HIERARCHICAL CLUSTERING

USING PYSPARK

DR. ALVIN ANG

CONTENTS

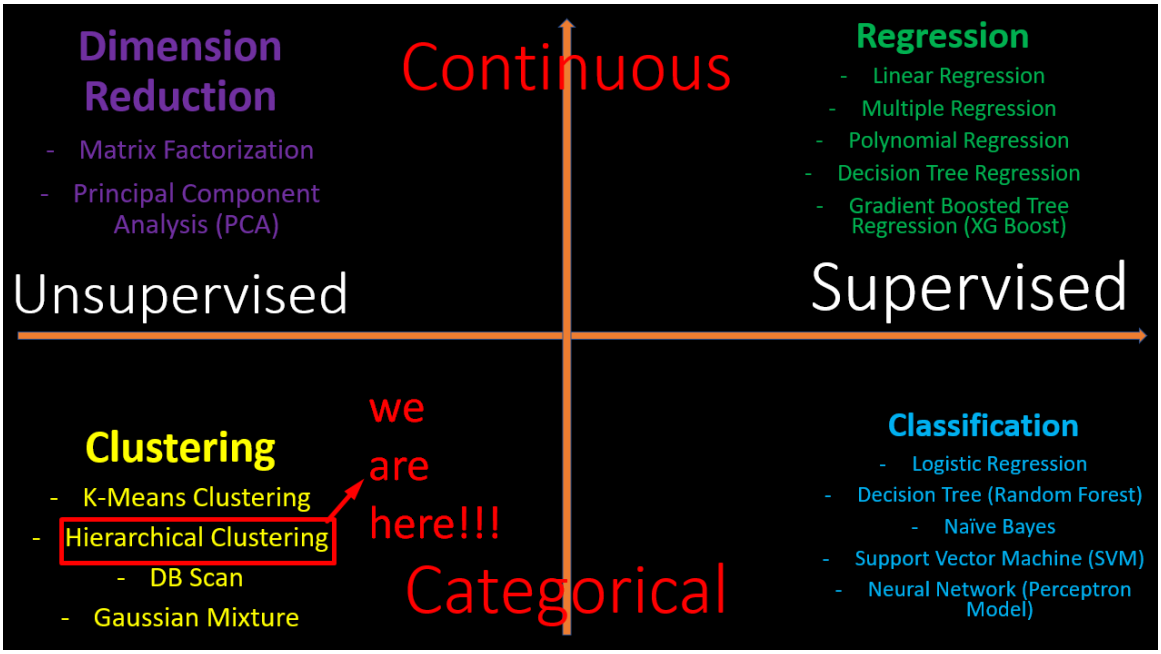
I. UNDERSTANDING HIERARCHICAL CLUSTERING = UNSUPERVISED MACHINE LEARNING

Most of the stuff here are abstracted frrom:

https://www.amazon.com/Machine-Learning-PySpark-Processing-Recommender/dp/1484241304

To learn more about Hierarchical Clustering:

https://www.alvinang.sg/s/Hierarchical-Clustering-with-Python-by-Dr-Alvin-Ang.pdf



- Above is a table categorizing the different Machine Learning algorithms.
- Objective of Hierarchical Clustering is to predict a CATEGORY.

A. IMPORTANT POINT TO NOTE ABOUT PREDICTION USING HIERARCHICAL CLUSTERING

- You CAN'T use Hierarchical Clustering to predict new datasets.[1]
- Hierarchical clustering is not designed to predict cluster labels for new observations.
- The reason why this is happening is because it just links data points according to their distances and it is not defining "regions" for each cluster.
- If you really need to predict, use K Means Clustering.

[1] https://stackoverflow.com/questions/64589016/how-to-predict-the-cluster-label-of-a-new-observation-using-a-hierarchical-clust

II. HIERARCHICAL CLUSTERING USING PYSPARK – CASE I LOCATING CLUSTER CENTROIDS (SALES.CSV)

IPYNB:
https://www.alvinang.sg/s/Hierarchical_Clustering_In_Spark_With_Bisecting_K_Means.ipynb

A. IMPORT THE DATA

importing the data

```
[58] from pyspark import SparkFiles

     url = 'https://www.alvinang.sg/s/sales.csv'
     spark.sparkContext.addFile(url)

     df_sales = spark.read.csv(SparkFiles.get("sales.csv"), \
                               header=True, inferSchema=True)
     df_sales
```

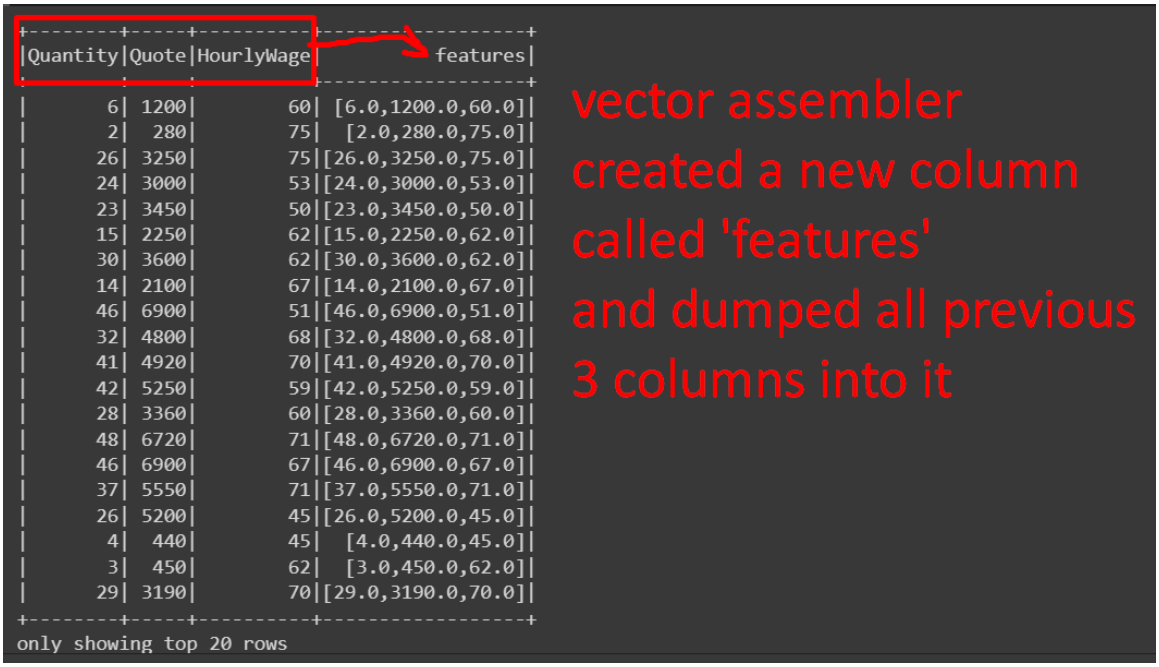
```
DataFrame[RowID: int, OrderID: int, OrderDate: string, OrderMonthYear: string, Quantity:
```

B. VECTOR ASSEMBLER: TRANSFORM THE DATA

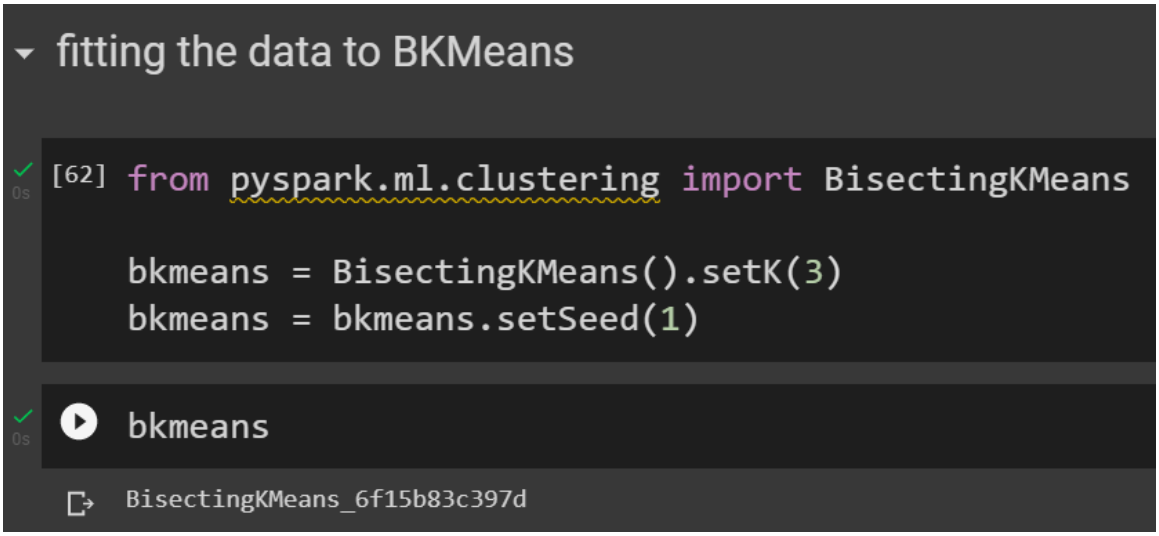
transforming the data

```
[59] from pyspark.ml.feature import VectorAssembler

     vectorAssembler = VectorAssembler(
         inputCols = ['Quantity', 'Quote', 'HourlyWage'],
         outputCol = 'features'
     )
```

```
     df_sales1 = vectorAssembler.transform(df_sales)

     df_sales1.select(['Quantity','Quote','HourlyWage','features']).show()
```



C. FITTING DATA TO BKMEANS



D. LOCATING THE CENTROIDS

finding the Centroids

```
model5 = bkmeans.fit(df_sales1)
centers2 = model5.clusterCenters()
centers2.sort(key=np.linalg.norm)

display(centers2)
```

location of the 3 centroids

```
[array([  7.01476015, 945.61808118,  61.17158672]),
 array([  20.79304112, 2775.37053773,   61.11387257]),
 array([  38.21652563, 5555.94201095,   60.80587357])]
```

III. HIERARCHICAL CLUSTERING USING PYSPARK – CASE II PREDICTING NUMBER OF CLUSTERS (IRIS DATASET)

Referenced mainly from:

https://www.data4v.com/tutorial-hierarchical-clustering-in-spark-with-bisecting-k-means/

IPYNB:

https://www.alvinang.sg/s/Hierarchical_Clustering_In_Spark_With_Bisecting_K_Means.ipynb

A. POINTS TO NOTE

- PySpark is unable to perform Hierarchical Clustering directly.
- Because its MLlib library doesn't contain it.
- However, in Spark, both K means and Hierarchical Clustering are combined using a version of K-Means called as Bisecting K-Means.
- It is a divisive hierarchical clustering algorithm (won't be explained here).

B. START A SPARK SESSION

https://www.alvinang.sg/s/How_To_Start_A_Spark_Session.ipynb

```
[4] !apt-get install openjdk-8-jdk-headless -qq > /dev/null

    !wget -q https://dlcdn.apache.org/spark/spark-3.2.1/spark-3.2.1-bin-hadoop3.2.tgz

[6] !tar xf spark-3.2.1-bin-hadoop3.2.tgz

[7] !pip install -q findspark

[8] import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.2.1-bin-hadoop3.2"

[9] os.environ["SPARK_HOME"]

'/content/spark-3.2.1-bin-hadoop3.2'

[10] import findspark
     findspark.init()

[11] from pyspark.sql import SparkSession
     spark = SparkSession.builder.master("local[*]").getOrCreate()

[12] print(spark.version)

3.2.1
```

C. LOAD THE IRIS DATASET

```
import pandas as pd
from sklearn.datasets import load_iris
from pyspark.sql import SparkSession

df_iris = load_iris(as_frame=True)
```

```
df_iris
```

```
{'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Charac
 'data':      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns],
'data_module': 'sklearn.datasets.data',
'feature_names': ['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)'],
'filename': 'iris.csv',
'frame':      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     target
0         0
1         0
2         0
3         0
4         0
```

- As can be seen, the Iris dataset is stored as a Dictionary with many key-value pairs

D. DATA EXTRACTION

```
[12] pd_df_iris = pd.DataFrame(df_iris.data, columns = df_iris.feature_names)
     pd_df_iris['target'] = pd.Series(df_iris.target)
```

- We extract out frrom the Iris Dictionary:
    - 'data' → as values in the rows
    - 'feature_names' → as column headers

```
pd_df_iris
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2

we create a new column called 'target' and put the df_iris['target']

dictionary values into it

E. LOAD DATASET TO SPARK DATAFRAME

```
[13] spark_df_iris = spark.createDataFrame(pd_df_iris)
     spark_df_iris = spark_df_iris.drop("target")
```

```
spark_df_iris.show()
```

```
+-----------------+----------------+-----------------+----------------+
|sepal length (cm)|sepal width (cm)|petal length (cm)|petal width (cm)|
+-----------------+----------------+-----------------+----------------+
|              5.1|             3.5|              1.4|             0.2|
|              4.9|             3.0|              1.4|             0.2|
|              4.7|             3.2|              1.3|             0.2|
|              4.6|             3.1|              1.5|             0.2|
|              5.0|             3.6|              1.4|             0.2|
|              5.4|             3.9|              1.7|             0.4|
|              4.6|             3.4|              1.4|             0.3|
|              5.0|             3.4|              1.5|             0.2|
|              4.4|             2.9|              1.4|             0.2|
|              4.9|             3.1|              1.5|             0.1|
```

spark dataframe now stores the data but the 'target' column is dropped

F. VECTOR ASSEMBLER

VectorAssembler

```
from pyspark.ml.feature import VectorAssembler

assemble=VectorAssembler(inputCols=[
    'sepal length (cm)',
    'sepal width (cm)',
    'petal length (cm)',
    'petal width (cm)'],outputCol = 'iris_features')

assembled_data=assemble.transform(spark_df_iris)
```

```
assembled_data.show()
```

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	iris_features
5.1	3.5	1.4	0.2	[5.1,3.5,1.4,0.2]
4.9	3.0	1.4	0.2	[4.9,3.0,1.4,0.2]
4.7	3.2	1.3	0.2	[4.7,3.2,1.3,0.2]
4.6	3.1	1.5	0.2	[4.6,3.1,1.5,0.2]
5.0	3.6	1.4	0.2	[5.0,3.6,1.4,0.2]
5.4	3.9	1.7	0.4	[5.4,3.9,1.7,0.4]
4.6	3.4	1.4	0.3	[4.6,3.4,1.4,0.3]
5.0	3.4	1.5	0.2	[5.0,3.4,1.5,0.2]
4.4	2.9	1.4	0.2	[4.4,2.9,1.4,0.2]
4.9	3.1	1.5	0.1	[4.9,3.1,1.5,0.1]
5.4	3.7	1.5	0.2	[5.4,3.7,1.5,0.2]
4.8	3.4	1.6	0.2	[4.8,3.4,1.6,0.2]
4.8	3.0	1.4	0.1	[4.8,3.0,1.4,0.1]

we use Vector Assembler to create a new column called iris_features where we take all previous 4 columns and dump them into it...

G. USING THE CLUSTERING EVALUATOR TO FIGURE OUT HOW MANY CLUSTERS

```
from pyspark.ml.clustering import BisectingKMeans
from pyspark.ml.evaluation import ClusteringEvaluator

silhouette_scores=[]
evaluator = ClusteringEvaluator(featuresCol='iris_features', metricName='silhouette')

for K in range(2,11):
    BKMeans_=BisectingKMeans(featuresCol='iris_features', k=K, minDivisibleClusterSize =1)
    BKMeans_fit=BKMeans_.fit(assembled_data)
    BKMeans_transform=BKMeans_fit.transform(assembled_data)
    evaluation_score=evaluator.evaluate(BKMeans_transform)
    silhouette_scores.append(evaluation_score)
```

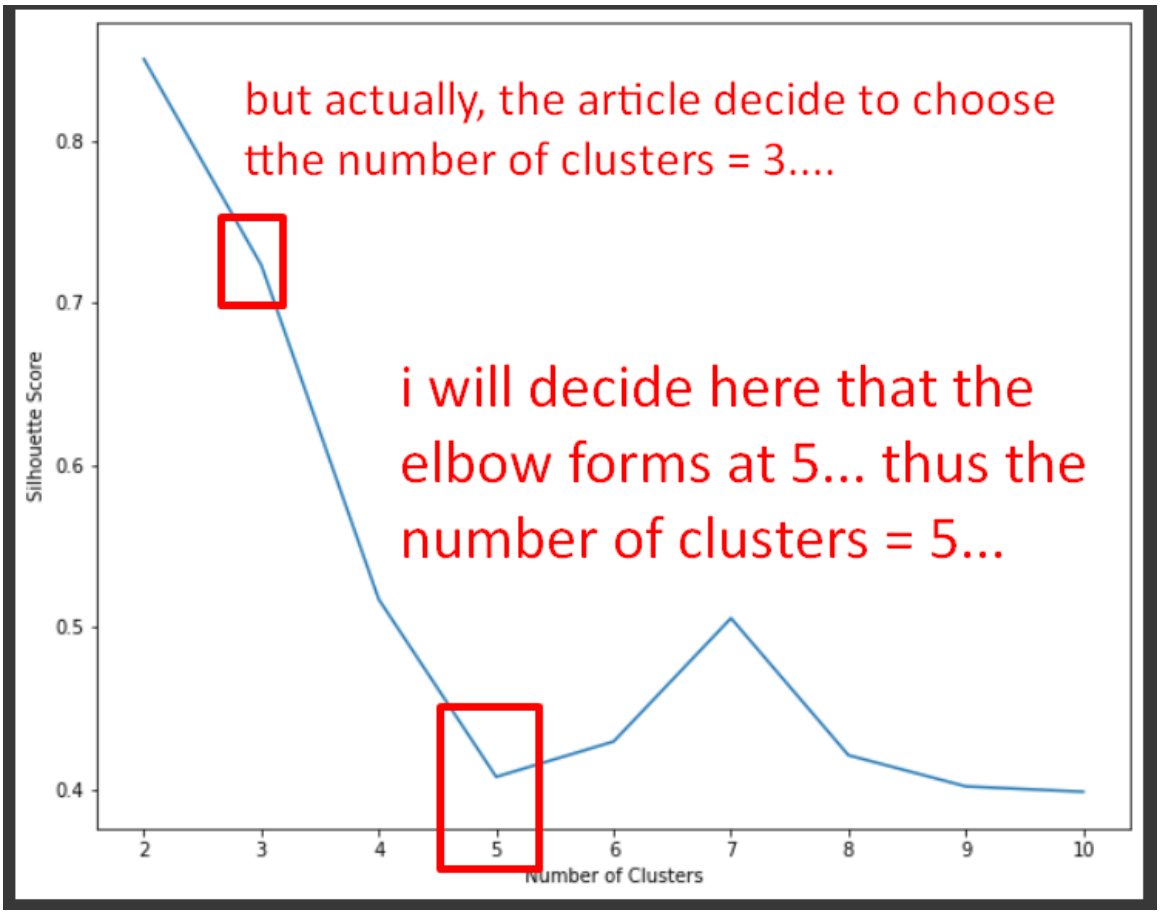
```
silhouette_scores
```

```
[0.8503512229251465,
 0.7232089398466957,
 0.5171303891686024,
 0.4076299389603104,
 0.42940938304585574,
 0.5055011696390824,
 0.42096575425537813,
 0.4017369113499781,
 0.3984297987174316]
```

- We import the Clustering Evaluator to find out the "Silhouette Scores" (or errors) as we increase the Clusters from 2 to 11.
- The Scores / Error goes down as the clusters increase because the Error is measured from every point to the centroid of the cluster.
- Thus, if we have the number of centroids = number of points, the Error = 0 because the distance = 0.
- This is displayed in the Elbow diagram in the next part.

H. PLOTTING THE ELBOW DIAGRAM

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1, figsize =(10,8))
ax.plot(range(2,11),silhouette_scores)
ax.set_xlabel('Number of Clusters')
ax.set_ylabel('Silhouette Score')
```



- This plot plots the Silhouette Scores as per previous section (from 2 to 11).

I. FITTING THE DATA TO BK MEANS

```
[31] BKMeans_=BisectingKMeans(featuresCol='iris_features', k=5)
     BKMeans_Model=BKMeans_.fit(assembled_data)
     BKMeans_transform=BKMeans_Model.transform(assembled_data)
```

we decide upon 5 clusters

```
BKMeans_transform.show()
```

```
+-----------------+----------------+-----------------+----------------+--------------------+----------+
|sepal length (cm)|sepal width (cm)|petal length (cm)|petal width (cm)|      iris_features|prediction|
+-----------------+----------------+-----------------+----------------+--------------------+----------+
|              5.1|             3.5|              1.4|             0.2|[5.1,3.5,1.4,0.2]|         1|
|              4.9|             3.0|              1.4|             0.2|[4.9,3.0,1.4,0.2]|         0|
|              4.7|             3.2|              1.3|             0.2|[4.7,3.2,1.3,0.2]|         0|
|              4.6|             3.1|              1.5|             0.2|[4.6,3.1,1.5,0.2]|         0|
|              5.0|             3.6|              1.4|             0.2|[5.0,3.6,1.4,0.2]|         1|
|              5.4|             3.9|              1.7|             0.4|[5.4,3.9,1.7,0.4]|         1|
|              4.6|             3.4|              1.4|             0.3|[4.6,3.4,1.4,0.3]|         0|
|              5.0|             3.4|              1.5|             0.2|[5.0,3.4,1.5,0.2]|         0|
|              4.4|             2.9|              1.4|             0.2|[4.4,2.9,1.4,0.2]|         0|
|              4.9|             3.1|              1.5|             0.1|[4.9,3.1,1.5,0.1]|         0|
|              5.4|             3.7|              1.5|             0.2|[5.4,3.7,1.5,0.2]|         1|
|              4.8|             3.4|              1.6|             0.2|[4.8,3.4,1.6,0.2]|         0|
|              4.8|             3.0|              1.4|             0.1|[4.8,3.0,1.4,0.1]|         0|
|              4.3|             3.0|              1.1|             0.1|[4.3,3.0,1.1,0.1]|         0|
|              5.8|             4.0|              1.2|             0.2|[5.8,4.0,1.2,0.2]|         1|
```

The dataset was fitted to the BK Means algorithm and it created a new column called 'prediction'

And it used 'iris_features' to predict

thus, for each row, the prediction is either 0 / 1 / 2 / 3 / 4

J. PCA

```
[41] from pyspark.ml.feature import PCA as PCAml
     pca = PCAml(k=2, inputCol="iris_features", outputCol="pca")
     pca_model = pca.fit(assembled_data)
     pca_transformed = pca_model.transform(assembled_data)
```

```
[46] pca_transformed.show()
```

```
+-----------------+----------------+-----------------+----------------+--------------------+--------------------+
|sepal length (cm)|sepal width (cm)|petal length (cm)|petal width (cm)|      iris_features|                 pca|
+-----------------+----------------+-----------------+----------------+--------------------+--------------------+
|              5.1|             3.5|              1.4|             0.2|[5.1,3.5,1.4,0.2]|[-2.8182395066394...|
|              4.9|             3.0|              1.4|             0.2|[4.9,3.0,1.4,0.2]|[-2.7882234453146...|
|              4.7|             3.2|              1.3|             0.2|[4.7,3.2,1.3,0.2]|[-2.6133745635497...|
|              4.6|             3.1|              1.5|             0.2|[4.6,3.1,1.5,0.2]|[-2.7570222769675...|
|              5.0|             3.6|              1.4|             0.2|[5.0,3.6,1.4,0.2]|[-2.7736485960544...|
|              5.4|             3.9|              1.7|             0.4|[5.4,3.9,1.7,0.4]|[-3.2215054997645...|
|              4.6|             3.4|              1.4|             0.3|[4.6,3.4,1.4,0.3]|[-2.6818273818683...|
```

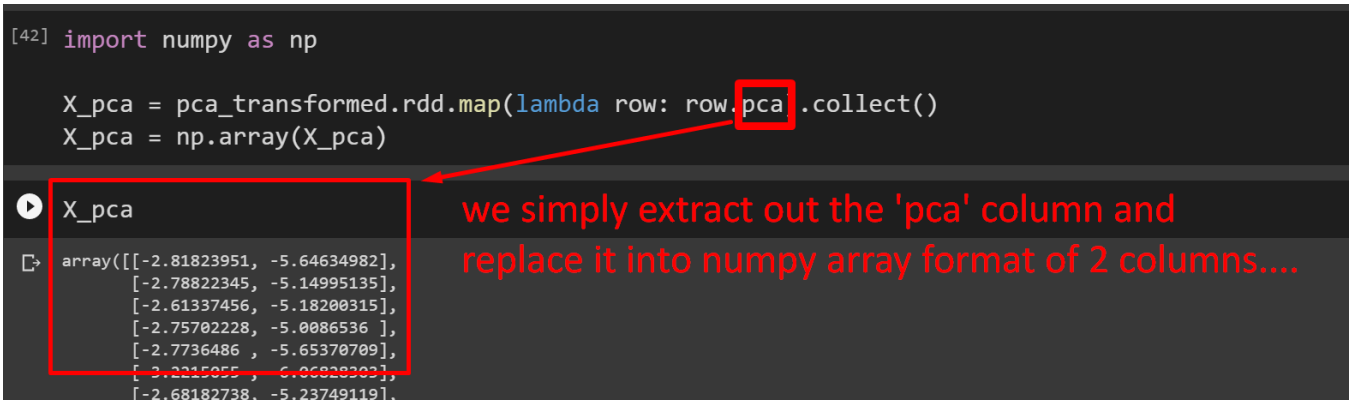
what PCA does is to compress the 4 columns (which have already been placed into'iris_features' and convert into 2 columns in 'pca'

- We do PCA because we want to plot the Clusters onto a 2D diagram. (for better visibility).
- To learn more about PCA: https://www.alvinang.sg/s/Principal-Component-Analysis-PCA-with-Python-and-PySpark-by-Dr-Alvin-Ang.pdf
- Because there are currently 4 columns / features / dimensions:
    - Sepal length

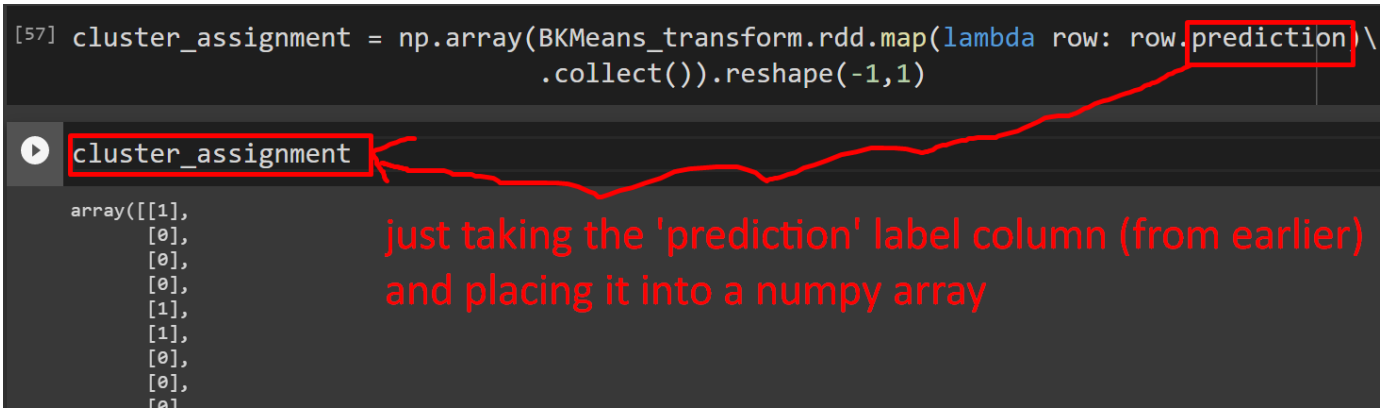
- Sepal width
- Petal length
- Petal width

- Yet we want to visualize this 4-dimensional data into 2, thus we need to compress these 4 columns into 2 using PCA.

1. EXTRACTING OUT 'PCA' COLUMN INTO 'X_PCA'

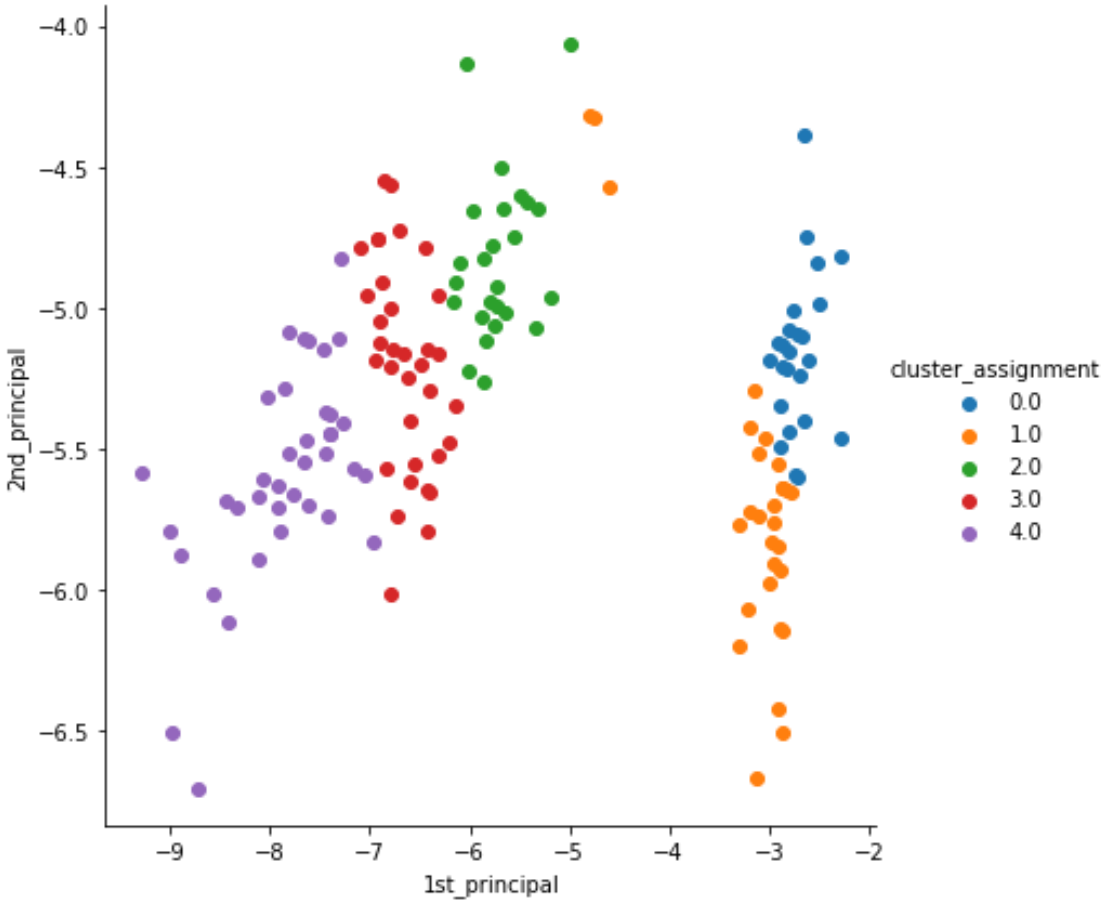


2. EXTRACTING OUT 'PREDICTION' COLUMN INTO 'CLUSTER_ASSIGNMENT'



K. PLOTTING

```
import seaborn as sns
import matplotlib.pyplot as plt

pca_data = np.hstack((X_pca,cluster_assignment))

pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal", "2nd_principal","cluster_assignment"))
sns.FacetGrid(pca_df,hue="cluster_assignment", height=6).map(plt.scatter, '1st_principal', \
                                                             '2nd_principal' ).add_legend()
plt.show()
```



- 5 classes.

ABOUT DR. ALVIN ANG

Dr. Alvin Ang earned his Ph.D., Masters and Bachelor degrees from NTU, Singapore. He is a scientist, entrepreneur, as well as a personal/business advisor. More about him at www.AlvinAng.sg.