CIS 4130

Robert Plumbi

Project Title: Yellow Taxi Data Analysis

Contents

Introduction:

The project's objective is performing data analysis on New York Taxi trips 2019 that can provide insights into transportation patterns, customer behavior, and may contribute to urban planning considerations. Look for trends and patterns based on time of day, day of the week, and season.

Create visualizations (e.g., time series plots, histograms) to gain insights into the data's characteristics.

Data Collection:

Description of the Yellow Taxi dataset: These records are generated from the trip record submissions made by yellow taxi Technology Service Providers (TSPs). Each row represents a single trip in a yellow taxi. The trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off taxi zone locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

Source of the data.

Table ID https://www.kaggle.com/datasets/microize/newyork-yellow-taxi-trip-data-2020-2019?select=yellow_tripdata_2019-01.csv

Using the AWS EC2 user, the date is downloaded from Kaggle, uploaded into the aws bucket and then unzipped. (see codes appendix A. Download codes)

Explanation of data features and variables.

Number of rows	84,598,433
Total logical bytes	15.09 GB

Some fields are coded to show number such as he RatecodeID which is the rate effect in the end of the trip. These are numbers from 1 to 6 starting with the standard fare, airport, negotiated fare or group fare. Payment_type is another field that is coded with numbers from 1 to 6 and those represent credit card cash and so on.

Data Cleaning and Preprocessing:

- Identify and handle missing values.
- Remove duplicates.
- Convert data types if necessary.
- Outlier detection and treatment.
- Any data transformation performed.

Exploratory Data Analysis (EDA):

- The exploratory data codes shows some statistical summaries and visualizations. The description function displayed mean, median, mode Range, variance, standard deviation. That indicated that the data was not cleaned. It contained errors related to the date, amount, duplications or null. Notice below the data from other years and duplications. As well it showed other data that were not within the range of acceptable measurement for specific attributes. For example the statistics showed negative values for distance or amount paid.

See codes in the appendix A Exploratory Data Analysis

```
>>> df.describe()
       VendorID            tpep_pickup_datetime           tpep_dropoff_datetime  passenger_count  trip_distance
count   7019375                         7019375                         7019375          7019375        7019375
mean          2  2019-02-14 21:55:19.563049216  2019-02-14 22:12:27.681014272                2              3
min           1             2008-12-31 06:57:04             2008-12-31 07:25:04                0              0
25%           1             2019-02-07 19:39:18             2019-02-07 19:55:11                1              1
50%           2             2019-02-14 17:06:57             2019-02-14 17:27:00                1              2
75%           2             2019-02-22 04:34:06             2019-02-22 05:09:43                2              3
max           4             2038-02-17 21:55:54             2038-02-18 21:13:21                9            702
std           1                             NaN                             NaN                1              4
```

```
>>> print(results)
              count    min    max   mean
tolls_amount
-26.26            1  29.00  29.00  29.00
-26.13            1   0.00   0.00   0.00
-25.50            1   0.34   0.34   0.34
-18.26            1   0.33   0.33   0.33
-11.52            2   0.00  20.26  10.13
...             ...    ...    ...    ...
 500.05           1  10.50  10.50  10.50
 593.28           1  13.60  13.60  13.60
 765.76           2   4.10  10.50   7.30
 766.66           1   3.20   3.20   3.20
 771.52           1   9.50   9.50   9.50

[933 rows x 4 columns]
>>> results = df.groupby('fare_amount').trip
>>> print(results)
             count    min    max   mean
fare_amount
-400.00          1   0.00   0.00   0.00
-270.00          1   0.00   0.00   0.00
-250.00          1   0.00   0.00   0.00
-235.00          1   0.99   0.99   0.99
-215.32          1   0.00   0.00   0.00
...            ...    ...    ...    ...
 1196.35         1   0.00   0.00   0.00
 1256.00         1   0.00   0.00   0.00
 17669.73        1   0.00   0.00   0.00
 90000.00        1   0.00   0.00   0.00
 671123.14       1  10.60  10.60  10.60

[6263 rows x 4 columns]
```

Data Processing Pipeline:

In this section, we detail the steps taken to process the Yellow Taxi dataset.

5.1 Data Cleaning:

As Identified in the exploratory analysis, first will remove the missing values, negative values, using imputation techniques for the following attributes

pickup_datetime >=01/01/2019

pickup_datetime <01/01/2020

dropoff_datetime - pickup_datetime > 0

passenger_count >0

trip_distance >0

tip_amount >=0

tolls_amount >=0

mta_tax >=0

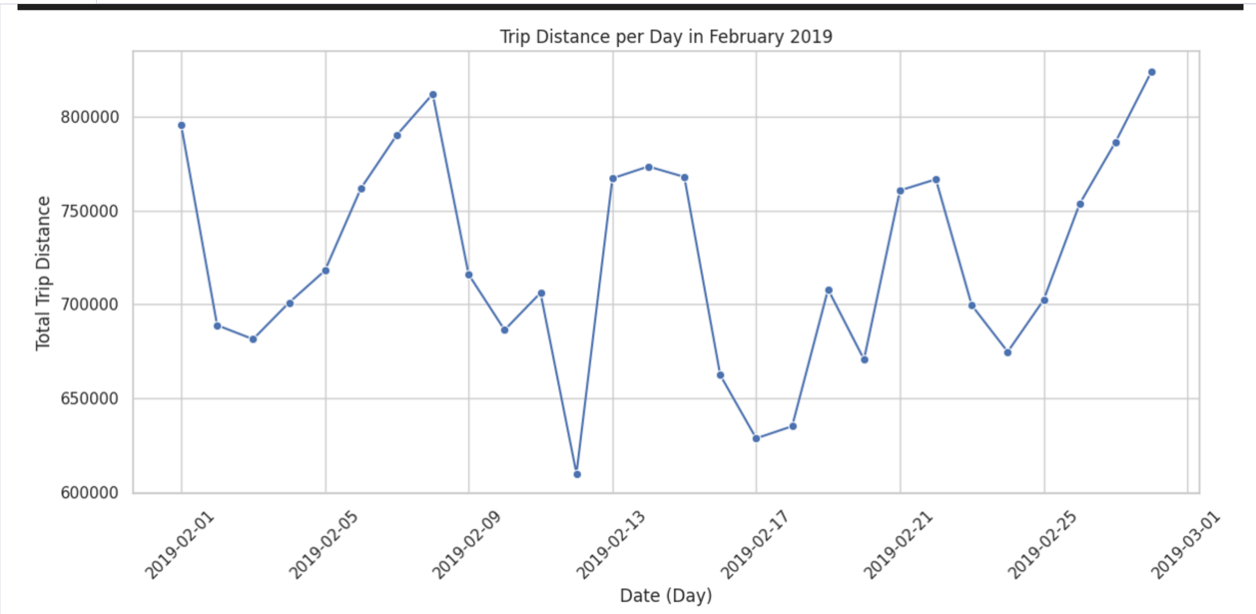
fare_amount >0

total_amount >0

See codes in the codes appendix. [*Cleaning Codes*]

Removed duplicate records to ensure data integrity.

All the cleaned files are saved in the new folder called raw. output_file_path = 's3://my-data-bucket-rp/raw/ See codes in Appendix A. Following graph shows that data does not have negative or null values.



5.2 Feature Engineering:

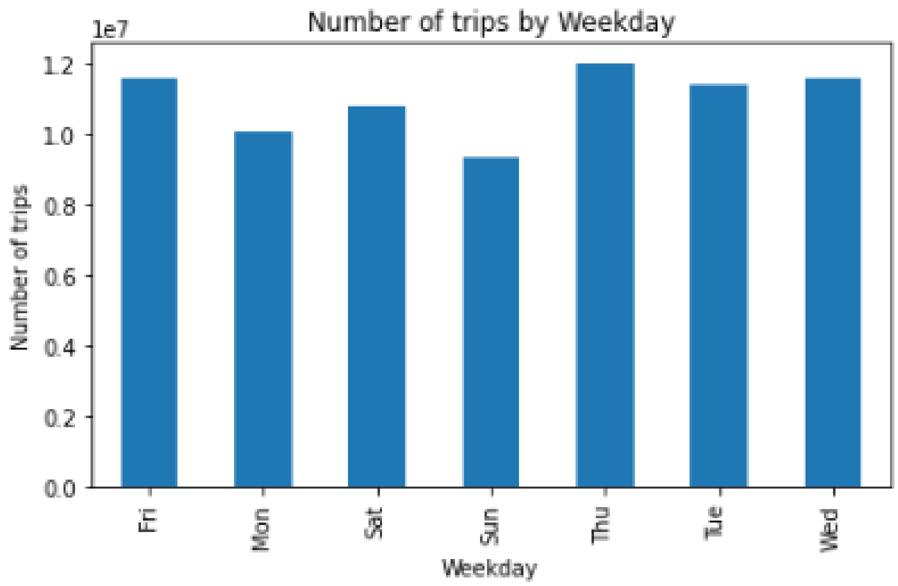
A new feature created is the percentage of the tip a mount in a a new column. As well created the new column with year, month and weekday etc.
[Codes for machine learning with parquet]

```
--------+-----------+-----------+------------+----------------+----------------+--------------+
der_date|pickup_date|pickup_year|pickup_month|pickup_yearmonth|pickup_dayofweek|pickup_weekend|
--------+-----------+-----------+------------+----------------+----------------+--------------+
19-05-01| 2019-05-01|       2019|           5|         2019-05|             Wed|           0.0|
19-05-01| 2019-05-01|       2019|           5|         2019-05|             Wed|           0.0|
19-05-01| 2019-05-01|       2019|           5|         2019-05|             Wed|           0.0|
19-05-01| 2019-05-01|       2019|           5|         2019-05|             Wed|           0.0|
19-05-01| 2019-05-01|       2019|           5|         2019-05|             Wed|           0.0|
19-05-01| 2019-05-01|       2019|           5|         2019-05|             Wed|           0.0|
```

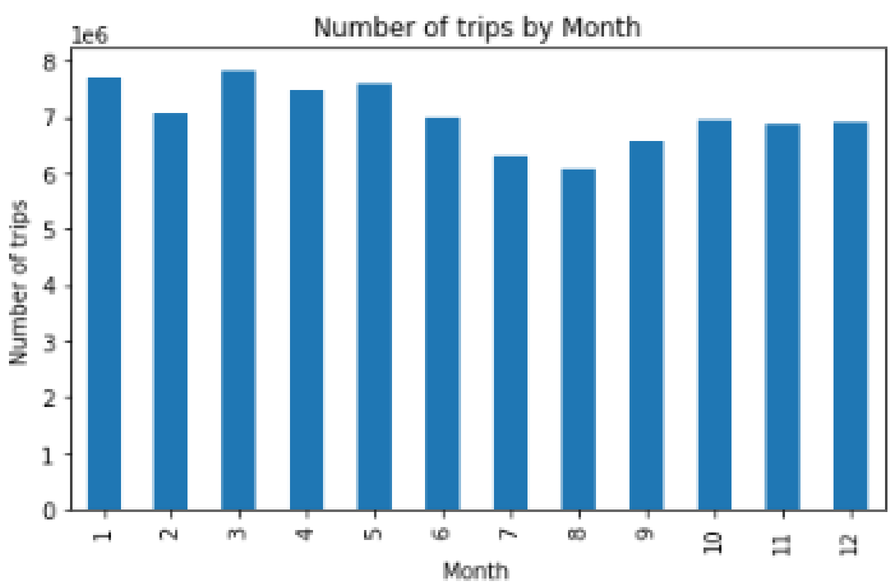
5.3 Aggregation and Transformation:

- Aggregated data on a monthly as well as weekday basis for trend analysis, shows that

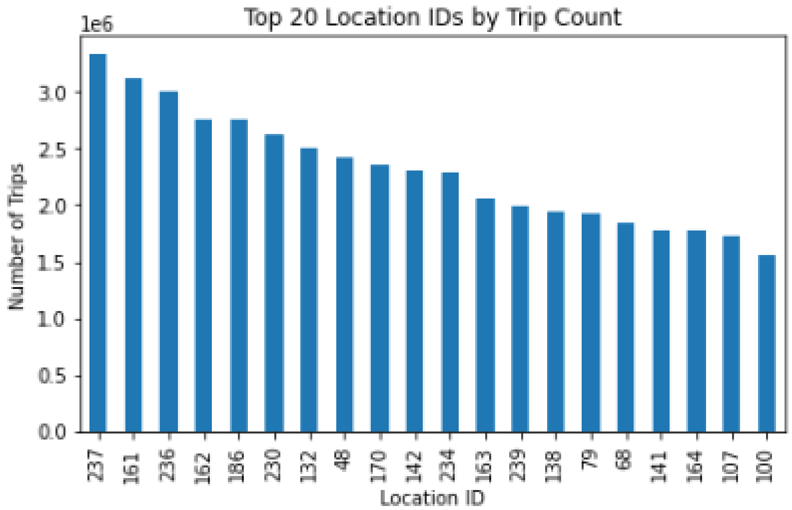
•

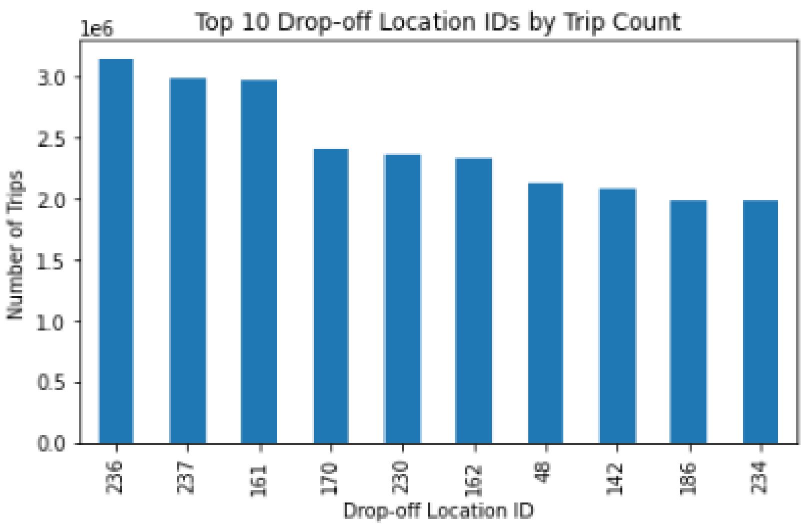


Number of trips by Weekday
1e7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Number of trips
Fri
Mon
Sat
Sun
Thu
Tue
Wed
Weekday

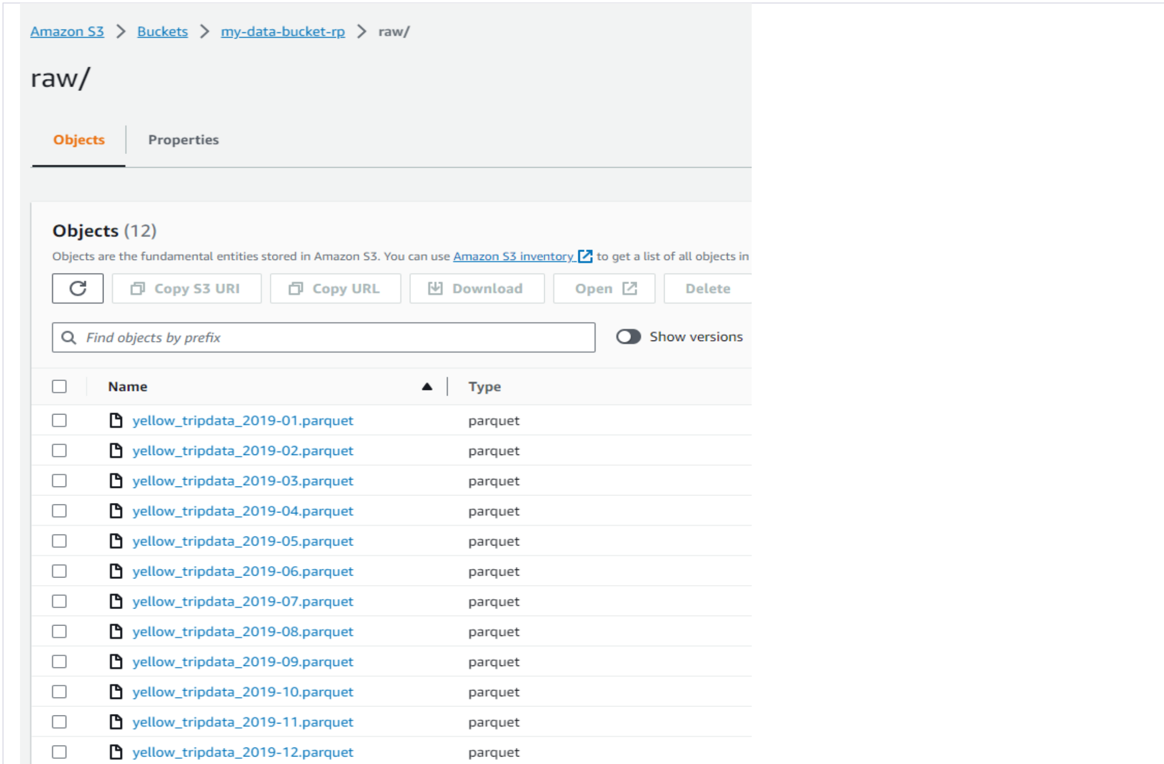


Number of trips by Month
1e6
8
7
6
5
4
3
2
1
0
Number of trips
1
2
3
4
5
6
7
8
9
10
11
12
Month





The cleaned dataset ready for analysis is saved in the raw folder.



Modeling:

1. Machine learning models applied are logistic regression and linear regression.

The machine learning has two datasets created by splitting the original data in training and testing specifically in ratio of 0.7 and 0.3. First, I created a label for the tip amount percentage by dividing the tip by total amount. For the logistic regression I used two attributes "trip_distance", "PULocationID" to predict whether a threshold of a certain tip amount will happen or not. I started with 30% tip, 0.3. The predicted probability for this to happen results 0.726.

I changed the threshold for defining the "label" column to **0.2** instead of **0.3**, and the area under ROC decreased from 0.726 to 0.531. Then I changed it again to **0.4** and the result increased, it changed to 0.779. The more I increase the threshold the more accurate is able to predict it.

Results and insights from the modeling phase.

```
Area Under ROC: 0.7260227566966024
```

```
+----------+-----------+--------------------+--------------------+-------------+
|numClasses|numFeatures|     interceptVector|   coefficientMatrix|isMultinomial|
+----------+-----------+--------------------+--------------------+-------------+
|         2|          2|[-0.7426170050605...|-0.32799128736165...|        false|
+----------+-----------+--------------------+--------------------+-------------+
```

```
------------+-----+-------------+--------------------+--------------------+----------+
airport_fee|label|     features|       rawPrediction|         probability|prediction|
------------+-----+-------------+--------------------+--------------------+----------+
       null|  0.0| [6.6,107.0]|[2.83080493359098...|[0.94431794188089...|       0.0|
       null|  0.0| [1.0,234.0]|[0.90318989125193...|[0.71160458489341...|       0.0|
       null|  0.0| [4.8,164.0]|[2.19963921092673...|[0.90021710722390...|       0.0|
       null|  0.0| [0.7,170.0]|[0.85058215322676...|[0.70068924813821...|       0.0|
       null|  0.0| [2.2,226.0]|[1.30250314210883...|[0.78625595634036...|       0.0|
       null|  1.0| [5.0,230.0]|[2.21801689371001...|[0.90185580700496...|       0.0|
       null|  0.0|  [1.1,79.0]|[1.04688582418209...|[0.74017644369999...|       0.0|
       null|  0.0|[18.0,132.0]|[6.55201902819223...|[0.99857480301785...|       0.0|
       null|  1.0| [1.5,230.0]|[1.07004738794422...|[0.74460592759967...|       0.0|
       null|  0.0| [3.1,186.0]|[1.62631383084890...|[0.83566404461185...|       0.0|
       null|  0.0|[17.1,132.0]|[6.25682686956674...|[0.99808634902045...|       0.0|
       null|  1.0| [1.9,164.0]|[1.24846447757793...|[0.77703394171660...|       0.0|
       null|  1.0| [1.2,233.0]|[0.96950361197713...|[0.72502054607213...|       0.0|
       null|  1.0| [0.9,170.0]|[0.91618041069909...|[0.71426319922795...|       0.0|
       null|  0.0| [1.6,179.0]|[1.13933514257647...|[0.75755754946858...|       0.0|
       null|  0.0| [1.1,138.0]|[1.00467349226309...|[0.73197645014420...|       0.0|
       null|  0.0|  [0.0,74.0]|[0.68967272434859...|[0.66589411869952...|       0.0|
       null|  0.0| [1.0,230.0]|[0.90605174426339...|[0.71219154852915...|       0.0|
```

Root Mean Squared Error (RMSE): 0.397480744358655

R-squared (R2): 0.047508766833290084

RMSE of 2.35 means that, on average, the model's predictions are off by approximately 2.35 units in the same unit as your target variable (tip_amount)

2. In linear regression an R-squared value of 0.31 indicates that approximately 31.25% of the variance in the tip_amount can be explained by the features (trip_distance, RatecodeID, fare_amount)

   The model has a moderate level of predictive power, as indicated by the R-squared value, and the RMSE provides an average measure of the model's prediction accuracy.

3. I changed the model to use different attributes such as 'trip_distance', 'PULocationID', 'fare_amount' and the result shows that the RMSE is lower. These two attributes can predict a better tip amount be more than 30%. However, the result suggests that these two attributes seems to have no impact into the amount of the tip.

4. Correlation Matrix

   The following matrix shows correlation between the fields. This graph suggests that tip amount and payment type has some correlation -0.45. The smaller the payment type the bigger tip. The smallest number is credit card and the second is cash. As well tip amount and trip distance relationship has a weak correlation 0.24, the longer the distance, the bigger the tip. All Other fields seem to have no correlation. (see codes in Apprenix A Correlation)

Conclusions:

1. From linear regression R-squared value of 0.31 indicates that approximately 31.25% of the variance in the tip_amount can be explained by the features (trip_distance, RatecodeID, fare_amount)

2. The model has a moderate level of predictive power, as indicated by the R-squared value, and the RMSE provides an average measure of the model's prediction accuracy.

3. Correlation Matrix

   The the matrix sugesst a moderate correlation between the tip amount and payment type -0.45. The smaller the number that shows payment type the bigger tip, which corresponds to credit card. As well tip amount and trip distance relationship has a weak correlation of 0.24, the longer the distance, the bigger the tip. All the other fields seem to have no correlation with each other.

4. The other conclusion is that January, March and May have the highest number of trips of the yellow taxi and August has the lowest traffic.

5. The area that taxi trip is requested the most is Upper East Side South and Upper East Side North in Manhattan.

6. On the weekdays Yellow Taxi is used most is Thursday and Friday, while the lowest day is Sunday.

GitHub Repository:

https://github.com/RobertPlumbi/Yellow-Taxi-/blob/main/cis_4130_project_milestone_7_Plumbi_Robert.pdf

Code Citations:

Professor Holowczak notes
Holowczak, R. (n.d.). *Lecturer Notes for CIS4130*. Cuny Baruch College. https://bbhosted.cuny.edu/webapps/blackboard/execute/content/file?cmd=view&content_id=_80423034_1&course_id=_2269208_1

Acknowledgments:

Guidance received from Professor Richard Holowczak.

I am thankful for the time and effort invested by Professor Holowczak in providing instructions in every step, encouragement, and valuable suggestions that have greatly contributed to this project. I am grateful for the opportunity to learn under his mentorship and am confident that the skills acquired during this project will have a lasting impact on my academic and professional journey.

Once again, thank you, Professor Holowczak, for your dedication and support.

References:

Mohanasundaram, S. (2022, May 2). *Newyork Taxi Trip Data*. Kaggle. https://www.kaggle.com/datasets/microize/newyork-yellow-taxi-trip-data-2020-2019

Holowczak, R. (n.d.). *Lecturer Notes for CIS4130*. Cuny Baruch College. https://bbhosted.cuny.edu/webapps/blackboard/execute/content/file?cmd=view&content_id=_80423034_1&course_id=_2269208_1

Holowczak, R. (n.d.). *Lecturer Notes for CIS4130*. Cuny Baruch College. https://bbhosted.cuny.edu/webapps/blackboard/execute/content/file?cmd=view&content_id=_80141530_1&course_id=_2269208_1

Appendix A

Downloading and saving codes

1. kaggle datasets download -d microize/newyork-yellow-taxi-trip-data-2020-2019

2. aws s3 cp ~/newyork-yellow-taxi-trip-data-2020-2019 s3://my-data-bucket-rp/

3. aws s3 cp s3://my-data-bucket-rp/newyork-yellow-taxi-trip-data-2020-2019.zip .

4. unzip newyork-yellow-taxi-trip-data-2020-2019.zip
   aws s3 cp yellow_tripdata_2019-02.csv s3://my-data-bucket-rp/landing/yellow_tripdata_2019-02.csv

Exploratory Data Analysis

```
import boto3

s3 = boto3.resource('s3')

for bucket in s3.buckets.all():

   print(bucket.name)

import pandas as pd

s3 = boto3.client('s3')

df = pd.read_csv('s3://my-data-bucket-rp/landing/yellow_tripdata_2019-02.csv')

df.dtypes

df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

df['tpep_dropoff_datetime'] = pd.to_datetime(df['tpep_dropoff_datetime'])

df_feb_2019 = df[(df['tpep_pickup_datetime'].dt.year == 2019) &
(df['tpep_pickup_datetime'].dt.month == 2)]

pd.options.display.float_format = '{:.0f}'.format

df_feb_2019.describe()

df = pd.read_csv('s3://my-data-bucket-rp/landing/yellow_tripdata_2019-02.csv')

results = df.groupby('PULocationID').trip_distance.agg(['count', 'min', 'max', 'mean'])

print(results)
```

```
results = df.groupby('tolls_amount').trip_distance.agg(['count', 'min', 'max', 'mean'])
print(results)
```

saving the results

```
results.to_csv('s3://my-data-bucket-rp/trip_analysis.csv')
```

pip install --upgrade s3fs

Following code shows the plot of the distance rider initiated from a random location ID=123.

```
import boto3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
location_id_to_plot = 123
filtered_data = df[df['PULocationID'] == location_id_to_plot]
filtered_data['tpep_pickup_datetime'] =
pd.to_datetime(filtered_data['tpep_pickup_datetime'])
filtered_data.set_index('tpep_pickup_datetime', inplace=True)
daily_trip_distance = filtered_data['trip_distance'].resample('D').sum()
sns.set(style='whitegrid')
plt.figure(figsize=(12, 6))  # Optional: Set the figure size
sns.lineplot(data=daily_trip_distance, marker='o', palette='tab10')
plt.title(f'Trip Distance per Day for PULocationID {location_id_to_plot}')
plt.xlabel('Date')
plt.ylabel('Total Trip Distance')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('trip_distance_per_day.png')
s3.upload_file('trip_distance_per_day.png', bucket_name, object_name)
```

Following code is used to plot the entire dataset trip distances by each day during February 2019.

```
import boto3

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

s3_path = 's3://my-data-bucket-rp/landing/yellow_tripdata_2019-02.csv'

df = pd.read_csv(s3_path)

df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

df_feb_2019 = df[(df['tpep_pickup_datetime'].dt.year == 2019) &
(df['tpep_pickup_datetime'].dt.month == 2)]

# Group data by day and sum trip distances

daily_trip_distance =
df_feb_2019.groupby(df_feb_2019['tpep_pickup_datetime'].dt.date)['trip_distance'].sum()

sns.set(style='whitegrid')

plt.figure(figsize=(12, 6))

plot = sns.lineplot(data=daily_trip_distance, marker='o', palette='tab10')

# Customize the x-axis labels to display only the day

plot.xaxis.set_major_formatter(plt.FixedFormatter(daily_trip_distance.index.strftime("%
d")))

plt.title('Trip Distance per Day in February 2019')

plt.xlabel('Date (Day)')

plt.ylabel('Total Trip Distance')

plt.xticks(rotation=45)  # Rotate x-axis labels for better readability

plt.tight_layout()

plt.savefig('trip_distance_per_day.png')

bucket_name = 'my-data-bucket-rp'

object_name = 'path/to/sot.png'

s3 = boto3.client('s3')

s3.upload_file('trip_distance_per_day.png', bucket_name, object_name)
```

Cleaning Codes

```
import boto3
```

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import io

s3_path = 's3://my-data-bucket-rp/landing/yellow_tripdata_2019-02.csv'

df = pd.read_csv(s3_path)

df = (df['tpep_pickup_datetime'] >= '2019-03-01') & (df['tpep_pickup_datetime'] < '2019-04-01') &  (df[passenger_count] >0 & df[trip_distance] >0 & df[tip_amount] >=0 & df[tolls_amount] >=0 & df[mta_tax] >=0 & df[ fare_amount] >0  & df[ total_amount] >=0

dropoff_datetime - pickup_datetime > 0

df = df[mask]

csv_buffer = io.StringIO()

df.to_csv(csv_buffer, index=False)

bucket_name = 'my-data-bucket-rp'

file_key = 'cleaned/mar_2019.csv'

s3.upload_fileobj(io.BytesIO(csv_buffer.getvalue().encode()), bucket_name, file_key)
```

Using Spark in DataBrick environment

```
spark

import os

import pandas as pd

access_key = '**********'

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")

aws_region = "us-east-2"

sc._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)
```

```python
sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)

sc._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3." + aws_region +

".amazonaws.com")

file_path = 's3://my-data-bucket-rp/landing/yellow_tripdata_2019-05.csv'


sdf = spark.read.csv(file_path, sep='\t', header=True, inferSchema=True)

sdf.count()
```

```python
from pyspark.sql import SparkSession

spark = SparkSession \

    .builder \

    .appName("Python Spark SQL basic example") \

    .config("spark.some.config.option", "some-value") \

    .getOrCreate()

df = spark.read.csv('s3a://my-data-bucket-
rp/landing/yellow_tripdata_2019-05.csv', header=True)

df.show(5)

date_mask = (col('tpep_pickup_datetime') >= '2019-05-01') &
(col('tpep_pickup_datetime') < '2019-06-01')

numeric_mask = (col('passenger_count') > 0) & (col('trip_distance') > 0)
& (col('tip_amount') >= 0) & (col('tolls_amount') >= 0) &
(col('mta_tax') >= 0) & (col('fare_amount') > 0) & (col('total_amount')
> 0)

filtered_df = df.filter(date_mask & numeric_mask)

filtered_df.coalesce(1).write.csv('s3a://my-data-bucket-
rp/cleaned/yellow_tripdata_2019-05.csv', header=True, mode='overwrite')
```

Convert CSV files into parquet files

```python
import pandas as pd

import boto3

import pyarrow as pa
```

```
import pyarrow.parquet as pq

s3 = boto3.client("s3", region_name='us-east-2',
aws_access_key_id=' ***************',

aws_secret_access_key='****************')

bucket_name = 'my-data-bucket-rp'

key = 'cleaned/taxi-12.csv'

obj = s3.get_object(Bucket=bucket_name, Key=key)

df = pd.read_csv(obj['Body'])

table = pa.Table.from_pandas(df)

parquet_path= 's3://my-data-bucket-rp/raw/yellow_tripdata_2019-
12'

pq.write_to_dataset(table=table, root_path=parquet_path)
```

Codes for machine learning with parquet

```
spark

import os

import boto3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from pyspark.sql.functions import col, isnan, when, count, udf, to_date, year,
month, date_format, size, split

from pyspark.ml.stat import Correlation

from pyspark.ml.feature import VectorAssembler

from pyspark.sql.functions import to_date


df = df.withColumn('pickup_date', to_date(df['tpep_pickup_datetime'], 'yyyy-
MM-dd'))

df = df.withColumn("pickup_year", year(col("order_date")))

df = df.withColumn("pickup_month", month(col("order_date")))

df = df.withColumn("pickup_yearmonth", date_format(col("order_date"), "yyyy-
MM"))
```

```
df = df.withColumn("pickup_dayofweek", date_format(col("order_date"), "E"))

df = df.withColumn("pickup_weekend", when(df.pickup_dayofweek ==
'Saturday',1.0).when(df.pickup_dayofweek == 'Sunday', 1.0).otherwise(0))

df.show()
```

Logistic Regression Estimator

```
import os

from pyspark.sql import SparkSession

from pyspark.ml.feature import Imputer, VectorAssembler, StringIndexer

from pyspark.ml.classification import LogisticRegression

from pyspark.ml import Pipeline

from pyspark.ml.evaluation import BinaryClassificationEvaluator

access_key = '********************'

secret_key = ‘***********************'

os.environ['AWS_ACCESS_KEY_ID'] = access_key

os.environ['AWS_SECRET_ACCESS_KEY'] = secret_key

aws_region = "us-east-2"

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")

spark = SparkSession.builder.appName("LogisticRegressionExample").getOrCreate()

from pyspark.sql.functions import col, count, when

file_path_parquet = 's3://my-data-bucket-rp/raw/yellow_tripdata_2019-05.parquet'

sdf = spark.read.parquet(file_path_parquet)

sdf = sdf.withColumn("label", when(sdf.tip_amount / sdf.fare_amount >= 0.3,
1.0).otherwise(0.0))

trainingData, testData = sdf.randomSplit([0.7, 0.3], seed=3456)
```

```
assembler = VectorAssembler(inputCols=["trip_distance", "PULocationID"],
outputCol="features")

lr = LogisticRegression(featuresCol="features", labelCol="label")

pipeline = Pipeline(stages=[assembler, lr])

model = pipeline.fit(trainingData)

predictions = model.transform(testData)

evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")

auc = evaluator.evaluate(predictions)

print(f"Area Under ROC: {auc}")
```

```
# Save the best model

model_path = "Taxi_logistic_regression_model"

bestModel.write().overwrite().save(s3://my-data-bucket-rp/model/first)
```

Linear Regression model

```
from pyspark.ml.regression import LinearRegression

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("LinearRegressionExample").getOrCreate()

selected_columns = ['trip_distance', 'RatecodeID', 'fare_amount', 'tip_amount']

file_path_parquet = 's3://my-data-bucket-rp/raw/yellow_tripdata_2019-05.parquet'

df = spark.read.parquet(file_path_parquet).select(selected_columns)

df = df.withColumn("label", when(df.tip_amount / df.fare_amount >= 0.3,
1.0).otherwise(0.0))

assembler = VectorAssembler(inputCols=['trip_distance', 'RatecodeID', 'fare_amount'],
outputCol='features', handleInvalid="skip")

df = assembler.transform(df)

(trainingData, testData) = df.randomSplit([0.7, 0.3], seed=42)
```

```python
lr = LinearRegression(featuresCol='features', labelCol='tip_amount')

lr_model = lr.fit(trainingData)

predictions = lr_model.transform(testData)

evaluator = RegressionEvaluator(labelCol="tip_amount", predictionCol="prediction",
metricName="rmse")

rmse = evaluator.evaluate(predictions)

print(f"Root Mean Squared Error (RMSE): {rmse}")

evaluator_r2 = RegressionEvaluator(labelCol="tip_amount", predictionCol="prediction",
metricName="r2")

r2 = evaluator_r2.evaluate(predictions)

print(f"R-squared (R2): {r2}")
```

```python
# Assemble features into a single column

assembler = VectorAssembler(inputCols=['trip_distance', 'PULocationID',
'fare_amount'], outputCol='features', handleInvalid="skip")

df = assembler.transform(df)


# Split the data into training and test sets

(trainingData, testData) = df.randomSplit([0.7, 0.3], seed=42)


# Create a Linear Regression model

lr = LinearRegression(featuresCol='features', labelCol='label')


# Fit the model

lr_model = lr.fit(trainingData)


# Make predictions on the test set

predictions = lr_model.transform(testData)
```

```python
# Evaluate the model

evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction",
metricName="rmse")

rmse = evaluator.evaluate(predictions)

print(f"Root Mean Squared Error (RMSE): {rmse}")


evaluator_r2 = RegressionEvaluator(labelCol="label", predictionCol="prediction",
metricName="r2")

r2 = evaluator_r2.evaluate(predictions)

print(f"R-squared (R2): {r2}")
```

Vizualization

```python
import os

from pyspark.sql import SparkSession

from pyspark.ml.feature import Imputer, VectorAssembler, StringIndexer

from pyspark.ml.classification import LogisticRegression

from pyspark.ml import Pipeline

from pyspark.ml.evaluation import BinaryClassificationEvaluator

from pyspark.sql.functions import col

# Set AWS credentials

access_key = ****************

secret_key = **********************************

os.environ['AWS_ACCESS_KEY_ID'] = access_key

os.environ['AWS_SECRET_ACCESS_KEY'] = secret_key

aws_region = "us-east-2"

sc._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)

sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)
```

```python
sc._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3." + aws_region +
".amazonaws.com")

# Create a Spark session

spark = SparkSession.builder.appName("TaxiDataProcessing").getOrCreate()


from pyspark.sql import SparkSession

from pyspark.sql.functions import col

parquet_file_path = 's3://my-data-bucket-rp/raw/'


selected_columns = [

    "VendorID",

    "tpep_pickup_datetime",

    "tpep_dropoff_datetime",

    "trip_distance",

    "PULocationID",

    "DOLocationID",

    "payment_type",

    "fare_amount",

    "tip_amount",

    "total_amount"

]

df = spark.read.parquet(parquet_file_path).select(selected_columns)

df.show()
```

```python
spark

import os

import boto3

import pandas as pd
```

```python
import matplotlib.pyplot as plt

import seaborn as sns

from pyspark.sql.functions import col, isnan, when, count, udf, to_date, year,
month, date_format, size, split

from pyspark.ml.stat import Correlation

from pyspark.ml.feature import VectorAssembler

from pyspark.sql.functions import to_date


df = df.withColumn('pickup_date', to_date(df['tpep_pickup_datetime'], 'yyyy-
MM-dd'))

df = df.withColumn("pickup_year", year(col("order_date")))

df = df.withColumn("pickup_month", month(col("order_date")))

df = df.withColumn("pickup_yearmonth", date_format(col("order_date"), "yyyy-
MM"))

df = df.withColumn("pickup_dayofweek", date_format(col("order_date"), "E"))

df = df.withColumn("pickup_weekend", when(df.pickup_dayofweek ==

'Saturday',1.0).when(df.pickup_dayofweek == 'Sunday', 1.0).otherwise(0))

df.show()
```

Matrix correlation between the fields

```python
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation
import pandas as pd

vector_column = "correlation_features"
numeric_columns = [
    "VendorID",
    "trip_distance",
    "PULocationID",
    "DOLocationID",
    "payment_type",
    "fare_amount",
    "tip_amount",
    "total_amount"
```

```
]
assembler = VectorAssembler(inputCols=numeric_columns,
outputCol=vector_column)
cdf_vector = assembler.transform(df).select(vector_column)
matrix = Correlation.corr(cdf_vector, vector_column).collect()[0][0]
correlation_matrix = matrix.toArray().tolist()
correlation_matrix_df = pd.DataFrame(data=correlation_matrix,
columns=numeric_columns, index=numeric_columns)
sns.set(style="white")
plt.figure(figsize=(16, 5))
sns.heatmap(correlation_matrix_df, annot=True, cmap="coolwarm", fmt=".2f")
plt.savefig("correlation_matrix.png")
plt.show()
```

Count trips by Month.

```
import matplotlib.pyplot as plt

import seaborn as sns

import io

from pyspark.sql.functions import year, month

import boto3


df = df.withColumn("pickup_month", month(col("tpep_pickup_datetime")))

pldf = df.filter(col("pickup_year") == 2019).groupby("pickup_month") \

    .count().sort("pickup_month").toPandas()

myplot = pldf.plot.bar(x='pickup_month', y='count', legend=False)

myplot.set(xlabel='Month', ylabel='Number of trips')

myplot.set(title='Number of trips by Month')

myplot.figure.set_tight_layout(True)

myplot.get_figure().savefig("pickup_count_by_month.png")

img_data = io.BytesIO()

myplot.get_figure().savefig(img_data, format='png', bbox_inches='tight')

img_data.seek(0)
```

```
s3 = boto3.client('s3', aws_access_key_id=************************,
aws_secret_access_key=*************************************)

s3.upload_fileobj(img_data, 'my-data-bucket-rp',
'Reports/pickup_count_by_month.png')
```

Count trips by weekday.

```
weekday_counts =
df.groupBy("pickup_dayofweek").agg(count("*").alias("count")).sort("pickup_dayofweek"
).toPandas()

myplot = weekday_counts.plot.bar(x='pickup_dayofweek', y='count', legend=False)

myplot.set(xlabel='Weekday', ylabel='Number of trips')

myplot.set(title='Number of trips by Weekday')

myplot.figure.set_tight_layout(True)
```

Count trips and sort by pick up Location.

```
top_locations =
df.groupBy("PULocationID").agg(count("*").alias("trip_count")).sort(col("trip_
count").desc()).limit(20).toPandas()

myplot = top_locations.plot.bar(x='PULocationID', y='trip_count',
legend=False)

myplot.set(xlabel='Location ID', ylabel='Number of Trips')

myplot.set(title='Top 20 Location IDs by Trip Count')

myplot.figure.set_tight_layout(True)
```

Count trips and sort by drop off Location.

```
top_dropoff_locations =
df.groupBy("DOLocationID").agg(count("*").alias("trip_count")).sort(col("trip_
count").desc()).limit(10).toPandas()

myplot = top_dropoff_locations.plot.bar(x='DOLocationID', y='trip_count',
legend=False)

myplot.set(xlabel='Drop-off Location ID', ylabel='Number of Trips')
```

```
myplot.set(title='Top 10 Drop-off Location IDs by Trip Count')

myplot.figure.set_tight_layout(True)
```