# New York Taxi data set analysis

A PROJECT REPORT

***Submitted by***

**TEAM -2**


| | |
|---|---|
| P. NANDIESWAR REDDY | BL.EN.U4AIE20046 |
| RITHVIKA ALAPATI | BL.EN.U4AIE20054 |
| SAI ASWATH S | BL.EN.U4AIE20056 |


**19AIE214 – Big Data Analytics**

**B.Tech. in Computer Science and Engineering (Artificial Intelligence)**



AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDYAPEETHAM

BANGALORE 560 035


June – 2022

# TABLE OF CONTENTS

# 1. Abstract

Predictive analysis, a machine learning analytical technique, will be studied in this project. Many companies, including Ola, Uber, and others, use machine learning and artificial intelligence to solve the problem of accurate fare prediction. To obtain the most accurate value, we suggest comparing techniques for prediction modelling, such as regression and classification. Those who are interested in fare forecasting will find this analysis useful. Previously, the fare was only determined by the distance travelled, but with the advancement of technology, the fare is now influenced by the time of day, the location, the number of passengers, the amount of traffic, the number of hours, the base fare, and other factors. The analysis is focused on supervised learning, whose one application is prediction, in machine learning.

# 2. Introduction

Taxi rides in New York City paint a vivid picture of city life. The millions of rides taken each month can provide information about traffic patterns, road closures, and large-scale events that draw many New Yorkers. With the rise of ride-sharing apps, it is becoming increasingly important for taxi companies to provide visibility into their estimated fare, as competing apps provide these metrics upfront. Predicting a ride's fare, for example, can assist passengers in determining the best time to begin their commute or drivers in determining which of two potential rides will be more profitable. Furthermore, this transparency into fare will attract customers when ridesharing services implement surge pricing.

Only data that would be available at the start of a ride was used to predict fare. This includes pickup and drop-off coordinates, trip distance, start time, passenger count, and a rate code indicating the standard rate was used.

we have used supervised learning approach, because it suits the best as per the requirement of predictive analysis. Prediction is performed as data is collected from past, the model is trained to handle new data and predict the desired output.

we made the decision to use Apache Spark Mlib to attempt and create some predictive models by applying machine learning techniques to the data set. Decision tree and GBT (Gradient Boosting-based Tree) models were used to predict fare amount.

# 3. Problem statement

In this project, we look at the dataset from the NYC Taxi data and predict the fare amount of taxi trips in New York City. we will use several machine learning methods for the prediction task, and the models will be evaluated using the root mean squared error.

# 4. Clustering

A Spark cluster is a combination of a Driver Program, Cluster Manager, and Worker Nodes that work together to complete tasks.

Fig 1: Clustering setup

We used 3 systems for clustering:

Master – 192.168.1.107 ➔ 16gb (8 core)

1. Slave1 – 192.168.1.104 ➔ 16gb (8 core)
2. Slave2 – 192.168.1.105 ➔ 16gb (12 core)
3. Master – 192.168.1.107 ➔ 16gb (8 core)

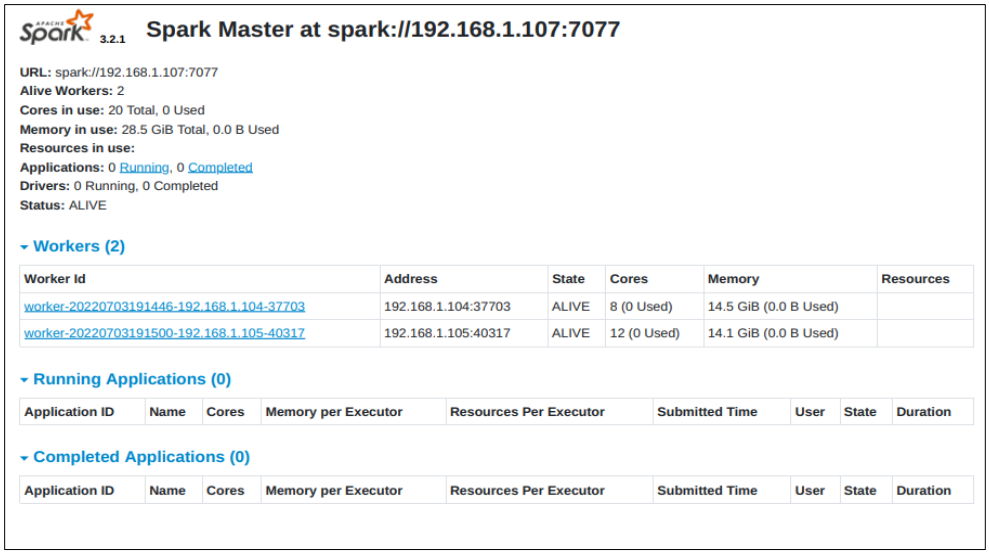
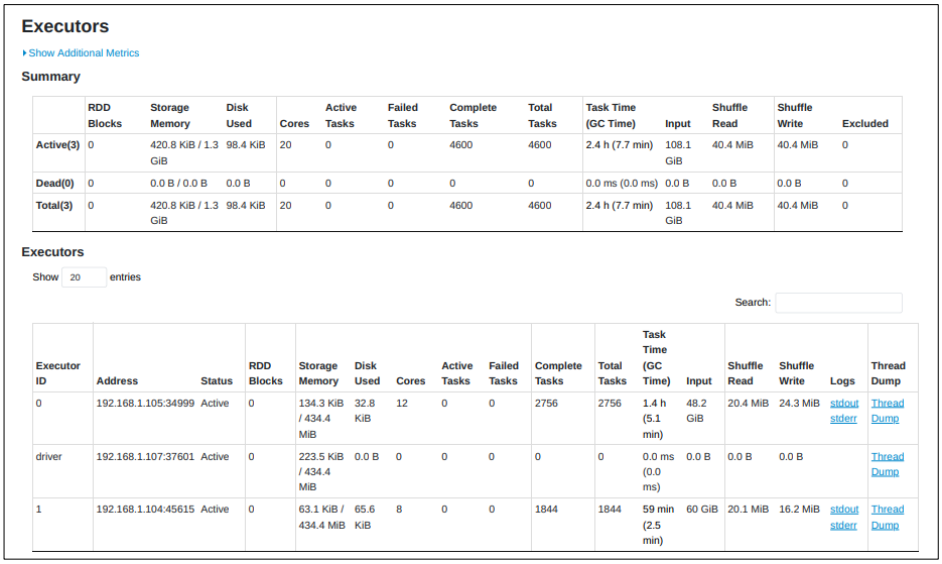



Fig 2: Spark Master

To access the spark master, we have assigned IP address of the master to slaves and type in the IP address of the master with the port number.192.168.1.107:8080 is the URL to get access of master. We can observe from above fig 2 that there are 2 workers connected to the master. To access the jobs we use Ip address of the master with port number as 4040 i.e.,192.168.1.107:4040.



## Executors

Show Additional Metrics

### Summary

| | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Excluded |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Active(3) | 0 | 420.8 KiB / 1.3 GiB | 98.4 KiB | 20 | 0 | 0 | 4600 | 4600 | 2.4 h (7.7 min) | 108.1 GiB | 40.4 MiB | 40.4 MiB | 0 |
| Dead(0) | 0 | 0.0 B / 0.0 B | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0.0 ms (0.0 ms) | 0.0 B | 0.0 B | 0.0 B | 0 |
| Total(3) | 0 | 420.8 KiB / 1.3 GiB | 98.4 KiB | 20 | 0 | 0 | 4600 | 4600 | 2.4 h (7.7 min) | 108.1 GiB | 40.4 MiB | 40.4 MiB | 0 |

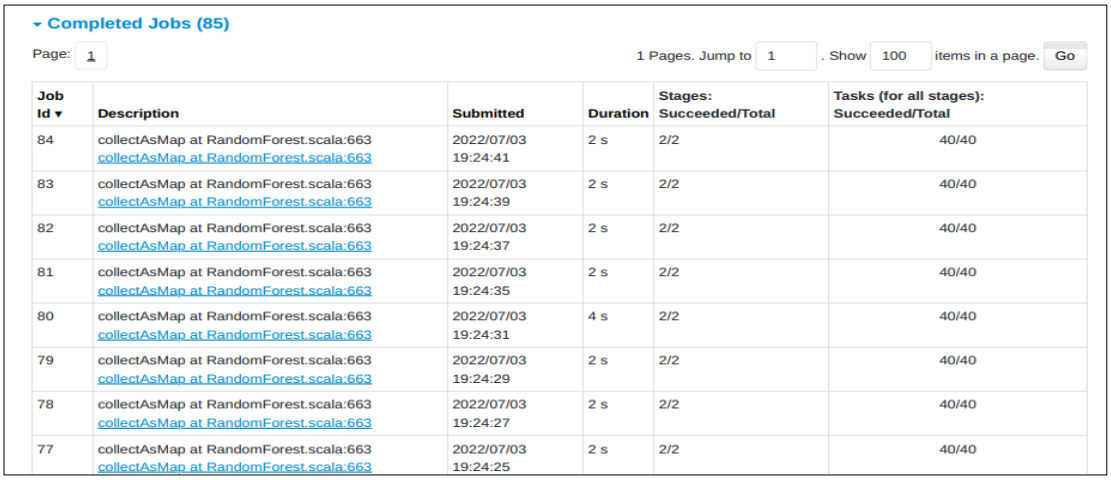
### Executors

Show 20 entries

Search:

| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Logs | Thread Dump |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 192.168.1.105:34999 | Active | 0 | 134.3 KiB / 434.4 MiB | 32.8 KiB | 12 | 0 | 0 | 2756 | 2756 | 1.4 h (5.1 min) | 48.2 GiB | 20.4 MiB | 24.3 MiB | stdout stderr | Thread Dump |
| driver | 192.168.1.107:37601 | Active | 0 | 223.5 KiB / 434.4 MiB | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0.0 ms (0.0 ms) | 0.0 B | 0.0 B | 0.0 B | | Thread Dump |
| 1 | 192.168.1.104:45615 | Active | 0 | 63.1 KiB / 434.4 MiB | 65.6 KiB | 8 | 0 | 0 | 1844 | 1844 | 59 min (2.5 min) | 60 GiB | 20.1 MiB | 16.2 MiB | stdout stderr | Thread Dump |

Fig 3: executors



## Completed Jobs (85)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 84 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:41 | 2 s | 2/2 | 40/40 |
| 83 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:39 | 2 s | 2/2 | 40/40 |
| 82 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:37 | 2 s | 2/2 | 40/40 |
| 81 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:35 | 2 s | 2/2 | 40/40 |
| 80 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:31 | 4 s | 2/2 | 40/40 |
| 79 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:29 | 2 s | 2/2 | 40/40 |
| 78 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:27 | 2 s | 2/2 | 40/40 |
| 77 | collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663 | 2022/07/03 19:24:25 | 2 s | 2/2 | 40/40 |

Fig 4: Jobs completed after executing the program

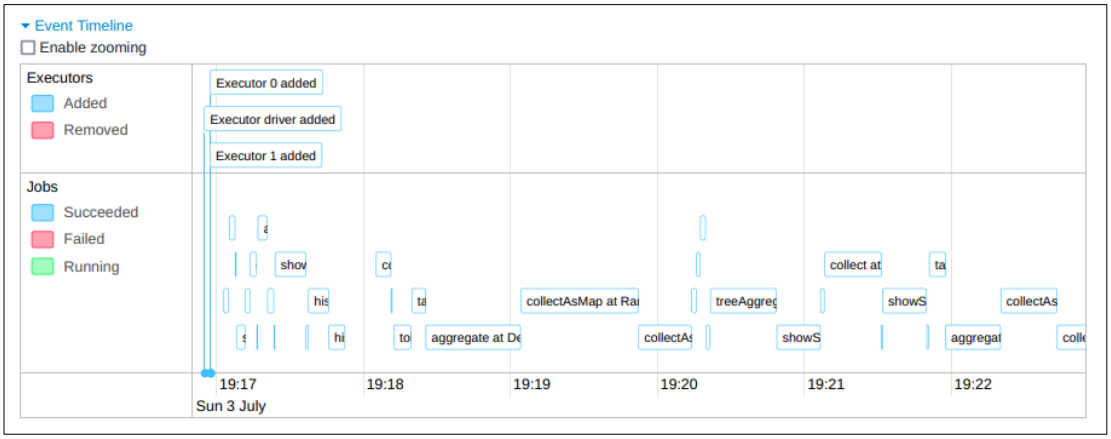



Fig 5: The timeline of each event happened

System Monitor of Master and slaves:

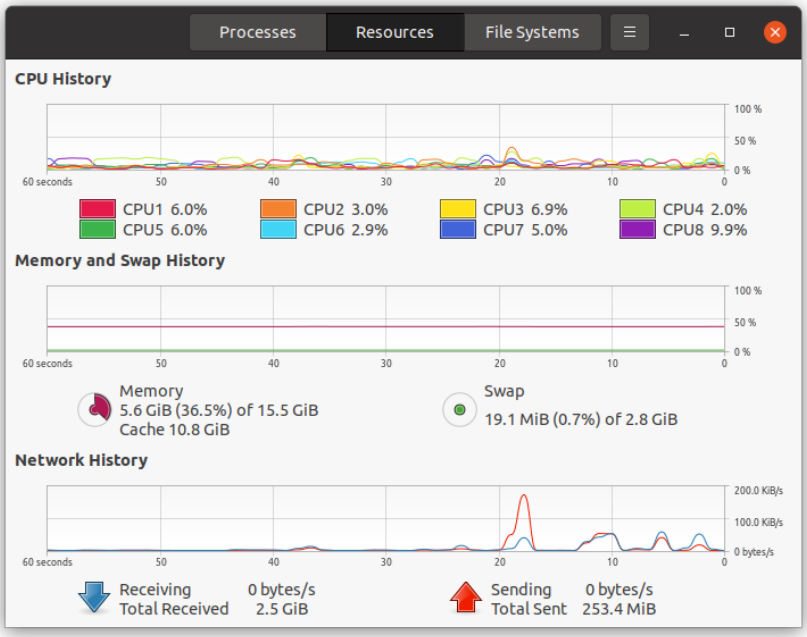



Fig 6: Master system monitor while executing the program

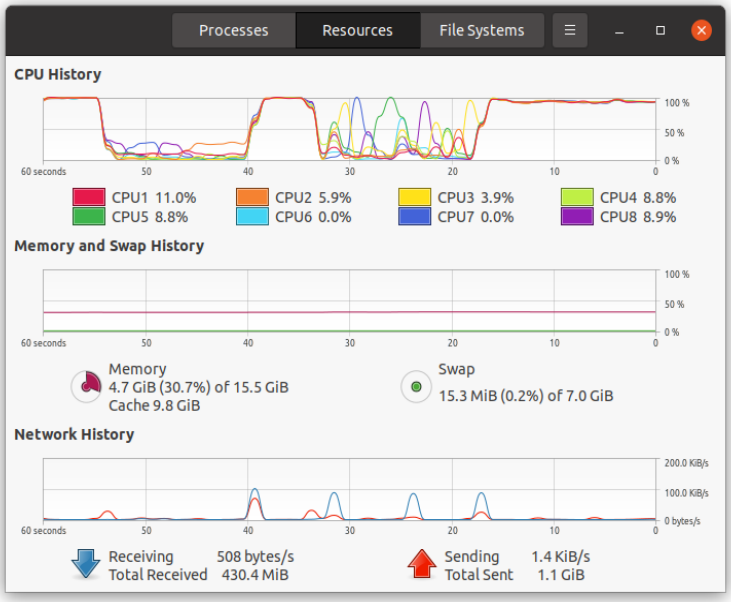



Fig 7: Slave 1 system monitor while executing the program

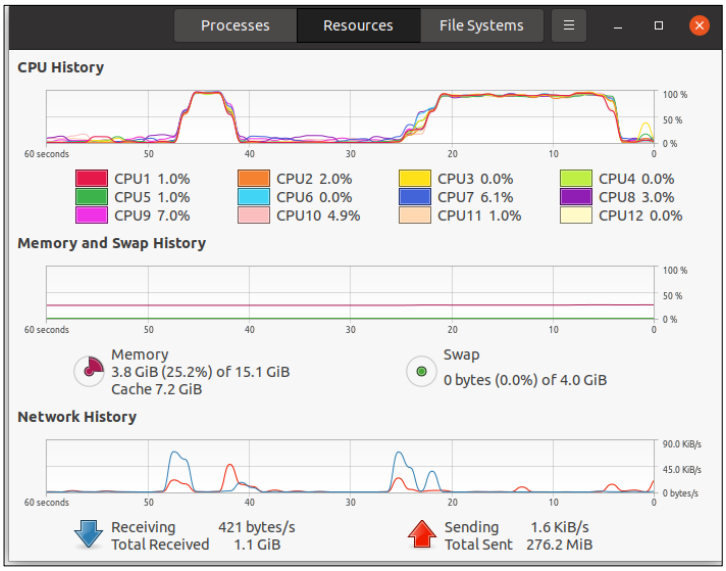



Fig 8: Slave 2 system monitor while executing the program

# 5. WORKFLOW

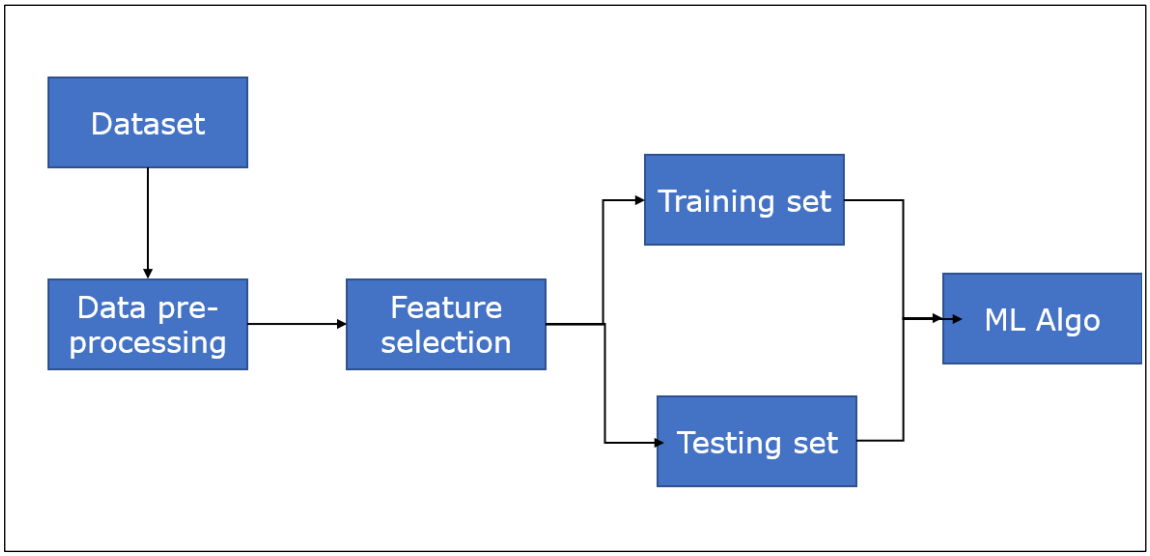



Fig 9: workflow of the project

## 5.1 Dataset

This project's data are all subsets of New York City Taxi data, which contains observations of taxi rides in New York City between 2015 and 2016. The total data is divided into two categories: yellow taxis, which mostly operate in Manhattan, and green taxis, which mostly operate in the city's outskirts. The data for taxi rides in January 2016 were used, but the models were validated using additional data. Because each month contains approximately 12 million observations and there were computational constraints, subsets of the monthly data were used for model building and other subsets for validation.

The original dataset contains features as pickup and drop off locations, as longitude and latitude coordinates, time and date of

pickup and drop off, ride fare, tip amount, payment type, trip distance and passenger count.

The dataset link: https://www.kaggle.com/datasets/sohaibanwaar1203/taxidemandfarepredictiondataset

The dataset consists of 6 pairs of trip/fare compressed CSV files of size 11GB. Each file contains about 14 million records, and the trip/fare files are matched line by line.

The dataset contain features of:

- id - a unique identifier for each trip
- vendor_id - a code indicating the provider associated with the trip record
- pickup_datetime - date and time when the meter was engaged
- dropoff_datetime - date and time when the meter was disengaged
- passenger_count - the number of passengers in the vehicle (driver entered value)
- 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
- store_and_fwd_flag - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- trip_duration - duration of the trip in seconds

The data was processed to extract year, month, day, weekday, hour, and minute features from each ride's date and time to model and account for traffic in the predictions, two additional features were calculated from the data: rides in an hour and average speed during the hour. The number of rides started within an hour of each observation is represented by rides in hour, and the average speed is the average speed of all those rides.

## 5.2 Data preprocessing

Data preprocessing is an important part of the data cleaning process in machine learning. It is essentially a process for converting raw data into clean data. The data will be cleaned in two ways: missing data and noisy data.

Missing Data:

Total missing values are dropped during the cleaning process; prior to cleaning, there were many missing values that were filled using the median method.

Other methods exist, such as mean and mode, but mode can produce biased results. As a result, mean and median are preferable.

Noisy Data:

Those values which are out of the range of longitude and latitude are eliminated. There are values which are equal to 0 we have also removed them.

## 5.3 Feature selection

In Machine learning, it is also known as attribute or variable selection. Basically, this is the process of selecting those attributes which

contribute most to the target variable. Here fare_amount is the target variable or prediction variable, while others are independent variables.

## 5.4 Models and methodology

After data pre-processing an important step comes and that is modelling also known as model selection. Model selection is the process of selecting a model fare_amount. This is a Regression problem. In regression problems, the dependent variable is continuous. In classification problems, the dependent variable is categorical So, we are going to deal with regression models on training data and predict it on test data. In this project, we are using Decision tree and GBT(Gradient Boosting-based Tree) is a tree-based algorithm which can be used to solve regression, classification problems and Linear regression model, which is also helpful in regression models. We splitted the whole data into 2 parts: train data (70%) and test data(30%). After that different model are approached.

**Decision tree regression:**

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets. The result is a tree with decision nodes and leaf nodes.

**GBT (Gradient Boosting-based Tree):**

Gradient boosting is a machine learning technique that is commonly used in regression and classification tasks. It returns a prediction model in the form of an ensemble of weak prediction models, usually decision trees. When a decision tree is used as the weak learner, the resulting algorithm is known as gradient-boosted trees, and it typically outperforms random forest. A gradient-boosted trees model

is constructed in the same stage-wise manner as other boosting methods, but it extends the other methods by allowing optimization of any differentiable loss function.

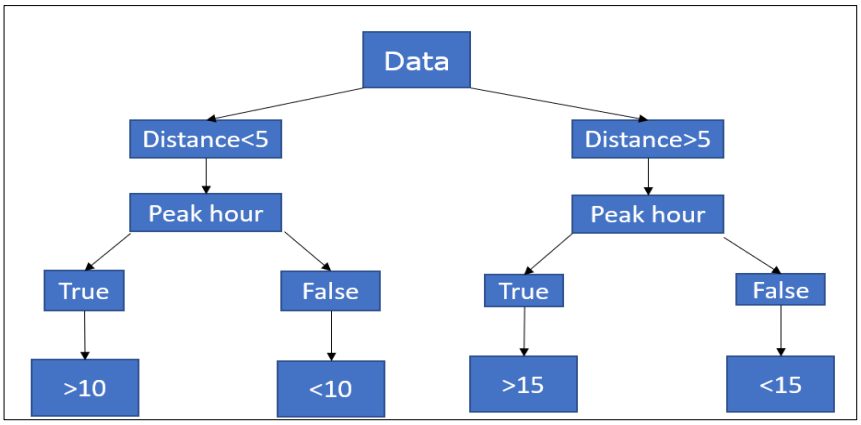



Fig 10: Working of the Model using decision tree and GBT regression

# 6. Implementation

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType, IntegerType
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.regression import GBTRegressor
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
spark=SparkSession.builder.appName("TaxiApp").config("spark.sql.executi
on.arrow.pyspark.enabled", "true").getOrCreate()
plt.style.use('ggplot')
spark
data = spark.read.csv('yellow_tripdata_2015*.csv')

data.createOrReplaceTempView('NY_taxi')
data.count()
data.printSchema()
```

```
data =
data.drop('_c4','_c7','_c8','_c11','_c12','c13','_c14','_c15','_c16','_
c17')


data.printSchema()
data = data.withColumn("_c18", data["_c18"].cast(DoubleType()))
data = data.withColumn("_c5", data["_c5"].cast(DoubleType()))
data = data.withColumn("_c6", data["_c6"].cast(DoubleType()))
data = data.withColumn("_c9", data["_c9"].cast(DoubleType()))
data = data.withColumn("_c10", data["_c10"].cast(DoubleType()))
data = data.withColumn("_c3", data["_c3"].cast(IntegerType()))
data.select(F.min('_c18').alias('min'),
F.max('_c18').alias('max')).show()
data.select('_c18').describe().toPandas()
data.select('_c18').approxQuantile("_c18",[0.1, 0.25, 0.5, 0.75, 0.9],
0.01)
data.createOrReplaceTempView('NY_taxi')
req = """select * from NY_taxi where _c18>0 and _c18<100"""
data = spark.sql(req)
data.count()
data.select([F.count(F.when(F.isnan(c),c)).alias(c) for c in
data.columns]).show()
hist = data.select('_c18').rdd.flatMap(lambda x: x).histogram(100)
plt.figure(figsize=(22,6))
sns.barplot(list(map(lambda x: int(x), hist[0][:-1])), hist[1])
plt.show()
p = 0.017453292519943295
data = data.withColumn('distance', 0.6213712*12742*F.asin((0.5-
F.cos((data['_c10']-data['_c6'])*p)/2 +  F.cos(data['_c6']*p) *
F.cos(data['_c10']*p) * (1-F.cos((data['_c9']-
data['_c5'])*p))/2)**0.5))
data.filter(data['distance']>0).count()
data = data.filter(data['distance']>0)
data = data.filter(data['distance']<150)
data = data.withColumn('d_lon', data['_c5'] - data['_c9'] )
data = data.withColumn('d_lat', data['_c6'] - data['_c10'])
data = data.withColumn('lon_lat', (data['d_lon']**2 +
data['d_lat']**2)**0.5)
data = data.withColumn('dev_ll', data['d_lat']/data['lon_lat'])
g =  180/np.pi
```

```
data = data.withColumn('direction', F.when(data['d_lon']>0,
g*F.asin(data['dev_ll'])).when((data['d_lon']<0) & (data['d_lat']>0),
180-g*F.asin(data['dev_ll'])).when((data['d_lon']<0) &
(data['d_lat']<0), -180 - g*F.asin(data['dev_ll'])).otherwise(0))
data = data.filter(data['_c3']>0)
data = data.withColumn('dayofweek', F.dayofweek(data['_c1']))
data = data.withColumn('hour', F.hour(data['_c1']))
data = data.withColumn('peak_hours', F.when((data['hour']>=16) &
(data['hour']<20) & (data['dayofweek']!=7)&(data['dayofweek']!=0),
1).otherwise(0))
data = data.withColumn('night_time', F.when((data['hour']>=20) |
(data['hour']<6), 1).otherwise(0))
(trainingData, testData) = data.randomSplit([0.7, 0.3], seed=66)
continuous_variables =
['dayofweek','hour','peak_hours','night_time','_c3','distance','directi
on']
assembler =
VectorAssembler(inputCols=continuous_variables,outputCol='features')


trainingData =
assembler.setHandleInvalid("skip").transform(trainingData)
testData = assembler.setHandleInvalid("skip").transform(testData)
trainingData.limit(3).toPandas()['features'][0]
dt = DecisionTreeRegressor(featuresCol='features', labelCol='_c18')

model = dt.fit(trainingData)
predictions = model.transform(testData)
print(model)
evaluator =
RegressionEvaluator(labelCol='_c18',  predictionCol="prediction",
metricName="rmse")
rmse_train = evaluator.evaluate(model.transform(trainingData))
print("Train Root Mean Squared Error (RMSE) on test data = %g" %
rmse_train)
model.featureImportances
testData.select(F.mean(testData['_c18'])).show()
avr =
testData.agg(F.mean(testData['_c18']).alias("mean")).collect()[0]["mean
"]
testData_upd = testData.withColumn('subt',((testData['_c18'] -
avr)**2)**0.5)
```

```
testData_upd.agg(F.mean(testData_upd['subt'])).show()
gbt = GBTRegressor(featuresCol='features', labelCol='_c18')
m = gbt.fit(trainingData)
predictions = m.transform(testData)
evaluator =
RegressionEvaluator(labelCol='_c18',  predictionCol="prediction",
metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("GBT model. Root Mean Squared Error (RMSE) on test data = %g" %
rmse)
m.featureImportances
BB_zoom = (-74.3, -73.7, 40.5, 40.9)
nyc_map_zoom = plt.imread('/home/sai/Desktop/NYC_map_main.jpeg')
fig, axs = plt.subplots(1, 2, figsize=(16,10))
df = data.select('_c5', '_c6', '_c9', '_c10').limit(10000).toPandas()
axs[0].scatter(df._c5, df._c6, zorder=1, alpha=0.3, c='r', s=1)
axs[0].set_xlim((BB_zoom[0], BB_zoom[1]))
axs[0].set_ylim((BB_zoom[2], BB_zoom[3]))
axs[0].set_title('Pickup locations')
axs[0].imshow(nyc_map_zoom, zorder=0, extent=BB_zoom)
axs[1].scatter(df._c9, df._c10, zorder=1, alpha=0.3, c='r', s=1)
axs[1].set_xlim((BB_zoom[0], BB_zoom[1]))
axs[1].set_ylim((BB_zoom[2], BB_zoom[3]))
axs[1].set_title('Dropoff locations')
axs[1].imshow(nyc_map_zoom, zorder=0, extent=BB_zoom)
fig.show()
```

Imported librarie

**import pyspark.sql.functions as F** - List of built-in functions available for DataFrame.

**from pyspark.sql.types import DoubleType, IntegerType** – importing required datatypes
**from pyspark.ml.feature import VectorAssembler** – creates a single vector from multiple colums
**from pyspark.ml.regression import DecisionTreeRegressor** – for

regression in form of trees

**from pyspark.ml.evaluation import RegressionEvaluator** – evaluator for regression

**import numpy as np**

import pandas as pd - for handling arrays and dataframes

**import matplotlib.pyplot as plt**- creates a plotting area in a figure, plots some lines in a plotting area.

**import seaborn as sns** - for data visualization.

```
data = spark.read.csv('yellow_tripdata_2015*.csv')
data.createOrReplaceTempView('NY_taxi')
data.count()
data.printSchema()
```

The above code reads all csv files and merge them into one data frame create a temporary view SQL query for the reading data and verifies data by counting it and projecting schema of the data.

```
>>> data.count()
10906859
>>>
>>> data.printSchema()
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c14: string (nullable = true)
 |-- _c15: string (nullable = true)
 |-- _c16: string (nullable = true)
 |-- _c17: string (nullable = true)
 |-- _c18: string (nullable = true)

>>>
```

Fig 11: Projecting schema of the data

```
data =
data.drop('_c4','_c7','_c8','_c11','_c12','c13','_c14','_c15','_c16','_
c17')
data.printSchema()
data = data.withColumn("_c18", data["_c18"].cast(DoubleType()))
data.select(F.min('_c18').alias('min'),
F.max('_c18').alias('max')).show()
data.select('_c18').describe().toPandas()
data = data.filter(data['_c3']>0)
```

The above code is for data pre-processing, we drop the unwanted data columns from data to avoid computing the unwanted data check, whether the data is dropped by projecting schema, all the data columns are in type string. we should Cast them to double or integer data type accordingly to check min and max of taxi fare (to remove negative and outliers) checking mean and std_dev of data removes rides with passenger count with 0.

```
k>>> data.select(F.min('_c18').alias('min'), F.max('_c18').alias('max')).show()
+------+---------+
|   min|      max|
+------+---------+
|-958.4|111271.65|
+------+---------+

>>> data.select('_c18').describe().toPandas()
/home/team2/spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/pandas/conversion.py:87: UserWarning: toPand
wever, failed by the reason below:
  PyArrow >= 1.0.0 must be installed; however, it was not found.
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.
  warnings.warn(msg)
  summary                _c18
0   count            10906858
1    mean  15.641395247926114
2  stddev   36.41280207334372
3     min              -958.4
4     max           111271.65
>>>
```

Fig 12: Minimum and maximum of taxi fare

```
req = """select * from NY_taxi where _c18>0 and _c18<100"""
data = spark.sql(req)
data.count()
data.select([F.count(F.when(F.isnan(c),c)).alias(c) for c in
data.columns]).show()
```

The above code is used to remove outliers from data and check the count of rows and remove null values from data.

```
>>> data.printSchema()
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c18: string (nullable = true)
```

Fig 13: Data after removing outliners

```
hist = data.select('_c18').rdd.flatMap(lambda x: x).histogram(100)
plt.figure(figsize=(22,6))
sns.barplot(list(map(lambda x: int(x), hist[0][:-1])), hist[1])
plt.show()
```

The above code is for data visualization and analysis of taxi fare price vs no of trips.

```
p = 0.017453292519943295
data = data.withColumn('distance', 0.6213712*12742*F.asin((0.5-
F.cos((data['_c10']-data['_c6'])*p)/2 +  F.cos(data['_c6']*p) *
F.cos(data['_c10']*p) * (1-F.cos((data['_c9']-
data['_c5'])*p))/2)**0.5))
```

The code above uses latitude and longitude to get the distance and direction between two places. We then used the Haversine formula to determine the distance.

```
data = data.withColumn('dayofweek', F.dayofweek(data['_c1']))
data = data.withColumn('hour', F.hour(data['_c1']))
```

```
data = data.withColumn('peak_hours', F.when((data['hour']>=16) &
(data['hour']<20) & (data['dayofweek']!=7)&(data['dayofweek']!=0),
1).otherwise(0))
data = data.withColumn('night_time', F.when((data['hour']>=20) |
(data['hour']<6), 1).otherwise(0))
```

The code above is used to assess the day and hour to determine how highly dependent each hour is on the other, dividing daytime from night-time, and obtaining prices.

```
(trainingData, testData) = data.randomSplit([0.7, 0.3], seed=66)
continuous_variables =
['dayofweek','hour','peak_hours','night_time','_c3','distance','directi
on']
assembler =
VectorAssembler(inputCols=continuous_variables,outputCol='features')
trainingData =
assembler.setHandleInvalid("skip").transform(trainingData)
testData = assembler.setHandleInvalid("skip").transform(testData)
```

The above code divides the data into training and testing data of the model and skips the invalid data.

```
dt = DecisionTreeRegressor(featuresCol='features', labelCol='_c18')
predictions = model.transform(testData)
evaluator =
RegressionEvaluator(labelCol='_c18',  predictionCol="prediction",
metricName="rmse")
```

The above code is used to predict the taxi fare using decision tree regression and calculate the Root Mean Square Error (RMSE) to measure the difference between the predicted taxi fare and actual fare amount.

```
>>> evaluator = RegressionEvaluator(labelCol='_c18',  predictionCol="prediction", metricName="rmse")
>>> rmse_train = evaluator.evaluate(model.transform(trainingData))
>>> print("Train Root Mean Squared Error (RMSE) on test data = %g" % rmse_train)
Train Root Mean Squared Error (RMSE) on test data = 4.82634
>>> model.featureImportances
SparseVector(7, {1: 0.0002, 3: 0.0053, 5: 0.9856, 6: 0.009})
>>>
>>> testData.select(F.mean(testData['_c18'])).show()
+-----------------+
|        avg(_c18)|
+-----------------+
|15.45886646518172|
+-----------------+
```

Fig 14: RMSE on trained data and average of the taxi fare amount

```
>>> predictions = m.transform(testData)
>>> evaluator = RegressionEvaluator(labelCol='_c18',  predictionCol="prediction", metricName="rmse")
>>> rmse = evaluator.evaluate(predictions)
>>> print("GBT model. Root Mean Squared Error (RMSE) on test data = %g" % rmse)
GBT model. Root Mean Squared Error (RMSE) on test data = 4.48036
```

Fig15: RMSE of the GBT regression on test data

# Results

We will evaluate performance of validation dataset. For evaluation, Root Mean Square Error (RMSE) are widely adopted in many recommendation systems to measure the difference between the predicted scores and actual scores.

The model which has the lowest value of RMSE is considered to the best model and the most accurate one also. According to our calculations, we found that GBT is the most suitable model for this regression problem.

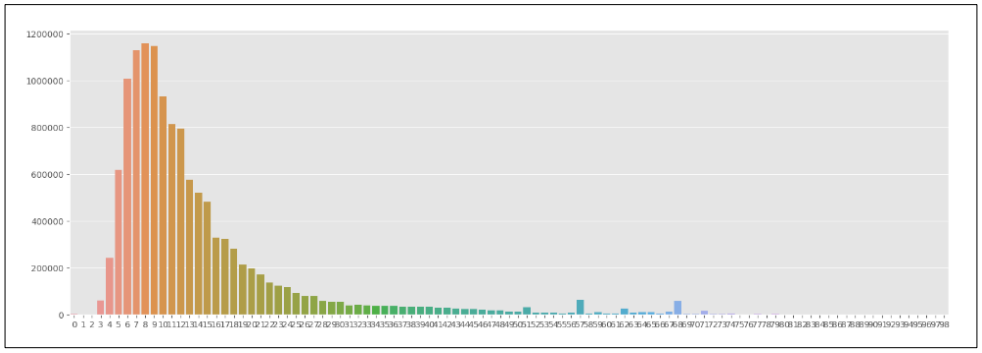


Fig 16: Graph of taxi fare price vs no of trips

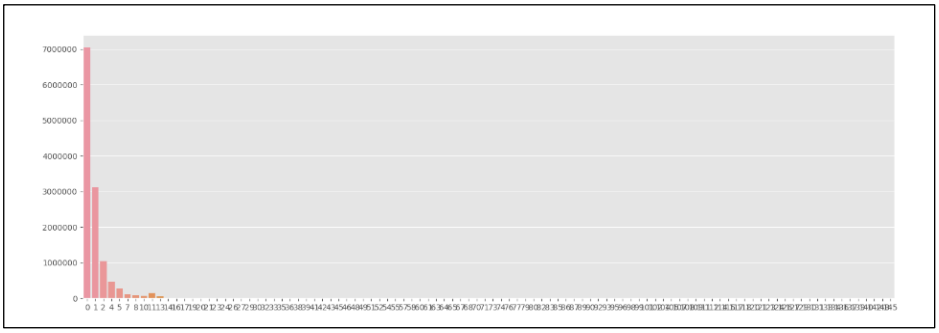


Fig 17: Graph of distance vs no of trips

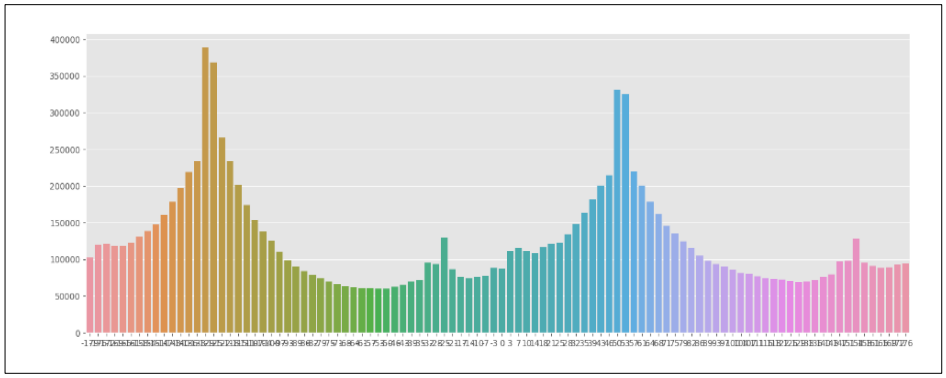


Fig 18: Graph of Direction vs no of trips

## Coordinate Transformation

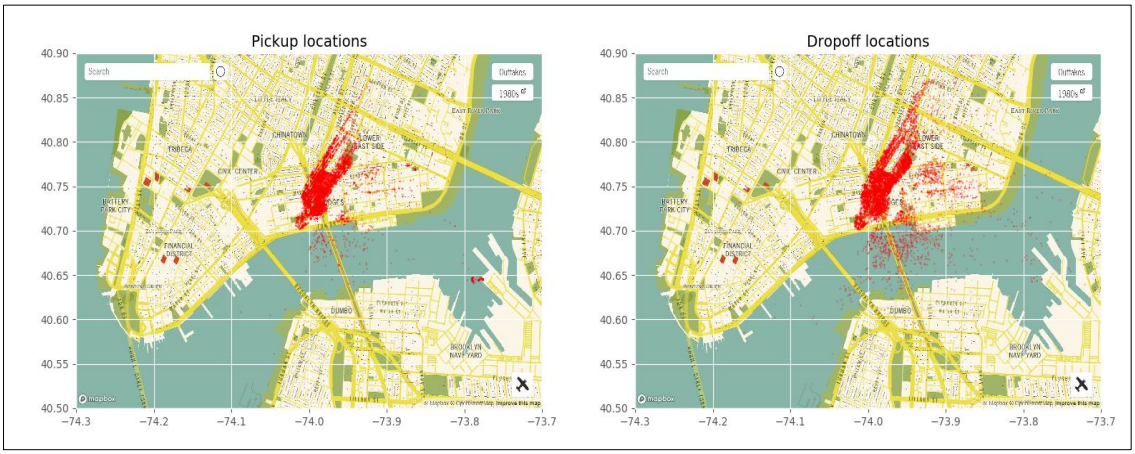



Fig 19: Transformation of location coordinates

Transforming the coordinates was another method used to further model the effect of the pickup and drop off locations. Most Manhattan's streets and avenues are organised in a grid pattern. With the hypothesis that the avenue or street may explain some of the effect of location, transforming the coordinates so that the splits in the GBT are aligned and perpendicular to the avenues and streets may yield better predictions

Time consumed to run 2016 January taxi dataset (2 Gb) is **7 min 26 sec**

Time consumed to run 2015 and 2016 taxi dataset(11GB) is **36 min 15 sec**

# 7. CONCLUSION

After training and testing the results shown are fairly accurate. Decision tree and GBT is useful in regression that helps to find the linear relation among the variables. Hence, we reached to the conclusion that GBT is the best because it gives more accurate value as compared to Decision tree regression model. That is why GBT algorithm is the best fit for the model selection as it has the lowest RMSE value. More further future scope is there if will apply more different types of approaches like XgBoost Regression technique or Ridge Regression technique.

# 8. REFERENCE

[1] Vanajakshi, L., S. C. Subramanian, and R. Sivanandan. "Travel time prediction under heterogeneous traffic conditions using global positioning system data from buses." IET intelligent transport systems 3.1 (2009): 1-9.

[2] https://www.analyticsvidhya.com/blog/2021/01/exploratory-data-analysis-on-nyc-taxi-trip-duration-dataset/

[3] https://towardsdatascience.com/new-york-taxi-data-set-analysis-7f3a9ad84850