Flight Delays ML Project

Class: CIS 4130
Section: CMWA
By: Susan Jiang
Email: susan.jiang@baruchmail.cuny.edu

Milestone 1-Project Proposal:

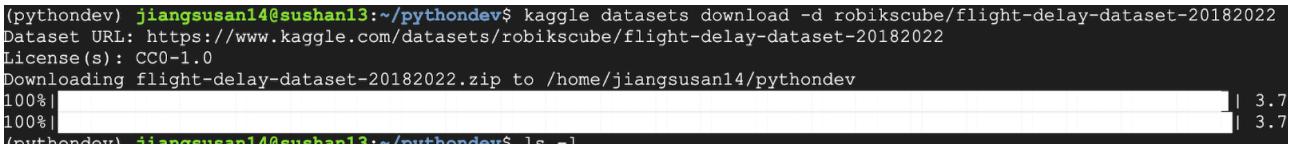
Flight delays and cancellations are common inconveniences that most travelers experience, especially, during the holiday seasons or due to extreme weather conditions. To avoid these issues, and to make better traveling accommodations, it would be helpful to analyze historical flight delays dataset to determine the airlines most reliable airline. The dataset to be analyzed for this project comes from Kaggle called Flight Status Prediction. This database consists of records of flight delays and cancellations from January 1st 2018, to December 31st, 2022. The dataset consists of attributes like flight dates (broken down into "yyyymmdd" format, fiscal quarters, days of month, and days of week), airline, origin, destination (city and state), canceled (boolean), driveted (boolean), CRS Departure time, departure time, CRS arrival time, (actual) arrival time, arrival time delay, air time, CRS elapsed time, actual elapsed time, distance, carrier delays, and tail number.

For this project, I will be forecasting the airline that will most likely have delays during this holiday season. To find this information, I am going to isolate each airline's delay information, and compare their data with each other. The variables that I will be using to compare the airline's delay discrepancy are flight dates, delay departure minutes, actual departure time, CRS arrival time, and actual arrival. In addition, I will be using variables like airlines, cancellation, and diverted to identify the correlation between airlines and their number of cancellations or diversions. Lastly, the machine learning model that I will be using to implement this plan is ridge regression.

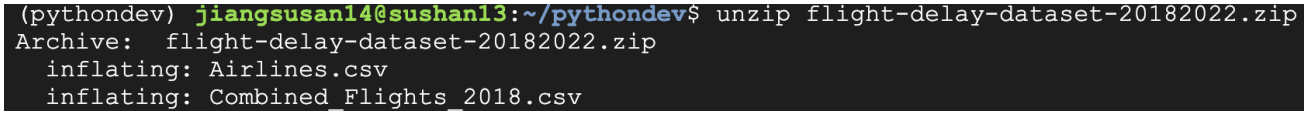
Milestone 2-Data Collection:

First, I created a virtual machine instance that has 50GB of boot disk on Google Cloud Platform to ensure that there is enough space to download the dataset and other software packages. While running the VM instance, I created a Kaggle directory (mkdir .kaggle), uploaded the Kaggle API Token file, and secured the file (chmod 600 .kaggle/kaggle.json). I installed additional software: ZIP utilities, pip3, and virtual environment tools.

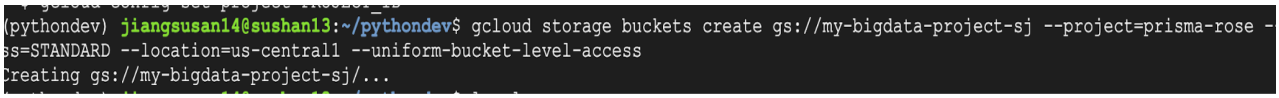
Second, I created a Python virtual environment, changed the directory, activated the virtual environment, and installed Kaggle tools. To retrieve and download the dataset from Kaggle to the virtual environment, I copied the API command from the dataset on Kaggle to the vm:



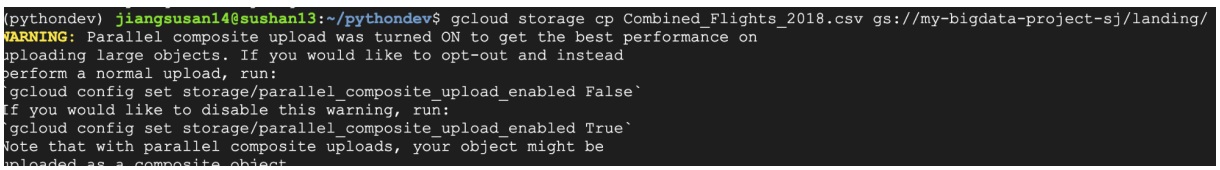
After the files are downloaded, I unzip the file:



Next, I created a bucket in Google Cloud storage called my-big data-project-sj:



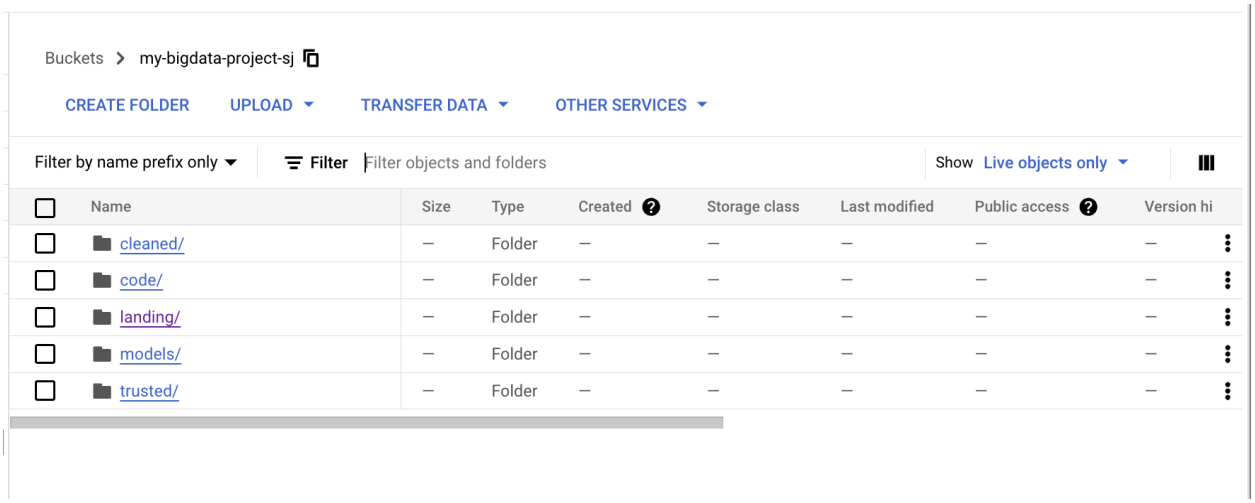
I copied the dataset files to the bucket and into a folder called landing using the following command:

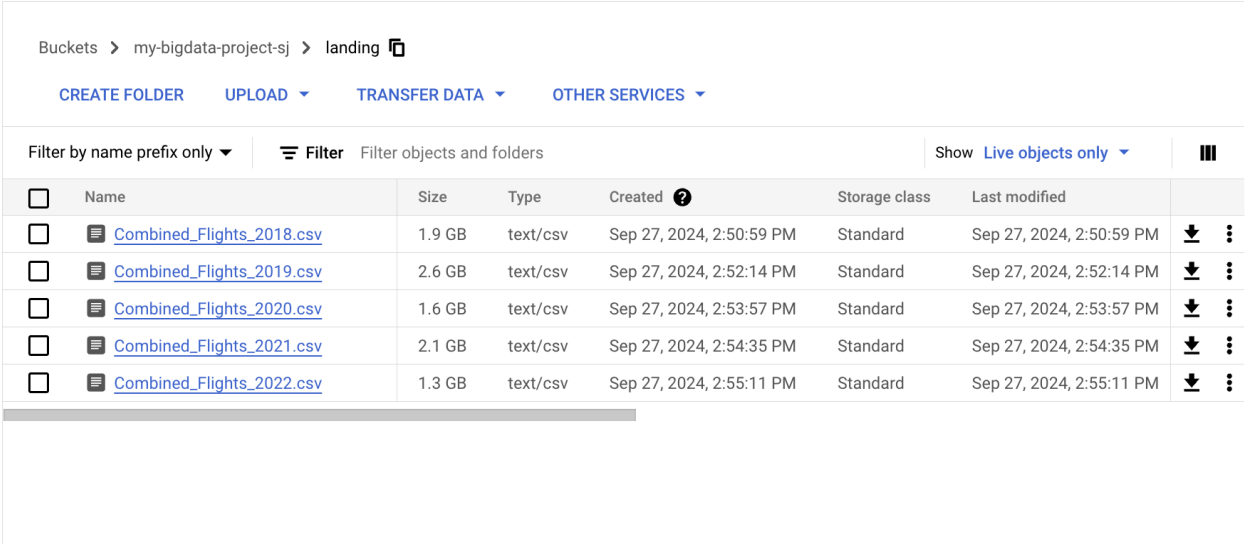


Since my dataset contains multiple files, I ran the same code, but changed the year in the file name. To ensure that my files are all copied to my landing folder, I ran a list command:

```
(pythondev) jiangsusan14@sushan13:~/pythondev$ gcloud storage ls gs://my-bigdata-project-sj/landing/
gs://my-bigdata-project-sj/landing/Combined_Flights_2018.csv
gs://my-bigdata-project-sj/landing/Combined_Flights_2019.csv
gs://my-bigdata-project-sj/landing/Combined_Flights_2020.csv
gs://my-bigdata-project-sj/landing/Combined_Flights_2021.csv
gs://my-bigdata-project-sj/landing/Combined_Flights_2022.csv
```

Lastly, on Google Cloud Storage, I created additional folders called cleaned, code, models and trusted. The results of milestone 2 is a bucket called my-bigdata-project-sj and a landing folder containing my dataset files:



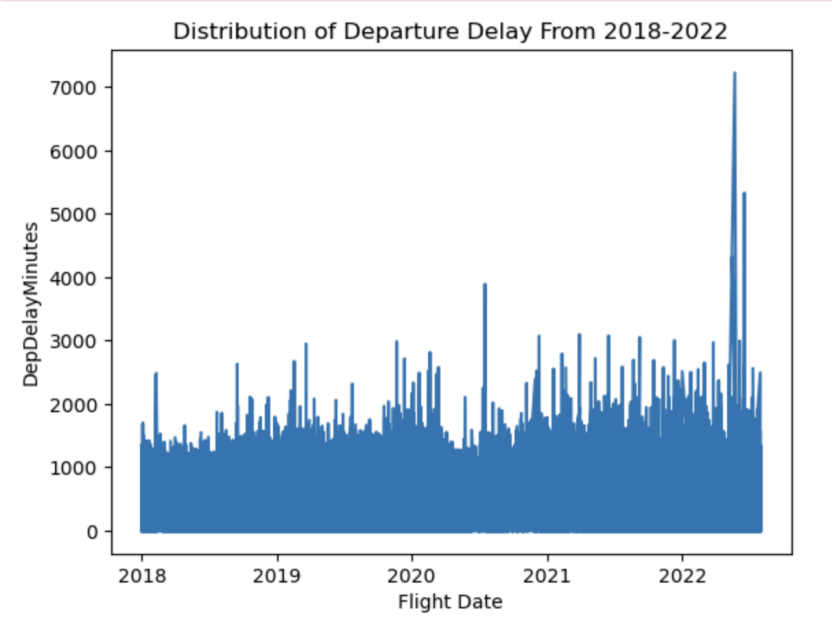


Milestone 3 Exploratory Data Analysis and Data Cleaning:

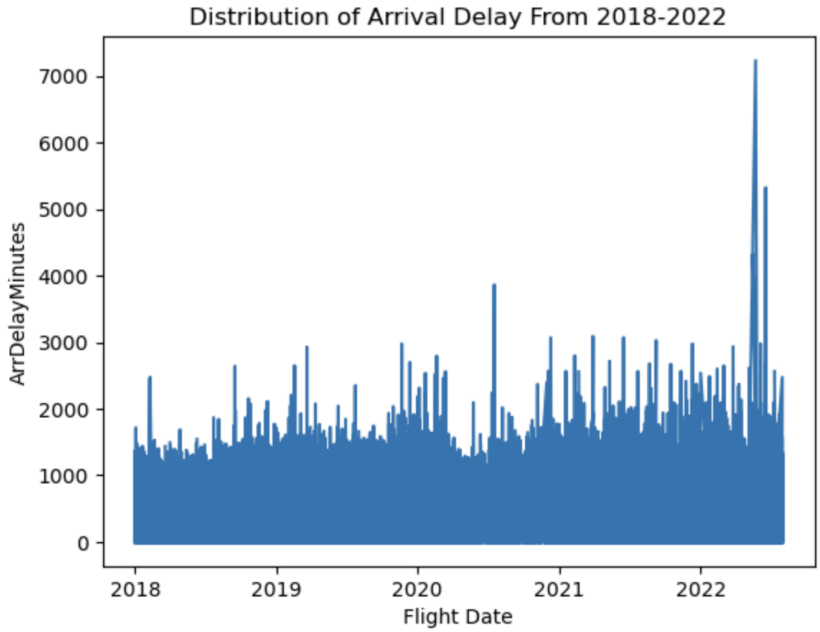
For the EDA section, I chose to use Pyspark to explore my data. In the EDA section, I am loading my csv file, finding the schema of the dataset, and counting the total number of records. After running my single cluster, I load my files onto a Jupyter notebook into a spark dataframe. Then I used the schema function to get a list of my variables (columns) and their data type. Some variables include: FlightDate, Airline: string, Origin, Dest, Canceled, Diverted, CRSDepTime, DepTime, DepDelayMinutes, DepDelay, and more. Next, I used the count function to find the total record, which is 2,919,3782 records. Then, I used the max and min function to find the dates that my files covered and discovered that the flight delay dataset consisted of records from 2018-02-01 through 2022-07-31.

The second section of EDA, I focused on retrieving the summary for the column in the dataset that has an integer or double data type. I ran a couple of summaries on the columns that have the data type integer and double. For the ArrDelay column , I found that the minimum is -1290 minutes (21 hours), maximum, 7232 minutes (5days), average is 3 minutes, and total null value is 846,183 null values. In addition, for the column that has the data type. double, I ran a loop through the columns in the spark dataframe. Some important information I found after executing the loop was that departure delays have an average of 9 minutes, minimum of -1280 minutes (21 hours), and maximum of 7223 minutes (5 days). The negative value represents early departure. When counting null values in the dataset, I ran a loop that counted the number of nulls for each column in the dataframe. Through this process I found that there are some null values with the highest being 846,183 nulls for ArrDelayMinute. In the data cleaning phase, I will be cleaning these values.

The last section of my EDA, I created a visual to show the distribution of my data. My first graph shows the distribution of departure delays. The graph below shows that the highest delay was in 2020.



In addition, each year, there is a peak in flight delays, and this would be interesting to investigate. In addition, I have created a graph that shows the distribution of arrival delays.



Similarly to the departure delay, arrival delays look almost identical to it. This finding supports the summary of the dataset from the section sections of the EDA because arrival delay and departure delays were almost identical.

Lastly, I worked on cleaning my dataset, and saving it as a parquet file. In the data cleaning phase, I cleaned my dataset by dropping all rows that had null values because there were variables that I needed for the next milestone, but had null values. As well as, I dropped all extreme values in departure delays by setting a threshold of 2 days (2880 minutes). Some challenges I believe that I'm going to face during the feature engineering is getting used to coding with PySpark. In the EDA section, I struggled a lot with coding because of my limited knowledge. There were times where I had an idea for an EDA, but I didn't know how to code it.

Milestone 4 Feature Engineering and Modeling:

The first step of milestone 4 is to feature engineering my attributes. The attributes I'm using are Year, Quarter, Month, Dayofmonth, DayOfWeek, Airline, Origin, Dest, Canceled, Diverted , DepDelaysMinutes, and ArrivalDelaysMinutes. Boolean attributes (e.g. Canceled and Diverted) are encoded as binary values (0=False and 1=True). String attributes (e.g. Airline, Origin, and Dest) are feature engineered using StringIndexer to create individual indexes. String indexes, and integer attributes are featured engineered using One Hot Encoding to create individual vectors. Vectors, indexes, float, and boolean attributes are assembled into a vector called by using VectorAssembler. In addition, I created a ridge regression estimator that will predict DepDeplayMinutes. Then, I created a regression pipeline that included the indexer, assembler, econder, and ridge regression estimator. Lastly, I transformed the dataframe by fitting the regression pipeline to the current dataframe and saved it to the model folder.

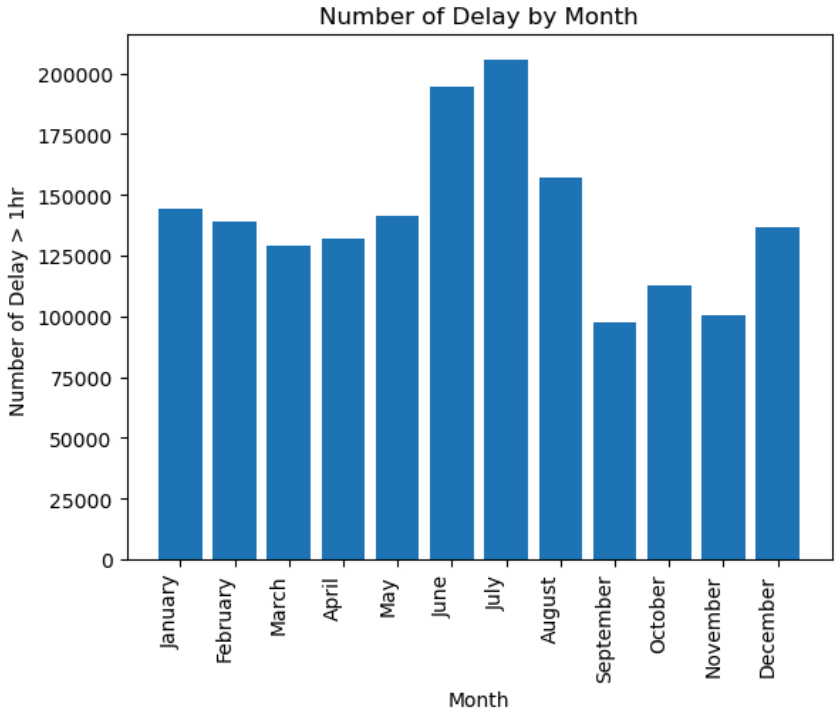
The second step of milestone 4 is to create the model to predict flight delays. First, I split the dataset into 70% training set and 30% testing set. Second, I created a cross validator (3 fold) and trained the training set. The average metric of the models in the 3-fold cross validation was around 9.3. Then, I isolated the best model and trained the testing dataset. Lastly, I calculated the RMSE and R^2 of the model and got the result of about 8.08 and 0.96 respectively. The main challenge of creating a model is to reduce the RMSE because a high RMSE means that the model's prediction was prone to error.

Milestone 5: Data Visualization

In milestone 5, I created four visualizations for my data and prediction results. The parameters of the best model are ElasticParam=0.5 and RegParam: 0.1. Based on model coefficient, "Airline" and "ArrDelayMinutes" were the most important features for the model.

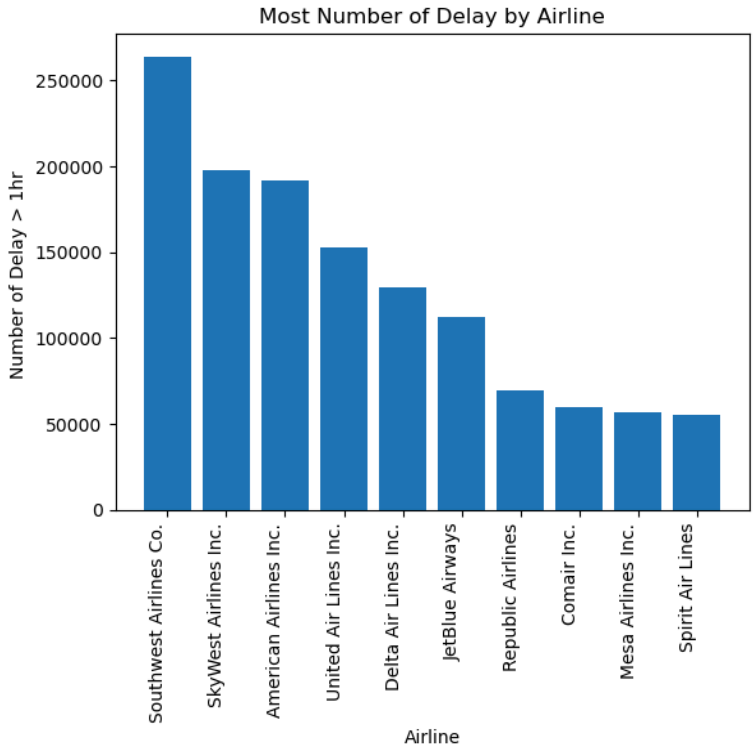
Visualization 1: Number of Delay by Month

The graph below shows the number of delays based on month. I graphed all months where "DepDelayMinutes" are greater than 60 minutes (=1 hour). Based on this bar graph, it illustrates that July, June, and August are the months with the most delay. The reason why these months have the most delays is because it's when passengers are often going on summer vacation.



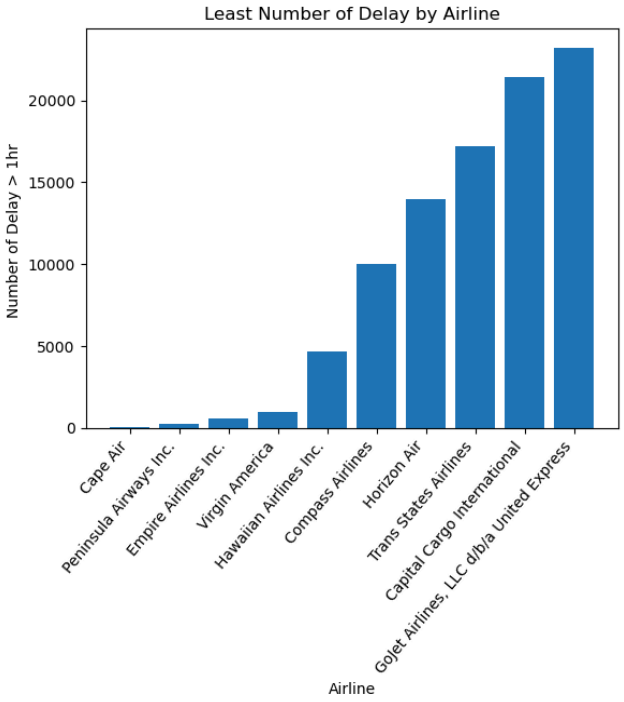
Visualization 2: Top Ten Most Delayed Airline

The second graph represents the top ten airlines that have the most delays. These are airlines that passengers should avoid based on historical airline delay information. Airlines to avoid are Southwest Airlines, Skywest Airline, American Airline, United Airline, Delta Airline, etc. The graph is logical because some of these airlines are considered the largest airlines, for example, Southwest, American, United and Delta airlines.



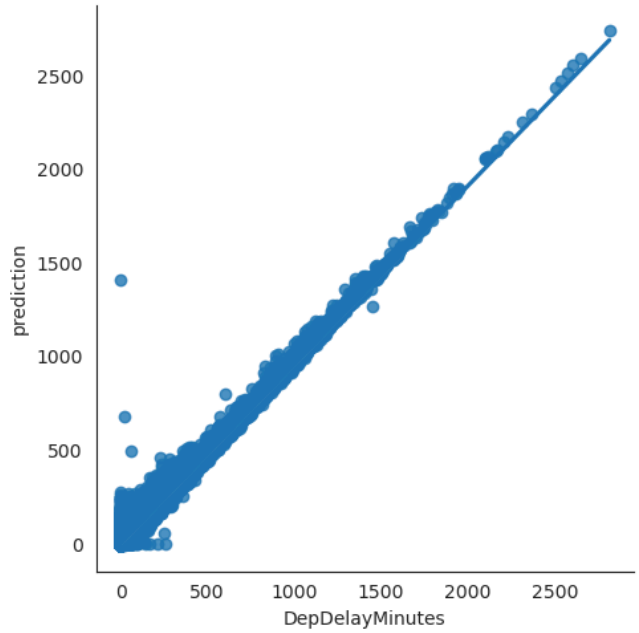
Visualization 3: Top Ten Least Delay Airline

The third graph shows the top ten airlines with the least amount of delays. These are airlines that travelers might want to consider booking with. Airlines with the least amount of delays are Cape Air, Peninsula Airway, Empire Airlines, Virgin Americans, Hawaiian Airline, Compass Airline, etc. These airlines are typically smaller and more regional.



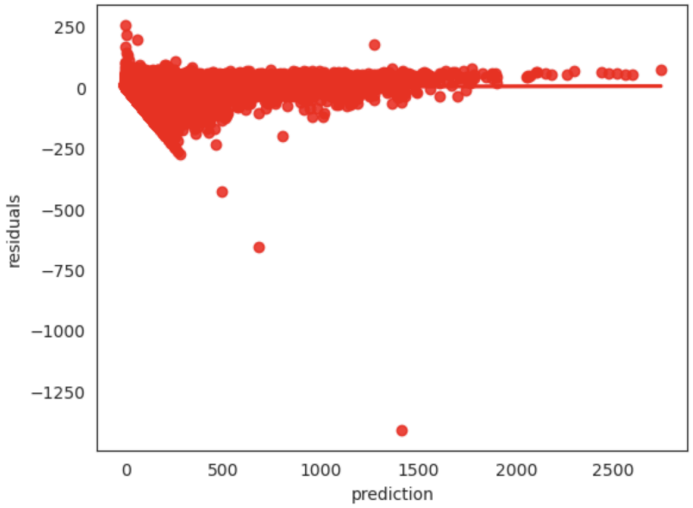
Visualization 4: DepDelayMinutes vs. Prediction

The fourth graph below shows the relationship between the “DepDelayMinutes” and “prediction” for 15% of the dataset. This graph shows that the linear regression model predicts departure delay somewhat accurately for actual departure delays greater than 2000 minutes (approximately 1 day). In addition, there is a larger cluster in the left hand side of the graph and this illustrates that when actual departure delays are lower, the models tend to over-predicted.



Visualization 5: Residual Plot

The fifth graph below is the residual of 15% of the dataset. The residual is calculated by subtracting the prediction delay minutes from actual departure delay minutes. In the graph there is a cluster of near negative residual, and this illustrates that the linear regression model tends to overpredict the delay time.



Milestone 6: Summary:

In conclusion, the creation of the ridge regression model somewhat accurately predicts the delay minutes of the airline. The model tends to overpredict flight delays, but this can be fixed by finding the right hyperparameter for more accurate predictions. Based on the exploration of historical flight delay data, the least delay airline tends to be regional and mide-size airlines (e.g.Cape Air, Peninsula Airway, Empire Airlines, etc). This is one aspect that travelers might consider when choosing which airline to travel with. In addition, data shows that the best months to travel are the autumn months (e.g. September-November), and the worst months to travel are during the summer months (e.g. June and July). These are the two factors that travelers need to consider when booking their flights for a stress-free trip.

GetHubLink: https://github.com/susanjiang02/flight_delay

Appendix A

```
mkdir .kaggle

mv kaggle. json . kaggle/

chmod 600 .kaggle/kaggle.json

sudo apt -y install zip

sudo apt -y install python3-pip python3.11-venv

python3 -m venv pythondev

python3 -m venv pythondev

cd pythondev

source bin/activate

pip3 install kaggle

kaggle datasets list

kaggle datasets download -d robikscube/flight-delay-dataset-20182022

Is -1

sudo apt -y install zip

unzip -1 flight-delay-dataset-20182022.zip

cloud storage buckets create gs://my-bigdata-project-sj --project=prisma-rose
--default-storage-cl SS=STANDARD
--location=us-centrall --uniform-bucket-level-access

gcloud storage cp Combined_Flights_ 2018.csv gs://my-bigdata-project-sj/landing/

gcloud storage cp Combined_Flights_ 2019.csv gs://my-bigdata-project-sj/landing/
```

```
gcloud storage cp Combined_Flights_ 2020.csv gs://my-bigdata-project-sj/landing/

gcloud storage cp Combined_Flights_ 2021.csv gs://my-bigdata-project-sj/landing/

gcloud storage cp Combined_Flights_ 2022.csv gs://my-bigdata-project-sj/landing/
```

Appendix B

```
sdf=spark.read.csv("gs://my-bigdata-project-sj/landing/", header=True, inferSchema=True)

print(f'The total of number of record is {sdf.count()}')

from pyspark.sql.functions import isnan, isnull, when, count, col

#Stats on the days people usually take flights
sdf.select("Year","Quarter","Month","DayOfMonth","DayOfWeek").summary().show()

#Get a list of numeric 'double' columns
numeric_columns = [c for c, t in sdf.dtypes if t == 'double']
print(numeric_columns)

# Loop through list of 'double' columns and get summary stats
from pyspark.sql.functions import isnan, isnull, when, count, col
for mycol in numeric_columns:
  # Get summary statistics
  sdf.select(mycol).summary().show()
  # Count null values
  sdf.select( [count(when(isnull(mycol),1))]).show()

from pyspark.sql.functions import min, max, year, month, dayofmonth, hour
sdf.select(min("FlightDate"), max("FlightDate")).show()

for col_name in sdf.columns:
  null_count=sdf.filter(col(col_name).isNull()).count()
  print(f'Total null records in {col_name} : {null_count}')

import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
pd.set_option('display.float_format', '{:.2f}'.format)
pd.set_option('display.width', 1000)
```

```
#Distribution of Departure Delay From 2018-2022
df=sdf.select("FlightDate","DepDelayMinutes").toPandas()
plt.plot(df['FlightDate'], df['DepDelayMinutes'])
plt.title("Distribution of Departure Delay From 2018-2022")
plt.xlabel("Flight Date")
plt.ylabel("DepDelayMinutes")
plt.show()

#Distribution of Arrival Delay From 2018-2022
df=sdf.select("FlightDate","ArrDelayMinutes").toPandas()
plt.plot(df['FlightDate'], df['ArrDelayMinutes'])
plt.title("Distribution of Arrival Delay From 2018-2022")
plt.xlabel("Flight Date")
plt.ylabel("ArrDelayMinutes")
plt.show()
```

Appendix C

```
sdf=spark.read.csv("gs://my-bigdata-project-sj/landing/", header=True, inferSchema=True)

sdf.show(truncate=False)

#drop all rows with null values

sdf = sdf.na.drop("any")
print(sdf.head())

threshold=2880
sdf=sdf.filter(col("DepDelayMinutes")<=threshold)
sdf.count()


#Create a loop to check all roles with nulls values are dropped
from pyspark.sql.functions import isnan, isnull, count, col
for col_name in sdf.columns:
    null_count=sdf.filter(col(col_name).isNull()).count()
    print(f'Total null records in {col_name} : {null_count}')

#save df as parquets into cleaning folder
output_file_path="gs://my-bigdata-project-sj/cleaned/cleaned_flight_delay.parquet"
sdf.write.parquet(output_file_path)
```

Appendix D

```
from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.regression import LinearRegression, GeneralizedLinearRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator, RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import numpy as np

#read the parquet from cleaned folder
sdf=spark.read.parquet("gs://my-bigdata-project-sj/cleaned/cleaned_flight_delay.parquet/",
header=True, inferSchema=True)
sdf.printSchema()



#Encode Boolean values as True=1 and False=0
sdf = sdf.withColumn("Cancelled_value", when(col("Cancelled") == True,
1.0).otherwise(0.0))
sdf = sdf.withColumn("Diverted_value", when(col("Diverted") == True, 1.0).otherwise(0.0))

# Create an indexer for the string based columns.
string_columns=["Airline","Origin","Dest"]
string_output=["AirlineIndex","OriginIndex","DestIndex"]
indexer = StringIndexer(inputCols=string_columns, outputCols=string_output,
handleInvalid="keep")

# Create an encoder for the indexes and the integer column.
indexed_columns=["AirlineIndex","OriginIndex","DestIndex","Year","Quarter","Month","Day
ofMonth","DayOfWeek"]
indexed_output=["AirlineVector","OriginVector","DestVector","YearVector","QuarterVector","
MonthVector","DayofMonthVector","DayOfWeekVector"]
encoder = OneHotEncoder(inputCols=indexed_columns,
                        outputCols=indexed_output, dropLast=True, handleInvalid="keep")

# Create an assembler for the individual feature vectors and the float/double columns
vector_columns=["AirlineVector","OriginVector","DestVector","Cancelled_value","Diverted_
value","YearVector","QuarterVector","MonthVector","DayofMonthVector","DayOfWeekVecto
r","ArrDelayMinutes"]
```

```
assembler = VectorAssembler(inputCols=vector_columns, outputCol="features")

# Create a Ridge Regression Estimator
ridge_reg = LinearRegression(labelCol='DepDelayMinutes',  elasticNetParam=0,
regParam=0.1)

# Create the pipeline Indexer
regression_pipe = Pipeline(stages=[indexer, encoder, assembler, ridge_reg])

transformed_sdf = regression_pipe.fit(sdf).transform(sdf)

# Review the transformed features
print("Transformed features")
transformed_sdf.select("Airline","Cancelled_value","Diverted_value","DepDelayMinutes","fe
atures").show(10, truncate=False)

#Saving feature data to trusted folder
output_file_path="gs://my-bigdata-project-sj/trusted/trusted_flight_delay.parquet"
transformed_sdf.write.parquet(output_file_path)



# Create a regression evaluator (to get RMSE, R2, RME, etc.)
evaluator = RegressionEvaluator(labelCol='DepDelayMinutes')

# take sample of data
sdf=sdf.sample(False, 0.01)

# Split the data into training and test sets
trainingData, testData = transformed_sdf.randomSplit([0.70, 0.3], seed=42)

# Create a grid to hold hyperparameters
grid = ParamGridBuilder()

#Hyperparameters
params = ParamGridBuilder() \
.addGrid(ridge_reg.fitIntercept, [True, False]) \
.addGrid(ridge_reg.regParam, [0.001,0.01, 0.1, 1, 10]) \
.addGrid(ridge_reg.elasticNetParam, [0, 0.25, 0.5, 0.75, 1]) \
.build()
```

```
# Build the parameter grid
grid = grid.build()

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=regression_pipe,
                    estimatorParamMaps=grid,
                    evaluator=evaluator,
                    numFolds=3)

# Train the models
all_models  = cv.fit(trainingData)

# Show the average performance over the three folds
print(f"Average metric {all_models.avgMetrics}")

# Get the best model from all of the models trained
bestModel = all_models.bestModel

# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(testData)

# Show the predicted flight delay
test_results.select("Airline","Origin","Dest","Cancelled_value","Diverted_value","DepDelay
Minutes", "prediction").show(truncate=False)

# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse}  R-squared:{r2}")

model_path =  'gs://my-bigdata-project-sj/models/flight_delay_ridge_regression_model'
bestModel.write().overwrite().save(model_path)
```

Appendix E

```
import io
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.ml import PipelineModel
from pyspark.sql.functions import *
from google.cloud import storage

#Load Model
model_path = 'gs://my-bigdata-project-sj/models/flight_delay_ridge_regression_model'
pipeline = PipelineModel.load(model_path)

#Extract model from Pipeline
mymodel = pipeline.stages[-1]

print("Coefficients: ", mymodel.coefficients)

print("Paramter: ", mymodel.extractParamMap)

print("ElasticParam: ",mymodel.getElasticNetParam())

print("RegParam: ",mymodel.getRegParam())

#Load in dataset from trusted folders
file_path="gs://my-bigdata-project-sj/trusted/trusted_flight_delay.parquet"
transformed_sdf = spark.read.parquet(file_path)

trainingData, testData = transformed_sdf.randomSplit([0.70, 0.3], seed=42)

#dataframe already has prediction and feature column
transformed_sdf.printSchema()

#Create dataframe for delay count by month
monthly_delay_sdf=transformed_sdf.where(col('DepDelayMinutes') >=
60).groupby('Month').count().sort('Month').select('Month','count')
monthly_delay_sdf.show(5)

#convert spark df to pandas
```

```
monthly_df=monthly_delay_sdf.toPandas()

#Graph that shows which month has the month delays
fig = plt.figure(facecolor='white')
#Graph month and number of delay
plt.bar(monthly_df['Month'], monthly_df['count'])

#Graph labels
plt.xlabel("Month")
plt.ylabel("Number of Delay > 1hr ")
plt.title("Number of Delay by Month")
plt.xticks([ 1, 2, 3, 4 , 5, 6, 7, 8, 9, 10, 11, 12], ['January', 'February', 'March', 'April', 'May',
'June', 'July','August', 'September', 'October', 'November', 'December'])
plt.xticks(rotation=90, ha='right')
plt.savefig("delay_month_matplotlib.png")
plt.show()

# Save the plot to GCS
import io
from google.cloud import storage

# Create a buffer
img_data = io.BytesIO()
# Write the Matplotlib figure to the buffer
fig.savefig(img_data, format='png', bbox_inches='tight')

# Rewind the pointer to the start of the data
img_data.seek(0)
# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket('my-bigdata-project-sj')

# Create a blob to hold the data.
blob = bucket.blob("/figures/delay_month.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)

#Create spark df with the ten top airline with the most delay
```

```
most_airline_delay_sdf=transformed_sdf.where(col('DepDelayMinutes') >=
60).groupby('Airline').count().orderBy(col("count").desc()).limit(10).select('Airline','count')
airline_delay_sdf.show(10)

#convert to pandas
most_airline_df=most_airline_delay_sdf.toPandas()

#Graph that shows the Airline with the most delays
fig2 = plt.figure(facecolor='white')
plt.bar(most_airline_df['Airline'], most_airline_df['count'])

plt.xlabel("Airline")
plt.ylabel("Number of Delay > 1hr ")
plt.title("Most Number of Delay by Airline")
plt.xticks(rotation=90, ha='right')
plt.savefig("most_airline_delay.png")
plt.show()

#save figure to GCS
# buffer to hold the figure
img_data = io.BytesIO()
# Write the Matplotlib figure to the buffer
fig2.savefig(img_data, format='png', bbox_inches='tight')

# Rewind the pointer to the start of the data
img_data.seek(0)
# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket('my-bigdata-project-sj')

# Create a blob to hold the data.
blob = bucket.blob("/figures/most_airline_delay.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)

# Create sdf of the top 10 airlines with the least delays
min_airline_delay_sdf=transformed_sdf.where(col('DepDelayMinutes') >=
60).groupby('Airline').count().orderBy(col("count").asc()).limit(10).select('Airline','count')
min_airline_delay_sdf.show(10)
```

```
min_airline_df=min_airline_delay_sdf.toPandas()

#Graph that shows the top 10 Airline with the most delays
fig3 = plt.figure(facecolor='white')
plt.bar(min_airline_df['Airline'], min_airline_df['count'])

plt.xlabel("Airline")
plt.ylabel("Number of Delay > 1hr ")
plt.title("Least Number of Delay by Airline")
plt.xticks(rotation=50, ha='right')
plt.savefig("least_airline_delay.png")
plt.show()

#Save figure to GCS
#  buffer to hold the figure
img_data = io.BytesIO()
# Write the Matplotlib figure to the buffer
fig3.savefig(img_data, format='png', bbox_inches='tight')

# Rewind the pointer to the start of the data
img_data.seek(0)
# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket('my-bigdata-project-sj')

# Create a blob to hold the data.
blob = bucket.blob("/figures/least_airline_delay.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)

#create a 25% sample of the sdf
sample_sdf=transformed_sdf.sample(False,0.15)

regression_df=sample_sdf.select('DepDelayMinutes','prediction').toPandas()

# Use seaborn plots to illustrate 'DepDelayMinutes' vs. 'prediction'
sns.set_style("white")
relationship_plot=sns.lmplot(x='DepDelayMinutes', y='prediction', data=regression_df)
```

```
# Saving Seaborn Relationship plot to GCS
img_data = io.BytesIO()
# Figure to the buffer
relationship_plot.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data
img_data.seek(0)

# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket('my-bigdata-project-sj')

# Create a blob to hold the data.
blob = bucket.blob("/figures/relationship_plot.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)

#Calculate residuals
regression_df['residuals'] = regression_df['DepDelayMinutes'] - regression_df['prediction']

# Graph residual plots
sns.set_style("white")

# Create a relationship plot between tip and prediction
residual_plot=sns.regplot(x = 'prediction', y = 'residuals', data = regression_df, scatter = True,
color = 'red')

# Saving Seaborn Residual plot to GCS
img_data = io.BytesIO()
# Figure to the buffer
residual_plot.savefig(img_data, format='png', bbox_inches='tight')
# Rewind the pointer to the start of the data
img_data.seek(0)

# Connect to Google Cloud Storage
storage_client = storage.Client()
# Point to the bucket
bucket = storage_client.get_bucket('my-bigdata-project-sj')
```

```
# Create a blob to hold the data.
blob = bucket.blob("/figures/residual_plot.png")
# Upload the img_data contents to the blob
blob.upload_from_file(img_data)
```