Data Engineer coding assessment

Provided Assessment information: -

 # Belong Data Engineer Coding Exercise *Expected artefacts* *· Documentation - approach, architecture, etc.* *· Tests.* *· No notebooks please, prefer a script.* *· Command line to run the application in an AWS environment (or* *locally) with instructions.* *· Submission - github/any open public repository preferred.* *## Source DataPedestrian Counting System – 2009 to Present (counts per* *hour)* *(https://data.melbourne.vic.gov.au/Transport/Pedestrian-Counting-System 2009-to-Present-counts-/b2ak-trbp)* *https://data.melbourne.vic.gov.au/Transport/Pedestrian-Counting-System Sensor-Locations/h57g-5234 (https://data.melbourne.vic.gov.au/Transport* */Pedestrian-Counting-System-Sensor-Locations/h57g-5234)* *## Extract StatsNeed to show contrived of expected outputs.* *- Top 10 (most pedestrians) locations by day* *- Top 10 (most pedestrians) locations by month## Load- Data into S3 in* *an appropriate format for future querying* *Statistical data into an appropriate data store*

Setting up the platform: -

A Linux based operating system (UBUNTU) is used for this assessment along with Apache Spark (Pyspark).

Please follow the following steps to set up the Pyspark environment in your Linux based system [LINK]

Installing Python and libraries.

Installing Python and Python installation package

```
sudo apt install python3
```

```
sudo apt install python3-pip
```

Installing Python Libraries and requirements: -

- Sodapy - `pip install sodapy` - Used to fetch data from data.melbourne.vic.gov.au API
- Pyspark - `pip install pyspark` - Python based spark extension for data processing and querying
- Pandas - `pip install Pandas` - Extensive data processing library- In our case only used to upload data to S3 in “Parquet” format.
- Boto3, S3fs - `pip install BOTO3, pip install s3fs` - AWS libraries to access s3 and s3 file systems.

Understanding the code and approach: -

Step 1: Reading data from “data.melbourne.vic.gov.au” API

```
client = Socrata("data.melbourne.vic.gov.au", None)
df = spark.createDataFrame(client.get("b2ak-trbp", limit=1000000))
location = spark.createDataFrame(client.get("h57g-5234", limit=20000))
```

df contains the data of pedestrian count for each sensor type. With a limit of 1000000 rows.

Location contains the data for the sensor location.

`client.get("b2ak-trbp", limit=1000000)`

returns a json object of the data which is then converted into the spark data frame.

Step 2: Querying the data and finding out certain statistics:

Top 10 (most pedestrians) locations by day And - Top 10 (most pedestrians) locations by day.

To get daily counts:

```
daily = df.withColumn("hourly_counts", df["hourly_counts"].cast(IntegerType()))\
            .groupBy('sensor_id','day')\
            .sum('hourly_counts')\
            .orderBy('sum(hourly_counts)')\
            .groupBy('sensor_id')\
            .avg('sum(hourly_counts)')\
            .orderBy(F.col('avg(sum(hourly_counts))').desc())\
            .limit(10)\
            .withColumn('avg_daily_counts',F.col('avg(sum(hourly_counts))'))\
            .join(location,on='sensor_id',how="left")\
            .select('sensor_name','sensor_description','avg_daily_counts')\
            .orderBy(F.col('avg_daily_counts').desc())\
            .withColumn('avg_daily_counts',F.floor(F.col('avg_daily_counts')))
print(daily.show())
```

Understanding the code:

1. Converting 'hourly_counts' from string to integer type :
   `df.withColumn("hourly_counts", df["hourly_counts"].cast(IntegerType()))`
2. Grouping the data by 'sensor_id' and 'day' by taking the sum of hourly_counts, This will give us the sum of counts in a day for each sensor.
   `groupBy('sensor_id','day') .sum('hourly_counts')`
3. Again, grouping by sensor ID and taking the average of the count for each sensor for all days this will give us the sensors which are being highly used on average. Ordering by the counts and taking the top 10 rows. To get the top 10 highly used sensors.
   ```
   .groupBy('sensor_id')\
    .avg('sum(hourly_counts)')\
   .orderBy(F.col('avg(sum(hourly_counts))').desc())\
   .limit(10)
   ```
4. Joining the top 10 sensor ids with sensor location dataframe, selecting the required columns.
   ```
   .withColumn('avg_daily_counts',F.col('avg(sum(hourly_counts))'))\
   .join(location,on='sensor_id',how="left")\
   .select('sensor_name','sensor_description','avg_daily_counts')\
   .orderBy(F.col('avg_daily_counts').desc())\
   .withColumn('avg_daily_counts',F.floor(F.col('avg_daily_counts')))
   ```

Implementing the similar logic to get the top monthly sensor locations.

```
monthly = df.withColumn("hourly_counts", df["hourly_counts"].cast(IntegerType()))\
            .groupBy('sensor_id','month')\
            .sum('hourly_counts')\
            .orderBy('sum(hourly_counts)')\
            .groupBy('sensor_id')\
            .avg('sum(hourly_counts)')\
            .orderBy(F.col('avg(sum(hourly_counts))').desc())\
            .limit(10)\
            .withColumn('avg_monthly_counts',F.col('avg(sum(hourly_counts))'))\
            .join(location,on='sensor_id',how="left")\
            .select('sensor_name','sensor_description','avg_monthly_counts')\
            .orderBy(F.col('avg_monthly_counts').desc())\
            .withColumn('avg_monthly_counts',F.floor(F.col('avg_monthly_counts')))
print(monthly.show())
```

The output looks like below:

‘

```
WARNING:root:Requests made without an app_token will be subject to strict throttling limits.
21/10/02 21:00:09 WARN TaskSetManager: Stage 31 contains a task of very large size (10293 KiB). The maximum recommen
ded task size is 1000 KiB.

+-----------+-------------------+----------------+
|sensor_name|  sensor_description|avg_daily_counts|
+-----------+-------------------+----------------+
|    Swa123_T|      Town Hall (West)|          899693|
|      FliS_T|Flinders Street S...|          661773|
|    Bou283_T|Bourke Street Mal...|          639918|
|    Swa295_T|    Melbourne Central|          625434|
|    Bou292_T|Bourke Street Mal...|          620141|
|     PriNW_T|       Princes Bridge|          588886|
|        QV_T|        State Library|          450565|
|    Wil277_T|    Flagstaff Station|          384452|
|    Col270_T|Australia on Collins|          355437|
|     Col15_T|Collins Place (So...|          288966|
+-----------+-------------------+----------------+

None
21/10/02 21:00:17 WARN TaskSetManager: Stage 37 contains a task of very large size (10293 KiB). The maximum recommen
ded task size is 1000 KiB.

+-----------+-------------------+------------------+
|sensor_name|  sensor_description|avg_monthly_counts|
+-----------+-------------------+------------------+
|    Swa123_T|      Town Hall (West)|            787231|
|      FliS_T|Flinders Street S...|            661773|
|    Eli274_T|Flinders St-Eliza...|            642010|
|    Bou283_T|Bourke Street Mal...|            639918|
|    Swa295_T|    Melbourne Central|            625434|
|    Bou292_T|Bourke Street Mal...|            620141|
|       Swa31|Flinders La-Swans...|            564078|
|    Wil277_T|    Flagstaff Station|            538233|
|      MCEC_T|Melbourne Convent...|            517118|
|     PriNW_T|       Princes Bridge|            515275|
+-----------+-------------------+------------------+

None
21/10/02 21:00:24 WARN TaskSetManager: Stage 43 contains a task of very large size (10293 KiB). The maximum recommen
ded task size is 1000 KiB.
```

’

Step 3: Joining the datasets and uploading to S3

Joining the datasets and uploading to S3 using parquet format which is partitioned by date.

Joining the datasets and fetcing the date out of date_time column using regular experssions

Along with preparing AWS credentials: -

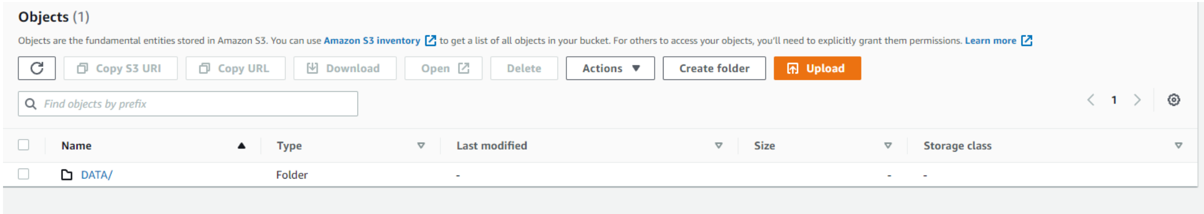
```
final_df = df.join(location, on='sensor_id')\
            .withColumn('date',F.regexp_extract(F.col('date_time'),'\d\d\d\d-\d\d-\d\d',0))

accessKeyId='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
secretAccessKey='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
aws_credentials={'key': accessKeyId, 'secret': secretAccessKey}
```

Uploading the data to S3 in parquet format:

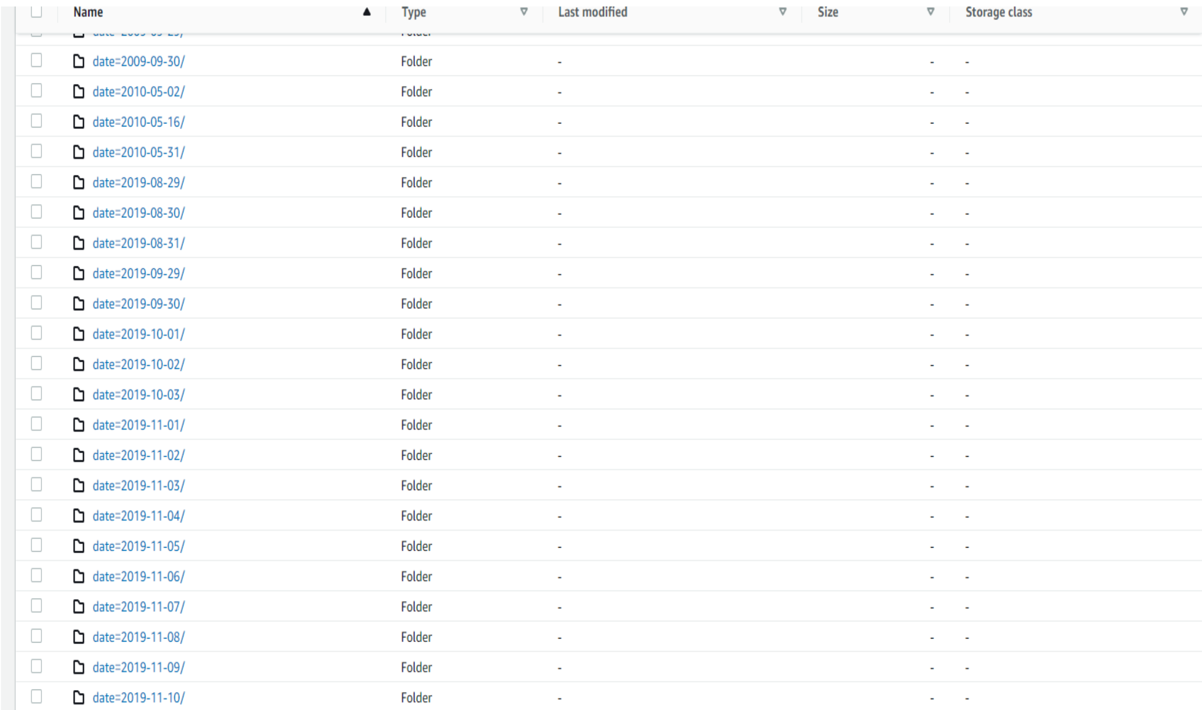
Please note: In ideal situations we will be using pyspark to upload data directly to S3 bucket, But here we are using pandas to upload data to s3 bucket.

```
final_df=final_df.toPandas()
final_df=final_df.loc[:,~final_df.columns.duplicated()]\
            .to_parquet('s3://victoria-pedestrian-counter/DATA/',storage_options= aws_credentials,partition_cols='date')
sc.stop()
```

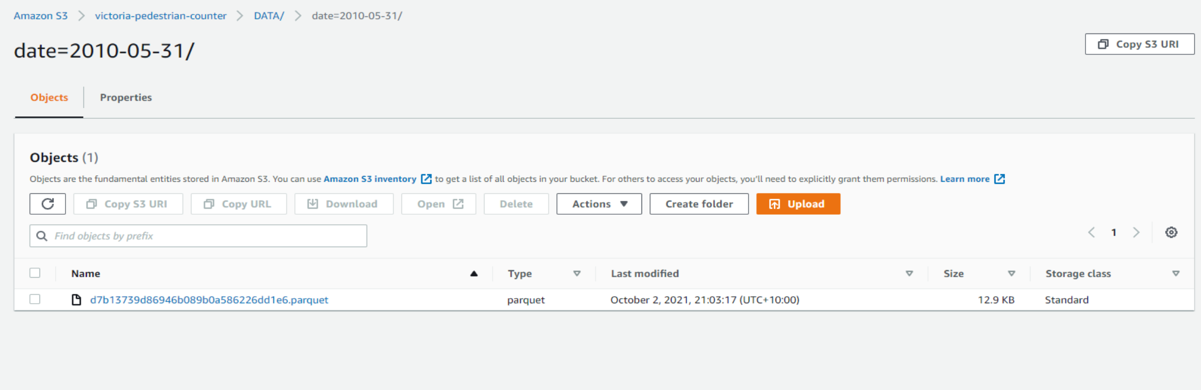
The data is uploaded in the parquet format in DATA/ directory:



The data is partitioned by date column. Which makes it easier to query through the data for future calculations.



Here is a example of how the data looks like in a sample date folder.


Amazon S3 > victoria-pedestrian-counter > DATA/ > date=2010-05-31/
date=2010-05-31/
Copy S3 URI
Objects
Properties
Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
1
Name
Type
Last modified
Size
Storage class
d7b13739d86946b089b0a586226dd1e6.parquet
parquet
October 2, 2021, 21:03:17 (UTC+10:00)
12.9 KB
Standard