学习概况

理解小雨大佬的第一份代码，对自己不懂的部分添加注释，B站的数据挖掘的视频的观看，对数据挖掘的原理有一个大致的了解。

首先是对代码阅读部分：

对小雨的代码自己先阅读，逐句敲打，然后对自己不懂得进行百度意思，也自己在代码上添加一些注释这样。

先读取数据

```
import pandas as pd
data_balance = pd.read_csv('Purchase Redemption Data/user_balance_table.csv')
print(data.head())
```

```
   user_id  report_date  tBalance  yBalance  total_purchase_amt  \
0        1     20140805     20385     20383                   2
1        1     20140808     20391     20389                   2
2        1     20140811     20397     20395                   2
3        1     20140814     20403     20401                   2
4        1     20140817     20409     20407                   2

   direct_purchase_amt  purchase_bal_amt  purchase_bank_amt  total_redeem_amt  \
0                    0                 0                  0                 0
1                    0                 0                  0                 0
2                    0                 0                  0                 0
3                    0                 0                  0                 0
4                    0                 0                  0                 0

   consume_amt  transfer_amt  tftobal_amt  tftocard_amt  share_amt  category1  \
0            0             0            0             0          2        NaN
1            0             0            0             0          2        NaN
2            0             0            0             0          2        NaN
3            0             0            0             0          2        NaN
4            0             0            0             0          2        NaN

   category2  category3  category4
0        NaN        NaN        NaN
1        NaN        NaN        NaN
2        NaN        NaN        NaN
3        NaN        NaN        NaN
4        NaN        NaN        NaN
```

由图所示，数据只有前面与日期有关的标签，没有日期、时间等，所以在这里我们给数据添加上去，方便后续使用时间序列来分析，

```
# 为数据集添加时间戳  在数据末尾添加标签，方便使用时间序列分析
data_balance['date'] = pd.to_datetime(data_balance['report_date'],
format="%Y%m%d")  # 转为时间格式(datetime64[ns])
data_balance['day'] = data_balance['date'].dt.day
data_balance['month'] = data_balance['date'].dt.month
data_balance['year'] = data_balance['date'].dt.year
data_balance['weekday'] = data_balance['date'].dt.weekday
data_balance['week'] = data_balance['date'].dt.isocalendar().week
print(data_balance.head())
```

```
   user_id  report_date  tBalance  yBalance  total_purchase_amt  \
0        1     20140805     20385     20383                   2
1        1     20140808     20391     20389                   2
2        1     20140811     20397     20395                   2
3        1     20140814     20403     20401                   2
4        1     20140817     20409     20407                   2

   direct_purchase_amt  purchase_bal_amt  purchase_bank_amt  total_redeem_amt  \
0                    0                 0                  0                 0
1                    0                 0                  0                 0
2                    0                 0                  0                 0
3                    0                 0                  0                 0
4                    0                 0                  0                 0

   consume_amt  ...  category1  category2  category3  category4       date  \
0            0  ...        NaN        NaN        NaN        NaN 2014-08-05
1            0  ...        NaN        NaN        NaN        NaN 2014-08-08
2            0  ...        NaN        NaN        NaN        NaN 2014-08-11
3            0  ...        NaN        NaN        NaN        NaN 2014-08-14
4            0  ...        NaN        NaN        NaN        NaN 2014-08-17

   day  month  year week  weekday
0    5      8  2014   32        1
1    8      8  2014   32        4
2   11      8  2014   33        0
3   14      8  2014   33        3
4   17      8  2014   33        6

[5 rows x 24 columns]
```

由上图可知，已经在末尾添加了时间的数据信息，

```
# 聚合时间数据
total_balance = data_balance.groupby(['date'])[['total_purchase_amt',
'total_redeem_amt']].sum().reset_index()
print(total_balance)  # 测试数据集
```

上面的第一行代码有点长，我逐一解释：

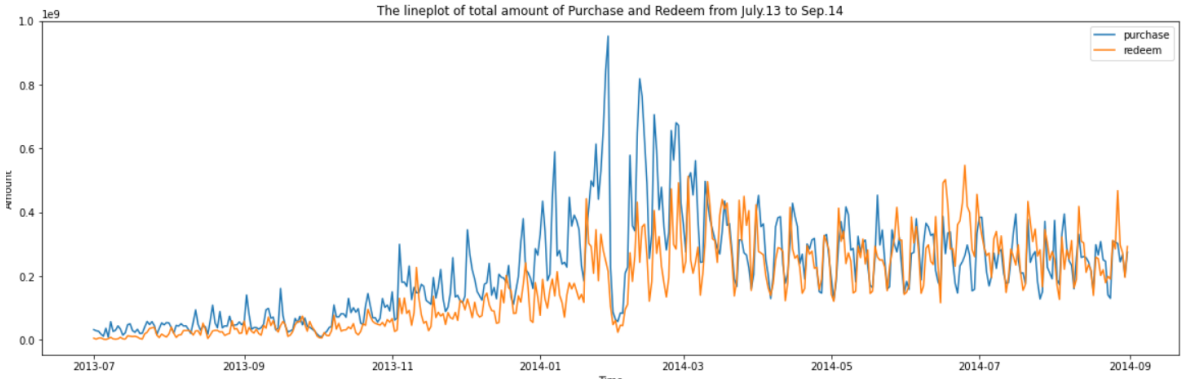
首先：groupby函数主要的作用是进行数据的分组以及分组后地组内运算，在分组之后我们对其每组每天的买入卖出求和，然后使用重置索引（在每行对应的数据前面加序列号），这样方便下方对数据进行可视化输出。

显示结果如下：

```
          date  total_purchase_amt  total_redeem_amt
0   2013-07-01            32488348           5525022
1   2013-07-02            29037390           2554548
2   2013-07-03            27270770           5953867
3   2013-07-04            18321185           6410729
4   2013-07-05            11648749           2763587
..         ...                 ...               ...
422 2014-08-27           302194801         468164147
423 2014-08-28           245082751         297893861
424 2014-08-29           267554713         273756380
425 2014-08-30           199708772         196374134
426 2014-08-31           275090213         292943033
```

```
# 画出每日总购买与赎回量的时间序列图
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(20,6))
plt.plot(total_balance['date'],
total_balance['total_purchase_amt'],label='purchase')
plt.plot(total_balance['date'],
total_balance['total_redeem_amt'],label='redeem')

plt.legend(loc='best')
plt.title("The lineplot of total amount of Purchase and Redeem from July.13 to
Sep.14")
plt.xlabel("Time")
plt.ylabel("Amount")
plt.show()
```

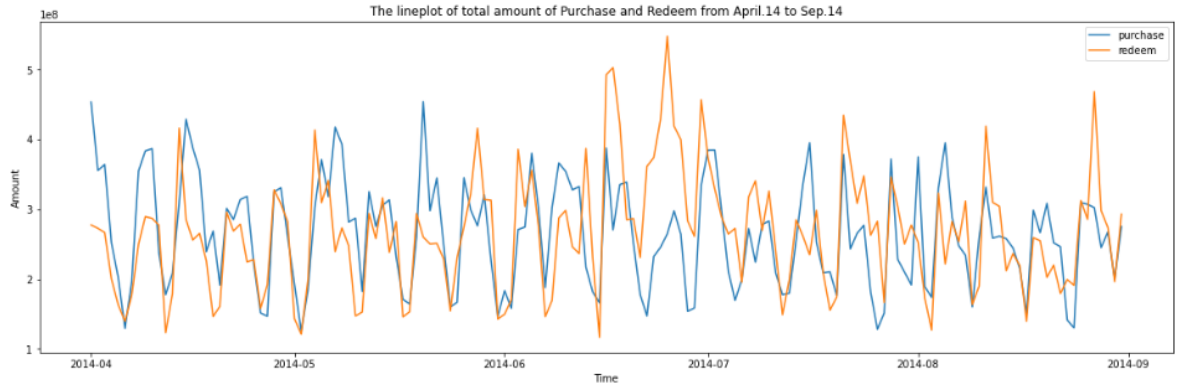
将数据可视化显示在图表中：



现在采取部分数据进行处理以及预测9月后的数据，因为在由上图明显可知，在三月前的数据跌宕起伏，在2013年的数据相关性有太少，所以，综合以上考虑，采用4月份以后的数据

```
# 画出4月份以后的时间序列图
import datetime as dt
total_balance_1 = total_balance[total_balance['date'].dt.date >=
datetime.date(2014,4,1)]
fig = plt.figure(figsize=(20,6))
plt.plot(total_balance_1['date'],
total_balance_1['total_purchase_amt'],label='purchase')
plt.plot(total_balance_1['date'],
total_balance_1['total_redeem_amt'],label='redeem')
plt.legend()
plt.title("The lineplot of total amount of Purchase and Redeem from April.14 to
Sep.14")
plt.xlabel("Time")
plt.ylabel("Amount")
plt.show()
```

以下为数据可视化图解

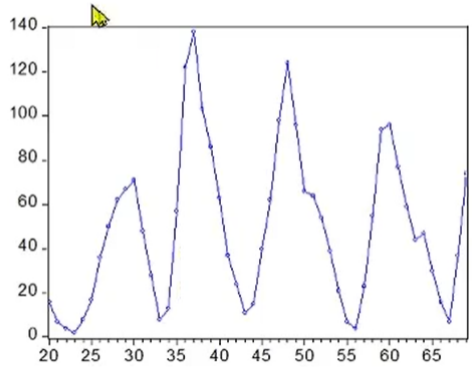


B站学习部分

对上课所说的自己截图以及记下笔记：

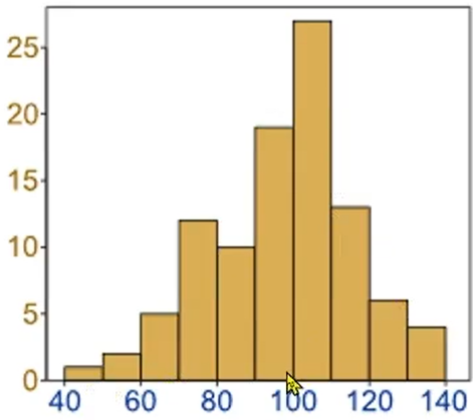
绘图方法表示数据的方法以及好处：

时序图：



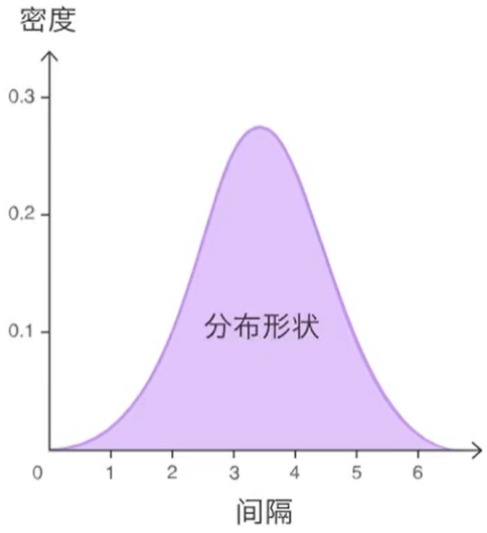
方便观察数据的特点

直方图



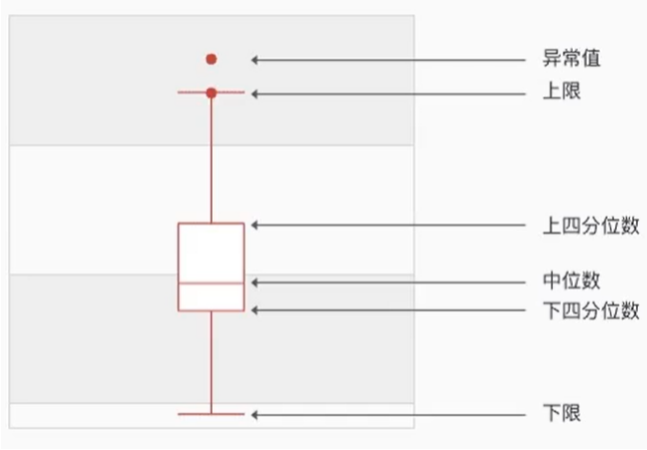
便于观察数据的分布

密度曲线图



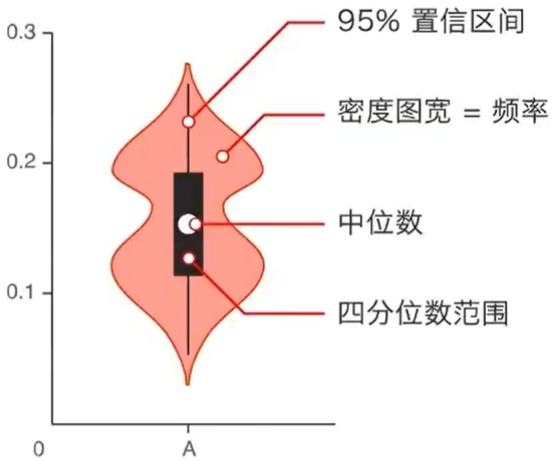
理解概率密度函数的分布图

箱型图



便于查看数据的异常状况，以及不同数据之际的分布的对比

小提琴图



相当于进阶版箱型图，可以看出某个值附近分布的概率

量化方法：相关性分析

- 定类变量：名义型变量；性别，工作类型，城市所作地，邮政编码
- 定序变量：不仅分类，还按某种特性排序；两值得差无意义；教育程度，收入能力，消费能力，电影评分
- 定距变量：可看、比较大小、差有意义得变量

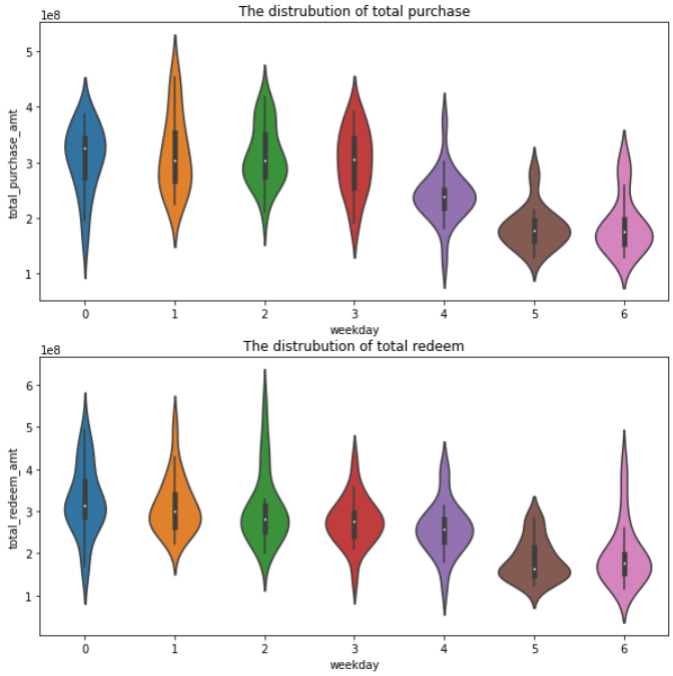
变量得相关性分析方法：

	定类	定序	定距
定类	卡方类测量	卡方类测量	Eta 系数
定序		● Spearman 相关系数 ● 同序-异序对测量	Spearman 相关系数
定距			Pearson 相关系数

今天把剩下的第一部分的代码阅读完毕，感觉作者在剩下的部分更多的是利用提供的数据给我们进行更多的讲解数据分析的部分。下面我以我的看法总结一下作者对数据分析的常用操作

小提琴的构建以及绘制，通过小提琴图的绘制，我们可以直接在图上看出哪些位置的密度大，可以说小提琴图与箱形图很像

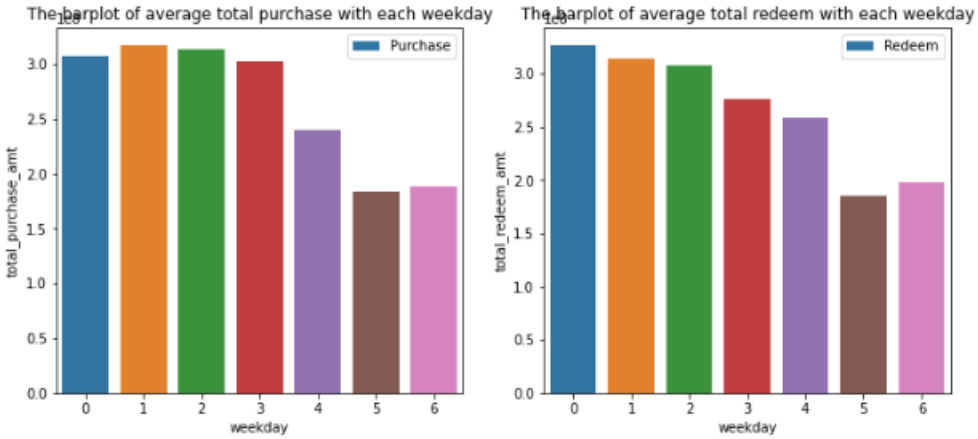
```
a = plt.figure(figsize=(10,10))
scatter_para = {'marker':'.', 's':3, 'alpha':0.3}
line_kws = {'color':'k'}
plt.subplot(2,1,1)
plt.title('The distrubution of total purchase')
sns.violinplot(x='weekday', y='total_purchase_amt', data = total_balance_1,
scatter_kws=scatter_para, line_kws=line_kws)
plt.subplot(2,1,2)
plt.title('The distrubution of total redeem')
sns.violinplot(x='weekday', y='total_redeem_amt', data = total_balance_1,
scatter_kws=scatter_para, line_kws=line_kws)
plt.show()
```



由上图可以推测出赎回的分布倒是看得出比较稳定，而购买的分布没看的出有明显的规律

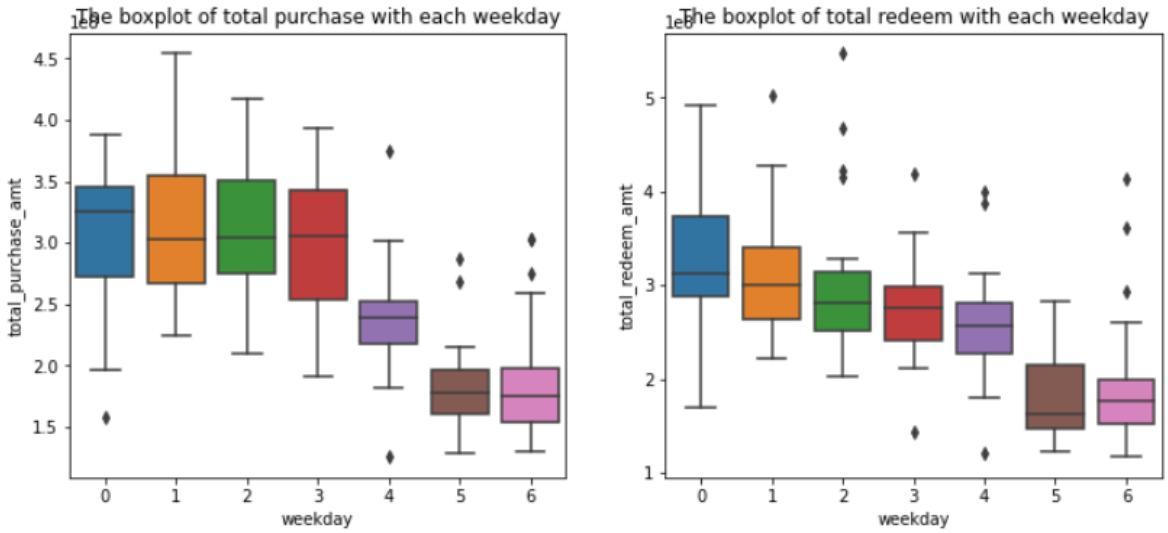
而我这里我对line_kws以及scatter_para这两个参数存在疑惑，不太清楚它让图形改变了什么（事实上我注释掉这两个配置图形一点变化都没有）

```
# 按周一到周末对数据聚合后取均值
week_sta = total_balance_1[['total_purchase_amt', 'total_redeem_amt',
'weekday']].groupby('weekday', as_index=False).mean()

# 分析周一到周末的中位数特征
plt.figure(figsize=(12, 5))
ax = plt.subplot(1,2,1)
plt.title('The barplot of average total purchase with each weekday')
ax = sns.barplot(x="weekday", y="total_purchase_amt", data=week_sta,
label='Purchase')
ax.legend()
ax = plt.subplot(1,2,2)
plt.title('The barplot of average total redeem with each weekday')
ax = sns.barplot(x="weekday", y="total_redeem_amt", data=week_sta,
label='Redeem')
ax.legend()
plt.show()
```



这个柱状图我认为可以理解为将上方数据的不同的表现形式，图形的绘制方法即是传统的加限定标签绘制、载入数据的那种方法

```
# 画出翌日的箱型图

plt.figure(figsize=(12, 5))
ax = plt.subplot(1,2,1)
plt.title('The boxplot of total purchase with each weekday')
ax = sns.boxplot(x="weekday", y="total_purchase_amt", data=total_balance_1)
ax = plt.subplot(1,2,2)
plt.title('The boxplot of total redeem with each weekday')
ax = sns.boxplot(x="weekday", y="total_redeem_amt", data=total_balance_1)
```



箱形图的构建也没什么特别的，我认为这里给出也是为了能让我们熟悉多一个让数据可视化的方式。

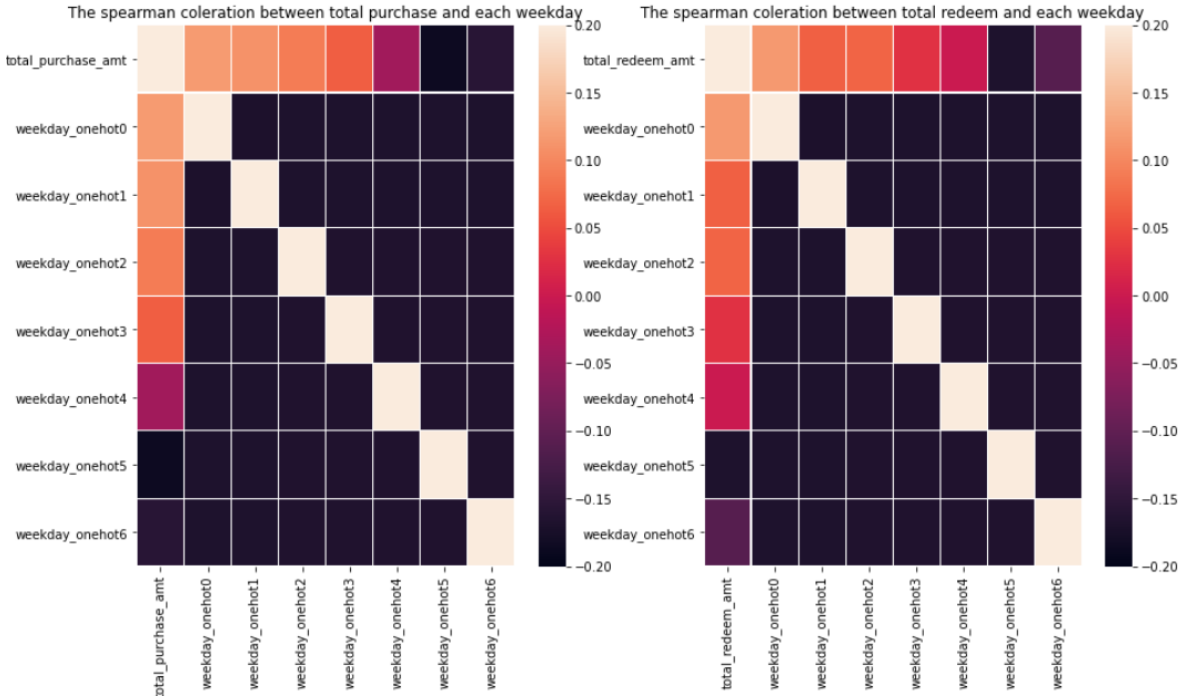
下面这段代码先对数据进行处理，再进行绘制图形

```
# 使用OneHot方法将翌日特征划分，获取划分后特征

from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
total_balance = total_balance.reset_index()
week_feature =
encoder.fit_transform(np.array(total_balance['weekday']).reshape(-1,
1)).toarray()
week_feature = pd.DataFrame(week_feature,columns=
['weekday_onehot']*len(week_feature[0]))
feature = pd.concat([total_balance, week_feature], axis = 1)
[['total_purchase_amt', 'total_redeem_amt','weekday_onehot','date']]
feature.columns = list(feature.columns[0:2]) + [x+str(i) for i,x in
enumerate(feature.columns[2:-1])] + ['date']
```

对这段代码的理解：

- 使用onehot将数据从分类特征的元素转化为一个可以用来计算的值。
- 使用方法fit_transform()直接一步到位的将数据转为可计算的值
- 如果不加 toarray() 的话，输出的是稀疏的存储格式，即索引加值的形式，所以在末尾添加toarray
- 然后DataFrame化

```
# 画出划分后翌日特征与标签的斯皮尔曼相关性

f, ax = plt.subplots(figsize = (15, 8))
plt.subplot(1,2,1)
plt.title('The spearman coleration between total purchase and each weekday')
sns.heatmap(feature[[x for x in feature.columns if x not in ['total_redeem_amt',
'date'] ]].corr('spearman'),linewidths = 0.1, vmax = 0.2, vmin=-0.2)
plt.subplot(1,2,2)
plt.title('The spearman coleration between total redeem and each weekday')
sns.heatmap(feature[[x for x in feature.columns if x not in
['total_purchase_amt', 'date'] ]].corr('spearman'),linewidths = 0.1,  vmax =
0.2, vmin=-0.2)
```



依然是构图与绘图，在这里这个图传达了表示两个变量之间联系的密切程度（斯皮尔曼相关性），它和[相关系数r]一样，取值在-1到+1之间，所不同的是它是建立在等级的基础上计算的。

总结：发现接下来的代码部分以及讲解与上文其实可以说是有一点相似的地方，更多的是将数据可视化的呈现出来，然后对呈现出的数据进行分析，可以说，据我理解，数据的分析建立在将数据可视化呈现的基础上，这里就不再呈现以及写出来了。