E_Commerce新零售分析系统

1. 第一章 案例概述

1.1 案例背景

在互联网、移动互联网的带动下，新零售行业崛起。

在这两年里，盒马鲜生、超级物种、无人货架、天猫小店、苏宁小店等"新物种"已经让我们看到了新零售先锋企业的锋芒;而传统百货商超、购物中心、便利店等在人工智能、大数据、智慧物流的改造下，也焕然一新，搭上了新零售的高速列车。智能终端的普及、大数据等技术的不断革新，进一步开拓了线下的场景消费，使得消费者不再受时间和空间制约。万物皆可新零售，似乎已经成为必然的趋势。

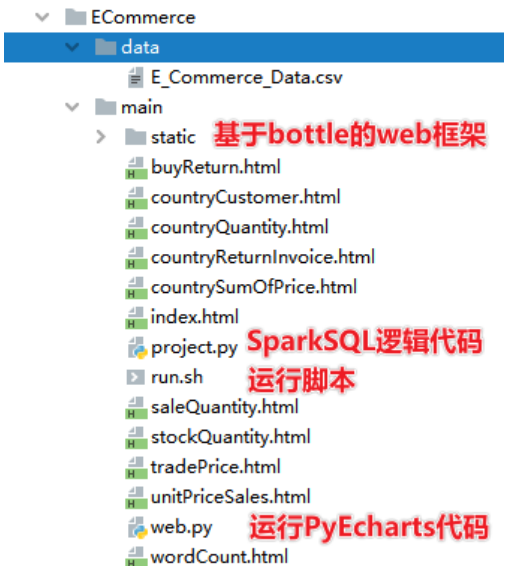
近年来，"新零售"正在焕发勃勃生机。2020年，我国实物商品网上零售额占社会消费品零售总额的比重接近四分之一，使用电子支付的成人比例超过八成，全国快递业务量达到800多亿件。其中，上海的贡献很大，2020年，上海社会消费品零售总额达到1.59万亿元，稳居全国城市首位，网络购物交易额1.17万亿元，位居全国前列。

1.2 架构流程

数据科学流程：

1.3 业务模块

创建包如下：



2. 第二章 环境搭建

2.1 安装必备的包

Bottle是一个快速、简洁、轻量级的基于WSIG的微型Web框架，此框架除了Python的标准库外，不依赖任何其他模块。安装方法是，打开Linux终端，执行如下命令：

```
pip install pyspark
Pip install pyspark[sql]
```

```
pip install bottle
```

启动注意事项：

```
progress
Traceback (most recent call last):
  File "/export/pyfolder1/ECommerce/main/project.py", line 13, in <module>
    df = spark.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('file:///export/pyfol
erce/data/E_Commerce_Data_Clean.csv')
  File "/export/servers/spark/python/lib/pyspark.zip/pyspark/sql/readwriter.py", line 204, in load
  File "/export/servers/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py", line 1304, in __call__
  File "/export/servers/spark/python/lib/pyspark.zip/pyspark/sql/utils.py", line 117, in deco
pyspark.sql.utils.AnalysisException: Path does not exist: file:/export/pyfolder1/ECommerce/data/E_Commerce_Data_Clean.csv
Traceback (most recent call last):
  File "web.py", line 1, in <module>                                    错误异常1：请查看本机文件路径
    from bottle import route, run, static_file
ModuleNotFoundError: No module named 'bottle' 按照课件安装bottle  pip install bottle
```

启动：

```
sh run.sh
```

截图如下：

```
(base) [root@node1 main]# sh run.sh    启动脚本
2021-09-16 22:06:41,267 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2021-09-16 22:06:41,990 INFO spark.SparkContext: Running Spark version 3.1.2
2021-09-16 22:06:42,031 INFO resource.ResourceUtils: ==============================================================
2021-09-16 22:06:42,031 INFO resource.ResourceUtils: No custom resources configured for spark.driver.
2021-09-16 22:06:42,034 INFO resource.ResourceUtils: ==============================================================
2021-09-16 22:06:42,034 INFO spark.SparkContext: Submitted application: spark_project
2021-09-16 22:06:42,060 INFO resource.ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: ,
 -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amoun
2021-09-16 22:06:42,072 INFO resource.ResourceProfile: Limiting resource is cpu
2021-09-16 22:06:42,073 INFO resource.ResourceProfileManager: Added ResourceProfile id: 0
2021-09-16 22:06:42,121 INFO spark.SecurityManager: Changing view acls to: root
2021-09-16 22:06:42,121 INFO spark.SecurityManager: Changing modify acls to: root
2021-09-16 22:06:42,121 INFO spark.SecurityManager: Changing view acls groups to:
2021-09-16 22:06:42,121 INFO spark.SecurityManager: Changing modify acls groups to:
2021-09-16 22:06:42,121 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(root); gr
ermissions: Set(); users  with modify permissions: Set(root); groups with modify permissions: Set()
2021-09-16 22:06:42,321 INFO util.Utils: Successfully started service 'sparkDriver' on port 32833.
2021-09-16 22:06:42,353 INFO spark.SparkEnv: Registering MapOutputTracker
2021-09-16 22:06:42,384 INFO spark.SparkEnv: Registering BlockManagerMaster
2021-09-16 22:06:42,404 INFO storage.BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
2021-09-16 22:06:42,404 INFO storage.BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
2021-09-16 22:06:42,408 INFO spark.SparkEnv: Registering BlockManagerMasterHeartbeat
2021-09-16 22:06:42,424 INFO storage.DiskBlockManager: Created local directory at /tmp/blockmgr-4f2864a9-4bb1-4940-b30e-fdf934fbce47
2021-09-16 22:06:42,448 INFO memory.MemoryStore: MemoryStore started with capacity 366.3 MiB
2021-09-16 22:06:42,468 INFO spark.SparkEnv: Registering OutputCommitCoordinator
2021-09-16 22:06:42,558 INFO util.log: Logging initialized @2674ms to org.sparkproject.jetty.util.log.Slf4jLog
2021-09-16 22:06:42,634 INFO server.Server: jetty-9.4.40.v20210413; built: 2021-04-13T20:42:42.668Z; git: b881a572662e1943a14ae12e7e1207989f218b74; jvm 1.8
2021-09-16 22:06:42,655 INFO server.Server: Started @2772ms
2021-09-16 22:06:42,677 WARN util.Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
2021-09-16 22:06:42,687 INFO server.AbstractConnector: Started ServerConnector@4d348113{HTTP/1.1, (http/1.1)}{0.0.0.0:4041}
```

访问Url：http://192.168.88.161:9999/

```
done -> buyReturn , save to -> static/buyReturn.json
done -> unitPriceSales , save to -> static/unitPriceSales.json
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:9999/
Hit Ctrl-C to quit.

192.168.88.1 - - [16/Sep/2021 22:08:12] "GET / HTTP/1.1" 200 5645
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /countryCustomer.html HTTP/1.1" 200 2018
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /countryQuantity.html HTTP/1.1" 200 2010
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /countrySumOfPrice.html HTTP/1.1" 200 2081
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /stockQuantity.html HTTP/1.1" 200 2004
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /wordCount.html HTTP/1.1" 200 16424
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /countryReturnInvoice.html HTTP/1.1" 200 2043
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /tradePrice.html HTTP/1.1" 200 2146
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /saleQuantity.html HTTP/1.1" 200 1638
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /buyReturn.html HTTP/1.1" 200 2453
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /unitPriceSales.html HTTP/1.1" 200 1858
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/jquery-3.2.1.min.js HTTP/1.1" 200 86659
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/echarts-4.7.0.min.js HTTP/1.1" 200 777892
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/echarts-wordcloud.min.js HTTP/1.1" 200 128307
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/countrySumOfPrice.json HTTP/1.1" 200 1197
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/countryCustomer.json HTTP/1.1" 200 179
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/stockQuantity.json HTTP/1.1" 200 182
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/countryReturnInvoice.json HTTP/1.1" 200 179
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/countryQuantity.json HTTP/1.1" 200 214
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/tradePrice.json HTTP/1.1" 200 415
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/saleQuantity.json HTTP/1.1" 200 6953
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/unitPriceSales.json HTTP/1.1" 200 116942
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/buyReturn.json HTTP/1.1" 200 642
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /static/wordCount.json HTTP/1.1" 200 5258
192.168.88.1 - - [16/Sep/2021 22:08:12] "GET /favicon.ico HTTP/1.1" 404 747
```

3. 第三章 统计分析

3.1 数据描述

数据集E_Commerce_Data.csv包含541909个记录，时间跨度为2010-12-01到2011-12-09，每个记录由8个属性组成，具体的含义如下表：

字段名称 类型 含义 举例

- InvoiceNo string 订单编号（退货订单以C开头） 536365
- StockCode string 产品代码 85123A
- Description string 产品描述 WHITE METAL LANTERN
- Quantity integer 购买数量（负数表示退货） 6
- InvoiceDate string 订单日期和时间 12/1/2010 8:26
- UnitPrice double 单价（英镑） 3.39
- CustomerID integer 客户编号 17850
- Country string 国家名称 United Kingdom

3.2 数据预处理

接着，使用如下命令进入pyspark的交互式编程环境，对数据进行初步探索和清洗：

```
cd /export/server/spark #进入Spark安装目录
./bin/pyspark --master
```

Shell 命令

（1）读取在HDFS上的文件，以csv的格式读取，得到DataFrame对象。

```
>>>df=spark.read.format('csv').options(header='true',
inferschema='true').load('E_Commerce_Data.csv')
```

（2）查看数据集的大小，输出541909，不包含标题行

```
>>> df.count()
```

（3）打印数据集的schema，查看字段及其类型信息。输出内容就是上文中的属性表。

```
>>> df.printSchema()
```

（4）创建临时视图data。

```
>>> df.createOrReplaceTempView("data")
```

（5）由于顾客编号CustomID和商品描述Description均存在部分缺失，所以进行数据清洗，过滤掉有缺失值的记录。特别地，由于CustomID为integer类型，所以该字段若为空，则在读取时被解析为0，故用df[“CustomerID”]!=0 条件过滤。

```
>>> clean=df.filter(df["CustomerID"]!=0).filter(df["Description"]!="")
```

（6）查看清洗后的数据集的大小，输出406829。

```
>>> clean.count()
```

（7）数据清洗结束。根据要求，预处理后需要将数据写入HDFS。将清洗后的文件以csv的格式，写入E_Commerce_Data_Clean.csv中（实际上这是目录名，真正的文件在该目录下，文件名类似于part-00000），需要确保HDFS中不存在这个目录，否则写入时会报“already exists”错误。

```
>>>
clean.write.format("csv").options(header='true',inferschema='true').save('E_Commerce_Data_Clean.csv')
```

最后，数据预处理完成。接下来将使用python编写应用程序，对清洗后的数据集进行统计分析。

3.3 统计分析

3.3.1 前序工作

创建project.py文件，用于编写应用程序。

首先，导入需要用到的python模块。

```
# -*- coding: utf-8 -*-
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType, DoubleType, IntegerType, StructField, StructType
import json
import os
```

接着，获取spark sql的上下文。

```
sc = SparkContext('local', 'spark_project')
sc.setLogLevel('WARN')
spark = SparkSession.builder.getOrCreate()
```

3.3.2 数据读取

从HDFS中以csv的格式读取清洗后的数据目录E_Commerce_Data_Clean.csv，程序会取出该目录下的所有数据文件，得到DataFrame对象，并创建临时视图data用于后续分析。

```
df = spark.read.format('csv').options(header='true', inferschema='true').load('E_Commerce_Data_Clean.csv')
df.createOrReplaceTempView("data")
```

3.3.3 结果数据保存

为方便统计结果的可视化，将结果导出为json文件供web页面渲染。使用save方法导出数据：

```
def save(path, data):
    with open(path, 'w') as f:
        f.write(data)
```

准备工作完成。接下来对数据进行分析，分为概览和关系两个部分。

3.3.4 数据分析

3.3.4.1 客户数最多的10个国家

每个客户由编号CustomerID唯一标识，所以客户的数量为COUNT(DISTINCT CustomerID)，再按照国家Country分组统计，根据客户数降序排序，筛选出10个客户数最多的国家。得到的countryCustomerDF为DataFrame类型，执行collect()方法即可将结果以数组的格式返回。

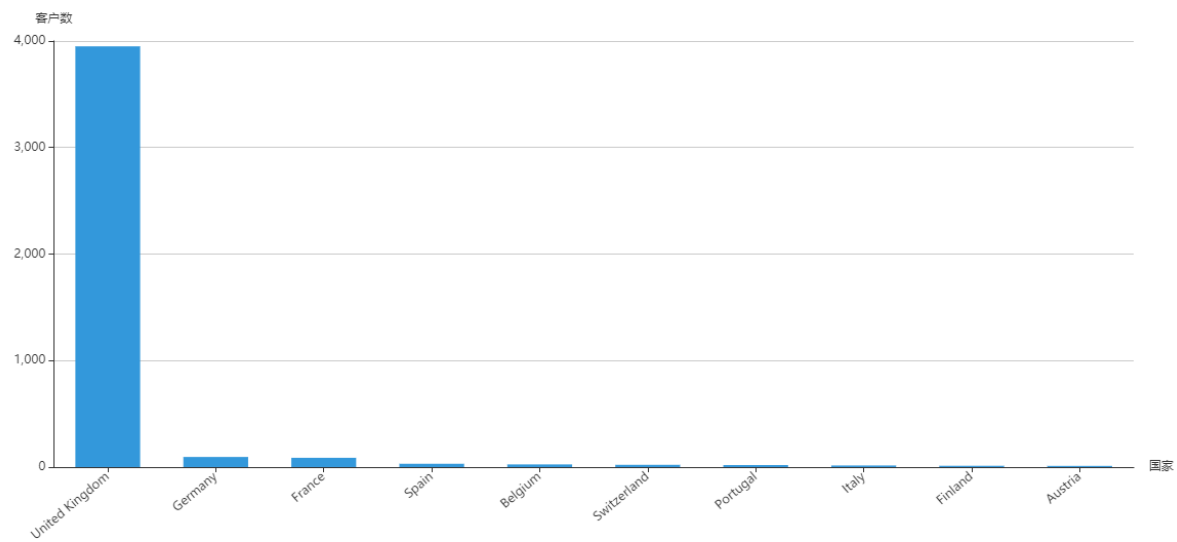
```
# 1 客户数最多的10个国家
'''
(1)每个客户由编号CustomerID唯一标识，所以客户的数量为COUNT(DISTINCT CustomerID)，
(2)再按照国家Country分组统计，
(3)根据客户数降序排序，筛选出10个客户数最多的国家
'''
def countryCustomer():
    countryCustomerDF = spark.sql(
    """
    SELECT Country,COUNT(DISTINCT CustomerID) AS countOfCustomer
    FROM data
    GROUP BY Country
    ORDER BY countOfCustomer DESC
    LIMIT 10
    """)
    # countryCustomerDF.show()
    return countryCustomerDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[国家名称,客户数]
```

结果截图：

分析：英国的客户最多，达到3950个，数量远大于其他国家；其次是德国、法国、西班牙等



3.3.4.2 销量最高的10个国家

Quantity字段表示销量，因为退货的记录中此字段为负数，所以使用SUM(Quantity)即可统计出总销量，即使有退货的情况。再按照国家Country分组统计，根据销量降序排序，筛选出10个销量最高的国家。得到的countryQuantityDF为DataFrame类型，执行collect()方法即可将结果以数组的格式返回。

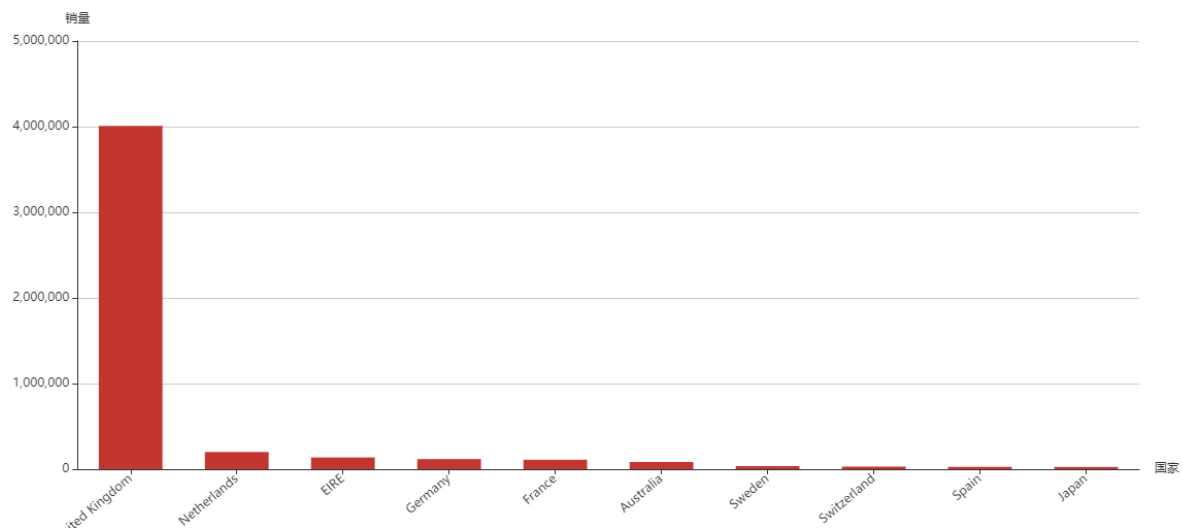
```
# 2 销量最高的10个国家
'''
(1)SUM(Quantity)即可统计出总销量，即使有退货的情况。
(2)再按照国家Country分组统计，
(3)根据销量降序排序，
(4)筛选出10个销量最高的国家。
'''
def countryQuantity():
    countryQuantityDF = spark.sql("""
    SELECT Country,SUM(Quantity) AS sumOfQuantity
    FROM data
    GROUP BY Country
    ORDER BY sumOfQuantity DESC
    LIMIT 10
    """)
    # countryQuantityDF.show()
    return countryQuantityDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[国家名称,销量]
```

结果分析：

英国的销量最高，达到4008533件，数量远大于其他国家；其次是新西兰、爱尔兰、德国等



3.3.4.3 各个国家的总销售额分布情况

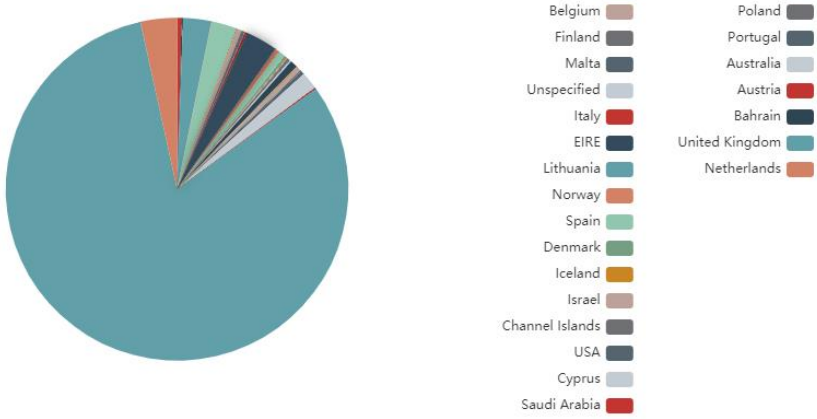
UnitPrice 字段表示单价，Quantity字段表示销量，退货的记录中Quantity字段为负数，所以使用SUM(UnitPrice*Quantity)即可统计出总销售额，即使有退货的情况。再按照国家Country分组统计，计算出各个国家的总销售额。得到的countrySumOfPriceDF为DataFrame类型，执行collect()方法即可将结果以数组的格式返回。

```
# 3 各个国家的总销售额分布情况
'''
(1)UnitPrice 字段表示单价，Quantity字段表示销量，退货的记录中Quantity字段为负数，
所以使用SUM(UnitPrice*Quantity)即可统计出总销售额，即使有退货的情况。
(2)再按照国家Country分组统计，计算出各个国家的总销售额。
'''
def countrySumOfPrice():
    countrySumOfPriceDF = spark.sql("""
    SELECT Country,SUM(UnitPrice*Quantity) AS sumOfPrice
    FROM data
    GROUP BY Country
    """)
    # countrySumOfPriceDF.show()
    return countrySumOfPriceDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[国家名称,总销售额]
```

英国的总销售额最高，达到6767873.394英镑，占比81.54%



3.3.4.4 销量最高的10个商品

Quantity字段表示销量，退货的记录中Quantity字段为负数，所以使用SUM(Quantity)即可统计出

总销量，即使有退货的情况。再按照商品编码StockCode分组统计，计算出各个商品的销量。得到的stockQuantityDF为DataFrame类型，执行collect()方法即可将结果以数组的格式返回。

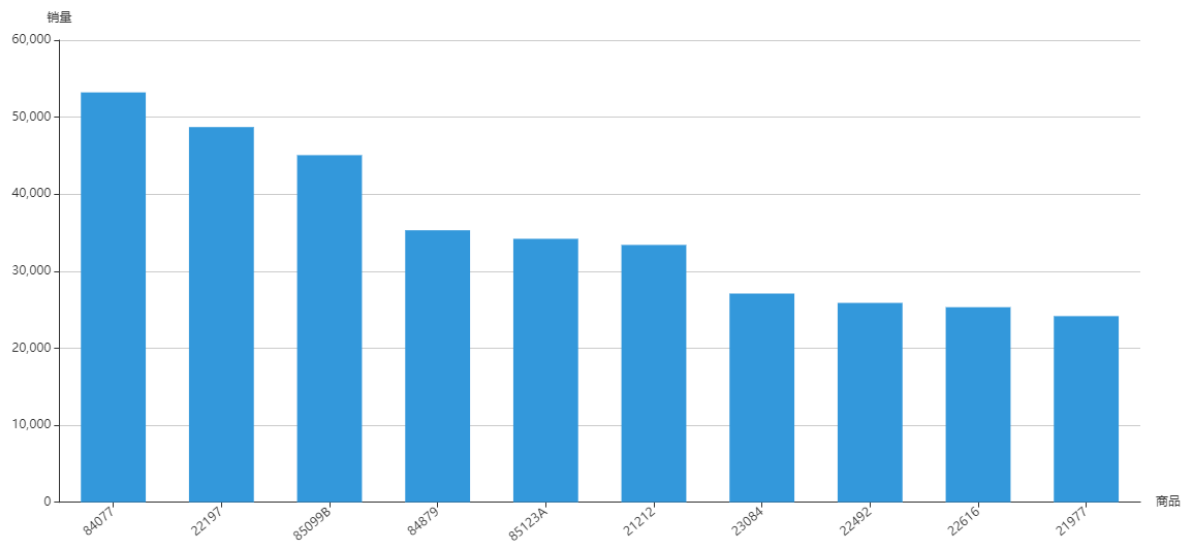
```
# 4 销量最高的10个商品
'''
(1)Quantity字段表示销量，退货的记录中Quantity字段为负数，所以使用SUM(Quantity)即可统计出总销量，即使有退货的情况。
(2)再按照商品编码StockCode分组统计，计算出各个商品的销量。
'''
def stockQuantity():
    stockQuantityDF = spark.sql(
        """
        SELECT StockCode,SUM(Quantity) AS sumOfQuantity
        FROM data
        GROUP BY StockCode
        ORDER BY sumOfQuantity
        DESC LIMIT 10"
        """)
    # stockQuantityDF.show()
    return stockQuantityDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[商品编号,销量]
```

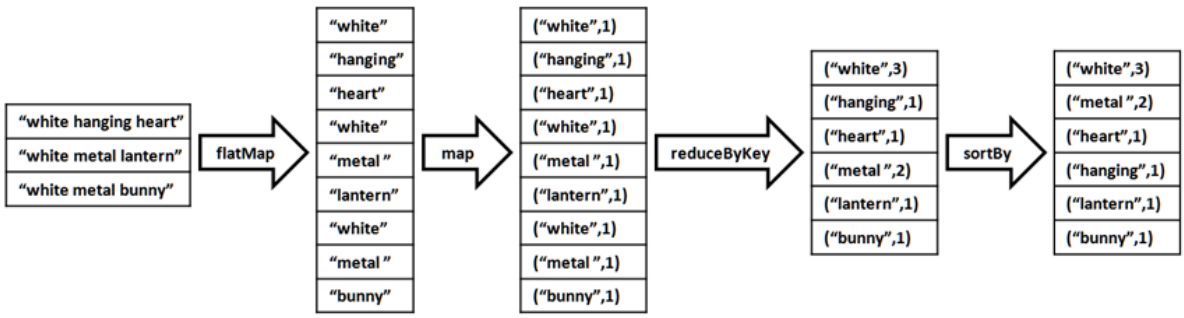
结果如下：

编号为84077的商品销量最高，达到53215件；销量Top3的商品在数量上差距并不大



3.3.4.5 商品描述的热门关键词Top300

Description字段表示商品描述，由若干个单词组成，使用LOWER(Description)将单词统一转换为小写。此时的结果为DataFrame类型，转化为rdd后进行词频统计，再根据单词出现的次数进行降序排序，流程图如下：



得到的结果为RDD类型，为其制作表头wordCountSchema，包含word和count属性，分别为string类型和integer类型。调用createDataFrame()方法将其转换为DataFrame类型的wordCountDF，将word为空字符串的记录剔除掉，调用take()方法得到出现次数最多的300个关键词，以数组的格式返回。

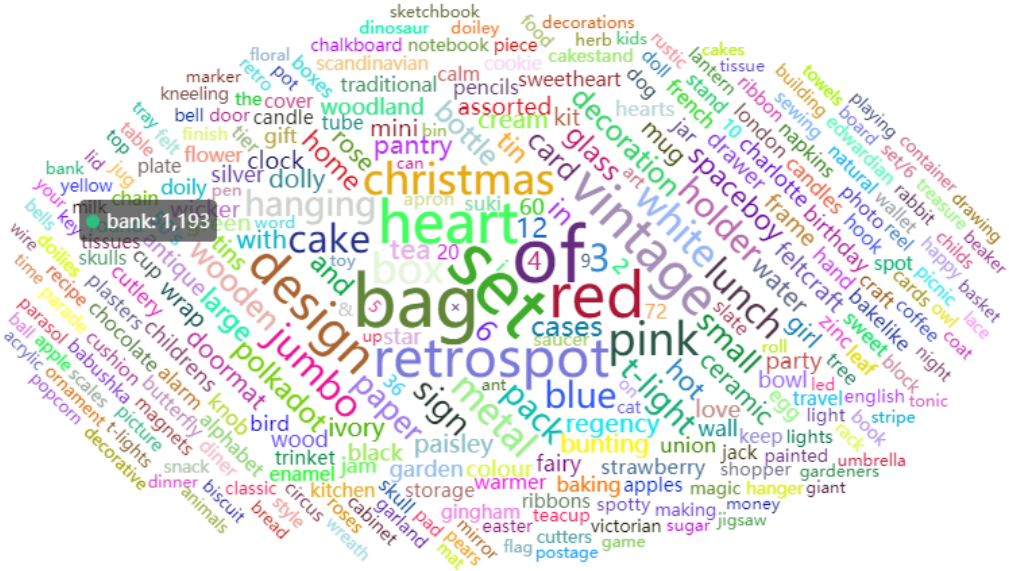
```
# 5 商品描述的热门关键词Top300
'''
(1)Description字段表示商品描述，由若干个单词组成，使用LOWER(Description)将单词统一转换为小写。
#SELECT LOWER(Description) as description from data
(2)此时的结果为DataFrame类型，使用df.withColumn生成words列，使用爆炸explode函数将单词扁平化
(3)利用words分组统计并根据count进行降序排序
(4)过滤掉空字符串，最后利用df.take(300)返回
'''

def wordCount():
    wordCountStep1 = spark.sql("SELECT LOWER(Description) as description from data")
    from pyspark.sql import functions as F
    df_new = wordCountStep1.withColumn("words", F.explode(F.split(F.col("description"), " ")))
    count_order_byDF = df_new.groupBy("words").count().orderBy("count", ascending=False)
    wordCountDF = count_order_byDF.filter(count_order_byDF["words"] != '')
    return wordCountDF.take(300)
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[关键词,次数]
```

结果分析：

热门关键词包括bag、red、heart、pink、christmas、cake等



3.3.4.6 退货订单数最多的10个国家

InvoiceNo字段表示订单编号，所以订单总数为COUNT(DISTINCT InvoiceNo)，由于退货订单的编号的首个字母为C，例如C540250，所以利用WHERE InvoiceNo LIKE 'C%'子句即可筛选出退货的订单，再按照国家Country分组统计，根据退货订单数降序排序，筛选出10个退货订单数最多的国家。得到的countryReturnInvoiceDF为DataFrame类型，执行collect()方法即可将结果以数组的格式返回。

```
# 6 退货订单数最多的10个国家
'''
（1）InvoiceNo字段表示订单编号，所以订单总数为COUNT(DISTINCT InvoiceNo)，
（2）由于退货订单的编号的首个字母为C，例如C540250，所以利用WHERE InvoiceNo LIKE 'C%'子句即可筛选出退货的订单，
（3）再按照国家Country分组统计，
（4）根据退货订单总数降序排序，
（5）筛选出10个退货订单数最多的国家。
'''
def countryReturnInvoice():
    countryReturnInvoiceDF = spark.sql("""
    SELECT Country,COUNT(DISTINCT InvoiceNo) AS countOfReturnInvoice
    FROM data
    WHERE InvoiceNo LIKE 'C%'
    GROUP BY Country
```

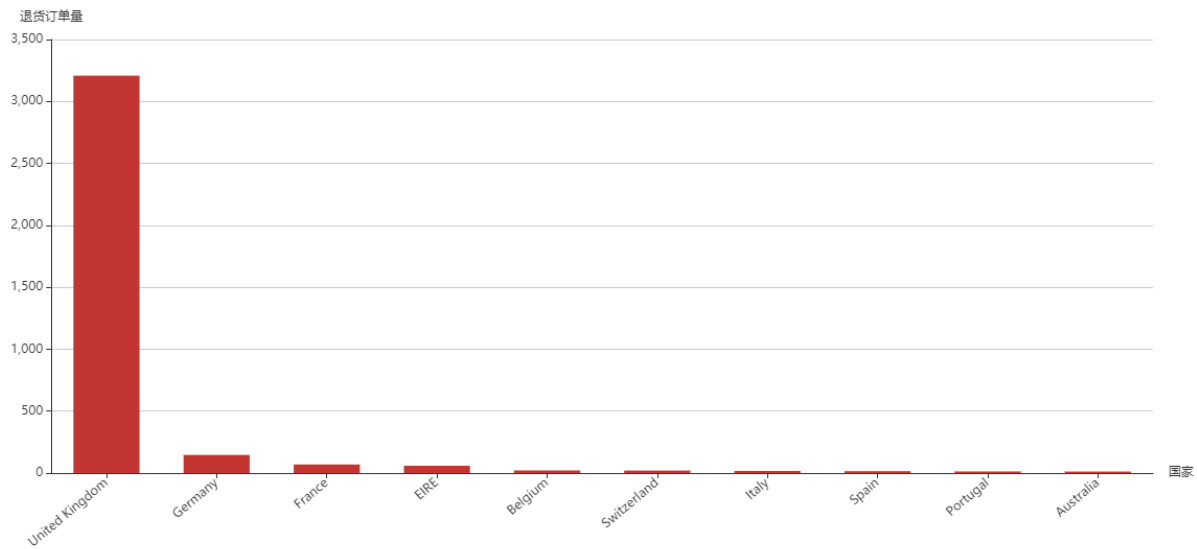
```
    ORDER BY countOfReturnInvoice DESC
    LIMIT 10
    """)
    # countryReturnInvoiceDF.show()
    return countryReturnInvoiceDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[国家名称,退货订单数]
```

结果如下：

英国的退货订单数最多，达到3208个，数量远大于其他国家；其次是德国、法国、爱尔兰等



3.3.4.7 商品的平均单价与销量的关系

由于商品的单价UnitPrice是不断变化的，所以使用平均单价AVG(DISTINCT UnitPrice)来衡量一个商品。再利用SUM(Quantity)计算出销量，将结果按照商品的编号进行分组统计，执行collect()方法即可将结果以数组的格式返回。

```
# 10 商品的平均单价与销量的关系
'''
(1)由于商品的单价UnitPrice是不断变化的，所以使用平均单价AVG(DISTINCT UnitPrice)来衡量一个商品。
(2)再利用SUM(Quantity)计算出销量，
(3)将结果按照商品的编号StockCode进行分组统计，
(4)执行collect()方法即可将结果以数组的格式返回。
'''
def unitPriceSales():
```

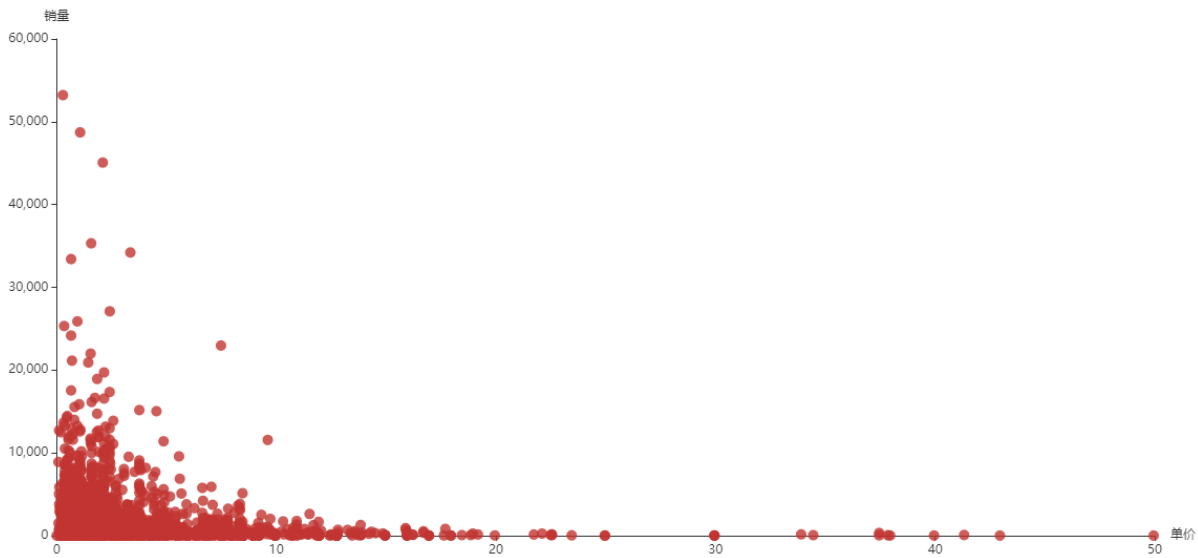
```
    unitPriceSalesDF = spark.sql(
        """
        SELECT StockCode,AVG(DISTINCT UnitPrice) AS avgUnitPrice,SUM(Quantity) AS sumOfQuantity
        FROM data
        GROUP BY StockCode"
        """)
    # unitPriceSalesDF.show()
    return unitPriceSalesDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[商品编号,平均单价,销量]
```

结果分析

总体上看，商品的销量随着平均单价的升高而下降



3.3.4.8 [练习]月销售额随时间的变化趋势

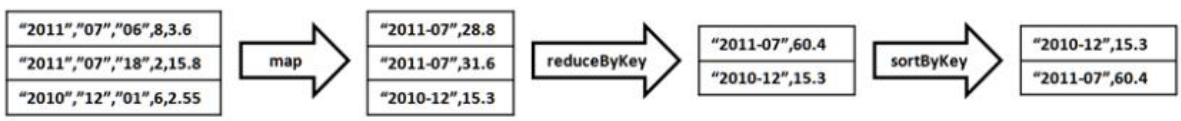
统计月销售额需要3个字段的信息，分别为订单日期InvoiceDate，销量Quantity和单价UnitPrice。由于InvoiceDate字段格式不容易处理，例如“8/5/2011 16:19”，所以需要对这个字段进行格式化操作。由于统计不涉及小时和分钟数，所以只截取年月日部分，并且当数值小于10时补前置0来统一格式，期望得到年、月、日3个独立字段。先实现formatData()方法，利用rdd对日期、销量和单价字段进行处理。

```
# 7和8的准备工作
```

```
def formatData():
    tradeRDD = df.select("InvoiceDate","Quantity","UnitPrice",).rdd
    result1 = tradeRDD.map(lambda line: (line['InvoiceDate'].split(" ")[0], line['Quantity'] , line['UnitPrice']))
    result2 = result1.map(lambda line: (line[0].split("/"), line[1], line[2]))
    result3 = result2.map(lambda line: (line[0][2], line[0][0] if len(line[0][0])==2 else "0"+line[0][0], line[0][1] if len(line[0][1])==2 else "0"+line[0][1], line[1], line[2]))
    return result3
```

流程图如下：

由于要统计的是月销售额的变化趋势，所以只需将日期转换为"2011-08"这样的格式即可。而销售额表示为单价乘以销量，需要注意的是，退货时的销量为负数，所以对结果求和可以表示销售额。RDD的转换流程如下：



得到的结果为RDD类型，为其制作表头schema，包含date和tradePrice属性，分别为string类型和double类型。调用createDataFrame()方法将其转换为DataFrame类型的tradePriceDF，调用collect()方法将结果以数组的格式返回。

```
# 7 月销售额随时间的变化趋势
'''
于要统计的是月销售额的变化趋势，所以只需将日期转换为"2011-08"这样的格式即可。
而销售额表示为单价乘以销量，需要注意的是，退货时的销量为负数，
所以对结果求和可以表示销售额。RDD的转换流程如下：
'''
def tradePrice():
    result3 = formatData()
    result4 = result3.map(lambda line:(line[0]+"-"+line[1],line[3]*line[4]))
    result5 = result4.reduceByKey(lambda a,b:a+b).sortByKey()
    schema = StructType([StructField("date", StringType(), True),StructField("tradePrice", DoubleType(), True)])
    tradePriceDF = spark.createDataFrame(result5, schema)
```

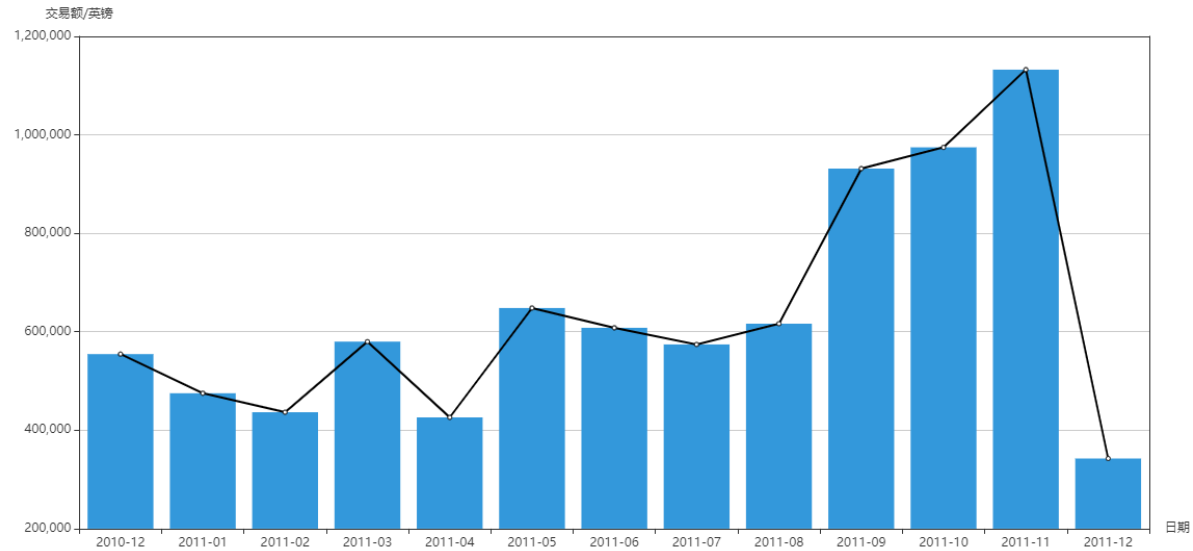
```
    # tradePriceDF.show()
    return tradePriceDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[日期,销售额]
```

结果：

销售额较高的月份主要集中在下半年；由于该公司主要售卖礼品，并且下半年的节日较多，所以销售额比上半年高；2011年12月的销售额较低是因为数据只统计到2011/12/9



3.3.4.9 [练习]日销量随时间的变化趋势

由于要统计的是日销量的变化趋势，所以只需将日期转换为"2011-08-05"这样的格式即可。先调用上例的formatData()方法对日期格式进行格式化。RDD的转换流程如下：

得到的结果为RDD类型，为其制作表头schema，包含date和saleQuantity属性，分别为string类型和integer类型。调用createDataFrame()方法将其转换为DataFrame类型的saleQuantityDF，调用collect()方法将结果以数组的格式返回。

```
```

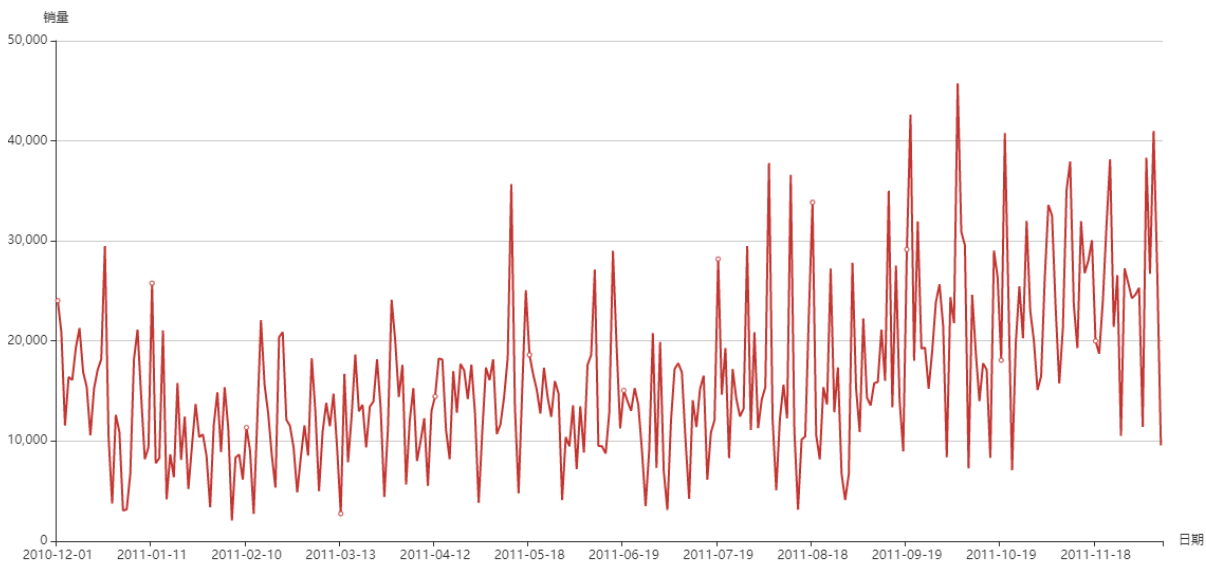
```
# 8 日销量随时间的变化趋势
'''
得到的结果为RDD类型，为其制作表头schema，包含date和saleQuantity属性，分别为string类型和integer类型。调用createDataFrame()方法将其转换为DataFrame类型的saleQuantityDF，调用collect()方法将结果以数组的格式返回。
'''
def saleQuantity():
    result3 = formatData()
    result4 = result3.map(lambda line:(line[0]+"-"+line[1]+"-"+line[2],line[3]))
    result5 = result4.reduceByKey(lambda a,b:a+b).sortByKey()
    schema = StructType([StructField("date", StringType(), True),StructField("saleQuantity", IntegerType(), True)])
    saleQuantityDF = spark.createDataFrame(result5, schema)
    # saleQuantityDF.show()
    return saleQuantityDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[日期,销量]
```

结果分析：

下半年的日销量整体上高于上半年；2011年的10月5号达到日销量的最高纪录45741件



3.3.4.10 [练习]各国的购买订单量和退货订单量的关系

InvoiceNo字段表示订单编号，退货订单的编号的首个字母为C，例如C540250。利用

COUNT(DISTINCT InvoiceNo)子句统计订单总量，再分别用WHERE InvoiceNo LIKE 'C%'和WHERE InvoiceNo NOT LIKE 'C%'统计出退货订单量和购买订单量。接着按照国家Country分组统计，得到的returnDF和buyDF均为DataFrame类型，分别表示退货订单和购买订单，如下所示：

returnDF

Country	countOfReturn
"Sweden"	10
"Singapore"	3
.....	.....

buyDF

Country2	countOfBuy
"Sweden"	36
"Singapore"	7
.....	.....

再对这两个DataFrame执行join操作，连接条件为国家Country相同，得到一个DataFrame。但是这个DataFrame中有4个属性，包含2个重复的国家Country属性和1个退货订单量和1个购买订单量，为减少冗余，对结果筛选3个字段形成buyReturnDF。如下所示：

buyReturnDF

Country	countOfReturn	Country2	countOfBuy
"Sweden"	10	"Sweden"	36
"Singapore"	3	"Singapore"	7
.....	.....	.....	.....

select

buyReturnDF

Country	countOfReturn	countOfBuy
"Sweden"	10	36
"Singapore"	3	7
.....	.....	.....

最后执行collect()方法即可将结果以数组的格式返回。

```
# 9 各国的购买订单量和退货订单量的关系
'''
(1)InvoiceNo字段表示订单编号，退货订单的编号的首个字母为C，例如C540250。利用COUNT(DISTINCT InvoiceNo)子句统计订单总量，
(2)再分别用WHERE InvoiceNo LIKE 'C%'和WHERE InvoiceNo NOT LIKE 'C%'统计出退货订单量和购买订单量。
(3)接着按照国家Country分组统计，得到的returnDF和buyDF均为DataFrame类型，分别表示退货订单和购买订单
(4)再对这两个DataFrame执行join操作，连接条件为国家Country相同，得到一个DataFrame。
(5)但是这个DataFrame中有4个属性，包含2个重复的国家Country属性和1个退货订单量和1个购买订单量，为减少冗余，对结果筛选3个字段形成buyReturnDF。
'''
def buyReturn():
    returnDF = spark.sql("SELECT Country AS Country,COUNT(DISTINCT InvoiceNo) AS countOfReturn FROM data WHERE InvoiceNo LIKE 'C%' GROUP BY Country")
    buyDF = spark.sql("SELECT Country AS Country2,COUNT(DISTINCT InvoiceNo) AS countOfBuy FROM data WHERE InvoiceNo NOT LIKE 'C%' GROUP BY Country2")
    buyReturnDF = returnDF.join(buyDF, returnDF["Country"] == buyDF["Country2"], "left_outer")
    buyReturnDF = buyReturnDF.select(buyReturnDF["Country"],buyReturnDF["countOfBuy"],buyReturnDF["countOfReturn"])
    # buyReturnDF.show()
```

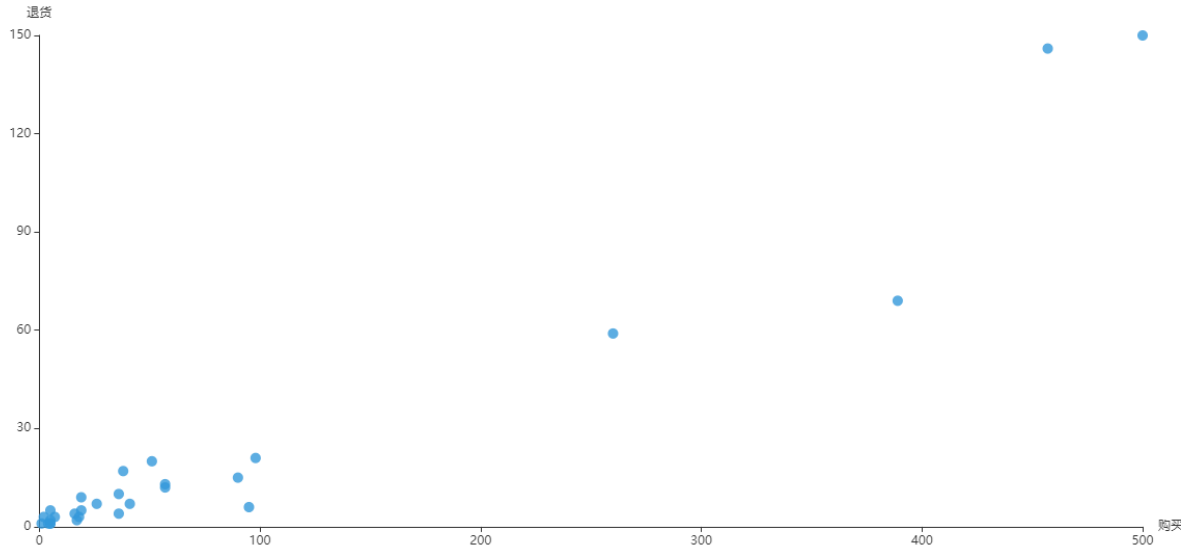
```
    return buyReturnDF.collect()
```

最后调用save方法就可以将结果导出至文件了，格式如下：

```
[国家名称,购买订单数,退货订单数]
```

结果分析

购买订单量越大的国家，退货订单量往往也越大



3.3.4.11 小结

在project.py中添加main函数，将上面的分析过程整合起来方便进行调用，代码如下：

```
if __name__ == "__main__":

    base = "static/"
    if not os.path.exists(base):
        os.mkdir(base)

    m = {
        "countryCustomer": {
            "method": countryCustomer,
            "path": "countryCustomer.json"
        },
        "countryQuantity": {
            "method": countryQuantity,
            "path": "countryQuantity.json"
        },
        "countrySumOfPrice": {
            "method": countrySumOfPrice,
            "path": "countrySumOfPrice.json"
        },
        "stockQuantity": {
```

```
            "method": stockQuantity,
            "path": "stockQuantity.json"
        },
        "wordCount": {
            "method": wordCount,
            "path": "wordCount.json"
        },
        "countryReturnInvoice": {
            "method": countryReturnInvoice,
            "path": "countryReturnInvoice.json"
        },
        "tradePrice": {
            "method": tradePrice,
            "path": "tradePrice.json"
        },
        "saleQuantity": {
            "method": saleQuantity,
            "path": "saleQuantity.json"
        },
        "buyReturn": {
            "method": buyReturn,
            "path": "buyReturn.json"
        },
        "unitPriceSales": {
            "method": unitPriceSales,
            "path": "unitPriceSales.json"
        }
    }

    for k in m:
        p = m[k]
        f = p["method"]
        save(base + m[k]["path"], json.dumps(f()))
        print ("done -> " + k + " , save to -> " + base + m[k]["path"])
```

上面的代码将所有的函数整合在变量 m中，通过循环调用上述所有方法并导出json文件到当前路径的static目录下。

最后利用如下指令运行分析程序：

```
cd /usr/local/spark
./bin/spark-submit project.py
```

3.3.4.12 数据开发所有代码

如下：

```
# -*- coding: utf-8 -*-
# Program function:
from pyspark import SparkContext
from pyspark.sql import SparkSession
```

```
from pyspark.sql.types import StringType, DoubleType, IntegerType, StructField, StructType
import json
import os

# 准备环境
sc = SparkContext('local', 'spark_project')
sc.setLogLevel('WARN')
spark = SparkSession.builder.getOrCreate()

# (1)从HDFS中读取数据集为DataFrame
df = spark.read.format('csv').options(header='true', inferschema='true').load(
    'file:///export/data/pyspark_workspace/ECommerce/data/E_Commerce_Data_Clean.csv')
df.createOrReplaceTempView("data")

# 1 客户数最多的10个国家
'''
(1)每个客户由编号CustomerID唯一标识，所以客户的数量为COUNT(DISTINCT CustomerID)，
(2)再按照国家Country分组统计，
(3)根据客户数降序排序，筛选出10个客户数最多的国家
'''


def countryCustomer():
    countryCustomerDF = spark.sql(
        """
        SELECT Country,COUNT(DISTINCT CustomerID) AS countOfCustomer
        FROM data
        GROUP BY Country
        ORDER BY countOfCustomer DESC
        LIMIT 10
        """)
    # countryCustomerDF.show()
    return countryCustomerDF.collect()


# 2 销量最高的10个国家
'''
(1)SUM(Quantity)即可统计出总销量，即使有退货的情况。
(2)再按照国家Country分组统计，
(3)根据销量降序排序，
(4)筛选出10个销量最高的国家。
'''


def countryQuantity():
    countryQuantityDF = spark.sql("""
    SELECT Country,SUM(Quantity) AS sumOfQuantity
    FROM data
    GROUP BY Country
```

```
    ORDER BY sumOfQuantity DESC
    LIMIT 10
    """)
    # countryQuantityDF.show()
    return countryQuantityDF.collect()


# 3 各个国家的总销售额分布情况
'''
(1)UnitPrice 字段表示单价，Quantity字段表示销量，退货的记录中Quantity字段为负数，
所以使用SUM(UnitPrice*Quantity)即可统计出总销售额，即使有退货的情况。
(2)再按照国家Country分组统计，计算出各个国家的总销售额。
'''


def countrySumOfPrice():
    countrySumOfPriceDF = spark.sql("""
    SELECT Country,SUM(UnitPrice*Quantity) AS sumOfPrice
    FROM data
    GROUP BY Country
    """)
    # countrySumOfPriceDF.show()
    return countrySumOfPriceDF.collect()


# 4 销量最高的10个商品
'''
(1)Quantity字段表示销量，退货的记录中Quantity字段为负数，所以使用SUM(Quantity)即可统计出总销量，即使有退货的情况。
(2)再按照商品编码StockCode分组统计，计算出各个商品的销量。
'''
def stockQuantity():
    stockQuantityDF = spark.sql(
        """
        SELECT StockCode,SUM(Quantity) AS sumOfQuantity
        FROM data
        GROUP BY StockCode
        ORDER BY sumOfQuantity
        DESC LIMIT 10"
        """)
    # stockQuantityDF.show()
    return stockQuantityDF.collect()

# 5 商品描述的热门关键词Top300
'''
(1)Description字段表示商品描述，由若干个单词组成，使用LOWER(Description)将单词统一转换为小写。
(2)此时的结果为DataFrame类型，转化为rdd后进行词频统计，再根据单词出现的次数进行降序排序，流程图如下：
'''
def wordCount():
    wordCountStep1 = spark.sql("SELECT LOWER(Description) as description from data")
```

```
    from pyspark.sql import functions as F
    df_new = wordCountStep1.withColumn("words", F.explode(F.split(F.col("description"), " ")))
    count_order_byDF = df_new.groupBy("words").count().orderBy("count", ascending=False)
    wordCountDF = count_order_byDF.filter(count_order_byDF["words"] != '')
    return wordCountDF.take(300)

# 6 退货订单数最多的10个国家
'''
 (1) InvoiceNo字段表示订单编号，所以订单总数为COUNT(DISTINCT InvoiceNo)，
 (2) 由于退货订单的编号的首个字母为C，例如C540250，所以利用WHERE InvoiceNo LIKE 'C%'子句即可筛选出退货的订单，
 (3) 再按照国家Country分组统计，
 (4) 根据退货订单总数降序排序，
 (5) 筛选出10个退货订单数最多的国家。
'''
def countryReturnInvoice():
    countryReturnInvoiceDF = spark.sql("""
    SELECT Country,COUNT(DISTINCT InvoiceNo) AS countOfReturnInvoice
    FROM data
    WHERE InvoiceNo LIKE 'C%'
    GROUP BY Country
    ORDER BY countOfReturnInvoice DESC
    LIMIT 10
    """)
    # countryReturnInvoiceDF.show()
    return countryReturnInvoiceDF.collect()
# 10 商品的平均单价与销量的关系
'''
(1)由于商品的单价UnitPrice是不断变化的，所以使用平均单价AVG(DISTINCT UnitPrice)来衡量一个商品。
(2)再利用SUM(Quantity)计算出销量，
(3)将结果按照商品的编号StockCode进行分组统计，
(4)执行collect()方法即可将结果以数组的格式返回。
'''
def unitPriceSales():
    unitPriceSalesDF = spark.sql(
        """
        SELECT StockCode,AVG(DISTINCT UnitPrice) AS avgUnitPrice,SUM(Quantity) AS sumOfQuantity
        FROM data
        GROUP BY StockCode"
        """)
    # unitPriceSalesDF.show()
    return unitPriceSalesDF.collect()

# 扩展部分
# 7和8的准备工作
def formatData():
    tradeRDD = df.select("InvoiceDate","Quantity","UnitPrice",).rdd
    result1 = tradeRDD.map(lambda line: (line['InvoiceDate'].split(" ")[0], line['Quantity'] , line['UnitPrice']))
    result2 = result1.map(lambda line: (line[0].split("/"), line[1], line[2]))
    result3 = result2.map(lambda line: (line[0][2], line[0][0] if len(line[0][0])==2 else "0"+line[0][0], line[0][1] if len(line[0][1])==2
```

```
else "0"+line[0][1], line[1], line[2]))
    return result3
# 7 月销售额随时间的变化趋势
'''
于要统计的是月销售额的变化趋势，所以只需将日期转换为"2011-08"这样的格式即可。
而销售额表示为单价乘以销量，需要注意的是，退货时的销量为负数，
所以对结果求和可以表示销售额。RDD的转换流程如下：
'''
def tradePrice():
    result3 = formatData()
    result4 = result3.map(lambda line:(line[0]+"-"+line[1],line[3]*line[4]))
    result5 = result4.reduceByKey(lambda a,b:a+b).sortByKey()
    schema = StructType([StructField("date", StringType(), True),StructField("tradePrice", DoubleType(), True)])
    tradePriceDF = spark.createDataFrame(result5, schema)
    # tradePriceDF.show()
    return tradePriceDF.collect()
# 8 日销量随时间的变化趋势
'''
得到的结果为RDD类型，为其制作表头schema，包含date和saleQuantity属性，分别为string类型和integer类型。
调用createDataFrame()方法将其转换为DataFrame类型的saleQuantityDF，调用collect()方法将结果以数组的格式返回。
'''
def saleQuantity():
    result3 = formatData()
    result4 = result3.map(lambda line:(line[0]+"-"+line[1]+"-"+line[2],line[3]))
    result5 = result4.reduceByKey(lambda a,b:a+b).sortByKey()
    schema = StructType([StructField("date", StringType(), True),StructField("saleQuantity", IntegerType(), True)])
    saleQuantityDF = spark.createDataFrame(result5, schema)
    # saleQuantityDF.show()
    return saleQuantityDF.collect()
if __name__ == '__main__':
    # print("# 1 客户数最多的10个国家",countryCustomer())
    # print("# 2 销量最高的10个国家",countryQuantity())
    # print("# 3 各个国家的总销售额分布情况", countrySumOfPrice())
    # print("# 4 销量最高的10个商品", countrySumOfPrice())
    # print("# 5 商品描述的热门关键词Top300", wordCount())
    # print("# 6 退货订单数最多的10个国家", countryReturnInvoice())
    # print("# 10 商品的平均单价与销量的关系", countryReturnInvoice())
    # print("# 7 月销售额随时间的变化趋势", tradePrice())
    print("# 8 日销量随时间的变化趋势", saleQuantity())
```

3.3.5 可视化方法

Echarts是一个纯Javascript的图表库，可以流畅地运行在PC和移动设备上，兼容当前绝大部分浏览器，底层依赖轻量级的Canvas类库ZRender，提供直观，生动，可交互，可高度个性化定制的数据

可视化图表。

编写web.py程序，实现一个简单的web服务器，代码如下：

```
from bottle import route, run, static_file
import json

@route('/static/<filename>')
def server_static(filename):
    return static_file(filename, root="./static")

@route("/<name:re:.*\.html>")
def server_page(name):
    return static_file(name, root=".")

@route("/")
def index():
    return static_file("index.html", root=".")

run(host="0.0.0.0", port=9999)
```

Python

bottle服务器对接收到的请求进行路由，规则如下：

（1）访问/static/时，返回静态文件

（2）访问/.html时，返回网页文件

（3）访问/时，返回首页index.html

服务器的9999端口监听来自任意ip的请求（前提是请求方能访问到这台服务器）。

首页index.html的主要代码如下(由于篇幅较大，只截取主要的部分)

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,height=device-height">
    <title>E-Commerce-Data 在线零售业务数据分析</title>
```

```
<style>
/* 省略 */
    </style>
</head>

<body>
    <div class="container">
    /* 只展示第一个统计结果的代码，其余省略 */
        <div class="chart-group">
            <h3>(1) 客户数最多的10个国家
            <br>
            <small style="font-size: 72%;">
                ——英国的客户最多，达到3950个，数量远大于其他国家；其次是德国、法国、西班牙等
            </small>
            </h3>
            <iframe src="countryCustomer.html" class="frame" frameborder="0"></iframe>
        </div>
    </div>
    <script>document.body.clientHeight;</script>
</body>
</html>
```

HTML

图表页通过一个iframe嵌入到首页中。以第一个统计结果的网页countryCustomer.html为例，展示主要代码：

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <style>
/* 省略 */
    </style>
</head>
```

```
<body>
<div id="chart" style="width:95%;height:95%;"></div>
<script src="static/jquery-3.2.1.min.js"></script>
<script src="static/echarts-4.7.0.min.js"></script>
<script>
    var myChart = echarts.init(document.getElementById('chart'));
    myChart.setOption(
    {
        color: ['#3398DB'],
        tooltip: {
            trigger: 'axis',
            axisPointer: {
                type: 'shadow'
            }
        },
        grid: {
            left: '3%',
            right: '4%',
            bottom: '3%',
            containLabel: true
        },
        xAxis: [
            {
                name: '国家',
                data: [],
                axisTick: {
                    alignWithLabel: true
                },
                axisLabel: {
                    interval:0,
                    rotate:40
                }
```

```
            }
        ],
        yAxis: [
            {
                name: '客户数',
            }
        ],
        series: [
            {
                name: '客户数',
                type: 'bar',
                barWidth: '60%',
                data: []
            }
        ]
});
myChart.showLoading();
$.getJSON("/static/countryCustomer.json", data => {
        var names=[];
        var nums=[];

        data = data.map(v => ({
            country: v[0],
            customer: parseInt(v[1]),
        }))

        for(var i=0;i<data.length;i++){
            names.push(data[i].country);
            nums.push(data[i].customer);
        }
        myChart.setOption({
            xAxis: {
                data: names
```

```
                },
                series: [{
                    data: nums
                }]
            });
            myChart.hideLoading();
        })
</script>
</body>
</html>
```

3.3.6 项目执行

代码完成后，在代码所在的根目录下执行以下指令启动web服务器：

```
python3 web.py
```

若打印出以下信息则表示web服务启动成功。接着，可以通过使用浏览器访问网页的方式查看统计结果。

```
Bottle v0.12.18 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:9999/
Hit Ctrl-C to quit.
```

为方便运行程序，编写run.sh脚本，内容如下。首先向spark提交project.py程序对数据进行统计分析，生成的json文件会存入当前路径的static目录下；接着运行web.py程序，即启动web服务器对分析程序生成的json文件进行解析渲染，方便用户通过浏览器查看统计结果的可视化界面。

```
#!/bin/bash
cd /usr/local/spark
./bin/spark-submit project.py
python3 web.py
```

如下：

```
(pyspark_env) [root@node1 main]# sh run.sh
2021-07-15 10:10:21,042 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... u
es where applicable
2021-07-15 10:10:21,621 INFO spark.SparkContext: Running Spark version 3.1.2
2021-07-15 10:10:21,652 INFO resource.ResourceUtils: ==============================================================
2021-07-15 10:10:21,653 INFO resource.ResourceUtils: No custom resources configured for spark.driver.
2021-07-15 10:10:21,653 INFO resource.ResourceUtils: ==============================================================
2021-07-15 10:10:21,653 INFO spark.SparkContext: Submitted application: spark_project
2021-07-15 10:10:21,674 INFO resource.ResourceProfile: Default ResourceProfile created, executor resources: Map
amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHe
, vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
2021-07-15 10:10:21,686 INFO resource.ResourceProfile: Limiting resource is cpu
2021-07-15 10:10:21,686 INFO resource.ResourceProfileManager: Added ResourceProfile id: 0
2021-07-15 10:10:21,721 INFO spark.SecurityManager: Changing view acls to: root
2021-07-15 10:10:21,721 INFO spark.SecurityManager: Changing modify acls to: root
2021-07-15 10:10:21,722 INFO spark.SecurityManager: Changing view acls groups to:
2021-07-15 10:10:21,722 INFO spark.SecurityManager: Changing modify acls groups to:
2021-07-15 10:10:21,722 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled;
ssions: Set(root); groups with view permissions: Set(); users  with modify permissions: Set(root); groups with
()
2021-07-15 10:10:21,880 INFO util.Utils: Successfully started service 'sparkDriver' on port 43528.
2021-07-15 10:10:21,909 INFO spark.SparkEnv: Registering MapOutputTracker
2021-07-15 10:10:21,937 INFO spark.SparkEnv: Registering BlockManagerMaster
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:9999/
Hit Ctrl-C to quit.
```

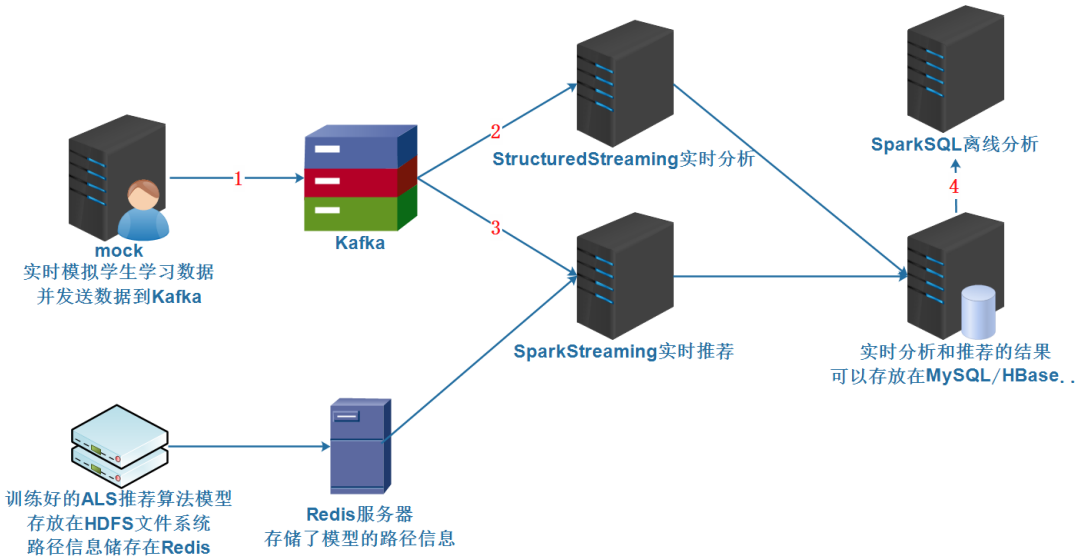
在线教育学生学习情况离线分析

1. 第一章 案例概述

1.1 案例背景

在互联网、移动互联网的带动下，教育逐渐从线下走向线上，在线教育近几年一直处于行业的风口浪尖，那随着基础设施的不断完善，用户需求也发生不少变化，因此传统教育机构、新兴互联网企业都在探索在线教育的最佳模式。随着在线教育的发展和普及，越来越多的选择在线教育，越来越多的公司也加入的竞争行列中。但是学生个人情况不同，影响学习效果/考试成绩的因素也众多，那么如何充分利用现有数据，对数据进行价值挖掘，找出影响学生学习效果/考试成绩的关键因素，并加以提升或改进，以提高教学效果，改善教学品质，提升学生考试成绩，这一需求已经成为各大在线教育企业亟需解决的问题也是广大学子的共同诉求。

1.2 架构流程



1.3 业务模块

2. 第二章 需求分析

- 需求1:各科目热点题分析

要求: 找到Top50热点题对应科目，然后统计这些科目中，分别包含这几道热点题的条目数

举例:

热点题

题号 热度 学科

1 100 数学

2 99 数学

3 98 语文

最终结果:

学科 热点题数量

数学 2

语文 1

- 需求2:各科目推荐题分析

要求: 找到Top20热点题对应的推荐题目，然后找到推荐题目对应的科目，并统计每个科目分别包含推荐题目的条数

举例:

热点题对应的学科和推荐题号

题号 热度 学科 推荐题号

1 100 数学 2,3,4

2 99 数学 3,4,5

3 98 语文 6,7,8

最终结果

学科 推荐题数量

数学 4道

语文 3道

复杂一点:有可能推荐的题不是同一个学科的

题号 热度 推荐题号

1 100 2,3,4

2 99 3,4,5

3 98 6,7,8

推荐题号 对应学科

2 数学

3 语文

4 数学

5 数学

6 数学

7 语文

8 语文

最终结果

学科 推荐的热点题数量

数学 4

语文 3

3. 第三章 数据描述

```
'''
root
 |-- student_id: string (nullable = true)
 |-- recommendations: string (nullable = true)
 |-- textbook_id: string (nullable = true)
 |-- grade_id: string (nullable = true)
 |-- subject_id: string (nullable = true)
 |-- chapter_id: string (nullable = true)
 |-- question_id: string (nullable = true)
 |-- score: integer (nullable = true)
 |-- answer_time: string (nullable = true)
 |-- ts: timestamp (nullable = true)
'''
jdbcDF.printSchema()
'''
+----------+----------------------+----------+--------+------------+---------------+-----------+-----+-------------------+-------------------+
|student_id|       recommendations|textbook_id|grade_id|  subject_id|     chapter_id|question_id|score|        answer_time|                 ts|
+----------+----------------------+----------+--------+------------+---------------+-----------+-----+-------------------+-------------------+
| 学生ID_20|题目ID_1656,题目ID_75...|  教材ID_2|年级ID_6|科目ID_1_数学|章节ID_chapter_3|题目ID_2022|    7|2020-12-08 19:43:02|2020-12-08 19:43:02|
| 学生ID_31|题目ID_1824,题目ID_66...|  教材ID_1|年级ID_1|科目ID_1_数学|章节ID_chapter_3| 题目ID_58|    0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_36|题目ID_1775,题目ID_28...|  教材ID_1|年级ID_1|科目ID_3_英语|章节ID_chapter_2| 题目ID_149|    0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_22|题目ID_484,题目ID_115...|  教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_1|题目ID_1921|   10|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_5|科目ID_3_英语|章节ID_chapter_3| 题目ID_892|    6|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_5|科目ID_2_语文|章节ID_chapter_3| 题目ID_836|   10|2020-12-08 19:43:04|2020-12-08 19:43:04|
```

```
| 学生ID_37|题目ID_643,题目ID_567...|    教材ID_1|年级ID_2|科目ID_3_英语|章节ID_chapter_1| 题目ID_310|     5|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_48|题目ID_604,题目ID_394...|    教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_2|题目ID_1949|     0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_48|题目ID_604,题目ID_394...|    教材ID_2|年级ID_6|科目ID_3_英语|章节ID_chapter_2|题目ID_2124|     3|2020-12-08 19:43:04|2020-12-08 19:43:04|
|  学生ID_5|题目ID_763,题目ID_861...|    教材ID_1|年级ID_6|科目ID_1_数学|章节ID_chapter_2| 题目ID_940|     2|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_17|题目ID_345,题目ID_128...|    教材ID_2|年级ID_5|科目ID_2_语文|章节ID_chapter_2|题目ID_1888|     2|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_17|题目ID_345,题目ID_128...|    教材ID_1|年级ID_6|科目ID_2_语文|章节ID_chapter_3|题目ID_1003|     4|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_17|题目ID_345,题目ID_128...|    教材ID_1|年级ID_5|科目ID_2_语文|章节ID_chapter_3| 题目ID_830|     0|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_18|题目ID_628,题目ID_213...|    教材ID_1|年级ID_2|科目ID_3_英语|章节ID_chapter_2| 题目ID_336|     1|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_35|题目ID_234,题目ID_586...|    教材ID_1|年级ID_6|科目ID_1_数学|章节ID_chapter_1| 题目ID_912|    10|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_49|题目ID_42,题目ID_525,...|    教材ID_2|年级ID_3|科目ID_1_数学|章节ID_chapter_1|题目ID_1447|     9|2020-12-08 19:43:03|2020-12-08 19:43:03|
|  学生ID_8|题目ID_516,题目ID_142...|    教材ID_1|年级ID_4|科目ID_3_英语|章节ID_chapter_2| 题目ID_690|     8|2020-12-08 19:43:03|2020-12-08 19:43:03|
|  学生ID_9|题目ID_1474,题目ID_88...|    教材ID_2|年级ID_3|科目ID_2_语文|章节ID_chapter_3|题目ID_1549|     6|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_11|题目ID_1376,题目ID_19...|    教材ID_2|年级ID_2|科目ID_2_语文|章节ID_chapter_3|题目ID_1380|     7|2020-12-08 19:43:02|2020-12-08 19:43:02|
| 学生ID_13|题目ID_1093,题目ID_91...|    教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_1|题目ID_1927|     5|2020-12-08 19:43:03|2020-12-08 19:43:03|
+----------+-----------------------+------------+--------+-------------+---------------+-----------+------+-------------------+-------------------+
only showing top 20 rows
'''
```

4. 第四章 统计分析

4.1 SparkSQL离线分析入口程序

代码如下：

```
# -*- coding: utf-8 -*-
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import StringType, DoubleType, IntegerType, StructField, StructType
import json
import os

os.environ['SPARK_HOME'] = '/export/servers/spark'
PYSPARK_PYTHON = "/root/anaconda3/envs/pyspark_env/bin/python"
# 当存在多个版本时，不指定很可能会导致出错
os.environ["PYSPARK_PYTHON"] = PYSPARK_PYTHON
os.environ["PYSPARK_DRIVER_PYTHON"] = PYSPARK_PYTHON

sc = SparkContext('local', 'spark_project')
sc.setLogLevel('WARN')
spark = SparkSession.builder.getOrCreate()
```

4.2 加载数据

如下代码：

```
# Loading data from a JDBC source
jdbcDF = spark.read \
      .format("jdbc") \
      .option("driver", "com.mysql.jdbc.Driver") \
      .option("url",
"jdbc:mysql://node1:3306/?serverTimezone=UTC&characterEncoding=utf8&useUnicode=true") \
      .option("dbtable", "edu.t_recommended") \
      .option("user", "root") \
      .option("password", "123456") \
      .load()
jdbcDF.show()
'''
root
 |-- student_id: string (nullable = true)
 |-- recommendations: string (nullable = true)
 |-- textbook_id: string (nullable = true)
 |-- grade_id: string (nullable = true)
 |-- subject_id: string (nullable = true)
 |-- chapter_id: string (nullable = true)
 |-- question_id: string (nullable = true)
 |-- score: integer (nullable = true)
 |-- answer_time: string (nullable = true)
 |-- ts: timestamp (nullable = true)
'''
jdbcDF.printSchema()
'''
+----------+--------------------+-----------+--------+------------+---------------+-----------+-----+-------------------+-------------------+
|student_id|     recommendations|textbook_id|grade_id|  subject_id|     chapter_id|question_id|score|        answer_time|                 ts|
+----------+--------------------+-----------+--------+------------+---------------+-----------+-----+-------------------+-------------------+
| 学生ID_20|题目ID_1656,题目ID_75...|  教材ID_2|年级ID_6|科目ID_1_数学|章节ID_chapter_3|题目ID_2022|    7|2020-12-08 19:43:02|2020-12-08 19:43:02|
| 学生ID_31|题目ID_1824,题目ID_66...|  教材ID_1|年级ID_1|科目ID_1_数学|章节ID_chapter_3| 题目ID_58|    0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_36|题目ID_1775,题目ID_28...|  教材ID_1|年级ID_1|科目ID_3_英语|章节ID_chapter_2| 题目ID_149|    0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_22|题目ID_484,题目ID_115...|  教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_1|题目ID_1921|   10|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_5|科目ID_3_英语|章节ID_chapter_3| 题目ID_892|    6|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_5|科目ID_2_语文|章节ID_chapter_3| 题目ID_836|   10|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_2|科目ID_3_英语|章节ID_chapter_1| 题目ID_310|    5|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_48|题目ID_604,题目ID_394...|  教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_2|题目ID_1949|    0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_48|题目ID_604,题目ID_394...|  教材ID_2|年级ID_6|科目ID_3_英语|章节ID_chapter_2|题目ID_2124|    3|2020-12-08 19:43:04|2020-12-08 19:43:04|
|  学生ID_5|题目ID_763,题目ID_861...|  教材ID_1|年级ID_6|科目ID_1_数学|章节ID_chapter_2| 题目ID_940|    2|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_17|题目ID_345,题目ID_128...|  教材ID_2|年级ID_5|科目ID_2_语文|章节ID_chapter_2|题目ID_1888|    2|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_17|题目ID_345,题目ID_128...|  教材ID_1|年级ID_6|科目ID_2_语文|章节ID_chapter_3|题目ID_1003|    4|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_17|题目ID_345,题目ID_128...|  教材ID_1|年级ID_5|科目ID_2_语文|章节ID_chapter_3| 题目ID_830|    0|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_18|题目ID_628,题目ID_213...|  教材ID_1|年级ID_2|科目ID_3_英语|章节ID_chapter_2| 题目ID_336|    1|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_35|题目ID_234,题目ID_586...|  教材ID_1|年级ID_6|科目ID_1_数学|章节ID_chapter_1| 题目ID_912|   10|2020-12-08 19:43:04|2020-12-08 19:43:04|
```

```
| 学生ID_49|题目ID_42,题目ID_525,...|    教材ID_2|年级ID_3|科目ID_1_数学|章节ID_chapter_1|题目ID_1447|      9|2020-12-08 19:43:03|2020-12-08 19:43:03|
|  学生ID_8|题目ID_516,题目ID_142...|    教材ID_1|年级ID_4|科目ID_3_英语|章节ID_chapter_2| 题目ID_690|      8|2020-12-08 19:43:03|2020-12-08 19:43:03|
|  学生ID_9|题目ID_1474,题目ID_88...|    教材ID_2|年级ID_3|科目ID_2_语文|章节ID_chapter_3|题目ID_1549|      6|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_11|题目ID_1376,题目ID_19...|    教材ID_2|年级ID_2|科目ID_2_语文|章节ID_chapter_3|题目ID_1380|      7|2020-12-08 19:43:02|2020-12-08 19:43:02|
| 学生ID_13|题目ID_1093,题目ID_91...|    教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_1|题目ID_1927|      5|2020-12-08 19:43:03|2020-12-08 19:43:03|
+----------+------------------------+------------+--------+-------------+---------------+-----------+-------+-------------------+-------------------+
only showing top 20 rows
'''
```

或者使用csv数据导入

```
# 从Mysql导出的数据以\t进行分割
data = spark.read.format("csv") \
      .option("header", True) \
      .option("sep", "\t") \
      .load("file:///export/data/pyspark_workspace/EduAnalysis/data/eduxxx.csv")
print(data.printSchema())
jdbcDF = data
```

4.3 离线业务分析1:各科目热点题分析

代码如下：

```
# 3-业务实现:
# TODO需求1: 各科目热点题分析
"""
要求: 找到Top50热点题对应的科目，然后统计这些科目中，分别包含这几道热点题的条目数
热点题
题号 热度(数量) 学科
1    100   数学
2    99    数学
3    98    语文

最终结果:
学科   热点题数量
数学   2
语文 1
"""
# 方式1: SQL风格
allInfoDS = jdbcDF
allInfoDS.createOrReplaceTempView("t_answer")
spark.sql(
      """
```

```
    SELECT
    subject_id, count(t_answer.question_id) AS hot_question_count
    FROM
    (SELECT question_id, count(*) AS frequency
    FROM t_answer
    GROUP BY question_id
    ORDER BY frequency  DESC LIMIT 50) t1
    JOIN t_answer ON t1.question_id = t_answer.question_id
    GROUP BY subject_id
    ORDER BY hot_question_count DESC
    """).show()
# +-------------+------------------+
# |    subject_id|hot_question_count|
# +-------------+------------------+
# |科目ID_3_英语|               299|
# |科目ID_1_数学|               295|
# |科目ID_2_语文|               260|
# +-------------+------------------+
# 方式2:DSL风格
# 1.Top50热点题
# top50DS: Dataset[Row(题目id,题目热度)]
# 根据question_id进行排序，进行count统计，在根据count降序排序得到Top50热点题目top50DS
top50DS = allInfoDS.groupBy("question_id").count().orderBy("count", ascending=False).limit(50)
# 2.和原始数据进行关联,得到题目所属学科,热点题数量
# top50DS与原始的jdbcDF根据问题id进行Join，在根据学科id-subject_id进行分组，count统计，得到结果
top50DS.join(allInfoDS, "question_id")\ #join之后得到题目id，题目数量(热度)，学科id
.groupBy("subject_id").count().orderBy("count", ascending=False).show()
# +-------------+-----+
# |    subject_id|count|
# +-------------+-----+
# |科目ID_3_英语|  299|  热点题目数量有多少道题目
# |科目ID_1_数学|  295|
# |科目ID_2_语文|  260|
# +-------------+-----+
```

4.4 离线业务分析2:各科目推荐题分析

代码如下:

```
# TODO 需求2:各科目推荐题分析
# 要求: 找到Top20热点题对应的推荐题目，然后找到推荐题目对应的科目，并统计每个科目分别包含推荐题目的条数
"""
热点题对应的学科和推荐题号
题号  热度  学科    推荐题号
1      100   数学     234
2      99     数学     345
3      98     语文     678
最终结果
学科     推荐题数量
数学        4道
语文        3道

复杂一点:有可能推荐的题不是同一个学科的
题号  热度     推荐题号
1      100        234
2      99         345
3      98         678
推荐题号  对应学科
2             数学
3             语文
4             数学
5             数学
6             数学
7             语文
8             语文
最终结果
学科    推荐的热点题数量
数学        4
语文        3
"""
spark.sql(
    """SELECT t4.subject_id,COUNT(*) AS frequency
    FROM
    (SELECT
    DISTINCT(t3.question_id),t_answer.subject_id
    FROM
    (SELECT EXPLODE(SPLIT(t2.recommendations, ',')) AS question_id
```

```
        FROM
        (SELECT recommendations
        FROM
        (SELECT question_id,COUNT(*) AS frequency
        FROM t_answer
        GROUP BY question_id
        ORDER BY frequency
        DESC LIMIT 20) t1
        JOIN t_answer
        ON t1.question_id = t_answer.question_id) t2) t3
        JOIN t_answer
        ON t3.question_id = t_answer.question_id) t4
        GROUP BY t4.subject_id
        ORDER BY    frequency DESC
        """).show()
# 方式2:DSL风格
# 1.统计热点题Top20--子查询t1       得到top20DS[题目id,题目热度]
top20DS = allInfoDS.groupBy("question_id").count().orderBy('count',ascending=False).limit(20)
# 2.将t1和原始表t_answer关联,得到热点题Top20的推荐题列表t2
# 题目id,推荐列表
# recommendListDF[题目id,推荐列表题目ids字符串]
# 如[题目ID_999,推荐列表ids字符串:题目ID_44,题目ID_232,题目ID_2118,题目ID_1265]
recommendListDF = top20DS.join(allInfoDS, "question_id")
# 3.用SPLIT将recommendations中的ids用","切为数组,然后用EXPLODE将列转行,并记为t3
# 将上面的数据推荐列表ids字符串:
# 题目ID_44,题目ID_232,题目ID_2118,题目ID_1265
# 变为:
# 题目id
# 题目ID_44
# 题目ID_232
# 题目ID_2118
# 题目ID_1265
# 先用,号切割,再用explod将一行变为多行
# Creates a new row for each element in the given array or map column.
# 为给定数组或映射列中的每个元素创建新行。得到推荐的题目
questionIdDF = recommendListDF.select(F.explode(F.split("recommendations",",")).alias("question_id"))
print("questionIdDF show")
#questionIdDF.show()
# 4.对推荐的题目进行去重,将t3和t_answer原始表进行join,得到每个推荐的题目所属的科目,记为t4
# questionIdAndSubjectIdDF[推荐的题号,学科]
# dropDuplicates案例注意分析
questionIdAndSubjectIdDF =
questionIdDF.distinct().join(allInfoDS.dropDuplicates(["question_id"]),on="question_id").select("question_id",
"subject_id")
```

```
# 5.统计各个科目包含的推荐的题目数量并倒序排序(已去重)
questionIdAndSubjectIdDF.groupBy("subject_id").count().orderBy("count",ascending=False).show()
'''
# +-------------+-----+
# |subject_id     |count|
# +-------------+-----+
# |科目ID_3_英语|106   |
# |科目ID_1_数学|91    |
# |科目ID_2_语文|69    |
# +-------------+-----+
# '''
```

4.5 完整代码

```
# -*- coding: utf-8 -*-
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import StringType, DoubleType, IntegerType, StructField, StructType
import json
import os

os.environ['SPARK_HOME'] = '/export/servers/spark'
PYSPARK_PYTHON = "/root/anaconda3/envs/pyspark_env/bin/python"
# 当存在多个版本时，不指定很可能会导致出错
os.environ["PYSPARK_PYTHON"] = PYSPARK_PYTHON
os.environ["PYSPARK_DRIVER_PYTHON"] = PYSPARK_PYTHON

sc = SparkContext('local', 'spark_project')
sc.setLogLevel('WARN')
spark = SparkSession.builder.getOrCreate()

# Loading data from a JDBC source
jdbcDF = spark.read \
    .format("jdbc") \
    .option("driver", "com.mysql.jdbc.Driver") \
    .option("url",
"jdbc:mysql://node1:3306/?serverTimezone=UTC&characterEncoding=utf8&useUnicode=true") \
```

```
        .option("dbtable", "edu.t_recommended") \
        .option("user", "root") \
        .option("password", "123456") \
        .load()
jdbcDF.show()
'''
root
 |-- student_id: string (nullable = true)
 |-- recommendations: string (nullable = true)
 |-- textbook_id: string (nullable = true)
 |-- grade_id: string (nullable = true)
 |-- subject_id: string (nullable = true)
 |-- chapter_id: string (nullable = true)
 |-- question_id: string (nullable = true)
 |-- score: integer (nullable = true)
 |-- answer_time: string (nullable = true)
 |-- ts: timestamp (nullable = true)
'''
jdbcDF.printSchema()
'''
+----------+--------------------+-----------+--------+------------+---------------+-----------+-----+---
---------------+-------------------+
|student_id|     recommendations|textbook_id|grade_id|  subject_id|
chapter_id|question_id|score|    answer_time|                 ts|
+----------+--------------------+-----------+--------+------------+---------------+-----------+-----+---
---------------+-------------------+
| 学生ID_20|题目ID_1656,题目ID_75...|  教材ID_2|年级ID_6|科目ID_1_数学|章节ID_chapter_3|题目ID_2022|
7|2020-12-08 19:43:02|2020-12-08 19:43:02|
| 学生ID_31|题目ID_1824,题目ID_66...|  教材ID_1|年级ID_1|科目ID_1_数学|章节ID_chapter_3|  题目ID_58|
0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_36|题目ID_1775,题目ID_28...|  教材ID_1|年级ID_1|科目ID_3_英语|章节ID_chapter_2| 题目ID_149|
0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_22|题目ID_484,题目ID_115...|  教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_1|题目ID_1921|
10|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_5|科目ID_3_英语|章节ID_chapter_3| 题目ID_892|
6|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_5|科目ID_2_语文|章节ID_chapter_3| 题目ID_836|
10|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_37|题目ID_643,题目ID_567...|  教材ID_1|年级ID_2|科目ID_3_英语|章节ID_chapter_1| 题目ID_310|
5|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_48|题目ID_604,题目ID_394...|  教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_2|题目ID_1949|
0|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_48|题目ID_604,题目ID_394...|  教材ID_2|年级ID_6|科目ID_3_英语|章节ID_chapter_2|题目ID_2124|
3|2020-12-08 19:43:04|2020-12-08 19:43:04|
```

```
|   学生ID_5|题目ID_763,题目ID_861...|     教材ID_1|年级ID_6|科目ID_1_数学|章节ID_chapter_2| 题目ID_940|
2|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_17|题目ID_345,题目ID_128...|     教材ID_2|年级ID_5|科目ID_2_语文|章节ID_chapter_2|题目ID_1888|
2|2020-12-08 19:43:03|2020-12-08 19:43:03|
| 学生ID_17|题目ID_345,题目ID_128...|     教材ID_1|年级ID_6|科目ID_2_语文|章节ID_chapter_3|题目ID_1003|
4|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_17|题目ID_345,题目ID_128...|     教材ID_1|年级ID_5|科目ID_2_语文|章节ID_chapter_3| 题目ID_830|
0|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_18|题目ID_628,题目ID_213...|     教材ID_1|年级ID_2|科目ID_3_英语|章节ID_chapter_2| 题目ID_336|
1|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_35|题目ID_234,题目ID_586...|     教材ID_1|年级ID_6|科目ID_1_数学|章节ID_chapter_1| 题目ID_912|
10|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_49|题目ID_42,题目ID_525,...|     教材ID_2|年级ID_3|科目ID_1_数学|章节ID_chapter_1|题目ID_1447|
9|2020-12-08 19:43:03|2020-12-08 19:43:03|
|   学生ID_8|题目ID_516,题目ID_142...|     教材ID_1|年级ID_4|科目ID_3_英语|章节ID_chapter_2| 题目ID_690|
8|2020-12-08 19:43:03|2020-12-08 19:43:03|
|   学生ID_9|题目ID_1474,题目ID_88...|     教材ID_2|年级ID_3|科目ID_2_语文|章节ID_chapter_3|题目ID_1549|
6|2020-12-08 19:43:04|2020-12-08 19:43:04|
| 学生ID_11|题目ID_1376,题目ID_19...|     教材ID_2|年级ID_2|科目ID_2_语文|章节ID_chapter_3|题目ID_1380|
7|2020-12-08 19:43:02|2020-12-08 19:43:02|
| 学生ID_13|题目ID_1093,题目ID_91...|     教材ID_2|年级ID_5|科目ID_3_英语|章节ID_chapter_1|题目ID_1927|
5|2020-12-08 19:43:03|2020-12-08 19:43:03|
+----------+----------------------+-----------+--------+------------+---------------+-----------+-----+---
---------------+------------------+
only showing top 20 rows
'''
# 3-业务实现:
# TODO需求1: 各科目热点题分析
"""
要求: 找到Top50热点题对应的科目，然后统计这些科目中，分别包含这几道热点题的条目数
热点题
题号 热度 学科
1    100   数学
2    99    数学
3    98    语文

最终结果:
学科   热点题数量
数学   2
语文 1
"""
# 方式1: SQL风格
allInfoDS = jdbcDF
allInfoDS.createOrReplaceTempView("t_answer")
```

```
spark.sql(
    """
    SELECT
    subject_id, count(t_answer.question_id) AS hot_question_count
    FROM
    (SELECT question_id, count(*) AS frequency
    FROM t_answer
    GROUP BY question_id
    ORDER BY frequency
    DESC LIMIT 50) t1
    JOIN t_answer
    ON t1.question_id = t_answer.question_id
    GROUP BY subject_id
    ORDER BY hot_question_count
    DESC
    """).show()
# +-------------+------------------+
# |    subject_id|hot_question_count|
# +-------------+------------------+
# |科目ID_3_英语|               299|
# |科目ID_1_数学|               295|
# |科目ID_2_语文|               260|
# +-------------+------------------+
# 方式2:DSL风格
# 1.Top50热点题
# top50DS: Dataset[Row(题目id,题目热度)]
top50DS = allInfoDS.groupBy("question_id").count().orderBy("count", ascending=False).limit(50)
# 2.和原始数据进行管理,得到题目所属学科,热点题数量
top50DS.join(allInfoDS, "question_id").groupBy("subject_id").count().orderBy("count", ascending=False).show()
# +-------------+-----+
# |    subject_id|count|
# +-------------+-----+
# |科目ID_3_英语|  299|
# |科目ID_1_数学|  295|
# |科目ID_2_语文|  260|
# +-------------+-----+
# TODO 需求2:各科目推荐题分析
# 要求: 找到Top20热点题对应的推荐题目，然后找到推荐题目对应的科目，并统计每个科目分别包含推荐题目的条数
"""
热点题对应的学科和推荐题号
题号 热度 学科   推荐题号
1    100  数学    234
2    99   数学    345
```

```
3	98	语文	678
最终结果
学科	推荐题数量
数学	4道
语文	3道

复杂一点:有可能推荐的题不是同一个学科的
题号 热度	推荐题号
1	100	234
2	99	345
3	98	678
推荐题号 对应学科
2	数学
3	语文
4	数学
5	数学
6	数学
7	语文
8	语文
最终结果
学科	推荐的热点题数量
数学	4
语文	3
"""
spark.sql(
    """SELECT t4.subject_id,COUNT(*) AS frequency
    FROM
    (SELECT
    DISTINCT(t3.question_id),t_answer.subject_id
    FROM
    (SELECT EXPLODE(SPLIT(t2.recommendations, ',')) AS question_id
    FROM
    (SELECT recommendations
    FROM
    (SELECT question_id,COUNT(*) AS frequency
    FROM t_answer
    GROUP BY question_id
    ORDER BY frequency
    DESC LIMIT 20) t1
    JOIN t_answer
    ON t1.question_id = t_answer.question_id) t2) t3
    JOIN t_answer
    ON t3.question_id = t_answer.question_id) t4
    GROUP BY t4.subject_id
```

```
        ORDER BY    frequency DESC
        """).show()
# 方式2:DSL风格
# 1.统计热点题Top20--子查询t1
# top20DS[题目id,题目热度]
top20DS = allInfoDS.groupBy("question_id").count().orderBy('count',ascending=False).limit(20)
# 2.将t1和原始表t_answer关联,得到热点题Top20的推荐题列表t2
# 题目id,推荐列表
# recommendListDF[题目id,推荐列表题目ids字符串]
# 如:[题目ID_999,推荐列表ids字符串:题目ID_44,题目ID_232,题目ID_2118,题目ID_1265]
recommendListDF = top20DS.join(allInfoDS, "question_id")
# 3.用SPLIT将recommendations中的ids用","切为数组,然后用EXPLODE将列转行,并记为t3
# 将上面的数据推荐列表ids字符串:
# 题目ID_44,题目ID_232,题目ID_2118,题目ID_1265
# 变为:
# 题目id
# 题目ID_44
# 题目ID_232
# 题目ID_2118
# 题目ID_1265
# 先用,号切割,再用explod将一行变为多行
# Creates a new row for each element in the given array or map column.
# 为给定数组或映射列中的每个元素创建新行。
questionIdDF = recommendListDF.select(F.explode(F.split("recommendations",",")).alias("question_id"))
print("questionIdDF show")
#questionIdDF.show()
# 4.对推荐的题目进行去重,将t3和t_answer原始表进行join,得到每个推荐的题目所属的科目,记为t4
# questionIdAndSubjectIdDF[推荐的题号,学科]
# dropDuplicates案例注意分析
questionIdAndSubjectIdDF =
questionIdDF.distinct().join(allInfoDS.dropDuplicates(["question_id"]),on="question_id").select("question_id",
"subject_id")
# 5.统计各个科目包含的推荐的题目数量并倒序排序(已去重)
questionIdAndSubjectIdDF.groupBy("subject_id").count().orderBy("count",ascending=False).show()
'''
# +-------------+-----+
# |subject_id     |count|
# +-------------+-----+
# |科目ID_3_英语|106    |
# |科目ID_1_数学|91     |
# |科目ID_2_语文|69     |
# +-------------+-----+
# '''
```

4.6 测试

- 各科目热点题分析

```
+-------------+------------------+
|   subject_id|hot_question_count|
+-------------+------------------+
|科目ID_3_英语|                34|
|科目ID_1_数学|                25|
|科目ID_2_语文|                16|
+-------------+------------------+
```

- 各科目推荐题分析

```
+-------------+---------+
|   subject_id|frequency|
+-------------+---------+
|科目ID_2_语文|       24|
|科目ID_1_数学|       18|
|科目ID_3_英语|       18|
+-------------+---------+
```

[实战作业]零售业务离线分析案例

1. 第一章 案例概述

1.1 案例背景

某公司是做零售相关业务，旗下出品各类收银机.

目前公司的收银机已经在全国铺开，在各个省份均有店铺使用.

机器是联网的，每一次使用都会将售卖商品数据上传到公司后台.

老板现在想对省份维度进行统计分析

2. 第二章 需求分析

4个需求开发

1. 各省 销售 指标 每个省份的销售额统计

2. TOP3 销售省份中，有多少家店铺 日均销售额 1000+

3. TOP3 省份中 各个省份的平均单单价

4. TOP3 省份中，各个省份的支付类型比例

2个操作

1. 将需求结果写出到mysql

2. 将数据写入到Spark On Hive中

需求1和3比较简单

需求2比较复杂需求4略复杂

3. 第三章 数据描述

数据存放在课程资料: mini.json


```
{
    "discountRate": 1,
    "storeShopNo": "None",
    "dayOrderSeq": 9,
    "storeDistrict": "雨花区",
    "isSigned": 0,
    "storeProvince": "湖南省",
    "origin": 0,
    "storeGPSLongitude": "113.0008864402771",
    "discount": 0,
    "storeID": 2057,
    "productCount": 2,
    "operatorName": "OperatorName",
    "operator": "NameStr",
    "storeStatus": "open",
    "storeOwnUserTel": 12345678910,
    "payType": "cash",
    "discountType": 2,
    "storeName": "禄兴食品商店",
    "storeOwnUserName": "OwnUserNameStr",
    "dateTS": 1550553826000,
    "smallChange": 0,
    "storeGPSName": "None",
    "erase": 0,
    "product": [
```

```
        {
            "count": 1,
            "name": "HA夫山泉天然水1L",
            "unitID": 2,
            "barcode": "6921168520015",
            "pricePer": 3,
            "retailPrice": 3,
            "tradePrice": 0,
            "categoryID": 11
        },
        {
            "count": 1,
            "name": "百事可乐600ml",
            "unitID": 2,
            "barcode": "6924882496116",
            "pricePer": 3,
            "retailPrice": 3,
            "tradePrice": 0,
            "categoryID": 11
        }
    ],
    "storeGPSAddress": "None",
    "orderID": "155055382510220575799",
    "moneyBeforeWholeDiscount": 6,
    "storeCategory": "normal",
    "receivable": 6,
    "faceID": "",
    "storeOwnUserId": 1999,
    "paymentChannel": 0,
    "paymentScenarios": "OTHER",
    "storeAddress": "StoreAddress",
    "totalNoDiscount": 6,
    "payedTotal": 6,
    "storeGPSLatitude": "28.172069610039234",
    "storeCreateDateTS": 1541483175000,
    "storeCity": "长沙市",
    "memberID": "0"
}
```


Schema信息

```
root
 |-- corporator: string (nullable = true)
 |-- dateTS: long (nullable = true)
 |-- dayOrderSeq: long (nullable = true)
 |-- discount: double (nullable = true)
 |-- discountRate: long (nullable = true)
 |-- discountType: long (nullable = true)
 |-- erase: double (nullable = true)
 |-- faceID: string (nullable = true)
 |-- isSigned: long (nullable = true)
 |-- memberID: string (nullable = true)
 |-- moneyBeforeWholeDiscount: double (nullable = true)
 |-- operator: string (nullable = true)
 |-- operatorName: string (nullable = true)
 |-- orderID: string (nullable = true)
 |-- origin: long (nullable = true)
 |-- payStatus: long (nullable = true)
 |-- payType: string (nullable = true)
```

```
|-- payedTotal: double (nullable = true)
|-- paymentChannel: long (nullable = true)
|-- paymentScenarios: string (nullable = true)
|-- product: array (nullable = true)
|    |-- element: struct (containsNull = true)
|    |    |-- barcode: string (nullable = true)
|    |    |-- categoryID: long (nullable = true)
|    |    |-- count: long (nullable = true)
|    |    |-- name: string (nullable = true)
|    |    |-- pricePer: double (nullable = true)
|    |    |-- retailPrice: double (nullable = true)
|    |    |-- tradePrice: double (nullable = true)
|    |    |-- unitID: long (nullable = true)
|-- productCount: double (nullable = true)
|-- receivable: double (nullable = true)
|-- serverSaved: boolean (nullable = true)
|-- smallChange: double (nullable = true)
|-- storeAddress: string (nullable = true)
|-- storeCategory: string (nullable = true)
|-- storeCity: string (nullable = true)
|-- storeCreateDateTS: long (nullable = true)
|-- storeDistrict: string (nullable = true)
|-- storeGPSAddress: string (nullable = true)
|-- storeGPSLatitude: string (nullable = true)
|-- storeGPSLongitude: string (nullable = true)
|-- storeGPSName: string (nullable = true)
|-- storeID: long (nullable = true)
|-- storeName: string (nullable = true)
|-- storeOwnUserId: long (nullable = true)
|-- storeOwnUserName: string (nullable = true)
|-- storeOwnUserTel: long (nullable = true)
|-- storeProvince: string (nullable = true)
|-- storeShopNo: string (nullable = true)
|-- storeStatus: string (nullable = true)
|-- totalNoDiscount: double (nullable = true)
```

尽管字段很多，但是我们用到的只需要几个即可:

1. storeProvince 省份
2. storeID 店铺ID

3. receivable 订单营收金额

4. dateTS 订单时间戳

5. payType 订单支付类型

额外：这份数据是数据产生就是大宽表的一类形式，比较方便使用.

但是会丢失相关的维度信息表. 需要单独的维度 可能需要从数据中导出 或者重新去加载额外的维度数据.

4. 第四章 统计分析

4.1 数据清洗

过滤掉:

1. 单价大于10000的订单

2. 省份为字符串null的数据

```
spark = SparkSession.builder¥
    .config("spark.sql.shuffle.paratitions", "4")¥
    .appName("test")¥
    .master("local[*]")¥
    .getOrCreate()
sc = spark.sparkContext
df = spark.read.json("data/mini.json").¥
    filter("receivable < 10000 and storeProvince != 'null'").¥
    dropna(thresh=3, subset=['storeProvince', 'storeCity', 'storeDistrict'])
```

4.2 需求1-各省 销售 指标 每个省份的销售额统计

省分组，求和销售额，排序金额 即可

```
# 各省 销售 指标
province_sale_money_df = df.groupBy("storeProvince").sum('receivable').¥
    withColumnRenamed("sum(receivable)", "money").¥
    withColumn('money', F.round('money', 2)).¥
    orderBy('money', ascending=False)
```

4.3 需求2-TOP3 销售省份中，有多少家店铺 日均销售额 1000+

4.3.1 基础准备

1. 先基于需求1的结果，找到TOP3的省份
2. 然后，和原有表进行JOIN
3. 得到只有这3个省份的数据
4. 将JOIN结果 加入缓存

```
top3_province_df = province_sale_money_df.limit(3).select('storeProvince').withColumnRenamed("storeProvince", "top3_province")
top3_province_joined_df = df.join(top3_province_df, top3_province_df['top3_province'] == df['storeProvince'], "inner")
# 缓存
top3_province_joined_df.cache()
```

4.3.2 需求2代码

基于上面缓存的DF

```
top3_province_joined_df.groupBy('storeProvince', 'storeID', F.from_unixtime(df['dateTS'].substr(0, 10), 'yyyy-MM-dd').alias('day')). ¥
    sum('receivable').¥
    filter('sum(receivable) > 1000').¥
    dropDuplicates(subset=['storeID']).¥
    groupBy("storeProvince").count().show()
```

4.4 需求3-TOP3 省份中 各个省份的平均单单价

```
# TOP3 省份中 各个省份的平均单单价
top3_province_joined_df.groupBy("storeProvince").avg('receivable').show()
```

4.5 需求4-TOP3 省份中，各个省份的支付类型比例

需要用到开窗函数，也就是SQL风格和DSL风格混搭

同时需要注册一个UDF

```
# TOP3 省份中，各个省份的支付类型比例
def to_percent(data):
    return str(round(data * 100)) + '%'
udf_to_percent = F.udf(to_percent, StringType())
top3_province_joined_df.select("storeProvince", "payType").createTempView("province_paytype")
spark.sql("""
    SELECT storeProvince, payType, (COUNT(*) / pcnt) AS percent FROM (SELECT *, COUNT(*) OVER(PARTITION BY storeProvince) AS pcnt FROM province_paytype) AS sub
    GROUP BY storeProvince, payType, pcnt
""").select('storeProvince', 'payType', udf_to_percent('percent')).show()
```

4.6 清除缓存

```
top3_province_joined_df.unpersist()
```

4.7 完整代码

```
# coding:utf8

from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import StringType

if __name__ == '__main__':
    spark = SparkSession.builder¥
        .config("spark.sql.shuffle.paratitions", "4")¥
        .appName("test")¥
        .master("local[*]")¥
        .getOrCreate()
    sc = spark.sparkContext

    df = spark.read.json("data/mini.json").¥
        filter("receivable < 10000 and storeProvince != 'null'").¥
        dropna(thresh=3, subset=['storeProvince', 'storeCity', 'storeDistrict'])
    # df.printSchema()
```

```
    # 各省 销售 指标
    province_sale_money_df = df.groupBy("storeProvince").sum('receivable').¥
        withColumnRenamed("sum(receivable)", "money").¥
        withColumn('money', F.round('money', 2)).¥
        orderBy('money', ascending=False)

    # TOP3 销售省份中，有多少家店铺 日均销售额 1000+
    top3_province_df = province_sale_money_df.limit(3).select('storeProvince').withColumnRenamed("storeProvince", "top3_province")
    top3_province_joined_df = df.join(top3_province_df, top3_province_df['top3_province'] == df['storeProvince'], "inner")
    # 缓存
    top3_province_joined_df.cache()
    top3_province_joined_df.groupBy('storeProvince', 'storeID', F.from_unixtime(df['dateTS'].substr(0, 10), 'yyyy-MM-dd').alias('day')). ¥
        sum('receivable').¥
        filter('sum(receivable) > 1000').¥
        dropDuplicates(subset=['storeID']).¥
        groupBy("storeProvince").count().show()

    # TOP3 省份中 各个省份的平均单单价
    top3_province_joined_df.groupBy("storeProvince").avg('receivable').show()

    # TOP3 省份中，各个省份的支付类型比例
    def to_percent(data):
        return str(round(data * 100)) + '%'

    udf_to_percent = F.udf(to_percent, StringType())

    top3_province_joined_df.select("storeProvince", "payType").createTempView("province_paytype")
    spark.sql("""
        SELECT storeProvince, payType, (COUNT(*) / pcnt) AS percent FROM (SELECT *, COUNT(*) OVER(PARTITION BY storeProvince) AS pcnt FROM
province_paytype) AS sub
        GROUP BY storeProvince, payType, pcnt
    """).select('storeProvince', 'payType', udf_to_percent('percent')).show()

    top3_province_joined_df.unpersist()

    spark.stop()
```