关卡 4

基于 litemall 的数据挖掘分析

华为技术有限公司

目录

1 电商数据挖掘简单操作

1.1 实验介绍

1.1.1 关于本实验

本实验主要介绍如何利用数据挖掘的基本算法，包括回归和关联两大场景，从订单数据出发，进行相关预测和分析，通过数据指导后续经营决策。

1.1.2 实验目的

- 帮助学员了解并掌握如何利用 Python 完成基本的数据挖掘，并能对预测结果进行相应的分析。

1.2 实验代码

1.2.1 利用回归算法，分析周别与累计营销额的关系

步骤 1　将 DWS 里的数据导入到 Python 中

首先，导入通用的依赖包。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import psycopg2
```

如果出现某个依赖包不存在，可以参考随堂练习手册的内容进行安装。

参考之前的 get_table() 方法，连接 DWS，并完成执行相应的 SQL 语言：

```
def get_table( ip,sql, port, database, user, password):
    '''
    连接 DWS，操作相应的 sql 语句来进行查询，注意要把 ip, port, database, user 和 password 换成对应的才行。
    '''
    connection = psycopg2.connect(host=ip, port=port, database=database, user=user, password=password) #连接到对应的数据库，具体请看数据库编程的内容
    connection.set_client_encoding('utf-8') # 把编码格式换成 utf8，以防止出现乱码
    cursor = connection.cursor() #创建游标
    cursor.execute(sql) #执行传入的 sql 语言，不同的 sql 语言执行结果不同，需要在调用函数时指定
    rows=cursor.fetchall() #获取所读取的内容，存储形式为元组
    cursor.close() #关闭游标
    connection.close() #关闭数据库连接
    df = pd.DataFrame(rows,index=range(1,len(rows)+1)) #将元组形式的数据转化为 DataFrame 的形式，方便数据分析
    return df #函数的返回值为我们得到的 DataFrame
```

下面执行我们的第一条 SQL 语言，用于查询 2020 年 3 月各省 GMV：

```
'''注意需要把 ip, port, database, user 和 password 换成对应的才行：'''
ip_DWS='xxx.xxx.xxx.xxx' #DWS 对应的公网 ip
port_DWS='8000' #默认 8000 不用修改
db_DWS='xxx' #修改为 DWS 中对应的数据库名
username_DWS='xxx' #购买 DWS 服务时自定义的用户名
password_DWS='********' #购买 DWS 服务时自定义的密码

sql1 = '''select order_id,order_status,user_key,order_price,add_date from target.orders;'''
sql_datedim = 'select date_key, full_date,week_num_in_year from target.date_dimension;'

df_order_data = get_table(ip_DWS,sql1,port_DWS,db_DWS,username_DWS,password_DWS)
df_order_data.columns = ['order_id','order_status','user_key','order_price','add_date']
date_dim = get_table(ip_DWS,sql_datedim,port_DWS,db_DWS,username_DWS,password_DWS)
date_dim.columns = ['date key','full date','week num in year']
```

这一部分和之前可视化的基本相同。

如果想查看前 5 条数据，可以用：

```
df_order_data.head(5) #查询前 5 条数据。修改括号内的数字可修改查询的数据量
```

显示结果为（**此处需要大家提交对应的截图至作业模板中**）：

	order_id	order_status	user_key	order_price	add_date
1	1	401	12251	3688.00	20200409
2	2	401	1516	2199.00	20200229
3	4	401	7635	2199.00	20200226
4	5	401	6428	2199.00	20200120
5	8	401	3088	16999.00	20200129

步骤 2　简单数据处理与特征工程

根据数据挖掘的基本流程，在正式开始回归分析之前，我们需要完成数据处理和特征工程。

简单来说，我们需要把 2020-03-01 这样的日期转化为对应的周别，也就是对应该年度第几周。我们可以通过 date_dim 这张表里的对应关系，创建一个字典，然后利用 map 函数快速完成转化：

```
price_and_time = df_order_data.loc[:,['order_price','add_date']] #取出每一笔订单的价格以及订单添加的时间
date2weeknum = dict(zip(date_dim['date key'],date_dim['week num in year'])) #创建一个字典，把日期和周别一一对应
week_num = price_and_time['add_date'].map(date2weeknum).map(int) #调用 map 函数，按照之前创建的字典完成数据转化，然后统一转化成 int 类型
columns_new = ['order_price','add_date','week_num'] #新的列名
df = price_and_time.reindex(columns=columns_new,fill_value=0) #利用 reindex 方法，创建新的 DataFrame，其中列名就是 columns_new，新的列先用 0 填充
df.loc[:,'week_num']=week_num #传入 week_num 数据到新列中
```

至此，我们已完成了“日期”到“周别”的转化。

看一下前 5 条数据能否正确显示：

```
df.head(5)
```

看到结果如下，说明已经可以正确执行操作了（**此处需要大家提交对应的截图至作业模板中**）。

	order_price	add_date	week_num
1	3688.00	20200409	15
2	2199.00	20200229	9
3	2199.00	20200226	9
4	2199.00	20200120	4
5	16999.00	20200129	5

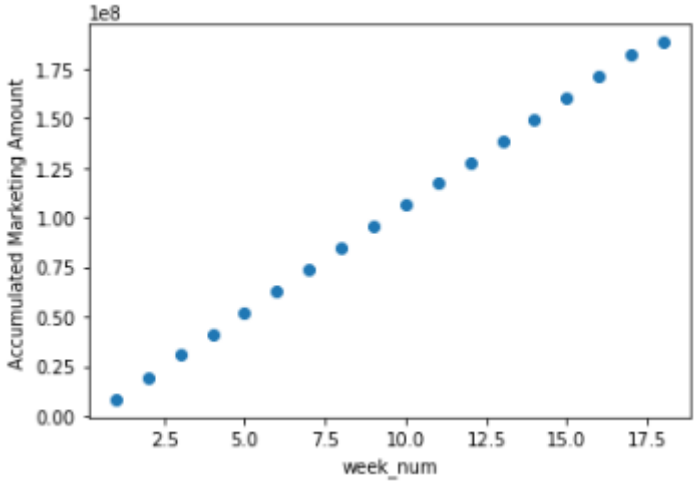
此外，我们要分析的是该周之前所有订单的累计营销额，所以还需要进行累加求和：

```
x = sorted(list(set(week_num))) #利用 set 函数进行去重，然后转化为列表，并完成排序。排序后的列表就是对应的自变量。
x = np.array(x).reshape((-1, 1)) #后续的回归算法要求 x 是一个二维数组，reshape 方法可帮助我们进行转化。
#对因变量 y 的处理包含了比较多的步骤：
#1.使用 DataFrame 的 query 方法，查询满足 week_num 等于对应 x 中各值的数据；
#2.对查询出来的结果，取出'order_price'这一列，进行求和操作，以便获得每周的营销额；
#3.以上步骤在列表推导式内完成，所以得到的结果是一个列表，然后调用 np.cumsum 方法，对该列表进行累加。
y = np.cumsum([float(df.query('week_num==%d'%n).order_price.sum()) for n in x])
```

至此，我们完成了特征的转化。可以利用 matplotlib 画出自变量和因变量之间的散点图：

```
plt.figure(1)
plt.scatter(x,y)
plt.xlabel('week_num')
plt.ylabel('Accumulated Marketing Amount')
```

散点图如下所示（**此处需要大家提交对应的截图至作业模板中**）：



可以看到，我们面对的其实是一个简单的一元回归问题。

显然，我们只需要考虑有监督的机器学习算法。在有监督的机器学习过程中，划分训练集和测试集是一个非常实用的方法。通常来说，我们可以基于训练集完成模型的训练，再利用该模型作用于测试集上完成预测，以便进行模型评估。

在 Python 中，sklearn 是专门负责机器学习的第三方模块，而在 sklearn 中就有专门的方法：sklearn.model_selection.train_test_split 方法，帮助我们划分测试集与训练集。其基本语法为：

X_train,X_test, y_train, y_test =sklearn.model_selection.train_test_split(train_data,train_target,test_size=0.25, random_state=0,stratify=y_train)

其中输出的就是拆分之后的结果。而其中比较重要的参数包括 *train_data* 所要划分的数据集，*test_size：* 样本占比，以及 *random_state：* 随机种子。如果不指定样本占比，默认值就是 0.25，也就是测试集占 1/4，训练集占 3/4. 我们实验中使用的就是默认值，**感兴趣的同学请尝试一下，使用不同的比例，对于最后的结果会有什么影响**?

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0) #random_state 指定了随机的状态，使得每次随机的结果都是固定的，这样我们的结果总是可以复现。
```

下面我们比较不同的回归算法，找出其中最合适的方法，以便完成上半年营销总额的预测。

步骤 3　使用线性回归算法，完成上半年营销总额的分析

根据上一步骤中呈现的散点图，我们可以发现，所有数据大致呈现线性的关系，所以我们可以首先尝试一下线性回归模型。

通过简单调用 sklearn.linear_model 里的 LinearRegression 方法来完成模型训练。其基本用法如下：

sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)

其中 *fit_intercept* 表示是否计算截距。默认是需要计算截距，也就是不从原点出发。*Normalize* 表示是否标准化，标准化就是将服从正态分布的数据调整为均值为 0，方差为 1 的状态。*copy_X* 表示是否覆盖原 X，而 *n_jobs* 表示是否多核并发执行。

```
from sklearn.linear_model import LinearRegression
model_lr = LinearRegression() #生成线性回归模型对象。所有参数使用默认参数即可。
model_lr.fit(x_train, y_train) #调用 fit 方法，完成模型的训练
print('The function should be y=%dx+%d'%(model_lr.coef_,model_lr.intercept_)) #取出系数和截距，生成一元一次函数。该函数就是我们的一元线性回归的模型了。
```

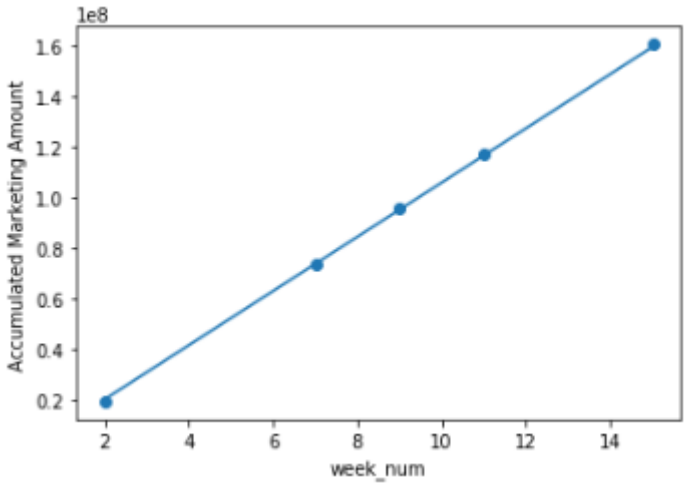
输出结果（**此处需要大家提交对应的截图至作业模板中**）：

```
The function should be y=10741465x+-1530868
```

那么该模型的表现如何呢？通过简单的可视化，我们应该能大致了解模型的拟合程度：

```
y_pred1 = model_lr.predict(x_test) #调用之前训练好的 model_lr 对象的 predict 方法，作用在测试集上
plt.figure(2) #展示在同一张图里
plt.scatter(x_test,y_test) #将测试集的实际值绘制成散点图
plt.plot(x_test,y_pred1) #将训练出来的预测值绘制成折线图
plt.xlabel('week_num')
plt.ylabel('Accumulated Marketing Amount')
plt.show()
```

绘制结果如下所示（**此处需要大家提交对应的截图至作业模板中**）：



可以看到，拟合程度还是非常不错的。更进一步，在胶片中我们提到，回归模型最常用的评估标准有均方误差（MSE）和平均绝对误差（MAE）两种。这里我们选用 MSE 来进行

量化评估，在 Python 中只需要调用 sklearn.metrics 里的 mean_squared_error 方法就可以完成计算：

```
from sklearn.metrics import mean_squared_error
print('The MSE of LR is {:e}'.format(mean_squared_error(y_test/np.linalg.norm(y_test),
y_pred1/np.linalg.norm(y_test))))
```

由于我们的因变量 y 的值比较大，计算 MSE 之后绝对误差非常大，此时相对误差更能反映模型评估的结果。而相对误差的计算也比较简单，只需要同除以真实值的范数即可。得到结果如下所示（**此处需要大家提交对应的截图至作业模板中**）：

```
The MSE of LR is 9.208710e-06
```

这说明我们的误差还是可以接受的。而 2020 年上半年的最后一周是第 26 周，所以根据我们的模型可以预测，上半年的营销总额大约是：

```
model_lr.predict([[26]])
```

输出结果（**此处需要大家提交对应的截图至作业模板中**）：

```
array([2.77747226e+08])
```

这说明，到了第 26 周，上半年的营销总额大约是 277747226 元，也就是 2 亿 7 千万左右。

步骤 4　使用决策树回归模型算法，完成上半年营销总额的分析

另一种常用的回归算法就是决策树，或者又叫做回归树算法。在 Python 中完成决策树回归也非常简单，只需要调用 sklearn.tree 中的 DecisionTreeRegressor 方法即可完成。基本用法如下：

sklearn.tree.DecisionTreeRegressor(criterion='mse', splitter='best', max_features=None, max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, random_state=None, max_leaf_nodes=None, presort=False)

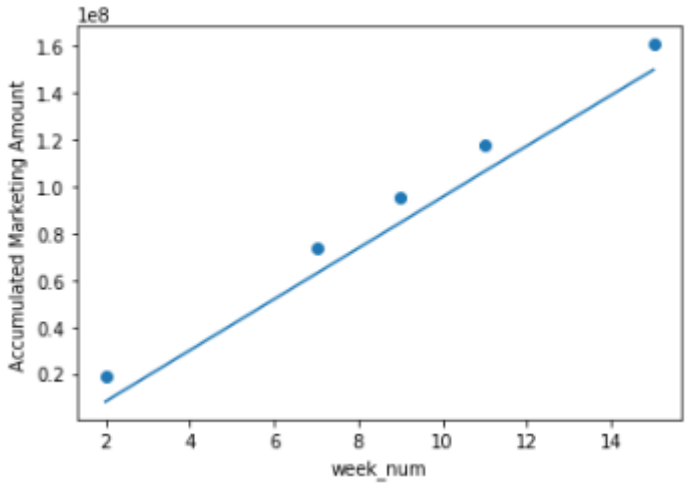
其中常用的参数包括 criterion='mse'，表示训练的结果是使得均方误差达到最小；splitter 表示产生分支的策略，默认使用最佳策略，也可选随机策略；max_features 表示在每个分支处最多考虑几个特征。

```
from sklearn.tree import DecisionTreeRegressor
model_DT = DecisionTreeRegressor() #生成决策树回归模型
model_DT.fit(x_train,y_train) #调用 fit 方法进行训练
```

此时我们没有办法像线性回归那样给出具体的表达式，不过我们还是可以画出训练的模型：

```
y_pred2 = model_DT.predict(x_test)
plt.figure(3)
plt.scatter(x_test,y_test) #绘制真实值的散点图
plt.plot(x_test,y_pred2) #绘制预测值的折线图
plt.xlabel('week_num')
plt.ylabel('Accumulated Marketing Amount')
plt.show()
```

结果如图所示（**此处需要大家提交对应的截图至作业模板中**）：



肉眼可见，模型在测试集上的表现并不是特别好。同样的，我们可以计算并输出对应的（相对）均方误差：

```
print('The MSE of DT is {:e}'.format(mean_squared_error(y_test/np.linalg.norm(y_test),
y_pred2/np.linalg.norm(y_test))))
```

输出结果如下（**此处需要大家提交对应的截图至作业模板中**）：

```
The MSE of DT is 2.187856e-03
```

可以看到，效果确实不如线性回归好了。**大家也可以尝试一下，选择不同的参数，对于结果会有什么影响吗？如果有影响或者没有影响，又是什么原因导致的呢？**

步骤 5　使用随机森林算法进行回归分析

刚才我们看到，使用只有一颗树的决策树回归算法，效果并不好。现在尝试使用一种集成算法——随机森林算法进行回归分析。一般来说，在随机森林算法中我们需要指定决策树的数量：

```
from sklearn.ensemble import RandomForestRegressor
model_RF = RandomForestRegressor(n_estimators=10) #这里使用 10 个决策树，大家也可以试试不同数量的决策树对于结果的影响
model_RF.fit(x_train,y_train)
```

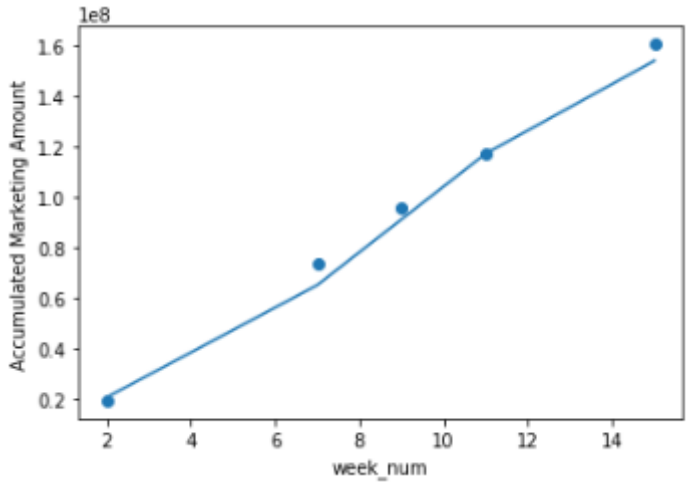
如果对于分析预测结果的精度要求比较高，我们还需要通过别的方式来找到最佳的参数。比如利用交叉验证的方法来进行**网格搜索**，以便找到最佳参数。这里我们就先不演示了，

感兴趣的学员**可以作为创新点进行思考**。如果进行了网格搜索，请输出一下网格搜索提供的最佳参数至作业模板之中。

绘制上述模型：

```
y_pred4 = model_RF.predict(x_test)
plt.figure(5)
plt.scatter(x_test,y_test)
plt.plot(x_test,y_pred4)
plt.xlabel('week_num')
plt.ylabel('Accumulated Marketing Amount')
plt.show()
```

结果如下（**此处需要大家提交对应的截图至作业模板中**）：



计算并输出对应的（相对）均方误差：

```
print('The MSE of RF is {:e}'.format(mean_squared_error(y_test/np.linalg.norm(y_test),
y_pred4/np.linalg.norm(y_test))))
```

输出结果如下（**此处需要大家提交对应的截图至作业模板中**）：

```
The MSE of RF is 5.164359e-04
```

创新指导小 Tips:

上述实验过程里，我们只为大家介绍了线性回归，回归决策树以及随机森林三种常用的模型。但是事实上，对于回归问题，我们还有非常多不同的算法，比如考虑使用 BaggingRegressor 等其他回归算法。如果使用了其他方法，还可以分析一下这些方法的与实验中介绍的方法相比有哪些优缺点，或是有什么联系，并且**将自己的想法与截图一起提交至作业模板之中即可**。

另外，其实我们模型评估的标准有很多，这里我们介绍的是（相对）均方误差，大家也可以试试其他的评估标准，比如 MAE，R^2误差等等，多多进行比较，并尝试分析一下这几种评估标准的优缺点，将结果**提交至作业模板中即可。**

1.2.2 利用关联算法，构建简单推荐系统

步骤 1　将 DWS 里的数据导入到 Python 中

同样的，再次导入通用的依赖包。

```
import pandas as pd
import numpy as np
import psycopg2
```

调用上一小节定义的 get_table 方法，将对应的表导入 Python：

```
sql1 = '''select order_id,order_status,user_key from target.orders;'''
sql2 = '''select order_id,goods_key from target.order_goods;'''

df_order = get_table(ip_DWS,sql1,port_DWS,db_DWS,username_DWS,password_DWS)
df_order.columns = ['order_id','order_status','user_key']
df_order_goods = get_table(ip_DWS,sql2,port_DWS,db_DWS,username_DWS,password_DWS)
df_order_goods.columns = ['order_id','goods_key']
```

这一部分基本相同，不过此时我们不需要 date_dim 这张表，而是需要 order_goods 里的 goods_key 用作分析，因此需要把 sql 语句换成对应的查询语句。

现在我们有了两张表，一张是 df_order，另一张是 df_order_goods。两张表里都有我们需要的数据，所以我们需要进行关联。在 Python 里，将两张表关联形成一张表的方式有很多，这里我们推荐使用 merge 方法。merge 方法可以帮助我们自动识别两张表里相同的列名（也就是相同的字段名），然后根据对应的数据进行关联：

```
df = pd.merge(df_order,df_order_goods,sort=False)
```

由于我们没有指定关联的键名，默认就是所有公共的列名，在这里其实就是 order_id 这一列了；另外我们也没有指定关联规则。其实 merge 方法默认是 inner join，也就是取交集；如果熟悉数据库 SQL 语法的学员可能还知道其他几种关联规则，比如 outer 是取并集，left 和 right 则分别对应以左表为主还是以右表为主。根据业务需要，通过改变 merge 方法里的 how 参数，就可以轻松更改关联规则了。最后 sort 参数表示是否需要进行排序，默认是需要的，但指定 sort=False 可以加快关联的效率。

用 head 简单查看一下几张表的数据：

```
df_order .head(5)
```

显示结果为（**此处需要大家提交对应的截图至作业模板中**）：

	order_id	order_status	user_key
1	1	401	12251
2	2	401	1516
3	4	401	7635
4	5	401	6428
5	8	401	3088

```
df_order_goods.head(5)
```

显示结果为（**此处需要大家提交对应的截图至作业模板中**）：

	order_id	goods_key
1	2	6
2	8	1
3	12	7
4	15	4
5	20	9

```
df.head(5)
```

显示结果为（**此处需要大家提交对应的截图至作业模板中**）：

	order_id	order_status	user_key	goods_key
0	1	401	12251	4
1	2	401	1516	6
2	4	401	7635	6
3	5	401	6428	6
4	8	401	3088	1

步骤 2　简单数据处理与特征工程

进行数据挖掘之前，我们需要把商品转化为对应的类别。同样可以用 map 函数进行操作，不过我们需要先创建一个字典，用于将 goods_key 转化为商品类别：

```
phones_id,laptops_id,earphone_id,wearable_id,pad_id =
range(1,10),range(10,14),range(14,19),range(19,25),range(25,30) #将 goods_key 按商品类别切分
id2type={}
for i in phones_id:
    id2type[i]='Phone'
for i in laptops_id:
    id2type[i]='Laptop'
for i in earphone_id:
    id2type[i]='Earphone'
for i in wearable_id:
    id2type[i]='Wearable_Device'
for i in pad_id:
    id2type[i]='Huawei-Pad'
```

同样的，调用 map 方法批量执行转化工作：

```
goods_type = df['goods_key'].map(id2type)
df_new = df.reindex(list(df.columns).append('goods_type'),fill_value=0)
df_new['goods_type'] = goods_type
```

看一下前 5 条数据：

```
df_new.head(5)
```

如下结果，说明已经正确执行操作了。（**此处需要大家提交对应的截图至作业模板中**）

	order_id	order_status	user_key	goods_key	goods_type
0	1	401	12251	4	Phone
1	2	401	1516	6	Phone
2	4	401	7635	6	Phone
3	5	401	6428	6	Phone
4	8	401	3088	1	Phone

下面我们利用 Apriori 算法进行关联分析。

步骤 3　编写代码实现 Apriori 算法

由于 Python 还没有集成 Apriori 算法，所以我们需要自己完成算法的实现。

```
def apriori(D, minSup):
    ''' 频繁项集用 keys 表示，
      key 表示项集中的某一项，
      cutKeys 表示经过剪枝步的某 k 项集。
      C 表示某 k 项集的每一项在事务数据库 D 中的支持计数
    '''
    #先求出 1 项集合及其支持计数，也就是该种类商品出现的次数。注意此处 C1 的 key 为项集，value 是计数：
    C1 = {}
    for T in D:
        for I in T:
            if I in C1:
                C1[I] += 1
            else:
                C1[I] = 1

    print(C1) #输出每一类商品的购买量

    keys1 = [[i] for i in C1.keys()] #将商品类别转化为 1 项集，使用的是列表嵌套列表的形式

    #对 keys1(1 项集 ) 进行剪枝步：
    #只有出现频率大于等于最小支持度的集合才会被考虑，从而完成剪枝
    keys = sorted([k for k in keys1 if C1[k[0]] / len(D) >= minSup])

    all_keys = [] #用于存储关联项集，作为 Apriori 的输出
    while keys != []:
        C = getC(D, keys) #getC 是我们自定义的函数，用于某 k 项集的每一项在事务数据库 D 中的支持计数。具体的函数主体内容请看后续定义。
        cutKeys = getCutKeys(keys, C, minSup, len(D)) #对 k 项集进行剪枝步。getCutKeys 同样是我们自定义的函数。具体的函数主体内容请看后续定义。
        for key in cutKeys:
            all_keys.append(key) #将剪枝之后的所有 k 项集添加到 all_keys 里，作为最后的输出
        keys = aproiri_gen(cutKeys) # aproiri_gen 是自定义的连接步函数。所谓连接步，简单理解就是对不同的 k 项集求并集，以便实现“连接”。函数主体请看后续定义。

    return all_keys
```

上面的代码就是 Apriori 算法的主体。可以看到，其中有非常多的自定义函数。下面我们就来一一展示这些关键的自定义函数。

首先是 getC，用于计算某 k 项集的每一项在事务数据库 D 中的支持计数：

```
def getC(D, keys):
    '''对 keys 中的每一个 key 进行计数'''
    C = [] #用于存储支持计数
    for key in keys: #遍历传入的 k 项集，其中 key 是长度为 k 的列表
        c = 0 #初始化计数为 0
        for T in D: #对 D 进行遍历。在我们的实验里，T 就对应每一名用户所购买的商品类别的集合
            #如果 key 包含在 T 中，才会进行计数，否则就跳过这一个 k 项集：
            if set(key)<=set(T):
                c += 1
            else:
                continue
        C.append(c)
    return C #返回计数结果
```

然后是 getCutKeys，也就是广义的剪枝步的实现。这个函数比较简单，主要思想就是去掉出现频率小于最小支持度 minSup 的那些 k 项集：

```
def getCutKeys(keys, C, minSup, length):
    '''剪枝步的实现'''
    #获得索引和元素
    for i, key in enumerate(keys):
        if float(C[i]) / length < minSup: #判断 key 对应的频率是否小于最小支持度
            keys.remove(key) #如果频率小于最小支持度，就从 keys 里移除这个 k 项集
    return keys
```

最后一个函数叫做 aproiri_gen，也就是所谓的“连接步”。其主要思想就是将两个 k 项集进行连接，以便得到它们的并集：

```
def aproiri_gen(keys1):
    '''连接步'''
    keys2 = [] #用于存储新得到的集合
    for i,k1 in enumerate(keys1): #遍历所有 keys1 中的 k 项集
        for k2 in keys1[i+1:]: #为了避免重复计算，只考虑 k1 之后的那些 k 项集，完成取并集的操作
            if k1 != k2:
                key = sorted(list(set(k1+k2))) #取并集之后还需要完成排序，保证唯一性
                if key not in keys2: #做好去重工作，以免重复计算
                    keys2.append(key)
    return keys2
```

至此，我们就完成了 Apriori 算法在 Python 的简单实现。

步骤 4　利用 Apriori 算法，挖掘商品的关联特性

有了 Ariori 算法，接下来就是挖掘不同种类的商品之间的关联规则了。

之前我们已经完成了商品 id 到商品类别的特征转化，下面我们需要将这些订单信息按照用户 id 进行分组，也就是将同一个用户所购买的不同商品的类别放在同一个列表内，这就是我们之前提到的 T，然后把所有的 T 组合成一个大的列表 D：

```
D = [list(group[-1]) for group in df_new.groupby('user_key')['goods_type']]
```

这里我们用到了 groupby 方法，这个方法可以帮助我们实现按所指定的关键字快速分组的操作，熟悉 SQL 语法的学员应该不会对此感到陌生。

有了列表 D，我们只需要将其传入 Apriori 函数中，同时指定最小支持度 minSup 就可以了：

```
F = apriori(D, minSup=0.8) #这里我们要求的最小支持度为 0.8，算是比较严格的参数
```

将 F 的信息打印出来：：

```
print('frequent itemset:\n', F)
```

输出结果如下（**此处需要大家提交对应的截图至作业模板中**）：

```
frequent itemset:
 [['Earphone'], ['Huawei-Pad'], ['Phone'], ['Earphone', 'Phone'], ['Huawei-Pad', 'Phone']]
```

请大家分析一下以上结果有什么含义。并且通过分析以上结果，我们能为 litemall 商城提供什么样的经营策略呢？**请大家把自己的分析结果与经营策略提交至作业模板的对应位置。**

创新指导小 Tips:

上述实验过程里，我们指定了最小支持度为 0.8，这其实是一个比较严格的参数了。大家可以尝试分析一下，不同的最小支持度对于我们的结果会有什么影响？**把分析结果提交到作业模板中的对应位置即可**。

另外，学有余力的同学还可以试一试，能否在 Python 中实现其他的关联算法？比如 FP-growth 算法，FreeSpan 算法等等，方便大家展现自己的能力与特点。

1.3 创新指导

本手册只是介绍了回归和关联这两大常见的机器学习任务。但是在实际使用过程中，分类场景和聚类场景出现的频率绝不会比回归和关联低。因此**各位同学可以想一想，针对我们的数据，有哪些场景可以用上分类和聚类算法，而相关的算法以及参数又要如何选择呢**？

1.4 本章小结

本实验基于前几个关卡的数据，通过回归分析与关联分析，对经营决策进行了一系列的挖掘，并通过挖掘的结果，尝试对后续的经营决策进行相关指导，以体现数据的价值。