Cover Page

John Sy
CIS 4130 Project
johnpatrick.sy@baruchmail.cuny.edu
mr.johnpatricksy@gmail.com

Project Link: https://github.com/jpcsy15/Flight-Prices-Predictions-Project

Milestone #1 (Proposal):

https://www.kaggle.com/datasets/dilwong/flightprices

Many people have or have had aspirations to travel somewhere, whether it's to visit their home country or to simply have a vacation. For this project, I would like to conduct it based on a database consisting of flight prices between April 16th, 2022 and November 5th, 2022. The key columns consist of legID, searchDate (when the record was obtained from Expedia), flightDate, starting/destinationAirport, fareBasisCode, travelDuration, elapsedDays, isBasicEconomy, isRefundable, isNonStop, baseFare (without taxes), totalFare, seatsRemaining, totalTravelDistance, and other additional information.

Since we are limited to 8 months within 2022, I predict that flights that are around the summertime (fiightDate, June-August) will be the most expensive (totalFare). To demonstrate this trend, I will be producing a linear regression model.

Milestone 2 (Data Acquisition):

How did you download the dataset via API?

1. Install the necessary packages needed

2. Download your Kaggle API token and upload the file (kaggle.json) to your Linux VM in Google Cloud Storage

3. Setup your credentials using this code and follow instructions
gcloud auth application-default login

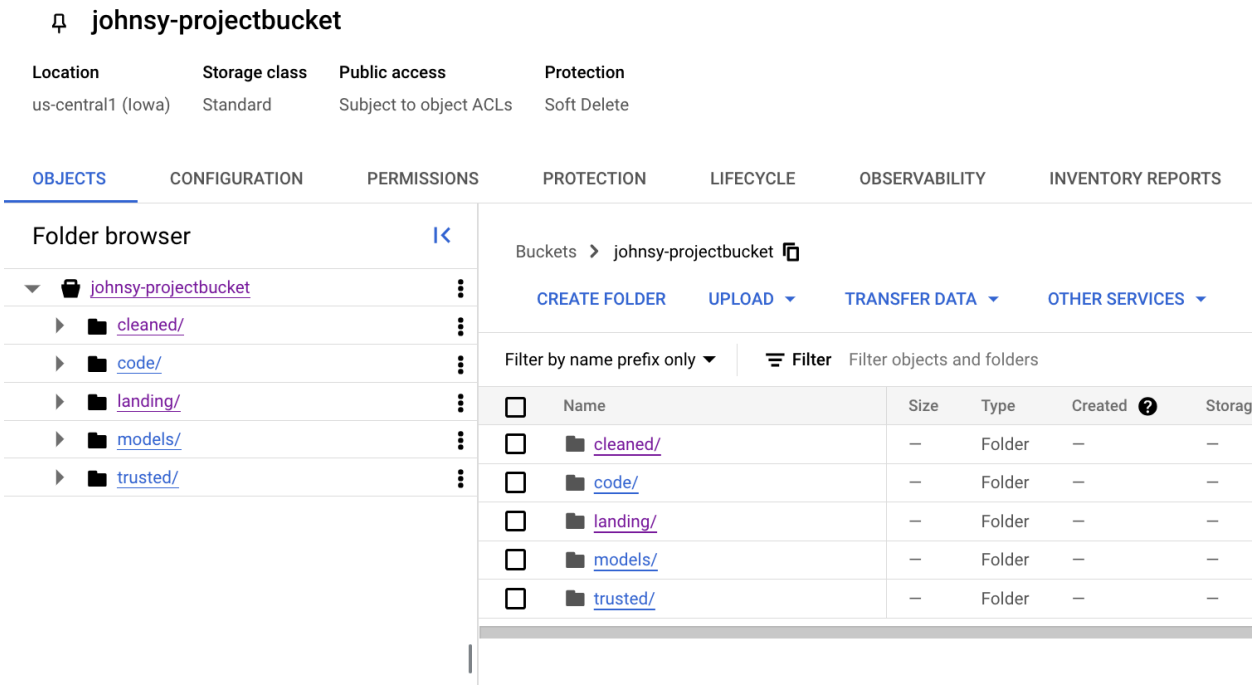
4. Download the dataset using Kaggle onto Linux VM
kaggle datasets download -d dilwong/flightprices

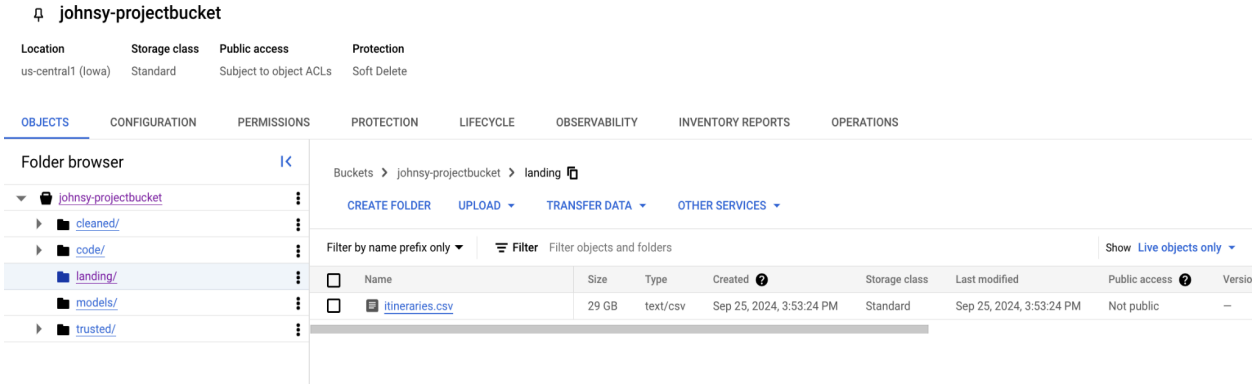
5. Unzip the dataset
unzip -l flightprices.zip
unzip flightprices.zip itineraries.csv

6. Upload file to bucket in Google Cloud Storage
python3 -m venv pythondev
cd pythondev; source bin/activate

```
import pandas as pd
from io import StringIO
from google.cloud import storage
import requests

storage_client = storage.Client()
bucket = storage_client.get_bucket('xxx’)
blob = bucket.blob("itineraries.csv")
blob.upload_from_filename("itineraries.csv")
```


johnsy-projectbucket
Location
us-central1 (Iowa)
Storage class
Standard
Public access
Subject to object ACLs
Protection
Soft Delete
OBJECTS
CONFIGURATION
PERMISSIONS
PROTECTION
LIFECYCLE
OBSERVABILITY
INVENTORY REPORTS
Folder browser
johnsy-projectbucket
cleaned/
code/
landing/
models/
trusted/
Buckets
johnsy-projectbucket
CREATE FOLDER
UPLOAD
TRANSFER DATA
OTHER SERVICES
Filter by name prefix only
Filter
Filter objects and folders
Name
Size
Type
Created
Storag
cleaned/ – Folder – –
code/ – Folder – –
landing/ – Folder – –
models/ – Folder – –
trusted/ – Folder – –


johnsy-projectbucket
Location
us-central1 (Iowa)
Storage class
Standard
Public access
Subject to object ACLs
Protection
Soft Delete
OBJECTS
CONFIGURATION
PERMISSIONS
PROTECTION
LIFECYCLE
OBSERVABILITY
INVENTORY REPORTS
OPERATIONS
Folder browser
johnsy-projectbucket
cleaned/
code/
landing/
models/
trusted/
Buckets
johnsy-projectbucket
landing
CREATE FOLDER
UPLOAD
TRANSFER DATA
OTHER SERVICES
Filter by name prefix only
Filter
Filter objects and folders
Show
Live objects only
Name
Size
Type
Created
Storage class
Last modified
Public access
Versio
itineraries.csv
29 GB
text/csv
Sep 25, 2024, 3:53:24 PM
Standard
Sep 25, 2024, 3:53:24 PM
Not public

Milestone 3 (Exploratory Data Analysis and Data Cleaning):

Total records: 82138753
Unique records: 82138753
Duplicate records: 0

List of Variables: ['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']

Number of Missing Fields:
totalTravelDistance: 6094532 missing
segmentsEquipmentDescription: 1557592 missing
(Every other variable has 0 missing values.)

Key Stats for Base Fare (baseFare):
Average: $292.66
Standard Deviation: $183.19
Minimum: $0.01
Maximum: $7,662.33

Key Stats for Total Fare (totalFare):
Average: $340.39
Standard Deviation: $196.03
Minimum: $19.59
Maximum: $8,260.61

Key Stats for Seats Remaining (seatsRemaining):
Average: 5.98
Standard Deviation: 2.88
Minimum: 0
Maximum: 10

Key Stats for Total Travel Distance (only using flights that have data) (totalTravelDistance):
Average: 1609.9 miles
Standard Deviation: 857.3 miles
Minimum: 89 miles
Maximum: 7,252 miles

Key Stats for Flight Duration (in seconds):
Average: 10,868 seconds (181.13 minutes or a little over 3 hours)
Standard Deviation: 5,085 seconds (84.75 minutes)
Minimum: 1,003 seconds (16.7 minutes)
Maximum: 9,960 seconds (166 minutes or a little over 2 hours)

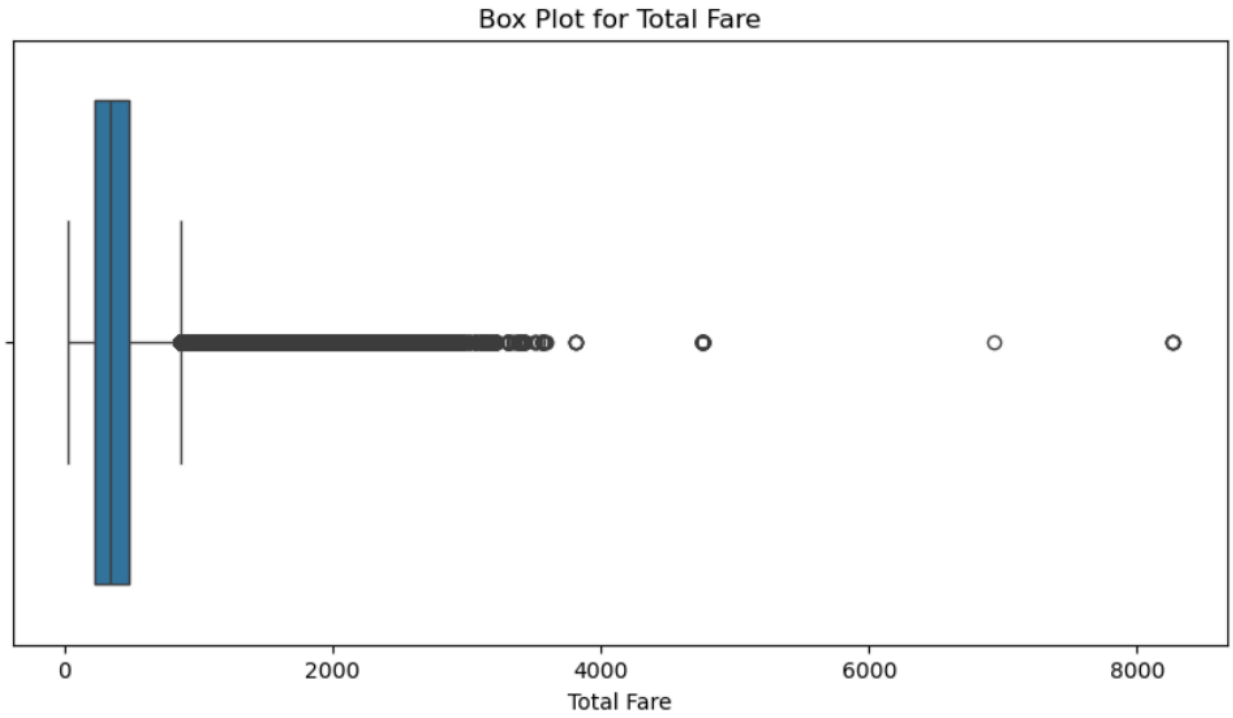
```
 |-- legId: string (nullable = true)
 |-- searchDate: date (nullable = true)
 |-- flightDate: date (nullable = true)
 |-- startingAirport: string (nullable = true)
 |-- destinationAirport: string (nullable = true)
 |-- fareBasisCode: string (nullable = true)
 |-- travelDuration: string (nullable = true)
 |-- elapsedDays: integer (nullable = true)
 |-- isBasicEconomy: boolean (nullable = true)
 |-- isRefundable: boolean (nullable = true)
 |-- isNonStop: boolean (nullable = true)
 |-- baseFare: double (nullable = true)
 |-- totalFare: double (nullable = true)
 |-- seatsRemaining: integer (nullable = true)
 |-- totalTravelDistance: integer (nullable = true)
 |-- segmentsDepartureTimeEpochSeconds: string (nullable = true)
 |-- segmentsDepartureTimeRaw: string (nullable = true)
 |-- segmentsArrivalTimeEpochSeconds: string (nullable = true)
 |-- segmentsArrivalTimeRaw: string (nullable = true)
 |-- segmentsArrivalAirportCode: string (nullable = true)
 |-- segmentsDepartureAirportCode: string (nullable = true)
 |-- segmentsAirlineName: string (nullable = true)
 |-- segmentsAirlineCode: string (nullable = true)
 |-- segmentsEquipmentDescription: string (nullable = true)
 |-- segmentsDurationInSeconds: string (nullable = true)
 |-- segmentsDistance: string (nullable = true)
 |-- segmentsCabinCode: string (nullable = true)
```

startingAirport	count
LAX	8073281
LGA	5919323
BOS	5883876
SFO	5706482
DFW	5674959
ORD	5503476
CLT	5494510
ATL	5312028

|MIA |4930213|
|PHL |4726187|
+---------------+-------+

+------------------+-------+
|destinationAirport|count |
+------------------+-------+
LAX	8006721
LGA	6093450
DFW	5957280
BOS	5801538
ORD	5717699
SFO	5586204
CLT	5411448
ATL	5211169
MIA	5103390
PHL	4703822
+------------------+-------+

Min searchDate: 04-16-2022
Max searchDate: 10-05-2022
Min flightDate: 04-17-2022
Max flightDate: 11-19-2022



With this milestone, I was able to find some key statistics that'll help me to analyze the dataset. Besides finding the min/max/avg/stdev for the numeric variables/columns, I checked to see if any of them had any missing (null) values, and I uncovered that totalTravelDistance and segmentsEquipmentDescription had an abundance of missing values, meanwhile, every other column had 0 missing values. With this in mind, I decided to remove the two variables as I converted my CSV file into a Parquet file. Additionally, I decided to remove were elapsedDays and isRefundable because I felt that it wouldn't be necessary to have for later milestones. As for challenges, one I can face could be the various data types that are in the dataset, which could make it difficult when transforming the data. Additionally as shown on the Box Plot chart above, there were a number of outliers that could impact my data analysis, with several high prices (totalFare) for flights. During this milestone, I unfortunately ran into some issues, such as the kernel repeatedly dying due to the lack of resources I initially had. On top of this, it took a long period of time to transform the data due to how large the dataset was, so data cleaning was very much needed.

Milestone 4 (Feature Engineering and Modeling):

Results of Prediction and Linear Regression Model for Total Flight Prices (totalFare)
Root Mean Squared Error (RMSE): 13.925931648252083
R-squared (R2): 0.9949473527360314
Coefficients:
[0.1394584481736591,0.10611827060547191,0.22298244858027386,195.32881586001366,-7.273361814674778]
Intercept: 40.58924048844299

Results of Predictions for Total Flight Prices (totalFare)

legId	totalFare	prediction
0000606b4f492009f...	567.2	559.6954788607804
0000a1ddcbc75a555...	466.61	464.6086603509158
0000b1d0849d0a57b...	266.6	280.4765479245457
0000fa8b4a7d87760...	108.6	114.39757328565281
0000fa8b4a7d87760...	108.6	114.39757328565281
0000fcd4adff61fa5...	315.6	331.7192792625999
000175f2039cf317f...	710.6	705.5019931448012
00017c52fca6543ff...	352.6	366.08858829463315
00017fa2e84336ad9...	488.6	491.09724296781064
000247b8444e2d793...	509.1	508.0823490513128
00025c179c798575c...	367.6	381.20034317018377
000280e4343980a89...	172.6	182.35691863171834
0002d1d80a375297f...	445.6	438.77688739045846
00033508a850faff8...	138.6	157.66905094529858
00033c1feeb14752e...	326.6	333.55860523999826
00033c1feeb14752e...	326.6	333.55860523999826
00035ba00966c1407...	627.1	630.0825997299708
000389eb2b97b80cf...	187.6	197.9769265398584
000389eb2b97b80cf...	190.6	190.8511629166315
0003ab32e8f358cf6...	477.61	481.0194280877913
0004177dc56cdfbe0...	591.6	603.1732785376439
0004549149c6286c5...	758.1	763.257164259293
...		
000a0d67141d4dd77...	620.4	616.6671104035037

only showing top 50 rows

Results of Predicting Difference of the date a flight is booked vs the date of a flight (searchDate vs flightDate)
Root Mean Squared Error (RMSE): 16.13041879435989
R-squared (R2): 0.023111568956743156

Model Coefficients:
[0.022974865447223478,0.03164339973320451,-1.1453038676327105,2.2714283482350037]
Model Intercept: 23.65792408220409

Results of Predictions for daysUntilFlight (flightDate - searchDate)

legId	daysUntilFlight	prediction
0000127cc14ee3561...	11	27.628601128505906
000019591cfd2d4eb...	37	27.9509717732128
00002c6c6336d555f...	38	23.543878344411457
00004b25b95582a5f...	9	28.91427352507564
0000e2a7d019d7f5f...	48	27.98461522206546
000104a297929150b...	36	20.715816411331257
000157009cc2e4560...	15	30.10933328987609
000171746d9c469d1...	18	28.31137941563287
0001a56fb4d60cc3e...	47	30.059857455657884
0001a56fb4d60cc3e...	11	29.926062887271904
0001a588df49bdc64...	18	25.14477072624059
0001ebcbce92de0b6...	32	26.477006064678214
000229a9485e5ee01...	33	27.450066846414767
00023404a34e37b88...	6	24.763502625454677
0002d66907bb3436e...	10	23.48884009508031
0002f7e22022105e5...	4	27.113059061966105
000348acdf42e8a9c...	6	30.127908128245153
00042dff7a95e7260...	41	21.341197897477727
000497edad22f1c63...	11	26.329432791321267
0004d8181fae1af41...	58	28.1595784617096
0004f8e8dee4b4df9...	19	22.96962584326227
0004fbb724347b531...	38	26.437256105273157
...		
000e308fce4ee8566...	20	22.891869043684704

only showing top 50 rows

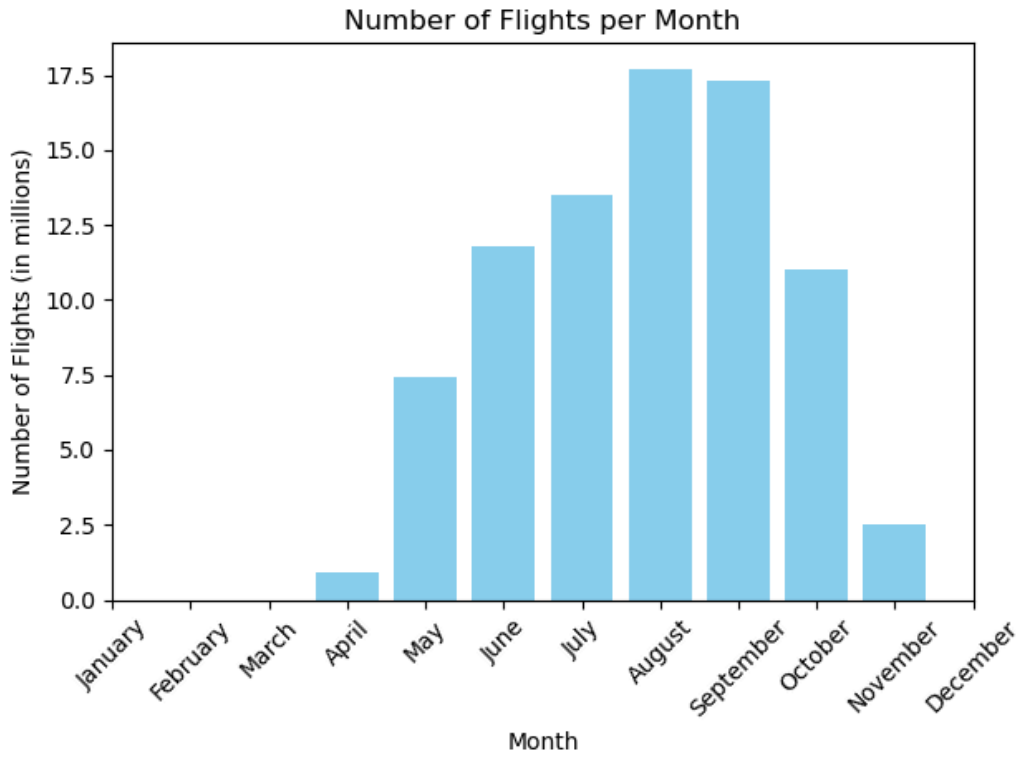
In this portion of the project, I was able to perform feature engineering and modeling on my cleaned dataset. To start off, I created a Dataproc cluster via Google Cloud Platform in order to conduct some analysis, such as using a regression model on top of making predictions on how much the total prices would be for each flight (totalFare), along with predicting what would the difference be for the date of when a flight is booked (searchDate) vs the date of when the actual flight is (flightDate).

As for the difference between dates, the regression model provided a model intercept of about 23.66, indicating that when all the predictor variables in the model are equal to 0, there's a mean value of about 24 days between when a flight is booked vs the actual flight date. As for the actual model itself, it didn't do the best at achieving an accurate prediction, with an R^2 of about 0.023 (extremely low) and an RMSE of about 16 days (large discrepancy for predicting time).

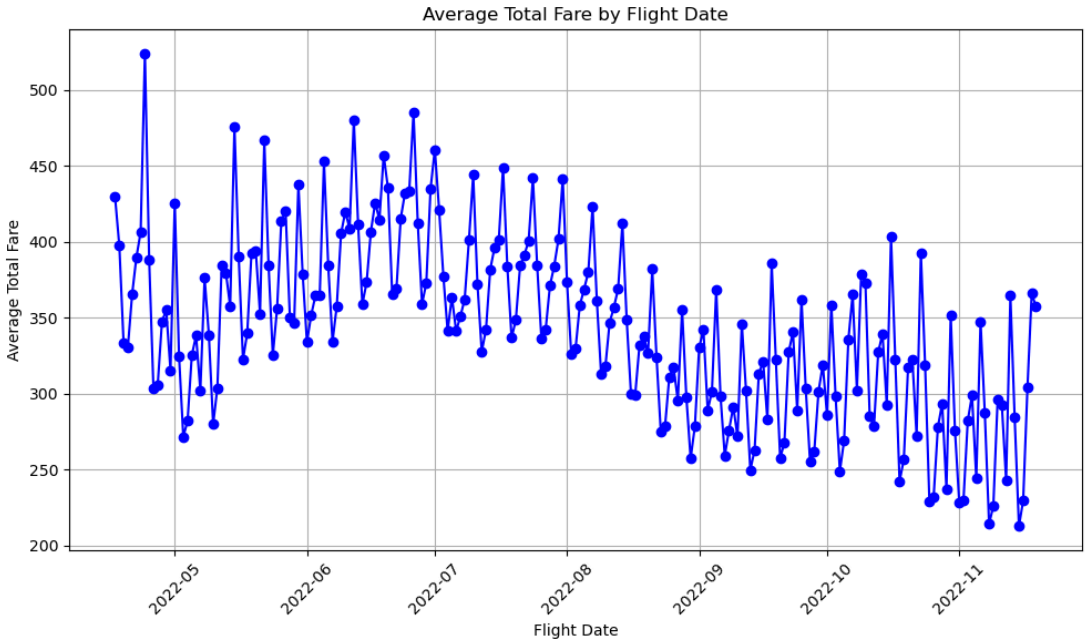
When predicting the flight prices (totalFare), it was able to achieve an R^2 of about 0.995 along with an RMSE of about 13.93. These are really good indicators, especially in comparison to the last one. The 13.93 RSME indicates that there was only a decent difference between the totalFare and the predictions, which is significant when you take into account the number of outliers, of a few extremely expensive prices, within this dataset for the totals of flight prices.

Throughout this milestone, I experienced several challenges. For example, I had to remake my parquet file for some reason since I wasn't able to access my original parquet file. It could've somehow got corrupted or deleted. Additionally, my kernel repeatedly kept dying, especially when I was experimenting around with the cleaned data, so I had to repartition my cluster numerous times while also mostly using proportional sample sizes. Moreover, I kept receiving errors at times when trying to create my predictions models. For example when playing around with the code, I wanted to keep experimenting with predicting prices, however, I tried changing the datatype of the column to a double or integer from a string, but the syntax wouldn't interpret them at all regardless of what I tried to do. All in all, it made me realize the importance of cleaning the data to make creating and finding data statistics/analysis much more simpler.

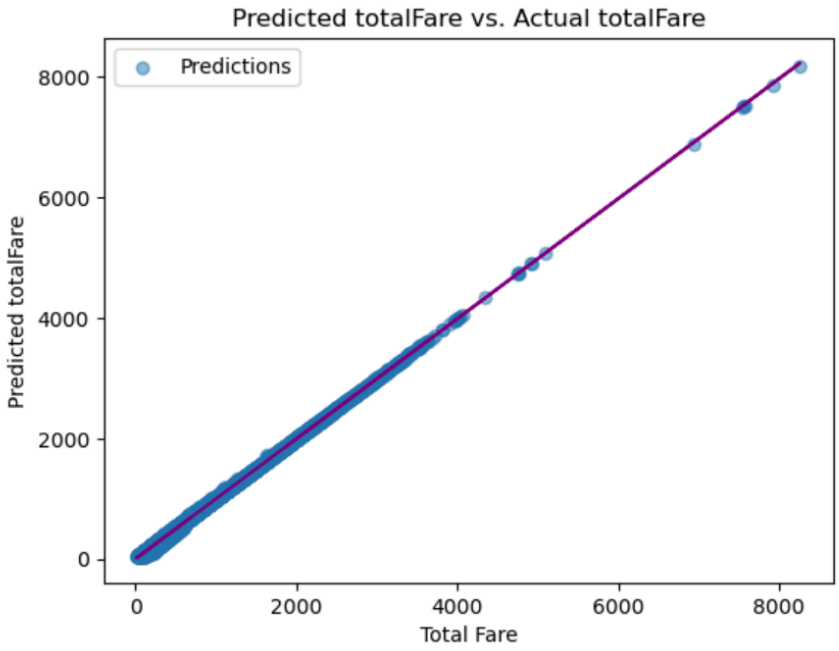
Milestone 5 (Data Visualization):



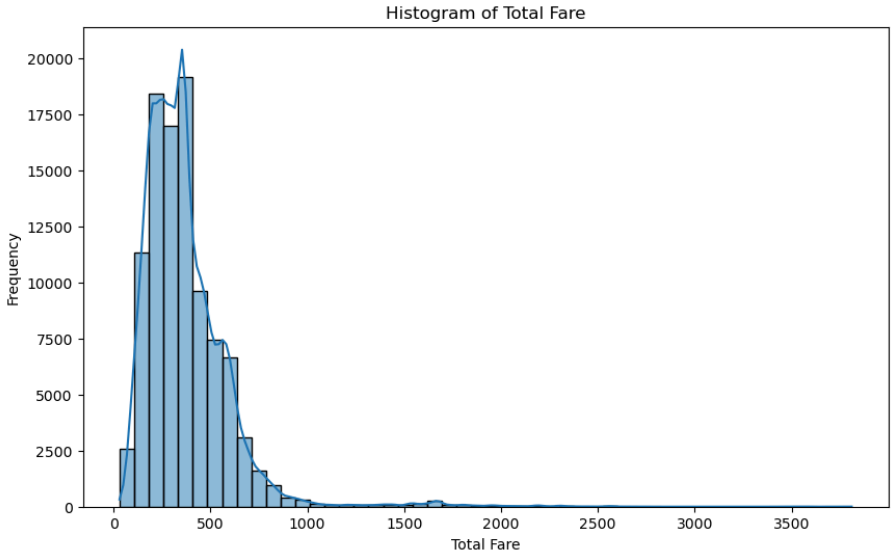
This chart essentially displays the number of flights that occur per month. Based on the output it shows, July, August, and September seemed to have the most number of flights. Initially, my prediction was that June, July, and August would have the most since June is when summer break usually starts for students, and everyone wants to travel during those times, but September had the second most. I additionally thought that March and November would have at least more flights due to spring break and Thanksgiving, respectively. However, that wasn't the case.



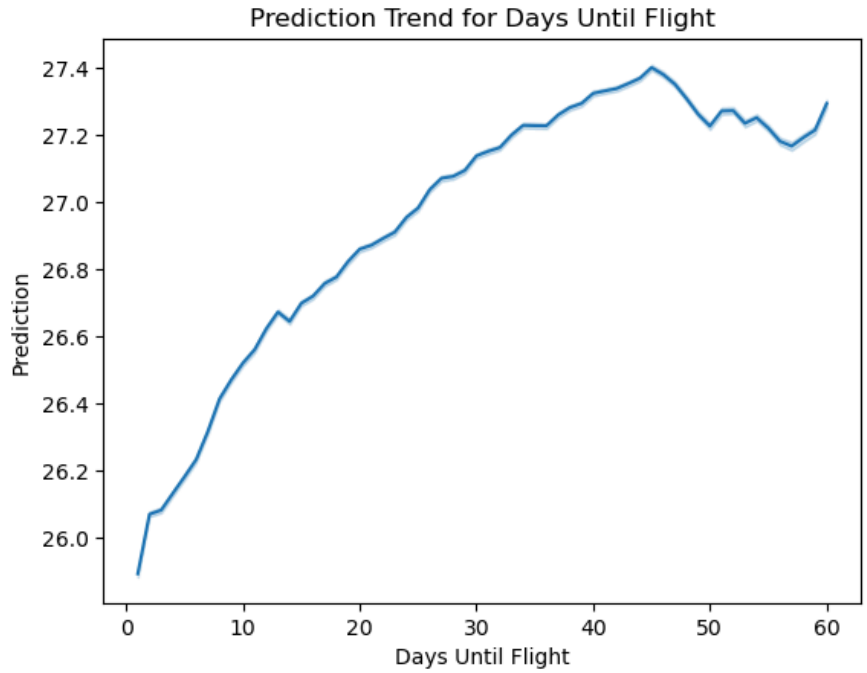
I created this chart, which essentially displays the average total fare for each month. Based on the output, I was surprised because I originally thought that flights in the summer would be significantly more expensive, however, it seems to be just marginally more expensive.



I created this chart to compare how the predictions for totalFare would look compared to the actual totalFare and they seem to be pretty accurate. What surprised me is that it was able to create an accurate prediction for the outliers, or the more expensive flights as well.



This chart displays the total frequency of totalFares shown in the dataset. As expected, the most common prices for totalFares would be from about $200-400. It also was able to show some of the few outliers, showing the more expensive flights.



This is essentially a line chart essentially displaying the predicted daysUntilFlight and comparing it with the actual daysUntilFlight on the x-axis. One thing I noticed is that as the x-axis goes up, the more inaccurate the prediction seems to get for some reason.

Milestone 6 (Summary and Conclusions):

With this project, I was able to create a pipeline analyzing a dataset based on the prices of flights from April to November 2022. I originally predicted that June through August would have the most expensive flights. Throughout the various milestones, I was able to conduct data cleaning, exploratory data analysis (EDA), and feature engineering in order to inevitably create a Machine Learning model to create predictions on totalFare and daysUntilFlight through a linear regression model.

In order to achieve this, a few columns that seemed unnecessary were removed, whether it had many null values or it just wasn't needed at all. Next, the dataset was converted into a parquet file, which made it more efficient to read/analyze the data. As a result of the data cleaning, I was able to conduct EDA which enabled me to to create several visualizations along with the predictions models. Based on the "Average Total Fare by Flight Date" chart, it didn't really justify my original theory of overall expensive flights during the summer time compared to other months, since I thought it would be a much more extensive difference. Although the predictions model for daysUntilFlight didn't provide the best results, totalFare was able to provide very accurate predictions, seeming to demonstrate a strong relationship with the baseFare and seatsRemaining columns.

There were definitely improvements that could've been done to improve this project, such as using CrossValidator to validate the models, and removing the outliers from the dataset to potentially improve the predictions models that were created. On top of this, there were several challenges that I faced, such as having to repartition the data cluster numerous times due to how demanding the data was, the frequent long downtime with processing code, and having to clean the dataset several times throughout the project.

Overall, this project was a great and fun learning experience for me. It certainly helped to reinforce topics that were went over through class, such as the use of ML with PySpark, GCP, and data visualization.

Appendix B (EDA Source Code)

```
#How I found the missing fields for each column
missing_values = df.select([count(when(col(c).isNull(), c)).alias(c) for c in
df.columns]).collect()[0].asDict()
print(“Number of missing fields in each column:”)
for column, count in missing_values.items():
    print(f"{column}: {count}")


#The 10 Most Common Starting/Destination Airports
spark = SparkSession.builder.appName("MostCommonAirports").getOrCreate()
common_starting_airports =
df_cleaned.groupBy("startingAirport").count().orderBy(desc("count"))
common_starting_airports.show(10, truncate=False)
common_destination_airports =
df_cleaned.groupBy("destinationAirport").count().orderBy(desc("count"))
common_destination_airports.show(10, truncate=False)


#Finding the min/max dates for flightDate
df_spark = spark.read.parquet("xxx")
min_max_dates = df_spark.agg(
    F.min("flightDate").alias("min_flightDate"),
    F.max("flightDate").alias("max_flightDate")
)
min_max_dates.show()


#Finding the min/max dates for searchDate
df_spark = spark.read.parquet("xxx")
min_max_dates = df_spark.agg(
    F.min("searchDate").alias("min_searchDate"),
    F.max("searchDate").alias("max_searchDate")
)
min_max_dates.show()


#Box Plot using Total Fare for Outliers
df_totalFare_spark = df_spark.select("totalFare").limit(10000000)
df_totalFare = df_totalFare_spark.toPandas()
plt.figure(figsize=(10, 5))
sns.boxplot(x=df_totalFare[“totalFare”])
plt.title(“Box Plot for Total Fare”)
```

```
plt.xlabel(“Total Fare”)
plt.show()
```

Appendix C (Data Cleaning Code)

```
#Dropping totalTravelDistance and segmentsEquipmentDescription Columns and Converting From CSV to Parquet File
columns_to_drop = ["totalTravelDistance", "segmentsEquipmentDescription"]
df_cleaned = df_original.drop(*columns_to_drop)
df_cleaned.write.csv("xxx", header=True)
#Used coalesce here


#Dropping segmentsAirlineCode, segmentsCabinCode, isRefundable
df_cleaned = df_check.drop("segmentsAirlineCode", "segmentsCabinCode", "isRefundable")
updated_parquet_output_path = "xxx"
df_cleaned.write.mode("overwrite").parquet(updated_parquet_output_path)
```

Appendix D (Feature Engineering and Modeling Code)

```
#Packages that I used for Milestone 4
from pyspark.sql.types import IntegerType, DoubleType
from pyspark.sql.functions import col, datediff
from pyspark.ml.feature import StringIndexer, VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator


#Creating Predictions Model for totalFare
#Dropping null values just in case
df = df.dropna(subset=["totalFare", "baseFare", "seatsRemaining", "startingAirport", "destinationAirport", "segmentsAirlineCode"])

categorical_cols = ["startingAirport", "destinationAirport", "segmentsAirlineCode"]
numeric_cols = ["baseFare", "seatsRemaining"]
target_col = "totalFare"

indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_index") for col in categorical_cols]
assembler = VectorAssembler(inputCols=numeric_cols, outputCol="numeric_features")
scaler = StandardScaler(inputCol="numeric_features", outputCol="scaled_features")
all_features = [f"{col}_index" for col in categorical_cols] + ["scaled_features"]
```

```
final_assembler = VectorAssembler(inputCols=all_features, outputCol="features")
lr = LinearRegression(featuresCol="features", labelCol=target_col, predictionCol="prediction")
pipeline = Pipeline(stages=indexers + [assembler, scaler, final_assembler, lr])

#Using a Sample Size
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)
pipeline_model = pipeline.fit(train_data)
predictions = pipeline_model.transform(test_data)
evaluator_rmse = RegressionEvaluator(labelCol=target_col, predictionCol="prediction",
metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol=target_col, predictionCol="prediction",
metricName="r2")

#Finding key statistics
rmse = evaluator_rmse.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")


#Getting Predictions for totalFare
predictions_df = spark.read.parquet(predictions_path)
predictions_df.show(50)


#Finding Key Statistics from Predictions of totalFare, Accessing Linear Regression Model
predictions_df = spark.read.parquet(predictions_path)
evaluator_rmse = RegressionEvaluator(labelCol="totalFare", predictionCol="prediction",
metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol="totalFare", predictionCol="prediction",
metricName="r2")

#Calculating RMSE and R²
rmse = evaluator_rmse.evaluate(predictions_df)
r2 = evaluator_r2.evaluate(predictions_df)
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
lr_model = loaded_model.stages[-1]
print("Coefficients:", lr_model.coefficients)
print("Intercept:", lr_model.intercept)


#Predicting Difference of searchDate & flightDate
df = df.withColumn("daysUntilFlight", datediff(col("flightDate"), col("searchDate")))
```

```
df = df.filter(col("daysUntilFlight") > 0)
df = df.withColumn("baseFare", col("baseFare").cast(DoubleType()))
df = df.withColumn("seatsRemaining", col("seatsRemaining").cast(IntegerType()))

#Dropping null values
df = df.dropna(subset=["daysUntilFlight", "baseFare", "seatsRemaining", "startingAirport",
"destinationAirport"])

categorical_cols = ["startingAirport", "destinationAirport"]
numeric_cols = ["baseFare", "seatsRemaining"]
target_col = "daysUntilFlight"
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_index") for col in categorical_cols]
assembler = VectorAssembler(inputCols=numeric_cols, outputCol="numeric_features")
scaler = StandardScaler(inputCol="numeric_features", outputCol="scaled_features")
feature_cols = [f"{col}_index" for col in categorical_cols] + ["scaled_features"]
final_assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
lr = LinearRegression(featuresCol="features", labelCol=target_col, predictionCol="prediction")

#Creating pipeline
pipeline = Pipeline(stages=indexers + [assembler, scaler, final_assembler, lr])
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)
pipeline_model = pipeline.fit(train_data)
predictions = pipeline_model.transform(test_data)
evaluator_rmse = RegressionEvaluator(labelCol=target_col, predictionCol="prediction",
metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol=target_col, predictionCol="prediction",
metricName="r2")

#Finding key statistics
rmse = evaluator_rmse.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
#Placed file path here
predictions.select("legId", target_col,
"prediction").write.mode("overwrite").parquet(predictions_path)
pipeline_model.write().overwrite().save(model_path)


#Getting Predictions Model for daysUntilFlight (Difference of searchDate & flightDate)
predictions_df = spark.read.parquet(predictions_path)
predictions_df.show(50)
```

```
#Finding Key Statistics from Predictions of Difference for searchDate and flightDate
loaded_model = PipelineModel.load(model_path)
regression_model = loaded_model.stages[-1]
from pyspark.ml.regression import LinearRegressionModel
if isinstance(regression_model, LinearRegressionModel):
    print("Model Coefficients:", regression_model.coefficients)
    print("Model Intercept:", regression_model.intercept)
else:
    print("Fail!")
```

Appendix E (Data Visualization Code)

```
#Creating Bar Chart to Find Frequency of Flights per Month
plt.bar(flights_per_month_pd["month"], flights_per_month_pd["count"] / 1e6, color="skyblue")
plt.title("Number of Flights per Month")
plt.xlabel("Month")
plt.ylabel("Number of Flights (in millions)")
plt.xticks(range(1, 13), labels=[
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"], rotation=45)
plt.tight_layout()
plt.show()
```

```
#Creating Line Chart for Average Total Fare by Flight Date
df_fare_by_date = df_spark.groupBy("flightDate").agg(F.mean("totalFare").alias("avgTotalFare"))
#Convert to Pandas DataFrame
df_fare_by_date_pd = df_fare_by_date.toPandas()
df_fare_by_date_pd["flightDate"] = pd.to_datetime(df_fare_by_date_pd["flightDate"])

#Sort by flightDate so it’s in chronological order
df_fare_by_date_pd = df_fare_by_date_pd.sort_values("flightDate")
plt.figure(figsize=(10, 6))
plt.plot(df_fare_by_date_pd["flightDate"], df_fare_by_date_pd["avgTotalFare"], marker='o',
linestyle='-', color='b')

#Adding labels and title
plt.xlabel("Flight Date")
plt.ylabel("Average Total Fare")
plt.title("Average Total Fare by Flight Date")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
```

```
plt.show()


#Scatter Plot for Predicted totalFare vs Actual totalFare
agg_data = predictions.groupBy("totalFare").agg(F.avg("prediction").alias("avg_prediction"))
agg_data_pd = agg_data.toPandas()
plt.scatter(agg_data_pd["totalFare"], agg_data_pd["avg_prediction"], alpha=0.5,
label="Predictions")

#Included a trendline here
z = np.polyfit(agg_data_pd["totalFare"], agg_data_pd["avg_prediction"], 1)
p = np.poly1d(z)
plt.plot(agg_data_pd["totalFare"], p(agg_data_pd["totalFare"]), color="purple", linestyle="-")

#Labels and title
plt.title("Average Prediction vs. Total Fare")
plt.xlabel("Total Fare")
plt.ylabel("Average Predicted Fare")
plt.legend()
plt.show()


#Histogram for totalFare
plt.figure(figsize=(10, 6))
#Adding line
sns.histplot(df_totalFare['totalFare'], bins=50, kde=True)
plt.title('Histogram of Total Fare')
plt.xlabel('Total Fare')
plt.ylabel('Frequency')
plt.show()


#Creating a Line Chart for Predictions vs daysUntilFlight
sns.lineplot(data=predict, x="daysUntilFlight", y="prediction")
plt.title("Prediction Trend for Days Until Flight")
plt.xlabel("Days Until Flight")
plt.ylabel("Prediction")
plt.show()
```