Data Science Course Exercise Workbook

Introduction to In-Memory DEDPQ: SPARK

Course Material

Stuart Brown, May 2017

Data Science – SPARK

Stuart Brown, May 2017

08

**Fall**

Contents

[Summary and Intro 3](#_Toc485109605)

[Section 1: Stocks Data 3](#_Toc485109606)

[Exercise 1: Load the Data 3](#_Toc485109607)

[Exercise 2: Query the Stock Data 4](#_Toc485109608)

[Section 2: Email Messages 8](#_Toc485109609)

[Exercise 1: Prying Eyes 8](#_Toc485109610)

[Summary 10](#_Toc485109611)

# Summary and Intro

We all know by now that SPARK is amazing, it is super-fast at processing our queries due to keeping much of its operations in-memory. As another example, imagine you have a petabyte of data (10¹⁵, literally a quadrillion bytes) the first time load into memory may take a while, but all subsequent querying will happen in milliseconds depending on the cluster size.

This workbook aims to aid and facilitate Data Science trainees learning. In particular this workbook will look for the trainees to utilize SPARK and associated Scala language.

For this workbook the trainees will need to use the following datasets:

1. Stocks
2. Email\_Messages

# Section 1: Stocks Data

This section will focus on the 6 Stock data files, please complete all exercises.

**Scenario:**

Actually “getting to grips” is pretty good for this data set.

**Data:**

For the following exercises in this section we will be using the Stocks data. This data is in 6 CSV files: appl\_daily, face\_daily, nasdaq\_daily, nflx\_daily, twtr\_daily, yhoo\_daily.

Each of these files has the same following attributes: Date, Open, High, Low, Close, Volume, Adj\_Close.

# **Exercise 1: Load the Data**

In order for best performance of our queries through SPARK we will first need to load the data.

#### Question 1:

Load the 6 stock data files using sc.textFile. You will then need to map this to a case class(), also filter off the header row.

#### Answer

**val apple = sc.textFile("hdfs://localhost:54310/user/stuart.brown/apple/appl\_daily.csv")**

**case class appClass(date:String, open:Double, high:Double, low:Double, close:Double, volume:Long, adjClose:Double)**

**val header = apple.first()**

**val appleT = apple.filter(row => row != header)**

**val appleData = appleT.map(x=>x.split(",")).map(x=>appClass(x(0).toString,x(1).toDouble,x(2).toDouble,x(3).toDouble,x(4).toDouble,x(5).toLong,x(6).toDouble))**

**OR**

**val appTbl = apple.mapPartitionsWithIndex{(idx, iter) => if (idx == 0) iter.drop(1) else iter}**

**val appData = appTbl.map(x=>x.split(",")).map(x=>appClass(x(0).toString,x(1).toDouble,x(2).toDouble,x(3).toDouble,x(4).toDouble,x(5).toLong,x(6).toDouble))**

Repeat for the other 5 files.

# **Exercise 2: Query the Stock Data**

This exercise is formed of 9 questions that will involve querying the data that you have just loaded.

Please write the commands/code you used. It will be beneficial!

#### Question 1:

To kick off our analysis of the Stock data let’s see how the Highs of Apple have changed over time. Map the columns first to then sortByKey.

#### Answer

**appData.map(x=>(x.date,x.high)).sortByKey(true,1).foreach(println)**

#### Question 2:

Great, now let’s join together all the datasets by the date inclusively. You may need to split and convert to string.

#### Answer

**val appleRDD = apple.map(rec => (rec.split(",")(0).toString,rec))**

**val facebookRDD = facebook.map(rec => (rec.split(",")(0).toString,rec))**

**val nasdaqRDD = nasdaq.map(rec => (rec.split(",")(0).toString,rec))**

**val netflixRDD = netflix.map(rec => (rec.split(",")(0).toString,rec))**

**val twitterRDD = twitter.map(rec => (rec.split(",")(0).toString,rec))**

**val yahooRDD = yahoo.map(rec => (rec.split(",")(0).toString,rec))**

**val fbjoin = appleRDD.join(facebookRDD)**

**val nasjoin = fbjoin.join(nasdaqRDD)**

**val netjoin = nasjoin.join(netflixRDD)**

**val twjoin = netjoin.join(twitterRDD)**

**val joined = twjoin.join(yahooRDD)**

**joined.foreach(println)**

#### Question 3:

Gaining some measure on the performance of the stock value, what is the average difference between high and adj\_close of each company across all years? Map the mean to the loaded data.

#### Answer

**appData.map(x=>(x.high – x.adjClose)).mean**

repeat/union for the other stock data sets

#### Question 4:

Union your answers from question 3, what is the average difference between high and adj\_close of all the companies together?

#### Answer

**Case class test(company:String, value:Double)**

**val u = appData.map(x=>(“app”, x.high - x.adjClose)).union(fbData.map(x=>(“fb”,x.high - x.adjClose))).union(nasData.map(x=>(“nas”,x.high - x.adjClose))).union(netData.map(x=>(“net”,x.high - x.adjClose))).union(twData.map(x=>(“tw”,x.high - x.adjClose))).union(yahData.map(x=>(“ya”,x.high - x.adjClose)))**

**val utest = u.map(x => test(x.\_1.toString, x.\_2.toDouble))**

**val utest1 = utest.map(x => (x.company, x.value.toInt)).combineByKey((x:Int) => (x,1), (acc:(Int,Int),x) => (acc.\_1 + x, acc.\_2 + 1), (acc1:(Int,Int), acc2:(Int,Int)) => (acc1.\_1 + acc2.\_1, acc1.\_2 + acc2.\_2))**

**val uavg = utest1.map(x => (x.\_1, “%.2f”.format(x.\_2.\_1.toDouble / x.\_2.\_2).toDouble))**

**uavg.foreach(println)**

**or**

**val unionDiff = appData.map(x=>(x.high - x.adjClose)).union(fbData.map(x=>(x.high - x.adjClose))).union(nasData.map(x=>(x.high - x.adjClose))).union(netData.map(x=>(x.high - x.adjClose))).union(twData.map(x=>(x.high - x.adjClose))).union(yahData.map(x=>(x.high - x.adjClose)))**

**unionDiff.mean**

#### Question 5:

It is also interesting to find out how high each company peaked. For this, what was each company’s greatest high and lowest low?

#### Answer

**val maxHighA = appData.map((x=>(x.high))).max**

**val minLowA = appData.map((x=>(x.low))).min**

And repeat for the others

#### Question 6:

While interesting to know what the greatest highs and lowest lows were, to give it substantial context, when did these occur? We have the filter condition and we will need to map it to the date.

#### Answer

**val maxHighTimeA = appData.filter(x=>(x.high == maxHighA)).map(\_.date).foreach(println)**

**val minHighTimeA = appData.filter(x=>(x.high == minHighA)).map(\_.date).foreach(println)**

And repeat for the others

#### Question 7:

Alter your code from the previous questions to find out when the greatest volumes were traded for each company.

#### Answer

**val maxVolA = appData.map(x=>(x.volume)).max**

**val maxVolTimeA = appData.filter(x=>(x.volume== maxVolA)).map(\_.date).foreach(println)**

Repeat for the other companies

#### Question 8:

Did these trades cause the open to increase or decrease, and by how much?

#### Answer

**val nextTrade = appData.filter(x => (x.date >= maxVolTimeA)).map(x =>(x.date, x.open)).sortByKey(true,1).take(2).foreach(println)**

**Or**

**val maxVolA = appData.map(x=>(x.volume)).max**

**val maxVolD = appData.filter(x=>(x.volume == maxVolA))**

**maxVolD.collect**

**val maxOpen = maxVolD.map(x=>(x.open))**

**maxOpen.collect** (28.187461)

**val nextDay = appData.filter(x.date == “2000-10-02”)).collect**

**val nextOpen = nextDay.map(x=>(x.open))** (26.6875)

Apple decreased repeat for others

#### Question 9:

We have been asked by another department to send on all the stock data, do this by union-ing all 6 data sets and include another column for the company they refer too?

#### Answer

**val union = appData.map(x=>(x.date, x.open, x.high, x.low, x.close, x.volume, x.adjClose, "Apple")).union(fbData.map(x=>(x.date, x.open, x.high, x.low, x.close, x.volume, x.adjClose, "Facebook"))).union(nasData.map(x=>(x.date, x.open, x.high, x.low, x.close, x.volume, x.adjClose, "Nasdaq"))).union(netData.map(x=>(x.date, x.open, x.high, x.low, x.close, x.volume, x.adjClose, "Netflix"))).union(twData.map(x=>(x.date, x.open, x.high, x.low, x.close, x.volume, x.adjClose, "Twitter"))).union(yahData.map(x=>(x.date, x.open, x.high, x.low, x.close, x.volume, x.adjClose, "Yahoo")))**

**union.collect**

# Section 2: Email Messages

This section will focus on the email messages data file, please complete all exercises.

**Scenario:**

Think of some sort of explanation to validate the reason for prying into people’s emails.

**Data:**

For the following exercises in this section we will be using the email messages data file. Within the file you will be able to determine: Mail id, Message Id, Date, From, To, Subject and message text. You will need to convert the .ods to a .csv file.

# **Exercise 1: Prying Eyes**

This exercise is formed of 6 questions that will involve querying the data that you have just loaded.

Please write the commands/code you used. It will be beneficial!

Be careful, not even your emails are safe.

#### Question 1:

Phillip Allen is one of our newest project managers coordinating our latest project. We would like to ensure he is keeping good contact with the right people. Who received the most emails from Mr. Allen?

It is interesting to note that sort by values, only keys, to overcome this you can map to a swap and then reduce by the value which is not the key.

#### Answer

**val email = sc.textFile("/home/stuart.brown/Desktop/Email\_Messages2.txt")**

**case class emailClass(mail:String, messageId:String, date:String, from:String, to:String, subject:String, message:String)**

**val emailData = email.map(x=>x.split("\t")).map(x=>emailClass(x(0).toString,x(1).toString,x(2).toString,x(3).toString,x(4).toString,x(5).toString,x(6).toString))**

**emailData.map(x=>(x.to,1)).reduceByKey((x,y)=>x+y).map(item=>item.swap).sortByKey(false,1).map(item=>item.swap).first**

#### Question 2:

We know that Mr. Allen likes to write highly detailed emails to explain technicalities of the project. Who has received the longest message from Mr. Allen?

#### Answer

**emailData.map(x=>(x.message.length,x.to)).sortByKey(false,1).first**

#### Question 3:

Continuing on from question 2, which message was this and when did it occur?

#### Answer

**val most = emailData.map(x=>(x.message.length, (x.to, x.messageId))).sortByKey(false,1).first**

**val most = emailData.map(x=>(x.message.length, (x.to, x.date))).sortByKey(false,1).first**

#### Question 4:

Now that we know about the longest message Phillip has sent, what is the average length of his messages?

#### Answer

**val emailAvg = emailData.map((x=>(x.message.length))).mean**

OR

**emailData.map(map(x=>(1,(x.message.length,1))).reduceByKey((x,y)=>(x.\_1 + y.\_1, x.\_2 + y.\_2)).map(x=>(x.\_2)).map(x=>(x.\_1/x.\_2)).collect**

#### Question 5:

Gathering that Phillip likes to send long emails to his colleagues, what are some of the key-points mentioned in these emails? You should also filter off words like “the”, “to” and “a” etc.

#### Answer

**val words = emailData.flatMap(x=>(x.message.split(" "))).map(word=>(word,1)).reduceByKey{case(x,y)=> x + y}.map(item=>item.swap).sortByKey(true,1).collect**

**val filteredWords = words.filter(\_.\_2 != "the").filter(\_.\_2 != "to").filter(\_.\_2 != "and").filter(\_.\_2 != "of").filter(\_.\_2 != "").filter(\_.\_2 != "a").filter(\_.\_2 != "for").filter(\_.\_2 != "on").filter(\_.\_2 != "is").filter(\_.\_2 != "in").filter(\_.\_2 != "you").filter(\_.\_2 != "I").filter(\_.\_2 != "have").filter(\_.\_2 != "be").filter(\_.\_2 != "that")**

**filteredWords.foreach(println)**

OR

**emailData.map(x=>x.message).flatMap(x=>x.split(“ ”)).map(x=>(x,1)).reduceByKey((x,y)=>(x+y)).map(x=>x.swap).sortByKey(false,1).filter(x => x.\_2 != “ ”.take(15)**

#### Question 6:

Time is valuable; we like to make sure we get as much value out of Phillip as possible. What time do emails frequently take place by Date Time, Date, Weekday and by hour?

#### Answer

By Date Time:

**emailData.map(x=>(x.date,1)).reduceByKey((x,y)=>x+y).map(item=>item.swap).sortByKey(true,1).map(item=>item.swap).foreach(println)**

By Date:

**emailData.map(x=>x.date).map(x=>x.split(“,”)).filter(x=>x(0) != “Date”).map(x=>(x(1),1)).reduceByKey((x,y)=>x+y).map(x=>x.swap).sortByKey(false,1).take(5)**

By Weekday:

**emailData.map(x=>x.date).map(x=>x.split(“,”)).filter(x=>x(0) != “Date”).map(x=>(x(0),1)).reduceByKey((x,y)=>x+y).map(x=>x.swap).sortByKey(false,1).take(5)**

By Hour:

**emailData.map(x=>x.date).map(x=>x.split(“,”)).filter(x=>x(0) != “Date”).map(x=>(x(2))).map(x=>x.split(“:”)).map(x=>(x(0),1)).reduceByKey((x,y)=>x+y).map(x=>x.swap).sortByKey(false,1).take(5)**

# Summary

You should now have an awareness and appreciation for using SPARK from the command line and writing SPARK-Scala to query through datasets. As you will now understand; map, filter, reduceByKey and sortByKey are some of the most commonly used transformations in SPARK.