Data Loading, Manipulation, and Visualization

Introdução Engenharia Informática

Mário Antunes

December 01, 2025

Exercises

Objective

In this laboratory, you will explore the famous **Titanic dataset**. Your goal is to analyze passenger data to understand who survived the disaster. You will simulate a real-world scenario where data comes in different formats (CSV, JSON, Excel), clean that data, visualize it, and build a web application to process it.

Part 0: Environment Setup

We will use **Docker** to create a consistent coding environment.

1. Create Project Structure

Run these commands in your terminal:

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

2. The Dockerfile

Create a file named `Dockerfile` inside `ex11` folder:

```
# 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. The Docker Compose File

Create `docker-compose.yml` inside `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. Launch and Generate Data

Run `docker compose up --build` in your terminal. Open `http://localhost:8888` (password: `titanic`).

Create a new notebook and download the titatic dataset from elearning. Place it in the data folder previously created.

Part 1: Data Loading

In the real world, you might receive passenger lists in Excel from HR, or JSON from a web API. Let's learn to load all of them.

```
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))
```

Part 2: Data Pre-processing

The Titanic dataset is notorious for having missing Age values and occasionally messy data. We will simulate "dirty" data and fix it.

Step 1: Identify Issues

We created the dataset with a missing value (`None`) in the Age column for passenger 6. Let's also verify if there are 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())
```

Step 2: Inject an Outlier

Let's manually add an error to simulate a data entry mistake (e.g., someone typing age 200 instead of 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()
```

Step 3: Fix the Data

1. **Missing Values:** Fill missing Age with the median age of passengers.
2. **Outliers:** Filter out unrealistic ages (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())
```

Part 3: Data Visualization

We will now generate plots to explicitly verify the Survivability Rate. Instead of just counting survivors, we want to see the percentage of people who survived in different groups.

```
# 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()
```

Analysis:

- **Left Plot:** Verifies if richer passengers (1st Class) had a higher chance of survival compared to 3rd Class.
- **Right Plot:** Verifies the “Women and children first” protocol by comparing Male vs Female survival rates.

Part 4: Exporting Results

We will save the Survival Rate by Class analysis to the shared volume, as this is the most critical insight for our report.

```
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}")
```

Part 5: Web Application

We will create a simple website where a user can upload the `titanic.csv` and get a generated plot of **Survival Rate by Class**.

1. Structure

Create the folder `web_app` inside `titanic_lab`. Inside it, create `backend` and `frontend`.

2. Backend (FastAPI + Polars)

Create `web_app/backend/main.py`. This API reads the CSV, calculates survival rates using **Polars**, and draws a plot.

```
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>
    """)
```

Create `web_app/backend/Dockerfile`:

```
FROM python:3.9-slim
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)

Create `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. Nginx Config

Create `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. Run the App

Create `docker-compose-web.yml` in the root folder:

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

Run: `docker compose -f docker-compose-web.yml up --build` Go to `http://localhost:8080` and upload your generated `titanic.csv`.

Part 6: Optional Challenge

Objective: Filter by Gender.

1. **Modify Frontend:** Add a dropdown menu to `index.html` to select “All”, “Male”, or “Female”.
2. **Modify Backend:** Accept this new form field. Use Polars `filter` to subset the data before calculating the mean survival rate.

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