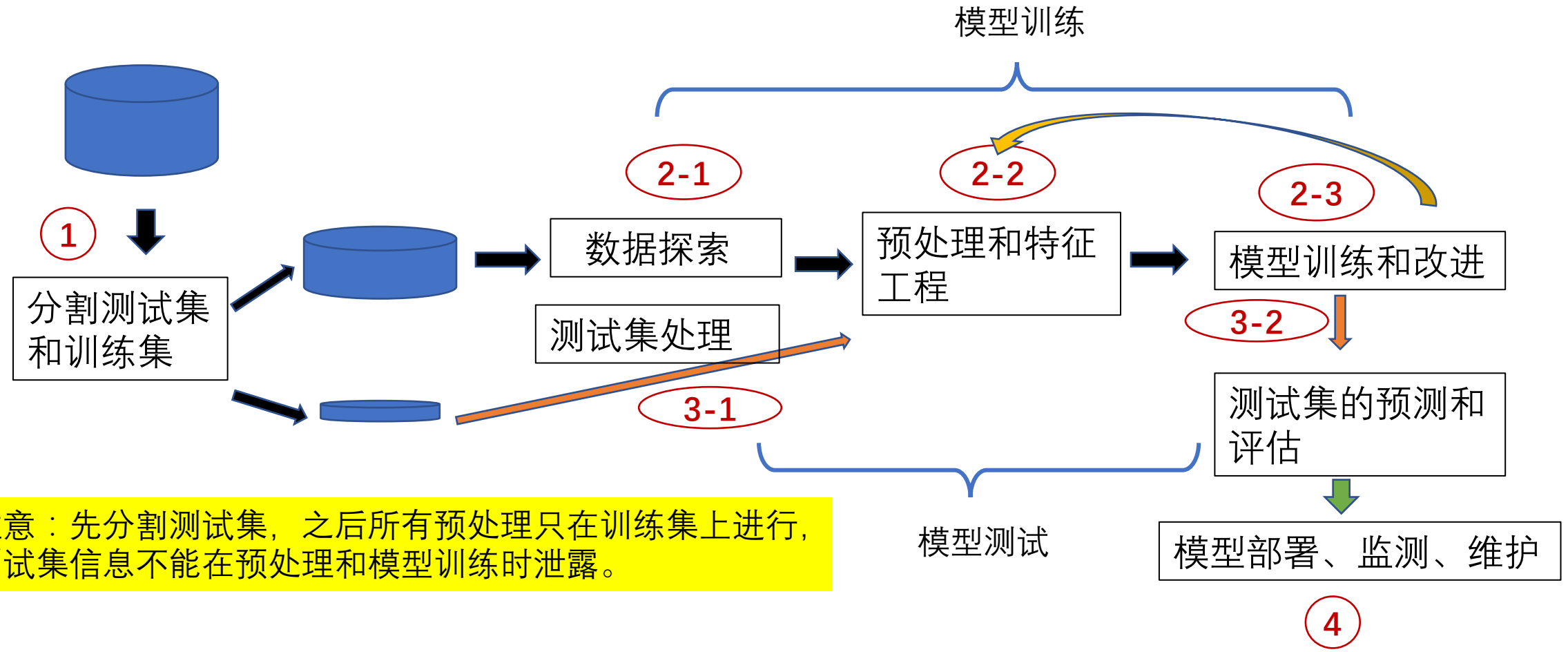
分类总结与案例实现

分类任务的整体流程



注意：先分割测试集，之后所有预处理只在训练集上进行，测试集信息不能在预处理和模型训练时泄露。

模型训练

过程：给定带标签的训练集，训练一个分类模型。

主要包括以下步骤：

1. 数据探索。查看数据集大小、特征数目和类型、分布、特征之间相关性、缺失情况等。
2. 预处理和特征工程。包括：异常值检测和处理、缺失值填充、不同类型特征的转换、特征规范化。根据特征数目的大小、特征之间的相关性以及计算和存储限制，决定是否要进行降维。
3. 选择分类算法进行模型训练。训练模型并基于验证集进行超参数调优。根据评估结果可能需要重新回到前面一步（预处理和特征工程）进行模型改进。

模型预测

过程：基于测试集对训练好的模型进行性能评估。

主要包括以下步骤：

1. 测试集处理。按照训练集的处理方法，对测试集进行处理。
2. 用训练好的模型预测测试集中样本的标签。

注意：步骤1中测试集的处理与训练集的处理方法一致，包括使用训练集的统计信息。比如对测试集中特征做规范化时，用到的最小值(min)，最大值,(max)，均值(mean)，标准差(std)都基于训练集。

快速查看数据

把数据用pd.DataFrame表示

1. 用.info()方法查看样本数目、特征数目、每个特征的数据类型和对应的non-null记录数目。
2. 每个特征的缺失值情况：df.isnull().sum() 得到缺失个数或df.isnull().any()判断是否缺失。
3. 用.value_counts()查看类别型特征的取值及对应样本数。
4. 用.describe()查看数值型特征（浮点型、整型）的统计信息。
5. 用.hist()打印数值型特征的分布情况。

快速查看数据

```
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

total_bedrooms有缺失值，ocean_proximity不是数值类型

快速查看数据

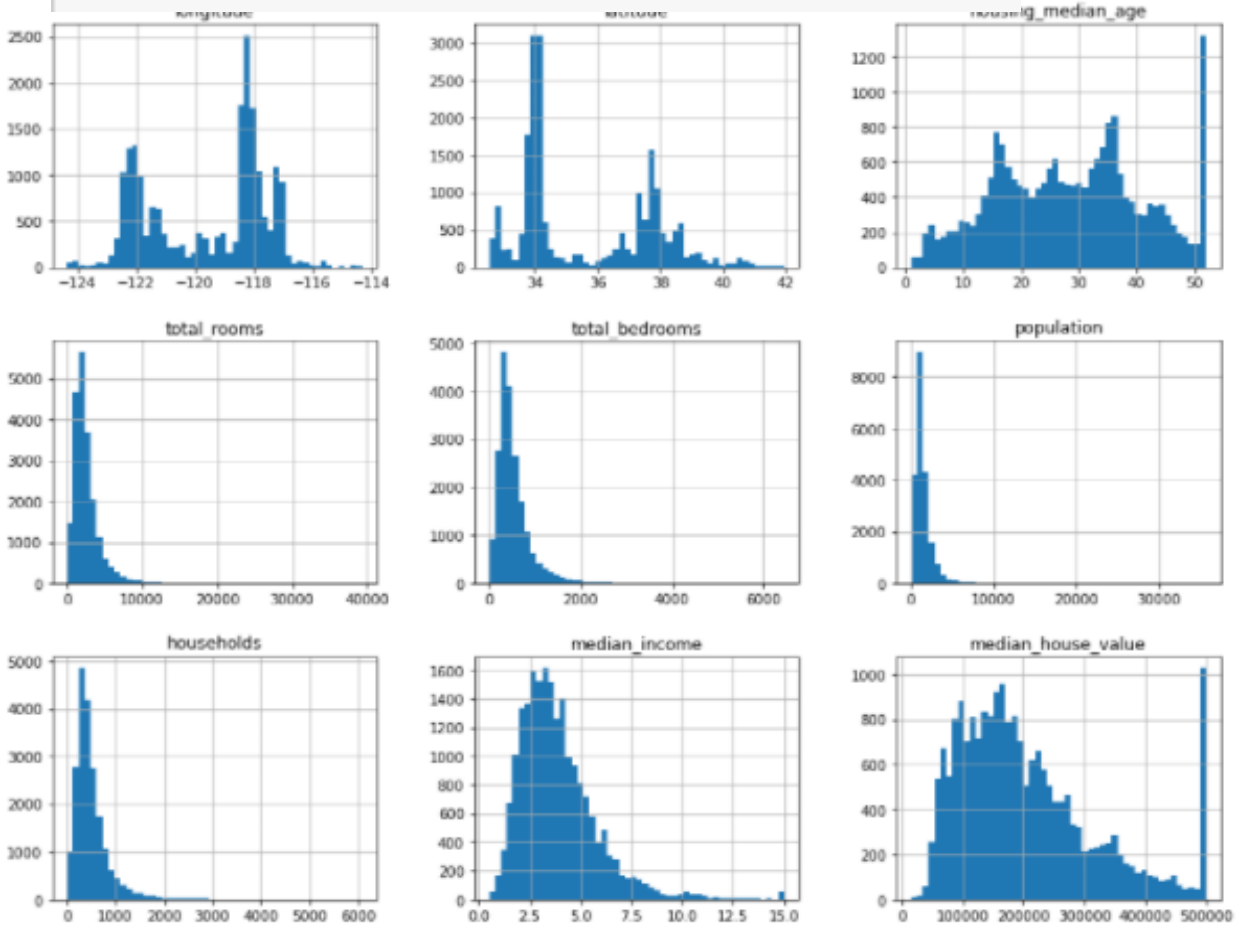
查看ocean_promixity的取值情况

```
housing["ocean_proximity"].value_counts()
```

```
<1H OCEAN      9136
INLAND         6551
NEAR OCEAN     2658
NEAR BAY       2290
ISLAND            5
Name: ocean_proximity, dtype: int64
```

查看其他**数值类型**特征的分布：统计信息或直方图

```
housing.hist(bins=50,figsize=(15,12))
plt.show()
```



不同特征的取值范围相差非常大，很多特征为“长尾”分布

分割测试集和训练集

- 设置随机种子，使得每次都产生一样的随机分割
- 数据集较小，或者类别之间大小很不平衡，采用分层采样

```
from sklearn.model_selection import StratifiedShuffleSplit
split=StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing['income_cat']):
    strat_train_set=housing.loc[train_index]
    strat_test_set=housing.loc[test_index]
```

一般20%作为测试集。如果数据集很大，那么这个比例可以减小，如对于100万的数据集，1%测试集即1万条就足够。

假设’ income_cat’ 为要采样的每个类别，通过对 “median_income” 截断得到。

```
housing[“income_cat”]=pd.cut(housing[“median_income”],
                             bins=[0., 1.5, 3.0, 4.5, 6.0, np.inf],
                             labels=[1,2,3,4,5])
```

案例：信用违约预测

背景说明

风险已经成为了今年金融市场的重要主题之一，银行作为贷方，随时都面临着借贷者违约的风险。传统的专家规则在金融科技时代逐渐过时，机器学习和金融业务的交叉也延伸到信贷领域。违约预测就是其中一重要应用。本实验基于信贷业务场景中一个月内的抽样数据，数据集有34个维度，Target表示客户在接下来一个月是否有违约。模型生成后可使用当前月的数据预测接下来一个月客户是否会违约。

问题分析

违约预测只有违约和没有预约两种结果，是个二分类问题。针对二分类问题，可使用的算法有 Logistic regression、朴素贝叶斯、支持向量机、决策树等。本实验选用上课讲过的最近邻和决策树做对比。

考虑到两个类别样本极度不均衡，模型评价采用混淆矩阵，precision, recall以及f1_score。通过过采样处理类别不平衡，以及调参来提升分类器的泛化性能。

导入数据并查看基本信息

第一步： 导入数据

```
In [52]:  ##读取数据
          path1=r"dataset-credit-default.csv"
          df = pd.read_csv(path1,encoding='utf-8')
```

第二步： 查看数据了解基本信息

```
In [4]:  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6462 entries, 0 to 6461
Data columns (total 34 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Cust_No                 6462 non-null   int64
 1   Target                  6462 non-null   int64
 2   Nation                  4325 non-null   float64
 3   Birth_Place             6462 non-null   int64
 4   Gender                  6462 non-null   int64
 5   Age                     6462 non-null   int64
 6   Marriage_State          4465 non-null   float64
 7   Highest Education       4529 non-null   float64
 8   House_State             3891 non-null   float64
 9   Work_Years              6462 non-null   int64
 10  Unit_Kind               2445 non-null   object
 11  Title                   2767 non-null   float64
 12  Occupation              2875 non-null   object
 13  Duty                    5693 non-null   float64
```

```
In [5]:  ##查看Target的分布，是否违约（1是，0否）
         df['Target'].value_counts()
```

```
Out[5]: 0    6341
        1     121
        Name: Target, dtype: int64
```

```
In [53]:  ### 删除无分类意义的特征列Cust_No
          del df['Cust_No']
```

注意：
Unit_Kind（工作单位性质）、Title（职务）、Industry（行业）和Occupation（岗位）被认为是类别型特征。虽然Title和Industry的数据类型为float64。

分出测试集

在实际应用中，测试集是未知数据，不能用于模型训练和调参。 将数据集作8:2的切分。由于此处样本量有6000多条，9:1分即测试集保留600多条也足够。以下采用分层采样进行分割。

```
#分割测试集
from sklearn.model_selection import StratifiedShuffleSplit
split=StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(df, df['Target']):
    strat_train_set=df.loc[train_index]
    strat_test_set=df.loc[test_index]
```

```
y_train=strat_train_set['Target'].copy()
X_train=strat_train_set.drop('Target',axis=1)
y_test=strat_test_set['Target'].copy()
X_test=strat_test_set.drop('Target',axis=1)
```

```
y_train.value_counts()
```

```
0    5072
1      97
Name: Target, dtype: int64
```

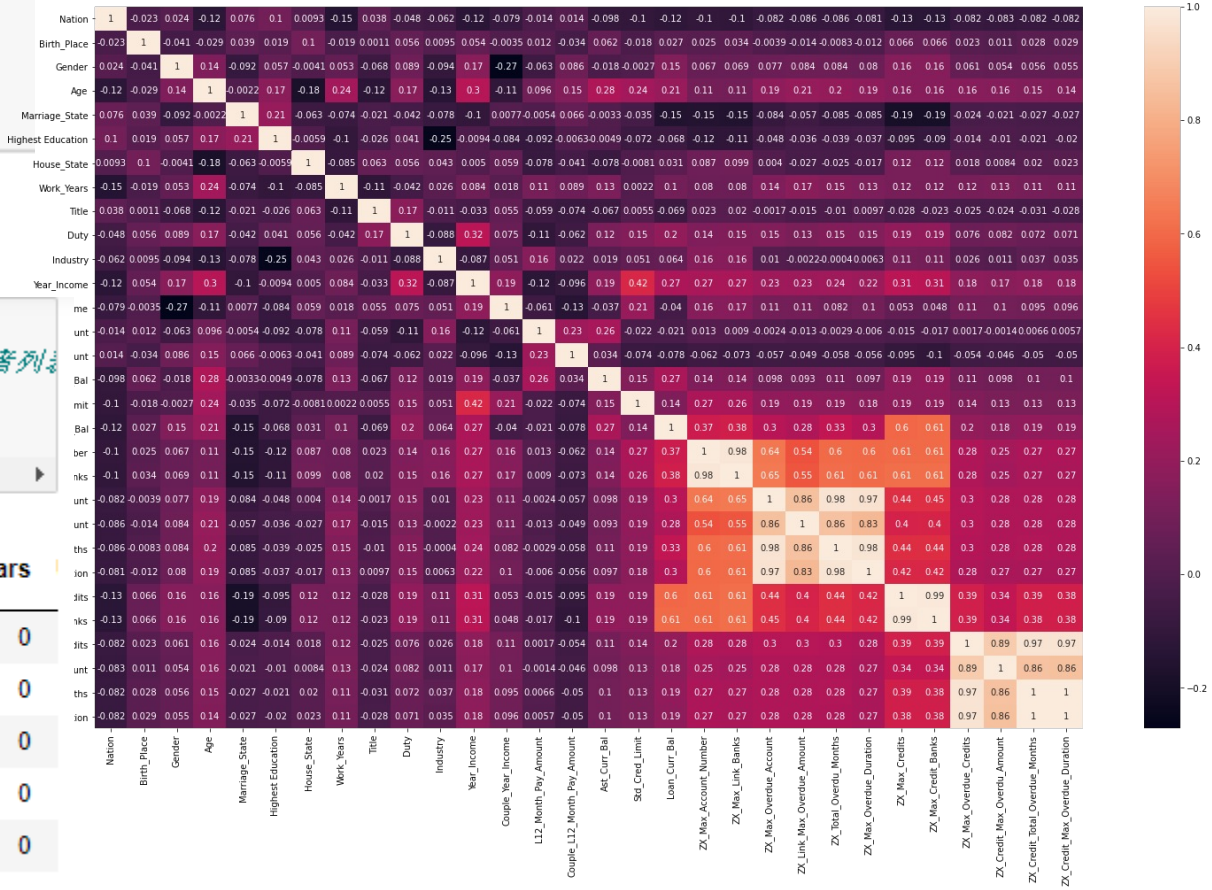
```
y_test.value_counts()
```

```
0    1269
1      24
Name: Target, dtype: int64
```

探索/预处理训练集

预处理1：通过相关系数，找出冗余特征并删除

```
#步骤 1——相关性分析，可视化观察特征相关性
#计算特征相关性
corr_matrix = X_train.corr(method='spearman')
```



丢弃部分特征使得任何两个特征之间的相关系数 < 设定阈值（如0.8）

```
#预处理1：丢弃特征对中的一个
cols_to_drop = np.unique([col[1] for col in cols_pair_to_drop])  #对于一维数组或者列表
X_train.drop(cols_to_drop, axis=1, inplace=True)
X_train.head()
```

	Nation	Birth_Place	Gender	Age	Marriage_State	Highest Education	House_State	Work_Years
1208	NaN	330602	1	57	NaN	NaN	NaN	0
4394	NaN	330282	1	35	NaN	NaN	NaN	0
3555	NaN	330621	1	57	20.0	71.0	1.0	0
3484	1.0	330621	1	59	20.0	81.0	1.0	0
2465	NaN	330621	0	23	10.0	71.0	3.0	0

5 rows × 24 columns

通过以上操作后特征数减少到24个

预处理2：分别对数值型和类别型特征的缺失值进行处理

```
#打印出缺失率最高的前15个特征以及对应的缺失率
df_missing_stat = pd.DataFrame(X_train.isnull().sum()/X_train.shape[0],columns=['missing_rate']).reset_index()
df_missing_stat.sort_values(by='missing_rate',ascending=False)[:15]
```

	index	missing_rate
16	Couple_L12_Month_Pay_Amount	0.694138
14	Couple_Year_Income	0.694138
8	Unit_Kind	0.622558
9	Title	0.573225
12	Industry	0.563358
10	Occupation	0.555620
6	House_State	0.400077
0	Nation	0.331592
4	Marriage_State	0.309731
5	Highest Education	0.300251
11	Duty	0.121300
7	Work_Years	0.000000
1	Birth_Place	0.000000
13	Year_Income	0.000000
3	Age	0.000000

可以看出缺失率最高的是Couple_Year_Income、Couple_L12_Month_Pay_Amount、Unit_Kind、Title、Industry和Occupation，缺失率都超过了50%。

其中，Couple_Year_Income和Couple_L12_Month_Pay_Amount是数值型变量；而Unit_Kind、Title、Industry和Occupation是类别型变量。

以下分别对数值型和类别型变量进行缺失值处理。

数值型特征的缺失值和异常值处理

```
def out_miss_fit(X_train_item,item):
    iqr = X_train_item.quantile(0.75) - X_train_item.quantile(0.25)
    q_abnormal_L = X_train_item < X_train_item.quantile(0.25) - 1.5 * iqr
    q_abnormal_U = X_train_item > X_train_item.quantile(0.75) + 1.5 * iqr
#取异常点的索引
    print(item + '中有' + str(q_abnormal_L.sum() + q_abnormal_U.sum()) + '个异常值')
    item_outlier_index = X_train_item[q_abnormal_L|q_abnormal_U].index
###得到不包含异常值的数据
    X_train_item_wo=X_train_item.copy()
    X_train_item_wo.drop(index = item_outlier_index,inplace=True)
    new_median=X_train_item_wo.median()
    new_max=X_train_item_wo.max()
    new_min=X_train_item_wo.min()
    return new_max, new_min, new_median

def out_miss_trans(X_train_item, new_max, new_min, new_median):
#用去除异常值的中位数填补缺失值
    X_train_item = X_train_item.fillna(new_median)
#用更新后的最大/最小替换异常值
    X_train_item.iloc[X_train_item > new_max] = new_max
    X_train_item.iloc[X_train_item < new_min] = new_min
    return X_train_item
```

方式一：删除

删除存在缺失/异常的记录（如果存在缺失的记录数目很少），或者删除对应属性（如果该属性缺失率非常高）

方式二：填充/替换

缺失值：可以计算去除异常值的中位数来填充缺失值

异常值：用删除异常值后的最大/最小值替换。

方式二可以保持样本数和特征数不变。

```
num_attrs =['Couple_Year_Income','Couple_L12_Month_Pay_Amount']
for item in num_attrs:
    new_max, new_min, new_median=out_miss_fit(X_train[item],item)
    #print(new_max, new_min, new_median)
    X_train[item]=out_miss_trans(X_train[item], new_max, new_min, new_median)
```

类别型特征缺失值处理

- 类别型变量缺失值，Unit_Kind、Title、Industry和Occupation

```
def cat_attr_fit(X_train):
### 查看仍有少量缺失值的特征
    null_col=[]
    for col in X_train.columns:
        if X_train[col].isnull().sum()>0:
            null_col.append(col)
    return null_col
def cat_attr_trans(X_train, null_col):
###使用众数填充缺失值
    for col_to_fill in null_col:
        X_train[col_to_fill] = X_train[col_to_fill].fillna(X_train[col_to_fill].mode()[0])
    return X_train
```

```
null_col=cat_attr_fit(X_train)
cat_attr_trans(X_train, null_col)
```

预处理3 类别型特征转数值型

最常用的方法是对非数值型特征（object类型）取值直接建立数值索引，即用整型数值代替字符串，或者进行独热编码。 前者不会增加特征数目，而后者会把一个类别特征编码成d维one-hot向量。

pandas 和 sklearn都有这两种编码的实现： pd.factorize(), pd.get_dummies() 或 sklearn.preprocessing.OrdinalEncoder, sklearn.preprocessing.OneHotEncoder

```
# 下面尝试 pd.factorize
def cal2num(X_train):
### 查看数据集剩余的名称性特征
    con_col=[]
    for col in X_train.columns:
        if X_train.dtypes[col] == np.object:
            con_col.append(col)
    for item in con_col:
        X_train[item] = pd.factorize(X_train[item])[0]
    return X_train
```

```
X_train=cal2num(X_train)
```

预处理4 对所有特征进行标准化

```
# 引入StandardScaler标准化工具库
from sklearn.preprocessing import StandardScaler
#对训练集做标准化
std_scaler = StandardScaler().fit(X_train)
X_train_std = std_scaler.transform(X_train)
```

第五步 模型训练和评估

因为该数据集只有两个类别，且类别大小差别很大，考虑打印混淆矩阵查看结果。

注意：accuracy在这里不合适，因为即使每次都预测为不违约，accuracy也超过了98%。可以用precision，recall以及f1-score。

```
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier

clf=KNeighborsClassifier()
clf.fit(X_train_std, y_train)
# 查看在训练集的误差
y_pred=clf.predict(X_train_std)
conf_mat=confusion_matrix(y_train,y_pred)
print('训练集上的预测结果')
print(conf_mat)
```

```
训练集上的预测结果
[[5071    1]
 [  96    1]]
```

```
from sklearn import tree
clf=tree.DecisionTreeClassifier()
clf.fit(X_train_std, y_train)
# 查看在训练集的误差
y_pred=clf.predict(X_train_std)
conf_mat=confusion_matrix(y_train,y_pred)
print('训练集上的预测结果')
print(conf_mat)
```

```
训练集上的预测结果
[[5072    0]
 [   0   97]]
```

决策树在训练集上拟合很好，训练误差为0，但有可能是过拟合。

下面采用交叉验证进一步确定决策树的拟合情况

决策树交叉验证结果

```
from sklearn import tree
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
clf = tree.DecisionTreeClassifier(random_state=42)
y_pred=cross_val_predict(clf, X_train_std, y_train, cv=10)
conf_mat=confusion_matrix(y_train,y_pred)
print('训练集上交叉验证的预测结果')
print(conf_mat)
p=precision_score(y_train,y_pred)
r=recall_score(y_train,y_pred)
f1=f1_score(y_train,y_pred)
print(p, r, f1)
```

10折

```
训练集上交叉验证的预测结果
[[4943  129]
 [  91    6]]
0.04444444444444446 0.06185567010309278 0.05172413793103448
```

从上面交叉验证的结果看出，之前的决策树确实过拟合。因为对看到过的训练样本误差为0，而在交叉验证中，由于对每个保留的fold不出现在训练集中，结果就差很多

说明模型过拟合！

怎么办？

1. 从数据上想办法
2. 从模型上调整

处理类别不平衡：对正样本过采样

利用imblearn库中的over_sampling.SMOTE方法

```
from imblearn import over_sampling
print(y_train.value_counts())
#不同版本的SMOTE的实现有不同输入参数,具体看官方对应版本的例子
smote_model = over_sampling.SMOTE(random_state=7,k_neighbors=5)
X_train_std_res,y_train_res = smote_model.fit_resample(X_train_std,y_train)
print(y_train_res.value_counts())
```

```
0    5072
1      97
Name: Target, dtype: int64
1    5072
0    5072
Name: Target, dtype: int64
```

```
clf = tree.DecisionTreeClassifier(random_state=42)
y_pred=cross_val_predict(clf, X_train_std_res, y_train_res, cv=10)
conf_mat=confusion_matrix(y_train_res,y_pred)
print('训练集上交叉验证的结果')
print(conf_mat)
p=precision_score(y_train_res,y_pred)
r=recall_score(y_train_res,y_pred)
f1=f1_score(y_train_res,y_pred)
print(p, r, f1)
```

```
训练集上的预测结果
[[4812  260]
 [ 129 4943]]
0.95002882952143 0.9745662460567823 0.9621411192214113
```

通过过采样，交叉验证的结果提高很多，接下去用过采样数据训练好的模型预测测试集。

在预测集上的结果

对测试集进行同样预处理后得到预测结果

```
# 按照前面的处理步骤处理测试集
#删除冗余特征
X_test.drop(cols_to_drop,axis=1,inplace=True)
num_attrs =['Couple_Year_Income','Couple_L12_Month_Pay_Amount']
#处理
for item in num_attrs:
    X_test[item]=out_miss_trans(X_test[item], new_max, new_min, new_median)
null_col=cat_attr_fit(X_test)
cat_attr_trans(X_test, null_col)
X_test=cal2num(X_test)
X_test_std = std_scaler.transform(X_test)
```

```
clf.fit(X_train_std, y_train)
# 查看在测试集的误差
y_pred=clf.predict(X_test_std)
conf_mat=confusion_matrix(y_test,y_pred)
print('原训练数据训练的模型在测试集上的预测结果')
print(conf_mat)
p=precision_score(y_test,y_pred)
r=recall_score(y_test,y_pred)
f1=f1_score(y_test,y_pred)
print(p, r, f1)
```

原属训练数据训练的模型在测试集上的预测结果
[[1220 49]
 [23   1]]
0.02 0.041666666666666664 0.02702702702702703

对比前面结果发现，过采样确实起到了一定作用，特别是提高了recall。

```
clf=tree.DecisionTreeClassifier(random_state=42)
clf.fit(X_train_std_res, y_train_res)

# 查看在测试集的误差
y_pred=clf.predict(X_test_std)
conf_mat=confusion_matrix(y_test,y_pred)
print('用过采样后的数据训练的模型在测试集上的预测结果')
print(conf_mat)
p_res=precision_score(y_test,y_pred)
r_res=recall_score(y_test,y_pred)
f1_res=f1_score(y_test,y_pred)
print(p_res, r_res, f1_res)
```

用过采样后的数据训练的模型在测试集上的预测结果
[[1199 70]
 [21   3]]
0.0410958904109589 0.125 0.06185567010309278

基于网格搜索的超参数调优

```
from sklearn.model_selection import GridSearchCV
clf = tree.DecisionTreeClassifier(random_state=42)
param_grid = [
  { 'min_samples_leaf': [ 1, 3, 5], 'criterion':['gini','entropy']},
 ]

grid= GridSearchCV(clf, param_grid, scoring='recall', cv=5)
grid.fit(X_train_std_res, y_train_res)
print('best params:{}'.format(grid.best_params_))
print('best score:{}'.format(grid.best_score_))
```

```
best params:{'criterion': 'entropy', 'min_samples_leaf': 1}
best score:0.9749545768113407
```

⬅ 指定要逐一搜索的参数网格

⬅ 指定评价指标

```
y_pred=grid.predict(X_test_std)
conf_mat=confusion_matrix(y_test,y_pred)
print('通过调参后的模型在测试集上的预测结果')
print(conf_mat)
p_res_tune=precision_score(y_test,y_pred)
r_res_tune=recall_score(y_test,y_pred)
f1_res_tune=f1_score(y_test,y_pred)
print(p_res_tune, r_res_tune, f1_res_tune)
```

```
通过调参后的模型在测试集上的预测结果
[[1217   52]
 [  21    3]]
0.05454545454545454 0.125 0.0759493670886076
```

对比调参前后，发现通过调参，precision, f1有一点提高,说明调参有作用。但对recall没有改进。可以尝试对其他参数进行搜索。

实验总结

违约预测问题面临着所需数据维度多、正负样本严重不均衡等问题，十分具有挑战性。

本实验主要使用pandas和sklearn进行数据预处理、数据分析，seaborn对相关系数矩阵可视化分析，SMOTE方法来过采样，使用网格搜索来寻找最优参数组合。

重点掌握整个建模流程以及常用预处理方法和步骤、过拟合的判断、测试集的预处理。

可以继续通过以下几个方面改进模型：

1. 有些类别型特征可以尝试用独热编码，而不是本实验中的整数索引。
2. 其他分类算法，如朴素贝叶斯， 或者集成学习模型如随机森林
3. 对更多参数进行搜索，且优化搜索范围。