databricksTaxi Price _ Data Preparation - Databricks

(https://databricks.com)

```
from pyspark.sql import SparkSession
from pyspark.sql.utils import AnalysisException
import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import col, to_timestamp
from pyspark.sql.functions import unix_timestamp, from_unixtime, round

spark = SparkSession.builder \
    .appName("read_s3_parquet") \
    .getOrCreate()


storage_account_name = ""
storage_account_access_key = ""
spark.conf.set(f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net", storage_account_access_key)
container_name = "taxidataset"

dem = spark.read.csv(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/dema
df = spark.read.parquet(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/t
weather = spark.read.csv(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/
holiday=spark.read.csv(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/ho
location_coordinates=spark.read.csv (f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-buc
```

Joining the weather dataset

```
weather_df=weather.select(F.col("datetime"),F.col("temp"))
weather_df = weather_df.withColumn("hourly_interval", F.date_trunc("hour", weather_df.datetime))
df = df.withColumn("hourly_interval", F.date_trunc("hour", df.tpep_pickup_datetime))
joined_df = df.join(weather_df, "hourly_interval", "left_outer")
joined_df = joined_df.drop("hourly_interval","datetime")
```

Joining the Holiday Dataset

```
holiday_data=holiday.select("date","holidayName")
data = joined_df.join(holiday_data, joined_df.tpep_pickup_datetime == holiday_data.date, "left_outer")
data=data.drop("date")
```

Joining Coordinates

```
location_coordinates= location_coordinates.select("LocationID","Latitude","Longitude")
data=data.join(location_coordinates,data.PULocationID==location_coordinates.LocationID)
data = data.withColumn("Latitude", F.col("Latitude").cast(DoubleType()))
data = data.withColumn("Longitude", F.col("Longitude").cast(DoubleType()))
```

New feature "isholiday"

```
#holiday = 1 , no holiday = 0
data = data.withColumn("isHoliday", F.when(data.holidayName.isNotNull(), F.lit(1)).otherwise(F.lit(0)))
data=data.drop("holidayName")
```

Modifying "isHoliday" to include weekends as holidays

```
data = data.withColumn('dayofweek', F.dayofweek(F.col('tpep_pickup_datetime')))
# New column'isWeekend' Saturdays and Sundays are as 1, others as 0
data = data.withColumn('isWeekend', F.when(F.col('dayofweek').isin([1, 7]), 1).otherwise(0))
data = data.withColumn(
    'isHoliday',
    F.when(
        (F.col('isHoliday') == 1) | (F.col('dayofweek').isin([7, 1])), 1
    ).otherwise(0)
)
data=data.drop("dayofweek","isWeekend")
```

New Feature trip_duration

```
data = data.withColumn(
    "trip_duration",
    F.round(
        (F.unix_timestamp("tpep_dropoff_datetime") - F.unix_timestamp("tpep_pickup_datetime")) / 60,
        2
    )
)
```

New Feature speed_mph

```
data = data.withColumn("trip_duration_hours", (F.col("trip_duration") / 60))
data = data.withColumn("speed_mph", F.round(F.col("trip_distance") / F.col("trip_duration_hours"),2))
data=data.drop('trip_duration_hours')
```

Splitting Date time to different Columns

```
data = data.withColumn("year", F.year("tpep_pickup_datetime")) \
    .withColumn("month", F.month("tpep_pickup_datetime")) \
    .withColumn("day", F.dayofmonth("tpep_pickup_datetime")) \
    .withColumn("hour", F.hour("tpep_pickup_datetime")) \
    .withColumn("minute", F.minute("tpep_pickup_datetime"))
```

Removing Tip Amount

```
data = data.withColumn("total_amount",(F.col("total_amount")-F.col("tip_amount")))
```

Adding Congestion surcharge to the total amount

```
def adjust_total_amount(data):
    sum_of_charges = col('fare_amount') + col('extra') + col('mta_tax') + \
                     col('tolls_amount') + col('improvement_surcharge') + \
                     col('congestion_surcharge')

    data = data.withColumn(
        'total_amount',
        F.when((F.col('total_amount')) != sum_of_charges, sum_of_charges)
        .otherwise(col('total_amount'))
    )

    return data
data = adjust_total_amount(data)
```

Adding airport fee to the total amount

```
data = data.withColumn(
    "total_amount",
    F.round(F.col("total_amount") + F.col("airport_fee"), 2)
)
```

Outliers

```
data = data.filter(
    (F.col('total_amount') > 5) &
    (F.col('total_amount') < 500) &
    (F.col('trip_duration') > 2) &
    (F.col('trip_duration') < 300) &
    (F.col('trip_distance') > 0.25) &
    (F.col('trip_distance') < 50) &
    (F.col('year') <= 2023) &
    (F.col('year') > 2018) &
    (F.col('passenger_count') > 0) &
    (F.col('passenger_count') < 7) &
    (F.col('speed_mph') <= 55)
)
```

Dropping unnecessary columns from the data

```
data=data.drop("tip_amount","fare_amount",'extra','mta_tax','tolls_amount','improvement_surcharge','congestion_s
urcharge',"airport_fee","RatecodeID","store_and_fwd_flag","LocationID")
```

```
data=data.na.drop()
```

```
display(data)
```

Loading final dataset to Azure

```
# data=data.coalesce(1)
# data.write.parquet(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-
bucket/final_dataset_1/")
```

Creating demand dataset

```
fin_df = df.select(F.col('tpep_pickup_datetime'), F.col('PULocationID'))
fin_df = fin_df.withColumn("pickup_timestamp", unix_timestamp("tpep_pickup_datetime", "yyyy-MM-dd
HH:mm:ss").cast("timestamp"))
fin_df = fin_df.withColumn("time_bins", from_unixtime(((unix_timestamp("pickup_timestamp") /
3600).cast("integer") * 3600), "yyyy-MM-dd HH:mm:ss"))
fin_df.drop(F.col('pickup_timestamp'))
demand_df = fin_df.groupBy(F.col('time_bins'),
F.col('PULocationID')).agg(F.count('time_bins').alias('no_of_pickups'))
demand_df = demand_df.orderBy(F.col('time_bins'))
```

Demand will be continued in "demand" notebook

databricks Price Forecasting (2)

(https://databricks.com)

```
from pyspark.sql import SparkSession
from pyspark.sql.utils import AnalysisException
import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import col, to_timestamp
from pyspark.sql.functions import unix_timestamp, from_unixtime, round
from pyspark.sql.types import StructType, StructField, IntegerType, TimestampType, DoubleType,BooleanType
from pyspark.sql.types import *

spark = SparkSession.builder \
    .appName("read_s3_parquet") \
    .getOrCreate()


storage_account_name = ""
storage_account_access_key = ""
spark.conf.set(f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net", storage_account_access_key)
container_name = "taxidataset"



df = spark.read.parquet(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/final_dat
dem = spark.read.csv(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/dema
```

Joining demand dataset with the original dataset with pickup datetime from original dataset and time bins from demand dataset.

```
dem=dem.withColumnRenamed("PULocationID","locationid")
dem = dem.withColumn("time_bins", to_timestamp(col("time_bins"), "yyyy-MM-dd HH:mm:ss"))
df = df.withColumn("tpep_pickup_datetime", to_timestamp(col("tpep_pickup_datetime"), "yyyy-MM-dd HH:mm:ss"))


df = df.withColumn('pickup_time_bins',
                   from_unixtime(round(unix_timestamp(col('tpep_pickup_datetime')) / 3600) * 3600))


df = df.join(dem,
                    (df['PULocationID'] == dem['locationid']) &
                    (df['pickup_time_bins'] == dem['time_bins']),
                    'inner')
```

Converting demand column to categorical column using the splitting point of three quartiles.

```
df = df.withColumn(
    "demand_category",
    F.when(col("no_of_pickups") <= 2, 1)
    .when((col("no_of_pickups") > 2) & (col("no_of_pickups") <= 19), 2)
    .otherwise(3)
)

# df=df.drop("no_of_pickups")
```

Create a new column with the squares of demand category to check if demand category squared can increase the impact of demand at the booking time on price prediction.

```
df=df.withColumn("squared_demand",F.col('demand_category')**2)
```

Keeping only necessary features for price prediction and dropping other columns.

```
data_ml=df.select("year","month","day","hour","minute","total_amount","temp","isHoliday","trip_duration","trip_d
istance","passenger_count","Latitude","Longitude","speed_mph","demand_category","squared_demand")
data_ml=data_ml.dropna()
```

Saving final cleaned dataset to azure data storage containers.

```
data = df.coalesce(1)
data.write.mode('overwrite').parquet(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pav
an-bucket/final_dataset_with_dem/")
```

Model Training

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

assembler = VectorAssembler(
    inputCols=["year","month","day","hour","minute",'temp','isHoliday',"passenger_count" ,'trip_duration',
'Latitude','Longitude", "trip_distance","speed_mph","demand_category","squared_demand"],
    outputCol='features'
)

data_ml = assembler.transform(data_ml)


train_data, test_data = data_ml.randomSplit([0.8, 0.2], seed=42)

lr = LinearRegression(featuresCol='features', labelCol='total_amount')
rf = RandomForestRegressor(featuresCol='features', labelCol='total_amount')
lr_model = lr.fit(train_data)
rf_model = rf.fit(train_data)
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
evaluator = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='r2')
evaluator_2 = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='rmse')
lr_r2 = evaluator.evaluate(lr_predictions)
rf_r2 = evaluator.evaluate(rf_predictions)
print(f"Linear Regression R^2: {lr_r2}")
print(f"Random Forest R^2: {rf_r2}")
lr_r2 = evaluator.evaluate(lr_predictions)
lr_rmse=evaluator_2.evaluate(lr_predictions)
rf_rmse = evaluator_2.evaluate(rf_predictions)
print(f"Random Forest RMSE: {lr_rmse}")
print(f"Random Forest RMSE: {rf_rmse}")

display(rf_predictions)
```

```
Linear Regression R^2: 0.9199944411463506
Random Forest R^2: 0.8915634094384501
Linear Regression R^2: 0.9199944411463506
Random Forest RMSE: 3.8313027973931852
Random Forest RMSE: 4.460405289736524
```

Table

	year	month	day	hour	minute	total_amount	temp	isHoliday	trip_d

1	2021	3	27	14	33	31.8	65.1	1	20.28
2	2021	3	29	0	45	12.8	53.2	0	8.5
3	2021	3	29	1	47	10.3	52.1	0	3.93
4	2021	3	29	6	29	35.3	46.1	0	25.17
5	2021	3	29	7	30	24.8	46.1	0	17.27
6	2021	3	29	7	33	11.8	46.1	0	9.28

8,978 rows | Truncated data

Above are the results of R squared score and RMSE which were chosen as metrics to evaluate the regression model. It can be seen that Linear regression and random forest classifier have almost similar R squared score.

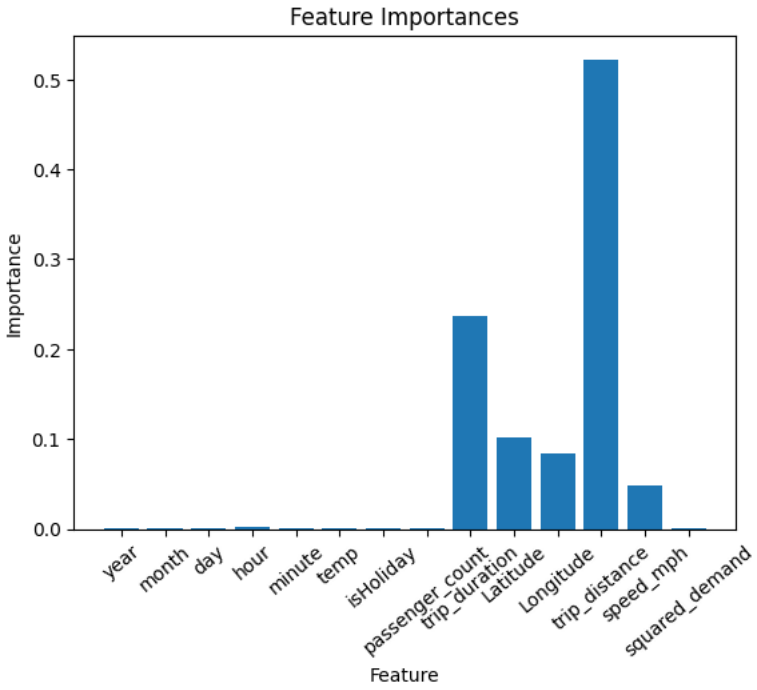
Random Forest Hyperparameter Tuning with parameters numTrees - [10, 20] and maxDepth - [5, 10]

```
from pyspark.ml.feature import VectorAssembler, MinMaxScaler
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
import matplotlib.pyplot as plt

feature_list = ["year","month","day","hour","minute",'temp','isHoliday',"passenger_count" ,'trip_duration',
'Latitude',"Longitude", "trip_distance","speed_mph","squared_demand"]
assembler = VectorAssembler(
    inputCols=feature_list,
    outputCol='features'
)
rf = RandomForestRegressor(featuresCol='features', labelCol='total_amount')
pipeline = Pipeline(stages = [assembler, rf])
paramGrid = (ParamGridBuilder()
    .addGrid(rf.numTrees, [10, 20])
    .addGrid(rf.maxDepth, [5, 10])
    .build())
evaluator = RegressionEvaluator(labelCol="total_amount", predictionCol="prediction", metricName="r2")
crossval = CrossValidator(estimator = pipeline, estimatorParamMaps = paramGrid, evaluator = evaluator, numFolds
= 3)
(trainingData, testData) = data_ml.randomSplit([0.8, 0.2])
cvModel = crossval.fit(trainingData)
predictions = cvModel.transform(testData)

rf_r2 = evaluator.evaluate(predictions)
print(f"Random Forest R^2: {rf_r2}")

display(rf_r2)
bestPipeline = cvModel.bestModel
bestModel = bestPipeline.stages[1]
importances = bestModel.featureImportances
x_values = list(range(len(importances)))
plt.bar(x_values, importances, orientation = 'vertical')
plt.xticks(x_values, feature_list, rotation=40)
plt.ylabel('Importance')
plt.xlabel('Feature')
plt.title('Feature Importances')
print('numTrees - ', bestModel.getNumTrees)
print('maxDepth - ', bestModel.getOrDefault('maxDepth'))
```

```
Random Forest R^2: 0.9225275006075078
```

```
0.9225275006075078
```

```
numTrees -  20
maxDepth -  10
```



Saving model to Azure

```
%scala
spark.sparkContext.hadoopConfiguration.set(
  "fs.azure.account.key.bia678.blob.core.windows.net",
  ""
)
```

```
# rf_model.save(f"wasb://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/rf_hyper/")
```

Scaling

2 Workers with 100% data

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

assembler = VectorAssembler(
    inputCols=["year","month","day","hour","minute",'temp','isHoliday',"passenger_count" ,'trip_duration',
'Latitude',"Longitude", "trip_distance","speed_mph","demand_category","squared_demand"],
    outputCol='features'
)

data_ml = assembler.transform(data_ml)


train_data, test_data = data_ml.randomSplit([0.8, 0.2], seed=42)

lr = LinearRegression(featuresCol='features', labelCol='total_amount')
rf = RandomForestRegressor(featuresCol='features', labelCol='total_amount',maxDepth=10,numTrees=20)
lr_model = lr.fit(train_data)
rf_model = rf.fit(train_data)
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
evaluator = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='r2')
evaluator_2 = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='rmse')
lr_r2 = evaluator.evaluate(lr_predictions)
rf_r2 = evaluator.evaluate(rf_predictions)
print(f"Linear Regression R^2: {lr_r2}")
print(f"Random Forest R^2: {rf_r2}")
lr_rmse=evaluator_2.evaluate(lr_predictions)
rf_rmse = evaluator_2.evaluate(rf_predictions)
print(f"Linear Regression RMSE: {lr_rmse}")
print(f"Random Forest RMSE: {rf_rmse}")
```

```
Linear Regression R^2: 0.9199944411463497
Random Forest R^2: 0.9199471360002306
Linear Regression RMSE: 3.831302797393211
Random Forest RMSE: 3.8324353034248535
```

3 Workers with 100% data

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

assembler = VectorAssembler(
    inputCols=["year","month","day","hour","minute",'temp','isHoliday',"passenger_count" ,'trip_duration',
'Latitude',"Longitude", "trip_distance","speed_mph","demand_category","squared_demand"],
    outputCol='features'
)

data_ml = assembler.transform(data_ml)


train_data, test_data = data_ml.randomSplit([0.8, 0.2], seed=42)

lr = LinearRegression(featuresCol='features', labelCol='total_amount')
rf = RandomForestRegressor(featuresCol='features', labelCol='total_amount',maxDepth=10,numTrees=20)
lr_model = lr.fit(train_data)
rf_model = rf.fit(train_data)
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
evaluator = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='r2')
evaluator_2 = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='rmse')
lr_r2 = evaluator.evaluate(lr_predictions)
rf_r2 = evaluator.evaluate(rf_predictions)
print(f"Linear Regression R^2: {lr_r2}")
print(f"Random Forest R^2: {rf_r2}")
lr_rmse=evaluator_2.evaluate(lr_predictions)
rf_rmse = evaluator_2.evaluate(rf_predictions)
print(f"Linear Regression RMSE: {lr_rmse}")
print(f"Random Forest RMSE: {rf_rmse}")
```

```
Linear Regression R^2: 0.9199944411463495
Random Forest R^2: 0.9199471411903842
Linear Regression RMSE: 3.8313027973932114
Random Forest RMSE: 3.832435179188646
```

4 Workers with 100% data

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

assembler = VectorAssembler(
    inputCols=["year","month","day","hour","minute",'temp','isHoliday',"passenger_count" ,'trip_duration',
'Latitude',"Longitude", "trip_distance","speed_mph","demand_category","squared_demand"],
    outputCol='features'
)

data_ml = assembler.transform(data_ml)


train_data, test_data = data_ml.randomSplit([0.8, 0.2], seed=42)

lr = LinearRegression(featuresCol='features', labelCol='total_amount')
rf = RandomForestRegressor(featuresCol='features', labelCol='total_amount',maxDepth=10,numTrees=20)
lr_model = lr.fit(train_data)
rf_model = rf.fit(train_data)
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
evaluator = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='r2')
evaluator_2 = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='rmse')
lr_r2 = evaluator.evaluate(lr_predictions)
rf_r2 = evaluator.evaluate(rf_predictions)
print(f"Linear Regression R^2: {lr_r2}")
print(f"Random Forest R^2: {rf_r2}")
lr_rmse=evaluator_2.evaluate(lr_predictions)
rf_rmse = evaluator_2.evaluate(rf_predictions)
print(f"Linear Regression RMSE: {lr_rmse}")
print(f"Random Forest RMSE: {rf_rmse}")
```

```
Linear Regression R^2: 0.9199944411463482
Random Forest R^2: 0.9199471390585762
Linear Regression RMSE: 3.831302797393243
Random Forest RMSE: 3.8324352302175306
```

```
from pyspark.sql import SparkSession


schema = StructType([
    StructField("Workers", IntegerType(), True),
    StructField("Time_Minutes", DoubleType(), True),
    StructField("Linear_Regression_R2", DoubleType(), True),
    StructField("Random_Forest_R2", DoubleType(), True)
])

data = [
    (2, 1.48 * 60, 0.9199944411463497, 0.9199471360002306),
    (3, 1.07 * 60, 0.9199944411463495, 0.9199471411903842),
    (4, 0.874 * 60, 0.9199944411463482, 0.9199471390585762)
]
df = spark.createDataFrame(data, schema=schema)
df.createOrReplaceTempView("workers_data")
```

```
%sql
SELECT Workers, Time_Minutes FROM workers_data
```

Table Visualization 1

	Workers	Time_Minutes
1	2	88.8
2	3	64.2
3	4	52.44

3 rows

```
%sql
SELECT Workers, Linear_Regression_R2, Random_Forest_R2 FROM workers_data
```

Table Visualization 1

	Workers	Linear_Regression_R2	Random_Forest_R2
1	2	0.9199944411463497	0.9199471360002306
2	3	0.9199944411463495	0.9199471411903842
3	4	0.9199944411463482	0.9199471390585762

3 rows

Scaling up

25% of data with 4 workers and 1 driver

```
totaldata = data_ml.count()
twenty_five = int(totaldata * 0.25)
print(twenty_five)
```

```
Cancelled
```

```
percent = 0.25
data_ml_sample = data_ml.sample(False,percent)
```

```
data_ml_sample.show()
```

```
+----+-----+---+----+------+------------+----+---------+-------------+-------------+---------------+----------+-----
------+---------+---------------+--------------+
|year|month|day|hour|minute|total_amount|temp|isHoliday|trip_duration|trip_distance|passenger_count|  Latitude|  Lon
gitude|speed_mph|demand_category|squared_demand|
+----+-----+---+----+------+------------+----+---------+-------------+-------------+---------------+----------+-----
------+---------+---------------+--------------+
|2021|    7| 15|   9|    25|        74.3|78.3|        0|        46.58|         16.9|            1.0|40.6895314|-74.1
744624|    21.77|              1|           1.0|
|2021|   11| 28|  16|    34|       90.05|41.0|        1|        78.97|        13.77|            1.0|40.6895314|-74.1
744624|    10.46|              1|           1.0|
|2022|    4| 14|  14|    21|       69.05|81.6|        0|        26.95|        13.74|            2.0|40.6895314|-74.1
744624|    30.59|              1|           1.0|
|2022|    2| 17|   7|    15|        21.0|52.1|        0|        15.27|          2.5|            1.0|40.8631189|-73.8
616476|     9.82|              1|           1.0|
|2021|    4| 17|  18|    25|        18.3|55.1|        1|        19.13|         3.54|            1.0|40.7258428|-73.9
774916|     11.1|              2|           4.0|
|2021|    4| 19|  11|    35|        20.8|62.9|        0|        20.35|          3.2|            1.0|40.7258428|-73.9
774916|     9.43|              2|           4.0|
|2021|    4| 22|  15|    56|        13.8|49.8|        0|        14.68|         1.61|            5.0|40.7258428|-73.9
774916|     6.58|              2|           4.0|
|2021|    4| 22|  15|    54|        12.8|49.8|        0|        12.92|         2.03|            1.0|40.7258428|-73.9
```

```
percent = 0.25
total_records = data_ml.count()
data_ml_sample_25 = data_ml.limit(int(total_records * percent))

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

assembler = VectorAssembler(
    inputCols=["year","month","day","hour","minute",'temp','isHoliday',"passenger_count" ,'trip_duration',
'Latitude',"Longitude", "trip_distance","speed_mph","demand_category","squared_demand"],
    outputCol='features'
)

data_ml_sample_25 = assembler.transform(data_ml_sample_25)
train_data, test_data = data_ml_sample_25.randomSplit([0.8, 0.2], seed=42)

lr = LinearRegression(featuresCol='features', labelCol='total_amount')
rf = RandomForestRegressor(featuresCol='features', labelCol='total_amount',maxDepth=10,numTrees=20)
lr_model = lr.fit(train_data)
rf_model = rf.fit(train_data)
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
evaluator = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='r2')
evaluator_2 = RegressionEvaluator(labelCol='total_amount', predictionCol='prediction', metricName='rmse')
lr_r2 = evaluator.evaluate(lr_predictions)
rf_r2 = evaluator.evaluate(rf_predictions)
print(f"Linear Regression R^2: {lr_r2}")
print(f"Random Forest R^2: {rf_r2}")
lr_rmse=evaluator_2.evaluate(lr_predictions)
rf_rmse = evaluator_2.evaluate(rf_predictions)
print(f"Linear Regression RMSE: {lr_rmse}")
print(f"Random Forest RMSE: {rf_rmse}")
```

```
Linear Regression R^2: 0.9213905664256661
Random Forest R^2: 0.9301000969807838
Linear Regression RMSE: 3.8628256203575178
Random Forest RMSE: 3.6541200236163736
```

50 percent of the data with 4 workers and 1 driver

```
Linear Regression R^2: 0.9199021723800136
Random Forest R^2: 0.9296063695070713
Linear Regression RMSE: 3.845726491427964
Random Forest RMSE: 3.640762243298417
```

databricks Demand Prediction

(https://databricks.com)

```
from pyspark.sql import SparkSession
from pyspark.sql.utils import AnalysisException
import pyspark.sql.functions as F
import pyspark.sql.types as T
from sklearn.metrics import r2_score
# from prophet import Prophet
from pyspark.sql.functions import unix_timestamp, from_unixtime
import pandas as pd
import numpy as np
from pyspark.ml.feature import VectorAssembler, StringIndexer, MinMaxScaler
from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, GBTClassifie
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.classification import GBTClassifier
import sklearn
from sklearn.metrics import classification_report, confusion_matrix
```

Building spark session and importing all datasets from azure storage containers.

```
spark = SparkSession.builder \
    .appName("read_s3_parquet") \
    .getOrCreate()
```

```
storage_account_name = "bia678"
storage_account_access_key = "Okp/ALAV3dep7SNjenw9n8nx5Oojv2DFtOOZ3FRYzzpjIOviTSS3JmpjJAsMskG1/bqZFn6Id2HO+AStB0MDTQ==
spark.conf.set(f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net", storage_account_access_key)
container_name = "taxidataset"

df = spark.read.parquet(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/t
weather = spark.read.csv(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/
holiday=spark.read.csv(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/ho
location_coordinates=spark.read.csv (f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-buc
dem = spark.read.csv(f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-bucket/dataset/dema
```

```
%scala
spark.sparkContext.hadoopConfiguration.set(
  "fs.azure.account.key.bia678.blob.core.windows.net",
  "Okp/ALAV3dep7SNjenw9n8nx5Oojv2DFtOOZ3FRYzzpjIOviTSS3JmpjJAsMskG1/bqZFn6Id2HO+AStB0MDTQ=="
)
```

Creating "is_weekend" column from the date information present in "time_bins" column. This column contains 0 for weekends and 1 for weekdays.

```
dem = dem.withColumn(
    "is_weekend",
    F.when(F.dayofweek(F.col("time_bins")).isin([1, 5]), 1).otherwise(0)
)

dem = dem.withColumn("is_weekend", F.col("is_weekend").cast("boolean"))
```

Joining with weather dataset with "time_bins" in demand dataset on "datetime" in weather dataset to add temperature feature to the demand dataset.

```
weather=weather.select("datetime","temp")
```

```
dem = dem.withColumn("time_bins", F.to_timestamp(F.col("time_bins"), "yyyy-MM-dd HH:mm:ss"))
weather = weather.withColumn("datetime", F.to_timestamp(F.col("datetime"), "yyyy-MM-dd'T'HH:mm:ss'Z'"))
dem = dem.join(weather, dem.time_bins == weather.datetime, "inner")
dem=dem.drop('datetime')
```

Filtering demand dataset to contain records only from Jan, 2021 to Dec, 2022.

```
dem = dem.where((F.col('time_bins') >= '2021-01-01 00:00:00') & (F.col('time_bins') < '2022-12-31
11:59:59')).orderBy(F.col('time_bins'))
```

Splitting time bins column to year, month, day, hour, minute columns for training the machine learning algorithm.

```
dem = dem.withColumn("year", F.year("time_bins")) \
    .withColumn("month", F.month("time_bins")) \
    .withColumn("day", F.dayofmonth("time_bins")) \
    .withColumn("hour", F.hour("time_bins")) \
    .withColumn("minute", F.minute("time_bins"))
```

Joining with location dataset on pickup location ID to get latitude and longitude data for each pickup location.

```
location_coordinates= location_coordinates.select("LocationID","Latitude","Longitude")
dem=dem.join(location_coordinates,dem.PULocationID==location_coordinates.LocationID)
```

Casting columns to appropriate datatypes.

```
dem = dem.withColumn("no_of_pickups", F.col("no_of_pickups").cast("int"))
dem = dem.withColumn("PULocationID", F.col("PULocationID").cast("int"))
dem = dem.withColumn("Latitude", F.col("Latitude").cast("double"))
dem = dem.withColumn("Longitude", F.col("Longitude").cast("double"))
dem=dem.drop('LocationID',"PULocationID")
```

Removing outliers in no_of_pickups column - Here, we are considering only records where number of pickups is lower than 200 since the frequency of records over 200 is very low and have been considered as outliers by box plot.

```
dem = dem.filter(F.col("no_of_pickups") <= 200)
```

Converting no_of_pickups to demand_category. The splitting point for each category has been obtained by breaking the dataset into 3 equal quartiles. 1/3rd of the data had less than or equal to 2 pickups and the next 1/3rd had pickups between 2 and 19 pickups.

```
dem = dem.withColumn(
    "demand_category",
    F.when(F.col("no_of_pickups") <= 2, 1)           # Low demand
    .when((F.col("no_of_pickups") > 2) & (F.col("no_of_pickups") <= 19), 2)  # Medium demand
    .otherwise(3)                               # High demand
)
```

Below, the final dataset is shown which is being used by the demand model to classify demand into low, medium or high.

```
display(dem)
```

Table Data Profile 2

	time_bins	no_of_pickups	is_weekend	temp	year	month	day	hou
1	2021-01-01T00:00:00Z	1	false	37.1	2021	1	1	0
2	2021-01-01T00:00:00Z	1	false	37.1	2021	1	1	0
3	2021-01-01T00:00:00Z	10	false	37.1	2021	1	1	0
4	2021-01-01T00:00:00Z	3	false	37.1	2021	1	1	0
5	2021-01-01T00:00:00Z	2	false	37.1	2021	1	1	0
6	2021-01-01T00:00:00Z	9	false	37.1	2021	1	1	0

10,000 rows | Truncated data

```
dem_df = dem
```

Model Training

```
assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

train_data, test_data = dem_df.randomSplit([0.8, 0.2], seed=42)

classifiers = {
    "Logistic Regression": LogisticRegression(labelCol="demand_category", featuresCol="scaledFeatures"),
    "Decision Tree": DecisionTreeClassifier(labelCol="demand_category", featuresCol="scaledFeatures"),
    "Random Forest": RandomForestClassifier(labelCol="demand_category", featuresCol="scaledFeatures"),
}

for name, classifier in classifiers.items():
    pipeline = Pipeline(stages=[assembler, featureScaler, classifier])

    model = pipeline.fit(train_data)

    predictions = model.transform(test_data)

    y_true = predictions.select(['demand_category']).collect()
    y_pred = predictions.select(['prediction']).collect()

    print(classification_report(y_true, y_pred))

    evaluator = MulticlassClassificationEvaluator(
        labelCol="demand_category", predictionCol="prediction", metricName="accuracy")

    accuracy = evaluator.evaluate(predictions)
    print(f"{name} classifier accuracy: {accuracy:.3f}")
```

```
              precision    recall  f1-score   support

           1       0.44      0.70      0.55    114530
           2       0.51      0.04      0.08     85348
           3       0.46      0.51      0.49     95046

    accuracy                           0.45    294924
   macro avg       0.47      0.42      0.37    294924
weighted avg       0.47      0.45      0.39    294924

Logistic Regression classifier accuracy: 0.452
              precision    recall  f1-score   support

           1       0.82      0.79      0.81    114530
           2       0.55      0.57      0.56     85348
           3       0.78      0.79      0.78     95046

    accuracy                           0.73    294924
   macro avg       0.72      0.72      0.72    294924
weighted avg       0.73      0.73      0.73    294924
```

Logistic regression, decision tree classifier and random forest classifier are the three models which have been tested on this dataset. We have chosen accuracy as the metric to compare the classification models. From the outputs above, it can be seen that decision tree classifier and random forest classifier make predictions with similar accuracy.

Hyperparameter tuning - Tuning decision tree model with parameters - maxDepth 5 and 10, maxBins 32 and 64 and minInstancesPerNode 1, 3 and 5.

```
labelIndexer = StringIndexer(inputCol="demand_category", outputCol="indexedLabel").fit(dem_df)

# Define a VectorAssembler to assemble the features into a "features" column
assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude","is_weekend"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

# Split the data into training and test sets
train_data, test_data = dem_df.randomSplit([0.8, 0.2], seed=42)

# Define the classifier algorithms
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="scaledFeatures", maxDepth=2)

pipeline = Pipeline(stages=[labelIndexer, assembler, featureScaler, dt])

dtparamGrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [5, 10])
             .addGrid(dt.maxBins, [32, 64])
             .addGrid(dt.minInstancesPerNode, [1, 3, 5])
             .build())

evaluator = MulticlassClassificationEvaluator(
        labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")

dtcv = CrossValidator(estimator = pipeline,
                      estimatorParamMaps = dtparamGrid,
                      evaluator = evaluator,
                      numFolds = 5)

dtcvModel = dtcv.fit(train_data)
predictions = dtcvModel.transform(test_data)
accuracy = evaluator.evaluate(predictions)
best_model = dtcvModel.bestModel
dtModel = best_model.stages[-1]
print(f"Accuracy: {accuracy:.3f}")
print('maxBins - ', dtModel.getMaxBins())
print('maxDepth - ', dtModel.getMaxDepth())
print('minInstancesPerNode - ', dtModel.getMinInstancesPerNode())
```

```
Accuracy: 0.820
maxBins -  64
maxDepth -  10
minInstancesPerNode -  5
```

Above are the best parameters predicted by CrossValidator and the accuracy has increased by 10%.

Rerunning decision tree classifier with tuned hyperparameters

```
assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

train_data, test_data = dem_df.randomSplit([0.8, 0.2], seed=42)
classifier = DecisionTreeClassifier(labelCol="demand_category",
                                    featuresCol="scaledFeatures",
                                    maxBins = 64,
                                    maxDepth = 10,
                                    minInstancesPerNode = 5)
pipeline = Pipeline(stages=[assembler, featureScaler, classifier])

model = pipeline.fit(train_data)

predictions = model.transform(test_data)

y_true = predictions.select(['demand_category']).collect()
y_pred = predictions.select(['prediction']).collect()

print(classification_report(y_true, y_pred))

evaluator = MulticlassClassificationEvaluator(
    labelCol="demand_category", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)
print(f"Decision tree classifier accuracy: {accuracy:.3f}")
```

```
              precision    recall  f1-score   support

           1       0.87      0.84      0.85    114530
           2       0.67      0.69      0.68     85348
           3       0.87      0.88      0.88     95046

    accuracy                           0.81    294924
   macro avg       0.80      0.80      0.80    294924
weighted avg       0.81      0.81      0.81    294924

Decision tree classifier accuracy: 0.810
```

Hyperparameter tuning - Random Forest - parameters used are maxDepth - 5, 10 and numTrees - 10, 20.

```
labelIndexer = StringIndexer(inputCol="demand_category", outputCol="indexedLabel").fit(dem_df)

# Define a VectorAssembler to assemble the features into a "features" column
assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude","is_weekend"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

# Split the data into training and test sets
train_data, test_data = dem_df.randomSplit([0.8, 0.2], seed=42)

# Define the classifier algorithms
dt = RandomForestClassifier(labelCol="indexedLabel", featuresCol="scaledFeatures")

pipeline = Pipeline(stages=[labelIndexer, assembler, featureScaler, dt])

dtparamGrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [5, 10])
             .addGrid(dt.numTrees, [10, 20])
             .build())

evaluator = MulticlassClassificationEvaluator(
        labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")

dtcv = CrossValidator(estimator = pipeline,
                      estimatorParamMaps = dtparamGrid,
                      evaluator = evaluator,
                      numFolds = 5)

dtcvModel = dtcv.fit(train_data)
predictions = dtcvModel.transform(test_data)
accuracy = evaluator.evaluate(predictions)
best_model = dtcvModel.bestModel
dtModel = best_model.stages[-1]
print(f"Accuracy: {accuracy:.3f}")
print('numTrees - ', dtModel.getNumTrees)
print('maxDepth - ', dtModel.getOrDefault('maxDepth'))
```

```
Accuracy: 0.794
numTrees -  20
maxDepth -  10
```

Saving the model to azure storage container using model.save

```
model.save(f"wasb://{container_name}@{storage_account_name}.blob.core.windows.net/pavan-
bucket/decision_tree_for_demand/")
```

Scaling with 2 workers and 1 driver for 100% of the data.

```
assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

train_data, test_data = dem_df.randomSplit([0.8, 0.2], seed=42)

classifiers = {
    "Logistic Regression": LogisticRegression(labelCol="demand_category", featuresCol="scaledFeatures"),
    "Decision Tree": DecisionTreeClassifier(labelCol="demand_category", featuresCol="scaledFeatures", maxBins =
64, maxDepth = 10, minInstancesPerNode = 5),
    "Random Forest": RandomForestClassifier(labelCol="demand_category", featuresCol="scaledFeatures", numTrees =
20, maxDepth = 10),
}

for name, classifier in classifiers.items():
    pipeline = Pipeline(stages=[assembler, featureScaler, classifier])

    model = pipeline.fit(train_data)

    predictions = model.transform(test_data)

    y_true = predictions.select(['demand_category']).collect()
    y_pred = predictions.select(['prediction']).collect()

    print(classification_report(y_true, y_pred))

    evaluator = MulticlassClassificationEvaluator(
        labelCol="demand_category", predictionCol="prediction", metricName="accuracy")

    accuracy = evaluator.evaluate(predictions)
    print(f"{name} classifier accuracy: {accuracy:.3f}")
```

```
              precision    recall  f1-score   support

           1       0.45      0.70      0.55    114876
           2       0.51      0.04      0.08     85001
           3       0.46      0.51      0.49     95338

    accuracy                           0.45    295215
   macro avg       0.47      0.42      0.37    295215
weighted avg       0.47      0.45      0.39    295215

Logistic Regression classifier accuracy: 0.453
              precision    recall  f1-score   support

           1       0.88      0.83      0.85    114876
           2       0.66      0.70      0.68     85001
           3       0.87      0.88      0.88     95338

    accuracy                           0.81    295215
   macro avg       0.80      0.80      0.80    295215
weighted avg       0.81      0.81      0.81    295215
```

Scaling for 3 workers and 1 driver with 100% of the data.

```
assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

train_data, test_data = dem_df.randomSplit([0.8, 0.2], seed=42)

classifiers = {
    "Logistic Regression": LogisticRegression(labelCol="demand_category", featuresCol="scaledFeatures"),
    "Decision Tree": DecisionTreeClassifier(labelCol="demand_category", featuresCol="scaledFeatures", maxBins =
64, maxDepth = 10, minInstancesPerNode = 5),
    "Random Forest": RandomForestClassifier(labelCol="demand_category", featuresCol="scaledFeatures", numTrees =
20, maxDepth = 10),
}

for name, classifier in classifiers.items():
    pipeline = Pipeline(stages=[assembler, featureScaler, classifier])

    model = pipeline.fit(train_data)

    predictions = model.transform(test_data)

    y_true = predictions.select(['demand_category']).collect()
    y_pred = predictions.select(['prediction']).collect()

    print(classification_report(y_true, y_pred))

    evaluator = MulticlassClassificationEvaluator(
        labelCol="demand_category", predictionCol="prediction", metricName="accuracy")

    accuracy = evaluator.evaluate(predictions)
    print(f"{name} classifier accuracy: {accuracy:.3f}")
```

```
              precision    recall  f1-score   support

           1       0.45      0.70      0.55    115153
           2       0.49      0.05      0.08     84623
           3       0.46      0.52      0.49     95697

    accuracy                           0.46    295473
   macro avg       0.47      0.42      0.37    295473
weighted avg       0.47      0.46      0.40    295473

Logistic Regression classifier accuracy: 0.455
              precision    recall  f1-score   support

           1       0.88      0.83      0.85    115153
           2       0.66      0.71      0.69     84623
           3       0.87      0.88      0.88     95697

    accuracy                           0.81    295473
   macro avg       0.80      0.81      0.80    295473
weighted avg       0.82      0.81      0.81    295473
```

Scaling 25% data with 4 workers and 1 driver.

```
percent = 0.25
total_records = dem_df.count()
dem_df_sample_25 = dem_df.limit(int(total_records * percent))


assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

train_data, test_data = dem_df_sample_25.randomSplit([0.8, 0.2], seed=42)

classifiers = {
    "Logistic Regression": LogisticRegression(labelCol="demand_category", featuresCol="scaledFeatures"),
    "Decision Tree": DecisionTreeClassifier(labelCol="demand_category", featuresCol="scaledFeatures", maxBins =
64, maxDepth = 10, minInstancesPerNode = 5),
    "Random Forest": RandomForestClassifier(labelCol="demand_category", featuresCol="scaledFeatures", numTrees =
20, maxDepth = 10),
}

for name, classifier in classifiers.items():
    pipeline = Pipeline(stages=[assembler, featureScaler, classifier])

    model = pipeline.fit(train_data)

    predictions = model.transform(test_data)

    y_true = predictions.select(['demand_category']).collect()
    y_pred = predictions.select(['prediction']).collect()

    print(classification_report(y_true, y_pred))

    evaluator = MulticlassClassificationEvaluator(
        labelCol="demand_category", predictionCol="prediction", metricName="accuracy")

    accuracy = evaluator.evaluate(predictions)
    print(f"{name} classifier accuracy: {accuracy:.3f}")
```

```
              precision    recall  f1-score   support

           1       0.38      0.87      0.53     27858
           2       0.28      0.02      0.04     21048
           3       0.33      0.11      0.17     24919

    accuracy                           0.37     73825
   macro avg       0.33      0.33      0.24     73825
weighted avg       0.33      0.37      0.26     73825

Logistic Regression classifier accuracy: 0.461
              precision    recall  f1-score   support

           1       0.81      0.91      0.85     29348
           2       0.66      0.59      0.63     20719
           3       0.89      0.84      0.87     23758

    accuracy                           0.80     73825
   macro avg       0.79      0.78      0.78     73825
weighted avg       0.79      0.80      0.79     73825
```

Scaling 50% data with 4 workers and 1 driver.

```
percent = 0.50
total_records = dem_df.count()
dem_df_sample_50 = dem_df.limit(int(total_records * percent))


assembler = VectorAssembler(
    inputCols=["temp", "year", "month", "day", "hour", "Latitude", "Longitude"],
    outputCol="features"
)

featureScaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

train_data, test_data = dem_df_sample_50.randomSplit([0.8, 0.2], seed=42)

classifiers = {
    "Logistic Regression": LogisticRegression(labelCol="demand_category", featuresCol="scaledFeatures"),
    "Decision Tree": DecisionTreeClassifier(labelCol="demand_category", featuresCol="scaledFeatures", maxBins =
64, maxDepth = 10, minInstancesPerNode = 5),
    "Random Forest": RandomForestClassifier(labelCol="demand_category", featuresCol="scaledFeatures", numTrees =
20, maxDepth = 10),
}

for name, classifier in classifiers.items():
    pipeline = Pipeline(stages=[assembler, featureScaler, classifier])

    model = pipeline.fit(train_data)

    predictions = model.transform(test_data)

    y_true = predictions.select(['demand_category']).collect()
    y_pred = predictions.select(['prediction']).collect()

    print(classification_report(y_true, y_pred))

    evaluator = MulticlassClassificationEvaluator(
        labelCol="demand_category", predictionCol="prediction", metricName="accuracy")

    accuracy = evaluator.evaluate(predictions)
    print(f"{name} classifier accuracy: {accuracy:.3f}")
```

```
              precision    recall  f1-score   support

           1       0.43      0.57      0.49     61793
           2       0.29      0.02      0.04     42206
           3       0.30      0.44      0.36     43274

    accuracy                           0.37    147273
   macro avg       0.34      0.34      0.30    147273
weighted avg       0.35      0.37      0.32    147273

Logistic Regression classifier accuracy: 0.462
              precision    recall  f1-score   support

           1       0.50      0.48      0.49     59220
           2       0.36      0.35      0.36     41674
           3       0.42      0.45      0.43     46379

    accuracy                           0.43    147273
   macro avg       0.43      0.43      0.43    147273
weighted avg       0.43      0.43      0.43    147273
```

Scaling 100% data with 4 workers and 1 driver.

```
              precision    recall  f1-score   support

           1       0.44      0.70      0.55    114530
           2       0.51      0.04      0.08     85348
           3       0.46      0.51      0.49     95046

    accuracy                           0.45    294924
   macro avg       0.47      0.42      0.37    294924
```

```
weighted avg       0.47      0.45      0.39    294924

Logistic Regression classifier accuracy: 0.452
              precision    recall  f1-score   support

           1       0.87      0.84      0.85    114530
           2       0.67      0.69      0.68     85348
           3       0.87      0.88      0.88     95046

    accuracy                           0.81    294924
   macro avg       0.80      0.80      0.80    294924
weighted avg       0.81      0.81      0.81    294924
```