**CIS 4130**

**Ivan Tan**
**ivan.tan1@baruchmail.cuny.edu**

# Milestone 1

**Summary:** Milestone 1 identifies and describes my plan to collect data on my chosen Kaggle dataset, identifying my key attributes and what I intend to predict.

**Dataset:**
https://www.kaggle.com/datasets/dilwong/flightprices?select=itineraries.csv

**Description:**
My dataset contains the price of one-way airplane tickets domestic within the United States on Expedia between April 16 2022 to October 05 2022. Each row purchasable ticket contains information such as flight id, flight date, departing & arriving airport, travel duration, total distance traveled, elapsed days, whether the seat is in economy, refundable, or non-stop, and lastly, it provides the price of the tickets in USD before and after taxes + other fees.

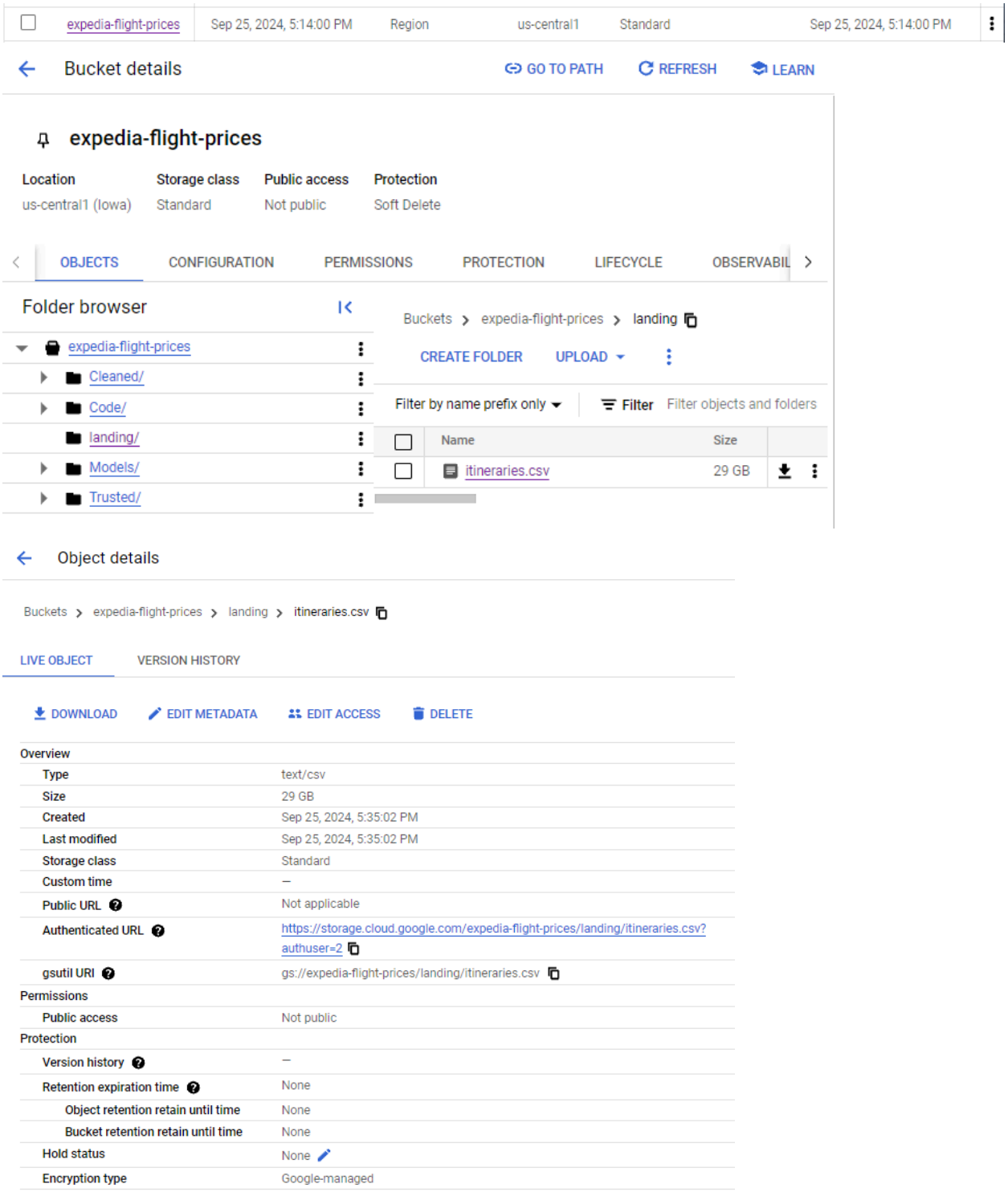
**Data Set Attributes:**

- Flight date: The date (YYYY-MM-DD) of the flight.
- Travel Duration: The travel duration in hours and minutes.
- Total Travel Distance: The total travel distance in miles.
    - Starting Airport: Three-character IATA airport code for the initial location.
    - Destination Airport: Three-character IATA airport code for the arrival location.
    - (Not sure if I need these 2)
- Seats Remaining: Integer for the number of seats remaining.
- Ticket Information
    - isBasicEconomy: Boolean for whether the ticket is for basic economy.
    - isRefundable: Boolean for whether the ticket is refundable.
    - isNonStop: Boolean for whether the flight is non-stop.

**What I intend to predict:**
I plan to predict the price of one-way airplane tickets within the United States. I will be creating a linear regression model using factors such as the time & date of the flight, number of remaining seats on the flight, total distance & duration traveled, and etc.

# Milestone 2

**Summary:** In milestone 2, I downloaded my Kaggle dataset and created a Google Cloud Storage bucket along with the creation of the folders on GCP: landing, cleaned, trusted, code, and models.



# Milestone 3

**Summary:** In milestone 3 I performed a EDA using PySpark to identify my columns, variables, and missing null values. I also developed a separate cleaning pyspark session to handle the missing data and remove unnecessary columns. In the en,d I save my cleaned data as a Parquet file in my cleaned folder,

After performing the EDA to identify the columns, variables, and number of null and missing values, I cleaned my dataset only to contain the certain variable columns I will use. For the null and missing values, I eliminated the rows with categorical values and replaced the null values for the numerical columns with their averages. The variables I have chosen, such as flight date, starting and arriving airport, travel duration, total distance traveled, if it is economy, refundable, or non-stop, days elapsed, and the number of seats remaining, are variables I believe that can help impact and determine the price of plane tickets. A challenge I believe I will face in feature engineering is dealing with the starting and destination airports. Because these values are categorical, it would be a challenge due to the large number of different airports in the United States.

**EDA OUTPUT**

Sampled Data (4/10 of total) - Number of observations: 32856720

Sampled Data (4/10 of total) - List of variables (columns):
['legId', 'searchDate', 'flightDate', 'startingAirport', 'destinationAirport', 'fareBasisCode', 'travelDuration', 'elapsedDays', 'isBasicEconomy', 'isRefundable', 'isNonStop', 'baseFare', 'totalFare', 'seatsRemaining', 'totalTravelDistance', 'segmentsDepartureTimeEpochSeconds', 'segmentsDepartureTimeRaw', 'segmentsArrivalTimeEpochSeconds', 'segmentsArrivalTimeRaw', 'segmentsArrivalAirportCode', 'segmentsDepartureAirportCode', 'segmentsAirlineName', 'segmentsAirlineCode', 'segmentsEquipmentDescription', 'segmentsDurationInSeconds', 'segmentsDistance', 'segmentsCabinCode']

Sampled Data (4/10 of total) - Number of missing fields:

legId: 0
searchDate: 0
flightDate: 0
startingAirport: 0
destinationAirport: 0
fareBasisCode: 0
travelDuration: 0
elapsedDays: 0
isBasicEconomy: 0

isRefundable: 0
isNonStop: 0
baseFare: 0
totalFare: 0
seatsRemaining: 0
totalTravelDistance: 2437093
segmentsDepartureTimeEpochSeconds: 0
segmentsDepartureTimeRaw: 0
segmentsArrivalTimeEpochSeconds: 0
segmentsArrivalTimeRaw: 0
segmentsArrivalAirportCode: 0
segmentsDepartureAirportCode: 0
segmentsAirlineName: 0
segmentsAirlineCode: 0
segmentsEquipmentDescription: 623444
segmentsDurationInSeconds: 0
segmentsDistance: 0
segmentsCabinCode: 0

Sampled Data (4/10 of total) - elapsedDays:
  Mean: 0.1496525216150608, StdDev: 0.35675796909842983, Min: 0, Max: 2

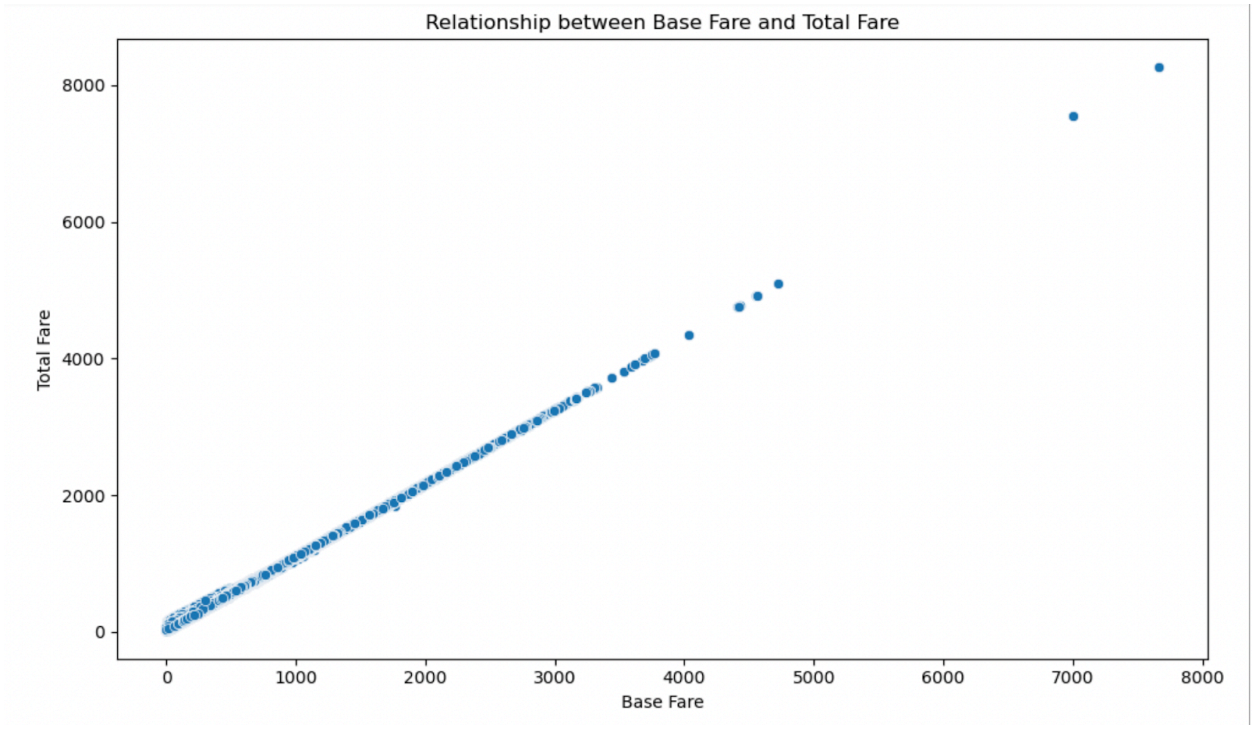
Sampled Data (4/10 of total) - baseFare:
  Mean: 292.6578764471919, StdDev: 183.21769937434837, Min: 0.01, Max: 7662.33

Sampled Data (4/10 of total) - totalFare:
  Mean: 340.38158671797026, StdDev: 196.05925307577652, Min: 19.59, Max: 8260.61

Sampled Data (4/10 of total) - seatsRemaining:
  Mean: 5.977040100168246, StdDev: 2.880197781037374, Min: 0, Max: 10

Sampled Data (4/10 of total) - totalTravelDistance:
  Mean: 1609.9039869226535, StdDev: 857.2751246032351, Min: 89, Max: 7252

**GRAPH**




**CLEANING OUTPUT**

Number of missing fields per column:
flightDate: 0
startingAirport: 0
destinationAirport: 0
travelDuration: 0
isBasicEconomy: 0
isRefundable: 0
isNonStop: 0
elapsedDays: 0
totalTravelDistance

```
|flightDate|startingAirport|destinationAirport|travelDuration|isBasicEconomy|isRefundable|isNonStop|elapsedDays|tot
alTravelDistance|seatsRemaining|
+----------+---------------+------------------+--------------+--------------+------------+---------+-----------+---
----------------+-------------+
|2022-04-17|ATL            |BOS               |PT2H29M       |false         |false       |true     |0          |947
|9            |
|2022-04-17|ATL            |BOS               |PT2H30M       |false         |false       |true     |0          |947
|4            |
|2022-04-17|ATL            |BOS               |PT2H30M       |false         |false       |true     |0          |947
|9            |
|2022-04-17|ATL            |BOS               |PT2H32M       |false         |false       |true     |0          |947
|8            |
|2022-04-17|ATL            |BOS               |PT2H34M       |false         |false       |true     |0          |947
|9            |
|2022-04-17|ATL            |BOS               |PT2H38M       |false         |false       |true     |0          |947
|7            |
|2022-04-17|ATL            |BOS               |PT4H12M       |false         |false       |false    |0          |956
|3            |
|2022-04-17|ATL            |BOS               |PT5H18M       |false         |false       |false    |0          |956
|3            |
|2022-04-17|ATL            |BOS               |PT5H32M       |false         |false       |false    |0          |956
|7            |
|2022-04-17|ATL            |BOS               |PT6H38M       |false         |false       |false    |0          |956
|7            |
|2022-04-17|ATL            |BOS               |PT4H46M       |false         |false       |false    |0          |947
|1            |
|2022-04-17|ATL            |BOS               |PT5H45M       |false         |false       |false    |0          |146
2            |3            |
|2022-04-17|ATL            |BOS               |PT5H59M       |false         |false       |false    |0          |146
2            |5            |
|2022-04-17|ATL            |BOS               |PT7H18M       |false         |false       |false    |0          |146
2            |3            |
|2022-04-17|ATL            |BOS               |PT8H10M       |false         |false       |false    |0          |146
2            |2            |
|2022-04-17|ATL            |BOS               |PT5H39M       |false         |false       |false    |0          |160
9            |0            |
|2022-04-17|ATL            |BOS               |PT2H38M       |false         |false       |true     |0          |947
|7            |
|2022-04-17|ATL            |BOS               |PT4H17M       |false         |false       |false    |0          |956
|1            |
|2022-04-17|ATL            |BOS               |PT4H36M       |false         |false       |false    |0          |956
|1            |
|2022-04-17|ATL            |BOS               |PT4H45M       |false         |false       |false    |0          |956
|1            |
+----------+---------------+------------------+--------------+--------------+------------+---------+-----------+---
----------------+-------------+
only showing top 20 rows
```

# Milestone 4

**Summary:** In milestone 4 I performed feature engineering to train my predictive model using PySpark. I document the features for each column, create a training and testing split for my data, create model evaluation metrics, and save my proceed data and models in their respective folders.

## Modeling Code Results

Root Mean Squared Error (RMSE) for Total Fare = 141.22827524250079
Mean Absolute Error (MAE) for Total Fare = 95.9769023731764
R-Squared (R2) for Total Fare = 0.47991675304019776
Feature Importances:
startingAirport_index: 0.18801202632853803
destinationAirport_index: 0.18153048245069697
travelDurationMinutes: 0.22097037547373313
elapsedDays: 0.0108156661224697
totalTravelDistance: 0.39867144962456214

My program first reads the cleaned data from the Cleaned/ folder, and then with the raw data set it transforms the data with relevant features in order for it to be suitable for the Random Forest Regressor. It covers my travelDurationMinutes, elapsedDays, and totalTravelDistanced, these numeric features are combined with the categorical variables like startingAirport_index and destinationAirport_index. For the isBasicEconomy and isRefundable columns which values are given in True and False, they are converted to 0 and 1s where the 1s represent True and 0 represents False. This helps to make sure that all the data are in a compatible format for the model. For the modeling section, it takes the pre processed flight data from the Trusted folder after it was been featured engineered and processes the numeric and categorical features through a feature vector. My regressor trains to predict the total fare of flights and calculates the RMSE MAE and R^2. In the results, my given RMSE is 141, MAE is 95, and R^2 is 0.47.

RMSE: The RMSE of 141 indicates that on average my model's predictions deviate from from the actual fare value by around 141 dollars.

MAE: The MAE of 95 shows that the average absolute error between my predicted and actual values for my fare prices is about 95 dollars

R^2: The R^2 value of 0/47 means that 47% of the variance in total fares is explained by the model or how well it fits my model. This means that the remaining 53% of the variation is due to factors not included in my model.

## Modeling Code Predictions Results

| prediction | travelDurationMinutes | elapsedDays | totalTravelDistance |
|---|---|---|---|
| 224.30164022201765 | 120 | 0 | 545 |
| 240.0746304357589 | 265 | 0 | 1090 |
| 224.30164022201765 | 112 | 0 | 545 |
| 223.56459621089124 | 129 | 0 | 545 |
| 258.0358708291695 | 274 | 0 | 990 |
| 317.23236157181447 | 390 | 0 | 990 |
| 382.21201454227236 | 696 | 1 | 918 |
| 388.75924037994776 | 585 | 1 | 2464 |
| 450.33292507693693 | 481 | 1 | 2622 |
| 400.8629247641373 | 513 | 0 | 2337 |

only showing top 10 rows

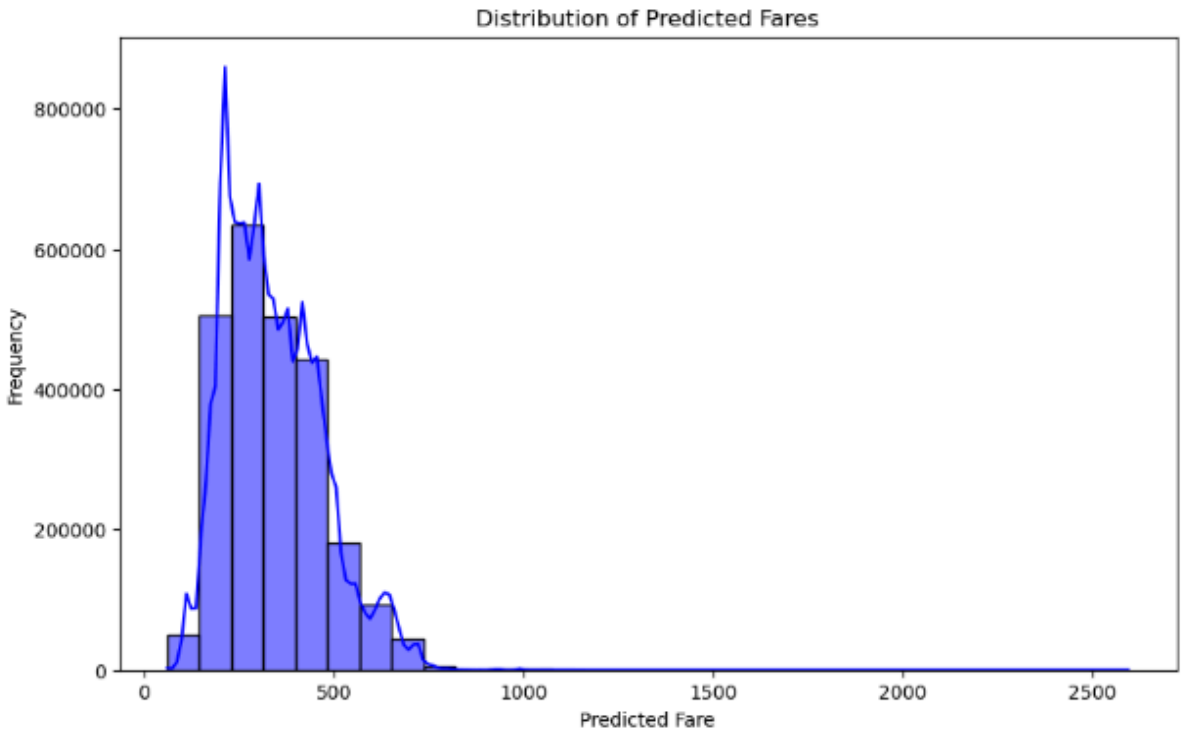
# Milestone 5

**Summary:** In milestone 5 I created 4 different visualisation graphs to showcase the different relationships and correlations between my features and their prediction results.

## RESULTS

```
Summary Statistics:
         prediction  travelDurationMinutes  totalTravelDistance
count  2.464076e+06           2.464076e+06         2.464076e+06
mean   3.402770e+02           4.277826e+02         1.609853e+03
std    1.286497e+02           2.242557e+02         8.247524e+02
min    6.034078e+01           0.000000e+00         8.900000e+01
25%    2.387391e+02           2.620000e+02         9.280000e+02
50%    3.200425e+02           4.090000e+02         1.582000e+03
75%    4.245109e+02           5.660000e+02         2.339000e+03
max    2.595928e+03           1.439000e+03         4.681000e+03
```

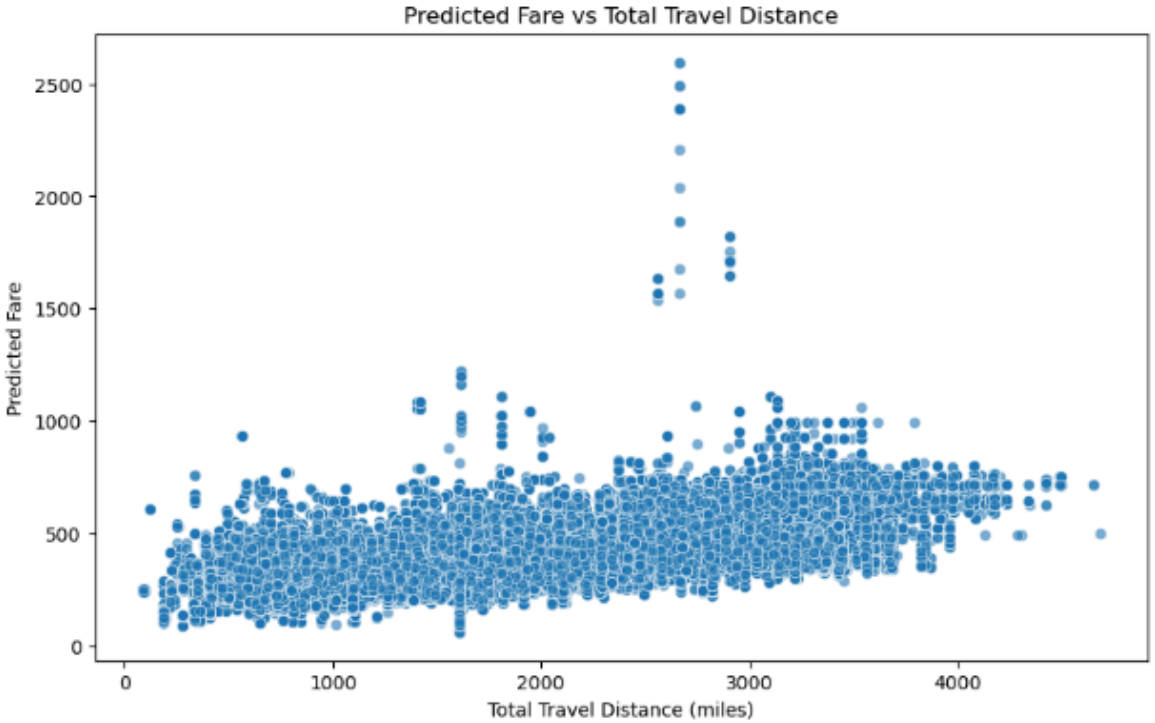
## Histogram: Distribution of Predicted Fares




- This histogram displays the distribution of predicted fare prices. This shows us that the average predicted fare ranges around the prices of $250 - $300 with a frequency of around 630000. We can also see that this histogram is right skewed

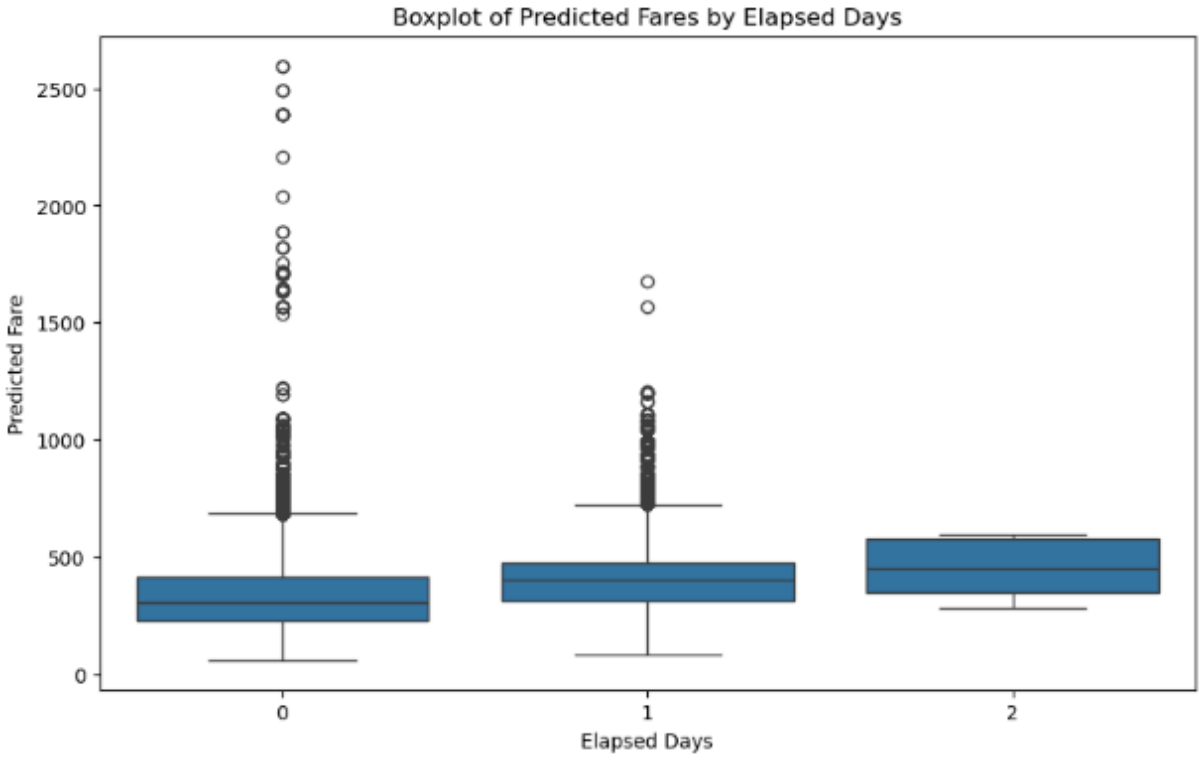
which means that the mean is greater than the median showing that we may have some influence of outliers with higher predicted fare prices.

## Scatter Plot: Predicted Fare vs. Total Travel Distance



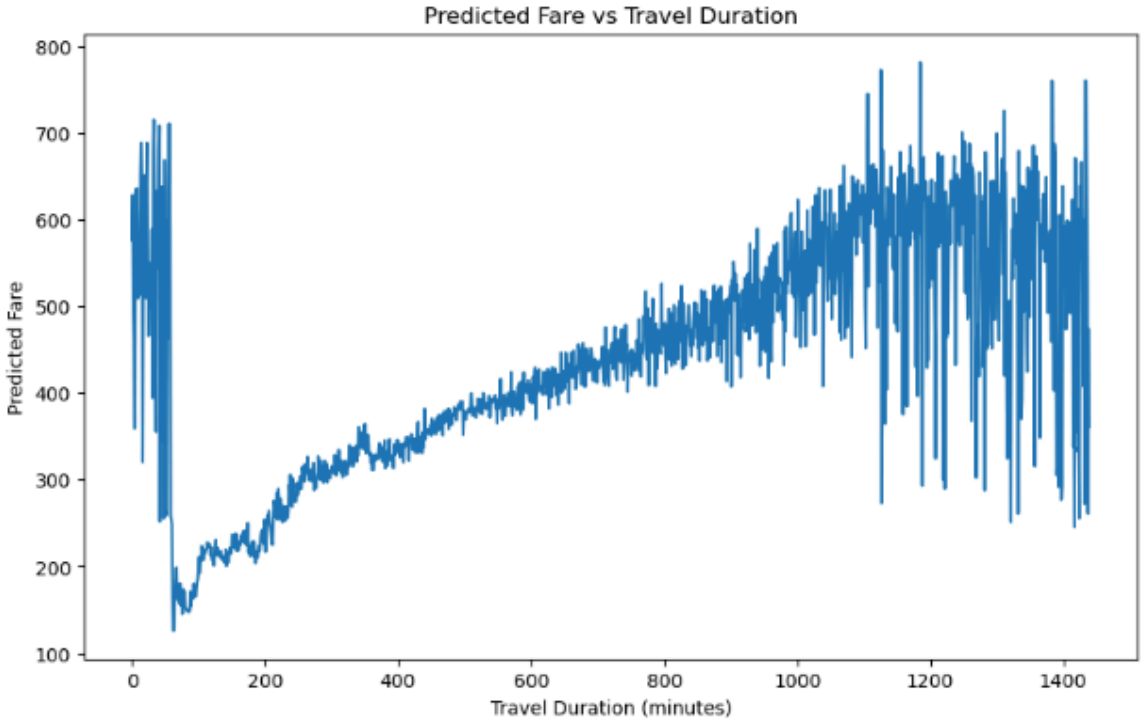

- This scatter plot shows the relationship between the predicted fare price and the total travel distance. Given the graph we can see that there is a positive correlation between predicted fare and total travel distance, we see that as total travel distance in miles increases, the predicted fare price also increases. Although there are some outliers, there is an overall weak positive correlation between predicted fare and total distanced traveled.

# Boxplot: Predicted Fares by Elapsed Days




- This boxplot shows the distribution of the relation between days elapsed and predicted fare price. We can see that a flights with less or 0 elapsed days is less expensive compared to flights with more elapsed days as the average fare price increases as elapsed days increase as well. We also see that there are a greater number of outliers past the 3rd quartile for 0 and 1 elapsed days, which shows that there are other factors that heavily influence the predicted fare price.

## Line Plot: Relationship between Predicted Fares & Travel Duration




- This line plot shows the relationship between predicted fare prices and the flight's travel duration in minutes. This graph shows an increase in predicted fare price as the travel duration increases. This demonstrates that there is a positive correlation between travel duration and fare cost however, we see that past 1000 minutes in travel duration, the predicted price varies greater with more fluctuations. Generally, we can see that longer flights are more expensive than shorter flights.

# Milestone 6

**Summary:** In milestone 6, I summarize the entire process of this project and share my project and code on my GitHub.

To begin with my data acquisition process, I got my Expedia airplane flight price dataset form Kaggle (LINK) and created a Google Cloud storage bucket to house all my storage folders. I begin by creating a VR instance using PySpark to conduct a Exploratory Data Analysis on my data which helped my identify my key columns and eliminate any null/missing values from my dataset, then I developed a cleaning notebook to handle the removal of any missing data and unnecessary columns. Afterwards going towards my my feature engineering and modeling part, I normalize the data and feature engineering, as well as training and testing it, using the random forest regression and training 80% of my data and testing the remaining 20%. The processed data and created models are then stored into their folders on my GCS bucket. Lastly, for my data visualization step, I use libraries like Matplotlib and Seaborn to create 4 visualizations to showcase my dataset and its predictions.

To conclude my project from the final results, we can predict that on average the predicted fare price of flights increase as factors such as travel duration (minutes), Elapsed Days, and Total travel distance increase. Although there may be other factors that can heavy influence the price of the flight fare that I have not included in my as part of my key attributes, we can see that generally, the factors I used contribute to a more expensive flight fare as longer duration travel and distance equates to more factors like fuel usage of the aircraft that customers pay for.

# Appendix A Code for data acquision

**Getting started, setting up the virtual machine**

1. mkdir .kaggle
2. mv kaggle.json .kaggle/
3. chmod 600 .kaggle/kaggle.json
4. ls -la .kaggle
5. sudo apt -y install zip
6. sudo apt -y install python3-pip python3.11-venv
7. python3 -m venv pythondev
8. Pythondev
9. cd pythondev
10. source bin/activate
11. pip3 install kaggle
12. kaggle datasets lis

**Kaggle API TOKEN**

1. kaggle datasets download -d dilwong/flightprices
2. ls -la
3. unzip flightprices.zip

**Request access to create a bucket**

1. gcloud storage buckets create gs://my-bucket --project=airplane-flight-prices --default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access
2. (ERROR)
3. gcloud auth login
4. gcloud storage buckets create gs://expedia-flight-prices --project=airplane-flight-prices --default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access (REPEAT)
5. gcloud storage cp itineraries.csv gs://expedia-flight-prices/landing/itineraries.csv

**Checking if the files have been stored on GCS**

1. gcloud storage ls gs://expedia-flight-prices/

# Appendix B Code for EDA

**EDA CODE**

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, min, max, avg, stddev, when
from pyspark.sql.types import IntegerType, FloatType, DoubleType, TimestampType

# Initialize Spark session
spark = SparkSession.builder \
    .appName("Expedia Flight Prices EDA") \
    .getOrCreate()

# Define EDA function using PySpark DataFrame
def perform_EDA(df, filename: str):
    # Number of observations (records)
    num_observations = df.count()
    print(f"{filename} - Number of observations: {num_observations}")

    # List of variables (columns)
    print(f"{filename} - List of variables (columns):")
    print(df.columns)

    # Count number of missing fields
    null_counts = df.select([count(when(col(c).isNull() | (col(c).cast("string") == "NaN"),
c)).alias(c) for c in df.columns])
    print(f"{filename} - Number of missing fields:")
    for row in null_counts.collect():
        for col_name, count_value in zip(df.columns, row):
            print(f"{col_name}: {count_value}")

    # Min, max, avg, and stddev for numeric variables
    numeric_cols = [f.name for f in df.schema.fields if isinstance(f.dataType, (IntegerType,
FloatType, DoubleType))]
    for col_name in numeric_cols:
        mean_value = df.select(avg(col(col_name))).first()[0]
        stddev_value = df.select(stddev(col(col_name))).first()[0]
        min_value = df.select(min(col(col_name))).first()[0]
        max_value = df.select(max(col(col_name))).first()[0]
        print(f"{filename} - {col_name}:")
        print(f"  Mean: {mean_value}, StdDev: {stddev_value}, Min: {min_value}, Max:
{max_value}")
```

```
    # Date variable statistics
    date_cols = [f.name for f in df.schema.fields if isinstance(f.dataType, TimestampType)]
    if date_cols:
        for col_name in date_cols:
            min_date = df.select(min(col(col_name))).collect()[0][0]
            max_date = df.select(max(col(col_name))).collect()[0][0]
            print(f"{col_name} - Min Date: {min_date}, Max Date: {max_date}")

# Set up bucket name and landing (you may need to configure access to the CSV through Spark)
source_file_path = "gs://expedia-flight-prices/landing/itineraries.csv"

# Load the CSV file into a Spark DataFrame
df = spark.read.csv(source_file_path, header=True, inferSchema=True)

# Sample 4/10 of the dataset randomly
sampled_df = df.sample(fraction=0.40, seed=42)

# Once sampled, perform EDA on the sampled DataFrame
perform_EDA(sampled_df, "Sampled Data (4/10 of total)")
```

# Graph code

```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert the sampled Spark DataFrame to a Pandas DataFrame
relationship_df = sampled_df.select('baseFare', 'totalFare').toPandas()

# Create the relationship plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='baseFare', y='totalFare', data=relationship_df)
plt.title('Relationship between Base Fare and Total Fare')
plt.xlabel('Base Fare')
plt.ylabel('Total Fare')
plt.tight_layout()
plt.show()
```

# Appendix C Code for data cleaning

**CLEANING CODE**

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, when, avg
from pyspark.sql.types import IntegerType, FloatType, DoubleType

# Initialize Spark session
spark = SparkSession.builder \
    .appName("Expedia Flight Prices Data Cleaning") \
    .getOrCreate()

# Load the raw data from the /landing folder
source_file_path = "gs://expedia-flight-prices/landing/itineraries.csv"
df = spark.read.csv(source_file_path, header=True, inferSchema=True)

# Step 1: Drop unnecessary columns, keeping only the specified ones
columns_to_keep = ['flightDate', 'startingAirport', 'destinationAirport',
                   'travelDuration', 'isBasicEconomy', 'isRefundable',
                   'isNonStop', 'elapsedDays', 'totalTravelDistance',
                   'seatsRemaining']

df = df.select(columns_to_keep)

# Step 2: Handle missing values
# Get a list of numeric columns
numeric_cols = [f.name for f in df.schema.fields if isinstance(f.dataType, (IntegerType,
FloatType, DoubleType))]

# Fill missing values in numeric columns with their average
for col_name in numeric_cols:
    mean_value = df.select(avg(col(col_name))).first()[0]
    df = df.fillna({col_name: mean_value})

# Get a list of categorical columns
categorical_cols = [f.name for f in df.schema.fields if f.dataType not in (IntegerType, FloatType,
DoubleType)]

# Drop rows with missing values in categorical columns
df = df.na.drop(subset=categorical_cols)
```

```
# Step 3: Count missing values for each column in a cleaner format
print("\nNumber of missing fields per column:")
null_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])
for row in null_counts.collect():
    for col_name, count_value in zip(df.columns, row):
        print(f"{col_name}: {count_value}")

# Display the cleaned DataFrame to verify changes
df.show(truncate=False)

# Stop the Spark session if needed
spark.stop()
```

# Appendix D Code for feature engineering and modeling

**Feature Engineering**

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, regexp_extract
from pyspark.ml.feature import StringIndexer, StandardScaler, VectorAssembler
from pyspark.ml import Pipeline

# Initialize Spark session
spark = SparkSession.builder.appName("FeatureEngineering").getOrCreate()

# Load cleaned data
cleaned_data_path = "gs://expedia-flight-prices/Cleaned/itineraries_cleaned.parquet/*"
df = spark.read.parquet(cleaned_data_path)

# Convert `isBasicEconomy` and `isRefundable` from True/False to 0/1
df = df.withColumn("isBasicEconomy", when(col("isBasicEconomy") == True,
1).otherwise(0))
df = df.withColumn("isRefundable", when(col("isRefundable") == True, 1).otherwise(0))

# Extract hours and minutes from the `travelDuration` column
df = df.withColumn("hours", regexp_extract(col("travelDuration"), "PT(\\d+)H",
1).cast("int"))
df = df.withColumn("minutes", regexp_extract(col("travelDuration"), "(\\d+)M",
1).cast("int"))

# Replace nulls with 0 for any missing hours or minutes
df = df.fillna({"hours": 0, "minutes": 0})

# Calculate total travel duration in minutes
df = df.withColumn("travelDurationMinutes", col("hours") * 60 + col("minutes"))

# Drop the temporary columns if no longer needed
df = df.drop("hours", "minutes")

# Define categorical and numeric columns
categorical_columns = ["startingAirport", "destinationAirport"]
numeric_columns = ["travelDurationMinutes", "elapsedDays", "totalTravelDistance"]
```

```
# StringIndexer for categorical features
indexers = [StringIndexer(inputCol=col, outputCol=col + "_index") for col in
categorical_columns]

# VectorAssembler to combine feature columns into a feature vector
assembler = VectorAssembler(
    inputCols=[col + "_index" for col in categorical_columns] + numeric_columns,
    outputCol="features"
)

# StandardScaler to scale the features
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")

# Create a pipeline for feature engineering
feature_pipeline = Pipeline(stages=indexers + [assembler, scaler])

# Transform the data
processed_data = feature_pipeline.fit(df).transform(df)

# Save processed data to the /trusted folder
processed_data.write.parquet("gs://expedia-flight-prices/Trusted/itineraries_processed.par
quet", mode="overwrite")

print("Feature engineering completed and data saved to the /Trusted folder.")
```

**Modeling Code**

```
from pyspark.sql import SparkSession
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import VectorAssembler

# Initialize Spark session
spark = SparkSession.builder.appName("PricePrediction").getOrCreate()

# Load the pre-processed data from the trusted folder
```

```
trusted_data_path = "gs://expedia-flight-prices/Trusted/itineraries_processed.parquet"
df = spark.read.parquet(trusted_data_path)

# Sample 50% of the data (fraction = 0.5)
df = df.sample(fraction=0.3, seed=123)

# Select features and target column
numeric_columns = ["travelDurationMinutes", "elapsedDays", "totalTravelDistance"]
categorical_columns = ["startingAirport_index", "destinationAirport_index"]  # Already indexed columns
target_column = "totalFare"

# If there is an existing "features" column, rename it to avoid conflict
df = df.withColumnRenamed("features", "old_features") if "features" in df.columns else df

# Assemble feature columns into a single feature vector
assembler = VectorAssembler(
    inputCols=categorical_columns + numeric_columns,  # Excluding "scaled_features" for feature importance
    outputCol="features"  # This can be renamed to something else if "features" already exists
)

df = assembler.transform(df)

# Define the Random Forest Regressor model
rf = RandomForestRegressor(featuresCol="features", labelCol=target_column)

# Select the feature columns and target column
features = categorical_columns + numeric_columns  # excluding "scaled_features"

# Split the data into training (80%) and testing (20%) sets
train_data, test_data = df.randomSplit([0.8, 0.2], seed=123)

# Set up cross-validation with hyperparameter tuning
paramGrid = ParamGridBuilder() \
```

```
    .addGrid(rf.numTrees, [10, 20, 30]) \
    .addGrid(rf.maxDepth, [5, 10, 15]) \
    .build()

# Regression evaluators for totalFare
rmse_evaluator = RegressionEvaluator(labelCol=target_column,
predictionCol="prediction", metricName="rmse")
mae_evaluator = RegressionEvaluator(labelCol=target_column,
predictionCol="prediction", metricName="mae")
r2_evaluator = RegressionEvaluator(labelCol=target_column,
predictionCol="prediction", metricName="r2")

cv = CrossValidator(estimator=rf,
                    estimatorParamMaps=paramGrid,
                    evaluator=rmse_evaluator,  # Evaluator for RMSE
                    numFolds=3)

# Train the model using cross-validation
cvModel = cv.fit(train_data)

# Make predictions on the test set
predictions = cvModel.transform(test_data)

# Evaluate the model using RMSE, MAE, and R2
rmse = rmse_evaluator.evaluate(predictions)
mae = mae_evaluator.evaluate(predictions)
r2 = r2_evaluator.evaluate(predictions)

# Print the evaluation metrics
print(f"Root Mean Squared Error (RMSE) for Total Fare = {rmse}")
print(f"Mean Absolute Error (MAE) for Total Fare = {mae}")
print(f"R-Squared (R2) for Total Fare = {r2}")

# Get feature importance from the trained model
rf_model = cvModel.bestModel  # Best model after cross-validation

# Extract feature importances
```

```
feature_importances = rf_model.featureImportances

# Map feature importance to the feature names
feature_names = categorical_columns + numeric_columns
importance_map = dict(zip(feature_names, feature_importances))

# Print the feature importances
print("Feature Importances:")
for feature, importance in importance_map.items():
    print(f"{feature}: {importance}")

# Save the trained model to the /models folder
model_output_path = "gs://expedia-flight-prices/Models/random_forest_model"
cvModel.bestModel.write().overwrite().save(model_output_path)
print(f"Trained model saved to {model_output_path}")
```

## Modeling Code Predictions

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import RandomForestRegressionModel  # Correct class for loading the model

# Initialize Spark session
spark = SparkSession.builder.appName("PricePrediction").getOrCreate()

# Load the pre-processed data from the trusted folder
trusted_data_path = "gs://expedia-flight-prices/Trusted/itineraries_processed.parquet"
df = spark.read.parquet(trusted_data_path)

# Sample 30% of the data for prediction (use the whole dataset or a specific subset)
df = df.sample(fraction=0.3, seed=123)

# Select features and target column
numeric_columns = ["travelDurationMinutes", "elapsedDays", "totalTravelDistance"]
categorical_columns = ["startingAirport_index", "destinationAirport_index"]  # Already indexed columns
```

```
# If there is an existing "features" column, rename it to avoid conflict
if "features" in df.columns:
    df = df.withColumnRenamed("features", "old_features")

# Assemble feature columns into a single feature vector
assembler = VectorAssembler(
    inputCols=categorical_columns + numeric_columns,  # Excluding "scaled_features"
for feature importance
    outputCol="features"  # This can be renamed to something else if "features" already
exists
)

df = assembler.transform(df)

# Load the pre-trained Random Forest model from the /Models folder
model_output_path = "gs://expedia-flight-prices/Models/random_forest_model"
loaded_model = RandomForestRegressionModel.load(model_output_path)

# Make predictions on the dataset
predictions = loaded_model.transform(df)

# Show the first few predictions in the output (print the predictions)
predictions.select("prediction", *numeric_columns).show(10, truncate=False)

# Save predictions to the Models folder with the name "predictions.parquet" and
overwrite if exists
predictions_output_path = "gs://expedia-flight-prices/Models/predictions.parquet"
predictions.write.mode("overwrite").parquet(predictions_output_path)
print(f"Predictions saved to {predictions_output_path}")
```

# Appendix E Code for visualization

**Data Visualization Code**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName("VisualizationPipeline").getOrCreate()

# Load predictions data
predictions_output_path = "gs://expedia-flight-prices/Models/predictions.parquet"
predictions_df = spark.read.parquet(predictions_output_path)

# Convert Spark DataFrame to Pandas for visualizations
predictions_pd = predictions_df.select(
    "prediction", "travelDurationMinutes", "elapsedDays", "totalTravelDistance"
).sample(fraction=0.1, seed=123).toPandas()

# Visualization 1: Distribution of Predicted Fares
plt.figure(figsize=(10, 6))
sns.histplot(predictions_pd["prediction"], kde=True, bins=30, color="blue")
plt.title("Distribution of Predicted Fares")
plt.xlabel("Predicted Fare")
plt.ylabel("Frequency")
plt.show()

# Visualization 2: Scatter Plot of Fare Prediction vs Travel Distance
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=predictions_pd["totalTravelDistance"],
    y=predictions_pd["prediction"],
    alpha=0.6
)
plt.title("Predicted Fare vs Total Travel Distance")
plt.xlabel("Total Travel Distance (miles)")
plt.ylabel("Predicted Fare")
```

```
plt.show()

# Visualization 3: Boxplot of Predicted Fares by Elapsed Days
plt.figure(figsize=(10, 6))
sns.boxplot(x=predictions_pd["elapsedDays"], y=predictions_pd["prediction"])
plt.title("Boxplot of Predicted Fares by Elapsed Days")
plt.xlabel("Elapsed Days")
plt.ylabel("Predicted Fare")
plt.show()

# Visualization 4: Relationship between Travel Duration and Predicted Fares
plt.figure(figsize=(10, 6))
sns.lineplot(
    x=predictions_pd["travelDurationMinutes"],
    y=predictions_pd["prediction"],
    ci=None
)
plt.title("Predicted Fare vs Travel Duration")
plt.xlabel("Travel Duration (minutes)")
plt.ylabel("Predicted Fare")
plt.show()

# Print summary statistics
print("Summary Statistics:")
print(predictions_pd[["prediction", "travelDurationMinutes",
"totalTravelDistance"]].describe())
```