# Ride Sharing Optimization

A Data Science Project in Python

Eugene Cheah
Master of Science in Information Systems
College of Engineering, Northeastern University
Boston, USA
Cheah.eug@husky.neu.edu

Mengting Yang
Master of Science in Information Systems
College of Engineering, Northeastern University
Boston, USA
Yang.meng@husky.neu.edu

Rahil Shah
Master of Science in Information Systems
College of Engineering, Northeastern University
Boston, USA
Shah.rah@husky.neu.edu

***Abstract*— We are aspiring data scientists working on a project to predict the prices of ride sharing services Uber and Lyft. We have retrieved data from Uber and Lyft and applied linear regression, machine learning algorithms, the Moving Average model, deep learning with TensorFlow, and the Facebook Prophet package. With the linear regression model, we got an accuracy of 46%; with the moving average model we got a Residual Sum of Squares value of 0.0063; with Deep Learning Neural Networks we achieved a low Mean Squared Error at 0.7.**

***Data science; regression; time series; machine learning; deep learning***

## I. Introduction

Whether you're a driver or passenger, you must've found yourself pondering whether to use Uber or Lyft at some point. Time and money are lost in the decision-making process. Our solution is to build an app that streamlines the decision process. By providing real-time and forecasted prices of both ride-sharing services, users can now make an informed decision. Our team has successfully automated the continuous extraction of data from Uber and Lyft. We have also applied models in linear regression, machine learning algorithms, moving average, deep learning recurrent neural network as well as the Facebook prophet package for forecasting the prices estimates of Uber and Lyft.

## II. Code with Documentation

### Machine Learning

Link:
https://github.com/rahilshah10/IS/tree/master/ADS/Final%20Project

### Moving Average Model Of Time Series

Link: https://github.com/MandyYang86/Ride-Optimization/tree/master/MA%20Model

### Deep Learning and Facebook Prophet

Link:
https://github.com/echeah/big_data_systems_and_intelligence_analytics/tree/master/rides_sharing_optimization_project

## III. Results & Discussion

### Linear Regression

#### *A. Introduction to Linear Regression*

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable causes the other (for example, higher SAT scores do not cause higher college grades), but that there is some significant association between the two variables.

A linear regression line has an equation of the form Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0). [10]

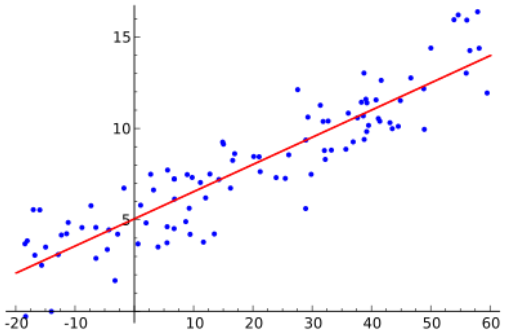


Fig.1 Regression Line

*B. Least squares Regression*

The most common method for fitting a regression line is the method of least-squares. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). Because the deviations are first squared, then summed, there are no cancellations between positive and negative values.

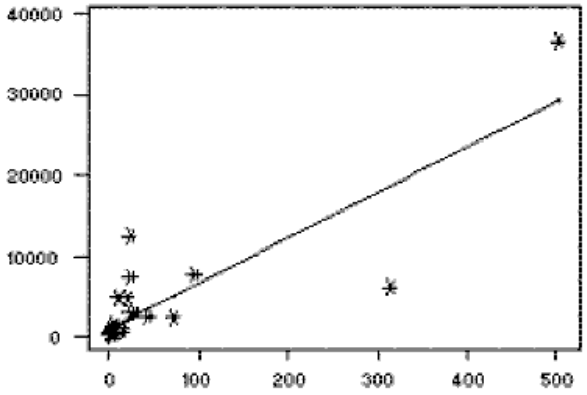


Fig.2 Least Squared regression

*C. Outliers and Influential Observations*

After a regression line has been computed for a group of data, a point which lies far from the line (and thus has a large residual value) is known as an outlier. Such points may represent erroneous data, or may indicate a poorly fitting regression line.

If a point lies far from the other data in the horizontal direction, it is known as an influential observation. The reason for this distinction is that these points have may have a significant impact on the slope of the regression line.

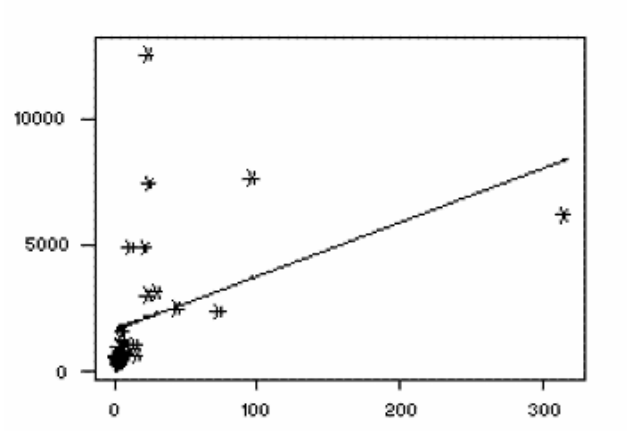


Fig.3 Influential Outliers

*D. Residuals*

Once a regression model has been fit to a group of data, examination of the residuals (the deviations from the fitted line to the observed values) allows the modeler to investigate the validity of his or her assumption that a linear relationship exists. Plotting the residuals on the y-axis against the explanatory variable on the x-axis reveals any possible non-linear relationship among the variables, or might alert the modeler to investigate lurking variables.

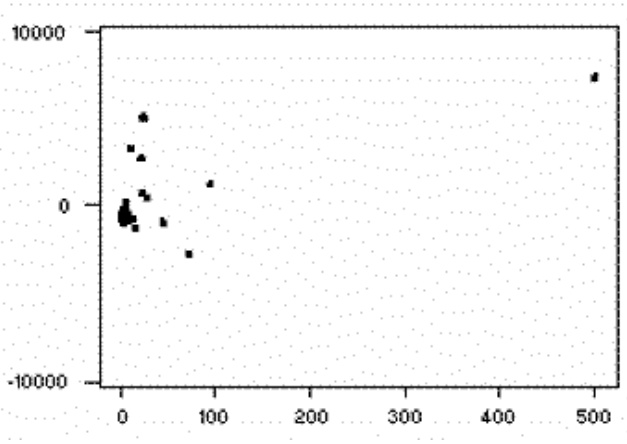


Fig.4 Residuals

*E. Correlation*

The strength of the linear association between two variables is quantified by the correlation coefficient.

Given a set of observations (x1, y1), (x2, y2),...(xn, yn), the formula for computing the correlation coefficient is given by

$$r = \frac{1}{n-1} \sum \left( \frac{x - \bar{x}}{s_x} \right) \left( \frac{y - \bar{y}}{s_y} \right)$$

The correlation coefficient always takes a value between -1 and 1, with 1 or -1 indicating perfect correlation (all points would lie along a straight line in this case). A positive correlation indicates a positive association between the variables (increasing values in one variable correspond to increasing values in the other variable), while a negative correlation indicates a negative association between the variables (increasing values is one variable correspond to decreasing values in the other variable). A correlation value close to 0 indicates no association between the variables.

Since the formula for calculating the correlation coefficient standardizes the variables, changes in scale or units of measurement will not affect its value. For this reason, the correlation coefficient is often more useful than a graphical depiction in determining the strength of the association between two variables.

*F. Finding the Accuracy*

For linear regression we need proper data types, that is, the data must not be in categorical form, so we can apply classification and regression. Thus, change the weather attribute into labels, and apply it as the feature.

```
weather
Clear      [0]
Clouds     [1]
Drizzle    [2]
Fog        [3]
Haze       [4]
Mist       [5]
Rain       [6]
Snow       [7]
Name: weather_label, dtype: object
```

Fig.5 Weather Labels

Set the index to data_time for ease of use and access. We use the features: (uber_distance, uber_duration, lyft_price_per_second, average_duration, weather_label) to predict our label: uber_price_per_second and, similarly, for lyft.

Generally, you want your features in machine learning to be in a range of -1 to 1. This may do nothing, but it usually speeds up processing and can also help with accuracy. Because this range is so popularly used, it is included in the preprocessing module of Scikit-Learn. To utilize this, you can apply preprocessing.scale to your X variable

The way this works is you take, for example, 75% of your data, and use this to train the machine learning classifier. Then you take the remaining 25% of your data, and test the classifier. Since this is your sample data, you should have the features and known labels. Thus, if you test on the last 25% of your data, you can get a sort of accuracy and reliability, often called the confidence score. There are many ways to do this, but, probably the best way is using the build in cross_validation provided, since this also shuffles your data for you.

The return here is the training set of features, testing set of features, training set of labels, and testing set of labels. Now, we're ready to define our classifier. There are many classifiers in general available through Scikit-Learn, and even a few specifically for regression.

The accuracy that for uber is 0.4608371 and for lyft is 0.48123823.

The errors between the actual and predicted values is found out to be:

```
('Mean Baseline Error: ', 27.401863785539764)
('Mean Linear Regression Error: ', 18.729525078588573)
('Mean Random Forest Error: ', 18.47665105800672)
('Mean XGBoost Error: ', 18.443395088760013)
```

## MOVING AVERAGE MODEL OF TIME SERIES

*A. Introduction of Time Series*

Time Series is a collection of data points collected at constant time intervals and is usually analyzed to determine long-term trends and to predict the future or perform other forms of analysis.

Time Series is time dependent and along with an increasing or decreasing trend, some Time Series have some form of seasonality trends which we can analysis.

*B. Introduction of Moving Average*

A Moving Average is a calculation to analyze data points by creating series of averages of different subsets of the full data set. Given a series of numbers and a fixed subset size, the first element of the moving average is obtained by taking the average of the initial fixed subset of the number series.

*1) Simple Moving Average:* The simplest form of a moving average, appropriately known as a simple moving average (SMA) [1], is calculated by taking the arithmetic mean of a given set of values. In other words, a set of numbers, are added together and then divided by the number in the set.

The way it moves is that the new values become available while the oldest data points must be dropped from the set. Since new data points come in to replace them, the data set is constantly "moving" to account for new data. This method of calculation ensures that only the current information is being accounted for.

*2) Exponential Moving Average:* In order to avoid the problem that each point in the data series uses the same weighting, regardless of where it occurs in the sequence, we can use a new kind of moving average calculation method - Exponential Moving Average. In this way, the most recent data is more significant than the older data and have a greater influence on the final result. Below is the EMA equation [2]:

$$EMA = (P * \alpha) + (Previous\ EMA * (1 - \alpha))$$

$$P = Current\ Price$$
$$\alpha = Smoothing\ Factor = \frac{2}{1 + N}$$
$$N = Number\ of\ Time\ Periods$$

There is no value available to use as the previous EMA and this small problem can be solved by starting the calculation with a simple moving average and continuing on with the above formula from there.

*3) Weighted Moving Average:* Exponential Moving Average is a kind of Weighted Moving Average since it places more emphasis on recent data.

Weighted Moving Average is a weighted average of the last n prices, where the weighting decreases with each previous price. This is a similar concept to the EMA, but the calculation for the WMA is different [3].

$$(Price\ X\ weighting\ factor)$$
$$+ (Price\ previous\ period\ X\ weighting\ factor - 1)$$

For example, if there are four prices you want a weighted moving average of, then the most recent weighting could be 4/10, the period before could have a weight of 3/10, the period prior to that could have a weighting of 2/10, and so on. 10 is a randomly picked number, and a weight of 4/10 means the most recent price will account for 40% of the value of the WMA. The price three periods ago only accounts of 10% of the WMA value.

### *C. Loading and Handling Time Series in Pandas*

Pandas has proven very successful as a tool for working with time series data. Using timedelta64 dtypes, we can easily manipulate the time series data.

After reading the csv file using pandas, we transfer the data type of 'date_time' column to datetime64[ns] and set it as index to help us analysis.

Since the original data is too large for time series analysis, we resample the 'uber_price_per_second' to 2 hours period using the mean of them.

### *D. Stationary*

A Time Series is said to be Stationary if its statistical properties such as mean, variance remain constant over time. Since most of the Time Series models work on the assumption that the Time Series is stationary. If a Time Series has a particular behavior over time, there is a very high probability that it will follow the same in the future.

Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series.

*1) Evaluation:* For practical purposes, we can assume the series to be stationary if it has constant statistical properties over time. For example:

*a) Constant Mean:* The mean of the series should not be a function of time rather should be a constant.

*b) Constant Variance:* The variance of the series should not a be a function of time. This property is known as homoscedasticity.

*c) An auto covariance that does not depend on time:* The covariance of the $i_{th}$ term and the $(i + m)_{th}$ term should not be a function of time.

When we plot the price per second of Uber price, we cannot observe the stationary directly. So, we define a function to test time series' stationary. Since we resample the data as 2-hour period, we set the window as 12 because we want to take one day for 24 hours. Then we will plot the original data, rolling mean and rolling standard of the time series we want to test. Also, in order to know the stationary in a more statistical way, we also use the Dickey-Fuller test. In statistics, the Dickey–Fuller test tests the null hypothesis that a unit root is present in an autoregressive model. The alternative hypothesis is stationarity. The regression model of Dickey–Fuller test can be written as [4]

$$\Delta y_t = (\rho - 1) y_{t-1} + \mu_t = \delta y_{t-1} + \mu_t$$

where Δ is the first difference operator. This model can be estimated and testing for a unit root is equivalent to testing $\delta = 0$ (where $\delta = \rho - 1$). Since the test is done over the residual term rather than raw data, it is not possible to use standard t-distribution to provide critical values. Therefore, this statistic t has a specific distribution simply known as the Dickey–Fuller table.

The test results comprise of a test statistic and some critical values for difference confidence levels.
If the test statistic is less than the critical value, we can reject the null hypothesis and say that the series is stationary.

The stationary plot of our Uber price is shown as Fig.6:

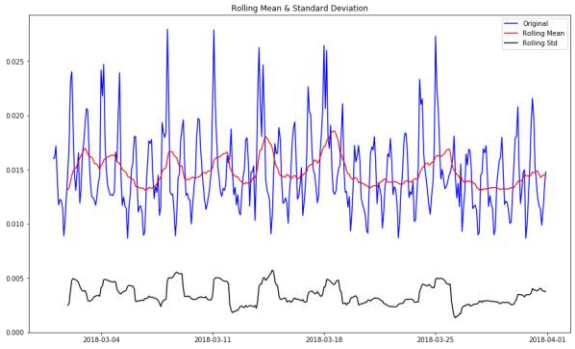



Fig.6 stationary of Uber price

The stationary result of our Uber price is shown as Table.1:

TABLE 1. UBER PRICE STATIONARY

| | |
|---|---|
| Test Statistic | -4.938296 |
| p-value | 0.000029 |
| #Lags Used | 17.000000 |
| Number of Observations Used | 354.000000 |
| Critical Value (1%) | -3.448958 |
| Critical Value (5%) | -2.869739 |
| Critical Value (10%) | -2.571138 |

The Test Statistic is -4.938296, which is less than Critical Value (1%) -3.448958. So, we have 99% confidence to say that the data is stationary.

However, we can still see some Seasonality like fluctuation in our dataset. So, we will do some effort to make our data more stable.

Usually, if a Time Series is non-stationarity, there are usually two reasons:

The first one is trend, it means varying mean over time. For example, the price grows over time. The second one is seasonality variations at specific time-frames. For example, people might have a tendency to buy cars in a particular month because of pay increment or festivals. Then we need to eliminate the impact of these two elements.

*E. Eliminating Trend and Seasonality*

*1) Reduce trend:* we can try to reduce trend using transformation which penalize higher values more than smaller values. We take the log transform method and then use moving average to remove the trend from the log data. After this, we test the stationary again, the result is shown below as Table.2:

TABLE 2. LOG MOVING AVERAGE

| | |
|---|---|
| Test Statistic | -6.530094e+00 |
| p-value | 9.919489e-09 |
| #Lags Used | 1.700000e+01 |
| Number of Observations Used | 3.430000e+02 |
| Critical Value (1%) | -3.449560e+00 |
| Critical Value (5%) | -2.870004e+00 |
| Critical Value (10%) | -2.571279e+00 |

The test statistic is smaller than the 1% critical values so we can say with 99% confidence that this is a stationary series. Also, the p value reduces a lot.

Also, we try Weighted Moving Average to see if there is any improvement and the stationary result is shown below as Table.3:

TABLE 3. LOG WEIGHTED MOVING AVERAGE

| | |
|---|---|
| Test Statistic | -5.936612e+00 |
| p-value | 2.312900e-07 |
| #Lags Used | 1.700000e+01 |
| Number of Observations Used | 3.540000e+02 |
| Critical Value (1%) | -3.448958e+00 |
| Critical Value (5%) | -2.869739e+00 |
| Critical Value (10%) | -2.571138e+00 |

The stationary does improve but not much and for the predict purpose, we will use simple moving average method which is efficient and easy to remove the trend.

*2) Reduce seasonality:* There are two ways of removing trend and seasonality:

*a) Differencing:* Taking the difference with a particular time lag.

In this technique, we take the difference of the observation at a particular instant with that at the previous instant. So, we shift one unit of our log data and test the stationary of the data as Table.4:

TABLE 4. DIFFERENCING

| | |
|---|---|
| Test Statistic | -5.632004 |
| p-value | 0.000001 |
| #Lags Used | 17.000000 |
| Number of Observations Used | 353.000000 |
| Critical Value (1%) | -3.449011 |
| Critical Value (5%) | -2.869763 |
| Critical Value (10%) | -2.571151 |

Also, the plot looks much better as Fig.7:

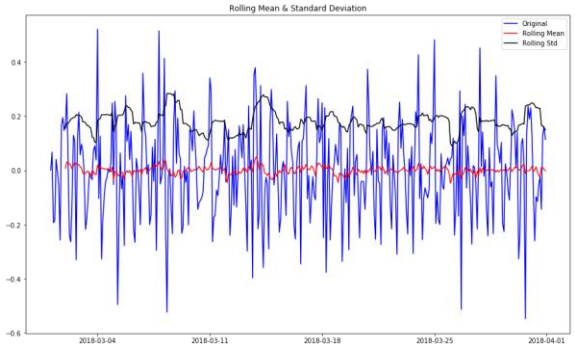


Fig.7 stationary of Uber price after differencing

P value reduces a lot and the rolling mean is closer to 0, which proves our data becomes more stable

*b) Decomposition:* Modeling both trend and seasonality and removing them from the model.
In this approach, both trend and seasonality are modeled separately and the remaining part of the series is returned.

We use seasonal_decompose from statsmodels.tsa.seasonal to split the trend, seasonal and residual from our data and plot them together as Fig.8:

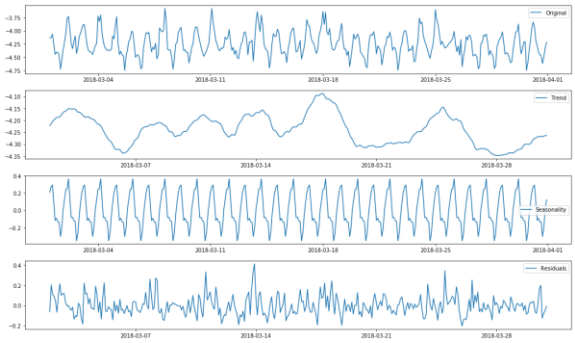


Fig.7 Decomposing

Then we test the stationary of our data after decomposing and show the result in Table.5:

TABLE 5. DECOMPOSING

| | |
|---|---|
| Test Statistic | -8.768752e+00 |
| p-value | 2.554897e-14 |
| #Lags Used | 9.000000e+00 |
| Number of Observations Used | 3.380000e+02 |
| Critical Value (1%) | -3.449846e+00 |
| Critical Value (5%) | -2.870129e+00 |
| Critical Value (10%) | -2.571346e+00 |

The result is not as good as Differencing, so for our project, we will take Differencing method.

*E. Forecasting a Time Series*

We will make model on the Time Series after differencing. Now we get a series with significant dependence among values. In this case we need to use some statistical models like ARIMA to forecast the data. ARIMA stands for Auto-Regressive Integrated Moving Averages.

The ARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation.

Given a time series of data $X_t$ where t is an integer index and the $X_t$ are real numbers, an $ARMA\ (p', q)$ model is given by [5]

$$X_t - \alpha X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

The predictors depend on the parameters (p, d, q) of the ARIMA model:

P stands for Number of AR (Auto-Regressive) terms, AR terms are just lags of dependent variable.

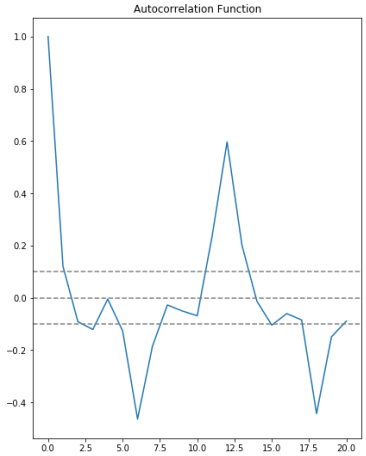
Q stands for Number of MA (Moving Average) terms. MA terms are lagged forecast errors in prediction equation.
D stands for number of differences.

An importance concern here is how to determine the value of 'p' and 'q'. We use Autocorrelation Function (ACF) plot to determine q and Partial Autocorrelation Function (PACF) plot to determine q.

Autocorrelation Function (ACF) is a measure of the correlation between the Time Series with a lagged version of itself. For instance, at lag 5, ACF would compare series at time instant '$t_1$'…'$t_2$' with series at instant '$t_{1-5}$…$t_{2-5}$ ($t_{1-5}$ and $t_2$ being end points).

Partial Autocorrelation Function (PACF) measures the correlation between the Time Series with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. For example, at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

We import acf, pacf from statsmodels.tsa.stattools and get the plots below as Fig.8:



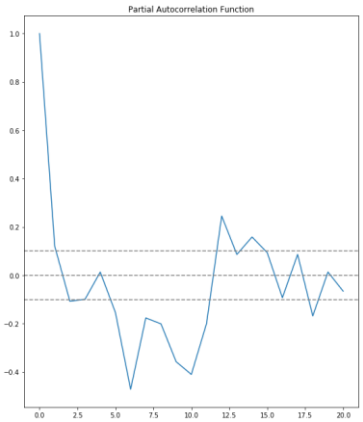




Fig.8 Autocorrelative function. The x-axis and y-axis represent the value of lag and the standard of AF and PAF

In these two plots, the two dotted lines on either side of 0 are the confidence intervals. p can be determined as the lag value where the PACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case p=2. q can be determined as the lag value where the ACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case q=2.

We make 3 different ARIMA models considering individual as well as combined effects.

*1) AR Model*

We take (p, d, q) as (2, 1, 0) to be used in AR Model and here is the result of Fig.9

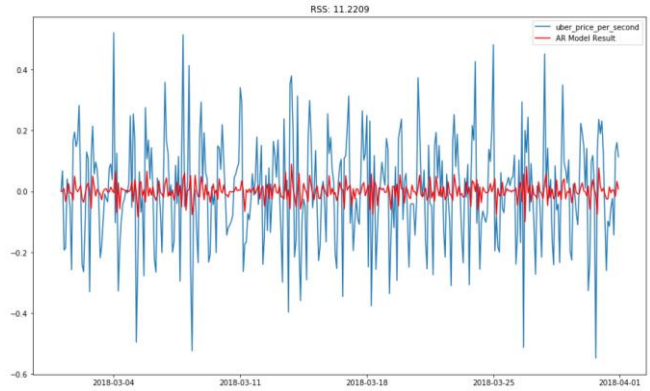



Fig.9 AR Model. The x-axis and y-axis represent the time periods and ride sharing price respectively

*2) MR Model*

RSS is 11.2209 of the AR Model which is really not good, so we try MA Model using (p, d, q) as (0, 1, 2) and show the result is Fig.10:

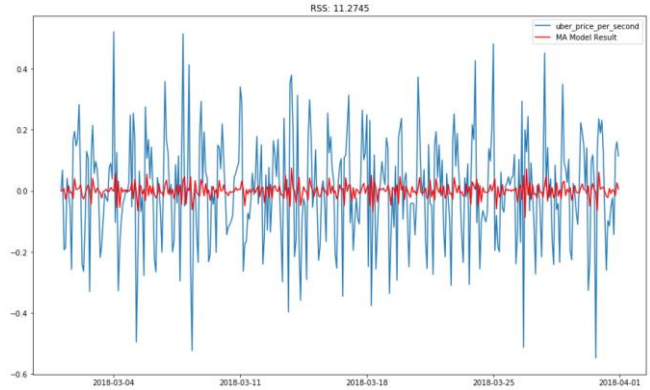



Fig.10 MR Model. The x-axis and y-axis represent the time periods and ride sharing price respectively

*3) Combined Model*

RSS is 11.2745 of the MR Model which is really not good as well, so we try to combine MA Model and AR Model using (p, d, q) as (2, 1, 2) and show the result is Fig 11:

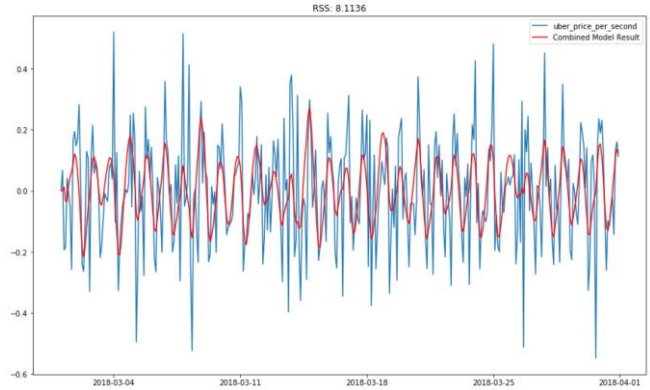



Fig.11 Combination Model. The x-axis and y-axis represent the time periods and ride sharing price respectively

RSS is 8.1336 of the Combined Model which is much better now, so we select Combined Model to predict.

*3) Predict*

First, we need to take these values back to the original scale and we step store the predicted results as a separate series and observe it.

| date_time | |
|---|---|
| 2018-03-01 02:00:00 | 0.000159 |
| 2018-03-01 04:00:00 | 0.000275 |
| 2018-03-01 06:00:00 | 0.012257 |
| 2018-03-01 08:00:00 | -0.034082 |
| 2018-03-01 10:00:00 | -0.034680 |

Freq: 2H, dtype: float64

Notice that these start from '2018-03-01 02:00:00' and not the first hour. This is because we took a lag by 1 and first element doesn't have anything before it to subtract from.

The way to convert the differencing to log scale is to add these differences consecutively to the base number. An easy way to do it is to first determine the cumulative sum at index and then add it to the base number. The cumulative sum can be found as:

| date_time | |
|---|---|
| 2018-03-01 02:00:00 | 0.000159 |
| 2018-03-01 04:00:00 | 0.000435 |
| 2018-03-01 06:00:00 | 0.012691 |
| 2018-03-01 08:00:00 | -0.021390 |
| 2018-03-01 10:00:00 | -0.056071 |

Freq: 2H, dtype: float64

Next, we need to add them to base number. For this we create a series with all values as base number and add the differences to it.

| date_time | |
|---|---|
| 2018-03-01 00:00:00 | -4.131946 |
| 2018-03-01 02:00:00 | -4.131787 |
| 2018-03-01 04:00:00 | -4.131512 |
| 2018-03-01 06:00:00 | -4.119255 |
| 2018-03-01 08:00:00 | -4.153337 |

Freq: 2H, dtype: float64

Last step is to take the exponent and compare with the original series. We get the prediction below as Fig.12:

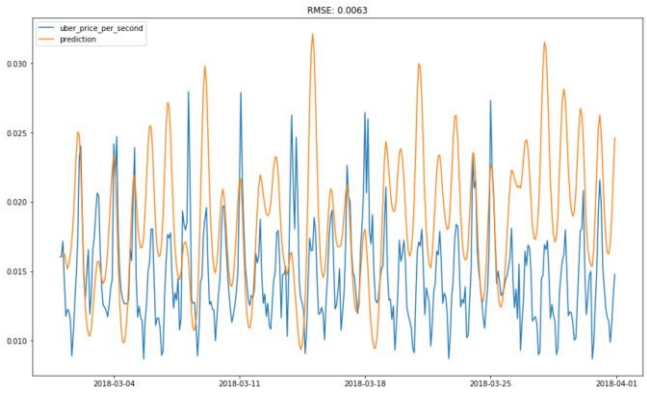



Fig.12 Prediction Result. The x-axis and y-axis represent the time periods and ride sharing price respectively

Finally, we have a forecast at the original scale. The RSS is 0.0063 which is not a very good forecast but we can predict some basic trend of Uber price. Next, we will try to find more accurate prediction method to apply.

## DEEP LEARNING

### *A. Introduction of Deep Learning*

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example [1]. Deep usually means the number of hidden layers in the neural network. The layers of deep networks can be more than 150. Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction. The flow of the deep learning is shown as Fig.1

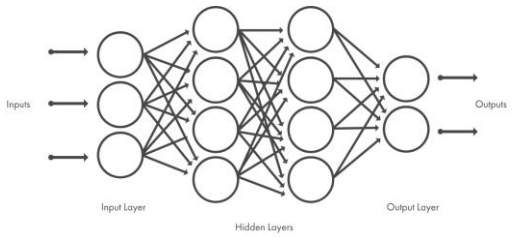



Fig.13 Flow of Deep Learning

### *B. Introduction of RNN*

RNN is Recurrent Neural Networks and we use this algorithm as one of our research methods: The idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs (and outputs) are independent of each other [2]. However, this doesn't work when we try to predict something based on the previous information. So, we introduce RNN as a new method. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. RNN also has e a "memory" which captures information about what

has been calculated so far. Here is what a typical RNN looks like in Fig2:

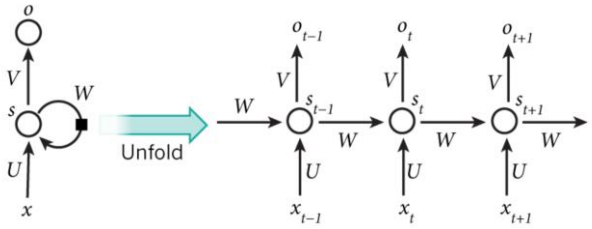



Fig.14 Typical RNN

$X_t$ is the input at time step t.
$S_t$ is the hidden state at time step t.
$O_t$ is the output at step t.

This diagram shows a RNN which is being unrolled into a full network. By unrolling we simply mean that we write out the network for the complete sequence.

The hidden state $S_t$ as the memory of the network which captures information about what happened in all the previous time steps.

*C. Introduction of LSTM*

LSTM (Long Short Term Memory networks) don't have a fundamentally different architecture from RNNs, but they use a different function to compute the hidden state. [3] The memory in LSTMs are called cells and they are used to take the from previous state input and current input. Internally these cells decide what to keep in memory. They then combine the previous state, the current memory, and the input. LSTMs are explicitly designed to avoid the long-term dependency problem as they can remember information for long periods of time.

LSTM have chain like structure, but the repeating module has a different structure. There are four neural network layers in LSTM like in Fig.3 and Fig.4

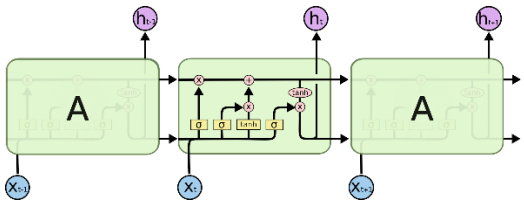



Fig.15 LSTM Structure

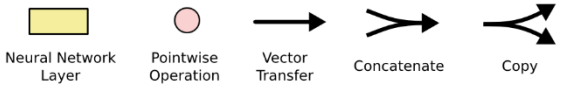



Fig.16 LSTM Structure explanation

In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The circles represent pointwise operations, while the boxes are learned neural network layers.

The key to LSTMs is the cell state which is kind of like a conveyor belt, the horizontal line running through the top of the diagram. It's very easy for information to just flow along it unchanged.

*D. Deep Learning with TensorFlow*

In this notebook, we'll be using TensorFlow. TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.[12] We have applied the ride-sharing data onto our TensorFlow models with different hyperparameter tunings.

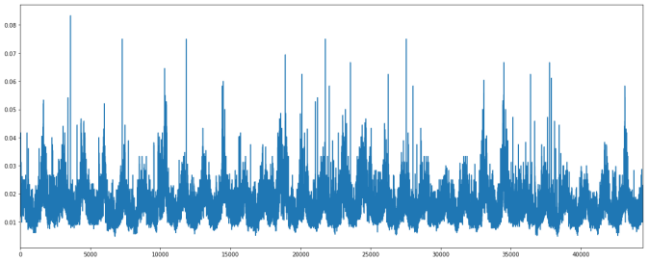


Fig.17 Uber Price Estimate. The x-axis and y-axis represent the Uber price per second and frequency respectively.

In Fig.3 we have a basic visualization of Uber price estimates and we're going to convert those data into array that can be broken up into training "batches" that we will feed into our RNN model. We have 44434 total observations in our March 2018 uberPOOL data. We want to make sure we have the same number of observations for each of our batch inputs. What we see is our training data set is made up of 444 batches, containing about 100 observations. Each observation is a sequence of a single value. The next step is to pull out our data.

Once we have our data reshaped, we'll create our TensorFlow graph that will do the computation. In our most optimal TF model, our hyperparameters were tuned with the following:

1. 100 periods per vector
2. 1 input vector
3. 500 hidden layers of neurons
4. 1 output vector
5. Activation function: ReLU
6. Learning rate: 0.001
7. Loss function: MSE
8. Gradient estimation: Adam Optimizer
9. 1000 epochs

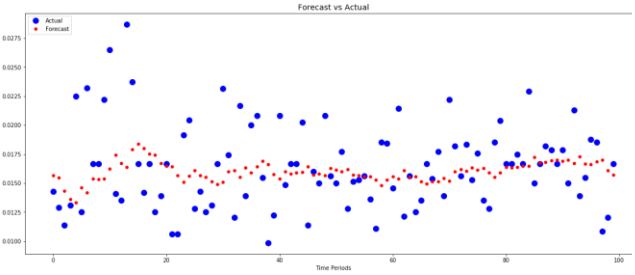



Fig.18 Uber Price Estimate Forecast vs Actual. The x-axis and y-axis represent the ride sharing price per second and time periods respectively.

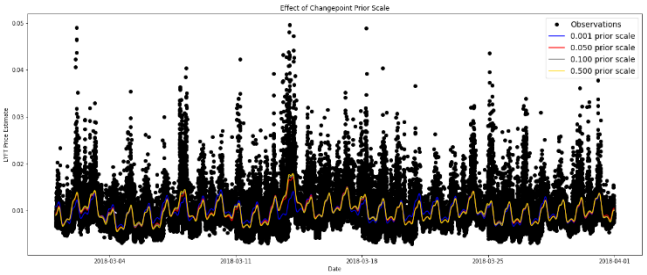



Fig.19 Effect of Changepoint Prior Scale. The x-axis and y-axis represent the ride sharing price and time periods respectively, with the effect of changepoint prior scaling of 0.001, 0.050, 0.100, and 0.500.

Fig.4 shows the plot of uberPOOL's forecast vs actual prices. We can see that the forecasted data points follow the actual prices pretty closely. After 100 epochs, the network reaches a plateau with an MSE of 0.7 and continues to be 0.7 following the next 900 epochs.

## FACEBOOK PROPOHET

Time series are one of the most common data types encountered in daily life. Financial prices, weather, home energy usage, and even weight are all examples of data that can be collected at regular intervals. Almost every data scientist will encounter time series in their daily work and learning how to model them is an important skill in the data science toolbox. One powerful yet simple method for analyzing and predicting periodic data is the additive model. The idea is straightforward: represent a time-series as a combination of patterns at different scales such as daily, weekly, seasonally, and yearly, along with an overall trend. Your energy use might rise in the summer and decrease in the winter, but have an overall decreasing trend as you increase the energy efficiency of your home. An additive model can show us both patterns/trends and make predictions based on these observations and the Prophet forecasting package developed by Facebook will help you do just that.

### *A. Modeling with Prophet*

The Facebook Prophet package was released in 2017 for Python and R, and data scientists around the world rejoiced. Prophet is designed for analyzing time series with daily observations that display patterns on different time scales. [11] It also has advanced capabilities for modeling the effects of holidays on a time-series and implementing custom changepoints.

We first import prophet and rename the columns in our data to the correct format. The Date column must be called 'ds' and the value column we want to predict 'y'. We then create prophet models and fit them to the data, much like a Scikit-Learn machine learning model.

When creating the prophet models, I set the changepoint to the default value of 0.05. This hyperparameter is used to control how sensitive the trend is to changes, with a higher value being more sensitive and a lower value less sensitive. This value is used to combat one of the most fundamental trade-offs in machine learning: bias vs. variance.

If we fit too closely to our training data, called overfitting, we have too much variance and our model will not be able to generalize well to new data. On the other hand, if our model does not capture the trends in our training data it is underfitting and has too much bias. When a model is underfitting, increasing the changepoint prior allows more flexibility for the model to fit the data, and if the model is overfitting, decreasing the prior limits the amount of flexibility. The effect of the changepoint prior scale can be illustrated by graphing predictions made with a range of values.

The higher the changepoint prior scale, the more flexible the model and the closer it fits to the training data. This may seem like exactly what we want, but learning the training data too well can lead to overfitting and an inability to accurately make predictions on new data. We therefore need to find the right balance of fitting the training data and being able to generalize to new data. As ride-sharing price estimates vary from day-to-day, and we want our model to capture this, we increased the flexibility after experimenting with a range of values.

### *B. Making Future Dataframes*

To make forecasts, we need to create what is called a future dataframe. We specify the number of future periods to predict (one day ahead) and the frequency of predictions (per minute). We then make predictions with the prophet model we created and the future dataframe.

Our future dataframes contain the estimated price for Uber and Lyft one day into the future. We can visualize predictions with the prophet plot function.

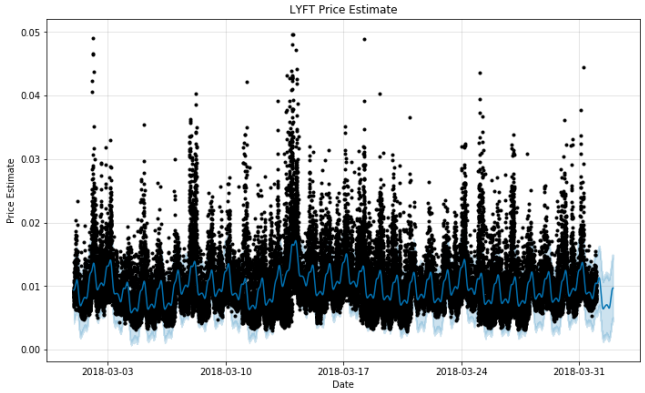



Fig.20 Lyft Price Estimates with One Day Forecast into the Future. The x-axis and y-axis represent the ride sharing price per second and time periods respectively.

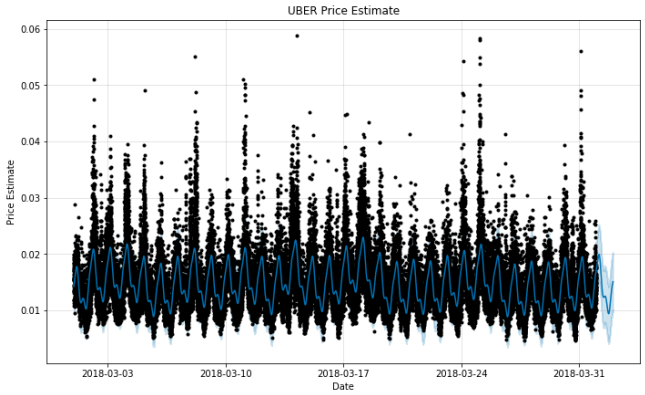



Fig.21 Uber Price Estimates with One Day Forecast into the Future. The x-axis and y-axis represent the ride sharing price per second and time periods respectively.

### *C. Trends and Patterns*

The last step of the analysis is to look at the overall trend and patterns. Prophet allows us to easily visualize the overall trend and the component patterns.

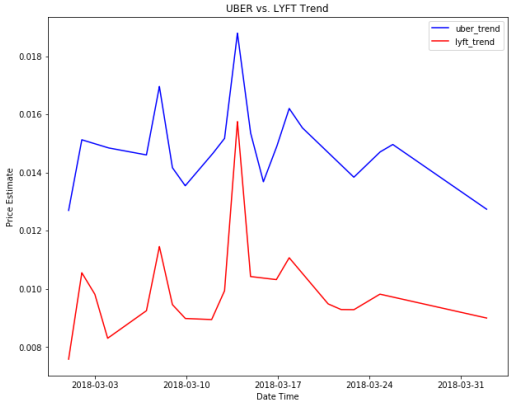



Fig.8 Uber vs Lyft Trend. The x-axis and y-axis represent the ride sharing price per second and time periods respectively.

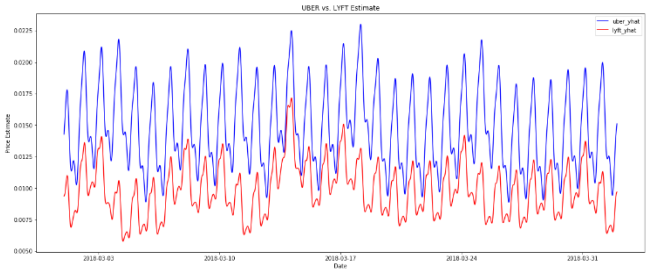



Fig.22 Uber vs Lyft Estimates. The x-axis and y-axis represent the ride sharing price per second and time periods respectively.

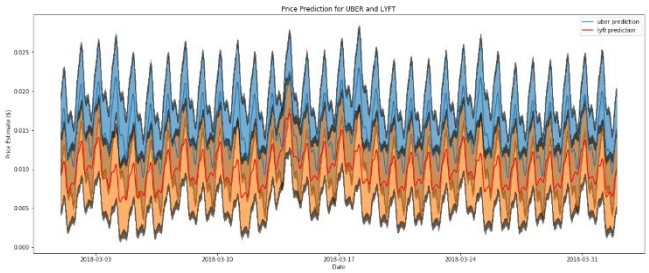



Fig.23 Price Predictions for Uber and Lyft. The x-axis and y-axis represent the ride sharing price per second and time periods respectively.

Fig.22, 23, and 24 show Lyft is generally cheaper than Uber.

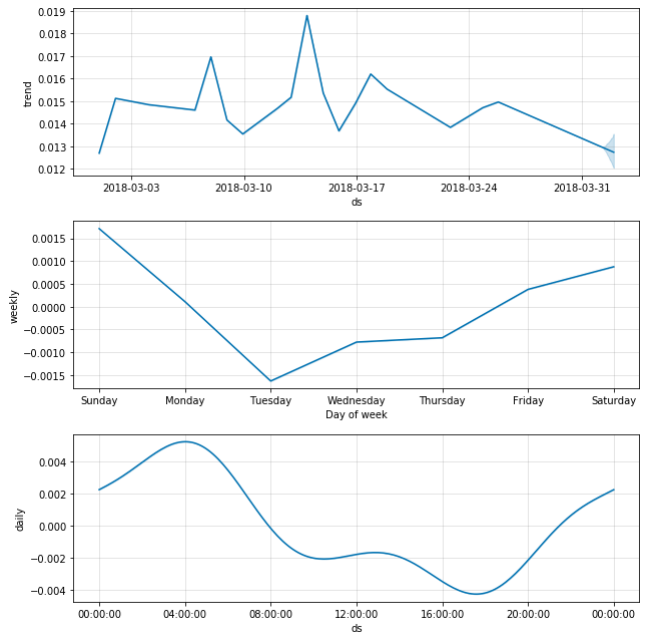



Fig.24 Uber Trend. The x-axis and y-axis represent the uberPOOL ride sharing price trend, weekly, daily and time periods respectively.

UberPOOL is generally more expensive during the weekends, especially on Sundays. Its daily trend shows a price surge between 2:00AM and 6:00AM.

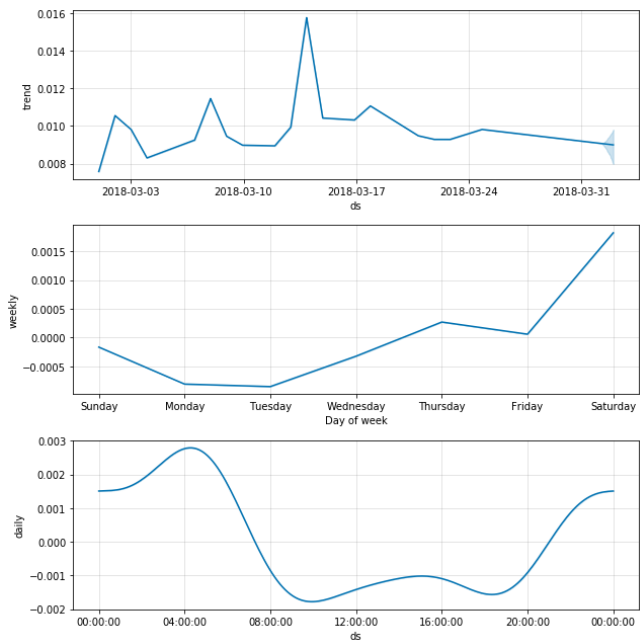



Fig.25 Lyft Trend. The x-axis and y-axis represent the Lyft Line ride sharing price trend, weekly, daily and time periods respectively.

Lyft Line is generally more expensive during the weekends as well, but especially on Saturdays. Its daily trend shows a price surge between 3:00AM and 5:00AM, which is a shorter window compared to uberPOOL.

## ACKNOWLEDGMENT

A special thank you to Professor Nik Bear Brown for providing guidance and thoughtful insights throughout the course of the project.

## REFERENCES

[1] https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
Time Series Forecasting Codes Python. Analytics Vidhya

[2] https://en.wikipedia.org/wiki/Moving_average
Moving average. Wikipedia

[3] https://www.investopedia.com/university/movingaverage/movingaverages1.asp#ixzz5D2bBkCmk
Moving Averages: What Are They? . Investopedia

[4] https://www.thebalance.com/simple-exponential-and-weighted-moving-averages-1031196
Simple, Exponential and Weighted Moving Averages. The Balance

[5] https://en.wikipedia.org/wiki/Dickey%E2%80%93Fuller_test
Dickey–Fuller test. Wikipedia

[6] https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
Autoregressive integrated moving average. Wikipedia

[7] https://www.mathworks.com/discovery/deep-learning.html
Deep Learning. MathWorks

[8] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
Recurrent Neural Networks Tutorial, Part 1 –Introduction to RNNs. WildML

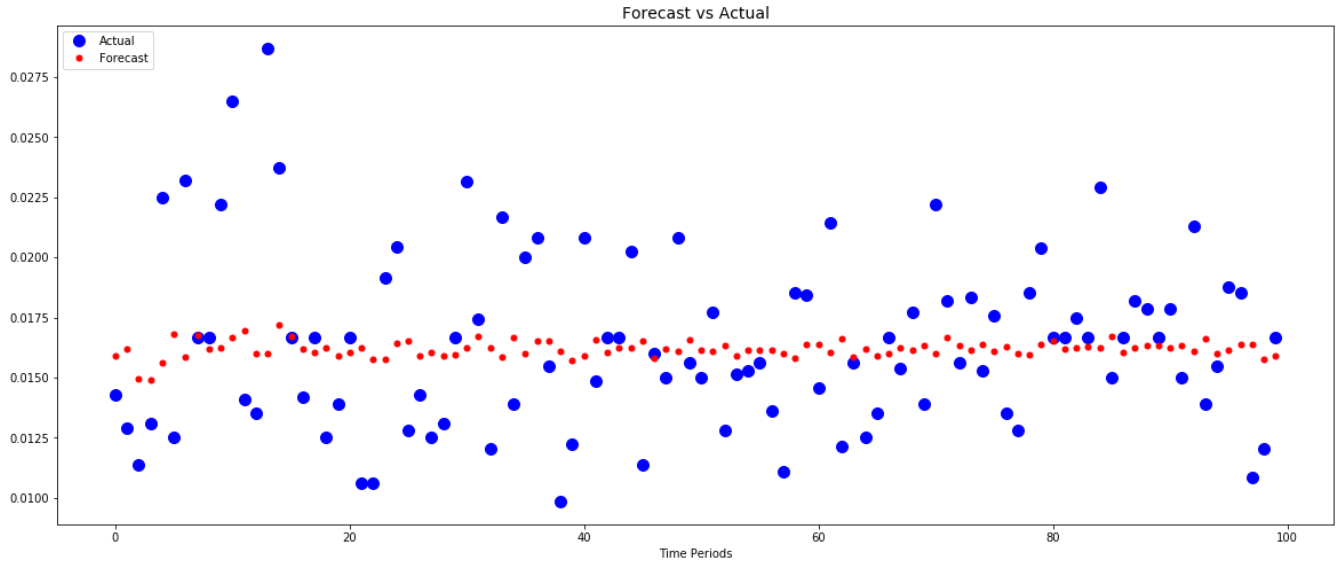
[9] http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Understanding LSTM Networks. Colah's blog
Autoregressive integrated moving average. Wikipedia

[10] http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm
Linear Regression

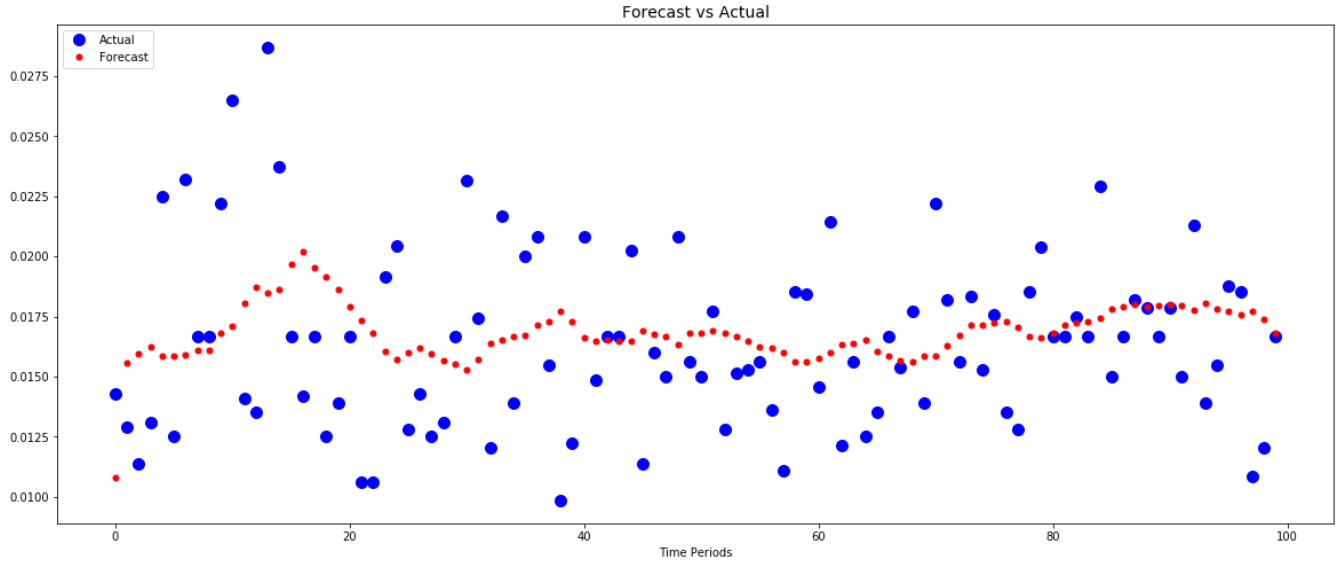
[11] https://towardsdatascience.com/time-series-analysis-in-python-an-introduction-70d5a5b1d52a
Time Series Analysis in Python: An Introduction

[12] https://en.wikipedia.org/wiki/TensorFlow
TensorFlow. Wikipedia.

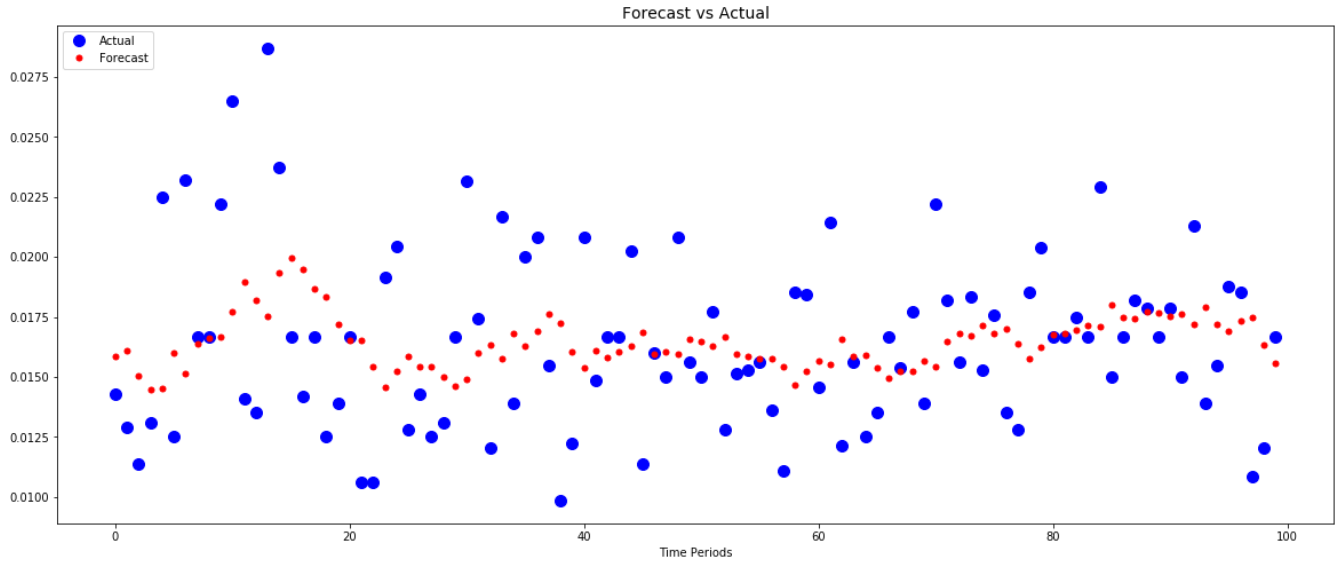
## APPENDIX




This figure shows our TF model with Sigmoid as the activation funtion. The x-axis and y-axis for this and the following figures represent the ride sharing price per second and time periods respectively.




This figure shows our TF model with Adadelta as the gradient estimation.




This figure shows our TF model with 100 hidden layers of neuron, instead of 500.