Regression

Linus Fackler, Justin Hardy, Fernando Colman, Isabelle Villegas

This data set contains information of over 10000 different plane rides in India in 2019, containing the price, duration, number of stops, and other information. The set can be found here: https://www.kaggle.com/datasets/ibrahimelsayed182/plane-ticket-price

Load the data

```
df <- read.csv("planetickets.csv", header = TRUE)
str(df)
```

```
## 'data.frame':    10683 obs. of  11 variables:
##  $ Airline        : chr  "IndiGo" "Air India" "Jet Airways" "IndiGo" ...
##  $ Date_of_Journey: chr  "24/03/2019" "1/05/2019" "9/06/2019" "12/05/2019" ...
##  $ Source         : chr  "Banglore" "Kolkata" "Delhi" "Kolkata" ...
##  $ Destination    : chr  "New Delhi" "Banglore" "Cochin" "Banglore" ...
##  $ Route          : chr  "BLR ? DEL" "CCU ? IXR ? BBI ? BLR" "DEL ? LKO ? BOM ?
COK" "CCU ? NAG ? BLR" ...
##  $ Dep_Time       : chr  "22:20" "05:50" "09:25" "18:05" ...
##  $ Arrival_Time   : chr  "01:10 22 Mar" "13:15" "04:25 10 Jun" "23:30" ...
##  $ Duration       : chr  "2h 50m" "7h 25m" "19h" "5h 25m" ...
##  $ Total_Stops    : chr  "non-stop" "2 stops" "2 stops" "1 stop" ...
##  $ Additional_Info: chr  "No info" "No info" "No info" "No info" ...
##  $ Price          : int  3897 7662 13882 6218 13302 3873 11087 22270 11087 8625
...
```

Data cleaning

First, we have to clean the data, since a lot of columns contain strings instead of integers or floats.

We throw out some of the columns that don't provide any useful information, such as "additional info".

```
df <- df[,c(1,3,4,6,8,9,11)]
```

We check how many NA rows there are.

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##     Airline      Source Destination    Dep_Time    Duration Total_Stops
##           0           0           0           0           0           0
##       Price
##           0
```

This data set contains no rows with missing values, so we don't have to make any changes.

Now, we check how many unique values the column Total_Stops has, before we change it from a String to an Integer.

```
length(unique(df$Total_Stops))
```

```
## [1] 6

unique(df$Total_Stops)

## [1] "non-stop" "2 stops"  "1 stop"   "3 stops"  ""         "4 stops"

sum(df$Total_Stops == "")

## [1] 1
```

We see that it has 6 unique values, one of them is just the empty string. There is only 1 row with this empty value, so we delete this row.

```
df <- df[!(df$Total_Stops == ""),]
unique(df$Total_Stops)

## [1] "non-stop" "2 stops"  "1 stop"   "3 stops"  "4 stops"
```

We change these to usable integer values and then change the type of the column to numeric.

```
df$Total_Stops[df$Total_Stops == "non-stop"] <- 0
df$Total_Stops[df$Total_Stops == "1 stop"] <- 1
df$Total_Stops[df$Total_Stops == "2 stops"] <- 2
df$Total_Stops[df$Total_Stops == "3 stops"] <- 3
df$Total_Stops[df$Total_Stops == "4 stops"] <- 4

unique(df$Total_Stops)

## [1] "0" "2" "1" "3" "4"

df <- transform(df, Total_Stops = as.integer(Total_Stops))
str(df)

## 'data.frame':    10682 obs. of  7 variables:
##  $ Airline    : chr  "IndiGo" "Air India" "Jet Airways" "IndiGo" ...
##  $ Source     : chr  "Banglore" "Kolkata" "Delhi" "Kolkata" ...
##  $ Destination: chr  "New Delhi" "Banglore" "Cochin" "Banglore" ...
##  $ Dep_Time   : chr  "22:20" "05:50" "09:25" "18:05" ...
##  $ Duration   : chr  "2h 50m" "7h 25m" "19h" "5h 25m" ...
##  $ Total_Stops: int  0 2 2 1 1 0 1 1 1 1 ...
##  $ Price      : int  3897 7662 13882 6218 13302 3873 11087 22270 11087 8625 ...
```

Now, for the column “Duration”, we will cut off everything after the ‘h’, so we only keep the hours of the flight.

```
df$Duration <- gsub("h.*","", df$Duration)
df$Duration[1:10]

##  [1] "2"  "7"  "19" "5"  "4"  "2"  "15" "21" "25" "7"
```

Then, we change the type also to numeric.

```
df <- transform(df, Duration = as.integer(Duration))

## Warning in eval(substitute(list(...)), `_data`, parent.frame()): NAs introduced
## by coercion
```

```
str(df)
```

```
## 'data.frame':    10682 obs. of  7 variables:
##  $ Airline    : chr  "IndiGo" "Air India" "Jet Airways" "IndiGo" ...
##  $ Source     : chr  "Banglore" "Kolkata" "Delhi" "Kolkata" ...
##  $ Destination: chr  "New Delhi" "Banglore" "Cochin" "Banglore" ...
##  $ Dep_Time   : chr  "22:20" "05:50" "09:25" "18:05" ...
##  $ Duration   : int  2 7 19 5 4 2 15 21 25 7 ...
##  $ Total_Stops: int  0 2 2 1 1 0 1 1 1 1 ...
##  $ Price      : int  3897 7662 13882 6218 13302 3873 11087 22270 11087 8625 ...
```

Upon checking, there is somehow an NA value in our column Duration now.

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##     Airline      Source Destination    Dep_Time    Duration Total_Stops
##           0           0           0           0           1           0
##       Price
##           0
```

We will simply delete this row, as 1 row does not matter.

```
df <- df[!(is.na(df$Duration)),]
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##     Airline      Source Destination    Dep_Time    Duration Total_Stops
##           0           0           0           0           0           0
##       Price
##           0
```

We're done with duration now.

We do the same for the departure time. We will cut off the minutes and just keep the hours.

```
df$Dep_Time <- gsub(":.*","", df$Dep_Time)
df$Dep_Time[1:10]
```

```
##  [1] "22" "05" "09" "18" "16" "09" "18" "08" "08" "11"
```

Now, we will transform it to an integer value.

```
df <- transform(df, Dep_Time = as.integer(Dep_Time))
str(df)
```

```
## 'data.frame':    10681 obs. of  7 variables:
##  $ Airline    : chr  "IndiGo" "Air India" "Jet Airways" "IndiGo" ...
##  $ Source     : chr  "Banglore" "Kolkata" "Delhi" "Kolkata" ...
##  $ Destination: chr  "New Delhi" "Banglore" "Cochin" "Banglore" ...
##  $ Dep_Time   : int  22 5 9 18 16 9 18 8 8 11 ...
##  $ Duration   : int  2 7 19 5 4 2 15 21 25 7 ...
##  $ Total_Stops: int  0 2 2 1 1 0 1 1 1 1 ...
##  $ Price      : int  3897 7662 13882 6218 13302 3873 11087 22270 11087 8625 ...
```

Train and Test sets

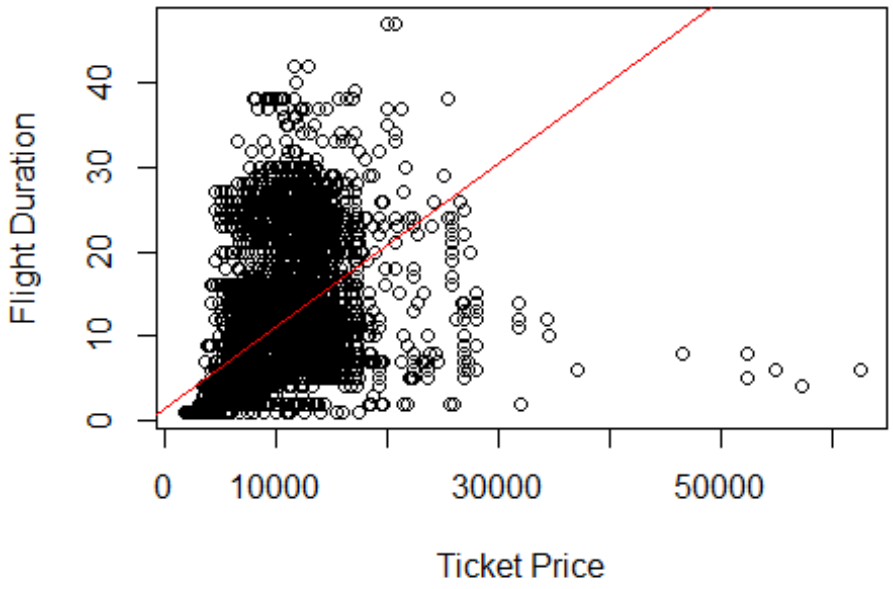
Divide into train and test sets

```
set.seed(1234)
i <- sample(1:nrow(df), round(nrow(df)*0.8), replace=FALSE)
train <- df[i, -9]
test <- df[-i, -9]
```

Data Exploration of Training Data

First, we look at how flight durations and ticket prices look together in a plot.

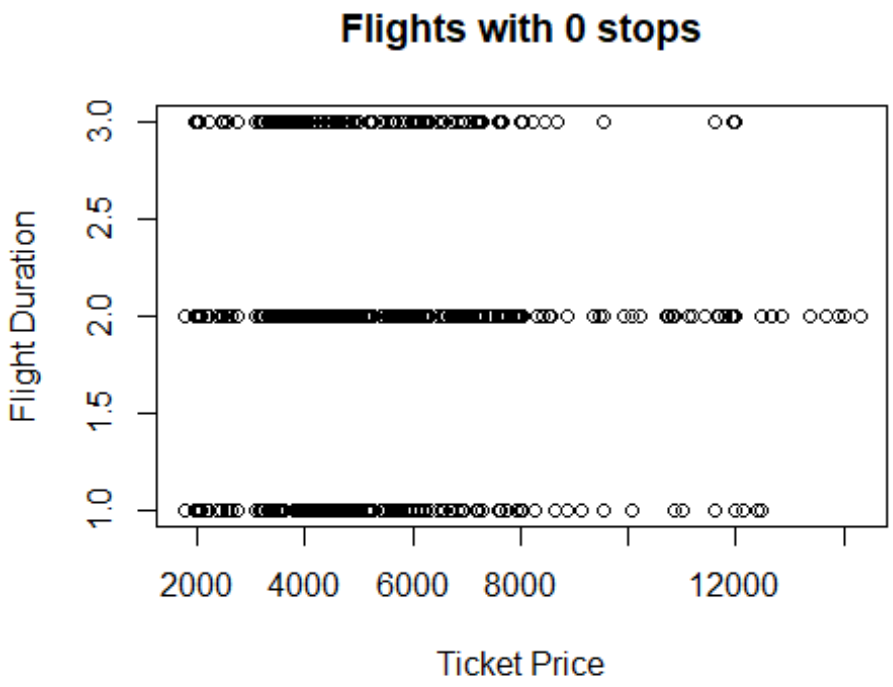
```
plot(train$Duration~train$Price, xlab="Ticket Price", ylab="Flight Duration")
abline(lm(train$Duration~train$Price), col="red")
```



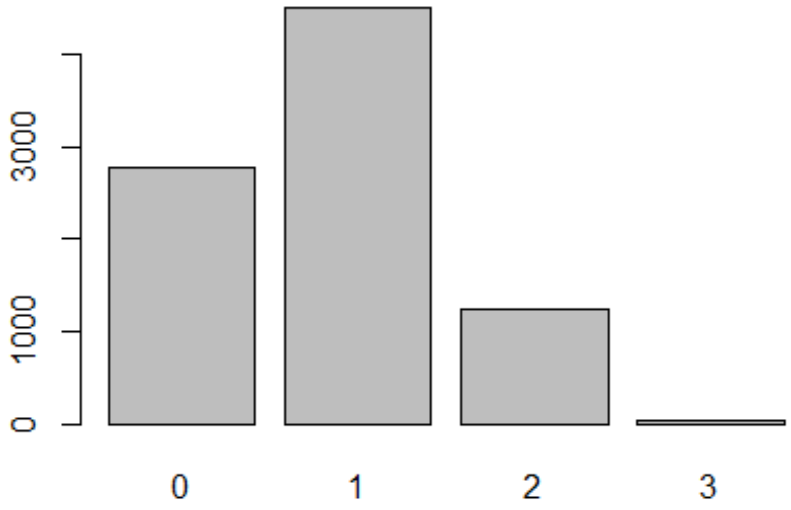
This shows us the general trend of longer flights resulting in higher ticket prices.

Now, we will see how

```
x <- train$Price[(train$Price < 15000) & (train$Total_Stops == 0)]
y <- train$Duration[(train$Duration < 20) & (train$Total_Stops == 0)]
plot(y[1:min(length(x),length(y))]~x[1:min(length(x),length(y))], xlab="Ticket
Price", ylab="Flight Duration", main="Flights with 0 stops")
```

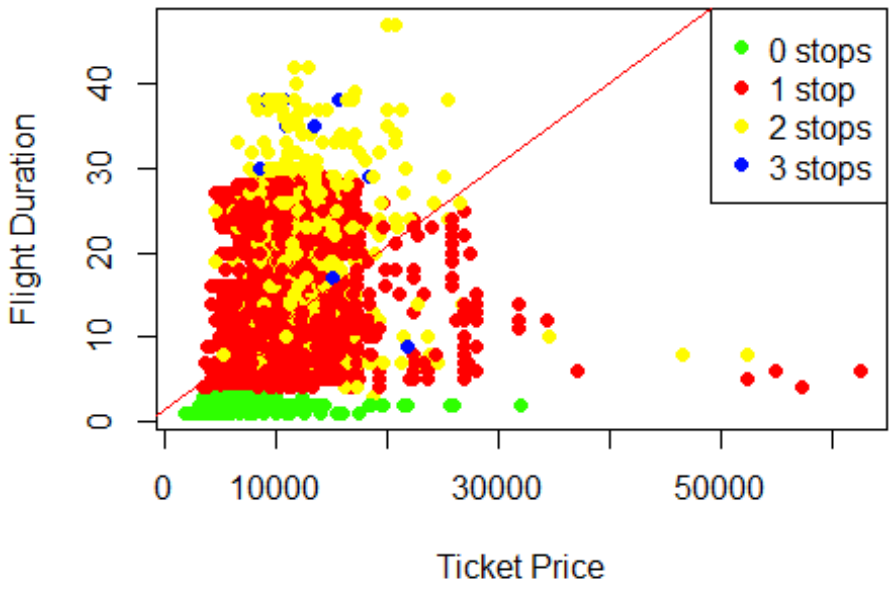



```
counts <- table(train$Total_Stops)
barplot(counts)
```



We see that there is no data where the number of stops is 4 in the training portion of the data, which is why we will leave it out in the next plot.

```
colors <- c("#2EFF00", #green for 0 stops
            "#FF0000", #red for 1 stop
            "#FFFB00", #yellow for 2 stops
            "#000CFF" #blue for 3 stops
            )

groups <- factor(train$Total_Stops)
plot(train$Duration~train$Price, xlab="Ticket Price", ylab="Flight Duration", pch
= 19, col=colors[groups])
legend("topright", legend=c("0 stops", "1 stop", "2 stops", "3 stops"), pch = 19,
col = colors[factor(levels(groups))])
abline(lm(train$Duration~train$Price), col="red")
```



This tells us that most short flights have 0 stops and are less than $10000. It also tells us that the number of stops doesn't necessarily affect the price, rather just the duration.

Linear Regression

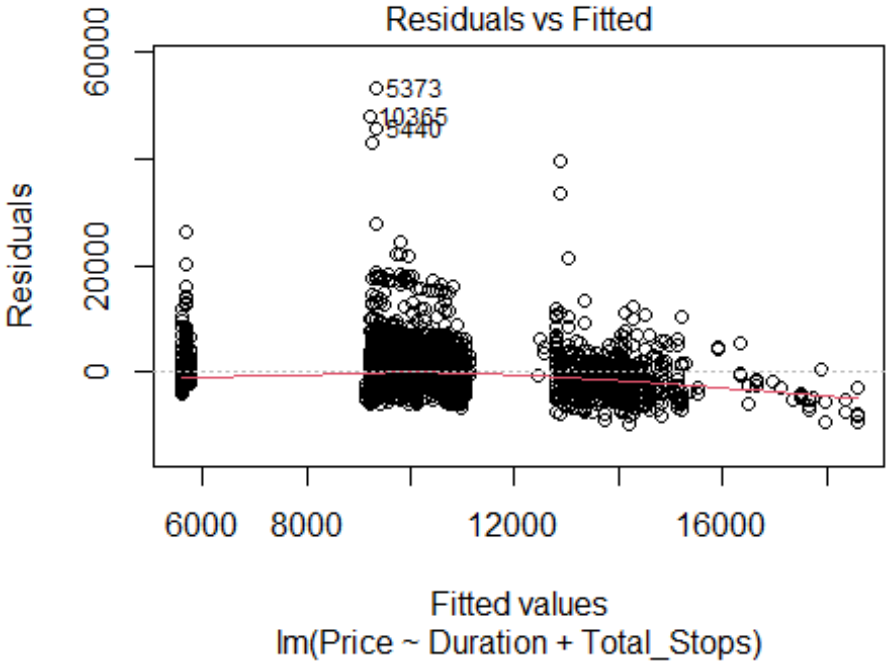
Model for predictors Duration + Total_Stops

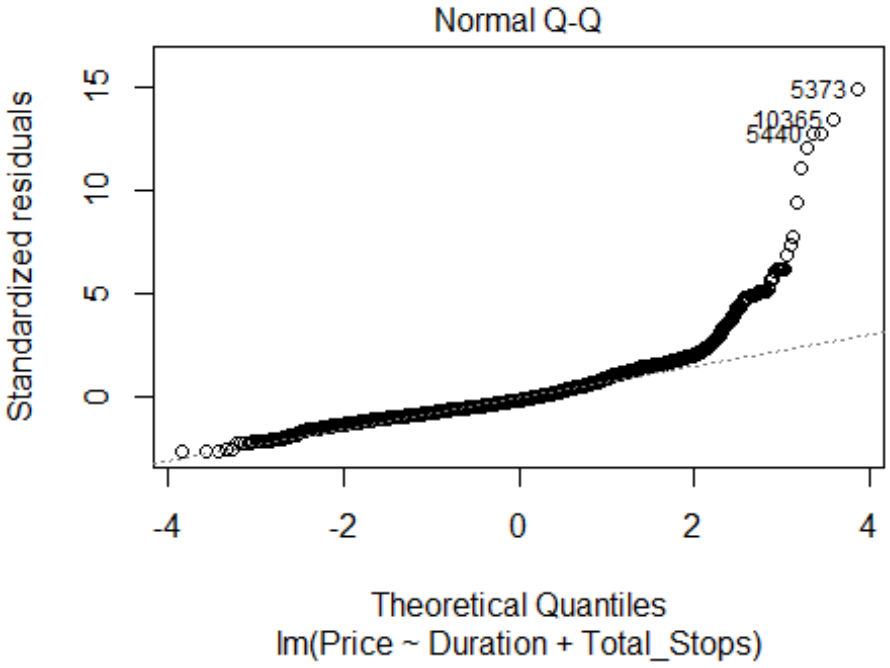
```
lm1 <- lm(Price ~ Duration + Total_Stops, data=train)
summary(lm1)

##
## Call:
## lm(formula = Price ~ Duration + Total_Stops, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

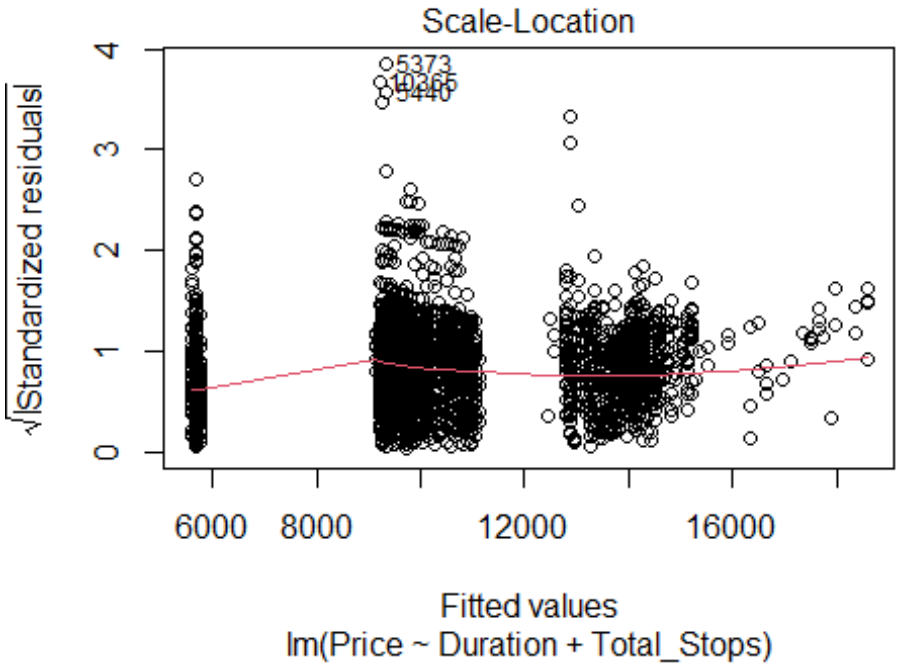
```
##  -9556  -2133   -784   1554  53073
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 5507.860     63.563   86.65   <2e-16 ***
## Duration      77.081      6.771   11.38   <2e-16 ***
## Total_Stops 3383.876     85.180   39.73   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3574 on 8542 degrees of freedom
## Multiple R-squared:  0.3822, Adjusted R-squared:  0.382
## F-statistic:  2642 on 2 and 8542 DF,  p-value: < 2.2e-16
```

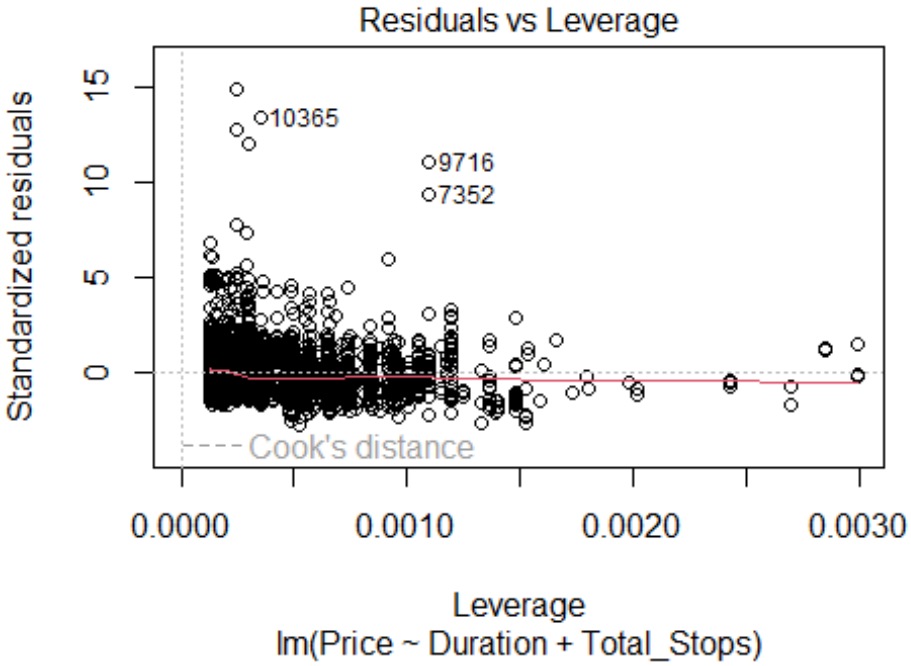
In this Linear Regression model we see the effect that the duration of the flight and number of stops have on the ticket price. Our R-squared value is around 0.38, which indicates that the duration and number of stops are not the best predictors for this model. They don't have as big of an effect on the ticket price as I thought. This can be because there are other factors that affect the flight, like the airline, since some are more luxurious and therefore more expensive, or the amount of days the ticket was purchased prior to the flight. If tickets are bought a couple of days before the flight, they are most likely more expensive than tickets that were purchased ahead in time. The fact that the p-value is less than 0.5 shows that this model is statistically significant.

```
plot(lm1)
```


Residuals vs Fitted
Residuals
60000
20000
0
5373
10365
5440
6000
8000
12000
16000
Fitted values
lm(Price ~ Duration + Total_Stops)


Normal Q-Q
Standardized residuals
15
10
5
0
5373
10365
5440
-4
-2
0
2
4
Theoretical Quantiles
lm(Price ~ Duration + Total_Stops)





Residuals vs Fitted: This represents the difference between the actual price of the plane ticket and our models prediction. Our model suggests some form of heteroscedasticity, meaning the variability of our predictions are not equally variable throughout. Some values are closer to the 0 line than others. Most values are following a linear relationship though. So there is a clear relationship between the predictors and the predicted value.

Normal Q-Q: Most data is in between -2 and 2 standard deviations, just like in a normal distributions. There is a bunch of extremes on the right side, which means it is very hard to predict what the price is going to be based off this model.

Scale-Location: This plot shows if the residuals are spread equally among our predictions in order to check homoscedasticity. Since there are clear trends in the plot, there is no equal variance in our residuals.

Residuals vs Leverage: This plot helps us find influential data points. We don't have any data points that have large leverage and also high residuals, meaning, there are few data points that have a big impact on the coefficients and the intercept of the model. So, we don't have any data points that we should remove necessarily.

Evaluate on the test set for predictors Duration and Total_Stops

```
pred1 <- predict(lm1, newdata = test)
cor1 <- cor(pred1, test$Price)
mse1 <- mean((pred1 - test$Price) ^2)
rmse1 <- sqrt(mse1)

print(paste("correlation:", cor1))

## [1] "correlation: 0.584158942107584"

print(paste("mse:", mse1))

## [1] "mse: 15580681.0188734"

print(paste("rmse:", rmse1))

## [1] "rmse: 3947.23713740046"
```

Model with all predictors

```
lma <- lm(Price ~., data = train)
summary(lma)

##
## Call:
## lm(formula = Price ~ ., data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -7932  -1407   -366   1157  40849
##
## Coefficients: (4 not defined because of singularities)
##                                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)                              6671.299    225.572  29.575  < 2e-16
## AirlineAir India                         1728.730    206.412   8.375  < 2e-16
## AirlineGoAir                               31.473    292.967   0.107  0.91445
## AirlineIndiGo                             327.285    197.325   1.659  0.09723
## AirlineJet Airways                       4316.456    194.800  22.158  < 2e-16
## AirlineJet Airways Business             44491.881   1303.749  34.126  < 2e-16
## AirlineMultiple carriers                 3727.769    212.660  17.529  < 2e-16
## AirlineMultiple carriers Premium economy  4405.548    853.178   5.164 2.48e-07
```

```
## AirlineSpiceJet                               -305.207    217.159  -1.405  0.15992
## AirlineTrujet                                -1400.622   2888.854  -0.485  0.62780
## AirlineVistara                                2245.938    237.025   9.476  < 2e-16
## AirlineVistara Premium economy                4067.229   1673.804   2.430  0.01512
## SourceChennai                                -2508.810    205.218 -12.225  < 2e-16
## SourceDelhi                                  -2532.786    124.337 -20.370  < 2e-16
## SourceKolkata                                -2483.059    122.905 -20.203  < 2e-16
## SourceMumbai                                 -4037.828    167.056 -24.171  < 2e-16
## DestinationCochin                                   NA         NA      NA       NA
## DestinationDelhi                             -3475.103    147.747 -23.521  < 2e-16
## DestinationHyderabad                                NA         NA      NA       NA
## DestinationKolkata                                  NA         NA      NA       NA
## DestinationNew Delhi                                NA         NA      NA       NA
## Dep_Time                                       17.510      5.555   3.152  0.00163
## Duration                                        9.995      5.908   1.692  0.09070
## Total_Stops                                  2649.529     82.742  32.022  < 2e-16
##
## (Intercept)                                  ***
## AirlineAir India                             ***
## AirlineGoAir
## AirlineIndiGo                                .
## AirlineJet Airways                           ***
## AirlineJet Airways Business                  ***
## AirlineMultiple carriers                     ***
## AirlineMultiple carriers Premium economy     ***
## AirlineSpiceJet
## AirlineTrujet
## AirlineVistara                               ***
## AirlineVistara Premium economy               *
## SourceChennai                                ***
## SourceDelhi                                  ***
## SourceKolkata                                ***
## SourceMumbai                                 ***
## DestinationCochin
## DestinationDelhi                             ***
## DestinationHyderabad
## DestinationKolkata
## DestinationNew Delhi
## Dep_Time                                     **
## Duration                                     .
## Total_Stops                                  ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2879 on 8525 degrees of freedom
## Multiple R-squared:  0.5998, Adjusted R-squared:  0.5989
## F-statistic: 672.4 on 19 and 8525 DF,  p-value: < 2.2e-16
```

Compared to our model with just Duration and Total_Stops as predictors, this model shows a higher R-squared value, indicating that there are more factors that influence the price of the plane ticket. As predicted above, the airline carrier makes a difference, since some tend to me more expensive.

Evaluate model with all predictors

```
preda <- predict(lma, newdata = test)
cora <- cor(preda, test$Price)
msea <- mean((preda - test$Price) ^2)
print(paste("cor=", cora))
```

```
## [1] "cor= 0.76968523775651"
```

```
print(paste("mse=", msea))
```

```
## [1] "mse= 9698939.7919547"
```

We see pretty decent results. Let's see how they compare to kNN.

kNN for regression

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
# fit the model
fit <- knnreg(train[,4:6],train[,7],k=3)

# evaluate
pred2 <- predict(fit, test[,4:6])
cor_knn1 <- cor(pred2, test$Price)
mse_knn1 <- mean((pred2 - test$Price) ^2)
print(paste("cor=", cor_knn1))
```

```
## [1] "cor= 0.679339454132071"
```

```
print(paste("mse=", mse_knn1))
```

```
## [1] "mse= 12759958.583699"
```

As we can see, the results for kNN weren't quite as good as the results for the Linear Regression model. (Cor for LinReg: ~0.77, kNN: ~0.68) A reason for this difference might be that we didn't scale the data for kNN, which works better on scaled data.

Scale the data for kNN

We are scaling both train and test data on the means and standard deviations of the training set. This is so that information about the test data does not leak into the scaling.

```
train_scaled <- train[4:6]
means <- sapply(train_scaled, mean)
stdvs <- sapply(train_scaled, sd)
train_scaled <- scale(train_scaled, center=means, scale=stdvs)
test_scaled <- scale(test[, 4:6], center=means, scale=stdvs)
```

kNN on scaled data

```
fit <- knnreg(train_scaled, train$Price, k=3)
pred3 <- predict(fit, test_scaled)
cor_knn2 <- cor(pred3, test$Price)
mse_knn2 <- mean((pred3 - test$Price) ^2)
print(paste("cor=", cor_knn2))
```

```
## [1] "cor= 0.680064162657665"
```

```
print(paste("mse=", mse_knn2))
```

```
## [1] "mse= 12739665.9924593"
```

The kNN now has a *slightly* higher cor and lower mse than before scaling, but still not higher than the Linear Regression. This might just be because we can only use 3 predictor values, because the other columns contain characters and not numeric values. Compared to the first Linear Regression model we made, though, using only Duration and Total_Stops as predictors, we have a high increase in Correlation and decrease in mse. (Cor for first lm model: 0.58, for kNN: 0.68)

Find the best k

We will try various values of k and plot the results.

```
cor_k <- rep(0, 20)
mse_k <- rep(0, 20)
i <- 1
for (k in seq(1, 39, 2)) {
  fit_k <- knnreg(train_scaled, train$Price, k=k)
  pred_k <- predict(fit_k, test_scaled)
  cor_k[i] <- cor(pred_k, test$Price)
  mse_k[i] <- mean((pred_k - test$Price)^2)
  print(paste("k=", k, cor_k[i], mse_k[i]))
  i <- i + 1
}
```

```
## [1] "k= 1 0.679532385149251 12775379.3365894"
## [1] "k= 3 0.680064162657665 12739665.9924593"
## [1] "k= 5 0.684164787758733 12582162.1263527"
## [1] "k= 7 0.683470613982923 12602502.1739109"
## [1] "k= 9 0.680956410633889 12683507.5976628"
## [1] "k= 11 0.671786051022922 12976684.7561946"
## [1] "k= 13 0.667703952965126 13105256.5607836"
## [1] "k= 15 0.670920631314766 13003740.4050598"
## [1] "k= 17 0.67083363096341 13008086.8167307"
## [1] "k= 19 0.668188392789619 13093213.7791912"
## [1] "k= 21 0.66732212881467 13120877.4467431"
## [1] "k= 23 0.666704476200499 13140518.5321617"
## [1] "k= 25 0.664163641127559 13219073.0067556"
## [1] "k= 27 0.662103413313666 13282726.2014171"
## [1] "k= 29 0.661556475971501 13301342.8765166"
## [1] "k= 31 0.662513906760468 13273991.243621"
## [1] "k= 33 0.661163394757784 13318670.9156331"
## [1] "k= 35 0.661938828399245 13296059.8344066"
```

```
## [1] "k= 37 0.661897464064353 13299176.0080797"
## [1] "k= 39 0.654935199111982 13508553.9760139"
```

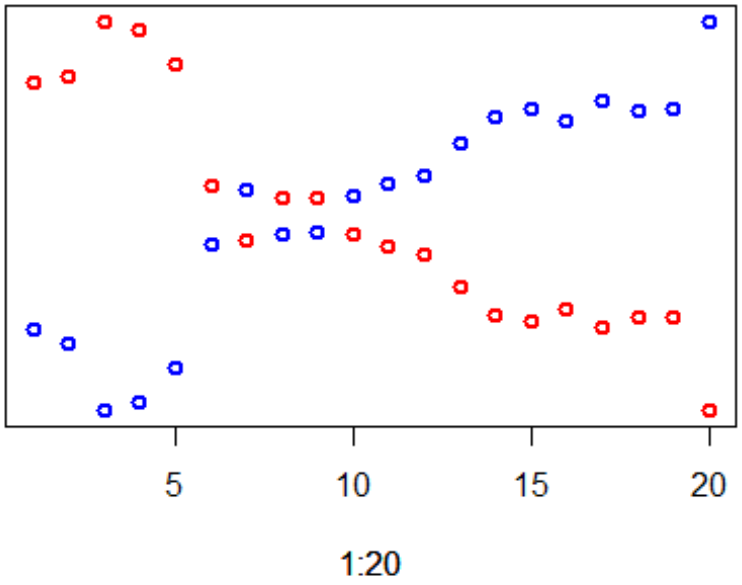
```
plot(1:20, cor_k, lwd=2, col='red', ylab="", yaxt='n')
par(new=TRUE)
plot(1:20, mse_k, lwd=2, col='blue', labels=FALSE, ylab="", yaxt='n')
```

```
## Warning in plot.window(...): "labels" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "labels" is not a graphical parameter
```

```
## Warning in box(...): "labels" is not a graphical parameter
```

```
## Warning in title(...): "labels" is not a graphical parameter
```



As we can see, the best value for k is at k = 5 (in this plot it is at 3, but that is because there is k=5 is at index 3 in lists cor_k and mse_k)

We can also check with min and max:

```
which.min(mse_k)
```

```
## [1] 3
```

```
which.max(cor_k)
```

```
## [1] 3
```

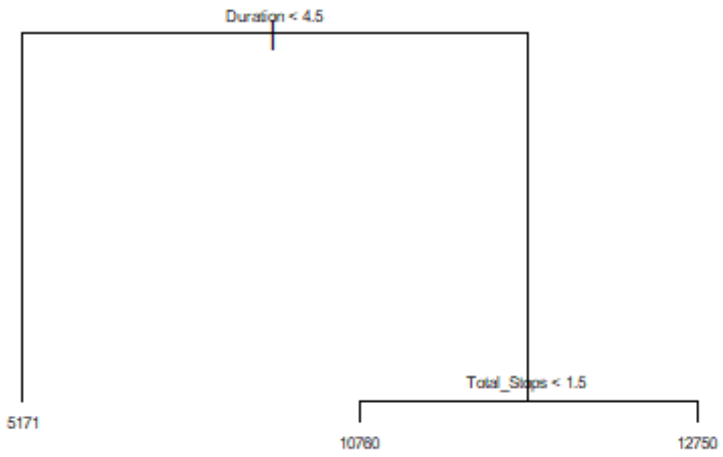
Since we have used k=3 (which was just a coincidence) in the above kNN regression already, we have our best data for this regression.

Decision Tree for regression

```
library(tree)
library(MASS)
tree1 <- tree(Price ~., data=train)
summary(tree1)

##
## Regression tree:
## tree(formula = Price ~ ., data = train)
## Variables actually used in tree construction:
## [1] "Duration"    "Total_Stops"
## Number of terminal nodes:  3
## Residual mean deviance:  11910000 = 1.017e+11 / 8542
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -8104.0 -1996.0  -393.2     0.0  1783.0 52040.0

pred4 <- predict(tree1, newdata=test)
cor_tree <- cor(pred4, test$Price)
print(paste("cor:", cor_tree))

## [1] "cor: 0.63007524717813"

mse_tree <- mean((pred4 - test$Price) ^2)
print(paste("mse:", mse_tree))

## [1] "mse: 14271349.8899186"
```

So far, the correlation and mse are worse than for either kNN or Linear Regression.

```
plot(tree1)
text(tree1, cex = 0.5, pretty = 0)
```



Concusion

Comparing all the results, kNN and Linear Regression performed the best. In the dataset there are only 3 predictors that are numeric values, which made it hard to fairly compare all 3 models. The Linear Regression model got the best results by using all predictors. The kNN got the best results considering it could only use 3 predictors. And even then, it was only slightly worse than the Linear Regression model using all predictors. The Decision tree performed slightly worse than the kNN, as it was also only using 2 predictors. For this dataset specifically, the Linear Regression was the most powerful model. I think these results strongly differ with different types of data sets.