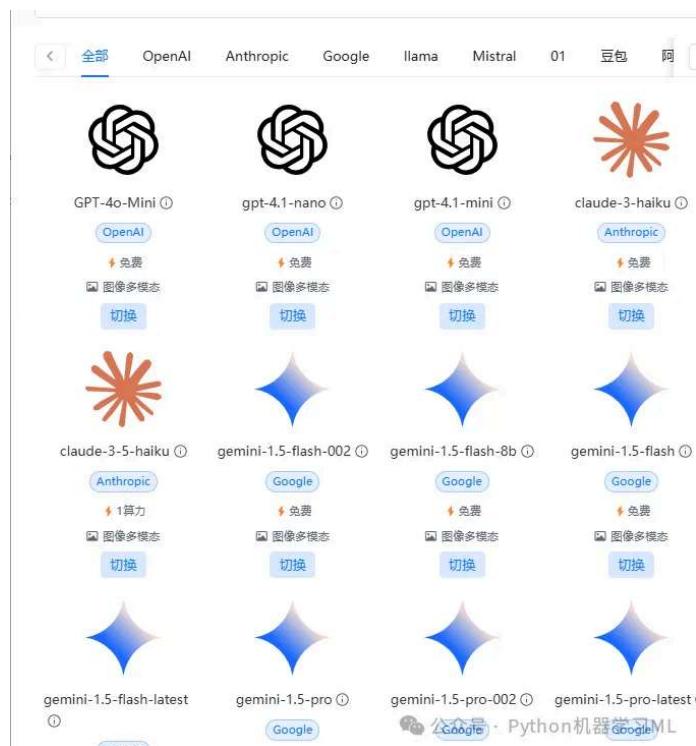


论文复现—基于随机森林、LightGBM、CatBoost、LightGBM、Optuna 优化与 SHAP 解释分析

原创 python_ML Python机器学习ML 2025年04月27日 00:21 重庆

Chaos AI Assistant 公益站 注册：<https://gpt-all.chat/auth?type=register&invite=MTM>

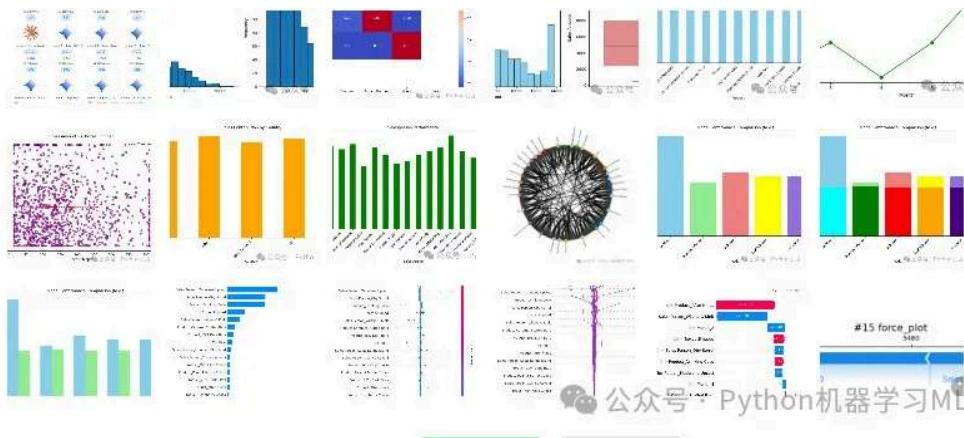
官网：<https://gpt-all.chat>



【数据，请加微信获取】



公众号



1. 数据准备充分：

代码细致地处理了数据的加载、清洗、类型转换、特征提取（时间特征、利润）和异常值控制，为后续分析和建模奠定了坚实的基础。

2. 探索性分析深入：

通过统计摘要和多种可视化手段（包括基础图表和高级图如 Chord 图、SHAP 图），脚本对数据进行了深入的探索，揭示了销售的关键模式、趋势、相关性以及不同维度（产品、时间、地区、人员）下的表现。

3. 建模流程规范：

遵循标准的机器学习流程，包括特征编码、数据划分、多种模型（决策树、随机森林及）的训练与评估。

4. 性能优化显著：

引入了 Optuna 进行系统的超参数优化，并通过交叉验证确保了优化过程的鲁棒性。优化后的模型性能得到了重新评估和比较，展示了调优对提升预测准确性的价值。

5. 模型解释性强：

项目没有止步于模型预测，而是利用先进的 SHAP 库对表现较好的 XGBoost 模型进行了深度解释。通过多种 SHAP 可视化，不仅量化了全局特征的重要性，还揭示了模型对单个样本做出预测的具体原因，大大增强了模型的可信度和可理解性。

6. 代码结构与实践：

代码组织清晰，注释详尽，使用了现代数据科学库，并考虑了结果的可视化与保存。

核心价值：不仅完成了一个具体的销售预测任务，更重要的是提供了一个**可复用、端到端的模板**，展示了如何结合数据分析、机器学习建模、自动化超参数优化和模型可解释性技术来解决实际业务问题。读者可以借鉴此流程和代码结构，将其应用于其他类似的数据集和回归（或稍作修改用于分类）任务中，通过调整数据处理、模型选择、优化策略和解释目标，来满足不同的分析需求。它强调了在追求预测精度的同时，理解模型决策过程的重要性。

第一部分 (阶段 1-21)

这部分完整地演示了一个典型的**数据分析与预处理流程**：从加载数据开始，通过清洗、特征工程、探索性分析（包括统计汇总和大量可视化）、异常值处理，到最后的特征编码，将原始数据逐步转化为适合机器学习模型使用的格式。每个阶段都有其明确的目的，并且许多步骤中的代码和思路都可以灵活地应用到其他不同的数据集和分析任务中。理解每个阶段的作用和其中使用的函数/方法，是进行有效数据分析的关键。

阶段一：导入所需库 (环境准备)

目的：加载所有必要的 Python 库。这些库提供了数据处理、数值计算、数据可视化、机器学习建模、模型评估和超参数调整等功能。这是任何数据分析或机器学习项目的第一步。

python

```
# -*- coding: utf-8 -*- # 声明文件编码为UTF-8，确保中文字符和注释正常处理
import time # 导入时间库，可用于测量代码执行时间或处理时间相关操作（此脚本中未直接用于计时）
```

```

# --- 计算与数据处理库 ---
import numpy as np # 导入 NumPy 库, 提供强大的 N 维数组对象和相关数学函数, 是科学计算的基础
import pandas as pd # 导入 Pandas 库, 提供 DataFrame 和 Series 等数据结构, 用于高效的数据处理

# --- 数据可视化库 ---
import seaborn as sns # 导入 Seaborn 库, 基于 Matplotlib, 提供更美观、更高级的统计图形绘制功能
import matplotlib.pyplot as plt # 导入 Matplotlib 的 pyplot 模块, 是 Python 中最常用的绘图库
import pandas as pd # 再次导入 Pandas 库(此行为冗余, 最佳实践是只导入一次, 但按要求保留所有导入)
import holoviews as hv # 导入 HoloViews 库, 用于创建声明式、可组合、交互式的数据可视化
from holoviews import opts # 从 HoloViews 导入 opts 模块, 用于配置和定制 HoloViews 图表的视觉风格

# --- 机器学习辅助与预处理库 ---
from sklearn.metrics import mean_squared_error # 从 Scikit-learn 导入均方误差函数, 用于评估模型性能
from sklearn.model_selection import train_test_split # 从 Scikit-learn 导入数据划分函数, 用于将数据集分为训练集和测试集

# --- 机器学习模型库 ---
from xgboost import XGBRegressor # 导入 XGBoost 库中的 XGBRegressor 类, 一个高效的梯度提升回归模型
from lightgbm import LGBMRegressor # 导入 LightGBM 库中的 LGBMRegressor 类, 一个速度更快、内存消耗更低的模型
from catboost import CatBoostRegressor # 导入 CatBoost 库中的 CatBoostRegressor 类, 一个能很好地处理类别特征的模型
from sklearn.tree import DecisionTreeRegressor # 从 Scikit-learn 导入决策树回归器模型
from sklearn.ensemble import RandomForestRegressor # 从 Scikit-learn 导入随机森林回归器模型,
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # 从 Scikit-learn 导入评估模型性能的指标

# --- 超参数优化库 ---
import optuna # 导入 Optuna 库, 一个用于自动化超参数搜索的框架
from sklearn.model_selection import cross_val_score # 从 Scikit-learn 导入交叉验证评分函数,

```

应用到其他数据集: 这一阶段的代码通常是标准配置。你需要根据你后续具体使用的模型、可视化工具或特定处理方法来增删导入的库。例如, 如果你处理的是文本数据, 可能需要导入 `nltk` 或 `spacy`; 如果做深度学习, 可能需要导入 `tensorflow` 或 `pytorch`。

阶段二：数据加载

目的: 从外部文件 (这里是 CSV 文件) 读取数据, 并将其加载到 Pandas DataFrame 对象中, 这是后续所有数据操作的基础。同时包含基本的错误处理, 以防文件加载失败。

python

```

try: # 尝试执行以下代码块, 用于捕获潜在的文件读取错误
    # 使用 Pandas 的 read_csv 函数读取指定路径下的 CSV 文件
    # 文件路径 '2025-4-26-公众号python机器学习MLChocolate Sales.csv' 是数据源
    # 读取成功后, 数据被存储在名为 df 的 DataFrame 变量中
    df = pd.read_csv('2025-4-26-公众号python机器学习MLChocolate Sales.csv')
except Exception as e: # 如果 try 代码块中发生任何异常(如文件不存在、路径错误、文件格式损坏等)
    # 打印一条错误消息, 提示数据文件加载失败, 并可以附带具体的错误信息 e (这里注释掉了具体错误信息)
    print(f'Error loading the data file: {e}') # 修改为打印具体错误信息

```

应用到其他数据集:

- **修改文件路径:**

将 `'2025-4-26-公众号python机器学习MLChocolate Sales.csv'` 替换为你的数据文件的实际路径和名称。

- **修改读取函数:**

如果你的数据不是 CSV 格式, 需要使用 Pandas 提供的其他读取函数, 例如:

- Excel 文件: `pd.read_excel('your_file.xlsx')`
- JSON 文件: `pd.read_json('your_file.json')`
- SQL 数据库: `pd.read_sql(query, connection_object)`

- 指定参数:

`read_csv` 有很多可选参数, 例如 `sep` (指定分隔符, 默认为逗号), `header` (指定表头行), `encoding` (指定文件编码, 如 'utf-8', 'gbk') 等, 根据你的文件特性进行设置。

阶段三：数据清洗与初步特征工程

目的: 对原始数据进行必要的清理, 使其格式规范、类型正确, 并从现有数据中派生出新的、可能对分析或建模有用的特征。

python

```
# --- 清洗 'Amount' 列 ---
# 选择 DataFrame 中的 'Amount' 列
# .str 是 Pandas Series 的字符串操作访问器
# .replace(r'\$,]', '', regex=True) 使用正则表达式移除字符串中的 '$' 符号和 ',' 逗号。r'' 表示
# .astype(float) 将清理后的字符串序列转换为浮点数（小数）类型, 以便进行数值计算
df['Amount'] = df['Amount'].str.replace(r'\$,]', '', regex=True).astype(float)

# --- 转换 'Date' 列为日期时间对象 ---
# 使用 Pandas 的 to_datetime 函数将 'Date' 列转换为标准的日期时间格式
# format='%d-%b-%y' 参数指定了原始日期字符串的格式: 日(dd)-月份缩写(例如Jan, Feb)-两位年份(yy)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%b-%y')

# --- 从 'Date' 列创建新特征 ---
# .dt 是 Pandas 日期时间序列的访问器, 可以方便地提取日期时间的各个部分
df['Month'] = df['Date'].dt.month # 提取 'Date' 列中的月份信息（1到12）, 并将其存储在新列 'Month'
df['Year'] = df['Date'].dt.year # 提取 'Date' 列中的年份信息（例如2022, 2023）, 并将其存储在新列 'Year'
```

应用到其他数据集:

- 识别需清洗的列:

检查你的数据中哪些列包含不规范的字符 (如货币符号、百分号、单位等) 或错误的数据类型。

- 调整清洗逻辑:

修改 `str.replace` 中的正则表达式或使用其他字符串处理方法 (如 `strip`, `split`) 来匹配你需要处理的模式。

- 调整日期格式:

如果你的日期格式不同, 需要相应修改 `pd.to_datetime` 中的 `format` 参数。例如, 'YYYY-MM-DD' 对应 `format='%Y-%m-%d'`, 'MM/DD/YYYY' 对应 `format='%m/%d/%Y'`。查阅 `strftime` 文档了解更多格式代码。

- 提取更多时间特征:

你可以从日期中提取更多有用的特征, 如:

- 星期几: `df['DayOfWeek'] = df['Date'].dt.dayofweek` (0=周一, 6=周日)
- 一年中的第几天: `df['DayOfYear'] = df['Date'].dt.dayofyear`
- 季度: `df['Quarter'] = df['Date'].dt.quarter`
- 是否周末: `df['IsWeekend'] = df['Date'].dt.dayofweek >= 5`

阶段四：数据基本信息概览

目的: 快速了解加载和初步处理后的数据集的基本情况, 包括数据的维度 (多少行、多少列)、每列的数据类型、是否有缺失值以及数值列的基本统计特征。

python

```
# --- 显示 DataFrame 的形状 ---
# df.shape 属性返回一个元组, 第一个元素是行数, 第二个元素是列数
print("\nShape of the DataFrame:", df.shape) # 在控制台打印 DataFrame 的形状信息
```

```

# --- 显示数据类型信息 ---
print("\nData types of each column:\n") # 打印提示性文字
# df.info() 方法打印 DataFrame 的简洁摘要，包括：
# - 总行数范围 (Index)
# - 每列的名称 (Column)
# - 每列的非空值数量 (Non-Null Count)，可以间接了解缺失值情况
# - 每列的数据类型 (Dtype)
# - DataFrame 的内存使用情况 (memory usage)
df.info()

# --- 显示 DataFrame 的内容（通常在交互式环境如 Jupyter Notebook 中自动渲染）---
# 在脚本或某些环境中，直接写变量名可能不会输出任何内容。
# 在 Jupyter Notebook 或类似REPL环境中，这会显示 DataFrame 的前几行和后几行。
# 如果想在任何环境下都打印，应使用 print(df) 或 print(df.head()) / print(df.tail())
df # 显示 DataFrame (在Jupyter等环境中)

# --- 生成描述性统计信息 ---
# df.describe() 方法计算数值类型列的常用描述性统计量，默认包括：
# - count: 非空值数量
# - mean: 平均值
# - std: 标准差
# - min: 最小值
# - 25%: 第1四分位数 (Q1)
# - 50%: 中位数 (Q2)
# - 75%: 第3四分位数 (Q3)
# - max: 最大值
# include='all' 参数使得该方法也尝试包含非数值（如对象类型、类别类型）列的统计信息：
# - count: 非空值数量
# - unique: 唯一值的数量
# - top: 最常出现的值
# - freq: 最常出现值的频率
print("\nDescriptive statistics:\n", df.describe(include='all')) # 打印描述性统计结果

```

应用到其他数据集：这些代码是通用的，适用于任何 Pandas DataFrame。它们是数据探索的起点，帮助你快速掌握数据的基本面貌。`df.info()` 对于检查数据类型是否符合预期以及快速发现缺失值非常有用。

`df.describe()` 则有助于理解数值数据的分布范围和中心趋势，以及类别数据的多样性。

阶段五：探索性数据分析 (EDA) - 分布分析 (可视化)

目的：通过直方图可视化关键数值变量 ('Amount' 和 'Boxes Shipped') 的分布情况，了解它们的频率分布形态，判断是否存在偏态、双峰或多峰等特征，以及大致的数据范围。

python

```

# --- 分析关键变量的分布 ---
print("\nDistribution of 'Amount' and 'Boxes Shipped':") # 打印分析目的的提示信息

# --- 创建图形和子图 ---
# plt.figure() 创建一个新的图形窗口 (Figure 对象)
# figsize=(12, 5) 参数设置图形的尺寸，宽度为 12 英寸，高度为 5 英寸
plt.figure(figsize=(12, 5))

# --- 绘制 'Amount' 的直方图 ---
# plt.subplot(nrows, ncols, index) 在图形窗口中创建一个子图网格，并指定当前绘图的子图位置
# 1, 2, 1 表示创建一个 1 行 2 列的网格，当前在第 1 个位置绘图
plt.subplot(1, 2, 1)
# plt.hist() 函数绘制直方图
# df['Amount'] 是要绘制的数据序列

```

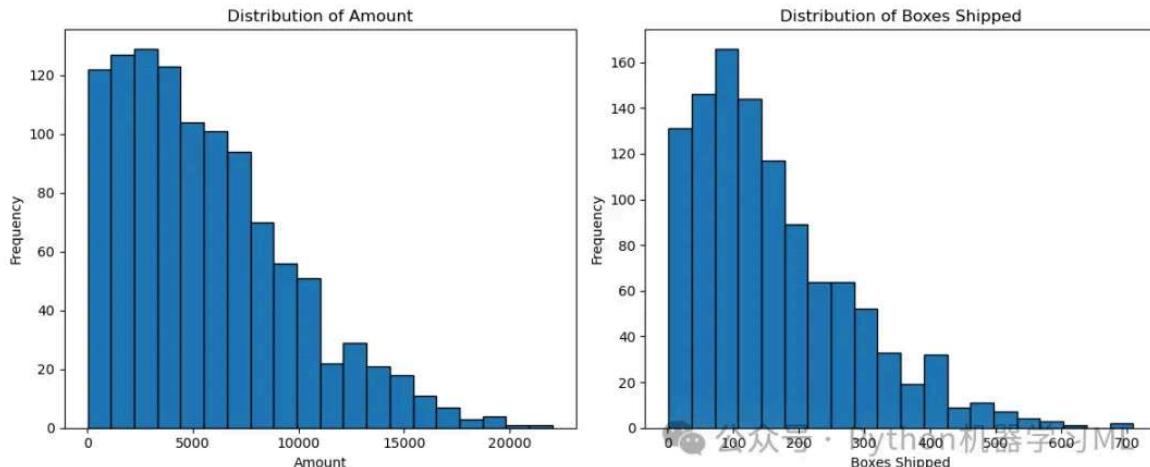
```

# bins=20 指定将数据范围划分为 20 个等宽的区间（条柱）
# edgecolor="black" 设置每个条柱的边缘颜色为黑色，使边界更清晰
plt.hist(df['Amount'], bins=20, edgecolor = "black")
plt.xlabel('Amount') # 设置当前子图的 x 轴标签文字
plt.ylabel('Frequency') # 设置当前子图的 y 轴标签文字
plt.title('Distribution of Amount') # 设置当前子图的标题

# --- 绘制 'Boxes Shipped' 的直方图 ---
# 1, 2, 2 表示在 1 行 2 列网格的第 2 个位置绘图
plt.subplot(1, 2, 2)
# 绘制 'Boxes Shipped' 列的直方图，参数含义同上
plt.hist(df['Boxes Shipped'], bins=20, edgecolor = "black")
plt.xlabel('Boxes Shipped') # 设置 x 轴标签
plt.ylabel('Frequency') # 设置 y 轴标签
plt.title('Distribution of Boxes Shipped') # 设置子图标题

# --- 调整布局并显示图形 ---
# plt.tight_layout() 自动调整子图参数（如间距），以给出紧凑的布局，防止标签重叠
plt.tight_layout()
# plt.show() 将所有已创建的图形显示出来。在脚本环境中必须调用此函数才能看到图形。
plt.show()

```



应用到其他数据集：

- 选择关键变量：**
将 'Amount' 和 'Boxes Shipped' 替换为你数据集中你最关心的、需要了解其分布的数值型变量名。
- 调整区间数量 (bins)：**
`bins` 的选择会影响直方图的外观和揭示的细节。可以尝试不同的值（如 10, 30, 50 或根据数据特点计算，如 Freedman-Diaconis rule）来找到最能反映数据分布的设置。
- 修改标签和标题：**
务必更新 `xlabel`, `ylabel`, `title` 为符合你所分析变量的实际含义。
- 分析其他变量：**
你可以为数据集中其他重要的数值变量重复这个过程，或者使用循环来绘制多个变量的分布图。

阶段六：探索性数据分析 (EDA) - 相关性分析 (可视化)

目的： 计算数据集中数值特征之间的线性相关性（皮尔逊相关系数），并通过热力图将其可视化。这有助于发现哪些变量倾向于一起变化（正相关或负相关），为特征选择或理解变量间关系提供线索。

python

```

# --- 计算并可视化相关性矩阵 ---
# df.corr() 方法计算 DataFrame 中所有数值类型列两两之间的皮尔逊相关系数，返回一个相关系数矩阵

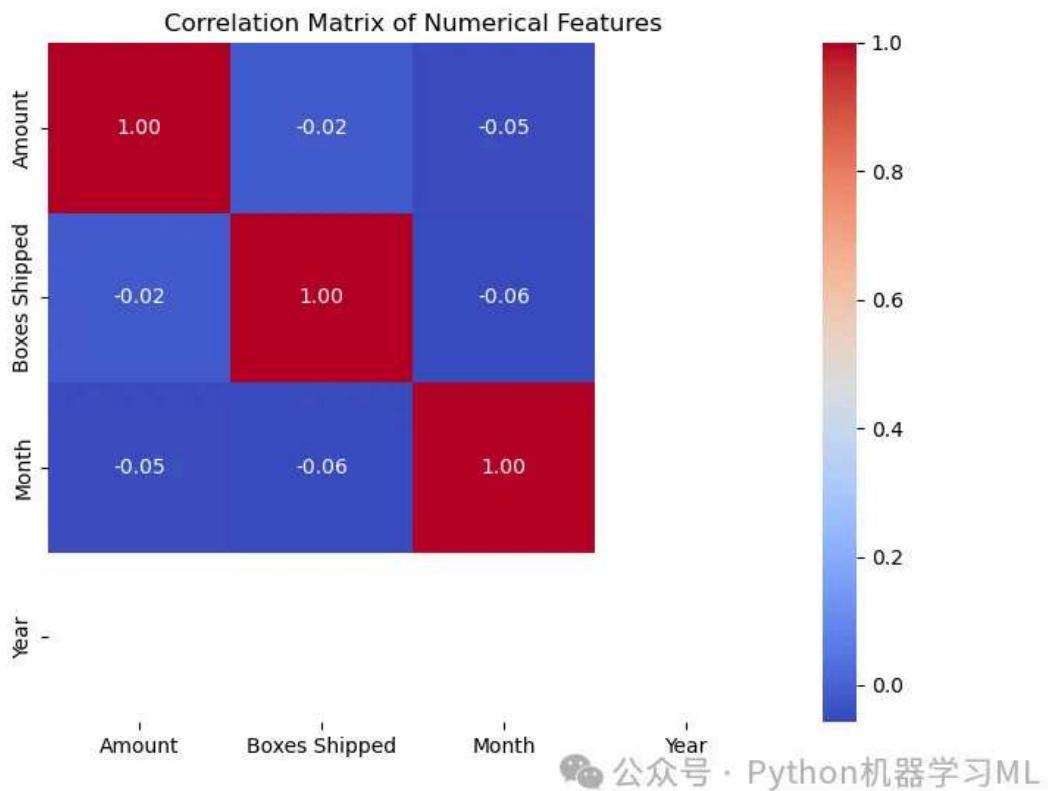
```

```

# numeric_only=True 参数确保只对数值类型的列进行计算。在较新版本的 Pandas 中，如果 DataFrame 1
correlation_matrix = df.corr(numeric_only=True)
print("\nCorrelation Matrix:\n", correlation_matrix) # 在控制台打印计算得到的相关系数矩阵

# --- 绘制相关性热力图 ---
# plt.figure() 创建一个新的图形窗口
# figsize=(8, 6) 设置图形尺寸
plt.figure(figsize=(8, 6))
# sns.heatmap() 函数用于绘制热力图
# correlation_matrix 是输入的数据矩阵
# annot=True 参数表示在每个单元格上显示对应的相关系数值
# cmap='coolwarm' 参数指定颜色映射方案。'coolwarm' 是一个发散型色图，通常蓝色表示负相关，红色表示正相关
# fmt=".2f" 参数设置单元格上标注数值的格式，这里是保留两位小数的浮点数。
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features') # 设置图形的标题
plt.show() # 显示绘制好的热力图

```



应用到其他数据集：

- **通用性：**

这部分代码通常可以直接应用于任何包含多个数值列的 DataFrame。`df.corr()` 和 `sns.heatmap()` 会自动处理所有数值列。

- **选择色图 (cmap)：**

你可以根据个人喜好或数据特点选择不同的 `cmap` 值。常见选择包括 'viridis', 'plasma', 'Blues', 'Reds', 'RdBu' 等。查阅 Matplotlib 或 Seaborn 文档获取更多色图选项。

- **处理大量特征：**

如果数值特征非常多，热力图可能会变得拥挤。这时可以考虑：

- 只选择一部分最重要的特征进行可视化。
- 不显示标注 (`annot=False`)，只观察颜色模式。
- 设置一个阈值，只显示相关性绝对值大于某个值的关系。
- 增大 `figsize`。

阶段七：缺失值检查

目的：检查数据集中每一列存在的缺失值 (NaN, None, NaT等) 的数量。了解缺失值的分布是数据清洗和预处理的重要一步，因为很多模型不能处理含有缺失值的数据。

python

```
# --- 检查缺失值 ---
# df.isnull() 返回一个与 df 形状相同、元素为布尔值 (True/False) 的 DataFrame，其中 True 表示对
# .sum() 方法沿着轴 0 (默认，即按列) 对布尔值进行求和。True 被当作 1，False 被当作 0。
# 最终结果是一个 Pandas Series，其索引是原 DataFrame 的列名，值是对应列中缺失值的总数。
print(df.isnull().sum()) # 在控制台打印每列的缺失值计数
```

应用到其他数据集：这行代码是完全通用的，直接复制粘贴即可用于检查任何 Pandas DataFrame 的缺失值情况。根据输出结果，你需要决定如何处理缺失值，常见策略包括：

- **删除：**
删除含有缺失值的行 (`df.dropna()`) 或列 (`df.dropna(axis=1)`)。适用于缺失比例很小或该行/列信息不重要的情况。
- **填充：**
使用特定值 (如 0、均值、中位数、众数) 或基于其他列的预测值来填充缺失值 (`df.fillna()`)。
- **使用能处理缺失值的模型：**
某些模型 (如 XGBoost, LightGBM, CatBoost) 可以直接处理缺失值，无需显式填充。

阶段八：异常值处理 (Winsorization)

目的：通过 Winsorization 方法来处理数值列中的极端异常值。这种方法不是删除异常值，而是将超出特定百分位数范围的值“拉回”到边界值，从而减小极端值对统计分析 (如均值、标准差) 和模型训练的过度影响，同时保留数据的原始数量。

python

```
# --- 定义 Winsorization 函数 ---
# 定义一个名为 winsorize_outliers 的函数，它接受两个参数：
# - series: 一个 Pandas Series 对象 (即 DataFrame 的一列)
# - limits: 一个元组，包含两个介于 0 和 1 之间的小数，分别代表下限和上限的分位数阈值。默认值为
def winsorize_outliers(series, limits=(0.05, 0.95)):
    # series.quantile(limits[0]) 计算输入 Series 的下限分位数 (例如 5% 分位数)
    lower_limit = series.quantile(limits[0])
    # series.quantile(limits[1]) 计算输入 Series 的上限分位数 (例如 95% 分位数)
    upper_limit = series.quantile(limits[1])
    # series.clip(lower, upper) 方法将 Series 中所有小于 lower_limit 的值替换为 lower_limit,
    # 所有大于 upper_limit 的值替换为 upper_limit，介于两者之间的值保持不变。
    winsorized_series = series.clip(lower_limit, upper_limit)
    # 返回经过 Winsorization 处理后的新 Series
    return winsorized_series

# --- 对指定列应用 Winsorization ---
# 使用 for 循环遍历一个包含需要处理的列名的列表 ['Amount', 'Boxes Shipped']
for col in ['Amount', 'Boxes Shipped']:
    # 对 DataFrame df 中的当前列 col 调用 winsorize_outliers 函数进行处理
    # 将返回的处理后的 Series 重新赋值给 df[col]，覆盖原始列
    df[col] = winsorize_outliers(df[col])
    # 使用 f-string 打印一条信息，告知用户哪一列的异常值已被 Winsorized 处理
    print(f"Winsorized outliers in '{col}'.")

# --- 显示处理后的描述性统计 ---
# 显示处理后的描述性统计
```

```
# 再次调用 df.describe() 来查看 Winsorization 处理后数据的统计摘要
# 对比处理前的 describe() 结果，可以观察到 min 和 max 值通常会变化，更接近 5% 和 95% 分位数的值
print(df.describe())
```

应用到其他数据集：

- **选择处理列：**

修改 `['Amount', 'Boxes Shipped']` 列表，包含你数据集中需要进行异常值处理的数值列名。通常应用于那些受极端值影响较大的变量。

- **调整分位数阈值 (limits)：**

`limits=(0.05, 0.95)` 是一个常用的设置，但并非唯一选择。你可以根据数据的具体分布和业务需求调整这个阈值，例如 `(0.01, 0.99)` 会处理更极端的值，而 `(0.1, 0.9)` 则处理范围更广。

- **理解局限性：**

Winsorization 是一种相对温和的异常值处理方法。它改变了数据的原始分布（尤其是在尾部），如果极端值本身包含重要信息，这种处理可能不适用。需要根据具体问题判断是否采用以及如何设置参数。其他异常值处理方法还包括删除、用中位数/均值替换、或者使用更鲁棒的模型。

阶段九：特征工程 - 计算利润

目的： 基于现有的 'Amount' (销售额) 列，创建一个新的特征列 'Profit' (利润)。这里假设了一个固定的利润率 (20%) 来计算利润。特征工程是根据对业务的理解，从原始数据中创造出对分析或建模更有价值的新信息的过程。

python

```
# --- 计算 'Profit' 列 ---
# 在 DataFrame df 中创建一个名为 'Profit' 的新列
# 新列的值等于 'Amount' 列的值乘以 0.20 (即 20%)
# 这假设了一个简单的模型：利润 = 销售额 * 利润率 (这里是 20%)
df['Profit'] = df['Amount'] * 0.20

# --- 显示更新后的 DataFrame (前几行) ---
# 使用 df.head() 方法查看 DataFrame 的前 5 行 (默认行数)
# 目的是快速检查 'Profit' 列是否已成功添加，并查看计算出的利润值是否符合预期
print(df.head())
```

应用到其他数据集：

- **调整计算逻辑：**

这是特征工程的核心。你需要根据你数据中的可用信息和业务逻辑来定义新特征的计算方式。例如：

- 如果你的数据中有成本 ('Cost') 列，利润可以更精确地计算为：`df['Profit'] = df['Amount'] - df['Cost']`
- 你可以计算单位价格：`df['UnitPrice'] = df['Amount'] / df['Quantity']` (如果存在 'Quantity' 列)
- 你可以创建比率特征：`df['ProfitMargin'] = df['Profit'] / df['Amount']`
- 你可以基于多个现有特征创建复合特征。

- **特征命名：**

给新创建的特征起一个清晰、有意义的名字。

阶段十：探索性数据分析 (EDA) - 销售分析 (汇总与排名)

目的： 对销售数据进行聚合分析，了解整体销售额的统计分布，并识别出销售额最高 (畅销) 和最低 (滞销) 的产品。这有助于了解产品层面的表现差异。

python

```
# --- 销售额分布统计 ---
# 对 'Amount' 列调用 .describe() 方法，计算并返回该列的描述性统计信息（计数、均值、标准差、最小值、最大值等）
```

```

sales_stats = df['Amount'].describe()
print(sales_stats) # 打印销售额的统计摘要，提供对整体销售水平和变异性的概览

# --- 识别畅销和滞销产品 ---
# df.groupby('Product') 根据 'Product' 列的值对 DataFrame 进行分组，得到一个 GroupBy 对象
# ['Amount'] 选择每个分组中的 'Amount' 列
# .sum() 对每个产品分组内的 'Amount' 值进行求和，得到每个产品的总销售额
product_sales = df.groupby('Product')['Amount'].sum()
# 结果 product_sales 是一个 Pandas Series，索引是产品名称，值是对应的总销售额
print("\nProduct Sales:\n", product_sales) # 打印每个产品的总销售额

# product_sales.idxmax() 方法返回具有最大值的索引（即销售额最高的产品名称）
best_selling_product = product_sales.idxmax()
# product_sales.idxmin() 方法返回具有最小值的索引（即销售额最低的产品名称）
worst_selling_product = product_sales.idxmin()
print(f"\nBest Selling Product: {best_selling_product}") # 使用 f-string 打印最畅销的产品名
print(f"Worst Selling Product: {worst_selling_product}") # 使用 f-string 打印最滞销的产品名

```

应用到其他数据集：

- **修改分组列：**

将 'Product' 替换为你数据中用于标识分析单元（如产品、客户、店铺、类别等）的列名。

- **修改聚合指标：**

将 'Amount' 替换为你关心的绩效指标列名（如 'Profit', 'Quantity', 'Visits', 'Clicks' 等）。

- **修改聚合函数：**

除了 .sum()，你还可以使用其他聚合函数，例如：

- 计算平均值: .mean()
- 计算交易次数/记录数: .count() 或 .size()
- 计算唯一客户数: .nunique() (需要先选择客户ID列)
- 计算最大/最小值: .max() / .min()

- **多级分组：**

你可以按多个列进行分组，如 df.groupby(['Category', 'Product'])['Amount'].sum() 来分析每个类别下各产品的销售额。

- **寻找 Top N / Bottom N：**

除了 idxmax() 和 idxmin()，你还可以使用 .nlargest(N) 和 .nsmallest(N) 来获取销售额最高或最低的 N 个产品。

阶段十一：探索性数据分析 (EDA) - 时间趋势与关系分析

目的： 分析销售额随时间（年、月）的变化趋势，并计算销售额与发货箱数这两个变量之间的线性相关性。这有助于理解业务的季节性、年度增长情况以及关键变量间的关系。

python

```

# --- 分析随时间变化的销售趋势 ---
# df.groupby('Year') 按 'Year' 列分组
# ['Amount'].sum() 计算每个年份的总销售额
yearly_sales = df.groupby('Year')['Amount'].sum()
print("\nYearly Sales:\n", yearly_sales) # 打印按年汇总的销售额

# df.groupby('Month') 按 'Month' 列分组
# ['Amount'].sum() 计算每个月份的总销售额（跨所有年份）
monthly_sales = df.groupby('Month')['Amount'].sum()
print("\nMonthly Sales:\n", monthly_sales) # 打印按月汇总的销售额（注意这可能混合了不同年份的）

# --- 销售额与发货箱数的关系 ---
# df['Amount'].corr(df['Boxes Shipped']) 计算 'Amount' 列与 'Boxes Shipped' 列之间的皮尔逊相关系数

```

```
# 该系数衡量两个变量线性关系的强度和方向，范围从 -1 (完全负相关) 到 +1 (完全正相关)，0 表示无相关性
correlation = df['Amount'].corr(df['Boxes Shipped'])
print(f"\nCorrelation between Sales Amount and Boxes Shipped: {correlation:.2f}") # 使用 f-string
```

应用到其他数据集：

- **检查时间特征：**

确保你已经从日期/时间列中提取了所需的周期特征（如 'Year', 'Month', 'Week', 'DayOfWeek'）。

- **选择分析周期：**

根据需要修改 `groupby()` 中的列名来进行不同时间粒度（年、季、月、周、日等）的趋势分析。

- **分析特定时间段：**

如果想看特定年份（如 2023 年）的月度趋势，可以先筛选数据：`df[df['Year'] == 2023].groupby('Month')[['Amount']].sum()`。

- **计算其他相关性：**

将 `df['Amount']` 和 `df['Boxes Shipped']` 替换为你想要探究其线性关系的其他任意两个数值列。

- **注意相关性局限：**

皮尔逊相关系数只衡量线性关系。两个变量可能存在非线性关系但线性相关系数接近于零。需要结合散点图等可视化手段来全面理解变量关系。

阶段十二：探索性数据分析 (EDA) - 按类别细分分析

目的： 进一步细分数据，分析不同国家/地区和销售人员的销售贡献。同时，进行多维度交叉分析，例如查看不同产品在各个国家的销售表现，这有助于发现地区偏好或销售策略的效果。

python

```
# --- 各国的销售贡献 ---
# df.groupby('Country') 按 'Country' 列分组
# ['Amount'].sum() 计算每个国家的总销售额
country_sales = df.groupby('Country')['Amount'].sum()
print("\nCountry Sales:\n", country_sales) # 打印按国家汇总的销售额

# --- 销售人员业绩 ---
# df.groupby('Sales Person') 按 'Sales Person' 列分组
# ['Amount'].sum() 计算每个销售人员的总销售额
salesperson_sales = df.groupby('Sales Person')['Amount'].sum()
print("\nSalesperson Performance:\n", salesperson_sales) # 打印按销售人员汇总的销售额

# --- 组合类别分析 - 示例：产品在各国的表现 ---
# df.groupby(['Product', 'Country']) 按 'Product' 和 'Country' 两列进行多级分组
# ['Amount'].sum() 计算每个 (产品, 国家) 组合的总销售额，结果是一个具有多级索引 (MultiIndex)
# .unstack() 方法将最后一级索引 ('Country') 从行索引“提升”为列索引。
# 结果 product_country_sales 是一个 DataFrame，其行索引是 'Product'，列索引是 'Country'，单元格值是销售额。
# 这非常适合比较同一个产品在不同国家的表现，或同一个国家对不同产品的偏好。
product_country_sales = df.groupby(['Product', 'Country'])['Amount'].sum().unstack()
print("\nProduct Performance by Country:\n", product_country_sales) # 打印产品在各国销售表现
```

应用到其他数据集：

- **修改分组列：**

将 `'Country'`, `'Sales Person'`, `'Product'` 替换为你数据中代表不同维度（如地区、渠道、客户细分、产品类别等）的分类列名。

- **选择聚合指标：**

同样，将 `'Amount'` 替换为你关心的指标。

- **使用 `unstack()` 进行交叉分析：**

当你按两个分类变量分组并聚合一个数值变量时，`.unstack()` 是一个强大的工具，可以将结果重塑为更易于比较的二维表格（交叉表或透视表）。你可以通过 `.unstack(level=...)` 来指定将哪一级索引转换为列。

• 处理缺失值 (NaN):

在 `unstack()` 后的结果中，如果某个组合（如某个产品在某个国家没有销售记录）不存在，其单元格的值会是 NaN (Not a Number)。你可以使用 `.fillna(0)` 将这些 NaN 替换为 0，如果这在你的业务场景下是合理的。

阶段十三：可视化 - 销售额分布 (直方图与箱线图)

目的： 使用两种不同的图表（直方图和箱线图）在同一个图形窗口中更全面地展示 'Amount'（销售额）的分布特征。直方图显示频率分布，箱线图则清晰地标示出中位数、四分位数、数据范围以及潜在的异常值。

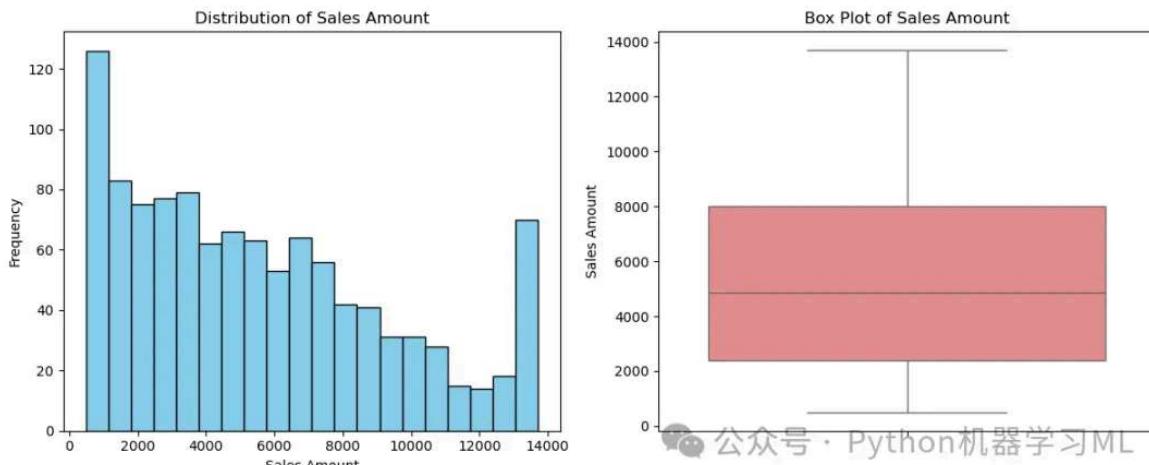
python

```
# plt.figure() 创建一个新的图形窗口
# figsize=(12, 5) 设置图形尺寸
plt.figure(figsize=(12, 5))

# --- 绘制直方图 ---
# plt.subplot(1, 2, 1) 在 1x2 网格的第 1 个位置创建子图
plt.subplot(1, 2, 1)
# 绘制 'Amount' 的直方图
# color='skyblue' 设置填充颜色
# edgecolor='black' 设置边框颜色
plt.hist(df['Amount'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Sales Amount') # x 轴标签
plt.ylabel('Frequency') # y 轴标签
plt.title('Distribution of Sales Amount') # 子图标题

# --- 绘制箱线图 ---
# plt.subplot(1, 2, 2) 在 1x2 网格的第 2 个位置创建子图
plt.subplot(1, 2, 2)
# 使用 Seaborn 的 boxplot 函数绘制箱线图
# y=df['Amount'] 指定数据源和绘制方向（垂直箱线图，数据在 y 轴上）
# color='lightcoral' 设置箱体的颜色
sns.boxplot(y=df['Amount'], color='lightcoral')
# 箱线图的关键元素：
# - 箱体顶部：第 3 四分位数 (Q3)
# - 箱体中间线：中位数 (Q2)
# - 箱体底部：第 1 四分位数 (Q1)
# - 箱体高度：四分位距 (IQR = Q3 - Q1)
# - 上须线 (whisker)：通常延伸到 Q3 + 1.5 * IQR 范围内的最大值
# - 下须线 (whisker)：通常延伸到 Q1 - 1.5 * IQR 范围内的最小值
# - 超出须线的点：通常被认为是潜在的异常值
plt.ylabel('Sales Amount') # y 轴标签（对于垂直箱线图，主要看 y 轴）
plt.title('Box Plot of Sales Amount') # 子图标题

plt.tight_layout() # 自动调整布局，防止重叠
plt.show() # 显示图形
```



应用到其他数据集：

- 替换变量：

将 `df['Amount']` 替换为你想要同时用直方图和箱线图进行分析的任何数值列。

- 调整样式：

你可以修改 `color`, `bins`, `edgecolor` 等参数来改变图表的外观。

- 水平箱线图：

如果想绘制水平箱线图，可以使用 `sns.boxplot(x=df['YourColumn'])` 并相应调整标签。

- 分组箱线图：

箱线图特别适合比较不同类别下数值变量的分布。例如，要比较不同产品 ('Product') 的销售额分布，可以使用 `sns.boxplot(x='Product', y='Amount', data=df)`。

阶段十四：特征编码 - One-Hot Encoding (创建副本)

目的： 将原始 DataFrame 中的分类特征 ('Sales Person', 'Country', 'Product') 转换为机器学习模型能够处理的数值格式。这里采用 One-Hot Encoding 方法，它为每个分类变量的每个唯一类别创建一个新的二元（0 或 1）特征列。为了后续某些可视化（如 Chord 图）仍然使用原始分类列，这里先将编码结果存储在一个副本 `df_copy` 中，而不是直接修改原始的 `df`。

python

```
# --- 识别需要编码的分类列 ---
# 定义一个列表 categorical_cols，包含要进行 One-Hot Encoding 的原始分类列的名称
categorical_cols = ['Sales Person', 'Country', 'Product']

# --- 对 DataFrame 副本应用 One-Hot Encoding ---
# pd.get_dummies() 是 Pandas 中执行 One-Hot Encoding 的主要函数
# df 是输入的原始 DataFrame
# columns=categorical_cols 指定对列表中的这些列进行编码
# drop_first=True 参数会在每个原始分类列的编码结果中删除第一个类别的对应列。
# 这样做是为了避免完全多重共线性 (dummy variable trap)，即新生成的编码列加和恒等于 1，这在某些模型中是不允许的。
# 对于树模型（如决策树、随机森林、XGBoost）通常不是必需的，但也是一种常见的做法。
# 将编码后的结果（包含原始数值列和新生成的编码列）存储在一个名为 df_copy 的新 DataFrame 中
df_copy = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# --- 显示原始 DataFrame 的前几行 ---
# 注意：这里打印的是原始 DataFrame df 的头部信息，而不是编码后的 df_copy。
# 目的是展示编码操作并未修改原始 df（因为结果存入了 df_copy），或者只是作为代码流程中的一个检查
print(df.head())
```

应用到其他数据集：

- 确定分类列：

修改 `categorical_cols` 列表，包含你数据集中所有需要转换为数值格式的分类变量（通常是 `object` 或 `category` 类型）。

- **选择编码方法：**

One-Hot Encoding 是最常用的方法之一，特别适用于名义型分类变量（类别间没有顺序关系）。对于有序分类变量（如 'Small', 'Medium', 'Large'），可以考虑 Ordinal Encoding。对于具有非常多类别的变量，可以考虑其他编码技术（如 Target Encoding, Binary Encoding）。

- `drop_first` 参数：

根据你后续使用的模型类型决定是否使用 `drop_first=True`。对于线性模型（如线性回归、逻辑回归），通常建议设置为 `True`。对于树模型，可以设置为 `False`。

- **处理新类别：**

`get_dummies` 默认情况下，如果在测试集或新数据中遇到训练时未见过的类别，会产生全零的编码行。可以通过设置 `dummy_na=True` 来为缺失值创建一个单独的编码列，或者设置 `handle_unknown='ignore'`（在某些库或未来版本中可能支持）来忽略未知类别。

阶段十五：可视化 - 基于编码数据的产品销售额（估算）

目的： 使用经过 One-Hot Encoding 处理后的 `df_copy` DataFrame 来可视化不同产品的“总销售额”。**请注意：** 这里的计算方法 (`sum() * mean() / count`) 是一种估算或简化，它实际上计算的是（每个产品的交易次数 * 数据集平均销售额 / 产品种类数）。这可能与直接按产品分组求和（`df.groupby('Product')['Amount'].sum()`）得到的结果不同，后者通常是更准确反映各产品总销售额的方法。此可视化展示的是基于这种特定计算逻辑的产品表现。

python

```
# --- 提取与产品相关的 One-Hot 编码列 ---
# 使用列表推导式从 df_copy 的所有列名中筛选出那些以 'Product_' 开头的列名
# 这些列是 'Product' 列经过 One-Hot Encoding 后生成的代表各个具体产品的新列
product_columns = [col for col in df_copy.columns if 'Product_' in col]

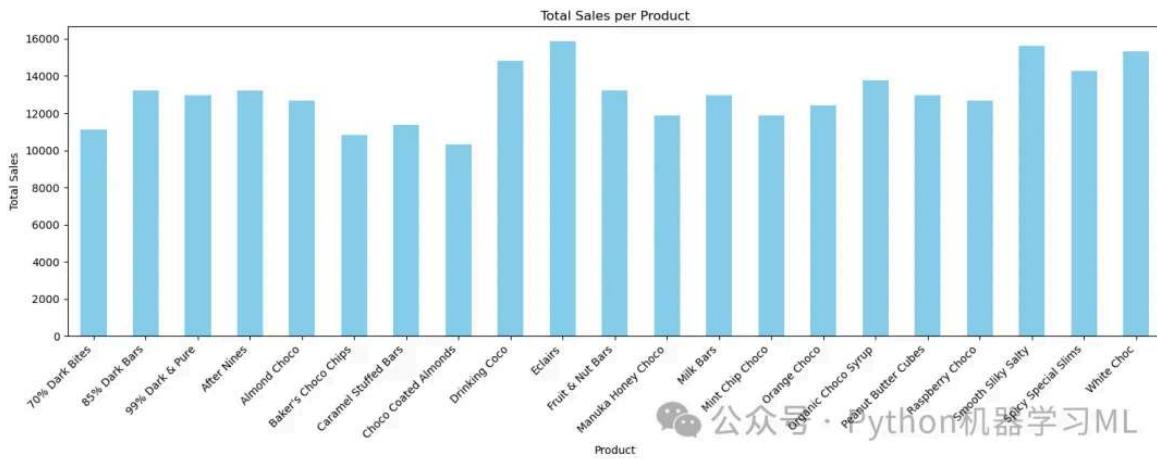
# --- 计算每个产品的估算“总销售额” ---
# df_copy[product_columns].sum() 对每个产品编码列求和，得到每个产品在数据集中出现的次数（交易量）
# df_copy['Amount'].mean() 计算整个数据集中 'Amount' 列的平均值（平均销售额）
# (次数 * 平均销售额) 可以看作是对每个产品基于其交易频率和整体平均售价的一个加权度量
# / len(product_columns) 除以产品种类的数量，似乎是为了进行某种平均化或标准化，其确切含义取决于上下文
# 结果 product_sales 是一个 Series，索引是产品编码列名，值是按上述方法计算的估算销售额。
product_sales = df_copy[product_columns].sum() * df_copy['Amount'].mean() / len(product_columns)

# --- 绘制产品销售额条形图 ---
# plt.figure() 创建图形窗口
plt.figure(figsize=(15, 6))
# product_sales.plot(kind='bar', ...) 对计算出的 product_sales Series 绘制条形图
# kind='bar' 指定图表类型为条形图
# color='skyblue' 设置条形颜色
# 返回一个 Axes 对象 ax，用于后续更精细的控制
ax = product_sales.plot(kind='bar', color='skyblue')
# ax.set_xticklabels(...) 设置 x 轴的刻度标签
# 使用列表推导式从 product_columns (编码列名，如 'Product_A', 'Product_B') 中提取出原始产品名
# col_name.split('_')[1] 按下划线分割字符串并取第二部分
# rotation=45 将 x 轴标签旋转 45 度以防重叠
ax.set_xticklabels([col_name.split('_')[1] for col_name in product_columns], rotation=45)
plt.xlabel('Product') # 设置 x 轴标签
plt.ylabel('Total Sales (Estimated)') # 设置 y 轴标签，强调这是估算值
plt.title('Estimated Total Sales per Product (Based on Count * Avg Amount)') # 设置图表标题
plt.xticks(ha='right') # 设置 x 轴刻度标签的水平对齐方式为右对齐，配合旋转角度，使标签更易读
```

```

plt.tight_layout() # 自动调整布局
plt.show() # 显示图形

```



应用到其他数据集：

- 修改列名前缀：

将 `'Product_'` 替换为你进行 One-Hot Encoding 时对应的分类列前缀（例如 `'Category_'`, `'Region_'` 等）。

- 调整可视化：

修改 `figsize`, `color`, `xlabel`, `ylabel`, `title` 等。

- 关键：理解计算逻辑：

如果你需要精确的总销售额可视化，强烈建议使用原始 `df` 和 `groupby().sum()` 的结果来绘图（如阶段十所示）。如果你确实需要按照代码中的这种特定逻辑进行可视化，请确保你理解其含义，并在标签和标题中清晰说明。

阶段十六：可视化 - 月度销售趋势

目的：绘制一条线图，展示总销售额随月份（1 到 12）的变化趋势。这有助于识别销售的季节性模式或周期性。

python

```

# plt.figure() 创建图形窗口
plt.figure(figsize=(12, 4)) # 设置尺寸，宽度 12，高度 4

# --- 绘制月度销售趋势线图 ---
# plt.subplot(1, 1, 1) 创建一个 1x1 网格的子图（实际上就是整个图形区域）
plt.subplot(1, 1, 1)
# df.groupby('Month')['Amount'].sum() 计算每个月的总销售额（跨所有年份）
# plt.plot() 函数绘制线图
# 第一个参数是 x 轴数据（这里是 groupby 结果的索引，即月份 1-12），第二个参数是 y 轴数据（月度
# marker='o' 在每个数据点位置绘制一个圆形标记
# linestyle='-' 指定使用实线连接数据点
# color='green' 设置线条和标记的颜色为绿色
plt.plot(df.groupby('Month')['Amount'].sum(), marker='o', linestyle='-', color='green')
plt.xlabel('Month') # 设置 x 轴标签
plt.ylabel('Total Sales') # 设置 y 轴标签
plt.title('Monthly Sales Trend (Aggregated Across Years)') # 设置图表标题，说明是跨年汇总的月
plt.xticks(range(1, 13)) # 确保 x 轴显示所有月份 1 到 12 的刻度
plt.grid(axis='y', linestyle='--', alpha=0.7) # 添加水平网格线，使其更容易比较不同月份的值

```

```
plt.tight_layout() # 自动调整布局  
plt.show() # 显示图形
```



应用到其他数据集：

- 确保有月份列：

确认你的 DataFrame 中有 'Month' 列（或类似的周期列，如 'Week', 'DayOfWeek'）。

- 替换指标：

将 `'Amount'` 替换为你想要按月（或其他周期）分析其趋势的数值指标。

- 修改样式：

调整 `figsize`, `marker`, `linestyle`, `color` 等参数。

- 分析特定时间段：

如果需要看某一年内的月度趋势，或者多年内按年分别展示月度趋势，需要先对数据进行筛选或使用 Seaborn 等库绘制更复杂的分组线图（`sns.lineplot(x='Month', y='Amount', hue='Year', data=df)`）。

阶段十七：可视化 - 销售额 vs 发货箱数 (散点图与趋势线)

目的：通过散点图直观地展示两个数值变量（'Amount' 和 'Boxes Shipped'）之间的关系模式。同时，添加一条线性回归趋势线（最佳拟合直线），以量化和可视化它们之间的线性关联程度。

python

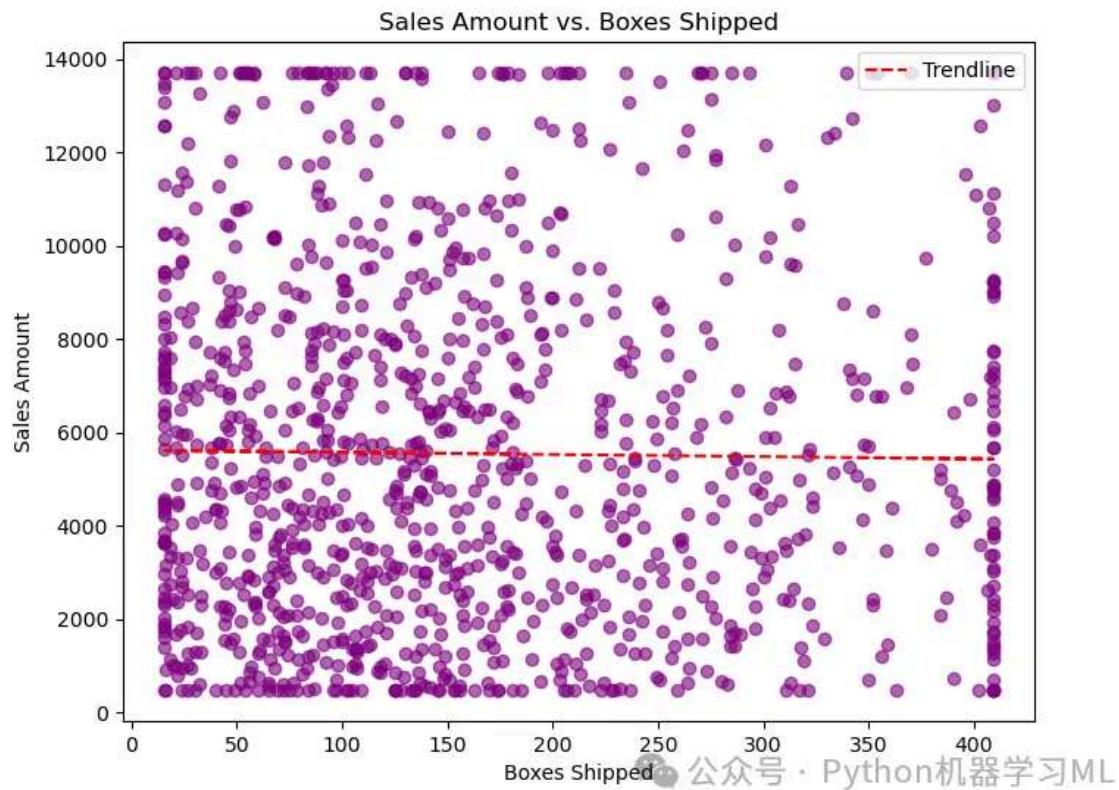
```
# plt.figure() 创建图形窗口  
plt.figure(figsize=(8, 6)) # 设置尺寸  
  
# --- 绘制散点图 ---  
# plt.scatter() 函数绘制散点图  
# 第一个参数是 x 轴数据 (df['Boxes Shipped'])  
# 第二个参数是 y 轴数据 (df['Amount'])  
# color='purple' 设置散点的颜色  
# alpha=0.6 设置散点的透明度 (0 完全透明, 1 完全不透明)。当点很多或重叠时, 使用半透明可以更好  
plt.scatter(df['Boxes Shipped'], df['Amount'], color='purple', alpha=0.6)  
plt.xlabel('Boxes Shipped') # 设置 x 轴标签  
plt.ylabel('Sales Amount') # 设置 y 轴标签  
plt.title('Sales Amount vs. Boxes Shipped') # 设置图表标题  
  
# --- 计算并绘制趋势线 ---  
# np.polyfit(x, y, deg) 函数执行最小二乘多项式拟合  
# x: x 坐标值 (df['Boxes Shipped'])  
# y: y 坐标值 (df['Amount'])  
# deg=1: 指定拟合多项式的阶数。1 表示线性拟合 ( $y = mx + c$ )。  
# 返回值 z 是一个包含多项式系数的数组, 对于线性拟合, z = [斜率 m, 截距 c]  
z = np.polyfit(df['Boxes Shipped'], df['Amount'], 1)  
# np.poly1d(z) 函数根据系数数组 z 创建一个多项式函数对象 p。现在你可以像调用函数一样使用 p(x)  
p = np.poly1d(z)
```

```

# plt.plot() 函数绘制拟合的趋势线
# x 轴数据仍然是原始的 df['Boxes Shipped']
# y 轴数据是使用拟合函数 p 计算出的对应 x 值的预测值 p(df['Boxes Shipped'])
# color='red' 设置线条颜色
# linestyle='--' 设置线条样式为虚线
# label='Trendline' 为这条线添加一个标签，用于图例显示
plt.plot(df['Boxes Shipped'], p(df['Boxes Shipped']), color='red', linestyle='--', label='Trendline')
plt.legend() # 显示图例框，其中会包含 'Trendline' 标签

plt.grid(True, linestyle='--', alpha=0.5) # 添加网格线，方便读数
plt.show() # 显示图形

```



应用到其他数据集：

- 选择变量：**
将 `df['Boxes Shipped']` 和 `df['Amount']` 替换为你想要探索其关系的任意两个数值列。
- 尝试非线性拟合：**
如果散点图显示出明显的曲线关系，你可以尝试更高阶的多项式拟合，只需将 `np.polyfit` 的 `deg` 参数改为大于 1 的整数（如 2 或 3）。但要注意高阶拟合可能导致过拟合。
- 使用 Seaborn：**
Seaborn 的 `sns.regplot(x='Boxes Shipped', y='Amount', data=df)` 或 `sns.lmplot(...)` 可以更方便地绘制散点图和回归拟合线（包括置信区间），通常代码更简洁，图形也更美观。
- 解释关系：**
观察散点的分布模式（是聚集、分散、线性、曲线、有无异常点？）和趋势线的斜率（正相关、负相关、或接近水平表示无线性关系）。结合相关系数一起判断关系强度。

阶段十八：可视化 - 基于编码数据的国家销售贡献（估算）

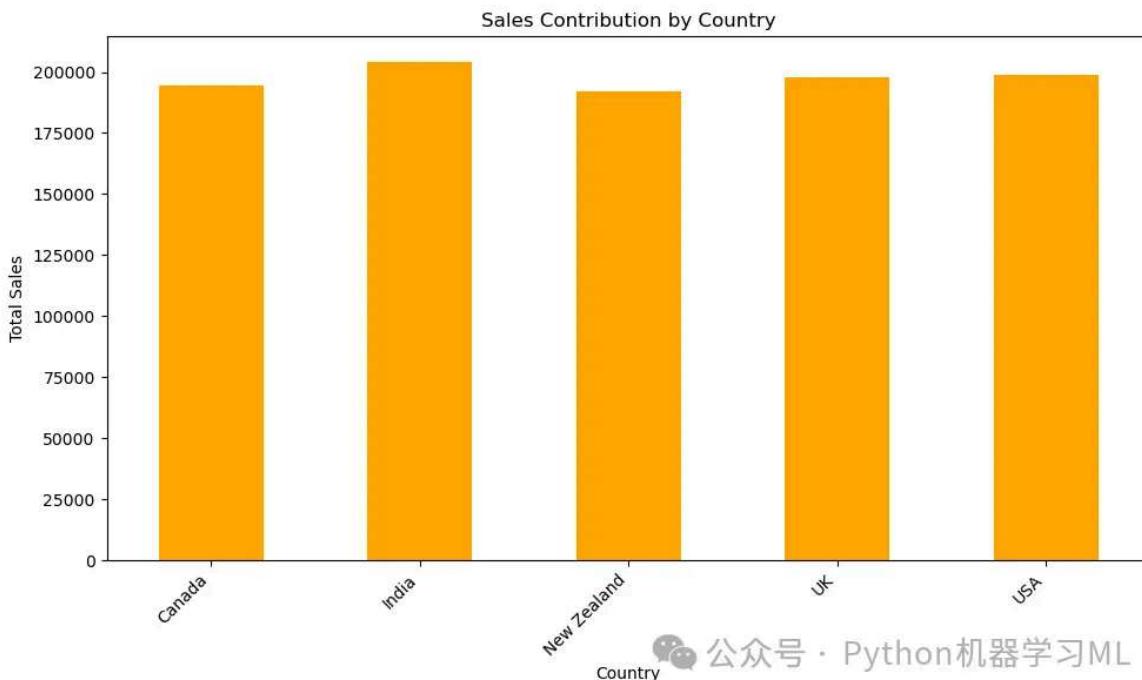
目的：类似于阶段十五对产品的处理，这里使用 One-Hot Encoding 后的国家列（`df_copy` 中以 'Country_开头的列）来估算并可视化各个国家的“总销售额”。同样，请注意这里的计算方式是基于（交易次数 * 平均销售额 / 国家数量）的估算，可能与直接按国家分组求和（`df.groupby('Country')['Amount'].sum()`）的结果不同。

python

```
# --- 提取与国家相关的 One-Hot 编码列 ---
# 使用列表推导式筛选出 df_copy 中列名以 'Country_' 开头的列
country_columns = [col for col in df_copy.columns if 'Country_' in col]

# --- 计算每个国家的估算“总销售额” ---
# 计算逻辑与阶段十五相同: (国家出现次数 * 数据集平均销售额 / 国家种类数)
# 结果 country_sales 是一个 Series, 索引是国家编码列名, 值是估算销售额。
country_sales = df_copy[country_columns].sum() * df_copy['Amount'].mean() / len(country_columns)

# --- 绘制国家销售贡献条形图 ---
# plt.figure() 创建图形窗口
plt.figure(figsize=(10, 6))
# 绘制条形图
# color='orange' 设置颜色
ax = country_sales.plot(kind='bar', color='orange')
# 设置 x 轴刻度标签, 从编码列名 (如 'Country_USA', 'Country_UK') 中提取国家名 ('USA', 'UK')
ax.set_xticklabels([col_name.split('_')[1] for col_name in country_columns], rotation=45)
plt.xlabel('Country') # x 轴标签
plt.ylabel('Total Sales (Estimated)') # y 轴标签, 注明是估算值
plt.title('Estimated Sales Contribution by Country') # 图表标题
plt.xticks(rotation=45, ha='right') # 旋转 x 轴标签并右对齐
plt.tight_layout() # 调整布局
plt.show() # 显示图形
```



应用到其他数据集：

- 修改列名前缀:

将 'Country_' 替换为你对应分类列 (如 'Region_', 'City_') 的 One-Hot Encoding 前缀。

- 调整可视化参数:

修改 `figsize`, `color`, 标签, 标题等。

- 选择计算方法:

再次强调, 如果需要精确的按国家/地区的总销售额, 应使用 `df.groupby('Country')['Amount'].sum()` 并基于该结果绘图。如果坚持使用当前代码的计算逻辑, 请确保理解其含义并清晰标注。

阶段十九：可视化 - 基于编码数据的销售人员业绩 (估算)

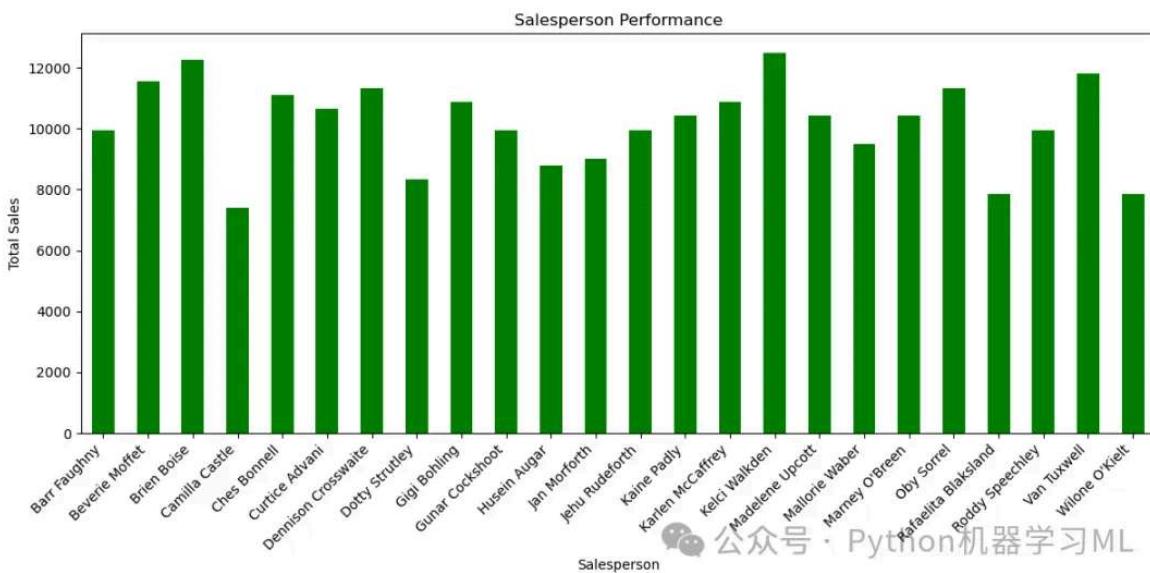
目的：采用与阶段十五和十八相同的逻辑，使用 One-Hot Encoding 后的销售人员列 (`df_copy` 中以 'Sales Person_' 开头的列) 来估算并可视化各位销售人员的“总销售额”或业绩。同样，**注意**这是一种基于 (交易次数 * 平均销售额 / 销售人员数量) 的估算方法。

python

```
# --- 提取与销售人员相关的 One-Hot 编码列 ---
# 筛选出 df_copy 中列名以 'Sales Person_' 开头的列
salesperson_columns = [col for col in df_copy.columns if 'Sales Person_' in col]

# --- 计算每个销售人员的估算“总销售额”/业绩 ---
# 计算逻辑同上：(销售人员处理的交易次数 * 数据集平均销售额 / 销售人员总数)
# 结果 salesperson_performance 是一个 Series
salesperson_performance = df_copy[salesperson_columns].sum() * df_copy['Amount'].mean() / len(df_copy)

# --- 绘制销售人员业绩条形图 ---
# plt.figure() 创建图形窗口
plt.figure(figsize=(12, 6))
# 绘制条形图
# color='green' 设置颜色
ax = salesperson_performance.plot(kind='bar', color='green')
# 设置 x 轴刻度标签，从编码列名 (如 'Sales Person_Alice', 'Sales Person_Bob') 中提取名字 ('Alice', 'Bob')
ax.set_xticklabels([col_name.split('_')[1] for col_name in salesperson_columns], rotation=45)
plt.xlabel('Salesperson') # x 轴标签
plt.ylabel('Total Sales (Estimated)') # y 轴标签，注明是估算值
plt.title('Estimated Salesperson Performance') # 图表标题
plt.xticks(rotation=45, ha='right') # 旋转 x 轴标签并右对齐
plt.tight_layout() # 调整布局
plt.show() # 显示图形
```



应用到其他数据集：

- 修改列名前缀：**
将 '`Sales Person_`' 替换为你对应分类列 (如 'Agent_', 'Employee_') 的 One-Hot Encoding 前缀。
- 调整可视化参数。**
- 选择计算方法：**
如前所述，若需精确业绩 (如总销售额)，应使用 `df.groupby('Sales Person')['Amount'].sum()` 并绘图。

阶段二十：可视化 - HoloViews Chord 图 (关系网络)

目的： 使用 HoloViews 库创建一个 Chord 图（弦图），这是一种用于可视化类别之间关系或流量的强大图表。在此例中，它展示了不同国家 ('Country') 和销售人员 ('Sales Person') 之间的联系强度，强度由他们之间发生的交易总额 ('Amount') 来表示。

```
python
```

```
# --- 初始化 HoloViews 的 Bokeh 后端 ---
# 这使得 HoloViews 图表能够利用 Bokeh 库生成交互式的 HTML 输出，可以在浏览器中查看和互动。
hv.extension('bokeh')

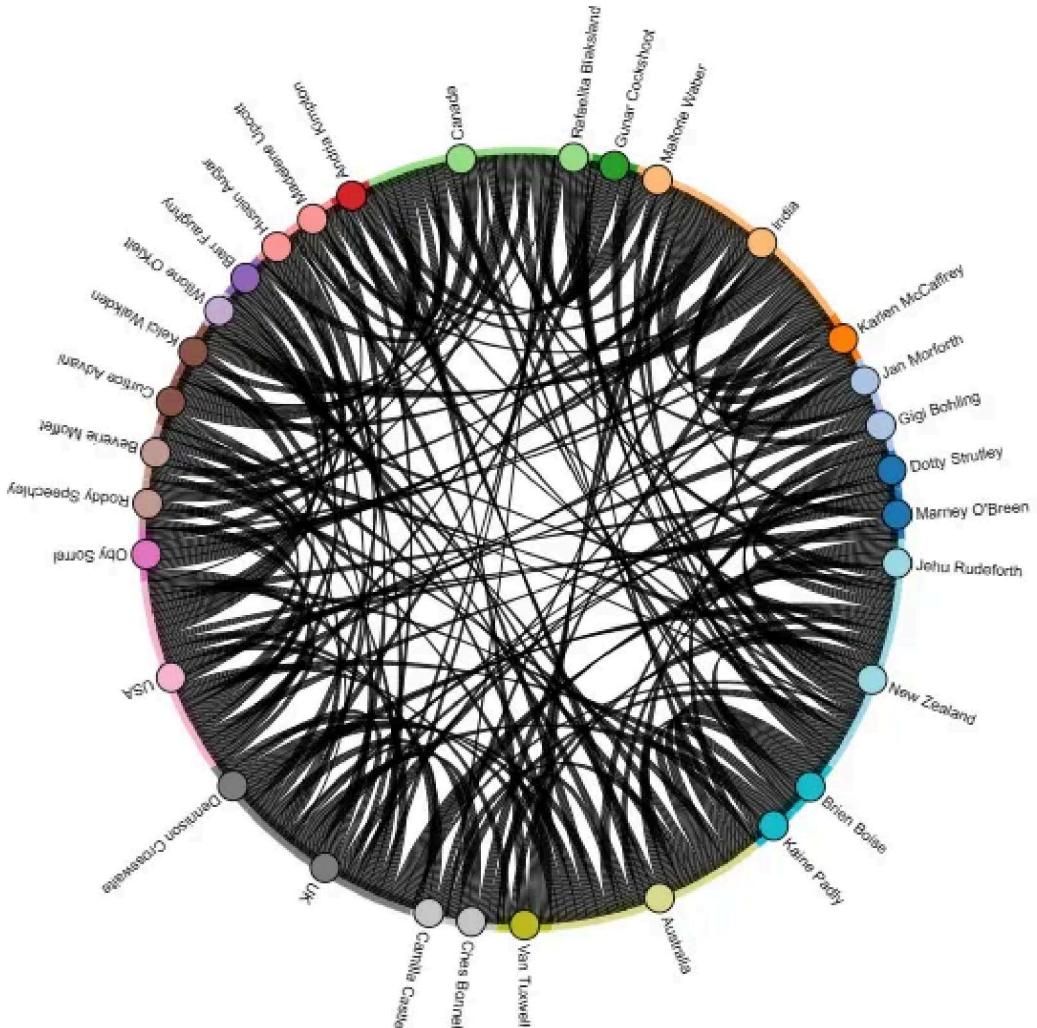
# --- 准备 Chord 图的数据 ---
# 1. 定义节点 (Nodes): Chord 图的圆周上的点代表不同的类别。
# 首先，获取所有唯一的国家名称和销售人员名称，合并成一个集合，再转换为列表，得到所有节点的桶
unique_labels = list(set(df['Country']).union(set(df['Sales Person'])))

# 然后，创建一个字典 label_map，将每个标签映射到一个从 0 开始的唯一整数 ID。这是 Chord 图内
label_map = {label: i for i, label in enumerate(unique_labels)}

# 2. 定义连接 (Links/Edges): 连接表示节点之间的关系或流量。
# 遍历原始 DataFrame df 的每一行。
# 对于每一行，提取 'Country'，'Sales Person'，和 'Amount'。
# 使用 label_map 将国家和销售人员名称转换为它们的整数 ID。
# 创建一个元组 (国家ID, 销售人员ID, 销售额)，表示从该国家流向该销售人员的流量（权重为销售额）
# 将所有这些元组收集到一个列表 links 中。
links = [(label_map[row['Country']], label_map[row['Sales Person']], row['Amount']) for _, row in df.iterrows()]

# 3. 创建节点数据集 (Nodes Dataset): HoloViews 需要一个明确的节点数据集。
# 创建一个 Pandas DataFrame，包含两列: 'index' (节点的整数 ID) 和 'label' (节点的原始名称)。
# 使用 hv.Dataset 将这个 DataFrame 包装起来，并指定 'index' 列作为关键维度 (kdims)。
nodes = hv.Dataset(pd.DataFrame({'index': list(label_map.values()), 'label': list(label_map.keys())}), kdims='index')

# --- 创建并配置 Chord 图 ---
# hv.Chord() 函数接收连接数据 (links) 和节点数据 (nodes) 作为输入，创建一个 Chord 图对象。
# .opts() 方法用于应用各种可视化配置选项。
chord = hv.Chord((links, nodes)).opts(
    # opts.Chord() 指定 Chord 图特有的选项:
    opts.Chord(labels='label',           # 指定使用节点数据中的 'label' 列来显示节点标签。
               cmap='Category20',      # 为节点设置颜色映射方案。'Category20' 是一组区分度较好的
               edge_cmap='viridis',     # 为连接边（弦）设置颜色映射方案。'viridis' 是一个常用的
               edge_color='Amount',     # 指定连接边的颜色深浅由其权重 ('Amount') 决定。权重越高
               node_color='index',       # 指定节点的颜色由其索引决定（通过上面设置的 cmap='Category20'）。
               node_size=20,             # 设置节点（圆周上的弧段）的大小（视觉上的粗细或长度可能
               width=800,                # 设置图表的宽度（像素）。
               height=800)              # 设置图表的高度（像素）。
)
# --- 显示 Chord 图 ---
# 在支持 HoloViews+Bokeh 的环境（如 Jupyter Notebook/Lab）中，直接写 Chord 图对象变量名即可渲染
chord
```



公众号 · Python机器学习ML

应用到其他数据集：

- 选择关系对：**

将 `'Country'` 和 `'Sales Person'` 替换为你想要可视化其相互关系的两个分类列。

- 选择权重列：**

将 `row['Amount']` 替换为表示这两个类别间联系强度或流量大小的数值列。

- 调整外观：**

探索 `opts.Chord` 中的其他参数来定制 Chord 图的外观，例如：

- `edge_line_width`

: 边线宽度。

- `node_line_color`

, `node_fill_color` : 节点边框和填充颜色。

- 不同的 `cmap` 和 `edge_cmap` 选项。

- `label_text_font_size`

: 标签字体大小。

- 理解图表：**

Chord 图中，每个类别对应圆周上的一段弧。弧的长度通常表示该类别的总流量（流入+流出）。节点之间的弦（连接带）表示它们之间的流量，弦的宽度通常与流量大小成正比。颜色可以用来区分节点或表示流量强度。将鼠标悬停在节点或弦上通常可以显示更详细的信息（交互性）。

阶段二十一：最终特征编码（用于模型训练）

目的：对原始 DataFrame `df` 进行最终的 One-Hot Encoding 处理，这次包括了之前创建的 'Month' 列，并将结果直接覆盖回 `df`。这一步是为了生成一个完全数值化的特征矩阵，可以直接作为大多数机器学习模型的输入。

python

```
# --- 识别需要编码的分类列（包括 Month）---
# 定义一个列表 categorical_cols, 包含所有需要在最终特征矩阵中进行 One-Hot Encoding 的原始分类
# 注意这次加入了 'Month' 列，意味着月份也被当作一个分类特征来处理（每个月是一个独立的类别）。
categorical_cols = ['Sales Person', 'Country', 'Product', 'Month']

# --- 对原始 DataFrame 应用 One-Hot Encoding 并覆盖 ---
# 再次使用 pd.get_dummies() 对原始 DataFrame df 进行编码
# columns=categorical_cols 指定要编码的列
# drop_first=True 同样用于可能的多重共线性问题
# dtype=float 指定新生成的编码列的数据类型为浮点数。这对于 Scikit-learn 等库中的某些模型或流程
# 最关键的是，这次编码的结果直接赋值回了变量 df，这意味着原始的分类列 ('Sales Person', 'Country')
# 替换为它们对应的 One-Hot 编码列。原始的数值列（如 'Amount', 'Boxes Shipped', 'Year', 'Profit'）
# 替换为它们对应的 One-Hot 编码列。原始的数值列（如 'Amount', 'Boxes Shipped', 'Year', 'Profit'）
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True, dtype=float)

# --- 显示最终处理后的 DataFrame ---
# 在支持的环境中，这会显示处理后的 DataFrame df 的内容。
# 此时的 df 应该只包含数值类型的列，包括原始数值列和所有新生成的 One-Hot 编码列。
# 这个 DataFrame 通常就是准备用于划分训练集/测试集并输入到机器学习模型中的最终特征集 (X)。
# （目标变量，如 'Amount' 或 'Profit'，在划分时会单独分离出来作为 y）。
df
```

应用到其他数据集：

- **确定最终特征集：**

仔细检查 `categorical_cols` 列表，确保它包含了所有你打算输入模型的原始分类特征。如果某个分类特征不应进入模型（例如，ID 列、或者与目标变量高度相关但未来不可用的列），就不应包含在此列表中。

- **考虑月份的处理方式：**

将月份作为纯粹的分类变量进行 One-Hot Encoding 是一种方法。但月份具有周期性和顺序性，有时也可以考虑其他处理方式，例如：

- 周期性编码：使用 `sin` 和 `cos` 函数将其转换为两个数值特征，以捕捉其周期性 (`np.sin(2 * np.pi * df['Month']/12)` , `np.cos(2 * np.pi * df['Month']/12)`)。
- 保持为数值：如果模型能处理有序数值特征，并且你认为月份的数值大小（1 到 12）本身有意义，也可以直接保留为数值。

- **数据类型 `dtype` :**

设置为 `float` 通常是安全的，但某些模型（如 LightGBM, CatBoost）可以直接处理整数类型的类别编码，有时 `int` 类型也可以。

第二部分(阶段 22-32):

这部分代码是机器学习流程的核心：将数据准备好后，定义任务（特征与目标），划分数据以进行可靠评估，尝试多种模型，评估它们的初始性能，然后利用自动化工具（Optuna）进行超参数搜索以提升性能，最后重新评估优化后的模型并进行可视化比较。整个过程清晰地展示了如何系统地训练、评估和优化机器学习模型，并提供了可以灵活应用于其他回归问题的代码框架。在应用时，关键在于正确选择目标变量、特征、评估指标，以及合理设置超参数搜索空间和试验次数。

阶段二十二：定义特征 (X) 和目标变量 (y)

目的：将经过预处理和编码的 DataFrame `df` 分成两部分：特征矩阵 `x` (模型用来学习的输入变量) 和目标向量 `y` (模型需要预测的变量)。

python

```
# --- 定义特征 (X) 和目标 (y) ---
# 从 DataFrame df 中创建特征矩阵 X
# df.drop(...) 方法用于移除指定的列
# ['Amount', 'Year', 'Date', 'Profit'] 是一个包含要移除的列名的列表：
#   - 'Amount': 这是我们要预测的目标变量，所以不能作为输入特征。
#   - 'Year': 年份信息。虽然在之前的EDA中有用，但这里决定不将其直接作为模型的输入特征。可能是因
#   - 'Date': 原始的日期时间对象列。它已经被分解为 'Month' 和 'Year'，并且 'Month' 被编码了，且
#   - 'Profit': 这是基于 'Amount' 计算出来的，如果目标是预测 'Amount'，那么 'Profit' 包含了目
# axis=1 参数指定沿着列的方向进行删除操作。
X = df.drop(['Amount', 'Year', 'Date', 'Profit'], axis=1)

# 将 DataFrame df 中的 'Amount' 列选作目标变量 y。
# y 是一个 Pandas Series，包含了模型需要学习预测的值。
y = df['Amount']
```

应用到其他数据集：

- **选择目标变量 (y)：**

将 `'Amount'` 替换为你数据集中你想要模型预测的那个列的名称。目标变量通常是你最关心的业务指标。

- **选择特征 (X)：**

- 唯一标识符列 (ID columns)。
- 已经被分解或编码的原始列 (如本例中的 'Date')。
- 与目标变量高度相关但可能在预测时不具备的特征 (未来信息泄露)。
- 基于领域知识判断不相关的特征。
- 首先，`y` 列必须从 `x` 中移除。
- 其次，移除任何直接或间接包含目标信息的列 (防止数据泄露)，例如在本例中移除基于 'Amount' 计算的 'Profit'。
- 移除不需要或不适合作为模型输入的列，例如：
- 你需要仔细检查 `df.columns` 并决定哪些列应该保留在 `x` 中，哪些应该通过 `df.drop()` 移除。

阶段二十三：数据划分 (训练集与测试集)

目的：将特征矩阵 `x` 和目标向量 `y` 分割成两个子集：训练集 (`x_train`, `y_train`) 和测试集 (`x_test`, `y_test`)。模型将在训练集上学习模式，然后在测试集上进行评估，以模拟模型在未见过的新数据上的表现。

python

```
# --- 将数据划分为训练集和测试集 ---
# 使用 Scikit-learn 的 train_test_split 函数进行划分
# X: 特征矩阵
# y: 目标向量
# test_size=0.2: 指定测试集所占的比例。0.2 表示将 20% 的数据划分为测试集，剩下的 80% 作为训练集
# random_state=42: 设置随机数生成器的种子。这确保了每次运行代码时，划分结果都是相同的，使得实验
# 函数返回四个部分：训练集特征、测试集特征、训练集目标、测试集目标
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --- 打印结果数据集的形状以验证划分 ---
# .shape 属性返回数据集的维度 (行数, 列数)
print("X_train shape:", X_train.shape) # 打印训练集特征的形状
print("X_test shape:", X_test.shape) # 打印测试集特征的形状
print("y_train shape:", y_train.shape) # 打印训练集目标的形状 (通常只有行数)
```

```
print("y_test shape:", y_test.shape) # 打印测试集目标的形状（通常只有行数）
# 检查行数是否匹配：X_train 和 y_train 的行数应相等，X_test 和 y_test 的行数应相等。
# 检查列数是否匹配：X_train 和 X_test 的列数应相等（都是特征的数量）。
# 检查总行数：X_train 行数 + X_test 行数应等于原始 x 的总行数。
```

应用到其他数据集：

- **调整测试集比例 (test_size)：**

常用的比例有 0.2, 0.25, 0.3。选择取决于你的数据集大小。数据量很大时，可以适当减小测试集比例；数据量很小时，可能需要使用交叉验证等技术，或者增大测试集比例以获得更可靠的评估。

- **设置 random_state :**

强烈建议设置此参数以保证结果的可复现性。

- **分层抽样 (Stratified Splitting):**

对于分类问题，如果目标变量的类别分布不均衡，建议在 `train_test_split` 中加入 `stratify=y` 参数，以确保训练集和测试集中目标类别的比例与原始数据大致相同。对于回归问题，通常不直接使用分层，但如果目标变量分布非常偏斜，有时会考虑基于目标变量分箱后的结果进行分层。

阶段二十四：初始化模型 (使用默认参数)

目的： 创建将要训练和比较的几种不同类型的回归模型的实例。在这一步，我们主要使用模型的默认超参数，或者只设置一个 `random_state` 来保证涉及随机性的模型（如决策树、随机森林、XGBoost、LightGBM、CatBoost中的某些内部操作）行为一致。

python

```
# --- 初始化模型 ---
# 创建决策树回归器模型的实例
# random_state=42 确保树的构建过程中的随机性是可复现的
dt_model = DecisionTreeRegressor(random_state=42)

# 创建随机森林回归器模型的实例
# 随机森林是多个决策树的集成，random_state 控制森林中所有树的构建以及特征抽样的随机性
rf_model = RandomForestRegressor(random_state=42)

# 创建 XGBoost 回归器模型的实例
# random_state 控制模型内部可能存在的随机性（例如特征子采样）
xgb_model = XGBRegressor(random_state=42)

# 创建 LightGBM 回归器模型的实例
# random_state 控制其内部随机性
lgb_model = LGBMRegressor(random_state=42)

# 创建 CatBoost 回归器模型的实例
# random_state 控制随机性
# verbose=0 参数设置 CatBoost 在训练过程中不打印任何输出信息（保持控制台清洁）
catb_model = CatBoostRegressor(random_state=42, verbose=0)
```

应用到其他数据集：

- **选择模型：**

你可以根据你的问题类型、数据特性（大小、是否有类别特征、是否需要解释性等）来选择要尝试的模型。这里包含了常用的树模型和梯度提升模型。你也可以加入其他类型的模型，如线性回归（`LinearRegression`）、支持向量回归（`SVR`）、K近邻回归（`KNeighborsRegressor`）等（需要先 `from sklearn.linear_model import LinearRegression` 等导入）。

- **设置 random_state :**

强烈建议为所有包含随机性的模型设置相同的 `random_state`，以确保结果可比较和复现。

- **初步调整参数：**

虽然这里主要用默认参数，但有时某些默认参数可能完全不适用（例如，默认迭代次数过少）。你可以根据对模型的了解进行非常初步的调整，但主要的参数调优将在后续阶段进行。

阶段二十五：训练初始模型

目的： 使用训练数据集 (`X_train`, `y_train`) 来训练（拟合）之前初始化的每个模型。模型通过学习训练数据中的模式来调整其内部参数。同时，记录每个模型的训练时间。

```
python
```

```
# --- 训练决策树模型 ---
start_time = time.time() # 记录训练开始时间
# 调用模型的 fit 方法进行训练
# X_train: 训练集特征
# y_train: 训练集目标
dt_model.fit(X_train, y_train)
end_time = time.time() # 记录训练结束时间
# 计算训练耗时并打印，格式化为保留两位小数的秒数
print(f"Decision Tree training time: {end_time - start_time:.2f} seconds")

# --- 训练随机森林模型 ---
start_time = time.time() # 记录开始时间
rf_model.fit(X_train, y_train) # 使用训练数据拟合随机森林模型
end_time = time.time() # 记录结束时间
print(f"Random Forest training time: {end_time - start_time:.2f} seconds") # 打印训练时间

# --- 训练 XGBoost 模型 ---
start_time = time.time() # 记录开始时间
xgb_model.fit(X_train, y_train) # 使用训练数据拟合 XGBoost 模型
end_time = time.time() # 记录结束时间
print(f"XGBoost training time: {end_time - start_time:.2f} seconds") # 打印训练时间

# --- 训练 LightGBM 模型 ---
start_time = time.time() # 记录开始时间
lgb_model.fit(X_train, y_train) # 使用训练数据拟合 LightGBM 模型
end_time = time.time() # 记录结束时间
print(f"LightGBM training time: {end_time - start_time:.2f} seconds") # 打印训练时间

# --- 训练 CatBoost 模型 ---
start_time = time.time() # 记录开始时间
catb_model.fit(X_train, y_train) # 使用训练数据拟合 CatBoost 模型
end_time = time.time() # 记录结束时间
print(f"CatBoost training time: {end_time - start_time:.2f} seconds") # 打印训练时间

print("Models trained successfully.") # 打印一条消息表示所有模型训练完成
```

应用到其他数据集：

- **通用流程：**

`model.fit(X_train, y_train)` 是 Scikit-learn 兼容模型进行训练的标准方法，适用于你选择的任何模型。

- **训练时间：**

记录训练时间有助于了解不同模型的计算开销，特别是在处理大数据集或需要频繁重新训练的场景下，这可能是一个重要的考虑因素。

- **监控输出：**

对于某些模型（如 XGBoost, LightGBM, CatBoost），你可以通过设置 `verbose` 参数来控制训练过程中输出信息的详细程度，这有助于监控训练进度和发现潜在问题。这里 CatBoost 设置了 `verbose=0` 关闭了输出。

阶段二十六：评估初始模型

目的：使用训练好的模型对从未见过的测试集（`X_test`）进行预测，并将预测结果与真实的测试集目标（`y_test`）进行比较，以评估模型的泛化能力。计算常用的回归评估指标来量化模型性能。

python

```
# --- 进行预测 ---
# 使用训练好的决策树模型对测试集特征 X_test 进行预测
dt_predictions = dt_model.predict(X_test)
# 使用训练好的随机森林模型对测试集特征 X_test 进行预测
rf_predictions = rf_model.predict(X_test)
# 使用训练好的 XGBoost 模型对测试集特征 X_test 进行预测
xgb_predictions = xgb_model.predict(X_test)
# 使用训练好的 CatBoost 模型对测试集特征 X_test 进行预测
catb_predictions = catb_model.predict(X_test)
# LightGBM 的预测
lgbm_predictions = lgb_model.predict(X_test) # 添加 LightGBM 的预测

# --- 计算并打印评估指标 ---
# --- 决策树 ---
# mean_squared_error(y_true, y_pred) 计算均方误差 (MSE)，值越小越好
dt_mse = mean_squared_error(y_test, dt_predictions)
# mean_absolute_error(y_true, y_pred) 计算平均绝对误差 (MAE)，值越小越好，单位与目标变量相同
dt_mae = mean_absolute_error(y_test, dt_predictions)
# r2_score(y_true, y_pred) 计算 R 方 (决定系数)，表示模型解释方差的比例，值越接近 1 越好 (可忽略)
dt_r2 = r2_score(y_test, dt_predictions)
print(f"Decision Tree MSE: {dt_mse:.4f}, MAE: {dt_mae:.4f}, R2: {dt_r2:.4f}") # 打印决策树的评估指标

# --- 随机森林 ---
rf_mse = mean_squared_error(y_test, rf_predictions) # 计算 MSE
rf_mae = mean_absolute_error(y_test, rf_predictions) # 计算 MAE
rf_r2 = r2_score(y_test, rf_predictions) # 计算 R2
print(f"Random Forest MSE: {rf_mse:.4f}, MAE: {rf_mae:.4f}, R2: {rf_r2:.4f}") # 打印随机森林的评估指标

# --- XGBoost ---
xgb_mse = mean_squared_error(y_test, xgb_predictions) # 计算 MSE
xgb_mae = mean_absolute_error(y_test, xgb_predictions) # 计算 MAE
xgb_r2 = r2_score(y_test, xgb_predictions) # 计算 R2
print(f"XGBoost MSE: {xgb_mse:.4f}, MAE: {xgb_mae:.4f}, R2: {xgb_r2:.4f}") # 打印 XGBoost 的评估指标

# --- LightGBM ---
# 应使用 lgbm_predictions。
lgbm_mse = mean_squared_error(y_test, lgbm_predictions)

lgbm_mae = mean_absolute_error(y_test, lgbm_predictions)

lgbm_r2 = r2_score(y_test, lgbm_predictions)

print(f"LightGBM MSE: {lgbm_mse:.4f}, MAE: {lgbm_mae:.4f}, R2: {lgbm_r2:.4f}") # 打印 LightGBM 的评估指标

# --- CatBoost ---
catb_mse = mean_squared_error(y_test, catb_predictions) # 计算 MSE
catb_mae = mean_absolute_error(y_test, catb_predictions) # 计算 MAE
catb_r2 = r2_score(y_test, catb_predictions) # 计算 R2
print(f"CatBoost MSE: {catb_mse:.4f}, MAE: {catb_mae:.4f}, R2: {catb_r2:.4f}") # 打印 CatBoost 的评估指标

# --- 将结果汇总到字典 ---
# 将所有模型的评估指标汇总到一个字典中
model_results = {
    "Decision Tree": {"MSE": dt_mse, "MAE": dt_mae, "R2": dt_r2},
    "Random Forest": {"MSE": rf_mse, "MAE": rf_mae, "R2": rf_r2},
    "XGBoost": {"MSE": xgb_mse, "MAE": xgb_mae, "R2": xgb_r2},
    "LightGBM": {"MSE": lgbm_mse, "MAE": lgbm_mae, "R2": lgbm_r2},
    "CatBoost": {"MSE": catb_mse, "MAE": catb_mae, "R2": catb_r2}
}
```

```

# 创建一个字典 model_results 用于存储每个模型的评估指标
model_results = {
    'Decision Tree': {'MSE': dt_mse, 'MAE': dt_mae, 'R2': dt_r2}, # 存储决策树结果
    'Random Forest': {'MSE': rf_mse, 'MAE': rf_mae, 'R2': rf_r2}, # 存储随机森林结果
    'XG Boost': {'MSE': xgb_mse, 'MAE': xgb_mae, 'R2': xgb_r2}, # 存储 XGBoost 结果
    'LGBM Boost': {'MSE': lgbm_mse, 'MAE': lgbm_mae, 'R2': lgbm_r2}, # 存储 LightGBM 结果
    'Cat Boost': {'MSE': catb_mse, 'MAE': catb_mae, 'R2': catb_r2}, # 存储 CatBoost 结果
}

print("\nInitial Model Results:") # 打印提示信息
print(model_results) # 打印包含所有模型评估结果的字典

```

应用到其他数据集：

- **通用评估流程:**

`model.predict(X_test)` 和使用 `sklearn.metrics` 中的函数计算指标是标准的评估流程。

- **选择评估指标:**

- **MSE:**

对大误差给予更高的惩罚（因为误差被平方）。单位是目标变量单位的平方，不易直观解释。

- **MAE:**

对所有误差给予相同的权重。单位与目标变量相同，更易于理解。对异常值没有 MSE 敏感。

- **RMSE (Root Mean Squared Error):**

`np.sqrt(mean_squared_error(y_test, predictions))` 是 MSE 的平方根，单位与目标变量相同，结合了 MSE 对大误差敏感和单位易解释的特点。

- **R²:**

衡量模型解释方差的能力，范围通常在 0 到 1 之间（最佳为 1），但可能为负数（表示模型比直接预测平均值还差）。易于比较不同模型，但不提供误差大小信息。

- **MAPE (Mean Absolute Percentage Error):**

`np.mean(np.abs((y_test - predictions) / y_test)) * 100` 计算平均绝对百分比误差，易于解释，但对 `y_test` 接近零的值非常敏感，且当 `y_test` 为零时无法计算。你需要根据你的业务目标和数据特性选择最合适的评估指标。例如，如果对大额预测错误非常敏感，MSE 或 RMSE 可能更合适；如果需要向非技术人员解释误差大小，MAE 或 RMSE（带单位）更好；如果想知道模型解释了多少变异性，看 R²。

- **检查代码逻辑:**

在复制粘贴评估代码时，务必确保每个模型的指标是使用其对应的预测结果（`*_predictions`）计算的，避免像示例中 LightGBM 那样的错误。

阶段二十七：可视化初始模型性能

目的： 使用条形图直观地比较各个模型在测试集上的性能（以 MSE 为例）。这有助于快速识别哪些模型在默认参数下表现相对较好或较差。

python

```

# --- 提取模型名称和 MSE 值 ---
model_names = list(model_results.keys()) # 从结果字典中获取所有模型的名称列表
mse_values = [model_results[model]['MSE'] for model in model_names] # 使用列表推导式从结果字

# --- 创建并显示条形图 ---
plt.figure(figsize=(10, 6)) # 创建图形窗口，设置尺寸（宽度10，高度6）
# plt.bar() 函数绘制条形图
# model_names: x 轴上的类别（模型名称）
# mse_values: 每个类别对应的条形高度（MSE 值）
# color=[...] 指定每个条形的颜色列表
plt.bar(model_names, mse_values, color=['skyblue', 'lightgreen', 'lightcoral', 'gold', 'mediu
plt.xlabel("Model") # 设置 x 轴标签
plt.ylabel("Mean Squared Error (MSE)") # 设置 y 轴标签

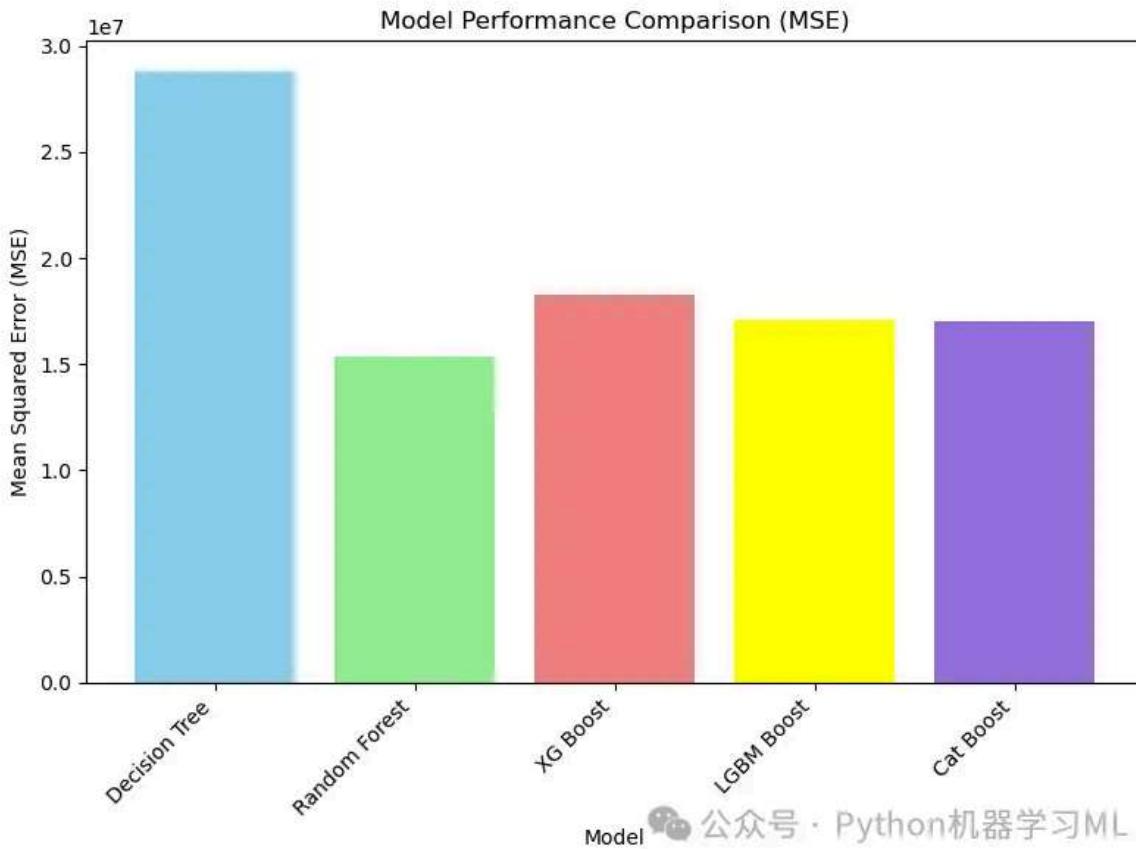
```

```

plt.title("Initial Model Performance Comparison (MSE)") # 设置图表标题
plt.xticks(rotation=45, ha="right") # 将 x 轴刻度标签旋转 45 度，并右对齐，以防重叠
plt.ylim(bottom=0, top=max(mse_values) * 1.1) # 设置 y 轴范围，从 0 开始，上限比最大 MSE 稍高
plt.grid(axis='y', linestyle='--', alpha=0.7) # 添加水平网格线
plt.tight_layout() # 自动调整布局
plt.show() # 显示图形

print("\nThe bar chart visualizes the Mean Squared Error (MSE) for each model trained with de")
print("Lower MSE indicates better performance on the test set.") # 解释 MSE 的含义：越低越好

```



应用到其他数据集：

- 可视化不同指标：**

你可以轻松修改代码来可视化 MAE 或 R^2 。只需更改提取值的代码（例如 `mae_values = [model_results[model]['MAE'] for model in model_names]`）和相应的 `ylabel`、`title`。注意 R^2 是越高越好，而 MSE/MAE 是越低越好，绘图时可能需要调整说明或 y 轴方向。

- 改变图表类型：**

除了条形图，你也可以使用点图（`plt.scatter`）或其他类型的图表来展示结果。

- 添加数值标签：**

可以通过循环遍历 `ax.patches`（在 `ax = plt.bar(...)` 之后）并使用 `ax.text()` 在每个条形上方添加具体的 MSE 数值标签，使图表信息更丰富。

- 排序：**

可以考虑在绘图前根据性能指标对模型进行排序（`sorted_results = sorted(model_results.items(), key=lambda item: item[1]['MSE'])`），然后按排序后的顺序绘图，使性能最好的模型更突出。

阶段二十八：设置 Optuna 超参数优化

目的： 定义使用 Optuna 库进行超参数自动优化的框架。这包括为每个待优化的模型定义一个“目标函数”（objective function），该函数描述了如何根据 Optuna 提供的参数建议来构建模型、使用交叉验证评估模型性能，并返回一个需要被 Optuna 最小化（或最大化）的分数。

```
python
```

```
# --- 定义 Optuna 的目标函数 ---
# 这个函数会被 Optuna 反复调用，每次调用传入一个 trial 对象和模型名称
# trial 对象用于建议本次试验要尝试的超参数组合
def objective(trial, model_name):
    # 使用 global 关键字确保函数内部可以访问在函数外部定义的 X_train 和 y_train 变量
    # 注意：虽然 global 在这里能工作，但更好的实践可能是将 X_train, y_train 作为参数传递给 objective
    global X_train, y_train

    # --- 根据 model_name 定义参数搜索空间并创建模型实例 ---
    if model_name == "DecisionTree":
        # 定义决策树的超参数搜索空间
        params = {
            # trial.suggest_int(name, low, high) 建议一个整数参数
            "max_depth": trial.suggest_int("max_depth", 2, 10), # 建议树的最大深度在 2 到 10 之间
            "min_samples_split": trial.suggest_int("min_samples_split", 2, 20), # 建议节点分裂所需的最小样本数
            "min_samples_leaf": trial.suggest_int("min_samples_leaf", 1, 20), # 建议叶节点所需的最小样本数
        }
        # 使用建议的参数创建决策树回归器实例
        model = DecisionTreeRegressor(**params, random_state=42) # **params 将字典解包为关键字参数

    elif model_name == "RandomForest":
        # 定义随机森林的超参数搜索空间
        params = {
            "n_estimators": trial.suggest_int("n_estimators", 50, 300), # 建议森林中树的数量在 50 到 300 之间
            "max_depth": trial.suggest_int("max_depth", 2, 10), # 建议每棵树的最大深度
            "min_samples_split": trial.suggest_int("min_samples_split", 2, 20), # 建议节点分裂最小样本数
            "min_samples_leaf": trial.suggest_int("min_samples_leaf", 1, 20), # 建议叶节点最小样本数
        }
        # 创建随机森林实例
        model = RandomForestRegressor(**params, random_state=42)

    elif model_name == "XGBoost":
        # 定义 XGBoost 的超参数搜索空间
        params = {
            "n_estimators": trial.suggest_int("n_estimators", 50, 300), # 建议 boosting 轮数（树的数量）
            "max_depth": trial.suggest_int("max_depth", 2, 10), # 建议每棵树的最大深度
            # trial.suggest_float(name, low, high, log=False) 建议一个浮点数参数
            "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3), # 建议学习率（步长）
            "subsample": trial.suggest_float("subsample", 0.5, 1.0), # 建议训练每棵树时使用的样本比例
            "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0), # 建议训练每棵树时使用
        }
        # 创建 XGBoost 实例
        model = XGBRegressor(**params, random_state=42, objective='reg:squarederror', booster='gbtree')

    elif model_name == "LightGBM":
        # 定义 LightGBM 的超参数搜索空间
        params = {
            "n_estimators": trial.suggest_int("n_estimators", 50, 300), # 建议 boosting 轮数
            "max_depth": trial.suggest_int("max_depth", -1, 10), # 建议最大深度 (-1 表示无限制)
            "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3), # 建议学习率
            "num_leaves": trial.suggest_int("num_leaves", 10, 200), # 建议每棵树的叶子节点数 (LightGBM 关
            "subsample": trial.suggest_float("subsample", 0.5, 1.0), # 建议样本采样比例
            # 'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0), # 可以添加特征采样
        }
        # 创建 LightGBM 实例

```

```

# verbose=-1 禁止 LightGBM 在交叉验证过程中打印任何信息
model = LGBMRegressor(**params, random_state=42, verbose=-1)

elif model_name == "CatBoost":
    # 定义 CatBoost 的超参数搜索空间
    params = {
        "iterations": trial.suggest_int("iterations", 50, 300), # 建议 boosting 轮数 (CatBoost 中叫 iterations)
        "depth": trial.suggest_int("depth", 2, 10), # 建议树的深度 (CatBoost 中叫 depth)
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3), # 建议学习率
        "l2_leaf_reg": trial.suggest_float('l2_leaf_reg', 1e-3, 10.0, log=True), # 可以添加 L2 正则
    }
    # 创建 CatBoost 实例
    # verbose=0 禁止 CatBoost 在交叉验证中打印信息
    model = CatBoostRegressor(**params, random_state=42, verbose=0)

    # --- 使用交叉验证评估当前参数组合的性能 ---
    # cross_val_score 在训练集 (X_train, y_train) 上执行 k 折交叉验证 (这里 cv=5)
    # model: 使用当前 trial 建议参数创建的模型实例
    # X_train, y_train: 训练数据
    # cv=5: 执行 5 折交叉验证。数据被分成 5 份，每次用 4 份训练，1 份验证，重复 5 次。
    # scoring="neg_mean_squared_error": 指定评估指标为负均方误差。Scikit-learn 的评分函数通常是越大越好，而 MSE 是越小越好。使用负 MSE (neg_mean_squared_error) 可以将最小化 MSE 的问题转化为最大化问题。
    # 或者在 Optuna 中最小化其相反数。
    # .mean(): 计算 5 折交叉验证得到的 5 个 neg_mean_squared_error 分数的平均值。
    score = cross_val_score(model, X_train, y_train, cv=5, scoring="neg_mean_squared_error")

    # Optuna 默认执行最小化操作。因为我们使用了负 MSE，它本身是越小越好（绝对值越大越好），所以我们需要返回它的相反数（即正 MSE），这样 Optuna 就会去最小化这个正 MSE。
    return -score

    # --- 定义运行 Optuna 优化的辅助函数 ---
    # 这个函数封装了为单个模型运行 Optuna 优化的过程
def tune_model(model_name, n_trials=20): # n_trials=20 设置 Optuna 尝试的总次数
    print(f"\nStarting hyperparameter tuning for {model_name}...") # 打印开始信息
    study = optuna.create_study(direction="minimize") # 创建一个 Optuna 的 study 对象。
    # direction="minimize" 告诉 Optuna 目标是最小化 objective 函数的返回值（也就是我们返回的正 MSE）
    study.optimize(lambda trial: objective(trial, model_name), n_trials=n_trials)

    # 优化完成后，打印找到的最佳参数组合
    print(f"Best parameters found for {model_name}: {study.best_params}")
    print(f"Best score (MSE) achieved during tuning: {-study.best_value:.4f}") # 打印最佳分数（转为正数）
    # 返回找到的最佳参数字典
    return study.best_params

```

应用到其他数据集：

- **调整搜索空间：**

这是最关键的部分。你需要根据你对模型的理解、数据的特性以及初步运行的结果来调整 `trial.suggest_*` 中的参数范围 (`low`, `high`)。范围太小可能错过最优解，范围太大则需要更多 `n_trials` 才能有效搜索。可以添加或删除要调整的超参数。查阅每个模型库的文档了解其重要参数及其合理范围。

- **修改评估指标 (`scoring`)：**

你可以在 `cross_val_score` 中使用 Scikit-learn 支持的其他评分指标，如 `"neg_mean_absolute_error"`, `"r2"` 等。如果使用 `"r2"`，因为 R^2 是越大越好，而 Optuna 默认最小化，你应该在 `objective` 函数中直接返回 `-score` (因为 `score` 是 R^2)，并将 `create_study` 的 `direction` 设置为 `"maximize"`。

- 调整交叉验证折数 (`cv`):

`cv=5` 是常用值。数据量较小时可以增加折数 (如 10)，数据量很大时可以减少折数 (如 3) 以加快速度。

- 调整试验次数 (`n_trials`):

`n_trials` 决定了优化的彻底程度。值越大，找到更好参数的可能性越高，但耗时也越长。通常需要根据计算资源和时间限制来权衡。可以从较小的值 (如 20-50) 开始，根据结果再决定是否增加。

- 优化 `objective` 函数:

避免在 `objective` 函数中使用 `global`，可以通过传递参数或使用类来封装。

阶段二十九：执行超参数优化

目的：调用之前定义的 `tune_model` 函数，为列表中的每种模型 (决策树、随机森林、XGBoost、LightGBM、CatBoost) 运行 Optuna 超参数搜索过程。

```
python
```

```
# --- 调整每个模型的超参数 ---
# 调用 tune_model 函数为 "DecisionTree" 模型进行优化，将返回的最佳参数存入 dt_params
dt_params = tune_model("DecisionTree", n_trials=50) # 增加试验次数到 50
# 为 "RandomForest" 模型进行优化，存入 rf_params
rf_params = tune_model("RandomForest", n_trials=50)
# 为 "XGBoost" 模型进行优化，存入 xgb_params
xgb_params = tune_model("XGBoost", n_trials=50)
# 为 "LightGBM" 模型进行优化，存入 lgb_params
lgb_params = tune_model("LightGBM", n_trials=50)
# 为 "CatBoost" 模型进行优化，存入 cat_params
cat_params = tune_model("CatBoost", n_trials=50)
```

应用到其他数据集：

- 选择要优化的模型:

你可以根据初步评估的结果或者计算资源的限制，选择只优化表现较好或你重点关注的模型。

- 调整 `n_trials` :

根据每个模型的复杂度和参数空间大小，以及你的时间预算，为不同的模型设置不同的 `n_trials` 值。通常梯度提升模型 (XGB, LGBM, CatBoost) 的参数空间更大，可能需要更多的试验次数。

阶段三十：使用优化后的参数重新训练模型

目的：使用 Optuna 找到的最佳超参数组合来创建新的模型实例，并在 整个训练数据集 (`X_train`, `y_train`) 上重新训练这些模型。

```
python
```

```
# --- 使用找到的最佳参数重新初始化模型 ---
# 使用 dt_params 字典中的最佳参数创建新的决策树实例
dt_model = DecisionTreeRegressor(**dt_params, random_state=42)
# 使用 rf_params 创建新的随机森林实例
rf_model = RandomForestRegressor(**rf_params, random_state=42)
# 使用 xgb_params 创建新的 XGBoost 实例
xgb_model = XGBRegressor(**xgb_params, random_state=42, objective='reg:squarederror', booster='gbtree')
# 使用 lgb_params 创建新的 LightGBM 实例
lgb_model = LGBMRegressor(**lgb_params, random_state=42, verbose=-1) # 保持 verbose=-1
# 使用 cat_params 创建新的 CatBoost 实例
```

```

# 注意: 这里变量名用的是 cat_model, 与之前的 catb_model 不同, 保持一致性会更好, 但代码能运行。
cat_model = CatBoostRegressor(**cat_params, random_state=42, verbose=0) # 保持 verbose=0

# --- 重新训练模型 ---
# 使用优化后的参数在整个训练集上训练决策树模型
print("\nRe-training models with optimized hyperparameters...")
start_time = time.time()
dt_model.fit(X_train, y_train)
end_time = time.time()
print(f"Optimized Decision Tree training time: {end_time - start_time:.2f} seconds")

# 训练优化后的随机森林模型
start_time = time.time()
rf_model.fit(X_train, y_train)
end_time = time.time()
print(f"Optimized Random Forest training time: {end_time - start_time:.2f} seconds")

# 训练优化后的 XGBoost 模型
start_time = time.time()
xgb_model.fit(X_train, y_train)
end_time = time.time()
print(f"Optimized XGBoost training time: {end_time - start_time:.2f} seconds")

# 训练优化后的 LightGBM 模型
start_time = time.time()
lgb_model.fit(X_train, y_train)
end_time = time.time()
print(f"Optimized LightGBM training time: {end_time - start_time:.2f} seconds")

# 训练优化后的 CatBoost 模型 (使用 cat_model 变量)
start_time = time.time()
cat_model.fit(X_train, y_train)
end_time = time.time()
print(f"Optimized CatBoost training time: {end_time - start_time:.2f} seconds")

print("\nModels re-trained successfully with optimized hyperparameters.")

```

应用到其他数据集:

- **通用流程:**

这个过程 (用找到的最佳参数实例化模型 -> 用 `fit(X_train, y_train)` 训练) 是标准的。

- **确保参数传递正确:**

使用 `**best_params` 的方式能方便地将 Optuna 返回的参数字典传递给模型构造函数。确保字典中的键与模型构造函数接受的参数名一致。

- **训练时间比较:**

比较优化后模型的训练时间与初始模型的训练时间。有时优化后的参数 (如更多的树 `n_estimators` 或更大的深度 `max_depth`) 可能导致训练时间增加。

阶段三十一：评估优化后的模型

目的: 使用经过超参数优化并重新训练过的模型，对测试集 (`x_test`) 进行预测，并重新计算评估指标，以了解超参数调整对模型在未见过数据上的性能提升效果。

python

```

# --- 使用优化后的模型进行预测 ---
# 使用优化后的决策树模型进行预测

```

```

dt_predictions = dt_model.predict(X_test)
# 使用优化后的随机森林模型进行预测
rf_predictions = rf_model.predict(X_test)
# 使用优化后的 XGBoost 模型进行预测
xgb_predictions = xgb_model.predict(X_test)
# 优化后的 LightGBM 模型进行预测
lgbm_predictions = lgb_model.predict(X_test) # 添加优化后 LightGBM 的预测
# 使用优化后的 CatBoost 模型进行预测 (使用 cat_model 变量)

catb_predictions = cat_model.predict(X_test) # 使用优化后的 CatBoost 模型预测

# --- 重新计算并打印评估指标 ---
print("\nEvaluating models with optimized hyperparameters on the test set:")
# --- 决策树 (优化后) ---
dt_mse = mean_squared_error(y_test, dt_predictions)
dt_mae = mean_absolute_error(y_test, dt_predictions)
dt_r2 = r2_score(y_test, dt_predictions)
print(f"Optimized Decision Tree MSE: {dt_mse:.4f}, MAE: {dt_mae:.4f}, R2: {dt_r2:.4f}")

# --- 随机森林 (优化后) ---
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_mae = mean_absolute_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)
print(f"Optimized Random Forest MSE: {rf_mse:.4f}, MAE: {rf_mae:.4f}, R2: {rf_r2:.4f}")

# --- XGBoost (优化后) ---
xgb_mse = mean_squared_error(y_test, xgb_predictions)
xgb_mae = mean_absolute_error(y_test, xgb_predictions)
xgb_r2 = r2_score(y_test, xgb_predictions)
print(f"Optimized XGBoost MSE: {xgb_mse:.4f}, MAE: {xgb_mae:.4f}, R2: {xgb_r2:.4f}")

# --- LightGBM (优化后) ---
lgbm_mse = mean_squared_error(y_test, lgbm_predictions)

lgbm_mae = mean_absolute_error(y_test, lgbm_predictions)

lgbm_r2 = r2_score(y_test, lgbm_predictions)

print(f"Optimized LightGBM MSE: {lgbm_mse:.4f}, MAE: {lgbm_mae:.4f}, R2: {lgbm_r2:.4f}")

# --- CatBoost (优化后) ---

catb_mse = mean_squared_error(y_test, catb_predictions)

catb_mae = mean_absolute_error(y_test, catb_predictions)

catb_r2 = r2_score(y_test, catb_predictions)

print(f"Optimized CatBoost MSE: {catb_mse:.4f}, MAE: {catb_mae:.4f}, R2: {catb_r2:.4f}")

# --- 将优化后的结果汇总到新字典 ---
# 创建一个新字典 opti_model_results 存储优化后模型的评估指标
opti_model_results = {
    'Decision Tree': {'MSE': dt_mse, 'MAE': dt_mae, 'R2': dt_r2},
    'Random Forest': {'MSE': rf_mse, 'MAE': rf_mae, 'R2': rf_r2},
    'XG Boost': {'MSE': xgb_mse, 'MAE': xgb_mae, 'R2': xgb_r2},
    'LightGBM': {'MSE': lgbm_mse, 'MAE': lgbm_mae, 'R2': lgbm_r2},
    'CatBoost': {'MSE': catb_mse, 'MAE': catb_mae, 'R2': catb_r2}
}

```

```

'LGBM Boost': {'MSE': lgbm_mse, 'MAE': lgbm_mae, 'R2': lgbm_r2},
'Cat Boost': {'MSE': catb_mse, 'MAE': catb_mae, 'R2': catb_r2},
}

print("\nOptimized Model Results:") # 打印提示信息
print(opti_model_results) # 打印包含优化后模型评估结果的字典

```

应用到其他数据集：

- **保持一致性：**

评估优化后模型的流程与评估初始模型基本相同，关键是使用优化后的模型 (`*_model`) 和它们在测试集上的新预测 (`*_predictions`)。

- **对比性能提升：**

将 `opti_model_results` 与之前的 `model_results` 进行比较，是判断超参数优化是否有效的关键。看 `MSE/MAE` 是否降低，`R2` 是否升高。

- **仔细检查变量名：**

在复制代码或修改时，特别注意确保每个模型的评估都使用了正确的预测变量。上面注释中标注了原始代码中可能存在的复制粘贴错误，需要修正才能得到正确评估。

阶段三十二：可视化比较优化前后性能

目的： 使用条形图直观地对比每个模型在超参数优化前后的性能差异（以 `MSE` 为例）。这有助于清晰地展示优化带来的效果。

python

```

# --- 提取优化前后的 MSE 值 ---
model_names = list(model_results.keys()) # 获取模型名称列表 (与 opti_model_results 的键应一致)
mse_values = [model_results[model]['MSE'] for model in model_names] # 提取优化前的 MSE 值
opti_mse_values = [opti_model_results[model]['MSE'] for model in model_names] # 提取优化后的

# --- 创建并显示对比条形图 ---
plt.figure(figsize=(12, 7)) # 创建图形窗口，设置稍大尺寸以便容纳两组条形

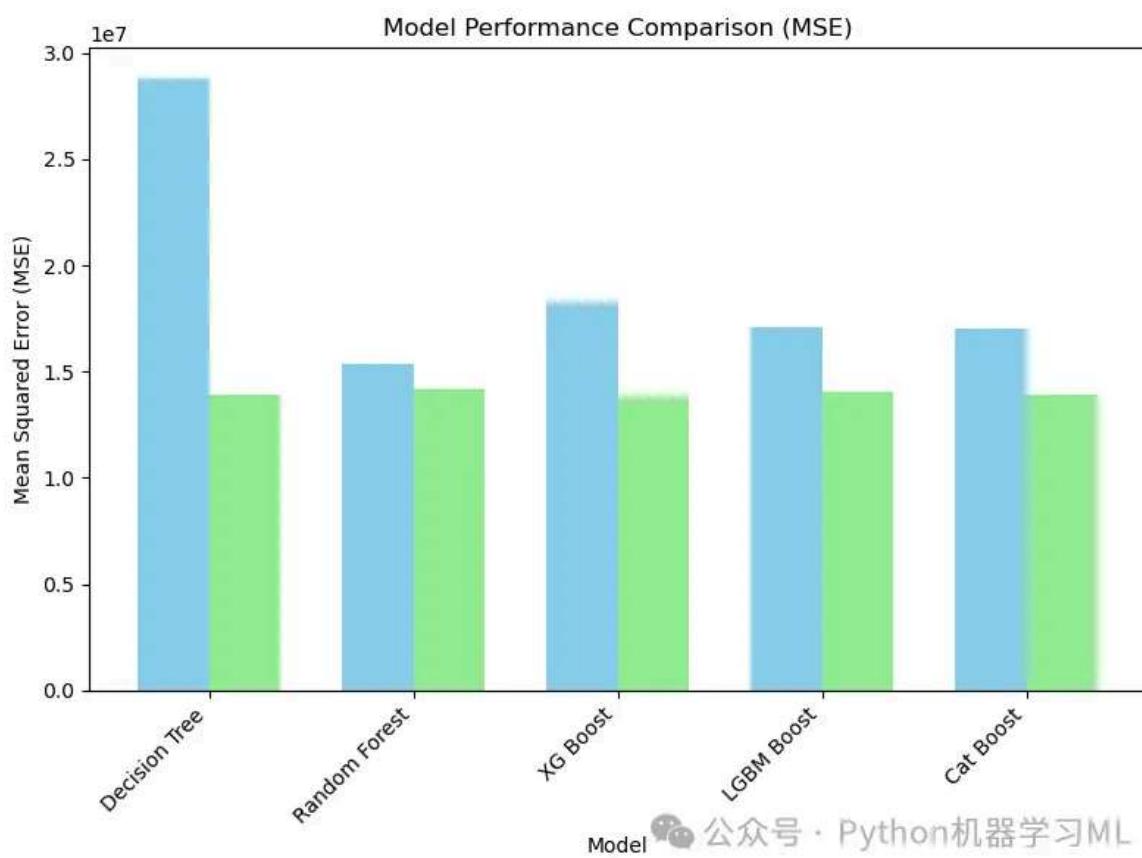
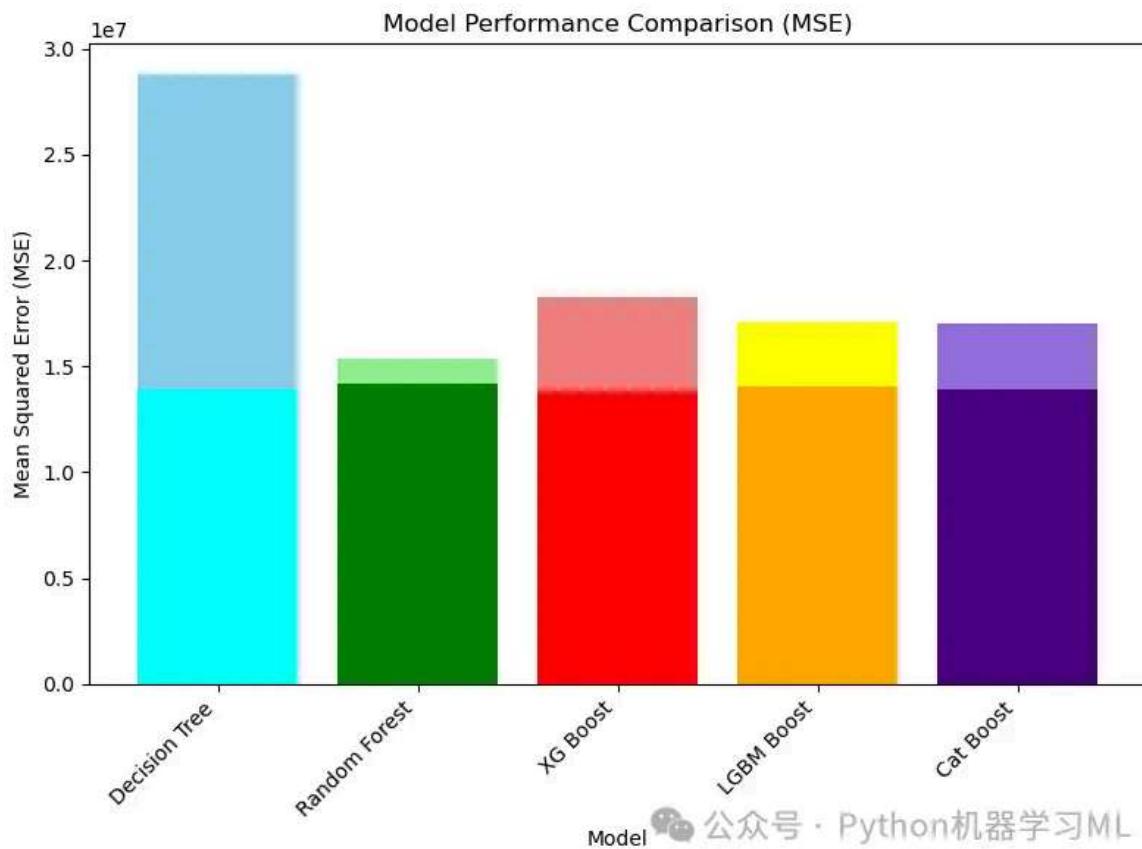
# --- 绘制条形图 (这里采用重叠绘制的方式) ---
# **注意：** 原代码直接连续调用 plt.bar，这会导致后绘制的条形覆盖在先绘制的条形之上。
# 这种方式可能不易比较。更好的方式是绘制分组条形图。
# 但我们按照原代码进行解释：
# 先绘制优化前的 MSE 条形 (较浅的颜色)
bar_width = 0.4# 定义条形宽度
index = np.arange(len(model_names)) # 创建模型名称对应的数字索引

# 绘制优化前的条形 (蓝色系)
plt.bar(index - bar_width/2, mse_values, bar_width, label='Initial MSE', color='skyblue')
# 绘制优化后的条形 (绿色系)，稍微偏移位置以便并排显示
plt.bar(index + bar_width/2, opti_mse_values, bar_width, label='Optimized MSE', color='lightgreen')

# --- 设置图表属性 ---
plt.xlabel("Model") # x 轴标签
plt.ylabel("Mean Squared Error (MSE) ↓") # y 轴标签，加箭头表示越低越好
plt.title("Model Performance Comparison: Initial vs. Optimized (MSE)") # 图表标题
# 设置 x 轴刻度位置和标签
plt.xticks(index, model_names, rotation=45, ha="right")
plt.legend() # 显示图例 (Initial MSE, Optimized MSE)
plt.grid(axis='y', linestyle='--', alpha=0.7) # 添加水平网格线
plt.tight_layout() # 自动调整布局
plt.show() # 显示图形

```

```
print("\nThe bar chart compares the Mean Squared Error (MSE) for each model before (Initial)  
print("A lower bar indicates better performance. Comparing the height difference for each mod
```



应用到其他数据集 / 改进可视化:

- **绘制分组条形图 (推荐):** 为了更清晰地比较, 可以使用分组条形图。这通常需要计算每个模型两组条形的位置, 示例如下 (替换上面 `plt.bar` 的部分) :

```
python

bar_width = 0.35# 定义条形宽度
index = np.arange(len(model_names)) # 创建模型名称对应的数字索引

plt.bar(index - bar_width/2, mse_values, bar_width, label='Initial MSE', color='skyblue')
plt.bar(index + bar_width/2, opti_mse_values, bar_width, label='Optimized MSE', color='lightgreen')

plt.xticks(index, model_names, rotation=45, ha="right") # 设置刻度在组的中间
```

- **可视化其他指标:** 同样可以提取并比较优化前后的 MAE 或 R² 值。
- **计算提升百分比:** 除了绝对值比较, 还可以计算每个模型性能提升的百分比 $\frac{\text{initial_mse} - \text{optimized_mse}}{\text{initial_mse}} * 100\%$, 并可能用文本或附加图表展示。
- **最终模型选择:** 结合优化后的性能指标 (MSE, MAE, R²)、训练/预测时间、模型的可解释性、部署复杂度等因素, 最终选择一个或几个最优模型用于实际应用。

第三部分 (阶段 33-39):

这部分代码利用 SHAP 库对训练好的 XGBoost 模型进行了深入的解释。它涵盖了 SHAP 的多种核心可视化方法:

- **全局解释:**
通过摘要图 (条形图和点图) 了解整体特征的重要性和影响方向。通过热图探索特征交互和样本聚类。
- **局部解释:**
通过决策图、瀑布图和力图深入理解模型对单个或一小组样本做出特定预测的原因。

这些解释对于理解模型行为、验证模型是否符合领域知识、识别潜在偏见、向利益相关者解释预测结果以及指导特征工程或模型改进都非常有价值。将这些 SHAP 分析方法应用于你自己的数据集和模型, 只需替换相应的模型对象和数据集即可。记住要根据你的具体问题 (回归/分类) 和模型类型来正确解读 SHAP 值和相关图表。

阶段三十三: 导入 SHAP 库并准备解释器

目的: 导入 SHAP 库以及其他绘图和数值计算所需的库。然后, 为我们选择要解释的模型 (这里是优化后的 XGBoost 模型 `xgb_model`) 创建一个 SHAP 解释器对象 (`Explainer`)。最后, 使用解释器计算测试集 (`X_test`) 中每个样本、每个特征的 SHAP 值。

```
python

# --- 导入 SHAP 和其他必要的库 ---
import shap # 导入 SHAP 库, 用于模型解释
import matplotlib.pyplot as plt # 导入 Matplotlib 的 pyplot 模块, SHAP 绘图底层依赖它
import numpy as np # 导入 NumPy 库, 用于数值操作, 例如处理 SHAP 值数组

# --- 为 XGBoost 模型创建 SHAP 解释器 ---
# shap.Explainer() 是创建解释器对象的核心函数
# 第一个参数是我们要解释的模型 (xgb_model)。SHAP 对多种模型类型有优化支持, 特别是树模型。
# 对于树模型 (如 XGBoost, LightGBM, CatBoost, RandomForest, DecisionTree), SHAP 会自动选择最
# 对于其他类型的模型, 可能需要提供训练数据作为背景数据 (e.g., shap.KernelExplainer(model.predict))
explainer = shap.Explainer(xgb_model)

# --- 计算 SHAP 值 ---
# 将 X_test 传递给 explainer, 以生成 SHAP 值
```

```

# 将解释器对象 explainer 应用于我们要解释的数据集 X_test
# explainer(X_test) 会计算 X_test 中每个样本、每个特征对应的 SHAP 值。
# 对于树模型，这个计算通常很快。
# 返回的 shap_values 是一个 SHAP Explanation 对象（或类似结构，取决于 SHAP 版本），它包含了：
#   - .values: 一个 NumPy 数组，形状通常是 (n_samples, n_features)，存储每个样本每个特征的 SHAP 值。
#   - .base_values: 一个标量或数组，表示模型的基准值（通常是模型在训练数据上的平均预测值  $E[f(X)]$ ）。
#   - .data: 输入的数据 (X_test)。
#   - .feature_names: 特征名称（如果可用）。
shap_values = explainer(X_test)

```

应用到其他数据集：

- **选择要解释的模型：**

将 `xgb_model` 替换为你想要解释的其他训练好的模型。确保 SHAP 支持该模型类型，或者选择合适的 Explainer 类型（如 `KernelExplainer`）。

- **选择要解释的数据：**

将 `X_test` 替换为你想要计算 SHAP 值的数据集。通常使用测试集来评估模型在未见过数据上的解释。也可以使用训练集（`X_train`）或特定样本子集。

- **理解 SHAP 值：**

一个特征的 SHAP 值 `shap_values.values[i, j]` 表示对于第 `i` 个样本，第 `j` 个特征的存在将预测结果从基准值 `explainer.expected_value`（或 `shap_values.base_values`）推高（正 SHAP 值）或拉低（负 SHAP 值）了多少。所有特征的 SHAP 值之和加上基准值应该等于该样本的模型预测值：

```
sum(shap_values.values[i, :]) + explainer.expected_value ≈ model.predict(X_test.iloc[[i]])[0]。
```

阶段三十四：SHAP 全局解释 - 特征重要性 (Summary Plot - Bar)

目的： 使用 SHAP 的摘要图（条形图模式）来展示全局特征重要性。这种图计算每个特征 SHAP 值的平均绝对值 (`mean(|SHAP value|)` across all samples)，并按重要性排序绘制条形图。这提供了一种比传统特征重要性（如基于分裂增益或排列重要性）更一致、更可靠的全局重要性度量。

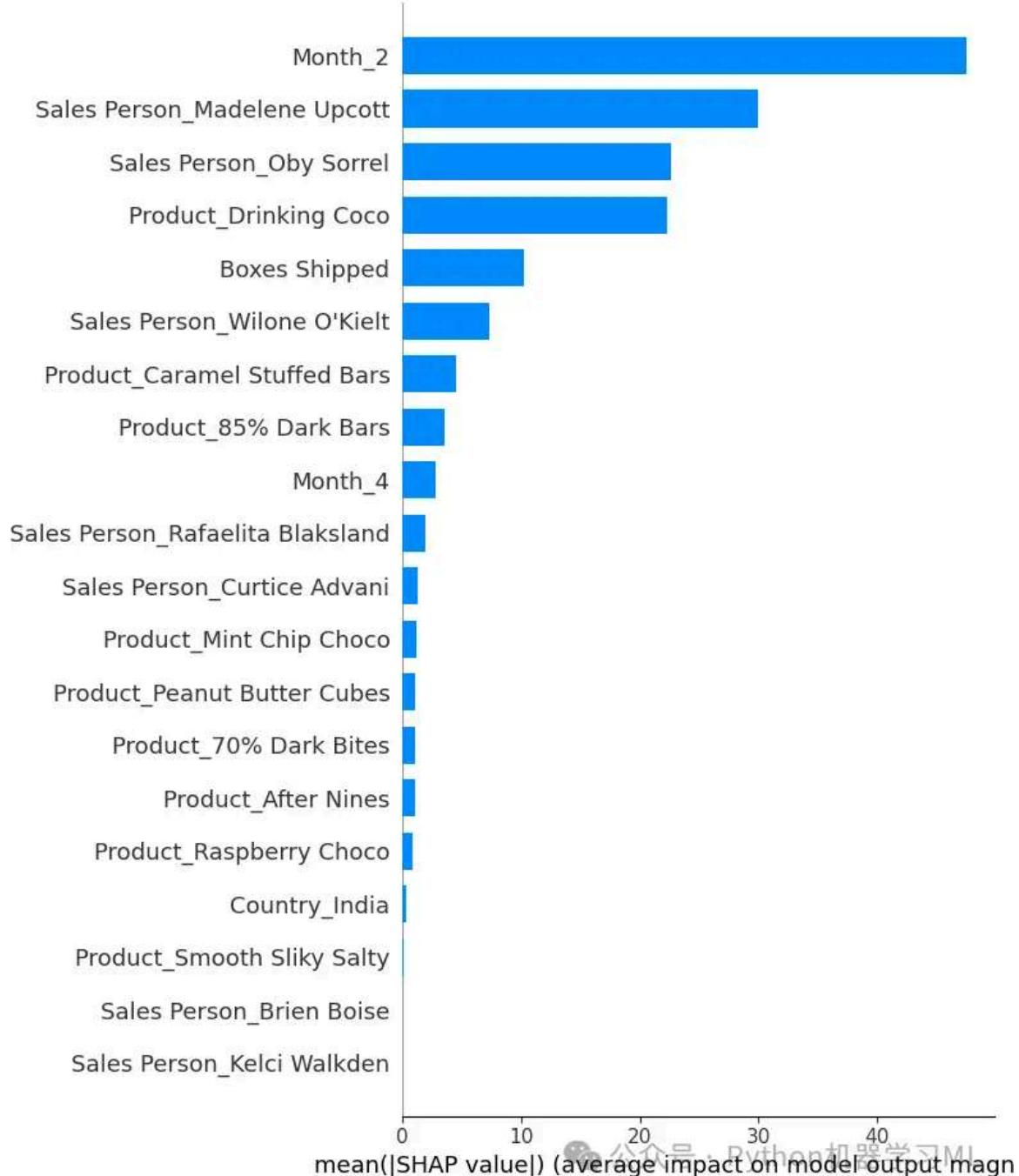
python

```

# --- 摘要图 - 显示特征重要性和影响方向（条形图模式）---
# shap.summary_plot() 是绘制多种 SHAP 摘要图的函数
# shap_values: 包含 SHAP 值的 Explanation 对象（或数组）
# X_test: 对应的特征数据，用于获取特征名称和可能的特征值信息
# plot_type="bar": 指定绘制条形图模式，显示全局特征重要性（基于平均绝对 SHAP 值）
shap.summary_plot(shap_values, X_test, plot_type="bar")

# --- 图形调整与保存 ---
plt.tight_layout() # 自动调整布局，防止标签重叠
# plt.savefig() 保存图形到文件
# "shap_feature_importance_bar.png": 文件名（修改了后缀以区分）
# dpi=300: 设置图像分辨率为 300 点每英寸，提高图像质量
# bbox_inches='tight': 尝试裁剪掉图像周围多余的空白区域
plt.savefig("shap_feature_importance_bar.png", dpi=300, bbox_inches='tight') # 修改了文件名
plt.show() # 在屏幕上显示图形

```



应用到其他数据集：

- **直接复用：**

这段代码通常可以直接复用，只要 `shap_values` 和 `x_test` 已经准备好。

- **解读图表：**

条形越长，表示该特征对模型预测的平均影响越大（无论方向），即该特征越重要。特征按重要性从高到低排序。

- **对比其他重要性：**

可以将 SHAP 的全局重要性与模型自带的重要性（如 `xgb_model.feature_importances_`）进行比较，SHAP 通常被认为更可靠。

阶段三十五：SHAP 全局解释 - 特征影响分布 (Summary Plot - Dot/Violin)

目的： 使用 SHAP 的摘要图（点图或小提琴图模式，这里代码调用了 `plot_type="dot"`）来更详细地展示每个特征的 SHAP 值分布情况。图中每一行代表一个特征，每个点代表一个样本在该特征上的 SHAP 值。点的颜色通常表示该样本在该特征上的原始值（高值/低值），点的水平位置表示 SHAP 值的大小和方向。这不仅显示了特征的重要性，还揭示了特征值如何影响预测方向（是推高还是拉低预测）。

```
# --- 摘要点图 - 显示每个特征的SHAP值分布 ---
# 再次调用 shap.summary_plot()
# plot_type="dot" (默认): 绘制点图模式。每个点是一个样本的 SHAP 值。
#   - 水平位置: SHAP 值 (正值推高预测, 负值拉低)
#   - 垂直位置: 按特征重要性排序
#   - 颜色: 通常表示特征值的大小 (高值/低值, 颜色条会显示映射关系)。
#   - 点的堆叠: 显示了 SHAP 值在该位置的密度。
# 也可以用 plot_type="violin", 它会用小提琴图展示分布形状, 有时更清晰。
shap.summary_plot(shap_values, X_test, plot_type="dot")

# --- 图形调整与保存 ---
plt.tight_layout() # 调整布局
# 注意: 原代码再次使用了相同的文件名 "shap_feature_importance.png", 这会覆盖之前的条形图。
# 建议使用不同的文件名。
plt.savefig("shap_feature_importance_dot.png", dpi=300, bbox_inches='tight') # 修改了文件名
plt.show() # 显示图形

# --- 打印特征列名 (辅助理解后续图表) ---
# 打印 X_test 的列名, 方便后续图表 (如决策图、瀑布图) 中核对特征名称
print("Features used in SHAP analysis:", X_test.columns)
```



应用到其他数据集：

- **直接复用：**
代码可直接复用。
- **解读图表：**
 - 观察特征的排序，与条形图一致。
 - 观察每个特征的点分布范围，范围越宽表示该特征对不同样本的影响差异越大。
 - 观察颜色与水平位置的关系：例如，如果某特征高值（红色）的点主要分布在 SHAP 值正半轴，低值（蓝色）的点主要分布在负半轴，则说明该特征值越高，越倾向于推高模型的预测值。反之亦然。如果颜色混合分布，说明关系可能更复杂或非单调。
- **选择 `plot_type` :**
"dot" 和 "violin" 各有优劣，可以都尝试一下看哪个效果更好。

阶段三十六：SHAP 局部解释 - 决策图 (Decision Plot)

目的： 使用 SHAP 决策图来可视化模型对一组样本（这里是前 50 个测试样本）的预测过程。图中每条线代表一个样本，从底部的基准值开始，随着特征的依次加入，线条根据该特征的 SHAP 值向上或向下移动，最终到达顶部的模型预测值。这有助于理解哪些特征对特定样本的预测起到了关键的推动作用，以及它们的累积效应。

```
python
```

```
# --- 决策图 - 显示样本的预测路径 ---
# shap.decision_plot() 用于绘制决策图
# explainer.expected_value: 模型的基准预测值  $E[f(X)]$ , 是决策路径的起点。
# shap_values.values[:50]: 选择前 50 个样本的 SHAP 值数组进行绘制。
# feature_names=np.array(X_test.columns): 提供特征名称列表, 用于在图中标注。确保与 SHAP 值列的
# show=False: 阻止 SHAP 库自动调用 plt.show(), 允许我们后续进行自定义调整和保存。
shap.decision_plot(explainer.expected_value, shap_values.values[:50],
                    feature_names=list(X_test.columns), # 使用 list() 转换确保兼容性
                    show=False) # 修改这里确保 feature_names 是列表

# --- 图形调整与保存 ---
plt.tight_layout() # 调整布局
plt.savefig("shap_decision_plot.png", dpi=300, bbox_inches='tight') # 保存图形
plt.show() # 显示图形
```



应用到其他数据集：

- **选择样本：**

修改 `shap_values.values[:50]` 中的切片范围来选择你感兴趣的样本进行可视化，例如 `shap_values.values[[10, 20, 30]]` 只绘制第 10, 20, 30 个样本。绘制过多样本会使图形混乱。

- **提供特征名：**

确保 `feature_names` 参数正确传递了与 `shap_values.values` 列顺序一致的特征名称列表。

- **解读图表：**

- 每条彩色线是一个样本。
- 线条的起点是基准值 (Expected Value)。
- 线条的终点是该样本的模型预测值。
- 线条在每个特征处的弯折表示该特征对预测的贡献 (SHAP 值)。向上弯折表示正贡献，向下弯折表示负贡献。
- X 轴通常按特征重要性排序 (可通过 `feature_order` 参数修改)。
- 可以比较不同样本的决策路径，看它们为何得到相似或不同的预测结果。

阶段三十七：SHAP 局部解释 - 瀑布图 (Waterfall Plot)

目的： 使用 SHAP 瀑布图来详细展示单个样本的预测是如何从基准值 $E[f(X)]$ 开始，通过累加每个特征的 SHAP 值，最终达到模型预测值 $f(x)$ 的过程。这对于深入理解某个特定预测结果的成因非常有用。代码中循环绘制了多个样本 (索引 0, 5, 10) 的瀑布图进行对比。

python

```
# --- 瀑布图 - 详细分析多个预测样本 ---
# 定义要分析的样本的索引列表
sample_indices = [0, 5, 10] # 选择第 0, 5, 10 个测试样本

# 循环遍历每个选定的样本索引
for i, idx in enumerate(sample_indices):
    # 注意：SHAP 版本更新可能导致 API 变化。以下代码尝试兼容较新版本。
    # 原始代码中 waterfall 的调用方式可能在更新版本中不推荐。
    # 我们需要创建一个 shap.Explanation 对象来传递给新的 waterfall API。

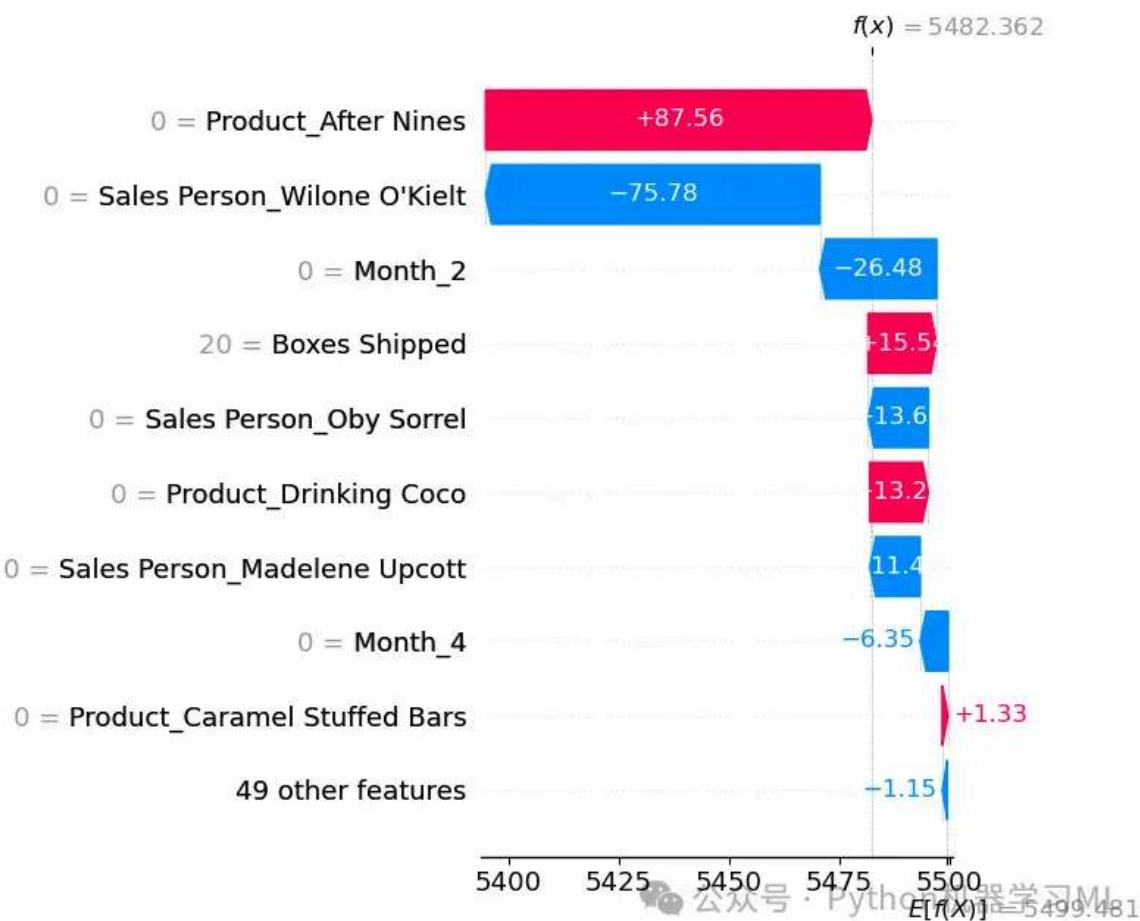
    # 创建 shap.Explanation 对象
    # values: 当前样本的 SHAP 值 (shap_values.values[idx])，注意用 idx 而不是 i)
    # base_values: 模型的基准值
    # data: 当前样本的原始特征值 (X_test.iloc[idx])
    # feature_names: 特征名称列表
    exp = shap.Explanation(values=shap_values.values[idx], # 使用 idx 获取对应样本的 SHAP 值
                            base_values=explainer.expected_value,
                            data=X_test.iloc[idx].values, # 使用 idx 获取对应样本数据，并转为
                            feature_names=list(X_test.columns)) # 确保是列表

    # plt.figure() 为每个样本的瀑布图创建新的图形窗口
    plt.figure(figsize=(10, 6)) # 调整了尺寸

    # --- 绘制瀑布图 ---
    # 使用 shap.plots.waterfall() 函数（注意路径可能因版本而异，较新版本推荐此方式）
    # 传入创建的 Explanation 对象 exp
    # max_display=15: 限制最多显示的特征数量（按绝对 SHAP 值大小排序），避免图形过于拥挤
    # show=False: 阻止自动显示，允许后续保存
    shap.plots.waterfall(exp, max_display=15, show=False) # 限制显示特征数量

    # --- 图形调整与保存 ---
    plt.title(f"Waterfall Plot for Sample Index {idx}") # 添加标题
    plt.tight_layout() # 调整布局
```

```
# 使用 f-string 为每个样本的瀑布图生成不同的文件名
plt.savefig(f"shap_waterfall_sample_{idx}.png", dpi=300, bbox_inches='tight')
plt.show() # 显示当前样本的瀑布图
```



应用到其他数据集：

- 选择样本：**
修改 `sample_indices` 列表，放入你想要单独分析的样本的索引。
- 检查 SHAP API：**
SHAP 库的 API 时有更新。绘制瀑布图（以及力图）的方式可能需要根据你安装的 SHAP 版本进行调整。
推荐查阅你所用 SHAP 版本的官方文档。上面提供的代码使用了较新的 `shap.Explanation` 和 `shap.plots.waterfall` 方式。
- 调整 `max_display`：**
根据特征数量和图表清晰度需求，调整 `max_display` 参数。
- 解读图表：**
 - 图的底部是基准值 $E[f(X)]$ 。
 - 每个条形代表一个特征对其 SHAP 值的贡献。红色条表示正贡献（推高预测），蓝色条表示负贡献（拉低预测）。条形的长度表示贡献的大小。
 - 特征按其绝对贡献大小排序（通常最重要的在顶部或底部）。
 - 所有条形的累加效应将基准值“推动”到最终的预测值 $f(x)$ （图的顶部）。

阶段三十八：SHAP 局部解释 - 力图 (Force Plot)

目的： 使用 SHAP 力图（单个样本版本）来可视化特定样本预测的“力量平衡”。它展示了哪些特征将预测值推高（红色部分，正 SHAP 值），哪些特征将预测值拉低（蓝色部分，负 SHAP 值），以及这些力量如何平衡，最终得到模型的预测输出值。代码中循环绘制了多个样本（索引 7, 15, 25）的力图。

```

# --- 力图 - 单个样本预测的特征贡献 ---
# 为多个不同样本创建力图以比较分析
sample_indices = [7, 15, 25] # 选择要分析的样本索引
print("\n--- Force Plot Explanation ---") # 打印提示信息
print(" - Force plot shows how features push the prediction away from the base value.") # 解释
print(" - Red features increase the prediction, Blue features decrease it.") # 解释颜色含义
print("-----")

# 循环遍历每个选定的样本索引
for sample_idx in sample_indices:
    # plt.figure() 为每个力图创建新图形窗口
    # 注意: 对于 matplotlib=True 的力图, 图形大小可能不由 plt.figure 控制, 而是在 force_plot 内部决定
    # plt.figure(figsize=(14, 3)) # 这行可能对 matplotlib 力图无效

    # --- 准备样本数据 (可选, 为了显示更清晰的特征值) ---
    # 创建当前样本特征数据的副本, 以防修改原始数据
    # sample_data = X_test.iloc[sample_idx].copy() # 使用 .iloc 获取对应行
    # 遍历样本数据中的每个特征值
    # for i in range(len(sample_data)): # 原代码的循环方式对 Pandas Series 可能不健壮
    # 将特征值四舍五入到两位小数, 以便在力图中显示更简洁的数值
    # sample_data.iloc[i] = round(sample_data.iloc[i], 2) # 使用 .iloc 修改值
    # 更简洁的四舍五入方式:
    sample_data_display = X_test.iloc[sample_idx].round(2) # 直接对整个 Series 调用 round

    # --- 绘制力图 (matplotlib 版本) ---
    print(f"\nGenerating Force Plot for Sample Index {sample_idx}...")
    # shap.force_plot() 用于绘制力图
    # explainer.expected_value: 基准值 E[f(X)]
    # shap_values.values[sample_idx]: 当前样本的 SHAP 值数组
    # sample_data_display: 当前样本的特征值 (用于在图中显示)。使用上面处理过的版本。
    # matplotlib=True: 指定生成 Matplotlib 版本的力图 (静态图)。如果为 False (默认), 会生成交互式力图
    # feature_names=list(X_test.columns): 提供特征名称列表。
    # show=False: 阻止自动显示。
    shap.force_plot(explainer.expected_value,
                    shap_values.values[sample_idx],
                    sample_data_display, # 使用处理后的数据
                    matplotlib=True,
                    feature_names=list(X_test.columns), # 确保是列表
                    show=False)

    # --- 图形调整与保存 ---
    # 获取当前的 Matplotlib 图形和坐标轴, 以便添加标题等
    fig = plt.gcf() # Get Current Figure
    # plt.title(f"Force Plot for Sample Index #{sample_idx}", fontsize=15) # 为图形添加标题
    # plt.suptitle(f"Force Plot for Sample Index #{sample_idx}", fontsize=15, y=1.02) # 使用 suptitle
    # plt.tight_layout() # 对于 force_plot(matplotlib=True) 可能效果有限或不需要
    plt.savefig(f"shap_force_plot_sample_{sample_idx}.png", dpi=300, bbox_inches='tight') # 保存为 PNG
    plt.show() # 显示图形

    # --- 打印额外分析信息 (回归问题可能不需要概率) ---
    # 原代码计算了一个 "预测概率", 这通常用于二分类问题 (通过 sigmoid 转换)。
    # 对于回归问题, 模型直接输出预测值, 不需要转换成概率。
    # pred_prob = 1 / (1 + np.exp(-explainer.expected_value - np.sum(shap_values.values[sample_idx])))
    predicted_value = explainer.expected_value + np.sum(shap_values.values[sample_idx]) # 回归问题直接使用预测值

    print(f"--- Sample #{sample_idx} Analysis ---")
    print(f"- Model Predicted Value: {predicted_value:.2f}") # 打印回归预测值

```

```

print(f"- Base Value (Average Prediction): {explainer.expected_value:.2f}") # 打印基准值
feature_names=list(X_test.columns) # 获取特征名列表
# np.argsort() 返回排序后的索引。负号表示降序（找到 SHAP 值最大的特征）。[:3] 取前三个。
print(f"- Top 3 features increasing prediction: {[feature_names[i] for i in np.argsort(-shap_
# np.argsort() 默认升序（找到 SHAP 值最小，即负得最多的特征）。[:3] 取前三个。
print(f"- Top 3 features decreasing prediction: {[feature_names[i] for i in np.argsort(shap_"
print("-----")

```



应用到其他数据集：

- 选择样本：**
修改 `sample_indices` 列表。
- 检查 SHAP API：**
同样，`force_plot` 的用法可能随版本变化。
- 选择 `matplotlib=True` 或 `False`：**
`True` 生成静态图，适合嵌入报告或论文。`False` 生成交互式 HTML 图，可以在 Jupyter Notebook 或浏览器中动态探索，鼠标悬停可看详细信息，通常更推荐用于探索阶段。如果用 `False`，保存方式会不同（通常需要 `shap.save_html(...)`）。
- 回归 vs 分类：**
注意力图和后续分析的解释。对于回归，SHAP 值解释的是对最终预测值（如销售额）的贡献。对于二分类，如果模型输出是对数几率 (logits)，SHAP 值解释的是对 logit 的贡献，需要转换才能得到对概率的解释。对于多分类，情况更复杂。
- 解读图表：**
 - 中间的粗竖线是模型的最终预测值 $f(x)$ 。
 - 图的“力量”从左到右累积。红色块代表正 SHAP 值（推高预测的特征），蓝色块代表负 SHAP 值（拉低预测的特征）。块的宽度表示该特征 SHAP 值的绝对大小。
 - 鼠标悬停（交互式版本）或查看标签（静态版本）可以看到每个块对应的特征名称和原始值。
 - 所有力量最终平衡在预测值 $f(x)$ 处。

阶段三十九：SHAP 全局解释 - 热图 (Heatmap Plot)

目的： 使用 SHAP 热图来可视化特征之间的交互效应以及特征值对预测的影响。热图的行通常代表样本（可以按某种方式排序），列代表特征（通常按重要性或聚类排序）。单元格的颜色表示该样本在该特征上的 SHAP 值。这种图有助于发现样本聚类模式、特征间的相互作用以及特征值如何共同影响大量样本的预测。

python

```

# --- SHAP值的热图 ---
# 重新计算 SHAP 值（如果之前的 shap_values 对象可能被修改或不适用）
# 这步在前面已经计算过一次，如果 shap_values 仍然有效，可以省略这步。
# explainer = shap.Explainer(xgb_model) # 如果 explainer 已存在，无需重复创建
# shap_values = explainer(X_test) # 如果 shap_values 已存在且包含所需数据，无需重复计算

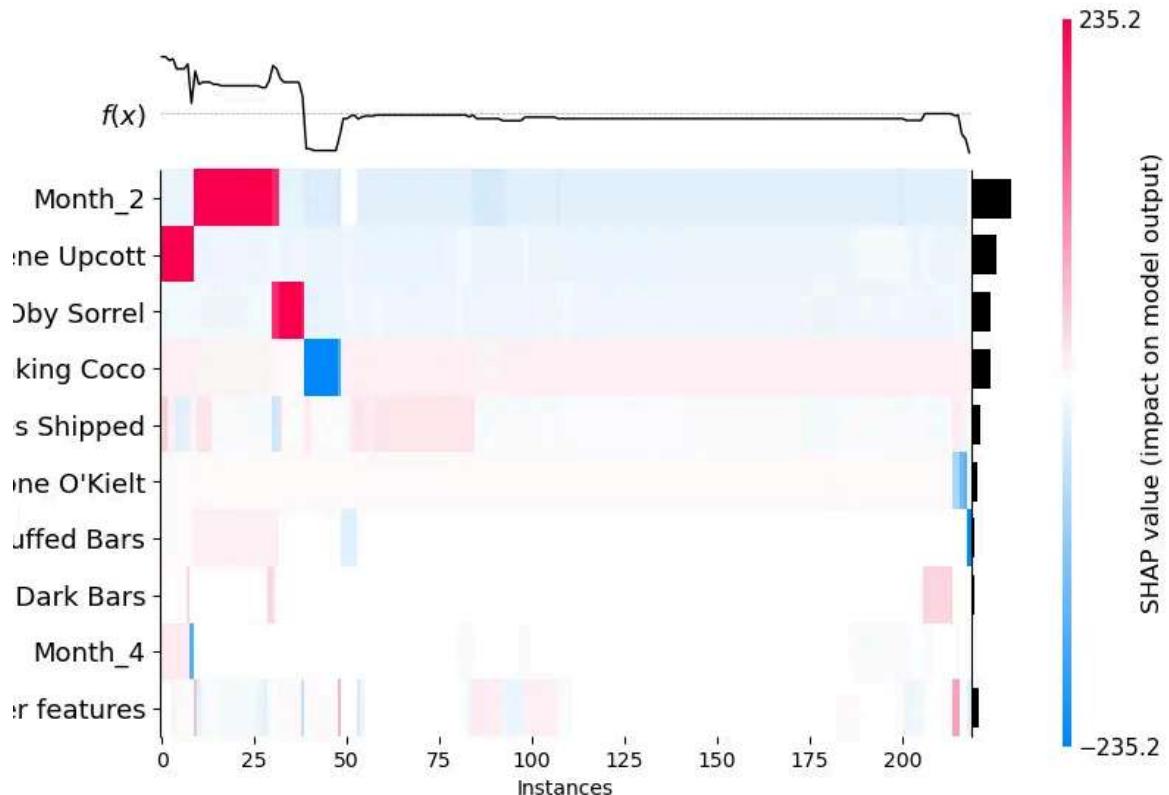
# --- 绘制 SHAP 热图 ---
# 使用 shap.plots.heatmap() 函数绘制热图
# shap_values: 包含 SHAP 值的 Explanation 对象（或数组）
# max_display=15: 限制显示的主要特征数量（按全局重要性排序），避免图形过于拥挤
# show=False: 允许后续保存（如果需要）
plt.figure(figsize=(10, 8)) # 创建图形窗口，热图通常需要较大空间
shap.plots.heatmap(shap_values, max_display=15, show=False) # 限制特征数量并阻止自动显示

```

```

# --- 保存或显示图形 ---
plt.title("SHAP Heatmap Plot") # 添加标题
plt.tight_layout() # 尝试调整布局（对 heatmap 可能效果有限）
plt.savefig("shap_heatmap.png", dpi=300, bbox_inches='tight') # 保存图形
plt.show() # 显示图形

```



公众号 · Python机器学习ML

应用到其他数据集：

- **直接复用：**

代码可直接复用，前提是 `shap_values` 已计算好。

- **调整 `max_display` :**

根据特征数量和希望看到的细节程度调整此参数。

- **解读图表：**

- **行排序：**

行（样本）通常是根据某种聚类算法（基于它们的 SHAP 值向量）进行排序的，相似预测模式的样本会聚集在一起。

- **列排序：**

列（特征）通常是根据全局 SHAP 重要性或特征间的交互效应聚类来排序的。

- **颜色：**

单元格颜色代表 SHAP 值。通常红色表示正 SHAP 值（推高预测），蓝色表示负 SHAP 值（拉低预测）。颜色的深浅表示 SHAP 值的绝对大小。

- **顶部图（特征值）：**

热图上方通常会有一个对应的条形图或点图，显示每个特征在对应样本上的原始值，颜色与热图单元格的 SHAP 值相关联，帮助理解特征值如何驱动 SHAP 值。

- 右侧图（预测值）：
热图右侧通常有一条线图，显示每个样本（按行的顺序）的模型预测输出值 $f(x)$ 。
 - 通过观察热图中的颜色模式、行聚类和列聚类，可以发现复杂的相互作用和模式，例如哪些特征组合倾向于导致高/低预测值，是否存在具有相似解释模式的样本群体等。
-



公众号

【数据，请加微信获取】

往期回顾

机器学习——因果推断方法的DeepIV和因果森林双重机器学习（CausalForestDML）示例

论文复现——肺癌数据高级模型比较与shap可视化分析代码解析

论文复现——肺癌预测数据分析与逻辑回归、朴素贝叶斯、支持向量机、随机森林、K近邻、XGBoost、深度神经网络模型评估代码解析

机器学习——材料力学XGBoost 分类，SHAP可视化分析、SMOTE解决不平衡、PCA降维、统计分析完整代码解析

机器学习——SHAP可解释分析、EDA、逻辑回归、支持向量机（SVM）电动车数据集Python完整代码

机器学习——集成学习、线性模型、支持向量机、K近邻、决策树、朴素贝叶斯、虚拟分类器分析电动车数据集Python完整代码

深度学习与图像分类：基于鸟类图片分类项目的实践python完整代码

机器学习——二元Logistic回归算法实战：从数据预处理到模型评估python完整代码

机器学习——使用Lazypredict选择最优模型、Optuna调优、PCA降维进行客户信息数据分类分析python完整代码

机器学习——处理多元分类数据的 7 种可视化方法Python 完整代码

机器学习——基于颜色直方图和 SVM 分类水果图像代码解析python



公众号

【数据，请加微信获取】



如果你对类似于这样的文章感兴趣。

欢迎关注、点赞、转发~

机器学习 66 shap 6 随机森林 1 optuna 2 CatBoost 1

机器学习 · 目录

[上一篇 · 论文复现——基于CT图像中肺癌分类的 EfficientNet 方法、类不平衡处理、交叉验证...](#)

个人观点，仅供参考