Chapter 9: K-Means Clustering

Ex1: 3D points

Dataset: data3D.csv.

Requirement:

- Read dataset
- Pre-process data
- Use K-means clustering algorithm to cluster 3D points in data3D.csv.

In [8]:
```
import findspark
findspark.init()
```

In [9]:
```
import pyspark
```

In [10]:
```
from pyspark.sql import SparkSession
```

In [11]:
```
spark = SparkSession.builder.appName('kmeans_3D_point').getOrCreate()
```

In [12]:
```
# Loads data.
data = spark.read.csv("data3D.csv", header=True,
                      inferSchema=True)
```

In [13]:
```
data.show(3)
```

```
+------+-----------------+------------------+------------------+
|    id|                x|                 y|                 z|
+------+-----------------+------------------+------------------+
|point0|5.647627534046943|-6.356222340123802|-7.240816026826695|
|point1|4.414367138680041|-10.32624175635328| 8.963324308916228|
|point2|5.005396944639823|-9.301070062115645| 10.35473056351597|
+------+-----------------+------------------+------------------+
only showing top 3 rows
```

In [14]:
```
from pyspark.sql.functions import col
```

In [17]:
```
data = data.select(['x', 'y', 'z'])
```

In [18]:
```
data.show(3)
```

```
+-----------------+------------------+------------------+
|                x|                 y|                 z|
+-----------------+------------------+------------------+
|5.647627534046943|-6.356222340123802|-7.240816026826695|
|4.414367138680041|-10.32624175635328| 8.963324308916228|
|5.005396944639823|-9.301070062115645| 10.35473056351597|
+-----------------+------------------+------------------+
only showing top 3 rows
```

Format from data

In [19]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [20]:
```
data.columns
```

Out[20]:
```
['x', 'y', 'z']
```

In [21]:
```
vec_assembler = VectorAssembler(inputCols = data.columns,
                                outputCol='features')
```

In [25]:
```
final_data = vec_assembler.transform(data)
```

Scale the Data

In [22]:
```
from pyspark.ml.feature import StandardScaler
```

In [23]:
```
scaler = StandardScaler(inputCol="features",
                        outputCol="scaledFeatures",
                        withStd=True,
                        withMean=False)
```

In [26]:
```
# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(final_data)
```

In [27]:
```
# Normalize each feature to have unit standard deviation.
final_data = scalerModel.transform(final_data)
```

In [28]:
```
final_data.show(3, False)
```

```
+-----------------+------------------+------------------+----------------------
-----------------------------------+-----------------------------------------------
----------------+
|x                |y                 |z                 |features
|scaledFeatures                                                      |
+-----------------+------------------+------------------+----------------------
-----------------------------------+-----------------------------------------------
----------------+
|5.647627534046943|-6.356222340123802|-7.240816026826695|[5.647627534046943,-6.
356222340123802,-7.240816026826695]|[1.0159673512169811,-0.81335799160424,-1.13
00631023636807] |
|4.414367138680041|-10.32624175635328|8.963324308916228 |[4.414367138680041,-1
0.32624175635328,8.963324308916228] |[0.7941127247055396,-1.3213715327025133,1.
3988923401033817]|
|5.005396944639823|-9.301070062115645|10.35473056351597 |[5.005396944639823,-9.
301070062115645,10.35473056351597] |[0.9004347126254774,-1.1901880174546189,1.6
160469899240197]|
+-----------------+------------------+------------------+----------------------
-----------------------------------+-----------------------------------------------
----------------+
only showing top 3 rows
```

Train the Model and Evaluate

Seleck k with minimun WSSSE: k between 2 - 10

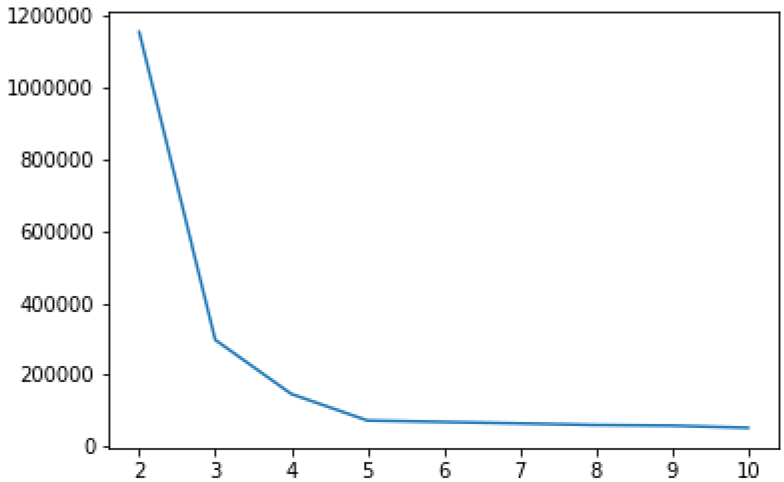
In [29]:
```
from pyspark.ml.clustering import KMeans
```

In [31]:
```
# Trains a k-means model.
k_list = []
wssse_list = []
for k in range(2,11):
    kmeans = KMeans(featuresCol='scaledFeatures',k=k)
    model = kmeans.fit(final_data)
    wssse = model.computeCost(final_data)
    k_list.append(k)
    wssse_list.append(wssse)
    print("With k =", k, "Set Sum of Squared Errors = " + str(wssse))
```

```
With k = 2 Set Sum of Squared Errors = 1155067.2563008904
With k = 3 Set Sum of Squared Errors = 297656.40920043044
With k = 4 Set Sum of Squared Errors = 146718.50451770602
With k = 5 Set Sum of Squared Errors = 72720.18504204818
With k = 6 Set Sum of Squared Errors = 68583.68208541229
With k = 7 Set Sum of Squared Errors = 64483.499921190276
With k = 8 Set Sum of Squared Errors = 60180.384358530224
With k = 9 Set Sum of Squared Errors = 58364.71279295459
With k = 10 Set Sum of Squared Errors = 52251.866281251554
```

In [32]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [33]:
```
plt.plot(k_list, wssse_list)
plt.show()
```



- According to Elbow Method, we choose k = 5. Look like there is very little gain after k=5, so we stick to that choice when processing the full data set.

Select k = 5

In [34]:
```
# Trains a k-means model.
kmeans = KMeans(featuresCol='scaledFeatures',k=5)
model = kmeans.fit(final_data)
```

In [35]:
```
# Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(final_data)
print("Within Set Sum of Squared Errors = " + str(wssse))
```

```
Within Set Sum of Squared Errors = 72720.18504204816
```

In [36]:
```
# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[-1.59030525  0.93719373  0.31581855]
[ 1.19780483 -0.7365009  -0.99420908]
[ 0.74896823 -1.2269015   1.46702482]
[-0.45182036  1.15367577  0.72369935]
[ 0.35333753 -0.87989287 -1.0735622 ]
```

In [37]:
```
predictions =  model.transform(final_data)
```

In [38]:
```
predictions.select("prediction").show(5)
```

```
+----------+
|prediction|
+----------+
|         1|
|         2|
|         2|
|         2|
|         0|
+----------+
only showing top 5 rows
```

In [ ]:
```
# Check number points of each cluster
```

In [60]:
```
predictions.groupBy('prediction').count().show()
```

```
+----------+------+
|prediction| count|
+----------+------+
|         1|199987|
|         3|200017|
|         4|200013|
|         2|200000|
|         0|199983|
+----------+------+
```

In [ ]:
```
# Our clustering algorithm created 5 equally sized clusters with K=5
```

In [39]:
```
data_result = predictions.select("prediction")
data_result.columns
```

Out[39]:
```
['prediction']
```

In [40]:
```
type(data_result)
```

Out[40]:
```
pyspark.sql.dataframe.DataFrame
```

In [41]:
```
final_data.show(3, False)
```

```
+-----------------+------------------+-----------------+---------------------
-----------------------------------+-----------------------------------------
----------------+
|x                |y                 |z                |features
|scaledFeatures                                                  |
+-----------------+------------------+-----------------+---------------------
-----------------------------------+-----------------------------------------
----------------+
|5.647627534046943|-6.356222340123802|-7.240816026826695|[5.647627534046943,-6.
356222340123802,-7.240816026826695]|[1.0159673512169811,-0.81335799160424,-1.13
00631023636807] |
|4.414367138680041|-10.32624175635328|8.963324308916228 |[4.414367138680041,-1
0.32624175635328,8.963324308916228] |[0.7941127247055396,-1.3213715327025133,1.
3988923401033817]|
|5.005396944639823|-9.301070062115645|10.35473056351597 |[5.005396944639823,-9.
301070062115645,10.35473056351597] |[0.9004347126254774,-1.1901880174546189,1.6
160469899240197]|
+-----------------+------------------+-----------------+---------------------
-----------------------------------+-----------------------------------------
----------------+
only showing top 3 rows
```

In [42]:
```
temp = final_data.select("scaledFeatures").rdd.map(lambda x: \
                                          x[0].toArray().tolist()).toDF()
```

In [43]:
```
temp.show(3)
```

```
+------------------+-------------------+-------------------+
|                _1|                 _2|                 _3|
+------------------+-------------------+-------------------+
|1.0159673512169811|  -0.81335799160424|-1.1300631023636807|
|0.7941127247055396|-1.3213715327025133| 1.3988923401033817|
|0.9004347126254774|-1.1901880174546189| 1.6160469899240197|
+------------------+-------------------+-------------------+
only showing top 3 rows
```

In [44]:
```
import pyspark.sql.functions as f
```

In [45]:
```
# since there is no common column between these two dataframes add row_index so t
temp=temp.withColumn('row_index', f.monotonically_increasing_id())
data_result=data_result.withColumn('row_index',
                                   f.monotonically_increasing_id())
temp = temp.join(data_result,
                 on=["row_index"]).sort("row_index").drop("row_index")
```

In [46]:
```
temp.show(3)
```

```
+------------------+------------------+------------------+----------+
|                _1|                _2|                _3|prediction|
+------------------+------------------+------------------+----------+
|1.0159673512169811|  -0.81335799160424|-1.1300631023636807|         1|
|0.7941127247055396|-1.3213715327025133| 1.3988923401033817|         2|
|0.9004347126254774|-1.1901880174546189| 1.6160469899240197|         2|
+------------------+------------------+------------------+----------+
only showing top 3 rows
```

In [48]:
```
temp = temp.select(col("_1").alias("x_scale"),
                   col("_2").alias("y_scale"),
                   col("_3").alias("z_scale"),
                   "prediction")
```

In [49]:
```
df = temp.toPandas()
```

In [54]:
```
df.head(3)
```

Out[54]:

	x_scale	y_scale	z_scale	prediction
0	1.015967	-0.813358	-1.130063	1
1	0.794113	-1.321372	1.398892	2
2	0.900435	-1.190188	1.616047	2

In [50]:
```
centers_df = pd.DataFrame(centers)
centers_df.head()
```

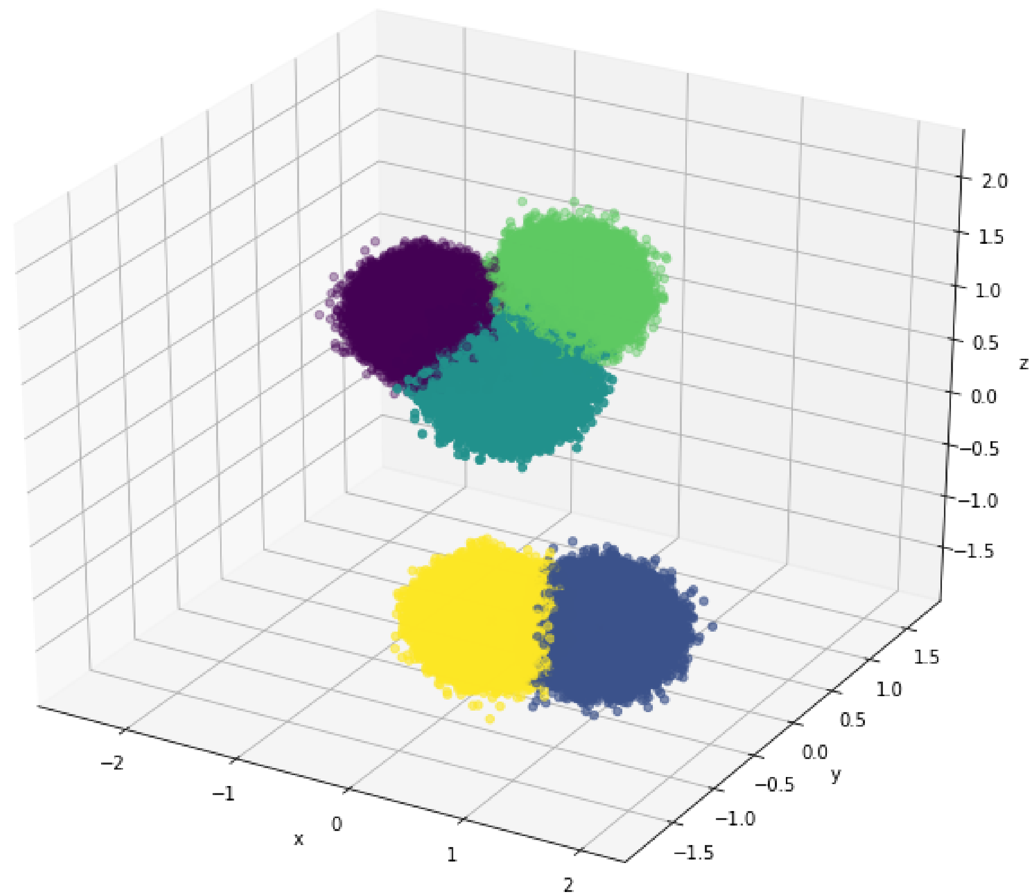
Out[50]:

	0	1	2
0	-1.590305	0.937194	0.315819
1	1.197805	-0.736501	-0.994209
2	0.748968	-1.226902	1.467025
3	-0.451820	1.153676	0.723699
4	0.353338	-0.879893	-1.073562

In [55]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [58]:
```
threedee = plt.figure(figsize=(12,10)).gca(projection='3d')
threedee.scatter(df.x_scale, df.y_scale,
                 df.z_scale,
                 c=df.prediction)
threedee.set_xlabel('x')
threedee.set_ylabel('y')
threedee.set_zlabel('z')

plt.show()
```



Combine results

In [51]:
```
# since there is no common column between these two dataframes add row_index s
final_data=final_data.withColumn('row_index',
                                 f.monotonically_increasing_id())
temp=temp.withColumn('row_index',
                     f.monotonically_increasing_id())
final_data = final_data.join(temp,
                             on=["row_index"]).sort("row_index").drop("row_index
```

In [52]:
```
final_data.show(3, False)
```

```
+-----------------+------------------+------------------+----------------------
-----------------------------------+---------------------------------------------
----------------+------------------+-------------------+-------------------+---
-------+
|x                |y                 |z                 |features
|scaledFeatures                                                        |x_scale
|y_scale            |z_scale             |prediction|
+-----------------+------------------+------------------+----------------------
-----------------------------------+---------------------------------------------
----------------+------------------+-------------------+-------------------+---
-------+
|5.647627534046943|-6.356222340123802|-7.240816026826695|[5.647627534046943,-6.
356222340123802,-7.240816026826695]|[1.0159673512169811,-0.81335799160424,-1.13
00631023636807] |1.0159673512169811|-0.81335799160424  |-1.1300631023636807|1
|
|4.414367138680041|-10.32624175635328|8.963324308916228 |[4.414367138680041,-1
0.32624175635328,8.963324308916228] |[0.7941127247055396,-1.3213715327025133,1.
3988923401033817]|0.7941127247055396|-1.3213715327025133|1.3988923401033817 |2
|
|5.005396944639823|-9.301070062115645|10.35473056351597 |[5.005396944639823,-9.
301070062115645,10.35473056351597] |[0.9004347126254774,-1.1901880174546189,1.6
160469899240197]|0.9004347126254774|-1.1901880174546189|1.6160469899240197 |2
|
+-----------------+------------------+------------------+----------------------
-----------------------------------+---------------------------------------------
----------------+------------------+-------------------+-------------------+---
-------+
only showing top 3 rows
```