**pyspark安装**

pip install pyspark

pip install pyspark[sql,ml,mllib] #安装pyspark sql,ml,mllib依赖

**三种数据结构**

RDDs (Resilient Distributed Datasets)

DataFrames SQL-like structured datasets with query operations

Datasets-A mixture of RDDs and DataFrames

Spark Configuration

**查看所有设定值**

spark.sql("SET -v").select("key", "value").show(n, truncate=False)

**查看特定设定值**

spark.conf.get("KEYNAME")

**修改设定值**

sqlContext.sql("set KEYNAME=VALUE")

**动态资源分配相关**

#适用于潮汐数据处理，应对数据处理量不稳定的情形，如流式数据或高峰SQL查询，使得sparkdriver可以动态地申请更多/更少的计算资源

spark.dynamicAllocation.enabled true

spark.dynamicAllocation.minExecutors 2

spark.dynamicAllocation.schedulerBacklogTimeout 1min

spark.dynamicAllocation.maxExecutors 20

spark.dynamicAllocation.executorIdleTimeout 2min

**Spark Executor内存管理**

#默认内存分配：60% execution memory, 40% storage memory, 300M reserved memory

# execution memory和storage memory可基于需求互相转换,reserved memory防内存溢出

spark.memory.fraction 0.6 #execution memory占比

spark.driver.memory 1g #driver内存，可增大以应对大collect()

spark.shuffle.file.buffer 1MB

**数据切分处理**

spark.sql.files.maxPartitionBytes 128MB #切得太小会增加总coordinating cost，得不偿失

spark.sql.shuffle.partitions 200 #小切分数量适用于小数据

Spark Session

**启动spark session**

import findspark

findspark.init() #寻找可用spark集群的环境变量

创建SparkSession对象 (PySpark 3+以后版本)

from pyspark.sql import SparkSession

spark= SparkSession.builder.appName("Spark Application Name").getOrCreate()

sc=spark.sparkContext

创建SparkContext对象 (PySpark 3+以前版本)

from pyspark import SparkContext

sc = SparkContext()

**关闭spark session**

sc.stop()

**查看spark session版本**

sc.version

**查看spark标准设置**

from pyspark.conf import SparkConf

conf = SparkConf()

print(conf.toDebugString())

**查看当前spark设置**

sc.sparkContext.getConf().getAll()

**查看主节点**

sc.master

RDD

RDD不能切片

**创建RDD**

sc.parallelize(local\_obj)

sc.textFile(FILE.txt)

**Transform & Action**

Transform类函数返回新的RDD，是惰性函数

Action类函数返回一个计算结果，是即时执行的

**常见RDD Transform**

RDD.map(func) #1对1的transform

RDD.flatMap(func) #1对0或多的transform

RDD.reduceByKey(func) #reduce操作如.reduceByKey(lambda x, y: x + y)，前置RDD的元素需为一键值对如(key,value)，可先通过map来改变RDD元素结构

RDD.sortByKey() #按键排序，RDD的元素需为一键值对如(key,value)

RDD.filter(boolean\_func) #过滤data中的元素

RDD.distinct #去重

**常见RDD Action**

RDD.collect() #将内容全部返回给SparkDriverMemory并打印

RDD.count() #返回RDD中元素个数

RDD.take(n) #返回RDD中前n个元素

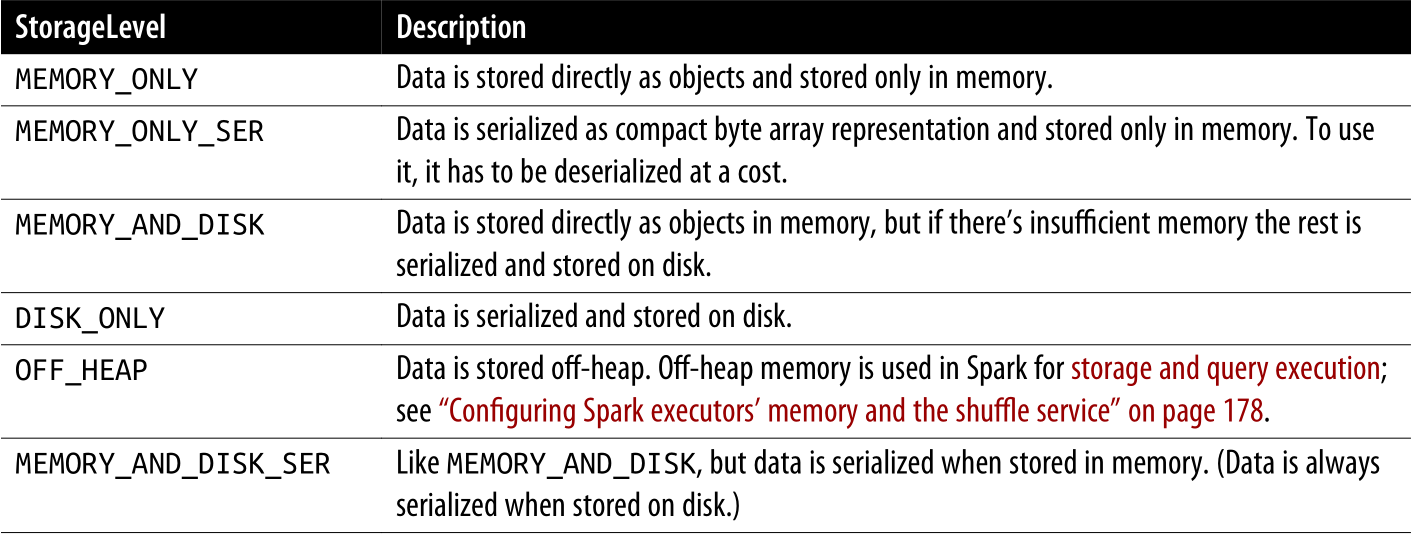
RDD.reduce #reduce操作

RDD.countByValue() #value\_counts()

**RDD暂存**

RDD.cache() #将RDD或计算结果存入内存以便后面调用

RDD.persist(\*StorageLevel.XXX) #同cache()，但可指定将RDD存入XXX



#数据在spark中都是分partition存储/计算的，cache()只会将第一次计算到的partition存起来，e.g. RDD.take(1)由于只用到了第一个partition，仅有第一个partition会被cache。在后续计算时，再计算并cache那些未用到过的partition

#partition是存储的基本单位，一个partition只能被完整而非部分地cache

DataFrame

import pyspark.sql.functions as f

**创建&导入&导出**

df =spark.read.text(path) #读取txt,csv

df =spark.read.parquet(foler\_path) #读取批文件(文件夹下有多个数据文件)

df =spark.read.json("customer.json") #读取json

df.toJSON() #转JSON

df.toPandas() #转pandas.DataFrame

df.write.save("FILE.json",format="json") #保存JSON

df.write.parquet("FILE.parquet") #保存parquet

spark.createDataFrame([(一行数据),…],列名list) #创建DataFrame1

spark.createDataFrame([ROW1,ROW2,…] ,列名list) #创建DataFrame2，ROW对象

row=Row(value1,value2,…) #创建一个Row对象

**基础信息**

df.show(n=20,\*truncate=True) #前20行(无返回),truncate控制显示截断

df.take(n) 或 df.limit(n) #前n行(返回一个DataFrame)

df.first() #第1行

df.count() #行数

df.printSchema() #打印表结构

df.cache() #结果加载到内存

df.columns #列名

df.collect() #返回全表并加载至内存

df.distinct(), df.A.distinct() #唯一值，去重

df.limit(10).toPandas().head()

**索引列**

df.A

df[‘A’]

f.col(‘A’)

**生成序号列**

from pyspark.sql.functions import monotonically\_increasing\_id

df.select('value',monotonically\_increasing\_id().alias('idx'))

**SQL式函数**

df.select(‘col1’,’col2’) #筛选列

df.select(‘col1’,’col2’,bool\_list) #筛选行

df.select(col(‘A’).alias(‘Name’)) #创建别名

df.select()

df.where(‘sql条件’) #等同于df.filter()

df.limit(n) #前n行

df.orderBy(col(‘A’), ascending=False) #排序

df.orderBy(["age","city"],ascending=[0,1]) #多列排序

df.groupBy([col(‘A’),…])

df.agg(聚合函数) #如DF.groupBy(‘Col1’).agg(sum(‘Col2’))

df.sort(col(‘A’).desc()) #排序

df.union(df) #行合并两个df

df.join(df2,on=[key1,key2],how=’inner’/’outer’/’left’/’right’) #表链接

df.select(‘A’, f.when(condition, value1).otherwise(value2)) #CASE WHEN，可用于创建列

df.select(df.A.between(value1,value2)) #BETWEEN函数

df.select(f.count(‘A’).alias(‘count’)) #计数

**行筛选**

df.filter(‘sql条件’)

df.where(‘sql条件’)

df[条件]

**成员运算**

df[df.A.isin(value1,value2)] #不用list

**创建列**

df.withColumn("newcol",func&transform&df列)

df.select(\*,lit(‘value’)) #创建一个常值列

df.withColumn("newcol",f.when(condition1,value1).when(condition2,value2).…) #条件赋值

**删除列**

df = df.drop("Col1", "Col2")

**列筛选**

df.select(’Col1’,’Col2’)

**列重命名**

df.withColumnRenamed(‘old\_name’,’new\_name’)

**聚合函数**

**import pyspark.sql.functions as f**

f.count() #计数

f.sum() #求和

f.max() #最大值

f.min() #最小值

f.avg() #均值

**字符串处理**

split(col(‘A’),delimiter) #字符串切分(返回列表列)

df.select(A,df.A.like(‘PATTERN’)) #正则匹配

df.filter(df.A.rlike('PATTERN')) #正则筛选

df[df.A.startwith(‘string’)] #头部匹配

df[df.A.endwith(‘string’)] #尾部匹配

df.select(df.A.substr(start,end)) #切片截取

F.substring(str, pos, len) #切片截取

F.length(col) #字符串长度

F.trim(col) #

F.upper(col) #转大写

F.lower(col) #转小写

**值替换**

df.replace(old\_value,new\_value)

**去重**

df=df.dropDuplicates()

**缺失值处理**

f.isnan() #缺失值判定

df.na.fill(value) #缺失值填充

df.na.drop(how='any'/’all’,\*thresh,\*subset) #缺失行丢弃

**抽样**

df.sample(withReplacement=False, fraction=None, seed=None)

**列表列处理**

df.A.getItem(n) #返回列表列中各n-th元素

f.explode(df[‘A’]) #将每一个列表列中的每一个元素都抽出来成一行，结合withColumn和select改变表结构

**更改数据类型&时间处理**

df=df.withColumn("colname", df["A"].cast('python\_datatype')) #int/float/str等

df=df.withColumn(“colname”,f.to\_timestamp(df[“colname”])) #字符串转时间类型

df=df.withColumn("colname", dayofweek("colname") #日期转星期几

f.hour(“colname”) #取小时

**数据重整(explode)**

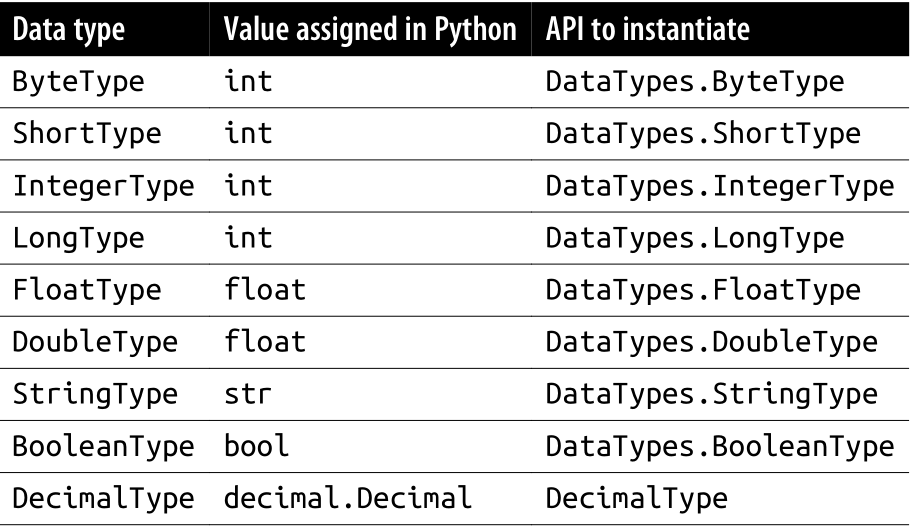
explode(df[‘A’])

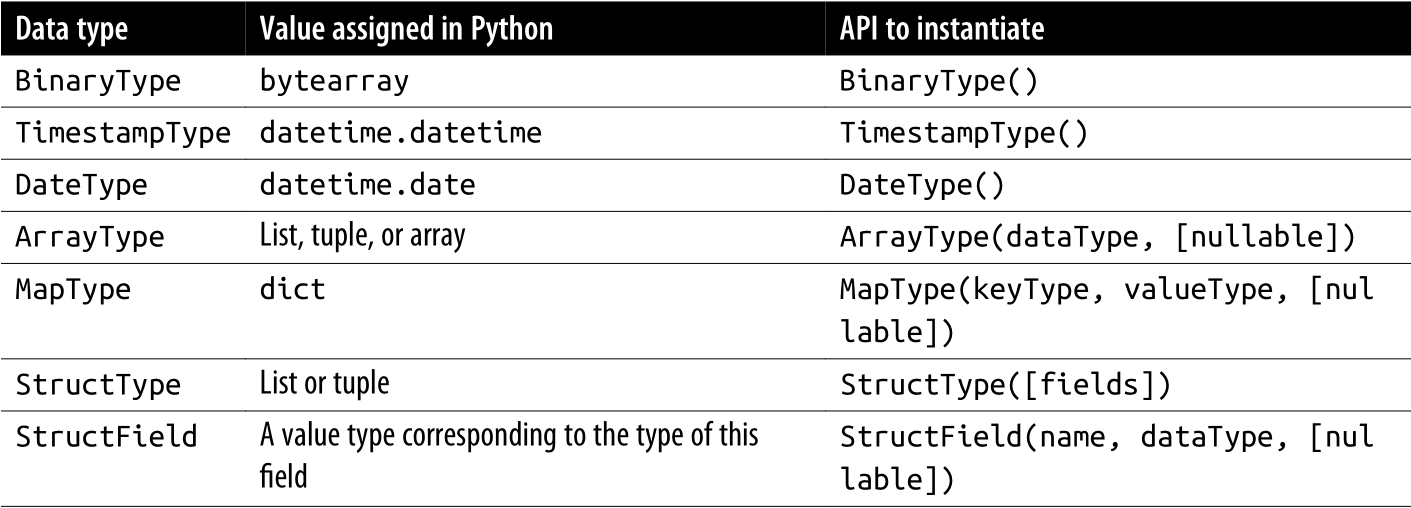
#将每一个列表列中的每一个元素都抽出来成一行，结合withColumn改变表结构

**数据分堆**

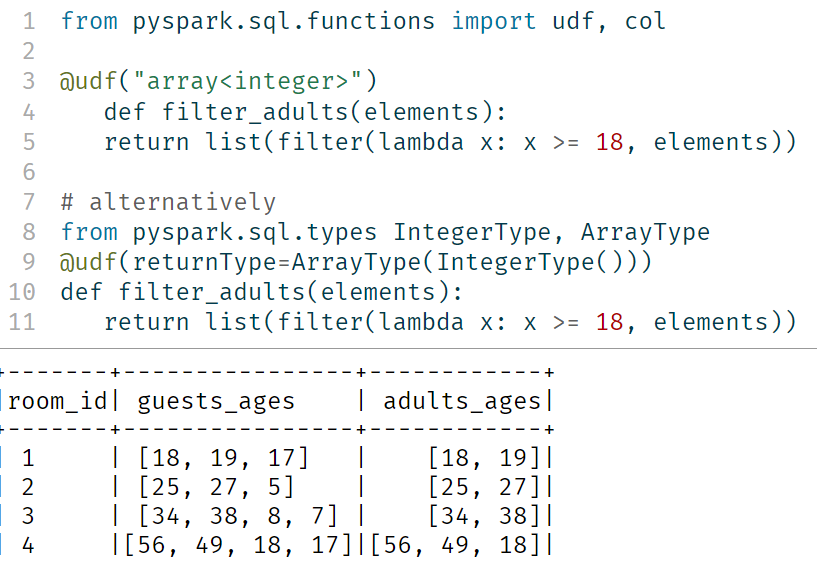
train\_data, validate\_data, test\_data = df.randomSplit([ratio1,ratio2,ratio3], seed)

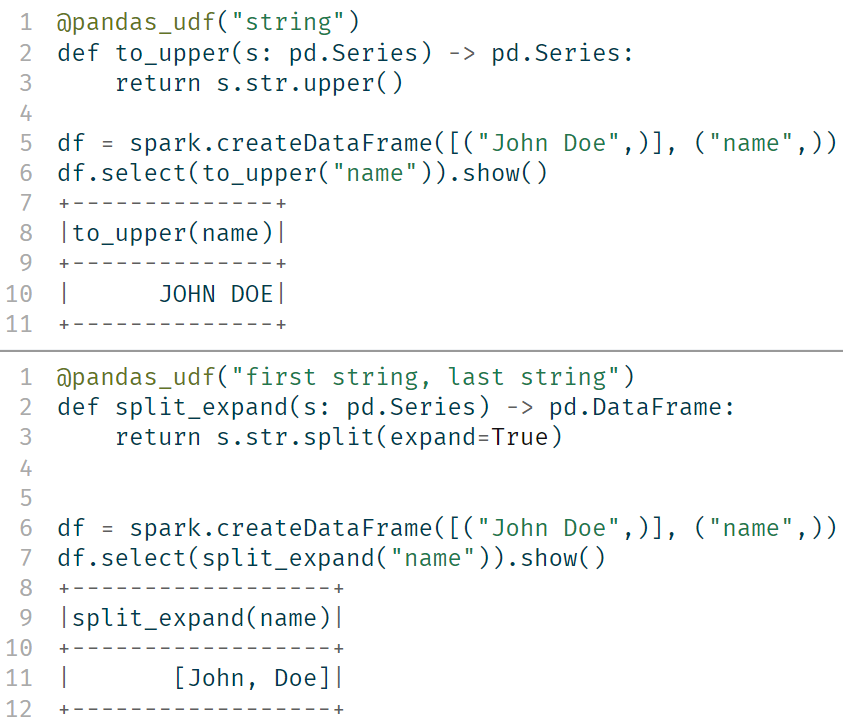
**DataFrame数据类型**





**自定义函数**





DStream

RDD构成的序列，用于处理流式数据

**创建StreamingContext**

#每个SparkSession只能同时执行一个StreamingContext

#StreamingContext和SparkContext同步终止

ssc=StreamingContext(sc,1)

SparkSQL

[**https://spark.apache.org/docs/2.3.0/api/sql/#regexp\_extract**](https://spark.apache.org/docs/2.3.0/api/sql/#regexp_extract)

[Cheat sheet PySpark SQL Python.indd](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PySpark_SQL_Cheat_Sheet_Python.pdf)

**启动SparkSQL API**

from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

**将DataFrame注册为SQL DataFrame**

schema = sqlContext.createDataFrame(data) #创建sql表对象

schema.registerTempTable("TABLE") #设置表名方法1

schema.cache()/spark.sql("cache table TABLE") #将表添加至缓存

df.createOrReplaceTempView("TABLE") #直接通过DataFrame进行注册

**SQL API**

spark.sql(‘SQL query’) #使用SQL对TABLE查询，返回df

返回结果是一个表，当查询结果是一个计算数时，可能需要. head()[0]之类的方法来获取想要的结果

**函数类**

regexp\_extract(value,PATTERN,idx=返回捕获组序号) #正则筛选

percentile(A,分位值) #取分位数

MLlib

<http://spark.apache.org/docs/3.1.1/ml-guide.html>

org.apache.spark.mllib #初级API，应用于RDD

org.apache.spark.ml #高级API，应用于DataFrame

## 预处理

**字符串水平转数字index**

SI=pyspark.ml.feature.StringIndexer(inputCol="col", outputCol="new\_col")

si=SI.fit(df)

res=si.transform(df)

**One-hot Encoding**

OHE=pyspark.ml.feature.OneHotEncoder(inputCol="col", outputCol="new\_col")

one=OHE.fit(df)

res=one.transform(df)

**多个特征组合为list**

VA = VectorAssembler(inputCols=[“col1”,”col2”,…], outputCol= "new\_col")

va=VA.fit(df)

res= va.transform(df)

#new\_col的形式为[col1,col2,…]

**数字水平转字符串**

IS= IndexToString(inputCol="col",outputCol="new\_col ",labels= ['Label\_A','Label\_B',…])

is=IS.fit(df)

res=is.transform(df)

**数据标准化**

Normalizer

StandardScaler

**分词**

Tokenizer

**去停用词**

StopWordsRemover

**分桶**

Bucketizer

QuantileDiscretizer

**Pipeline**

pipeline=pyspark.ml.Pipeline(stages=[processor\_instance1,processor\_instance 2,…])

pipeline\_instance=pipeline.fit(data)

result=pipeline\_instance.transform(data)

processor\_i = Processor()

.setInputCols([‘col1’,’col2’,…])

.setOutputCol(“colname”)

**Confusion Matrix**

sklearn.metrics.confusion\_matrix(y\_true, y\_pred)

## 分类模型

**随机森林**

from pyspark.ml.classification import RandomForestClassifier

model=RandomForestClassifier(labelCol="标签列名"

,featuresCol="features",numTrees,subsamplingRate)

#features列为通过VectorAssembler组合多个特征生成的list列

## Evaluator

metricName= "areaUnderROC"/"accuracy"/…

**二分类**

from pyspark.ml.evaluation import BinaryClassificationEvaluator

Evaluator=BinaryClassificationEvaluator(labelCol="真实标签列", predictionCol="预测标签列", metricName)

eval\_res = Evaluator.evaluate(model.transform(test\_df))

**多分类**

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

Evaluator=MulticlassClassificationEvaluator(labelCol="真实标签列", predictionCol="预测标签列", metricName)

eval\_res = Evaluator.evaluate(model.transform(test\_df))

HIVE

**启动HIVE API**

hive\_context = HiveContext(sc)

GraphX

**GraphFrames**

[GraphFrames User Guide (Python) (microsoft.com)](https://learn.microsoft.com/en-us/azure/databricks/_static/notebooks/graphframes-user-guide-py.html)