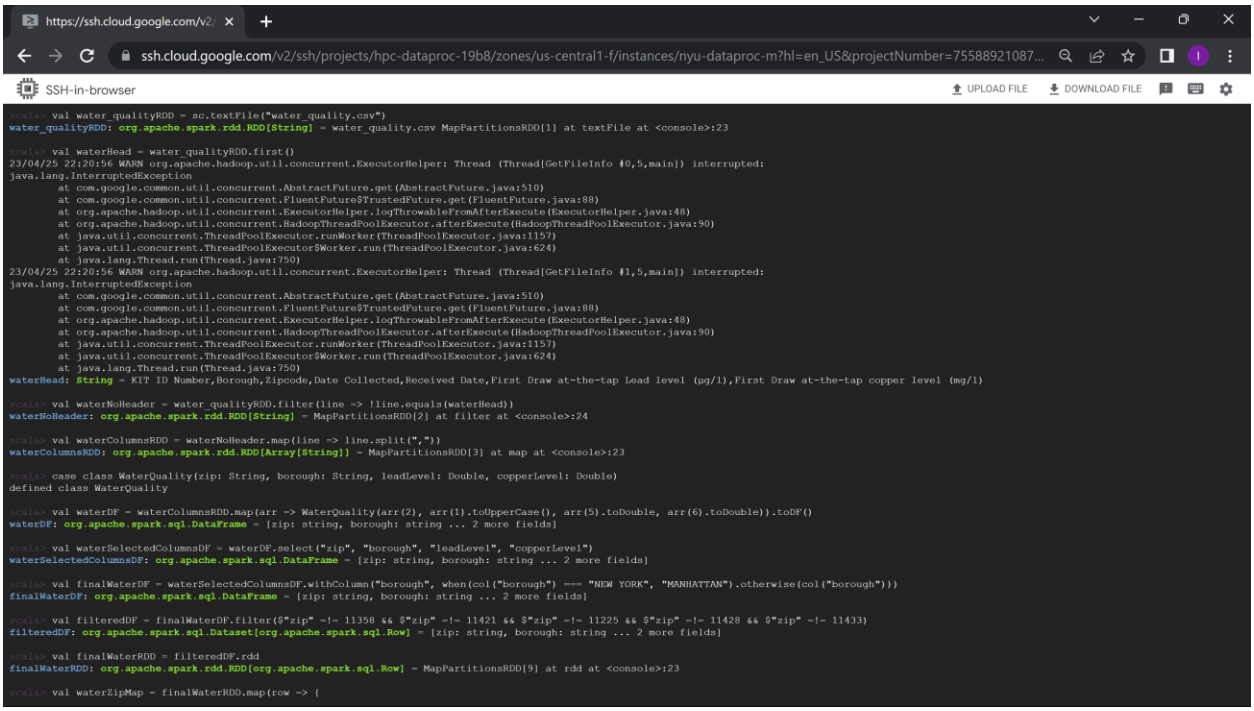
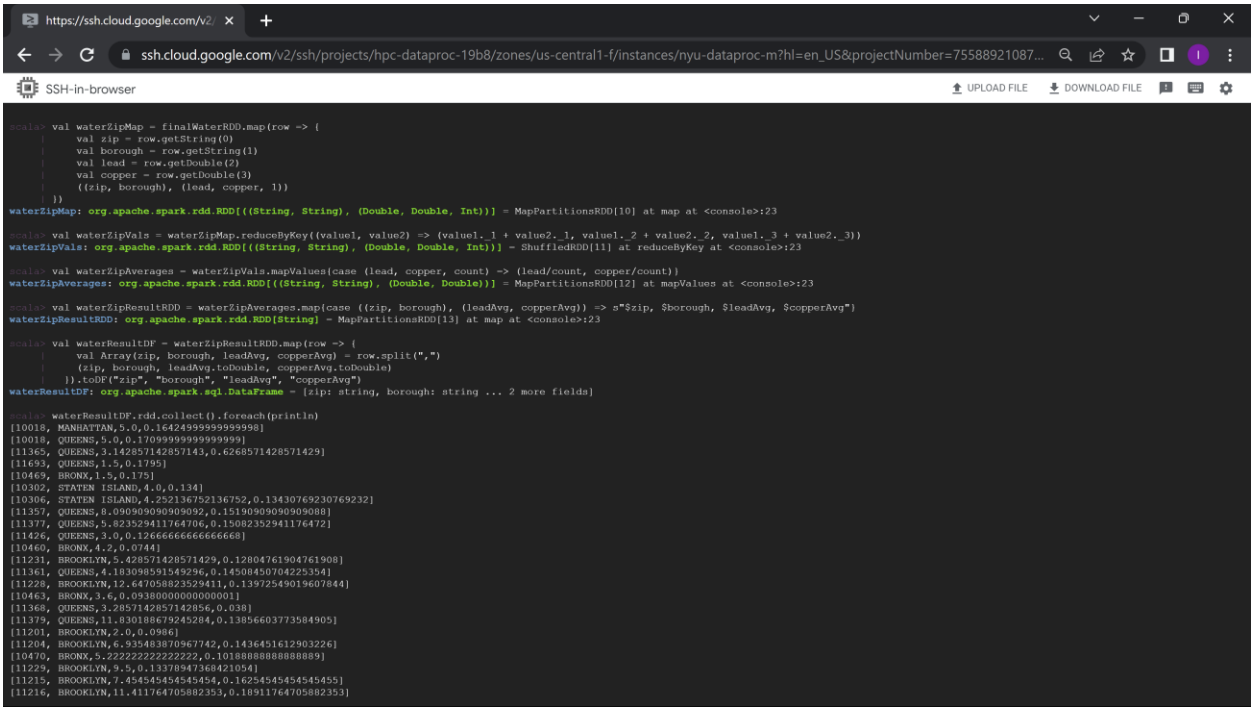
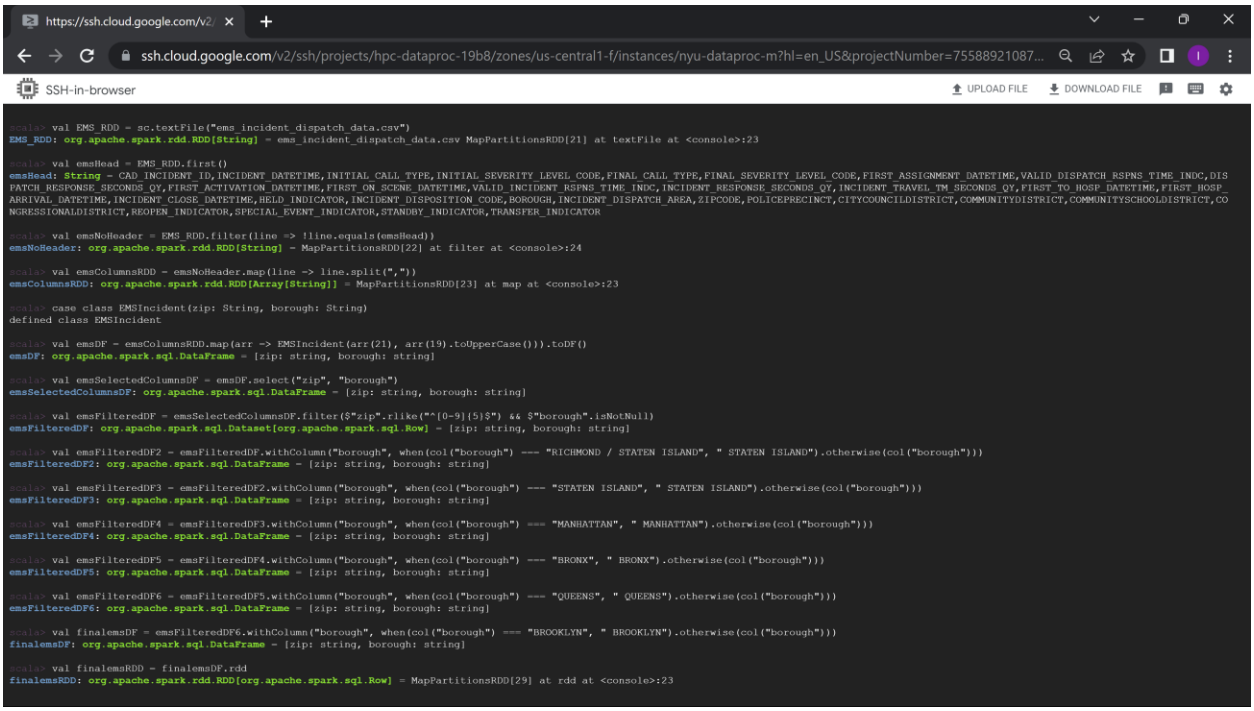


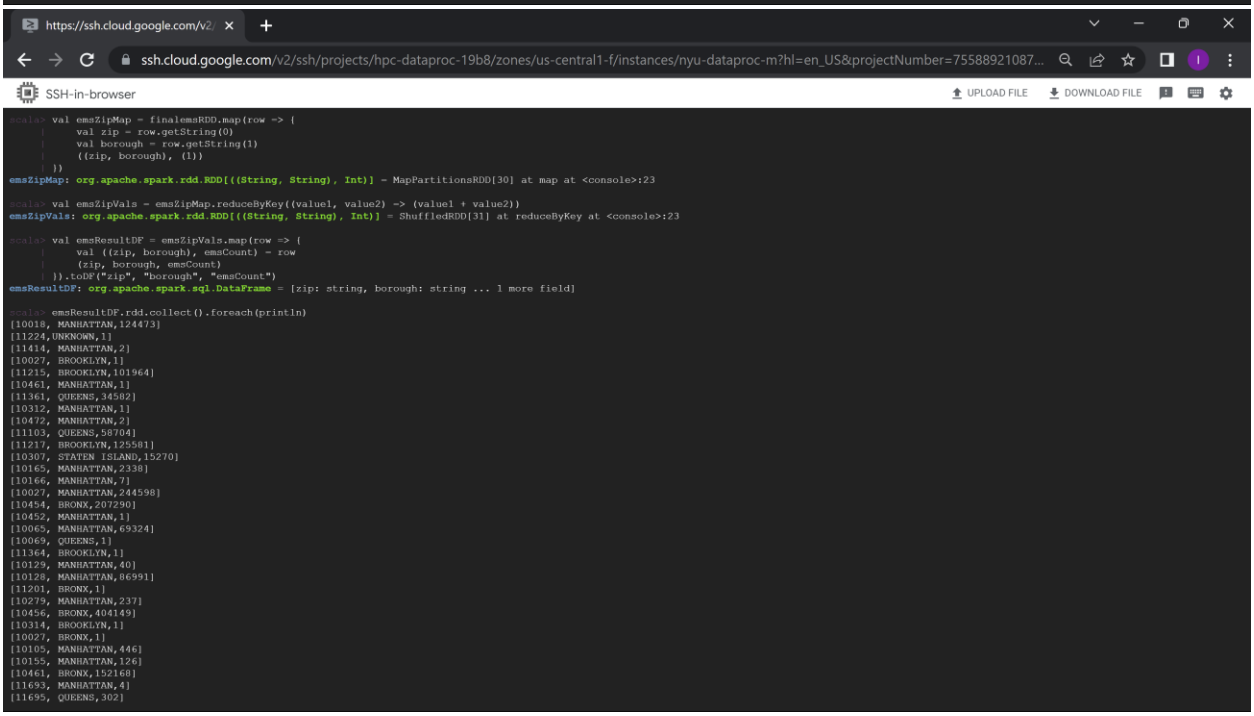

```
scala> val water_qualityRDD = sc.textFile("water_quality.csv")
water_qualityRDD: org.apache.spark.rdd.RDD[String] = water_quality.csv MapPartitionsRDD[1] at textFile at <console>:23

scala> val waterHead = water_qualityRDD.first()
23/04/25 22:20:56 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread (Thread[GetFileInfo #0,5,main]) interrupted:
java.lang.InterruptedException
        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:510)
        at com.google.common.util.concurrent.FluentFuture$TrustedFuture.get(FluentFuture.java:88)
        at org.apache.hadoop.util.concurrent.ExecutorHelper.logThrowableFromAfterExecute(ExecutorHelper.java:48)
        at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor.afterExecute(HadoopThreadPoolExecutor.java:90)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1157)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:750)
23/04/25 22:20:56 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread (Thread[GetFileInfo #1,5,main]) interrupted:
java.lang.InterruptedException
        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:510)
        at com.google.common.util.concurrent.FluentFuture$TrustedFuture.get(FluentFuture.java:88)
        at org.apache.hadoop.util.concurrent.ExecutorHelper.logThrowableFromAfterExecute(ExecutorHelper.java:48)
        at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor.afterExecute(HadoopThreadPoolExecutor.java:90)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1157)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:750)
waterHead: String = KIT ID Number,Borough,Zipcode,Date Collected,Received Date,First Draw at-the-tap Lead level (pg/l),First Draw at-the-tap copper level (mg/l)

scala> val waterNoHeader = water_qualityRDD.filter(line => !line.equals(waterHead))
waterNoHeader: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:24

scala> val waterColumnsRDD = waterNoHeader.map(line => line.split(","))
waterColumnsRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at map at <console>:23

scala> case class WaterQuality(zip: String, borough: String, leadLevel: Double, copperLevel: Double)
defined class WaterQuality

scala> val waterDF = waterColumnsRDD.map(arr => WaterQuality(arr(2), arr(1).toUpperCase(), arr(5).toDouble, arr(6).toDouble)).toDF()
waterDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 2 more fields]

scala> val waterSelectedColumnsDF = waterDF.select("zip", "borough", "leadLevel", "copperLevel")
waterSelectedColumnsDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 2 more fields]

scala> val finalWaterDF = waterSelectedColumnsDF.withColumn("borough", when(col("borough") === "NEW YORK", "MANHATTAN").otherwise(col("borough")))
finalWaterDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 2 more fields]

scala> val filteredDF = finalWaterDF.filter($"zip" =!= 11358 && $"zip" =!= 11421 && $"zip" =!= 11225 && $"zip" =!= 11428 && $"zip" =!= 11433)
filteredDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [zip: string, borough: string ... 2 more fields]

scala> val finalWaterRDD = filteredDF.rdd
finalWaterRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[9] at rdd at <console>:23

scala> val waterZipMap = finalWaterRDD.map(row => {
```



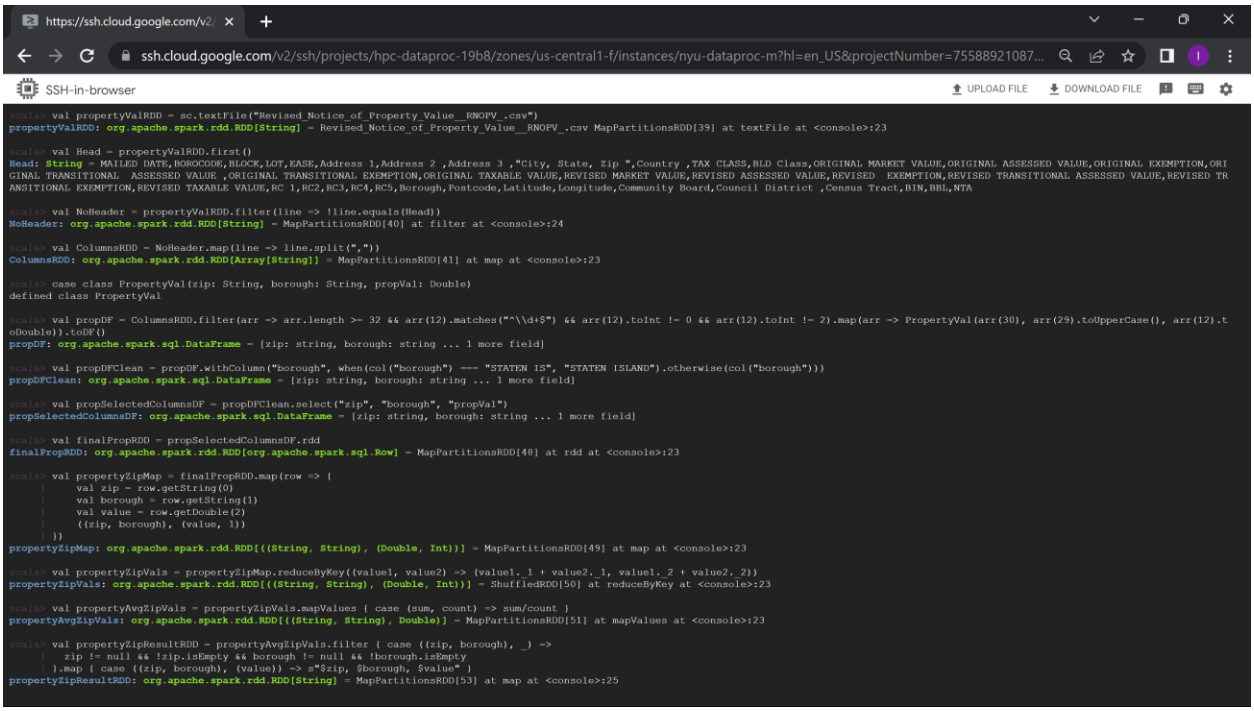

```
scala> val waterZipMap = finalWaterRDD.map(row => {
     |     val zip = row.getString(0)
     |     val borough = row.getString(1)
     |     val lead = row.getDouble(2)
     |     val copper = row.getDouble(3)
     |     ((zip, borough), (lead, copper, 1))
     | })
waterZipMap: org.apache.spark.rdd.RDD[((String, String), (Double, Double, Int))] = MapPartitionsRDD[10] at map at <console>:23

scala> val waterZipVals = waterZipMap.reduceByKey((value1, value2) => (value1._1 + value2._1, value1._2 + value2._2, value1._3 + value2._3))
waterZipVals: org.apache.spark.rdd.RDD[((String, String), (Double, Double, Int))] = ShuffledRDD[11] at reduceByKey at <console>:23

scala> val waterZipAverages = waterZipVals.mapValues{case (lead, copper, count) => (lead/count, copper/count)}
waterZipAverages: org.apache.spark.rdd.RDD[((String, String), (Double, Double))] = MapPartitionsRDD[12] at mapValues at <console>:23

scala> val waterZipResultRDD = waterZipAverages.map{case ((zip, borough), (leadAvg, copperAvg)) => s"$zip, $borough, $leadAvg, $copperAvg"}
waterZipResultRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[13] at map at <console>:23

scala> val waterResultDF = waterZipResultRDD.map(row => {
     |     val Array(zip, borough, leadAvg, copperAvg) = row.split(",")
     |     (zip, borough, leadAvg.toDouble, copperAvg.toDouble)
     |   }).toDF("zip", "borough", "leadAvg", "copperAvg")
waterResultDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 2 more fields]

scala> waterResultDF.rdd.collect().foreach(println)
[10018, MANHATTAN,5.0,0.16428999999999998]
[10018, QUEENS,5.0,0.17099999999999999]
[11365, QUEENS,3.1428571428571143,0.6268571428571429]
[11693, QUEENS,1.5,0.1795]
[10469, BRONX,1.5,0.175]
[10302, STATEN ISLAND,4.0,0.134]
[10306, STATEN ISLAND,4.252136752136752,0.1343076923076923]
[11357, QUEENS,8.090909090909092,0.15190909090909088]
[11377, QUEENS,5.823529411764706,0.15082352941176472]
[11426, QUEENS,3.0,0.12666666666666668]
[10460, BRONX,4.2,0.0744]
[11231, BROOKLYN,5.428571428571429,0.12804761904761908]
[11361, QUEENS,4.183098591549296,0.14508450704225354]
[11228, BROOKLYN,12.647058823529411,0.13972549019607844]
[10463, BRONX,3.6,0.09380000000000001]
[11368, QUEENS,3.2857142857142856,0.038]
[11379, QUEENS,11.830188679245284,0.13856603773584905]
[11201, BROOKLYN,2.0,0.0986]
[11204, BROOKLYN,6.935483870967742,0.14364516129032256]
[10470, BRONX,5.222222222222222,0.10188888888888889]
[11229, BROOKLYN,9.5,0.13378947368421054]
[11215, BROOKLYN,7.454545454545454,0.16254545454545455]
[11216, BROOKLYN,11.411764705882353,0.18911764705882353]
```

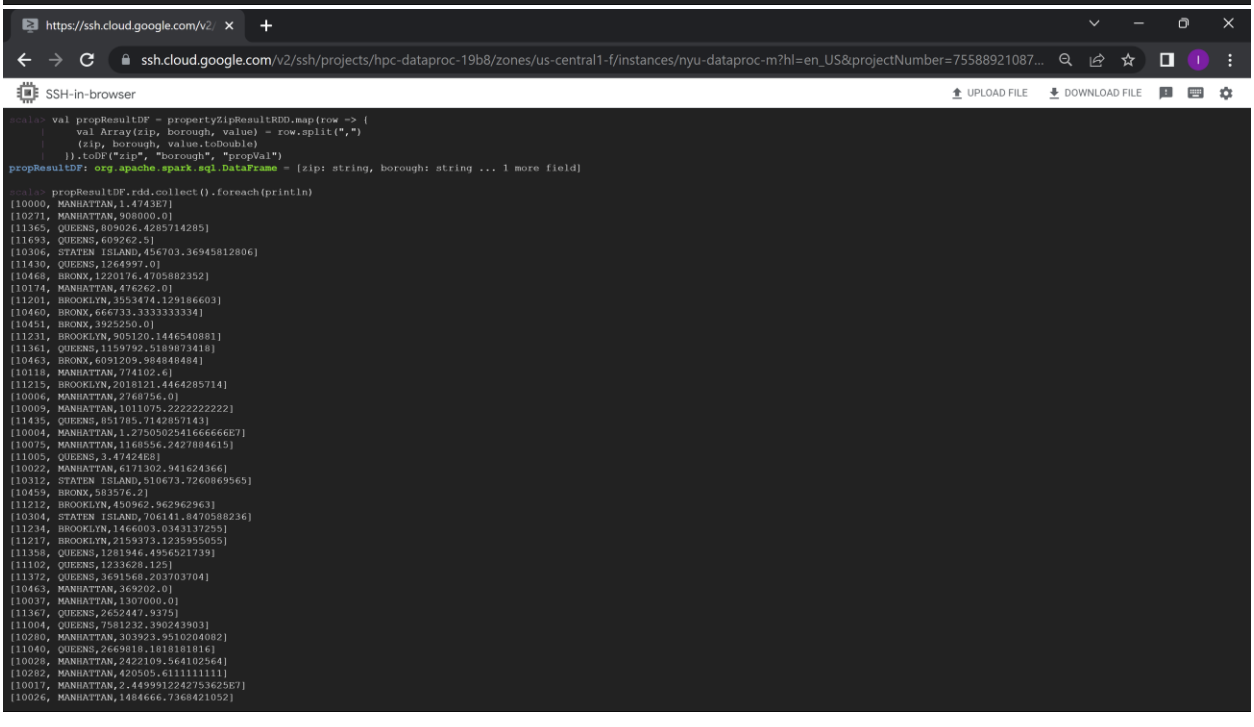


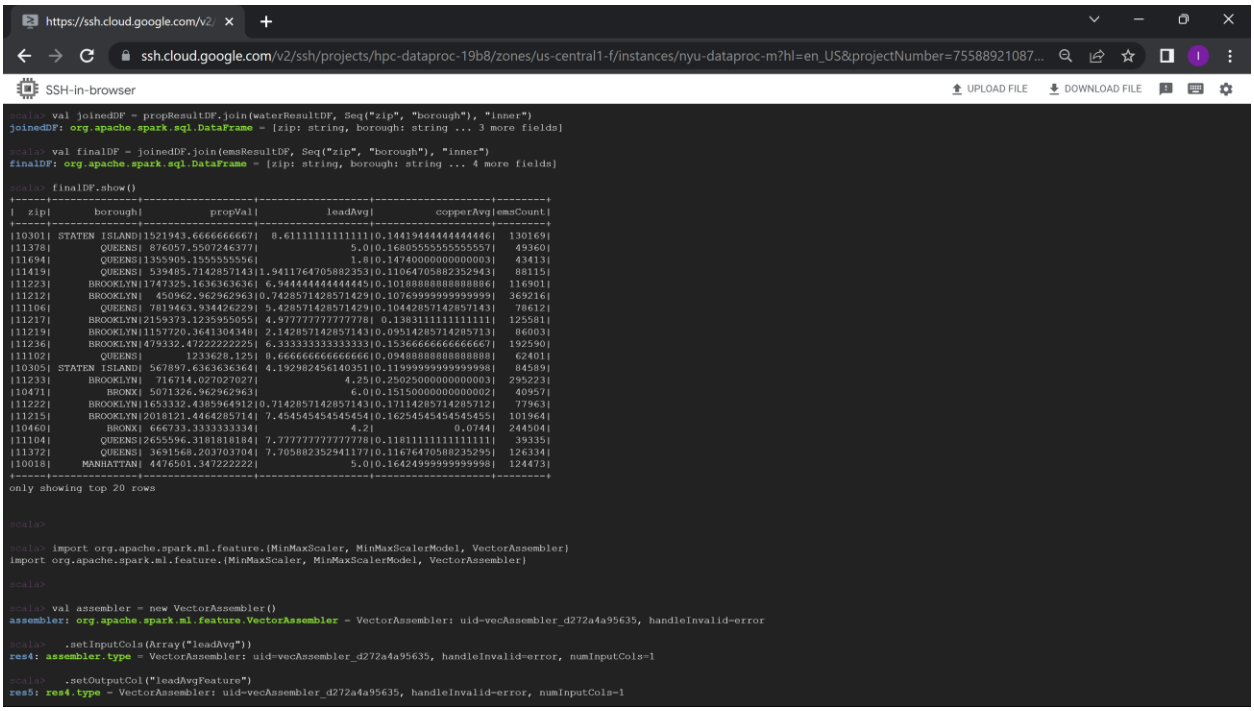
```
scala> val EMS_RDD = sc.textFile("ems_incident_dispatch_data.csv")
EMS_RDD: org.apache.spark.rdd.RDD[String] = ems_incident_dispatch_data.csv MapPartitionsRDD[21] at textFile at <console>:23

scala> val emsHead = EMS_RDD.first()
emsHead: String = CAD_INCIDENT_ID,INCIDENT_DATETIME,INITIAL_CALL_TYPE,INITIAL_SEVERITY_LEVEL_CODE,FINAL_CALL_TYPE,FINAL_SEVERITY_LEVEL_CODE,FIRST_ASSIGNMENT_DATETIME,VALID_DISPATCH_RSPNS_TIME_INDC,DISPATCH_RESPONSE_SECONDS_QY,FIRST_ACTIVATION_DATETIME,FIRST_ON_SCENE_DATETIME,VALID_INCIDENT_RSPNS_TIME_INDC,INCIDENT_RESPONSE_SECONDS_QY,INCIDENT_TRAVEL_TM_SECONDS_QY,FIRST_TO_HOSP_DATETIME,FIRST_HOSP_ARRIVAL_DATETIME,INCIDENT_CLOSE_DATETIME,HELD_INDICATOR,INCIDENT_DISPOSITION_CODE,BOROUGH,INCIDENT_DISPATCH_AREA,ZIPCODE,POLICEPRECINCT,CITYCOUNCILDISTRICT,COMMUNITYDISTRICT,COMMUNITYSCHOOLDISTRICT,CONGRESSIONALDISTRICT,REOPEN_INDICATOR,SPECIAL_EVENT_INDICATOR,STANDBY_INDICATOR,TRANSFER_INDICATOR

scala> val emsNoHeader = EMS_RDD.filter(line => !line.equals(emsHead))
emsNoHeader: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[22] at filter at <console>:24

scala> val emsColumnsRDD = emsNoHeader.map(line => line.split(","))
emsColumnsRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[23] at map at <console>:23

scala> case class EMSIncident(zip: String, borough: String)
defined class EMSIncident

scala> val emsDF = emsColumnsRDD.map(arr => EMSIncident(arr(21), arr(19).toUpperCase())).toDF()
emsDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val emsSelectedColumnsDF = emsDF.select("zip", "borough")
emsSelectedColumnsDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val emsFilteredDF = emsSelectedColumnsDF.filter($"zip".rlike("^[0-9]{5}$") && $"borough".isNotNull)
emsFilteredDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [zip: string, borough: string]

scala> val emsFilteredDF2 = emsFilteredDF.withColumn("borough", when(col("borough") === "RICHMOND / STATEN ISLAND", " STATEN ISLAND").otherwise(col("borough")))
emsFilteredDF2: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val emsFilteredDF3 = emsFilteredDF2.withColumn("borough", when(col("borough") === "STATEN ISLAND", " STATEN ISLAND").otherwise(col("borough")))
emsFilteredDF3: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val emsFilteredDF4 = emsFilteredDF3.withColumn("borough", when(col("borough") === "MANHATTAN", " MANHATTAN").otherwise(col("borough")))
emsFilteredDF4: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val emsFilteredDF5 = emsFilteredDF4.withColumn("borough", when(col("borough") === "BRONX", " BRONX").otherwise(col("borough")))
emsFilteredDF5: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val emsFilteredDF6 = emsFilteredDF5.withColumn("borough", when(col("borough") === "QUEENS", " QUEENS").otherwise(col("borough")))
emsFilteredDF6: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val finalemsDF = emsFilteredDF6.withColumn("borough", when(col("borough") === "BROOKLYN", " BROOKLYN").otherwise(col("borough")))
finalemsDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string]

scala> val finalemsRDD = finalemsDF.rdd
finalemsRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[29] at rdd at <console>:23
```

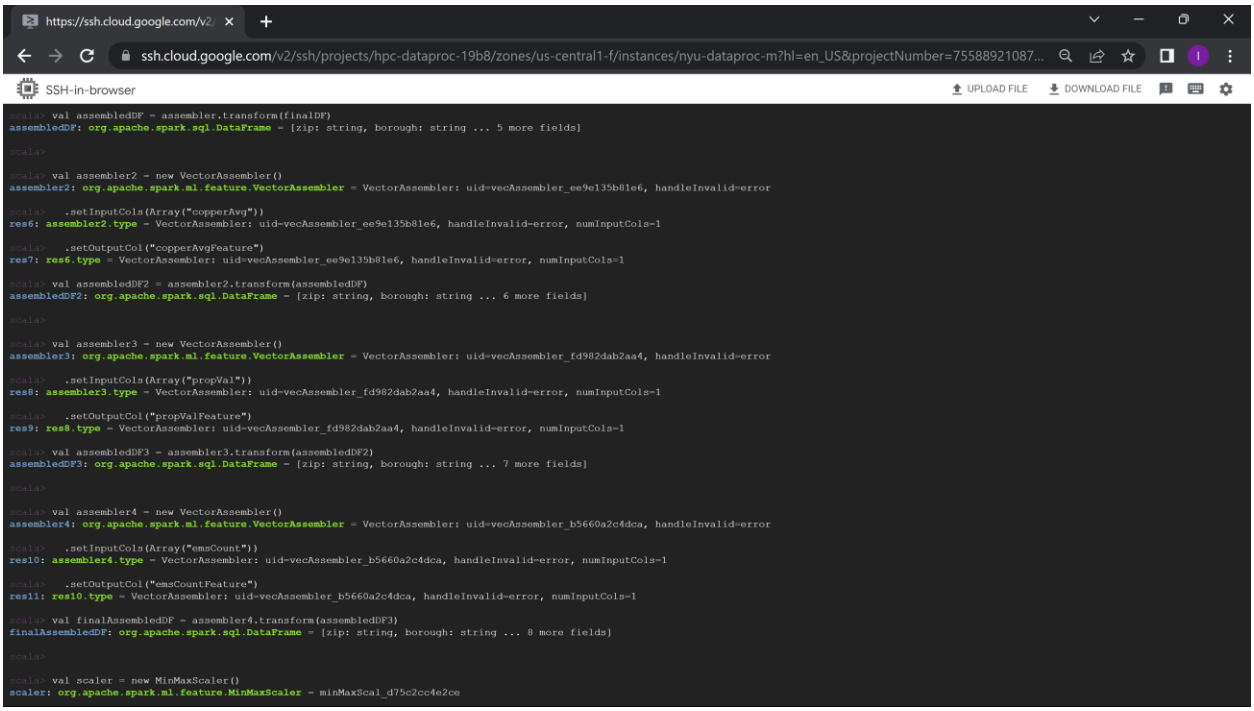


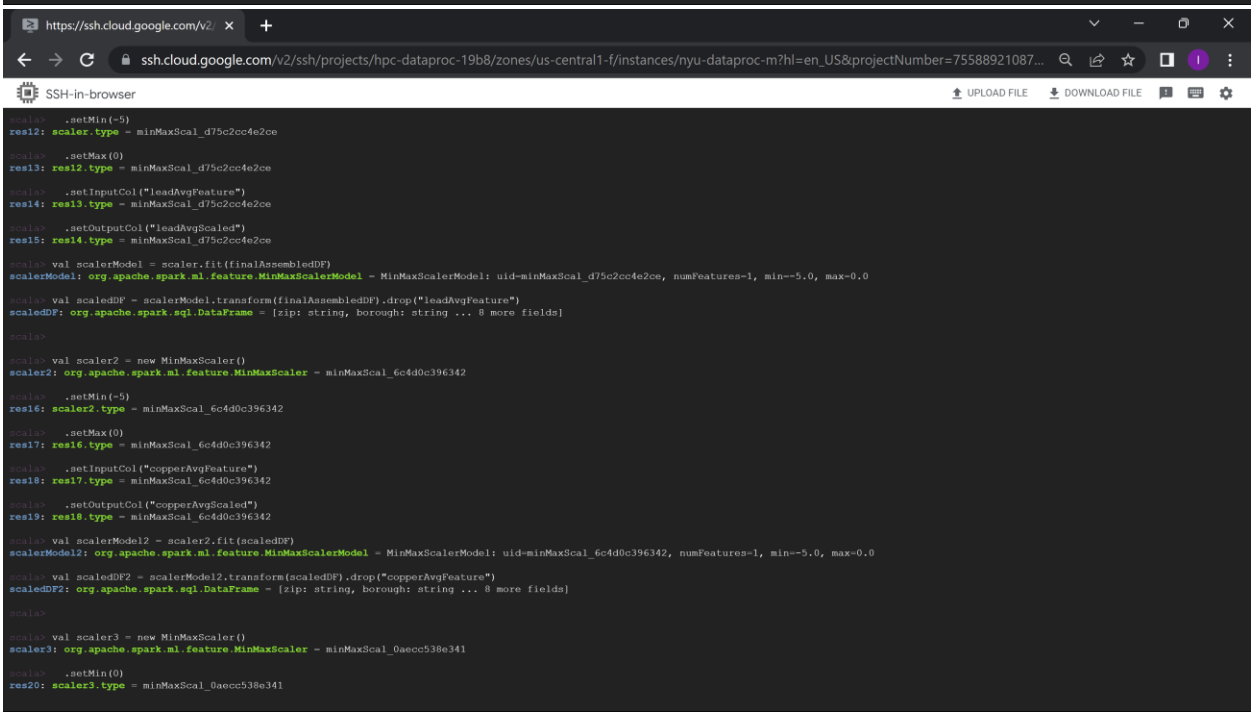
```
scala> val emsZipMap = finalemsRDD.map(row => {
     |     val zip = row.getString(0)
     |     val borough = row.getString(1)
     |     ((zip, borough), (1))
     | })
emsZipMap: org.apache.spark.rdd.RDD[((String, String), Int)] = MapPartitionsRDD[30] at map at <console>:23

scala> val emsZipVals = emsZipMap.reduceByKey((value1, value2) => (value1 + value2))
emsZipVals: org.apache.spark.rdd.RDD[((String, String), Int)] = ShuffledRDD[31] at reduceByKey at <console>:23

scala> val emsResultDF = emsZipVals.map(row => {
     |     val ((zip, borough), emsCount) = row
     |     (zip, borough, emsCount)
     | }).toDF("zip", "borough", "emsCount")
emsResultDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 1 more field]

scala> emsResultDF.rdd.collect().foreach(println)
[10018, MANHATTAN,124473]
[11224,UNKNOWN,1]
[11414, MANHATTAN,2]
[10027, BROOKLYN,1]
[11215, BROOKLYN,101964]
[10461, MANHATTAN,1]
[11361, QUEENS,34582]
[10312, MANHATTAN,1]
[10472, MANHATTAN,2]
[11103, QUEENS,58704]
[11217, BROOKLYN,125581]
[10307, STATEN ISLAND,15270]
[10165, MANHATTAN,2338]
[10166, MANHATTAN,7]
[10027, MANHATTAN,244598]
[10454, BRONX,207290]
[10452, MANHATTAN,1]
[10065, MANHATTAN,69324]
[10069, QUEENS,1]
[11364, BROOKLYN,1]
[10129, MANHATTAN,40]
[10128, MANHATTAN,86991]
[11201, BRONX,1]
[10279, MANHATTAN,237]
[10456, BRONX,404149]
[10314, BROOKLYN,1]
[10027, BRONX,1]
[10105, MANHATTAN,446]
[10155, MANHATTAN,126]
[10461, BRONX,152168]
[11693, MANHATTAN,4]
[11695, QUEENS,302]
```

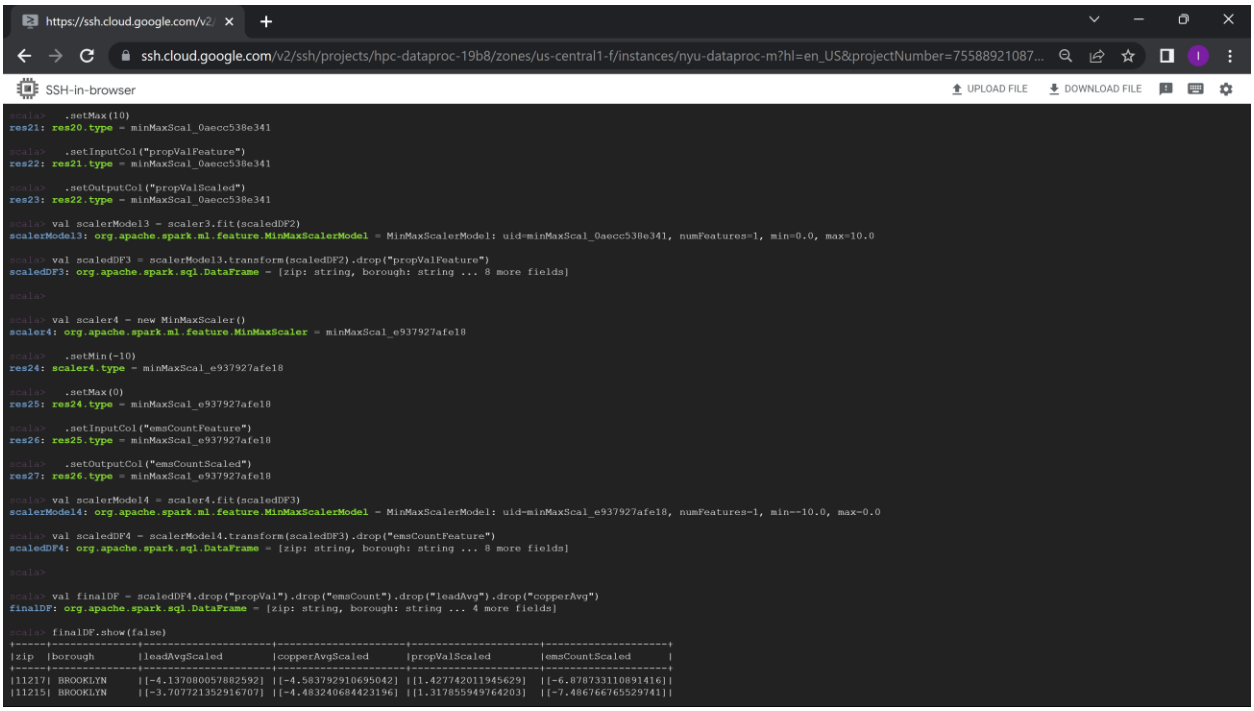

https://ssh.cloud.google.com/v2/
ssh.cloud.google.com/v2/ssh/projects/hpc-dataproc-19b8/zones/us-central1-f/instances/nyu-dataproc-m?hl=en_US&projectNumber=75588921087...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
scala> val propertyValRDD = sc.textFile("Revised_Notice_of_Property_Value__RNOPV_.csv")
propertyValRDD: org.apache.spark.rdd.RDD[String] = Revised_Notice_of_Property_Value__RNOPV_.csv MapPartitionsRDD[39] at textFile at <console>:23
scala> val Head = propertyValRDD.first()
Head: String = MAILED DATE,BOROCODE,BLOCK,LOT,EASE,Address 1,Address 2 ,Address 3 ,"City, State, Zip ",Country ,TAX CLASS,BLD Class,ORIGINAL MARKET VALUE,ORIGINAL ASSESSED VALUE,ORIGINAL EXEMPTION,ORIGINAL TRANSITIONAL ASSESSED VALUE ,ORIGINAL TRANSITIONAL EXEMPTION,ORIGINAL TAXABLE VALUE,REVISED MARKET VALUE,REVISED ASSESSED VALUE,REVISED EXEMPTION,REVISED TRANSITIONAL ASSESSED VALUE,REVISED TRANSITIONAL EXEMPTION,REVISED TAXABLE VALUE,RC 1,RC2,RC3,RC4,RC5,Borough,Postcode,Latitude,Longitude,Community Board,Council District ,Census Tract,BIN,BBL,NTA
scala> val NoHeader = propertyValRDD.filter(line => !line.equals(Head))
NoHeader: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[40] at filter at <console>:24
scala> val ColumnsRDD = NoHeader.map(line => line.split(","))
ColumnsRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[41] at map at <console>:23
scala> case class PropertyVal(zip: String, borough: String, propVal: Double)
defined class PropertyVal
scala> val propDF = ColumnsRDD.filter(arr => arr.length >= 32 && arr(12).matches("^\\d+$") && arr(12).toInt != 0 && arr(12).toInt != 2).map(arr => PropertyVal(arr(30), arr(29).toUpperCase(), arr(12).toDouble)).toDF()
propDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 1 more field]
scala> val propDFClean = propDF.withColumn("borough", when(col("borough") === "STATEN IS", "STATEN ISLAND").otherwise(col("borough")))
propDFClean: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 1 more field]
scala> val propSelectedColumnsDF = propDFClean.select("zip", "borough", "propVal")
propSelectedColumnsDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 1 more field]
scala> val finalPropRDD = propSelectedColumnsDF.rdd
finalPropRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[48] at rdd at <console>:23
scala> val propertyZipMap = finalPropRDD.map(row => {
val zip = row.getString(0)
val borough = row.getString(1)
val value = row.getDouble(2)
((zip, borough), (value, 1))
})
propertyZipMap: org.apache.spark.rdd.RDD[((String, String), (Double, Int))] = MapPartitionsRDD[49] at map at <console>:23
scala> val propertyZipVals = propertyZipMap.reduceByKey((value1, value2) => (value1._1 + value2._1, value1._2 + value2._2))
propertyZipVals: org.apache.spark.rdd.RDD[((String, String), (Double, Int))] = ShuffledRDD[50] at reduceByKey at <console>:23
scala> val propertyAvgZipVals = propertyZipVals.mapValues { case (sum, count) => sum/count }
propertyAvgZipVals: org.apache.spark.rdd.RDD[((String, String), Double)] = MapPartitionsRDD[51] at mapValues at <console>:23
scala> val propertyZipResultRDD = propertyAvgZipVals.filter { case ((zip, borough), _) =>
zip != null && !zip.isEmpty && borough != null && !borough.isEmpty
}.map { case ((zip, borough), (value)) => s"$zip, $borough, $value" }
propertyZipResultRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[53] at map at <console>:25

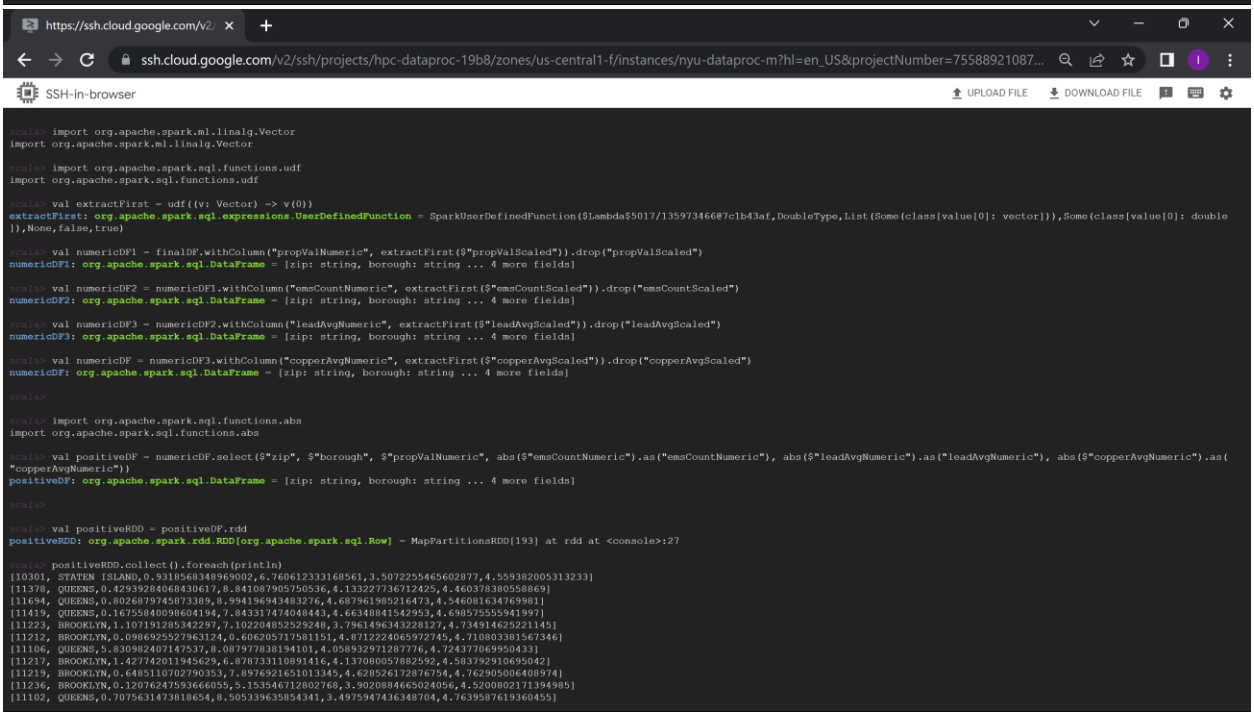

https://ssh.cloud.google.com/v2/
ssh.cloud.google.com/v2/ssh/projects/hpc-dataproc-19b8/zones/us-central1-f/instances/nyu-dataproc-m?hl=en_US&projectNumber=75588921087...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
scala> val propResultDF = propertyZipResultRDD.map(row => {
val Array(zip, borough, value) = row.split(",")
(zip, borough, value.toDouble)
}).toDF("zip", "borough", "propVal")
propResultDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 1 more field]
scala> propResultDF.rdd.collect().foreach(println)
[10000, MANHATTAN,1.4743E7]
[10271, MANHATTAN,908000.0]
[11365, QUEENS,809026.4285714285]
[11693, QUEENS,609262.5]
[10306, STATEN ISLAND,456703.36945812806]
[11430, QUEENS,1264997.0]
[10468, BRONX,1220176.4705882352]
[10174, MANHATTAN,476262.0]
[11201, BROOKLYN,3553474.129186603]
[10460, BRONX,666733.3333333334]
[10451, BRONX,3925250.0]
[11231, BROOKLYN,905120.1446540881]
[11361, QUEENS,1159792.5189873418]
[10463, BRONX,6091209.984848484]
[10118, MANHATTAN,774102.6]
[11215, BROOKLYN,2018121.4464285714]
[10006, MANHATTAN,2768756.0]
[10009, MANHATTAN,1011075.2222222222]
[11435, QUEENS,851785.7142857143]
[10004, MANHATTAN,1.2750502541666666E7]
[10075, MANHATTAN,1168556.2427884615]
[11005, QUEENS,3.47424E8]
[10022, MANHATTAN,6171302.941624366]
[10312, STATEN ISLAND,510673.7260869565]
[10459, BRONX,583576.2]
[11212, BROOKLYN,450962.962962963]
[10304, STATEN ISLAND,706141.8470588236]
[11234, BROOKLYN,1466003.0343137255]
[11217, BROOKLYN,2159373.1235955055]
[11358, QUEENS,1281946.4956521739]
[11102, QUEENS,1233628.125]
[11372, QUEENS,3691568.203703704]
[10463, MANHATTAN,369202.0]
[10037, MANHATTAN,1307000.0]
[11367, QUEENS,2652447.9375]
[11004, QUEENS,7581232.390243903]
[10280, MANHATTAN,303923.9510204082]
[11040, QUEENS,2669818.1818181816]
[10028, MANHATTAN,2422109.564102564]
[10282, MANHATTAN,420505.6111111111]
[10017, MANHATTAN,2.4499912242753625E7]
[10026, MANHATTAN,1484666.7368421052]

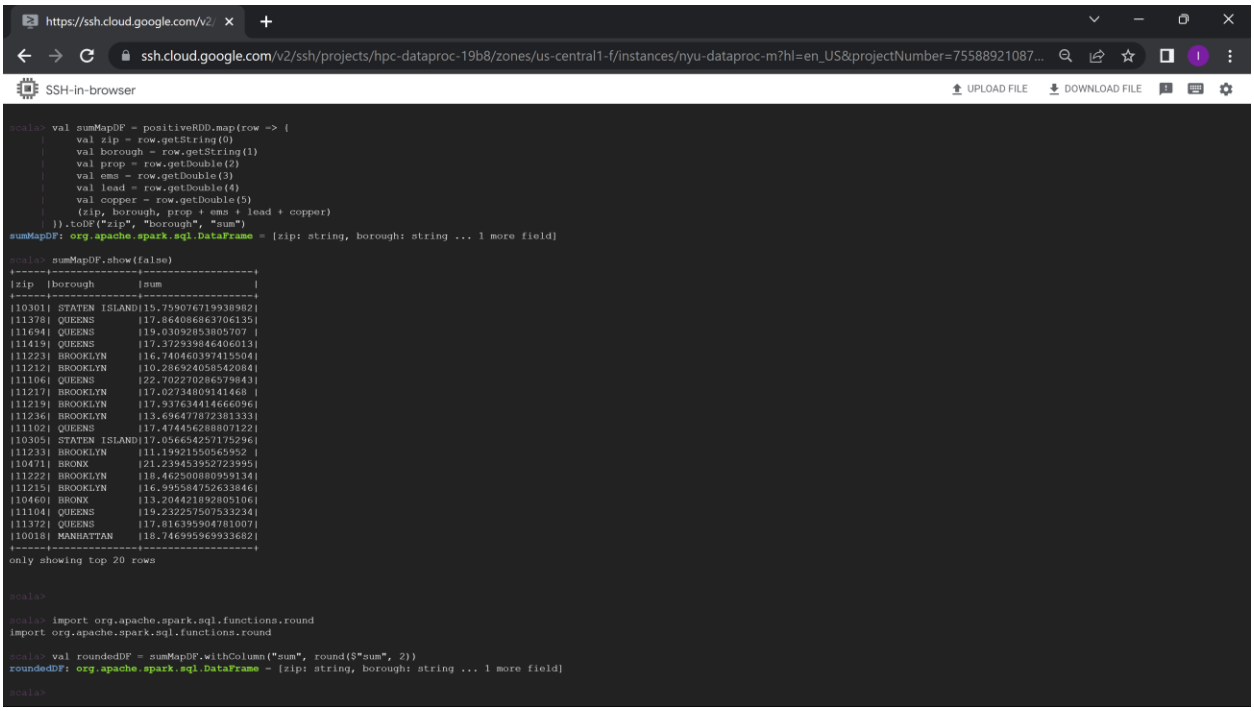

https://ssh.cloud.google.com/v2/
ssh.cloud.google.com/v2/ssh/projects/hpc-dataproc-19b8/zones/us-central1-f/instances/nyu-dataproc-m?hl=en_US&projectNumber=75588921087...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
scala> val joinedDF = propResultDF.join(waterResultDF, Seq("zip", "borough"), "inner")
joinedDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 3 more fields]
scala> val finalDF = joinedDF.join(emsResultDF, Seq("zip", "borough"), "inner")
finalDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 4 more fields]
scala> finalDF.show()
| zip| borough| propVal| leadAvg| copperAvg|emsCount|
|10301| STATEN ISLAND|1521943.6666666667| 8.611111111111111|0.14419444444444446| 130169|
|11378| QUEENS| 876057.5507246377| 5.0|0.16805555555555557| 49360|
|11694| QUEENS|1355905.1555555556| 1.8|0.14740000000000003| 43413|
|11419| QUEENS| 539485.7142857143|1.9411764705882353|0.11064705882352943| 88115|
|11223| BROOKLYN|1747325.1636363636| 6.944444444444445|0.10188888888888886| 116901|
|11212| BROOKLYN| 450962.962962963|0.7428571428571429|0.10769999999999999| 369216|
|11106| QUEENS| 7819463.934426229| 5.428571428571429|0.10442857142857143| 78612|
|11217| BROOKLYN|2159373.1235955055| 4.977777777777778| 0.1383111111111111| 125581|
|11219| BROOKLYN|1157720.3641304348| 2.142857142857143|0.09514285714285713| 86003|
|11236| BROOKLYN|479332.47222222225| 6.333333333333333|0.15366666666666667| 192590|
|11102| QUEENS| 1233628.125| 8.666666666666666|0.09488888888888888| 62401|
|10305| STATEN ISLAND| 567897.6363636364| 4.192982456140351|0.11999999999999998| 84589|
|11233| BROOKLYN| 716714.027027027| 4.25|0.25025000000000003| 295223|
|10471| BRONX| 5071326.962962963| 6.0|0.15150000000000002| 40957|
|11222| BROOKLYN|1653332.4385964912|0.7142857142857143|0.17114285714285712| 77963|
|11215| BROOKLYN|2018121.4464285714| 7.454545454545454|0.16254545454545455| 101964|
|10460| BRONX| 666733.3333333334| 4.2| 0.0744| 244504|
|11104| QUEENS|2655596.3181818184| 7.777777777777778|0.11811111111111111| 39335|
|11372| QUEENS| 3691568.203703704| 7.705882352941177|0.11676470588235295| 126334|
|10018| MANHATTAN| 4476501.347222222| 5.0|0.16424999999999998| 124473|
only showing top 20 rows
scala>
scala> import org.apache.spark.ml.feature.{MinMaxScaler, MinMaxScalerModel, VectorAssembler}
import org.apache.spark.ml.feature.{MinMaxScaler, MinMaxScalerModel, VectorAssembler}
scala>
scala> val assembler = new VectorAssembler()
assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_d272a4a95635, handleInvalid=error
scala> .setInputCols(Array("leadAvg"))
res4: assembler.type = VectorAssembler: uid=vecAssembler_d272a4a95635, handleInvalid=error, numInputCols=1
scala> .setOutputCol("leadAvgFeature")
res5: res4.type = VectorAssembler: uid=vecAssembler_d272a4a95635, handleInvalid=error, numInputCols=1


https://ssh.cloud.google.com/v2/
ssh.cloud.google.com/v2/ssh/projects/hpc-dataproc-19b8/zones/us-central1-f/instances/nyu-dataproc-m?hl=en_US&projectNumber=75588921087...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
scala> val assembledDF = assembler.transform(finalDF)
assembledDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 5 more fields]
scala>
scala> val assembler2 = new VectorAssembler()
assembler2: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_ee9e135b81e6, handleInvalid=error
scala> .setInputCols(Array("copperAvg"))
res6: assembler2.type = VectorAssembler: uid=vecAssembler_ee9e135b81e6, handleInvalid=error, numInputCols=1
scala> .setOutputCol("copperAvgFeature")
res7: res6.type = VectorAssembler: uid=vecAssembler_ee9e135b81e6, handleInvalid=error, numInputCols=1
scala> val assembledDF2 = assembler2.transform(assembledDF)
assembledDF2: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 6 more fields]
scala>
scala> val assembler3 = new VectorAssembler()
assembler3: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_fd982dab2aa4, handleInvalid=error
scala> .setInputCols(Array("propVal"))
res8: assembler3.type = VectorAssembler: uid=vecAssembler_fd982dab2aa4, handleInvalid=error, numInputCols=1
scala> .setOutputCol("propValFeature")
res9: res8.type = VectorAssembler: uid=vecAssembler_fd982dab2aa4, handleInvalid=error, numInputCols=1
scala> val assembledDF3 = assembler3.transform(assembledDF2)
assembledDF3: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 7 more fields]
scala>
scala> val assembler4 = new VectorAssembler()
assembler4: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_b5660a2c4dca, handleInvalid=error
scala> .setInputCols(Array("emsCount"))
res10: assembler4.type = VectorAssembler: uid=vecAssembler_b5660a2c4dca, handleInvalid=error, numInputCols=1
scala> .setOutputCol("emsCountFeature")
res11: res10.type = VectorAssembler: uid=vecAssembler_b5660a2c4dca, handleInvalid=error, numInputCols=1
scala> val finalAssembledDF = assembler4.transform(assembledDF3)
finalAssembledDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 8 more fields]
scala>
scala> val scaler = new MinMaxScaler()
scaler: org.apache.spark.ml.feature.MinMaxScaler = minMaxScal_d75c2cc4e2ce

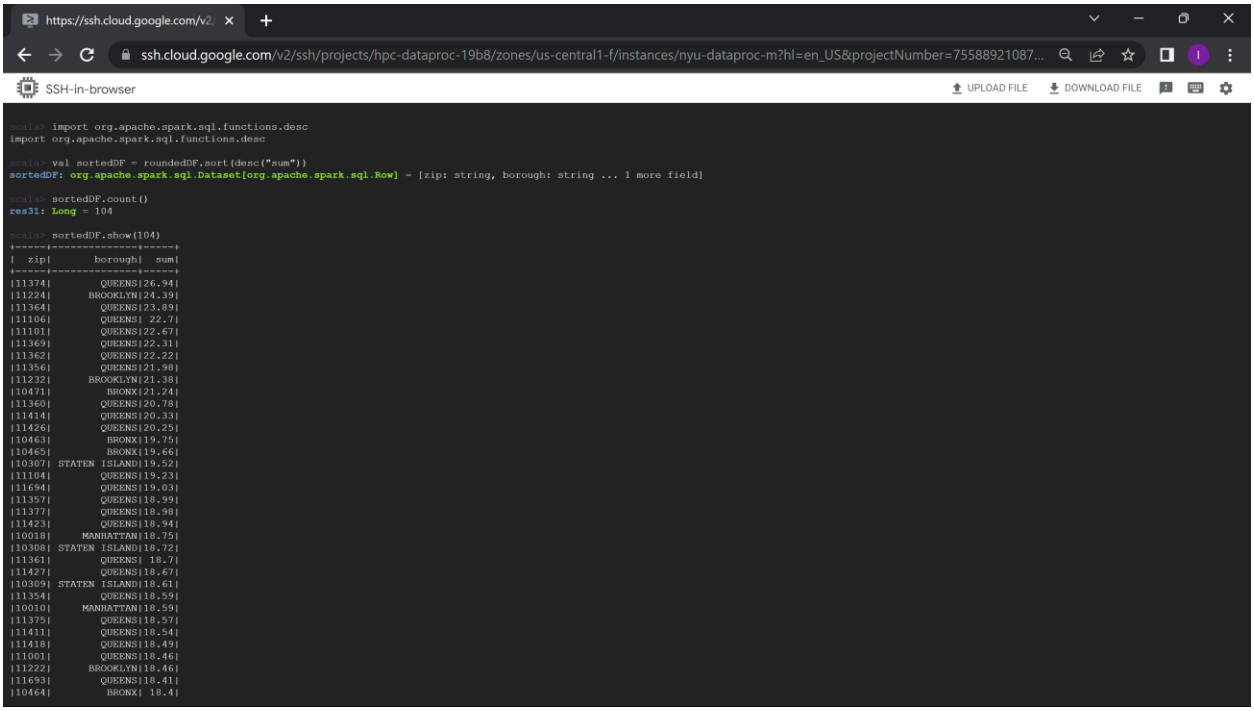

https://ssh.cloud.google.com/v2/
ssh.cloud.google.com/v2/ssh/projects/hpc-dataproc-19b8/zones/us-central1-f/instances/nyu-dataproc-m?hl=en_US&projectNumber=75588921087...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
scala> .setMin(-5)
res12: scaler.type = minMaxScal_d75c2cc4e2ce
scala> .setMax(0)
res13: res12.type = minMaxScal_d75c2cc4e2ce
scala> .setInputCol("leadAvgFeature")
res14: res13.type = minMaxScal_d75c2cc4e2ce
scala> .setOutputCol("leadAvgScaled")
res15: res14.type = minMaxScal_d75c2cc4e2ce
scala> val scalerModel = scaler.fit(finalAssembledDF)
scalerModel: org.apache.spark.ml.feature.MinMaxScalerModel = MinMaxScalerModel: uid=minMaxScal_d75c2cc4e2ce, numFeatures=1, min=-5.0, max=0.0
scala> val scaledDF = scalerModel.transform(finalAssembledDF).drop("leadAvgFeature")
scaledDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 8 more fields]
scala>
scala> val scaler2 = new MinMaxScaler()
scaler2: org.apache.spark.ml.feature.MinMaxScaler = minMaxScal_6c4d0c396342
scala> .setMin(-5)
res16: scaler2.type = minMaxScal_6c4d0c396342
scala> .setMax(0)
res17: res16.type = minMaxScal_6c4d0c396342
scala> .setInputCol("copperAvgFeature")
res18: res17.type = minMaxScal_6c4d0c396342
scala> .setOutputCol("copperAvgScaled")
res19: res18.type = minMaxScal_6c4d0c396342
scala> val scalerModel2 = scaler2.fit(scaledDF)
scalerModel2: org.apache.spark.ml.feature.MinMaxScalerModel = MinMaxScalerModel: uid=minMaxScal_6c4d0c396342, numFeatures=1, min=-5.0, max=0.0
scala> val scaledDF2 = scalerModel2.transform(scaledDF).drop("copperAvgFeature")
scaledDF2: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 8 more fields]
scala>
scala> val scaler3 = new MinMaxScaler()
scaler3: org.apache.spark.ml.feature.MinMaxScaler = minMaxScal_0aecc538e341
scala> .setMin(0)
res20: scaler3.type = minMaxScal_0aecc538e341


https://ssh.cloud.google.com/v2/
ssh.cloud.google.com/v2/ssh/projects/hpc-dataproc-19b8/zones/us-central1-f/instances/nyu-dataproc-m?hl=en_US&projectNumber=75588921087...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
scala> .setMax(10)
res21: res20.type = minMaxScal_0aecc538e341
scala> .setInputCol("propValFeature")
res22: res21.type = minMaxScal_0aecc538e341
scala> .setOutputCol("propValScaled")
res23: res22.type = minMaxScal_0aecc538e341
scala> val scalerModel3 = scaler3.fit(scaledDF2)
scalerModel3: org.apache.spark.ml.feature.MinMaxScalerModel = MinMaxScalerModel: uid=minMaxScal_0aecc538e341, numFeatures=1, min=0.0, max=10.0
scala> val scaledDF3 = scalerModel3.transform(scaledDF2).drop("propValFeature")
scaledDF3: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 8 more fields]
scala>
scala> val scaler4 = new MinMaxScaler()
scaler4: org.apache.spark.ml.feature.MinMaxScaler = minMaxScal_e937927afe18
scala> .setMin(-10)
res24: scaler4.type = minMaxScal_e937927afe18
scala> .setMax(0)
res25: res24.type = minMaxScal_e937927afe18
scala> .setInputCol("emsCountFeature")
res26: res25.type = minMaxScal_e937927afe18
scala> .setOutputCol("emsCountScaled")
res27: res26.type = minMaxScal_e937927afe18
scala> val scalerModel4 = scaler4.fit(scaledDF3)
scalerModel4: org.apache.spark.ml.feature.MinMaxScalerModel = MinMaxScalerModel: uid=minMaxScal_e937927afe18, numFeatures=1, min=-10.0, max=0.0
scala> val scaledDF4 = scalerModel4.transform(scaledDF3).drop("emsCountFeature")
scaledDF4: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 8 more fields]
scala>
scala> val finalDF = scaledDF4.drop("propVal").drop("emsCount").drop("leadAvg").drop("copperAvg")
finalDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 4 more fields]
scala> finalDF.show(false)
+-----+-------------+--------------------+--------------------+--------------------+--------------------+
|zip |borough |leadAvgScaled |copperAvgScaled |propValScaled |emsCountScaled |
+-----+-------------+--------------------+--------------------+--------------------+--------------------+
|11217|BROOKLYN |[-4.137080057882592]|[-4.583792910695042]|[1.4277420119456291]|[-6.878733110891416]|
|11215|BROOKLYN |[-3.707721352916707]|[-4.483240684423196]|[1.3178559497642031]|[-7.486766765529741]|


https://ssh.cloud.google.com/v2/
ssh.cloud.google.com/v2/ssh/projects/hpc-dataproc-19b8/zones/us-central1-f/instances/nyu-dataproc-m?hl=en_US&projectNumber=75588921087...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
scala> import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.linalg.Vector
scala> import org.apache.spark.sql.functions.udf
import org.apache.spark.sql.functions.udf
scala> val extractFirst = udf((v: Vector) => v(0))
extractFirst: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$5017/1359734668?c1b43af,DoubleType,List(Some(class[value[0]: vector])),Some(class[value[0]: double
]),None,false,true)
scala> val numericDF1 = finalDF.withColumn("propValNumeric", extractFirst($"propValScaled")).drop("propValScaled")
numericDF1: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 4 more fields]
scala> val numericDF2 = numericDF1.withColumn("emsCountNumeric", extractFirst($"emsCountScaled")).drop("emsCountScaled")
numericDF2: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 4 more fields]
scala> val numericDF3 = numericDF2.withColumn("leadAvgNumeric", extractFirst($"leadAvgScaled")).drop("leadAvgScaled")
numericDF3: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 4 more fields]
scala> val numericDF = numericDF3.withColumn("copperAvgNumeric", extractFirst($"copperAvgScaled")).drop("copperAvgScaled")
numericDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 4 more fields]
scala>
scala> import org.apache.spark.sql.functions.abs
import org.apache.spark.sql.functions.abs
scala> val positiveDF = numericDF.select($"zip", $"borough", $"propValNumeric", abs($"emsCountNumeric").as("emsCountNumeric"), abs($"leadAvgNumeric").as("leadAvgNumeric"), abs($"copperAvgNumeric").as(
"copperAvgNumeric"))
positiveDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 4 more fields]
scala>
scala> val positiveRDD = positiveDF.rdd
positiveRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[193] at rdd at <console>:27
scala> positiveRDD.collect().foreach(println)
[10301, STATEN ISLAND,0.9318568348969002,6.760612333168561,3.5072255465602877,4.559382005313233]
[11378, QUEENS,0.42939284068430617,8.841087905750536,4.1332277367124 25,4.460378380558869]
[11694, QUEENS,0.8026879745873389,8.994196943483276,4.687961985216473,4.546081634769981]
[11419, QUEENS,0.16755840098604194,7.843317474048443,4.663488415429 53,4.698575555941997]
[11223, BROOKLYN,1.1071912853422 97,7.102204852529248,3.7961496343228127,4.7349146252211 45]
[11212, BROOKLYN,0.09869255279631 24,0.6062057175811 51,4.871222406597 2745,4.7108033815673 46]
[11106, QUEENS,5.830982407147537,8.087977838194101,4.058932971287776,4.724377069950433]
[11217, BROOKLYN,1.427742011945629,6.878733110891416,4.137080057882592,4.583792910695042]
[11219, BROOKLYN,0.6485110702790353,7.897692165101 3345,4.628526172876754,4.762905006408974]
[11236, BROOKLYN,0.12076247593666055,5.153546712802768,3.902088466502 4056,4.520080217139 4985]
[11102, QUEENS,0.7075631473818654,8.505339635854341,3.497594743634 8704,4.763958761936 0455]




```
scala> val sumMapDF = positiveRDD.map(row => {
     |     val zip = row.getString(0)
     |     val borough = row.getString(1)
     |     val prop = row.getDouble(2)
     |     val ems = row.getDouble(3)
     |     val lead = row.getDouble(4)
     |     val copper = row.getDouble(5)
     |     (zip, borough, prop + ems + lead + copper)
     | }).toDF("zip", "borough", "sum")
sumMapDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 1 more field]

scala> sumMapDF.show(false)
+-----+-------------+-----------------+
|zip  |borough      |sum              |
+-----+-------------+-----------------+
|10301| STATEN ISLAND|15.7590767199389 82|
|11378| QUEENS      |17.8640868637061 35|
|11694| QUEENS      |19.0309285380570 7 |
|11419| QUEENS      |17.3293984640601 3 |
|11223| BROOKLYN    |16.7404603974155 04|
|11212| BROOKLYN    |10.2869240585420 84|
|11106| QUEENS      |22.7022702865798 43|
|11217| BROOKLYN    |17.0273480914146 8 |
|11219| BROOKLYN    |17.9376344146660 96|
|11236| BROOKLYN    |13.6964778723813 33|
|11102| QUEENS      |17.4744562888071 22|
|10305| STATEN ISLAND|17.0566542571752 96|
|11233| BROOKLYN    |11.1992155056595 2 |
|10471| BRONX       |21.2394539527239 95|
|11222| BROOKLYN    |18.4625008809591 34|
|11215| BROOKLYN    |16.9955847526338 46|
|10460| BRONX       |13.2044218928051 06|
|11104| QUEENS      |19.2322575075332 34|
|11372| QUEENS      |17.8163959047810 07|
|10018| MANHATTAN   |18.7469959699336 82|
+-----+-------------+-----------------+
only showing top 20 rows


scala>


scala> import org.apache.spark.sql.functions.round
import org.apache.spark.sql.functions.round

scala> val roundedDF = sumMapDF.withColumn("sum", round($"sum", 2))
roundedDF: org.apache.spark.sql.DataFrame = [zip: string, borough: string ... 1 more field]

scala>
```




```
scala> import org.apache.spark.sql.functions.desc
import org.apache.spark.sql.functions.desc

scala> val sortedDF = roundedDF.sort(desc("sum"))
sortedDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [zip: string, borough: string ... 1 more field]

scala> sortedDF.count()
res31: Long = 104

scala> sortedDF.show(104)
+-----+-------------+-----+
|  zip|      borough|  sum|
+-----+-------------+-----+
|11374|       QUEENS|26.94|
|11224|     BROOKLYN|24.39|
|11364|       QUEENS|23.89|
|11106|       QUEENS| 22.7|
|11101|       QUEENS|22.67|
|11369|       QUEENS|22.31|
|11362|       QUEENS|22.22|
|11356|       QUEENS|21.98|
|11232|     BROOKLYN|21.38|
|10471|        BRONX|21.24|
|11360|       QUEENS|20.78|
|11414|       QUEENS|20.33|
|11426|       QUEENS|20.25|
|10463|        BRONX|19.75|
|10465|        BRONX|19.66|
|10307| STATEN ISLAND|19.52|
|11104|       QUEENS|19.23|
|11694|       QUEENS|19.03|
|11357|       QUEENS|18.99|
|11377|       QUEENS|18.98|
|11423|       QUEENS|18.94|
|10018|    MANHATTAN|18.75|
|10308| STATEN ISLAND|18.72|
|11361|       QUEENS| 18.7|
|11427|       QUEENS|18.67|
|10309| STATEN ISLAND|18.61|
|11354|       QUEENS|18.59|
|10010|    MANHATTAN|18.59|
|11375|       QUEENS|18.57|
|11411|       QUEENS|18.54|
|11418|       QUEENS|18.49|
|11001|       QUEENS|18.46|
|11222|     BROOKLYN|18.46|
|11693|       QUEENS|18.41|
|10464|        BRONX| 18.4|
```

```
scala> sortedDF.write.format("csv")
res33: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@4946001a

scala>   .option("header", "true")
res34: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@4946001a

scala>   .save("hdfs://nyu-dataproc-m/user/ic2104_nyu_edu/final_project_18/results.csv")
```

```
ic2184_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -setfacl -R -m user:cr3152:rwx final_project_18
ic2184_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -setfacl -R -m default:user:cr3152:rwx final_project_18
ic2184_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -setfacl -R -m user:cl6405:rwx final_project_18
```

```
ic2184_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -setfacl -R -m default:user:cl6405:rwx final_project_18
ic2184_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -setfacl -R -m user:adm209:rwx final_project_18
ic2184_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -setfacl -R -m default:user:adm209:rwx final_project_18
ic2184_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -getfacl final_project_18
# file: final_project_18
# owner: ic2184_nyu_edu
# group: ic2184_nyu_edu
user::rwx
user:adm209:rwx
user:adm209_nyu_edu:r-x
user:cl6405:rwx
user:cl6405_nyu_edu:r-x
user:cr3152:rwx
user:cr3152_nyu_edu:r-x
group::---
mask::rwx
other::---
default:user::rwx
default:user:adm209:rwx
default:user:adm209_nyu_edu:r-x
default:user:cl6405:rwx
default:user:cl6405_nyu_edu:r-x
default:user:cr3152:rwx
default:user:cr3152_nyu_edu:r-x
default:group::---
default:mask::rwx
default:other::---

ic2184_nyu_edu@nyu-dataproc-m:~$
```