分析并预处理ad_feature数据集

In [ ]:
```
'''=======分析结果：只选取price作为特征数据=======
 |-- cateId: integer (nullable = true)  脱敏过的商品类目ID  个数： 6769
 |-- campaignId: integer (nullable = true)  脱敏过的广告计划ID  个数： 423436
 |-- customerId: integer (nullable = true)  脱敏过的广告主ID  个数： 255875
 |-- brandId: integer (nullable = true)  脱敏过的品牌ID  个数： 6769

以上四个特征均属于分类特征，但由于分类值个数均过于庞大，
如果去做热独编码处理，会导致数据过于稀疏
且当前我们缺少对这些特征更加具体的信息，（如商品类目具体信息、品牌具体信息等），
从而无法对这些特征的数据做聚类、降维处理 因此这里不选取它们作为特征

而只选取price作为特征数据，因为价格本身是一个统计类型连续数值型数据，
且能很好的体现广告的价值属性特征，通常也不需要做其他处理(离散化、归一化、标准化等)，
所以这里直接将当做特征数据来使用
'''
```

In [1]:
```
import os
# 配置pyspark和spark driver运行时 使用的python解释器
JAVA_HOME = '/root/bigdata/jdk'
PYSPARK_PYTHON = '/miniconda2/envs/py365/bin/python'
# 当存在多个版本时，不指定很可能会导致出错
os.environ['PYSPARK_PYTHON'] = PYSPARK_PYTHON
os.environ['PYSPARK_DRIVER_PYTHON'] = PYSPARK_PYTHON
os.environ['JAVA_HOME'] = JAVA_HOME
# 配置spark信息
from pyspark import SparkConf
from pyspark.sql import SparkSession

SPARK_APP_NAME = 'preprocessingAdFeature'
SPARK_URL = 'spark://192.168.58.100:7077'

conf = SparkConf()
config = (
    ('spark.app.name',SPARK_APP_NAME),# 设置启动的spark的app名称，没有提供，将随机产生
    ('spark.executor.memory','2g'), # 设置该app启动时占用的内存用量，Memory per Execut
    ('spark.master',SPARK_URL),
    ('spark.executor.cores','2') # 设置spark executor使用的CPU核心数，三台共6cores
)
# 查看更详细配置及说明：https://spark.apache.org/docs/latest/configuration.html
conf.setAll(config)

spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

1. 从hdfs中加载广告基本信息数据

In [2]:
```
df = spark.read.csv('/data/ad_feature.csv',header=True)
df.show()
```

```
+----------+-------+-----------+--------+------+-----+
|adgroup_id|cate_id|campaign_id|customer| brand|price|
+----------+-------+-----------+--------+------+-----+
|     63133|   6406|      83237|       1| 95471|170.0|
|    313401|   6406|      83237|       1| 87331|199.0|
|    248909|    392|      83237|       1| 32233| 38.0|
|    208458|    392|      83237|       1|174374|139.0|
|    110847|   7211|     135256|       2|145952|32.99|
|    607788|   6261|     387991|       6|207800|199.0|
|    375706|   4520|     387991|       6|  NULL| 99.0|
|     11115|   7213|     139747|       9|186847| 33.0|
|     24484|   7207|     139744|       9|186847| 19.0|
|     28589|   5953|     395195|      13|  NULL|428.0|
|     23236|   5953|     395195|      13|  NULL|368.0|
|    300556|   5953|     395195|      13|  NULL|639.0|
|     92560|   5953|     395195|      13|  NULL|368.0|
|    590965|   4284|      28145|      14|454237|249.0|
|    529913|   4284|      70206|      14|  NULL|249.0|
|    546930|   4284|      28145|      14|  NULL|249.0|
|    639794|   6261|      70206|      14| 37004| 89.9|
|    335413|   4284|      28145|      14|  NULL|249.0|
|    794890|   4284|      70206|      14|454237|249.0|
|    684020|   6261|      70206|      14| 37004| 99.0|
+----------+-------+-----------+--------+------+-----+
only showing top 20 rows
```

In [4]:
```
df.printSchema()
#看到str类型的数据就要想到下一步是将某些行转换成数值型的数据
```

```
root
 |-- adgroup_id: string (nullable = true)
 |-- cate_id: string (nullable = true)
 |-- campaign_id: string (nullable = true)
 |-- customer: string (nullable = true)
 |-- brand: string (nullable = true)
 |-- price: string (nullable = true)
```

In [8]:
```
# 可跳过，该cell属于延申学习：验证dropna()不能去掉str类型的缺失值

# pandas中选择某列使用df['某列的名字']，但是sparksql不能这样用，要使用sql语句，df.sele
# 可以使用df['某列的名字'].cast(某种数据类型如Longtype())
print('判断数据是否有空值：')
print('原始数据有%d行'%df.count())
# dropna()-某行数据有一个值是空值，就将该行删除，注意该方法不去掉str类型的null和NULL
print('去掉空值后数据有%d行'%df.dropna().count())
'''注意：看到dropna()前后行数没有变，但这不能说明没有null值，因为dropna()不能去掉str类
'''
```

```
判断数据是否有空值：
原始数据有846811行
去掉空值后数据有846811行
```

Out[8]:
```
'注意：看到dropna()前后行数没有变，但这不能说明没有null值，因为dropna()不能去掉str类型
的null值\n'
```

In [15]:

```
# 注意：由于本数据集中存在NULL字样的数据，无法直接设置schema，只能先将NULL类型的数据处
# 如果直接schema，下图会变成下下图：
# +----------+-------+-----------+--------+------+-----+
# |adgroup_id|cate_id|campaign_id|customer| brand|price|
# +----------+-------+-----------+--------+------+-----+
# |    375706|   4520|     387991|       6|  NULL| 99.0|

# +---------+------+----------+----------+-------+-----+
# |adgroupId|cateId|campaignId|customerId|brandId|price|
# +---------+------+----------+----------+-------+-----+
# |     null|  null|      null|      null|   null| null|

from pyspark.sql.types import StructType, StructField, IntegerType, FloatType
# 打印df结构信息
df.printSchema()
# 更改df表结构：更改列类型和列名称
ad_feature_df = df.\
    withColumn("adgroup_id", df.adgroup_id.cast(IntegerType())).withColumnRenamed("adg
    withColumn("cate_id", df.cate_id.cast(IntegerType())).withColumnRenamed("cate_id",
    withColumn("campaign_id", df.campaign_id.cast(IntegerType())).withColumnRenamed("c
    withColumn("customer", df.customer.cast(IntegerType())).withColumnRenamed("custome
    withColumn("brand", df.brand.cast(IntegerType())).withColumnRenamed("brand", "brar
    withColumn("price", df.price.cast(FloatType()))
ad_feature_df.printSchema()
ad_feature_df.show()
```

```
root
 |-- adgroup_id: string (nullable = true)
 |-- cate_id: string (nullable = true)
 |-- campaign_id: string (nullable = true)
 |-- customer: string (nullable = true)
 |-- brand: string (nullable = true)
 |-- price: string (nullable = true)

root
 |-- adgroupId: integer (nullable = true)
 |-- cateId: integer (nullable = true)
 |-- campaignId: integer (nullable = true)
 |-- customerId: integer (nullable = true)
 |-- brandId: integer (nullable = true)
 |-- price: float (nullable = true)

+---------+------+----------+----------+-------+-----+
|adgroupId|cateId|campaignId|customerId|brandId|price|
+---------+------+----------+----------+-------+-----+
|    63133|  6406|     83237|         1|  95471|170.0|
|   313401|  6406|     83237|         1|  87331|199.0|
|   248909|   392|     83237|         1|  32233| 38.0|
|   208458|   392|     83237|         1| 174374|139.0|
|   110847|  7211|    135256|         2| 145952|32.99|
|   607788|  6261|    387991|         6| 207800|199.0|
|   375706|  4520|    387991|         6|   null| 99.0|
|    11115|  7213|    139747|         9| 186847| 33.0|
|    24484|  7207|    139744|         9| 186847| 19.0|
|    28589|  5953|    395195|        13|   null|428.0|
|    23236|  5953|    395195|        13|   null|368.0|
|   300556|  5953|    395195|        13|   null|639.0|
```

```
|    92560|  5953|    395195|         13|    null|368.0|
|   590965|  4284|     28145|         14|  454237|249.0|
|   529913|  4284|     70206|         14|    null|249.0|
|   546930|  4284|     28145|         14|    null|249.0|
|   639794|  6261|     70206|         14|   37004| 89.9|
|   335413|  4284|     28145|         14|    null|249.0|
|   794890|  4284|     70206|         14|  454237|249.0|
|   684020|  6261|     70206|         14|   37004| 99.0|
+---------+------+----------+-----------+--------+-----+
only showing top 20 rows
```

In [20]:

```
#此处实验，说明dropna()的作用->某行数据有一个值是空值，就将该行删除
# # 查看是否有缺失值
# print('表中总共有%d行'%(ad_feature_df.count()))
# print('表中每列各有多少行：')
# print([str(c) + '列的行数' + str(ad_feature_df.select(c).count()) for c in ad_featur
# print([str(c) + '列的行数' + str(ad_feature_df.dropna().select(c).count()) for c in
# print([str(c) + '列的行数' + str(ad_feature_df.select(c).dropna().count()) for c in

# 没有缺失值
'''
表中总共有846811行
表中每列各有多少行：
['adgroupId列的行数846811', 'cateId列的行数846811', 'campaignId列的行数846811', 'custo
['adgroupId列的行数600481', 'cateId列的行数600481', 'campaignId列的行数600481', 'custo
['adgroupId列的行数846811', 'cateId列的行数846811', 'campaignId列的行数846811', 'custo
'''
```

...

In [16]:

```
# 查看是否有空值
# pandas中选择某列使用df['某列的名字']，但是sparksql不能这样用，要使用sql语句，df.sele
# 可以使用df['某列的名字'].cast(某种数据类型如Longtype())
print('判断数据是否有空值：')
print('原始数据有%d行'%ad_feature_df.count())
#dropna()-某行数据有一个值是空值，就将该行删除
print('去掉空值后数据有%d行'%ad_feature_df.dropna().count())
print('看看哪一列有空值：')
print([str(c) + '列的行数' + str(ad_feature_df.select(c).dropna().count()) for c in a
#有空值，brandId列有空值
```

```
判断数据是否有空值：
原始数据有846811行
去掉空值后数据有600481行
看看哪一列有空值：
['adgroupId列的行数846811', 'cateId列的行数846811', 'campaignId列的行数846811', 'custom
erId列的行数846811', 'brandId列的行数600481', 'price列的行数846811']
```

In [17]:
```
# 用-1替代空值
# fillna()替换两者的数值类型要相同
ad_feature_df = ad_feature_df.fillna(-1)
ad_feature_df.show()
```

```
+---------+------+----------+----------+-------+-----+
|adgroupId|cateId|campaignId|customerId|brandId|price|
+---------+------+----------+----------+-------+-----+
|    63133|  6406|     83237|         1|  95471|170.0|
|   313401|  6406|     83237|         1|  87331|199.0|
|   248909|   392|     83237|         1|  32233| 38.0|
|   208458|   392|     83237|         1| 174374|139.0|
|   110847|  7211|    135256|         2| 145952|32.99|
|   607788|  6261|    387991|         6| 207800|199.0|
|   375706|  4520|    387991|         6|     -1| 99.0|
|    11115|  7213|    139747|         9| 186847| 33.0|
|    24484|  7207|    139744|         9| 186847| 19.0|
|    28589|  5953|    395195|        13|     -1|428.0|
|    23236|  5953|    395195|        13|     -1|368.0|
|   300556|  5953|    395195|        13|     -1|639.0|
|    92560|  5953|    395195|        13|     -1|368.0|
|   590965|  4284|     28145|        14| 454237|249.0|
|   529913|  4284|     70206|        14|     -1|249.0|
|   546930|  4284|     28145|        14|     -1|249.0|
|   639794|  6261|     70206|        14|  37004| 89.9|
|   335413|  4284|     28145|        14|     -1|249.0|
|   794890|  4284|     70206|        14| 454237|249.0|
|   684020|  6261|     70206|        14|  37004| 99.0|
+---------+------+----------+----------+-------+-----+
only showing top 20 rows
```

In [21]:
```
# 查看各项特征的 种类数
print('adgroupId的个数：',ad_feature_df.groupby('adgroupId').count().count())
print('cateId的个数：',ad_feature_df.groupby('cateId').count().count())
print('campaignId的个数：',ad_feature_df.groupby('campaignId').count().count())
print('customerId的个数：',ad_feature_df.groupby('customerId').count().count())
print('brandId的个数：',ad_feature_df.groupby('brandId').count().count())
print('price的个数：',ad_feature_df.groupby('price').count().count())
ad_feature_df.where('price>10000').show()
```

```
adgroupId的个数： 846811
cateId的个数： 6769
campaignId的个数： 423436
customerId的个数： 255875
brandId的个数： 99815
price的个数： 14861
```

In [23]:
```
# 看看那些高价广告有多高价 >1w的广告
ad_feature_df.sort("price", ascending=False).show()
#sql中排序是orderBy，此处也可用
#ad_feature_df.where('price>10000').orderBy('price',ascending=False).show()
```

```
+---------+------+----------+----------+-------+-----------+
|adgroupId|cateId|campaignId|customerId|brandId|      price|
+---------+------+----------+----------+-------+-----------+
|   243384|   685|    218918|     31239| 278301|      1.0E8|
|   658722|  1093|    218101|    207754|     -1|      1.0E8|
|    31899|   685|    218918|     31239| 278301|      1.0E8|
|   468220|  1093|    270719|    207754|     -1|      1.0E8|
|   179746|  1093|    270027|    102509| 405447|      1.0E8|
|   554311|  1093|    266086|    207754|     -1|      1.0E8|
|   443295|  1093|     44251|    102509| 300681|      1.0E8|
|   513942|   745|      8401|     86243|     -1|8.8888888E7|
|   201060|   745|      8401|     86243|     -1|5.5555556E7|
|   289563|   685|     37665|    120847| 278301|      1.5E7|
|    35156|   527|    417722|     72273| 278301|      1.0E7|
|    33756|   527|    416333|     70894|     -1|  9900000.0|
|   335495|   739|    170121|    148946| 326126|  9600000.0|
|   218306|   206|    162394|      4339| 221720|  8888888.0|
|   213567|  7213|    239302|    205612| 406125|  5888888.0|
|   375920|   527|    217512|    148946| 326126|  4760000.0|
|   262215|   527|    132721|     11947| 417898|  3980000.0|
|   154623|   739|    170121|    148946| 326126|  3900000.0|
|   152414|   739|    170121|    148946| 326126|  3900000.0|
|   448651|   527|    422260|     41289| 209959|  3800000.0|
+---------+------+----------+----------+-------+-----------+
only showing top 20 rows
```

In [29]:
```
# 查看价格大于1w的广告有多少条
ad_feature_df.where('price>10000').count()
```

Out[29]: 6527

In [30]:
```
ad_feature_df.printSchema()
```

```
root
 |-- adgroupId: integer (nullable = true)
 |-- cateId: integer (nullable = true)
 |-- campaignId: integer (nullable = true)
 |-- customerId: integer (nullable = true)
 |-- brandId: integer (nullable = true)
 |-- price: float (nullable = false)
```

In [ ]:
```
'''
 |-- cateId: integer (nullable = true)  脱敏过的商品类目ID  个数： 6769
 |-- campaignId: integer (nullable = true)  脱敏过的广告计划ID  个数： 423436
 |-- customerId: integer (nullable = true)  脱敏过的广告主ID  个数： 255875
 |-- brandId: integer (nullable = true)  脱敏过的品牌ID  个数： 6769

以上四个特征均属于分类特征，但由于分类值个数均过于庞大，
如果去做热独编码处理，会导致数据过于稀疏
且当前我们缺少对这些特征更加具体的信息，（如商品类目具体信息、品牌具体信息等），
从而无法对这些特征的数据做聚类、降维处理 因此这里不选取它们作为特征

而只选取price作为特征数据，因为价格本身是一个统计类型连续数值型数据，
且能很好的体现广告的价值属性特征，通常也不需要做其他处理(离散化、归一化、标准化等)，
所以这里直接将当做特征数据来使用
'''
```

In [ ]:
```

```

In [ ]:
```

```