复旦大学 软件学院 院系

2023~2024 学年第 1 学期期末考试试卷

☑A 卷 ☐B 卷 ☐C 卷

课程名称：机器学习 课程代码：SOFT130090.01

开课院系：软件学院 考试形式：课程论文

姓名：侯斌洋 学号：21302010042 专业：软件工程

提示：请同学们秉持诚实守信宗旨，谨守考试纪律，摒弃考试作弊。学生如有违反学校考试纪律的行为，学校将按《复旦大学学生纪律处分条例》规定予以严肃处理。

（装订线内不要答题）

题号	1	2	3	4	5	6	7	8	总分
得分									

（以下为试卷正文**或课程论文题目**）

以给定的超市数据为基础，使用决策树（含集成算法）、贝叶斯网络、神经网络、聚类、回归分析等算法，分析零售业务的特征、商品类别畅销特点、销量主要影响、销售趋势预测等问题（具体问题自拟）。使用 Python 语言，给出具体的分析过程。

课程论文撰写要求：

1. 全文不少于 4000 字，图文并茂，逻辑清晰，论证透彻。
2. 针对给定的在数据，能利用课程讨论的内容，鼓励采用课堂没讲过的算法，做一定深度的预处理和分析工作，要求有多种方法的比较，论证所选方法的合理性。并能对分析结果从技术和专业领域等进行分析。
3. 语言要通顺，思路符合机器学习的项目流程。

产品销量预测

一、问题分析

（1）背景分析

在当今竞争激烈的市场环境中，企业为了合理高效地运营并取得竞争优势，迫切需要准确的销量预测模型。在近几年的疫情中这一点显得尤为明显，若能准确预测产品的销量情况，无疑能大大提高企业的风险抵抗能力。

然而随着市场环境的不断变化和销售环境的日益复杂，传统的统计方法已经不能满足需求，尤其是当要投放新产品时，传统的统计方法难以根据投放前的特征对其销量进行准确预测，使得企业在投放新产品时风险很高。因此，为了满足市场需求、优化库存管理、提高风险抵抗能力，使用先进的机器学习方法建立一个强大的销量预测模型至关重要。

（2）项目介绍

本项目根据某超市的历史订单数据，建立一个 Sales Volume Forecast 销量预测模型，在给定产品数据的情况下预测该产品未来四年（历史数据跨度为四年）在该超市的销量，从而帮助超市更好地进行市场分析，库存管理，供应链优化等。

二、数据预处理

（1）初步特征选择

原始数据中存在很多对模型训练无关紧要的特征，因此首先要对数据进行过滤，分析数据中的哪些特征属于无关特征并除去。

模型的输入为产品信息，输出为销量的预测值。

原始数据的特征为：订单 Id，利润率，记录数，制造商，产品名称，利润，发货日期，国家，地区，城市，子类别，客户名称，折扣，数量，省/自治区，类别，细分，订单日期，邮寄方式，销售额。

在原始数据中，订单 Id，发货日期，客户名称，细分，城市，省份，地区，订单日期这些特征与模型的输入输出无关，为防止无关数据导致模型过拟合，应去除；记录数和国家在所有数据中都相同，对模型的训练没有帮助，也应去除。注意到邮寄方式作为产品的副加服务，对产品销量也应当有一定的影响应当保留。

使用以下代码剔除无关特征并将新数据保存到 processed.xlsx 中。

```
df = pd.read_excel('超市.xls')
columns_to_drop = ['订单 Id', '发货日期', '客户名称', '细分', '地区', '城市', '省/自治区', '订单日期', '记录数', '国家']
df = df.drop(columns=columns_to_drop, axis=1)
print('初步特征选择后的数据:')
print(df.head(5))
df.to_excel('processed.xlsx', index=False)
```

之后 processed.xlsx 的数据如下所示：

	利润率	制造商	产品名称	利润	子类别	折扣	数量	类别	邮寄方式	销售额
0	-0.47	Fiskars	Fiskar 剪刀，蓝色	NaN	用品	0.4	2	办公用品	二级	￥130
1	0.34	GlobeWeis	GlobeWeis 搭扣信封,红色	￥43	信封	0.0	2	办公用品	标准级	￥125
2	0.13	Cardinal	Cardinal 孔加固材料,回收	￥4	装订机	0.4	2	办公用品	标准级	￥32

（2）处理数据缺失

注意到上面的数据中存在空白值，使用以下代码找到数据中的空白的列

```
print(df.columns[df.isnull().any()])
```

输出为：

```
Index(['利润'], dtype='object')
```

故只在 '利润' 这一列存在空白数据，注意到利润即为销售额与利润率的乘积，而销售额和利润率均不含空数据，故利润的全部空值都可以通过销售额乘利润率进行填充，注意到销售额和利润均含有￥符号，故去除￥使其转化为数值类型再进行填充。代码如下：

```
df['利润'] = df['利润'].replace('￥', '', regex=True).replace(',', '.', regex=True).astype(float)
df['销售额'] = df['销售额'].replace('￥', '',
```

```
regex=True).replace(',', '.', regex=True).astype(float)
    df['利润'].fillna(df['销售额'] * df['利润率'], inplace=True)
```

processed.xlsx 之后的数据如下所示，利润空值均被填充了。

	利润率	制造商	产品名称	利润	子类别	折扣	数量	类别	邮寄方式	销售额
0	-0.47	Fiskars	Fiskar 剪刀，蓝色	-61.1	用品	0.4	2	办公用品	二级	130
1	0.34	GlobeWeis	GlobeWeis 搭扣信封,红色	43	信封	0.0	2	办公用品	标准级	125
2	0.13	Cardinal	Cardinal 孔加固材料,回收	4	装订机	0.4	2	办公用品	标准级	32

由于其他列没有检测到空值，故该表的空值填充完成。

（3）特征拆分

注意到产品名称这一特征同时包含了制造商，产品名和颜色三个特征。为提高模型的可解释性，同时使机器学习算法更好地理解数据并发掘潜在信息，从而提高模型性能，将其拆分为三个特征。其中制造商特征与已存在的特征重合，故直接去掉。

使用下面的代码进行特征拆分。

```
df[['产品', '标签']] = df['产品名称'].str.split(',', expand=True)
df['产品'] = df['产品'].str.split(' ').str[-1]
df = df.drop(['产品名称'], axis=1)
```

之后的数据如下所示，产品名称被拆分为产品和标签这两项：

	利润率	制造商	利润	子类别	折扣	数量	类别	邮寄方式	销售额	产品	标签
0	-0.47	Fiskars	-61.10	用品	0.4	2	办公用品	二级	130.000	剪刀	蓝色
1	0.34	GlobeWeis	43.00	信封	0.0	2	办公用品	标准级	125.000	搭扣信封	红色
2	0.13	Cardinal	4.00	装订机	0.4	2	办公用品	标准级	32.000	孔加固材料	回收

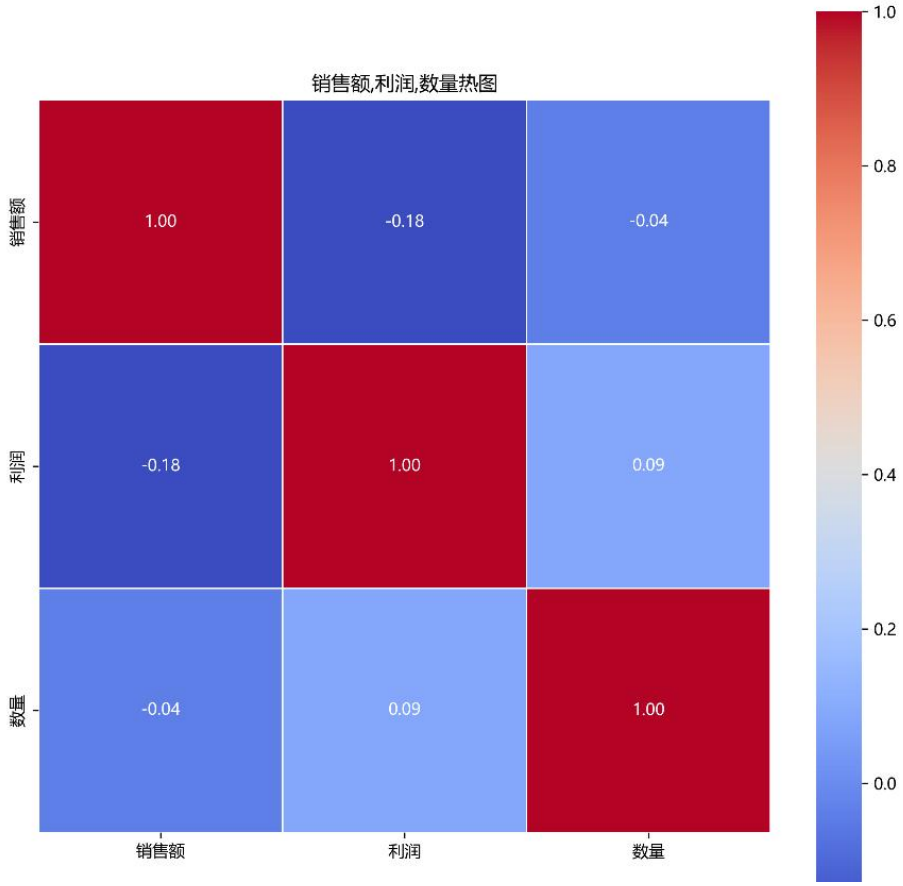
（4）特征转换

（i）首先，注意到：销售额 = 单个售价 * 折扣 * 数量； 利润 = 销售额 * 利润率。而数量是模型需要进行预测的值，销售额、利润和数量直接关联。故若直接采用销售额和利润这两个特征进行训练，则难以探索这两个特征与数量之间潜在的关系，如下所示。

使用下面的代码绘制销售额,利润,数量热图：

```
selected_columns = ['销售额', '利润', '数量']
df_selected = df[selected_columns]
correlation_matrix = df_selected.corr()
plt.figure(figsize=(10, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5, square=True)
plt.title('销售额,利润,数量热图')
plt.show()
```

输出如下：



由上图可以看出，此时销售额、利润与数量之间的相关性均不到 0.1。此时这两个特征受到其内含的数量这一特征的影响，因此并不能准确反应其所包含的单个产品售价和单个产品利润这两个特征与数量之间的关系。

因此应当剔除其中内含的与数量的直接关联。令 单个售价 = 销售额 / 数量； 单个利润 = 利润 / 数量，并用单个售价替代销售额，用单个利润替代利润。

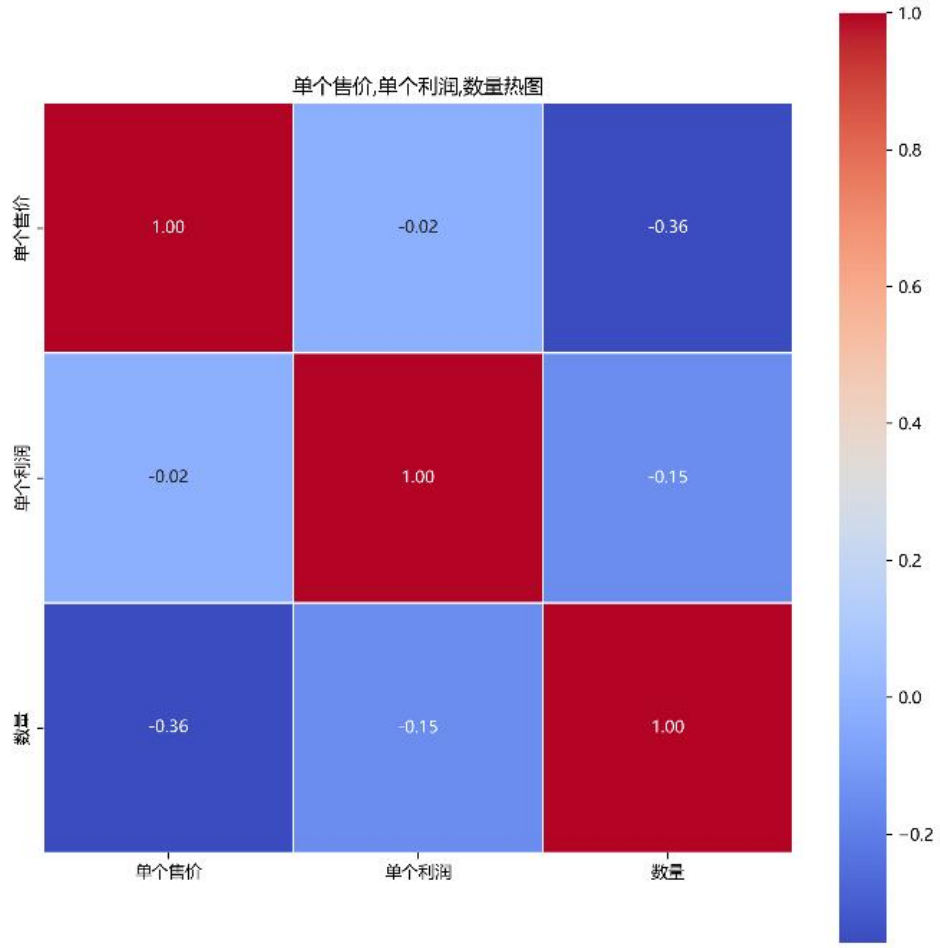
使用如下代码进行上述过程：

```
df = df.rename(columns={'销售额': '单个售价', '利润': '单个利润'})
df['单个售价'] = df['单个售价'] / df['数量']
df['单个利润'] = df['单个利润'] / df['数量']
```

processed.xlsx 之后的数据如下所示，销售额和利润特征被转换了。

	利润率	制造商	单个利润	子类别	折扣	数量	类别	邮寄方式	单个售价	产品	标签
0	-0.47	Fiskars	-30.550000	用品	0.4	2	办公用品	二级	65.000000	剪刀	蓝色
1	0.34	GlobeWeis	21.500000	信封	0.0	2	办公用品	标准级	62.500000	搭扣信封	红色
2	0.13	Cardinal	2.000000	装订机	0.4	2	办公用品	标准级	16.000000	孔加固材料	回收

之后重新绘制单个售价,单个利润,数量热图如下：



可以看出，销售额和利润在去除了与数量之间的直接关联并转换为单个售价和单个利润后，其与数量之间的相关性更强了，在之后的训练中这将更有利于模型对于数据的理解。

（ii）另外，由于模型要预测的是某产品的销量，而目前数据中的特征为每个订单的数量，故需要将数量特征转换为销量特征。

使用以下代码进行数量与销量之间的转换：

```
df['销量'] = df.groupby(['利润率', '制造商', '产品', '标签', '单个利润', '子类别', '折扣', '类别', '邮寄方式', '单个售价'])['数量'].transform('sum')
df = df.drop('数量', axis=1)
```

同时注意到数量转换成销量特征之后出现了很多重复数据，使用以下的代码

去重：

```
df = df.drop_duplicates()
```

之后输出数据表的大小可以发现从 9935 减少到了 8638。现在的数据如下所示，数量转换为了新的销量特征且重复数据已去除。

	利润率	制造商	单个利润	子类别	折扣	类别	邮寄方式	单个售价	产品	标签	销量
0	-0.47	Fiskars	-30.550000	用品	0.4	办公用品	二级	65.000000	剪刀	蓝色	2
1	0.34	GlobeWeis	21.500000	信封	0.0	办公用品	标准级	62.500000	搭扣信封	红色	4
2	0.13	Cardinal	2.000000	装订机	0.4	办公用品	标准级	16.000000	孔加固材料	回收	5

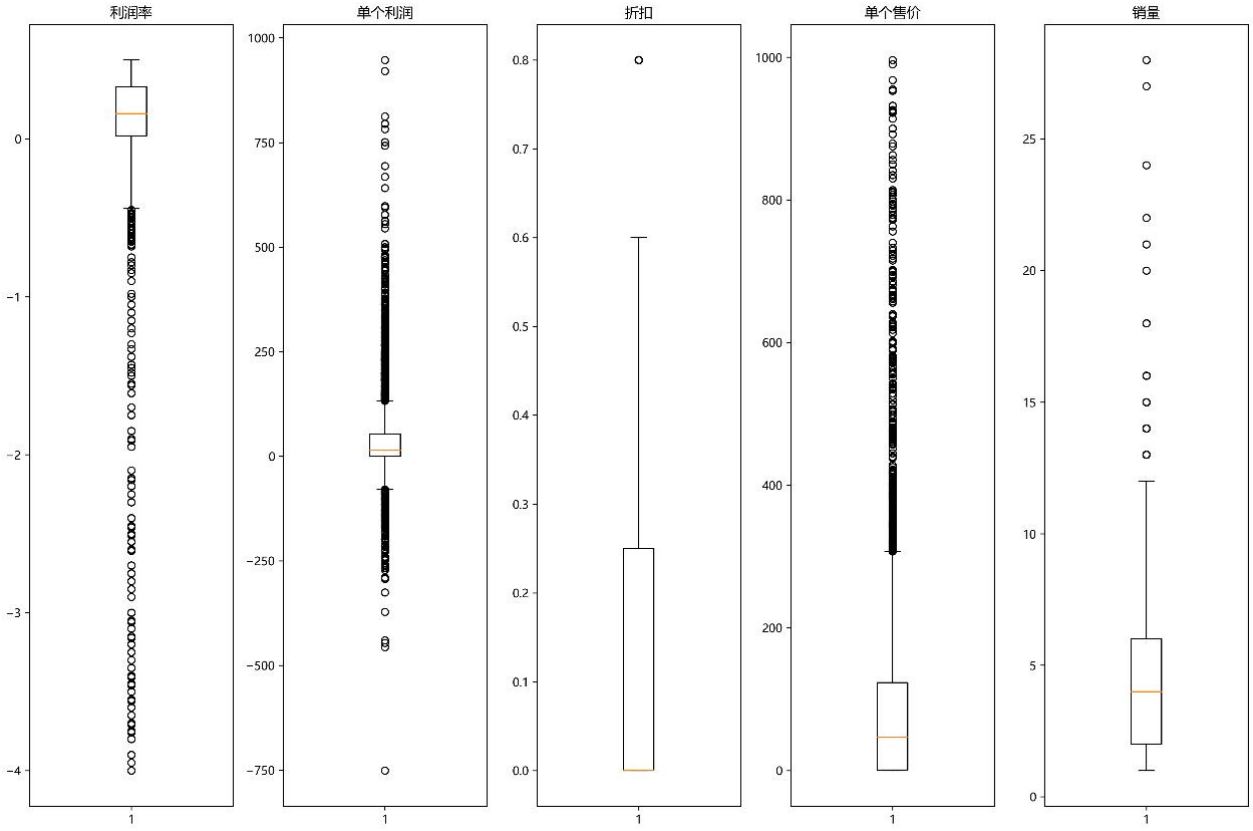
（5）处理噪点

考虑到数据集中可能存在不合理的数据，因此绘制数值型变量的箱型图来进行观察，类型变量将在之后的步骤中进行处理：

使用以下代码绘制箱型图：

```
columns = ['利润率', '单个利润', '折扣', '单个售价', '销量']
plt.figure(figsize=(15, 10))
for i, column in enumerate(columns, 1):
plt.subplot(1, len(columns), i)
plt.boxplot(df[column])
plt.title(column, fontproperties=font_prop)
plt.tight_layout()
plt.show()
```

输出如下：

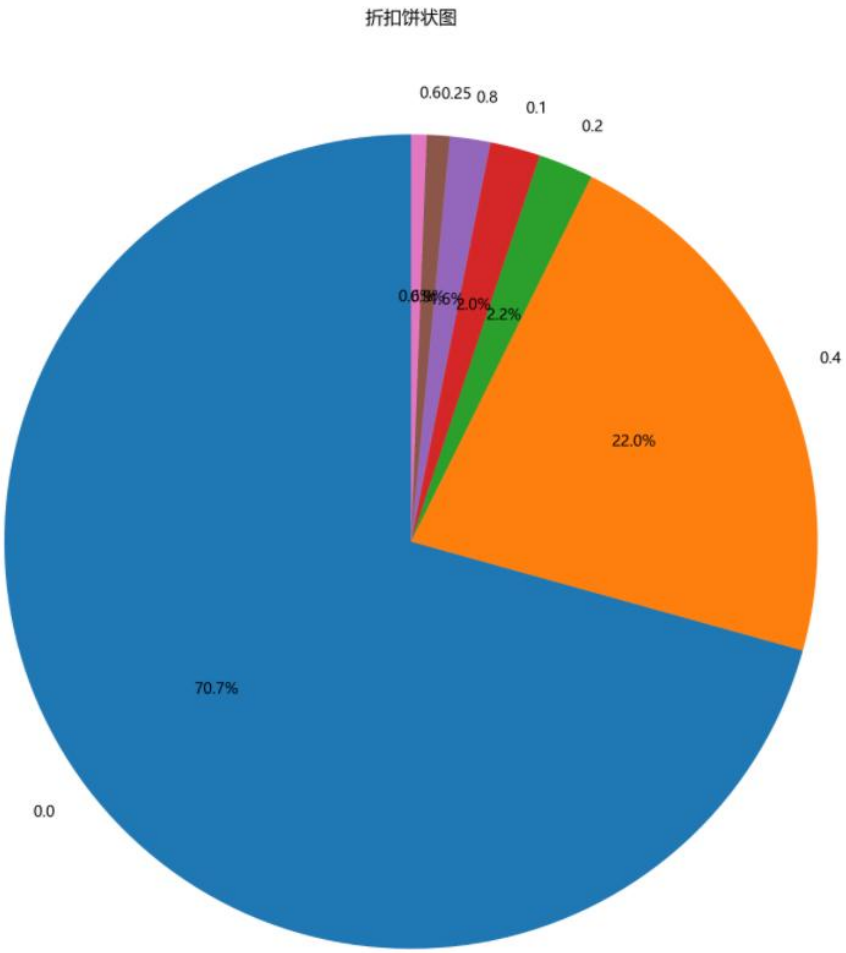


由上图可以看出，在折扣这一特征中出现了 0.8 这个非常突兀的值。根据日常生活经验可知这种程度的折扣应属于特殊情况，在正常情况下折扣力度不应如此大，这种折扣也是导致其他特征出现较多离群值的原因。

使用以下代码绘制折扣饼状图观察各个折扣所占比例。

```
value_counts = df['折扣'].value_counts()
labels = value_counts.index
sizes = value_counts.values
plt.figure(figsize=(12, 12))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('折扣饼状图')
plt.show()
```

输出如下：

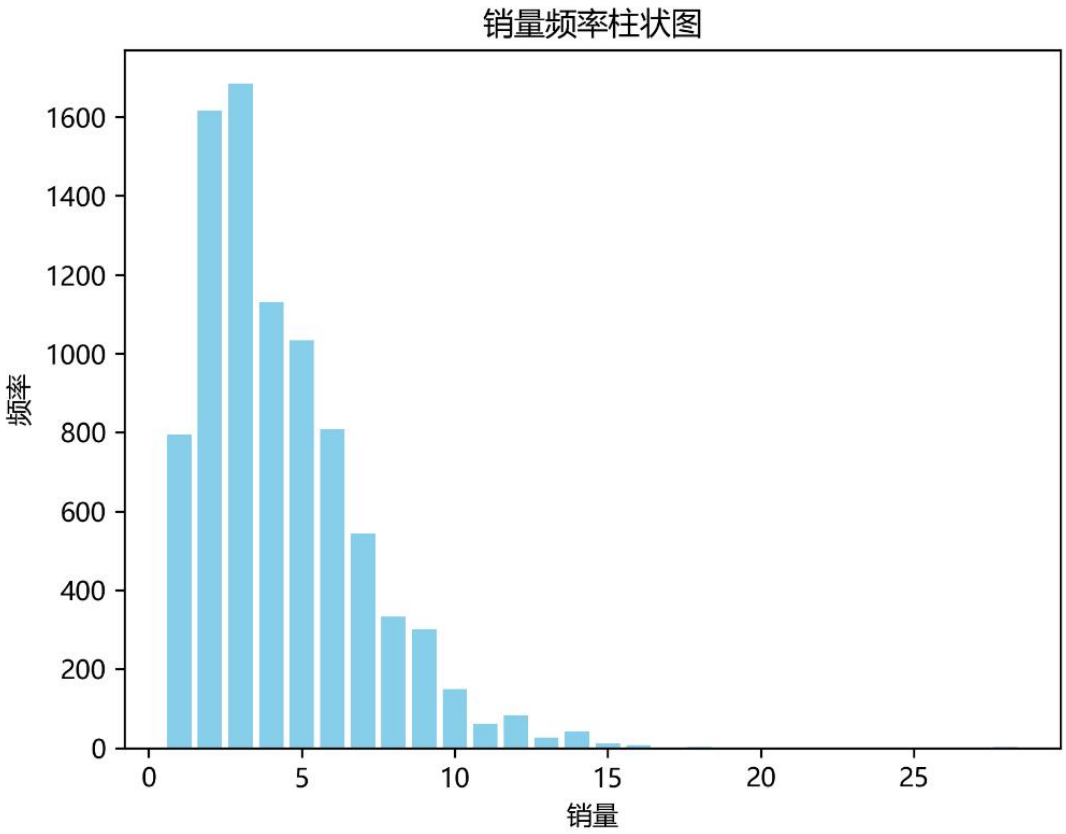


由上图可以看出，0.8 折所占比例很小，仅有 1.6%。再加上面的分析可以得出其属于噪点并去除。

此外，观察到销量这一特征中也存在很多离群点，使用下面的代码绘制销量频率柱状图：

```
value_counts = df['销量'].value_counts()
plt.bar(value_counts.index, value_counts.values, color='skyblue')
plt.xlabel('销量')
plt.ylabel('频率')
plt.title('销量频率柱状图')
plt.show()
```

输出为：

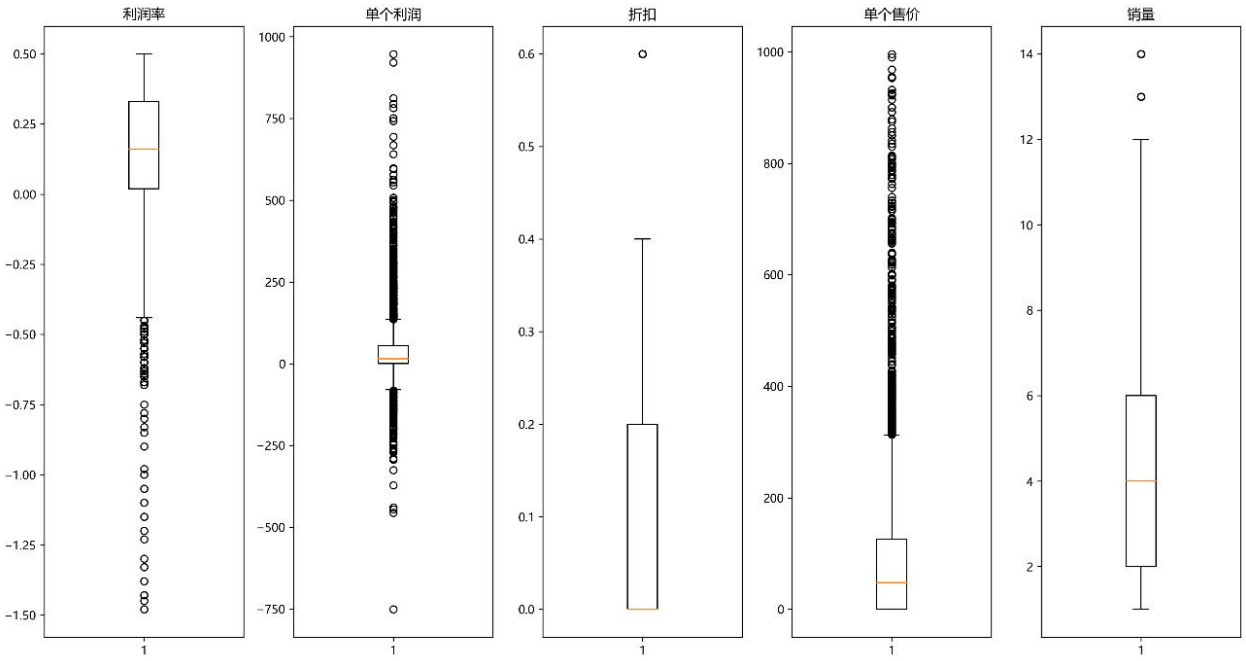


可以看出在销量大于 14 时，数据极少，几乎可以忽略不计。这说明绝大多数商品的销量都不会超过 14，为了防止这些极少数但是数值又较大的点对模型的影响，故将其作为噪点去除。

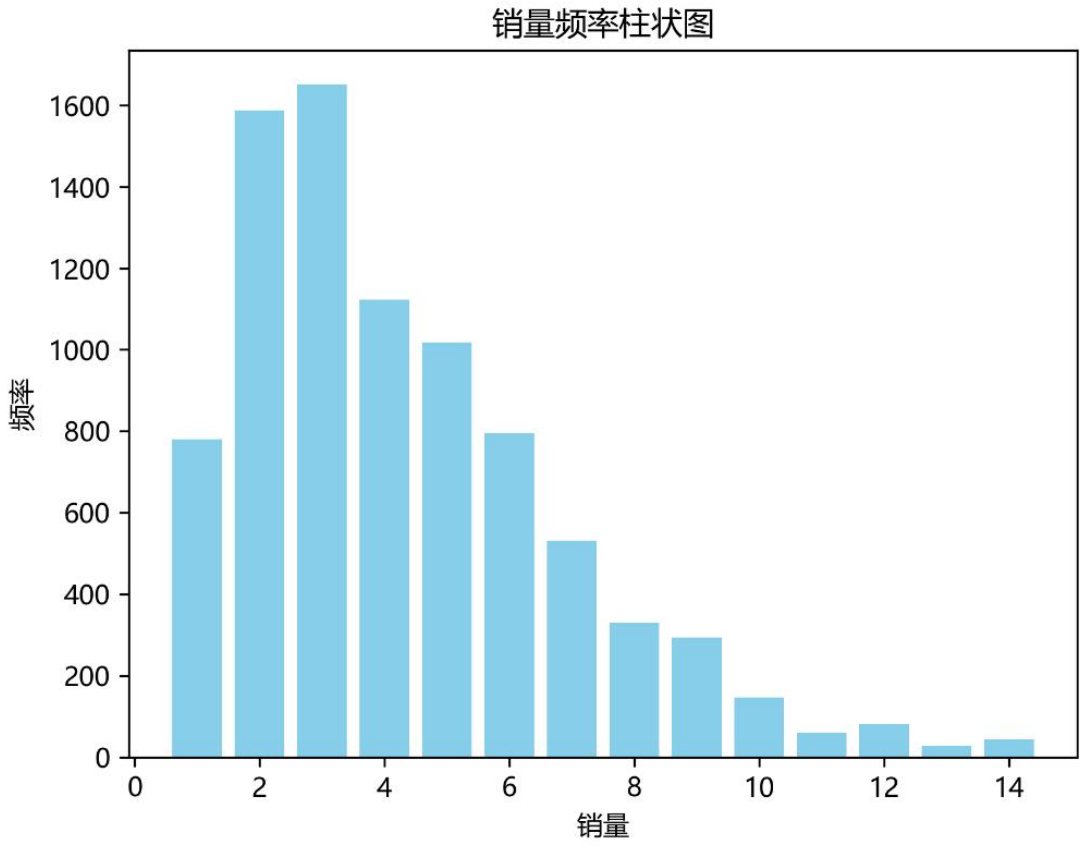
使用以下代码去除噪点：

```
df.drop(df[df['折扣'] == 0.8].index, inplace=True)
df.drop(df[df['销量'] > 14].index, inplace=True)
```

之后绘制箱型图如下：



绘制销量频率图如下：



虽然仍存在离群值，但总体上来看数据比之前更为合理。

（6）数值化：

目前得到的数据中还有很多类型变量，需要将其转换为数值类型。

使用以下代码获取每个特征中类型的数量：

```
print('制造商种类：' + str(df['制造商'].nunique()))
print('子类别种类：' + str(df['子类别'].nunique()))
print('类别种类：' + str(df['类别'].nunique()))
print('邮寄方式种类：' + str(df['邮寄方式'].nunique()))
print('产品种类：' + str(df['产品'].nunique()))
print('标签种类：' + str(df['标签'].nunique()))
```

输出如下：

制造商种类：72
子类别种类：17
类别种类：3
邮寄方式种类：4
产品种类：102
标签种类：30

由上面的输出可以看出，有的特征类型变量种类较多如产品种类有 102 种字符串，而有的则较少如类别种类仅有 3 种字符串。考虑到独热编码虽然不会损失原有信息，但对于种类较多的类型变量会引发维度爆炸；而目标编码可能会丢失一些数据原本的信息，但不会引入更多的维数。故采用独热编码和目标编码混合的方式进行数值化，对于类别较少的类型变量尽量采用独热编码，而对于类别较多的类型变量采用目标编码。

使用以下代码对类别和邮寄方式采用独热编码，对其他特征采用目标编码：

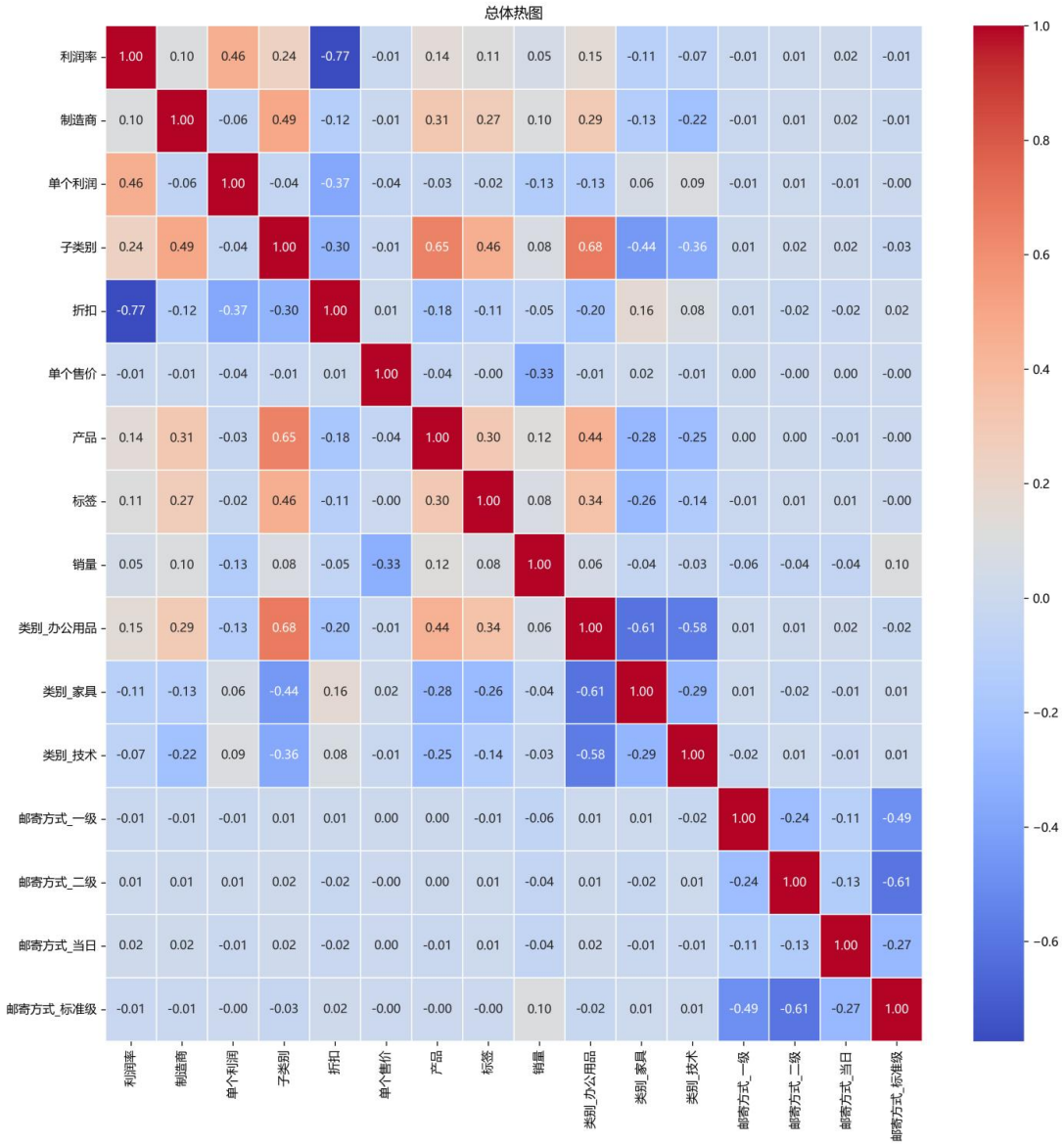
```
df = pd.get_dummies(df, columns=['类别', '邮寄方式'])
encoder = TargetEncoder()
columns_to_encode = ['制造商', '子类别', '产品', '标签']
for column in columns_to_encode:
df[column] = encoder.fit_transform(df[column], df['销量'])
```

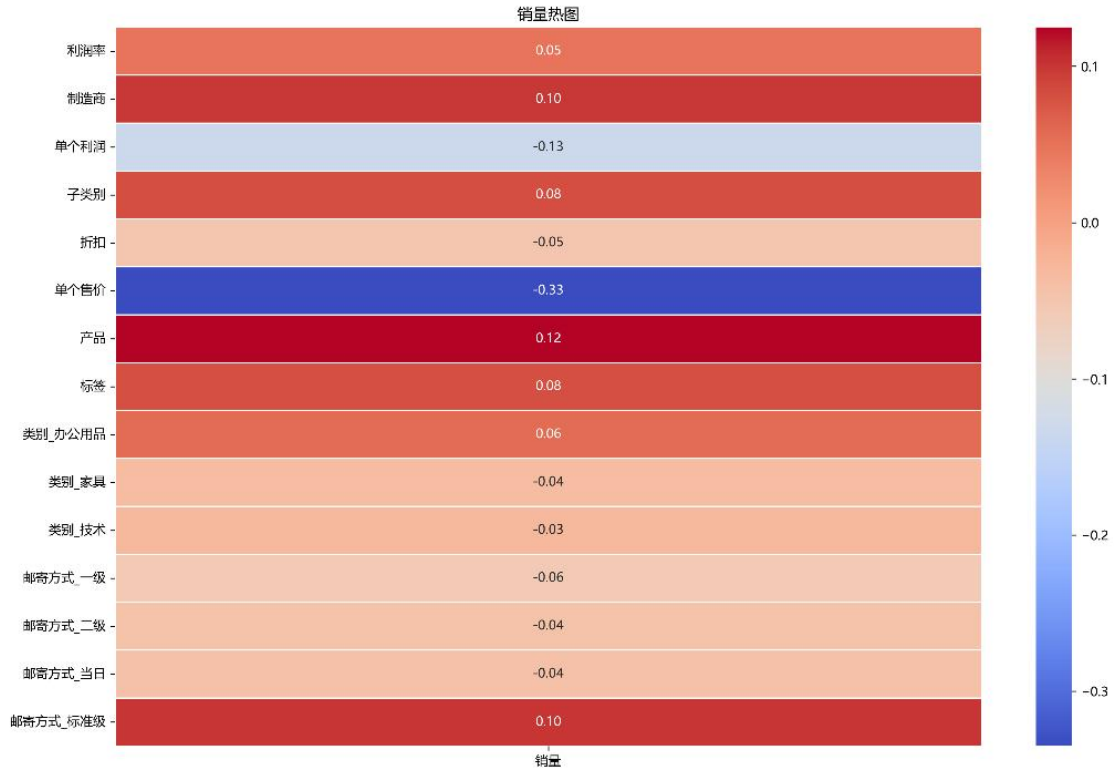
处理后的数据如下（为方便展示均截取两位小数）：

利润率	制造商	单个利润	子类别	折扣	单个售价	产品	标签	销量	类别_办公用品	类别_家具	类别_技术	邮寄方式_一级	邮寄方式_二级	邮寄方式_当日	邮寄方式_标准级
-0.47	4.18	-30.55	4.27	0.4	65	4.39	4.23	2	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
0.34	4.42	21.5	4.47	0	62.5	4.35	4.33	4	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
0.13	4.48	2	4.46	0.4	16	4.83	4.45	5	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

（7）进一步特征选择

首先通过绘制热图观察目前数据中各特征之间的相关性





由于绝对值小于 0.1 的相关性被认为是非常弱的，故为了降低特征维度，提高模型的泛化能力，可尝试去掉绝对值小于 0.1 的特征来进行训练。但由于热图计算的是线性相关性，而各种特征之间的关系可能不止线性，故不可贸然地直接去掉。在本项目中同时训练这两种数据，比较并选择更优的模型和数据。使用以下代码创建新的数据表并分别命名为 feature_all.xlsx 和 feature_selected.xlsx。分别对应未去除相关性小于 0.1 的特征的数据和去除相关性小于 0.1 的特征的数据。值得注意的是，在去除特征后需要再计算一次销量并重新去除重复和离群数据（经观察离群值仍为销量大于 14 的数据）：

```
columns_to_drop = ['利润率', '子类别', '折扣', '标签', '类别_办公用品', '类别_家具', '类别_技术', '邮寄方式_一级','邮寄方式_二级','邮寄方式_当日']
df = df.drop(columns=columns_to_drop, axis=1)
df['销量'] = df.groupby(['制造商', '产品', '单个利润', '单个售价', '邮寄方式_标准级'])['销量'].transform('sum')
df = df.drop_duplicates()
df.drop(df[df['销量'] > 14].index, inplace=True)
df.to_excel('feature_selected.xlsx', index=False)
```

上面代码得到的两个文件即为接下来要进行模型训练的数据。

三、模型训练与评估

（1）选定评估标准

回归分析的评价指标有很多，常见的有均方误差，均方根误差，平均绝对误差，确定系数等。这里选择均方误差 MSE 和确定系数 R^2 作为评价指标。

MSE 是预测误差的平方和的平均值，这种平方操作可以消除正负误差的影响，同时有利于数学计算和优化。MSE 对于异常值具有较强的敏感性，这意味着它会更强烈地惩罚那些预测误差较大的数据点，使得模型更关注预测值与实际值的接近程度。

R^2 衡量了模型对因变量变化的解释程度，其值在 0 到 1 之间，越接近 1 表示模型能够更好地解释目标变量的变化。这使得 R^2 对于向非专业人员解释模型性能更加直观。R^2 提供了一种相对比较不同模型拟合能力的方法。在比较不同模型时，可以通过 R^2 值的大小来判断哪个模型更好地拟合了数据。

实际进行模型训练时，将经过上面预处理的数据按照 8:2 分为训练集和测试集。在使用训练集训练模型时，采用网格搜索的方式寻找最优超参数，网格搜索采用负均方误差作为搜索指标，并采用 5 折交叉验证评价模型。模型训练完成后，使用测试集来检验模型的性能，采用 MSE 和 R^2 结合的方式进行评价，并选出最佳的模型。

（2）岭回归模型

岭回归（Ridge Regression）是一种线性回归的改进方法，它通过在损失函数中添加一个正则化项，可以有效地应对多重共线性的问题。

本项目使用的岭回归 API 为：sklearn.linear_model.Ridge。网格搜索 API 为：sklearn.model_selection.GridSearchCV。通过如下代码引入这些库：

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
```

此外，为了加速模型的训练，使用了 intel 的 sklearn 扩展。之后的几个模型也尽量使用该扩展来加速模型的训练。

要启动该加速扩展，只需在导入需要的 sklearn API 之前引入以下代码：

```
from sklearnex import patch_sklearn
patch_sklearn()
```

Ridge 的主要超参数如下：

（1）alpha：控制正则化的强度。

在本项目中超参数的搜索范围如下：

```
param_grid = {
'alpha': [0.1, 0.5, 1.0, 2.0, 5, 6, 7, 8 , 9, 10, 11, 12, 13, 14, 15.0,
17.5, 20.0]
}
```

上面的搜索范围是经过多次测试得到的。一开始为了快速探索，搜索范围可以比较小，然后训练模型并观察搜索出的最佳超参数。最佳超参数如果是在搜索范围某个项的边界处，则适当扩大该项在此边界处的搜索范围。如果是在搜索范围某个项的中间某个值，则在该值的周围细化搜索范围。然后继续训练模型并观察搜索出的最佳超参数，直到该超参数在搜索范围中比较符合预期。

模型训练的代码如下：

```
param_grid = {
'alpha': [0.1, 0.5, 1.0, 2.0, 5, 6, 7, 8 , 9, 10, 11, 12, 13, 14, 15.0,
17.5, 20.0]
}
df_all = pd.read_excel('feature_all.xlsx')
X_all = df_all.drop(['销量'],axis=1)
y_all = df_all['销量']
X_all, y_all = np.array(X_all), np.array(y_all)
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all,
test_size=0.2, random_state=42)
model = Ridge()
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

```
print("最佳超参数:", grid_search.best_params_)
print("最佳模型性能 (负均方误差):", grid_search.best_score_)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("在测试集上的均方误差:", mse)
r2 = r2_score(y_test, y_pred)
print("在测试集上的 R²值:", r2)
```

上面代码的主要思路为：首先定义超参数的搜索范围，然后读取之前预处理好的数据并转化为 numpy array 以便于模型的使用。之后划分数据集为训练集和测试集，按照 8:2 的比例划分。之后使用上面所述的库构建模型并执行网格搜索。由于网格搜索通过最大化评价指标的值来选择最佳超参数，而均方误差是一个损失函数，我们希望最小化它，因此在网格搜索中使用负均方误差，使其成为一个需要最大化的指标。网格搜索采用 5 折交叉验证，这与之前的训练集与测试集 8:2 相对应。之后在测试集上评估模型，计算并输出在测试集上的均方误差 MSE 和确定系数 R^2。

由于之后几个模型代码的主要思路均与此类似，基本上只有模型 API 和超参数搜索范围的差别，故在接下来的模型中不再详细介绍代码实现，而是关注超参数搜索范围和模型输出结果。

岭回归模型训练结果如下：

```
==================ALL==================
最佳超参数: {'alpha': 10}
最佳模型性能 (负均方误差): -5.332307979513909
在测试集上的均方误差: 5.9785274330344835
在测试集上的 R²值: 0.171708125740204
==================ALL==================
================SELECTED================
最佳超参数: {'alpha': 14}
最佳模型性能 (负均方误差): -5.995641142024539
在测试集上的均方误差: 5.989906625003307
在测试集上的 R²值: 0.14259512363079085
================SELECTED================
```

上面的 all 指的是未去掉相关性小于 0.1 的特征的结果，selected 指的是去掉相关性小于 0.1 的特征的结果，在数据预处理中步骤（7）中有具体描述，关于此项之后也不再赘述。

（3）Elastic Net 回归模型

Elastic Net 回归是一种结合了 Lasso 回归（L1 正则化）和 Ridge 回归（L2 正则化）的线性回归方法。它综合了 Lasso 回归的稀疏性和 Ridge 回归的正则化特性，通过两种正则化项来控制模型的复杂度。

本项目使用的 Elastic Net 回归 API 为：sklearn.linear_model.ElasticNet。

ElasticNet 的主要超参数如下：

（1）alpha：Elastic Net 正则化惩罚项的权重，结合了 L1 和 L2 正则化项。

（2）l1_ratio：L1 和 L2 正则化项的混合比例。

（3）max_iter：迭代的最大次数。

在本项目中超参数的搜索范围如下：

```
param_grid = {
'alpha': [0.001, 0.002, 0.003, 0.004, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 2.0],
'l1_ratio': [0.001, 0.002, 0.003, 0.004, 0.005, 0.01, 0.05, 0.1, 0.5],
'max_iter': [500, 1000, 1500],
}
```

Elastic Net 回归模型训练结果如下：

```
==================ALL====================
最佳超参数: {'alpha': 0.002, 'l1_ratio': 0, 'max_iter': 500}
最佳模型性能 (负均方误差): -5.332313601393271
在测试集上的均方误差: 5.978555438754639
在测试集上的 R²值: 0.171704245702737
==================ALL====================
================SELECTED==================
最佳超参数: {'alpha': 0.003, 'l1_ratio': 0, 'max_iter': 500}
最佳模型性能 (负均方误差): -5.9956419194053465
在测试集上的均方误差: 5.989988428636153
在测试集上的 R²值: 0.1425834141271045
================SELECTED==================
```

（4）决策树回归模型

决策树是一种非常灵活的算法，它通过对特征空间进行递归划分来建立模型。在每个节点上，决策树根据某个特征将数据集划分为两个子集，然后在子集上递归地重复这个过程，直到达到停止条件。在预测时，样本通过树的分支最终到达叶子节点，叶子节点的值即为预测值。决策树算法易于理解，构建较快且容错能力较强。但决策树模型非常容易过拟合，导致泛化能力不强。

本项目使用的决策树回归 API 为：sklearn.tree.DecisionTreeRegressor。

DecisionTreeRegressor 的主要超参数如下：

（1）max_depth：决策树的最大深度。

（2）min_samples_split：节点分裂的最小样本数。

在本项目中超参数的搜索范围如下：

```
param_grid = {
'max_depth': [3, 4, 5, 10, 15, 20, 25, 30],
'min_samples_split': [2, 5, 10, 15, 20, 25, 30],
}
```

决策树回归模型训练结果如下：

```
==================ALL====================
最佳超参数: {'max_depth': 6, 'min_samples_split': 30}
最佳模型性能 (负均方误差): -5.049975963124824
在测试集上的均方误差: 5.690257670730443
在测试集上的 R²值: 0.21164630523101025
==================ALL====================

================SELECTED==================
最佳超参数: {'max_depth': 6, 'min_samples_split': 35}
最佳模型性能 (负均方误差): -5.608856121820231
在测试集上的均方误差: 5.5513152742186715
在测试集上的 R²值: 0.20537579559091468
================SELECTED==================
```

（5）随机森林回归模型

随机森林回归是一种基于决策树的集成学习方法。它通过构建多个决策树，并将它们的预测结果进行平均来提高模型的性能。

本项目使用的随机森林回归API为：sklearn.ensemble.RandomForestRegressor

RandomForestRegressor的主要超参数如下：

（1）n_estimators：森林中树的数量。

（2）max_depth：每棵树的最大深度。

（3）min_samples_split：节点分裂的最小样本数。

在本项目中超参数的搜索范围如下：

```
param_grid = {
'n_estimators': [100, 200, 300], # 决策树的数量
'max_depth': [5, 6, 7, 8, 9, 10, 15], # 决策树的最大深度
'min_samples_split': [10, 15, 20, 25, 30]
}
```

随机森林回归模型训练结果如下：

```
==================ALL==================
最佳超参数: {'max_depth': 8, 'min_samples_split': 40,
'n_estimators': 150}
最佳模型性能 (负均方误差): -4.814321968859235
在测试集上的均方误差: 5.4084415858253445
在测试集上的R²值: 0.2506903634505646
==================ALL==================

================SELECTED=================
最佳超参数: {'max_depth': 7, 'min_samples_split': 40,
'n_estimators': 150}
最佳模型性能 (负均方误差): -5.363082118353599
在测试集上的均方误差: 5.3976326179569085
在测试集上的R²值: 0.22737417839410834
================SELECTED=================
```

（6）XGBoost 回归模型

XGBoost（eXtreme Gradient Boosting）是一种梯度提升树算法，被广泛用于回归和分类问题。XGBoost 通过组合多个弱学习器来构建一个强大的模型。

本项目使用的 XGBoost 回归 API 为：xgboost.XGBRegressor

XGBRegressor 的主要超参数如下：

（1）learning_rate：学习率，用于控制每棵树对最终模型的贡献。

（2）n_estimators：弱学习器的数量。

（3）max_depth：每棵树的最大深度。

在本项目中超参数的搜索范围如下：

```
param_grid = {
'learning_rate': [0.01, 0.03, 0.05, 0.07, 0.1, 0.2],
'n_estimators': [25, 50, 75, 100, 150, 200],
'max_depth': [3, 4, 5, 6, 7, 10, 15],
}
```

XGBoost 回归模型训练结果如下：

```
==================ALL==================
最佳超参数: {'learning_rate': 0.07, 'max_depth': 3, 'n_estimators': 150}
最佳模型性能 (负均方误差): -4.7580901491048575
在测试集上的均方误差: 5.335830892275311
在测试集上的 R²值: 0.2607501730149635
==================ALL==================

================SELECTED=================
最佳超参数: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 75}
最佳模型性能 (负均方误差): -5.355241783672942
在测试集上的均方误差: 5.457553013064786
在测试集上的 R²值: 0.218797076583371
================SELECTED=================
```

（7）梯度提升树回归模型

梯度提升树（Gradient Boosting Trees）是一种集成学习算法，利用加法模型和前向分步算法实现学习的优化过程。在回归问题中，该算法通过不断减小残差的梯度来逐步逼近目标值。

本项目使用的梯度提升树回归API为：sklearn.ensemble.GradientBoostingRegressor

GradientBoostingRegressor的主要超参数如下：

（1）n_estimators：弱分类器的数量。

（2）learning_rate：学习率，用于控制每棵树对最终模型的贡献。

（3）max_depth：每棵树的最大深度。

（4）min_samples_split：节点分裂所需的最小样本数。

在本项目中超参数的搜索范围如下：

```
param_grid = {
'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
'n_estimators': [50, 100, 150, 200],
'max_depth': [3, 4, 5, 6, 7, 10, 15, 20],
'min_samples_split': [2, 5, 10, 20, 30, 40],
}
```

梯度提升树回归模型训练结果如下：

```
==================ALL==================
最佳超参数: {'learning_rate': 0.05, 'max_depth': 3,
'min_samples_split': 20, 'n_estimators': 200}
最佳模型性能 (负均方误差): -4.73779156173512
在测试集上的均方误差: 5.304333683406015
在测试集上的R²值: 0.265113936911763
==================ALL==================
================SELECTED=================
最佳超参数: {'learning_rate': 0.15, 'max_depth': 3,
'min_samples_split': 10, 'n_estimators': 50}
最佳模型性能 (负均方误差): -5.308665210217837
在测试集上的均方误差: 5.395155612609416
在测试集上的R²值: 0.22772874092683093
================SELECTED=================
```

（8）LightGBM 回归模型

LightGBM（Light Gradient Boosting Machine）是一种梯度提升框架，专注于高效性和分布式计算。它是一种基于树的学习算法，对于大规模数据集和高维特征的情况下表现出色。

本项目使用的 LightGBM 回归 API 为：lightgbm.LGBMRegressor

LGBMRegressor 的主要超参数如下：

（1）n_estimators：树的数量。

（2）learning_rate：学习率，控制每棵树的影响程度。

（3）max_depth：每棵树的最大深度。

（4）min_child_samples：每个叶子节点所需的最小样本数。

在本项目中超参数的搜索范围如下：

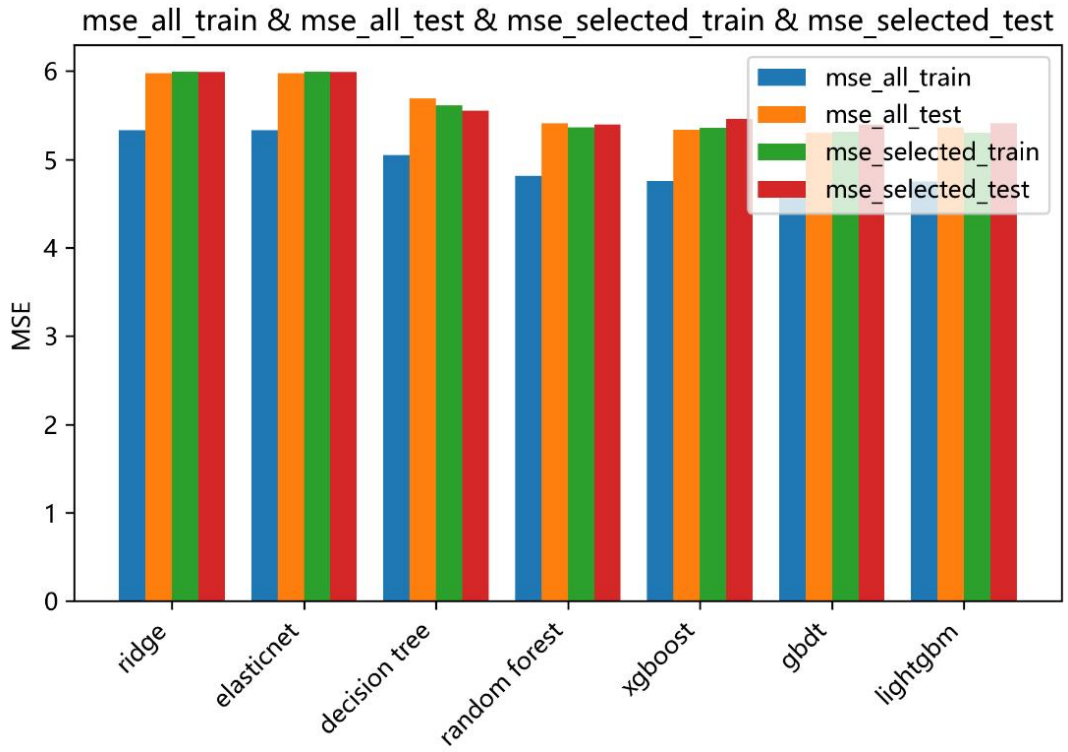
```
param_grid = {
'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
'n_estimators': [50, 75, 100, 150],
'max_depth': [3, 4, 5, 6, 7, 10, 15, 20, 25],
'min_child_samples': [15, 20, 25, 30, 35, 40],
}
```

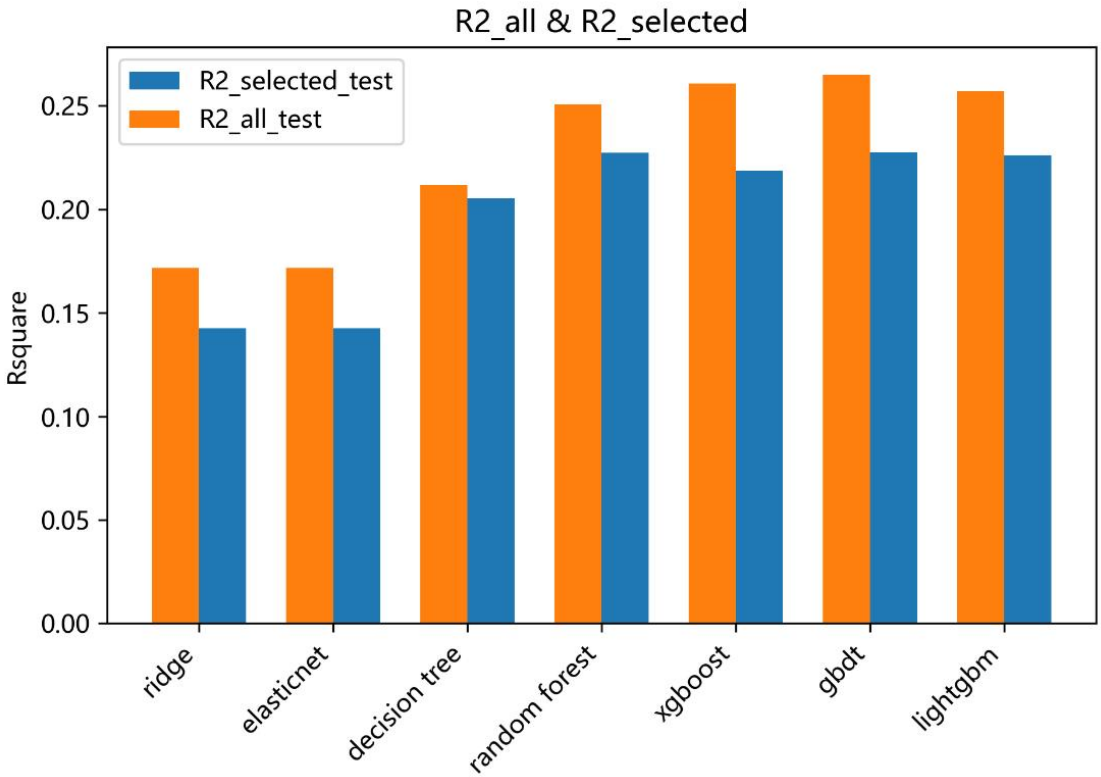
LightGBM 回归模型训练结果如下：

```
==================ALL==================
最佳超参数: {'learning_rate': 0.1, 'max_depth': 3,
'min_child_samples': 30, 'n_estimators': 100}
最佳模型性能 (负均方误差): -4.750771687261698
在测试集上的均方误差: 5.362673731712184
在测试集上的 R²值: 0.25703124623296525
==================ALL==================
================SELECTED=================
最佳超参数: {'learning_rate': 0.1, 'max_depth': 3,
'min_child_samples': 15, 'n_estimators': 100}
最佳模型性能 (负均方误差): -5.303406700294638
在测试集上的均方误差: 5.406304908421738
在测试集上的 R²值: 0.22613281277702235
================SELECTED=================
```

四、分析并选定模型

对上面算法的结果进行统计并画柱状图如下：





上图中的 mse_all_train 和 mse_all_test 是使用未经进一步筛选特征的数据训练出的模型结果，分别对应在训练集上的性能和在测试上的性能；mse_selected_train 和 mse_selected_test 是使用经进一步筛选特征的数据训练出的模型结果，分别对应在训练集上的性能和在测试上的性能。R2_all 和 R2_selected 分别对应未筛选特征的数据和筛选特征的数据在测试集上的拟合效果。

可以看出，对于未经筛选特征的数据，其训练 MSE 比测试 MSE 小很多，这说明过拟合现象较为严重；而经过筛选特征的数据训练误差和测试误差相近，说明模型的泛用性更强。但是由于筛选特征后可能丢失了一些信息，虽然泛用性更强，但对于有些模型其性能与未经筛选特征的数据差别不大。即使如此，这一步对于特征的筛选也是有必要的，可以看出对于决策树回归模型，未经筛选特征后的数据训练出的 MSE 比经筛选特征的 MSE 要明显大上一些，这也与之前提到的决策树容易过拟合相关，如果采用决策树回归模型的话，使用经过筛选特征的数据模型性能和泛用性都会更强。

此外，对于 R^2 而言，经进一步筛选特征的数据训练出的模型拟合效果相对较差，推测这也是由于筛选特征丢失了一些信息导致的。

总体上来看，无论是在测试集上的 MSE 还是 R^2 上，梯度提升树的性能都是最好的。同时注意到在训练的过程中 GBDT 的训练速度是最慢的，上面的网格搜索进行了 4.5 个小时，训练时间远长于其他模型。而 xgboost 和 lightbm 作为梯度提升树优化的算法尤其是在速度上优化的算法，训练速度明显快了很多，而性能也只是略差。故在数据量比较大和对时间比较敏感的情况下，可以采用 xgboost 和 lightgbm 来较快地得到结果。在本项目中由于 GBDT 的性能最佳，因而选用 GBDT 训练出的模型作为最终模型。

最后值得注意的是，模型的性能并不算优秀，这与数据有着密切的关系。在将数量转化成销量的这一步骤上数据从 9935 减少到 8638，只减少了很小的一部分比例。这说明每种产品的订单在数据集中出现的次数很少，几乎只有一两次，根据日常经验推断这显然是不合理的。对于一个超市来说，某种产品在 4 年内的订单数量极少会只有一两次。在本数据集中，制造商种类众多，有 72 种，组合出的产品种类更多，而每种产品的订单数量却很少，因此对于每种产品内部的区分信息就很少。

因此在机器学习中，训练数据具有至关重要的地位，它对模型的性能和泛化能力有着显著的影响。如果能提供更多更全面的历史订单数据，有理由相信模型的性能和泛化能力均会有明显的提升。

五、总结

本项目通过以下工作流程：

1. 问题分析
2. 数据预处理
3. 模型训练与评估
4. 分析并选定模型

构建了一个 Sales Volume Forecast 销量预测模型，将机器学习应用到生产实践的过程中并基于 GBDT 回归算法为超市提供了一个基于历史数据的决策支持工具。模型预测结果可以用于帮助超市管理人员制定合理的库存管理策略，提高超市的运营效率和利润。

在实践中，超市管理人员应尽可能多地收集历史数据以提高模型的性能。同时还应注意定期更新数据，逐渐淘汰老的数据并使用新的数据进行训练与测试，这样该模型才能更好地适应环境的变化，提供更为准确的预测结果。

参考文献:

[1] 赵卫东，董亮.机器学习 = Machine learning[M]. 北京: 人民邮电出版社, 2022.

[2] 赵卫东，董亮.Python 机器学习实战案例[M]. 北京: 清华大学出版社, 2022.