项目说明文档

该项目一共分为：

1. 提取关键词
2. 数据预处理
3. LDA建模
4. 用户画像分类
5. k-means聚类 用户类型划分
6. 数据分析（用户数据可视化）

1、提取关键词

步骤说明：

- **1.1 导入依赖库**
  - 使用`pandas`处理Excel数据，`jieba`进行中文分词，`numpy`计算矩阵，`sklearn`计算TF-IDF权重。
  - **方法**：通过`import`语句加载所需库。
- **1.2 加载自定义词典与停用词**
  - **目的**：优化分词效果，过滤无意义词汇。
  - 方法
    - `jieba.load_userdict("custom_dict.txt")`：加载自定义词典（如学科术语）。
    - `stopwords_cn.txt`：加载中文停用词表，过滤“的”、“了”等常见词。
  - **结果**：提升分词准确性，减少噪声干扰。

2. 合并Excel表格（`merge_excel_sheets`函数）

步骤说明：

- 2.1 读取所有Sheet
  - **方法**：`pd.ExcelFile(input_file)`读取输入文件的所有Sheet名称。
- 2.2 纵向合并数据
  - **方法**：循环读取每个Sheet，添加`来源sheet`列标记来源，使用`pd.concat`合并所有DataFrame。
- 2.3 删除冗余列
  - **方法**：合并后删除`视频时长`、`视频地址`等无关列。
  - **潜在问题**：代码中保存合并数据到Excel的操作（`to_excel`）在删除列之前，导致保存的文件仍包含冗余列。**建议调整顺序**：先删除列，再保存。
- 2.4 输出结果
  - **结果**：生成合并后的Excel文件（如`小学-总数据.xlsx`），总行数通过`print`输出。

3. 关键词提取（`tfidf_textrank`函数）

步骤说明：

- **3.1 自定义分词函数**
  - 方法
    - 使用`jieba.cut(use_paddle=True)`启动Paddle模式（适合专有名词分词）。
    - 过滤停用词、单字词、数字，但**未实现词性筛选**（注释中提到的"保留名词"未实际执行）。
  - **潜在问题**：可能包含动词或其他词性词汇，需通过`pseg`模块获取词性并过滤。
- **3.2 生成TF-IDF矩阵**
  - 方法
    - 将分词结果转换为空格分隔的字符串（`processed_docs`）。
    - `TfidfVectorizer`计算TF-IDF权重，`min_df=2`忽略低频词。
  - **结果**：得到每个词的全局TF-IDF均值（`global_tfidf`）。
- **3.3 TextRank关键词提取**
  - **方法**：`jieba.analyse.textrank`从全文提取Top20关键词，允许名词、动词等词性。
  - **结果**：生成关键词及其TextRank得分（`textrank_scores`）。
- **3.4 综合加权算法**
  - 方法
    - 加权公式：`0.6*TextRank + 0.4*TF-IDF`，并乘以词频权重`tf_weight`。
    - **词频计算问题**：`processed_docs.count(word)`错误（因`processed_docs`为字符串列表），应展开为词列表再统计。
  - **优化建议**：使用`collections.Counter`统计准确词频。
- **3.5 生成关键词排名**
  - **方法**：合并TF-IDF和TextRank的候选词，按综合得分排序。
  - **结果**：输出包含关键词和得分的Excel文件（如`小学-Top300关键词.xlsx`）。

4. 主程序执行流程

步骤说明：

- 4.1 循环处理各学段数据
  - **方法**：遍历`['小学','初中','高中']`，依次处理对应Excel文件。
- 4.2 数据清洗
  - **方法**：`df.dropna(subset=['内容'])`删除内容为空的行。
- 4.3 调用关键词提取函数
  - 结果
    ：每个学段生成两个文件：
    1. `{学段}-总数据.xlsx`：合并后的原始数据。
    2. `{学段}-Top100关键词.xlsx`：关键词排名表。

总结

- **输入**：分Sheet存储的Excel文件（如`小学学段老师.xlsx`）。
- 输出
    - 合并后的总数据文件（删除冗余列）。
    - 各学段Top300关键词文件（按综合得分排序）。
- **关键技术**：TF-IDF权重、TextRank算法、自定义分词、停用词过滤。

2、数据预处理

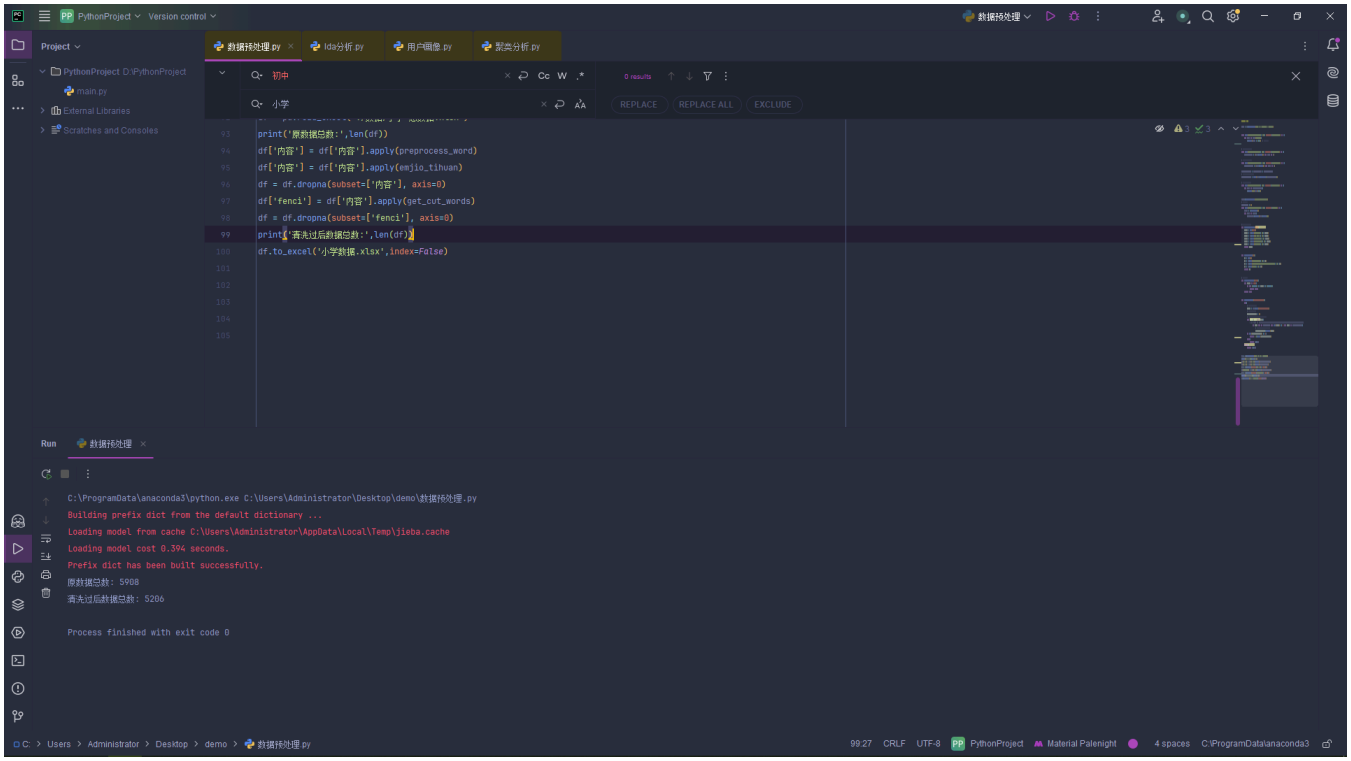
1. 代码功能概述

该代码用于对“小 初 高-总数据.xlsx”中的文本内容进行深度清洗和分词处理，结合自定义词典和停用词表，最终生成可用于后续分析的结构化数据。核心功能包括：

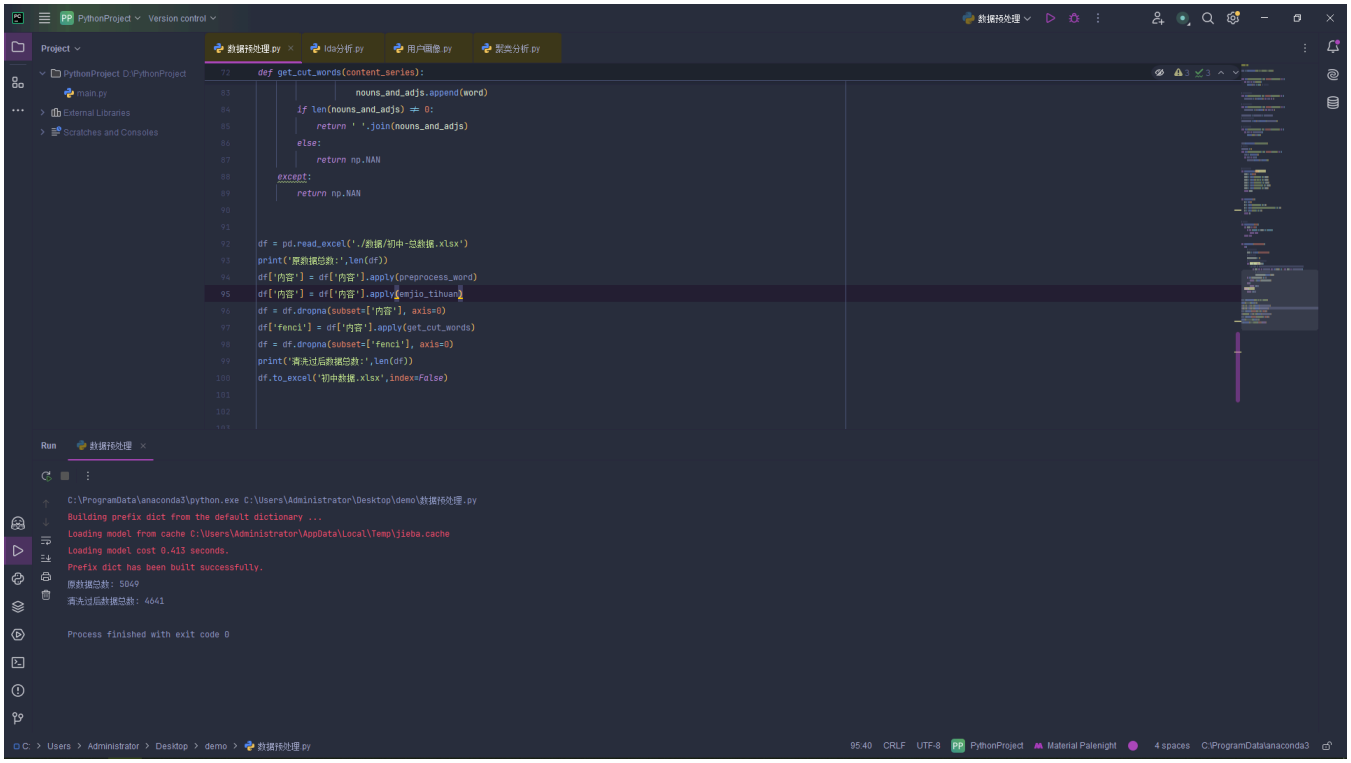
1. **合并自定义词典**：将现有词典与Top100关键词合并，优化分词效果。
2. **文本预处理**：去除标点、表情、特殊符号等噪声。
3. **中文词性过滤**：保留名词、动词等有效词汇。
4. **数据清洗**：删除空值及无效内容

最后效果：

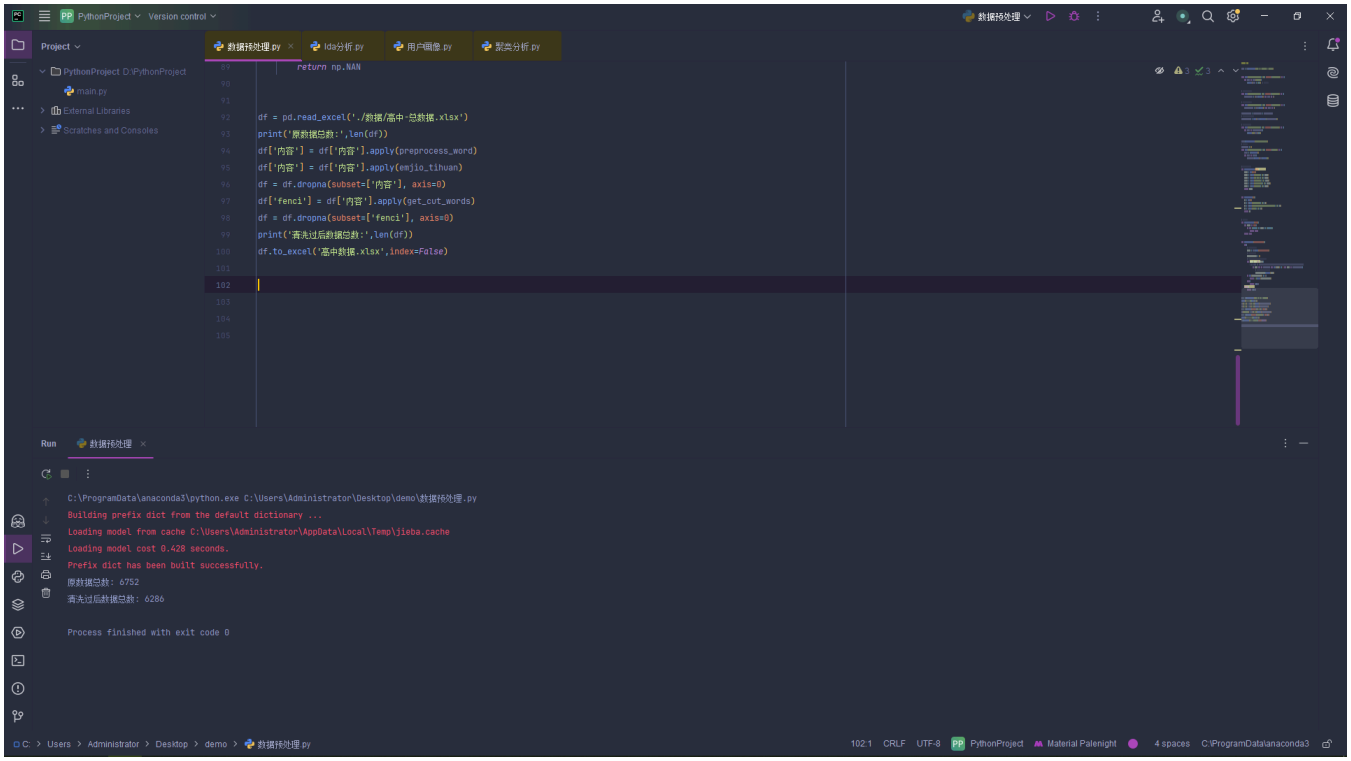
小学


print('原数据总数:',len(df))
df['内容'] = df['内容'].apply(preprocess_word)
df['内容'] = df['内容'].apply(emjio_tihuan)
df = df.dropna(subset=['内容'], axis=0)
df['fenci'] = df['内容'].apply(get_cut_words)
df = df.dropna(subset=['fenci'], axis=0)
print('清洗过后数据总数:',len(df))
df.to_excel('小学数据.xlsx',index=False)
C:\ProgramData\anaconda3\python.exe C:\Users\Administrator\Desktop\demo\数据预处理.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\Administrator\AppData\Local\Temp\jieba.cache
Loading model cost 0.394 seconds.
Prefix dict has been built successfully.
原数据总数: 5908
清洗过后数据总数: 5206
Process finished with exit code 0

初中


def get_cut_words(content_series):
nouns_and_adjs.append(word)
if len(nouns_and_adjs) != 0:
return ' '.join(nouns_and_adjs)
else:
return np.NAN
except:
return np.NAN
df = pd.read_excel('./数据/初中-总数据.xlsx')
print('原数据总数:',len(df))
df['内容'] = df['内容'].apply(preprocess_word)
df['内容'] = df['内容'].apply(emjio_tihuan)
df = df.dropna(subset=['内容'], axis=0)
df['fenci'] = df['内容'].apply(get_cut_words)
df = df.dropna(subset=['fenci'], axis=0)
print('清洗过后数据总数:',len(df))
df.to_excel('初中数据.xlsx',index=False)
C:\ProgramData\anaconda3\python.exe C:\Users\Administrator\Desktop\demo\数据预处理.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\Administrator\AppData\Local\Temp\jieba.cache
Loading model cost 0.413 seconds.
Prefix dict has been built successfully.
原数据总数: 5049
清洗过后数据总数: 4641
Process finished with exit code 0

高中



3、LDA建模

1. 代码功能概述

本代码用于对小学、初中、高中三个学段的分词数据进行**LDA主题建模**，通过无监督学习挖掘文本中的隐含主题，并生成可视化分析结果。核心功能包括：

1. **主题模型构建**：通过LDA算法提取文本主题。
2. **主题可视化**：生成交互式主题分布图。
3. **主题强度分析**：统计各主题的文章数量及权重。
4. **主题相似性分析**：计算主题间余弦相似度热力图。

2. 代码分步说明

步骤1：导入依赖库

```
Pythonimport pandas as pd  # 数据处理
from sklearn.metrics.pairwise import cosine_similarity  # 相似度计算
from gensim.models import LdaModel  # LDA模型
import pyLDAvis.gensim  # LDA可视化
```

- **关键库**：`gensim`用于LDA建模，`pyLDAvis`生成交互式可视化，`plotly`绘制高级图表。

步骤2：数据预处理

```
Pythondef lda(df):
    # 过滤停用词和短词
    train = []
    for line in df['fenci']:
        line = [word for word in line.split(' ')
               if len(word) >= 2 and word not in stop_word]
        train.append(line)
```

- **输入**：包含`fenci`列（空格分隔分词结果）的DataFrame。
- **处理**：去除长度<2的词和停用词，生成`train`列表作为模型输入。

步骤3：构建词典与语料库

```
Python# 创建词袋模型词典
dictionary = corpora.Dictionary(train)
# 转换为稀疏向量格式
corpus = [dictionary.doc2bow(text) for text in train]
```

- 输出
  - `dictionary`：词ID映射表（如`{'数学':0, '函数':1,...}`）。
  - `corpus`：文档-词频矩阵（如`[(0,1), (1,3)]`表示"数学"出现1次，"函数"出现3次）。

步骤4：LDA模型训练

```
Python# （原注释代码支持自动选择最佳主题数）
lda_model = LdaModel(corpus=corpus, id2word=dictionary,
                     num_topics=best_topic_number, iterations=400)
```

- 参数
  - `iterations=400`：增加迭代次数提升收敛性。

步骤5：主题可视化

```
Python# 生成交互式LDA可视化
vis_data = pyLDAvis.gensim.prepare(lda_model, corpus, dictionary)
pyLDAvis.save_html(vis_data, './整体-lda/整体-lda.html')
```

- **输出文件**：`整体-lda.html`（支持拖动查看主题词分布）。

步骤6：文档主题分配

```
Python# 为每篇文章分配主要主题
df['主题类型'] = [max(i, key=lambda x:x[1])[0]
                  for i in lda_model.get_document_topics(corpus)]
```

- **逻辑**：选择概率最高的主题作为文章标签。
- 输出列
  - `主题类型`：0/1/2表示所属主题编号。

步骤7：主题词提取与统计

```
Python# 提取每个主题的前20关键词
topic_words = lda_model.print_topics(num_words=20)
# 保存主题词及权重
df2 = pd.DataFrame({
    'Topic-主题词': [re.findall(r'"(\w+)"', str(topic)) for topic in topic_words],
    'Topic-权重': [re.findall(r'\d\.\d{3}', str(topic)) for topic in topic_words]
})
df2.to_csv('./整体-lda/主题词分布表.csv')
```

- 示例输出

Topic-主题词	Topic-权重
[数学, 函数, 导数...]	[0.123, 0.098...]

步骤8：主题强度分析

```
Python# 统计各主题文章数量
topic_counts = df['主题类型'].value_counts().sort_index()
# 计算主题强度（占比）
topic_strength = topic_counts / topic_counts.sum()
```

- **输出文件**：`特征词.csv`，包含主题词、文章数、强度比例。

步骤9：主题相似性热力图

```
Python# 计算主题间余弦相似度
topic_similarity = cosine_similarity(lda_model.get_topics())
# 生成交互式热力图
fig = px.imshow(topic_similarity, labels=dict(x="Topic", y="Topic"))
fig.write_html('./整体-lda/heatmap.html')
```

- **解读**：颜色越红表示主题越相似，蓝色表示差异大。

3. 执行流程与输出

```
Pythonif __name__ == '__main__':
    # 合并三学段数据
    df = pd.concat([pd.read_excel(f'{x}数据.xlsx') for x in ['小学','初中','高中']])
    lda(df)
```

- **输入文件**：`小学数据.xlsx`，`初中数据.xlsx`，`高中数据.xlsx`（需包含`fenci`列）。
- 输出目录

  ```
  ./整体-lda/
  ```

  ，包含：

- `整体-lda.html`：交互式主题可视化
- `heatmap.html`：主题相似性热力图
- `特征词.csv`：主题强度统计表
- `主题词分布表.csv`：详细主题词权重

运行结果如下：

```
C:\ProgramData\anaconda3\python.exe C:\Users\Administrator\Desktop\demo\lda分析.py

Process finished with exit code 0
```

4、用户画像

1. 代码功能概述

本代码用于整合多源数据（发帖内容、互动数据、用户信息），通过特征融合生成**教师用户画像**。核心功能包括：

1. **LDA主题分析**：提取用户发帖的主要主题。
2. **发帖行为统计**：计算发帖量、互动数据等指标。
3. **用户属性整合**：关联IP属地、粉丝量、博主类型等信息。
4. **画像合成**：合并所有特征并生成最终画像表。

2. 代码分步说明

步骤1：定义三大处理函数

函数1：`data_lda` - LDA主题分析

```
Pythondef data_lda(df, name):
    # 筛选指定作者数据
    df1 = df[df['作者'] == name]
    # 合并所有分词并统计主题分布
    list_word = [w for i in df1['fenci'] for w in i.split(" ")]
    main_topic = Counter(df1['主题类型']).most_common(1)[0][0]  # 高频主题
    return name, main_topic, ' '.join(list_word)
```

- **输入**：包含`fenci`（分词结果）、`主题类型`（LDA结果）的数据。
- **输出**：用户姓名、主要发帖主题、所有分词合并文本。

函数2：`data2` - 发帖行为统计

```
Pythondef data2(df, name):
    df1 = df[df['作者'] == name]
    # 关键指标计算
    posts = len(df1)  # 总发帖量
    length = df1['内容'].str.len().mean()  # 平均发帖长度
    # 处理数值型互动数据（万转数值）
    df1['喜欢'] = df1['喜欢'].apply(lambda x: float(x.replace('万',''))*10000 if '万' in
str(x) else x)
    like = df1['喜欢'].mean()  # 平均点赞
    comment = df1['评论'].mean()  # 平均评论
    collect = df1['收藏'].mean()  # 平均收藏
    # 高频发帖类型
    main_type = Counter(df1['笔记类型']).most_common(1)[0][0]
    return name, main_type, posts, length, like, comment, collect
```

- **输入**：原始发帖数据（含互动指标）。
- **输出**：发帖类型、互动指标均值等7个字段。

函数3：`data3` - 用户属性提取

```
Pythondef data3(df, name):
    df1 = df[df['作者'] == name]
    # 粉丝量/关注量处理
    df1['粉丝'] = df1['粉丝'].apply(lambda x: float(x.replace('万',''))*10000 if '万' in
str(x) else x)
    fan = df1['粉丝'].values[0]  # 粉丝数
    focus = df1['关注'].values[0]  # 关注数
    # 提取博主标签和IP属地
    tags = df1['tags'].str.extract(r"博主.*?(\w+)")[0].values[0]  # 标签解析
    ip = df1['用户属地'].str.split('：').str[1].values[0]  # 解析属地
    return name, ip, tags, fan, focus
```

- **输入**：用户信息表（含`tags`、`用户属地`等列）。
- **输出**：用户基础属性5个字段。

步骤2：主程序执行流程

1. 加载数据源

```
Python# LDA分析数据
df1 = pd.read_csv('./整体-lda/整体-lda_data.csv')  # 含主题类型
# 原始发帖数据
df2 = pd.concat([pd.read_excel(f'./数据/{x}学段-总数据.xlsx') for x in ['小学','初中','高中']])
# 用户属性数据
df3 = pd.concat([pd.read_excel('./数据/用户信息列表1.xlsx'),
                 pd.read_excel('./数据/用户信息列表2.xlsx')])
```

2. 并行处理三类数据

```
Python# 处理LDA数据
list_df1 = []
for name in df1['作者'].unique():
    record = data_lda(df1, name)
    list_df1.append(pd.DataFrame([record], columns=['name','lda','words']))
new_df1 = pd.concat(list_df1)

# 处理发帖行为数据（同理生成new_df2）
# 处理用户属性数据（同理生成new_df3）
```

3. 数据合并与增强

```
Python# 合并三类数据
merged_df = pd.merge(new_df1, new_df2, on='name')
merged_df = pd.merge(merged_df, new_df3, on='name')

# 补充人工生成字段
merged_df['性别'] = np.random.choice(['女','男'], size=len(merged_df), p=[0.9,0.1])  # 假设90%为女性
merged_df['博主类型'] = merged_df['博主类型'].fillna('其他博主')  # 空值填充
```

4. 输出结果

```
merged_df.to_excel('整体教师用户画像.xlsx', index=False)
```

3. 输出文件说明

字段说明:

字段名	描述	示例值
name	教师姓名	张老师
lda	主要发帖主题编号	2
words	合并后的所有分词	"数学 函数 导数..."
发帖类型	高频笔记类型	教学视频
平均喜好	平均点赞数	356.78
IP属地	用户注册地	广东
博主类型	账号标签	教育博主
粉丝数量	粉丝总量	12000

文件示例:

name	lda	words	发帖类型	粉丝数量	性别	...
张老师	0	数学 函数...	教学视频	12000	女	...

输出的内容：

红色为告警信息，无视即可

5、聚类分析

1. 代码功能概述

本代码用于对教师用户画像数据进行**K-Means聚类分析**，通过无监督学习发现用户群体的潜在分类模式，并生成多维可视化报告。核心功能包括：

1. **数据预处理**：特征标准化与编码。
2. **聚类分析**：通过肘部法则与轮廓系数确定最佳聚类数。
3. **可视化输出**：生成相关性热力图、聚类图、词云等。
4. **结果解释**：分析不同聚类群体的特征差异。

2. 代码分步说明

步骤1：数据加载与预处理

```
Python# 读取画像数据并清洗列名
data = pd.read_excel("整体教师用户画像.xlsx")
data.columns = [col.strip() for col in data.columns]

# 选择数值型特征并编码分类变量
numeric_cols = ['lda','总发帖量','平均发帖长度','平均喜好','平均评论数','平均收藏','粉丝数量','关注数量','发帖类型','IP属地','博主类型','性别']
data1 = data[numeric_cols].copy()
data1['发帖类型'] = LabelEncoder().fit_transform(data1['发帖类型'])  # 分类转数值
data1 = pd.DataFrame(StandardScaler().fit_transform(data1), columns=data1.columns)  # 标准化
```

- 关键操作
  - **标签编码**：将文本型分类（如`发帖类型`）转换为数值。
  - **标准化**：消除量纲差异（如`粉丝数量`与`平均喜好`）。

步骤2：特征相关性分析

```
Python# 生成交互式热力图
fig = px.imshow(
    data1.corr().round(2),
    color_continuous_scale="RdBu",
    title="Feature Correlation Heatmap"
)
fig.write_html(f"./{name}/feature_correlation_heatmap.html")
```

- **输出文件**：`feature_correlation_heatmap.html`
- **解读**：红色表示正相关（如`平均喜好`与`平均收藏`），蓝色表示负相关。

步骤3：确定最佳聚类数

```
Python# 计算不同K值的轮廓系数与肘部系数
range_n_clusters = range(2,11)
for n_clusters in range_n_clusters:
    labels = KMeans(n_clusters).fit_predict(data1)
    silhouette_scores.append(silhouette_score(data1, labels))
    wcss.append(KMeans().inertia_)
```

- 方法
  - **肘部法则**：寻找WCSS（簇内平方和）下降拐点。
  - **轮廓系数**：衡量聚类紧密度与分离度（值越接近1越好）。
- **输出文件**：`轮廓系数.csv`（包含K=2~10的评估指标）。

步骤4：模型训练与可视化

```
Python# 训练K-Means模型（硬编码K=3）
clf = KMeans(n_clusters=3).fit(data1)
data['聚类结果'] = clf.labels_

# PCA降维可视化
pca = PCA(n_components=2)
xy = pca.fit_transform(data1)
plt.scatter(xy[:,0], xy[:,1], c=data['聚类结果'])
plt.savefig(f'./{name}/聚类图.jpg')
```

- 输出文件
  - `聚类结果.csv`：带聚类标签的原始数据。
  - `聚类图.jpg`：二维投影后的分布图。

步骤5：生成聚类词云

```
Python# 遍历每个聚类生成词云
for cluster_id in data['聚类结果'].unique():
    words = ' '.join(data[data['聚类结果']==cluster_id]['words'])
    wc = WordCloud(font_path='simhei.ttf', mask=background_image)
    wc.generate(words)
    wc.to_file(f'./{name}/image/聚类{cluster_id}-词云图.png')
```

- **依赖数据**：`words`列（来自用户画像的分词结果）。
- 输出示例
  - 聚类0词云：高频词为"xxx、xxx、xx"
  - 聚类1词云：高频词为"xxx、xxx、xxx"

3. 输出文件说明

文件/目录	内容描述
`feature_correlation_heatmap.html`	交互式特征相关性矩阵
`轮廓系数.csv`	K=2~10的轮廓系数与肘部系数记录
`聚类结果.csv`	原始数据+聚类标签
`聚类图.jpg`	PCA降维后的二维聚类分布图
`image/聚类X-词云图.png`	各聚类的关键词可视化

输出内容：

代码提示选择2聚类最好，不过2个人感觉有点少，就聚成4个类，4也是一个突出的点

6、数据分析

1. 代码功能概述

本代码用于对聚类后的教师用户群体进行**多维特征可视化分析**，通过饼图、柱状图等形式展示各聚类群体的属性分布特征，支持精细化运营策略制定。核心功能包括：

1. **数据分箱**：将连续型指标（如发帖量、粉丝数）利用四分位数（25%、50%、75%、100%）转换为分类标签。
2. **分布可视化**：生成12类特征分布图，覆盖基础属性、互动指标、内容特征。
3. **聚类对比**：针对每个聚类群体独立生成全套分析图表。

2. 代码分步说明

步骤1：数据预处理与分箱

```
Python# 数值型指标分段处理（示例：总发帖量）
def post_number(x):
    x1 = int(x)
    if x1 <= 21: return '发帖频率：一般'
    elif 21 < x1 <= 28: return '发帖频率：较为积极'
    ...

df['总发帖量'] = df['总发帖量'].apply(post_number)  # 应用分段函数
```

- **分箱逻辑**：将连续值转换为分类标签（如将发帖量分为"一般/积极/频繁"）。
- **覆盖指标**：发帖量、发帖长度、点赞、评论、收藏、粉丝数、关注数等7个指标。

步骤2：定义可视化函数（以`post_type`为例）

```
Pythondef post_type(df, name):
    # 统计频次
    new_df = df['发帖类型'].value_counts()
    # 创建饼图
    fig, ax = plt.subplots(figsize=(16,9))
    ax.pie(new_df.values, labels=new_df.index,
           autopct=lambda p: f'{p:.1f}%\n({int(p*sum(new_df.values)/100)})',
           wedgeprops=dict(width=0.4))
    # 保存图表
    plt.savefig(f'./聚类-{name}/发帖类型分布情况.png')
```

- 通用结构
    1. 使用`value_counts()`统计分类频次。
    2. 创建带百分比标签的环形饼图。
    3. 保存到聚类专属目录（如`聚类-0/发帖类型分布情况.png`）。

步骤3：执行聚类分组分析

```
Python# 遍历每个聚类群体
for d in df['聚类结果'].unique():
    df2 = df[df['聚类结果'] == d]  # 筛选当前聚类数据
    post_type(df2, d)    # 发帖类型分析
    lda_type(df2, d)     # LDA主题分析
    blogger_type(df2, d) # 博主类型分析
    ... # 共调用12个分析函数
```

- 输出结构

```
 聚类-0/
├── 发帖类型分布情况.png
├── LDA分布情况.png
├── 博主类型分布情况.png
...（共12张图表）
```

3. 关键可视化类型说明

图表类型	对应函数	分析维度	示例输出
环形饼图	post_type	发帖类型分布	教学视频(65%)、习题解析(35%)
环形饼图	lda_type	内容主题分布	主题1(40%)、主题2(30%)、主题3(30%)
柱状图	ip_type	地域分布	广东(120人)、北京(80人)
分段环形饼图	fan_type	粉丝量级分布	低(50%)、中(30%)、高(20%)

4. 技术亮点与优化建议

技术亮点:

1. **自动化分箱**：通过阈值函数将连续数据转换为业务可解释的分类标签。
2. **批量可视化**：统一图表样式，支持多聚类群体快速对比分析。
3. **交互式设计**：饼图同时显示百分比与绝对值，增强可读性。

5. 输出结果应用

- **运营策略**：根据"高粉丝-高互动"群体特征，制定创作者激励计划。
- **内容优化**：针对"高频发帖-低互动"群体，推荐内容质量提升课程。
- **资源分配**：依据地域分布特征，调整区域化运营资源投入。

输出内容如下: