New York City (NYC) Yellow Taxi Trip 2020

Introduction

The project analyzes the impact of Covid-19 pandemic on yellow taxi users and taxicab owners.

Background

The transportation industry was greatly affected by the Covid-19 pandemic, and thus people's travel/transit behavior severely changed. This project aims at observing the change in trend related to NYC's Yellow Taxi's travels and forecast future trends.

Problem

The problem could be defined as:

- How has covid-19 pandemic affected NYC Yellow Taxi's?
- When or how long will it take for things to return to normal?

Target Audience

Medallion taxicab owners, and drivers whose businesses have been affected by the pandemic, and Yellow cab users/riders.

Data

The 2020 Yellow Taxi Trip data was used for this project. The trip data was collected from NYC Open data (2020 Yellow Taxi Trip Data | NYC Open Data (cityofnewyork.us)). The dataset contained 24.6 Million rows and 18 columns.

Table 1: Column Description (NYC Open data)

Column Name	Column Description
VendorID	A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	Number of passengers.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged
RateCodeID	The final rate code in effect at the end of the trip. 1= Standard rate, 2=JFK, 3=Newark, 4=Nassau or Westchester, 5=Negotiated fare, 6=Group ride

Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip
Payment_type	A numeric code signifying how the passenger paid for the trip. 1= Credit card 2= Cash 3= No charge 4= Dispute 5= Unknown 6= Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip
Total_amount	The total amount charged to passengers. Does not include cash tips

Location data

TLC taxi zone location data including location IDs, location names and corresponding boroughs for each location ID was retrieved from (https://s3.amazonaws.com/nyc-tlc/misc/taxi_zones.zip).

Methodology

Since it's a fairly large data, PySpark (a python interface for Apache Spark) was used for both data mining and modeling. The data analysis was carried out using DataBricks Cloud System which was connected to Amazon S3 where the data was stored. The timeseries forecasting model was created using Facebook Prophet.

Data Analysis and Visualization

In analyzing the dataset, the following tasks were performed.

- **Importing the required libraries for the analysis of the data.**

```
#Importing the necessary libraries

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DateType
from pyspark.sql.types import *
from pyspark.sql.functions import col
from pyspark.sql.functions import to_date
from pyspark.sql.functions import lit
from pyspark.sql.functions import sum, avg, max, min, mean, count
from pyspark.sql.functions import *
from datetime import datetime, date
import pandas as pd
from pyspark.sql.functions import to_timestamp
from pyspark.sql.functions import unix_timestamp, from_unixtime
from pyspark.sql.types import TimestampType
from pyspark.sql import functions as F
```

- **Specifying the file path and reading the files.**

```
# File paths

path='s3a://just-abdul-aws/NTA/Yellow_Taxi_Trip_Data_2020.csv'
path_location='s3a://just-abdul-aws/NTA/taxi_zones.csv'
```

Cmd 3

```
df = spark.read.csv(path, header=True, inferSchema=True)
df_location = spark.read.csv(path_location, header=True, inferSchema=True)
```

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [VendorID: integer, tpep_pickup_datetime: string ... 16 more fields]

df_location: pyspark.sql.dataframe.DataFrame = [X: double, Y: double ... 6 more fields]

```
df.limit(5).display()
df_location.limit(5).display()
```

(2) Spark Jobs

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocatio
1	1	01/01/2020 12:28:15 AM	01/01/2020 12:33:03 AM	1	1.2	1	N	238
2	1	01/01/2020 12:35:39 AM	01/01/2020 12:43:04 AM	1	1.2	1	N	239
3	1	01/01/2020 12:47:41 AM	01/01/2020 12:53:52 AM	1	0.6	1	N	238
4	1	01/01/2020 12:55:23 AM	01/01/2020 01:00:14 AM	1	0.8	1	N	238
5	2	01/01/2020 12:01:58 AM	01/01/2020 12:04:16 AM	1	0	1	N	193

Showing all 5 rows.

	X	Y	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough
1	-74.176785745214	40.689515648043	1	0.116357453189	0.0007823067885	Newark Airport	1	EWR
2	-73.826125770320	40.625724237751	2	0.43346966679	0.00486634037837	Jamaica Bay	2	Queens
3	-73.849478923859	40.865887541977	3	0.084341105901	0.00031441415682	Allerton/Pelham Gardens	3	Bronx
4	-73.977022921933	40.724152143671	4	0.043566527092	0.00011187194619	Alphabet City	4	Manhattan
5	-74.189929671237	40.550340123832	5	0.092146489857	0.00049795748936	Arden Heights	5	Staten Island

Showing all 5 rows.

- **Preprocessing and Cleaning the data.**

In [0]:
```
# To check if the Location ID's for the Trip data (Pick Up, Drop Off) and the Location data matches

loc_count = df_location.select('LocationID').distinct().count()
PU_count = df.select('PULocationID').distinct().count()
DO_count = df.select('DOLocationID').distinct().count()
```

In [0]:
```
print("The location data has: " + str(loc_count) + " distinct Location ID's")
print("Pick Up Location has: " + str(PU_count) + " distinct Location ID's")
print("Drop Off location data has: " + str(DO_count) + " distinct Location ID's")
```

The location data has: 260 distinct Location ID's Pick Up Location has: 262 distinct Location ID's Drop Off location data has: 263 distinct Location ID's

In [0]:
```
%%time

location = df_location.select("LocationID","zone","borough")
PU_location = location.withColumnRenamed("LocationID","PULocationID")
DO_location = location.withColumnRenamed("LocationID","DOLocationID")

PU_data = df.join(PU_location, "PULocationID", 'left')
PU_data = PU_data.withColumnRenamed("zone","PU_Zone")
PU_data = PU_data.withColumnRenamed("borough","PU_Borough")

data = PU_data.join(DO_location, "DOLocationID", 'left')
data = data.withColumnRenamed("zone","DO_Zone")
data = data.withColumnRenamed("borough","DO_Borough")
```

CPU times: user 2.79 ms, sys: 2.16 ms, total: 4.94 ms Wall time: 52.4 ms

In [0]:
```
data = data.withColumnRenamed("trip_distance","trip_distance(miles)")
```

In [0]:
```
# Peak-up timestamp

data = data.withColumn("PU_timestamp", from_unixtime(unix_timestamp("tpep_pickup_datetime", 'MM/dd/yyyy hh:mm:ss a')).cast(Ti
```

In [0]:
```
# Drop-Off timestamp

data = data.withColumn("DO_timestamp", from_unixtime(unix_timestamp("tpep_dropoff_datetime", 'MM/dd/yyyy hh:mm:ss a')).cast(T
```

In [0]:
```
data = data.withColumn("Year", year(col("DO_timestamp")))
```

- **The data Schema**

```
data.printSchema()
```

```
|-- DOLocationID: integer (nullable = true)
|-- PULocationID: integer (nullable = true)
|-- VendorID: integer (nullable = true)
|-- tpep_pickup_datetime: string (nullable = true)
|-- tpep_dropoff_datetime: string (nullable = true)
|-- passenger_count: integer (nullable = true)
|-- trip_distance(miles): string (nullable = true)
|-- RatecodeID: integer (nullable = true)
|-- store_and_fwd_flag: string (nullable = true)
|-- payment_type: integer (nullable = true)
|-- fare_amount: string (nullable = true)
|-- extra: string (nullable = true)
|-- mta_tax: string (nullable = true)
|-- tip_amount: string (nullable = true)
|-- tolls_amount: double (nullable = true)
|-- improvement_surcharge: double (nullable = true)
|-- total_amount: string (nullable = true)
|-- congestion_surcharge: double (nullable = true)
|-- PU_Zone: string (nullable = true)
|-- PU_Borough: string (nullable = true)
|-- DO_Zone: string (nullable = true)
```

- **Exploratory Data Analysis**.
  Using visualization to understand the dataset and generating meaningful relationships that exist in the dataset.

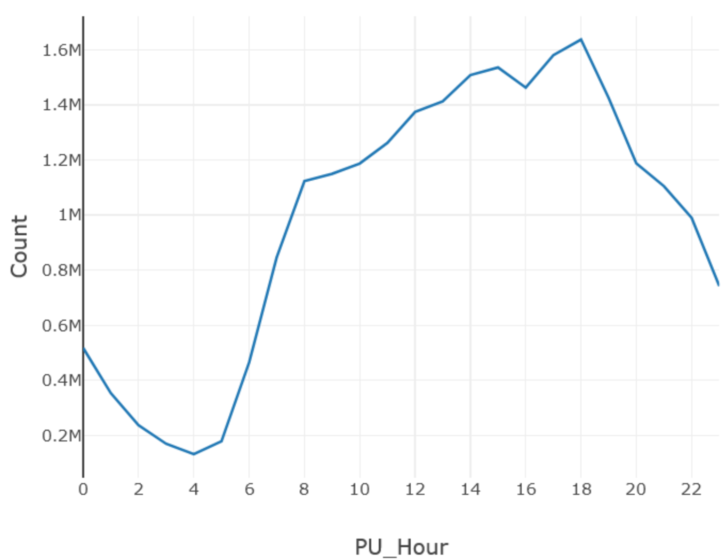


Figure 1: Peak hours (Between 5pm and 6pm)

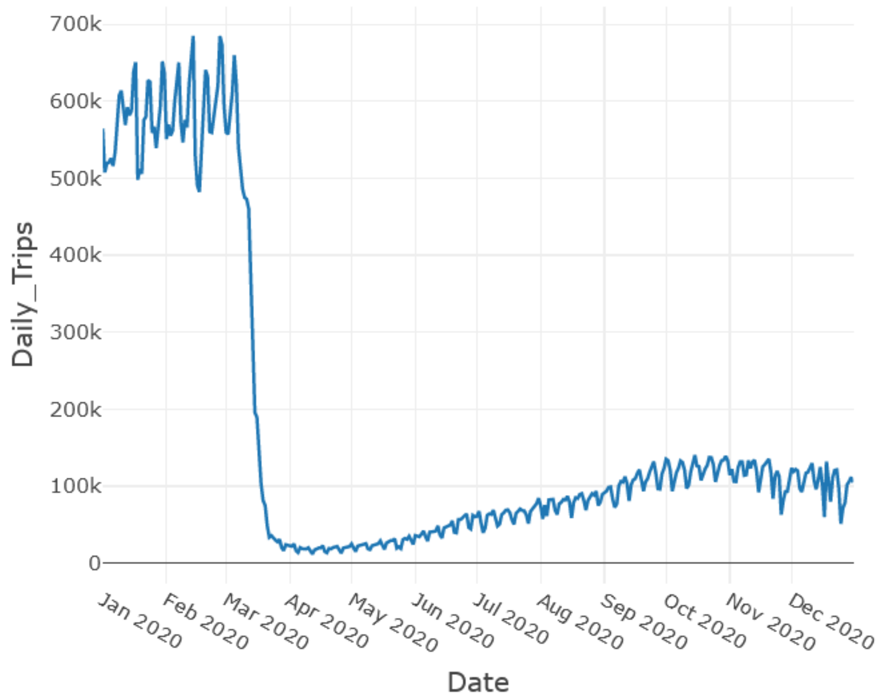


Figure 2: Drop in total daily trip from March 2020

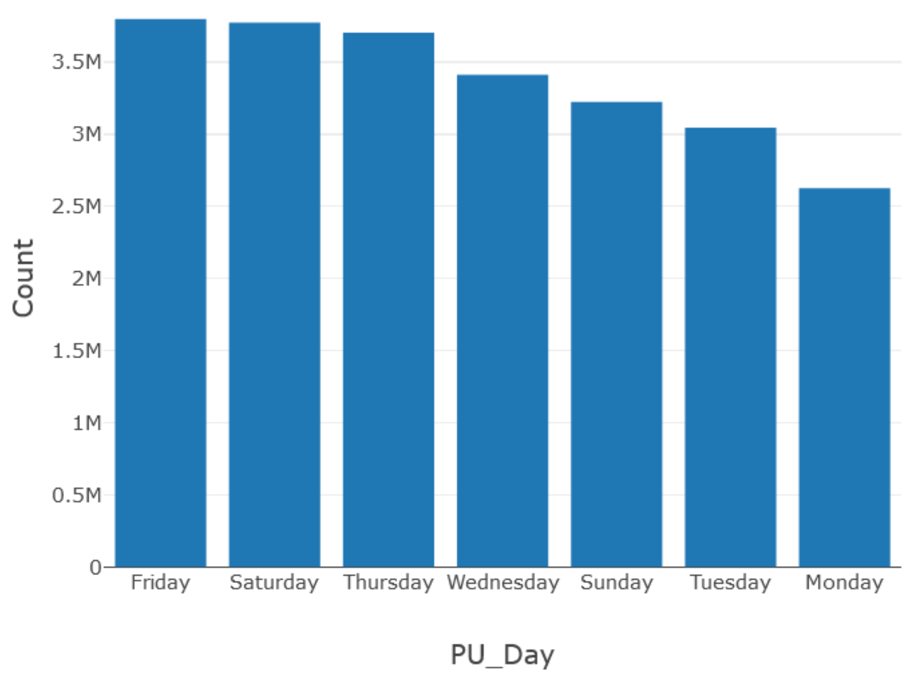


Figure 3: Peak days (Friday and Saturday)

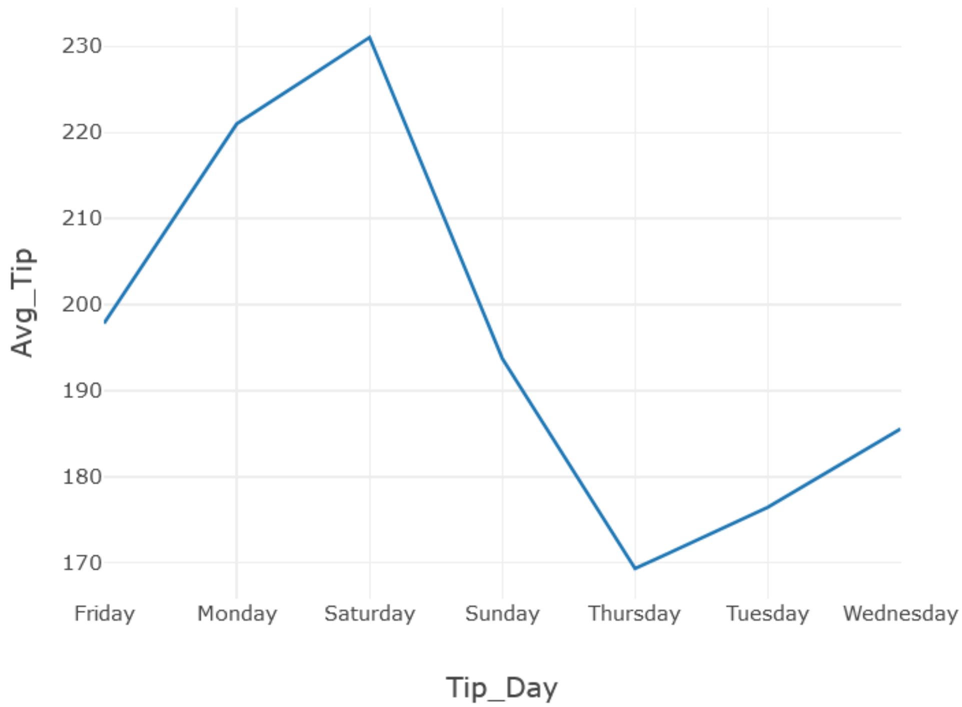


Figure 4: Top tips are given on Mondays and Saturdays

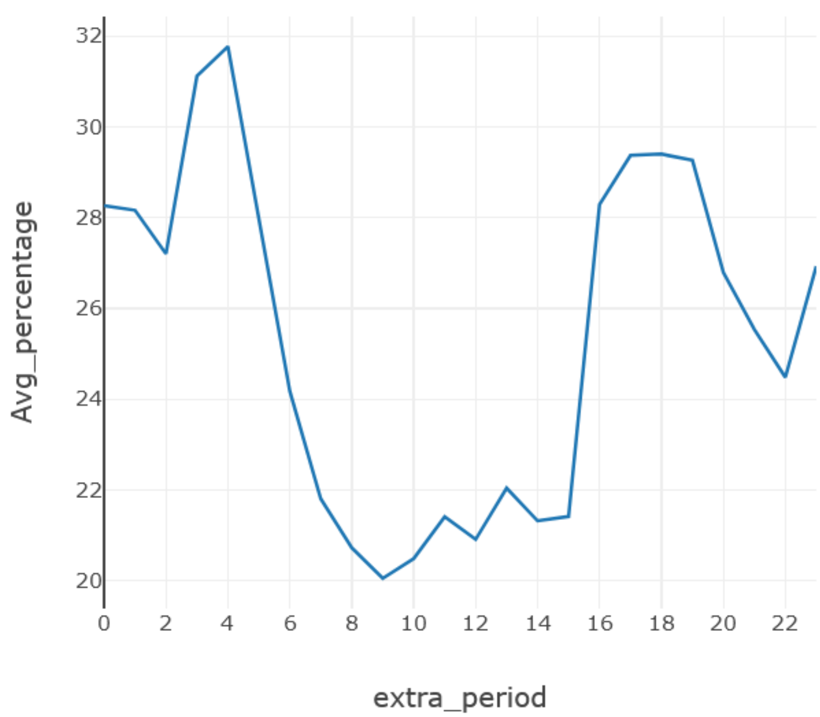


Figure 5: Riders pay extra 29% of their fare cost around 5pm – 7pm

- **Training the model**. Before training the model, the cleaned data was resampled, changing the frequency of the timestamp to daily intervals. This way, the data was reduced from around 24million rows, to 366 rows.

```
TS_ = data.groupBy(window("DO_timestamp", "1 day")).agg(sum("trip_distance(miles)").alias("Daily_Trips"))
TS_data = TS_.select(TS_.window.start.cast("timestamp").alias("Date"), "Daily_Trips").collect()
TS_data = spark.createDataFrame(TS_data)
TS_data = TS_data.sort(TS_data.Date.asc())
TS_data.display()
```

(3) Spark Jobs

TS_: pyspark.sql.dataframe.DataFrame = [window: struct, Daily_Trips: double]

TS_data: pyspark.sql.dataframe.DataFrame = [Date: timestamp, Daily_Trips: double]

	Date	Daily_Trips
1	2020-01-01T00:00:00.000+0000	564678.0799999975
2	2020-01-02T00:00:00.000+0000	507599.3600000032
3	2020-01-03T00:00:00.000+0000	519248.8700000045
4	2020-01-04T00:00:00.000+0000	520527.4100000029
5	2020-01-05T00:00:00.000+0000	525933.1599999996
6	2020-01-06T00:00:00.000+0000	516049.2499999970
7	2020-01-07T00:00:00.000+0000	533555.1600000025

Showing all 366 rows.

- **The time-series forecasting model was built using Facebook Prophet.**

```
# instantiate the model and set parameters
model = Prophet(
    interval_width=0.95,
    growth='linear',
    daily_seasonality=True,
    weekly_seasonality=True,
    yearly_seasonality=False,
    seasonality_mode='additive')

# fit the model to historical data
model.fit(TS_DF)
```

```
Out[75]: <fbprophet.forecaster.Prophet at 0x7f86f89e1fa0>
```

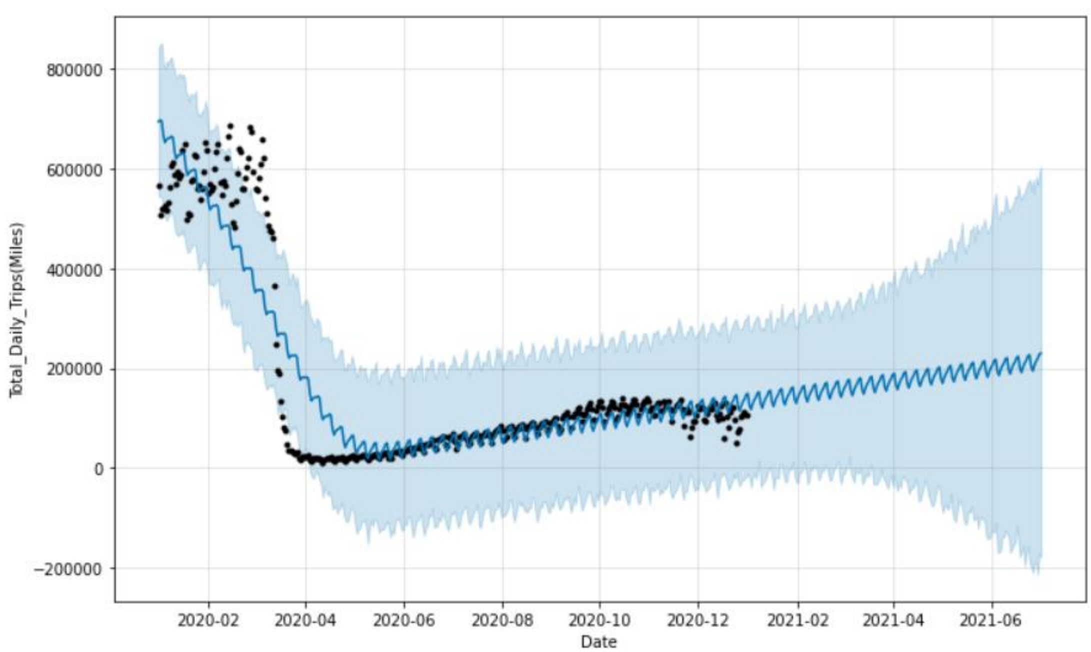
```
future_pd = model.make_future_dataframe(
    periods=183,
    freq='d',
    include_history=True
)

# predict over the dataset
forecast_pd = model.predict(future_pd)
```

- **Model Forecast.**

```
forecast_result.tail(10)
```

Out[85]:

	TimeStamp	forecast	Lower_bound	Upper_bound
539	2021-06-23	215486.944396	-171153.848105	565187.297295
540	2021-06-24	224050.166001	-157202.715873	572361.249971
541	2021-06-25	226660.842339	-191687.189124	578905.697354
542	2021-06-26	203196.289593	-198231.552438	555902.002130
543	2021-06-27	194381.199480	-210392.620794	577380.581907
544	2021-06-28	205261.835796	-189324.816720	551611.033289
545	2021-06-29	213372.179540	-191519.469139	583776.421670
546	2021-06-30	218585.932633	-214090.579308	568764.230938
547	2021-07-01	227149.154238	-171824.290064	591417.733581
548	2021-07-02	229759.830576	-179827.133370	603672.803350



Result and discussion

From the analysis the following deductions were made;

- Top pick up locations are: Upper East Side North and Upper East Side South, all in Manhattan
- Least pick up locations are: West Brighton and Rikers Island
- Peak periods: peak hours (5pm - 6pm), peak days (Fridays and Saturdays) and peak months (January and February).
- Riders pay an extra 25% of their actual fare cost for every ride taken. This value rises to around 30% in the evenings between 4pm and 7pm (Might be due to traffic congestion).
- Highest tips are given by riders picked up in rich neighborhood's like, Greenwich Village, Turtle Bay, Little Italy. These high tips are usually paid on Saturdays and Mondays between (1pm and 2pm) and around (5pm and 10pm).
- Due to the Covid-19 pandemic, total daily trip dropped by around 95% from 600,000 miles to 20,000 miles from March until July. The total daily trip has gradually increased since July 2020.
- From the model forecast, it’s most likely going to take far more than 6months (starting from Jan 1, 2021) for the total daily trip to return to normal.

Conclusion

From the analysis we saw that Covid-19 pandemic heavily affected the yellow taxicabs and it’s most likely going to take far more than 6months (starting from Jan 1, 2021) for things to return to normal.

Recommendation

I would recommend adding more trip data to the analysis at least adding 4 years (2016, 2017, 2018, 2019), and re-training the model using the newly added data, then forecasting the daily trips for a whole year.

I would also recommend trying other algorithms like Amazon’s Deep AR.