In [1]:	pip install pyspark
	Collecting pyspark Downloading pyspark-3.1.2.tar.gz (212.4 MB) 212.4 MB 70 kB/s s eta 0:00:01 212.4 MB
	59.2 MB 38.1 MB/s eta 0:00:05
	Created wheel for pyspark: filename=pyspark-3.1.2-py2.py3-none-any.whl size=212880768 sha256=fe3622 6ec06d21ff8662045c8609e7bcad227df1b0e9f2c3852605d0e5ba58db Stored in directory: /root/.cache/pip/wheels/a5/0a/c1/9561f6fecb759579a7d863dcd846daaa95f598744e71b 02c77
	Successfully built pyspark Installing collected packages: py4j, pyspark Successfully installed py4j-0.10.9 pyspark-3.1.2 WARNING: Running pip as root will break packages and permissions. You should install packages reliabl
	y by using venv: https://pip.pypa.io/warnings/venv Note: you may need to restart the kernel to use updated packages.
In [2]:	<pre>from pyspark.sql import functions as f import pandas as pd from pyspark.sql import DataFrameNaFunctions as DFna</pre>
	<pre>from pyspark.sql.functions import udf, col, when import matplotlib.pyplot as plt import pyspark as ps import os, sys, requests, json</pre>
	<pre>from pyspark.sql.functions import col,size,regexp_replace,lit from pyspark.ml.evaluation import RegressionEvaluator from pyspark.ml.recommendation import ALS</pre>
	<pre>from pyspark.ml.evaluation import RegressionEvaluator, MulticlassClassificationEvaluator from pyspark.ml.recommendation import ALS from pyspark.ml.tuning import CrossValidator, ParamGridBuilder</pre>
	<pre>from pyspark.ml import Pipeline from pyspark.sql import Row import numpy as np import math</pre>
	<pre>from pyspark.sql.functions import regexp_replace from pyspark.sql import SparkSession</pre>
	<pre>spark = SparkSession.builder.master("local[*]").config("spark.executor.memory", "70g").config("spark.dr iver.memory", "50g").config("spark.memory.offHeap.enabled", True).config("spark.memory.offHeap.size", "20 g").appName("sampleCodeForReference").getOrCreate()</pre>
In [3]:	<pre>sc = spark.sparkContext from pyspark.sql import SQLContext</pre>
	<pre>sqlContext = SQLContext(sc) pro=spark.read.csv('/input/data-science-for-good-careervillage/professionals.csv', header=True, quote=</pre>
	<pre>'"', sep=", ", multiLine=True) email=spark.read.csv('/input/data-science-for-good-careervillage/emails.csv', header=True, quote='"', s ep=",", multiLine=True) ques=spark.read.csv('/input/data-science-for-good-careervillage/questions.csv', header=True, quote='"'</pre>
	<pre>, sep=",", multiLine=True) match=spark.read.csv('/input/data-science-for-good-careervillage/matches.csv', header=True, quote='"', sep=",", multiLine=True)</pre>
	<pre>ans=spark.read.csv('/input/data-science-for-good-careervillage/answers.csv', header=True, quote='"', se p=",",multiLine=True) ans_score=spark.read.csv('/input/data-science-for-good-careervillage/answer_scores.csv', header=True, quote='"', sep=",",multiLine=True)</pre>
In [5]:	<pre>from pyspark.sql.functions import lit,row_number,col from pyspark.sql.window import Window</pre>
	<pre>w = Window().partitionBy(lit('a')).orderBy(lit('a')) ques = ques.withColumn("ques id", row number().over(w))</pre>
In [61:	<pre>pro = pro.withColumn("pro_id", row_number().over(w)) pro_ans1=spark.read.csv('/input/data-collaborative1/collaborative_label1.csv', header=True, quote='"',</pre>
	sep=",",multiLine=True,inferSchema=True) pro_ans0=spark.read.csv('/input/data-collaborative1/collaborative_label0.csv', header=True,quote='"', sep=",",multiLine=True,inferSchema=True)
	AnalysisException Traceback (most recent call last) <ipython-input-6-2d6f147e9804> in <module>> 1 pro ansl=spark.read.csv('/input/data-collaborative1/collaborative_label1.csv'. header=True.</module></ipython-input-6-2d6f147e9804>
	> 1 pro_ans1=spark.read.csv('/input/data-collaborative1/collaborative_label1.csv', header=True, quote='"', sep=",", multiLine=True, inferSchema=True) 2 pro_ans0=spark.read.csv('/input/data-collaborative1/collaborative_label0.csv', header=True, quote='"', sep=",", multiLine=True, inferSchema=True)
	/opt/conda/lib/python3.7/site-packages/pyspark/sql/readwriter.py in csv(self, path, schema, sep, enco ding, quote, escape, comment, header, inferSchema, ignoreLeadingWhiteSpace, ignoreTrailingWhiteSpace, nullValue, nanValue, positiveInf, negativeInf, dateFormat, timestampFormat, maxColumns, maxCharsPerC
	olumn, maxMalformedLogPerPartition, mode, columnNameOfCorruptRecord, multiLine, charToEscapeQuoteEsca ping, samplingRatio, enforceSchema, emptyValue, locale, lineSep, pathGlobFilter, recursiveFileLookup, modifiedBefore, modifiedAfter, unescapedQuoteHandling)
	<pre>735</pre>
	<pre>739</pre>
	<pre>1304 return_value = get_return_value(-> 1305</pre>
	<pre>for temp_arg in temp_args: /opt/conda/lib/python3.7/site-packages/pyspark/sql/utils.py in deco(*a, **kw) # Hide where the exception came from that shows a non-Pythonic</pre>
	<pre># JVM exception message> 117</pre>
	<pre>AnalysisException: Path does not exist: file:/kaggle/input/data-collaborative1/collaborative_label1.c sv</pre>
	<pre>pro_ques_final = pro_ans1.union(pro_ans0)</pre>
In []:	pro_ques_final
In []:	<pre>pro_ques_final=pro_ques_final.na.drop(subset=["check"])</pre>
In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2])</pre>
In []:	
In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2])</pre> <pre>from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01, coldStartStrategy="drop", implicitPrefs=False, userCol="pro_id", itemC ol="ques_id", ratingCol="check")</pre>
In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data</pre>
<pre>In []: In []:</pre>	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data predictions = model.transform(pro_ques_final)</pre> pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select('chec')
<pre>In []: In []: In []:</pre>	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data predictions = model.transform(pro_ques_final) pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select('check', 'pred')</pre> pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select
<pre>In []: In []: In []: In []:</pre>	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParame=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data predictions = model.transform(pro_ques_final) pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select('check','pred') pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select ('pro_id', 'ques_id', 'check', 'pred', 'prediction') from pyspark.sql.types import DoubleType pred = pred.withColumn("pred", pred["pred"].cast(DoubleType())) from pyspark.ml.evaluation import MulticlassClassificationEvaluator evaluator = MulticlassClassificationEvaluator (labelCol="check", metricName="accuracy", predictionCol="pred")</pre>
<pre>In []: In []: In []: In []:</pre>	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC</pre>
<pre>In []: In []: In []: In []:</pre>	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id",itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data predictions = model.transform(pro_ques_final) pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select('check','pred') pred final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select ('pro_id', 'ques_id', 'check', 'pred', 'prediction') from pyspark.sql.types import DoubleType pred = pred.withColumn("pred", pred("pred").cast(DoubleType())) from pyspark.ml.evaluation import MulticlassClassificationEvaluator evaluator = MulticlassClassificationEvaluator(labelCol="check",metricName="accuracy",predictionCol="pred") accuracy_val = evaluator.evaluate(pred) print(f"Accuracy:{accuracy_val*100:.5f)%") def recommendations_for_pro(pro_id): print('\ninfo of Professional: ') pro.filter(pro.pro_id==pro_id).drop('pro_id').show()</pre>
<pre>In []: In []: In []: In []:</pre>	<pre>(training, test) = pro_ques_final.randomSplit((0.8, 0.2)) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01, coldStartStrategy="drop", implicitPrefs=False, userCol="pro_id", itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data predictions = model.transform(pro_ques_final) pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('chec k', 'pred') pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select ('pro_id', 'ques_id', 'check', 'pred', 'prediction') from pyspark.sql.types import DoubleType pred = pred.withColumn("pred", pred("pred").cast(DoubleType())) from pyspark.ml.evaluation import MulticlassClassificationEvaluator evaluator = MulticlassClassificationEvaluator(labelCol="check", metricName="accuracy", predictionCol="pred") accuracy_val = evaluator.evaluate(pred) print(f"Accuracy:(accuracy_val*100:.5f)%") def recommendations for pro(pro_id): print('\alifo of Professional: ') pro.filter(pro.pro_id=pro_id).drop('pro_id').show() df=userRecs.filter(userRecs.pro_id=pro_id) x=df.select('recommendations').collect() df = sc.parallelize(x(0](0)).toDf(('id_ques', 'value')) df=df.join(ques,ques.ques id=df.id_ques).select('questions_id','questions_title','questions_body')</pre>
<pre>In []: In []: In []: In []:</pre>	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data predictions = model.transform(pro_ques_final) pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('chec k', 'pred') pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select ('pro_id', 'ques_id', 'check', 'pred', 'prediction') from pyspark.sql.types import DoubleType pred = pred.withColumn("pred", pred("pred"].cast(DoubleType())) from pyspark.ml.evaluation import MulticlassClassificationEvaluator evaluator = MulticlassClassificationEvaluator(labelCol="check", metricName="accuracy", predictionCol="pred") accuracy_val = evaluator.evaluate(pred) print("Ancuracy: (accuracy_val*100:.5f)%") def recommendations_for_pro(pro_id): print('Namfo of Professional: ') pro.filter(pro.pro_id==pro_id).drop('pro_id').show() df=userRecs.filter(userRecs.pro_id==pro_id) x=df.select('recommendations').collect() df = sc.parallellelize(x[0][0]).toDf(['id_ques', 'value'])</pre>
In []: In []: In []: In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC ol="ques_id", ratingCol="check") model = als.fit(training) # Evaluate the model by computing the RMSE on the test data predictions = model.transform(pro_ques_final) pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('check','pred') pred final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select ('pro_id', 'ques_id', 'check', 'pred', 'prediction') from pyspark.sql.types import DoubleType pred = pred.withColumn("pred", pred("pred").cast(DoubleType())) from pyspark.ml.evaluation import MulticlassClassificationEvaluator evaluator = MulticlassClassificationEvaluator(labelCol="check",metricName="accuracy",predictionCol="pre d") accuracy_val = evaluator.evaluate(pred) print(f"Accuracy:(accuracy_val*100:.5f)%") def recommendations_for_pro(pro_id): pro.filter(pro.pro_id=pro_id).drop('pro_id').show() df= sc.para[lealize(x[0][0]).toDf('id_ques','value')) df= sc.para[lealize(x[0][0]).toDf('id_ques','value')) df= sc.para[lealize(x[0][0]).toDf('id_ques','value')) pro.filter(pro.pro_id=pro_id) return df def add_recommendations_ques(ques_id): proRecs=model.recommendations_id==ques_id).select('ques_id').collect())[0][0]</pre>
In []: In []: In []: In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS (maxHer=2, regparam=0.01, coldStartStrategy="drop", implicitPrefs=False, userCol="pro_id", itemC ol="ques_id", ratingCol="check")</pre>
In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxIter=2, regParam=0.01, coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC col="ques_id", ratingCol="check") model = als.fit(training) * Evaluate the model by computing the RNSE on the test data predictions = model.transform(pro_ques_final) pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('chec X','pred') pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('pro_id', 'ques_id', 'check', 'pred', 'prediction') from pyspark.ml.evaluation.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select from pyspark.ml.evaluation import MulticlassClassificationEvaluator evaluator = MulticlassClassificationEvaluator('abelCol="check",metricName="accuracy",predictionCol="pred") accuracy_val = evaluator.evaluate(pred) print("Accuracy: (accuracy_val*100:.5f)\$") def recommendations for pro(pro_id).drop('pro_id').show() df=userRecs.filete('precommendations').collect() df = sc.parallelize(suesRecs.pro_id=pro_id) xdf.select('recommendations').collect() df=sc.parallelize(suesRecs.pro_id=pro_id) dr=proRecs=model.recommendforAhiltens(10) id=p.paravy(ques.filet(precommendations').collect() df=proRecs=model.recommendstons').collect() df=proRecs=model.recommendstons').collect() df=proRecs.filete('precomendations').collect() df=proRecs.fil</pre>
In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationSvaluator, MulticlassClassificationEvaluator ols = Mis(maxTer=2, regParam=0.0], coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemC ol="ques_id", ratingCol="check"]</pre>
In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator als = ALS(maxTer=2, regParam=0.0], coldStartStrategy="drop", implicitFrefs=False, userCol="pro_id", itemC col="ques_id", ratingCol="check"]</pre>
In []: In []: In []: In []:	<pre>(training, test) = pro_ques_final.randomSplit([0.8, 0.2]) from pyspark.ml.evaluation import SinaryClassificationEvaluator, MulticlassClassificationEvaluator</pre>
In []: In []: In []: In []:	<pre>training, best) = pro_ques_final.randomSpitt((0.8, 0.2)) from pyspack.ml.evaluation import BinaryClassificationTvaluator, MulticlassificationTvaluator also = Ald facation=0.01, old discretered and project of projec</pre>
In []: In []: In []: In []:	<pre>training, text) = pro_ques_final.randomSplit([0.8, 0.2]) trom pysperk.ml.evaluation.import SinaryClassificationEvaluator, NulticlassClassificationEvaluator als = AlS (maxiser2, responsed.01, coldstarStrategy="drop".implicitEvefo=False,userCol="pro_id", itself old-"ques_id", ratingCol-"beck") model = als.firstraining) # svaluer the model by computing the AMSES on the test data presidentions = radel.transform(pro_ques_final) predepredictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "i").otherwise("0")).select('check', 'pred') predepredictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "i").otherwise("0")).select('check', 'pred') predefinalepredictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "i").otherwise("0")).select('pre_id', 'ques_id', 'ques_id').prediction') **compression.ed.column('pred', 'pred').otes((bobleType())) **compression.ed.column('pred', pred' pred').otes((bobleType())) **print('Abcuracy (accuracy_val'=00.3f)*) **print('Abcurac</pre>
In []:	<pre>training, test) = pro_quee_final.randomSp.ti([0.8, 0.2]) from pyepack will evaluation import SinaryChassificationSvaluator, MulticleanChassificationSvaluator as = ASSESTED-2, response-0.01, colostartStrategy="crop", implicitPrets-False.userCol="pro_id", itemCol="quee_id", resingCol="fineCol", resingCol="fineCol", resingCol="fineCol", resingCol="fineCol", resingCol="fineCol", resingCol="fineCol", resingCol="fineCol", resingCol="fineCol", response-final) proce_predictions_withColumn("pred", f.wheaff.col("prediction") > 0.5, "1";.otherwise("0")).select("predictions.withColumn("pred", f.wheaff.col("prediction") > 0.5, "1";.otherwise("0")).select("predictions.withColumn("pred", f.wheaff.col("prediction") > 0.5, "1").otherwise("0")).select("predidtions.withColumn("pred", pred").pred") from pyepack.adl.types.import.CoulbType pred = pred.withColumn("pred", pred("pred").cast(CoulleType(!))) from pyepack.adl.types.import.CoulbType pred = pred.withColumn("pred", pred("pred").cast(CoulleType(!))) from pyepack.adl.types.import.CoulbType pred = pred.withColumn("pred", pred("pred").cast(CoulleType(!))) from pyepack.adl.types.import.CoulbType accuracy_val = evaluator = NulticleasCleasCleasCleasTiceTionCol="pred") accuracy_val = evaluator = NulticleasCleasCleasTiceTionCol="pred") accuracy_val = evaluator = NulticleasCleasTiceTionCol="pred") accuracy_val = evaluator = NulticleasCleasTiceTionCol="pred") accuracy_val = evaluator = NulticleasCleasCleasTiceTionCol="pred") accuracy_val = evaluator = NulticleasCleasTiceTionCol="pred") accuracy_val = evaluator = NulticleasCleasTiceTionCol=TiceTionCol=TiceTionCol=TiceTionCol=TiceTionCol=TiceTionC</pre>
In []:	<pre>tranning, besti = pro_ques_final.randem&plit([0.8, 0.2]) from pyspack.al.evaluation import SharpClassificationDealuator, WiltiolessClassificationDealuator als = ALD(seafter), resplacement() peddataStrategy="deup", amplicitPref="False.userCol="pro_id", itemS alt="ques_id", ranningCol="pro_id", itemS alt="ques_id", item</pre>
In []:	<pre>trowings, teat = pro_quan_final.rendomopliti(0.5, 0.2.) from pypark.ml.evaluation import MinaryDisacticationNowlustor, MiticlassClassificationNowlustor, Science 1. Asidomathare2, respectations. 2. delication of the season of the seaso</pre>
In []:	<pre>troining, test) = pro_quee_final.rendom@poit((0.8, 0.2)) from pympack.ml.evaluation import DiracyClassificationTvaluator. MiticlassClassificationTvaluator als = AlSimanic=-7, replacam=0.TlocdofartGiratepy="doop",implicitFrefa=False,userCol="pro_d", item@ ol="coust dir_acimpol="chempion" the RMMS on the feet data predictions = model.transform(pro_quee_final) predipredictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('chem predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('chem pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('chem pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('pred_cion') pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "l").otherwise("0")).select('pred_cion') pred_final=pred_pro_ind('pred_cion') pred_final=pred_pro_ind('pred_cion') pred_final=pred_pro_ind('pred_cion') pred_final=pred_pro_ind('pred_cion') pred_final=pred_pro_ind('pred_cion') pro_final=pro_ind('chempion') pro_final=pro_ind('pred_cion') print('Abscramp('geometry_vyl')100.5f)*) def model_final=pro_ind('report_ind('pred_cion')) pro_final=pro_ind('report_ind('pred_cion')) pro_final=pro_ind('pred_cion') pro_final=pro_ind</pre>
In []:	<pre>(recoining, test) = pro_quag_timil_random@pits(10.8, 0.2)] from pyspark.ml.evaluation import minoryclassicionicofevaluator, Multiclassicionicofevaluator als - Abditeative**), replaced="10", coldDestOrategy="domp", implicitive** also destructions and the property of the NEW on the rest data productions and the model to amporting the NEW on the rest data productions and the import minor the new force of the production of the new force of th</pre>
In []:	training, test) " propagatinal readendedit((0.0, 0.2)) from pyspark.ml.evaluation import BinaryClassificationSvaluator, MustinicacClassificationTvaluator also a Modification-2, regionard-10, on distantionsings-Mospy, implicitive-Febbus, section-100, feed also a Modification-2, regionard-10, on distantionsings-Mospy, implicitive-Febbus, section-100, feed in La. Fitterining # Destruct of a model Pyromagating the SOOT on the cest data predictions = model transforatory_move_limit) # Destructions = model transforatory_move_limit) # Productions on the model region of the cest data # Productions on the model region of the cest data # Productions on the model region of the cest data # Productions on the cest data for the cest data # Production of the model region of the cest data # Production of the model region of the cest data # Production of the model region of the cest data for the cest data # Production of the model region of the cest data for
In []:	(tealning, test) " pro ques final, randombplis(), 8, 0.2() from pympack.ml.evaluation import filmaryClass(ficationNvaluator, Millioteasclass(ficationNvaluator) also "Authorators"), dephases (illustrators also "Authorators"), dephases (illustrators"), deph
In []: In []:	Treatments, test = "sec_question.remonspets(1.0, 0.2)
In []: In []:	Prom pyspak.mi.orgination import BinaryClassicLectionDystater, NoticlassicLassicLectionDystates are national content in the procession of
In []: In []:	Treating tests = pos_quer_final.rendomp(it()0.6, 0.11) From pyppark.mi.orxivation import denotytic professionation () Multiples () Professionation () Colorate () () Co
In []: In []:	(custing, case = pro game_trantroudrospitis(5.4, 0.21)) From propert mi_evaluation import BinaryConstrictionProperties (
In []: In []:	Comparation Properties Final Engineering Comparation Properties Propert
In []: In []:	Transmission of the property o
In []: In []:	Compagned Accordance The Communication of the Communication