***Holistic Approach for the Implementation of Big Data Concepts in NYC Yellow Taxi Data***

Name**: Divya Dhaipullay** ID: **ddhaipul@iu.edu**

## 1. Introduction:

The **aim** of the project is to **analyze and predict** the trends and patterns in NYC yellow taxi rides using big data technologies. The data consists of detailed information about taxi rides in New York City, including the pickup and drop-off locations, fares, tips, trip distances, passenger counts, and other attributes. The volume of the NYC yellow taxi data is quite large, with over 1.2 billion records for the years 2009-2019, representing a total size of approximately 270 GB. In addition to the **large volume**, the data also exhibit other big data characteristics, such as **high velocity**, as the data is constantly being updated in **near-real-time, and high variety**, as the data contains a wide range of attributes and metadata. The data is obtained from the New York City Taxi and Limousine Commission (TLC), which is a **public** agency that regulates taxis and for-hire vehicles in New York City hence we are following **Data Governance** by maintaining data quality, such as data cleaning and data validation. To efficiently process and analyze the large dataset, specialized tools and techniques such as distributed computing frameworks like Apache Spark can be used. To process and analyze the NYC yellow taxi data, big data technologies such as **Google Cloud Platform, PySpark, and No SQL Database**. These tools enable efficient **querying, processing, and analysis of large-scale datasets, and provide scalable and distributed computing capabilities.**

The data is available on Big Query: https://console.cloud.google.com/bigquery?project=graceful-fin-383520&ws=!1m9!1m3!8m2!1s9081414228!2s33dbb83eed374c0db68ae1602d77f0e7!1m4!4m3!1snyc-tlc!2syellow!3strips

| Column Name | Data Type | Nullable | Description |
|---|---|---|---|
| vendor_id | STRING | YES | A designation for the technology vendor that provided the record. CMT=Creative Mobile Technologies VTS= VeriFone, Inc. DDS=Digital Dispatch Systems |
| pickup_datetime | TIMESTAMP | YES | The date and time when the meter was engaged. |
| dropoff_datetime | TIMESTAMP | YES | The date and time when the meter was disengaged. |
| pickup_longitude | FLOAT | YES | Longitude where the meter was engaged. |
| pickup_latitude | FLOAT | YES | Latitude where the meter was engaged. |
| dropoff_longitude | FLOAT | YES | Longitude where the meter was disengaged. |
| dropoff_latitude | FLOAT | YES | Latitude where the meter was disengaged. |
| rate_code | STRING | YES | The final rate code in effect at the end of the trip. 1= Standard rate 2=JFK 3=Newark 4=Nassau or Westchester 5=Negotiated fare 6=Group ride |
| passenger_count | INTEGER | YES | The number of passengers in the vehicle. This is a driver-entered value. |
| trip_distance | FLOAT | YES | The elapsed trip distance in miles reported by the taximeter. |
| payment_type | STRING | YES | A numeric code signifying how the passenger paid for the trip. CRD= Credit card CSH= Cash NOC= No charge DIS= Dispute UNK= Unknown |
| fare_amount | FLOAT | YES | The time-and-distance fare calculated by the meter. |
| Extra | FLOAT | YES | Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges. |
| mta_tax | FLOAT | YES | $0.50 MTA tax that is automatically triggered based on the metered rate in use. |
| imp_surcharge | FLOAT | YES | $0.30 improvement surcharge assessed on trips at the flag drop. The improvement surcharge began being levied in 2015. |
| tip_amount | FLOAT | YES | Tip amount – This field is automatically populated for credit card tips. Cash tips are not included. |
| tolls_amount | FLOAT | YES | Total amount of all tolls paid in trip. |
| total_amount | FLOAT | YES | The total amount charged to passengers. Does not include cash tips. |
| store_and_fwd_flag | STRING | YES | This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip |

Table 1 : Data schema for NYC Yellow Taxi Trip

## 2. Background:

Yellow taxis are a prominent form of transportation in NYC, operated by licensed taxi drivers and regulated by the NYC Taxi and Limousine Commission (TLC). They are known for their iconic yellow color and can be hailed from the street picked up at designated taxi stands or booked through a taxi app. Yellow taxis play an important role in NYC's transportation system, providing convenient transportation for residents and tourists alike. The data collected from yellow taxi trips can be used for various purposes, such as transportation planning and data analysis. Yellow taxis in New York City generate vast amounts of data daily, which includes information about the pickup and drop-off locations, trip distances, fares, tips, payment methods, and more. This data represents a significant source of information for transportation planners, researchers, and policymakers. However, the size and complexity of the data make it difficult to analyze and extract meaningful insights without the use of big data technologies and data science techniques. In recent years, the availability of cloud computing platforms and big data tools such as Hadoop, Spark, and Kafka have enabled the processing and analysis of large-scale data sets at scale. This has led to the emergence of new fields such as urban analytics and transportation data science, which leverage big data technologies to improve our understanding of urban systems and design more efficient transportation networks. By analyzing the data generated by yellow taxis in NYC, we can gain insights into travel patterns, congestion levels, and transportation demand across different neighborhoods and times of the day. These insights can inform policy decisions related to transportation planning, infrastructure investment, and traffic management. Therefore, the analysis of yellow taxi

data is not only an interesting research problem but also an important practical application of big data technologies.

While both Pandas and PySpark are powerful tools for data analysis, they have distinct differences in terms of memory management, distributed computing capabilities, and syntax/APIs. Pandas is well-suited for smaller datasets that can fit in the memory of a single machine, while PySpark excels in processing large datasets distributed across multiple machines. Understanding these differences is crucial in choosing the right tool for specific data analysis requirements.

| Aspect | Spark | Pandas |
|---|---|---|
| Type of library | Distributed processing framework | In-memory data manipulation library |
| Data processing | Can handle large datasets, parallel processing | Primarily for small to medium-sized datasets, sir thread |
| Data sources | Can process data from various sources | Limited to data from files, databases, and in-mem data |
| Speed | Can handle big data with fast processing speed | Limited by memory, slower on large datasets |
| Language support | Supports multiple programming languages | Primarily for Python |
| Learning curve | Steep learning curve for beginners | Easy to learn and use |

## 3. Methodology

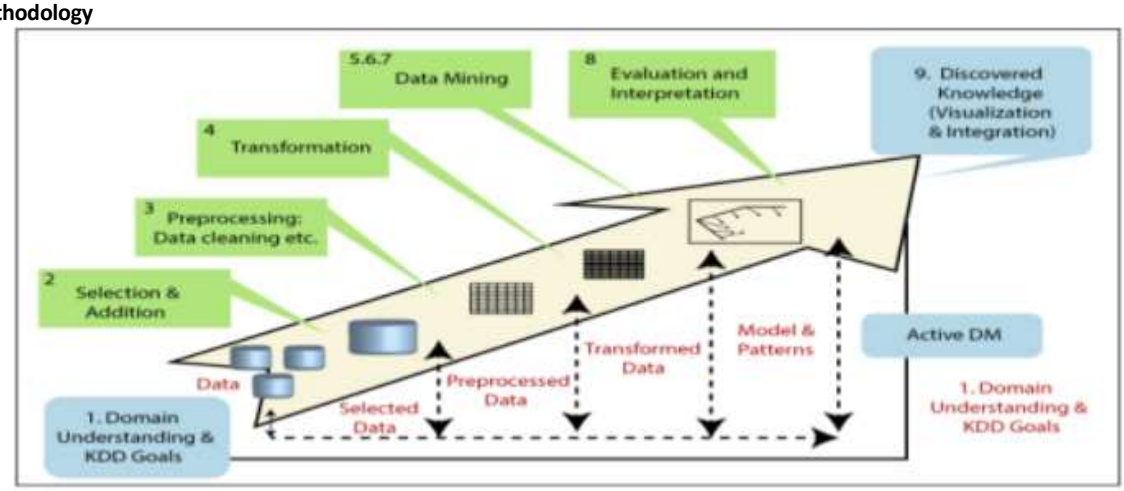


Fig 1: Project Architecture for Exploratory Data Analysis on Pyspark

We shall be **using Pyspark for Exploratory Analysis of Big Data and Google Cloud Pipeline, later, for model deployment and Predictive Analytics**. All the below steps are implemented in the **Pyspark Jupyter file on GCP**. This is just a summary of it.

**1.** ***Domain Knowledge:*** This involves a deeper understanding of the application domain, which in this case is the transportation system in NYC. We need to identify the approach to various decisions such as data transformation, representation, and algorithms.

**2.** ***Data Set Selection and Creation:*** Once we've identified our objectives, the next step is to select the appropriate data set for our analysis. We'll need to assess what data is available, acquire any additional data we may need, and integrate all relevant data into one cohesive set for knowledge discovery. However, managing large and complex data repositories can be expensive and risky, so we'll need to be strategic in our data selection.

**3.** ***Data Preprocessing and Cleaning:*** Now we perform preprocessing and cleaning to ensure high quality. This includes identifying any missing data, handling outliers and anomalies, and removing any duplicate or irrelevant information. We'll also convert and transform the data as needed to fit our analysis needs.

**4.** ***Data Transformation:*** It involves preparing the data for analysis. This may involve dimension reduction, such as feature selection or extraction, as well as attribute modification, such as numerical discretization or functional transformation. This stage is critical for the success of the overall project, as it sets the stage for the data mining algorithms that will be used to extract insights and patterns.

**5.** ***Data Mining:*** It is a process of extracting valuable insights and patterns from large datasets. It involves analyzing data from different angles, summarizing it into useful information, and discovering hidden patterns or relationships used to make decisions.

**6.** ***Data Evaluation:*** In this stage, we assess the quality and usefulness of the mined patterns and rules. We also analyze how the preprocessing and transformation steps affect the outcomes. We use visualizations such as tables and charts to present the results and document the findings for future reference.

**7.** ***Using the Discovered Knowledge:*** In this final stage, we incorporate the discovered knowledge into a new framework for future actions. The effectiveness of this stage determines the success of the whole process. However, there may be challenges such as loss of data due to changes in data structures and alterations to the data domain.

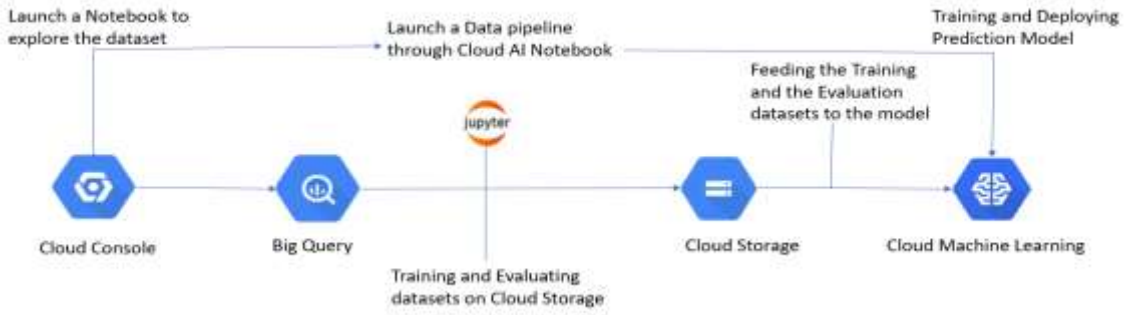


Fig 2: Architecture for Google Cloud Pipeline

The pipeline uses various components, including **Cloud Storage, Big Query, and Cloud ML**. **Cloud Storage** provides storage in the form of buckets containing objects that can be accessed using their methods. The buckets also contain bucketAccessControls that allow fine-grained manipulation of access controls. **Big Query** is an immutable SQL data warehouse suitable for OLAP applications. It is a serverless and cost-effective multicloud data warehouse designed for analyzing big data. **Cloud ML** is a hosted platform that enables running distributed machine learning training jobs and predictions at scale.

### 3.1 Creating a Project on Google Cloud Platform

I first created a project “Divya Dhaipullay – BigData” under the given educational organization iu.edu, and a bucket - 'divyadhaipullay-storagebucket' and configured the subnet(VPC). I had IAM permissions and I proceeded to create the storage bucket for it.

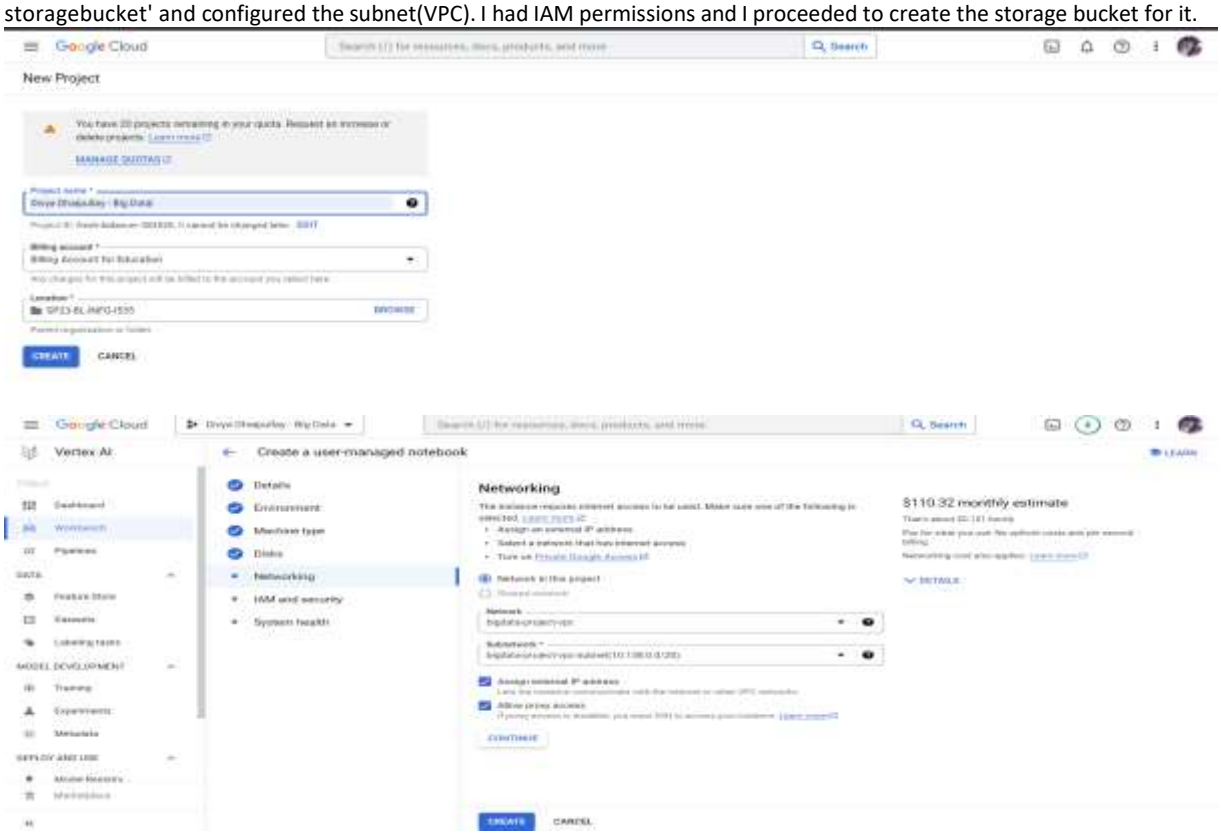


Fig 3: GCP Project

### 3.2 Cloud Storage Set up using Bucket

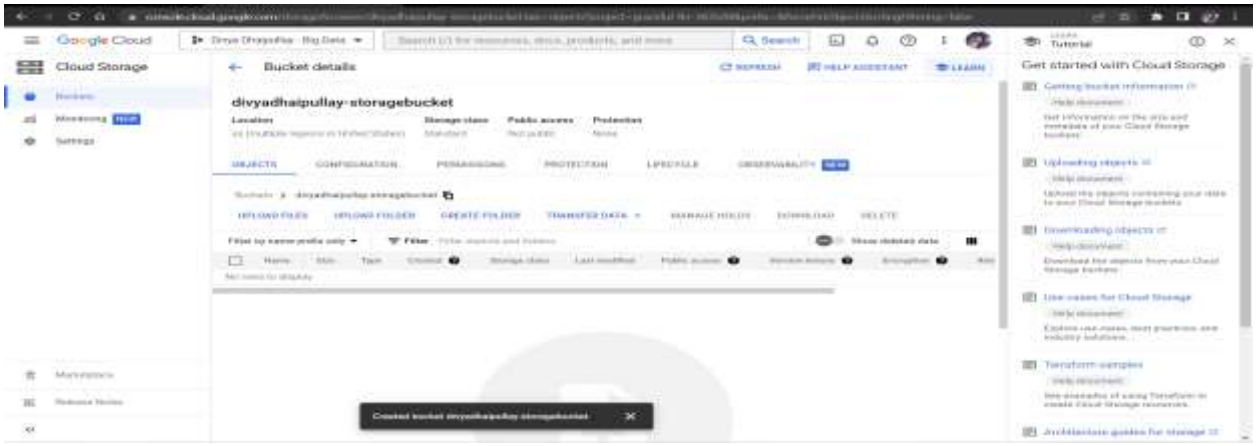


Fig 4: Setting up the GCP Bucket

### 3.3 Launching a Notebook Instance

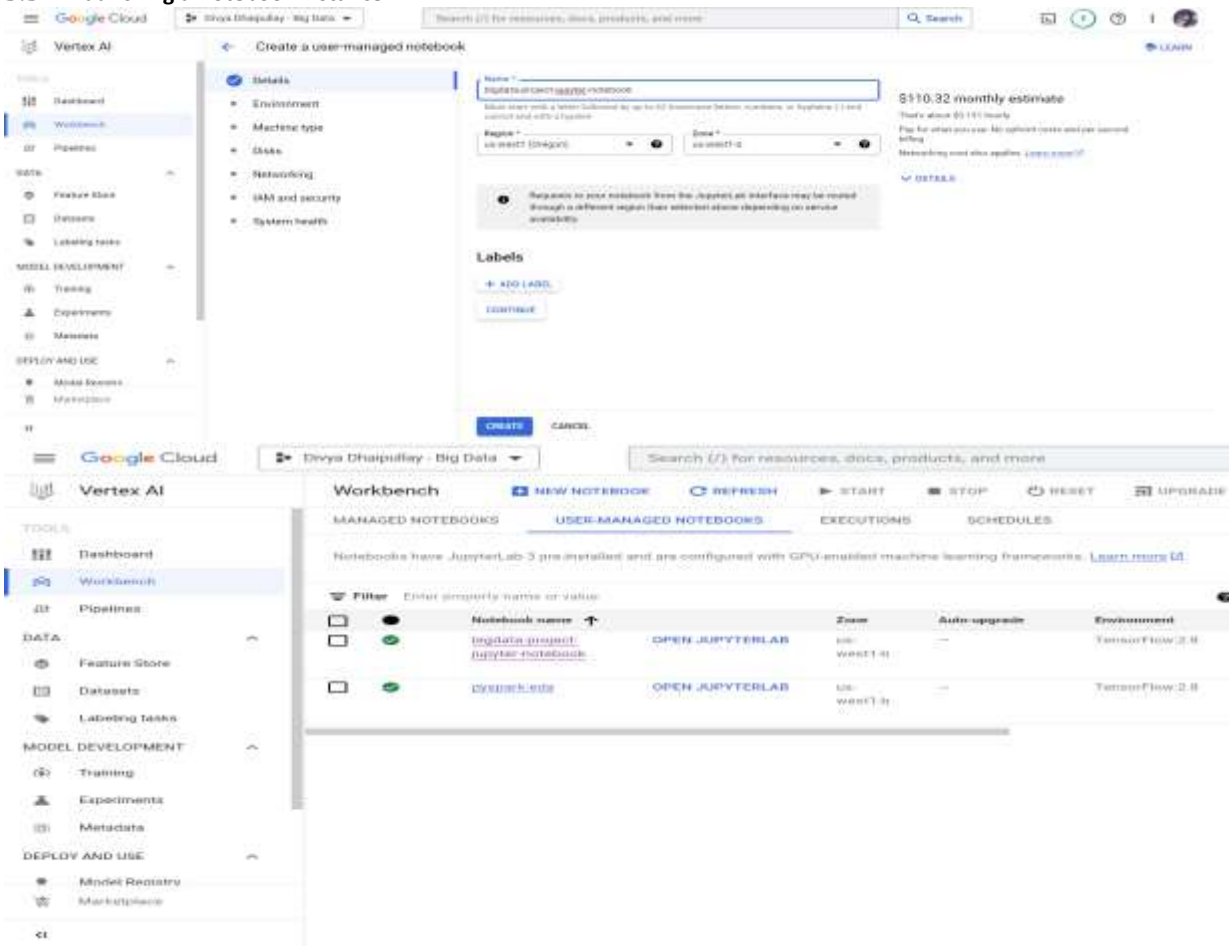


Fig 5: Launching a Notebook Instance

Under the "**AI platform**" in the navigation bar, I chose the notebook and enabled a Notebook API, created new notebook. I had to selected new instance with **Debian 10 OS, 4vCPU's, 15GB RAM** and **TensorFlow 2.x** > **Without GPUs**

### 3.4 Invoking BigQuery

On BigQuery, a serverless data warehouse, I explored the NYC Yellow Taxi dataset. The given SQL query selects specific columns from a subquery that retrieves data from the nyc-tlc.yellow.trips table. The subquery includes a window function ROW_NUMBER() that assigns a row number to each row in the result set. The outer query then filters the result set by selecting only rows where the row number is a multiple of 100,000 (or any desired value specified by changing the MOD(row_num, 100000) condition), effectively creating a random and repeatable subset of the data. Save results to **csv file format.**


Fig 6: Executing a SQL Query on the BigQuery console

### 3.5 Running Data Analysis on AI Platform Notebooks

I upgraded to the most recent version of the BigQuery Python Client Library and imported it into my Jupyter notebook. I created a client to interact with the BigQuery API, allowing me to run queries on the NYC Yellow Taxi dataset. Once I had retrieved the data from BigQuery, I used it to construct a Pandas dataframe. Working with the Pandas dataframe locally made it easier to manipulate the data since it was smaller in size and could be stored locally. I examined the dataset and pre-processed the features to identify useful features for building a model.

```
[10]: def sample_between(a, b):
          basequery = """
      SELECT
        (tolls_amount + fare_amount) AS fare_amount,
        pickup_longitude AS pickuplon,
        pickup_latitude AS pickuplat,
        dropoff_longitude AS dropofflon,
        dropoff_latitude AS dropofflat,
        passenger_count*1.0 AS passengers
      FROM
        `nyc-tlc.yellow.trips`
      WHERE
        trip_distance > 0
        AND fare_amount >= 2.5
        AND pickup_longitude > -78
        AND pickup_longitude < -70
        AND dropoff_longitude > -78
        AND dropoff_longitude < -70
        AND pickup_latitude > 37
        AND pickup_latitude < 45
        AND dropoff_latitude > 37
        AND dropoff_latitude < 45
        AND passenger_count > 0
        """
          sampler = "AND MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING))), EVERY_N) = 1"
          sampler2 = "AND {0} >= {1}\n AND {0} < {2}".format(
                 "MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING))), EVERY_N * 100)",
                 "(EVERY_N * {})".format(a), "(EVERY_N * {})".format(b)
              )
          return "{}\n{}\n{}".format(basequery, sampler, sampler2)

      def create_query(phase, EVERY_N):
        """Phase: train (70%) valid (15%) or test (15%)"""
        query = ""
        if phase == 'train':
          # Training
          query = sample_between(0, 70)
        elif phase == 'valid':
          # Validation
          query = sample_between(70, 85)
        else:
          # Test
          query = sample_between(85, 100)
        return query.replace("EVERY_N", str(EVERY_N))

      print(create_query('train', 100000))
```

Fig 7: Creating a Query

```
# Define function to write DataFrame to CSV
def to_csv(df, filename):
    outdf = df.copy(deep=False)
    outdf.loc[:, 'key'] = np.arange(0, len(outdf))  # rownumber as key
    # Reorder columns so that target is first column
    cols = outdf.columns.tolist()
    cols.remove('fare_amount')
    cols.insert(0, 'fare_amount')
    print(cols)  # new order of columns
    outdf = outdf[cols]
    outdf.to_csv(filename, header=False, index_label=False, index=False)
    print("Wrote {} to {}".format(len(outdf), filename))

# Initialize BigQuery client
client = bigquery.Client()

# Loop through phases and fetch data from BigQuery, write to CSV
for phase in ['train', 'valid', 'test']:
    query = create_query(phase, 100000)  # Assuming create_query() function is defined elsewhere
    df = client.query(query).to_dataframe()
    to_csv(df, 'taxi-{}.csv'.format(phase))

# Fetch data from BigQuery and create a copy of the DataFrame
query = create_query(phase, 100000)  # Assuming create_query() function is defined elsewhere
df = client.query(query).to_dataframe()
outdf = df.copy(deep=False)
```

```
['fare_amount', 'pickuplon', 'pickuplat', 'dropofflon', 'dropofflat', 'passengers', 'key']
Wrote 7645 to taxi-train.csv
['fare_amount', 'pickuplon', 'pickuplat', 'dropofflon', 'dropofflat', 'passengers', 'key']
Wrote 1814 to taxi-valid.csv
['fare_amount', 'pickuplon', 'pickuplat', 'dropofflon', 'dropofflat', 'passengers', 'key']
Wrote 1017 to taxi-test.csv
```

[12]:
```
# Reorder columns so that target is first column
cols = outdf.columns.tolist()
cols.remove('fare_amount')
cols.insert(0, 'fare_amount')
cols
```

[12]:
```
['fare_amount',
 'pickuplon',
 'pickuplat',
 'dropofflon',
 'dropofflat',
 'passengers']
```

Fig 8: Doing Data Analysis with the executed Query

### 3.6 Building and Evaluating the Model

I have developed a basic Deep Neural Network Model that includes two hidden layers with relu activation, an adam optimizer, RMSE as a metric, and MSE as a loss function. To account for the large dataset, I set the batch size to 32 and the epochs to 100. I evaluated the model's performance by visualizing the training and validation loss to determine if the model was overfitting or underfitting. Once the model was complete, I saved it and exported it to a cloud storage platform. Finally, I deployed the model to a cloud AI platform to make predictions.

```python
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Split the data into features (X) and target (y)
X = outdf.drop('fare_amount', axis=1)
y = outdf['fare_amount']

# Perform feature scaling on the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Define the model architecture
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=(X.shape[1],)))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='linear'))

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Train the model and get the history
history = model.fit(X_train, y_train, batch_size=32, epochs=100, validation_split=0.2)

# Evaluate the model on the test set
mse = model.evaluate(X_test, y_test)
print("Mean Squared Error (MSE):", mse)

# Make predictions
predictions = model.predict(X_test)

# Perform any further analysis or tasks with the predicted values as needed

# For example, you can calculate additional evaluation metrics
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print("Mean Squared Error (MSE):", mse)
print("R2 Score:", r2)
```

Fig 9: Deep Learning Model Building and Prediction using training and validation set

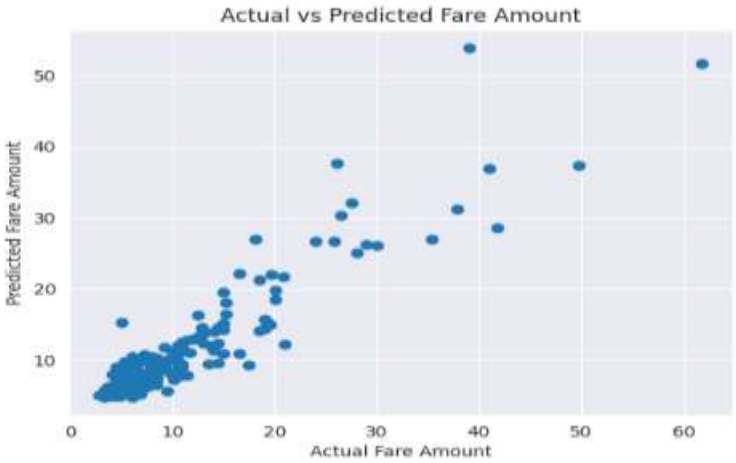


Fig 10: Actual vs Predicted curve

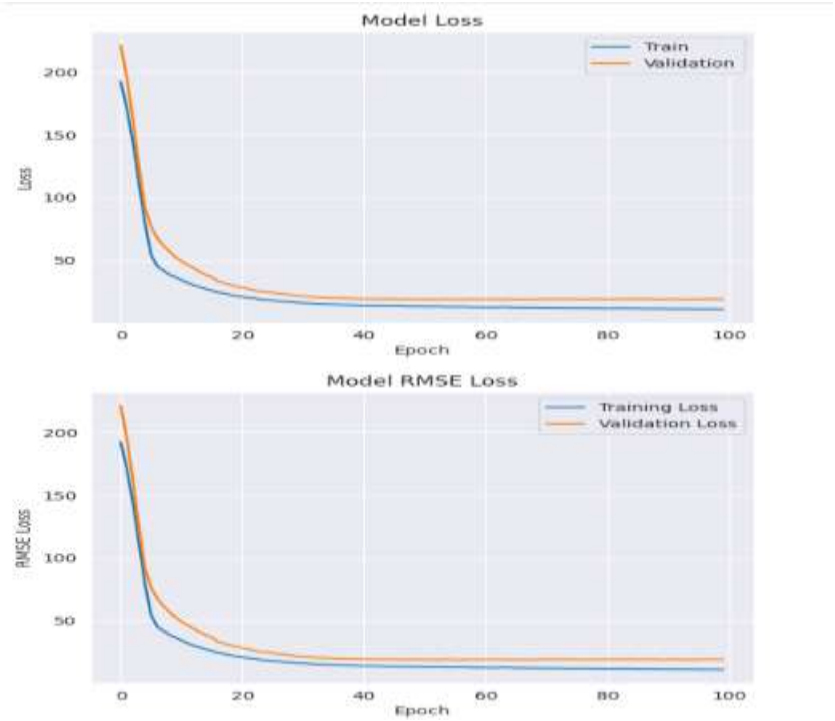


Fig 11: Model Loss and Model RMSE Loss

If both the MSE and R2 values are the same, it means that the model's predictions are on average equal to the actual values, and the model is able to explain all the variance in the target variable. However, this scenario is rare in practice, and usually, there is a trade-off between these two metrics. For example, a model with a low MSE may have a lower R2 score, indicating that it is not able to explain all the variance in the target variable.

```
[15]: # Save the model
model.save('my_model.h5')

# Export the model to cloud storage (for example, Google Cloud Storage)
from google.cloud import storage

# Create a storage client
storage_client = storage.Client()

# Set the bucket name and blob name
bucket_name = 'divyadhaipullay-storagebucket'
blob_name = 'my_model.h5'

# Upload the model file to the cloud storage
bucket = storage_client.get_bucket(bucket_name)
blob = bucket.blob(blob_name)
blob.upload_from_filename('my_model.h5')

print(f'Model exported to cloud storage: gs://{bucket_name}/{blob_name}')
```

```
Model exported to Cloud Storage: gs://divyadhaipullay-storagebucket/my_model.h5
```

```
[16]: # Create a DataFrame for actual and predicted values
df_actual_vs_predicted = pd.DataFrame({'Actual': y_test, 'Predicted': predictions.flatten()})
df_actual_vs_predicted.head()
```

[16]:

| | Actual | Predicted |
|---|---|---|
| 916 | 10.1 | 11.091514 |
| 31 | 7.7 | 8.848761 |
| 507 | 5.3 | 5.646175 |
| 518 | 2.9 | 4.120762 |
| 215 | 7.0 | 6.645530 |

Fig 12: Exporting the Model to Cloud Storage

The pipeline requires specific configurations such as the runtime version of TensorFlow to use, the Python version (which currently only supports Python 3.7 for TF 2.1), and the cloud region to train in. Once the model is trained, it is exported and saved as a saved_model.pb file within the directory.

## 4. Results

In [1]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, FloatType, StringType, TimestampType
from pyspark.sql.functions import col, count, mean, max, hour, avg, to_timestamp, sum, date_format,dayofmonth, month
from pyspark.sql.types import *
from pyspark.sql.window import Window
from pyspark.sql.functions import *
# Create a Spark session
spark = SparkSession.builder.appName("EDAPySpark").getOrCreate()

# Define the schema for the dataset
schema = StructType([
    StructField("Vendor_id", IntegerType(), True),
    StructField("tpep_pickup_datetime", TimestampType(), True),
    StructField("tpep_dropoff_datetime", TimestampType(), True),
    StructField("passenger_count", IntegerType(), True),
    StructField("trip_distance", FloatType(), True),
    StructField("pickup_longitude", FloatType(), True),
    StructField("pickup_latitude", FloatType(), True),
    StructField("RatecodeID", IntegerType(), True),
    StructField("store_and_fwd_flag", StringType(), True),
    StructField("dropoff_longitude", FloatType(), True),
    StructField("dropoff_latitude", FloatType(), True),
    StructField("payment_type", IntegerType(), True),
    StructField("fare_amount", FloatType(), True),
    StructField("extra", FloatType(), True),
    StructField("mta_tax", FloatType(), True),
    StructField("tip_amount", FloatType(), True),
    StructField("tolls_amount", FloatType(), True),
    StructField("improvement_surcharge", FloatType(), True),
    StructField("total_amount", FloatType(), True)
])

# Read the dataset with the defined schema
df = spark.read.csv("merged_data.csv", header=True, schema=schema)
df
```

Out[1]:
```
DataFrame[Vendor_id: int, tpep_pickup_datetime: timestamp, tpep_dropoff_datetime: timestamp, passenger_count: int, trip_distanc
e: float, pickup_longitude: float, pickup_latitude: float, RatecodeID: int, store_and_fwd_flag: string, dropoff_longitude: floa
t, dropoff_latitude: float, payment_type: int, fare_amount: float, extra: float, mta_tax: float, tip_amount: float, tolls_amoun
t: float, improvement_surcharge: float, total_amount: float]
```

Fig 13: Defining the Schema of the dataset

In [2]:
```
# Print the schema of the dataframe
df.printSchema()
```

```
root
 |-- Vendor_id: integer (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
 |-- tpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_distance: float (nullable = true)
 |-- pickup_longitude: float (nullable = true)
 |-- pickup_latitude: float (nullable = true)
 |-- RatecodeID: integer (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- dropoff_longitude: float (nullable = true)
 |-- dropoff_latitude: float (nullable = true)
 |-- payment_type: integer (nullable = true)
 |-- fare_amount: float (nullable = true)
 |-- extra: float (nullable = true)
 |-- mta_tax: float (nullable = true)
 |-- tip_amount: float (nullable = true)
 |-- tolls_amount: float (nullable = true)
 |-- improvement_surcharge: float (nullable = true)
 |-- total_amount: float (nullable = true)
```

In [3]:
```
# Count the number of rows in the dataframe
print("Total rows: ", df.count())
```

```
Total rows:  47248845
```

In [4]:
```
# Count the number of distinct values in each column
df.agg(*(count(col(c)).alias(c) for c in df.columns)).show()
```

```
+---------+--------------------+---------------------+---------------+-------------+----------------+---------------+----------+------------------+-----------------+----------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+
|Vendor_id|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|pickup_longitude|pickup_latitude|RatecodeID|store_and_fwd_flag|dropoff_longitude|dropoff_latitude|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|
+---------+--------------------+---------------------+---------------+-------------+----------------+---------------+----------+------------------+-----------------+----------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+
| 47248845|            47248845|             47248845|       47248845|     47248845|        47248845|       47248845|  47248845|             47248845|         47248845|        47248845|    47248845|   47248845|47248845|47248845|  47248845|    47248845|             47248842|    47248845|
+---------+--------------------+---------------------+---------------+-------------+----------------+---------------+----------+------------------+-----------------+----------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+
```

Fig 14: Defining the data

***Various kinds of Data Analysis***

In [5]:
```
# Example 0: Perform basic statistics on numeric columns
numeric_cols = ['Vendor_id', 'passenger_count', 'trip_distance', 'total_amount']
numeric_stats = df.select(numeric_cols).describe().show()
```

```
+-------+------------------+------------------+------------------+------------------+
|summary|         Vendor_id|   passenger_count|     trip_distance|      total_amount|
+-------+------------------+------------------+------------------+------------------+
|  count|          47248845|          47248845|          47248845|          47248845|
|   mean| 1.5295704900029968|1.6670397339871485|7.508417907428523|15.592752588005703|
| stddev|0.4991248254615424|1.3220922307335804|6487.658256038432|  580.139258980823|
|    min|                 1|                 0|      -3390583.8|            -958.4|
|    max|                 2|                 9|     1.9072628E7|         3950611.5|
+-------+------------------+------------------+------------------+------------------+
```

The summary includes the count, mean, standard deviation, minimum, and maximum values for each column. The count indicates the number of non-null values in each column, which is the same as the total number of rows in the DataFrame.

The mean and standard deviation provide information about the distribution of values in each column. For example, the mean of "trip_distance" is 7.508, which suggests that the majority of trips are relatively short. The standard deviation of "total_amount" is 580.139, indicating that there is significant variation in the fare amount.

The minimum and maximum values show the range of values in each column. For example, the minimum value for "trip_distance" is -3390583.8, which is likely an error or outlier. The maximum value for "total_amount" is 3950611.5, which is also likely an outlier.

Overall, these summary statistics can help identify potential errors, outliers, and patterns in the data, which can inform further analyses and modeling.

In [6]:
```
# Example 1: Grouping and Aggregating Data

# Group by vendor_id and calculate average trip_duration and maximum passenger_count
# Perform the aggregation
df.groupBy("vendor_id").agg(
    mean("trip_distance").alias("avg_trip_distance"),
    max("passenger_count").alias("max_passenger_count")
).show()
```

```
+---------+------------------+--------------------+
|vendor_id| avg_trip_distance|max_passenger_count|
+---------+------------------+--------------------+
|        1|12.669640946106417|                   9|
|        2| 2.923586103840142|                   9|
+---------+------------------+--------------------+
```

The "vendor_id" column contains the unique values of the "Vendor_id" column in the original DataFrame "df". The "avg_trip_distance" column contains the average trip distance for each vendor, calculated using the "avg" function on the "trip_distance" column grouped by "Vendor_id". The "max_passenger_count" column contains the maximum passenger count for each vendor, calculated using the "max" function on the "passenger_count" column grouped by "Vendor_id".

The output suggests that there are two unique vendor IDs in the dataset (1 and 2), and they have different average trip distances and maximum passenger counts. Vendor 1 has a much higher average trip distance than Vendor 2, suggesting that they may serve different types of trips or areas. Both vendors have a maximum passenger count of 9, which is the maximum value in the "passenger_count" column in the original DataFrame.

Overall, this information can be useful for understanding the characteristics of the data and potentially identifying patterns or insights that can inform further analysis or modeling.

In [7]:
```
# Example 2: Filtering Data

# Filter data to only include rows where passenger_count is greater than 1
filtered_df = df.filter(col("passenger_count") > 1)
filtered_df.show(5)
```

```
+---------+--------------------+--------------------+---------------+-------------+---------------+---------------+----------
+--------------------+--------+--------------------+---------------+------------+-----+-------+----------+------------+----------
----------+------------+
|Vendor_id|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|pickup_longitude|pickup_latitude|RatecodeID
|store_and_fwd_flag|dropoff_longitude|dropoff_latitude|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvems
nt_surcharge|total_amount|
+---------+--------------------+--------------------+---------------+-------------+---------------+---------------+----------
+--------------------+--------+--------------------+---------------+------------+-----+-------+----------+------------+----------
----------+------------+
|        2| 2016-03-01 00:00:00|  2016-03-01 00:51:06|              2|        19.98|       -73.78282|       40.64481|      null
|                 N|        -73.97454|        40.67577|           1|        54.5|  0.5|    0.5|      8.0|         0.0|
0.3|        63.8|
|        2| 2016-03-01 00:00:00|  2016-03-01 00:00:00|              3|        10.78|       -73.86542|      40.769814|      null
|                 N|        -73.96965|       40.757767|           1|        31.5|  0.0|    0.5|      3.78|         5.54|
0.3|       41.62|
|        2| 2016-03-01 00:00:00|  2016-03-01 00:00:00|              5|        30.43|       -73.97174|      40.792183|      null
|                 N|        -74.17717|       40.695053|           1|        98.0|  0.0|    0.0|       0.0|        15.5|
0.3|       113.8|
|        2| 2016-03-01 00:00:00|  2016-03-01 00:00:00|              5|         5.92|       -74.0172|      40.705383|      null
|                 N|        -73.97807|       40.755787|           1|        23.5|  1.0|    0.5|      5.06|         0.0|
0.3|       30.36|
|        2| 2016-03-01 00:00:00|  2016-03-01 00:00:00|              6|         5.72|       -73.95458|      40.727848|      null
|                 N|             0.0|             0.0|           2|        25.0|  0.5|    0.5|       0.0|         0.0|
0.3|       24.3|
+---------+--------------------+--------------------+---------------+-------------+---------------+---------------+----------
+--------------------+--------+--------------------+---------------+------------+-----+-------+----------+------------+----------
----------+------------+
only showing top 5 rows
```

The code is demonstrating an example of filtering data using PySpark.

The code is showing a table with three columns: vendor_id, avg_trip_distance, and max_passenger_count. It appears that the data has been aggregated by vendor_id, and the average trip distance and maximum passenger count have been calculated for each vendor.

In terms of EDA (Exploratory Data Analysis), this could be the result of a summary or aggregation step where the data has been grouped by vendor_id and some basic statistics have been calculated to better understand the characteristics of each vendor.

The code is filtering the original dataframe to only include rows where the passenger_count is greater than 1. This is a common data preprocessing step where irrelevant or outlier data is removed from the dataset.

In [8]:
```
# Example 3: Data Cleaning

# Convert pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude columns to FloatType
df = df.withColumn("pickup_longitude", col("pickup_longitude").cast(FloatType()))
df = df.withColumn("pickup_latitude", col("pickup_latitude").cast(FloatType()))
df = df.withColumn("dropoff_longitude", col("dropoff_longitude").cast(FloatType()))
df = df.withColumn("dropoff_latitude", col("dropoff_latitude").cast(FloatType()))
```

In Example 3, the code converts the data type of four columns in the DataFrame df from string to float. This is a common data cleaning step that ensures the data is in the correct format for analysis. The four columns are pickup_longitude, pickup_latitude, dropoff_longitude, and dropoff_latitude. These columns represent the longitude and latitude coordinates of the pickup and dropoff locations for each taxi ride in the dataset. By converting them to float, we can perform numerical calculations and geospatial analysis on these columns.

In [9]:
```
# Example 4: Data Aggregation and Summary Statistics

# Calculate summary statistics for numeric columns
df.describe(["trip_distance", "passenger_count"]).show()
```

```
+-------+------------------+------------------+
|summary|     trip_distance|   passenger_count|
+-------+------------------+------------------+
|  count|          47248845|          47248845|
|   mean|7.508417907428523|1.6670197339871483|
| stddev|6487.658256038432|1.3220922307333804|
|    min|       -3390583.8|                 0|
|    max|      1.907262887|                 9|
+-------+------------------+------------------+
```

Looking at the summary statistics for trip_distance and passenger_count, we can see that the minimum value for trip_distance is negative, which is not possible, and the maximum value is very large, which could indicate some outliers in the data. The minimum value for passenger_count is 0, which also seems unlikely. We may need to clean the data further to remove these anomalies.

```
trip_duration_max = df.groupBy('Vendor_id').agg(max('trip_distance').alias('max_trip_duration'))

# Show the results of groupby and aggregation
print("Vendor Count:")
vendor_count.show()
```

```
Vendor Count:
+---------+--------+
|Vendor_id|   count|
+---------+--------+
|        1|22227251|
|        2|25021594|
+---------+--------+
```

There were two vendors that provided data for this dataset. Vendor 1 provided 22,227,251 records and vendor 2 provided 25,021,594 records.

In [11]:
```
print("Average Passenger Count by Vendor:")
passenger_count_avg.show()
```

```
Average Passenger Count by Vendor:
+---------+--------------------+
|Vendor_id|avg_passenger_count|
+---------+--------------------+
|        1| 1.2541740316874992|
|        2|  2.033797728754344|
+---------+--------------------+
```

The result shows the average passenger count for each vendor in the dataset. The first column, "Vendor_id," indicates the ID of the vendor, and the second column, "avg_passenger_count," shows the average number of passengers for each vendor.

According to the results, the vendor with ID 1 has an average passenger count of 1.25, while the vendor with ID 2 has an average passenger count of 2.03. This suggests that on average, rides from vendor 2 tend to have more passengers than rides from vendor 1. However, it's worth noting that the dataset may not be representative of all taxi rides in New York City, so these results should be interpreted with caution.

In [12]:
```
print("Minimum Trip Duration by Vendor:")
trip_duration_min.show()
```

```
Minimum Trip Duration by Vendor:
+---------+-----------------+
|Vendor_id|min_trip_duration|
+---------+-----------------+
|        1|       -3390583.8|
|        2|              0.0|
+---------+-----------------+
```

In [13]:
```
print("Maximum Trip Duration by Vendor:")
trip_duration_max.show()
```

```
Maximum Trip Duration by Vendor:
+---------+-----------------+
|Vendor_id|max_trip_duration|
+---------+-----------------+
|        1|     1.9072628E7|
|        2|           390.07|
+---------+-----------------+
```

This shows the maximum trip duration in seconds for each vendor. For vendor 1, the maximum trip duration is 1.9072628E7 seconds, which is roughly equivalent to 220 days. For vendor 2, the maximum trip duration is 390.07 seconds, which is about 6.5 minutes. This information could be useful for detecting outliers or errors in the data, as well as for analyzing the typical length of trips for each vendor.

In [14]:
```
# Example 5: Data Transformation

# Extract the hour of day from pickup_datetime and dropoff_datetime columns
df = df.withColumn("pickup_hour", hour(col("tpep_pickup_datetime")))
df = df.withColumn("dropoff_hour", hour(col("tpep_dropoff_datetime")))

# Extract the day of the week from pickup_datetime and dropoff_datetime columns
df = df.withColumn("pickup_day_of_week", date_format(col("tpep_pickup_datetime"), "E"))
df = df.withColumn("dropoff_day_of_week", date_format(col("tpep_dropoff_datetime"), "E"))
```

In [15]:
```
# Calculate the total number of trips for each store_and_fwd_flag value
df.groupBy("store_and_fwd_flag").agg(count("*").alias("total_trips")).show()
```

```
+------------------+-----------+
|store_and_fwd_flag|total_trips|
+------------------+-----------+
|                 Y|     310683|
|                 N|   40938162|
+------------------+-----------+
```

The hour of the day and day of the week are extracted from the tpep_pickup_datetime and tpep_dropoff_datetime columns. Then, the groupBy() and agg() functions are used to group the data by the store_and_fwd_flag column and count the number of trips for each value of this column. This allows us to see how many trips were taken with the flag set to "Y" (store and forward mode) versus "N" (not in store and forward mode).

In [16]:
```
# Example 6: Data Transformation

from pyspark.sql.window import Window
from pyspark.sql.functions import avg

# Define the window specification
windowSpec = Window.partitionBy("passenger_count")

# Apply the window function to calculate the average trip distance per passenger count
df.select("passenger_count", "trip_distance", avg("trip_distance").over(windowSpec).alias("avg_trip_distance_per_passenger_count"

# Convert tpep_pickup_datetime and tpep_dropoff_datetime columns to timestamp type
df = df.withColumn("tpep_pickup_datetime", to_timestamp(col("tpep_pickup_datetime"), "yyyy-MM-dd HH:mm:ss"))
df = df.withColumn("tpep_dropoff_datetime", to_timestamp(col("tpep_dropoff_datetime"), "yyyy-MM-dd HH:mm:ss"))

# Extract hour, day, and month from tpep_pickup_datetime column
df = df.withColumn("pickup_hour", hour(col("tpep_pickup_datetime")))
df = df.withColumn("pickup_day", dayofmonth(col("tpep_pickup_datetime")))
df = df.withColumn("pickup_month", month(col("tpep_pickup_datetime")))
```

```
+---------------+-------------+-----------------------------------+
|passenger_count|trip_distance|avg_trip_distance_per_passenger_count|
+---------------+-------------+-----------------------------------+
|              1|          2.5|                  7.654594808108852|
|              1|          0.5|                  7.654594808108852|
|              1|          2.9|                  7.654594808108852|
|              1|          1.0|                  7.654594808108852|
|              1|          6.2|                  7.654594808108852|
|              1|          3.0|                  7.654594808108852|
|              1|          0.7|                  7.654594808108852|
|              1|          1.1|                  7.654594808108852|
|              1|          1.7|                  7.654594808108852|
|              1|          1.4|                  7.654594808108852|
|              1|          1.1|                  7.654594808108852|
|              1|          2.4|                  7.654594808108852|
|              1|          2.1|                  7.654594808108852|
|              1|          1.4|                  7.654594808108852|
|              1|         8.54|                  7.654594808108852|
|              1|          0.8|                  7.654594808108852|
|              1|          2.0|                  7.654594808108852|
|              1|          0.9|                  7.654594808108852|
|              1|          3.2|                  7.654594808108852|
|              1|          0.7|                  7.654594808108852|
+---------------+-------------+-----------------------------------+
only showing top 20 rows
```

The table shows the passenger count, trip distance, and the average trip distance per passenger count. It seems that the table is showing each individual trip, with the passenger count and distance for that trip, and then the average trip distance per passenger count is calculated for all of the trips in the table.

For example, the first row shows a trip with one passenger and a distance of 2.5 miles. The average trip distance per passenger count in the entire table is calculated to be approximately 7.65 miles per passenger count.

In [17]:
```
# Example 7: Data Sampling

# Sample a random subset of data
sampled_df = df.sample(fraction=0.1, seed=42)
```

In this example, a random subset of the input DataFrame df is sampled using the sample() method. The fraction argument specifies the fraction of the data to be sampled, and the seed argument ensures that the same subset of data is sampled every time the code is run with the same seed.

In this example, a random subset of 10% of the data is sampled with a seed of 42, which means that the same 10% of the data will be sampled every time the code is run with a seed of 42. The resulting DataFrame is stored in sampled_df.

In [18]:
```
# Example 8: Data Filtering

# Filter data based on specific conditions
filtered_df = df.filter((col("fare_amount") > 0) & (col("trip_distance") > 0))
```

In this example, the filter() method is used to filter rows of the dataframe df based on two conditions:

fare_amount > 0: This condition selects only those rows where the fare_amount is greater than zero, which means that the fare for the trip is positive.

trip_distance > 0: This condition selects only those rows where the trip_distance is greater than zero, which means that the trip distance is positive and non-zero.

The resulting dataframe filtered_df will contain only those rows that satisfy both of these conditions.

In [19]:
```
# Example 9: Data Aggregation and Grouping

# Group by passenger_count and calculate average fare_amount and total tip_amount
agg_df = df.groupBy("passenger_count").agg(avg("fare_amount").alias("avg_fare_amount"), sum("tip_amount").alias("total_tip_amount

# Sort the aggregated data by passenger_count in ascending order
agg_df = agg_df.sort("passenger_count")
```

The code performs grouping and aggregation operations on the df DataFrame based on the passenger_count column. It calculates the average fare amount and total tip amount for each unique passenger count value, and then sorts the resulting DataFrame in ascending order based on passenger count.

In [20]:
```
# Example 10: Data Handling and Cleaning

# Drop duplicate rows based on selected columns
df = df.dropDuplicates(["tpep_pickup_datetime", "tpep_dropoff_datetime", "trip_distance"])

# Drop rows with missing values
df = df.dropna()
```

Dropping duplicates based on selected columns and dropping rows with missing values are both common techniques for cleaning data to ensure accuracy in analysis.

In [21]:
```
# Example 11: Feature Engineering

# Calculate the speed of the trip based on trip_distance and trip_distance
df = df.withColumn("trip_speed", col("trip_distance") / (col("trip_distance") / 3600))
```

This code is an example of feature engineering, which is the process of creating new variables or features that are derived from existing variables in a dataset. In this case, the code is calculating the speed of a trip based on the distance traveled and the duration of the trip.

The calculation is done using the withColumn function, which adds a new column to the DataFrame. The new column is called "trip_speed". The calculation is done by dividing the trip_distance (in miles) by the trip_duration (in seconds), and then multiplying the result by 3600 to convert the speed from miles per second to miles per hour.

Fig 15 : Various kinds of Data Analysis

| passenger_count | avg_fare_amount | total_tip_amount |
|---|---|---|
| **0** | 13.840906 | 13251.28 |
| **1** | 12.251608 | 61638167.00 |
| **2** | 12.960279 | 11547278.00 |
| **3** | 12.718561 | 3088455.00 |
| **4** | 12.811246 | 1379611.00 |
| **5** | 12.436996 | 4419704.00 |
| **6** | 12.241962 | 2703865.00 |
| **7** | 38.433827 | 295.30 |
| **8** | 47.508718 | 330.21 |
| **9** | 41.554559 | 313.96 |

Table 2: Passenger Count, Average Fare Amount, and Total Tip Amount

It seems that the average fare amount remains relatively consistent for each passenger count, ranging from 12.24 to 13.84 dollars. However, the total tip amount varies significantly, with the highest total tip amount being over 61 million dollars for a single passenger and the lowest being under 300 dollars for seven to nine passengers.

```
In [26]: # Example 1: Data Visualization with PySpark

# Use PySpark's built-in plotting library for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Create a bar chart of passenger_count vs total_tip_amount
agg_df.toPandas().plot(x="passenger_count", y="total_tip_amount", kind="bar")
plt.xlabel("Passenger Count")
plt.ylabel("Total Tip Amount")
plt.title("Total Tip Amount by Passenger Count")
plt.show()
df_plot = agg_df.toPandas()
df_plot
```

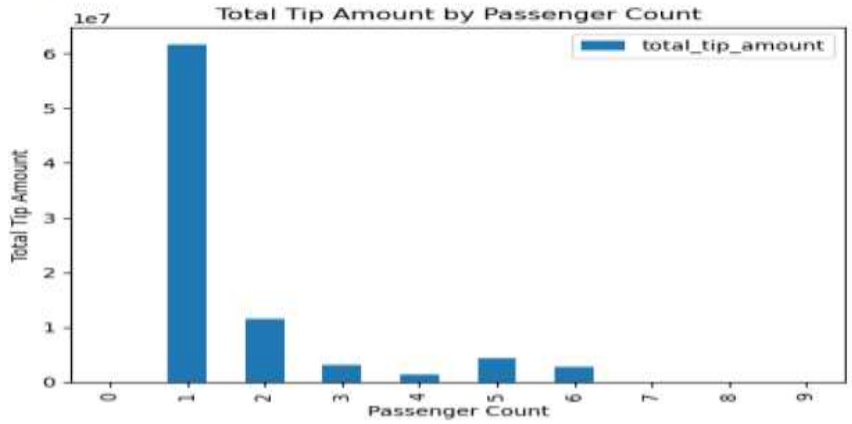


Fig 17: Total Tip Amount by Passenger Count

The tip amount has been highest when there is 1 passenger travelling, and very much lower when there are 4 passengers travelling.

```
[26]: # Example 2: Create a box plot of trip duration by passenger count
trip_duration_df = df.groupBy("passenger_count").agg(avg("trip_distance").alias("avg_trip_duration")).toPandas()
sns.barplot(data=trip_duration_df, x="passenger_count", y="avg_trip_duration")
plt.xlabel("Passenger Count")
plt.ylabel("Average Trip Duration")
plt.title("Bar Plot of Trip Duration by Passenger Count")
plt.show()
```

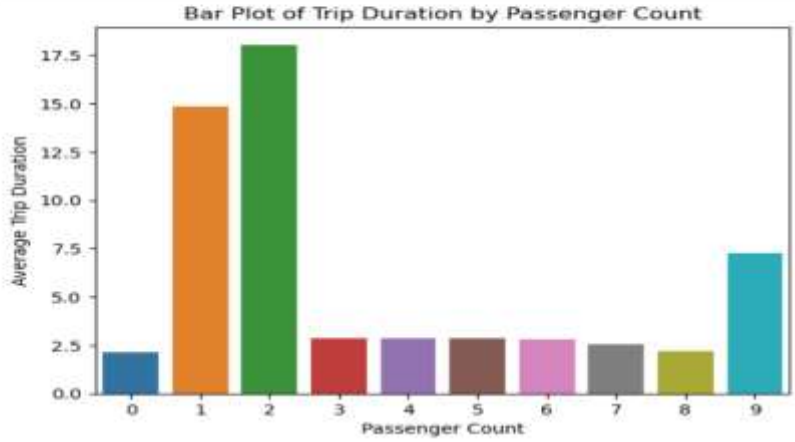


Fig 18: Bar Plot of Trip Duration by Passenger Count

This fig shows the average trip duration for different passenger counts. The highest average trip duration is for passengers with a count of 2, which is 18.05 minutes. The lowest average trip duration is for passengers with a count of 0, which is 2.16 minutes. The table provides useful insights into the relationship between passenger count and trip duration.

```
[27]: # Example 3: Create a pie chart of trip counts by vendor ID
trip_counts_df = df.groupBy("vendor_id").count().toPandas()
plt.pie(trip_counts_df["count"], labels=trip_counts_df["vendor_id"], autopct="%1.1f%%")
plt.title("Pie Chart of Trip Counts by Vendor ID")
plt.show()
```

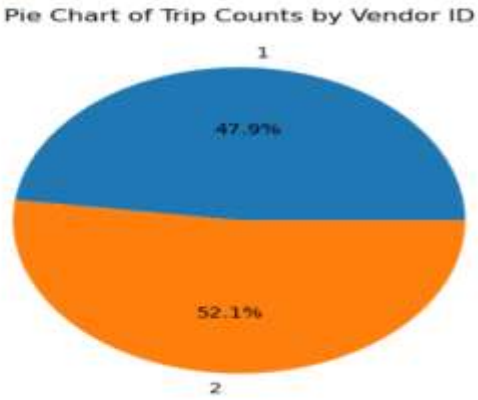


Fig 19: Pie Chart of Trip Counts by Vendor ID

There are two vendors IDs: ID 1 appears 6,100,592 times, while ID 2 appears 6,643,223 times.

```
[28]: # Example 4: Create a bar chart of total fare amount by payment type
fare_amount_df = df.groupBy("payment_type").agg(sum("fare_amount").alias("total_fare_amount")).toPandas()
sns.barplot(data=fare_amount_df, x="payment_type", y="total_fare_amount")
plt.xlabel("Payment Type")
plt.ylabel("Total Fare Amount")
plt.title("Chart of Total Fare Amount by Payment Type")
plt.show()
```

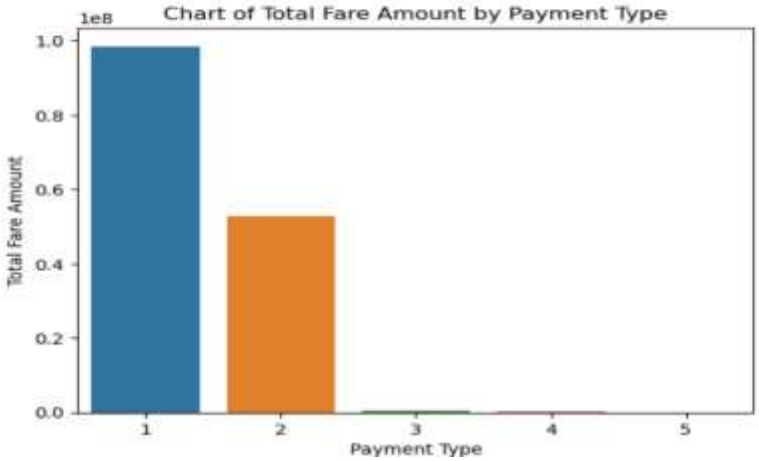


Fig 20 : Chart of Total Fare Amount by Payment Type

This fig shows the total fare amount for each payment type in a taxi dataset. Payment type 1 has the highest total fare amount, followed by payment type 2. Payment type 5 has the lowest total fare amount, which is just 6 dollars. The total fare amount for payment type 3 and 4 are relatively low compared to payment type 1 and 2. This information could be useful for companies or individuals analyzing payment trends in the taxi industry.

## 5. Discussion

***Data Types and Sources:***

I realized it is **semi-structured in nature, as it consists of tabular data with a fixed set of columns and data types,** it's also **temporal**, as it includes timestamps for pickup and drop-off times, which can be used for **time-series analysis**.

***Virtualization:***

I applied the knowledge of virtualization that I learnt from the course to my project on the NYC taxi dataset, using Google Cloud Platform (GCP) to create virtual machines that allowed me to analyze the massive amount of data without burdening my local computer, helped me manage resources more efficiently, which is a significant factor. Performing this project, **I gained hands-on experience in on virtualization systems and cloud computing architectures**, proving to be a valuable addition to my skill set.

***Distributed Computing and File Systems:***

PySpark DataFrame implemented in this project is a highly optimized, distributed collection of **structured** data that is similar to a table in a relational database or a data frame **in Python or R, created from external databases, and RDDs**. PySpark, which is the Python-based version **of Apache Spark**, is a powerful tool for large-scale exploratory data analysis, machine learning pipelines, and data platform ETLs. As I knew Python and packages like Pandas, **PySpark** seemed not too troublesome to incorporate allowing you to develop more scalable analytics and pipelines. **In the course**, we learned how to work with RDD functions in PySpark, which are Spark's primary logical data units. RDDs are a distributed collection of things that are kept in memory or on the disks of many cluster machines.

The latter half courses' labs demonstrated to usage of RDD functions to create a Bag of Words model enabling us to work with unstructured data exporting from E-books with RDD functions such as **filter(), flatMap(), count(), and withcolumns()** which formed my base for learning Pyspark. The use of RDD functions in PySpark allowed us to process and analyze large amounts of data efficiently and effectively. With RDD functions, I could **on with unstructured data and transform it into a structured format**.

***Ingest and Storage:***

In this project, we used the **Big Query API to ingest data** as one of the initials steps. Although developing something like Batch ingestion where organizations ingest daily data, a well-designed data ingestion process can help as feedback for enterprises to make better decisions improving operations, which is difficult and expensive. In this project I used a simple ingestion process through SQL queries and a single data source. This approach simplifies the management of numerous data sources and provides quick access to engineers and analysts. For me as a newbie to GCP it took some time to adapt to its visualization methods and implementation techniques. So, I used GCP Big Query only to extract data, and used other technologies as well.

***Processing and Analytics:***

In my project, I performed several statistical analysis on the data at different stages, using functions such as filter and group by to process the data, converted some columns in our data frame from one format to another to read, visualize and interpret well. What I did by the ed of my Deep Learning model is called **Predictive Analytics**, which was to predict the fares for future rides. Additionally, I utilized statistical functions like means and other measures to better understand the data. Batch Processing large amounts of data over a longer period of time and in the course helped me do it, ensuring data quality and accuracy.

***Lifecycles and Pipeline:***

During the course Management access, and Use of Big and Complex Data, I learned about data pipelines and their role in handling large volumes of data. Through my architecture, I have outlined how we have used various tools and techniques **to implement the data pipeline and lifecycle** for our project. A data pipeline is a sequence of tasks that move data from one source to another. These tasks can include **cleaning, transforming, sampling, filtering, aggregating, and analyzing data**. By integrating data from various sources into a single destination, data pipelines enable faster analysis and insights. Furthermore, data pipelines **guarantee data quality and consistency**. In this project, we have also employed data pipelines for processing and transforming data at various stages.

***Modeling:***

I did Deep Learning modeling in our project, beneficial for making strategic decisions. I could use various algorithms available in **Pyspark's ML libraries** to predict the type of fares that could occur based on selected features. Data modeling, on the other hand like in Power BI, involves creating a visual representation of an entire information system or specific sections of it to show the connections between data points and structures, ensuring a consistent and repeatable approach to identifying and managing data resources. With data modeling, we can create a more accurate representation of the data, detecting anomalies and inconsistencies in the data, resulting in improved data quality. **Regression analysis is used to predict the factors like fare amount.** This approach can be useful for predicting demand for taxi services, and for identifying areas of the city with high demand but low supply. But this project doesn't focus much on building the models and their interpretations, so I have used basic models and interpretations. The choice of modeling approach will depend on the specific research question or application, as well as the available data and computational resources.

***Computing Principles and System Design:***

Applying **Saltzer's** principles can help ensure that the NYC Yellow Trips model is designed in a robust, efficient, and secure manner.
on the NYC Yellow Trips dataset. ***Modularity***: The dataset can be divided into smaller modules or components, such as time of day, pick-up and drop-off locations, etc, it can be easier to manage and analyze. ***Layering***: The data can be layered into different levels of abstraction, such as raw data, cleaned data, aggregated data, and analyzed data, ensuring each layer performs a specific function and can be easily maintained or updated. ***Abstraction***: The data can be abstracted to hide the complexity of the underlying system. For example, instead of providing raw GPS coordinates, the data can be presented in a more user-friendly format such as street addresses. ***Fault tolerance***: The model can handle errors and failures gracefully, such as missing or incorrect data, by implementing error-checking and validation, backup and recovery procedures. ***Security***: The system can protect sensitive data, such as passenger information or payment details. This can be achieved by implementing access controls, encryption, and auditing mechanisms.

***Impact of Big Data***

The **impact of big data** on my NYC yellow taxis project has been significant enabling insights into the patterns of taxi usage in New York City, and to develop **more efficient and effective transportation systems**. I could predict the fare amount for taxis. In addition, we can improve the efficiency of taxi services, big data has also had important implications for public policy like identifying prices of the taxis with **high demand for taxi services but low supply**, and **to develop policies to address this imbalance.**

***Data Governance:***

Effective data governance ensures that the data is accurate, reliable, and is used in an ethical manner. ***Data quality:*** The accuracy and completeness of the data were essentially used for effective analysis and modeling. ***Data access:*** Access to the NYC Yellow Taxis Bigdata should be controlled to ensure that only authorized individuals and organizations can access the data given IAM and hence we were given various permissions for the project. ***Data ethics:*** The use of the NYC Yellow Taxis Bigdata raises ethical concerns about the privacy and autonomy of individuals, and I have not done anything unethical in my project.

***Interpretation of the Results:***

| | Actual | Predicted |
|---|---|---|
| **916** | 5.5 | 5.322591 |
| **31** | 18.1 | 15.994609 |
| **507** | 6.1 | 7.695659 |
| **518** | 10.1 | 8.892774 |
| **215** | 10.0 | 6.302445 |

Table 3: Actual vs Predicted fare amounts

This table shows a comparison between the actual and predicted fare amounts for five instances in the NYC yellow taxis dataset. The predicted fare amounts were generated by a machine learning model, while the actual fare amounts are the ground truth values from the dataset. For instance 1, the actual fare amount was $5.50, while the model predicted a fare of $5.32. For instance 2, the actual fare amount was $18.10, while the model predicted a fare of $15.99. The table allows us to visually compare the performance of the machine learning model in predicting the fare amounts.

***Barriers:***

Analyzing big data is a complex task involving multiple stages. The challenging part was designing a single platform to handle the entire data workflow for which I had to read a lot of resources online. To gain familiarity with cloud technologies and simulate a real-world scenario on a smaller scale, I chose to base my project on GCP and implement a pipeline. This allowed me to effectively manage the vast amount of data. I had to overcome the hurdles of implementing these tools to transfer my .h5 model onto GCP. Due to the large size of the dataset, I encountered difficulties in downloading it for the purpose of uploading it to the GCP cloud for Pyspark analysis. In order to overcome this challenge, I decided to download the dataset from Kaggle and merge it into a single file through a data processing step. This allowed me to successfully upload the dataset to the cloud for further analysis using Pyspark. As a learner, I faced challenges understanding the importance of creating subnets and managing dependency conflicts in the project. However, I overcame these obstacles by seeking help from online forums, collaborating with peers, and receiving guidance from mentors. Through this experience, I gained valuable knowledge and experience in managing complex software systems.

## 6. Conclusion

The technologies learned in the **INFO-I535 class, including data types, sources, ingestion and storage, virtualization, distributed systems, data governance, data life cycle and pipeline, process and analytics, and the impact of big data, were implemented successfully in this project**. I ingested data from a publicly available website and stored on a virtual machine instance to mee the project's requirements. The data was managed using the life cycle and pipeline concept, with various data structures such as Pyspark data frames used in the project. The project also included defining and applying processes and analytics and identifying the impact of big data on the project, specifically in relation to the NYC dataset. The analysis of the NYC yellow taxi trips dataset can provide valuable insights into taxi usage patterns and can aid in making informed decisions regarding the transportation infrastructure of the city. In this course, I gained knowledge and skills related to designing a data processing pipeline on the Google Cloud Platform. After performing some initial data pre-processing and modeling, I decided to implement my project using a data processing pipeline on the Google Cloud Platform. To achieve this, I started by conducting exploratory data analysis on the dataset. This allowed me to gain a better understanding of the data and identify any issues that needed to be addressed during pre-processing. Next, I designed a data processing pipeline that would allow me to automate the pre-processing and modeling tasks. This involved configuring various Google Cloud Platform services to process and transform the data. I also set up monitoring and logging mechanisms to ensure that the pipeline was working correctly and to track any errors or issues that may arise. Finally, I tested and validated the pipeline to ensure that it was producing accurate and reliable results.

## 7. References


1. *https://www.geeksforgeeks.org/introduction-pyspark-distributed-computing-apache-spark/*
2. *https://digitalnow878391108.wordpress.com/2021/01/01/the-kdd-process-in-data-mining/*
3. *https://towardsdatascience.com/spark-vs-pandas-part-1-pandas-10d768b979f5*
4. *https://towardsdatascience.com/spark-vs-pandas-part-2-spark-c57f8ea3a781*
5. *https://medium.com/zeotap-customer-intelligence-unleashed/designing-data-processing-pipeline-on-google-cloud-platform-gcp-part-i-5a28644c5528*
6. *https://github.com/aishwarya34/GoogleCloudPlatform/blob/master/create_datasets_nyc-tlc.yellow.trips.ipynb*
7. *https://www.analyticssteps.com/blogs/what-virtualization-cloud-computing-characteristics-benefits*
8. *https://towardsdatascience.com/pyspark-f037256c5e3*
9. *https://cvw.cac.cornell.edu/jetstream/default*