Notebook

June 20, 2025

1 Project 1 - Starter Notebook

[0]:
```
from pyspark import SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("my_project_1").getOrCreate()
```

Importing all spark data types and spark functions for your convenience.

[0]:
```
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

[0]:
```
# Read a CSV into a dataframe
# There is a smarter version, that will first check if there is a Parquet file␣
 ↪and use it
def load_csv_file(filename, schema):
  # Reads the relevant file from distributed file system using the given schema

  allowed_files = {'Daily program data': ('Daily program data', "|"),
                   'demographic': ('demographic', "|")}

  if filename not in allowed_files.keys():
    print(f'You were trying to access unknown file \"{filename}\". Only valid␣
 ↪options are {allowed_files.keys()}')
    return None

  filepath = allowed_files[filename][0]
  dataPath = f"dbfs:/mnt/coursedata2024/fwm-stb-data/{filepath}"
  delimiter = allowed_files[filename][1]

  df = spark.read.format("csv")\
    .option("header","false")\
    .option("delimiter",delimiter)\
    .schema(schema)\
    .load(dataPath)
  return df

# This dict holds the correct schemata for easily loading the CSVs
```

```
schemas_dict = {'Daily program data':
                  StructType([
                    StructField('prog_code', StringType()),
                    StructField('title', StringType()),
                    StructField('genre', StringType()),
                    StructField('air_date', StringType()),
                    StructField('air_time', StringType()),
                    StructField('Duration', FloatType())
                  ]),
                'viewing':
                  StructType([
                    StructField('device_id', StringType()),
                    StructField('event_date', StringType()),
                    StructField('event_time', IntegerType()),
                    StructField('mso_code', StringType()),
                    StructField('prog_code', StringType()),
                    StructField('station_num', StringType())
                  ]),
                'viewing_full':
                  StructType([
                    StructField('mso_code', StringType()),
                    StructField('device_id', StringType()),
                    StructField('event_date', IntegerType()),
                    StructField('event_time', IntegerType()),
                    StructField('station_num', StringType()),
                    StructField('prog_code', StringType())
                  ]),
                'demographic':
                  StructType([StructField('household_id',StringType()),
                    StructField('household_size',IntegerType()),
                    StructField('num_adults',IntegerType()),
                    StructField('num_generations',IntegerType()),
                    StructField('adult_range',StringType()),
                    StructField('marital_status',StringType()),
                    StructField('race_code',StringType()),
                    StructField('presence_children',StringType()),
                    StructField('num_children',IntegerType()),
                    StructField('age_children',StringType()), #format like␣
 ↪range - 'bitwise'
                    StructField('age_range_children',StringType()),
                    StructField('dwelling_type',StringType()),
                    StructField('home_owner_status',StringType()),
                    StructField('length_residence',IntegerType()),
                    StructField('home_market_value',StringType()),
                    StructField('num_vehicles',IntegerType()),
                    StructField('vehicle_make',StringType()),
                    StructField('vehicle_model',StringType()),
```

```
                    StructField('vehicle_year',IntegerType()),
                    StructField('net_worth',IntegerType()),
                    StructField('income',StringType()),
                    StructField('gender_individual',StringType()),
                    StructField('age_individual',IntegerType()),
                    StructField('education_highest',StringType()),
                    StructField('occupation_highest',StringType()),
                    StructField('education_1',StringType()),
                    StructField('occupation_1',StringType()),
                    StructField('age_2',IntegerType()),
                    StructField('education_2',StringType()),
                    StructField('occupation_2',StringType()),
                    StructField('age_3',IntegerType()),
                    StructField('education_3',StringType()),
                    StructField('occupation_3',StringType()),
                    StructField('age_4',IntegerType()),
                    StructField('education_4',StringType()),
                    StructField('occupation_4',StringType()),
                    StructField('age_5',IntegerType()),
                    StructField('education_5',StringType()),
                    StructField('occupation_5',StringType()),
                    StructField('polit_party_regist',StringType()),
                    StructField('polit_party_input',StringType()),
                    StructField('household_clusters',StringType()),
                    StructField('insurance_groups',StringType()),
                    StructField('financial_groups',StringType()),
                    StructField('green_living',StringType())
                ])
}
```

2 Read demogrphic data

[0]:
```
%%time
# demographic data filename is 'demographic'
demo_df = load_csv_file('demographic', schemas_dict['demographic'])
demo_df.count()
demo_df.printSchema()
print(f'demo_df contains {demo_df.count()} records!')
display(demo_df.limit(6))
```

```
root
 |-- household_id: string (nullable = true)
 |-- household_size: integer (nullable = true)
 |-- num_adults: integer (nullable = true)
 |-- num_generations: integer (nullable = true)
 |-- adult_range: string (nullable = true)
 |-- marital_status: string (nullable = true)
```

```
 |-- race_code: string (nullable = true)
 |-- presence_children: string (nullable = true)
 |-- num_children: integer (nullable = true)
 |-- age_children: string (nullable = true)
 |-- age_range_children: string (nullable = true)
 |-- dwelling_type: string (nullable = true)
 |-- home_owner_status: string (nullable = true)
 |-- length_residence: integer (nullable = true)
 |-- home_market_value: string (nullable = true)
 |-- num_vehicles: integer (nullable = true)
 |-- vehicle_make: string (nullable = true)
 |-- vehicle_model: string (nullable = true)
 |-- vehicle_year: integer (nullable = true)
 |-- net_worth: integer (nullable = true)
 |-- income: string (nullable = true)
 |-- gender_individual: string (nullable = true)
 |-- age_individual: integer (nullable = true)
 |-- education_highest: string (nullable = true)
 |-- occupation_highest: string (nullable = true)
 |-- education_1: string (nullable = true)
 |-- occupation_1: string (nullable = true)
 |-- age_2: integer (nullable = true)
 |-- education_2: string (nullable = true)
 |-- occupation_2: string (nullable = true)
 |-- age_3: integer (nullable = true)
 |-- education_3: string (nullable = true)
 |-- occupation_3: string (nullable = true)
 |-- age_4: integer (nullable = true)
 |-- education_4: string (nullable = true)
 |-- occupation_4: string (nullable = true)
 |-- age_5: integer (nullable = true)
 |-- education_5: string (nullable = true)
 |-- occupation_5: string (nullable = true)
 |-- polit_party_regist: string (nullable = true)
 |-- polit_party_input: string (nullable = true)
 |-- household_clusters: string (nullable = true)
 |-- insurance_groups: string (nullable = true)
 |-- financial_groups: string (nullable = true)
 |-- green_living: string (nullable = true)

demo_df contains 357721 records!

CPU times: user 84.8 ms, sys: 12.8 ms, total: 97.6 ms
Wall time: 24.2 s
```

3 Read Daily program data

[0]:
```
%%time
# daily_program data filename is 'Daily program data'
daily_prog_df = load_csv_file('Daily program data', schemas_dict['Daily program␣
 ↪data'])

daily_prog_df.printSchema()
print(f'daily_prog_df contains {daily_prog_df.count()} records!')
display(daily_prog_df.limit(6))
```

```
root
 |-- prog_code: string (nullable = true)
 |-- title: string (nullable = true)
 |-- genre: string (nullable = true)
 |-- air_date: string (nullable = true)
 |-- air_time: string (nullable = true)
 |-- Duration: float (nullable = true)

daily_prog_df contains 13194849 records!

CPU times: user 21 ms, sys: 4.67 ms, total: 25.6 ms
Wall time: 16.1 s
```

4 Read viewing data

[0]:
```
dataPath = "dbfs:/FileStore/ddm/10m_viewing"

viewing10m_df = spark.read.format("csv")\
    .option("header","true")\
    .option("delimiter",",")\
    .schema(schemas_dict['viewing_full'])\
    .load(dataPath)

display(viewing10m_df.limit(6))
print(f'viewing10m_df contains {viewing10m_df.count()} rows!')
```

```
viewing10m_df contains 9935852 rows!
```

5 Read reference data

Note that we removed the 'System Type' column.

[0]:
```
# Read the new parquet
ref_data_schema = StructType([
    StructField('device_id', StringType()),
    StructField('dma', StringType()),
```

```
    StructField('dma_code', StringType()),
    StructField('household_id', IntegerType()),
    StructField('zipcode', IntegerType())
])

# Reading as a Parquet
dataPath = f"dbfs:/FileStore/ddm/ref_data"
ref_data = spark.read.format('parquet') \
                    .option("inferSchema","true")\
                    .load(dataPath)

display(ref_data.limit(6))
print(f'ref_data contains {ref_data.count()} rows!')
```

```
ref_data contains 704172 rows!
```

[0]:
```
daily_prog_df = daily_prog_df.drop('air_time')
avg_duration = daily_prog_df.select(avg(col("Duration"))).first()[0]
suspicious =["Collectibles", "Art", "Snowmobile", "Public affairs", "Animated",␣
 ↪"Music"]
title_word = ["better", "girls", "the", "call"]
suspicious_genre = False
daily_prog_temp = daily_prog_df.withColumn(
    'cnt_title',
    (
    when(lower(col('title')).contains('better'),1).otherwise(0) +
    when(lower(col('title')).contains('girls'),1).otherwise(0) +
    when(lower(col('title')).contains('the'),1).otherwise(0) +
    when(lower(col('title')).contains('call'), 1).otherwise(0)
    )
)
daily_prog_temp = daily_prog_temp.withColumn(
    "genre_array",
    split(col("genre"), ",")
)

daily_prog_temp = daily_prog_temp.withColumn(
    "genre_array",
    expr("transform(genre_array, x -> trim(x))")
)

daily_prog_temp = daily_prog_temp.withColumn(
    "is_suspicious_genre",
    expr(f"""
        size(
            filter(
                genre_array,
```

```
                g -> array_contains(array({','.join([f'"{g}"' for g in␣
 ↪suspicious])}), g)
            )
        ) > 0
    """)
)
display(daily_prog_temp.limit(6))
print(f' contains {daily_prog_temp.count()} rows!')
```

```
 contains 13194849 rows!
```

[0]:
```
ref_data = ref_data.withColumn("household_id", lpad(col("household_id"), 8,␣
 ↪"0"))
```

[0]:
```
genre_lookup = daily_prog_temp \
    .select('prog_code', 'genre_array') \
    .filter(col('prog_code').isNotNull() & col('genre_array').isNotNull()) \
    .withColumn('genre', explode('genre_array')) \
    .select('prog_code', 'genre').dropDuplicates(['genre', 'prog_code'])


temp4 = viewing10m_df.select('prog_code', 'device_id') \
    .join(genre_lookup, on='prog_code', how='inner')

temp4 = temp4.join(ref_data.select('device_id', 'household_id'),␣
 ↪on='device_id', how='inner')

temp4 = temp4.join(demo_df.select('household_id', 'household_size'),␣
 ↪on='household_id', how='inner')

temp4 = temp4.select('genre', 'household_id', 'household_size').
 ↪dropDuplicates(['genre', 'household_id'])

temp4 = temp4.groupBy('genre').agg(
    sum('household_size').alias('total_count')
).orderBy(col('total_count').desc())

display(temp4.limit(5))
top5_total = temp4.limit(5).agg(
    sum('total_count').alias('total_viewers_top5')
)

display(top5_total)
```

[0]:
```
# Step 1: Count devices per DMA
dma_device_count_df = ref_data \
```

```
    .filter(col('DMA').isNotNull()) \
    .groupBy('DMA') \
    .agg(countDistinct('device_id').alias('total_device'))

# Step 2: Join household-level info (household_id, DMA, household_size)
dma_people_df = ref_data.select('household_id', 'DMA').
 ↪dropDuplicates(['household_id']) \
    .join(
        demo_df.select('household_id', 'household_size'),
        on='household_id',
        how='inner'
    )

# Step 3: Sum household sizes per DMA
dma_people_sum_df = dma_people_df.groupBy('DMA').agg(
    sum('household_size').alias('total_people')
)

# Step 4: Join both (device count + people count)
dma_stats_df = dma_device_count_df.join(dma_people_sum_df, on='DMA',␣
 ↪how='inner') \
    .orderBy(col('total_device').desc())

# Step 5: Total people in top 5 DMAs
top5_people_total = dma_stats_df.limit(5).agg(
    sum('total_people').alias('total_people_in_top5_dmas')
)

dma_stats_df = dma_stats_df.drop('total_people')
display(dma_stats_df.limit(5))
display(top5_people_total)
```

[0]:
```
program_title = demo_df.
 ↪select('household_id','household_size','presence_children') \
.filter(col('presence_children') == 'Y') \
.join(ref_data.select('household_id','device_id'),on='household_id', ␣
 ↪how='inner')

program_title = viewing10m_df.select('device_id','prog_code').
 ↪join(program_title,on='device_id',  how='inner')

program_views = daily_prog_df.select('prog_code', 'title') \
    .filter(col('title').isNotNull())

program_title = program_title.join(program_views,on='prog_code',  how='inner')
```

```
program_title = program_title.select('household_id', 'title', 'household_size').
 ↪dropDuplicates(['household_id', 'title'])

program_title = program_title.groupBy('title').agg(sum('household_size').
 ↪alias('total_household')).orderBy(col('total_household').desc())

display(program_title.limit(5))
top5_total = program_title.limit(5).agg(
    sum('total_household').alias('total_people_top5')
)
display(top5_total)
```

part 2.2:

[0]:
```
demo_clean = demo_df.select('household_id','net_worth', 'income')\
.withColumn('income',
                              when(col('income') == 'A', 10.0)
                              .when(col('income') == 'B', 11.0)
                              .when(col('income') == 'C', 12.0)
                              .when(col('income') == 'D', 13.0)
                              .otherwise(col('income').cast("double"))
                              )\
    .filter(col('income').isNotNull() & col('net_worth').isNotNull())  # ␣
 ↪filter rows manually
# demo_clean= demo_clean.fillna({'net_worth':0})
# demo_clean= demo_clean.fillna({'income':0})

max_vals = demo_clean.agg(
    max('income').alias('max_income'),
    max('net_worth').alias('max_net_worth')
)
row = max_vals.first()

max_income = row['max_income']
max_net_worth = row['max_net_worth']

demo_clean = demo_clean.join(ref_data.
 ↪select('household_id','DMA'),on='household_id',  how='inner')

# Step 1: Calculate avg income and net worth per DMA
dma_avg = demo_clean.groupBy('DMA').agg(
    avg('income').alias('avg_income'),
    avg('net_worth').alias('avg_net_worth')
)

# Step 2: Compute wealth_score after the averages
demo_clean = dma_avg.withColumn(
```

```
    'wealth_score',
    (col('avg_income') / max_income) + (col('avg_net_worth') / max_net_worth)
).orderBy(col('wealth_score').desc()).limit(10)

# Step 3: Display
display(demo_clean)
```

[0]:
```
from pyspark.sql.functions import split, explode, countDistinct, col

# Step 1: Save top 10 DMA names
top_10_dma_names = demo_clean.select("DMA").rdd.flatMap(lambda x: x).collect()

# Step 2: Get device_id and household_id for top 10 DMAs from Reference Data
top_dma_devices = ref_data.filter(col("DMA").isin(top_10_dma_names)) \
    .select("device_id", "household_id", "DMA")

# Step 3: Join with Program Viewing Data  (correct name: viewing10m_df)
view_with_dma = viewing10m_df.join(top_dma_devices, on="device_id", how="inner")

# Step 4: Join with Daily Program Data  (correct name: daily_prog_df)
view_with_genres = view_with_dma.join(
    daily_prog_df.select("prog_code", "genre"),
    on="prog_code",
    how="inner"
)

# Step 5: Split prog_genres to array and explode
view_with_genres = view_with_genres.withColumn("genre",␣
 ↪explode(split(col("genre"), ",")))

# Step 6: Group by DMA and Genre and count distinct devices
dma_genre_counts = view_with_genres.groupBy("DMA", "genre") \
    .agg(countDistinct("device_id").alias("device_count")) \
    .orderBy("DMA", col("device_count").desc())

# Step 7: Show results
display(dma_genre_counts)
```

[0]:
```
from pyspark.sql import Row
from pyspark.sql.functions import col

# Step 1:        10 DMA
top_dma_df = demo_clean.orderBy(col("wealth_score").desc()).select("DMA",␣
 ↪"wealth_score")
top_dma_list = top_dma_df.rdd.map(lambda row: (row["DMA"],␣
 ↪row["wealth_score"])).collect()
```

```
# Step 2:                    '), DMA, (
remaining_df = dma_genre_counts

# Step 3:          '
used_genres = set()

# Step 4:
result_rows = []

for dma_name, score in top_dma_list:
    #         DMA           '
    filtered = remaining_df.filter((col("DMA") == dma_name) & (~col("genre").
 ↪isin(used_genres)))

    #     11   '
    top_genres = [row["genre"] for row in filtered.orderBy(col("device_count").
 ↪desc()).limit(11).collect()]

    #          '
    used_genres.update(top_genres)

    #
    result_rows.append(Row(DMA=dma_name, wealth_score=score,␣
 ↪top_11_genres=top_genres))

# Step 5:        Spark
final_result_df = spark.createDataFrame(result_rows)

#
display(final_result_df)
```

This notebook was converted with convert.ploomber.io