid = lr_penalty_grid,
)

# K NEAREST NEIGHBORS
knn_tune <- tune_grid(
    knn_wf,
    resamples = airplane_folds,
    grid = knn_grid
)

# RANDOM FOREST
rf_tune <- tune_grid(
  rf_wf,
  resamples = airplane_folds,
  grid = rf_grid
)

# BOOSTED TREES
bt_tune <- tune_grid(
  bt_wf,
  resamples = airplane_folds,
  grid = bt_grid
)
```

4. Saving and Loading the Models

```r
save(ridge_tune, file = "ridge_tune.rda")
save(knn_tune, file = "knn_tune.rda")
save(rf_tune, file = "rf_tune.rda")
save(bt_tune, file = "bt_tune.rda")
```

```r
load("ridge_tune.rda")
load("knn_tune.rda")
load("rf_tune.rda")
load("bt_tune.rda")
```

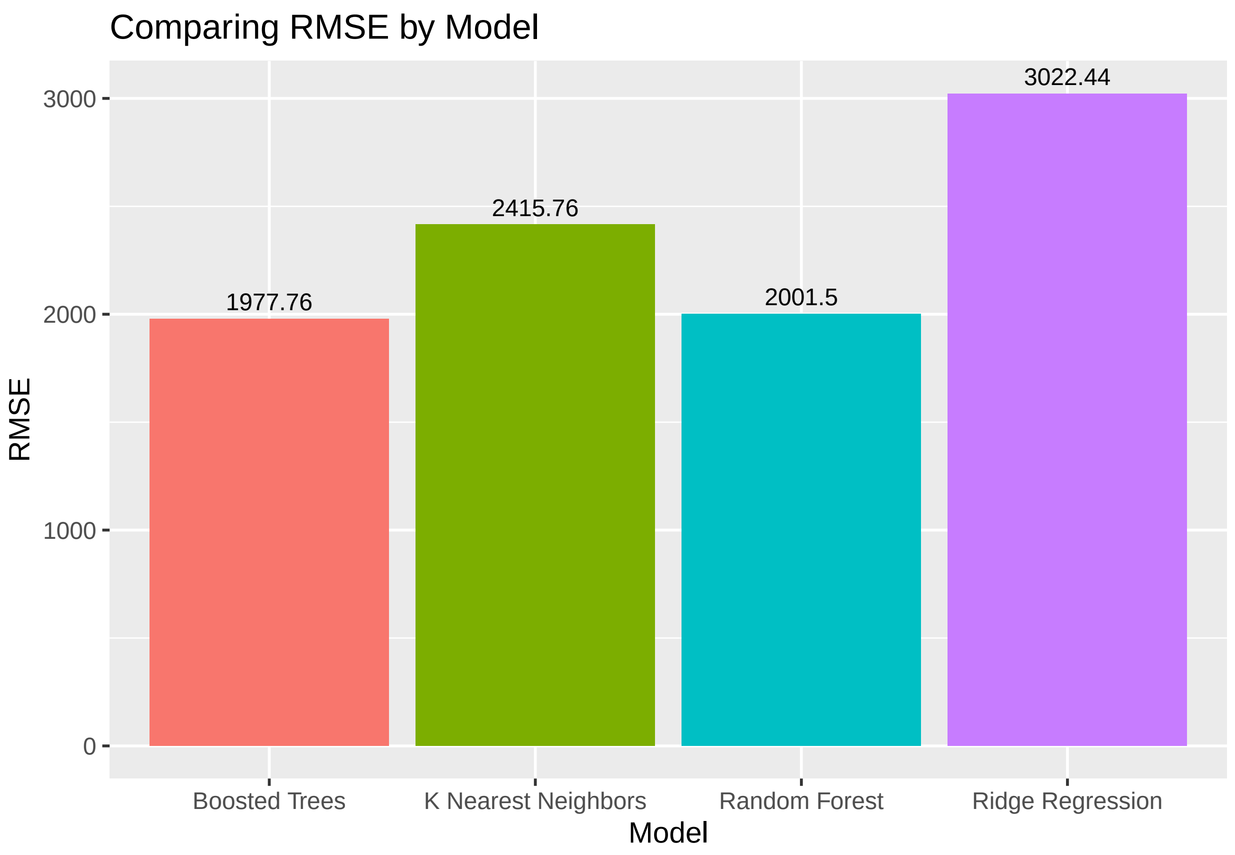
5. Collecting the Metrics of Each Model

```r
ridge_rmse <- collect_metrics(ridge_tune) %>%
  arrange(mean) %>% slice(51)

knn_rmse <- collect_metrics(knn_tune) %>%
  arrange(mean) %>% slice(11)

rf_rmse <- collect_metrics(rf_tune) %>%
  arrange(mean) %>% slice(126)

bt_rmse <- collect_metrics(bt_tune) %>%
  arrange(mean) %>% slice(601)
```

Results of Our Models

Now that we have successfully created our models, it is time to compare the results to see which ones performed the best. I have chosen to use Root Mean Square Error (RMSE) over R-Squared (Rˆ2) because it places more of an emphasis on accurately predicting individual values and minimizing prediction errors as opposed to Rˆ2, which focuses more on assessing the variance of the dependent variable that is explained by the model, giving us less information about the predictive accuracy of our models.

```r
all_models <- data.frame(Model = c("Ridge Regression", "K Nearest Neighbors",
    "Random Forest", "Boosted Trees"), RMSE = c(ridge_rmse$mean,
    knn_rmse$mean, rf_rmse$mean, bt_rmse$mean))

ggplot(all_models, aes(x = Model, y = RMSE)) + geom_bar(stat = "identity",
    aes(fill = Model)) + geom_text(aes(label = round(RMSE, 2)),
    vjust = -0.5, size = 3, color = "black") + theme(legend.position = "none") +
    labs(title = "Comparing RMSE by Model")
```



Here, we can see that the RMSE values of all of the models are fairly high. I believe this is due to the interval of the target variable `Price` ranging from 1,759 to 79,512, which is a fairly large range. Considering this, all of the RMSE values are fairly low compared to the range of the target variable. Of all of the models, the Boosted Trees model ended up performing the most successfully, with an RMSE score of 1977.76, followed closely by the Random Forest model with a RMSE score of 2001.5.

Results of the Best Model

But which parameters led to the best boosted tree model?

```
bt_rmse
```

```
## # A tibble: 1 x 9
##    mtry trees learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5   466        0.1 rmse    standard   1978.    10    51.7 Preprocessor1_M~
```

The best model is the one with 466 trees, 5 variables being randomly sampled at the creation of each tree, and a learn rate of 0.1. This model had an RMSE score of 1977.762.

Model Results on Actual Testing Data

Now that we have selected the most successful model, it is time to fit and train it one more time on the training set.

```
best_bt_train <- select_best(bt_tune, metric = 'rmse')
bt_final_wf_train <- finalize_workflow(bt_wf, best_bt_train)
bt_final_fit_train <- fit(bt_final_wf_train, data = airplane_train)
```

```r
save(bt_final_fit_train, file = "bt_final_fit_train.rda")
```

```r
load("bt_final_fit_train.rda")
airplane_tibble <- predict(bt_final_fit_train, new_data = airplane_test %>% select(-Price))
airplane_tibble <- bind_cols(airplane_tibble, airplane_test %>% select(Price))
save(airplane_tibble, file = "airplane_tibble.rda")
```

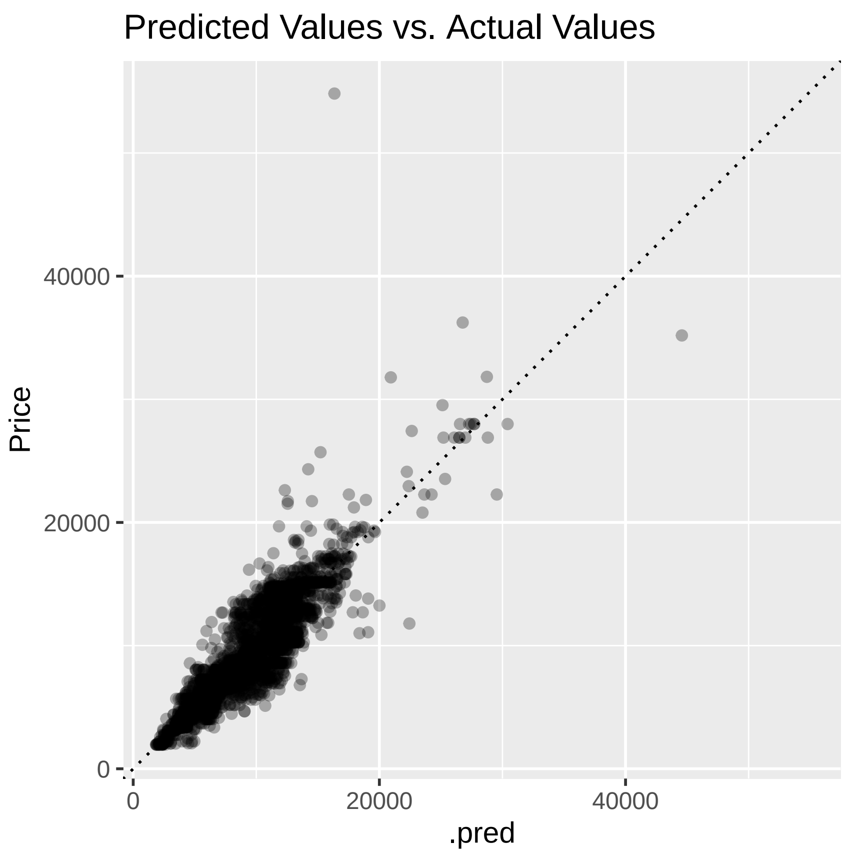
```r
load("airplane_tibble.rda")
airplane_metric <- metric_set(rmse)

# Collecting the rmse of the model on the testing data
airplane_tibble_metrics <- airplane_metric(airplane_tibble, truth = Price, estimate = .pred)
airplane_tibble_metrics
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       1875.
```

The model actually performed better on the testing set than on the training set, reducing the RMSE value by a whole 100 units! Let us compare the RMSE score to the relative range of `Price`.

As stated above, since the range of `Price` varies from 1,759 to 79,512, the RMSE value of 1874.879 is fairly low compared to this range, with the percentage of of RMSE relative to the range being equal to about 2.4%. This small percentage of RMSE relative to the range means that the prediction error is relatively small and that the model is more accurate in predicting the airfare. Let us now create a plot depicting the predicted values vs. the actual outcomes.

```r
airplane_tibble %>%
  ggplot(aes(x = .pred, y = Price)) +
  geom_point(alpha = 0.3) +
  geom_abline(lty = 3) +
  theme_grey() +
  coord_obs_pred() +
  labs(title = "Predicted Values vs. Actual Values")
```

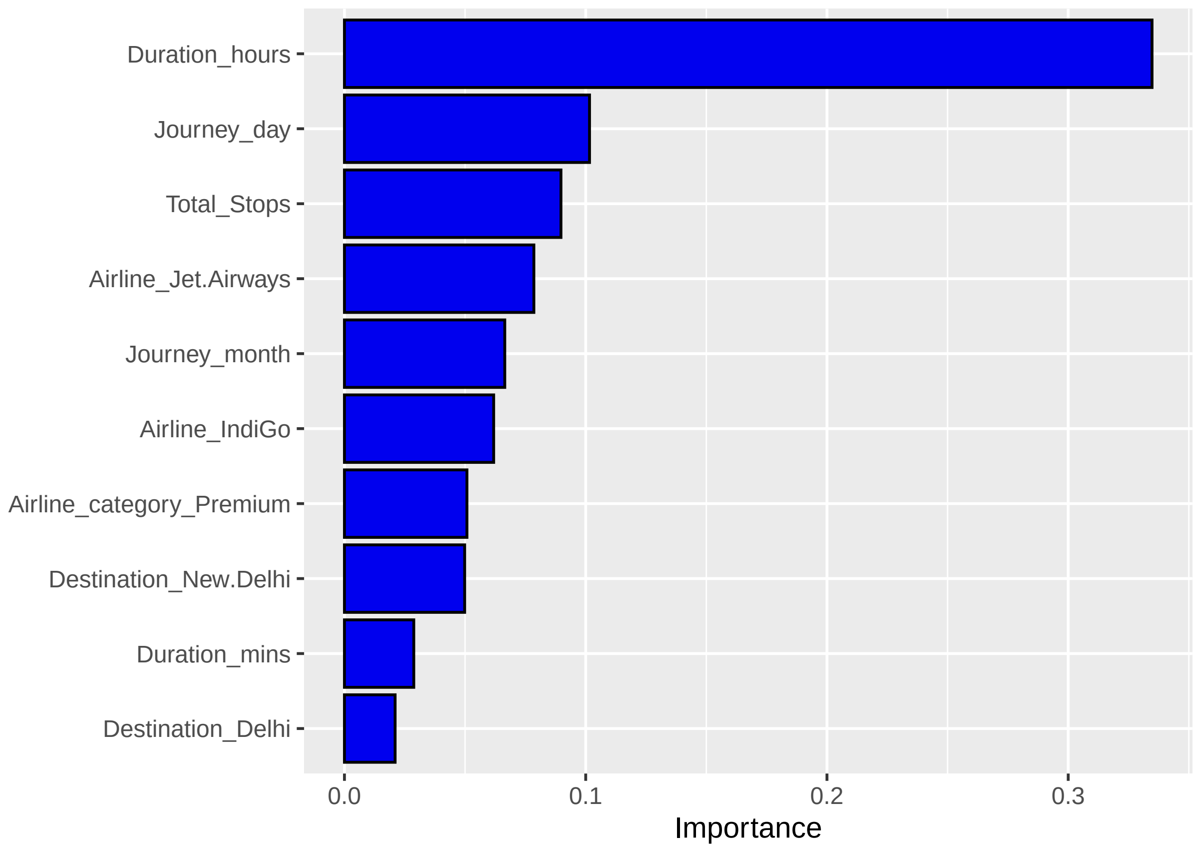


With the exception of a few outliers, most of the data lies either on or very close to the abstraction line, or the line where the prediction values equal the actual values. This means that the model was overall successful in predicting the airfare price!

Variable Importance Plot

In order to determine which predictor variables have the most impact on predicting the price of an airplane ticket, we can use a Variable Importance Plot to visualize this relationship.

```
bt_final_fit_train %>%
  extract_fit_engine() %>%
  vip(aesthetics = list(fill = "blue2", color = "black"))
```



From this graph, we can see that `duration_hours`, or the number of hours that flight is, has the most correlation with the price of the flight, which is not surprising considering the results of the correlation matrix above. Additionally, we can see from this graph that Jet Airways is the most expensive airline, followed by IndiGo. This is supported by the observation that the airline (whether or not it is Jet Airways or IndiGo) are the 5th and 7th most important variables overall.

Conclusion

The model that performed the best to predict the price of an airplane ticket was the Boosted Tree Model. This does not surprise me, as the dataset I chose is very large and complex with both numerical and categorical variables. The Boosted Tree model is the perfect fit for this type of dataset as it incorporates techniques such as tuning regularization parameters, utilizing gradient boosting and tree-based splitting, and incorporating class weights in order to improve the predictive performance of the model. This model was the best at capturing the complex relationship between the variables the best, as shown by its predictive accuracy.

The model that performed the worst was the Ridge Regression model. This outcome did not surprise me either. The results of the correlation matrix above indicated little to no linear relationship between the predictor variables and the target variable. However, it did indicate the presence of multicollinearity between some of the predictor variables. Since Ridge Regression assumes a linear relationship between the variables and does poorly when the variables are correlated with each other, this model was not the best choice and performed the worst out of all of them, as it was unable to capture the complexity of the data.

If I were to continue this project, I would like expand my dataset to include more predictor variables, such as country and year. More specifically, I would like to collect data from domestic flights in different countries and continents to determine how flight fares vary across geographical areas. Additionally, since this dataset only covers data from a singular year, 2019, it would be interesting to analyze the variation in flight fares of flights taken in different years, especially considering the impact of recent global events, such the COVID-19 pandemic and numerous geopolitical conflicts, on air travel.

Overall, creating and training models to predict the price of a domestic airplane ticket from India was my first glimpse into how to apply machine learning concepts to real world problems. I have always been really passionate about airplanes and the aviation industry in general, so having the opportunity to choose a topic to build my data analysis skills towards a topic of my choice was an incredibly rewarding experience. From learning data cleaning skills I had never utilized before, to creating and finetuning a model that fit the complexity of my data the best, this project gave me the perfect opportunity to advance my data science skills so that I can take part in more challenging projects in the future.