Python数据科学分析通用流程

0 启动Jupyter

- Terminal中

```
jupyter notebook
```

1. 数据导入 (Lecture 5)

1.1 必要库导入

```
# 基础数据处理
import pandas as pd
import numpy as np

# 数据可视化
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# 常用数据格式处理
import json
import xml.etree.ElementTree as ET
from bs4 import BeautifulSoup

# 机器学习相关
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score
```

1.2 基础数据导入示例

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

1.3 不同格式数据读取

CSV文件读取

```
# 基本读取
df = pd.read_csv("file.csv")

# 带参数读取
df = pd.read_csv("file.csv",
                 index_col="column_name",  # 设置索引列
                 parse_dates=["date"],     # 解析日期
                 sep="\t",                 # 设置分隔符
                 encoding="utf-8")         # 设置编码

# 检查数据基本信息
print(f"数据维度: {df.shape}")
print("\n数据类型信息:")
print(df.info())
print("\n数据预览:")
print(df.head())
```

JSON文件读取

```
# 方法1：直接读取JSON文件
df = pd.read_json("file.json")

# 方法2：使用json模块读取
with open("file.json", "r", encoding="utf-8") as f:
    data = json.load(f)
    df = pd.DataFrame(data)
```

XML文件读取

```
# 解析XML文件
tree = ET.parse("file.xml")
root = tree.getroot()

# 示例：提取所有book元素的信息
data = []
for book in root.findall("book"):
    book_data = {
        "title": book.find("title").text,
        "author": book.find("author").text,
        "year": book.find("year").text
    }
    data.append(book_data)
df = pd.DataFrame(data)
```

HTML文件读取

```
# 从文件读取HTML
with open("file.html", "r", encoding="utf-8") as f:
    soup = BeautifulSoup(f, "html.parser")

# 示例：提取表格数据
tables = pd.read_html("file.html")  # 返回所有表格的列表
df = tables[0]  # 获取第一个表格
```

- CSV文件

```
df = pd.read_csv("file.csv", index_col="column_name")
```

- JSON文件

```
import json
with open("file.json", "r") as f:
    data = json.load(f)
```

- XML文件

```
import xml.etree.ElementTree as ET
tree = ET.parse("file.xml")
```

- HTML文件

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_content, "html.parser")
```

1.3 初步检查

- 数据基本信息

```
df.info()  # 数据类型和非空值统计
df.head()  # 查看前几行
df.shape   # 查看维度
```

2. 数据清洗和预处理 (Lecture 6)

2.1 缺失值处理

缺失值检测

```
# 检查每列的缺失值数量
```

```
missing_counts = df.isnull().sum()
missing_percentages = (missing_counts / len(df)) * 100

# 创建缺失值统计DataFrame
missing_stats = pd.DataFrame({
    '缺失值数量': missing_counts,
    '缺失值比例%': missing_percentages
})
print(missing_stats[missing_stats['缺失值数量'] > 0])

# 可视化缺失值
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), yticklabels=False, cbar=True, cmap='viridis')
plt.title('Missing Value Heatmap')
plt.show()
```

缺失值处理

```
# 方法1：删除缺失值
df_cleaned = df.dropna()  # 删除任何包含缺失值的行
df_cleaned = df.dropna(subset=['column_name'])  # 删除特定列包含缺失值的行

# 方法2：填充数值型缺失值
df['col'] = df['col'].fillna(df['col'].mean())  # 均值填充
df['col'] = df['col'].fillna(df['col'].median())  # 中位数填充
df['col'] = df['col'].fillna(df['col'].mode()[0])  # 众数填充

# 方法3：时间序列数据的填充
df['col'] = df['col'].fillna(method='ffill')  # 前向填充
df['col'] = df['col'].fillna(method='bfill')  # 后向填充

# 方法4：分组填充
df['col'] = df['col'].fillna(df.groupby('category')['col'].transform('mean'))
```

- 检测缺失值

```
df.isnull().sum()  # 每列缺失值统计
df.isna().sum()    # 替代方法
```

- 处理缺失值
  - 删除：dropna()
  - 填充均值：fillna(df.mean())
  - 填充中位数：fillna(df.median())
  - 填充众数：fillna(df.mode())

    - 前向填充：fillna(method='ffill')
    - 后向填充：fillna(method='bfill')

2.2 重复值处理

- 检测重复值

```
df.duplicated().sum()
```

- 删除重复值

```
df.drop_duplicates()
```

2.3 异常值检测与处理

- 统计方法（3σ原则）

```
mean = df['column'].mean()
std = df['column'].std()
df[(df['column'] < mean-3*std) | (df['column'] > mean+3*std)]
```

- 箱线图法（IQR方法）

```
Q1 = df['column'].quantile(0.25)
Q3 = df['column'].quantile(0.75)
IQR = Q3 - Q1
df[(df['column'] < Q1-1.5*IQR) | (df['column'] > Q3+1.5*IQR)]
```

2.4 数据类型转换

- 数值转换

```
df['column'] = pd.to_numeric(df['column'])
```

- 日期转换

```
df['date'] = pd.to_datetime(df['date'])
```

- 类型转换

```
df['column'] = df['column'].astype('type')
```

2.5 特征工程

- 数值标准化/归一化

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

- 分箱处理

```
pd.cut() # 等宽分箱
pd.qcut() # 等频分箱
```

- 独热编码

```
pd.get_dummies()
```

3. 探索性数据分析 (Lecture 6,7,8)

3.1 基本统计分析

- 描述性统计

```
df.describe() # 数值列统计
df.describe(include=['object']) # 类别列统计
```

- 单变量分析
  - 集中趋势：mean(), median(), mode()
  - 离散趋势：std(), var(), quantile()
  - 分布情况：skew(), kurtosis()

3.2 分组聚合分析

基本分组统计

```
# 单列分组统计
group_stats = df.groupby('category_col')['value_col'].agg([
    'count',              # 计数
    'mean',               # 平均值
    'std',                # 标准差
    'min',                # 最小值
    'max',               # 最大值
    'sum'                # 求和
])
print("\n基本分组统计:")
```

```
print(group_stats)

# 多列分组统计
multi_group = df.groupby(['cat1', 'cat2']).agg({
    'value1': ['count', 'mean', 'std'],
    'value2': ['mean', 'sum']
})
print("\n多列分组统计:")
print(multi_group)

# 自定义聚合函数
def range_calc(x):
    return x.max() - x.min()

custom_agg = df.groupby('category_col')['value_col'].agg([
    'mean',
    'std',
    ('range', range_calc)  # 自定义聚合函数
])
```

透视表分析

```
# 基本透视表
pivot_table = pd.pivot_table(df,
                             values='value_col',       # 待统计的值
                             index='row_category',     # 行索引
                             columns='col_category',   # 列索引
                             aggfunc='mean')           # 统计函数

# 多值透视表
multi_pivot = pd.pivot_table(df,
                             values=['value1', 'value2'],
                             index='row_category',
                             columns='col_category',
                             aggfunc={
                                 'value1': 'mean',
                                 'value2': 'sum'
                             },
                             margins=True)  # 显示汇总行列

# 可视化透视表
plt.figure(figsize=(10, 6))
sns.heatmap(pivot_table, annot=True, fmt='.2f', cmap='YlOrRd')
plt.title('Pivot Table Heatmap')
plt.show()
```

交叉表分析

```python
# 基本交叉表
cross_tab = pd.crosstab(df['cat1'], df['cat2'])

# 带归一化的交叉表
norm_tab = pd.crosstab(df['cat1'], df['cat2'], normalize='index')  # 行归一化
norm_tab_all = pd.crosstab(df['cat1'], df['cat2'], normalize=True) # 总体归一化

# 带统计检验的交叉表分析
from scipy.stats import chi2_contingency

cross_tab = pd.crosstab(df['cat1'], df['cat2'])
chi2, p_value, dof, expected = chi2_contingency(cross_tab)
print(f"Chi-square statistic: {chi2:.4f}")
print(f"p-value: {p_value:.4f}")

# 交叉表可视化
plt.figure(figsize=(10, 6))
sns.heatmap(cross_tab, annot=True, fmt='d', cmap='Blues')
plt.title('Cross Tabulation Heatmap')
plt.show()
```

3.3 相关性分析

数值变量相关性

```python
# 计算相关系数矩阵
corr_matrix = df.corr(method='pearson')  # pearson相关系数
rank_corr = df.corr(method='spearman')   # spearman等级相关系数

# 相关性矩阵可视化
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix,
            annot=True,          # 显示数值
            fmt='.2f',           # 数值格式
            cmap='RdBu_r',       # 配色方案
            center=0,            # 中心值
            vmin=-1, vmax=1)     # 值范围
plt.title('Correlation Matrix Heatmap')
plt.show()

# 查找高相关性特征对
def get_high_correlations(corr_matrix, threshold=0.8):
    high_corr = []
    for i in range(len(corr_matrix.columns)):
        for j in range(i+1, len(corr_matrix.columns)):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                high_corr.append({

                    'var1': corr_matrix.columns[i],
```

```
                    'var1': corr_matrix.columns[i],
                    'var2': corr_matrix.columns[j],
                    'correlation': corr_matrix.iloc[i,j]
                })
    return pd.DataFrame(high_corr)

high_corrs = get_high_correlations(corr_matrix)
print("\n高相关性特征对:")
print(high_corrs)
```

类别变量关联性

```
# 计算Cramer's V统计量
def cramers_v(x, y):
    confusion_matrix = pd.crosstab(x, y)
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    min_dim = min(confusion_matrix.shape) - 1
    return np.sqrt(chi2 / (n * min_dim))

# 对所有类别变量对计算关联性
cat_cols = df.select_dtypes(include=['object', 'category']).columns
cat_correlations = pd.DataFrame(index=cat_cols, columns=cat_cols)

for i in cat_cols:
    for j in cat_cols:
        cat_correlations.loc[i,j] = cramers_v(df[i], df[j])

# 可视化类别变量关联性
plt.figure(figsize=(10, 8))
sns.heatmap(cat_correlations,
            annot=True,
            fmt='.2f',
            cmap='YlOrRd')
plt.title('Categorical Variables Association (Cramer\'s V)')
plt.show()
```

- 基本分组

```
df.groupby('column').agg(['mean', 'count'])
```

- 多层分组

```
df.groupby(['col1', 'col2']).agg({'col3': ['mean', 'sum']})
```

- 透视表

```
pd.pivot_table(df, values='val', index='idx', columns='col')
```

- 交叉表

```
pd.crosstab(df.col1, df.col2)
```

3.3 相关性分析

- 相关系数矩阵

```
df.corr() # Pearson相关系数
df.corr(method='spearman') # Spearman相关系数
```

- 协方差矩阵

```
df.cov()
```

4. 数据可视化 (Lecture 8)

4.0 需要导入的库

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

4.1 单变量可视化

数值变量

```
# 直方图
plt.figure(figsize=(10, 6))
plt.hist(df['numeric_col'],
         bins=30,                # 箱子数量
         color='skyblue',        # 颜色
         edgecolor='black',      # 边框颜色
         alpha=0.7)              # 透明度
plt.title('Distribution of numeric_col')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()

# 密度图

plt.figure(figsize=(10, 6))
```

```
plt.figure(figsize=(10, 6))
df['numeric_col'].plot(kind='kde',
                       color='darkblue')
plt.title('Density Plot')
plt.xlabel('Value')
plt.ylabel('Density')
plt.grid(True, alpha=0.3)
plt.show()

# 箱线图
plt.figure(figsize=(8, 6))
df.boxplot(column='numeric_col')
plt.title('Box Plot')
plt.grid(True, alpha=0.3)
plt.show()
```

类别变量

```
# 条形图
plt.figure(figsize=(10, 6))
value_counts = df['category_col'].value_counts()
value_counts.plot(kind='bar',
                  color='lightcoral')
plt.title('Frequency of Categories')
plt.xlabel('Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.grid(True, alpha=0.3)
plt.show()

# 饼图
plt.figure(figsize=(10, 8))
plt.pie(value_counts,
        labels=value_counts.index,
        autopct='%1.1f%%',      # 显示百分比
        startangle=90)
plt.title('Distribution of Categories')
plt.axis('equal')               # 保持圆形
plt.show()
```

4.2 双变量可视化

数值 vs 数值

```
# 散点图
plt.figure(figsize=(10, 6))
plt.scatter(df['x_col'],
            df['y_col'],
```

```
            df['y_col'],
            c='blue',            # 点的颜色
            alpha=0.5,           # 透明度
            s=50)                # 点的大小
plt.title('Scatter Plot')
plt.xlabel('X Value')
plt.ylabel('Y Value')
plt.grid(True, alpha=0.3)
plt.show()

# 带回归线的散点图
plt.figure(figsize=(10, 6))
sns.regplot(data=df,
            x='x_col',
            y='y_col',
            scatter_kws={'alpha':0.5},
            line_kws={'color': 'red'})
plt.title('Scatter Plot with Regression Line')
plt.show()

# 六边形密度图（适用于大数据集）
plt.figure(figsize=(10, 6))
plt.hexbin(df['x_col'],
           df['y_col'],
           gridsize=20,          # 六边形网格大小
           cmap='YlOrRd')      # 颜色映射
plt.colorbar(label='count')
plt.title('Hexbin Plot')
plt.xlabel('X Value')
plt.ylabel('Y Value')
plt.show()
```

类别 vs 数值

```
# 箱线图
plt.figure(figsize=(12, 6))
sns.boxplot(data=df,
            x='category_col',
            y='numeric_col',
            palette='Set3')
plt.title('Box Plot by Category')
plt.xticks(rotation=45)
plt.show()

# 小提琴图
plt.figure(figsize=(12, 6))
sns.violinplot(data=df,
               x='category_col',
               y='numeric_col',
```

```
                palette='Set3')
plt.title('Violin Plot by Category')
plt.xticks(rotation=45)
plt.show()

# 带分布的柱状图
plt.figure(figsize=(12, 6))
sns.barplot(data=df,
            x='category_col',
            y='numeric_col',
            ci=95,              # 95%置信区间
            palette='Set3')
plt.title('Bar Plot with CI')
plt.xticks(rotation=45)
plt.show()
```

4.3 多变量可视化

```
# 散点图矩阵
sns.pairplot(df[['num1', 'num2', 'num3', 'category']],
             hue='category',    # 按类别着色
             diag_kind='kde')   # 对角线显示密度图
plt.show()

# 多面网格图
g = sns.FacetGrid(df,
                  col='cat1',
                  row='cat2',
                  margin_titles=True)
g.map(plt.hist, 'numeric_col')

# 相关性热力图
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(),
            annot=True,
            cmap='RdYlBu',
            center=0)
plt.title('Correlation Heatmap')
plt.show()
```

4.4 时间序列可视化

```
# 基本时间序列图
plt.figure(figsize=(12, 6))
plt.plot(df.index,

         df['value_col'],
```

```
         linewidth=2,
         color='blue')
plt.title('Time Series Plot')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid(True, alpha=0.3)
plt.show()

# 多序列对比
plt.figure(figsize=(12, 6))
for column in ['series1', 'series2', 'series3']:
    plt.plot(df.index,
             df[column],
             label=column,
             alpha=0.7)
plt.title('Multiple Time Series')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# 季节性分解图
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(df['value_col'],
                                   period=12)  # 周期
fig = decomposition.plot()
plt.tight_layout()
plt.show()
```

5. 特定领域分析

5.1 时间序列分析 (Lecture 13)

- 需要导入的库

```
import pandas as pd
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
# 设置时间索引
df.index = pd.to_datetime(df['date_column'])
df = df.sort_index()

# 重采样
monthly_mean = df.resample('M').mean()  # 月度平均
weekly_sum = df.resample('W').sum()     # 周度总和
```

```
yearly_median = df.resample('Y').median()  # 年度中位数

# 移动平均
df['7D_MA'] = df['column'].rolling(window=7).mean()  # 7日移动平均
df['30D_MA'] = df['column'].rolling(window=30).mean()  # 30日移动平均

# 时间序列分解
decomposition = seasonal_decompose(df['column'], period=12)
fig = decomposition.plot()
plt.show()

# 绘制时间序列及其移动平均线
plt.figure(figsize=(12,6))
plt.plot(df.index, df['column'], label='Original')
plt.plot(df.index, df['7D_MA'], label='7-day MA')
plt.plot(df.index, df['30D_MA'], label='30-day MA')
plt.title('Time Series with Moving Averages')
plt.legend()
plt.show()
```

5.2 文本分析 (Lecture 12)

- 需导入的库:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.tokenize import word_tokenize
import nltk
```

```
# 文本预处理和词频统计
vectorizer = CountVectorizer(min_df=2, stop_words='english')
X = vectorizer.fit_transform(df['text_column'])

# 获取词频矩阵
word_freq_df = pd.DataFrame(X.toarray(),
                            columns=vectorizer.get_feature_names_out())

# TF-IDF向量化
tfidf = TfidfVectorizer(min_df=2, stop_words='english')
X_tfidf = tfidf.fit_transform(df['text_column'])

# 基本文本统计
df['text_length'] = df['text_column'].str.len()
df['word_count'] = df['text_column'].str.split().str.len()

# NLTK分词示例
nltk.download('punkt')
df['tokens'] = df['text_column'].apply(word_tokenize)
```

5.3 机器学习应用 (Lecture 10,11)

- 需导入的库:

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
```

```
# 回归分析
X = df[['feature1', 'feature2']]
y = df['target']

# 划分训练测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 线性回归
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print('R2 Score:', model.score(X_test, y_test))

# 分类分析
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# 模型评估
print('Accuracy:', accuracy_score(y_test, y_pred))
print('F1 Score:', f1_score(y_test, y_pred, average='weighted'))

# 混淆矩阵
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.show()

# 交叉验证
cv_scores = cross_val_score(model, X, y, cv=5)
print('CV Scores:', cv_scores)
print('Mean CV Score:', cv_scores.mean())
```