NJ Transit Rail Performance Prediction Report

Siyuan Tang | Chenghao Xu | Wenhuan Liu | Mengting Yang

Abstract

This is the final project for the class INFO6015 Data Science Engineer Methods Fall 2019. We work as a team to predict the future delay minutes of NJ transit trains. We retrieve data from existing dataset in Kaggle and applied Linear Regression model, ARIMA model and Facebook Prophet Prediction model.

Keywords: NJ Transit; Delay minute; Regression; Time Series; Machine Learning; Prediction

Introduction

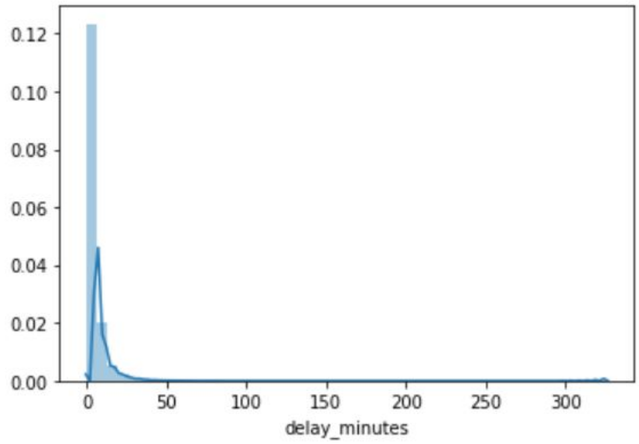
NJ Transit is the second largest commuter rail network in the United States by ridership; it spans New Jersey and connects the state to New York City. On the Northeast Corridor, the busiest passenger rail line in the United States, Amtrak also operates passenger rail service; together, NJ Transit and Amtrak operate nearly 750 trains across the NJ transit rail network.
Despite serving over 300,000 riders on the average weekday, no granular, trip-level performance data is publicly available for the NJ Transit rail network or Amtrak. This dataset aims to publicly provide such data.

This dataset contains monthly CSVs covering the performance of nearly every train trip on the NJ Transit rail network.

- Stop-level, minute resolution data on 287,000+ train trips (248,000+ NJ Transit trips, 38,000+ Amtrak trips)
- Coverage from March 1, 2018 to April 30, 2019 (updated monthly)
- Transparent reporting on train trips for which data was missing/invalid, or that were scraped or parsed incorrectly (97.5% of train trips were correctly captured)

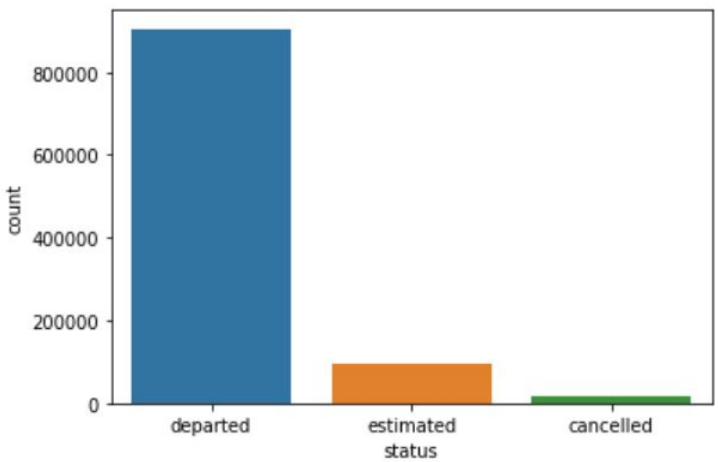
DATA VISUALIZATION

1. Probability Density Function for Delay

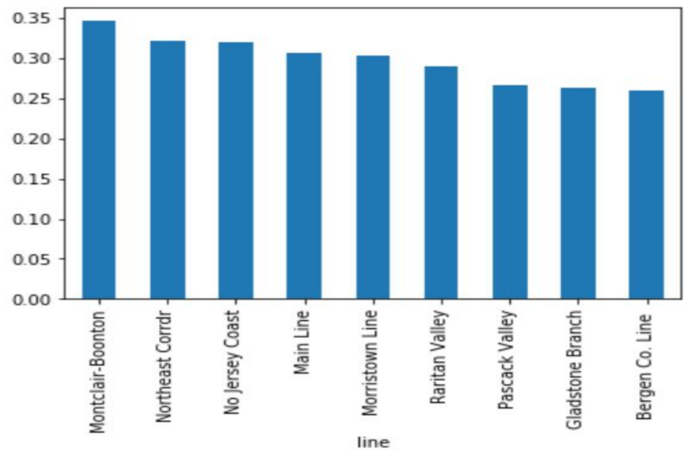


According to the histogram, we can know that there are some outliers with large delays

2. Status of trains from Nov 2018 - April 2019



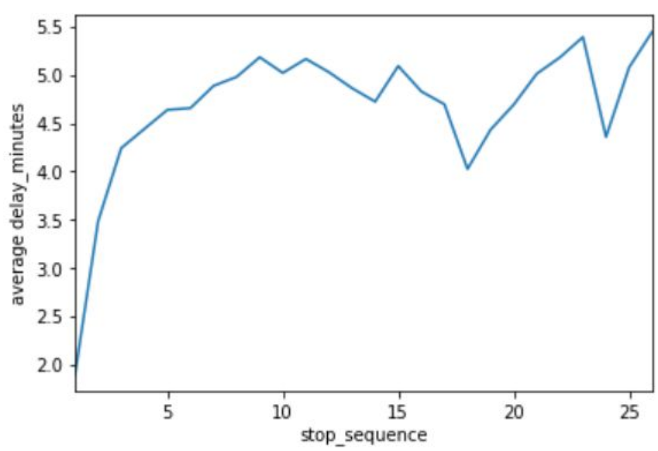
3. Lines' relationship with delay and cancellation




```
line
Bergen Co. Line       0.005272
Gladstone Branch      0.051631
Main Line             0.003927
Montclair-Boonton     0.018388
Morristown Line       0.019404
No Jersey Coast       0.019425
Northeast Corrdr      0.013485
Pascack Valley        0.006463
Raritan Valley        0.007298
dtype: float64
```

```
line
Bergen Co. Line       503
Gladstone Branch     4222
Main Line             391
Montclair-Boonton    1679
Morristown Line      3200
No Jersey Coast      2960
Northeast Corrdr     2380
Pascack Valley        451
Raritan Valley        487
Name: cancelled, dtype: int64
```

From the graph and the result we get, we can get the conclusion that all 9 lines has at least 25% of trips existing delay more than 5 minutes.



Also, we can see the delay and cancellation do not have much relation with different lines.

4. Stops' relationship with delay

From the above, we can see the stop sequence 's increasing will increase the delay time in some degree.

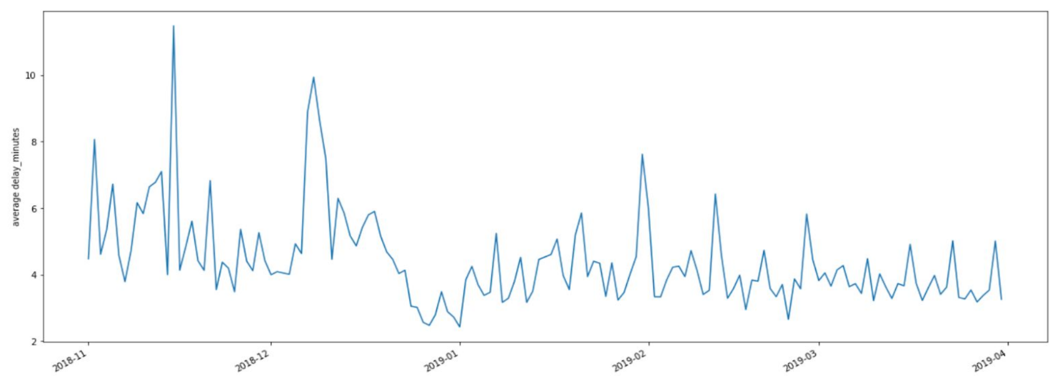
5. Origins and destinations' relationship with delay

```
from
Harriman            7.798562
North Branch        7.278237
Campbell Hall       6.885622
Wayne-Route 23      6.577950
Lincoln Park        6.376451
Mountain View       6.160504
Lebanon             6.117999
Tuxedo              6.051452
Aberdeen-Matawan    5.849044
Otisville           5.837905
Name: delay_minutes, dtype: float64
```

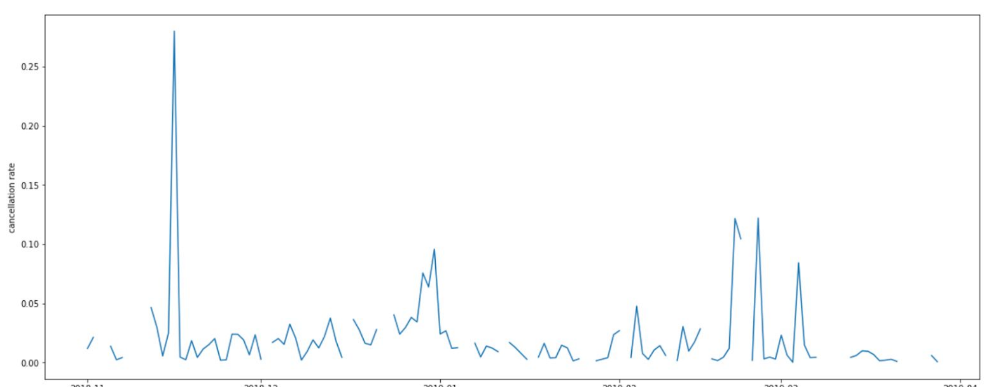
```
to
Salisbury Mills-Cornwall    7.990103
White House                 7.192601
Mountain View               6.504376
Lebanon                     6.300306
Tuxedo                      6.229354
North Branch                6.175628
Little Falls                6.086691
Aberdeen-Matawan            5.943596
Hazlet                      5.930178
Towaco                      5.854664
Name: delay_minutes, dtype: float64
```

The results show that the delay time does not have relation with origins and destinations.

6. Compare date with delays and cancellations to see when these delays and cancellations happen

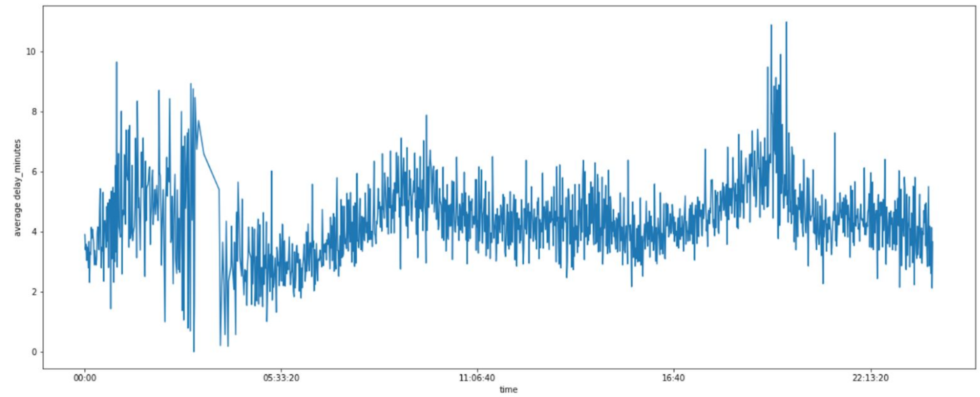


From the graph we can see that during 2018 Nov and Dec it has two times high delays.
And after our investigation, we get the reason is that because of the heavy snow.

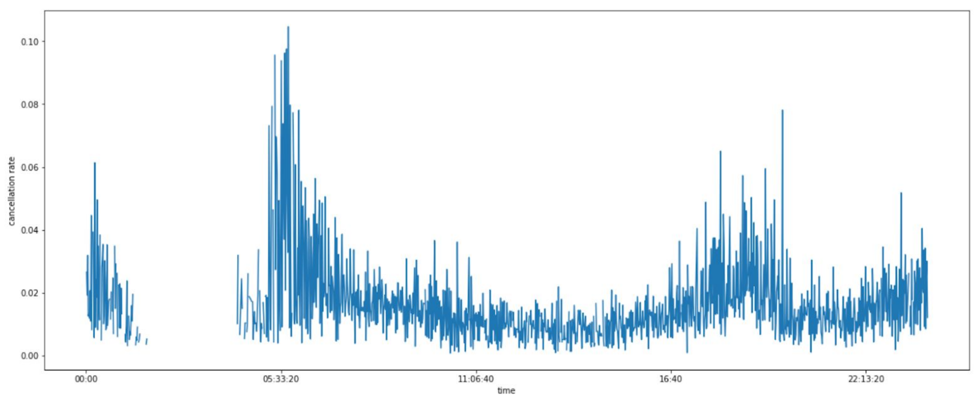


From the graph we can see that the cancellation rate is high in 2018 Nov and 2019 Feb.

7. Compare time with delays and cancellations to see when these delays and cancellations happen

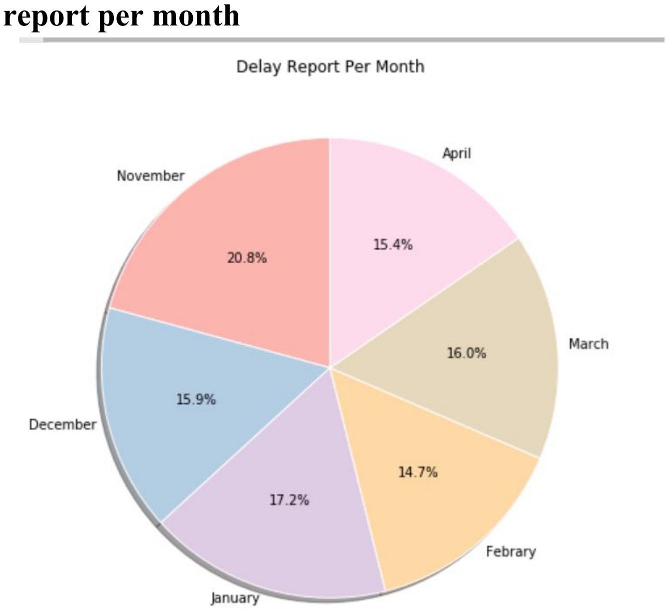


From the above we can see that the peak hour of delay is around 8pm and 1-3am.



From the above we can see that the peak hours of high cancellation is around 6am.

8. Delay report per month



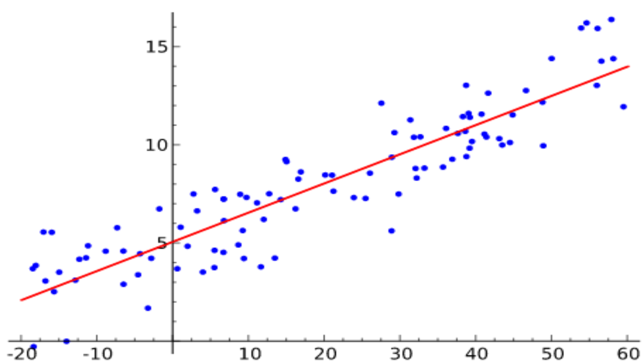
TIME SERIES ANALYSIS INTRODUCTION

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.

DELAY PREDICTION USING LINEAR REGRESSION MODEL

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).



Example of simple linear regression, which has one independent variable

1. Implement Linear Regression

The Linear Regression is always the first model to come up with when we need to predict some dataset. So in this case, first we need to analyze the data types and make sure it can be implemented as a linear regression model.

	date	train_id	stop_sequence	from	from_id	to	to_id	scheduled_time	actual_time	delay_minutes	status	line	type
0	11/1/18	3244	1	Long Branch	74	Long Branch	74	11/1/18 11:54	11/1/18 11:53	0.000000	departed	No Jersey Coast	NJ Transit
1	11/1/18	3244	2	Long Branch	74	Little Silver	73	11/1/18 12:02	11/1/18 12:02	0.316667	departed	No Jersey Coast	NJ Transit
2	11/1/18	3244	3	Little Silver	73	Red Bank	130	11/1/18 12:06	11/1/18 12:09	3.183333	departed	No Jersey Coast	NJ Transit
3	11/1/18	3244	4	Red Bank	130	Middletown NJ	85	11/1/18 12:12	11/1/18 12:13	1.300000	departed	No Jersey Coast	NJ Transit
4	11/1/18	3244	5	Middletown NJ	85	Hazlet	59	11/1/18 12:18	11/1/18 12:20	2.050000	departed	No Jersey Coast	NJ Transit

```
Data columns (total 10 columns):
train_id          1014624 non-null int64
stop_sequence     1014624 non-null int64
from_id           1014624 non-null int64
to_id             1014624 non-null int64
scheduled_time    1014624 non-null object
actual_time       1014624 non-null object
delay_minutes     1014624 non-null float64
line              1014624 non-null category
line_label        1014624 non-null int8
label             1014624 non-null float64
dtypes: category(1), float64(2), int64(4), int8(1), object(2)
```

So we are only using the essential columns required, which will be the features that will help us predict the outcome. Here the date time will help us in indexing the dataframe for ease of access, whereas, other columns will be predictors, that will allow us to forecast the future using the historical data we have until now.

Then we pick up data that can be used in this model. Meanwhile, Line_label is also be created to distinguish the different train names.

```
df[['line']] = df["line"].astype('category')
```

```
df['line_label'] = df["line"].cat.codes
```

```
df.groupby('line')['line_label'].unique()
```

```
line
Bergen Co. Line      [0]
Gladstone Branch     [1]
Main Line            [2]
Montclair-Boonton    [3]
Morristown Line      [4]
No Jersey Coast      [5]
Northeast Corrdr     [6]
Pascack Valley       [7]
Raritan Valley       [8]
Name: line_label, dtype: object
```

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.

```
X = np.array(df[['train_id','stop_sequence','from_id','to_id','line_label']])
X = preprocessing.scale(X)
```

Now comes the training and testing. 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.

```
X = np.array(df[['train_id','stop_sequence','from_id','to_id','line_label']])
X = preprocessing.scale(X)
```

```
y = np.array(df['delay_minutes'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

```
clf = LinearRegression()
results = clf.fit(X_train, y_train)
accuracy = clf.score(X_test, y_test)
print(accuracy) #means we can not use LR model to predict the delay!
```

```
0.0077642886537293565
```

Very bad score here, it means the dataset is definitely not linear predictable! Lets see how we can analyse the errors to understand this better.

```
output = pd.DataFrame()                        # Create a blank dataframe
output['delay_pred'] = delay_pred                # Add a column of predicted value of uber prices
output['actual'] = y_test                      # Add a column of the actual value of uber prices, we put them s
output['percent_linear_regression_error'] = abs(output['actual']-output['delay_pred'])*100/output['actual']
train_mean = np.mean(y_train)                   #Baseline prediction - is the average value of dependent variab
output['baseline_error'] = abs(output['actual']-train_mean)*100/output['actual']
output.head(n=50)
```

	delay_pred	actual	percent_linear_regression_error	baseline_error
0	5.042728	4.133333	22.001489	8.761335
1	4.883261	0.000000	inf	inf
2	4.826120	0.000000	inf	inf
3	4.925442	4.000000	23.136039	12.386713
4	5.266116	10.016667	47.426464	55.120115
5	4.825128	2.483333	94.300459	81.025578
6	5.459495	1.716667	218.028833	161.871953
7	4.826142	0.000000	inf	inf
8	4.096158	0.000000	inf	inf
9	4.625952	4.150000	11.468733	8.324543
10	4.023822	5.450000	26.168408	17.514339
11	4.513381	8.283333	45.512500	45.728750
12	4.157110	4.400000	5.520222	2.169739
13	4.415879	1.166667	278.503949	285.325874
14	5.359149	9.333333	42.580548	51.834266
15	3.659941	2.066667	77.093931	117.522671

We can see that the regression error is much less compared to the baseline error. It means we cannot simply use linear regression model to predict the value, we need to make some improvements.

2. Try to predict the future value

Since the prediction results do not have a strong connection with the coefficients in Linear Regression model. We tried to use Time Series to predict the future values.

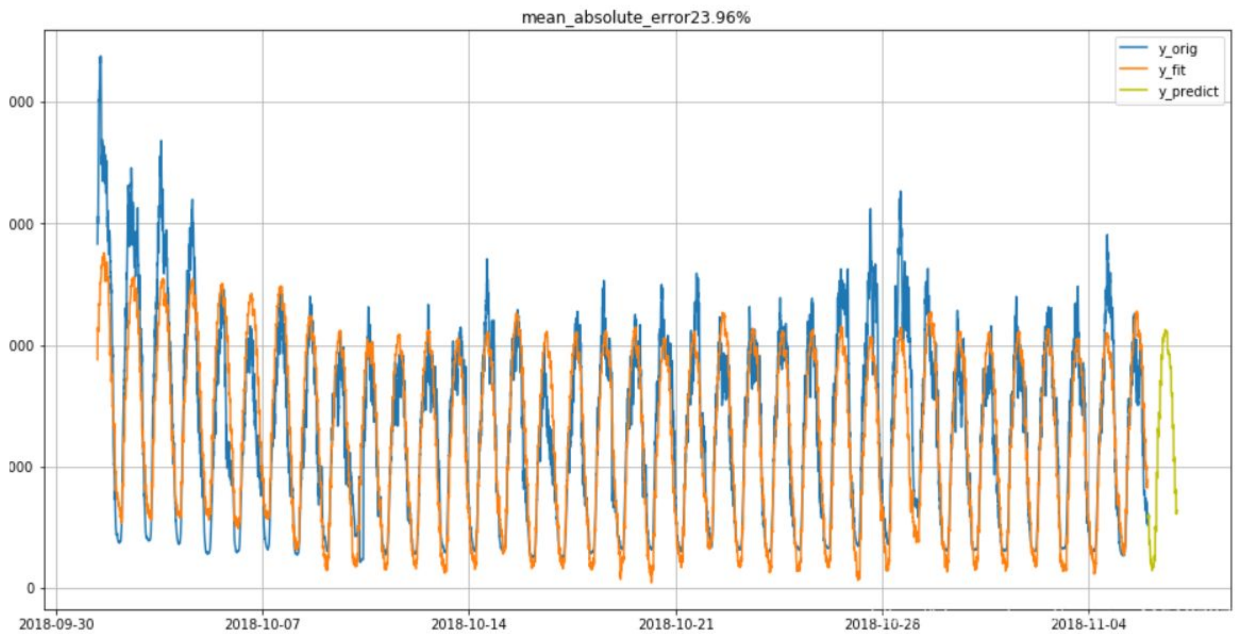
In short, a time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data.

And we consider the shift and time variation in this dataset.

day/hour/minute	day_avg/ hour_avg/ minute_agv
weekday/holiday	weekday_avg/ holiday_avg

```
# make feature
def build_feature(data, lag_start, lag_end, test_size, target_encoding=False, num_day_pred=1):
    # build future data with 0
    last_date = data["date"].max()
    pred_points = int(num_day_pred * 24)
    pred_date = pd.date_range(start=last_date, periods=pred_points + 1, freq="1h")
    pred_date = pred_date[pred_date > last_date]
    future_data = pd.DataFrame({"date": pred_date, "y": np.zeros(len(pred_date))})
    # concat future data and last data
    df = pd.concat([data, future_data])
    df.set_index("date", drop=True, inplace=True)
    #print(df)
    # make feature
    for i in range(lag_start, lag_end):
        df["lag_{}".format(i)] = df.y.shift(i)
    df["diff_lag_{}".format(lag_start)] = df["lag_{}".format(lag_start)].diff(1)
    df["hour"] = df.index.hour
    # df["day"] = df.index.day
    # df["month"] = df.index.month
    df["minute"] = df.index.minute
    df["weekday"] = df.index.weekday
    df["weekend"] = df.weekday.isin([5, 6]) * 1
    df["holiday"] = 0
    df.loc["2018-11-28 00:00:00":"2018-11-29 23:00:00","holiday"] = 1
    #print(df)
    # df["holiday"]
    # average feature
    if target_encoding:
        df["weekday_avg"] = list(map(cal_mean(df[:last_date], "weekday", "y").get, df.weekday))
        df["hour_avg"] = list(map(cal_mean(df[:last_date], "hour", "y").get, df.hour))
        df["weekend_avg"] = list(map(cal_mean(df[:last_date], "weekend", "y").get, df.weekend))
        df["minute_avg"] = list(map(cal_mean(df[:last_date], "minute", "y").get, df.minute))
        df = df.drop(["hour","minute","weekday", "weekend"], axis = 1)
    #df = pd.get_dummies(df, columns = ["hour", "minute", "weekday", "weekend"])
```

Use specific build_feature method to fit the time series data according to the different kinds of data. Then scale data by using this method and train the data in Linear Regression model. We can get the result of future prediction.



DELAY PREDICTION USING ARIMA

1. What is ARIMA Model

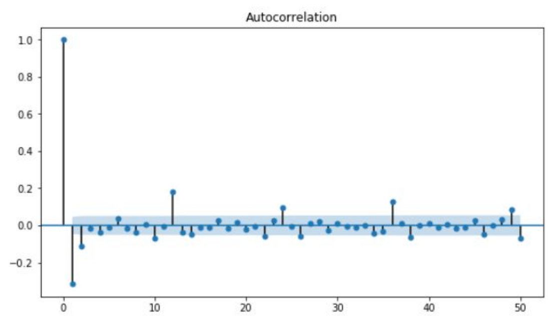
ARIMA stands for **Auto-Regressive Integrated Moving Averages**. The ARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation. The predictors depend on the parameters (p,d,q) of the ARIMA model:

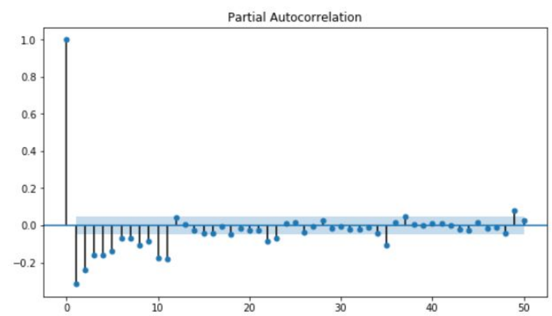
1. **Number of AR (Auto-Regressive) terms (p):** AR terms are just lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).
2. **Number of MA (Moving Average) terms (q):** MA terms are lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value.

3. **Number of Differences (d):** These are the number of nonseasonal differences, i.e. in this case we took the first order difference. So either we can pass that variable and put d=0 or pass the original variable and put d=1. Both will generate same results.

An importance concern here is how to determine the value of ‘p’ and ‘q’. We use two plots to determine these numbers. Let’s discuss them first.

1. **Autocorrelation Function (ACF):** It is a measure of the correlation between the the TS with a lagged version of itself. For instance at lag 5, ACF would compare series at time instant ‘t1’…’t2’ with series at instant ‘t1-5’…’t2-5’ (t1-5 and t2 being end points).
2. **Partial Autocorrelation Function (PACF):** This measures the correlation between the TS with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. Eg at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.





we use p=1, q=1 as we can see from acf and pacf above

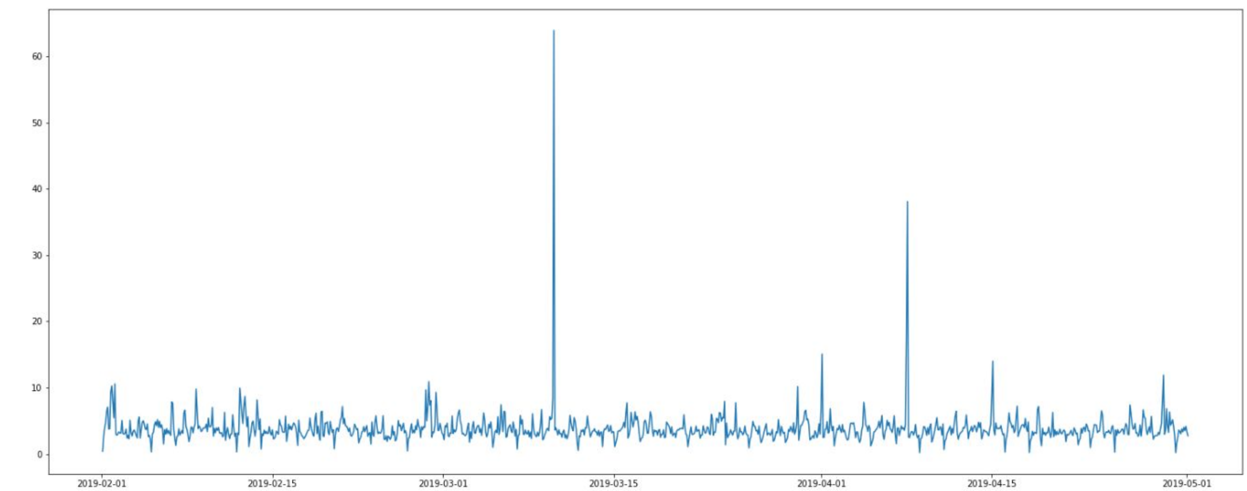
From these pictures above, we can see the data drop suddenly from the second position in both pictures. So we can get the value of p equals 1 and q equals one. This is the way to know p and q. After this step, we can use ARIMA model to do the prediction.

2.Make our data stationary

Although stationary assumption is taken in many TS model, most of the practical time series are not stationary. Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series. In this situation, we try to make the data as stationary as possible. Here are the steps to work on it.

2.1 Visualize the raw data

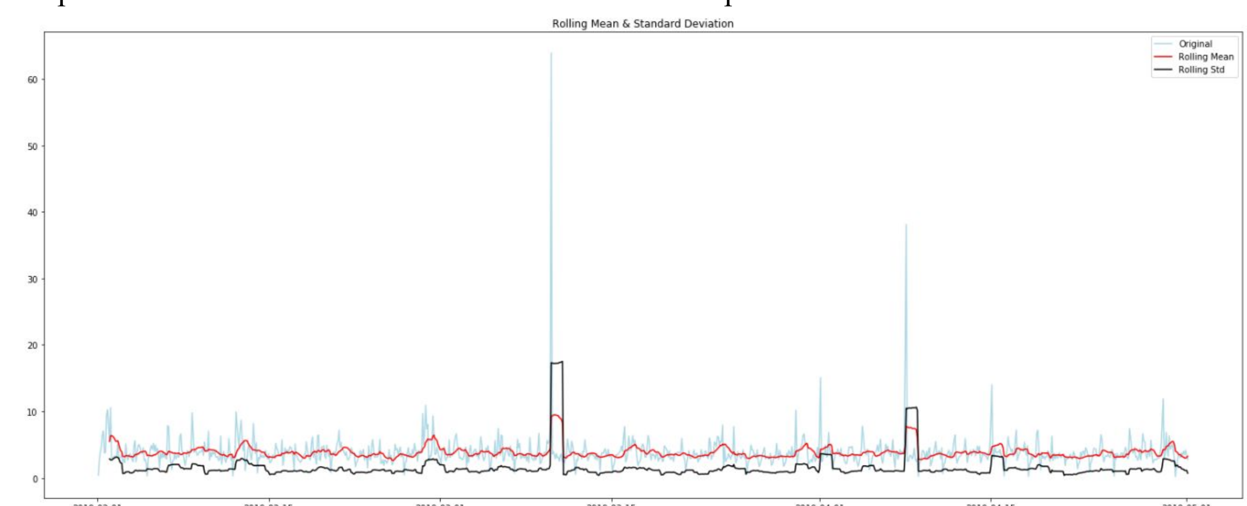
First, we plot the raw data and analyze visually. Due to the large amount of the dataset and the load of calculation is too large, we only pick up three month’s data. What’s more, we resample the delay time to the mean in each 2 hours.



There is some period fluctuation in the plot. But we cannot say it is stationary or not. Formally, we can check stationarity using the Dickey-Fuller Test.

2.2 Dickey-Fuller Test

Here the null hypothesis is that the TS is non-stationary. 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. We plotted standard deviation instead of variance to keep the unit similar to mean.



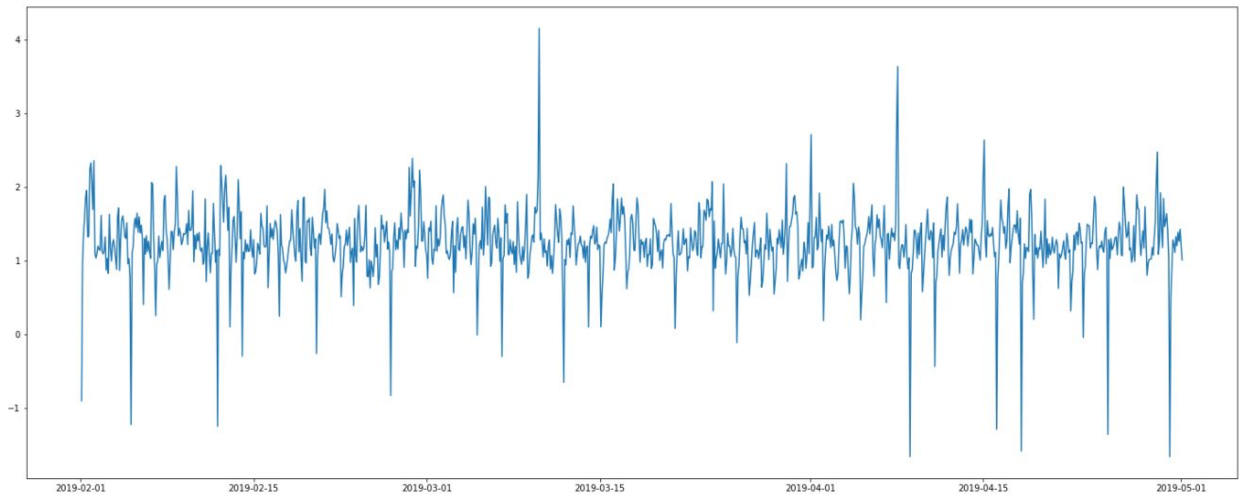

```
Results of Dickey-Fuller Test:
Test Statistic                  -26.116086
p-value                           0.000000
#Lags Used                        0.000000
Number of Observations Used    1068.000000
Critical Value (1%)              -3.436488
Critical Value (5%)              -2.864250
Critical Value (10%)             -2.568213
dtype: float64
```

We get the p value, test statistic and critical value. They shows us it reject the null hypothesis (H0), the data does not have a unit root and is stationary. the test statistic is less than Critical

Value (1%). It means we have a 99% confidence to say that the data is stationary. But we want our data be more stationary, we keep going to process the data.

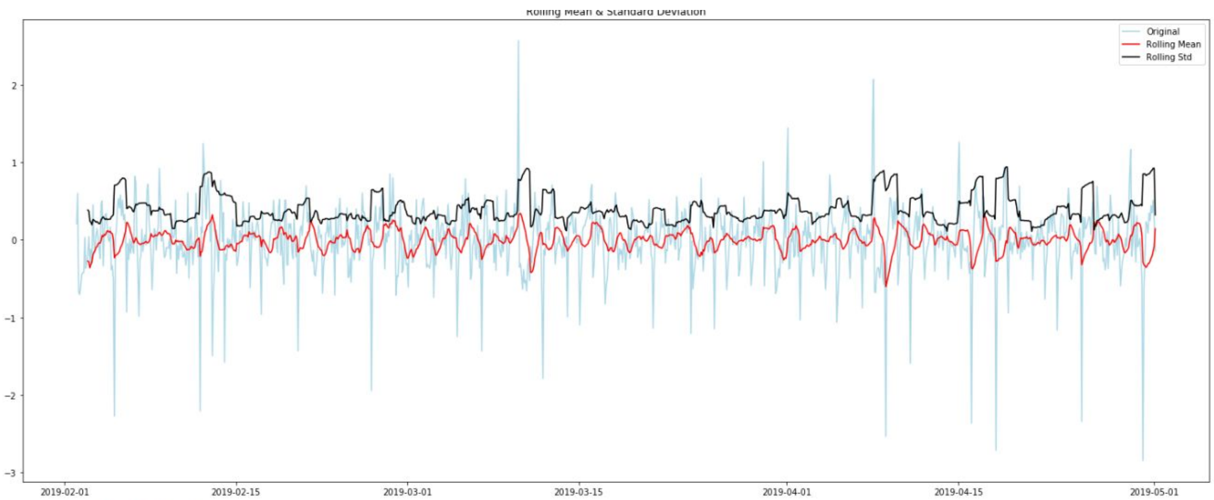
2.3 Transforming Time Series and Moving Average

We want to reduce the fluctuation range. So we can apply Log transformation which penalize higher values more than smaller values.



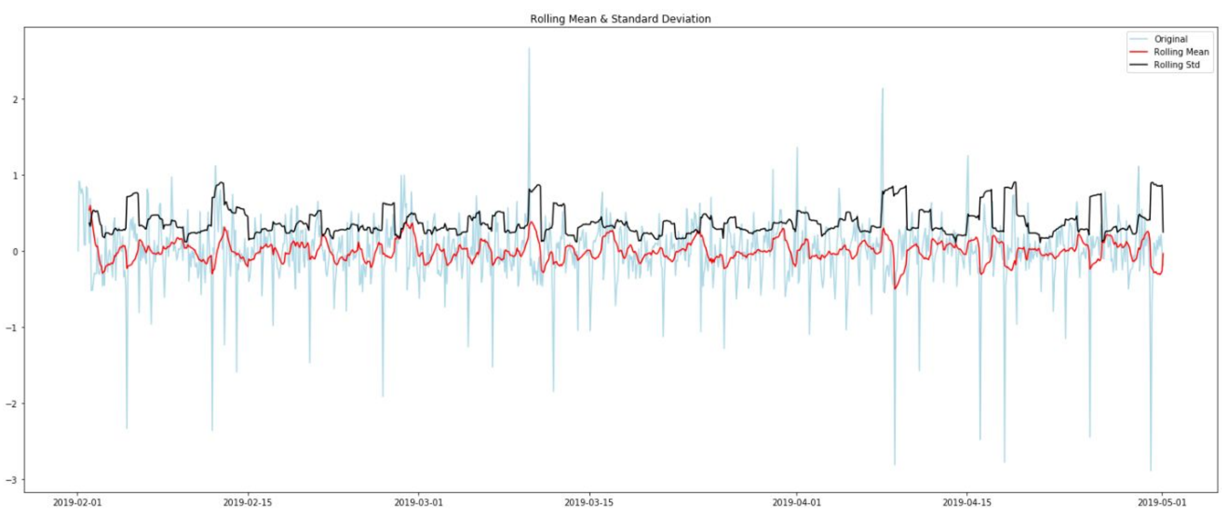
We use Moving Average Model to estimate the trend and remove it from our series. In this approach, we take average of ‘k’ consecutive values depending on the frequency of time series. As we take one day, so k is 12. Red line is the rolling mean. Since we are taking average of last 12 values, rolling mean is not defined for first 11 values. We need to drop Nan in series.

Let we see the stationary again.




```
Results of Dickey-Fuller Test:
Test Statistic                -1.215088e+01
p-value                        1.571369e-22
#Lags Used                     1.900000e+01
Number of Observations Used    1.038000e+03
Critical Value (1%)           -3.436666e+00
Critical Value (5%)           -2.864328e+00
Critical Value (10%)          -2.568255e+00
dtype: float64
```

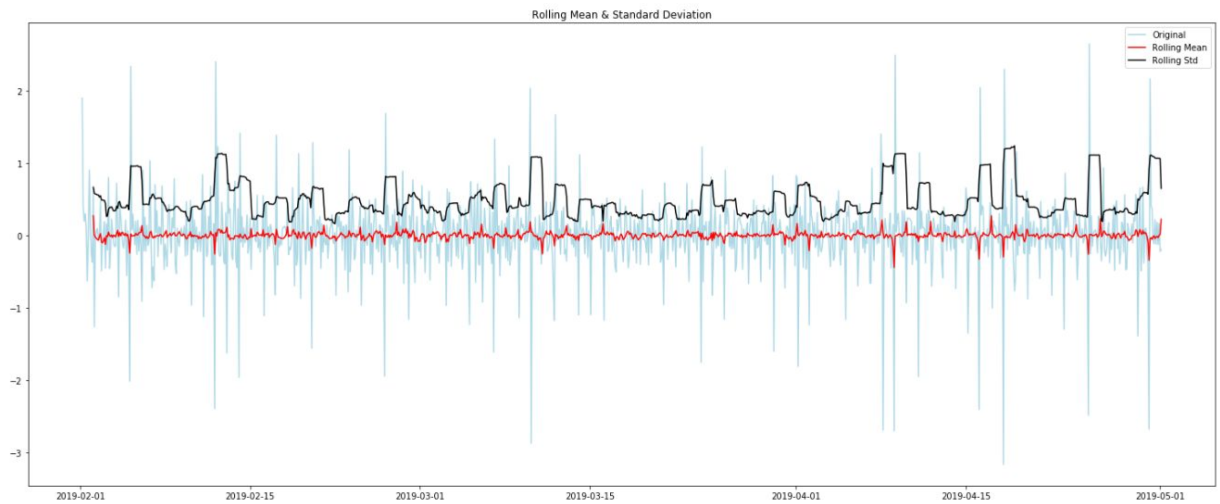
From the picture above, we can see the fluctuation range reduce and p value reduces a lot as well. This is just a simple moving average and we can take a Weighted Moving Average to improve it. In WMA model, it gives a higher weight for more recent value.




```
Results of Dickey-Fuller Test:
Test Statistic                -1.402241e+01
p-value                        3.551903e-26
#Lags Used                     7.000000e+00
Number of Observations Used    1.061000e+03
Critical Value (1%)           -3.436528e+00
Critical Value (5%)           -2.864268e+00
Critical Value (10%)          -2.568222e+00
dtype: float64
```

2.4 Difference

The simple trend reduction techniques discussed before don't work in all cases, particularly the ones with high seasonality. So we implement Difference in our time series.

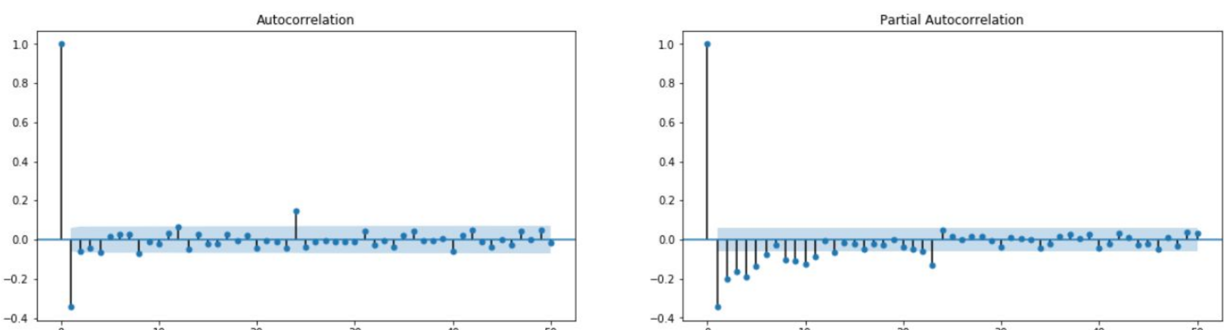



```
Results of Dickey-Fuller Test:
Test Statistic                -1.390684e+01
p-value                        5.602728e-26
#Lags Used                     2.200000e+01
Number of Observations Used    1.045000e+03
Critical Value (1%)           -3.436623e+00
Critical Value (5%)           -2.864310e+00
Critical Value (10%)          -2.568245e+00
```

The image of Rolling mean is almost a line near 0. The p value decreases significantly. This is the result we need.

3.Implementation

As we mentioned in the introduction of the ARIMA model. The parameter of p and q is important. We draw the picture of ACF and PACF to see the value of p and q.

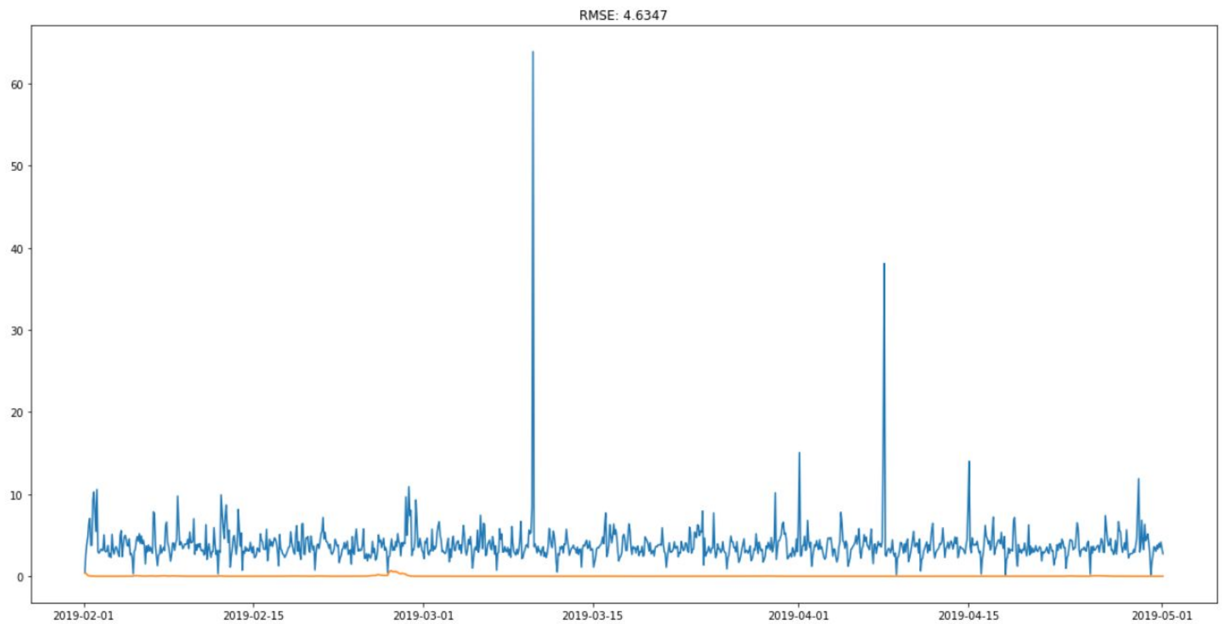


Then we get the value for both p and q are 1. We implement the time series in AR, MA and ARIMA.

```
print('ARIMA RSS: %.4f'% sum((ts_log_diff-results_ARIMA.fittedvalues)**2))
print('AR RSS: %.4f'% sum((results_AR.fittedvalues-ts_log_diff)**2))
print('MA RSS: %.4f'% sum((results_MA.fittedvalues-ts_log_diff)**2))
```

```
ARIMA RSS: 194.8127
AR RSS: 254.3815
MA RSS: 217.3101
```

We can see ARIMA model has the smallest number so that we use this model. We are going to take these values back to the original scale and do the prediction.



The RMSE has a small value which means our model has a good result of prediction.

FACEBOOK PROPHET PREDICTION REPORT

1. Introduction

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, seasonal, 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.

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. It also has advanced capabilities for modeling the effects of holidays on a time-series and implementing custom changepoints.

2. Modeling with Prophet

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, we 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.

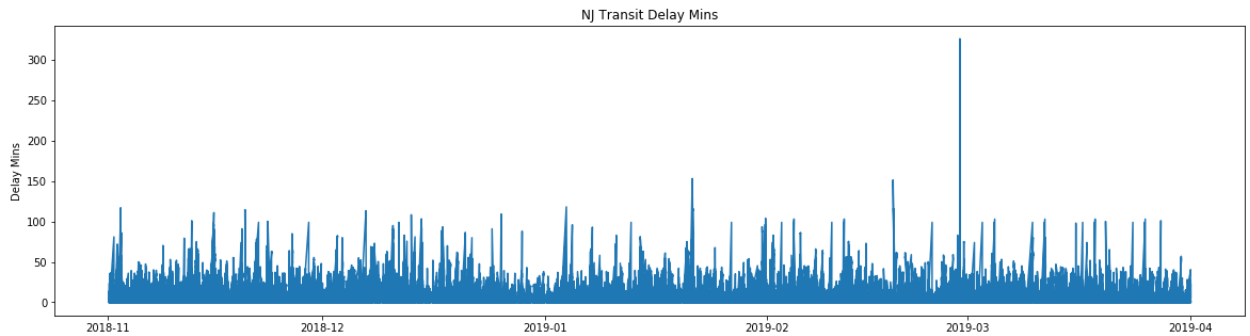
The higher the changepoint prior scale, the more flexible the model and the closer it fits to the training data. This may seem like exactly what we want but learning the training data too well can lead to overfitting and an inability to accurately make predictions on new data. We therefore need to find 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.

3. Basic Data Cleaning

We check the statistic information of the dataset and plot the data by setting 'scheduled_time' as index.

	stop_sequence	from_id	to_id	delay_minutes
count	1014624.0	1014624.0	1014624.0	1014624.0
mean	8.0	4481.0	4470.0	4.0
std	5.0	12077.0	12063.0	6.0
min	1.0	3.0	3.0	0.0
25%	4.0	62.0	61.0	1.0
50%	8.0	105.0	105.0	3.0
75%	12.0	138.0	138.0	5.0
max	26.0	43599.0	43599.0	326.0

The average delay minutes is 4 minutes and 75% is 5 minutes. It's strange that we have 326 minutes as our max value. So next, we draw a plot to have a deep look of this extreme value.



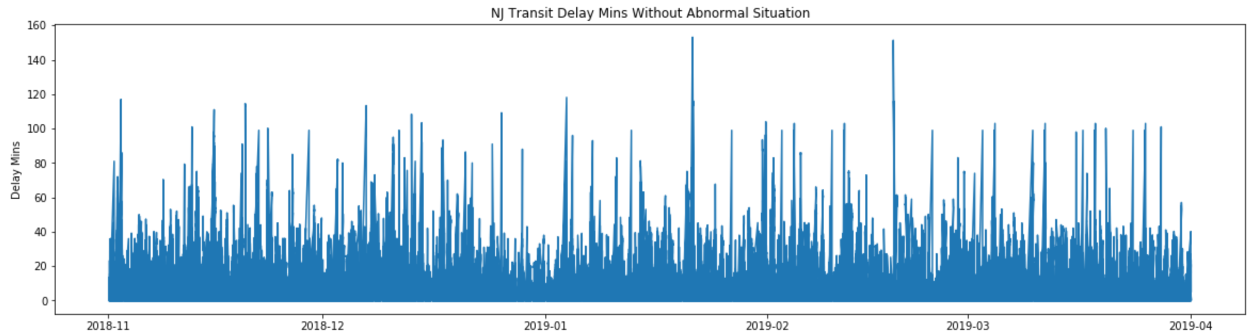
We can easily see that around the end of February, there is a high delay of NJ Transit. We then check this abnormal situation in detail.

scheduled_time	date	train_id	stop_sequence	from_id	to_id	scheduled_time	actual_time	delay_minutes
2019-02-27 21:19:00	2019-02-27	3284	2.0	74.0	73.0	2019-02-27 21:19:00	2019-02-28 02:44:00	325.0
2019-02-27 21:23:00	2019-02-27	3284	3.0	73.0	130.0	2019-02-27 21:23:00	2019-02-28 02:48:00	325.0
2019-02-27 21:28:00	2019-02-27	3284	4.0	130.0	85.0	2019-02-27 21:28:00	2019-02-28 02:54:00	326.0
2019-02-27 21:34:00	2019-02-27	3284	5.0	85.0	59.0	2019-02-27 21:34:00	2019-02-28 02:59:00	325.0
2019-02-27 21:38:00	2019-02-27	3284	6.0	59.0	37169.0	2019-02-27 21:38:00	2019-02-28 03:03:00	325.0
2019-02-27 21:47:00	2019-02-27	3284	7.0	37169.0	139.0	2019-02-27 21:47:00	2019-02-28 03:13:00	326.0

After checking, this delay started from Long Branch to South Amboy and the line is North Jersey Coast Line. The final station of this train is NEW YORK PENN STATION. We then find the reason of the delay. According to the Tweets from @NJTRANSIT on Feb 27, 2019, it said

Rail service in and out Penn Station New York is subject to up to 30-minute delays due to a disabled NJ TRANSIT train in one of the Hudson River Tunnels.
— NJ TRANSIT (@NJTRANSIT) February 27, 2019

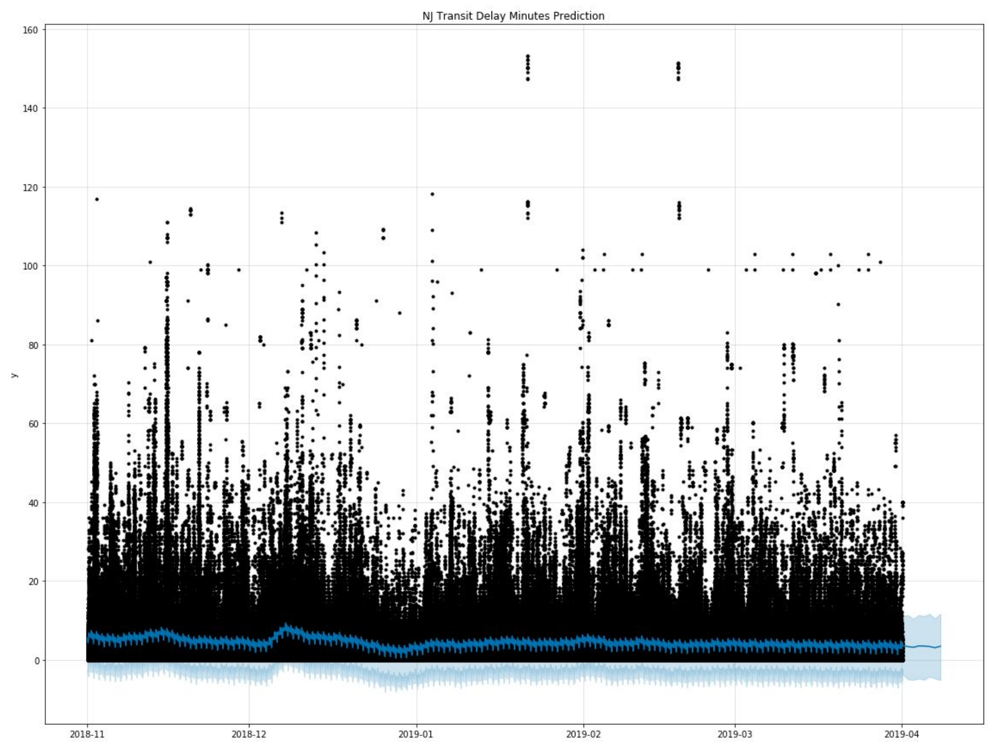
So, this delay was caused by a disabled train which is not happened frequently, we can remove these situations to have a better prediction. We extract the data except these extreme values as our new dataset and check the plot again.



We get a better data set which can represent normal NJ Transit this time. Next, we will apply Time Series Analysis with Facebook Prophet.

4. Making Future Data Frames

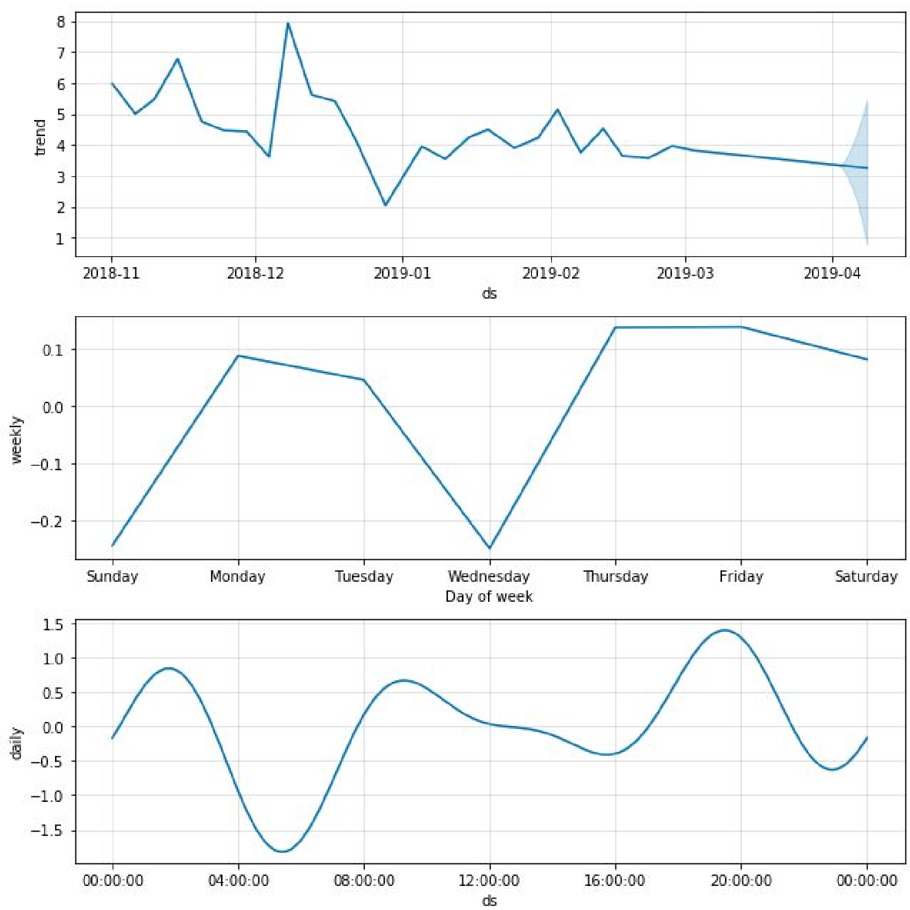
To make forecasts, we need to create what is called a future dataframe. We specify the number of future periods to predict (one week ahead which is 7 days) and the frequency of predictions (day). We then make predictions with the prophet model we created and the future dataframe. Our future dataframes contain the estimated minutes and hours 7 days into the future. We can visualize predictions with the prophet plot function.



This fig shows delay minutes estimates with one week forecast into the Future. The x-axis and y-axis represent time periods and delay minutes respectively.

5. Trends and Patterns

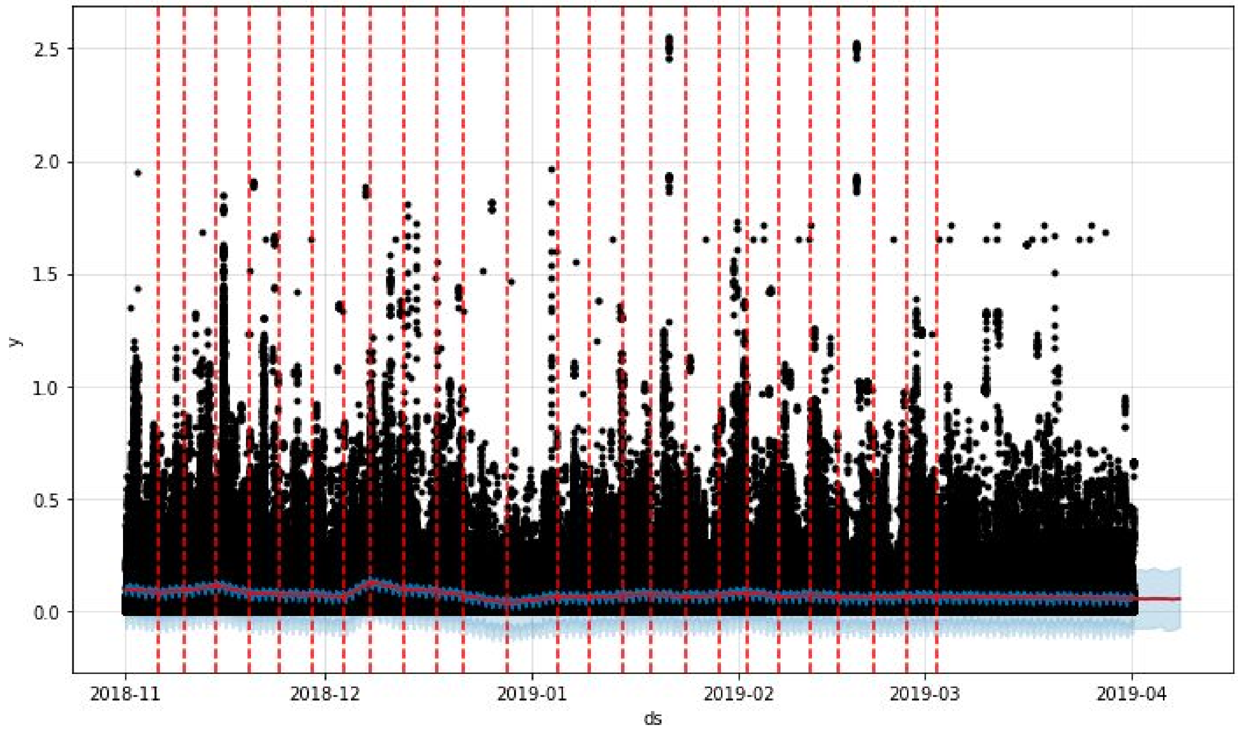
The last step of the analysis is to look at the overall trend and patterns. Prophet allows us to easily visualize the overall trend and the component patterns. We will split this plot into trend, yearly seasonality, and weekly seasonality of the time series.



According to the trend, the overall trend of delay minutes is decreasing which is good news. During a week, Sunday and Wednesday are great days to take NJ Transit because they have the least delay. On the opposite, trains on Thursday and Friday are more likely to be delayed.

During the day, time around 9 am and 7 pm are peak times because of work commuters and trains also got many delays around 2 am. This may be because there are fewer train staff during this time.

Prophet also detects changepoints by first specifying a large number of potential changepoints at which the rate is allowed to change. It then puts a sparse prior on the magnitudes of the rate changes (equivalent to L1 regularization) - this essentially means that Prophet has a large number of possible places where the rate can change but will use as few of them as possible.



6. Diagnostics

Prophet includes functionality for time series cross validation to measure forecast error using historical data. This is done by selecting cutoff points in history, and for each of them fitting the model using data only up to that cutoff point. We can then compare the forecasted values to the actual values.

This cross validation procedure can be done automatically for a range of historical cutoffs using the cross_validation function. We specify the forecast horizon (horizon), and then optionally the size of the initial training period (initial) and the spacing between cutoff dates (period). By default, the initial training period is set to three times the horizon, and cutoffs are made every half a horizon.

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2018-11-25 03:13:00	3.004663	-6.679809	12.765498	1.100000	2018-11-25 03:10:00
1	2018-11-25 03:22:00	2.836850	-7.195098	12.394706	0.000000	2018-11-25 03:10:00
2	2018-11-25 03:48:00	2.385678	-7.318251	12.243165	9.583333	2018-11-25 03:10:00
3	2018-11-25 03:54:00	2.294401	-7.659857	11.229298	3.583333	2018-11-25 03:10:00
4	2018-11-25 04:00:00	2.209808	-7.301374	11.838781	2.433333	2018-11-25 03:10:00

The output of cross_validation is a dataframe with the true values y and the out-of-sample forecast values yhat, at each simulated forecast date and for each cutoff date. In particular, a forecast is made for every observed point between cutoff and cutoff + horizon. This dataframe can then be used to compute error measures of yhat vs. y.

The metrics include mean squared error, MSE、 root mean squared error, RMSE、mean absolute error, MAE、mean absolute percent error, MAPE and the estimate coverage of yhat_lower and yhat_upper.

```
INFO:fbprophet:Skipping MAPE because y close to 0
```

	horizon	mse	rmse	mae	coverage
0	15:27:00	24.845259	4.984502	3.333910	0.941077
1	15:28:00	24.880525	4.988038	3.337846	0.940909
2	15:29:00	24.989466	4.998947	3.345184	0.940615
3	15:30:00	25.084558	5.008449	3.349188	0.940500
4	15:31:00	25.120285	5.012014	3.352801	0.940371

The estimate coverage of yhat_lower and yhat_upper is high which means our prediction covers most real data.

REFERENCE

[1]Prophet Python API: https://facebook.github.io/prophet/docs/quick_start.html#python-api

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

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

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

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

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