02_Data_Exploration_with_Athena

April 15, 2022

1 Packages

[1]:
```
import boto3
import sagemaker
from pyathena import connect

import numpy as np
import pandas as pd
import seaborn as sns
from pathlib import Path

import matplotlib.pyplot as plt

%matplotlib inline
%config InlineBackend.figure_format='retina'

sns.set_style = "seaborn-whitegrid"

sns.set(
    rc={
        "font.style": "normal",
        "axes.facecolor": "white",
        "grid.color": ".8",
        "grid.linestyle": "-",
        "figure.facecolor": "white",
        "figure.titlesize": 20,
        "text.color": "black",
        "xtick.color": "black",
        "ytick.color": "black",
        "axes.labelcolor": "black",
        "axes.grid": True,
        "axes.labelsize": 10,
        "xtick.labelsize": 10,
        "font.size": 10,
        "ytick.labelsize": 10,
    }
)
```

2 Set-up

[2]:
```
sess = sagemaker.Session()
bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = boto3.Session().region_name
```

[3]:
```
ingest_create_athena_db_passed = False
```

3 Create Athena Database

[4]:
```
# Set Athena database & table
database_name = "ads508"
table_name = "flight_departure_delays"
```

[5]:
```
# Set S3 staging directory -- this is a temporary directory used for Athena␣
↪queries
s3_staging_dir = "s3://{0}/ads508/athena/staging".format(bucket)
```

[6]:
```
conn = connect(region_name=region, s3_staging_dir=s3_staging_dir)
```

[7]:
```
statement = "CREATE DATABASE IF NOT EXISTS {}".format(database_name)
print(statement)
```

```
CREATE DATABASE IF NOT EXISTS ads508
```

[8]:
```
pd.read_sql(statement, conn)
```

[8]:
```
Empty DataFrame
Columns: []
Index: []
```

4 Verify The Database Has Been Created Succesfully

[9]:
```
statement = "SHOW DATABASES"

df_show = pd.read_sql(statement, conn)
df_show.head(5)
```

[9]:

	database_name
0	ads508
1	default
2	dsoaws

[10]:
```
if database_name in df_show.values:
    ingest_create_athena_db_passed = True
```

[11]:
```
%store ingest_create_athena_db_passed
```

```
Stored 'ingest_create_athena_db_passed' (bool)
```

5 Download Data from Public S3 Bucket

[12]:
```
# Public Flight Data
s3_client = boto3.client("s3")

BUCKET='ads-508-airline'
KEY='transformed/ON_TIME_REPORTING_12.csv'

response = s3_client.get_object(Bucket=BUCKET, Key=KEY)
dec_flight = pd.read_csv(response.get("Body"))
dec_flight.head()
```

[12]:

	DAY_OF_MONTH	DAY_OF_WEEK	OP_UNIQUE_CARRIER	TAIL_NUM	ORIGIN	DEST
0	8	7	WN	N8651A	STL	SAN
1	8	7	WN	N939WN	STL	SAT
2	8	7	WN	N7741C	STL	SAT
3	8	7	WN	N550WN	STL	SEA
4	8	7	WN	N8319F	STL	SFO

	DEP_DEL15	DEP_TIME_BLK	ARR_TIME_BLK	CANCELLED	CRS_ELAPSED_TIME	DISTANCE
0	0.0	1100-1159	1300-1359	0.0	245.0	1557.0
1	0.0	1200-1259	1400-1459	0.0	145.0	786.0
2	0.0	2100-2159	0001-0559	0.0	140.0	786.0
3	0.0	0900-0959	1200-1259	0.0	275.0	1709.0
4	1.0	1800-1859	2000-2059	0.0	270.0	1735.0

	DISTANCE_GROUP	CARRIER_DELAY	WEATHER_DELAY	NAS_DELAY	SECURITY_DELAY
0	7	0.0	0.0	18.0	0.0
1	4	NaN	NaN	NaN	NaN
2	4	NaN	NaN	NaN	NaN
3	7	NaN	NaN	NaN	NaN
4	7	NaN	NaN	NaN	NaN

	LATE_AIRCRAFT_DELAY
0	0.0
1	NaN
2	NaN
3	NaN
4	NaN

[13]:
```
file_path = Path('../src/data/transformed_data/ON_TIME_REPORTING_12.csv')

if file_path.is_file():
    None
else:
    dec_flight.to_csv('../src/data/transformed_data/ON_TIME_REPORTING_12.csv',␣
 ↪index=False)
```

6 Set S3 Destination Location(Our S3 Private Bucket)

[14]:
```
s3_private_path_csv = "s3://{}/ads508/data".format(bucket)
print(s3_private_path_csv)
```

```
s3://sagemaker-us-east-1-229768475194/ads508/data
```

[15]:
```
%store s3_private_path_csv
```

```
Stored 's3_private_path_csv' (str)
```

7 Copy Downloaded Local Data to our Private S3 Bucket in this Account

[16]:
```
!aws s3 cp $file_path $s3_private_path_csv/
```

```
upload: ../src/data/transformed_data/ON_TIME_REPORTING_12.csv to s3://sagemaker-
us-east-1-229768475194/ads508/data/ON_TIME_REPORTING_12.csv
```

[17]:
```
!aws s3 ls $s3_private_path_csv/
```

```
2022-03-21 17:49:04   45671905 ON_TIME_REPORTING_12.csv
```

8 Create Table in Database

[18]:
```
# SQL statement to execute
statement = """CREATE EXTERNAL TABLE IF NOT EXISTS {}.{}(
         DAY_OF_MONTH int,
         DAY_OF_WEEK int,
         OP_UNIQUE_CARRIER string,
         TAIL_NUM string,
         ORIGIN string,
         DEST string,
         DEP_DEL15 float,
         DEP_TIME_BLK string,
         ARR_TIME_BLK string,
         CANCELLED float,
```

```
         CRS_ELAPSED_TIME float,
         DISTANCE float,
         DISTANCE_GROUP int,
         CARRIER_DELAY float,
         WEATHER_DELAY float,
         NAS_DELAY float,
         SECURITY_DELAY float,
         LATE_AIRCRAFT_DELAY float



)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
LOCATION '{}'
TBLPROPERTIES ('skip.header.line.count'='1')""".format(
    database_name, table_name, s3_private_path_csv
)

print(statement)
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS ads508.flight_departure_delays(
         DAY_OF_MONTH int,
         DAY_OF_WEEK int,
         OP_UNIQUE_CARRIER string,
         TAIL_NUM string,
         ORIGIN string,
         DEST string,
         DEP_DEL15 float,
         DEP_TIME_BLK string,
         ARR_TIME_BLK string,
         CANCELLED float,
         CRS_ELAPSED_TIME float,
         DISTANCE float,
         DISTANCE_GROUP int,
         CARRIER_DELAY float,
         WEATHER_DELAY float,
         NAS_DELAY float,
         SECURITY_DELAY float,
         LATE_AIRCRAFT_DELAY float



)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
```

```
LINES TERMINATED BY '
'
LOCATION 's3://sagemaker-us-east-1-229768475194/ads508/data'
TBLPROPERTIES ('skip.header.line.count'='1')
```

[19]:
```
# Droping table if needed
#statement2 = "DROP TABLE {}.{}".format(database_name, table_name)
#pd.read_sql(statement2, conn)
```

[20]:
```
pd.read_sql(statement, conn)
```

[20]:
```
Empty DataFrame
Columns: []
Index: []
```

9 Verify the table has been created successfully

[21]:
```
statement = "SHOW TABLES in {}".format(database_name)

df_show = pd.read_sql(statement, conn)
df_show.head(5)
```

[21]:
```
                  tab_name
0  flight_departure_delays
```

10 Run a sample query

[22]:
```
statement = """SELECT * FROM {}.{}
                LIMIT 10""".format(
    database_name, table_name
)

print(statement)
```

```
SELECT * FROM ads508.flight_departure_delays
                LIMIT 10
```

[23]:
```
pd.read_sql(statement, conn)
```

[23]:

	day_of_month	day_of_week	op_unique_carrier	tail_num	origin	dest	\
0	4	3	WN	N738CB	MDW	OMA	
1	4	3	WN	N7869A	MDW	ONT	
2	4	3	WN	N938WN	MDW	ORF	
3	4	3	WN	N902WN	MDW	ORF	
4	4	3	WN	N8584Z	MDW	PDX	

```
5             4            3                WN   N744SW    MDW  PHL
6             4            3                WN   N8501V    MDW  PHL
7             4            3                WN   N776WN    MDW  PHL
8             4            3                WN   N423WN    MDW  PHL
9             4            3                WN   N720WN    MDW  PHL

   dep_del15 dep_time_blk arr_time_blk  cancelled  crs_elapsed_time  distance  \
0        1.0    2200-2259    2300-2359        0.0              85.0     423.0
1        1.0    1900-1959    2100-2159        0.0             260.0    1706.0
2        0.0    2100-2159    0001-0559        0.0             115.0     704.0
3        1.0    1300-1359    1600-1659        0.0             125.0     704.0
4        0.0    0800-0859    1100-1159        0.0             280.0    1751.0
5        0.0    0001-0559    0800-0859        0.0             110.0     668.0
6        1.0    2100-2159    0001-0559        0.0             105.0     668.0
7        0.0    1500-1559    1800-1859        0.0             120.0     668.0
8        1.0    1100-1159    1400-1459        0.0             115.0     668.0
9        NaN    0900-0959    1100-1159        1.0             110.0     668.0

   distance_group  carrier_delay  weather_delay  nas_delay  security_delay  \
0               2            1.0            0.0        0.0             0.0
1               7           17.0            0.0        0.0             0.0
2               3            NaN            NaN        NaN             NaN
3               3            NaN            NaN        NaN             NaN
4               8            NaN            NaN        NaN             NaN
5               3            NaN            NaN        NaN             NaN
6               3           19.0            0.0        0.0             0.0
7               3            NaN            NaN        NaN             NaN
8               3           18.0            0.0        0.0             0.0
9               3            NaN            NaN        NaN             NaN

   late_aircraft_delay
0                 32.0
1                  0.0
2                  NaN
3                  NaN
4                  NaN
5                  NaN
6                  0.0
7                  NaN
8                  0.0
9                  NaN
```

11 Review the New Athena Table in the Glue Catalog

[24]:
```
from IPython.core.display import display, HTML

display(
    HTML(
        '<b>Review <a target="top" href="https://console.aws.amazon.com/glue/
 ↪home?region={}#">AWS Glue Catalog</a></b>'.format(
            region
        )
    )
)
```

```
<IPython.core.display.HTML object>
```

11.1 1. What days of the month are best and worst for departure delays?

[25]:
```
statement = """WITH counts AS (SELECT day_of_month,
                sum(case DEP_DEL15 when 0 then 1 end) on_time,
                sum(case DEP_DEL15 when 1 then 1 end) delayed
                FROM {}.{}
                GROUP BY day_of_month
                ORDER BY day_of_month),

                total_sum as (SELECT *,  (on_time + delayed) as total
                FROM counts)

                SELECT *,
                delayed / CAST(total AS double) as percent_delayed
                FROM total_sum
                ORDER BY percent_delayed
                """.format(
    database_name, table_name
)

print(statement)
```

```
WITH counts AS (SELECT day_of_month,
                sum(case DEP_DEL15 when 0 then 1 end) on_time,
                sum(case DEP_DEL15 when 1 then 1 end) delayed
                FROM ads508.flight_departure_delays
                GROUP BY day_of_month
                ORDER BY day_of_month),

                total_sum as (SELECT *,  (on_time + delayed) as total
                FROM counts)
```

```
SELECT *,
delayed / CAST(total AS double) as percent_delayed
FROM total_sum
ORDER BY percent_delayed
```

[26]:
```
pd.read_sql(statement, conn)
```

[26]:

	day_of_month	on_time	delayed	total	percent_delayed
0	7	14303	1732	16035	0.108014
1	25	14579	1884	16463	0.114438
2	8	17762	2459	20221	0.121606
3	5	18275	2906	21181	0.137198
4	6	18070	3116	21186	0.147078
5	10	16760	2909	19669	0.147898
6	15	17126	3057	20183	0.151464
7	12	17992	3217	21209	0.151681
8	24	13902	2769	16671	0.166097
9	14	13307	2824	16131	0.175067
10	31	14214	3059	17273	0.177097
11	27	17091	3851	20942	0.183889
12	16	16932	3954	20886	0.189313
13	4	16227	3893	20120	0.193489
14	26	16736	4232	20968	0.201831
15	13	16839	4357	21196	0.205558
16	20	16868	4483	21351	0.209967
17	21	15726	4212	19938	0.211255
18	11	15720	4410	20130	0.219076
19	19	16478	4771	21249	0.224528
20	18	15605	4790	20395	0.234861
21	9	15855	4907	20762	0.236345
22	3	15347	4769	20116	0.237075
23	30	15674	5119	20793	0.246189
24	29	15595	5144	20739	0.248035
25	17	14537	4943	19480	0.253747
26	22	15206	5730	20936	0.273691
27	23	15125	5795	20920	0.277008
28	28	14195	5449	19644	0.277387
29	2	14894	6402	21296	0.300620
30	1	15156	7014	22170	0.316373

Summary: Days of the month show differences, but no obvious pattern - let's explore day of the week.

11.2 2. What day of the week is the best and worst for depature delays?

[27]:
```
statement = """WITH counts AS (SELECT day_of_week,
                 sum(case DEP_DEL15 when 0 then 1 end) on_time,
                 sum(case DEP_DEL15 when 1 then 1 end) delayed
                 FROM {}.{}
                 GROUP BY day_of_week
                 ORDER BY day_of_week),

                 total_sum as (SELECT *,  (on_time + delayed) as total
                 FROM counts)

                 SELECT *,
                 delayed / CAST(total AS double) as percent_delayed
                 FROM total_sum
                 ORDER BY percent_delayed
                 """.format(
    database_name, table_name
)

print(statement)
```

```
WITH counts AS (SELECT day_of_week,
                 sum(case DEP_DEL15 when 0 then 1 end) on_time,
                 sum(case DEP_DEL15 when 1 then 1 end) delayed
                 FROM ads508.flight_departure_delays
                 GROUP BY day_of_week
                 ORDER BY day_of_week),

                 total_sum as (SELECT *,  (on_time + delayed) as total
                 FROM counts)

                 SELECT *,
                 delayed / CAST(total AS double) as percent_delayed
                 FROM total_sum
                 ORDER BY percent_delayed
```

[28]:
```
pd.read_sql(statement, conn)
```

[28]:

	day_of_week	on_time	delayed	total	percent_delayed
0	4	69481	15126	84607	0.178780
1	5	68868	15807	84675	0.186678
2	3	62131	14977	77108	0.194234
3	2	74760	18449	93209	0.197932
4	6	57531	14217	71748	0.198152
5	7	80845	23404	104249	0.224501
6	1	78480	26177	104657	0.250122

Summary: The best days of the week are 4 (Thursday) @ 17.9% and 5 (Friday) @ 18.7%. The worst days of the week are 1 (Monday) @ 25% and 7 (Sunday) @ 22.5%.

11.3 3. What distance groups perform best and worst for departure delays?

[29]:
```
statement = """WITH counts AS (SELECT distance_group,
                sum(case DEP_DEL15 when 0 then 1 end) on_time,
                sum(case DEP_DEL15 when 1 then 1 end) delayed
                FROM {}.{}
                GROUP BY distance_group
                ORDER BY distance_group),

                total_sum as (SELECT *,  (on_time + delayed) as total
                FROM counts)

                SELECT *,
                delayed / CAST(total AS double) as percent_delayed
                FROM total_sum
                ORDER BY percent_delayed
                """.format(
    database_name, table_name
)

print(statement)
```

```
WITH counts AS (SELECT distance_group,
                sum(case DEP_DEL15 when 0 then 1 end) on_time,
                sum(case DEP_DEL15 when 1 then 1 end) delayed
                FROM ads508.flight_departure_delays
                GROUP BY distance_group
                ORDER BY distance_group),

                total_sum as (SELECT *,  (on_time + delayed) as total
                FROM counts)

                SELECT *,
                delayed / CAST(total AS double) as percent_delayed
                FROM total_sum
                ORDER BY percent_delayed
```

[30]:
```
pd.read_sql(statement, conn)
```

[30]:

	distance_group	on_time	delayed	total	percent_delayed
0	1	63452	14152	77604	0.182362
1	3	98656	24926	123582	0.201696
2	4	77200	20107	97307	0.206635

3	2	117550	30683	148233	0.206992
4	6	21412	5642	27054	0.208546
5	10	12621	3554	16175	0.219722
6	5	54398	15357	69755	0.220156
7	7	20416	5856	26272	0.222899
8	9	7205	2086	9291	0.224518
9	11	8931	2640	11571	0.228157
10	8	10255	3154	13409	0.235215

Summary: The best distance groups are 1 (<250 miles) @ 18.2% and 3 (500-749 miles) @ 21.2%. The worst distance groups are 8 (1750-1999 Miles) @ 23.5% and 11 (>2500 Miles) @ 23.5%.

11.4 4. Number of Unique Values

[31]:
```
statement = """SELECT COUNT(DISTINCT op_unique_carrier) as op_unique_carrier,
                  COUNT(DISTINCT tail_num) as tail_num,
                  COUNT(DISTINCT origin) as origin,
                  COUNT(DISTINCT dest) as dest,
                  COUNT(DISTINCT dep_time_blk) as dep_time_blk,
                  COUNT (DISTINCT arr_time_blk) as arr_time_blk
               FROM {}.{}

               """.format(
    database_name, table_name
)

print(statement)
```

```
SELECT COUNT(DISTINCT op_unique_carrier) as op_unique_carrier,
                  COUNT(DISTINCT tail_num) as tail_num,
                  COUNT(DISTINCT origin) as origin,
                  COUNT(DISTINCT dest) as dest,
                  COUNT(DISTINCT dep_time_blk) as dep_time_blk,
                  COUNT (DISTINCT arr_time_blk) as arr_time_blk
               FROM ads508.flight_departure_delays
```

[32]:
```
pd.read_sql(statement, conn)
```

[32]:

	op_unique_carrier	tail_num	origin	dest	dep_time_blk	arr_time_blk
0	17	5479	350	350	19	19

NOTE:
High cardinality in Tail_Num, Origin, Dest make analysis difficult.

11.5 5. What are the best and worst performing airlines for departure delays?

[33]:
```
statement = """WITH counts AS (SELECT op_unique_carrier,
                 sum(case DEP_DEL15 when 0 then 1 end) on_time,
                 sum(case DEP_DEL15 when 1 then 1 end) delayed
                 FROM {}.{}
                 GROUP BY op_unique_carrier
                 ORDER BY op_unique_carrier),

                 total_sum as (SELECT *,  (on_time + delayed) as total
                 FROM counts)

                 SELECT *,
                 delayed / CAST(total AS double) as percent_delayed
                 FROM total_sum
                 ORDER BY percent_delayed
                 """.format(
    database_name, table_name
)

print(statement)
```

```
WITH counts AS (SELECT op_unique_carrier,
                sum(case DEP_DEL15 when 0 then 1 end) on_time,
                sum(case DEP_DEL15 when 1 then 1 end) delayed
                FROM ads508.flight_departure_delays
                GROUP BY op_unique_carrier
                ORDER BY op_unique_carrier),

                total_sum as (SELECT *,  (on_time + delayed) as total
                FROM counts)

                SELECT *,
                delayed / CAST(total AS double) as percent_delayed
                FROM total_sum
                ORDER BY percent_delayed
```

[34]:
```
pd.read_sql(statement, conn)
```

[34]:

	op_unique_carrier	on_time	delayed	total	percent_delayed
0	HA	6618	651	7269	0.089558
1	DL	68764	12736	81500	0.156270
2	9E	19174	3960	23134	0.171177
3	AA	65242	14001	79243	0.176684
4	MQ	21869	4877	26746	0.182345
5	NK	14064	3146	17210	0.182801
6	YX	23081	5263	28344	0.185683

7	UA	41276	9889	51165	0.193277
8	OO	56526	14016	70542	0.198690
9	OH	18804	5218	24022	0.217218
10	EV	8569	2409	10978	0.219439
11	AS	16700	5073	21773	0.232995
12	YV	14221	4338	18559	0.233741
13	G4	7124	2191	9315	0.235212
14	F9	9135	3007	12142	0.247653
15	WN	83724	29540	113264	0.260807
16	B6	17205	7842	25047	0.313091

Summary: We note a wide range of percent of departures delayed by carrier. This could indicate that carrier-specific data (such as staffing) could be good indicators for predicting delays.

Mean % Delayed departures = 20.7% Worst performaing carriers = B6 (JetBlue) @ 31.3% and WN (Southwest) @ 26.1% Best performing carriers = HA (Hawaiian Airlines) @ 9% and DL (Delta Airlines) @ 15.6%

11.6 6. What are the best and worst performing time blocks for departure delays?

[35]:
```
statement = """WITH counts AS (SELECT dep_time_blk,
                sum(case DEP_DEL15 when 0 then 1 end) on_time,
                sum(case DEP_DEL15 when 1 then 1 end) delayed
                FROM {}.{}
                GROUP BY dep_time_blk
                ORDER BY dep_time_blk),

                total_sum as (SELECT *,  (on_time + delayed) as total
                FROM counts)

                SELECT *,
                delayed / CAST(total AS double) as percent_delayed
                FROM total_sum
                ORDER BY percent_delayed
                """.format(
    database_name, table_name
)

print(statement)
```

```
WITH counts AS (SELECT dep_time_blk,
                sum(case DEP_DEL15 when 0 then 1 end) on_time,
                sum(case DEP_DEL15 when 1 then 1 end) delayed
                FROM ads508.flight_departure_delays
                GROUP BY dep_time_blk
                ORDER BY dep_time_blk),
```

```
total_sum as (SELECT *,  (on_time + delayed) as total
FROM counts)

SELECT *,
delayed / CAST(total AS double) as percent_delayed
FROM total_sum
ORDER BY percent_delayed
```

[36]:
```
pd.read_sql(statement, conn)
```

[36]:

	dep_time_blk	on_time	delayed	total	percent_delayed
0	0600-0659	42566	3952	46518	0.084956
1	0001-0559	17565	1964	19529	0.100568
2	0700-0759	36653	4376	41029	0.106656
3	0800-0859	34384	5254	39638	0.132550
4	0900-0959	30593	5706	36299	0.157194
5	1000-1059	30423	6876	37299	0.184348
6	1100-1159	30424	7374	37798	0.195090
7	1200-1259	30721	8133	38854	0.209322
8	1300-1359	27005	7768	34773	0.223392
9	2300-2359	3738	1098	4836	0.227047
10	1400-1459	28186	8609	36795	0.233972
11	1500-1559	27086	8999	36085	0.249383
12	1600-1659	27000	8989	35989	0.249771
13	1700-1759	29585	10279	39864	0.257852
14	2200-2259	12515	4737	17252	0.274577
15	1800-1859	25551	9795	35346	0.277118
16	2000-2059	21364	8626	29990	0.287629
17	2100-2159	13859	5864	19723	0.297318
18	1900-1959	22878	9758	32636	0.298995

Answer: Best times = 6-659am @ 8.5% and 1201am - 6am @ 10% Worst times = 7-759pm @ 29.9% and 9-959pm @ 29.7%

12 Store Variables for the Next Notebooks

[37]:
```
%store
```

```
Stored variables and their in-db values:
auto_ml_job_name                                -> 'automl-dm-10-22-14-04'
autopilot_endpoint_arn                          -> 'arn:aws:sagemaker:us-
east-1:229768475194:endpoint
autopilot_endpoint_name                         -> 'automl-dm-
ep-10-22-58-33'
autopilot_model_arn                             -> 'arn:aws:sagemaker:us-
east-1:229768475194:model/au
```

```
autopilot_model_name                                -> 'automl-dm-
model-10-22-58-32'
autopilot_train_s3_uri                              -> 's3://sagemaker-us-
east-1-229768475194/data/amazon
comprehend_train_s3_uri                             -> 's3://sagemaker-us-
east-1-229768475194/data/amazon
ingest_create_athena_db_passed                      -> True
ingest_create_athena_table_parquet_passed           -> True
ingest_create_athena_table_tsv_passed               -> True
s3_private_path_csv                                 -> 's3://sagemaker-us-
east-1-229768475194/ads508/data
s3_private_path_tsv                                 -> 's3://sagemaker-us-
east-1-229768475194/ads508/data
s3_public_path_csv                                  ->
's3://ads-508-airline/transformed'
s3_public_path_tsv                                  -> 's3://amazon-reviews-
pds/tsv'
setup_dependencies_passed                           -> True
setup_iam_roles_passed                              -> True
setup_s3_bucket_passed                              -> False
```

13 Release Resources

[38]:
```
%%html

<p><b>Shutting down your kernel for this notebook to release resources.</b></p>
<button class="sm-command-button" data-commandlinker-command="kernelmenu:
 ↪shutdown" style="display:none;">Shutdown Kernel</button>

<script>
try {
    els = document.getElementsByClassName("sm-command-button");
    els[0].click();
}
catch(err) {
    // NoOp
}
</script>
```

```
<IPython.core.display.HTML object>
```

[39]:
```
%%javascript

try {
    Jupyter.notebook.save_checkpoint();
    Jupyter.notebook.session.delete();
}
```

```
catch(err) {
    // NoOp
}
```

```
<IPython.core.display.Javascript object>
```