Boston University Metropolitan College

BOSTON
UNIVERSITY

MET CS777- Big Data Analysis

Flight Price Analysis Using Pyspark

Nidhi Desai

1.Introduction

Project Goal

This project aims to examine flight pricing information sourced from Expedia for significant US airports between April 16, 2022, and October 5, 2022. The dataset encompasses various flight-related details, including fare information, flight schedules, and attributes of each flight leg. The objective is to uncover patterns and insights related to flight prices

Scope of the Project

- Exploratory Data Analysis (EDA): Analyze the dataset to identify trends, fare distributions, and factors affecting flight pricing.
- Predictive Modeling: Train a machine learning model to predict flight prices using various features and evaluate the performance.
- Clustering Analysis: Apply clustering techniques to segment flights into groups based on similarities to understand customer behaviors and fare patterns.

2.Dataset Description

The dataset used for analysis contains 27 columns of which we will focus on the following columns for the analysis:

- **searchDate**: The date (YYYY-MM-DD) when the entry was recorded from Expedia.
- **flightDate**: The date (YYYY-MM-DD) of the flight.
- **startingAirport**: Three-character IATA code of the departure airport.
- **destinationAirport**: Three-character IATA code of the arrival airport.
- **isBasicEconomy**: Boolean indicating whether the ticket is for basic economy.
- **isRefundable**: Boolean indicating if the ticket is refundable.
- **isNonStop**: Boolean indicating if the flight is non-stop.
- **baseFare**: The price of the ticket (in USD).
- **totalFare**: The total ticket price including taxes and fees (in USD).
- **seatsRemaining**: Number of seats remaining for the flight.
- **totalTravelDistance**: Total travel distance in miles (may be missing).
- **segmentsDepartureTimeEpochSeconds**: Departure times in Unix format for each leg of the trip, separated by '||'.
- **segmentsArrivalTimeEpochSeconds**: Arrival times in Unix format for each leg of the trip, separated by '||'.
- **segmentsAirlineName**: Airline names servicing each leg of the trip, separated by '||'.
- **segmentsCabinCode**: Cabin classes for each leg of the trip, separated by '||'.
- **segmentsDistance**: Distances traveled for each leg of the trip, separated by '||'.

3.Overview

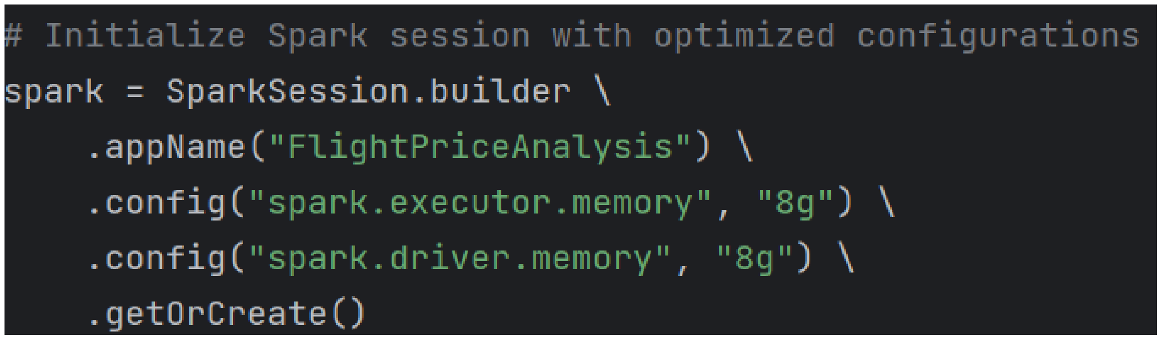
Overview of PySpark

PySpark is the Python API for Apache Spark, a very powerful open-source distributed computing system with high-performance data processing and analytics capabilities. It is positioned to provide scalable data processing and sophisticated analytics in large datasets and, hence, can be applied to big data cases. In the following project, PySpark will be used for data preprocessing, EDA, predictive modeling, and clustering.

Google Cloud Platform.

The script is run on GCP, which offers an infrastructure for high performance, assisting data processing and analysis. In terms of scalable resources and managed services to run PySpark workflows, GCP avails computational capacity adjustment based on dataset size and task complexity. This integration of PySpark with GCP gives users cloud-based scalability, reliability, and access to advanced tools and services for the management and analysis of big data.

Setup and Configuration



```python
# Initialize Spark session with optimized configurations
spark = SparkSession.builder \
    .appName("FlightPriceAnalysis") \
    .config("spark.executor.memory", "8g") \
    .config("spark.driver.memory", "8g") \
    .getOrCreate()
```

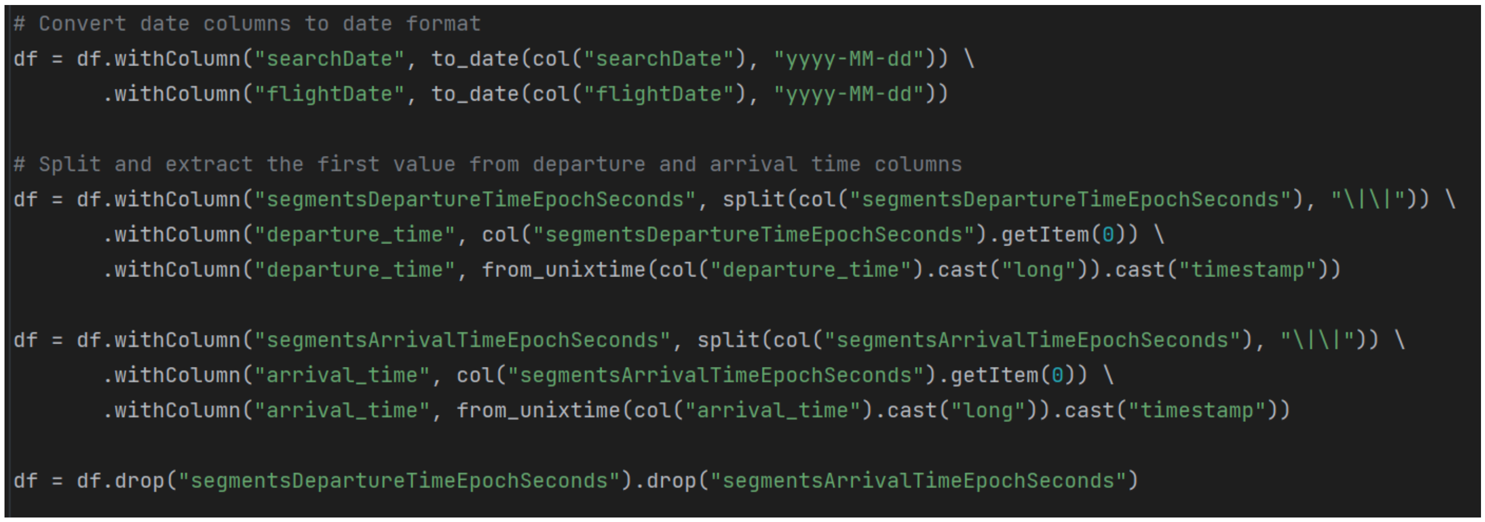
These settings are specified to ensure that the Spark jobs run efficiently and can handle large volumes of data. The session is created using SparkSession.builder, which allows for easy management and configuration of Spark tasks.

4.Preprocessing

The preprocessing of the dataset involves several key steps to ensure data quality and prepare it for analysis. These are the preprocessing steps performed in this project:

Data Transformation (Date and Time Conversion):

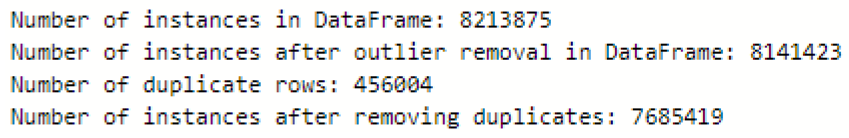
Converting data into appropriate formats and structures makes it suitable for analysis**.** searchDate and flightDate are converted to date format using to_date(). segmentsDepartureTimeEpochSeconds and segmentsArrivalTimeEpochSeconds (both in Unix format) are split into individual components. The first value from these components is extracted and converted to timestamp format using from_unixtime(). After this the original column was dropped. conversion into standardized formats allows for effective time-based analysis, while transforming categorical variables into numerical formats facilitates their use in machine learning models.



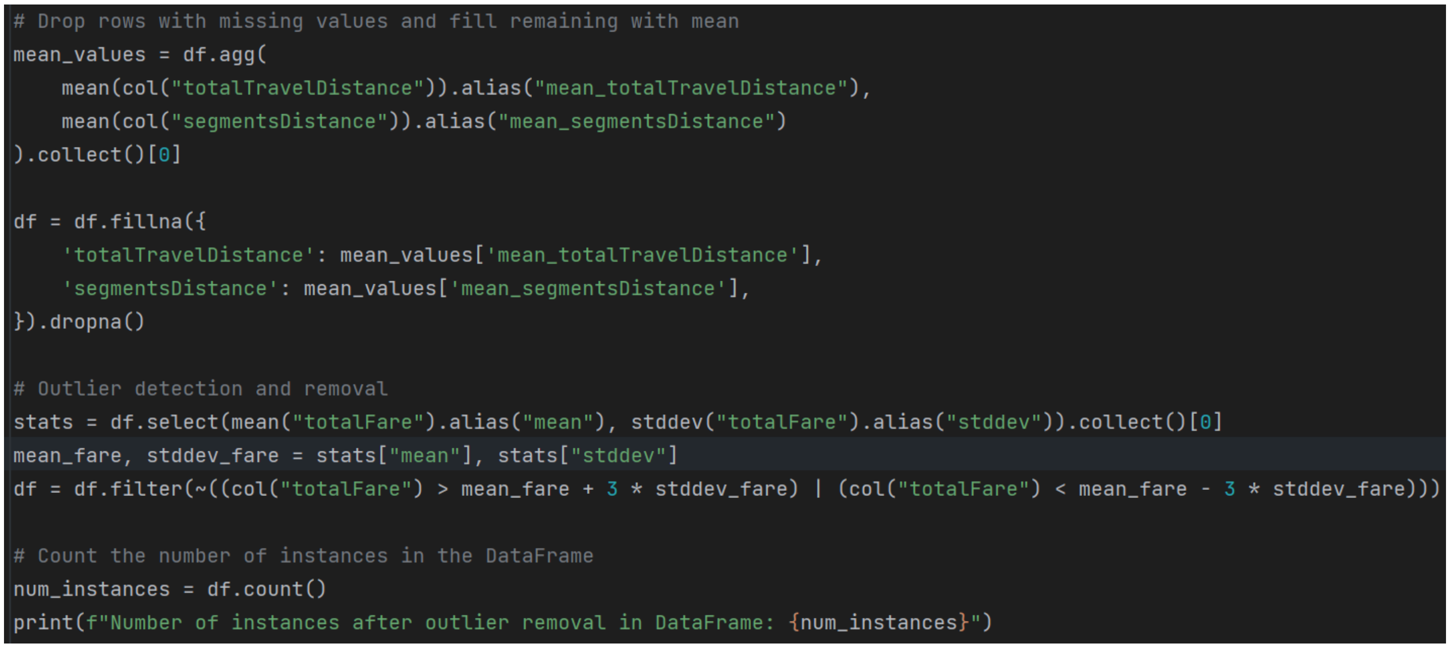
```
# Convert date columns to date format
df = df.withColumn("searchDate", to_date(col("searchDate"), "yyyy-MM-dd")) \
       .withColumn("flightDate", to_date(col("flightDate"), "yyyy-MM-dd"))

# Split and extract the first value from departure and arrival time columns
df = df.withColumn("segmentsDepartureTimeEpochSeconds", split(col("segmentsDepartureTimeEpochSeconds"), "\|\|")) \
       .withColumn("departure_time", col("segmentsDepartureTimeEpochSeconds").getItem(0)) \
       .withColumn("departure_time", from_unixtime(col("departure_time").cast("long")).cast("timestamp"))

df = df.withColumn("segmentsArrivalTimeEpochSeconds", split(col("segmentsArrivalTimeEpochSeconds"), "\|\|")) \
       .withColumn("arrival_time", col("segmentsArrivalTimeEpochSeconds").getItem(0)) \
       .withColumn("arrival_time", from_unixtime(col("arrival_time").cast("long")).cast("timestamp"))

df = df.drop("segmentsDepartureTimeEpochSeconds").drop("segmentsArrivalTimeEpochSeconds")
```

Noise and Outlier Removal:

Outliers and noise can distort the results of statistical analyses and machine learning models. By identifying and removing outliers, preprocessing ensures that the models are trained on data that accurately represents the underlying patterns, leading to more reliable outcomes. Further, the dataset handles missing values by filling in the totalTravelDistance and segmentsDistance columns with their respective mean values and dropping any remaining rows with missing data. Outliers in the totalFare column are identified and removed based on statistical thresholds, specifically those values that fall outside three standard deviations from the mean. Duplicate rows are also eliminated to ensure data integrity.



```
Number of instances in DataFrame: 8213875
Number of instances after outlier removal in DataFrame: 8141423
Number of duplicate rows: 456004
Number of instances after removing duplicates: 7685419
```



```python
# Drop rows with missing values and fill remaining with mean
mean_values = df.agg(
    mean(col("totalTravelDistance")).alias("mean_totalTravelDistance"),
    mean(col("segmentsDistance")).alias("mean_segmentsDistance")
).collect()[0]

df = df.fillna({
    'totalTravelDistance': mean_values['mean_totalTravelDistance'],
    'segmentsDistance': mean_values['mean_segmentsDistance'],
}).dropna()

# Outlier detection and removal
stats = df.select(mean("totalFare").alias("mean"), stddev("totalFare").alias("stddev")).collect()[0]
mean_fare, stddev_fare = stats["mean"], stats["stddev"]
df = df.filter(~((col("totalFare") > mean_fare + 3 * stddev_fare) | (col("totalFare") < mean_fare - 3 * stddev_fare)))

# Count the number of instances in the DataFrame
num_instances = df.count()
print(f"Number of instances after outlier removal in DataFrame: {num_instances}")
```

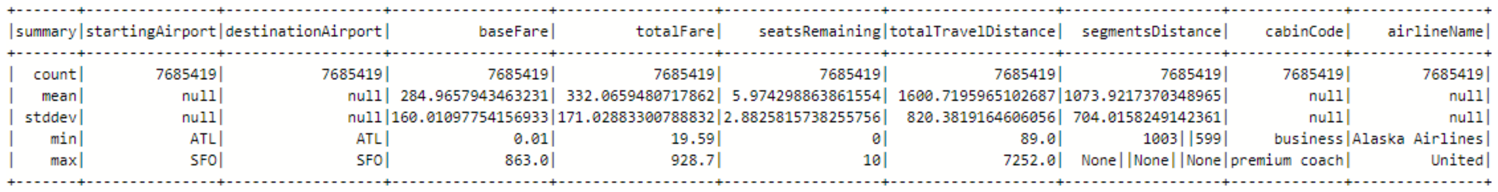
Feature Engineering:

New features like departure_time and arrival_time are derived from Unix timestamp columns. The dataset is prepared for modeling by creating feature vectors using VectorAssembler.

5.Exploratory Data Analysis (EDA)

Descriptive Statistics:

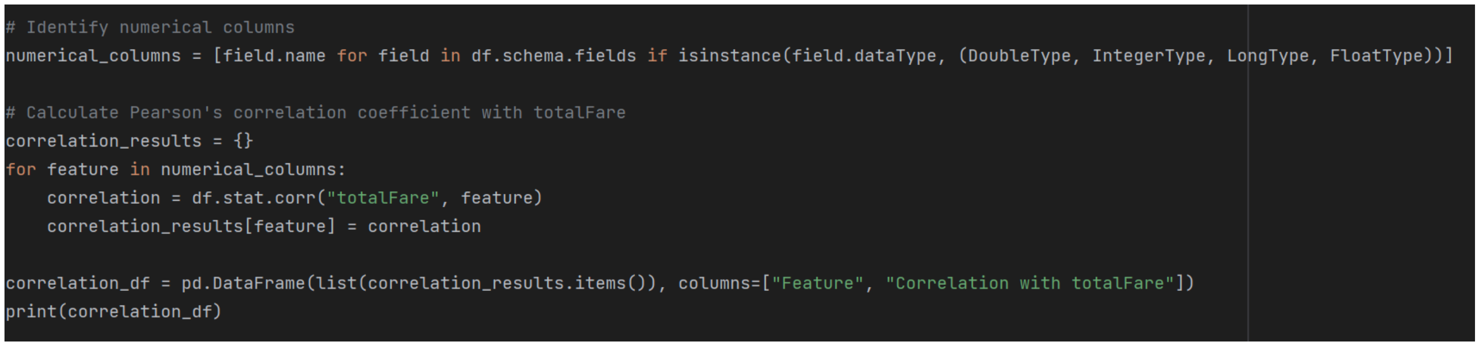
Summary Statistics: The .describe() function is used to compute summary statistics for numerical columns. This includes measures such as mean, standard deviation, minimum, and maximum values, providing a high-level overview of the data distribution and variability.



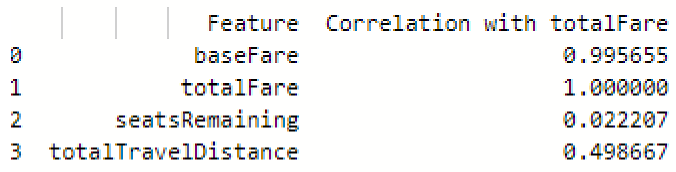
```
+-------+---------------+------------------+------------------+-----------------+------------------+-------------------+------------------+-------------+---------------+
|summary|startingAirport|destinationAirport|          baseFare|        totalFare|    seatsRemaining|totalTravelDistance|  segmentsDistance|    cabinCode|    airlineName|
+-------+---------------+------------------+------------------+-----------------+------------------+-------------------+------------------+-------------+---------------+
|  count|        7685419|           7685419|           7685419|          7685419|           7685419|            7685419|           7685419|      7685419|        7685419|
|   mean|           null|              null| 284.9657943463231| 332.0659480717862| 5.974298863861554| 1600.7195965102687|1073.9217370348965|         null|           null|
| stddev|           null|              null|160.01097754156933|171.02883300788832|2.8825815738255756|  820.3819164606056| 704.0158249142361|         null|           null|
|    min|            ATL|               ATL|              0.01|            19.59|                 0|               89.0|         1003||599|     business|Alaska Airlines|
|    max|            SFO|               SFO|             863.0|            928.7|                10|             7252.0|  None||None||None|premium coach|         United|
+-------+---------------+------------------+------------------+-----------------+------------------+-------------------+------------------+-------------+---------------+
```

Correlation Analysis:

Pearson Correlation Coefficient: The correlation between the totalFare and other numerical features is calculated using Pearson's correlation coefficient. This helps in understanding the strength and direction of linear relationships between totalFare and features such as baseFare, seatsRemaining, and totalTravelDistance.



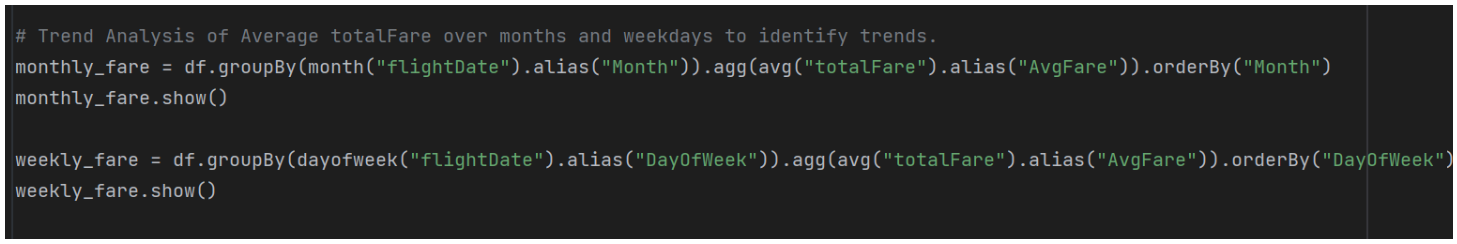
```
# Identify numerical columns
numerical_columns = [field.name for field in df.schema.fields if isinstance(field.dataType, (DoubleType, IntegerType, LongType, FloatType))]

# Calculate Pearson's correlation coefficient with totalFare
correlation_results = {}
for feature in numerical_columns:
    correlation = df.stat.corr("totalFare", feature)
    correlation_results[feature] = correlation

correlation_df = pd.DataFrame(list(correlation_results.items()), columns=["Feature", "Correlation with totalFare"])
print(correlation_df)
```



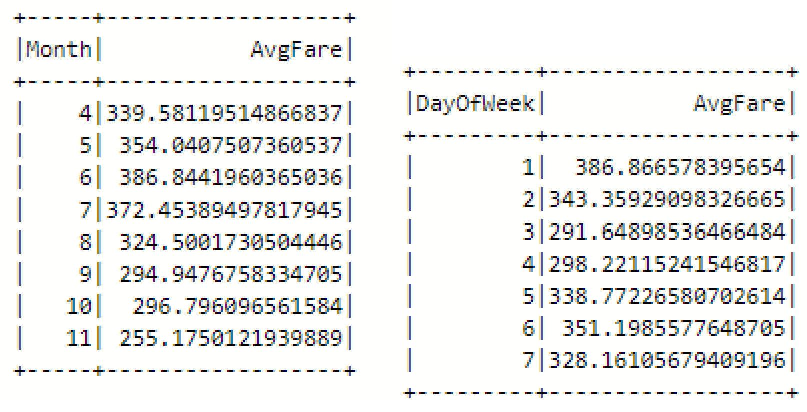
```
             Feature  Correlation with totalFare
0           baseFare                    0.995655
1          totalFare                    1.000000
2     seatsRemaining                    0.022207
3  totalTravelDistance                  0.498667
```

Trend Analysis:

Monthly Average Fare: The average totalFare is computed for each month using the flightDate column. This analysis helps identify seasonal trends or variations in flight prices throughout the year.
Weekly Average Fare: The average totalFare is also computed for each day of the week. This reveals trends related to specific days, which might affect fare pricing patterns.



```
# Trend Analysis of Average totalFare over months and weekdays to identify trends.
monthly_fare = df.groupBy(month("flightDate").alias("Month")).agg(avg("totalFare").alias("AvgFare")).orderBy("Month")
monthly_fare.show()

weekly_fare = df.groupBy(dayofweek("flightDate").alias("DayOfWeek")).agg(avg("totalFare").alias("AvgFare")).orderBy("DayOfWeek")
weekly_fare.show()
```

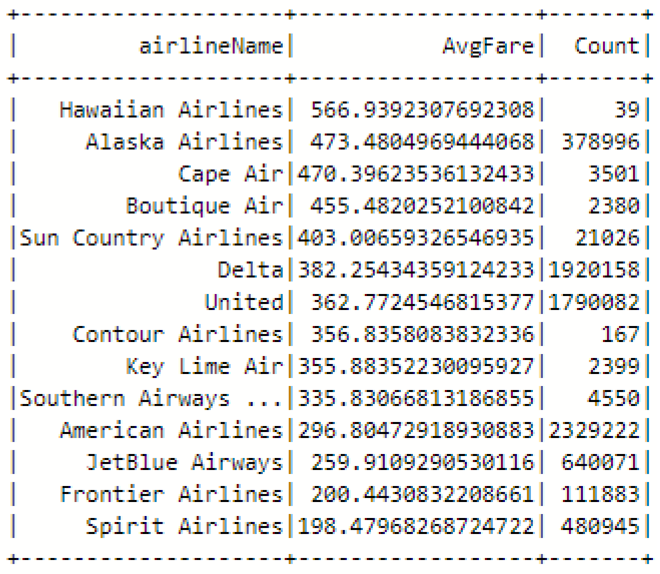


```
+-----+------------------+
|Month|           AvgFare|
+-----+------------------+
|    4|339.58119514866837|
|    5| 354.0407507360537|
|    6| 386.8441960365036|
|    7|372.45389497817945|
|    8| 324.5001730504446|
|    9| 294.9476758334705|
|   10|  296.796096561584|
|   11| 255.1750121939889|
+-----+------------------+
```

```
+---------+------------------+
|DayOfWeek|           AvgFare|
+---------+------------------+
|        1|  386.866578395654|
|        2|343.35929098326665|
|        3|291.64898536466484|
|        4|298.22115241546817|
|        5|338.77226580702614|
|        6| 351.1985577648705|
|        7|328.16105679409196|
+---------+------------------+
```

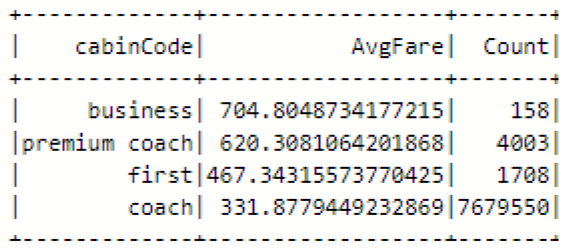
Fare Analysis by Categories:

Airline Fare Distribution: The average totalFare is calculated for each airline (airlineName). This analysis provides insights into fare differences between airlines and can highlight which airlines tend to have higher or lower average fares.



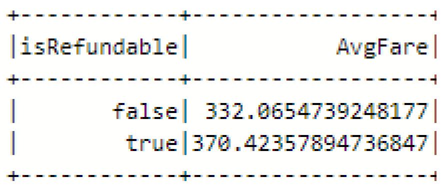
```
+--------------------+-----------------+-------+
|         airlineName|           AvgFare|  Count|
+--------------------+-----------------+-------+
|   Hawaiian Airlines| 566.9392307692308|     39|
|     Alaska Airlines| 473.4804969444068| 378996|
|            Cape Air|470.39623536132433|   3501|
|        Boutique Air| 455.482025210084 2|   2380|
|Sun Country Airlines|403.00659326546935|  21026|
|               Delta|382.25434359124233|1920158|
|              United| 362.7724546815377|1790082|
|    Contour Airlines| 356.8358083832336|    167|
|        Key Lime Air|355.88352230095927|   2399|
|Southern Airways ...|335.83066813186855|   4550|
|   American Airlines|296.80472918930883|2329222|
|     JetBlue Airways| 259.9109290530116| 640071|
|   Frontier Airlines| 200.4430832208661| 111883|
|     Spirit Airlines|198.47968268724722| 480945|
+--------------------+-----------------+-------+
```

Cabin Class Fare Distribution: The average totalFare is computed for each cabin class (cabinCode). This helps in understanding how fares vary between different cabin classes, such as economy, business, or first class.



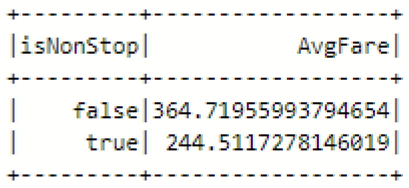
```
+-------------+------------------+-------+
|    cabinCode|           AvgFare|  Count|
+-------------+------------------+-------+
|     business| 704.8048734177215|    158|
|premium coach| 620.3081064201868|   4003|
|        first|467.34315573770425|   1708|
|        coach| 331.8779449232869|7679550|
+-------------+------------------+-------+
```

Refundability Analysis: The average totalFare is compared between refundable and non-refundable tickets. This analysis shows how the ability to refund a ticket influences its price.

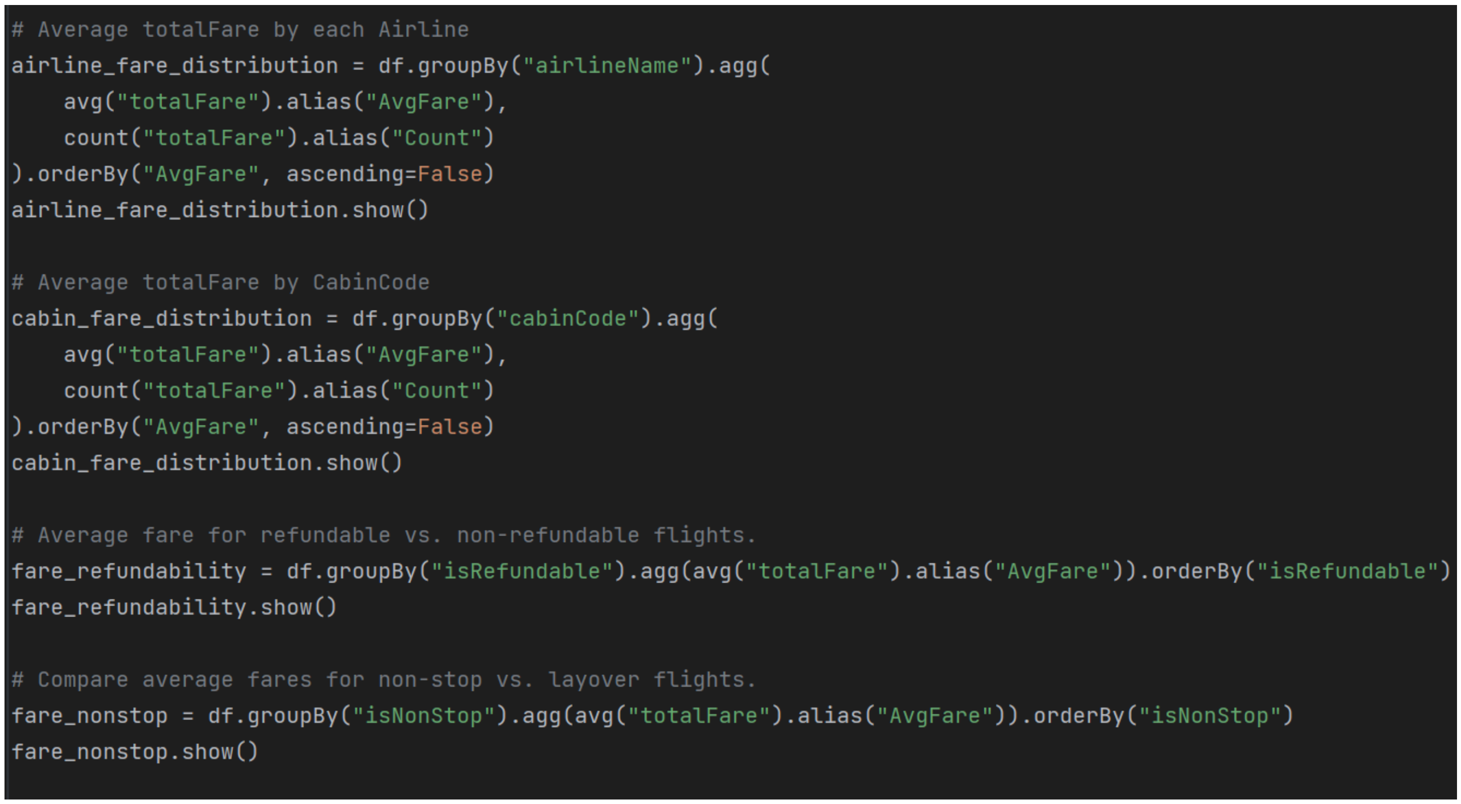


```
+------------+------------------+
|isRefundable|           AvgFare|
+------------+------------------+
|       false| 332.0654739248177|
|        true|370.42357894736847|
+------------+------------------+
```

Non-Stop vs. Layover Analysis: The average totalFare is analyzed based on whether the flight is non-stop or includes layovers. This helps determine if non-stop flights tend to be more expensive than those with layovers.



```
+---------+------------------+
|isNonStop|           AvgFare|
+---------+------------------+
|    false|364.71955993794654|
|     true| 244.5117278146019|
+---------+------------------+
```

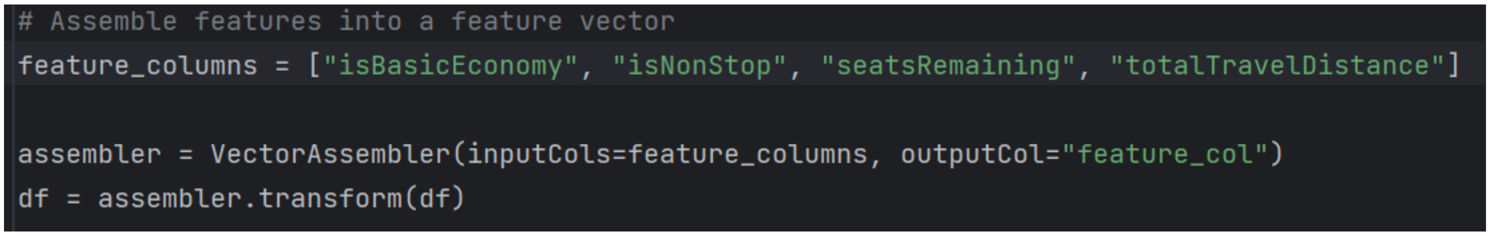


```python
# Average totalFare by each Airline
airline_fare_distribution = df.groupBy("airlineName").agg(
    avg("totalFare").alias("AvgFare"),
    count("totalFare").alias("Count")
).orderBy("AvgFare", ascending=False)
airline_fare_distribution.show()

# Average totalFare by CabinCode
cabin_fare_distribution = df.groupBy("cabinCode").agg(
    avg("totalFare").alias("AvgFare"),
    count("totalFare").alias("Count")
).orderBy("AvgFare", ascending=False)
cabin_fare_distribution.show()

# Average fare for refundable vs. non-refundable flights.
fare_refundability = df.groupBy("isRefundable").agg(avg("totalFare").alias("AvgFare")).orderBy("isRefundable")
fare_refundability.show()

# Compare average fares for non-stop vs. layover flights.
fare_nonstop = df.groupBy("isNonStop").agg(avg("totalFare").alias("AvgFare")).orderBy("isNonStop")
fare_nonstop.show()
```

6.Model Building and evaluation

The provided script outlines a typical machine learning workflow for predicting totalFare based on various flight attributes. The process involves data preparation, model training, evaluation, and feature importance analysis.

Data Preparation:
Feature Selection: Essential flight characteristics like isBasicEconomy, isRefundable, isNonStop, seatsRemaining, and totalTravelDistance are chosen as potential predictors of totalFare.
Feature Engineering: These selected features are combined into a single numerical vector using VectorAssembler for efficient model input.


```
# Assemble features into a feature vector
feature_columns = ["isBasicEconomy", "isNonStop", "seatsRemaining", "totalTravelDistance"]

assembler = VectorAssembler(inputCols=feature_columns, outputCol="feature_col")
df = assembler.transform(df)
```

Data Splitting:

The dataset is divided into training (80%) and testing (20%) sets. The training set is used to build the model, while the testing set evaluates its performance on unseen data.

Model Training:

Random Forest Regressor: This ensemble learning algorithm is employed to predict the continuous totalFare value.

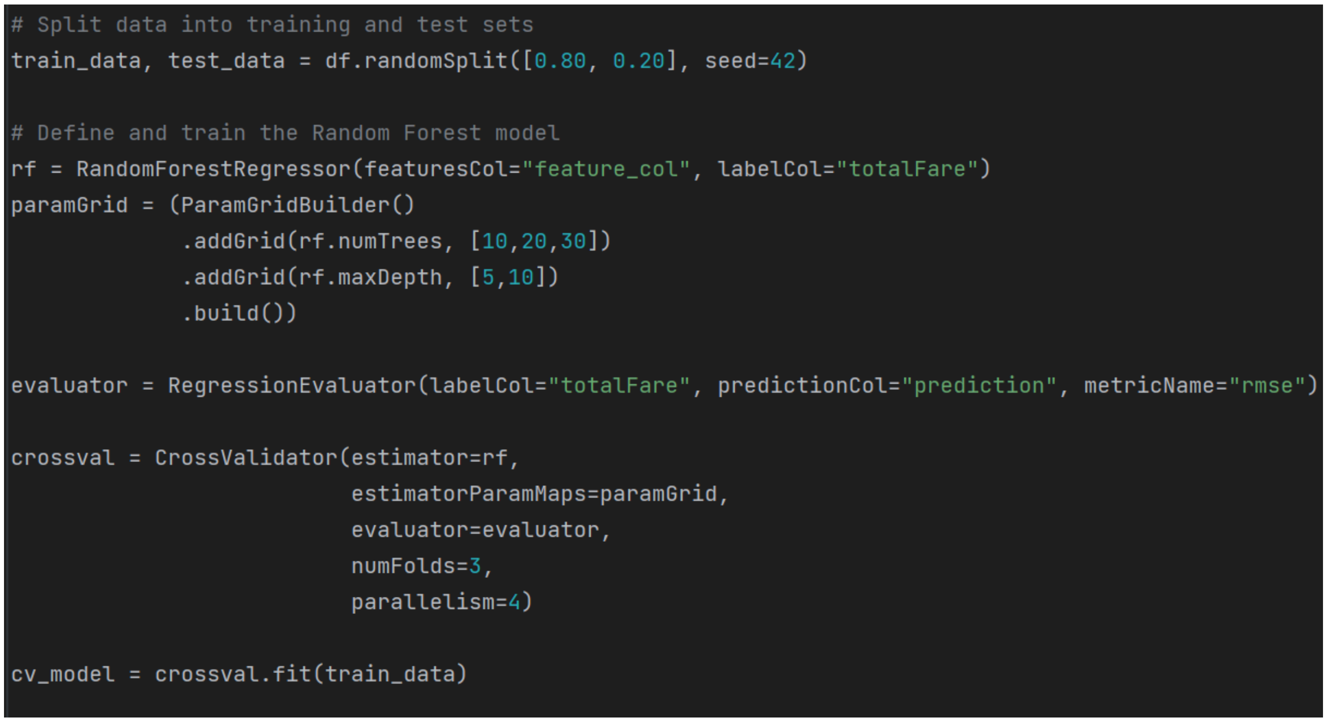
Why Random Forest Regressor?

Robustness to Overfitting: Random Forests aggregate multiple decision trees, reducing the risk of overfitting compared to single decision trees. This ensemble method enhances model generalization to unseen data.

Feature Importance: RandomForestRegressor provides a measure of feature importance, helping in understanding which features contribute most to the prediction. This insight aids in feature selection and model interpretability.

Robust to Missing Values and Outliers: Can handle missing values and outliers more effectively than many other regression algorithms. This robustness makes it suitable for real-world datasets that often have imperfect data.

Hyperparameter Tuning: A grid search approach is used to optimize model performance by experimenting with different combinations of numTrees and maxDepth parameters. CrossValidator is used to assess model performance across multiple folds for robust evaluation.



```
# Split data into training and test sets
train_data, test_data = df.randomSplit([0.80, 0.20], seed=42)

# Define and train the Random Forest model
rf = RandomForestRegressor(featuresCol="feature_col", labelCol="totalFare")
paramGrid = (ParamGridBuilder()
             .addGrid(rf.numTrees, [10,20,30])
             .addGrid(rf.maxDepth, [5,10])
             .build())

evaluator = RegressionEvaluator(labelCol="totalFare", predictionCol="prediction", metricName="rmse")

crossval = CrossValidator(estimator=rf,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=3,
                          parallelism=4)

cv_model = crossval.fit(train_data)
```

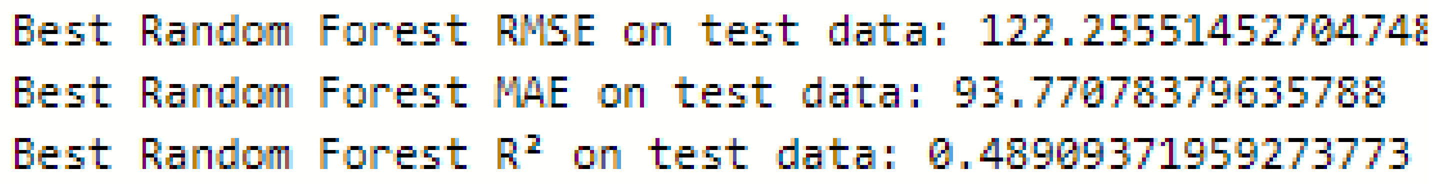
Performance Metrics:

The model's predictive power is assessed using RMSE, MAE, and R-squared.

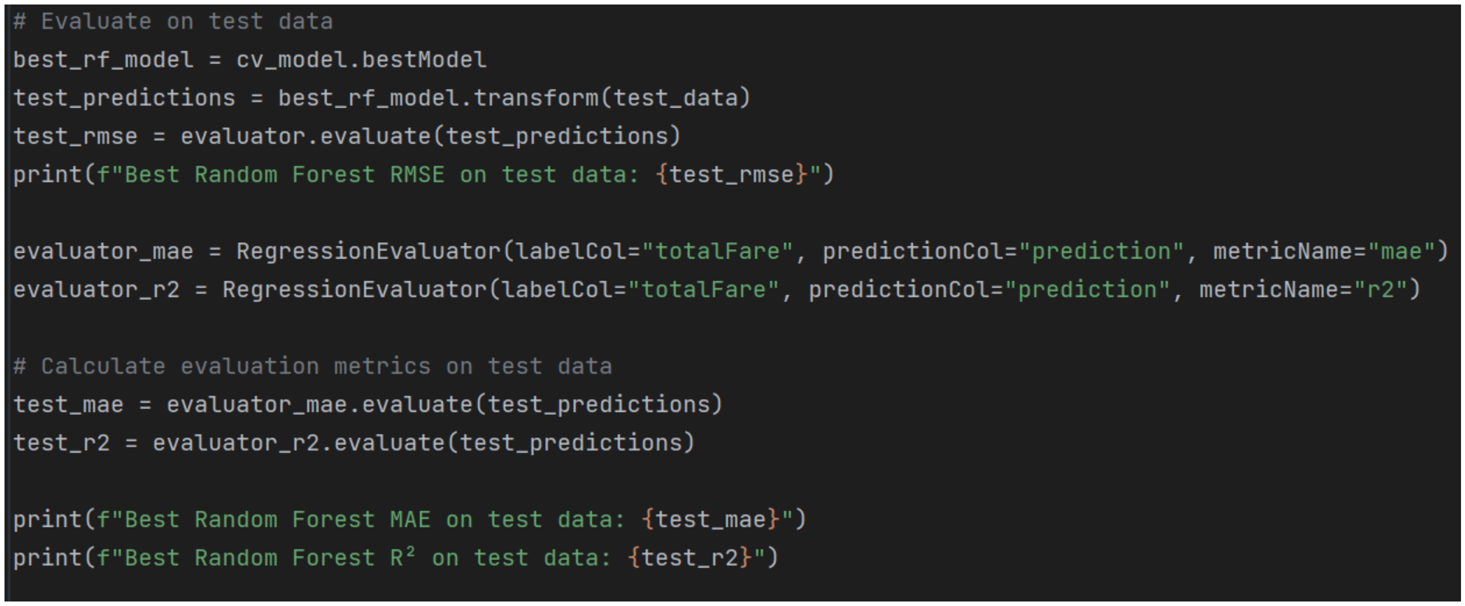
RMSE: Measures the average prediction error in the original units of the target variable.

MAE: Provides a more intuitive understanding of average error magnitude, less sensitive to outliers.

R-squared: Indicates the proportion of variance in totalFare explained by the model.



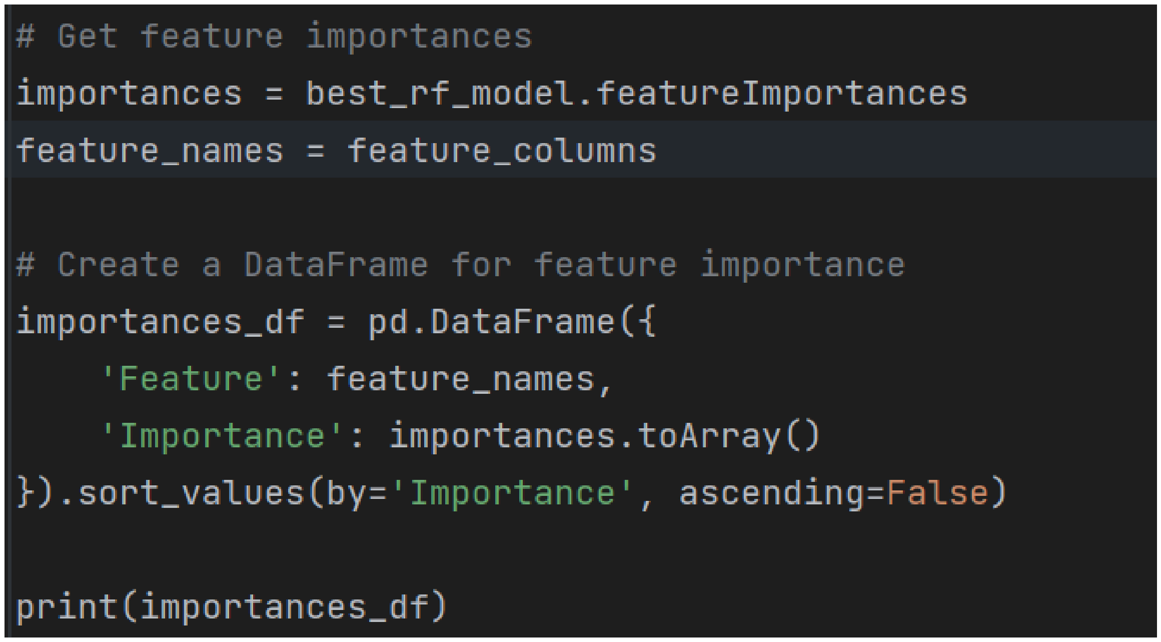
```
Best Random Forest RMSE on test data: 122.2555145270474
Best Random Forest MAE on test data: 93.77078379635788
Best Random Forest R² on test data: 0.4890937195927377
```



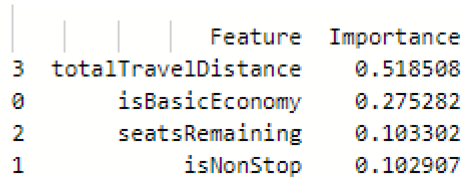
```python
# Evaluate on test data
best_rf_model = cv_model.bestModel
test_predictions = best_rf_model.transform(test_data)
test_rmse = evaluator.evaluate(test_predictions)
print(f"Best Random Forest RMSE on test data: {test_rmse}")

evaluator_mae = RegressionEvaluator(labelCol="totalFare", predictionCol="prediction", metricName="mae")
evaluator_r2 = RegressionEvaluator(labelCol="totalFare", predictionCol="prediction", metricName="r2")

# Calculate evaluation metrics on test data
test_mae = evaluator_mae.evaluate(test_predictions)
test_r2 = evaluator_r2.evaluate(test_predictions)

print(f"Best Random Forest MAE on test data: {test_mae}")
print(f"Best Random Forest R² on test data: {test_r2}")
```

Feature Importance:

The relative contribution of each feature to the model's predictions is determined using featureImportances. This helps identify the most influential factors affecting totalFare.



```python
# Get feature importances
importances = best_rf_model.featureImportances
feature_names = feature_columns

# Create a DataFrame for feature importance
importances_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances.toArray()
}).sort_values(by='Importance', ascending=False)

print(importances_df)
```

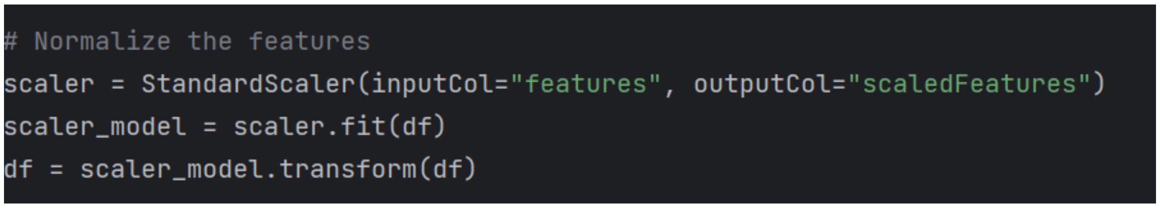


```
              Feature  Importance
3  totalTravelDistance    0.518508
0      isBasicEconomy    0.275282
2      seatsRemaining    0.103302
1           isNonStop    0.102907
```

7.Cluster Analysis

One-Hot Encoding is a technique used to convert categorical variables into a numerical format that can be easily used by machine learning algorithms. This process transforms categorical data into a binary matrix where each category is represented by a unique binary vector.

Feature Normalization: The assembled feature vector is normalized using StandardScaler to standardize the feature values. This step helps ensure that all features contribute equally to the clustering process.

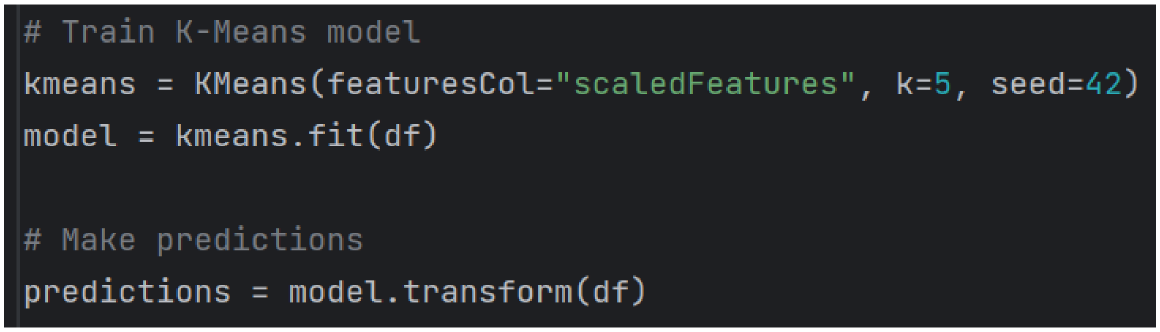


```
# Normalize the features
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(df)
df = scaler_model.transform(df)
```

K-Means Clustering:

Model Training: A K-Means clustering model is trained with k=5 clusters. K-Means is a popular clustering algorithm that partitions the data into a specified number of clusters based on feature similarity.

Prediction: The trained K-Means model is used to predict cluster assignments for each data point.

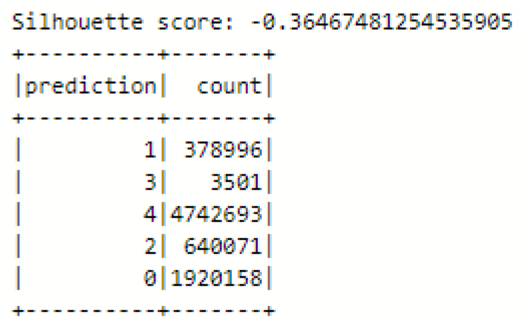


```
# Train K-Means model
kmeans = KMeans(featuresCol="scaledFeatures", k=5, seed=42)
model = kmeans.fit(df)

# Make predictions
predictions = model.transform(df)
```

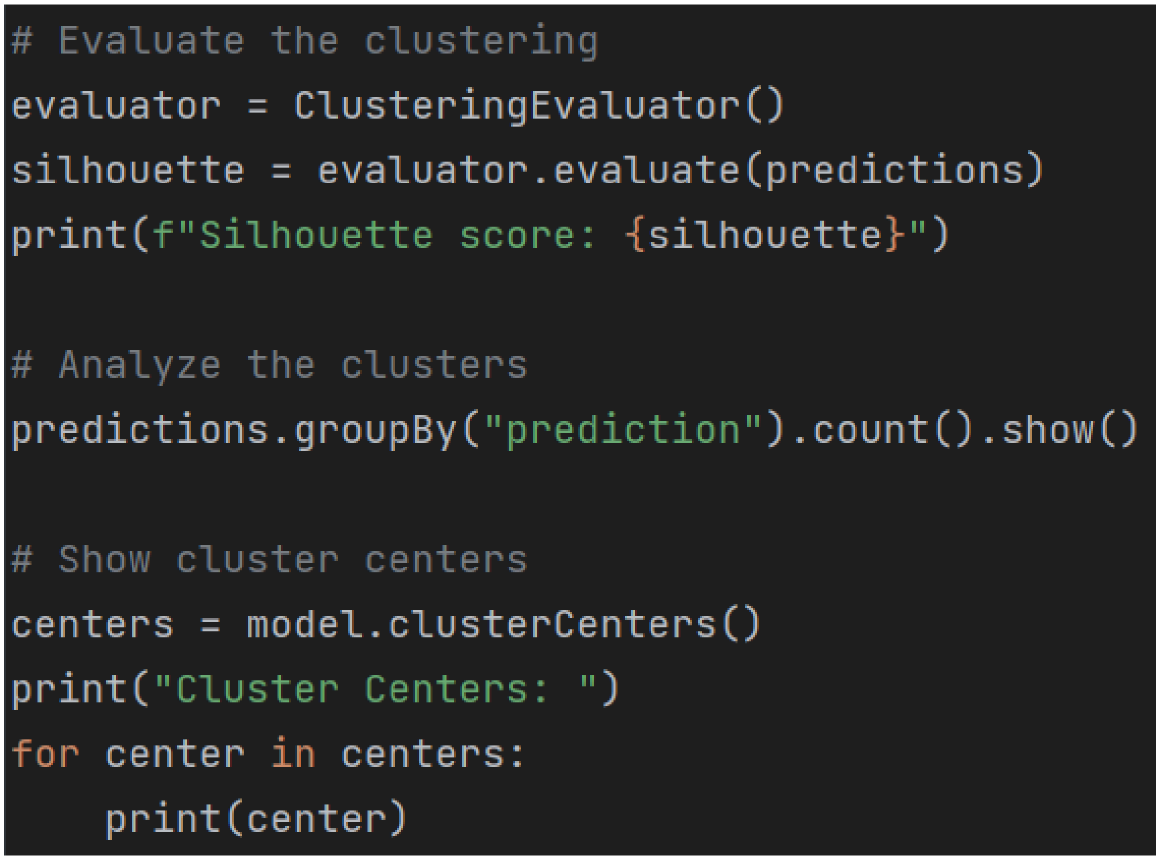
Evaluation and Analysis:

Silhouette Score: The clustering quality is evaluated using the Silhouette score, which measures how similar each point is to its own cluster compared to other clusters. A higher Silhouette score indicates better-defined clusters. The Silhouette score in this case suggests that the clusters may not be well-separated.

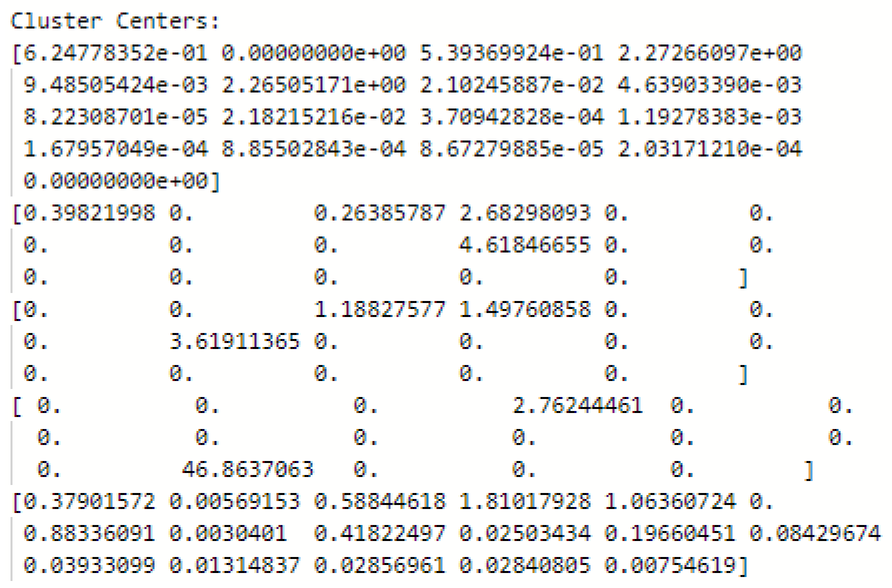
Cluster Analysis: The distribution of data points across clusters is analyzed, and the cluster centers are examined to understand the characteristics of each cluster. Cluster sizes vary significantly, with Cluster 4 being the largest and Cluster 3 the smallest.



```
Silhouette score: -0.36467481254535905
+----------+-------+
|prediction|  count|
+----------+-------+
|         1| 378996|
|         3|   3501|
|         4|4742693|
|         2| 640071|
|         0|1920158|
+----------+-------+
```



```python
# Evaluate the clustering
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print(f"Silhouette score: {silhouette}")

# Analyze the clusters
predictions.groupBy("prediction").count().show()

# Show cluster centers
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```



```
Cluster Centers:
[6.24778352e-01 0.00000000e+00 5.39369924e-01 2.27266097e+00
 9.48505424e-03 2.26505171e+00 2.10245887e-02 4.63903390e-03
 8.22308701e-05 2.18215216e-02 3.70942828e-04 1.19278383e-03
 1.67957049e-04 8.85502843e-04 8.67279885e-05 2.03171210e-04
 0.00000000e+00]
[0.39821998 0.         0.26385787 2.68298093 0.         0.
 0.         0.         0.         4.61846655 0.         0.
 0.         0.         0.         0.         0.        ]
[0.         0.         1.18827577 1.49760858 0.         0.
 0.         3.61911365 0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
[ 0.          0.          0.          2.76244461  0.          0.
  0.          0.          0.          0.          0.          0.
  0.         46.8637063   0.          0.          0.        ]
[0.37901572 0.00569153 0.58844618 1.81017928 1.06360724 0.
 0.88336091 0.0030401  0.41822497 0.02503434 0.19660451 0.08429674
 0.03933099 0.01314837 0.02856961 0.02840805 0.00754619]
```

8.Conclusion

The results show a comprehensive analysis of flight data. The dataset has been preprocessed, including categorical encoding and feature scaling. The K-Means clustering yielded five clusters with a silhouette score of -0.365, indicating poor cluster cohesion and separation. The cluster centers reflect varying patterns in the scaled features. In the Random Forest regression model, the best RMSE is 122.26, MAE is 93.77, and R^2 is 0.49, suggesting moderate prediction accuracy. Feature importance highlights totalTravelDistance and isBasicEconomy as significant. Clustering results are diverse, with imbalances in cluster sizes, indicating challenges in clustering quality.

9.Future work

☐ Advanced Feature Engineering: Explore additional features such as flight duration and fare basis codes to enhance model performance.

☐ Model Selection: Experiment with alternative machine learning models, such as Gradient Boosting or Neural Networks, for better prediction accuracy.

☐ Temporal Analysis: Investigate seasonal and time-based trends more deeply to understand how flight prices fluctuate over different periods.

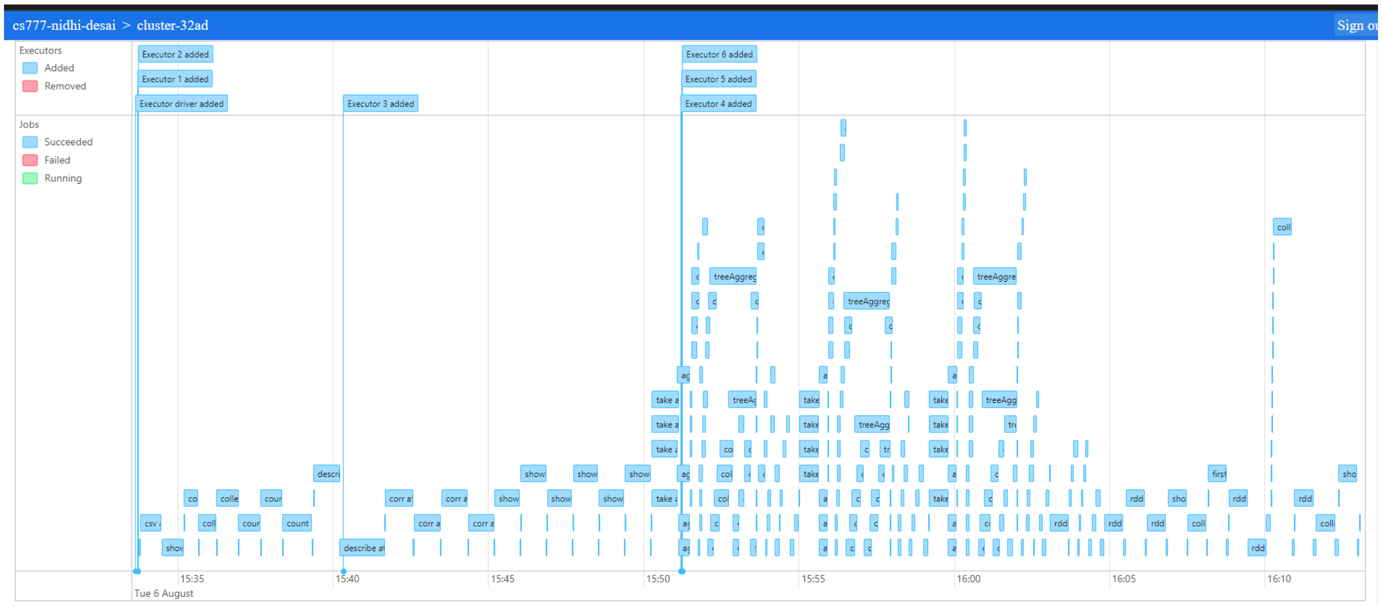
10.Refrences

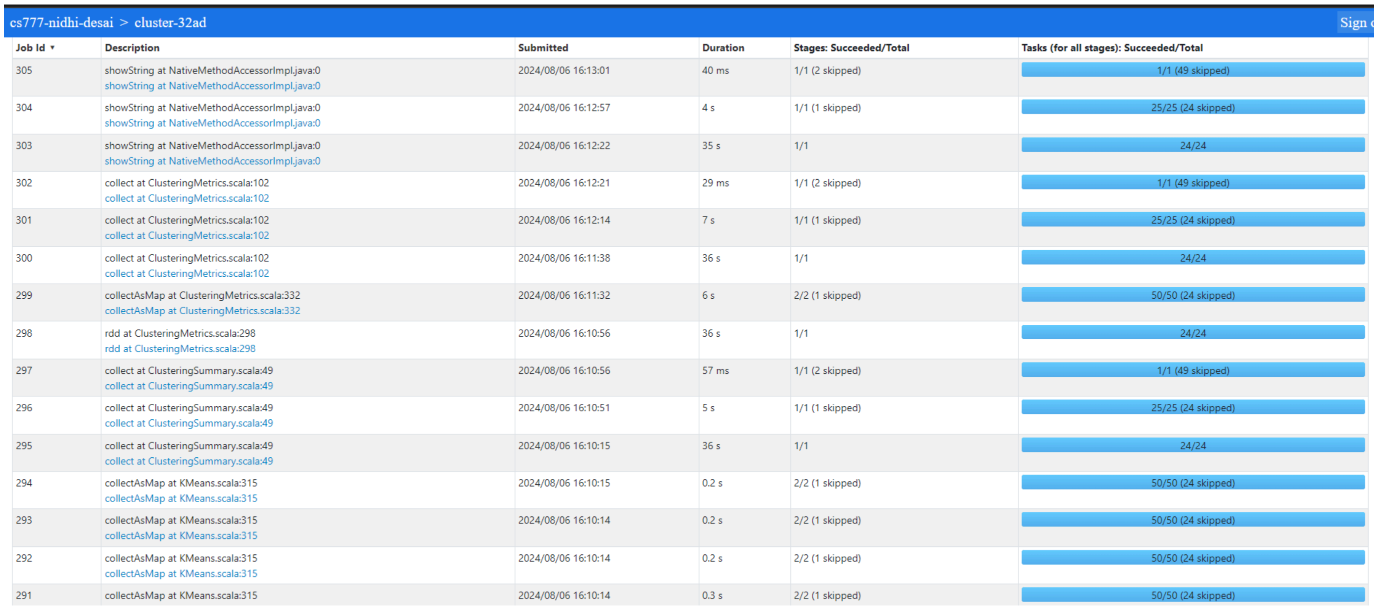
• Apache Spark Documentation URL: https://spark.apache.org/docs/

- Google Cloud Platform Documentation:URL: https://cloud.google.com/docs
- Stack Overflow URL: https://stackoverflow.com/
- Kaggle Dataset URL: https://www.kaggle.com/datasets/dilwong/flightprices

GITHUB REPOSITORY: https://github.com/nidhi4848/CS777_Project

SPARK HISTORY SERVER




cs777-nidhi-desai > cluster-32ad

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
305	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2024/08/06 16:13:01	40 ms	1/1 (2 skipped)	1/1 (49 skipped)
304	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2024/08/06 16:12:57	4 s	1/1 (1 skipped)	25/25 (24 skipped)
303	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2024/08/06 16:12:22	35 s	1/1	24/24
302	collect at ClusteringMetrics.scala:102 collect at ClusteringMetrics.scala:102	2024/08/06 16:12:21	29 ms	1/1 (2 skipped)	1/1 (49 skipped)
301	collect at ClusteringMetrics.scala:102 collect at ClusteringMetrics.scala:102	2024/08/06 16:12:14	7 s	1/1 (1 skipped)	25/25 (24 skipped)
300	collect at ClusteringMetrics.scala:102 collect at ClusteringMetrics.scala:102	2024/08/06 16:11:38	36 s	1/1	24/24
299	collectAsMap at ClusteringMetrics.scala:332 collectAsMap at ClusteringMetrics.scala:332	2024/08/06 16:11:32	6 s	2/2 (1 skipped)	50/50 (24 skipped)
298	rdd at ClusteringMetrics.scala:298 rdd at ClusteringMetrics.scala:298	2024/08/06 16:10:56	36 s	1/1	24/24
297	collect at ClusteringSummary.scala:49 collect at ClusteringSummary.scala:49	2024/08/06 16:10:56	57 ms	1/1 (2 skipped)	1/1 (49 skipped)
296	collect at ClusteringSummary.scala:49 collect at ClusteringSummary.scala:49	2024/08/06 16:10:51	5 s	1/1 (1 skipped)	25/25 (24 skipped)
295	collect at ClusteringSummary.scala:49 collect at ClusteringSummary.scala:49	2024/08/06 16:10:15	36 s	1/1	24/24
294	collectAsMap at KMeans.scala:315 collectAsMap at KMeans.scala:315	2024/08/06 16:10:15	0.2 s	2/2 (1 skipped)	50/50 (24 skipped)
293	collectAsMap at KMeans.scala:315 collectAsMap at KMeans.scala:315	2024/08/06 16:10:14	0.2 s	2/2 (1 skipped)	50/50 (24 skipped)
292	collectAsMap at KMeans.scala:315 collectAsMap at KMeans.scala:315	2024/08/06 16:10:14	0.2 s	2/2 (1 skipped)	50/50 (24 skipped)
291	collectAsMap at KMeans.scala:315	2024/08/06 16:10:14	0.3 s	2/2 (1 skipped)	50/50 (24 skipped)