我们需要对原数据中的时间戳进行处理，将其转换为标准的日期和时间格式，以便于进一步分析。在转换后的数据中，我们将提取出日期、时间、以及小时三个特征，并将其分别存入新的特征列中，以丰富数据的结构并提升分析的精度。

代码如下：

```
import datetime
import time

data['time_stamp'] = pd.to_datetime(data['time_stamp'], unit='s')
data['time_stamp']

# 从转换后的数据中分别提取:日期、时间、小时，组成新的列
data['date'] = data['time_stamp'].dt.date
data['time'] = data['time_stamp'].dt.time
data['hour'] = data['time_stamp'].dt.hour

# 调整数据集data的列顺序:将'date'、'time'、'hour'这三列数据调至'time_stamp'列后
columns = ['adgroup_id', 'cate_id', 'campaign_id', 'customer', 'brand', 'price',
           'userid', 'cms_segid', 'cms_group_id', 'final_gender_code', 'age_level',
           'shopping_level', 'occupation', 'new_user_class_level', 'pvalue_level',
           'time_stamp', 'date', 'time', 'hour', 'pid', 'nonclk', 'clk']
data = data[columns]
```

cms_segid	cms_group_id	final_gender_code	...	occupation	new_user_class_level	pvalue_level	time_stamp	date	time	hour	pid	nonclk	clk
35.0	4.0	2.0	...	0.0	2.0	2.0	2017-05-13 11:33:07	2017-05-13	11:33:07	11	430539_1007	1	0
5.0	2.0	2.0	...	1.0	2.0	2.0	2017-05-09 06:25:22	2017-05-09	06:25:22	6	430539_1007	1	0
5.0	2.0	2.0	...	1.0	1.0	2.0	2017-05-12 14:27:38	2017-05-12	14:27:38	14	430548_1007	1	0
5.0	2.0	2.0	...	1.0	2.0	1.0	2017-05-13 05:09:34	2017-05-13	05:09:34	5	430548_1007	1	0
21.0	3.0	2.0	...	0.0	2.0	3.0	2017-05-06 10:20:22	2017-05-06	10:20:22	10	430539_1007	1	0

图11　时间戳数据处理后表格

为了满足任务需求，我们需要从 `raw_sample.csv` 中提取用户 ID（`user`）、广告组 ID（`adgroup_id`）、未点击情况（`nonclk`）和点击情况（`clk`）这四个字段。提取后，将删除这些字段中包含空值的记录，并移除表头，以便对数据进行清理和标准化处理。

接下来，将三个数据表合并成一个完整的数据表。在合并过程中，采用 Map Side Join 方法：在 Map 阶段使用内存中的小表来关联大表，从而避免使用 Reducer，提高处理效率。为防止内存溢出，我们首先将 `rawlog` 与 `user_profile.csv` 进行连接，之后再与 `ad_feature.csv` 合并。通过这种合并方式，我们能够构建包含用户特征和广告特征的数据表，便于分析用户点击率的影响。代码实现如下：

```
public class MapSideMapper extends Mapper<LongWritable, Text, NullWritable, RawUserAd> {
	private Map<Integer, String> joinDataAd = new HashMap<>();

	@Override
	protected void setup(Context context) throws IOException, InterruptedException {
		Path[] path = DistributedCache.getLocalCacheFiles(context.getConfiguration());
		BufferedReader reader = new BufferedReader(new FileReader(path[0].toString()));
		// 存 ad
		String str = null;
		while ((str = reader.readLine()) != null) {
			String[] s = str.split(",");
			// 过滤表头
			if (s[0].startsWith("a"))
				continue;
			joinDataAd.put(Integer.valueOf(s[0]), str);
		}
	}

	@Override
	protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
		String[] values = value.toString().split("\t");
		RawUserAd rawUserAd = new RawUserAd();
		rawUserAd.setAdgrId(Integer.valueOf(values[0]));
		rawUserAd.setClk(Integer.valueOf(values[2]));
		rawUserAd.setNoclk(Integer.valueOf(values[1]));
		rawUserAd.setUserId(Integer.valueOf(values[3]));
		rawUserAd.setGender(Integer.valueOf(values[4]));
		rawUserAd.setAgeLevel(Integer.valueOf(values[5]));
		rawUserAd.setPvalueLevel(Integer.valueOf(values[6]));
		// 连接 ad
		int adId = Integer.parseInt(values[1]);
		/**
		 * (0) `adgroup_id` int,
```

```
             * (1) `cat_id` int,
             * (5) `price` double
             */
            if (!joinDataAd.containsKey(adId))
                return;
            String adFeature = joinDataAd.get(adId);
            String[] features = adFeature.split(",");
            if (features.length < 6)
                return;
            // 过滤 null
            if (features[0].equals("") || features[1].equals("") || features[5].equals(""))
                return;
            rawUserAd.setCatId(Integer.valueOf(features[1]));
            rawUserAd.setPrice(Double.valueOf(features[5]));
            context.write(NullWritable.get(), rawUserAd);
        }
}
```

此外，实验还引入了 Spark 来进行数据处理。为了进一步完善分析，从天池官网下载了一份名为 `behavior.log` 的用户行为数据，总数据量为 27G。这份数据详细记录了用户的各种行为，如点击、加购、收藏和购买等。实验中新增了一个名为 `btag` 的字段，用于明确标识这些行为类型。

这份丰富的数据为实验提供了更深入的用户行为视角，有助于精细化分析和挖掘用户行为特征，以更全面地理解用户在平台上的互动和参与度。

behavior.log的字段说明如下：

字段名	说明
user	用户ID（int）
timestamp	时间戳（bigint）
btag	用户行为（pv,buy,cart,fav等，String）
cate	类别ID（int）
brand	品牌（int）

对behavior.log数据进行处理，首先构造结构对象，代码如下：

```
schema = StructType([
    StructField("user", IntegerType()),
    StructField("timestamp", LongType()),
    StructField("btag", StringType()),
    StructField("cate", IntegerType()),
    StructField("brand", IntegerType())
])
behave_df = spark.read.csv("../data/behavior_log.csv", header=True, schema=schema)
behave_df = behave_df.withColumn("timestamp",
behave_df["timestamp"].cast(TimestampType()))
behave_df = behave_df.withColumn("date", behave_df["timestamp"].substr(0, 10))
behave_df = behave_df.withColumn("hour", behave_df["timestamp"].substr(12,
2).cast(IntegerType()))
behave_df = behave_df.drop("timestamp")
behave_df.show()
behave_df.printSchema()
behave_df.registerTempTable("all_data")
```

我们将根据给定的 `schema` 读取数据，并对时间戳字段进行转换，以便后续分析和使用。转换后的时间戳将被格式化为日期和小时的形式，从而更方便地进行时间相关的操作和分析。

接下来，我们将把处理后的数据集命名为 `"all_data"` 并注册为一个表，以便后续工作中能够快速访问和调用。

经过上述步骤处理后，最终得到的数据如下所示：

```
+------+----+-----+------+----------+----+
|  user|btag| cate| brand|      date|hour|
+------+----+-----+------+----------+----+
|558157|  pv| 6250| 91286|2017-05-03|   0|
|558157|  pv| 6250| 91286|2017-05-03|   0|
|558157|  pv| 6250| 91286|2017-05-03|   0|
|728690|  pv|11800| 62353|2017-05-03|  10|
|332634|  pv| 1101|365477|2017-05-03|  19|
|857237|  pv| 1043|110616|2017-05-03|  21|
|619381|  pv|  385|428950|2017-05-03|   9|
|467042|  pv| 8237|301299|2017-05-03|   8|
|467042|  pv| 8237|301299|2017-05-03|   8|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
|991528|  pv| 7270|274795|2017-05-03|  11|
```

图12　点击数据统计结果

浏览得到的数据后发现behavior.log中的日期字段存在问题，检查日期范围：

```
tmp = spark.sql("select max(date) ,min(date) from all_data")
tmp.show()
tmp.count()
```

```
+----------+----------+
| max(date)| min(date)|
+----------+----------+
|2036-07-11|1902-02-04|
+----------+----------+
```

图13　日期范围

在分析过程中，我们发现 `behavior.log` 数据中存在一些错误的日期信息，这可能会对后续的统计和分析工作造成偏差。为确保数据的准确性和一致性，我们需要对这些错误的日期进行修正。

经过清洗和修正后，我们将原始的 7 亿条数据筛选为时间窗口内的 289,915,052 条有效数据。随后，我们将 `user_profile.csv` 文件导入，并将处理后的数据保存为 `usr_behave.csv`，以供后续的用户行为分析、行为转化研究、用户价值评估等任务中使用。

userid	final_gender_code	age_level	pvalue_level	shopping_level	occupation	new_user_class_level	user	btag	cate	brand	date	hour
100010	2	2	null	3	1	4	100010	pv	6027	105661	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	buy	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	buy	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	buy	4505	342760	2017-05-09	17
100010	2	2	null	3	1	4	100010	pv	6735	224985	2017-05-12	12
100010	2	2	null	3	1	4	100010	pv	6735	224985	2017-05-12	12
100010	2	2	null	3	1	4	100010	pv	6735	224985	2017-05-12	12
100010	2	2	null	3	1	4	100010	pv	6735	224985	2017-05-12	12
100010	2	2	null	3	1	4	100010	pv	6735	224985	2017-05-12	12
100010	2	2	null	3	1	4	100010	pv	6735	224985	2017-05-12	12
100010	2	2	null	3	1	4	100010	pv	6735	224985	2017-05-12	12
100010	2	2	null	3	1	4	100010	pv	6735	235044	2017-05-12	12

图14　整理后数据

五、数据分析

5.1 点击事件分析

需要注意的是，由于物理设备算力的限制，数据分析实验仅在抽样数据上进行分析和可视化。

（1）点击量与点击率的分布

根据 adgroup_id 进行聚合，统计出每条广告的点击量和点击率。输出数据结构如下图所示，分别对应广告 ID、点击量、未点击量、点击率。

```
[hadoop@namenode1 ~]$ hdfs dfs -head /taobao/output_raw_cc/part-r-00000
1       0       1       0.0000
10      0       3       0.0000
100     0       4       0.0000
1000    0       1       0.0000
10000   1       131     0.0076
100000  1       7       0.1250
100001  0       2       0.0000
100002  0       22      0.0000
100003  0       2       0.0000
100004  0       1       0.0000
100005  0       4       0.0000
100006  1       22      0.0435
100007  0       1       0.0000
100008  0       36      0.0000
100009  0       1       0.0000
10001   0       2       0.0000
100010  0       1       0.0000
100011  1       7       0.1250
100012  0       5       0.0000
100013  1       3       0.2500
100014  0       1       0.0000
100015  0       2       0.0000
```

图15　点击数据统计结果

对上述数据进行抽样调查，根据广告被点击次数进行分层，分析点击量和点击率的分布。

代码如下：

```
data_adgroup = data.groupby('adgroup_id')['clk'].agg({
    '展示量': 'count',
    '点击量': sum,
    '点击率': np.mean
})
data_adgroup['点击量区间'] = data_adgroup.点击量.values   # 添加新列用来分区
```

```
间
data_adgroup.sort_values(by='点击量', ascending=False)

# 对所有广告点击量进行分段
bins = [0, 10, 20, 50, 100, 100000]
labels = ['<10', '10-20', '20-50', '50-100', '>=100']
data_adgroup['点击量区间'] = pd.cut(data_adgroup.点击量区间, bins=bins,
labels=labels, right=False)
data_adgroup.sort_values(by='点击量', ascending=False)
data_adgroup_clk = data_adgroup.groupby('点击量区间').agg({
        '点击量': ['count', sum],
        '展示量': sum
})
data_adgroup_clk.columns = ['广告量', '点击量', '展示量']
data_adgroup_clk['点击率'] = data_adgroup_clk.点击量.values / data_adgroup_clk.
展示量.values * 100
data_adgroup_clk
```

上述代码将数据中一部分数据抽样作为统计数据，在原数据基础上得到抽样数据分布，如图16所示：

点击量区间	广告量	点击量	展示量	点击率
<10	104662	55637	1288072	4.319401
10-20	1191	15835	260511	6.078438
20-50	488	14071	224591	6.265166
50-100	75	4890	75303	6.493765
>=100	17	2678	35332	7.579531

图16　抽样数据统计分布

在抽样数据基础上，实验通过绘制不同类型图像对淘宝广告数据的点击量和点击率分布进行可视化显示。

```
"""
利用堆积柱形图-折线图可视化分析各点击数区间的广告点击率情况
"""

from pyecharts import options as opts
```

```
from pyecharts.charts import Bar, Line, Scatter
from pyecharts.faker import Faker

name = data_adgroup_clk.index.tolist()
v0 = data_adgroup_clk.展示量.tolist()
v1 = (data_adgroup_clk.展示量  - data_adgroup_clk.点击量).tolist()
v2 = data_adgroup_clk.点击量.tolist()
v3 = data_adgroup_clk.点击率.tolist()
v3 = [round(i, 2) for i in v3]
v4 = data_adgroup_clk.广告量.tolist()
b = dict(zip(name, v4))
words = list(zip(name, v4))
bar = (
      Bar(init_opts=opts.InitOpts(width="640px", height="460px"))
      .add_xaxis(name)
      .add_yaxis("无点击量", v1, stack="stack1", category_gap="60%")
      .add_yaxis("点击量", v2, stack="stack1", category_gap="60%")
      .extend_axis(
            yaxis=opts.AxisOpts(axislabel_opts=opts.LabelOpts(formatter="{value} %"),
interval=1)
      )
      .set_series_opts(label_opts=opts.LabelOpts(position="right"))
      .set_global_opts(
            title_opts=opts.TitleOpts(title="各点击数区间的广告点击率情况", pos_left="30%"),
            yaxis_opts=opts.AxisOpts(axislabel_opts=opts.LabelOpts(formatter="{value}"),
max_=2500000),
            legend_opts=opts.LegendOpts(pos_top="6%", pos_left="50%")
      )
)
line = Line().add_xaxis(name).add_yaxis("点击率", v3, yaxis_index=1)
bar.overlap(line)
bar.render_notebook()
```

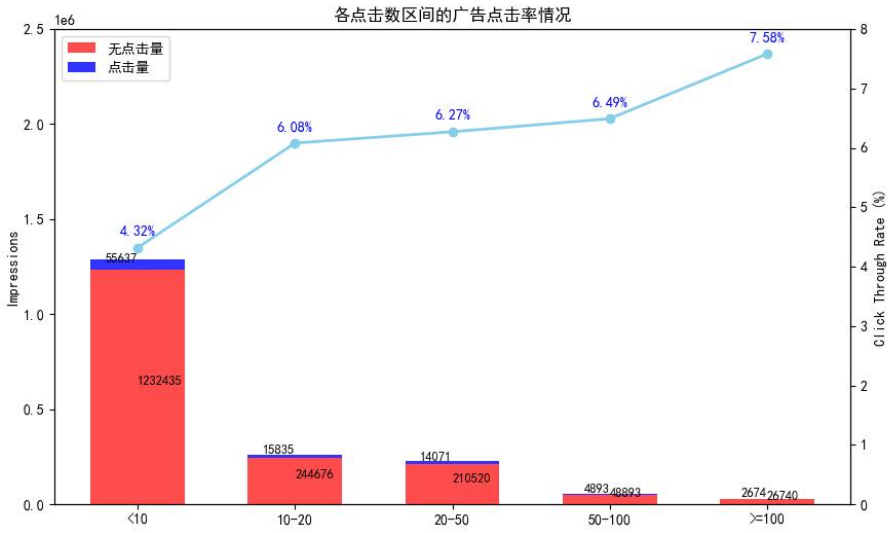

图17　点击数据统计分布

如图17所示，广告点击次数越高，其点击率也相应提升，表明广告的点击率与其价值呈正相关关系。然而，本次调查中点击次数超过100的广告占比极低，可能由于样本量较小，无法完全代表整体数据分布。

为更直观地展示广告的分布情况，可以绘制点击次数区间的占比图，显示不同点击数区间广告的占比分布。

代码如下：

```
"""
利用玫瑰图描绘点击用户所在的城市层次分布
"""
from pyecharts import options as opts
from pyecharts.charts import Pie

pie = (
    Pie(init_opts=opts.InitOpts(width="666px", height="365px"))
    .add("", words, center=["48%", "62%"], radius=["40%", "75%"], rosetype="radius")
    .set_global_opts(title_opts=opts.TitleOpts(title="各点击数区间的广告在整体广告中的占比分布", pos_right="22%"),
                     legend_opts=opts.LegendOpts(pos_top="6%", pos_right="16%"))
    .set_series_opts(label_opts=opts.LabelOpts(formatter="{b}: {c}({d}%)"))
)
pie.render_notebook()
```

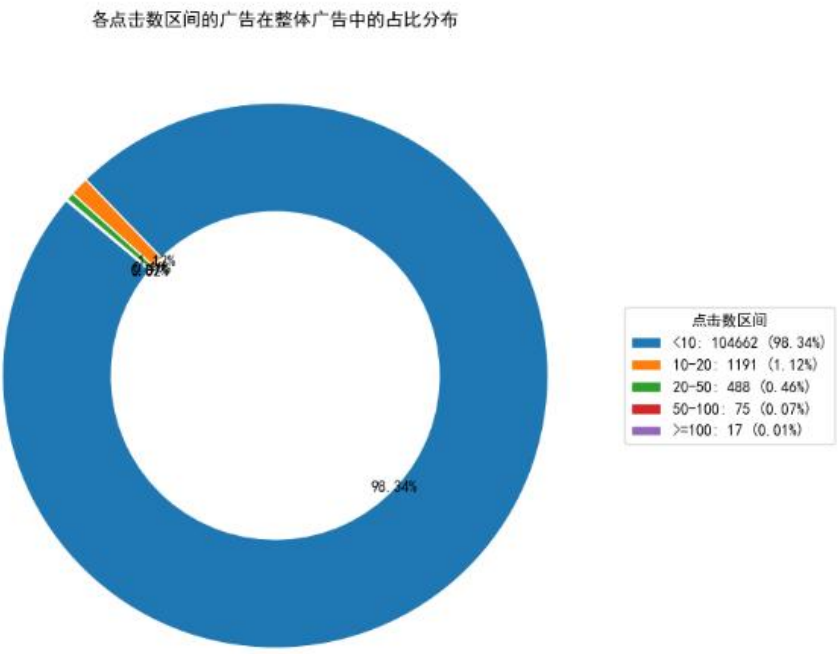

图18　点击数据统计分布

如图18所示，低价值广告在整体广告中占比高达98.34%，而高价值广告仅占0.08%。这种极大差异表明，广告价值测试阶段存在严重的费用浪费问题。

从数据分析来看，低价值广告的高占比反映出广告投放策略的改进空间。针对这些低价值广告，应采用更加精准的定向投放策略，以减少资源浪费。相对而言，高价值广告的占比虽然低，但往往带来更高的转化和回报，进一步突显了优化广告策略的必要性。对于高、低价值广告，应合理安排投放数量和策略，以实现更精准的投放效果和更高效益。

需要注意的是，这些分析基于抽样数据，尚不能完全代表整体数据，还需进一步抽样验证。

（2）整体广告点击占比

对数据进行分析，再点击数据基础上绘制图像进行可视化

代码如下：

```
counts = data.clk.value_counts()
ratio = data.clk.value_counts() / len(data) * 100
data_clk = pd.DataFrame({'Count': counts, 'Ratio(%)': ratio})

# 绘制饼图对广告整体点击情况进行可视化
from pyecharts import options as opts
from pyecharts.charts import Pie
b = data['clk']
b = b.value_counts()
b = dict(b)
b['无点击人数'] = b.pop(0)
b['点击人数'] = b.pop(1)
name = pd.DataFrame(b.keys())
value = pd.DataFrame(b.values())
name = name[0].tolist()
value = value[0].tolist()
words = list(zip(list(name), list(value)))
h = list(zip(list(b.keys()), list(b.values())))
pie = Pie(init_opts=opts.InitOpts(width="500px", height="320px"))
pie.add("", words)
pie.set_global_opts(title_opts=opts.TitleOpts(title="广告整体点击情况",
pos_left="40%"),
                    legend_opts=opts.LegendOpts(pos_top="20%",
pos_left="80%"))
pie.set_series_opts(label_opts=opts.LabelOpts(formatter="{b}:
{c}({d}%)"))
pie.render_notebook()
```

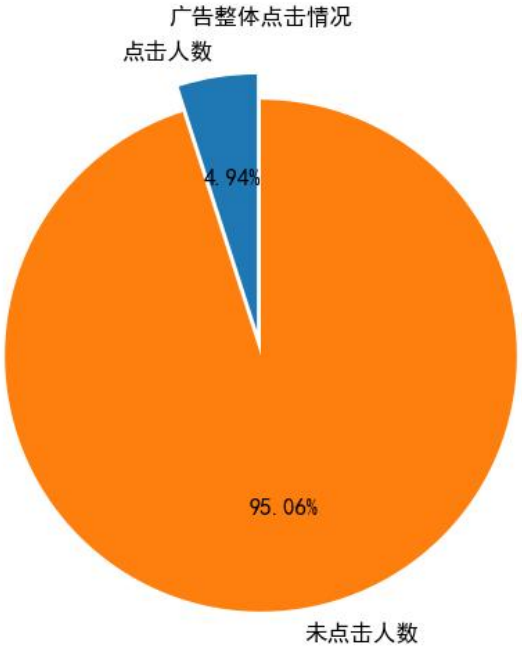

图19　点击数据统计结果

根据数据分析，广告的整体点击量仅占展示量的4.94%，即广告的点击率为4.94%，可见很多广告在展示过程中并没有吸引到用户或者说没有投放到有需要的用户群体。

（3）点击最高的 10 条广告

利用 Java 中 TreeMap 的结构解决 Top10 问题。

代码如下：

```
// Map 部分代码
public class ClickCountTopNMapper extends Mapper<Object, Text, NullWritable, Text> {
    private TreeMap<Integer, Text> clickTimesMap = new TreeMap<Integer, Text>(); //
TreeMap是有序KV集合
    // 广告id, 点击量, 未点击量, 点击率
    // 0, 1, 2, 3

    @Override
    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
        if (value == null) { // 空行不做处理
            return;
        }
```

```
            String[] strs = value.toString().split("\t");
            String adgrpId = strs[0];
            String clickCount = strs[1];
            if (adgrpId == null || clickCount == null) { // ID或访问次数为空，则不处理
                return;
            }
            // 将访问次数作为KEY、将行记录（广告ID+点击次数）作为VALUE，放入TreeMap
中按KEY自动升序排列
            clickTimesMap.put(Integer.parseInt(clickCount), new Text(adgrpId + "\t" +
clickCount));
            // 如果TreeMap中元素超过N个，将第一个（KEY最小的）元素删除
            if (clickTimesMap.size() > 10) {
                clickTimesMap.remove(clickTimesMap.firstKey());
            }
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            for (Text val : clickTimesMap.values()) {
                context.write(NullWritable.get(), val); // 在cleanup()中完成Map输出
            }
        }
}
```

代码采用Map&Reduce结构设计，上述代码为Map部分代码。

```
// Reduce 部分代码
public class ClickCountTopNReducer extends Reducer<NullWritable, Text, Text, Text> {
    private TreeMap<Integer, Text> clickTimesMap = new TreeMap<Integer, Text>();
    // 广告id 点击量

    @Override
    public void reduce(NullWritable key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        // 因为key为空，所以各个文件中的value都在同一个组内作为reduce的输入
        for (Text val : values) {
            String[] strs = val.toString().split("\t");
            clickTimesMap.put(Integer.parseInt(strs[1]), new Text(strs[0])); // 点击量，广告id
            if (clickTimesMap.size() > 10) {
                clickTimesMap.remove(clickTimesMap.firstKey());
            }
        }
    }
    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
        // 将TreeMap反序处理，降序输出top10
        NavigableMap<Integer, Text> inverseMap = clickTimesMap.descendingMap(); // 获得
TreeMap反序
        Iterator<Integer> iter = inverseMap.keySet().iterator();
        while (iter.hasNext()) {
            // 获取key
            Integer clickCount = iter.next();
```

```
                Text res = new Text(inverseMap.get(clickCount));
                context.write(res, new Text(clickCount.toString()));
            }
        }
}
```

上述代码为Reduce主要逻辑部分代码，对统计结果进行可视化 ，

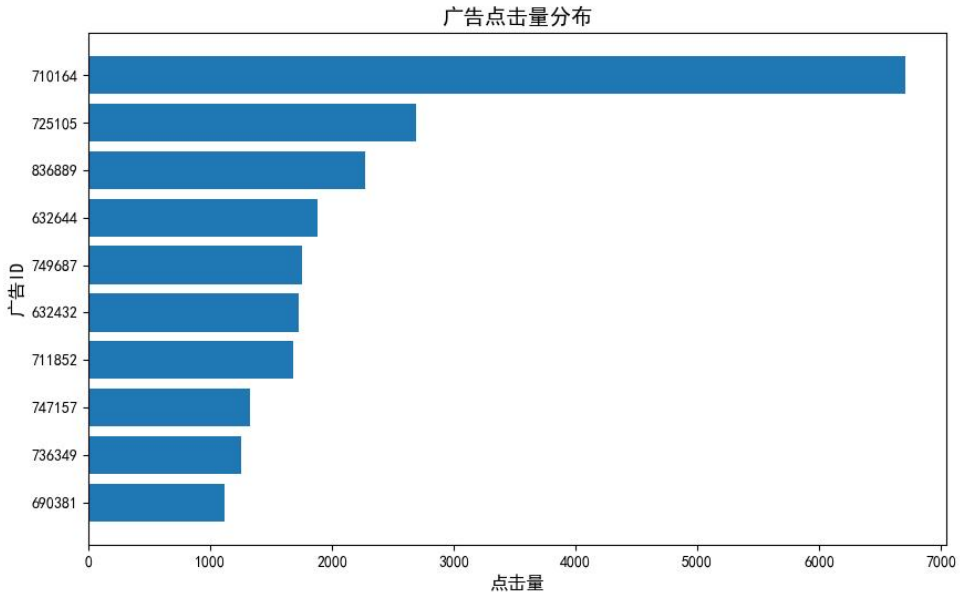

图20 点击量最高的10条统计

根据图20所示的数据，我们可以观察到最高点击量为6707的广告是一个相对突出的值。大部分广告的点击量在1000-2000之间。

另外，仿照点击量最高的数据分析中的思路，得到输出结果，对输出结果进行分析，得到点击率最高的10条统计（处理思路同样利用 Java 中 TreeMap分析数据），此处代码不再赘述，但是分析思路和过程与上述求取点击量前10的部分代码类似。