NYC Taxi Insights: Analyzing Trip and Fare Data for Industry Optimization

Milestone 5: Problem definition

Group 11

Sanay Shah

Venkata Sai Prasad Aka

Abdul Mateen

857-318-6077 (Tel of Student 1)
857-260-8799 (Tel of Student 2)
857-381-1139 (Tel of Student 3)

shah.sana@northeastern.edu
aka.v@northeastern.edu
mateen.a@northeastern.edu

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Percentage of Effort Contributed by Student 3: 50%

Signature of Student 1: Sanay Shah

Signature of Student 2: Venkata Sai Prasad Aka

Signature of Student 3: Abdul Mateen

Submission Date: 03/31/2023

Problem Definition
The New York City taxi industry plays a vital role in the city's transportation system. However, with the rise of rideshare services and changing urban dynamics, it is essential to analyze taxi trip data to understand trends, optimize operations, and remain competitive in the market. The dataset provided contains both trip data and fare data, which we will analyze to gain insights into the taxi industry's performance and customer behavior.

Objective:
Our objective is to analyze the New York City taxi trip data and fare data to gain insights into the industry's trends, customer preferences, and areas of improvement. We aim to better understand the factors affecting the taxi industry's performance and identify opportunities for growth and optimization.

Dataset:

https://databank.illinois.edu/datasets/IDB-9610843#

The dataset contains two files for each trip: one with trip data and the other with fare data. A small sample of the New York City taxi fare data is provided, where each row corresponds to an occupied taxi trip. The dataset includes information such as medallion, hack_license, vendor_id, pickup_datetime, payment type, fare_amount, surcharge, mta_tax, tip_amount, tolls_amount, and total amount.

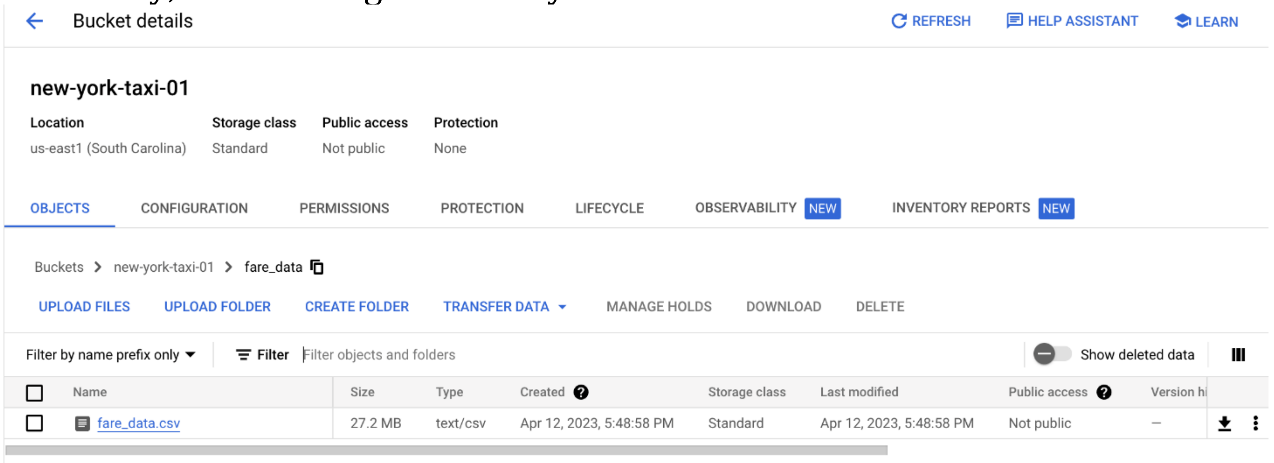
Analysis Goals:

1. Analyze the seasonality and trends in taxi trip demand, including patterns related to time of day, day of the week, and month.
2. Examine the relationship between trip distance, duration, and fare, aiming to identify any discrepancies or inefficiencies in pricing.
3. Investigate customer preferences in payment types and their impact on tip amounts and overall driver income.
4. Identify areas with high taxi demand and low supply, indicating opportunities for improved service and increased revenue.
5. Examine the performance of different taxi vendors, comparing operational efficiency and customer satisfaction to identify best practices and areas for improvement.

Cloud Storage:

GCP (Google Cloud Platform) Cloud Storage is a cloud-based object storage service provided by Google. It allows you to store and retrieve data from anywhere on the internet, at any time, with high availability and durability.
Cloud Storage is designed to be highly scalable, so you can store and access vast amounts of data without worrying about capacity planning or infrastructure maintenance. The service is optimized for both high-performance and cost-efficiency, offering different storage classes to suit your specific needs.

Overall, GCP Cloud Storage is a powerful and flexible storage solution that can help you to manage and store your data securely, cost-effectively, and with high reliability.

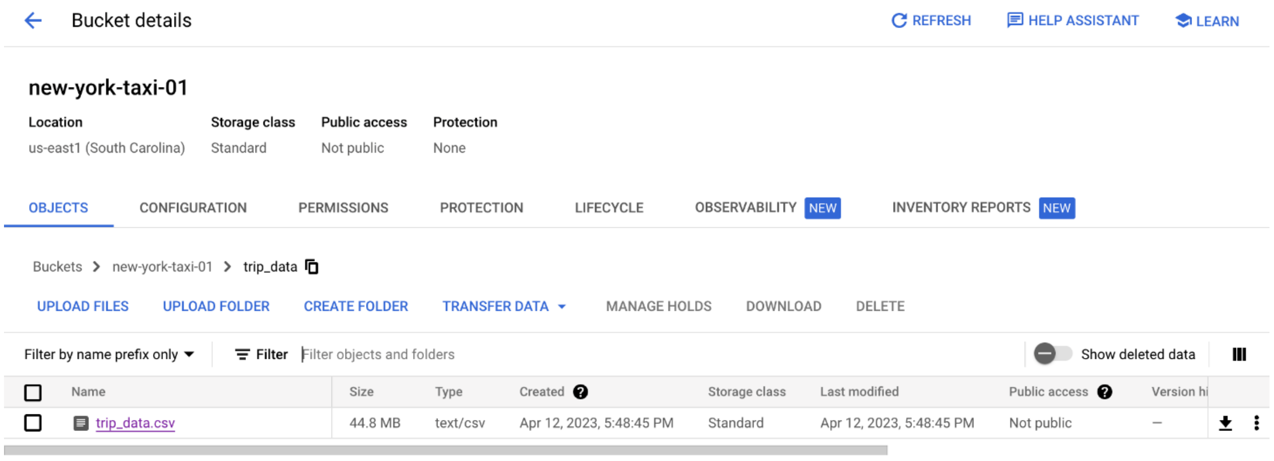


new-york-taxi-01

Location	Storage class	Public access	Protection
us-east1 (South Carolina)	Standard	Not public	None

Buckets > new-york-taxi-01 > fare_data

Name	Size	Type	Created	Storage class	Last modified	Public access	Version hi
fare_data.csv	27.2 MB	text/csv	Apr 12, 2023, 5:48:58 PM	Standard	Apr 12, 2023, 5:48:58 PM	Not public	–

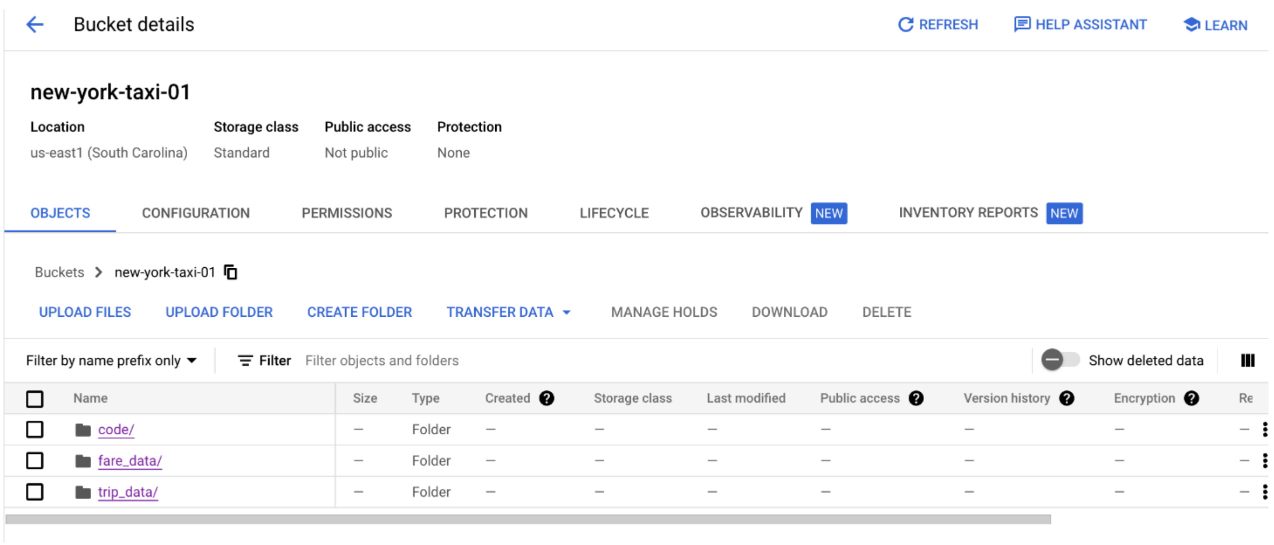


new-york-taxi-01

Location	Storage class	Public access	Protection
us-east1 (South Carolina)	Standard	Not public	None

Buckets > new-york-taxi-01 > trip_data

Name	Size	Type	Created	Storage class	Last modified	Public access	Version hi
trip_data.csv	44.8 MB	text/csv	Apr 12, 2023, 5:48:45 PM	Standard	Apr 12, 2023, 5:48:45 PM	Not public	–


Bucket details

REFRESH HELP ASSISTANT LEARN

new-york-taxi-01

Location	Storage class	Public access	Protection
us-east1 (South Carolina)	Standard	Not public	None

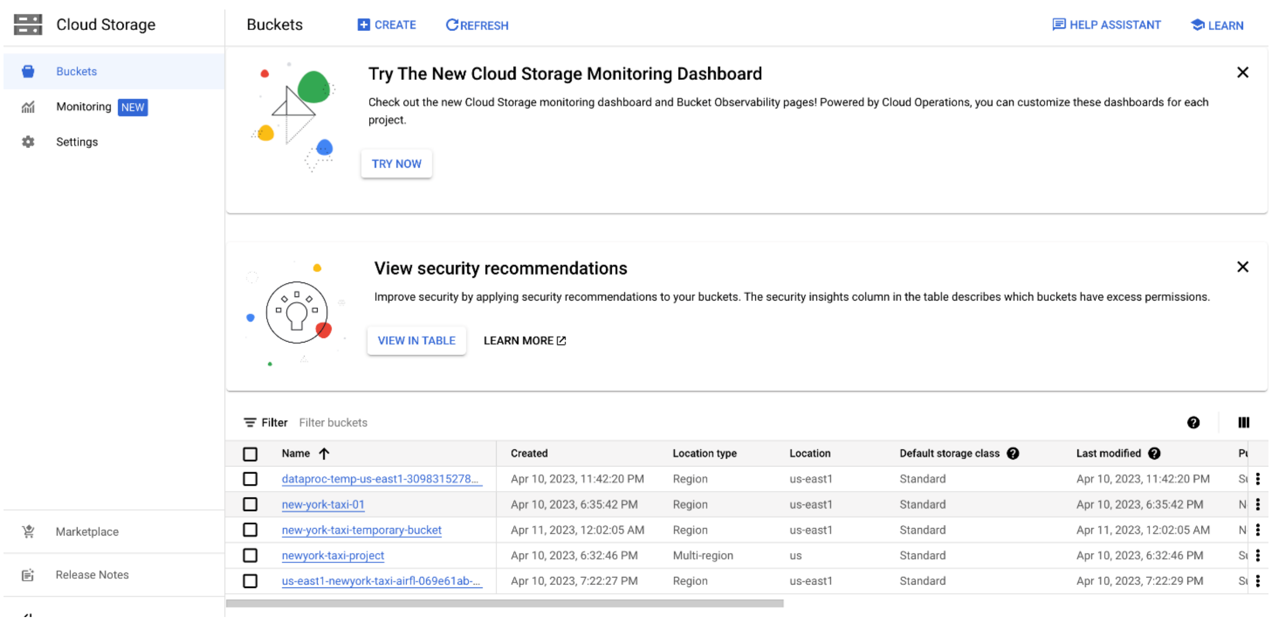
OBJECTS CONFIGURATION PERMISSIONS PROTECTION LIFECYCLE OBSERVABILITY NEW INVENTORY REPORTS NEW

Buckets > new-york-taxi-01

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA MANAGE HOLDS DOWNLOAD DELETE

Filter by name prefix only Filter Filter objects and folders Show deleted data

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Re
code/	—	Folder	—	—	—	—	—	—	—
fare_data/	—	Folder	—	—	—	—	—	—	—
trip_data/	—	Folder	—	—	—	—	—	—	—


Cloud Storage

Buckets CREATE REFRESH HELP ASSISTANT LEARN

Buckets
Monitoring NEW
Settings

Try The New Cloud Storage Monitoring Dashboard

Check out the new Cloud Storage monitoring dashboard and Bucket Observability pages! Powered by Cloud Operations, you can customize these dashboards for each project.

TRY NOW

View security recommendations

Improve security by applying security recommendations to your buckets. The security insights column in the table describes which buckets have excess permissions.

VIEW IN TABLE LEARN MORE

Filter Filter buckets

Name	Created	Location type	Location	Default storage class	Last modified	Pu
dataproc-temp-us-east1-3098315278...	Apr 10, 2023, 11:42:20 PM	Region	us-east1	Standard	Apr 10, 2023, 11:42:20 PM	S
new-york-taxi-01	Apr 10, 2023, 6:35:42 PM	Region	us-east1	Standard	Apr 10, 2023, 6:35:42 PM	N
new-york-taxi-temporary-bucket	Apr 11, 2023, 12:02:05 AM	Region	us-east1	Standard	Apr 11, 2023, 12:02:05 AM	N
newyork-taxi-project	Apr 10, 2023, 6:32:46 PM	Multi-region	us	Standard	Apr 10, 2023, 6:32:46 PM	S
us-east1-newyork-taxi-airfl-069e61ab-...	Apr 10, 2023, 7:22:27 PM	Region	us-east1	Standard	Apr 10, 2023, 7:22:29 PM	S

Marketplace
Release Notes

We have two different data sources which contains Newyork Fare data and Newyork Trip data.

Data Wrangling:

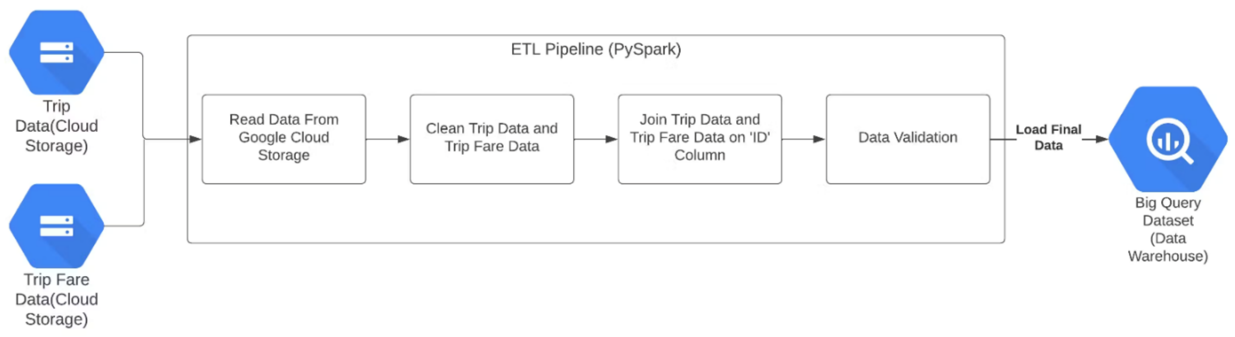
Before uploading the data into GCP Cloud Storage, we did few data wrangling like creating trip_id column, and data cleaning is a crucial step in data analysis as it ensures the accuracy and quality of the data. Handling null values is one of the critical aspects of data cleaning as null values can lead to inaccurate analysis and conclusions.

After data wragling, created a GCP project and cloud storage bucket and uploaded tripdata and faredata and set permissions to access the bucket.

BigQuery:

BigQuery is a powerful data warehousing solution that provides organizations with the ability to store, manage, and analyze massive volumes of data quickly and efficiently in a cost-effective way.

Below is the ETL pipeline,


Trip Data(Cloud Storage)
Trip Fare Data(Cloud Storage)
ETL Pipeline (PySpark)
Read Data From Google Cloud Storage
Clean Trip Data and Trip Fare Data
Join Trip Data and Trip Fare Data on 'ID' Column
Data Validation
Load Final Data
Big Query Dataset (Data Warehouse)

Dataproc:

Google Dataproc is a fully managed cloud-based service from Google Cloud Platform that provides a highly scalable and cost-effective solution for running Apache Hadoop, Apache Spark, and other big data processing frameworks. It simplifies the process of deploying, configuring, and managing big data clusters by automating many of the time-consuming tasks involved in setting up and maintaining a Hadoop or Spark cluster. After uploading the data sources in cloud Storage, extracted data from cloud storage. Now cleaned the trip data and fare data by removing leading white space from column names in both data sources and also changed few data types like converting string to timestamp.
Now, Joined trip data and fare data using using 'ID' column.
For Data validation, dropped null values in 'passenger_count', 'trip_time_in_secs', 'trip_distance', 'fare_amount'.

Filtered the data in such a way that 'passenger_count', 'trip_time_in_secs', 'trip_distance', 'fare_amount' must be greater than 0.

After Data Validation, Loaded the Final data into Big Query.

Implementation:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_timestamp
import os

def create_spark_session():
    spark = SparkSession.builder \
        .master("yarn") \
        .appName('new-york-taxi-data-pipeline') \
        .getOrCreate()
    return spark


if __name__ == '__main__':
    spark = create_spark_session()

    #Get the environment variables
    TRIP_DATA_PATH = os.environ.get('TRIP_DATA_PATH', 'gs://new-york-
taxi-01/fare_data_final.csv')
    FARE_DATA_PATH = os.environ.get('FARE_DATA_PATH', 'gs://new-york-
taxi-01/fare_data_final.csv')
    BUCKET_NAME = os.environ.get('BUCKET_NAME', 'new-york-taxi-01')
    BQ_DATASET = os.environ.get('BQ_DATASET',
'new_york_taxi_bq_dataset')
    BQ_TABLE = os.environ.get('BQ_TABLE', 'new_york_taxi_bq_table')

    #Extraction phase
    ##read data from cloud storage
    trip_data = spark.read.option("header", True).option("inferSchema",
True).csv(TRIP_DATA_PATH)
    fare_data = spark.read.option("header", True).option("inferSchema",
True).csv(FARE_DATA_PATH)


    #Transformation Phase
    ##remove leading white space from column names
    ###trip data
    old_column_names = [
        ' hack_license',
        ' vendor_id',
        ' rate_code',
        ' store_and_fwd_flag',
        ' pickup_datetime',
        ' dropoff_datetime',
        ' passenger_count',
        ' trip_time_in_secs',
        ' trip_distance',
        ' pickup_longitude',
        ' pickup_latitude',
        ' dropoff_longitude',
        ' dropoff_latitude'
        ]

    new_column_names = [
        'hack_license',
        'vendor_id',
        'rate_code',
        'store_and_fwd_flag',
        'pickup_datetime',
        'dropoff_datetime',
        'passenger_count',
        'trip_time_in_secs',
        'trip_distance',
        'pickup_longitude',
        'pickup_latitude',
        'dropoff_longitude',
        'dropoff_latitude'
        ]

    for i in range(len(old_column_names)):
        trip_data = trip_data.withColumnRenamed(old_column_names[i],
new_column_names[i])

    ###fare data
    old_column_names = [
        ' hack_license',
        ' vendor_id',
        ' pickup_datetime',
        ' payment_type',
        ' fare_amount',
        ' surcharge',
        ' mta_tax',
        ' tip_amount',
        ' tolls_amount',
        ' total_amount'
        ]
    new_column_names = [
        'hack_license',
        'vendor_id',
        'pickup_datetime',
        'payment_type',
        'fare_amount',
        'surcharge',
        'mta_tax',
        'tip_amount',
        'tolls_amount',
        'total_amount'
        ]

    for i in range(len(old_column_names)):
        fare_data = fare_data.withColumnRenamed(old_column_names[i],
new_column_names[i])

    ###convert string to timestamp
    trip_data = trip_data.withColumn('pickup_datetime',
to_timestamp('pickup_datetime', 'M/d/yy H:mm')) \
                .withColumn('dropoff_datetime',
to_timestamp('dropoff_datetime', 'M/d/yy H:mm'))


    ##Join Trip Data and Fare Data
    trip_data.createOrReplaceTempView("trip_data")
    fare_data.createOrReplaceTempView("fare_data")

    final_df = spark.sql('''
    select
        a.trip_id,
        a.medallion,
        a.hack_license,
        a.vendor_id,
        a.rate_code,
        a.pickup_datetime,
        a.dropoff_datetime,
        a.passenger_count,
        a.trip_time_in_secs,
        a.trip_distance,
        a.pickup_longitude,
        a.pickup_latitude,
        a.dropoff_longitude,
        a.dropoff_latitude,
        b.payment_type,
        b.fare_amount,
        b.surcharge,
        b.mta_tax,
        b.tip_amount,
        b.tolls_amount,
        b.total_amount
    from trip_data as a
        left join fare_data as b
            on a.trip_id = b.trip_id
    '''
                    )

    ##data validation and data accuracy
    ###drop null values in 'passenger_count', 'trip_time_in_secs',
'trip_distance', 'fare_amount'
    final_df = final_df.na.drop(subset=[
    'passenger_count',
    'trip_time_in_secs',
    'trip_distance',
    'fare_amount'
    ])

    ###'passenger_count', 'trip_time_in_secs', 'trip_distance', 'fare_amount'
must be greater than 0
    final_df = final_df.filter((final_df.passenger_count > 0) & \
                        (final_df.trip_time_in_secs > 0) & \
                        (final_df.trip_distance > 0) & \
                        (final_df.fare_amount > 0)
                       )


    #Loading Phase
    ##Saving the final to BigQuery
    ###Use the Cloud Storage bucket for temporary BigQuery export data
used by the connector.
    spark.conf.set('temporaryGcsBucket', BUCKET_NAME)

    final_df.write.format('bigquery') \
    .option('table', f'{BQ_DATASET}.{BQ_TABLE}') \
    .mode('append') \
    .save()
```

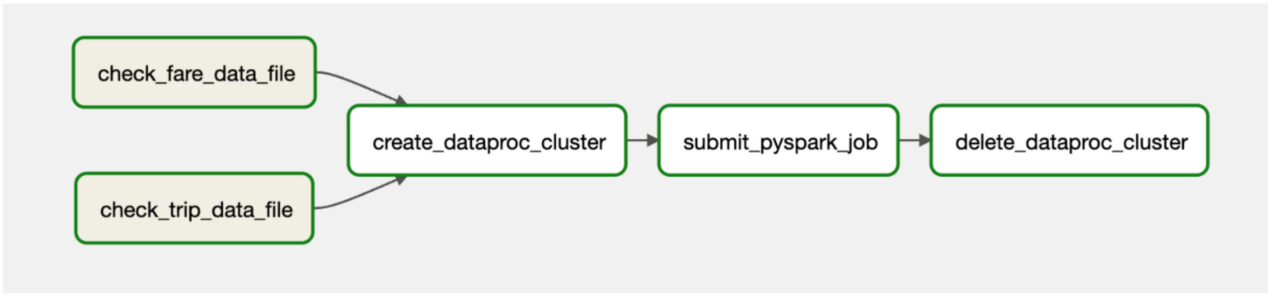
By using Airflow, we can automate the whole process and also manage the cluster in such a way that it deletes the cluster for every cycle.

By this way Once the job is complete, Dataproc cluster will be deleted to save costs.

Airflow:

Apache Spark is a solution that helps a lot with distributed data processing. To automate this task, a great solution is scheduling these tasks within Apache Airflow.

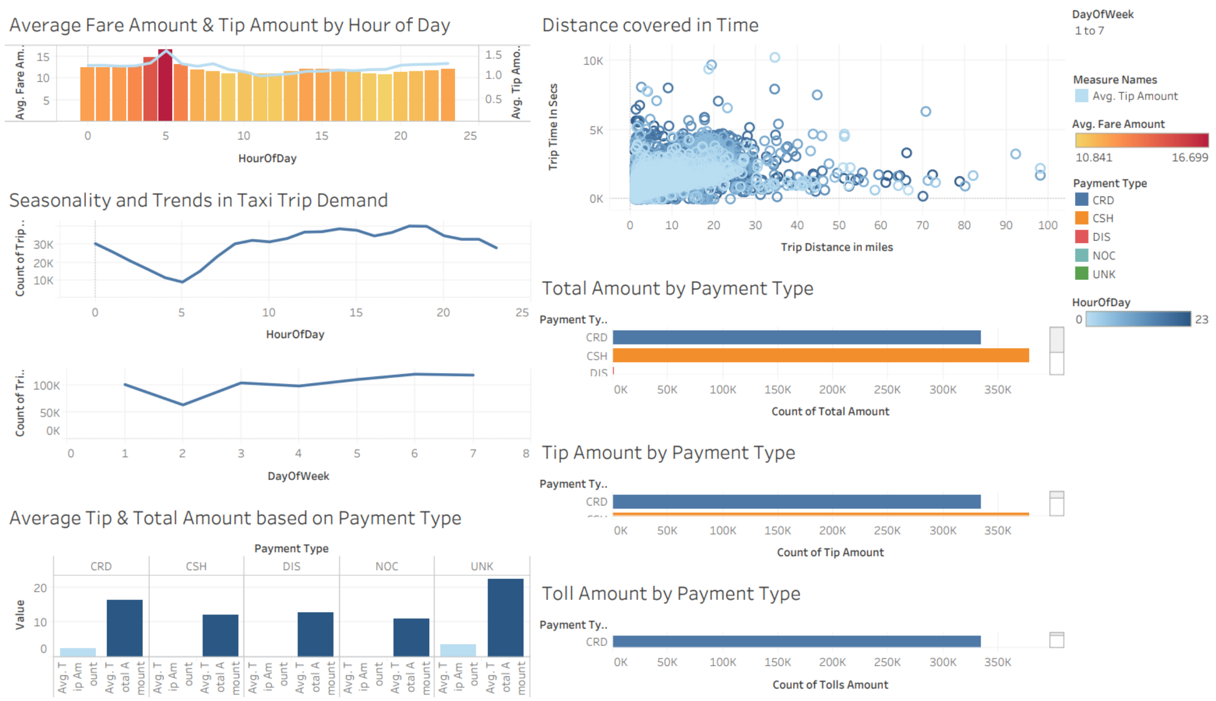
Created a PySpark script to perform data processing task and set up an Airflow environment in GCP, which includes creating a Cloud Composer environment, installing necessary packages, and creating an Airflow DAG to schedule and monitor the PySpark job.



Implementation:

```
from airflow import DAG
import os
from datetime import datetime, timedelta

#gcp operators
from airflow.providers.google.cloud.sensors import gcs

#dataproc
from airflow.providers.google.cloud.operators.dataproc import (
DataprocCreateClusterOperator,
DataprocDeleteClusterOperator,
DataprocSubmitJobOperator,
ClusterGenerator
)

#other operators
from datetime import datetime, timedelta

default_args = {
    "owner": "airflow",
    "email_on_failure": False,
    "email_on_retry": False,
    "email": "sanay321@gmail.com",
    "retries": 0,
    "retry_delay": timedelta(minutes=5),
    "start_date": datetime.today(),
    "end_date": datetime.today() + timedelta(weeks=26),
    "schedule_interval": "0 0 */2 * *"
}

##Get the environment variables
CLUSTER_NAME = os.environ.get('CLUSTER_NAME', 'new-york-taxi-
dataproc-cluster')
GCP_PROJECT_ID = os.environ.get('GCP_PROJECT_ID', "newyork-taxi-
project")
REGION = os.environ.get('REGION', 'us-east1')
ZONE = os.environ.get('ZONE', "us-east1-b")

####check operator
BUCKET_NAME = os.environ.get('BUCKET_NAME', "new-york-taxi-01")
TRIP_DATA_FILE_NAME_PREFIX =
os.environ.get('TRIP_DATA_FILE_NAME_PREFIX', "trip_data/trip_data")
FARE_DATA_FILE_NAME_PREFIX =
os.environ.get('FARE_DATA_FILE_NAME_PREFIX', "fare_data/fare_data")

#Others
TEMP_BUCKET = os.environ.get('TEMP_BUCKET', "new-york-taxi-
temporary-bucket")

#DataProc Cluster Configurations
CLUSTER_GENERATOR_CONFIG = ClusterGenerator(
project_id=GCP_PROJECT_ID,
zone=ZONE,
master_machine_type="n2-standard-2",
worker_machine_type="n2-standard-2",
num_workers=2,
worker_disk_size=300,
master_disk_size=300,
storage_bucket=TEMP_BUCKET,
).make()

#PySpark Job Configurations
PYSPARK_JOB = {
    "reference": {"project_id": GCP_PROJECT_ID},
    "placement": {"cluster_name": CLUSTER_NAME},
    "pyspark_job": {
    "main_python_file_uri": "gs://new-york-taxi-
01/code/newyork_taxi_etl.py",
    "jar_file_uris": [
    "gs://spark-lib/bigquery/spark-3.1-bigquery-0.28.0-preview.jar"
]
},
}


#Airflow DAG
with DAG(dag_id="new-york-taxi-pipeline",start_date =datetime(2023, 4,
8), end_date=datetime(2023, 5, 27) + timedelta(weeks=6) ,
schedule_interval="0 0 */2 * *", default_args=default_args, tags=['new-
york-taxi'], catchup=False) as dag:
    check_trip_data_file = gcs.GCSObjectsWithPrefixExistenceSensor(
    task_id = "check_trip_data_file",
    bucket = BUCKET_NAME,
    prefix = TRIP_DATA_FILE_NAME_PREFIX,
    google_cloud_conn_id = 'google_cloud_storage_default'
)

    check_fare_data_file = gcs.GCSObjectsWithPrefixExistenceSensor(
    task_id = "check_fare_data_file",
    bucket = BUCKET_NAME,
    prefix = FARE_DATA_FILE_NAME_PREFIX,
    google_cloud_conn_id = 'google_cloud_storage_default'
)

create_dataproc_cluster = DataprocCreateClusterOperator(
     task_id="create_dataproc_cluster",
     cluster_name=CLUSTER_NAME,
     project_id=GCP_PROJECT_ID,
     region=REGION,
     cluster_config=CLUSTER_GENERATOR_CONFIG,
)
submit_pyspark_job = DataprocSubmitJobOperator(
     task_id="submit_pyspark_job", job=PYSPARK_JOB, region=REGION,
project_id=GCP_PROJECT_ID)

delete_dataproc_cluster = DataprocDeleteClusterOperator(
     task_id="delete_dataproc_cluster",
     project_id=GCP_PROJECT_ID,
    cluster_name=CLUSTER_NAME,
    region=REGION,
)

[check_trip_data_file, check_fare_data_file] >> create_dataproc_cluster >>
submit_pyspark_job >> delete_dataproc_cluster
```

Analysis:



Once the data is pushed to the Bigquery, It is connected to the Tableau for analysis. As we can see from the above image, we analysed Average fare amount & Tip Amount by Hour of the day. Here, at the hour 5 there is a gradual increase in Dare amount and Tip amount.

In seasonality and Trends in Taxi Trip Demand, the analysis is by day of the week and hour of the day where we can observe that, during the hour 5 there was a very less demand and during hour 14 high demand.

Analysis is also done on payment types made on Total amount, Tip amount, Toll amount where customers made more payments by cash compared to the other payment methods.

Conclusion:

So we conclude here that, the project aims is to analyze New York City taxi trip and fare data to gain insights into the industry's trends, customer preferences, and areas of improvement. The project involves data wrangling, uploading the data to GCP Cloud Storage, setting up an Airflow environment in GCP to automate data processing tasks using Apache Spark, and creating a PySpark script to perform data processing which aims to analyze trip demand trends, investigate customer preferences in payment types, and identify areas with high demand and low supply to optimize taxi industry operations.