CIS 4130

Big Data Technologies

Kelly Huang

Milestone 1 Proposal: a description of the data set and your project's goals
Milestone 2 Data Acquistion: a description of the steps you took to acquire the data
Milestone 3 EDA and Data Cleaning: a description (with graphs) of your EDA findings and Data Cleaning approach
Milestone 4 Feature Engineering and Modeling: a description (with outputs) of the feature engineering and modeling steps.
Milestone 5 Visualization: A description of the different visuzliations with screen shots of the graphs, charts, etc.
Milestone 6 Summary and Conclusions
Appendix A Code for data acquision
Appendix B Code for EDA
Appendix C Code for data cleaning
Appendix D Code for feature engineering and modeling
Appendix E Code for visualization

Milestone 1 Proposal

Describe : This dataset is a CSV file where each row is a purchasable ticket found on Expedia between 2022-04-16 and 2022-10-05, to/from the following airports: ATL, DFW, DEN, ORD, LAX, CLT, MIA, JFK, EWR, SFO, DTW, BOS, PHL, LGA, IAD, OAK.

Link : the AI command : kaggle datasets download -d dilwong/flightprices
Website link: https://www.kaggle.com/datasets/dilwong/flightprices

Column:
- legId: An identifier for the flight.
- searchDate: The date (YYYY-MM-DD) on which this entry was taken from Expedia.
- flightDate: The date (YYYY-MM-DD) of the flight.
- startingAirport: Three-character IATA airport code for the initial location.
- destinationAirport: Three-character IATA airport code for the arrival location.
- fareBasisCode: The fare basis code.
- travelDuration: The travel duration in hours and minutes.
- elapsedDays: The number of elapsed days (usually 0).
- isBasicEconomy: Boolean for whether the ticket is for basic economy.
- isRefundable: Boolean for whether the ticket is refundable.

Plan: I will predict when has the highest percentage of flights by focusing on the ‘flightDate’ column. I will use logistic regression to determine the likelihood of a higher number of flights occurring on different dates. Logistic regression is suitable because it predicts the probability of an event occurring or not occurring.

Milestone 2 Data Acquisition

python3 -m venv pythondev
cd pythondev
source bin/activate
pip3 install kaggle
kaggle datasets download -d dilwong/flightprices # download database
unzip -1 flightprices.zip
unzip flightprices.zip # get itineraries.csv
gcloud storage buckets create gs://my-bigdata-project-kh
--project=root-opus-435315-s4
    default-storage-class=STANDARD --location=us-central1
    -uniform-bucket-level-access
gcloud auth login
gcloud storage buckets create gs://my-bigdata-project-kh
--project=root-opus-435315-s4
    -default-storage-class=STANDARD --location=us-central1
    -uniform-bucket-level-access
Create folder:
gcloud storage cp itineraries.csv gs://my-bigdata-project-kh/cleaned/
gcloud storage cp itineraries.csv gs://my-bigdata-project-kh/trusted/
gcloud storage cp itineraries.csv gs://my-bigdata-project-kh/models/
gcloud storage cp itineraries.csv gs://my-bigdata-project-kh/code/

Milestone 3 EDA and Data Cleaning:

```
import pandas as pd
import pyarrow
from google.cloud import storage
import matplotlib.pyplot as plt
```

Appendix A (Code for Data Acquisition)

```
client = storage.Client()
bucket_name = 'my-bigdata-project-kh'
file_name = 'landing/itineraries.csv'
output_bucket_name = 'my-bigdata-project-kh'
output_path = 'cleaned/'

bucket = client.get_bucket(bucket_name)
blob = bucket.blob(file_name)
blob.download_to_filename('itineraries.csv')
```

Appendix B (code for EDA)

```
chunk_size = 10_000_000
chunk_number = 0

#EDA function
def perform_EDA(df, chunk_label):
    print(f"Number of Observations: {len(df)}")
    print("Columns:", df.columns.tolist())
    print("Data Types:", df.dtypes)
```

```
Number of Observations: 2138753
Columns: ['legId', 'searchDate', 'flightDate', 'startingAirport', 'destinationAirport', 'fareBasisCode', 'travelDuration', 'elapsedDays', 'isBasicEconomy', 'isRefundable', 'isNonStop',
'baseFare', 'totalFare', 'seatsRemaining', 'totalTravelDistance', 'segmentsDepartureTimeEpochSeconds', 'segmentsDepartureTimeRaw', 'segmentsArrivalTimeEpochSeconds', 'segmentsArrivalTim
eRaw', 'segmentsArrivalAirportCode', 'segmentsDepartureAirportCode', 'segmentsAirlineName', 'segmentsAirlineCode', 'segmentsEquipmentDescription', 'segmentsDurationInSeconds', 'segments
Distance', 'segmentsCabinCode']
Data Types: legId                                object
searchDate                           object
flightDate                           object
startingAirport                      object
destinationAirport                   object
fareBasisCode                        object
travelDuration                       object
elapsedDays                           int64
isBasicEconomy                         bool
isRefundable                           bool
isNonStop                              bool
baseFare                            float64
totalFare                           float64
seatsRemaining                        int64
totalTravelDistance                 float64
segmentsDepartureTimeEpochSeconds    object
segmentsDepartureTimeRaw             object
segmentsArrivalTimeEpochSeconds      object
segmentsArrivalTimeRaw               object
segmentsArrivalAirportCode           object
segmentsDepartureAirportCode         object
segmentsAirlineName                  object
segmentsAirlineCode                  object
segmentsEquipmentDescription         object
segmentsDurationInSeconds            object
segmentsDistance                     object
segmentsCabinCode                    object
dtype: object
```

```
# Number of missing
missing_data = df.isna().sum()
print("Missing Data by Column:\n", missing_data)
```

```
Missing Data by Column:
 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                  150191
segmentsDepartureTimeEpochSeconds         0
segmentsDepartureTimeRaw                  0
segmentsArrivalTimeEpochSeconds           0
segmentsArrivalTimeRaw                    0
segmentsArrivalAirportCode                0
segmentsDepartureAirportCode              0
segmentsAirlineName                       0
segmentsAirlineCode                       0
segmentsEquipmentDescription          38447
segmentsDurationInSeconds                 0
segmentsDistance                      21931
segmentsCabinCode                         0
dtype: int64
```

```
# min/max/avg/stdev for all numeric variables
print("Descriptive Statistics:\n", df.describe())
```

```
Descriptive Statistics:
        elapsedDays      baseFare     totalFare  seatsRemaining  totalTravelDistance
count  2.138753e+06  2.138753e+06  2.138753e+06    2.138753e+06         1.988562e+06
mean   1.488545e-01  2.614892e+02  3.067703e+02    6.285963e+00         1.661930e+03
std    3.559687e-01  1.805778e+02  1.937991e+02    2.765032e+00         8.901088e+02
min    0.000000e+00  4.100000e-01  2.459000e+01    0.000000e+00         8.900000e+01
25%    0.000000e+00  1.300000e+02  1.652000e+02    5.000000e+00         9.170000e+02
50%    0.000000e+00  2.223300e+02  2.621000e+02    7.000000e+00         1.517000e+03
75%    0.000000e+00  3.627900e+02  4.142000e+02    9.000000e+00         2.461000e+03
max    2.000000e+00  7.344190e+03  7.918600e+03    1.000000e+01         4.654000e+03
```

```
# Plot histograms for numerical columns
for col in df.select_dtypes(include=['float64', 'int64']).columns:
    df[col].hist(bins=20)
    plt.title(f"{col} - {chunk_label}")
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

Appendix C (code for data cleaning)

```
chunk_cleaned = chunk.dropna()
# Rename columns to replace spaces with underscores
chunk_cleaned.columns = chunk_cleaned.columns.str.replace(' ', '_')

  # Upload to GCS
bucket = client.get_bucket(output_bucket_name)
cleaned_blob =
bucket.blob(f"{output_path}itineraries_cleaned_chunk_{chunk_number}.
parquet")
cleaned_blob.upload_from_filename(output_file_path)
print(f"Uploaded chunk {chunk_number} to GCS at
{output_path}itineraries_cleaned_chunk_{chunk_number}.parquet")

# Increment the chunk counter
chunk_number += 1
```

```
...
Uploaded chunk 0 to GCS at cleaned/itineraries_cleaned_chunk_0.parquet
Uploaded chunk 1 to GCS at cleaned/itineraries_cleaned_chunk_1.parquet
Uploaded chunk 2 to GCS at cleaned/itineraries_cleaned_chunk_2.parquet
Uploaded chunk 3 to GCS at cleaned/itineraries_cleaned_chunk_3.parquet
Uploaded chunk 4 to GCS at cleaned/itineraries_cleaned_chunk_4.parquet
Uploaded chunk 5 to GCS at cleaned/itineraries_cleaned_chunk_5.parquet
Uploaded chunk 6 to GCS at cleaned/itineraries_cleaned_chunk_6.parquet
Uploaded chunk 7 to GCS at cleaned/itineraries_cleaned_chunk_7.parquet
Uploaded chunk 8 to GCS at cleaned/itineraries_cleaned_chunk_8.parquet
>>>
```

Milestone 4 Feature Engineering and Modeling:

```
from pyspark.ml.feature import MinMaxScaler, StringIndexer,
OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.sql.functions import unix_timestamp
from pyspark.sql.functions import when, col
from pyspark.ml.feature import VectorAssembler
from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder from
pyspark.ml.evaluation import BinaryClassificationEvaluator

#Load file from Cleaned file
spark =
SparkSession.builder.appName("FeatureEngineeringAndModeling").get
OrCreate()
df = spark.read.option("header",
"true").csv("gs://my-bigdata-project-kh/cleaned/itineraries.csv")
df.write.parquet("gs://my-bigdata-project-kh/cleaned/itineraries_parquet"
)
cleaned_data_path =
"gs://my-bigdata-project-kh/cleaned/itineraries_parquet"
data = spark.read.parquet(cleaned_data_path)
data.show()
data.printSchema()
```

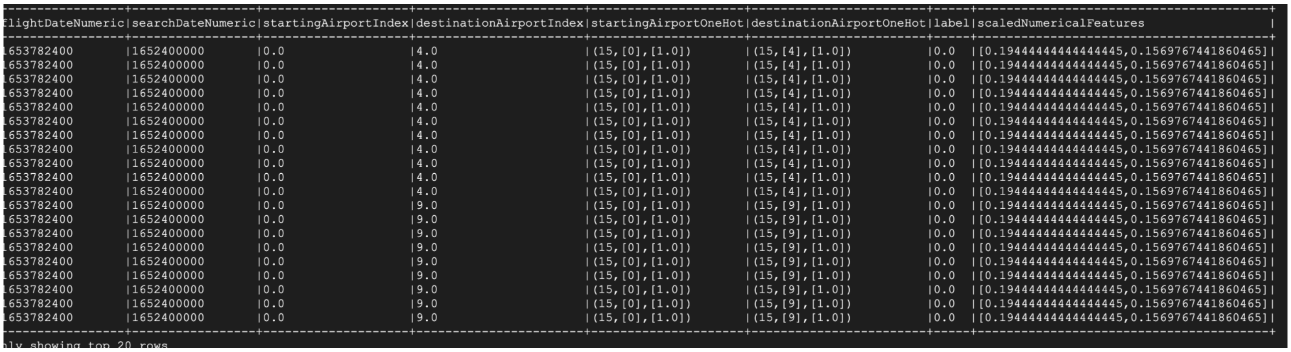
```
>>> data.printSchema()
root
 |-- legId: string (nullable = true)
 |-- searchDate: string (nullable = true)
 |-- flightDate: string (nullable = true)
 |-- startingAirport: string (nullable = true)
 |-- destinationAirport: string (nullable = true)
 |-- fareBasisCode: string (nullable = true)
 |-- travelDuration: string (nullable = true)
 |-- elapsedDays: string (nullable = true)
 |-- isBasicEconomy: string (nullable = true)
 |-- isRefundable: string (nullable = true)
 |-- isNonStop: string (nullable = true)
 |-- baseFare: string (nullable = true)
 |-- totalFare: string (nullable = true)
 |-- seatsRemaining: string (nullable = true)
 |-- totalTravelDistance: string (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)
```

```
------+------------------+----------------------------+--------------------+--------------------------+------------------+----------------------+---------------
----------+------------------+-------------------+---------------------------+---------------------+----------------+-----------------+
|               legId|searchDate|flightDate|startingAirport|destinationAirport|fareBasisCode|travelDuration|elapsedDays|isBasicEconomy|isRefundable|isNonStop|baseFare|totalFare|seatsRem
aining|totalTravelDistance|segmentsDepartureTimeEpochSeconds|segmentsDepartureTimeRaw|segmentsArrivalTimeEpochSeconds|segmentsArrivalTimeRaw|segmentsArrivalAirportCode|segmentsDeparture
AirportCode| segmentsAirlineName|segmentsAirlineCode|segmentsEquipmentDescription|segmentsDurationInSeconds|segmentsDistance|segmentsCabinCode|
+--------------------+----------+----------+---------------+------------------+-------------+--------------+-----------+--------------+------------+---------+--------+---------+--------
------+------------------+----------------------------+--------------------+--------------------------+------------------+----------------------+---------------
----------+------------------+-------------------+---------------------------+---------------------+----------------+-----------------+
|8ed6dd65e00f159d7...|2022-05-13|2022-05-29|            LAX|               ORD|       RA14NR|      PT13H55M|          1|         False|       False|    False|  346.00|   453.58|
     0|               NULL|            1653891480||16539...|    2022-05-29T23:18:...|            1653907200||16539...|  2022-05-30T06:40:...|                  ATL||ORD|
   LAX||ATL|Spirit Airlines||...|             NK||NK|            ||Airbus A319|           15720||7680|      None||None|     coach||coach|
|7102e5ea167514cde...|2022-05-13|2022-05-29|            LAX|               ORD|     QH7OAVMN|       PT6H44M|          0|         False|       False|    False|  407.44|   461.60|
     3|               2206|            1653837600||16538...|    2022-05-29T08:20:...|            1653848700||16538...|  2022-05-29T13:25:...|                  AUS||ORD|
   LAX||AUS|Alaska Airlines||...|             AS||UA|     Embraer 175||Airb...|           11100||9840|        1236||970|     coach||coach|
|dfcd054d67d394f63...|2022-05-13|2022-05-29|            LAX|               ORD|     MATOA0MQ|       PT6H12M|          0|         False|       False|    False|  432.56|   488.60|
     9|               1810|            1653856800||16538...|    2022-05-29T13:40:...|            1653862920||16538...|  2022-05-29T16:22:...|                  SLC||ORD|
   ONT||SLC|         Delta||Delta|             DL||DL|       Boeing 737-800|||           6120||11220|        559||1251|     coach||coach|
|bf1a0df0a12540d4b...|2022-05-13|2022-05-29|            LAX|               ORD|     LH4OASMN|      PT13H12M|          1|         False|       False|    False|  444.65|   507.20|
     4|               2681|            1653883380||16539...|    2022-05-29T21:03:...|            1653892980||16539...|  2022-05-29T23:43:...|                  SEA||ORD|
   ONT||SEA|Alaska Airlines||...|             AS||AS|     Boeing 737-900||B...|           9600||15000|        958||1723|     coach||coach|
|30a3fe835af56f533...|2022-05-13|2022-05-29|            LAX|               ORD|     NH4OAVMN|      PT12H44M|          1|         False|       False|    False|  463.25|   527.19|
     2|               2186|            1653889200||16539...|    2022-05-29T22:40:...|            1653893940||16539...|  2022-05-29T23:59:...|                  SFO||ORD|
   LAX||SFO|Alaska Airlines||...|             AS||AS|     Embraer 175||Airb...|           4740||15840|        339||1847|     coach||coach|
|a9f54f545f44ac7c3...|2022-05-13|2022-05-29|            LAX|               ORD|     MH0OAJMN|       PT8H14M|          0|         False|       False|    False|  506.98|   568.60|
     7|               2679|            1653856500||16538...|    2022-05-29T13:35:...|            1653866700||16538...|  2022-05-29T16:25:...|                  SEA||ORD|
   LAX||SEA|Alaska Airlines||...|             AS||AS|     AIRBUS INDUSTRIE ...|          10200||14640|        956||1723|     coach||coach|
|703bbe1e292b44b66...|2022-05-13|2022-05-29|            LAX|               ORD|     KA7QA0MQ|          PT6H|          1|         False|       False|    False|  509.77|   571.60|
     9|               1875|            1653893940||16539...|    2022-05-29T23:59:...|            1653907080||16539...|  2022-05-30T05:38:...|                  MSP||ORD|
   LAX||MSP|        Delta||United|             DL||UA|     Airbus A321||Boei...|           13140||5400|        1534||341|     coach||coach|
|88927369637f3d94e...|2022-05-13|2022-05-29|            LAX|               ORD|     MH4OASMN|       PT10H5M|          1|         False|       False|    False|  515.35|   577.60|
     7|               2681|            1653872100||16538...|    2022-05-29T17:55:...|            1653881760||16539...|  2022-05-29T20:36:...|                  SEA||ORD|
   ONT||SEA|Alaska Airlines||...|             AS||AS|     Boeing 737-900||B...|           9660||14700|        958||1723|     coach||coach|
|4bf9a73a0917b1953...|2022-05-13|2022-05-29|            LAX|               ORD|     MH4OASMN|      PT11H19M|          0|         False|       False|    False|  519.07|   587.20|
     3|               2681|            1653845400||16538...|    2022-05-29T10:30:...|            1653855000||16538...|  2022-05-29T13:10:...|                  SEA||ORD|
   ONT||SEA|Alaska Airlines||...|             AS||AS|     Boeing 737-900||B...|           9600||14640|        958||1723|     coach||coach|
|4a3c2566879282223...|2022-05-13|2022-05-29|            LAX|               ORD|     KH7OASMN|      PT11H15M|          1|         False|       False|    False|  565.58|   637.20|
     1|               2679|            1653867900||16538...|    2022-05-29T16:45:...|            1653878100||16539...|  2022-05-29T19:35:...|                  SEA||ORD|
   LAX||SEA|Alaska Airlines||...|             AS||AS|     Boeing 737-900||B...|          10200||14700|        956||1723|     coach||coach|
|c8d8266cbe02289e5...|2022-05-13|2022-05-29|            LAX|               ORD|     HH7OASMN|       PT8H15M|          1|         False|       False|    False|  612.09|   681.60|
     1|               2679|            1653878700||16538...|    2022-05-29T19:45:...|            1653888900||16539...|  2022-05-29T22:35:...|                  SEA||ORD|
   LAX||SEA|Alaska Airlines||...|             AS||AS|     Airbus A320||Boei...|          10200||14700|        956||1723|     coach||coach|
|bef43ded52c540a3b...|2022-05-13|2022-05-29|            LAX|               PHL|       TA14NR|       PT5H16M|          1|         False|       False|     True|  133.00|   189.59|
     0|               NULL|                  1653884700|    2022-05-29T21:25:...|                  1653903660|  2022-05-30T05:41:...|                       PHL|
        LAX|     Spirit Airlines|                 NK|                        NULL|                    18960|            None|            coach|
|1515c6638bde80329...|2022-05-13|2022-05-29|            LAX|               PHL|     SUAIZNN3|       PT7H18M|          0|         False|       False|    False|  208.37|   247.60|
    10|               2491|            1653854040||16538...|    2022-05-29T12:54:...|            1653864600||16538...|  2022-05-29T17:50:...|                  DFW||PHL|
   ONT||DFW|American Airlines...|             AA||AA|     Airbus A321||Airb...|          10560||11820|       1193||1298|     coach||coach|
|a3bcb8cb0501cfb35...|2022-05-13|2022-05-29|            LAX|               PHL|     SUAIZNN3|       PT7H27M|          0|         False|       False|    False|  208.37|   247.60|
```

Appendix D

```
data = data.withColumn("totalFare", col("totalFare").cast(DoubleType()))

data = data.withColumn("seatsRemaining",
col("seatsRemaining").cast(DoubleType()))

indexer = StringIndexer(inputCols=['startingAirport', 'destinationAirport'],
outputCols=['startingAirportIndex', 'destinationAirportIndex'])

one_hot_encoder = OneHotEncoder(inputCols=['startingAirportIndex',
'destinationAirportIndex'], outputCols=['startingAirportOneHot',
'destinationAirportOneHot'])

assembler = VectorAssembler(inputCols=['seatsRemaining', 'totalFare'],
outputCol="features")


scaler = MinMaxScaler(inputCol='features',
outputCol='scaledNumericalFeatures')

data = data.withColumn("label", when(col("seatsRemaining") > 5,
1.0).otherwise(0.0))

pipeline = Pipeline(stages=[indexer, one_hot_encoder, assembler,
scaler])
pipeline_model = pipeline.fit(data)
processed_data = pipeline_model.transform(data)
processed_data.select(['flightDateNumeric',
'searchDateNumeric','startingAirportIndex', 'destinationAirportIndex',
```

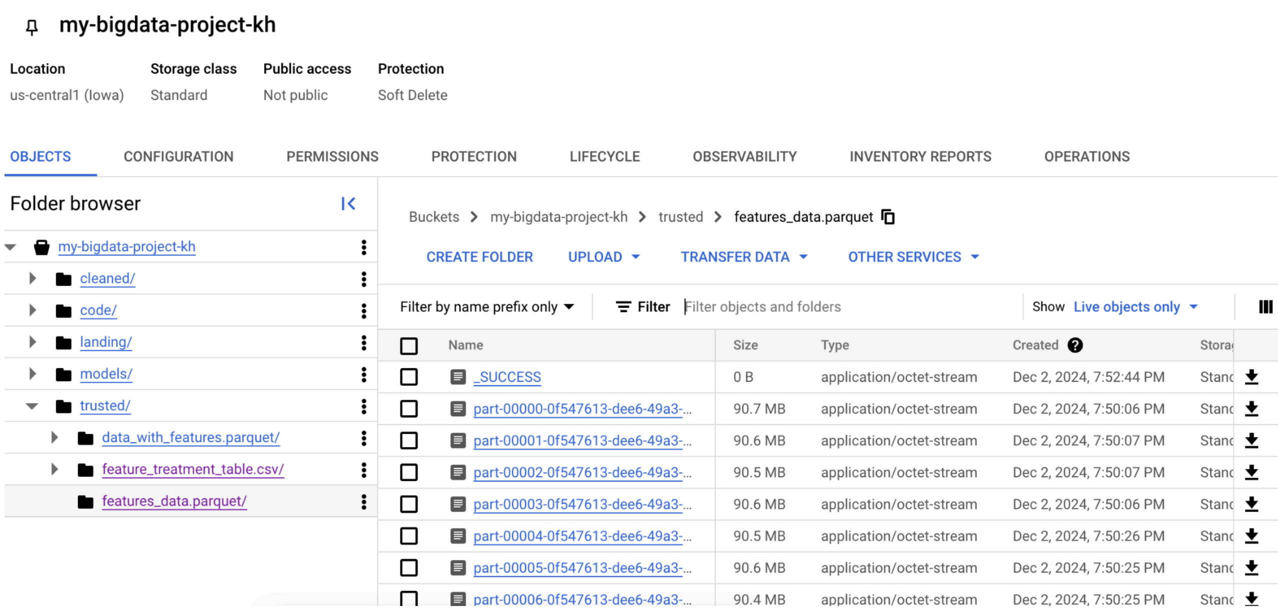
'startingAirportOneHot', 'destinationAirportOneHot',
'label','scaledNumericalFeatures']).show(truncate=False)



flightDateNumeric	searchDateNumeric	startingAirportIndex	destinationAirportIndex	startingAirportOneHot	destinationAirportOneHot	label	scaledNumericalFeatures
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	4.0	(15,[0],[1.0])	(15,[4],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]
1653782400	1652400000	0.0	9.0	(15,[0],[1.0])	(15,[9],[1.0])	0.0	[0.19444444444444445,0.1569767441860465]

only showing top 20 rows

trusted_data_path =
"gs://my-bigdata-project-kh/trusted/features_data.parquet"
processed_data.write.parquet(trusted_data_path)



train_data, test_data = processed_data.randomSplit([0.7, 0.3], seed=42)
lr = LogisticRegression(featuresCol="scaledNumericalFeatures",
labelCol="label")
model = lr.fit(train_data)

#24/12/03 00:32:04 WARN Instrumentation: [c615ff99] All labels are the same value and fitIntercept=true, so the coefficients will be zeros. Training is not needed.

test_results = model.transform(test_data)

test_results.select('scaledNumericalFeatures', 'label', 'rawPrediction', 'probability', 'prediction').show()

```
>>> test_results.select('scaledNumericalFeatures', 'label', 'rawPrediction', 'probability', 'prediction').show()
+-----------------------+-----+--------------------+-----------+----------+
|scaledNumericalFeatures|label|       rawPrediction|probability|prediction|
+-----------------------+-----+--------------------+-----------+----------+
|   [0.24074074074074...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.32407407407407...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.24074074074074...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.31944444444444...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.20833333333333...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.21296296296296...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.21296296296296...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.30555555555555...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.22222222222222...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.34722222222222...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.34259259259259...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.33333333333333...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.33333333333333...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.33333333333333...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.25462962962962...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.27314814814814...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.27314814814814...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.31481481481481...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.25,0.127906976...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
|   [0.25,0.238372093...|  0.0|[Infinity,-Infinity]|  [1.0,0.0]|       0.0|
+-----------------------+-----+--------------------+-----------+----------+
only showing top 20 rows
```

Cross Validation

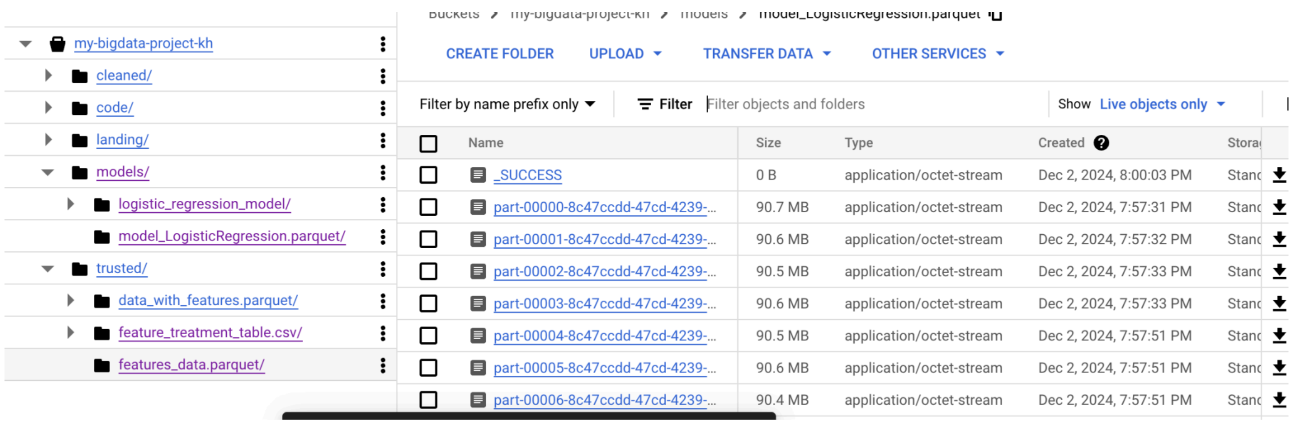
param_grid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.1, 0.01]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()

evaluator = BinaryClassificationEvaluator(labelCol="label", metricName="areaUnderROC")

crossval = CrossValidator(estimator=lr, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=5)
cv = crossval.fit(train_data)

best_model = cv.bestModel

model_data_path =
"gs://my-bigdata-project-kh/models/model_LogisticRegression.parquet"
processed_data.write.parquet(model_data_path)

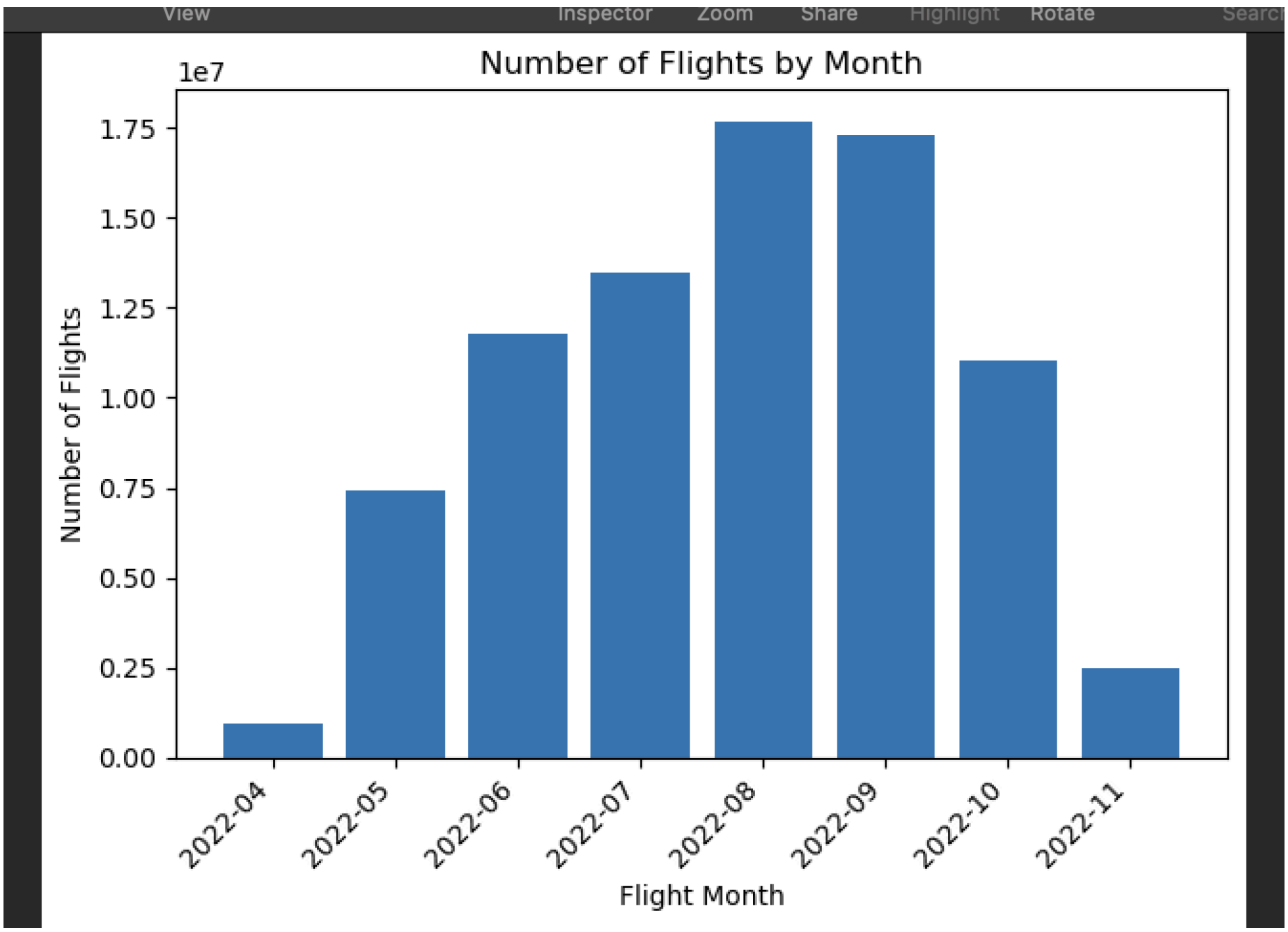


Milestone 5 Visualization:

import matplotlib.pyplot as plt
import pandas as pd
import io
from google.cloud import storage
from pyspark.sql.functions import date_format

data_with_month = data.withColumn("order_month",
date_format("flightDate", "yyyy-MM"))
summary_data =
data_with_month.groupBy("order_month").count().sort("order_month")
df = summary_data.toPandas()

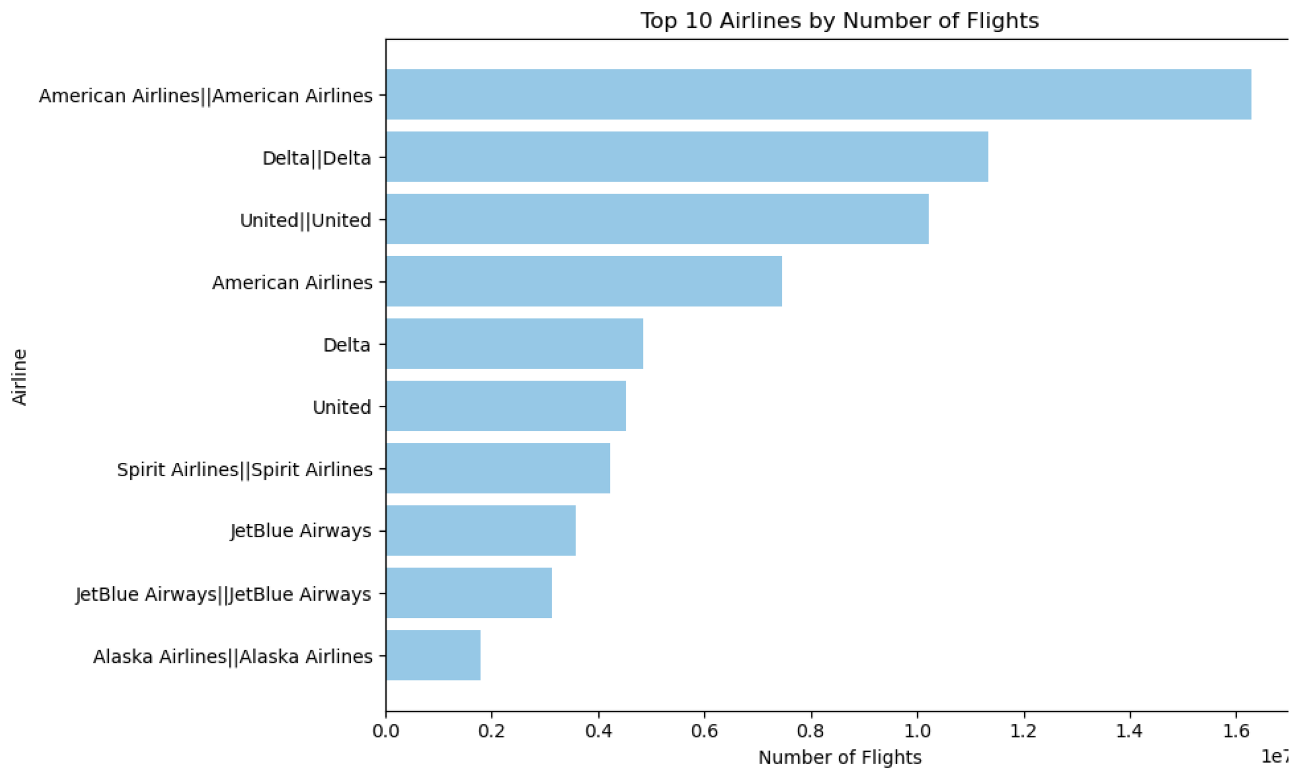
```
fig = plt.figure(facecolor='white')
plt.bar(df['order_month'], df['count'])
plt.xlabel("Flight Month")
plt.ylabel("Number of Flights")
plt.title("Number of Flights by Month")
plt.xticks(rotation=45, ha='right')
fig.tight_layout()
plt.savefig("flight_count_by_month_matplotlib.png")
```



Report: August and September have the highest number of flights in a year.

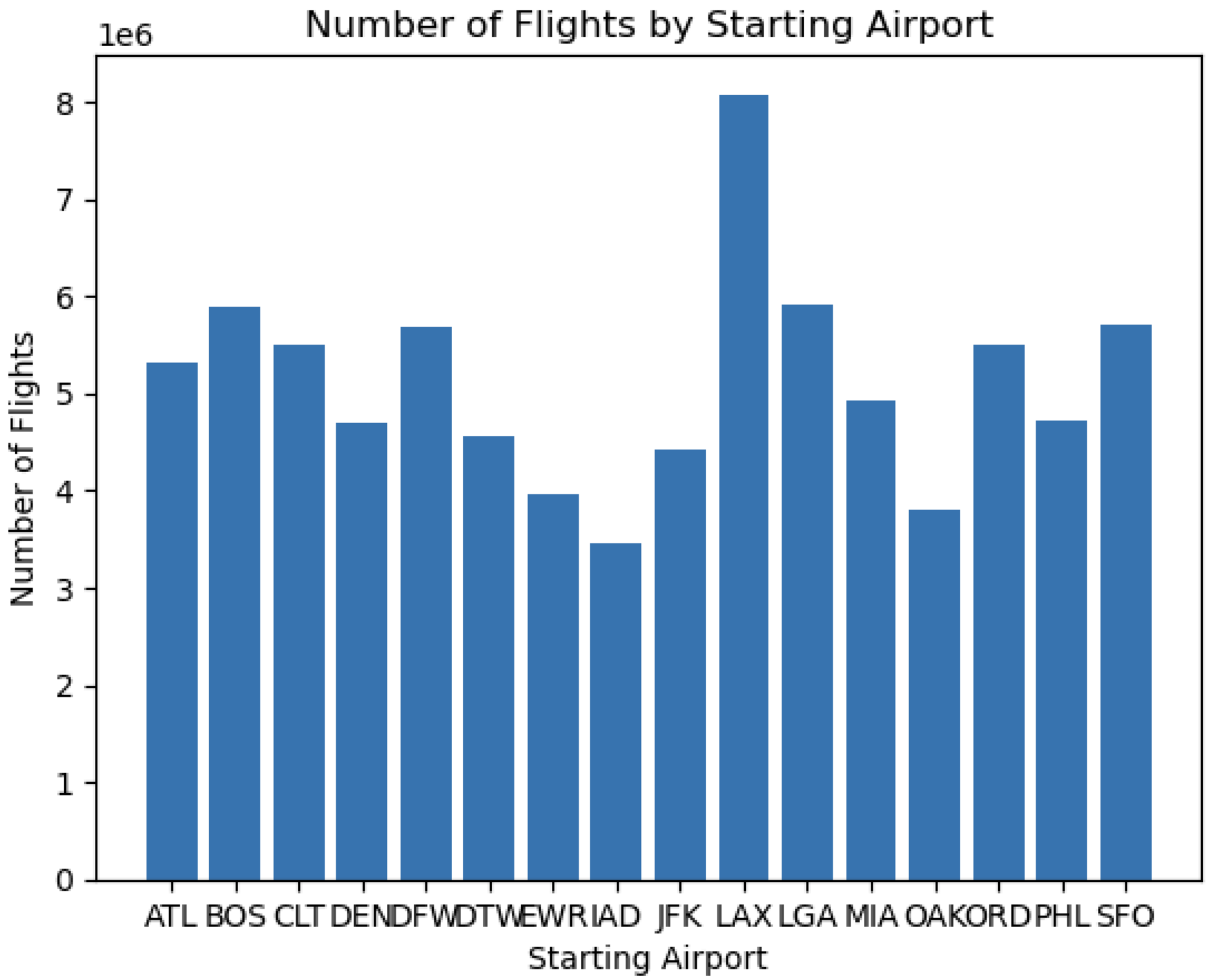
```
#save image
img_data = io.BytesIO()
fig.savefig(img_data, format='png', bbox_inches='tight')
```

```
img_data.seek(0)
storage_client = storage.Client()
bucket = storage_client.get_bucket('my-bigdata-project-kh')
blob = bucket.blob("figures/flight_count_by_month_matplotlib.png")
blob.upload_from_file(img_data)
```



Report: American Airlines is the most popular, followed by Delta Airlines.

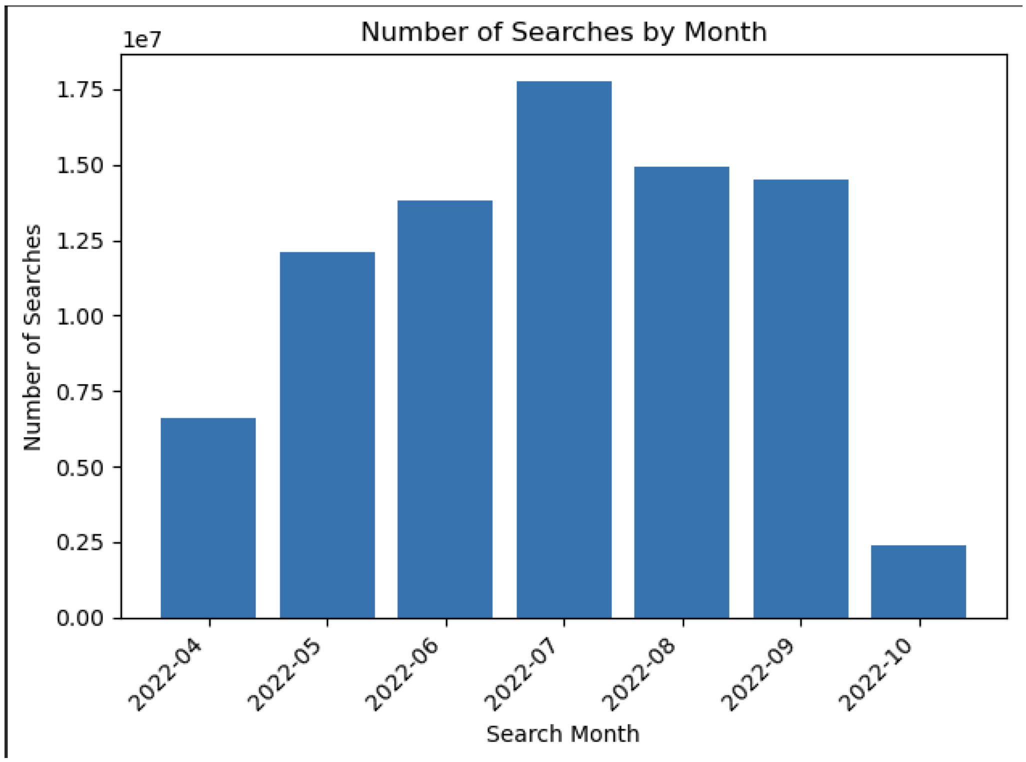
```
top_10_airlines = (data.groupBy("segmentsAirlineName")
.count().orderBy("count", ascending=False) .limit(10) .toPandas())
fig, ax = plt.subplots(figsize=(10, 6), facecolor='white')
ax.barh(top_10_airlines['segmentsAirlineName'],
top_10_airlines['count'], color='skyblue')
ax.set_xlabel('Number of Flights')
ax.set_ylabel('Airline')
ax.set_title('Top 10 Airlines by Number of Flights')
ax.invert_yaxis()
plt.tight_layout()
plt.savefig("top_10_airlines_by_flights.png")
```



Report: In this chart, LAX has the highest number of flights compared to other starting airports, while the flight counts for the remaining airports are relatively similar. This indicates that more people prefer to travel from LAX, making it the busiest airport in terms of flight activity.

```
airport_counts_df =
data.groupby('startingAirport').count().sort('startingAirport').toPandas()
fig = plt.figure(facecolor='white')
plt.bar(airport_counts_df['startingAirport'], airport_counts_df['count'])
plt.title("Number of Flights by Starting Airport")
```

```
plt.xlabel("Starting Airport")
plt.ylabel("Number of Flights")
plt.savefig("number_of_flights_by_starting_airport.png")

img_data = io.BytesIO()
fig.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
storage_client = storage.Client()
bucket = storage_client.get_bucket('my-bigdata-project-kh')
blob = bucket.blob("figures/number_of_flights_by_starting_airport.png")
blob.upload_from_file(img_data)
```

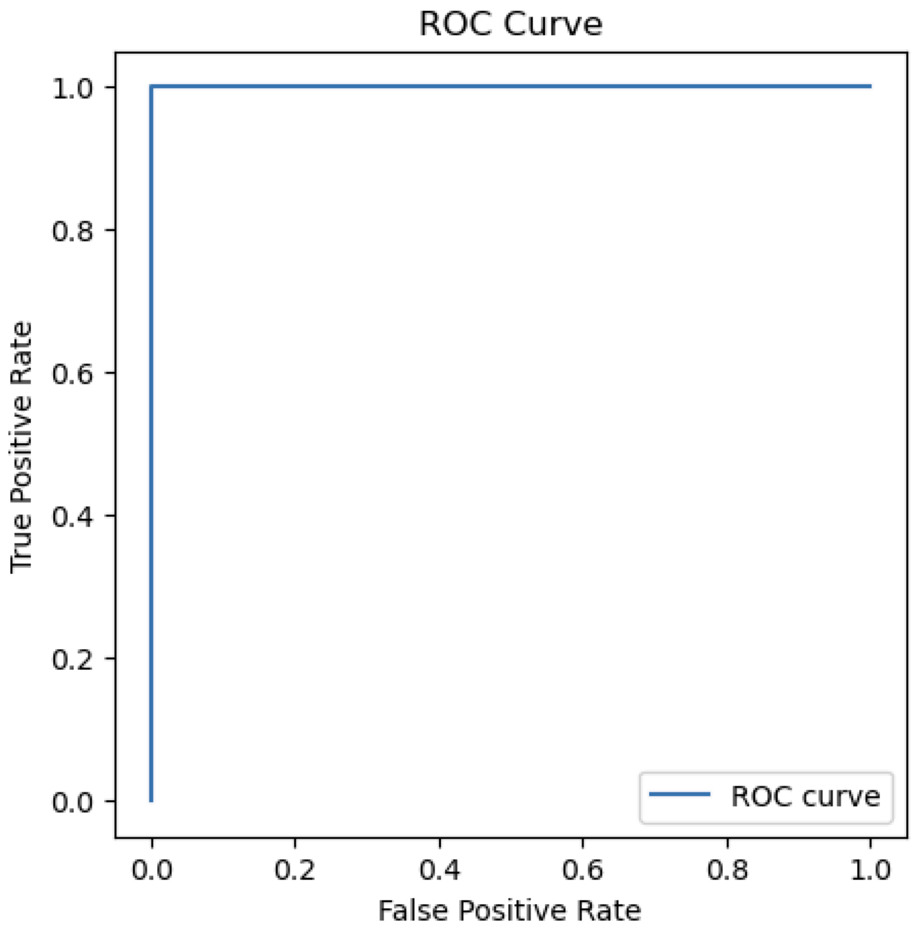


Report: Most people search for flights 1-2 months ahead. August and September are the busiest months, with July seeing a spike in flight searches.

```
data_with_months = data.withColumn("search_month",
date_format("searchDate", "yyyy-MM"))
summary_search_month =
data_with_months.groupBy("search_month").count().sort("search_mont
h")
fig = plt.figure(facecolor='white')
plt.bar(summary_search_month['search_month'],
summary_search_month['count'])
plt.xlabel("Search Month")
plt.ylabel("Number of Searches")
plt.title("Number of Searches by Month")
```

```
plt.xticks(rotation=45, ha='right')
fig.tight_layout()


img_data = io.BytesIO()
fig.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
storage_client = storage.Client()
bucket = storage_client.get_bucket('my-bigdata-project-kh')
blob = bucket.blob("figures/Number_of_Searches_by_Month.png")
blob.upload_from_file(img_data)
```

6:



Report : perfect discrimination with an AUC of 1.0, indicating it successfully differentiates between classes.

```
import matplotlib.pyplot as plt
import io
from google.cloud import storage

roc_data = best_model.summary.roc
fpr = [row['FPR'] for row in roc_data.collect()]
tpr = [row['TPR'] for row in roc_data.collect()]

plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label="ROC curve")
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
plt.legend(loc="lower right")

img_data = io.BytesIO()
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
storage_client = storage.Client()
bucket = storage_client.get_bucket('my-bigdata-project-kh')
blob = bucket.blob("figures/roc1.png")
blob.upload_from_file(img_data)
```

7:

```
>>> print(f"AUC: {auc}")
AUC: 1.0
>>> print(calculate_recall_precision(cm))
(0.03993320927827111, 0.0561498603220698, 0.12147228221964443, 0.07679956543744833)
>>> 
```

Report : the low values for accuracy (approximately 3.99%), precision (5.61%), recall (12.15%), and F1 score (7.68%) suggest that the model struggles with a high false positive rate, resulting in poor performance overall despite the ideal ROC outcome.

```
auc = evaluator.evaluate(predictions)
print(f"AUC: {auc}")
cm =
predictions.groupby('label').pivot('prediction').count().fillna(0).collect()
def calculate_recall_precision(cm):
        tn = cm[0][1]
        fp = cm[0][2]
        fn = cm[1][1]
        tp = cm[1][2]
        precision = tp / (tp + fp)
        recall = tp / (tp + fn)
        accuracy = (tp + tn) / (tp + tn + fp + fn)
        f1_score = 2 * ((precision * recall) / (precision + recall))
        return accuracy, precision, recall, f1_score

print( calculate_recall_precision(cm) )
```

Summary

1. Flight Prediction Analysis

- **Objective:** Predict the likelihood of high flight numbers based on the 'flightDate' column using logistic regression.
- Logistic regression was used to predict flight numbers on specific dates.
- The model shows low accuracy (3.99%), precision (5.61%), recall (12.15%), and F1 score (7.68%). These metrics indicate the model struggles with a high false positive rate, leading to poor overall performance despite achieving perfect discrimination with an AUC of 1.0. The ideal ROC outcome suggests the model is effective at distinguishing between high and low flight dates.

2. Flight Search Trends : Most people tend to search for flights 1-2 months ahead of their planned travel dates. August and September are the busiest months for flight searches.A significant spike in searches occurs in July, likely due to summer travel planning.

3. Airport Flight Activity : LAX (Los Angeles International Airport) leads in flight activity, with the highest number of flights departing compared to other airports. Flight counts from other airports are relatively similar, showing a strong preference for LAX as a starting point for travelers.

4. Airline Preferences : American Airlines is the most popular airline, followed by Delta Airlines, suggesting a strong preference for these carriers among travelers.

5. Flight Activity Overview : Peak Months: The busiest months for flights are August and September, which align with the high number of flight searches during these periods.