|  |  |  |  |
| --- | --- | --- | --- |
| **Sr No** | **Practical Names** | **Date** | **Sign** |
| 01. | PySpark Data Reading and Display Example. |  |  |
| 02. | Combining DataFrames with PySpark. |  |  |
| 03. | Collect, filter,map, MapReduce example using PySpark. |  |  |
| 04. | Creating a spark session using the configuration and Dataframe creation. |  |  |
| 05. | PySpark Word Count and Data Manipulation Example. |  |  |
| 06. | Real-Time Word Count with PySpark Streaming. |  |  |
| 07. | Creating a temporary view of dataframe to use sql query with Spark session. |  |  |
| 08. | Creating User defined function with spark session. |  |  |
| 09. | Pyspark using mlib library working with linear regression. |  |  |
| 10. | Pyspark using mlib library working with logistics regression. |  |  |
| 11. | Pyspark using mlib library working with Naive Bayes. |  |  |
| 12. | Pyspark using mlib library working with k mean clustering. |  |  |
| 13. | Case Study |  |  |

Practical no :- 1

from pyspark.sql import SparkSession

#initialize the spark

import pyspark

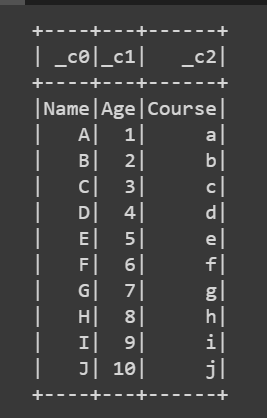
spark = SparkSession.builder.appName("Example 01").getOrCreate()

#read data

data\_df = spark.read.csv("students.csv")

data\_df.show()

output :-



Practical no 2:-

! pip install pyspark

#joining two dataframe

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example 02").getOrCreate()

#create data frame

data1 = [("Shubham",1),("Anil",2),("Raj",3)]

data2 = [("Shubham","Engineer"),("Anil","Teacher"),("Raj","Student")]

col1 = ["Name","RollNo"]

col2 = ["Name","Profession"]

#creating dataframe using spark

df1 = spark.createDataFrame(data1, col1)

df2 = spark.createDataFrame(data2, col2)

#join 2 dataframe

join\_df = df1.join(df2,"Name","inner")

print("Inner Function")

join\_df2=df1.join(df2,"Name","outer")

join\_df.show()

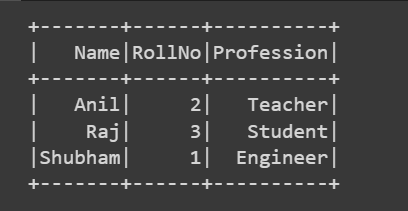
print("Outer Function")

join\_df.show()

#to stop sparkseesion

spark.stop()

output:-



Practical no 3:-

! pip install pyspark

#write a program on pyspark working with fucntion: collect, filter, map reduce, reduce, map

#collect function

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example 05").getOrCreate()

data = [1,2,3,4,5]

RDD = spark.sparkContext.parallelize(data)

collected\_data = RDD.collect()

print(collected\_data)

spark.stop()

output:-



#filter function

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example 06").getOrCreate()

data = [1,2,3,4,5,6,7,8,9]

RDD = spark.sparkContext.parallelize(data)

filter\_RDD = RDD.filter(lambda x: x%2 == 0)

filter\_data = filter\_RDD.collect()

print(filter\_data)

output:-



#map function

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example 07").getOrCreate()

data = [1,2,3,4,5,6,7,8,9]

RDD = spark.sparkContext.parallelize(data)

square\_RDD = RDD.map(lambda x:x\*x)

square\_data = square\_RDD.collect()

print(square\_data)

spark.stop()

output:-



#mapreduce function

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example 08").getOrCreate()

data1 = [1,2,3]

RDD = spark.sparkContext.parallelize(data1)

square\_RDD = RDD.map(lambda x:x\*x).reduce(lambda a,b: a+b)

print(square\_RDD)

spark.stop()

output:-



Practical no 4 :-

from pyspark.conf import SparkConf

from pyspark.sql import SparkSession

#create a spark configuration

spark\_conf= SparkConf()

spark\_conf.set("spark.app.name","dataframe example")

spark\_conf.set("spark.executor.memory","2g")

spark\_conf.set("spark.cores.max","2")

#create a spark seesion using the configuration

spark=SparkSession.builder.config(conf= spark\_conf).getOrCreate()

#sample data

data=[("rahul",25),("umang",26),("raj",24),("sam",30),("soham",23)]

columns=["name","age"]

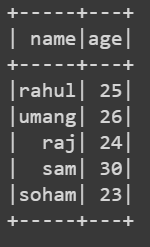
#create a dataframe from the sample data

df= spark.createDataFrame(data,columns)

df.show()

spark.stop()

output :-



Practical no 5:-

#write a program working with wordcount , distinct operation, flatmap & print number of character

from pyspark import SparkContext

conf= SparkConf().setAppName("WordCount Example")

sc= SparkContext.getOrCreate(conf = conf)

data=["apple orange","banana apple","orange banana","apple apple"]

RDD= sc.parallelize(data)

word\_counts= RDD.flatMap(lambda line: line.split(" ")).map(lambda word:(word,1)).reduceByKey(lambda a,b:a+b)

print("wordcounts")

print(word\_counts.collect())

# distinct operation

distinct\_word=RDD.flatMap(lambda lines: lines.split(" ")).distinct()

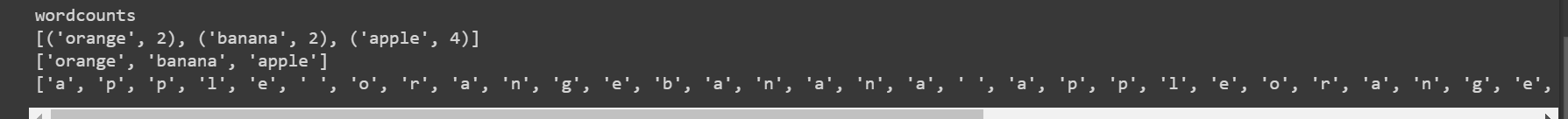
print(distinct\_word.collect())

#flatmap operation to create a list of characters

char\_list= RDD.flatMap(lambda line: list(line))

print(char\_list.collect())

output:-



Practical no 6:-

#write a program working with pyspark streaming using word reduce count

from pyspark import SparkContext

from pyspark.streaming import StreamingContext

sc.stop()

#create a spark context and streaming context

sc = SparkContext("local[2]","PysparkStreamingWordReduce")

ssc = StreamingContext(sc,1)

#create a socket stream to listen for data on local host

lines = ssc.socketTextStream("localhost", 9999)

#defining the window for 10seconds and sliding interval for 2second

ws = lines.window(10,2)

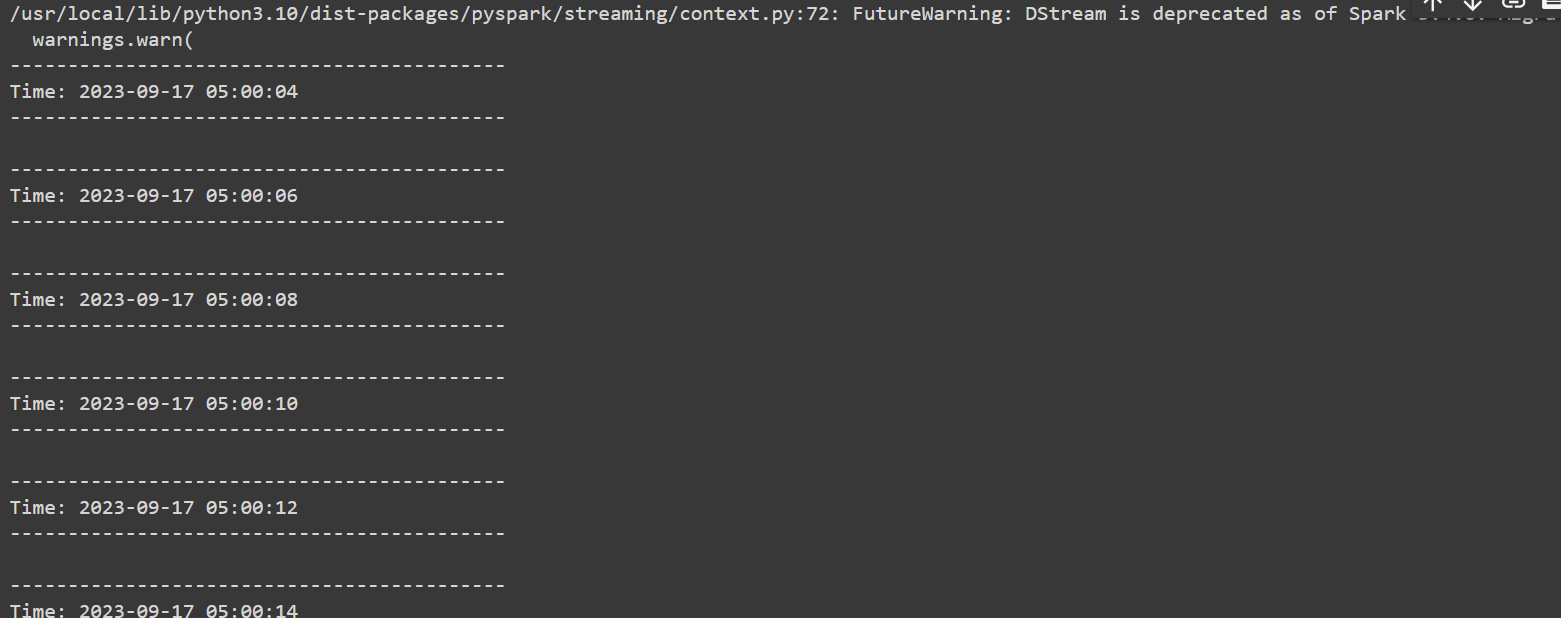
word\_count = ws.flatMap(lambda line:line.split(" ")).map(lambda word:(word,1)).reduceByKey(lambda x,y:x+y)

word\_count.pprint()

ssc.start()

ssc.awaitTermination()

output:-



Practical no 7

!pip install pyspark

from pyspark.sql import SparkSession

rom pyspark.sql import SparkSession

from pyspark.sql.functions import col

#create a spark seesion

spark = SparkSession.builder.appName("example1").getOrCreate()

#read the csv file

df = spark.read.csv("person.csv",header=True,inferSchema=True)

df.printSchema()

#create a temprorary view of the data frame to use sql query

df.createOrReplaceTempView("employee")

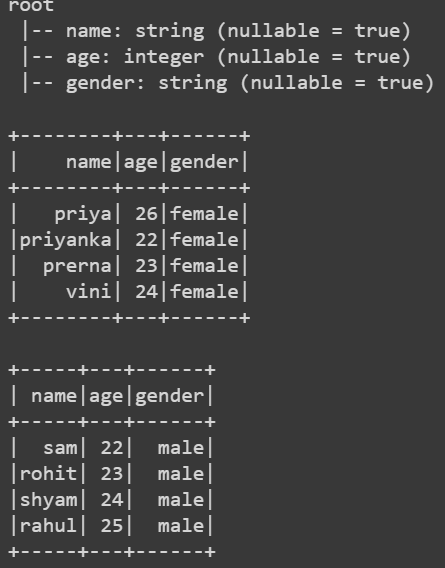
result = spark.sql("select \* from employee where age > 21 and gender = 'female'")

result.show()

result = spark.sql("select \* from employee where age > 21 and gender = 'male'")

result.show()

output:-



Practical no 8 :-

!pip install pyspark

from pyspark.sql import SparkSession

# using udf (user define function)

from pyspark.sql import SparkSession

from pyspark.sql.functions import udf

from pyspark.sql.types import StringType

#create a spark seesion

spark = SparkSession.builder.appName("example1").getOrCreate()

#read the csv file

df = spark.read.csv("person.csv",header=True,inferSchema=True)

df.printSchema()

# define a function that we need to use as a udf

def greet(name):

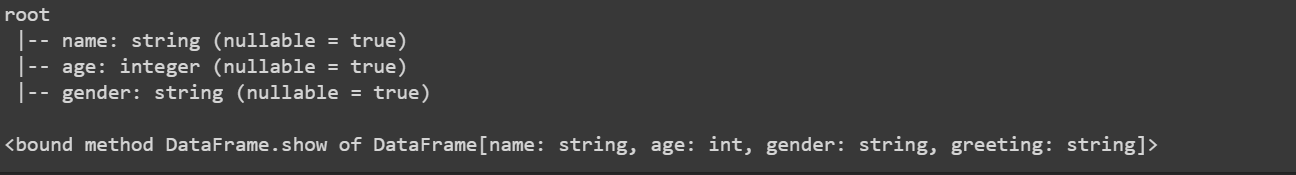
  return f"hello,{name}"

greet\_udf = udf(greet,StringType())

df\_greeting = df.withColumn("greeting",greet\_udf(df["name"]))

df\_greeting.show

output :-



Practical no 9:-

import pyspark

from pyspark.ml.regression import LinearRegression

from pyspark.ml.evaluation import RegressionEvaluator # based on datat we need to evaluate intercept,coefficient, accuracy and erros

spark = pyspark.sql.SparkSession.builder.appName("Linear Regression").getOrCreate()

data = spark.read.csv("data\_price.csv",header=True,inferSchema=True)

print("---Print the CSV File----")

data.show()

print("----Structure of the data---")

data.printSchema()

#print("----Print the columns of the csv file in the form of list----")

#data.columns

#vectors(VectorAssembler)=joining two columns(independent var) into one and creating new feature

print("")

from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=["age","Exper"],outputCol="Independent")

output=assembler.transform(data)

print("----Structure of the output variable---")

output.show()

#print("----Print the columns of the output var in the form of list----")

#output.columns

final\_data=output.select('Independent','Salary')

print("----Print the independent and dependent columns----")

final\_data.show()

#

train\_data,test\_data=final\_data.randomSplit([0.7,0.3])

regressor = LinearRegression(featuresCol="Independent",labelCol="Salary")

regressor = regressor.fit(train\_data)

regressor.coefficients

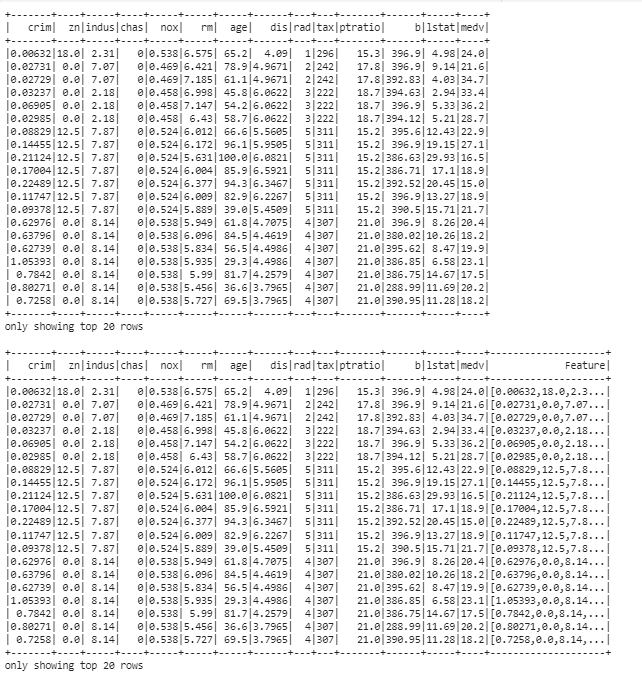
regressor.intercept

print("---prediction of salary with actual and expected----")

pred\_result = regressor.evaluate(test\_data)

pred\_result.predictions.show()

output :-

****

****

practical no 10:-

from pyspark.sql import SparkSession

from pyspark import SparkFiles

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

from pyspark.ml.feature import VectorAssembler

spark=pyspark.sql.SparkSession.builder.appName("Example 2").getOrCreate()

data=spark.read.csv("data.csv",inferSchema=True, header=True)

data.show()

data.columns

# Rename the columns for better readability

columns = ['id', 'diagnosis'] + [f'feature\_{i}' for i in range(1, 32)]

data = data.toDF(\*columns)

#Map 'M' (malignant) to 1 and 'B' (benign) to 0

data = data.withColumn("label", (data["diagnosis"] == "M").cast("integer")).drop("diagnosis")

feature\_columns = [f'feature\_{i}' for i in range(1, 25)]

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

data = assembler.transform(data)

train\_data, test\_data = data.randomSplit([0.8, 0.2], seed=42)

logistic\_regression = LogisticRegression(featuresCol="features", labelCol="label")

model = logistic\_regression.fit(train\_data)

coefficients = model.coefficients

intercept = model.intercept

print('Coefficients: ', coefficients)

print('Intercept: {:.3f}'.format(intercept))

predictions = model.transform(test\_data)

# AUC-ROC

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", labelCol="label")

auc = evaluator.evaluate(predictions)

# Accuracy, Precision, and Recall

multi\_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")

accuracy = multi\_evaluator.evaluate(predictions, {multi\_evaluator.metricName: "accuracy"})

precision = multi\_evaluator.evaluate(predictions, {multi\_evaluator.metricName: "weightedPrecision"})

recall = multi\_evaluator.evaluate(predictions, {multi\_evaluator.metricName: "weightedRecall"})

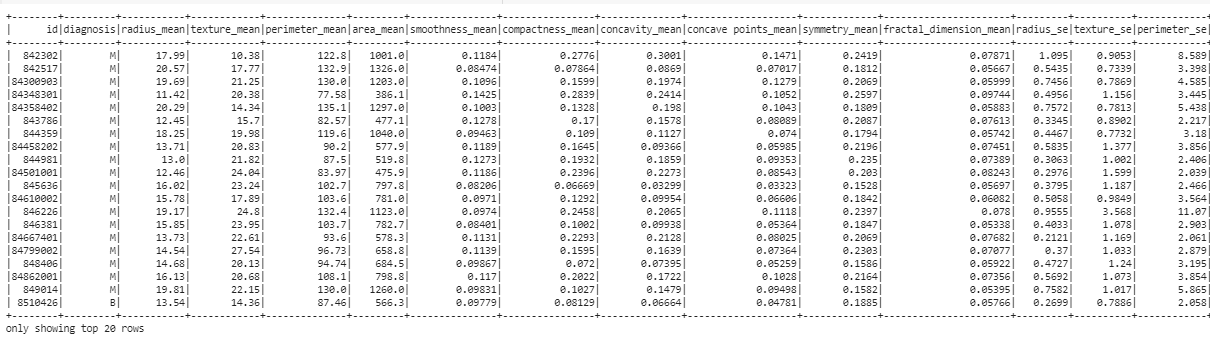
print(f"AUC-ROC: {auc:.4f}")

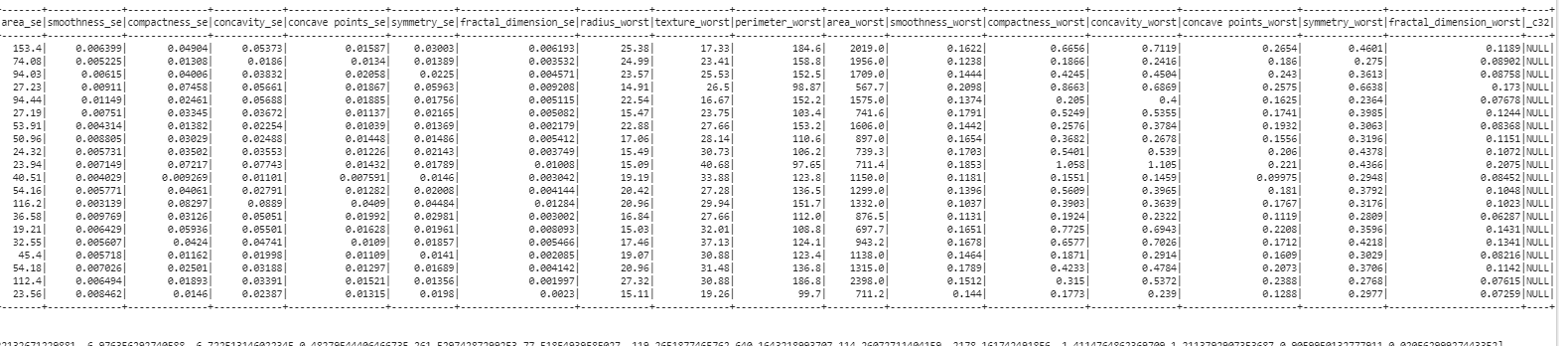
print(f"Accuracy: {accuracy:.4f}")

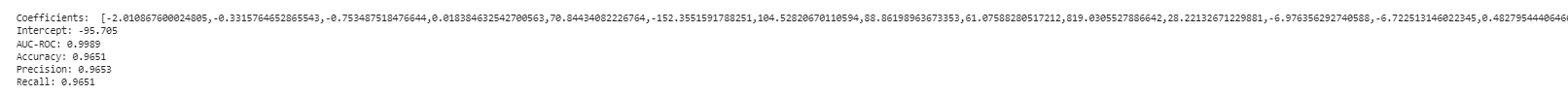
print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

output :-

****

****

****

Practical no 11 :-

#Naive Bayes

#Loading the library

from pyspark import SparkContext

from pyspark.sql import SQLContext

from pyspark.ml.classification import NaiveBayes

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.ml.feature import VectorAssembler

from sklearn.metrics import confusion\_matrix

from sklearn.datasets import load\_iris

import pandas as pd

#Preparing the data

iris=load\_iris()

df\_iris=pd.DataFrame(iris.data,columns=iris.feature\_names)

print(df\_iris.head())

df\_iris['label']=pd.Series(iris.target)

print(df\_iris.tail())

#Define the sql content and create

sc=SparkContext.getOrCreate()

SqlContext=SQLContext(sc)

data=SqlContext.createDataFrame(df\_iris)

data.printSchema()

#Combine the features of data and separate labels while using VectorAssembler

features=iris.feature\_names

va=VectorAssembler(inputCols=features,outputCol='features')

va\_df=va.transform(data)

va\_df=va\_df.select(['features','label'])

print("Combining the features into one column")

va\_df.show(3)

#Split the data into training and testing

(train,test)=va\_df.randomSplit([0.9,0.1])

#Prediction and accurancy

#Decision Tree Classifier by using naive bayes class and fit the model into train data

nb=NaiveBayes(smoothing=1.0,modelType='multinomial')

nb=nb.fit(train)

pred=nb.transform(test)

print("Prediction")

pred.show(7)

#predict the test data and check the accuracy matrix

evaluator=MulticlassClassificationEvaluator(predictionCol='prediction')

acc=evaluator.evaluate(pred)

print('Prediction accuracy',acc)

y\_pred=pred.select("Prediction").collect()

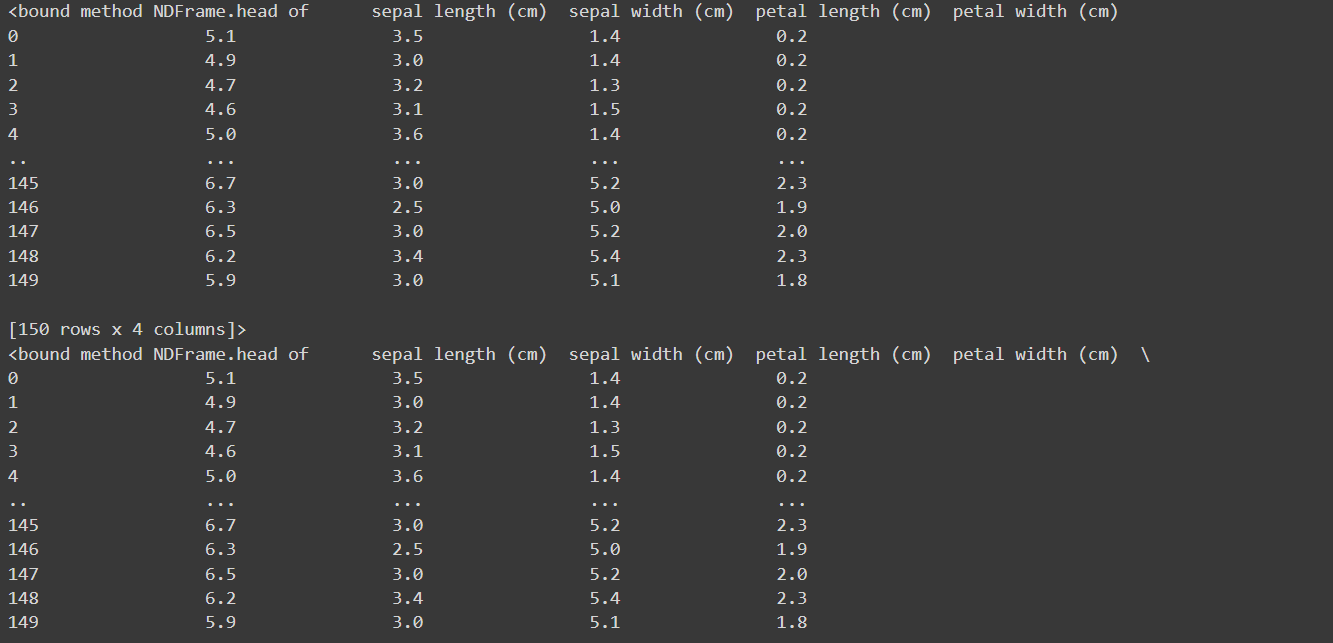
y\_orig=pred.select("label").collect()

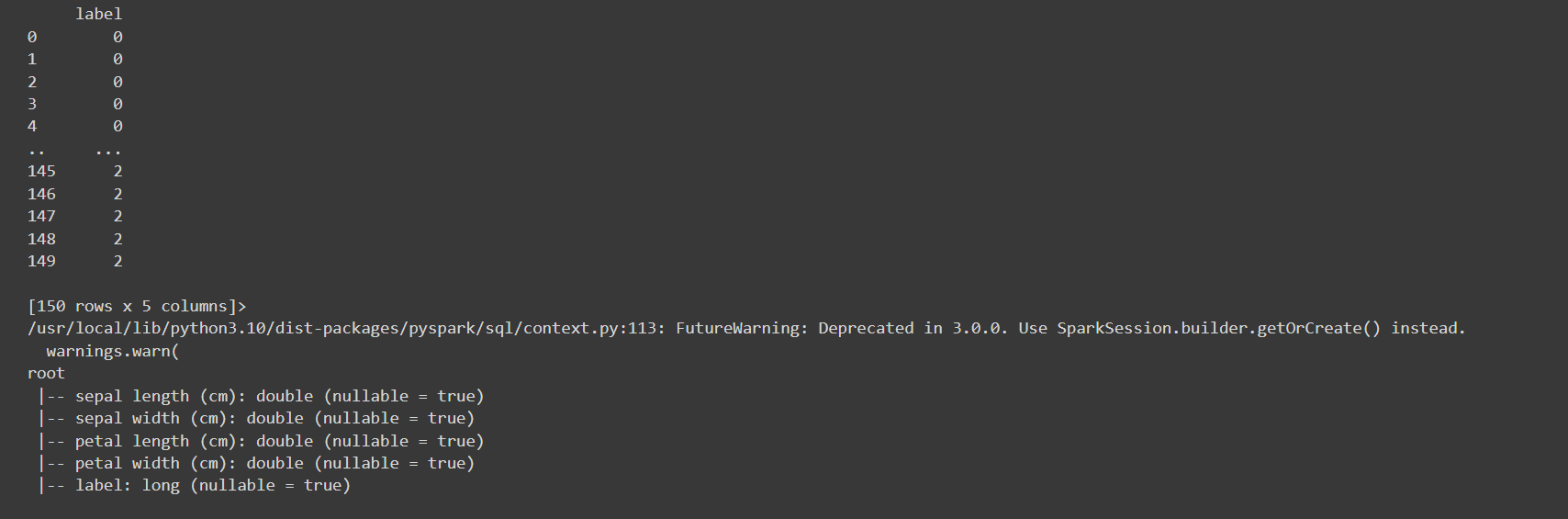
cm=confusion\_matrix(y\_orig,y\_pred)

print("Confusion Matrxi",cm)

sc.stop()

output :-





Practical no 12 :-

from pyspark.sql import SparkSession

from pyspark.ml.clustering import KMeans

from pyspark.ml.feature import VectorAssembler

import matplotlib.pyplot as plt

# Create a Spark session

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

# Load the Iris dataset

iris\_data = spark.read.csv("iris.csv", header=True, inferSchema=True)

# Select relevant features (attributes) for clustering

feature\_columns = ["sepallength", "sepalwidth", "petallength", "petalwidth"]

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

data = assembler.transform(iris\_data)

# Train a K-Means clustering model

kmeans = KMeans().setK(3).setSeed(1)  # Set the number of clusters (K) to 3

model = kmeans.fit(data)

# Get cluster centers

cluster\_centers = model.clusterCenters()

print("Cluster Centers:")

for center in cluster\_centers:

    print(center)

# Assign data points to clusters

predictions = model.transform(data)

# Visualize the clustering results

import pandas as pd

# Convert the PySpark DataFrame to a Pandas DataFrame for visualization

df = predictions.select("features", "prediction").toPandas()

# Plot the clustered data points

plt.scatter(df["features"].apply(lambda x: x[0]), df["features"].apply(lambda x: x[1]), c=df["prediction"], cmap="rainbow")

plt.xlabel("Sepal Length (cm)")

plt.ylabel("Sepal Width (cm)")

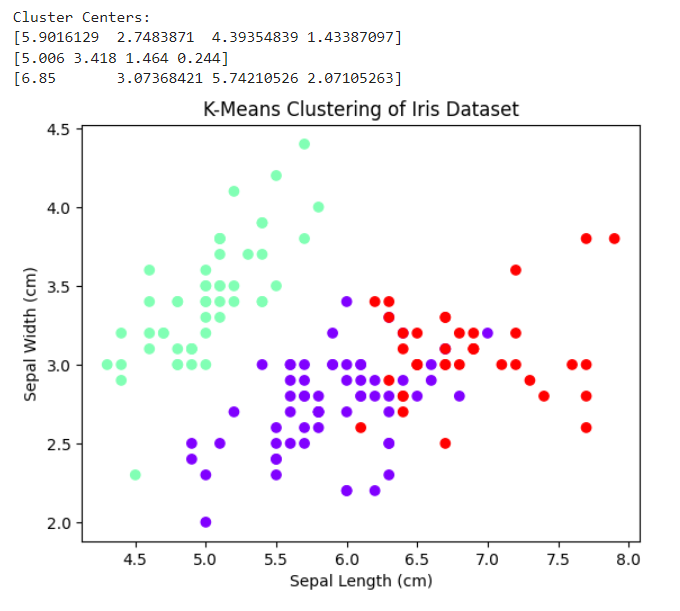
plt.title("K-Means Clustering of Iris Dataset")

plt.show()

# Stop the Spark session

spark.stop()

output :-

****