Big Data Technologies

NYC Yellow Taxi Trips Data with AWS

By SHARIA HOQUE: SHARIA.HOQUE@baruchmail.cuny.edu

Professor Richard Holowczak

Table of Contents

Milestone 1: Proposal

The **new_york_taxi_trips** dataset is collected from Google Public Datasets.

I intend to use **5 tables** out of 27 tables; one taxi zone dictionary and 4 tables of yellow taxi trip records from 2019-2022 which have around total **35 GB** of data in big-query.

The new_york_taxi_trips dataset's six tables contain the following attributes:

- *1 table of taxi zone dictionary:* **4 attributes**
    - zone_id, zone_name, borough and zone_geom
- *5 tables of Yellow taxi trip records from 2019-2022*: **23 attributes**
    - vendor_id, pickup_datetime, dropoff_datetime, store_and_fwd_flag, rate_code, passenger_count, trip_distance, fare_amount, extra, mta_tax, tip_amount, tolls_amount, ehail_fee, airport_fee, total_amount, payment_type, distance_between_service, time_between_service, trip_type, imp_surcharge, pickup_location_id, dropoff_location_id, data_file_year, data_file_month.

From these yellow taxi tables, I like to predict.

- The **time length of trips** in other word, the time difference between pickup-drop-off time based on the trip distance, year, location, total-cost, air-port fee, pament_type.

Milestone 2: Data Acquisition

The steps for extracting and collecting Big Query Data to Amazon s3 are below:

Collecting Big Query Data:

1. Big Query is a Google's serverless data warehouse. Under Big Query, **a project, cis-4130-project, is created** with the target 5 tables (translation table, 2019-2022 yellow taxi datasets) from the big query's public NY taxi datasets.
2. Cloud storage is a service for storing objects in Google Cloud. Therefore, to store the 8 tables from the project, a cloud storage **bucket is created.**
3. Before transferring to the bucket, if any table has more than 1 GB, I **divided each table by monthly** by using the following code in appendix 1 in Cloud shell (Google Cloud Terminal). (Reason: if any file is more than 1 GB, google cloud does not allow to store in Bucket)
    - After dividing one table into 12 table, if any table still has more than 1 GB, that table is divided by manually using SQL queries.
4. There are two ways the **tables are transferred into the Bucket**:
    a. Manually
    b. Using the following script in appendix 1, Cloud shell (Google Cloud Terminal):

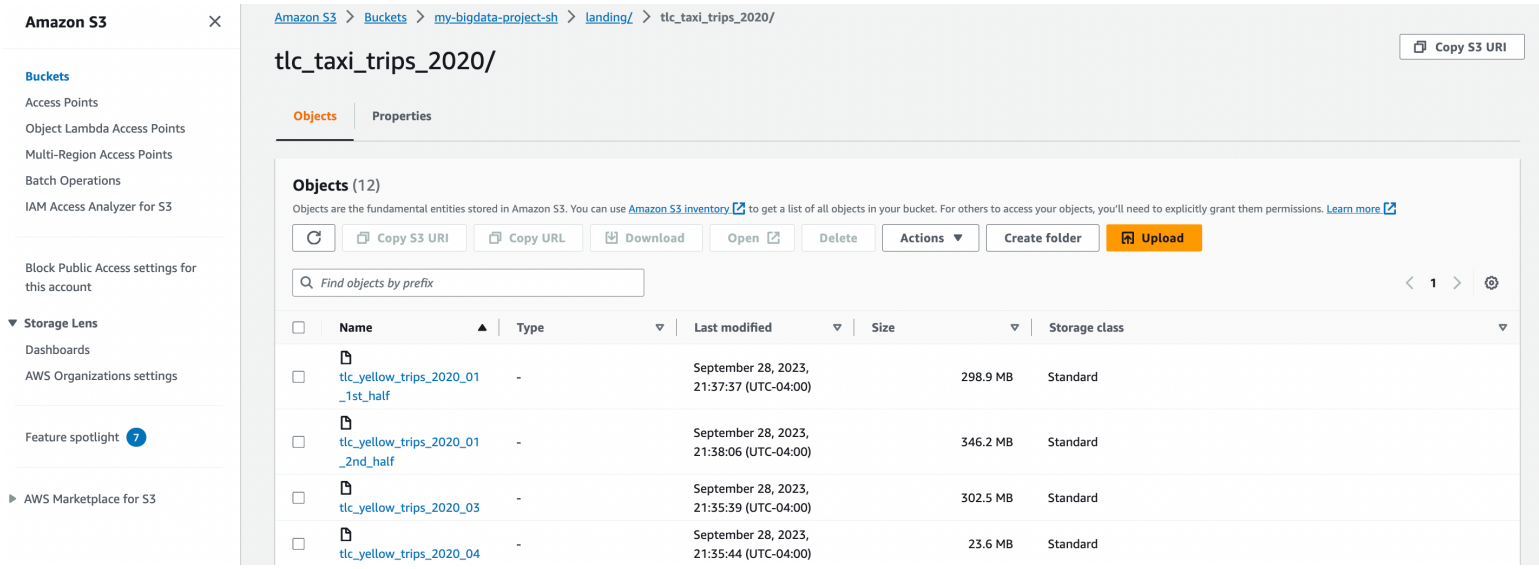
Transferring to AWS S3

5. One **Instance** was created in AWS EC2 under IAM user:
    a. Region: us-east-2
    b. Platform: Linux/UNIX
    c. Instance type: t2.micro
    d. Volume: 30GB
6. **Bucket**, "my-bigdata-project-sh" is created to store tables.
    a. Under my-bigdata-project-sh folder, landing, raw, trusted, curated and models folders are created
7. After creating instances, bucket, and downloading requirement packages in AWS CLI, I **downloaded gcloud** on Amazon Linux by following codes in appendix 1:

Note: More description in this link: https://saturncloud.io/blog/how-to-install-gcloud-on-amazon-linux-without-invalid-syntax-error/

8. Then **copy the files/table**s from Gcloud bucket:
9. Finally **upload the copies** file into AWS S3 bucket using AWS CLI:
10. Finally moved the uploaded files into the landing folder (located in AWS S3 Bucket)

Sample of 2020 taxi dataset in AWS S3 landing folder:



Update on Milestone 2:

I used Curl function to download as parquet files or the year 2018-2022 from the Taxi website and saved in AWS S3. (Appendix 1)

Milestone 3: Exploratory Data Analysis

This EDA is based on the **January 2022 yellow** taxi trips data. The below EDA is created by using jupyter Notebook in Databrick where the dataset is extracted from AWS S3 landing.

- It has total 2463900 records and 20 attributes.

#	Column	Dtype
0	vendor_id	int64
1	pickup_datetime	object
2	dropoff_datetime	object
3	passenger_count	float64
4	trip_distance	float64
5	rate_code	float64
6	store_and_fwd_flag	object
7	payment_type	int64
8	fare_amount	float64
9	extra	float64
10	mta_tax	float64
11	tip_amount	float64
12	tolls_amount	float64
13	imp_surcharge	float64
14	airport_fee	float64
15	total_amount	float64
16	pickup_location_id	int64
17	dropoff_location_id	int64
18	data_file_year	int64
19	data_file_month	`

#	Column	Dtype
0	vendor_id	int64
1	pickup_datetime	datetime64[ns, UTC]
2	dropoff_datetime	datetime64[ns, UTC]
3	passenger_count	float64
4	trip_distance	float64
5	rate_code	float64
6	store_and_fwd_flag	object
7	payment_type	int64
8	fare_amount	float64
9	extra	float64
10	mta_tax	float64
11	tip_amount	float64
12	tolls_amount	float64
13	imp_surcharge	float64
14	airport_fee	float64
15	total_amount	float64
16	pickup_location_id	int64
17	dropoff_location_id	int64
18	data_file_year	int64
19	data_file_month	int64
20	pickup_date	datetime64[ns]
21	pickup_time	object
22	dropoff_date	datetime64[ns]
23	dropoff_time	object

- This image above shows the datatypes for 20 attributes.
    - The pickup and drop-off time are both objects. In this case, those two fields are converted into datetime64 datatypes and divided into four field by separating date and time.
- There are no null and duplicates records.
- There are types of fields with IDs which require translation for analyzes purposes.
    - four types of vendor_id: The translations for this field will be added manually
        - 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc; 5 = unknown, 6 = unkonwn
    - 6 types of payment type: The translations for this field will be added manually
        - 0= Unknown; 1= Credit card; 2= Cash; 3= No charge; 4= Dispute; 5= Unknown; 6= Voided trip
    - Rate code: The translations for this field will be added manually

        - 1= Standard rate; 2=JFK; 3=Newark ; 4=Nassau ;Westchester 5=Negotiated fare; 6=Group ride; 99 = Unknown
    - 262 types of Pick-up location id: Zone dictionary needs to be added to get the borough and zone names.
    - 262 types of drop-off location id: Zone dictionary needs to be added to get the borough and zone names.

Note: The more translation for this dataset is in the below link:
https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf

	index	passenger_count	trip_distance	rate_code	fare_amount	extra	mta_tax	tip_amount	tolls_amount	imp_surcharge	airport_fee	total_amount
0	count	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06	2.392397e+06
1	min	0.000000e+00	0.000000e+00	1.000000e+00	-4.800000e+02	-4.500000e+00	-5.000000e-01	-1.252200e+02	-3.140000e+01	-3.000000e-01	-1.250000e+00	-4.803000e+02
2	max	9.000000e+00	6.510000e+02	9.900000e+01	4.010923e+05	3.350000e+01	1.659000e+01	8.888800e+02	1.933000e+02	3.000000e-01	1.250000e+00	4.010956e+05
3	mean	1.389436e+00	3.099714e+00	1.415511e+00	1.280726e+01	1.034310e+00	4.913714e-01	2.368710e+00	3.748254e-01	2.966535e-01	8.250042e-02	1.902459e+01
4	median	1.000000e+00	1.710000e+00	1.000000e+00	9.000000e+00	5.000000e-01	5.000000e-01	2.000000e+00	0.000000e+00	3.000000e-01	0.000000e+00	1.430000e+01

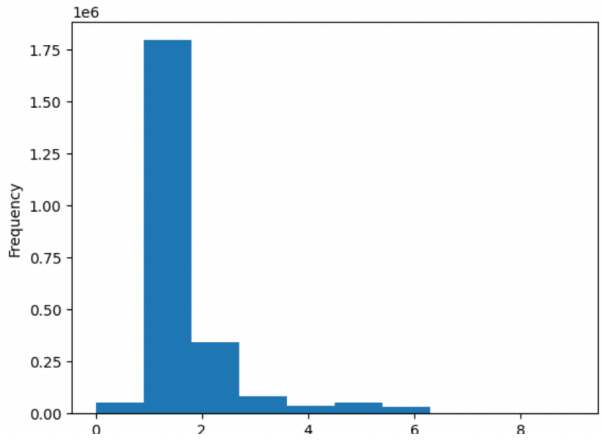
- The above shows the count, min, max, mean, median for different fields. The different types of payment have negative values for minimum values which are outliers.

	Weeks	Total Count	Min_total_Amount	Max_total_amount	Mean_total_amount	Median Total Amount	sum_total_amount
0	2022-00	61317	-108.0	499.00	14.425181	9.5	884508.82
1	2022-01	503582	-480.0	401092.32	13.996555	9.0	7048413.00
2	2022-02	549219	-335.0	640.50	12.322440	9.0	6767718.28
3	2022-03	579091	-250.0	720.00	12.306576	9.0	7126627.44
4	2022-04	549756	-175.0	650.00	12.359779	9.0	6794862.59
5	2022-05	149432	-388.0	550.00	13.503971	9.5	2017925.35

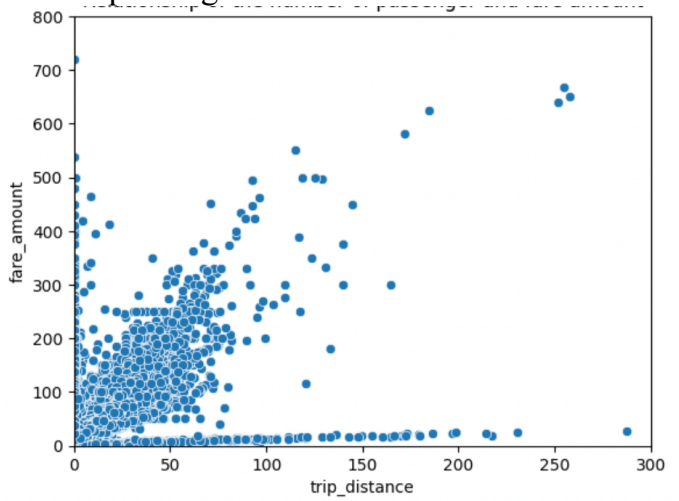
- Image above is statistics for **Weekly Fare Amount**

	Weeks	Total Count	Min_passenger_count	Max_passenger_count	Mean_passenger_count	Median passenger_count	sum_passenger_count
0	2022-00	61317	0.0	7.0	1.571701	1.0	96372.0
1	2022-01	503582	0.0	8.0	1.415754	1.0	712948.0
2	2022-02	549219	0.0	9.0	1.382654	1.0	759380.0
3	2022-03	579091	0.0	9.0	1.382674	1.0	800694.0
4	2022-04	549756	0.0	8.0	1.360271	1.0	747817.0
5	2022-05	149432	0.0	6.0	1.384382	1.0	206871.0

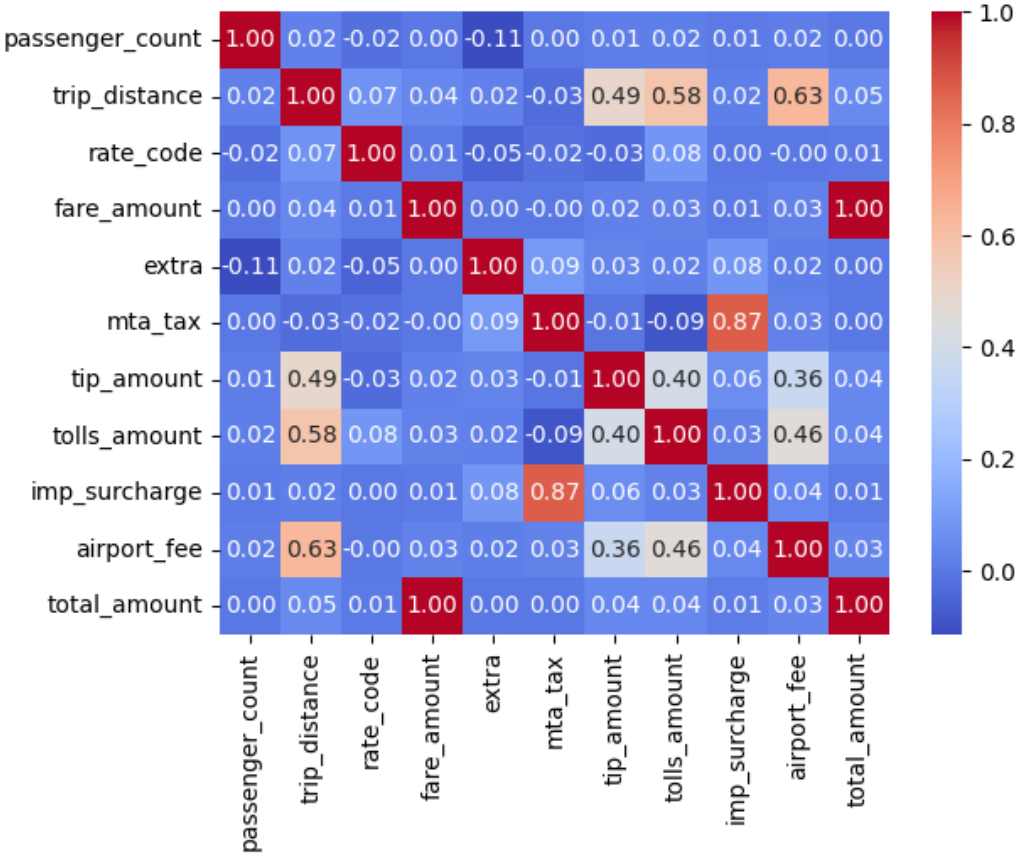
- Image above is statistics for **Weekly Passenger**



- The most common number of passengers is **one**



- The image above shows that the more trip distance is the higher the fare amount.



- This image is the correlation between fields.
    - In this case, there is a strong positive relationship between mta_tax and imp_surchage

Milestone 4: Feature Engineering and Modeling

Location of code: Milestone 4: Feature engineering & ML Model

Data Cleaning:

The Data cleaning description below is based on Yellow Taxi January 2022. This dataset is joined with Taxi - Dictionary dataset to get the fields for pickup location and drop-off location. After merging the datasets, it has now 25 fields with 25K values.

Fields and Schema Before cleaning	Fields and Schema After cleaning
root |-- VendorID: long (nullable = true) |-- tpep_pickup_datetime: timestamp (nullable = true) |-- tpep_dropoff_datetime: timestamp (nullable = true) |-- passenger_count: double (nullable = true) |-- trip_distance: double (nullable = true) |-- RatecodeID: double (nullable = true) |-- store_and_fwd_flag: string (nullable = true) |-- PULocationID: long (nullable = true) |-- DOLocationID: long (nullable = true) |-- payment_type: long (nullable = true) |-- fare_amount: double (nullable = true) |-- extra: double (nullable = true) |-- mta_tax: double (nullable = true) |-- tip_amount: double (nullable = true) |-- tolls_amount: double (nullable = true) |-- improvement_surcharge: double (nullable = true) |-- total_amount: double (nullable = true) |-- congestion_surcharge: double (nullable = true) |-- airport_fee: double (nullable = true) |-- Borough: string (nullable = true) |-- Zone: string (nullable = true) |-- service_zone: string (nullable = true) |-- Borough: string (nullable = true) |-- Zone: string (nullable = true) |-- service_zone: string (nullable = true)	root |-- passenger_count: double (nullable = true) |-- trip_distance: double (nullable = true) |-- fare_amount: double (nullable = true) |-- tip_amount: double (nullable = true) |-- tolls_amount: double (nullable = true) |-- airport_fee: double (nullable = true) |-- Pickup_Borough: string (nullable = true) |-- Dropoff_Borough: string (nullable = true) |-- Pickup_time: string (nullable = true) |-- Dropoff_time: string (nullable = true) |-- Pickup_date: string (nullable = true) |-- Dropoff_date: string (nullable = true) |-- TimeDifference in min: double (nullable = true)

Cleaning Stages:

⇒ For Yellow taxi 2022, Jan dataset, there are 71503 null values for the passenger count field and two more fields.
- In this case, the Null values are removed

⇒ There are no duplicates values

⇒ For both fields, Pickup datetime and Dropoff Datetime is converted into four fields by separating the date and time for as the image above (right-image, in the red box)

⇒ The field Borough (from taxi-dictionary) is converted into two borough fields based on Pickup and drop-off location ID.

⇒ A new Field Time_difference_In_min is added which is calculated by getting the minutes difference from Dropoff_time and the PickUp time.

⇒ Finally, the unnecessary fields are excluded from the dataset and the final output is saved in the

⇒ Excluded the outlier such as the negative value for fare amount is negative.

⇒ Replaced passenger count 0 with 1

Updated Cleaning Stage:

⇒ Excluded the outlier using standard deviation of fare_amount, trip_distance, TimeDifference_in_min (difference between drop-off time and pickup time), trip amount

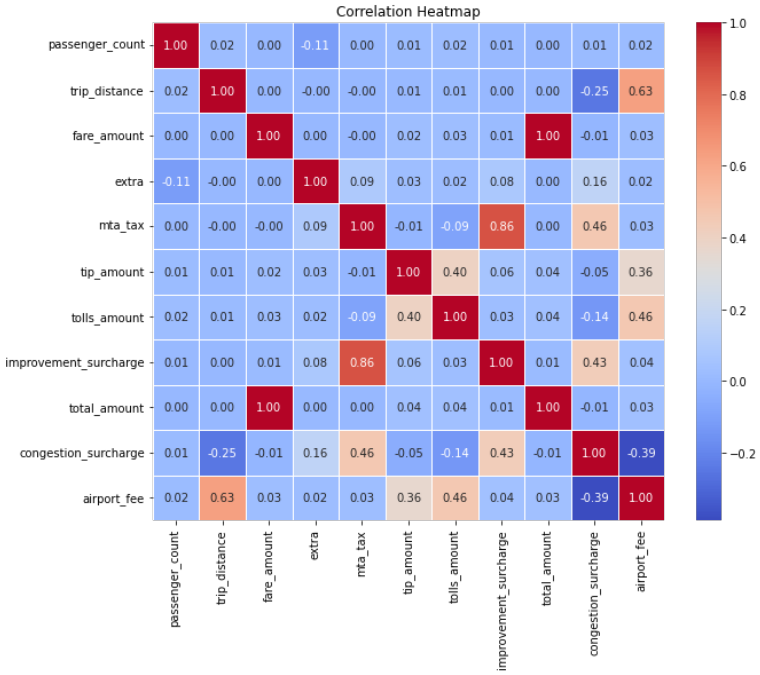
As a result, there are total 13 column as the image below.

A sample of Yellow Taxi data after cleaning:

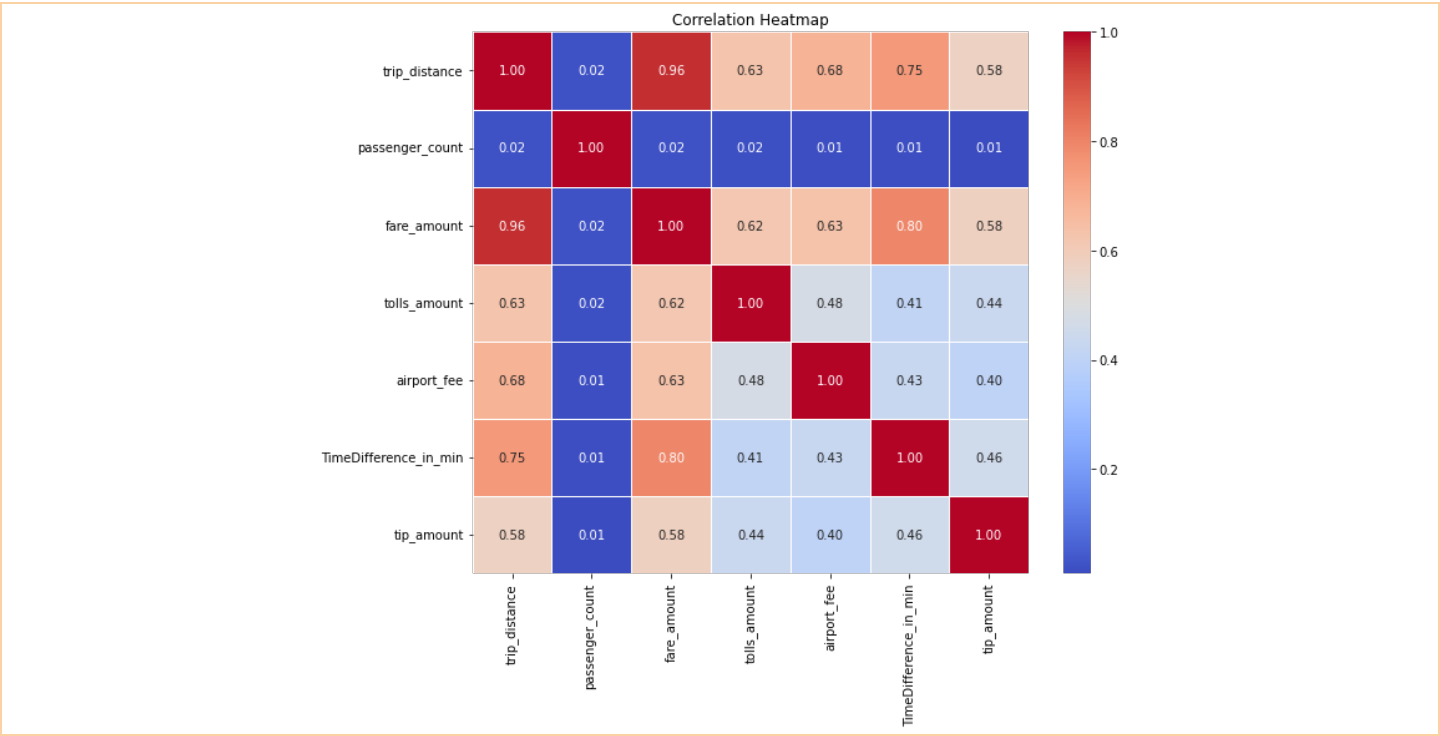
passenger_c	trip_distance	fare_amount	tip_amount	tolls_amoun	airport_fee	Pickup_Boro	Dropoff_Bor	Pickup_time	Dropoff_tim	Pickup_date	Dropoff_date	TimeDifferer
2	3.8	14.5	3.65	0	0	Manhattan	Manhattan	0:35:40	0:53:29	1/1/22	1/1/22	17.8166667
1	2.1	8	4	0	0	Manhattan	Manhattan	0:33:43	0:42:07	1/1/22	1/1/22	8.4
1	0.97	7.5	1.76	0	0	Manhattan	Manhattan	0:53:21	1:02:19	1/1/22	1/1/22	8.96666667
1	1.09	8	0	0	0	Manhattan	Manhattan	0:25:21	0:35:23	1/1/22	1/1/22	10.0333333
1	4.3	23.5	3	0	0	Manhattan	Manhattan	0:36:48	1:14:20	1/1/22	1/1/22	37.5333333

Note: The code for data cleaning located in this Appendix under Data cleaning section

Correlation before cleaning



Correlation After cleaning



Data Modeling:

The below description of feature engineering dataset based on yellow taxi trip data 2022 January which is located in the raw folder in AWS s3 as parquet format.

⇒ The target is to predict the **TimeDifference_in_min** based on fields, Pickup_Borough,Dropoff_Borough,trip_distance, 'passenger_count', fare_amount,tolls_amount, airport_fee, tim_amount.

- In this case, the string data-types coulmns, Pickup_Borough,Dropoff_Borough, are converted into indexer then encoder.
- Finally assembler is created using desired feature vectors and the float/double columns and the output will store in the **features** column.
- A pipeline is created using the indexer, encoder, assembler to fit and transform the entire dataset

ime	Dropoff_tim	Pickup_date	Dropoff_date	TimeDifferer	Pickup_Boroug	Dropoff_Bor	Pickup_Borough_Vector	Dropoff_Borough_Vector	features
5:40	0:53:29	1/1/22	1/1/22	17.8166667	0	0	{"vectorType":"sparse","length":7,"ind	{"vectorType":"sparse","length":7,"indices	{"vectorType":"sparse","length"
3:43	0:42:07	1/1/22	1/1/22	8.4	0	0	{"vectorType":"sparse","length":7,"ind	{"vectorType":"sparse","length":7,"indices	{"vectorType":"sparse","length"
3:21	1:02:19	1/1/22	1/1/22	8.96666667	0	0	{"vectorType":"sparse","length":7,"ind	{"vectorType":"sparse","length":7,"indices	{"vectorType":"sparse","length"

Trial one: LinearRegression

⇒ ML model **LinearRegression** is used to predict the tip amount as the tip amount is continues variable

⇒ The dataset is split into 70% training data and 30% test data.

⇒ The metrics to evaluate the model on the training dataset:

- Training Dataset RMSE: 5.836739
- Training Dataset r2: 0.664520 (66.04% of the variance in the target variable)

⇒ A sample of Testing dataset after using the model

Pickup_Borough	Dropoff_Borough	trip_distance	passenger_count	fare_amount	tolls_amount	airport_fee	TimeDifference_in_min	tip_amount	prediction
Queens	Queens	0	0	-3	0	0	1.4833333333333334	0	1.0093222358112313
Manhattan	Manhattan	0	0	0	0	0	0.5833333333333334	0	1.4967211683116055
Manhattan	Manhattan	0	0	0	0	0	0.23333333333333334	0	1.4966970063891782
Manhattan	Manhattan	0	0	0	0	0	0.65	0	1.496725770582544

⇒ The metrics to evaluate the model on the testing dataset:

- Training Dataset RMSE: 5.836739

- Training Dataset r2: 0.664520 (66.04% of the variance in the target variable which is very low)

⇒ The training dataset is saved in the trusted folder in S3 and the model is saved in the model folder
Note: The code for this feature engineering and modeling in this appendix (section 2 & 3)

Updated Data Modeling:

Trial 2: Cross Validator with Linear regression (folds of 3)

⇒ Cross Validator with the hyperparameter grid and its schema **Num-Fold of 3** was performed. As a result, the metrics has the same result with only using linear Regression model.

Evaluation on Training Dataset:

- RMSE: 5.866501002979682
- R-squared:0.6618367430563542
- MSE: 34.41583401796161

Evaluation on Testing Dataset:

- *RMSE: 5.798944003068859*
- *R-squared:0.6696697767150597*
- *MSE: 33.62775155072828*

Trial 2.5: Cross Validator with Linear regression (folds of 5)

Num-folds of 5 was also performed as a result the RMSE and R-Squared the both values remain the **same**

⇒ Statistics of prediction of TimeDifference_in_min VS its actual values

- The mean for both prediction and the time_in_difference_in_min is same while the min and max have the huge difference.

⇒

	summary	prediction	TimeDifference_in_min
1	count	699951	699951
2	mean	12.674140533226272	12.66846922141698
3	stddev	8.252676173582094	10.115533874273577
4	min	-47.86845510432565	0.0
5	max	97.366994292618	657.2

Trial 3 GeneralizedLinearRegression with family Poisson:

Train evaluation metrics:
RMSE: 7.6347150911621595
R-squared:0.4260005051407372
MSE: 58.28887452321923

Test evaluation metrics:
RMSE: 7.669960373846633
R-squared:0.42507758212273683
MSE: 58.828292136377584

Test and train accuracy for R-Squared is **lower** than the Trial 2: Cross Validator with Linear regression (folds of 3).

Therefore, the best Model is the trial 2: Cross Validator with Linear regression (folds of 3)

Challenges:
The challenges of this milestone were to merge the two datasets because I had to merge trip data its two different location ID with one single ID from the dictionary. Another challenge is to find the difference of minutes of a pickup and drop-off time as it involves many data conversions.

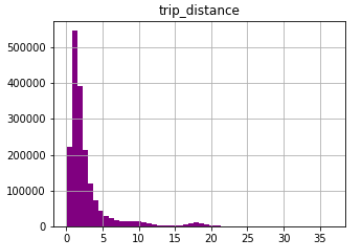
Milestone 5: Data Visualizing

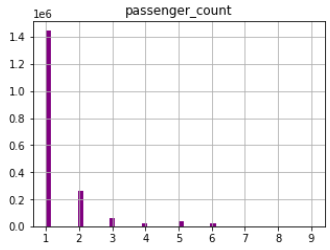
The following visualization is done by using python-pandas packages: Matplotlib, Seaborn.
Location in Appendix: Milestone 5: DATA Visualization

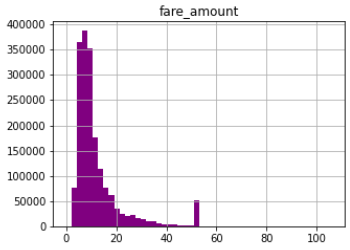
EDA visualization:

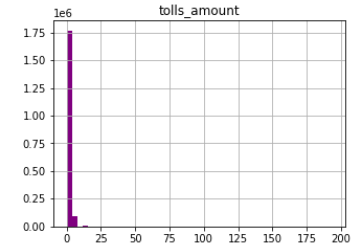
1. Histograms for different fields:

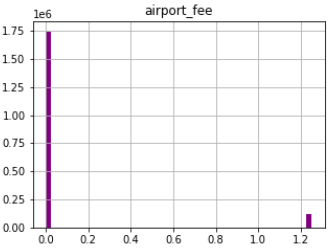
- No outlier: all fields are strongly right skewed

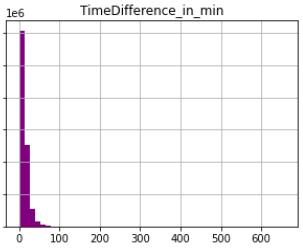


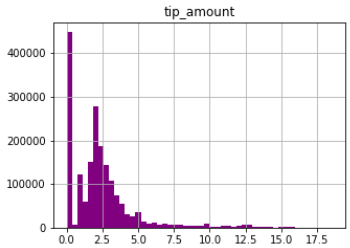






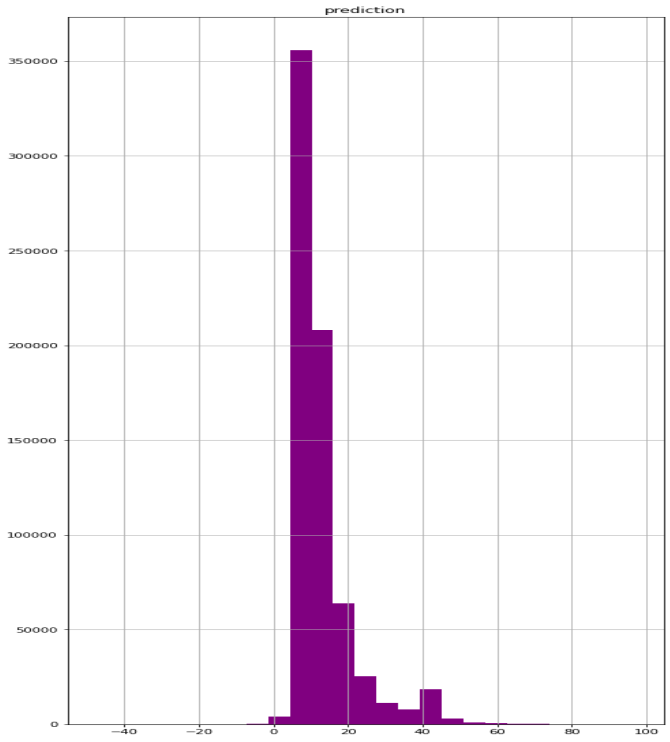


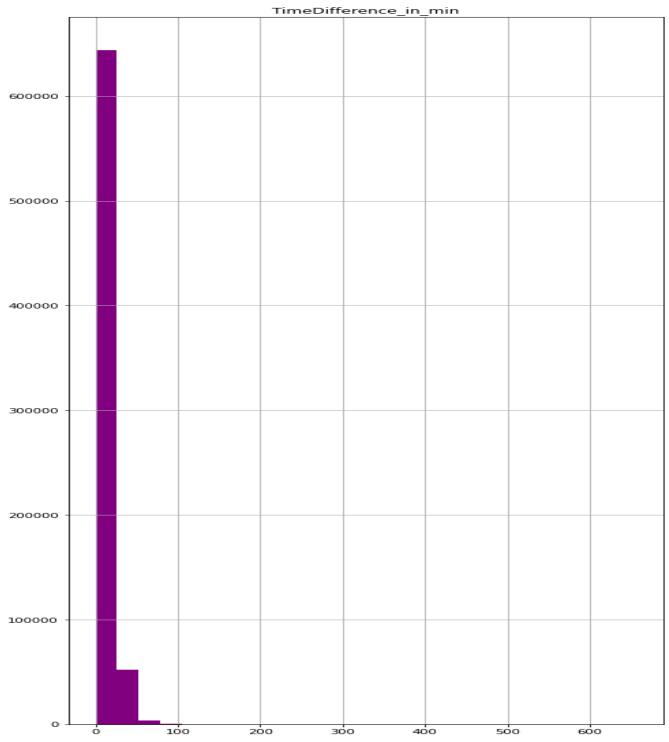




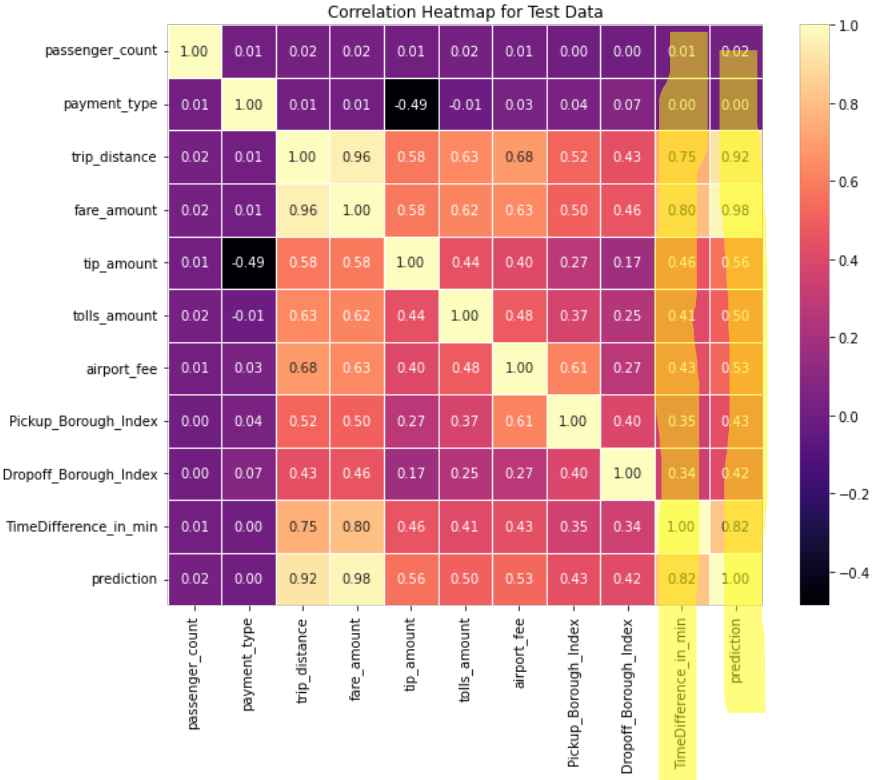
TEST Data Visualization:

⇒ 2. **Histogram** of the Prediction vs the actual values of time in difference



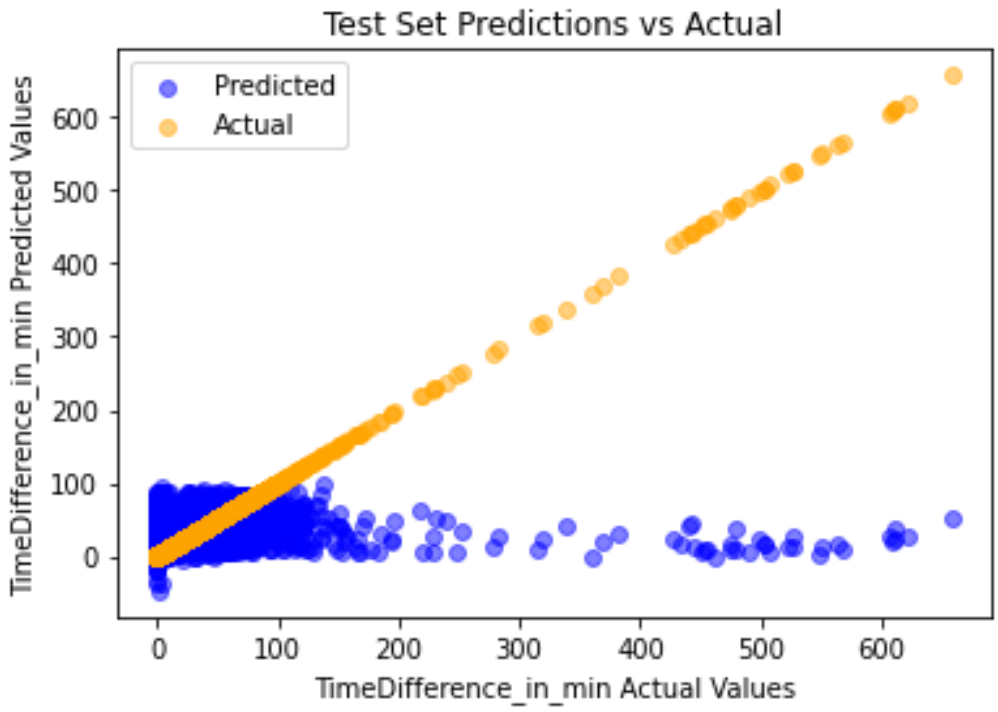


⇒ 3. **Correlation heatmap** comparison with Prediction vs Actual Time in difference



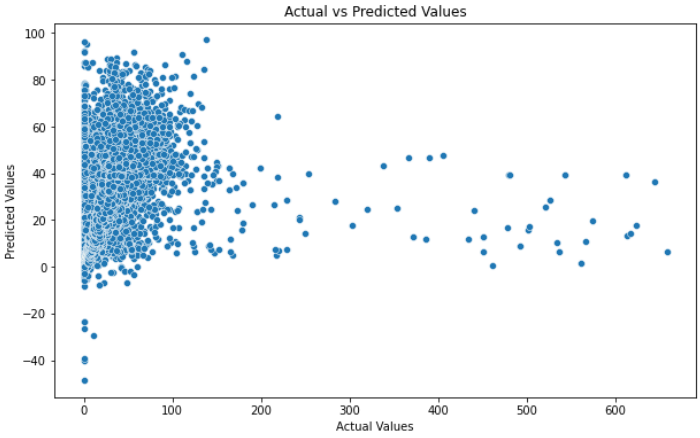
- Based on this correlation for the prediction value Time Difference in Min has **stronger positive** correlation than the actual value of Time Difference

⇒ **4. Scatter Plot for actual vs predicted**:

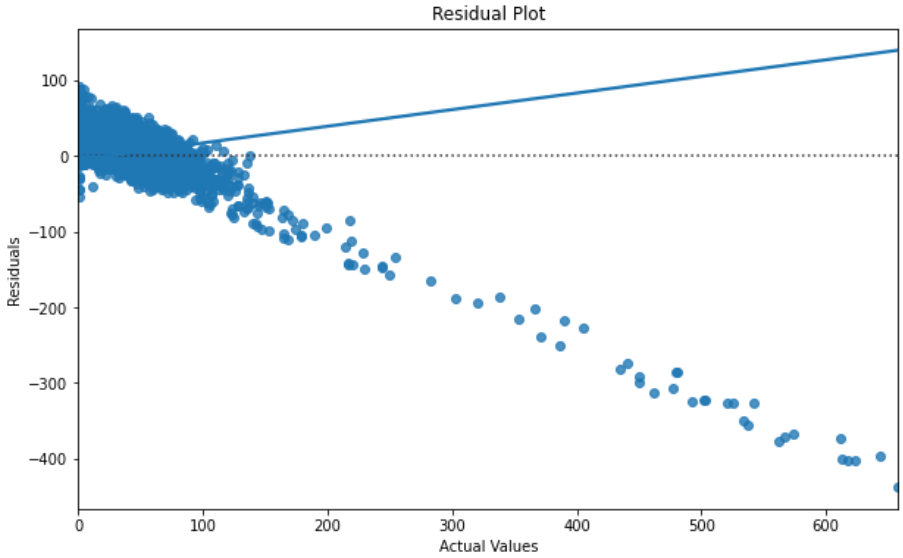


- In this scatter plot the yellow is the actual values of Time Difference in min where it is strongly positive correlated when it is less than 200 minutes.
- The blue scatterplot, the predicted of Time Difference in min is also positive correlated but not highly as actual values and it is less than 100 correlated.

⇒ **5. Alternative** visualization of Scatter Plot for actual vs predicted:



⇒ **6. Residual Plots:**



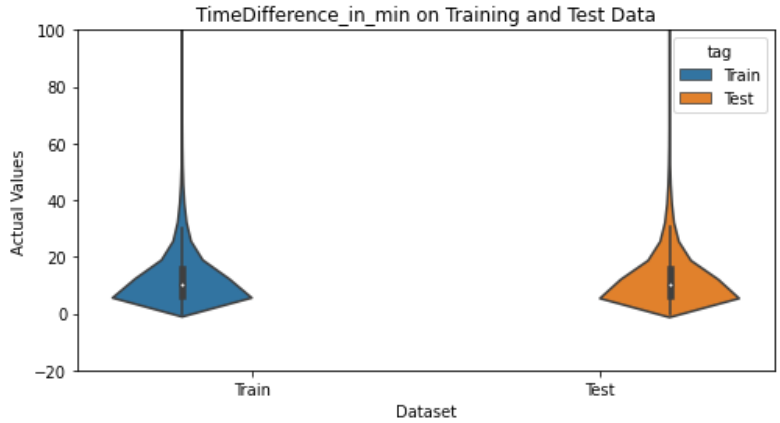
- This is a residual plot where it describes differences between the observed values and the corresponding predicted values.
- In this plot it is not linear and has negative residuals values

⇒ Violin Plot:

Reference/Credit: https://juanitorduz.github.io/lm_viz/

- Violin plot is like a box plot, but it provides more information about the density.

Actual values Time_difference_in_min plot:



- Both testing and training datasets violin plots are the same.
- Both are positive and the higher density is at **5-25.**

Predictions of Time_difference_in_min plot:

- Both testing and training datasets violin plots are almost the same.
- Their higher density is at 5-15 which is lower than the actual values.
- Also, it has a slightly density at 40 which does not exist in the actual values for both training and test datasets.
- There is also negative density.

Milestone 6: **Summary and Conclusion**

This project focuses on Yellow Taxi Trip data from 2019-2022. Those trip data are formatted as paraquet of 5 different files based on the years. Those datasets are collected from the website, https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

First step of this project is **Data Acquisition.** Initially I used Google public big datasets for Yellow Taxi Trip data to download and store it to the AWS S3. I used AWS CLI to download the gcloud so that I can copy and upload the dataset from Google public big dataset to AWS s3. Alternatively, I used the yellow taxi trip Data from TLC website to download and store it to S3 in landing folder using Curl. To do so, I created one Instance in AWS EC2 under IAM user. Then a bucket “my-bigdata-project-sh” is created to store the yellow taxi

datasets. Under my-bigdata-project-sh folder, I created multiple folders, landing, raw, trusted, curated and models.

Then I performed **EDA** to explore the data. There are 25 fields where 5 fields are string datatypes, and 20 fields are integer or float datatypes or datetime. There are a few null values but no duplicates values. The all fields are strongly right skewed and has outliers for majority fields.

The next step is to **feature engineering and modeling**.

Before this step, I created a column name 'time_difference_in_min' by calculating the difference between drop-off time and the pickup time. Then I cleaned the null values and excluded the unnecessary fields. I replaced the number passenger value 0 with 1 and used standard deviation to exclude the outlier for different fields. Finally cleaned dataset is saved in the AWS S3 under raw folder. As a result, of data cleaning the correlations between fields are increased positively.

In data modeling stage, the string data-types columns, Pickup_Borough,Dropoff_Borough, are converted into indexer then encoder. Finally, assembler is created using desired feature vectors and the float/double columns and the output will store in the features column. Finally, a pipeline is built using the indexer, encoder, assembler to fit and transform the entire dataset to predict the time_difference_in_min.

I split the tidy data into two parts: 80% for training and 20% for testing. Then, I tried various machine learning methods like Linear Regression and Generalized Linear Regression with the Poisson family to get accurate results. I also used a Cross Validator with Linear Regression, testing it with 3 and 5 folds. Finally, I found the best model using the Cross Validator with 3 folds, which helped optimize the models with the best hyperparameters. The accuracy rates were 66.1% for training and 66.9% for testing, showing how well the model performed on both sets of data. Model is saved in AWS S3 model folder.

Finally, I created various visualizations to compare the actual time values with the predicted ones. I used histogram plots, a correlation heatmap, scatter plot, residual plot, and a violin plot for this purpose. These visuals revealed that the model predicts negative values for the number of passengers. However, it showed

stronger connections with other fields than with the actual time values. The visualizations are saved in the AWS S3.

Milestone 7: **GitHub**

https://github.com/shariahoque01/Big-Data-Technologies-with-NYC-Taxi-Data

A repository, Big-Data-Technologies-with-NYC-Taxi-Data, is created where it has a PDF version of project's steps and description as well as codes in the appendix section.

Appendix

1. Appendix : Data Acquisition

- Dividing the big data into Monthly.

```
for month in {01..12}; do
  table_name="Yellow_Taxi.tlc_yellow_trips_2020_$month"
  sql_query="CREATE OR REPLACE TABLE $table_name AS
    SELECT
    * FROM \`cis-4130-project.Yellow_Taxi.tlc_yellow_trips_2020\`
    WHERE EXTRACT(YEAR FROM pickup_datetime) = 2020
    AND EXTRACT(MONTH FROM pickup_datetime) = $month;"

  echo "Executing SQL query for $table_name"
  bq query --use_legacy_sql=false --nouse_legacy_sql <<< "$sql_query"
done
```

- Transferring it to the Google bucket:

```
for month in {01..12}; do
  table_name="Yellow_Taxi.tlc_yellow_trips_2020_$month"
  destination="gs://cis_4130_project_bucket/$table_name"
  echo "Copying $table_name to $destination"
  bq extract --destination_format CSV "cis-4130-project:$table_name" $destination
done
```

- Transferring the files into the AWS: downloaded G-gloud in AWS cli

```
sudo yum install python3 python3-pip
curl https://sdk.cloud.google.com | bash
gcloud init
```

-

gsutil cp gs://cis_4130_project_bucket/tlc_yellow_trips_2022/ ~/*

aws s3 cp ~/tlc_yellow_trips_2022_01 s3://sep.20.first.draft.bucket/tlc_taxi_trips_2022/

```
for month in {01..12}; do
  aws s3 cp tlc_yellow_trips_2021_$month s3://sep.20.first.draft.bucket/tlc_taxi_trips_2021/
done
```

The updated code to download datasets from TLC

```
SOURCE_PREFIX="https://d37ci6vzurychx.cloudfront.net/trip-data/"
DEST_BUCKET="s3://my-bigdata-project-sh/landing/tlc_taxi_trips_2021/"
for month in {01..12}; do
   URL="${SOURCE_PREFIX}yellow_tripdata_2021-${month}.parquet"
   curl -SL $URL | aws s3 cp - $DEST_BUCKET"yellow_tripdata_2021-$month.parquet"
   sleep 1
done
```

2. Appendix : EDA (Milestone 3)

```
taxi_2022_01_df = pd.read_csv("s3://my-bigdata-project-
sh/landing/tlc_taxi_trips_2022/tlc_yellow_trips_2022_01", sep=',', on_bad_lines='skip')
print(taxi_2022_01_df.head(3))
print(taxi_2022_01_df.shape)
print(taxi_2022_01_df.columns)
print(taxi_2022_01_df.describe())
print(taxi_2022_01_df.values)
print(taxi_2022_01_df.info())
print(taxi_2022_01_df.duplicated().sum()
print(taxi_2022_01_df.columns[taxi_2022_01_df.isnull().any()].tolist())
taxi_2022_01_df = taxi_2022_01_df.dropna(axis=0, subset=['passenger_count'])
taxi_2022_01_df.isnull().sum()
from IPython.display import display
numeric_coulmns = taxi_2022_01_df[[ 'passenger_count',
      'trip_distance', 'rate_code',
      'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
      'imp_surcharge', 'airport_fee', 'total_amount' ]]
mean_numeric_coulmns = numeric_coulmns.agg( ['count', 'min', 'max', 'mean', 'median']).reset_index()
```

```
display("Average for different fields:\n",mean_numeric_coulmns)
weekly_sum = taxi_2022_01_df.groupby(taxi_2022_01_df['pickup_date'].dt.strftime('%Y-
%U'))['fare_amount'].agg( ['count', 'min', 'max', 'mean', 'median','sum']).reset_index()

# Rename the columns for clarity
weekly_sum.columns = ['Weeks','Total Count', 'Min_total_Amount', 'Max_total_amount',
'Mean_total_amount', 'Median Total Amount','sum_total_amount']


display(weekly_sum)
weekly_sum = taxi_2022_01_df.groupby(taxi_2022_01_df['pickup_date'].dt.strftime('%Y-
%U'))['passenger_count'].agg( ['corr','count', 'min', 'max', 'mean', 'median','sum']).reset_index()

# Rename the columns for clarity
weekly_sum.columns = ['Weeks','Total Count', 'Min_passenger_count', 'Max_passenger_count',
'Mean_passenger_count', 'Median passenger_count','sum_passenger_count']

display(weekly_sum)


taxi_2022_01_df['passenger_count'].plot(kind='hist', bins=10)

taxi_2022_01_df['trip_distance'].plot(kind='hist', bins=2)

ax = sns.scatterplot(data=taxi_2022_01_df, x='passenger_count', y='fare_amount');

ax.set_title("Relationship of the number of passenger and fare amount")
ax.set_ylim(0, 1000)

ax = sns.scatterplot(data=taxi_2022_01_df, x='trip_distance', y='fare_amount');

ax.set_title("Relationship of the number of passenger and fare amount")
ax.set_xlim((0,300))
ax.set_ylim(0, 800)

sub_df = taxi_2022_01_df[['passenger_count',
      'trip_distance', 'rate_code',
      'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
      'imp_surcharge', 'airport_fee', 'total_amount' ]]

sns.pairplot(taxi_2022_01_df,hue='class');
sns.heatmap(sub_df.corr(),annot= True, cmap='coolwarm', fmt='.2f')
```

3.

3. Appendix : Data Cleaning (Milestone 4)

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
```

```
from pyspark.ml import Pipeline
# Import the logistic regression model
from pyspark.ml.classification import LogisticRegression, LogisticRegressionModel
# Import the evaluation module
from pyspark.ml.evaluation import *
# Import the model tuning module
from pyspark.ml.tuning import *
import numpy as np
spark = SparkSession.builder.getOrCreate()

import os

import pandas as pd
from pyspark.sql.functions import col
from pyspark.ml.stat import Correlation
from pyspark.ml.stat import Summarizer


---------------credentials & spark packages-----------------
access_key = 'AOV74J'
secret_key = ''
os.environ['AWS_ACCESS_KEY_ID'] = access_key
os.environ['AWS_SECRET_ACCESS_KEY'] = secret_key
encoded_secret_key = secret_key.replace("/", "%2F").replace("+", "%2B")
# Set aws_region to where your S3 bucket was created
aws_region = "us-east-2"
spark.sparkContext._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)
spark.sparkContext._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)
spark
spark = SparkSession.builder \
    .appName("Databricks Shell") \
    .config("spark.jars.packages", "org.apache.hadoop:hadoop-aws:your_hadoop_version")
\
    .getOrCreate()


yellow_tripdata_2022_01 = spark.read.parquet("s3://my-bigdata-project-
sh/landing/tlc_taxi_trips_2022/yellow_tripdata_2022-01.parquet", engine='pyarrow')
taxi_dictionary = spark.read.csv("s3://my-bigdata-project-
sh/landing/taxi+_zone_lookup.csv", header=True)
yellow_tripdata_2022_01 = yellow_tripdata_2022_01 \
    .join(taxi_dictionary.alias("pu"),
          yellow_tripdata_2022_01["PULocationID"] == col("pu.LocationID"),
          "inner") \
    .drop("LocationID")  # Drop redundant LocationID column

# Alias for the second join (DOLocationID)
```

```
yellow_tripdata_2022_01 = yellow_tripdata_2022_01 \
    .join(taxi_dictionary.alias("do"),
          yellow_tripdata_2022_01["DOLocationID"] == col("do.LocationID"),
          "inner") \
    .drop("LocationID")

num_rows = yellow_tripdata_2022_01.count()
num_columns = len(yellow_tripdata_2022_01.columns)

print('The number of RAW rows: ', num_rows)
print('The number of RAW columns: ', num_columns)

new_columns = [f'{col}_{i}' if col == 'Borough' else col for i, col in
enumerate(yellow_tripdata_2022_01.columns, 1)]

yellow_tripdata_2022_01 = yellow_tripdata_2022_01.toDF(*new_columns)



yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumnRenamed("Borough_20",
"Pickup_Borough")
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumnRenamed("Borough_23",
"Dropoff_Borough")

yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumn("Pickup_time",
date_format("tpep_pickup_datetime", "HH:mm:ss"))
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumn("Dropoff_time",
date_format("tpep_dropoff_datetime", "HH:mm:ss"))
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumn("Pickup_date",
date_format("tpep_pickup_datetime", "yyyy-MM-dd"))
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumn("Dropoff_date",
date_format("tpep_dropoff_datetime", "yyyy-MM-dd"))


yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumn(
    "TimeDifference_in_min",
    abs((expr("CAST(Dropoff_time AS TIMESTAMP)").cast("long") - expr("CAST(Pickup_time
AS TIMESTAMP)").cast("long")) / 60)
)


columns_to_drop = ['PULocationID','DOLocationID','VendorID',  'RatecodeID',
'store_and_fwd_flag',  'extra', 'mta_tax', 'improvement_surcharge', 'total_amount',
'congestion_surcharge', 'LocationID',  'Zone', 'service_zone',
'tpep_pickup_datetime','tpep_dropoff_datetime']
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.drop(*columns_to_drop)
```

```
null_columns = yellow_tripdata_2022_01.columns
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.na.drop(subset=null_columns)

num_rows = yellow_tripdata_2022_01.count()
num_columns = len(yellow_tripdata_2022_01.columns)

print('The number of rows: ', num_rows)
print('The number of columns: ', num_columns)

      yellow_tripdata_2022_01.printSchema()
```

4.

UPDATED Data cleaning

```
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.filter(
    (col('fare_amount') > 0) & (col('trip_distance') > 0) & (col('tip_amount') >= 0) &
(col('tolls_amount') >= 0) & (col('airport_fee') >= 0)
)
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.filter(
    (col('fare_amount') < 1000 ) )
yellow_tripdata_2022_01 = yellow_tripdata_2022_01.withColumn(
    'passenger_count',
    when(col('passenger_count') == 0, 1).otherwise(col('passenger_count'))
)
selected_columns = ['trip_distance', 'passenger_count', 'fare_amount', 'tolls_amount',
'airport_fee', 'TimeDifference_in_min', 'tip_amount']

correlation_matrix = yellow_tripdata_2022_01.select(selected_columns).toPandas().corr()


# Display a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title("Correlation Heatmap")
plt.show()

def filter_data_by_stddev(df, column, multiplier):
    # Calculate mean and stddev
    mean_val = df.agg({column: 'mean'}).collect()[0][0]
    stddev_val = df.agg({column: 'stddev'}).collect()[0][0]

    # Create lower and upper bounds
```

```
    low_bound = mean_val - (multiplier * stddev_val)
    hi_bound = mean_val + (multiplier * stddev_val)

    # Filter the data
    filtered_df = df.filter((col(column) > low_bound) & (col(column) < hi_bound))

    return filtered_df
yellow_tripdata_2022_01 = filter_data_by_stddev(yellow_tripdata_2022_01, 'fare_amount',
8)
yellow_tripdata_2022_01 = filter_data_by_stddev(yellow_tripdata_2022_01,
'passenger_count', 8)
yellow_tripdata_2022_01 = filter_data_by_stddev(yellow_tripdata_2022_01,
'trip_distance', 8)
# 6.5
yellow_tripdata_2022_01 = filter_data_by_stddev(yellow_tripdata_2022_01,
'TimeDifference_in_min', 6)
yellow_tripdata_2022_01 = filter_data_by_stddev(yellow_tripdata_2022_01, 'tip_amount',
6)
selected_columns = ['trip_distance', 'passenger_count', 'fare_amount', 'tolls_amount',
'airport_fee', 'TimeDifference_in_min', 'tip_amount', 'payment_type']

correlation_matrix = yellow_tripdata_2022_01.select(selected_columns).toPandas().corr()


# Display a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title("Correlation Heatmap")
plt.show()

yellow_tripdata_2022_01.write.parquet("s3://my-bigdata-project-
sh/raw/Taxi_Trip_2022/yellow_tripdata_2022_01", mode="overwrite")
```

5. Appendix : **Milestone 4: Feature engineering & ML Model**

```
yellow_tripdata_2022_01 = spark.read.parquet('s3://my-bigdata-project-
sh/raw/Taxi_Trip_2022/', engine='pyarrow')
trainingData, testData = yellow_tripdata_2022_01.randomSplit([0.7, 0.3], seed=42)

indexer = StringIndexer(inputCols=['Pickup_Borough','Dropoff_Borough'],
outputCols=['Pickup_Borough_Index','Dropoff_Borough_Index'])
encoder = OneHotEncoder(inputCols=['Pickup_Borough_Index', 'Dropoff_Borough_Index'],
outputCols=['Pickup_Borough_Vector', 'Dropoff_Borough_Vector'], dropLast=False)
assembler =
VectorAssembler(inputCols=['Pickup_Borough_Vector','Dropoff_Borough_Vector','trip_dista
```

```
nce', 'fare_amount','tolls_amount','airport_fee','tip_amount',' 'payment_type'],
outputCol="features")

#cross-validation

from pyspark.ml.regression import LinearRegression
lr = LinearRegression(labelCol='TimeDifference_in_min')
evaluator = RegressionEvaluator(labelCol='TimeDifference_in_min')

yellow_tripdata_2022_01_pipe = Pipeline(stages=[indexer, encoder, assembler,lr])
grid = ParamGridBuilder()
# Build the parameter grid
grid = grid.build()
# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=yellow_tripdata_2022_01_pipe, estimatorParamMaps=grid,
evaluator=evaluator, numFolds=3)
# Train the models
all_models = cv.fit(trainingData)
# Get the best model from all of the models trained
bestModel = all_models.bestModel
# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(testData)


test_results = bestModel.transform(testData)
# Show the predicted tip
# display(test_results.limit(5))
# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
mse = evaluator.evaluate(test_results, {evaluator.metricName: "mse"})

print(f"RMSE: {rmse} R-squared:{r2} MSE: {mse}")

train_results = bestModel.transform(trainingData)
# Show the predicted tip
display(train_results.limit(5))
# Calculate RMSE and R2
rmse = evaluator.evaluate(train_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(train_results,{evaluator.metricName:'r2'})
mse = evaluator.evaluate(train_results, {evaluator.metricName: "mse"})

print(f"RMSE: {rmse} R-squared:{r2} MSE: {mse}")
```

```
test_result = test_results.withColumn("prediction_error", col("prediction") -
col("TimeDifference_in_min"))

# Show the DataFrame with predicted tip and prediction error
test_result.select("prediction", "TimeDifference_in_min", "prediction_error").show(5)
```

#Linear regression only:

```
yellow_tripdata_2022_01 = spark.read.parquet('s3://my-bigdata-project-
sh/raw/Taxi_Trip_2022/', engine='pyarrow')


indexer = StringIndexer(inputCols=['Pickup_Borough','Dropoff_Borough'],
outputCols=['Pickup_Borough_Index','Dropoff_Borough_Index'])
encoder = OneHotEncoder(inputCols=['Pickup_Borough_Index', 'Dropoff_Borough_Index'],
outputCols=['Pickup_Borough_Vector', 'Dropoff_Borough_Vector'], dropLast=False)
assembler =
VectorAssembler(inputCols=['Pickup_Borough_Vector','Dropoff_Borough_Vector','trip_dista
nce', 'fare_amount','tolls_amount','airport_fee','tip_amount'], outputCol="features")

yellow_tripdata_2022_01_pipe = Pipeline(stages=[indexer, encoder, assembler])

transformed_sdf =
yellow_tripdata_2022_01_pipe.fit(yellow_tripdata_2022_01).transform(yellow_tripdata_202
2_01)

trainingData, testData = transformed_sdf.randomSplit([0.7, 0.3], seed=42)
lr = LinearRegression(labelCol='TimeDifference_in_min')
model = lr.fit(trainingData)
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)

Test_results = model.transform(testData)
rmse = evaluator.evaluate(Test_results, {evaluator.metricName:'rmse'})
mse = evaluator.evaluate(Test_results, {evaluator.metricName: "mse"})
r2 =evaluator.evaluate(Test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2} MSE: {mse}")

train_results = model.transform(trainingData)
rmse = evaluator.evaluate(train_results, {evaluator.metricName:'rmse'})
mse = evaluator.evaluate(train_results, {evaluator.metricName: "mse"})
r2 =evaluator.evaluate(train_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2} MSE: {mse}")
```

```
# GeneralizedLinearRegression

from pyspark.ml.regression import GeneralizedLinearRegression


Glr = GeneralizedLinearRegression(labelCol='TimeDifference_in_min', family="poisson")

model = Glr.fit(trainingData)
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)
trainingSummary = model.summary

train_results = model.transform(trainingData)
rmse = evaluator.evaluate(train_results, {evaluator.metricName:'rmse'})
mse = evaluator.evaluate(train_results, {evaluator.metricName: "mse"})
r2 =evaluator.evaluate(train_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2} MSE: {mse}")


Test_results = model.transform(testData)

rmse = evaluator.evaluate(Test_results, {evaluator.metricName:'rmse'})
mse = evaluator.evaluate(Test_results, {evaluator.metricName: "mse"})
r2 =evaluator.evaluate(Test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2} MSE: {mse}")


test_results.write.parquet("s3://my-bigdata-project-sh/trusted/Taxi_2022/", mode="overwrite")
bestModel.write().overwrite().save('s3://my-bigdata-project-sh/models/taxi_2022/')
```

6. Appendix: Milestone 5: DATA Visualization

```
sampled_df = yellow_tripdata_2022_01.sample(fraction=0.8, seed=42)

pandas_df = sampled_df.select('trip_distance', 'passenger_count', 'fare_amount',
'tolls_amount', 'airport_fee', 'TimeDifference_in_min', 'tip_amount').toPandas()

#Histplot
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(18, 13))

# Saving the visualization in s3
```

```
img_data = io.BytesIO()
fig.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)

s3 = s3fs.S3FileSystem(key=access_key, secret=secret_key, anon=False)
with s3.open('s3://my-bigdata-project-sh/curated/Histograms_of_all_fields.png', 'wb')
as f:
    f.write(img_data.getbuffer())



# correlation_matrix
selected_columns = ['passenger_count', 'trip_distance', 'fare_amount', 'tip_amount',
'tolls_amount', 'airport_fee',  'TimeDifference_in_min', 'Pickup_Borough_Index',
'Dropoff_Borough_Index',  'prediction']

correlation_matrix = test_results.select(selected_columns).toPandas().corr()


# Display a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='magma', fmt=".2f", linewidths=.5)
plt.title("Correlation Heatmap for Test Data")
plt.show()
# Saving the visualization in s3

with s3.open('s3://my-bigdata-project-sh/curated/Correlation_Heatmap.png', 'wb') as f:
    f.write(img_data.getbuffer())

# Histograms of predictions vs Actual values

test_results_visual = test_results.select('prediction', 'TimeDifference_in_min')
test_results_pandas_df = test_results_visual.toPandas()

plt.figure(figsize=(10, 10))
plt.hist(test_results_pandas_df['prediction'], bins=25, color="#800080")
plt.title("Histogram of Predicted Values")
plt.xlabel("Predicted Values")
plt.ylabel("Frequency")
# Saving the visualization in s3
img_data1 = io.BytesIO()
plt.savefig(img_data1, format='png', bbox_inches='tight')
img_data1.seek(0)

plt.figure(figsize=(10, 10))
plt.hist(test_results_pandas_df['TimeDifference_in_min'], bins=25, color="#800080")
```

```
plt.title("Histogram of TimeDifference_in_min")
plt.xlabel("TimeDifference_in_min")
plt.ylabel("Frequency")
# Saving the visualization in s3
img_data2 = io.BytesIO()
plt.savefig(img_data2, format='png', bbox_inches='tight')
img_data2.seek(0)

s3 = s3fs.S3FileSystem(key=access_key, secret=secret_key, anon=False)

with s3.open('s3://my-bigdata-project-sh/curated/Histogram_Predicted_Values.png', 'wb')
as f:
    f.write(img_data1.getbuffer())

with s3.open('s3://my-bigdata-project-sh/curated/Histogram_TimeDifference_in_min.png',
'wb') as f:
    f.write(img_data2.getbuffer())


# Plotting the predictions
plt.scatter(test_results_df["TimeDifference_in_min"], test_results_df["prediction"],
alpha=0.5)
plt.title("Test Set Predictions vs Actual")
plt.xlabel("TimeDifference_in_min Actual Values")
plt.ylabel("TimeDifference_in_min Predicted Values")
plt.show()
# Saving the visualization in s3
with s3.open('s3://my-bigdata-project-sh/curated/Scatterplot_Actual_vs_Predicted.png',
'wb') as f:
    f.write(img_data.getbuffer())

# alternative
pandas_df = test_results.select('TimeDifference_in_min',  'prediction').toPandas()
plt.figure(figsize=(10, 6))
sns.scatterplot(x='TimeDifference_in_min', y='prediction', data=pandas_df)
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()



# Residual Plots
plt.figure(figsize=(10, 6))
sns.residplot(x='TimeDifference_in_min', y='prediction', data=pandas_df, lowess=True)
plt.title('Residual Plot')
```

```
plt.xlabel('Actual Values')
plt.ylabel('Residuals')
plt.show()

#Violin Plot; Reference/Credit: https://juanitorduz.github.io/lm_viz/

train_predictions = bestModel.transform(trainingData)
test_predictions = bestModel.transform(testData)

train_results = train_predictions.select('TimeDifference_in_min',
'prediction').toPandas()
test_results = test_predictions.select('TimeDifference_in_min',
'prediction').toPandas()

train_results['tag'] = 'Train'
test_results['tag'] = 'Test'
pred_df = pd.concat([train_results, test_results])

fig, ax = plt.subplots(figsize=(8, 6))
sns.violinplot(x='tag', y='prediction', data=pred_df, hue='tag', palette=['#1f77b4',
'#ff7f0e'])
ax.set(title='Model Predictions on Training and Test Data', xlabel='Dataset',
ylabel='Model Prediction')
plt.show()
# Saving the visualization in s3

img_data = io.BytesIO()
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)

s3 = s3fs.S3FileSystem(key=access_key, secret=secret_key, anon=False)

with s3.open('s3://my-bigdata-project-sh/curated/Violinplot_Model_Predictions.png',
'wb') as f:
    f.write(img_data.getbuffer())




fig, ax = plt.subplots(figsize=(8, 4))
sns.violinplot(x='tag', y='TimeDifference_in_min', data=pred_df, hue='tag',
palette=['#1f77b4', '#ff7f0e'])
ax.set(title='TimeDifference_in_min on Training and Test Data', xlabel='Dataset',
ylabel='Actual Values')
ax.set_ylim(-20, 100)
plt.show()
```
