Increment - 2

Uber Fare Prediction

TEAM MEMBERS:

Sailesh Pilla (11593815)

Yaswanth Sai Satish Sreerama (11601948)

Kishore Kumar Paila (11600316)

Introduction:

Our Project is about Uber, the largest taxi company in the world, serving thousands of consumers every day. I noticed some open-source data from Kaggle. Due to the service's global reach, it is necessary to precisely compute fare pricing considering several factors, including distance, weather, time, and demand. We are trying to predict fares for customers concerning every other aspect, as we want perfect food once they book a ride. The user needs to know how much the charge is from a specific location to the destination location.

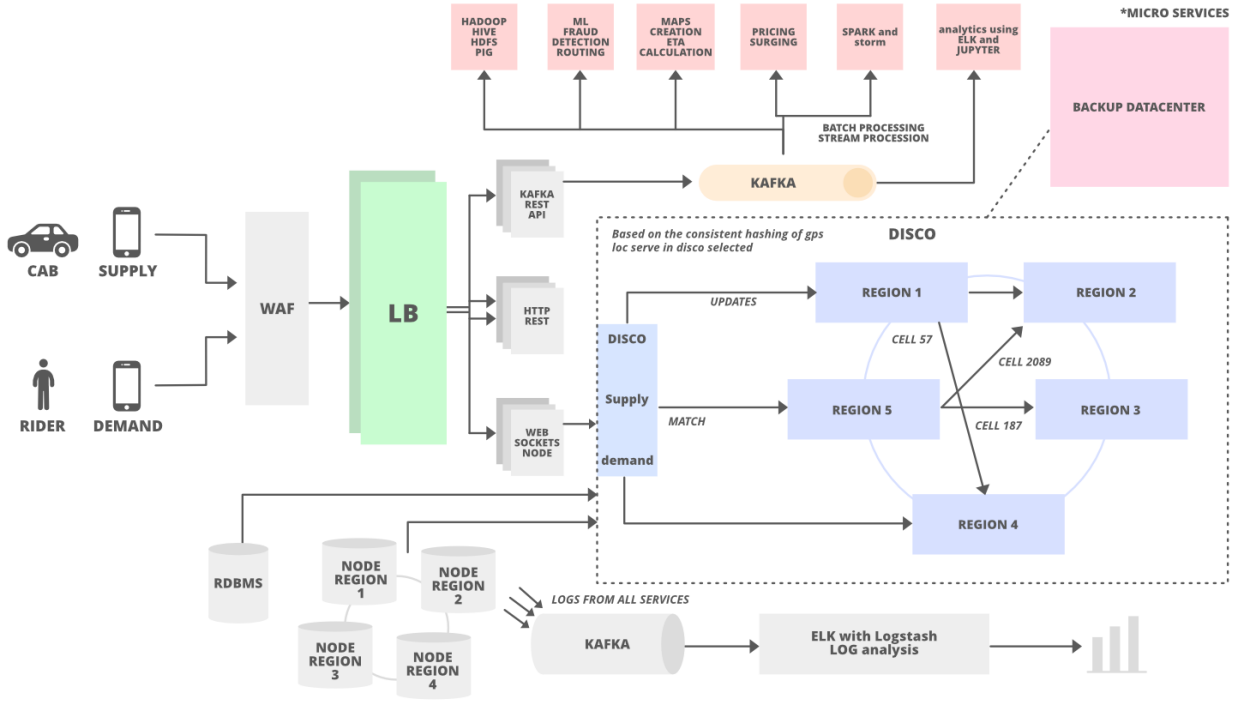
Background:

Uber Technologies, Inc.(uber) was known as uber cabs and was invented by Garret Cam; a computer program in San Francisco that provides a transport facility by allowing users to book a car like a taxi. Camp and his friends spent **$800** to hire a driver, which he felt was heavy. He wanted to reduce the cost of transportation, and that’s when he came up with the idea that sharing the price with other people would make it affordable; they built a prototype with his friends. In 2011

uber started, and it was almost 1.5 * times more of a taxi, but it has its advantages. As time progressed, they moved into the food industry by setting up Uber Eats, a food delivery app. Uber has been spread worldwide, and uber is a top transportation company. They have used multiple machine learning models to improve the accuracy of fare prediction.

Uber Model:

Architectural Diagram of UBER



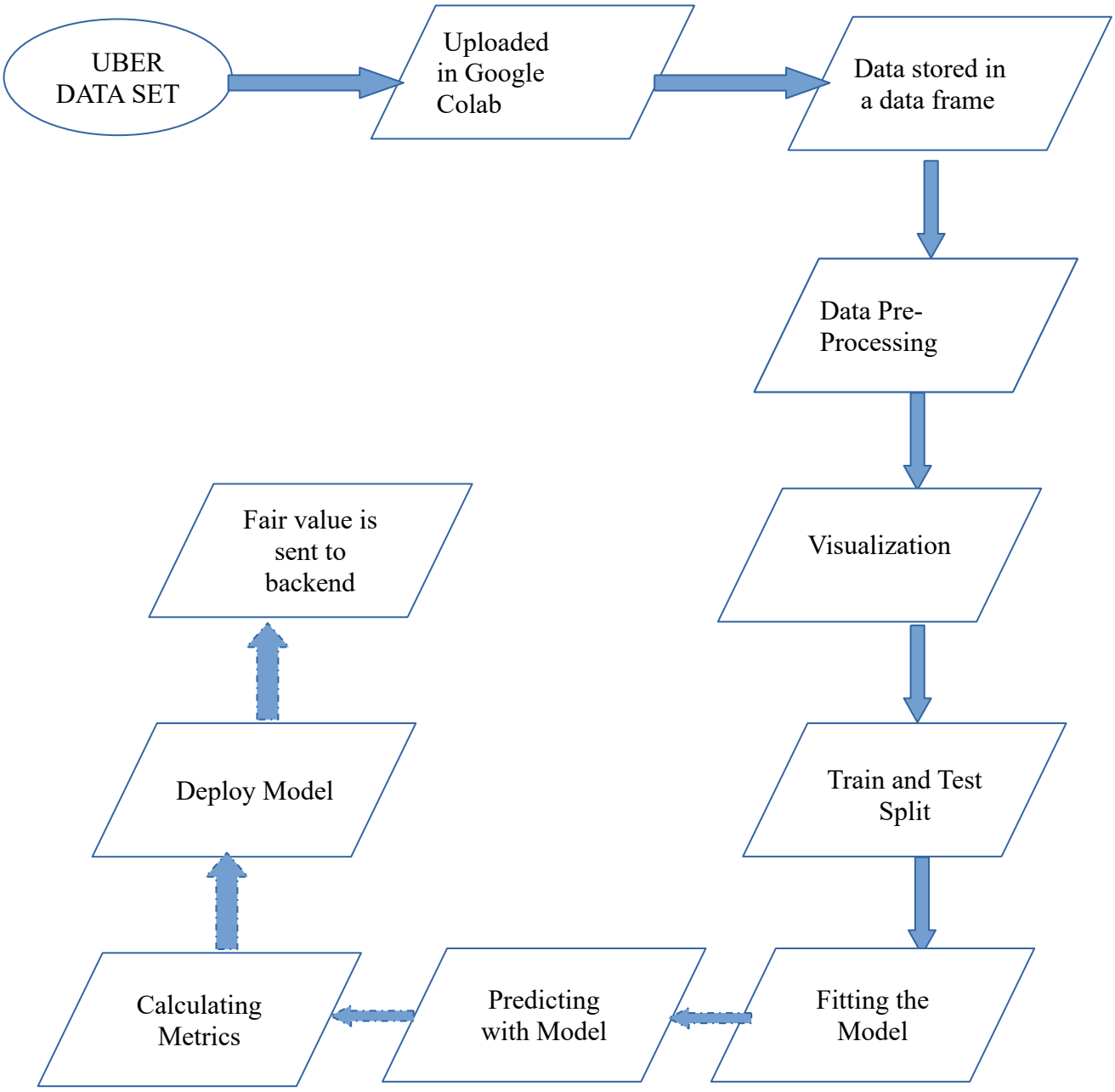
As per the architectural diagram of Uber, This is more like a demand (rider) and supply (cabs) policy as the user demands from his uber mobile app, and he gets his need supplied by uber as they use a web application framework for security reasons to detect robots and regions which are not supported by the user and the user sends through the WAF and then it goes through a load balancer which checks through different layers based on IP of the load balancer. KAFKA Rest API is an endpoint to consume all locations of each data for every 4 secs. Those details will be sent to DISCO to keep all the states of the cabs alive and to show them to users. Every call happens with a firewall and using KAFKA Ap; it goes to different servers. A copy is also saved in the database and

dispatch optimization to protect and keep up to date with the location of the respective cabs. Web sockets help synchronize client-to-server data transfer. The dispatch system is written in Node.js. When the user requests a driver, it goes into the web socket, and the requirement will be known for either cab or a ride. It will provide all the information like where is the user's location and destination location and what type of car everything is sent to the backend, and once the APIs are forwarded to the backend using Hadoop/HDFS, which is essential data that handles large amounts of data and checks with Maps ETA and then it uses the model's algorithm to see how much is the prediction concerning different factors that will be calculated in the backend depending on the changes that are being done and then at last fare will be calculated with the help of ML models like basic regression to neural networks.

Workflow Diagram

Initially, data is taken from Kaggle and uploaded to collab, which into a data frame, then started pre-processing like statistical analysis by checking the data shape, head, and description. Data cleaning is also needed as data might contain duplicate information that can be the same rows or identical columns of the data. It might have missing values, which must be filled with repeated mean, median, mode, or zero values per the dataset. We need to check for the outliers once the data is cleaned, and once it's done, remove outliers. Create necessary features concerning the existing features. Now data is split into train and test data, and once the information is broken, the model must be fitted concerning the Machine Learning algorithm. Whatever values are provided on the training dataset, and once it is trained, now use the trained model and predict it with test data. Then calculate the necessary metrics and deploy the model as the fare is calculated. This model can be used in the architecture to send that fare value to the backend.

Workflow Diagram



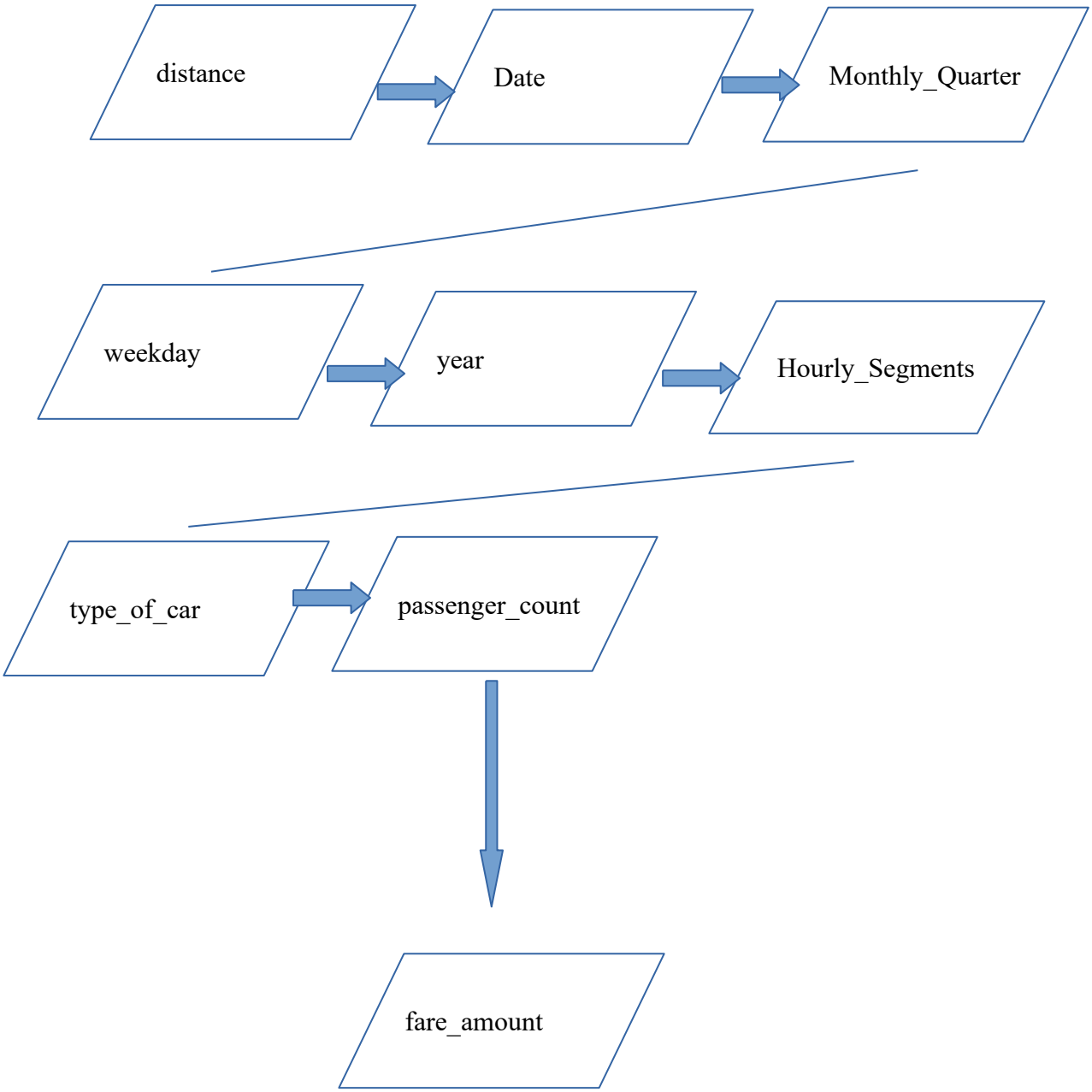
Dataset: (Uber Dataset)

The attained data set contains over two lakh travellers and spans 2009 to 2015.

- key - a unique identifier for each trip
- fare_amount - the cost of each trip in USD
- pickup_datetime - date and time when the meter was engaged
- passenger_count - the number of passengers in the vehicle
- pickup_longitude - the longitude where the meter was engaged
- pickup_latitude - the latitude where the meter was engaged
- dropoff_longitude - the longitude where the meter was disengaged
- dropoff_latitude - the latitude where the meter was disengaged

Our Dataset has different features, including latitude and longitude values and pickup date time, along with passenger count and fare amount. My dataset consists of 200000 rows and nine columns and has missing values.

Detail design of feature with Diagrams:



From the diagram, the features are derived after necessary pre-processing steps as we calculated the distance from the elements using latitude and longitude of pickup and drop off. Then, with the pickup date time, we figured out the year, month, date, and time, as the monthly quarter's features and then computed the fare amount using these features.

Analysis of data:

1. Thoroughly understand the dataset and determine whether the data is correct, and if not, perform a clean-up.

2. Create a regression model over the training data that has been separated.

3. Evaluate the model and compare R2 and Root mean squared error scores to find the projected model error so that you may understand how accurate your predictions were.

2. Checking head of the datasets

```
1 uber_data.head()
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

Info of the data:

```
1 uber_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         200000 non-null  int64
 1   key                200000 non-null  object
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

Describing Data:

```
uber_data.describe()
```

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

Several columns and rows in the data:

```
uber_data.shape
```

```
(200000, 9)
```

Haversine function to return distance by taking pickup and dropoff locations:

```
def haversine (lon_1, lon_2, lat_1, lat_2):

    lon_1, lon_2, lat_1, lat_2 = map(np.radians, [lon_1, lon_2, lat_1, lat_2])  #Degrees to Radians


    diff_lon = lon_2 - lon_1
    diff_lat = lat_2 - lat_1


    km = 2 * 6371 * np.arcsin(np.sqrt(np.sin(diff_lat/2.0)**2 +
                                      np.cos(lat_1) * np.cos(lat_2) * np.sin(diff_lon/2.0)**2))

    return km
```

```
uber_data['distance']= haversine(uber_data['pickup_longitude'],uber_data['dropoff_longitude'],
                                 uber_data['pickup_latitude'],uber_data['dropoff_latitude'])
```

Retrieving year, month weekday, and hour by using to_date from pandas:

```
uber_data.pickup_datetime=pd.to_datetime(uber_data.pickup_datetime)
```

```
uber_data['year'] = uber_data.pickup_datetime.dt.year
uber_data['month'] = uber_data.pickup_datetime.dt.month
uber_data['weekday'] = uber_data.pickup_datetime.dt.weekday
uber_data['hour'] = uber_data.pickup_datetime.dt.hour
```

Dividing the to_date into segments and monthly quarters for a better understanding of data:

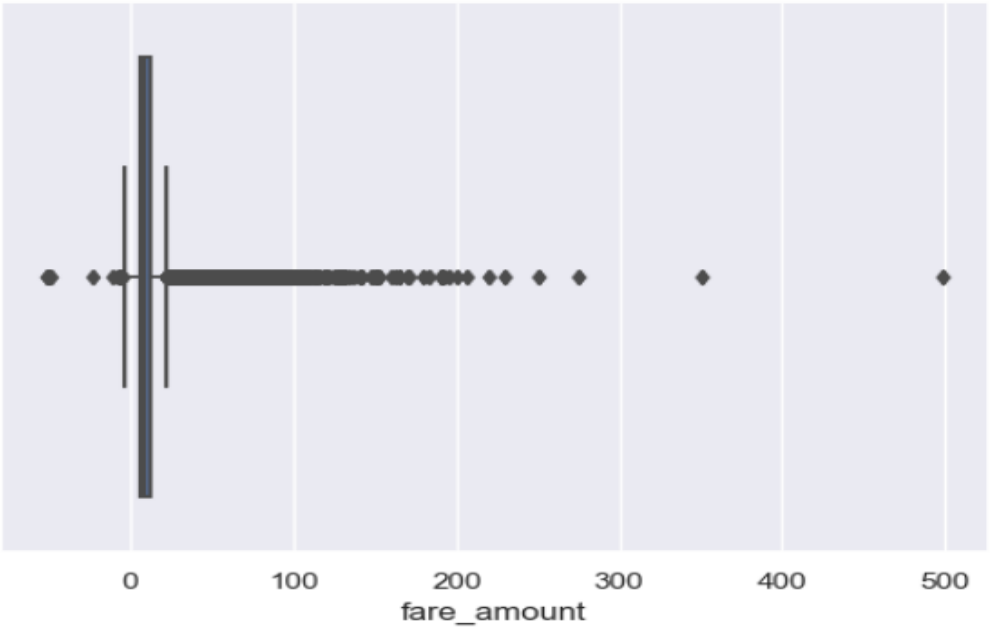
```
uber_data['Monthly_Quarter'] = uber_data.month.map({1:'Q1',2:'Q1',3:'Q1',4:'Q2',5:'Q2',6:'Q2',7:'Q3',
                                    8:'Q3',9:'Q3',10:'Q4',11:'Q4',12:'Q4'})
uber_data['Hourly_Segments'] = uber_data.hour.map({0:'H1',1:'H1',2:'H1',3:'H1',4:'H2',5:'H2',6:'H2',7:'H2',8:'H3',
                                   9:'H3',10:'H3',11:'H3',12:'H4',13:'H4',14:'H4',15:'H4',16:'H5',
                                   17:'H5',18:'H5',19:'H5',20:'H6',21:'H6',22:'H6',23:'H6'})
```

Visualizations representing the data:

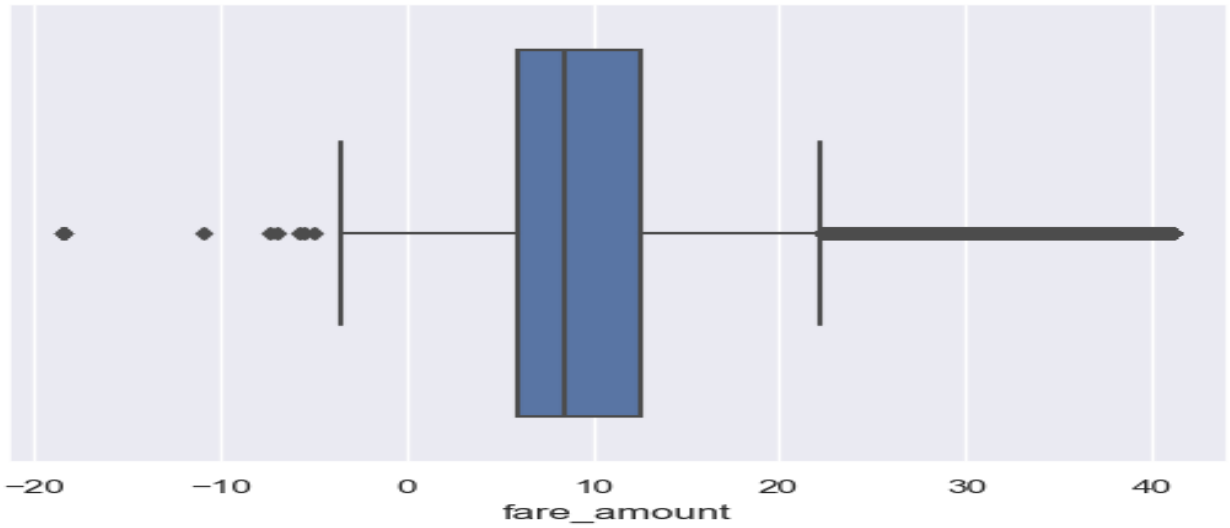
Box Plot with Outliers:

```
sns.boxplot(x=uber_data['fare_amount'])
```

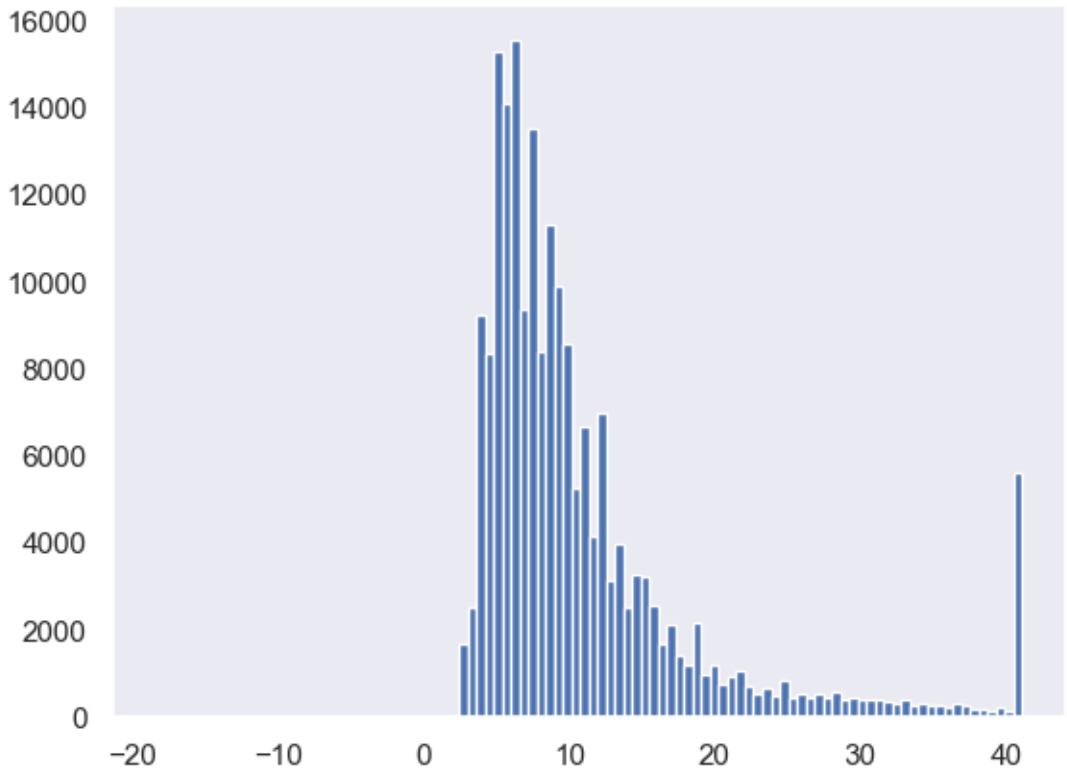
```
<AxesSubplot: xlabel='fare_amount'>
```



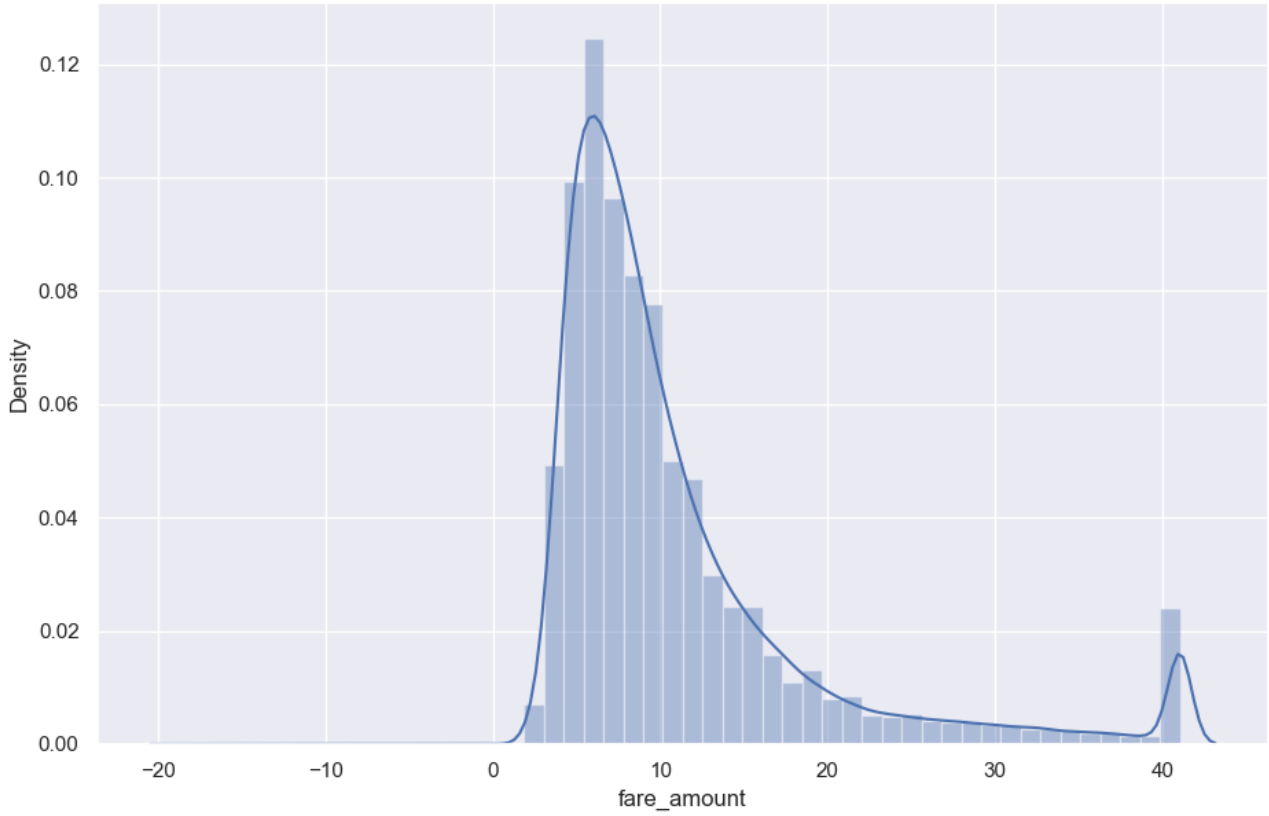
Box plot after removing Outliers:



Histogram:

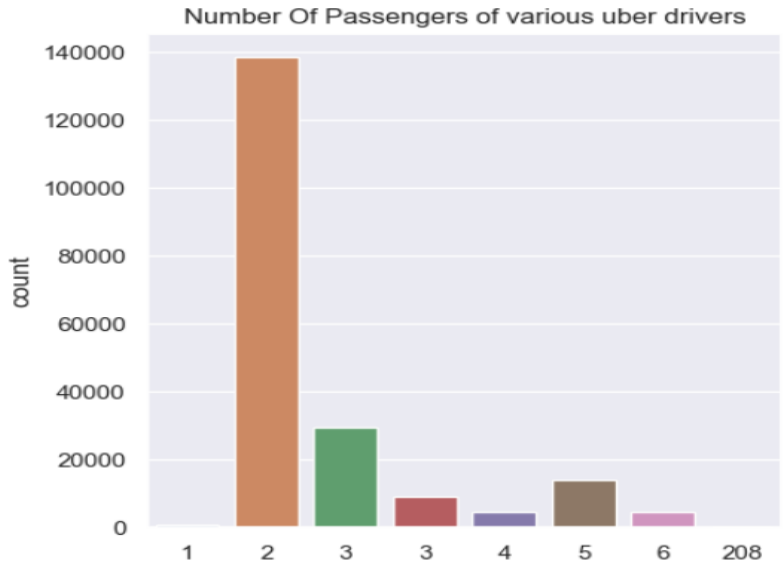


Data Distribution using Distplot:



Count Plot:

```
fig, ax = plt.subplots(figsize = (5, 5))
sns.countplot(x = uber_data.passenger_count.values, data=uber_data)
labels = [item.get_text() for item in ax.get_xticklabels()]
labels[0] = '1'
labels[1] = '2'
labels[2] = '3'
ax.set_xticklabels(labels)
ax.set_title("Number Of Passengers of various uber drivers")
plt.show()
```

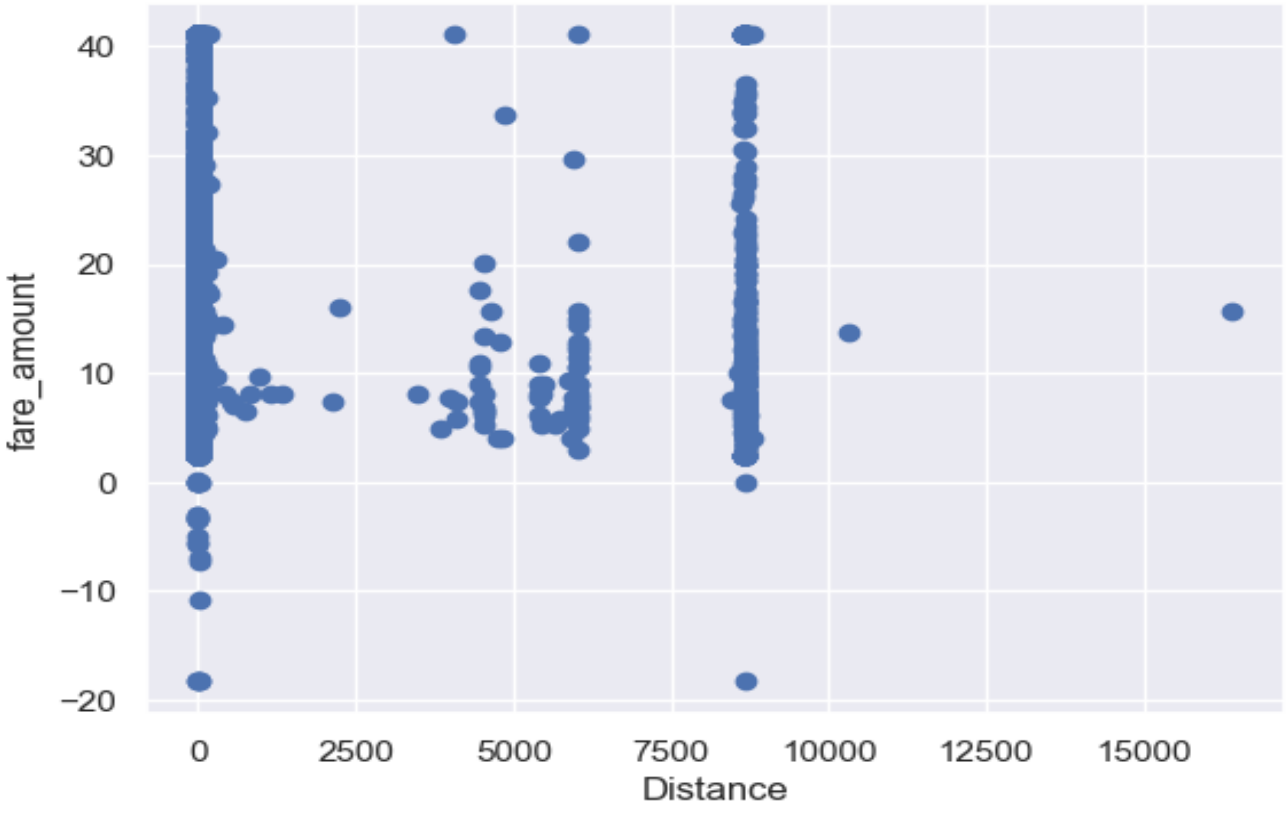


Heat Map:

```
#heat map
plt.figure(figsize=(10,5))
hm = sns.heatmap(uber_data.corr(),annot=True)
plt.show()
```

```
C:\Users\yaswa\AppData\Local\Temp\ipykernel_10192\2389066797.py:3: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  hm = sns.heatmap(uber_data.corr(),annot=True)
```

Scatter Plot:



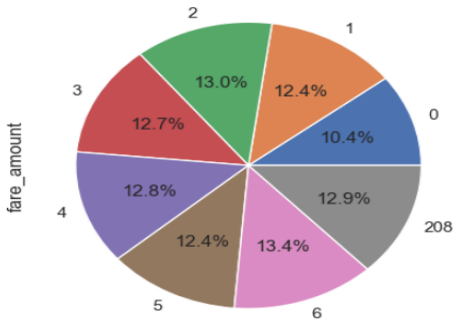
Pie Chart:

```
grouped=uber_data.groupby('passenger_count')
print(grouped['fare_amount'].mean())
```

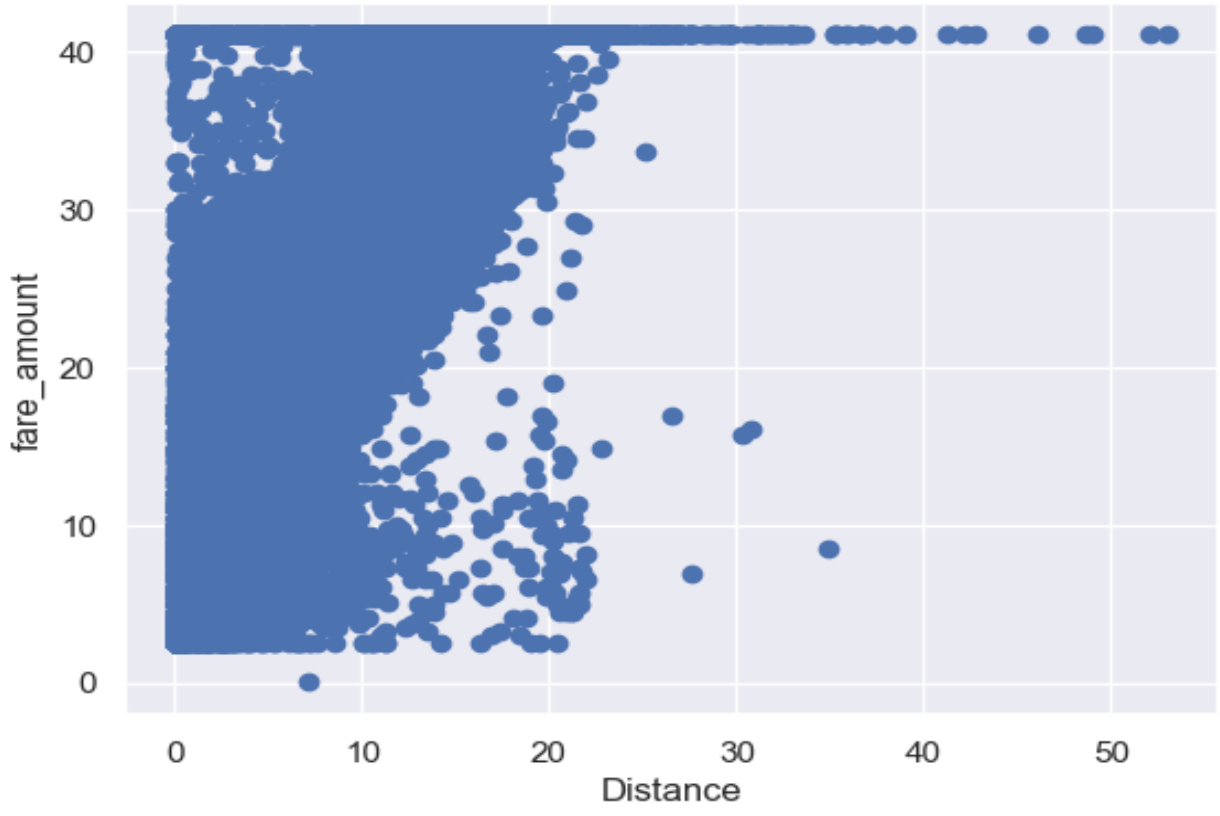
```
passenger_count
0       9.439266
1      11.254158
2      11.784452
3      11.486731
4      11.642472
5      11.199698
6      12.158537
208    11.700000
Name: fare_amount, dtype: float64
```

```
grouped['fare_amount'].mean().plot(kind='pie',y='fare_amount',autopct='%1.1f%%')
```

```
<AxesSubplot: ylabel='fare_amount'>
```



Scatter Plot after removing irrelevant rows:



Splitting the dataset into train and test sets:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Implementation:

```
from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)

print("Training set score: {:.2f}".format(l_reg.score(X_train, y_train)))
print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))
```

```
Training set score: 0.80
Test set score: 0.8014349
```

Predicted and actual values after fitting the training data into linear regression:

```
y_pred = l_reg.predict(X_test)
df = {'Actual': y_test, 'Predicted': y_pred}

from tabulate import tabulate
print(tabulate(df, headers = 'keys', tablefmt = 'psql'))
```

```
|    12.5  |    16.2194  |
|     6.5  |     7.89846 |
|     7.3  |     7.39028 |
|    19.7  |    24.4064  |
|     4.1  |     4.51103 |
|    15    |    15.3584  |
|    13.7  |    17.8741  |
|     6.1  |     6.07715 |
|     5.5  |     5.28881 |
|    31.3  |    21.9703  |
|     4.9  |     7.04995 |
|     8.1  |     4.29591 |
|     6    |     6.69743 |
|    15    |     7.17872 |
|     8.5  |    11.132   |
|    10    |    10.3526  |
|     8.5  |     7.38459 |
|    16    |    11.5931  |
|    16.9  |    17.3298  |
```

Results:

Checking the accuracy of the data with predicting:

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
#print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 2.2843774732419555
Mean Squared Error: 17.95984340408041
Root Mean Squared Error: 4.237905544497235
```

Statistical Results:

1. One Sample T-Test

```
uber_data1['fare_amount'].mean()
```

```
10.999986181927715
```

```
scipy.stats.ttest_1samp(uber_data1['fare_amount'],popmean=11)
```

```
Ttest_1sampResult(statistic=-0.0007581454529918717, pvalue=0.9993950882877489)
```

2. Two Sample paired T-Test

```
uber_data2=uber_data1['fare_amount'][:96745]
```

```
uber_data3=uber_data1['fare_amount'][96745:]
```

```
scipy.stats.ttest_rel(uber_data2,uber_data3)
```

```
Ttest_relResult(statistic=0.4252172814795518, pvalue=0.6706792331557414)
```

Analysis:

- After inspecting the dataset, I discovered that it is in CSV format with comma-separated values, which I must read. As I'm doing it in Google Colab, I'm uploading my dataset using the Google Colab library and using the browse function to upload the dataset and save it in a data frame to calculate the fare from an Uber dataset, where each feature denotes a distinct representation, I intended to use a multiple linear regression model.
- The key represents the transaction number for that trip, which will be distinct. If we want to know how a specific trip went, we can check it with the key to find out how long it took, when the pickup time was, how many passengers traveled, what kind of car was used more frequently, and whether long or short distances were traveled.
- Before removing invalid data or filling it with mode values, all necessary analyses must be completed, including checking the data's head and null values.

- Once the necessary pre-processing processes have been completed, we can build new features, such as distance calculations utilizing pickup and drop-off latitude and longitude with all four parts, and then drop the other four features to reduce the number of columns.
- Toto sees the surge in fare depending on time by pickup time column; we can create a detailed analysis by creating features like the type of car using passenger count and another feature like time of travel and beginning bins in part with morning, noon, evening, and night. Then we can drop this column and use the newly created feature as it will give more insights for the prediction.
- Once all necessary fields have been made available, mummification or one-hot encoding will turn continuous data into categorical variables. As soon as the data is prepared, we divide the X and Y variables into independent and dependent ones in a ratio of 70:30 for the train and test runs.
- Once the data has been divided, the model will need to be fitted using a multiple linear regression model once the data has been trained. Now, to determine residual errors, we predict the data for the train fare amount for which we already have a number. We then compare it to the projected value and note any differences.
- Then, use the same model on a test dataset to examine how it will predict, how accurate it will be with observed and predicted values, and how the train and test data are fitted. At this point, you should calculate the MSE, RMSE, and R Square metrics to determine how well your model predicts. By drawing the best-fit line, you may utilize any new data to predict, estimate its accuracy, and visually assess your model's performance.

Implementation: Work Completed:

- **Description:** After obtaining a dataset, we completed the necessary pre-processing procedures, followed by data analysis, visualization, and training and testing split data.

Responsibility:

- **Sailesh** conducted the necessary research to identify the best dataset to make the prediction. He then uploaded the dataset and performed a few pre-processing operations, such as computing null values, reading the dataset, and cleaning the dataset.
- **Yaswanth** completed the feature engineering portion by adding new features, deleting unnecessary columns, performing the required categorical encoding, mummifying data using a single hot encoding operation or label encoder, and completing the needed visualizations to gain insights into the removal of unnecessary features.
- **Kishore** completed the necessary visualizations, the train-test split, and the multiple linear regression model and determined the model's accuracy to apply the model to any additional dataset and calculated Statistical Values.

Work Implemented and Contributions:

- Sailesh - 30% (Data Pre-processing)
- Yaswanth – 35% (Data Cleaning and adding new features)
- Kishore – 35% (Data modeling and Statistical Approach)

Implementation Status Report:

We have implemented the linear regression to the uber dataset and predicted the Mean Square Error, Root Mean Squared Error, and T-Test for the dataset.

References:

[1]Fares Dataset

[2]Fare Amount Prediction

[3]Comparing Fares

GitHub Link: **https://github.com/YaswanthSreerama2203/UberFarePrediction**

Video Demonstration:

https://myunt-my.sharepoint.com/:v:/g/personal/yaswanthsaisatishsreerama_my_unt_edu/EbfbUYfZ7XVDoS9O0poKjOkBMRKaOEhvlJj59jcd5jLj7w