Pujan Patel
CIS 4130
Dr. Richard Holowczak
Semester Project
https://github.com/Pupat3l/NYC-trips

Proposal

The dataset that I have decided to utilize for the final project is called "NYC FHV(Uber/Lyft) Trip Data Expanded (2019-2022)" and can be found on the Kaggle website. The dataset provides detailed information about every For-Hire Vehicle (which includes Uber and Lyft) rides in the New York City. The dataset was originally found on the official nyc.gov website. Information regarding the trip included in the dataset is TLC license number, date and time of the pick-up and drop-off, taxi zone, passenger pick-up and drop-off time, and tip etc. The dataset also has information regarding the distance travelled, NYC taxes, tolls, and surcharges for the LGA and JFK airport. Overall, it includes all the data generated from a ride.

There is a link for both the original dataset and the dataset from Kaggle that can be downloaded and be used for analysis. The link for the download from Kaggle is here. The size of data is 19 GB and is accounted from 2019-2022. The update is annual therefore, the next update would be in 2023.

Talking about the dataset, the dataset has potential for many analyses and predictions. However, I was considering one of two options which were tip forecasting or revenue forecasting. Tip forecasting would aim to predict the amount of tip a customer will give based on the variables like trip time, trip miles, trip cost, tax, airport fees, tolls, etc. Revenue forecasting is slightly easier where it would focus on generating predictions for revenues based on historic data i.e. revenue trend on the previous rides multiplied by an inflation index to get a proximate ride today. Sticking to only one of these two option, my main priority would be the tip forecasting.

Data Acquisition

Since, the collection of data that I will be using is ranging from 2019-2022 and the size of the data is 19 GB, I have decided that I will extract the data from the source and store it in AWS Simple Storage System (S3) bucket. This will allow me to utilize cloud resource and computing power which would be more efficient in comparison to my MacBook Air. Therefore, in this milestone I created an EC2 instance which will allow me to utilize the server to download the data directly from the source and store it on EC2. I attempted to utilize Kaggle's interface to download the data in EC2. Below is the code snippet used to accomplish that, and a picture of the Kaggle dataset files downloaded.

```
#installed kaggle using the commands from class
pip3 install kaggle
```

```
mkdir .kaggle
nano .kaggle/kaggle.json
#entered my Kaggle API Key and saved
#secured the folder
chmod 600 .kaggle/kaggle.json
#check
kaggle datasets list
#API Command: kaggle datasets download -d jeffsinsel/nyc-fhvhv-
data
#Created S3 bucket and folders
#checked the files from the target dataset
kaggle datasets files -d jeffsinsel/nyc-fhvhv-data
```

```
[ec2-user@ip-172-31-24-246 ~]$ kaggle datasets files -d jeffsinsel/nyc-fhvhv-data
name                                     size  creationDate
---------------------------------------  -----  -------------------
fhvhv_tripdata_2020-02.parquet           533MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-04.parquet           110MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-07.parquet           492MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-04.parquet           351MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-05.parquet           153MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-11.parquet           288MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-07.parquet           423MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-02.parquet           388MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-06.parquet           376MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-11.parquet           392MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-03.parquet           449MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-08.parquet           416MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-08.parquet           365MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-11.parquet           535MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-11.parquet           443MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-05.parquet           369MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-03.parquet           330MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-04.parquet           434MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-01.parquet           357MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-02.parquet           489MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-03.parquet           583MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-04.parquet           534MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-08.parquet           277MB  2023-03-10 01:44:27
data_dictionary_trip_records_hvfhs.pdf   126KB  2023-03-10 01:44:27
fhvhv_tripdata_2020-09.parquet           300MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-10.parquet           472MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-01.parquet           295MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-06.parquet           511MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-07.parquet           248MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-02.parquet           289MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-05.parquet           544MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-12.parquet           289MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-09.parquet           375MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-09.parquet           437MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-03.parquet           351MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-09.parquet           492MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-12.parquet           392MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-01.parquet           507MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-10.parquet           410MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-10.parquet           524MB  2023-03-10 01:44:27
fhvhv_tripdata_2019-08.parquet           478MB  2023-03-10 01:44:27
fhvhv_tripdata_2021-07.parquet           378MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-06.parquet           189MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-06.parquet           437MB  2023-03-10 01:44:27
taxi+_zone_lookup.csv                     12KB  2023-03-10 01:44:27
fhvhv_tripdata_2019-12.parquet           550MB  2023-03-10 01:44:27
fhvhv_tripdata_2022-05.parquet           447MB  2023-03-10 01:44:27
fhvhv_tripdata_2020-10.parquet           329MB  2023-03-10 01:44:27
[ec2-user@ip-172-31-24-246 ~]$
```

After that I created the S3 bucket named ‘pp-nyc-trip-data’ to store the data for this project and created specific folders for the requirement. Then I downloaded the individual dataset to EC2 and then unzipped the file and then copied the file to S3 bucket. Below is the code snippet to achieve (Appendix A) that I ran to do the process:

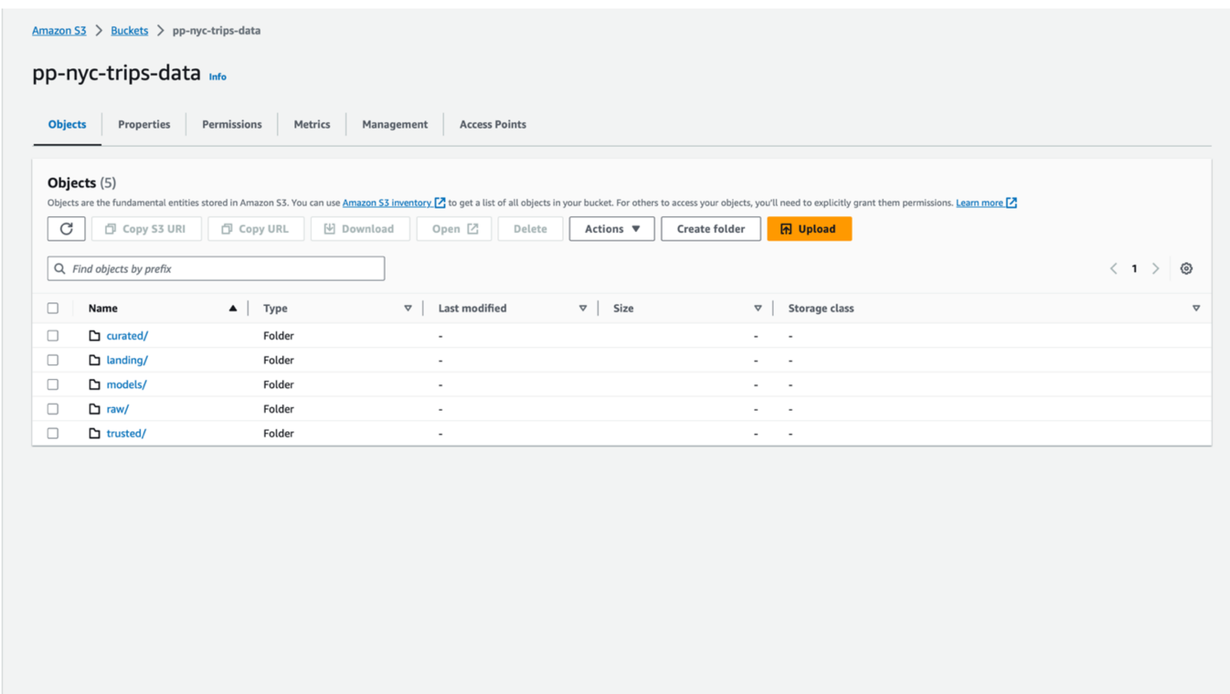
```
#downloaded file
kaggle datasets download -d jeffsinsel/nyc-fhvhv-data -f
fhvhv_tripdata_2019-02.parquet
#unzip
unzip fhvhv_tripdata_2019-02.parquet
#copying to s3
aws s3 cp fhvhv_tripdata_2019-02.parquet s3://pp-nyc-trips-
data/landing/fhvhv_tripdata_2019-02.parquet
#checking if its visible in s3
aws s3 ls s3://pp-nyc-trips-data/landing/
```

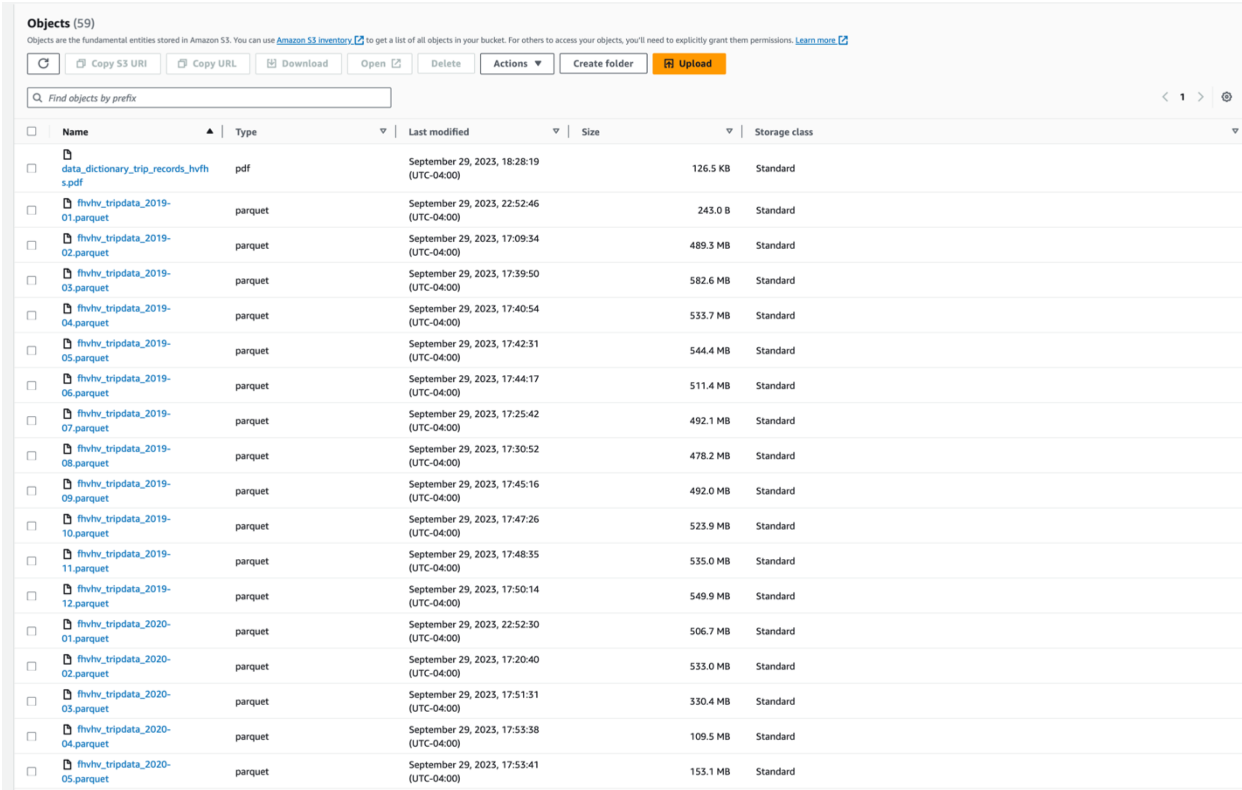
```
#yes
#removed the files from the local system
rm fhvhv_tripdata_2019-02.parquet.zip
rm fhvhv_tripdata_2019-02.parquet
```

This process was repeated until I ran into a '404 – Not found' error while downloading so after consulting with professor, I ran the Curl method to download the files to EC2 and then used the part of the code from above to transport the downloaded file to S3 bucket. Below is the code snipped (Appendix A) that shows the Curl method and the pic of result in EC2.

```
curl -L -o taxi_zones.shx
https://d37ci6vzurychx.cloudfront.net/trip-data/taxi_zones/taxi_zones.shx
```

At the end of this process, I was able to extract the data from the source and store it to S3 bucket in /landing folder. Below are the pics of bucket and the landing folder depicting the success in extracting the dataset.





Objects (59)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
data_dictionary_trip_records_hvfhs.pdf	pdf	September 29, 2023, 18:28:19 (UTC-04:00)	126.5 KB	Standard
fhvhv_tripdata_2019-01.parquet	parquet	September 29, 2023, 22:52:46 (UTC-04:00)	243.0 B	Standard
fhvhv_tripdata_2019-02.parquet	parquet	September 29, 2023, 17:09:34 (UTC-04:00)	489.3 MB	Standard
fhvhv_tripdata_2019-03.parquet	parquet	September 29, 2023, 17:39:50 (UTC-04:00)	582.6 MB	Standard
fhvhv_tripdata_2019-04.parquet	parquet	September 29, 2023, 17:40:54 (UTC-04:00)	533.7 MB	Standard
fhvhv_tripdata_2019-05.parquet	parquet	September 29, 2023, 17:42:31 (UTC-04:00)	544.4 MB	Standard
fhvhv_tripdata_2019-06.parquet	parquet	September 29, 2023, 17:44:17 (UTC-04:00)	511.4 MB	Standard
fhvhv_tripdata_2019-07.parquet	parquet	September 29, 2023, 17:25:42 (UTC-04:00)	492.1 MB	Standard
fhvhv_tripdata_2019-08.parquet	parquet	September 29, 2023, 17:30:52 (UTC-04:00)	478.2 MB	Standard
fhvhv_tripdata_2019-09.parquet	parquet	September 29, 2023, 17:45:16 (UTC-04:00)	492.0 MB	Standard
fhvhv_tripdata_2019-10.parquet	parquet	September 29, 2023, 17:47:26 (UTC-04:00)	523.9 MB	Standard
fhvhv_tripdata_2019-11.parquet	parquet	September 29, 2023, 17:48:35 (UTC-04:00)	535.0 MB	Standard
fhvhv_tripdata_2019-12.parquet	parquet	September 29, 2023, 17:50:14 (UTC-04:00)	549.9 MB	Standard
fhvhv_tripdata_2020-01.parquet	parquet	September 29, 2023, 22:52:30 (UTC-04:00)	506.7 MB	Standard
fhvhv_tripdata_2020-02.parquet	parquet	September 29, 2023, 17:20:40 (UTC-04:00)	533.0 MB	Standard
fhvhv_tripdata_2020-03.parquet	parquet	September 29, 2023, 17:51:31 (UTC-04:00)	330.4 MB	Standard
fhvhv_tripdata_2020-04.parquet	parquet	September 29, 2023, 17:53:38 (UTC-04:00)	109.5 MB	Standard
fhvhv_tripdata_2020-05.parquet	parquet	September 29, 2023, 17:53:41 (UTC-04:00)	153.1 MB	Standard

Exploratory Data Analysis

After many tries and finally using t2.xl on EC2 instance, I was finally able to finished my milestone 3. I analyzed 3 files from same month but different years. Files analyzed were "fhvhv_tripdata_2019-05.parquet", "fhvhv_tripdata_2020-05.parquet", and "fhvhv_tripdata_2021-05.parquet" and were extracted from my AWS S3 bucket. I utilized jupyter notebook on EC2 instance to finish the EDA. Libraries used were pandas, ParquetFile, and pyarrow. After reading and loading the data of 2019 to trips_2019_df, the result of analysis show the following:

There are 24 columns in the dataset with 22,329,247 entries in the dataset. The name of the columns and their data types are displayed below. Here is the summary summary statistics for trips_2019_df:

```python
trips_2019_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22329247 entries, 0 to 22329246
Data columns (total 24 columns):
 #   Column                Dtype
---  ------                -----
 0   hvfhs_license_num     object
 1   dispatching_base_num  object
 2   originating_base_num  object
 3   request_datetime      datetime64[us]
 4   on_scene_datetime     datetime64[us]
 5   pickup_datetime       datetime64[us]
 6   dropoff_datetime      datetime64[us]
 7   PULocationID          int64
 8   DOLocationID          int64
 9   trip_miles            float64
 10  trip_time             int64
 11  base_passenger_fare   float64
 12  tolls                 float64
 13  bcf                   float64
 14  sales_tax             float64
 15  congestion_surcharge  float64
 16  airport_fee           object
 17  tips                  float64
 18  driver_pay            float64
 19  shared_request_flag   object
 20  shared_match_flag     object
 21  access_a_ride_flag    object
 22  wav_request_flag      object
 23  wav_match_flag        object
dtypes: datetime64[us](4), float64(8), int64(3), object(9)
memory usage: 4.0+ GB
```

```python
trips_2019_df.describe()
```

	request_datetime	on_scene_datetime	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	trip_miles	trip_time	base_passenger_fa
count	22329242	16144607	22329247	22329247	2.232925e+07	2.232925e+07	2.232925e+07	2.232925e+07	2.232925e+(
mean	2019-05-16 05:00:14.684075	2018-07-21 12:58:15.585311	2019-05-16 05:05:27.939704	2019-05-16 05:25:14.753266	1.387979e+02	1.416115e+02	4.746521e+00	1.177314e+03	1.774145e+(
min	2019-04-30 23:38:11	1970-01-01 00:00:00	2019-05-01 00:00:00	2019-05-01 00:02:27	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	-1.189000e+(
25%	2019-05-08 14:21:39.250000	2019-05-07 21:47:20.500000	2019-05-08 14:26:18	2019-05-08 14:49:23	7.500000e+01	7.500000e+01	1.580000e+00	5.850000e+02	7.370000e+(
50%	2019-05-15 23:53:31.500000	2019-05-15 15:21:27	2019-05-15 23:58:07	2019-05-16 00:16:15	1.400000e+02	1.410000e+02	2.890000e+00	9.450000e+02	1.206000e+(
75%	2019-05-23 19:23:52	2019-05-23 13:47:49	2019-05-23 19:28:19	2019-05-23 19:49:02	2.110000e+02	2.170000e+02	5.743000e+00	1.511000e+03	2.126000e+(
max	2019-05-31 23:59:04	2019-05-31 23:59:53	2019-05-31 23:59:59	2019-06-01 13:12:50	2.650000e+02	2.650000e+02	4.177900e+02	8.206100e+04	2.371270e+(
std	NaN	NaN	NaN	NaN	7.513764e+01	7.753089e+01	5.523643e+00	8.726274e+02	1.794388e+(

There are several missing fields in the dataset. The number of fields missing in each column are displayed below in the picture. The airport_fee column has many null values which indicates that the ride wasn't to/from airport.

The maximum and minimum date in the dataset are provided below in the picture. As you can notice, there is an invalid data entry here which is "1970-01-01 00:00:00" in "on_screen_datetime". This indicates the data needs some cleaning.

```
missing_values

hvfhs_license_num              0
dispatching_base_num         293
originating_base_num     5710175
request_datetime               5
on_scene_datetime        6184640
pickup_datetime                0
dropoff_datetime               0
PULocationID                   0
DOLocationID                   0
trip_miles                     0
trip_time                      0
base_passenger_fare            0
tolls                          0
bcf                            0
sales_tax                      0
congestion_surcharge           0
airport_fee             22329247
tips                           0
driver_pay                     0
shared_request_flag            0
shared_match_flag              0
access_a_ride_flag             0
wav_request_flag               0
wav_match_flag          14539567
dtype: int64
```

```
print(min_date)
print(max_date)
```

```
request_datetime    2019-04-30 23:38:11
on_scene_datetime   1970-01-01 00:00:00
pickup_datetime     2019-05-01 00:00:00
dropoff_datetime    2019-05-01 00:02:27
dtype: datetime64[us]
request_datetime    2019-05-31 23:59:04
on_scene_datetime   2019-05-31 23:59:53
pickup_datetime     2019-05-31 23:59:59
dropoff_datetime    2019-06-01 13:12:50
dtype: datetime64[us]
```

After reading and loading the data of 2020 to trips_2020_df, the result of analysis show the following: The trip data from 2020 also indicates the same column name and types. You can notice how the entries has dropped to 6,089,998 in the dataset. The reason behind was the covid-19 pandemic during which quarantine measures were in place and limited travel was recommended.

Here is the summary statistic for trips_2020_df:

```
trips_2020_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6089999 entries, 0 to 6089998
Data columns (total 24 columns):
 #   Column                 Dtype
---  ------                 -----
 0   hvfhs_license_num      object
 1   dispatching_base_num   object
 2   originating_base_num   object
 3   request_datetime       datetime64[us]
 4   on_scene_datetime      datetime64[us]
 5   pickup_datetime        datetime64[us]
 6   dropoff_datetime       datetime64[us]
 7   PULocationID           int64
 8   DOLocationID           int64
 9   trip_miles             float64
 10  trip_time              int64
 11  base_passenger_fare    float64
 12  tolls                  float64
 13  bcf                    float64
 14  sales_tax              float64
 15  congestion_surcharge   float64
 16  airport_fee            float64
 17  tips                   float64
 18  driver_pay             float64
 19  shared_request_flag    object
 20  shared_match_flag      object
 21  access_a_ride_flag     object
 22  wav_request_flag       object
 23  wav_match_flag         object
dtypes: datetime64[us](4), float64(9), int64(3), object(8)
memory usage: 1.1+ GB
```

```
trips_2020_df.describe()
```

	request_datetime	on_scene_datetime	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	trip_miles	trip_time	base_passenger_fa
count	6089998	4362119	6089999	6089999	6.089999e+06	6.089999e+06	6.089999e+06	6.089999e+06	6.089999e+
mean	2020-05-17 07:47:05.142800	2020-05-17 05:01:19.106944	2020-05-17 07:51:44.016943	2020-05-17 08:06:45.809860	1.311385e+02	1.346605e+02	4.716445e+00	9.018230e+02	1.755503e+
min	2020-04-30 22:47:58	2020-04-30 22:55:22	2020-05-01 00:00:00	2020-05-01 00:00:53	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	-1.034800e+
25%	2020-05-09 14:08:02	2020-05-09 11:13:54	2020-05-09 14:11:50.500000	2020-05-09 14:26:25	6.500000e+01	6.800000e+01	1.730000e+00	5.140000e+02	8.850000e+
50%	2020-05-17 14:32:12	2020-05-17 10:17:59	2020-05-17 14:36:36	2020-05-17 14:51:39	1.290000e+02	1.320000e+02	3.120000e+00	7.720000e+02	1.373000e+
75%	2020-05-25 07:05:06.500000	2020-05-25 00:31:59.500000	2020-05-25 07:09:40	2020-05-25 07:23:58	2.030000e+02	2.100000e+02	5.890000e+00	1.139000e+03	2.162000e+
max	2020-06-01 00:10:00	2020-05-31 23:59:48	2020-05-31 23:59:59	2020-06-01 09:39:18	2.650000e+02	2.650000e+02	5.976600e+02	8.580400e+04	1.265570e+
std	NaN	NaN	NaN	NaN	7.682826e+01	7.868463e+01	5.056233e+00	5.739106e+02	1.336668e+

Below are the images of missing values and min-max date_time. there are several missing values in fields like on_scene_datetime, originating_base_num, and airport_fee. Same reasons from before why. The maximum and minimum date in the dataset are provided below in the picture. In this picture, the max request_datetime has value of “2020-06-01 00:10:00” which should really be a part of June and not month.

```
missing_values = trips_2020_df.isnull().sum()
missing_values
```

```
hvfhs_license_num              0
dispatching_base_num           0
originating_base_num     1727879
request_datetime               1
on_scene_datetime        1727880
pickup_datetime                0
dropoff_datetime               0
PULocationID                   0
DOLocationID                   0
trip_miles                     0
trip_time                      0
base_passenger_fare            0
tolls                          0
bcf                            0
sales_tax                      0
congestion_surcharge           0
airport_fee              6089895
tips                           0
driver_pay                     0
shared_request_flag            0
shared_match_flag              0
access_a_ride_flag             0
wav_request_flag               0
wav_match_flag                 0
dtype: int64
```

```
print(min_date)
print(max_date)
```

```
request_datetime    2020-04-30 22:47:58
on_scene_datetime   2020-04-30 22:55:22
pickup_datetime     2020-05-01 00:00:00
dropoff_datetime    2020-05-01 00:00:53
dtype: datetime64[us]
request_datetime    2020-06-01 00:10:00
on_scene_datetime   2020-05-31 23:59:48
pickup_datetime     2020-05-31 23:59:59
dropoff_datetime    2020-06-01 09:39:18
dtype: datetime64[us]
```

After reading and loading the data of 2020 to trips_2020_df, the result of analysis show the following:

There are same number of columns from previous two years' dataset. However, the trips went up again to 14,719,170 which is different than during pandemic time entries.

Here is the summary statistics for trips_2021_df:

```
trips_2021_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14719171 entries, 0 to 14719170
Data columns (total 24 columns):
 #   Column                Dtype
---  ------                -----
 0   hvfhs_license_num     object
 1   dispatching_base_num  object
 2   originating_base_num  object
 3   request_datetime      datetime64[us]
 4   on_scene_datetime     datetime64[us]
 5   pickup_datetime       datetime64[us]
 6   dropoff_datetime      datetime64[us]
 7   PULocationID          int64
 8   DOLocationID          int64
 9   trip_miles            float64
 10  trip_time             int64
 11  base_passenger_fare   float64
 12  tolls                 float64
 13  bcf                   float64
 14  sales_tax             float64
 15  congestion_surcharge  float64
 16  airport_fee           float64
 17  tips                  float64
 18  driver_pay            float64
 19  shared_request_flag   object
 20  shared_match_flag     object
 21  access_a_ride_flag    object
 22  wav_request_flag      object
 23  wav_match_flag        object
dtypes: datetime64[us](4), float64(9), int64(3), object(8)
memory usage: 2.6+ GB
```

```
trips_2021_df.describe()
```

	request_datetime	on_scene_datetime	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	trip_miles	trip_time	base_passenger_fa
count	14719171	10813298	14719171	14719171	1.471917e+07	1.471917e+07	1.471917e+07	1.471917e+07	1.471917e+0
mean	2021-05-16 16:59:28.648864	2021-05-16 16:19:07.072201	2021-05-16 17:05:36.224712	2021-05-16 17:23:50.030519	1.369580e+02	1.407047e+02	4.843593e+00	1.093808e+03	2.264384e+
min	2021-04-30 23:28:12	2021-04-30 23:53:20	2021-05-01 00:00:00	2021-05-01 00:00:38	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	-1.097100e+0
25%	2021-05-08 18:49:14	2021-05-08 19:45:09	2021-05-08 18:55:56	2021-05-08 19:13:48	7.200000e+01	7.400000e+01	1.650000e+00	5.740000e+02	1.174000e+
50%	2021-05-16 16:38:32	2021-05-16 14:25:46	2021-05-16 16:44:58	2021-05-16 17:03:59	1.380000e+02	1.410000e+02	3.014000e+00	8.960000e+02	1.807000e+
75%	2021-05-24 09:01:09	2021-05-24 07:43:11	2021-05-24 09:07:02.500000	2021-05-24 09:25:47	2.100000e+02	2.160000e+02	5.956000e+00	1.387000e+03	2.792000e+
max	2021-06-01 00:00:00	2021-05-31 23:59:55	2021-05-31 23:59:59	2021-06-01 03:18:54	2.650000e+02	2.650000e+02	4.121300e+02	4.905200e+04	1.493980e+0
std	NaN	NaN	NaN	NaN	7.610257e+01	7.840692e+01	5.493553e+00	7.710549e+02	1.731347e+

In May 2021, there were missing fields however surprisingly none of the rides had null value in airport_fee. Finally, the min dates and max dates are below and min. request_datetime, min. on_screen_datetime, max. request_datetime seems incorrect since it can be used different months.

```
missing_values=trips_2021_df.isnull().sum()
missing_values
```

```
hvfhs_license_num             0
dispatching_base_num          0
originating_base_num    3907028
request_datetime              0
on_scene_datetime       3905873
pickup_datetime               0
dropoff_datetime              0
PULocationID                  0
DOLocationID                  0
trip_miles                    0
trip_time                     0
base_passenger_fare           0
tolls                         0
bcf                           0
sales_tax                     0
congestion_surcharge          0
airport_fee                   0
tips                          0
driver_pay                    0
shared_request_flag           0
shared_match_flag             0
access_a_ride_flag            0
wav_request_flag              0
wav_match_flag                0
dtype: int64
```

```
print(min_date)
print(max_date)
```

```
request_datetime    2021-04-30 23:28:12
on_scene_datetime   2021-04-30 23:53:20
pickup_datetime     2021-05-01 00:00:00
dropoff_datetime    2021-05-01 00:00:38
dtype: datetime64[us]
request_datetime    2021-06-01 00:00:00
on_scene_datetime   2021-05-31 23:59:55
pickup_datetime     2021-05-31 23:59:59
dropoff_datetime    2021-06-01 03:18:54
dtype: datetime64[us]
```

Overall, there aren’t many inconsistencies and incorrect data entry. Some needs fixes but at the same time, it is from a reputable source so lets see if I have to fix the issue. Besides that, summary statistics, description, and missing values gives a good idea on what to expect from this dataset.

Feature Engineering and Modeling

Next was the data cleaning and feature engineering step. For the data cleaning step, I decided to use databricks instead of jupyter notebook on EC2 instance. First, I set up the community edition of the databricks, followed by creating a jupyter notebook.
The cleaning was started by viewing the data, counting the rows and dropping the duplicates and comparing the rows. There weren’t any duplicates. Then, I dropped dispatching_base_num and

originating_base_num columns since it is irrelevant to my tip forecasting model. Followed by that, the null values in airport_fees column were corrected. After that shared_request_flag, shared_match_flag, access_a_ride_flag , wav_request_flag, wav_match_flag columns were dropped due to irrelevancy. Below is the code snippet (Appendix C) used to drop columns.

```
#drop dispatch_base_num and originating_base_num
sdf=sdf.drop("dispatching_base_num","originating_base_num") sdf.show(5)

sdf=sdf.drop('shared_request_flag','shared_match_flag','access_a_ride_flag','
wav_request_flag','wav_match_flag')
```

After printing the current schema, I transformed the timestamp object to date_time object while extracting individual parts of the data. After that I added a weekend column to see if a ride was taken during a weekend or not. Below is the code snippet (Appendix C) that I used to create new date_time object columns for pickup_datetime:

```
#transform time data to year,month,date, day of month, day of week, hour,
minute, second sdf=sdf.withColumn("pickup_year",F.year('pickup_datetime'))
sdf=sdf.withColumn("pickup_month",F.month('pickup_datetime'))
sdf=sdf.withColumn("pickup_day_of_month",F.dayofmonth('pickup_datetime'))
sdf=sdf.withColumn("pickup_day_of_week",F.dayofweek('pickup_datetime'))
sdf=sdf.withColumn("pickup_hour",F.hour('pickup_datetime'))
sdf=sdf.withColumn("pickup_minute",F.minute('pickup_datetime'))
sdf=sdf.withColumn("pickup_second",F.second('pickup_datetime'))
sdf.printSchema()
```

Similarly, I transformed the dropoff_datetime as well using similar code. Finally, after that I uploaded all data to s3 bucket in raw folder.

For Feature Engineering part of the assessment, I indexed the only string column in my dataset which was the license number that determines the for-hire vehicle brand using StringIndexer. For doing this, I ran 'hvfhs_license_num' in the StringIndexer, since that was the only string column in the cleaned dataset. For doing this I created a string list called integer_cols to make it efficient. Then, added the 'vector' to the name of the columns. And then called OneHotEncoder(). In this way, I was able to transform all date_time objects that were integer (as a result of my cleaning) to double using encoder. Below is the code snippet (Appendix D) and a picture of this process:

```
integer_cols = [col_name for col_name, col_type in indexed_sdf.dtypes if
col_type == 'int']
integer_cols

out_cols=[name+'Vector' for name in integer_columns] out_cols
```

```
#encode all columns
encoder = OneHotEncoder(inputCols=integer_cols,outputCols=out_cols,
dropLast=False)
encoded_sdf = encoder.fit(indexed_sdf).transform(indexed_sdf)

Encoded_sdf
```

```
▸ encoded_sdf: pyspark.sql.dataframe.DataFrame = [hvfhs_license_num: string, PULocationID: long ... 43 more fields]
Out[92]: DataFrame[hvfhs_license_num: string, PULocationID: bigint, DOLocationID: bigint, trip_miles: double, trip_time: bigint, base_passenger_fare: double, tolls: double, bc
f: double, sales_tax: double, congestion_surcharge: double, airport_fee: int, tips: double, driver_pay: double, pickup_year: int, pickup_month: int, pickup_day_of_month: int,
pickup_day_of_week: int, pickup_hour: int, pickup_minute: int, pickup_second: int, dropoff_year: int, dropoff_month: int, dropoff_day_of_month: int, dropoff_day_of_week: int,
dropoff_hour: int, dropoff_minute: int, dropoff_second: int, pickup_weekend: double, dropoff_weekend: double, licenseIndex: double, airport_feeVector: vector, pickup_yearVecto
r: vector, pickup_monthVector: vector, pickup_day_of_monthVector: vector, pickup_day_of_weekVector: vector, pickup_hourVector: vector, pickup_minuteVector: vector, pickup_seco
ndVector: vector, dropoff_yearVector: vector, dropoff_monthVector: vector, dropoff_day_of_monthVector: vector, dropoff_day_of_weekVector: vector, dropoff_hourVector: vector, d
ropoff_minuteVector: vector, dropoff_secondVector: vector]
Command took 31.50 seconds -- by pujan.patel2@baruchmail.cuny.edu at 11/17/2023, 11:31:25 PM on PP_11/17
```

However, I forgot to remove 'tips' from the double_cols because the goal of the model was to predict tips. After that, I ran all the columns which are now in double data type only excluding 'Tips' in the VectorAssembler(). Later, I followed the steps from the pizza pipeline for tip prediction using linear regression. This included the creation of linear regression on 'tips' column and then regression evaluator that could be used to evaluate the model after. Then the pipeline was created using assembler, encoder, assembler, and linear regression. Here is the code snippet (Appendix D) for linear regression creation and then pipeline.

```
# Create a Linear Regression Estimator

linear_reg = LinearRegression(labelCol='tips')

# Create a regression evaluator (to get RMSE, R2, RME, etc.)
evaluator = RegressionEvaluator(labelCol='tips')
# Create the pipeline Indexer is stage 0 and Linear Regression (linear_reg)
is stage 3
regression_pipe = Pipeline(stages=[indexer, encoder, assembler, linear_reg
```

Then the dataset was already splitted into two parts: training data(0.70), testing data(0.30) in the seed 45 earlier on. The grid was created to hold the hyperparameters and then CrossValidator() function was called using regression_pipe, grid, evaluator, and 3 as numfolds were used respectively. Then the all_models was created using fit() on the training data. Then, I ran the average metrics and received 3.0048.

```
# Show the average performance over the three folds
print(f"Average metric {all_models.avgMetrics}")

Average metric [3.004797063005306]
```

After that, the test_results was creating using the bestModel() function to transform the testing data from the split. Here is the comparison of the 'tips' and 'prediction' in test_results.

```
# Show the predicted tip
test_results.select('tips', 'prediction').show(truncate=False)
```

▸ (1) Spark Jobs

```
+-----+-------------------+
|tips |prediction         |
+-----+-------------------+
|0.0  |0.284086453993483  |
|5.41 |0.9499471432337381 |
|11.81|2.6935281317851683 |
|0.0  |0.0340536303298864 |
|0.0  |0.09358739440505615|
|0.0  |0.22553834486534874|
|0.0  |0.09999110137844514|
|0.0  |0.09402759841413466|
|0.0  |0.1514696859920135 |
|0.0  |0.05334365797109697|
|0.0  |0.22484031202949128|
|0.0  |0.17417348384950737|
|0.0  |0.09108429708166743|
|0.0  |0.2139659362471653 |
|0.0  |0.06873004443725195|
|0.0  |0.10529199976820947|
|0.0  |0.5118792857555439 |
|0.0  |0.06096091945054294|
```

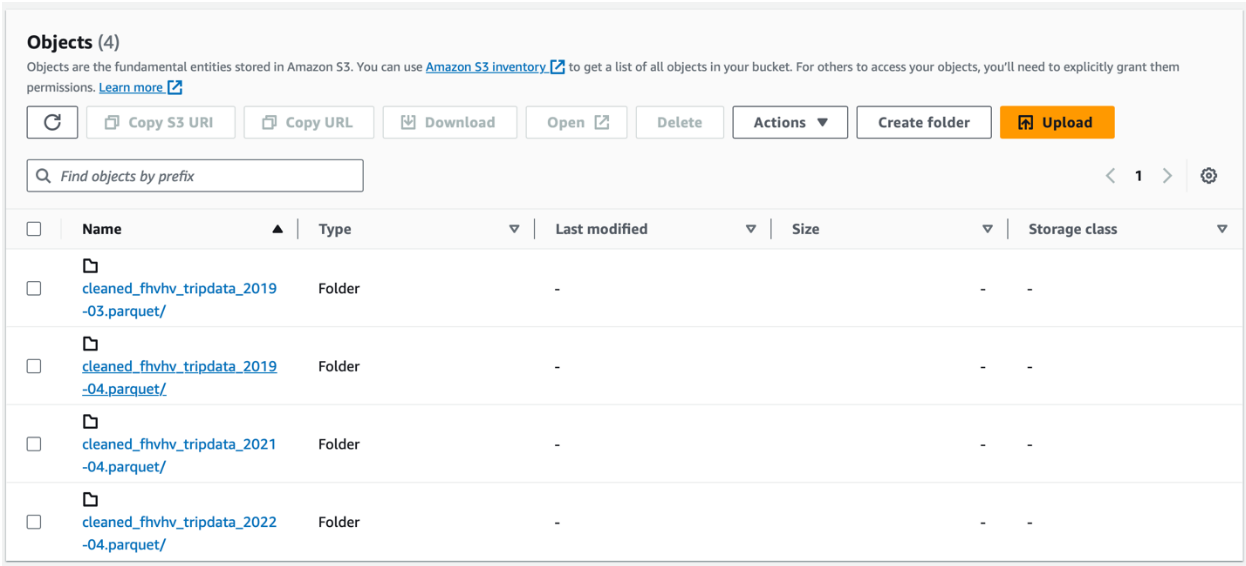
After that, I ran the RMSE and R2 test to look at the margin of error. This indicates that my model was 3 dollars off from actual tips, meaning it is far from accurate. The difference of 3 dollars off from tips makes a big difference hence, why it is not as accurate as it should be. This model needs to be more accurate. Later, I saved the model to S3 bucket /models folder.

```
# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse}  R-squared:{r2}")
```

▸ (2) Spark Jobs

```
RMSE: 3.0025210074411395  R-squared:0.14380613757518212
```

Difficulties faced was deciding on whether to use featureHasher or continue with StringIndexer, Encoder, VectorAssembler. The cleaning part was straightforward however, one of my files was corrupted and I figured that out after few days so time was wasted there. Besides that the cleaning of some files was done, however, the rest of the files is remaining. It should be fairly easy since cleaning is now automated. Here is the snapshot of cleaned dataset in s3 bucket:

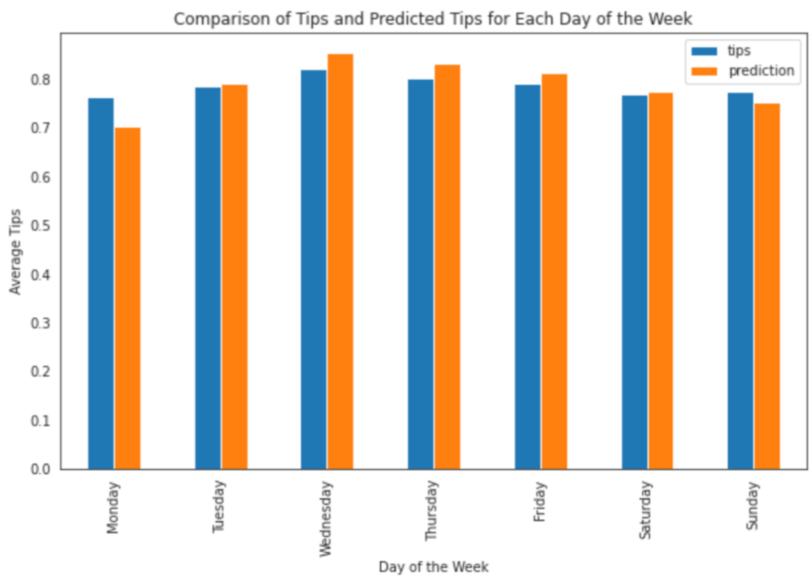


Data Visualizing:

Moving on to the visualization on the metrics, the first visualization I conducted was bar plot comparing the tips and prediction. For this visualization, I first extracted a sdf based on required columns which were ‘tips’, ‘prediction’, ‘base_passenger_fare’, ‘dropoff_day_of_week’ and converted to pandas dataframe. Furthermore, I created a mapping dictionary to convert dropoff_day_of_week from integer value to a categorical value of actual names of days. Here is the code snippet (Appendix D) for mapping:

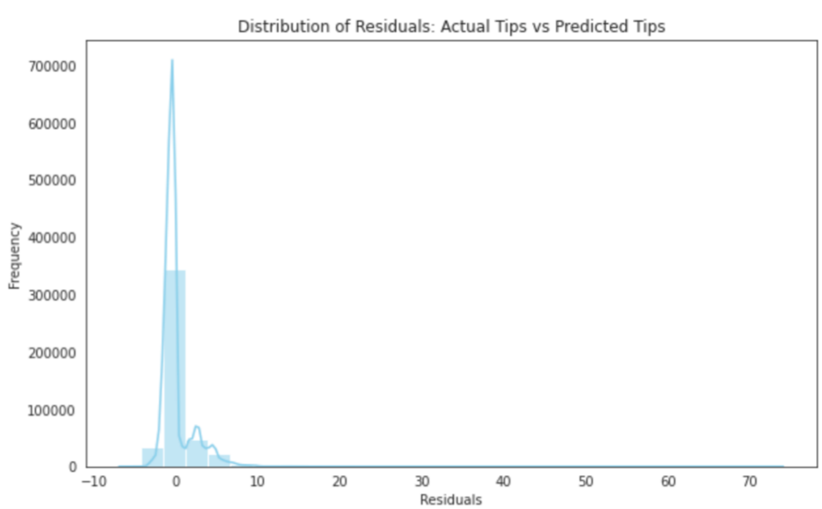
```
# The Spark dataframe test_results holds the original 'tip' as well as the
'prediction'
# Select and convert to a Pandas dataframe
df = test_results.select('tips','prediction',
'base_passenger_fare','dropoff_day_of_week').toPandas()
day_mapping = {1: 'Monday', 2: 'Tuesday', 3: 'Wednesday', 4: 'Thursday', 5:
'Friday', 6: 'Saturday', 7: 'Sunday'}
df['dropoff_day_of_week'] = df['dropoff_day_of_week'].map(day_mapping)
# Make sure the days column is a categorical variable for proper ordering on
the x-axis
df['dropoff_day_of_week'] = pd.Categorical(df['dropoff_day_of_week'],
categories=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday'], ordered=True)
df
```

I created a relationship plot and noticed some outliers so I decided to remove them. Then, I filtered the data to remove the outliers which were fares greater than 40 and tips greater than 100. Then I grouped the data frame based on week day and then created a bar plot shown below:



The graph indicates that the predicted tips and actual tips are very close on Tuesday and Saturday, other days the difference is bigger. This shows that model's prediction is inconsistent with overpredicting three times and underpredicting two times.

Then I created a sample of data from test_results so the commands run faster. Filtered the data again and then I created residuals column in the filtered data frame by subtracting prediction from tips and then create a histogram to visualize the spread. Below is the pic of histogram:



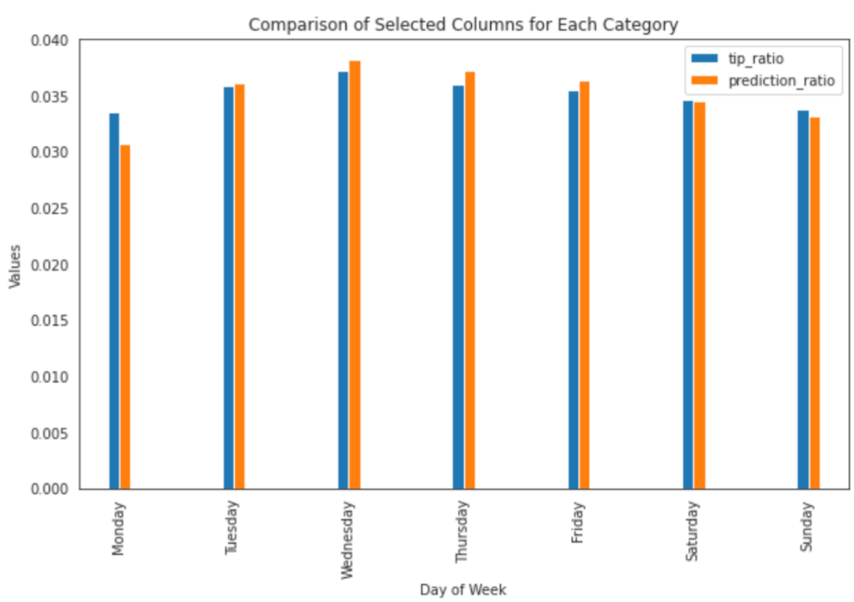
The graph indicates that the distribution of residuals is mostly near 0 which shows that on average the model's prediction and actual tips are accurate and there is no systematic bias in the predictions.

For my third visualization, I wanted to compare the tip ratio and prediction ration over day of week. In other words, I wanted to see how much percent of trip fare a rider pays in tips and compare that with prediction from model. To achieve this I had to create a sample data frame that consists of all factors in fares, tips, predictions, and dropoff_day_of_week. Below is the code snippet (Appendix D) that I ran:

```
check['total']=check[['base_passenger_fare','tolls','bcf','sales_tax','conges
tion_surcharge','airport_fee']].sum(axis=1)
check
```

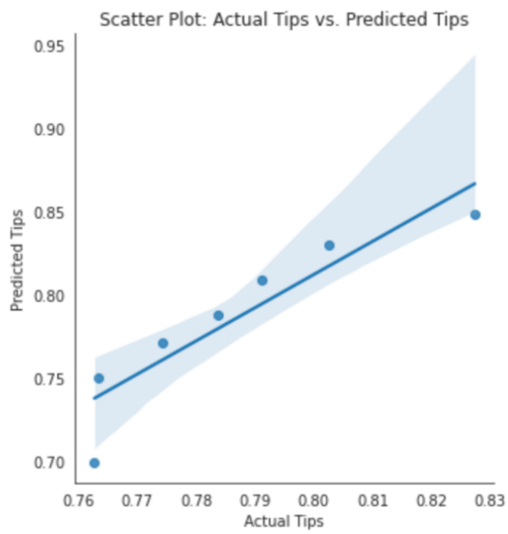
Then, I grouped the data frame on day of week followed by re-indexing to make sure it is in order. Below are the images for the dataframe that is ready to be plotted and a bar plot to visualize the comparison.

day_name	base_passenger_fare	tolls	bcf	sales_tax	congestion_surcharge	airport_fee	driver_pay	tips	prediction	dropoff_day_of_week	total	tip_ratio	prediction_ratio
Monday	18.744298	0.556865	0.584578	1.686289	1.060159	0.095637	15.432327	0.762679	0.699472	1.0	22.727827	0.033557	0.030776
Tuesday	17.965537	0.486552	0.557731	1.617797	1.088052	0.114158	14.497510	0.783637	0.787883	2.0	21.829827	0.035898	0.036092
Wednesday	18.315438	0.451344	0.566490	1.641175	1.136647	0.093046	14.771992	0.827363	0.848148	3.0	22.204140	0.037262	0.038198
Thursday	18.397702	0.441799	0.568750	1.647173	1.126928	0.086545	14.885227	0.802516	0.829728	4.0	22.268896	0.036038	0.037260
Friday	18.381627	0.430918	0.567674	1.644547	1.116889	0.089444	14.971118	0.791051	0.808566	5.0	22.231099	0.035583	0.036371
Saturday	18.499988	0.436278	0.571307	1.657097	1.059565	0.088681	15.079466	0.774337	0.771503	6.0	22.312916	0.034704	0.034577
Sunday	18.679833	0.501979	0.578124	1.671032	1.108293	0.063587	15.347511	0.763253	0.750603	7.0	22.602848	0.033768	0.033208



The graph indicates that tip ratio and prediction ratio over days of week are pretty much similar to the first visualization. It can be inferred that monday is the worst day for making tips, whereas Wednesday is the best day to make tips.

Finally for the last visualization, I decided to do a scatter plot to see the spread of the data points. I used the same sample and filtered data frame to create the visualization below. The visualization is below:



This graph indicates that there is a positive linear relation between the variables: predicted tips and actual tips. It also shows that majority of the data is within the shaded region with one outside and one on the tip. This visualization also indicates that model is performing well.

Challenges faced during the visualization was the outliers which skewed the graph on the first attempt, but after professor's suggestion, I was able to filter out the data frame during visualization to better understand the graphs.

Summary

To summarize the project, I decided to use the For-Hire vehicle rides data recorded in New York City from Kaggle and the actual source to create a machine learning pipeline that incorporate big data pipeline. The dataset included information regarding a taxi ride which could be exploited to create a tip prediction model. Technologies used throughout the project was AWS (EC2, S3), Databricks community edition to use jupyter notebook. To start off the project, I extracted the data from the source and downloaded to AWS EC2 instance and then transported to AWS S3 bucket. During this process, I ran into a 404 error when trying to download from Kaggle, so I used Curl and downloaded directly from the source. After that, I conducted exploratory data analysis where I understood the data and what are its types and how can I take advantage of the data. The challenge during that process was reading the data into a data frame since, the size of the dataset was gigantic. By switching over to t2.xl instance of EC2 I was able to finish it. Later, I moved to data bricks to take advantage of clusters. After that I performed data cleaning and feature engineering on dataset using data bricks and PySpark. The challenges as this stage was debating on what to use for feature engineering and using 'tips' as part of my features to create the most accurate model ever. After updating the features list, I decided to use linear regression combined with StringIndexer, OneHotEncoder, and VectorAssembler to create a model. The model was created and evaluated it was sort of accurate.

Followed by that was the data visualization where I used seaborn, matplotlib, and pandas libraries to create some visualizations. The visualizations compared the tips vs prediction which resulted in a well accurate model. Then, histogram and scatter plot suggested that most of the data points were in a good range and on-average the model is accurate. The challenges faced during this part is the outliers which skewed the visualization a bit, so I filtered the data and then performed the visualization. In conclusion, a well accurate model was created through machine learning pipeline that incorporate big data pipeline. The biggest takeaway was learning the machine learning pipeline and how to utilize Amazon Web Services as a resource for the pipeline.

Appendix A:

```
#installed kaggle using the commands from class pip3 install kaggle
mkdir .kaggle
nano .kaggle/kaggle.json

#entered my Kaggle API Key and saved

#secured the folder
chmod 600 .kaggle/kaggle.json

#check

kaggle datasets list

#API Command: kaggle datasets download -d jeffsinsel/nyc-fhvhv-data

#Created S3 bucket and folders

#checked the files from the target dataset
kaggle datasets files -d jeffsinsel/nyc-fhvhv-data

#downloaded file
kaggle datasets download -d jeffsinsel/nyc-fhvhv-data - f
fhvhv_tripdata_2019-02.parquet

#unzip
unzip fhvhv_tripdata_2019-02.parquet

#copying to s3
aws s3 cp fhvhv_tripdata_2019-02.parquet s3://pp-nyc- trips-
data/landing/fhvhv_tripdata_2019-02.parquet

#checking if its visible in s3
aws s3 ls s3://pp-nyc-trips-data/landing/

#removed the files from the local system rm fhvhv_tripdata_2019-
02.parquet.zip
rm fhvhv_tripdata_2019-02.parquet

#repeating steps for all //only going to write description for 1 and last
#1
#downloading other files starting from top
kaggle datasets download -d jeffsinsel/nyc-fhvhv-data - f
fhvhv_tripdata_2020-02.parquet

#unzip downloaded file
unzip fhvhv_tripdata_2020-02.parquet

#copying to s3
aws s3 cp fhvhv_tripdata_2020-02.parquet s3://pp-nyc- trips-
data/landing/fhvhv_tripdata_2020-02.parquet

#checking if its visible in s3
aws s3 ls s3://pp-nyc-trips-data/landing/
```

```
#yes

#removed the files from the local system rm fhvhv_tripdata_2020-
02.parquet.zip
rm fhvhv_tripdata_2020-02.parquet

. . .

#last file downloaded was
curl -L -o taxi_zones.shx https://d37ci6vzurychx.cloudfront.net/trip-
data/taxi_zones/taxi_zones.shx

#copying to s3
aws s3 cp taxi_zones.shx s3://pp-nyc-trips- data/landing/wtaxi_zones.shx

#checking if its visible in s3
aws s3 ls s3://pp-nyc-trips-data/landing/

#removed the files from the local system rm taxi_zones.shx
```

Appendix B:

```
#import statements
import pandas as pd
Source Code for EDA
# !pip install pyarrow to install parquet package from pyarrow.parquet import
ParquetFile
import pyarrow as pa

#load data to pandas from s3 trips_2019_df=pd.read_parquet("s3://pp-nyc-
trips-data/landing/fhvhv_tripdata_2019-05.parquet")

#view dataframe
trips_2019_df
#get info on dataframe trips_2019_df.info()

#summary statistics trips_2019_df.describe()

#missing values calculation
missing_values = trips_2019_df.isnull().sum() missing_values

#min and max dates for all date columns in the dataframe
min_date =
trips_2019_df[['request_datetime','on_scene_datetime','pickup_datetime','drop
off_datetime']].min()

max_date =
trips_2019_df[['request_datetime','on_scene_datetime','pickup_datetime','drop
off_datetime']].max()
print(min_date)
print(max_date)

#2020_05 file data analysis trips_2020_df=pd.read_parquet("s3://pp-nyc-trips-
data/landing/fhvhv_tripdata_2020-05.parquet")
```

```
#view datasets
trips_2020_df
#get info on 2020_05 file trips_2020_df.info()

#perform summary statistic trips_2020_df.describe()

#missing values calculation
missing_values = trips_2020_df.isnull().sum() missing_values

#min and max date calculation
min_date =
trips_2020_df[['request_datetime','on_scene_datetime','pickup_datetime','drop
off_datetime']].min()

max_date =
trips_2020_df[['request_datetime','on_scene_datetime','pickup_datetime','drop
off_datetime']].max()

print(min_date)
print(max_date)
#2021_05 file data analysis trips_2021_df=pd.read_parquet("s3://pp-nyc-trips-
data/landing/fhvhv_tripdata_2021-05.parquet")

#view dataset
trips_2021_df
#get information on the file trips_2021_df.info()

#perform summary statistic trips_2021_df.describe()

#calculate missing values missing_values=trips_2021_df.isnull().sum()
missing_values

#find min and max dates for dates from dataset
min_date =
trips_2021_df[['request_datetime','on_scene_datetime','pickup_datetime','drop
off_datetime']].min()

max_date =
trips_2021_df[['request_datetime','on_scene_datetime','pickup_datetime','drop
off_datetime']].max()

print(min_date)
print(max_date)
```

Appendix C:

```
# Databricks notebook source
spark

# COMMAND ----------

import pandas as pd
import seaborn as sns
```

```
import matplotlib as mpl
import sklearn
import numpy
import scipy
import plotly
import bs4 as bs
import urllib.request
import boto3
import pyspark.sql.functions as F

# COMMAND ----------
import os
from pyspark.sql.functions import col, isnull, when, count, udf

# COMMAND ----------

# To work with Amazon S3 storage, set the following variables using your AWS
Access Key and Secret Key

# Set the Region to where your files are stored in S3.

access_key = 'xyz'

secret_key = 'xyz'

# COMMAND ----------
# Set the environment variables so boto3 can pick them up later
os.environ['AWS_ACCESS_KEY_ID'] = access_key
os.environ['AWS_SECRET_ACCESS_KEY'] = secret_key
encoded_secret_key = secret_key.replace("/", "%2F")
aws_region = "us-east-2"
bucket_name = "pp-nyc-trips-data"

# COMMAND ----------

# Update the Spark options to work with our AWS Credentials
sc._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3." + aws_region +
".amazonaws.com")
sc._jsc.hadoopConfiguration().set("fs.s3a.bucket."+bucket_name
+".endpoint.region", aws_region)
# COMMAND ----------

filename="landing/fhvhv_tripdata_2019-05.parquet"

file_path = 's3a://' + bucket_name +"/" + filename

sdf = spark.read.parquet(file_path)

# COMMAND ----------

sdf.head(10)

# COMMAND ----------
```

```
sdf.printSchema()

# COMMAND ----------

sdf.show()

# COMMAND ----------

sdf.count()

# COMMAND ----------

sdf.dropDuplicates()

sdf.count()

# COMMAND ----------

#samples=sdf.sample(False,0.25)

#samples.summary().show()

# COMMAND ----------

#samples.select([count(when(isnull(c), c)).alias(c) for c in
samples.columns]).show()

# COMMAND ----------

#drop dispatch_base_num and originating_base_num

sdf=sdf.drop("dispatching_base_num","originating_base_num")

sdf.show(5)

# COMMAND ----------

sdf.select('airport_fee').show()

# COMMAND ----------

#changing null values to 0 in airport fee

sdf = sdf.withColumn('airport_fee', when(sdf['airport_fee'].isNull(),
0).otherwise(sdf['airport_fee']))

sdf.select('airport_fee').show(10)

# COMMAND ----------

#check summary

sdf.summary()

# COMMAND ----------
```

```
sdf.select([count(when(isnull(c), c)).alias(c) for c in sdf.columns]).show()

# COMMAND ----------

sdf.select('wav_match_flag').show()

# COMMAND ----------

sdf=sdf.drop('shared_request_flag','shared_match_flag','access_a_ride_flag','
wav_request_flag','wav_match_flag')

# COMMAND ----------

sdf.printSchema()

# COMMAND ----------

#transform time data to year,month,date, day of month, day of week, hour,
minute, second

sdf=sdf.withColumn("pickup_year",F.year('pickup_datetime'))

sdf=sdf.withColumn("pickup_month",F.month('pickup_datetime'))

sdf=sdf.withColumn("pickup_day_of_month",F.dayofmonth('pickup_datetime'))

sdf=sdf.withColumn("pickup_day_of_week",F.dayofweek('pickup_datetime'))

sdf=sdf.withColumn("pickup_hour",F.hour('pickup_datetime'))

sdf=sdf.withColumn("pickup_minute",F.minute('pickup_datetime'))

sdf=sdf.withColumn("pickup_second",F.second('pickup_datetime'))

sdf.printSchema()

# COMMAND ----------

#transform time data to year,month,date, day of month, day of week, hour,
minute, second

sdf=sdf.withColumn("dropoff_year",F.year('dropoff_datetime'))

sdf=sdf.withColumn("dropoff_month",F.month('dropoff_datetime'))

sdf=sdf.withColumn("dropoff_day_of_month",F.dayofmonth('dropoff_datetime'))

sdf=sdf.withColumn("dropoff_day_of_week",F.dayofweek('dropoff_datetime'))

sdf=sdf.withColumn("dropoff_hour",F.hour('dropoff_datetime'))

sdf=sdf.withColumn("dropoff_minute",F.minute('dropoff_datetime'))

sdf=sdf.withColumn("dropoff_second",F.second('dropoff_datetime'))
```

```
sdf.printSchema()

# COMMAND ----------

sdf.show(10)

# COMMAND ----------

sdf=sdf.drop('request_datetime','on_scene_datetime','dropoff_datetime','picku
p_datetime')

# COMMAND ----------

#add pickup_weekend

sdf=sdf.withColumn("pickup_weekend",
when(sdf.pickup_day_of_week==6,1.0).when(sdf.pickup_day_of_week==7,1.0).other
wise(0))

#add dropoff_weekend

sdf=sdf.withColumn("dropoff_weekend",
when(sdf.dropoff_day_of_week==6,1.0).when(sdf.dropoff_day_of_week==7,1.0).oth
erwise(0))

# COMMAND ----------

sdf.show(5)

# COMMAND ----------

index=filename.index('/')+1

filename = 'cleaned_'+filename[index:]

filename

# COMMAND ----------

output_file_path="s3://pp-nyc-trips-dats/raw/"+filename

output_file_path

# COMMAND ----------

sdf = sdf.repartition(1)

sdf.write.parquet(output_file_path)

# COMMAND ----------

len(sdf.columns)
```

Appendix D:

```
# Databricks notebook source
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import sklearn
import numpy
import scipy
import plotly
import bs4 as bs
import urllib.request
import boto3
import pyspark.sql.functions as F
import os
from pyspark.sql.functions import col, isnull, when, count, udf
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
# Import the evaluation module
from pyspark.ml.evaluation import *
# Import the model tuning module
from pyspark.ml.tuning import *



# COMMAND ----------

spark

# COMMAND ----------

# To work with Amazon S3 storage, set the following variables using your AWS
Access Key and Secret Key
# Set the Region to where your files are stored in S3.
access_key = 'xyz'
secret_key = 'xyz'

# COMMAND ----------

# Set the environment variables so boto3 can pick them up later
os.environ['AWS_ACCESS_KEY_ID'] = access_key
os.environ['AWS_SECRET_ACCESS_KEY'] = secret_key
encoded_secret_key = secret_key.replace("/", "%2F")
aws_region = "us-east-2"
bucket_name = "pp-nyc-trips-data"

# COMMAND ----------

# Update the Spark options to work with our AWS Credentials
sc._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3." + aws_region +
".amazonaws.com")
sc._jsc.hadoopConfiguration().set("fs.s3a.bucket."+bucket_name
+".endpoint.region", aws_region)
```

```
# COMMAND ----------

filename="raw/cleaned_fhvhv_tripdata_2021-05.parquet"

file_path = 's3a://' + bucket_name +"/" + filename

sdf = spark.read.parquet(file_path)

# COMMAND ----------

sdf.count()

# COMMAND ----------

len(sdf.columns)

# COMMAND ----------

training,test=sdf.randomSplit([0.70,0.30], seed=45)

# COMMAND ----------

indexer =
StringIndexer(inputCol='hvfhs_license_num',outputCol='licenseIndex',
handleInvalid="keep")

indexed_sdf=indexer.fit(sdf).transform(sdf)



# COMMAND ----------

integer_cols = [col_name for col_name, col_type in indexed_sdf.dtypes if
col_type == 'int']

integer_cols

# COMMAND ----------

double_cols = [col_name for col_name,col_type in indexed_sdf.dtypes if
(col_type == 'double' and col_name != 'tips')]

double_cols

# COMMAND ----------

out_cols=[name+'Vector' for name in integer_cols]

out_cols

# COMMAND ----------

#encode all columns
```

```
encoder = OneHotEncoder(inputCols=integer_cols,outputCols=out_cols,
dropLast=True,handleInvalid="keep")

encoded_sdf = encoder.fit(indexed_sdf).transform(indexed_sdf)

encoded_sdf

# COMMAND ----------

input_cols=out_cols+double_cols

input_cols

# COMMAND ----------

assembler = VectorAssembler(inputCols=input_cols, outputCol="features")

# COMMAND ----------

# Create a Linear Regression Estimator

linear_reg = LinearRegression(labelCol='tips')

# Create a regression evaluator (to get RMSE, R2, RME, etc.)

evaluator = RegressionEvaluator(labelCol='tips')

# COMMAND ----------

# Create the pipeline   Indexer is stage 0 and Linear Regression (linear_reg)
is stage 3

regression_pipe = Pipeline(stages=[indexer, encoder, assembler, linear_reg])

# COMMAND ----------

# Create a grid to hold hyperparameters

grid = ParamGridBuilder()

# Build the parameter grid

grid = grid.build()

# COMMAND ----------



# Create the CrossValidator using the hyperparameter grid

cv = CrossValidator(estimator=regression_pipe,

                    estimatorParamMaps=grid,

                    evaluator=evaluator,
```

```
                    numFolds=3)

# COMMAND ----------
# Train the models
all_models  = cv.fit(training)
# COMMAND ----------
# Show the average performance over the three folds
print(f"Average metric {all_models.avgMetrics}")
# COMMAND ----------
# Get the best model from all of the models trained
bestModel = all_models.bestModel
# COMMAND ----------
# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(test)
# COMMAND ----------
# Show the predicted tip
test_results.select('tips', 'prediction').show(truncate=False)
# COMMAND ----------
# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse}  R-squared:{r2}")
# COMMAND ----------
model_name="tip_prediction_model_"+filename[-13:-8]
model_path="s3://pp-nyc-trips-data/models/"+model_name
model_path
# COMMAND ----------
# Save the model to S3
```

```
bestModel.save(model_path)

# COMMAND ----------

test_results

# COMMAND ----------

sdf = test_results.repartition(1)

sdf

# COMMAND ----------

output_file_path='s3://pp-nyc-trips-data/trusted/test_results_'+filename[-
13:-8]

output_file_path

# COMMAND ----------

sdf.write.parquet(output_file_path)

# COMMAND ----------

import matplotlib.pyplot as plt

# COMMAND ----------

# The Spark dataframe test_results holds the original 'tip' as well as the
'prediction'

# Select and convert to a Pandas dataframe

df =
test_results.select('tips','prediction','base_passenger_fare','dropoff_day_of
_week').toPandas()

df

# COMMAND ----------

day_mapping = {1: 'Monday', 2: 'Tuesday', 3: 'Wednesday', 4: 'Thursday', 5:
'Friday', 6: 'Saturday', 7: 'Sunday'}

df['dropoff_day_of_week'] = df['dropoff_day_of_week'].map(day_mapping)

df

# COMMAND ----------

# Make sure the days column is a categorical variable for proper ordering on
the x-axis
```

```
df['dropoff_day_of_week'] = pd.Categorical(df['dropoff_day_of_week'],
categories=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday'], ordered=True)

df

# COMMAND ----------

df_filtered = df[(df['base_passenger_fare'] <= 40) & (df['tips'] <= 100)]

df_filtered

# COMMAND ----------

grouped_df = df_filtered.groupby('dropoff_day_of_week')[['tips',
'prediction']].mean()

# COMMAND ----------

grouped_df

# COMMAND ----------

ax = grouped_df.plot(kind='bar', figsize=(10, 6))

ax.set_xlabel('Day of the Week')

ax.set_ylabel('Average Tips')

ax.set_title('Comparison of Tips and Predicted Tips for Each Day of the
Week')

plt.show()

# COMMAND ----------

# Plot a line chart

ax = grouped_df[['tips', 'prediction']].plot(kind='line', marker='o',
figsize=(10, 6))

# Add labels and title

plt.xlabel('Day of the Week')

plt.ylabel('Tips')

plt.title('Comparison of Tips and Predicted Tips for Each Day of the Week')

# Show the legend

plt.legend(["Actual Tips", "Predicted Tips"])

# Show the plot
```

```
plt.show()

# COMMAND ----------

sample, big = test_results.randomSplit([0.1,0.9], seed=42)

# COMMAND ----------

residual_df=sample.select('tips','prediction','base_passenger_fare').toPandas
()

residual_df

# COMMAND ----------

residual_df=residual_df[(residual_df['base_passenger_fare'] <= 40) &
(residual_df['tips'] <= 100)]

residual_df

# COMMAND ----------

# Calculate residuals

residual_df['residuals'] = residual_df['tips'] - residual_df['prediction']

# Create a residual plot

plt.figure(figsize=(10, 6))

sns.histplot(residual_df['residuals'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of Residuals: Actual Tips vs Predicted Tips')

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.show()

# COMMAND ----------

fares_cols=['base_passenger_fare','tolls','bcf','sales_tax','congestion_surch
arge','airport_fee','driver_pay','tips','prediction','dropoff_day_of_week']

# COMMAND ----------

sample, big = test_results.randomSplit([0.1,0.9], seed=42)

# COMMAND ----------

df=sample.select(fares_cols).toPandas()

# COMMAND ----------
```

```
check=df[(df['base_passenger_fare'] <= 40) & (df['tips'] <= 100)]

# COMMAND ----------

check['total']=check[['base_passenger_fare','tolls','bcf','sales_tax','conges
tion_surcharge','airport_fee']].sum(axis=1)

check

# COMMAND ----------

# Map integer values to day names

check['day_name'] = check['dropoff_day_of_week'].map({1: 'Monday', 2:
'Tuesday', 3: 'Wednesday', 4: 'Thursday', 5: 'Friday', 6: 'Saturday', 7:
'Sunday'})

# COMMAND ----------

check = check.groupby('day_name').mean()

# COMMAND ----------

check['tip_ratio']=check['tips']/check['total']

check['prediction_ratio']=check['prediction']/check['total']

# COMMAND ----------

check

# COMMAND ----------

grouped_df = check.groupby('day_name').mean().reindex(['Monday', 'Tuesday',
'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

grouped_df

# COMMAND ----------

# Plot grouped bar chart for selected columns

ax = grouped_df[['tip_ratio','prediction_ratio']].plot(kind='bar',
figsize=(10, 6), width=0.2)

# Add labels and title

plt.xlabel('Day of Week')

plt.ylabel('Values')

plt.title('Comparison of Selected Columns for Each Category')

# Show the legend
```

```
plt.legend(['tip_ratio','prediction_ratio'])

# Show the plot

plt.show()

# COMMAND ----------

# Assuming df is your DataFrame with columns 'tip' and 'prediction'

sns.lmplot(x='tips', y='prediction', data=check)

plt.xlabel('Actual Tips')

plt.ylabel('Predicted Tips')

plt.title('Scatter Plot: Actual Tips vs. Predicted Tips')

plt.show()
```