浙江工商大學

课 程 设 计

题目：综合三种不同分类算法的个人信用评估模型比较研究

课程名称：探索性数据（课程设计）

学　　院：统计与数学学院

专　　业：应用统计

班　　级：应用统计二班

学　　号：22020040149

学生姓名：王振宽

二〇二三 年 六 月

基于三种不同分类算法的个人贷款模型比较研究

摘要

随着金融行业的不断发展和个人贷款需求的增加，准确预测个人是否需要贷款的模型变得愈发重要。本文的目标是对个人贷款数据集进行探索，研究其中的特征与标签之间的关系，并尝试构建一个准确率高的预测模型。

首先，我们进行了探索性数据分析。这一阶段包括对数据集中各个变量的分布进行分析，以了解它们的特点和范围。我们还进行了缺失值分析，确定是否有任何缺失数据，并考虑采取适当的处理方式。另外，我们还对异常值进行了处理，以确保数据的准确性和可靠性。

接下来，我们使用了一系列方法来分析变量之间的关系。其中包括相关性分析，帮助我们确定变量之间的线性关系强度，并找到与贷款需求相关的主要特征。我们还使用了箱线图等可视化工具，探索了不同变量与贷款需求之间的关联。例如，我们可能发现收入与贷款需求呈现正相关关系，即收入较高的个人更有可能需要贷款。同时，我们还可能观察到教育水平与贷款需求呈现负相关关系，即教育水平较高的个人更少需要贷款。

在特征与标签之间的关系分析之后，我们着手构建预测模型。为了实现这一目标，我们选择了三个常见的分类模型：逻辑回归、朴素贝叶斯和梯度提升决策树（GBDT）。为了评估模型的性能，我们将数据集划分为训练集和测试集，其中 80%的数据用于训练模型，20%用于评估模型在未见过数据上的表现。

通过比较三个模型在测试集上的表现，我们发现 GBDT 模型表现最优，其准确率达到了惊人的 99%。这意味着我们的模型能够高度准确地预测个人是否需要贷款。

总结起来，本文通过深入的探索性数据分析为我们提供了对个人贷款数据集的全面了解。通过构建和比较多个分类模型，我们证明了 GBDT 模型在准确率方面表现出色。这些研究结果为金融机构和其他相关领域提供了有价值的参考，

可以帮助他们更好地预测个人是否需要贷款。此外，本研究也为个人贷款领域的进一步研究奠定了基础。

关键词：个人贷款，探索性数据分析，逻辑回归，朴素贝叶斯，GBDT

目录

一、 问题背景

这个案例是关于一家银行（Thera Bank），其管理层希望探索将其负债客户转变为个人贷款客户（同时保留他们作为储户）的方法。 该银行去年为负债客户开展的一项活动显示，成功的转化率超过 9%。 这鼓励零售营销部门设计具有更好目标营销的活动，以用最少的预算提高成功率。

个人贷款作为金融行业中的一种重要业务形式，对于金融机构和借款人都具有重要的经济和社会意义。金融机构通过个人贷款活动可以提供融资服务，满足客户的资金需求，推动经济增长；而借款人则可以通过个人贷款满足个人消费、投资和创业的需要。因此，如何准确地预测客户是否会响应个人贷款活动，对金融机构和借款人都具有重要的决策参考价值。

过去的研究已经证明了个人贷款响应预测在金融业务决策中的重要性。例如，通过预测客户的贷款响应行为，金融机构可以更好地了解客户的借款意愿和偿还能力，从而更加精确地定价和定量化风险，有效管理贷款风险。此外，准确的贷款响应预测还可以帮助金融机构优化市场营销策略，提高市场活动的效果和投资回报率。而对于借款人来说，了解自己是否会响应个人贷款活动，可以帮助其做出更明智的借款决策，合理规划个人财务和债务。

然而，个人贷款响应预测面临着一系列的挑战和问题。首先，个人贷款响应行为受到多个因素的影响，包括客户的个人特征（如年龄、性别、职业等）、金融历史（如信用分数、收入、债务情况等）、市场环境（如经济状况、利率水平等）等，这使得贷款响应行为呈现出复杂的非线性关系。其次，个人贷款响应预测需要处理大量的多源异构数据，如客户的个人信息、金融历史、市场营销活动数据等，这对数据处理和特征工程提出了较高的要求。此外，传统的统计方法和传统的机器学习算法在面对这些复杂性和异构性时可能存在一定的局限性，需要寻找更加精确和有效的预测方法。

因此，基于以上问题和挑战，研究如何预测客户是否会响应个人贷款活动成为了金融数据科学和金融风险管理的一个热门研究方向。随着大数据和人工智能技术的快速发展，越来越多的研究者开始采用先进的预测模型和算法，以提高个人贷款响应预测的准确性和实用性。

二、 数据集介绍

我们的数据及来自 Kaggle 数据竞赛 Personal Loan Modeling。

数据集文件包含 5000 个客户的数据。这些数据包括客户人口统计信息（年龄、收入等）、客户与银行的关系（抵押、证券账户等），以及客户对上次个人贷款活动（个人贷款）的反应。 在这 5000 名客户中，只有 480 名 (= 9.6%) 接受了之前活动中提供给他们的个人贷款。

数据集混合了数字和分类属性，但所有分类数据都用数字表示。 此外，一些预测变量严重倾斜（长尾），使得数据预处理成为数据的一个有趣但不太具有挑战性的方面。

表 1：数据集特征及其含义

特征名	含义
ID	客户 ID
Age	客户的年龄（整年）
Experience	专业经验年数
Income	客户的年收入
ZIP Code	家庭地址邮政编码
Family	客户家庭规模
CCAvg	平均每月信用卡支出
Education	教育程度
Mortgage	房屋抵押贷款的价值
Personal Loan	客户是否接受了上次活动中提供的个人贷款
Securities Account	客户在银行是否有证券账号
CD Account	客户是否在银行有存款证明 (CD) 帐户
Online	客户是否使用网上银行设施
CreditCard	客户是否使用本行发行的信用卡

我们得到数据集的基本信息如下表所示：

表 2：数据集的基本信息

特征名	缺失值情况	数据类型
ID	5000 non-null	int64
Age	5000 non-null	int64
Experience	5000 non-null	int64
Income	5000 non-null	int64
ZIP Code	5000 non-null	int64
Family	5000 non-null	int64
CCAvg	5000 non-null	float64
Education	5000 non-null	int64
Mortgage	5000 non-null	int64
Personal Loan	5000 non-null	int64
Securities Account	5000 non-null	int64
CD Account	5000 non-null	int64
Online	5000 non-null	int64
CreditCard	5000 non-null	int64

可以看到，我们的数据集没有空值，且数据类型都是连续性变量。

三、 探索性数据分析

探索性数据分析（Exploratory Data Analysis，简称 EDA）是一种数据分析方法，主要是通过绘图、统计量计算、数据可视化等方式，对数据集进行初步的探索和分析。目的是发现数据集中的规律、趋势、异常点和缺陷等特征，为后续的数据挖掘和建模提供参考。

通过 EDA，可以帮助我们更好地了解数据集的特征和分布情况，为后续的数据分析和建模提供更好的基础和参考。

3.1 描述性统计分析

从上面的数据集介绍我们不难看出，我们的数据集数据类型全是连续性变量，我们首先进行描述性统计分析，对数据集的各个变量进行描述性统计，如平均值、中位数、方差、标准差、最大值、最小值等，了解数据集的分布和基本特征。

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000	5000.000000
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937938	1.881000	56.498800	0.096000	0.104400	0.06040	0.596800
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747659	0.839869	101.713802	0.294621	0.305809	0.23825	0.490589
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.00000	0.000000
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	0.00000	0.000000
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	0.00000	1.000000
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	0.00000	1.000000
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	1.00000	1.000000

图 1：描述性统计分析的结果

3.2 每一列的唯一值数量

用来初步了解数据集每一列的特征，例如某些列的唯一值数量很少，可能代表着这些列是类别型数据，而不适合用于数值分析。另外，如果某些列的唯一值数量很多，可能代表着这些列是连续型数据，可能需要进一步探索这些数据的分布情况。绘图使用了条形图（barh）来表示每一列的数量，颜色为紫色。

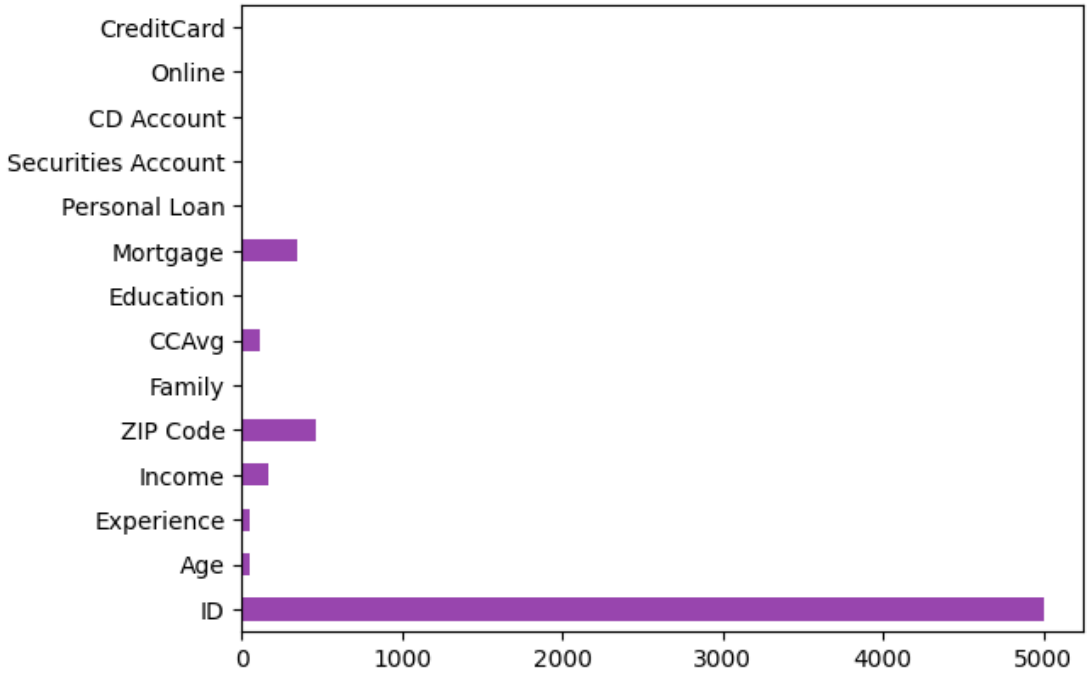


图 2：唯一值数量

我们建立表格如下所示：

表 3：每个特征唯一值的数量

特征名	唯一值数量
ID	5000
Age	45
Experience	47

Income	162
ZIP Code	467
Family	4
CCAvg	108
Education	3
Mortgage	347
Personal Loan	2
Securities Account	2
CD Account	2
Online	2
CreditCard	2

对每列不同取值数量的分析：

ID 列有 5000 个不同的取值，每个客户的 ID 是唯一的，可以作为索引列。Age 列有 45 个不同的取值，表示数据集中客户的年龄，该列数据可以分为不同的年龄组进行分析。Experience 列有 47 个不同的取值，表示客户的工作经验，可能存在一些异常值，需要进一步检查。Income 列有 162 个不同的取值，表示客户的收入，可以分为不同的收入组进行分析。ZIP Code 列有 467 个不同的取值，表示客户的邮政编码，可以用于地理位置分析。Family 列有 4 个不同的取值，表示客户的家庭成员数量，可以分为不同的家庭大小组进行分析。CCAvg 列有 108 个不同的取值，表示客户每个月的信用卡平均消费，可以分为不同的消费水平组进行分析。Education 列有 3 个不同的取值，表示客户的教育程度，可以分为不同的教育程度组进行分析。Mortgage 列有 347 个不同的取值，表示客户的按揭金额，可以分为有按揭和无按揭两组进行分析。Personal Loan 列有 2 个不同的取值，表示客户是否接受个人贷款，可以分为有个人贷款和无个人贷款两组进行分析。Securities Account、CD Account、Online 和 CreditCard 列均有 2 个不同的取值，表示客户是否持有证券账户、存款证明、网上银行和信用卡，可以分别进行分析。

我将特征分为数值和分类，在得到每个特征的 max 和 min 后，我发现 Experience 栏有一些错误的负值。使用不同的颜色对数值型特征和分类特征进行可视化展示。接下来遍历数值型特征和分类特征，并得到每个特征的最大值和最小值。最后，对于每个特征，得到该特征的唯一值数量，并判断如果唯一值的数量等于数据集的行数，则将该特征从数据集中删除。

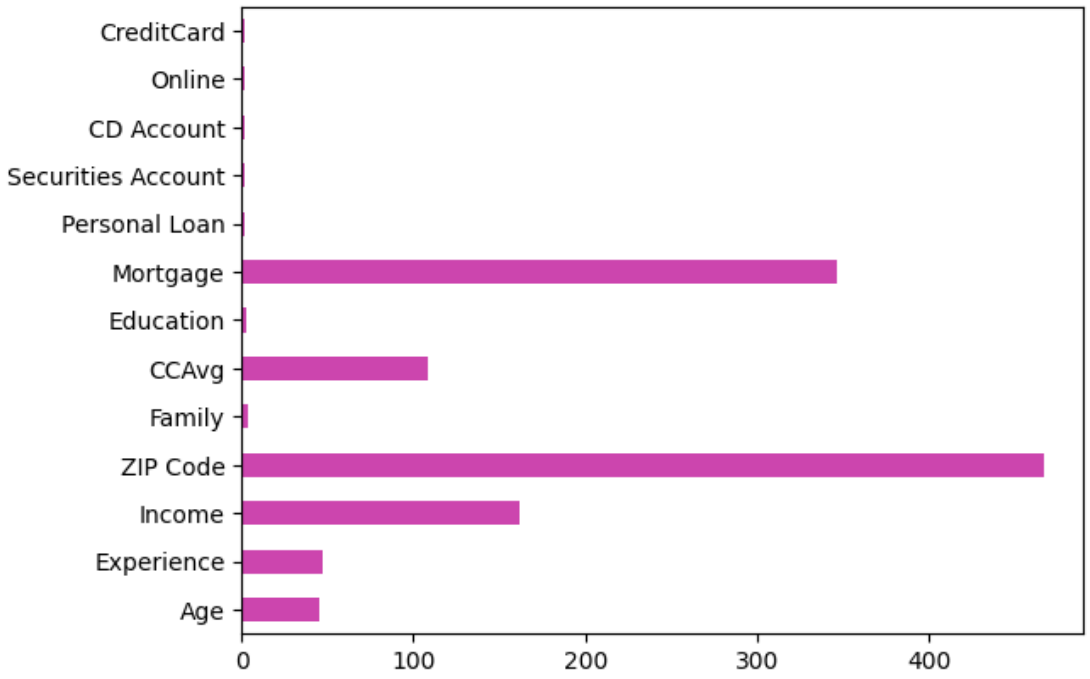


图 3：处理后的唯一值数量

3.3 数据的预处理

我发现 Experience 栏有一些错误的负值，将其取绝对值处理。另一方面，我必须在 CCAvg 列中进行更改。 CCAvg：每个人每月花费我们可以使用正则表达式概念来更改此列，最后我必须将每月金额转换为年度金额以便我可以将此列用于其余数字列

经过处理后我们的数据集类型分别为：

表 4：处理后的数据集数据类型

特征名	数据类型
ID	int64
Age	int64
Experience	int64
Income	int64

ZIP Code	int64
Family	int64
CCAvg	int64
Education	int64
Mortgage	int64
Personal Loan	int64
Securities Account	int64
CD Account	int64
Online	int64
CreditCard	int64

将每个邮政编码转换为虚拟变量是我们可以实现的解决方案之一。此外，邮政编码是分层的。如果我们采用前两位或三位数字，并根据这些数字进行二值化，将获得一定数量的区域信息，这比单个邮政编码获得更多数据。

92717、92634 和 96651 这三个邮政编码，我们必须删除，也许它们提供了错误的地址或者他们是在美国以外或任何其他原因。 无论如何，不应该给这些人贷款。

3.4 地区的分布情况

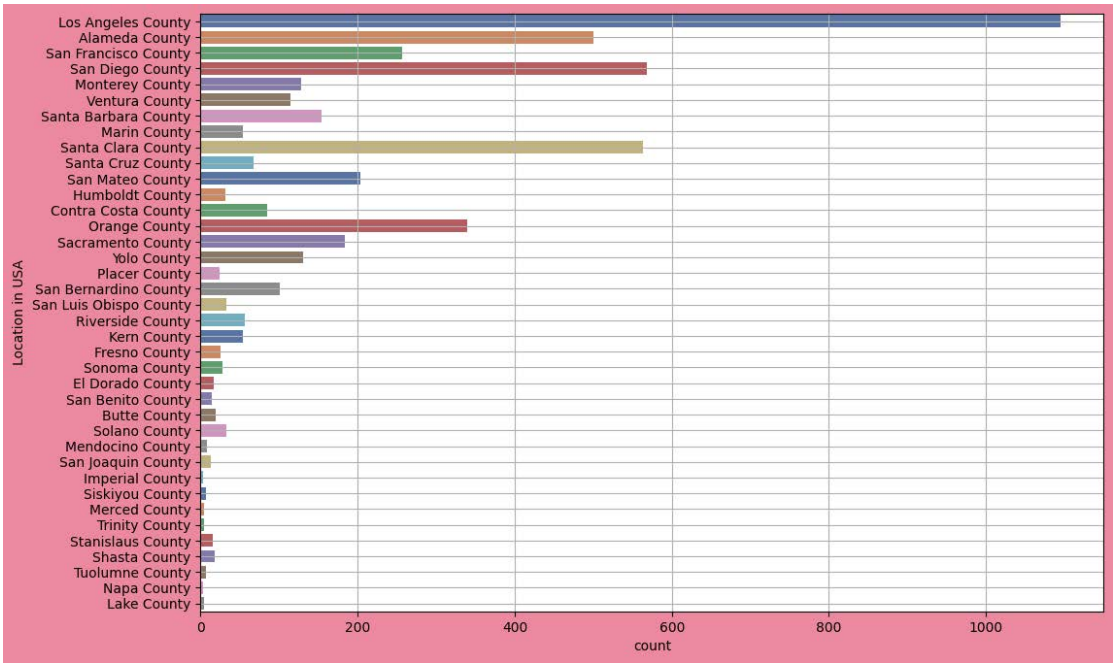


图 4：地区的分布情况

根据上图，出现频率最高的是洛杉矶市，Lake、Napa 等城市出现频率最低。

3.5 可视化

我们得到分类特征的频率图如下所示：

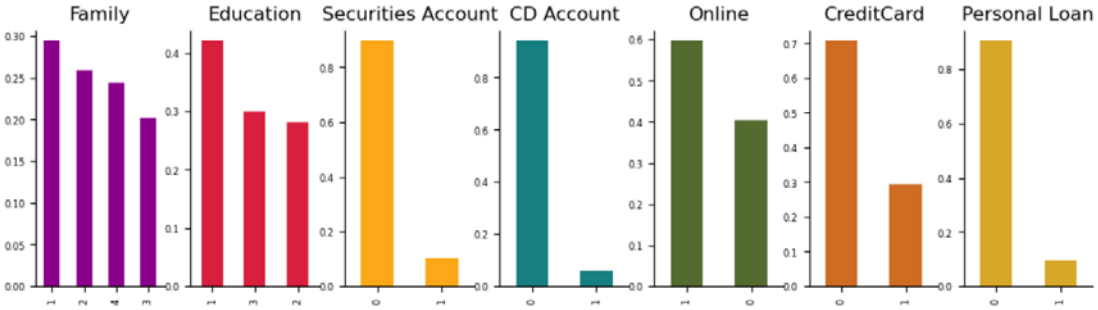
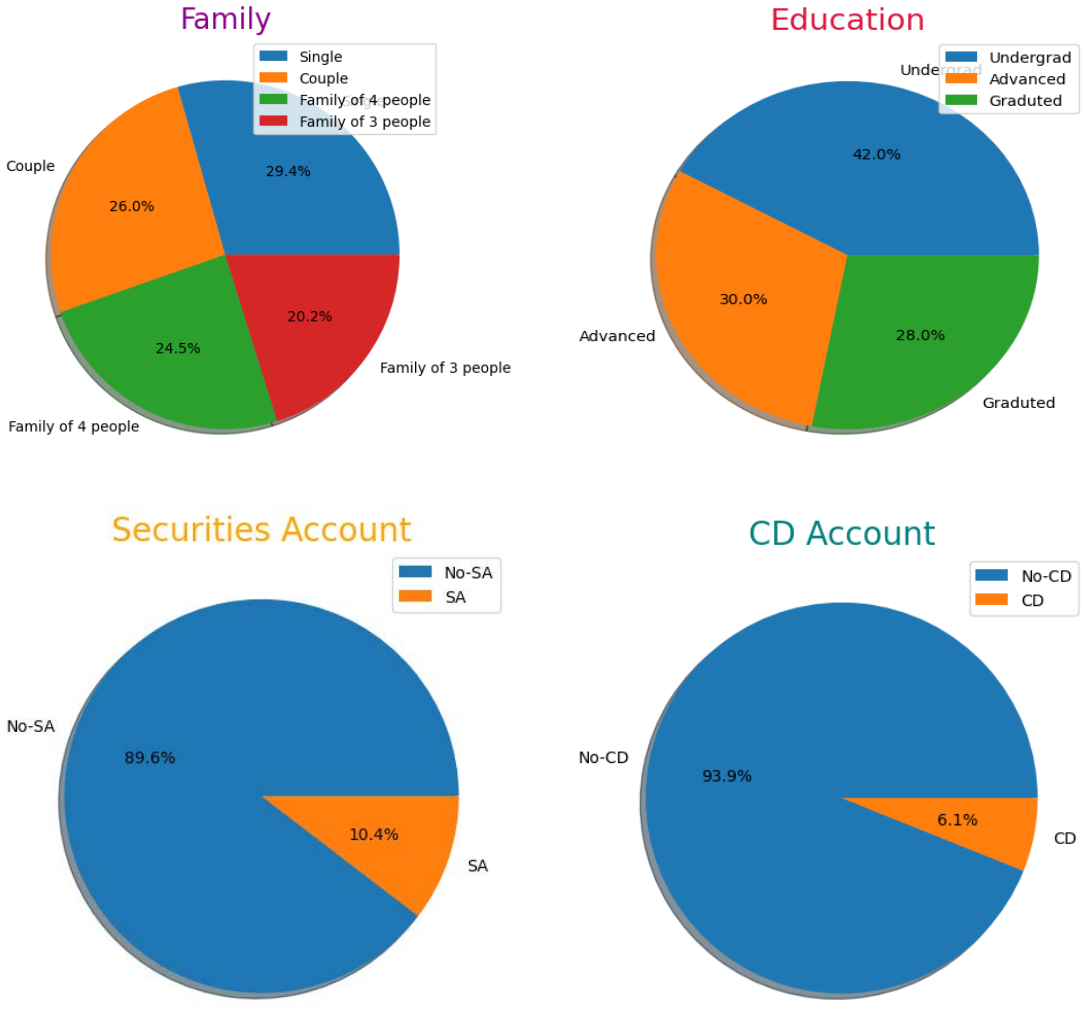


图 5：分类特征的频率图

绘制相关的饼图如下所示：



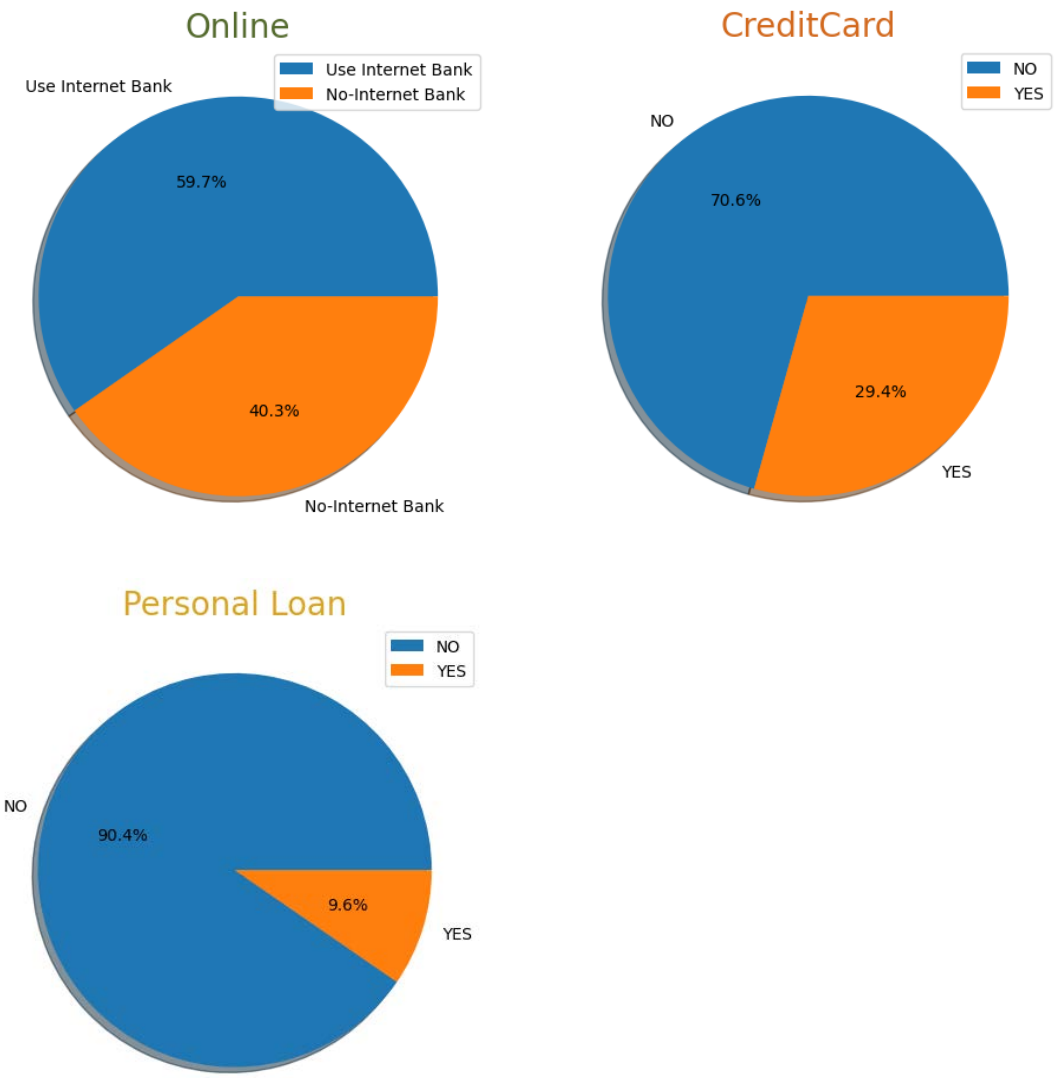


图 6：分类特征的饼图

根据上面的图表，我们得出：

1. 单身人士比其他人多；

2. 本科生和那些没有安全帐户的人与其他群体相比更频繁；

3. 使用互联网服务较少的人和没有信用卡的人比其他人多；

4. 最重要的是，没有贷款的人数远远多于贷款的人数。

所以我们有不平衡的数据集。此外，我们在 CD 账户和证券账户中的分配不平衡。

接下来看一下数值特征饼图：

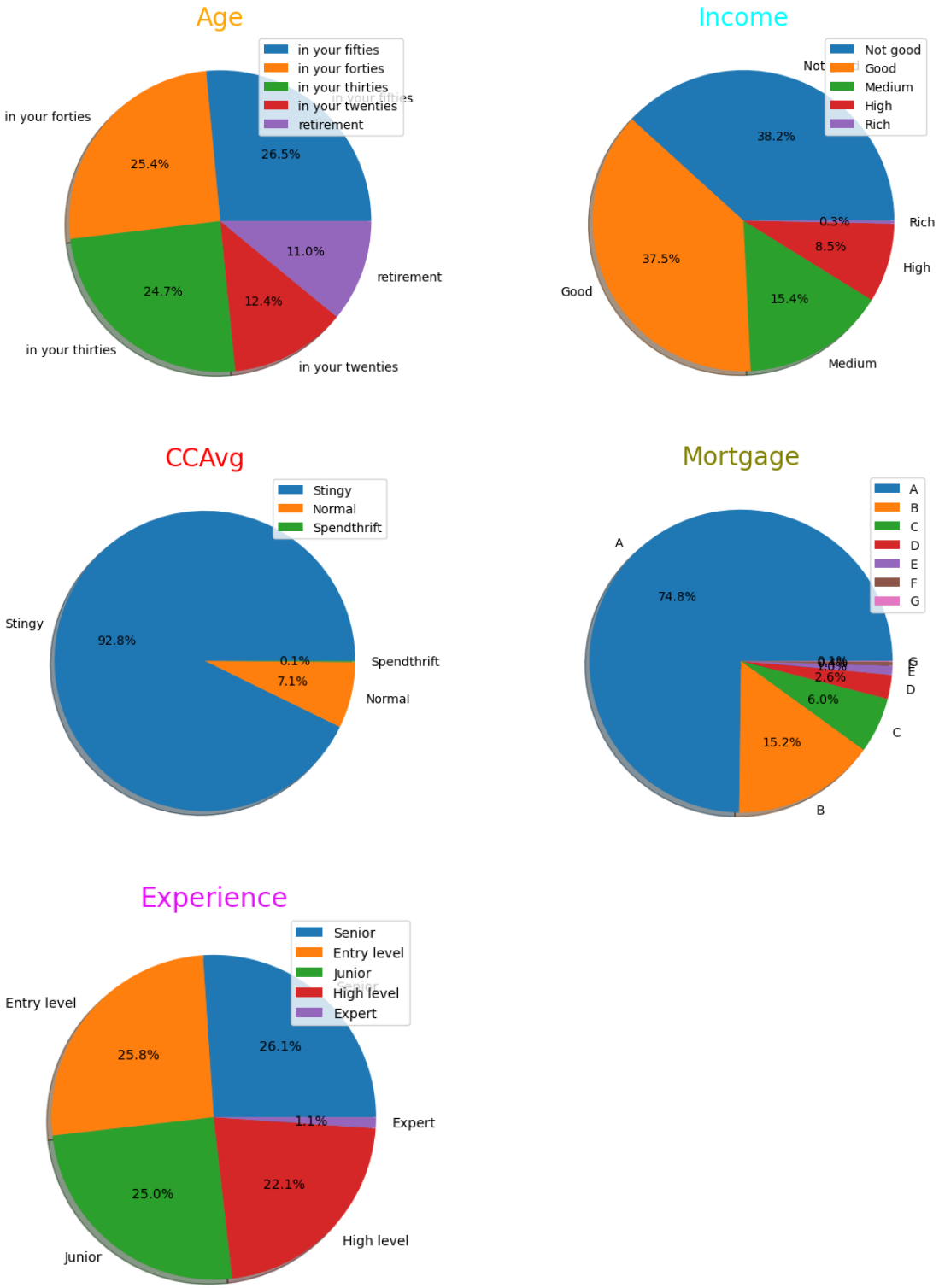


图 7：数值特征饼图

40 到 50 岁之间，收入很低的人，小气的人，抵押贷款低于 100 美元的人和工作经验不到 10 年的人更多。

接下来检查其他列对收入的影响：

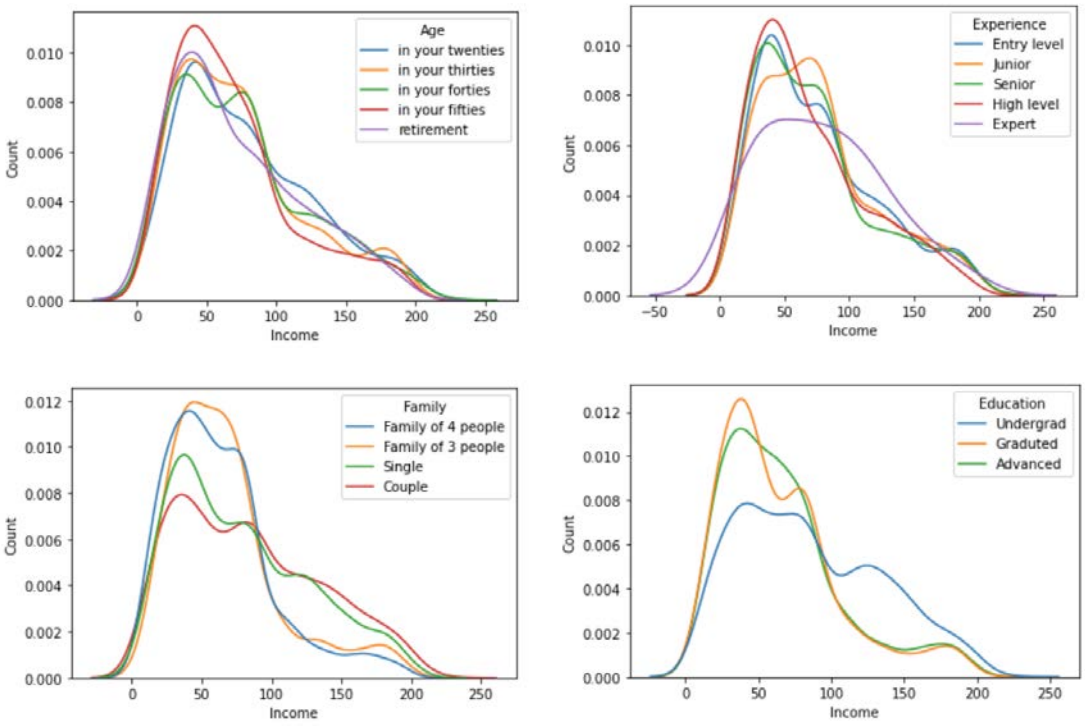


图 8：其他列对收入的影响

收入超过 150 美元的 20 至 30 岁人群数量，高于其他年龄组，而有趣的一点是，收入在 100 美元以上的夫妻或二人家庭的数量比其他类型的家庭要多，另一个有趣的一点是，收入超过 100 美元的本科生比受过教育的人多。

3.6 检查缺失值

对于每个数值型特征，计算出特征的四分位数（Q1, Q3）、四分位数间距（IQR）、上下边缘值（maximum, minimum），然后检查每个特征的值是否小于最小值或大于最大值。如果是，则视为异常值。

我们对齐进行可视化，如下图所示：

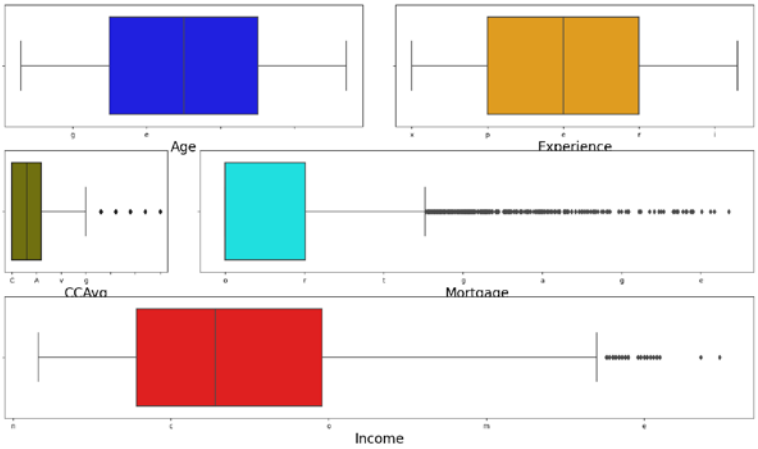


图 9：异常值的检验

绘制此图表的目的是为了更好地了解具有正态分布的年龄和经验特征的分布，我们还可以看到数据集中没有噪声或错误数据，也没有影响模型结果的异常值。

3.7 数据集中多个变量之间的散点图以及变量的分布情况

我们根据 Personal Loan 列进行分组，即根据个人贷款是否被批准对数据进行分组，并用不同颜色标记。这使得我们可以轻松地比较不同分组之间的关系和分布情况。

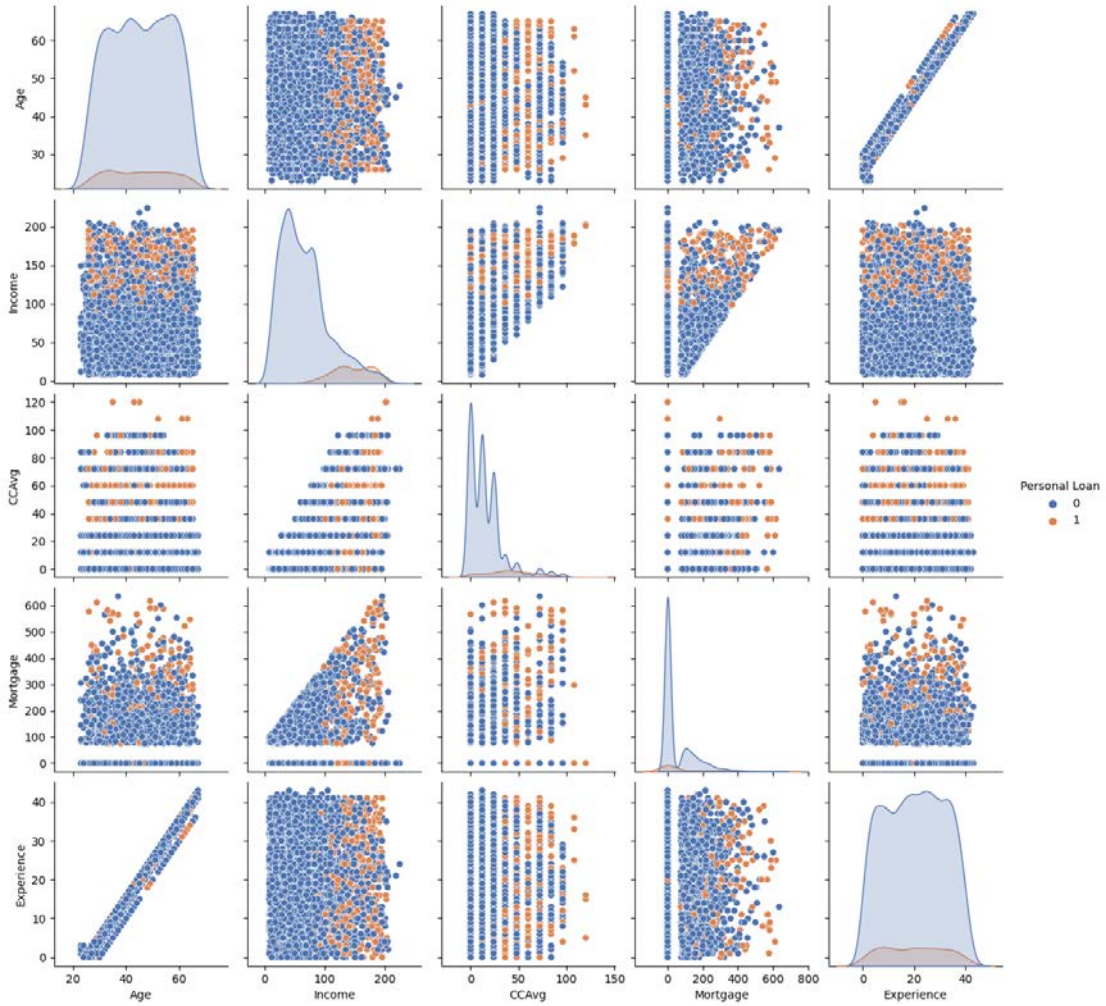


图 10：数据集中多个变量之间的散点图以及变量的分布情况

我们从上图中得到的信息如下：

1. 年龄和经验之间的相关性如此之高。（这对于模型构建可能会有问题）

2. 收入与 Mortgage 和 CCAvg 几乎有很高的相关性；

3. Mortgage 等某些功能的值为零。 正如上面提到的，这些值没有错；

4. Be IMBALANCED 可以在每个功能图中清楚地看到它本身。

3.8 相关性分析

绘制数据集中各列的相关系数矩阵的热力图如下：

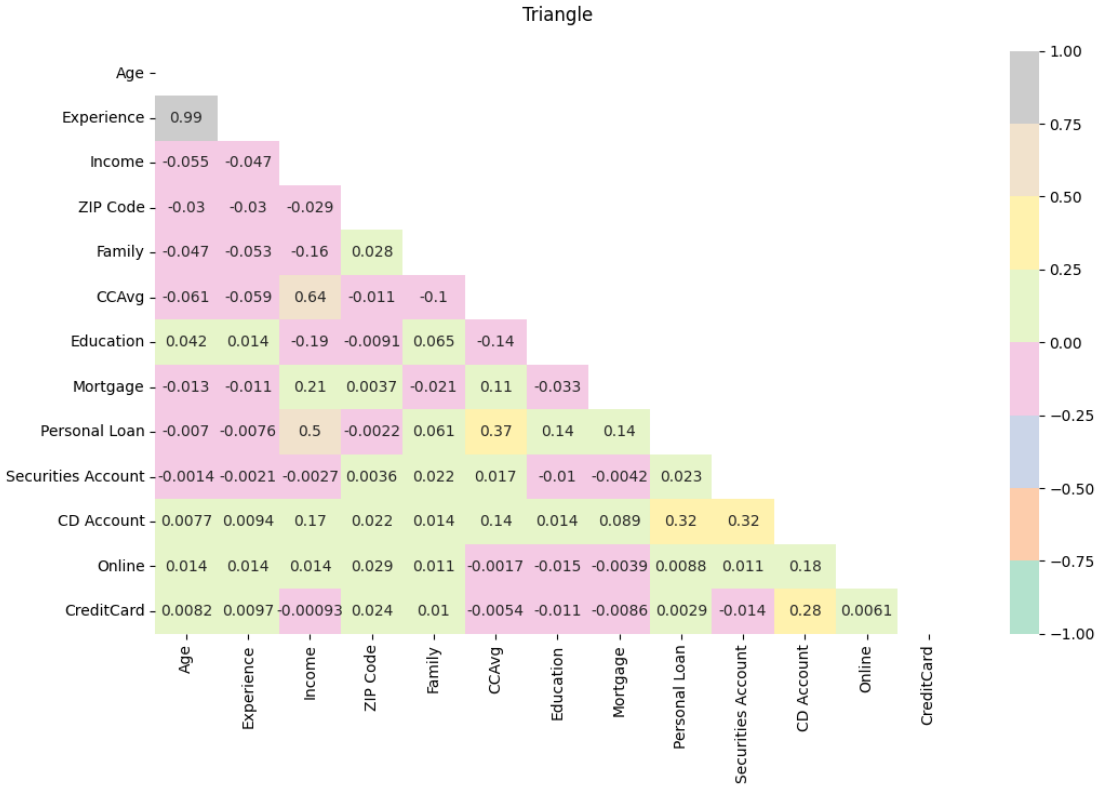


图 11：相关性热力图

在这里，我们可以看到每个特征与其他特征以及自身的相关性。 我们上面提到的一个重点是年龄和经验之间的相关性非常高，这个问题对于构建模型来说可能很糟糕。因此，我们要删除这两列之一，因为 Experience 的特征在 Age 中，所以没有必要删除它。

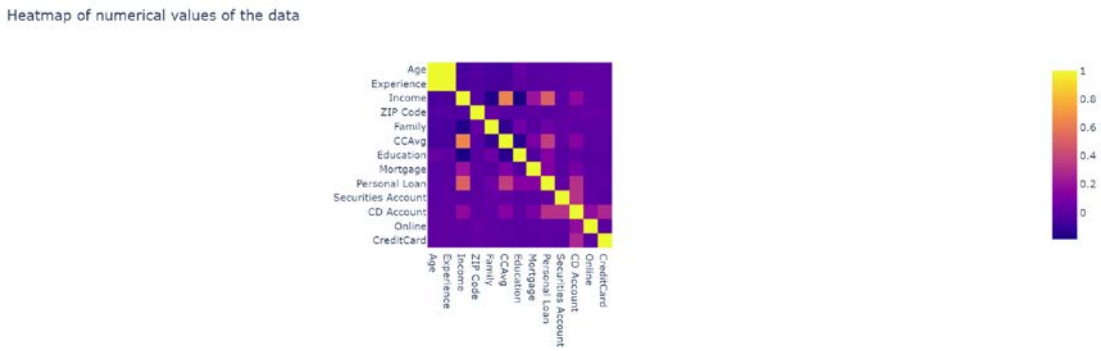


图 12：相关性热力图 2

从这两个图表中，我们可以看到 Income 、 CCAvg 和 CD Account 的特征与 Personal Loan 最相关。在下一级，教育和抵押贷款更相关。

四、 模型的建立与求解

4. 1 LogisticRegresion 模型

逻辑回归（Logistic Regression）是一种用于解决二分类问题的线性分类模型。该模型通过将输入特征与权重进行线性组合，然后将结果通过 sigmoid 函数进行压缩到 0 到 1 之间的概率范围内，以此来预测样本属于正类的概率。如果这个概率大于等于一个预定的阈值，通常是 0.5，那么这个样本就被分类为正类，否则被分类为负类。

逻辑回归是一种简单而常用的分类算法，具有计算速度快、易于理解和实现等优点。它还可以扩展到多类别分类问题（Multinomial Logistic Regression）和正则化（Regularized Logistic Regression）等情况。

我们使用默认参数，得到的结果为：训练集上的准确率为0.9078549848942599，测试集上的准确率为 0.9024144869215291。

	precision	recall	f1-score	support
0	0.93	0.96	0.95	903
1	0.45	0.30	0.36	91
accuracy			0.90	994
macro avg	0.69	0.63	0.65	994
weighted avg	0.89	0.90	0.89	994

图 13：逻辑回归的分类结果

此时得到的混淆矩阵为：

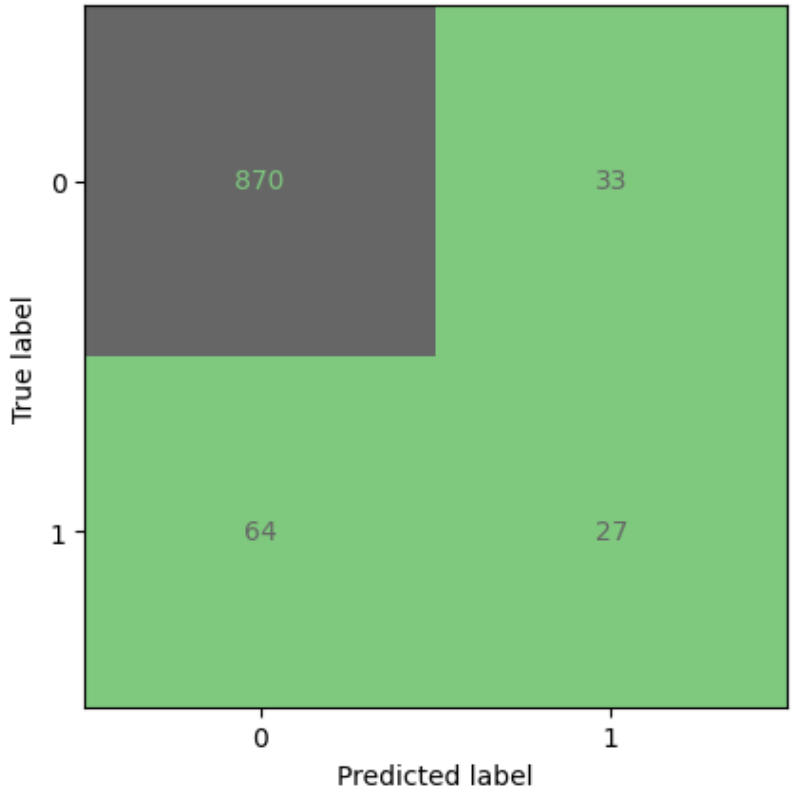


图 14：混淆矩阵

让我们改变一些参数：例如 C，我们得到的结果如下所示：

表 5：改变参数后的逻辑回归结果

类别	Results
Accuracy	0.902414
Precision	0.690739
Recall	0.630079
F1-score	0.652406

我们将一起看到结果。虽然已经获得了 Accuracy ，但是如果我们仔细观察与个人贷款功能相关的饼图，我们会发现已经贷款的人和没有贷款的人之间的分布不平衡。

我只是通过 f1 分数检查模型，因为 f1 是通过精度和召回率计算的。

另一方面，没有设置控制参数，这些是初步结果。我们经过调参后得到的最佳模型为：

LogisticRegression(C=4.281332398719396, max_iter=10000, random_state=0, solver='newton-cg')

此时的混淆矩阵为：

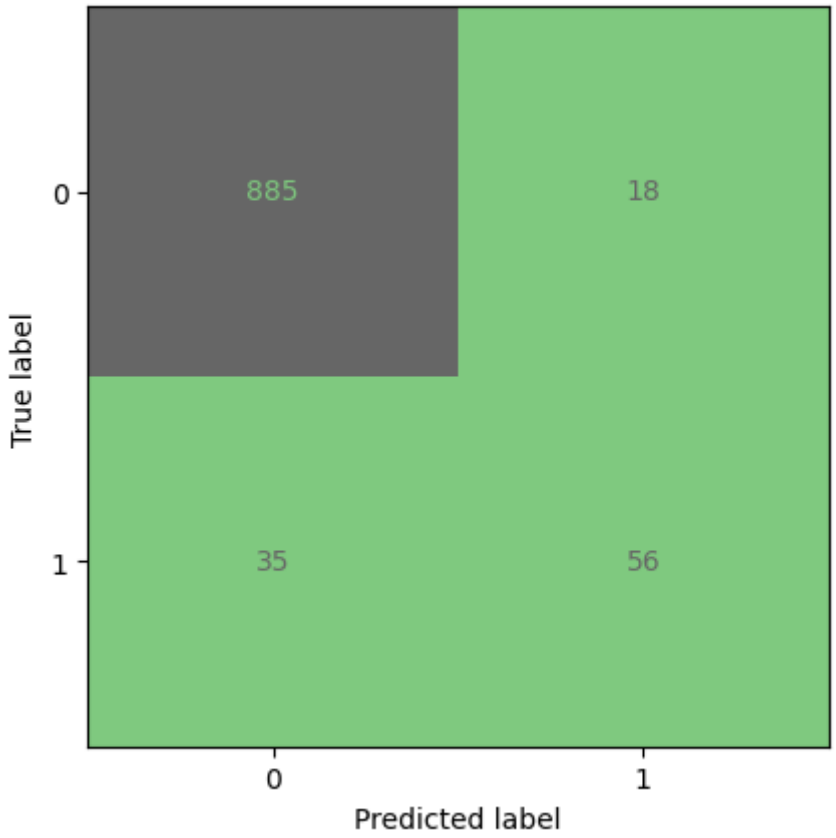


图 15：调参后的混淆矩阵

58 个人申请了贷款，我们给了他们贷款，这是这个数据集中最重要的事情。888 人没有要贷款，我们也没给他们贷款。另一方面，在这个模型中，我们认为只有 48 个人是错误的。

表 6：调参后的模型结果

类别	Results
Accuracy	0.946680
Precision	0.859357
Recall	0.797726
F1-score	0.824857

4.2 Naive Bayes 模型

朴素贝叶斯（Naive Bayes）是一种基于贝叶斯定理和特征条件独立假设的分类算法。朴素贝叶斯的分类思想很简单：对于给定的数据集，首先基于特征条件独立假设计算每个类别的条件概率，然后基于贝叶斯定理计算出新样本属于各个类别的后验概率，最终将新样本归类到后验概率最大的类别中。

具体来说，朴素贝叶斯算法假设各个特征之间是独立的，即对于一个分类问题，每个特征对于分类的影响是相互独立的。根据贝叶斯定理，可以得到一个后验概率公式：

$$P(y|x_1, x_2, ..., x_n) = \frac{P(x_1, x_2, ..., x_n|y)P(y)}{P(x_1, x_2, ..., x_n)}$$

对于一个新的样本，我们可以分别计算其属于每个类别的后验概率，然后将其归为后验概率最大的那个类别。

在不同的朴素贝叶斯模型中，我们选择了互补模型。 因为我们的数据集是不平衡的。

我们得出的结果为：训练集上的准确率为 0.7887713997985901，测试集上的准确率为 0.8038229376257545。

得到的混淆矩阵为：

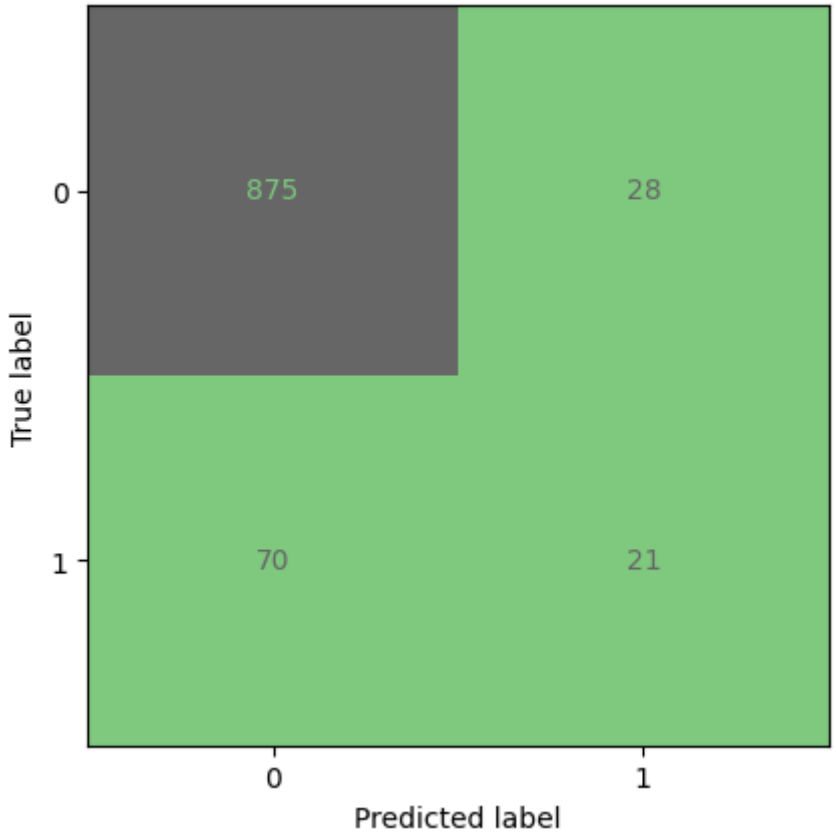


图 16：朴素贝叶斯模型的混淆矩阵

训练集和测试集的准确率分别为：

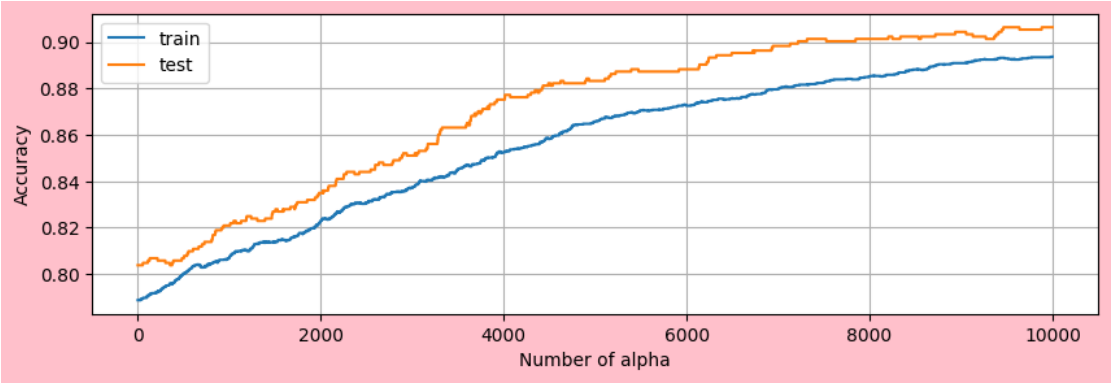


图 17：训练集和测试集的准确率

我们得到的模型的各项结果为：

表 7：朴素贝叶斯模型的结果

类别	Results
Accuracy	0.901408
Precision	0.677249
Recall	0.599881
F1-score	0.623485

4.3 GBDT 模型

GBDT，即梯度提升决策树（Gradient Boosting Decision Tree），是一种集成学习算法，通常用于解决分类和回归问题。

在 GBDT 中，每个决策树都是由多个弱决策树（shallow decision trees）组成的。模型训练时，先用一个简单模型（如决策树）拟合数据，然后根据模型误差计算权重，将权重分配给新的模型，重复该过程直到达到预设的迭代次数或模型性能收敛。最终将所有模型的预测结果相加，得到最终预测结果。

GBDT 模型的优点在于它能够在处理大规模高维数据时实现很好的性能，也很适合处理非线性问题。此外，由于该模型采用的是决策树作为基础模型，因此可以很好地处理缺失数据和异常值。

缺点在于它可能过度拟合，特别是在训练数据较少的情况下。此外，由于 GBDT 是串行计算，无法并行化处理，因此在处理大规模数据时可能需要较长的时间。

到目前为止，我们已经检查了 2 个模型并获得了结果。但他们的得分都没有超过 90%。

由于我在项目中多次强调我们的数据集是不平衡的，我们需要寻找一个模型来克服这个弱点。适用于此数据集的这些模型之一是决策树。

我们使用最初的模型，得到的结果分别为：训练集上的准确率是 0.9944612286002014，测试集上的准确率为 0.9879275653923542。

	precision	recall	f1-score	support
0	0.99	1.00	0.99	903
1	0.98	0.89	0.93	91
accuracy			0.99	994
macro avg	0.98	0.94	0.96	994
weighted avg	0.99	0.99	0.99	994

我们得到的混淆矩阵为：

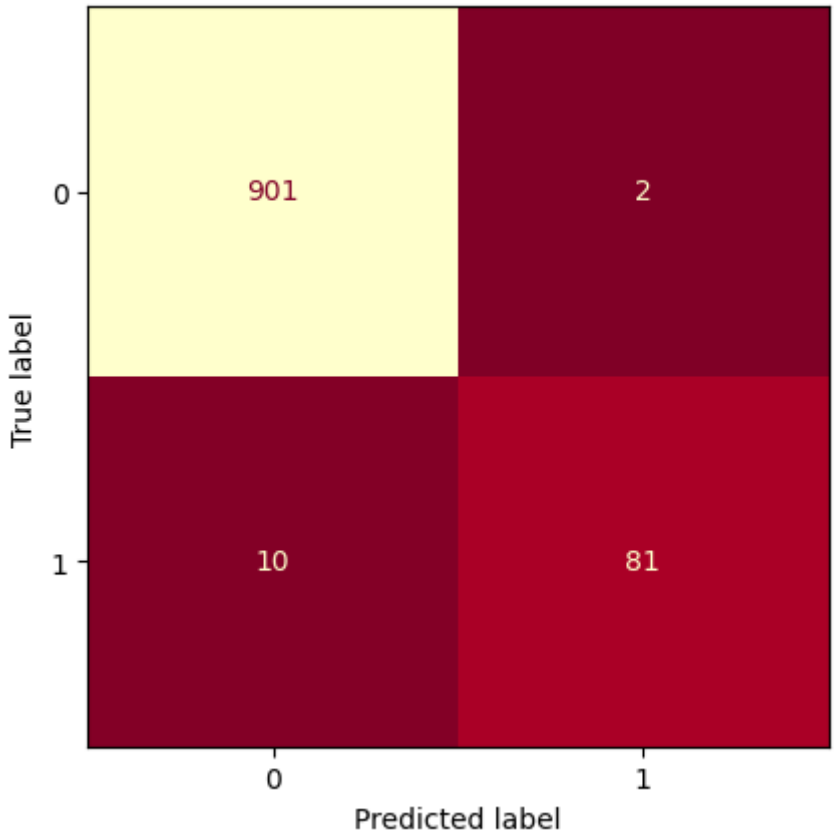


图 18：GBDT 模型的混淆矩阵

82 人申请贷款，我们给了他们贷款，而在此之前我们最好的模型，即回归，这个值等于 58 人。901 人没有申请贷款，我们也没有给他们贷款。而这个回归值等于 888 人。

五、 总结

根据上面的结论，我们汇总三个算法的结果，如下表所示：

表 8：三种算法结果对比

类别	LogisticRegression	Navie Bayes	GBDT
Accuracy	0.946680	0.901408	0.987928
Precision	0.859357	0.677249	0.982463
Recall	0.797726	0.599881	0.943948
F1-score	0.824857	0.623485	0.962210

根据上表我们不难看出，GradientBoostingClassifier 模型优于 Logistic Regression。对我们的数据集有一个较好的泛化能力！

参考文献

[1] Hu, J., Ren, X., & Zhang, X. (2020). Personal loan default prediction with machine learning. Journal of Computational Science, 41, 101106.

[2] Fawcett, T. (2004). ROC graphs: Notes and practical considerations for researchers. Machine learning, 31(1), 1-38.

[3] Wang, C., & Yu, J. (2015). Personal loan risk assessment based on decision tree algorithm. International Journal of Advancements in Computing Technology, 7(2), 5-12.

[4] Zhu, Z., Zhang, Y., & Sun, W. (2019). Personal credit risk assessment based on machine learning algorithms. Financial Innovation, 5(1), 1-11.

[5] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.

[6] https://www.kaggle.com/datasets/teertha/personal-loan-modeling

[7] https://scikit-learn.org/stable/modules/classes.html

附录

```
import re    #for Regular Expressions Analysing
import warnings
from uszipcode import SearchEngine # for searching the US zipcodes

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import
ComplementNB,BernoulliNB,GaussianNB,MultinomialNB,CategoricalNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn import metrics
from sklearn.metrics import
confusion_matrix,classification_report,ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.model_selection import cross_val_score, cross_validate
import plotly.express as px

df = pd.read_csv('./Bank_Personal_Loan_Modelling.csv')
df
```

```
df.info()

df.describe(include = 'all')

df.nunique().plot(kind='barh',color='#9845ae')
print(df.nunique())

n_columns=['Age','Income','CCAvg','Mortgage','Experience']
colors_n=['orange','aqua','red','olive','#e011f5']
cat_columns=['Family','Education','Securities Account','CD
Account','Online','CreditCard','Personal Loan']
colors_cat=['#8B008B','#DC143C','#FFA500','#008080','#556B2F','#D2691E','#DAA
520']
print('numerical features:')
for n_item in (n_columns):
    print(f'{n_item}:',' max=',(df[n_item].max()))
    print('\t','min=',(df[n_item].min()))
print('**************************')
print('categorical features:')
for cat_item in (cat_columns):
    print(f'{cat_item}:',' max=',(df[cat_item].max()))
    print('\t','min=',(df[cat_item].min()))
print('**************************')
for item in df.columns:
    print(f'{item}:',len(df[item].unique()))
    if (len(df[item].unique())==df.shape[0]):
        df.drop(item,inplace=True,axis=1)
df

df.nunique().plot(kind='barh',color='#cc45ae')
```

```
df['Experience']=df['Experience'].abs()

len(df[df['Experience']<0])

(df['CCAvg'].dtypes)

df['CCAvg'].replace(to_replace=r'/',value=r'.',regex=True,inplace=True)

df['CCAvg']=df['CCAvg'].astype(float).astype('int64')

df['CCAvg']=df['CCAvg']*12

df.dtypes

counties={}
engine = SearchEngine()
for item in df['ZIP Code'].unique():
    try:
        zipcode = engine.by_zipcode(item)
        county=zipcode.county
    except AttributeError as at:
        county=item
        print('we cant find the county of this zipcode')
    finally:
        counties.update({item:county})

pattern=re.compile(r'[0-9]+')
for key,value in counties.items():
    if re.match(pattern,str(value)):
        print(f'{key}:{value}')
```

```
df['Location']=df['ZIP Code'].map(counties) # here I ampped every unique location to their zip code

[len(df[df['Location']==i])for i in [92717,96651,92634]]

for i in [92717,96651,92634]:
    df.drop((df[df['Location']==i].index),inplace=True)

df.drop(df[df['Location']==''].index,inplace=True)

df.reset_index(inplace=True,drop=True)

plt.figure(figsize=(12,8),facecolor='#ec89a0')
sns.countplot(data=df,y='Location',palette='deep')
plt.xlabel('count')
plt.ylabel('Location in USA')
plt.grid()
plt.show()

fig, ax = plt.subplots(1,7, figsize=(12,3), dpi=100)
for i,c in enumerate(cat_columns):
df[c].value_counts(normalize=True).plot(kind='bar',color=colors_cat[i],ax=ax[i],title=c,fontsize=6)
    ax[i].spines['right'].set_color('white')
    ax[i].spines['top'].set_color('white')

df1=df.replace(to_replace={'Family':[1,2,3,4],'Education':[1,2,3],'Securities Account':[0,1],
                           'CD Account':[0,1],'Online':[0,1],'CreditCard':[0,1],'Personal Loan':[0,1]},
               value={'Family':['Single','Couple','Family of 3 people','Family of 4 people'],
```

```
                    'Education':['Undergrad','Graduted','Advanced'],
                    'Securities Account':['No-SA','SA'],
                    'CD Account':['No-CD','CD'],
                    'Online':['No-Internet Bank','Use Internet Bank'],
                    'CreditCard':['NO','YES'],
                    'Personal Loan':['NO','YES']})
for i,c in enumerate(cat_columns):
    plt.figure(figsize =(5.5, 5.5))
    plt.pie(df1[c].value_counts() ,labels=list(df1[c].value_counts().index),shadow =
True,autopct='%1.1f%%')
    plt.legend()
    plt.title(c,color=colors_cat[i],fontsize=20)

df_Age=pd.cut(x=df['Age'],bins=[20,30,40,50,60,70],labels=['in your twenties','in
your thirties',
                                                    'in your
forties','in your fifties','retirement'],include_lowest=True)
df_Experience=pd.cut(x=df['Experience'],bins=[0,10,20,30,40,50],labels=['Entry
level','Junior',

'Senior','High level','Expert'],include_lowest=True)
df_Income=pd.cut(x=df['Income'],bins=[0,50,100,150,200,250],labels=['Not
good','Good',

'Medium','High','Rich'],include_lowest=True)
df_CCAvg=pd.cut(x=df['CCAvg'],bins=[0,50,100,150],labels=['Stingy','Normal' ,'Spe
ndthrift'],include_lowest=True)

df_Mortgage=pd.cut(x=df['Mortgage'],bins=[0,100,200,300,400,500,600,700],labels=
['A','B','C','D','E','F','G'],include_lowest=True)

df2=pd.concat([df_Age,df_Experience,df_Income,df_CCAvg,df_Mortgage],axis=1)
```

```
for i,c in enumerate(n_columns):
    plt.figure(figsize =(5.5, 5.5))
    plt.pie(df2[c].value_counts() ,labels=list(df2[c].value_counts().index),shadow =
True,autopct='%1.1f%%')
    plt.legend()
    plt.title(c,color=colors_n[i],fontsize=20)

df3=pd.concat([df2['Age'],df2['Experience'],df1['Family'],df1['Education']],axis=1)
for col in df3.columns:
    sns.kdeplot(data=df, x='Income', hue=col, label=col)
    plt.xlabel('Income')
    plt.ylabel('Count')
    plt.show()

outliers_indexes=[]
for col in n_columns:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3-q1
    maximum = q3 + (1.5 * iqr)
    minimum = q1 - (1.5 * iqr)
    outlier_samples = df[(df[col] < minimum) | (df[col] > maximum)]
    outliers_indexes.extend(outlier_samples.index.tolist())
print(outlier_samples)
outliers_indexes = list(set(outliers_indexes))

fig=plt.figure(figsize=(20,15))
gs=mpl.gridspec.GridSpec(4,4)
ax0=fig.add_subplot(gs[0,0:2])
ax1=fig.add_subplot(gs[0,2:])
```

```
ax2=fig.add_subplot(gs[1,0])
ax3=fig.add_subplot(gs[1,1:])
ax4=fig.add_subplot(gs[2,0:])

# for i,c in enumerate(n_columns):
sns.boxplot(data=df,x='Age',color='blue',ax=ax0)
ax0.set_xticklabels('Age')
ax0.xaxis.label.set_size(20)

sns.boxplot(data=df,x='Experience',color='orange',ax=ax1)
ax1.set_xticklabels('Experience')
ax1.xaxis.label.set_size(20)

sns.boxplot(data=df,x='CCAvg',color='olive',ax=ax2)
ax2.set_xticklabels('CCAvg')
ax2.xaxis.label.set_size(20)

sns.boxplot(data=df,x='Mortgage',color='aqua',ax=ax3)
ax3.set_xticklabels('Mortgage')
ax3.xaxis.label.set_size(20)

sns.boxplot(data=df,x='Income',color='red',ax=ax4)
ax4.set_xticklabels('Income')
ax4.xaxis.label.set_size(20)

df.isnull().sum()

df.drop_duplicates(inplace=True)
df.shape

fig,ax=plt.subplots(1,1,figsize=(12,7))
```

```
mask=np.triu(np.ones_like(df.corr()))
heatmap=sns.heatmap(df.corr(),vmin=-
1,vmax=1,mask=mask,cmap='Pastel2',annot=True)
heatmap.set_title('Triangle',fontdict={'fontsize':12},pad=20)

fig = px.imshow(df.corr(), title='Heatmap of numerical values of the data')
fig.show()

df.drop('Experience',axis=1,inplace=True)

X=df.drop(['Personal Loan','Location'],axis=1)
Y=df['Personal Loan']

print('X shape: ', X.shape)
print('Y shape: ', Y.shape)

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)

reglog=LogisticRegression()

print('X_train shape: ', X_train.shape)
print('Y_train shape: ', Y_train.shape)

reglog.fit(X_train, Y_train)
y_pred_reg = reglog.predict(X_test)

print('score of train',reglog.score(X_train,Y_train))
print('score of test',reglog.score(X_test,Y_test))

print(classification_report(Y_test, y_pred_reg))
```

```
ConfusionMatrixDisplay.from_estimator(reglog, X_test, Y_test, colorbar=False,
cmap='Accent')
plt.grid(False)

result_reg=pd.DataFrame(data=[accuracy_score(Y_test, y_pred_reg),
                              precision_score(Y_test, y_pred_reg,
average='macro'),
                              recall_score(Y_test, y_pred_reg,
average='macro'),
                              f1_score(Y_test, y_pred_reg,
average='macro')],
                        index=['Accuracy','Precision','Recall','F1-score'],
                        columns = ['LogisticRegression Results'])
result_reg

gridsearch_log=LogisticRegression(C= 4.281332398719396,class_weight= None,
                                  dual= False,fit_intercept= True,
                                  intercept_scaling= 1,l1_ratio= None,
                                  max_iter= 10000,multi_class='auto',n_jobs= None,
                                  penalty='l2',random_state= 0,
                                  solver= 'newton-cg',tol= 0.0001,verbose= 0,
                                  warm_start= False)

fig,ax=plt.subplots(figsize=(10,3),facecolor='pink')
ax.plot(k_neighbours,train_acc,label="Train")
ax.plot(k_neighbours,test_acc,label="Test")
plt.ylabel('Accuracy')
plt.xlabel('Number of Neighbors')
plt.grid()
plt.legend()
```

```
fig,ax=plt.subplots(figsize=(10,3),facecolor='pink')
ax.plot(alphas,train_acc_nv,label='train')
ax.plot(alphas,test_acc_nv,label='test')
plt.ylabel('Accuracy')
plt.xlabel('Number of alpha')
plt.grid()
plt.legend()

best_result_nv=pd.DataFrame(data=[accuracy_score(Y_test, y_pred_nv1),
                                  precision_score(Y_test, y_pred_nv1,
average='macro'),
                                  recall_score(Y_test, y_pred_nv1,
average='macro'),
                                  f1_score(Y_test, y_pred_nv1,
average='macro')],
                            index=['Accuracy','Precision','Recall','F1-score'],
                            columns = ['Navie Bayes Results'])
best_result_nv

result_clf=pd.DataFrame(data=[accuracy_score(Y_test, y_pred_clf),
                                  precision_score(Y_test, y_pred_clf,
average='macro'),
                                  recall_score(Y_test, y_pred_clf,
average='macro'),
                                  f1_score(Y_test, y_pred_clf,
average='macro')],
                            index=['Accuracy','Precision','Recall','F1-score'],
                            columns = ['GradientBoostingClassifier'])
result_clf
```