In [0]:

```
from pyspark.sql.types import *

from pyspark import SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("my project 1").getOrCreate()
sc = spark.sparkContext

# 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_PD_file(filename_or_dir, schema):
    dataPath = "/mnt/ddscoursedatastorage/fwm-stb-data/" + filename_or_dir
    df = spark.read.format("csv")\
      .option("header","false")\
      .option("delimiter", "|")\
      .schema(schema)\
      .load(dataPath)
    return df
```

In [0]:

```
df1 = spark.read.csv("/mnt/ddscoursedatastorage/dds-students/test.csv")
```

Reading the data files

Reading Refrence Data

In [0]:

```
# Reading the Reference Parquet files

ref_data = spark.read.parquet('/ref_data_raw').withColumnRenamed("_device-id","device_id"
)\
                                                .withColumnRenamed("_dma","dma")\
                                                .withColumnRenamed("_dma-code","dma_cod
e")\
                                                .withColumnRenamed("_household-id","hou
sehold_id")\
                                                .withColumnRenamed("_household-type","h
ousehold_type")\
                                                .withColumnRenamed("_system-type","syst
em_type")\
                                                .withColumnRenamed("_zipcode","zipcode"
)
ref_data_count = ref_data.count()
print(ref_data_count)
```

```
203581233
```

Reading Daily Program Data

In [0]:

```
# Reading the Daily Programs CSV file

daily_prog_schema =  StructType([StructField('prog_code',StringType()),
                     StructField('title',StringType()),
                     StructField('genre',StringType()),
                     StructField('air_date',StringType()),
                     StructField('air_time',StringType()),
                     StructField('Duration',FloatType())
                                  ])
```

```
daily_prog_data = load_PD_file("Daily program data/" , daily_prog_schema)
```

Reading Program Viewing Data

In [0]:

```
#Reading the 2.5% sample of the viewing data from a Parquet file

viewing_data = spark.read.parquet('/sample_viewing_2_5percent')

print(f'There are {viewing_data.count():,} entries in viewing_data dataframe!')
```

```
There are 130,289,194 entries in viewing_data dataframe!
```

Reading Demographic Data

In [0]:

```
# Reading the Demographic CSV file

demographic_schema =  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 - 'bi
twise'
                      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())
                                                 ])
demographic_data = load_PD_file("demographic/" , demographic_schema)
```

Question 2.1

In [0]:

```
from pyspark.sql.functions import *

# Removimg irrelevant columns and rows that contain null values

ref_data_Q2 = ref_data.distinct().select('device_id', 'dma')\
                      .filter((col('dma').isNotNull()) | (col('dma') != 'null')).distinc
t().cache()

daily_prog_data_Q2 = daily_prog_data.select('prog_code', 'genre').dropna()\
                                    .filter((col('prog_code') != 'null') | (col('genre'
) != 'null')).distinct().cache()

viewing_data_Q2 = viewing_data.distinct().select('prog_code', 'device_id').dropna()\
                              .filter((col('prog_code') != 'null') | (col('device_id') !
= 'null')).cache()
```

In [0]:

```
# Finding the connections between dmas and genres, using three of the given relations.
joined_1 = ref_data_Q2.dropna().filter(col('device_id') != 'null')\
                                .join(viewing_data_Q2, on='device_id', how='inner').cach
e()
joined = daily_prog_data_Q2.withColumn("single_genres", explode(split(col("genre"), ',')
))\
                        .drop('genre').join(joined_1.drop('device_id'), on='prog_code',
how='inner').cache()
joined.count()
```

Out[8]: 448056204

In [0]:

```
# Counting the number of views (non-distinct viewing events from 'viewing_data' relation)
for every (single) genre
dma_genre_count = joined.groupBy('dma', 'single_genres')\
                        .count().withColumnRenamed('count', 'views_count').cache()
```

In [0]:

```
from pyspark.sql.functions import *

# Creating a dataframe that contains all the genres that exist in the database
all_single_genres = dma_genre_count.select('single_genres').distinct().cache()
# Creating a dataframe that contains all the dmas that exist in the database
all_dma = dma_genre_count.select('dma').distinct().cache()

# Doing all necessary joins in order to make sure every dma has an entry for every existi
ng genre,
# and wherever the were no viewing events for a certain genre in that dma, we'll put a ze
ro in the 'views_count' column.
# Plus, ordering the relation as follows:
# all entries of a certain dma are one after the other in alphabethical order,
# then within each dma, order by number of vies of each genre.

all_combinations_dma_genres = all_single_genres.crossJoin(all_dma).cache()

dma_all_genres_without_zeros = all_combinations_dma_genres.join(dma_genre_count,
                                                                on=['dma', 'single_genr
es'],
                                                                how='left').cache()
dma_all_genres_count = dma_all_genres_without_zeros.fillna(0)\
                                                   .orderBy(desc('dma'), desc('views_co
unt')).cache()

all_single_genres.unpersist()
```

```
all_dma.unpersist()
all_combinations_dma_genres.unpersist()
```

Out[10]: DataFrame[single_genres: string, dma: string]

In [0]:

```
dma_all_genres_count.display()
```

dma	single_genres	views_count
Wilkes Barre-Scranton-Hztn	News	1236131
Wilkes Barre-Scranton-Hztn	Reality	1155820
Wilkes Barre-Scranton-Hztn	Sitcom	684124
Wilkes Barre-Scranton-Hztn	Talk	643980
Wilkes Barre-Scranton-Hztn	Comedy	603677
Wilkes Barre-Scranton-Hztn	Crime drama	590688
Wilkes Barre-Scranton-Hztn	Drama	558360
Wilkes Barre-Scranton-Hztn	Children	526718
Wilkes Barre-Scranton-Hztn	Action	520320
Wilkes Barre-Scranton-Hztn	Documentary	497372

In [0]:

```
# For each dma in the given list, print the 5 most viewed genres.
dma_list = ['Waco-Temple-Bryan', 'New York', 'Washington, DC (Hagrstwn)']

for dma in dma_list:
    tempi = dma_all_genres_count.filter(col('dma') == dma).cache()
    tempi.show(5)
    tempi.unpersist()
dma_all_genres_count.unpersist()
```

```
+-----------------+-------------+-----------+
|              dma|single_genres|views_count|
+-----------------+-------------+-----------+
|Waco-Temple-Bryan|      Reality|     317154|
|Waco-Temple-Bryan|         News|     294323|
|Waco-Temple-Bryan|         Talk|     191534|
|Waco-Temple-Bryan|       Sitcom|     184341|
|Waco-Temple-Bryan|        Drama|     159823|
+-----------------+-------------+-----------+
only showing top 5 rows

+--------+-------------+-----------+
|     dma|single_genres|views_count|
+--------+-------------+-----------+
|New York|         News|     260400|
|New York|      Reality|     207070|
|New York|         Talk|     138032|
|New York|       Sitcom|     124509|
|New York|  Documentary|     105943|
+--------+-------------+-----------+
only showing top 5 rows

+--------------------+-------------+-----------+
|                 dma|single_genres|views_count|
+--------------------+-------------+-----------+
|Washington, DC (H...|      Reality|     309259|
|Washington, DC (H...|         News|     275033|
|Washington, DC (H...|       Sitcom|     217373|
|Washington, DC (H...|       Comedy|     182383|
|Washington, DC (H...|     Children|     176962|
+--------------------+-------------+-----------+
only showing top 5 rows
```

Out[11]: DataFrame[dma: string, single_genres: string, views_count: bigint]

Question 2.2

In [0]:

```
from pyspark.sql.functions import *

# Altering values in demographic data:
# Converting households' incomes to numbers we can work with,
# Handling null values of certain columns.
demographic_data_Q2 = demographic_data.select('household_id', 'net_worth', 'income')\
                                        .withColumn('income', when(demographic_data.income
== 'A', '10')
                                                                .when(demographic_data.inco
me == 'B', '11')
                                                                .when(demographic_data.inco
me == 'C', '12')
                                                                .when(demographic_data.inco
me == 'D', '13')
                                                                .when((demographic_data.inc
ome == 'null') |
                                                                    (demographic_data.incom
e.isNull()), '0')
                                                                .otherwise(demographic_data
.income)          )\
                                        .withColumn('income', col('income').cast("int"))\
                                        .withColumn('net_worth', when((demographic_data.ne
t_worth == 'null') |
                                                                        (demographic_data.n
et_worth.isNull()), 0)
                                                                    .otherwise(demographic_d
ata.net_worth)).cache()

# Filtering out null values and irrelevant columns
ref_data_Q22 = ref_data.select('household_id', 'dma').filter((col('dma').isNotNull()) |
(col('dma') != 'null')).cache()
```

In [0]:

```
# calculating the maximum income of household in the data
max_income = demographic_data_Q2.select(max("income")).first()[0]
# calculatig the maximum net worth of household in the data
max_net_worth = demographic_data_Q2.select(max("net_worth")).first()[0]
```

In [0]:

```
from pyspark.sql.functions import *

# Calculating the wealth score of each dma.
# First, we'll find the connection between dmas and households using join,
# then calculating wealth score according to given formula,
# and lastly filtering out unnecessary columns.
dma_net_worth_income = ref_data_Q22.filter((col('household_id').isNotNull()) | (col('hous
ehold_id') != 'null'))\
                                        .join(demographic_data_Q2.filter((col('household_id')
.isNotNull()) |
                                                                        (col('household_id'
) != 'null')),
                                                 'household_id', 'left').cache()
wealth_per_dma = dma_net_worth_income.groupBy('dma')\
                                        .agg(avg('net_worth').alias('avg_networth'), avg('i
ncome').alias('avg_income'))
wealth_per_dma = wealth_per_dma.withColumn('wealth_score',
                                                 ((col('avg_networth')/max_net_worth) + (col('a
vg_income')/max_income)))\
                                    .select('dma', 'wealth_score').distinct().cache()

# Ordering the relation according to wealth score and then the number of views.
```

```
genres_per_dma = wealth_per_dma.join(dma_all_genres_count, 'dma', 'left')
genres_per_dma = genres_per_dma.distinct().orderBy(desc('wealth_score'), desc('views_coun
t')).cache()
```

In [0]:

```
# Creating a list of all (and only) dmas, arranged in order from wealthiest to least weal
thy
all_dma_by_order_from_wealthiest = genres_per_dma.select('dma').distinct().collect()
all_dma_by_order_from_wealthiest = [str(dma).split("'")[1] for dma in all_dma_by_order_fr
om_wealthiest]
```

In [0]:

```
used_genre = 'null'
# Creating a copy of the dataframe
genres_per_dma_updater = genres_per_dma.cache()
# Creating a copy of the dataframe's schema only
dma_genre_df = genres_per_dma_updater.limit(0)

# Go through all dmas in the data- starting at the wealthiest in descending order.
for dma in all_dma_by_order_from_wealthiest:
    # Initialize an empty dataframe for each dma so that we can add entries to it later.
    flag = -1
    # 8 in the required number of genres for each dma.
    for i in range(8):
        # Gets all the legal entries where the genres of each dma are not used,
        # and therfore can be chosen later for a specific dma.
        genres_per_dma_updater = genres_per_dma_updater.filter(col('single_genres') != u
sed_genre)\
                                                        .orderBy(desc('views_count')).ca
che()
        # Taking 1 entry of the required dma and its genre- ordered descending by number
of views.
        entry_dma_genre = genres_per_dma_updater.filter(col('dma') == dma).orderBy(desc(
'views_count')).limit(1).cache()
        genres_per_dma_updater.unpersist()
        if entry_dma_genre.count() == 0:
            # If there are no unused genres for this dma, get out of the loop.
            flag = i
            break
        # Adding the entry to the results dataframe
        dma_genre_df = dma_genre_df.union(entry_dma_genre)
        # Extract the genre that was chosen one step earlier,
        # so that we can remove all entries containing this genre in the next loop.
        used_genre = entry_dma_genre.select('single_genres').first()[0]
        entry_dma_genre.unpersist()

    if flag == 0:
        # We are here only if there was no genre to add to a certain dma at all.
        # In this case, we are adding an entry for this dma with an empty string as its g
enre.
        entry_dma_genre = genres_per_dma.filter(col('dma') == dma).limit(1)\
                                        .withColumn('single_genres', lit(None))\
                                        .withColumn('views_count', lit(0)).cache()
        dma_genre_df = dma_genre_df.union(entry_dma_genre)
        entry_dma_genre.unpersist()

# Combining all the registries of the same dma together, such that its genres are in a li
st.
# If this dma has no genres, it will contain an empty list.
dma_genre_df = dma_genre_df.orderBy(desc('views_count')).drop('views_count')\
                           .groupBy('dma', 'wealth_score')\
                           .agg(collect_list('single_genres').alias('genres_list'))
```

In [0]:

```
# Ordering the dmas by wealth score and then printing according to requested format.
dma_genre_by_wealth = dma_genre_df.orderBy(desc('wealth_score'))
dma_genre_by_wealth.display()
dma_genre_by_wealth.show(25)
```

dma	wealth_score	genres_list
San Antonio	1.623931623931624	List()
Baltimore	1.323423017384553	List(News, Reality, Talk, Comedy, Sitcom, Drama, Sports event, Documentary)
Detroit	1.310195919188455	List(Sports non-event, Adventure, Action, Crime drama, House/garden, Suspense, Children, Entertainment)
Austin	1.2711323615888497	List(Animated, Football, Crime, Fantasy, Newsmagazine, Mystery, Consumer, Special)
Sacramnto-Stkton-Modesto	1.2668983216070457	List(Music, Shopping, Law, Cooking, Science fiction, Game show, Interview, Sports talk)
San Francisco-Oak-San Jose	1.2523400679268373	List(Educational, Basketball, Western, Animals, Spanish, Travel, Nature, Outdoors)
Seattle-Tacoma	1.2461203339681173	List(Auto, Soap, Fashion, Docudrama, Soccer, Historical drama, Anthology, Biography)

```
+--------------------+------------------+--------------------+
|                 dma|      wealth_score|         genres_list|
+--------------------+------------------+--------------------+
|         San Antonio| 1.623931623931624|                  []|
|           Baltimore| 1.323423017384553|[News, Reality, T...|
|             Detroit| 1.310195919188455|[Sports non-event...|
|              Austin|1.2711323615888497|[Animated, Footba...|
|Sacramnto-Stkton-...|1.2668983216070457|[Music, Shopping,...|
|San Francisco-Oak...|1.2523400679268373|[Educational, Bas...|
|      Seattle-Tacoma|1.2461203339681173|[Auto, Soap, Fash...|
|Cleveland-Akron (...| 1.219783174336312|[Military, Awards...|
|              Toledo| 1.208659916433023|[History, Home im...|
|Harrisburg-Lncstr...|1.2079220508100939|[Public affairs, ...|
|        Philadelphia| 1.205012520599199|[Romance-comedy, ...|
|           Lexington|1.1977755217465422|[Adults only, Hor...|
|             Houston|1.1913376528875275|[Bicycle racing, ...|
|        Portland, OR|1.1353253977210136|[Mixed martial ar...|
|Washington, DC (H...|1.1144216105587266|[Standup, Debate,...|
|       Augusta-Aiken|1.1128715077227525|                  []|
|             Atlanta|1.1111980389237475|[Environment, Par...|
|Little Rock-Pine ...| 1.082137552102464|[Surfing, Holiday...|
|Sioux Falls(Mitch...| 1.073030643581126|[Cheerleading, Ga...|
|Miami-Ft. Lauderdale|  1.06682853472421|[Boat, Bullfighti...|
|              Eureka| 1.053123551146194|[Holiday special,...|
|    Dallas-Ft. Worth|1.0517613106688342|[Arm wrestling, D...|
|   Fargo-Valley City|1.0446334537058672|[Cricket, Speed s...|
|Greenville-N.Bern...|1.0406826988440798|[Fencing, Badmint...|
|            Amarillo|1.0374296708935984|[Holiday-children...|
+--------------------+------------------+--------------------+
only showing top 25 rows
```

Question 3

In [0]:

```
# Creating a path to put the partitions in:
path = '/Tomer_and_Amit'
dbutils.fs.mkdirs(path)

# Creating the required dataframe, with the genres as singles and joining all the program
s details.
daily_prog_data_Q3 = daily_prog_data.withColumn("single_genres", explode(split(col("genre
"), ',')))\
                                        .drop('genres').cache()
dma_no_empty_genres = dma_genre_by_wealth.filter(size(col('genres_list')) > 0)\
                                            .withColumn("single_genres", explode(col("genre
s_list")))\
                                            .drop('genres_list', 'wealth_score').cache()
dma_genre_programs = dma_no_empty_genres.join(daily_prog_data_Q3, 'single_genres', 'inner
```

```
')\
                                                .withColumnRenamed('single_genres', 'genre').cac
he()
daily_prog_data_Q3.unpersist()
dma_no_empty_genres.unpersist()

# Hierarchal partitioning according to dma and genre, and saving the dataframe as csv.
dma_genre_programs.write.option("header", True)\
                   .partitionBy("dma", "genre")\
                   .mode("overwrite")\
                   .csv(path)
```