**华中科技大学**

**计算机科学与技术学院**

**《机器学习》结课报告**



专 业： 数据科学与大数据技术

班 级： 大数据2101班

学 号： U202115652

姓 名： 李嘉鹏

成 绩：

指导教师： 何琨

**完成日期：2023年5月21日**

目录

[**1 机器学习第六次实验：鸢尾花三分类问题 2**](#_Toc135567696)

[1.1 实验要求 2](#_Toc135567697)

[1.2 程序设计与实现 2](#_Toc135567698)

[1.3 实验环境与平台 3](#_Toc135567699)

[1.4 结果与分析 3](#_Toc135567700)

[1.5 个人体会 4](#_Toc135567701)

[**2 机器学习第六次实验：MNIST手写数据集识别 5**](#_Toc135567702)

[2.1 实验要求 5](#_Toc135567703)

[2.2 程序设计与实现 5](#_Toc135567704)

[2.3 实验环境与平台 6](#_Toc135567705)

[2.4 结果与分析 6](#_Toc135567706)

[2.5 个人体会 7](#_Toc135567707)

[**3 机器学习大作业：Kaggle 2017年房价预测（House Prices - Advanced Regression Techniques） 8**](#_Toc135567708)

[3.1 实验要求 8](#_Toc135567709)

[3.2 设计与实现 8](#_Toc135567710)

[3.2.1 数据集特点分析 8](#_Toc135567711)

[3.2.2 数据清洗与预处理 11](#_Toc135567712)

[3.2.3 将类别属性数值化并利用随机森林实现特征选择 12](#_Toc135567713)

[3.2.4 数据的相关性分析、方差分析与标准化 16](#_Toc135567714)

[3.2.5 通过主成分分析PCA进行降维 18](#_Toc135567715)

[3.2.6 利用Kmeans算法实现聚类（二分类） 19](#_Toc135567716)

[3.2.7 训练、线性回归并得到预测结果 21](#_Toc135567717)

[3.3 实验环境与平台 21](#_Toc135567718)

[3.4 结果与分析 21](#_Toc135567719)

[3.5 个人体会 22](#_Toc135567720)

附录一 实验六代码 23

[附录二 大作业代码 30](#_Toc135567720)

# 1 机器学习第六次实验：鸢尾花三分类问题

# 实验要求

对于鸢尾花的三分类问题，需要对Iris数据集进行训练和测试，并根据提示补全Python代码。数据集包含150个样本，分为3类，分别是Iris-setosa、Iris-versicolor和Iris-virginica，其中每类包含50个样本，而每个样本数据又包含4个属性特征。因此可以通过这4个属性特征预测鸢尾花属于三个种类中的哪一类。最终需要准确率达到95%以上。

# 程序设计与实现

首先观察数据集特征，如图1-1、图1-2所示，可以发现150个鸢尾花数据集被分为三组，每组包含50个数据，每个数据前四项为其特征。

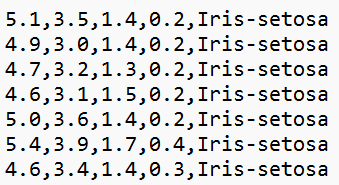
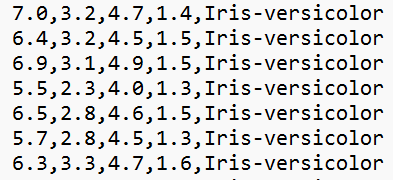
 

图1-1：Iris-setosa数据集节选 图1-2：Iris-versicolor数据集节选

首先，要获取每个类的前五个样本信息。使用load\_iris函数读取鸢尾花数据集，将其特征向量存储在numpy数组x\_data中。接着，使用同一函数的target属性读取每个样本对应的标签（即类别），并将其变形为一个150行×1列的数组y\_data。最后，使用hstack函数将x\_data和y\_data按列连接在一起，得到包含全部150个样本信息的二维数组data。其中第1-4列分别对应鸢尾花四个特征，第5列对应类别标签。代码如下：

|  |
| --- |
| x\_data = load\_iris().data y\_data = load\_iris().target.reshape(150, 1) data = np.hstack((x\_data, y\_data)) |

接下来分别将Iris-setosa、Iris-versicolor、Iris-virginica对应为0、1、2三个标签。这里，可以定义一个字典label\_dict，用来存储类别与数字标签之间的映射关系，键值0、1、2分别对应三个类别。代码如下：

|  |
| --- |
| label\_dict = {0: 0, 1: 1, 2: 2} |

然后通过随机选择的方式将数据集分成训练集和验证集两部分，其中训练集占总数据集的80%。先用np.random.randint函数生成长为len(Y) \* 0.8、包含了从0到len(Y)-1之间的整数的数组，也就是将数据集按照8:2的比例划分成训练集和验证集，该数组的元素代表了选出来的训练集样本在原数据集中所对应的下标索引。最后生成验证集的下标索引数组test\_idx。这一部分的代码如下：

|  |
| --- |
| train\_idx = np.random.randint(0, len(Y), int(len(Y) \* 0.8)) test\_idx = np.array([i **for** i **in** range(0, len(Y)) **if** i **not in** train\_idx]) |

下面将测试数据转换为MindSpore数据集。将X\_test和Y\_test打包成一个列表并赋值给变量XY\_test，并使用GeneratorDataset函数创建一个MindSpore数据集ds\_test，该数据集的两个字段分别对应X\_test和Y\_test。最后仿照原始程序，使用shuffle函数打乱ds\_test，利用大小为120的缓冲区进行存储，并使用batch函数对ds\_test进行分批处理，每批大小为32。代码如下：

|  |
| --- |
| XY\_test = list(zip(X\_test, Y\_test)) ds\_test = dataset.GeneratorDataset(XY\_test, [**'x'**, **'y'**]) ds\_test = ds\_test.shuffle(buffer\_size=120).batch(32, drop\_remainder=**True**) |

接下来，使用交叉熵损失计算。定义损失函数SoftmaxCrossEntropyWithLogits，采用均值作为损失函数的缩放因子来计算平均损失。同时使用动量优化器优化参数，learning\_rate=0.05表示学习率设置为0.05，即每次更新参数时的步长大小；momentum=0.9表示动量设置为0.9，即在参数更新时保持历史梯度的加权平均。代码如下：

|  |
| --- |
| loss = nn.SoftmaxCrossEntropyWithLogits(sparse=**True**, reduction=**'mean'**) opt = nn.optim.Momentum(net.trainable\_params(), learning\_rate=0.05, momentum=0.9) |

# 1.3 实验环境与平台

本次实验在华为云实验平台完成。在华为云ModelArts平台上创建AI框架为Mindspore-1.5、硬件环境为Ascend 910+ARM的开发环境，并创建好搭建了MindSpore环境的Jupyter Notebook，在线上填入代码运行并测试。

# 1.4 结果与分析

运行程序，使用model.train进行模型的训练，迭代25次，每个迭代周期结束时都对测试集进行验证并输出测试集上的损失函数值，如图1-3所示。最终的准确率为0.98438，损失函数值为0.1665，达到了预期目标，预测效果较好。

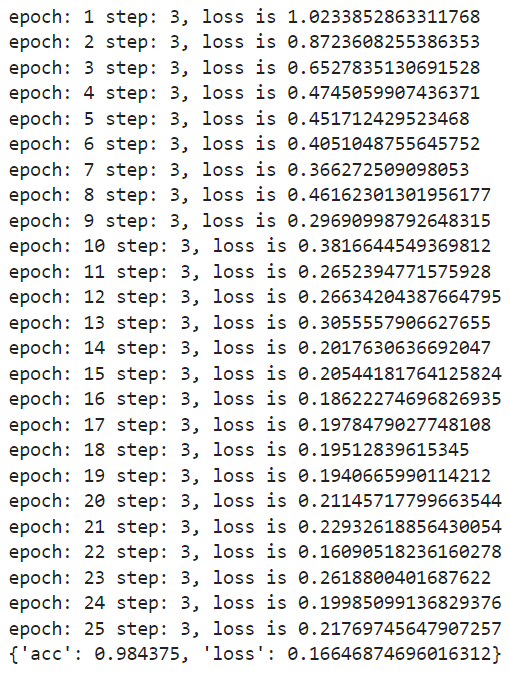


图1-3：鸢尾花三分类问题测试结果

# 1.5 个人体会

这次实验相对来说比较容易实现，主要是要理解题目要我达到什么目的。鸢尾花数据集是机器学习领域公认的入门数据集，在进行鸢尾花三分类问题的实验中，我首先了解了Iris数据集的特征和标签分布情况，并使用Python中的sklearn库进行数据处理和模型训练。通过在MindSpore下使用交叉熵损失计算，得到了一个准确率较高的分类器。

这次实验让我更深入地理解了机器学习的基本原理和实际应用过程，也提高了我对Python编程和数据分析的技能水平。

# 2 机器学习第六次实验：MNIST手写数据集识别

# 2.1 实验要求

对于MNIST手写数字识别，需要对60000个训练集数据（手写数字的图片）进行学习，并在10000个测试集数据上测试，准确率要达到95%以上。

# 2.2 程序设计与实现

首先对原始数据集进行放缩和归一化，将图像的高度和宽度都设为256，并使用除以255的方式对图像进行归一化操作，将其值转换到0~1之间。代码是：

|  |
| --- |
| resize\_height = 32 resize\_width = 32rescale = 1/255. |

接下来对数据集进行打乱（batch）、分批（shuffle）和重复（repeat）操作，以便后续训练模型时使用。调用 shuffle函数对数据集进行打乱，接着调用 batch函数将每一批的数据打包成一个batch，其中drop\_remainder=True表示丢弃最后不足一个batch的样本。最后调用repeat函数对整个数据集重复操作，重复repeat\_size次，这样就可以在训练模型时多次使用该数据集进行迭代更新。代码如下：

|  |
| --- |
| mnist\_ds = mnist\_ds.shuffle(buffer\_size=buffer\_size) mnist\_ds = mnist\_ds.batch(batch\_size, drop\_remainder=**True**) mnist\_ds = mnist\_ds.repeat(repeat\_size) |

建立LeNet5网络结构神经网络，搭建神经网络时定义一些层和模块。首先定义两个卷积层，其中self.conv1的输出通道数为6；self.conv2的输入通道数为6，输出通道数为16，二者的卷积核大小均为5×5。接着定义三个全连接层，其中self.fc1的输入节点数为400，输出节点数为120，采用正态分布初始化；self.fc2的输入节点数为120，输出节点数为84；self.fc3的输入节点数为84，输出节点数为num\_class，即分类数。

最后，定义一个ReLU层self.relu、一个步长为2的最大池化层self.max\_pool2d、一个将输入展平为一维向量的层的池化窗口self.flatten。代码如下：

|  |
| --- |
| self.conv1 = nn.Conv2d(num\_channel, 6, 5, pad\_mode=**'valid'**) self.conv2 = nn.Conv2d(6, 16, 5, pad\_mode=**'valid'**) self.fc1 = nn.Dense(16 \* 5 \* 5, 120, weight\_init=Normal(0.02)) self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02)) self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02)) self.relu = nn.ReLU() self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2) self.flatten = nn.Flatten() |

下面要使用上一步定义好的运算构建前向网络。首先将输入x经过第一个卷积层conv1，使用激活函数进行激活，再经过最大池化操作得到新的特征图x。将这个x再经过第二个卷积层conv2，重复上述操作，得到另一个新的特征图 x。对其进行扁平化操作并转化为向量，方便之后的全连接层进行处理。

然后，将上一步得到的向量x作为输入，依次经过三个全连接层进行线性变换，得到最终的输出结果。这样就实现了神经网络模型的前向传播过程，将输入数据x从输入层经过一系列的卷积、池化和全连接后输出。核心代码是：

|  |
| --- |
| x = self.max\_pool2d(self.relu(self.conv1(x))) x = self.max\_pool2d(self.relu(self.conv2(x))) x = self.flatten(x) x = self.relu(self.fc1(x)) x = self.relu(self.fc2(x)) x = self.fc3(x) |

最后只需分别定义训练和验证的方法。训练过程中，每次迭代结束后都输出损失函数的值loss和梯度更新次数step，最后验证环境输出准确率acc。

# 2.3 实验环境与平台

本次实验在华为云实验平台完成。在华为云ModelArts平台上创建AI框架为Mindspore-1.5、硬件环境为Ascend 910+ARM的开发环境，并创建好搭建了MindSpore环境的Jupyter Notebook，在线上填入代码运行并测试。

# 2.4 结果与分析

运行程序，总共执行1875次梯度更新，每次迭代结束时都对测试集进行验证并输出测试集上的损失函数值，如图2-1所示。最终的准确率为0.96915，损失函数值为0.0841，达到了预期目标，训练效果较好。

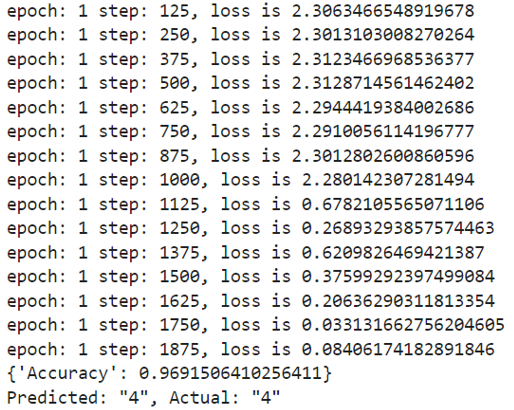


图2-1：MNIST手写数字识别测试结果

# 2.5 个人体会

在这次实验中，我对60000个训练集数据进行了学习，并在10000个测试集数据上进行了测试。我建立了LeNet5神经网络，并通过1875次梯度更新来优化模型。

通过本次实验，我深刻认识到了数据处理的重要性。对于模型训练而言，数据的质量和数量是决定模型效果的关键因素之一。同时，我也学会了如何利用深度学习框架TensorFlow来建立神经网络模型，并掌握了梯度下降算法的基本原理和使用方法。总之，我不仅学到了理论知识，还锻炼了实践操作能力，对未来的学习有很大帮助。

# 3 机器学习大作业：Kaggle 2017年房价预测（House Prices - Advanced Regression Techniques）

# 3.1 实验要求

大作业是Kaggle平台上的一个竞赛题目，主要是根据给定的房价数据集（包含影响房价的各种因素以及最终的房价），预测具有同样属性集的测试集的房价（链接：<https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/code?competitionId=5407>）。

具体来说，题目给定了79个房价的影响因素，要求对测试集中的每个id预测其房价。最终将预测结果提交到Kaggle上评测，评分采用均方误差RMSE表示，这个值越低越好（代表预测结果越准确）。

# 3.2 设计与实现

大作业的整体流程是：对数据进行清洗和预处理；将类别属性数值化并利用随机森林（Ramdom Forest）实现特征选择；进行数据的相关性分析、方差分析与标准化；通过主成分分析PCA进行降维；利用Kmeans算法实现聚类（二分类）；训练、线性回归并得到预测结果。

# 3.2.1 数据集特点分析

首先分析给定的数据集。数据集以csv的格式呈现，文件名为“test.csv”，其中包含了全部79个影响房价的属性以及每个id对应的房价，如图3-1所示。

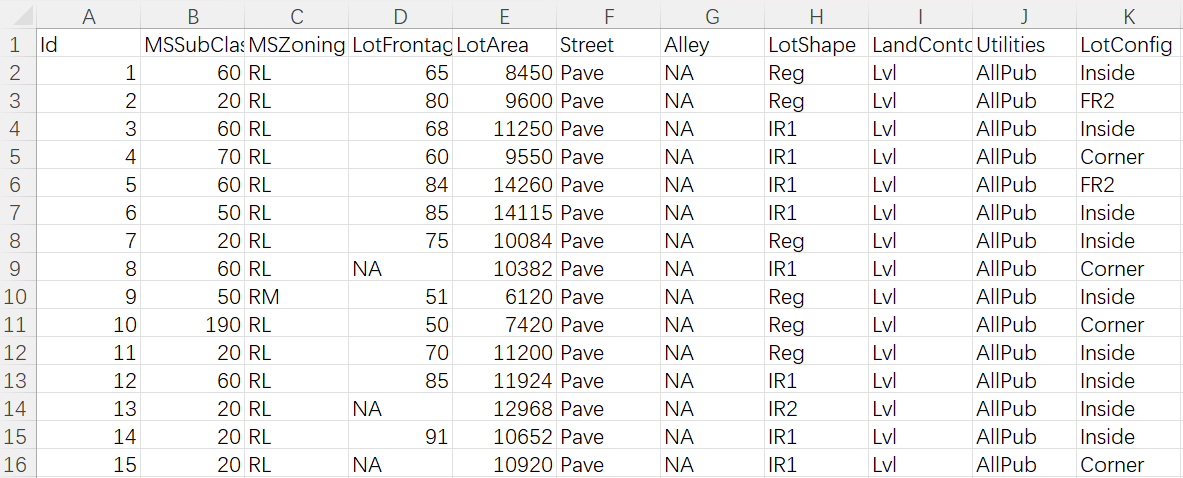


图3-1：房价数据集预览

数据集共1461行，除第一行为属性名称外，每一行表示一个房屋售价样本，包含79个属性和1个房价值。其中这79个属性的说明如表3-1所示（其中数值属性的类别标记为1，类别属性标记为2）。

表3-1：影响房价的79个属性说明

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | 属性名 | 类别 | 含义 | # | 属性名 | 类别 | 含义 |
| 1 | MSSubClass | 1 | 建筑类别 | **41** | TotalBsmtSF | 1 | 地下室总面积 |
| 2 | MSZoning | 2 | 分区分类 | **42** | Electrical | 2 | 电气系统 |
| 3 | LotFrontage | 1 | 街道长度 | **43** | 1stFlrSF | 1 | 一楼面积 |
| 4 | LotArea | 1 | 地块面积 | **44** | 2ndFlrSF | 1 | 二楼面积 |
| 5 | Street | 2 | 道路通行方式 | **45** | LowQualFinSF | 1 | 低质量完成面积 |
| 6 | Alley | 2 | 巷道通行方式 | **46** | GrLivArea | 1 | 地上起居面积 |
| 7 | LotShape | 2 | 财产类型 | **47** | BsmtFullBath | 1 | 地下室全浴室 |
| 8 | LandContour | 2 | 财产平整度 | **48** | BsmtHalfBath | 1 | 地下室半浴室 |
| 9 | Utilities | 2 | 公共设施 | **49** | FullBath | 1 | 地上全浴室 |
| 10 | LotConfig | 2 | 地块配置 | **50** | HalfBath | 1 | 地上半浴室 |
| 11 | LandSlope | 2 | 倾斜程度 | **51** | BedroomAbvGr | 1 | 卧室数量 |
| 12 | Neighborhood | 2 | 地理位置 | **52** | KitchenAbvGr | 1 | 厨房数量 |
| 13 | Condition1 | 2 | 靠近铁路 | **53** | KitchenQual | 2 | 厨房质量 |
| 14 | Condition2 | 2 | 靠近铁路 | **54** | TotRmsAbvGrd | 1 | 总房间数 |
| 15 | BldgType | 2 | 住宅类型 | **55** | Functional | 2 | 功能评级 |
| 16 | HouseStyle | 2 | 住宅风格 | **56** | Fireplaces | 1 | 壁炉数量 |
| 17 | OverallQual | 1 | 总体装修质量 | **57** | FireplaceQu | 2 | 壁炉质量 |
| 18 | OverallCond | 1 | 总体评级 | **58** | GarageType | 2 | 车库位置 |
| 19 | YearBuilt | 1 | 建造年份 | **59** | GarageYrBlt | 1 | 车库建造年份 |
| 20 | YearRemodAdd | 1 | 改造年份 | **60** | GarageFinish | 2 | 车库装修 |
| 21 | RoofStyle | 2 | 屋顶类型 | **61** | GarageCars | 1 | 车位大小 |
| 22 | RoofMatl | 2 | 屋顶材料 | **62** | GarageArea | 1 | 车库面积 |
| 23 | Exterior1st | 2 | 外部材料 | **63** | GarageQual | 2 | 车库质量 |
| 24 | Exterior2nd | 2 | 外部材料 | **64** | GarageCond | 2 | 车库条件 |
| 25 | MasVnrType | 2 | 饰面类型 | **65** | PavedDrive | 2 | 铺设车道 |
| 26 | MasVnrArea | 1 | 饰面面积 | **66** | WoodDeckSF | 1 | 木板面积 |
| 27 | ExterQual | 2 | 外材质量 | **67** | OpenPorchSF | 1 | 露台面积 |
| 28 | ExterCond | 2 | 外材条件 | **68** | EnclosedPorch | 1 | 封闭露台面积 |
| 29 | Foundation | 2 | 地基类型 | **69** | 3SsnPorch | 1 | 三季阳光房面积 |
| 30 | BsmtQual | 2 | 地下室高 | **70** | ScreenPorch | 1 | 带纱窗的阳光房面积 |
| 31 | BsmtCond | 2 | 地下状况 | **71** | PoolArea | 1 | 泳池面积 |
| 32 | BsmtExposure | 2 | 地下室墙体类别 | **72** | PoolQC | 2 | 泳池质量 |
| 33 | BsmtFinType1 | 2 | 地下室装修质量 | **73** | Fence | 2 | 围栏质量 |
| 34 | BsmtFinSF1 | 1 | 完成面积 | **74** | MiscFeature | 2 | 杂项功能 |
| 35 | BsmtFinType2 | 2 | 地下室装修质量 | **75** | MiscVal | 1 | 杂项价值 |
| 36 | BsmtFinSF2 | 1 | 完成面积 | **76** | MoSold | 1 | 出售月份 |
| 37 | BsmtUnfSF | 1 | 未完成面积 | **77** | YrSold | 1 | 出售年份 |
| 38 | Heating | 2 | 供暖方式 | **78** | SaleType | 2 | 销售类型 |
| 39 | HeatingQC | 2 | 供暖质量 | **79** | SaleCondition | 2 | 销售条件 |
| 40 | CentralAir | 2 | 中央空调 | **-** | SalePrice | 1 | 最终售价 |

观察数据集可知以下三点：

1. 数据集中影响房价的因素太多，需要找到影响房价的主要属性。
2. 属性可以被分为两部分：数值属性（用数字进行量化，共37种）和类别属性（用类别表示，共42种）。对类别属性，考虑使用某种方法将其转变为数值属性，方便处理。例如，上表中属性CentralAir只存在两种取值Y或N，据此可以将其替换为1或0。
3. 很多属性的列中存在大量空值NAN，不利于后续训练，需要进行数据的清洗和预处理。

# 3.2.2 数据清洗与预处理

首先将所有形如""、"--"、"?"、"na"、"NAN"、"nan", ''的值都视为空值（NAN），并将其替换为空。这样就完成了对空值的集中处理。

此时还没有将类别属性数值化，故先分析数值属性。绘制训练集全部数值属性相关性示意图，如图3-2所示。

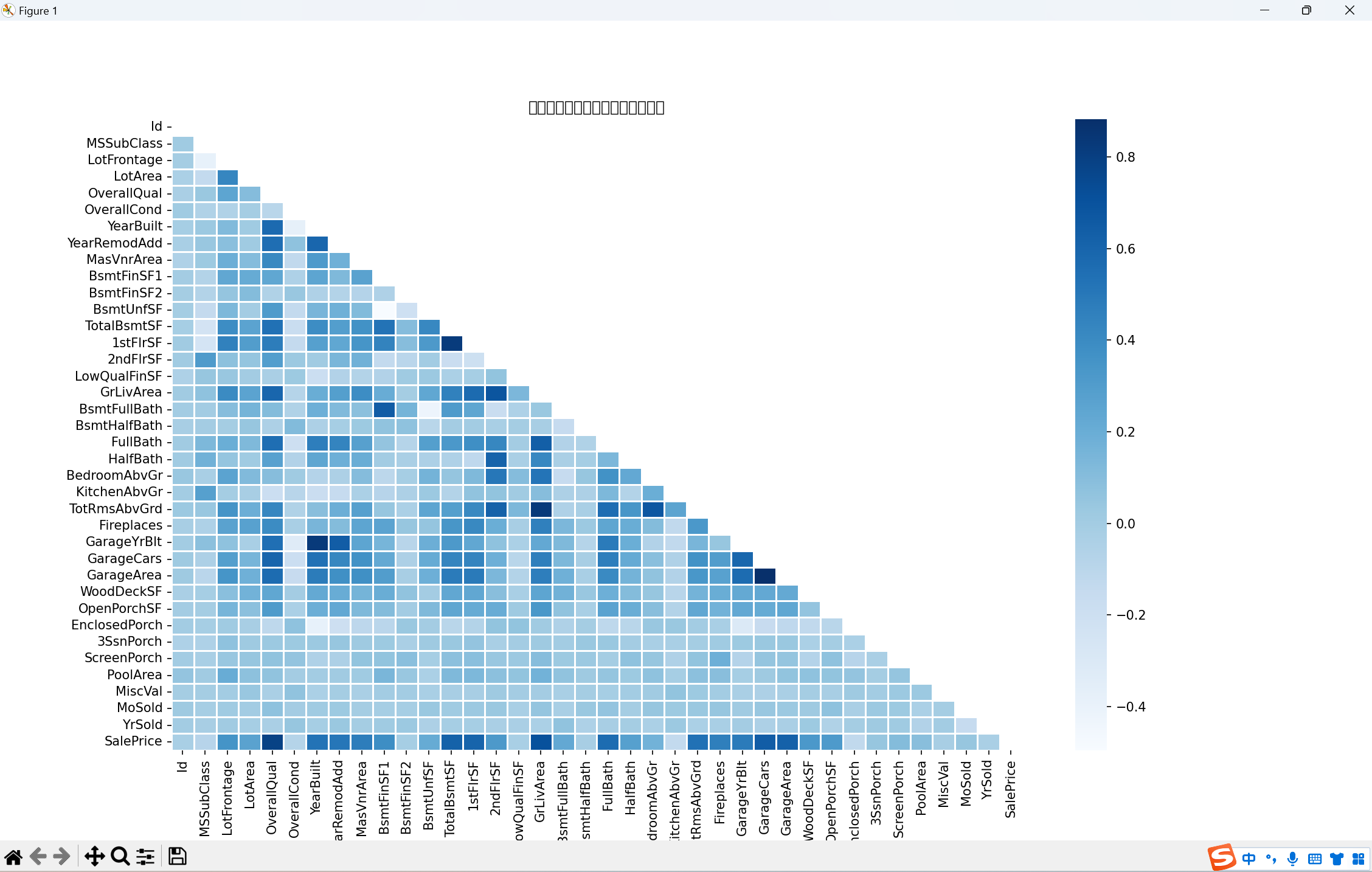


图3-2：训练集数值属性相关性示意图

上图中，颜色越深代表两种属性间的相关性越强。可见不同的属性对对房价的影响程度不同。

接下来，再进一步绘制每个属性的空值密集程度图，如图3-3所示。若某个属性包含很多空值，则表现为大量空白，反之则不会出现大量空白。

可见，大部分属性的空值出现次数较少，而少部分属性则存在极大空缺，这部分属性需要被移除掉，以避免出现数据偏差和模型性能下降。在实际操作中，我统计了每个属性的空缺值，若超过600个值空缺则将其视为无关变量，在后续学习中不再考虑这些因素的影响。这些属性是：Alley、FireplaceQu、PoolQC、Fence、MiscFeature。

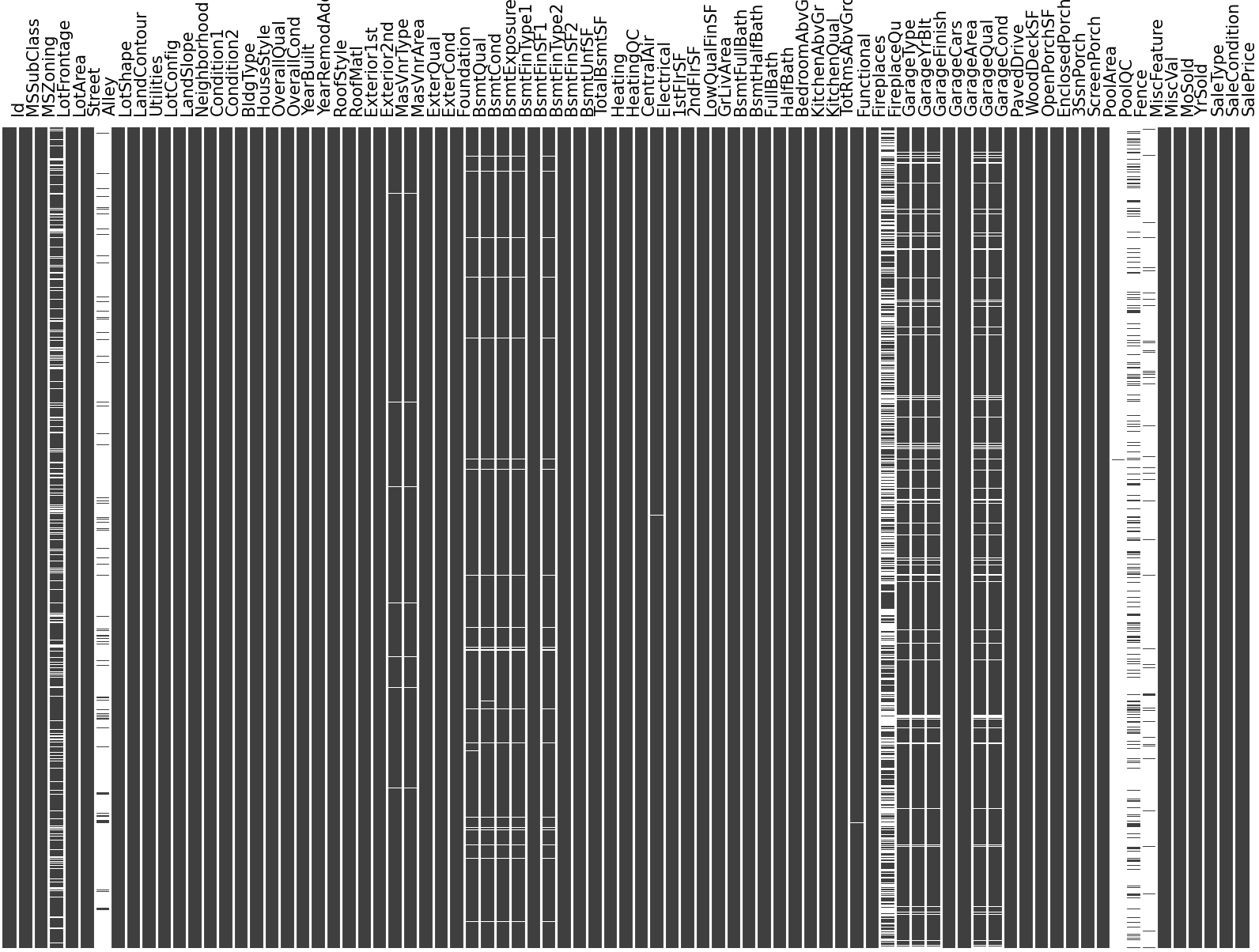


图3-3：训练集空值密集程度图

下面对剩余列的缺失值进行填充，其中数值属性用这一列的平均值填充，类别属性用这一列出现次数最多的属性填充，代码如下。

|  |
| --- |
| **for** attributes **in** full\_dataset:  **if** attributes **in** numerical\_attr:  full\_dataset[attributes] = full\_dataset[attributes].fillna(value=full\_dataset[attributes].median())  **else**:  full\_dataset[attributes] = full\_dataset[attributes].fillna(value=full\_dataset[attributes].value\_counts().idxmax()) |

# 3.2.3 将类别属性数值化并利用随机森林实现特征选择

上面已经得到了类别属性的总数为42。为了从这么多的属性中获得相对重要的类别属性及其对应的取值，这里需要引入sklearn库中的随机森林分类器RandomForestClassifier、特征选择器SelectFromModel。同时为了避免过拟合，需要引入train\_test\_spilt函数对训练集和测试集进行分割。

首先，从train\_data中取得类别属性，使用pd.get\_dummies对每个类别属性进行独热编码，转化成二进制数值。这样可以将类别属性转化为编码。通过循环遍历每个属性，将所有的类别属性都转换成了数值属性。接着使用train\_test\_split函数将数据集分割成训练集和测试集，比例为3:1。核心代码如下：

|  |
| --- |
| **for** at **in** X:  X = pd.concat([X, pd.get\_dummies(X[at], prefix=at, drop\_first=**True**)], axis=1, sort=**False**)X.drop(columns=[at], inplace=**True**)  X\_train\_st, X\_test\_st, y\_train\_st, y\_test\_st = train\_test\_split(X, Y, test\_size=0.25) |

接下来通过随机森林分类器和特征选择器对类别属性进行筛选，保留最重要的类别属性。创建150棵树进行随机森林处理，并利用RFC作为估计器，选取重要度大于全部类别属性取值重要度平均值的类别属性取值。

|  |
| --- |
| rfc = RandomForestClassifier(n\_estimators=150)sel = SelectFromModel(estimator=rfc, threshold=**'mean'**)  sel.fit(X, Y)  print(**"门限threshold: "**, sel.estimator\_.feature\_importances\_.mean()) print(sel.get\_support())  selected\_features = X\_train\_st.columns[(sel.get\_support())]print(**"高于门限值的类别属性取值个数："**, len(selected\_features), **"\n这些属性的值是："**) print(selected\_features) |

程序输出了门限值（即重要度的平均值）以及高于门限值的全部类别属性取值、全部类别属性取值的重要度，如图3-4、图3-5所示。可见，有73个类别属性的具体取值（注意不是73个属性）高于门限值。

此后，为了进一步找出重要的类别属性，还要将高于门限值的类别属性取值的重要度从高到低排序并输出，同时绘制重要度排名前十的类别属性取值直方图，如图3-6、图3-7所示。



图3-4：高于门限值的全部类别属性取值

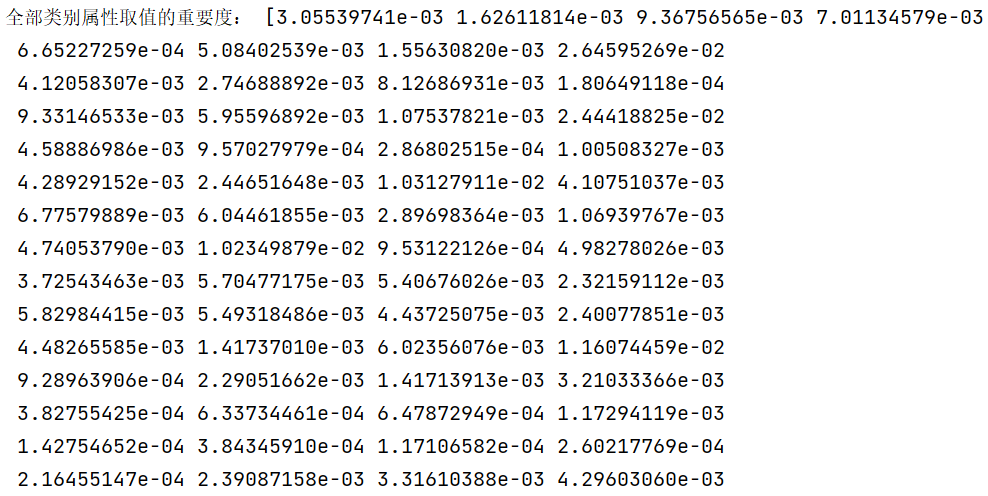


图3-5：全部类别属性取值的重要度（节选）



图3-6：类别属性取值的重要度（从高到低排序）

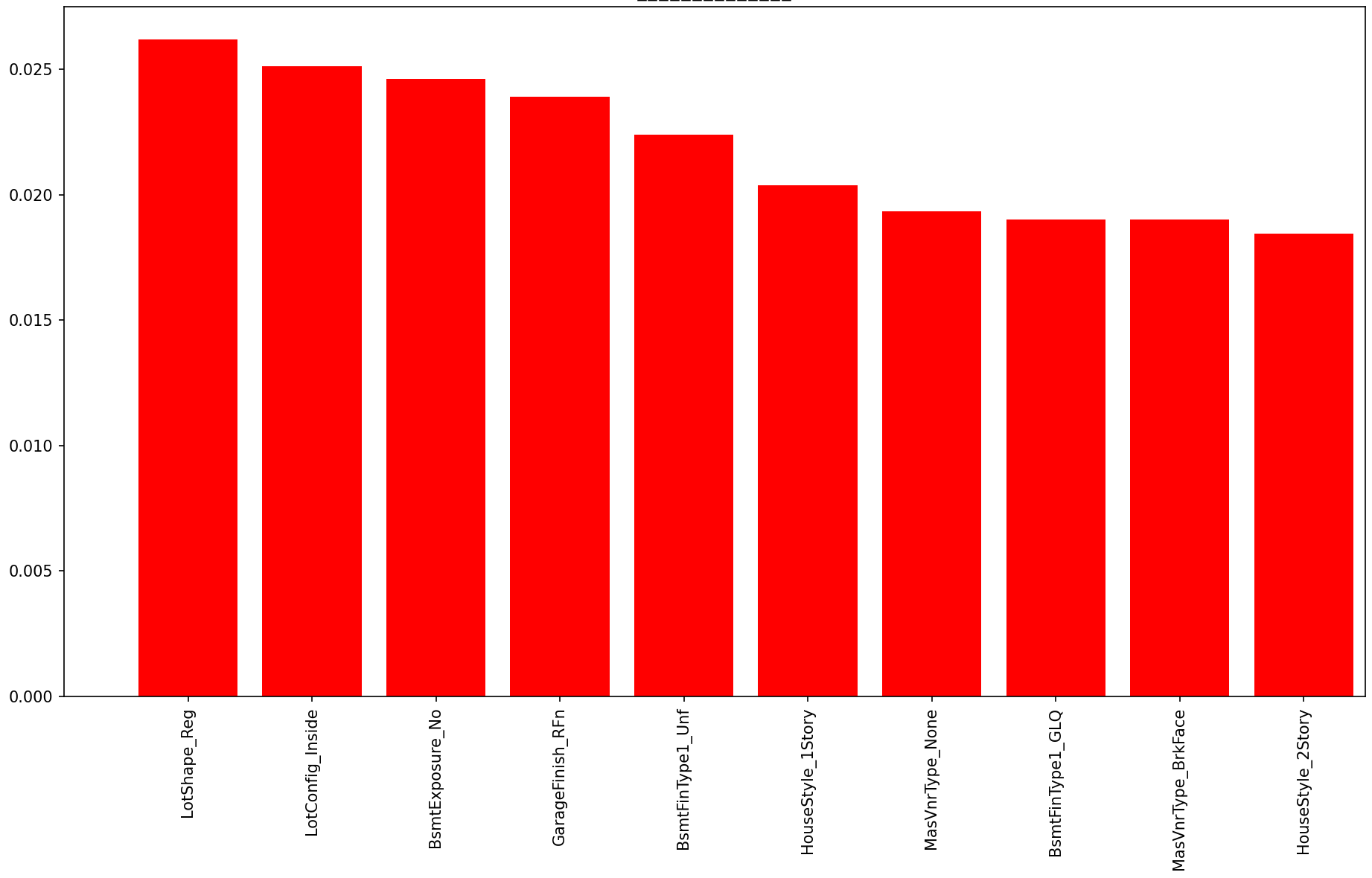


图3-7：重要度排名前十的类别属性取值直方图

对这些值所属的类别属性进行数值化，分别是：GarageFinish、LotShape、BsmtExposure、BsmtFinType1、LotConfig、HouseStyle、MasVnrType。其中以GarageFinish属性为例，数值化后的效果如图3-8所示。即在该类别属性上，两种取值RFn和Unf分别变为新的两列，并做0-1化处理。

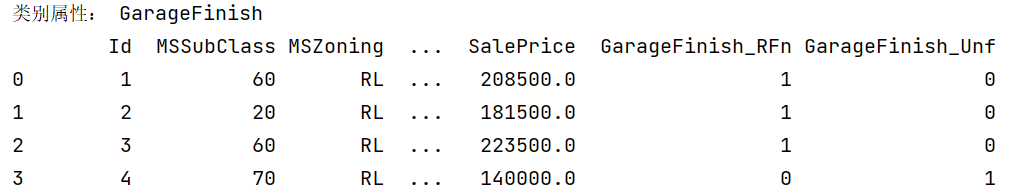


图3-8：类别属性数值化（以GarageFinish属性为例）

至此，已经完成了重要的特征选择工作，并可以画出完整的训练集属性相关性示意图，如图3-9所示。其含义已在前一部分解释过，此处不再赘述。

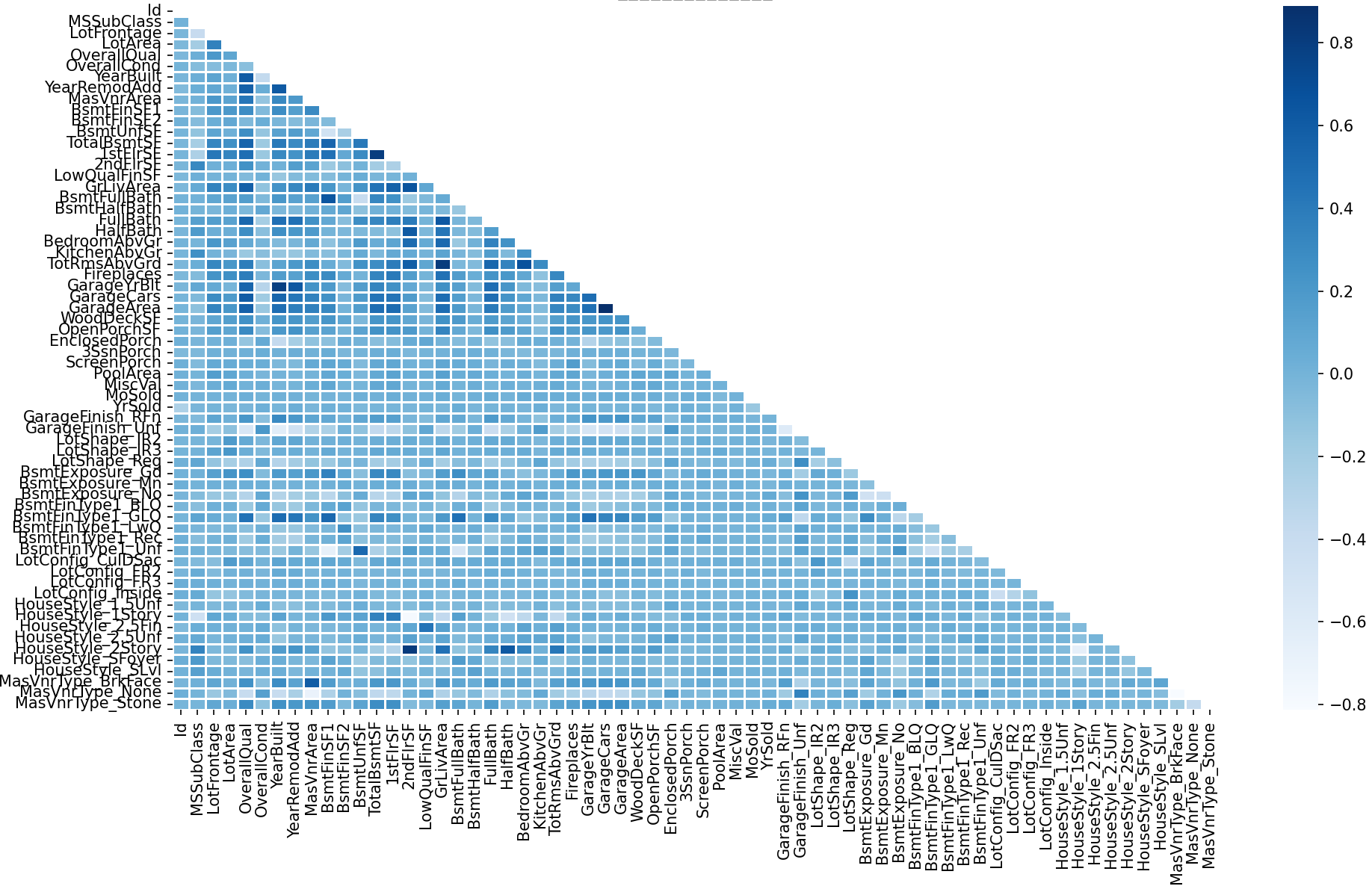


图3-9：完整的训练集属性相关性示意图

# 3.2.4 数据的相关性分析、方差分析与标准化

经过上一步操作，已经得到了需要处理的全部属性的数值形式。

但关键问题在于：各个属性之间的值域差别太大，有的取值范围是在0和1之间二选一，有的是在1~10之间，还有的是在0~2000之间。巨大的尺度差异会在后续聚类过程中放大某些因素对聚类的影响，同时会在很大程度上忽略某些因素。为此，需要对数据进行标准化，使各个维度的数据在决定各个数据点的特征上具有同等地位。因此下面首先对数据进行相关性分析和方差分析。

**（1）相关性分析**

计算每种属性与房价Saleprice的相关系数，并创建一个名为CORR的列保存，如图3-10所示。可以看到一些属性的相关系数为正，代表其与房价成正相关；反之，若相关系数为负，代表其与房价成负相关。相关系数的绝对值越大，代表其对房价的影响越大。



图3-10：全部属性对房价的相关系数

**（2）方差分析和标准化**

遍历每一个属性，计算其平均值、方差与标准化方差（标准差）并输出结果，如图3-11所示。然后导入scipy.stats库中的zscore函数来对数据进行标准化。使用zscore函数进行标准化后得到的是一个标准正态分布，即均值为0、标准差为1的分布，有利于后续聚类操作。Zscore的计算公式如下。

其中，是原始数据，是原始数据的平均值，是原始数据的标准差。



图3-11：方差分析

实现数据标准化的代码是：

|  |
| --- |
| **from** scipy.stats **import** zscore old\_full\_dataset = full\_dataset full\_dataset[numerical\_attr] = zscore(full\_dataset[numerical\_attr]) |

# 3.2.5 通过主成分分析PCA进行降维

上面得到的属性种数仍然过多，因此考虑使用主成分分析（PCA）对数据进行降维，其原理解释如下。

在PCA中，可解释方差贡献率是指每个主成分能够解释原始数据的方差占比。它表示每个主成分对于原始数据的重要程度，即在保持数据尽可能多的信息的同时，减少数据的维数。通常情况下会优先选择保留解释方差贡献率较高的主成分，以尽可能地保留原始数据的信息。

根据这一原理，首先导入sklearn.decomposition中的PCA模块并fit该模型，然后计算各个属性的可解释方差贡献率（表示每个属性解释数据的方差占比）并输出，如图3-12所示。

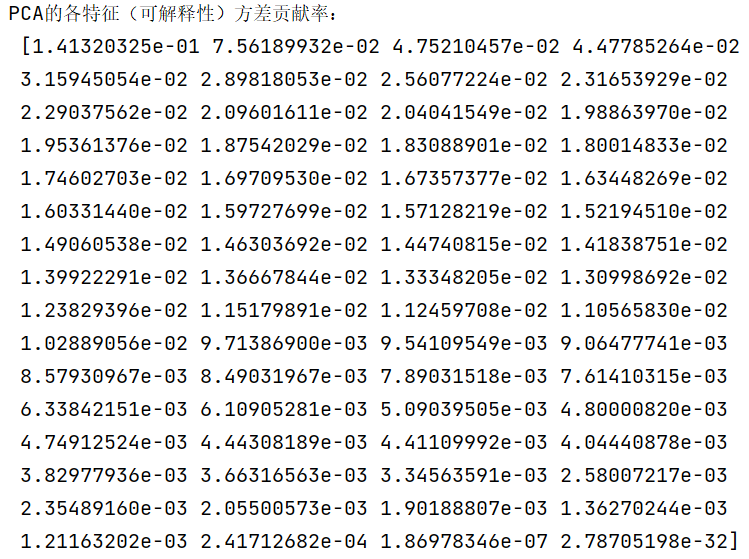


图3-12：各个主成分的可解释方差贡献率

使用np.cumsum函数计算累积方差贡献率，并将其绘制成曲线图，如图3-13所示。在这里，将保留特征数量设为20%，即当累计方差占比达到0.72时截止。由这个图可以看出，红线以下的属性个数为24，它们就是要保留的属性。

接下来，删除ID列等不必要的列，并选择需要进行PCA处理的数值属性。设定要保留的主成分数量为24，即将数据降至24维。最后将主成分分析结果转换为DataFrame对象即可。

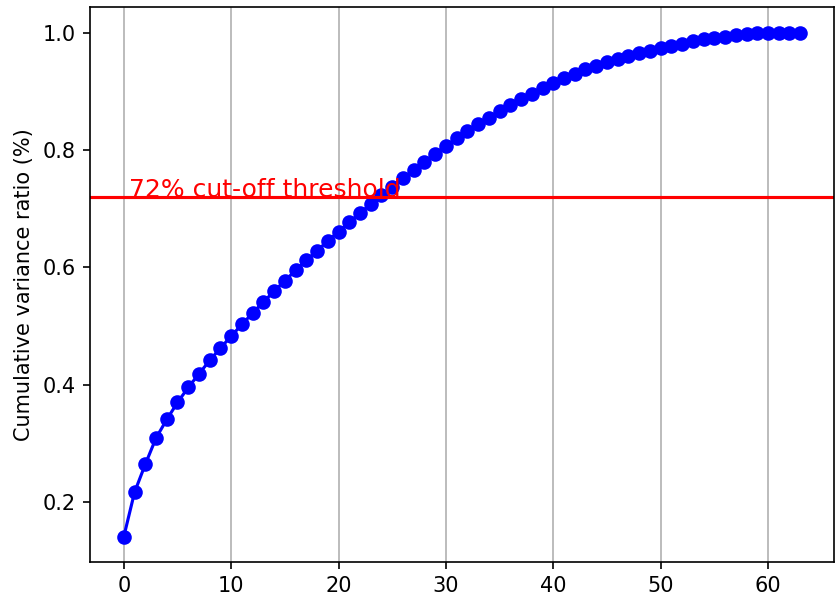


图3-13：累计方差贡献率示意图

主成分分析中每个主成分解释数据的方差百分比如图3-14所示。这24个主成分的重要性依次递减，但总体上最大化地保留了原始数据集的特征。

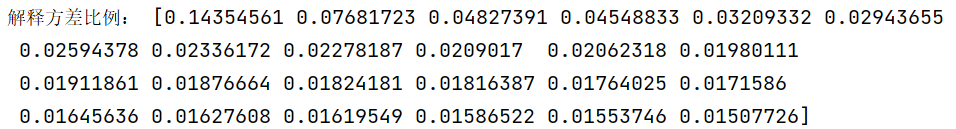


图3-14：每个主成分解释数据的方差百分比

# 3.2.6 利用Kmeans算法实现聚类（二分类）

通过PCA，我将属性降维到了24维，下面将训练集中的1460条数据进行聚类。聚类后，认为每个聚类（簇）中的数据具有高相似性，并在下一步对每个聚类单独进行训练和线性回归，由此得到测试集的预测结果。

首先通过聚类算法的Elbow Method确定最优聚类数量。Elbow Method是一种常用的聚类分析方法，其基本思想是通过求解不同的聚类数对应的簇内平方和（SSE）或其它衡量指标，找到一个“拐点”。在这个拐点之前，随着聚类数目的增加，模型的性能提升较为明显，但在这个点之后，再增加聚类数并不能显著地提高模型的性能表现，因此这个拐点通常被认为是最优聚类数目的估计值。该方法可以帮助用户选择合适的聚类数目，并可以避免聚类数量过多或者过少的问题。

在本实验中，我使用Elbow Method时引入了两种评价指标：每个簇的质点与簇内样本点的距离误差平方和（distortions）和样本距离最近的聚类中心的距离误差总和（inertias）。实际上，前者表现出的效果更好，因为它对离聚类中心点越远的点的惩罚越大。具体实现时采用欧式距离计算即可，逐个计算聚类数量在1~15时的距离误差平方和。核心代码如下：

|  |
| --- |
| inertias = []distortions = []K = range(1, 15) **for** k **in** K:  kmeanModel = KMeans(n\_clusters=k).fit(X)  kmeanModel.fit(X)  distortions.append(sum(np.min(cdist(X, kmeanModel.cluster\_centers\_, **'euclidean'**), axis=1)) / X.shape[0])  inertias.append(kmeanModel.inertia\_) |

结果如图3-15所示。可见，k=2为明显的拐点，因此本问题中最优聚类数量为2。

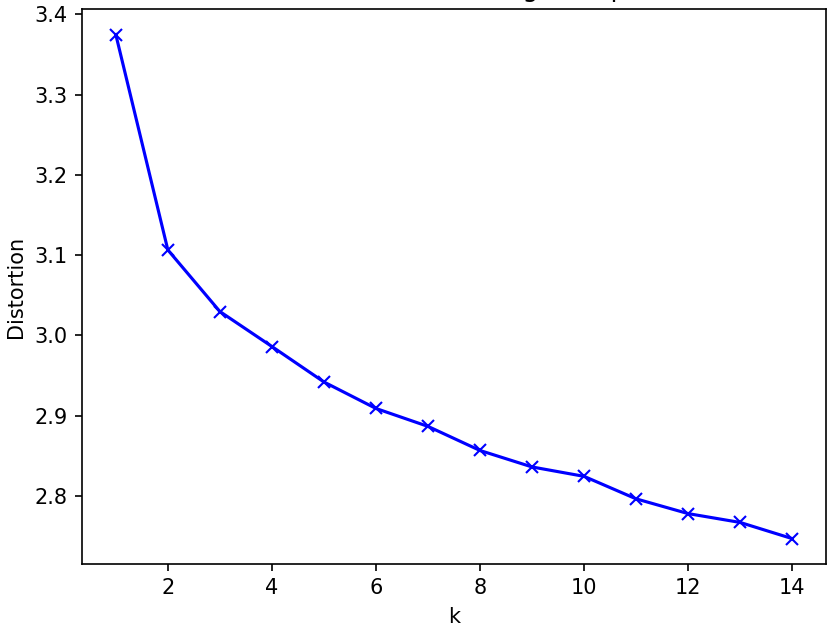


图3-15：以Distortion为评价指标，利用Elbow Method确定最优聚类数量

据此即可对数据集应用Kmeans聚类算法进行二分类（n\_clusters=2），将数据集分为两个互不重叠的簇。然后用predict方法获取每个数据点所属的簇。最后，将全部数据分别保存到cluster\_0/cluster\_1两个DataFrame中，并输出各自的数量，如表3-2所示。

表3-2：两个聚类的元素数量

|  |  |
| --- | --- |
| 聚类 | 元素数量 |
| 1 | 1528 |
| 2 | 1391 |

两个聚类中的元素数量近似相同，说明聚类具有均匀性。

# 3.2.7 训练、线性回归并得到预测结果

从sklearn.linear\_model库中引入线性回归函数LinearRegression，并对两个聚类分别进行训练和线性回归预测。

对于每个聚类，根据Saleprice是否为空来确定该行数据进入训练集还是测试集。将训练集和测试集分别存储在train\_df和test\_df中，并提取出Id列用于最终结果的合并。接着，分别从train\_df和test\_df中删除Cluster列（因为已经完成聚类，不再需要分类标签）、Id列、Saleprice列，并将剩余特征值作为X\_train和X\_test，Saleprice列作为Y\_train和Y\_test。对X\_train和Y\_train进行线性回归拟合得到预测值，并将其与对应的Id存储起来。最后，将聚类结果和预测结果合并在一起，形成Saleprice的DataFrame并按照Id排序，将其保存为csv文件并输出为saleprice.csv。

至此，已经得到了预测结果，见“结果与分析”章节。

综上所述，大作业的算法流程图如图3-16所示。

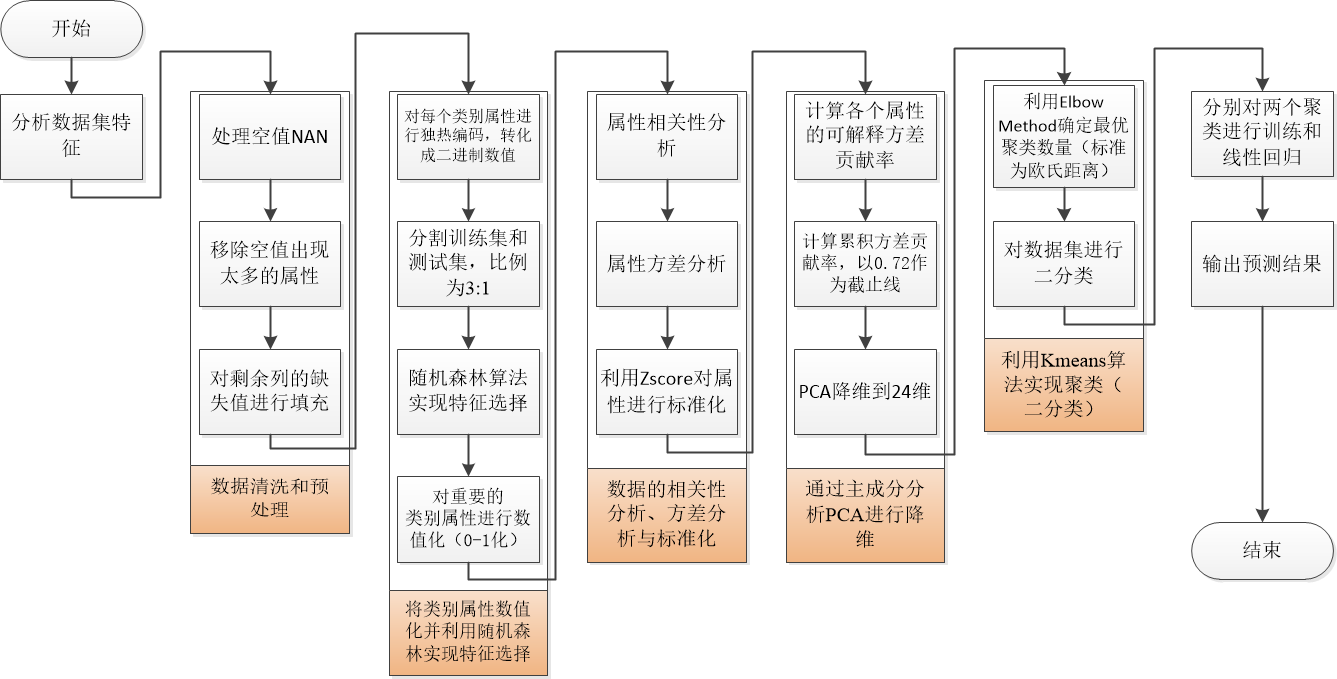


图3-16：大作业算法流程图

# 3.3 实验环境与平台

实验平台是Windows 11操作系统下的PyCharm Community 2023.1（64 bit）。

# 3.4 结果与分析

运行ML\_HousingRegression.py，程序生成预测结果saleprice.csv，如图3-17所示。

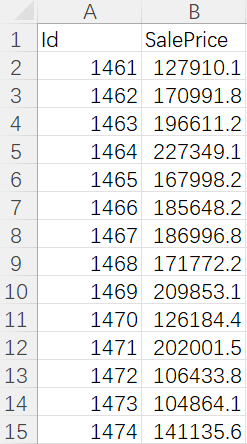
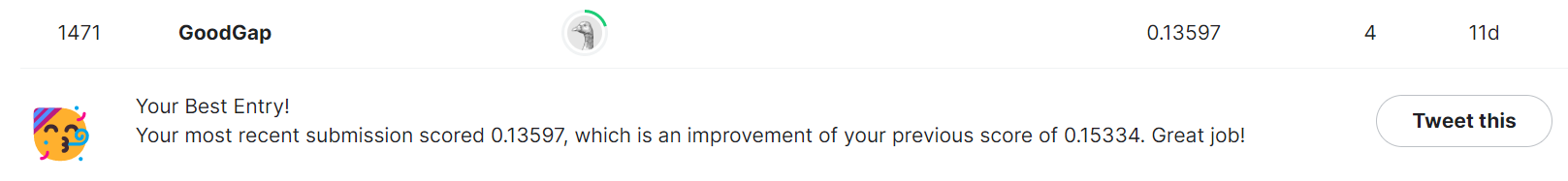


图3-17：房价预测结果（节选）

将结果上传到Kaggle平台评测，得分为0.13597，预测效果好。



# 3.5 个人体会

这是本学期机器学习课程的大作业，我主要是实现了对房价的预测。整个流程并非一帆风顺，而是遇到了一些问题，不过经过个人努力都得以解决。例如一开始没有将训练集和测试集进行分割，导致出现了过拟合，后来找到问题并将其以3:1的比例分割就能解决。还有，一开始在PCA降维时，不知道降到多少维才能较好地保留数据集原始特征。经过不断调参，最终发现截止线在0.72左右时效果较好，太高后降维效果比较差，太低了则会丢失很多重要信息。

这个过程中，根据最后的预测效果，我还进行了一些灵活的优化，例如一开始的预测效果不太好，且训练时间较长，后来我发现了有些属性对结果影响很大，有些却影响不大。根据这一发现，启发我引入了随机森林和主成分分析PCA筛选出重要的属性并进行降维。同时我还用了机器学习领域中的多种方法，例如Kmeans聚类、线性回归等经典算法，进一步提高了我综合运用多种思路解决问题的能力，以及具体问题具体分析的思维。

总的来说，这学期我学到了很多关于机器学习的知识和方法。此前我从未真正上首实践过机器学习的算法，但我发现这是一门非常有趣和实用的学科，它可以帮助我更好地理解数据，并从数据中发现有用的规律和趋势。同时，我也深刻认识到了机器学习在实际应用中的问题和挑战，例如如何选择合适的算法、如何避免过拟合等等。总之，通过这学期的学习，我更加深入地了解了机器学习的本质和应用，收获颇丰。

**附录一 实验六代码**

**（一）鸢尾花三分类问题 yuanweihua\_experiment.py**

**import** os  
*# os.environ['DEVICE\_ID'] = '0'***import** csv  
**import** numpy **as** np  
**from** pprint **import** pprint  
**import** random  
  
**import** mindspore **as** ms  
**from** mindspore **import** nn  
*# from mindspore import context***from** mindspore **import** dataset  
**from** mindspore.train.callback **import** LossMonitor  
**from** sklearn.datasets **import** load\_iris  
  
**def** create\_dataset(data\_path):  
 *#* ***Todo 每个类的前五个样本信息*** x\_data = load\_iris().data  
 y\_data = load\_iris().target.reshape(150, 1)  
 data = np.hstack((x\_data, y\_data))  
  
 *#* ***Todo 分别将Iris-setosa，Iris-versicolor，Iris-virginica对应为0，1，2三类*** label\_dict = {0: 0, 1: 1, 2: 2}  
 X = np.array([[float(x) **for** x **in** s[:-1]] **for** s **in** data[:150]], np.float32)  
 Y = np.array([label\_dict[s[-1]] **for** s **in** data[:150]], np.int32)  
  
 *#* ***Todo Using random choice and split dataset into train set and validation set by 8:2.*** train\_idx = np.random.randint(0, len(Y), int(len(Y) \* 0.8))  
 test\_idx = np.array([i **for** i **in** range(0, len(Y)) **if** i **not in** train\_idx])  
 X\_train, Y\_train = X[train\_idx], Y[train\_idx]  
 X\_test, Y\_test = X[test\_idx], Y[test\_idx]  
  
 *# Convert the training data to MindSpore Dataset.* XY\_train = list(zip(X\_train, Y\_train))  
 ds\_train = dataset.GeneratorDataset(XY\_train, [**'x'**, **'y'**])  
 ds\_train = ds\_train.shuffle(buffer\_size=120).batch(32, drop\_remainder=**True**)  
  
 *# Convert the test data to MindSpore Dataset.* XY\_test = list(zip(X\_test, Y\_test))  
 ds\_test = dataset.GeneratorDataset(XY\_test, [**'x'**, **'y'**])  
 ds\_test = ds\_test.shuffle(buffer\_size=120).batch(32, drop\_remainder=**True**)  
 **return** ds\_train, ds\_test  
  
**def** softmax\_regression(ds\_train, ds\_test):  
 net = nn.Dense(4, 3)  
  
 *#* ***Todo 使用交叉熵损失计算*** loss = nn.SoftmaxCrossEntropyWithLogits(sparse=**True**, reduction=**'mean'**)  
 *#* ***Todo 使用动量优化器优化参数，其中学习率设置为0.05，动量设置为0.9*** opt = nn.optim.Momentum(net.trainable\_params(), learning\_rate=0.05, momentum=0.9)  
  
 model = ms.train.Model(net, loss, opt, metrics={**'acc'**, **'loss'**})  
 model.train(25, ds\_train, callbacks=[LossMonitor(per\_print\_times=ds\_train.get\_dataset\_size())], dataset\_sink\_mode=**False**)  
 metrics = model.eval(ds\_test)  
 print(metrics)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 **import** argparse  
 parser = argparse.ArgumentParser()  
 parser.add\_argument(**'--data\_url'**, default=**'iris.data'**, help=**'Location of data.'**)  
 args, unknown = parser.parse\_known\_args()  
  
 **if** args.data\_url.startswith(**'s3'**):  
 *#* ***Todo:设置路径*** data\_path = **""  
 import** moxing  
 moxing.file.copy\_parallel(src\_url=os.path.join(args.data\_url, **'iris.data'**), dst\_url=data\_path)  
 **else**:  
 data\_path = os.path.abspath(args.data\_url)  
  
 softmax\_regression(\*create\_dataset(data\_path))

**（二）MNIST手写数字识别 mnist\_experiment.py**

**import** os  
**import** argparse  
**from** mindspore **import** context  
**from** mindspore.train.callback **import** Callback, LossMonitor  
  
parser = argparse.ArgumentParser(description=**'MindSpore LeNet Example'**)  
parser.add\_argument(**'--device\_target'**, type=str, default=**"CPU"**, choices=[**'Ascend'**, **'GPU'**, **'CPU'**])  
  
args = parser.parse\_known\_args()[0]  
*# context.set\_context(mode=context.GRAPH\_MODE, device\_target=args.device\_target)***import** os  
**import** requests  
  
requests.packages.urllib3.disable\_warnings()  
  
**def** download\_dataset(dataset\_url, path):  
 filename = dataset\_url.split(**"/"**)[-1]  
 save\_path = os.path.join(path, filename)  
 **if** os.path.exists(save\_path):  
 **return  
 if not** os.path.exists(path):  
 os.makedirs(path)  
 res = requests.get(dataset\_url, stream=**True**, verify=**False**)  
 **with** open(save\_path, **"wb"**) **as** f:  
 **for** chunk **in** res.iter\_content(chunk\_size=512):  
 **if** chunk:  
 f.write(chunk)  
 print(**"The {} file is downloaded and saved in the path {} after processing"**.format(os.path.basename(dataset\_url), path))  
  
train\_path = **"datasets/MNIST\_Data/train"**test\_path = **"datasets/MNIST\_Data/test"**download\_dataset(**"https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/train-labels-idx1-ubyte"**, train\_path)  
download\_dataset(**"https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/train-images-idx3-ubyte"**, train\_path)  
download\_dataset(**"https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/t10k-labels-idx1-ubyte"**, test\_path)  
download\_dataset(**"https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/t10k-images-idx3-ubyte"**, test\_path)  
  
**import** mindspore.dataset **as** ds  
**import** mindspore.dataset.transforms.c\_transforms **as** C  
**import** mindspore.dataset.vision.c\_transforms **as** CV  
**from** mindspore.dataset.vision **import** Inter  
**from** mindspore **import** dtype **as** mstype  
  
**def** create\_dataset(data\_path, batch\_size=32, repeat\_size=1,num\_parallel\_workers=1):  
 *# 定义数据集* mnist\_ds = ds.MnistDataset(data\_path)  
 *#* ***Todo 设置放缩的大小*** resize\_height = 32  
 resize\_width = 32  
 *#* ***Todo 归一化*** rescale = 1/255.  
 shift = 0.0  
 rescale\_nml = 1 / 0.3081  
 shift\_nml = -1 \* 0.1307 / 0.3081  
  
 *# 定义所需要操作的map映射* resize\_op = CV.Resize((resize\_height, resize\_width), interpolation=Inter.LINEAR)  
 rescale\_nml\_op = CV.Rescale(rescale\_nml, shift\_nml)  
 rescale\_op = CV.Rescale(rescale, shift)  
 hwc2chw\_op = CV.HWC2CHW()  
 type\_cast\_op = C.TypeCast(mstype.int32)  
  
 *# 使用map映射函数，将数据操作应用到数据集* mnist\_ds = mnist\_ds.map(operations=type\_cast\_op, input\_columns=**"label"**, num\_parallel\_workers=num\_parallel\_workers)  
 mnist\_ds = mnist\_ds.map(operations=[resize\_op, rescale\_op, rescale\_nml\_op, hwc2chw\_op], input\_columns=**"image"**, num\_parallel\_workers=num\_parallel\_workers)  
  
 *#* ***Todo 进行shuffle、batch、repeat操作*** buffer\_size = 10000  
 mnist\_ds = mnist\_ds.shuffle(buffer\_size=buffer\_size)  
 mnist\_ds = mnist\_ds.batch(batch\_size, drop\_remainder=**True**)  
 mnist\_ds = mnist\_ds.repeat(repeat\_size)  
  
 **return** mnist\_ds  
  
**import** mindspore.nn **as** nn  
**from** mindspore.common.initializer **import** Normal  
  
  
*#* ***Todo 根据LeNet5网络结构神经网络*class** LeNet5(nn.Cell):  
 *"""  
 Lenet网络结构  
 """* **def** \_\_init\_\_(self, num\_class=10, num\_channel=1):  
 super(LeNet5, self).\_\_init\_\_()  
 self.conv1 = nn.Conv2d(num\_channel, 6, 5, pad\_mode=**'valid'**)  
 self.conv2 = nn.Conv2d(6, 16, 5, pad\_mode=**'valid'**)  
 self.fc1 = nn.Dense(16 \* 5 \* 5, 120, weight\_init=Normal(0.02))  
 self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02))  
 self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02))  
 self.relu = nn.ReLU()  
 self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)  
 self.flatten = nn.Flatten()  
  
 **def** construct(self, x):  
 *# 使用定义好的运算构建前向网络* x = self.max\_pool2d(self.relu(self.conv1(x)))  
 x = self.max\_pool2d(self.relu(self.conv2(x)))  
 x = self.flatten(x)  
 x = self.relu(self.fc1(x))  
 x = self.relu(self.fc2(x))  
 x = self.fc3(x)  
 **return** x  
  
**class** StepLossAccInfo(Callback):  
 **def** \_\_init\_\_(self, model, eval\_dataset, steps\_loss, steps\_eval):  
 self.model = model  
 self.eval\_dataset = eval\_dataset  
 self.steps\_loss = steps\_loss  
 self.steps\_eval = steps\_eval  
  
 **def** step\_end(self, run\_context):  
 cb\_params = run\_context.original\_args()  
 cur\_epoch = cb\_params.cur\_epoch\_num  
 cur\_step = (cur\_epoch-1)\*1875 + cb\_params.cur\_step\_num  
 self.steps\_loss[**"loss\_value"**].append(str(cb\_params.net\_outputs))  
 self.steps\_loss[**"step"**].append(str(cur\_step))  
 **if** cur\_step % 125 == 0:  
 acc = self.model.eval(self.eval\_dataset, dataset\_sink\_mode=**False**)  
 self.steps\_eval[**"step"**].append(cur\_step)  
 self.steps\_eval[**"acc"**].append(acc[**"Accuracy"**])  
  
*# 实例化网络*net = LeNet5()  
  
*# MindSpore支持的损失函数有SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss等。*net\_loss = nn.SoftmaxCrossEntropyWithLogits (sparse=**True**, reduction=**'mean'**)  
  
*# MindSpore支持的优化器有Adam、AdamWeightDecay、Momentum等。这里使用Momentum优化器为例。*net\_opt = nn.Momentum (net.trainable\_params(), learning\_rate=0.01, momentum=0.9)  
  
**from** mindspore.train.callback **import** ModelCheckpoint, CheckpointConfig  
  
config\_ck = CheckpointConfig(save\_checkpoint\_steps=1875, keep\_checkpoint\_max=10)  
  
ckpoint = ModelCheckpoint(prefix=**"checkpoint\_lenet"**, config=config\_ck)  
  
**from** mindspore.nn **import** Accuracy  
**from** mindspore.train.callback **import** LossMonitor  
**from** mindspore **import** Model  
  
**def** train\_net(model, epoch\_size, data\_path, repeat\_size, ckpoint\_cb, sink\_mode):  
 *"""定义训练的方法"""  
 # 加载训练数据集* ds\_train = create\_dataset(os.path.join(data\_path, **"train"**), 32, repeat\_size)  
 eval\_dataset = create\_dataset(os.path.join(mnist\_path, **"test"**), 32)  
 steps\_loss = {**"step"**: [], **"loss\_value"**: []}  
 steps\_eval = {**"step"**: [], **"acc"**: []}  
 step\_loss\_acc\_info = StepLossAccInfo(model, eval\_dataset, steps\_loss, steps\_eval)  
 model.train(epoch\_size, ds\_train, callbacks=[ckpoint\_cb, LossMonitor(125), step\_loss\_acc\_info],  
 dataset\_sink\_mode=sink\_mode)  
  
**def** test\_net(model, data\_path):  
 *"""定义验证的方法"""* param\_dict = load\_checkpoint(**"checkpoint\_lenet-1\_1875.ckpt"**)  
 load\_param\_into\_net(net, param\_dict)  
 ds\_eval = create\_dataset(os.path.join(mnist\_path, **"test"**))  
 acc = model.eval(ds\_eval, dataset\_sink\_mode=**False**)  
 print(**"Acc:{}"**.format(acc))  
  
train\_epoch = 1  
mnist\_path = **"./datasets/MNIST\_Data"**dataset\_size = 1  
model = Model(net, net\_loss, net\_opt, metrics={**"Accuracy"**: Accuracy()})  
train\_net(model, train\_epoch, mnist\_path, dataset\_size, ckpoint, **False**)  
test\_net(model, mnist\_path)  
  
**from** mindspore **import** load\_checkpoint, load\_param\_into\_net  
*# 加载已经保存的用于测试的模型*param\_dict = load\_checkpoint(**"checkpoint\_lenet-1\_1875.ckpt"**)  
*# 加载参数到网络中*load\_param\_into\_net(net, param\_dict)  
  
**import** numpy **as** np  
**from** mindspore **import** Tensor  
  
*# 定义测试数据集，batch\_size设置为1，则取出一张图片*ds\_test = create\_dataset(os.path.join(mnist\_path, **"test"**), batch\_size=1).create\_dict\_iterator()  
data = next(ds\_test)  
  
*# images为测试图片，labels为测试图片的实际分类*images = data[**"image"**].asnumpy()  
labels = data[**"label"**].asnumpy()  
  
*# 使用函数model.predict预测image对应分类*output = model.predict(Tensor(data[**'image'**]))  
predicted = np.argmax(output.asnumpy(), axis=1)  
  
*# 输出预测分类与实际分类*print(**f'Predicted: "{**predicted[0]**}", Actual: "{**labels[0]**}"'**)

**附录二 大作业代码**

**import** pandas **as** pd  
**import** numpy **as** np  
**import** seaborn **as** sns  
**import** matplotlib.pyplot **as** plt  
**import** statistics **as** st  
**import** missingno **as** msno  
**import** warnings  
warnings.filterwarnings(**"ignore"**)  
  
missing\_values\_set = [**""**, **"--"**, **"?"**, **"na"**, **"NAN"**, **"nan"**, **''**] *# 将这些关键字都视为空值（NAN）*test\_data = pd.read\_csv(**"./test.csv"**, na\_values=missing\_values\_set)  
train\_data = pd.read\_csv(**"./train.csv"**, na\_values=missing\_values\_set)  
  
*# 绘制训练集数值属性相关性示意图*correlation\_train = train\_data.corr()  
mask = np.zeros\_like(correlation\_train)  
mask[np.triu\_indices\_from(mask)] = **True**sns.heatmap(correlation\_train, mask=mask, xticklabels=correlation\_train.columns, yticklabels=correlation\_train.columns, linewidths=0.6, cmap=**'Blues'**)  
plt.title(**"训练集中的数值属性相关性示意图"**)  
plt.show()  
  
frames = [train\_data, test\_data]  
full\_dataset = pd.concat(frames, sort=**False**, ignore\_index=**True**) *# 将训练集和测试集合并*full\_dataset = pd.DataFrame(data=full\_dataset)  
SalePrice = full\_dataset[**"SalePrice"**]  
full\_dataset.drop(columns=[**'SalePrice'**])  
SalePrice.fillna(**""**, inplace=**True**) *# 将空值变为""*ID = pd.DataFrame(data=full\_dataset[**"Id"**])  
  
*# 绘制每列数据的密集程度图（非空）*msno.matrix(full\_dataset, labels=**True**)  
plt.xticks(rotation=**'vertical'**, fontsize=10)  
plt.title(**"各属性的空值占比（NAN）示意图"**)  
plt.show()  
  
*# print(full\_dataset.info())  
  
# 数据预处理preprocessing，搜索数据集中空值超过600个的列并删去*many\_empty = [attribute **for** attribute **in** full\_dataset **if** full\_dataset[attribute].isnull().sum() > 600] *##['Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature']*full\_dataset = full\_dataset.drop(columns=many\_empty)  
  
*# 获得数值和类别（非数值）属性*numerical\_attr = [i **for** i **in** full\_dataset.select\_dtypes(include=**'number'**)]  
categorical\_attr = [i **for** i **in** full\_dataset.select\_dtypes(include=**'object'**)]  
  
*# 填充剩余列的缺失值。数值属性用这一列的平均值填充，类别属性用这一列出现次数最多的属性填充***for** attributes **in** full\_dataset:  
 **if** attributes **in** numerical\_attr:  
 full\_dataset[attributes] = full\_dataset[attributes].fillna(value=full\_dataset[attributes].median())  
 **else**:  
 full\_dataset[attributes] = full\_dataset[attributes].fillna(value=full\_dataset[attributes].value\_counts().idxmax())  
  
*# 下面会将类别属性数值化，并利用RFC和SelectFromModel筛选出重要的非数值属性***from** sklearn.ensemble **import** RandomForestClassifier *# 随机森林分类器***from** sklearn.feature\_selection **import** SelectFromModel *# 特征选择***from** sklearn.model\_selection **import** train\_test\_split *# 避免overfitting，只关注训练集*X = pd.DataFrame(data=train\_data[categorical\_attr])  
X.drop(columns=[**'SalePrice'**], inplace=**True**)  
*# print(X)*Y = train\_data[**'SalePrice'**] *# X和Y是训练集的属性，其中X不含售价，Y只含售价***for** at **in** X:  
 X = pd.concat([X, pd.get\_dummies(X[at], prefix=at, drop\_first=**True**)], axis=1, sort=**False**) *# 利用get\_dummies函数将类别属性转化为编码* X.drop(columns=[at], inplace=**True**)  
  
X\_train\_st, X\_test\_st, y\_train\_st, y\_test\_st = train\_test\_split(X, Y, test\_size=0.25) *# 对X和Y分别划分训练集和测试集，比例为3:1  
  
# 特征选择  
# 下面首先对类别属性进行操作*rfc = RandomForestClassifier(n\_estimators=150) *# 创建150棵树进行随机森林处理*sel = SelectFromModel(estimator=rfc, threshold=**'mean'**) *# 利用RFC作为估计器，选取重要度大于全部重要度平均值的属性*sel.fit(X, Y)  
print(**"门限threshold: "**, sel.estimator\_.feature\_importances\_.mean())  
  
*# print(sel.get\_support()) # 打印属性是否高于threshold的矩阵（T/F）*selected\_features = X\_train\_st.columns[(sel.get\_support())] *# 获取属性名称及取值*print(**"高于门限值的类别属性取值个数："**, len(selected\_features), **"\n这些属性的值是："**)  
print(selected\_features)  
  
importances = sel.estimator\_.feature\_importances\_ *# 获取类别属性特征的重要度并从高到低排序*Features\_importance = pd.DataFrame(data=X.columns.values, columns=[**'Features'**])  
Features\_importance[**'Importance'**] = importances  
Features\_importance = Features\_importance.sort\_values(by=[**'Importance'**], axis=0, ascending=**False**)  
print(**"这些类别属性取值的重要度："**)  
print(Features\_importance)  
  
print(**"全部类别属性取值的重要度："**, importances)  
indices = np.argsort(importances)[::-1] *# 返回需要对数组进行排序的索引*plt.figure(figsize=(10, 10))  
plt.title(**"从高到低排列的类别属性取值的重要度"**)  
plt.bar(range(X.shape[1]), importances[indices], color=**"r"**, align=**"center"**)  
plt.xticks(range(X.shape[1]), Features\_importance[**'Features'**], rotation=**'vertical'**)  
plt.xlim([-1, 9.5])  
plt.show()  
  
*# 将排名前十的类别属性数值化*cat\_to\_num = [**"GarageFinish"**, **"LotShape"**, **"BsmtExposure"**, **"BsmtFinType1"**, **"LotConfig"**, **"HouseStyle"**, **"MasVnrType"**]  
**for** categoricals **in** cat\_to\_num:  
 full\_dataset = pd.concat([full\_dataset, pd.get\_dummies(full\_dataset[categoricals], prefix=categoricals, drop\_first=**True**)], axis=1, sort=**False**)  
 full\_dataset.drop(columns=[categoricals], inplace=**True**)  
 print(**"类别属性："**, categoricals) *# 分别将上述类别属性数值化并展示结果* print(full\_dataset)  
  
numerical\_attr = [i **for** i **in** full\_dataset.select\_dtypes(include=**'number'**)]  
  
*# 绘制训练集的完整属性相关性示意图*correlation = full\_dataset.corr()  
mask = np.zeros\_like(correlation)  
mask[np.triu\_indices\_from(mask)] = **True**sns.heatmap(correlation, mask=mask, xticklabels=correlation.columns, yticklabels=correlation.columns, linewidths=0.6, cmap=**'Blues'**)  
plt.title(**'训练集的完整属性相关性示意图'**)  
plt.show()  
  
*# 相关性分析*correlation[**'CORR'**] = correlation.sum(axis=0) *# 计算每种属性与saleprice的相关系数，并创建一个名为CORR的列*feature\_correlation = correlation[**'CORR'**].sort\_values(ascending=**False**)  
print(**"属性相关性分析：\n"**, feature\_correlation)  
  
*# 方差分析*vanalysis=[]  
**for** attribute **in** numerical\_attr:  
 mean = st.mean(full\_dataset[attribute])  
 *# print('\n head of {} is : {} '.format(attribute, full\_dataset[attribute].head()))  
 # full\_dataset[attribute].value\_counts().plot(kind='hist', bins=20) #绘制每种属性的值与出现频率的直方图  
 # plt.title(attribute)  
 # plt.show()* standard\_variance = st.stdev(full\_dataset[attribute]) *# 均值方差标准化* variance = st.variance(full\_dataset[attribute])  
 vanalysis.append([attribute, mean, variance, standard\_variance])  
  
vanalysis = pd.DataFrame(vanalysis, columns=[**'属性'**, **'平均值'**, **'方差'**, **'标准化方差'**]).sort\_values(by=[**'标准化方差'**], ascending=**False**)  
print(**"方差分析：\n"**, vanalysis)  
  
*# 使用Z-score实现数据标准化***from** scipy.stats **import** zscore  
  
old\_full\_dataset = full\_dataset  
full\_dataset[numerical\_attr] = zscore(full\_dataset[numerical\_attr]) *# 对目前的数据集计算zscore  
  
# PCA对数据进行降维处理***from** sklearn.decomposition **import** PCA  
pca\_search = PCA().fit(full\_dataset[numerical\_attr]) *# fit pca*fig, ax = plt.subplots()  
print(**"PCA的各特征（可解释性）方差贡献率：\n"**, pca\_search.explained\_variance\_ratio\_)  
y = np.cumsum(pca\_search.explained\_variance\_ratio\_)  
xi = np.arange(0, len(y), step=1) *# 特征数量*plt.plot(xi, y, marker=**'o'**, color=**'b'**)  
plt.xlabel(**'成分数量'**)  
plt.ylabel(**'Cumulative variance ratio (%)'**) *# 累计方差贡献率，取0.72之前的属性*plt.title(**'The number of components after dimension reduction'**)  
plt.axhline(y=0.72, color=**'r'**, linestyle=**'-'**)  
plt.text(0.5, 0.72, **'72% cut-off threshold'**, color=**'red'**, fontsize=12)  
ax.grid(axis=**'x'**)  
plt.show()  
  
data\_to\_pca = full\_dataset.drop(columns=[**"Id"**]) *# 舍去id属性*numerical\_attr\_pca = [i **for** i **in** data\_to\_pca.select\_dtypes(include=**'number'**)] *# 获取PCA处理后的数值属性*pca = PCA(n\_components=24) *# 降维后的维度为24，即保留24种特征*principal\_analysis = pca.fit\_transform(data\_to\_pca[numerical\_attr\_pca]) *# 拟合*distortions = []  
principal\_analysis = pd.DataFrame(data=principal\_analysis, columns=[**'P1'**, **'P2'**, **'P3'**, **'P4'**, **'P5'**, **'P6'**, **'P7'**, **'P8'**, **'P9'**, **'P10'**, **'P11'**, **'P12'**, **'P13'**, **'P14'**, **'P15'**, **'P16'**, **'P17'**, **'P18'**, **'P19'**, **'P20'**, **'P21'**, **'P22'**, **'P23'**, **'P24'**]) *# 创建dataframe*print(**"主成分分析（PCA）：\n "**, principal\_analysis)  
print(**"解释方差比例："**, pca.explained\_variance\_ratio\_)  
  
**from** scipy.spatial.distance **import** cdist  
**from** sklearn.cluster **import** KMeans  
  
*#计算聚类数量*inertias = [] *# 样本距离最近的聚类中心的总和*distortions = [] *# 每个簇的质点与簇内样本点的平方距离误差和，即畸变程度*K = range(1, 15)  
**for** k **in** K:  
 kmeanModel = KMeans(n\_clusters=k).fit(X)  
 kmeanModel.fit(X)  
 distortions.append(sum(np.min(cdist(X, kmeanModel.cluster\_centers\_, **'euclidean'**), axis=1)) / X.shape[0])  
 inertias.append(kmeanModel.inertia\_)  
  
*# 利用kmeans算法的elbow method原理确定合适的聚类数量*plt.plot(K, distortions, **'bx-'**)  
plt.xlabel(**'k'**)  
plt.ylabel(**'Distortion'**)  
plt.title(**'The Elbow Method'**)  
plt.show()  
*#elbow with inertias:  
# plt.plot(K, inertias, 'bx-')  
# plt.xlabel('k')  
# plt.ylabel('inertias')  
# plt.title('The Elbow Method')  
# plt.show()*pca\_data = principal\_analysis  
clustering = KMeans(n\_clusters=2, random\_state=4) *# 二分类*clustering.fit(pca\_data)  
  
predict = clustering.predict(pca\_data)  
pca\_data[**"Cluster"**] = predict *# 在pca\_data中创建三列：聚类（0或1）、Id、售价*pca\_data[**"Id"**] = ID.values  
pca\_data[**"SalePrice"**] = SalePrice.values  
  
cluster\_0 = pd.DataFrame(data=pca\_data[pca\_data[**'Cluster'**] == 0], columns=[**'P1'**, **'P2'**, **'P3'**, **'P4'**, **'P5'**, **'P6'**, **'P7'**, **'P8'**, **'P9'**, **'P10'**, **'P11'**, **'P12'**, **'P13'**, **'P14'**, **'P15'**, **'P16'**, **'P17'**, **'P18'**, **'P19'**, **'P20'**, **'P21'**, **'P22'**, **'P23'**, **'P24'**, **'Id'**, **'Cluster'**, **'SalePrice'**])  
cluster\_1 = pd.DataFrame(data=pca\_data[pca\_data[**'Cluster'**] == 1], columns=[**'P1'**, **'P2'**, **'P3'**, **'P4'**, **'P5'**, **'P6'**, **'P7'**, **'P8'**, **'P9'**, **'P10'**, **'P11'**, **'P12'**, **'P13'**, **'P14'**, **'P15'**, **'P16'**, **'P17'**, **'P18'**, **'P19'**, **'P20'**, **'P21'**, **'P22'**, **'P23'**, **'P24'**, **'Id'**, **'Cluster'**, **'SalePrice'**])  
  
print(**"第一个聚类的数量："**, len(cluster\_0))  
print(**"第二个聚类的数量："**, len(cluster\_1))  
  
*# 对两个聚类分别进行训练和线性回归***from** sklearn.linear\_model **import** LinearRegression  
result = []  
id = []  
clusters = [cluster\_0, cluster\_1]  
  
**for** i **in** range(0, 2):  
 train = []  
 test = []  
 **for** row **in** clusters[i].values:  
 x = row[26] *# SalePrice* **if** x == **""**: *# 根据是否已有saleprice，确定每一行进入训练集还是测试集* test.append(row)  
 **else**:  
 train.append(row)  
  
 train\_df = pd.DataFrame(train, columns=[**'P1'**, **'P2'**, **'P3'**, **'P4'**, **'P5'**, **'P6'**, **'P7'**, **'P8'**, **'P9'**, **'P10'**, **'P11'**, **'P12'**, **'P13'**, **'P14'**, **'P15'**, **'P16'**, **'P17'**, **'P18'**, **'P19'**, **'P20'**, **'P21'**, **'P22'**, **'P23'**, **'P24'**, **'Id'**, **'Cluster'**, **'SalePrice'**])  
 test\_df = pd.DataFrame(test, columns=[**'P1'**, **'P2'**, **'P3'**, **'P4'**, **'P5'**, **'P6'**, **'P7'**, **'P8'**, **'P9'**, **'P10'**, **'P11'**, **'P12'**, **'P13'**, **'P14'**, **'P15'**, **'P16'**, **'P17'**, **'P18'**, **'P19'**, **'P20'**, **'P21'**, **'P22'**, **'P23'**, **'P24'**, **'Id'**, **'Cluster'**, **'SalePrice'**])  
 Id = pd.DataFrame(data=test\_df[**"Id"**])  
  
 train\_df.drop(columns=[**"Cluster"**], inplace=**True**)  
 test\_df.drop(columns=[**"Cluster"**], inplace=**True**)  
  
 X\_train = train\_df.drop(columns=[**"Id"**, **"SalePrice"**])  
 Y\_train = train\_df[**"SalePrice"**]  
 X\_test = test\_df.drop(columns=[**"Id"**, **"SalePrice"**])  
 Y\_test = test\_df[**"SalePrice"**]  
  
 regression = LinearRegression(n\_jobs=-1)  
 regression.fit(X\_train, Y\_train)  
 prediction = regression.predict(X\_test)  
 id.append(Id)  
 result.append(prediction)  
  
result\_for\_cluster0 = pd.DataFrame(data=result[0], columns=[**"cluster\_predict"**]) *# 第一个聚类的预测结果*id0 = id[0]  
result\_for\_cluster0[**"Id"**] = id0.values  
  
result\_for\_cluster1 = pd.DataFrame(data=result[1], columns=[**"cluster\_predict"**]) *# 第二个聚类的预测结果*id1 = id[1]  
result\_for\_cluster1[**"Id"**] = id1.values  
  
*# 得到结果*saleprice = pd.concat([result\_for\_cluster0, result\_for\_cluster1], axis=0, sort=**False**)  
saleprice.sort\_values(by=[**'Id'**], inplace=**True**)  
saleprice.rename(columns={**"cluster\_predict"**: **"SalePrice"**}, inplace=**True**)  
  
saleprice = pd.DataFrame(data=saleprice, columns=[**"Id"**, **'SalePrice'**])  
saleprice.to\_csv(**r"./saleprice.csv"**, index=**False**)