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 93.5 ms, sys: 13.1 ms, total: 107 ms
Wall time: 23.8 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 17.6 ms, sys: 7.37 ms, total: 25 ms
Wall time: 15.1 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]:
```
viewing_clean_df = viewing10m_df.select(['prog_code', 'device_id'])

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

```
 contains 9935852 rows!
```

[0]:
```
ref_data = ref_data.select(['device_id','household_id'])
ref_data = ref_data.withColumn('temp', lit(1))
device_count_df = ref_data.groupBy('household_id').agg(sum('temp').
 ↪alias('device_count'))
ref_data = ref_data.join(device_count_df, on='household_id', how='left')
ref_data = ref_data.drop('temp')
display(ref_data.limit(6))
print(f' contains {ref_data.count()} rows!')
```

```
 contains 704172 rows!
```

[0]:
```
demo_df = demo_df.
 ↪select(['household_id','num_adults','age_individual','age_2','vehicle_make','income'])
demo_df = demo_df.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")))
house_avg = demo_df.agg(avg(col('income'))).first()[0]
demo_df = demo_df.withColumn('cond_3',
                              when(
                                 (col('num_adults') == 2) &␣
 ↪(abs(col('age_individual')-col('age_2')) <= 6), True)
                              .otherwise(False))
```

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

```
 contains 357721 rows!
```

[0]:
```
daily_prog_temp = 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_temp.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
    """)
)
daily_prog_temp = daily_prog_temp.drop('genre')
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"))
```

part 1.2 :

[0]:
```
# Step 1: Get only households with Toyota vehicles
toyota_households_df = demo_df.filter(col('vehicle_make') == 91).
 ↪select('household_id')

# Step 2: Join viewing data to household mapping
viewing_ref_df = viewing_clean_df.join(ref_data, on='device_id', how='left')

# Step 3: Join only Toyota households (inner join filters early)
toyota_viewings_df = viewing_ref_df.join(toyota_households_df,␣
 ↪on='household_id', how='inner')

# Step 4: Get unique program codes watched by these households
toyota_prog_df = toyota_viewings_df.select('prog_code').distinct() \
    .withColumn('has_toyota_viewing', lit(True))

print(f' contains {toyota_prog_df.count()} rows!')
```

```
 contains 36494 rows!
```

[0]:
```
temp2 = daily_prog_df.select('prog_code', 'air_date', 'air_time', 'Duration')

temp2 = temp2.withColumn('air_date', to_date(col('air_date'), 'yyyyMMdd'))
temp2 = temp2.withColumn(
    'air_start',
    to_timestamp(concat_ws(' ', col('air_date'), col('air_time')), 'yyyy-MM-dd␣
 ↪HHmmss')
)

temp2 = temp2.withColumn('duration_int', floor(col('Duration')).cast('int'))

temp2 = temp2.withColumn(
    'air_end',
    expr("CAST(air_start AS TIMESTAMP) + duration_int * INTERVAL 1 MINUTE")
)
temp2 = temp2.filter(
    (
        (dayofmonth(col('air_date')) == 13) &
        (date_format(col('air_date'), 'E') == 'Fri')
    ) |
    (
        (dayofmonth(col('air_date')) == 12) &
        (date_format(col('air_date'), 'E') == 'Thu') &
        (dayofmonth(col('air_end')) == 13) &
```

```
        (date_format(col('air_end'), 'E') == 'Fri')
    )
)

friday_prog_df = temp2.select('prog_code').distinct() \
    .withColumn('is_friday', lit(True))

print(f"Friday the 13th programs (including Thursday overlap): {friday_prog_df.
 ↪count()}")
```

```
Friday the 13th programs (including Thursday overlap): 24350
```

[0]:
```
high_device_households_df = ref_data \
    .filter(col('device_count') > 3) \
    .select('device_id', 'household_id')

low_income_households_df = demo_df \
    .filter(col('income') <= house_avg) \
    .select('household_id')

cond_5_viewings_df = viewing_clean_df \
    .join(high_device_households_df, on='device_id', how='inner') \
    .join(low_income_households_df, on='household_id', how='inner')

cond_5_df = cond_5_viewings_df.select('prog_code').distinct() \
    .withColumn('cond_5', lit(True))

print(f"Condition #5 programs count: {cond_5_df.count()}")
```

```
Condition #5 programs count: 36396
```

[0]:
```
final_df = viewing_clean_df.join(ref_data, on='device_id', how='left')
print(f' contains {final_df.count()} rows1!')

final_df = final_df.join(demo_df.select('household_id', 'cond_3'),␣
 ↪on='household_id', how='left')
print(f' contains {final_df.count()} rows2!')

daily_clean = daily_prog_temp \
    .select('prog_code', 'title', 'duration', 'is_suspicious_genre',␣
 ↪'cnt_title').dropDuplicates(['prog_code'])

final_df = final_df.join(daily_clean, on='prog_code', how='left')
print(f' contains {final_df.count()} rows3!')

final_df = final_df.join(toyota_prog_df.select('prog_code',␣
 ↪'has_toyota_viewing'), on='prog_code', how='left')
print(f' contains {final_df.count()} rows4!')
```

```
final_df = final_df.join(friday_prog_df, on='prog_code', how='left')
print(f' contains {final_df.count()} rows5!')

final_df = final_df.join(cond_5_df, on='prog_code', how='left')
print(f' contains {final_df.count()} rows6!')
```

```
 contains 9935852 rows1!
 contains 9935852 rows2!
 contains 9935852 rows3!
 contains 9935852 rows4!
 contains 9935852 rows5!
 contains 9935852 rows6!
```

[0]:
```
final_df = final_df.fillna({
    'cond_3': False,
    'is_suspicious_genre': False,
    'cnt_title': 0,
    'has_toyota_viewing': False,
    'is_friday': False,
    'cond_5': False
})
```

[0]:
```
final_df = final_df.withColumn(
    'cond_1',
    when(col('duration').isNotNull() & (col('duration') > avg_duration), 1).
 ↪otherwise(0)
)

final_df = final_df.withColumn(
    'cond_2',
    when(col('has_toyota_viewing') == True, 1).otherwise(0)
)

final_df = final_df.withColumn(
    'cond_3',
    when(col('cond_3') == True, 1).otherwise(0)
)

final_df = final_df.withColumn(
    'cond_4',
    when(col('is_friday') == True, 1).otherwise(0)
)

final_df = final_df.withColumn(
    'cond_5',
    when(col('cond_5') == True, 1).otherwise(0)
```

```
)

final_df = final_df.withColumn(
    'cond_6',
    when(col('is_suspicious_genre') == True, 1).otherwise(0)
)

final_df = final_df.withColumn(
    'cond_7',
    when(col('cnt_title') >= 2, 1).otherwise(0)
)

# Add total score across all 7 flags
final_df = final_df.withColumn(
    'malicious_score',
    col('cond_1') + col('cond_2') + col('cond_3') +
    col('cond_4') + col('cond_5') + col('cond_6') + col('cond_7')
)

# Final malicious flag: at least 4 conditions met
final_df = final_df.withColumn(
    'is_malicious',
    when(col('malicious_score') >= 4, True).otherwise(False)
)
```

[0]:
```
malicious_titles = final_df.groupBy('title').agg(
    count('*').alias('total_count'),
    sum(when(col('is_malicious'), 1).otherwise(0)).alias('malicious_count')
).withColumn(
    'percent', col('malicious_count') / col('total_count')
).filter(
    col('percent') > 0.4
).orderBy(col('percent').desc())

# Show top 20 malicious titles
display(malicious_titles.limit(20))
```

[0]:
```
daily_prog_df.groupBy('title').agg(countDistinct('prog_code').
 ↪alias('num_prog_codes')).orderBy('num_prog_codes', ascending=False).show(20,␣
 ↪truncate=False)
```

```
+-------------------------+--------------+
|title                    |num_prog_codes|
+-------------------------+--------------+
|College Basketball       |3526          |
|MLB Baseball             |2804          |
|NULL                     |2590          |
|College Football         |2363          |
```

```
|NHL Hockey                |1697          |
|NBA Basketball            |1342          |
|Family Feud               |1188          |
|High School Football      |1119          |
|Women's College Basketball|1100          |
|Today                     |1090          |
|Judge Judy                |800           |
|College Baseball          |744           |
|Doctor Who                |701           |
|RightThisMinute           |636           |
|The Simpsons              |609           |
|Dr. Phil                  |598           |
|High School Basketball    |586           |
|Jerry Springer            |575           |
|Saturday Night Live       |568           |
|Maury                     |535           |
+--------------------------+--------------+
only showing top 20 rows
```

[0]:
```
daily_prog_df.groupBy('prog_code').agg(count('title').alias('title_count')).
  ↪orderBy('title_count', ascending=False).show(10)
```

```
+--------------+-----------+
|     prog_code|title_count|
+--------------+-----------+
|SH000000010000|      74680|
|SH009109720000|      58908|
|SH009109710000|      58908|
|SH000193310000|      33422|
|SH007772610000|      29267|
|SH000191120000|      28713|
|SH018126570000|      24746|
|SH000191160000|      23901|
|SH011027320000|      23143|
|SH015669460000|      20083|
+--------------+-----------+
only showing top 10 rows
```

[0]:
```
daily_prog_df.groupBy('prog_code').agg(countDistinct('title').
  ↪alias('num_titles')).filter(col('num_titles') > 1).show()
```

```
+--------------+----------+
|     prog_code|num_titles|
+--------------+----------+
|EP016186860070|         2|
|EP017516880027|         2|
|EP000048400405|         2|
```

```
|SH014712800000|         2|
|EP020942350002|         2|
|EP017516880047|         2|
|EP017825740017|         2|
|EP015515980009|         2|
|EP022802080002|         2|
|SH022422810000|         2|
|EP021732660001|         2|
|SH022250880000|         2|
|SH020932090000|         2|
|MV003834600000|         2|
|SH021553930000|         2|
|SH021548840000|         2|
|SH016285800000|         2|
|SH021551390000|         2|
|MV005735750000|         2|
|EP017825740015|         2|
+--------------+----------+
only showing top 20 rows
```

This notebook was converted with convert.ploomber.io