BIL 401 – INTRODUCTION TO BIG DATA

FINAL REPORT

Sena Ezgi Anadollu 201401017
Beyza Altanlar 151101040

Flight Price Prediction Using Apache Spark and MLlib Framework

1st Sena Ezgi Anadollu
Department of Artificial Intelligence Engineering
TOBB University of Economics and Technology
Ankara,Turkey
sena@anadollu.com

2nd Beyza Altanlar
Department of Computer Engineering
TOBB University of Economics and Technology
Ankara,Turkey
beyzaaltanlar@gmail.com

***Abstract*—Airlines dynamically adjust ticket prices depending on various factors and use special algorithms to determine the optimal pricing policy.**

This paper introduces methods for the analysis of airline flight data, focusing on flight price prediction within the Apache Spark environment. The factors influencing ticket pricing are considered using PySpark. Among the factors focused on while performing data analysis are airline type, days left for departure, departure time and arrival time, source and destination city, and class type. The present study on flight price prediction applies linear regression and decision tree regression algorithms to perform more advanced analysis with the help of MLlib in PySpark. A novel dataset consisting of 300153 distinct flight booking options collected for 50 days, from February 11th to March 31st, 2022 is used.

***Index Terms*—Feature Analysis, PySpark, Apache Spark, Flight Price Prediction, Big Data**

I. Introduction

Apache Spark is an open-source computing framework designed for big data processing and analytics. It provides high-level abstractions and tools for efficiently processing large datasets in a distributed and parallel fashion.

PySpark is the Python API for Apache Spark allowing developers to write Spark applications using Python.

MLlib (Machine Learning Library) is a library in Apache Spark that is specifically built for machine learning tasks and offers a wide range of algorithms and tools for this purpose.

Linear regression is a statistical method that enables us to model the relationship between a dependent variable and one or more independent variables.

Decision trees, operate by recursively partitioning the data into subsets, with each partition maximally homogeneous with respect to the target variable.

In this study, we examine flight data to obtain insights into the factors influencing ticket pricing and acquire vital information about airline flights in the Apache Spark environment. In addition, we aim to produce accurate predictions and uncover the relationships between different variables by employing machine learning models.

This paper's organization is as follows: Section II presents a literature review containing studies related to flight price prediction. Our methods assembled for the objective are presented in Section III. Section IV provides the analytics of the data and the results of utilized ML algorithms before the discussion part in Section V.

II. Related Work

Due to the development of flight ticket price policies and global interest, there has been a surge in research and projects related to flight ticket price prediction. This section is dedicated to those projects that are similar to our own.

[1]This study is similar to our project in aspects of utilizing Spark and machine learning techniques to forecast flight ticket prices. It involved a substantial dataset of roughly 20 million records (4.68 gigabytes). The researchers employed four regression machine learning algorithms - Random Forest, Gradient Boost Tree, Decision Tree, and Factorization Machines - in their analysis. To assess performance and generalization, they utilized Cross Validator and Training Validator functions. Based on the findings, the Gradient Boost Tree emerged as the most effective algorithm with the highest accuracy.

[2]In a study on predicting airline fares using machine learning, a dataset of 1814 Aegean Airlines flight data was utilized. The study employed various methods such as Multilayer Perceptron, Generalized Regression Neural Network, Extreme Learning Machine, Random Forest Regression Tree, Regression Tree, Bagging Regression Tree, Regression SVM, and Linear Regression, resulting in different outcomes. The Bagging Regression Tree method was found to yield the best results.

[3]In the analysis of flights between Delhi and Mumbai, K-nearest neighbors, linear regression, support vector machine, Multilayer Perceptron, Gradient Boosting, and Random Forest Algorithms were applied to a dataset. The Decision Tree model produced the most favorable results.

[4]The study investigated six targets in four different ways, utilizing eight machine learning models, six deep learning models, and two quantum machine learning models. The results showed that at least three models could achieve an accuracy percentage of 89% to 99%, with the highest success rate achieved through QML methods. The study also found that QML methods can provide effective solutions for airfare estimation. Comparing ML and DL, QML methods were found to yield the best results.

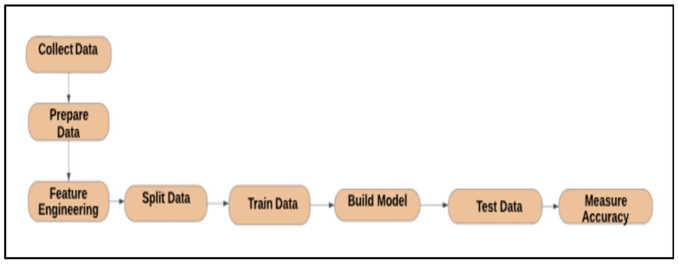


Fig. 1. Workflow Architecture Diagram

[3]Various machine learning algorithms have been assessed for their effectiveness in predicting flight prices. These include Linear Regression, Decision Tree, Random Forest, K-nearest neighbors, Multilayer Perceptron, Support Vector Machine (SVM) and Gradient Boosting. Among these, Random Forest and Multilayer Perceptron are the most successful. In an effort to enhance prediction accuracy, a "stacked" prediction model has been introduced, which combines the outcomes of Random Forest and Multilayer Perceptron by assigning weights. Although the "stacked" prediction model displays encouraging results, it implies that future research can further optimize the model by considering additional features such as the number of seats.

Although we found that the aforementioned projects employed various programming languages and datasets, they are similar to ours with regard to their machine-learning approaches. Our approach is unique in that it is designed to run on the Apache Spark environment.

III. Proposed Implementation

The data is inspected in some aspects to gain insight into the data. Those aspects are price changes according to airline companies, the effect of last-minute ticket purchases on prices, varying departure and arrival times, price fluctuations based on source and destination cities, and the differences between Economy and Business class ticket prices.

We have determined to use linear regression and decision trees in our model. The rationale for selecting these algorithms is due to their frequent usage in machine learning for continuous and numerical target variables.

In order to build the Machine Learning models, a workflow was established as shown in Figure 4. The first step was to collect the data from an appropriate source. For this study, the data was obtained from the "Ease My Trip" website from Kaggle. Once the data was collected, it underwent processing and preparation for use in the models. This included tasks such as cleaning the data, scaling and normalizing the features, and transforming the data into a format suitable for the model.

The choice between PySpark data frames and RDDs (Resilient Distributed Dataset) is a critical consideration when performing data processing operations. It is widely acknowledged that data frames are a better option due to their inherent advantages over RDDs. When data frames are used, the processing is performed on the data in the JVM (Java Virtual Machine) land, which means that the data does not have to be written back and forth. Therefore, we preferred data frame over RDD because it eliminates the need to move data back and forth, reducing data I/O, and significantly improving processing time.

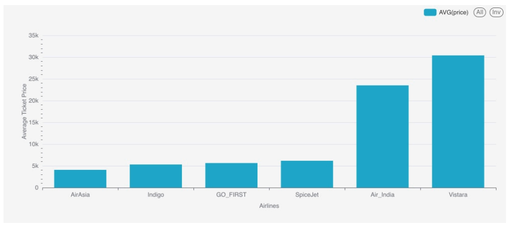


Fig. 2. Average Ticket Price by Airline

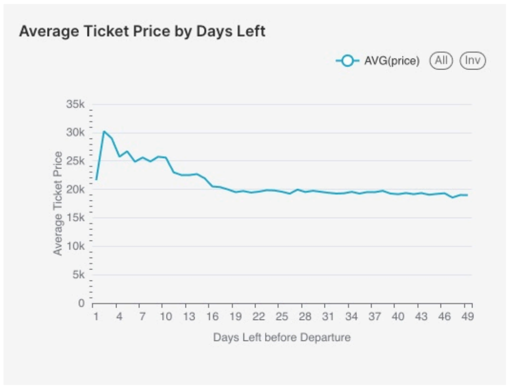


Fig. 3. Average Ticket Price by Days Left

Before building a machine-learning model, it is crucial to divide the data into two parts - a training set and a testing set. The purpose of the training set is to teach the model, and the testing set is used to evaluate the model's performance and ensure that it also performs well on unseen data. After dividing and preparing the data, the next step is to train the machine-learning models.

After the training process, it is crucial to test and validate the model using a designated testing set. We use the Root Mean Square Error (RMSE) and Coefficient of Determination (R^2) metrics to measure the accuracy of the models since they are suitable for the selected regression algorithms. This approach ensures that the model's performance is reliable and consistent, as it allows us to evaluate its predictive capabilities. The use of evaluation metrics also helps to identify areas for potential improvement.

IV. Results

A. Data Analytics

The dataset includes 300153 different flight reservation options. There is a cleaned dataset containing 11 features. These features include airline name, flight code, source city, departure time, number of stops, arrival time, destination city, class, duration, remaining time to departure, and ticket price.

The bar chart in Fig. 2, depicts the average ticket prices for different airlines. There is variation in ticket prices among the

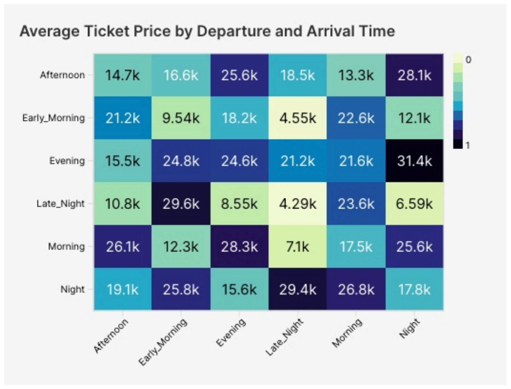


Fig. 4. Average Ticket Price by Departure and Arrival Time

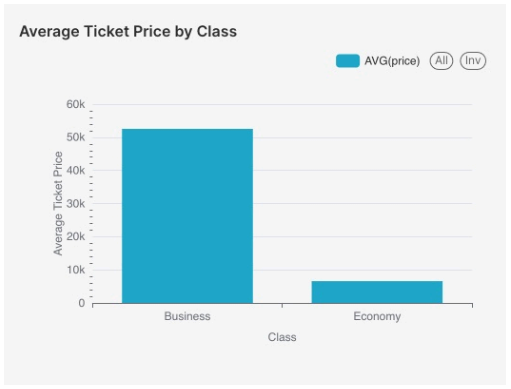


Fig. 5. Average Ticket Price by Class

different airlines and Airline Vista stands out with the highest average ticket price.

The line plot in Fig. 3 illustrates how average ticket prices vary based on the number of days left before departure. Prices generally appear to increase as the departure date approaches, emphasizing the advantage of early booking. However, there is a slight dip in prices for tickets booked a day or two before departure.

The heatmap in Fig. 4 showcases the average ticket prices based on both departure and arrival times. The y-axis represents departure times while the x-axis represents arrival times. Flights departing in the early morning or late evening,

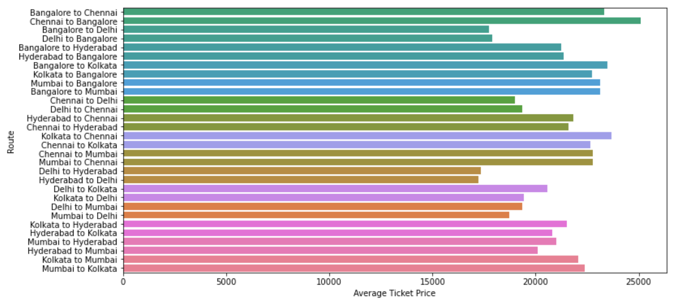


Fig. 6. Average Ticket Price by Source and Destination

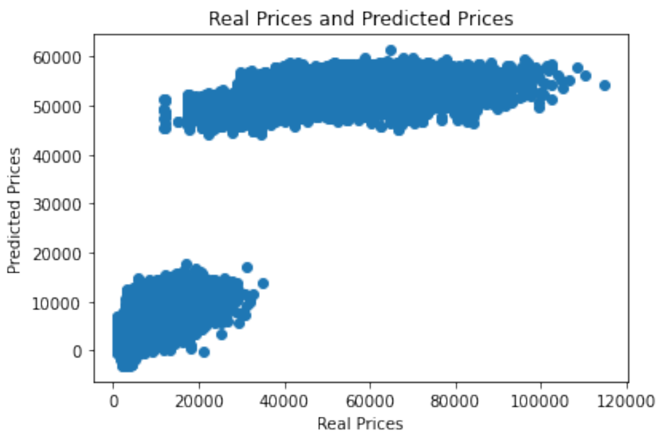


Fig. 7. Prices and Predicted Prices by Logistic Regression

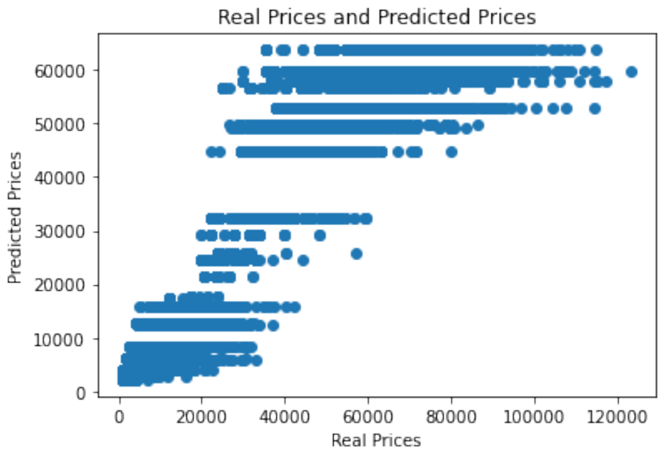


Fig. 8. Prices and Predicted Prices by Decision Tree Regression

especially with midday arrivals, tend to have lower prices. In contrast, peak travel times during the day exhibit higher average prices.

This visualization in Fig. 5 provides insights into the cost differences between the two classes. Business class tickets are considerably more expensive than Economy class tickets.

The average ticket prices for various source and destination city pairs are shown in Fig. 6. Notably, routes involving Delhi consistently have lower prices, whereas those involving Chennai have higher prices than other cities.

B. Evaluation of Machine Learning Algorithms

The Linear Regression model was trained with the divided data set such that 80% of it is used for training and 20% for testing. Fig. 7 shows the relation between the price and the predicted price obtained from the Linear regression model indicating that the model accurately predicts flight prices in some cases, while other estimates deviate significantly. The RMSE calculated from the results is 7277.87 while R^2 is 0.8969.

The other model used for the prediction of flight prices in this research was the decision tree regression model. For this particular use, 70% of the data was allocated as training and 30% as testing. Fig. 8 shows the relation between the price and the predicted price obtained from the Decision Tree regression model indicating that the predicted and actual values are weakly associated with each other. The RMSE obtained from this model is 5299.67 while R^2 is 0.9456.

A lower RMSE indicates better predictive performance. In this case, the Decision Tree Regression Model (5299.67) outperforms the Linear Regression Model (7277.87), suggesting that it provides more accurate predictions. R^2 measures the proportion of variance in the dependent variable that is predictable from the independent variables. A higher R^2 indicates a better fit of the model to the data. Here, the Decision Tree Regression Model (0.9456) has a higher R^2 compared to the Linear Regression Model (0.8969), suggesting that it explains a larger portion of the variance in the target variable.

None of the models gave strong accuracy in price prediction. However, among the two models, decision tree regression has a slightly better performance.

V. Discussion

To achieve better results, alternative regression models such as Random Forest or Gradient Boosting can be considered. These models are expected to improve the prediction accuracy by addressing the limitations of the Linear Regression and Decision Tree Regression models, providing more complex algorithms to capture the complex relationships between the features and target variables.

References

[1] Philip Wong, Phue Thant, Pratiksha Yadav, Ruta Antaliya, and Jongwook Woo. Using spark machine learning models to perform predictive analysis on flight ticket pricing data. *arXiv preprint arXiv:2310.07787*, 2023.

[2] Konstantinos Tziridis, Th Kalampokas, George A Papakostas, and Kostas I Diamantaras. Airfare prices prediction using machine learning techniques. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 1036–1039. IEEE, 2017.

[3] Supriya Rajankar and Neha Sakharkar. A survey on flight pricing prediction using machine learning. *Internatıonal Journal Of Engıneerıng Research & Technology (Ijert)*, 8(6):1281–1284, 2019.

[4] Theofanis Kalampokas, Konstantinos Tziridis, Nikolaos Kalampokas, Alexandros Nikolaou, Eleni Vrochidou, and George A Papakostas. A holistic approach on airfare price prediction using machine learning techniques. *IEEE Access*, 2023.

Sena Ezgi Anadollu - 201401017 Beyza Altanlar - 151101040

```
import pyspark as spark

from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.sql.types import *

spark=SparkSession.builder.appName('FlightPricePrediction').getOrCreat
e()

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
23/12/18 06:39:49 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where
applicable


df_pyspark=spark.read.csv('Clean_Dataset.csv',header=True,inferSchema=
True)
df_pyspark.show()

+-----+---------+-------+-----------+--------------+-----
+------------+----------------+-------+--------+---------+-----+
|index|  airline| flight|source_city|departure_time|stops|
arrival_time|destination_city|  class|duration|days_left|price|
+-----+---------+-------+-----------+--------------+-----
+------------+----------------+-------+--------+---------+-----+
|    0| SpiceJet|SG-8709|      Delhi|       Evening| zero|
Night|          Mumbai|Economy|    2.17|        1| 5953|
|    1| SpiceJet|SG-8157|      Delhi| Early_Morning| zero|
Morning|          Mumbai|Economy|    2.33|        1| 5953|
|    2|  AirAsia| I5-764|      Delhi| Early_Morning| zero|
Early_Morning|          Mumbai|Economy|    2.17|        1| 5956|
|    3|  Vistara| UK-995|      Delhi|       Morning| zero|
Afternoon|          Mumbai|Economy|    2.25|        1| 5955|
|    4|  Vistara| UK-963|      Delhi|       Morning| zero|
Morning|          Mumbai|Economy|    2.33|        1| 5955|
|    5|  Vistara| UK-945|      Delhi|       Morning| zero|
Afternoon|          Mumbai|Economy|    2.33|        1| 5955|
|    6|  Vistara| UK-927|      Delhi|       Morning| zero|
Morning|          Mumbai|Economy|    2.08|        1| 6060|
|    7|  Vistara| UK-951|      Delhi|     Afternoon| zero|
Evening|          Mumbai|Economy|    2.17|        1| 6060|
|    8| GO_FIRST| G8-334|      Delhi| Early_Morning| zero|
Morning|          Mumbai|Economy|    2.17|        1| 5954|
|    9| GO_FIRST| G8-336|      Delhi|     Afternoon| zero|
Evening|          Mumbai|Economy|    2.25|        1| 5954|
|   10| GO_FIRST| G8-392|      Delhi|     Afternoon| zero|
Evening|          Mumbai|Economy|    2.25|        1| 5954|
|   11| GO_FIRST| G8-338|      Delhi|       Morning| zero|
Afternoon|          Mumbai|Economy|    2.33|        1| 5954|
|   12|   Indigo|6E-5001|      Delhi| Early_Morning| zero|
Morning|          Mumbai|Economy|    2.17|        1| 5955|
|   13|   Indigo|6E-6202|      Delhi|       Morning| zero|
Afternoon|          Mumbai|Economy|    2.17|        1| 5955|
|   14|   Indigo| 6E-549|      Delhi|     Afternoon| zero|
Evening|          Mumbai|Economy|    2.25|        1| 5955|
|   15|   Indigo|6E-6278|      Delhi|       Morning| zero|
Morning|          Mumbai|Economy|    2.33|        1| 5955|
|   16|Air_India| AI-887|      Delhi| Early_Morning| zero|
Morning|          Mumbai|Economy|    2.08|        1| 5955|
|   17|Air_India| AI-665|      Delhi| Early_Morning| zero|
Morning|          Mumbai|Economy|    2.17|        1| 5955|
|   18|  AirAsia| I5-747|      Delhi|       Evening|  one|
Early_Morning|          Mumbai|Economy|   12.25|        1| 5949|
|   19|  AirAsia| I5-747|      Delhi|       Evening|  one|
Morning|          Mumbai|Economy|   16.33|        1| 5949|
+-----+---------+-------+-----------+-------------+-----
+-------------+---------------+-------+--------+---------+-----+
only showing top 20 rows
```

```python
# Columns are categorized based on which type of data they have.
continuous_features = ['duration', 'days_Left', 'price']
categorical_features = ['airline', 'flight', 'source_city',
'departure_time', 'stops', 'arrival_time', 'destination_city',
'class']
```

```python
from pyspark.ml.feature import StringIndexer, VectorAssembler,
VectorIndexer
from pyspark.ml.regression import LinearRegression
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
```

```
/Users/ezgi/opt/anaconda3/lib/python3.9/site-packages/scipy/
__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is
required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

```python
df_pyspark.select(continuous_features).describe().toPandas()
```

```
  summary            duration           days_Left               price
0   count              300153              300153              300153
1    mean  12.221020812719066  26.004750910369044  20889.660523133203
2  stddev   7.191997238119004   13.56100368709358  22697.767366074422
```

```
3      min                  0.83                      1           1105
4      max                 49.83                     49         123071
```

```
df_pyspark.printSchema()
```

```
root
 |-- index: integer (nullable = true)
 |-- airline: string (nullable = true)
 |-- flight: string (nullable = true)
 |-- source_city: string (nullable = true)
 |-- departure_time: string (nullable = true)
 |-- stops: string (nullable = true)
 |-- arrival_time: string (nullable = true)
 |-- destination_city: string (nullable = true)
 |-- class: string (nullable = true)
 |-- duration: double (nullable = true)
 |-- days_left: integer (nullable = true)
 |-- price: integer (nullable = true)
```

```
df_pyspark.count()
```

```
300153
```

The dataset includes 300153 flight reservation options

```
df_pyspark.distinct().count()
```

```
300153
```

All values are distinct

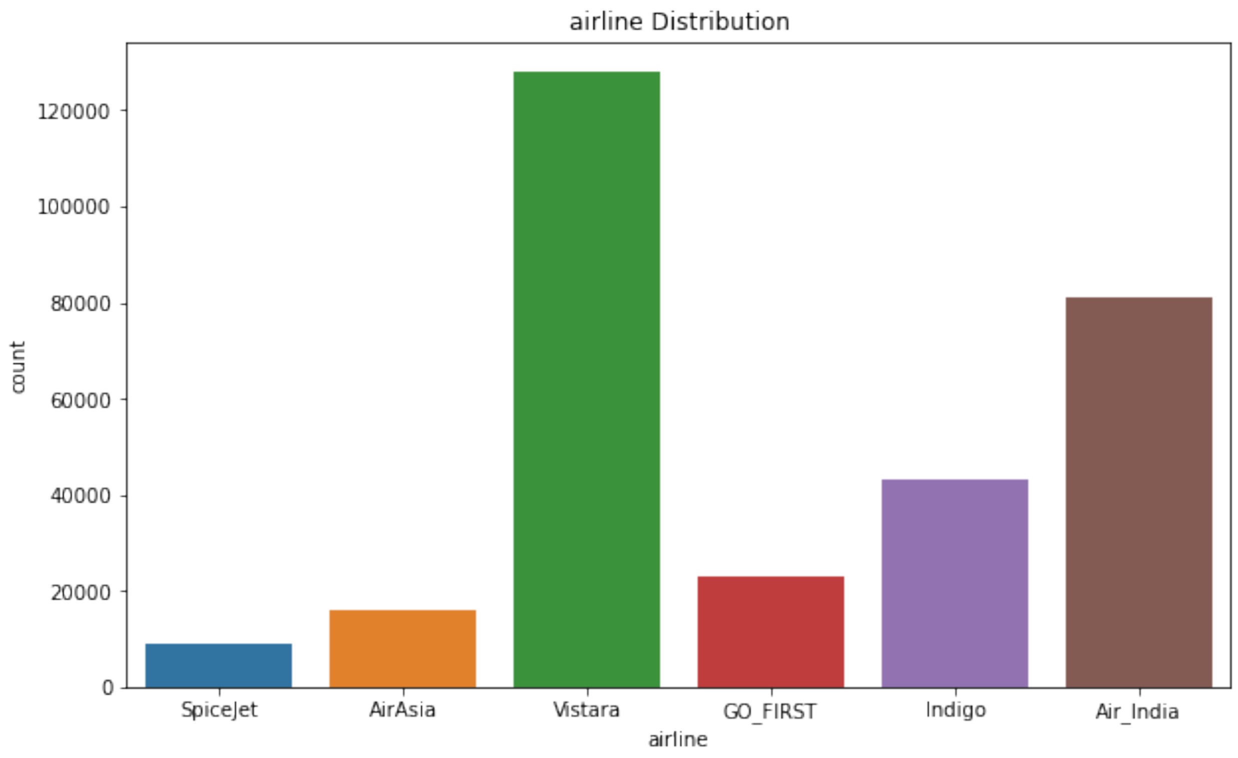
```
df_pyspark_dropped = df_pyspark.na.drop()
df_pyspark_dropped.count()
```

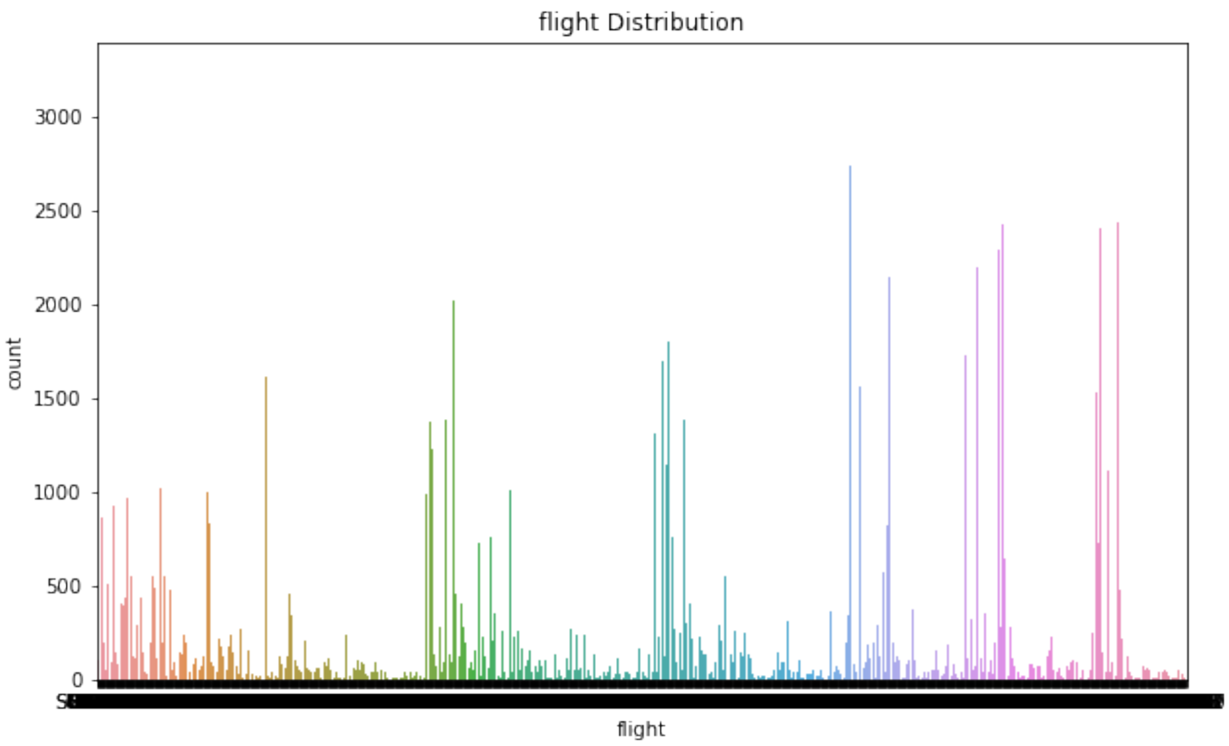
```
300153
```

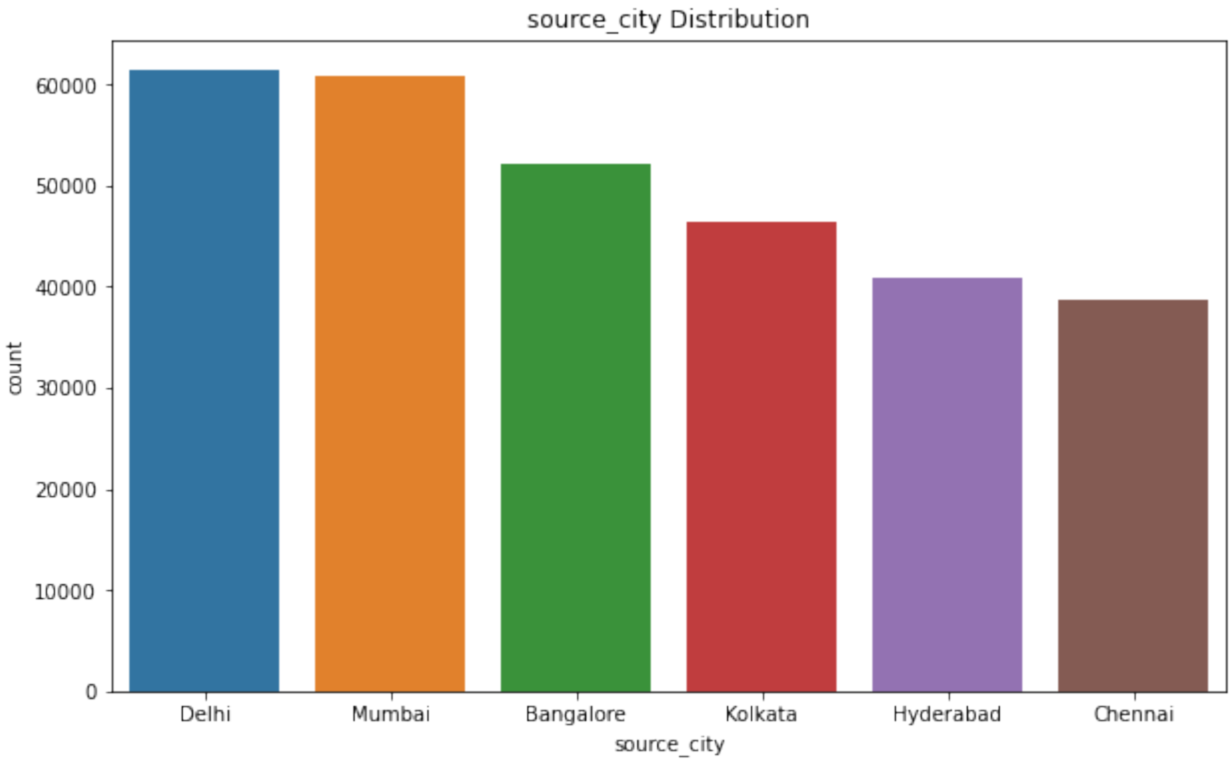
There is not any null value

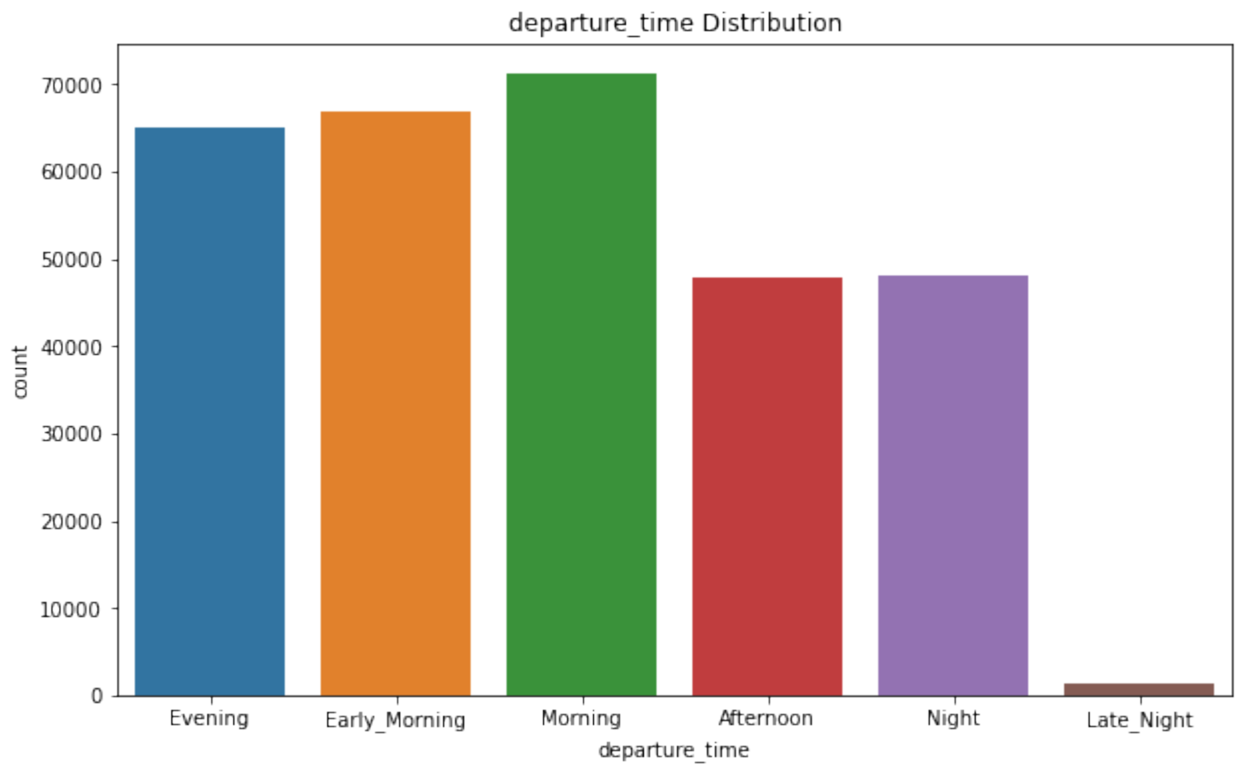
Data Analysis and Visualition

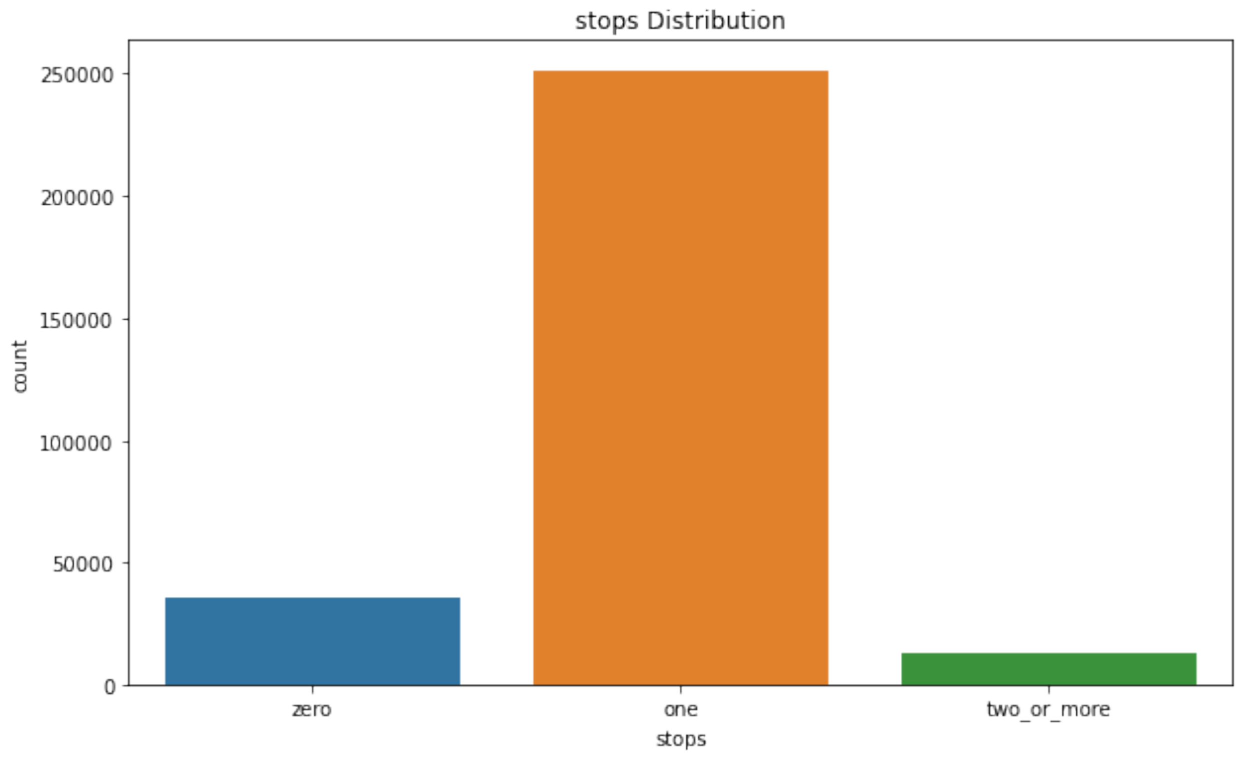
```
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql.functions import col

# Distribution for categorical variables
for feature in categorical_features:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=feature, data=df_pyspark.toPandas())
    plt.title(f"{feature} Distribution")
    plt.show()

# Distribution for continuous variables
for feature in continuous_features:
    plt.figure(figsize=(10, 6))
    sns.histplot(df_pyspark.select(col(feature)).toPandas(), bins=30,
kde=True)
    plt.title(f"{feature} Distribution")
    plt.show()
```

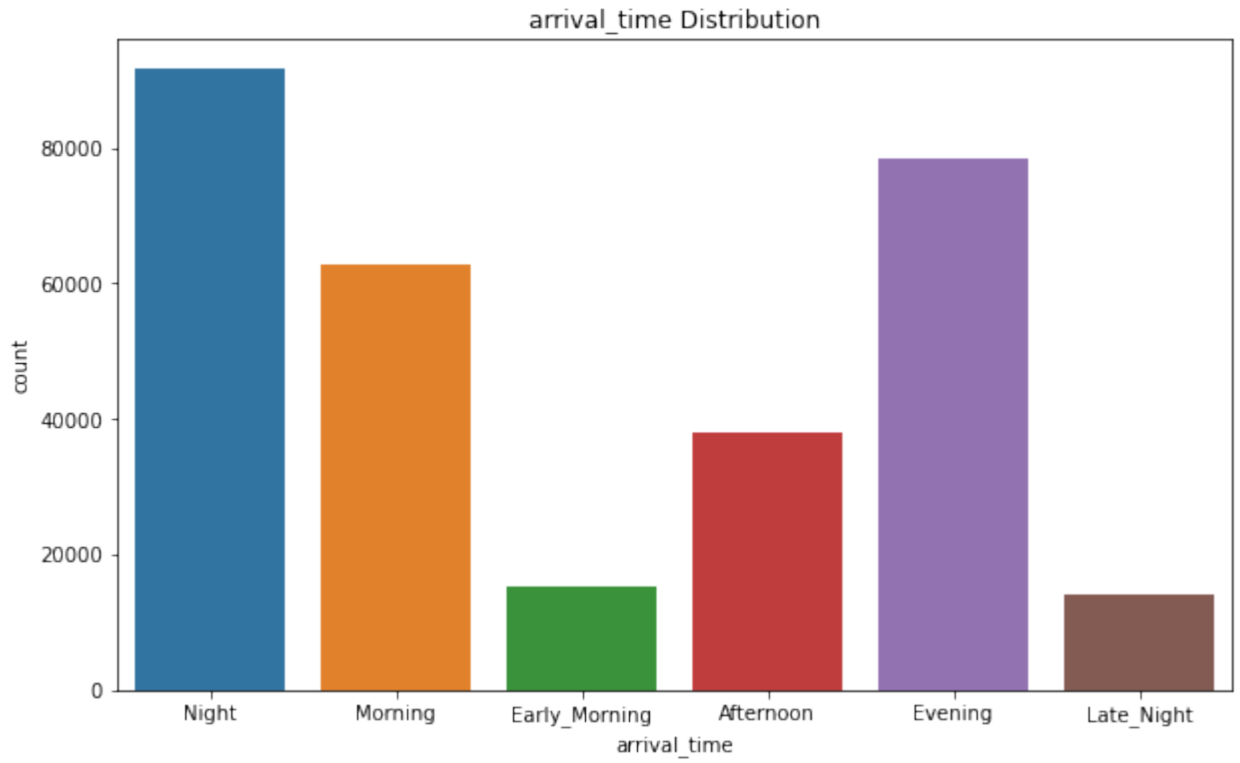


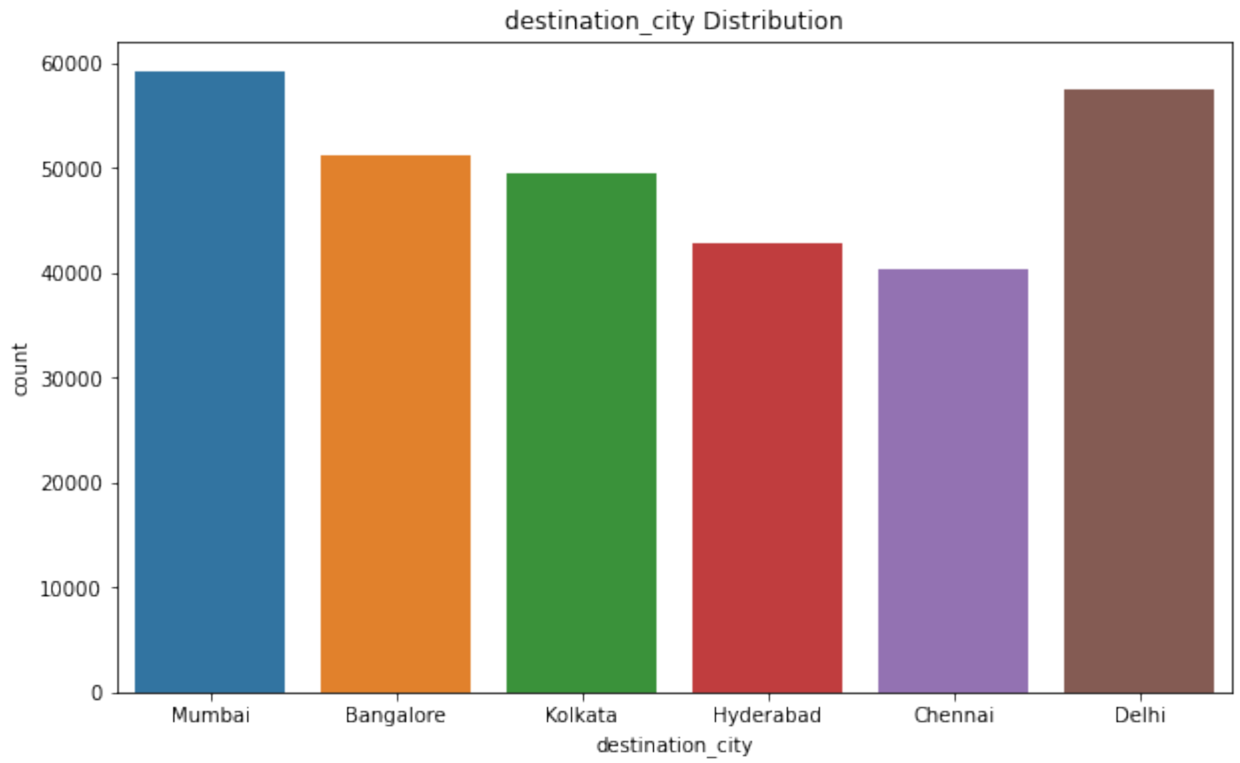

flight Distribution
count
3000
2500
2000
1500
1000
500
0
flight

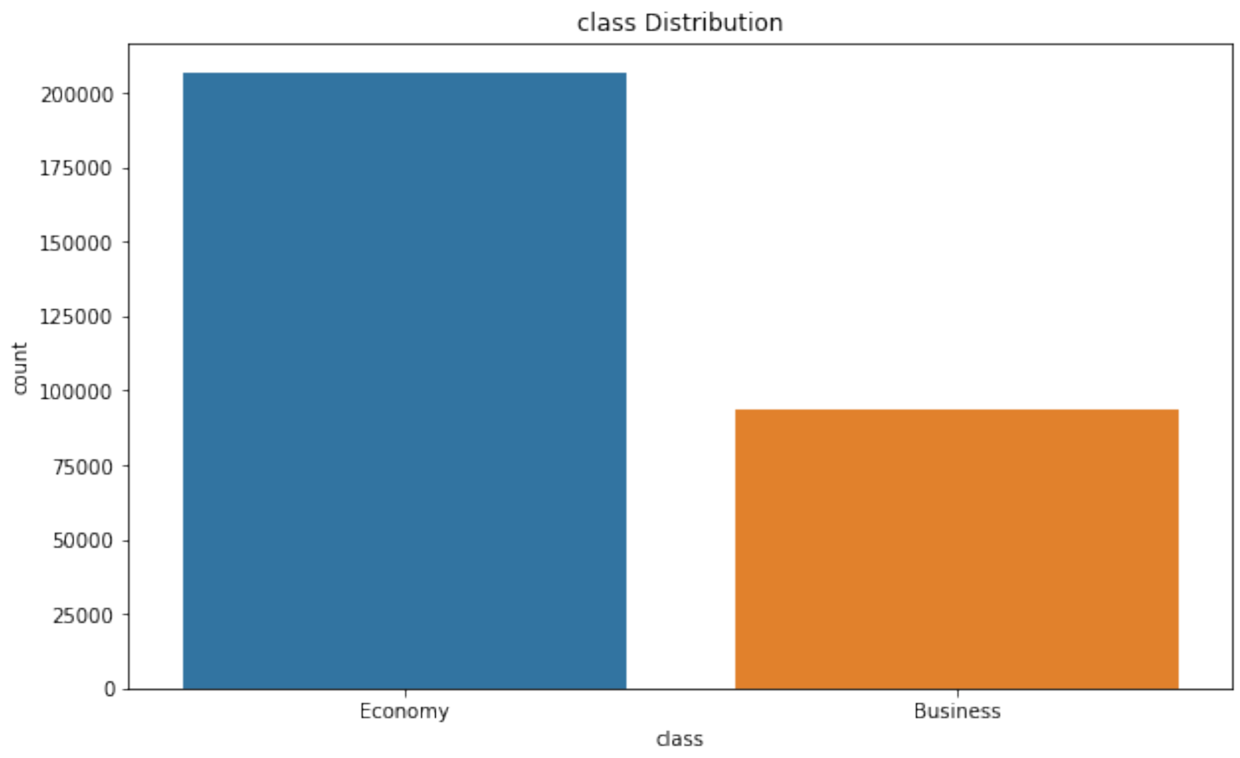

source_city Distribution
count
60000
50000
40000
30000
20000
10000
0
Delhi
Mumbai
Bangalore
Kolkata
Hyderabad
Chennai
source_city

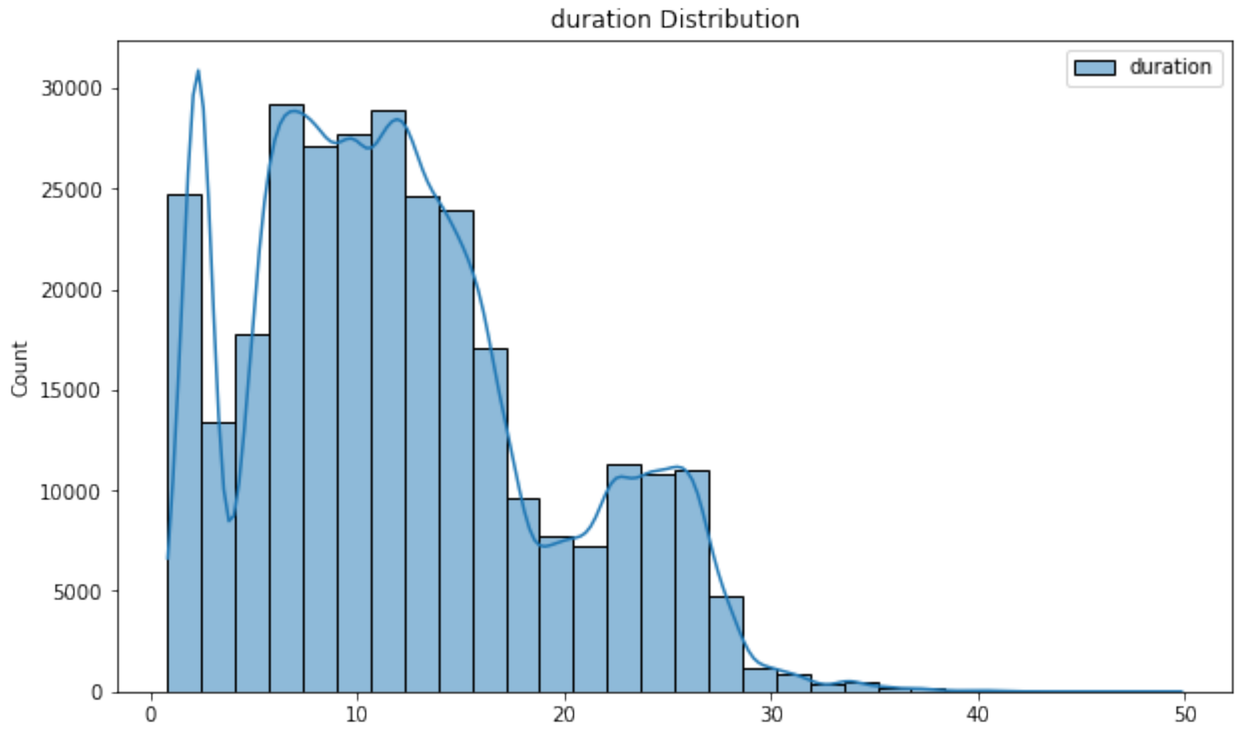

departure_time Distribution
count
70000
60000
50000
40000
30000
20000
10000
0
Evening
Early_Morning
Morning
Afternoon
Night
Late_Night
departure_time

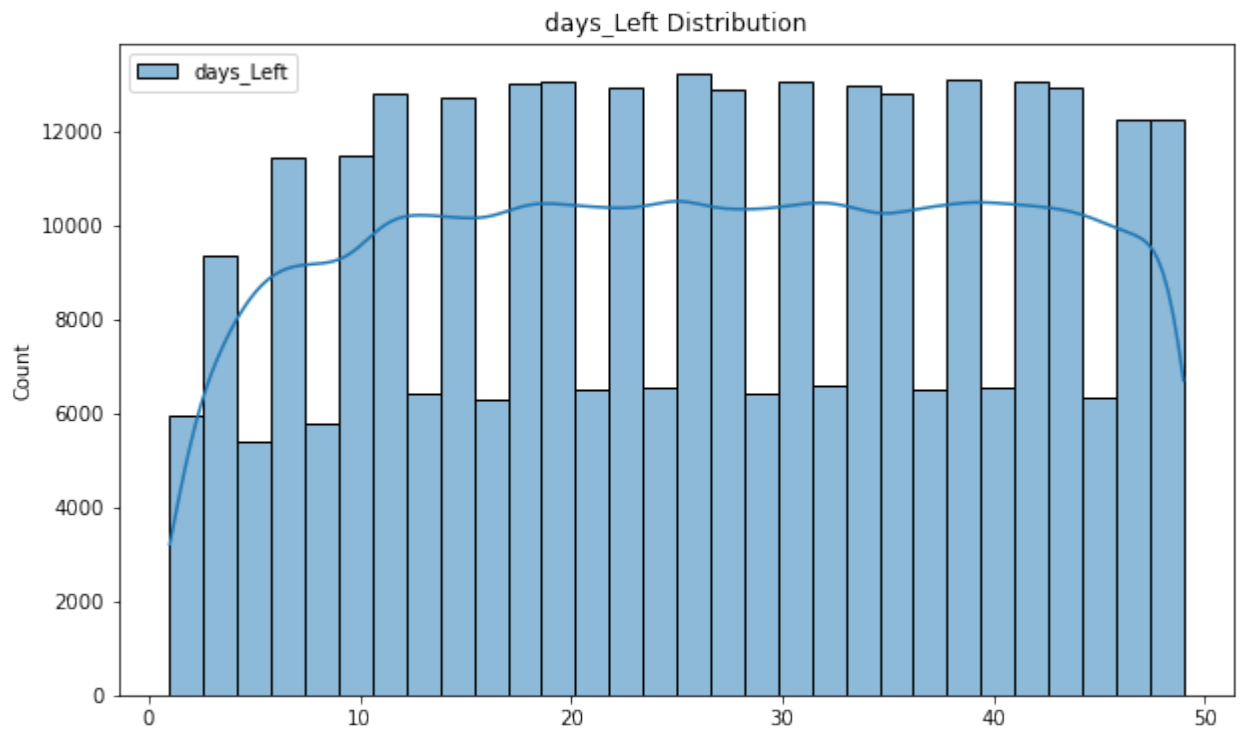

stops Distribution
count
250000
200000
150000
100000
50000
0
zero
one
two_or_more
stops

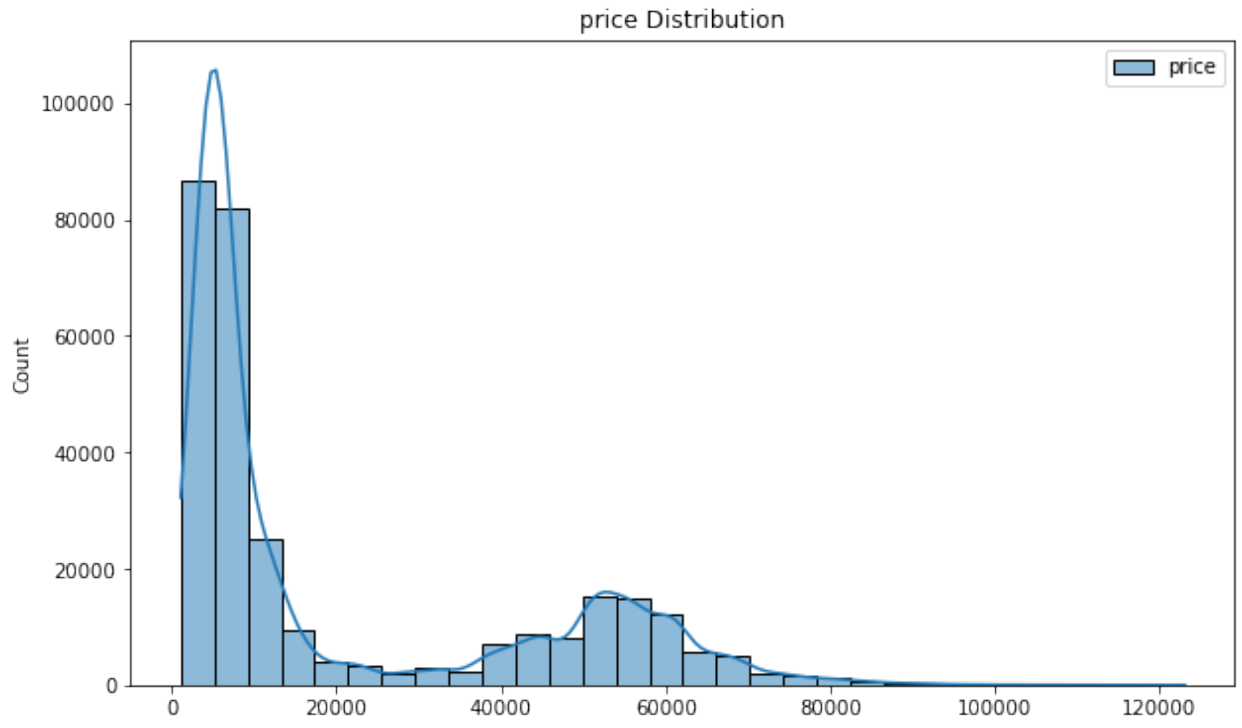

arrival_time Distribution
count
0
20000
40000
60000
80000
Night
Morning
Early_Morning
Afternoon
Evening
Late_Night
arrival_time


destination_city Distribution
count
0
10000
20000
30000
40000
50000
60000
Mumbai
Bangalore
Kolkata
Hyderabad
Chennai
Delhi
destination_city


class Distribution
200000
175000
150000
125000
100000
75000
50000
25000
0
count
Economy
Business
class

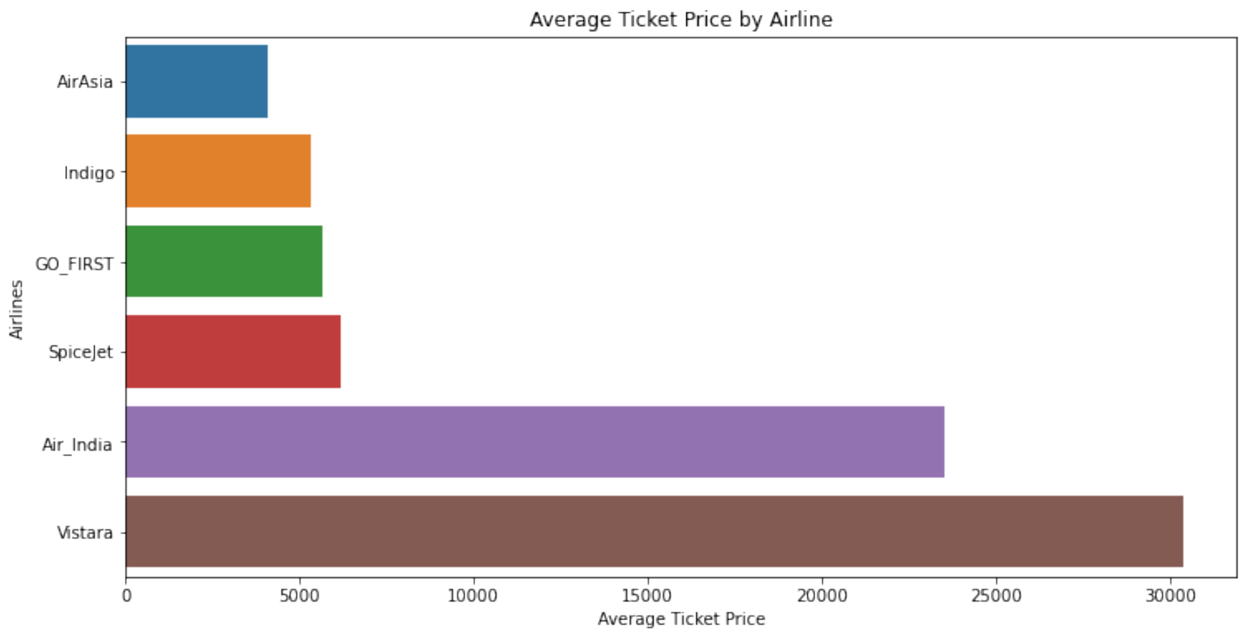

duration Distribution
duration
30000
25000
20000
15000
10000
5000
0
Count
0
10
20
30
40
50





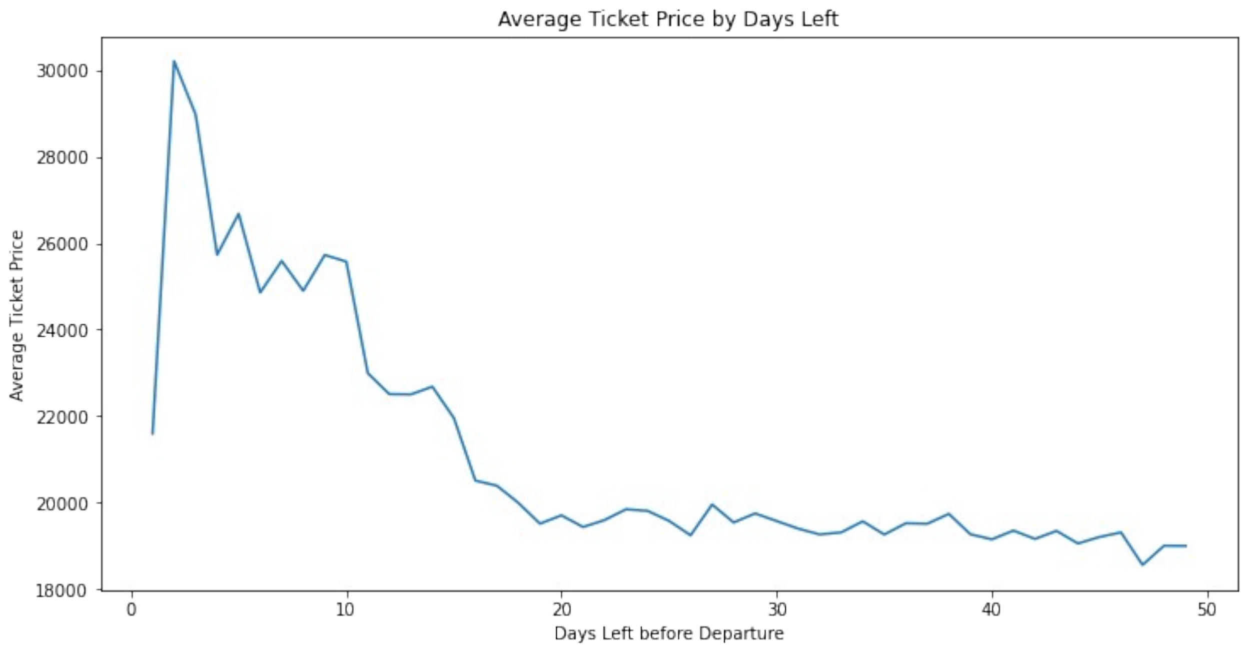
```
# Group by Airline and calculate average price
avg_price_by_airline = df_pyspark.groupBy("airline").agg({"price":
"avg"}).sort("avg(price)")

# Visualization
```

```
plt.figure(figsize=(12, 6))
sns.barplot(x="avg(price)", y="airline",
data=avg_price_by_airline.toPandas())
plt.title("Average Ticket Price by Airline")
plt.xlabel("Average Ticket Price")
plt.ylabel("Airlines")
plt.show()
```



Airline Vista has the highest average ticket price

```
# Group by Days Left and calculate average price
avg_price_by_days_left = df_pyspark.groupBy("days_left").agg({"price":
"avg"}).sort("days_left")

# Visualization
plt.figure(figsize=(12, 6))
sns.lineplot(x="days_left", y="avg(price)",
data=avg_price_by_days_left.toPandas())
plt.title("Average Ticket Price by Days Left")
plt.xlabel("Days Left before Departure")
plt.ylabel("Average Ticket Price")
plt.show()
```

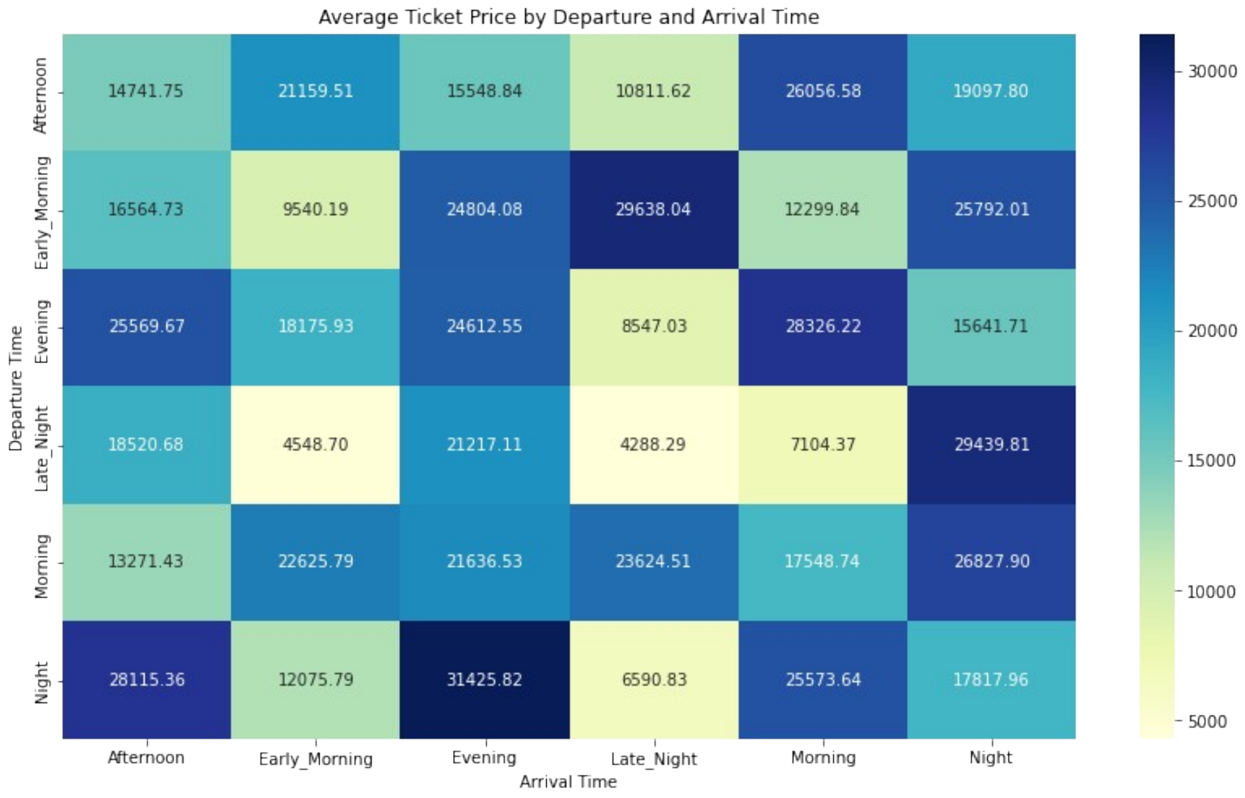


Prices generally increase as the departure date approaches. However, there is a slight dip in prices for tickets booked a day or two before departure

```
# Group by Departure Time and Arrival Time and calculate average price
avg_price_by_time = df_pyspark.groupBy("departure_time",
"arrival_time").agg({"price": "avg"})

# Convert PySpark DataFrame to Pandas DataFrame for visualization
avg_price_by_time_pd = avg_price_by_time.toPandas()

# Visualization
plt.figure(figsize=(14, 8))
sns.heatmap(
    avg_price_by_time_pd.pivot("departure_time", "arrival_time",
"avg(price)"),
    annot=True,
    cmap="YlGnBu",
    fmt=".2f"
)
plt.title("Average Ticket Price by Departure and Arrival Time")
plt.xlabel("Arrival Time")
plt.ylabel("Departure Time")
plt.show()
```

```
/var/folders/k7/rkwcp6pd0vn42m0g7j349s9r0000gn/T/
ipykernel_57763/2976931535.py:10: FutureWarning: In a future version
of pandas all arguments of DataFrame.pivot will be keyword-only.
```

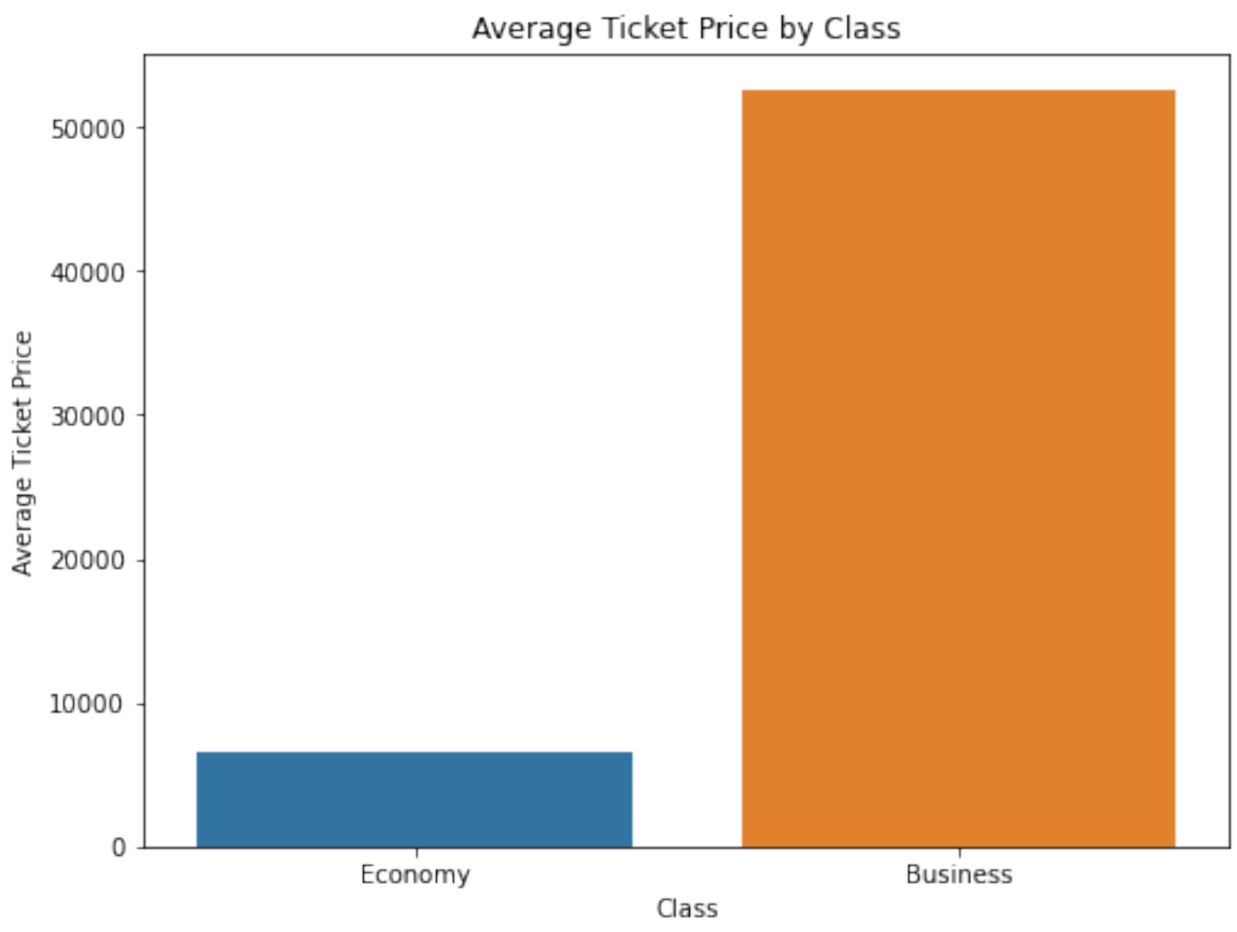
```
  avg_price_by_time_pd.pivot("departure_time", "arrival_time",
"avg(price)"),
```



Departure \ Arrival	Afternoon	Early_Morning	Evening	Late_Night	Morning	Night
Afternoon	14741.75	21159.51	15548.84	10811.62	26056.58	19097.80
Early_Morning	16564.73	9540.19	24804.08	29638.04	12299.84	25792.01
Evening	25569.67	18175.93	24612.55	8547.03	28326.22	15641.71
Late_Night	18520.68	4548.70	21217.11	4288.29	7104.37	29439.81
Morning	13271.43	22625.79	21636.53	23624.51	17548.74	26827.90
Night	28115.36	12075.79	31425.82	6590.83	25573.64	17817.96

Flights departing in the early morning or late evening, especially with midday arrivals, tend to have lower prices. In contrast, peak travel times during the day exhibit higher average prices.

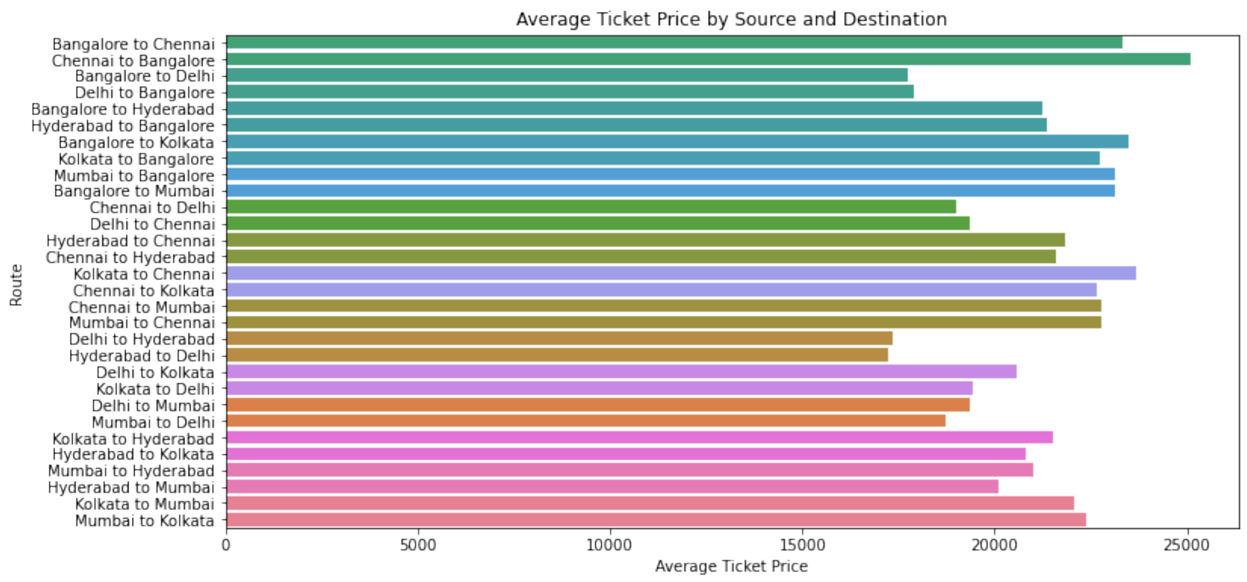
```
# Group by Class and calculate average price
avg_price_by_class = df_pyspark.groupBy("class").agg({"price": "avg"})

# Convert PySpark DataFrame to Pandas DataFrame for visualization
avg_price_by_class_pd = avg_price_by_class.toPandas()

# Visualization
plt.figure(figsize=(8, 6))
sns.barplot(x="class", y="avg(price)", data=avg_price_by_class_pd)
plt.title("Average Ticket Price by Class")
plt.xlabel("Class")
plt.ylabel("Average Ticket Price")
plt.show()
```



Business class tickets are considerably more expensive than Economy class tickets

```
# Create a function to use for creating new column

def identify_same_pairs(route):
    cities = route.split(" to ")
    return "-".join(sorted(cities))


# Group by Source City and Destination City and calculate average
price
avg_price_by_route = df_pyspark.groupBy("source_city",
"destination_city").agg({"price": "avg"})

# Convert PySpark DataFrame to Pandas DataFrame for visualization
avg_price_by_route_pd = avg_price_by_route.toPandas()

# Add "Route" and "Sorted Route" columns to use for sorting
avg_price_by_route_pd["Route"] = avg_price_by_route_pd["source_city"]
+ " to " + avg_price_by_route_pd["destination_city"]
avg_price_by_route_pd["Sorted Route"] =
avg_price_by_route_pd["Route"].apply(identify_same_pairs)
```

```
# Sort DataFrame based on the new "Route" column
avg_price_by_route_pd_sorted = (
    avg_price_by_route_pd.sort_values(by="Sorted Route")
    .reset_index(drop=True)
)

# Create a color palette
color_palette = sns.color_palette("husl",
n_colors=len(avg_price_by_route_pd_sorted) // 2)

# Visualization
plt.figure(figsize=(12, 6))
sns.barplot(
    x="avg(price)",
    y="Route",
    data=avg_price_by_route_pd_sorted,
    palette=sorted(color_palette * 2 ) # Repeat the color palette for
each pair

)
plt.title("Average Ticket Price by Source and Destination")
plt.xlabel("Average Ticket Price")
plt.ylabel("Route")
plt.show()
```



routes involving Delhi have lower prices while routes involving Chennai have higher prices than other cities.

Machine Learning Models

1.1. Decision Tree Regression

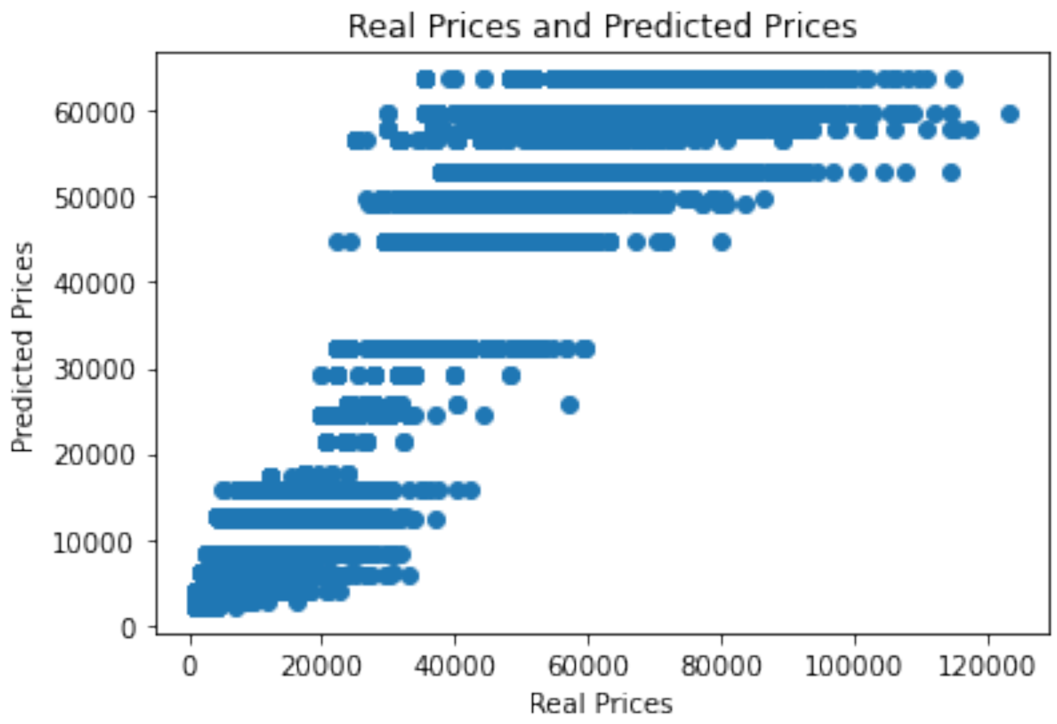
```
from pyspark.sql.functions import col

# Detects problematic categorical features in a PySpark DataFrame
based on a count threshold.
# The features that have higher number of distinct values than the
specified threshold cause error in Decision Tree Regression

def detect_problematic_categorical_features(df_pyspark,threshold=32):
    problematic_features = []

    # Iterate through columns in the DataFrame schema
    for column in df_pyspark.schema:
        # Check if the column data type is a categorical feature)
        if "StringType" in str(column.dataType):
            # Calculate the distinct count of values in the
categorical feature
            distinct_count =
df_pyspark.select(col(column.name)).distinct().count()

            # Check if the distinct count exceeds the specified
threshold
            if distinct_count > threshold:

                problematic_features.append(column.name)

    return problematic_features

# Detect and handle problematic categorical features
problematic_features =
detect_problematic_categorical_features(df_pyspark)
categorical_features_dropped = [col for col in categorical_features if
col not in problematic_features]
df_pyspark_dropped = df_pyspark.select(['duration', 'days_left',
'price'] + categorical_features_dropped)

# Create StringIndexer for each categorical feature
indexers = [StringIndexer(inputCol=col,
outputCol=col+"_index").fit(df_pyspark_dropped) for col in
categorical_features_dropped]

# Create a pipeline to apply StringIndexers to the DataFrame
pipeline = Pipeline(stages=indexers)
df_pyspark_indexed =
pipeline.fit(df_pyspark_dropped).transform(df_pyspark_dropped)

# Drop unnecessary columns
selected_features = ['duration', 'days_left', 'price'] + [col+"_index"
```

```
for col in categorical_features_dropped]
df_pyspark_final = df_pyspark_indexed.select(selected_features)

# Assemble features into a single vector
assembler = VectorAssembler(inputCols=['duration', 'days_left'] +
[col+"_index" for col in categorical_features_dropped],
outputCol='features')
df_pyspark_assembled = assembler.transform(df_pyspark_final)

# Split the data into training and testing sets
(train_data, test_data) = df_pyspark_assembled.randomSplit([0.7, 0.3])

# Create a Decision Tree Regression model and train it
classifier = DecisionTreeRegressor(labelCol = 'price', featuresCol =
'features')
#pipeline = Pipeline(stages= indexers + [assembler, classifier])
model = classifier.fit(train_data)

# Make predictions on the test data
predictions = model.transform(test_data)
predictions.select("prediction", "price", "features").show()
```

```
+------------------+-----+--------------------+
|        prediction|price|            features|
+------------------+-----+--------------------+
| 8139.783439490446| 5102|[1.58,1.0,3.0,1.0...|
| 5733.385383567862| 5943|[1.83,6.0,2.0,1.0...|
| 5733.385383567862| 5943|[1.83,11.0,2.0,1....|
| 5733.385383567862| 5943|[1.83,13.0,2.0,1....|
|3784.0134831460673| 2738|[1.83,16.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,18.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,20.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,23.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,25.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,31.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,32.0,2.0,1....|
| 2139.342668863262| 3424|[1.83,35.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,40.0,2.0,1....|
| 2139.342668863262| 2738|[1.83,41.0,2.0,1....|
| 5733.385383567862| 6521|[1.92,6.0,2.0,1.0...|
| 5733.385383567862| 6521|[1.92,7.0,2.0,1.0...|
| 5733.385383567862| 6153|[1.92,14.0,2.0,1....|
| 2139.342668863262| 3108|[1.92,18.0,2.0,1....|
| 2139.342668863262| 3108|[1.92,19.0,2.0,1....|
| 2139.342668863262| 3108|[1.92,23.0,2.0,1....|
+------------------+-----+--------------------+
only showing top 20 rows
```

1.2. Evaluation of Decision Tree Regression Performance

```
# Evaluate the model's performance using Root Mean Squared Error
(RMSE)
evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE) on test data: {rmse}")

# Evaluate the model's performance using Coefficient of Determination
(R²)
evaluator2 = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="r2")
r2 = evaluator2.evaluate(predictions)
print(f"Coefficient of determination (R2) on test data: {r2}")

Root Mean Squared Error (RMSE) on test data: 5299.6741252199945
Coefficient of determination (R2) on test data: 0.9455754403796409

predictions_pd = predictions.select("prediction", "price").toPandas()
plt.scatter(predictions_pd["price"], predictions_pd["prediction"])
plt.xlabel("Real Prices")
plt.ylabel("Predicted Prices")
plt.title("Real Prices and Predicted Prices")
plt.show()
```



2.1. Linear Regression

```
# Use StringIndexer to convert categorical features to numerical
indices
indexers = [StringIndexer(inputCol=col,
outputCol=col+"_index").fit(df_pyspark) for col in
categorical_features]

# Create a pipeline to execute the StringIndexer transformations
pipeline = Pipeline(stages=indexers)
df_pyspark_indexed = pipeline.fit(df_pyspark).transform(df_pyspark)

# Drop unnecessary columns
selected_features = ['duration', 'days_left', 'price'] + [col+"_index"
for col in categorical_features]
df_pyspark_final = df_pyspark_indexed.select(selected_features)

# Assemble features into a single vector
assembler = VectorAssembler(inputCols=['duration', 'days_left'] +
[col+"_index" for col in categorical_features], outputCol='features')
df_pyspark_assembled = assembler.transform(df_pyspark_final)

# Split the data into training and testing sets
train_data, test_data = df_pyspark_assembled.randomSplit([0.8, 0.2],
seed=123)

# Train a Linear Regression model
```

```
lr = LinearRegression(featuresCol='features', labelCol='price')
model = lr.fit(train_data)

# Make predictions on the test data
predictions = model.transform(test_data)
predictions.select("prediction", "price", "features").show()
```

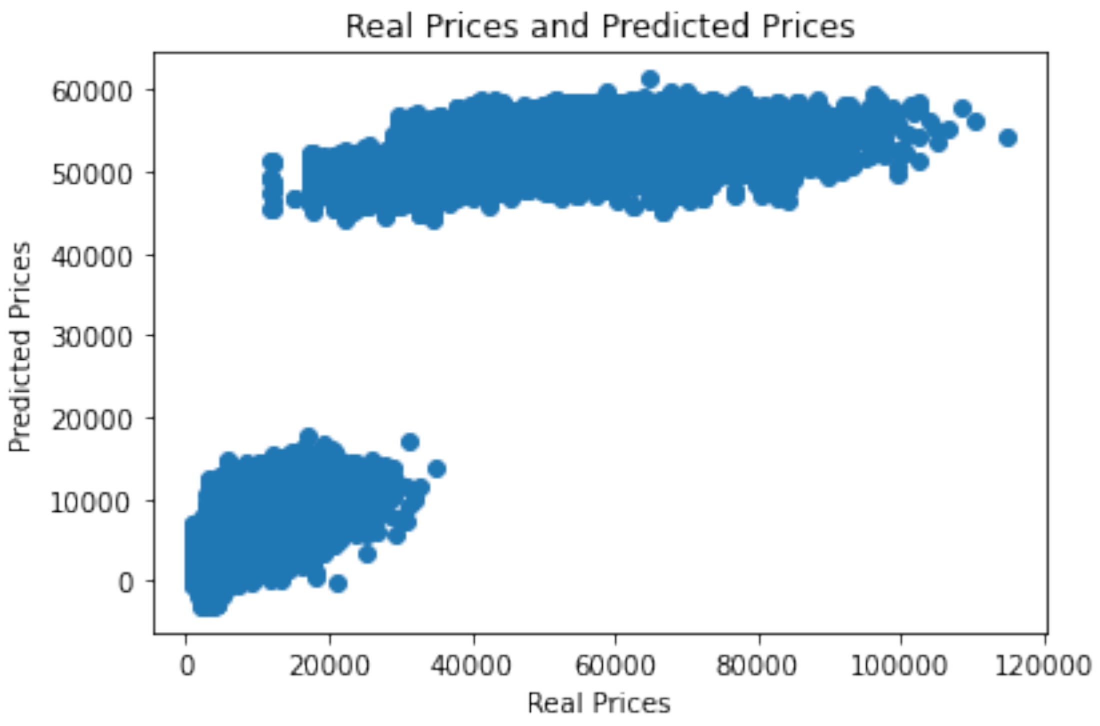
```
23/12/18 06:43:01 WARN Instrumentation: [691581e3] regParam is zero,
which might cause numerical instability and overfitting.


+------------------+-----+--------------------+
|        prediction|price|            features|
+------------------+-----+--------------------+
| 6887.886137410519|15708|[1.83,1.0,2.0,814...|
|  6370.75286893356| 5943|[1.83,5.0,2.0,814...|
| 5595.052966218123| 5943|[1.83,11.0,2.0,81...|
| 5465.769649098884| 5943|[1.83,12.0,2.0,81...|
|4819.3530635026855| 2738|[1.83,17.0,2.0,81...|
| 4172.936477906487| 2738|[1.83,22.0,2.0,81...|
|3397.2365751910493| 2738|[1.83,28.0,2.0,81...|
|3267.9532580718105| 3108|[1.83,29.0,2.0,81...|
|2880.1033067140916| 2738|[1.83,32.0,2.0,81...|
| 2750.819989594852| 2738|[1.83,33.0,2.0,81...|
|2621.5366724756122| 3108|[1.83,34.0,2.0,81...|
|2233.6867211178933| 2738|[1.83,37.0,2.0,81...|
| 1975.120086879414| 2738|[1.83,39.0,2.0,81...|
| 6341.788610149189| 5943|[1.92,3.0,2.0,815...|
| 5307.522073195271| 6521|[1.92,11.0,2.0,81...|
| 4531.822170479834| 2738|[1.92,17.0,2.0,81...|
|    2463.289096572| 2738|[1.92,33.0,2.0,81...|
| 2075.439145214281| 3108|[1.92,36.0,2.0,81...|
|1429.0225596180826| 3424|[1.92,41.0,2.0,81...|
| 6640.563738391747| 5955|[2.0,2.0,1.0,131....|
+------------------+-----+--------------------+
only showing top 20 rows
```

2.2. Evaluation of Linear Regression Performance

```
# Evaluate the model's performance using Root Mean Squared Error
(RMSE)
evaluator = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE) on test data: {rmse}")

# Evaluate the model's performance using Coefficient of Determination
(R²)
evaluator2 = RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="r2")
r2 = evaluator2.evaluate(predictions)
print(f"Coefficient of determination (R2) on test data: {r2}")
```

```
Root Mean Squared Error (RMSE) on test data: 7277.873117983088
Coefficient of determination (R2) on test data: 0.8968568169149093
```

```
predictions_pd = predictions.select("prediction", "price").toPandas()
plt.scatter(predictions_pd["price"], predictions_pd["prediction"])
plt.xlabel("Real Prices")
plt.ylabel("Predicted Prices")
plt.title("Real Prices and Predicted Prices")
plt.show()
```


Real Prices and Predicted Prices
Predicted Prices
Real Prices