Read Sub Demographic Data

In [0]:

```
from pyspark.sql.types import *

import pyspark.sql.functions as F

from pyspark import SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("my project 2")\
    .config("spark.kryoserializer.buffer.max", "512m")\
    .config('spark.jars.packages', 'org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0')\
        .getOrCreate()
sc = spark.sparkContext


for_students_demo_path = "/mnt/ddscoursedatastorage/ro/fwm-stb-data/proj_B_demographic"
demographic_df = spark.read.parquet(for_students_demo_path).cache()
#display(demographic_df)
```

In [0]:

```
demographic_df.count()
```

Out[2]: 357721

Feature Extraction

In [0]:

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

In [0]:

```
# calculatig the maximum household size of household in the data
max_household_size = demographic_df.select(max("household_size")).first()[0]

# calculatig the maximum num adults of household in the data
max_num_adults = demographic_df.select(max("num_adults")).first()[0]

# calculatig the maximum num generations of household in the data
max_num_generations = demographic_df.select(max("num_generations")).first()[0]

# calculatig the maximum length residence of household in the data
max_length_residence = demographic_df.select(max("length_residence")).first()[0]

# calculatig the maximum home market value of household in the data
max_home_market_value = demographic_df.select(max("home_market_value")).first()[0]

# calculatig the maximum net worth of household in the data
max_net_worth = demographic_df.select(max("net_worth")).first()[0]

# calculating the maximum education highest of household in the data
demographic_df = demographic_df.withColumn("education_highest", demographic_df.education_
highest.cast(DoubleType())).cache()
max_education_highest = demographic_df.select(max("education_highest")).first()[0]
```

In [0]:

```
#Normalizing the required columns
demographic_df_normalized = demographic_df.withColumn('household_size', (col('household_s
ize')/max_household_size))\
                                                .withColumn('num_adults', (col('num_adults')/
```

```
max_num_adults))\
                                                .withColumn('num_generations', (col('num_gener
ations')/max_num_generations))\
                                                .withColumn('length_residence', (col('length_r
esidence')/max_length_residence))\
                                                .withColumn('home_market_value', (col('home_ma
rket_value')/max_home_market_value))\
                                                .withColumn('net_worth', (col('net_worth')/max
_net_worth))\
                                                .withColumn("education_highest", (col('educati
on_highest')/max_education_highest))\
                                                .cache()
demographic_df.unpersist()
```

```
Out[5]: DataFrame[household_id: bigint, household_size: int, num_adults: int, num_generat
ions: int, marital_status: string, race_code: string, dwelling_type: string, home_owner_s
tatus: string, length_residence: int, home_market_value: double, net_worth: double, gende
r_individual: string, education_highest: double]
```

In [0]:

```
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
```

In [0]:

```
demographic_df_encoded = demographic_df_normalized.cache()
# List of categorical columns to index later and than present them as a feature vector
categorical_columns = ["marital_status", "race_code", "dwelling_type", "home_owner_status
", "gender_individual"]
# Indexing each categorical column and insert the indexed values as a new column with an
'index' suffix in its name
indexed_columns = [column + "_index" for column in categorical_columns]

indexers = [StringIndexer(inputCol=column,
                          outputCol=column + "_index") for column in categorical_columns
]
# Representing the categorical indexed columns as a feature vector

one_hot_encoders = [OneHotEncoder(inputCol=column + "_index",
                                  outputCol=column + "_encoded") for column in categoric
al_columns]
# Making sure all actions happen in the required order
encoding_pipeline = Pipeline(stages=indexers + one_hot_encoders)
model = encoding_pipeline.fit(demographic_df_encoded)
demographic_df_encoded = model.transform(demographic_df_encoded).cache()

# Dropping unused columns to make running time more efficient
demographic_df_encoded = demographic_df_encoded.drop(*(categorical_columns + indexed_col
umns)).cache()

demographic_df_encoded.show()
```

```
+------------+-----------------+-------------------+-----------------+--------------
---+-----------------+---------+-----------------+---------------------+-------------
---+---------------------+-------------------------+-------------------------+
|household_id|    household_size|         num_adults|   num_generations|  length_residenc
e|home_market_value|net_worth|education_highest|marital_status_encoded|race_code_encoded|
dwelling_type_encoded|home_owner_status_encoded|gender_individual_encoded|
+------------+-----------------+-------------------+-----------------+--------------
---+-----------------+---------+-----------------+---------------------+-------------
---+---------------------+-------------------------+-------------------------+
|          85|0.2222222222222222|0.16666666666666666|0.6666666666666666|                1.
0|            0.125|     0.05|             0.25|             (3,[],[])|    (3,[0],[1.0])
|        (1,[0],[1.0])|            (1,[0],[1.0])|                (1,[],[])|
|        2073|0.1111111111111111|0.16666666666666666|0.6666666666666666|                1.
0|             0.15|      0.1|             0.25|         (3,[0],[1.0])|    (3,[2],[1.0])
|        (1,[0],[1.0])|            (1,[0],[1.0])|                (1,[],[])|
|        2523|0.7777777777777778|                1.0|               1.0|                1
.0|              0.1|      0.1|              0.5|         (3,[0],[1.0])|    (3,[0],[1.0]
)|        (1,[0],[1.0])|            (1,[0],[1.0])|            (1,[0],[1.0])|
```

```
|        2717|0.3333333333333333| 0.3333333333333333|0.6666666666666666|0.733333333333333
3|            0.125|      0.2|             0.75|          (3,[1],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|             (1,[0],[1.0])|
|        3364|0.2222222222222222| 0.3333333333333333|0.6666666666666666|                1.
0|             0.1|      0.1|             0.25|          (3,[0],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|             (1,[0],[1.0])|
|        4046|0.4444444444444444|                 0.5|                1.0|                0
.4|            0.075|     0.05|             0.25|          (3,[0],[1.0])|    (3,[0],[1.0]
)|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        4303|0.1111111111111111|0.16666666666666666|0.3333333333333333|                1.
0|             0.15|      0.2|             0.25|          (3,[1],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|             (1,[0],[1.0])|
|        4559|0.3333333333333333| 0.3333333333333333|0.6666666666666666|                0.
8|            0.175|      0.2|              0.5|          (3,[1],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        5277|0.3333333333333333| 0.3333333333333333|0.6666666666666666|                1.
0|            0.125|     0.02|              0.5|          (3,[0],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|                 (1,[],[])|             (1,[0],[1.0])|
|        5440|0.1111111111111111|0.16666666666666666|0.3333333333333333|0.533333333333333
3|            0.225|      0.2|             0.25|          (3,[1],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        6856|0.4444444444444444| 0.3333333333333333|0.6666666666666666|                0.
6|            0.125|      0.2|             0.25|          (3,[2],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        6991|0.1111111111111111|0.16666666666666666|0.6666666666666666|                1.
0|            0.125|      0.1|             0.25|          (3,[0],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|             (1,[0],[1.0])|
|        7605|0.5555555555555556| 0.6666666666666666|                1.0|                1
.0|             0.2|      0.2|             0.25|          (3,[0],[1.0])|    (3,[0],[1.0]
)|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        8627|0.2222222222222222| 0.3333333333333333|0.3333333333333333|                1.
0|             0.2|      0.2|              0.5|          (3,[0],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        8953|0.4444444444444444|                 0.5|                1.0|                1
.0|             0.05|      0.2|             0.25|          (3,[2],[1.0])|    (3,[0],[1.0]
)|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        9552|0.8888888888888888| 0.6666666666666666|                1.0|0.93333333333333
33|            0.175|      0.2|             0.25|          (3,[0],[1.0])|    (3,[0],[1.0]
)|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|        9663|0.2222222222222222| 0.3333333333333333|0.3333333333333333|0.866666666666666
7|            0.025|     0.05|             0.25|          (3,[0],[1.0])|    (3,[1],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|       10326|0.4444444444444444| 0.6666666666666666|0.6666666666666666|0.866666666666666
7|            0.15|      0.2|             0.25|          (3,[0],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|                 (1,[],[])|
|       10843|0.5555555555555556| 0.3333333333333333|0.6666666666666666|                1.
0|            0.225|      0.2|              0.5|          (3,[0],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|             (1,[0],[1.0])|
|       11586|0.2222222222222222|0.16666666666666666|0.6666666666666666|0.333333333333333
3|             0.3|      0.1|             0.25|          (3,[1],[1.0])|    (3,[0],[1.0])
|        (1,[0],[1.0])|             (1,[0],[1.0])|             (1,[0],[1.0])|
+------------+------------------+-------------------+------------------+---------------
---+-----------------+---------+-----------------+----------------------+--------------
---+----------------------+-------------------------+-------------------------+
only showing top 20 rows
```

In [0]:

```
from pyspark.ml.feature import VectorAssembler

# assigning
demographic_df_feature_vector = demographic_df_encoded.cache()
demographic_df_encoded.unpersist()

# the columns needed to be presented in the feature vector later
fields = demographic_df_feature_vector.drop("household_id").columns

# Representing the fields (specified above) as a feature vector

assembler = VectorAssembler(
```

```
    inputCols=fields,
    outputCol="features")

demographic_df_feature_vector = assembler.transform(demographic_df_feature_vector).cache(
)
demographic_df_feature_vector.select("household_id", "features").show(7, truncate=False)
```

```
+------------+-------------------------------------------------------------------------------
-------------------------------------------------------+
|household_id|features
|
+------------+-------------------------------------------------------------------------------
-------------------------------------------------------+
|85          |[0.2222222222222222,0.1666666666666666,0.6666666666666666,1.0,0.125,0.05,0
.25,0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0]                |
|2073        |[0.1111111111111111,0.1666666666666666,0.6666666666666666,1.0,0.15,0.1,0.2
5,1.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,0.0]                  |
|2523        |[0.7777777777777778,1.0,1.0,1.0,0.1,0.1,0.5,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0
,1.0]                                                   |
|2717        |[0.3333333333333333,0.3333333333333333,0.6666666666666666,0.733333333333333
3,0.125,0.2,0.75,0.0,1.0,0.0,1.0,0.0,0.0,1.0,1.0,1.0]|
|3364        |[0.2222222222222222,0.3333333333333333,0.6666666666666666,1.0,0.1,0.1,0.25,
1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.0]                    |
|4046        |[0.4444444444444444,0.5,1.0,0.4,0.075,0.05,0.25,1.0,0.0,0.0,1.0,0.0,0.0,1.0
,1.0,0.0]                                               |
|4303        |[0.1111111111111111,0.1666666666666666,0.3333333333333333,1.0,0.15,0.2,0.2
5,0.0,1.0,0.0,1.0,0.0,0.0,1.0,1.0,1.0]                  |
+------------+-------------------------------------------------------------------------------
-------------------------------------------------------+
only showing top 7 rows
```

In [0]:

```
from pyspark.ml.functions import vector_to_array
from pyspark.ml.feature import PCA

# assigning
demographic_df_pca = demographic_df_feature_vector.cache()
demographic_df_feature_vector.unpersist()

# Reducting dimension of feature vector to 2
# Allows us to represent the data in a low dimensional representation with a good approxi
mation of the original data
pca = PCA().setInputCol("features")\
           .setOutputCol("pca")\
           .setK(2)
demographic_df_pca = pca.fit(demographic_df_pca).transform(demographic_df_pca).cache()

demographic_df_pca.select("household_id", "pca").show(7, truncate=False)
```

```
+------------+----------------------------------------------+
|household_id|pca                                           |
+------------+----------------------------------------------+
|85          |[-0.35625542326615856,-0.11097025338253631]   |
|2073        |[-1.0925316168664254,0.12397733233143128]     |
|2523        |[-1.4149098090986307,-0.9701798366702089]     |
|2717        |[0.06329368821084108,-1.158369672110097]      |
|3364        |[-1.2929454215296108,-0.9714139901152653]     |
|4046        |[-1.1192228102668804,-0.0063753503650831895]|
|4303        |[0.07476265653874331,-1.1529793790408362]     |
+------------+----------------------------------------------+
only showing top 7 rows
```

In [0]:

```
# Separating the 2 components of the low dimensional representation of the feature vector
to 2 different columns
demographic_df_pca = (demographic_df_pca
                        .withColumn("axis", vector_to_array("pca"))
                        .select(fields + [col("axis")[i] for i in range(2)])).cache()
```

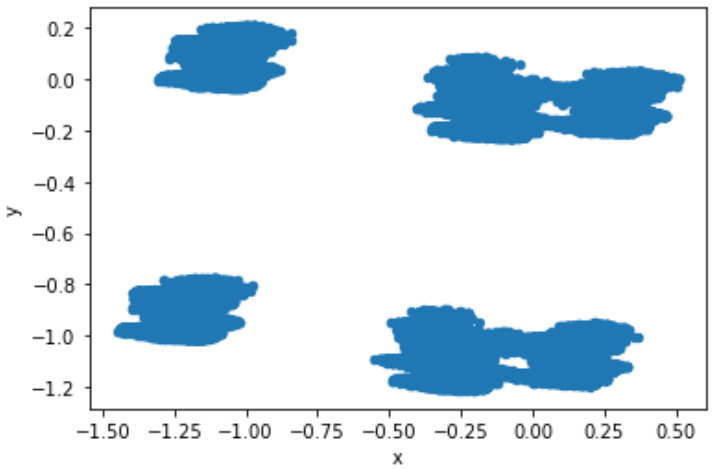
In [0]:

```
import pandas as pd

# Renaming the new 2 columns of the separated components of the pca result as x and y cor
respondingly
demographic_df_plot = demographic_df_pca.withColumnRenamed('axis[0]', 'x')\
                                        .withColumnRenamed('axis[1]', 'y').cache()
demographic_df_pca.unpersist()

# Plotting the results of the pca
# Allows us to visual analysing the created clusters
demographic_df_plot.toPandas().plot.scatter(x='x', y='y')
demographic_df_plot.unpersist()
```

```
/databricks/spark/python/pyspark/sql/pandas/conversion.py:119: UserWarning: toPandas atte
mpted Arrow optimization because 'spark.sql.execution.arrow.pyspark.enabled' is set to tr
ue; however, failed by the reason below:
  Unable to convert the field marital_status_encoded. If this column is not necessary, yo
u may consider dropping it or converting to primitive type before the conversion.
Direct cause: Unsupported type in conversion to Arrow: VectorUDT()
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is se
t to true.
  warn(msg)
Out[41]: DataFrame[household_size: double, num_adults: double, num_generations: double, l
ength_residence: double, home_market_value: double, net_worth: double, education_highest:
double, marital_status_encoded: vector, race_code_encoded: vector, dwelling_type_encoded:
vector, home_owner_status_encoded: vector, gender_individual_encoded: vector, x: double,
y: double]
```



Clustering

In [0]:

```
from pyspark.ml.clustering import KMeans

demographic_df_feature_vector.cache()

# Clustering the data according to the feature vector (training the algorithm on the demo
graphic_df_feature_vector data) into 6 clusters and assigning the cluster for each househ
old in a new column named cluster, using Kmeans algorithm
km = KMeans().setK(6).setSeed(3)
kmModel = km.fit(demographic_df_feature_vector)
kmModel.setFeaturesCol("features")
kmModel.setPredictionCol("cluster")
```

```
Out[12]: KMeansModel: uid=KMeans_ea254c8a53d3, k=6, distanceMeasure=euclidean, numFeature
s=16
```

In [0]:

```
# Performing the Kmeans algorithm to assign clusters to each household
```

```
demographic_df_kmeans = kmModel.transform(demographic_df_feature_vector).cache()
demographic_df_feature_vector.unpersist()
```

```
Out[13]: DataFrame[household_id: bigint, household_size: double, num_adults: double, num_
generations: double, length_residence: double, home_market_value: double, net_worth: doub
le, education_highest: double, marital_status_encoded: vector, race_code_encoded: vector,
dwelling_type_encoded: vector, home_owner_status_encoded: vector, gender_individual_encod
ed: vector, features: vector]
```

In [0]:

```
from pyspark.ml.linalg import Vectors

# Extracting the final cluster's centroids
centroids = kmModel.clusterCenters()

# Calculating the distance between each household and its cluster's centroid (according t
o the household feature vector)
dist_udf = F.udf(lambda features, cluster: float(Vectors.squared_distance(features, cent
roids[cluster])) ** 0.5)

# Calculating the distance specified above for each household
demographic_df_kmeans = demographic_df_kmeans.withColumn("distance", dist_udf(F.col('feat
ures'), F.col('cluster'))).cache()

demographic_df_kmeans.select("household_id", "features", "cluster", "distance").show(7,
truncate=False)
```

```
+------------+------------------------------------------------------------------------------
-----------------------------------------------------------+-------+------------------+
|household_id|features
|cluster|distance          |
+------------+------------------------------------------------------------------------------
-----------------------------------------------------------+-------+------------------+
|85          |[0.2222222222222222,0.1666666666666666,0.6666666666666666,1.0,0.125,0.05,0
.25,0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0]                  |0      |0.9986709922036282|
|2073        |[0.1111111111111111,0.1666666666666666,0.6666666666666666,1.0,0.15,0.1,0.2
5,1.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,0.0]                    |3      |0.8343119833761155|
|2523        |[0.7777777777777778,1.0,1.0,1.0,0.1,0.1,0.5,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0
,1.0]                                                     |1      |0.9026956010989436|
|2717        |[0.3333333333333333,0.3333333333333333,0.6666666666666666,0.7333333333333333
,0.125,0.2,0.75,0.0,1.0,0.0,1.0,0.0,0.0,1.0,1.0,1.0]|2      |0.6860553339107255|
|3364        |[0.2222222222222222,0.3333333333333333,0.6666666666666666,1.0,0.1,0.1,0.25,
1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.0]                      |1      |0.3251117743137002|
|4046        |[0.4444444444444444,0.5,1.0,0.4,0.075,0.05,0.25,1.0,0.0,0.0,1.0,0.0,0.0,1.0
,1.0,0.0]                                                 |0      |0.6022088823142537|
|4303        |[0.1111111111111111,0.1666666666666666,0.3333333333333333,1.0,0.15,0.2,0.2
5,0.0,1.0,0.0,1.0,0.0,0.0,1.0,1.0,1.0]                    |2      |0.7241865099609797|
+------------+------------------------------------------------------------------------------
-----------------------------------------------------------+-------+------------------+
only showing top 7 rows
```

In [0]:

```
from pyspark.sql.window import Window

# Partition the data by its clusters and in each cluster window sorting the households ac
cording to the distance we calculated earlier in acsending order
windowSpec = Window.partitionBy("cluster").orderBy("distance")
# Adding the index of the row of each entry/household for each window (cluster)
demographic_df_kmeans_window = demographic_df_kmeans.withColumn("row_number", row_number(
).over(windowSpec)).cache()
demographic_df_kmeans.unpersist()

# Getting the thirds and seventeenths subsets by taking each third and seventeenths row f
or each window (cluster) correspondingly separatly
thirds_subset = demographic_df_kmeans_window.where((col("row_number") % 3) == 0).cache()
seventeenths_subset = demographic_df_kmeans_window.where((col("row_number") % 17) == 0).
cache()
```

In [0]:

In [0]:

```
demographic_df_kmeans_window.unpersist()
thirds_subset.select("household_id", "cluster", "distance", "row_number").show()
thirds_subset.unpersist()
```

```
+------------+-------+--------------------+----------+
|household_id|cluster|            distance|row_number|
+------------+-------+--------------------+----------+
|       21444|      1|0.18475593031097687|         3|
|     2818306|      1|0.18526527964914238|         6|
|     1443732|      1|0.18526527964914238|         9|
|     3996824|      1| 0.1853431462289361|        12|
|     3777986|      1| 0.1853431462289361|        15|
|     2502868|      1| 0.1853431462289361|        18|
|     2167101|      1|0.18585088622990523|        21|
|     4028555|      1|0.18585088622990523|        24|
|     3800211|      1|0.18760672622982932|        27|
|     2797025|      1|0.18760672622982932|        30|
|     2274960|      1|0.18760672622982932|        33|
|     2233668|      1|0.18760672622982932|        36|
|     3791317|      1|0.18818504668448124|        39|
|     3994551|      1|0.18818504668448124|        42|
|     2146729|      1|0.18818504668448124|        45|
|     3617208|      1|0.18910762518173438|        48|
|     2814052|      1| 0.1896813695934358|        51|
|     4164515|      1| 0.1896813695934358|        54|
|     2028881|      1| 0.1936693410642834|        57|
|       92919|      1| 0.1936693410642834|        60|
+------------+-------+--------------------+----------+
only showing top 20 rows
```

```
Out[16]: DataFrame[household_id: bigint, household_size: double, num_adults: double, num_
generations: double, length_residence: double, home_market_value: double, net_worth: doub
le, education_highest: double, marital_status_encoded: vector, race_code_encoded: vector,
dwelling_type_encoded: vector, home_owner_status_encoded: vector, gender_individual_encod
ed: vector, features: vector, cluster: int, distance: string, row_number: int]
```

In [0]:

```
seventeenths_subset.select("household_id", "cluster", "distance", "row_number").show()
seventeenths_subset.unpersist()
```

```
+------------+-------+--------------------+----------+
|household_id|cluster|            distance|row_number|
+------------+-------+--------------------+----------+
|        8972|      1| 0.1853431462289361|        17|
|     3939052|      1|0.18760672622982932|        34|
|     2814052|      1| 0.1896813695934358|        51|
|     1465901|      1| 0.1942296108662116|        68|
|     3802857|      1|0.19608713359186963|        85|
|     2828482|      1|0.19674303987002664|       102|
|     1337137|      1|0.19729458128966731|       119|
|     2851208|      1|0.19894944991252952|       136|
|     1926031|      1| 0.2009069980480161|       153|
|     3175740|      1| 0.2047040768781287|       170|
|     1600418|      1| 0.2050919831215171|       187|
|     2793320|      1| 0.2056211312333509|       204|
|     2223363|      1|0.20682557620789077|       221|
|     1577978|      1|0.20735030031975427|       238|
|     3055935|      1|0.20818640577924993|       255|
|     3626930|      1|0.20891809129488123|       272|
|      106345|      1|0.21079197024981283|       289|
|      115234|      1|0.21233284938675112|       306|
|     2765482|      1| 0.2123401726365636|       323|
|     2754197|      1| 0.2130824724981291|       340|
+------------+-------+--------------------+----------+
only showing top 20 rows
```

```
Out[17]: DataFrame[household_id: bigint, household_size: double, num_adults: double, num_
generations: double, length_residence: double, home_market_value: double, net_worth: doub
le, education_highest: double, marital_status_encoded: vector, race_code_encoded: vector,
dwelling_type_encoded: vector, home_owner_status_encoded: vector, gender_individual_encod
```

```
ed: vector, features: vector, cluster: int, distance: string, row_number: int]
```

Visual Analysis

Read Static viewing data from Kafka

In [0]:

```
from pyspark.sql.functions import *
kafka_server = "kafka96224.eastus.cloudapp.azure.com:29092"
topic='viewstatic'
OFFSETS_PER_TRIGGER = 50000
SCHEMA = "device_id STRING, event_date STRING, event_time STRING, station_num STRING, pro
g_code STRING, household_id LONG"

static_df = spark.read\
                  .format("kafka")\
                  .option("kafka.bootstrap.servers", kafka_server)\
                  .option("subscribe", topic)\
                  .option("startingOffsets", "earliest")\
                  .option("failOnDataLoss",False)\
                  .load().cache()

static_view_data = static_df.select(from_csv(decode("value", "US-ASCII"), schema=SCHEMA)
.alias("value"))\
                            .select("value.*").na.drop().cache()
static_df.unpersist()
# display(static_view_data)
```

```
Out[18]: DataFrame[key: binary, value: binary, topic: string, partition: int, offset: big
int, timestamp: timestamp, timestampType: int]
```

In [0]:

```
static_view_data.count()
```

```
Out[19]: 7223153
```

Cluster's Viewing Analysis

In [0]:

```
# Getting only the relevant fields from the given dataframe
static_view_data_relevant_info = static_view_data.drop("event_date", "event_time").cache
()
static_view_data.unpersist()

# Dividing the given viewing events to clusters according to the k-means clustering we di
d earlier
viewing_events = static_view_data_relevant_info\
                                  .join(demographic_df_kmeans_window.select("household_id
", "cluster", "row_number"),
                                        on='household_id', how='inner')\
                                  .cache()
viewing_events.show()
```

```
+------------+------------+-----------+--------------+-------+----------+
|household_id|   device_id|station_num|     prog_code|cluster|row_number|
+------------+------------+-----------+--------------+-------+----------+
|     1496400|000000033449|      59444|MV000963020000|      2|      9963|
|     1477911|000000033633|      16374|EP018001900333|      1|     35048|
|     1464390|0000000792ce|      60179|EP018001900333|      2|     12174|
|     1491604|00000007a06a|      19746|SH000299490000|      1|     17610|
|     1468157|00000007a196|      61812|EP002654380201|      4|      6204|
|     3250955|00000007a236|      11207|EP003169780032|      4|     11572|
|     1462990|0000000a0707|      79051|EP006331691288|      0|     57108|
|     1478303|0000000cb631|      21247|EP005927330074|      0|     54232|
```

```
|     1492519|0000000dee65|      10142|EP003118650048|      1|     81762|
|     1479402|0000000e5290|      11066|EP000548690053|      5|     12327|
|     3092685|0000001095d5|      10153|EP010221680112|      0|     94023|
|     1455470|000000178c84|      19600|EP014284730643|      0|     42550|
|     1510764|0000001b3149|      10989|EP002654380201|      2|     23909|
|     1450476|0000001d2023|      58718|SH018377390000|      1|     88717|
|     1478486|0000001fffd3|      67567|SH011815290000|      2|      8842|
|     1496452|0000002276ca|      16123|EP001151270222|      5|      4628|
|     1472737|000000227923|      16123|EP001151270222|      0|     90408|
|     1455695|000000232ad3|      36212|SH006609190000|      0|     54517|
|     1476706|0000002334c7|      44366|EP000191775951|      1|     67389|
|     1473431|0000002cfa80|      14321|MV001811990000|      1|      3973|
+------------+------------+-----------+--------------+-------+----------+
only showing top 20 rows
```

In [0]:

```
# Calculate the percentage of views in every station w.r.t the given dataframe
def count_viewing_events(df):
    num_rows = df.count()
    return df.groupBy('station_num', 'general_percent_events').agg(((count(col('household
_id'))/num_rows)*100)\
                                                .alias("percent_num_events"))

# Calculating the diff_rank value for each station as defined in bullet no. 4
def substruct_columns(df, col1='percent_num_events', col2='general_percent_events'):
    df.withColumn('diff_rank', df[col1]-df[col2]).drop(col1, col2).orderBy(col('diff_ran
k').desc()).show(7, truncate=False)

# Printing te
def print_title(cluster, subset=''):
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    print(subset, ", cluster", cluster, ":")
```

In [0]:

```
# Calculating the percentage of views in every station w.r.t the entire dataframe (withou
t dividing to clusters)
num_rows = viewing_events.count()
station_general_popularity_rating = viewing_events.groupBy('station_num').agg(((count(col
('household_id'))/num_rows)*100)\
                                                        .alias("general_percent_events")).cac
he()

# Divide the data to clusters according to the k-means division from earlier,
# and finding the diff_rank value of each station for each set:
# 3 sets for each cluster, one of the full data, another is the 3rds subset, and the last
is the 17ths subset.
for cluster in range(6):
    cluster_df = viewing_events.filter(col('cluster') == cluster)\
                                .join(station_general_popularity_rating, 'station_num', '
inner').cache()
                                # Adding the column of the percentage of views in every s
tation w.r.t the entire dataframe


    print_title(cluster, "full data")
    # Calculating the diff_rank value for each station in the full data
    substruct_columns(count_viewing_events(cluster_df))

    print_title(cluster, "thirds subset")
    # Calculating the diff_rank value for each station in the 3rds subset
    substruct_columns(count_viewing_events(cluster_df.where((col("row_number") % 3) == 0
)))

    print_title(cluster, "seventeenths subset")
    # Calculating the diff_rank value for each station in the 17rds subset
    substruct_columns(count_viewing_events(cluster_df.where((col("row_number") % 17) ==
0)))
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
full data , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|16374      |0.15923312579402493|
|60179      |0.14842073350135188|
|14902      |0.11097756387408242|
|11221      |0.11008892738106524|
|49788      |0.10625692103643791|
|61522      |0.10264767102404004|
|11069      |0.08976342159548512|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thirds subset , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.31254032258329056|
|74796      |0.1863563815667193 |
|14902      |0.15242273757985458|
|61522      |0.1262364192961224 |
|11221      |0.10482651554376776|
|31709      |0.0958460310449806 |
|11865      |0.08921634648613072|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
seventeenths subset , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|14902      |0.5769473259115614 |
|11221      |0.3911958839718861 |
|74796      |0.31833500072735776|
|11066      |0.2637622527462131 |
|12131      |0.21366377572852113|
|12574      |0.204954939334816  |
|10335      |0.1953586276967552 |
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
full data , cluster 1 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.2722119018380882 |
|16374      |0.16436943002324877|
|19606      |0.1553240200859785 |
|11661      |0.12009206177267501|
|14771      |0.11737229337020638|
|57708      |0.11567424946744015|
|11713      |0.11211306315569408|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thirds subset , cluster 1 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.3257844309953972 |
|16374      |0.24645356348572767|
|19606      |0.20289683072756137|
|32645      |0.159782239836183  |
|11713      |0.14509478696830153|
|57708      |0.13523485370831273|
|56905      |0.1289513287195525 |
+-----------+-------------------+
```

```
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
seventeenths subset , cluster 1 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.28003766213989745|
|11150      |0.2283738314740873 |
|14771      |0.22062384036886917|
|31258      |0.21836150390360862|
|16616      |0.20818063818017263|
|16331      |0.18193791686009664|
|58515      |0.15870132749277654|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
full data , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|15433      |0.11791950188739703|
|10145      |0.10630224656836074|
|11865      |0.08286711119238352|
|10510      |0.07806408039842058|
|11187      |0.0738676106597328 |
|21762      |0.07351547418256477|
|34240      |0.07267485985629957|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thirds subset , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|16615      |0.2315125425597323 |
|15433      |0.18577040793834743|
|11187      |0.17533518315085106|
|12574      |0.16603415578248382|
|11865      |0.16167063242779958|
|11069      |0.15310719832753994|
|61623      |0.13955625920803805|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
seventeenths subset , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|15433      |0.4636533024595092 |
|16615      |0.40436055486139344|
|14909      |0.3403616163267565 |
|10142      |0.32774289405195   |
|12574      |0.260043981687216  |
|53041      |0.2597496887294996 |
|18279      |0.2486768691882114 |
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
full data , cluster 3 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|12131      |1.056068359926223  |
|10222      |0.7488988720765152 |
|11118      |0.7024079960894666 |
|10171      |0.64983723941455   |
|59684      |0.5041490318723634 |
```

```
|59061      |0.5011905167250051 |
|44714      |0.40364589844594684|
|10730      |0.35146801237026104|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thirds subset , cluster 3 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|12131      |1.0897531262690612 |
|10222      |0.884679428690159  |
|11118      |0.8396550589842099 |
|10171      |0.756328220117088  |
|10559      |0.5590651851841111 |
|44714      |0.41829718903832136|
|31738      |0.3929774164488179 |
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
seventeenths subset , cluster 3 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|11118      |0.8599170424831888|
|56905      |0.7207532651789998|
|12131      |0.6974424589571326|
|10222      |0.6192752164448058|
|11150      |0.5734656895072621|
|12458      |0.5562752293844562|
|55887      |0.5523719501377466|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
full data , cluster 4 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|35513      |1.1901430650335738|
|70387      |0.945459560515887 |
|11706      |0.8962938474500584|
|10918      |0.8507134976471332|
|12131      |0.7687667798064091|
|10179      |0.7128876721695723|
|10171      |0.6626916471313644|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thirds subset , cluster 4 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|35513      |1.1046927036461247|
|11706      |0.9486558852530687|
|70387      |0.9327355291584859|
|10918      |0.7675099314386936|
|11561      |0.7386222846752675|
|10171      |0.6727230123668573|
|10179      |0.6606831130501856|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
seventeenths subset , cluster 4 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|12131      |1.2448686852944777|
|35513      |1.184330348328002 |
```

```
|10171      |1.0223184642155   |
|70387      |0.9302948503882572|
|10918      |0.8262846990017919|
|16615      |0.8142938758481012|
|10153      |0.6279588910862992|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
full data , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.2987210477817668 |
|49788      |0.23767370375439967|
|59684      |0.14301584761738728|
|32645      |0.11185536490272607|
|99995      |0.09096949902615714|
|58646      |0.08850283040695028|
|14902      |0.08600050175988172|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
thirds subset , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.3096251896706763 |
|49788      |0.2518478624181806 |
|59684      |0.15199265000281337|
|14771      |0.1474698195681141 |
|99995      |0.14657116586663   |
|14902      |0.1295171259638206 |
|64241      |0.10802550614325668|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
seventeenths subset , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|59684      |0.9002360741864    |
|30646      |0.6720441208603702 |
|14771      |0.551159945146154  |
|60179      |0.4806567004581681 |
|47540      |0.47319229688287623|
|32645      |0.4464661579335747 |
|49788      |0.3399283869467018 |
+-----------+-------------------+
only showing top 7 rows
```

Read Streaming viewing data from Kafka

In [0]:

```
topic = "viewstream"
streaming_df = spark.readStream\
                   .format("kafka")\
                   .option("kafka.bootstrap.servers", kafka_server)\
                   .option("subscribe", topic)\
                   .option("startingOffsets", "earliest")\
                   .option("failOnDataLoss",False)\
                   .option("maxOffsetsPerTrigger", OFFSETS_PER_TRIGGER )\
                   .load()\
                   .select(F.from_csv(F.decode("value", "US-ASCII"), schema=SCHEMA).alias
("value")).select("value.*")
```

Dynamic Data Analysis - Streaming

In [0]:

```
# Only taking the relevant columns from the 3rds subset.
thirds_subset = thirds_subset.select('household_id', 'cluster')
```

In [0]:

```
# Calculating the diff_rank value for each station in the given dataframe.
def calculate_diff_rank(df, col1='viewing_num', col2='general_percent_events'):
    # Getting the number of events in the dataframe
    num_events = df.groupBy().sum('viewing_num').collect()[0][0]
    # Calculating the percentage of views in every station w.r.t the given dataframe
    df = df.withColumn('percent_num_events', (df[col1]/num_events)*100).drop(col1)
    # Substructing the percentage of views in every station w.r.t the entire data
    # from the percentage of views in every station w.r.t the given dataframe.
    df.withColumn('diff_rank', df['percent_num_events']-df[col2])\
        .drop('percent_num_events', col2)\
        .orderBy(col('diff_rank').desc())\
        .show(7, truncate=False)
```

In [0]:

```
schema = StructType([
    StructField("station_num", StringType()),
    StructField("viewing_num", DoubleType())
])

batch_schema = StructType([
    StructField("station_num", StringType()),
    StructField("general_num_events", DoubleType()),
])

# Initializing global variables, to be used inside the following 'handle_batch' function.
max_num_batches = 0
empty_df = spark.createDataFrame(spark.sparkContext.emptyRDD(), schema)
clusters_stations_count = [empty_df for _ in range(6)] #a list of (currently) empty data
frames,
                                                        #containing the fields 'station_
num', 'viewing_num'.
unioned_batches = spark.createDataFrame(spark.sparkContext.emptyRDD(), batch_schema)
num_rows = 0
```

In [0]:

```
# Divide the data to clusters according to the k-means division from earlier,
# and finding the diff_rank value of each station for each cluster, using only the 3rds s
ubset.
def handle_batch(batch, epoch_num):
    global max_num_batches
    global clusters_stations_count
    global unioned_batches
    global num_rows
    max_num_batches = epoch_num #because we want to know the serial number of the current
batch

    num_rows += batch.count() #counting the number of events in all the batches up until
the current one (inclusive)
    # Counting the number of events per station in all the batches recieved up until the
current one (inclusive).
    unioned_batches = batch.groupBy('station_num')\
                            .agg(count(col('household_id')).alias("general_num_events"))
\
                            .union(unioned_batches)\
                            .groupBy('station_num')\
                            .agg(sum(col('general_num_events')).alias("general_num_event
s")).cache()
```

```
    # Calculating the percentage of views in every station w.r.t the entire dataframe (wi
thout dividing to clusters)
    station_general_percent = unioned_batches.withColumn('general_percent_events', (col('
general_num_events')/num_rows)*100)\
                                                .drop('general_num_events')\
                                                .cache()
    unioned_batches.unpersist()
    # Divide the data to clusters according to the k-means division from earlier.
    batch_to_clusters = thirds_subset.join(batch, 'household_id', 'inner').cache()

    for cluster in range(6):
        print_title(cluster, 'batch number ' + str(epoch_num))
        # Counting the number of events per station in every cluster (separately) for the
current batch,
        # and then adding these countings to the countings of previous batches.
        clusters_stations_count[cluster] = batch_to_clusters.filter(col('cluster') == cl
uster)\
                                                                .groupBy('station_num')\
                                                                .agg(count('household_id').
alias('viewing_num'))\
                                                                .union(clusters_stations_co
unt[cluster])\
                                                                .groupBy('station_num')\
                                                                .agg(sum('viewing_num').ali
as('viewing_num')).cache()

        # Adding the column of the percentage of views in every station w.r.t the entire
dataframe.
        clusters_stations_count_with_general = clusters_stations_count[cluster].join(sta
tion_general_percent,
                                                                                        's
tation_num',
                                                                                        'i
nner').cache()
        # At last, calculate the diff_rank for every station in this cluster,
        # according to the viewing events given by this batch or by previous ones.
        calculate_diff_rank(clusters_stations_count_with_general)
        clusters_stations_count_with_general.unpersist()
```

In [0]:

```
import time

# For every batch-size number of viewing events it recieves,
# the following code performs a calculation of the diff_rank value for every station numb
er of every cluster.
streaming = streaming_df.writeStream\
                        .foreachBatch(handle_batch)\
                        .start()

# The code should stop running after 5 batches are done being analysed.
while (max_num_batches < 5):
    time.sleep(2)
streaming.stop()
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 0 , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|32645      |0.5698011049723757 |
|11164      |0.4200662983425415 |
|74796      |0.3600773480662983 |
|14902      |0.3335690607734807 |
|60179      |0.33286187845303883|
|10021      |0.32127071823204423|
|11221      |0.3115690607734807 |
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 0 , cluster 1 :
```

```
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|32645      |0.3660064205457463 |
|56905      |0.33888924558587474|
|19606      |0.33128410914927764|
|57708      |0.2641829855537721 |
|11661      |0.24702728731942214|
|58646      |0.24669662921348312|
|45507      |0.23463242375601923|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 0 , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|14771      |0.35614461846051326|
|99995      |0.34017794068643803|
|16615      |0.33624458513828714|
|61522      |0.312377874041986  |
|11865      |0.30247784071976   |
|16409      |0.26776674441852716|
|12131      |0.25298900366544486|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 0 , cluster 3 :
+-----------+------------------+
|station_num|diff_rank         |
+-----------+------------------+
|11118      |1.8721290322580644|
|58574      |1.4589354838709676|
|64490      |0.872483870967742 |
|10222      |0.8552580645161291|
|21883      |0.8248709677419355|
|59684      |0.6908709677419356|
|65732      |0.6580645161290322|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 0 , cluster 4 :
+-----------+------------------+
|station_num|diff_rank         |
+-----------+------------------+
|10171      |1.6332320217096334|
|14765      |1.1799077340569877|
|35513      |1.1168521031207599|
|70387      |0.9808521031207599|
|11561      |0.9311668928086838|
|12131      |0.9160651289009497|
|44940      |0.8973242876526457|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 0 , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|51529      |0.45766630316248635|
|10179      |0.4161788440567067 |
|16616      |0.4083075245365322 |
|64241      |0.38578189749182124|
|58515      |0.3524100327153763 |
|21214      |0.3462050163576882 |
|57394      |0.2912562704471101 |
+-----------+-------------------+
only showing top 7 rows
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 1 , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|32645      |0.3885284898690249 |
|10021      |0.26711138475316254|
|16615      |0.24711138475316252|
|11221      |0.22952804209112287|
|14902      |0.22888917496921546|
|60179      |0.21733370648158523|
|58646      |0.20533370648158522|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 1 , cluster 1 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|11661      |0.3253979067739122 |
|11713      |0.24612375230157968|
|60179      |0.24041147398003693|
|36069      |0.23157805988952418|
|10179      |0.22028248861323774|
|20288      |0.20772584552766743|
|57391      |0.19621513712569055|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 1 , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|10142      |0.47225876010781676|
|16615      |0.2559353099730458 |
|10918      |0.24847169811320757|
|16123      |0.24323719676549865|
|10510      |0.21461994609164423|
|10402      |0.20946630727762805|
|10518      |0.20785175202156336|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 1 , cluster 3 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|19320      |1.9479262493934983|
|11118      |1.5562047549733138|
|51529      |0.8561242115477925|
|58574      |0.7440436681222706|
|10222      |0.6678020378457059|
|10171      |0.5216443474041728|
|44714      |0.5107214944201844|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 1 , cluster 4 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|10171      |1.1746328469630136|
|35513      |1.0305812690872072|
|14765      |0.890976586359009 |
|11367      |0.8307828299966068|
|11207      |0.7821109602986087|
|11561      |0.7429172039362062|
```

```
|70387      |0.7199172039362061|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 1 , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|16616      |0.34517137265865255|
|58623      |0.33790970086664807|
|10145      |0.31447358121330715|
|31042      |0.3041713726586526 |
|64490      |0.2920841487279843 |
|34432      |0.28817137265865256|
|64241      |0.28621498462398665|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 2 , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|19320      |0.3802567550740879 |
|60179      |0.2742380774498816 |
|10021      |0.26415664300834263|
|16615      |0.2599907857053916 |
|14902      |0.2581723322126759 |
|11221      |0.22999701158012686|
|74796      |0.18903922301083287|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 2 , cluster 1 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.3302798136918885 |
|11661      |0.2640699534229721 |
|16374      |0.25845241233851857|
|36069      |0.24138412865805436|
|19606      |0.20705246506722913|
|11713      |0.19400228491080052|
|20288      |0.17971913173389578|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 2 , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|10142      |0.3153977340488965 |
|58646      |0.2958902802623733 |
|16615      |0.2909433512224211 |
|12131      |0.21570661896243293|
|19630      |0.2152271914132379 |
|10510      |0.20096243291592128|
|14767      |0.19200596302921885|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 2 , cluster 3 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|19320      |1.8553516763477023|
|11118      |1.2204877203709366|
|10222      |0.8710557423825537|
```

```
|51529      |0.8602977682665852|
|44714      |0.7322922653622745|
|10171      |0.6622064608172835|
|16288      |0.5934338122898196|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 2 , cluster 4 :
+-----------+------------------+
|station_num|diff_rank         |
+-----------+------------------+
|10171      |1.3620028005011422|
|70387      |0.9541049450954382|
|12131      |0.9329560026531064|
|35513      |0.8370240990493036|
|11367      |0.7549148795047534|
|11561      |0.7210240990493035|
|11207      |0.6546623922175547|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 2 , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|58623      |0.4155642733777449 |
|64490      |0.3758494942018258 |
|16616      |0.36714433752775727|
|10178      |0.3378302491981249 |
|34432      |0.30181100419442397|
|59684      |0.2570017271157167 |
|15591      |0.24825808043424624|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 3 , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|19320      |0.3701925859957697 |
|60179      |0.2846869642658356 |
|11221      |0.2689960480908382 |
|14902      |0.2376013024602026 |
|10021      |0.2251942558165424 |
|74796      |0.21006044751196706|
|16615      |0.2072138483802738 |
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 3 , cluster 1 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.3111390139013901 |
|36069      |0.28807900790079   |
|16374      |0.2526605160516051 |
|20288      |0.22253050305030503|
|11661      |0.20702700270027002|
|11713      |0.20656650665066517|
|19606      |0.18852300230023   |
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 3 , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
```

```
|16615      |0.35898002663115836|
|58646      |0.2548355525965379 |
|12131      |0.21739081225033274|
|10142      |0.20542543275632497|
|57394      |0.17907856191744342|
|11187      |0.1727796271637817 |
|11097      |0.16545739014647132|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 3 , cluster 3 :
+-----------+------------------+
|station_num|diff_rank         |
+-----------+------------------+
|19320      |1.3211378555798685|
|11118      |0.9979737417943108|
|10222      |0.8634190371991246|
|51529      |0.7536739606126914|
|44714      |0.7119824945295404|
|10171      |0.6992472647702408|
|16288      |0.5916914660831509|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 3 , cluster 4 :
+-----------+------------------+
|station_num|diff_rank         |
+-----------+------------------+
|10171      |1.2440050093926116|
|70387      |1.128089542892924 |
|35513      |0.9485763932373199|
|12131      |0.8887874139010643|
|11207      |0.7347808390732624|
|11367      |0.6497980588603631|
|14765      |0.6438850970569819|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 3 , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|64490      |0.3729514249613927 |
|16616      |0.3651001684683419 |
|58623      |0.35384170995367115|
|59684      |0.3245685104590762 |
|10057      |0.29137336796293695|
|10178      |0.27410016846834195|
|34432      |0.23970988347606342|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 4 , cluster 0 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|19320      |0.38432703484491115|
|60179      |0.3284562324774152 |
|14902      |0.2755071336389123 |
|10021      |0.2684919941257622 |
|74796      |0.25613208134929455|
|11221      |0.2208876685505763 |
|16615      |0.21609776155934324|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 4 , cluster 1 :
```

```
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|60179      |0.3299520567261591 |
|16374      |0.28519630957656794|
|36069      |0.2475885258450506 |
|11661      |0.2184192578864671 |
|20288      |0.20839041134523184|
|19606      |0.20658848555658516|
|16062      |0.18763466419564084|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 4 , cluster 2 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|16615      |0.3414402151983861 |
|12131      |0.25619071956960326|
|11187      |0.2186665770006725 |
|42642      |0.2126698049764627 |
|58646      |0.19861708137188971|
|58452      |0.19244707464694016|
|16123      |0.18412131809011428|
+-----------+-------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 4 , cluster 3 :
+-----------+------------------+
|station_num|diff_rank         |
+-----------+------------------+
|11118      |1.1786573164381184|
|19320      |1.1054478042006597|
|10171      |0.8935755597986459|
|10222      |0.8573372678354453|
|44714      |0.7786305849678875|
|51529      |0.7376154139906267|
|59684      |0.6636344384655442|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 4 , cluster 4 :
+-----------+------------------+
|station_num|diff_rank         |
+-----------+------------------+
|70387      |1.3340688845401174|
|10171      |1.3061424657534242|
|35513      |1.091170254403131 |
|12131      |0.9145925636007827|
|11207      |0.6540031311154598|
|44940      |0.6482880626223091|
|14765      |0.6178755381604695|
+-----------+------------------+
only showing top 7 rows

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 4 , cluster 5 :
+-----------+-------------------+
|station_num|diff_rank          |
+-----------+-------------------+
|16616      |0.4062964028776978 |
|59684      |0.37231654676258996|
|10057      |0.3655064748201438 |
|58623      |0.32111654676258994|
|64490      |0.3106553956834532 |
|10178      |0.2600503597122302 |
|74796      |0.25087050359712226|
+-----------+-------------------+
only showing top 7 rows
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
batch number 5 , cluster 0 :
```