Project 1 - Starter Notebook

```python
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.

```python
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

```python
# 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',IntegerType()),
                    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())
                  ])
}
```

Read demogrphic data

```python
%%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: integer (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!
```

household_id	household_size	num_adults	num_generations	adult_range	marital_status	race_code	presence_children	num_children	age_children	age_range_children	dwelling_type	home_owner_status	length_residence	home_m
15	2	2	1	000000000000100000000	S	B	null	null	0000000000000000000	000000000000000	S	O	5	
24	2	2	1	000000000100000000000	null	W	null	null	0000000000000000000	000000000000000	M	O	null	
26	null	null	null	000000000000000000000	null	null	null	null	0000000000000000000	000000000000000	S	null	null	
28	3	2	2	000000110000000000000	S	W	Y	1	0000010000000000000	000001000000000	S	O	3	
35	1	1	1	000000000100000000000	null	W	null	null	0000000000000000000	000000000000000	null	null	null	
36	null	null	null	000000000000000000000	null	null	null	null	0000000000000000000	000000000000000	null	null	null	

```
CPU times: user 20.3 ms, sys: 10.9 ms, total: 31.3 ms
Wall time: 20.4 s
```

Read Daily program data

```python
%%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!
```

prog_code	title	genre	air_date	air_time	Duration
EP000000250035	21 Jump Street	Crime drama	20151219	050000	60.0
EP000000250035	21 Jump Street	Crime drama	20151219	110000	60.0
EP000000250063	21 Jump Street	Crime drama	20151219	180000	60.0
EP000000510007	A Different World	Sitcom	20151219	100000	30.0
EP000000510008	A Different World	Sitcom	20151219	103000	30.0
EP000000510159	A Different World	Sitcom	20151219	080300	29.0

```
CPU times: user 13.1 ms, sys: 5.37 ms, total: 18.5 ms
Wall time: 8.14 s
```

Read viewing data

```python
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!')
```

mso_code	device_id	event_date	event_time	station_num	prog_code
01540	0000000050f3	20150222	193802	61812	EP009279780033
01540	0000000050f3	20150222	195314	31709	EP021056430002
01540	0000000050f3	20150222	200151	61812	EP009279780033
01540	000000005518	20150222	111139	46784	EP004891370013
01540	000000005518	20150222	190000	14771	EP012124070127
01540	000000005518	20150222	200000	14771	EP010237320166

```
viewing10m_df contains 9935852 rows!
```

Read reference data

Note that we removed the 'System Type' column.

```python
# 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!')
```

device_id	dma	dma_code	household_id	zipcode
0000000050f3	Toledo	547	1471346	43609
000000006785	Amarillo	634	1924512	79119
000000007320	Lake Charles	643	3154808	70634
000000007df9	Lake Charles	643	1924566	70601
000000009595	Lexington	541	1600886	40601
000000009c6a	Houston	618	1924713	77339

```
ref_data contains 704172 rows!
```

Part 1

1.1

```python
from pyspark.sql.functions import col, lower, to_date, dayofmonth, date_format, avg, count, sum as spark_sum, abs, lit, coalesce
from functools import reduce

# 1.1 Extract and add features - conditions 1, 4, 6, 7

daily_prog_df = daily_prog_df.withColumn("air_date_parsed", to_date(col("air_date"), "yyyyMMdd")) \
    .withColumn("day", dayofmonth(col("air_date_parsed"))) \
    .withColumn("weekday", date_format(col("air_date_parsed"), "E"))

# 1st Condition: Duration > avg duration
avg_duration = daily_prog_df.select(avg("Duration")).first()[0]
prog_data_cond = daily_prog_df.withColumn("cond_1", col("Duration") > avg_duration)

# 6th Condition: Genre contains specific values
genre_keywords = ['Collectibles', 'Art', 'Snowmobile', 'Public affairs', 'Animated', 'Music']
genre_conditions = [col("genre").contains(keyword) for keyword in genre_keywords]
prog_data_cond = prog_data_cond.withColumn("cond_6", reduce(lambda a, b: a | b, genre_conditions))

# 7th Condition: Title contains at least 2 of the specific words
word_list = ['better', 'girls', 'the', 'call']
prog_data_cond = prog_data_cond.withColumn(
    "words_array",
    split(lower(col("title")), "\\s+")
)
flag_columns = [array_contains(col("words_array"), word).cast("int") for word in word_list]
prog_data_cond = prog_data_cond.withColumn(
    "title_flag_count",
    reduce(lambda a, b: a + b, flag_columns)
)
prog_data_cond = prog_data_cond.withColumn(
    "cond_7",
    col("title_flag_count") >= 2
)

prog_data_cond = prog_data_cond.drop("words_array")

# 4th Condition: Aired on Friday the 13th
prog_data_cond = prog_data_cond.withColumn("cond_4", (col("day") == 13) & (col("weekday") == "Fri"))

display(prog_data_cond.limit(6))
```

prog_code	title	genre	air_date	air_time	Duration	air_date_parsed	day	weekday	cond_1	cond_6	title_flag_count	cond_7	cond_4
EP000000250035	21 Jump Street	Crime drama	20151219	050000	60.0	2015-12-19	19	Sat	false	false	0	false	false
EP000000250035	21 Jump Street	Crime drama	20151219	110000	60.0	2015-12-19	19	Sat	false	false	0	false	false
EP000000250063	21 Jump Street	Crime drama	20151219	180000	60.0	2015-12-19	19	Sat	false	false	0	false	false
EP000000510007	A Different World	Sitcom	20151219	100000	30.0	2015-12-19	19	Sat	false	false	0	false	false
EP000000510008	A Different World	Sitcom	20151219	103000	30.0	2015-12-19	19	Sat	false	false	0	false	false
EP000000510159	A Different World	Sitcom	20151219	080300	29.0	2015-12-19	19	Sat	false	false	0	false	false

```python
# 1.1 Extract and add features - conditions 2,3,5
# 2nd Condition : Household has a vehicle made by 'Toyota'

# Join viewing data with household and demographic information
viewing_with_household = viewing10m_df.select(
    "device_id", "event_date", "event_time", "prog_code").join(
    ref_data.select("household_id", "device_id"),
    on="device_id",
    how="left")

viewing_demo = viewing_with_household.join(
    demo_df.select("household_id", "vehicle_make", "num_adults", "age_individual", "age_2", "income"),
    on="household_id",
    how="left"
)

# Filter only viewings where vehicle_make is Toyota ('91')
toyota_viewings = viewing_demo.filter(col("vehicle_make") == "91")

# Get distinct prog_codes watched by Toyota households
toyota_prog_codes_df = toyota_viewings.select("prog_code").distinct().withColumn("cond_2", lit(True))

# Join with main and set cond_2 flag
prog_data_cond = prog_data_cond.join(
    toyota_prog_codes_df,
    on="prog_code",
    how="left"
).withColumn(
    "cond_2",
    coalesce(col("cond_2"), lit(False))
)
display(prog_data_cond.limit(1000))
```

prog_code	title	genre	air_date	air_time	Duration	air_date_parsed	day	weekday	cond_1	cond_6	title_flag_count	cond_7	cond_4	cond_2
EP000005361169	The Best of the Joy of Painting	How-to,Art	20151219	180000	30.0	2015-12-19	19	Sat	false	true	1	false	false	false
EP000013866222	EastEnders	Soap,Drama	20151219	163000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false
EP000000250063	21 Jump Street	Crime drama	20151219	180000	60.0	2015-12-19	19	Sat	false	false	0	false	false	false
EP000019490059	Greatest Sports Legends	Sports non-event,Biography,Football	20151219	120000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false
EP000037900772	Sewing With Nancy	Educational,How-to	20151219	150000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false
EP000037900772	Sewing With Nancy	Educational,How-to	20151219	153000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false
EP000003690043	Are You Being Served?	Sitcom	20151219	123000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false
EP000017710056	Full House	Sitcom	20151220	030000	30.0	2015-12-20	20	Sun	false	false	0	false	false	false

```python
# 3rd Condition : Family with exactly 2 adults with age difference <= 6 between them

two_adult_households = viewing_demo.filter(col("num_adults") == 2)

# Calculate age difference between two adults
age_diff_df = two_adult_households.withColumn(
    "age_diff", abs(col("age_2") - col("age_individual"))
)

# Keep rows where age difference is 6 or less
age_diff_df_cond = age_diff_df.filter(col("age_diff") <= 6)

# Get distinct program codes that meet the condition
prog_codes_by_age_diff = age_diff_df_cond.select("prog_code").distinct().withColumn("cond_3", lit(True))

# Join with main data and set cond_3
prog_data_cond = prog_data_cond.join(
    prog_codes_by_age_diff,
    on="prog_code",
    how="left"
).withColumn(
    "cond_3",
    coalesce(col("cond_3"), lit(False))
)
display(prog_data_cond.limit(100))
```

prog_code	title	genre	air_date	air_time	Duration	air_date_parsed	day	weekday	cond_1	cond_6	title_flag_count	cond_7	cond_4	cond_2	cond_3
EP000000250063	21 Jump Street	Crime drama	20151219	180000	60.0	2015-12-19	19	Sat	false	false	0	false	false	false	false
EP000003690043	Are You Being Served?	Sitcom	20151219	123000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false	false
EP000003240027	The Andy Griffith Show	Sitcom	20151219	230000	30.0	2015-12-19	19	Sat	false	false	1	false	false	false	false
EP000000510180	A Different World	Sitcom	20151219	103000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false	false
EP000000510180	A Different World	Sitcom	20151219	133000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false	false
EP000003240109	The Andy Griffith Show	Sitcom	20151220	023000	30.0	2015-12-20	20	Sun	false	false	1	false	false	false	false
EP000002040083	All in the Family	Sitcom	20151219	213000	30.0	2015-12-19	19	Sat	false	false	1	false	false	false	false
EP000003690041	Are You Being Served?	Sitcom	20151219	113000	30.0	2015-12-19	19	Sat	false	false	0	false	false	false	false

```python
# 5th Condition : Household with more then 3 devices and income less than average
# Compute average income
avg_income = demo_df.select(avg(col("income"))).first()[0]

# Count number of devices per household
device_count_df = ref_data.groupBy("household_id").agg(
    countDistinct("device_id").alias("num_devices")
)
# Join income and device info
household_info_df = demo_df.join(device_count_df, on="household_id", how="inner")

# Keep households with > 3 devices and income < average
qualified_households_df = household_info_df.filter(
    (col("num_devices") > 3) & (col("income") < avg_income)
).select("household_id").distinct()

# Get device_ids from these households
qualified_devices_df = ref_data.join(
    qualified_households_df, on="household_id", how="inner"
).select("device_id").distinct()

# Get program codes watched on these devices
qualified_prog_codes_df = viewing10m_df.join(
    qualified_devices_df, on="device_id", how="inner"
).select("prog_code").distinct().withColumn("cond_5", lit(True))

# Join to main DataFrame and set cond_5 flag
prog_data_cond = prog_data_cond.join(
    qualified_prog_codes_df,
    on="prog_code",
    how="left"
).withColumn(
    "cond_5",
    coalesce(col("cond_5"), lit(False))
)
```

1.2

```python
# 1.2

condition_cols = [f"cond_{i}" for i in range(1, 8)]

# Calculate the number of conditions met per row
prog_data_cond = prog_data_cond.withColumn(
    "num_conditions_met",
    reduce(lambda a, b: a + b, [col(c).cast("int") for c in condition_cols])
)

# Flag malicious programs
prog_data_cond = prog_data_cond.withColumn(
    "is_malicious",
    when(col("num_conditions_met") >= 4, True).otherwise(False)
)

# Group by title and calculate malicious percentage
malicious_summary_df = prog_data_cond.groupBy("title").agg(
    (spark_sum(col("is_malicious").cast("int")) / count("*")).alias("malicious_percentage"),
    count("*").alias("total_records")
)

# Filter titles where > 40% are malicious and order by percentage descending
top_malicious_titles_df = malicious_summary_df.filter(
    col("malicious_percentage") > 0.4
).orderBy(col("malicious_percentage").desc()).limit(20)

display(top_malicious_titles_df.limit(20))
```

title	malicious_percentage	total_records
Philomena	1.0	658
Sabata	1.0	39
Weekends With Alex Witt	1.0	160
Lone Rider	1.0	96
Fox34 Weather Nation	1.0	2299
Today's Country: NASH	1.0	701

Meet the Browns	1.0	63
[illegible]	[illegible]	[illegible]