目录

1. 实验一： 金融数据清洗

1.1. 实验目的

完成本实验，您应该能够：

- 了解数据清洗的一般流程
- 掌握使用 Scala 语言编写 Spark 程序进行数据清洗的方法

1.2. 实验要求

- 熟悉 Scala 基本语法
- 熟悉 Spark Core 常用 RDD 算子用法

1.3. 实验环境

本实验所需之主要资源环境如表 1-1 所示。

表 1-1 资源环境

服务器集群	3 个以上节点，节点间网络互通，各节点最低配置：双核 CPU、8GB 内存、100G 硬盘
运行环境	CentOS 7.4
服务和组件	HDFS、YARN、MapReduce 等，其他服务根据实验需求安装。各软件版本如下： Hadoop 2.7.1 Scala 2.11.8 Spark 2.0.0 MySQL 5.7.18 JDK 1.8.231

1.4. 实验视图

数据清洗流程如图 1-1

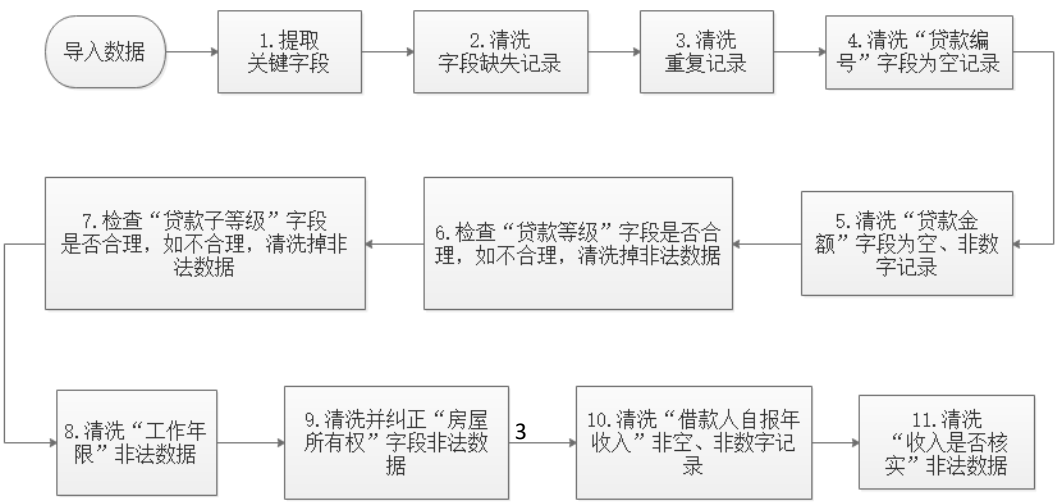


图 1-1 部署流程

1.5. 实验过程

1.5.1. 实验任务一：实验环境准备

1.5.1.1. 步骤一：数据准备

源数据包含 74 个字段，本实验需要从中选择 8 个关键字段，对应的字段索引如表 1-2 所示。

表 1-2 关键字段

序号	字段名	字段索引	备注
1	id	0	借款编号
2	loan_amnt	2	借款人申请的贷款金额
3	grade	7	借款等级
4	sub_grade	8	借款子等级
5	emp_length	10	借款人工作时长
6	home_ownership	11	房屋所有权状态，为 RENT，OWN，MORTAGE,OTHER
7	annual_inc	12	借款自报的年收入
8	verification_status	13	借款人收入是否核实

数据清洗的第一步就需要将关键字段提取出来。

每个字段的详细说明如下：

1. “贷款等级”：分为【A、B、C、D、E、F、G】共 6 个级别，级别越高，贷款利率越高。其中“A”级最低，平均贷款年利率在 7%左右；“G”级最高，平均贷款年利率在 22%左右。
2. “借款子等级”：每个级别又分为 5 个子等级，如“A”级又分为【A1、A2、A3、A4、A5】共 5 个子等级，利率也从“A1”至“A5”逐级提高；
3. “工作年限”：为借款人在平台中提供的工作时长，如“4 years”；
4. “借款人自报的年收入”：为借款人提交借款申请时所提供的年收入金额；
5. “借款人收入是否核实”：为平台是否对借款人所提供的年收入情况进行核实，字段值为【Verified（已核实）、Source Verified（来源已核实）、Not Verified（未核实）】中的一种。本项目需要对借款人收入进行分析，因此需要剔除“未核实”收入的数据，避免结果的偏颇；
6. “房屋所有权状态”：体现借款人的房屋状况，字段值为【RENT（租住）、OWN（自主住房）、MORTGAGE（按揭）、OTHER（其他）】中的一种。如果出现其他值，则为非法字段。

1.5.2. 实验任务二：数据清洗

数据清洗的流程图如下：

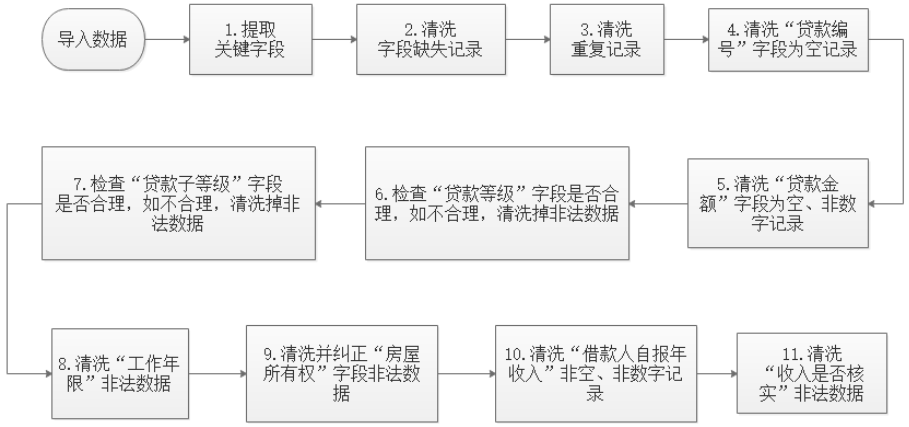


图 1-2 数据清洗流程图

数据源位于 HDFS 集群的/finance/input 路径下，文件名为 loan.csv，如图所示：

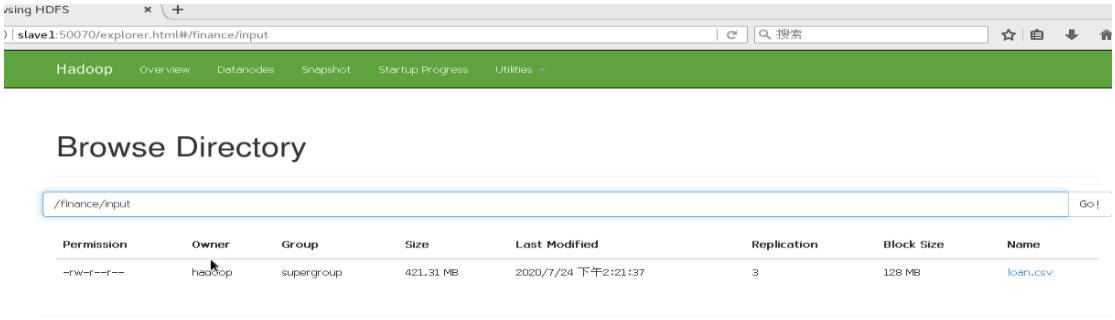


图 1-3 数据源

1.5.2.1. 步骤一：提取关键字段

Scala 类：Step1.scala，读取 HDFS 文件转换为 RDD，进行字段提取，代码如下：

```
import org.apache.spark.{SparkConf, SparkContext}
import scala.util.{Success, Try}
object Step1 {
    def verify(str: String, dtype: String):Boolean = {
    var c:Try[Any] = null
    if("double".equals(dtype)) {
       c = scala.util.Try(str.toDouble)
    } else if("int".equals(dtype)) {
       c = scala.util.Try(str.toInt)
    }
    val result = c match {
       case Success(_) => true;
       case _ =>    false;
    }
    result
  }
  def main(args: Array[String]): Unit = {
  System.setProperty("HADOOP_USER_NAME", "hadoop")
  val sparkConf = new SparkConf().setAppName("Step1").setMaster("local[*]")
```

```
    val sc = new SparkContext(sparkConf)
    val data = sc.textFile("hdfs://master:8020/finance/input/loan.csv")
    val input = data.map(line => line.split(",")(0) + "," + line.split(",")(2) + ","
      + line.split(",")(8) + "," + line.split(",")(9) + "," + line.split(",")(11) +
      "," + line.split(",")(12) + "," + line.split(",")(13)+","+line.split(",")(14))
    println("共有数据"+input.count()+"条")
    sc.stop()
      }
  }
```

运行结果如下：

共有数据 887379 条

1.5.2.2. 步骤二：清洗字段缺失记录

源数据中的 8 个关键字段可能有缺失，因此需要将少于 8 个字段的数据清洗掉，在类中添加以下代码进行缺失字段的清洗：

```
//查看字段是否缺失，剩余字段 876947
val clean = input.filter(x => x.split(",").length == 8)
println("剩余数据"+clean.count()+"条")
println("清洗掉"+(input.count()-clean.count())+"条")
```

运行结果如下：

剩余数据 887379 条

清洗掉 0 条

1.5.2.3. 步骤三：清洗重复记录

源数据有可能存在重复数据，需要将重复记录清洗掉。在类中添加以下代码：

```
//清洗重复值——无重复数据
val rep = clean.distinct();
println("清洗掉"+(input.count()-rep.count())+"条")
```

运行结果如下：

清洗掉 0 条

1.5.2.4. 步骤四：清洗“贷款编号”字段

“贷款编号”字段可能存在空值，因此需要将这样的记录清洗掉。在类中添加代码：

```
//清洗 id 为空的记录
val idnn = rep.filter(_.split(",")(0).length > 0)
println("清洗掉"+(rep.count()-idnn.count())+"条")
```

运行结果如下：

清洗掉 0 条

1.5.2.5. 步骤五：清洗“贷款金额”字段

“贷款金额”字段可能为空，也可能出现非数值的非法字段，这样的数据会对后续的分析工作产生影响，因此需要清洗掉。在类中添加代码：

```
//贷款金额，是否空值，是否为数字
val loan = idnn.filter(_.split(",")(1).length > 0)
.filter(x => verify(x.split(",")(1), "double"))
println("剩余"+loan.count()+"条")
```

```
println("清洗掉"+(idnn.count()-loan.count())+"条")
```

运行结果如下：

剩余 887379 条

清洗掉 0 条

1.5.2.6. 步骤六：清洗“贷款等级”字段

检查“贷款等级”字段是否存在非法字段，可以对该字段值进行聚合统计进行判断，添加如下代码：

```
//2 贷款等级，聚合查看分布
val grade = loan.map(x => (x.split(",")(2), 1)).reduceByKey(_ + _)
println(grade.collect().mkString(","))
```

运行结果如下：

(D,139542),(A,148202),(E,70705),(B,254535),(F,23046),(G,5489),(C,245860)

可以看出，该字段均为 A-G 内的等级值，因此不需要做进一步处理

1.5.2.7. 步骤七：清洗“贷款子等级”字段

用聚合统计的方法对“贷款子等级”字段进行检查，添加如下代码：

```
//贷款子等级，聚合查看分布
val subgrade = loan.map(x => (x.split(",")(3), 1)).reduceByKey(_ + _)
println(subgrade.collect().mkString(","))
```

运行结果如下：

(A5,44816),(E1,18268),(A1,22913),(D4,25558),(C3,50161),(G3,981),(B2,48781),(F2,5392),(E5,9575),(E2,17004),(B3,56323),(D5,21389),(D1,36238),(G4,663),(C4,48857),(F3,4433),(A2,22485),(A3,23457),(G5,576),(B4,55626),(C5,41219),(D2,29803),(F4,3409),(C1,53387),(G1,1871),(E3,14134),(D3,26554),(B5,48833),(C2,52236),(F1,7218),(A4,34531),(G2,1398),(B1,44972),(E4,11724),(F5,2594)

可以看出，该字段没有非法值，因此不需要进一步处理。

1.5.2.8. 步骤八：清洗“工作年限”字段

查看“工作年限”字段，合理字段为“3 years”等，可以将字段值中不包含“years”的数据清洗掉，添加代码如下：

```
//工作年限清洗
val workyears = loan.filter(x => x.split(",")(4).contains("year"))
println("剩余"+workyears.count()+"条")
println("清洗掉"+(loan.count()-workyears.count())+"条")
```

运行结果如下：

剩余 826402 条

清洗掉 60977 条

1.5.2.9. 步骤九：清洗“房屋所有权”字段

“房屋所有权”字段值应为“RENT（租住）”、“OWN（自住）”、“MORTGAGE（按揭）”、“OTHER（其他）”中的一种，其他值则为非法字段，需要进行清洗。添加如下代码：

```
//房屋租赁状态，为”RENT，OWN，MORTGAGE,OTHER“中的一种
  val house = workyears.filter(x => {
    val state = Array("RENT", "OWN", "MORTGAGE", "OTHER")
    var h = false
```

```
        for (i <- 0 to state.length - 1) {
          if (state(i).equals(x.split(",")(5)))
            h = true
        }
        h
      })
    val housecount=house.count()
    println("清洗掉非法字段"+(workyears.count()-housecount)+"条")
    println("剩余记录"+housecount+"条")
```

结果如下：

清洗掉非法字段 51 条

剩余数据 826351 条

1.5.2.10. 步骤十：清洗“借款人自报年收入”字段

清洗“借款人自报年收入”非空、非数字记录。添加代码如下：

```
//年收入，是否空值是否数字
val incomenn = house.filter(_.split(",")(6).length > 0)
val income = incomenn.filter(x => verify(x.split(",")(6), "double"))
println("清洗掉空值字段数据"+(house.count()-incomenn.count())+"条")
println("清洗掉非数值字段数据"+(incomenn.count()-income.count())+"条")
println("剩余数据"+income.count()+"条")
```

结果如下：

清洗掉空值字段数据 0 条

清洗掉非数值字段数据 0 条

剩余数据 826351 条

1.5.2.11. 步骤十一：清洗“收入是否核实”字段

清洗“收入是否核实”字段。将“未核实”数据清洗掉。添加如下代码：

```
//收入是否核实 Verified,Source Verified,Not Verified
val incomeFalse=income.filter(x=>x.split(",")(7).contains("Not"))
val out=income.subtract(incomeFalse)
println("清洗掉未验证收入数据"+incomeFalse.count()+"条")
println("剩余数据"+out.count()+"条")
```

结果如下：

清洗掉未验证收入数据 256729 条

剩余数据 569622 条

至此，我们已完成数据的所有清洗工作，剩余合法数据 569622 条。

1.5.2.12. 步骤十二：保存文件

将清洗后的文件保存到集群上，保存路径为/finance/cleanout，添加如下代码：

```
out.repartition(1).saveAsTextFile("hdfs://master:8020/finance/cleanout")
```

jar 包已经编译好放到/opt/software 目录下。启动 spark 程序

```
[hadoop@master ~]$ /usr/local/src/spark/bin/spark-submit --class Step1 /opt/software/Finance.jar
```

完成后在浏览器中检查文件是否已保存成功：

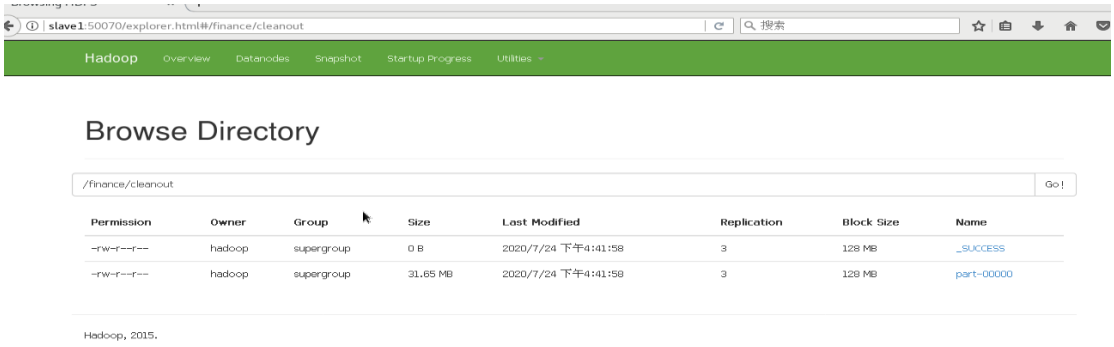


图 1-4 保存清洗文件

查看该文件的行数，指令与结果如下：

[hadoop@master ~]$ hdfs dfs -cat /finance/cleanout/part-00000 | wc -l

569622

查看该文件的前 10 行数据，指令与结果如下：

[hadoop@master ~]$ hdfs dfs -cat /finance/cleanout/part-00000 | head -10

60800386,15000.0,C,C2,5 years,RENT,52000.0,Verified

5524829,10000.0,C,C4,1 year,RENT,38000.0,Verified

64340155,15850.0,F,F1,3 years,MORTGAGE,60000.0,Source Verified

1541694,23675.0,B,B2,10+ years,OWN,54000.0,Verified

65956221,35000.0,C,C2,2 years,RENT,79000.0,Source Verified

46094626,16000.0,C,C4,5 years,MORTGAGE,58000.0,Verified

658701,22000.0,E,E5,9 years,RENT,60000.0,Verified

49653243,24000.0,D,D4,10+ years,RENT,81732.0,Verified

6795304,30000.0,G,G2,10+ years,RENT,95000.0,Verified

1553773,30000.0,F,F2,4 years,MORTGAGE,80000.0,Verified

2. 实验二 数据分析

2.1. 实验目的

完成本实验，您应该能够：

- 掌握 Spark 进行数据分析的方法

2.2. 实验要求

- 熟悉 Scala 基本语法
- 熟悉 Spark Core 常用 RDD 算子用法

2.3. 实验环境

本实验所需之主要资源环境如表 2-1 所示。

表 2-1 资源环境

服务器集群	3 个以上节点，节点间网络互通，各节点最低配置：双核 CPU、8GB 内存、100G 硬盘
运行环境	CentOS 7.4
服务和组件	HDFS、YARN、MapReduce 等，其他服务根据实验需求安装。各软件版本如下： Hadoop 2.7.1 Scala 2.11.8 Spark 2.0.0 MySQL 5.7.18 JDK 1.8.231

2.4. 实验过程

2.4.1. 实验任务一：借款金额分布

2.4.1.1. 步骤一：任务分析

本实验将借款金额分为以下区间，统计在每个区间的人数：

1. 大于 0 小于等于 1000 美元
2. 大于 1000 小于等于 5000 美元
3. 大于 5000 小于等于 10000 美元
4. 大于 10000 小于等于 20000 美元
5. 大于 20000 小于等于 50000 美元
6. 大于 50000 美元

2.4.1.2. 步骤二：代码实现

“Step2”，添加代码如下：

```
import org.apache.spark.{SparkConf, SparkContext}

object Step2 {
//借款金额分布
def main(args: Array[String]): Unit = {
  System.setProperty("HADOOP_USER_NAME", "hadoop")
  val sparkConf=new SparkConf().setAppName("Step2").setMaster("local[*]")
  val sc=new SparkContext(sparkConf)
  val data=sc.textFile("hdfs://master:8020/finance/input/loan.csv")
//贷款金额范围
val arr=Array(0,1000,5000,10000,20000,50000)
val fundedAmount=data.map(_.split(",")(2).toDouble)
      .map(x=>{
        var amount=arr.length
        for(i<-0 to arr.length-2){
          if (x > arr(i) && x <= arr(i + 1)) {
            amount=i
          }
        }
        if(amount<arr.length-1){
          (arr(amount)+"-"+arr(amount+1),1)
        }
        else{
          ("50000-更多",1)
        }
      }
      )
  .reduceByKey(_+_).sortBy(x=>x._1.split("-")(0).toDouble)
      .map(x=>x._1+","+x._2)
  fundedAmount.repartition(1).saveAsTextFile("hdfs://master:8020/finance/out/step2")
  sc.stop()
}

}
```

```
[hadoop@master ~]$ /usr/local/src/spark/bin/spark-submit --class Step2 /opt/software/Finance.jar
```

查看集群上是否已生成结果文件：

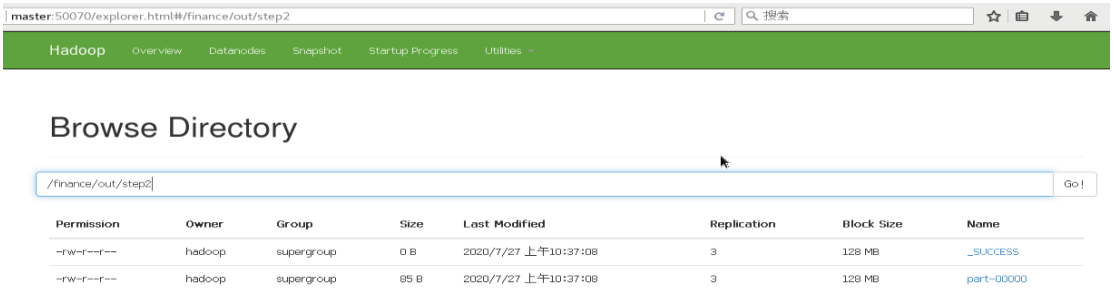


图 2-1 借款金额分布结果

查看文件内容：

```
[hadoop@master ~]$ hdfs dfs -cat /finance/out/step2/part-00000
0-1000,2623
1000-5000,100057
5000-10000,232900
10000-20000,350962
20000-50000,200837
```

2.4.2. 实验任务二：借款等级分布

2.4.2.1. 步骤一：任务分析

不同的借款等级意味着不同的借款利率，等级越高，借款利率越高。借款利率与借款金额、还款时间都有关系。

P2P 借贷平台的很大一部分利润来自于借款利率，本项目的源数据中借款等级位于 A-G 之间，每个等级又分为 5 个子等级，本实验将分析每个子等级的人数分布。

2.4.2.2. 步骤二：代码实现

Scala 类“Step3”，写入以下代码：

```
import org.apache.spark.{ SparkConf, SparkContext}

import scala.collection.mutable

object Step3 {
//借款等级分析，按照子等级统计人数
def main(args: Array[String]): Unit = {
    System.setProperty("HADOOP_USER_NAME", "hadoop")
    val sparkConf=new SparkConf().setAppName("Step3").setMaster("local[*]")
    val sc=new SparkContext(sparkConf)
    val data=sc.textFile("hdfs://master:8020/finance/cleanout/part-00000")
  //借款等级
    val grade=data.map(x=>(x.split(",")(2)+","+x.split(",")(3),1))
      .reduceByKey(_+_).map(x=>(x._1.split(",")(0),x._1.split(",")(1)+","+x._2))
    val out=grade.sortBy(x=>x._2.split(",")(0))
      .map(x=>x._1+","+x._2).repartition(1)
    out.saveAsTextFile("hdfs://master:8020/finance/out/step3")
    sc.stop()
```

```
}
}
```

[hadoop@master ~]$ /usr/local/src/spark/bin/spark-submit --class Step3 /opt/software/Finance.jar

查看结果文件：

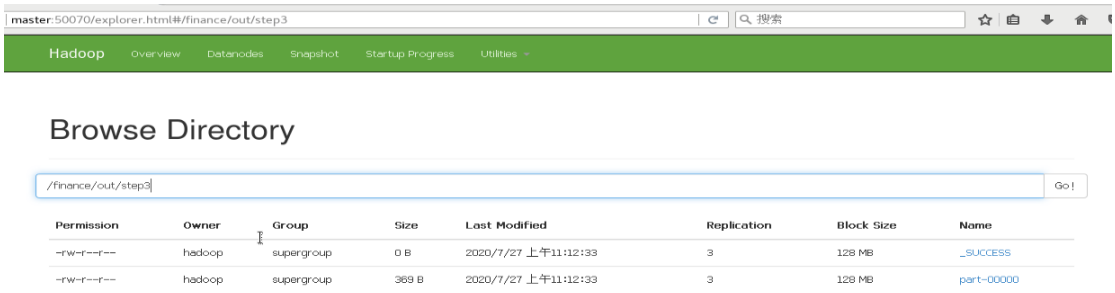


图 2-2 借款等级分布结果

查看文件前 10 行：

```
[hadoop@master ~]$ hdfs dfs -cat /finance/out/step3/part-00000 | head -10
A,A1,9654
A,A2,10206
A,A3,11892
A,A4,18350
A,A5,25503
B,B1,23498
B,B2,27320
B,B3,32698
B,B4,33683
B,B5,30564
```

2.4.3. 实验任务三：借款等级与借款金额的关联关系

2.4.3.1. 步骤一：任务分析

本实验分析各借款等级中，不同借款金额区间的人数分布。其中借款等级仅分析 A-G 级即可，不需分析子等级的人数分布。

2.4.3.2. 步骤二：代码实现

Scala 类“Step4”，代码如下：

```
import org.apache.spark.{SparkConf, SparkContext}

object Step4 {
  //借款等级与借款金额的关联关系
  //46804490,35000.0,C,C4,< 1 year,RENT,110000.0
  //5696129,8450.0,B,B4,7 years,MORTGAGE,62000.0
  //26078378,24000.0,B,B1,10+ years,MORTGAGE,60000.0
  def main(args: Array[String]): Unit = {
    System.setProperty("HADOOP_USER_NAME", "hadoop")
```

```
        val sparkConf=new SparkConf().setAppName("Step4").setMaster("local[*]")
        val sc=new SparkContext(sparkConf)
        val data=sc.textFile("hdfs://master:8020/finance/cleanout/part-00000")
        //贷款金额范围
        val arr=Array(0,1000,5000,10000,20000,50000)
        val
    fundedAmount=data.map(x=>(x.split(",")(2)+","+x.split(",")(1),x.split(",")(1).toDouble))
            .map(x=>{
              var amount=arr.length
              for(i<-0 to arr.length-2){
                if (x._2 > arr(i) && x._2 <= arr(i + 1)) {
                  amount=i
                }
              }
              if(amount<arr.length-1){
                (x._1.split(",")(0)+","+ arr(amount)+"-"+arr(amount+1),1)
              }
              else{
                (x._1.split(",")(0)+","+"50000-更多",1)
              }
            }
            )
            .reduceByKey(_+_).sortBy(x=>x._1.split(",")(1).split("-")(0).toDouble)
            .map(x=>(x._1.split(",")(0),x._1.split(",")(1)+","+x._2))

    fundedAmount.map(x=>x._1+","+x._2).repartition(1).saveAsTextFile("hdfs://master:8020
    /finance/out/step4")
        sc.stop()
        //      println(fundedAmount.take(2).mkString("---"))
      }
    }
```

[hadoop@master ~]$ /usr/local/src/spark/bin/spark-submit --class Step4 /opt/software/Finance.jar

检查结果文件：

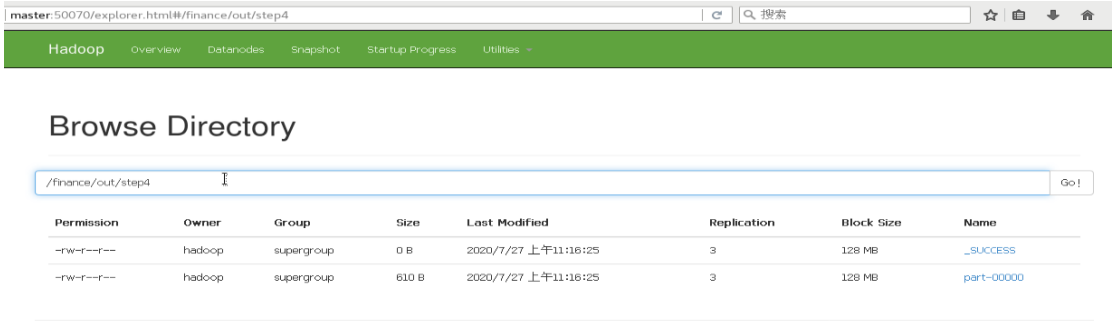


图 2-3 借款等级与借款金额的关联关系结果

查看文件的前 10 行：

```
[hadoop@master ~]$ hdfs dfs -cat /finance/out/step4/part-00000 | head -10
D,0-1000,221
F,0-1000,19
B,0-1000,211
E,0-1000,77
A,0-1000,57
C,0-1000,417
G,0-1000,3
G,1000-5000,134
A,1000-5000,4983
E,1000-5000,2819
```

2.4.4. 实验任务四：借款金额与工作年限、年收入的关联关系

2.4.4.1. 步骤一：任务分析

客户是否能按时还款，与其是否有固定收入、收入如何有密切关系，因此分析客户的工作状态、年收入非常重要，对于借款金额较大、收入较低的客户，其风险相对较高，需要重点关注。本任务的目的在于分析客户的行为，帮助借贷平台最大限度规避风险。

源数据中客户的工作年限以年为单位，此处我们沿用源数据中的工作年限界定方式；将年收入分为如下几个区间：

1. 大于 0 小于等于 1 万美元；
2. 大于 1 万小于等于 5 万美元；
3. 大于 5 万小于等于 10 万美元；
4. 大于 10 万小于等于 20 万美元；
5. 大于 20 万美元。

2.4.4.2. 步骤二：代码实现

Scala 类“Step5”，添加如下代码：

```
import org.apache.spark.{SparkConf, SparkContext}

object Step5 {
  //借款金额与工作年限、年收入的关联关系
  //46804490,35000.0,C,C4,< 1 year,RENT,110000.0
  //5696129,8450.0,B,B4,7 years,MORTGAGE,62000.0
  //26078378,24000.0,B,B1,10+ years,MORTGAGE,60000.0
  def main(args: Array[String]): Unit = {
    System.setProperty("HADOOP_USER_NAME", "hadoop")
    val sparkConf=new SparkConf().setAppName("Step5").setMaster("local[*]")
    val sc=new SparkContext(sparkConf)
    val data=sc.textFile("hdfs://master:8020/finance/cleanout/part-00000")
```

```
//贷款金额范围
val arr=Array(0,1000,5000,10000,20000,50000)
val out1=data.map(x=>(x.split(",")(4)+","+x.split(",")(1),x.split(",")(1).toDouble))
  .map(x=>{
    var amount=arr.length
    for(i<-0 to arr.length-2){
      if (x._2 > arr(i) && x._2 <= arr(i + 1)) {
        amount=i
      }
    }
    if(amount<arr.length-1){
      (x._1.split(",")(0)+","+ arr(amount)+"-"+arr(amount+1),1)
    }
    else{
      (x._1.split(",")(0)+","+"50000-更多",1)
    }
  }
  )
  .reduceByKey(_+_).sortBy(x=>x._1.split(",")(1).split("-")(0).toDouble)
  .map(x=>x._1+","+x._2)
out1.repartition(1).saveAsTextFile("hdfs://master:8020/finance/out/step5_1")
//年收入金额范围
val income=Array(0,10000,50000,100000,200000)
val out2=data.map(x=>(x.split(",")(6)+","+x.split(",")(1),x.split(",")(1).toDouble))
  .map(x=>{
    var amount=arr.length
    for(i<-0 to arr.length-2){
      if (x._2 > arr(i) && x._2 <= arr(i + 1)) {
        amount=i
      }
    }
    if(amount<arr.length-1){
      (x._1.split(",")(0)+","+ arr(amount)+"-"+arr(amount+1),1)
    }
    else{
      (x._1.split(",")(0)+","+"50000-更多",1)
    }
  }
  ).map(x=>{
  val y=x._1.split(",")(0).toDouble
  var index=income.length
  for(j<-0 to income.length-2){
    if(y>income(j)&&y<=income(j+1)){
      index=j
```

```
            }
          }
          if(index<income.length-1){
            (income(index)+"--"+income(index+1)+","+x._1.split(",")(1),x._2)
          }else{
            ("150000-更多收入"+","+x._1.split(",")(1),x._2)
          }
        }) .reduceByKey(_+_)
        out2.repartition(1).map(x=>x._1+","+x._2)
          .sortBy(x=>x.split(",")(0).split("-")(0).toDouble)
          .sortBy(x=>x.split(",")(1).split("-")(1).toDouble)
          .saveAsTextFile("hdfs://master:8020/finance/out/step5_2")
        sc.stop()
      }
    }
```

[hadoop@master ~]$ /usr/local/src/spark/bin/spark-submit --class Step5 /opt/software/Finance.jar

借款金额与工作年限关系的结果文件保存在集群地址/finance/out/step5-1 中，查看文件前 10 行结果如下：

```
[hadoop@master ~]$ hdfs dfs -cat /finance/out/step5_1/part-00000 | head -10
6 years,0-1000,42
2 years,0-1000,80
10+ years,0-1000,338
4 years,0-1000,71
1 year,0-1000,73
8 years,0-1000,47
< 1 year,0-1000,113
7 years,0-1000,54
5 years,0-1000,75
9 years,0-1000,33
```

借款金额与年收入关系的结果文件保存在集群地址/finance/out/step5_2 中，查看文件前 10 行结果如下：

```
[hadoop@master ~]$ hdfs dfs -cat /finance/out/step5_2/part-00000 | head -10
0--10000,0-1000,11
10000--50000,0-1000,573
50000--100000,0-1000,357
100000--200000,0-1000,60
150000-更多收入,0-1000,4
0--10000,1000-5000,151
10000--50000,1000-5000,23747
50000--100000,1000-5000,17168
100000--200000,1000-5000,3154
150000-更多收入,1000-5000,273
```

2.4.5. 实验任务五：借款金额与房屋所有状态的关联关系

2.4.5.1. 步骤一：任务分析

房屋是个人财产的最好体现，房屋所有权状态体现着借款人的财务状况，该状态分为为【RENT（租住）、OWN（自主住房）、MORTGAGE（按揭）、OTHER（其他）】中的其中一种，可以通过统计每种状态下的各区间人数分析两者的关联情况。

2.4.5.2. 步骤二：代码实现

Scala 类“Step6”，添加代码如下：

```
import org.apache.spark.{SparkConf, SparkContext}

object Step6 {
//分析借款人贷款金额与房屋所有状态的关联关系
def main(args: Array[String]): Unit = {
  System.setProperty("HADOOP_USER_NAME", "hadoop")
  val sparkConf=new SparkConf().setAppName("Step6").setMaster("local[*]")
  val sc=new SparkContext(sparkConf)
  val data=sc.textFile("hdfs://master:8020/finance/cleanout/part-00000")
  //贷款金额范围
  val arr=Array(0,1000,5000,10000,20000,50000)
  val out=data.map(x=>(x.split(",")(5)+","+x.split(",")(1),x.split(",")(1).toDouble))
    .map(x=>{
      var amount=arr.length
      for(i<-0 to arr.length-2){
        if (x._2 > arr(i) && x._2 <= arr(i + 1)) {
          amount=i
        }
      }
      if(amount<arr.length-1){
        (x._1.split(",")(0)+","+ arr(amount)+"-"+arr(amount+1),1)
      }
      else{
        (x._1.split(",")(0)+","+"50000-更多",1)
      }
    }
    )
    .reduceByKey(_+_).sortBy(x=>x._1.split(",")(1).split("-")(0).toDouble)
    .map(x=>x._1+","+x._2).repartition(1)
  out.saveAsTextFile("hdfs://master:8020/finance/out/step6")
  sc.stop()
}
}
```

```
[hadoop@master ~]$ /usr/local/src/spark/bin/spark-submit --class Step6
```

/opt/software/Finance.jar

查看结果文件的前 10 行：

```
[hadoop@master ~]$ hdfs dfs -cat /finance/out/step6/part-00000 | head -10
MORTGAGE,0-1000,336
OWN,0-1000,116
RENT,0-1000,553
RENT,1000-5000,23037
OTHER,1000-5000,12
MORTGAGE,1000-5000,16891
OWN,1000-5000,4553
RENT,5000-10000,63982
OTHER,5000-10000,22
MORTGAGE,5000-10000,50046
```

3. 实验三 数据可视化

3.1. 实验目的

完成本实验，您应该能够：

- 掌握 Flask 导入数据的方法
- 掌握 Echarts 创建图表的方法

3.2. 实验要求

- 熟悉 Python 基本语法
- 熟悉 Mysql 基本查询语句

3.3. 实验环境

本实验所需之主要资源环境如表 1-1 所示。

表 3-1 资源环境

服务器集群	3 个以上节点，节点间网络互通，各节点最低配置：双核 CPU、8GB 内存、100G 硬盘
运行环境	CentOS 7.4
服务和组件	HDFS、YARN、MapReduce 等，其他服务根据实验需求安装。各软件版本如下： Hadoop 2.7.1 Python 3.7 (使用到 flask、pymysql、numpy 包) MySQL 5.7.18 (安装在 master 节点上) JDK 1.8.231

3.4. 实验视图

数据可视化流程如图 1-1 所示：

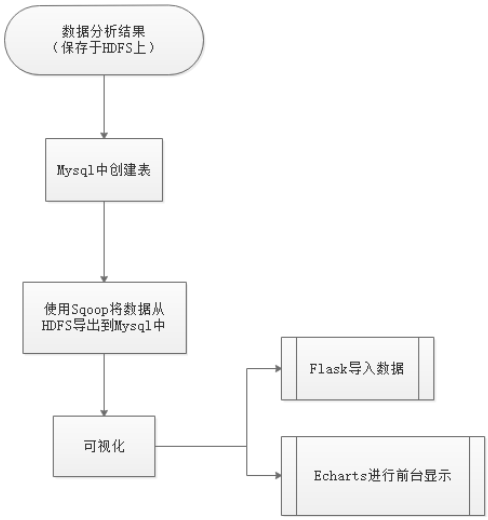


图 3-1 可视化流程图

3.5. 实验过程

3.5.1. 实验任务一：Mysql 中创建表

首先，需要在 Mysql 中创建数据库 finance，在该库中创建表，用于存储分析结果。根据分析结果，需要创建以下几张表：

表 3-2 Mysql 中的表结构

序号	表名	字段	数据类型	备注
1	借款金额分布表：loan_amount	zone	varchar(50)	借款金额区间
		num	int	人数
2	借款等级分布表：loan_grade	grade	varchar(5)	借款等级
		subgrade	varchar(5)	借款子等级
		num	int	人数
3	借款等级与借款金额关系表：grade_amount	grade	varchar(5)	借款等级
		amount	varchar(50)	借款金额区间
		num	int	人数
4	借款金额与工作年限关系表：amount_wkyear	wkyears	varchar(50)	工作年限区间
		amount	varchar(50)	借款金额区间
		num	int	人数
5	借款金额与年收入关系表：amount_salary	salary	varchar(50)	年收入区间
		amount	varchar(50)	借款金额区间
		num	int	人数
6	借款金额与房屋所有权状态关系表：amount_house	house	varchar(50)	房屋状态
		amount	varchar(50)	借款金额区间
		num	int	人数

具体指令如下：

[hadoop@master ~]$ mysql -uroot -pPassword123$

mysql > CREATE DATABASE finance;

mysql > USE finance;

mysql > CREATE TABLE loan_amount(zone VARCHAR(50),num INT);

mysql > CREATE TABLE loan_grade(grade VARCHAR(5),subgrade VARCHAR(5),num INT);

mysql > CREATE TABLE grade_amount (grade VARCHAR(5),amount VARCHAR(50),num INT);

mysql > CREATE TABLE amount_wkyear(wkyears VARCHAR(50),amount VARCHAR(50),num INT);

mysql > CREATE TABLE amount_salary (salary VARCHAR(50),amount VARCHAR(50),num INT);

mysql > CREATE TABLE amount_house (house VARCHAR(50),amount VARCHAR(50),num INT);

在 Mysql 中检查如下：

```
mysql> show tables;
+------------------+
| Tables_in_finance |
+------------------+
| amount_house      |
| amount_salary     |
| amount_wkyear     |
| grade_amount      |
| loan_amount       |
| loan_grade        |
+------------------+
6 rows in set (0.00 sec)
```

图 3-2 Mysql 中的库和表示意图

3.5.2. 实验任务二：Sqoop 导出数据导 Mysql

分析结果均已保存在集群上：

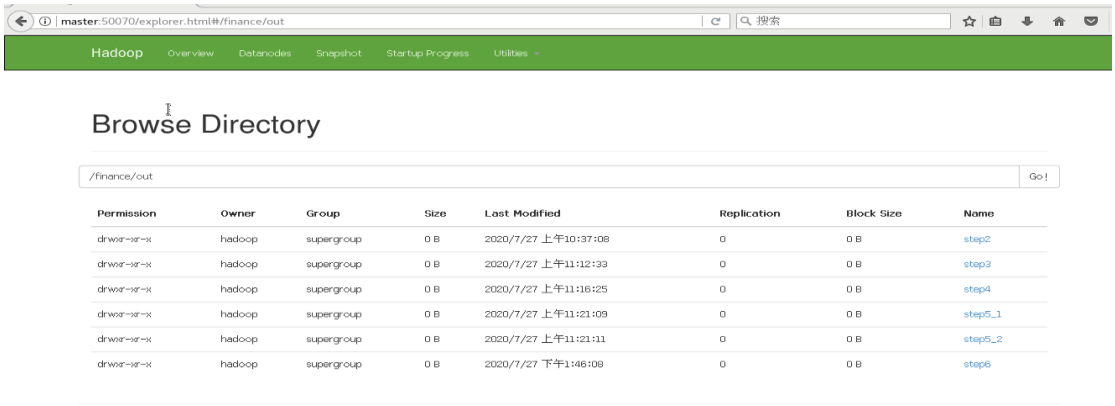


图 3-3 数据分析结果文件

需要使用 Sqoop 将数据导入到 Mysql 表中。在 master 节点的/home/hadoop 文件夹下，创建文件夹 data，再创建文件 finance_export.conf，用于保存 Sqoop 导出配置。

[hadoop@master ~]$ mkdir data && cd data

[hadoop@master data]$ vim finance_export.conf

首先导出“借款金额分布表”，也即从集群 step2/ part-00000 文件中导出到 Mysql 中的 loan_amount 表中。需要注意的是，源文件中各个字段间用英文逗号隔开。因此配置文件可以编写如下：

export

--connect
jdbc:mysql://master:3306/finance?useUnicode=true&characterEncoding=UTF-8
--username
root
--password
Password123$
--table
loan_amount
--export-dir
/finance/out/step2/part-00000
--input-null-string
\\N
--input-null-non-string
\\N
--fields-terminated-by
,

编辑完毕后退出。执行如下指令进行导出：

[hadoop@master data]$ sqoop --options-file finance_export.conf

导出完成后，查询表格数据，结果如下：

```
mysql> select * from loan_amount;
+-------------+--------+
| zone        | num    |
+-------------+--------+
| 5000-10000  | 232900 |
| 10000-20000 | 350962 |
| 20000-50000 | 200837 |
| 0-1000      |   2623 |
| 1000-5000   | 100057 |
+-------------+--------+
5 rows in set (0.00 sec)
```

图 3-4 loan_amount 查询结果

表明已导出成功。其他数据的导出可依次更改集群路径及表名即可。

3.5.3. 实验任务三：数据可视化

本项目中使用 Flask 框架将数据从 Mysql 中读出，使用 Echarts 进行图表展示。

3.5.3.1. 步骤一：创建项目

[hadoop@master data]$ su - root
[root@master ~]# ./pycharm-community-2020.1.2/bin/pycharm.sh

在 Pycharm 中创建项目 h3c_flask，项目结构如下：

1. 在该项目中创建文件夹 static，将 Echarts 的 js 文件放入该文件夹下；
2. 在该项目中创建文件夹 templates，用于保存网页文件；
3. 在根目录下创建 python 文件 Start.py；

整个项目结构如下：

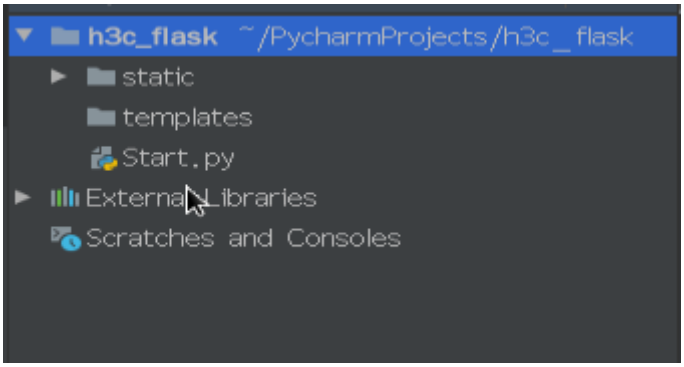


图 3-5 项目结构

使用 pip install 指令安装 pymysql 包。

3.5.3.2. 步骤二：借款金额分布可视化

本实验将 loan_amount 表中的数据用柱状图展示出来，因此需要在 Start.py 文件中引入 flask 包，并创建路由，引入 Echarts 插件，进行可视化展示。

1. 在 Start.py 文件中，加入以下代码：

```
from flask import Flask, render_template
import pymysql,numpy as np

app = Flask(__name__)
#  借款金额分布——柱状图
@app.route("/1")
def bar():
    data = GetData()
    xData = []
    yData = []
    for i in data:
        xData.append(i[0])
        yData.append(i[1])
    return render_template("bar.html", data={"xData": xData, "yData": yData})
    pass
def GetData():
    connect = pymysql.connect(
        host="master",
        user="root",
        password="Password123$",
        database="finance",
        charset="utf8"
    )
    cursor = connect.cursor()
    # 1 借款金额分布
    cursor.execute("SELECT * FROM loan_amount ORDER BY
CAST(SUBSTRING_INDEX(zone,'-',1) AS SIGNED);")
    data = cursor.fetchall()
```

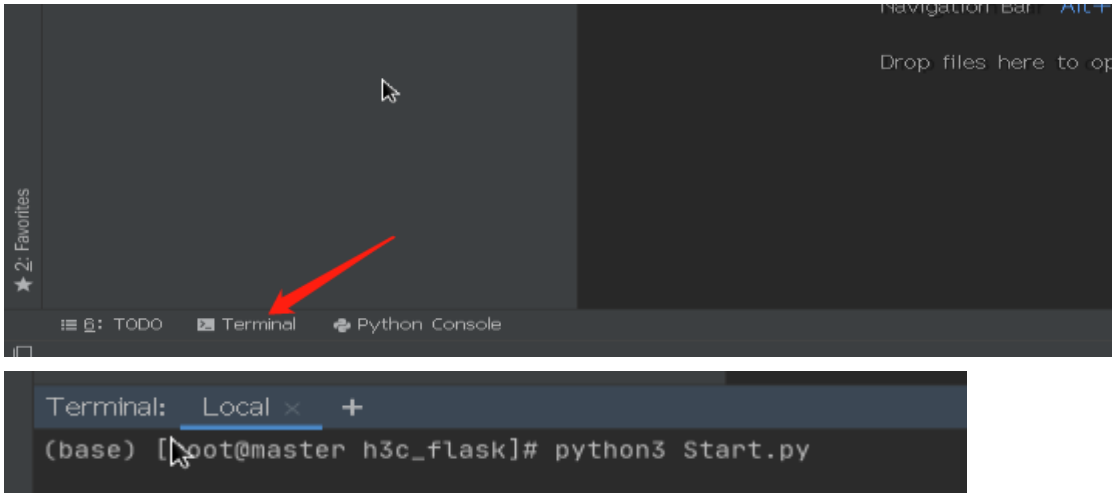
```
    cursor.close()
    connect.close()
    return [i for i in data]
app.run(debug=True)
```

2. 在 templates 文件夹中新建 html 文件，文件名为 bar.html，填入以下代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>借款金额分布</title>
    <script src="../static/echarts.min.js"></script>
</head>
<body>
    <div id="root" style="width: 1000px;height: 600px"></div>
    <script>
        root = document.getElementById("root")
        echart = echarts.init(root)
        echart.setOption({
            title :{
                text :"借款金额分布图",
                left:"center",
                textStyle:{
                    fontSize:"200%"
                }
            },
            legend:{
                left:"right",
                top:"center",
            },
            xAxis:{
                type:"category",
                data:{{data.xData|safe}},
                name:"借款金额：美元"
            },
            yAxis:{
                name:"人数"
            },
            series:[{
                name:"人数",
                type:"bar",
                label:{
                    show:true,
                    position:'top'
                },
```

```
                data:{{data.yData}}
            }]
        })
    </script>
</body>
</html>
```

3. 运行程序，在浏览器中打开网址 http://127.0.0.1:5000/1，可以看到可视化结果如下图所示：



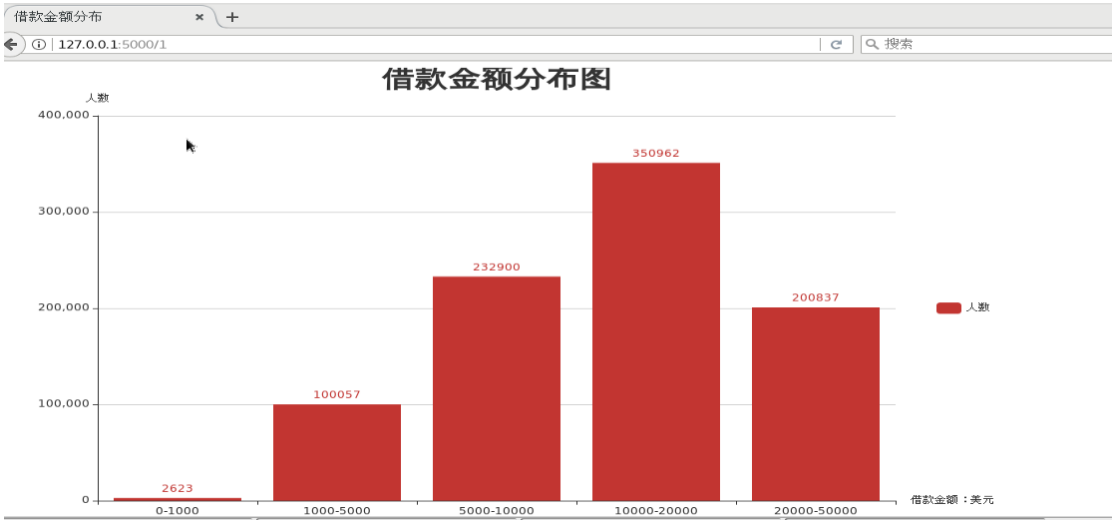


图 3-6 借款金额分布可视化结果图

3.5.3.3. 步骤三：借款等级分布可视化

loan_grade 表中有三个字段，分别是借款等级、子等级、人数，本实验要将每个等级的子等级显示出来，需要使用堆叠条形图进行可视化。

1. 在 Start.py 中加入如下代码：

```
# 借款等级分布图——堆叠条形图
@app.route("/2")
def double_bar():
    data=GetData()
    fData=[]
```

```
        vData= []
        grade=[1,2,3,4,5]
        for j in range(0,5):
            for i in range(j*7,j*7+7):
                vData.append(data[i][2])
            # print(fData[j])
            fData.append({"category":grade[j],"value":vData})
            vData=[]

        return render_template("double_bar.html",data=fData)
        pass
```

将获取数据的语句改为如下代码：

```
# 2 借款等级分布 需要注释掉上一条命令
cursor.execute("SELECT * FROM loan_grade ORDER BY RIGHT(subgrade,1),grade")
```

2. 在 template 文件夹中新建 double_bar.html 文件，在文件中加入如下代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>借款等级分布</title>
    <script src="../static/echarts.min.js"></script>
</head>
<body>
    <div id="root" style="width: 1000px;height: 600px">

    </div>
    <script>
        root = document.getElementById("root")
        echart = echarts.init(root)
        var series1=[];
        var length={{data|tojson}}.length
        for(var i=0;i<length;i++){
            series1.push({
                name:i+1,
                type:'bar',
                stack:'总量',
                label:{
                        show:true,
                        position:'insideRight'
                    },
                data:{{data|tojson}}[i].value
            })
        }
        echart.setOption({
```

```
                title :{
                    text :"借款等级分布图",
                    left:"center",
                    textStyle:{
                        fontSize:"200%"
                    }
                },
                 tooltip: {
                    trigger: 'axis',
                    axisPointer: {
                        type: 'shadow'
                    }
                },
                legend:{
                    left:"center",
                    top:"bottom",
                    data:['1','2','3','4','5'],
                    name:'子等级'
                },
                xAxis:{
                    type:'value',
                    name:'人数'
                },
                yAxis:{
                    type:'category',
                    data:['A','B','C','D','E','F','G'],
                    name:'等级'
                },
                series:series1
            })
        </script>
    </body>
</html>
```

3. 在浏览器中输入地址 http://127.0.0.1:5000/2，查看结果如下图：

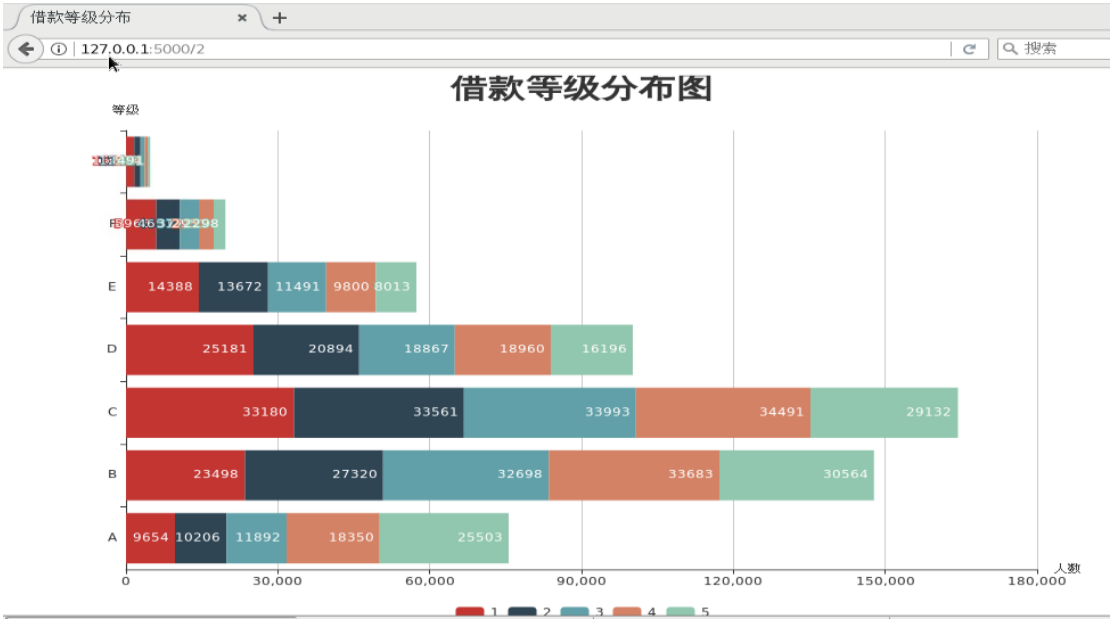


图 3-7 借款等级分布可视化图

3.5.3.4. 步骤四：借款等级与借款金额的关联关系可视化

表 grade_amount 中包含三个字段借款等级、借款金额区间、人数，也需要使用堆叠条状图将数据进行可视化，显示每个借款金额区间内的各借款等级的人数分布。

1. 在 Start.py 文件中加入以下代码：

```
# 借款等级与借款金额分布图——堆叠条形图
@app.route("/3")
def grade_amount():
    data=GetData()
    fData=[]
    vData= []
    grade=['A','B','C','D','E','F','G']
    for j in range(0, len(grade)):
        for i in range(j*5,j*5+5):
            vData.append(data[i][2])
        # print(fData[j])
        fData.append({"category":grade[j],"value":vData})
        vData=[]

    return render_template("grade_amount.html",data=fData)
    pass
```

将获取数据处的代码改为：

```
# 3 借款等级与借款金额的关系 需要注释掉上一条命令
cursor.execute("SELECT * FROM grade_amount ORDER BY
grade,CAST(SUBSTRING_INDEX(amount,'-',1) AS SIGNED)")
```

2. 在 template 文件夹中新建 HTML 文件，文件名为 grade_amount.html，文件内容写入以下代码：

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>借款等级与借款金额关系</title>
    <script src="../static/echarts.min.js"></script>
</head>
<body>
    <div id="root" style="width: 1000px;height: 600px">
    </div>
    <script>
        root = document.getElementById("root")
        echart = echarts.init(root)
        console.log({{data|tojson}})
        var series1=[];
        var zone=[];
        var length={{data|tojson}}.length
        for(var i=0;i<length;i++){
            zone.push({{data|tojson}}[i].category)
        }
        for(var i=0;i<length;i++){
            series1.push({
                name:zone[i],
                type:'bar',
                stack:'总量',
                data:{{data|tojson}}[i].value
            })
        }
        console.log(series1)
        echart.setOption({
            title :{
                text :"借款等级与借款金额关系图",
                left:"center",
                textStyle:{
                    fontSize:"200%"
                }
            },
             tooltip: {
                trigger: 'axis',
                axisPointer: {
                    type: 'shadow'
                }
            },
            legend:{
                left:"center",
```

```
                top:"bottom",
                data:zone,
                name:'等级',
                show:true
            },
            xAxis:{
                type:'value',
                name:'人数'
            },
            yAxis:{
                type:'category',

data:['0-1000','1000-5000','5000-10000','10000-20000','20000-50000'],
                name:'借款金额区间'
            },
            series:series1
        })
    </script>
</body>
</html>
```

3. 在浏览器中输入 http://127.0.0.1:5000/3，显示结果如下：

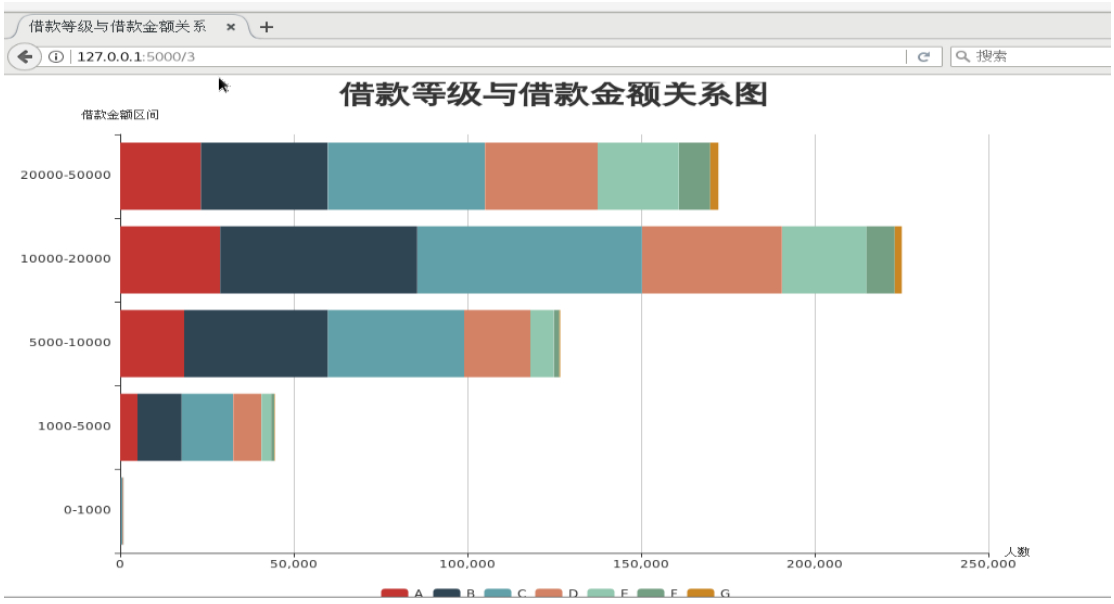


图 3-8 借款等级与借款金额的关联关系可视化图

3.5.3.5. 步骤五：借款金额与工作年限、年收入的关联关系可视化

表 amount_wkyear 中包含三个字段工作年限、借款金额区间、人数，也需要使用堆叠条状图将数据进行可视化，显示每个借款金额区间内的工作年限的人数分布。

1. 在 Start.py 文件中加入以下代码：

```
# 借款金额与工作年限分布图——堆叠条形图
@app.route("/4")
def amount_wkyears():
```

```
        data=GetData()
        fData=[]
        vData= []
        tmp=[]
        for i in data:
            tmp.append(i[0])
        wkyear = np.unique(tmp)
        for j in range(0, len(wkyear)):
            for i in range(j*5,j*5+5):
                vData.append(data[i][2])
            fData.append({"category":wkyear[j],"value":vData})
            vData=[]

        return render_template("amount_wkyears.html",data=fData)
        pass
```

将获取数据处的代码更改为：

4 借款金额与工作年限的关系 需要注释掉上一条命令

```
cursor.execute("SELECT * FROM amount_wkyear ORDER BY CAST( LEFT(wkyears,2) AS SIGNED),CAST(SUBSTRING_INDEX(amount,'-',1) AS SIGNED)")
```

2. 在 template 文件夹中新建 HTML 文件，文件名为 amount_wkyears.html，文件内容写入以下代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>借款金额与工作年限关系图</title>
    <script src="../static/echarts.min.js"></script>
</head>
<body>
    <div id="root" style="width: 1000px;height: 600px"></div>
    <script>
        root = document.getElementById("root")
        echart = echarts.init(root)
        console.log({{data|tojson}})
        var series1=[];
        var zone=[];
        var length={{data|tojson}}.length
        for(var i=0;i<length;i++){
            zone.push({{data|tojson}}[i].category)
        }
        for(var i=0;i<length;i++){
            series1.push({
                name:zone[i],
```

```
                type:'bar',
                stack:'总量',
                data:{{data|tojson}}[i].value
            })
        }
        console.log(series1)
        echart.setOption({
            title :{
                text :"借款金额与工作年限关系图",
                left:"center",
                textStyle:{
                    fontSize:"200%"
                }
            },
             tooltip: {
                trigger: 'axis',
                axisPointer: {
                    type: 'shadow'
                }
            },
            legend:{
                left:"center",
                top:"bottom",
                data:zone,
                name:'等级',
                show:true
            },
            xAxis:{
                type:'value',
                name:'人数'
            },
            yAxis:{
                type:'category',

data:['0-1000','1000-5000','5000-10000','10000-20000','20000-50000'],
                name:'借款金额区间'
            },
            series:series1
        })
    </script>
</body>
</html>
```

3. 在浏览器中输入 http://127.0.0.1:5000/4，显示结果如下：

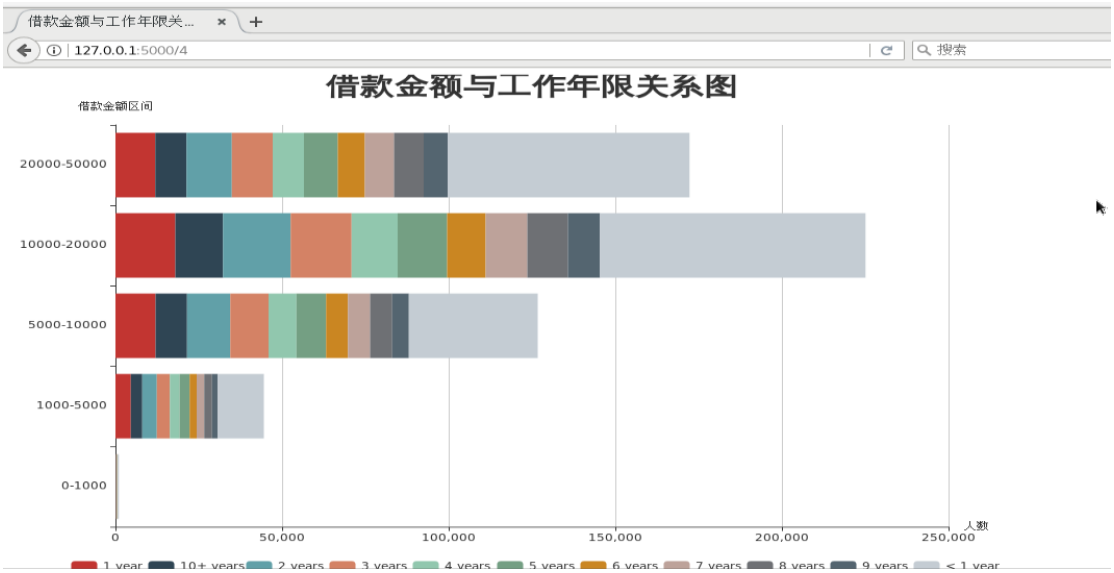


图 3-9 借款金额与工作年限关联关系可视化图

表 amount_salary 中包含三个字段年收入区间、借款金额区间、人数，需要使用堆叠条形图进行可视化。步骤如下：

1. 在 Start.py 文件中加入以下代码：

```
# 借款金额与年收入分布图——堆叠条形图
@app.route("/5")
def amount_salary():
    data=GetData()
    fData=[]
    vData= []
    tmp=[]
    for i in data:
        tmp.append(i[0])
    salary = np.unique(tmp)
    salary[3],salary[4]=salary[4],salary[3]
    for j in range(0, len(salary)):
        for i in range(j*5,j*5+5):
            if(j>0):
                m=i-1
                vData.append(data[m][2])
            else:
                vData.append(data[i][2])
        fData.append({"category":salary[j],"value":vData})
        vData=[]
    return render_template("amount_salary.html",data=fData)
    pass
```

将获取数据处的代码更改如下：

```
# 5 借款金额与年收入的关系  需要注释掉上一条命令
cursor.execute("SELECT * FROM amount_salary ORDER BY
CAST(SUBSTRING_INDEX(salary,'_',1) AS SIGNED),CAST(SUBSTRING_INDEX(amount,'_',1)
```

```
AS SIGNED)")
```

2. 在 template 文件夹中新建 HTML 文件，文件名为 amount_salary.html，文件内容写入以下代码：

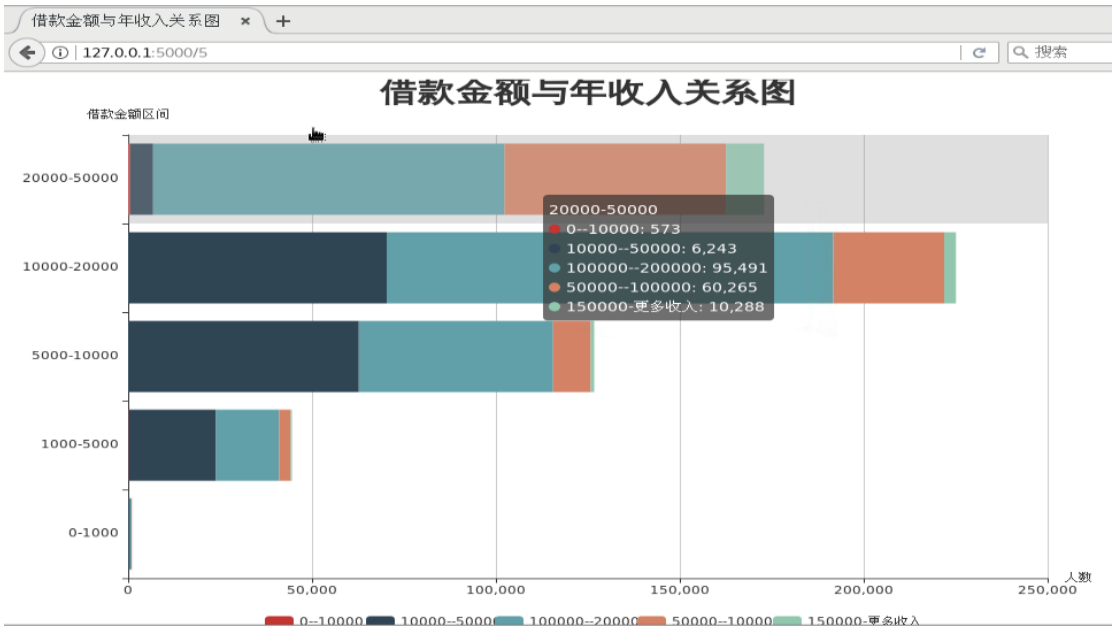
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>借款金额与年收入关系图</title>
    <script src="../static/echarts.min.js"></script>
</head>
<body>
    <div id="root" style="width: 1000px;height: 600px"></div>
    <script>
        root = document.getElementById("root")
        echart = echarts.init(root)
        console.log({{data|tojson}})
        var series1=[];
        var zone=[];
        var length={{data|tojson}}.length
        for(var i=0;i<length;i++){
            zone.push({{data|tojson}}[i].category)
        }
        for(var i=0;i<length;i++){
            series1.push({
                name:zone[i],
                type:'bar',
                stack:'总量',
                data:{{data|tojson}}[i].value
            })
        }
        console.log(series1)
        echart.setOption({
            title :{
                text :"借款金额与年收入关系图",
                left:"center",
                textStyle:{
                    fontSize:"200%"
                }
            },
             tooltip: {
                trigger: 'axis',
                axisPointer: {
                    type: 'shadow'
```

```
                    }
                },
                legend:{
                    left:"center",
                    top:"bottom",
                    data:zone,
                    name:'等级',
                    show:true
                },
                xAxis:{
                    type:'value',
                    name:'人数'
                },
                yAxis:{
                    type:'category',

    data:['0-1000','1000-5000','5000-10000','10000-20000','20000-50000'],
                    name:'借款金额区间'
                },
                series:series1
            })
        </script>
    </body>
</html>
```

3. 在浏览器中输入 http://127.0.0.1:5000/5，显示结果如下：

图 3-10



借款金额与年收入的关联关系可视化图

3.5.3.6. 步骤六：借款金额与房屋所有权状态的关联关系可视化

表 amount_house 中包含三个字段房屋状态、借款金额区间、人数，本实验使用堆叠柱状图

将数据进行可视化，显示每个借款金额区间内的各房屋所有权状态的人数分布。

1. 在 Start.py 文件中加入以下代码：

```
# 借款金额与房屋所有权状态——堆叠柱状图
@app.route("/6")
def amount_house():
    data=GetData()
    fData=[]
    vData= []
    tmp=[]
    for i in data:
        tmp.append(i[0])
    house = np.unique(tmp)
    for j in range(0, len(house)):
        # 由于“OTHER”状态下的“0-1000"没有值因此需要在该处增加数据 0
        for i in range(j*5,j*5+5):
            if(j==1 and i==5):
                vData.append(0)
            if(j==1 and i>5):
                i=i-1
            if(j>1):
                i=i-1
            vData.append(data[i][2])
        fData.append({"category":house[j],"value":vData})
        vData=[]
    return render_template("amount_house.html",data=fData)
    pass
```

将获取数据处代码更改如下：

```
# 6 借款金额与房屋所有权状态的关系 需要注释掉上一条命令
cursor.execute("SELECT * FROM amount_house ORDER BY
house,CAST(SUBSTRING_INDEX(amount,'_',1) AS SIGNED)")
```

2. 在 template 文件夹中新建 HTML 文件，文件名为 amount_house.html，文件内容写入以下代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>借款金额与房屋所有权状态关系图</title>
    <script src="../static/echarts.min.js"></script>
</head>
<body>
    <div id="root" style="width: 1000px;height: 600px">

    </div>
```

```
<script>
    root = document.getElementById("root")
    echart = echarts.init(root)
    var length={{data|tojson}}.length
    var house=[]
    var series1=[];
    for(var i=0;i<length;i++){
        house.push({{data|tojson}}[i].category);
    }
    var zone=['0-1000','1000-5000','5000-10000','10000-20000','20000-50000']
    for(var i=0;i<length;i++){
        series1.push({
            name:house[i],
            type:'bar',
            data:{{data|tojson}}[i].value,
            label:{
                show:true,
                position:'top'
            }
        })
    }

    echart.setOption({
        title :{
            text :"借款金额与房屋状态关系图",
            left:"center",
            textStyle:{
                fontSize:"200%"
            }
        },
        legend:{
            left:"center",
            top:"bottom",
            data:house
        },
        xAxis:[{
            type:'category',
            data:zone
        }],
        yAxis:[{
            type:'value'
        }],
        series:series1
    })
```

```
        </script>
    </body>
</html>
```

3. 在浏览器中输入 http://127.0.0.1:5000/6，显示结果如下：

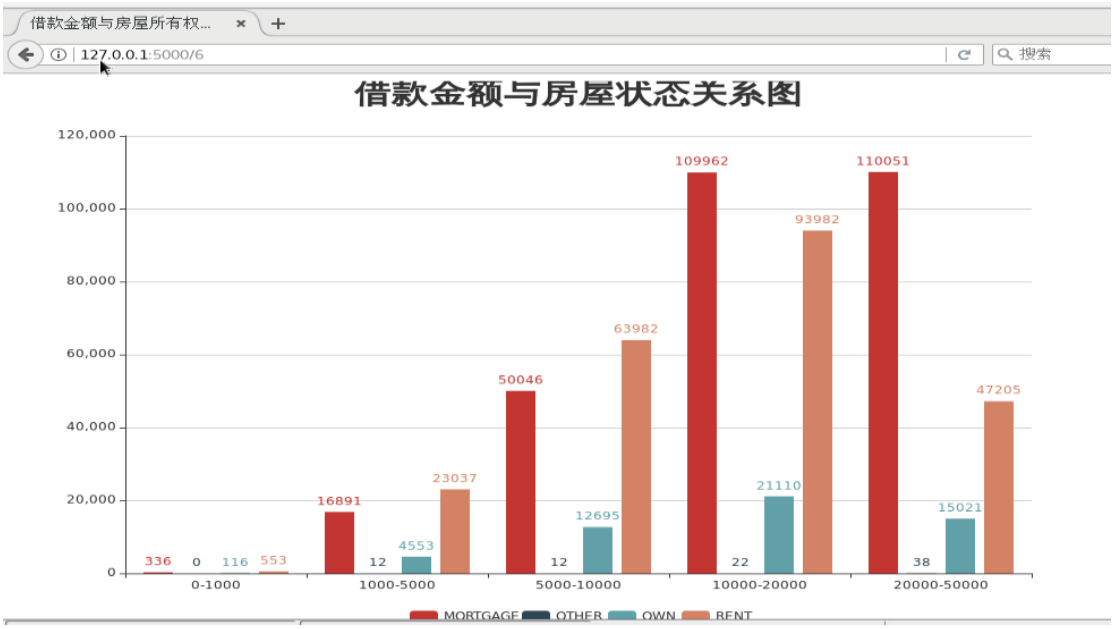


图 3-11 借款金额与房屋状态关联关系可视化图