به نام خدا

Big Data

در تمامی فایل های پروژه سلول های ابتدایی به راه اندازی pyspark و همچنین کتابخانه ی Graphframe پرداخته شده است.

سوال ۱-۱ :

```
def clean_str(x):
  punc='.!?,'
  for ch in punc:
    clean_str = clean_str.replace(ch, '')
  return clean_str


sc=spark.sparkContext
filename ="1.txt"
txtfile = sc.textFile(filename)
words = txtfile.map(clean_str)
words =  words.flatMap(lambda satir: satir.split(" "))
words = words.filter(lambda x:x!='')
words=words.map(lambda  word:(word,1))
words_count=words.reduceByKey(lambda x,y:(x+y)).sortByKey()
```

در این قسمت ابتدا با استفاده از دستور Sc.textfile فایل مورد نظر ورودی سوال را میخوانیم و برای حذف علایم نگارشی نظیر که در سوال ذکر شده است از تابع clean_str استفاده کرده ایم که تمامی این علایم را با " جایگزین میکند . پس از آن کلمات را به وسیله ی علامت فاصله با تابع Flatmap جدا کرده و در متغیر words ذخیره میکنیم . پس از آن برای

شمارش تعداد کلمات موجود در متن از تابع map و متغیر ۱ به ازای هر کلمه استفاده میکنیم و در نهایت آن ها را با هم جمع میکنیم و متغیر word count را که تعداد را نشان میدهد میسازیم .

سوال ۱ بخش دوم:

برای این بخش به این شکل عمل میکنیم که ابتدا خطوط را از فایل میخوانیم و بعد کلمات را با استفاده از علامت فاصله جدا میکنیم سپس در مرحله ی بعد با استفاده از یک مرحله فیلتر و یک مرحله map کردن ابتدا تمامی حروفی را که حروف الفبا هستند و حرف ابتدایی کلمات هستند را استخراج میکنم و یک دوتایی به ازای هر حرف و عدد ۱ برای شمارش ذخیره میکنیم (متغیر pair) و در نهایت برای این که جمع تمامی حروف ابتدایی کلمات را در بیاوریم از تابع reduced key استفاده کردیم و تعداد تکرار هر حرف را استخراج کردیم .

```
lines = sc.textFile("1.txt")

words = lines.flatMap(lambda line: line.split())

letters = words.filter(lambda letter: letter.isalpha()) \
               .map(lambda word: word[0].lower())

pairs = letters.map(lambda letter: (letter, 1))

counts = pairs.reduceByKey(lambda n1, n2: n1 + n2)

counts.saveAsTextFile("out1-part2")
sc.stop()
```

در شکل زیر خروجی این بخش مشخص شده است .

part-00001

```
('a', 622)
('f', 144)
('w', 341)
('t', 946)
('e', 100)
('o', 304)
('k', 21)
('n', 54)
('u', 54)
('m', 136)
('x', 5)
('q', 11)
('v', 28)
```

همانطور که مشخص است حرف m ۱۳۶ بار تکرار شده است .

سوال ۱ بخش ۵ :

در این بخش دو کلمه ای ها را استخراج میکنیم و پر تکرار ترین های آن را نمایش می دهیم .

```
from operator import add

def parse_bigrams(line):
        bigrams = []
        unigrams = line.strip().split()
        for i in range(len(unigrams) - 1):
            bigram = unigrams[i].lower() + ":" + unigrams[i+1].lower()
            bigrams.append(bigram)

        return bigrams


lines = sc.textFile("1.txt")
counts = lines.flatMap(parse_bigrams).map(lambda x: (x, 1)).reduceByKey(add)
output = counts.collect()

total_bigram_occurrences = 0
bigram_counts = []
for (bigram, count) in output:
      total_bigram_occurrences += count
      bigram_counts.append((bigram, count))
bigram_counts.sort(key=lambda x: x[1], reverse=True)
print(bigram_counts)
```

```
[('of:the', 56), ('in:the', 53), ('and:the', 15), ('this:is', 15), ('all:of', 13), ('it:is', 13), ('on:the', 12), (
```

در این قسمت با استفاده از تابعی تحت عنوان parse bigram تمامی دو کلمه ای هایی که در متن وجود دارد استخراج میکنیم . نحوه ی کارکرد این تابع به این شکل است که کلمات را دو تا دو تا جلو پیمایش میکند و آنها را ذخیره میکند و باز میگرداند . در بخش بعدی با استفاده از تابع map آن ها را به صورت یک دوتایی همراه عدد ۱ ذخیره میکنیم که در آینده بتوانیم آن ها را بشماریم . در حلقه ی for تمامی دوکلمه ای ها را به همراه تعداد تکرار آن ها در متغیر total-bigram جمع آوری میکنیم و در bigram_counts آن را به صورت افزایشی مرتب سازی میکنیم . و نشان میدهیم خروجی این قسمت در تصویر بالا مشخص است .

سوال ۲ :

قسمت ۱ :

در ابتدا با استفاده از تابع Spark.read فایل لاگ را میخوانیم . از آن جایی که به طور پیشفرض همه ی محتوای یک خط داخل یک ستون قرار میگیرد بنابراین باید قسمت host را با استفاده از علامت space جداسازی کنیم . پس با استفاده از split محتوای ستون value را جداسازی کرده و در خط بعد با استفاده از قسمت اول آرایه ای که بعد از split تولید میشود یک ستون جدید ایجاد میکنیم که شامل آدرسها است و ستون value را حذف میکنیم . دیتا فریم به شکلی که مشاهده میکنید در می آید و برای این که تعداد host های یکتا را بشماریم از group by بر روی ستون آدرس ها استفاده میکنیم و ستون بعدی را تعداد تکرار هر host میگذاریم . در مجموع در این فایل لاگ ۸۱۹۸۴ host یکتا وجود داشت که خروجی در شکل زیر و کد آن مشخص است .

```
import pyspark.sql.functions as F

df = spark.read.text("Log")
split_col = f.split(df['value'], ' ')
df = df.withColumn('address', split_col.getItem(0))
df2 = df.drop("value")
df2.show()
hostcount =  df2.groupBy("address").count().select('address' , F.col('count').alias('F'))
hs = hostcount.orderBy('F' , ascending=False)
print(hostcount.count())
```

```
+--------------------+-----+
|             address|    F|
+--------------------+-----+
|piweba3y.prodigy.com|17609|
|piweba4y.prodigy.com|11676|
|piweba1y.prodigy.com| 9879|
|  alyssa.prodigy.com| 7880|
| siltb10.orl.mmc.com| 7573|
|piweba2y.prodigy.com| 5936|
|  edams.ksc.nasa.gov| 5458|
|        163.206.89.4| 4906|
|         news.ti.com| 4881|
|disarray.demon.co.uk| 4486|
|   poppy.hensa.ac.uk| 4327|
|www-d1.proxy.aol.com| 4200|
|www-a2.proxy.aol.com| 4199|
|  vagrant.vf.mmc.com| 4146|
|       198.133.29.18| 4136|
|www-d4.proxy.aol.com| 3981|
|    webgate1.mot.com| 3843|
|www-b3.proxy.aol.com| 3766|
|  e659229.boeing.com| 3737|
|www-b2.proxy.aol.com| 3734|
+--------------------+-----+
only showing top 20 rows

81984
```

قسمت ۲ :

در این بخش باید متوسط درخواست های هر میزبان را به صورت روزانه استخراج کنیم.

برای این بخش همانند بخش قبلی آدرس ها را با استفاده از space در آوردیم و برای در آوردن روز از وجود داشتن علامت " : " در بخش تاریخ استفاده میکنیم و روز ها را در می آوریم . سپس ستون های اضافی را حذف میکنیم (ستون های value و time مانند بخش قبلی حذف میشوند) حال برای شمارش groupby را بر اساس دو ستون address و day که پیش از این استخراج شد انجام میدهیم و شمارش را روی این دو عدد قرار میدهیم . و در نهایت خروجی را مانند شکل زیر نمایش میدهیم (به صورت مرتب شده)

```
from pyspark.sql.functions import count, avg


df = df.withColumn('time', split_col.getItem(3))
df2 = df.drop("value")
# df2.show()
spli = f.split(df2['time'], ':')
df2 = df.withColumn('day', spli.getItem(0))

df3 = df2.drop("value")
df3 = df3.drop("time")
hostdayinoutcount =  df3.groupBy("address", "day").count().select('address' , 'day' , F.col('count').alias('Frequency'))
hostdayinoutcount.orderBy('address' , ascending=True).show()

# df3.show()
```

```
+--------------------+------------+---------+
|             address|         day|Frequency|
+--------------------+------------+---------+
|         ***.novo.dk|[11/Jul/1995|       16|
|      007.thegap.com|[06/Jul/1995|       30|
|      007.thegap.com|[09/Jul/1995|       11|
|      007.thegap.com|[23/Jul/1995|        4|
|01-dynamic-c.rott...|[28/Jul/1995|        1|
|01-dynamic-c.woki...|[05/Jul/1995|        3|
|01-dynamic-c.woki...|[27/Jul/1995|        9|
|01-dynamic-c.woki...|[10/Jul/1995|       11|
|01-dynamic-c.woki...|[12/Jul/1995|        5|
|02-dynamic-c.woki...|[27/Jul/1995|        4|
|02-dynamic-c.woki...|[07/Jul/1995|        9|
|03-dynamic-c.woki...|[04/Jul/1995|       12|
|03-dynamic-c.woki...|[20/Jul/1995|        3|
|04-dynamic-c.rott...|[03/Jul/1995|       22|
|04-dynamic-c.woki...|[04/Jul/1995|       28|
|05-dynamic-c.rott...|[06/Jul/1995|       12|
|05-dynamic-c.rott...|[05/Jul/1995|       14|
|05-dynamic-c.woki...|[15/Jul/1995|        9|
|06-dynamic-c.rott...|[14/Jul/1995|       11|
|06-dynamic-c.rott...|[04/Jul/1995|       10|
+--------------------+------------+---------+
only showing top 20 rows
```

قسمت ۳ :

برای دریافت تعداد فایل های گیف درخواست شده به این شکل عمل میکنیم که در هر خط از فایل به دنبال الگوی ".gif" میگردیم و در نهایت تعداد آن ها را گزارش می دهیم . این عمل با استفاده از filter انجام میشود .

```
# df.show()
df = df.drop("time")
df = df.drop("address")
# df.show()
df2 = df.filter(df.value.like('%.gif%'))
print(df2.count())
```

```
1034638
```

قسمت ۴ :

در این قسمت با استفاده از تعداد تکرار دامنه هایی که در قسمت ۱ در آوردیم و در ابتدا مرتب سازی آن ها با استفاده از orderby آن را از یک فیلتر رد میکنیم و در ابتدا فقط آن دامنه هایی را درخواست می کنیم تعداد تکرار آن ها از ۳ بیشتر باشد . پس از آن دیتا فریم شامل تمام دامنه هایی است که حداقل ۴ بار تکرار داشته اند . سپس با استفاده از امکانات regular expr یک فیلتر روی بخش آدرس ها اعمال می کنیم تا تمامی ip ها حذف شوند .

```
hs = hostcount.orderBy('F' , ascending=True)
df = hs.filter("F > 3 and")
expr = "^((?!-)[A-Za-z0-9-]{1,63}(?<!-)\\.)+[A-Za-z]{2,6}$"
dk = df.filter(df["address"].rlike(expr))

dk.show()
```

```
+--------------------+---+
|             address|  F|
+--------------------+---+
|hwwmac.larc.nasa.gov|  4|
| braemar.demon.co.uk|  4|
|berlin.vas.viewlo...|  4|
|dr4umac1.med.virg...|  4|
| scorch.doc.ic.ac.uk|  4|
|blv-pm1-ip9.halcy...|  4|
| dram.cmu.susx.ac.uk|  4|
|mac12.bnf23.ulava...|  4|
|  guest7.cni.mid.net|  4|
|   bend6.bendnet.com|  4|
|ix-sj12-17.ix.net...|  4|
|n113.solano.commu...|  4|
|ix-lv4-04.ix.netc...|  4|
|    dwkm107.usa1.com|  4|
|      lom000.wwa.com|  4|
|annex-v32bis-47.s...|  4|
|cust46.max1.ffx1....|  4|
|ix-phi1-01.ix.net...|  4|
|   mufasa.ee.pdx.edu|  4|
|ppp1-06.inre.asu.edu|  4|
+--------------------+---+
only showing top 20 rows
```

قسمت ۵ :

در این قسمت با استفاده از علامت " ابتدا هر خط را split میکنیم و در مرحله ی بعد با استفاده از علامت فاصله به status code میرسیم . سپس آن ها را شمارش کرده و با استفاده از تابع order by آن را مرتب میکنیم . شمارش به وسیله ی group by و همانند مراحل قبل انجام میشود .

```
df = spark.read.text("Log")
split_col = f.split(df['value'], '"')
df = df.withColumn('statusbebad', split_col.getItem(2))
df2 = df.drop("value")
split_col2 = f.split(df['statusbebad'], ' ')
df2 = df2.withColumn('status code', split_col2.getItem(1))
df2 = df2.drop("statusbebad")

# df2.show()

# df2.show()
statuscount =  df2.groupBy("status code").count().select('status code'  , F.col('count').alias('Frequ
statuscount.orderBy('Frequency' , ascending=False).show(5)
```

```
+-----------+---------+
|status code|Frequency|
+-----------+---------+
|        200|  1709982|
|        304|   133349|
|        302|    46925|
|        404|    10878|
|        500|       62|
+-----------+---------+
only showing top 5 rows
```

سوال 3

در این سوال از داده های 60 روز کاری در سازمان بورس استفاده شده است . پس از استخراج داده ها آن ها را در یک پوشه به نام Datasets در گوگل درایو آپلود کرده و پس از آن با پیمایش بر روی تمامی فایل ها محتوای آنها را در یک دیتا فریم ذخیره کرده و لیستی از دیتا فریم ها تشکیل می دهیم تا در تمرینات بعد بتوانیم بر روی این لیست پیمایش کرده و پاسخ سوالات را بدهیم. در این قسمت همچنین ستون اطلاعات و شرکت ها و تاریخ به هر دیتا فریم اضافه میشود .

```
import os
import re
import pyspark.sql.functions as F
from pyspark.sql.functions import lit
DFlist = []
directory = "Datasets"
for filename in os.scandir(directory):
    if ((filename.path.endswith(".csv")) and ("MarketWatchPlus" in filename.path)):
        FileN = filename.path[25:-4]
        df = spark.read.text(filename.path)
        split_col = F.split(df['value'], ',')
        df = df.withColumn('شرکت', split_col.getItem(0))
        df = df.filter((df.شرکت != 'نماد'))
        df = df.withColumn('تاریخ', lit(FileN))
        df = df.withColumnRenamed('value' , 'اطلاعات')

        df.show(truncate=True)
        DFlist.append(df)
        print(FileN)
    else:
        continue
DFlist[0].show()
```

قسمت 1 :

در این قسمت ابتدا آخرین فایل موجود در پروژه را به وسیله ی توابع مخصوص باز میکنیم و اطلاعات شرکت و تاریخ و قیمت را در ستون های مختلف جایگذاری کرده و با استفاده از توابع دیتافریم یعنی تابع order by گرانترین سهام را نمایش می دهیم . (10 تا عدد) پس از آن با استفاده از توابع SQL در spark و نوشتن یک دستور SQL به صورتی که نام شرکت و قیمت ها را بر اساس قیمت به صورت نزولی نشان دهد پاسخ این سوال را از دیدگاه SQL SPARK نیز بدست می آوریم . لازم به ذکر است برای استفاده از توابع SQL SPARK یک View از دیتا فریم ایجاد شده است که نام آن را PriceQ گذاشتیم .

```
#Question 1 - both approach

import pyspark.sql.functions as F
from pyspark.sql.functions import lit

df = spark.read.text("/content/Datasets/MarketWatchPlus-1400_3_18.csv")
split_col = F.split(df['value'], ',')
df = df.withColumn('Company', split_col.getItem(0))
df = df.withColumn('Date', lit(14000320))
df = df.withColumn('Price', split_col.getItem(7))
df = df.drop("value")
df = df.filter((df.Company != 'نماد'))
# Dataframe
df.orderBy(df.Price.cast('int') , ascending=False).show(10,truncate=False)

# SQL
df.createGlobalTempView("PriceQ")

spark.sql("SELECT company , Price FROM global_temp.PriceQ  ORDER BY price DESC;").show()
```

```
+----------+--------+-------+
|Company   |Date    |Price  |
+----------+--------+-------+
|1654488|14000320|04پ0008پست|
|1640999|14000320|08پ0008پست|
|1316530|14000320|  002سمپا|
|1165665|14000320|   001سفت|
|1089000|14000320|    38اراد|
|1074500|14000320|05پ0111سکه|
|1072000|14000320|03پ0112سکه|
|1070300|14000320|04پ0012سکه|
|1070202|14000320|01پ0012سکه|
|1070030|14000320|02پ0011سکه|
+----------+--------+-------+
only showing top 10 rows
```

قسمت 2 :

برای حل این سوال نیز همانند بخشی قبلی بر روی لیستی که در مرحله ی پیش پردازش بدست آوردیم عمل میکنیم و در یک حلقه ی For تمامی دیتا فریم ها را با هم الحاق می کنیم زیرا اطلاعات ماه های اخیر را میخواهیم . در ابتدا ستون حجم را به int تبدیل میکنیم (این ستون به صورت پیش فرض string بود و نمی توان روی آن عملگر sum اعمال کرد) پس از آن برای حل سوال با دیدگاه Dataframe با استفاده از توابع sum و همچنین گروه بندی ابتدا داده ها را بر اساس نام

شرکت گروه بندی و پس از آن حجم مجموع هر شرکت را بدست می آوریم . همچنین در نهایت برای حل سوال از دیدگاه SQL دستور SQL ای بنویسیم که مجموع ستون حجم را در کنار نام شرکت ها پس از گروه بندی به صورت نزولی مرتب نشان دهد .

```
#Q2

import pyspark.sql.functions as F
from pyspark.sql.functions import sum as _sum
from pyspark.sql.functions import count as _count



mainDF = DFlist[0]
for i in range(1,len(DFlist)):
    mainDF = mainDF.union(DFlist[i])

split_col = F.split(mainDF['اطلاعات'], ',')
mainDF = mainDF.withColumn('stringhajm', split_col.getItem(3))
mainDF = mainDF.withColumn("حجم", mainDF["stringhajm"].cast("int"))

mainDF = mainDF.drop("اطلاعات")
mainDF = mainDF.drop('stringhajm')
mainDF.show()
# DAtaframe  Appr
mainDF =  mainDF.groupBy("شرکت").sum('حجم')

mainDF.orderBy('sum(حجم)' , ascending=False).show()

mainDF = mainDF.withColumnRenamed('شرکت'  , 'company')
mainDF = mainDF.withColumnRenamed('حجم'  , 'Hajm')
#SQL APPR
# mainDF.createGlobalTempView("mainView")

spark.sql("SELECT SUM(Hajm),company FROM global_temp.mainView GROUP BY company ORDER BY SUM(Hajm) DESC;").show()
```

خروجی این قسمت به شرح زیر است .

```
+-----------+---------+
|  sum(Hajm)|  company|
+-----------+---------+
|17358088095|    خودرو|
|14883735471|    خسایا|
| 9659578499|   وتجارت|
| 9360198795|     کمند|
| 7834266581|    وبملت|
| 7405957505|    خبهمن|
| 7190163229|    کرمان|
| 7125231050|   وبصادر|
| 6339459487|    خگستر|
| 6097139582| امین یکم|
| 4579388359|     ثبتنا|
| 4114081377|    وپارس|
| 3826500000|    کمند2|
| 3794822812|      ذوب|
| 3274299505|     فملی|
| 3032517414|    فولاد|
| 2944351138|    ثنران|
| 2483874646|     ثسنا|
| 2201408227|  بترانس4|
| 1931900000|امین یکم2|
+-----------+---------+
only showing top 20 rows
```

سوال ۴ :

قسمت ۱ :

در این بخش برای این که بتوانیم گراف را بسازیم ابتدا محتوای فایل های راس ها و یال ها را میخوانیم و آن ها را نامگذاری میکنیم . طبق استاندارد تابع gameframes که در واقع سازنده ی گراف است باید دو عدد دیتا فریم به عنوان ورودی بگیرد که اولی دیتا فریم راس ها و دومی دیتا فریم های یال هاست . برای استخراج هر کدام با استفاده از " " هر خط در فایل را جداسازی می کنیم و ستون های دیتا فریم جدید را به صورت src , dst , id , name نامگذاری میکنیم . id در واقع همان شناسه ی راس هاست و name توضیح مقاله است . پس از آن با استفاده از GameFrame گراف را میسازیم .

```
from graphframes import GraphFrame
import pyspark.sql.functions as F

df = spark.read.text("edges.txt")
split_col = f.split(df['value'], '  ')
df = df.withColumn('src', split_col.getItem(0))
df = df.withColumn('dst', split_col.getItem(1))
edges = df.drop("value")
# edges.show()

df = spark.read.text("Vertex.txt")
split_col = f.split(df['value'], '  ')
df = df.withColumn('id', split_col.getItem(0))
df = df.withColumn('name', split_col.getItem(1))
vertex =  df.drop("value")
g  = GraphFrame(vertex , edges)
```

قسمت ۲:

```
dstcount =  edges.groupBy("dst").count().select('dst' , F.col('count').alias('degree'))
srccount =  edges.groupBy("src").count().select('src' , F.col('count').alias('degree'))


dstcount.orderBy('degree' , ascending=False).show()
srccount.orderBy('degree' , ascending=False).show()
```

در این قسمت با استفاده از group by روی راس هایی که به عنوان ورودی کار کرده اند تعداد تکرار آن ها را در می آوریم(با استفاده از تابع count) و در نهایت آن ها را به صورت مرتب نمایش میدهیم . در این قسمت یال های پرتکرار خروجی را هم نشان داده ایم .

در شکل پایین خروجی این قسمت مشخص شده است . پرتکرار ترین راس ها در راس های مقصد(همان درجه ی بیشترین ورودی) در شکل مشخص شده است . همچنین درجه ی ورودی نیز مشخص شده است .

```
+-------------------+------+
|                dst|degree|
+-------------------+------+
| 946065507707541358|   327|
|3856212023725725593|   322|
|8978262722425160811|   316|
|6245498229508734555|   185|
|7264519433548233535|   180|
|5362090331808156011|   179|
| 277710621679830671|   149|
|1984578398767042266|   145|
|2395551540800395672|   134|
|5395033957924805072|   130|
|4254821691068011447|   119|
|7050959159375889025|   116|
|3282427710539568335|   115|
|4512002392249809415|   115|
|4254821691068016541|   113|
|4254821691068027845|   107|
|6623304200448015171|   105|
|2625575280669536231|   104|
|6908101889982380382|   102|
| 330493081995023431|   102|
+-------------------+------+
only showing top 20 rows
```

همچنین درجه ی خروجی نیز به همان ترتیب محاسبه شده و در یک دیتا فریم دیگر ذخیره میشود .

```
+-------------------+------+
|                src|degree|
+-------------------+------+
|3841755165517709241|   264|
|4768697715794291382|   167|
| 946065507707541358|   141|
|1984578398767042266|   132|
|1749892575156253660|   130|
```

قسمت ۳ :

در این قسمت برای استخراج connected component ها از توابع Grafframe استفاده شده است . بخش ها را استفاده از تابع استخراج کردیم و آن ها را به ترتیب نشان داده ایم .

```
[190] CC = g.connectedComponents()
      CC.select("id", "component").orderBy("component").show()
```

قسمت ۴ :

برای پیدا کردن ده تا از مقالات برتر در واقع باید ۱۰ تا از مقالاتی را پیدا کنیم که بیشترین ورودی را داشته اند .

برای این کار نیازمند آن هستیم که ابتدا ۱۰ تا از پرتکرار ترین درجه ورودی را پیدا کنیم و در نهایت یک join بین دو دیتا فریم یعنی بین دیتا فریمی که ۱۰ تا از پرتکرار ترین راس ها و دیتا فریمی که نام مقالات در آن ذخیره شده است انجام دهیم.

```
bestarticle = dstcount.orderBy('degree' , ascending=False)
joinresult = bestarticle.join(vertex,vertex.id ==  bestarticle.dst,"inner")
joinresult.orderBy('degree' , ascending=False).show()
```

```
+------------------+------+------------------+--------------------+
|               dst|degree|                id|                name|
+------------------+------+------------------+--------------------+
| 946065507707541358|   327| 946065507707541358| The Football League|
|3856212023725725593|   322|3856212023725725593|National Football...|
|8978262722425160811|   316|8978262722425160811|Australian Footba...|
|6245498229508734555|   185|6245498229508734555|Southern Football...|
|7264519433548233535|   180|7264519433548233535|Football League F...|
|5362090331808156011|   179|5362090331808156011|Football League S...|
| 277710621679830671|   149| 277710621679830671|All-Ireland Senio...|
|1984578398767042266|   145|1984578398767042266|Scottish Football...|
|2395551540800395672|   134|2395551540800395672|Football League T...|
|5395033957924805072|   130|5395033957924805072|Pro Football Hall...|
|4254821691068011447|   119|4254821691068011447| Football League Two|
|7050959159375889025|   116|7050959159375889025|Carlton Football ...|
|3282427710539568335|   115|3282427710539568335|Western Football ...|
|4512002392249809415|   115|4512002392249809415|American Football...|
|4254821691068016541|   113|4254821691068016541| Football League One|
|4254821691068027845|   107|4254821691068027845| Football League Cup|
|6623304200448015171|   105|6623304200448015171|Essendon Football...|
|2625575280669536231|   104|2625575280669536231|Collingwood Footb...|
| 330493081995023431|   102| 330493081995023431|Melbourne Footbal...|
|6908101889982380382|   102|6908101889982380382|The Football Asso...|
+------------------+------+------------------+--------------------+
only showing top 20 rows
```

برای این کار پس از آن که پرتکرار ترین راس ها را به ترتیب در یک دیتا فریم ذخیره کردیم آن ها را به صورت یک join با دیتا فریمی که اطلاعات راس ها در آن ذخیره شده است (vertex) به جدولی تبدیل میکنیم که مقالات پرتکرار را لیست میکند . join روی آیدی راس ها انجام شده است و مرتب سازی نهایی بر اساس تعداد درجه ی ورودی (پرتکرار ترین ها) انجام شده است . لازم به ذکر است اگر بخواهیم ۱۰ تا را فقط نمایش دهیم در خط آخر در تابع show عدد 10 را پاس می دهیم .