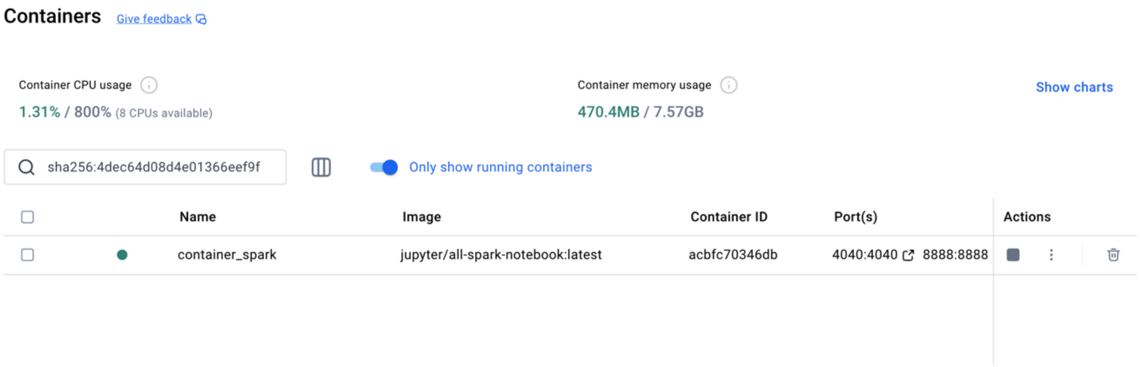
Spark Practice

CONTENTS

PREREQUISITES

- Install Spark locally using one of the methods described here or in Docker.

I used Docker container to launch the Spark.



Container CPU usage
1.31% / 800% (8 CPUs available)

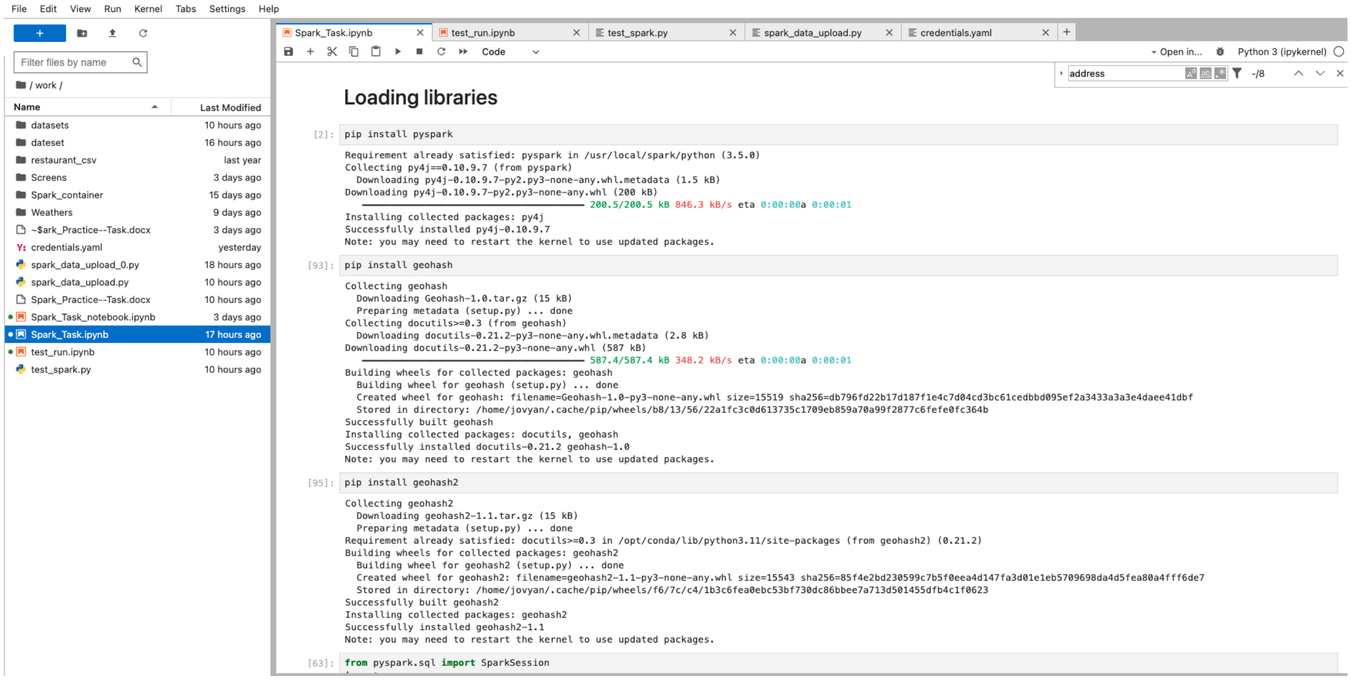
Container memory usage
470.4MB / 7.57GB

sha256:4dec64d08d4e01366eef9f

□		Name	Image	Container ID	Port(s)	Actions
□	●	container_spark	jupyter/all-spark-notebook:latest	acbfc70346db	4040:4040 8888:8888	

- Create a Spark ETL job to read data from a local storage. You can find the data in the Spark Practice—Dataset file on the page with the task description.

Downloaded the datasets to local disk and imported them by indicating the path in jupyter notebook.



/ work /

Name	Last Modified
datasets	10 hours ago
dateset	16 hours ago
restaurant_csv	last year
Screens	3 days ago
Spark_container	15 days ago
Weathers	9 days ago
~$ark_Practice--Task.docx	3 days ago
credentials.yaml	yesterday
spark_data_upload_0.py	18 hours ago
spark_data_upload.py	10 hours ago
Spark_Practice--Task.docx	10 hours ago
Spark_Task_notebook.ipynb	3 days ago
Spark_Task.ipynb	17 hours ago
test_run.ipynb	10 hours ago
test_spark.py	10 hours ago

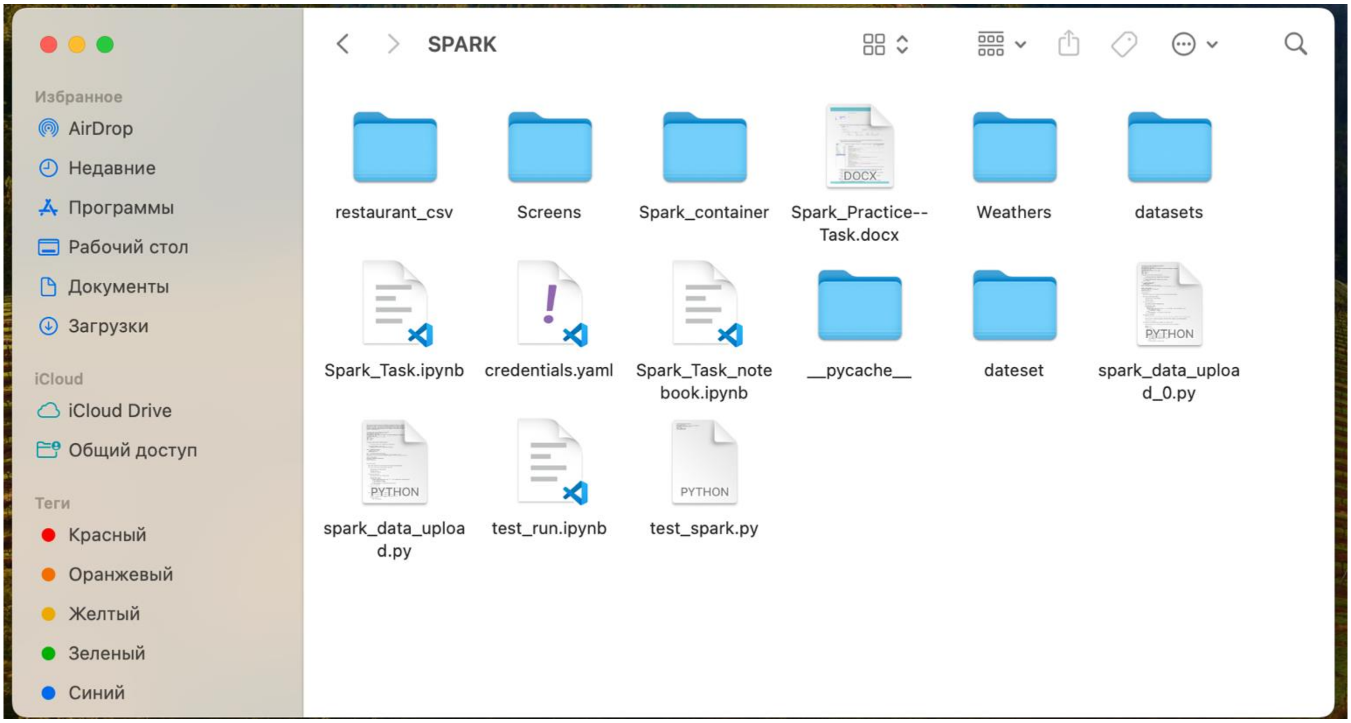
Loading libraries

```
[2]: pip install pyspark

Requirement already satisfied: pyspark in /usr/local/spark/python (3.5.0)
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl.metadata (1.5 kB)
Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.5/200.5 kB 846.3 kB/s eta 0:00:00a 0:00:01
Installing collected packages: py4j
Successfully installed py4j-0.10.9.7
Note: you may need to restart the kernel to use updated packages.
```

```
[93]: pip install geohash

Collecting geohash
  Downloading Geohash-1.0.tar.gz (15 kB)
  Preparing metadata (setup.py) ... done
Collecting docutils>=0.3 (from geohash)
  Downloading docutils-0.21.2-py3-none-any.whl.metadata (2.8 kB)
Downloading docutils-0.21.2-py3-none-any.whl (587 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 587.4/587.4 kB 348.2 kB/s eta 0:00:00a 0:00:01
Building wheels for collected packages: geohash
  Building wheel for geohash (setup.py) ... done
  Created wheel for geohash: filename=Geohash-1.0-py3-none-any.whl size=15519 sha256=db796fd22b17d187f1e4c7d04cd3bc61cedbbd095ef2a3433a3a3e4daee41dbf
  Stored in directory: /home/jovyan/.cache/pip/wheels/b8/13/56/22a1fc3c0d613735c1709eb859a70a99f2877c6fefe0fc364b
Successfully built geohash
Installing collected packages: docutils, geohash
Successfully installed docutils-0.21.2 geohash-1.0
Note: you may need to restart the kernel to use updated packages.
```

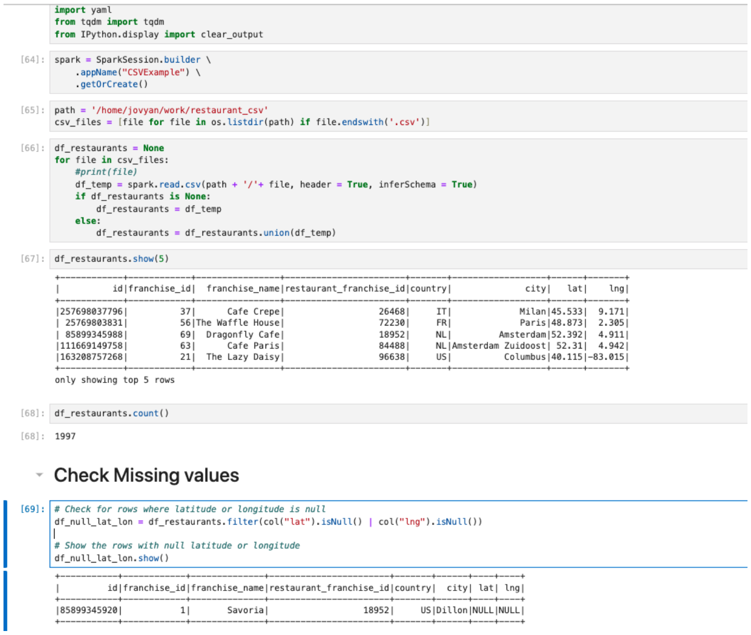
```
[95]: pip install geohash2

Collecting geohash2
  Downloading geohash2-1.1.tar.gz (15 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: docutils>=0.3 in /opt/conda/lib/python3.11/site-packages (from geohash2) (0.21.2)
Building wheels for collected packages: geohash2
  Building wheel for geohash2 (setup.py) ... done
  Created wheel for geohash2: filename=geohash2-1.1-py3-none-any.whl size=15543 sha256=85f4e2bd230599c7b5f0eea4d147fa3d01e1eb5709698da4d5fea80a4fff6de7
  Stored in directory: /home/jovyan/.cache/pip/wheels/f6/7c/c4/1b3c6fea0ebc53bf730dc86bbee7a713d501455dfb4c1f0623
Successfully built geohash2
Installing collected packages: geohash2
Successfully installed geohash2-1.1
Note: you may need to restart the kernel to use updated packages.
```

```
[63]: from pyspark.sql import SparkSession
```



TASK

- Check restaurant data for incorrect (null) values (latitude and longitude). For incorrect values, map latitude and longitude from the OpenCage Geocoding API in a job via the REST API.

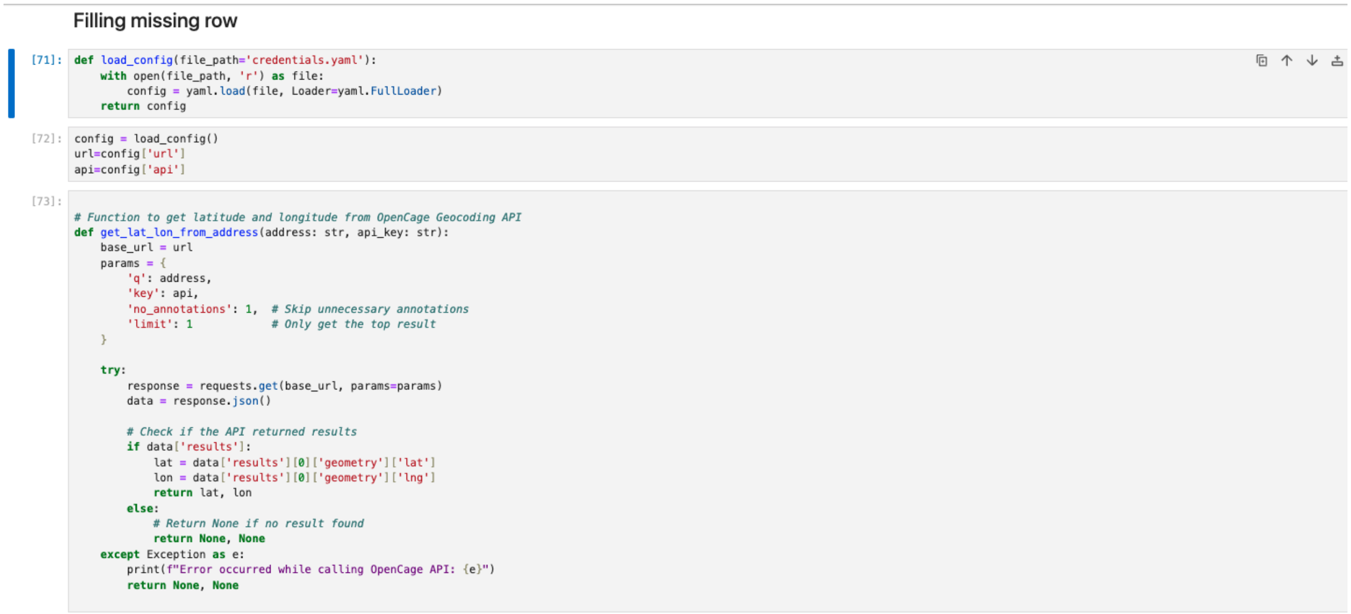


```
import yaml
from tqdm import tqdm
from IPython.display import clear_output

[64]: spark = SparkSession.builder \
          .appName("CSVExample") \
          .getOrCreate()

[65]: path = '/home/jovyan/work/restaurant_csv'
      csv_files = [file for file in os.listdir(path) if file.endswith('.csv')]

[66]: df_restaurants = None
      for file in csv_files:
          #print(file)
          df_temp = spark.read.csv(path + '/'+ file, header = True, inferSchema = True)
          if df_restaurants is None:
              df_restaurants = df_temp
          else:
              df_restaurants = df_restaurants.union(df_temp)

[67]: df_restaurants.show(5)

      +------------+------------+----------------+-----------------------+-------+------------------+------+-------+
      |          id|franchise_id|  franchise_name|restaurant_franchise_id|country|              city|   lat|    lng|
      +------------+------------+----------------+-----------------------+-------+------------------+------+-------+
      |257698037796|          37|      Cafe Crepe|                  26468|     IT|             Milan|45.533|  9.171|
      | 25769803831|          56|The Waffle House|                  72230|     FR|             Paris|48.873|  2.305|
      | 85899345988|          69|  Dragonfly Cafe|                  18952|     NL|         Amsterdam|52.392|  4.911|
      |111669149758|          63|      Cafe Paris|                  84488|     NL|Amsterdam Zuidoost| 52.31|  4.942|
      |163208757268|          21|  The Lazy Daisy|                  96638|     US|          Columbus|40.115|-83.015|
      +------------+------------+----------------+-----------------------+-------+------------------+------+-------+
      only showing top 5 rows

[68]: df_restaurants.count()

[68]: 1997
```

Check Missing values

```
[69]: # Check for rows where latitude or longitude is null
      df_null_lat_lon = df_restaurants.filter(col("lat").isNull() | col("lng").isNull())

      # Show the rows with null latitude or longitude
      df_null_lat_lon.show()

      +-----------+------------+--------------+-----------------------+-------+------+----+----+
      |         id|franchise_id|franchise_name|restaurant_franchise_id|country|  city| lat| lng|
      +-----------+------------+--------------+-----------------------+-------+------+----+----+
      |85899345920|           1|       Savoria|                  18952|     US|Dillon|NULL|NULL|
      +-----------+------------+--------------+-----------------------+-------+------+----+----+
```

1. The process of reading dataset is done iteratively, due to several separate csv files. They were accumulated in an empty file. Dataset “Restaurant” contains 1997 rows, it was found 1 row with missing features latitude and longitude.
2. Filling the missing rows. API and Link of Service for Request is hiden in Credentials.yaml file.

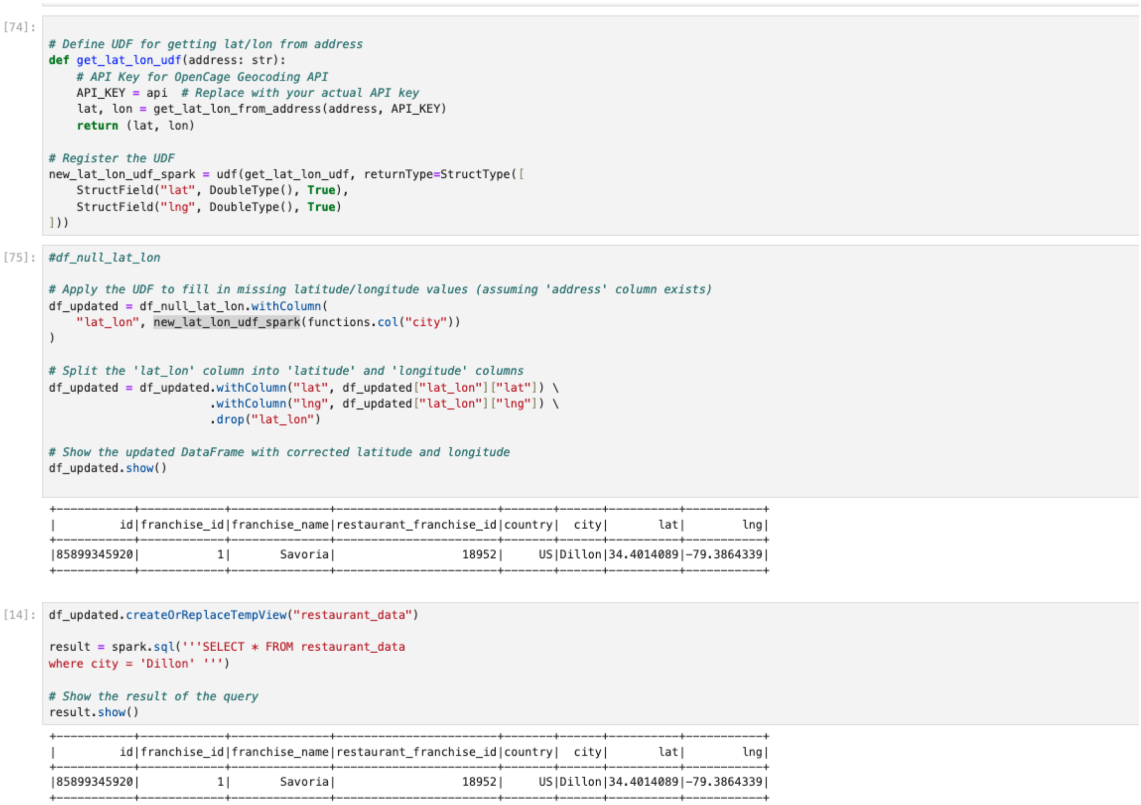


Filling missing row

```
[71]: def load_config(file_path='credentials.yaml'):
          with open(file_path, 'r') as file:
              config = yaml.load(file, Loader=yaml.FullLoader)
          return config
```

```
[72]: config = load_config()
      url=config['url']
      api=config['api']
```

```
[73]:
      # Function to get latitude and longitude from OpenCage Geocoding API
      def get_lat_lon_from_address(address: str, api_key: str):
          base_url = url
          params = {
              'q': address,
              'key': api,
              'no_annotations': 1,  # Skip unnecessary annotations
              'limit': 1            # Only get the top result
          }

          try:
              response = requests.get(base_url, params=params)
              data = response.json()

              # Check if the API returned results
              if data['results']:
                  lat = data['results'][0]['geometry']['lat']
                  lon = data['results'][0]['geometry']['lng']
                  return lat, lon
              else:
                  # Return None if no result found
                  return None, None
          except Exception as e:
              print(f"Error occurred while calling OpenCage API: {e}")
              return None, None
```



```
[74]:
# Define UDF for getting lat/lon from address
def get_lat_lon_udf(address: str):
    # API Key for OpenCage Geocoding API
    API_KEY = api  # Replace with your actual API key
    lat, lon = get_lat_lon_from_address(address, API_KEY)
    return (lat, lon)

# Register the UDF
new_lat_lon_udf_spark = udf(get_lat_lon_udf, returnType=StructType([
    StructField("lat", DoubleType(), True),
    StructField("lng", DoubleType(), True)
]))
```

```
[75]: #df_null_lat_lon

# Apply the UDF to fill in missing latitude/longitude values (assuming 'address' column exists)
df_updated = df_null_lat_lon.withColumn(
    "lat_lon", new_lat_lon_udf_spark(functions.col("city"))
)

# Split the 'lat_lon' column into 'latitude' and 'longitude' columns
df_updated = df_updated.withColumn("lat", df_updated["lat_lon"]["lat"]) \
                       .withColumn("lng", df_updated["lat_lon"]["lng"]) \
                       .drop("lat_lon")

# Show the updated DataFrame with corrected latitude and longitude
df_updated.show()
```

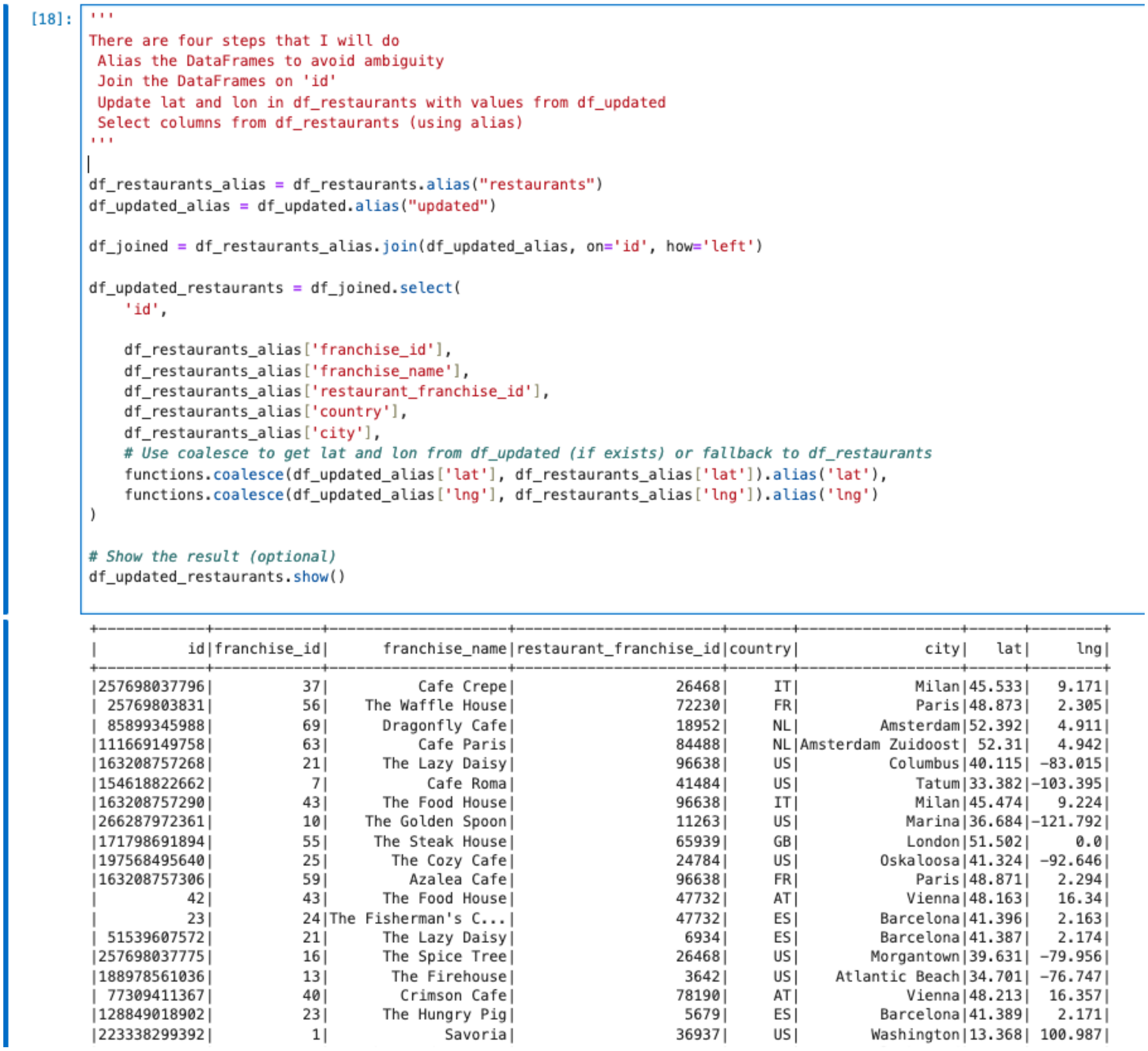
```
+-----------+------------+--------------+-----------------------+-------+------+----------+-----------+
|         id|franchise_id|franchise_name|restaurant_franchise_id|country|  city|       lat|        lng|
+-----------+------------+--------------+-----------------------+-------+------+----------+-----------+
|85899345920|           1|       Savoria|                  18952|     US|Dillon|34.4014089|-79.3864339|
+-----------+------------+--------------+-----------------------+-------+------+----------+-----------+
```

```
[14]: df_updated.createOrReplaceTempView("restaurant_data")

result = spark.sql('''SELECT * FROM restaurant_data
where city = 'Dillon' ''')

# Show the result of the query
result.show()
```

```
+-----------+------------+--------------+-----------------------+-------+------+----------+-----------+
|         id|franchise_id|franchise_name|restaurant_franchise_id|country|  city|       lat|        lng|
+-----------+------------+--------------+-----------------------+-------+------+----------+-----------+
|85899345920|           1|       Savoria|                  18952|     US|Dillon|34.4014089|-79.3864339|
+-----------+------------+--------------+-----------------------+-------+------+----------+-----------+
```

Once we fill the missing rows, we get modified and completed dataset with rows that had missing values.

3. Then, I update the initial dataset with the actual data. I decided to continue my task in this way so that I don't need to transmit all the rows to be modified, and I minimize the risk of losing data or getting incorrect data.



```
[18]: '''
There are four steps that I will do
 Alias the DataFrames to avoid ambiguity
 Join the DataFrames on 'id'
 Update lat and lon in df_restaurants with values from df_updated
 Select columns from df_restaurants (using alias)
'''

df_restaurants_alias = df_restaurants.alias("restaurants")
df_updated_alias = df_updated.alias("updated")

df_joined = df_restaurants_alias.join(df_updated_alias, on='id', how='left')

df_updated_restaurants = df_joined.select(
    'id',

    df_restaurants_alias['franchise_id'],
    df_restaurants_alias['franchise_name'],
    df_restaurants_alias['restaurant_franchise_id'],
    df_restaurants_alias['country'],
    df_restaurants_alias['city'],
    # Use coalesce to get lat and lon from df_updated (if exists) or fallback to df_restaurants
    functions.coalesce(df_updated_alias['lat'], df_restaurants_alias['lat']).alias('lat'),
    functions.coalesce(df_updated_alias['lng'], df_restaurants_alias['lng']).alias('lng')
)

# Show the result (optional)
df_updated_restaurants.show()
```

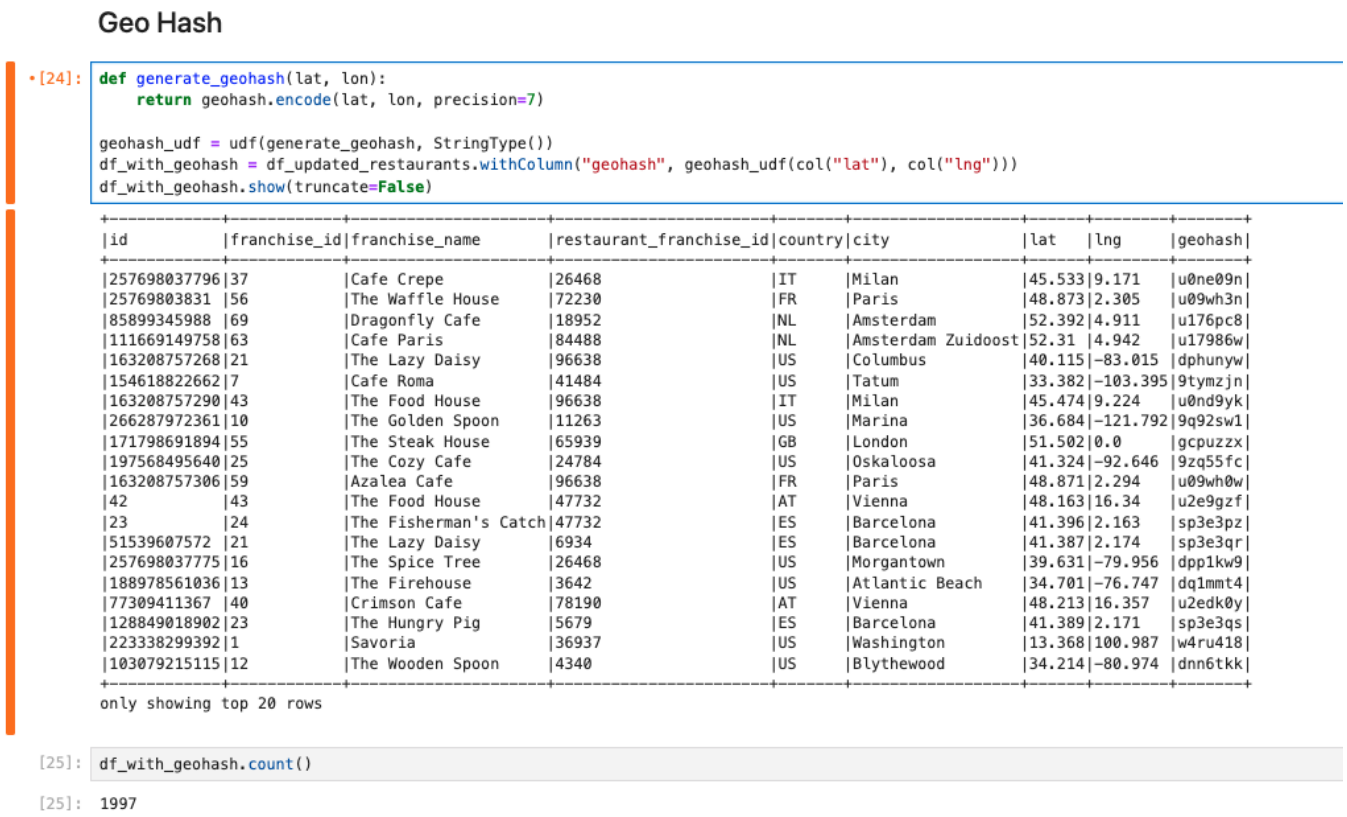
```
+------------+------------+--------------------+-----------------------+-------+------------------+------+--------+
|          id|franchise_id|      franchise_name|restaurant_franchise_id|country|              city|   lat|     lng|
+------------+------------+--------------------+-----------------------+-------+------------------+------+--------+
|257698037796|          37|          Cafe Crepe|                  26468|     IT|             Milan|45.533|   9.171|
| 25769803831|          56|    The Waffle House|                  72230|     FR|             Paris|48.873|   2.305|
| 85899345988|          69|      Dragonfly Cafe|                  18952|     NL|         Amsterdam|52.392|   4.911|
|111669149758|          63|          Cafe Paris|                  84488|     NL|Amsterdam Zuidoost| 52.31|   4.942|
|163208757268|          21|      The Lazy Daisy|                  96638|     US|          Columbus|40.115| -83.015|
|154618822662|           7|           Cafe Roma|                  41484|     US|             Tatum|33.382|-103.395|
|163208757290|          43|      The Food House|                  96638|     IT|             Milan|45.474|   9.224|
|266287972361|          10|    The Golden Spoon|                  11263|     US|            Marina|36.684|-121.792|
|171798691894|          55|     The Steak House|                  65939|     GB|            London|51.502|     0.0|
|197568495640|          25|       The Cozy Cafe|                  24784|     US|         Oskaloosa|41.324| -92.646|
|163208757306|          59|         Azalea Cafe|                  96638|     FR|             Paris|48.871|   2.294|
|          42|          43|      The Food House|                  47732|     AT|            Vienna|48.163|   16.34|
|          23|          24|The Fisherman's C...|                  47732|     ES|         Barcelona|41.396|   2.163|
| 51539607572|          21|      The Lazy Daisy|                   6934|     ES|         Barcelona|41.387|   2.174|
|257698037775|          16|      The Spice Tree|                  26468|     US|        Morgantown|39.631| -79.956|
|188978561036|          13|       The Firehouse|                   3642|     US|    Atlantic Beach|34.701| -76.747|
| 77309411367|          40|        Crimson Cafe|                  78190|     AT|            Vienna|48.213|  16.357|
|128849018902|          23|      The Hungry Pig|                   5679|     ES|         Barcelona|41.389|   2.171|
|223338299392|           1|             Savoria|                  36937|     US|        Washington|13.368| 100.987|
```

○

3.

- Generate a geohash by latitude and longitude using a geohash library like geohash-java. Your geohash should be four characters long and placed in an extra column.

Generating Geohash.



Geo Hash

[24]:
```
def generate_geohash(lat, lon):
    return geohash.encode(lat, lon, precision=7)

geohash_udf = udf(generate_geohash, StringType())
df_with_geohash = df_updated_restaurants.withColumn("geohash", geohash_udf(col("lat"), col("lng")))
df_with_geohash.show(truncate=False)
```

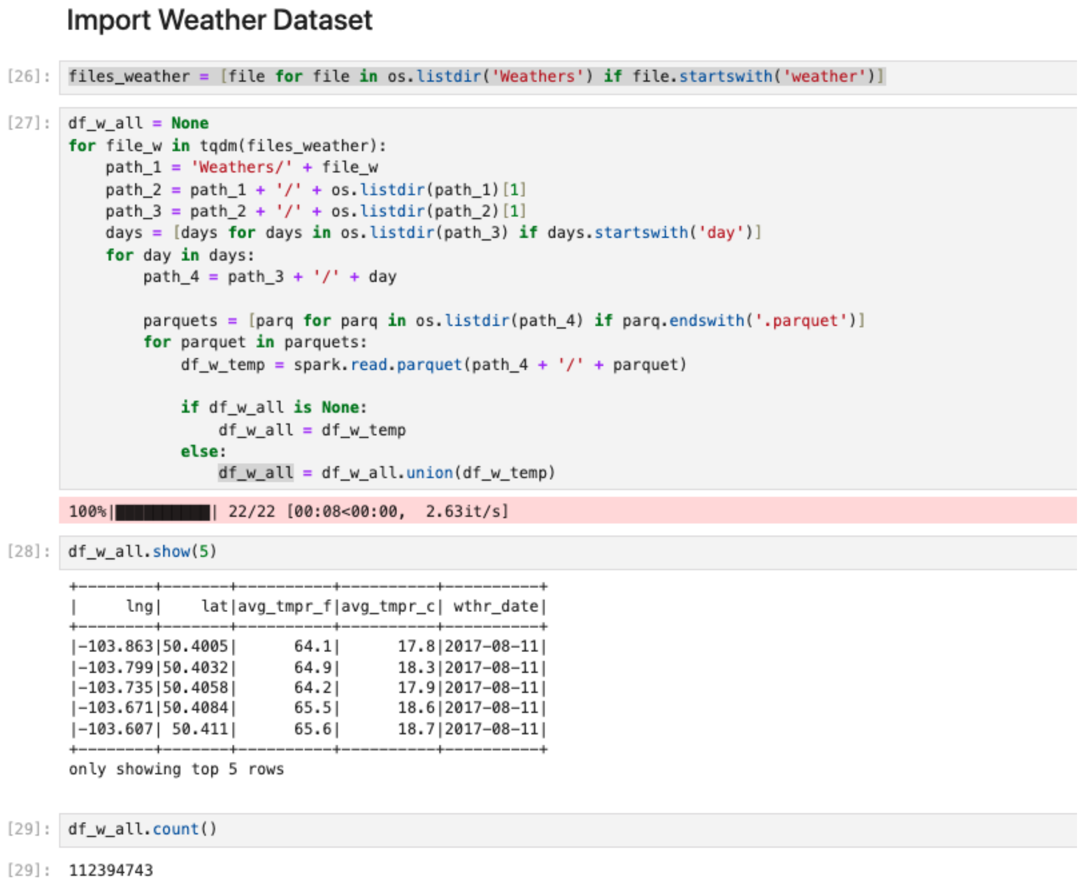
```
+------------+------------+---------------------+-----------------------+-------+------------------+------+--------+-------+
|id          |franchise_id|franchise_name       |restaurant_franchise_id|country|city              |lat   |lng     |geohash|
+------------+------------+---------------------+-----------------------+-------+------------------+------+--------+-------+
|257698037796|37          |Cafe Crepe           |26468                  |IT     |Milan             |45.533|9.171   |u0ne09n|
|25769803831 |56          |The Waffle House     |72230                  |FR     |Paris             |48.873|2.305   |u09wh3n|
|85899345988 |69          |Dragonfly Cafe       |18952                  |NL     |Amsterdam         |52.392|4.911   |u176pc8|
|111669149758|63          |Cafe Paris           |84488                  |NL     |Amsterdam Zuidoost|52.31 |4.942   |u17986w|
|163208757268|21          |The Lazy Daisy       |96638                  |US     |Columbus          |40.115|-83.015 |dphunyw|
|154618822662|7           |Cafe Roma            |41484                  |US     |Tatum             |33.382|-103.395|9tymzjn|
|163208757290|43          |The Food House       |96638                  |IT     |Milan             |45.474|9.224   |u0nd9yk|
|266287972361|10          |The Golden Spoon     |11263                  |US     |Marina            |36.684|-121.792|9q92sw1|
|171798691894|55          |The Steak House      |65939                  |GB     |London            |51.502|0.0     |gcpuzzx|
|197568495640|25          |The Cozy Cafe        |24784                  |US     |Oskaloosa         |41.324|-92.646 |9zq55fc|
|163208757306|59          |Azalea Cafe          |96638                  |FR     |Paris             |48.871|2.294   |u09wh0w|
|42          |43          |The Food House       |47732                  |AT     |Vienna            |48.163|16.34   |u2e9gzf|
|23          |24          |The Fisherman's Catch|47732                  |ES     |Barcelona         |41.396|2.163   |sp3e3pz|
|51539607572 |21          |The Lazy Daisy       |6934                   |ES     |Barcelona         |41.387|2.174   |sp3e3qr|
|257698037775|16          |The Spice Tree       |26468                  |US     |Morgantown        |39.631|-79.956 |dpp1kw9|
|188978561036|13          |The Firehouse        |3642                   |US     |Atlantic Beach    |34.701|-76.747 |dq1mmt4|
|77309411367 |40          |Crimson Cafe         |78190                  |AT     |Vienna            |48.213|16.357  |u2edk0y|
|128849018902|23          |The Hungry Pig       |5679                   |ES     |Barcelona         |41.389|2.171   |sp3e3qs|
|223338299392|1           |Savoria              |36937                  |US     |Washington        |13.368|100.987 |w4ru418|
|103079215115|12          |The Wooden Spoon     |4340                   |US     |Blythewood        |34.214|-80.974 |dnn6tkk|
+------------+------------+---------------------+-----------------------+-------+------------------+------+--------+-------+
only showing top 20 rows
```

[25]:
```
df_with_geohash.count()
```

[25]:
```
1997
```

- Left-join weather and restaurant data using the four-character geohash. Make sure to avoid data multiplication and keep your job idempotent.

Before the left join, I read the weather dataset iteratively by exctracting each data and collected them into 1 spark dataframe



Import Weather Dataset

```
[26]: files_weather = [file for file in os.listdir('Weathers') if file.startswith('weather')]
```

```
[27]: df_w_all = None
      for file_w in tqdm(files_weather):
          path_1 = 'Weathers/' + file_w
          path_2 = path_1 + '/' + os.listdir(path_1)[1]
          path_3 = path_2 + '/' + os.listdir(path_2)[1]
          days = [days for days in os.listdir(path_3) if days.startswith('day')]
          for day in days:
              path_4 = path_3 + '/' + day

              parquets = [parq for parq in os.listdir(path_4) if parq.endswith('.parquet')]
              for parquet in parquets:
                  df_w_temp = spark.read.parquet(path_4 + '/' + parquet)

                  if df_w_all is None:
                      df_w_all = df_w_temp
                  else:
                      df_w_all = df_w_all.union(df_w_temp)
```

```
100%|██████████| 22/22 [00:08<00:00,  2.63it/s]
```

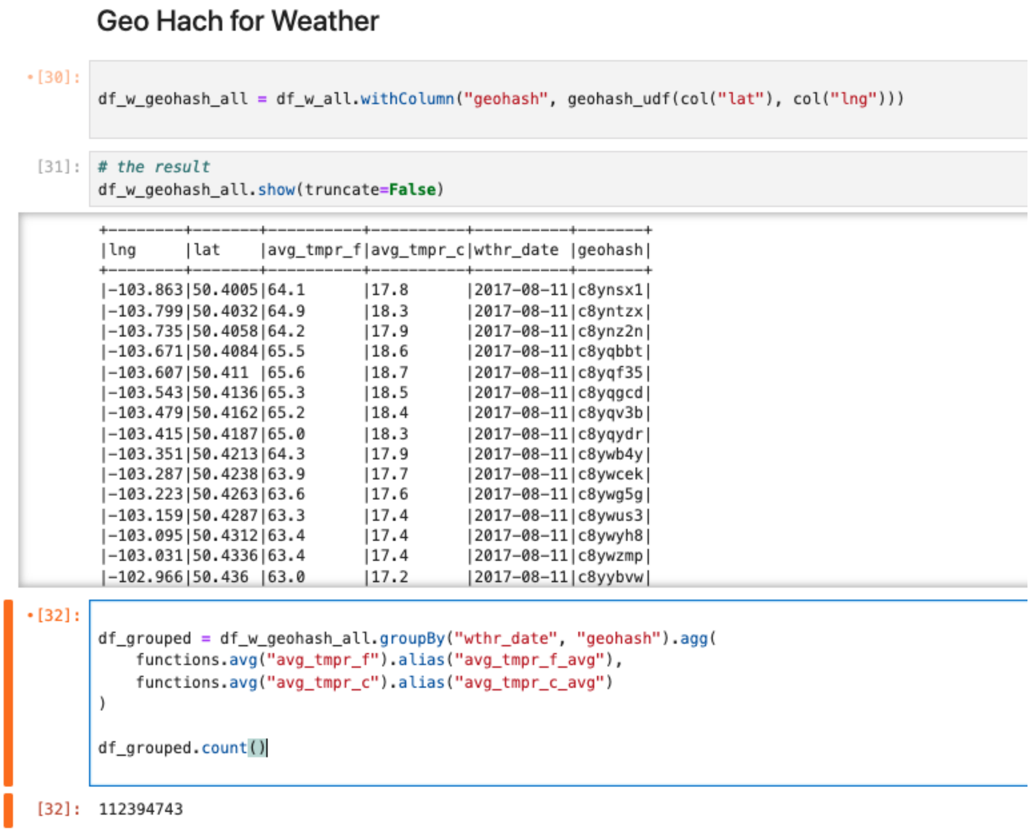
```
[28]: df_w_all.show(5)
```

```
+--------+-------+----------+----------+----------+
|     lng|    lat|avg_tmpr_f|avg_tmpr_c| wthr_date|
+--------+-------+----------+----------+----------+
|-103.863|50.4005|      64.1|      17.8|2017-08-11|
|-103.799|50.4032|      64.9|      18.3|2017-08-11|
|-103.735|50.4058|      64.2|      17.9|2017-08-11|
|-103.671|50.4084|      65.5|      18.6|2017-08-11|
|-103.607| 50.411|      65.6|      18.7|2017-08-11|
+--------+-------+----------+----------+----------+
only showing top 5 rows
```

```
[29]: df_w_all.count()
```

```
[29]: 112394743
```

Since the dataset is missing the Geohash, I generated geohash fro weather dataset.



Geo Hach for Weather

```
[30]: df_w_geohash_all = df_w_all.withColumn("geohash", geohash_udf(col("lat"), col("lng")))
```

```
[31]: # the result
      df_w_geohash_all.show(truncate=False)
```

```
+--------+-------+----------+----------+----------+-------+
|lng     |lat    |avg_tmpr_f|avg_tmpr_c|wthr_date |geohash|
+--------+-------+----------+----------+----------+-------+
|-103.863|50.4005|64.1      |17.8      |2017-08-11|c8ynsx1|
|-103.799|50.4032|64.9      |18.3      |2017-08-11|c8yntzx|
|-103.735|50.4058|64.2      |17.9      |2017-08-11|c8ynz2n|
|-103.671|50.4084|65.5      |18.6      |2017-08-11|c8yqbbt|
|-103.607|50.411 |65.6      |18.7      |2017-08-11|c8yqf35|
|-103.543|50.4136|65.3      |18.5      |2017-08-11|c8yqgcd|
|-103.479|50.4162|65.2      |18.4      |2017-08-11|c8yqv3b|
|-103.415|50.4187|65.0      |18.3      |2017-08-11|c8yqydr|
|-103.351|50.4213|64.3      |17.9      |2017-08-11|c8ywb4y|
|-103.287|50.4238|63.9      |17.7      |2017-08-11|c8ywcek|
|-103.223|50.4263|63.6      |17.6      |2017-08-11|c8ywg5g|
|-103.159|50.4287|63.3      |17.4      |2017-08-11|c8ywus3|
|-103.095|50.4312|63.4      |17.4      |2017-08-11|c8ywyh8|
|-103.031|50.4336|63.4      |17.4      |2017-08-11|c8ywzmp|
|-102.966|50.436 |63.0      |17.2      |2017-08-11|c8yybvw|
```

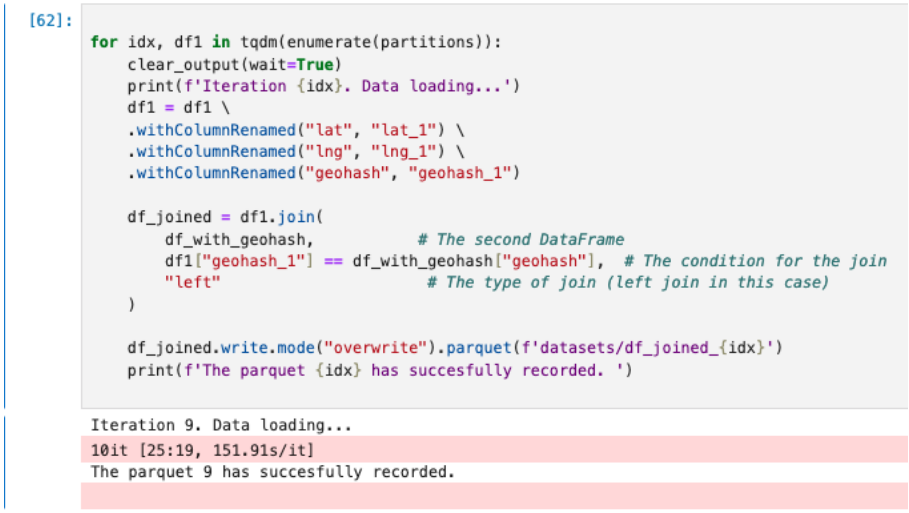
```
[32]: df_grouped = df_w_geohash_all.groupBy("wthr_date", "geohash").agg(
          functions.avg("avg_tmpr_f").alias("avg_tmpr_f_avg"),
          functions.avg("avg_tmpr_c").alias("avg_tmpr_c_avg")
      )

      df_grouped.count()

[32]: 112394743
```

Note: Development and testing should be done locally in your IDE environment. Development and testing is proceeded locally in my machine(Jupyter lab)

○ Store the enriched data (i.e., the joined data with all the fields from both datasets) in the local file system, preserving data partitioning in the parquet format.

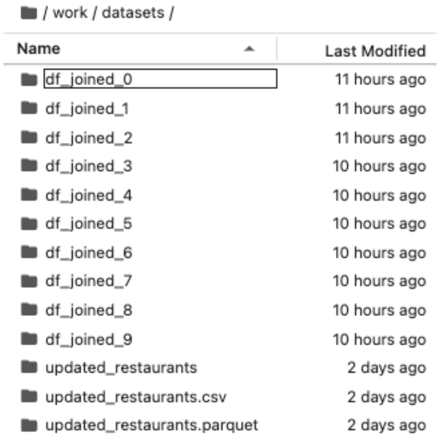
The process of storing data was quiet challenging. Due to a large amount of data, it is better first decide in which way it is optimal in term of memory and time . Also, it depends on business task, if the business ask to provide specific group or segment of data, then better to do preprocessing and save only required data. In this task, we are asked to store all data, therefore, I splitted the whole data into 10 parts and saved separately as parquet file. **Parquet** is the most popular file format for Spark and many big data frameworks. It is a columnar storage format, meaning it stores data in columns rather than rows. Parquet supports efficient compression techniques, which reduce storage costs. It compresses better than row-based formats like CSV or JSON.

Join and saving table



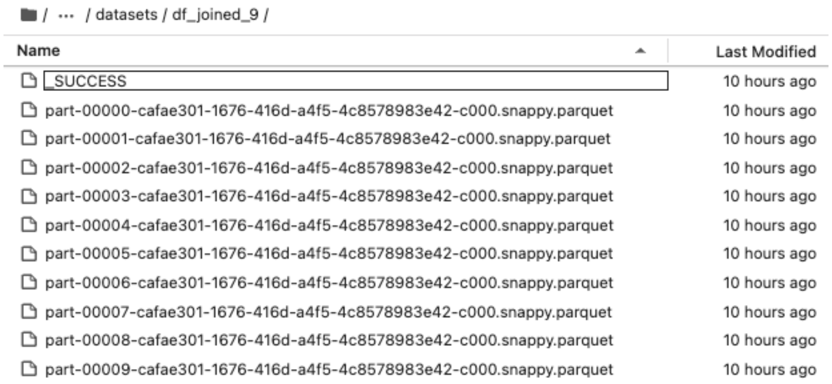
```
[62]:
for idx, df1 in tqdm(enumerate(partitions)):
    clear_output(wait=True)
    print(f'Iteration {idx}. Data loading...')
    df1 = df1 \
    .withColumnRenamed("lat", "lat_1") \
    .withColumnRenamed("lng", "lng_1") \
    .withColumnRenamed("geohash", "geohash_1")

    df_joined = df1.join(
        df_with_geohash,            # The second DataFrame
        df1["geohash_1"] == df_with_geohash["geohash"],  # The condition for the join
        "left"                       # The type of join (left join in this case)
    )

    df_joined.write.mode("overwrite").parquet(f'datasets/df_joined_{idx}')
    print(f'The parquet {idx} has succesfully recorded. ')
```

```
Iteration 9. Data loading...
10it [25:19, 151.91s/it]
The parquet 9 has succesfully recorded.
```



/ work / datasets /

Name	Last Modified
df_joined_0	11 hours ago
df_joined_1	11 hours ago
df_joined_2	11 hours ago
df_joined_3	10 hours ago
df_joined_4	10 hours ago
df_joined_5	10 hours ago
df_joined_6	10 hours ago
df_joined_7	10 hours ago
df_joined_8	10 hours ago
df_joined_9	10 hours ago
updated_restaurants	2 days ago
updated_restaurants.csv	2 days ago
updated_restaurants.parquet	2 days ago



/ … / datasets / df_joined_9 /

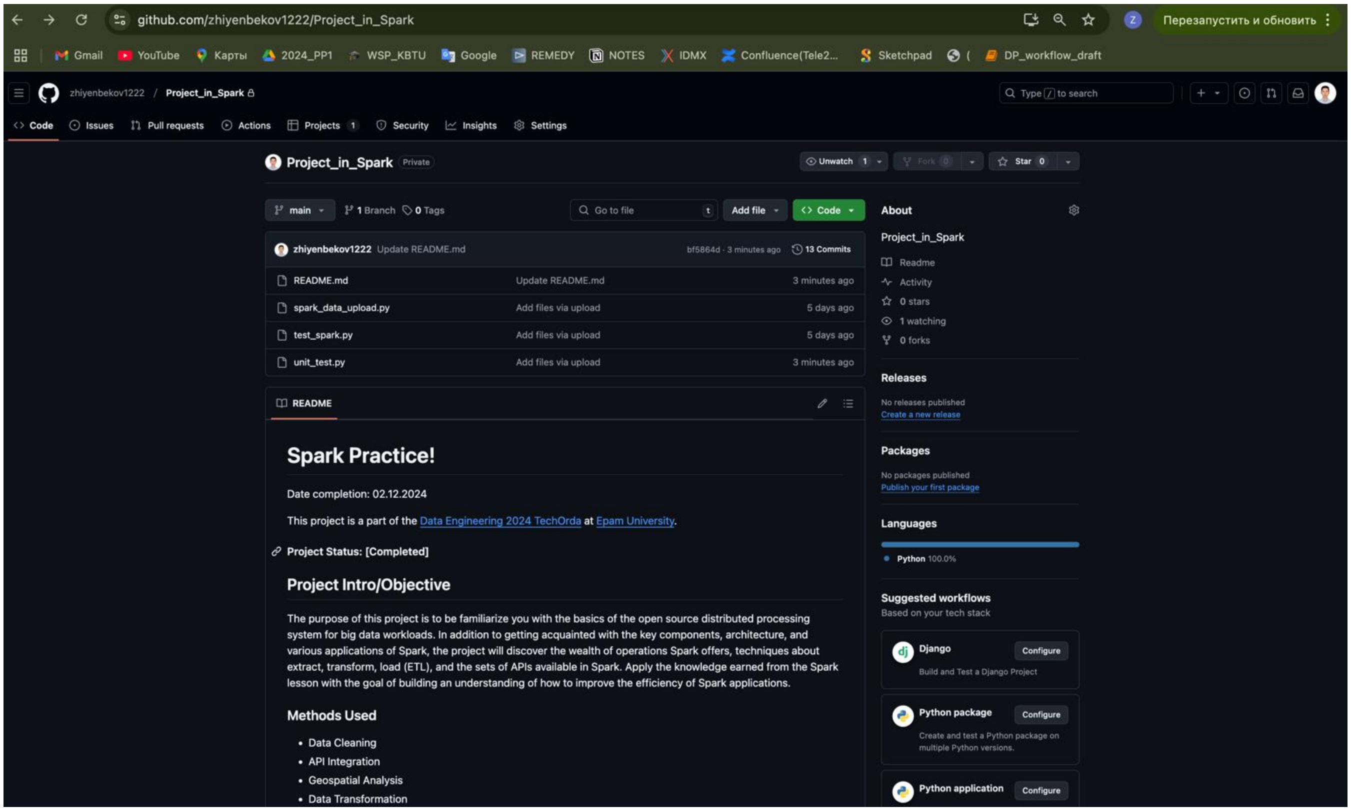
Name	Last Modified
_SUCCESS	10 hours ago
part-00000-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00001-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00002-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00003-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00004-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00005-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00006-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00007-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00008-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago
part-00009-cafae301-1676-416d-a4f5-4c8578983e42-c000.snappy.parquet	10 hours ago

You are expected to:

- Upload the source code and implement tests

  Upload your fully documented homework with screenshots and comments in the task Readme MD file with the repo link. https://github.com/zhiyenbekov1222/Project_in_Spark.git

To completely finish the data, I crated a repository in GitHub and completed Readme MarkDown file. Also, I source code provided in **.py format and ready to run!**



zhiyenbekov1222 Update README.md		bf5864d · 3 minutes ago 13 Commits
README.md	Update README.md	3 minutes ago
spark_data_upload.py	Add files via upload	5 days ago
test_spark.py	Add files via upload	5 days ago
unit_test.py	Add files via upload	3 minutes ago

Spark Practice!

Date completion: 02.12.2024

This project is a part of the Data Engineering 2024 TechOrda at Epam University.

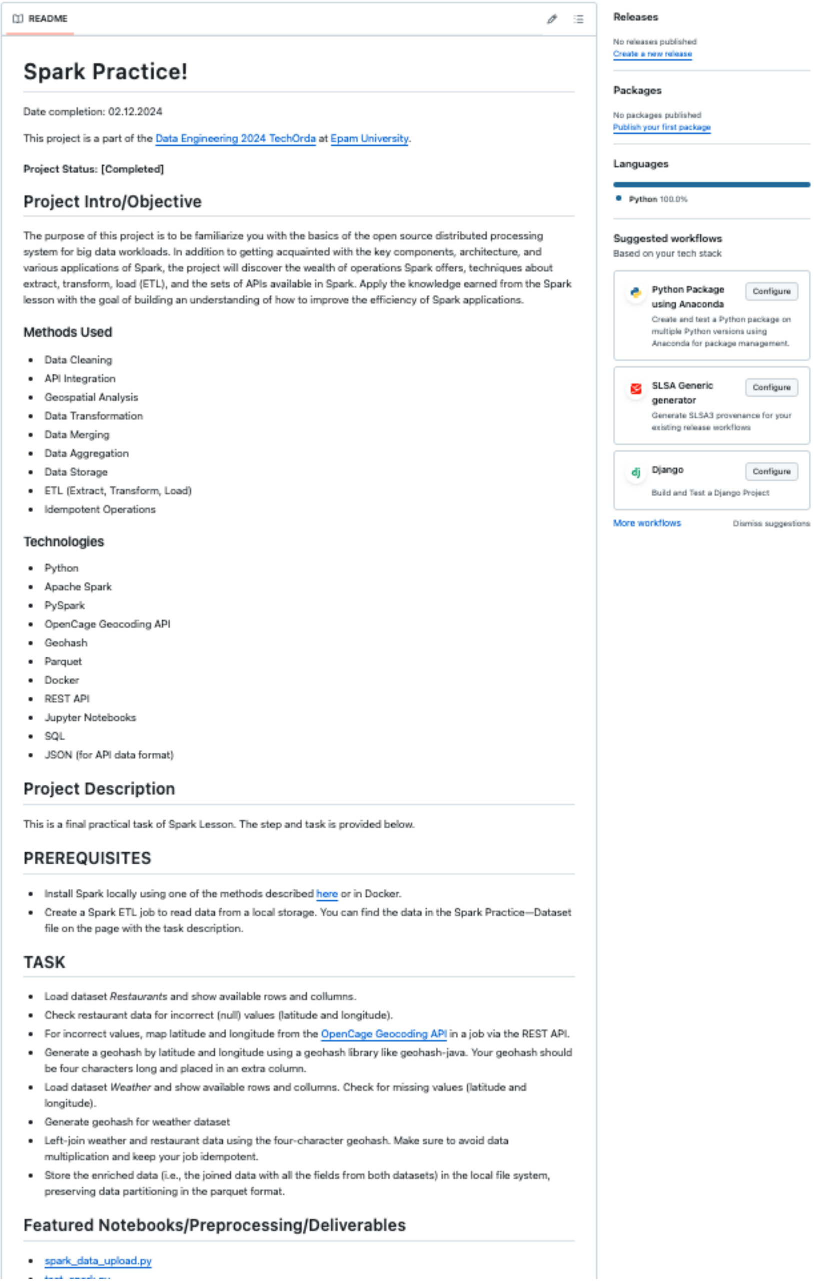
Project Status: [Completed]

Project Intro/Objective

The purpose of this project is to be familiarize you with the basics of the open source distributed processing system for big data workloads. In addition to getting acquainted with the key components, architecture, and various applications of Spark, the project will discover the wealth of operations Spark offers, techniques about extract, transform, load (ETL), and the sets of APIs available in Spark. Apply the knowledge earned from the Spark lesson with the goal of building an understanding of how to improve the efficiency of Spark applications.

Methods Used

- Data Cleaning
- API Integration
- Geospatial Analysis
- Data Transformation



Spark Practice!

Date completion: 02.12.2024

This project is a part of the Data Engineering 2024 TechOrda at Epam University.

Project Status: [Completed]

Project Intro/Objective

The purpose of this project is to be familiarize you with the basics of the open source distributed processing system for big data workloads. In addition to getting acquainted with the key components, architecture, and various applications of Spark, the project will discover the wealth of operations Spark offers, techniques about extract, transform, load (ETL), and the sets of APIs available in Spark. Apply the knowledge earned from the Spark lesson with the goal of building an understanding of how to improve the efficiency of Spark applications.

Methods Used

- Data Cleaning
- API Integration
- Geospatial Analysis
- Data Transformation
- Data Merging
- Data Aggregation
- Data Storage
- ETL (Extract, Transform, Load)
- Idempotent Operations

Technologies

- Python
- Apache Spark
- PySpark
- OpenCage Geocoding API
- Geohash
- Parquet
- Docker
- REST API
- Jupyter Notebooks
- SQL
- JSON (for API data format)

Project Description

This is a final practical task of Spark Lesson. The step and task is provided below.

PREREQUISITES

- Install Spark locally using one of the methods described here or in Docker.
- Create a Spark ETL job to read data from a local storage. You can find the data in the Spark Practice—Dataset file on the page with the task description.

TASK

- Load dataset *Restaurants* and show available rows and collumns.
- Check restaurant data for incorrect (null) values (latitude and longitude).
- For incorrect values, map latitude and longitude from the OpenCage Geocoding API in a job via the REST API.
- Generate a geohash by latitude and longitude using a geohash library like geohash-java. Your geohash should be four characters long and placed in an extra column.
- Load dataset *Weather* and show available rows and collumns. Check for missing values (latitude and longitude).
- Generate geohash for weather dataset
- Left-join weather and restaurant data using the four-character geohash. Make sure to avoid data multiplication and keep your job idempotent.
- Store the enriched data (i.e., the joined data with all the fields from both datasets) in the local file system, preserving data partitioning in the parquet format.

Featured Notebooks/Preprocessing/Deliverables

- spark_data_upload.py
- test_spark.py

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

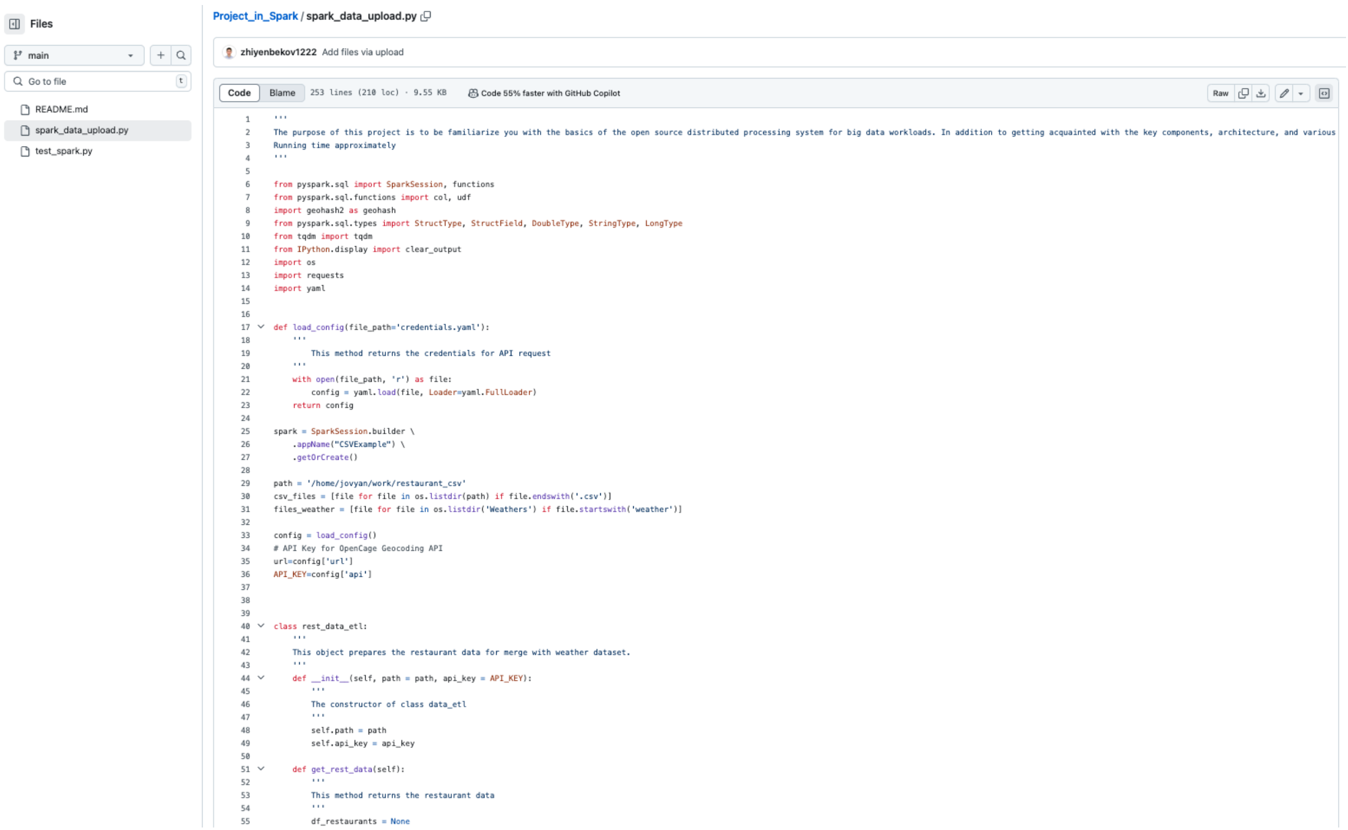
- Python 100.0%

Suggested workflows
Based on your tech stack

Python Package using Anaconda
Create and test a Python package on multiple Python versions using Anaconda for package management.

SLSA Generic generator
Generate SLSA3 provenance for your existing release workflows

Django
Build and Test a Django Project



Project_in_Spark / spark_data_upload.py

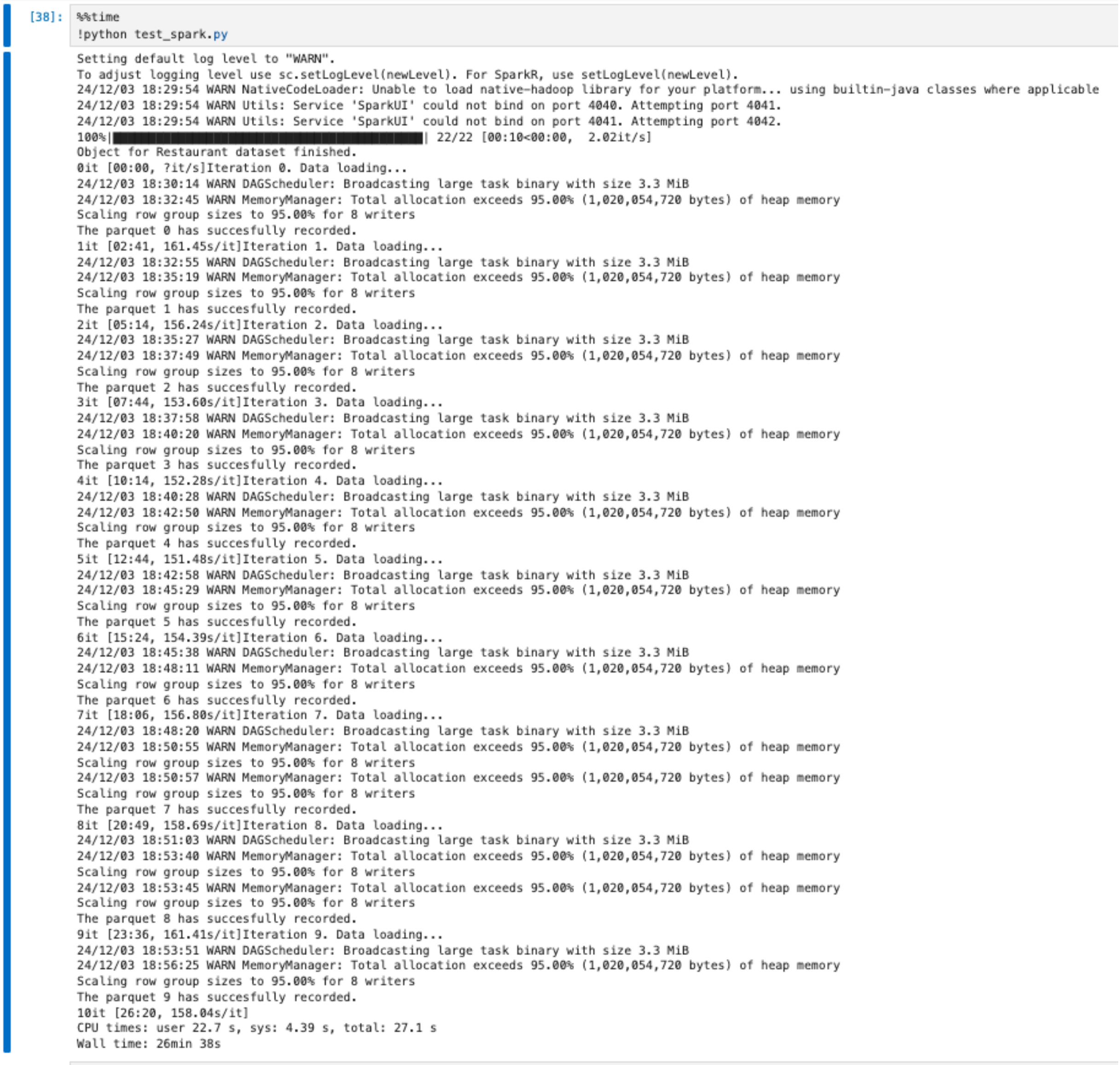
zhiyenbekov1222 Add files via upload

253 lines (210 loc) · 9.55 KB

```
'''
The purpose of this project is to be familiarize you with the basics of the open source distributed processing system for big data workloads. In addition to getting acquainted with the key components, architecture, and various
Running time approximately
'''

from pyspark.sql import SparkSession, functions
from pyspark.sql.functions import col, udf
import geohash2 as geohash
from pyspark.sql.types import StructType, StructField, DoubleType, StringType, LongType
from tqdm import tqdm
from IPython.display import clear_output
import os
import requests
import yaml


def load_config(file_path='credentials.yaml'):
    '''
        This method returns the credentials for API request
    '''
    with open(file_path, 'r') as file:
        config = yaml.load(file, Loader=yaml.FullLoader)
    return config

spark = SparkSession.builder \
    .appName("CSVExample") \
    .getOrCreate()

path = '/home/jovyan/work/restaurant_csv'
csv_files = [file for file in os.listdir(path) if file.endswith('.csv')]
files_weather = [file for file in os.listdir('Weathers') if file.startswith('weather')]

config = load_config()
# API Key for OpenCage Geocoding API
url=config['url']
API_KEY=config['api']



class rest_data_etl:
    '''
    This object prepares the restaurant data for merge with weather dataset.
    '''
    def __init__(self, path = path, api_key = API_KEY):
        '''
        The constructor of class data_etl
        '''
        self.path = path
        self.api_key = api_key

    def get_rest_data(self):
        '''
        This method returns the restaurant data
        '''
        df_restaurants = None
```

The logs in testing the file. Totally, the running time of the process is roughly 27 minutes.

Below the screen…



```
[38]: %%time
      !python test_spark.py

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/03 18:29:54 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/12/03 18:29:54 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
24/12/03 18:29:54 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
100%|████████████████████████████████████████| 22/22 [00:10<00:00,  2.02it/s]
Object for Restaurant dataset finished.
0it [00:00, ?it/s]Iteration 0. Data loading...
24/12/03 18:30:14 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:32:45 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 0 has succesfully recorded.
1it [02:41, 161.45s/it]Iteration 1. Data loading...
24/12/03 18:32:55 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:35:19 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 1 has succesfully recorded.
2it [05:14, 156.24s/it]Iteration 2. Data loading...
24/12/03 18:35:27 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:37:49 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 2 has succesfully recorded.
3it [07:44, 153.60s/it]Iteration 3. Data loading...
24/12/03 18:37:58 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:40:20 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 3 has succesfully recorded.
4it [10:14, 152.28s/it]Iteration 4. Data loading...
24/12/03 18:40:28 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:42:50 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 4 has succesfully recorded.
5it [12:44, 151.48s/it]Iteration 5. Data loading...
24/12/03 18:42:58 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:45:29 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 5 has succesfully recorded.
6it [15:24, 154.39s/it]Iteration 6. Data loading...
24/12/03 18:45:38 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:48:11 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 6 has succesfully recorded.
7it [18:06, 156.80s/it]Iteration 7. Data loading...
24/12/03 18:48:20 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:50:55 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
24/12/03 18:50:57 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 7 has succesfully recorded.
8it [20:49, 158.69s/it]Iteration 8. Data loading...
24/12/03 18:51:03 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:53:40 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
24/12/03 18:53:45 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 8 has succesfully recorded.
9it [23:36, 161.41s/it]Iteration 9. Data loading...
24/12/03 18:53:51 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
24/12/03 18:56:25 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
The parquet 9 has succesfully recorded.
10it [26:20, 158.04s/it]
CPU times: user 22.7 s, sys: 4.39 s, total: 27.1 s
Wall time: 26min 38s
```