Forecasting Median Daily Fares of Chicago Taxi Drivers in 2017 with an Autoregressive Integrated Moving Average (ARIMA) Model

2018 Texas A&M Institute of Data Science Competition

Jose Alfaro, Steve Broll, Kevin Chou

Executive Summary

The Chicago taxi industry has changed in recent years as the ridesharing companies Uber and Lyft have rapidly grown in the city. A topic of special interest is how individual taxi drivers have been affected by the changes to the industry as a whole. The team consisting of Jose Alfaro, Steve Broll, and Kevin Chou will study changes to the industry on both a large-scale and an individual level with the following topics of interest:

- Exploratory data analysis and visualizations
- Predicting density of taxi pickups by location
- Forecasting median daily salary of taxi drivers

This project will be using data provided by the Texas A&M Institute of Data Science taken from the City of Chicago's public database. The programming languages R and Python will be used for the analysis and modeling of the data, and Perl will be used to subset the data for preliminary analysis. Exploratory data analysis will first be performed in R, and a random forest model will be trained in python for predicting taxi pickups by location. The data has seasonal components as well as a trend component, and therefore will be treated as a time series problem. The time series modeling method used will be Autoregressive Integrated Moving Average (ARIMA), specifically one model using the auto.arima function from the forecast package in R and the other created by manually choosing coefficients by examining the Autocorrelation function (ACF) and Partial Autocorrelation function (PACF) plots to create a seasonal ARIMA model forecasting daily median salary. A combination of AIC, ACF1, and RMSE will be used to compare the auto ARIMA model, the manual ARIMA model, and a naive Time Series Linear Model (TSLM) using seasonal and trend components.

Introduction

Chicago today is a city of about 2.7 million people filled with tourist locations, business skyscrapers, and nightlife hotspots. The combination of high density destinations and strong tourist industry create a huge market for taxis, and Chicago's taxi drivers have historically had a higher salary than those in most other cities, with a median of about $36,000 a year, with a few hotspots being moneymakers for these drivers. In recent years, however, the rideshare industry has changed the landscape of the transit markets in Chicago.

Two competing ridesharing companies, Uber and Lyft, have seen rapid growth in recent years across the United States. While it's possible that a portion of their business comes from growth in the ridesharing/taxi market itself, the rise of these ridesharing companies (and other smaller companies) has resulted in the decline of the taxi industry in many areas. Chicago's taxi industry

in particular has taken a hit, as Uber and Lyft grew in the city with no major restrictions being placed on the ridesharing companies until the summer of 2016. The data used in this competition contains four years of training data and a segment of 2017's data, which allow us to both explore Chicago's taxi industry as a whole and fit predictive models on variables of interest. We are interested in what changes individual taxi drivers in Chicago experience, and forecasting future changes to these taxi drivers is of huge interest to the city of Chicago, especially as they consider adding harsher legislation to counter ridesharing companies. Forecasting the median daily fares will give us an idea of both how much business a typical taxi driver will see in a day as well as how much they will earn. In doing so, we aim to predict the future prospects of a typical Chicago taxi driver.

Exploratory Data Analysis

In order to efficiently perform exploratory data analysis, we used a subset containing 1% of the data for each of the provided training datasets so that our subset's yearly proportions were the same as those of the full data's. Hourly, daily, monthly and yearly trends were compared over repeated subsets to demonstrate that the trends in each subset were representative of the full data's trends. It should be noted that the EDA plots and summary statistics examined in this section of the report are all run on the same 1% subset.

To create the visualizations used in Section 1 of the appendix, we created our subsets from the data with Perl, then loaded those subsets into R. We then processed the data to match up column names and to bind the data for 2013-2016 together. We used basic R commands like summary, median, and plot(x,y) for general exploration, and used functions from the ggplot package to produce the tables shown in the appendix. Segments of our cleaned up code are given at the bottom of the appendix.

While exploring the data, we discovered some large scale trends. The number of rows (trip counts) in each dataset was noticeably different, with a decline from 2014 to 2016. Fares, trip lengths, and trip distances declined as well, though less dramatically than counts. Tips and tolls do not follow yearly trends. Each subset contained some extreme outliers, with some distances or fares being extraordinarily high relative to the other. We found that taking the medians of these variables with outliers by day/week/month erased the effects of the outliers, and that it was not necessary for EDA to exclude these outliers.

Before taking a deep dive into how individual taxi drivers have been affected by changes to the industry, it is important to study the underlying seasonal trends and year to year changes of the industry as a whole. To effectively do so, we looked at median trip duration and the total number of trips respectively by hour of day, day of week, and month of year. Median trip duration was chosen as it is highly related to trip fare, our primary variable of interest for analysis. Trip count

was chosen to show both the seasonality of taxi demand and the changes in demand from 2013 to 2016.

We will begin by studying median trip duration over the years. First, we look at **Figure(1.1)**. With a slight exception in 2013 each year has very similar trends, with a peak from May to June followed by a steady decline until another peak in October. 2016 appears to be smaller in magnitude than the previous years, suggesting that trip duration might be declining. **Figure(1.2)** shows us that the proportions of trip duration by day of the week remain constant across the years, with Friday and Saturday peaking before a sharp decline to Sunday. Like in the previous figure, the trends in **Figure(1.3)** appear constant over the years. Trip duration bottoms out at 5am and, unsurprisingly, peaks sharply from approximately 5-8 PM, when many people get out of work and/or go to dinner.

Next, we will take a close look at total trip counts. **Figure(1.4)** clearly shows the difference from year to year, with a rise from 2013 to 2014 and a sharp drop from 2015 to 2016.

The interval March-June is the highest overall, though 2013 and 2016 have more of a general upward and downward trend, respectively. Like with time duration, there is a peak each year in October. **Figures(1.5)** and **(1.6)** have trends identical to those in **Figures(1.2)** and **(1.3)**, while the trends are slightly different on the monthly scale.

By studying the trends of counts and median duration, we are beginning to paint a picture of Chicago taxi trips. A variable of interest that has not yet been examined, location, can provide us with a more literal picture of these trips. Since taxi trip counts showed the clearest trend over years while having very similar trends by day and hour, we will take a closer look at the density of taxi trips by location and look for key points by multiple levels. Specifically, we will look at the counts at each coordinate (to 4 decimals) by hour of day, day of week, month of year, and across years. In addition to visualizing the data from the training data, we wanted to see how useful some basic machine learning models could be in predicting future counts at a given coordinate.

Using the 1% subset, we tried out two ML techniques that we believed could be successful in this endeavor: Random forests and K-nearest neighbor algorithms. After performing initial tests on the 1% subset for comparisons, we found that random forests were both significantly faster and produced much lower errors and error rates.

Predicting Pickup Density using Random Forest

To predict taxi pickup density, we count the number of taxi pickups in the city of Chicago at a unique longitude and latitude coordinate, year, month, day of the month, and hour of day. We then do some investigations to see if we can find any explanations for any trends we find.

Model Results

After running our model, we find that location (latitude and longitude) was by far the most important feature in predicting number of pickups. We also saw that while there were definitely some trends in number of pickups at all levels of time, the most distinguishable trends were found at different hours of the day. Because we will turn our focus on pickup density at the hour level.

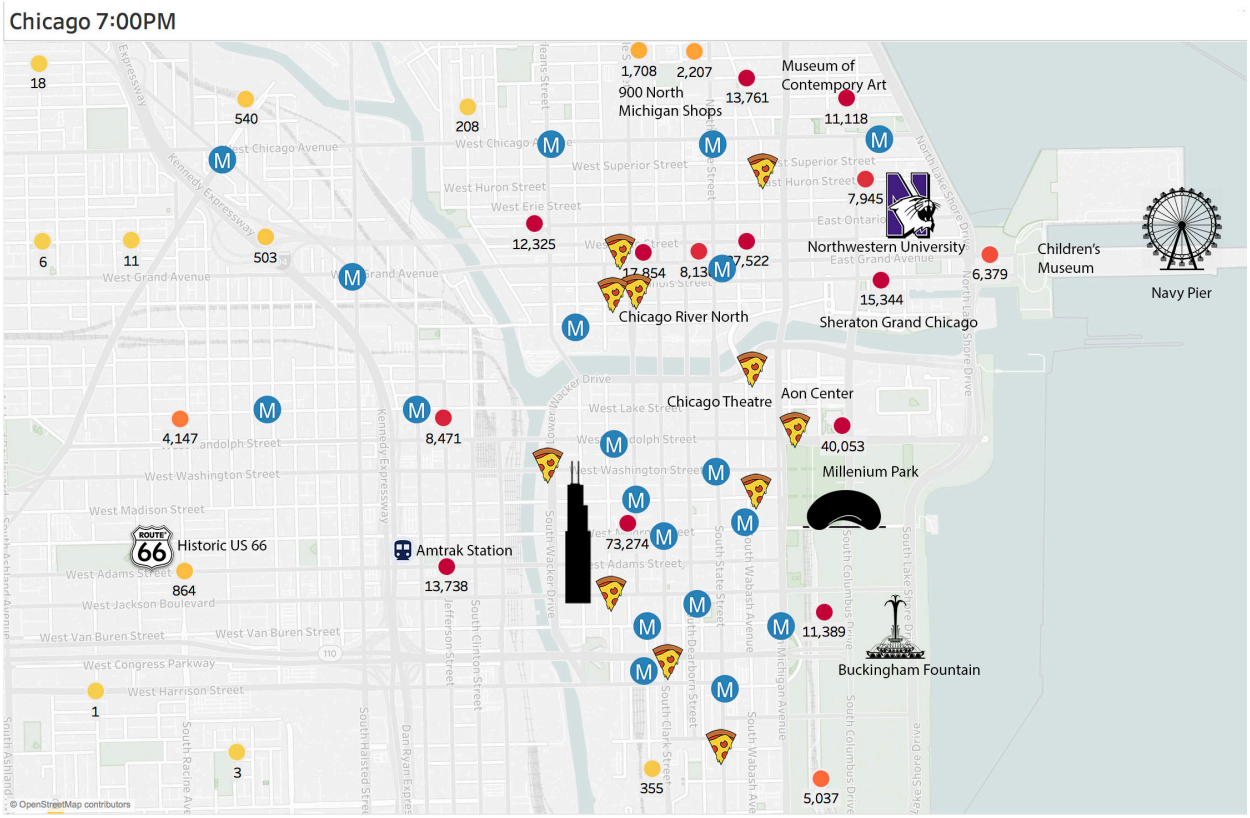


Figure (2.1)

Figure (2.1) is a map of Chicago at 7:00 the busiest time of the day we found, where the pickups are most concentrated. The blue "M" icons represent metro stations for the Chicago metro subways and the pizza icons represent restaurants that serve Chicago's famous deep dish pizza. Notable tourist attractions are also represented on the map with their respective icons. From **Figure (2.1)** we can see that the two points with the most pickups (73,274 and 40,053) are the

closest in proximity with the two hottest Chicago tourist attractions: Willis Tower and the Bean. These high density points can also be found in near proximity to the Chicago metro stations. However, we believe that the biggest influencing factors for taxi pickups is the proximity to the metro stations. With the one of the largest metro stations systems in the U.S, only second to New York City, and Chicago being one of the biggest cities for business, it makes sense why that dense pickup locations are all in close proximity to a metro station. To see how the Chicago taxi pickup density changes over at all levels of time, we can use the links provided below:

- **Pickups by Year:** https://streamable.com/756m1
- **Pickups by Day:** https://streamable.com/ve4fk
- **Pickups by Month:** https://streamable.com/jjxbn
- **Predicted Pickups by Day:** https://streamable.com/jakyi
- **Predicted Pickups by Month:** https://streamable.com/uyezj
- **Pickups by Hour:** https://streamable.com/04e1x

Assessing the model

Performing prediction on the 2017 test data, we get only a 43.27% accuracy rate, not so good. However, for our particular study, a low prediction accuracy does not necessary mean that our model is not good. This accuracy score only accounts for the number of pickups that our model has predicted exactly and does not factor in the proximity of our prediction, and considering the scale of the data we are working with, the proximity to the exact solution is pretty important to assess. For example, if a specific location had 1,000 true pickups, but our model predicted that same location would have 998 pickups. That prediction is not exactly correct but since we are working with a dataset containing several million pickups, a prediction that is off by only a factor of 2 is fairly accurate. So to take a further inspection on how good our model is, we must take into account how far our predictions are off from the true values and we can do that by calculating the root mean square error.

RMSE

Our calculated RMSE for our model was 0.141, but that is in log space so we must transform back. Doing so, we obtain a value of 1.382. That means that the other 56% of our predictions are only a factor of 1.382 or less away from the true value. Now remember we are working with a dataset containing several million pickups, so our model being only a factor of 1.382 pickups away from the true pickup values means that model is predicting pickup values very close to the actual number of pickups. Knowing this, now our 43% prediction accuracy is actually quite considerably good.

Modeling and Analysis

After conducting our initial exploratory data analysis on the 1% subsetted data, we concluded that there was a strong change in median fare over time in a taxi driver's median daily fare and time variables such as year, month, day, and hour. We chose to analyze the data with respect to daily values as opposed to hourly, weekly, or yearly time intervals. This is simply because hourly values contained too many sources of variation (seasonal effects), weekly values did not provide enough information as to determine the effects of weekends vs weekdays, and yearly values did not provide nearly enough information to produce any significant analysis. **Figure(3.1)** demonstrates the daily median taxi fare from 2013 to 2016. From this plot, it is clear that there exists a correlation between the two variables as well as some form of seasonal effect which dictates the pattern of the data at certain parts of the year. This further supports our claim for the need of some form of a time series model.

The next step in analyzing this time dependent data, was to find out if the general trends shown by the 1% subset of the median daily fare date resemble that of the full data (**Figure(3.1)**). As it turned out, the general trend between days and median fares were quite similar, however, the plots produced seemed to contain a lot of noise which made it difficult to see a clear trend. Therefore, to remedy this, we computed both weekly and monthly moving averages in order to minimize the noise and get a clearer picture of the overall trend direction. A moving average is the mean of the data during several consecutive periods and is computed by averaging a series of past and future numerical values from a time series data. For instance, we calculated the weekly moving average by taking a series of 7 sequential numbers (3 prior observations, the current observation at time t, and the following 3 observations) to create a single point. Likewise, we calculated the monthly moving average by taking a series of 30 sequential numbers and averaging them to create a single point. This concept can be visualized in **Figure(3.2)**. It is important to note that the median taxi fare over time is represented as red while the weekly and monthly moving averages are represented as blue and green respectively. Also, it is clear from this plot that the raw data contains high levels of variation and fluctuates over time making it difficult to detect a general trend while the monthly moving average seems to be too robust and and doesn't respond well towards seasonal effects. With that being said, the weekly moving average seems to be the best method for analyzing trends in the data as it is not too robust towards variation and displays seasonal effects while not being overly sensitive to them at the same time.

Since the weekly moving average proved to be the more efficient form of measurement, we decomposed the data with respect to the weekly moving average in order to analyze the seasonal effects as well as the general trend of the data with minimal noise. This can be seen in **Figure(3.3)** which shows the raw data being plotted in the top row, the seasonal effects of the data in the second row, the general trend throughout the years in the third row, and the residuals of the data with respect to time on the fourth row. However, in order to conduct a valid time

series analysis, we must remove the seasonality effects from the data. **Figure(3.4)** shows the

weekly moving average with the seasonality component subtracted from the original series, hence a deseasonal moving average. We then performed an Augmented Dickey-Fuller test on the deseasonal data in order to determine the stationary status of the weekly moving. Since the test provided a p-value of 0.01, we rejected the null hypothesis and concluded that the data was indeed stationary. However, looking back at our deseasonal weekly moving average plot from **Figure(3.4)**, we see that the data has unequal variances with no stable mean. This means that we are unable to extrapolate any meaningful information by conducting time series analysis. To remedy this, we differentiated the deaseasonal weekly moving average data in order to adjust for the unequal variances and unstable mean. Then, we conducted another Augmented Dickey-Fuller test to ensure we once again had stationary data. **Figure(3.5)** demonstrates the data after differentiating and shows a clear mean throughout the data centered at around y = 0 with a variance that is about the same throughout.

After making sure that our assumptions were met, we proceeded to create both autocorrelation and partial correlation plots in order to determine the severity of the correlation within the data. The autocorrelation function gives the correlation of a time series with its own lagged values while the partial autocorrelation function gives the partial correlation of a time series with it's own lagged values while controlling the values at the shorter "legs". The plots for the autocorrelation and partial correlation functions are displayed in **Figure(3.6)** and **Figure (3.7)** respectively. In **Figure(3.6)**, we see that there are significant auto correlations at lags 1, 2, 3, and 4 while the partial correlation plots in **Figure(3.7)** showed significant spikes at lags 1 and 8. This indicated that we needed to test models that incorporated autoregressive (AR) factors or moving averages (MA). In other words, we needed to form an ARIMA model with components of 1,2,3,4 or 7. Since our autocorrelation plot (**Figure(3.6)**) inverts right after the 4th lag, we decided to use p = 3 as our autocorrelation coefficient. Also, since our partial autocorrelation plot (**Figure(3.7)**) seems to have a repeating spike at every 8th lag, which could be explained by a weekly seasonal effect, it suggests that we must use q = 8 in our model. The last component of the ARIMA model was determined earlier on in the analysis when we determined the stationarity of the data after differencing. Since we achieve equal mean and variances throughout the data by differencing only once, we conclude that the final ARIMA parameter d is equal to 1.

After obtaining our own ARIMA order values (3,1,8) by hand, we wanted to compare our model against the optimal model obtained by the function auto.arima located in the forecast package in R. The function auto.arima finds the optimal order values by minimizing and comparing the AIC values of all possible order coefficients. As one could imagine, this process takes a very long time to compute, so therefore we opted to use a stepwise auto.arima parameter in order to cut down run time dramatically. After running the auto.arima function, the optimal order coefficients produced were (4,1,4), which differed greatly compared to our handpicked (3,1,8).

In order to test the importance of the seasonal coefficients, we took our handpicked "naive" model and made an identical copy to which we turned off seasonality thus ultimately resulting in 3 different ARIMA models. The first ARIMA model shown as **Figure(3.8)**, is our handpicked naive model with seasonality turned off. It is important to notice that the data provided for 2013-2016 is shaded in as black while the forecasted predictions are displayed in dark blue. The forecasted data spikes up to about $150 and then plateaus for the duration of 2017. This is due to the lack of seasonality coefficients. It is also important to recognize that there are two shaded regions produced in the forecast of all three models. These shaded regions are prediction intervals with dark blue pertaining to an 80% prediction interval and light blue to a 95% prediction interval. Note that the prediction intervals get wider as time increases. This is natural since things become harder to predict, and thus more uncertain, as time increases. The second ARIMA model, shown as **Figure(3.9)**, is again our handpicked naive model, however, this time seasonality is turned on. Since seasonality is turned on for this model, we are able to see the forecast adapt to seasonal trends and better forecast what the median daily fare in 2017 would be like for the average taxi driver in Chicago. Lastly, the model depicted in **Figure(3.10)** shows the optimized AIC chosen order coefficients selected by auto.arima. Again, this model has seasonality activated and thus is able to forecast future values while taking into account seasonal trends. It is important to note that these prediction intervals for this model are much larger than those in **Figure(3.9)**.

The last thing we did to ensure that our ARIMA model was the correct way of modeling this data, was to produce an entirely different time series model, namely a time series linear model. The two major differences between an ARIMA model and a time series linear model are that the time series linear model is not produced from stationary data and does not take into account autocorrelation.

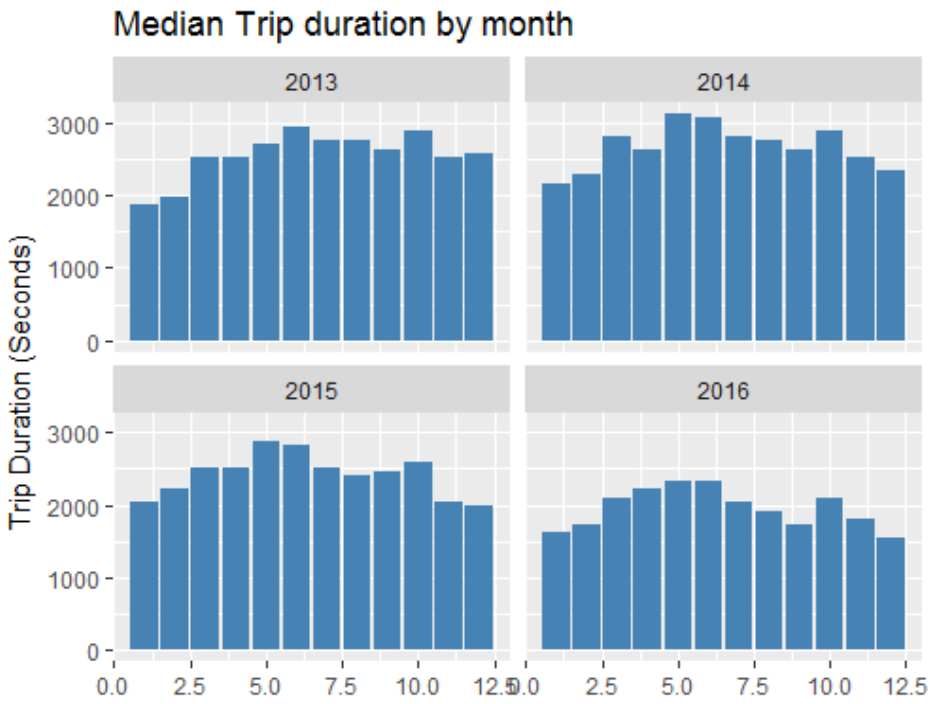
After having these 4 models to compare to, we took a look at several measures of accuracy to assess model fit (**Figure(3.11)**). By taking a look at the root mean squared error (RMSE), we can clearly see that the time series linear model has the lowest accuracy since it has an RMSE of 40.25 while the other three models have an RMSE ranging between 1.86 to 2.47. As stated before, the time series linear model does not take into account autocorrelation, which is further demonstrated in **Figure(3.11)** where ACF1 is marked as NA for the time series linear model. All in all, it seemed as if ARIMA was the way to go in terms of modeling this time series data since all of the measures of accuracy were around the same range for the ARIMA models while the time series linear model was always higher.

Figure(3.12) shows which of the ARIMA models served as the best fit for the data and assess the quality of each model. Ultimately it came down to AIC, AICc, and BIC values to determine which of the three ARIMA models was best. As it turns out, the naive model with seasonality activated was the best fit model for the data since its AIC, AICc, and BIC values were drastically lower than the other two models. In conclusion, our handpicked naive model

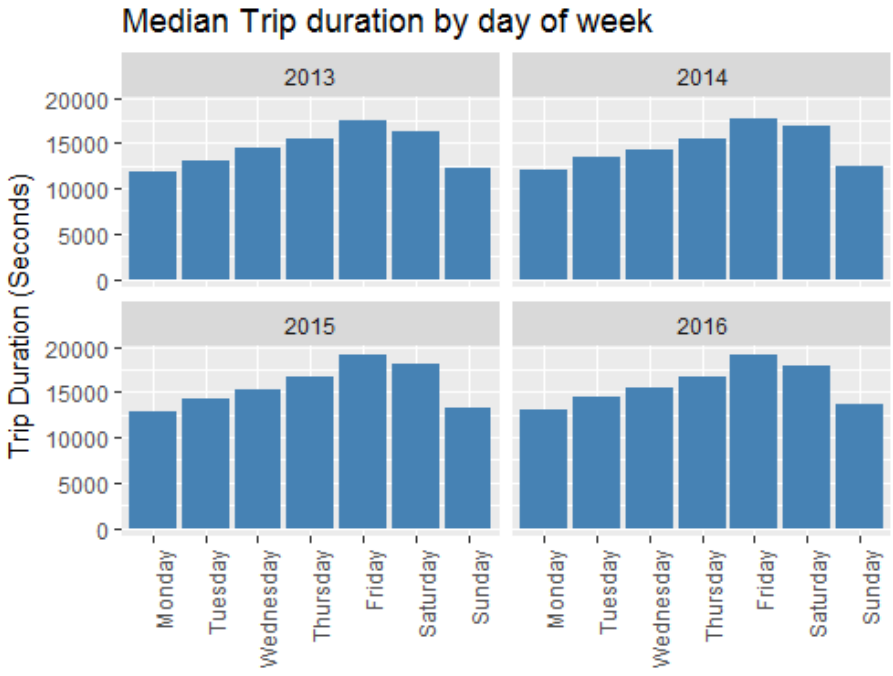
with seasonality out performed the model selected by auto.arima and is thus the best model for forecasting what the median daily fare in 2017 would look like for the average taxi in Chicago.

Appendix

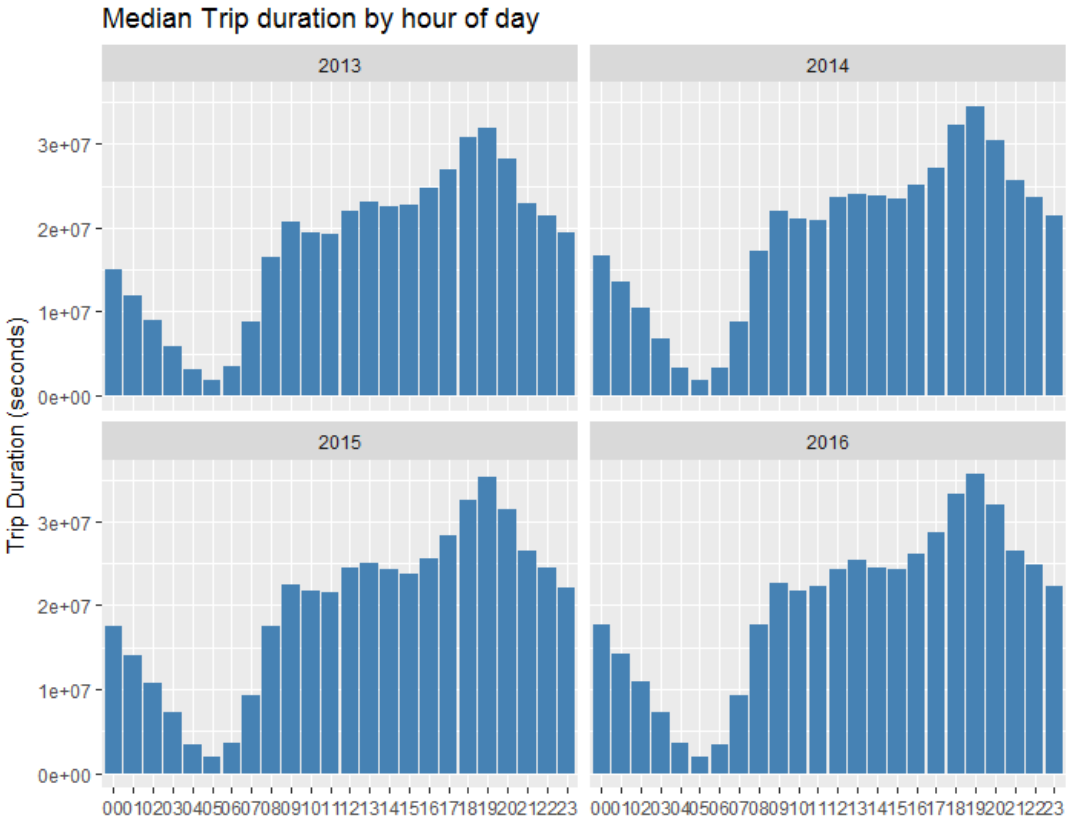
1.1



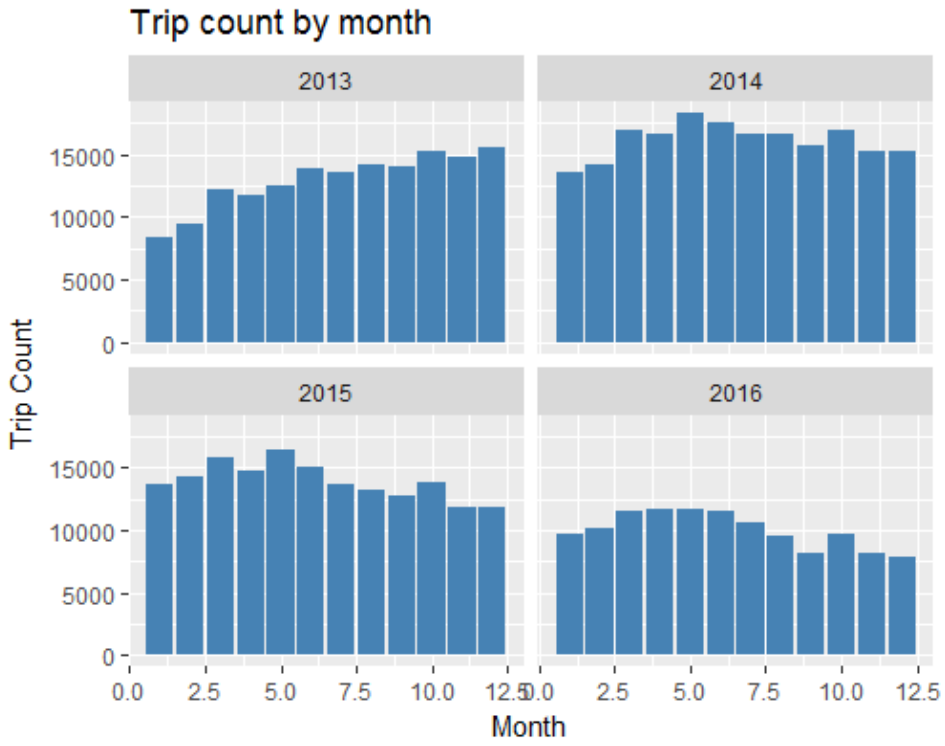
1.2



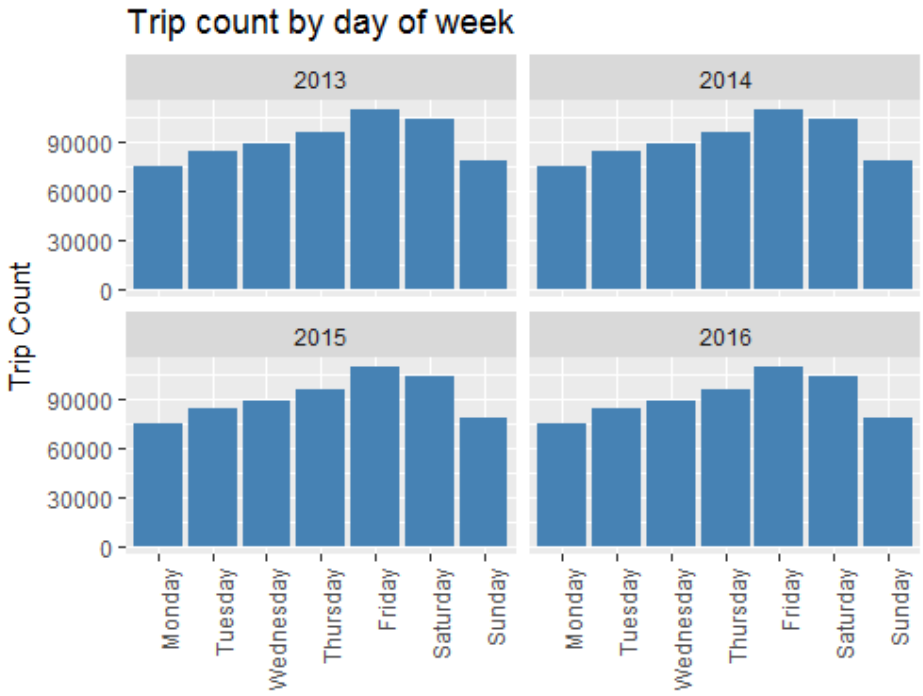
1.3


Median Trip duration by hour of day
2013
2014
2015
2016
Trip Duration (seconds)
3e+07
2e+07
1e+07
0e+00
000102030405060708091011121314151617181920212223

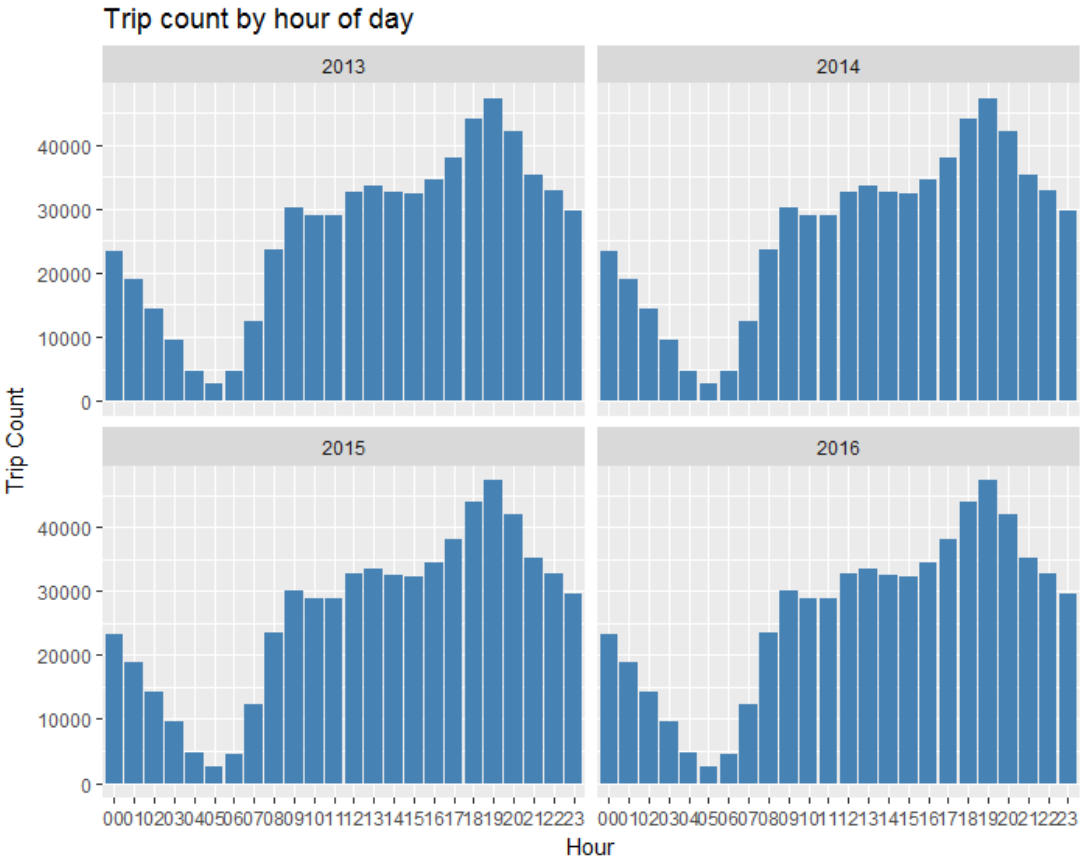
1.4


Trip count by month
2013
2014
2015
2016
Trip Count
15000
10000
5000
0
0.0
2.5
5.0
7.5
10.0
12.5
Month

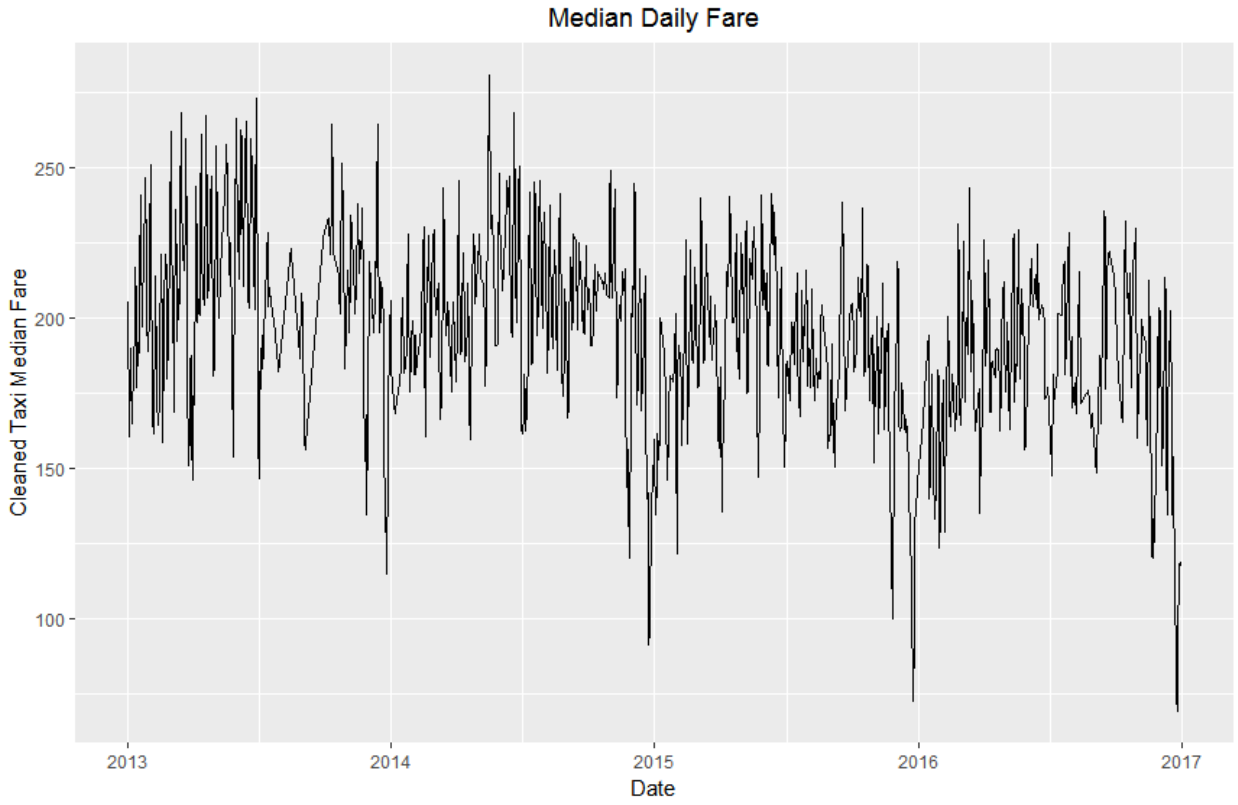
1.5


Trip count by day of week
2013
2014
2015
2016
Trip Count
90000
60000
30000
0
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

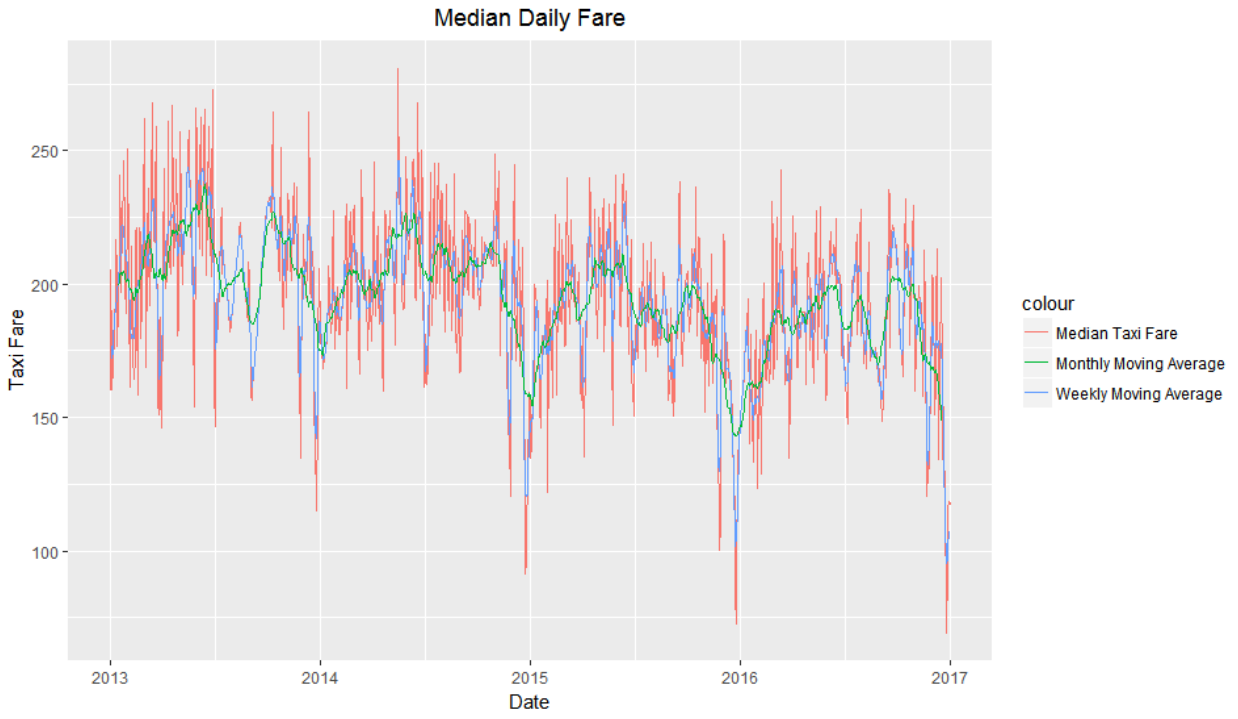
1.6


Trip count by hour of day
2013
2014
2015
2016
Trip Count
40000
30000
20000
10000
0
000102030405060708091011121314151617181920212223
Hour

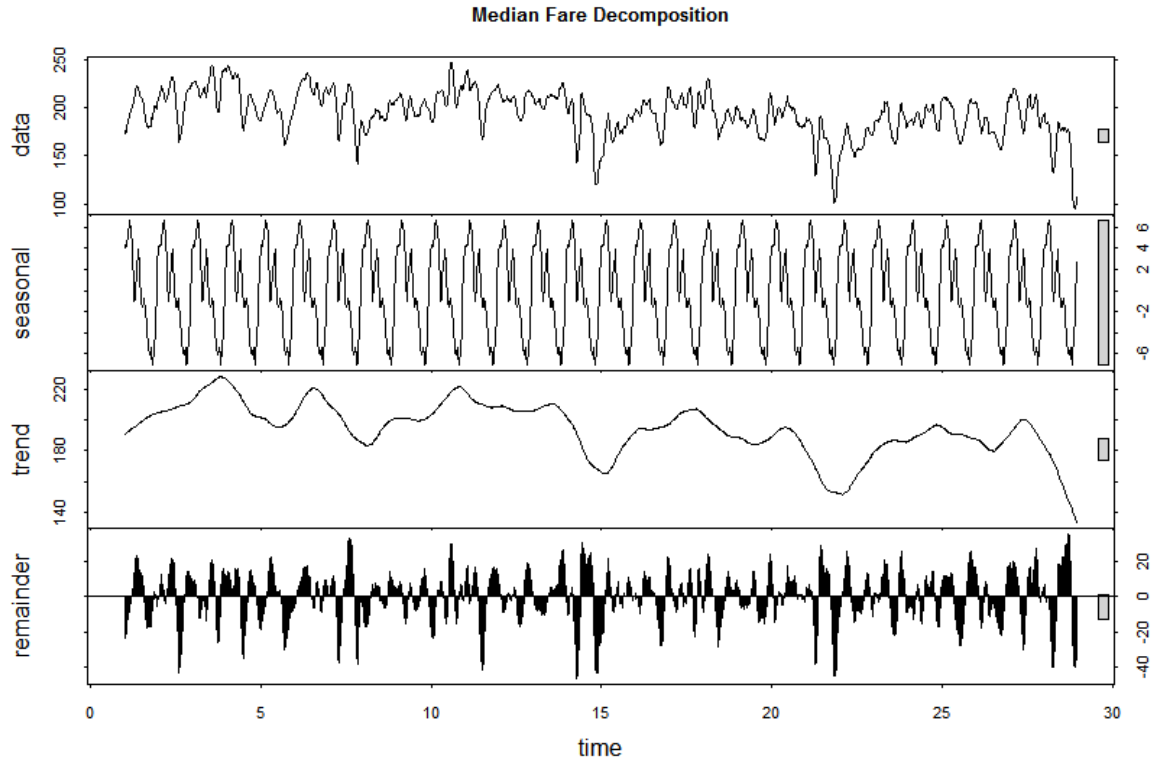
3.1


Median Daily Fare
Cleaned Taxi Median Fare
250
200
150
100
2013
2014
2015
2016
2017
Date

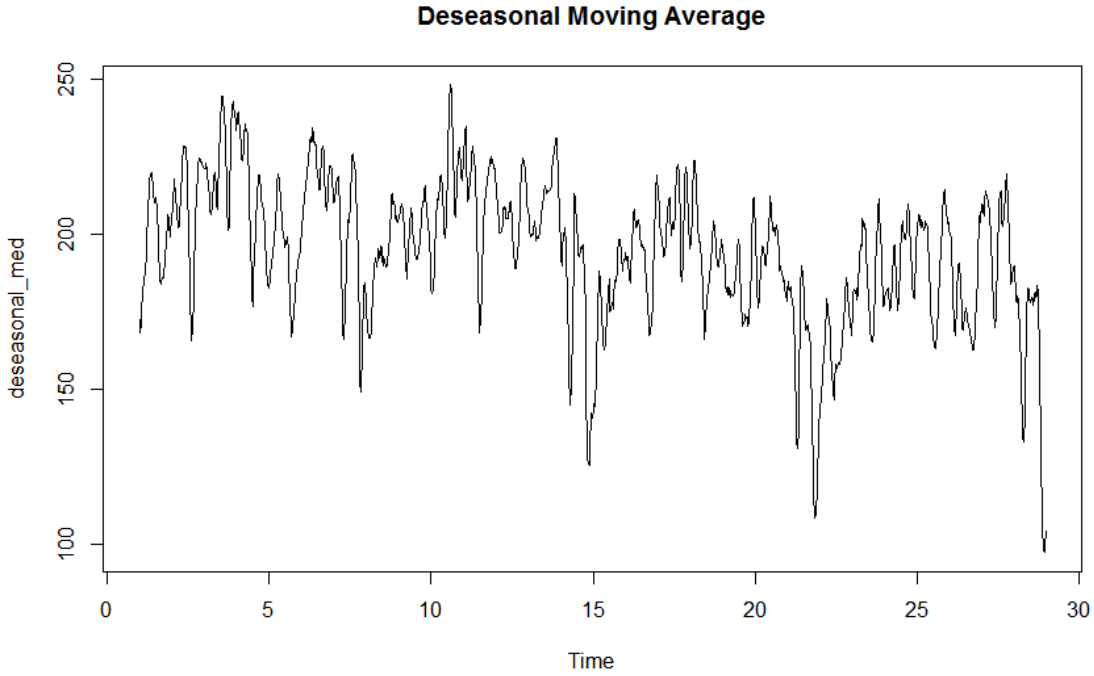
3.2


Median Daily Fare
Taxi Fare
250
200
150
100
2013
2014
2015
2016
2017
Date
colour
Median Taxi Fare
Monthly Moving Average
Weekly Moving Average

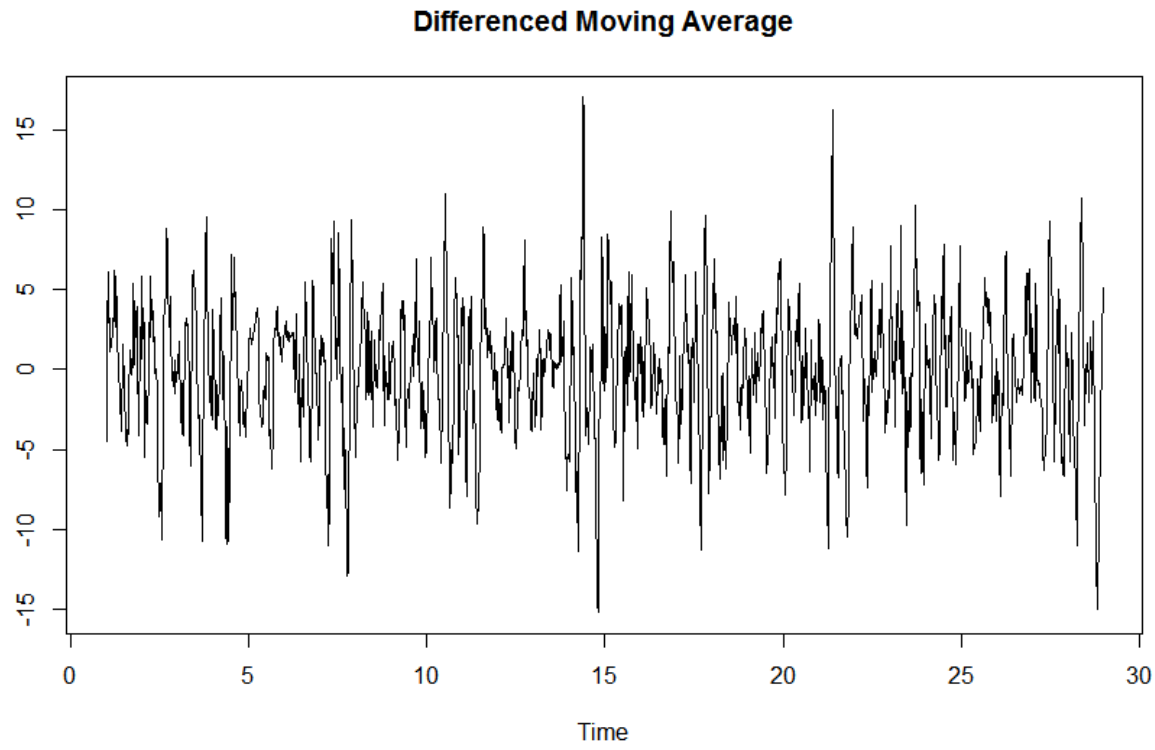
3.3

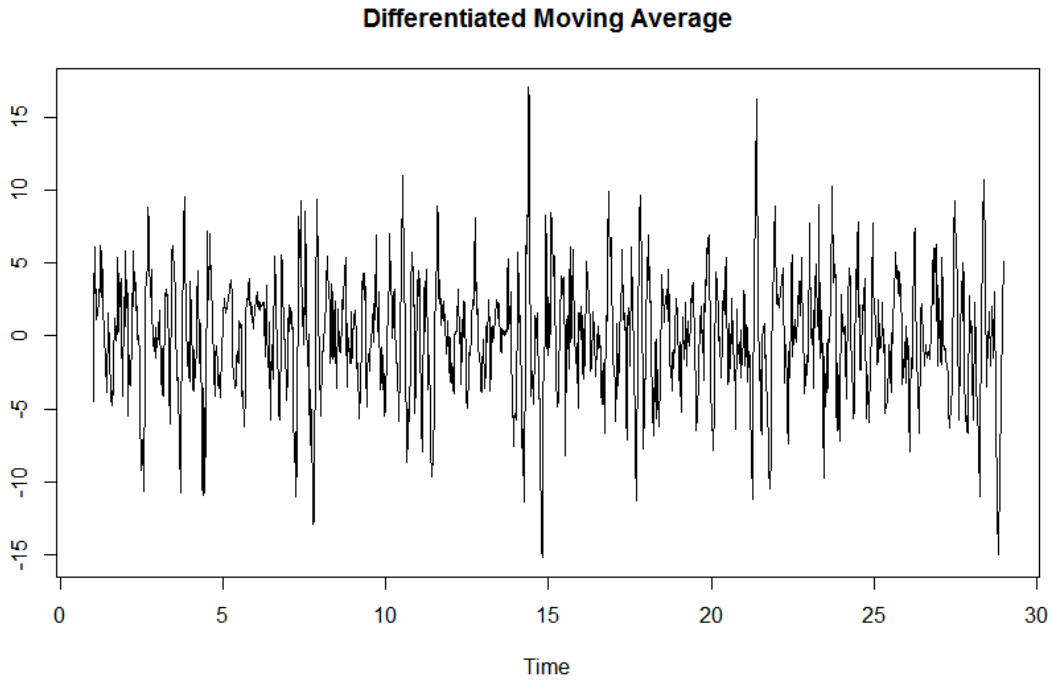

Median Fare Decomposition
data
seasonal
trend
remainder
time

3.4

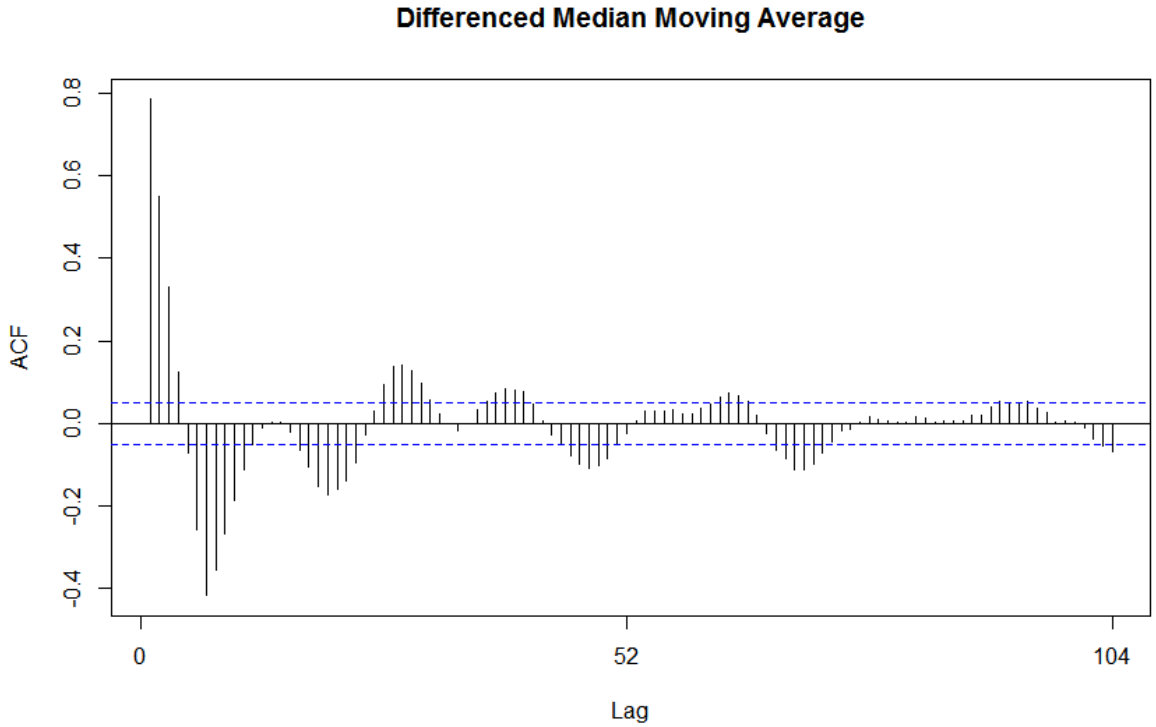

Deseasonal Moving Average
deseasonal_med
Time

3.5

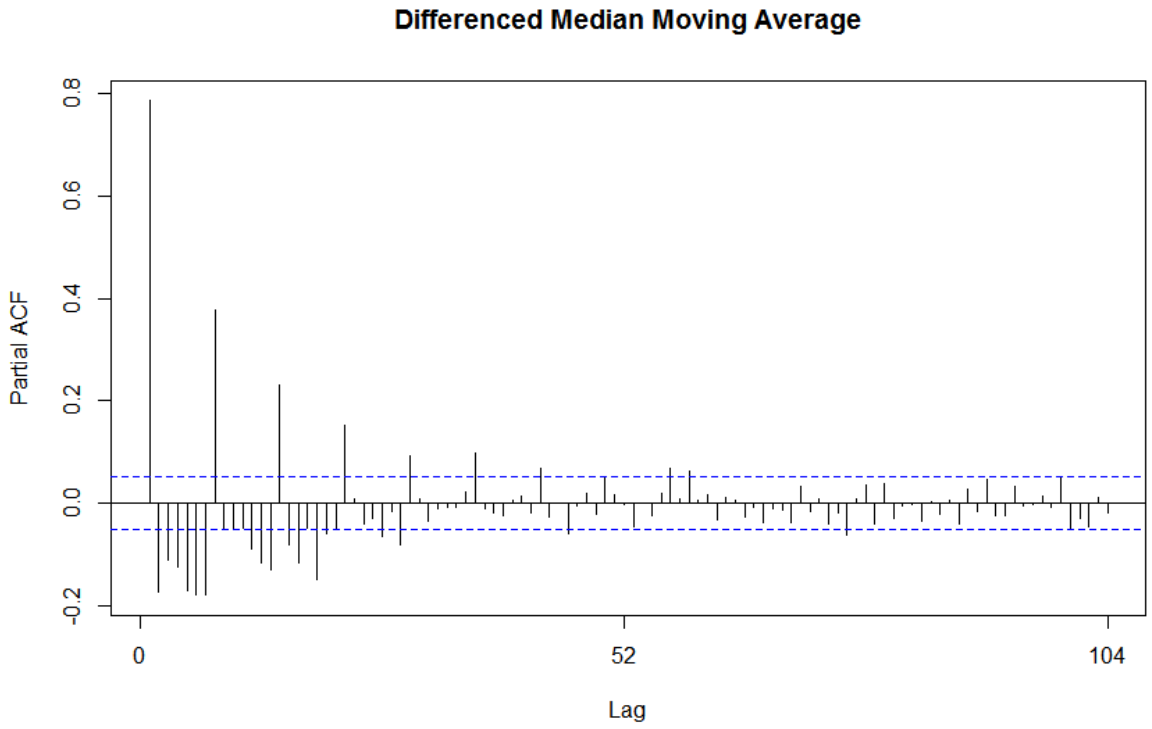

Differenced Moving Average
15
10
5
0
-5
-10
-15
0
5
10
15
20
25
30
Time


Differentiated Moving Average
15
10
5
0
-5
-10
-15
0
5
10
15
20
25
30
Time

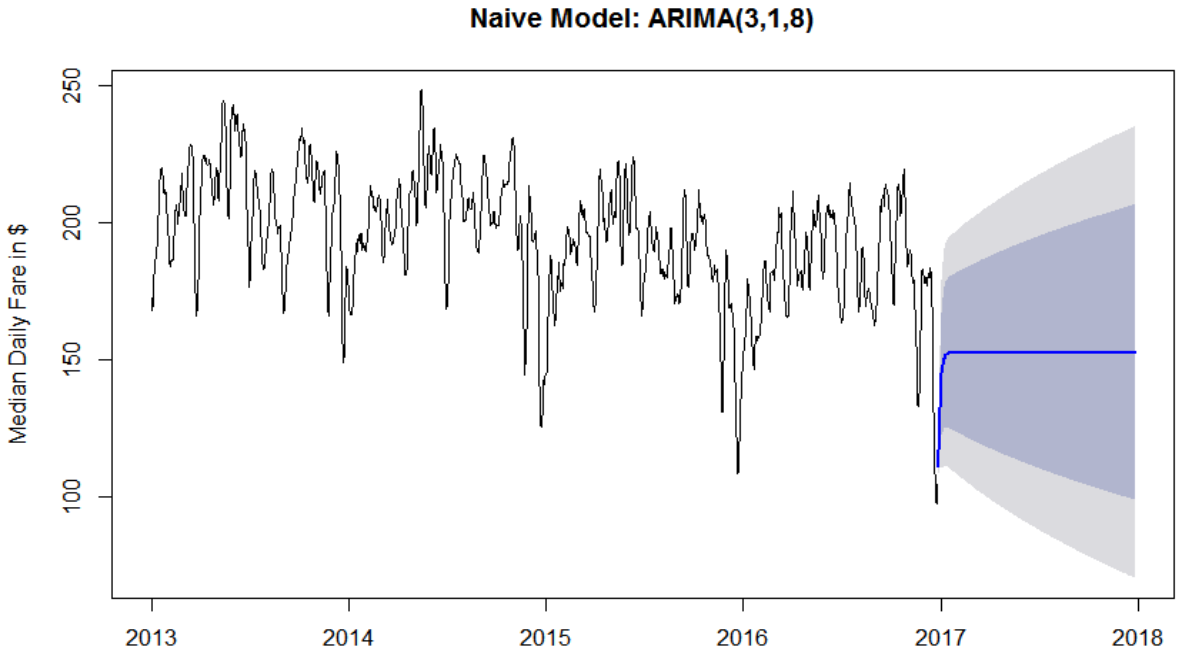
3.6


Differenced Median Moving Average
ACF
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
0
52
104
Lag

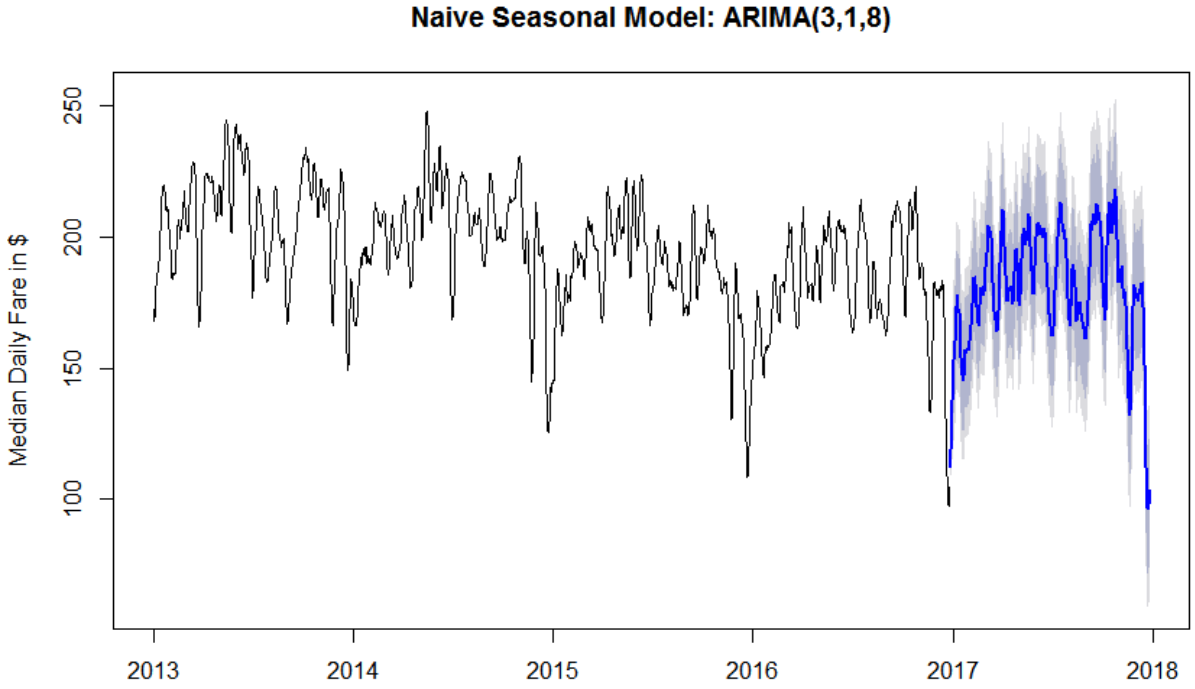
3.7


Differenced Median Moving Average
Partial ACF
0.8
0.6
0.4
0.2
0.0
-0.2
0
52
104
Lag

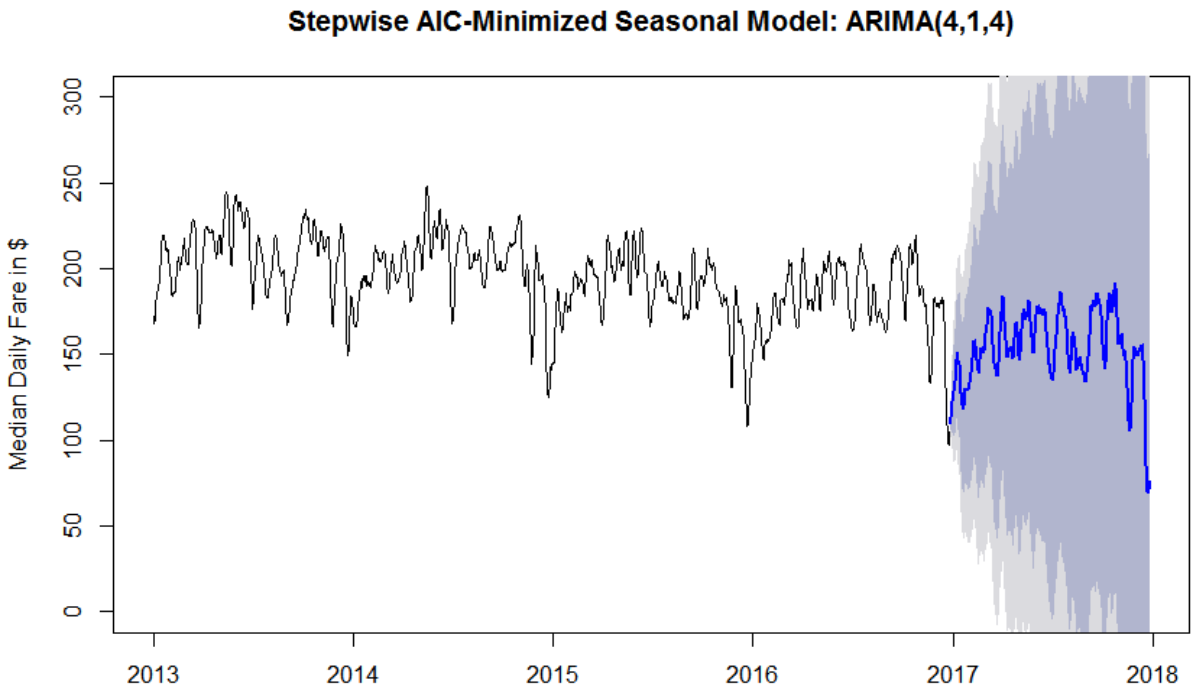
3.8


Naive Model: ARIMA(3,1,8)
Median Daily Fare in $
250
200
150
100
2013
2014
2015
2016
2017
2018

3.9


Naive Seasonal Model: ARIMA(3,1,8)
Median Daily Fare in $
250
200
150
100
2013
2014
2015
2016
2017
2018

3.10



3.11

Model Type:	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1
Naïve ARIMA	-0.0236982	1.863818	1.438127	-1.98E-02	0.755237	0.094616	-0.000498745
Naïve ARIMA Seasonal	0.0209041	2.119157	1.456505	6.35E-03	0.778575	0.095825	0.001427282
Auto-ARIMA	-0.0079869	2.465654	1.705533	9.73E-04	0.915273	0.112209	0.01645578
Time Series Linear Model	-0.3919451	40.25319	30.78456	-1.09E+01	18.27563	1.279084	NA

3.12

Model Type:	AIC	AICc	BIC
Naïve ARIMA	5980.22	5980.44	6043.61
Naïve ARIMA Seasonal	5082.77	5083.06	5142.69
Auto-ARIMA	5402.3	5402.47	5447.24

edaCode

```
library(data.table)
library(dplyr)
library(magrittr)
library(MASS)
library(ggplot2)
library(gridExtra)
library(kableExtra)

taxi_2013 <- fread('subset_2013.csv')
taxi_2014 <- fread('subset_2014.csv')
taxi_2015 <- fread('subset_2015.csv')
taxi_2016 <- fread('subset_2016.csv')
taxi_2017 <- fread('subset_2017.csv')

taxi_2013[, 12 := NULL]
names(taxi_2014) <- names(taxi_2013)
names(taxi_2015) <- names(taxi_2013)
names(taxi_2016) <- names(taxi_2013)
names(taxi_2017) <- names(taxi_2013)

taxi_df <- data.table(rbindlist(list(taxi_2013, taxi_2014, taxi_2015, taxi_2016)),
                      Year = rep(c(2013, 2014, 2015, 2016), times = c(nrow(taxi_2013), nrow(taxi_2014),

taxi_df <- na.omit(taxi_df)
taxi_2017 <- na.omit(taxi_2017)

date_time_2013 <- strptime(taxi_2013$`Trip Start Timestamp`, format = '%m/%d/%Y %I:%M:%S %p')
date_time_2014 <- strptime(taxi_2014$`Trip Start Timestamp`, format = '%m/%d/%Y %I:%M:%S %p')
date_time_2015 <- strptime(taxi_2015$`Trip Start Timestamp`, format = '%m/%d/%Y %I:%M:%S %p')
date_time_2016 <- strptime(taxi_2016$`Trip Start Timestamp`, format = '%m/%d/%Y %I:%M:%S %p')
date_time <- strptime(taxi_df$`Trip Start Timestamp`, format = '%m/%d/%Y %I:%M:%S %p')
date_time2 <- strptime(taxi_df$`Trip End Timestamp`, format = '%m/%d/%Y %I:%M:%S %p')

# Add month column
taxi_df[, Month := as.numeric(strftime(date_time, '%m'))]

g1 <- taxi_df[, .(TripDuration = `Trip Seconds`, Month), by = Year] %>%
  ggplot(aes(Month, TripDuration)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Duration (Seconds)', title = 'Trip duration by month')
g1

g2 <- taxi_df[, .(TotalTrip = sum(`Trip Seconds`)), by = list(`Taxi ID`, Month, Year)][, .(Median = medi
  ggplot(aes(Month, Median)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Duration (Seconds)', title = 'Median Trip duration by month')
g2
```

```
g3 <- taxi_df[, .(Month), by = Year][, .(TripCount = .N), by = c('Year', 'Month')] %>%
  ggplot(aes(Month, TripCount)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = 'Month', y = 'Trip Count', title = 'Trip count by month')
g3

g4 <- taxi_df[, .(TripDistance = `Trip Miles`, Month), by = Year] %>%
  ggplot(aes(Month, TripDistance)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = 'Month', y = 'Distance', title = 'Distance travelled by month')
g4

g5 <- taxi_df[, .(TotalTrip = sum(`Trip Miles`)), by = list(`Taxi ID`, Month, Year)][, .(Median = median
  ggplot(aes(Month, Median)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Distance (Miles)', title = 'Median Trip distance by month')
g5

g6 <- taxi_df[, .(TripDuration = `Trip Seconds`,
                  Day = factor(weekdays(as.Date(date_time)), levels = c('Monday', 'Tuesday', 'Wednesday
  ggplot(aes(Day, TripDuration)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Duration (seconds)', title = 'Trip duration by day of week') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
g6

g7 <- taxi_df[, .(TaxiID = `Taxi ID`,
                  TripDuration = `Trip Seconds`,
                  Day = factor(weekdays(as.Date(date_time)), levels = c('Monday', 'Tuesday', 'Wednesday
  ggplot(aes(Day, Median)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Duration (Seconds)', title = 'Median Trip duration by day of week ') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
g7

g8 <- taxi_df[, .(Day = factor(weekdays(as.Date(date_time)), levels = c('Monday', 'Tuesday', 'Wednesday
  ggplot(aes(Day, TripCount)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Count', title = 'Trip count by day of week') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
g8

g9 <- taxi_df[, .(TripDistance = `Trip Miles`,
                  Day = factor(weekdays(as.Date(date_time)), levels = c('Monday', 'Tuesday', 'Wednesday
  ggplot(aes(Day, TripDistance)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
```

```
  labs(x = '', y = 'Trip Distance (Miles)', title = 'Trip distance by day of week') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
g9

g10 <- taxi_df[, .(TaxiID = `Taxi ID`,
                   TripDistance = `Trip Miles`,
                   Day = factor(weekdays(as.Date(date_time)), levels = c('Monday', 'Tuesday', 'Wednesday
  ggplot(aes(Day, Median)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Distance (Miles)', title = 'Median Trip distance by day of week') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
g10

g11 <- taxi_df[, .(TripDuration = `Trip Seconds`, Hour = strftime(date_time, '%H')), by = Year] %>%
  ggplot(aes(Hour, TripDuration)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = 'Hour', y = 'Trip Duration (seconds)', title = 'Trip duration by hour of day')
g11

g12 <- taxi_df[, .(TripDuration = `Trip Seconds`,
                    Hour = strftime(date_time, '%H')), by = Year] %>%
  ggplot(aes(Hour, TripDuration)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Duration (seconds)', title = 'Median Trip duration by hour of day')
g12

g13 <- taxi_df[, .(Hour = strftime(date_time, '%H')), by = Year][, .(TripCount = .N), by = c('Year', 'H
  ggplot(aes(Hour, TripCount)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = 'Hour', y = 'Trip Count', title = 'Trip count by hour of day')
g13

g14 <- taxi_df[, .(TripDistance = `Trip Miles`, Hour = strftime(date_time, '%H')), by = Year] %>%
  ggplot(aes(Hour, TripDistance)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = 'Hour', y = 'Trip Distance (Miles)', title = 'Trip distance by hour of day')
g14

g15 <- taxi_df[, .(TaxiID = `Taxi ID`,
                   TripDistance = `Trip Miles`,
                   Hour = strftime(date_time, '%H')), by = Year][, .(TotalTrip = sum(TripDistance)), by =
  ggplot(aes(Hour, Median)) +
  geom_bar(stat = 'identity', fill = 'steelblue') +
  facet_wrap(~ as.factor(Year)) +
  labs(x = '', y = 'Trip Distance (Miles)', title = 'Median Trip distance by hour of day')
g15
```

Random Forest

April 20, 2018

1 Random Forest Pickups

1.1 Importing libraries

```
In [1]: import os
        os.chdir('/Users/Kevin/Documents/TAMIDS Taxi Data')

        import numpy as np
        import pandas as pd
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import mean_squared_error
        from sklearn.metrics import accuracy_score
        import random
        import math
        import datetime
        import time
```

1.2 Data

```
In [2]: x_train = pd.read_csv('PickupData.csv')
        x_test = pd.read_csv('PickupDataTest.csv')
```

A preview of our data.

```
In [3]: x_train.head()
```

```
Out[3]:    Year_num  Month  Hour  Day  WDay   Latitude  Longitude  Pickups  Month_num  \
        0  0.202740      3    17   15     5  41.879255 -87.642649        2   0.466667
        1  0.106849      2    19    8     5  41.893216 -87.637844        1   0.233333
        2  0.339726      5    20    4     6  41.916005 -87.675095        3   0.100000
        3  0.695890      9    12   11     3  41.884987 -87.620993        4   0.333333
        4  0.953425     12    21   14     6  41.906026 -87.675312        2   0.433333

           Hour_num   Day_num  Year_sin  Year_cos  Month_sin  Month_cos  Hour_sin  \
        0  0.750000  0.678571  0.956235  0.292600   0.207912  -0.978148 -1.000000
        1  0.833333  0.690476  0.622047  0.782980   0.994522   0.104528 -0.866025
```

```
2  0.875000  0.839286  0.845249 -0.534373   0.587785   0.809017 -0.707107
3  0.541667  0.363095 -0.942761 -0.333469   0.866025  -0.500000 -0.258819
4  0.916667  0.845238 -0.288482  0.957485   0.406737  -0.913545 -0.500000

       Hour_cos   Day_sin   Day_cos
0 -1.836970e-16 -0.900969 -0.433884
1  5.000000e-01 -0.930874 -0.365341
2  7.071068e-01 -0.846724  0.532032
3 -9.659258e-01  0.757972 -0.652287
4  8.660254e-01 -0.826239  0.563320
```

1.2.1 Defining Time Variables

The time granularity with respect to the granularity of location will be defined as bins per day (hours per day). We must adjust our timestamps accordingly. Note that our data was processed in R. * **Hour_num**: Using a 24-hour clock, the hours will be defined from 1-24 (1:00am is 1, 12:00am is 24). Using the **strptime()** function, the hours are defined from 0-23, so we just add 1 to correct this. The formula for this calculation is:

$$HourNum = (Hour + 1)/24 \tag{1}$$

- **Day_num**: The day of the week will be defined from 0-6 (Monday is 0, Sunday is 6), and we will incorporate hour of day in this calculation also. The formula for this calculation is:

$$DayNum = (((WDay - 1) \mod 7) + (Hour + 1)/24)/7 \tag{2}$$

  WDay is the day of the week. Example: Friday, 2:00pm (Day = 5, Hour = 14)

$$DayNum = ((5 \mod 7) + (14 + 1)/24)/7 \tag{3}$$

  We do **WDay**-1 to count all of the days of the week previous to Friday. Since our range of integers for week is defined on the interval [0,6] we will be doing calculations for day of the week on mod 7. Then finally we divide by 7, the number of days in a week.
- **Month_num**: The day of the month.

$$MonthNum = (Day - 1)/30 \tag{4}$$

- **Year_num**: The day of the year will defined on a scale of 1-365. The **strptime()** function defines the day of the year from 0-364, so like hour, we add an correction factor of 1. The formula for this calculation is:

$$YearNum = (YearDay + 1)/365 \tag{5}$$

Note: For 2016, we used 366 to count since it is a leap year.

- **Sine** and **Cosine**: We capture cyclicality by applying sine and cosine functions to our calculated time, month, day, and year numbers.

1.2.2 Create response variables

```
In [4]: y_pickup_train = np.log10(x_train['Pickups']+1)
        y_pickup_test = np.log10(x_test['Pickups']+1)
```

```
In [5]: # Remove pickup and wday column from training and testing data
        pickup_train = x_train.drop(['Pickups', 'WDay'], axis=1)
        pickup_test = x_test.drop(['Pickups', 'WDay'], axis=1)
```

1.3 Model Building

1.3.1 Optimizing the random forest hyper-parameters

Before we run our model, we want to find the best parameters to run the model with, in particular, we want to optimize the number of trees to use, the depth of the trees, and the number of variables considered at each split. We will use a 5-fold cross validation function to find the best parameters.

```
In [6]: def optimize_params(clf, params, X, y):
            cv = GridSearchCV(clf, param_grid=params, n_jobs=1, cv=5, verbose=3)
            cv.fit(X, y)
            print('Best parameters:', cv.best_params_)
            print('Best score:', cv.best_score_)
            print('Best estimators:', cv.best_estimator_)
```

Defining the range of parameters Note that **n_estimators** is the number of trees, **max_features** is the number of variables considered at each split, and **max_depth** is the depth of the trees.

```
In [91]: rf_estimator = RandomForestRegressor(n_estimators=20, n_jobs=-1)

         params = {'n_estimators': [1, 2, 5, 10, 30, 50, 100],
                   'max_features': ['auto', 'log2'],
                   'max_depth': [10, 20, 50, 100]
                  }

         start_time = time.time()
         best_params = optimize_params(rf_estimator, params, pickup_train, y_pickup_train)
         time.time() - start_time
```

```
Fitting 5 folds for each of 56 candidates, totalling 280 fits
[CV] max_depth=10, max_features=auto, n_estimators=1 ...
[CV]  max_depth=10, max_features=auto, n_estimators=1, score=0.559440, total=   2.2s
[CV] max_depth=10, max_features=auto, n_estimators=1 ...


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.3s remaining:    0.0s


[CV]  max_depth=10, max_features=auto, n_estimators=1, score=0.612251, total=   2.2s
[CV] max_depth=10, max_features=auto, n_estimators=1 ...
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    4.6s remaining:    0.0s


[CV]  max_depth=10, max_features=auto, n_estimators=1, score=0.615317, total=   2.1s
[CV] max_depth=10, max_features=auto, n_estimators=1 ...
[CV]  max_depth=10, max_features=auto, n_estimators=1, score=0.611700, total=   2.1s
[CV] max_depth=10, max_features=auto, n_estimators=1 ...
[CV]  max_depth=10, max_features=auto, n_estimators=1, score=0.558787, total=   2.1s
[CV] max_depth=10, max_features=auto, n_estimators=2 ...
[CV]  max_depth=10, max_features=auto, n_estimators=2, score=0.565525, total=   2.6s
[CV] max_depth=10, max_features=auto, n_estimators=2 ...
[CV]  max_depth=10, max_features=auto, n_estimators=2, score=0.616767, total=   2.6s
[CV] max_depth=10, max_features=auto, n_estimators=2 ...
[CV]  max_depth=10, max_features=auto, n_estimators=2, score=0.620318, total=   2.6s
[CV] max_depth=10, max_features=auto, n_estimators=2 ...
[CV]  max_depth=10, max_features=auto, n_estimators=2, score=0.616743, total=   2.6s
[CV] max_depth=10, max_features=auto, n_estimators=2 ...
[CV]  max_depth=10, max_features=auto, n_estimators=2, score=0.565599, total=   2.5s
[CV] max_depth=10, max_features=auto, n_estimators=5 ...
[CV]  max_depth=10, max_features=auto, n_estimators=5, score=0.568239, total=   3.9s
[CV] max_depth=10, max_features=auto, n_estimators=5 ...
[CV]  max_depth=10, max_features=auto, n_estimators=5, score=0.619466, total=   3.9s
[CV] max_depth=10, max_features=auto, n_estimators=5 ...
[CV]  max_depth=10, max_features=auto, n_estimators=5, score=0.622651, total=   3.9s
[CV] max_depth=10, max_features=auto, n_estimators=5 ...
[CV]  max_depth=10, max_features=auto, n_estimators=5, score=0.618336, total=   4.0s
[CV] max_depth=10, max_features=auto, n_estimators=5 ...
[CV]  max_depth=10, max_features=auto, n_estimators=5, score=0.569112, total=   3.9s
[CV] max_depth=10, max_features=auto, n_estimators=10 ...
[CV]  max_depth=10, max_features=auto, n_estimators=10, score=0.568881, total=   8.2s
[CV] max_depth=10, max_features=auto, n_estimators=10 ...
[CV]  max_depth=10, max_features=auto, n_estimators=10, score=0.620670, total=   8.2s
[CV] max_depth=10, max_features=auto, n_estimators=10 ...
[CV]  max_depth=10, max_features=auto, n_estimators=10, score=0.623815, total=   8.2s
[CV] max_depth=10, max_features=auto, n_estimators=10 ...
[CV]  max_depth=10, max_features=auto, n_estimators=10, score=0.619361, total=   8.2s
[CV] max_depth=10, max_features=auto, n_estimators=10 ...
[CV]  max_depth=10, max_features=auto, n_estimators=10, score=0.570462, total=   8.2s
[CV] max_depth=10, max_features=auto, n_estimators=30 ...
[CV]  max_depth=10, max_features=auto, n_estimators=30, score=0.569266, total=  21.4s
[CV] max_depth=10, max_features=auto, n_estimators=30 ...
[CV]  max_depth=10, max_features=auto, n_estimators=30, score=0.621259, total=  21.6s
[CV] max_depth=10, max_features=auto, n_estimators=30 ...
[CV]  max_depth=10, max_features=auto, n_estimators=30, score=0.624219, total=  21.5s
[CV] max_depth=10, max_features=auto, n_estimators=30 ...
[CV]  max_depth=10, max_features=auto, n_estimators=30, score=0.620186, total=  21.5s
[CV] max_depth=10, max_features=auto, n_estimators=30 ...
[CV]  max_depth=10, max_features=auto, n_estimators=30, score=0.570721, total=  21.7s
```

```
[CV] max_depth=100, max_features=log2, n_estimators=30 ...
[CV]  max_depth=100, max_features=log2, n_estimators=30, score=0.646503, total=  11.9s
[CV] max_depth=100, max_features=log2, n_estimators=30 ...
[CV]  max_depth=100, max_features=log2, n_estimators=30, score=0.839244, total=   9.6s
[CV] max_depth=100, max_features=log2, n_estimators=30 ...
[CV]  max_depth=100, max_features=log2, n_estimators=30, score=0.853031, total=  10.0s
[CV] max_depth=100, max_features=log2, n_estimators=30 ...
[CV]  max_depth=100, max_features=log2, n_estimators=30, score=0.719004, total=  12.4s
[CV] max_depth=100, max_features=log2, n_estimators=30 ...
[CV]  max_depth=100, max_features=log2, n_estimators=30, score=0.507412, total=  10.4s
[CV] max_depth=100, max_features=log2, n_estimators=50 ...
[CV]  max_depth=100, max_features=log2, n_estimators=50, score=0.651389, total=  14.3s
[CV] max_depth=100, max_features=log2, n_estimators=50 ...
[CV]  max_depth=100, max_features=log2, n_estimators=50, score=0.842260, total=  16.1s
[CV] max_depth=100, max_features=log2, n_estimators=50 ...
[CV]  max_depth=100, max_features=log2, n_estimators=50, score=0.856257, total=  16.0s
[CV] max_depth=100, max_features=log2, n_estimators=50 ...
[CV]  max_depth=100, max_features=log2, n_estimators=50, score=0.721911, total=  15.8s
[CV] max_depth=100, max_features=log2, n_estimators=50 ...
[CV]  max_depth=100, max_features=log2, n_estimators=50, score=0.511711, total=  15.8s
[CV] max_depth=100, max_features=log2, n_estimators=100 ...
[CV]  max_depth=100, max_features=log2, n_estimators=100, score=0.653762, total=  31.4s
[CV] max_depth=100, max_features=log2, n_estimators=100 ...
[CV]  max_depth=100, max_features=log2, n_estimators=100, score=0.844691, total=  31.7s
[CV] max_depth=100, max_features=log2, n_estimators=100 ...
[CV]  max_depth=100, max_features=log2, n_estimators=100, score=0.857889, total=  33.4s
[CV] max_depth=100, max_features=log2, n_estimators=100 ...
[CV]  max_depth=100, max_features=log2, n_estimators=100, score=0.724079, total=  32.1s
[CV] max_depth=100, max_features=log2, n_estimators=100 ...
[CV]  max_depth=100, max_features=log2, n_estimators=100, score=0.514242, total=  31.3s


[Parallel(n_jobs=1)]: Done 280 out of 280 | elapsed: 558.9min finished


Best parameters: {'max_depth': 50, 'max_features': 'log2', 'n_estimators': 100}
Best score: 0.719075403196
Best estimators: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=50,
           max_features='log2', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=100, n_jobs=-1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)


Out[91]: 33574.71210980415
```

After performing cross validation, here are the results we found: * **max_depth** = 50 * **max_features** = log2 * **n_estimators** = 100

1.3.2 Fitting the model

We will use the parameters we found in the previous to fit our random forest model.

```
In [6]: reg = RandomForestRegressor(n_estimators=100, max_depth=50, max_features='log2', n_jobs
```

```
In [7]: reg.fit(pickup_train, y_pickup_train)
```

```
Out[7]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=50,
                    max_features='log2', max_leaf_nodes=None,
                    min_impurity_split=1e-07, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=100, n_jobs=1, oob_score=False, random_state=None,
                    verbose=0, warm_start=False)
```

1.3.3 Make predictions and evaluating results

```
In [8]: predicted_pickups = reg.predict(pickup_test)
        accuracy_score(np.round(np.power(10, y_pickup_test)-1, decimals=0),
                       np.round(np.power(10, predicted_pickups)-1, decimals=0))
```

```
Out[8]: 0.43662059133453168
```

Calculate RMSE

```
In [10]: rmse = np.sqrt(mean_squared_error(predicted_pickups, y_pickup_test))
         rmse_back = np.power(10, rmse)
         print("RMSE (log-space) = %0.3f" %rmse)
         print("RMSE = %0.3f" %rmse_back)
```

```
RMSE (log-space) = 0.141
RMSE = 1.382
```

Most important features

```
In [11]: import operator
         varImp = dict(zip(list(pickup_train.columns.values), reg.feature_importances_))
         varImp_sorted = sorted(varImp.items(), key=operator.itemgetter(1), reverse=True)
         varImp_sorted
```

```
Out[11]: [('Latitude', 0.27621509508875908),
          ('Longitude', 0.23507218037847466),
          ('Year_num', 0.050127152421569302),
          ('Year_cos', 0.050009352118430349),
          ('Year_sin', 0.049514385038255677),
          ('Day_num', 0.049068821142499879),
          ('Day_cos', 0.0310088031001606),
          ('Month_num', 0.029413920719945082),
          ('Day', 0.029375558051922376),
```

```
('Month_cos', 0.028045684698258282),
('Hour_num', 0.02783126044194419),
('Day_sin', 0.027064419463383539),
('Month_sin', 0.026630089073298717),
('Hour', 0.025994081524676652),
('Hour_sin', 0.025958827259884246),
('Month', 0.019470573634985684),
('Hour_cos', 0.019199795843551602)]
```

Write predicted values to csv file We will take our predicted pickups and output these values to a csv to create visualizations. We will be using Tableau to visualize our predicted values.

```
In [12]: predicted_df = pd.DataFrame({'Latitude': pickup_test['Latitude'],
                                      'Longitude': pickup_test['Longitude'],
                                      'Year': np.repeat(2017, pickup_test.shape[0]),
                                      'Day': pickup_test['Day'],
                                      'Hour': pickup_test['Hour'],
                                      'Month': pickup_test['Month'],
                                      'WDay': x_test['WDay'],
                                      'YDay': np.round(pickup_test['Year_num']*365),
                                      'Pickups': np.round(np.power(10, predicted_pickups)-1, de
         predicted_df.to_csv('PredictedPickups.csv', index=False)
```

TimeSeries

```
## Set Working Directory
setwd("C:/Users/Jose/Documents/Comp_Data/Taxi_Data/ARIMA Data")
library(data.table)
## Import Data (From Subset 1)
Count_Data <- read.csv("ARIMACount.csv")
Fare_Data <- read.csv("ARIMAFare.csv")
Total_Data <- read.csv("ARIMATotal.csv")
Full_Data <- fread("FullTaxi.csv")
## Change Data type to Date

Count_Data <- Count_Data[-1465,]
Count_Data$Date <- as.Date(Count_Data$Date)
Fare_Data$Date <- as.Date(Fare_Data$Date)
Total_Data$Date <- as.Date(Total_Data$Date)




########## Time Series: Count Data ##########
ggplot(Count_Data, aes(Date, Count)) + geom_line() + scale_x_date("Year")  + ylab("Taxi Rides") +
  xlab("") + ggtitle("Taxi Count") + theme(plot.title = element_text(hjust = 0.5))

count_ts <- ts(Count_Data[,c('Count')])
Count_Data$clean_count <- tsclean(count_ts)

ggplot() +
  geom_line(data = Count_Data, aes(x = Date, y = clean_count)) +  scale_x_date('Year') + ylab('Taxi Rid

## Calculates Moving averages for Month and Week
Count_Data$count_ma = ma(Count_Data$clean_count, order = 7) # using the clean count with no outliers
Count_Data$count_ma30 = ma(Count_Data$clean_count, order = 30)

## Plots Counts with Moving AVerage for Week and Month
ggplot() +
  geom_line(data = Count_Data, aes(x = Date, y = clean_count, colour = "Counts")) +
  geom_line(data = Count_Data, aes(x = Date, y = count_ma,   colour = "Weekly Moving Average"))  +
  geom_line(data = Count_Data, aes(x = Date, y = count_ma30, colour = "Monthly Moving Average"))  +
  ylab('Taxi Rides')+ ggtitle("Taxi Count Moving Average") + theme(plot.title = element_text(hjust = 0.5

## Calculates Seasonal components using stl
count_ma <- ts(na.omit(Count_Data$count_ma), frequency=30)
decomp1 <- stl(count_ma, s.window="periodic")

## seasadj() removes the seasonality by subtracting the seasonal component from the original series
deseasonal_count <- seasadj(decomp1)
plot(decomp1, main = "Taxi Count Trends")

Arima_Count <- Arima(ts(deseasonal_count, frequency = 365), order = c(1,1,1), seasonal = c(0,1,0))
Count_Forecast <- forecast(Arima_Count, h = 365)
plot(Count_Forecast)
```

```
#
# ## ADF tests for stationarity. Ho: Series is non-stationary
# adf.test(count_ma, alternative = "stationary")  ## Non-stationary because of large p-val
#
# ## Autocorrelation Plot
# Acf(count_ma, main='')
# ## PArtial Autocorrelation Plot
# Pacf(count_ma, main='')
#
# ## Differs by 1
# count_d1 = diff(deseasonal_count, differences = 1)
# plot(count_d1)
#
# adf.test(count_d1, alternative = "stationary")
#
# ## ACF and PACF for Differenced Series
# Acf(count_d1, main='ACF for Differenced Series')  #Sig at 1, 2 and beyond
# Pacf(count_d1, main='PACF for Differenced Series')
#
# tsdisplay(residuals(fit), lag.max=45, main='(3,1,4) Model Residuals')
#
# fit2 = arima(deseasonal_count, order=c(3,1,7))
# fit2
#
# tsdisplay(residuals(fit2), lag.max=15, main="Seasonal Model Residuals")
#
# fcast <- forecast(fit2, h=30)
# plot(fcast, xlab = "Months", ylab = "Taxi Count")
#
# ## "Starts" data earlier on
# hold <- window(ts(deseasonal_count), start=1400)
# ## Segments part of data to forecast
# fit_no_holdout = arima(ts(deseasonal_count[-c(1400:1455)]), order=c(3,1,7))
# ## Forecasts
# fcast_no_holdout <- forecast(fit_no_holdout,h=55)
# ## Plots
# plot(fcast_no_holdout, main=" ")
# lines(ts(deseasonal_count))
#
# fit_w_seasonality = auto.arima(deseasonal_count, seasonal=TRUE)
# fit_w_seasonality
#
# seas_fcast <- forecast(fit_w_seasonality, h=50)
# plot(seas_fcast)


########## Time Series: Median Fare ##########
ggplot(Fare_Data, aes(Date, Median)) + geom_line() + scale_x_date('Day')  + ylab("Median Taxi Fairs (Da
  xlab("") + ggtitle("Median Base Fare") + theme(plot.title = element_text(hjust = 0.5))

med_ts <- ts(Fare_Data[,c('Median')])
Fare_Data$clean_med <- tsclean(med_ts)
```

```r
ggplot() +
  geom_line(data = Fare_Data, aes(x = Date, y = clean_med)) + ylab('Cleaned Taxi Median Fair')

## Calculates Moving averages for Month and Week
Fare_Data$med_ma = ma(Fare_Data$clean_med, order = 7) # using the clean count with no outliers
Fare_Data$med_ma30 = ma(Fare_Data$clean_med, order = 30)

## Plots Counts with Moving AVerage for Week and Month
ggplot() +
  geom_line(data = Fare_Data, aes(x = Date, y = clean_med, colour = "Median Taxi Fare")) +
  geom_line(data = Fare_Data, aes(x = Date, y = med_ma,   colour = "Weekly Moving Average"))  +
  geom_line(data = Fare_Data, aes(x = Date, y = med_ma30, colour = "Monthly Moving Average"))  +
  ylab('Taxi Fare')

## Calculates Seasonal components using stl
med_ma <- ts(na.omit(Fare_Data$med_ma), frequency=30)
decomp2 <- stl(med_ma, s.window="periodic")

## seasadj() removes the seasonality by subtracting the seasonal component from the original series
deseasonal_med <- seasadj(decomp2)
plot(decomp2)

Arima_med <- Arima(ts(deseasonal_med, frequency = 365), order = c(1,1,2), seasonal = c(0,1,0))
Med_Forecast <- forecast(Arima_med, h = 365)
plot(Med_Forecast)
# ## ADF tests for stationarity. Ho: Series is non-stationary
# adf.test(med_ma, alternative = "stationary")  ## Non-stationary because of large p-val
#
# ## Autocorrelation Plot
# Acf(med_ma, main='')
# ## PArtial Autocorrelation Plot
# Pacf(med_ma, main='')
#
# ## Differs by 1
# med_d1 = diff(deseasonal_med, differences = 1)
# plot(med_d1)
#
# adf.test(med_d1, alternative = "stationary")
#
# ## ACF and PACF for Differenced Series
# Acf(med_d1, main='ACF for Differenced Series')  #Sig at 1, 2 and beyond
# Pacf(med_d1, main='PACF for Differenced Series')
#
# auto.arima(deseasonal_med, seasonal=FALSE)
# fit <- auto.arima(deseasonal_med, seasonal=FALSE)
#
# tsdisplay(residuals(fit), lag.max=45, main='(1,1,3) Model Residuals')
#
# fit2 = arima(deseasonal_med, order=c(1,1,3))
# fit2
#
# tsdisplay(residuals(fit2), lag.max=15, main="Seasonal Model Residuals")
#
```

```
# fcast <- forecast(fit2, h=30)
# plot(fcast, xlab = "Months", ylab = "Taxi Count")
#
# #
# # hold <- window(ts(deseasonal_cnt), start=1400)
# # fit_no_holdout = arima(ts(deseasonal_cnt[-c(1400:1455)]), order=c(1,1,3))
# # fcast_no_holdout <- forecast(fit_no_holdout,h=55)
# # plot(fcast_no_holdout, main=" ")
# # lines(ts(deseasonal_cnt))
# #
# # fit_w_seasonality_med = auto.arima(deseasonal_med, seasonal=TRUE)
# # fit_w_seasonality_med
# #
# # seas_fcast_med <- forecast(fit_w_seasonality_med, h=50)
# # plot(seas_fcast_med)
#
#
# #
# # fit_w_seasonality = auto.arima(deseasonal_cnt, seasonal=TRUE, D = 1)
# # fit_w_seasonality
#
# start_time <- Sys.time()
# plot(forecast(auto.arima(ts(deseasonal_med, frequency = 365), D = 1), h=365), main = "Taxi Median Far
# plot(forecast(auto.arima(ts(deseasonal_med, frequency = 365), D = 1, parallel = TRUE, stepwise = FALS
#
# end_time <- Sys.time()
# tot_time<- end_time - start_time



########## Time Series: Median Total Cost ##########
ggplot(Total_Data, aes(Date, Median)) + geom_line() + scale_x_date('Day')  + ylab("Median Taxi Total Cos
  xlab("") + ggtitle("Median Total Cost") + theme(plot.title = element_text(hjust = 0.5))

med_total_ts <- ts(Total_Data[,c('Median')])
Total_Data$clean_med_total <- tsclean(med_total_ts)

ggplot() +
  geom_line(data = Total_Data, aes(x = Date, y = clean_med_total)) + ylab('Cleaned Taxi Median Total Cos

## Calculates Moving averages for Month and Week
Total_Data$med_total_ma = ma(Total_Data$clean_med_total, order = 7) # using the clean count with no out
Total_Data$med_total_ma30 = ma(Total_Data$clean_med_total, order = 30)

## Plots Counts with Moving AVerage for Week and Month
ggplot() +
  geom_line(data = Total_Data, aes(x = Date, y = clean_med_total, colour = "Median Taxi Total Cost")) +
  geom_line(data = Total_Data, aes(x = Date, y = med_total_ma,   colour = "Weekly Moving Average"))  +
  geom_line(data = Total_Data, aes(x = Date, y = med_total_ma30, colour = "Monthly Moving Average"))  +
  ylab('Total Cost (In Dollars)')

## Calculates Seasonal components using stl
med_total_ma <- ts(na.omit(Total_Data$med_total_ma), frequency=30)
```

```r
decomp3 <- stl(med_total_ma, s.window="periodic")


## seasadj() removes the seasonality by subtracting the seasonal component from the original series
deseasonal_med_total <- seasadj(decomp3)
plot(decomp3)

Arima_Total <- Arima(ts(deseasonal_med_total, frequency = 365),order = c(5,1,2), seasonal = c(0,1,0))
Total_Forecast <- forecast(Arima_Total, h = 365)
plot(Total_Forecast)

Arima_Total_Adjusted <- Arima(ts(deseasonal_med_total, frequency = 365),order = c(1,1,2), seasonal = c(
Total_Adjusted_Forecast<- forecast(Arima_Total_Adjusted, h = 365)
plot(Total_Adjusted_Forecast)
```

ARIMA

```
setwd("C:/Users/Jose/Documents/Comp_Data/Taxi_Data/ARIMA Data")
library(data.table)
library(dplyr)
library(magrittr)
library(MASS)
library(ggplot2)
library(gridExtra)
library(randomForest)
library(mlr)
library(forecast)
library(tseries)

#taxi_2013 <- fread('Chicago_taxi_trips2013.csv')[,c(2,3,13)]
#taxi_2014 <- fread('Chicago_taxi_trips2014.csv')[,c(2,3,12)]
#taxi_2015 <- fread('Chicago_taxi_trips2015.csv')[,c(2,3,12)]
#taxi_2016 <- fread('Chicago_taxi_trips2016.csv')[,c(2,3,12)]
taxi_2017 <- fread('Chicago_taxi_trips2017.csv')[,c(2,3,12)]
taxi_2017COPY <- taxi_2017
#names(taxi_2014) <- names(taxi_2013)
#names(taxi_2015) <- names(taxi_2013)
#names(taxi_2016) <- names(taxi_2013)

#names(taxi_2017) <- names(taxi_df[,1:3])
#taxi_df <- data.table(rbindlist(list(taxi_2013, taxi_2014, taxi_2015, taxi_2016)))
#taxi_df <- na.omit(taxi_df)
# taxi_2017COPY$Fare <- ifelse(taxi_2017COPY$Fare, NA, taxi_2017COPY$Fare)
taxi_2017COPY$Fare[taxi_2017COPY$Fare == ""] <- NA
taxi_2017COPY <- na.omit(taxi_2017COPY)

#date_time <- strptime(taxi_df$`Trip Start Timestamp`, '%m/%d/%Y %I:%M:%S %p')
date_time_2017 <- strptime(taxi_2017COPY$`Trip Start Timestamp`, '%m/%d/%Y %I:%M:%S %p')

# taxi_2017[, `:=` (Month = as.numeric(strftime(date_time_2017, '%m')),
#                   Year = as.numeric(strftime(date_time_2017, '%Y')),
#                   Day = as.numeric(strftime(date_time_2017, "%d")),
#                   Date = as.Date(strftime(date_time_2017, "%Y-%m-%d")),
#                   ]
taxi_2017COPY$Month <- as.numeric(strftime(date_time_2017, '%m'))
taxi_2017COPY$Year <- as.numeric(strftime(date_time_2017, '%Y'))
taxi_2017COPY$Day <- as.numeric(strftime(date_time_2017, "%d"))
taxi_2017COPY$Date <- strftime(date_time_2017, "%Y-%m-%d")

#taxi_2017$X <- 1:nrow(taxi_2017)

taxi_med17 <- taxi_2017COPY[, .(TotalFare = sum(as.numeric(gsub('[$]', '', Fare))), Date), by = list(`Ta

# taxi_med_2017 <- taxi_2017[, .(TotalFare = sum(as.numeric(gsub('[$]', '', Fare))),
#                         Date), by = list(`Taxi ID`, Month, Day, Year)][, .(Median = median(TotalFare),
# $Date <- as.Date(with(taxi_med, paste(Year, Month, Day,sep="-")), "%Y-%m-%d")
# Fare_Data <- taxi_med
```

```
taxi_med_2017 <- taxi_2017[, .(TotalFare = sum(as.numeric(gsub('[$]', '', Fare)))), by = list(`Taxi ID`
taxi_med$Date <- as.Date(with(taxi_med, paste(Year, Month, Day,sep="-")), "%Y-%m-%d")


#med_ts <- ts(Fare_Data[,c('Median')])
#Fare_Data$clean_med <- tsclean(med_ts)

ggplot() +
  geom_line(data = Fare_Data, aes(x = Date, y = clean_med)) + ylab('Cleaned Taxi Median Fare') +
  ggtitle("Median Daily Fare") + theme(plot.title = element_text(hjust = .5))

## Calculates Moving averages for Month and Week
#Fare_Data$med_ma = ma(Fare_Data$clean_med, order = 7) # using the clean count with no outliers
#Fare_Data$med_ma30 = ma(Fare_Data$clean_med, order = 30)
## Plots Counts with Moving AVerage for Week and Month
ggplot() +
  geom_line(data = Fare_Data, aes(x = Date, y = clean_med, colour = "Median Taxi Fare")) +
  geom_line(data = Fare_Data, aes(x = Date, y = med_ma,   colour = "Weekly Moving Average"))  +
  geom_line(data = Fare_Data, aes(x = Date, y = med_ma30, colour = "Monthly Moving Average"))  +
  ylab('Taxi Fare')  +
  ggtitle("Median Daily Fare") + theme(plot.title = element_text(hjust = .5))

## Calculates Seasonal components using stl
#med_ma <- ts(na.omit(Fare_Data$med_ma), frequency=52)
#decomp <- stl(med_ma, s.window="periodic")

## seasadj() removes the seasonality by subtracting the seasonal component from the original series
deseasonal_med <- seasadj(decomp)
plot(decomp, main = 'Median Fare Decomposition')
plot(deseasonal_med, main = 'Deseasonal Moving Average')

adf.test(med_ma, alternative = "stationary")
adf.test(deseasonal_med, alternative = "stationary")
Acf(med_ma, main='Median Moving Average') ## EXAMPLE PURPOSES ONLY NOT ANALYSIS
Pacf(med_ma, main='Median Moving Average') ## SEE ABOVE
med_d1 <- diff(deseasonal_med,difference = 1)
plot(med_ma)
plot(med_d1, main = 'Differentiated Moving Average', ylab = '')
adf.test(med_d1, alternative = 'stationary')
Acf(med_d1, main='Differenced Median Moving Average')
Pacf(med_d1, main='Differenced Median Moving Average')

#Arima_med_naive <- Arima(ts(deseasonal_med, frequency = 365), order = c(3,1,8))
#Med_Forecast_naive <- forecast(Arima_med_naive, h = 365)
plot(Med_Forecast_naive, main = 'Naive Model: ARIMA(3,1,8)', ylab = 'Median Daily Fare in $', xaxt = 'n
axis(1, at=1:6, labels=2013:2018)

#Arima_med_naive_seas <- Arima(ts(deseasonal_med, frequency = 365), order = c(3,1,8), seasonal = c(0,1,
#Med_Forecast_naive_seas <- forecast(Arima_med_naive_seas, h = 365)
plot(Med_Forecast_naive_seas, main = 'Naive Seasonal Model: ARIMA(3,1,8)', ylab = 'Median Daily Fare in


#autoarima <- auto.arima(deseasonal_med, D = 1, approximation = TRUE)
```

```r
#equivalent to autoarima <- Arima(ts(deseasonal_med,frequency = 365), order = c(4,1,4), seasonal = c(0,
plot(forecast(autoarima, h = 365), main = 'Stepwise AIC-Minimized Seasonal Model: ARIMA(4,1,4)', ylab =
taxi_med17$Fitted <- tail(Med_Forecast_naive_seas$fitted, n = 212)
taxi_med17$AAFitted <- tail(autoarima$fitted, n = 212)
ggplot(taxi_med17) + aes(x = Date, y = Median) + geom_point() + geom_smooth(method = "lm")
plot_internal

ggplot(taxi_med17) + aes(x = Date) + geom_point(aes(y = Median)) + geom_point(aes(y = Fitted), color =



#tsfit <- tslm(Median~trend + season, data = ts(Fare_Data, freq = 365))
tsfit
```