销售数据分析与预测

方案描述

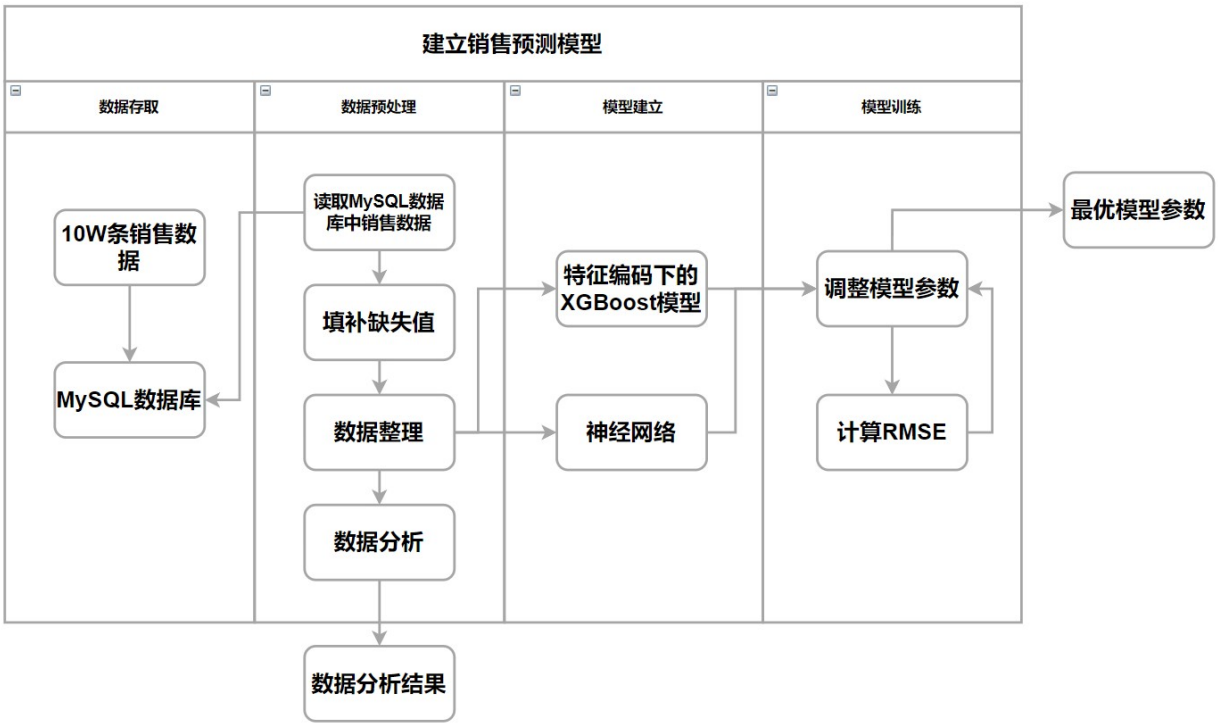
- 课题周期: 2023-07-04 ~ 2023-07-08
- 团队成员:
  - 企业导师：伍国栋、王一方
  - 课题小组: 第三组，小组成员与分工见下图

序号	姓名	角色/分工
1	李墨	搭建主要项目框架，包括设计数据库的读写操作、训练XGBoost及神经网络模型、设计API接口与测试
2	潘诚	参与API接口的设计
3	谭启泰	参与API接口的设计
4	林瑞奇	设计项目方案、制作PPT
5	李毓晟	绘制项目架构图
6	李岳锴	整理设计方案、绘制数据分析图、训练CatBoost模型
7	常雨欣	绘制数据分析图、训练XGBoost模型
8	王以柔	绘制数据分析图、训练Catboost模型
9	梅洁楠	训练XGBoost模型

整体架构设计

数据分析和预测层

数据分析和预测层负责对销售数据进行分析和预测，包括特征分析、模型训练和预测结果输出等操作。



系统关键设计点

本系统采用了一系列先进的技术和工具，以实现高效、稳定和可扩展的服务。以下是系统的关键设计点：

- 编程语言和数据库：系统的核心代码使用 Python 编写，Python 的简洁和强大的库使得代码易于编写和维护。同时，我们选择 MySQL 作为数据库，它的稳定性和高效性使得数据存储和查询变得简单快捷。

```
import python
import MySQLdb
```

- 轻量应用服务器：我们选择在 Ubuntu 20.04 轻量云服务器上运行 API 后端以及前端网页，这样可以节省资源，提高系统的响应速度。

```
lsb_release -a
```

- **Nginx** 和 **Flask**：我们使用 Nginx 作为反向代理服务器，它可以处理静态文件和 SSL/TLS 加密，提高系统的安全性和性能。我们使用 Flask 作为 Web 框架，它的轻量和灵活性使得 Web 应用的开发变得简单快捷。

```
sudo apt install nginx
pip install flask
```

- 实时预测：我们的前端网页通过 JavaScript 动态收集表格信息，实现实时预测。这样可以提供更好的用户体验，使得用户可以即时获取预测结果。

```
var data = collectData();
var result = predict(data);
```

- 模型部署：我们将 XGBoost 模型的最优模型参数和文件预处理所需字典部署到服务器，实现实时高性能预测。这样可以提高预测的准确性和效率。

```
import xgboost as xgb
model = xgb.XGBRegressor()
```

- 服务管理：我们使用 gunicorn 启动 Flask 程序，利用 Ubuntu systemd 管理开机自启动服务。这样可以保证服务的稳定运行，减少因为系统重启等原因导致的服务中断。

```
gunicorn myapp:app
systemctl enable myapp
```

- 神经网络训练和 **GPU** 加速：我们使用 PyTorch 进行神经网络训练，配合 2*PCIE A100 显卡进行 GPU 加速训练。这样可以大大提高模型训练的速度，缩短模型开发的周期。

```
import torch
model = torch.nn.Sequential()
```

- 数据分析和绘图：我们使用 SciencePlots 库进行数据分析绘图，使得图像更加优雅和科学。这样可以更好地理解数据，提供更好的数据可视化效果。

```
import matplotlib.pyplot as plt
plt.style.use('science')
```

- 模型优化：通过 XGBoost 模型结合 GridSearch 方案，找到最佳参数，优化模型性能。这样可以提高模型的预测准确性，提供更好的服务。

```
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(model, params)
```

总的来说，本系统采用了一系列先进的技术和工具，以实现高效、稳定和可扩展的服务。我们期待在未来的工作中，继续提高系统的性能和用户体验。

体会心得，未来的发展与设想

在这个项目中，我们学到了许多宝贵的知识和经验。首先，我们学习了如何使用 Python 和 MySQL 进行数据处理和分析，这使我们能够更有效地处理大量的数据，并从中提取有用的信息。

其次，我们学习了如何使用 Nginx 和 Flask 搭建 Web 服务，这使我们能够更方便地与用户交互，并提供更好的用户体验。最后，我们学习了如何使用 XGBoost 和 GridSearch 进行模型训练和优化，这使我们能够更准确地预测结果，并提高了我们的服务质量。

这个项目对我们的启发主要有两方面。一方面，我们认识到了数据分析和机器学习在解决实际问题中的重要性。通过这个项目，我们看到了如何将理论知识应用到实践中，解决实际的问题。另一方面，我们也认识到了团队合作的重要性。在这个项目中，我们需要协同工作，共同解决问题，这使我们更加明白团队合作的力量。

对于这个项目的未来发展，我们有许多设想。首先，我们希望能够进一步优化我们的模型，提高预测的准确性。我们计划使用更多的数据进行训练，使用更复杂的模型，以及使用更先进的优化算法。其次，我们希望能够提供更多的服务，比如提供更多的预测结果，提供更详细的数据分析报告，等等。最后，我们希望能够进一步提高我们的服务质量，比如提高服务的稳定性，提高服务的响应速度，等等。

总的来说，我们对这个项目充满了信心和期待。我们相信，通过我们的努力，我们能够使这个项目变得更好，为用户提供更好的服务。

XGBoost 预测 API 文档

概述

此 API 旨在使用预训练的 XGBoost 模型进行预测。该模型已在用户购买数据集上进行训练，并使用有关用户和他们购买的产品的各种特征进行预测。

端点

API 有一个单一的端点 `http://82.156.141.61/predict`，接受 POST 以及 GET 请求。

请求格式

请求应为包含以下键的 JSON 对象：

- `User_ID`：用户 ID 列表。
- `Product_ID`：产品 ID 列表。
- `Gender`：性别指标列表（0 代表女性，1 代表男性）。
- `Age`：年龄组指标列表。
- `Occupation`：职业代码列表。
- `City_Category`：城市类别代码列表。
- `Stay_In_Current_City_Years`：用户在当前城市居住年数的列表。
- `Marital_Status`：婚姻状态指标列表（0 代表单身，1 代表已婚）。
- `Product_Category_1`：主要产品类别代码列表。
- `Product_Category_2`：次要产品类别代码列表。
- `Product_Category_3`：第三产品类别代码列表。

以下是有效请求的示例：

```python
import requests

url = "http://82.156.141.61/predict"
# url = "http://127.0.0.1:5000/predict"

data = {
    "User_ID": [1005001, 1005001, 1005005],
    "Product_ID": ["P00113142", "P00001042", "P00057942"],
    "Gender": ['F', 'F', 'M'],
    "Age": ['26-36', '26-35', '46-50'],
    "Occupation": [2, 17, 16],
    "City_Category": ['B', 'B', 'B'],
    "Stay_In_Current_City_Years": ['1', '1', '2'],
    "Marital_Status": [0, 1, 0],
    "Product_Category_1": [1, 1, 1],
    "Product_Category_2": [5, 2, 2],
    "Product_Category_3": [12, 16, 6]
}

response = requests.post(url, json=data)
print(response.status_code)
print(response.text)
```

响应格式

响应将是包含以下键的 JSON 对象：

- `prediction`：：预测购买金额的列表。
- `rmse`：模型的均方根误差（RMSE）。

以下是有效响应的示例：

```
{
    "prediction": [15432.095703125,12817.9111328125,13056.142578125],
    'rmse': 2461.28,
    'R^2':  0.7530
}
```

错误处理

如果请求格式不正确，API 将返回 400 Bad Request 错误。如果存在内部服务器错误，API 将返回 500 Internal Server Error。

XGBoost Prediction API Documentation

Overview

This API is designed to make predictions using a pre-trained XGBoost model. The model has been trained on a dataset of user purchases, and it uses various features about the users and the products they bought to make its predictions.

Endpoint

The API has a single endpoint at `http://82.156.141.61/predict` which accepts POST and GET requests.

Request Format

The request should be a JSON object containing the following keys:

- `User_ID`: A list of user IDs.
- `Product_ID`: A list of product IDs.
- `Gender`: A list of gender indicators (0 for female, 1 for male).
- `Age`: A list of age group indicators.
- `Occupation`: A list of occupation codes.
- `City_Category`: A list of city category codes.
- `Stay_In_Current_City_Years`: A list of the number of years the user has lived in the current city.
- `Marital_Status`: A list of marital status indicators (0 for single, 1 for married).
- `Product_Category_1`: A list of primary product category codes.
- `Product_Category_2`: A list of secondary product category codes.
- `Product_Category_3`: A list of tertiary product category codes.

Here is an example of a valid request:

```
import requests

url = "http://82.156.141.61/predict"

data = {
    "User_ID": [1005001, 1005001, 1005005],
    "Product_ID": ["P00113142", "P00001042", "P00057942"],
    "Gender": ['F', 'F', 'M'],
    "Age": ['26-36', '26-35', '46-50'],
    "Occupation": [2, 17, 16],
    "City_Category": ['B', 'B', 'B'],
    "Stay_In_Current_City_Years": ['1', '1', '2'],
    "Marital_Status": [0, 1, 0],
    "Product_Category_1": [1, 1, 1],
    "Product_Category_2": [5, 2, 2],
```

```
    "Product_Category_3": [12, 16, 6]
}


response = requests.post(url, json=data)
print(response.status_code)
print(response.text)
```

Response Format

The response will be a JSON object containing the following keys:

- `prediction`: A list of predicted purchase amounts.
- `rmse`: The root mean square error (RMSE) of the model.

Here is an example of a valid response:

```
{
    "prediction": [15432.095703125,12817.9111328125,13056.142578125],
    'rmse': 2461.28,
    'R^2':  0.7530
}
```

Error Handling

If the request is not properly formatted, the API will return a 400 Bad Request error. If there is an internal server error, the API will return a 500 Internal Server Error.

课题内容

场景

- 课题背景: 数据分析和预测
- 课题目标: 通过分析销售数据，分析用户特性分布和相关性，并预测销售额。

Stakeholder

- 业务用户: 直接用户
- 业务 **owner**: 掌握用户的购买力情况

OKR（要验证什么，含业务验证和技术验证）

- **O**: 销售数据的分析与预测
- 业务验证:
    a. 分析数据特征的分布和相关性
    b. 预测销售额

前提

1. 给定一组销售训练数据和测试数据；将数据导入到数据库，后续所有动作的数据都基于数据库读写。

基础功能

1. 将数据导入到数据库，后续所有动作的数据都基于数据库读写；
2. 分析出销售数据各特征的分布情况，以及各特征与购买金额的相关性；
3. 预测给定验证集中的销售额，并计算预测结果的均方根误差 (RMSE)；

挑战项

1. 发布 REST API，调用 API 输出预测结果和预测指标 RMSE

业务目标

1. 分析给定数据的各个特征的分布，以及各个特征与销售额的相关性；
2. 预测给定验证集的销售额。

课题内容

给定条件

给定一组销售数据，有训练数据（train）和验证数据（test），训练集的量级为 10w 级，两份数据均为文档形式提供，该数据为经过过敏转换过的数据。

数据列表包括：

- User_ID: 用户唯一 ID
- Product_ID: 产品唯一 ID
- Gender: 性别
- Age: 年龄
- Occupation: 职业
- City_Category: 城市类别
- Stay_In_Current_City_Years: 本地城市居住年数
- Marital_Status: 婚姻状态
- Product_Category_1: 产品分类 1
- Product_Category_2: 产品分类 2
- Product_Category_3: 产品分类 3
- Purchase: 销售额

课题任务

1. 完成训练数据的特征分析，并输出分析结果；
2. 生成一个预测模型和模型的 R 方值；
3. 基于生成的模型，预测出验证数据的销售金额，同时给出预测结果评价值（采用均方根误差：RMSE）.
4. R^2 的计算公式参考:

$$R^2=1-\frac{\sum_{i=1}^{n}\left(\hat{y}_i-y_i\right)^2}{\sum_{i=1}^{n}\left(\acute{y}-y_i\right)^2}$$

, 其中, $y^{(i)}$ 是真实值, $\hat{y}^{(i)}$ 表示预测值, $\acute{y}$ 表示样本均值。模型效果评价指标 RMSE 的公式参考:

$$RMSE=\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(y_i-\hat{y}_i\right)^2}$$

详细说明

1. 整体流程介绍：
2. 将数据导入到数据库，后续所有动作的数据都基于数据库读写（不限定数据库类型）。
3. 针对训练数据，分析出数据各个特征的分布情况，以及所有特征与购买金额的相关性；分析结果用代码集中输出到一个 pdf 文档中。
4. 使用训练数据训练一个模型；生成模型后，用 R2 评估该模型的稳定性。
5. 基于生成的模型，预测给定验证集的销售金额，同时需要输出预测值的评价指标 RMSE
6. 加分内容：将预测模型发布为一个服务，通过 RESTFUL API 的方式提供预测能力；调用 API，输入其他特征，得到销售金额预测结果和 RMSE 指标。

课题基本要求

- 环境限定说明：编程语言：不限定，建议 Python\Java，数据库类型：不限定，建议 MySQL
- 需要给出设计方案，设计方案的形式不做强制要求，建议参考 UML 标准建模语言
- 预测模型的说服力指标参考：R^2>80%
- 预测效果的评价参考：(1) R^2>80%, (2) RMSE 值越小; 即在模型有说服力的前提下，销售预测结果越要接近真实值
- 性能要求：分析处理 10w 级数据，秒级出结果
- 加分项要求：发布的 API 服务具备一定的软件工程要求，没有 bug，代码结构和逻辑清晰

方案实现

在 Ubuntu 18.04 上安装 MySQL 并导入数据

1. 安装 MySQL 服务器：

```
sudo apt update
sudo apt install mysql-server
```

在安装过程中，系统可能会提示你设置 MySQL 的 root 用户密码。

1. 安装 Python 的 MySQL 库：

```
pip install pymysql
```

我们需要 Python 的 MySQL 库来连接 MySQL 数据库。

1. 导入数据： 首先，我们需要在 MySQL 中创建一个数据库和表格。你可以使用以下命令登录 MySQL：

```
mysql -u root -p
```

然后，创建一个名为 sales 的数据库，然后在该数据库中创建一个名为 train 的表格：

```
CREATE DATABASE sales;
USE sales;
CREATE TABLE train (
    User_ID INT,
    Product_ID VARCHAR(20),
    Gender CHAR(1),
    Age VARCHAR(10),
    Occupation INT,
    City_Category CHAR(1),
    Stay_In_Current_City_Years VARCHAR(10),
    Marital_Status INT,
    Product_Category_1 INT,
    Product_Category_2 INT,
    Product_Category_3 INT,
    Purchase INT
);
```

同理在该数据库中创建一个名为 test 的表格：

```
CREATE DATABASE sales;
USE sales;
CREATE TABLE test (
    User_ID INT,
    Product_ID VARCHAR(20),
    Gender CHAR(1),
    Age VARCHAR(10),
    Occupation INT,
    City_Category CHAR(1),
    Stay_In_Current_City_Years VARCHAR(10),
    Marital_Status INT,
    Product_Category_1 INT,
    Product_Category_2 INT,
    Product_Category_3 INT,
    Purchase INT
);
```

1. 更改 root 用户的身份验证方法：

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'your_password';
FLUSH PRIVILEGES;
```

安装 SQLAlchemy 库。使用以下命令安装：

```
pip install sqlalchemy
```

数据导入 MYSQL

使用以下代码创建一个 SQLAlchemy 引擎并将数据导入到 MySQL 数据库：

```
import pandas as pd
from sqlalchemy import create_engine

# 读取Excel文件
data = pd.read_excel('/root/train.xlsx')

# 创建SQLAlchemy引擎
engine = create_engine('mysql+pymysql://root:root
%40%40123@localhost:3306/sales')

# 将数据导入到MySQL数据库
data.to_sql('train', engine, if_exists='append', index=False)


539069

import pandas as pd
from sqlalchemy import create_engine

# 读取Excel文件
data = pd.read_excel('/root/test.xlsx')

# 创建SQLAlchemy引擎
engine = create_engine('mysql+pymysql://root:root
%40%40123@localhost:3306/sales')

# 将数据导入到MySQL数据库
data.to_sql('test', engine, if_exists='append', index=False)

300
```

数据探索

在开始建模之前，我们首先需要对数据进行探索，了解数据的基本情况。

```
# 导入必要的库
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import math
from sqlalchemy import create_engine

# 创建SQLAlchemy引擎
engine = create_engine('mysql+pymysql://root:root
%40%40123@localhost:3306/sales')

# 设置显示参数
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
sns.set(style='whitegrid', palette='muted', color_codes=True)
plt.rcParams['figure.figsize'] = [15, 10]

# 定义列的数据类型
dtype_dict = {
    'User_ID': int,
    'Product_ID': str,
    'Gender': str,
    'Age': str,
    'Occupation': int,
    'City_Category': str,
    'Stay_In_Current_City_Years': str,
    'Marital_Status': int,
    'Product_Category_1': int,
    'Product_Category_2': int,
    'Product_Category_3': int,
    'Purchase': int
}



# 读取数据
train_data = pd.read_sql('SELECT * FROM train LIMIT 35000',
engine).fillna(-1)
test_data = pd.read_sql('SELECT * FROM test', engine)

# 修改数据类型
for col, dtype in dtype_dict.items():
    train_data[col] = train_data[col].astype(dtype)
#     test_data[col] = test_data[col].astype(dtype)

# 显示数据的前5行
train_data.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	\
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	

```
3  1000001  P00085442      F  0-17          10             A
4  1000002  P00285442      M   55+          16             C

  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  \
0                          2               0                   3
1                          2               0                   1
2                          2               0                  12
3                          2               0                  12
4                         4+               0                   8

   Product_Category_2  Product_Category_3  Purchase
0                  -1                  -1      8370
1                   6                  14     15200
2                  -1                  -1      1422
3                  14                  -1      1057
4                  -1                  -1      7969
```

```
# 查看数据的基本信息
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35000 entries, 0 to 34999
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   User_ID                     35000 non-null  int64
 1   Product_ID                  35000 non-null  object
 2   Gender                      35000 non-null  object
 3   Age                         35000 non-null  object
 4   Occupation                  35000 non-null  int64
 5   City_Category               35000 non-null  object
 6   Stay_In_Current_City_Years  35000 non-null  object
 7   Marital_Status              35000 non-null  int64
 8   Product_Category_1          35000 non-null  int64
 9   Product_Category_2          35000 non-null  int64
 10  Product_Category_3          35000 non-null  int64
 11  Purchase                    35000 non-null  int64
dtypes: int64(7), object(5)
memory usage: 3.2+ MB
```

```
# 查看数值型特征的基本统计信息
train_data.describe()
```

```
            User_ID    Occupation  Marital_Status  Product_Category_1
\
count  3.500000e+04  35000.000000    35000.000000        35000.000000

mean   1.002699e+06      8.233086        0.403686            5.309286

std    1.546769e+03      6.539343        0.490643            3.729653
```

```
min    1.000001e+06      0.000000        0.000000             1.000000

25%    1.001340e+06      3.000000        0.000000             1.000000

50%    1.002760e+06      7.000000        0.000000             5.000000

75%    1.004022e+06     14.000000        1.000000             8.000000

max    1.005397e+06     20.000000        1.000000            18.000000


       Product_Category_2  Product_Category_3      Purchase
count        35000.000000        35000.000000  35000.000000
mean             6.447029            3.149886   9268.761086
std              6.565583            6.699634   4943.420216
min             -1.000000           -1.000000    185.000000
25%             -1.000000           -1.000000   5850.750000
50%              5.000000           -1.000000   8048.000000
75%             14.000000            8.000000  12025.000000
max             18.000000           18.000000  23958.000000

test_data.head()

   User_ID Product_ID Gender    Age  Occupation City_Category  \
0  1005001  P00113142      F  26-35           2             B
1  1005002  P00001042      M  26-35          17             B
2  1005005  P00057942      M  46-50          16             B
3  1005008  P00057942      M  26-35          20             B
4  1005011  P00015542      M  18-25           4             B

  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  \
0                          1               0                   1
1                          1               1                   1
2                          2               0                   1
3                          1               1                   1
4                          1               0                   1

   Product_Category_2  Product_Category_3 Purchase
0                   5                  12     None
1                   2                  16     None
2                   2                   6     None
3                   2                   6     None
4                   2                  13     None
```

数据分析

在进行建模之前，我们需要对数据进行深入的分析。我们将查看各个特征与目标变量（销售额）之间的关系。

1. 查看数据基本信息

```
# 显示数据的前5行
train_data = pd.read_sql('SELECT * FROM train LIMIT 35000', engine)
train_data.head()
```

```
   User_ID Product_ID Gender   Age  Occupation City_Category  \
0  1000001  P00069042      F  0-17          10             A
1  1000001  P00248942      F  0-17          10             A
2  1000001  P00087842      F  0-17          10             A
3  1000001  P00085442      F  0-17          10             A
4  1000002  P00285442      M   55+          16             C

  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  \
0                          2               0                   3
1                          2               0                   1
2                          2               0                  12
3                          2               0                  12
4                         4+               0                   8

   Product_Category_2  Product_Category_3  Purchase
0                 NaN                 NaN      8370
1                 6.0                14.0     15200
2                 NaN                 NaN      1422
3                14.0                 NaN      1057
4                 NaN                 NaN      7969
```

```
test_data.head()
```

```
   User_ID Product_ID Gender    Age  Occupation City_Category  \
0  1005001  P00113142      F  26-35           2             B
1  1005002  P00001042      M  26-35          17             B
2  1005005  P00057942      M  46-50          16             B
3  1005008  P00057942      M  26-35          20             B
4  1005011  P00015542      M  18-25           4             B

  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  \
0                          1               0                   1
1                          1               1                   1
2                          2               0                   1
3                          1               1                   1
4                          1               0                   1

   Product_Category_2  Product_Category_3 Purchase
0                   5                  12     None
1                   2                  16     None
2                   2                   6     None
3                   2                   6     None
4                   2                  13     None
```

```
# 查看数据的基本信息
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35000 entries, 0 to 34999
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   User_ID                     35000 non-null  int64
 1   Product_ID                  35000 non-null  object
 2   Gender                      35000 non-null  object
 3   Age                         35000 non-null  object
 4   Occupation                  35000 non-null  int64
 5   City_Category               35000 non-null  object
 6   Stay_In_Current_City_Years  35000 non-null  object
 7   Marital_Status              35000 non-null  int64
 8   Product_Category_1          35000 non-null  int64
 9   Product_Category_2          23948 non-null  float64
 10  Product_Category_3          10573 non-null  float64
 11  Purchase                    35000 non-null  int64
dtypes: float64(2), int64(5), object(5)
memory usage: 3.2+ MB
```

由上面的 Non-Null 的数值可以看出，train_data 数据中存在部分缺失值

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 0 to 299
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   User_ID                     300 non-null    int64
 1   Product_ID                  300 non-null    object
 2   Gender                      300 non-null    object
 3   Age                         300 non-null    object
 4   Occupation                  300 non-null    int64
 5   City_Category               300 non-null    object
 6   Stay_In_Current_City_Years  300 non-null    object
 7   Marital_Status              300 non-null    int64
 8   Product_Category_1          300 non-null    int64
 9   Product_Category_2          300 non-null    int64
 10  Product_Category_3          300 non-null    int64
 11  Purchase                    0 non-null      object
dtypes: int64(6), object(6)
memory usage: 28.2+ KB
```

由上面的 Non-Null 的数值可以看出，test_data 特征中没有缺失值，不需要对 test_data 做缺失值处理

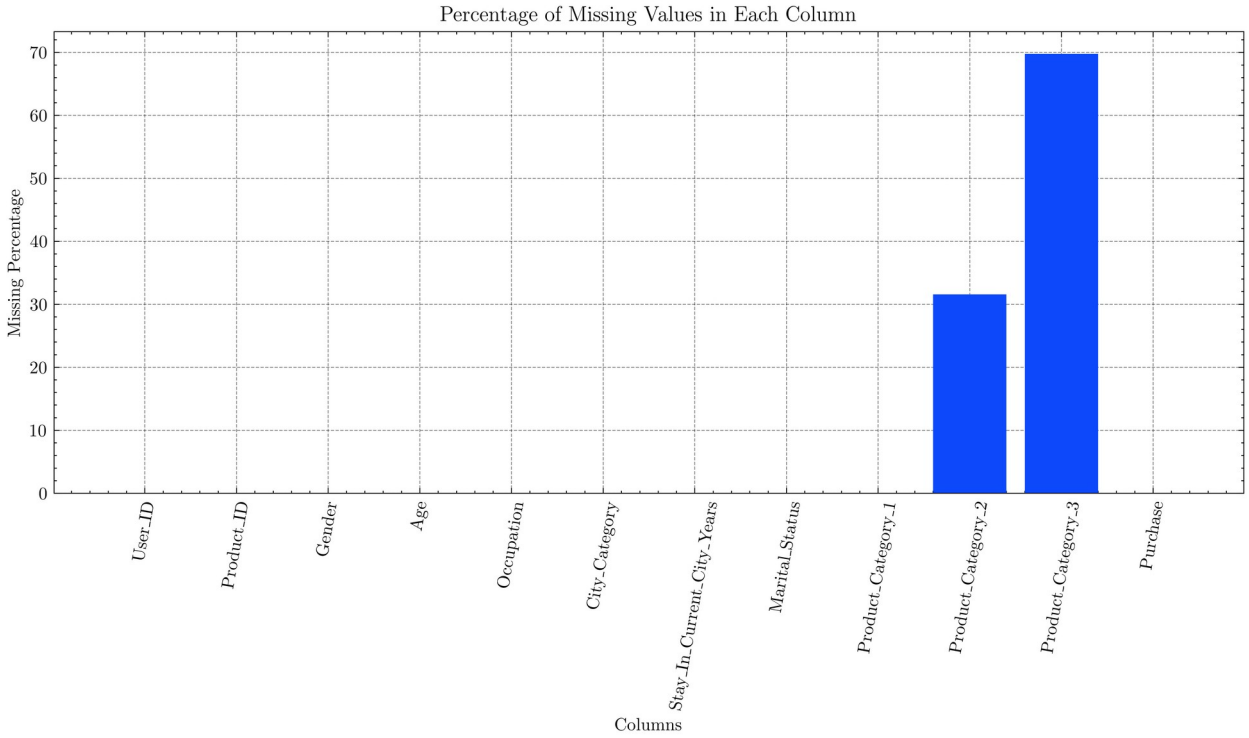
```
# 查看数值型特征的基本统计信息
train_data.describe()
```

```
            User_ID    Occupation  Marital_Status  Product_Category_1
\
count  3.500000e+04  35000.000000    35000.000000        35000.000000

mean   1.002699e+06      8.233086        0.403686            5.309286

std    1.546769e+03      6.539343        0.490643            3.729653

min    1.000001e+06      0.000000        0.000000            1.000000

25%    1.001340e+06      3.000000        0.000000            1.000000

50%    1.002760e+06      7.000000        0.000000            5.000000

75%    1.004022e+06     14.000000        1.000000            8.000000

max    1.005397e+06     20.000000        1.000000           18.000000


       Product_Category_2  Product_Category_3      Purchase
count        23948.000000        10573.000000  35000.000000
mean             9.883832           12.737444   9268.761086
std              5.059090            4.107650   4943.420216
min              2.000000            3.000000    185.000000
25%              5.000000            9.000000   5850.750000
50%              9.000000           15.000000   8048.000000
75%             15.000000           16.000000  12025.000000
max             18.000000           18.000000  23958.000000
```

缺失值可视化和处理

```
import matplotlib.pyplot as plt
import scienceplots
import seaborn as sns
plt.style.use('default')
plt.style.use(['science','high-vis','grid'])

plt.rcParams['figure.dpi'] = 600
# 可视化缺失值
missing_percent = train_data.isnull().mean() * 100

# 绘制条形图
plt.figure(figsize=(10, 6))
plt.bar(missing_percent.index, missing_percent.values)
plt.xticks(rotation=80)  # 设置x轴标签的旋转角度，以便更好地显示标签
plt.xlabel('Columns')
plt.ylabel('Missing Percentage')
plt.title('Percentage of Missing Values in Each Column')
```

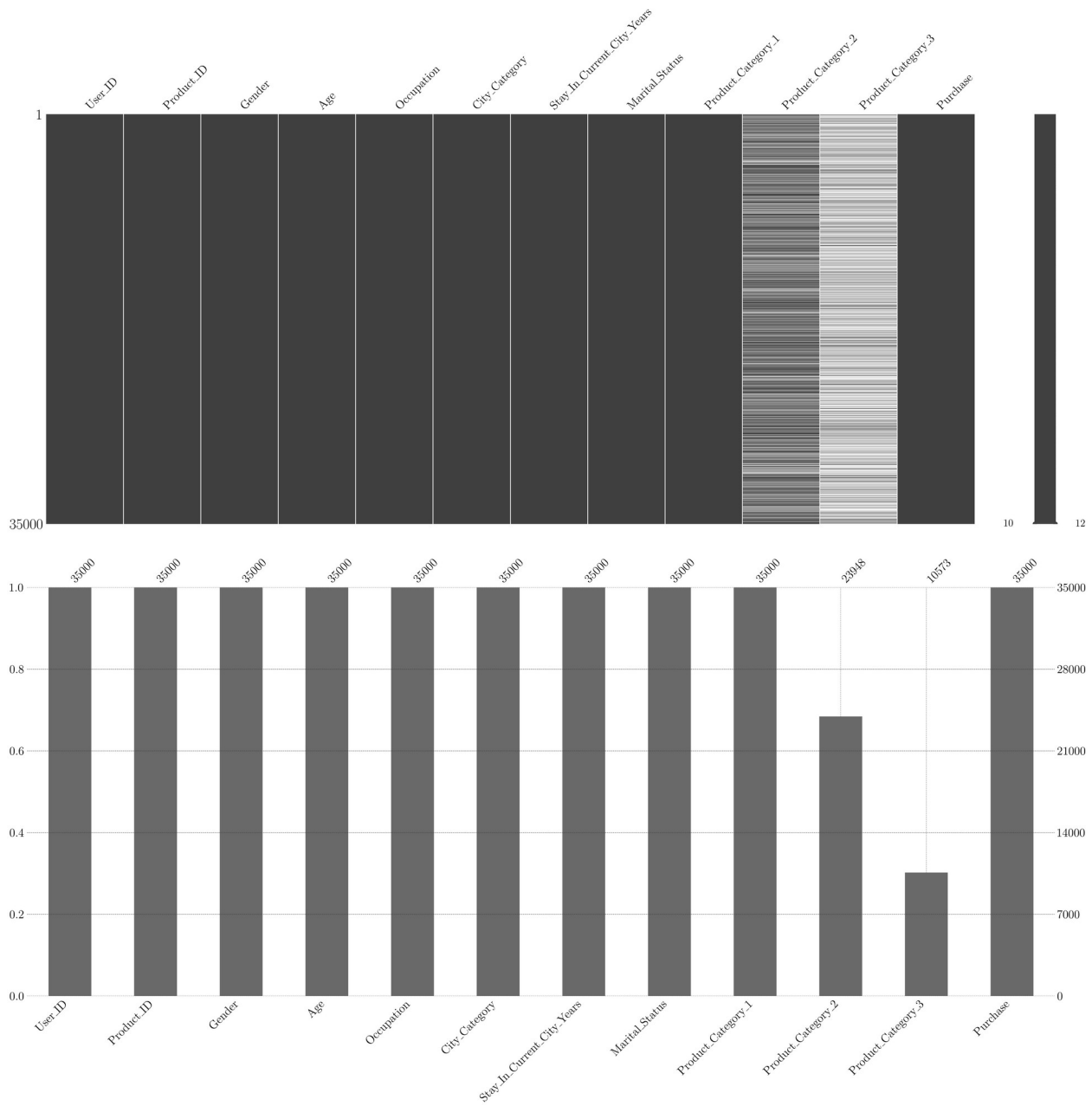
```
plt.tight_layout()
plt.show()
```




```
!pip install missingno

Collecting missingno
  Downloading missingno-0.5.2-py3-none-any.whl (8.7 kB)
Requirement already satisfied: matplotlib in
./anaconda3/lib/python3.10/site-packages (from missingno) (3.7.0)
Requirement already satisfied: scipy in
./anaconda3/lib/python3.10/site-packages (from missingno) (1.10.0)
Requirement already satisfied: numpy in
./anaconda3/lib/python3.10/site-packages (from missingno) (1.23.5)
Requirement already satisfied: seaborn in
./anaconda3/lib/python3.10/site-packages (from missingno) (0.12.2)
Requirement already satisfied: python-dateutil>=2.7 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(1.0.5)
Requirement already satisfied: packaging>=20.0 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(22.0)
Requirement already satisfied: pyparsing>=2.3.1 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(3.0.9)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(1.4.4)
Requirement already satisfied: fonttools>=4.22.0 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(4.25.0)
Requirement already satisfied: pillow>=6.2.0 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(9.4.0)
Requirement already satisfied: cycler>=0.10 in
./anaconda3/lib/python3.10/site-packages (from matplotlib->missingno)
(0.11.0)
Requirement already satisfied: pandas>=0.25 in
./anaconda3/lib/python3.10/site-packages (from seaborn->missingno)
(1.5.3)
Requirement already satisfied: pytz>=2020.1 in
./anaconda3/lib/python3.10/site-packages (from pandas>=0.25->seaborn-
>missingno) (2022.7)
Requirement already satisfied: six>=1.5 in
./anaconda3/lib/python3.10/site-packages (from python-dateutil>=2.7-
>matplotlib->missingno) (1.16.0)
Installing collected packages: missingno
Successfully installed missingno-0.5.2
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

```
import missingno as msno
msno.matrix(train_data)
plt.show()

msno.bar(train_data)
plt.show()
```



对于'Product_Category_2'列和'Product_Category_3'列的缺失值，我们可以采取以下几种策略：

1. 删除含有缺失值的行：这种方法简单，但可能会导致信息的丢失。
2. 填充缺失值：可以使用某个固定值、平均值、中位数或众数来填充缺失值。
3. 使用模型预测缺失值：这种方法可能会更准确，但计算复杂度较高。

在这里，我们选择使用固定值-1来填充'Product_Category_2'列和'Product_Category_3'列的缺失值。

```
# 读取数据
train_data = pd.read_sql('SELECT * FROM train', engine).fillna(-1)
```

现在，我们已经处理了所有的缺失值，可以开始进行数据分析和建模了。

数据相关性分析

本数据特征中大量是离散的。离散型变量的特点是取值为有限的离散值，这与连续型变量的特点截然不同，因此在计算离散型变量之间的相关性时，不能使用 Pearson 或 Spearman 相关系数。Pearson 相关系数是基于线性关系的，用于衡量两个连续型变量之间的线性相关性，而 Spearman 相关系数则是基于秩次关系的，用于衡量两个连续型变量之间的单调相关性。这两种相关系数都无法直接应用于离散型变量。

为了计算离散型变量之间的相关性，需要使用一些特殊的相关系数，其中比较常用的是 Cramer's V 相关系数和 Phi-K 相关系数（也称为 Phik）。

- Cramer's V 相关系数是一种基于卡方检验（chi-squared test）的相关系数，用于衡量两个离散型变量之间的相关性。Cramer's V 相关系数的取值范围为 0 到 1，其中 0 表示两个变量独立，1 表示两个变量完全相关。Cramer's V 的计算公式如下：

$$V = \sqrt{\frac{\chi^2/n}{min(k-1, r-1)}}$$

  其中，χ^2 是卡方值，n 是样本数量，k 和 r 分别是两个变量的类别数。

- Phi-K 相关系数也是一种基于卡方检验的相关系数，用于衡量两个二元离散型变量之间的相关性。Phi-K 相关系数的取值范围也是 0 到 1，其中 0 表示两个变量独立，1 表示两个变量完全相关。Phi-K 的计算公式如下：

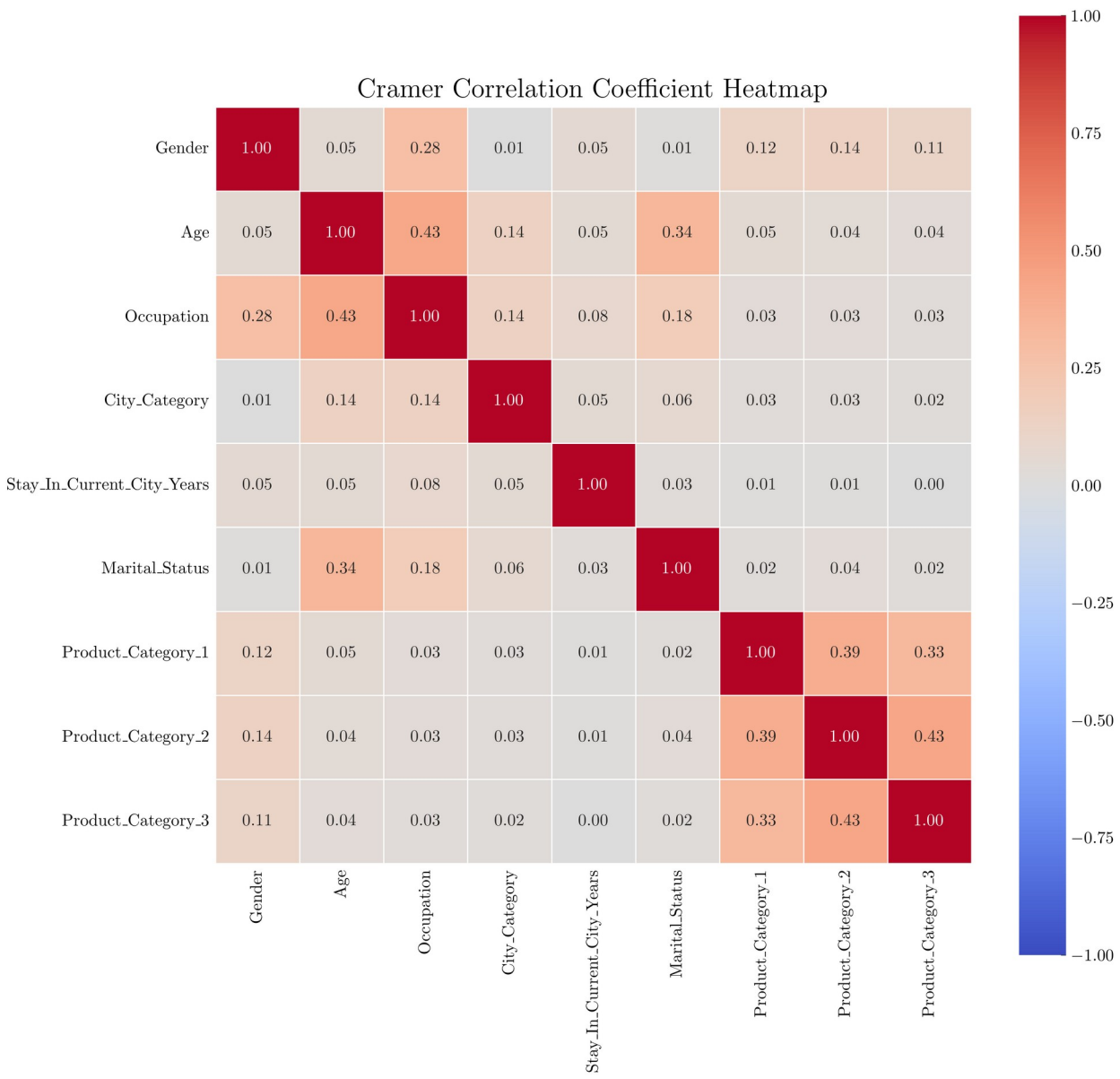
$$\phi = \sqrt{\frac{\chi^2}{n}}$$

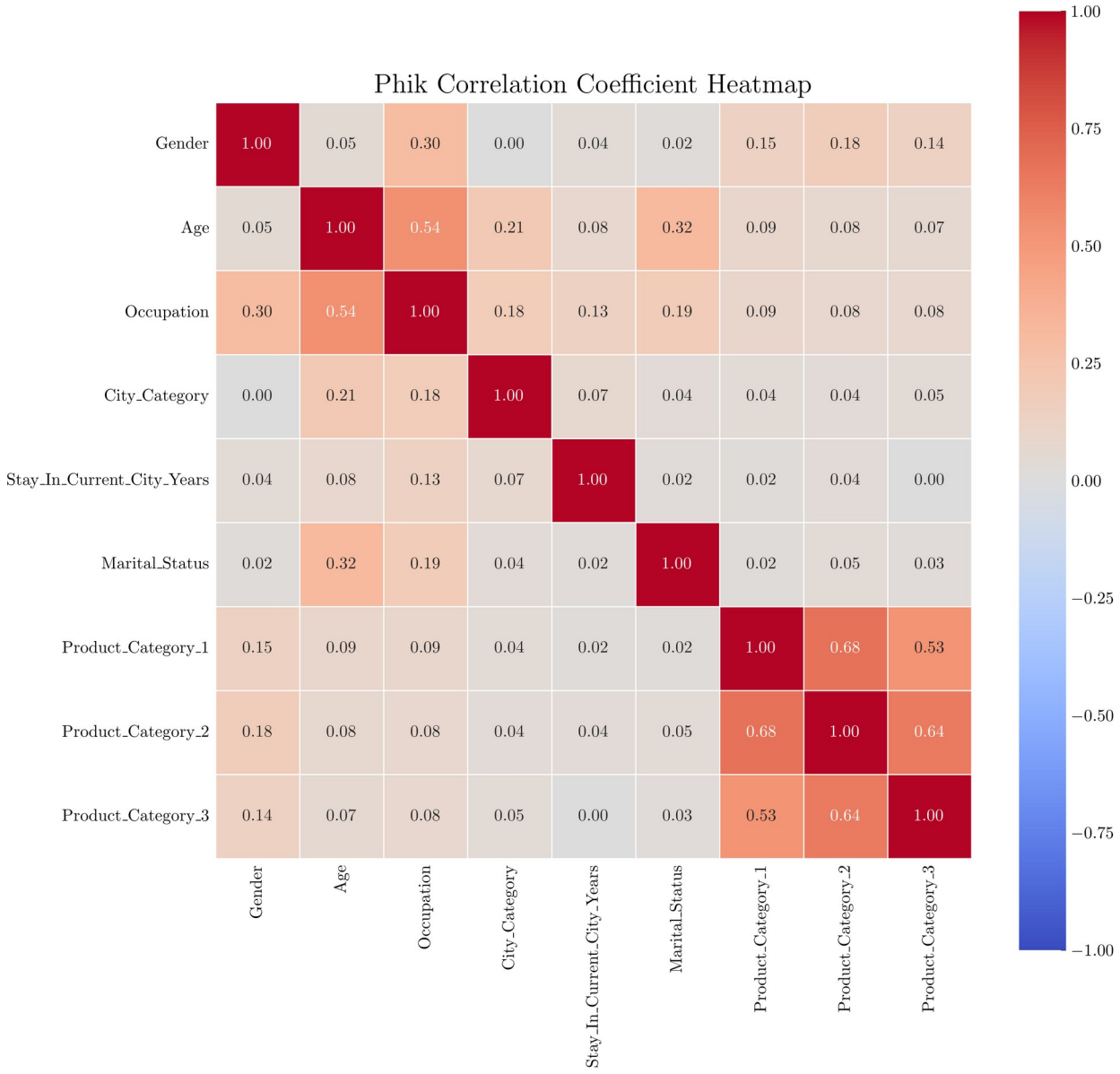
  其中，χ^2 是卡方值，n 是样本数量。

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from phik import phik_matrix
from scipy.stats import chi2_contingency


def cramers_v(x, y):
    confusion_matrix = pd.crosstab(x, y)
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2_corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    r_corr = r - ((r-1)**2)/(n-1)
    k_corr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2_corr / min(k_corr-1, r_corr-1))
```

```
def heat_map(corr_matrix, title):
    plt.figure(figsize=(10, 10))
    sns.heatmap(
        corr_matrix,
        annot=True,
        fmt=".2f",
        cmap="coolwarm",
        vmin=-1,
        vmax=1,
        square=True,
        linewidths=0.5,
    )
    plt.title(title, fontsize=16)
    plt.show()


# 选择离散变量
discrete_columns = ["Gender", "Age", "Occupation", "City_Category",
"Stay_In_Current_City_Years",

"Marital_Status","Product_Category_1","Product_Category_2","Product_Ca
tegory_3"]
# 计算Cramer相关系数
cramer_corr = pd.DataFrame(
    [
        [cramers_v(train_data[col1], train_data[col2]) for col1 in
discrete_columns]
        for col2 in discrete_columns
    ],
    columns=discrete_columns,
    index=discrete_columns,
)

# 计算Phik相关系数
phik_corr = train_data[discrete_columns].phik_matrix()

# 绘制热力图
heat_map(cramer_corr, "Cramer Correlation Coefficient Heatmap")
heat_map(phik_corr, "Phik Correlation Coefficient Heatmap")
```

```
interval columns not set, guessing: ['Occupation', 'Marital_Status',
'Product_Category_1', 'Product_Category_2', 'Product_Category_3']
```


Cramer Correlation Coefficient Heatmap
Gender 1.00 0.05 0.28 0.01 0.05 0.01 0.12 0.14 0.11
Age 0.05 1.00 0.43 0.14 0.05 0.34 0.05 0.04 0.04
Occupation 0.28 0.43 1.00 0.14 0.08 0.18 0.03 0.03 0.03
City_Category 0.01 0.14 0.14 1.00 0.05 0.06 0.03 0.03 0.02
Stay_In_Current_City_Years 0.05 0.05 0.08 0.05 1.00 0.03 0.01 0.01 0.00
Marital_Status 0.01 0.34 0.18 0.06 0.03 1.00 0.02 0.04 0.02
Product_Category_1 0.12 0.05 0.03 0.03 0.01 0.02 1.00 0.39 0.33
Product_Category_2 0.14 0.04 0.03 0.03 0.01 0.04 0.39 1.00 0.43
Product_Category_3 0.11 0.04 0.03 0.02 0.00 0.02 0.33 0.43 1.00
Gender
Age
Occupation
City_Category
Stay_In_Current_City_Years
Marital_Status
Product_Category_1
Product_Category_2
Product_Category_3
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



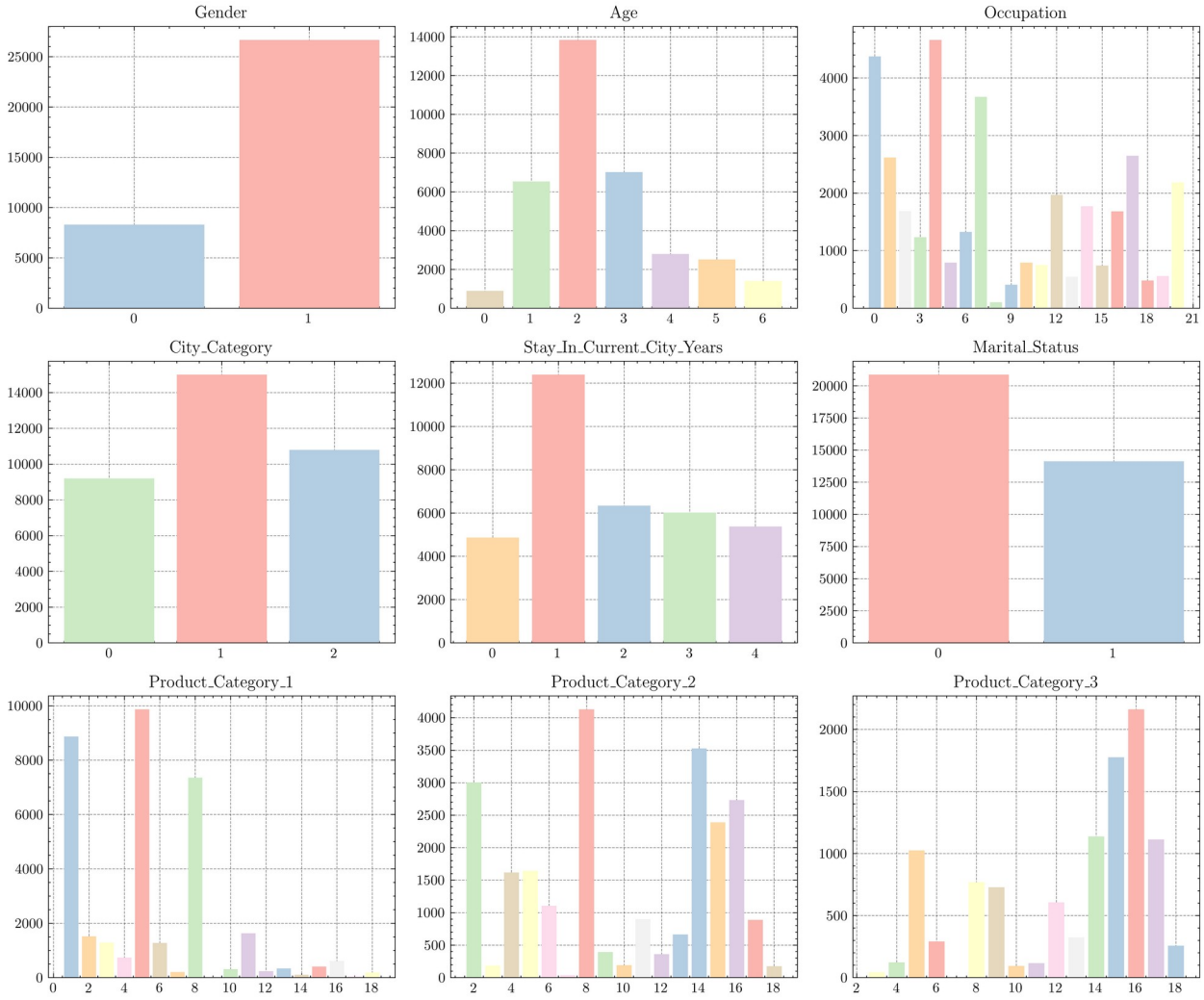
数据分布情况分析

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import matplotlib.cm as cm

data = train_data[discrete_columns].copy()

# 将类别型变量转换为数值型
le = LabelEncoder()
data.loc[:, 'Gender'] = le.fit_transform(data['Gender'])
data.loc[:, 'Age'] = le.fit_transform(data['Age'])
data.loc[:, 'City_Category'] = le.fit_transform(data['City_Category'])
```

```
data.loc[:, 'Stay_In_Current_City_Years'] =
data['Stay_In_Current_City_Years'].astype(str)
data.loc[:, 'Stay_In_Current_City_Years'] =
le.fit_transform(data['Stay_In_Current_City_Years'])

# 获取Pastel1颜色映射
pastel1_cmap = cm.get_cmap('Pastel1', 9)

# 创建一个包含所有子图的大图
fig, axs = plt.subplots(3, 3, figsize=(12, 10))
for i, col in enumerate(data.columns):
    # 确定子图的位置
    ax = axs[i // 3, i % 3]
    counts = data[col].value_counts()
    # 绘制条形图
    ax.bar(counts.index, counts.values, color=pastel1_cmap.colors)
    ax.set_title(col)
    # 设置x坐标为整数
    ax.xaxis.set_major_locator(plt.MaxNLocator(integer=True))

plt.tight_layout()
plt.show()
```

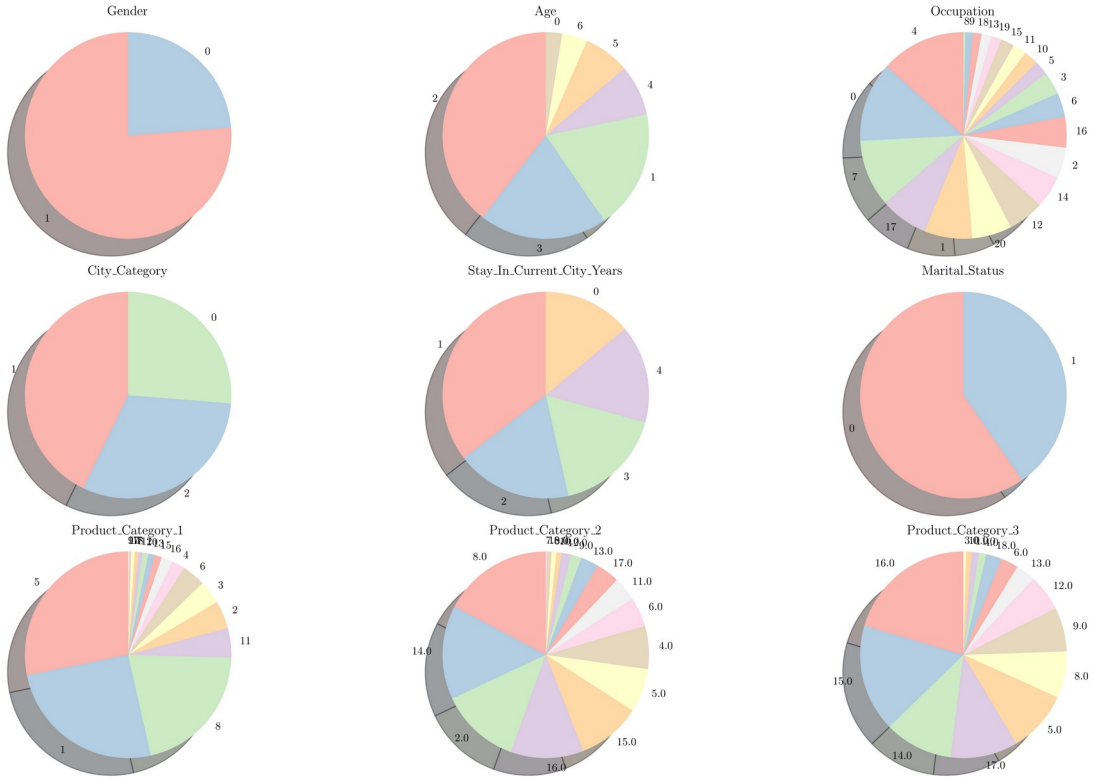
```
/tmp/ipykernel_15146/2182468980.py:10: DeprecationWarning: In a future
version, `df.iloc[:, i] = newvals` will attempt to set the values
inplace instead of always setting a new array. To retain the old
behavior, use either `df[df.columns[i]] = newvals` or, if columns are
non-unique, `df.isetitem(i, newvals)`
  data.loc[:, 'Gender'] = le.fit_transform(data['Gender'])
/tmp/ipykernel_15146/2182468980.py:11: DeprecationWarning: In a future
version, `df.iloc[:, i] = newvals` will attempt to set the values
inplace instead of always setting a new array. To retain the old
behavior, use either `df[df.columns[i]] = newvals` or, if columns are
non-unique, `df.isetitem(i, newvals)`
  data.loc[:, 'Age'] = le.fit_transform(data['Age'])
/tmp/ipykernel_15146/2182468980.py:12: DeprecationWarning: In a future
version, `df.iloc[:, i] = newvals` will attempt to set the values
inplace instead of always setting a new array. To retain the old
behavior, use either `df[df.columns[i]] = newvals` or, if columns are
non-unique, `df.isetitem(i, newvals)`
  data.loc[:, 'City_Category'] =
le.fit_transform(data['City_Category'])
/tmp/ipykernel_15146/2182468980.py:14: DeprecationWarning: In a future
version, `df.iloc[:, i] = newvals` will attempt to set the values
inplace instead of always setting a new array. To retain the old
behavior, use either `df[df.columns[i]] = newvals` or, if columns are
non-unique, `df.isetitem(i, newvals)`
  data.loc[:, 'Stay_In_Current_City_Years'] =
le.fit_transform(data['Stay_In_Current_City_Years'])
/tmp/ipykernel_15146/2182468980.py:17: MatplotlibDeprecationWarning:
```

```
The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  pastel1_cmap = cm.get_cmap('Pastel1', 9)
```




```
# 绘制饼图
fig, axs = plt.subplots(3, 3, figsize=(16, 10))
for i, col in enumerate(data.columns):
    # 确定子图的位置
    ax = axs[i // 3, i % 3]
    # 绘制饼图
    counts = data[col].value_counts()
    ax.pie(counts.values, labels=counts.index, autopct=None,
shadow=True, startangle=90, colors=pastel1_cmap.colors)
    ax.axis('equal')
    ax.set_title(col)
```

```
plt.tight_layout()
plt.show()
```



通过以上的条形图，我们可以对各个特征的分布情况进行如下分析：

- Gender 变量：该变量是表示性别的二分类变量，取值 F（女性）和 M（男性），由上面的条形图我们可知，该数据中包含的男用户的数据量远远高于女用户，性别为男性的用户数据大致占全部数据的四分之三，而性别为女性的用户数据大致占全部数据的四分之一。
- Age 变量：该变量表示用户年龄，由分布条形图可以看出，数据的用户年龄集中在 18-45 岁之间，其中 26-35 岁的用户最多，包含超过 200000 条数据。
- Occupation 变量：该变量表示用户职业，在原数据中，用数字 0-20 表示了 21 种不同的用户职业。由条形图可知，数据中用户的职业较为分散，其中，从事职业类别为 8、9、13、18 的用户数据较少，从事职业类别为 0、4、7 的的用户较多。
- City_Category 变量：该变量表示的是用户所居住的城市类别，分为 A、B、C 三个层次，由条形图可知，居住在 B类城市的用户数量较多，大致为 220000，其余居住在 A 类和 C 类城市的用户数量较为平均，大致为 150000。
- Stay_In_Current_City_Years 变量：该变量表示的是用户在本地城市居住年数，数值 0-3 分别表示用户在本地城市居住 0-3 年，数值 4 则表

示用户在本地城市居住 4 年及以上，通过条形图可知，在本地城市居住 1 年的用户数据最多，超过 175000 条，而其余居住年限的用户数量则分布较为平均。

- Marital_Status 变量：该变量表示用户的婚姻状态，其中 0 表示未婚，1 表示已婚，由条形图可知，数据中未婚用户数量较多，大约在 300000-350000 之间，而已婚用户数量较少，大约在 200000-250000 之间。
- Product_Category_1 变量：该变量表示产品分类 1，用数值 1-18 分别表示 18 类产品，通过条形图可知，类别为 7、9、12、14、17、18 的产品数据非常少，而数据集中分布在产品分类 1 为 1、5、8 的产品上。
- Product_Category_2 变量：该变量表示产品分类 2，用数值 2-18 分别表示 17 类产品，通过条形图可知，类别为 3、7、10、18 的产品数据非常少，而数据集中分布在产品分类 2 为 8 的产品上。
- Product_Category_3 变量：该变量表示产品分类 3，用数值 3-18 分别表示 16 类产品，通过条形图可知，产品分类 3 的数据分布非常极端，基本全部集中在类别为 16 的产品上，而其余类别的产品所包含数据量则非常少

更多数据分析结果请见"数据分析.html"文件

建立预测模型

我们将使用 XGBoost 模型进行预测。XGBoost 是一种优化的分布式梯度提升库，它能够快速准确地解决许多数据科学问题。

经过多个模型对比以及多次调整参数，最终我们构建的 XGBoost 模型，在全部训练集上取得了最佳的拟合效果。

XGBoost 是 Boosting 算法的其中一种。Boosting 是一种将许多弱分类器集成在一起形成一个强分类器的框架。集成学习的主要手段就是反复训练多个模型，并将这些模型通过一定方式组合在一起，形成一个高性能的强大的集成模型。在 Boosting 算法体系中，我们一般采用迭代串行的形式生成一系列模型，然后将这些模型进行线性加权相加，得到最终集成学习器。 假设已经迭代到 m-1 次，得到的集成模型为：

$$F_{m-1}(x)=\sum_{k=1}^{m-1}\alpha_k f_k(x)$$

那么在下一次迭代中，它应该让新生成的集成模型在训练集上损失最小：

$$(\alpha_m, f_m(x)) \quad \leftarrow \arg\min_{\alpha, f(x)} E_{y,X}\psi\left(y, F_{m-1}(x)+\alpha f(x)\right)$$

$$F_m(x) \qquad\qquad \leftarrow F_{m-1}(x)+\alpha_m f_m(x)$$

XGBoost 是基于梯度提升树(Gradient Boosting Decision Tree)改进而来的。在 GBDT 的基础上，XGBoost 通过最小化损失函数，结合控制模型复杂程度的正则化项，优化目标函数。其步骤如下：

1. 确定初始损失函数与正则项：

$$ J\left(f_{t}\right)=\sum_{i=1}^{n} L\left(y_{i} \hat{y}_{i}^{(t-1)}+f_{t}\left(x_{i}\right)\right)+\Omega\left(f_{t}\right)+C \\ \Omega\left(f_{t}\right)=\gamma^{T_{t}}+\frac{1}{2} \lambda \sum_{j=1}^{T} w_{j}^{2} $$

1. 生成第 t 棵树后，将目标函数改写为：

$$J(f_t)=\sum_{i=1}^{n} L\left(y_i \hat{y}_i^{(t-1)}+f_t(x_i)\right)+\Omega(f_t)+C$$

1. 对目标函数求一、二阶导数：

$$g_i=\frac{\partial L\left(y_i,\hat{y}_i^{(t-1)}\right)}{\partial \hat{y}_i^{(t-1)}}, h_i=\frac{\partial^2 L\left(y_i,\hat{y}_i^{(t-1)}\right)}{\partial \hat{y}_i^{(t-1)}}$$

1. 利用二阶 Taylor 展开式，将目标函数在$f_t=0$处近似：

$$J(f_t)\approx\sum_{i=1}^{n}\left[L\left(y_i,\hat{y}_i^{(t-1)}\right)+g_i f_t(x_i)+\frac{1}{2}h_i f_i^2(x_i)\right)+\Omega(f_t)+C$$

此时目标函数只依赖于每个数据点的在误差函数上的一阶导数和二阶导数，利于在训练过程中求解。

其他细节设计如下：

编码：在这里，我们选择使用 target coding 的方式对类别进行编码：

- 合并 Product_Category_1、Product_Category_2、Product_Category_3 为一列
- 对新生成的列进行编码，并计算其平均 Purchase 值

```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.7.6-py3-none-manylinux2014_x86_64.whl (200.3
MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.3/200.3 MB 5.4 MB/s eta
0:00:0000:0100:01
ent already satisfied: scipy in ./anaconda3/lib/python3.10/site-
packages (from xgboost) (1.10.0)
Requirement already satisfied: numpy in
./anaconda3/lib/python3.10/site-packages (from xgboost) (1.23.5)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.6
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

LabelEncoder()

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
from sklearn.preprocessing import LabelEncoder

# 读取数据
data = pd.read_sql('SELECT * FROM train LIMIT 500000', engine)

# 将类别型变量转换为数值型
le = LabelEncoder()
for col in data.columns[data.dtypes == 'object']:
    data[col] = le.fit_transform(data[col])

# 定义特征和目标变量
X = data.drop('Purchase', axis=1)
y = data['Purchase']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=123)

# 实例化模型
model = XGBRegressor()

# 训练模型
model.fit(X_train, y_train)

# 预测
y_pred = model.predict(X_test)

# 计算RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

rmse
```

```
2613.5933197685877
```

Target Encoding

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
```

```
# 定义列的数据类型
dtype_dict = {
    'User_ID': int,
    'Product_ID': str,
    'Gender': str,
    'Age': str,
    'Occupation': int,
    'City_Category': str,
    'Stay_In_Current_City_Years': str,
    'Marital_Status': int,
    'Product_Category_1': int,
    'Product_Category_2': int,
    'Product_Category_3': int,
    'Purchase': int
}

# 读取数据
data = pd.read_sql('SELECT * FROM train', engine)

# 处理NaN值，可以使用众数、平均数等方式，这里我选择用-1表示NaN
data = data.fillna(-1)

# 修改数据类型
for col, dtype in dtype_dict.items():
    data[col] = data[col].astype(dtype)


# 合并Product_Category_1、Product_Category_2、Product_Category_3为一个列
data['Product_Category_combined'] =
data['Product_Category_1'].astype(str) + '_' +
data['Product_Category_2'].astype(str) + '_' +
data['Product_Category_3'].astype(str)

# 删除原来的Product_Category_1、Product_Category_2、Product_Category_3
列
data = data.drop(['Product_Category_1', 'Product_Category_2',
'Product_Category_3'], axis=1)

# 指定分类变量列
categorical_cols = ['User_ID', 'Product_ID', 'Gender', 'Age',
'Occupation',
                    'City_Category', 'Stay_In_Current_City_Years',
'Marital_Status',
                    'Product_Category_combined']

# 创建存储各分类变量平均Purchase值的字典
mean_dict = {}
for col in categorical_cols:
    mean_dict[col] = data.groupby(col)['Purchase'].mean().to_dict()
```

```
# 对分类变量进行目标编码
for col in categorical_cols:
    data[col] = data[col].map(mean_dict[col])


# 定义特征和目标变量
X = data.drop('Purchase', axis=1)
y = data['Purchase']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=123)

# 实例化模型
model = XGBRegressor(seed=42)

# 网格搜索参数
# param_grid = {
#     "learning_rate": [0.01, 0.1, 0.2],
#     "max_depth": [3, 4, 5, 6],
#     "min_child_weight": [1, 2, 3, 4],
#     "subsample": [0.5, 0.6, 0.7, 0.8],
#     "colsample_bytree": [0.5, 0.6, 0.7, 0.8],
#     "n_estimators": [50, 100, 200],
#     "tree_method" : ['gpu_hist']
# }

# # 网格搜索参数
# param_grid = {
#     "learning_rate": [0.25,0.3],
#     "max_depth": [5, 6, 7],
#     "n_estimators": [300,400,500],
# }

# 网格搜索参数
param_grid = {
    "learning_rate": [0.25],
    "max_depth": [6],
    "n_estimators": [400],
}

# 网格搜索
grid_search = GridSearchCV(
    model,
    param_grid,
    scoring="r2",
    n_jobs=-1,
    cv=5,
    verbose=100,
```

```
)


grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters found: ", grid_search.best_params_)
print("Best score found: ", grid_search.best_score_)

# 使用最佳参数训练XGBoost模型，并命名为xgb_regressor
best_xgb_regressor = grid_search.best_estimator_
best_xgb_regressor.fit(X_train, y_train)

# 使用XGBoost模型预测
y_pred_xgb = best_xgb_regressor.predict(X_test)

# 计算RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_xgb))

rmse
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best parameters found:  {'learning_rate': 0.25, 'max_depth': 6,
'n_estimators': 400}
Best score found:  0.7530897523110764
```

```
2461.285260891015
```

```
# 读取数据
data = pd.read_sql('SELECT * FROM train', engine)
data.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	\
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	

	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	\
0	2	0	3	
1	2	0	1	
2	2	0	12	
3	2	0	12	
4	4+	0	8	

	Product_Category_2	Product_Category_3	Purchase
0	NaN	NaN	8370
1	6.0	14.0	15200
2	NaN	NaN	1422

```
3                14.0                  NaN      1057
4                 NaN                  NaN      7969
```

```
import pickle

# 将字典保存为 pickle 文件
with open('mean_dict.pickle', 'wb') as handle:
    pickle.dump(mean_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)

# 从 pickle 文件中读取字典
with open('mean_dict.pickle', 'rb') as handle:
    mean_dict = pickle.load(handle)

X.head()
```

```
        User_ID    Product_ID       Gender          Age
Occupation  \
0   9808.264706  11857.941441  8810.345066  9019.346394  9051.455474

1   9808.264706  16303.456294  8810.345066  9019.346394  9051.455474

2   9808.264706   1240.323232  8810.345066  9019.346394  9051.455474

3   9808.264706   1454.871642  8810.345066  9019.346394  9051.455474

4  10662.539474   7718.695000  9504.475041  9451.831428  9459.188627


   City_Category  Stay_In_Current_City_Years  Marital_Status  \
0    8958.195552                 9397.345338      9333.80102
1    8958.195552                 9397.345338      9333.80102
2    8958.195552                 9397.345338      9333.80102
3    8958.195552                 9397.345338      9333.80102
4    9844.015620                 9346.043294      9333.80102

   Product_Category_combined
0               11768.337607
1               15452.130682
2                1334.127356
3                1375.287766
4                7512.551869
```

调参记录

第 1 次调参

参数设置

```
param_grid = {
    "learning_rate": [0.2, 0.25],
```

```
    "max_depth": [4, 5],
    "n_estimators": [200, 250, 300],
}
```

调参过程

执行命令：

```
grid_search = GridSearchCV(
    model,
    param_grid,
    scoring="r2",
    n_jobs=-1,
    cv=5,
    verbose=1,
)

grid_search.fit(X_train, y_train)
```

调参结果

- 最佳参数：`{'learning_rate': 0.25, 'max_depth': 5, 'n_estimators': 300}`
- 最佳分数：0.7513632065238093

第 2 次调参

参数设置

```
param_grid = {
    "learning_rate": [0.25, 0.3],
    "max_depth": [5, 6, 7],
    "n_estimators": [300, 400, 500],
}
```

调参过程

执行命令：

```
grid_search = GridSearchCV(
    model,
    param_grid,
    scoring="r2",
    n_jobs=-1,
    cv=5,
    verbose=1,
)

grid_search.fit(X_train, y_train)
```

调参结果

- 最佳参数：`{'learning_rate': 0.25, 'max_depth': 6, 'n_estimators': 400}`
- 最佳分数：0.7530897523110764

Ridge 回归

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge

# 实例化模型
model = Ridge(alpha=1)

# 训练模型
model.fit(X_train, y_train)

# 进行预测
y_pred_linear = model.predict(X_test)

# 计算R^2值
r2 = r2_score(y_test, y_pred_linear)
r2
```

```
0.735168193796518
```

Target Encoding + 交互项

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor

# 读取数据
data = pd.read_sql('SELECT * FROM train', engine)

# 处理NaN值，可以使用众数、平均数等方式，这里我选择用-1表示NaN
data = data.fillna(-1)
# 指定分类变量列
categorical_cols = ['User_ID', 'Product_ID', 'Gender', 'Age',
'Occupation',
                    'City_Category', 'Stay_In_Current_City_Years',
'Marital_Status',
                    'Product_Category_1', 'Product_Category_2',
'Product_Category_3']
```

```
# 创建存储各分类变量平均 Purchase 值的字典
mean_dict = {}
for col in categorical_cols:
    mean_dict[col] = data.groupby(col)['Purchase'].mean().to_dict()

# 对分类变量进行目标编码
for col in categorical_cols:
    data[col] = data[col].map(mean_dict[col])

# 合并 Product_Category_1、Product_Category_2、Product_Category_3 为一个列
data['Product_Category_combined'] =
data['Product_Category_1'].astype(str) + '_' +
data['Product_Category_2'].astype(str) + '_' +
data['Product_Category_3'].astype(str)

# 对新生成的列进行编码，并计算其平均 Purchase 值
le = LabelEncoder()
data['Product_Category_combined'] =
le.fit_transform(data['Product_Category_combined'])
mean_dict['Product_Category_combined'] =
data.groupby('Product_Category_combined')['Purchase'].mean().to_dict()
data['Product_Category_combined'] =
data['Product_Category_combined'].map(mean_dict['Product_Category_comb
ined'])

# 删除原来的 Product_Category_1、Product_Category_2、Product_Category_3
列
data = data.drop(['Product_Category_1', 'Product_Category_2',
'Product_Category_3'], axis=1)

# 获取所有的特征列
feature_cols = data.columns.tolist()
feature_cols.remove('Purchase')

# 对每一对特征，计算它们的交互项
for i in range(len(feature_cols)):
    for j in range(i+1, len(feature_cols)):
        # 创建新的交互项列
        data[feature_cols[i]+'_'+feature_cols[j]] =
data[feature_cols[i]] * data[feature_cols[j]]




# 定义特征和目标变量
X = data.drop('Purchase', axis=1)
y = data['Purchase']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=123)

# 实例化模型
model = XGBRegressor(seed=42)

# 网格搜索参数
# param_grid = {
#     "learning_rate": [0.01, 0.1, 0.2],
#     "max_depth": [3, 4, 5, 6],
#     "min_child_weight": [1, 2, 3, 4],
#     "subsample": [0.5, 0.6, 0.7, 0.8],
#     "colsample_bytree": [0.5, 0.6, 0.7, 0.8],
#     "n_estimators": [50, 100, 200],
#     "tree_method" : ['gpu_hist']
# }

# 网格搜索参数
param_grid = {
    "learning_rate": [0.25],
    "max_depth": [6],
    "n_estimators": [400],
}

# 网格搜索
grid_search = GridSearchCV(
    model,
    param_grid,
    scoring="r2",
    n_jobs=-1,
    cv=5,
    verbose=100,
)


grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best parameters found: ", grid_search.best_params_)
print("Best score found: ", grid_search.best_score_)

# 使用最佳参数训练模型
best_xgb_clf = grid_search.best_estimator_
best_xgb_clf.fit(X_train, y_train)

# 预测
y_pred = best_xgb_clf.predict(X_test)

# 计算RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

```
rmse

Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 5/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 5/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.749 total time= 6.6min
[CV 2/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 2/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.753 total time= 6.3min
[CV 4/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 4/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.753 total time= 7.1min
[CV 1/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 1/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.751 total time= 6.3min


KeyboardInterrupt
```

效果提升不大，时间消耗太大。遂放弃。

神经网络

```
!pip install tensorflow

Collecting tensorflow
  Downloading tensorflow-2.13.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (524.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 524.1/524.1 MB 5.1 MB/s eta
0:00:0000:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.5/126.5 kB 2.3 MB/s eta
0:00:00a 0:00:01
ent already satisfied: numpy<=1.24.3,>=1.22 in
./anaconda3/lib/python3.10/site-packages (from tensorflow) (1.23.5)
Requirement already satisfied: setuptools in
./anaconda3/lib/python3.10/site-packages (from tensorflow) (65.6.3)
Collecting grpcio<2.0,>=1.24.3
  Downloading grpcio-1.56.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.2 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.2/5.2 MB 8.7 MB/s eta
0:00:00:00:0100:01m
ator<2.14,>=2.13.0
  Downloading tensorflow_estimator-2.13.0-py2.py3-none-any.whl (440
kB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 440.8/440.8 kB 4.2 MB/s eta
0:00:0000:01
color>=1.1.0
  Downloading termcolor-2.3.0-py3-none-any.whl (6.9 kB)
Collecting flatbuffers>=23.1.21
  Downloading flatbuffers-23.5.26-py2.py3-none-any.whl (26 kB)
Collecting protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!
=4.21.5,<5.0.0dev,>=3.20.3
  Downloading protobuf-4.23.3-cp37-abi3-manylinux2014_x86_64.whl (304
kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 304.5/304.5 kB 3.1 MB/s eta
0:00:0000:01
ent already satisfied: typing-extensions<4.6.0,>=3.6.6 in
./anaconda3/lib/python3.10/site-packages (from tensorflow) (4.4.0)
Collecting tensorflow-io-gcs-filesystem>=0.23.1
  Downloading tensorflow_io_gcs_filesystem-0.32.0-cp310-cp310-
manylinux_2_12_x86_64.manylinux2010_x86_64.whl (2.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.4/2.4 MB 5.1 MB/s eta
0:00:000:00:01
ent already satisfied: six>=1.12.0 in ./anaconda3/lib/python3.10/site-
packages (from tensorflow) (1.16.0)
Collecting google-pasta>=0.1.1
  Downloading google_pasta-0.2.0-py3-none-any.whl (57 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.5/57.5 kB 1.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 4.9 MB/s eta
0:00:000:00:01
>=2.3.2
  Downloading opt_einsum-3.3.0-py3-none-any.whl (65 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.5/65.5 kB 1.9 MB/s eta
0:00:00
anylinux2010_x86_64.whl (22.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.9/22.9 MB 6.5 MB/s eta
0:00:00:00:0100:01
ent already satisfied: h5py>=2.9.0 in ./anaconda3/lib/python3.10/site-
packages (from tensorflow) (3.7.0)
Requirement already satisfied: packaging in
./anaconda3/lib/python3.10/site-packages (from tensorflow) (22.0)
Requirement already satisfied: wrapt>=1.11.0 in
./anaconda3/lib/python3.10/site-packages (from tensorflow) (1.14.1)
Collecting tensorboard<2.14,>=2.13
  Downloading tensorboard-2.13.0-py3-none-any.whl (5.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.6/5.6 MB 6.6 MB/s eta
0:00:00:00:0100:01m
ent already satisfied: wheel<1.0,>=0.23.0 in
./anaconda3/lib/python3.10/site-packages (from astunparse>=1.6.0-
>tensorflow) (0.38.4)
Collecting google-auth<3,>=1.6.3
  Downloading google_auth-2.21.0-py2.py3-none-any.whl (182 kB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 182.1/182.1 kB 3.3 MB/s eta
0:00:00a 0:00:01
ent already satisfied: werkzeug>=1.0.1 in
./anaconda3/lib/python3.10/site-packages (from
tensorboard<2.14,>=2.13->tensorflow) (2.2.2)
Collecting tensorboard-data-server<0.8.0,>=0.7.0
  Downloading tensorboard_data_server-0.7.1-py3-none-
manylinux2014_x86_64.whl (6.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.6/6.6 MB 6.2 MB/s eta
0:00:00:00:0100:01
ent already satisfied: requests<3,>=2.21.0 in
./anaconda3/lib/python3.10/site-packages (from
tensorboard<2.14,>=2.13->tensorflow) (2.28.1)
Requirement already satisfied: markdown>=2.6.8 in
./anaconda3/lib/python3.10/site-packages (from
tensorboard<2.14,>=2.13->tensorflow) (3.4.1)
Collecting rsa<5,>=3.1.4
  Downloading rsa-4.9-py3-none-any.whl (34 kB)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
./anaconda3/lib/python3.10/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow) (0.2.8)
Requirement already satisfied: urllib3<2.0 in
./anaconda3/lib/python3.10/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow) (1.26.14)
Collecting cachetools<6.0,>=2.0.0
  Downloading cachetools-5.3.1-py3-none-any.whl (9.3 kB)
Collecting requests-oauthlib>=0.7.0
  Downloading requests_oauthlib-1.3.1-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: certifi>=2017.4.17 in
./anaconda3/lib/python3.10/site-packages (from requests<3,>=2.21.0-
>tensorboard<2.14,>=2.13->tensorflow) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in
./anaconda3/lib/python3.10/site-packages (from requests<3,>=2.21.0-
>tensorboard<2.14,>=2.13->tensorflow) (3.4)
Requirement already satisfied: charset-normalizer<3,>=2 in
./anaconda3/lib/python3.10/site-packages (from requests<3,>=2.21.0-
>tensorboard<2.14,>=2.13->tensorflow) (2.0.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in
./anaconda3/lib/python3.10/site-packages (from werkzeug>=1.0.1-
>tensorboard<2.14,>=2.13->tensorflow) (2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
./anaconda3/lib/python3.10/site-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow) (0.4.8)
Collecting oauthlib>=3.0.0
  Downloading oauthlib-3.2.2-py3-none-any.whl (151 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 151.7/151.7 kB 2.8 MB/s eta
0:00:00a 0:00:01
color, tensorflow-io-gcs-filesystem, tensorflow-estimator,
tensorboard-data-server, rsa, protobuf, opt-einsum, oauthlib, keras,
```

```
grpcio, google-pasta, gast, cachetools, astunparse, absl-py, requests-oauthlib, google-auth, google-auth-oauthlib, tensorboard, tensorflow
Successfully installed absl-py-1.4.0 astunparse-1.6.3 cachetools-5.3.1 flatbuffers-23.5.26 gast-0.4.0 google-auth-2.21.0 google-auth-oauthlib-1.0.0 google-pasta-0.2.0 grpcio-1.56.0 keras-2.13.1 libclang-16.0.0 oauthlib-3.2.2 opt-einsum-3.3.0 protobuf-4.23.3 requests-oauthlib-1.3.1 rsa-4.9 tensorboard-2.13.0 tensorboard-data-server-0.7.1 tensorflow-2.13.0 tensorflow-estimator-2.13.0 tensorflow-io-gcs-filesystem-0.32.0 termcolor-2.3.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv

!nvidia-smi

Thu Jul  6 23:09:40 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.182.03   Driver Version: 470.182.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-PCI...  On   | 00000000:00:05.0 Off |                    0 |
| N/A   29C    P0    36W / 250W |      3MiB / 40536MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-PCI...  On   | 00000000:00:06.0 Off |                    0 |
| N/A   30C    P0    34W / 250W |      3MiB / 40536MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

```
+-----------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  GPU
Memory |
|        ID   ID
Usage      |
|
=============================================================================
=======|
|  No running processes found
|
+-----------------------------------------------------------------------------
--------+
```

```python
import torch
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import torch.nn as nn
import torch.optim as optim
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.metrics import r2_score

import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor

# 定义列的数据类型
dtype_dict = {
    'User_ID': int,
    'Product_ID': str,
    'Gender': str,
    'Age': str,
    'Occupation': int,
    'City_Category': str,
    'Stay_In_Current_City_Years': str,
```

```
    'Marital_Status': int,
    'Product_Category_1': int,
    'Product_Category_2': int,
    'Product_Category_3': int,
    'Purchase': int
}

# 读取数据
data = pd.read_sql('SELECT * FROM train', engine)

# 处理NaN值，可以使用众数、平均数等方式，这里我选择用-1表示NaN
data = data.fillna(-1)

# 修改数据类型
for col, dtype in dtype_dict.items():
    data[col] = data[col].astype(dtype)

# 合并Product_Category_1、Product_Category_2、Product_Category_3为一个列
data['Product_Category_combined'] =
data['Product_Category_1'].astype(str) + '_' +
data['Product_Category_2'].astype(str) + '_' +
data['Product_Category_3'].astype(str)

# 删除原来的Product_Category_1、Product_Category_2、Product_Category_3
列
data = data.drop(['Product_Category_1', 'Product_Category_2',
'Product_Category_3'], axis=1)

# 指定分类变量列
categorical_cols = ['User_ID', 'Product_ID', 'Gender', 'Age',
'Occupation',
                    'City_Category', 'Stay_In_Current_City_Years',
'Marital_Status',
                    'Product_Category_combined']

# 创建存储各分类变量平均Purchase值的字典
mean_dict = {}
for col in categorical_cols:
    mean_dict[col] = data.groupby(col)['Purchase'].mean().to_dict()

# 对分类变量进行目标编码
for col in categorical_cols:
    data[col] = data[col].map(mean_dict[col])


# 定义特征和目标变量
X = data.drop('Purchase', axis=1)
y = data['Purchase']

# 划分训练集和测试集
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=123)

# 标准化特征变量
scaler_X = StandardScaler().fit(X_train)
X_train_scaled = torch.tensor(scaler_X.transform(X_train),
dtype=torch.float).cuda()
X_test_scaled = torch.tensor(scaler_X.transform(X_test),
dtype=torch.float).cuda()

# 标准化目标变量
scaler_y = StandardScaler().fit(y_train.values.reshape(-1, 1))
y_train_scaled =
torch.tensor(scaler_y.transform(y_train.values.reshape(-1, 1)),
dtype=torch.float).cuda()
y_test_scaled =
torch.tensor(scaler_y.transform(y_test.values.reshape(-1, 1)),
dtype=torch.float).cuda()

# 定义神经网络结构
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(X_train_scaled.shape[1], 128)  # 第一层隐藏
层，128 个神经元
        self.fc2 = nn.Linear(128, 64)  # 第二层隐藏层，64 个神经元
        self.fc3 = nn.Linear(64, 1)  # 输出层

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(X_train_scaled.shape[1], 256)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(256, 128)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(128, 64)
        self.dropout3 = nn.Dropout(0.5)
        self.fc4 = nn.Linear(64, 32)
        self.dropout4 = nn.Dropout(0.5)
        self.fc5 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout1(x)
        x = torch.relu(self.fc2(x))
```

```
        x = self.dropout2(x)
        x = torch.relu(self.fc3(x))
        x = self.dropout3(x)
        x = torch.relu(self.fc4(x))
        x = self.dropout4(x)
        return self.fc5(x)

# 创建神经网络模型并将其命名为 neural_net_model
neural_net_model = Net()
neural_net_model = neural_net_model.cuda()

# 定义优化器
optimizer = optim.Adam(neural_net_model.parameters())

# 训练模型
for epoch in range(10000):
    optimizer.zero_grad()
    output = neural_net_model(X_train_scaled)
    loss = torch.sum((output - y_train_scaled.view_as(output))**2) /
X_train_scaled.size(0)  # 使用 R^2 作为损失函数
    loss.backward()
    optimizer.step()

# 预测
neural_net_model.eval()
with torch.no_grad():
    y_pred_scaled = neural_net_model(X_test_scaled).cpu().numpy()

# 反标准化预测值
y_pred_neural = scaler_y.inverse_transform(y_pred_scaled)

# 计算模型的 RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_neural))
print(f'Test RMSE: {rmse}')

# 计算 R^2 值
r2 = r2_score(y_test, y_pred_neural)

print("R^2:", r2)
```

```
Test RMSE: 2496.1952754161875
R^2: 0.74850267878286
```

应用预测模型

我们将使用 XGBoost 模型进行预测 test 数据集。

```
import pandas as pd
from sqlalchemy import create_engine
```

```
test_data_orginal = pd.read_sql('SELECT * FROM test',
engine).iloc[:, :-1]
test_data = pd.read_sql('SELECT * FROM test', engine).iloc[:, :-1]

# 处理NaN值
test_data = test_data.fillna(-1)

# 合并Product_Category_1、Product_Category_2、Product_Category_3为一个列
test_data['Product_Category_combined'] =
test_data['Product_Category_1'].astype(str) + '_' +
test_data['Product_Category_2'].astype(str) + '_' +
test_data['Product_Category_3'].astype(str)

# 删除原来的Product_Category_1、Product_Category_2、Product_Category_3
列
test_data = test_data.drop(['Product_Category_1',
'Product_Category_2', 'Product_Category_3'], axis=1)

# 对分类变量进行目标编码
for col in categorical_cols:
    # 用全局平均值填充新的分类变量
    test_data[col] = test_data[col].map(lambda x:
mean_dict[col].get(x, np.mean(list(mean_dict[col].values()))))

# 使用训练好的模型进行预测
test_data_orginal['Purchase'] = best_xgb_regressor.predict(test_data)

test_data_orginal.head()
# 将预测结果写回到MySQL数据库
# test_data.to_sql('test', engine, if_exists='replace', index=False)
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	\
0	1005001	P00113142	F	26-35	2	B	
1	1005002	P00001042	M	26-35	17	B	
2	1005005	P00057942	M	46-50	16	B	
3	1005008	P00057942	M	26-35	20	B	
4	1005011	P00015542	M	18-25	4	B	

	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	\
0	1	0	1	
1	1	1	1	
2	2	0	1	
3	1	1	1	
4	1	0	1	

	Product_Category_2	Product_Category_3	Purchase
0	5	12	14958.499023
1	2	16	13481.602539
2	2	6	13594.751953

```
3                    2                    6  13712.898438
4                    2                   13   9909.993164
```

```
data = {
    "User_ID": [1005002, 1005001, 1005005],
    "Product_ID": ["P00113142", "P00001042", "P00057942"],
    "Gender": ['M', 'F', 'M'],
    "Age": ['26-35', '26-35', '46-50'],
    "Occupation": [2, 17, 16],
    "City_Category": ['B', 'B', 'B'],
    "Stay_In_Current_City_Years": ['1', '1', '2'],
    "Marital_Status": [0, 1, 0],
    "Product_Category_1": [1, 1, 1],
    "Product_Category_2": [5, 2, 2],
    "Product_Category_3": [12, 16, 6]
}

# 将数据转换为 pandas DataFrame
data = pd.DataFrame(data)
# 从 pickle 文件中读取字典
with open('mean_dict.pickle', 'rb') as handle:
    mean_dict = pickle.load(handle)
# 处理 NaN 值，可以使用众数、平均数等方式，这里我选择用 -1 表示 NaN
data = data.fillna(-1)

# 合并 Product_Category_1、Product_Category_2、Product_Category_3 为一个列
data['Product_Category_combined'] =
data['Product_Category_1'].astype(str) + '_' + data[
    'Product_Category_2'].astype(str) + '_' +
data['Product_Category_3'].astype(str)

# 删除原来的 Product_Category_1、Product_Category_2、Product_Category_3
列
data = data.drop(['Product_Category_1', 'Product_Category_2',
'Product_Category_3'], axis=1)

# 指定分类变量列
categorical_cols = ['User_ID', 'Product_ID', 'Gender', 'Age',
'Occupation',
                    'City_Category', 'Stay_In_Current_City_Years',
'Marital_Status',
                    'Product_Category_combined']

# 对分类变量进行目标编码
for col in categorical_cols:
    data[col] = data[col].map(mean_dict[col])
print(data)
# 使用模型进行预测
prediction = best_xgb_regressor.predict(xgb.DMatrix(data))
prediction
```

```
       User_ID    Product_ID       Gender          Age   Occupation  \
0  9509.976744  15234.938420  9504.475041  9314.932818  9017.601130
1  8522.172727  13745.679435  8810.345066  9314.932818  9907.916594
2  7924.031111  14127.864516  9504.475041  9285.327474  9459.188627

   City_Category  Stay_In_Current_City_Years  Marital_Status  \
0    9198.350923                 9318.819305     9333.801020
1    9198.350923                 9318.819305     9333.578539
2    9198.350923                 9397.345338     9333.801020

   Product_Category_combined
0                        NaN
1                        NaN
2                        NaN

array([15175.287, 13100.292, 14654.364], dtype=float32)

mean_dict
```

应用集成模型

我们将使用 XGBoost 模型和神经网络模型融合进行预测 test 数据集。

```
y_pred_neural

array([[ 6076.4756],
       [16354.056 ],
       [ 8418.236 ],
       ...,
       [ 5199.623 ],
       [ 8317.617 ],
       [16941.59  ]], dtype=float32)

y_pred_xgb

array([ 6648.1567, 15706.401 ,  7679.3604, ...,  5752.555 ,
7976.618 ,
       18538.61  ], dtype=float32)

import pandas as pd
from sqlalchemy import create_engine

# 加载原始测试数据和要进行预测的测试数据
test_data_orginal = pd.read_sql('SELECT * FROM test',
engine).iloc[:, :-1]
test_data = test_data_orginal.copy()

# 处理 NaN 值
test_data = test_data.fillna(-1)
```

```
# 合并Product_Category_1、Product_Category_2、Product_Category_3为一个列
test_data['Product_Category_combined'] =
test_data['Product_Category_1'].astype(str) + '_' +
test_data['Product_Category_2'].astype(str) + '_' +
test_data['Product_Category_3'].astype(str)

# 删除原来的Product_Category_1、Product_Category_2、Product_Category_3
列
test_data = test_data.drop(['Product_Category_1',
'Product_Category_2', 'Product_Category_3'], axis=1)

# 对分类变量进行目标编码
for col in categorical_cols:
    # 用全局平均值填充新的分类变量
    test_data[col] = test_data[col].map(lambda x:
mean_dict[col].get(x, np.mean(list(mean_dict[col].values()))))

# 使用XGBoost模型进行预测
y_pred_xgb = best_xgb_regressor.predict(test_data)

# # 将测试集特征标准化，注意使用训练集的标准化参数
# test_data_scaled = torch.tensor(scaler_X.transform(test_data),
dtype=torch.float).cuda()

# # 使用神经网络模型进行预测
# neural_net_model.eval()
# with torch.no_grad():
#     y_pred_scaled_neural =
neural_net_model(test_data_scaled).cpu().numpy()

# # 反标准化预测值
# y_pred_neural = scaler_y.inverse_transform(y_pred_scaled_neural)

# # 将两个模型的预测结果合并为一个新的特征
# ensemble_features = np.vstack((np.squeeze(y_pred_neural),
y_pred_xgb)).T


# # 使用元模型对测试集进行预测
# y_pred_avg = meta_model.predict(ensemble_features)


# 将预测结果添加到原始测试数据集
test_data_orginal['Purchase'] = y_pred_xgb

# 将预测结果保存为Excel文件
test_data_orginal.to_excel('prediction.xlsx', index=False)


test_data_orginal.head()
```

```
# 将预测结果写回到MySQL数据库
# test_data.to_sql('test', engine, if_exists='replace', index=False)
```

```
   User_ID Product_ID Gender    Age  Occupation City_Category  \
0  1005001  P00113142      F  26-35           2             B
1  1005002  P00001042      M  26-35          17             B
2  1005005  P00057942      M  46-50          16             B
3  1005008  P00057942      M  26-35          20             B
4  1005011  P00015542      M  18-25           4             B

  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  \
0                          1               0                   1
1                          1               1                   1
2                          2               0                   1
3                          1               1                   1
4                          1               0                   1

   Product_Category_2  Product_Category_3      Purchase
0                   5                  12  14958.499023
1                   2                  16  13481.602539
2                   2                   6  13594.751953
3                   2                   6  13712.898438
4                   2                  13   9909.993164
```

保存模型

```
# 保存模型
best_xgb_regressor.save_model('xgboost_model.json')

# 加载模型
best_xgb_regressor = xgb.Booster()
best_xgb_regressor.load_model('xgboost_model.json')
```

```
[CV 1/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 1/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.752 total time= 1.4min
[CV 4/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 4/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.755 total time= 1.4min
[CV 5/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 5/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.750 total time= 1.4min
[CV 2/5; 1/1] START learning_rate=0.25, max_depth=6,
n_estimators=400.....
[CV 2/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.755 total time= 1.4min
[CV 3/5; 1/1] START learning_rate=0.25, max_depth=6,
```

```
n_estimators=400.....
[CV 3/5; 1/1] END learning_rate=0.25, max_depth=6, n_estimators=400;,
score=0.754 total time= 1.5min
```

API 设计

这个文档记录了在 Ubuntu 服务器上部署应用程序的步骤。主要步骤包括在 Ubuntu 18.04 上安装 MySQL，然后在 Ubuntu 20.04 上安装 Nginx，修改配置文件，配置 Gunicorn 的代理，配置 Flask，通过端口转发把 Nginx 的 8000 端口交给 Gunicorn，最后通过 Gunicorn 启动 Flask 应用程序。

在 Ubuntu 20.04 云服务器上安装 Nginx 并修改配置文件

1. 更新系统包列表：

```
sudo apt-get update
```

1. 安装 Nginx：

```
sudo apt-get install nginx
```

1. 启动 Nginx：

```
sudo systemctl start nginx
```

1. 确保 Nginx 在启动时自动运行：

```
sudo systemctl enable nginx
```

1. 修改 Nginx 配置文件：

```
sudo nano /etc/nginx/sites-available/default
```

在这个文件中，你可以设置服务器的监听端口，服务器的根目录，以及其他的服务器配置。

配置 Gunicorn 的代理

1. 安装 Gunicorn：

```
pip install gunicorn
```

1. 在 Nginx 配置文件中设置 Gunicorn 代理：

```
sudo nano /etc/nginx/sites-available/default
```

在这个文件中，你需要设置一个新的 location 块，使得 Nginx 可以将请求代理到 Gunicorn。例如：

```
location / {
    proxy_pass http://localhost:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

这个配置将 Nginx 的 8000 端口的请求代理到 Gunicorn。

配置 Flask 并启动应用程序

1. 安装 Flask：

```
pip install flask
```

1. 创建一个新的 Flask 应用程序。创建一个新的 Python 文件

```
from flask import Flask, request, jsonify, render_template_string
import pickle
import pandas as pd
import xgboost as xgb

# 加载模型
model = xgb.Booster()
model.load_model('xgboost_model.json')
# 从pickle文件中读取字典
with open('mean_dict.pickle', 'rb') as handle:
    mean_dict = pickle.load(handle)
# 定义列的数据类型
dtype_dict = {
    'User_ID': int,
    'Product_ID': str,
    'Gender': str,
    'Age': str,
    'Occupation': int,
    'City_Category': str,
    'Stay_In_Current_City_Years': str,
    'Marital_Status': int,
    'Product_Category_1': int,
    'Product_Category_2': int,
    'Product_Category_3': int,
}
# print(mean_dict)

# 设置显示选项
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
app = Flask(__name__)


@app.route('/predict', methods=['POST'])
def predict():
    # 获取请求中的数据
    data = request.get_json(force=True)

    # 将数据转换为 pandas DataFrame
    data = pd.DataFrame(data)

    # 处理 NaN 值，可以使用众数、平均数等方式，这里我选择用 -1 表示 NaN
    data = data.fillna(-1)

    # 修改数据类型
    for col, dtype in dtype_dict.items():
        data[col] = data[col].astype(dtype)

    # 合并 Product_Category_1、Product_Category_2、Product_Category_3 为一
个列
    data['Product_Category_combined'] =
data['Product_Category_1'].astype(str) + '_' + data[
        'Product_Category_2'].astype(str) + '_' +
data['Product_Category_3'].astype(str)

    # 删除原来的
Product_Category_1、Product_Category_2、Product_Category_3 列
    data = data.drop(['Product_Category_1', 'Product_Category_2',
'Product_Category_3'], axis=1)

    # 指定分类变量列
    categorical_cols = ['User_ID', 'Product_ID', 'Gender', 'Age',
'Occupation',
                        'City_Category', 'Stay_In_Current_City_Years',
'Marital_Status',
                        'Product_Category_combined']
    # print(data)
    # 对分类变量进行目标编码
    for col in categorical_cols:
        data[col] = data[col].map(mean_dict[col])
    # print(data)
    # 使用模型进行预测
    prediction = model.predict(xgb.DMatrix(data))

    # 返回预测结果和 RMSE
    return jsonify({'prediction': prediction.tolist(), 'rmse':
2461.28, 'R^2':  0.7530})


@app.route('/predict')
```

```
def home():
    return render_template_string("""
    <!DOCTYPE html>
    <html>
    <head>
        <title>销售数据分析与预测</title>
        <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap
.min.css" rel="stylesheet">
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"
></script>
        <script>
        $(document).ready(function(){
            $('#predict-form').on('submit', function(e) {
                e.preventDefault();

                let data = {
                    "User_ID": [$('#User_ID').val() ? parseInt($
('#User_ID').val()) : null],
                    "Product_ID": [$('#Product_ID').val()],
                    "Gender": [$('#Gender').val()],
                    "Age": [$('#Age').val()],
                    "Occupation": [$('#Occupation').val() ? parseInt($
('#Occupation').val()) : null],
                    "City_Category": [$('#City_Category').val()],
                    "Stay_In_Current_City_Years": [$
('#Stay_In_Current_City_Years').val()],
                    "Marital_Status": [$('#Marital_Status').val() ?
parseInt($('#Marital_Status').val()) : null],
                    "Product_Category_1": [$
('#Product_Category_1').val() ? parseInt($
('#Product_Category_1').val()) : null],
                    "Product_Category_2": [$
('#Product_Category_2').val() ? parseInt($
('#Product_Category_2').val()) : null],
                    "Product_Category_3": [$
('#Product_Category_3').val() ? parseInt($
('#Product_Category_3').val()) : null]
                };



$.ajax({
    url: '/predict',
    method: 'POST',
    contentType: 'application/json',
    data: JSON.stringify(data),
    success: function(response) {
        $('#result').html('Prediction: ' + response['prediction'] + ',
```

```
RMSE: ' + response['rmse']  + ', R^2: ' + response['R^2']);
    }
});

            });
        });
        </script>
    </head>
    <body>
        <div class="container mt-5">
    <h1 class="text-center">哈工大-华为数据领域创新实践课合作项目:销售数据分析与预测</h1>
            <div class="text-center">
            <h4>课题周期: 2023-07-04 ~ 2023-07-08</p>
             <p>企业导师：伍国栋、王一方</p>
            <p>课题小组: 第三组</p>
            <p>团队成员:李岳锴、李墨、王以柔、常雨欣、梅洁楠、林瑞奇、李毓晟、谭启泰、潘诚</p>
        </div>
 <form id="predict-form" class="mt-5">
    <div class="form-group">
        <label for="User_ID">User ID (例如: 1005001)</label>
        <input type="number" class="form-control" id="User_ID" value="1005001">
    </div>
    <div class="form-group">
        <label for="Product_ID">Product ID (例如: P00113142)</label>
        <input type="text" class="form-control" id="Product_ID" value="P00113142">
    </div>
    <div class="form-group">
        <label for="Gender">Gender (例如: F)</label>
        <input type="text" class="form-control" id="Gender" value="F">
    </div>
    <div class="form-group">
        <label for="Age">Age (例如: 26-35)</label>
        <input type="text" class="form-control" id="Age" value="26-35">
    </div>
    <div class="form-group">
        <label for="Occupation">Occupation (例如: 2)</label>
        <input type="number" class="form-control" id="Occupation" value="2">
    </div>
    <div class="form-group">
        <label for="City_Category">City Category (例如: B)</label>
        <input type="text" class="form-control" id="City_Category" value="B">
    </div>
```

```
    <div class="form-group">
        <label for="Stay_In_Current_City_Years">Stay In Current City
Years (例如: 1)</label>
        <input type="text" class="form-control"
id="Stay_In_Current_City_Years" value="1">
    </div>
    <div class="form-group">
        <label for="Marital_Status">Marital Status (例如: 0)</label>
        <input type="number" class="form-control" id="Marital_Status"
value="0">
    </div>
    <div class="form-group">
        <label for="Product_Category_1">Product Category 1 (例如:
1)</label>
        <input type="number" class="form-control"
id="Product_Category_1" value="1">
    </div>
    <div class="form-group">
        <label for="Product_Category_2">Product Category 2 (例如:
5)</label>
        <input type="number" class="form-control"
id="Product_Category_2" value="5">
    </div>
    <div class="form-group">
        <label for="Product_Category_3">Product Category 3 (例如:
12)</label>
        <input type="number" class="form-control"
id="Product_Category_3" value="12">
    </div>
    <button type="submit" class="btn btn-primary">Predict</button>
</form>
<div id="result" class="mt-5 text-center" style="font-size:
1.5em;"></div>


</div>

    </body>
    </html>
    """)

#
# if __name__ == '__main__':
#     app.run(debug=True)


if __name__ == '__main__':
    pass
```

1. 使用 Gunicorn 启动 Flask 应用程序：

```
gunicorn -w 4 app:app
```

在这个命令中，`-w 4` 表示使用 4 个 worker 进程，`app:app` 表示应用程序的 Python 模块和 Flask 应用程序实例的名称。

Debug 过程语句

主要 Debug 步骤包括查看 Nginx 错误日志，重新加载 Nginx，安装 Flask 和其他 Python 库。

查看 Nginx 错误日志

使用以下命令查看 Nginx 错误日志的最后 50 行：

```
sudo tail -n 50 /var/log/nginx/error.log
```

这个命令可以帮助你查找和解决 Nginx 的问题。

重新加载 Nginx

使用以下命令重新加载 Nginx：

```
sudo systemctl reload nginx
```

当你修改了 Nginx 的配置文件后，你需要重新加载 Nginx 以使这些更改生效。

安装 Python 库

使用以下命令安装 Flask，Pandas 和 XGBoost：

```
pip3 install flask
pip install flask pandas xgboost
```

Flask 是一个轻量级的 Web 应用框架，Pandas 是一个强大的数据处理库，XGBoost 是一个优化的分布式梯度提升库，用于实现机器学习和数据科学。

```
import os

num_cores = os.cpu_count()

print(f"You have {num_cores} cores available on your machine.")

You have 12 cores available on your machine.

# 查看训练数据中有但测试数据中没有的特征
missing_features = set(X_train.columns) - set(test_data.columns)
print("Missing features:", missing_features)
```

```
Missing features: set()
```

```
! pip install tqdm
! sudo apt-get update
import os
from tqdm import tqdm

def install_package_with_apt(package_name):
    os.system(f"sudo apt-get install --allow-change-held-packages --
yes {package_name}")

def install_package_with_pip(package_name):
    os.system(f"pip install {package_name}")

apt_packages = [
    "texlive-latex-recommended",
    "dvipng",
    "texlive-latex-extra",
    "texlive-fonts-recommended",
    "cm-super",
]

pip_packages = [
    "SciencePlots",
]

for package in tqdm(apt_packages):
    install_package_with_apt(package)

for package in tqdm(pip_packages):
    install_package_with_pip(package)
```

附：部分环境安装指引

说明：该部分提供课题可能用到的一些环境准备指导，如果选择的环境和软件不在本章给定的范围内，请自行百度/Google 搜索。

MySQL 安装指引（Windows 系统）

1. MySQL 的安装可以参考在线指引：http://c.biancheng.net/view/7135.html.
2. 如果需要 MySQL 可视化管理工具，可选择 Navicat, SQLyog 等

Python 环境搭建（Windows 系统）

1. Python 开发运行环境安装可参考在线指引：https://blog.csdn.net/weixin_42596407/article/details/131043908
2. 为了便于安装扩展包，在安装 python 时，建议同时勾选安装 pip.