API Data Extraction and Loading Challenge

Individual Work

Roberto Reynoso

Url for the data I am grabbing from.

```
url = 'https://api.safegraph.com/v2/graphql'
url_rm = 'https://rickandmortyapi.com/graphql/'
# sfkey
# Url for the data information.
```

This runs the url by their key.

```
%run ./keys
```

This code is making the request to get the API data information.

```
# This runs the urls by their key ^
# Initiate API connection
transport = RequestsHTTPTransport(
    url=url,
    verify=True,
    retries=3,
    headers={'Content-Type': 'application/json', 'apikey': sfkey})

client = Client(transport=transport, fetch_schema_from_transport=True)

# Initiate API connection
transport_rm = RequestsHTTPTransport(
    url=url_rm,
    verify=True,
    retries=3,
    headers={'Content-Type': 'application/json'})

client_rm = Client(transport=transport_rm, fetch_schema_from_transport=True)
# This code is making the request to get the API data information.
```

This sets up the Schema and it names each column and gives there type accordingly, The Boolean value, is for whether it can be null or not.

```
# schema setup
schema_rm = StructType([
  StructField("name",StringType(),True),
  StructField("species",StringType(),True),
  StructField("status",StringType(),True),
  StructField("image",StringType(),True),
  StructField('episode', ArrayType(MapType(StringType(), StringType(), True))),
])

schema = StructType([
  StructField("placekey",StringType(),True),
  StructField("location_name",StringType(),True),
  StructField("parent_placekey",StringType(),True),
  StructField("street_address",StringType(),True),
  StructField("city",StringType(),True),
  StructField("date_range_end",StringType(),True),
  StructField("date_range_start",StringType(),True),
  StructField("poi_cbg",StringType(),True),
  StructField("postal_code", StringType(),True),
  StructField("iso_country_code",StringType(),True),
  StructField("median_dwell",DoubleType(),True),
  StructField("raw_visit_counts",LongType(),True),
  StructField("raw_visitor_counts",LongType(),True),
  StructField("visits_by_day", ArrayType(LongType(), True)),
  StructField("region",StringType(),True),
  StructField("device_type", MapType(StringType(), LongType(), True)),
  StructField("bucketed_dwell_times", MapType(StringType(), LongType(), True)),
  StructField("distance_from_home",LongType(),True),
  StructField("related_same_day_brand", MapType(StringType(), LongType(), True)),
  StructField("related_same_month_brand", MapType(StringType(), LongType(), True)),
  StructField("visitor_country_of_origin", MapType(StringType(), LongType(), True)),
  StructField("visitor_daytime_cbgs", MapType(StringType(), LongType(), True)),
  StructField("visitor_home_aggregation", MapType(StringType(), LongType(), True)),
  StructField("visitor_home_cbgs", MapType(StringType(), LongType(), True)),
  StructField("normalized_visits_by_total_visits",DoubleType(),True),
  StructField("normalized_visits_by_state_scaling",DoubleType(),True),
  StructField("normalized_visits_by_total_visitors",DoubleType(),True),
  StructField("normalized_visits_by_region_naics_visits",DoubleType(),True),
  StructField("normalized_visits_by_region_naics_visitors",DoubleType(),True)
])

schema_core = StructType([
  StructField("placekey",  StringType(),True),
  StructField("location_name",  StringType(),True),
  StructField('brands', ArrayType(MapType(StringType(), StringType(), True))),
  StructField("naics_code", LongType(),True),
  StructField("latitude", DoubleType(),True),
  StructField("longitude", DoubleType(),True),
  StructField("street_address",  StringType(),True),
  StructField("city",  StringType(),True),
```

```
  StructField("postal_code",  StringType(),True),
  StructField("region",  StringType(),True),
  StructField("closed_on",  StringType(),True),
  StructField("opened_on",  StringType(),True),
  StructField("parent_placekey",  StringType(),True)
])

print("Read `schema_rm`, 'schema' and 'schema_core'")

# This sets up the Schema and it names each column and gives there type accordingly, The Boolear
```

Here we make a query for the information from the data set to grab what we need.

```
query_rm = """
query {
    characters {
    results {
      name
      status
      species
      gender
      image
      episode{
        episode
        name
        air_date
      }
    }
  }
}
"""


# weekly_patterns function

query_sg = """query {
  search(filter: {
     --FILTERS--
    address: {
      region: "--STATENAME--"
    }
  }){
    places {
      results(first: 500 after: "--ENDCURSER--") {
        pageInfo { hasNextPage, endCursor}
        edges {
          node {
            monthly_patterns (start_date: "--DATESTART--" end_date: "--DATEEND--") {
              placekey
              parent_placekey
              location_name
              street_address
              city
              region
              postal_code
              iso_country_code
              date_range_start
              date_range_end
              raw_visit_counts
              raw_visitor_counts
              visits_by_day
              device_type
              poi_cbg
```

```
                visitor_home_cbgs
                visitor_home_aggregation
                visitor_daytime_cbgs
                visitor_country_of_origin
                distance_from_home
                median_dwell
                bucketed_dwell_times
                related_same_day_brand
                related_same_month_brand
                normalized_visits_by_total_visits
                normalized_visits_by_state_scaling
                normalized_visits_by_total_visitors
                normalized_visits_by_region_naics_visits
                normalized_visits_by_region_naics_visitors
              }
            }
          }
        }
      }
    }
  }
"""

# Here we make a query for the information from the data set to grab what we need.
```

Here we can change what information we want to grab from the data, by replacing the values for the specific sections.

```
query_sg_text = query_sg\
  .replace("--STATENAME--", "WY")\
  .replace("--FILTERS--", '')\
  .replace("--DATESTART--", "2022-01-01")\
  .replace("--DATEEND--", "2022-03-01")\
  .replace("--ENDCURSER--", "")
#.replace("--ENDCURSER--", "UGxhY2U6MjIzLTIyMkA1dzktajZqLTNrZiwxLjEzMDM4OTFfc2c6MTZiYTdlYzFiY2Q4
print(query_sg_text)
# .replace("--FILTERS--", 'brand: "Starbucks"')\

# Here we can change what information we want to grab from the data, by replacing the values for
```

Here we then convert the data into readable and usable data.

```
sgIter = client.execute(gql(query_sg_text))
```

This here is able to go through the data in it's sections.

```
# This here is able to go through the data in it's sections.
pageInformation = sgIter['search']['places']['results']['pageInfo']
nextPaging = pageInformation['endCursor']
edgesIter = sgIter['search']['places']['results']['edges']
sgIter = [dat.pop('node') for dat in edgesIter]
sgIter = [dat.pop('monthly_patterns') for dat in sgIter]
#sgIter = [dat.pop('safegraph_core') for dat in sgIter]
sgIter
```

Checking the Path here.

```
# Checking the Path here.
if Path('/tmp/safegraph').exists():
  print("do")
  shutil.rmtree('/tmp/safegraph')
  os.mkdir('/tmp/safegraph')
  dbutils.fs.mkdirs("/FileStore/safegraph")
else:
  os.mkdir('/tmp/safegraph')
  dbutils.fs.mkdirs("/FileStore/safegraph")
  print("hi")
```

This then writes the data to our docs.

```
with jsonlines.open("/tmp/safegraph/safegraph_example2.jl", 'w') as writer:
  writer.write_all(sgIter)
  writer.close()
print(os.listdir("/tmp/safegraph"))
dbutils.fs.ls("file:/tmp/safegraph")
```

dbutils helps us interact with our data. The fs.ls lists the contents of a directory. The fs.cp copies a file or directory, possibly across filesystems.

```
# dbutils.fs.cp("File:/tmp/safegraph/safegraph_example.jl", "dbfs:/FileStore/safegraph/safegraph
dbutils.fs.cp("file:/tmp/safegraph/", "dbfs:/FileStore/safegraph/", recurse = True)
dbutils.fs.ls("/FileStore/safegraph")
```

This then reads the data so we can then display it and use it.

```
# dfsg = spark.read.json("dbfs:/FileStore/safegraph/safegraph_example.jl", schema = schema)
dfsg = spark.read.json("dbfs:/FileStore/safegraph", schema = schema)
```

Removes a Database, Creates a Database.

```
%sql
-- IF EXISTS (DATABASE test_table)
-- DROP (DATABASE|SCHEMA) [IF EXISTS] test_table [RESTRICT | CASCADE];
DROP DATABASE test_table CASCADE;
CREATE DATABASE test_table;
```

Cleans up the data and gets rid of duplicates and then Saves the table.

```
# Creates a Database and removes the Database. ^
dfsg = dfsg.na.drop(subset=["placekey"])
dfsg = dfsg.persist()
print("Count with all json files: " + str(dfsg.count()))

dfsg = dfsg.dropDuplicates(['placekey', 'date_range_start'])
dfsg.persist()
print("Count after duplicates: " + str(dfsg.count()))

dfsg.repartition(10).write.format("delta").saveAsTable("test_table.test_table")

dfsg.unpersist()

print("done")

print(dfsg.limit(5).show())
# Cleans up the data and gets rid of duplicates and then Saves the table.
```

Grabbing a portion of information using Sql.

```
# Grabbing a portion of information using Sql.
df = spark.sql("SELECT * FROM test_table.test_table")
df = spark.table("test_table.test_table")

display(df)
```