Project 1 part 2

In [0]:

```python
from pyspark import SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("project_1_part_2").getOrCreate()
```

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

In [0]:

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

In [0]:

```python
# Read a CSV into a dataframe
# There is a smarter version, that will first check if there is a Parquet fi
def load_csv_file(filename, schema):
  # Reads the relevant file from distributed file system using the given sch

  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 vali
    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()),
                              #changed to match the type in ref_data for com
                    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 r
                    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

In [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: 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	adu
15	2	2	1	00000000000010
24	2	2	1	00000000010000
26	null	null	null	00000000000000
28	3	2	2	00000011000000
35	1	1	1	00000000010000
36	null	null	null	00000000000000

```
CPU times: user 24 ms, sys: 12 ms, total: 36.1 ms
Wall time: 23.1 s
```

Read Daily program data

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

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 11.8 ms, sys: 9.33 ms, total: 21.2 ms
Wall time: 8.36 s
```

Read viewing data

In [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!')
```

mso_code	device_id	event_date	event_time	station_num	prog_co
01540	0000000050f3	20150222	193802	61812	EP0092797800
01540	0000000050f3	20150222	195314	31709	EP0210564300
01540	0000000050f3	20150222	200151	61812	EP0092797800
01540	000000005518	20150222	111139	46784	EP0048913700
01540	000000005518	20150222	190000	14771	EP0121240701
01540	000000005518	20150222	200000	14771	EP0102373201

```
viewing10m_df contains 9935852 rows!
```

Read reference data

Note that we removed the 'System Type' column.

In [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!')
```

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!
```

In [0]:

part 2.1

section 1: 5 genres

firse we will create table that tells the size of every houshold

In [0]:

```
hhAndSize = demo_df.select(['household_id','household_size']).distinct()
#dissmising the duplicates
hhAndSize = hhAndSize.na.drop(subset=['household_size','household_id'])
display(hhAndSize.limit(10))
```

household_id	household_size
109	1
98	3
15	2
61	2
24	2
28	3
56	2
85	2
40	2
35	1

now we will select 'Device_id' and 'Houshold ID' from 'Referance Data' and we will NJ it with 'hhAndSize'

In [0]:

```
ref = ref_data.select(['household_id','device_id'])
#dissmising the null values
```

```
ref = ref.na.drop(how='any', subset=['household_id','device_id'])
display(ref.limit(10))
```

household_id	device_id
1471346	0000000050f3
1924512	000000006785
3154808	000000007320
1924566	000000007df9
1600886	000000009595
1924713	000000009c6a
1924725	000000009daa
2935414	000000009e5a
3521041	00000000a215
1924784	00000000a290

In [0]:

```
devByhh = ref.join(hhAndSize,'household_id', how = 'inner')
display(devByhh.limit(10))
```

household_id	device_id	household_size
3427	44e08edfe932	5
3427	001bd75b30fe	5
3427	001bd77477c6	5
3427	44e08ef7cd29	5
4879	0021bee1daf3	5
4879	44e08edfd518	5
4879	10ea5940e24e	5
4879	0021bef19dc4	5
5147	10ea5940b2ed	4
5147	0021bef2a645	4

now we will check 'daily program data' for duplication then we will join it with 'program viewing data' to the program to the device

In [0]:

```
dailyNoDupl = daily_prog_df.distinct()
dailyNoDupl = dailyNoDupl.select(['prog_code','genre'])
dailyAndView = dailyNoDupl.join(viewing10m_df,'prog_code',how = 'inner')
dailyAndView = dailyAndView.select(['prog_code','genre','device_id'])
#dissmising the null values
dailyAndView = dailyAndView.na.drop(how='any', subset=['device_id','genre','
display(dailyAndView.limit(10))
```

prog_code	genre	device_id
EP000000260097	Sitcom	000041a37ef0
EP000000260097	Sitcom	00000265d104
EP000000260097	Sitcom	00000226a78e
EP000000260097	Sitcom	00001078916a
EP000000260097	Sitcom	0014f8b88d1c
EP000000260097	Sitcom	001bd7605c3c
EP000000260097	Sitcom	0021be456116
EP000000260097	Sitcom	0021be484f5b
EP000000260097	Sitcom	00407bb8c914
EP000000260097	Sitcom	10ea5916caf0

now we will join the dailyAndView with devbyhh to assign every program the numbers of views

In [0]:

```
hhAndProg = dailyAndView.join(devByhh,'device_id',how = 'inner')
display(hhAndProg.limit(10))
```

device_id	prog_code	genre	household_id	household_size
000041779852	EP000000260110	Sitcom	408756	2
0021be1e9cdc	EP000000260110	Sitcom	58579	1
44e08ee2164a	EP000000260110	Sitcom	3869126	2
000000af852d	EP000000260110	Sitcom	3657702	6
000004b0bc5f	EP000000260110	Sitcom	3633287	3
000010572a4a	EP000000260110	Sitcom	3616528	3
00001078994a	EP000000260110	Sitcom	3671153	1
000015cb6d27	EP000000260110	Sitcom	3637273	4
000f21441fdc	EP000000260110	Sitcom	3226666	3
001692e3f81c	EP000000260110	Sitcom	2652026	1

In [0]:

```
hpr_split = hhAndProg.withColumn("genre_array", split(hhAndProg["genre"], ",

# explode the array into multiple rows
hpr_exploded = hpr_split.select('household_id', 'device_id', 'prog_code', 'h

display(hpr_exploded.limit(10))
```

household_id	device_id	prog_code	household_size	genre
408756	000041779852	EP000000260110	2	Sitcom
58579	0021be1e9cdc	EP000000260110	1	Sitcom
3869126	44e08ee2164a	EP000000260110	2	Sitcom
3657702	000000af852d	EP000000260110	6	Sitcom
3633287	000004b0bc5f	EP000000260110	3	Sitcom
3616528	000010572a4a	EP000000260110	3	Sitcom
3671153	00001078994a	EP000000260110	1	Sitcom
3637273	000015cb6d27	EP000000260110	4	Sitcom
3226666	000f21441fdc	EP000000260110	3	Sitcom
2652026	001692e3f81c	EP000000260110	1	Sitcom

the assumption in the question is that wenever a device in a household sees a program then everyone in the household saw it so if the same program viewed from several devices we have to make sure we won't count them again. moreover in this task we classify by the exposure to a genre so it does not matter if a family is watching only one program with some genre or 10 programs with this exsact genre the contribiution of the two house holds to the popularity of the genre is the same so for every household we are going to drop all duplications of household and genre

In [0]:

```
#making sure that household contributes to genre only once
hpr_clean = hpr_exploded.dropDuplicates(['genre','household_id']).select(['h
```

now we are going to aggrigate by genre sum the nuber of viewers in each one and take the top 5 **finnaly we display the result as requiered**

In [0]:

```
#aggrigating by genre
hpr_res = (hpr_clean.groupBy('genre').agg(sum('household_size').alias('sum_h

#computing the number of people affected by the top 5 genres
sum_res = hpr_res.agg(sum('sum_household_size').alias('total_top5')).collect
#displaing only the genres
hpr_res = hpr_res.select('genre')
display(hpr_res)
print(sum_res)
```

genre
News
Reality
Talk
Comedy
Sitcom

```
2775929
```

the top 5 most popular DMAs

first we will create a table that showes the number of devices in different DMA's and picking the 5 with the most devices

In [0]:

```
ref_data_noNull = ref_data.na.drop(subset=['dma','dma_code','household_id','
ref_no_dupl = ref_data_noNull.select(['device_id','dma','dma_code','househol
ref_clean = ref_data.select(['device_id','dma','dma_code','household_id','de
ref_res = ref_clean.groupBy('dma').count().alias('count').orderBy('count', a
display(ref_res)
```

dma	count
Charleston-Huntington	44803
Wilkes Barre-Scranton-Hztn	43561
Seattle-Tacoma	29892
Toledo	27169
Little Rock-Pine Bluff	27133

now we need to know how many people we will save if we will choose this devices. so we will make another table that showes the population of every DMA join it with the ref_res table and then sum the number of people, display ref_res, and print the sum

In [0]:

```
dmaAndSize = hhAndSize.join(ref_no_dupl,'household_id', how = 'inner')
dmaAndSize_clean = dmaAndSize.dropDuplicates(['household_id','dma']).select(
dmaAndSize_final = dmaAndSize_clean.groupBy('dma').sum('household_size').ali
dma_popularAndSize = dmaAndSize_final.join(ref_res,'dma',how = 'inner')
display(dma_popularAndSize)
```

dma	sum(household_size)	count
Little Rock-Pine Bluff	31652	27133
Seattle-Tacoma	35124	29892
Toledo	24108	27169
Wilkes Barre-Scranton-Hztn	42844	43561
Charleston-Huntington	60656	44803

displaying final results

In [0]:

```
#computing the number of people affected by the top 5 genres
res_sum = dma_popularAndSize.agg(sum('sum(household_size)').alias('total_top
ref_res = ref_res.orderBy('count', ascending=False).limit(5).select('dma')
display(ref_res)
print(res_sum)
```

dma
Charleston-Huntington
Wilkes Barre-Scranton-Hztn
Seattle-Tacoma
Toledo
Little Rock-Pine Bluff

```
194384
```

top 5 most popular programs

first we need to create table that have only households with children and the size of the household

In [0]:

```
#relating only to households with size
clean_demo_df = demo_df.dropna(subset=['household_size'])
#relating only to hh (households) with children in them (also dropping null
hhWithChildren = clean_demo_df.filter(demo_df.presence_children == 'Y').sele
display(hhWithChildren.limit(10))
hhWithChildren_clean = hhWithChildren.dropDuplicates(['household_id'])
```

household_id	household_size
28	3
85	2
98	3
122	3
177	5
210	3
228	3
237	3
238	2
241	4

now we will create another table that connects program titles with viewing by household (general)

In [0]:
```
#first step is to get the unique programs
daily_clean = daily_prog_df.distinct()
#second step is to take only the relevant columns before joining
daily_title = daily_clean.select(['prog_code','title'])
#third step is to join with the viewing data to associate the title with dev
titleAndView = daily_title.join(viewing10m_df,'prog_code',how = 'inner')
#fourth step is to take only the relevant columns before joining
titleAndView = titleAndView.select(['prog_code','title','device_id'])
devByhhNoSize = devByhh.drop('household_size')
#fifth step is to join with devByhhNoSize to associate the title with househ
hhAndtitle = titleAndView.join(devByhhNoSize,'device_id',how = 'inner')
display(hhAndtitle.limit(10))
```

device_id	prog_code	title	household_id
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351
75000019049878e	EP000000510011	A Different World	2127351

In [0]:
```
#drop null values from title
```

```
hhAndtitle = hhAndtitle.dropna(subset=['title'])
```

now to get the first part of this task and display the 5 most popular programs by household with children. so first we are going to join hhAndtitle with hhWithChildren so all the housholds that we are considering are household with children

In [0]:

```
#joining hhAndtitle with hhWithChildren_clean to get the number of children
hhWithChildren_title = hhAndtitle.join(hhWithChildren_clean,'household_id',h
display(hhWithChildren_title.limit(10))
```

household_id	device_id	prog_code	title	household_size
2127351	75000019049878e	EP000000510011	A Different World	3
2127351	75000019049878e	EP000000510011	A Different World	3
2127351	75000019049878e	EP000000510011	A Different World	3
2127351	75000019049878e	EP000000510011	A Different World	3
2127351	75000019049878e	EP000000510011	A Different World	3
2127351	75000019049878e	EP000000510011	A Different World	3
			A Different	

then we make sure that every household cotributes only once at most for every title

In [0]:

```
hhWithChildren_title = hhWithChildren_title.dropDuplicates(['household_id','
```

then we aggrigate by title and then sum by household size and getting the top 5

In [0]:

```
top_5prog = (hhWithChildren_title.groupBy('title').agg(sum('household_size')
top_5prog = top_5prog.select(['title'])
display(top_5prog)
```

title
College Basketball
Paid Programming
SportsCenter
The Big Bang Theory
Today

now that we have the top 5 we need to get the total number of people affected regardless to the children condition so we are going to crete again table that associets household size to program title, clean it and join it with the top 5 title to get the total number of people affected **finally we display the resaults as required**

In [0]:

```
sum_top5prog = hhAndtitle.join(devByhh,'device_id',how = 'inner')
sum_top5prog = sum_top5prog.dropDuplicates(['household_id','title'])
sum_top5prog = sum_top5prog.groupBy('title').agg(sum('household_size').alias
sum_top5prog = sum_top5prog.join(top_5prog,'title',how = 'inner')


total_sum_top5prog = sum_top5prog.agg(sum('sum_household_size').alias('total
sum_top5prog_final = sum_top5prog.select('title')
display(sum_top5prog_final)
print(total_sum_top5prog)
```

title
Today
The Big Bang Theory
College Basketball
SportsCenter
Paid Programming

```
612101
```

part 2.2

first we need to create a table that connects DMA to its households

In [0]:

```
dmaAndhh = dmaAndSize.select(['household_id','dma'])
display(dmaAndhh.limit(10))
```

household_id	dma
2935392	Jonesboro
1447214	Toledo
2935402	San Angelo
1924619	Kansas City
2935412	Houston
1962640	Dallas-Ft. Worth
2935416	Springfield, MO
2507423	Springfield, MO
2505146	Sherman-Ada
1924852	Austin

part a: computing each dma its welth score

claening the demo_df column transformung every A-D to 10-13

In [0]:

```
# Replace values in the 'income' column
demo_df = demo_df.withColumn(
    "income",
    when(col("income") == "A", 10)
    .when(col("income") == "B", 11)
    .when(col("income") == "C", 12)
    .when(col("income") == "D", 13)
    .otherwise(None)  # or you can keep original value with .otherwise(col("
    .cast(IntegerType())
)
```

joining dmaAndhh with demo_df and selecting the relevant columns for our calculations

In [0]:

```
#joining dmaAndhh with demo_df and selecting the relevant cols
dmaAndhh_demo = dmaAndhh.join(demo_df,'household_id',how = 'inner')
dmaAndhh_demo = dmaAndhh_demo.select(['household_id','dma','income','net_wor
#cleaning duplications and null values
dmaAndhh_demo = dmaAndhh_demo.dropDuplicates(['household_id','dma'])
dmaAndhh_demo = dmaAndhh_demo.na.drop(subset=['income','net_worth'])
dmaAndhh_demo = dmaAndhh_demo.drop('household_id')
display(dmaAndhh_demo.limit(10))
```

dma	income	net_worth
Seattle-Tacoma	13	7
Houston	10	9
Houston	13	6
Fargo-Valley City	13	7
Toledo	10	8
Abilene-Sweetwater	10	6
San Angelo	12	7
Lexington	13	8
Shreveport	12	7
Toledo	10	7

now we will aggregate by dma and produce the table with all the data we need to compute 'welth score'

In [0]:

```
dmaWithvar = dmaAndhh_demo.groupBy('dma').agg(
    max("income").alias("max_income"),
    avg("income").alias("avg_income"),
    max("net_worth").alias("max_net_worth"),
    avg("net_worth").alias("avg_net_worth")
)
```

now we create new column that computes welth score based on the dmaWithvar table

In [0]:

```
dma_Wscore = dmaWithvar.withColumn("wealth_score", (col("avg_net_worth") / c
dma_Wscore = dma_Wscore.select(['dma','wealth_score'])
dma_Wscore = dma_Wscore.orderBy('wealth_score', ascending=False).limit(10)
display(dma_Wscore)
```

dma	wealth_score
San Antonio	2.0
Buffalo	1.8233974358974359
Erie	1.8105413105413106
Los Angeles	1.807692307692308
Columbia-Jefferson City	1.800976800976801
Miami-Ft. Lauderdale	1.783180132236736
Clarksburg-Weston	1.7765567765567765
San Francisco-Oak-San Jose	1.770963270963271
Baltimore	1.767389049815828
Bend, OR	1.761985936501172

varifing the odd result in san antonio

```
In [0]: san_antonio = dmaWithvar.filter(dmaWithvar['dma'] == 'San Antonio')
        san_antonio = san_antonio.select(['max_income','avg_income','max_net_worth',
        display(san_antonio)
```

max_income	avg_income	max_net_worth	avg_net_worth
11	11.0	7	7.0

```
In [0]: san_antonio = demo_df.join(ref_data,'household_id',how='inner').filter(ref_d
        display(san_antonio.limit(10))
```

income	net_worth	household_id	dma
11	7	1920273	San Antonio

appearently it's right according to the given equeision the net worth is highest if you have only one row becuse then you have avg income/ net worth== max income/ net worth regardless to the actuak wealth..

part b giving every dma its top genres without repetitions

- first we will table that associet genre with devices and DMA's using the dailyAndView dataframe we've already created then we will aggregate the data by 'genre' and 'dma' and write it to paquet file there we will adress one dma at a time

In [0]:

```
dailyAndView = dailyAndView.dropDuplicates(['device_id','genre'])
genreAndDma = dailyAndView.join(ref_data,'device_id',how = 'inner')
genreAndDma = genreAndDma.select(['genre','dma','device_id'])
#minimizing the data to the relevant DMA's
genreAndDma = genreAndDma.join(dma_Wscore,'dma', how = 'inner')
genreAndDma = genreAndDma.groupBy('genre','dma').agg(count('device_id').alia
#writing the data to parquet file partitioned by dma
genreAndDma.write.mode("overwrite").partitionBy("dma").parquet("output/parti
```

now we have to make a set of all the genres possible

In [0]:

```
total_genres = daily_prog_df.na.drop(subset=["genre"]).select(['genre']).dis
total_genres_list = total_genres.rdd.flatMap(lambda x: x).collect()
dma_list = dma_Wscore.select('dma').rdd.flatMap(lambda x: x).collect()
```

now we will create the final column were every dma from the top 10 wealthiest gets the unique genre list by order

In [0]:

```
#creating the list we will put all the genre lists in
listOfGenreLists = []
# creating the genre lists of the top genres for each DMA
for dmacur in dma_list:
    #counter to get max 11 genres
    i = 0
    # the specific list of the current DMA
    dmacur_list = []
    #reading the parquet file for the current DMA

    df = spark.read.parquet("dbfs:/output/partitioned_by_dma").filter(f"dma
    #ordering the genres before putting them in the list
    df = df.orderBy('count', ascending=False)
    #inserting the ordered genres to the list
    df_list = df.select('genre').rdd.flatMap(lambda x: x).collect()
    #running over the current genre list to find the top remaining genres ir
    for item in df_list :
        if i < 11:
            #if the total genre list is empty no need to compare just insert
            if len(total_genres_list)==0:
                continiue
            if item in total_genres_list:
                dmacur_list.append(item)
                i += 1
                total_genres_list.remove(item)
    listOfGenreLists.append(dmacur_list)
```

now after we've got the lists for each dma all we need is to add the list we've created as a new column in the 'dma_Wscore' table

In [0]:

```
# Create list of Rows: (dma, genres)
rowlist = [Row(dma=dma_list[i], top_genres=listOfGenreLists[i]) for i in rar

# Create a DataFrame from it
genre_df = spark.createDataFrame(rowlist)
```

```python
# Make sure your DMA column in other DataFrame is named "dma" and is string
# Join the new genre_df to dma_df on "dma"
dma_with_genres = dma_Wscore.join(genre_df, on="dma", how="left")
display(genre_df)
```

dma	top_genres
San Antonio	List()
Buffalo	List(News, Reality, Sitcom, Drama, Talk,News, Crime drama, Comedy, Consumer, Game show, Music, Shopping)
Erie	List()
Los Angeles	List(Sports non-event,News,Sports talk, Talk, Documentary, Reality,Cooking, Reality,Documentary, Children,Educational, Sports non-event, Children,Sitcom, Comedy-drama, Reality,Auction,Collectibles, Reality,Home improvement,House/garden)
Columbia-Jefferson City	List(Sports event,Basketball, Reality,Law, Sitcom,Animated, Talk,Comedy, Reality,Crime, Reality,House/garden, Romance-comedy, Newsmagazine, Newsmagazine,Entertainment, Talk,Interview,Politics, Weather)
Miami-Ft.	List()

display the final results

In [0]:
```python
display(dma_with_genres)
```

dma	wealth_score	top_genres
San Antonio	2.0	List()
Buffalo	1.8233974358974359	List(News, Reality, Sitcom, Drama, Talk,News, Crime drama, Comedy, Consumer, Game show, Music, Shopping)
Erie	1.8105413105413106	List()
Los Angeles	1.807692307692308	List(Sports non-event,News,Sports talk, Talk, Documentary, Reality,Cooking, Reality,Documentary, Children,Educational, Sports non-event, Children,Sitcom, Comedy-drama, Reality,Auction,Collectibles, Reality,Home improvement,House/garden)
Columbia-Jefferson	1.800976800976801	List(Sports event,Basketball, Reality,Law, Sitcom,Animated, Talk,Comedy, Reality,Crime, Reality,House/garden, Romance-comedy,