Yellow Taxi Demand Prediction

Tinsae Alemayehu

April 2019

Contents

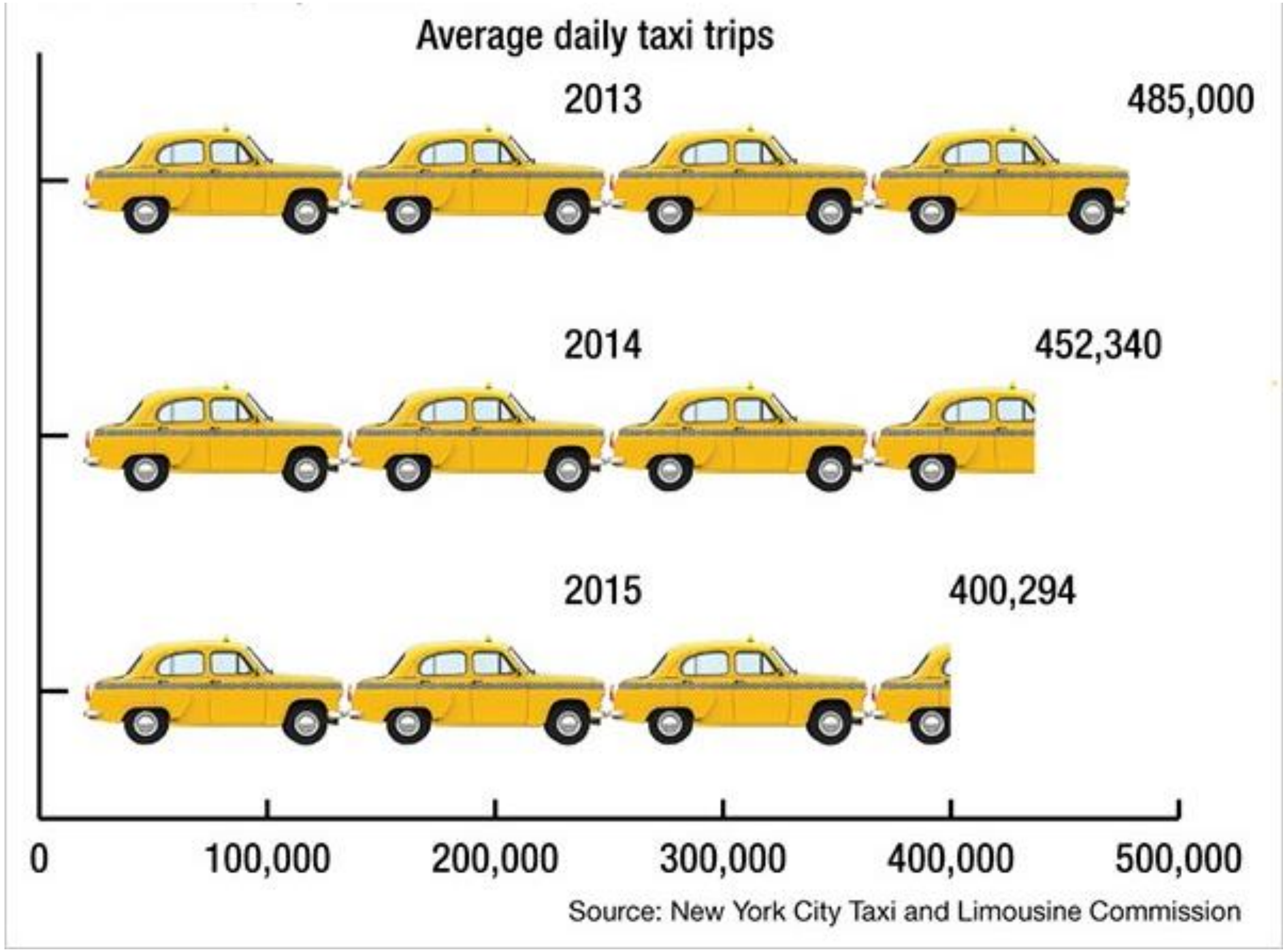
Yellow Taxis

Keep 💯 of the fares.

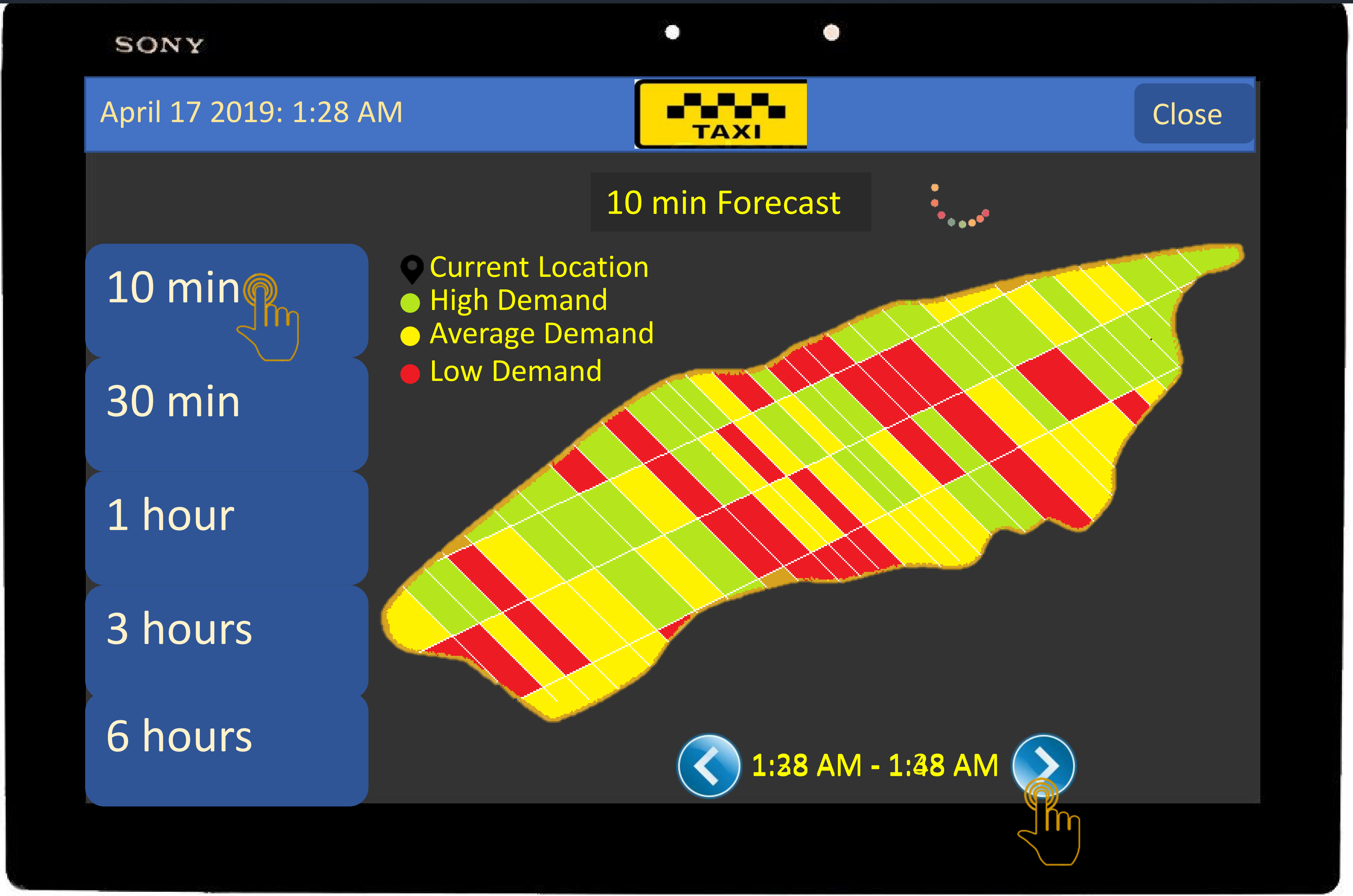
The Uber Effect

Number of Uber trips outpaced yellow taxis for the first time in 2018

65000 VS 13500



The Product


SONY
April 17 2019: 1:28 AM
TAXI
Close
10 min Forecast
10 min
30 min
1 hour
3 hours
6 hours
Current Location
High Demand
Average Demand
Low Demand
1:38 AM - 1:48 AM

Initial Exploration

The data on Kaggle is cleaned and resampled data for 2016

NYC OpenData

170 M records, 16 GB space

Initial Exploration

DASK

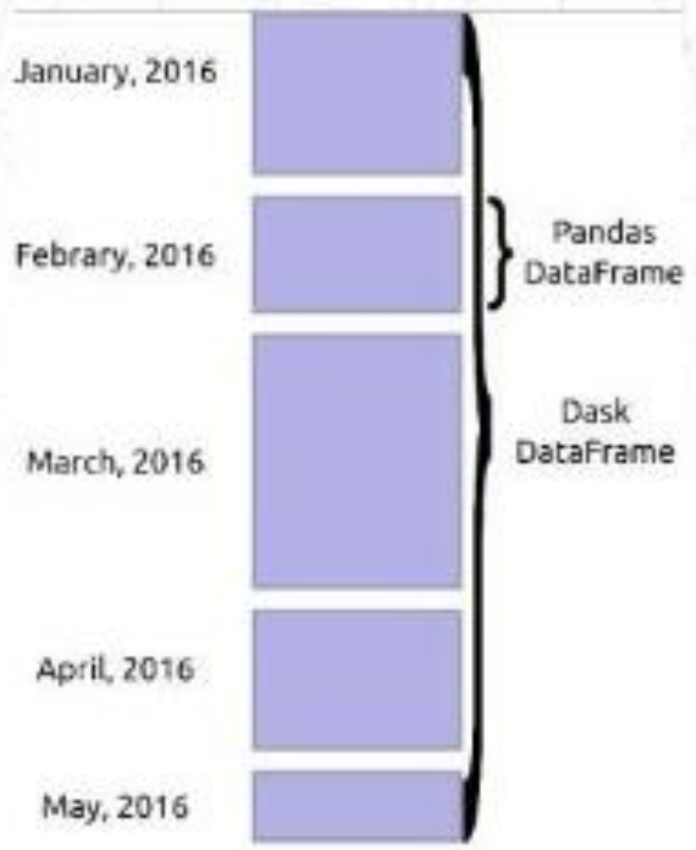
- **Same API**

```
import dask.dataframe as dd
df = dd.read_parquet(‘s3://bucket/accounts/2017')
df.groupby(df.name).value.mean().compute()
```

- **Efficient Timeseries Operations**

```
# Use the pandas index for efficient operations
df.loc[‘2017-01-01’]
df.value.rolling(10).std()
df.value.resample(‘10m’).mean()
```

- **Co-developed with Pandas and by the Pandas developer community**



Initial Exploration

DASK

- 2017: 165 Partitions
- 2018: 92 Partitions

part.0.parquet	17,477 KB
part.1.parquet	18,698 KB
part.2.parquet	16,644 KB
part.3.parquet	18,888 KB

- ⚠ Dask doesn't support "index_col" keyword argument
- ⚠ Setting new index from unsorted column is expensiv
- ⚠ Positional row indexing(iloc) & label indexing(loc) is not possible

Initial Exploration

Parsing Date

tpep_pickup_datetime
2017 Jan 09 11:13:28 AM
2017 Jan 09 11:32:27 AM
2017 Jan 09 11:38:20 AM

```
y2017 = dd.read_csv(
'2017_trips.csv',
dtype=dtypes_s
parse_dates=
["tpep_pickup_datetime",
"tpep_dropoff_datetime"])
```

Google Cloud

Load dates as strings

```
tpep_pickup_datetime: object
tpep_dropoff_datetime: object
```

Initial Exploration

Parsing Date

'datetime64[ns]'

```
# parse dates # convert string to numpy date type
def parse_dates(df):
  df = df.assign(tpep_pickup_datetime=pd.to_datetime(df['tpep_pickup_datetime'],
="%m/%d/%Y %I:%M:%S %p"))
df = df.assign(tpep_dropoff_datetime=pd.to_datetime(df['tpep_dropoff_datetime'],
="%m/%d/%Y %I:%M:%S %p"))
return df
# map_partition applies a function to each partition # delayed
y2017_ddf = y2017_ddf.map_partitions(parse_dates, meta=meta_all)
y2018_ddf = y2018_ddf.map_partitions(parse_dates, meta=meta_all)
```

Initial Exploration

set-index DASK

Remove Erroneous Dates

2068

Dec 31 to Jan 01

2001

02/02/2009 11:00 PM
to
02/02/2009 11:20 PM

'datetime64[ns]'

Dask is fast when indexed by time

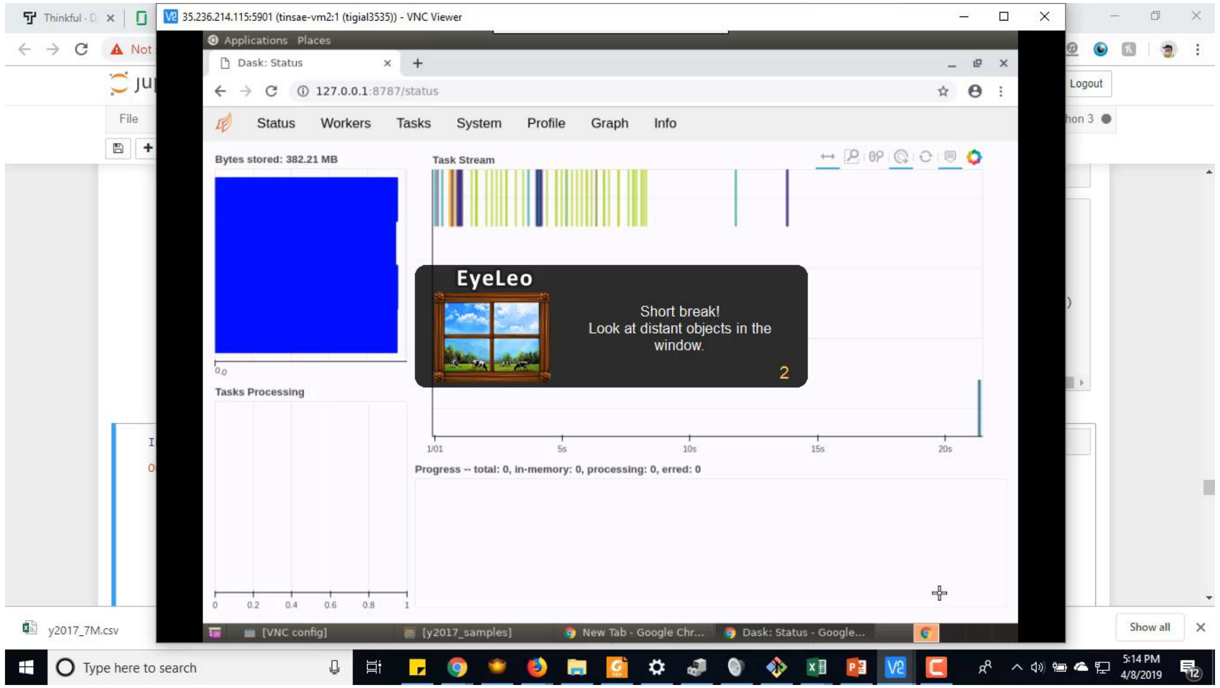
Parquet

Decrease memory burden

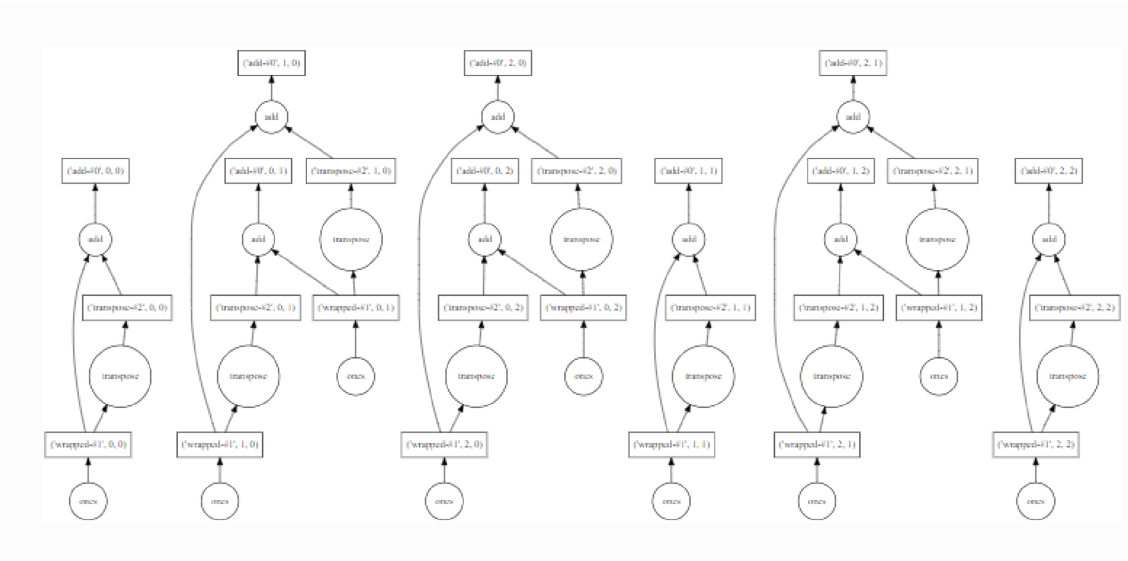
	VendorID	tpep_dropoff_datetime
tpep_pickup_datetime		
2017-01-01 00:00:00	2.0	2017-01-01 00:00:00
2017-01-01 00:00:02	1.0	2017-01-01 00:03:50
2017-01-01 00:00:02	2.0	2017-01-01 00:39:22
2017-01-01 00:00:03	1.0	2017-01-01 00:06:58
2017-01-01 00:00:05	1.0	2017-01-01 00:08:33

Initial Exploration

DASK



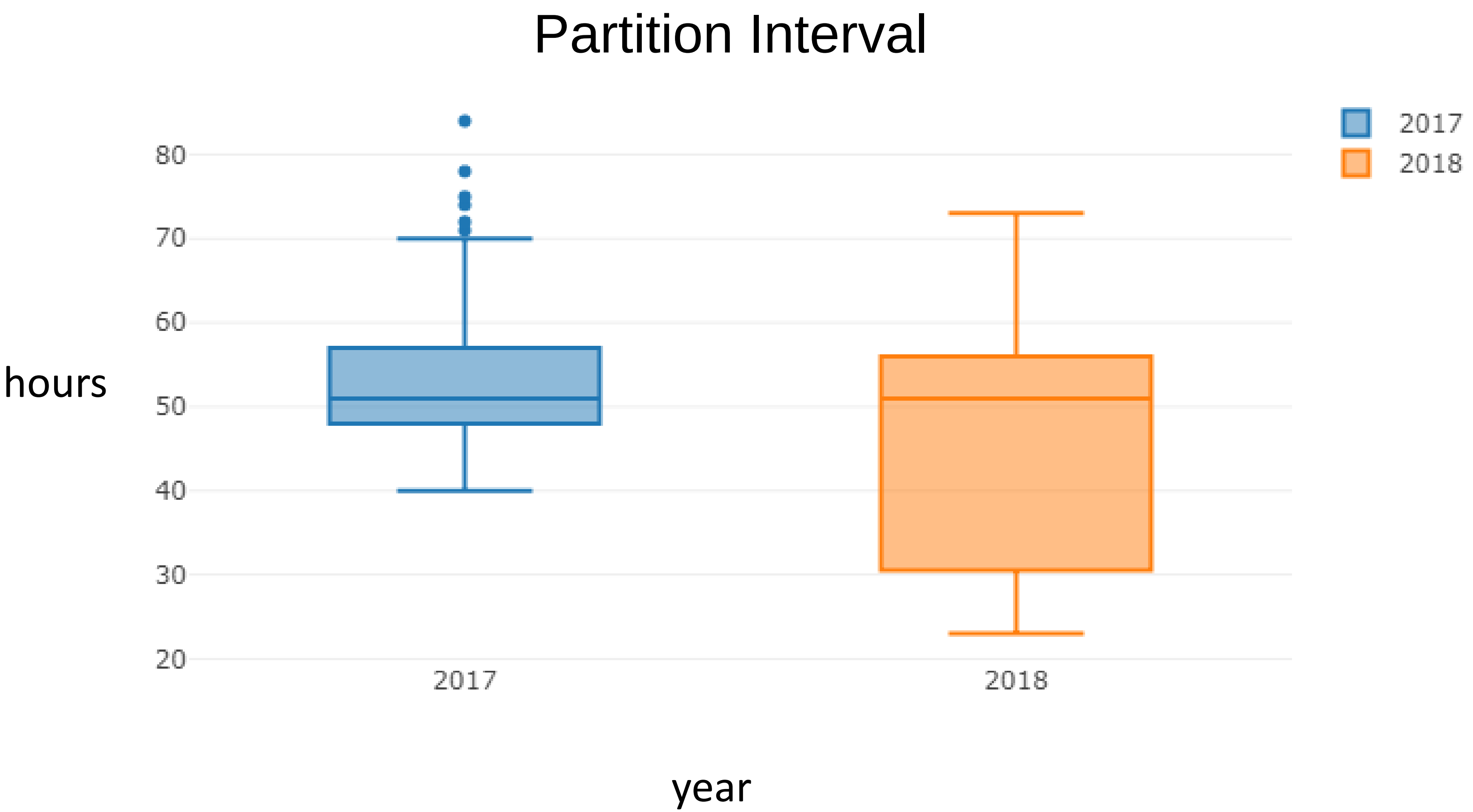

```python
import dask.array as da
x = da.ones((15, 15), chunks=(5, 5))
y = x + x.T
# y.compute()
y.visualize(filename='transpose.svg')
```



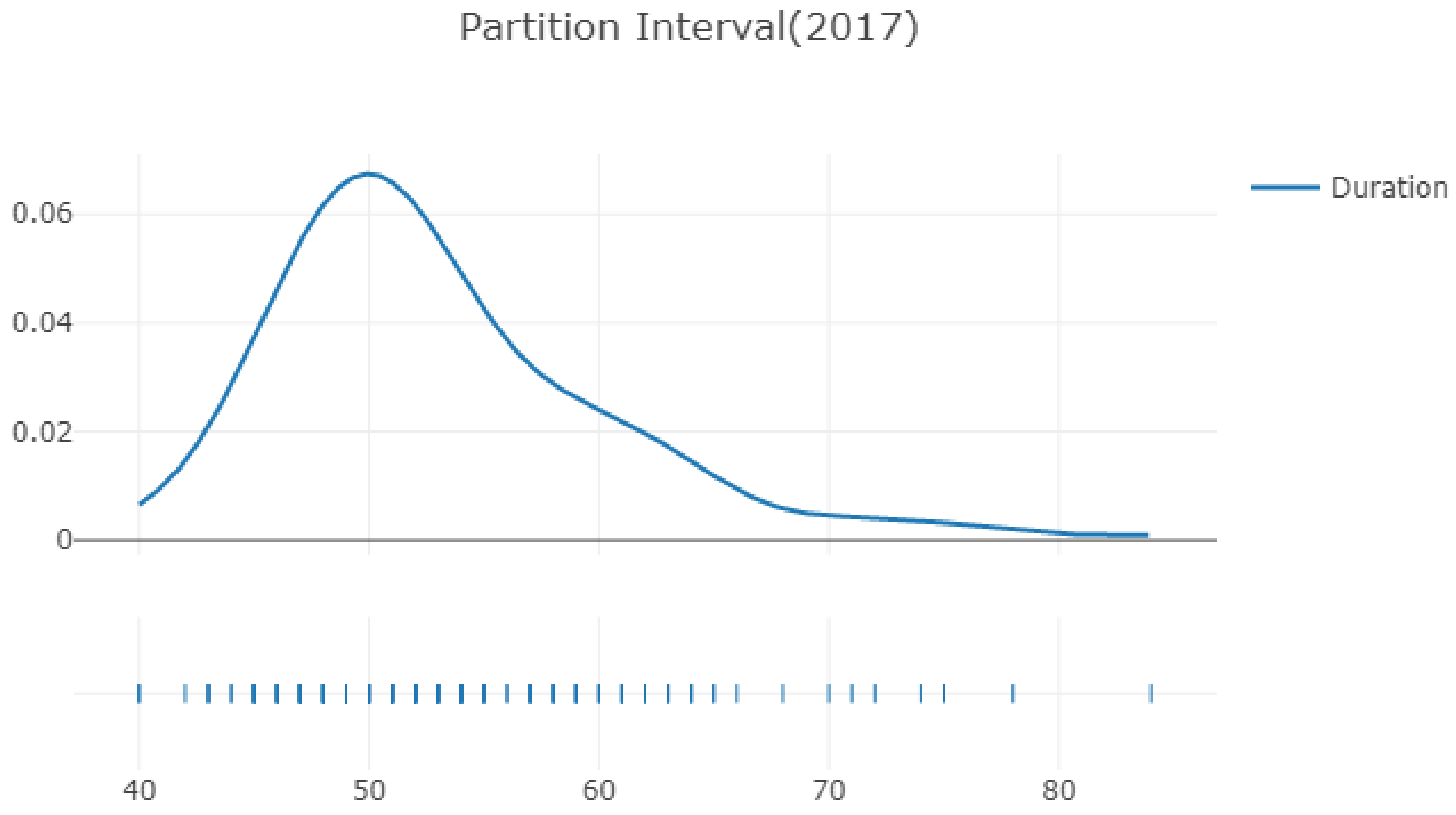
EDA: Analyzing Locations and Time Interval

	division	prev_division	difference	difference(hours)
88	2018-06-21 09:42:44	2018-06-19 01:44:01	2 days 07:58:43	56.0
89	2018-06-23 14:53:44	2018-06-21 09:42:44	2 days 05:11:00	53.0
90	2018-06-26 09:14:23	2018-06-23 14:53:44	2 days 18:20:39	66.0
91	2018-06-28 15:06:40	2018-06-26 09:14:23	2 days 05:52:17	54.0
92	2018-12-21 16:39:43	2018-06-28 15:06:40	176 days 01:33:03	4226.0

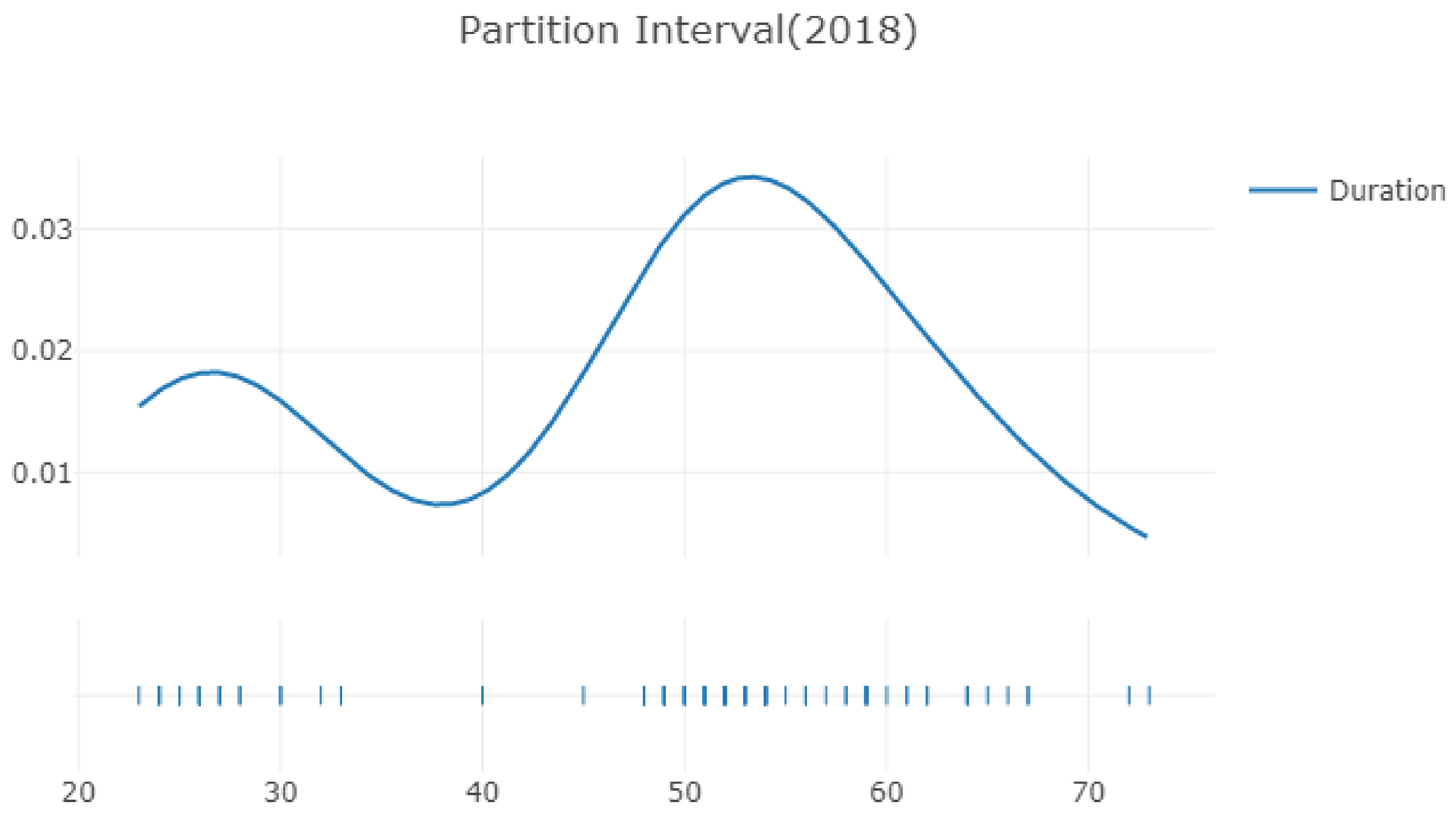
EDA: Analyzing Locations and Time Interval



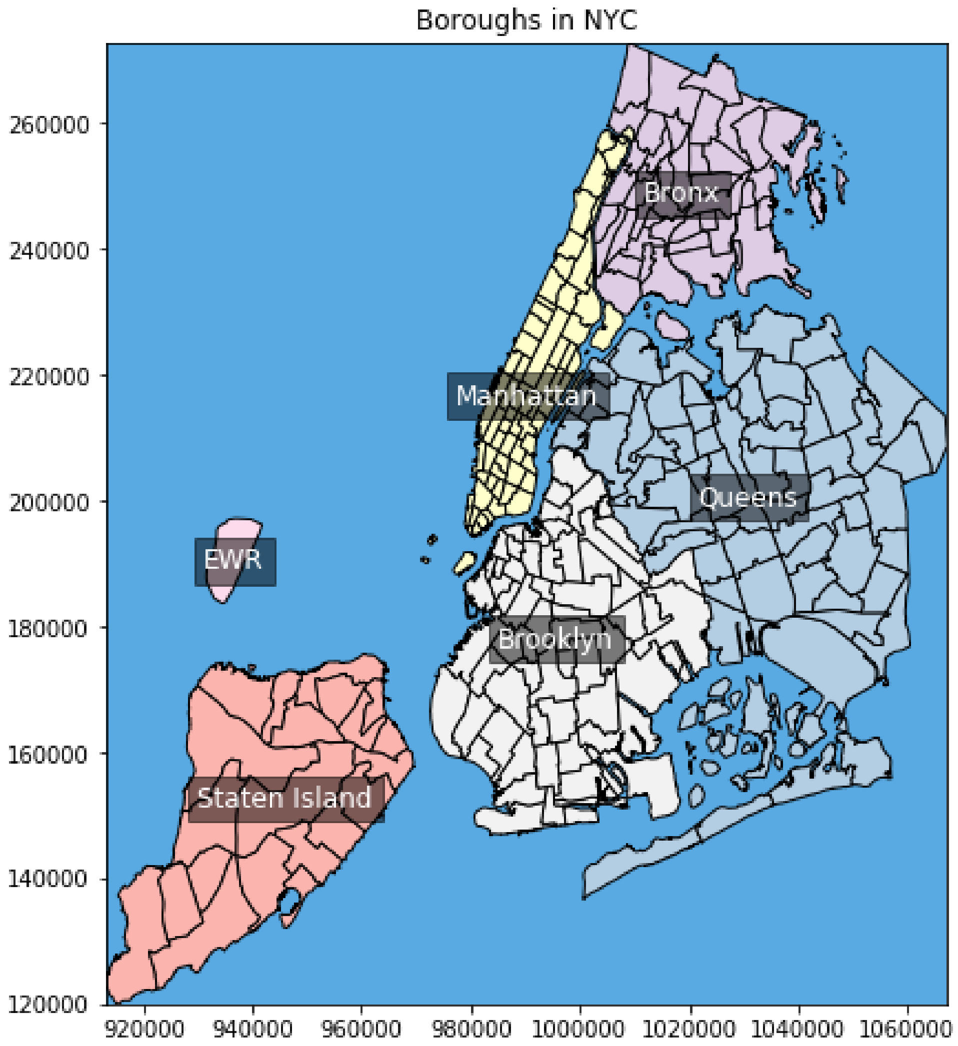
EDA: Analyzing Locations and Time Interval

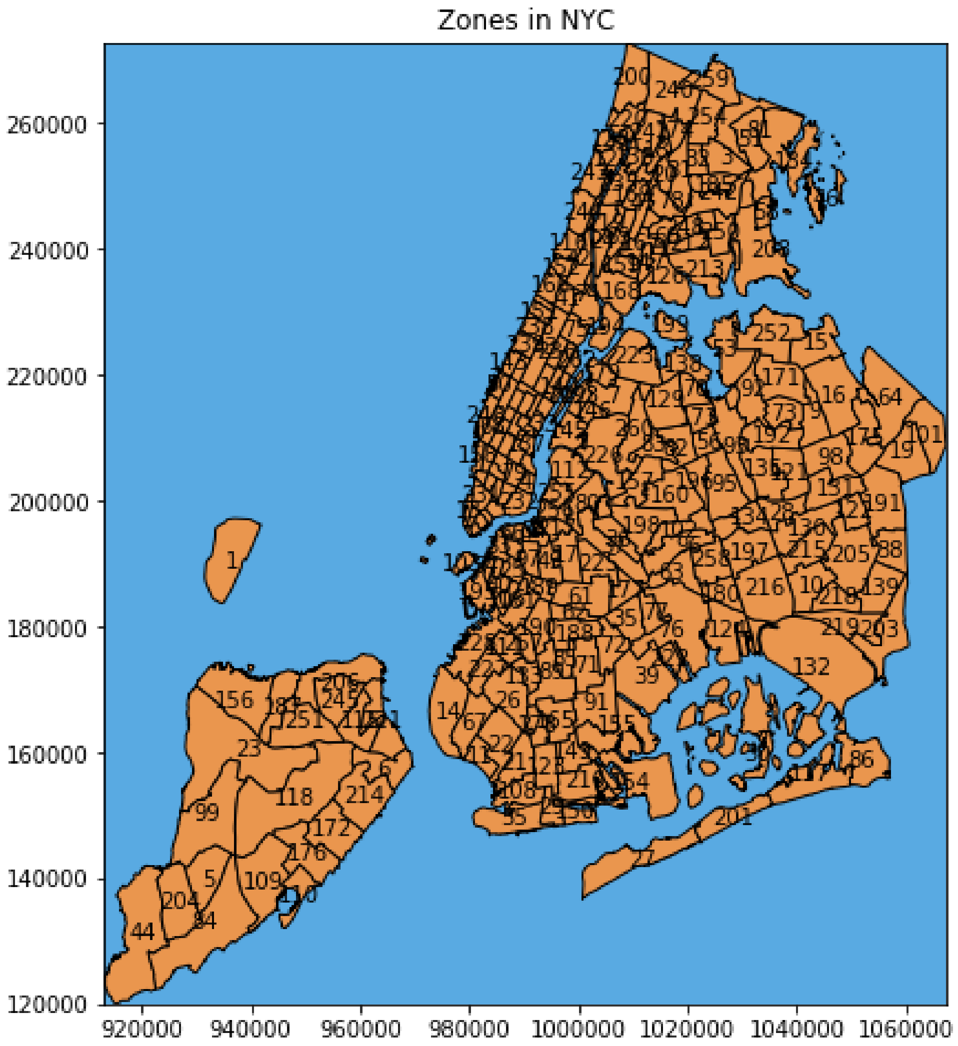


EDA: Analyzing Locations and Time Interval



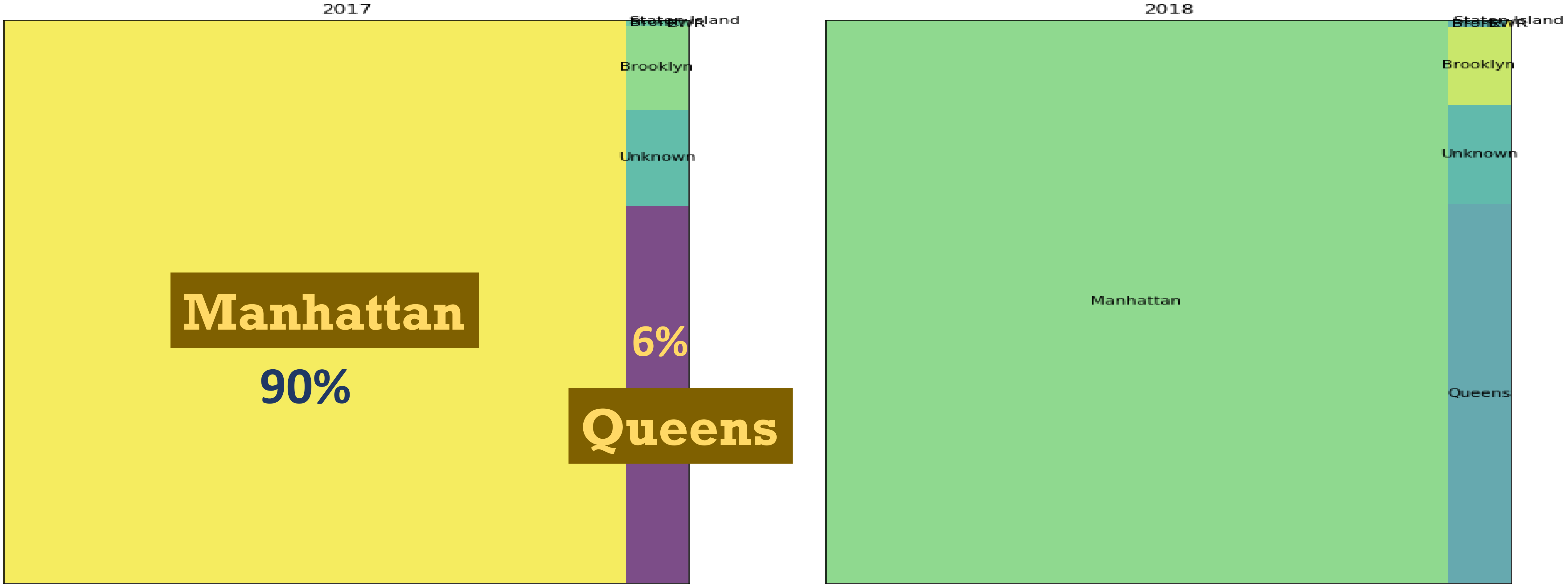
EDA: Analyzing Locations and Time Interval





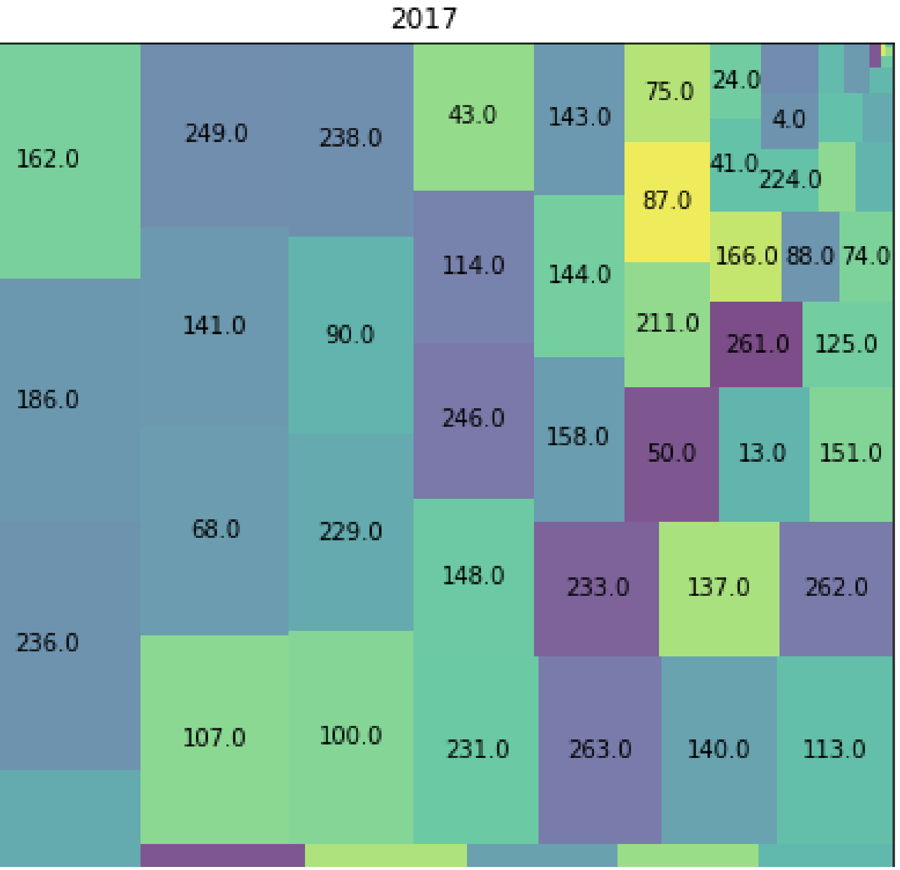
EDA: Analyzing Locations and Time Interval

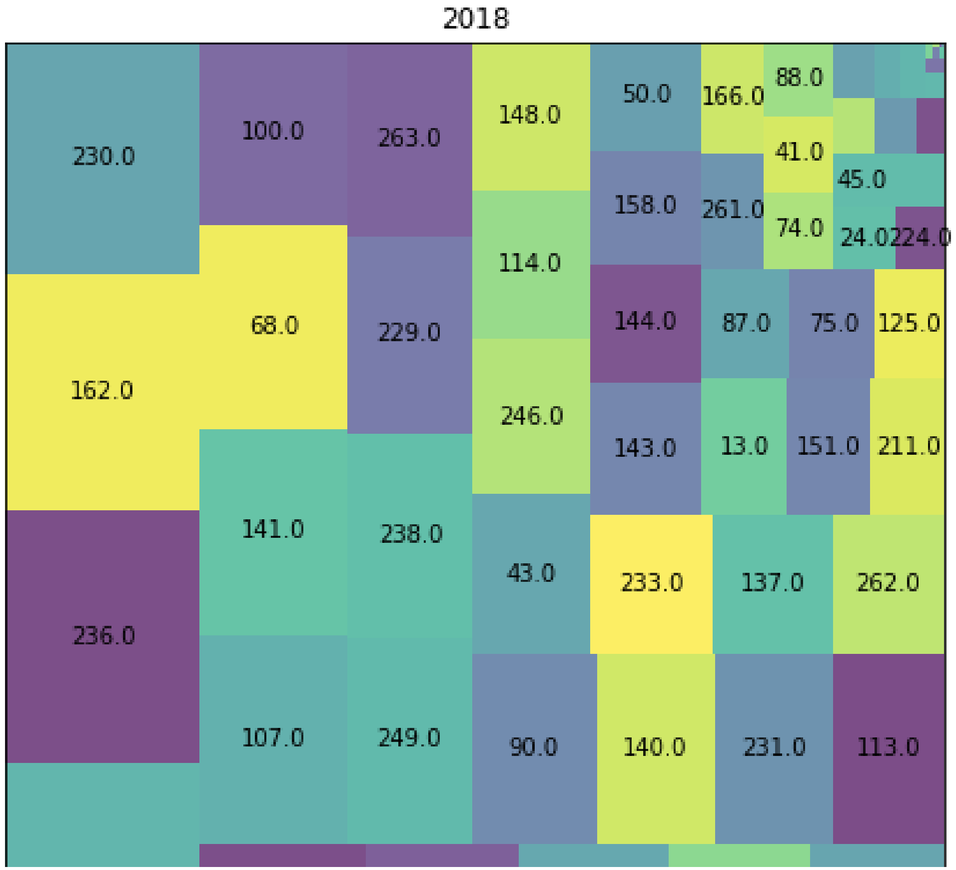
Number of Trips



EDA: Analyzing Locations and Time Interval

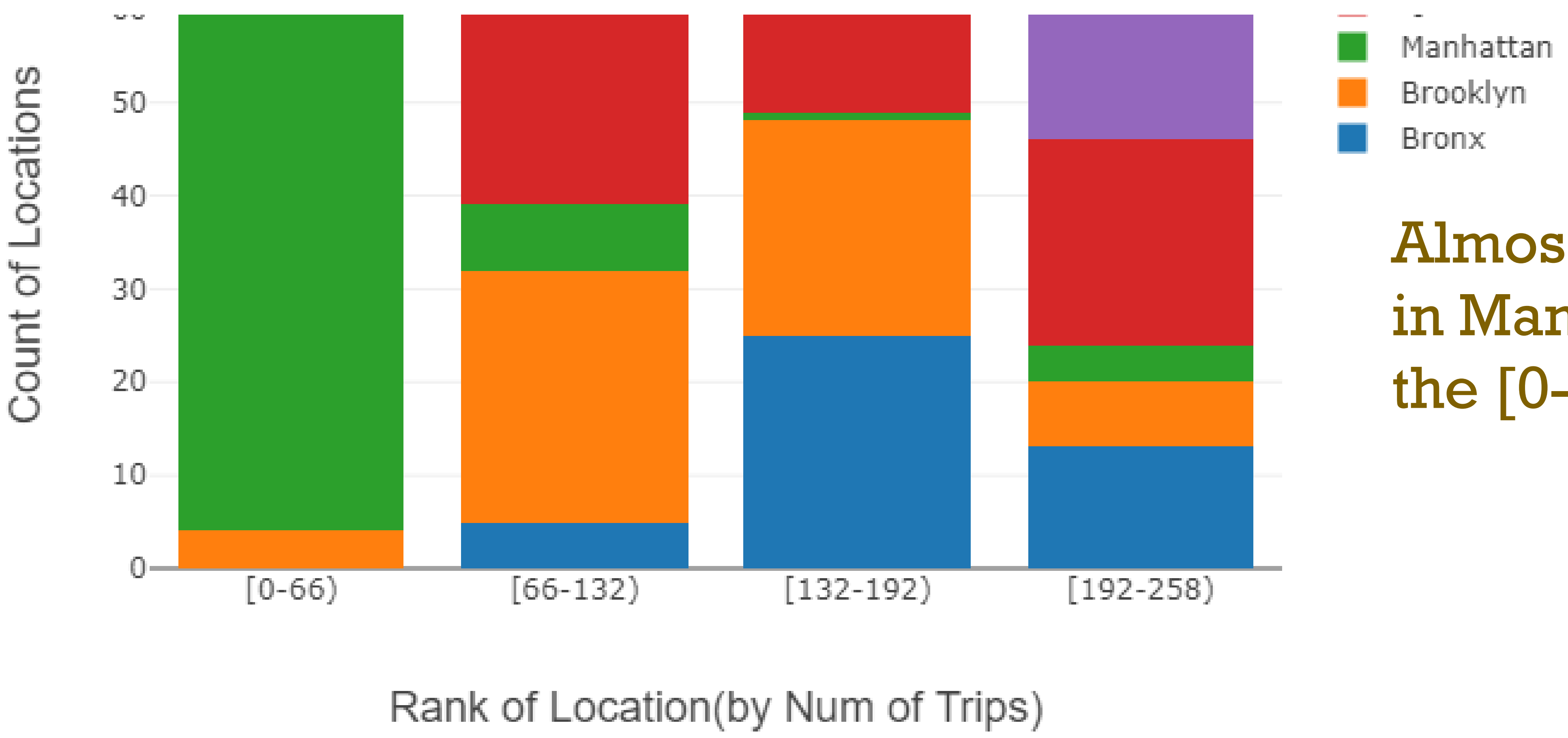
Trips in Manhattan





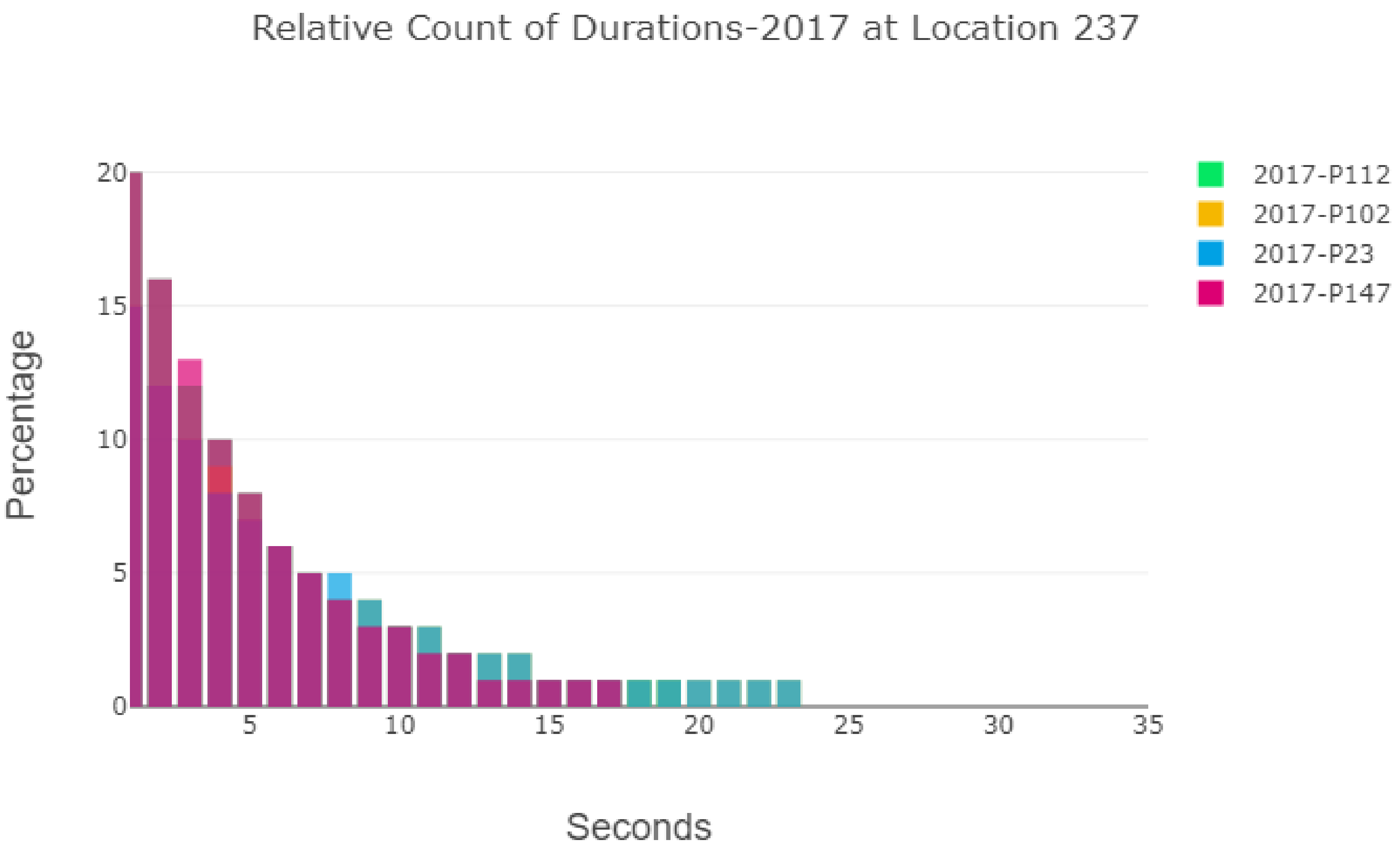
95 % of trips that originated from Manhattan ended in Manhattan.

EDA: Analyzing Locations and Time Interval



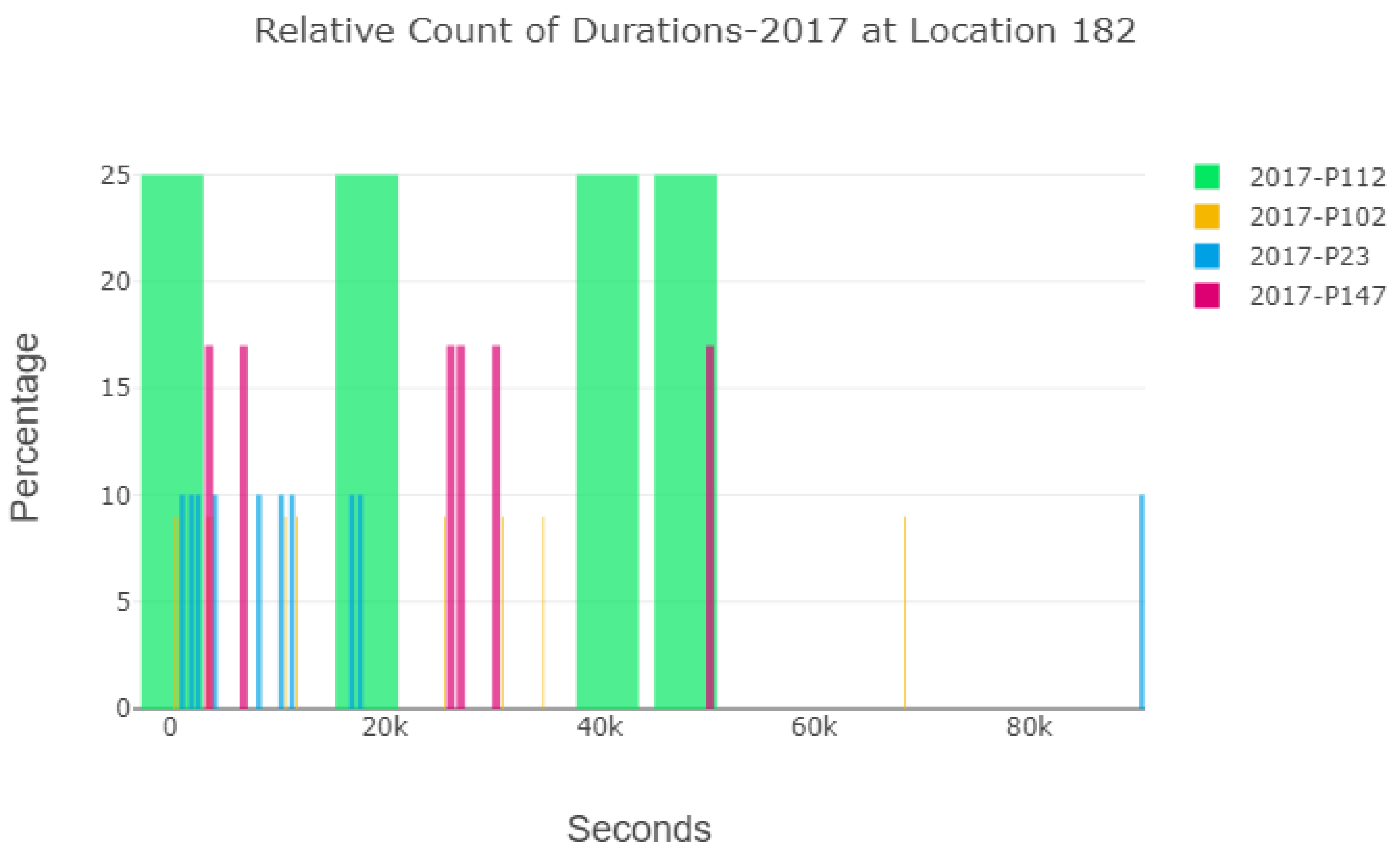
Almost all locations in Manhattan are in the [0-66] bar

EDA: Analyzing Locations and Time Interval



There are trips every 25 seconds at most at location 237.

EDA: Analyzing Locations and Time Interval



Trip interval goes up to 90k (25 hours) for location 182

Data Cleaning and Feature Selection

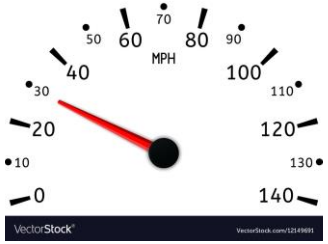
Passenger Count

Remover passenger counts which are greater than 6 or less than zero

Trip Distance / Fares

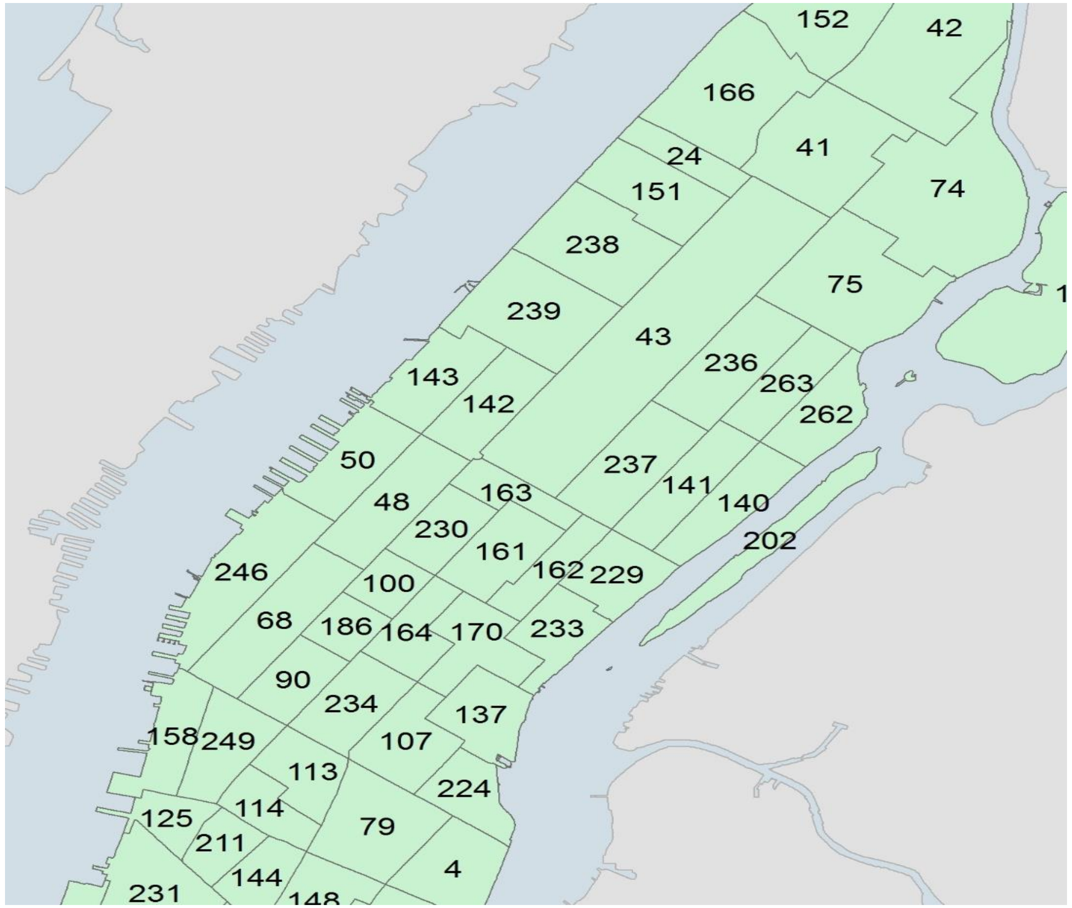
Standard fare only. Remove negative or unreasonable distances

Duration/Speed



Keep durations which are greater than 0

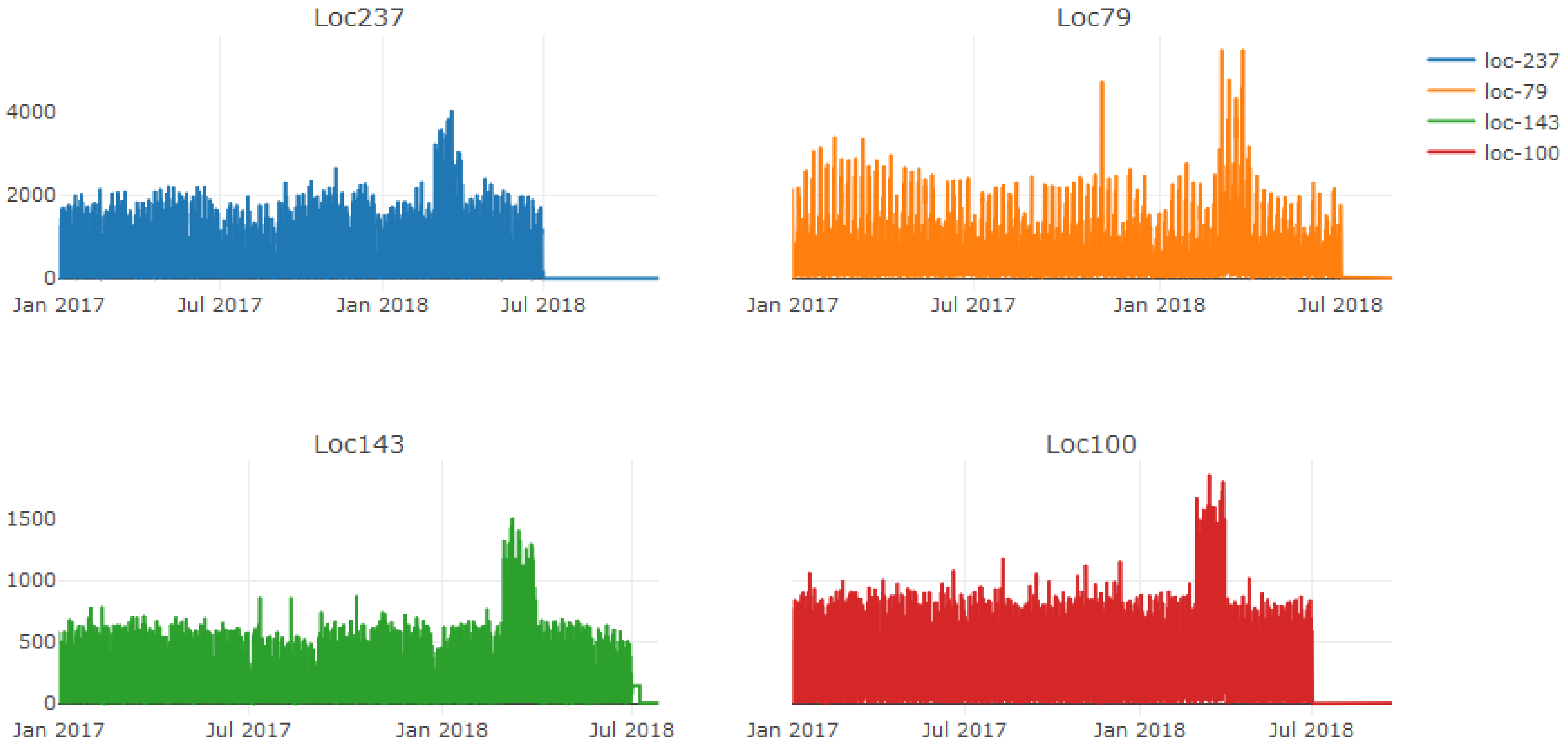
Data Aggregation



- The average trip duration in Manhattan is around 10 minutes
- 75% of the trips are less than 2 miles
- Group by location and resample every 10 minutes
- Apply sum to fares_amout, passenger_count and trip_distanc

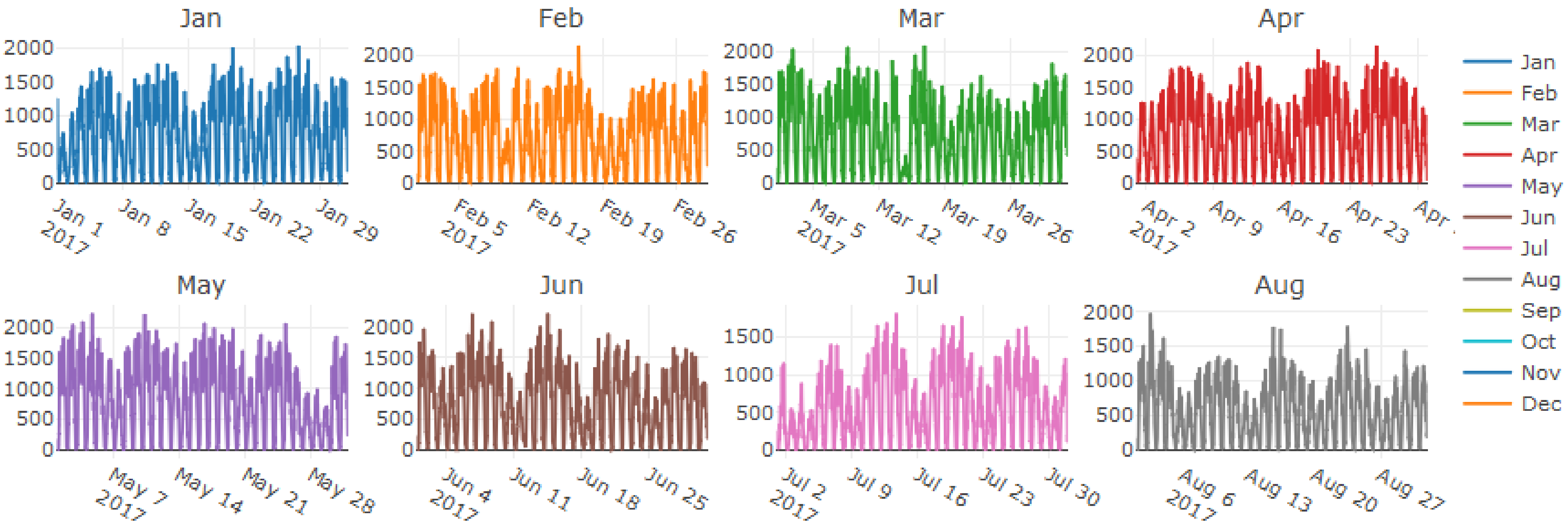
Univariate time-series Forecasting

Overall



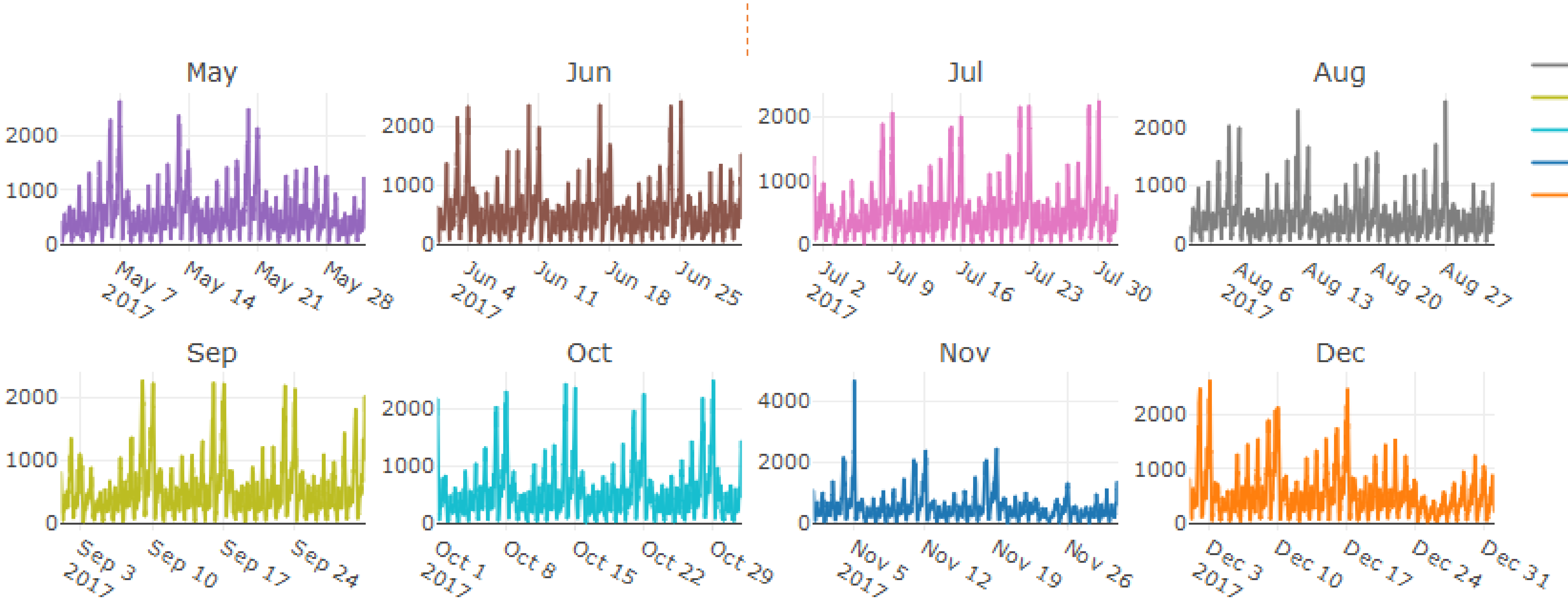
Univariate time-series Forecasting

Monthly-Loc 237



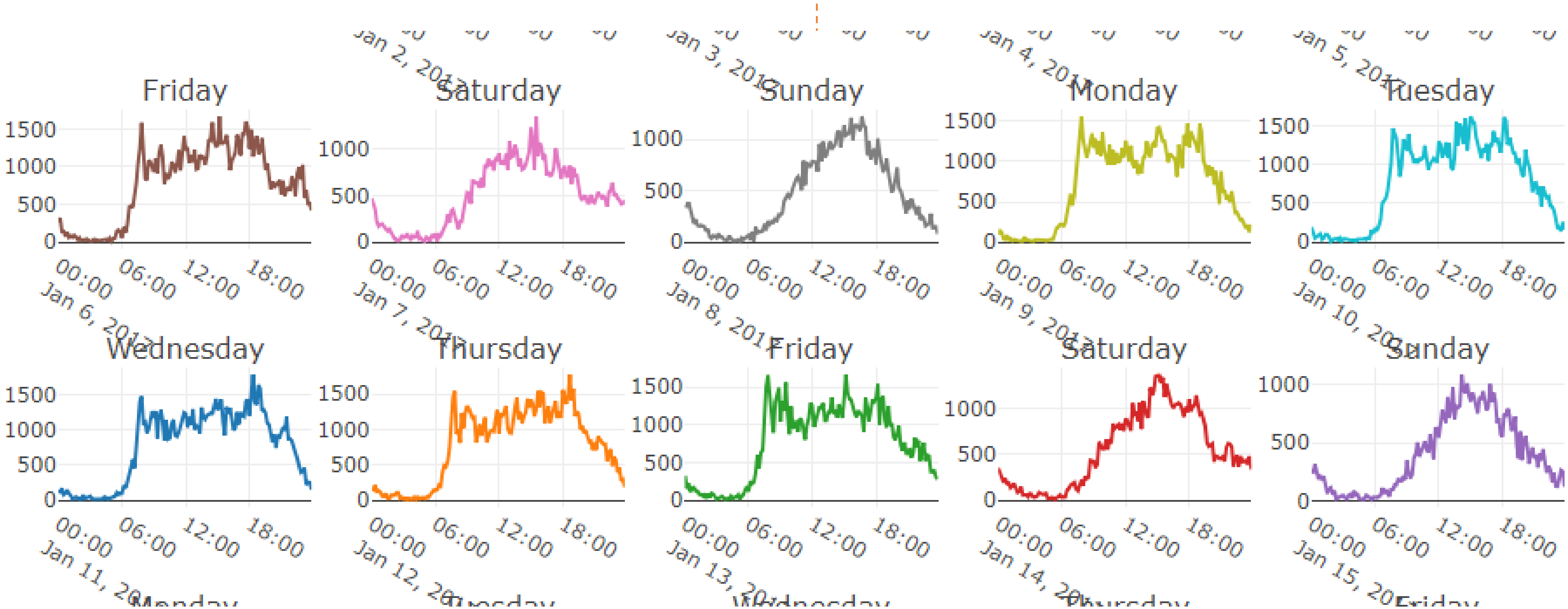
Univariate time-series Forecasting

Monthly- Loc 79



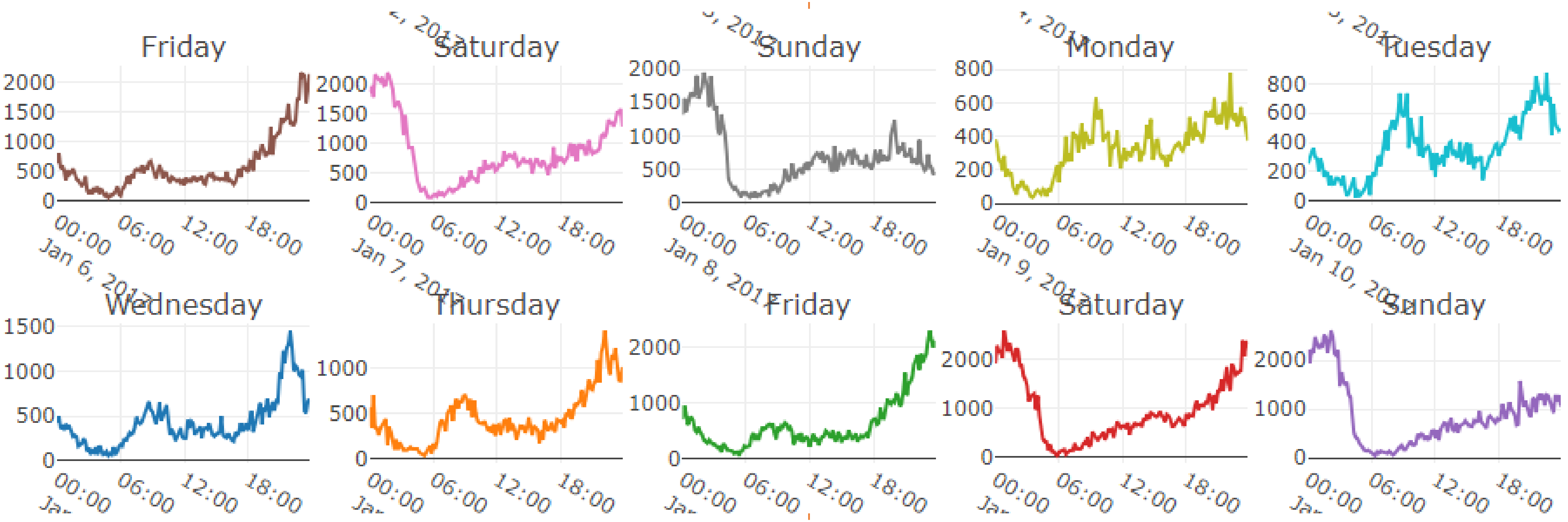
Univariate time-series Forecasting

Daily Pattern at Loc 237



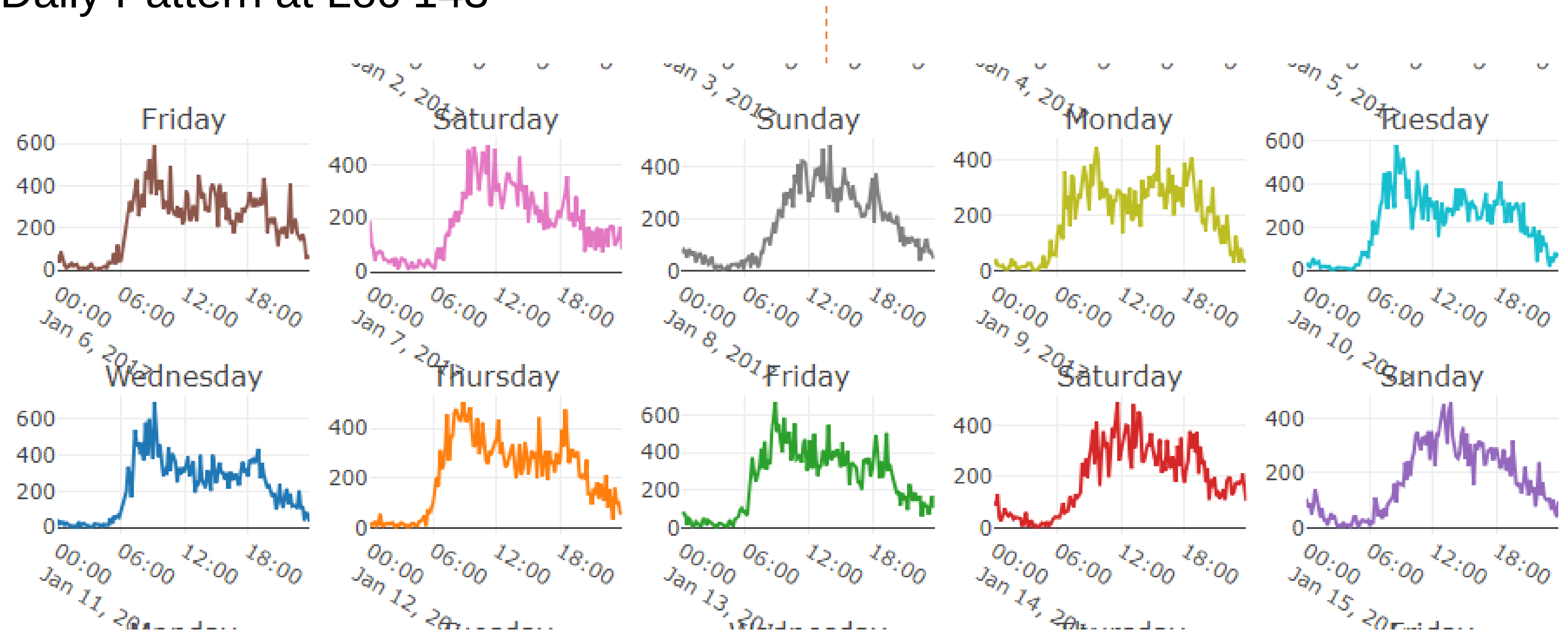
Univariate time-series Forecasting

Daily Pattern at Loc 79



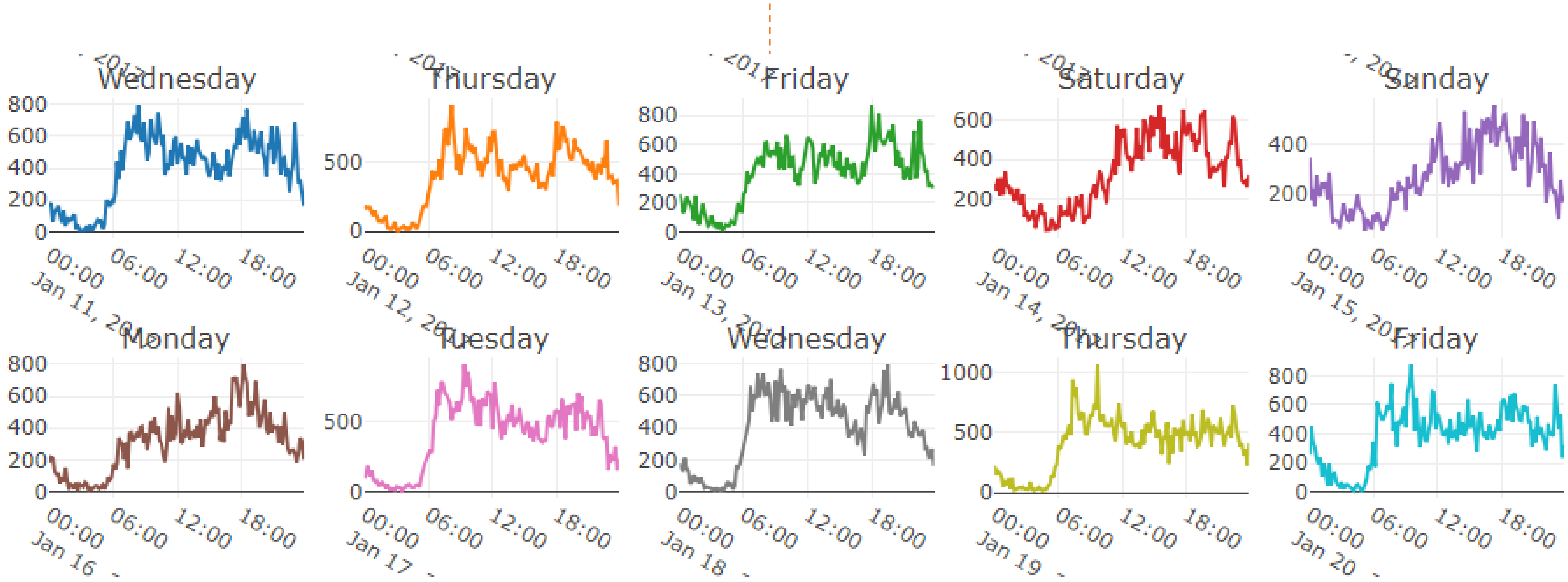
Univariate time-series Forecasting

Daily Pattern at Loc 143



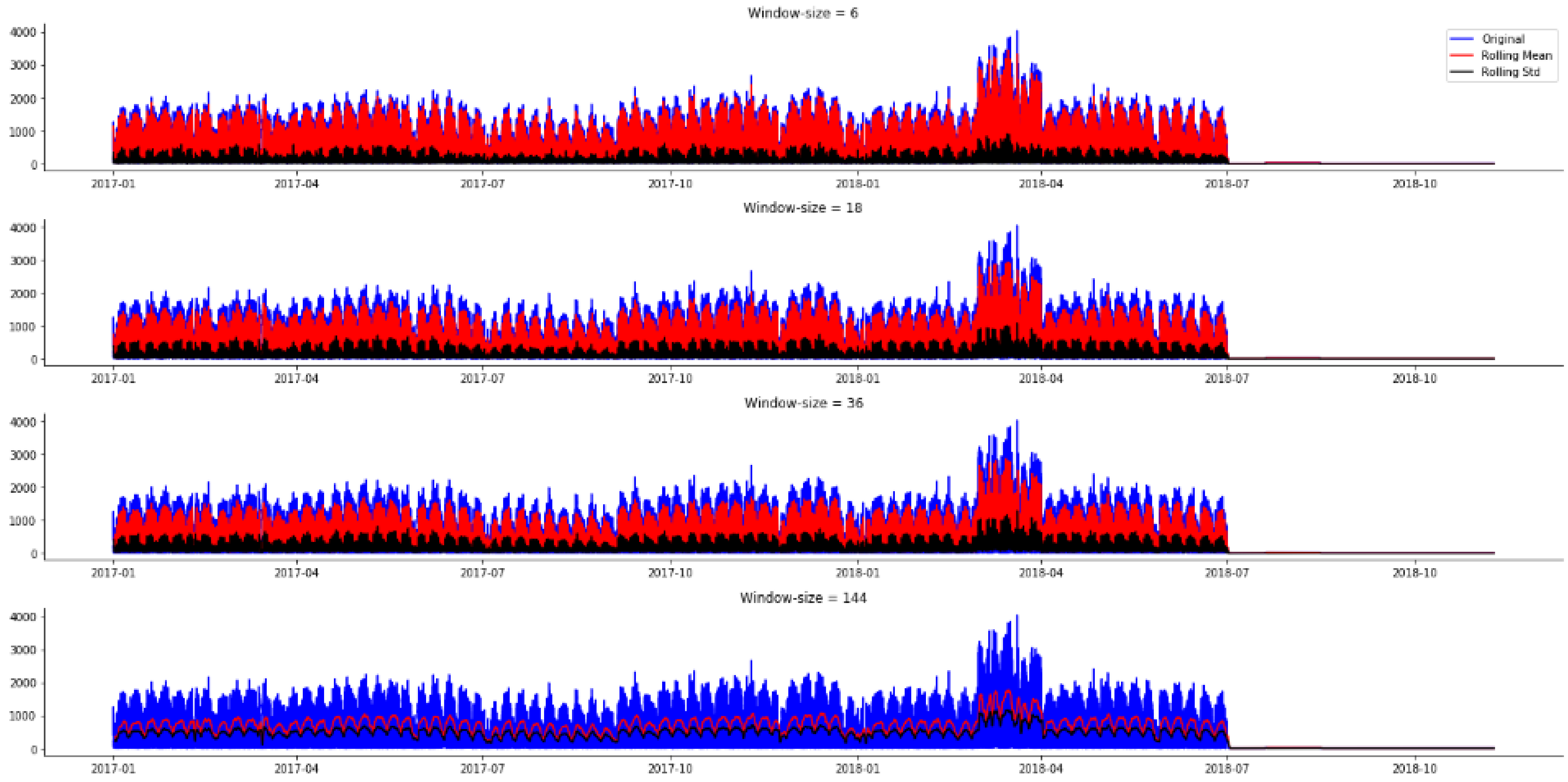
Univariate time-series Forecasting

Daily Pattern at Loc 100



Univariate time-series Forecasting

Rolling Statistics



Test of Stationarity: ADF Test

```
Test Statistic                    -28.128789
p value                             0.000000
Flags used                         68.000000
Number of Observations Used     97421.000000
Critical value 1%                  -3.430417
Critical value 5%                  -2.861570
Critical value 10%                 -2.566786
dtype: float64
```

```
Test Statistic                    -38.411830
p value                             0.000000
Flags used                         65.000000
Number of Observations Used     82317.000000
Critical value 1%                  -3.430429
Critical value 5%                  -2.861575
Critical value 10%                 -2.566789
dtype: float64
```

```
perform_dicky_fuller_test(loc143_df.fare_amount)
```

```
1
365
{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0}
```

Test of Stationarity: ADF Test

p-value > 0.05: Accept H0, the data has a unit root and is non-stationary
p-value ≤ 0.05: Reject H0. the data does not have a unit root and is stationary

```
Test Statistic                   -28.128789
p value                            0.000000
Flags used                        68.000000
Number of Observations Used    97421.000000
Critical value 1%                 -3.430417
Critical value 5%                 -2.861570
Critical value 10%                -2.566786
dtype: float64
```

```
Test Statistic                   -38.411830
p value                            0.000000
Flags used                        65.000000
Number of Observations Used    82317.000000
Critical value 1%                 -3.430429
Critical value 5%                 -2.861575
Critical value 10%                -2.566789
dtype: float64
```

```
perform_dicky_fuller_test(loc143_df.fare_amount)
```

```
1
365
{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0}
```

Kwiatkowski-Phillips Test

Null Hypothesis: The process is trend stationary.
Alternate Hypothesis: The series has a unit root (series is not

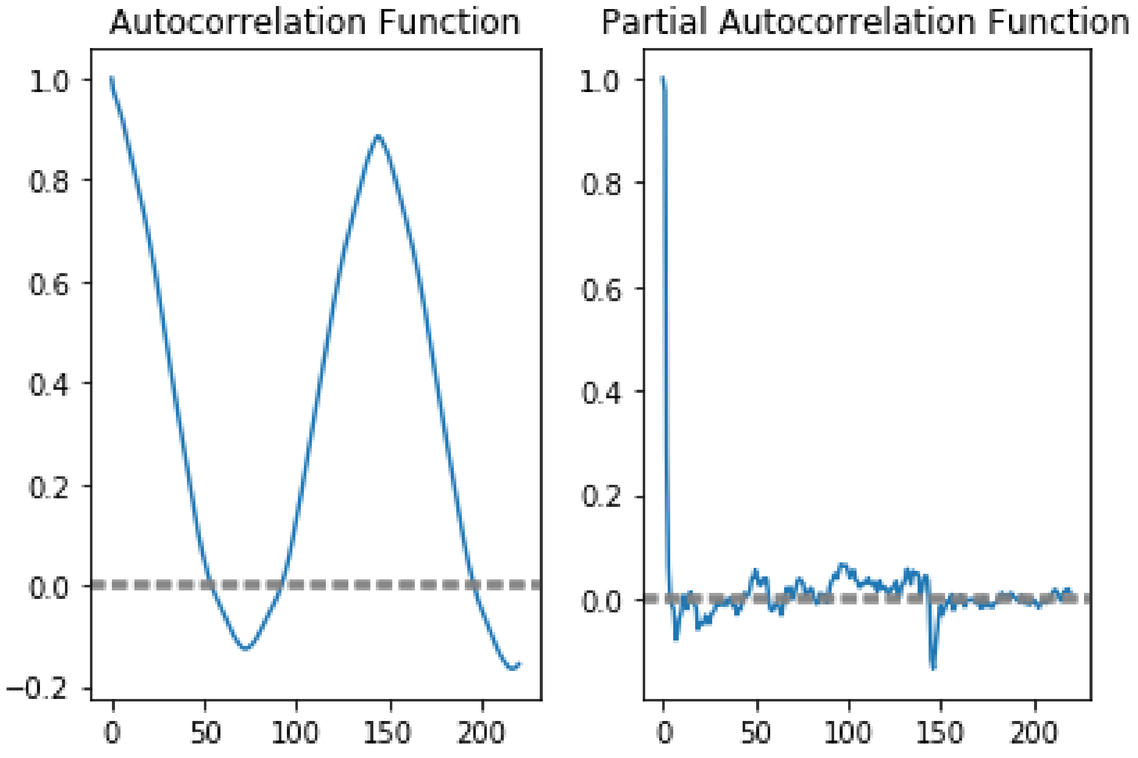
```
Results of KPSS Test:
Test Statistic            22.724934
p-value                    0.010000
Lags Used                 68.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
dtype: float64
```

```
Results of KPSS Test:
Test Statistic            20.584865
p-value                    0.010000
Lags Used                 66.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
dtype: float64
```

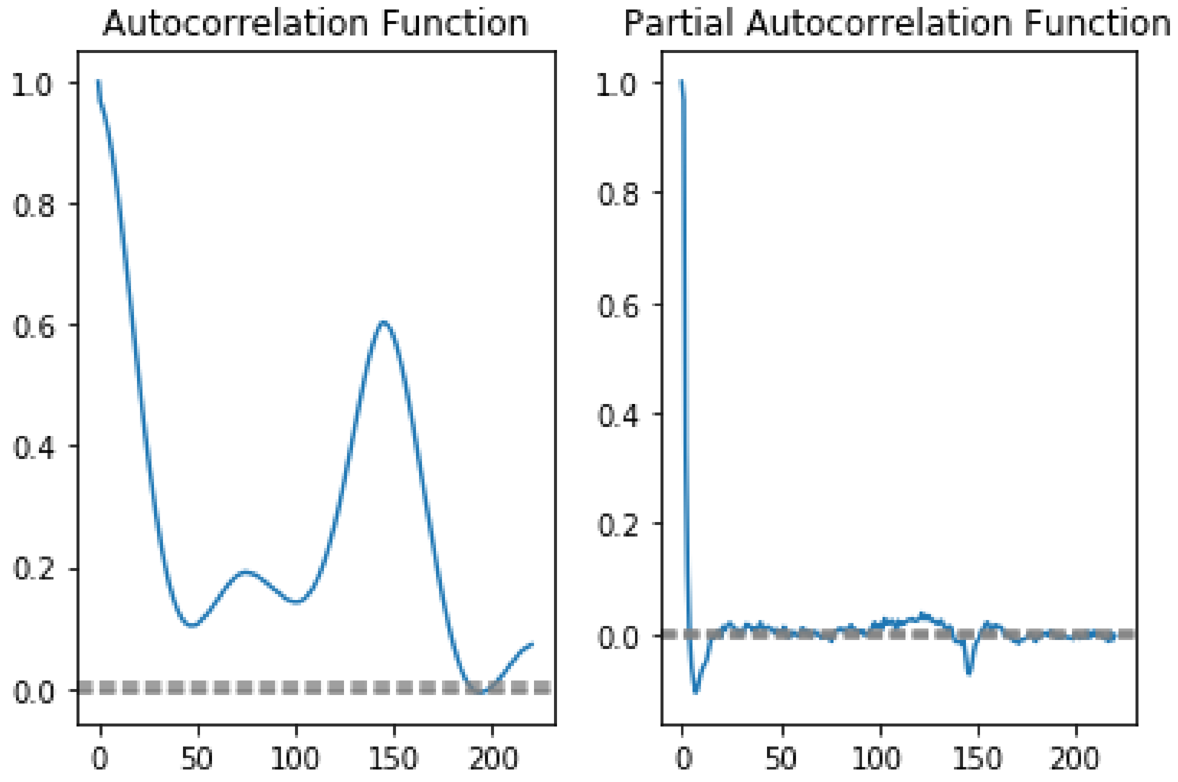
KPSS shows that the data is not stationary while ADF says it is. ADF measures difference stationarity while KPSS measures trend stationarity. ARIMA differencing can remove at least a linear trend.

PACF and ACF plots

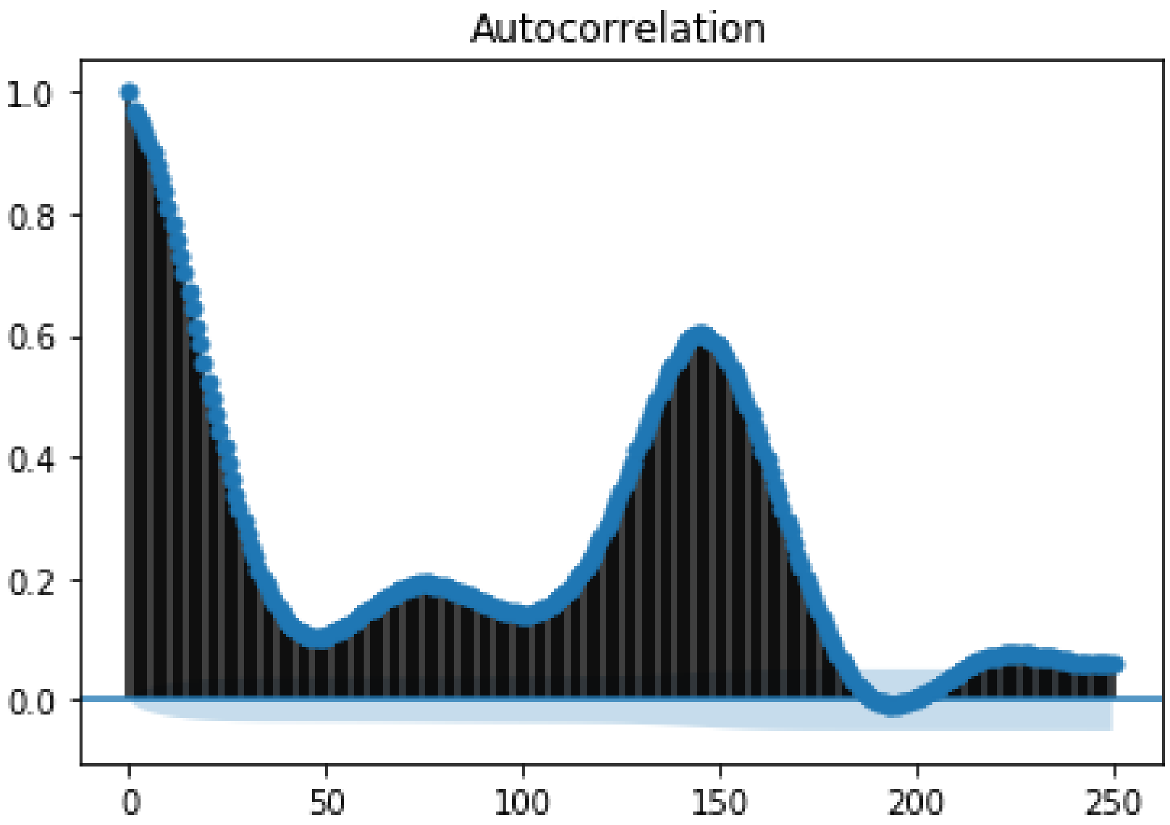
```
plot_acf_pacf(loc237_df.fare_amount, lags=220)
```

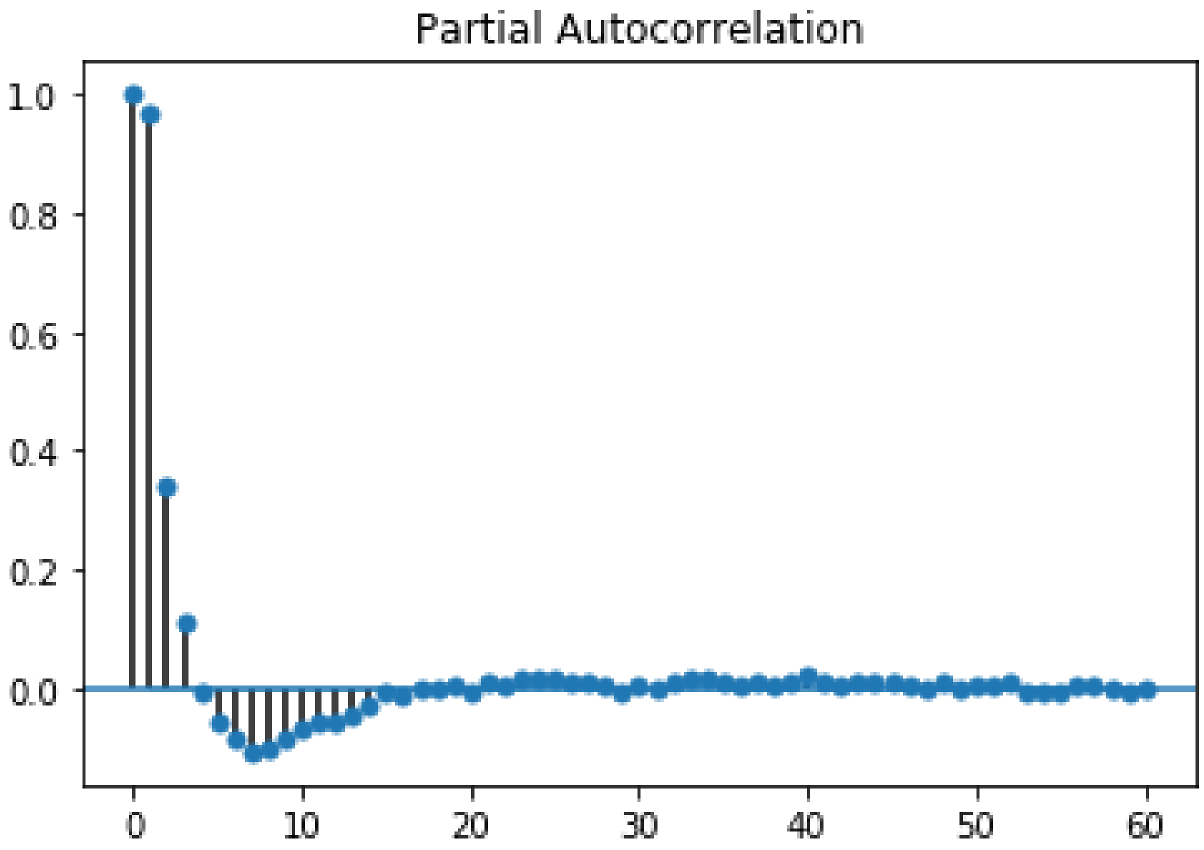



```
plot_acf_pacf(loc79_df.fare_amount, lags=220)
```



PACF and ACF plots





PACF and ACF plots

Autocorrelation plot shows that the number of lags(q) for MA shall be somewhere between 50 and 60 for location 237 and between 195 and 205 for location 79.

Partial auto correlation for both locations shows that p is between 0 and 5

Grid search for the best hyperparameters based on AIC

```
(3, 0, 9)
656091.7336373809
```

Walk Forward Validation

Split the dataset into training and test sets.
Walk the time steps in the test dataset.

Train an ARIMA model.

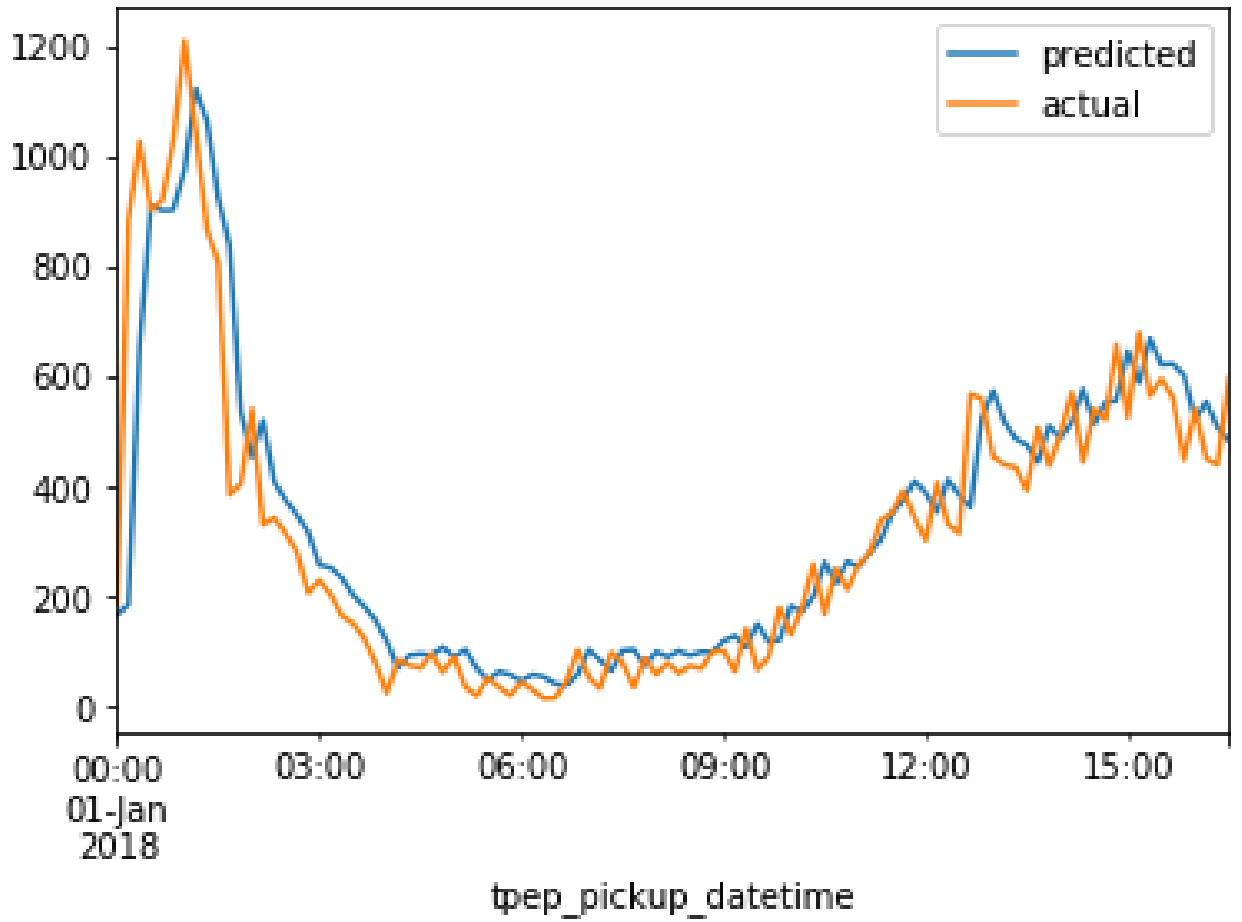
Make a one-step prediction.

Store prediction; get and store actual observation.

Calculate error score for predictions compared to expected values.

The model is trained on 2017 data and tested on 2018 data.

RMSE $113

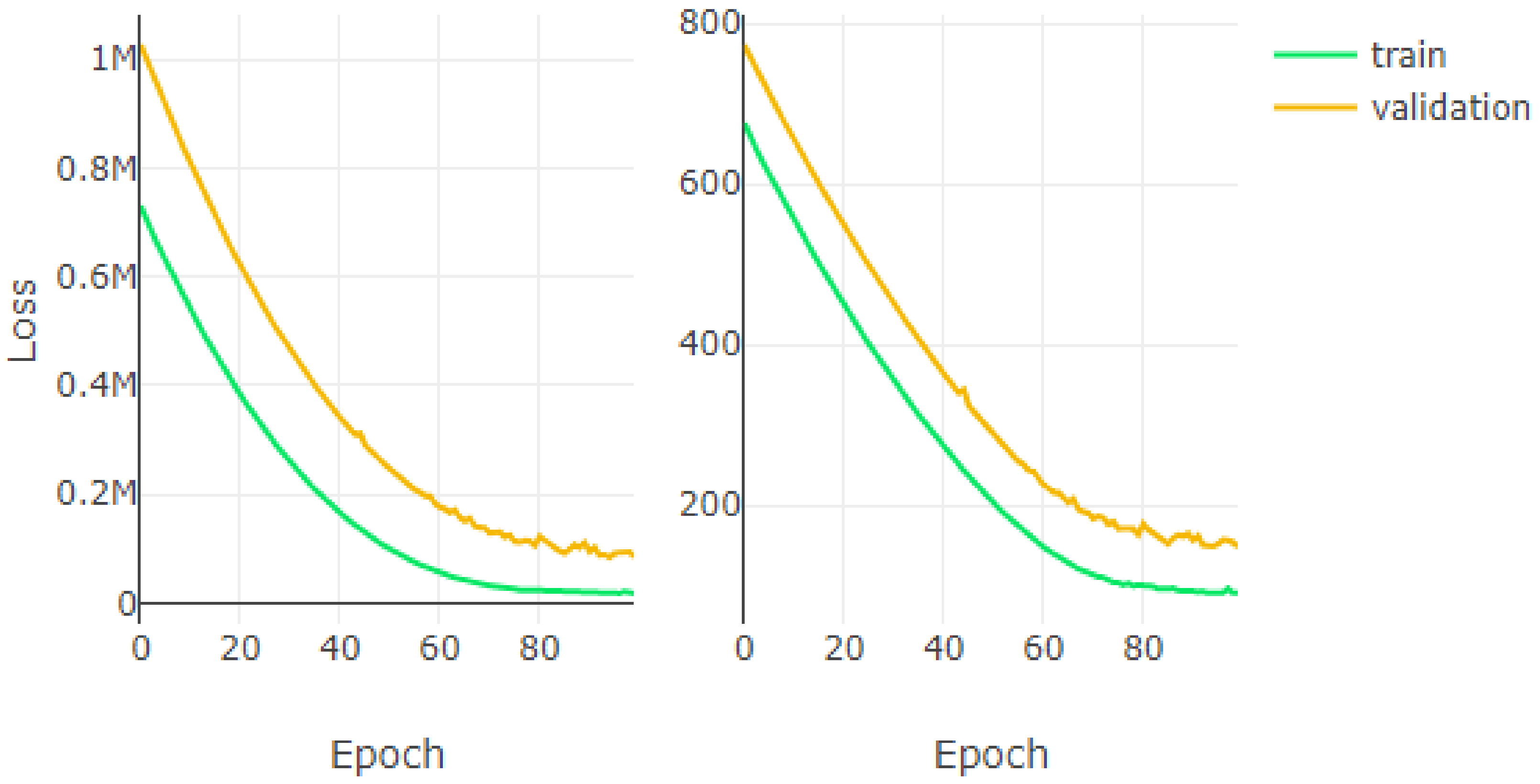


LSTM Networks

Vanilla LSTM(24 steps back)

```
(52536, 24, 1) (52536, 1)
(26040, 24, 1) (26040, 1)
90.41663697046542,

147.8938028713525
```

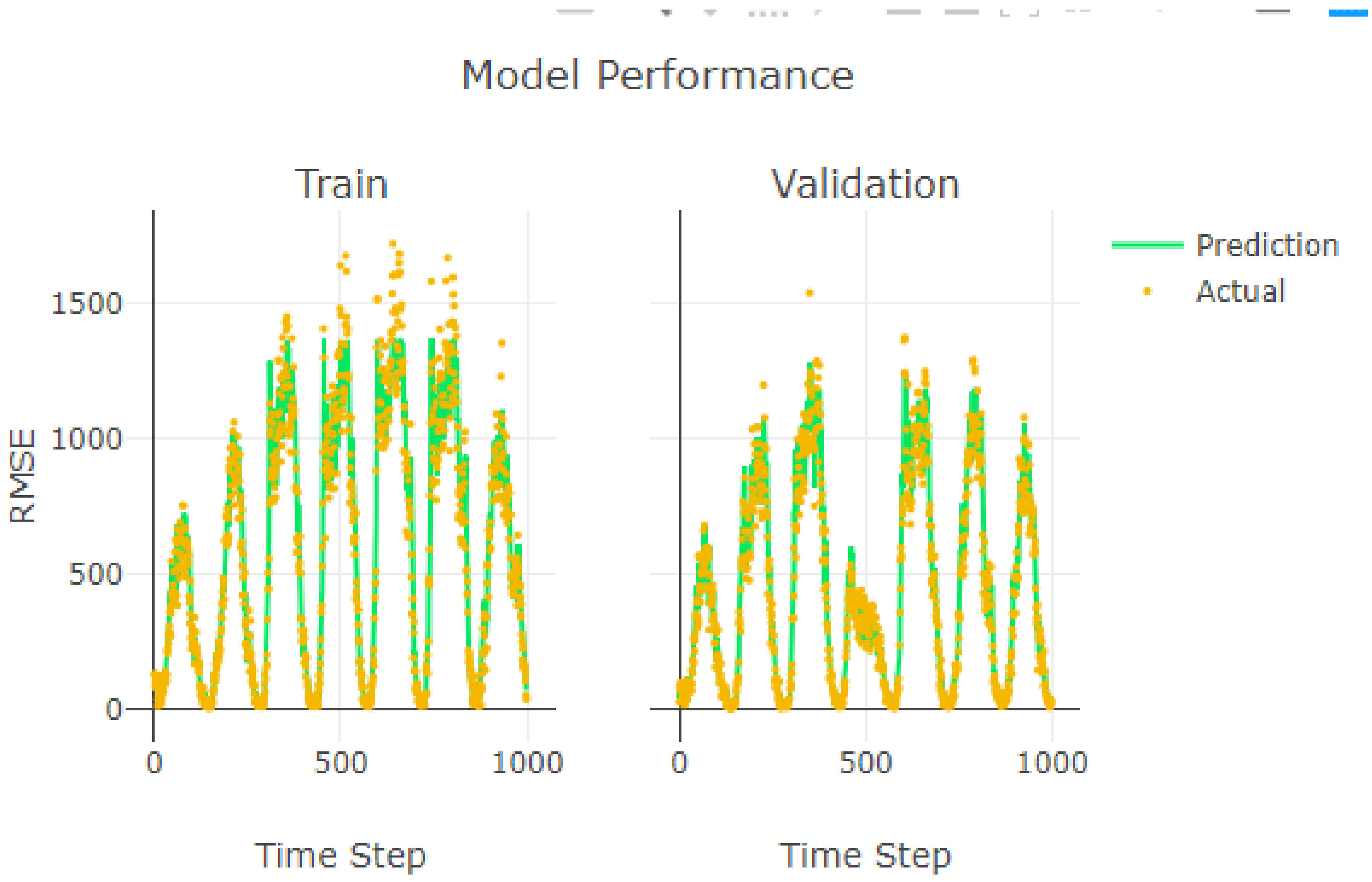
Is the model learning?



LSTM Networks

Vanilla LSTM(24 steps back)

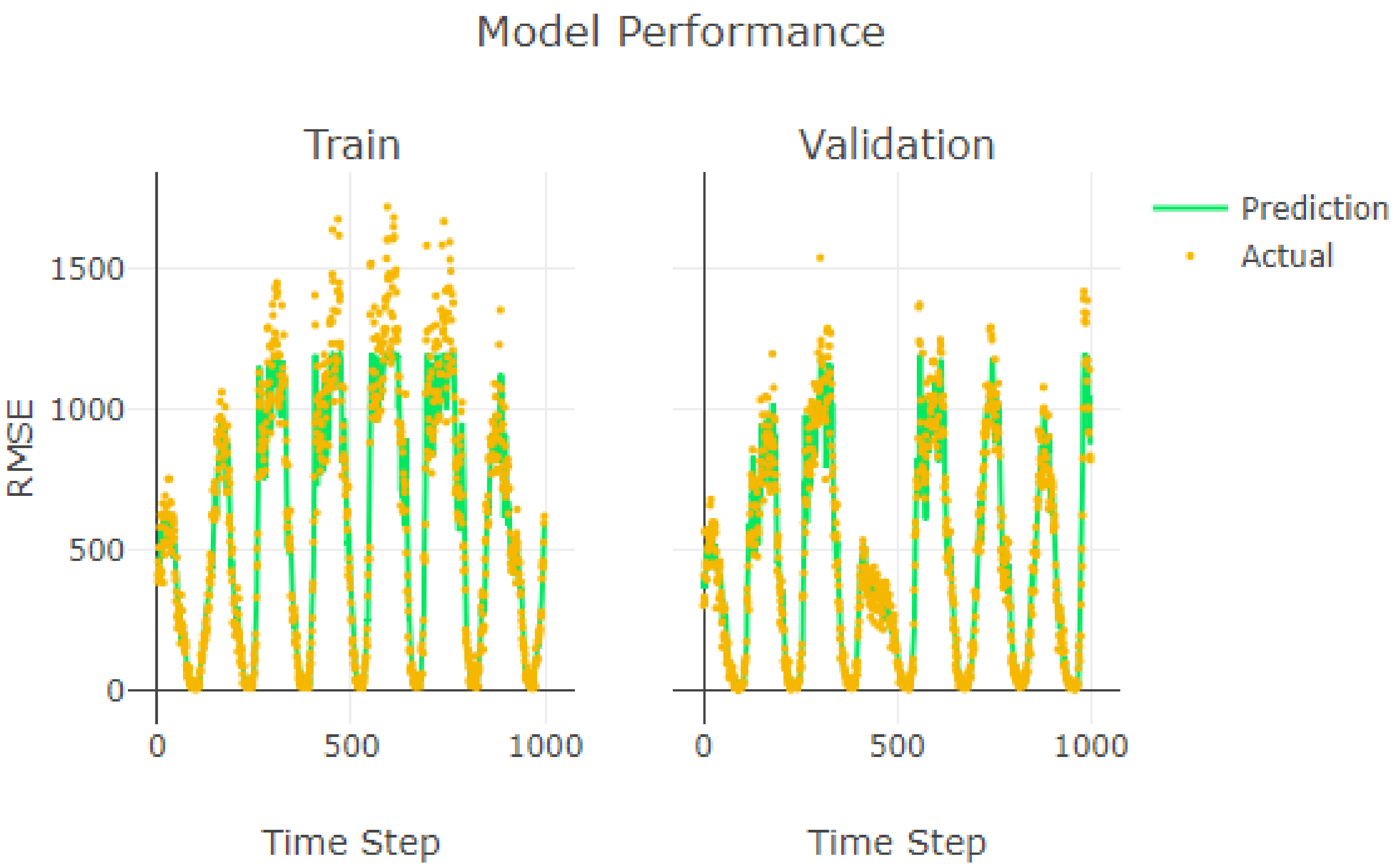
```
(52536, 24, 1) (52536, :
(26040, 24, 1) (26040, :
90.41663697046542,

147.8938028713525
```



LSTM Networks

Vanilla LSTM(72 steps back)

```
(52536, 72, 1) (52536, 1)
(26040, 72, 1) (26040, 1)
102.62558234688204,
171.80584667366665
```

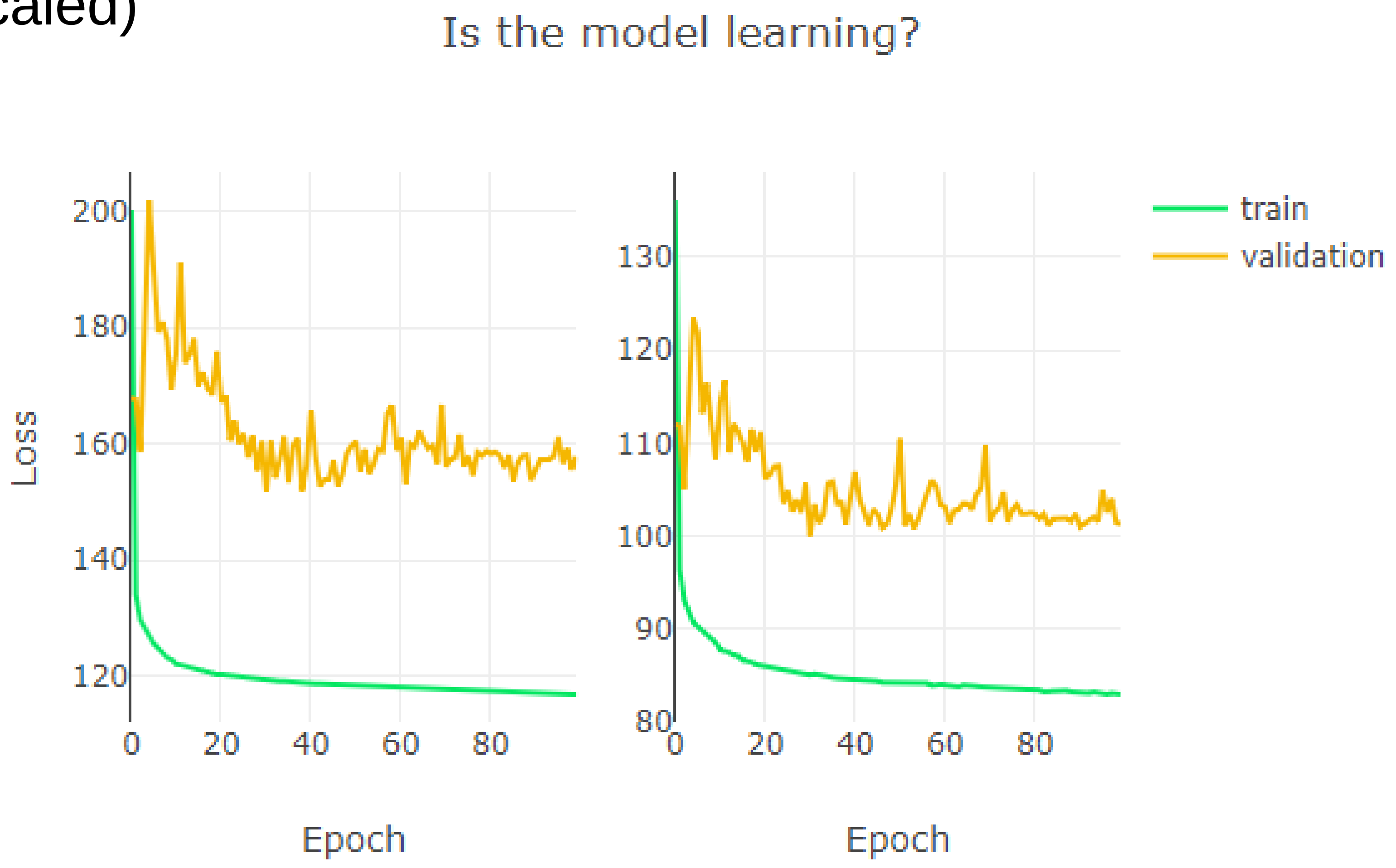


LSTM Networks

Vanilla LSTM(24 steps back, rescaled)

0.030218334914499517,
0.03710274413634326
[[82.94120754]],

[[101.26750489]]

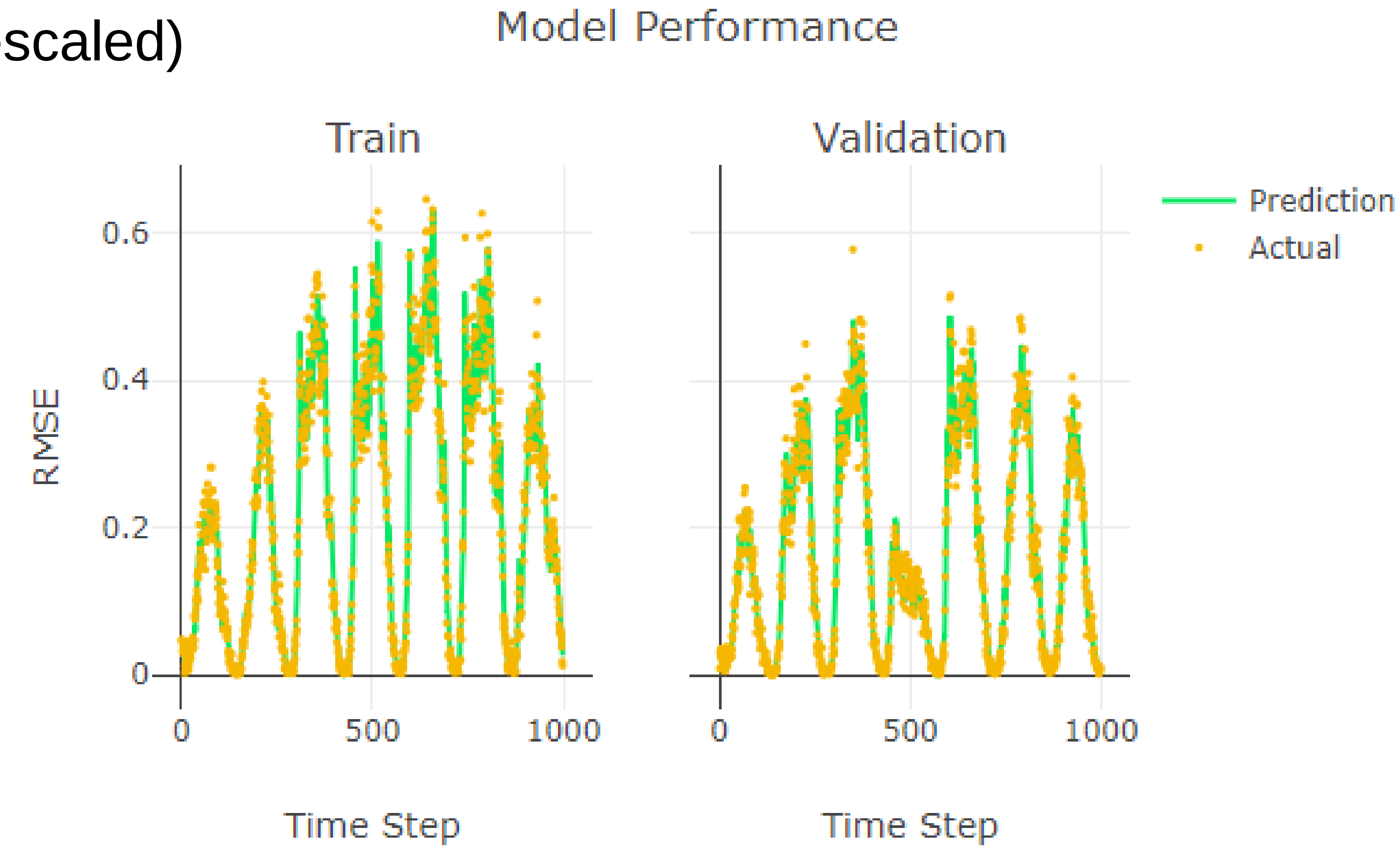


LSTM Networks

Vanilla LSTM(24 steps back, rescaled)

0.030218334914499517,
0.03710274413634326
[[82.94120754]],

[[101.26750489]]



Conclusion

- Stacked LSTM performed similar to Vanilla LSTM($100 - $109), more hyperparameter tuning is needed
- Generalized model RMSE is $250
- Consider Drop-off location
- Sequence to Sequence Prediction
- Integrate it with other ride-sharing time-series

Project-Link: Github

References

Tom Augspurger
TomAugspurger

Analytics Vidhya
Learn everything about analytics

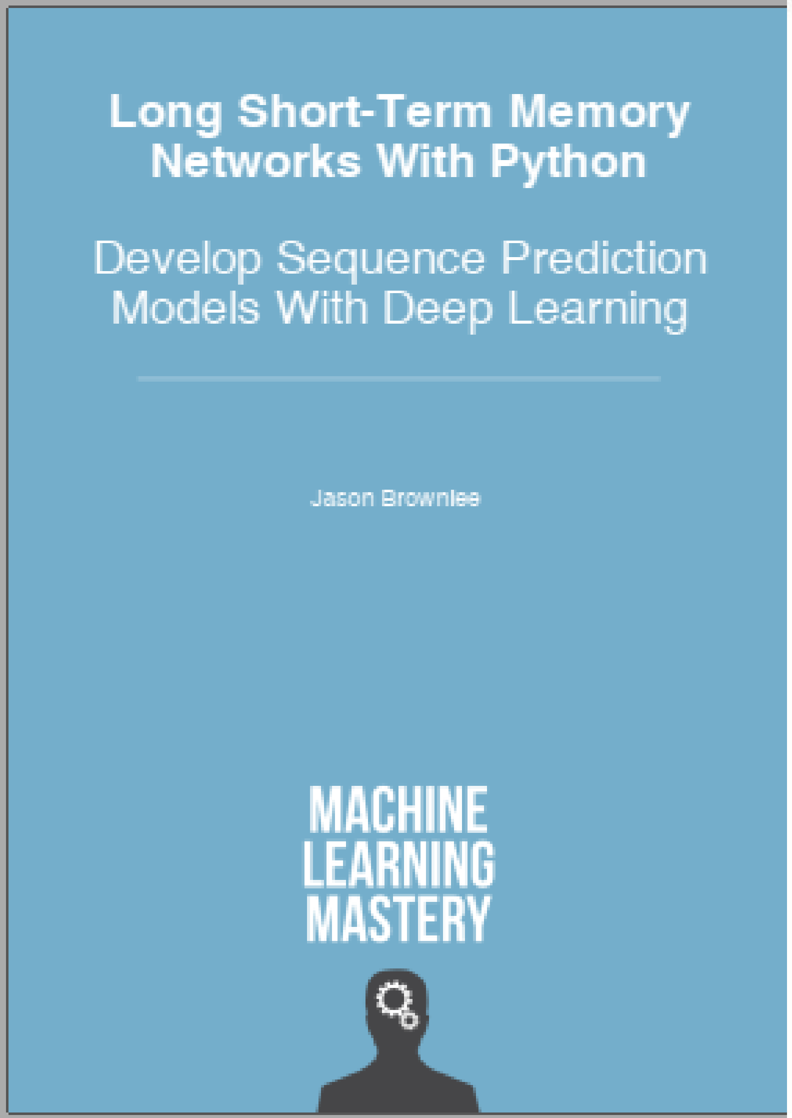
Towards Data Science

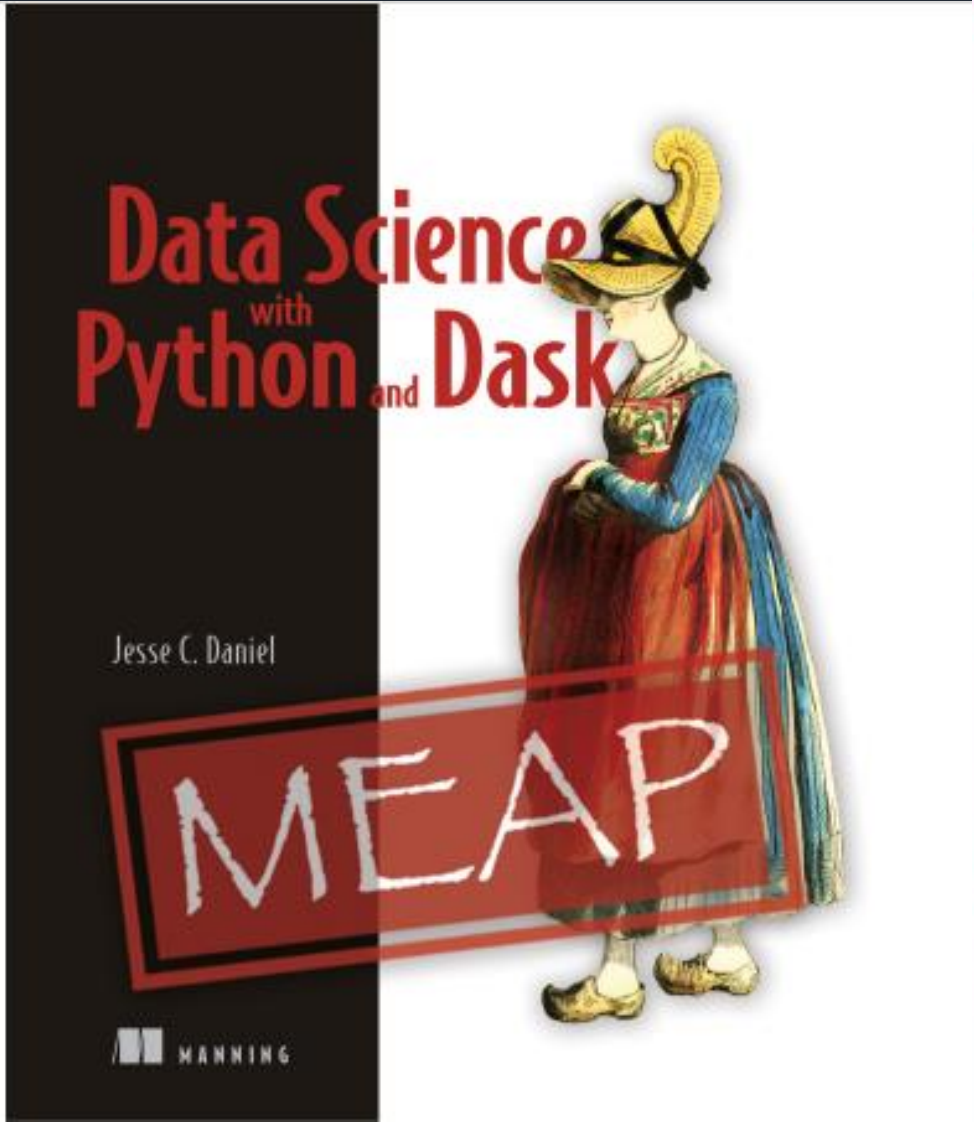
MACHINE LEARNING MASTERY

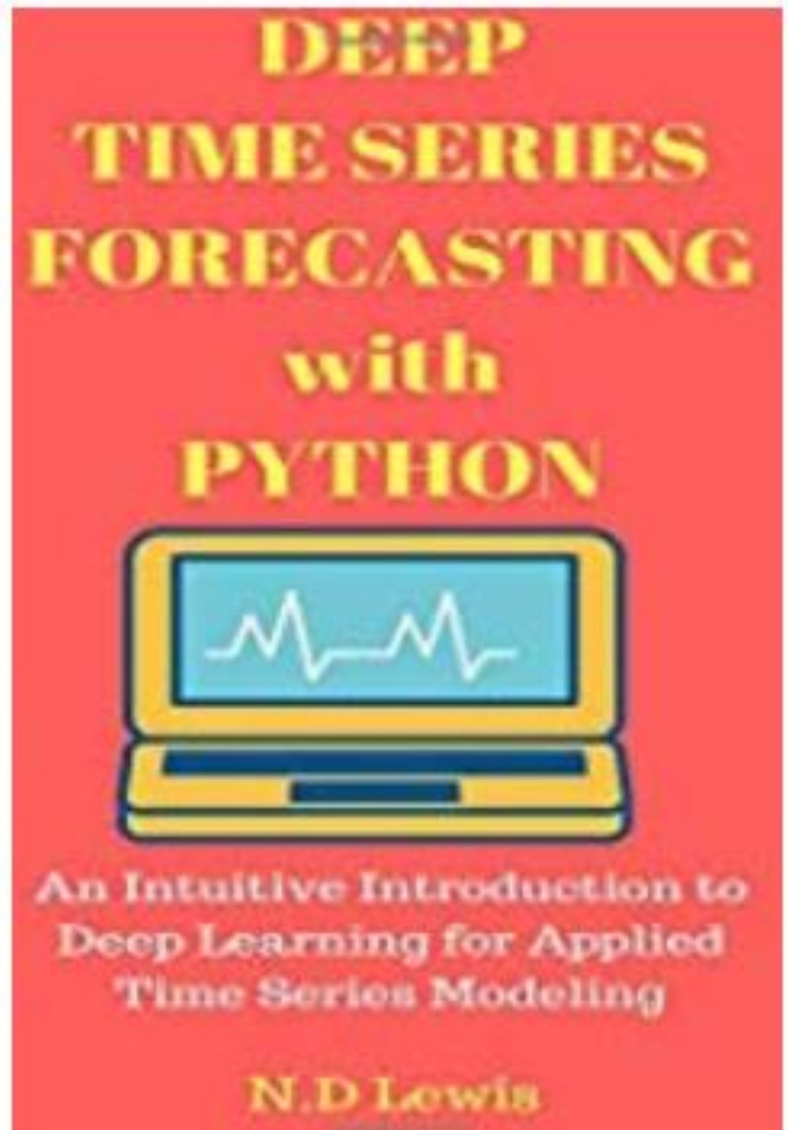
sentdex - YouTube
youtube.com

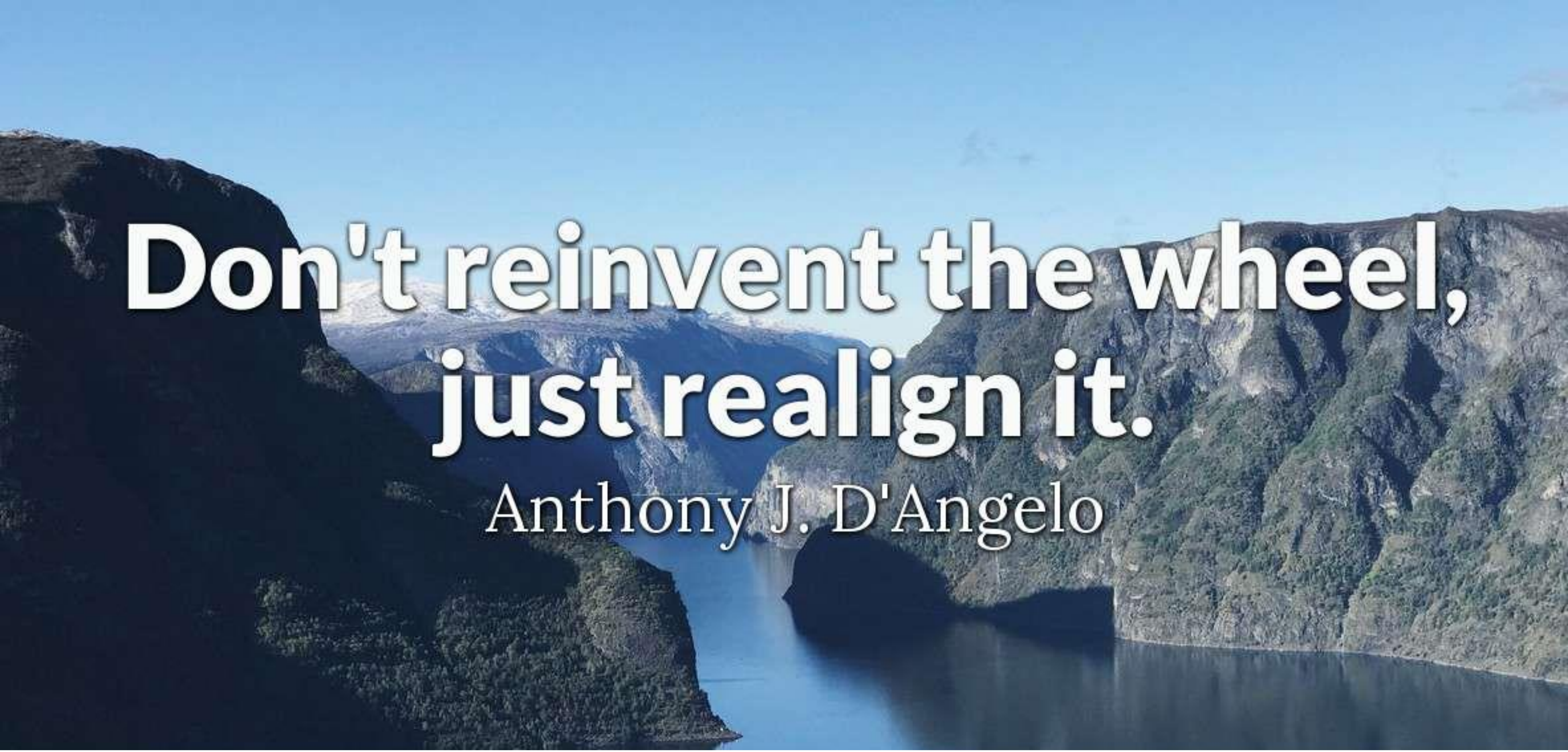
[Project-Link: Github]()

References


Long Short-Term Memory Networks With Python
Develop Sequence Prediction Models With Deep Learning
Jason Brownlee
MACHINE LEARNING MASTERY


Data Science with Python and Dask
Jesse C. Daniel
MEAP
MANNING


DEEP TIME SERIES FORECASTING with PYTHON
An Intuitive Introduction to Deep Learning for Applied Time Series Modeling
N.D Lewis



Thank you