```
---
title: "Predictive Police Ticketing: Using Machine Learning to Detect
Patterns"
author: "Nathan Siefken"
date: "2/3/2019"
output:
  pdf_document: default
  html_document:
    df_print: paged
---
```

Motivation

Imagine that you are in downtown Los Angeles and you are the waiting for your name to be called at the building department on a Wednesday. Your business project is on hold pending these building permits. It is 12 noon and your number is about to be called, but, this took longer than you expected, and your parking meter is about to run out. What do you do? Or maybe you need to run into the Santa Monica grocery store to buy milk for your baby at 8am on a Saturday. You find a parking spot just in front, but it's turns out to be a red curb. The cry of your baby is pulling at the threads of your maternal/paternal instincts. Is it worth a ticket? Are you likely to get a ticket?

I am not encouraging you to break the law. In fact, you have 0 chance (assuming no human police error) of getting a ticket if you follow the rules. However, life is full of circumstances, and a person may want to have more information to help aid them in the decisions. Knowledge is empowering, no matter the decision one decides to take.

Goal

I am going to use machine learning to detect patterns of police ticketing in Los Angeles. I will train a couple of models and keep a running table of results as we progress through the report. Before, I get into the machine learning process, I will need to wrangle the data to make it friendlier to use. Then, I will do a exploratory data analysis, full of visualizations to help better understand the data.

Then, I will subset my data into a group that my machines will learn from and another that I will test my models on. We will find out if Linear Regression Model or a Tree Regression Model preforms better with our data. To determine the best method, I will use Root Mean Squared Error or RMSE as a guide. Kaggle, also uses RMSE or MAE as the metric for judging their competitions.

Lastly for fun, I provided a link to a fun interactive dashboard tool that allows the user to manipulate a heatmap and a cluster map just by changing the dropdown menu in the search parameters. I built this reactive geo-mapping tool completely in R.

Data Set

Data Set

I found this data set on, Kaggle, an online community of data scientists and machine learners, owned by Google, Inc. This data set, is maintained and regularly updated by Kaggle which is acquired from the city of Los Angeles organization page.

Kaggle Data Set: https://www.kaggle.com/cityofLA/los-angeles-parking-citations/home
Los Angeles Organization page : https://data.lacity.org

*The variable includes the parking citations with latitude / longitude in US Feet coordinates. *

\pagebreak

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.

```{r, warning=FALSE}
# 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)
}

```

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

# loading libraries
library(tidyr)
library(dplyr)
library(ggplot2)
library(lubridate) # for working with dates
library(plotly) # for interactive plots
library(janitor)
library(leaflet) # Geomapping
library(colorRamps)
```

```
library(proj4)
library(validate)
library(stringr)
library(rpart)
library(rpart.plot)
library(caret)
library(knitr)
FTP<- read.csv("FTP.csv", stringsAsFactors = FALSE)
```

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.

```{r pressure, echo=FALSE}
#Now to put our time into a format that we can use
FTP$Issue.time<-sub("(\\d*)(\\d{2})", "\\1:\\2", FTP$Issue.time) # put in delimitter
# The single digit strings were missed, so this code will convert them
FTP$Issue.time <- str_replace(FTP$Issue.time, "^([0-9])$",":0\\1")
# Now we will need to pad our times with a 0 before the ":" for all data that was less than 1:00
FTP$Issue.time <- sprintf("%04s", FTP$Issue.time)
```
In addition, there are a few 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.
```{r}
#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.

```{r}

#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 data frame
FTP<- cbind(FTP, data.frame(project(data.frame(FTP$Latitude,
FTP$Longitude), proj = pj, inverse = TRUE)))
str(FTP)
FTP <- FTP[-9:-10] #This removes the Latitude and Longitude  in Feet
from our table
names(FTP)[c(9, 10)] <- c('Longitude', 'Latitude') #Rename column names
of converted

```

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

```{r}
#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 these rows will not affect our results.

<p>
```{r}
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.
# We have no further use for it, we shall remove it.
```

##Exploratory Data Analysis section:
First, let's see how much revenue was generated last month just from ticketing.

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


Filter top 10 Violations
```{r}
TopViolations <- FTP %>%
  group_by(Violation.Description) %>%
  tally() %>%
  arrange(-n) %>%
  head(10)

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


Now, let's graph top 10 Violations for the past month.

```{r}
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 bulk of the tickets are "NoPark/Street Clean". There also, appears to be some peaks and valleys. We will investigate further to understand what this could mean.

```{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 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()
```

Lets visualiz this data to make it much easier to interpret.
```{r}
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.
```{r}
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)) +
  xlab("Day of the Week") + ylab("Total Ticket  Given Out")+
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))

```

Now, 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.

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 counts towards 23 or 11:00 PM.

```{r}
# This will create our table
table(FTP$Weekday, FTP$Hour)
#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))
```

We can see there are some patterns, but they are a little hard to make put what they are. We will add some color to the days of the week to make it easier to read.

```{r}
# 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)

```

Now, that is informative! We can see a distinction between days and frequency. Let's further make a distinction between the weekdays and the weekends.

```{r}
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)

```

Now it is a little easier to see what is going on.

Another way to interpret this information is with a heatmap:

```{r}
# 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.

```{r}
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
Before we get started I wanted to re-label the variables so that the machine learning output will be easier to interpret. The we will split the partition the data into a training and test set.

```{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_Frequency) from our test set to their corresponding predictors from our models:

   \[
    \makebox[\linewidth]{$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}
{\Big(\frac{d_i -f_i}{\sigma_i}\Big)^2}}$}
    \]

```{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
```

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

```

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 to prove that we will get a larger RMSE with another value than our baseline.

```{r}
predictions <- rep(1000, nrow(test))
NotAsGood <- RMSE(test$Frequency, predictions)
NotAsGood
```
RMSE of 725.29 is not as good as 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()
```

#Linear Regression

Linear regression is a global model, where there is a single predictive formula that is used to determine an entire data-space. When the independent variables interact with each other in linear fashion this model works really well.

Now let's run a linear regression model to see if we can improve on our baseline model

\[
\makebox[\linewidth]{${Y}_{}=a+b_1X_1+b_2X_2 +\varepsilon$}
\]

Y is your prediction, a is the intercept, b is the slop, X is the observed score on the independent variable and ε represents the residuals.

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

```
```

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

LR_rmse <- RMSE(test$Frequency, LR_pred)
LR_rmse
```
The equation for how Sum of square errors SSE is related to Root Mean Square Error RMSE is as follows :

$$
\makebox[\linewidth]{$RMSE = \sqrt{\frac{SSE}{N}}$}
$$

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

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

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

##Regression Tree

Let's see if we can further improve this with a regression tree or CART model

Regression trees use recursive partitioning to separate data in to smaller regions that are similar. This method works really well when the data interacts in complicated nonlinear ways. The tree will start with a root node often times have branches that partition the data will lead to the leaves or terminal nodes.
<p>

```{r}
Tree = rpart(Frequency~ Weekday + Hour, data=my_train)
prp(Tree)
```

```

```{r}
Tree_pred = predict(Tree, newdata=test)
Tree_rmse <- RMSE(test$Frequency, Tree_pred)
Tree_rmse

```

Let's interpret this tree using the two examples from the beginning.

The first person's meter was about to expire at noon. Since, "12" is not listed in the root node, you would go to the first branch on the right. It was on Wednesday, which leads you to the far-right leaf or terminal node, 2970. This means at this time day on this day of the week, our model predicts that there are 2,970 tickets being issued. This is a lot.

The second person is considering illegally parking at 8am, which is also not in the root node. Again, we will progress to the right side of the tree. It is a Saturday; we would move left leaf from this branch. In this circumstance, there are only 258 tickets being issued at this time for this day of the week. This is pretty low.

To take the risk or not, is not for me to decide, I am just putting forth additional information for that person to better make their decision.

#Cross Validation

When using machine learning, it is important to be mindful of overfitting or underfitting models. Cross validations allow for an opportunity to understand our tree a little better while finding out if our tree is the right size.

We can evaluate our tree by plotting the cp. Our tree has 6 terminal nodes and has a cp of .011. The graph below shows us that the first split gives us the largest improvement and as we continue to split the smaller the improvement. Now why don't we prune our tree to have just 3 or 5 terminal nodes, where it appears that we have diminishing returns? It turns out that the rpart function automatically applying a range of cost complexity that determines the size our tree.
```{r}
plotcp(Tree)
```

I am going to prune our tree to have five leaves and see how it predicts our test set. For this data set, I agree with where rpart dictated the size of our tree to be. The difference in RMSE was a lot. If the
```

difference in the RMSE was marginal, it would have been an indicator of overfitting and I would have pruned it back.

```{r}
Tree1 <- rpart(Frequency~ Weekday + Hour, data=my_train,
    control = list(cp = .021))
prp(Tree1)

Tree1_pred = predict(Tree1, newdata=test)
Tree1_rmse <- RMSE(test$Frequency, Tree1_pred)
Tree1_rmse

```

#Results
```{r}

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

We can see that the regression tree performed very well against the baseline and linear regression model. During our data exploration analysis, we could see in our scatter plot a cloud of points at the top of the graph as well as at the bottom, which shows that the data isn't very linear. With that in mind, linear regression did out preform our baseline model.

#Conclusion:

We have effectively used machine learning methods to detect patterns of police ticketing in Los Angeles and successfully predicted the amount of tickets being issued on any given hour, on any given day of the week. We learned choosing the most appropriate machine for your data is important because they can vary in accuracy. For our example situations, these two people could have used this information provided by our model to help with their decision making.

\pagebreak

Bonus Section :

I was able to utilize the Longitude and Latitude coordinates with a cluster map and a heat map. Since, I was unable to display these reactive maps on a PDF, I used shiny to create a web-based dashboard

completely in R. You can find it at the following URL:
<p>
https://nathans.shinyapps.io/LA_Parking_Violations