Test PDF

Nathan Siefken

2/3/2019

Background and Motivation *

Data Set*

Goal *

The goal of my project is do to a deep analysis of the data and determine patterns as a human then independently create different machines that will learn from the data these patterns. To create the machines I will train each machine learning algorithm from the training subset to predict the amount of parking citation depending on the time of day and the day of the week in Los Angeles.

To do this, I will first wrangle the data and then I will do some exploratory data analysis. Once can see patterns as a human, I will create a baseline model which I will compare my two methods of machine learning, Linear Regression and Regression Tree and ultimately determine if they would see the same patters I did in my analysis which will prove what machine learning is.

During analysis we will review the Root Mean Squared Error or RMSE as a guide, to determine the best method. RMSE allows us the flexibility to compare the results from different types of models, Linear Regression and Regression Tree. Kaggle competitions also use RMSE or MAE as the metric for judging competitions. I will keep track of the RMSE in a matrix for ease of comparison in the results section.

Data Loading and Setup

We will utilize and load several packages from CRAN to assist with our analysis. These will be automatically downloaded and installed during code execution.

I will do some data analysis exploration and then before get into the machine learning section, I will split the data into a training and validation set (20%).

```r
# This code chunk simply makes sure that all the libraries used here are installed.
packages <- c("knitr","dplyr",  "tidyr", "caret", "ggplot2", "caret",
              "plotly","lubridate","leaflet", "stringr","rpart.plot", "rpart")
if ( length(missing_pkgs <- setdiff(packages, rownames(installed.packages()))) > 0) {
message("Installing missing package(s): ", paste(missing_pkgs, collapse = ", "))
install.packages(missing_pkgs)
}
```

Intro:

Cleaning our Data

We will combine the Issue.Date and the Issue.Time into its own column. To do this we will need to restructure the data into a time series friendly format.

Currently there is an incorrect time stamp of "T00:00:00" Issue.date variables that needs to be removed.

```r
#Removing excess information
FTP$Issue.Date <- sub("T.*", "", FTP$Issue.Date)
```

We will use some string processing techniques to clean up our Issue.Time column.

In addition, there are Longitude and Latitude variables that have a default value of 99999 that will need to be removed, so that we can plot the coordinates of the tickets.

```
#We can notice that there is a default of 99999 when a coordinate isn't entered.
#We will remove these
FTP<- FTP %>%
  filter(Latitude != 99999)
```

The following code converts the coordinates from US feet to Longitude and Latitude coordinates. Then we remove the coordinate columns that are in US feet.

```
summary(FTP[9:10])# Our Before
```

```
##     Latitude          Longitude
##  Min.   :6363049   Min.   :1715875
##  1st Qu.:6431997   1st Qu.:1837244
##  Median :6459106   Median :1846081
##  Mean   :6455205   Mean   :1850100
##  3rd Qu.:6476722   3rd Qu.:1860871
##  Max.   :6513183   Max.   :1941801
```

```
#Create projection element to convert from US Feet coordinates to normal lat lon
pj <- "+proj=lcc +lat_1=34.03333333333333 +lat_2=35.46666666666667 +lat_0=33.5 +
lon_0=-118 +x_0=2000000 +y_0=500000.0000000002 +ellps=GRS80 +datum=NAD83 +
to_meter=0.3048006096012192 no_defs"

#Add converted latitude longitude to FTP dataframe
FTP<- cbind(FTP, data.frame(project(data.frame(FTP$Latitude, FTP$Longitude),
                                    proj = pj, inverse = TRUE)))
FTP <- FTP[-9:-10] #This removes the Latitude and Logitude in Feet from our table
names(FTP)[c(9, 10)] <- c('Longitude', 'Latitude') #Rename column names of converted
#longitude latitude. Now our data is looking clean and usable
summary(FTP[9:10]) # Our After
```

```
##    Longitude        Latitude
##  Min.   :50.46   Min.   :34.09
##  1st Qu.:50.96   1st Qu.:35.05
##  Median :51.18   Median :35.19
##  Mean   :51.12   Mean   :35.23
##  3rd Qu.:51.31   3rd Qu.:35.30
##  Max.   :51.74   Max.   :36.13
```

After all the cleaning of our data, now we can combine Issue.Date and Issue.Time, so we can easily seperate out the Days of the week and Hours of the day and place them into their own column. It will become clear why I would do this further in the report.

```
#combined the date and time
FTP$Date <- as.POSIXlt(paste(FTP$Issue.Date, FTP$Issue.time), format="%Y-%m-%d %H:%M")
#Seperate the days of the week tickets where given and place it into a column
FTP$Weekdays <- weekdays(FTP$Date)
#Seperate the Hours of the day tickets where given and place it into a column
FTP$Hour <- FTP$Date$hour
```

Finally, there are some NAs that we need to eliminate. Since there are so few, simply removing them will not affect our results.

```
FTP <- na.omit(FTP)# Very few for this many observation (less than 1%)
FTP <- FTP[-11] #We will remove our date column because we are
#not able to preform some calculations when a column is in POSIXlt format.
#Since we have no further use for it, we shall remove it.
summary(FTP)  # This is our sqeaky clean table
```

```
##  Ticket.number        Issue.Date         Issue.time
##  Min.   :1.068e+09   Length:130298      Length:130298
##  1st Qu.:4.346e+09   Class :character   Class :character
##  Median :4.346e+09   Mode  :character   Mode  :character
##  Mean   :4.293e+09
##  3rd Qu.:4.347e+09
##  Max.   :4.348e+09
##     Route                Agency      Violation.code
##  Length:130298      Min.   : 1.00   Length:130298
##  Class :character   1st Qu.:53.00   Class :character
##  Mode  :character   Median :54.00   Mode  :character
##                     Mean   :53.17
##                     3rd Qu.:55.00
##                     Max.   :58.00
##  Violation.Description  Fine.amount       Longitude        Latitude
##  Length:130298         Min.   : 25.00   Min.   :50.46   Min.   :34.09
##  Class :character      1st Qu.: 63.00   1st Qu.:50.96   1st Qu.:35.05
##  Mode  :character      Median : 73.00   Median :51.18   Median :35.19
##                        Mean   : 70.77   Mean   :51.12   Mean   :35.23
##                        3rd Qu.: 73.00   3rd Qu.:51.31   3rd Qu.:35.30
##                        Max.   :363.00   Max.   :51.74   Max.   :36.13
##    Weekdays              Hour
##  Length:130298      Min.   : 0.00
##  Class :character   1st Qu.: 8.00
##  Mode  :character   Median :11.00
##                     Mean   :11.29
##                     3rd Qu.:14.00
##                     Max.   :23.00
```

Analysing the data

First, lets see how much revenue was generated last month just from ticketing.

```
revenue <- FTP %>% summarize(Revenue = sum(Fine.amount))
revenue %>% knitr::kable()
```

Revenue
9221737

Filter top 10 Violations

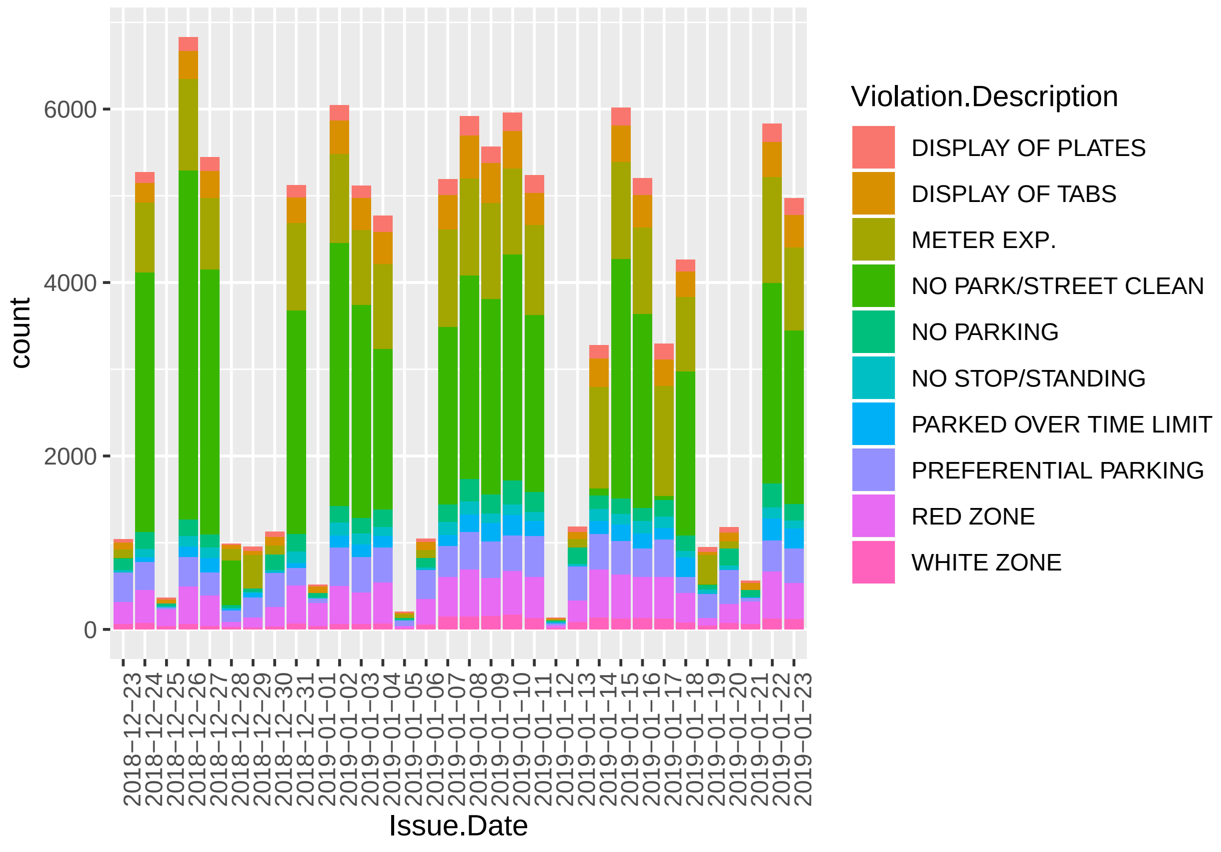
```
TopViolations <- FTP %>%
  group_by(Violation.Description) %>%
  tally() %>%
  arrange(-n) %>%
  head(10)
```

```
TopViolations %>% knitr::kable()
```

Violation.Description	n
NO PARK/STREET CLEAN	43155
METER EXP.	20913
RED ZONE	10894
PREFERENTIAL PARKING	9637
DISPLAY OF TABS	7757
NO PARKING	4941
DISPLAY OF PLATES	3825
PARKED OVER TIME LIMIT	3261
WHITE ZONE	2652
NO STOP/STANDING	2578

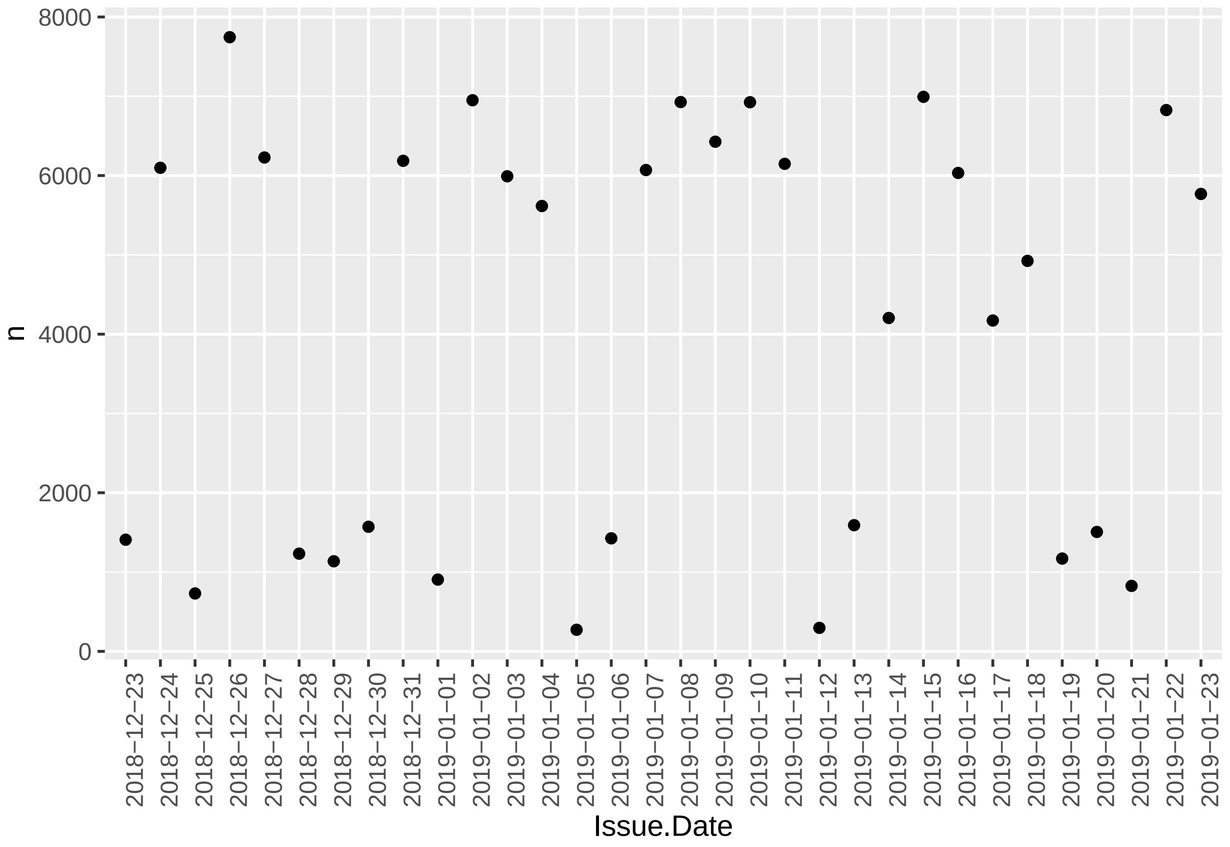
Now, lets graph top 10 Violations for the past month.

```
TopViolationsLastYears <- FTP %>%
  filter(Violation.Description %in%
           TopViolations$Violation.Description)

p <- ggplot(TopViolationsLastYears, aes(Issue.Date)) +
  geom_bar(aes(fill=Violation.Description), stat='count')+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
#Plot the data), stat='count')
p
```



The lions share of the tickets are “NoPark/Street Clean”, also, there apears to be some peaks and valleys. We will investigate further to understand this pattern a little better.

```r
#This one would be better for a month
DailyParkingViolation <- FTP %>%
  group_by(Issue.Date) %>%
  tally() %>%
  ggplot(aes(x=Issue.Date, y=n)) +
  geom_point()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

DailyParkingViolation
```



It appears there is a cloud of data points towards the top and the bottom of the graph. That is interesting, it appears that about for every five that are in the upper cloud, two points follow in the lower part of the graph. This would suggest to me that the points on the upper part of the graph are weekdays and the lower part are weekends.

The following code will show a table of frequency of tickets per day of the week and in fact this confirms our theory.

```r
table(FTP$Weekday) %>% knitr::kable()
```

Var1	Freq
Friday	17920
Monday	23383
Saturday	2874
Sunday	7501
Thursday	23316

Var1	Freq
Tuesday	22381
Wednesday	32923

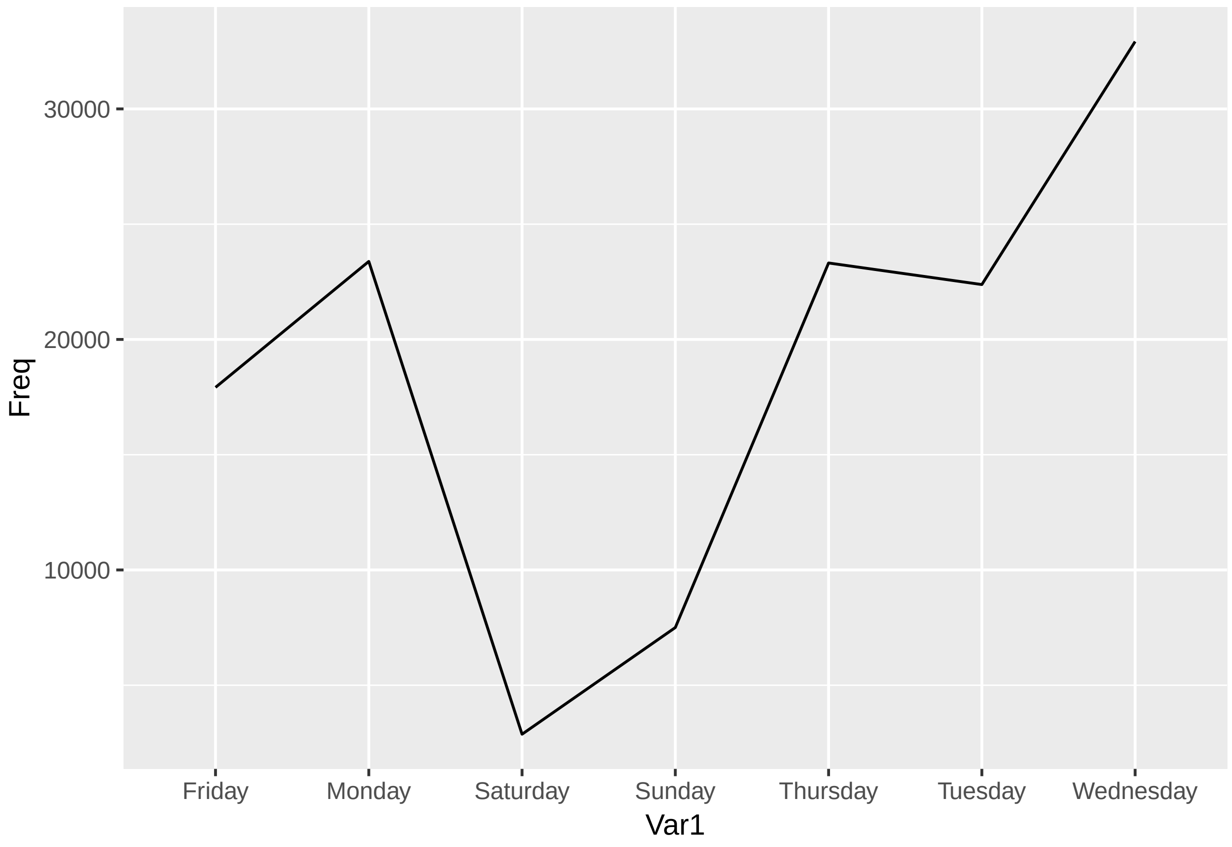
We aren't going to stop at the day of the week, we should investigate the time of day. In case you are not familiar with the 24-hour clock, 0 is midnight that runs to 23:59 (11:59 PM).

```
table(FTP$Hour)
```

```
##
##     0     1     2     3     4     5     6     7     8     9    10    11
##  1438  3256  3115  1990  1579  1295  1729  2490 17109  8968 16545 11670
##    12    13    14    15    16    17    18    19    20    21    22    23
## 16307  7935  5602  3833  6012  4222  4249  3642  2330  1785  2029  1168
```

Tables are interesting to look at but a visualization will make it much easier to interpret.

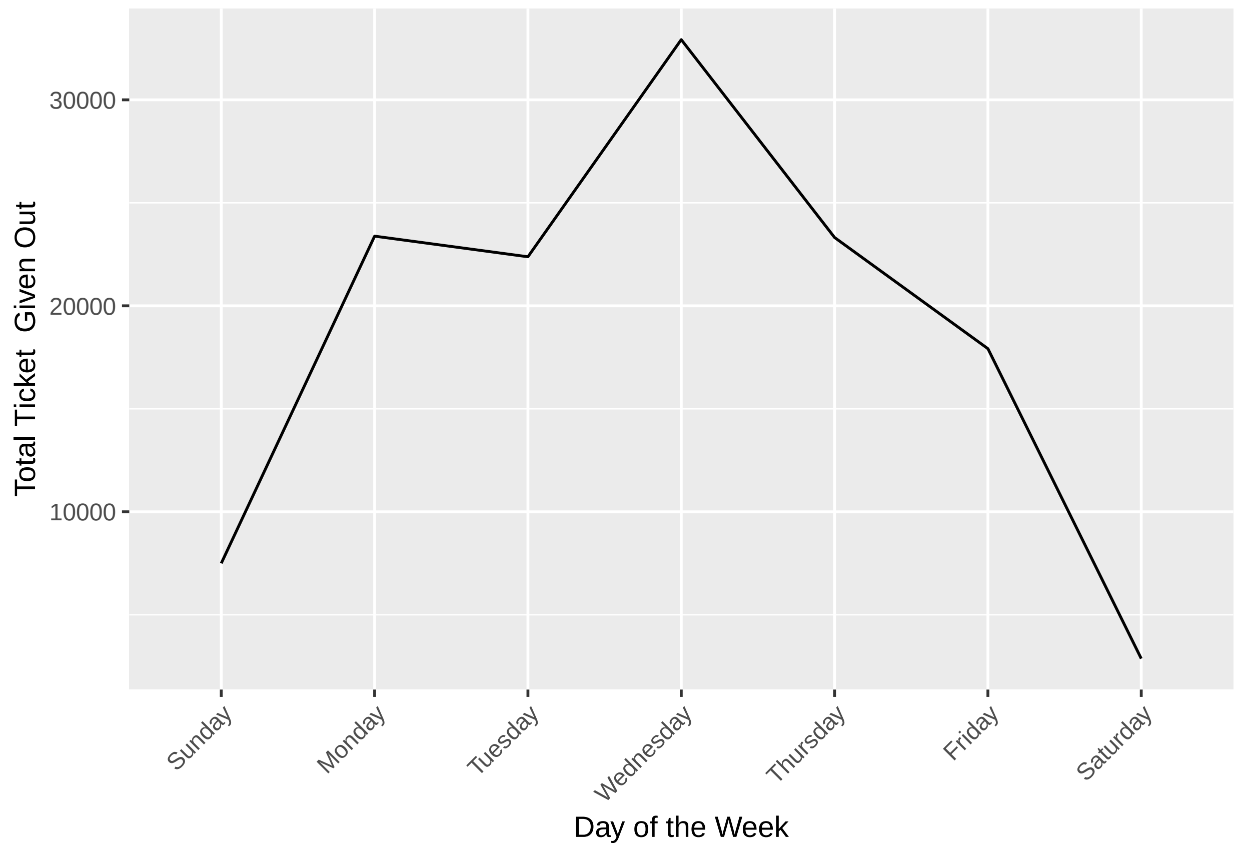
```
WeekdayCounts = as.data.frame(table(FTP$Weekday))

ggplot(WeekdayCounts, aes(x=Var1, y=Freq)) + geom_line(aes(group=1))
```



To make the graph easier to read we will label the x and y axis and put the days of the week in order.

```
WeekdayCounts$Var1 = factor(WeekdayCounts$Var1, ordered=TRUE,
                            levels=c("Sunday","Monday", "Tuesday","Wednesday",
                                     "Thursday", "Friday","Saturday"))
#We can change the Var1 variable to be an ordered factor variable
ggplot(WeekdayCounts, aes(x=Var1, y=Freq)) + geom_line(aes(group=1)) +
```

```r
  xlab("Day of the Week") + ylab("Total Ticket  Given Out")+
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
```



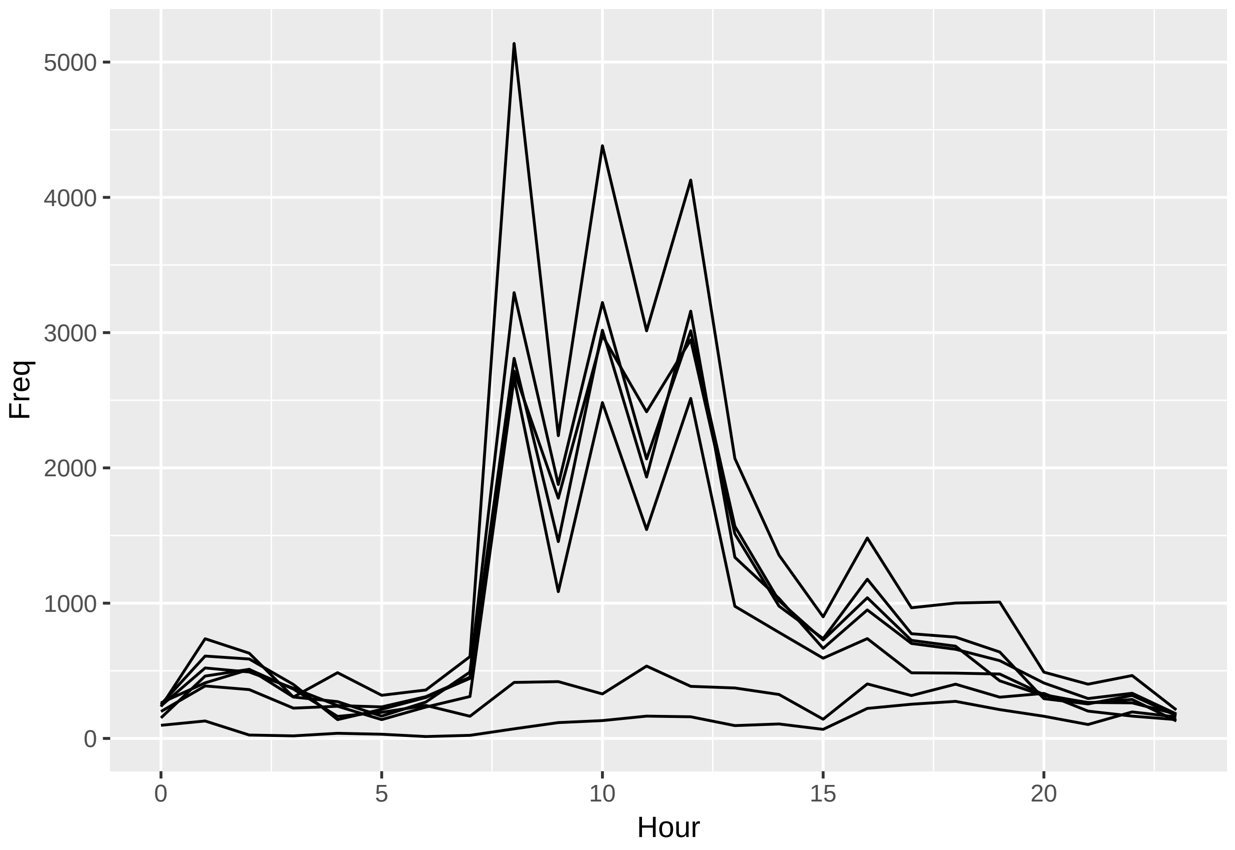
Wow, it looks like the most predictive ticketing day for the LA police is Wednesday. Now, let's add the hour of the day and visualize any patterns.

First, I'll create a frequency table for the weekday and hour and then we will put it into a data frame. The I'll visualize it with a line graph.

```r
# This will create our table
table(FTP$Weekday, FTP$Hour) %>% knitr::kable()
```

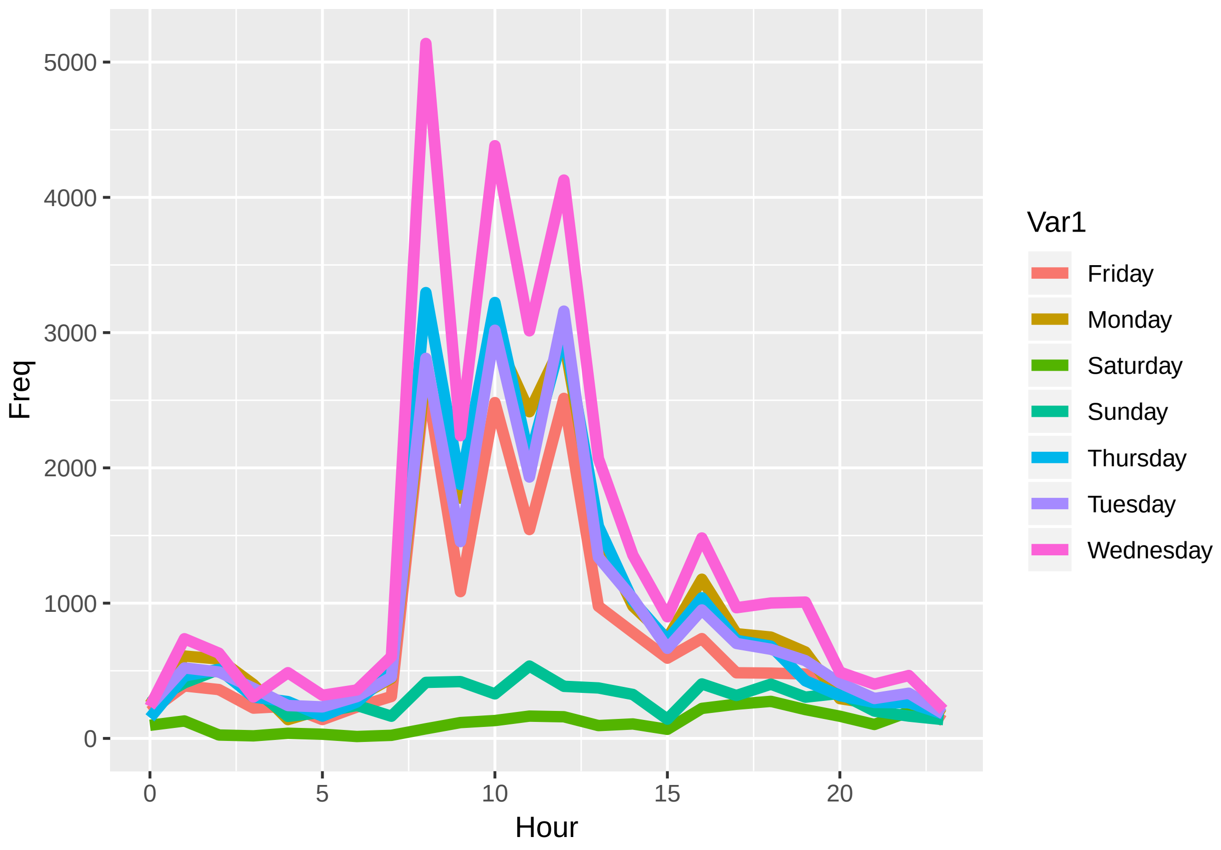
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1
Friday	199	388	361	224	238	139	232	310	2664	1085	2483	1544	2514	977	784	593	73
Monday	252	609	587	399	139	215	303	444	2716	1776	2978	2415	2947	1512	978	738	117
Saturday	97	129	25	19	38	31	14	23	71	117	132	165	160	95	107	67	22
Sunday	263	411	510	365	163	193	244	164	414	420	329	535	385	373	325	142	40
Thursday	152	461	511	306	272	165	269	491	3296	1877	3223	2066	3014	1569	1018	728	104
Tuesday	240	521	491	371	243	233	309	453	2810	1455	3018	1932	3159	1339	1035	666	95
Wednesday	235	737	630	306	486	319	358	605	5138	2238	4382	3013	4128	2070	1355	899	148

```r
#We will save this as a data frame
DayHourCounts = as.data.frame(table(FTP$Weekday, FTP$Hour))
# Convert the second variable, Var2, to numbers and call it Hour:
DayHourCounts$Hour = as.numeric(as.character(DayHourCounts$Var2))
# Create out plot:
ggplot(DayHourCounts, aes(x=Hour, y=Freq)) + geom_line(aes(group=Var1))
```



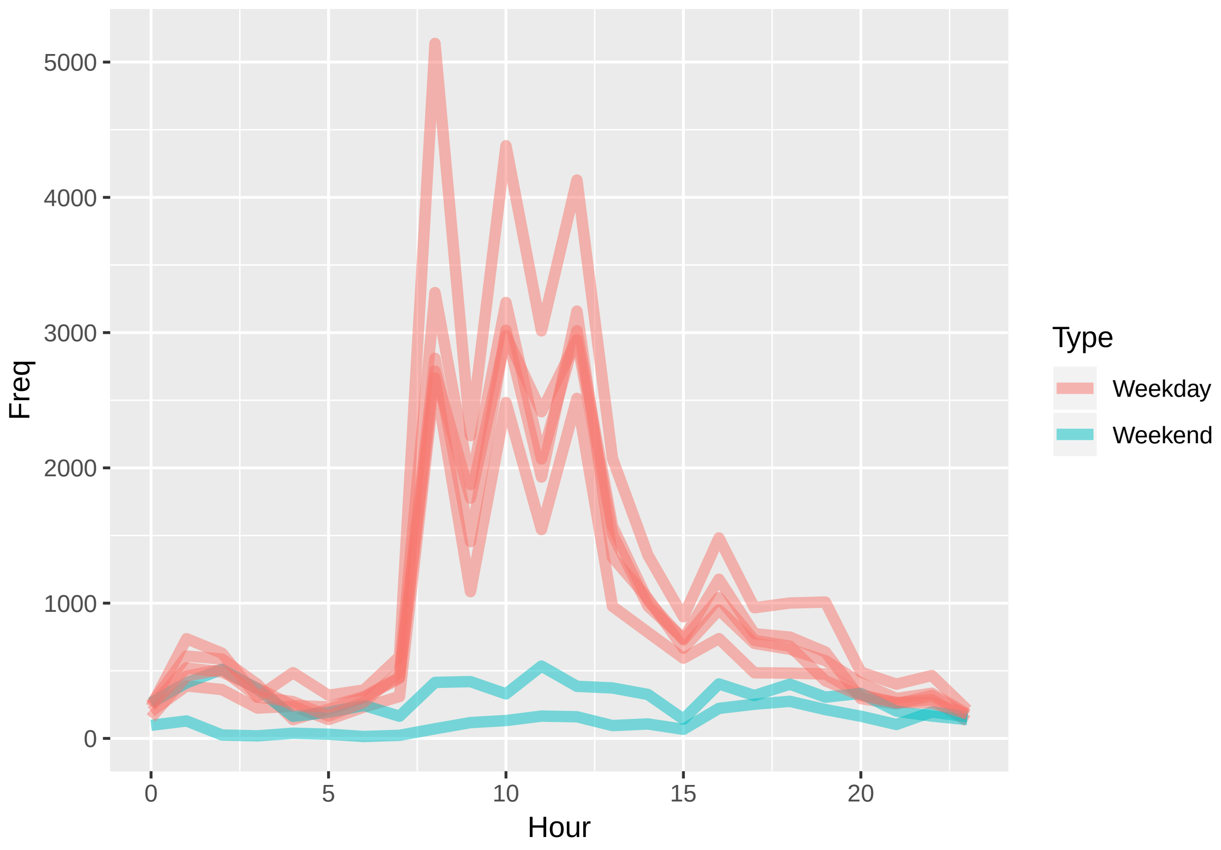
Some color will make this graph a lot informative

```
# Fix the order of the days and add color
ggplot(DayHourCounts, aes(x=Hour, y=Freq)) + geom_line(aes(group=Var1,
                                                        color=Var1), size=2)
```



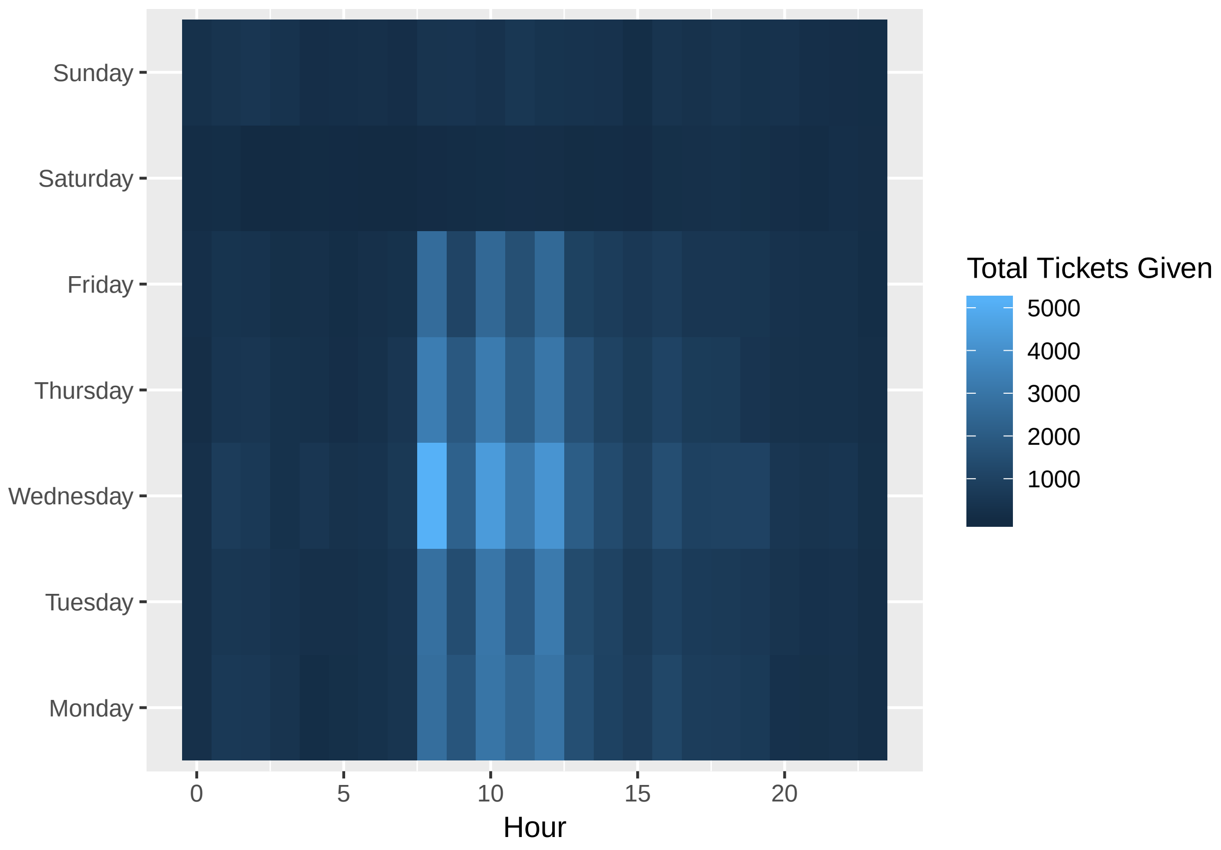
There is a distinction in the frequency tickets are given on weekdays vs weekends. Let's separate the two and visualize.

```
DayHourCounts$Type = ifelse((DayHourCounts$Var1 == "Sunday") |
                              (DayHourCounts$Var1 == "Saturday"),
                            "Weekend", "Weekday")
# Redo our plot, this time coloring by Type:

ggplot(DayHourCounts, aes(x=Hour, y=Freq)) +
  geom_line(aes(group=Var1,color=Type), size=2,alpha=0.5)
```



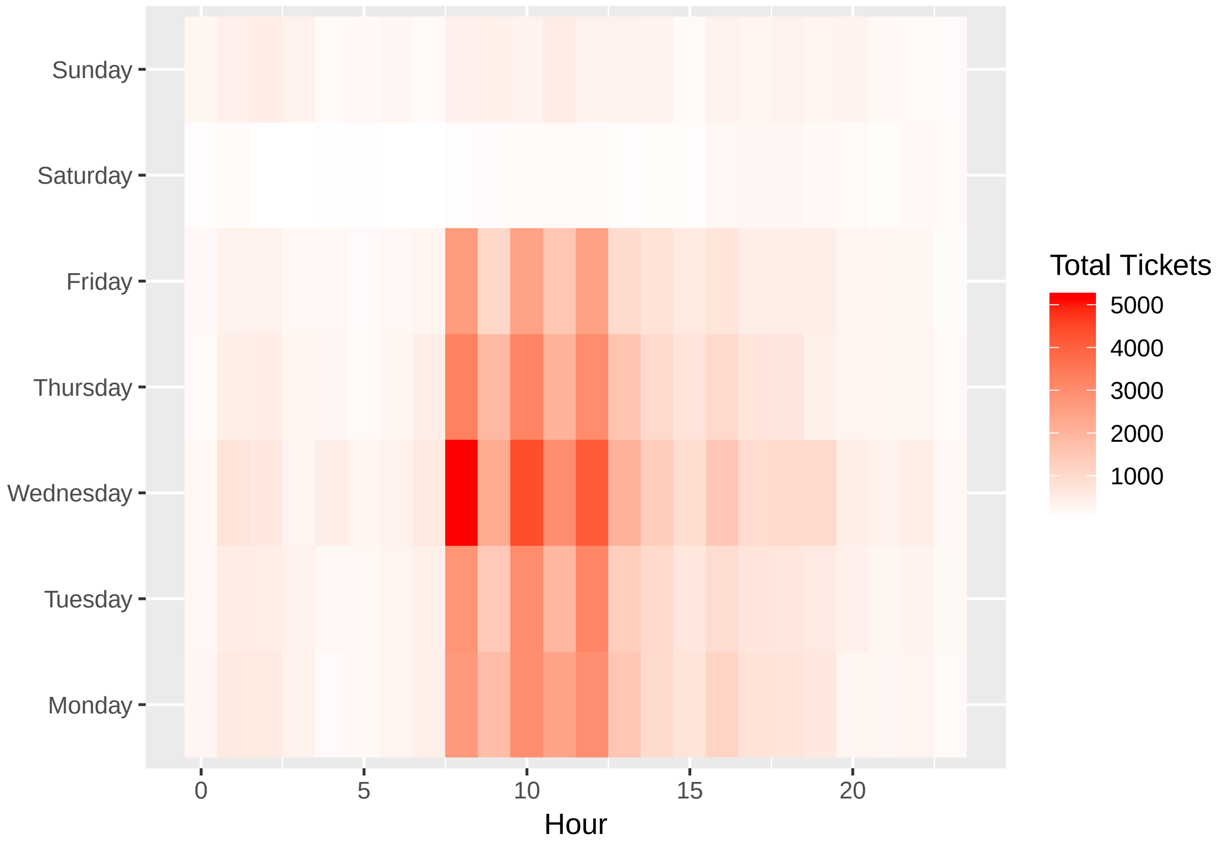
Another way to interpret this information is with a heatmap:

```
# Fix the order of the days:
DayHourCounts$Var1 = factor(DayHourCounts$Var1, ordered=TRUE,
                            levels=c("Monday", "Tuesday", "Wednesday",
                                     "Thursday", "Friday", "Saturday", "Sunday"))
#Change the label on the legend, and get rid of the y-label:
ggplot(DayHourCounts, aes(x = Hour, y = Var1)) + geom_tile(aes(fill = Freq)) +
  scale_fill_gradient(name="Total Tickets Given") +
  theme(axis.title.y = element_blank())
```



The dark blue is the low frequency and the light blue is the high frequency but let's give this heat map some heat and give and change the color scheme to red and white. Also, I will label the key.

```
ggplot(DayHourCounts, aes(x = Hour, y = Var1)) + geom_tile(aes(fill = Freq)) +
  scale_fill_gradient(name="Total Tickets", low="white", high="red") +
  theme(axis.title.y = element_blank())
```



The heatmap confirms that 8am, 10, and 12 am are the most popular times to get a ticket

Machine Learning Methods

It is always best practice to split your data into Before we get started I wanted to re-label the variables so that machine learning output will be easier to interpret.

```r
DayHourCounts <- setNames(DayHourCounts, c("Weekday","Hour","Frequency",
                                           "HourAsNumber", "Weekday/Weekend"))
#
set.seed(21)
test_index <- createDataPartition(y = DayHourCounts$Frequency, times = 1,
                                  p = 0.2, list = FALSE)
my_train <- DayHourCounts[-test_index,]
test <- DayHourCounts[test_index,]
```

The following code generates a function that will calculate the RMSE for actual values (true_ratings) from our test set to their corresponding predictors from our models:

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(\frac{d_i - f_i}{\sigma_i}\right)^2}$$

```r
RMSE <- function(true_Frequency, predicted_Frequency){
  sqrt(mean((true_Frequency - predicted_Frequency)^2))
}
```

Baseline Model

We will start with our baseline, the most basic prediction model. This is the average for all hours across all days and use this average to predict our ratings.

```r
mu_hat <- mean(my_train$Frequency)
mu_hat
```

```
## [1] 808.3182
```

Now that we have our $\hat{\mu}$ we can determine RMSE for our baseline method.

```r
Baseline_rmse <- RMSE(test$Frequency, mu_hat)
Baseline_rmse
```

```
## [1] 656.3121
```

We are getting a RMSE of about 656. Our prediction is on average 656 of citation off the actual amount of citations that are given for each day. This is the case for a couple of reasons, there seems to be days of very high amounts and then days that are very low tickets that are given. However, for this model this is the lowest RMSE we can have. To demonstrate we will use 1000 and 750 to prove that we will get a larger RMSE with another value than our baseline.

```r
predictions <- rep(1000, nrow(test))
RMSE(test$Frequency, predictions)
```

```
## [1] 725.2938
```

RMSE of 725.29 is worse than our baseline.

The following code will keep track with all of our test with this matrix.

```r
rmse_results <- data_frame(method = "Baseline", RMSE = Baseline_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	656.3121

Linear Regression

Now lets run a linear regression model to see if we can improve on our baseline model

$$y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

```r
LR = lm(Frequency ~ Weekday + Hour, data=my_train)
summary(LR)
```

```
##
## Call:
## lm(formula = Frequency ~ Weekday + Hour, data = my_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1668.79  -169.13   -50.72   243.51  2109.30
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   139.75     250.38   0.558  0.57796
## Weekday.L    -815.89     137.32  -5.942 3.95e-08 ***
## Weekday.Q    -382.49     135.49  -2.823  0.00572 **
## Weekday.C     365.33     132.31   2.761  0.00683 **
## Weekday^4     425.15     128.98   3.296  0.00135 **
## Weekday^5      16.33     130.60   0.125  0.90074
## Weekday^6     264.81     127.47   2.077  0.04028 *
## Hour1         255.40     355.50   0.718  0.47414
## Hour2         305.25     326.78   0.934  0.35245
## Hour3          68.07     376.43   0.181  0.85685
## Hour4         210.85     355.52   0.593  0.55444
## Hour5          52.20     338.93   0.154  0.87790
## Hour6         136.63     338.83   0.403  0.68762
## Hour7         215.96     326.78   0.661  0.51018
## Hour8        2304.39     326.78   7.052 2.16e-10 ***
## Hour9        1094.66     339.28   3.226  0.00169 **
## Hour10       2223.82     326.78   6.805 7.07e-10 ***
## Hour11       1400.51     338.84   4.133 7.35e-05 ***
## Hour12       2445.78     337.23   7.253 8.17e-11 ***
## Hour13        805.57     378.54   2.128  0.03574 *
## Hour14        655.46     353.68   1.853  0.06673 .
## Hour15        400.59     376.39   1.064  0.28971
## Hour16        726.61     378.56   1.919  0.05773 .
## Hour17        480.29     376.14   1.277  0.20455
## Hour18        487.20     338.93   1.437  0.15364
## Hour19        387.20     338.93   1.142  0.25595
## Hour20        193.11     326.78   0.591  0.55587
## Hour21        184.66     353.68   0.522  0.60273
## Hour22         39.63     338.82   0.117  0.90713
## Hour23         28.07     378.59   0.074  0.94104
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 555.6 on 102 degrees of freedom
## Multiple R-squared:  0.7634, Adjusted R-squared:  0.6961
## F-statistic: 11.35 on 29 and 102 DF,  p-value: < 2.2e-16
```

```r
LR_pred <- predict(LR, newdata=test)

LR_rmse <- RMSE(test$Frequency, LR_pred)
LR_rmse
```

```
## [1] 462.2858
```

If you happen to be more familiar with the Sum Square Errors or SSE for validation. I wanted to provide you the following code to validate my RMSE metric, as well as the RMSE function.

```r
LR_sse <- sum((LR_pred- test$Frequency)^2)
LR_sse
```

```
## [1] 7693492
```

```r
RMSE <- sqrt(LR_sse/nrow(test))
RMSE
```

```
## [1] 462.2858
```

When in math, facts are facts, when in doubt, prove your math :)

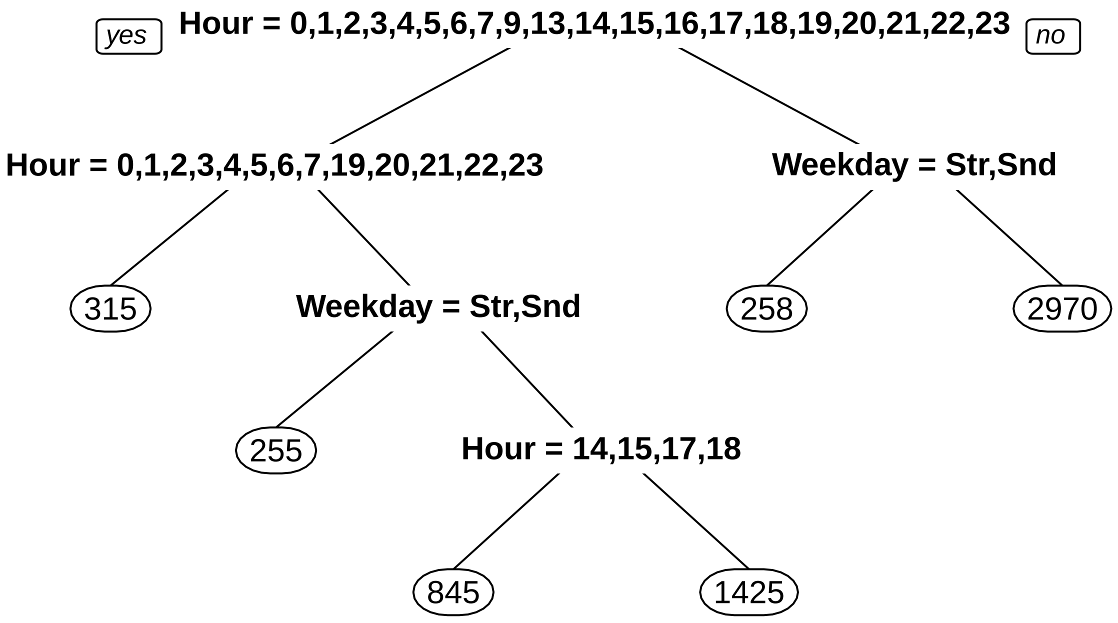
```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Linear Regression Model",
                                     RMSE = LR_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	656.3121
Linear Regression Model	462.2858

Regression Tree

Lets se if we can further improve this with a regression tree or CART model

```
tree = rpart(Frequency~ Weekday + Hour, data=my_train)
prp(tree)
```


yes
Hour = 0,1,2,3,4,5,6,7,9,13,14,15,16,17,18,19,20,21,22,23
no
Hour = 0,1,2,3,4,5,6,7,19,20,21,22,23
Weekday = Str,Snd
315
Weekday = Str,Snd
258
2970
255
Hour = 14,15,17,18
845
1425

```
tree_pred = predict(tree, newdata=test)
tree_sse = sum((tree_pred - test$Frequency)^2)
tree_sse
```

```
## [1] 2109051
```

Results

```r
Tree_rmse <- sqrt(tree_sse/nrow(test))
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regression Tree Model",
                                     RMSE = Tree_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	656.3121
Linear Regression Model	462.2858
Regression Tree Model	242.0429

Conclusion:

What I really enjoyed about this project was that I able to do some extensive exploratory data analysis to know the patterns that I hope my machine would realize. Then, I was able to build different machines and test their predictions. I could then determine which was the best method for this project. Ultimately the beauty of this project, was finding out that my machine would learn from the data and see pattern from this data and learn. With this validation you can know that no matter how patterns change in parking ticket citations, the machine will learn and adapt.

Some further insights :

I really enjoyed some other tools in R that I'm unable to display in a PDF because they are reactive is a cluster map and a heatmap. I added some screan shots below because I thought it was worth sharing. (I plan to put some pics of the reactive maps and maybe create a quick shiny app and place a link for them to click onto.)