New York Taxi Trip Duration BigData Analysis

Adarsh Dudhat
University of Calgary
Calgary, Canada
adarsh.dudhat@ucalgary.ca

Ayman Shahriar
University of Calgary
Calgary, Canada
ayman.shahriar@ucalgary.ca

Anh Tuan Pham
University of Calgary
Calgary, Canada
anhtuan.pham@ucalgary.ca

Abstract—**New York city is one of the major cities of the world, and taxi cabs are a popular form of transport for its residents. That is why in this project, we have explored ways of using machine learning models to predict the duration of taxi trips when given details such as start and end coordinates, and the starting time. A dataset from Kaggle containing the data of more than 0.5 million taxi trips was used in this project. The data was cleaned, and visualized, and a positive linear correlation between the distance and trip duration could be seen in certain visualizations. A K-means clustering model was created to divide the segments into groups based on important trip data. A multiple linear regression model was then created to predict the trip duration, but this model did not have satisfactory performance. Finally, a gradient boosted trees model was created to predict the trip duration, and the performance of this model is much better than the linear regression model. This project aims to be particularly useful towards those, who want to check their trip time in NYC. For future work, we would consider experimenting with different machine learning models.**

I. INTRODUCTION

New York City (NYC) is one of the biggest cities in the world, with a population of around 18,000,000 people [1]. Due to its large population, traffic congestion is one of the most critical problems in New York. In addition, taxi cabs are a popular choice of transport when traveling within this city. That is why in this project, we chose to predict the trip duration for a taxi in New York using the "NYC taxis trip duration dataset" which can be found here and so people can get an idea of how long their taxi trip will take and plan their trips beforehand.

The current dataset is one released by the NYC Taxi and Limousine Commission, which includes pickup time, geo coordinates, number of passengers, and several other variables. The dataset contains more than 510,000 rows.

Since the dataset is from a Kaggle challenge (where people compete to use the provided data to create the best visualizations, machine learning models and reports), there are a lot of machine learning models that were built using this dataset. However, we found out that the accuracy of these models is not very high, which means that there is a possibility of developing models that can achieve higher accuracy and reliability. Moreover, the current machine learning models were built using mostly Tensorflow and other CNN techniques, while we manipulate the data and create a model using SkLearn and PySpark.

In our solution, we have created several visualizations that show a relationship between trip duration and other variables, also we demonstrated how the mean passenger count, total trips, distance and duration vary on different weekdays. Also, we have created 3 machine learning models to either gain insights into the structure of the data or try to predict the duration of a trip. That includes regression models like linear regression, gradient boosted trees, and K-means clustering.

II. BACKGROUND AND RELATED WORK

A. Technical Background

To understand the report, it requires a good understanding of the dataset, supervised and unsupervised machine learning techniques such as linear regressions, gradient boosted trees and KMeans clustering.

B. Review of existing work pertinent to Project

We used only one dataset, as opposed to combining several datasets, so there was not much work that needed to be done prior to starting with the project. But it was important to understand the dataset and values of each label column.

III. METHODOLOGY

A. Experiment Setup

To run the notebook, users should have the following python packages installed in their environment: warning, PySpark, Numpy, pandas, matplotlib, and so on. To complete the setup, please check README.md

B. Data Cleaning and Preprocessing

To clean the data, the team has decided to check for values that are not valid such as NaN, trip_duration < 0, and many others.

In [57]:
```
# Change Datatype for each column.

def change_type(df, column_name, column_type, is_time):
    if column_type == "int" or column_type == "float":
        new_df = df.withColumn(column_name, df[column_name].cast(column_type))
    elif column_type == "bool":
        new_df = df.withColumn(column_name, functions.when(df[column_name] == 'N', 0).otherwise(1))
    elif is_time == True:
        new_df = df.withColumn(column_name, to_timestamp(df[column_name], "yyyy-MM-dd HH:mm:ss"))
    else:
        new_df = df

    return new_df
```

In [58]:
```
# Calling change_type function for each column.

column_types = ["int", "time", "time", "int", "float", "float", "float", "float", "bool", "int"]
count = 0

for col in df.dtypes:
    if column_types[count] == "time":
        df = change_type(df, col[0], column_types[count], True)
    else:
        df = change_type(df, col[0], column_types[count], False)
    count += 1
```

In [59]:
```
# Printing Datatype for all columns.

for col in df.dtypes:
    print(col[0]+" , "+col[1])
```

```
vendor_id , int
pickup_datetime , timestamp
dropoff_datetime , timestamp
passenger_count , int
pickup_longitude , float
pickup_latitude , float
dropoff_longitude , float
dropoff_latitude , float
store_and_fwd_flag , int
trip_duration , int
```

Fig. 1. Changing column type from String.

While preparing the data for the machine learning models, there were several operations performed on the dataset. We have changed the type of each column from string to either integer, double, string or timestamp according to its values as shown in the above figure (Fig. 1).

```
In [60]:
# Calculate distance between two places.

def distance(pickUpLat1, dropOffLat2, pickUpLon1, dropOffLon2):
    longitude = radians(dropOffLon2) - radians(pickUpLon1)
    x = sin(longitude / 2)**2 + cos(radians(pickUpLat1)) * cos(radians(dropOffLat2)) * sin(longitude / 2)**2

    temp = 2 * asin(sqrt(x))
    earth_radius = 6371
    return(temp * earth_radius)

In [61]:
# Calculating distance between pickup and dropoff latitude and longtitude. In (K.M.)

distance_func = udf(distance, FloatType())
df = df.withColumn("distance", distance_func(df.pickup_latitude, df.dropoff_latitude, df.pickup_longitude, df.dropc
```

Fig. 2. Calculating distance for each trip based on its geographics.

We have calculated the distance between pickup latitude, longitude and dropoff latitude and longitude for all the trips as shown in the above figure (Fig. 2).

```
In [63]:
# Remove all trips that have trip_duration more than 7200 seconds which is equivalent to more than 2 hours and less
# Remove all trips which total distance is less than 1.5 and more than 200 KM because NYC is not big than 60 KM.

df = df.filter((df.trip_duration > 180) & (df.trip_duration < 7200) & (df.distance > 1.5) & (df.distance < 60))

In [64]:
df.show()
```

```
+---------+----------------+-----------------+---------------+----------------+---------------+-----------------
+----------------+-----------+-------------+---------+
|vendor_id|    pickup_datetime|   dropoff_datetime|passenger_count|pickup_longitude|pickup_latitude|dropoff_longitude
|dropoff_latitude|store_and_fwd_flag|trip_duration| distance|
+---------+----------------+-----------------+---------------+----------------+---------------+-----------------
+----------------+-----------+-------------+---------+
|        2|2016-03-14 17:24:55|2016-03-14 17:32:30|              1|      -73.982155|      40.767937|       -73.96463
|       40.765602|          0|          455|2.4444735|
|        1|2016-06-12 00:43:35|2016-06-12 00:54:38|              1|      -73.980415|      40.738564|       -73.99948
|       40.73115|          0|          663|2.6599078|
|        2|2016-01-19 11:35:24|2016-01-19 12:10:48|              1|      -73.97903|       40.76394|       -74.00533
|       40.710087|          0|         2124| 3.669969|
|        1|2016-06-17 22:34:59|2016-06-17 22:40:40|              4|      -73.96902|       40.75784|       -73.957405
|       40.765896|          0|          341|1.6197599|
|        2|2016-05-21 07:54:58|2016-05-21 08:20:49|              1|      -73.96928|       40.79778|       -73.92247
|       40.76056|          0|         1551| 6.528439|
```

Fig. 3. Filter the trips based on distance and Example DF after filtering.

We keep only trips whose trip duration is between 3 minutes to 2 hours as taxi trips in New York City might not take more than 2 hours at any time. In addition to that we believe that taxi rides in New York city may always be less than 200 kilometers as no taxi accepts rides as the destination is more than 200 kilometers far away. (Fig. 3)

Add timeline for each ride

```
In [65]:
df = df.withColumn("week_day", date_format(df["pickup_datetime"], "E"))
df = df.withColumn('year', date_format(df["pickup_datetime"], "y"))
df = df.withColumn('month', date_format(df["pickup_datetime"], "M"))
df = df.withColumn('quarter_of_year', quarter('pickup_datetime'))
df = df.withColumn('hour', date_format(df["pickup_datetime"], "H"))

df = df.drop("pickup_datetime")
df = df.drop("dropoff_datetime")

In [66]:
df.show()
```

```
+---------+---------------+---------------+---------------+-----------------+---------------+-----------------+---
----------+---------+-------+----+-----+---------------+----+
|vendor_id|passenger_count|pickup_longitude|pickup_latitude|dropoff_longitude|dropoff_latitude|store_and_fwd_flag|tri
p_duration| distance|week_day|year|month|quarter_of_year|hour|
+---------+---------------+---------------+---------------+-----------------+---------------+-----------------+---
----------+---------+-------+----+-----+---------------+----+
|        2|              1|      -73.982155|      40.767937|       -73.96463|      40.765602|                N|
455|2.4444735|    Mon|2016|    3|              1|  17|
|        1|              1|      -73.980415|      40.738564|       -73.99948|       40.73115|                N|
663|2.6599078|    Sun|2016|    6|              2|   0|
|        2|              1|      -73.97903|       40.76394|       -74.00533|       40.710087|                N|
2124| 3.669969|    Tue|2016|    1|              1|  11|
|        1|              4|      -73.96902|       40.75784|      -73.957405|       40.765896|                N|
341|1.6197599|    Fri|2016|    6|              2|  22|
|        2|              1|      -73.96928|       40.79778|       -73.92247|       40.76056|                N|
1551| 6.528439|    Sat|2016|    5|              2|   7|
|        1|              1|      -73.99948|       40.7384|       -73.98579|      40.732815|                N|
255|1.9105701|    Fri|2016|    5|              2|  23|
|        2|              1|      -73.98265|      40.76384|       -74.00223|       40.73299|                N|
1274|2.7310188|    Tue|2016|    5|              2|  22|
```

Fig. 4. Adding timeline for each ride and Example DF.

Last but not the least, we added quarter, year and day of the week to the dataset to get more accurate predictions from the machine learning model. (Fig. 4)

C. Data Visualization

From the figure below (Fig. 5) we can see that the total number of trips is higher on Friday and Saturday with the Average Passenger count higher on Sunday as well because people are more likely to go out with the family at the start of the weekend.

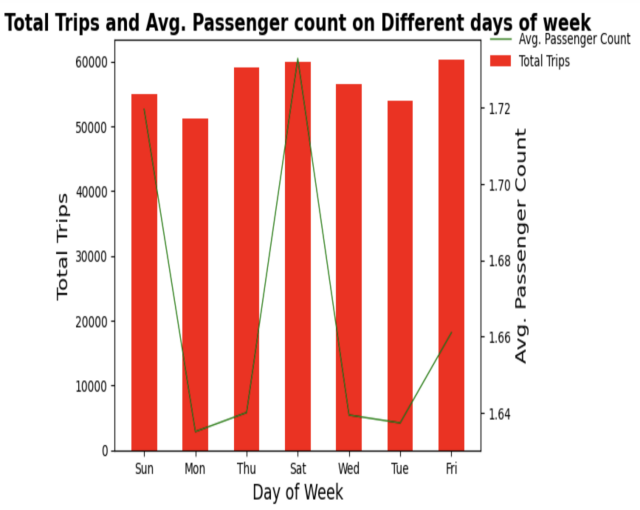

Fig. 5. Total trips and average passengers count on different week days.

This figure (Fig. 5) also shows us that as the weekend arrives people are more likely to take taxi trips to their destination with their families and friends.

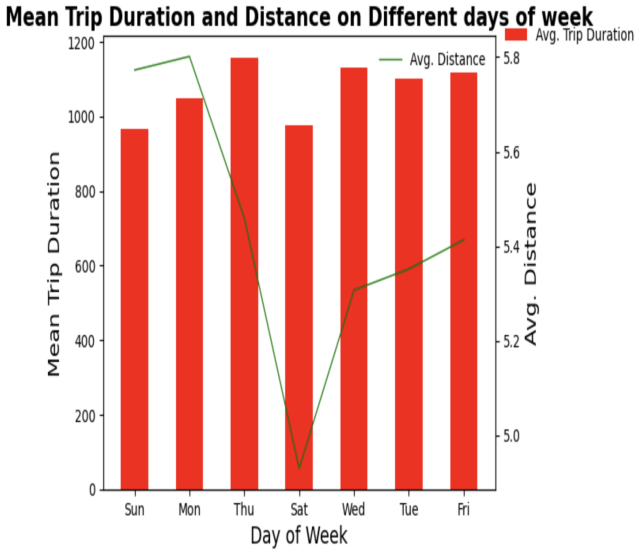

Fig. 6. Average trip duration and distance on different week days.

We can conclude from the above figure (Fig. 6) that trip duration is higher on the weekend compared to weekdays.

This figure (Fig. 6) helps us to match our initial expectation that trip duration is lower on weekends as there would be less traffic because most of people are not working.

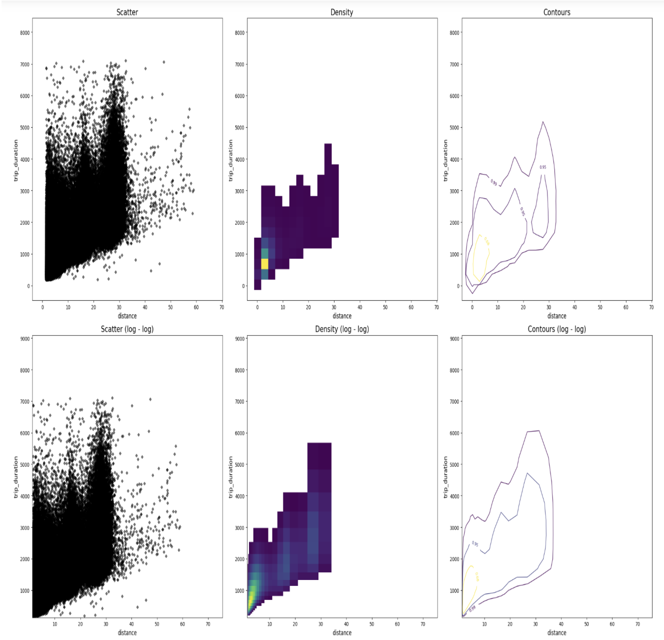

Fig. 7. Scatter, Density and Contours(log) between distance and trip duration.

While predicting and getting detailed insights of the data regarding any duration of the trip the most important variable to consider is a distance and also most people rely on the distance of two places to predict their duration of the trip.

From the figure above (Fig. 7), we can say that there is a linear positive correlation between the distance and the duration of the trip. Also, from the density and Contours plots, we can say that most of the data has distance and trip duration at small scale.

Around 70% of the trip data has a distance less than 10 kilometers and its trip duration is less than 2000 seconds which is nearly an hour. Hence, we can conclude that data contains some outliers as most of the data is compacted and there are some trips whose trip duration and distance is very huge.

D. Data Preparation for Regression

As mentioned above, we have done some data processing to remove potential outliers. As both regression models require all features to be compressed into a vector, so we have converted a week_day column into an index column (Fig. 8).

```
In [11]:
# Convert week day string column into index column.
label_index = StringIndexer(inputCol = 'week_day', outputCol = 'week_day_index')
df = label_index.fit(df).transform(df)
df.show()

+---------+---------------+----------------+---------------+----------------+---------------+------------------+-------------+---------+
--------+----+-----+---------------+----+--------------+
|vendor_id|passenger_count|pickup_longitude|pickup_latitude|dropoff_longitude|dropoff_latitude|store_and_fwd_flag|trip_duration| distance|
week_day|year|month|quarter_of_year|hour|week_day_index|
+---------+---------------+----------------+---------------+----------------+---------------+------------------+-------------+---------+
--------+----+-----+---------------+----+--------------+
|        2|              1|      -73.982155|      40.767937|       -73.96463|      40.765602|                 0|          455|2.4444735|
Mon|2016|    3|              1|  17|           6.0|
|        1|              1|      -73.980415|      40.738564|       -73.99948|       40.73115|                 0|          663|2.6599078|
Sun|2016|    6|              2|   0|           4.0|
|        2|              1|       -73.97903|       40.76394|       -74.00533|      40.710087|                 0|         2124| 3.669969|
Tue|2016|    1|              1|  11|           5.0|
|        1|              4|       -73.96902|       40.75784|      -73.957405|      40.765896|                 0|          341|1.6197599|
Fri|2016|    6|              2|  22|           0.0|
|        2|              1|       -73.96928|       40.79778|       -73.92247|       40.76056|                 0|         1551| 6.528439|
Sat|2016|    5|              2|   7|           1.0|
```

Fig. 8. Converting week day string values column into integers.

Also, we have vectorised all the features and converted them into a PySpark dataframe as shown in the figure below (Fig. 9.)

```
In [12]:
# Creating a list of input columns headers
input_col = [i[0] for i in df.dtypes if i[1] == "int" or i[1] == "double"]
input_col.remove("trip_duration")

# Creating a dataframe of input columns.
input_features = VectorAssembler(inputCols = input_col, outputCol = "input_features")
input_column = input_features.transform(df)
input_column.select("input_features").show()

23/01/15 23:58:49 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by set
ting 'spark.sql.debug.maxToStringFields'.
+--------------------+
|      input_features|
+--------------------+
|[2.0,1.0,-73.9821...|
|[1.0,1.0,-73.9804...|
|[2.0,1.0,-73.9790...|
|[1.0,4.0,-73.9690...|
|[2.0,1.0,-73.9692...|
|[1.0,1.0,-73.9994...|
|[2.0,1.0,-73.9826...|
|[2.0,4.0,-73.9915...|
|[2.0,2.0,-73.9629...|
|[2.0,1.0,-73.9921...|
|[1.0,1.0,-74.0039...|
|[1.0,1.0,-73.9803...|
|[2.0,1.0,-73.9795...|
|[1.0,1.0,-73.9935...|
|[2.0,1.0,-73.9552...|
|[2.0,1.0,-73.9565...|
|[1.0,1.0,-73.9837...|
|[2.0,1.0,-73.9942...|
|[1.0,1.0,-73.9821...|
|[1.0,1.0,-73.9709...|
+--------------------+
only showing top 20 rows

In [13]:
data_frame = input_column.select("input_features", "trip_duration")
data_frame.show(5)

+--------------------+-------------+
|      input_features|trip_duration|
+--------------------+-------------+
|[2.0,1.0,-73.9821...|          455|
|[1.0,1.0,-73.9804...|          663|
```

Fig. 9. DataFrame for the regression machine learning model.

In both regression models, we have used the trip_duration column as an output column and other columns as an input column. And after all processing, 70% of the DataFrame split into training and 30% DataFrame was for testing. So, both regression models will have enough data to identify trends in the trip_duration of NYC taxi trips.

E. Data Preparation for Clustering

KMeans clustering requires all features to be an indexer, we have converted a week_day column into an index column (Fig. 10).

```
In [83]:
# Convert a week_day string column to an index.
df = df.drop(df["store_and_fwd_flag"])
week_day_col = StringIndexer(inputCol = 'week_day', outputCol = 'week_day_index')
df = week_day_col.fit(df).transform(df)
df.show(5)

+---------+---------------+----------------+---------------+----------------+---------------+-------------+---------+--------+----+-----
+---------------+----+--------------+
|vendor_id|passenger_count|pickup_longitude|pickup_latitude|dropoff_longitude|dropoff_latitude|trip_duration| distance|week_day|year|month
|quarter_of_year|hour|week_day_index|
+---------+---------------+----------------+---------------+----------------+---------------+-------------+---------+--------+----+-----
+---------------+----+--------------+
|        2|              1|      -73.982155|      40.767937|       -73.96463|      40.765602|          455|2.4444735|     Mon|2016|    3
|              1|  17|           6.0|
|        1|              1|      -73.980415|      40.738564|       -73.99948|       40.73115|          663|2.6599078|     Sun|2016|    6
|              2|   0|           4.0|
|        2|              1|       -73.97903|       40.76394|       -74.00533|      40.710087|         2124| 3.669969|     Tue|2016|    1
|              1|  11|           5.0|
|        1|              4|       -73.96902|       40.75784|      -73.957405|      40.765896|          341|1.6197599|     Fri|2016|    6
|              2|  22|           0.0|
```

Fig. 10. Converting week day string values column into integers.

All other input data requirements are mostly similar to the regression models, most of the data processing has been done in the similar way for Clustering algorithms.

IV. ALGORITHMS

A. Linear Regression

To identify the best parameters and overcome overfitting problem for the linear regression machine learning model, we used CrossValidator for regParam, elasticNetParam, maxIter and fitIntercept input parameters for the model.

```
In [21]:
# Print best parameter of the model.
print("Best Regression Param: ", model.bestModel._java_obj.getRegParam())
print("Best Elastic Net Param: ", model.bestModel._java_obj.getElasticNetParam())
print("Best maxIter Param: ", model.bestModel._java_obj.getMaxIter())
print("Best fitIntercept Param: ", model.bestModel._java_obj.getFitIntercept())
print("Intercept: ", model.bestModel.intercept)

# Acuracy and coefficients of the linear equation.
print("Coefficient of Determination(Accuracy on Training data): ", str(round(model.bestModel.summary.r2*100, 2)), "%")
print("Coefficients: ", model.bestModel.coefficients)
```

```
Best Regression Param:  0.01
Best Elastic Net Param:  0.25
Best maxIter Param:  20
Best fitIntercept Param:  True
Intercept:  -19235.005975346867
Coefficient of Determination(Accuracy on Training data):  50.12 %
Coefficients:  [0.7493259107076 41,2.2572463385882204,-325.336075706018,-217.67552819562678,-536.4434332987057,-862.7021553118498,75.327
96525925,82.03011447964624,0.0,18.879690528403284,22.082995444444144,5.383600297732522,-12.106268796570658]
```

Fig. 11. Best Parameters for linear regression model.

Above figure (Fig. 11) shows the best input parameters for the linear regression model. With these parameters, the accuracy of the model on training DataFrame is roughly 50%.

```
In [22]:
# Predict.
prediction = model.transform(test)

prediction.show(5)
```

```
[Stage 7523:>                                                       (0 + 1) / 1]
+--------------------+-------------+------------------+
|      input_features|trip_duration|        prediction|
+--------------------+-------------+------------------+
|[1.0,1.0,-74.2310...|         1800| 4318.493897415774|
|[1.0,1.0,-74.0215...|          340| 885.8448166915368|
|[1.0,1.0,-74.0184...|          667|1244.3454351175642|
|[1.0,1.0,-74.0180...|         3553|1487.2242294263888|
|[1.0,1.0,-74.0179...|         1054|1244.1507614153415|
+--------------------+-------------+------------------+
only showing top 5 rows
```

```
In [23]:
# Evaluator for testing data.
test_pred_evaluator = RegressionEvaluator(predictionCol = "prediction", labelCol = "trip_duration", metricName = "r2")

# Print RMSE and accuracy of the model on testing data.
print("RMSE: ", evaluator.evaluate(prediction))
print("Coefficient of Determination(Accuracy on Test Data): ", str(round(test_pred_evaluator.evaluate(prediction)*100, 4)), "%")
```

```
RMSE:  500.76489579072614
[Stage 7525:>                                                       (0 + 1) / 1]
Coefficient of Determination(Accuracy on Test Data):  50.3203 %
```

Fig. 12. Prediction, RMSE and accuracy on test data.

Figure 12. shows the root mean square error (RMSE) on training DataFrame which is nearly 500 and accuracy of the model on testing data is slightly less than 50.5%.

B. Gradient Boosted Trees Regression

Similar to the linear regression model, to identify the best parameters and overcome overfitting, we used CrossValidator on maxDepth, maxIter and maxBins input parameters of the gradient boosted trees model.

```
In [27]:
# Print best depth, bins, maxIter, impurity and losstype.
print("Best maxDepth Param: ", model_gbtr.bestModel._java_obj.getMaxDepth())
print("Best maxBins Param: ", model_gbtr.bestModel._java_obj.getMaxBins())
print("Best maxIter Param: ", model_gbtr.bestModel._java_obj.getMaxIter())
print("Best impurity Param: ", model_gbtr.bestModel._java_obj.getImpurity())
print("Best lossType Param: ", model_gbtr.bestModel._java_obj.getLossType())
```

```
Best maxDepth Param:  10
Best maxBins Param:  64
Best maxIter Param:  20
Best impurity Param:  variance
Best lossType Param:  squared
```

Fig. 13. Best Parameters for Gradient boosted trees model.

Figure 13 shows the best input parameters for the gradient boosted trees regression model and with these input parameters we achieved the accuracy of 75% on testing data with root mean square error (RMSE) nearly 360.

```
In [29]:
# Evaluator for testing data for gradient boosted trees regression model.
test_pred_evaluator_gbtr = RegressionEvaluator(predictionCol = "prediction", labelCol = "trip_duration", metricName = "r2")

# Print RMSE and accuracy of the model on testing data for gradient boosted trees regression.
print("RMSE: ", gbt_evaluator.evaluate(prediction_gbtr))
print("Coefficient of Determination(Accuracy on Test Data): ", str(round(test_pred_evaluator_gbtr.evaluate(prediction_gbtr)*100, 4)), "%"
```

```
RMSE:  362.0912865178779
[Stage 18732:>                                                      (0 + 1) / 1]
Coefficient of Determination(Accuracy on Test Data):  74.0255 %
```

Fig. 14. RMSE and coefficient of determination of the gradient boosted tress regression model.

Hence, we can conclude from the evidence that Gradient boosted trees regression model is better machine learning model compared to linear regression because of lower RMSE and higher Coefficient of Determination.

C. KMeans Clustering

1. Finding optimum number of clusters

The K-means clustering is an unsupervised machine learning technique that groups similar data points together based on underlying similarities and patterns. Aim of the KMeans clustering was to find out if data points within the same clusters have similar trip durations or not. If any correlation was found, it becomes easier to find out which properties of the dataset are strongly influencing the trip duration.

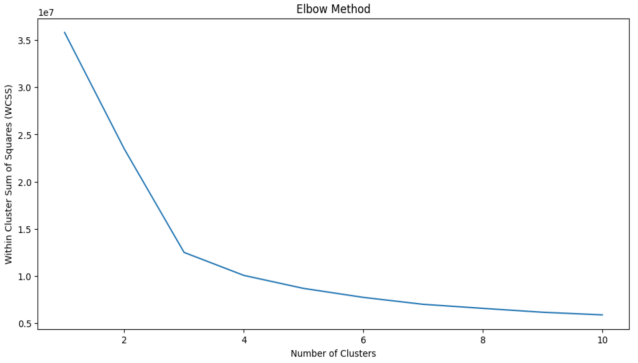

Fig. 15. Elbow Method (Determination of optimum clusters).

While working with data that has multiple dimensions, it is not possible to visualize the data and decide on the optimum number of clusters to use for the K-means clustering model. However, there is a mathematical way to determine the optimum number of clusters, called the "elbow method". To use this method, we have to create clustering models using a different range of clusters, and for each cluster, we need to compute the Within Cluster Sum of Squares (WCSS). WCSS is the within-cluster-sum-of-squares which means the sum of squared distance between each point and the centroid of the cluster. With this method in (Fig. 15), we found the optimum number of clusters is k = 4.

```
In [87]:
#Using StandardScaler to set all the columns in the dataframe to the same scale.
std_scaler = StandardScaler()
X = std_scaler.fit_transform(input_col_pandas_df)

X = pd.DataFrame(X, columns = input_col_pandas_df.columns)
X
```

Out[87]:

	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	distance	year	month	quarter_of_year	hour	v
0	0.932331	-0.508690	-0.295293	0.608563	0.085584	0.394951	-0.482016	0.0	-0.315667	-1.013856	0.527376	
1	-1.072580	-0.508690	-0.257719	-0.345022	-0.723479	-0.542761	-0.447158	0.0	1.469916	0.986333	-2.057367	
2	0.932331	-0.508690	-0.227810	0.478802	-0.859290	-1.116052	-0.283727	0.0	-1.506056	-1.013856	-0.384886	
3	-1.072580	1.778467	-0.011651	0.280767	0.253316	0.402953	-0.615457	0.0	1.469916	0.986333	1.287595	
4	0.932331	-0.508690	-0.017265	1.577407	1.064351	0.257718	0.178782	0.0	0.874721	0.986333	-0.993061	

Fig. 16. Set all column data at the same scale for KMeans.

To fit the data into KMeans cluster and set all the columns data at the same scale, we used StandardScaler data preprocessing function and used output to fit in our KMeans model.

```
In [89]:
# Graph out the size of each cluster.
input_col_pandas_df['cluster'] = clusters_prediction
cluster_counts = input_col_pandas_df.groupby(['cluster'])['cluster'].count()

plt.figure(figsize=(12, 6), dpi=100)

plt.bar(['0', '1', '2', '3'], cluster_counts.values, width = 0.3)

plt.xlabel('Cluster')
plt.ylabel('Total Data points Count in a Cluster')

plt.show()
```

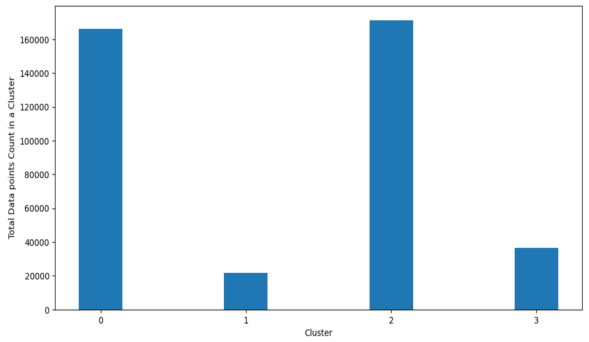

Fig. 17. Total data points in each cluster.

As we have seen before that almost 70% of the data points are nearly clustered together, that leads to the conclusion that most of the data points might reside in the same cluster. As we can see in the above figure (Fig, 17) that most of the data points are in cluster 0 and 2 as we expected.

2. Cluster diagnoses

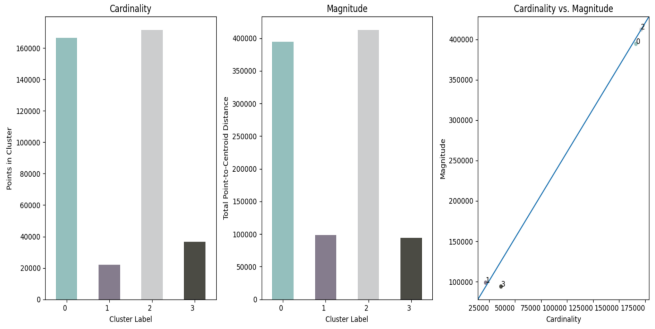

Fig. 18. Cluster cardinality, magnitude and Cardinality vs. Magnitude.

Now, it is important to get good cluster diagnoses. There are three main diagnoses that we have prepared which includes cluster cardinality, cluster magnitude and cluster cardinality vs. cluster magnitude. Cluster cardinality means the total number of data points within each cluster. While cluster magnitude shows the total points to centroid distance per cluster. That is used to check how widely data points are spread in each cluster. As we can see in the above figure (Fig. 18) clusters with higher cardinality tend to have higher magnitude.

3. Cluster characteristics

For a business person and a data scientist, it is important to identify the key characteristics for each cluster in order to know what each cluster stands for. Hence, we have generated a box plot (Fig. 19) that shows each feature distribution per cluster.

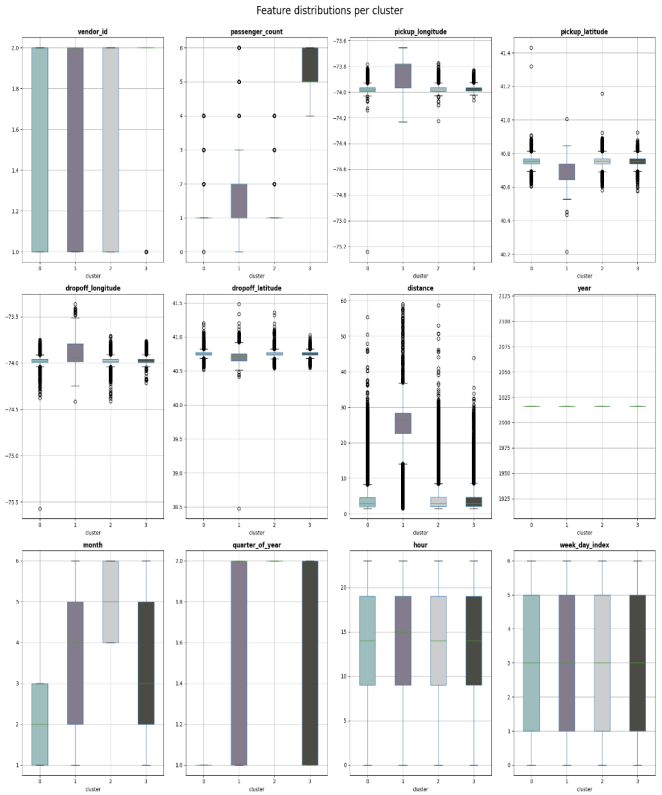

Fig. 19. Box plot for the feature distribution for each cluster.

From this box plot (Fig. 19), we can conclude that passenger_count, pickup_longtitude, dropoff_longtitude, distance, and month are the main features for a data point to reside in a particular cluster.

```
In [94]:
# Calculate mean for each feature in each cluster.
X_mean = pd.concat([pd.DataFrame(X.mean().drop('cluster'), columns=['mean']), X.groupby('cluster').mean().T], axis = 1)
X_mean.drop(columns=['mean'], inplace=True)

X_mean
```

Out[94]:

	0	1	2	3
vendor_id	-0.098066	0.055865	-0.109343	0.925092
passenger_count	-0.300208	0.025165	-0.290359	2.711194
pickup_longitude	-0.158642	2.642086	-0.154917	-0.130937
pickup_latitude	0.131116	-2.222794	0.127117	0.135909
dropoff_longitude	-0.101386	1.659185	-0.095520	-0.082480
dropoff_latitude	0.063276	-1.059515	0.060757	0.060466
distance	-0.187411	3.043984	-0.171285	-0.163478
year	0.000000	0.000000	0.000000	0.000000
month	-0.884719	0.023120	0.861630	-0.027033
quarter_of_year	-1.013856	0.021634	0.986333	-0.023090
hour	0.004716	0.037538	-0.006411	-0.013838
week_day_index	-0.007353	0.083785	-0.000665	-0.013494

Fig. 20. Mean for each feature per cluster.

Last but not the least radar plot, which is another best way to summarize all relevant information in one plot. In order to avoid crashes from the visualization and maintain the scale of the plot, we have used the StandardScale DataFrame and calculated the mean for each feature per cluster (Fig. 20) and generated the radar plot as shown in the figure below (Fig. 21).

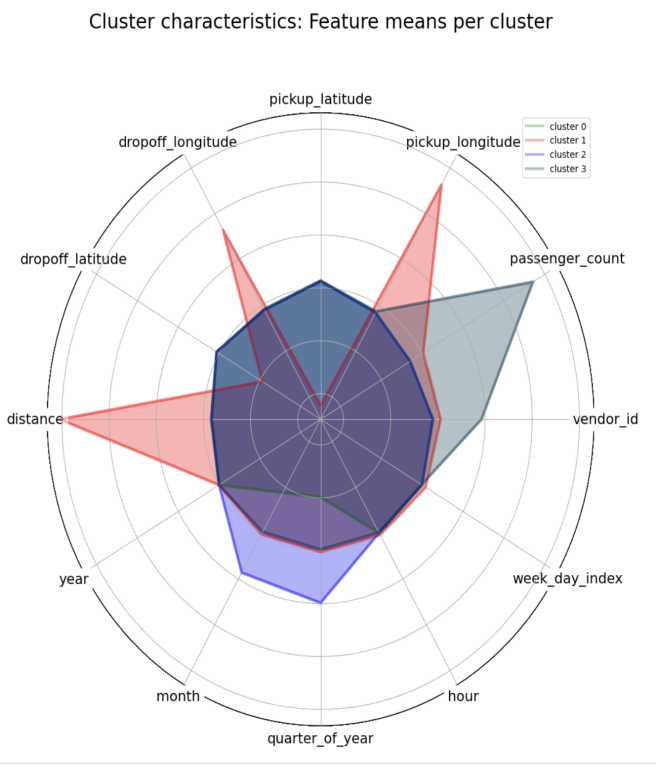

Fig. 21. Radar plot.

From the above plot (Fig. 21), the decision for a datapoint to be in cluster 1 and 3 highly depends on the distance, dropoff_longtitude, pickup_logtitude and passenger_count respectively. While data points in cluster 2 highly depended on the month and quarter of the year.

V. RESULT, CONCLUSION AND FUTURE WORK

This entire project involves 6 major steps: Exploring the dataset, Data Cleaning, Data Preprocessing, Data Visualization, Clustering and Regression Machine Learning models.

By visualizing a distance and trip duration on both linear and log-log plots, we saw that these two properties display a certain amount of positive linear correlation. Alongside, roughly 70% of the NYC trip has a distance less than 10 kilometers and their trip duration is less than 2000 seconds which is less than half an hour. That led us to the conclusion that data has a lot of outliers in terms of distance and trip duration. In addition to this, we reached the conclusion that the number of trips and average passenger counts increase as weekends arrive. On the other hand, trip duration seems to be higher during weekdays (Fig. 6).

As mentioned before, we used CrossValidator to find the best possible parameters and generalize a pattern in the data for our machine learning models. The root mean square error (RMSE) for the linear regression model was 500.76 and coefficient of determination was 50.32%. Hence, the linear regression model is not the best machine learning model for predicting the NYC taxi trip duration.

On the other hand, with the gradient boosted trees regression, the root mean square error is 362 and coefficient of determination (accuracy) is roughly 75% because of the better hyper parameter tuning and optimization on different loss functions. Hence, we can conclude that gradient boosted trees regression provides us better prediction compared to linear regression model.

For the clustering, it was difficult to see the differences in clusters. Although, we have performed several iterations while changing the parameters and by plotting several graphs to identify the patterns. Still we have seen some of the features like distance, passenger_counts, pickup_longtitude, and dropoff_longtitude really help us to clusterized the data as shown in the above figures (Fig. 18, 19, 21).

To use this huge data set and build machine learning models, it requires to have really good hardwares with good unified memory, better SSD storage and most importantly a great GPU. We have limited hardware resources, hence we were not able to experiment our machine learning model with several different hyper tuning parameters. So, for the future iteration, it is worthwhile to experiment with different hyper tuning parameters for the regression model.

While we were only able to use the Elbow method to determine the best number of clusters for our data because of the limited resources. For the future, we would like to use silhouette analysis, Agglomerative hierarchical Clustering and Gaussian Mixtures model to identify the best number of clusters and do better cluster diagnoses.

VI. REFERENCES

https://www.macrotrends.net/cities/23083/new-yorkcity/population#:~:text=The%20current%20metro%20area%

https://www.kaggle.com/competitions/nyc-taxi-trip-duration

https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

https://towardsdatascience.com/best-practices-for-visualizing-your-cluster-results-20a3baac7426

https://towardsdatascience.com/common-mistakes-in-cluster-analysis-and-how-to-avoid-them-eb960116d773