

Big Data Analysis of NYC TLC Trip Record Data

Frances Lin

March 2021

Background and Objective

My answer to the question “Where would you like to visit next?” is always Iceland. This is true even prior to COVID-19. While we are all stuck at home 24/7, I thought it might be nice to at least send my mind off to somewhere magical. However, after long hours search with minimal luck, I “settled” for this still quite interesting and fairly well-known, large (>100 GB) open dataset: NYC TLC trip record data.

The objective of this project is to answer the following questions of interest, which include

1. How does monthly average fare amount vary by year (2019 vs 2020) and by taxi type (yellow taxi vs green taxi)?
2. How does yearly total count of high volume fhv (for-hire vehicle) services vary by year (2019 vs 2020) and by license type (Uber, Lyft, Via, Juno)?
3. What are the top 6 drop-off locations and how do they vary by year (2019 vs 2020) and by vehicle type (yellow taxi, green taxi, fhv, hfhv)?

1. Obtain

Overview of Dataset

TLC trip record data is obtained from the New York City Taxi & Limousine Commission’s site: [link](#). Overall, the data is split across multiple **csv** files. It is composed of more than one table of related data. The data is accessed through creating multiple **txt** files that contain list of **urls** and using the **wget** command in VM. The entire dataset is 128.7 GB in total. (All the files were uploaded to **Google Storage** but only a subset of it were used, which is 35.2662 GB in total.)

Data Description

The yellow and green taxi and for-hire vehicle (fhv) trip records contain records from Jan 2019 to Dec 2020. The high volume for-hire vehicle (hfhv) trip records contain records from Feb 2019 to Dec 2020. (*Note that on August 14, 2018, a Local Law of 2018 was signed to create a new license category for TLC-licensed businesses that currently or plan to dispatch more than 10,000 for-hire vehicle (fhv) trips in NYC per day. These businesses include Juno, Uber, Via, and Lyft.*)

Each record contains trips within a month. Individual row in each record represents individual trip.

The taxi+ zone lookup record contains location information such as **Borough** (e.g. Brooklyn, Manhattan) and **Zone** (e.g. Crown Heights North, Upper East Side North). Corresponding maps in **jpgs** can be found on the same site that contains these trip records.

Fields Description

The yellow and green taxi trip records contain fields such as **VendorID** (1=Creative Mobile Technologies, LLC or 2=VeriFone Inc.), **tpep_pickup_datetime**, **tpep_dropoff_datetime**, **PULocationID**, **DOLocationID**, **Trip_distance**, **Payment_type**, **Fare_amount**, **Tip_amount**, **Total_amount**, etc. The green taxi trip records has additional field **Trip_type** that is used to indicate whether it is 1=Street-hail or 2=Dispatch.

The fhv & hfhv trip records contain fields such as Dispatching_base_num, Pickup_datetime, DropOff_datetime, PULocationID, DOLocationID, and SR_Flag (1=shared ride or null). The hfhv trip records has additional field Hvfhs_license_num that is used to identify businesses: Juno, Uber, Via, and Lyft.

The taxi zone lookup record contains fields such as LocationID, Borough, Zone, and service_zone. LocationID can be used to identify PULocationID (i.e. pick-up location) and DOLocationID (i.e. drop-off location) in all other records.

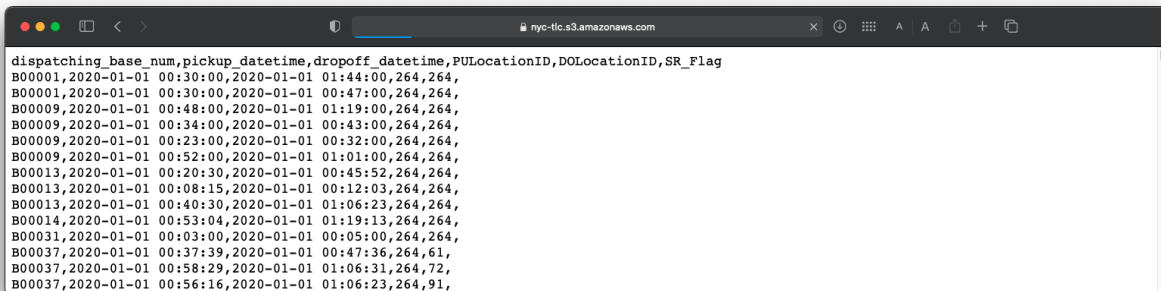
Sample of Initial Data

a sample of *Yellow Taxi Trip Records 2020-01*:

a sample of *Green Taxi Trip Records 2020-01*:

VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	RatecodeID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount	tolls_amount	ehail_fee	improvement_surcharge	total_amount	payment_type	trip_type	congestion_surcharge				
2	2019-12-18 15:52:30	2019-12-18 15:54:39	N	1	264	264	5	.00	3	5	0.5	0.5	0	0.1	0	0.3	4.81	1	1	0			
2	2020-01-01 00:45:58	2020-01-01 00:56:39	N	5	66	65	2	1.28	20	0	0	4	06	0	0	3	24	36	1	2	0		
2	2020-01-01 00:41:38	2020-01-01 00:52:49	N	1	181	228	1	2.47	10	5	0.5	0.5	3	54	0	0	3	15	34	1	1	0	
1	2020-01-01 00:52:46	2020-01-01 01:14:21	N	1	129	263	2	6.30	21	3	25	0	5	0	0	0	3	25	05	2	1	2	75
1	2020-01-01 00:19:57	2020-01-01 00:30:56	N	1	210	150	1	2.30	10	0	5	0.5	0.5	0	0	0	3	11	3	1	1	0	
1	2020-01-01 00:52:33	2020-01-01 01:09:54	N	1	35	39	1	3.00	13	5	0.5	0.5	0	0	0	0	3	14	8	1	1	0	
2	2020-01-01 00:10:18	2020-01-01 00:22:16	N	1	25	61	1	2.77	11	0	5	0.5	0.5	0	0	0	3	12	3	2	1	0	
2	2020-01-01 01:03:14	2020-01-01 01:29:45	N	1	225	89	1	4.98	20	5	0.5	0.5	0	0	0	0	3	21	8	2	1	0	
2	2020-01-01 00:04:11	2020-01-01 00:09:48	N	1	129	129	1	.71	5	0.5	0.5	0.5	0	0	0	0	3	6	8	2	1	0	
2	2020-01-01 00:25:52	2020-01-01 00:32:16	N	1	129	83	1	.80	5	0.5	0.5	0.5	0	0	0	0	3	6	8	2	1	0	
2	2020-01-01 00:47:32	2020-01-01 00:59:25	N	1	82	173	1	1.52	9	5	0.5	0.5	0	0	0	0	3	10	8	2	1	0	
1	2020-01-01 00:26:40	2020-01-01 00:40:42	N	1	74	69	1	3.80	14	0	5	0.5	0.5	0	0	0	3	15	3	2	1	0	
2	2020-01-01 00:38:47	2020-01-01 00:46:02	N	1	74	41	1	1.12	6	5	0.5	0.5	0	0	0	0	3	7	8	1	1	0	

a sample of *For-Hire Vehicle Trip Records 2020-01*:



dispatching_base_num	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	SR_Flag
B00001	2020-01-01 00:30:00	2020-01-01 01:44:00	264	264	
B00001	2020-01-01 00:30:00	2020-01-01 00:47:00	264	264	
B00009	2020-01-01 00:48:00	2020-01-01 01:19:00	264	264	
B00009	2020-01-01 00:34:00	2020-01-01 00:43:00	264	264	
B00009	2020-01-01 00:23:00	2020-01-01 00:32:00	264	264	
B00009	2020-01-01 00:52:00	2020-01-01 01:01:00	264	264	
B00013	2020-01-01 00:20:30	2020-01-01 00:45:52	264	264	
B00013	2020-01-01 00:08:15	2020-01-01 00:12:03	264	264	
B00013	2020-01-01 00:40:30	2020-01-01 01:06:23	264	264	
B00014	2020-01-01 00:53:04	2020-01-01 01:19:13	264	264	
B00031	2020-01-01 00:03:00	2020-01-01 00:05:00	264	264	
B00037	2020-01-01 00:37:39	2020-01-01 00:47:36	264	61	
B00037	2020-01-01 00:58:29	2020-01-01 01:06:31	264	72	
B00037	2020-01-01 00:56:16	2020-01-01 01:06:23	264	91	

a sample of *High Volume For-Hire Vehicle Trip Records 2020-01*:

a sample of *Taxi Zone Lookup Record*:

LocationID	Borough	Zone	service_zone
1	EWR	Newark Airport	EWR
2	Queens	Jamaica Bay	Boro Zone
3	Bronx	Allerton/Pelham Gardens	Boro Zone
4	Manhattan	Alphabet City	Yellow Zone
5	Staten Island	Arden Heights	Boro Zone
6	Staten Island	Arrochar/Fort Wadsworth	Boro Zone
7	Queens	Astoria	Boro Zone
8	Queens	Astoria Park	Boro Zone

Initially, I attempted to make **API** request since the data is host on **Amazon Web Services** and can be copied to **S3** bucket. To save time for processing in **Google BigQuery** and **Apache Spark**, I directly downloaded the **csv** files from the site to an external hard drive and compressed them into multiple **zip** files instead. The total of the 4 uncompressed file is approx. 132.68 GB, which is compressed to approx. 26.3 GB.

In retrospect, it makes more sense to do as follows: To load data to **BigQuery**, I revisited Week 6's Big Data Module. First, in **Compute Engine**, I created a VM instance with 200 GB and logged in remotely via SSH. Next, I tried to connect SSH to the external hard drive where I saved the 4 **zip** files but did not succeed. Instead, I created a **txt** that contains a list of **urls** and used the **wget** commend to load the data from the site to a bucket in **Storage**.

Inside the bucket, I repeated the process and created separate folders so that trip records of **yellow taxi** (**yellow_data**: 72 objects, 63.8 GB), **green taxi** (**green_data**: 72 files, 35.4 GB), **fhv** (**fhv_data**: 72 files, 35.4 GB) and **hfhv** (**hfhv_data**: 23 files, 22.3 GB) were stored separately.

I had trouble previewing the entire dataset in **Dataprep** (I thought they did not load entirely,) so I repeated the above process. But this time, instead of separating records by type, I separated records further by year. The combined dataset (35.2662 GB) breaks down to:

record of

- yellow_data_2020: 12 files, 2.1 GB, yellow_data_2019: 12 files, 7.3 GB
- green_data_2020: 12 files, 150 MB, green_data_2019: 12 files, 528 MB
- fhv_data_2020: 12 files, 688.2 MB, fhv_data_2019: 12 files, 2.2 GB
- hfhv_data_2020: 12 files, 8.4 GB, hfhv_data_2019: 12 files, 13.9 GB

snippet of Linux command to load data to GCS (Google Cloud Storage)

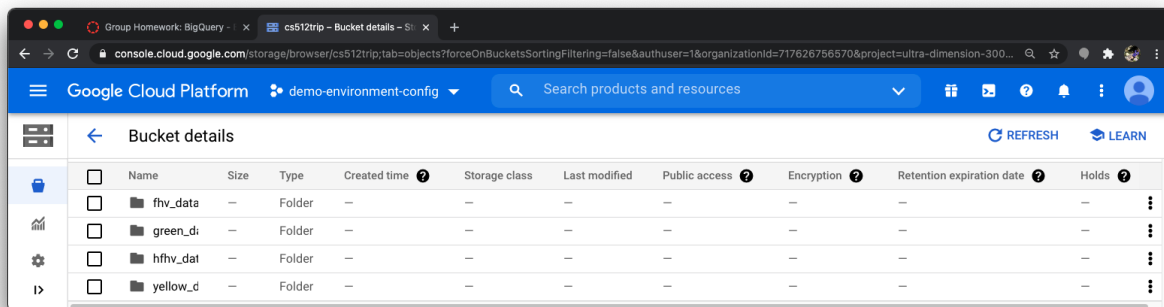
```
# Create a directory and set it to current directory
# Reference: https://canvas.oregonstate.edu/courses/1799395/assignments/8130281?module_item_id=20176237
mkdir yellow_data_2020
cd yellow_data_2020

# Create a txt file that contains a list of urls
cat > url_yellow_2020.txt

# Download multiple urls
# Reference: https://shapeshed.com/unix-wget/#how-to-download-multiple-urls
wget -i url_yellow_2020.txt

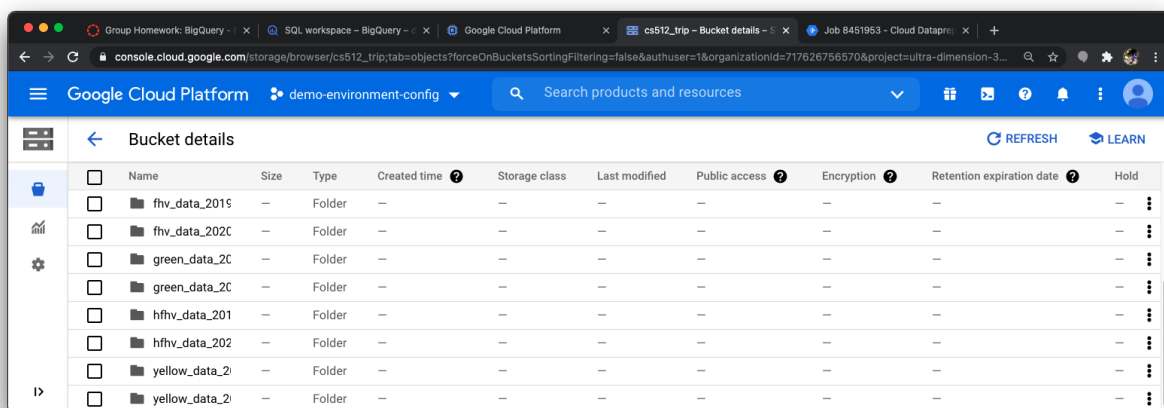
# Copy to bucket > yellow_data: 72 objects, 63.8 GB
mv url_yellow_2020.txt ../
cd ..
gsutil -m cp -r yellow_data_2020/ gs://cs512_trip
```

bucket details in Storage (~ 128.7 GB):



Name	Size	Type	Created time	Storage class	Last modified	Public access	Encryption	Retention expiration date	Holds
fhv_data	0	Folder	—	—	—	—	—	—	—
green_data	0	Folder	—	—	—	—	—	—	—
hfhv_data	0	Folder	—	—	—	—	—	—	—
yellow_data	0	Folder	—	—	—	—	—	—	—

bucket details in Storage - Updated (~ 35.2662 GB):



Name	Size	Type	Created time	Storage class	Last modified	Public access	Encryption	Retention expiration date	Hold
fhv_data_2019	0	Folder	—	—	—	—	—	—	—
fhv_data_2020	0	Folder	—	—	—	—	—	—	—
green_data_20	0	Folder	—	—	—	—	—	—	—
green_data_20	0	Folder	—	—	—	—	—	—	—
hfhv_data_201	0	Folder	—	—	—	—	—	—	—
hfhv_data_202	0	Folder	—	—	—	—	—	—	—
yellow_data_2	0	Folder	—	—	—	—	—	—	—
yellow_data_2	0	Folder	—	—	—	—	—	—	—

2. Scrub

Overview of Data Wrangling

I realize now that data wrangling can be an exceedingly long and still ongoing process. Overall, data were wrangled in **Dataprep**, **BigQuery**, and **R**, and the process continues in **Spark**.

To BigQuery

Once a dataset is created in **BigQuery**, using **Dataprep**, I attempted to transfer the data from **GCS** to **BigQuery**. The job for yellow taxi alone took more than 2 hours and still did not complete. Since it seemed that not all data were loaded to **Dataprep** successfully (they were!), as mentioned, I repeated all of the above process: I separated trip records not only by type (i.e. yellow, green taxi, fhv, fhfv) but also by year (i.e. 2015, 2016, and on) and reloaded them to **Storage**.

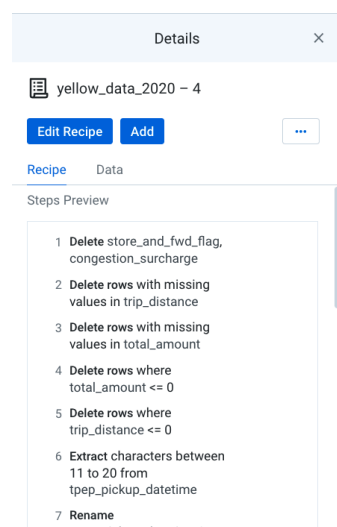
It makes more sense to proceed with this route (by type & by years) as the final route, although it seems like the previous route (by type only) and then querying by **TIMESTAMP** is doable too. *(2015's dataset, for example, contains features such as longitude and latitude that were represented as Taxi Zone in 2019's and 2020's dataset, which further confirms that it may be best to also separate the records by year.)*

Data wrangling in Dataprep

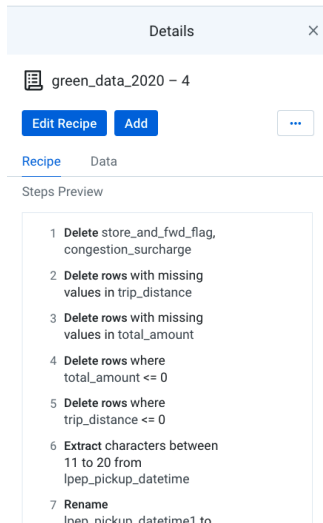
In **Dataprep**, I created a recipe for yellow trip records (and an equivalent recipe was copied and changed input to green trip records.) Basic steps are as followed:

1. Delete columns not of interest
2. Delete rows with missing values in `trip_distance` & `total_amount`
3. Delete rows where values ≤ 0 for both `trip_distance` & `total_amount`
4. Extract time from both `tpep_pickup_datetime` & `tpep_dropoff_datetime` (Could be done later in **Datapro** too.)
5. Delete columns not of interest

recipe of *Yellow Taxi Trip Records 2020-01 to 2020-12*:



recipe of *Green Taxi Trip Records 2020-01 to 2020-12*:



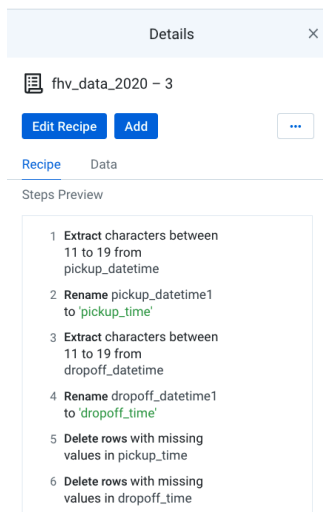
I created a recipe for fhv trip records (and an equivalent recipe was copied and changed input to hfhv trip records.) Basic steps are as followed:

1. Extract time from both `pickup_datetime` & `dropoff_datetime` (Again, could be done later in **Dataprocc** too.)
2. Delete rows with missing values in `pickup_time` & `dropoff_time`
3. Delete rows where values ≤ 0 for both `trip_distance` & `total_amount`

For hfhv trip records only:

4. Add a column named `license` indicating whether it is ran by Juno, Uber, Via, or Lyft

recipe of *For-Hire Vehicle Trip Records 2020-01 to 2020-12*:



recipe of *High Volume For-Hire Vehicle Trip Records 2020-01 to 2020-12*:

Details

×

hfhv_data_2020 – 2

Edit Recipe

Add

...

Recipe

Data

Steps Preview

- 1 Extract characters between 11 to 19 from pickup_datetime
- 2 Rename pickup_datetime1 to 'pickup_time'
- 3 Extract characters between 11 to 19 from dropoff_datetime
- 4 Rename dropoff_datetime1 to 'dropoff_time'
- 5 Delete rows with missing values in pickup_time
- 6 Delete rows with missing values in dropoff_time
- 7 Extract characters between

Preview screens in BigQuery

preview of *Yellow & Green Taxi Trip Records 2020-01 to 2020-12*:

yellow_data_2020_20210309_002840

COPY TABLE

DELETE TABLE

EXPORT

Schema

Details

Preview

Row	tpep_pickup_datetime	tpep_pickup_time	tpep_dropoff_datetime	tpep_dropoff_time	trip_distance	PULocationID	DOLocationID	payment_t
1	2020-11-13T10:29:43	10:29:43	2020-11-13T10:40:44	10:40:44	1.44	78	136	
2	2020-11-19T09:27:00	09:27:00	2020-11-19T09:44:00	09:44:00	4.53	213	242	
3	2020-11-18T05:53:44	05:53:44	2020-11-18T06:20:31	06:20:31	18.17	216	14	
4	2020-11-20T12:45:00	12:45:00	2020-11-20T13:58:00	13:58:00	4.17	188	89	

Rows per page:

100

1 - 100 of 24231126

First page

>| Last page

green_data_2020

COPY TABLE

DELETE TABLE

EXPORT

Schema

Details

Preview

Row	lpep_pickup_datetime	lpep_pickup_time	lpep_dropoff_datetime	lpep_dropoff_time	PULocationID	DOLocationID	trip_distance	total_amou
1	2020-12-02T08:19:46	08:19:46	2020-12-02T09:28:06	09:28:06	133	265	27.76	92.
2	2020-12-03T08:29:42	08:29:42	2020-12-03T09:33:08	09:33:08	94	227	22.46	68.
3	2020-12-03T10:26:26	10:26:26	2020-12-03T10:30:50	10:30:50	37	37	0.3	!
4	2020-12-03T15:08:34	15:08:34	2020-12-03T15:30:28	15:30:28	56	255	6.55	26.

Rows per page:

100

1 - 100 of 1662536

First page

>| Last page

preview of *For-Hire Vehicle Trip Records 2020-01 to 2020-12*:

fhv_data_2020								
<div> COPY TABLE DELETE TABLE EXPORT ▼ </div>								
<div> Schema Details Preview </div>								
Row	dispatching_base_num	pickup_datetime	pickup_time	dropoff_datetime	dropoff_time	PULocationID	DOLocationID	SR_Flag
1	B02715	2020-01-18T05:58:34	05:58:34	2020-01-18T06:15:04	06:15:04	162	138	
2	B03047	2020-01-18T05:24:13	05:24:13	2020-01-18T05:34:21	05:34:21	182	185	
3	B01282	2020-01-18T06:46:11	06:46:11	2020-01-18T06:54:02	06:54:02	264	39	
4	B01285	2020-01-18T06:20:35	06:20:35	2020-01-18T06:34:16	06:34:16	264	250	
<div> Rows per page: 100 ▼ 1 - 100 of 13545130 First page < < > > Last page </div>								

preview of *High Volume For-Hire Vehicle Trip Records 2020-01 to 2020-12*:

hfhv_data_2020								
<div> COPY TABLE DELETE TABLE EXPORT ▼ </div>								
<div> Schema Details Preview </div>								
Row	hvfhs_license_num	license	dispatching_base_num	pickup_datetime	pickup_time	dropoff_datetime	dropoff_time	PULocationID
1	HV0003	Uber	B02395	2020-02-20T11:27:10	11:27:10	2020-02-20T11:38:31	11:38:31	25
2	HV0003	Uber	B02395	2020-02-20T11:49:20	11:49:20	2020-02-20T12:24:27	12:24:27	25
3	HV0003	Uber	B02395	2020-02-20T12:25:41	12:25:41	2020-02-20T12:47:19	12:47:19	25
4	HV0003	Uber	B02395	2020-02-20T12:06:24	12:06:24	2020-02-20T12:14:25	12:14:25	25
<div> Rows per page: 100 ▼ 1 - 100 of 143234711 First page < < > > Last page </div>								

To PySpark

To load data to **Spark**, I revisited Week 7's Spark Module and followed part of the starter code provided for the Plane Distance assignment. Once the **py** file is created, I uploaded this file to **Storage**. Then, I created a cluster and submitted a job in **Dataproc**.

test snippet of Python script to run Dataproc job

```
# Connect to Spark
sc = pyspark.SparkContext() # run Spark applications
#PACKAGE_EXTENSIONS= ('gs://hadoop-lib/bigquery/bigquery-connector-hadoop2-latest.jar')

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output'.format(bucket)

conf={
    # change project id, dataset id, table id
    'mapred.bq.project.id':project,
    'mapred.bq.gcs.bucket':bucket,
    'mapred.bq.temp.gcs.path':input_directory,
    'mapred.bq.input.project.id': "ultra-dimension-300900",
    'mapred.bq.input.dataset.id': 'nyctrip_data',
    'mapred.bq.input.table.id': 'yellow_data_2020_20210309_002840',
}

# Pull table from big query
```



```
table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf = conf)
```

```
# Convert table to a json like object
vals = table_data.values()
vals = vals.map(lambda line: json.loads(line))
pprint.pprint(vals.first())
```

****test job output of *Yellow Taxi Trip Records 2020-01 to 2020-12****

Job output
LINE WRAP: OFF

```

21/03/09 19:22:13 INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat: Total input files to process : 4
[{"tpep_pickup_datetime":"2020-11-13T10:29:43","tpep_pickup_time":"10:29:43","tpep_dropoff_datetime":"2020-11-13T10:40:44","tpep_dropoff_time":"10:40:44","trip
":{"tpep_pickup_datetime":"2020-11-19T09:27:00","tpep_pickup_time":"09:27:00","tpep_dropoff_datetime":"2020-11-19T09:44:00","tpep_dropoff_time":"09:44:00","trip
":{"tpep_pickup_datetime":"2020-11-18T05:53:44","tpep_pickup_time":"05:53:44","tpep_dropoff_datetime":"2020-11-18T06:20:31","tpep_dropoff_time":"06:20:31","trip
":{"tpep_pickup_datetime":"2020-11-20T12:45:00","tpep_pickup_time":"12:45:00","tpep_dropoff_datetime":"2020-11-20T13:58:00","tpep_dropoff_time":"13:58:00","trip
":{"tpep_pickup_datetime":"2020-11-13T11:08:00","tpep_pickup_time":"11:08:00","tpep_dropoff_datetime":"2020-11-13T11:28:00","tpep_dropoff_time":"11:28:00","trip
21/03/09 19:22:18 INFO com.google.cloud.hadoop.repackaged.gcs.com.google.cloud.hadoop.gcsio.GoogleCloudStorageFileSystem: Successfully repaired 'gs://dataproc-
21/03/09 19:22:18 INFO org.sparkproject.jetty.server.AbstractConnector: Stopped Spark@472d7a03(HTTP/1.1, (http/1.1)){0.0.0.0:0}

Job output is complete

```

For now, I just wanted to show that the table loaded successfully as a **JSON** object. We will come back to **Spark** later in the **4. Model** section.

3. Explore (and More Scrub)

A lot of the exploration were done in the previous **2. Scrub** section. Further exploration were done going back to **Dataprep**. Indeed, it seemed like it is much easier to extract **datetime** and **time** further to **year**, **month**, **day**, **hour**, **minute**, and **sec** in **Dataprep**. I went back to pull **datetime** out for all the records.

preview of *Yellow Taxi Trip Records 2020-01 to 2020-12* - Updated:

yellow_data_2020_20210310_070223
🔍 👤 📄 🗑️ DELETE TABLE 📤 EXPORT ▼

Schema Details **Preview**

Row	tpep_pickup_datetime	pickup_hr	pickup_day	pickup_mon	pickup_yr	tpep_dropoff_datetime	dropoff_hr	dropoff_day	dropoff_mon	d
1	2020-01-01T00:30:18	0	1	1	2020	2020-01-01T01:01:03	1	1	1	
2	2020-01-01T00:05:23	0	1	1	2020	2020-01-01T00:18:58	0	1	1	
3	2020-01-01T00:58:46	0	1	1	2020	2020-01-01T01:26:53	1	1	1	
4	2020-01-01T00:34:04	0	1	1	2020	2020-01-01T00:36:44	0	1	1	

Rows per page: 100 ▼ 1 - 100 of 24231126 First page < < > > Last page

Results in this section were mostly done through querying in **BigQuery** first. I attempted to join tables using a wildcard table: [link](#). However, because of some of the syntax differences between **BigQuery** and standard **SQL**, I did not quite succeed and will come back to this if time permits. Since the resulting dataset is relatively small and to save time, I downloaded query results to my local drive and further manipulated (e.g. reformatting, joining) and plotted the data using the **tidyverse**, **dplyr**, and **ggplot2** package of **R** in local drive instead.

Data Analysis Questions

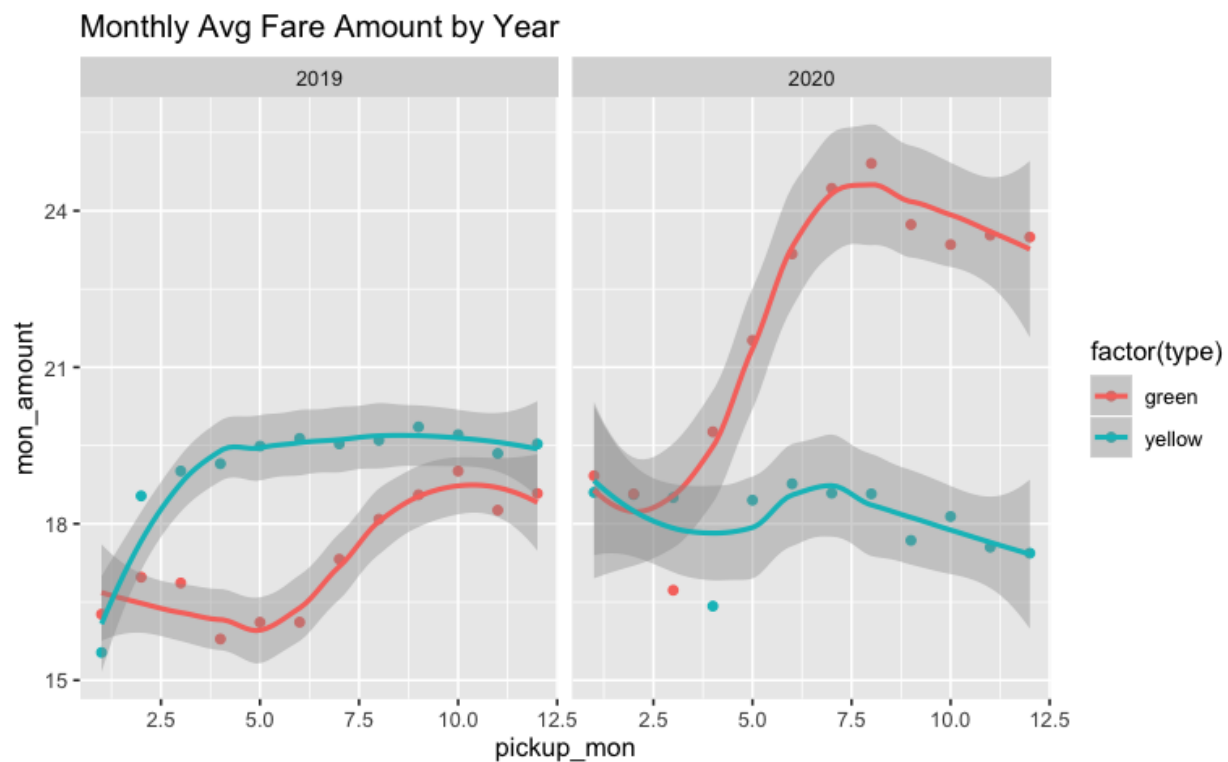
For the 3 data analysis questions, I am interested in:

Question 1: How does monthly average fare amount vary by year (2019 vs 2020) and by taxi type (yellow taxi vs green taxi)?

To answer this question, I used the **AVG** and **GROUP BY** function in **BigQuery** to compute average fare amount by month. Then, I downloaded query results for all records and loaded them in **R**. I further reformatted and combined the dataframes from there since the **ggplot** package of R requires table format to be in the long format (This can be done using the **tidyr** package too.) Then I made some scatterplots and overlaid smooth lines.

Results show that while it appears that monthly average fare amount for yellow taxi shows a decreasing trend and that for green taxi shows an increasing one, the differences may not be significant (Max difference is \$9.116099, which occurs at green taxi trip records.) I am also unsure why there seems to be an increase of monthly avg fare amount for 2020 green taxi but information could often be lost through aggregation. I also think that perhaps median could do a better job than mean does. Further wrangling and/or analysis could look into that.

Figure 1.



snippet of SQL script to query data in BigQuery

```
# Compute monthly avg of total_amount
SELECT
pickup_mon,
AVG(total_amount) AS mon_amount,
FROM trip_data.yellow_data_2019
GROUP BY pickup_mon
```

```
ORDER BY pickup_mon ASC;
```

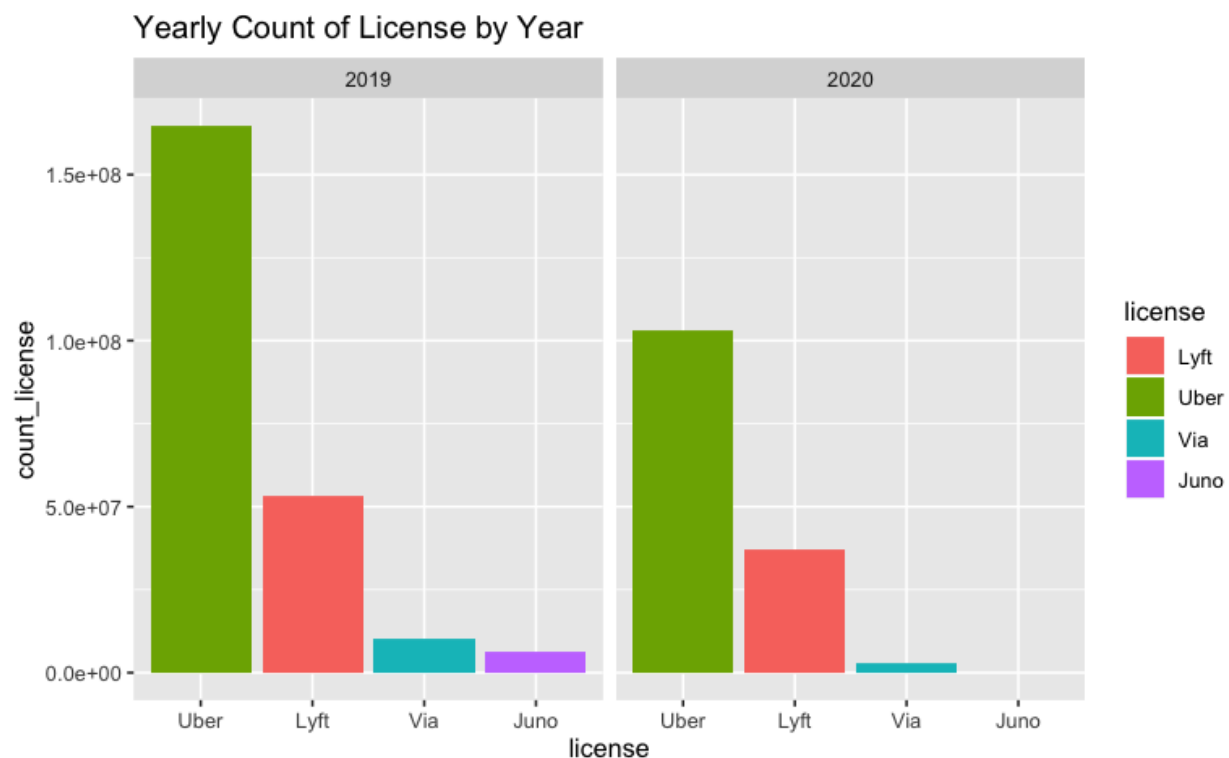
snippet of R script to manipulate and plot data can be found in [Lin_Plotting.pdf](#) (or [Lin_Plotting.Rmd](#))

Question 2: How does yearly total count of high volume fhv (for-hire vehicle) services vary by year (2019 vs 2020) and by license type (Uber, Lyft, Via, Juno)?

To answer this question, I followed similar steps: I counted yearly total count of dispatch by license type for high volume fhv. Recall that to be considered as fhfv, businesses either currently dispatch or plan to dispatch more than 10,000 fhv trips per day in NYC. Note also that Juno is no longer considered as fhfv as of 2020, but I included them in the resulting dataframe for plotting purposes.

Results show that Uber dispatched the most fhv trips overall and Lyft dispatched the second most trips from 2019-2020. And, not to my surprise is that while the number of trips for all businesses still remain high (>10,000 trips/day in NYC), it appears that all businesses have been affected during the the COVID-19 disruption. The number of trips are consistently lower and as mentioned, Juno is no longer considered as fhfv as of 2020. Further wrangling and/or analysis could look into exactly which month did the drop in trip occur. My guess is that it occurred early 2020 when COVID-19 restrictions were in effect in NYC.

Figure 2.



snippet of SQL script to query data in BigQuery

```
# Compute count yearly license
SELECT
  license,
  COUNT(license) AS count_license
FROM trip_data.hfhv_data_2020
GROUP BY license
```

```
ORDER BY count_license DESC;
```

snippet of R script to manipulate and plot data can be found in [Lin_Plotting.pdf](#) (or [Lin_Plotting.Rmd](#))

Question 3: What are the top 6 drop-off locations and how do they vary by year (2019 vs 2020) and by vehicle type (yellow taxi, green taxi, fhv, hfhv)?

To answer this question, I counted yearly total count of drop-off location ID, ordered them and selected top 6 count of drop-off location ID in **BigQuery** for all vehicle types. Once I have query results for all vehicle types, I joined these tables along with the taxi+ zone lookup table using location ID as key and using the `full_join` function in the `dplyr` package of **R**. Then I plotted the count of top 6 drop-off location in term of Borough and Zone by year & by vehicle type.

Results show that while Manhattan is top drop-off location for yellow taxi and Queens, Brooklyn, and Queens are top drop-off locations for hfhv. There are a lot of unknown drop-off location for fhv (<10,000 trips/day in NYC). This is because in the taxi+ zone table ID=264 corresponds to Unknown Borough and NV Zone and ID=265 corresponds to Unknown Borough and null Zone.

Comparing the records across 2019 and 2020, we also see that similar trend as a result of COVID-19 disruption can be observed here. I get curious about specific zone of the drop-off location, so I also include a plot of top drop-off locations further broken down by zone, which is shown in Figure 4.

Figure 3.

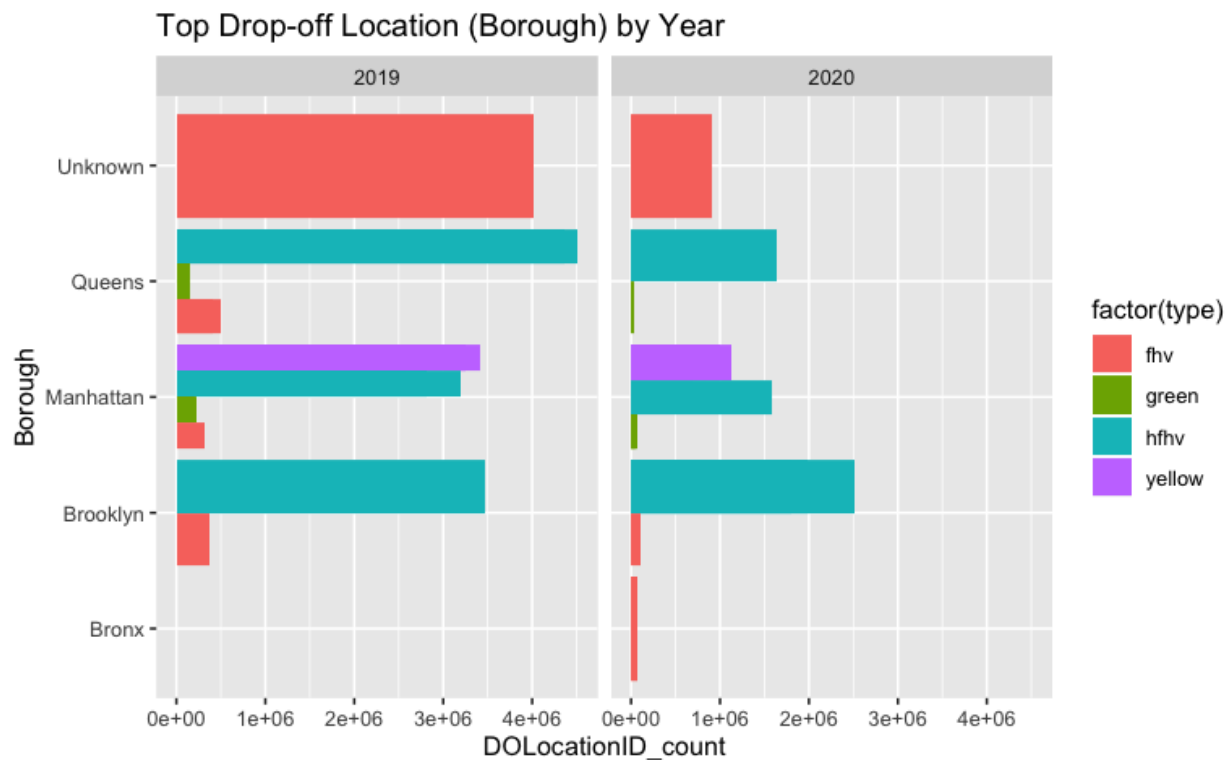
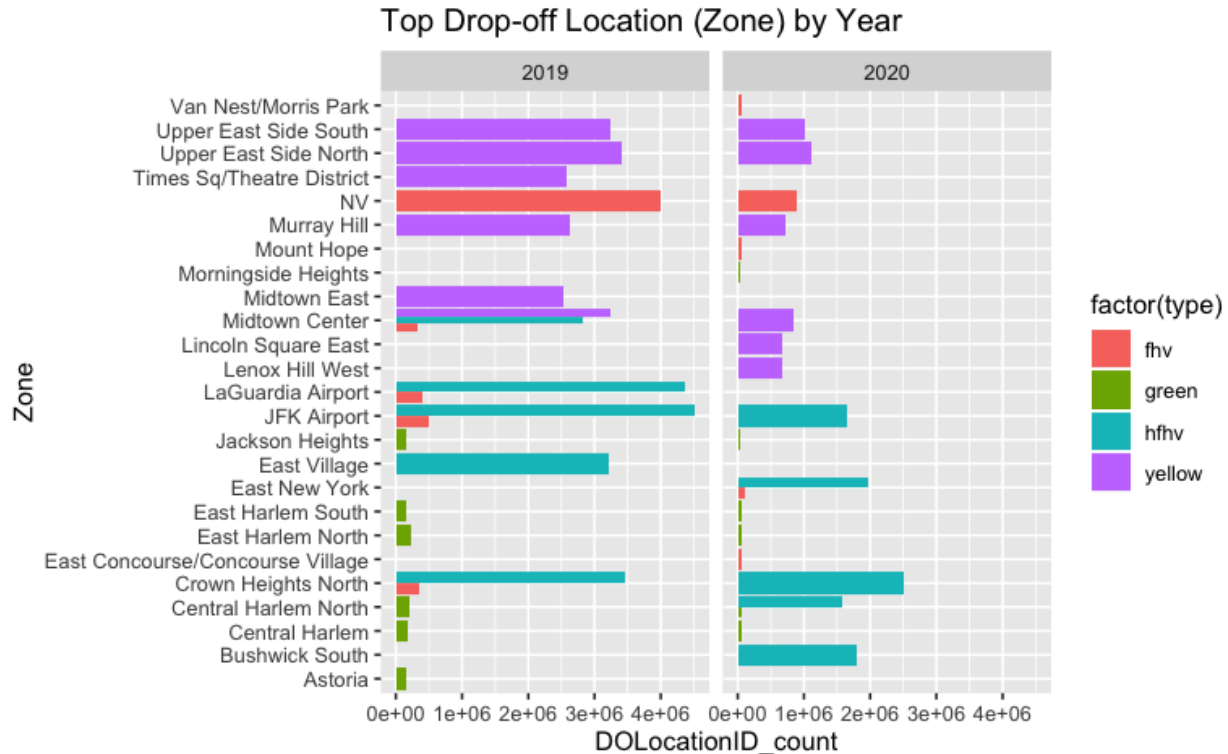


Figure 4.



snippet of SQL script to query data in BigQuery

```
# Compute top 6 count of DOLocationID
SELECT
DISTINCT DOLocationID,
COUNT(DOLocationID) as DOLocationID_count
FROM trip_data.hfhv_data_2020
GROUP BY DOLocationID
ORDER BY DOLocationID_count DESC
LIMIT 6;
```

snippet of R script to manipulate and plot data can be found in [Lin_Plotting.pdf](#) (or [Lin_Plotting.Rmd](#))

4. Model

A lot of the modeling were done in the previous **3. Explore** section. However, during this stage, I notice that there are additional wrangling that needs to be done. Given how much time I have left for this project and the remaining days of **Google Cloud**'s free trial, I am just gonna do as much as I could and leave the rest for future analysis.

To PySpark - Updated

To load and model data in **Spark**, I revisited Week 7's Spark and Week 8's Spark Specifics Module.

Bonus Question 1: How does monthly average fare amount vary for 2020 yellow taxi records?

I have to admit that spending 1-2 weeks on **Spark** prior to completing this project is NOT enough. Instead of going back to work on additional wrangling in **Data**, I decide to spend the remaining 2 days working in

Spark to get additional practice for it, even if it means I will lose points for accuracy for this project.

Average trip distance (in meter) and average fare amount (in \$) were computed loading the table from **BigQuery** and converting it to a **PySpark** dataframe first. Then, I computed mean, standard deviation and count of average trip distance and average fare amount by month using the **mean**, **stddev** and **count** along with the **groupBy** and **agg** function from the **pyspark.sql** module.

Results show that May has the longest mean trip distance (8.6103 m) and June has the highest mean fare amount (\$18.7660). However, since the standard deviation is too high for both **trip_distance** and **total_amount** for some months, further wrangling should consider creating an IQR (interquartile range) function either using build-in or **map(lambda x:)** function to identify outliers and further filter out the outliers.

```
+-----+-----+-----+-----+
|pickup_mon|avg(trip_distance)|stddev_samp(trip_distance)|count(trip_distance)|
+-----+-----+-----+-----+
|          7|4.4518635390300245|          482.28774289655644|          772188|
|         11| 4.205144641983849|          445.6264180067282|         1483188|
|          3|3.1527776105306087|          208.3966193939616|         2965225|
|          8| 4.51186787270762|          339.0286619643721|          975016|
|          5| 8.610368613222507|          788.0716701210021|          336793|
|          9| 4.365925615783489|          446.6348380792307|          1312695|
|          6| 4.246858455393626|           342.203886469089|          530081|
|          1| 2.968089891868472|           83.73389022813812|         6317192|
|         10| 3.576072340784534|          282.6028806567841|         1650166|
|         12| 6.163896306510167|          894.0040816367627|         1437551|
|          4| 4.157984192049327|          299.66132677636926|          230456|
|          2|2.8946657780171594|          40.527459490047455|         6220575|
+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
|pickup_mon| avg(total_amount)|stddev_samp(total_amount)|count(total_amount)|
+-----+-----+-----+-----+
|          7|18.585729792231376|          14.33372914300685|          772188|
|         11| 17.55121064270372|         124.93683514431557|         1483188|
|          3|18.501154053175494|          389.9437664659958|         2965225|
|          8|18.572788916927294|          190.3048652763212|          975016|
|          5| 18.45184195321015|          14.79325219081066|          336793|
|          9|17.680980998195682|          11.85518455438868|          1312695|
|          6|18.766022097942624|          14.672790280717793|          530081|
|          1|18.600994187967427|          14.101483341856811|         6317192|
|         10|18.138897976417812|          777.2273359813906|         1650166|
|         12|17.435560898038982|          13.612546930941804|         1437551|
|          4|16.421087725317346|          12.463365276397795|          230456|
|          2|18.558698606989672|          13.853541917402083|         6220575|
+-----+-----+-----+-----+
```

snippet of Python script and job output can be found in the Appendix section of this .pdf

Bonus Question 2: What is the yearly average fare amount and avg distance for 2020 yellow taxi records?

It has become very clear that we will either have to do further wrangling here in **PySpark** and **Dataprocc** or go back to **Dataprep** for it since ensuring these values are non-negative in **Dataprep** is not enough. The output is shown as follows.

```

+-----+-----+
|summary|    trip_distance|
+-----+-----+
|  count|          24231126|
|   mean|3.5814057339315424|
| stddev| 327.8228554760617|
|    min|              0.01|
|    max|          350914.88|
+-----+-----+

+-----+-----+
|summary|    total_amount|
+-----+-----+
|  count|          24231126|
|   mean|18.342393123230288|
| stddev|249.58870028047656|
|    min|              0.3|
|    max|          998325.6|
+-----+-----+

```

snippet of Python script and job output can be found in the Appendix section of this .pdf

5. Interpreting (and Discussing)

Results can be found in the **3. Explore** section, but to sum up, it appeared that the total amount of trips have reduced since the COVID-19 disruption. Specifically, monthly average fare amount for yellow taxi shows a decreasing trend from 2019-2020. In addition, the number of trips from 2020 for all hfhv businesses are consistently lower than that from 2019.

Results are not error free, but I am okay with what I have at this point. After all, the purpose of EDA (e.g. plot, summary statistics) is to ensure that the values are accurate and questions are not only interesting but also potentially statistically significant prior to doing any modeling, which I unfortunately ran out of time for. The following is what I will do in **Spark** next if time permits:

1. Create the IQR function to identify outliers
2. Remove the outliers
3. Re-compute summary statistics, etc

Finally, I plan to sign up CS513 Applied Machine Learning and use this dataset for further wrangling and analysis, but my **Google Cloud** free trial is few days away from expiring. I also just get accepted to the PhD in statistics program here at OSU, so I am going to take my time to take CS534 Machine Learning instead — what a challenging yet rewarding class it is, all I want to say!

Appendix

snippet of Python script to run Dataproc job — Bonus Question 1

```
import pyspark
from pyspark.sql import SparkSession
import pprint
import json
from pyspark.sql.types import StructField, StructType
from pyspark.sql.types import StringType, FloatType, IntegerType, DateType, TimestampType
from pyspark.sql import functions

# Connect to Spark
sc = pyspark.SparkContext() # run Spark applications
#PACKAGE_EXTENSIONS= ('gs://hadoop-lib/bigquery/bigquery-connector-hadoop2-latest.jar')

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output'.format(bucket)

conf={
    # change project id, dataset id, table id
    'mapred.bq.project.id':project,
    'mapred.bq.gcs.bucket':bucket,
    'mapred.bq.temp.gcs.path':input_directory,
    'mapred.bq.input.project.id': "ultra-dimension-300900",
    'mapred.bq.input.dataset.id': 'trip_data',
    'mapred.bq.input.table.id': 'yellow_data_2020_20210310_070223',
}

# Pull table from big query
table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf = conf)

# Convert table to a json like object
vals = table_data.values()
vals = vals.map(lambda line: json.loads(line))
#pprint.pprint(vals.first()) # good as of 03/13/2021

# Define a To_numb function
def To_numb(x):
    x['pickup_hr'] = int(x['pickup_hr'])
    x['pickup_day'] = int(x['pickup_day'])

    x['pickup_yr'] = int(x['pickup_yr'])
    x['dropoff_hr'] = int(x['pickup_hr'])
    x['dropoff_day'] = int(x['pickup_day'])
    x['dropoff_mon'] = int(x['pickup_mon'])
    x['dropoff_yr'] = int(x['pickup_yr'])
    x['trip_distance'] = float(x['trip_distance'])
```



```

x['PULocationID'] = int(x['PULocationID'])
x['DOLocationID'] = int(x['DOLocationID'])
x['total_amount'] = float(x['total_amount'])
return x

# Apply To_numb function to int or float variables otherwise schema won't work
vals = vals.map(To_numb)

# Create a dataframe object

# schema
# https://spark.apache.org/docs/3.0.0-preview/sql-ref-datatypes.html
schema = StructType([
    StructField('tpep_pickup_datetime', StringType(), True), # TimestampType() Not in the form?
    StructField("pickup_hr", IntegerType(), True),
    StructField("pickup_day", IntegerType(), True),
    StructField("pickup_mon", StringType(), True), # change back to StringType()
    StructField("pickup_yr", IntegerType(), True),
    StructField("tpep_dropoff_datetime", StringType(), True), # TimestampType() Not in the form?
    StructField("dropoff_hr", IntegerType(), True),
    StructField("dropoff_day", IntegerType(), True),
    StructField("dropoff_mon", IntegerType(), True),
    StructField("dropoff_yr", IntegerType(), True),
    StructField("trip_distance", FloatType(), True),
    StructField("PULocationID", IntegerType(), True),
    StructField("DOLocationID", IntegerType(), True),
    StructField("total_amount", FloatType(), True)])

#pprint.pprint(vals.first()) # good as of 03/13/2021

# Initialize spark
# https://spark.apache.org/docs/2.0.0/sql-programming-guide.html#sql
spark = SparkSession\
    .builder\
    .appName("PythonSQL")\
    .getOrCreate()

# Create a df
df1 = spark.createDataFrame(vals, schema= schema)
df1.repartition(6) # partition to 6 partitions # could partition by key too

#pprint.pprint(vals.first()) # good as of 03/09/2020

# Need a To_numb function as well

# Compute summary statistics
#df1.describe("trip_distance").show() # good
#df1.describe(["trip_distance", "total_amount"]).show() # does not work

#df1.describe("trip_distance").show()
#df1.describe("total_amount").show()

# Query data

```

```
# https://towardsdatascience.com/beginners-guide-to-pyspark-bbe3b553b79f

# Compute monthly avg trip_distance & total_amount
# df1.select('trip_distance'
#           ).groupBy('pickup_mon')\
#           .mean()\
#           .show()
#AttributeError: 'GroupedData' object has no attribute 'describe'

# Try this instead
# https://stackoverflow.com/questions/51632126/pyspark-how-to-calculate-avg-and-count-in-a-single-groupby
df1.groupBy('pickup_mon').agg(functions.mean('trip_distance'), functions.stddev('trip_distance'), functions.count('trip_distance'))
df1.groupBy('pickup_mon').agg(functions.mean('total_amount'), functions.stddev('total_amount'), functions.count('total_amount'))

# Compute stddev next
# Could probably compute IQR from functions or from the stddev from above too

# Consider map(lambda x:) for individual variable and compute summary statistics; it may be faster?

# Delete the temporary files
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)

## Back to Google Cloud, Week 7
## Upload this file to Storage's cs512_trip
## Create cluster and submit job in Dataproc
## Copy & paste this to Jar files
## gs://hadoop-lib/bigquery/bigquery-connector-hadoop2-latest.jar
```

Job output — Bonus Question 1

```
21/03/15 23:21:58 INFO org.sparkproject.jetty.util.log: Logging initialized @4408ms to org.sparkproject
21/03/15 23:21:58 INFO org.sparkproject.jetty.server.Server: jetty-9.4.36.v20210114; built: 2021-01-14T
21/03/15 23:21:58 INFO org.sparkproject.jetty.server.Server: Started @4528ms
21/03/15 23:21:58 INFO org.sparkproject.jetty.server.AbstractConnector: Started ServerConnector@1cc6b5d
21/03/15 23:21:59 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at cluster-
21/03/15 23:21:59 INFO org.apache.hadoop.yarn.client.AHSPProxy: Connecting to Application History server
21/03/15 23:21:59 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
21/03/15 23:21:59 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-type
21/03/15 23:22:02 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application app
21/03/15 23:22:03 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at cluster-
21/03/15 23:22:05 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: BigQuery connector version 1
21/03/15 23:22:05 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from defa
21/03/15 23:22:05 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from giv
21/03/15 23:22:05 INFO com.google.cloud.hadoop.io.bigquery.BigQueryConfiguration: Using working path: '
21/03/15 23:25:51 INFO com.google.cloud.hadoop.io.bigquery.UnshardedExportToCloudStorage: Setting FileI
21/03/15 23:25:51 INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat: Total input files to proc
+-----+-----+-----+-----+
|pickup_mon|avg(trip_distance)|stddev_samp(trip_distance)|count(trip_distance)|
+-----+-----+-----+-----+
|          7|4.4518635390300245|          482.28774289655644|          772188|
```

11	4.205144641983849	445.6264180067282	1483188
3	3.1527776105306087	208.3966193939616	2965225
8	4.51186787270762	339.0286619643721	975016
5	8.610368613222507	788.0716701210021	336793
9	4.365925615783489	446.6348380792307	1312695
6	4.246858455393626	342.203886469089	530081
1	2.968089891868472	83.73389022813812	6317192
10	3.576072340784534	282.6028806567841	1650166
12	6.163896306510167	894.0040816367627	1437551
4	4.157984192049327	299.66132677636926	230456
2	2.8946657780171594	40.527459490047455	6220575

pickup_mon	avg(total_amount)	stddev_samp(total_amount)	count(total_amount)
7	18.585729792231376	14.33372914300685	772188
11	17.55121064270372	124.93683514431557	1483188
3	18.501154053175494	389.9437664659958	2965225
8	18.572788916927294	190.3048652763212	975016
5	18.45184195321015	14.79325219081066	336793
9	17.680980998195682	11.85518455438868	1312695
6	18.766022097942624	14.672790280717793	530081
1	18.600994187967427	14.101483341856811	6317192
10	18.138897976417812	777.2273359813906	1650166
12	17.435560898038982	13.612546930941804	1437551
4	16.421087725317346	12.463365276397795	230456
2	18.558698606989672	13.853541917402083	6220575

```
21/03/15 23:31:18 INFO com.google.cloud.hadoop.repackaged.gcs.com.google.cloud.hadoop.gcsio.GoogleCloudStorage
21/03/15 23:31:18 INFO org.sparkproject.jetty.server.AbstractConnector: Stopped Spark@1cc6b5d9{HTTP/1.1}
Job output is complete
```

snippet of Python script to run Dataproc job — Bonus Question 2

```
import pyspark
from pyspark.sql import SparkSession
import pprint
import json

from pyspark.sql.types import StructField, StructType
from pyspark.sql.types import StringType, FloatType, IntegerType, DateType, TimestampType

# Connect to Spark
sc = pyspark.SparkContext() # run Spark applications
#PACKAGE_EXTENSIONS= ('gs://hadoop-lib/bigquery/bigquery-connector-hadoop2-latest.jar')

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output'.format(bucket)

conf={
    # change project id, dataset id, table id
```

```

    'mapred.bq.project.id':project,
    'mapred.bq.gcs.bucket':bucket,
    'mapred.bq.temp.gcs.path':input_directory,
    'mapred.bq.input.project.id': "ultra-dimension-300900",
    'mapred.bq.input.dataset.id': 'trip_data',
    'mapred.bq.input.table.id': 'yellow_data_2020_20210310_070223',
}

# Pull table from big query
table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf = conf)

# Convert table to a json like object
vals = table_data.values()
vals = vals.map(lambda line: json.loads(line))
#pprint.pprint(vals.first()) # good as of 03/13/2021

# Define a To_numb function
def To_numb(x):
    x['pickup_hr'] = int(x['pickup_hr'])
    x['pickup_day'] = int(x['pickup_day'])
    x['pickup_mon'] = int(x['pickup_mon'])
    x['pickup_yr'] = int(x['pickup_yr'])
    x['dropoff_hr'] = int(x['pickup_hr'])
    x['dropoff_day'] = int(x['pickup_day'])
    x['dropoff_mon'] = int(x['pickup_mon'])
    x['dropoff_yr'] = int(x['pickup_yr'])
    x['trip_distance'] = float(x['trip_distance'])
    x['PULocationID'] = int(x['PULocationID'])
    x['DOLocationID'] = int(x['DOLocationID'])
    x['total_amount'] = float(x['total_amount'])
    return x

# Apply To_numb function to int or float variables otherwise schema won't work
vals = vals.map(To_numb)

# Create a dataframe object

# schema
# https://spark.apache.org/docs/3.0.0-preview/sql-ref-datatypes.html
schema = StructType([
    StructField('tpep_pickup_datetime', StringType(), True), # TimestampType() Not in the form?
    StructField("pickup_hr", IntegerType(), True),
    StructField("pickup_day", IntegerType(), True),
    StructField("pickup_mon", IntegerType(), True),
    StructField("pickup_yr", IntegerType(), True),
    StructField("tpep_dropoff_datetime", StringType(), True), # TimestampType() Not in the form?
    StructField("dropoff_hr", IntegerType(), True),
    StructField("dropoff_day", IntegerType(), True),
    StructField("dropoff_mon", IntegerType(), True),
    StructField("dropoff_yr", IntegerType(), True),

```

```

    StructField("trip_distance", FloatType(), True),
    StructField("PULocationID", IntegerType(), True),
    StructField("DOLocationID", IntegerType(), True),
    StructField("total_amount", FloatType(), True)])

# pprint.pprint(vals.first()) # good as of 03/13/2021

# Initialize spark
# https://spark.apache.org/docs/2.0.0/sql-programming-guide.html#sql
spark = SparkSession\
    .builder\
    .appName("PythonSQL")\
    .getOrCreate()

# Create a df
df1 = spark.createDataFrame(vals, schema= schema)
df1.repartition(6) # partition to 6 partitions

# pprint.pprint(vals.first()) # good as of 03/09/2020

# Need a To_numb function as well

# Compute summary statistics
# df1.describe("trip_distance").show() # good
# df1.describe(["trip_distance", "total_amount"]).show() # does not work

df1.describe("trip_distance").show()
df1.describe("total_amount").show()

# Delete the temporary files
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)

## Back to Google Cloud, Week 7
## Upload this file to Storage's cs512_trip
## Create cluster and submit job in Dataproc
## Copy & paste this to Jar files
## gs://hadoop-lib/bigquery/bigquery-connector-hadoop2-latest.jar

```

Job output — Bonus Question 2

```

21/03/14 03:10:05 INFO org.sparkproject.jetty.util.log: Logging initialized @4957ms to org.sparkproject
21/03/14 03:10:06 INFO org.sparkproject.jetty.server.Server: jetty-9.4.36.v20210114; built: 2021-01-14T
21/03/14 03:10:06 INFO org.sparkproject.jetty.server.Server: Started @5091ms
21/03/14 03:10:06 INFO org.sparkproject.jetty.server.AbstractConnector: Started ServerConnector@4eed634
21/03/14 03:10:06 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at cluster-1
21/03/14 03:10:07 INFO org.apache.hadoop.yarn.client.AHSPProxy: Connecting to Application History server
21/03/14 03:10:07 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
21/03/14 03:10:07 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-typ
21/03/14 03:10:09 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application app
21/03/14 03:10:10 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at cluster-1
21/03/14 03:10:13 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: BigQuery connector version 1
21/03/14 03:10:13 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from defa

```

```

21/03/14 03:10:13 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from given
21/03/14 03:10:13 INFO com.google.cloud.hadoop.io.bigquery.BigQueryConfiguration: Using working path: '/'
21/03/14 03:12:31 INFO com.google.cloud.hadoop.io.bigquery.UnshardedExportToCloudStorage: Setting FileI
21/03/14 03:12:32 INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat: Total input files to proc

```

```

+-----+-----+
|summary|    trip_distance|
+-----+-----+
|  count|          24231126|
|   mean|3.5814057339315424|
| stddev|327.8228554760617|
|    min|              0.01|
|    max|          350914.88|
+-----+-----+

```

```

+-----+-----+
|summary|    total_amount|
+-----+-----+
|  count|          24231126|
|   mean|18.342393123230288|
| stddev|249.58870028047656|
|    min|              0.3|
|    max|          998325.6|
+-----+-----+

```

```

21/03/14 03:18:02 INFO com.google.cloud.hadoop.repackaged.gcs.com.google.cloud.hadoop.gcsio.GoogleCloud
21/03/14 03:18:02 INFO org.sparkproject.jetty.server.AbstractConnector: Stopped Spark@4eed6343{HTTP/1.1
Job output is complete

```