

# 大数据技术与应用(二)课程设计

课程名： 大数据技术与应用(二)课程设计

题 目： 基于 Spark 的出租房屋大数据分析

班 级：

学 号：

姓 名：

# 目录

1. 项目背景与功能.....	1
1.1 项目背景.....	1
1.2 项目功能.....	1
1.3 运行环境.....	1
2. 数据集与数据预处理.....	1
2.1 原始数据集.....	1
2.1.1 数据集说明.....	1
2.1.2 原始数据集展示 (前 10 条).....	2
2.2 数据预处理.....	3
2.2.1 Excel 数据预处理 .....	3
2.2.2 数据字段说明.....	3
2.3 数据上传.....	3
2.3.1 处理数据.....	3
2.3.2 上传至 HDFS .....	4
3. 数据分析.....	5
3.1 使用 Spark SQL 进行统计分析 (至少 5 个).....	5
3.1.1 统计租金前 10 的市辖区.....	5
3.1.2 统计市辖区出租房屋的最大面积和最小面积.....	5
3.1.3 统计市辖区出租房屋的平均面积及平均租金.....	6
3.1.4 统计地级市租金在 1000 以上的房屋数量.....	7
3.1.5 统计地级市一房一厅一卫的房屋最低租金.....	7
4. Flask 与 ECharts 数据大屏可视化.....	7
4.1 开发环境及伪动态算法介绍.....	8
4.1.1 利用 Flask 与 ECharts .....	8
4.1.2 伪动态算法建模.....	8
4.2 Flask+ ECharts 可视化.....	9
4.2.1 数据导入 mysql.....	9
4.2.2 总体情况预览可视化分析.....	9
4.2.3 深圳、佛山租房各项指标对比.....	10
4.2.4 不同朝向的租房房间数量、楼层可视化分析.....	11
4.2.5 地区均价与面积对比分析.....	12
4.2.6 房源数量 TOP10 分析 .....	13
4.2.7 租房类型对比.....	14
4.2.8 租房覆盖率分析展示.....	14
4.3 总体大屏可视化结果展示.....	15
5. Spark 机器学习 .....	16
5.1 K-Means 聚类分析.....	16
5.1.1 数据预处理.....	16
5.1.2 特征工程.....	17
5.1.3 K-Means 模型聚类.....	17
5.1.4 模型评估.....	18
5.1.5 模型优化.....	19

5.1.6 结论.....	19
5.2 Lasso 回归模型预测租金 .....	19
5.2.1 数据预处理.....	19
5.2.2 特征工程.....	20
5.2.3 构建模型.....	20
5.2.4 模型评估.....	20
5.2.5 参数调优.....	21
6. 实时分析.....	22
6.1 出租房屋数量实时分析.....	22
6.1.1 实验环境准备.....	22
6.1.2 数据处理和 Python 操作 Kafka .....	23
6.1.3 Spark Streaming 实时处理数据.....	24
6.1.4 分析系统.....	26
7. 总结.....	27

# 1. 项目背景与功能

## 1.1 项目背景

此次项目是分析广东七地二手房的房子情况，数据是来自链家的二手房，此次的爬取的数据是 2023 年 5 月最新的数据，数据包含了十个字段，都是爬取后在 excel 做了简单的数据预处理，最后导入虚拟机。

## 1.2 项目功能

本项目通过 EXCEL 对出租房屋数据集进行预处理，并使用 Spark SQL 进行了租金前十的市辖区、各市辖区出租房屋的最大面积、最小面积、平均面积等数据分析；使用 Flask+Echarts 对数据进行大屏可视化；使用 K-Means 聚类对出租房屋进行聚类分析，根据聚类结果可知，该数据集可分为三类；使用 Lasso 回归模型进行预测租金，该模型优化后 RMSE 值为 1074；

## 1.3 运行环境

使用 Ubuntu16.04 的虚拟机版本，软件方面使用的有 Hadoop2.7.1、Spark2.1.0、HBase1.1.5、JDK1.8、Scala2.11.8、MySQL、Kafka\_2.11-0.10.2.0、Sbt、Maven3.3.9、Hive2.1.0、Pycharm2016.3、python3.8

# 2. 数据集与数据预处理

## 2.1 原始数据集

### 2.1.1 数据集说明

<input type="checkbox"/>	清远租房信息_清远出租房源 房屋出租价格【...	<div>启动</div> <div>未运行</div>	未设置	-
	我的任务组 9 天前	<div>已停止 采集到3000条数据(重...</div>	未设置	9分31秒
<input type="checkbox"/>	湛江租房信息_湛江出租房源 房屋出租价格【...	<div>启动</div> <div>未运行 全部</div>	未设置	-
	我的任务组 9 天前	<div>已停止 采集到2400条数据(重... 全部</div>	未设置	9分40秒
<input type="checkbox"/>	惠州租房信息_惠州出租房源 房屋出租价格【...	<div>启动</div> <div>未运行</div>	未设置	-
	我的任务组 9 天前	<div>已停止 采集到3000条数据(重...</div>	未设置	10分3秒
<input type="checkbox"/>	珠海租房信息_珠海出租房源 房屋出租价格【...	<div>启动</div> <div>未运行</div>	未设置	-
	我的任务组 9 天前	<div>已停止 采集到1700条数据(重...</div>	未设置	6分9秒
<input type="checkbox"/>	深圳租房信息_深圳出租房源 房屋出租价格【...	<div>启动</div> <div>未运行</div>	未设置	-
	我的任务组 9 天前	<div>已停止 采集到1700条数据(重...</div>	未设置	47分17
<input type="checkbox"/>	中山租房信息_中山出租房源 房屋出租价格【...	<div>启动</div> <div>未运行</div>	未设置	-
	我的任务组 9 天前	<div>已停止 采集到1900条数据(重...</div>	未设置	8分4秒
<input type="checkbox"/>	佛山租房信息_佛山出租房源 房屋出租价格【...	<div>启动</div> <div>未运行</div>	未设置	-
	我的任务组 9 天前	<div>已停止 采集到1700条数据(重...</div>	未设置	7分8秒

图 2-1 数据集采集

采集时间：2023-5-8

采集网址：<https://fs.lianjia.com/zufang/>、<https://sz.lianjia.com/zufang/>、<https://zs.lianjia.com/zufang/>、<https://zh.lianjia.com/zufang/>、<https://hz.lianjia.com/zufang/>、<https://zj.lianjia.com/zufang/>、

<https://qy.lianjia.com/zufang/>

内容：佛山、深圳、珠江、湛江、惠州、清远、中山七个城市的租房信息

### 2.1.2 原始数据集展示 (前 10 条)

	A	B	C	D	E	F	G	H
1	珠海	香洲区-吉大-德丰大厦	80.00m <sup>2</sup>	东南	3室1厅1卫	中 楼 层 (11层)	1000 元/月	
2	珠海	香洲区-吉大-海湾花园	134.00m <sup>2</sup>	东南	3室1厅2卫	低 楼 层 (29层)	16000 元/月	
3	珠海	香洲区-吉大-嘉年华国际公寓	48.40m <sup>2</sup>	东	1室1厅1卫	低 楼 层 (29层)	2300 元/月	
4	珠海	香洲区-吉大-雍和花园	267.00m <sup>2</sup>	东南	5室2厅2卫	高 楼 层 (22层)	3500 元/月	
5	珠海	香洲区-吉大-园林花园二期	89.00m <sup>2</sup>	东南	2室2厅1卫	中 楼 层 (17层)	3300 元/月	
6	珠海	香洲区-翠微-东方新地	111.68m <sup>2</sup>	东南	3室2厅2卫	高楼层 (7层)	2700 元/月	
7	珠海	香洲区-前山-荣泰河庭	79.00m <sup>2</sup>	西南	2室2厅1卫	低 楼 层 (11层)	2500 元/月	
8	珠海	香洲区-老香洲-中珠水晶堡	120.00m <sup>2</sup>	南北	4室1厅0卫	低 楼 层 (20层)	2000 元/月	
9	珠海	香洲区-老香洲-华富街	121.00m <sup>2</sup>	南	3室2厅2卫	高楼层 (8层)	2200 元/月	
10	珠海	香洲区-老香洲-梅华东路284号	74.00m <sup>2</sup>	南	2室1厅1卫	高楼层 (5层)	2700 元/月	

就绪 辅助功能: 一切就绪

图 2-2 原始数据集

## 2.2 数据预处理

### 2.2.1 Excel 数据预处理

	A	B	C	D	E	F	G	H	I
1	district	address	area(m <sup>2</sup> )	orientation	room	living	bathroom	floor	rent
2	珠海	香洲区	397	南	9	4	8	4	100000
3	珠海	香洲区	556	东南	5	2	0	36	55600
4	珠海	香洲区	316	东南	4	2	4	42	40000
5	珠海	香洲区	255	南北	4	2	3	17	30000
6	珠海	香洲区	379.24	东南	6	2	3	24	29000
7	珠海	香洲区	395	西	5	3	4	3	26000
8	珠海	香洲区	299.43	东南	5	2	4	23	22000
9	珠海	香洲区	211	东南	4	2	3	44	20000
10	珠海	香洲区	260	东	1	0	0	33	20000

图 2-3 数据集预处理

### 2.2.2 数据字段说明

列名称	说明
district	字符串类型，所在地级市
address	字符串类型，所在市辖区
area	浮点数类型，出租房屋面积
orientation	字符串类型，出租房屋朝向
room	整数类型，室数
living	整数类型，厅数
bathroom	整数类型，卫生间数
floor	整数类型，所在楼层
rent	整数类型，租金

## 2.3 数据上传

### 2.3.1 处理数据

```
hadoop@dblab-VirtualBox:~$ cd /usr/local/hadoop
hadoop@dblab-VirtualBox:/usr/local/hadoop$ ./sbin/start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-na
menode-dblab-VirtualBox.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hadoop-da
tanode-dblab-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-ha
dooop-secondarynamenode-dblab-VirtualBox.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hadoop-resource
manager-dblab-VirtualBox.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hadoop-n
odemanager-dblab-VirtualBox.out
hadoop@dblab-VirtualBox:/usr/local/hadoop$ jps
3299 NameNode
3811 ResourceManager
3940 NodeManager
4246 Jps
3657 SecondaryNameNode
3455 DataNode
```

图 2-4 启动 Hadoop

```
hadoop@dblab-VirtualBox:~$ cd /usr/local/bigdata/dataset
hadoop@dblab-VirtualBox:/usr/local/bigdata/dataset$ head -10 rent.csv
珠海,香洲区,397,南,9,4,8,4,100000,
珠海,香洲区,556,东南,5,2,0,36,55600,
珠海,香洲区,316,东南,4,2,4,42,40000,
珠海,香洲区,255,南北,4,2,3,17,30000,
珠海,香洲区,379.24,东南,6,2,3,24,29000,
珠海,香洲区,395,西,5,3,4,3,26000,
珠海,香洲区,299.43,东南,5,2,4,23,22000,
珠海,香洲区,211,东南,4,2,3,44,20000,
珠海,香洲区,260,东,1,0,0,33,20000,
珠海,香洲区,252,东,5,1,4,41,20000,
```

图 2-5 数据展示

```
#!/bin/bash

infile=$1
outfile=$2
awk -F "," 'BEGIN{
srand();
    id=0;
}
{
    id=id+1;
    print id"\t"$1"\t"$2"\t"$3"\t"$4"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9
}' $infile> $outfile
~
~
~
~
```

图 2-6 pre\_rent.sh

```
hadoop@dblab-VirtualBox:/usr/local/bigdata/dataset$ bash ./pre_rent.sh rent.csv
rent.txt
hadoop@dblab-VirtualBox:/usr/local/bigdata/dataset$ head -10 rent.txt
1 珠海 香洲区 397 南 9 4 8 4 100000
2 珠海 香洲区 556 东南 5 2 0 36 55600
3 珠海 香洲区 316 东南 4 2 4 42 40000
4 珠海 香洲区 255 南北 4 2 3 17 30000
5 珠海 香洲区 379.24 东南 6 2 3 24 29000
6 珠海 香洲区 395 西 5 3 4 3 26000
7 珠海 香洲区 299.43 东南 5 2 4 23 22000
8 珠海 香洲区 211 东南 4 2 3 44 20000
9 珠海 香洲区 260 东 1 0 0 33 20000
10 珠海 香洲区 252 东 5 1 4 41 20000
```

图 2-7 执行 shell 脚本并查看处理后数据

### 2.3.2 上传至 HDFS

```
hadoop@dblab-VirtualBox:/usr/local/hadoop$ ./bin/hdfs dfs -mkdir -p /bigdata/dataset
hadoop@dblab-VirtualBox:/usr/local/hadoop$ ./bin/hdfs dfs -put /usr/local/bigdata/dataset/rent.txt /bigdata/dataset
hadoop@dblab-VirtualBox:/usr/local/hadoop$ ./bin/hdfs dfs -cat /bigdata/dataset/rent.txt | head -10
1 珠海 香洲区 397 南 9 4 8 4 100000
2 珠海 香洲区 556 东南 5 2 0 36 55600
3 珠海 香洲区 316 东南 4 2 4 42 40000
4 珠海 香洲区 255 南北 4 2 3 17 30000
5 珠海 香洲区 379.24 东南 6 2 3 24 29000
6 珠海 香洲区 395 西 5 3 4 3 26000
7 珠海 香洲区 299.43 东南 5 2 4 23 22000
8 珠海 香洲区 211 东南 4 2 3 44 20000
9 珠海 香洲区 260 东 1 0 0 33 20000
10 珠海 香洲区 252 东 5 1 4 41 20000
cat: Unable to write to output stream.
```

图 2-8 成功上传图示

### 3. 数据分析

#### 3.1 使用 Spark SQL 进行统计分析 (至少 5 个)

##### 3.1.1 统计租金前 10 的市辖区

```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> val df = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("file:///usr/local/bigdata/dataset/rent.csv").toDF("district","address","area","orientation","room","living","bathroom","floor","rent","null")
df: org.apache.spark.sql.DataFrame = [district: string, address: string ... 8 more fields]

scala> df.createOrReplaceTempView("myrent")
```

图 3-7 导入数据至 Spark sql

```
scala> val top10 = spark.sql("SELECT address, AVG(rent) AS avg_rent FROM myrent GROUP BY address ORDER BY avg_rent DESC LIMIT 10")
top10: org.apache.spark.sql.DataFrame = [address: string, avg_rent: double]
```

```
scala> top10.show()
+-----+-----+
|address|      avg_rent|
+-----+-----+
| 盐田区 | 12738.235294117647|
| 南山区 | 11677.045454545454|
| 福田区 | 10140.510204081633|
| 龙华区 | 9114.492753623188|
| 罗湖区 | 7211.458762886598|
| 宝安区 | 6505.376344086021|
| 白云 | 4357.830882352941|
| 龙岗区 | 4354.83203125|
| 香洲区 | 4308.927524429967|
| 光明区 | 4137.037037037037|
+-----+-----+
```

```
scala> top10.rdd.map(row => row.mkString(",")).saveAsTextFile("file:///usr/local/bigdata/spark_sql/top_10_rent")
```

图 3-8 查询结果图示

##### 3.1.2 统计市辖区出租房屋的最大面积和最小面积

```
scala> val rent_show = spark.sql("SELECT address,avg(area) AS avg_area,avg(rent) AS avg_rent FROM myrent group by(address)")
rent_show: org.apache.spark.sql.DataFrame = [address: string, avg_area: double ... 1 more field]
```

```
scala> rent_show.show()
+-----+-----+-----+
|address|      avg_area|      avg_rent|
+-----+-----+-----+
| 三角镇 | 87.82181818181817| 1651.8181818181818|
| 遂溪县 | 141.06790697674418| 2239.5348837209303|
| 廉江市 | 108.25| 1545.0|
| 古城镇 | 115.75833333333333| 2865.0|
| 赤坎区 | 99.41789855072467| 2843.7507246376813|
| 东升镇 | 110.24469387755101| 1965.3061224489795|
| 佛冈县 | 115.0| 1500.0|
| 三水 | 91.40081818181817| 1607.8181818181818|
| 仲恺高新技术开发区 | 98.74971428571429| 2134.714285714286|
| 东区 | 90.32144578313257| 2374.253012048193|
| 霞山区 | 92.93375728155345| 2736.617475728155|
| 惠城 | 104.48824840764331| 2603.076433121019|
| 禅城 | 75.7373157894737| 2158.521052631579|
| 斗门区 | 100.77454545454542| 1905.4545454545455|
| 龙华区 | 104.35043478260869| 9114.492753623188|
| 福田区 | 80.66530612244895| 10140.510204081633|
| 金湾 | 103.04999999999998| 2324.733727810651|
| 东凤镇 | 87.57515151515151| 1842.4242424242425|
| 西区 | 85.68765217391307| 1952.5217391304348|
| 坡头区 | 107.49653846153845| 2107.269230769231|
+-----+-----+-----+
only showing top 20 rows
```

```
scala> rent_show.rdd.map(row => row.mkString(",")).saveAsTextFile("file:///usr/local/bigdata/spark_sql/rent_show")
```



```
scala> val area_show = spark.sql("SELECT address,MAX(area) AS max_area, MIN(area) AS min_area from myrent
group by(address)")
area_show: org.apache.spark.sql.DataFrame = [address: string, max_area: double ... 1 more field]

scala> area_show.show()
+-----+-----+-----+
| address|max_area|min_area|
+-----+-----+-----+
| 三角镇|148.0|34.5|
| 遂溪县|784.0|33.0|
| 廉江市|126.0|80.0|
| 古城镇|216.0|47.1|
| 赤坎区|300.0|10.0|
| 东升镇|220.0|74.0|
| 佛冈县|120.0|110.0|
| 三水|170.0|30.0|
| 仲恺高新技术产业开发区|144.0|46.0|
| 东区|360.0|15.0|
| 霞山区|1000.0|5.0|
| 惠城|610.0|30.0|
| 禅城|183.0|16.0|
| 斗门区|329.43|21.0|
| 龙华区|750.0|14.9|
| 福田区|511.78|9.0|
| 金湾区|203.0|30.0|
| 东风镇|200.0|36.0|
| 西区|214.0|28.0|
| 坡头区|150.0|38.0|
+-----+-----+-----+
only showing top 20 rows

scala> area_show.rdd.map(row => row.mkString(",")).saveAsTextFile("file:///usr/local//bigdata/spark_sql/
area_show")
```

图 3-9 统计结果图示

### 3.1.3 统计市辖区出租房屋的平均面积及平均租金

```
scala> val rent_show = spark.sql("SELECT address,avg(area) AS avg_area,avg(rent) AS avg_rent FROM myrent
group by(address)")
rent_show: org.apache.spark.sql.DataFrame = [address: string, avg_area: double ... 1 more field]

scala> rent_show.show()
+-----+-----+-----+
| address|avg_area|avg_rent|
+-----+-----+-----+
| 三角镇|87.82181818181817|1651.8181818181818|
| 遂溪县|141.06790697674418|2239.5348837209303|
| 廉江市|108.25|1545.0|
| 古城镇|115.75833333333333|2865.0|
| 赤坎区|99.41789855072467|2843.7507246376813|
| 东升镇|110.24469387755101|1965.3061224489795|
| 佛冈县|115.0|1500.0|
| 三水|91.40081818181817|1607.8181818181818|
| 仲恺高新技术产业开发区|98.74971428571429|2134.714285714286|
| 东区|90.32144578313257|2374.253012048193|
| 霞山区|92.93375728155345|2736.617475728155|
| 惠城|104.48824840764331|2603.076433121019|
| 禅城|75.7373157894737|2158.521052631579|
| 斗门区|100.77454545454542|1905.4545454545455|
| 龙华区|104.35043478260869|9114.492753623188|
| 福田区|80.66530612244895|10140.510204081633|
| 金湾区|103.04999999999998|2324.733727810651|
| 东风镇|87.57515151515151|1842.4242424242425|
| 西区|85.68765217391307|1952.5217391304348|
| 坡头区|107.49653846153845|2107.269230769231|
+-----+-----+-----+
only showing top 20 rows

scala> rent_show.rdd.map(row => row.mkString(",")).saveAsTextFile("file:///usr/local//bigdata/spark_sql/
rent_show")
```

图 3-10 统计结果图示

### 3.1.4 统计地级市租金在 1000 以上的房屋数量

```
scala> val count_rent = spark.sql("SELECT district,count(*) as count_district FROM myrent where rent>1000 group by(district)")
count_rent: org.apache.spark.sql.DataFrame = [district: string, count_district: bigint]

scala> count_rent.show()
+-----+-----+
|district|count_district|
+-----+-----+
|清远|1645|
|湛江|1831|
|惠州|2370|
|中山|1639|
|珠海|1544|
|深圳|1156|
|佛山|1440|
+-----+-----+

scala> count_rent.rdd.map(row => row.mkString(",")).saveAsTextFile("file:///usr/local/bigdata/spark_sql/count_rent")
```

图 3-11 统计结果图示

### 3.1.5 统计地级市一房一厅一卫的房屋最低租金

```
scala> val min_rent = spark.sql("SELECT address,min(rent) as min_rent FROM myrent where room>1 and living>1 and bathroom>1 group by(address)")
min_rent: org.apache.spark.sql.DataFrame = [address: string, min_rent: int]

scala> min_rent.show()
+-----+-----+
|address|min_rent|
+-----+-----+
|三角镇|1300|
|遂溪县|900|
|廉江市|2300|
|古城镇|2800|
|赤坎区|900|
|东升镇|1500|
|佛冈县|1500|
|三水|750|
|仲恺高新技术产业开发区|1600|
|东区|700|
|霞山区|1000|
|惠城|900|
|禅城|1500|
|斗门区|500|
|龙华区|2300|
|福田区|5200|
|金湾区|1000|
|东凤镇|1600|
|西区|1500|
|坡头区|1300|
+-----+-----+
only showing top 20 rows

scala> min_rent.rdd.map(row => row.mkString(",")).saveAsTextFile("file:///usr/local/bigdata/spark_sql/min_rent")
```

图 3-12 统计结果图示

## 4. Flask 与 ECharts 数据大屏可视化

本节中利用数学建模的知识构建了一个伪动态数据读取模型，配合 Flask 框架与 ECharts 绘图，构建了一个大屏可视化 web 界面，具体界面入下图所示。



图 4-1 数据大屏展示整体

## 4.1 开发环境及伪动态算法介绍

### 4.1.1 利用 Flask 与 ECharts

本文在 window 环境下利用 python3.9 搭建动态 web 应用。具体环境信息入下所示。

表 4-1 动态 web 可视化项目所用配置

序号	运行环境
1	Linux(Ubuntu16.04)
2	Win11
3	Hadoop 2.7.1
4	Python 3.9
5	Pymysql 1.0.3
6	Flask 2.3.2
7	Echarts

### 4.1.2 伪动态算法建模

本文所构建算法从读取数据部分入手，使不同时间序列下读取数据行数不同，使可视化结果中数据呈现动态的变化，关键式入下所示。

$$y = \begin{cases} 100 + \log_2 2^t & 0 \leq t \leq 15 \\ 211.8 * t - 80 & 16 \leq t \leq 60 \end{cases} \quad (4-1)$$

其中 y 为本次读取的数据条数，t 为程序所处的时间环境，由下式所决定。

$$t = \begin{cases} t + 1 & 0 \leq t \leq 60 \\ 15 & t > 60 \end{cases} \quad (4-2)$$

具体代码如下图所示。

```

#慢开始阶段-》对数
def slow_start(nowt):#0-15s
    return int(100+math.log(nowt,2)*200)
#print(slow_start(math.pow(2,15)))#->3100
#拥塞控制阶段-》线性
def curd(nowt):#16-60s
    return int(211.8*nowt-80)

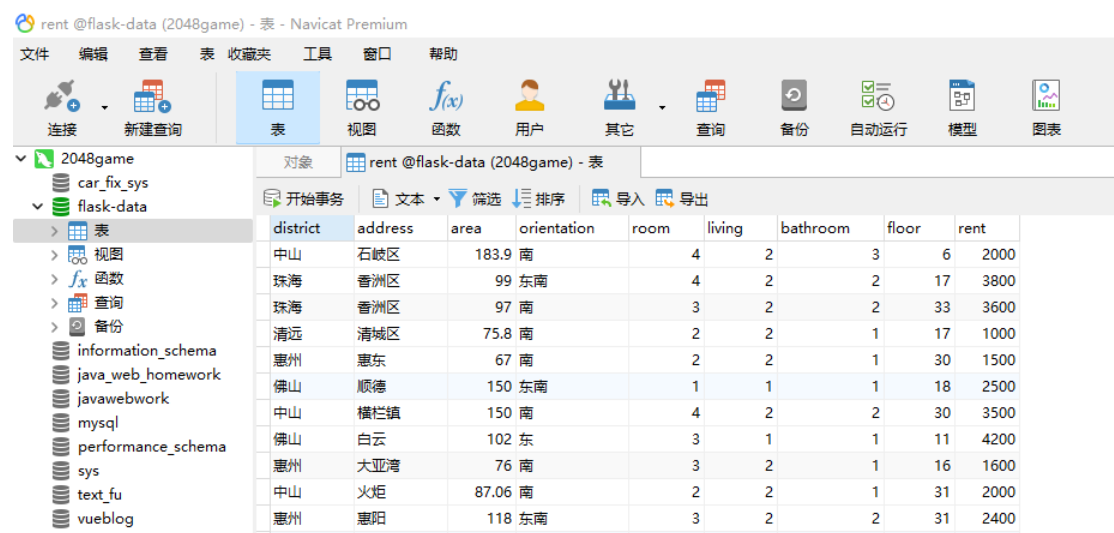
```

图 4-2 快启动-缓增长代码

## 4.2 Flask+ ECharts 可视化

### 4.2.1 数据导入 mysql

本文在 window 环境下通过 Navicat 软件直接将数据导入到 mysql 中。具体如下图所示。



district	address	area	orientation	room	living	bathroom	floor	rent
中山	石岐区	183.9	南	4	2	3	6	2000
珠海	香洲区	99	东南	4	2	2	17	3800
珠海	香洲区	97	南	3	2	2	33	3600
清远	清城区	75.8	南	2	2	1	17	1000
惠州	惠东	67	南	2	2	1	30	1500
佛山	顺德	150	东南	1	1	1	18	2500
中山	横栏镇	150	南	4	2	2	30	3500
佛山	白云	102	东	3	1	1	11	4200
惠州	大亚湾	76	南	3	2	1	16	1600
中山	火炬	87.06	南	2	2	1	31	2000
惠州	惠阳	118	东南	3	2	2	31	2400

图 4-3 数据导入 mysql

### 4.2.2 总体情况预览可视化分析

在本节中将房源数量、房源均面积、房间数量、租房均价 4 个指标可视化的展示在图上。让读者更容易了解整体租房信息。

#### ● Flask 代码

```

@app.route('/l1')
def get_l1_data():
    # 统计 area 的均值
    order = len(df)#订单数量
    profit = int(df['area'].mean())#租房面积均值
    customer = int(df['room'].sum())#房间数量
    ATV = int(df['rent'].mean())#租房均价
    #print(ATV)
    return jsonify({"order": order, "profit": profit, "customer": customer, "ATV": ATV})

```

图 4-4 分析 1Flask 代码

#### ● Web 代码

```
function get_r1_data() {
    $.ajax({
        url: "/1",
        success: function(data) {
            // data=JSON.parse(data)

            $("#order").html(data.order)
            $("#profit").html(data.profit)
            $("#customer").html(data.customer)
            $("#ATV").html(data.ATV)
        },
        error: function(xhr, type, errorThrown) {

        }
    })
}
```

图 4-5 分析 1web 代码

- 可视化结果展示



图 4-6 分析 1 可视化展示

#### 4.2.3 深圳、佛山租房各项指标对比

通过雷达图可以清楚的分析在相同的环境下，深圳与佛山两地二手房的价格差别，本次选取租房中的房间数量、平均面积、均价、楼层层数、客厅数量五个指标进行可视化。

- Flask 代码

```
@app.route('/l2')
def get_l2_data():
    # 过滤district为佛山和深圳的数据
    dfnew = df[df['district'].isin(['佛山', '深圳'])]

    # 分别计算两地的平均值
    mean_df = dfnew.groupby('district').mean().round(2)

    # 转换为列表输出
    mean_list = mean_df[['area', 'room', 'living', 'floor', 'rent']].values.tolist()
    columns_list = ['area平均值', 'room平均值', 'living平均值', 'floor平均值', 'rent平均值']
    return jsonify({"new_customer": mean_list[0], "old_customer": mean_list[1]})
```

图 4-7 分析 2Flask 代码

- Web 代码

```

series: [{
  name: '雷达图',
  type: 'radar',
  tooltip: {
    trigger: 'item'
  },
},
data: [{
  name: '佛山市',
  value: data['new_customer'],
  lineStyle: {
    normal: {
      color: '#03b48e',
      width: 2,
    }
  },
  areaStyle: {
    normal: {
      color: '#03b48e',
      opacity: .4
    }
  },
},
symbolSize: 0,

```

图 4-8 分析 2web 代码

- 可视化结果展示



图 4-9 分析 2 可视化展示

#### 4.2.4 不同朝向的租房房间数量、楼层可视化分析

通过对比不同朝向的租房房间数量、楼层等分析大致确定租房市场对于各个朝向的房源的容忍性。

- Flask 代码

```

@app.route('/l3')
def get_l3_data():#have_done
    # 分组并计算均值
    result = df.groupby(['orientation'])[['room', 'floor']].mean().reset_index().round(0)
    index = result["orientation"].tolist()
    old_customer = result["room"].tolist()
    new_customer = result["floor"].tolist()

    return jsonify({"index": index, "new_customer": new_customer, "old_customer": old_customer})

```

图 4-10 分析 3Flask 代码

- Web 代码

```

success: function (data) {
    // data=JSON.parse(data)

    var option = {
        legend: {
            icon: "circle",
            top: "0",
            width: "100%",
            right: "center",
            itemWidth: 12,
            itemHeight: 10,
            data: ['楼层数', '房间数量'],
            textStyle: {
                color: "rgba(255,255,255,.5)",
            },
        },
        tooltip: {
            trigger: 'axis',
            axisPointer: {
                type: 'shadow',
                lineStyle: {
                    color: '#dddcdb'
                }
            }
        }
    };

```

图 4-11 分析 3web 代码

- 可视化结果展示

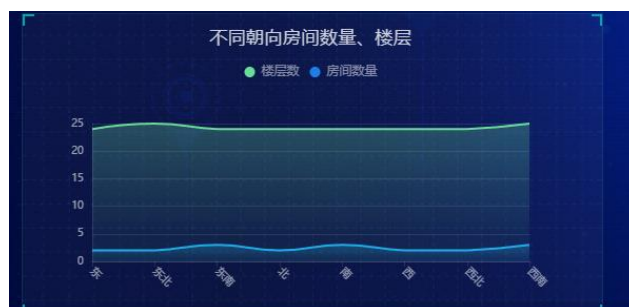


图 4-12 分析 3 可视化展示

#### 4.2.5 地区均价与面积对比分析

通过比较不同地区的租房价格与面积的对比，可以直观的分析出不同地区的租房的价格差异、比较性价比。

- Flask 代码

```

result = df.groupby('address')[['rent', 'area']].mean().reset_index().round(2)
#print(type(result)) # 输出DataFrame的类型
index = result['address'].tolist()[:10] # 获取address列并转换为列表
sales = result['rent'].tolist()[:10] # 获取rent列并转换为列表
profit = result['area'].tolist()[:10] # 获取area列并转换为列表
return jsonify({"index": index, "sales": sales, "profit": [0 for _ in range(10)], "profit_rate": profit})

```

图 4-13 分析 4Flask 代码

- Web 代码

```

series: [
    {
        name: '租金/元',
        type: 'bar',
        barGap: '-100%',
        barWidth: 10,
        itemStyle: {
            normal: {
                barBorderRadius: 5,
                color: new echarts.graphic.LinearGradient(
                    0, 0, 0, 1,
                    [
                        {offset: 0, color: 'rgba(0,254,204,0.4)'},
                        // {offset: 0.2, color: 'rgba(150,107,211,0.3)'},
                        {offset: 1, color: 'rgba(38,144,207,0.1)'}
                    ]
                )
            }
        }
    },
]

```

图 4-14 分析 4web 代码

- 可视化结果展示



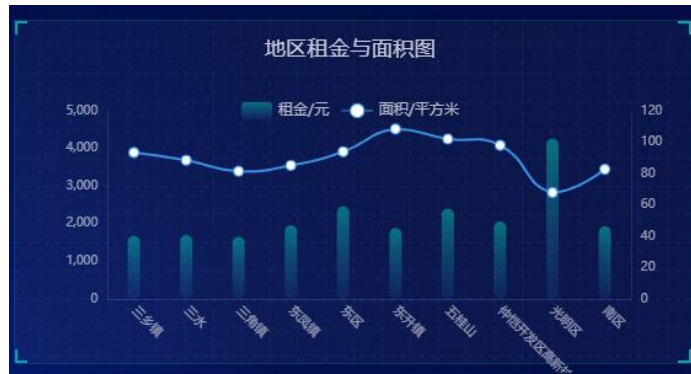


图 4-15 分析 4 可视化展示

#### 4.2.6 房源数量 TOP10 分析

通过聚合排序各个不同地区的房源数量，并将结果展示到页面中，可以清楚的分析到哪些地区的房源饱和、哪些的稀缺。

- Flask 代码

```
@app.route('/r21')
def get_r21_data():#have_done
    top_10 = df.groupby('address').size().reset_index(name='count').sort_values('count', ascending=False).head(10)
    #print(top_10)
    #product_df = get_product().sort_values('sales', ascending=False).head(10)
    return jsonify({"product": top_10['address'].tolist(), "sales": top_10['count'].tolist()[::-1]})
```

图 4-16 分析 5Flask 代码

- Web 代码

```
series: [
  {
    type: "bar",
    barWidth: "40%",
    barGap: 5,
    itemStyle: {
      normal: {
        color: new echarts.graphic.LinearGradient(1, 0, 0, 0,
          [
            {
              offset: 0,
              color: "#00fecc"
            },
            {
              offset: 0.8,
              color: "#2690cf"
            }
          ]
        ),
        false
      }
    }
  }
],
```

图 4-17 分析 5web 代码

- 可视化结果展示



图 4-18 分析 5 可视化展示



### 4.2.7 租房类型对比

针对不同价位的租房，本文将其分为 3 类，通过堆叠柱状图来分析与展示这 3 类房源的均面积、均价、数量的比较，结果清晰明了。

- Flask 代码

```
@app.route('/r3')#have_done
def get_r3_data():#客户类型对比
    # 将rent字段分成三个区间
    bins = [0, 5000, 10000, float('inf')]
    labels = ['平价租房', '轻奢房源', '至尊套房']
    df['rent_category'] = pd.cut(df['rent'], bins=bins, labels=labels)

    # 分组计算
    result = df.groupby('rent_category').agg({
        'rent': ['count', 'mean'],
        'area': 'mean'
    }).round(2)\
    # 修改列名
    result.columns = ['.'.join(col).strip() for col in result.columns.values]
    result['rent_mean'] = round(result['rent_mean'] / result['rent_mean'].sum(), 3)
    result['rent_count'] = round(result['rent_count'] / result['rent_count'].sum(), 3)
    result['area_mean'] = round(result['area_mean'] / result['area_mean'].sum(), 3)
    return jsonify({"data": result.values.tolist(),})
```

图 4-19 分析 6Flask 代码

- Web 代码

```
}},
grid: {
    left: '3%',
    right: '4%',
    bottom: '3%',
    containLabel: true
},
xAxis: {
    show: false
},
yAxis: {
    type: 'category',
    data: ['房租均价', '房源数量', '房源均面积'],
    axisTick: {show: false},
    axisLine: {show: false},
    axisLabel: {
        textStyle: {
            color: 'rgba(255,255,255,0.7)',
        },
    },
}
```

图 4-20 分析 6web 代码

- 可视化结果展示



图 4-21 分析 6 可视化展示

### 4.2.8 租房覆盖率分析展示

通过对比本次数据中出现的房源地区来分析该网站的房源覆盖的范围，通过

饼图将结果清晰展示出来。

● Flask 代码

```
@app.route('/c1')
def get_c1_data():#have_done
    #城区覆盖率
    # 统计 district 的数量
    district_count = df[district].nunique()
    achieving_rate = str(int(round(district_count / 21 * 100, 0))) + '%'
    #区县覆盖率
    address_count=df[address].nunique()
    year_achieving_rate = str(int(round(address_count / 121 * 100, 0))) + '%'

    return jsonify({"sales": district_count, "target": 21, "achieving_rate": achieving_rate,
                    "year_sales": address_count, "year_target": 121, "year_achieving_rate": year_achieving_rate})
```

图 4-22 分析 7Flask 代码

● Web 代码

```
URL: '/c1',
success: function (data) {
    $("#month_sales").html(data.sales)
    $("#year_sales").html(data.year_sales)

    var option1 = {
        title: {
            text: '市覆盖率',
            x: 'center',
            y: 'bottom',
            textStyle: {
                fontWeight: 'bold',
                color: 'rgb(255,255,255,0.7)',
                fontSize: '16',
            },
            padding:[0,0,15,0] // 上右下左
        },
        color: ['rgba(176, 212, 251, .1)'],
        series: [{
```

图 4-23 分析 7web 代码

● 可视化结果展示



图 4-24 分析 7 可视化展示

4.3 总体大屏可视化结果展示

具体可视化结果如下图所示，请注意，本文使用的大屏展示是实时变化的，但图片无法展示效果。



图 4-25 最终大屏可视化展示

## 5. Spark 机器学习

### 5.1 K-Means 聚类分析

#### 5.1.1 数据预处理

##### ● 导入所需包

```
scala> import org.apache.spark.ml.feature.{StandardScaler, VectorAssembler, StringIndexer}
import org.apache.spark.ml.feature.{StandardScaler, VectorAssembler, StringIndexer}

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

scala> import org.apache.spark.sql.{DataFrame, SparkSession}
import org.apache.spark.sql.{DataFrame, SparkSession}

scala> import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.clustering.KMeans

scala> import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer, OneHotEncoder}
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer, OneHotEncoder}

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.Pipeline

scala> import org.apache.spark.ml.feature.{OneHotEncoder, StringIndexer, VectorAssembler}
import org.apache.spark.ml.feature.{OneHotEncoder, StringIndexer, VectorAssembler}

scala> import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.linalg.Vector

scala> import spark.implicits._
import spark.implicits._

scala> import org.apache.spark.ml.tuning.ParamGridBuilder
import org.apache.spark.ml.tuning.ParamGridBuilder

scala> import org.apache.spark.ml.tuning.CrossValidator
import org.apache.spark.ml.tuning.CrossValidator

scala> import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
```

图 5-1

##### ● 读入数据

```
scala> val df = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("file:///usr/local/bigdata/dataset/rent.csv").toDF("district", "address", "area", "orientation", "room", "living", "bathroom", "floor", "rent", "null")
df: org.apache.spark.sql.DataFrame = [district: string, address: string ... 8 more fields]

scala>

scala> val colsToDrop = Seq("null")
colsToDrop: Seq[String] = List(null)

scala> val data = df.drop(colsToDrop:_)
data: org.apache.spark.sql.DataFrame = [district: string, address: string ... 7 more fields]

scala>
```

图 5-2

### 5.1.2 特征工程

#### ● 字符串编码和独热编码

```
scala> // 字符串编码和独热编码

scala> val districtIndexer = new StringIndexer().setInputCol("district").setOutputCol("districtIndex")
districtIndexer: org.apache.spark.ml.feature.StringIndexer = strIdx_2f13681cd29a

scala> val districtEncoded = new OneHotEncoder().setInputCol("districtIndex").setOutputCol("districtVec")
districtEncoded: org.apache.spark.ml.feature.OneHotEncoder = oneHot_5c3ec2361a4e

scala> val addressIndexer = new StringIndexer().setInputCol("address").setOutputCol("addressIndex")
addressIndexer: org.apache.spark.ml.feature.StringIndexer = strIdx_874c09baf9a0

scala> val addressEncoded = new OneHotEncoder().setInputCol("addressIndex").setOutputCol("addressVec")
addressEncoded: org.apache.spark.ml.feature.OneHotEncoder = oneHot_36df09831ce8

scala>
```

#### ● 将特征向量合并为一个向量列

```
scala> // 将特征向量合并为一个向量列

scala> val assembler = new VectorAssembler().setInputCols(Array("districtVec", "addressVec", "area", "room", "living", "bathroom", "floor")).setOutputCol("features")
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_b641ceeaeebc

scala>
```

图 5-3

#### ● 将 Pipeline 组合到一个流程中

```
scala> // 将Pipeline组合到一个流程中

scala> val pipeline = new Pipeline().setStages(Array(districtIndexer, addressIndexer, districtEncoded, addressEncoded, assembler))
pipeline: org.apache.spark.ml.Pipeline = pipeline_6a49e11bfd57

scala>
```

图 5-4

### 5.1.3 K-Means 模型聚类

#### ● 使用流程，处理数据并拟合 K-Means 模型

```
scala> val pipelineModel = pipeline.fit(data)
pipelineModel: org.apache.spark.ml.PipelineModel = pipeline_6a49e11bfd57

scala> val preprocessedDF = pipelineModel.transform(data)
preprocessedDF: org.apache.spark.sql.DataFrame = [district: string, address: string ... 12 more fields]

scala> val kmeans = new KMeans().setK(3).setFeaturesCol("features").setPredictionCol("prediction")
kmeans: org.apache.spark.ml.clustering.KMeans = kmeans_a2e5d2c950cc

scala> val model = kmeans.fit(preprocessedDF)
23/05/20 00:11:28 WARN KMeans: The input data is not directly cached, which may hurt performance if its parent RDDs are also uncached.
23/05/20 00:11:30 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
23/05/20 00:11:30 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
23/05/20 00:11:33 WARN KMeans: The input data was not directly cached, which may hurt performance if its parent RDDs are also uncached.
model: org.apache.spark.ml.clustering.KMeansModel = kmeans_a2e5d2c950cc

scala>
```

图 5-5 模型图示

#### ● 使用 K-Means 模型进行预测，并展示结果

图 5-6

[illegible]

图 5-7

```
scala> // 输出每个簇的大小
scala> prediction.groupBy("prediction").agg(avg("districtIndex"),avg("area"), avg("room"), avg("living"), avg("bathroom"), avg("floor"), avg("rent")).orderBy("prediction").show()
+-----+-----+-----+-----+-----+-----+-----+
|prediction|avg(districtIndex)|      avg(area)|      avg(room)|      avg(living)|      avg(bathroom)|      avg(floor)|      avg(rent)|
+-----+-----+-----+-----+-----+-----+-----+
|0|                0|0.2708333333333333|206.0772916666667|4.548011111111111|2.09375|2.979166666666667|17.34027777777778|13452.27083333333|
|1|                1|0.390388659792486|669.2602380952381|6.695242380952381|3.19864088117180221|0.6540831365935195|22.247366013049877|19106.39038865979|
|2|                2|2.347408081079547|110.01863972974643|3.144861634030142|1.925165648954138|1.687670529828821|24.67088823697545|2910.84832791955|
```

图 5-8

```
scala>
scala> //输出每个簇的数据统计信息
scala> predictions.groupBy("prediction").count.orderBy("prediction").show()
+-----+-----+
|prediction|count|
+-----+-----+
|          0|  288|
|          1| 4641|
|          2| 7697|
+-----+-----+
```

图 5-9

- WSSSE（集合内误差平方和）度量聚类的有效性

图 5-10

18



## 5.1.5 模型优化

```
scala> //平分k均值

scala> import org.apache.spark.ml.clustering.BisectingKMeans
import org.apache.spark.ml.clustering.BisectingKMeans

scala> val bkm = new BisectingKMeans().setK(3).setSeed(1)
bkm: org.apache.spark.ml.clustering.BisectingKMeans = bisecting-kmeans_4a9bf34c3cfb

scala> val bk_model = bkm.fit(preprocessedDF)
23/05/20 00:11:41 WARN BisectingKMeans: The input RDD 112 is not directly cached, which may hurt performance if its parent RDDs are also not cached.
bk_model: org.apache.spark.ml.clustering.BisectingKMeansModel = bisecting-kmeans_4a9bf34c3cfb

scala> val bk_predictions = bk_model.transform(preprocessedDF)
bk_predictions: org.apache.spark.sql.DataFrame = [district: string, address: string ... 13 more fields]

scala> bk_predictions.groupBy("prediction").count.orderBy("prediction").show()
+-----+-----+
|prediction|count|
+-----+-----+
|0|2949|
|1|5453|
|2|4224|
+-----+-----+

scala> bk_model.computeCost(preprocessedDF)
res6: Double = 1.2667938287165424E7
```

图 5-11

使用二分 k-Means 模型得到聚类 WSSSE 分数为 1.266>1.102，可得上述 K-Means 模型已为当前区间最优解。

## 5.1.6 结论

根据模型聚类所得结果，该数据集可分为三类：类型 0-面价最大，房间数最多，租金最高，多数位于中山、湛江等地区，适宜租来做为厂房/工作室等多人工作的场地；类型 1-面积最小，房间数最少，租金最少，但仍处于两千以上，推测是处于出行便利或市中心，多数处于深圳等发达城市，适宜独居的上班人员；类型 2-面积适中，租金适中，适合合租、一家人租住；

## 5.2 Lasso 回归模型预测租金

### 5.2.1 数据预处理

#### ● 导包

```
scala> import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.regression.LinearRegression

scala> import org.apache.spark.ml.regression.LinearRegressionModel
import org.apache.spark.ml.regression.LinearRegressionModel

scala> import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.VectorAssembler

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.evaluation.RegressionEvaluator

scala> import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}

scala> import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}
import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}

scala> import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.VectorAssembler
```

图 5-12

#### ● 字符串编码和独热编码

```
scala> // 字符串编码和独热编码

scala> val districtIndexer = new StringIndexer().setInputCol("district").setOutputCol("districtIndex")
districtIndexer: org.apache.spark.ml.feature.StringIndexer = strIdx_2f13681cd29a

scala> val districtEncoded = new OneHotEncoder().setInputCol("districtIndex").setOutputCol("districtVec")
districtEncoded: org.apache.spark.ml.feature.OneHotEncoder = oneHot_5c3ec2361a4e

scala> val addressIndexer = new StringIndexer().setInputCol("address").setOutputCol("addressIndex")
addressIndexer: org.apache.spark.ml.feature.StringIndexer = strIdx_874c09baf9a0

scala> val addressEncoded = new OneHotEncoder().setInputCol("addressIndex").setOutputCol("addressVec")
addressEncoded: org.apache.spark.ml.feature.OneHotEncoder = oneHot_36df09831ce8

scala>
```

图 5-13

## 5.2.2 特征工程

### ● 将特征向量合并为一个向量列

```
scala> // 将特征向量合并为一个向量列

scala> val assembler = new VectorAssembler().setInputCols(Array("districtVec", "addressVec", "area", "room", "living", "bathroom", "floor")).setOutputCol("features")
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_b641ceaeebc

scala>
```

图 5-14

### ● 将 Pipeline 组合到一个流程中

```
scala> // 将Pipeline组合到一个流程中

scala> val pipeline = new Pipeline().setStages(Array(districtIndexer, addressIndexer, districtEncoded, addressEncoded, assembler))
pipeline: org.apache.spark.ml.Pipeline = pipeline_6a49e11bfd57

scala>
```

图 5-15

## 5.2.3 构建模型

### ● 定义、拟合 Lasso 回归模型

```
scala> // 定义、拟合Lasso回归模型

scala> val lasso = new LinearRegression().setElasticNetParam(1).setRegParam(0.01).setLabelCol("rent")
lasso: org.apache.spark.ml.regression.LinearRegressionModel = linReg_893c20a50509

scala>

scala> val lassoModel = lasso.fit(preprocessedDF)
lassoModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_893c20a50509

scala>
```

图 5-16

## 5.2.4 模型评估

### ● Lasso 回归模型系数

```
scala> // 打印Lasso回归模型系数
Lasso Coefficients: [-789.3918077257117, -1265.462510335086, -487.7304082114919, -699.0043161984365, 159.04826646375787, -259.3870882475447, -613.7210013877834, -795.0832684065222, 650.4551165262494, -245.36834048858753, -873.873628755805, -218.1782018058011, -635.3527853234178, -464.8217925256166, -438.8404651454265, 7121.158831920505, 1475.211288559279, 1588.1601591173096, -882.8204180500671, -39.6519609711515, -384.42018534609807, -1633.3943832026001, 4886.748322492086, -372.483101394664, 8589.4807132121246, -661.376680324321, -1405.616143628659, -547.2818545209566, -643.813953119401, -383.3143714375508, -1424.452639311458, -83.0833775862322, 3678.082522037517, -1440.682954844353, -428.19214254430506, 5358.830600276543, -2124.1958550417266, -448.142699559587256, -998.7934706247808, -973.083815293017, -587.419031139273, -1013.853396253604, -1365.6749345839211, -1832.9655980297986, -1442.2236432602997, -2359.71055169077186, -948.6394415412018, 3268.315341760599, -504.9119888061931, -835.89825848219, 1857.659380284511, -524.507958260041, -464.9454042433091, -619.6562611509047, -658.2385622002809, -788.1162831144252, 987.9632816770654, 113.4429275535257, -933.5752796159195, -343.53566910921967, 328.35350494384977, -23.685347341756515, -1822.71736316099385, 1133.6043660607313, -538.2528205917655, 46.73497416025644, -417.54906996175765, -323.1143711101194, 383.3291240495014, 33.79940826211468]
```

图 5-17

### ● 查看预测结果

```
scala> lassoPredictions.show(10)
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|district|address|area|orientation|room|living|bathroom|floor|rent|districtIndex|addressIndex|districtVec|addressVec|features|prediction|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|珠海|珠海|香洲区|556.0|东南|5|2|0|36|556000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|24519.156564909755|
|珠海|珠海|香洲区|316.0|东南|4|2|4|42|400000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|15456.424720746662|
|珠海|珠海|香洲区|255.0|南北|4|2|3|17|300000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|11377.27696599665|
|珠海|珠海|香洲区|379.24|东南|6|2|3|24|290000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|16585.12787432364|
|珠海|珠海|香洲区|395.0|西|5|3|4|3|260000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|17089.647316574567|
|珠海|珠海|香洲区|299.43|东南|5|2|4|23|220000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|13622.288371863857|
|珠海|珠海|香洲区|211.0|东|4|2|3|44|200000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|10233.522125758465|
|珠海|珠海|香洲区|260.0|东|1|0|0|33|200000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|12900.630948990774|
|珠海|珠海|香洲区|252.0|东|5|3|4|41|200000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|12337.15226992496|
|珠海|珠海|香洲区|197.55|南|1|0|0|19|185000|4.0|2.0|[6,[4],[1.0]]|(59,[2],[1.0])|(70,[4,8,65,66,67,...]|9508.840896638454|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 10 rows
```

图 5-18

- 均方根误差(RMSE)评估

```
scala> // 均方根误差(RMSE)评估

scala> val lassoEvaluator = new RegressionEvaluator().setLabelCol("rent").setPredictionCol("prediction").setMetricName("rmse")
lassoEvaluator: org.apache.spark.ml.evaluation.RegressionEvaluator = regEval_e83aff8d056d

scala> val lassoPredictions = lassoModel.transform(preprocessedDF)
lassoPredictions: org.apache.spark.sql.DataFrame = [district: string, address: string ... 13 more fields]

scala> val lassoRmse = lassoEvaluator.evaluate(lassoPredictions)
lassoRmse: Double = 2712.8004081058803
```

图 5-19

## 5.2.5 参数调优

- 设置 Lasso 参数网格并定义评估器

```
scala> // 设置Lasso参数网格

scala> val lassoParamGrid = new ParamGridBuilder().addGrid(lasso.regParam, Array(0.01, 0.1, 1.0, 10.0)).build()
lassoParamGrid: Array[org.apache.spark.ml.param.ParamMap] =
Array({
  linReg_893c20a50509-regParam: 0.01
}, {
  linReg_893c20a50509-regParam: 0.1
}, {
  linReg_893c20a50509-regParam: 1.0
}, {
  linReg_893c20a50509-regParam: 10.0
})

scala>

scala> // 定义评估器

scala> val evaluator = new RegressionEvaluator().setLabelCol("rent").setPredictionCol("prediction").setMetricName("mae")
evaluator: org.apache.spark.ml.evaluation.RegressionEvaluator = regEval_a799fb8e1b3f

scala>

scala> // 使用交叉验证调优Lasso回归模型

scala> val lassoCV = new CrossValidator().setEstimator(lasso).setEvaluator(evaluator).setEstimatorParamMaps(lassoParamGrid).setNumFolds(5)
lassoCV: org.apache.spark.ml.tuning.CrossValidator = cv_c7af18dd70ab
```

图 5-20

- 使用交叉验证调优 Lasso 回归模型



```
scala> val lassoCV = new CrossValidator().setEstimator(lasso).setEvaluator(evaluator).setEstimatorParamM
aps(lassoParamGrid).setNumFolds(5)
lassoCV: org.apache.spark.ml.tuning.CrossValidator = cv_c7af18dd70ab

scala>

scala> val lassoCVModel = lassoCV.fit(preprocessedDF)
23/05/20 00:12:15 WARN Executor: 1 block locks were not released by TID = 752:
[rdd_347_0]
23/05/20 00:12:16 WARN Executor: 1 block locks were not released by TID = 758:
[rdd_347_0]
23/05/20 00:12:16 WARN Executor: 1 block locks were not released by TID = 764:
[rdd_347_0]
23/05/20 00:12:16 WARN Executor: 1 block locks were not released by TID = 770:
[rdd_347_0]
23/05/20 00:12:18 WARN Executor: 1 block locks were not released by TID = 780:
[rdd_461_0]
23/05/20 00:12:18 WARN Executor: 1 block locks were not released by TID = 786:
[rdd_461_0]
23/05/20 00:12:18 WARN Executor: 1 block locks were not released by TID = 792:
[rdd_461_0]
23/05/20 00:12:19 WARN Executor: 1 block locks were not released by TID = 798:
[rdd_461_0]
23/05/20 00:12:19 WARN Executor: 1 block locks were not released by TID = 808:
[rdd_575_0]
23/05/20 00:12:20 WARN Executor: 1 block locks were not released by TID = 814:
[rdd_575_0]
23/05/20 00:12:20 WARN Executor: 1 block locks were not released by TID = 820:
[rdd_575_0]
23/05/20 00:12:20 WARN Executor: 1 block locks were not released by TID = 826:
[rdd_575_0]
23/05/20 00:12:21 WARN Executor: 1 block locks were not released by TID = 836:
[rdd_689_0]
23/05/20 00:12:21 WARN Executor: 1 block locks were not released by TID = 842:
[rdd_689_0]
23/05/20 00:12:22 WARN Executor: 1 block locks were not released by TID = 848:
[rdd_689_0]
23/05/20 00:12:22 WARN Executor: 1 block locks were not released by TID = 854:
[rdd_689_0]
23/05/20 00:12:23 WARN Executor: 1 block locks were not released by TID = 864:
[rdd_803_0]
23/05/20 00:12:23 WARN Executor: 1 block locks were not released by TID = 870:
[rdd_803_0]
23/05/20 00:12:23 WARN Executor: 1 block locks were not released by TID = 876:
[rdd_803_0]
23/05/20 00:12:23 WARN Executor: 1 block locks were not released by TID = 882:
[rdd_803_0]
lassoCVModel: org.apache.spark.ml.tuning.CrossValidatorModel = cv_c7af18dd70ab
```

图 5-21

## ● 打印最优参数

```
scala> // 打印Lasso交叉验证模型最优参数和MAE

scala> println(s"Lasso CV Model - Best Parameter: ${lassoCVModel.bestModel.asInstanceOf[LinearRegression
Model].getRegParam}")
Lasso CV Model - Best Parameter: 10.0

scala> val lassoPredictions = lassoCVModel.transform(preprocessedDF)
lassoPredictions: org.apache.spark.sql.DataFrame = [district: string, address: string ... 13 more fields
]

scala> val lassoMAE = evaluator.evaluate(lassoPredictions)
lassoMAE: Double = 1074.4850092626211

scala> println(s"Lasso CV Model - MAE: $lassoMAE")
Lasso CV Model - MAE: 1074.4850092626211
```

图 5-22

结果可得最优 Parameter 参数为 10，RMSE 值也从 2710 降至 1074，证明模型调优有效。

## 6. 实时分析

### 6.1 出租房屋数量实时分析

#### 6.1.1 实验环境准备

##### ● python 工程目录结构

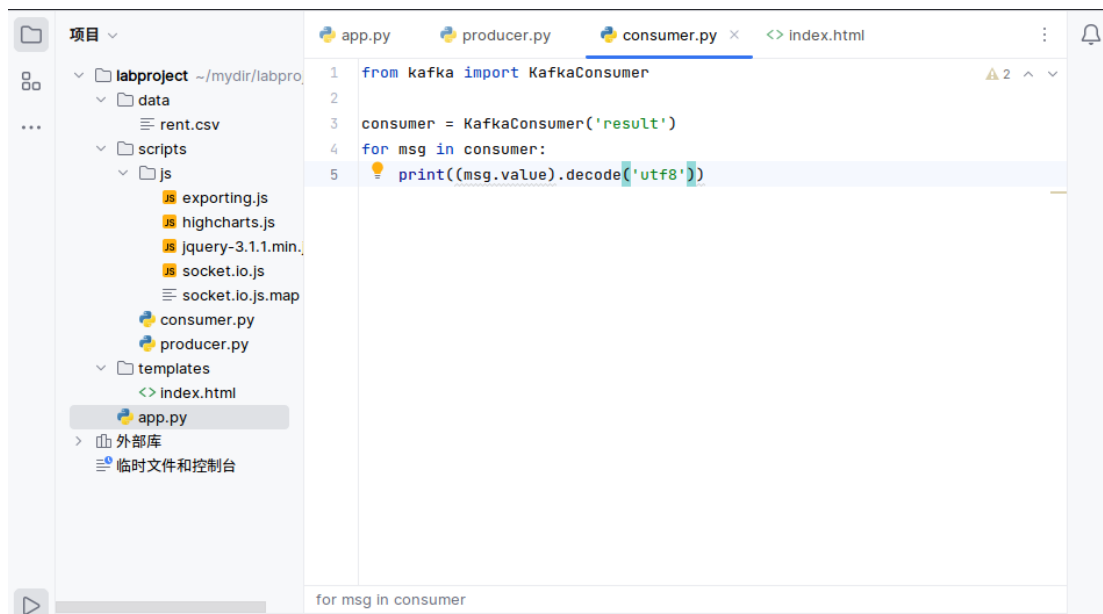


图 6-1

## 6.1.2 数据处理和 Python 操作 Kafka

### ● 启动 kafka 服务

```
hadoop@dblab-VirtualBox:~$ cd /usr/local/kafka
hadoop@dblab-VirtualBox:/usr/local/kafka$ bin/kafka-server-start.sh config/server.properties
[2023-05-30 14:05:00,177] INFO KafkaConfig values:
    advertised.host.name = null
    advertised.listeners = null
    advertised.port = null
    authorizer.class.name =
    auto.create.topics.enable = true
    auto.leader.rebalance.enable = true
    background.threads = 10
    broker.id = 0
    broker.id.generation.enable = true

hadoop@dblab-VirtualBox:~$ cd /usr/local/kafka
hadoop@dblab-VirtualBox:/usr/local/kafka$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
[2023-05-30 14:04:48,210] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-05-30 14:04:48,217] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2023-05-30 14:04:48,217] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2023-05-30 14:04:48,217] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
```

图 6-2

### ● 编写生产者文件

```

1 # coding: utf-8
2 import csv
3 import time
4 from kafka import KafkaProducer
5
6 # 实例化一个KafkaProducer示例, 用于向Kafka投递消息
7 producer = KafkaProducer(bootstrap_servers='localhost:9092')
8 # 打开数据文件
9 csvfile = open("../data/rent.csv", "r")
10 # 生成一个可用于读取csv文件的reader
11 reader = csv.reader(csvfile)
12
13 for line in reader:
14     rent = line[8] # 性别在每行日志代码的第9个元素
15     if rent == 'rent':
16         continue # 去除第一行表头
17     time.sleep(0.1) # 每隔0.1秒发送一行数据
18     # 发送数据, topic为'sex'
19     producer.send('rent', line[8].encode('utf8'))

```

图 6-3

- 编写消费者文件

```

1 from kafka import KafkaConsumer
2
3 consumer = KafkaConsumer('result')
4 for msg in consumer:
5     print((msg.value).decode('utf8'))

```

图 6-4

### 6.1.3 Spark Streaming 实时处理数据

- 编写日志文件

```

StreamingExample.scala

package org.apache.spark.examples.streaming
import org.apache.spark.internal.Logging
import org.apache.log4j.{Level, Logger}
/** Utility functions for Spark Streaming examples. */
object StreamingExamples extends Logging {
  /** Set reasonable logging levels for streaming if the user has not configured log4j. */
  def setStreamingLogLevels() {
    val log4jInitialized = Logger.getRootLogger.getAllAppenders.hasMoreElements
    if (!log4jInitialized) {
      // We first log something to initialize Spark's default logging, then we override the
      // logging level.
      logInfo("Setting log level to [WARN] for streaming example." +
        " To override add a custom log4j.properties to the classpath.")
      Logger.getRootLogger.setLevel(Level.WARN)
    }
  }
}

```

图 6-5

- 编写工程主文件

```

object KafkaWordCount {
  implicit val formats = DefaultFormats //数据格式化时需要
  def main(args: Array[String]): Unit={
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCount <zkQuorum> <group> <topics> <numThreads>")
      System.exit(1)
    }
    StreamingExamples.setStreamingLogLevels()
    /* 输入四个参数分别代表着
    * 1. zkQuorum 为zookeeper地址
    * 2. group为消费者所在的组
    * 3. topics该消费者所消费的topics
    * 4. numThreads开启消费topic线程的个数
    */
    val Array(zkQuorum, group, topics, numThreads) = args
    val sparkConf = new SparkConf().setAppName("KafkaWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(1))
    ssc.checkpoint(".") //这里表示把检查点文件写入分布式文件系统HDFS,所以要启动Hadoop
    //ssc.checkpoint("file:///usr/local/spark/mycode/kafka/checkpoint")
    // 将topics转换成topic->numThreads的哈希表
    val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap
    // 创建连接Kafka的消费者链接
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)
    val words = lines.flatMap(_.split(" "))//将输入的每行用空格分割成一个个word
    // 对每一秒的输入数据进行reduce,然后将reduce后的数据发送给Kafka
    val wordCounts = words.map(x => (x, 1L))
    .reduceByKeyAndWindow(_+_._2, Seconds(1), Seconds(1), 1).foreachRDD(rdd => {
      if(rdd.count != 0){
        val props = new HashMap[String, Object]()
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "127.0.0.1:9092")
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
          "org.apache.kafka.common.serialization.StringSerializer")
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
          "org.apache.kafka.common.serialization.StringSerializer")
        // 实例化一个Kafka生产者
        val producer = new KafkaProducer[String, String](props)
        // rdd.collect即将rdd中数据转化为数组,然后write函数将rdd内容转化为json格式
        val str = write(rdd.collect)
        // 封装成Kafka消息,topic为"result"
        val message = new ProducerRecord[String, String]("result", null, str)
        // 给Kafka发送消息
        producer.send(message)
      }
    })
    ssc.start()
    ssc.awaitTermination()
  }
}

```

图 6-6

### ● 编写 simple.sbt

```

name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.0"
libraryDependencies += "org.apache.spark" % "spark-streaming-kafka-0-8_2.11" % "2.1.0"
libraryDependencies += "org.json4s" %% "json4s-jackson" % "3.2.11"

```

图 6-7

### ● 打包上述程序



[illegible]

## ● 结果显示



本项目通过 EXCEL 对出租房屋数据集进行预处理，并使用 Spark SQL 进行了租金前十的市辖区、各市辖区出租房屋的最大面积、最小面积、平均面积等数据分析；使用 Flask+Echarts 对数据进行大屏可视化；使用 K-Means 聚类对出租房屋进行聚类分析，根据聚类结果可知，该数据集可分为三类；使用 Lasso 回归模型进行预测租金，该模型优化后 RMSE 值为 1074；

总体来说，这是一个非常实用和有用的项目，可以为房屋租赁市场的参与者提供很好的参考和指导。