PREDICTING TRANSIT RIDERSHIP FOR A CITY

Capstone Project 1: Final Report Siri Surabathula

Problem Statement

As urban population levels rise, they lead to increasing traffic congestion, which in turn worsens mobility and carbon-emissions problems for cities. Public transit systems can be key in managing the urban mobility and emissions crises. However, the success of transit systems depends on their ability to scale and their responsiveness to regional variations in demand. Both short-term and long-term prediction models are hence vital to effectively solving problems of urban mobility.

This project aims at predicting transit ridership at different stations across a large city over the short-term using weather, traffic, taxi/cab and gas price data. The geo-spatial aspect of transit, taxi/cab and traffic data was taken into account while building the model for prediction.

Prospective clients for the model could be various city transportation authorities (like NYC DOT, NYC MTA) and city planning authorities (like City of New York)

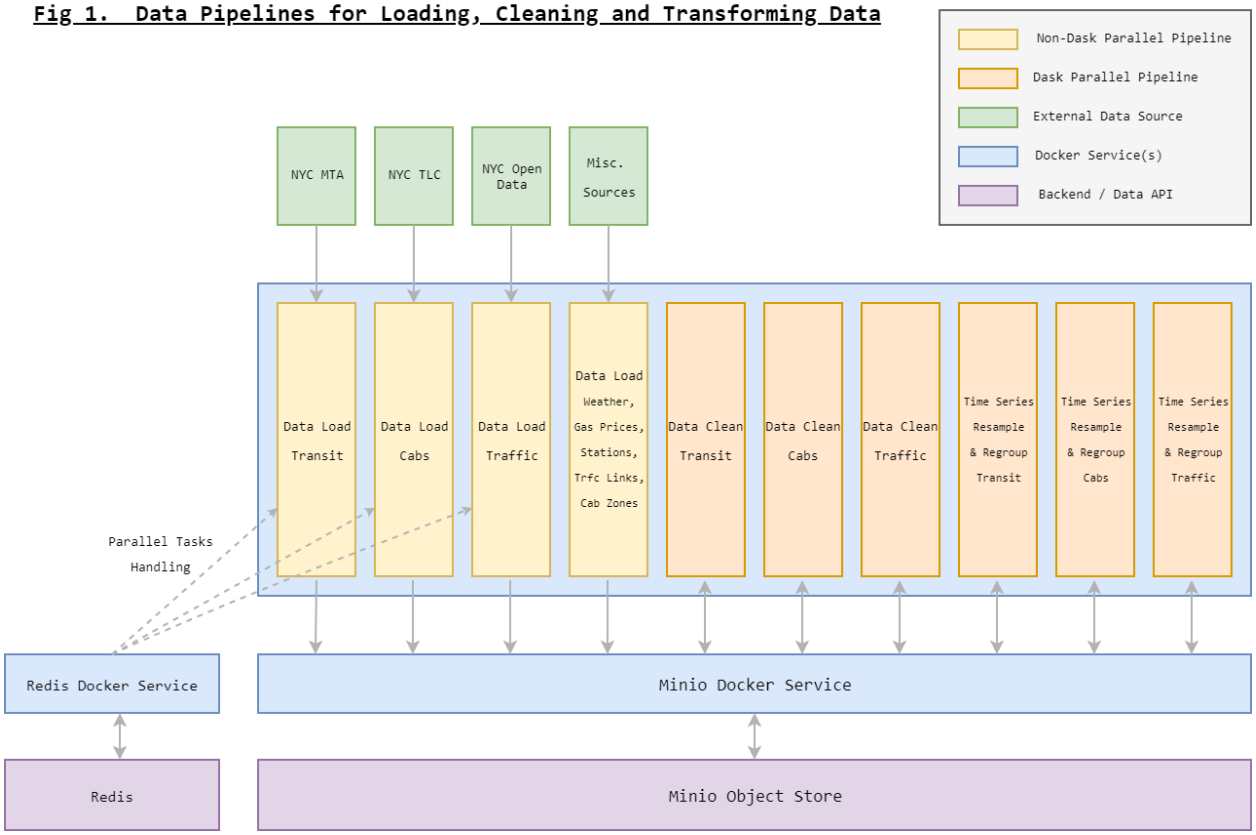
City planning authorities could use the ridership predictions to efficiently allocate resources and modulate the frequency of trains and buses along specific routes.

Description of Data-sets and Data Wrangling

Data Pipelines with Dask, Minio and Docker:

Considerable amount of time and effort was spent to clean and wrangle the data. Most of the datasets (like the NYC cabs data) were massive in size and called for parallel processing techniques. Data Pipelines were setup using Dask, Minio, Redis and Docker. The code-base for this setup is on github at sirisurab/transpred. Figure 1 below show a high-level diagram of the data pipelines.

Fig 1. Data Pipelines for Loading, Cleaning and Transforming Data



Dask:

The python API Dask was used for most of the data cleaning and transformation (time series resampling and group-by operations) for the transit, cabs and traffic data (each being 8GB or larger in size)

Dask is a parallel computing python library that provides

1. Dynamic task scheduling using either multiprocessor, multithreading or distributed/cluster architecture
2. Big-data collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments

For this project, Dask DataFrame was leveraged along with the Dask Distributed (using a mock 'cluster' of a scheduler and workers running on the local machine)

A Dask DataFrame is a large parallel DataFrame composed of many smaller Pandas DataFrames, split along the index. This allows for several pandas DataFrame methods to be carried out in parallel, leading to both quicker processing (on multiple cores) as well as eliminating the need to keep the entire dataframe in memory at the same time.

Both the quicker processing (using an 8 core machine) and the low memory usage were critical requirements for cleaning and processing cabs and traffic data which were each in excess of 12GB in size. Fig 1 above shows the Dask Data Pipelines (in orange) used for data cleaning and transformation.

Minio:

Minio is an object-storage server with an API similar to the Amazon S3 API. It provides a light-weight Python API and is compatible with a variety of back-end options including file systems. It is best suited for storing unstructured data.

For this project, Minio was used to convert the local file system into a transaction-safe object-store for the parallel data pipelines. Fig 1 above shows the Minio object-store (in purple).

Redis:

Redis is a light-weight in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists and sets among others.

For this project, Redis was used as a message broker for queuing scheduled tasks for those data loading pipelines which were not implemented in Dask and hence could not use the Dask Scheduler (Dask Scheduler is Dask's task scheduling mechanism)

Docker:

Docker is a container platform which provides the benefits of environment and platform virtualization using 'containers' and at the same time helps streamline the build and deploy processes. Docker swarm allows for running several docker services in a clustered manner, thus providing a suitable environment for parallel processing.

For this project, the build and deploy of services like Minio and Redis as well as multiple parallel instances of Dask processes and jupyter notebooks was managed with Docker running in swarm-mode. Fig 1 above shows the Docker services (in blue).

Geospatial Processing :

The transit, cabs and traffic data have a geospatial aspect associated with them. Transit data is collected for every MTA station in NYC. Each station has geographical coordinates (latitude and longitude) associated with it. Cab data is comprised of individual rides (each observation in the cab dataset is a ride). Each ride has a dropoff and pickup cab-zone associated with it (NYC is divided into several cab-zones and each zone is a polygon formed by connecting several geographical coordinates). Traffic data is comprised of traffic speed readings along specific 'links' (links are paths or lines connecting several geographic coordinates along NYC streets)
Since the goal of this project is to predict transit ridership for specific stations, there was a need to find a mechanism for associating cab and traffic data with transit data while taking into account the geospatial relationship between the station where the prediction was being made and the cab zones where the cab data was recorded as well as between the station and the traffic link where the traffic data was recorded. This was achieved by weighting each observation of the cab and traffic data w.r.t. to its relationship with every station where predictions were to be made. This is further illustrated below (assuming a specific station A):

1. Circles of radii (starting from 0.5 mile to 9.5 miles with increments of 0.5 mile) were drawn around station A. These circles represent 'circles of influence' or zones for station A and were used to find intersection with traffic and cab data.

2. For each observation in the cab data, the cab zone for the drop-off point was tested for intersection with the circles around station A. The smallest circle that intersected was picked and the cab ride observation was weighted with the inverse of the square of the radius of this circle. (i.e. number of passengers and ride distance were divided by the radius squared)
3. For each observation in the traffic data, the traffic link for the observation was tested for intersection with the circles around station A. The smallest circle that intersected was picked and the traffic observation was weighted with the inverse of the square of the radius of this circle. (i.e. traffic speed was divided by the radius squared)

Geopandas:

Geopandas is a python library for geospatial data processing. It extends the datatypes used by pandas to allow spatial operations on geometric types. Geometric operations are performed by shapely. Geopandas further depends on fiona for file access and descartes and matplotlib for plotting.

The intersections (described above) between station circles and cab zones as well as those between station circles and traffic links are complex and expensive and cannot be handled by pandas alone. Geopandas (which in turn uses shapely) provides efficient methods for performing these intersections.

Data Sets:

Transit Stations:

Source : NYC MTA (Subway Stations Data)

Description : Description of all the subway stations in NYC. Useful fields were 'STOP_NAME' (station name) and 'GTFS Latitude' and 'GTFS Longitude' (geographic coordinates of the station)

Processing : This dataset was processed by the python file refbase_loader.py and saved to minio bucket ref-base as a GIS shapefile stations.zip

Following data issues were addressed :

1. There was no unique identifier that represented stations across the NYC MTA database : The 'STATION' column of the MTA turnstile data-set was the only identifier for the station in that set. The contents of this column differed significantly from the 'STOP_NAME' column of the stations dataset. For example, the station named 'TIMES SQ-42 ST' in one set was represented as 'Times Sq - 42 St' in the other. Although, issues like this were easy to fix, there were a fair number of cases where a station named 'Astoria - Ditmars Blvd' did not have any obvious match in the other data set. A possible cause for cases like this was the use of alternate names for the same station ('Astoria - Ditmars Blvd' station was earlier known as 'Second Avenue'). Cases like this were hard, if not impossible to match. A python string-matching library called 'fuzzy-wuzzy' was used to find the best matches using 3 Levenshtein closeness methods (normal ratio, partial ratio and token sort ratio). The match

was accepted only if one of the three matching methods returned a ratio of 88% or higher. The matching station names from the turnstile dataset were added to the 'STATION' column of the stations dataset.

2. The columns 'GTFS Latitude' and 'GTFS Longitude' required further processing in order to be readily consumable for joins (geospatial) across different datasets (for example with the traffic and cab datasets) : The python geopandas library was leveraged for this purpose. 'GTFS Latitude' and 'GTFS Longitude' were merged into a single 'Point' geometry (shapely.geometry.Point) and the entire dataset converted to a geopandas GeoDataFrame. This allowed for fairly easy (though sometimes computationally expensive) joins across datasets using geometry attributes like points, lines and polygons. Circles of radii (starting from 0.5 mile to 9.5 miles with increments of 0.5 mile), were also drawn around each station and added to a new geometry columns containing the circles as polygons. These circles represent 'circles of influence' or zones for each station and were used to find intersection with traffic and cab data.

Transit Turnstile Data:

Source : NYC MTA (Subway Stations Turnstile Data : 4-hour frequency, 2016 and 2017)

Description : Transit ridership (turnstile entry and exit counts) of all the subway stations in NYC. Useful fields were 'STATION' (station name), 'DATE' (date) , 'TIME' (time) , 'ENTRIES' (entry count) , 'EXITS' (exit count)

Processing : This dataset was processed by the python file data_clean/tasks.py (using dask) and saved to minio bucket cl-transit in parquet format

Following data issues were addressed :

1. 'DATE' and 'TIME' occurred as separate string columns : These two were merged and converted to type 'datetime64[ns]'. This column was also used as the index (after the rest of cleaning was complete)
2. 'EXITS' (and 'ENTRIES') columns had cumulative reading of the turnstile unit : the pandas.Series.diff method was used to calculate the difference with the previous reading.
3. Turnstile units would reset randomly once in a while, resulting in outliers in the 'EXITS' (and 'ENTRIES') columns (abnormally high values or negative values) : These outliers were identified and filtered out by calculating the interquartile range and rejecting all rows with 'EXITS' (or 'ENTRIES') with values greater than 3 times the interquartile range or with negative values.

Cab Zones Data:

Source : NYC TLC (Taxi Zones Data)

Description : Shapefile containing polygons representing all the cab zones in NYC

Processing : This dataset was processed by the python file refbase_loader.py and saved to minio bucket ref-base as a GIS shapefile taxi_zones.zip

Following data issues were addressed :

1. The cab zone polygons were converted to shapely Polygons and the entire data-set was converted to a Geopandas DataFrame and saved back to GIS shapefile.

Cab Rides Data:

Source : NYC TLC (Taxi and Cab Trip Data : every taxi/cab trip in NYC for 2016 and 2017)

Description : Every taxi/cab trip in NYC. Useful fields were 'dropoff_datetime', 'dropoff_latitude', 'dropoff_longitude' , 'pickup_datetime', 'pickup_latitude', 'pickup_longitude', 'passenger_count'

Processing : This dataset was processed by the python file data_clean/tasks.py (using dask) and saved to minio bucket cl-gcabs and cl-ycabs in parquet format

Following data issues were addressed :

1. The dataset for 2016 and 2017 was too large and called for parallel processing techniques: The python API Dask was leveraged for this. Dask partitioned the large datasets into multiple pandas DataFrames and allowed for parallel processing on them.
2. Some records in this data-set had the columns 'dropoff_latitude', 'dropoff_longitude' (and 'pickup_latitude', 'pickup_longitude') which required further processing in order to be readily consumable for joins (geospatial) with the Stations dataset : The python geopandas library was leveraged for this purpose. 'dropoff_latitude', 'dropoff_longitude' were merged into a single 'Point' geometry (shapely.geometry.Point) and the entire dataset was converted to a geopandas GeoDataFrame. This data was then joined (geospatial join) with the Cab zone data to find the cab zone for each cab ride.
3. The remaining records in the data-set did not have dropoff and pickup coordinates and instead had the cab zone. These records did not require joining with the cab zone data
4. The entire data-set was then regrouped by day (frequency='1D') and cab-zone (cab-zone ID) using an aggregation function of ‘sum’, such that each observation now signified the total number of passengers dropped off on a specific day in a specific cab-zone.

Traffic Links Data:

Source : NYC Open data (Traffic Links Data)

Description : Traffic Links (sets of geographic coordinates) each representing a stretch or road/street over which the average speed of traffic was recorded. Useful fields were 'LINK_ID', 'LINK_POINTS', 'BOROUGH'

Processing : This dataset was processed by the python file refbase_loader.py and saved to minio bucket ref-base as a GIS shapefile traffic_links.zip

Following data issues were addressed :

1. The column 'LINK_POINTS' required further processing in order to be readily consumable for joins (geospatial) with the Stations dataset : The python geopandas library was leveraged for this purpose. The contents of 'LINK_POINTS' were merged into a single 'LineString' geometry (shapely.geometry.LineString) and the entire dataset converted to a geopandas GeoDataFrame. The dataset was then saved in GIS format.

Traffic Speed Data :

Source : NYC Open data (Traffic speed data recorded at various locations in NYC for 2016 and 2017)

Description : Traffic speed data recorded at various locations in NYC. Useful fields were 'LINK_ID', 'DATETIME', 'SPEED'

Processing : This dataset was processed by the python file data_clean/tasks.py (using dask) and saved to minio bucket dl-traffic in parquet format

Following processing was done :

1. The entire data-set was regrouped by day (frequency='1D') and traffic-link ID (link ID) using an aggregation function of 'mean', such that each observation now signified the average speed of traffic on a specific day along a specific traffic link.

Weather Data:

Source : National Climatic Data Center (daily temperature, rainfall and snowfall data for NYC for 2016 and 2017)

Description : Daily temperature, rainfall and snowfall data for NYC. Useful fields were 'DATE' ,'PRCP', 'SNOW', 'TMAX', 'TMIN'

Processing : This dataset was processed by the python file refbase_loader.py and saved to minio bucket ref-base as weather.csv

Following data issues were addressed :

1. 'DATE' in string format : This was converted to type 'datetime64[ns]'. This column was also used as the index (after the rest of cleaning was complete)
2. Average daily temperature : This was calculated by finding the mean of 'TMIN' and 'TMAX' and added as 'TAVG'

Gas Price Data:

Source : US Energy Information Administration (Weekly Retail Gasoline and Diesel Prices for NYC for 2016 and 2017)

Description : Weekly Retail Gasoline and Diesel Prices for NYC. Useful fields were 'DATE', 'PRICE'

Processing : This dataset was processed by the python file refbase_loader.py and saved to minio bucket ref-base as gas.csv

This dataset did not need further processing.

Exploratory Data Analysis

Time Series Analysis

The seasonality and trend for each time series was studied using time plots, Dickey-Fuller stationarity test, Autocorrelation and Partial Autocorrelation plots as well as frequency plots (histograms)

The test results and plots are at EDA_Modeling.ipynb (section EDA and Stationarity Tests)

and are further described below.

Transit Data

Station Exits :

Seasonality:

The Autocorrelation and Partial Autocorrelation plots (Figures 2b and 2c below) indicate that the data is non-stationary and has a lag of 7 days.

Results of Dickey-Fuller Test:

Test Statistic	-4.346754
p-value	0.000368
#Lags Used	20.00000
#Obs. Used	710.0000
Critical Value (1%)	-3.439594
Critical Value (5%)	-2.865619
Critical Value (10%)	-2.568942

Trend:

The time series plot (Figure 2a below) indicates that the data does not have a trend

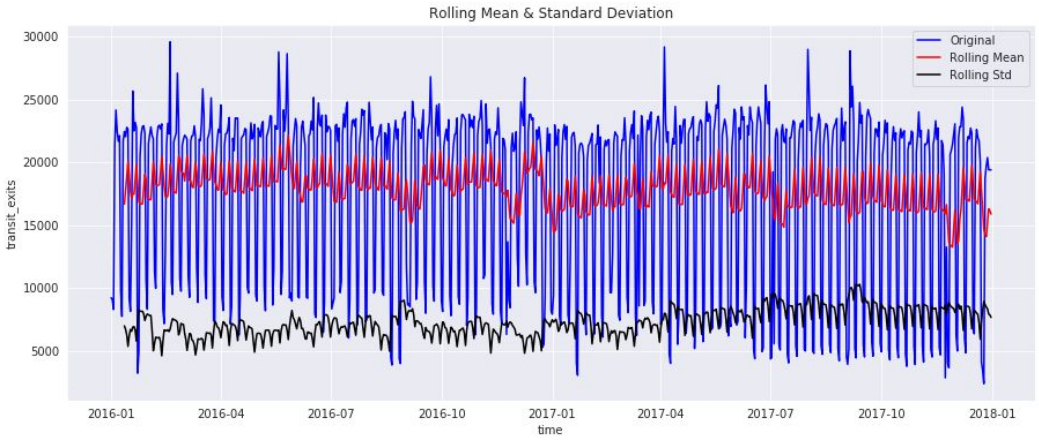


Fig 2a. Time Series plot of transit exits at the Wall Street station

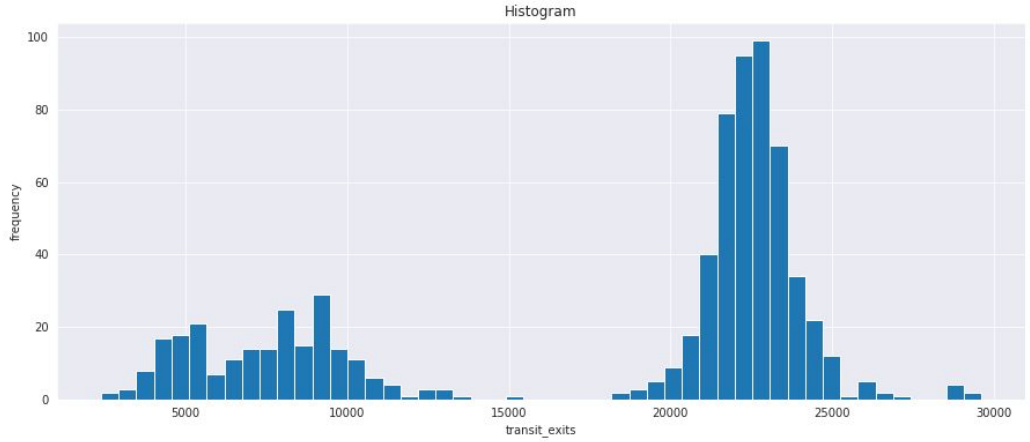


Fig 2b. Histogram of number of exits at the Wall Street station

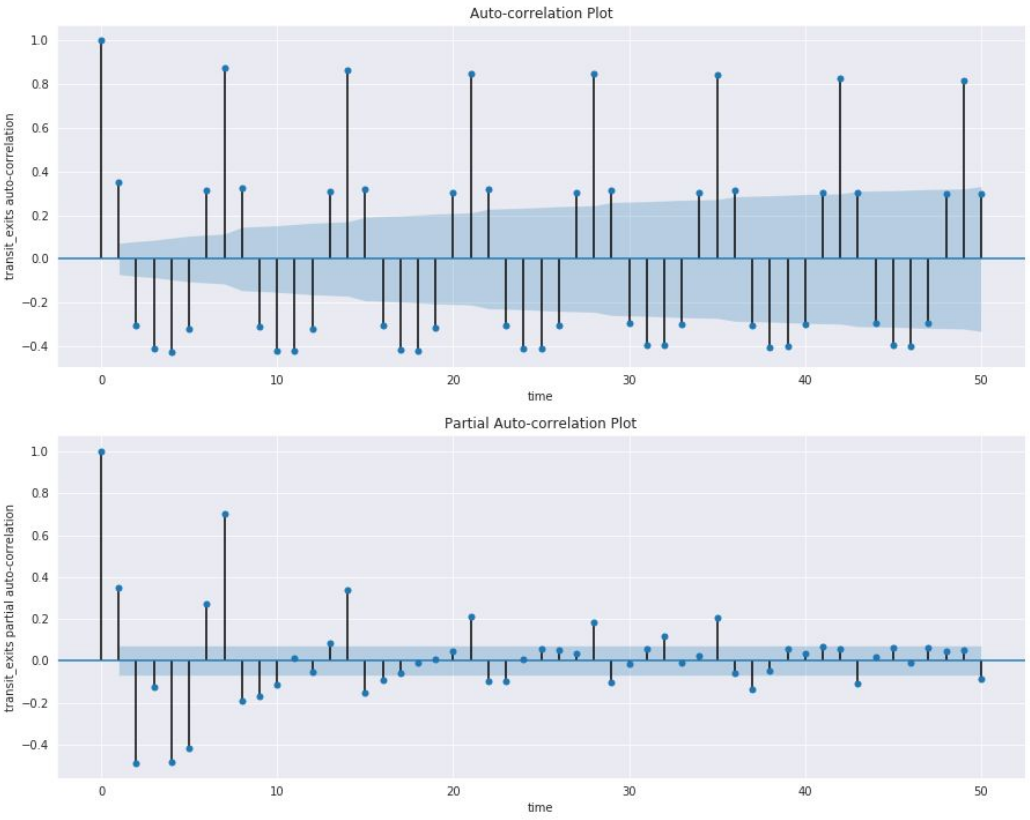


Fig 2c. Autocorrelation and Partial autocorrelation plots for number of exits at the Wall Street station

Station Entries :

Seasonality:

The Autocorrelation and Partial Autocorrelation plots (Figures 3b and 3c below) indicate that the data is non-stationary and has a lag of 7 days.

Results of Dickey-Fuller Test:

Test Statistic	-4.158597
p-value	0.000774
#Lags Used	20.00000
#Obs. Used	710.0000
Critical Value (1%)	-3.439594
Critical Value (5%)	-2.865619
Critical Value (10%)	-2.568942

Trend:

The time series plot (Figure 3a below) indicates that the data does not have a trend.

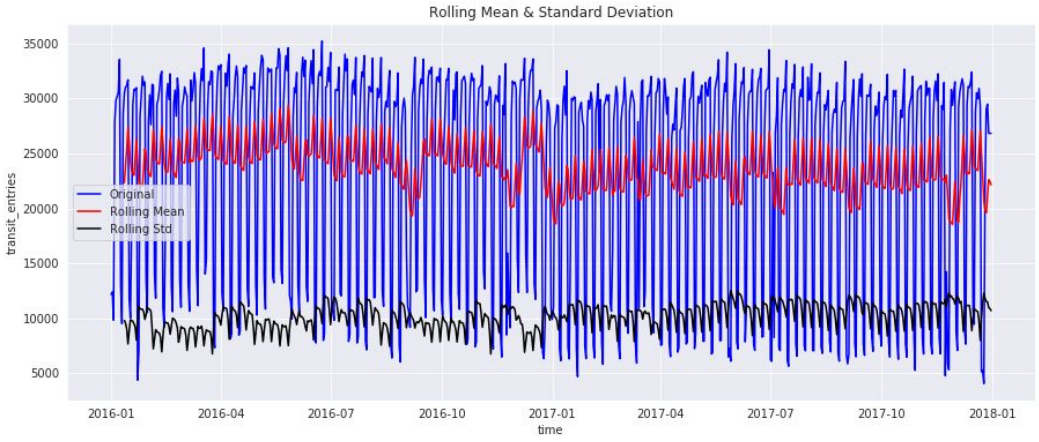


Fig 3a. Time Series plot of transit entries at the Wall Street station

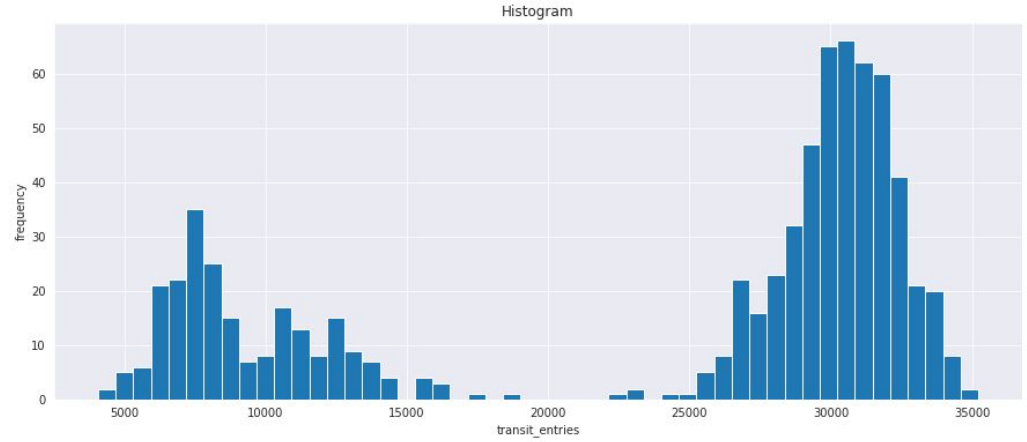


Fig 3b. Histogram of number of entries at the Wall Street station

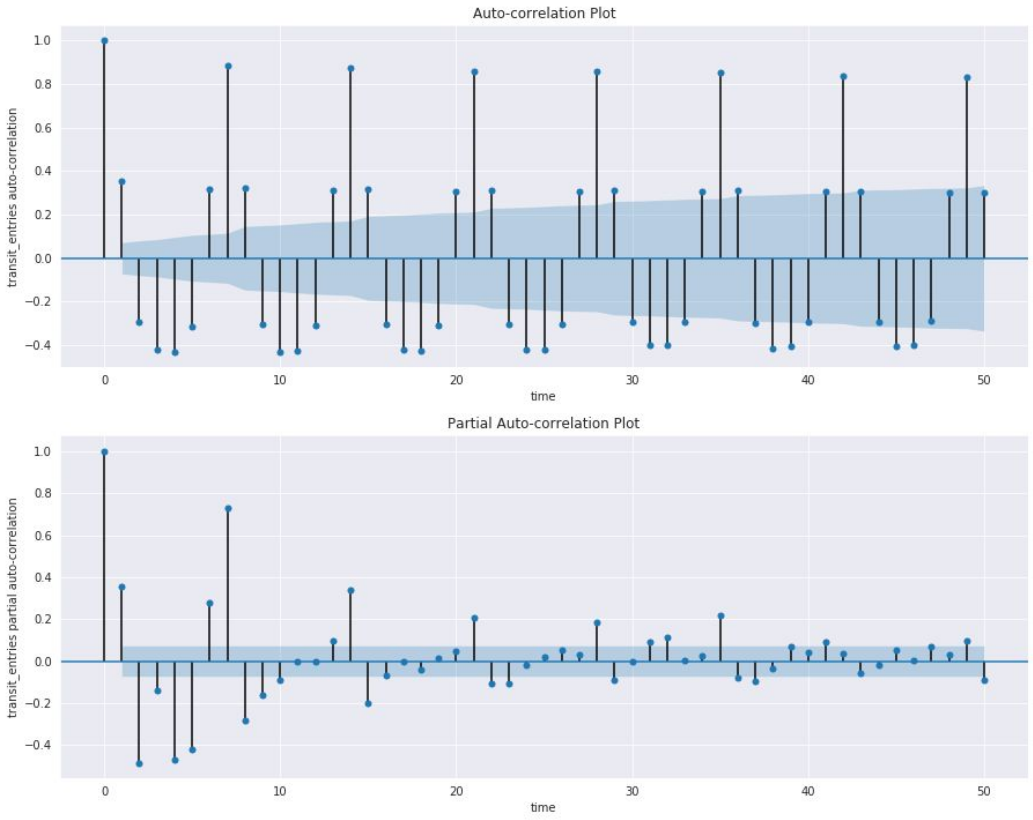


Fig 3c. Autocorrelation and Partial autocorrelation plots for number of entries at the Wall Street station

Taxi/Cab Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots (Figures 4b and 4c below) indicate that the data is non-stationary and with lags of 7 days and 1 day.

Results of Dickey-Fuller Test:

Test Statistic	-2.457215
p-value	0.126234
#Lags Used	20.00000
#Obs. Used	710.0000
Critical Value (1%)	-3.439594
Critical Value (5%)	-2.865619
Critical Value (10%)	-2.568942

Trend:

The time series plot (Figure 4a below) indicates that the data has a varying trend, where it decreases from the January to September of 2016; after which it rises, then drops abruptly in January 2017 and again rises until March 2017. It exhibits a downward trend again until September 2016, after which it rises once again. It appears that the trend itself has a seasonality of 1 year.

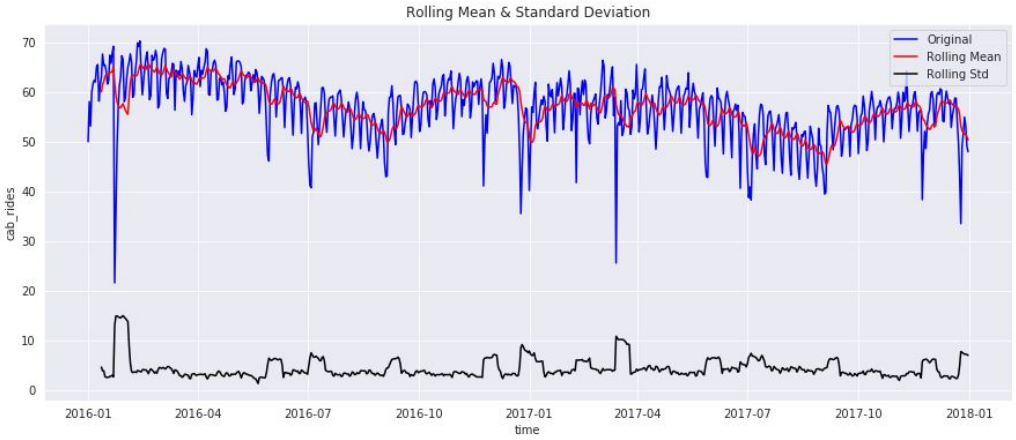


Fig 4a. Time Series plot of cab rides in the vicinity of the Wall Street station

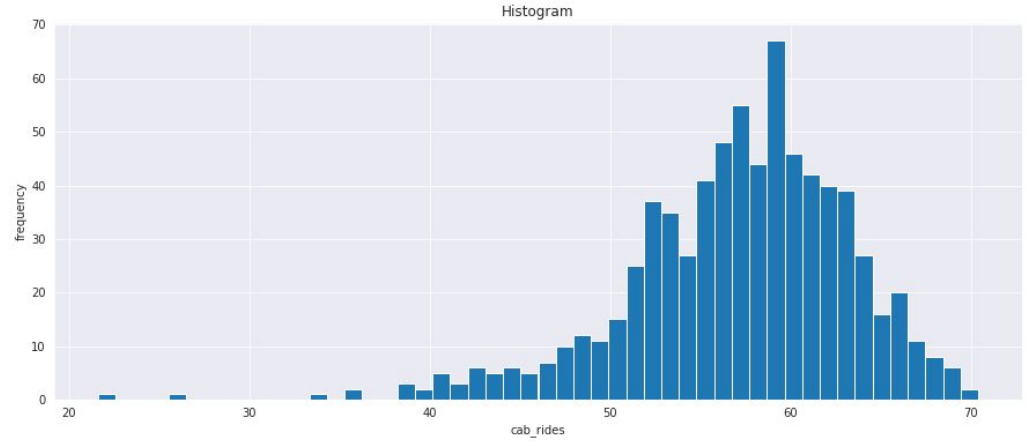


Fig 4b. Histogram of number of cab rides in the vicinity of the Wall Street station

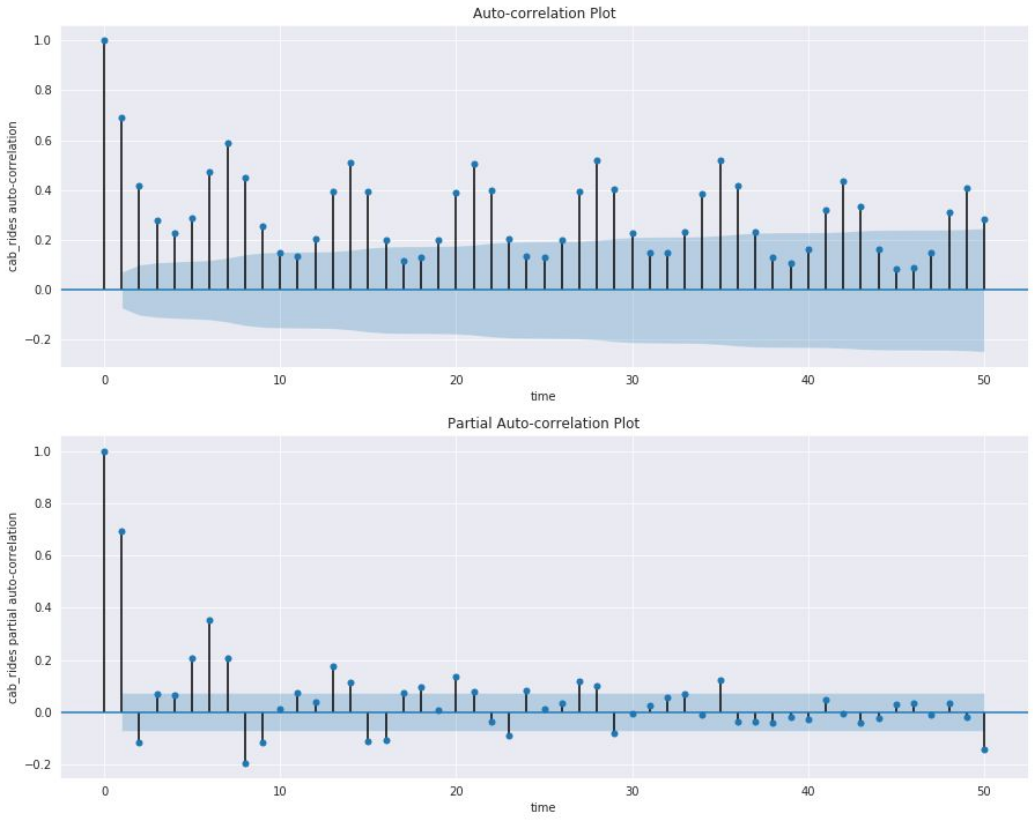


Fig 4c. Autocorrelation and Partial autocorrelation plots for number of cab rides in the vicinity of the Wall Street station

Traffic Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots (Figures 5b and 5c below) indicate that the data is non-stationary and has a lag of 1 day.

Results of Dickey-Fuller Test:

Test Statistic	-2.204295
p-value	0.204690
#Lags Used	13.00000
#Obs.s Used	717.0000
Critical Value (1%)	-3.439503
Critical Value (5%)	-2.865579
Critical Value (10%)	-2.568921

Trend:

The time series plot (Figure 5a below) indicates that the data has a general downward trend from January 2016 to October 2017, after which it begins to exhibit an upward trend.

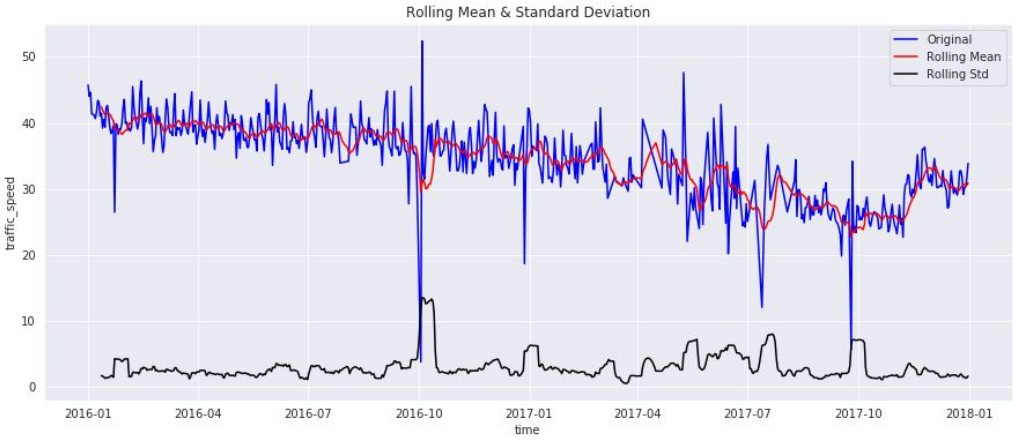


Fig 5a. Time Series plot of traffic speed in the vicinity of the Wall Street station

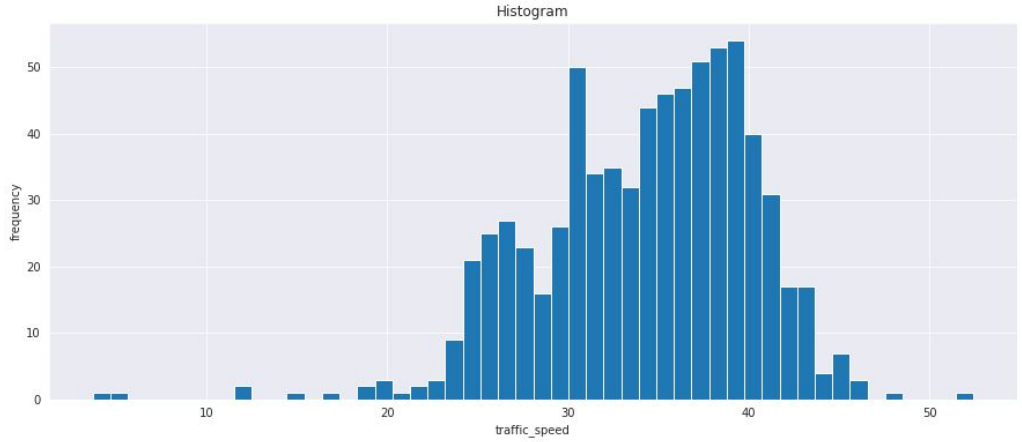


Fig 5b. Histogram of traffic speed in the vicinity of the Wall Street station

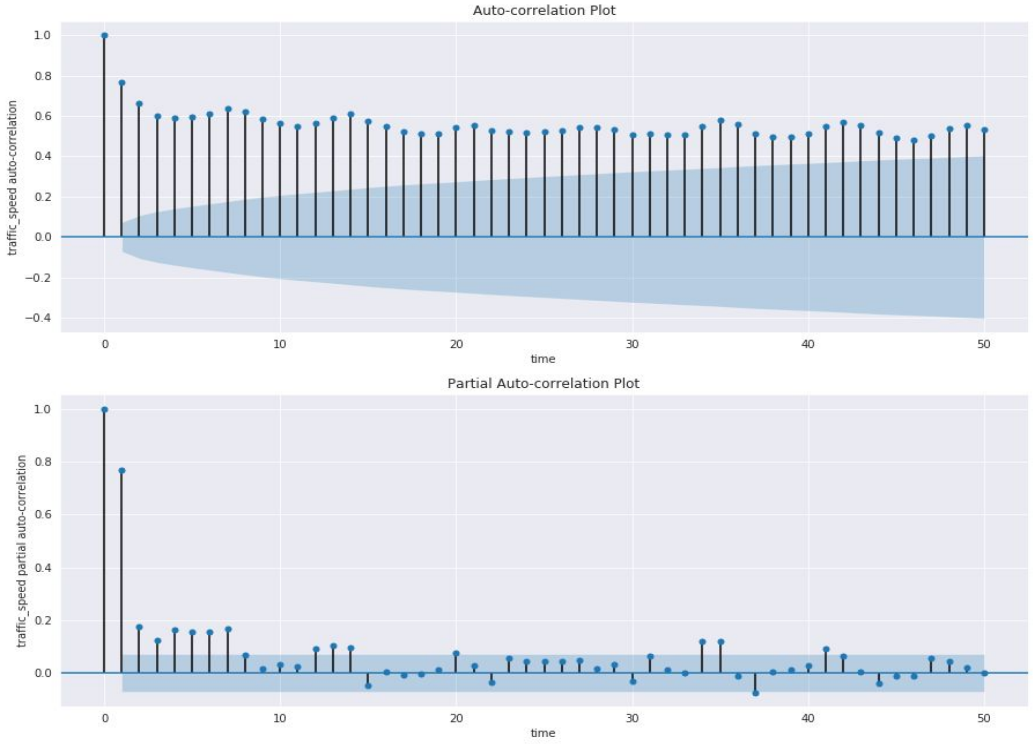


Fig 5c. Autocorrelation and Partial autocorrelation plots for traffic speed in the vicinity of the Wall Street station

Gas Price Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots (Figures 6c and 6d below) indicate that the data has significant autocorrelation at lag 1.

Results of Dickey-Fuller Test:

Test Statistic	-1.954157
p-value	0.307028
#Lags Used	15.00000
#Obs. Used	715.0000
Critical Value (1%)	-3.439529
Critical Value (5%)	-2.865591
Critical Value (10%)	-2.568927

Trend:

The time series plots (Figure 6a and 6b below) for the un-differenced and differenced price (first order difference with lag=1) shows a local level with significant variance and an initially decreasing trend (slope) which stabilizes after July 2016.

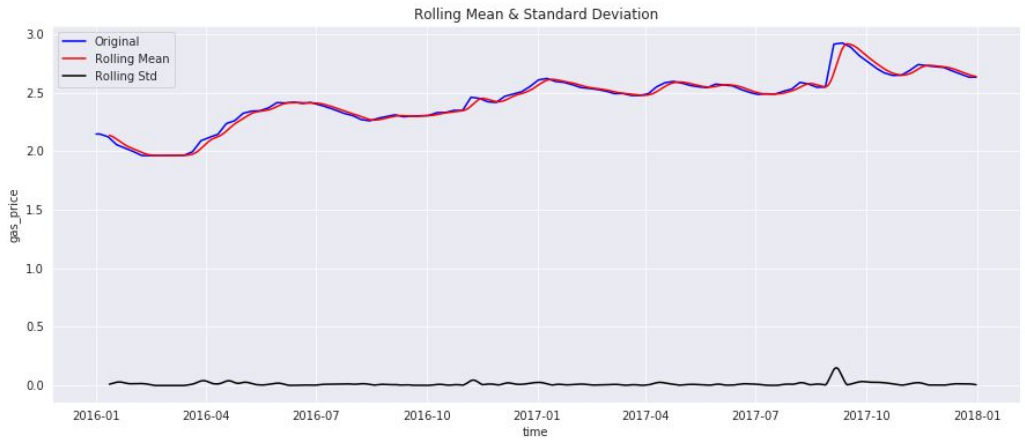


Fig 6a. Time Series plot of gas price for NYC

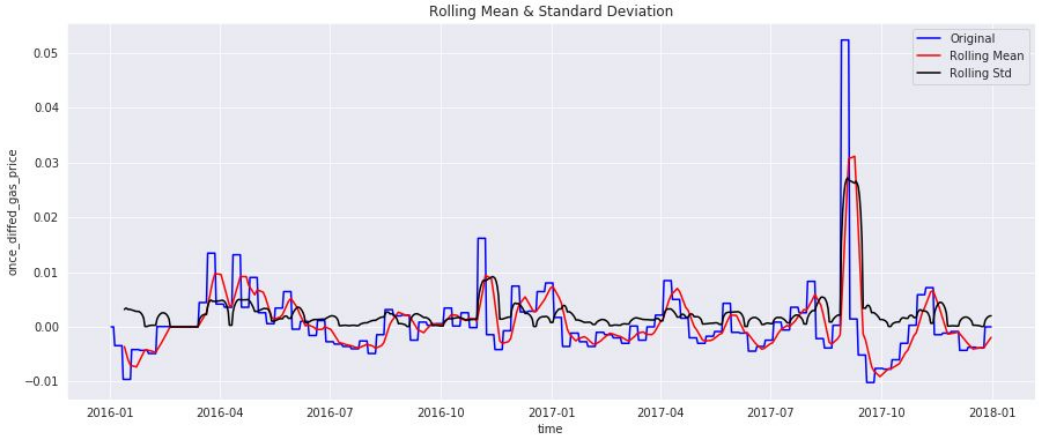


Fig 6b. Time Series plot of once-differenced gas price for NYC

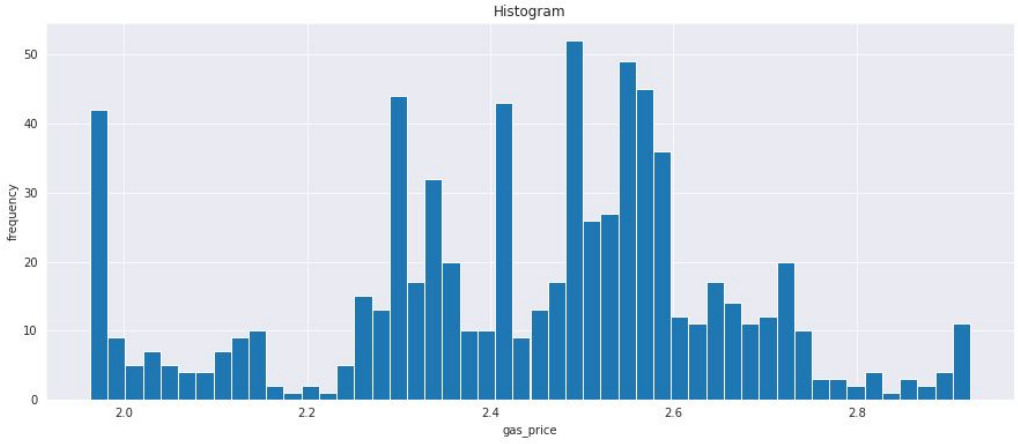


Fig 6c. Histogram of gas price in NYC

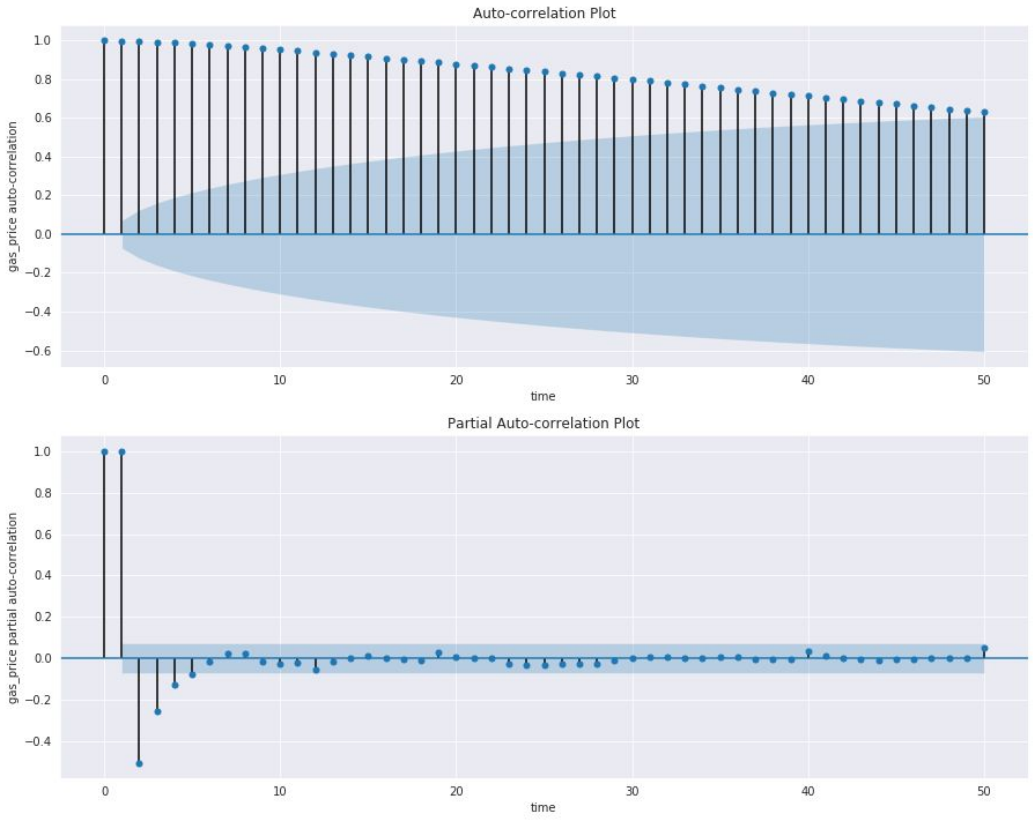


Fig 6d. Autocorrelation and Partial autocorrelation plots for gas price in NYC

Weather Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots (Figures 7b and 7c below) indicate that the data has significant autocorrelation at lag 1 (positive correlation) and has an obvious seasonality of 365 days (annual)

Results of Dickey-Fuller Test:

Test Statistic	-1.220396
p-value	0.664761
#Lags Used	14.00000
#Obs. Used	716.0000
Critical Value (1%)	-3.439516
Critical Value (5%)	-2.865585
Critical Value (10%)	-2.568924

Trend:

The time series plot (Figure 7a below) shows a local level with significant variance and annual seasonality as well as a very gentle overall decreasing trend (slope)

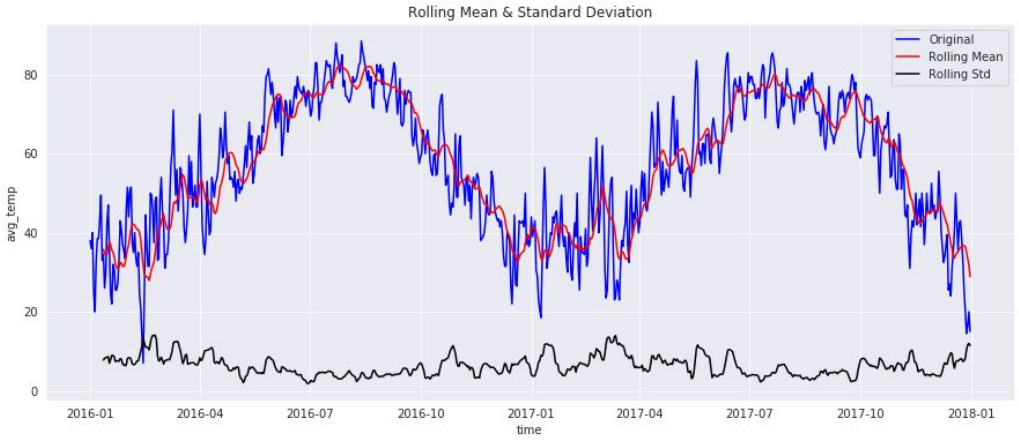


Fig 7a. Time Series plot of avg. temperature for NYC

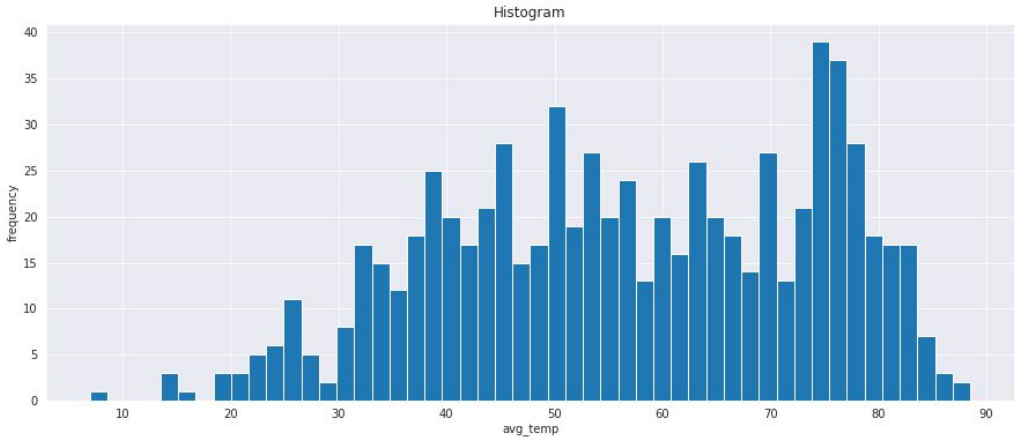


Fig 7b. Histogram of avg. temperature in NYC

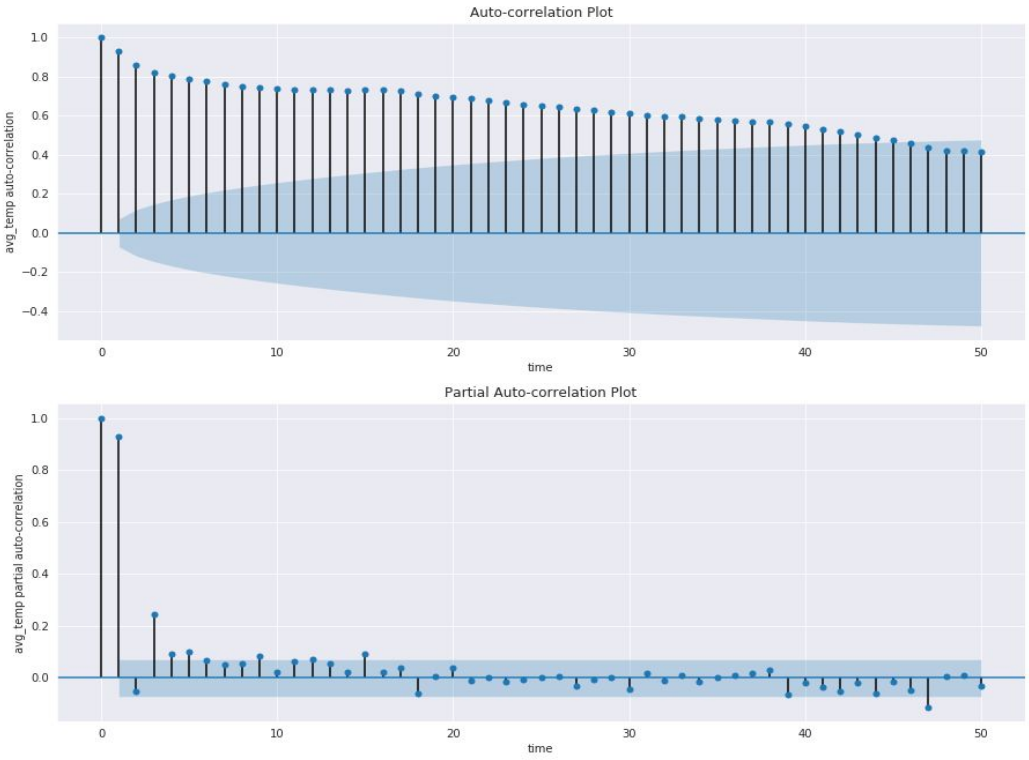


Fig 7c. Autocorrelation and Partial autocorrelation plots for avg. temperature in NYC

Studying Variable Relationships

Relationships between the various exogenous variables (cab rides, traffic speed, gas price and mean daily temperature) with the endogenous variables (transit station exit and entry counts) were studied. This study ignored the time component for all the series.

The plots are at EDA_nonts.ipynb (section EDA for each of the 5 selected stations)

and are further described below.

Transit counts vs Cab-rides:

For most stations (Fig 8a below) , there is a negative correlation between transit station (exit and entry) counts and the number of cab rides in the vicinity of the station. For the Wall Street (Fig 8b below) and Court Square stations there is a positive correlation between transit entry counts and cab rides in the vicinity of the station.

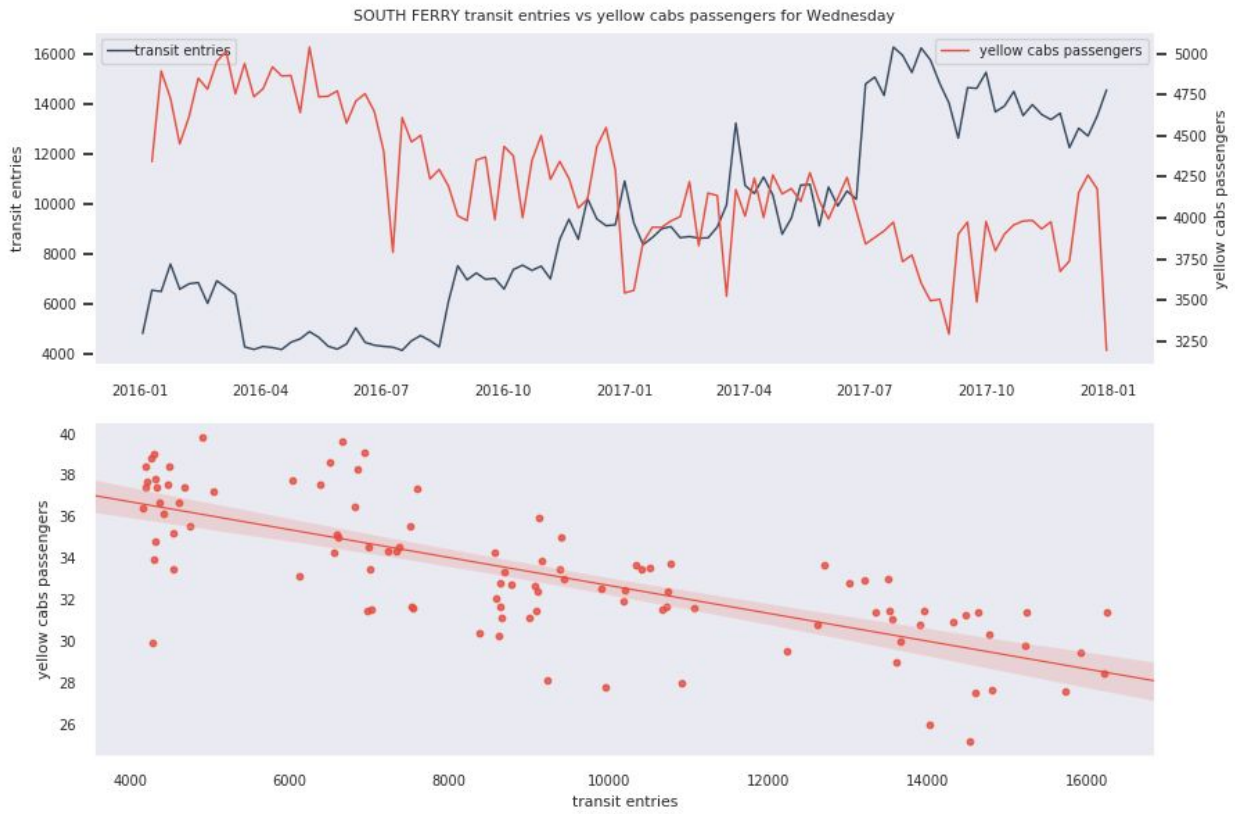


Fig 8a. Relationship between Transit Entries and Cab Rides for South Ferry Station

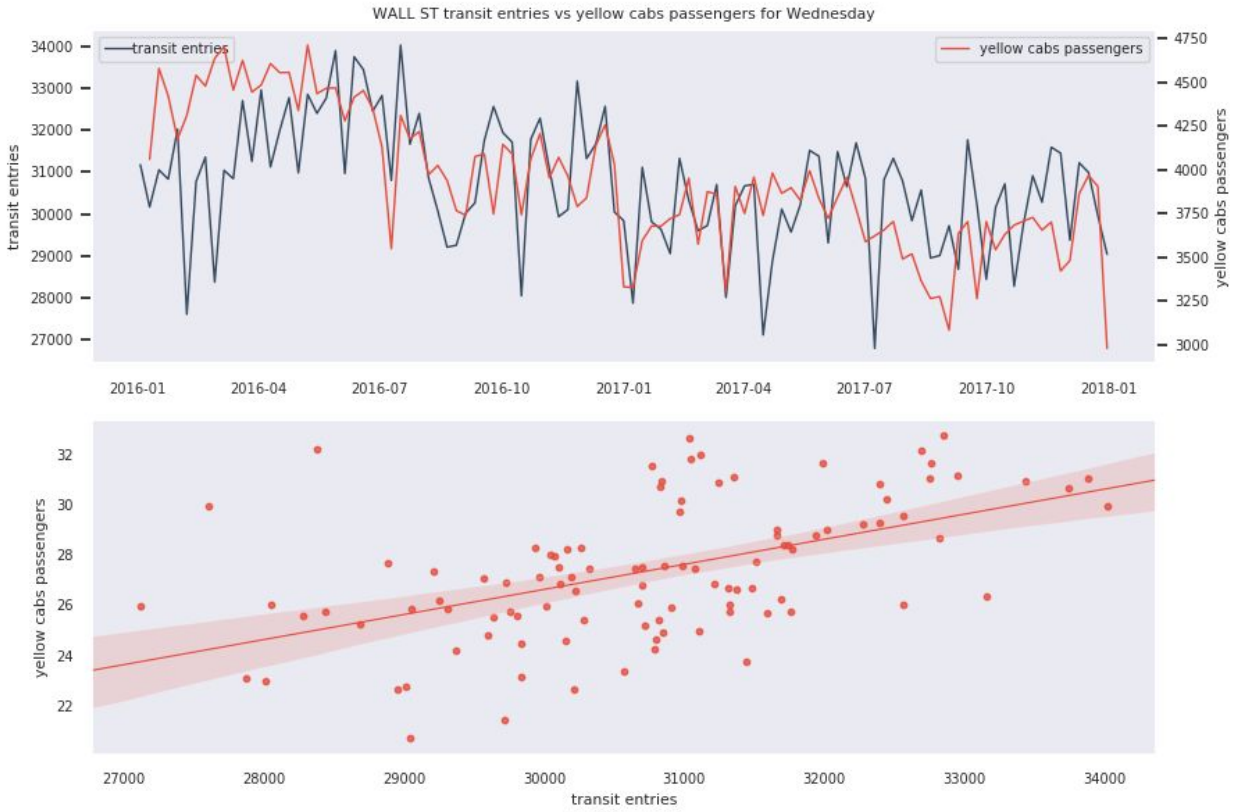


Fig 8b. Relationship between Transit Entries and Cab Rides for Wall Street Station

Transit counts vs Traffic speed:

For most stations (Fig 9a below), there is a negative correlation between transit station (exit and entry) counts and the traffic speed in the vicinity of the station. For the Wall Street (Fig 9b below) and Court Square stations there is a slight positive correlation between transit entry counts and traffic speed in the vicinity of the station.

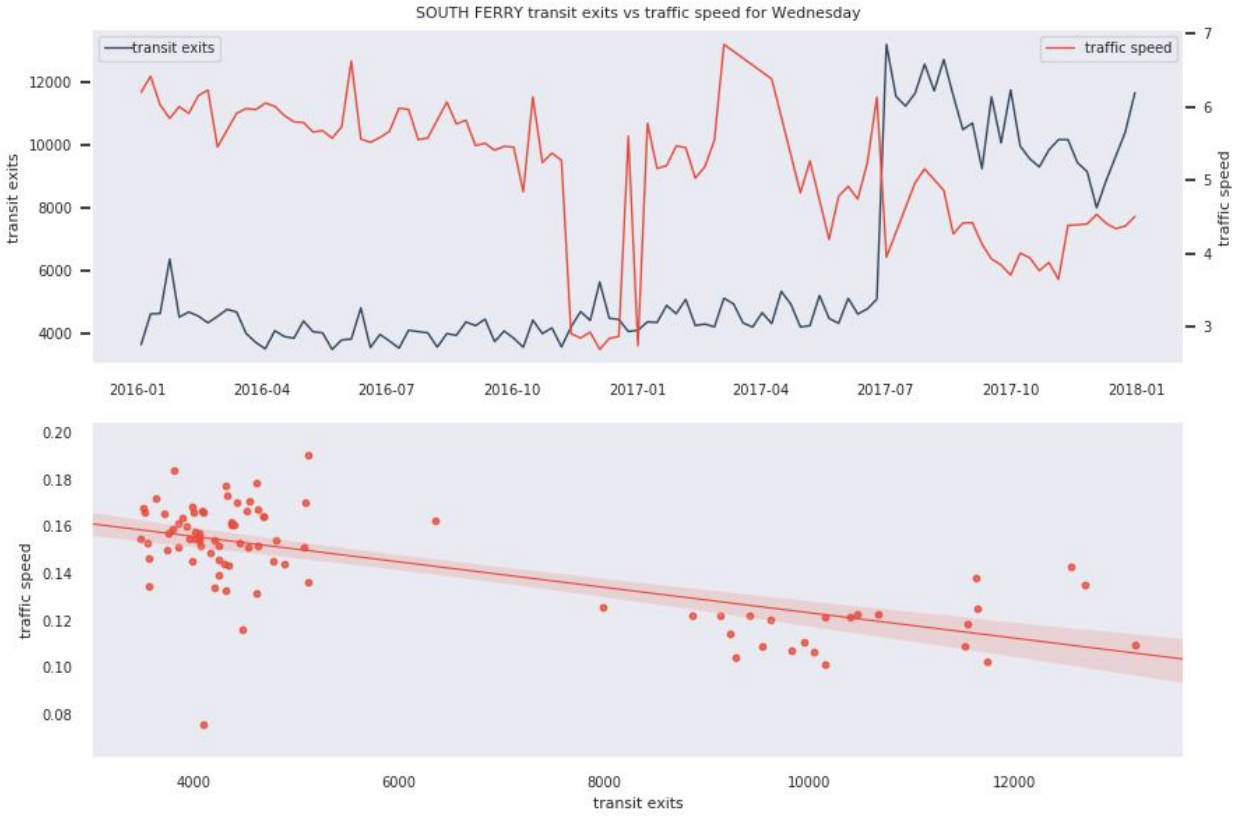


Fig 9a. Relationship between Transit Exits and Traffic Speed for South Ferry Station

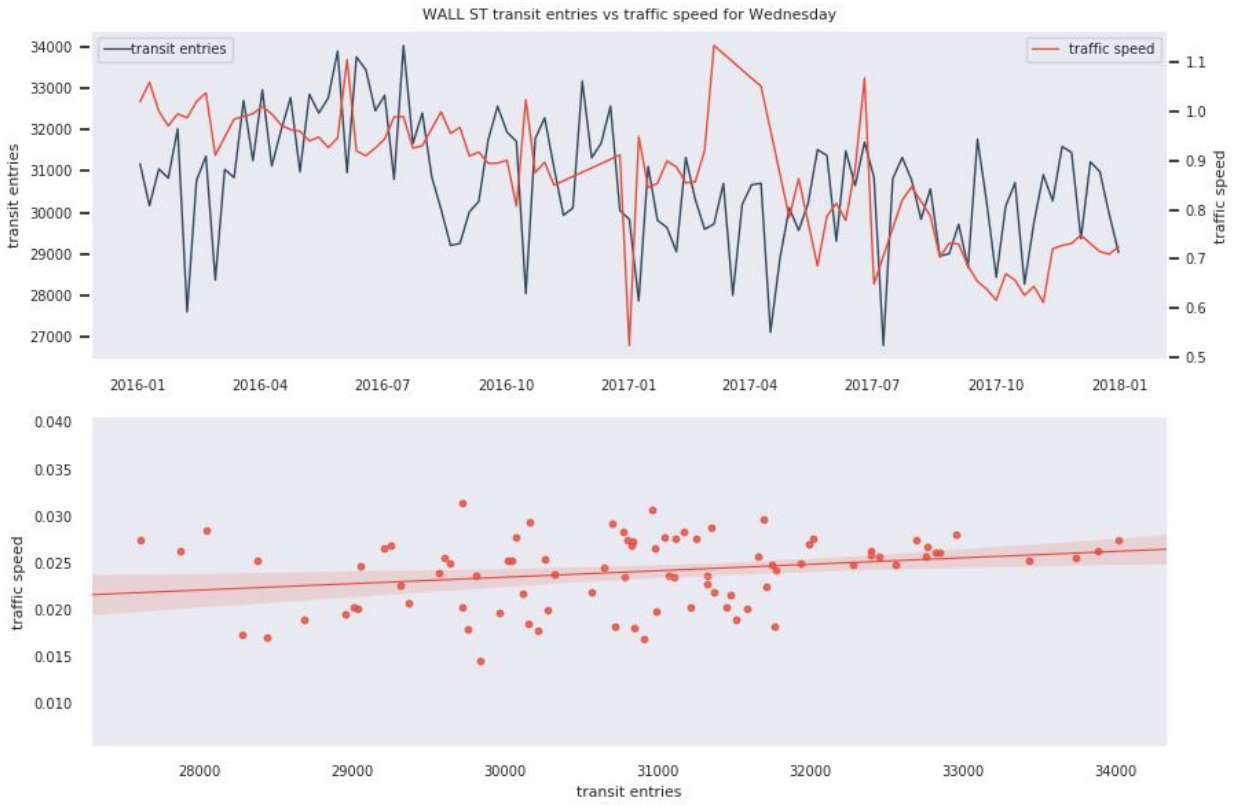


Fig 9b. Relationship between Transit Entries and Traffic Speed for Wall Street Station

Transit counts vs Gas price:

For most stations (Fig 10a below), there is a positive correlation between transit station (exit and entry) counts and gas price. For the Wall Street (Fig 10b below) and Court Square stations there is a negative correlation between transit entry counts and gas price.

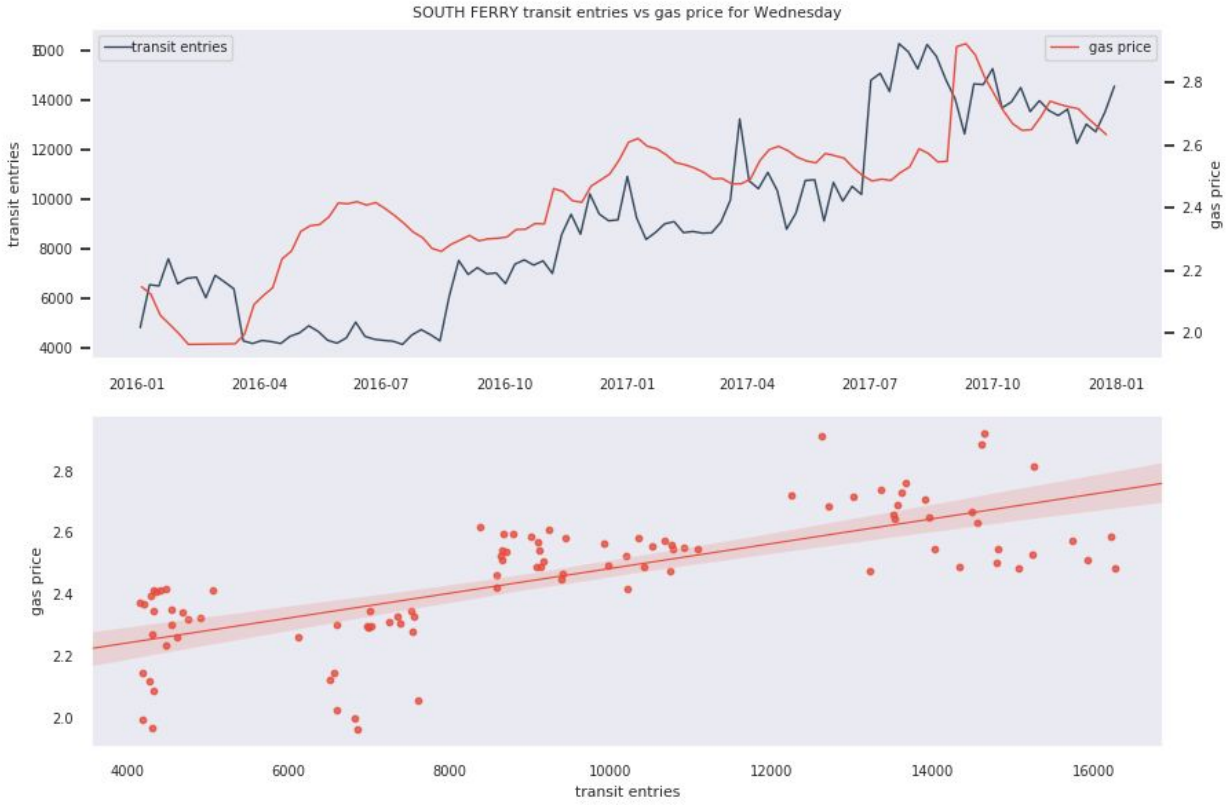


Fig 10a. Relationship between Transit Entries and Gas Price for South Ferry Station

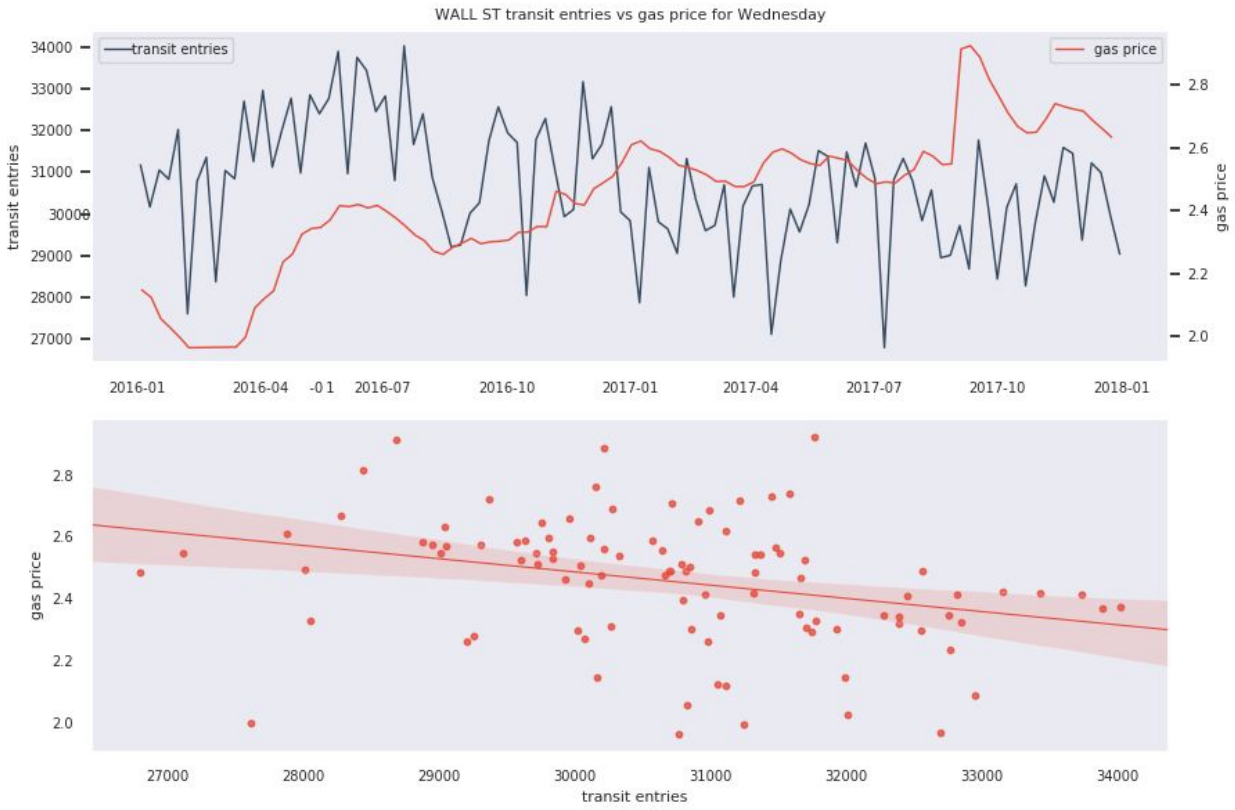


Fig 10b. Relationship between Transit Entries and Gas Price for Wall Street Station

Transit counts vs Weather (daily mean temperature):

For some stations (Fig 11a below) , there is a positive correlation between transit station (exit and entry) counts and mean daily temperature, while for some others (Fig 11b below) there does not seem to be much correlation between weather and transit usage.

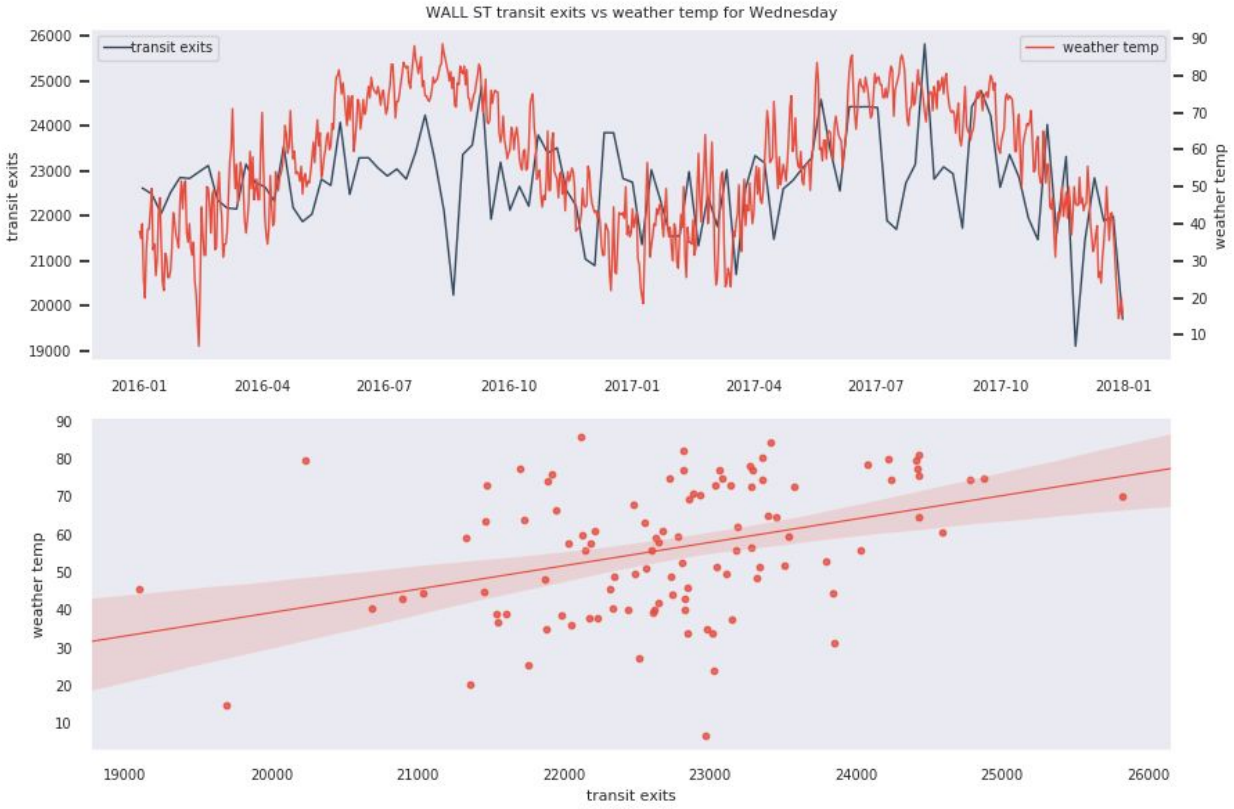


Fig 11a. Relationship between Transit Exits and Avg. Temperature for Wall Street Station

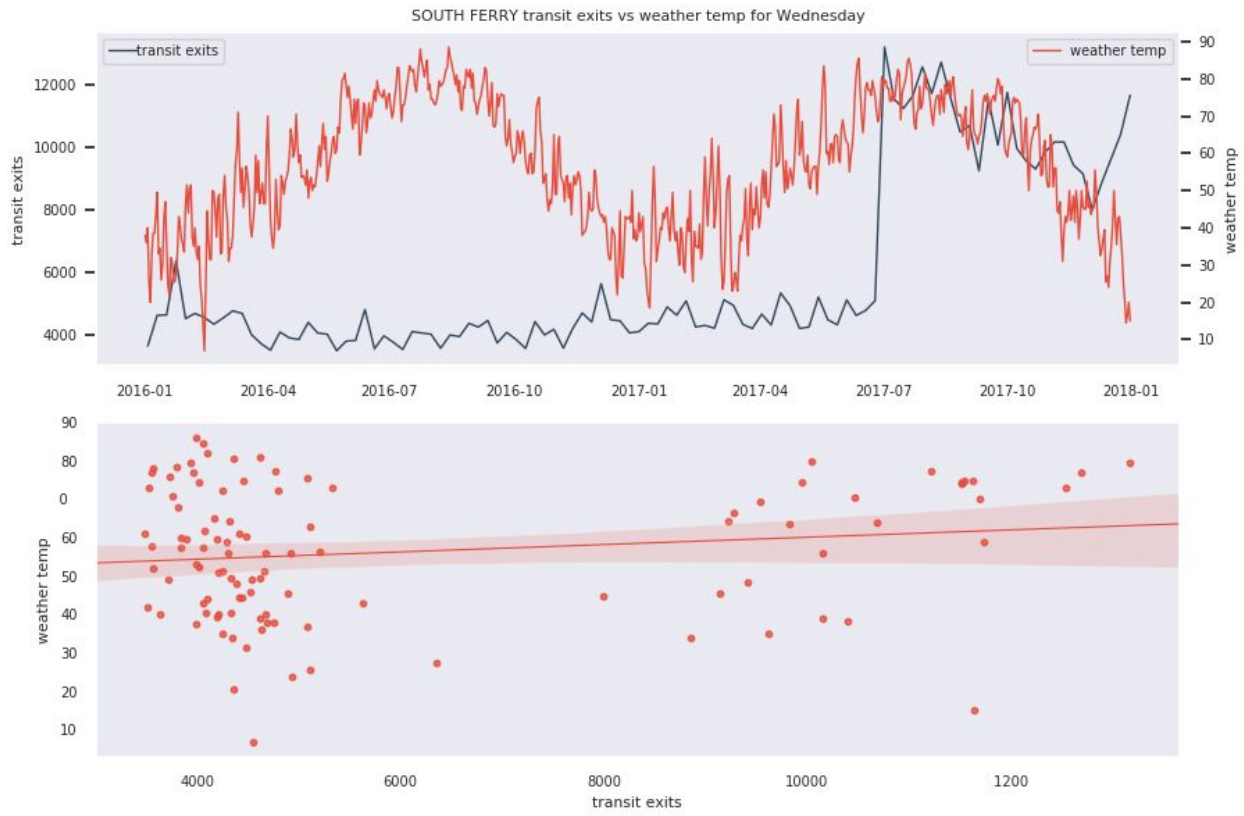


Fig 11b. Relationship between Transit Exits and Avg. Temperature for South Ferry Station

Inferences

1. The time series analysis indicates that all the series have autocorrelation components which need to be removed from the data before applying a statistical model for prediction.
2. Most series (transit, cabs, traffic) are non-stationary with lag = 1 or lag = 7. Some data like cabs have lags at 1 and 7.
3. The non-time series analysis indicates that there is significant cross-correlation (time independent) between the target variables (transit entry and exit counts) and the predictor variables (cabs, traffic, gas and weather)
4. After **making all the variables stationary**, and verifying that each of the **transformed series has an approximate normal distribution**, a **Multivariate ARMA Model like VARMA (Vector Autoregressive Moving Average)** can be applied for making predictions.

Results and In-depth analysis

Transforming the Time Series data and Normality tests

The exogenous variables (cab, traffic, gas and weather) were transformed to remove trend and seasonality. The endogenous variables were left untransformed as the models (VARMAX and VECM) took care of this. The processing steps are at EDA_Modeling.ipynb (section Transforming the Time Series)

Transit Data

The data was left untransformed as the models used later (VARMAX and VECM) took care of the transformation of endogenous variables.

Taxi/Cab Data

Difference smoothing was carried out using lags 7 and 1. The resulting double differenced data has an approximate normal distribution.

Figures 12a and 12b below show the histogram and acf plots of the once differenced cab rides (lag=7). The standard deviation has fallen from 6.265045 to 5.590606 after the first difference. There is still a significant auto-correlation at lag=1

Figures 12c and 12d below show the histogram and acf plots of the twice differenced cab rides(lags=7, 1). The standard deviation has fallen from 5.590606 to 5.224780 from the first to second difference. There is no significant auto-correlation.

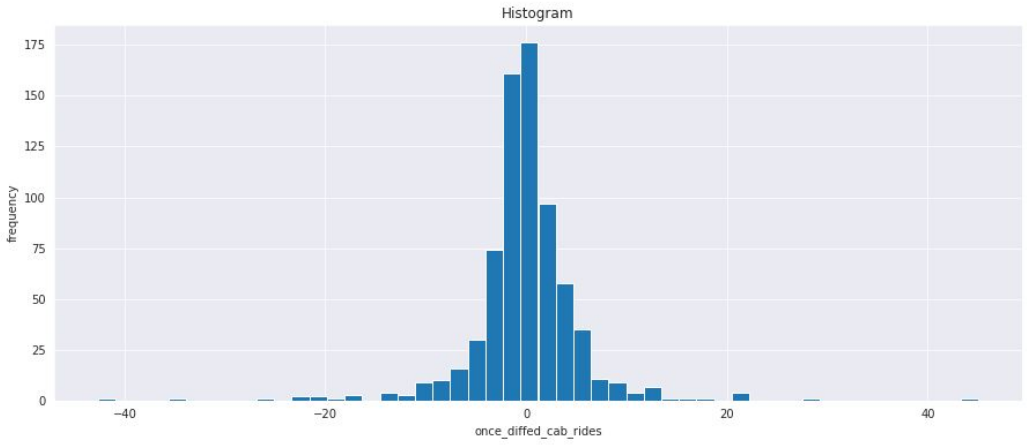


Fig 12a. Histogram of once differenced cab rides in the vicinity of the Wall Street station

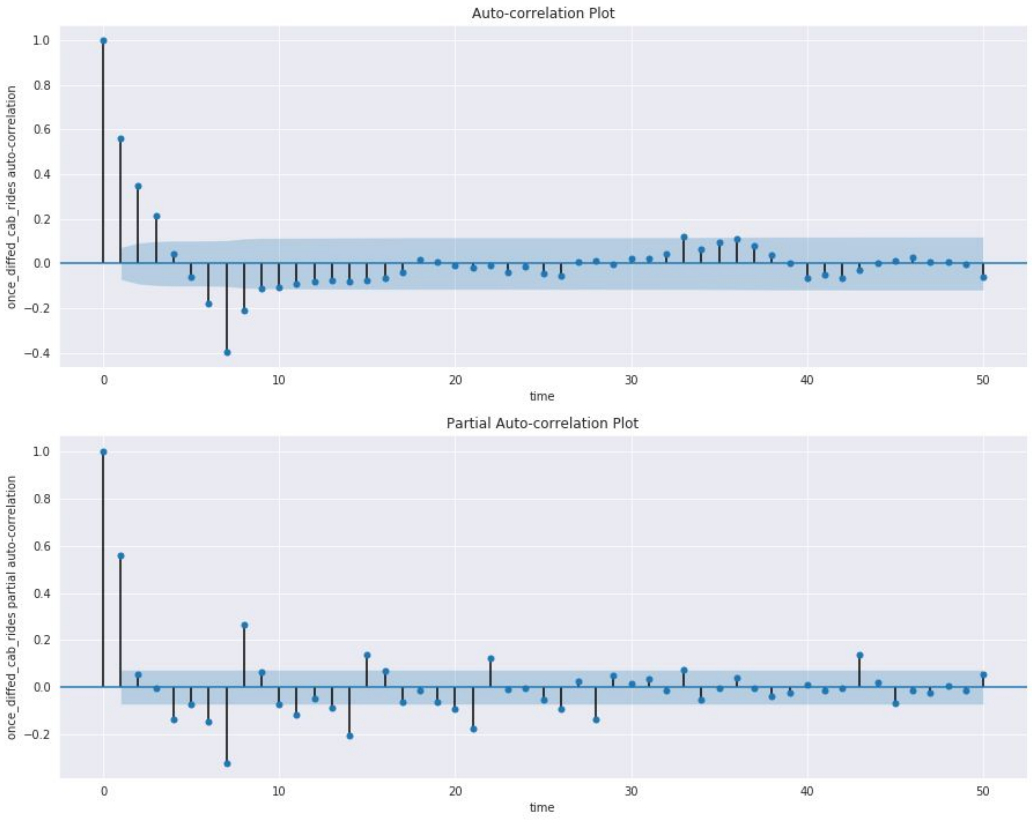


Fig 12b. ACF and PACF plots of once differenced cab rides in the vicinity of the Wall Street station

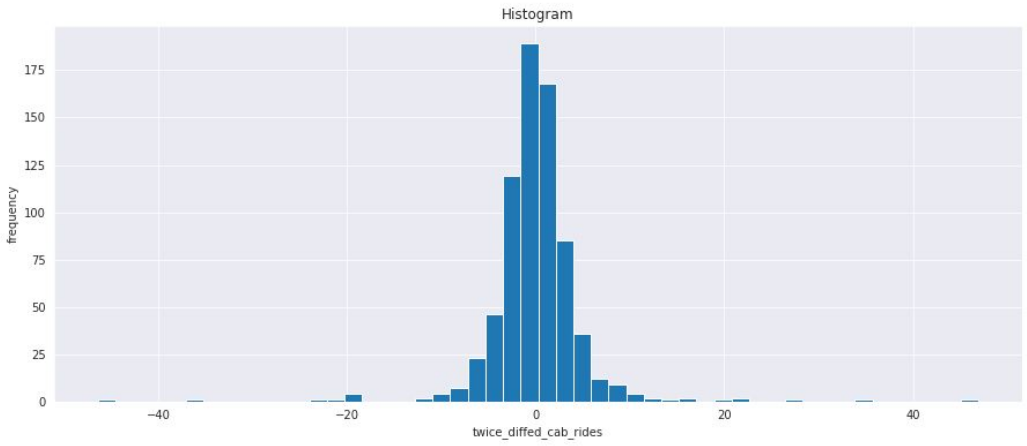


Fig 12c. Histogram of twice differenced cab rides in the vicinity of the Wall Street station

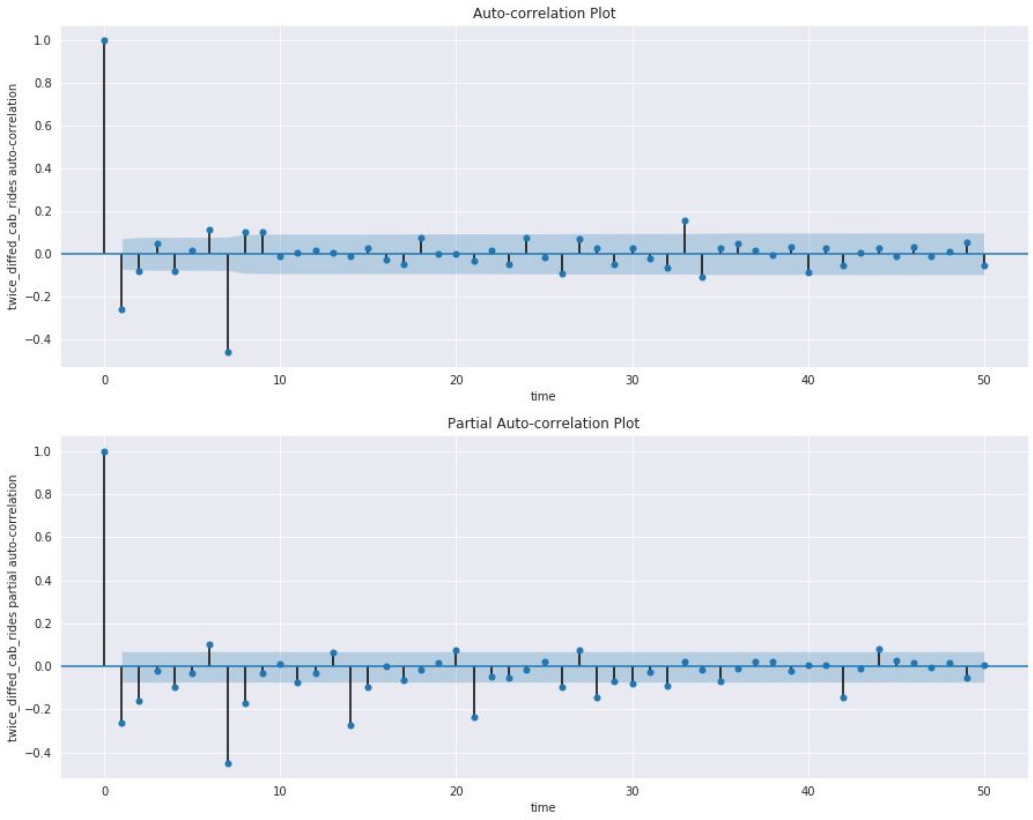


Fig 12d. ACF and PACF plots of twice differenced cab rides in the vicinity of the Wall Street station

Traffic Data

Difference smoothing was carried out using lag 1. The resulting single differenced data has an approximate normal distribution.

Figures 13a and 13b below show the histogram and acf plots of the once differenced traffic speed data (lag=1). The standard deviation has fallen from 5.891836 to 3.988434 after the first difference. There is no significant auto-correlation.

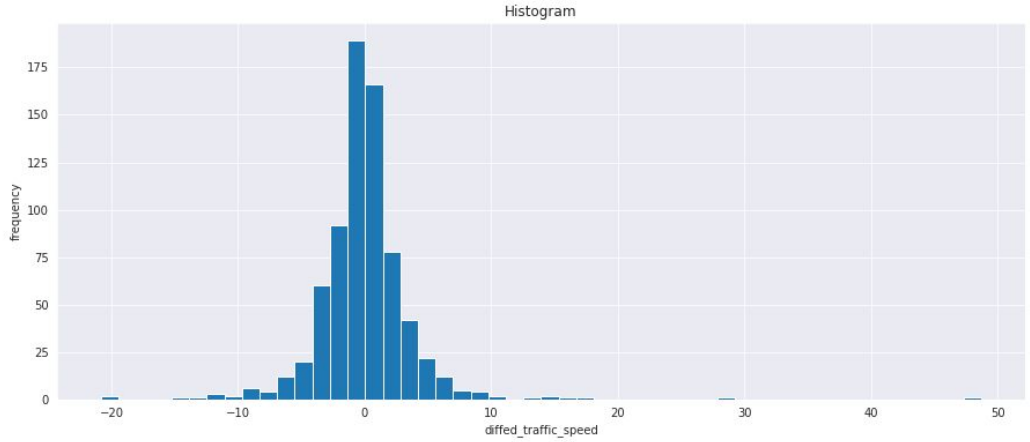


Fig 13a. Histogram of once differenced traffic speed in the vicinity of the Wall Street station

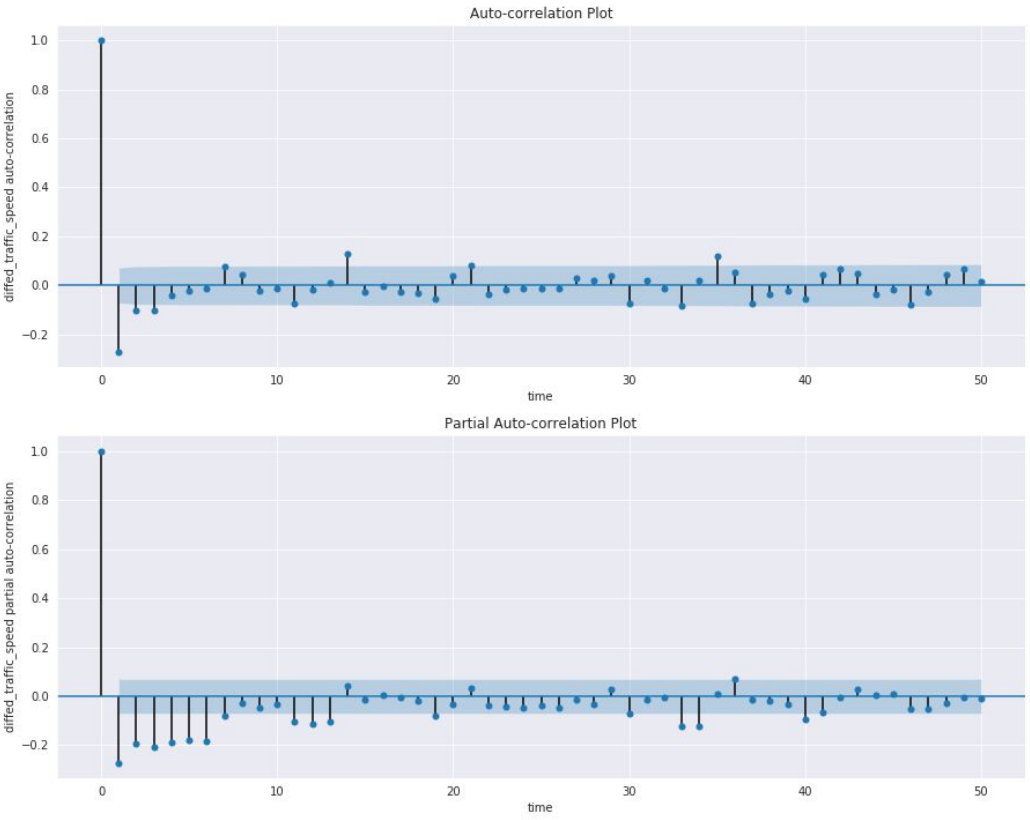


Fig 13b. ACF and PACF plots of once differenced traffic speed in the vicinity of the Wall Street station

Gas Price Data

Difference smoothing was carried out twice using lag 1 each time. The resulting double differenced data has an approximate normal distribution.

Figures 14a and 14b below show the histogram and acf plots of the once differenced gas price (lag=1). The standard deviation has fallen from 0.219972 to 0.006760 after the first difference. There is still a significant auto-correlation at lag=1

Figures 14c and 14d below show the histogram and acf plots of the twice differenced gas price (lags=1, 1). The standard deviation has fallen from 0.006760 to 0.003190 from the first to second difference. There is no significant auto-correlation.

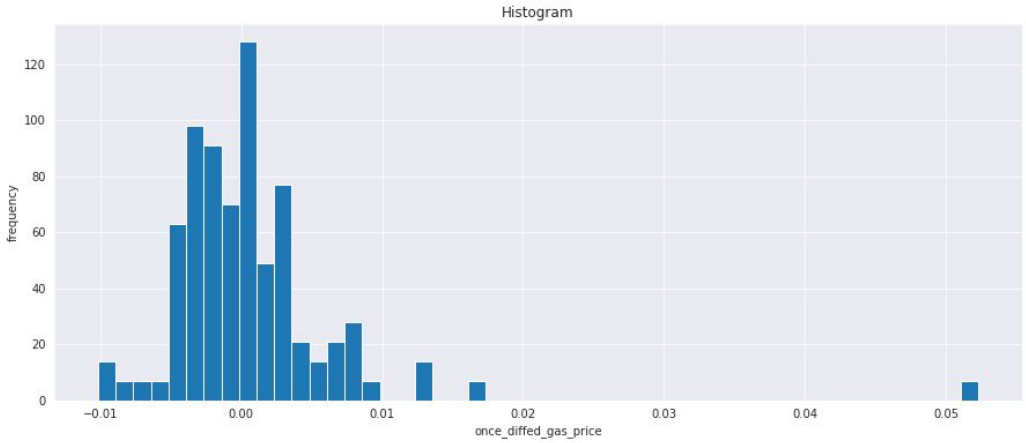


Fig 14a. Histogram of once differenced gas price for NYC

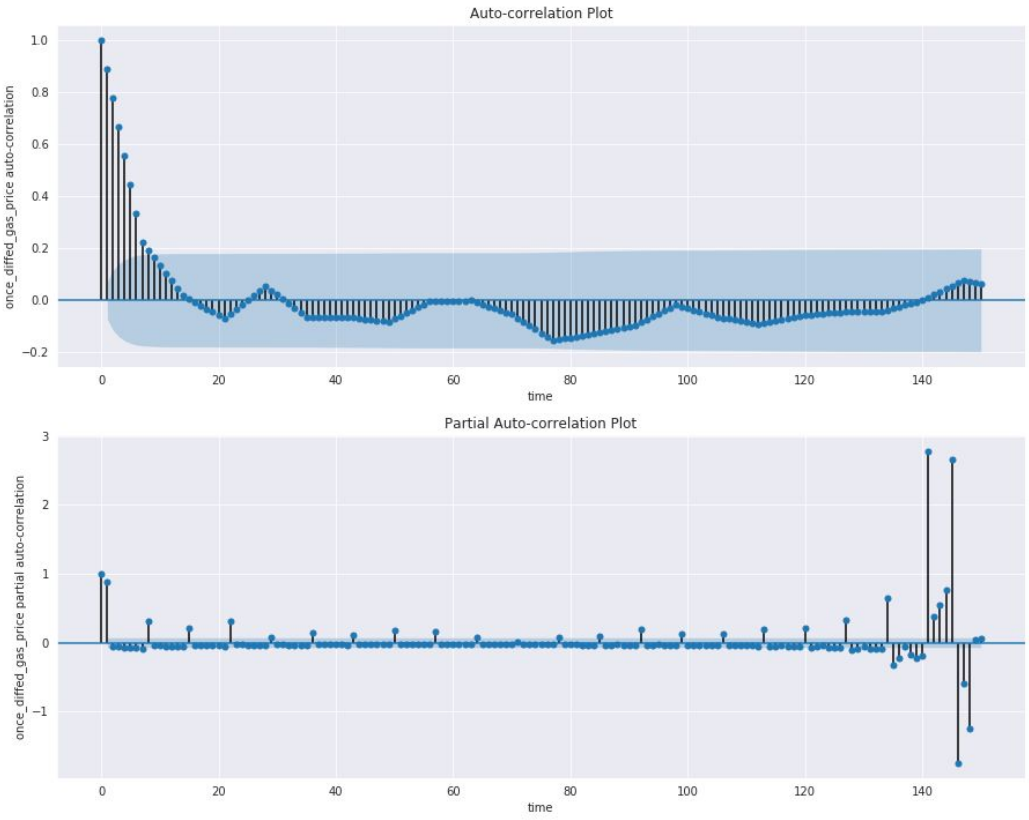


Fig 14b. ACF and PACF plots of once differenced gas price for NYC

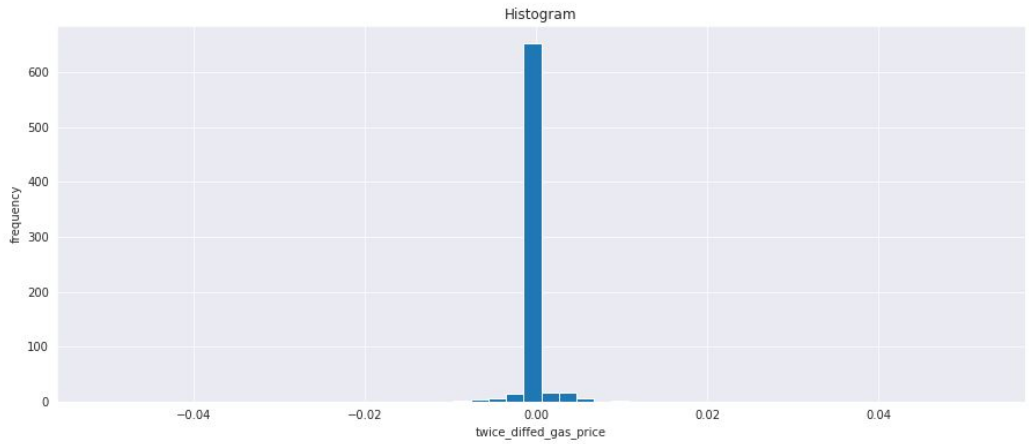


Fig 14c. Histogram of twice differenced gas price for NYC

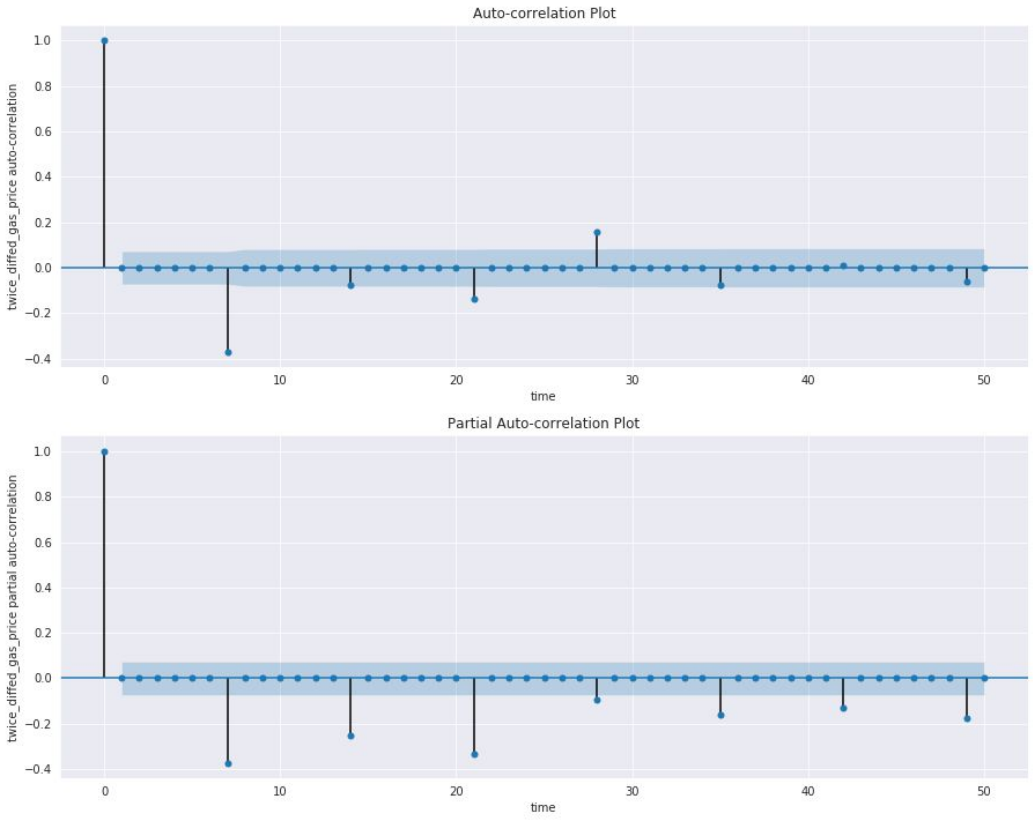


Fig 14d. ACF and PACF plots of twice differenced gas price for NYC

Weather Data

Difference smoothing was carried out using lag 1. The resulting single differenced data has an approximate normal distribution.

Figures 15a and 15b below show the histogram and acf plots of the once differenced avg. temperature (lag=1). The standard deviation has fallen from 16.979780 to 6.143966 after the first difference. There is no significant auto-correlation.

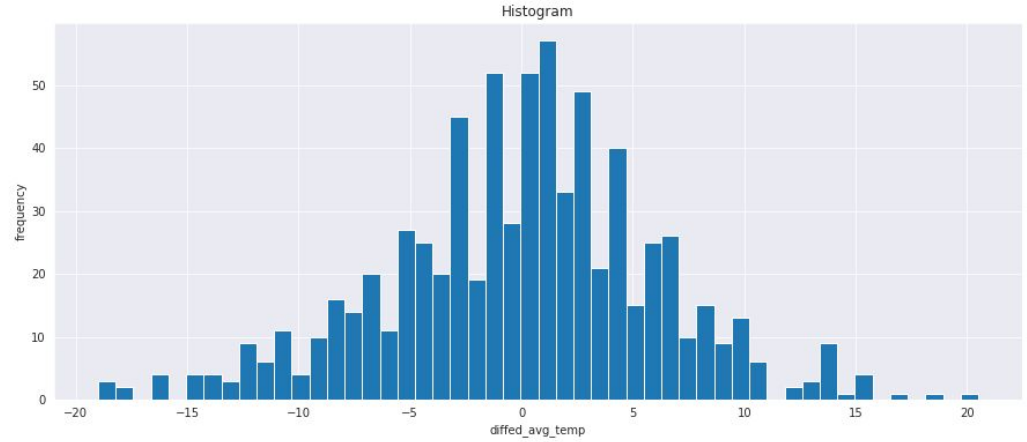


Fig 15a. Histogram of once differenced avg. temperature for NYC

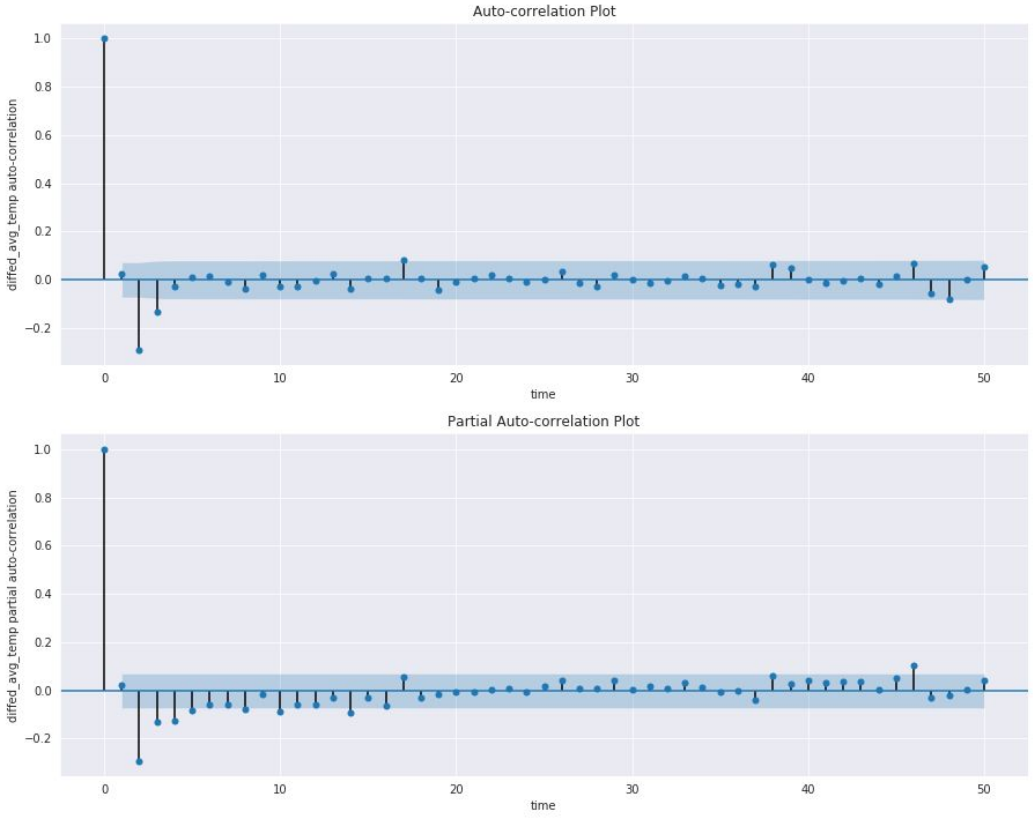


Fig 15b. ACF and PACF plots of once differenced avg. temperature for NYC

Modeling

The processing steps are at EDA_Modeling.ipynb (section Modeling)

The modeling was carried out for the station 'Wall Street' using data for years 2016 and 2017

Train-Test Split

The data was split into training and hold-out sets using data from 1st January 2016 through 30th June 2017 as the training data and from 1st July 2017 through 31st December 2017 as the hold-out set.

VARMAX Model

Vector Autoregressive Moving Average with eXogenous regressors model (statsmodels model VARMAX) was used.

It is the multivariate extension of the ARMA model where each variable is modeled as a linear function of past lags of itself (endogenous AR components), past lags of the other variables (exogenous AR components) and moving average (MA) components.

The hyperparameters of the model are:

1. the order of AR terms denoted by p
2. the order of MA terms denoted by q

The assumption of the model is that the residuals of the model are independent and identically distributed and have a standard normal distribution.

Model Selection

5-fold cross-validation was used to find the optimum order of the VARMAX model (p, q)

The python class TimeSeriesSplit from scikit-learn was used to split the data for cross-validation.

The optimum order for the model (using aic score) was determined as p=3, q=3

Forecasting

The forecasted values for transit entry and exit counts for the station for the period 1st July 2017 - 31st December 2017 were computed using the tuned VARMAX model. Plots were created to compare actual and predicted values (Figures 16a and 16b for transit exits and Figures 16c and 16d for transit entries show these plots)

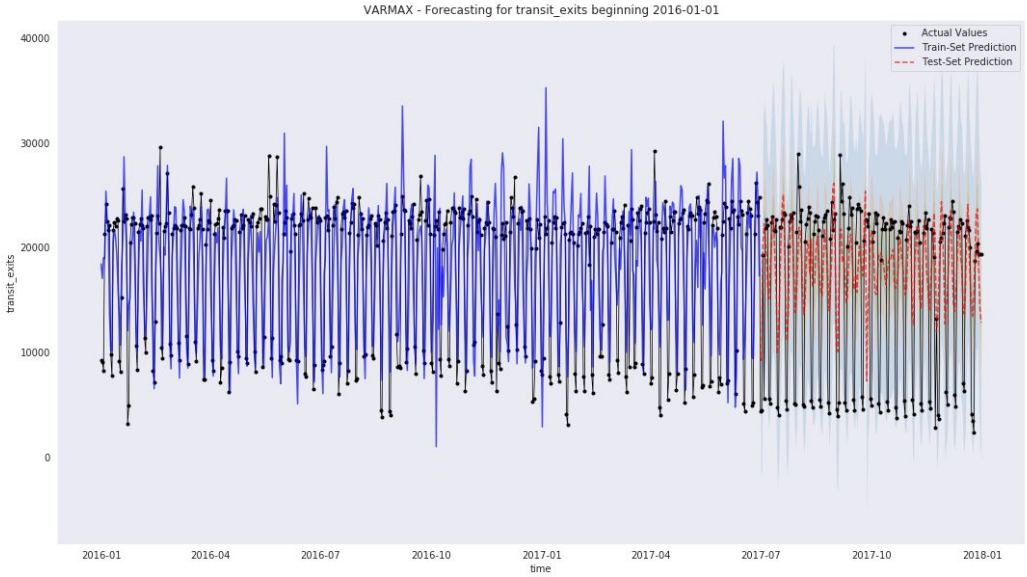


Fig 16a. Actual vs Predicted Value plots for test and train sets using VARMAX for the Exits at Wall Street station

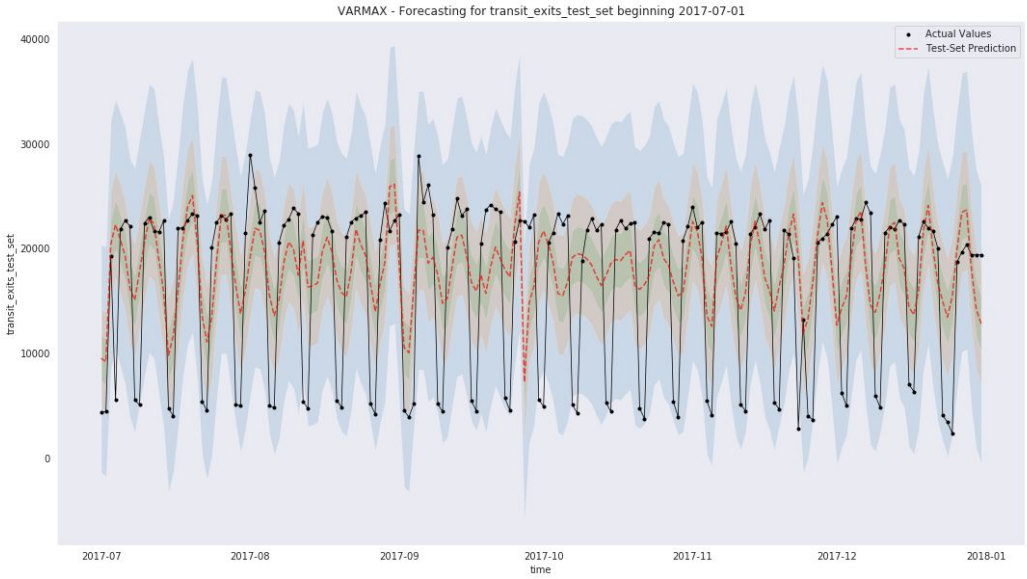


Fig 16b. Actual vs Predicted Value plots for test set only, using VARMAX for the Exits at Wall Street station

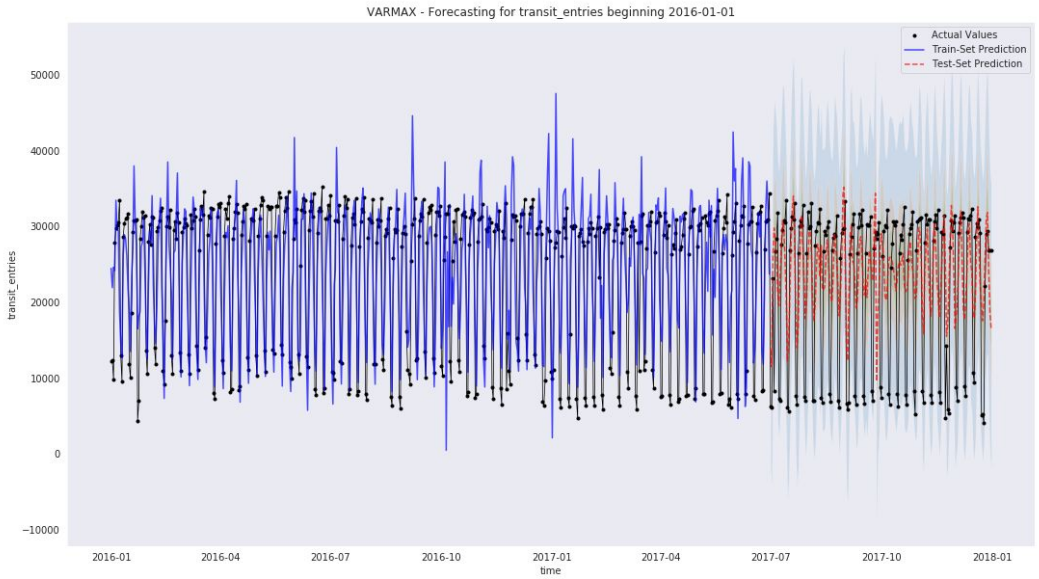


Fig 16c. Actual vs Predicted Value plots for test and train sets using VARMAX for the Entries at Wall Street station

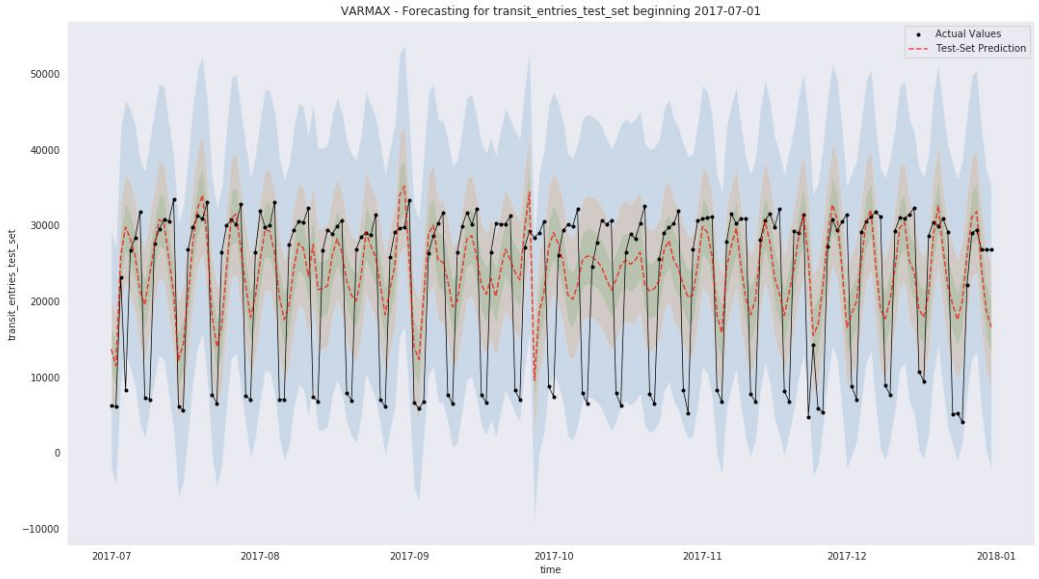


Fig 16d. Actual vs Predicted Value plots for test set only, using VARMAX for the Entries at Wall Street station

Model Residuals

The model residuals were plotted to check for normality. The plots (Figures 17a and 17b) indicate an approximate normal distribution

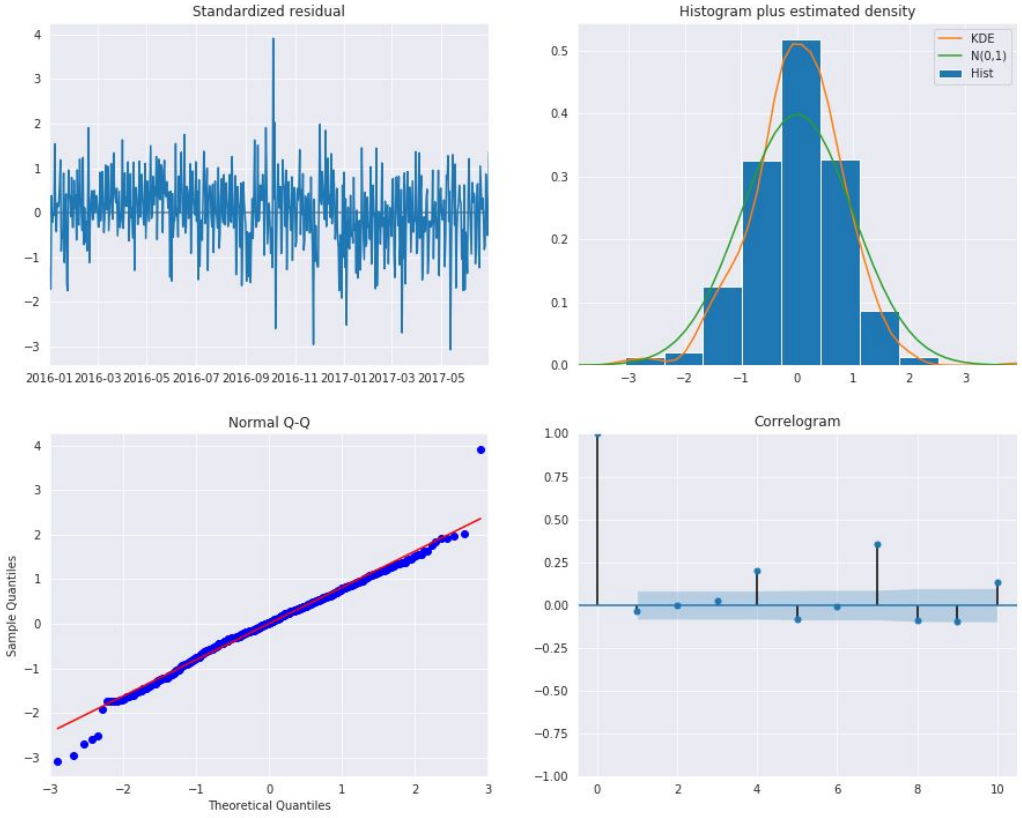


Fig 17a. Diagnostic plots for the residuals from VARMAX for number of exits at Wall Street station

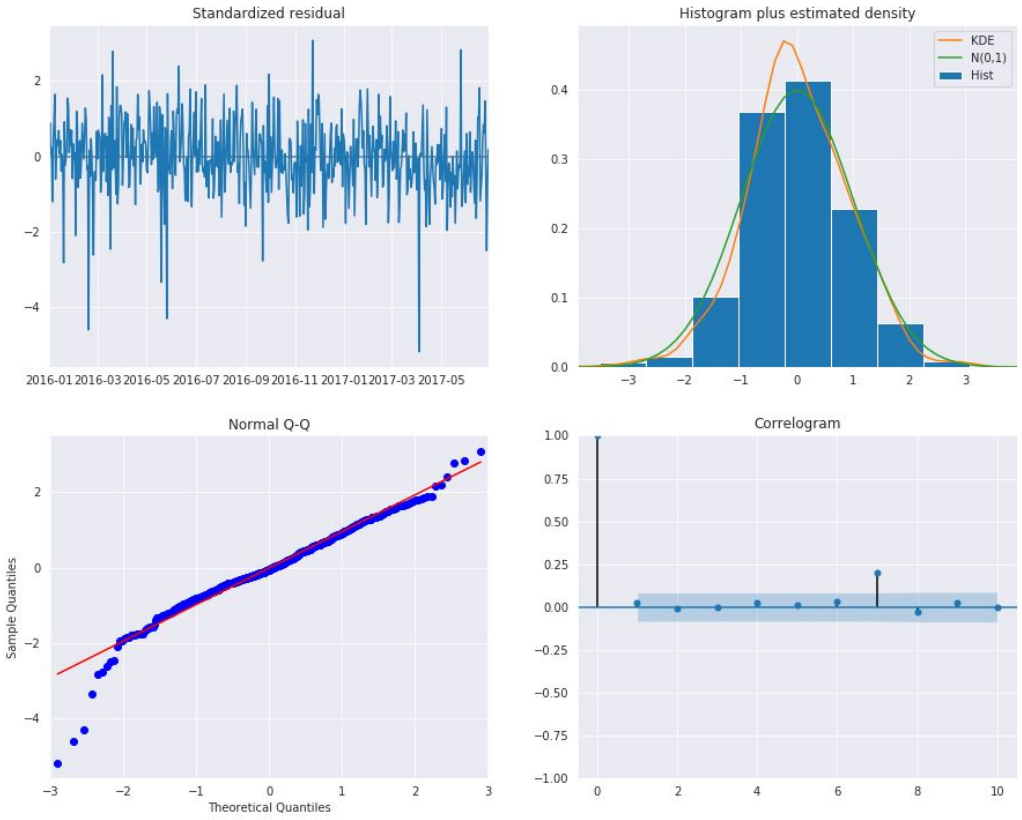


Fig 17b. Diagnostic plots for the residuals from VARMAX for number of entries at Wall Street station

Heteroskedasticity Test :

(where the null hypothesis implies no heteroskedasticity, i.e. the variance does not vary with time)

Transit Passengers (exiting the station) :
t-statistic 1.4471757664411473
p-value 0.013037722320413319

Transit Passengers (entering the station) :
t-statistic 0.794863263031284
p-value 0.1223898072811961

The above results show that the residuals for entries have significant heteroskedasticity while those for exits do not.

Further modeling of Residuals :

The residuals were further modeled using an ARMA model (to remove residual heteroskedasticity)

Forecasting after Modeling the Residuals

The forecasted values for transit entry and exit counts for the station for the period 1st July 2017 - 31st December 2017 were computed using the tuned VARMAX model (with ARMA applied to residuals). Plots were created to compare actual and predicted values (Figures 18a and 18b for transit entries)

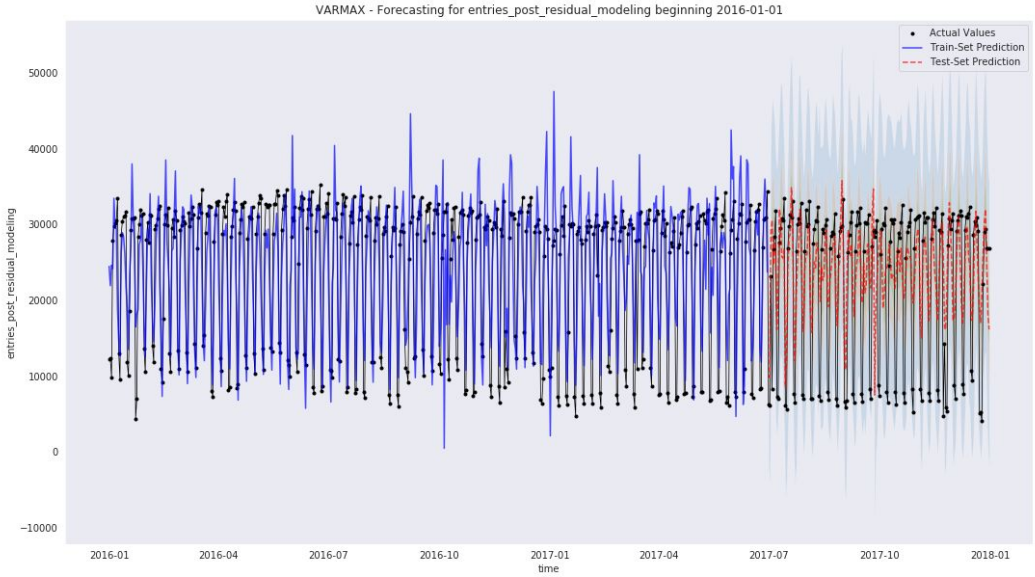


Fig 18a. Actual vs Predicted Value plots for test and train sets using VARMAX+ARMA for the Entries at Wall Street station

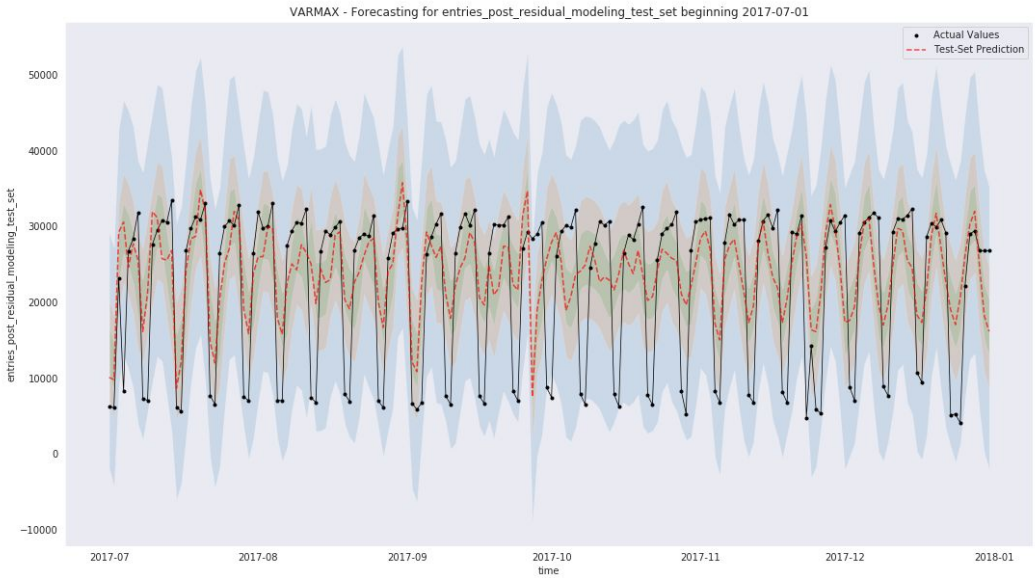


Fig 18b. Actual vs Predicted Value plots for test set only, using VARMAX+ARMA for the Entries at Wall Street station

Model Accuracy Score

The accuracy score (RMSE) for the model was computed as follows:

Transit Exits (VARMAX only):
6773.6

Transit Entries (VARMAX only):
9039.96

Transit Entries (VARMAX+ARMA):
8309.92

VECM Model

Vector Error Correction Model (statsmodels model VECM) was used.

It belongs to a category of multiple time series models most commonly used for data where the underlying variables have a long-run stochastic trend, also known as cointegration. If two integrated variables share a common stochastic trend such that a linear combination of these variables is stationary, they are called cointegrated. In a system of variables, there may be several linearly independent cointegrating vectors. In that case linear combinations of these vectors are also cointegrating vectors because linear combinations of stationary variables are stationary (ref)

The hyperparameters of the model are:

1. the number of lagged differences denoted by k_ar
2. cointegration rank denoted by coint_rank

The assumption of the model is that the residuals of the model are independent and identically distributed and have a standard normal distribution.

Model Selection

5-fold cross-validation was used to find the optimum parameters of the VECM model (k_ar and coint_rank)

The parameter for the number of periods in a cyclical season was set to 7, due to the mostly weekly seasonality in the data.

The python class TimeSeriesSplit from scikit-learn was used while splitting the data for cross-validation.

The optimum parameters for the model (using log-likelihood score) was determined as k_ar=3, coint_rank=1

Forecasting

The forecasted values for transit entry and exit counts for the station for the period 1st July 2017 - 31st December 2017 were computed using the tuned VECM model. Plots were created to compare actual and predicted values

(Figures 19a and 19b for transit exits and Figures 19c and 19d for transit entries show these plots)

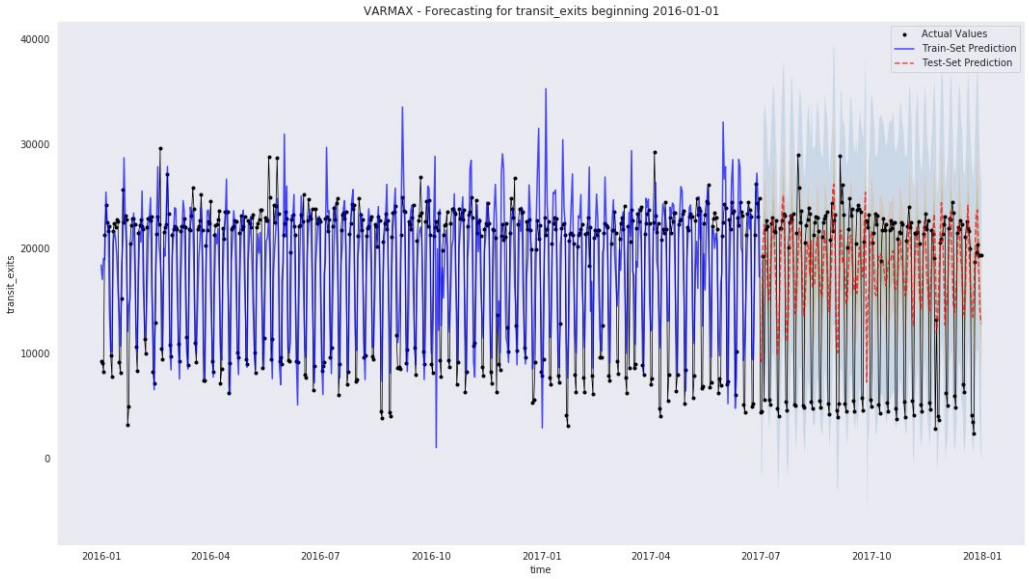


Fig 19a. Actual vs Predicted Value plots for test and train sets using VECM for the Exits at Wall Street station

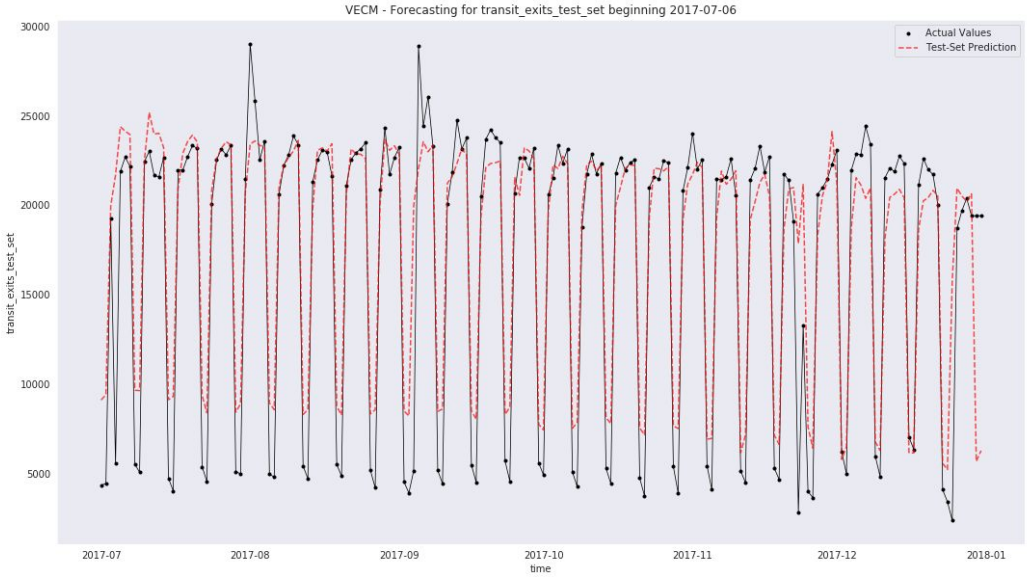


Fig 19b. Actual vs Predicted Value plots for test set only, using VECM for the Exits at Wall Street station

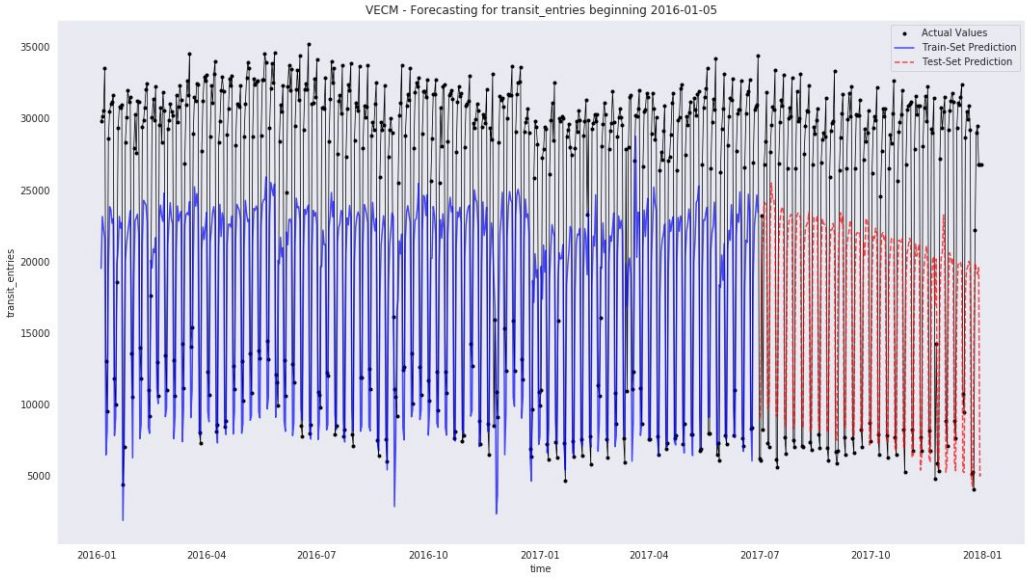


Fig 19c. Actual vs Predicted Value plots for test and train sets using VECM for the Entries at Wall Street station

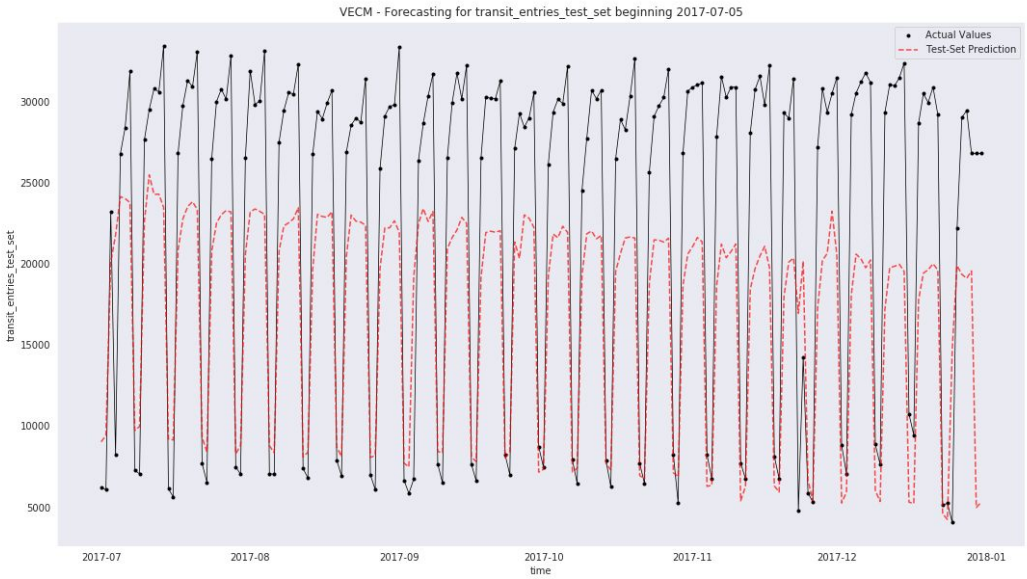


Fig 19d. Actual vs Predicted Value plots for test set only, using VECM for the Entries at Wall Street station

Model Residuals

The model residuals were tested to check for normality.

Normality test for VECM residuals for Exits:

significance level	0.050000
t-statistic	1408.256322
p-value	0.000000

Normality test for VECM residuals for Entries:

significance level	0.050000
t-statistic	1183.128583
p-value	0.000000

Model Accuracy Score

The accuracy score (RMSE) for the model was computed as follows:

Transit Exits (VECM):

3439.37

Transit Entries (VECM):

4428.06

Conclusion

It is obvious from the prediction plots and the accuracy scores that the VECM model performs significantly better than the VARMAX model for both transit entries and exits.

Further Exploration

Further exploration of time series regression models was performed using Dynamic Linear Regression, GAS Regression and Beta-t-EGARCH Regression models. The results and plots are at EDA_Modeling.ipynb (section Further experimentation with other Time Series Forecasting Models)

References

Dask and Geopandas

1. http://matthewrocklin.com/blog/work/2017/01/12/dask-dataframes
2. https://towardsdatascience.com/geospatial-operations-at-scale-with-dask-and-geopandas-4d92d00eb7e8
3. http://matthewrocklin.com/blog/work/2017/09/21/accelerating-geopandas-1
4. https://nbviewer.jupyter.org/urls/gist.githubusercontent.com/mrocklin/ba6d3e2376e478c344af7e874e6fcbb1/raw/e0db89644f78f4371ee30fbdd517ce9bd6032a5e/nyc-taxi-geospatial.ipynb
5. https://github.com/mrocklin/dask-geopandas

Setup with Docker, Redis, Minio

1. https://www.dataquest.io/blog/install-and-configure-docker-swarm-on-ubuntu/
2. https://medium.com/@james.p.edwards42/redis-swarm-12039b486b89
3. https://blog.garage-coding.com/2016/02/05/bash-fifo-jobqueue.html
4. https://testdriven.io/asynchronous-tasks-with-flask-and-redis-queue

Time Series Analysis

1. https://people.duke.edu/~rnau/411arim3.htm
2. http://www-personal.umich.edu/~lkilian/SVARch03.pdf
3. https://pyflux.readthedocs.io/en/latest/
4. https://otexts.org/fpp2/stationarity.html
5. https://www.youtube.com/watch?v=_vQ0W_qXMxk
6. https://theintelligenceofinformation.wordpress.com/2018/06/18/time-series-analysis-in-python-and-r/
7. https://mxbu.github.io/logbook/2017/01/04/forecasting-swiss-gdp-growth-with-statsmodels/
8. https://machinelearningmastery.com/make-sample-forecasts-arima-python/
9. https://machinelearningmastery.com/time-series-seasonality-with-python/