卡通人物

低可信度描述已自动生成

**《大数据分析》**

**期末作业综述**

**2024年1月4日**

银行客户认购产品预测分析

# 数据预处理

## 数据读取

读取数据训练集和测试集到train\_data和test\_data

train\_data = pd.read\_csv('/content/drive/MyDrive/final\_test/train.csv')

test\_data = pd.read\_csv('/content/drive/My Drive/final\_test/test.csv')

## 数据分布观察

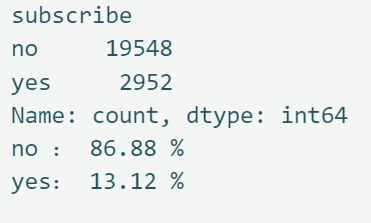
用value\_counts()计算了目标特征'subscribe' 列每个值出现的次数，并打印了打印了’yes’和’no’两者占比。

subscribe\_counts = train\_data['subscribe'].value\_counts()

print(subscribe\_counts)

print('no ：', round(train\_data['subscribe'].value\_counts()['no']/len(train\_data) \* 100,2), '%')

print('yes：', round(train\_data['subscribe'].value\_counts()['yes']/len(train\_data) \* 100,2), '%')



可见yes数量远小于no，正负样本占比不平衡。

## 缺失值处理

利用isnull()查找数据中的缺失值，每一列分别求和，得到各个特征值的缺失值情况。

missing\_values = train\_data.isnull().sum()

print(missing\_values)



由结果得到每一列均无缺失值，无需进行缺失值处理。

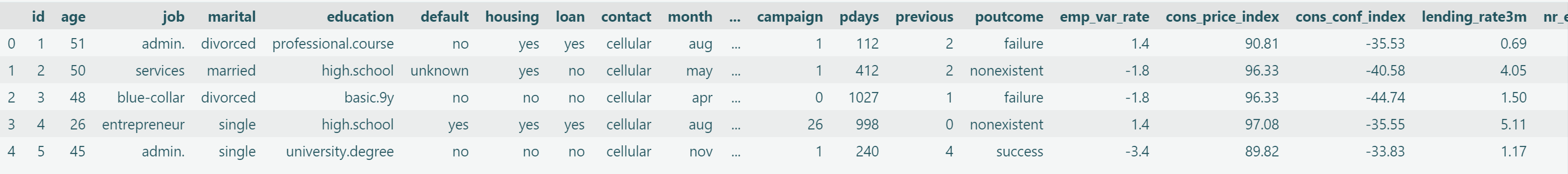
## 编码映射

通过打印数据的前五行，以及打印每一列的数据类型，查看数据情况。

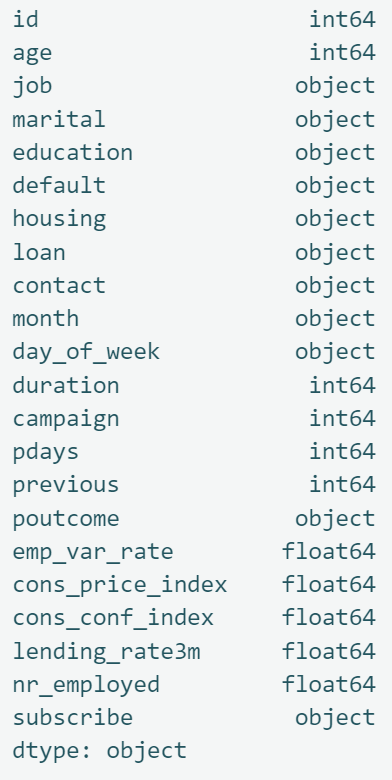
train\_data.head()

data\_types = train\_data.dtypes

print(data\_types)







可以看到有些特征为非数字类型，为了更好让模型理解和捕捉特征之间的相关性以及后续的数据处理，我们将文字特征转为数字特征，我们使用LabelEncoding进行编码。

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

features\_to\_encode = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day\_of\_week', 'poutcome', 'subscribe']

features\_to\_encode2 = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day\_of\_week', 'poutcome']

for feature in features\_to\_encode:

    train\_data[feature] = label\_encoder.fit\_transform(train\_data[feature])

for feature in features\_to\_encode2:

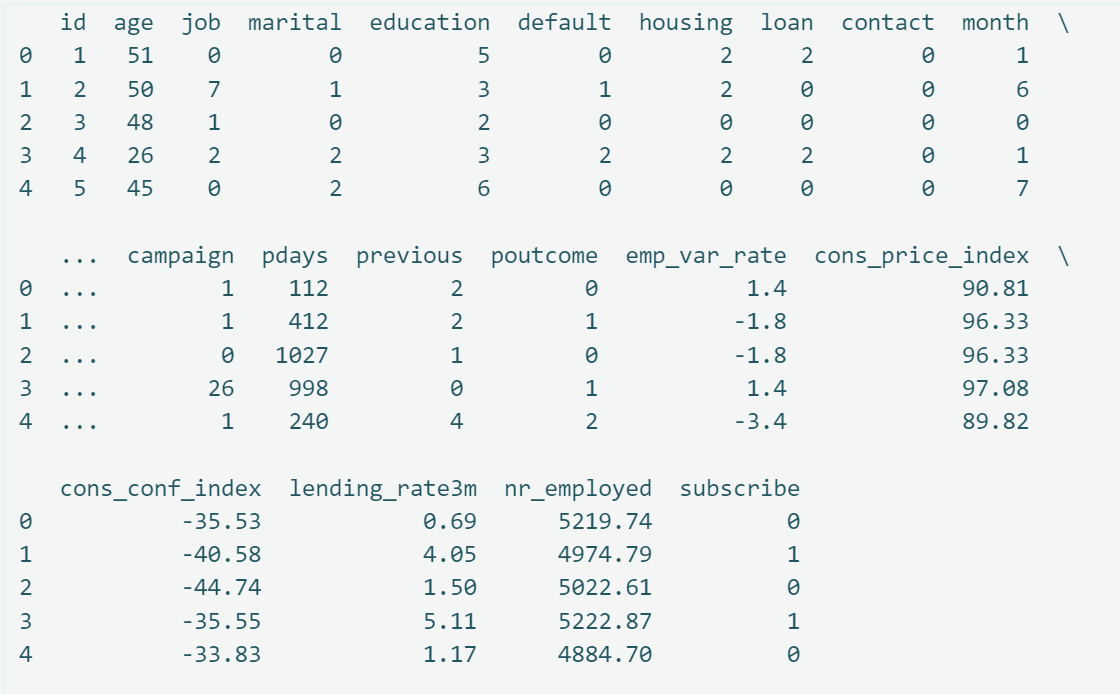
    test\_data[feature] = label\_encoder.fit\_transform(test\_data[feature])

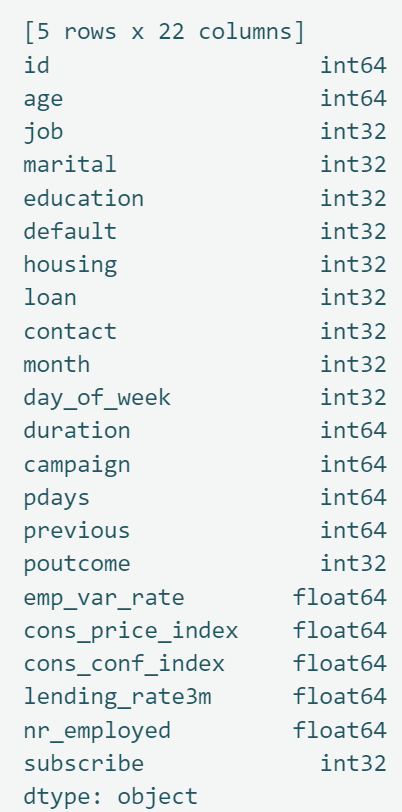
查看编码后的数据

print(train\_data.head())

data\_types = train\_data.dtypes

print(data\_types)





通过LabelEncoding将非数字特征转为数字特征。

## 数据标准化

由于本预测的特征较多，部分模型会对特征尺度敏感，从而影响模型的性能和预测结果，我们对age, emp\_var\_rate, pdays等特征进行min-max标准化，从而去除特征尺度带来的影响，保证特征权值平衡，提高算法的收敛速度，也消除了部分异常值对算法性能的影响。

# 对定量数据进行 min-max标准化

from sklearn.preprocessing import MinMaxScaler

columns\_to\_normalize = ['age','duration','campaign','pdays','previous','emp\_var\_rate','cons\_price\_index','cons\_conf\_index','lending\_rate3m','nr\_employed']

# 创建MinMaxScaler对象

scaler = MinMaxScaler()

# 使用fit\_transform进行标准化

train\_data[columns\_to\_normalize] = scaler.fit\_transform(train\_data[columns\_to\_normalize])

test\_data[columns\_to\_normalize] = scaler.fit\_transform(test\_data[columns\_to\_normalize])

train\_data.head()

## 数据分箱

为了离散化大量连续变从而达到简化模型的复杂，处理连续变量和目标变量非线性关系的目的我们采用等宽分箱将数据均匀分配到具有固定宽度的离散箱中。

分箱操作优点：

1. 可以减少离群值和噪声对整体的影响，从而增强模型的鲁棒性。
2. 通过对变量的离散化，帮助捕捉特征和目标之间的非线性关系，能够提高模型的表达能力。
3. 降低数据的维度，从而达到降低模型复杂度和过拟合风险的目的。
4. 离散后的特征利于模型的迭代，计算结果方便，
5. 特征离散后，模型会更加稳定。

*# 需要进行分箱的数值型特征列表*

numeric\_features = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp\_var\_rate', 'cons\_price\_index', 'cons\_conf\_index', 'lending\_*rate3m'*, 'nr\_employed']

# 定义分箱的数量

num\_bins = 5

# 对每个数值型特征进行等宽分箱

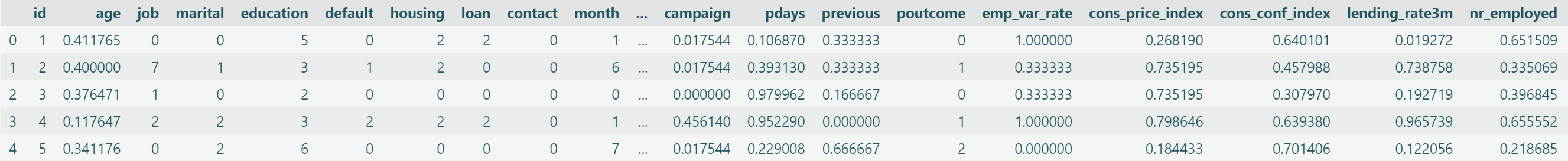
for feature in numeric\_features:

    train\_data[feature + '\_bin'] = pd.cut(train\_data[feature], bins=num\_bins, labels=False)

    test\_data[feature + '\_bin'] = pd.cut(test\_data[feature], bins=num\_bins, labels=False)

# 查看分箱后的数据

train\_data.head()



分箱后的数据

## 特征筛选

通过计算特征之间的相关性矩阵，并用热力图进行可视化，我们可以查看特征之间的相关性，

通过设置阈值将一些与目标特征不相关或是相关性极小的特征进行剔除，从而减少模型的复杂度，防止过拟合噪声，提高计算效率。

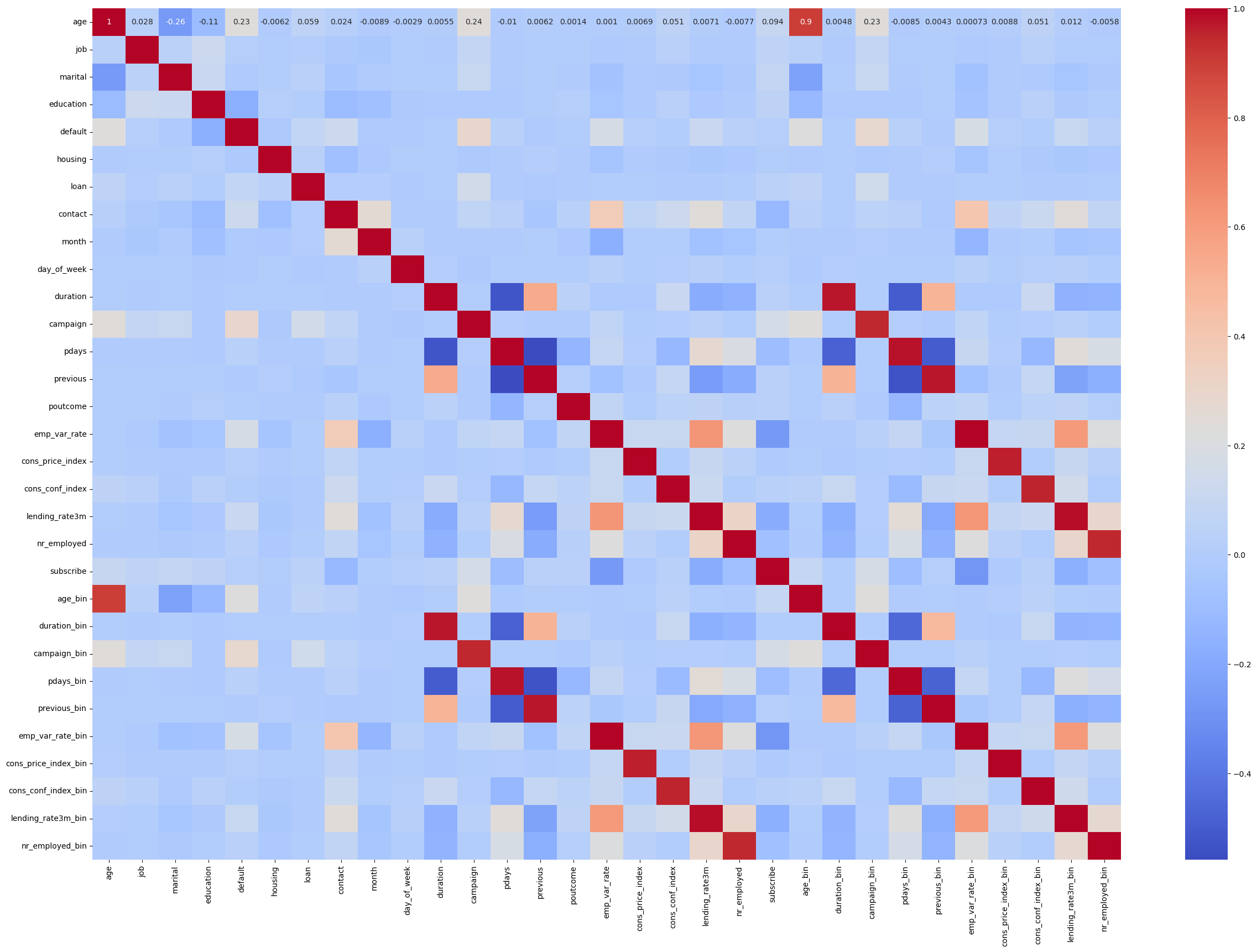
import seaborn as sns

import matplotlib.pyplot as plt

corr\_matrix = train\_data.corr()

plt.figure(figsize=(30, 20))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')



# 设置相关性阈值

correlation\_threshold = 0.01

# 根据相关性阈值筛选特征

selected\_features = []

for feature in corr\_matrix:

    if abs(corr\_matrix[feature]['subscribe']) >= correlation\_threshold:

        selected\_features.append(feature)

# 剔除相关性较低的特征

train\_data\_filtered = train\_data[selected\_features]

selected\_features = [feature for feature in selected\_features if feature != 'subscribe']

test\_data\_filtered = test\_data[selected\_features]

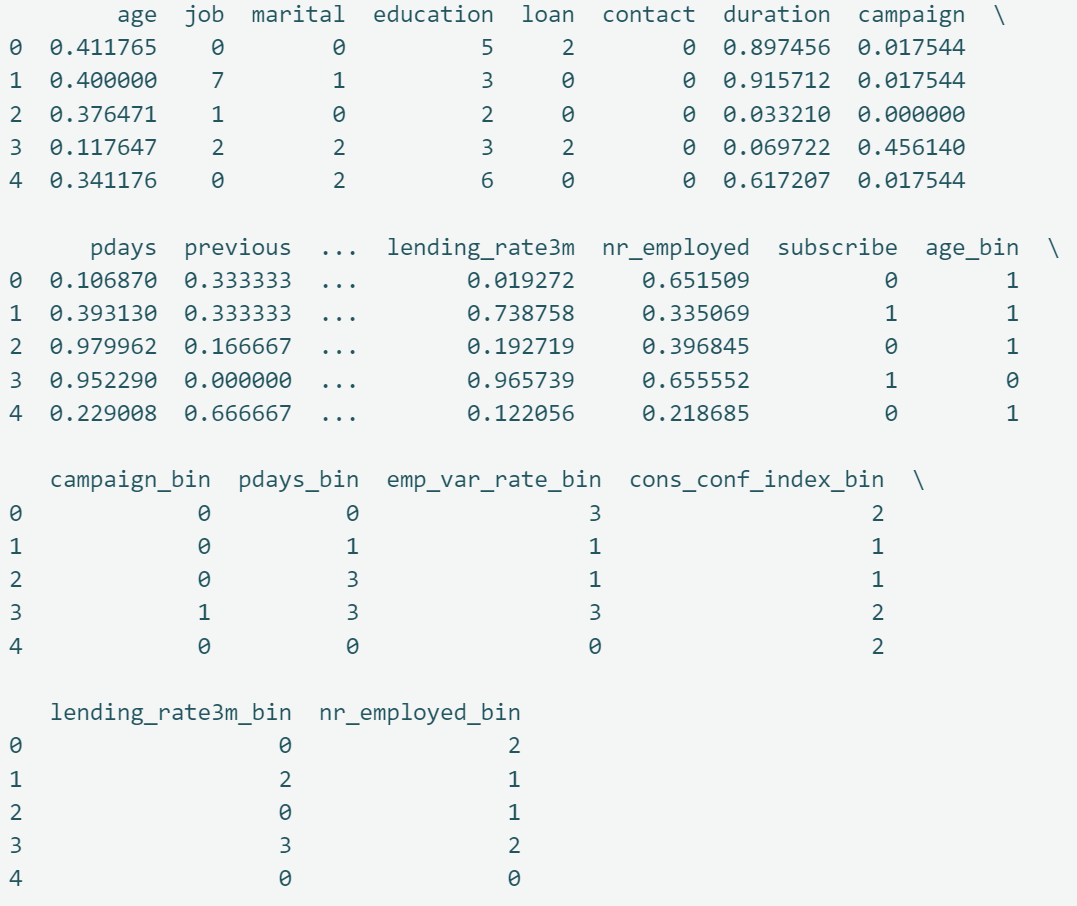
# 查看筛选后的数据

print(selected\_features)

print(train\_data\_filtered.head())

['age', 'job', 'marital', 'education', 'default', 'loan', 'contact', 'day\_of\_week', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'emp\_var\_rate', 'cons\_price\_index', 'cons\_conf\_index', 'lending\_rate3m', 'nr\_employed', 'age\_bin', 'duration\_bin', 'campaign\_bin', 'pdays\_bin', 'previous\_bin', 'emp\_var\_rate\_bin', 'cons\_price\_index\_bin', 'cons\_conf\_index\_bin', 'lending\_rate3m\_bin', 'nr\_employed\_bin']

筛选出如上的特征，进行训练。



剔除后的数据集前五行显示

# 模型选择与训练

# 2.1 划分验证集

from sklearn.model\_selection import train\_test\_split

# 划分特征和目标变量

X = train\_data\_filtered.drop('subscribe', axis=1).values

y = train\_data\_filtered['subscribe'].values

test = test\_data\_filtered.values

# 划分训练集和验证集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=20)

# 将训练集和测试集转换为 DataFrame

X\_train = pd.DataFrame(X\_train)

X\_test = pd.DataFrame(X\_test)

y\_train = pd.DataFrame(y\_train)

y\_test = pd.DataFrame(y\_test)

test = pd.DataFrame(test)

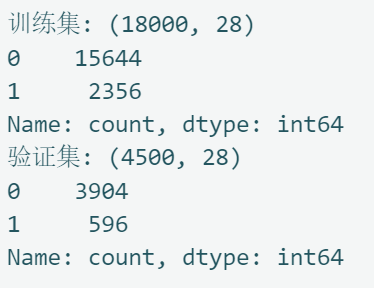
# 查看训练集和验证集的大小

print("训练集:", X\_train.shape)

print(y\_train.value\_counts())

print("验证集:", X\_test.shape)

print(y\_test.value\_counts())



## 2.2模型选择与参数调整

我们使用K折交叉验证对常见的分类模型进行了初步选择和评估，包括适合处理高维度特征的线性模型逻辑回归Logistic Regression，构建出行结构进行分类的决策树Decision Tree，处理高维特征效果好的支持向量机Support Vector Machine，以及基于梯度提升的梯度提升Gradient Boosting、LightGBM和XGBoost。

#模型的初步选择与评估

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score, roc\_auc\_score, accuracy\_score, classification\_report

from sklearn.model\_selection import cross\_val\_score

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import accuracy\_score

classifiers = {

    'LogisticRegression':LogisticRegression(),

    'DecisionTree':DecisionTreeClassifier(),

    'LGBM':LGBMClassifier(),

    'XGB':XGBClassifier(),

    'GradientBoosting':GradientBoostingClassifier(),

}

#交叉验证

def accuracy\_score(X\_train, y\_train):

    for key, classifier in classifiers.items():

        classifier.fit(X\_train, y\_train)

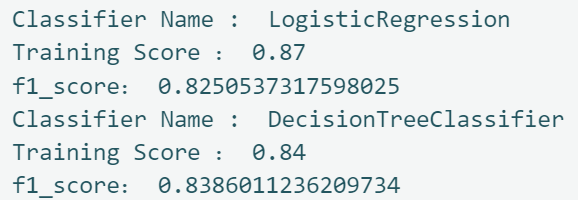
        training\_score = cross\_val\_score(classifier, X\_train, y\_train, cv=5) # 5折交叉验证

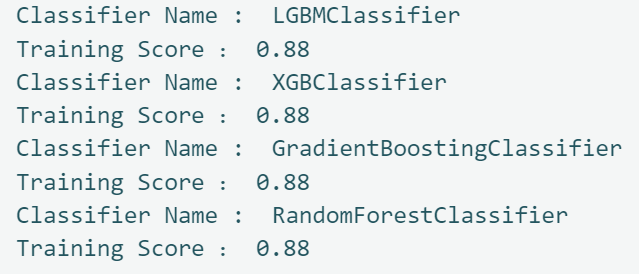
        print("Classifier Name : ", classifier.\_\_class\_\_.\_\_name\_\_)

        print("Training Score ：", round(training\_score.mean(), 2))

#调用函数，进行模型的初步选择与评估

accuracy\_score(X\_train, y\_train)





可见梯度算法的三个模型训练得分情况较好。

接着使用网格搜索获得最优参数，对模型进行优化。

from sklearn.model\_selection import cross\_val\_score

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import accuracy\_score

# 定义模型列表和参数网格

models = [

    {

        'model': LGBMClassifier(),

        'param\_grid': {

         'n\_estimators':[300],

         'learning\_rate':[0.2, 0.02],

         'subsample':[0.6],

         'colsample\_bytree':[0.6],

         'max\_depth':[9, 11, 13],

         'is\_unbalance':[False],

         'num\_boost\_round':[5000],

         'metric':['binary\_logloss'],

         'feature\_fraction':[0.6, 0.75, 0.9],

         'bagging\_fraction':[0.6, 0.75, 0.9],

         'lambda\_l1':[0, 0.1, 0.4, 0.5],

         'lambda\_l2':[0, 10, 15, 35],

         'cat\_smooth':[1, 10, 15, 20]

        }

    },

    {

        'model': XGBClassifier(),

        'param\_grid': {

            'n\_estimators':[50, 100, 150, 200],

            'max\_depth':[5, 6, 7, 8, 9, 10],

            'learning\_rate':[0.05, 0.1, 0.15, 0.2],

            'subsample': [0.6, 0.7, 0.8, 0.9, 1],

            'colsample\_bytree':[0.6, 0.7, 0.8, 0.9, 1]

        }

    },

    {

        'model': GradientBoostingClassifier(),

        'param\_grid': {

            'n\_estimators': [50, 100, 200],

            'learning\_rate': [0.01, 0.05,0.1,],

            'max\_depth': [6,8,10],

            'min\_samples\_split':[2,3,4,5],

            'min\_samples\_leaf':[1,2]

        }

}

]

# 对每个模型进行参数调整和性能评估

for model\_info in models:

    model = model\_info['model']

    param\_grid = model\_info['param\_grid']

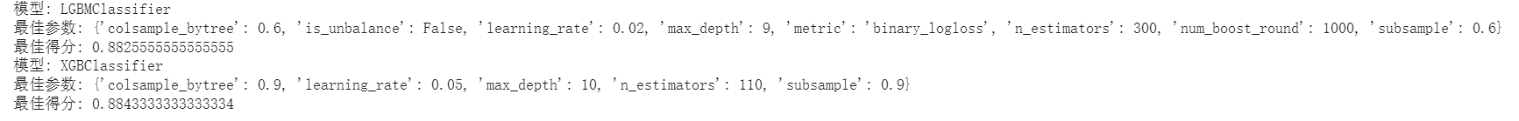
    grid\_search = GridSearchCV(model, param\_grid, cv=5, error\_score='raise')

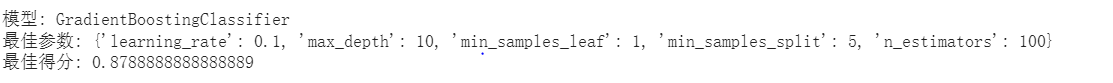
    grid\_search.fit(X\_train, y\_train)

    print("模型:", model.\_\_class\_\_.\_\_name\_\_)

    print("最佳参数:", grid\_search.best\_params\_)

    print("最佳得分:", grid\_search.best\_score\_)





## 2.4 模型训练

对模型进行训练，这里将交叉验证时得分最好的三个模型进行对比。

from xgboost import XGBClassifier

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score, roc\_auc\_score, accuracy\_score, classification\_report

from sklearn.metrics import precision\_recall\_fscore\_support

# model = LGBMClassifier(learning\_rate=0.02, max\_depth=9, n\_estimators=300, subsample=0.6, colsample\_bytree=0.6,boost\_round = 1000,  is\_unbalance = False, metrics = 'binary\_logloss')

model = XGBClassifier(learning\_rate=0.05, max\_depth=10, n\_estimators=110, subsample=0.9, colsample\_bytree=0.9)

#model = GradientBoostingClassifier(learning\_rate = 0.1,max\_depth = 10, min\_samples\_leaf = 1, min\_samples\_split = 5, n\_estimators = 100)

model.fit(X\_train, y\_train)

# 预测验证集

y\_pred = model.predict(X\_test)

# 计算准确率和F1分数

accuracy = accuracy\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred, average='weighted')

micro\_f1 = f1\_score(y\_test, y\_pred, average='micro')

recall = recall\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

print("Precision:", precision)

print("Recall:", recall)

print("Micro-F1:", micro\_f1)

print("accuracy:", accuracy)

print("f1\_score:", f1)

# 输出分类报告

class\_report = classification\_report(y\_test, y\_pred)

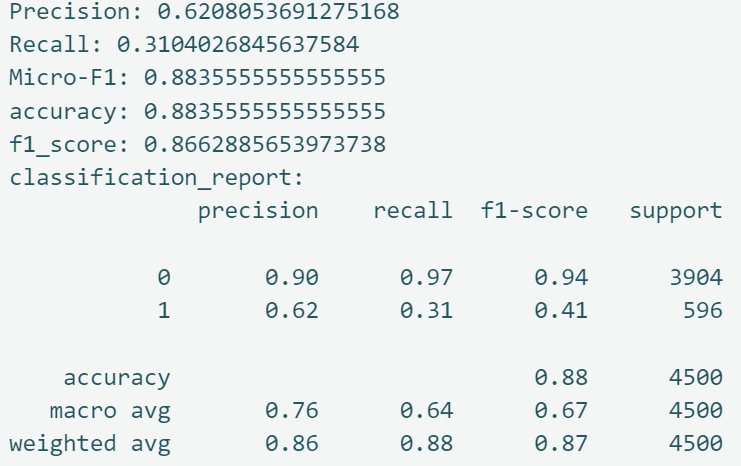
print("classification\_report:")

print(class\_report)

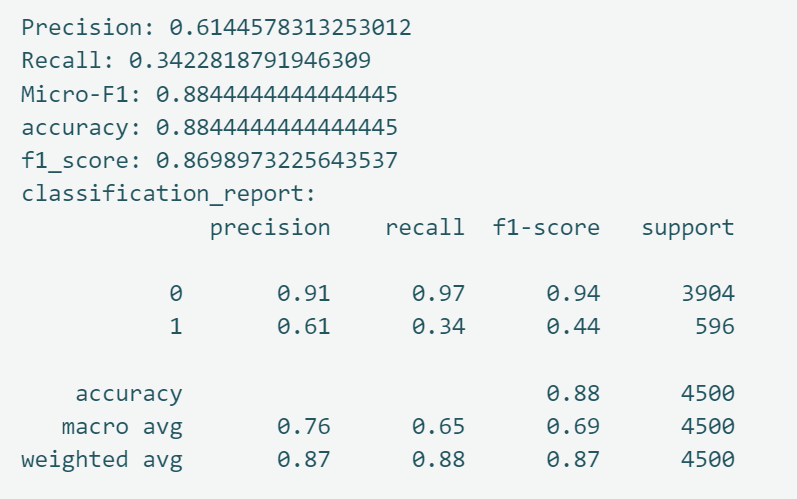
## 2.5 模型评价

## 2.5.1对比指标

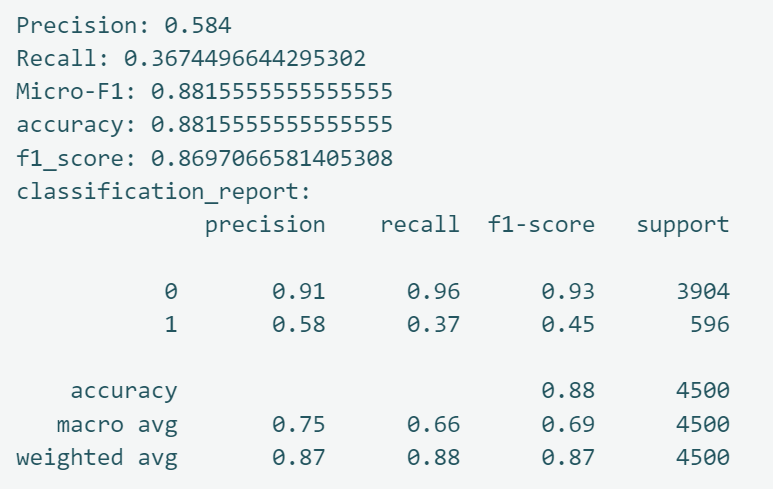
LGBM



XGB



GradientBoostingClassifier



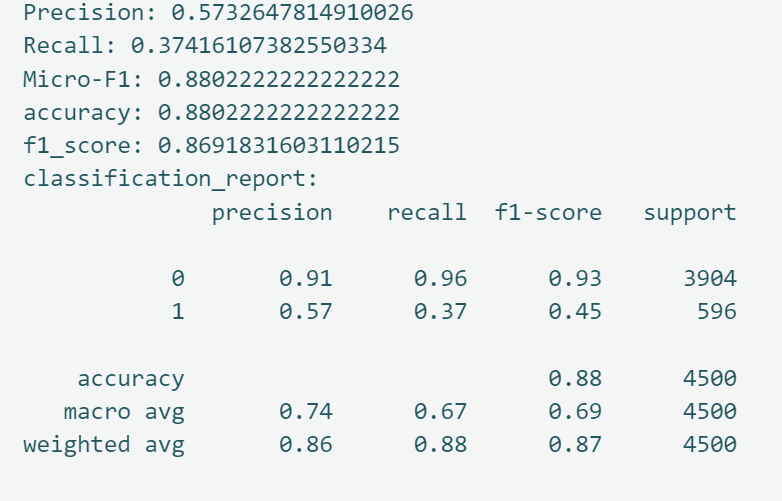
## 2.5.2 模型效果

由对比结果，选择XGB，XGB的accuracy和f1值均为最高，虽然XGB模型的准确度没有LGBM的高，但是综合考虑模型的准确度和辨识能力来看，XGB的f1值最高，其模型的预测结果最好，说明它在捕捉真实正例和减少误报之间找到了一个较好的平衡，并且在识别正例和排除负例方面具有较高的能力。

在数据集中正负样本不平衡，no>>yes的情况下，XGB模型通过加权损失函数，给予少数类别样本更高的权重，使得模型在预测时对少数样本有更高的召回率和准确度，也使得整体的预测效果最好。

这使得 XGBoost 模型在解决分类问题时更为优秀。

我们再将其未调参结果进行对比，明显发现调参后的模型预测更优。



发现调参后模型达到最佳，通过调节学习率、树的最大深度、叶子节点的最小样本数等参数，可以更好让模型适应正负样本不均衡的数据分布，提高模型在不平衡数据中的性能。

## 2.6 预测测试集

test\_pred = model.predict(test)

print("预测结果:", test\_pred)

result = pd.read\_csv(r"C:\Users\qqqq\Desktop\数据\after\_submission.csv")

subscribe\_map ={1: 'yes', 0: 'no'}

result['subscribe'] = [subscribe\_map[x] for x in test\_pred]

result.to\_csv(r"C:\Users\qqqq\Desktop\数据\after\_submission.csv", index=False)

阿里天池在线提交成绩：

