Finding Flight Delay Trends

DAT500 Project Group 17

Bhakti Prabhu & Stephan F. W. Brandasu

Faculty of Science and Technology

Why & What

the use case

As a traveller:

1. Not miss any important meetings/events/functions.
2. Pre-plan journey
3. Have idea of buffer time while flight booking

As an airline:

1. know when to increase workforce.
2. Opportunity to improve over competitors.

Objective:

1. visualise delay trends over time
2. how well do the airlines catch up when departing late
3. how often are flights cancelled
4. are longer flights delayed more often

The Dataset

From 20GB of unstructured .txt files to 6GB of .csv

```
ubuntu@namenode:~$ hadoop fs -du -h /txt
382.2 M  764.4 M  /txt/2018-01.txt
349.4 M  698.9 M  /txt/2018-02.txt
410.1 M  820.2 M  /txt/2018-03.txt
400.0 M  799.9 M  /txt/2018-04.txt
413.9 M  827.9 M  /txt/2018-05.txt
420.6 M  841.2 M  /txt/2018-06.txt
433.6 M  867.2 M  /txt/2018-07.txt
428.1 M  856.2 M  /txt/2018-08.txt
392.9 M  785.7 M  /txt/2018-09.txt
414.5 M  828.9 M  /txt/2018-10.txt
394.6 M  789.3 M  /txt/2018-11.txt
399.7 M  799.4 M  /txt/2018-12.txt
391.6 M  783.3 M  /txt/2019-01.txt
358.1 M  716.1 M  /txt/2019-02.txt
423.9 M  847.7 M  /txt/2019-03.txt
410.5 M  821.0 M  /txt/2019-04.txt
427.2 M  854.4 M  /txt/2019-05.txt
428.0 M  856.0 M  /txt/2019-06.txt
```

```
ubuntu@namenode:~$ hadoop fs -du -h /csv
1.4 G    2.8 G   /csv/2018.csv
1.4 G    2.9 G   /csv/2019.csv
936.6 M  1.8 G   /csv/2020.csv
1.1 G    2.3 G   /csv/2021.csv
1.2 G    2.5 G   /csv/2022.csv
5.9 G    11.8 G  /csv/all-years.csv
```

The Dataset

- year
- quarter
- month
- day_of_month
- day_of_week
- fl_date
- op_unique_carrier
- tail_num
- op_carrier_fl_num
- origin_airport_id
- origin_airport_seq_id
- origin_city_market_id
- origin
- origin_city_name
- origin_state_nm
- dest_airport_id
- dest_airport_seq_id
- dest_city_market_id
- dest
- dest_city_name
- dest_state_nm
- dep_delay
- dep_delay_new
- dep_del15
- arr_delay
- arr_delay_new
- arr_del15
- cancelled
- cancellation_code
- diverted
- air_time
- distance
- distance_group
- carrier_delay
- weather_delay
- nas_delay
- security_delay
- late_aircraft_delay

- year
- month
- fl_date
- op_unique_carrier
- origin_airport_id
- dest_airport_id
- dep_delay_new
- arr_delay_new
- cancelled
- diverted
- air_time

- year
- month
- op_unique_carrier
- origin_airport_id
- dest_airport_id
- max_arr_delay
- max_arr_delay_fl_date
- avg_arr_delay
- med_arr_delay
- avg_time_recovered
- nr_diverted
- avg_airtime
- flight_count
- nr_cancelled

Mapping

```
import sys

row = []

for line in sys.stdin:
  line = line.strip().replace(',','').split()

  if line[0] == "LATE_AIRCRAFT_DELAY":
    try:
      data = line[1]
    except IndexError:
      data = ' '
    row.append(data)
    print(','.join(row))
    row = []
  else:
    try:
      data = ' '.join(line[1:])
    except IndexError:
      data = ' '
    row.append(data)
```

- create array of each row
- remove any unwanted commas from cell
- check for last column of row
- catch error in case of no data in current column
- print row

Reading the data

```
flight_data = spark.read.csv('hdfs://namenode:9000/csv/'+sys.argv[1]+'.csv', schema=flightSchema)\
.withColumn('FL_DATE', to_date(to_timestamp('FL_DATE', 'M/d/yyyy h:mm:ss a')))

flight_data = flight_data.select( 'year'
                        , 'month'
                        , 'fl_date'
                        , 'op_unique_carrier'
                        , 'origin_airport_id'
                        , 'dest_airport_id'
                        , 'dep_delay_new'
                        , 'arr_delay_new'
                        , 'cancelled'
                        , 'diverted'
                        , 'air_time')

flight_data = flight_data.na.drop(subset=['year', 'origin_airport_id', 'dest_airport_id', 'fl_date'])
flight_data = flight_data.fillna({'arr_delay_new': 0.0})
```

- select only relevant columns
- drop any rows that are missing essential information
- fill 0.0 in rows to avoid issues during calculations

Manipulating the data - step 1

```
# grab the fl_date of the flight with the highest delay for a given group
windowSpec = Window.partitionBy(
      'year'
    , 'month'
    , 'op_unique_carrier'
    , 'origin_airport_id'
    , 'dest_airport_id').orderBy(col('arr_delay_new').desc())

arr_delay_dates = flight_data.withColumn(
        'rank'
    , rank().over(windowSpec)
).filter(
    col('rank') == 1
).groupBy(
      'year'
    , 'month'
    , 'op_unique_carrier'
    , 'origin_airport_id'
    , 'dest_airport_id'
).agg(
round(max('arr_delay_new'), 2).alias('max_arr_delay')
, first('fl_date').alias('max_arr_delay_fl_date')
)
```

- create a window sorted by the arrival delay
- select the top delay date by only selecting single row

Manipulating the data - step 2

```
flight_data = flight_data.groupBy('year'
, 'month'
, 'op_unique_carrier'
, 'origin_airport_id'
, 'dest_airport_id').agg( round(avg('arr_delay_new'), 2).alias('avg_arr_delay')
    , round(percentile_approx('arr_delay_new', 0.5), 2).alias('med_arr_delay')
    , round(avg(col('dep_delay_new') - col('arr_delay_new')), 2).alias('avg_time_recovered')
    , sum('diverted').alias('nr_diverted')
    , round(avg('air_time'), 2).alias('avg_airtime')
    , count('*').alias('flight_count')
    , sum('cancelled').alias('nr_cancelled'))

flight_data = arr_delay_dates.join(flight_data
    , on=['year', 'month', 'op_unique_carrier', 'origin_airport_id', 'dest_airport_id']
    , how='left')
```

- do a groupby select for to grab delay statistics
- each result is rounded to 2 decimals to avoid ugly numbers
- join the result of this operation with the previous one

Upserting into the DeltaTable

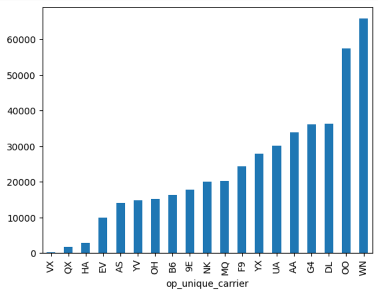
```python
# Checking if Table exists
if DeltaTable.isDeltaTable(spark, "hdfs://namenode:9000/spark-warehouse/sample_flight_table"):
    # Perform the upsert operation
    deltaDF = DeltaTable.forPath(spark, "hdfs://namenode:9000/spark-warehouse/
      sample_flight_table")
    merge_condition = "existing.year = upsert.year \
    AND existing.month = upsert.month \
    AND existing.op_unique_carrier = upsert.op_unique_carrier \
    AND existing.origin_airport_id = upsert.origin_airport_id \
    AND existing.dest_airport_id = upsert.dest_airport_id "

    deltaDF.alias('existing') \
        .merge(flight_data.alias('upsert'), merge_condition) \
        .whenMatchedUpdateAll() \
        .whenNotMatchedInsertAll() \
        .execute()
else:
    # Create new delta table
    flight_data.write.format("delta").mode("overwrite").saveAsTable("sample_flight_table")
```

Skew & Spill

Optmization (spark-defaults.conf)

Spill	Memory	Disk
Before	3.5 GiB	177.8 MiB
After	128.4 MiB	7.1 MiB

```
spark.executor.memory                   6g
spark.exector.instances                 3
spark.executor.cores                    4
spark.sql.shuffle.partition             64
spark.sql.adaptive.skewedJoin.enabled   true
spark.sql.adaptive.skewJoin.enabled     true
```



Serialization

Using UDF's to clean the data

```
def replace_null(value, default):
  if value is None:
    return default
  return value

def drop_null(*cols):
  for col in cols:
    if col is None:
      return False
  return True

replace_null_udf = udf(lambda value, default:
      replace_null(value, default), FloatType
      ())
drop_null_udf = udf(lambda *cols: drop_null(*
      cols), BooleanType())

flight_data = flight_data.filter(drop_null_udf
      (*[col(c) for c in ['year', '
      origin_airport_id', 'dest_airport_id', '
      fl_date']]))
flight_data = flight_data.withColumn('
      arr_delay_new', replace_null_udf(col('
      arr_delay_new'), lit(0.0)))
```

Performance difference between each type of join

	normal	**udf**
minutes	2.8	7.1

Shuffle

sort merge join

```
arr_delay_dates = arr_delay_dates.sort(['year', 'month', 'op_unique_carrier', 'origin_airport_id', 'dest_airport_id'])
flight_data = flight_data.sort(['year', 'month', 'op_unique_carrier', 'origin_airport_id', 'dest_airport_id'])

# join the highest delay with the res of the group
flight_data = arr_delay_dates.join(flight_data
, on=['year', 'month', 'op_unique_carrier', 'origin_airport_id', 'dest_airport_id']
, how='left')
```

broadcast join

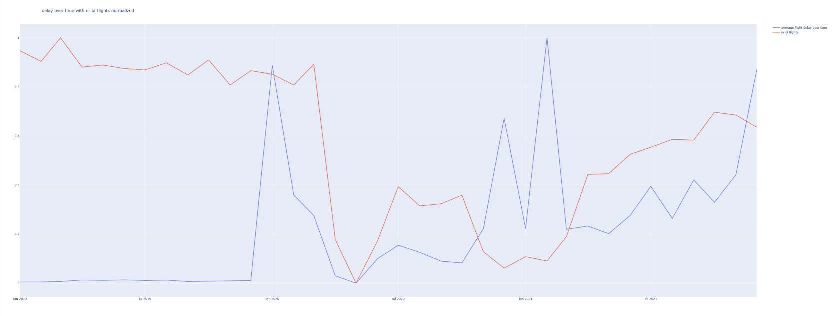
```
flight_data = arr_delay_dates.join(
    broadcast(flight_data)
    , on=['year', 'month', 'op_unique_carrier', 'origin_airport_id', 'dest_airport_id']
    , how='left')
```

	Sort Merge	Broadcast
Minutes	4.1	3.2

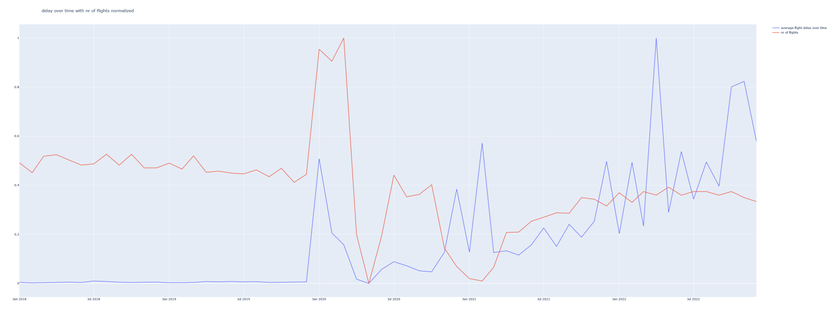
Graphs from the delta table

```
builder = SparkSession.builder.appName('flight_plot')
spark = configure_spark_with_delta_pip(builder).getOrCreate()

flight_data = spark.read.format('delta').load('hdfs://namenode:9000/spark-warehouse/
       flight_data_table')

flight_data = flight_data.filter((flight_data.op_unique_carrier == 'AA') & (flight_data.
       origin_airport_id == 12892) & (flight_data.dest_airport_id == 12478))

flight_data = flight_data.orderBy('year', 'month')

flight_data = flight_data.withColumn('year_month', concat('year', lit('-'), 'month'))

flight_data = flight_data.toPandas()

avgdelay = normalize(flight_data, "avg_arr_delay")
flightcount = normalize(flight_data, "flight_count")

data = [
plt.Scatter(x=flight_data.year_month
            , y=avgdelay
            , name='average flight delay over time'
            , text=flight_data.avg_arr_delay),
plt.Scatter(x=flight_data.year_month
            , y=flightcount
            , name='nr of flights'
            , text=flight_data.flight_count),
]

fig = plt.Figure(data, layout_title_text='delay over time with nr of flights normalized')
plot(fig, filename='plot.html')
```

A Graph!


delay over time with nr of flights normalized
average flight delay over time
nr of flights
1
0.8
0.6
0.4
0.2
0
Jan 2019
Jul 2019
Jan 2020
Jul 2020
Jan 2021
Jul 2021

A Graph! with a new year



A Graph! with bad information for 1 year year