DSE 230: Programming Assignment 4.1 - K-Means Cluster Analysis

---

Tasks:

- Work with `minute_weather.csv`
  - Use scikit-learn to perform k-means clustering (25%)
  - Explore parallelism with scikit-learn for k-means clustering (10%)
  - Explore parallelism with dask for k-means clustering (65%)
- Submission on Gradescope (2 files)
  - Completed notebook (.ipynb) or PDF with results under **PA4.1 Notebook**
    - Make sure that all expected outputs are present
  - An executable script (.py) exported from this notebook under **PA4.1**

Due date: Friday 5/28/2021 at 11:59 PM PST

Setup

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

Scikit-Learn (25%)

1.1 (5%) Load Data

- Load the "minute_weather.csv" into the Pandas dataframe
- Drop the two columns ["rowID", "hpwren_timestamp"] from the dataframe
- Print out the column names (features) from the output of the previous step

In [2]:
```
# Load the "minute_weather.csv" into the Pandas dataframe
df = pd.read_csv("minute_weather.csv")
```

In [3]:
```
# Drop the two columns ["rowID", "hpwren_timestamp"] from the dataframe
df = df.drop(["rowID", "hpwren_timestamp"], axis = 1)
```

In [4]:
```
# Print out the column names (features) from the output of the previous step
print(df.columns)
```

```
Index(['air_pressure', 'air_temp', 'avg_wind_direction', 'avg_wind_speed',
       'max_wind_direction', 'max_wind_speed', 'min_wind_direction',
       'min_wind_speed', 'rain_accumulation', 'rain_duration',
       'relative_humidity'],
      dtype='object')
```

In [5]:
```
# Drop null values
df = df.dropna()
```

1.2 (5%) Data preprocessing and normalization using sklearn

- Perform train and test split with 80% of the original dataset being the training dataset and 20% of the original dataset being the testing dataset.
  - Pass `random_state=seed` to `train_test_split` for reproducing results
- Print the number of samples from both train and test dataset, and the summary statistics of training dataset.
- Perform feature normalization on both the train dataset and the test dataset using StandardScaler from sklearn library. Only **train** data should be used for scaling
- Print out the mean and standard deviation along the feature columns of both the train and the test dataset.

(your output of the mean and std should be a vector of shape (1, number of features) make sure you clearly label your results)

In [6]:
```
seed=30
# Perform train and test split with 80% of the original dataset being the traini

train, test = train_test_split(df, test_size=0.2, random_state=seed)
```

In [7]:
```
# Print the number of samples from both train and test dataset, and the summary
print("Number of train samples:", len(train))
print("Number of test samples:", len(test))

print("\nSummary statistics of training dataset:\n", train.describe())
```

```
Number of train samples: 1269458
Number of test samples: 317365

Summary statistics of training dataset:
        air_pressure      air_temp  avg_wind_direction  avg_wind_speed  \
count  1.269458e+06  1.269458e+06        1.269458e+06    1.269458e+06
mean   9.168290e+02  6.185852e+01        1.619974e+02    2.774637e+00
std    3.050942e+00  1.183310e+01        9.516754e+01    2.061157e+00
min    9.050000e+02  3.164000e+01        0.000000e+00    0.000000e+00
25%    9.148000e+02  5.270000e+01        6.200000e+01    1.300000e+00
50%    9.167000e+02  6.242000e+01        1.820000e+02    2.200000e+00
75%    9.187000e+02  7.088000e+01        2.170000e+02    3.800000e+00
max    9.295000e+02  9.932000e+01        3.590000e+02    3.230000e+01

       max_wind_direction  max_wind_speed  min_wind_direction  min_wind_speed  \
count        1.269458e+06    1.269458e+06        1.269458e+06    1.269458e+06
mean         1.634496e+02    3.400203e+00        1.667738e+02    2.133283e+00
std          9.232584e+01    2.423860e+00        9.740912e+01    1.745411e+00
min          0.000000e+00    1.000000e-01        0.000000e+00    0.000000e+00
25%          6.800000e+01    1.600000e+00        7.700000e+01    8.000000e-01
50%          1.870000e+02    2.700000e+00        1.800000e+02    1.600000e+00
75%          2.230000e+02    4.600000e+00        2.120000e+02    3.000000e+00
max          3.590000e+02    3.600000e+01        3.590000e+02    3.200000e+01

       rain_accumulation  rain_duration  relative_humidity
count       1.269458e+06   1.269458e+06       1.269458e+06
```

```
mean        1.763524e-03   5.217258e-01       4.761155e+01
std         9.511593e-01   8.049277e+01       2.621206e+01
min         0.000000e+00   0.000000e+00       7.000000e-01
25%         0.000000e+00   0.000000e+00       2.470000e+01
50%         0.000000e+00   0.000000e+00       4.470000e+01
75%         0.000000e+00   0.000000e+00       6.810000e+01
max         6.550100e+02   6.330500e+04       9.300000e+01
```

In [8]:
```
# Perform feature normalization on both the train dataset and the test dataset u
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scalerModel = scaler.fit(train)

train_df = scalerModel.transform(train)
test_df = scalerModel.transform(test)
```

In [9]:
```
# Print out the mean and standard deviation along the feature columns of both th
# your output of the mean and std should be a vector of shape (1, number of feat

train_mean = pd.DataFrame(train.mean()).transpose()
print("Mean of the train dataset:\n")
train_mean
```

```
Mean of the train dataset:
```

Out[9]:

	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_wind_direction	max_wind_s
0	916.828996	61.85852	161.997358	2.774637	163.449599	3.40

In [10]:
```
test_mean = pd.DataFrame(test.mean()).transpose()
print("Mean of the test dataset:\n")
test_mean
```

```
Mean of the test dataset:
```

Out[10]:

	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_wind_direction	max_wind_
0	916.835256	61.842257	161.83767	2.772816	163.217025	3.3

In [11]:
```
train_std = pd.DataFrame(train.std()).transpose()
print("Standard deviation of the train dataset:\n")
train_std
```

```
Standard deviation of the train dataset:
```

Out[11]:

	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_wind_direction	max_wind_
0	3.050942	11.833097	95.16754	2.061157	92.325835	2.4

In [12]:
```
test_std = pd.DataFrame(test.std()).transpose()
print("Standard deviation of the test dataset:\n")
test_std
```

```
Standard deviation of the test dataset:
```

Out[12]:

	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_wind_direction	max_wind_s
0	3.053774	11.831213	95.370391	2.059166	92.532645	2.42

Build Clustering Model

1.4 (10%) KMeans Clustering model with sklearn

- Use the normalized training dataset to fit a K-means model with 9 clusters
  - Pass `random_state=seed` to `KMeans` for reproducing results
- Print out the cluster centers found by the model
- Print out the computational performance by adding "%%time" at the top of the cell

In [13]:

```
%%time
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=9, random_state=seed).fit(train_df)
print('cluster centers:\n', kmeans.cluster_centers_)
```

```
cluster centers:
 [[ 2.59868315e-01  1.75228455e-01 -1.13790121e+00 -4.92643104e-01
  -1.03013843e+00 -5.17690463e-01 -1.30453954e+00 -4.33255888e-01
  -1.79603979e-03 -4.65640190e-03 -3.20088310e-01]
 [-2.08209794e-01  5.52504796e-01  3.90729412e-01  5.42615774e-01
   4.92516732e-01  4.81142090e-01  2.29786096e-01  5.92013801e-01
  -1.83654932e-03 -6.09172449e-03 -1.78486455e-01]
 [ 1.24012683e+00 -2.12520363e-01 -1.15096346e+00  1.78204711e+00
  -1.05358787e+00  1.87346820e+00 -1.30046064e+00  1.58664606e+00
  -1.83395671e-03 -5.51395856e-03 -1.10716615e+00]
 [-2.78753032e-01 -1.02693213e+00  4.45427665e-01 -3.16633180e-01
   6.12278062e-01 -3.01395137e-01  2.10001190e-01 -3.31822591e-01
  -1.37065237e-03  2.77529357e-03  1.22513780e+00]
 [-1.51536685e-01 -5.76224663e-01 -3.32718879e-01  1.11201165e+00
  -1.96221054e-01  1.10009030e+00 -5.28087370e-01  1.10769536e+00
   6.48866201e+02  6.26290184e+02  5.88347083e-01]
 [ 2.98608026e-01  2.74314807e-01 -1.52556938e+00 -6.12719203e-01
  -1.20436255e+00 -5.38770930e-01  1.79835155e+00 -6.63748632e-01
  -1.83135656e-03 -5.48292590e-03 -3.55760639e-01]
 [-1.05033632e+00 -8.95957477e-01  4.26928281e-01  1.71158401e+00
   5.16922476e-01  1.64809233e+00  2.56493393e-01  1.73392403e+00
   3.79512399e-04  2.15989194e-02  9.74117263e-01]
 [ 8.95382148e-02  6.81535980e-01  7.12827142e-01 -6.59229047e-01
   9.23138741e-01 -6.40688385e-01  4.46912037e-01 -6.62672088e-01
  -1.84563402e-03 -6.23430697e-03 -5.69404064e-01]
 [ 2.36649596e-01  2.80028672e-01  1.89146265e+00 -6.49450373e-01
  -1.54778304e+00 -5.71150454e-01  1.46577055e+00 -7.07959152e-01
  -1.83849484e-03 -5.60747995e-03 -2.63493922e-01]]
CPU times: user 2min 15s, sys: 1min 40s, total: 3min 56s
Wall time: 21.7 s
```

Evaluate Model

1.5 (5%) Evaluate KMeans clustering model with sklearn

- Print out the inertia_ variable of the model, and explain what the number means in KMeans model
- Print out the within-cluster sum of squares (WSSE) on the train and test

Check documentations on KMeans at https://scikit-learn.org/stable/modules/clustering.html

In [14]:
```
print('inertia variable =', kmeans.inertia_, '\nIt means how far away the points
```

```
inertia variable = 4200082.403069052
It means how far away the points within a cluster are. Lower values are better a
nd zero is optimal.
```

In [15]:
```
print('WSSE_train =', -kmeans.score(train_df))
print('WSSE_test =', -kmeans.score(test_df))
```

```
WSSE_train = 4200082.403069052
WSSE_test = 1273945.1265774432
```

Parallelism with Scikit-Learn (10%)

2.1 (10%) Single machine parallelism using **all** the cores

- Fit the model with single-machine parallelism using scikit-learn and joblib (via `n_jobs` parameter)
  - Pass `random_state=seed` to `KMeans` for reproducing results
- Print out the WSSE on train and test
- Use %%time to print out the computational performance

Note that your model's parameters and seed setting should remain the same from the previous questions

In [16]:
```
%%time
from sklearn.cluster import KMeans
import joblib
kmeans_p = KMeans(n_clusters = 9, random_state = seed)

with joblib.Parallel(n_jobs=-1):
    kmeans_p.fit(train_df)

print('WSSE_train =', -kmeans_p.score(train_df))
print('WSSE_test =', -kmeans_p.score(test_df))
```

```
WSSE_train = 4200082.403069052
WSSE_test = 1273945.1265774432
CPU times: user 2min 25s, sys: 1min 36s, total: 4min 2s
Wall time: 22.4 s
```

Parallelism with Dask (65%)

Multi-machine parallelism using Dask's scalable k-means algorithm

Create and connect to client

3.1 (5%) Setup the Dask distributed client

- Create a Dask distributed client with 2 workers
- Print out the Dask client information

In [17]:
```
import joblib
from dask.distributed import Client

# Start and connect to local client
client = Client(n_workers=2)

# client = Client("scheduler-address:8786")  # connecting to remote cluster
```

In [18]:
```
client
```

Out[18]:

Client	Cluster
Scheduler: tcp://127.0.0.1:44141	**Workers:** 2
Dashboard: http://127.0.0.1:8787/status	**Cores:** 16
	Memory: 15.64 GiB

Load Data into Dask DataFrame

3.2 (5%) Load the data into Dask Dataframe

- Load the dataset into Dask Dataframe
- Use %%time to print out the loading efficiency of the operation

In [19]:
```
%%time
import dask.dataframe as dd
df_dd = dd.read_csv('minute_weather.csv')
df_dd.head()
```

```
CPU times: user 218 ms, sys: 52.7 ms, total: 271 ms
Wall time: 1.37 s
```

Out[19]:

	rowID	hpwren_timestamp	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_
0	0	2011-09-10 00:00:49	912.3	64.76	97.0	1.2	
1	1	2011-09-10 00:01:49	912.3	63.86	161.0	0.8	
2	2	2011-09-10 00:02:49	912.3	64.22	77.0	0.7	
3	3	2011-09-10 00:03:49	912.3	64.40	89.0	1.2	
4	4	2011-09-10 00:04:49	912.3	64.40	185.0	0.4	

Explore Data using Dask

3.3 (5%) Summary statistics

- Print out the shape of the dataframe
- Print the first 10 rows of the dask dataframe
- Print the summary statistics on all the features of the dask dataframe

In [20]:
```
print('Shape of the dataframe:\n', '(', df_dd.shape[0].compute(), ',', len(df_dd
```

```
Shape of the dataframe:
 ( 1587257 , 13 )
```

In [21]:
```
# Print the first 10 rows of the dask dataframe
df_dd.head(10)
```

Out[21]:

	rowID	hpwren_timestamp	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_
0	0	2011-09-10 00:00:49	912.3	64.76	97.0	1.2	
1	1	2011-09-10 00:01:49	912.3	63.86	161.0	0.8	
2	2	2011-09-10 00:02:49	912.3	64.22	77.0	0.7	
3	3	2011-09-10 00:03:49	912.3	64.40	89.0	1.2	
4	4	2011-09-10 00:04:49	912.3	64.40	185.0	0.4	
5	5	2011-09-10 00:05:49	912.3	63.50	76.0	2.5	
6	6	2011-09-10 00:06:49	912.3	62.78	79.0	2.4	
7	7	2011-09-10 00:07:49	912.3	62.42	86.0	2.0	
8	8	2011-09-10 00:08:49	912.3	62.24	105.0	1.4	
9	9	2011-09-10 00:09:49	912.3	62.24	93.0	0.4	

In [22]:
```
#Print the summary statistics on all the features of the dask dataframe
print(df_dd.describe().compute())
```

```
              rowID  air_pressure      air_temp  avg_wind_direction  \
count  1.587257e+06  1.587257e+06  1.587257e+06        1.586824e+06
mean   7.936280e+05  9.168301e+02  6.185144e+01        1.619654e+02
std    4.582018e+05  3.051593e+00  1.183362e+01        9.520812e+01
min    0.000000e+00  9.050000e+02  3.164000e+01        0.000000e+00
25%    3.966915e+05  9.150000e+02  5.810000e+01        9.900000e+01
50%    7.933840e+05  9.166000e+02  6.422000e+01        1.890000e+02
75%    1.185527e+06  9.185000e+02  7.988000e+01        2.170000e+02
max    1.587256e+06  9.295000e+02  9.950000e+01        3.590000e+02

       avg_wind_speed  max_wind_direction  max_wind_speed  min_wind_direction  \
```

count	1.586824e+06	1.586824e+06	1.586824e+06	1.586824e+06
mean	2.774272e+00	1.634030e+02	3.399813e+00	1.668264e+02
std	2.060758e+00	9.236723e+01	2.423167e+00	9.746275e+01
min	0.000000e+00	0.000000e+00	1.000000e-01	0.000000e+00
25%	1.300000e+00	9.700000e+01	1.700000e+00	1.020000e+02
50%	2.200000e+00	1.960000e+02	2.800000e+00	1.830000e+02
75%	3.900000e+00	2.260000e+02	4.700000e+00	2.120000e+02
max	3.230000e+01	3.590000e+02	3.600000e+01	3.590000e+02

	min_wind_speed	rain_accumulation	rain_duration	relative_humidity
count	1.586824e+06	1.587256e+06	1.587256e+06	1.587257e+06
mean	2.133130e+00	1.854836e-03	5.361460e-01	4.760837e+01
std	1.745345e+00	9.609716e-01	8.114766e+01	2.621454e+01
min	0.000000e+00	0.000000e+00	0.000000e+00	7.000000e-01
25%	9.000000e-01	0.000000e+00	0.000000e+00	3.000000e+01
50%	1.700000e+00	0.000000e+00	0.000000e+00	4.540000e+01
75%	3.100000e+00	0.000000e+00	0.000000e+00	7.000000e+01
max	3.200000e+01	6.550100e+02	6.330500e+04	9.300000e+01

Prepare Data using Dask

3.4 (5%) Data Preparation with Dask DataFrame

- Drop the ["rowID", "hpwren_timestamp"] two columns from the dataframe
- Perform 80/20 train and test split with `random_state=seed` (same as the previous task but in dask)
- Print out the number of samples in train and test dataset

Note that numbers of samples are slightly difference since Dask and scikit-learn are different implementations, and also due to round-off differences.

In [23]:
```
df_dd = df_dd.dropna()
```

In [24]:
```
# Drop the ["rowID", "hpwren_timestamp"] two columns from the dataframe
df_dd = df_dd.drop(["rowID", "hpwren_timestamp"], axis = 1)
```

In [25]:
```
# Perform 80/20 train and test split with random_state=seed (same as the previou
from dask_ml.model_selection import train_test_split
train_dd, test_dd = train_test_split(df_dd, test_size=0.2, random_state=seed, sh
```

In [26]:
```
# Print out the number of samples in train and test dataset
print('Number of samples in train dataset:\n', train_dd.shape[0].compute())
print('Number of samples in test dataset:\n', test_dd.shape[0].compute())
```

```
Number of samples in train dataset:
 1269940
Number of samples in test dataset:
 316883
```

3.5 (10%) Data preprocessing and normalization with Dask

- Perform feature normalization using the Dask library. Use only the **train** data for scaling.
- Print out the summary statistics of the transformed features in train and test dataframes

- Comments on your observation on the summary statistics of the transformed features in train and test dataframes

In [27]:
```
# Perform feature normalization using the Dask library. Use only the train data
from dask_ml.preprocessing import StandardScaler
scaler_1 = StandardScaler()
scalerModel_1 = scaler_1.fit(train_dd)
```

In [28]:
```
# Print out the summary statistics of the transformed features in train and test
train_df_1 = scalerModel_1.transform(train_dd)
test_df_1 = scalerModel_1.transform(test_dd)
print(train_df_1.describe().compute())
print(test_df_1.describe().compute())
```

```
       air_pressure      air_temp  avg_wind_direction  avg_wind_speed  \
count  1.269940e+06  1.269940e+06        1.269940e+06    1.269940e+06
mean  -5.306615e-14 -9.653990e-16       -2.873638e-17    2.802021e-16
std    1.000000e+00  1.000000e+00        1.000000e+00    1.000000e+00
min   -3.877920e+00 -2.553604e+00       -1.700512e+00   -1.346138e+00
25%   -6.001063e-01 -3.175365e-01       -6.817017e-01   -7.151437e-01
50%   -7.565603e-02  1.996491e-01        2.845927e-01   -2.783013e-01
75%    5.471286e-01  1.507824e+00        5.786823e-01    5.468455e-01
max    4.152724e+00  3.165861e+00        2.070137e+00    1.433165e+01

       max_wind_direction  max_wind_speed  min_wind_direction  min_wind_speed  \
count        1.269940e+06    1.269940e+06        1.269940e+06    1.269940e+06
mean         9.650410e-17    1.478894e-16        2.264892e-17   -1.555882e-16
std          1.000000e+00    1.000000e+00        1.000000e+00    1.000000e+00
min         -1.768636e+00   -1.361679e+00       -1.711532e+00   -1.222212e+00
25%         -7.318650e-01   -7.012527e-01       -6.651105e-01   -7.063686e-01
50%          3.536318e-01   -2.472095e-01        1.658710e-01   -2.478411e-01
75%          6.784687e-01    5.370470e-01        4.633830e-01    5.545820e-01
max          2.118579e+00    1.234217e+01        1.971461e+00    1.711889e+01

       rain_accumulation  rain_duration  relative_humidity
count       1.269940e+06   1.269940e+06       1.269940e+06
mean        7.553370e-20  -1.555435e-18       4.603192e-16
std         1.000000e+00   1.000000e+00       1.000000e+00
min        -1.643600e-03  -7.504024e-03      -1.789382e+00
25%        -1.643600e-03  -7.504024e-03      -6.610106e-01
50%        -1.643600e-03  -7.504024e-03      -8.394808e-02
75%        -1.643600e-03  -7.504024e-03       8.546130e-01
max         8.559063e+02   1.005067e+03       1.732130e+00
        air_pressure       air_temp  avg_wind_direction  avg_wind_speed  \
count  316883.000000  316883.000000       316883.000000   316883.000000
mean       -0.000926      -0.000944            0.003220        0.002202
std         1.001140       0.999764            0.999946        1.001258
min        -3.877920      -2.523181           -1.700512       -1.346138
25%        -0.600106      -0.302325           -0.576670       -0.715144
50%        -0.075656       0.214860            0.274090       -0.278301
75%         0.547129       1.538247            0.578682        0.546846
max         4.152724       3.181072            2.070137       14.234575

       max_wind_direction  max_wind_speed  min_wind_direction  min_wind_speed  \
count       316883.000000   316883.000000       316883.000000   316883.000000
mean             0.003384        0.001872           -0.000271        0.002063
std              1.000712        1.001013            0.999366        1.001805
min             -1.768636       -1.361679           -1.711532       -1.222212
25%             -0.629000       -0.701253           -0.654851       -0.706369
50%              0.353632       -0.247209            0.165871       -0.247841
75%              0.700125        0.537047            0.463383        0.554582
```

```
max              2.118579       13.456640            1.971461       16.889625

       rain_accumulation  rain_duration  relative_humidity
count      316883.000000  316883.000000      316883.000000
mean            0.003909       0.005042          -0.000705
std             1.972456       2.075231           0.999625
min            -0.001644      -0.007504          -1.789382
25%            -0.001644      -0.007504          -0.728732
50%            -0.001644      -0.007504          -0.083948
75%            -0.001644      -0.007504           0.850798
max           836.488625     897.899657           1.732130
```

In [29]:

```
# Comments on your observation on the summary statistics of the transformed feat
print('By observing on the summary statistics of the transformed features in tra
```

```
By observing on the summary statistics of the transformed features in train and
test dataframe, we can tell that basically both train and test dataframes have s
imilar central tendency, dispersion and distribution.
```

Build Dask K-Means Model

3.6 (15%) KMeans clustering model with dask

- Fit KMeans model with Dask cluster library with the transformed Dask dataframe, you should set cluster number `n_clusters` and `random_state` as the same number as previous task
- Print out the computational performance using %%time

Note that Dask's K-Means estimator uses kmeans|| as the default algorithm. To compare to scikit-learn's implementation of k-means, use k-means++ instead.

In [30]:

```
!pip3 install --upgrade dask
```

```
Requirement already up-to-date: dask in /usr/local/lib/python3.8/dist-packages
(2021.5.0)
Requirement already satisfied, skipping upgrade: fsspec>=0.6.0 in /usr/local/li
b/python3.8/dist-packages (from dask) (0.8.7)
Requirement already satisfied, skipping upgrade: partd>=0.3.10 in /usr/local/li
b/python3.8/dist-packages (from dask) (1.1.0)
Requirement already satisfied, skipping upgrade: cloudpickle>=1.1.1 in /usr/loca
l/lib/python3.8/dist-packages (from dask) (1.6.0)
Requirement already satisfied, skipping upgrade: toolz>=0.8.2 in /usr/local/lib/
python3.8/dist-packages (from dask) (0.11.1)
Requirement already satisfied, skipping upgrade: pyyaml in /usr/local/lib/python
3.8/dist-packages (from dask) (5.4.1)
Requirement already satisfied, skipping upgrade: locket in /usr/local/lib/python
3.8/dist-packages (from partd>=0.3.10->dask) (0.2.1)
```

In [31]:

```
!pip3 install --upgrade dask[distributed]
```

```
Requirement already up-to-date: dask[distributed] in /usr/local/lib/python3.8/di
st-packages (2021.5.0)
Requirement already satisfied, skipping upgrade: cloudpickle>=1.1.1 in /usr/loca
l/lib/python3.8/dist-packages (from dask[distributed]) (1.6.0)
Requirement already satisfied, skipping upgrade: fsspec>=0.6.0 in /usr/local/li
b/python3.8/dist-packages (from dask[distributed]) (0.8.7)
Requirement already satisfied, skipping upgrade: toolz>=0.8.2 in /usr/local/lib/
python3.8/dist-packages (from dask[distributed]) (0.11.1)
```

```
Requirement already satisfied, skipping upgrade: pyyaml in /usr/local/lib/python
3.8/dist-packages (from dask[distributed]) (5.4.1)
Requirement already satisfied, skipping upgrade: partd>=0.3.10 in /usr/local/li
b/python3.8/dist-packages (from dask[distributed]) (1.1.0)
Requirement already satisfied, skipping upgrade: distributed==2021.05.0; extra =
= "distributed" in /usr/local/lib/python3.8/dist-packages (from dask[distribute
d]) (2021.5.0)
Requirement already satisfied, skipping upgrade: locket in /usr/local/lib/python
3.8/dist-packages (from partd>=0.3.10->dask[distributed]) (0.2.1)
Requirement already satisfied, skipping upgrade: tornado>=6.0.3; python_version
>= "3.8" in /usr/local/lib/python3.8/dist-packages (from distributed==2021.05.0;
extra == "distributed"->dask[distributed]) (6.1)
Requirement already satisfied, skipping upgrade: zict>=0.1.3 in /usr/local/lib/p
ython3.8/dist-packages (from distributed==2021.05.0; extra == "distributed"->das
k[distributed]) (2.0.0)
Requirement already satisfied, skipping upgrade: setuptools in /usr/lib/python3/
dist-packages (from distributed==2021.05.0; extra == "distributed"->dask[distrib
uted]) (45.2.0)
Requirement already satisfied, skipping upgrade: click>=6.6 in /usr/local/lib/py
thon3.8/dist-packages (from distributed==2021.05.0; extra == "distributed"->dask
[distributed]) (7.1.2)
Requirement already satisfied, skipping upgrade: psutil>=5.0 in /usr/local/lib/p
ython3.8/dist-packages (from distributed==2021.05.0; extra == "distributed"->das
k[distributed]) (5.8.0)
Requirement already satisfied, skipping upgrade: tblib>=1.6.0 in /usr/local/lib/
python3.8/dist-packages (from distributed==2021.05.0; extra == "distributed"->da
sk[distributed]) (1.7.0)
Requirement already satisfied, skipping upgrade: sortedcontainers!=2.0.0,!=2.0.1
in /usr/local/lib/python3.8/dist-packages (from distributed==2021.05.0; extra ==
"distributed"->dask[distributed]) (2.3.0)
Requirement already satisfied, skipping upgrade: msgpack>=0.6.0 in /usr/local/li
b/python3.8/dist-packages (from distributed==2021.05.0; extra == "distributed"->
dask[distributed]) (1.0.2)
Requirement already satisfied, skipping upgrade: heapdict in /usr/local/lib/pyth
on3.8/dist-packages (from zict>=0.1.3->distributed==2021.05.0; extra == "distrib
uted"->dask[distributed]) (1.0.1)
```

In [32]:

```
%%time

# Fit KMeans model with Dask cluster library with the transformed Dask dataframe
from dask_ml.cluster import KMeans
km = KMeans(n_clusters=9, random_state = seed, init = 'k-means++')

with joblib.parallel_backend("dask"):
    km.fit(train_df_1)
```

```
CPU times: user 4.93 s, sys: 9.88 s, total: 14.8 s
Wall time: 39.4 s
```

Evaluate Dask K-Means Model

3.7 (5%) Analyse hyperparameters

- Print out the inertia_ of KMeans model
- Print out the computational efficiency with %%time
- Double check if the dataframes and hyperparameters are the same for both scikit-learn K-Means model and Dask K-Means model. Is the inertia_ you printed different from your answer from the previous question? Explain your observation.

3.8 (10%) Dask K-Means estimator does not have a score() method. As an easy fix, we can

instantiate a scikit-learn K-Means estimator with the fitted Dask model (i.e., just copy the cluster centers over) to use the scikit-learn K-Means score method.

- Print out the cluster centers found by the Dask KMeans model
- Instantiate a scikit-learn KMeans estimator and assign the cluster centers with the one from Dask model
- Print out the WSSE on train and test using score method. (Note that WSSE is the within-cluster sum of **square** error)

In [33]:
```
%%time

#Print out the inertia_ of KMeans model
print('inertia=', km.inertia_)
print('Yes, it is different from the previous one. This is smaller than the prev
```

```
inertia= 2141245.023499641
Yes, it is different from the previous one. This is smaller than the previous on
e.
Scikit-learn uses joblib for single-machine parallelism. This lets you train mos
t estimators using all the cores of your laptop or workstation.
Dask registers a joblib backend. This lets you train those estimators using all
the cores of your cluster , by changing one line of code.
This is most useful for training large models on medium-sized datasets.
CPU times: user 512 µs, sys: 0 ns, total: 512 µs
Wall time: 721 µs
```

In [34]:
```
# Print out the cluster centers found by the Dask KMeans model
print('Cluster centers:\n', km.cluster_centers_)
```

```
Cluster centers:
 [[ 8.67568330e-02  6.80799368e-01  7.09207336e-01 -6.56061337e-01
   9.19466658e-01 -6.37833792e-01  4.42295818e-01 -6.59366884e-01
  -1.63246209e-03 -7.18240568e-03 -5.69294272e-01]
 [-1.04989034e+00 -8.99434463e-01  4.27912491e-01  1.71149918e+00
   5.18963271e-01  1.64852440e+00  2.55550337e-01  1.73344578e+00
   1.12121305e-03  2.84005042e-02  9.77898545e-01]
 [-3.37881143e-01  1.54015100e-01  5.15663141e-01 -6.18067587e-01
   7.32608219e-01 -6.80614334e-01  2.47943290e-01 -5.63078801e-01
   7.94419104e+02  7.54183706e+02 -4.57951893e-02]
 [ 2.61146200e-01  1.76164500e-01 -1.13673663e+00 -4.92097495e-01
  -1.02901165e+00 -5.17393952e-01 -1.30437347e+00 -4.32514216e-01
  -1.58615790e-03 -5.38087901e-03 -3.18825525e-01]
 [ 2.38372321e-01  2.80952553e-01  1.89120133e+00 -6.47753138e-01
  -1.54644191e+00 -5.69341762e-01  1.46507515e+00 -7.06269983e-01
  -1.62819605e-03 -6.47615522e-03 -2.64449578e-01]
 [-2.79921116e-01 -1.02759405e+00  4.47569683e-01 -3.17779495e-01
   6.14614812e-01 -3.02381947e-01  2.10131957e-01 -3.32964892e-01
  -1.04689228e-03  4.25760841e-03  1.22609311e+00]
 [ 1.23968516e+00 -2.14826985e-01 -1.14965623e+00  1.78324294e+00
  -1.05236090e+00  1.87473199e+00 -1.30135878e+00  1.58771183e+00
  -1.61684579e-03 -6.25570678e-03 -1.10630441e+00]
 [-2.09938975e-01  5.49448707e-01  3.92030059e-01  5.50787547e-01
   4.93904299e-01  4.88697286e-01  2.29561009e-01  6.00672580e-01
  -1.62254800e-03 -7.03422406e-03 -1.74526654e-01]
 [ 2.96464307e-01  2.71964329e-01 -1.52395295e+00 -6.12085182e-01
  -1.20317049e+00 -5.38264637e-01  1.79654726e+00 -6.63457731e-01
  -1.61746882e-03 -6.24235367e-03 -3.55792842e-01]]
```

In [35]:
```
# Instantiate a scikit-learn KMeans estimator and assign the cluster centers wit
```

```
from sklearn.cluster import KMeans
km_2 = KMeans(n_clusters=9, random_state = seed)
km_2.fit(km.cluster_centers_)
```

Out[35]:
```
KMeans(n_clusters=9, random_state=30)
```

In [36]:
```
# Print out the WSSE on train and test using score method. (Note that WSSE is th
print('WSSE_train =', -km_2.score(train_df_1))
print('WSSE_test =', -km_2.score(test_df_1))
```

```
WSSE_train = 4242460.296426644
WSSE_test = 1431340.0547391435
```

In [37]:
```
# Another way is to just assign dask_model.cluster_centers_ to sklearn_model.clu
kmeans.cluster_centers_ = km.cluster_centers_
print('WSSE_train =', -kmeans.score(train_df_1))
print('WSSE_test =', -kmeans.score(test_df_1))
```

```
WSSE_train = 4242460.296426644
WSSE_test = 1431340.0547391435
```

Stop the Dask Client

3.9 (5%) Stop the dask client

In [38]:
```
client.shutdown()
```

```
distributed.client - ERROR - Failed to reconnect to scheduler after 10.00 second
s, closing client
_GatheringFuture exception was never retrieved
future: <_GatheringFuture finished exception=CancelledError()>
asyncio.exceptions.CancelledError
```