《金融大数据处理技术》实验3报告

嵇泽同 191870068

一、环境配置

Windows10 + Hadoop3.3.0 + Spark3.1.2 + IDEA + maven

Task1:pom.xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example</groupId>
    <artifactId>myhadoop3</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Hadoop -->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>3.3.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-hdfs</artifactId>
            <version>3.3.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-core</artifactId>
            <version>3.3.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-jobclient</artifactId>
            <version>3.3.0</version>
        </dependency>
    </dependencies>
</project>
```

Task2-4：pom.xml：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>SparkTest</artifactId>
  <version>1.0-SNAPSHOT</version>
  <inceptionYear>2008</inceptionYear>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spark.version>3.1.2</spark.version>
    <scala.version>2.12</scala.version>
    <hadoop.version>3.3.0</hadoop.version>
  </properties>

  <repositories>
    <repository>
      <id>scala-tools.org</id>
      <name>Scala-Tools Maven2 Repository</name>
      <url>http://scala-tools.org/repo-releases</url>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>scala-tools.org</id>
      <name>Scala-Tools Maven2 Repository</name>
      <url>http://scala-tools.org/repo-releases</url>
    </pluginRepository>
  </pluginRepositories>

  <dependencies>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_${scala.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_${scala.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-hive_${scala.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-streaming_${scala.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>3.3.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-mllib_${scala.version}</artifactId>
      <version>${spark.version}</version>
    </dependency>
    <dependency>
      <groupId>org.scala-lang</groupId>
      <artifactId>scala-library</artifactId>
      <version>2.12.15</version>
    </dependency>
    <dependency>
      <groupId>org.scala-tools</groupId>
      <artifactId>maven-scala-plugin</artifactId>
      <version>2.12</version>
    </dependency>
    <dependency>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-eclipse-plugin</artifactId>
      <version>2.5.1</version>
    </dependency>
    <dependency>
```

```
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.4</version>
        <scope>test</scope>
      </dependency>
      <dependency>
        <groupId>org.specs</groupId>
        <artifactId>specs</artifactId>
        <version>1.2.5</version>
        <scope>test</scope>
      </dependency>
    </dependencies>

    <build>
      <sourceDirectory>src/main/scala</sourceDirectory>
      <testSourceDirectory>src/test/scala</testSourceDirectory>
      <plugins>
        <plugin>
          <groupId>org.scala-tools</groupId>
          <artifactId>maven-scala-plugin</artifactId>
          <executions>
            <execution>
              <goals>
                <goal>compile</goal>
                <goal>testCompile</goal>
              </goals>
            </execution>
          </executions>
          <configuration>
            <scalaVersion>2.12.15</scalaVersion>
            <args>
              <arg>-target:jvm-1.5</arg>
            </args>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-eclipse-plugin</artifactId>
          <configuration>
            <downloadSources>true</downloadSources>
            <buildcommands>
              <buildcommand>ch.epfl.lamp.sdt.core.scalabuilder</buildcommand>
            </buildcommands>
            <additionalProjectnatures>
              <projectnature>ch.epfl.lamp.sdt.core.scalanature</projectnature>
            </additionalProjectnatures>
            <classpathContainers>

 <classpathContainer>org.eclipse.jdt.launching.JRE_CONTAINER</classpathContainer
>

 <classpathContainer>ch.epfl.lamp.sdt.launching.SCALA_CONTAINER</classpathContai
ner>
            </classpathContainers>
          </configuration>
        </plugin>
      </plugins>
    </build>
```

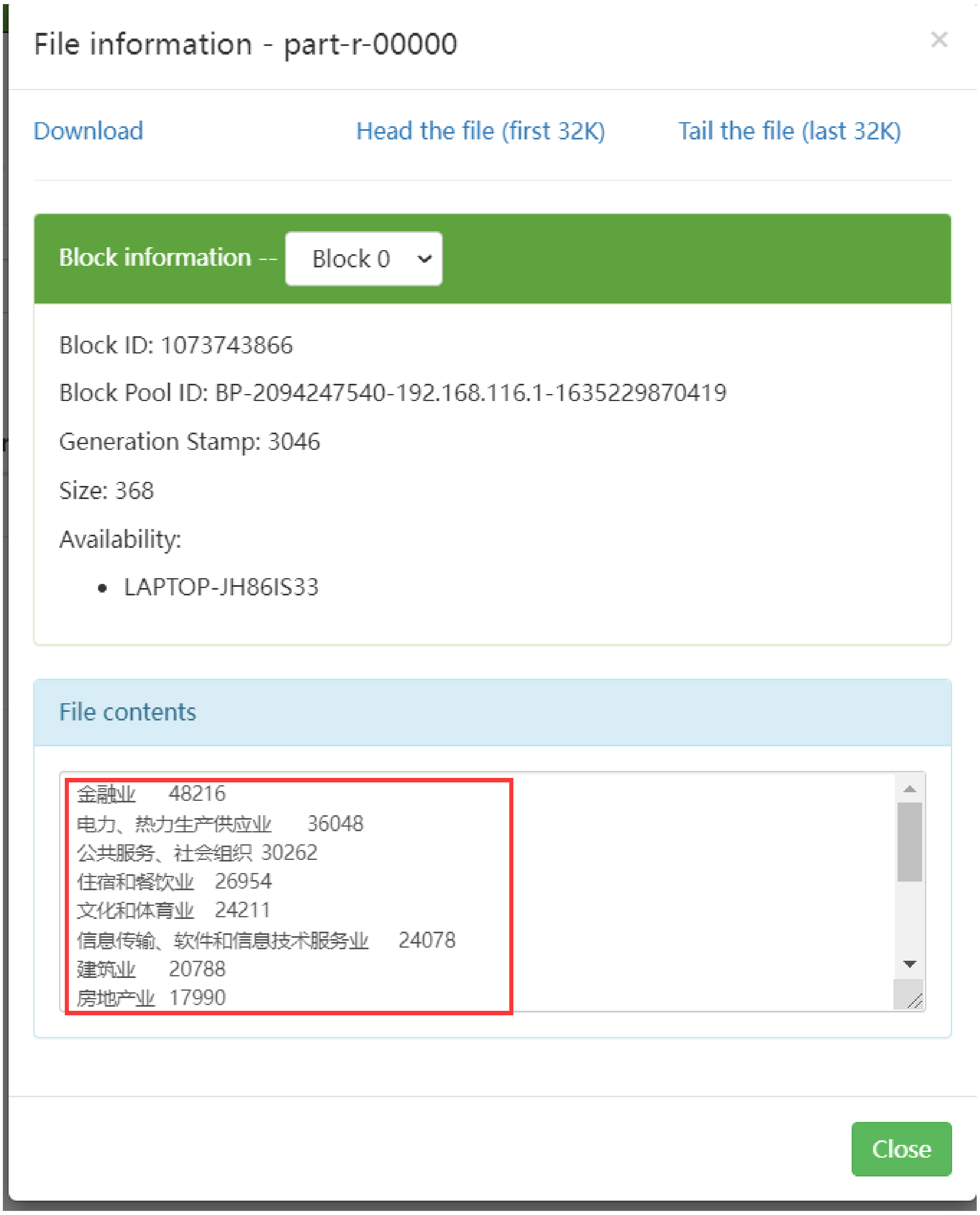
```
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.scala-tools</groupId>
        <artifactId>maven-scala-plugin</artifactId>
        <configuration>
          <scalaVersion>${scala.version}</scalaVersion>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
</project>
```

二、任务一

要求：

编写 MapReduce 程序，统计每个工作领域 industry 的网贷记录的数量，并按数量从大到小进行排序。

结果展示：


File information - part-r-00000

Download   Head the file (first 32K)   Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073743866

Block Pool ID: BP-2094247540-192.168.116.1-1635229870419

Generation Stamp: 3046

Size: 368

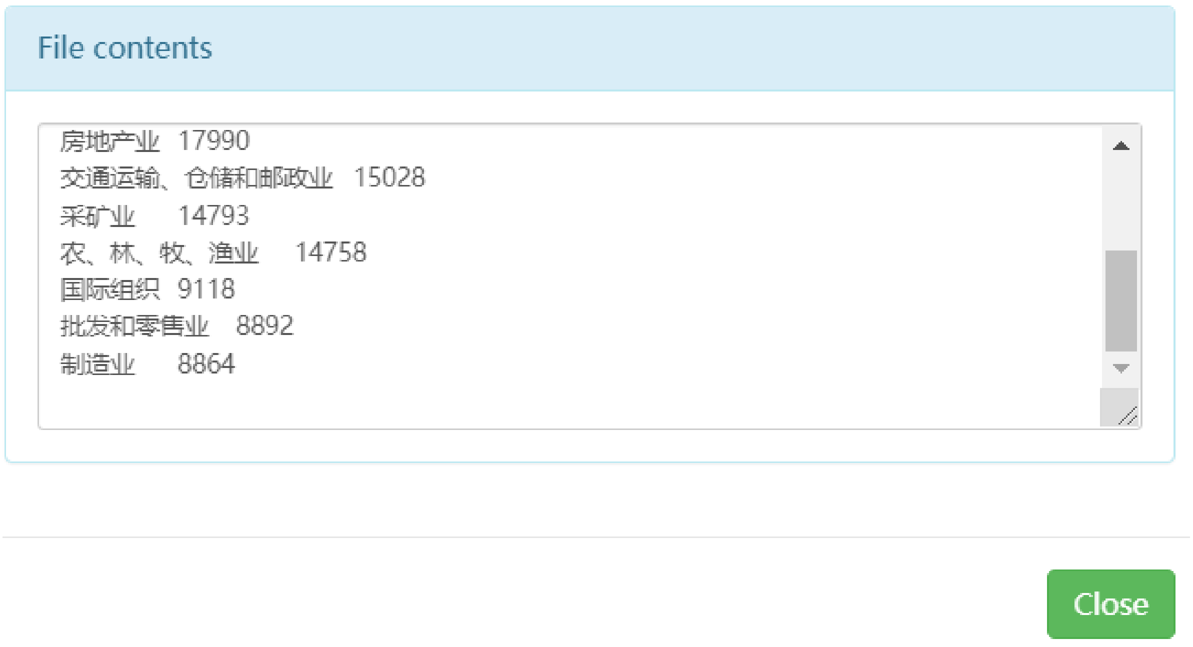
Availability:

- LAPTOP-JH86IS33

File contents

```
金融业	48216
电力、热力生产供应业	36048
公共服务、社会组织	30262
住宿和餐饮业	26954
文化和体育业	24211
信息传输、软件和信息技术服务业	24078
建筑业	20788
房地产业	17990
```

Close



```
房地产业	17990
交通运输、仓储和邮政业	15028
采矿业	14793
农、林、牧、渔业	14758
国际组织	9118
批发和零售业	8892
制造业	8864
```

实现思路：

类似于之前实现过的排序版词频统计，运用ChainMapper和ChainReducer：

```
job.setJarByClass(IndustryDefault.class);
//  Mapper1
Configuration map1Conf = new Configuration(false);
ChainMapper.addMapper(job,IndustryMapper.class,Object.class,Text.class,Text.clas
s,IntWritable.class,map1Conf);
//  Reducer1
Configuration reduce1Conf = new Configuration(false);
ChainReducer.setReducer(job,IndustryReducer.class,Text.class,IntWritable.class,T
ext.class,IntWritable.class, reduce1Conf);
//  Mapper2
Configuration map2Conf = new Configuration(false);
ChainReducer.addMapper(job,SortMapper.class,Text.class,IntWritable.class,Text.cl
ass,IntWritable.class,map2Conf);
```

Mapper1用来从原始文档中提取industry属性，输出<industry,1>格式；Reducer1接收Mapper1的输出，并且累加统计出每个industry出现的总数，输出<industry,num>格式；Mapper2接收Reducer1的输出，此时Reducer1的输出是乱序的，Mapper2负责接收这些乱序键值对并将其存入内存中的HashMap，并最后对所有存入的键值对进行排序并输出。这一部分在内存中进行，因为虽然原始数据量很大，但经过Mapper1和Reducer1的处理，此时接收到的数据仅为有限的<industry,num>键值对，数据量较小，可以在内存中进行处理。

Mapper1：

对原始文本按照","进行分词，并提取出industry属性

```java
public static class IndustryMapper extends Mapper<Object, Text, Text,
IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String[] words = value.toString().split(",");
            String industry = words[10];
            if (!industry.equals("industry"))
                context.write(new Text(industry), one);
        }
    }
```

Reducer1:

简单地对Mapper1输出的<industry,1>进行累加并输出

```java
public static class IndustryReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable value:values) {
                sum = sum + value.get();
            }
            context.write(key,new IntWritable(sum));
        }
    }
```

Mapper2:

建立HashMap存储Reucer1传过来的<industry,num>，接收并存储完毕后进行排序，将排好序的<industry,num>键值对输出

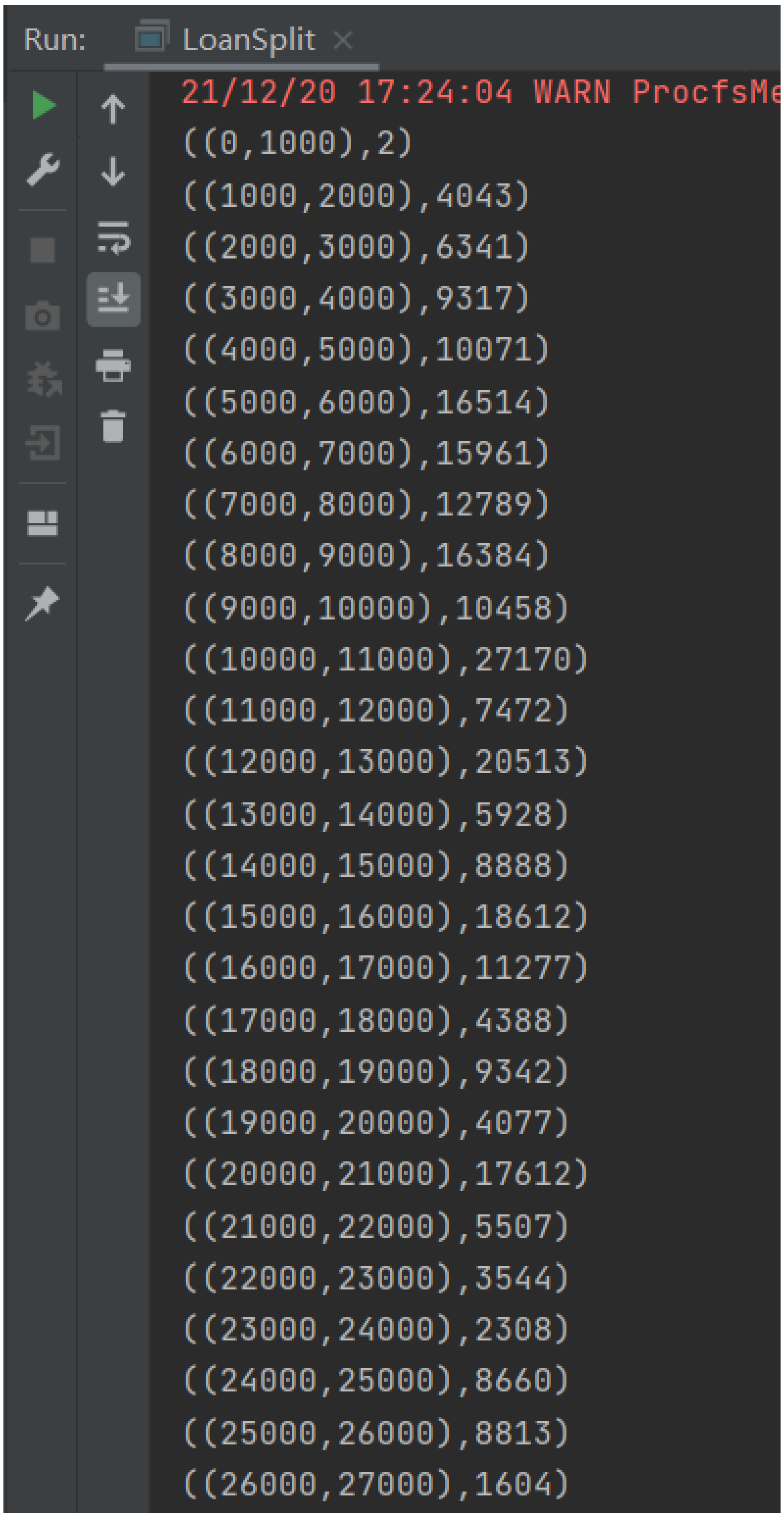
```java
public static class SortMapper extends Mapper<Text, IntWritable, Text,
IntWritable> {
        HashMap<String, Integer> hm;
        protected void setup(Context context) throws IOException,
InterruptedException{
            hm = new HashMap<String, Integer>();
        }

        public void map(Text key, IntWritable value, Context context) throws
IOException, InterruptedException {
            hm.put(key.toString(),value.get());
        }

        protected void cleanup(Context context) throws IOException,
InterruptedException {
            List<Map.Entry<String, Integer>> list = new
ArrayList<Map.Entry<String, Integer>>(hm.entrySet());
            list.sort(new Comparator<Map.Entry<String, Integer>>() {
                @Override
                public int compare(Map.Entry<String, Integer> o1,
Map.Entry<String, Integer> o2) {
                    return o2.getValue().compareTo(o1.getValue());
                }
            });
            for (Map.Entry<String, Integer> entry: list) {
                context.write(new Text(entry.getKey()), new
IntWritable(entry.getValue()));
            }
        }
    }
```

三、任务二

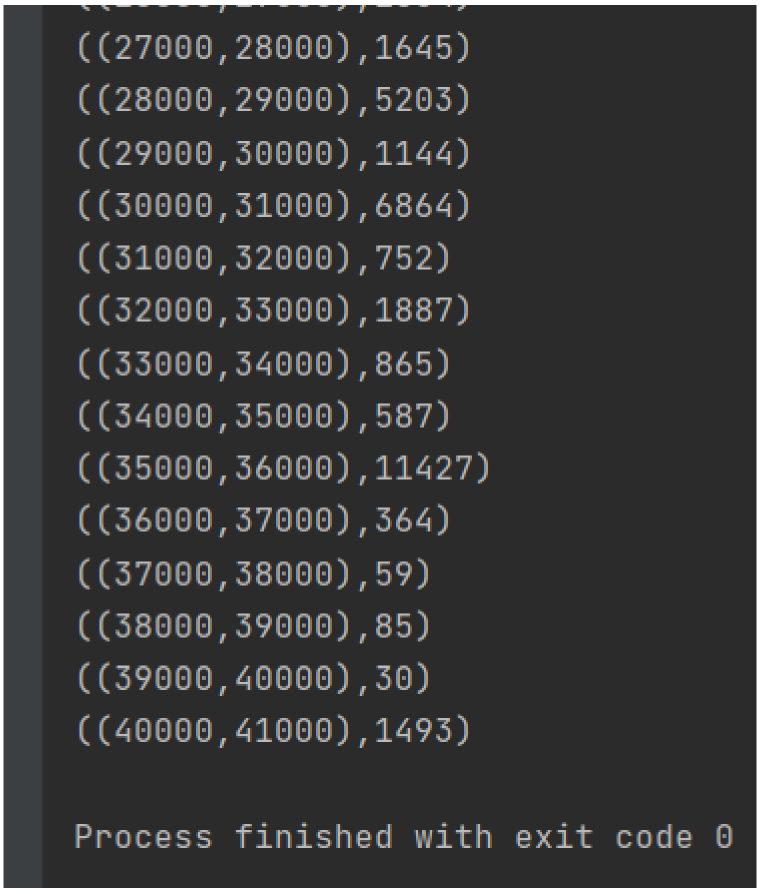
要求：

编写 Spark 程序，统计网络信用贷产品记录数据中所有用户的贷款金额 total_loan 的分布情况。以 1000 元为区间进行输出。

结果展示：



```
21/12/20 17:24:04 WARN ProcfsMe
((0,1000),2)
((1000,2000),4043)
((2000,3000),6341)
((3000,4000),9317)
((4000,5000),10071)
((5000,6000),16514)
((6000,7000),15961)
((7000,8000),12789)
((8000,9000),16384)
((9000,10000),10458)
((10000,11000),27170)
((11000,12000),7472)
((12000,13000),20513)
((13000,14000),5928)
((14000,15000),8888)
((15000,16000),18612)
((16000,17000),11277)
((17000,18000),4388)
((18000,19000),9342)
((19000,20000),4077)
((20000,21000),17612)
((21000,22000),5507)
((22000,23000),3544)
((23000,24000),2308)
((24000,25000),8660)
((25000,26000),8813)
((26000,27000),1604)
```



```
((27000,28000),1645)
((28000,29000),5203)
((29000,30000),1144)
((30000,31000),6864)
((31000,32000),752)
((32000,33000),1887)
((33000,34000),865)
((34000,35000),587)
((35000,36000),11427)
((36000,37000),364)
((37000,38000),59)
((38000,39000),85)
((39000,40000),30)
((40000,41000),1493)

Process finished with exit code 0
```

实现思路：

关键在于从每一行的原始数据中提取出total_loan的值，并且得出其属于哪个区间：

```
//  该方法接收一行原始数据作为输入参数，返回"(2000,3000)"形式的字符串
def splitTotalLoan(record: String): String = {
    val totalLoan = record.split(",")(2)
    //  获取到的totalLoan为double类型的字符串，需要提取其整数部分
    val thousand = totalLoan.split("""\.""")(0).toInt / 1000
    val lowBound = thousand * 1000
    val highBound = (thousand+1) * 1000
    return "("+lowBound+","+highBound+")"
  }
```

能够从每一行原始数据中提取出对应的字符串后，接下来的步骤就类似词频统计，对每一行map即可：

```
val splitTL = splitTotalLoan _
dataWithoutHeader.map(splitTL) //  (2000,3000)
  .map((_,1)) //  ((2000,3000),1)
  .reduceByKey(_+_) //  ((2000,3000),num)
  .sortBy(_._1.split(",")(0).split("""\(""")(1).toInt)  //  排序
  .collect()
  .foreach(println)
```

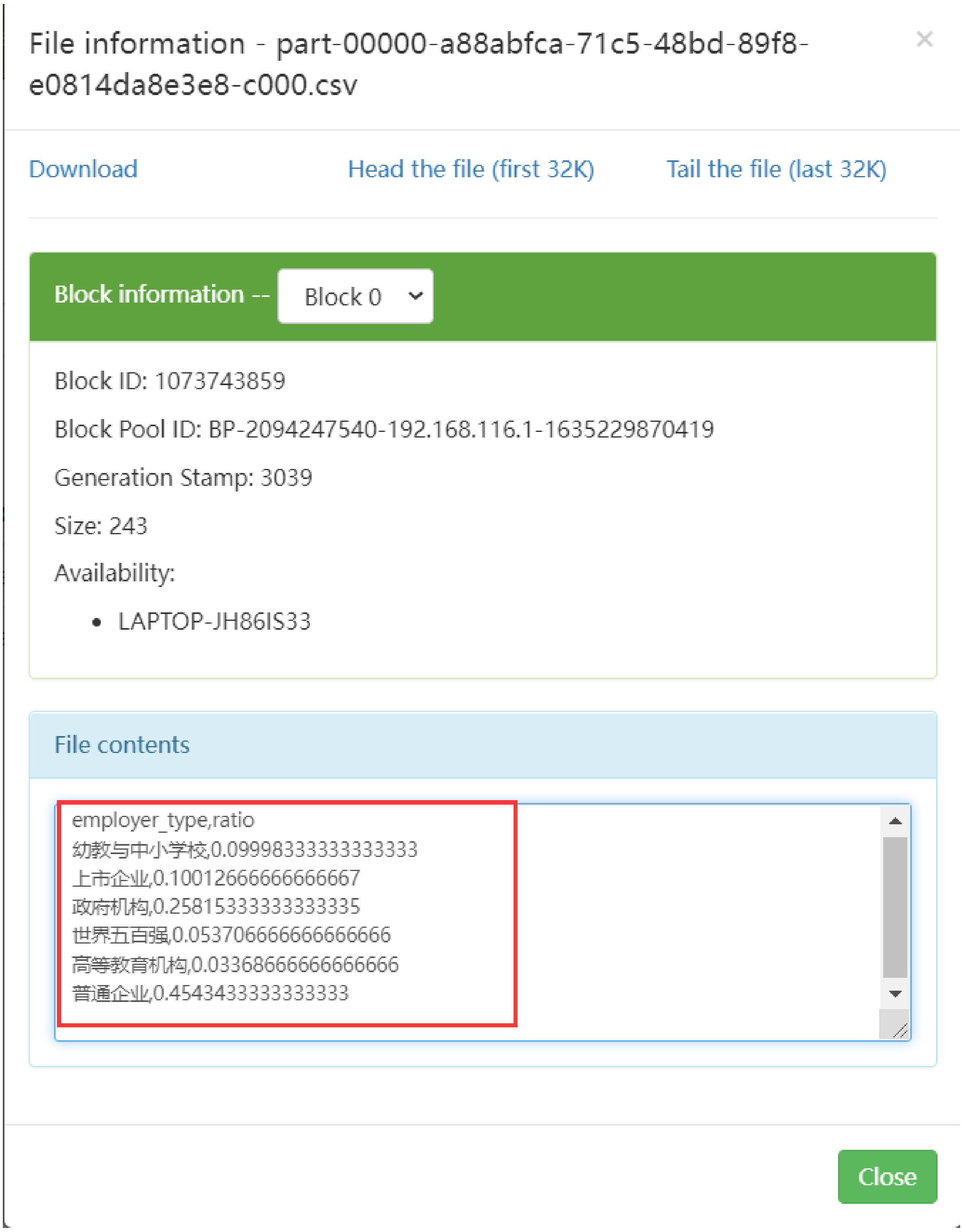
四、任务三

Part1

要求：

统计所有用户所在公司类型 employer_type 的数量分布占比情况。

结果展示：


File information - part-00000-a88abfca-71c5-48bd-89f8-e0814da8e3e8-c000.csv

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073743859

Block Pool ID: BP-2094247540-192.168.116.1-1635229870419

Generation Stamp: 3039

Size: 243

Availability:

- LAPTOP-JH86IS33

File contents

```
employer_type,ratio
幼教与中小学校,0.09998333333333333
上市企业,0.10012666666666667
政府机构,0.25815333333333335
世界五百强,0.053706666666666666
高等教育机构,0.033686666666666666
普通企业,0.4543433333333333
```

实现思路：

要运用SparkSQL，首先要读入数据。我采用的是编程方式指定类型：

```
val schema = new StructType(Array(
      StructField("loan_id",DataTypes.StringType),
      StructField("user_id",DataTypes.StringType),
      StructField("total_loan",DataTypes.DoubleType),
      StructField("year_of_loan",DataTypes.IntegerType),
      StructField("interest",DataTypes.DoubleType),
      StructField("monthly_payment",DataTypes.DoubleType),
      StructField("loan_class",DataTypes.StringType),
      StructField("sub_class",DataTypes.StringType),
      StructField("work_type",DataTypes.StringType),
      StructField("employer_type",DataTypes.StringType),
      StructField("industry",DataTypes.StringType),
      StructField("work_year",DataTypes.StringType),
      StructField("house_exist",DataTypes.IntegerType),
      StructField("house_loan_status",DataTypes.IntegerType),
      StructField("censor_status",DataTypes.IntegerType),
      StructField("marriage",DataTypes.IntegerType),
      StructField("offsprings",DataTypes.IntegerType),
      StructField("issue_date",DataTypes.StringType),
      StructField("use",DataTypes.IntegerType),
      StructField("post_code",DataTypes.StringType),
      StructField("region",DataTypes.StringType),
      StructField("debt_loan_ratio",DataTypes.DoubleType),
      StructField("del_in_18month",DataTypes.IntegerType),
      StructField("scoring_low",DataTypes.IntegerType),
      StructField("scoring_high",DataTypes.IntegerType),
      StructField("pub_dero_bankrup",DataTypes.IntegerType),
      StructField("early_return",DataTypes.IntegerType),
      StructField("early_return_amount",DataTypes.IntegerType),
      StructField("early_return_amount_3mon",DataTypes.DoubleType),
      StructField("recircle_b",DataTypes.IntegerType),
      StructField("recircle_u",DataTypes.DoubleType),
      StructField("initial_list_status",DataTypes.IntegerType),
      StructField("earlies_credit_mon",DataTypes.StringType),
      StructField("title",DataTypes.IntegerType),
      StructField("policy_code",DataTypes.IntegerType),
      StructField("f0",DataTypes.IntegerType),
      StructField("f1",DataTypes.IntegerType),
      StructField("f2",DataTypes.IntegerType),
      StructField("f3",DataTypes.IntegerType),
      StructField("f4",DataTypes.IntegerType),
      StructField("f5",DataTypes.IntegerType),
      StructField("is_default",DataTypes.BooleanType),
    ))

val df=spark.read.schema(schema)
.format("csv")
.option("header","true")
.load("hdfs://127.0.0.1:8900/user/jzt/train_data")

df.createOrReplaceTempView("loan")
```

在将数据保存为Dataframe之后，要获取不同employer_type的占比就很容易了，一句“GROUP BY employer_type”的SQL语句就可以完成：

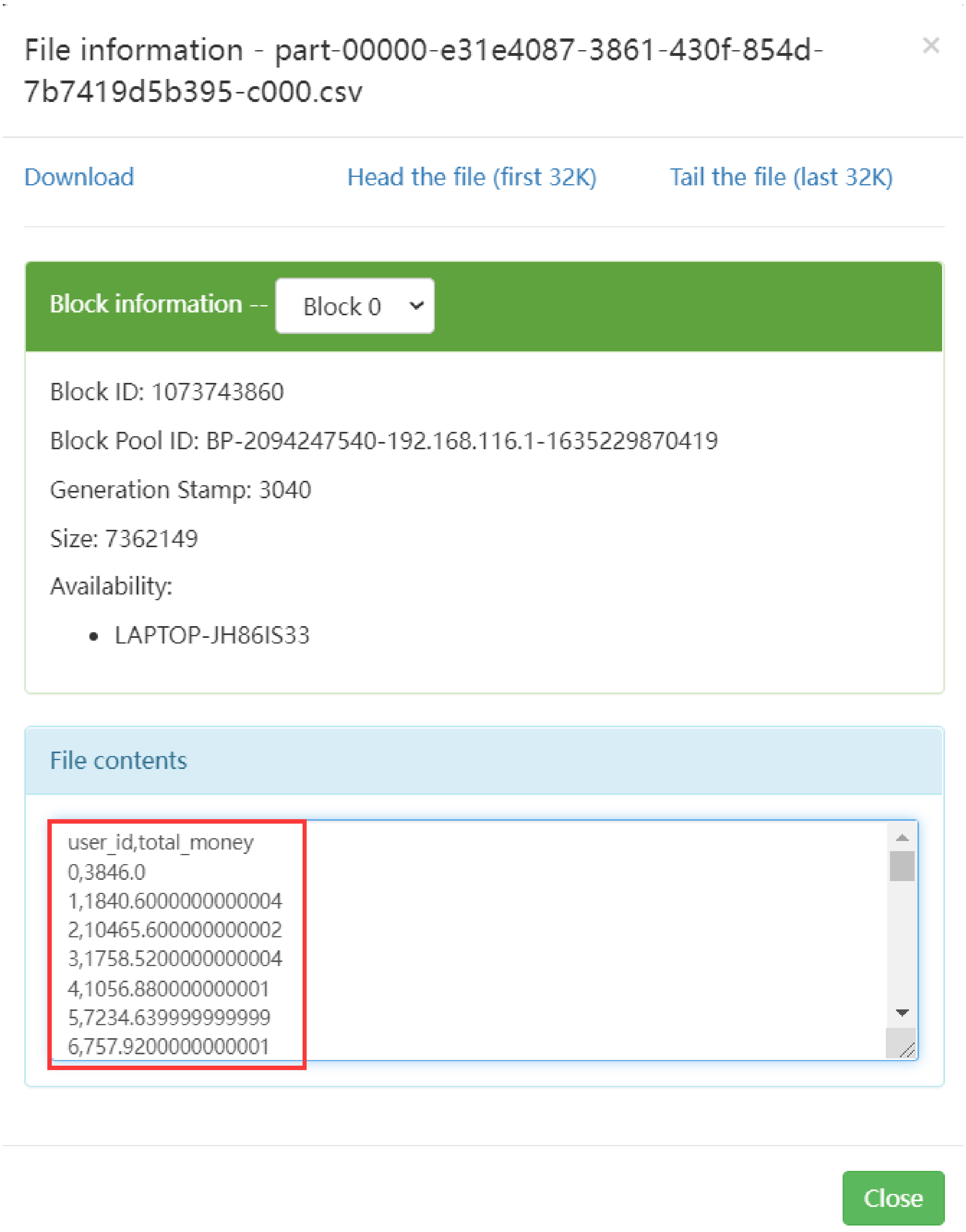
```
val emp_type_ratio = spark.sql("SELECT employer_type, COUNT(*)/(" +
      "SELECT COUNT(*) FROM loan" +
      ") AS ratio FROM loan GROUP BY employer_type")

 emp_type_ratio.coalesce(1).write.option("header","true").csv("hdfs://127.0.0.1:
8900/user/jzt/emp_type_ratio.csv")
```

值得注意的是，在保存结果时如果不调用.coalesce(1)，则结果会被分片保存到多个文件中，加上.coalesce(1)之后则会保存到一个文件中，两种保存方式在实际中各有其应用场景，此处我选择将其保存到同一个文件中。

Part2

要求：统计每个用户最终须缴纳的利息金额。

结果展示：



File information - part-00000-e31e4087-3861-430f-854d-7b7419d5b395-c000.csv

Download   Head the file (first 32K)   Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073743860

Block Pool ID: BP-2094247540-192.168.116.1-1635229870419

Generation Stamp: 3040

Size: 7362149

Availability:

- LAPTOP-JH86IS33

File contents

```
user_id,total_money
0,3846.0
1,1840.6000000000004
2,10465.600000000002
3,1758.5200000000004
4,1056.880000000001
5,7234.639999999999
6,757.9200000000001
```

实现思路：

由于先前导入数据为Dataframe时已经显式指定了各列的数据类型，因此在本问题中直接运算即可，直接一句SQL语句就能得出结果：

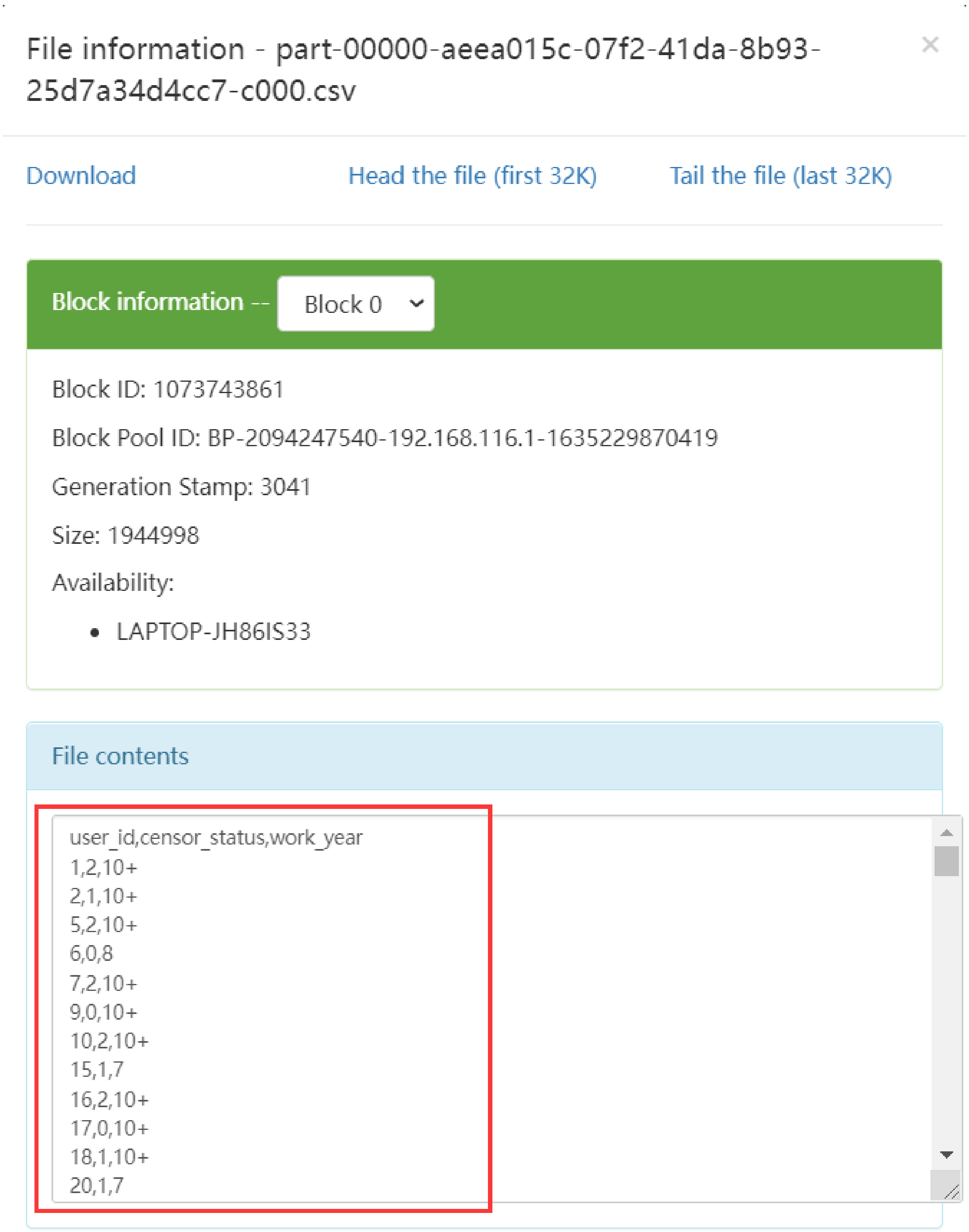
```
//    2.统计每个用户最终须缴纳的利息金额。
    val id_total_mon = spark.sql("SELECT user_id, (year_of_loan  *
monthly_payment * 12.0 - total_loan) AS total_money FROM loan")

 id_total_mon.coalesce(1).write.option("header","true").csv("hdfs://127.0.0.1:89
00/user/jzt/id_total_mon.csv")
```

Part3

要求:

统计工作年限 work_year 超过 5 年的用户的房贷情况 censor_status 。

结果展示:



File information - part-00000-aeea015c-07f2-41da-8b93-25d7a34d4cc7-c000.csv

Download　Head the file (first 32K)　Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073743861

Block Pool ID: BP-2094247540-192.168.116.1-1635229870419

Generation Stamp: 3041

Size: 1944998

Availability:

- LAPTOP-JH86IS33

File contents

```
user_id,censor_status,work_year
1,2,10+
2,1,10+
5,2,10+
6,0,8
7,2,10+
9,0,10+
10,2,10+
15,1,7
16,2,10+
17,0,10+
18,1,10+
20,1,7
```

实现思路:

由于原始数据中工作年限work_year是以字符串“x years”的形式保存的，因此想要对工作年限进行筛选的话，最好先用withColumn提取出字符串对应的数字并另成一列以供后续查询。（原始数据中有一些行的工作年限work_year为缺失值，我采取的方式是直接将这些缺失值全部填补为0。）要提取出字符串中的数字，直接用正则表达式“[^0-9]”是一种选择，该表达式可以只保留原始字符串中的数字。但需要注意原始数据中不光有“x years”这样的形式，还有"<1 year"和"10+ years"，若直接用上述正则表达式，则

前两者分别会被处理为“1”和“10”，但是<1不等于1,10+也同样不等于10，针对这种情况，我选择将正则表达式设置为"[^0-9<+]"，这样在后续筛选时只需判断处理后的结果是否为"10+"或者>5即可。

```
val updatedDf =
df.na.fill(value="0",cols=Array("work_year")).withColumn("work_year",
regexp_replace
    (col("work_year"), "[^0-9+<]", ""))
    updatedDf.createOrReplaceTempView("updated_loan")
    val censor_status_over5 = spark.sql("SELECT user_id, censor_status,
work_year FROM updated_loan WHERE " +
      "work_year='10+' OR work_year>5")

 censor_status_over5.coalesce(1).write.option("header","true").csv("hdfs://127.0
.0.1:8900/user/jzt/censor_status_over5.csv")
```

五、任务四

要求：

根据给定的数据集，基于 Spark MLlib 或者Spark ML编写程序预测有可能违约的借贷人，并评估实验结果的准确率。

数据加载：

和任务三一样，通过编程显式指定数据类型的方法加载数据：

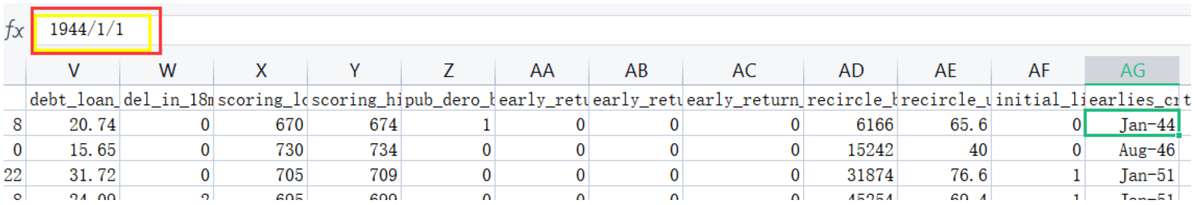
```
val schema = new StructType(Array(
      StructField("loan_id",DataTypes.StringType),
      StructField("user_id",DataTypes.StringType),
      StructField("total_loan",DataTypes.DoubleType),
      StructField("year_of_loan",DataTypes.IntegerType),
      StructField("interest",DataTypes.DoubleType),
      StructField("monthly_payment",DataTypes.DoubleType),
      StructField("loan_class",DataTypes.StringType),
      StructField("sub_class",DataTypes.StringType),
      StructField("work_type",DataTypes.StringType),
      StructField("employer_type",DataTypes.StringType),
      StructField("industry",DataTypes.StringType),
      StructField("work_year",DataTypes.StringType),
      StructField("house_exist",DataTypes.IntegerType),
      StructField("house_loan_status",DataTypes.IntegerType),
      StructField("censor_status",DataTypes.IntegerType),
      StructField("marriage",DataTypes.IntegerType),
      StructField("offsprings",DataTypes.IntegerType),
      StructField("issue_date",DataTypes.StringType),
      StructField("use",DataTypes.IntegerType),
      StructField("post_code",DataTypes.StringType),
      StructField("region",DataTypes.StringType),
      StructField("debt_loan_ratio",DataTypes.DoubleType),
      StructField("del_in_18month",DataTypes.IntegerType),
      StructField("scoring_low",DataTypes.IntegerType),
      StructField("scoring_high",DataTypes.IntegerType),
      StructField("pub_dero_bankrup",DataTypes.IntegerType),
      StructField("early_return",DataTypes.IntegerType),
      StructField("early_return_amount",DataTypes.IntegerType),
      StructField("early_return_amount_3mon",DataTypes.DoubleType),
      StructField("recircle_b",DataTypes.IntegerType),
      StructField("recircle_u",DataTypes.DoubleType),
      StructField("initial_list_status",DataTypes.IntegerType),
      StructField("earlies_credit_mon",DataTypes.StringType),
      StructField("title",DataTypes.IntegerType),
      StructField("policy_code",DataTypes.IntegerType),
      StructField("f0",DataTypes.IntegerType),
      StructField("f1",DataTypes.IntegerType),
      StructField("f2",DataTypes.IntegerType),
      StructField("f3",DataTypes.IntegerType),
      StructField("f4",DataTypes.IntegerType),
      StructField("f5",DataTypes.IntegerType),
      StructField("is_default",DataTypes.BooleanType),
    ))

val df=spark.read.schema(schema)
.format("csv")
.option("header","true")
.load("hdfs://127.0.0.1:8900/user/jzt/train_data")

df.createOrReplaceTempView("loan")
```

数据处理

①原始数据中"class","sub_class","work_type","employer_type","industry"这几个字段是字符串类型，可以通过StringIndexer将其按照出现频率转化为0、1、2、3这样的数字；

```
val indexed =new StringIndexer()

.setInputCols(Array("class","sub_class","work_type","employer_type","industry"))

.setOutputCols(Array("indexed_class","indexed_sub_class","indexed_work_type","indexed_employer_type",
        "indexed_industry"))
      .setHandleInvalid("keep")
      .fit(df)
      .transform(df)
      .drop("class","sub_class","work_type","employer_type","industry")
```

②loan_id和user_id显然只是身份标识，对于预测是否违约没有任何帮助，故直接去除这两个特征；同样，post_code(借款人邮政编码的前三位)和title(借款人提供的网络贷款名称)在本人看来对于预测也没有帮助，故同样去除。此外，注意到earlies_credit_line(网络贷款信用额度开立的月份)这个特征的数据很奇怪：



fx 1944/1/1

	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG
	debt_loan_	del_in_18m	scoring_lo	scoring_hi	pub_dero_b	early_retu	early_retu	early_return_	recircle_b	recircle_u	initial_li	earlies_cr
8	20.74	0	670	674	1	0	0	0	6166	65.6	0	Jan-44
0	15.65	0	730	734	0	0	0	0	15242	40	0	Aug-46
22	31.72	0	705	709	0	0	0	0	31874	76.6	1	Jan-51
8	24.09	2	695	699	0	0	0	0	45254	69.4	1	Jan-51

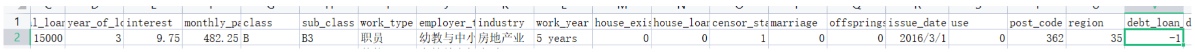
> “信息时代的到来 ——1946年电子计算机的诞生1946年2月15日,世界上第一台电子计算机在美国宾夕法尼亚大学正式投入使用。它的正式名称叫“电子数字积分器和计算机”(ENIAC),是一个重达30吨的庞然大物,有两层楼高,占地170平方米,装有1.8万只电子管、7万只电阻、1万只电容...”
> 计算机是何时诞生的?_百度知道

电子计算机1946年才出现，但是数据却显示有人在1944年就开立了贷款信用额度！

因此考虑到这个特征可能存在程度未知的污染，且又因为贷款信用额度开立月份这个特征在直觉上和是否违约不存在什么关联，因此同样去除这个特征。

```
val indexed =new StringIndexer()
      ...
      .drop("loan_id","user_id","post_code","title","earlies_credit_mon")
```

③注意到debt_loan_ratio(债务收入比)这个特征，应当是非负数，但是原始数据中该特征存在负数：



	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	l_loar	year_of_lc	interest	monthly_pa	class	sub_class	work_type	employer_t	industry	work_year	house_exis	house_loar	censor_sta	marriage	offsprings	issue_date	use	post_code	region	debt_loan_
2	15000	3	9.75	482.25	B	B3	职员	幼教与中小	房地产业	5 years	0	0	1	0	0	2016/3/1	0	362	35	-1

需要对这类脏数据进行过滤：

```
val indexed =new StringIndexer()
      ...
      .filter("debt_loan_ratio >= 0")
```

④work_year这个特征在原始数据中为字符串，而且还存在缺失，故对其进行映射处理：

```
def parseWorkYear(workYear:String) = {
      workYear match {
        case "< 1 year" => 0
        case "1 year" => 1
        case "2 years" => 2
        case "3 years" => 3
        case "4 years" => 4
        case "5 years" => 5
        case "6 years" => 6
        case "7 years" => 7
        case "8 years" => 8
        case "9 years" => 9
        case "10+ years" => 10
        case _ => 0
      }
    }

    val udf_parse_work_year = udf(parseWorkYear _)

    val indexed2 = indexed.withColumn("parsed_work_year",
udf_parse_work_year(indexed("work_year")))
                      .drop("work_year")
```

⑤原始数据中issue_date为日期形式，不利于建设特征向量，故采取取离差的方式，取issue_date和“2007/7/1”（所有issue_date中的最小值）的日差作为特征：

```
val indexed2 = indexed....
                    .withColumn("issue_date_diff",
datediff(indexed("issue_date"), lit("2007-07-01")))
                    .drop("issue_date")
```

⑥原始数据中的total_loan数值较大，作全部除以1000的处理：

```
val indexed2 = indexed....
                    .withColumn("total_loan", indexed("total_loan")/1000)
                    .withColumnRenamed("is_default","label")
```

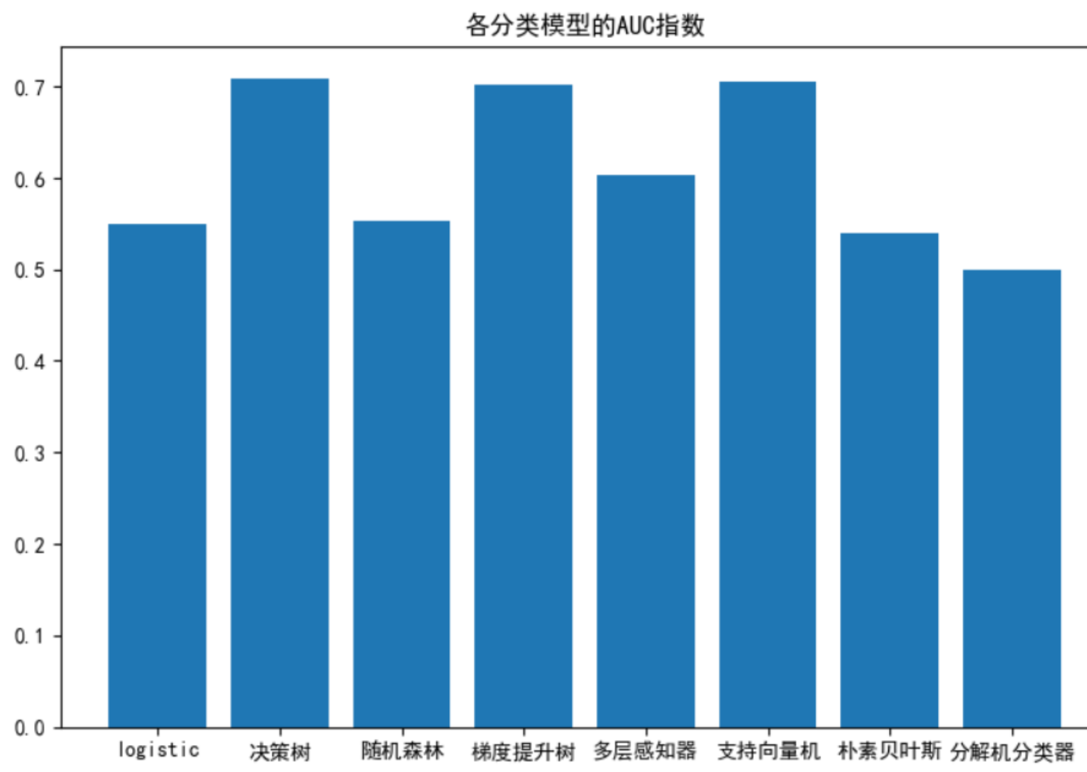
至此，数据处理已基本完成，将所有处理后的特征保存为一个特征向量：

```
val assembler: VectorAssembler = new
VectorAssembler().setHandleInvalid("skip").setInputCols(Array(
      "monthly_payment","house_exist","house_loan_status","censor_status",

 "marriage","offsprings","use","debt_loan_ratio","del_in_18month","scoring_low",
"scoring_high",

 "initial_list_status","pub_dero_bankrup","early_return","early_return_amount","
early_return_amount_3mon","recircle_b","recircle_u",
      "policy_code","f0","f1","f2","f3","f4","f5","indexed_industry",

 "indexed_class","indexed_work_type","indexed_employer_type","parsed_work_year"
    )).setOutputCol("features")

    val assmblerDf:DataFrame = assembler.transform(indexed2)
```

模型评估

将上述处理后的数据喂入ml库中的若干分类模型，参数均为spark官网上给出的示例参数，由于是二分类问题，故以areaUnderROC（AUC）作为评价指标。

尝试过的模型及对应的平均AUC分别为：logistic回归模型（AUC=0.5）、决策树模型（AUC=0.708）、随机森林模型（AUC=0.553）、梯度提升树分类器（AUC=0.702）、多层感知器分类器（AUC=0.604）、线性支持向量机（AUC=0.706）、朴素贝叶斯分类（AUC=0.539）、分解机分类器（AUC=0.500）



以线性支持向量机模型为例，给出示例代码如下：

```
  val splits = assmblerDf.randomSplit(Array(0.8, 0.2))
  val train = splits(0)
  val test = splits(1)

  val lsvc = new LinearSVC()
      .setMaxIter(10)
      .setRegParam(0.1)

  val lsvcModel = lsvc.fit(train)
  val result = lsvcModel.transform(test)
  val evaluator2 = new
BinaryClassificationEvaluator().setMetricName("areaUnderROC")
  val accuracy2 = evaluator2.evaluate(result)
  print(accuracy2)
```

对于AUC指数最高的决策树和线性支持向量机模型，我再次尝试探索进一步提升模型表现。

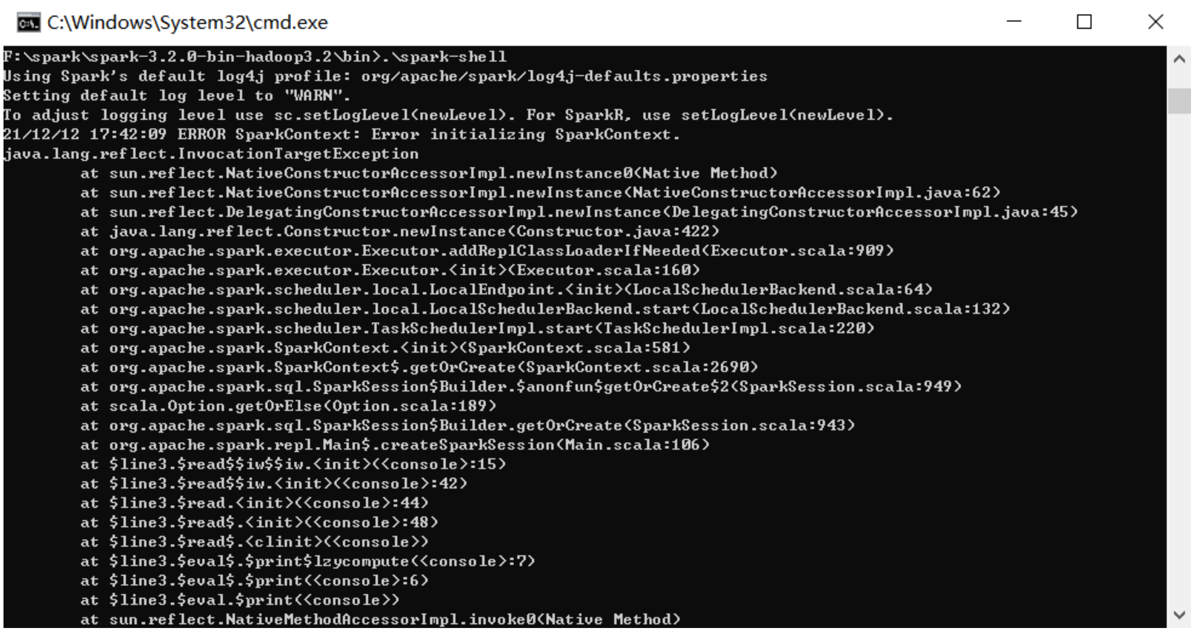
本次实验中我采取的是最直接的缩减特征个数方法。考虑到total_loan,year_of_loan,interest三者决定了monthly_payment，而monthly_payment（每月需支付额）更直接地衡量了还债压力，故尝试直接去掉total_loan,year_of_loan,interest这三个特征，只保留monthly_payment特征。结果发现AUC指数有了显著提升，决策树模型的平均AUC从0.708提高到0.742，线性支持向量机模型的平均AUC从0.706提高到0.722。

之后我又尝试了一些其他的改进方法，结果效果不甚明显。考虑到本次实验目的不在于最大化模型性能，且ml库的模型较为固定，能对模型加以改动的地方基本只有参数，故没有再继续进行更加深入的研究探索。

六、实验中遇到的问题

①spark-shell启动报错

在windows10下安装并运行spark3.2.0的shell时报出如下错误：



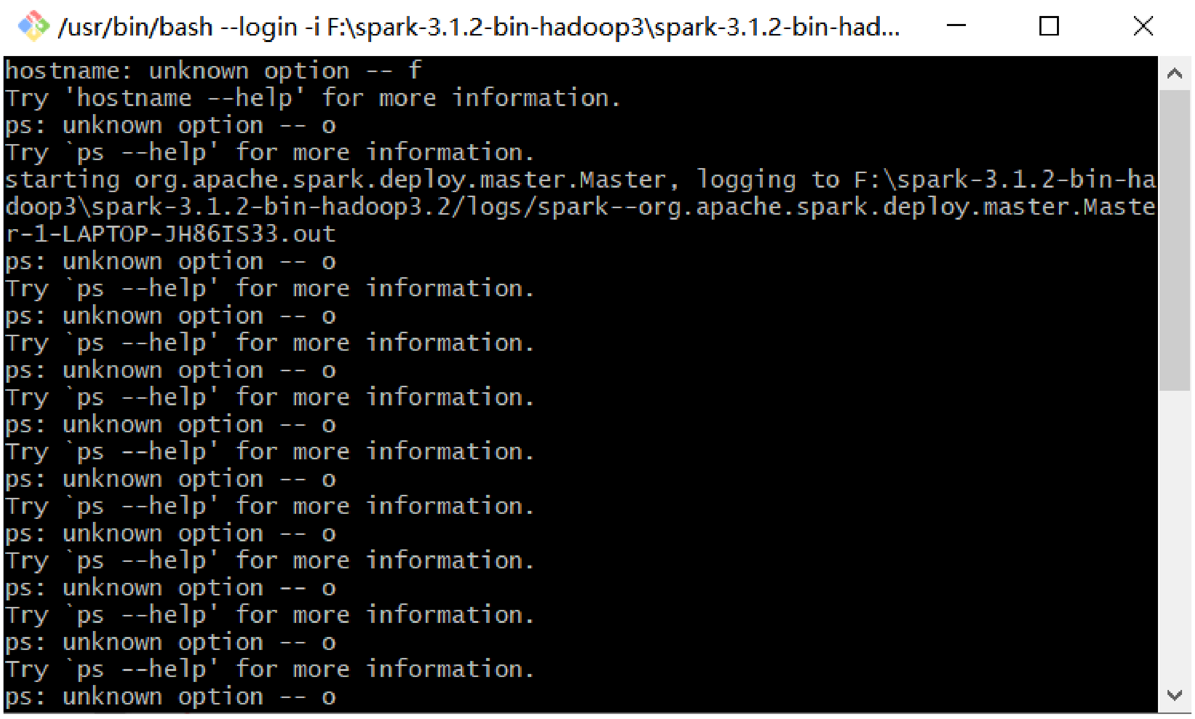
C:\Windows\System32\cmd.exe

```
F:\spark\spark-3.2.0-bin-hadoop3.2\bin>.\spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/12/12 17:42:09 ERROR SparkContext: Error initializing SparkContext.
java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:422)
        at org.apache.spark.executor.Executor.addReplClassLoaderIfNeeded(Executor.scala:909)
        at org.apache.spark.executor.Executor.<init>(Executor.scala:160)
        at org.apache.spark.scheduler.local.LocalEndpoint.<init>(LocalSchedulerBackend.scala:64)
        at org.apache.spark.scheduler.local.LocalSchedulerBackend.start(LocalSchedulerBackend.scala:132)
        at org.apache.spark.scheduler.TaskSchedulerImpl.start(TaskSchedulerImpl.scala:220)
        at org.apache.spark.SparkContext.<init>(SparkContext.scala:581)
        at org.apache.spark.SparkContext$.getOrCreate(SparkContext.scala:2690)
        at org.apache.spark.sql.SparkSession$Builder.$anonfun$getOrCreate$2(SparkSession.scala:949)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.SparkSession$Builder.getOrCreate(SparkSession.scala:943)
        at org.apache.spark.repl.Main$.createSparkSession(Main.scala:106)
        at $line3.$read$$iw$$iw.<init>(<console>:15)
        at $line3.$read$$iw.<init>(<console>:42)
        at $line3.$read.<init>(<console>:44)
        at $line3.$read$.<init>(<console>:48)
        at $line3.$read$.<clinit>(<console>)
        at $line3.$eval$.$print$lzycompute(<console>:7)
        at $line3.$eval$.$print(<console>:6)
        at $line3.$eval.$print(<console>)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

上网查询后，改用spark3.1.2版本，不再报该错。

②通过starat-all.sh启动spark报错

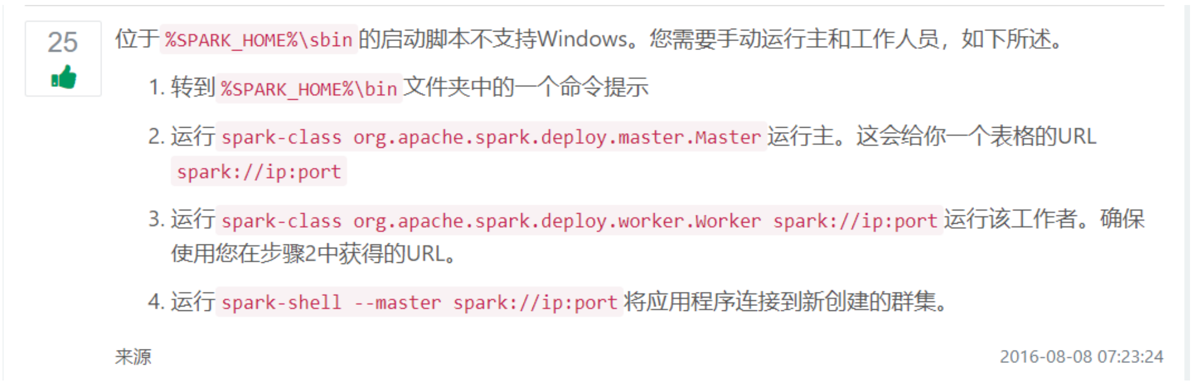
windows下通过start-all.sh启动spark报错：



/usr/bin/bash --login -i F:\spark-3.1.2-bin-hadoop3\spark-3.1.2-bin-had...

```
hostname: unknown option -- f
Try 'hostname --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
starting org.apache.spark.deploy.master.Master, logging to F:\spark-3.1.2-bin-ha
doop3\spark-3.1.2-bin-hadoop3.2/logs/spark--org.apache.spark.deploy.master.Maste
r-1-LAPTOP-JH86IS33.out
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
Try `ps --help' for more information.
ps: unknown option -- o
```

上网查找后发现是由于windows下不支持通过该方式启动，需要改成如下方式：



25

位于`%SPARK_HOME%\sbin`的启动脚本不支持Windows。您需要手动运行主和工作人员，如下所述。

1. 转到`%SPARK_HOME%\bin`文件夹中的一个命令提示
2. 运行`spark-class org.apache.spark.deploy.master.Master`运行主。这会给你一个表格的URL `spark://ip:port`
3. 运行`spark-class org.apache.spark.deploy.worker.Worker spark://ip:port`运行该工作者。确保使用您在步骤2中获得的URL。
4. 运行`spark-shell --master spark://ip:port`将应用程序连接到新创建的群集。

来源 2016-08-08 07:23:24

修改启动方式后即可在windows下正常启动spark。

③df.col"获取不到"列:

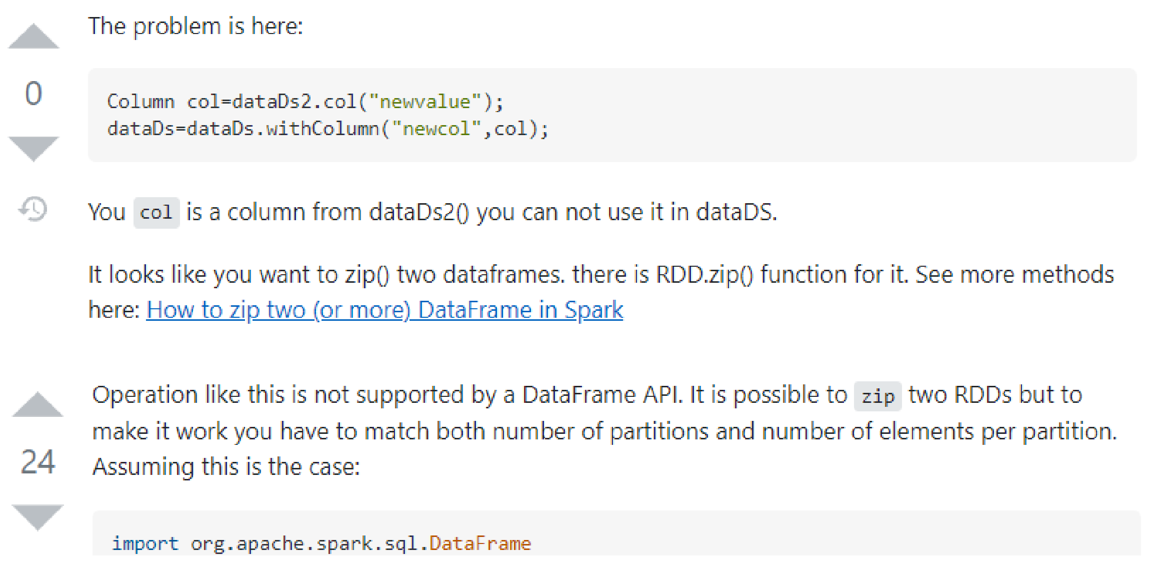
需求为：在已有表parsed_loan中通过sql查询得到一个新的dataframe：result，result只有一列，为diff。需要在另一个dataframe：indexed中通过withColumn方法添加一个新列，新列的内容和diff一致。

由于withColumn方法可以在第二参数传入col类型的值，故我第一反应是直接通过result.col("diff")获取result这个dataframe的col，并且作为withColumn的参数。但是却报错说找不到diff这个列，莫名其妙。

```
val result = spark.sql( sqlText = "SELECT datediff(issue_date,'2007-07-01') AS diff FROM parsed_loan")
indexed.withColumn( colName = "issue_date_diff", result.col( colName = "diff"))
indexed.show()
```

```
Exception in thread "main" org.apache.spark.sql.AnalysisException: Resolved attribute(s) diff#429 missing from total_loan#2,year_of_loan#3,interest#4,monthly_payment#5,work_typ
!Project [total_loan#2, year_of_loan#3, interest#4, monthly_payment#5, work_type#8, employer_type#9, industry#10, work_year#11, house_exist#12, house_loan_status#13, censor_sta
+- Project [total_loan#2, year_of_loan#3, interest#4, monthly_payment#5, work_type#8, employer_type#9, industry#10, work_year#11, house_exist#12, house_loan_status#13, censor_s
   +- Project [loan_id#0, user_id#1, total_loan#2, year_of_loan#3, interest#4, monthly_payment#5, class#6, sub_class#7, work_type#8, employer_type#9, industry#10, work_year#11,
      +- Relation[loan_id#0,user_id#1,total_loan#2,year_of_loan#3,interest#4,monthly_payment#5,class#6,sub_class#7,work_type#8,employer_type#9,industry#10,work_year#11,house_ex
```

尝试了很多办法都没有解决这个问题，最后终于在网上找到了问题所在：



0

The problem is here:

```
Column col=dataDs2.col("newvalue");
dataDs=dataDs.withColumn("newcol",col);
```

You `col` is a column from dataDs2() you can not use it in dataDS.

It looks like you want to zip() two dataframes. there is RDD.zip() function for it. See more methods here: How to zip two (or more) DataFrame in Spark

24

Operation like this is not supported by a DataFrame API. It is possible to `zip` two RDDs but to make it work you have to match both number of partitions and number of elements per partition. Assuming this is the case:

```
import org.apache.spark.sql.DataFrame
```

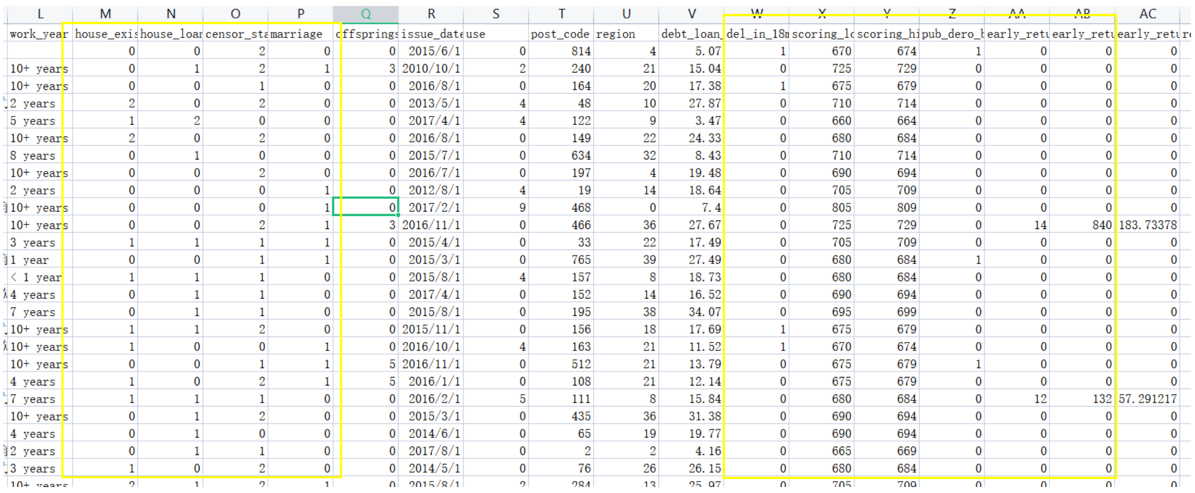
大意是不能在一个dataframe之外的地方用这个dataframe的列，因此无法将某一个dataframe的列单独作为另一个dataframe的withColumn方法的参数。

解决方法：直接在withColumn的参数中生成列，而非取其他已有dataframe的列即可：

```
val indexed2 = indexed.withColumn( colName = "issue_date_diff", datediff(indexed("issue_date"), lit( literal = "2007-07-01")))
```

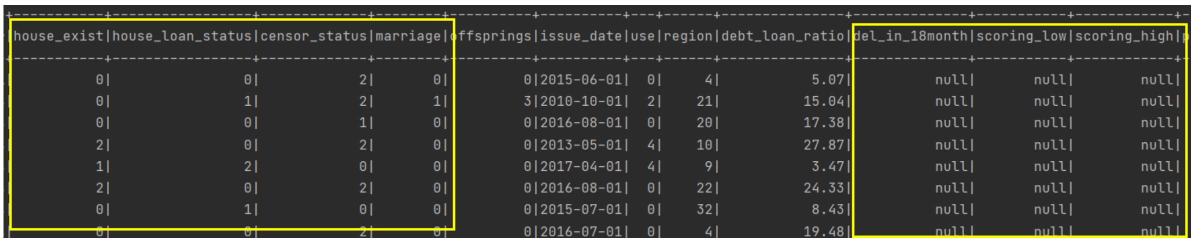
④原始数据中的整型数据读入失败

在原始csv中显示house_exist、del_in_18month等字段都为Int型字段：



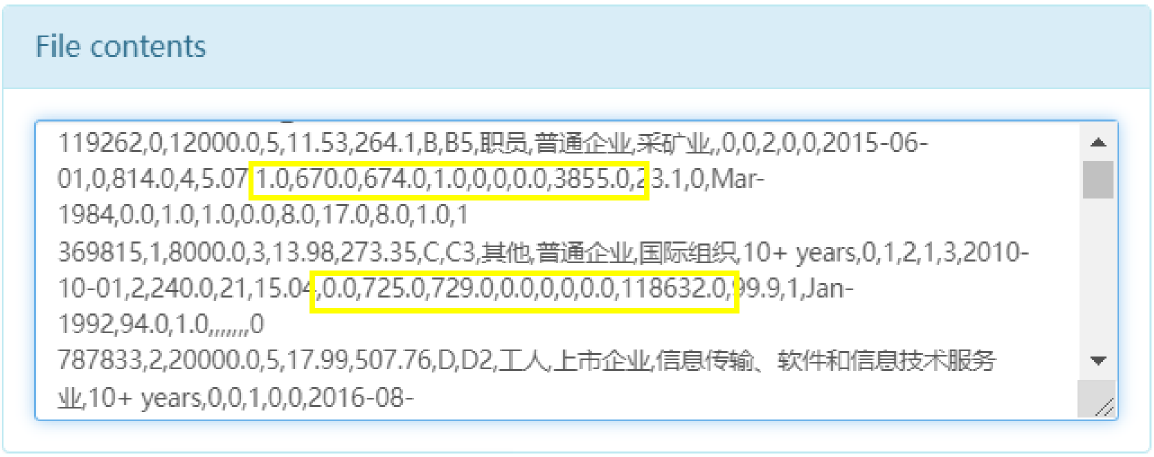
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
work_year	house_exis	house_loar	censor_sta	marriage	cffsprings	issue_date	use	post_code	region	debt_loan_	del_in_18m	scoring_lc	scoring_hi	pub_dero_t	early_retu	early_retu	early_retu
	0	0	2	0	0	2015/6/1	0	814	4	5.07	1	670	674	1	0	0	0
10+ years	0	1	2	1	3	2010/10/1	2	240	21	15.04	0	725	729	0	0	0	0
10+ years	0	0	1	0	0	2016/8/1	0	164	20	17.38	1	675	679	0	0	0	0
2 years	2	0	2	0	0	2013/5/1	4	48	10	27.87	0	710	714	0	0	0	0
5 years	1	2	0	0	0	2017/4/1	4	122	9	3.47	0	660	664	0	0	0	0
10+ years	2	0	2	0	0	2016/8/1	0	149	22	24.33	0	680	684	0	0	0	0
8 years	0	1	0	0	0	2015/7/1	0	634	32	8.43	0	710	714	0	0	0	0
10+ years	0	0	2	0	0	2016/7/1	0	197	4	19.48	0	690	694	0	0	0	0
2 years	0	0	0	1	0	2012/8/1	4	19	14	18.64	0	705	709	0	0	0	0
10+ years	0	0	0	1	0	2017/2/1	9	468	0	7.4	0	805	809	0	0	0	0
10+ years	0	0	2	1	3	2016/11/1	0	466	36	27.67	0	725	729	0	14	840	183.73378
3 years	1	1	1	1	0	2015/4/1	0	33	22	17.49	0	705	709	0	0	0	0
1 year	0	0	1	1	0	2015/3/1	0	765	39	27.49	0	680	684	1	0	0	0
< 1 year	1	1	1	0	0	2015/8/1	4	157	8	18.73	0	680	684	0	0	0	0
4 years	0	1	1	0	0	2017/4/1	0	152	14	16.52	0	690	694	0	0	0	0
7 years	0	1	1	0	0	2015/8/1	0	195	38	34.07	0	695	699	0	0	0	0
10+ years	1	1	2	0	0	2015/11/1	0	156	18	17.69	1	675	679	0	0	0	0
10+ years	1	0	0	1	0	2016/10/1	4	163	21	11.52	1	670	674	0	0	0	0
10+ years	0	0	1	1	5	2016/11/1	0	512	21	13.79	0	675	679	1	0	0	0
4 years	1	0	2	1	5	2016/1/1	0	108	21	12.14	0	675	679	0	0	0	0
7 years	1	1	1	0	0	2016/2/1	5	111	8	15.84	0	680	684	0	12	132	57.291217
10+ years	0	1	2	0	0	2015/3/1	0	435	36	31.38	0	690	694	0	0	0	0
4 years	0	1	0	0	0	2014/6/1	0	65	19	19.77	0	690	694	0	0	0	0
2 years	0	1	1	0	0	2017/8/1	0	2	2	4.16	0	665	669	0	0	0	0
3 years	1	0	2	0	0	2014/5/1	0	76	26	26.15	0	680	684	0	0	0	0
10+ years	2	1	2	1	0	2015/8/1	2	284	13	25.97	0	705	709	0	0	0	0

但是用sparkSQL设置读入类型为IntegerType时，发现有的字段能成功读入，但有的会读入失败显示null：



```
+-----------+-----------------+-------------+--------+----------+----------+---+------+---------------+--------------+-----------+------------+-
|house_exist|house_loan_status|censor_status|marriage|offsprings|issue_date|use|region|debt_loan_ratio|del_in_18month|scoring_low|scoring_high|p
+-----------+-----------------+-------------+--------+----------+----------+---+------+---------------+--------------+-----------+------------+-
          0|                0|            2|       0|         0|2015-06-01|  0|     4|           5.07|          null|       null|        null|
          0|                1|            2|       1|         3|2010-10-01|  2|    21|          15.04|          null|       null|        null|
          0|                0|            1|       0|         0|2016-08-01|  0|    20|          17.38|          null|       null|        null|
          2|                0|            2|       0|         0|2013-05-01|  4|    10|          27.87|          null|       null|        null|
          1|                2|            0|       0|         0|2017-04-01|  4|     9|           3.47|          null|       null|        null|
          2|                0|            2|       0|         0|2016-08-01|  0|    22|          24.33|          null|       null|        null|
          0|                1|            0|       0|         0|2015-07-01|  0|    32|           8.43|          null|       null|        null|
          0|                0|            2|       0|         0|2016-07-01|  0|     4|          19.48|          null|       null|        null|
```

后来经过排查发现，虽然csv中有的字段显示的是不含小数的整数，但是将csv保存到hdfs中，在hdfs中查看时则显示的是小数：



File contents

```
119262,0,12000.0,5,11.53,264.1,B,B5,职员,普通企业,采矿业,,0,0,2,0,0,2015-06-
01,0,814.0,4,5.07,1.0,670.0,674.0,1.0,0,0,0.0,3855.0,23.1,0,Mar-
1984,0.0,1.0,1.0,0.0,8.0,17.0,8.0,1.0,1
369815,1,8000.0,3,13.98,273.35,C,C3,其他,普通企业,国际组织,10+ years,0,1,2,1,3,2010-
10-01,2,240.0,21,15.04,0.0,725.0,729.0,0,0,0,0,0.0,118632.0,99.9,1,Jan-
1992,94.0,1.0,,,,,,,0
787833,2,20000.0,5,17.99,507.76,D,D2,工人,上市企业,信息传输、软件和信息技术服务
业,10+ years,0,0,1,0,0,2016-08-
```

应该是当时csv设计制作时将该字段设置成了小数类型，虽然实际存储的都是整数，但用整数类型去读则会失败，需要用Double或者Float字段读入。