一、环境安装 (Windows 系统)

本节将指导您在Windows操作系统上配置进行Python数据分析所需的环境。

1. **安装Python：**
    - 访问Python官方网站：https://www.python.org/
    - 点击“Downloads”菜单，然后选择“Windows”。
    - 下载最新稳定版本的Python安装程序（例如，“Windows installer (64-bit)”）。
    - 运行下载的`.exe`安装程序。
    - 在安装向导的第一个界面，**请务必勾选“Add Python X.Y to PATH”选项**（其中X.Y是Python版本号），这样可以在命令行中直接使用`python`命令。
    - 选择“Install Now”进行默认安装，或选择“Customize installation”以选择安装路径和组件（通常默认安装即可）。
    - 等待安装完成，然后点击“Close”。
2. **验证Python和pip安装：**
    - 按下 `Win + R` 键，输入 `cmd` 并按回车键，打开命令提示符。
    - 在命令提示符窗口中，输入 `python --version` 并按回车键。如果安装成功，会显示Python的版本号（如 `Python 3.10.x`）。
    - 接着，输入 `pip --version` 并按回车键。同样，会显示pip的版本号。`pip`是Python的包管理器，用于安装其他库。
3. **创建并激活Python虚拟环境（推荐）：**
    - **目的：** 虚拟环境可以隔离项目所需的库，避免不同项目间的库版本冲突。
    - **创建项目文件夹：** 在您方便的位置创建一个新的文件夹来存放项目文件，例如 `D:\PythonProjects\MovieAnalysis`。
    - **打开命令提示符并进入项目文件夹：**
        - 按下 `Win + R` 键，输入 `cmd` 并按回车键。
        - 使用 `cd` 命令进入您的项目文件夹。例如：`cd D:\PythonProjects\MovieAnalysis`
    - **创建虚拟环境：** 在项目文件夹中运行以下命令来创建虚拟环境（命名为 `venv`）：
        - `python -m venv venv`
        - （`venv`是虚拟环境的名称，你可以自定义，但`venv`是最常见的命名。）
    - **激活虚拟环境：**
        - 在项目文件夹的命令提示符中输入以下命令并按回车键：
            - `.\venv\Scripts\activate`
        - 成功激活后，命令提示符的前面会显示虚拟环境的名称，例如 `(venv) D:\PythonProjects\MovieAnalysis>`。
4. **安装项目所需库：**
    - **确保虚拟环境已激活**（命令提示符前有 `(venv)`）。
    - 使用pip安装Pandas、NumPy、Matplotlib和Jupyter Notebook：
        - `pip install pandas numpy matplotlib jupyter`
    - 等待所有库安装完成。

5. **启动Jupyter Notebook：**
    - 在已激活虚拟环境的命令提示符窗口中，输入以下命令并按回车键：
        - `jupyter notebook`
    - 您的默认Web浏览器会自动打开，显示Jupyter Notebook的界面。

二、数据获取

本节介绍如何获取用于项目分析的IMDb电影数据集。

1. **查找数据集来源：**
    - 最常见的IMDb数据集来源是 **Kaggle**。在Kaggle网站搜索“IMDb movies dataset”或类似关键词。
    - 你也可以在其他数据科学社区或资源网站上寻找。
2. **下载数据集文件：**
    - 找到一个适合的数据集后，根据网站的指示下载数据集文件。
    - **文件格式：** 数据集通常以CSV（逗号分隔值）文件的形式提供。确保你下载的是`.csv`文件。
    - **保存位置：** 将下载的`.csv`数据集文件**保存到你之前创建的Python项目文件夹中**（例如，你的项目文件夹是`D:\PythonProjects\MovieAnalysis`，就把`movies.csv`（或你下载的文件名）放在这个文件夹里）。
    - **了解数据内容：** 在下载前或下载后，浏览数据集的描述信息，了解其中包含的列（如电影标题、评分、票房、类型、时长、预算、上映年份等）。

三、数据清洗 (数据处理)

本职责的核心是使用Pandas库加载数据并进行预处理，确保数据质量，为后续分析做好准备。

1. **启动Jupyter Notebook并创建新Notebook：**
    - 确保你的项目文件夹（如`D:\PythonProjects\MovieAnalysis`）已在命令提示符中，并且虚拟环境已激活。
    - 运行`jupyter notebook`，在浏览器中打开Jupyter界面。
    - 点击“New” -> “Python 3 (ipykernel)” 创建一个新的Notebook文件。
2. **加载数据集：**
    - 在Notebook的第一个代码单元格中，导入Pandas库并加载你的CSV文件。
    - **示例代码：**

```
import pandas as pd

# 假设你的数据集文件名为 'movies.csv'，并且已保存在当前项目文件夹中
# 如果文件在子文件夹（如 'data'）中，路径应为 'data/movies.csv'
try:
    df = pd.read_csv('movies.csv')
```

```
    print("数据集加载成功！")
except FileNotFoundError:
    print("错误：未找到 'movies.csv' 文件。请确认文件路径和文件名是否正
确。")
```

- 运行此单元格（按下 `Shift + Enter`）。

3. **初步数据概览：**

- 在新的单元格中，查看数据的基本信息以了解其结构。
- **示例代码：**

```
if 'df' in locals(): # 确保数据集已成功加载
    print("--- 数据前5行预览 ---")
    print(df.head())  # 显示前5行记录

    print("\n--- 数据信息概览 ---")
    print(df.info())  # 显示每列的名称、非空值数量和数据类型

    print("\n--- 数值列描述性统计 ---")
    print(df.describe()) # 提供数值型列的统计摘要（计数、均值、标准差、最
小值、25%/50%/75%分位数、最大值）
```

- 运行此单元格。

4. **处理缺失值：**

- 检查每列的缺失值数量。
- **示例代码：**

```
if 'df' in locals():
    print("\n--- 每列的缺失值数量 ---")
    print(df.isnull().sum())
```

- 根据缺失值情况决定处理策略（如删除含有缺失值的行/列，或填充缺失值）。
- **示例：填充缺失值**（假设用平均值填充“评分”列，用0填充“预算”列）：

```
if 'df' in locals():
    # 填充“评分”列的缺失值（如果存在）
    if 'Rating' in df.columns:
        mean_rating = df['Rating'].mean()
        df['Rating'].fillna(mean_rating, inplace=True)
        print(f"已用平均评分 ({mean_rating:.2f}) 填充 'Rating' 列的缺
失值。")

    # 填充“预算”列的缺失值（如果存在）
    if 'Budget' in df.columns:
```

```
        df['Budget'].fillna(0, inplace=True)
        print("已用0填充 'Budget' 列的缺失值。")

    # 重新检查缺失值数量
    print("\n--- 填充缺失值后，剩余的缺失值数量 ---")
    print(df.isnull().sum())
```

- 运行此单元格。

5. **处理格式错误并提取关键字段：**
   - **提取“年份”字段：**
     - 如果你的数据集中有一个类似“Release_Date”的列，可以从中提取年份。
     - **示例代码：**

```
if 'Release_Date' in df.columns:
    # 尝试将日期列转换为日期时间格式，errors='coerce'会将无法解析的值设为NaT
    df['Release_Date'] = pd.to_datetime(df['Release_Date'], errors='coerce')
    # 提取年份
    df['Year'] = df['Release_Date'].dt.year
    print("已从 'Release_Date' 列提取 'Year' 列。")
else:
    print("警告：'Release_Date' 列不存在，无法提取年份。")
```

   - **处理“预算”和“票房”等数值型字段：**
     - 这些列可能包含货币符号（如'$'）、千位分隔符（如','）或空白字符串，需要转换为数值类型。
     - **示例代码（处理“Budget”列）：**

```
if 'Budget' in df.columns:
    # 确保是字符串类型以进行替换操作
    df['Budget'] = df['Budget'].astype(str)
    # 移除 '$', ',' 和可能的空格
    df['Budget'] = df['Budget'].str.replace('$', '', regex=False).str.replace(',', '', regex=False).str.strip()
    # 将空字符串转换为NaN，然后使用pd.to_numeric转换为数值类型
    df['Budget'] = pd.to_numeric(df['Budget'], errors='coerce')
    print("已处理 'Budget' 列，并转换为数值类型。")
```

     - 对“票房”（如 'Gross' 或 'Revenue' 列）进行类似处理。
   - **处理“电影类型”（Genre）：**
     - 如果类型信息是字符串列表形式（如“Action|Adventure|Sci-Fi”），可能需要将其拆分或转换为适合分析的格式。
     - **示例代码（假设类型为“Action|Adventure”）：**

```
if 'Genre' in df.columns:
    # 如果类型是字符串 'Action|Adventure|Sci-Fi'
    # 我们可以先将其转换为列表
    df['Genre_List'] = df['Genre'].str.split('|')
    print("已将 'Genre' 列拆分为列表格式。")
    # 可选：如果需要统计每个类型的电影数量，可以进一步处理
```

- **处理完以上步骤后，再次运行 `df.info()` 和 `df.describe()`，检查数据类型和值的范围是否已按预期修正。**

四、探索性数据分析 (EDA)

本职责的核心是使用Pandas和NumPy库来计算各种统计量，并深入分析电影的类型、评分、票房等关键指标之间的关系。

1. **统计不同类型电影数量：**
   - **操作：** 计算数据集中每种电影类型的出现次数。
   - **示例代码：**

```
if 'Genre' in df.columns:
    # 如果Genre是字符串，先处理成列表（如上一步骤）
    if 'Genre_List' not in df.columns:
        df['Genre_List'] = df['Genre'].str.split('|')

    # 展开列表，计算每个单独类型的数量
    all_genres = [genre for sublist in df['Genre_List'] for genre
in sublist if isinstance(sublist, list)]
    genre_counts = pd.Series(all_genres).value_counts()
    print("\n--- 不同电影类型的数量统计 ---")
    print(genre_counts)
else:
    print("警告：'Genre' 列不存在，无法统计类型数量。")
```

   - 运行此单元格。

2. **统计不同类型电影的平均评分：**
   - **操作：** 计算每种电影类型的平均评分。
   - **示例代码：**

```
if 'Genre_List' in df.columns and 'Rating' in df.columns:
    # 需要先将DataFrame展平，使得每行代表一个“电影-类型”的组合
    df_exploded = df.explode('Genre_List')
    # 按电影类型分组，计算平均评分
    avg_rating_by_genre = df_exploded.groupby('Genre_List')
```

```
['Rating'].mean().sort_values(ascending=False)
    print("\n--- 不同电影类型的平均评分 ---")
    print(avg_rating_by_genre)
else:
    print("警告：'Genre_List' 或 'Rating' 列不存在，无法计算平均评
分。")
```

- 运行此单元格。

3. **分析年份与评分/票房的关系：**

- **操作：** 观察电影的平均评分或平均票房随年份的变化趋势。
- **示例代码（分析平均评分随年份变化）：**

```
if 'Year' in df.columns and 'Rating' in df.columns:
    # 按年份分组，计算平均评分
    avg_rating_by_year = df.groupby('Year')['Rating'].mean()
    print("\n--- 各年份平均评分 ---")
    print(avg_rating_by_year.head()) # 显示前几年的数据

    # 可以考虑只分析近几十年的数据以获得更明显的趋势
    recent_years_df = df[(df['Year'] >= 1980) & (df['Year'] <=
2023)] # 示例范围
    avg_rating_recent = recent_years_df.groupby('Year')
['Rating'].mean()
    print("\n--- 近期各年份平均评分 ---")
    print(avg_rating_recent.head())
else:
    print("警告：'Year' 或 'Rating' 列不存在，无法分析年份与评分关系。")
```

- 运行此单元格。

4. **分析预算与票房的关系：**

- **操作：** 使用散点图或计算相关系数，来查看电影的预算与其最终票房收入之间的关系。
- **示例代码（计算相关系数）：**

```
if 'Budget' in df.columns and 'Gross' in df.columns: # 假设有Gross
列代表票房
    # 确保Budget和Gross都是数值类型且没有大量缺失值
    # 移除可能影响相关性的NaN值
    df_corr = df[['Budget', 'Gross']].dropna()
    if not df_corr.empty:
        correlation = df_corr['Budget'].corr(df_corr['Gross'])
        print(f"\n--- 预算与票房之间的皮尔逊相关系数 ---")
        print(f"Budget vs Gross: {correlation:.3f}")

        # 解读：
        # 接近1：强正相关（预算越高，票房越高）
```

```
        # 接近0：弱相关或无相关
        # 接近-1：强负相关（预算越高，票房越低，这在此场景下不太可能）
    else:
        print("警告：无法计算预算与票房的相关性，因为数据不足或存在问题。")
else:
    print("警告：'Budget' 或 'Gross' 列不存在，无法分析预算与票房关
系。")
```

- 运行此单元格。

5. **分析电影时长与评分的关系：**
   - **操作：** 检查电影时长是否影响观众的评分。
   - **示例代码：**

```
if 'Runtime' in df.columns and 'Rating' in df.columns: # 假设有
Runtime列
    # 假设Runtime是分钟数，且是数值类型
    # 可以计算时长的平均值、中位数等
    avg_runtime = df['Runtime'].mean()
    median_runtime = df['Runtime'].median()
    print(f"\n--- 电影时长统计 ---")
    print(f"平均时长: {avg_runtime:.1f} 分钟")
    print(f"中位数时长: {median_runtime:.1f} 分钟")

    # 如果想看时长分组的平均评分，可以先将时长分段
    # 例如，将时长分成“短片”、“中等时长”、“长片”
    # bins = [0, 90, 120, df['Runtime'].max()]
    # labels = ['Short', 'Medium', 'Long']
    # df['Runtime_Category'] = pd.cut(df['Runtime'], bins=bins,
labels=labels, right=False)
    # avg_rating_by_runtime_cat = df.groupby('Runtime_Category')
['Rating'].mean()
    # print("\n--- 不同时长分类的平均评分 ---")
    # print(avg_rating_by_runtime_cat)
else:
    print("警告：'Runtime' 或 'Rating' 列不存在，无法分析时长与评分关
系。")
```

   - 运行此单元格。

五、数据可视化 (使用Matplotlib)

本职责的核心是使用Matplotlib库创建图表，将前面步骤中的分析结果以直观的方式呈现出来。

1. **导入Matplotlib库：**
   - 在Notebook中，确保已导入Matplotlib。

- **示例代码：**

```
import matplotlib.pyplot as plt
import seaborn as sns # Seaborn基于Matplotlib，常用于美化图表
sns.set(style="whitegrid") # 设置 Seaborn 的默认样式
```

- 运行此单元格。

2. **绘制评分分布直方图：**

- **目的：** 展示所有电影评分的分布情况。
- **示例代码：**

```
if 'Rating' in df.columns:
    plt.figure(figsize=(10, 6)) # 设置图表尺寸
    plt.hist(df['Rating'].dropna(), bins=20, color='skyblue',
edgecolor='black')
    plt.title('Distribution of Movie Ratings', fontsize=16)
    plt.xlabel('Rating', fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
    plt.grid(axis='y', alpha=0.75) # 显示网格线
    plt.show() # 显示图表
else:
    print("警告：'Rating' 列不存在，无法绘制评分分布直方图。")
```

- 运行此单元格。

3. **绘制不同类型电影的平均评分条形图：**

- **目的：** 直观比较不同电影类型的平均得分高低。
- **示例代码：**

```
if 'avg_rating_by_genre' in locals(): # 假设已在EDA步骤中计算
    plt.figure(figsize=(12, 7))
    avg_rating_by_genre.plot(kind='bar',
color=sns.color_palette('viridis', len(avg_rating_by_genre))) # 使
用 Seaborn 调色板
    plt.title('Average Rating by Genre', fontsize=16)
    plt.xlabel('Genre', fontsize=12)
    plt.ylabel('Average Rating', fontsize=12)
    plt.xticks(rotation=45, ha='right') # 旋转X轴标签以防重叠
    plt.tight_layout() # 自动调整布局，防止标签被截断
    plt.show()
else:
    print("警告：未找到 'avg_rating_by_genre' 数据，请先完成EDA中的相关
计算。")
```

- 运行此单元格。

4. **绘制预算与票房的散点图：**

- **目的：** 展示电影预算与其票房收入的关系。
- **示例代码：**

```
if 'Budget' in df.columns and 'Gross' in df.columns:
    # 清理数据，移除预算或票房为0或NaN的情况，以获得更清晰的图
    plot_df = df.dropna(subset=['Budget', 'Gross'])
    plot_df = plot_df[(plot_df['Budget'] > 0) & (plot_df['Gross']
> 0)] # 过滤掉非正值

    if not plot_df.empty:
        plt.figure(figsize=(10, 6))
        plt.scatter(plot_df['Budget'], plot_df['Gross'],
alpha=0.5, color='salmon')
        plt.title('Budget vs. Gross Revenue', fontsize=16)
        plt.xlabel('Budget', fontsize=12)
        plt.ylabel('Gross Revenue', fontsize=12)
        plt.ticklabel_format(style='plain', axis='both') # 避免科学
计数法显示
        plt.grid(True)
        plt.show()
    else:
        print("警告：在过滤后，没有足够的预算与票房数据可供绘制散点图。")
else:
    print("警告：'Budget' 或 'Gross' 列不存在，无法绘制散点图。")
```

- 运行此单元格。

5. **绘制年份与平均评分的折线图：**

- **目的：** 展示电影平均评分随时间（年份）的变化趋势。
- **示例代码：**

```
if 'avg_rating_by_year' in locals(): # 假设已在EDA步骤中计算
    plt.figure(figsize=(12, 6))
    # 可以选择只绘制部分年份的数据，例如最近20-30年
    recent_avg_rating =
avg_rating_by_year[avg_rating_by_year.index >= (df['Year'].max()
- 30)] # 绘制最近30年的数据
    recent_avg_rating.plot(kind='line', marker='o', color='teal')
    plt.title('Average Movie Rating Over Years (Last 30 Years)',
fontsize=16)
    plt.xlabel('Year', fontsize=12)
    plt.ylabel('Average Rating', fontsize=12)
    plt.grid(True)
    plt.show()
else:
```

```
    print("警告：未找到 'avg_rating_by_year' 数据，请先完成EDA中的相关
计算。")
```

- 运行此单元格。