探索Ebay汽车销售数据

8207200203 计算机2006班 翟衍博

1.项目介绍

项目介绍：我们将使用来自eBay Kleinanzeigen（德国eBay网站的分类部分）的二手车数据集。该项目的目的是清理数据并分析包括的二手车清单。
数据集介绍：
dateCrawled— 首次抓取该广告的时间。 所有字段值均从该日期开始。
name— 汽车名称
seller— 卖方是私人还是经销商
offerType— 清单类型
price— 广告上出售汽车的价格
abtest— 清单是否包含在A / B测试中
vehicleType— 车辆类型
yearOfRegistration— 汽车首次注册的年份
gearbox— 变速箱类型。
powerPS— PS中汽车的动力。
model— 汽车型号名称
km— 汽车行驶了多少公里
monthOfRegistration— 汽车首次注册的月份
fuelType— 汽车使用哪种燃料
brand——汽车的品牌
notRepairedDamage——如果汽车有尚未修理的损坏
dateCreated——创建eBay清单的日期
nrOfPictures——广告中的图片数量
postalCode— 车辆位置的邮政编码
lastSeenOnline— 抓取工具上次在线看到此广告的时间

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
autos=pd.read_csv("autos.csv",encoding="Latin-1")
#指定encoding，默认utf-8会报错
```

In [3]:

```
autos.info()
print('------------------------------------------------------------\n')
print(autos.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   dateCrawled          50000 non-null  object
 1   name                 50000 non-null  object
 2   seller               50000 non-null  object
 3   offerType            50000 non-null  object
 4   price                50000 non-null  object
 5   abtest               50000 non-null  object
 6   vehicleType          44905 non-null  object
 7   yearOfRegistration   50000 non-null  int64 
 8   gearbox              47320 non-null  object
 9   powerPS              50000 non-null  int64 
 10  model                47242 non-null  object
 11  odometer             50000 non-null  object
 12  monthOfRegistration  50000 non-null  int64 
 13  fuelType             45518 non-null  object
 14  brand                50000 non-null  object
 15  notRepairedDamage    40171 non-null  object
 16  dateCreated          50000 non-null  object
 17  nrOfPictures         50000 non-null  int64 
 18  postalCode           50000 non-null  int64 
 19  lastSeen             50000 non-null  object
dtypes: int64(5), object(15)
memory usage: 7.6+ MB
------------------------------------------------------------


           dateCrawled                                              name  \
0  2016-03-26 17:47:46                    Peugeot_807_160_NAVTECH_ON_BOARD
1  2016-04-04 13:38:56          BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik
2  2016-03-26 18:57:24                       Volkswagen_Golf_1.6_United
3  2016-03-12 16:58:10  Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan...
4  2016-04-01 14:38:50  Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg...

   seller offerType   price   abtest vehicleType  yearOfRegistration  \
0  privat   Angebot  $5,000  control         bus                2004   
1  privat   Angebot  $8,500  control   limousine                1997   
2  privat   Angebot  $8,990     test   limousine                2009   
3  privat   Angebot  $4,350  control  kleinwagen                2007   
4  privat   Angebot  $1,350     test       kombi                2003   

     gearbox  powerPS   model   odometer  monthOfRegistration fuelType  \
0    manuell      158  andere  150,000km                    3      lpg   
1  automatik      286     7er  150,000km                    6   benzin   
2    manuell      102    golf   70,000km                    7   benzin   
3  automatik       71  fortwo   70,000km                    6   benzin   
4    manuell        0   focus  150,000km                    7   benzin   

        brand notRepairedDamage          dateCreated  nrOfPictures  \
0     peugeot              nein  2016-03-26 00:00:00             0   
1         bmw              nein  2016-04-04 00:00:00             0   
2  volkswagen              nein  2016-03-26 00:00:00             0   
3       smart              nein  2016-03-12 00:00:00             0   
```

```
4        ford              nein  2016-04-01 00:00:00            0

   postalCode             lastSeen
0       79588  2016-04-06 06:45:54
1       71034  2016-04-06 14:45:08
2       35394  2016-04-06 20:15:37
3       33729  2016-03-15 03:16:28
4       39218  2016-04-01 14:38:50
```

观察到的结果:

1. 数据集包括20列

2. 有些列中存在空值

3. 大多数列为object类，即为字符串类型，后续可能要将其转为数值类型进行处理

2.清洗列名

In [4]:

```
#打印列名
print(autos.columns)
```

```
Index(['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest',
       'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'model',
       'odometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage', 'dateCreated', 'nrOfPictures', 'postalCode',
       'lastSeen'],
      dtype='object')
```

In [5]:

```
#将列名从camelcase转换为snakecase
autos.columns = ['date_crawled', 'name', 'seller', 'offer_type', 'price', 'ab_test',
       'vehicle_type', 'registration_year', 'gearbox', 'power_ps', 'model',
       'odometer', 'registration_month', 'fuel_type', 'brand',
       'unrepaired_damage', 'ad_created', 'num_photos', 'postal_code',
       'last_seen']
```

In [6]:

```
print(autos.head())
```

```
          date_crawled                                               name  \
0  2016-03-26 17:47:46                    Peugeot_807_160_NAVTECH_ON_BOARD
1  2016-04-04 13:38:56          BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik
2  2016-03-26 18:57:24                       Volkswagen_Golf_1.6_United
3  2016-03-12 16:58:10  Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan...
4  2016-04-01 14:38:50  Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg...

   seller offer_type   price  ab_test vehicle_type  registration_year  \
0  privat    Angebot  $5,000  control          bus               2004
1  privat    Angebot  $8,500  control    limousine               1997
2  privat    Angebot  $8,990     test    limousine               2009
3  privat    Angebot  $4,350  control   kleinwagen               2007
4  privat    Angebot  $1,350     test        kombi               2003

     gearbox  power_ps   model   odometer  registration_month fuel_type  \
0    manuell       158  andere  150,000km                   3       lpg
1  automatik       286     7er  150,000km                   6    benzin
2    manuell       102    golf   70,000km                   7    benzin
3  automatik        71  fortwo   70,000km                   6    benzin
4    manuell         0   focus  150,000km                   7    benzin

        brand unrepaired_damage          ad_created  num_photos  postal_code  \
0     peugeot              nein  2016-03-26 00:00:00           0        79588
1         bmw              nein  2016-04-04 00:00:00           0        71034
2  volkswagen              nein  2016-03-26 00:00:00           0        35394
3       smart              nein  2016-03-12 00:00:00           0        33729
4        ford              nein  2016-04-01 00:00:00           0        39218

            last_seen
0  2016-04-06 06:45:54
1  2016-04-06 14:45:08
2  2016-04-06 20:15:37
3  2016-03-15 03:16:28
4  2016-04-01 14:38:50
```

我们将列名从camelcase转换为snakecase，目的是便于后续将列名进行改写，规范化列名，提高描述性。

3.初步探索与清洗

In [7]:

```
autos.describe(include='all')
```

Out[7]:

	date_crawled	name	seller	offer_type	price	ab_test	vehicle_type	registration_
count	50000	50000	50000	50000	50000	50000	44905	50000.00
unique	48213	38754	2	2	2357	2	8	
top	2016-04-02 11:37:04	Ford_Fiesta	privat	Angebot	$0	test	limousine	
freq	3	78	49999	49999	1421	25756	12859	
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2005.07
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	105.71
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1000.00
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1999.00
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2003.00
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2008.00
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9999.00

注意： 几乎所有具有一个值的列都将被删除

任何列都需要更多调查。

存储为文本的任何数字数据示例都需要清洗。

In [8]:

```
autos["num_photos"].value_counts()
```

Out[8]:

```
0    50000
Name: num_photos, dtype: int64
```

In [9]:

```
#丢掉数据
autos = autos.drop(["num_photos", "seller", "offer_type"], axis=1)
```

In [10]:

```
#转化数据
autos["price"] = (autos["price"]
                          .str.replace("$","")
                          .str.replace(",","")
                          .astype(int)
                          )
autos["price"].head()
```

```
C:\Users\Zhai yanbo\AppData\Local\Temp\ipykernel_107644\144416225.py:2: FutureWarnin
g: The default value of regex will change from True to False in a future version. In
addition, single character regular expressions will *not* be treated as literal stri
ngs when regex=True.
  autos["price"] = (autos["price"]
```

Out[10]:

```
0    5000
1    8500
2    8990
3    4350
4    1350
Name: price, dtype: int32
```

In [11]:

```
autos["odometer"] = (autos["odometer"]
                           .str.replace("km","")
                           .str.replace(",","")
                           .astype(int)
                           )
#重命名
autos.rename({"odometer": "odometer_km"}, axis=1, inplace=True)
autos["odometer_km"].head()
```

Out[11]:

```
0    150000
1    150000
2     70000
3     70000
4    150000
Name: odometer_km, dtype: int32
```

4.探索Odometer和Price列

In [12]:

```
autos["odometer_km"].value_counts()
```

Out[12]:

```
150000    32424
125000     5170
100000     2169
90000      1757
80000      1436
70000      1230
60000      1164
50000      1027
5000        967
40000       819
30000       789
20000       784
10000       264
Name: odometer_km, dtype: int64
```

我们可以看到，这里面的"odometer_km"值是大约的值，而且里程数越大的数量越大。

In [13]:

```
autos["price"].value_counts().head(20)
print(autos["price"].unique().shape)
print(autos["price"].describe())
```

```
(2357,)
count    5.000000e+04
mean     9.840044e+03
std      4.811044e+05
min      0.000000e+00
25%      1.100000e+03
50%      2.950000e+03
75%      7.200000e+03
max      1.000000e+08
Name: price, dtype: float64
```

与“odometer_km”类似，这里的“price”也是大约的值，而且是价格越低数量越多。

In [14]:

```
autos["price"].value_counts().sort_index(ascending=False).head(20)
```

Out[14]:

```
99999999    1
27322222    1
12345678    3
11111111    2
10000000    1
3890000     1
1300000     1
1234566     1
999999      2
999990      1
350000      1
345000      1
299000      1
295000      1
265000      1
259000      1
250000      1
220000      1
198000      1
197000      1
Name: price, dtype: int64
```

In [15]:

```
autos["price"].value_counts().sort_index(ascending=True).head(20)
```

Out[15]:

```
0     1421
1      156
2        3
3        1
5        2
8        1
9        1
10       7
11       2
12       3
13       2
14       1
15       2
17       3
18       1
20       4
25       5
29       1
30       7
35       1
Name: price, dtype: int64
```

In [16]:

```
autos = autos[autos["price"].between(1,351000)]
autos["price"].describe()
```

Out[16]:

```
count     48565.000000
mean       5888.935591
std        9059.854754
min           1.000000
25%        1200.000000
50%        3000.000000
75%        7490.000000
max      350000.000000
Name: price, dtype: float64
```

5.探索date列

In [17]:

```
autos[['date_crawled','ad_created','last_seen']][0:5]
```

Out[17]:

	date_crawled	ad_created	last_seen
0	2016-03-26 17:47:46	2016-03-26 00:00:00	2016-04-06 06:45:54
1	2016-04-04 13:38:56	2016-04-04 00:00:00	2016-04-06 14:45:08
2	2016-03-26 18:57:24	2016-03-26 00:00:00	2016-04-06 20:15:37
3	2016-03-12 16:58:10	2016-03-12 00:00:00	2016-03-15 03:16:28
4	2016-04-01 14:38:50	2016-04-01 00:00:00	2016-04-01 14:38:50

In [18]:

```
(autos["date_crawled"]
        .str[:10]
        .value_counts(normalize=True, dropna=False)
        .sort_index()
        )
```

Out[18]:

```
2016-03-05    0.025327
2016-03-06    0.014043
2016-03-07    0.036014
2016-03-08    0.033296
2016-03-09    0.033090
2016-03-10    0.032184
2016-03-11    0.032575
2016-03-12    0.036920
2016-03-13    0.015670
2016-03-14    0.036549
2016-03-15    0.034284
2016-03-16    0.029610
2016-03-17    0.031628
2016-03-18    0.012911
2016-03-19    0.034778
2016-03-20    0.037887
2016-03-21    0.037373
2016-03-22    0.032987
2016-03-23    0.032225
2016-03-24    0.029342
2016-03-25    0.031607
2016-03-26    0.032204
2016-03-27    0.031092
2016-03-28    0.034860
2016-03-29    0.034099
2016-03-30    0.033687
2016-03-31    0.031834
2016-04-01    0.033687
2016-04-02    0.035478
2016-04-03    0.038608
2016-04-04    0.036487
2016-04-05    0.013096
2016-04-06    0.003171
2016-04-07    0.001400
Name: date_crawled, dtype: float64
```

In [19]:

```
(autos["date_crawled"]
        .str[:10]
        .value_counts(normalize=True, dropna=False)
        .sort_values()
        )
```

Out[19]:

```
2016-04-07    0.001400
2016-04-06    0.003171
2016-03-18    0.012911
2016-04-05    0.013096
2016-03-06    0.014043
2016-03-13    0.015670
2016-03-05    0.025327
2016-03-24    0.029342
2016-03-16    0.029610
2016-03-27    0.031092
2016-03-25    0.031607
2016-03-17    0.031628
2016-03-31    0.031834
2016-03-10    0.032184
2016-03-26    0.032204
2016-03-23    0.032225
2016-03-11    0.032575
2016-03-22    0.032987
2016-03-09    0.033090
2016-03-08    0.033296
2016-04-01    0.033687
2016-03-30    0.033687
2016-03-29    0.034099
2016-03-15    0.034284
2016-03-19    0.034778
2016-03-28    0.034860
2016-04-02    0.035478
2016-03-07    0.036014
2016-04-04    0.036487
2016-03-14    0.036549
2016-03-12    0.036920
2016-03-21    0.037373
2016-03-20    0.037887
2016-04-03    0.038608
Name: date_crawled, dtype: float64
```

In [20]:

```
(autos["last_seen"]
        .str[:10]
        .value_counts(normalize=True, dropna=False)
        .sort_index()
        )
```

Out[20]:

```
2016-03-05    0.001071
2016-03-06    0.004324
2016-03-07    0.005395
2016-03-08    0.007413
2016-03-09    0.009595
2016-03-10    0.010666
2016-03-11    0.012375
2016-03-12    0.023783
2016-03-13    0.008895
2016-03-14    0.012602
2016-03-15    0.015876
2016-03-16    0.016452
2016-03-17    0.028086
2016-03-18    0.007351
2016-03-19    0.015834
2016-03-20    0.020653
2016-03-21    0.020632
2016-03-22    0.021373
2016-03-23    0.018532
2016-03-24    0.019767
2016-03-25    0.019211
2016-03-26    0.016802
2016-03-27    0.015649
2016-03-28    0.020859
2016-03-29    0.022341
2016-03-30    0.024771
2016-03-31    0.023783
2016-04-01    0.022794
2016-04-02    0.024915
2016-04-03    0.025203
2016-04-04    0.024483
2016-04-05    0.124761
2016-04-06    0.221806
2016-04-07    0.131947
Name: last_seen, dtype: float64
```

In [21]:

```
print(autos["ad_created"].str[:10].unique().shape)
(autos["ad_created"]
        .str[:10]
        .value_counts(normalize=True, dropna=False)
        .sort_index()
        )
```

(76,)

Out[21]:

```
2015-06-11    0.000021
2015-08-10    0.000021
2015-09-09    0.000021
2015-11-10    0.000021
2015-12-05    0.000021
                ...
2016-04-03    0.038855
2016-04-04    0.036858
2016-04-05    0.011819
2016-04-06    0.003253
2016-04-07    0.001256
Name: ad_created, Length: 76, dtype: float64
```

In [22]:

```
autos["registration_year"].describe()
```

Out[22]:

```
count    48565.000000
mean      2004.755421
std         88.643887
min       1000.000000
25%       1999.000000
50%       2004.000000
75%       2008.000000
max       9999.000000
Name: registration_year, dtype: float64
```

汽车注册年份中有一些异常值，如最小值的1000和最大值的9999

6.处理不正确的正确年份数据

汽车注册年份可接受的最低值可以定为：1900年，因为1886汽车才刚发明，要经过一定时间的发展才能走入大众视野，让大众注册使用；

汽车注册年份可接受的最高值可以定为：2016年，因为数据集是2016年的，依次2016年以上注册年份的车辆绝对不正确。

In [23]:

```
(~autos["registration_year"].between(1900,2016)).sum() / autos.shape[0]
```

Out[23]:

0.038793369710697

In [24]:

```
autos = autos[autos["registration_year"].between(1900,2016)]
autos["registration_year"].value_counts(normalize=True).head(10)
```

Out[24]:

```
2000    0.067608
2005    0.062895
1999    0.062060
2004    0.057904
2003    0.057818
2006    0.057197
2001    0.056468
2002    0.053255
1998    0.050620
2007    0.048778
Name: registration_year, dtype: float64
```

大多数汽车在过去的20年被注册，即1996-2016

7:按品牌探索价格

循环执行聚合。 流程如下所示:

-确定我们要汇总的唯一值

-创建一个空字典来存储我们的汇总数据

-遍历唯一值（val），并针对每个值:

-通过唯一值对数据框进行分组
-计算我们感兴趣的任何一列的平均值
-将val/mean 作为 k/v（键值对）分配给字典。

In [25]:

```
autos["brand"].value_counts(normalize=True)
```

Out[25]:

```
volkswagen        0.211264
bmw               0.110045
opel              0.107581
mercedes_benz     0.096463
audi              0.086566
ford              0.069900
renault           0.047150
peugeot           0.029841
fiat              0.025642
seat              0.018273
skoda             0.016409
nissan            0.015274
mazda             0.015188
smart             0.014160
citroen           0.014010
toyota            0.012703
hyundai           0.010025
sonstige_autos    0.009811
volvo             0.009147
mini              0.008762
mitsubishi        0.008226
honda             0.007840
kia               0.007069
alfa_romeo        0.006641
porsche           0.006127
suzuki            0.005934
chevrolet         0.005698
chrysler          0.003513
dacia             0.002635
daihatsu          0.002506
jeep              0.002271
subaru            0.002142
land_rover        0.002099
saab              0.001649
jaguar            0.001564
daewoo            0.001500
trabant           0.001392
rover             0.001328
lancia            0.001071
lada              0.000578
Name: brand, dtype: float64
```

In [26]:

```
brand_counts = autos["brand"].value_counts(normalize=True)
common_brands = brand_counts[brand_counts > .05].index
print(common_brands)
```

```
Index(['volkswagen', 'bmw', 'opel', 'mercedes_benz', 'audi', 'ford'], dtype='objec
t')
```

In [27]:

```
brand_mean_prices = {}

for brand in common_brands:
    brand_only = autos[autos["brand"] == brand]
    mean_price = brand_only["price"].mean()
    brand_mean_prices[brand] = int(mean_price)

brand_mean_prices
```

Out[27]:

```
{'volkswagen': 5402,
 'bmw': 8332,
 'opel': 2975,
 'mercedes_benz': 8628,
 'audi': 9336,
 'ford': 3749}
```

- 'bmw','mercedes_benz','audi'是价格高的类型
- 'volkswagen'是价格居中的类型
- 'Ford','Opel'是价格低的类型

8.将聚合数据存储在DataFrame中

对于排名前6位的品牌，让我们使用汇总来了解这些汽车的平均行驶里程，以及是否与均价存在的明显的联系。

In [28]:

```
bmp_series = pd.Series(brand_mean_prices)
pd.DataFrame(bmp_series, columns=["mean_price"])
```

Out[28]:

	mean_price
volkswagen	5402
bmw	8332
opel	2975
mercedes_benz	8628
audi	9336
ford	3749

In [29]:

```
brand_mean_mileage = {}

for brand in common_brands:
    brand_only = autos[autos["brand"] == brand]
    mean_mileage = brand_only["odometer_km"].mean()
    brand_mean_mileage[brand] = int(mean_mileage)

mean_mileage = pd.Series(brand_mean_mileage).sort_values(ascending=False)
mean_prices = pd.Series(brand_mean_prices).sort_values(ascending=False)
```

In [30]:

```
brand_info = pd.DataFrame(mean_mileage,columns=['mean_mileage'])
brand_info
```

Out[30]:

	mean_mileage
bmw	132572
mercedes_benz	130788
opel	129310
audi	129157
volkswagen	128707
ford	124266

In [31]:

```
brand_info["mean_price"] = mean_prices
brand_info
```

Out[31]:

	mean_mileage	mean_price
bmw	132572	8332
mercedes_benz	130788	8628
opel	129310	2975
audi	129157	9336
volkswagen	128707	5402
ford	124266	3749

从上述表格中可以得出：一般情况下，较昂贵的车有较高的里程；较便宜的车有较低的里程，但里程数相差不大。

总结与进一步思考：

这个指导性项目中，我们练习了应用各种pandas方法来探索和理解有关汽车清单的数据集。 以下是您可以考虑的一些后续步骤：

数据清理下一步： 识别使用德语单词的分类数据，对其进行翻译，然后将值映射到对应的英语单词 将日期转换为统一的数值数据，因此"2016-03-21"成为整数20160321。 查看名称列中是否存在可以提取为新列的特定关键字 分析下一步： 查找最常见的品牌/型号组合 将odometer_km分成几组，并使用汇总查看平均价格是否遵循基于里程的任何模式。 有损坏的汽车比没有损坏的汽车便宜多少？