Web programming

Introdução Engenharia Informática

Mário Antunes

November 17, 2025

Exercícios

Objetivo

Neste laboratório, irá explorar o famoso **conjunto de dados do Titanic**. O seu objetivo é analisar os dados dos passageiros para perceber quem sobreviveu ao desastre. Irá simular um cenário do mundo real onde os dados chegam em diferentes formatos (CSV, JSON, Excel), limpar esses dados, visualizá-los e construir uma aplicação web para os processar.

Parte 0: Configuração do Ambiente

Iremos utilizar o **Docker** para criar um ambiente de programação consistente.

1. Criar Estrutura do Projeto

Execute estes comandos no seu terminal:

```
mkdir ex11
cd ex11
mkdir notebooks
mkdir data
mkdir output
```

2. O Dockerfile

Crie um ficheiro chamado `Dockerfile` dentro da pasta `ex11`:

```
# Use a lightweight Jupyter notebook image
FROM quay.io/jupyter/minimal-notebook:latest

# Switch to root to install system packages
USER root

# Install Python libraries for Data Science and Web
RUN pip install numpy pandas polars matplotlib
seaborn openpyxl fastapi uvicorn python-multipart

# Switch back to the standard user
USER ${NB_UID}
```

3. O Ficheiro Docker Compose

Crie `docker-compose.yml` dentro de `titanic_lab`:

```
services:
  notebook:
    build: .
    ports:
```

```
      - "8888:8888"
    volumes:
      - ./notebooks:/home/jovyan/work
      - ./data:/home/jovyan/data
      - ./output:/home/jovyan/output
    environment:
      - JUPYTER_TOKEN=titanic
```

4. Lançar e Gerar Dados

Execute `docker compose up --build` no seu terminal. Abra `http://localhost:8888` (palavra-passe: `titanic`).

Crie um novo *notebook* e descarregue o conjunto de dados do titanic do elearning. Coloque-o na pasta data criada anteriormente.

Parte 1: Carregamento de Dados

No mundo real, pode receber listas de passageiros em Excel dos RH, ou JSON de uma API web. Vamos aprender a carregar todos eles.

```
import pandas as pd
import polars as pl

# 1. Load CSV (Comma Separated Values)
# Most common format in Data Science
df_csv = pd.read_csv('../data/titanic.csv')
print("--- Loaded from CSV ---")
print(df_csv.head(3))

# 2. Load JSON (JavaScript Object Notation)
# Common when dealing with Web APIs
df_json = pd.read_json('../data/titanic.json')
print("\n--- Loaded from JSON ---")
print(df_json.head(3))

# 3. Load Excel (.xlsx)
# Common in business environments
df_excel = pd.read_excel('../data/titanic.xlsx')
print("\n--- Loaded from Excel ---")
print(df_excel.head(3))
```

Parte 2: Pré-processamento de Dados

O conjunto de dados do Titanic é conhecido por ter valores de Age (Idade) em falta e ocasionalmente dados desorganizados. Iremos simular dados "sujos" e corrigi-los.

Passo 1: Identificar Problemas

Criámos o conjunto de dados com um valor em falta (`None`) na coluna Age para o passageiro 6. Vamos também verificar se existem *outliers*.

```
# Check for null values
print("Missing values per column:")
print(df_csv.isnull().sum())

# Describe gives us statistics. Look at 'Age' max value.
print("\nStatistics:")
print(df_csv.describe())
```

Passo 2: Injetar um Outlier

Vamos adicionar manualmente um erro para simular um erro de introdução de dados (e.g., alguém escrever idade 200 em vez de 20).

```
df_dirty = df_csv.copy()
# Inject an impossible age
df_dirty.loc[0, 'Age'] = 200

# Visualization: Boxplot reveals the outlier
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,2))
sns.boxplot(x=df_dirty['Age'])
plt.title("Age Distribution (With Outlier)")
plt.show()
```

Passo 3: Corrigir os Dados

1. **Valores em Falta:** Preencher Age em falta com a mediana das idades dos passageiros.
2. **Outliers:** Filtrar idades irrealistas (e.g., > 100).

```
# 1. Fill missing Age with Median
median_age = df_dirty['Age'].median()
df_dirty['Age'] = df_dirty['Age'].fillna(median_age)

# 2. Remove Outliers (Cap Age at 100)
df_clean = df_dirty[df_dirty['Age'] ⩽ 100].copy()

print("Cleaned Max Age:", df_clean['Age'].max())
print("Missing Ages:", df_clean['Age'].isnull().sum())
```

Parte 3: Visualização de Dados

Vamos agora gerar gráficos para verificar explicitamente a Taxa de Sobrevivência. Em vez de apenas contar os sobreviventes, queremos ver a percentagem de pessoas que sobreviveram em diferentes grupos.

```
# Calculate Global Survival Rate
global_rate = df_clean['Survived'].mean() * 100
print(f"Overall Survival Rate: {global_rate:.2f}%")

# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Plot 1: Survival Rate by Class
# sns.barplot automatically calculates the Mean (Survival Rate) and confidence interval
sns.barplot(data=df_clean, x='Pclass', y='Survived', palette='viridis', ax=axes[0])
axes[0].set_title("Survival Rate by Passenger Class")
axes[0].set_ylabel("Survival Probability (0-1)")
axes[0].set_xlabel("Class (1=1st, 3=3rd)")
axes[0].set_ylim(0, 1)

# Plot 2: Survival Rate by Gender
sns.barplot(data=df_clean, x='Sex', y='Survived', palette='pastel', ax=axes[1])
axes[1].set_title("Survival Rate by Gender")
axes[1].set_ylabel("Survival Probability (0-1)")
axes[1].set_ylim(0, 1)

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

Análise:

- **Gráfico da Esquerda:** Verifica se os passageiros mais ricos (1ª Classe) tiveram uma maior probabilidade de sobrevivência em comparação com a 3ª Classe.
- **Gráfico da Direita:** Verifica o protocolo "Mulheres e crianças primeiro" comparando as taxas de sobrevivência Masculinas vs Femininas.

Parte 4: Exportar Resultados

Iremos guardar a análise da Taxa de Sobrevivência por Classe no volume partilhado, pois esta é a perceção mais crítica para o nosso relatório.

```
save_path = '../output/survival_rate_analysis.png'

plt.figure(figsize=(6,5))
# Re-drawing the specific plot for export
barplot = sns.barplot(data=df_clean, x='Pclass', y='Survived', palette='viridis')

# Add labels on top of bars for clarity
for p in barplot.patches:
    barplot.annotate(format(p.get_height(), '.2f'),
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha = 'center', va = 'center',
                     xytext = (0, 9),
                     textcoords = 'offset points')

plt.title("Official Report: Survival Rate by Class")
plt.xlabel("Passenger Class")
plt.ylabel("Survival Rate")
plt.ylim(0, 1)

plt.savefig(save_path, dpi=300)
print(f"Survival analysis saved to {save_path}")
```

Parte 5: Aplicação Web

Iremos criar um site simples onde um utilizador pode carregar o `titanic.csv` e obter um gráfico gerado da **Taxa de Sobrevivência por Classe**.

1. Estrutura

Crie a pasta `web_app` dentro de `ex11`. Dentro dela, crie `backend` e `frontend`.

2. Backend (FastAPI + Polars)

Crie `web_app/backend/main.py`. Esta API lê o CSV, calcula as taxas de sobrevivência usando **Polars**, e desenha um gráfico.

```
from fastapi import FastAPI, UploadFile, File
from fastapi.responses import HTMLResponse
import polars as pl
import matplotlib.pyplot as plt
import io
import base64

app = FastAPI()

@app.post("/analyze")
async def analyze_titanic(file: UploadFile = File( ... )):
    # 1. Read CSV data using Polars
    content = await file.read()
```

```
    df = pl.read_csv(content)

    # 2. Group by Pclass and calculate Mean Survival
    # (Polars syntax is different from Pandas!)
    stats = df.group_by("Pclass").agg(
        pl.col("Survived").mean().alias("Survival_Rate")
    ).sort("Pclass")

    # 3. Generate Plot
    plt.figure(figsize=(8, 6))
    # We define colors: Red for low survival, Green for high
    colors = ['#ff9999' if rate < 0.5 else '#99ff99'
    for rate in stats['Survival_Rate']]

    plt.bar(stats['Pclass'], stats['Survival_Rate'],
    color=colors, edgecolor='black')
    plt.xlabel("Passenger Class (1st, 2nd, 3rd)")
    plt.ylabel("Survival Rate (0.0 to 1.0)")
    plt.title("Titanic: Survival Rate by Class")
    plt.xticks(stats['Pclass'])
    plt.ylim(0, 1)

    # 4. Save to Buffer
    buf = io.BytesIO()
    plt.savefig(buf, format="png")
    buf.seek(0)
    img_str = base64.b64encode(buf.read()).decode("utf-8")
    plt.close()

    # 5. Return HTML
    return HTMLResponse(content=f"""
        <div style="text-align:center; font-family:sans-serif;">
            <h1>Analysis Result</h1>
            <p>Based on {df.height} passenger records.</p>
            <img src="{img_str}" />
            <br><br>
            <a href="/">Analyze Another File</a>
        </div>
    """)
```

Crie web_app/backend/Dockerfile:

```
FROM python:3.12-trixie
RUN pip install fastapi uvicorn python-multipart polars matplotlib
COPY main.py .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

3. Frontend (HTML)

Crie web_app/frontend/index.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Titanic Analyzer</title>
    <style>
        body { font-family: sans-serif; display: flex; justify-content: center;
        padding-top: 50px; background-color: #f0f2f5; }
        .card { background: white; padding: 30px; border-radius: 10px;
        box-shadow: 0 4px 8px rgba(0,0,0,0.1); width: 400px; text-align: center; }
        button { background-color: #007bff; color: white; border: none;
        padding: 10px 20px; border-radius: 5px; cursor: pointer; }
```

```
        button:hover { background-color: #0056b3; }
    </style>
</head>
<body>
    <div class="card">
        <h2>Titanic Data Upload</h2>
        <p>Upload your <code>titanic.csv</code> file to see survival rates by class.</p>
        <form action="/api/analyze" method="post" enctype="multipart/form-data">
            <input type="file" name="file" accept=".csv" required>
            <br><br>
            <button type="submit">Generate Report</button>
        </form>
    </div>
</body>
</html>
```

4. Configuração Nginx

Crie `web_app/nginx.conf`:

```
events {}
http {
    server {
        listen 80;
        location / {
            root /usr/share/nginx/html;
            index index.html;
        }
        location /api/ {
            proxy_pass http://backend:8000/;
        }
    }
}
```

5. Executar a Aplicação

Crie `docker-compose-web.yml` na pasta raiz (`web_app`):

```yaml
services:
  backend:
    build: ./backend
  frontend:
    image: nginx:alpine
    ports:
      - "8080:80"
    volumes:
      - ./frontend:/usr/share/nginx/html
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - backend
```

Execute: `docker compose -f docker-compose-web.yml up --build` Vá a `http://localhost:8080` e carregue o seu `titanic.csv` gerado.

Parte 6: Desafio Opcional

Objetivo: Filtrar por Género.

1. **Modificar Frontend:** Adicione um menu *dropdown* ao `index.html` para selecionar "All" (Todos), "Male" (Masculino), ou "Female" (Feminino).
2. **Modificar Backend:** Aceite este novo campo de formulário. Use o filtro do Polars para subconjunto dos dados antes de calcular a taxa média de sobrevivência.

```python
# Example Polars Filter
if gender ≠ "All":
    df = df.filter(pl.col("Sex") == gender.lower())
```