#installing pyspark module onto colab

!pip install pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("test").getOrCreate()

spark

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

#performing k-means on sample dataframe created from pyspark

df = spark.createDataFrame([

                                [0, 35.3, 37.5,32.9],

                               [1, 41.4, -23.5,-15.4],

                               [2, 28.3, -13.3,10.2],

                               [3, 09.5, -9.0,1.0],

                               [4, 62.8, -18.23,17.66],

    [5, 63.8, -18.33,10.7],

    [6, 82.8, -17.23,3.33],

    [7, 52.8, -13.43,9.976],

    [8, 72.8, 48.23,11.13],

    [9, 65.8, 15.43,14.144],

    [10, 42.8, -13.23,-10.15],

    [11,11.65,23.14,39.25],

[12,32.34,56.23,10.23],

[13,23.56,-76.21,-23.93],

[14,45.78,-33.33,-9.27],

[15,59.89,-28.47,94.23],

[16,-46.83,29.04,48.23],

[17,45.36,72.83,-92.21],

[18,-26.37,49.28,40.73],

[19,29.49,-39.20,56.93],

[20,85.28,39.20,-49.21],

[21,42.45,-29.47,19.38],

[22,-22.16,40.72,91.30],

[23,20.39,40.17,20.30],

[24,-10.45,-36.59,-12.84],

[25,76.39,15.25,-29.12],

[26,5.60,20.45,29.14],

[27,39.19,-19.49,20.16],

[28,-14.27,38.47,-17.35],

[29,16.19,-38.15,32.43],

[30,56.26,-16.37,13.23],

[31,-94.38,16.19,34.13],

[32,52.14,36.17,-78.35],

[33,-41.25,98.23,75.26],

[34,-87.96,21.58,-47.69],

[35,21.75,63.48,56.42],

[36,43.21,36.14,-74.12],

[37,22.12,-47.26,31.45],

[38,-87.69,12.43,74.63],

[39,21.74,25.33,24.85],

[40,45.63,-75.96,12.27],

[41,22.74,56.34,94.25],

[42,10.25,-3.46,14.16],

[43,-42.12,84.23,12.17],

[44,45.65,-75.23,-36.12],

[45,23.62,-47.36,11.85],

[46,-77.25,74.66,-41.85],

[47,64.26,-33.78,-24.46],

[48,54.72,-21.42,45.78],

[49,24.48,-45.13,24.12],

[50,45.21,-42.12,-31.98],

[51,52.41,34.16,-45.74],

[52,-13.12,45.16,82.56],

[53,84.67,16.73,81.42],

[54,42.19,-47.84,12.35],

[55,-89.52,-86.44,-46.13],

[56,75.96,25.66,41.26],

[57,54.62,46.13,87.34],

[58,-46.42,-28.32,52.70],

[59,65.47,31.24,-66.41],

[60,96.58,-24.30,-50.21]

     ],["ID","Att\_1", "Att\_2","Att\_3"])

df.show()

Table

Description automatically generated

#Vector Assembler

from pyspark.ml.linalg import Vector

from pyspark.ml.feature import VectorAssembler

vecAssembler = VectorAssembler(inputCols=["Att\_1", "Att\_2","Att\_3"], outputCol="features")

new\_df = vecAssembler.transform(df)

new\_df.show()

Table

Description automatically generated

# K-means using pyspark

from pyspark.ml.clustering import KMeans

kmeans = KMeans(k=2, seed=1)

model = kmeans.fit(new\_df.select('features'))

transformed = model.transform(new\_df)

transformed.show()

Table

Description automatically generated

model.transform(new\_df).groupBy("prediction").count().show()

A picture containing shape

Description automatically generated

df1= df.toPandas()

df1

Table

Description automatically generated

#find the optimal number of clusters using elbow method

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import numpy as np

WCSS = []

for i in range(1,11):

    model = KMeans(n\_clusters = i,init = 'k-means++')

    model.fit(df1)

    WCSS.append(model.inertia\_)

fig = plt.figure(figsize = (7,7))

plt.plot(range(1,11),WCSS, linewidth=4, markersize=12,marker='o',color = 'green')

plt.xticks(np.arange(11))

plt.xlabel("Number of clusters")

plt.ylabel("WCSS (Within-Cluster Sum of Square)")

plt.show()

Chart, line chart

Description automatically generated

#since elbow is formed at cluster value k=4

model = KMeans(n\_clusters = 4, init = "k-means++")

k\_means = model.fit(df1)

k\_means.cluster\_centers\_

Table

Description automatically generated with medium confidence

df1['clusters']  = k\_means.labels\_

df1.head()

Table

Description automatically generated

df1['clusters'].value\_counts()

Text

Description automatically generated with medium confidence

import seaborn as sns

sns.scatterplot(x='Att\_1',y='Att\_2',hue='clusters',data=df1)

Chart, scatter chart

Description automatically generated

sns.scatterplot(x='Att\_2',y='Att\_3',hue='clusters',data=df1)

Chart, scatter chart

Description automatically generated

sns.scatterplot(x='Att\_3',y='Att\_1',hue='clusters',data=df1)

Chart, scatter chart

Description automatically generated

#Calculation of silhouette score with distance measure as squared Euclidean

from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator

silhouette\_score=[]

evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='features',

                                metricName='silhouette', distanceMeasure='squaredEuclidean')

for i in range(2,10):

    KMeans\_algo=KMeans(featuresCol='features', k=i)

    KMeans\_fit=KMeans\_algo.fit(new\_df)

    output=KMeans\_fit.transform(new\_df)

    score=evaluator.evaluate(output)

    silhouette\_score.append(score)

    print("Silhouette Score:",score)

Text

Description automatically generated

#Visualizing the silhouette scores in a plot

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1, figsize =(8,6))

ax.plot(range(2,10),silhouette\_score)

ax.set\_xlabel('k')

ax.set\_ylabel('cost')

Chart, line chart

Description automatically generated

#Calculation of silhouette score with distance measure as cosine

from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator

silhouette\_score=[]

evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='features',

                                metricName='silhouette', distanceMeasure='cosine')

for i in range(2,10):

    KMeans\_algo=KMeans(featuresCol='features', k=i)

    KMeans\_fit=KMeans\_algo.fit(new\_df)

    output=KMeans\_fit.transform(new\_df)

    score=evaluator.evaluate(output)

    silhouette\_score.append(score)

    print("Silhouette Score:",score)

Text

Description automatically generated

#Visualizing the silhouette scores in a plot

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1, figsize =(8,6))

ax.plot(range(2,10),silhouette\_score)

ax.set\_xlabel('k')

ax.set\_ylabel('cost')

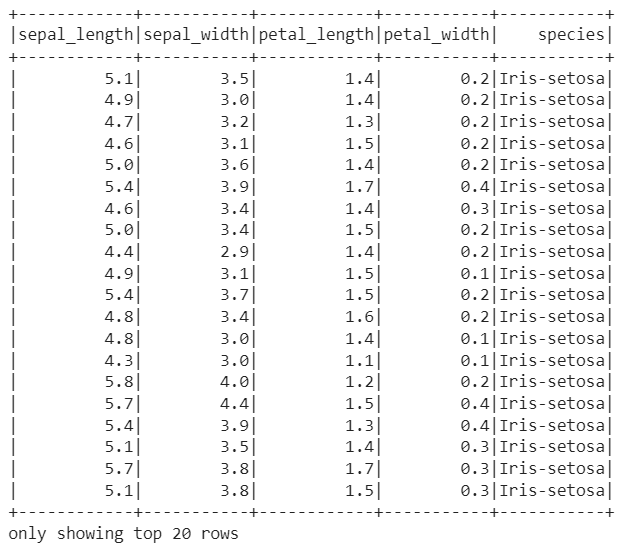
Chart, line chart

Description automatically generated

# Performing k-means upon iris dataset

data\_df = spark.read.csv("/content/drive/MyDrive/IRIS.csv",inferSchema =True,header = True)

data\_df.show()



data\_df.count() #150

len(data\_df.columns) #5

data\_df.printSchema()

Text

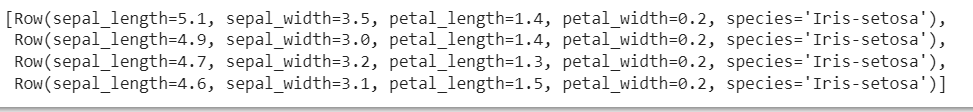
Description automatically generated

data\_df.describe().show()

Table

Description automatically generated

data\_df.head(4)



import seaborn as sns

data\_df1 = data\_df.toPandas()

data\_df1.head(4)

sns.pairplot(data\_df1[['sepal\_length','sepal\_width','petal\_length','petal\_width']])

Graphical user interface, application, table

Description automatically generated

Chart, histogram

Description automatically generated

data\_df.groupby("species").count().show()

Table

Description automatically generated

data\_df.groupby("petal\_length").count().show()

Table

Description automatically generated

# vector assembler

from pyspark.ml.linalg import Vector

from pyspark.ml.feature import VectorAssembler

data\_df.columns

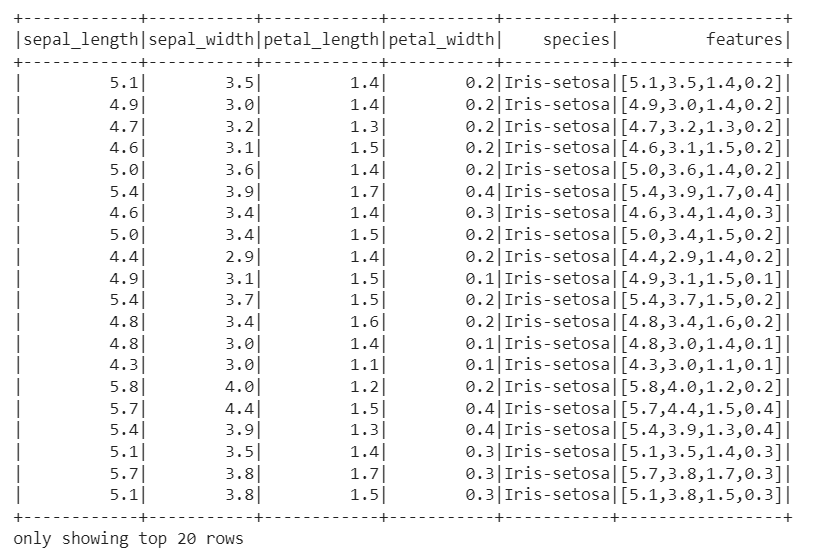


input\_columns = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

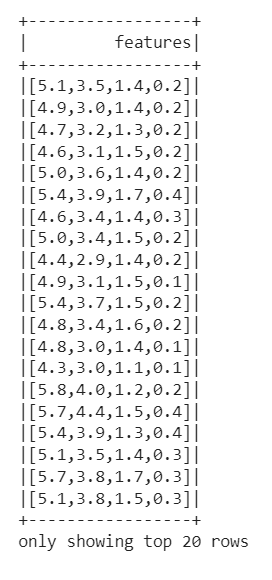
vec\_Assembler =  VectorAssembler(inputCols=input\_columns,outputCol = "features")

final\_data = vec\_Assembler.transform(data\_df)

final\_data.show()



final\_data.select("features").show()



# k-means

from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator

kmeans = KMeans(featuresCol="features",k=3)

model = kmeans.fit(final\_data)

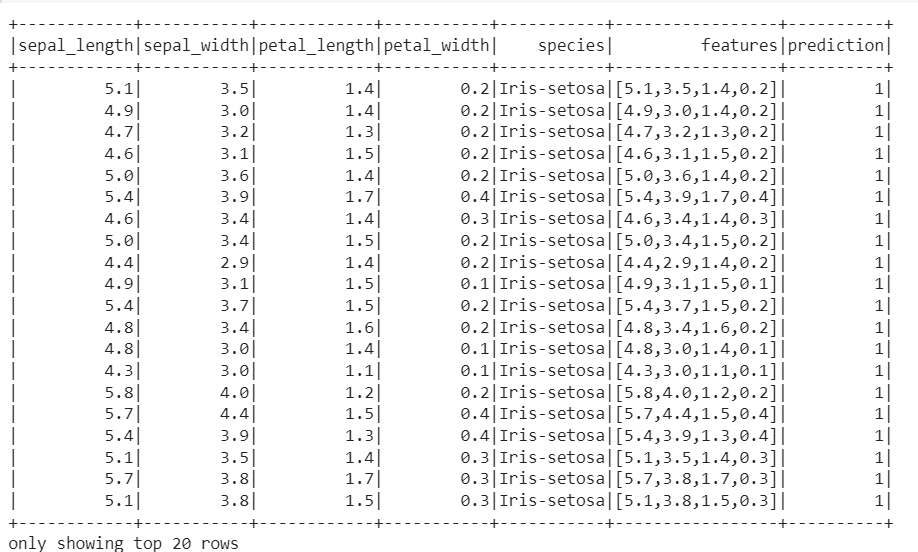
model



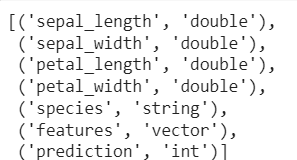
model.transform(final\_data).groupBy("prediction").count().show()

predictions= model.transform(final\_data)

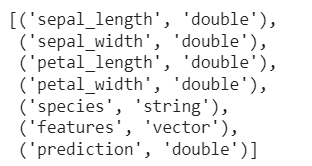
predictions.show()



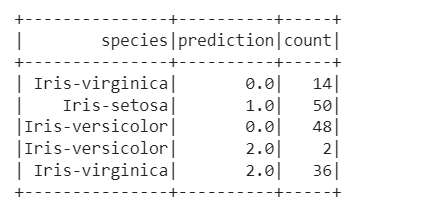
predictions.dtypes



predictions = predictions.withColumn("prediction",predictions.prediction.cast('double'))



predictions.groupBy("species","prediction").count().show()



from pyspark.sql.window import Window

import pyspark.sql.functions as F

from pyspark.sql.functions import row\_number

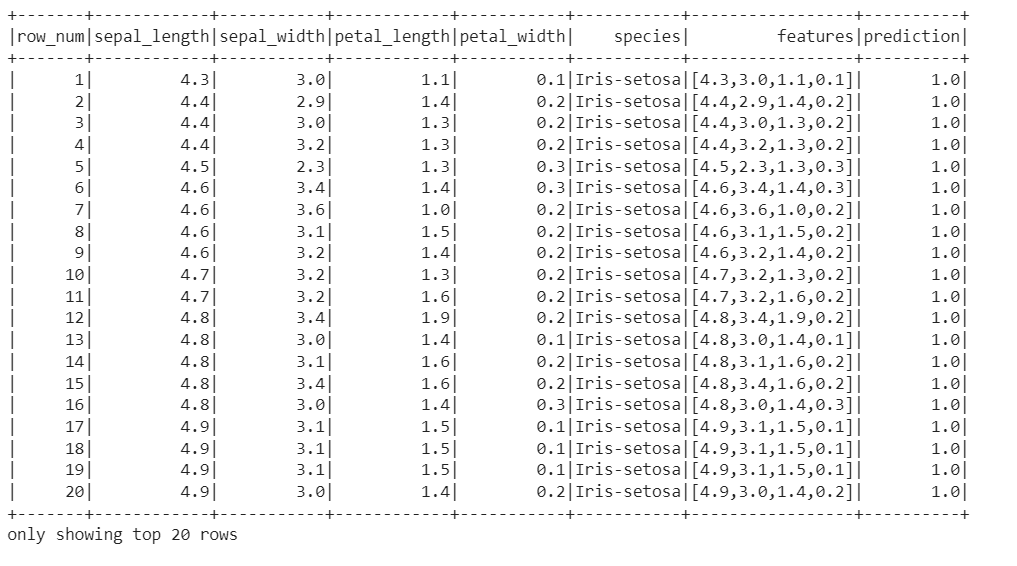
predictions = predictions.select( F.row\_number().over(Window.partitionBy().orderBy(predictions['sepal\_length'])).alias("row\_num"),"sepal\_length","sepal\_width","petal\_length","petal\_width","species","features","prediction")

predictions.show()

vecAssembler = VectorAssembler(inputCols=input\_columns, outputCol="test\_cen")

df\_kmeans = vecAssembler.transform(predictions).select('row\_num', 'features')

df\_kmeans.show()

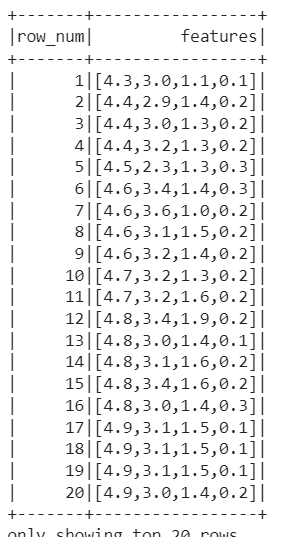


# vector assembler

vecAssembler = VectorAssembler(inputCols=input\_columns, outputCol="test\_cen")

df\_kmeans = vecAssembler.transform(predictions).select('row\_num', 'features')

df\_kmeans.show()



# estimation of silhouette using squared Euclidean

from pyspark.ml.clustering import KMeans

import numpy as np

from pyspark.ml.evaluation import ClusteringEvaluator

silhouette\_score=[]

evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='features',

                                metricName='silhouette', distanceMeasure='squaredEuclidean')

for i in range(2,10):

    KMeans\_algo=KMeans(featuresCol='features', k=i)

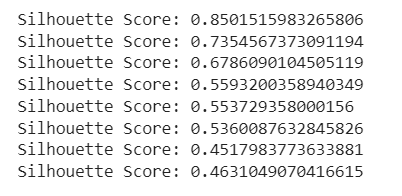
    KMeans\_fit=KMeans\_algo.fit(final\_data)

    output=KMeans\_fit.transform(final\_data)

    score=evaluator.evaluate(output)

    silhouette\_score.append(score)

    print("Silhouette Score:",score)



#Visualizing the silhouette scores in a plot

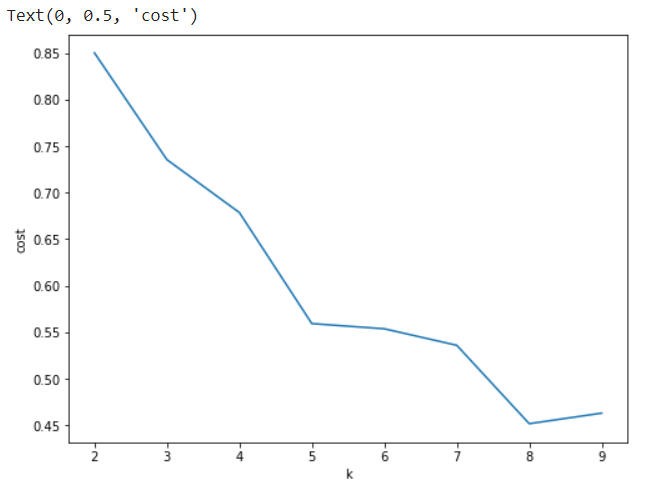
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1, figsize =(8,6))

ax.plot(range(2,10),silhouette\_score)

ax.set\_xlabel('k')

ax.set\_ylabel('cost')



# estimation of silhouette using cosine

from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator

silhouette\_score=[]

evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='features',

                                metricName='silhouette', distanceMeasure='cosine')

for i in range(2,10):

    KMeans\_algo=KMeans(featuresCol='features', k=i)

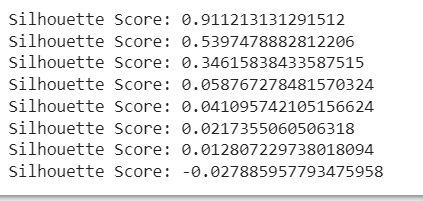
    KMeans\_fit=KMeans\_algo.fit(final\_data)

    output=KMeans\_fit.transform(final\_data)

    score=evaluator.evaluate(output)

    silhouette\_score.append(score)

    print("Silhouette Score:",score)

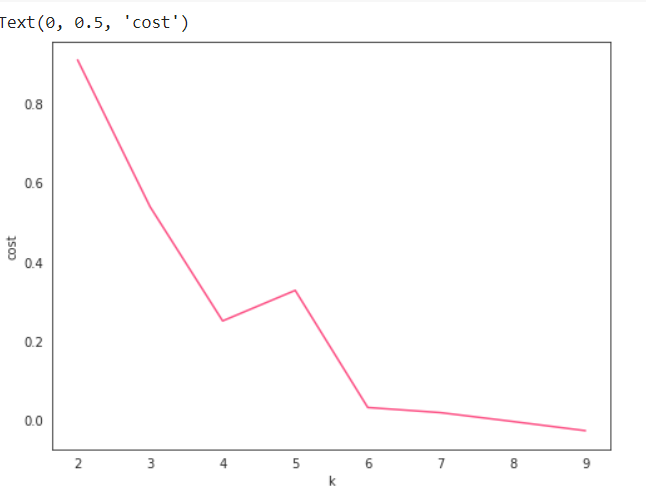


fig, ax = plt.subplots(1,1, figsize =(8,6))

ax.plot(range(2,10),silhouette\_score)

ax.set\_xlabel('k')

ax.set\_ylabel('cost')



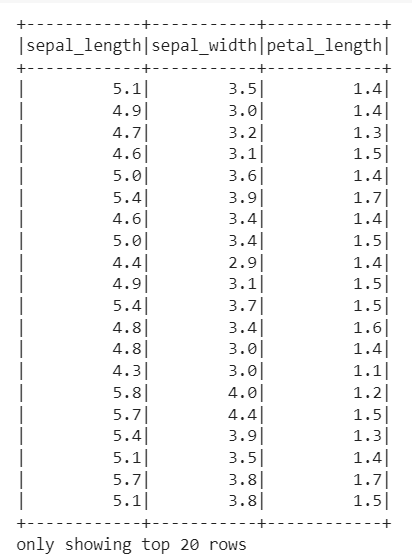
# 3D scatter plotting

data\_df = spark.read.csv("/content/drive/MyDrive/IRIS.csv",inferSchema =True,header = True)

data\_df = data\_df.na.drop()

data\_df1 = data\_df.drop('petal\_width','species')

data\_df1.show()



#data\_df1.columns

feature\_columns = ['id', 'sepal\_length', 'sepal\_width', 'petal\_length']

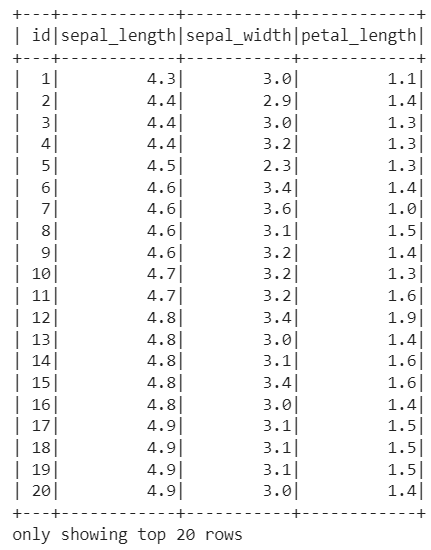
from pyspark.sql.window import Window

import pyspark.sql.functions as F

from pyspark.sql.functions import row\_number

data\_df1 = data\_df1.select( F.row\_number().over(Window.partitionBy().orderBy(data\_df1['sepal\_length'])).alias("id"),"sepal\_length","sepal\_width","petal\_length")

data\_df1.show()

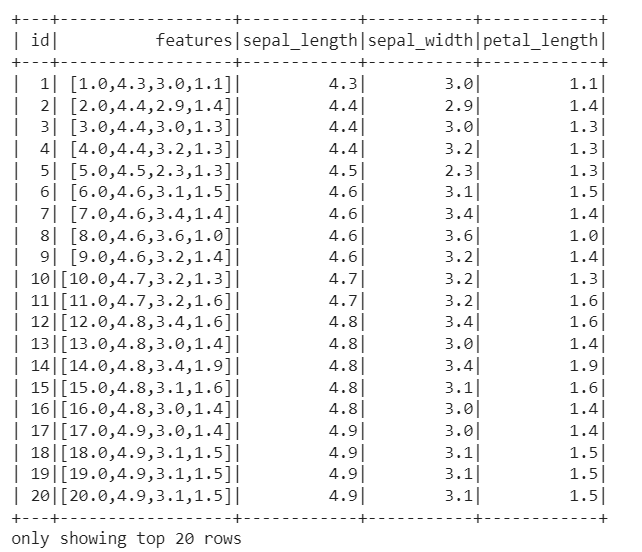


#vector Assembler

vecAssembler = VectorAssembler(inputCols=feature\_columns, outputCol="features")

df\_kmeans = vecAssembler.transform(data\_df1).select('id', 'features','sepal\_length','sepal\_width','petal\_length')

df\_kmeans.show()



# k-means

k = 10

kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")

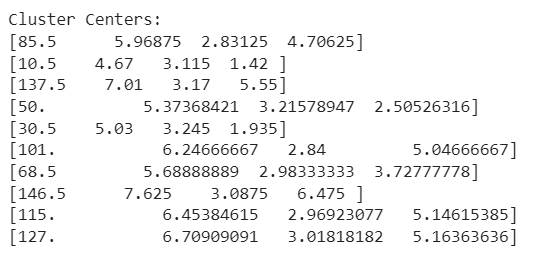
model = kmeans.fit(df\_kmeans)

centers = model.clusterCenters()

print("Cluster Centers: ")

for center in centers:

    print(center)



transformed = model.transform(df\_kmeans).select('id', 'prediction','sepal\_length','sepal\_width','petal\_length')

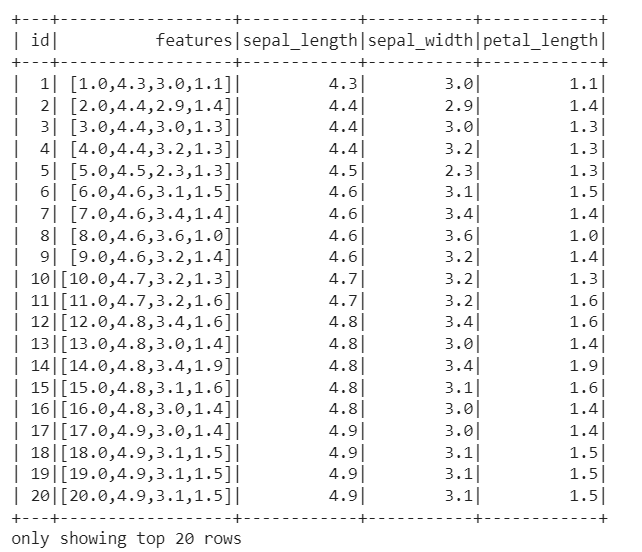
rows = transformed.collect()

from pyspark.sql import SQLContext

sqlContext = spark

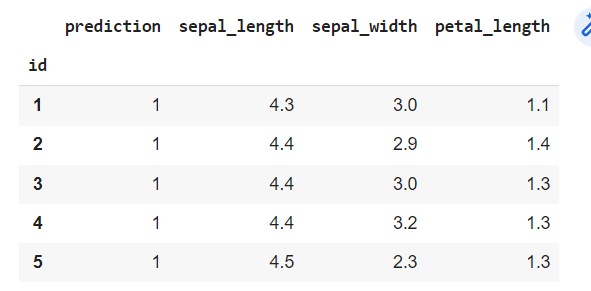
df\_pred = sqlContext.createDataFrame(rows)

df\_pred.show()



pddf\_pred = df\_pred.toPandas().set\_index('id')

pddf\_pred.head()



threedee = plt.figure(figsize=(12,10)).gca(projection='3d')

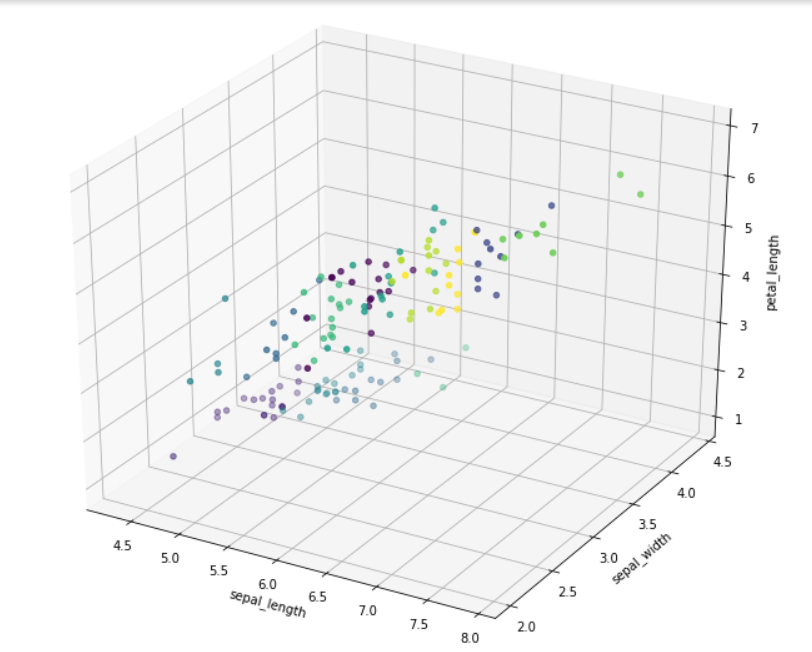
threedee.scatter(pddf\_pred.sepal\_length, pddf\_pred.sepal\_width, pddf\_pred.petal\_length, c=pddf\_pred.prediction)

threedee.set\_xlabel('sepal\_length')

threedee.set\_ylabel('sepal\_width')

threedee.set\_zlabel('petal\_length')

plt.show()



# K-means clustering upon Mall\_customers

!pip install pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("test").getOrCreate()

spark

# imports

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from pyspark import SparkContext

from pyspark.ml.clustering import KMeans

from pyspark.ml.feature import VectorAssembler

from pyspark.sql import SQLContext

%matplotlib inline

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.graph\_objs as go

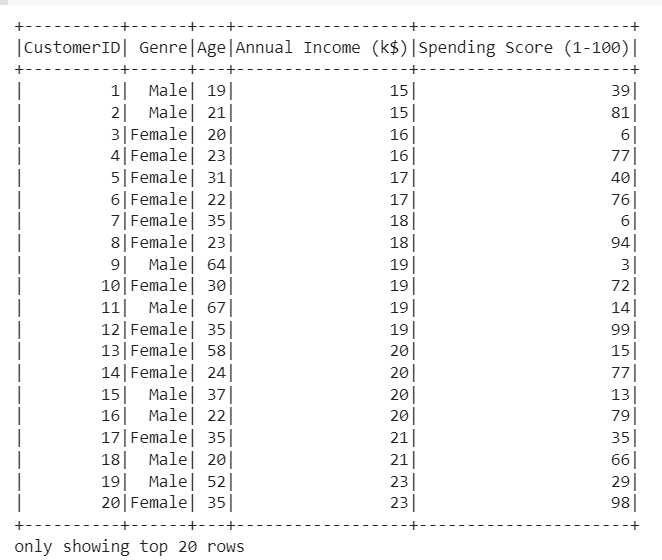
from plotly import tools

from plotly.subplots import make\_subplots

import plotly.offline as py

data\_df = spark.read.csv("/content/drive/MyDrive/Mall\_Customers.csv",inferSchema =True,header = True)

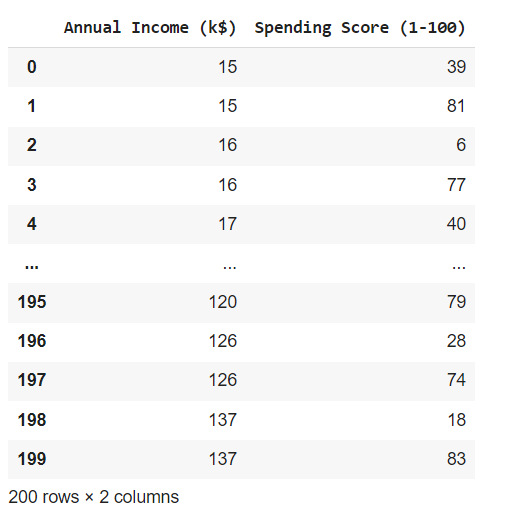
data\_df.show()



x = data\_df.select("Annual Income (k$)","Spending Score (1-100)")

x1= x.toPandas()

x1



#find the optimal number of clusters using elbow method

from sklearn.cluster import KMeans

WCSS = []

for i in range(1,11):

    model = KMeans(n\_clusters = i,init = 'k-means++')

    model.fit(x1)

    WCSS.append(model.inertia\_)

fig = plt.figure(figsize = (7,7))

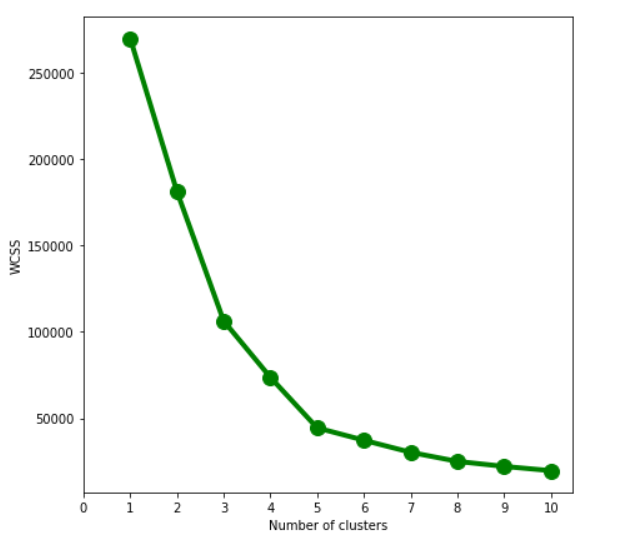
plt.plot(range(1,11),WCSS, linewidth=4, markersize=12,marker='o',color = 'green')

plt.xticks(np.arange(11))

plt.xlabel("Number of clusters")

plt.ylabel("WCSS")

plt.show()

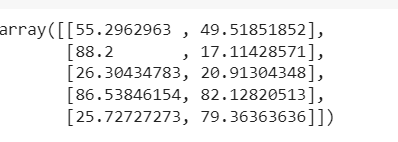


# since elbow forms at k=5 we select k value as 5

model = KMeans(n\_clusters = 5, init = "k-means++")

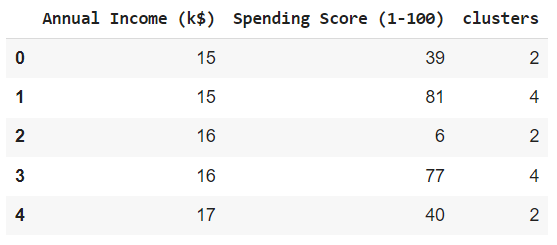
k\_means = model.fit(x1)

k\_means.cluster\_centers\_

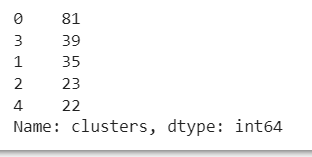


x1['clusters']  = k\_means.labels\_

x1.head()



x1['clusters'].value\_counts()



sns.scatterplot(x='Spending Score (1-100)',y='Annual Income (k$)',hue='clusters',data=x1)

