CIS 660 DATA MINING

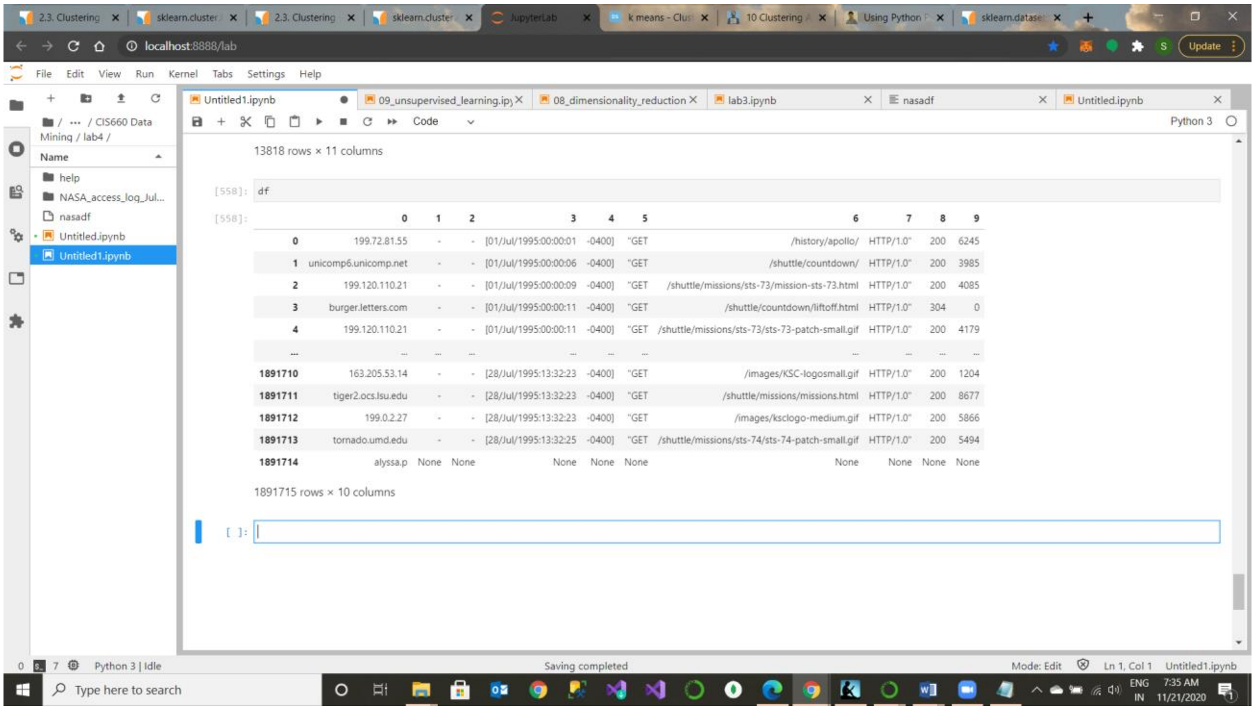
Lab 4 :Clustering with NASA Webserver Log Data and Feature Selection with PCA tools and also working with different values of K (Extra Credit)

Fathima Syeda

CSU ID-2790024

Part 1: Feature Selection, Cleaning, and Preprocessing to Construct an Input from Data Source:

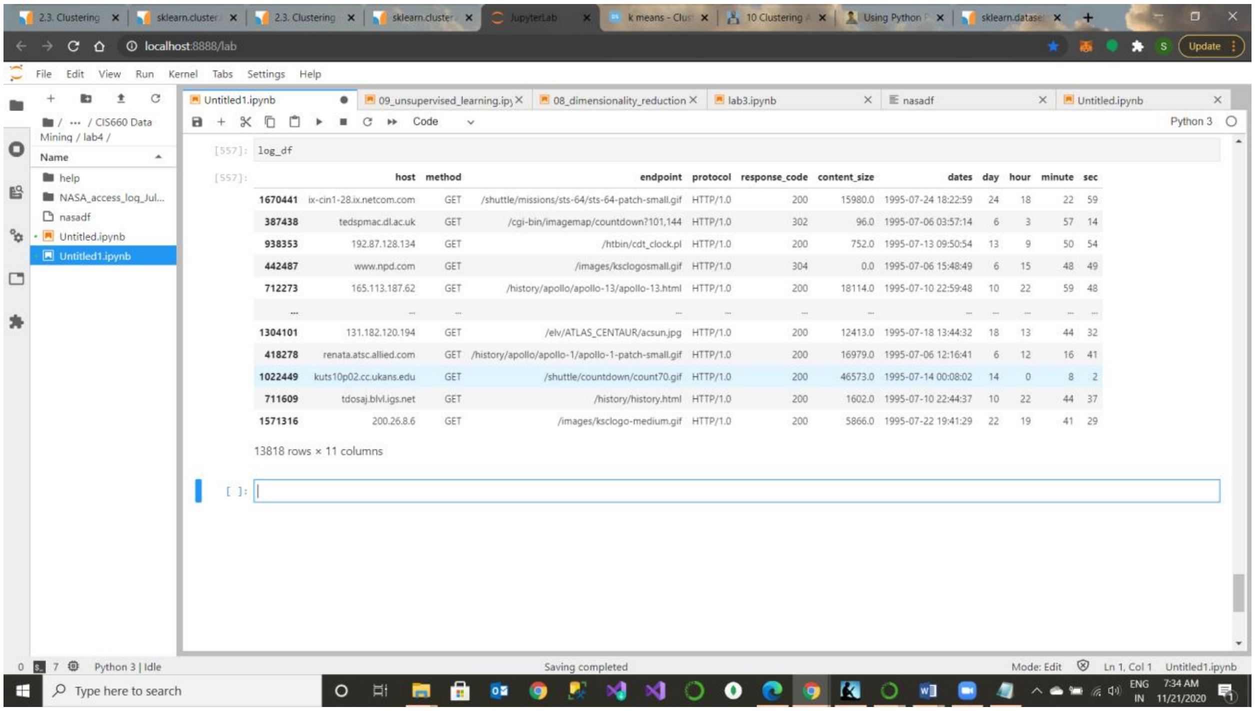
The dataset used in this NASA_access_log_Jul95 server log dataset. This consists of 1800000 logs but of this we a random sample of 14000 logs are chosen .



The dataset consists of 9 columns, namely ---host , 'client_identd' , 'user_id' , 'date_time' 'method', 'endpoint' ,'protocol', 'response_code' , 'content_size' .

- The client_identd and user_id columns conatin null values and hence those columns are dropped .
- The date –time column is transformed into a pandas data time object and additional hour, min, sec columns are found form it and added to the df.
- Certain rows have missing content_size values and irregular response codes, those rows are eliminated.

After cleaning the dataset looks as follows:



```
[557]: log_df
```

	host	method	endpoint	protocol	response_code	content_size	dates	day	hour	minute	sec
1670441	ix-cin1-28.ix.netcom.com	GET	/shuttle/missions/sts-64/sts-64-patch-small.gif	HTTP/1.0	200	15980.0	1995-07-24 18:22:59	24	18	22	59
387438	tedspmac.dl.ac.uk	GET	/cgi-bin/imagemap/countdown?101,144	HTTP/1.0	302	96.0	1995-07-06 03:57:14	6	3	57	14
938353	192.87.128.134	GET	/htbin/cdt_clock.pl	HTTP/1.0	200	752.0	1995-07-13 09:50:54	13	9	50	54
442487	www.npd.com	GET	/images/ksclogosmall.gif	HTTP/1.0	304	0.0	1995-07-06 15:48:49	6	15	48	49
712273	165.113.187.62	GET	/history/apollo/apollo-13/apollo-13.html	HTTP/1.0	200	18114.0	1995-07-10 22:59:48	10	22	59	48
...	...	...	...	...	...	...	...	...	...	...	...
1304101	131.182.120.194	GET	/elv/ATLAS_CENTAUR/acsun.jpg	HTTP/1.0	200	12413.0	1995-07-18 13:44:32	18	13	44	32
418278	renata.atsc.allied.com	GET	/history/apollo/apollo-1/apollo-1-patch-small.gif	HTTP/1.0	200	16979.0	1995-07-06 12:16:41	6	12	16	41
1022449	kuts10p02.cc.ukans.edu	GET	/shuttle/countdown/count70.gif	HTTP/1.0	200	46573.0	1995-07-14 00:08:02	14	0	8	2
711609	tdosaj.blvl.igs.net	GET	/history/history.html	HTTP/1.0	200	1602.0	1995-07-10 22:44:37	10	22	44	37
1571316	200.26.8.6	GET	/images/ksclogo-medium.gif	HTTP/1.0	200	5866.0	1995-07-22 19:41:29	22	19	41	29

13818 rows × 11 columns

The cleaned data is now ready for transformation – normalization for numerical data and One Hot Encode from categorical data , we also discretize continuous attributes.

Host and endpoint have too many unique values and hence are not considered as features to be fed to the clustering algorithm,

The date-time column is removed too as information has been extracted from it into hour , min, sec ,day

Continuous values: Day , Hour, Min ,sec

These are discretized into bins of 5

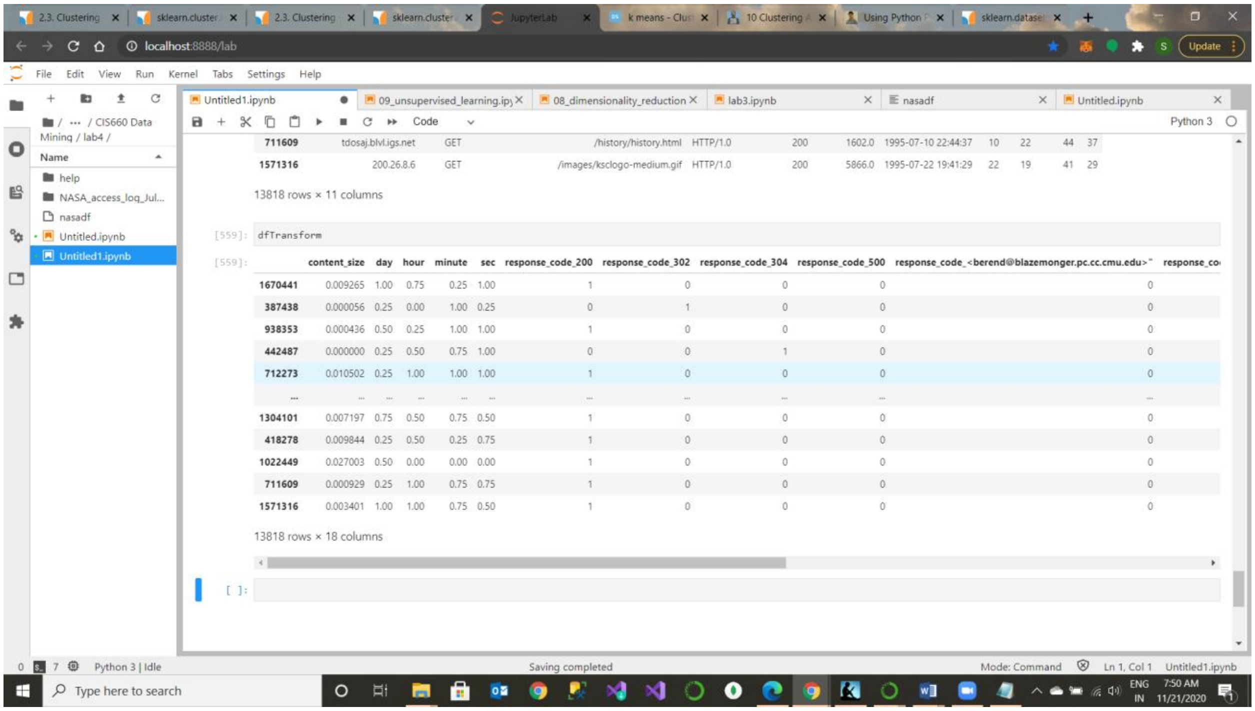
Numerical Values: Day , Hour, Min ,sec, content_size

These values are normalized using the MinMAx Scaler.

Categorical values: method , protocol ,response_code

These values are one hot encoded

After data pre-processing the final training data looks as follows:

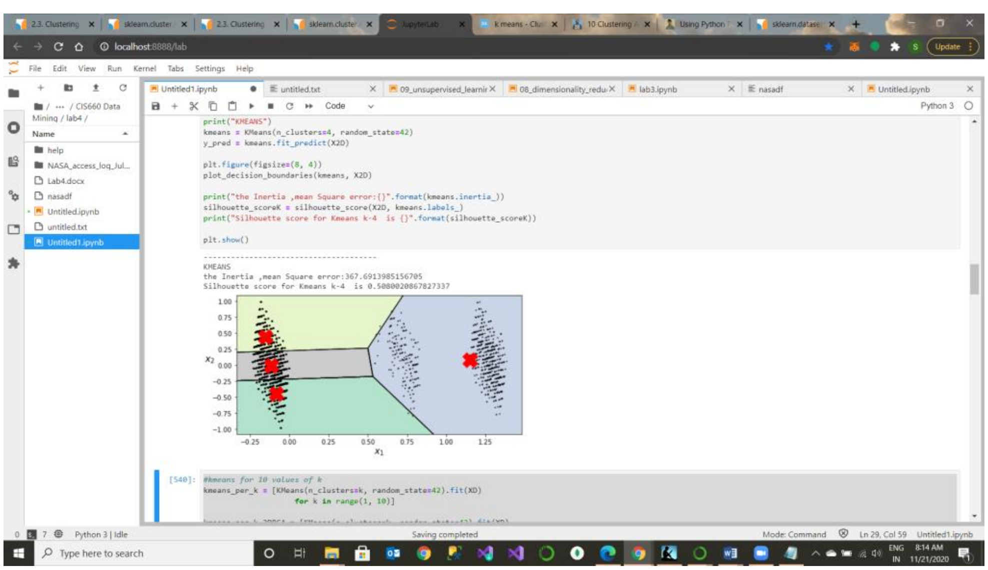


PART-2_1 Data Analytic Experiment with Two different Clustering of Your Choice.

K-Means clustering Algorithm

KMeans plot for k=4

the Inertia ,mean Square error:367.6913985156705
Silhouette score for Kmeans k-4 is 0.5080020867827337



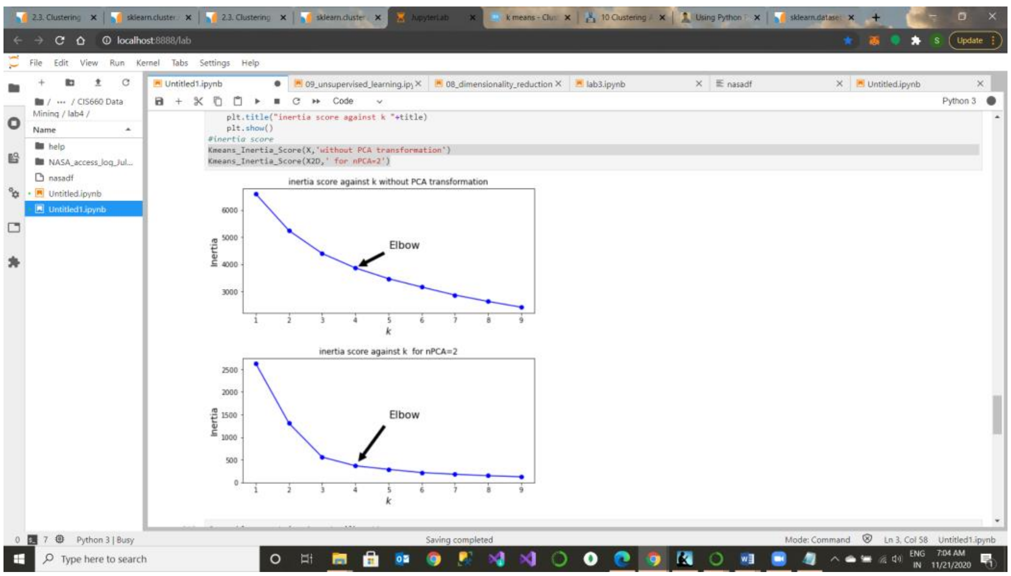
EXTRA CREDIT

We consider the different values of k from 1 to 10 to train the kmeans algorithm on.

The silhouette score and the Mean Square Score for each of the k values is found and compared on a graph below.

The above is done on the training dataset as it is and also by applying PCA reduction tools with nPCA=2

Comparison of Inertia(MSE) scores for k=1 to 10 with PCA and non-PCA transformed dataset:

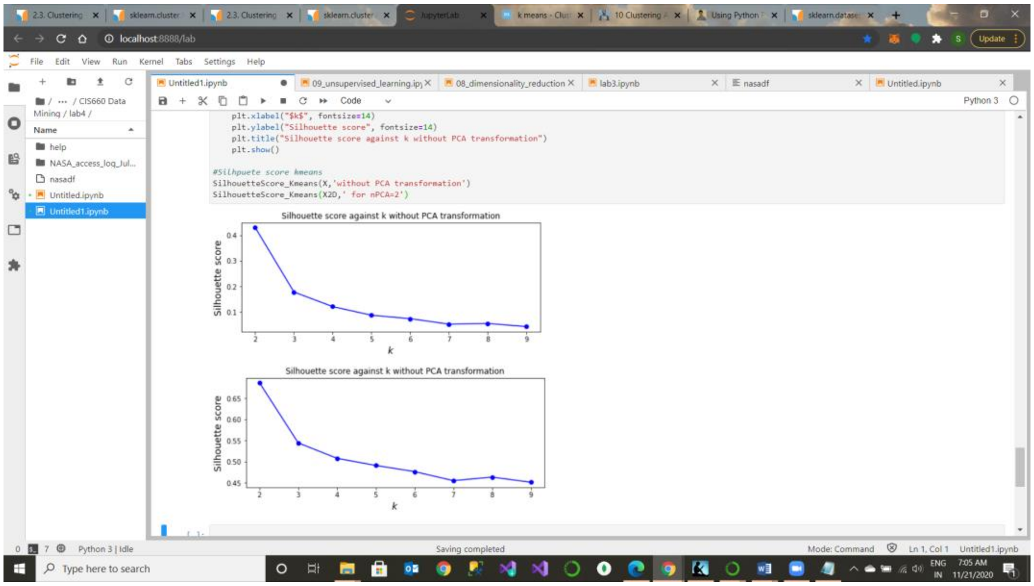


As you can see, there is an elbow at **k=4**, which means that less clusters than that would be bad, and more clusters would not help much and might cut clusters in half. So k =4 is a pretty good choice.

we cannot simply take the value of k that minimizes the inertia, since it keeps getting lower as we increase k. Indeed, the more clusters there are, the closer each instance will be to its closest centroid, and therefore the lower the inertia will be.

Comparison of Silhouette scores for k=1 to 10 with PCA and non-PCA transformed dataset:

As you can see, this visualization is much richer than the previous one: in particular, although it confirms that k=4 is a very good choice, but it also underlines the fact that k=5 is quite good as well.



Thus k=4 ,5 is the best choice for the given dataset when trained on KMEANS clustering algorithm.

DBSCAN

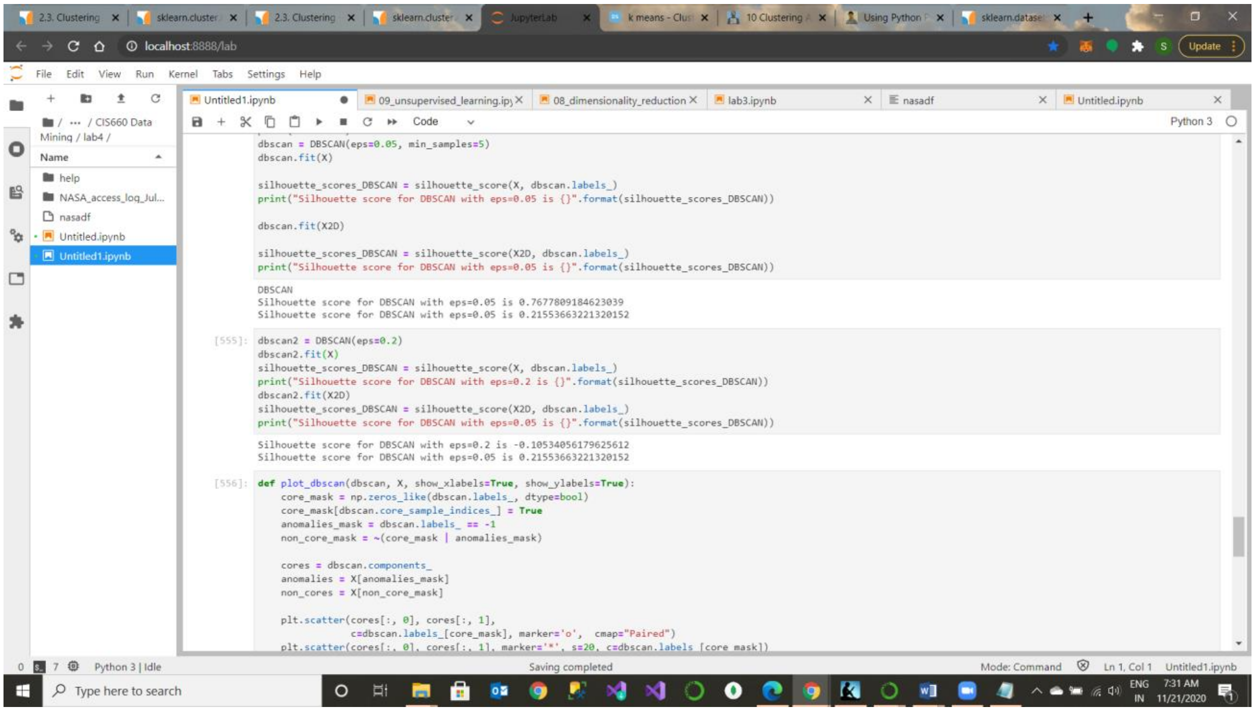
We train two DBSCAN clustering algorithms one with ep=0.05 and the other with ep=0.2

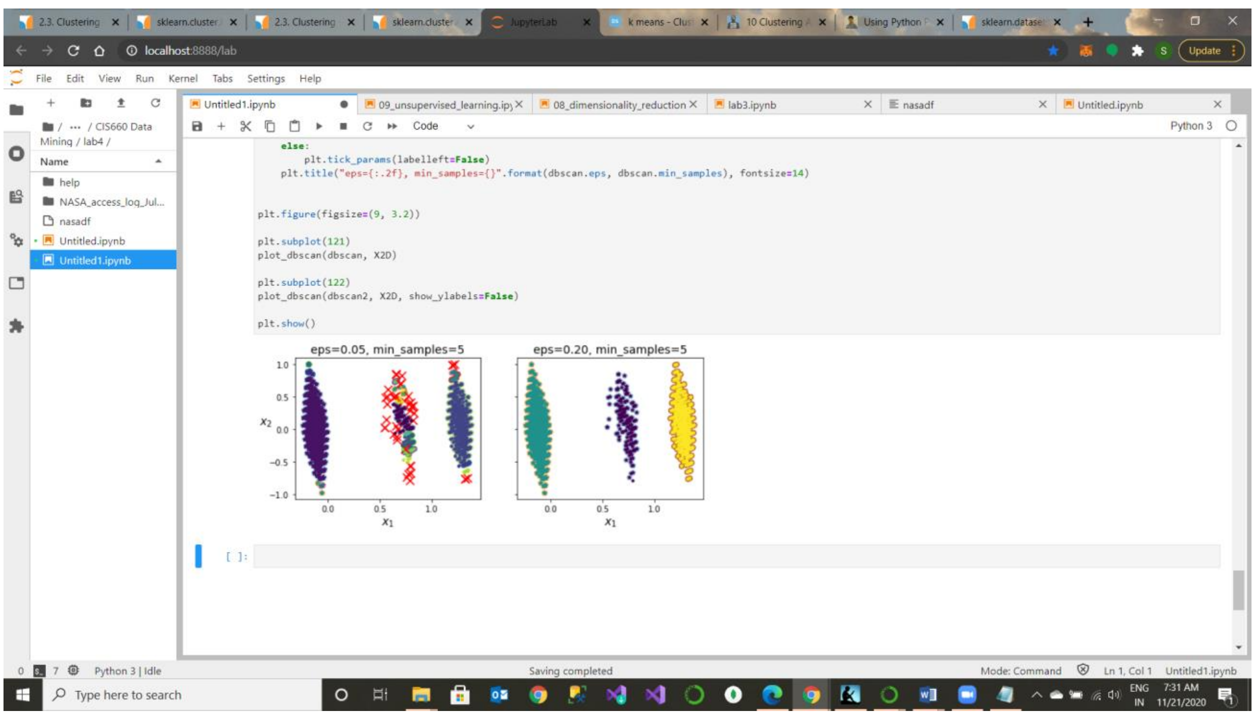
Silhouette score for DBSCAN with eps=0.05 is 0.7677809184623039
Silhouette score for DBSCAN with eps=0.05 is 0.21553663221320152

And

Silhouette score for DBSCAN with eps=0.2 is -0.10534056179625612
Silhouette score for DBSCAN with eps=0.2 is 0.21553663221320152

Thus we understand that DBSCAN with eps=0.05 is better than eps=0.2





RESULTS:

- Kmeans is a least-squares optimization, whereas DBSCAN finds density-connected regions.

- Our experiment works best with DBSCAN as the data consists of dense regions seprated by sparse regions and DBSCAN is good at clustering these kinds of data. Means works to minimize the least squares
- From our experiment we get that DBSCAN has better accuracy and performs better than Kmeans algorithm
- The silhouette score of DBSCAN is 0.75 whereas that of KMeans k=4 is 0.5
- We compared the inertias and silhouette scores of various k values and found that the best value of k =4 as there is an elbow at **k=4**, which means that less clusters than that would be bad, and more clusters would not help much and might cut clusters in half.
- But after finding the silhouette score we get that both k=4,5 are good options.
- We also see that KMeans performs better when PCA dimensionality reduction is done on the data before feeding it to KMeans algorithm. For our dataset nPCA=2 yields the best results.

Source Code:

```
from sklearn.cluster import KMeans

from sklearn.cluster import DBSCAN

import numpy as np

import re

import pandas as pd

import datetime

from sklearn import preprocessing

from sklearn.preprocessing import  MinMaxScaler

import matplotlib.pyplot as plt

from sklearn.metrics import silhouette_score


N_file=open('NASA_access_log_Jul95/access_log_Jul95',encoding='latin1')


Lines=N_file.readlines()


df=pd.DataFrame([line.split() for line in Lines])


df.drop(columns=[i for i in range(10,56)],axis=1,inplace=True)
```

```
df



log_df=df.sample(14000)
log_df=log_df.dropna()

def parse_apache_time(s):
   month_map = {'Jan': 1, 'Feb': 2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6, 'Jul':7,
   'Aug':8,  'Sep': 9, 'Oct':10, 'Nov': 11, 'Dec': 12}

   """ Convert Apache time format into a Python datetime object
   Args:
      s (str): date and time in Apache time format
   Returns:
      datetime: datetime object (ignore timezone for now)
   """
   return datetime.datetime(int(s[7:11]),
                       month_map[s[3:6]],
                       int(s[0:2]),
                       int(s[12:14]),
                       int(s[15:17]),
                       int(s[18:20]))



#removing rows with '' in prtotocol
log_df=log_df[log_df[8]!='']
```

```
#modifying the date columns
M=[l.replace('[','') for l in log_df[3]]
dates=[parse_apache_time(s) for s in M]
log_df['dates']=dates
log_df.drop(columns=[1,2,3,4],inplace=True)

#renaming columns
log_df.rename(columns={0:'host', 5: 'method',  6: 'endpoint',
       7:'protocol',
       8:'response_code',
       9:'content_size'},inplace=True)

log_df['method']=[l.replace('"','') for l in log_df['method']]
#log_df['content_size']=[l.replace('\n','') for l in log_df['content_size']]
#removing empty protocol rows
log_df['protocol']=[l.replace('"','') for l in log_df['protocol']]



#getting the dates
log_df['dates'] = pd.to_datetime(log_df['dates'],
 format = '%Y-%m-%dT%H:%M:%SZ',
 errors = 'coerce')

#log_df['year'] = log_df['dates'].dt.year
#log_df['month'] = log_df['dates'].dt.month
log_df['day'] = log_df['dates'].dt.day
log_df['hour'] = log_df['dates'].dt.hour
log_df['minute'] = log_df['dates'].dt.minute
```

```
log_df['sec'] = log_df['dates'].dt.second



log_df=log_df[log_df.content_size!='-']
log_df['content_size']=log_df['content_size'].astype('float64',copy=False)

#Data preprocessing
dfTransform=log_df[['method','protocol','response_code','content_size','day','hour','minute','sec']].copy
()

#normalization and discretization



minmax=MinMaxScaler()
dfTransform['day']=pd.qcut(log_df['day'], q=5,labels=[0,1,2,3,4])
dfTransform['day']=minmax.fit_transform(dfTransform[['day']])

dfTransform['hour']=pd.qcut(log_df['hour'], q=5,labels=[0,1,2,3,4])
dfTransform['hour']=minmax.fit_transform(dfTransform[['hour']])

dfTransform['minute']=pd.qcut(log_df['minute'], q=5,labels=[0,1,2,3,4])
dfTransform['minute']=minmax.fit_transform(dfTransform[['minute']])

dfTransform['sec']=pd.qcut(log_df['sec'], q=5,labels=[0,1,2,3,4])
dfTransform['sec']=minmax.fit_transform(dfTransform[['sec']])

dfTransform['content_size']=minmax.fit_transform(dfTransform[['content_size']])
```

```
#OneHotEncoding
OHE=pd.get_dummies(log_df['response_code'],prefix='response_code')
dfTransform=dfTransform.drop('response_code',axis=1)
dfTransform=dfTransform.join(OHE)



OHE=pd.get_dummies(log_df['method'],prefix='method')
dfTransform=dfTransform.drop('method',axis=1)
dfTransform=dfTransform.join(OHE)



OHE=pd.get_dummies(log_df['protocol'],prefix='protocol')
dfTransform=dfTransform.drop('protocol',axis=1)
dfTransform=dfTransform.join(OHE)

#splitting into train and validation
df_valid = dfTransform.sample(frac = 0.3, random_state = 42)
X = dfTransform.drop(df_valid.index)

from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)

pca = PCA(n_components = 5)
X5D = pca.fit_transform(X)

#plotting the data
def plot_data(X):
```

```
    plt.plot(X[:, 0], X[:, 1], 'k.', markersize=2)


def plot_centroids(centroids, weights=None, circle_color='w', cross_color='r'):
    if weights is not None:
        centroids = centroids[weights > weights.max() / 10]
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='o', s=30, linewidths=4,
                color=circle_color, zorder=10, alpha=0.9)
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=50, linewidths=25,
                color=cross_color, zorder=11, alpha=1)


def plot_decision_boundaries(clusterer, X, resolution=1000, show_centroids=True,
                             show_xlabels=True, show_ylabels=True):
    mins = X.min(axis=0) - 0.1
    maxs = X.max(axis=0) + 0.1
    xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                         np.linspace(mins[1], maxs[1], resolution))
    Z = clusterer.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                cmap="Pastel2")
    plt.contour(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                linewidths=1, colors='k')
    plot_data(X)
    if show_centroids:
        plot_centroids(clusterer.cluster_centers_)
```

```
    if show_xlabels:
        plt.xlabel("$x_1$", fontsize=14)
    else:
        plt.tick_params(labelbottom=False)
    if show_ylabels:
        plt.ylabel("$x_2$", fontsize=14, rotation=0)
    else:
        plt.tick_params(labelleft=False)

k = 4
print("--------------------------------------")
print("KMEANS")
kmeans = KMeans(n_clusters=4, random_state=42)
y_pred = kmeans.fit_predict(X2D)

plt.figure(figsize=(8, 4))
plot_decision_boundaries(kmeans, X2D)

print("the Inertia ,mean Square error:{}".format(kmeans.inertia_))
silhouette_scoreK = silhouette_score(X2D, kmeans.labels_)
print("Silhouette score for DBSCAN with eps=0.05 is {}".format(silhouette_scoreK))

plt.show()

#kmeans for 10 values of k
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(XD)
                for k in range(1, 10)]

kmeans_per_k_2DPCA = [KMeans(n_clusters=k, random_state=42).fit(XD)
```

```
            for k in range(1, 10)]

def Kmeans_Inertia_Score(XD,title,kmeans_per_k):

    inertias = [model.inertia_ for model in kmeans_per_k]

    plt.figure(figsize=(8, 3.5))
    plt.plot(range(1, 10), inertias, "bo-")
    plt.xlabel("$k$", fontsize=14)
    plt.ylabel("Inertia", fontsize=14)
    plt.annotate('Elbow',
                 xy=(4, inertias[3]),
                 xytext=(0.55, 0.55),
                 textcoords='figure fraction',
                 fontsize=16,
                 arrowprops=dict(facecolor='black', shrink=0.1)
                )
    plt.title("inertia score against k "+title)
    plt.show()
#inertia score
print("KMEANS Inertia- Mean square error score")
Kmeans_Inertia_Score(X,'without PCA transformation',kmeans_per_k)

Kmeans_Inertia_Score(X2D,' for nPCA=2',kmeans_per_k_2DPCA)



def SilhouetteScore(XD,title,labels,kmeans_per_k):

    silhouette_scoresO = [silhouette_score(XD, model.labels_)
```

```
                    for model in kmeans_per_k[1:]]
    plt.figure(figsize=(8, 3))
    plt.plot(range(2, 10), silhouette_scoresO, "bo-")
    plt.xlabel("$k$", fontsize=14)
    plt.ylabel("Silhouette score", fontsize=14)
    plt.title("Silhouette score against k without PCA transformation")
    plt.show()

#Silhpuete score kmeans
SilhouetteScore(X,'without PCA transformation',kmeans.labels_,kmeans_per_k)
SilhouetteScore(X2D,' for nPCA=2',kmeans.labels_,kmeans_per_k_2DPCA)

#DBSCAN
from sklearn.cluster import DBSCAN
print("---------------------------------------------------------")
print("DBSCAN ")
dbscan = DBSCAN(eps=0.05, min_samples=5)
dbscan.fit(X)

silhouette_scores_DBSCAN = silhouette_score(X, dbscan.labels_)
print("Silhouette score for DBSCAN with eps=0.05 is {}".format(silhouette_scores_DBSCAN))

dbscan.fit(X2D)

silhouette_scores_DBSCAN = silhouette_score(X2D, dbscan.labels_)
print("Silhouette score for DBSCAN with eps=0.05 is {}".format(silhouette_scores_DBSCAN))

dbscan2 = DBSCAN(eps=0.2)
dbscan2.fit(X)
```

```
silhouette_scores_DBSCAN = silhouette_score(X, dbscan.labels_)
print("Silhouette score for DBSCAN with eps=0.2 is {}".format(silhouette_scores_DBSCAN))
dbscan2.fit(X2D)
silhouette_scores_DBSCAN = silhouette_score(X2D, dbscan.labels_)
print("Silhouette score for DBSCAN with eps=0.05 is {}".format(silhouette_scores_DBSCAN))

def plot_dbscan(dbscan, X, show_xlabels=True, show_ylabels=True):
    core_mask = np.zeros_like(dbscan.labels_, dtype=bool)
    core_mask[dbscan.core_sample_indices_] = True
    anomalies_mask = dbscan.labels_ == -1
    non_core_mask = ~(core_mask | anomalies_mask)

    cores = dbscan.components_
    anomalies = X[anomalies_mask]
    non_cores = X[non_core_mask]

    plt.scatter(cores[:, 0], cores[:, 1],
                c=dbscan.labels_[core_mask], marker='o',  cmap="Paired")
    plt.scatter(cores[:, 0], cores[:, 1], marker='*', s=20, c=dbscan.labels_[core_mask])
    plt.scatter(anomalies[:, 0], anomalies[:, 1],
                c="r", marker="x", s=100)
    plt.scatter(non_cores[:, 0], non_cores[:, 1], c=dbscan.labels_[non_core_mask], marker=".")
    if show_xlabels:
        plt.xlabel("$x_1$", fontsize=14)
    else:
        plt.tick_params(labelbottom=False)
    if show_ylabels:
        plt.ylabel("$x_2$", fontsize=14, rotation=0)
    else:
```

```
        plt.tick_params(labelleft=False)
    plt.title("eps={:.2f}, min_samples={}".format(dbscan.eps, dbscan.min_samples), fontsize=14)
plt.figure(figsize=(9, 3.2))
plt.subplot(121)
plot_dbscan(dbscan, X2D)
plt.subplot(122)
plot_dbscan(dbscan2, X2D, show_ylabels=False)
plt.show()
```