Untitled

April 24, 2023

[2]:
```
# Set up environment variables
PROJECT = !gcloud config get-value project
PROJECT = PROJECT[0]
import os
BUCKET = "{}-dsongcp".format(PROJECT)
REGION = "us-central1"
os.environ["BUCKET"] = BUCKET
```

0.1 Exploration using BigQuery

[3]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import google.cloud.bigquery as bigquery

bq = bigquery.Client()
```

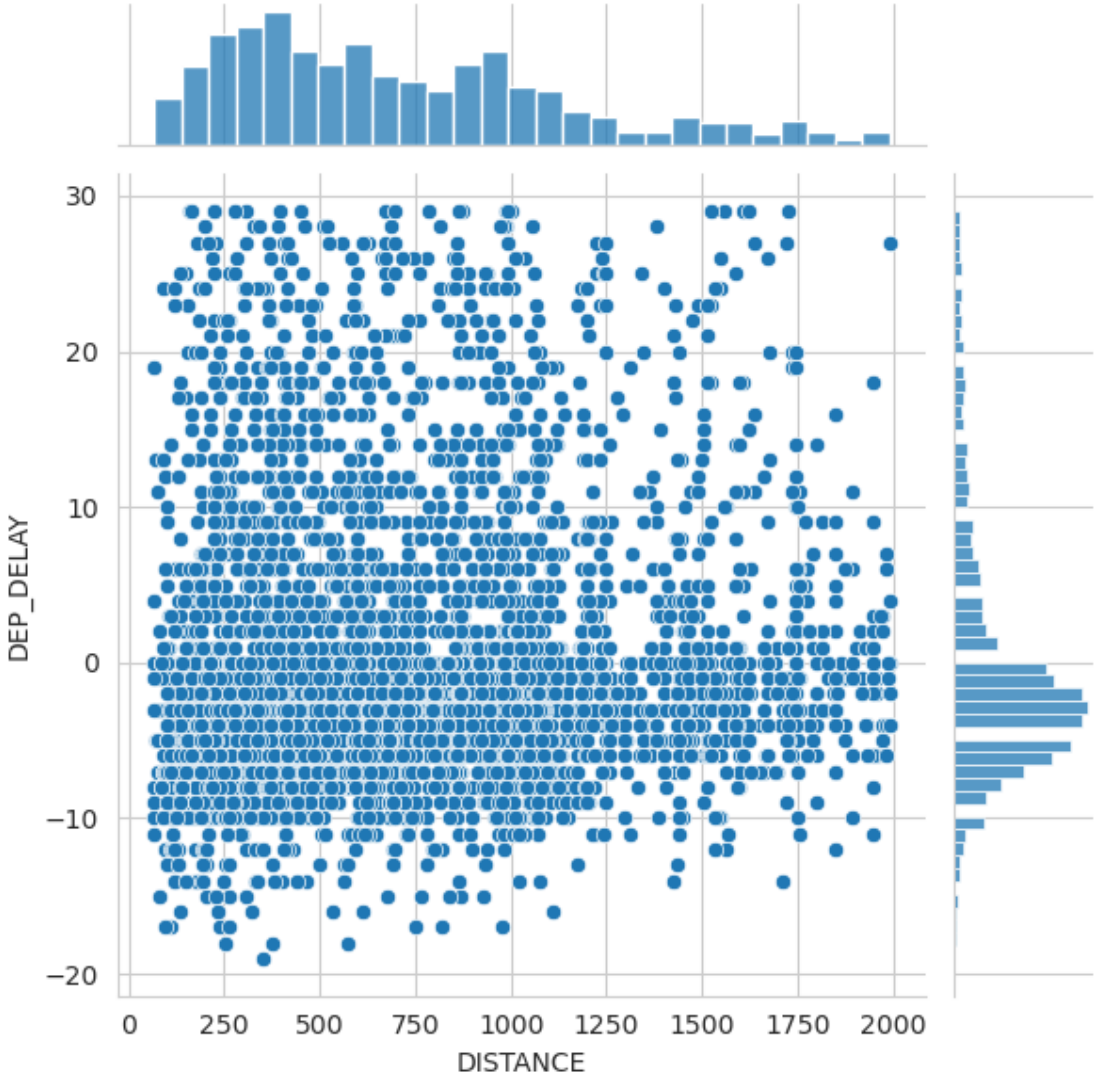
[5]:
```
sql = """
SELECT DISTANCE, DEP_DELAY
FROM dsongcp.flights_tzcorr
WHERE RAND() < 0.001 AND dep_delay > -20 AND dep_delay < 30 AND distance < 2000
"""
df = bq.query(sql).to_dataframe()
```

[7]:
```
print(df.head())
```

```
   DISTANCE  DEP_DELAY
0     120.0       -1.0
1     440.0       -6.0
2     228.0       -3.0
3    1616.0       -2.0
4    1009.0       -3.0
```

[9]:
```
sns.set_style("whitegrid")
g = sns.jointplot(x=df["DISTANCE"], y=df["DEP_DELAY"])
kind = "hex"
```

```
height = 10
joint_kws = {"gird_size": 20}
```



[11]:
```
from pyspark.sql import SparkSession
# creating spark session
spark = SparkSession .builder .appName("Bayes classification using Spark") .
 ↪getOrCreate()
```

[14]:
```
# Read the time-corrected JSON files from the Google cloud storage bucket:
inputs = "gs://{}/flights/tzcorr/all_flights-*".format(BUCKET)
flights = spark.read.json(inputs)
```

[15]:
```
# Employ SQL on the dataframe by creating a temporary view (it is available␣
↪only within this Spark session):
flights.createOrReplaceTempView("flights")
```

[16]:
```
# Employ SQL to query the flights view, for example by using this command:
results = spark.sql("SELECT COUNT(*) FROM flights WHERE dep_delay > -20 AND␣
↪CAST(distance AS FLOAT) < 2000")
results.show()
```

```
[Stage 2:=====================================================>   (32 + 2) / 34]

+--------+
|count(1)|
+--------+
| 5357273|
+--------+
```

[17]:
```
# Create a CSV file of the training days Google BigQuery table and save data to␣
↪cloud storage bucket:
sql = """SELECT * FROM dsongcp.trainday"""
df = bq.query(sql).to_dataframe()
df.to_csv("trainday.csv", index= False)
```

[18]:
```
%%bash
gsutil cp trainday.csv gs://${BUCKET}/flights/trainday.csv
```

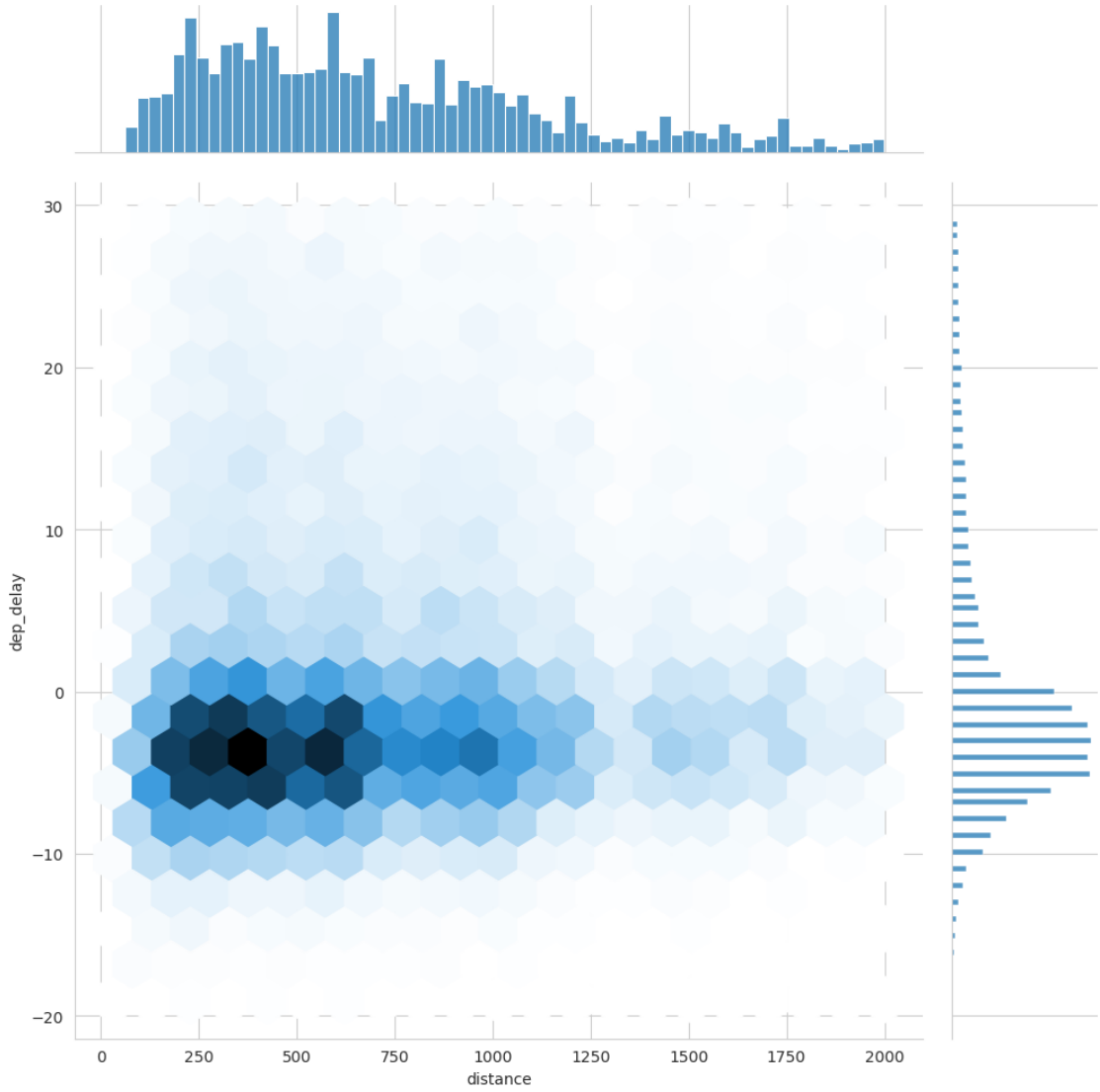
```
Copying file://trainday.csv [Content-Type=text/csv]…
/ [1 files][  5.8 KiB/  5.8 KiB]
Operation completed over 1 objects/5.8 KiB.
```

[20]:
```
# Create the traindays dataframe from the CSV file trainday.csv using the␣
↪following code:
from pyspark.sql.types import StructType, StructField, StringType, BooleanType
schema = StructType([
    StructField("FL_DATE", StringType(), True),
    StructField("is_train_day", BooleanType(), True)
])
traindays = spark.read .option("header", "true") .option("inferSchema", "true")␣
↪.csv("gs://{}/flights/trainday.csv".format(BUCKET))
traindays.createOrReplaceTempView("traindays")
```

[21]:
```
# restrict the flights dataframe to contain only training days using an SQL␣
↪join operation:
```

```
statement = """SELECT f.FL_DATE AS date, CAST(distance as FLOAT) AS distance,␣
 ↪dep_delay,
IF(arr_delay < 15, 1, 0) AS ontime
FROM flights f JOIN traindays t
ON f.FL_DATE == t.FL_DATE WHERE
t.is_train_day AND f.dep_delay IS NOT NULL ORDER BY f.dep_delay DESC """
flights = spark.sql(statement)
```

[22]:
```
# Create a hexbin plot using Spark (repeat of what you did in BigQuery, except␣
 ↪that you now restrict to train days only):
df = flights[(flights['distance'] < 2000) & (flights['dep_delay'] > -20) &␣
 ↪(flights['dep_delay'] < 30)]
pdf = df.sample(False, 0.02, 20).toPandas()  # to 100,000 rows approx on␣
 ↪complete dataset
g = sns.jointplot(x=pdf['distance'], y=pdf['dep_delay'], kind="hex", height=10,␣
 ↪joint_kws={'gridsize':20})
```



[23]:
```
# Finding thresholds that make the two quantized variables uniformly␣
 ↪distributed is straightforward using the approximate quantiles method:
distthresh = flights.approxQuantile('distance', list(np.arange(0, 1.0, 0.2)), 0.
 ↪02)
distthresh[-1] = float('inf')
print(distthresh)
```

```
[Stage 16:=========================>                              (4 + 5) / 9]
```

```
[31.0, 331.0, 541.0, 813.0, inf]
```

[24]:
```
# quantize the departure delay thresholds into equal boundaries:
delaythresh = flights.approxQuantile("dep_delay", list(np.arange(0, 1.0, 0.2)),␣
↪0.05)
delaythresh[-1] = float("inf")
print(delaythresh)
```

```
[Stage 22:=============================================>             (7 + 2) / 9]
[-82.0, -5.0, -3.0, 0.0, inf]
```

0.2 Bayes classification

[25]:
```
# Find the flights that belong to the mth distance bin and nth delay bin by␣
↪slicing the full set of flights:
import pyspark.sql.functions as F
import pandas as pd

df = pd.DataFrame(columns = ["dist_thresh", "delay_thresh", "frac_ontime"])

for m in range(0, 2):
    for n in range(0, len(delaythresh)-1):
        bdf = flights[(flights["distance"] >= distthresh[m]) &␣
↪(flights['distance'] < distthresh[m+1])
            & (flights['dep_delay'] >= delaythresh[n]) & (flights['dep_delay']␣
↪< delaythresh[n+1])]
        ontime_frac = bdf.agg(F.sum("ontime")).collect()[0][0] / bdf.agg(F.
↪count("ontime")).collect()[0][0]
        print(m, n, ontime_frac)
        df = df.append({
            "dist_thresh": distthresh[m],
            "delay_thresh": delaythresh[n],
            "frac_ontime": ontime_frac}, ignore_index = True)
```

```
0 0 0.9795324781979586

0 1 0.9732700511013729

0 2 0.9644298176408268

0 3 0.5500322747224374
```

```
1 0 0.9779420684019332
```

```
1 1 0.9751335815133887
```

```
1 2 0.9681868803476844
```

```
[Stage 85:=====================================================> (33 + 1) / 34]
```

```
1 3 0.5763487238807807
```

[26]:
```
# Fine-tune the delay threshold around the decision boundary:
delaythresh = range(10, 20)
df = pd.DataFrame(columns=['dist_thresh', 'delay_thresh', 'frac_ontime'])
for m in range(0, len(distthresh)-1):
    for n in range(0, len(delaythresh)-1):
        bdf = flights[(flights['distance'] >= distthresh[m])
             & (flights['distance'] < distthresh[m+1])
             & (flights['dep_delay'] >= delaythresh[n])
             & (flights['dep_delay'] < delaythresh[n+1])]
        ontime_frac = bdf.agg(F.sum('ontime')).collect()[0][0] / bdf.agg(F.
 ↪count('ontime')).collect()[0][0]
        print (m, n, ontime_frac)
        df = df.append({
            'dist_thresh': distthresh[m],
            'delay_thresh': delaythresh[n],
            'frac_ontime': ontime_frac
        }, ignore_index=True)
```

```
0 0 0.8137951450562463
```

```
0 1 0.7901898734177215
```

```
0 2 0.7645924627519719
```

```
0 3 0.7214854111405835
```

```
0 4 0.6870363139398672
```

```
0 5 0.6392009987515606
```

0 6 0.6092511013215859

0 7 0.556468654613759

0 8 0.5275194772555919

1 0 0.834435261707989

1 1 0.8127323420074349

1 2 0.7945911139729556

1 3 0.7678386763185109

1 4 0.7386728505173029

1 5 0.6957947530864198

1 6 0.6517402376910016

1 7 0.6123407109322603

1 8 0.5716976689427832

2 0 0.8324673678874668

2 1 0.8211325966850829

2 2 0.7979426891991183

2 3 0.7733044286156705

```
2 4 0.7498322147651006

2 5 0.7072135785007072

2 6 0.6897522522522522

2 7 0.6706309211852449

2 8 0.6179225202213708

3 0 0.83638779691047

3 1 0.8149313328090023

3 2 0.8052236369572314

3 3 0.7886468370339338

3 4 0.7651509419686767

3 5 0.7372527385924816

3 6 0.7196197061365601

3 7 0.6871413199426112
[Stage 373:=====================================================>   (32 + 2) / 34]
3 8 0.665420739888825
```

[27]:
```
# To find the delay threshold for each distance threshold where the value is␣
 ↪closest to the 0.70 decision boundary, run the following code:
df['score'] = abs(df['frac_ontime'] - 0.7)
```

```
bayes = df.sort_values(['score']).groupby('dist_thresh').head(1).
 ↪sort_values('dist_thresh')
print(bayes)
```

```
    dist_thresh  delay_thresh  frac_ontime     score
4          31.0          14.0     0.687036  0.012964
14        331.0          15.0     0.695795  0.004205
23        541.0          15.0     0.707214  0.007214
34        813.0          17.0     0.687141  0.012859
```

[28]:
```
# Write out the table bayes as a CSV file to Google cloud storage bucket:
bayes.to_csv('gs://{}/flights/bayes.csv'.format(BUCKET), index=False)
!gsutil cat gs://{BUCKET}/flights/bayes.csv
```

```
dist_thresh,delay_thresh,frac_ontime,score
31.0,14.0,0.6870363139398672,0.012963686060132762
331.0,15.0,0.6957947530864198,0.004205246913580152
541.0,15.0,0.7072135785007072,0.007213578500707252
813.0,17.0,0.6871413199426112,0.012858680057388772
```

0.3 Evaluating the Model

[29]:
```
diststhresh[-1] = 100000
for m in range(0, len(diststhresh)-1):
    statement = """
SELECT
  '{0:.0f}-{1:.0f} miles' AS bin,
  ROUND(SUM(IF(dep_delay < {2:f} AND arr_delay < 15, 1, 0))/COUNT(*), 2) AS␣
 ↪correct_nocancel,
  ROUND(SUM(IF(dep_delay >= {2:f} AND arr_delay < 15, 1, 0))/COUNT(*), 2) AS␣
 ↪false_positive,
  ROUND(SUM(IF(dep_delay < {2:f} AND arr_delay >= 15, 1, 0))/COUNT(*), 2) AS␣
 ↪false_negative,
  ROUND(SUM(IF(dep_delay >= {2:f} AND arr_delay >= 15, 1, 0))/COUNT(*), 2) AS␣
 ↪correct_cancel,
  COUNT(*) AS total_flights
FROM flights f
JOIN traindays t
ON f.FL_DATE == t.FL_DATE
WHERE
  t.is_train_day == 'False' AND
  f.distance >= {0:f} AND f.distance < {1:f}
""".format( diststhresh[m], diststhresh[m+1], bayes[ bayes['dist_thresh'] ==␣
 ↪diststhresh[m] ]['delay_thresh'].values[0] )
    eval_flights = spark.sql(statement)
    eval_flights.show()
```

```
+------------+---------------+-------------+-------------+-------------+----
---------+
|
bin|correct_nocancel|false_positive|false_negative|correct_cancel|total_flights|
+------------+---------------+-------------+-------------+-------------+----
---------+
|31-331 miles|           0.79|         0.03|         0.03|         0.13|
326713|
+------------+---------------+-------------+-------------+-------------+----
---------+



+-------------+----------------+-------------+-------------+-------------+---
----------+
|
bin|correct_nocancel|false_positive|false_negative|correct_cancel|total_flights|
+-------------+----------------+-------------+-------------+-------------+---
----------+
|331-541 miles|            0.79|         0.03|         0.03|         0.14|
311447|
+-------------+----------------+-------------+-------------+-------------+---
----------+



+-------------+----------------+-------------+-------------+-------------+---
----------+
|
bin|correct_nocancel|false_positive|false_negative|correct_cancel|total_flights|
+-------------+----------------+-------------+-------------+-------------+---
----------+
|541-813 miles|            0.77|         0.04|         0.04|         0.14|
327936|
+-------------+----------------+-------------+-------------+-------------+---
----------+

[Stage 389:=================================================> (33 + 1) / 34]
+---------------+---------------+-------------+-------------+-------------+
-------------+
|
bin|correct_nocancel|false_positive|false_negative|correct_cancel|total_flights|
+---------------+---------------+-------------+-------------+-------------+
-------------+
|813-100000 miles|           0.77|         0.04|         0.05|         0.13|
```

```
654370|
+----------------+----------------+--------------+--------------+--------------+
-------------+
```

[ ]: