

Lab 6: Working with the Eurostat API in Python

Course: Programming for Data Science
Professor: Dr. Carlos J. Costa

October 21, 2026

1 Objectives

By the end of this lab, you should be able to:

- Access the Eurostat API using Python.
- Explore and search the Eurostat dataset catalogue.
- Retrieve datasets and identify their dimensions and variables.
- Clean, filter, and prepare data for analysis.
- Create different types of visualizations to explore and communicate data.
- Interpret charts and identify patterns, trends, and differences between countries.
- Develop a reusable Python class that encapsulates Eurostat data retrieval and visualization.

2 Installation and Setup

Eurostat provides statistical information about European countries, including economic performance, population, employment, energy, transport, and the environment.

In this lab, you will use Python to retrieve and analyze data directly from Eurostat.

Install the required libraries:

```
1 !pip install eurostat pandas matplotlib seaborn
```

Import the libraries:

```
1 import eurostat
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

Set a visualization style:

```
1 sns.set_theme(style="whitegrid")
```

Task 1. Execute the code and confirm that all libraries are available.

3 Exploring Available Datasets

Eurostat organizes its data into datasets. Each dataset has a unique code and a description.

Retrieve the dataset catalogue:

```

1 toc_df = eurostat.get_toc_df()
2
3 print(toc_df.head())
4 print(toc_df.columns)
5 print(toc_df.shape)

```

The `head()` method displays the first five rows. The `columns` attribute identifies the available fields, and `shape` returns the number of rows and columns.

Display additional information:

```

1 toc_df.info()

```

Task 2. Answer the following questions:

1. How many rows and columns does the catalogue contain?
2. What information is available for each dataset?
3. Why is a dataset code useful when working with an API?

4 Searching for Relevant Datasets

Suppose we want to investigate fishing fleets in European countries.

Search the catalogue using the keyword `fleet`:

```

1 fleet_datasets = eurostat.subset_toc_df(
2     toc_df, "fleet"
3 )
4
5 print(fleet_datasets)

```

You can also search for other topics:

```

1 gdp_datasets = eurostat.subset_toc_df(
2     toc_df, "GDP"
3 )
4
5 population_datasets = eurostat.subset_toc_df(
6     toc_df, "population"
7 )
8
9 print(gdp_datasets.head())
10 print(population_datasets.head())

```

Search results may include several datasets with different levels of detail or different statistical measures.

Task 3. Search for datasets related to:

- Employment
- Renewable energy
- Tourism

Select one dataset and record its code and description. Explain why the dataset could be useful for an economic or management analysis.

5 Inspecting Dataset Parameters

Before downloading a dataset, investigate its available dimensions and parameters.

For example, use the fishing fleet dataset TAG00083, if it is available in the current catalogue:

```
1 dataset_code = "TAG00083"
2
3 parameters = eurostat.get_pars(dataset_code)
4
5 print(parameters)
```

The parameters may identify dimensions such as country, year, fleet category, or statistical unit. These dimensions determine which observations can be selected for analysis.

Task 4. Answer the following questions:

1. Identify the geographical and temporal dimensions.
2. Identify any additional classification variables.
3. Explain how these dimensions could be used to filter the dataset.

If the example dataset is no longer available, select a valid dataset from your search results and replace `dataset_code` accordingly.

6 Retrieving and Exploring Data

Download the selected dataset:

```
1 raw_data = eurostat.get_data_df(dataset_code)
2
3 print(raw_data.head())
4 print(raw_data.shape)
5 print(raw_data.columns)
6 print(raw_data.index)
```

Inspect its structure:

```
1 raw_data.info()
```

The structure of the returned DataFrame depends on the dataset and the library's representation. Some datasets contain dimensions in columns and observations in rows; others use a wide format, with time periods represented by columns.

Do not assume that every Eurostat dataset has the same structure. Always inspect the returned data before cleaning it.

If the data uses a wide format, examine its transposed representation:

```
1 print(raw_data.T.head(10))
```

Transposing a DataFrame exchanges its rows and columns. This can help reveal the relationship between countries, years, and statistical values.

Task 5. Describe the structure of your dataset. Identify where the country identifiers, time periods, and statistical values are stored.

7 Preparing and Cleaning Data

Before visualization, check whether the data contains missing values, unexpected data types, or unnecessary columns.

First, examine missing values:

```
1 print(raw_data.isna().sum())
```

Inspect the data types:

```
1 print(raw_data.dtypes)
```

If a numerical variable is stored as text, it may need conversion.

```
1 # Replace "value_column" with the actual column name.
2
3 # raw_data["value_column"] = pd.to_numeric(
4 #     raw_data["value_column"],
5 #     errors="coerce"
6 # )
```

The argument `errors="coerce"` converts values that cannot be interpreted as numbers into missing values (NaN).

If the dataset contains missing values, investigate their meaning before deciding how to handle them. A missing observation is not necessarily equivalent to zero.

For datasets that return time periods as columns, inspect the transposed data to identify any metadata or header rows:

```
1 transposed = raw_data.T
2
3 print(transposed.head(10))
```

In the original fishing fleet example, the following transformation may be appropriate if inspection confirms that the fifth row contains the headers and subsequent rows contain observations:

```
1 df = transposed.iloc[5:].copy()
2
3 df.columns = transposed.iloc[4].tolist()
4
5 print(df.head())
```

This is a dataset-specific transformation, not a universal Eurostat cleaning procedure. Check the resulting column names and values before continuing.

Task 6. Prepare a clean DataFrame suitable for analysis. Document the cleaning decisions you made and explain why they were necessary.

8 Data Visualization

Data visualization is an essential part of exploratory data analysis. Different charts answer different questions, and the choice of visualization should depend on the data and the research objective.

In this section, you will create line charts, bar charts, histograms, and scatterplots using Matplotlib and Seaborn.

8.1 Line Chart: Analyzing Trends Over Time

A line chart is useful for analyzing how a variable evolves over time.

Suppose the cleaned dataset contains years as the index and country codes as columns. We can compare Portugal, Spain, and France.

```

1 countries = ["PT", "ES", "FR"]
2
3 available_countries = [
4     country for country in countries
5     if country in df.columns
6 ]
7
8 if available_countries:
9     df[available_countries].plot(
10         kind="line",
11         marker="o",
12         figsize=(10, 5)
13     )
14
15 plt.title("Fishing Fleet Comparison")
16 plt.xlabel("Year")
17 plt.ylabel("Number of vessels")
18 plt.legend(title="Country")
19 plt.tight_layout()
20 plt.show()

```

If the dataset measures something other than the number of vessels, replace the vertical-axis label with the correct unit.

Interpretation questions:

1. Which country has the highest values?
2. Is there an increasing or decreasing trend?
3. Do the countries follow similar patterns?
4. Are there periods with substantial changes?

8.2 Bar Chart: Comparing Countries

A bar chart is useful for comparing values between countries at a particular point in time.

If your cleaned data contains one row per country and numerical values in a column named `value`, you can create a bar chart as follows.

```

1 # Replace "country" and "value" with
2 # the actual column names in your dataset.
3
4 # selected_year = "2024"
5 # year_data = df[df["year"] == selected_year]
6
7 # year_data = year_data.sort_values(
8 #     "value", ascending=False
9 # )
10
11 # plt.figure(figsize=(10, 5))
12 # sns.barplot(
13 #     data=year_data,
14 #     x="country",
15 #     y="value"
16 # )
17
18 # plt.title("Comparison Between Countries")
19 # plt.xlabel("Country")

```

```

20 # plt.ylabel("Value")
21 # plt.xticks(rotation=45)
22 # plt.tight_layout()
23 # plt.show()

```

The code is a template. Adapt the column names and year to your selected dataset.

Interpretation questions:

1. Which countries have the highest and lowest values?
2. Are the differences between countries substantial?
3. What additional information would you need before explaining these differences?

8.3 Histogram: Examining the Distribution

A histogram shows how numerical observations are distributed.

For example, if you have a numerical column named `value`:

```

1 # Replace "value" with the actual numerical column.
2
3 # values = pd.to_numeric(
4 #     df["value"], errors="coerce"
5 # ).dropna()
6
7 # plt.figure(figsize=(8, 5))
8 # sns.histplot(values, bins=15, kde=True)
9
10 # plt.title("Distribution of Observations")
11 # plt.xlabel("Value")
12 # plt.ylabel("Frequency")
13 # plt.tight_layout()
14 # plt.show()

```

The histogram helps identify concentration, dispersion, skewness, and potentially unusual observations.

Interpretation questions:

1. Where are most observations concentrated?
2. Does the distribution appear symmetric or skewed?
3. Are there observations that appear unusually high or low?

8.4 Scatterplot: Investigating Relationships

A scatterplot helps explore the relationship between two numerical variables.

For example, suppose a dataset contains GDP per capita and another economic indicator for each country.

```

1 # Replace the example column names with
2 # the actual variables in your dataset.
3
4 # plt.figure(figsize=(8, 5))
5 # sns.scatterplot(
6 #     data=df,
7 #     x="gdp_per_capita",
8 #     y="value"
9 # )
10

```

```

11 # plt.title(
12 #     "Relationship Between GDP per Capita and Value"
13 # )
14 # plt.xlabel("GDP per capita")
15 # plt.ylabel("Value")
16 # plt.tight_layout()
17 # plt.show()

```

Each point represents an observation. A visible upward or downward pattern may suggest an association between the variables.

A scatterplot alone does not establish causality.

Interpretation questions:

1. Does the relationship appear positive, negative, or unclear?
2. Are there potential outliers?
3. Could other variables explain the observed pattern?

8.5 Improving the Quality of Charts

A useful visualization should communicate information clearly.

For every chart, check that it includes:

- A descriptive title.
- Meaningful axis labels.
- Appropriate units.
- A legend when necessary.
- Readable labels and an appropriate figure size.
- A chart type suitable for the analytical question.

Avoid unnecessary decoration and misleading axis scales.

Task 7. Produce at least three different types of charts using your selected dataset. For each chart, write two or three sentences explaining what it reveals.

9 Encapsulating the Workflow in a Python Class

A Python class allows us to organize related operations into a reusable component.

We will create a class named `EuroStatExplorer` to search the catalogue, inspect parameters, retrieve data, and generate visualizations.

The class deliberately separates data retrieval from data cleaning because different Eurostat datasets can have different structures.

```

1  import eurostat
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5
6
7  class EuroStatExplorer:
8      """Explore, retrieve, and visualize Eurostat data."""
9
10     def __init__(self):
11         """Load the Eurostat dataset catalogue."""

```

```

12     self.toc_df = eurostat.get_toc_df()
13     self.raw_data = None
14
15     def search_dataset(self, keyword):
16         """Search for datasets matching a keyword."""
17         return eurostat.subset_toc_df(
18             self.toc_df, keyword
19         )
20
21     def get_parameters(self, dataset_code):
22         """Retrieve the dimensions of a dataset."""
23         return eurostat.get_pars(dataset_code)
24
25     def load_dataset(self, dataset_code):
26         """Retrieve a dataset from Eurostat."""
27         self.raw_data = eurostat.get_data_df(
28             dataset_code
29         )
30         return self.raw_data
31
32     def inspect_dataset(self):
33         """Display basic information about loaded data."""
34         if self.raw_data is None:
35             raise ValueError(
36                 "Load a dataset before inspecting it."
37             )
38
39         print("Shape:", self.raw_data.shape)
40         print("Columns:", self.raw_data.columns.tolist())
41         print("Data types:")
42         print(self.raw_data.dtypes)
43         print("First observations:")
44         print(self.raw_data.head())
45
46     def plot_line(self, data, title, xlabel, ylabel):
47         """Create a line chart from a prepared DataFrame."""
48         ax = data.plot(
49             kind="line",
50             marker="o",
51             figsize=(10, 5)
52         )
53
54         ax.set_title(title)
55         ax.set_xlabel(xlabel)
56         ax.set_ylabel(ylabel)
57
58         plt.tight_layout()
59         plt.show()
60
61     def plot_bar(
62         self, data, x, y, title, xlabel, ylabel
63     ):
64         """Create a bar chart from a prepared DataFrame."""
65         plt.figure(figsize=(10, 5))
66
67         sns.barplot(data=data, x=x, y=y)
68
69         plt.title(title)

```

```

70         plt.xlabel(xlabel)
71         plt.ylabel(ylabel)
72         plt.xticks(rotation=45)
73
74         plt.tight_layout()
75         plt.show()
76
77     def plot_scatter(
78         self, data, x, y, title, xlabel, ylabel
79     ):
80         """Create a scatterplot from a prepared DataFrame."""
81         plt.figure(figsize=(8, 5))
82
83         sns.scatterplot(data=data, x=x, y=y)
84
85         plt.title(title)
86         plt.xlabel(xlabel)
87         plt.ylabel(ylabel)
88
89         plt.tight_layout()
90         plt.show()
91
92     def plot_histogram(
93         self, data, column, title, xlabel
94     ):
95         """Create a histogram of a numerical variable."""
96         values = pd.to_numeric(
97             data[column], errors="coerce"
98         ).dropna()
99
100        plt.figure(figsize=(8, 5))
101
102        sns.histplot(values, bins=15, kde=True)
103
104        plt.title(title)
105        plt.xlabel(xlabel)
106        plt.ylabel("Frequency")
107
108        plt.tight_layout()
109        plt.show()

```

Notice that the visualization methods receive a prepared DataFrame and the names of the relevant columns. This makes the class reusable across datasets with different structures.

10 Testing the Class

Create an instance of the class:

```
1 es = EuroStatExplorer()
```

Search for fishing fleet datasets:

```
1 es.search_dataset("fleet")
```

Inspect the parameters of a selected dataset:

```
1 es.get_parameters("TAG00083")
```

Load the dataset and inspect its structure:

```

1 es.load_dataset("TAG00083")
2
3 es.inspect_dataset()

```

Once the data has been cleaned and its structure is understood, prepare a DataFrame appropriate for the selected visualization.

For example, if `clean_df` contains a numerical variable named `value` and a country variable named `country`, you can create a bar chart:

```

1 es.plot_bar(
2     data=clean_df,
3     x="country",
4     y="value",
5     title="Fishing Fleet by Country",
6     xlabel="Country",
7     ylabel="Number of vessels"
8 )

```

Replace `clean_df` and its column names with those from your own cleaning step. This example assumes that each row represents an observation and that the columns contain the specified variables.

Task 8. Test the class with another dataset. Search for it, inspect its parameters, retrieve the data, prepare the DataFrame, and create at least two visualizations using the class methods.

11 Discussion

Answer the following questions:

1. How does Eurostat organize and identify its datasets?
2. What are the advantages of retrieving data directly through an API instead of manually downloading files?
3. Why can cleaning procedures differ between datasets?
4. What information can a line chart reveal that a bar chart may not show as clearly?
5. When is a scatterplot more appropriate than a histogram?
6. How can inappropriate axis scales or labels mislead the reader?
7. What are the advantages of encapsulating data retrieval and visualization in a Python class?
8. How could the class be extended to support filtering by country, year, or statistical category?

12 Practical Exercise

Choose a Eurostat dataset related to GDP, population, employment, tourism, energy, or another topic of interest.

Complete the following tasks:

1. Search for and identify a suitable dataset.
2. Record its code, description, and available parameters.

3. Retrieve the data programmatically.
4. Inspect its structure and document the necessary cleaning operations.
5. Create a line chart to show a trend over time, if the dataset has a time dimension.
6. Create a bar chart to compare at least five countries, if appropriate.
7. Create a histogram or scatterplot to investigate a distribution or relationship between numerical variables.
8. Explain the main findings revealed by the charts.
9. Use the `EuroStatExplorer` class to reproduce at least two of your visualizations.

Deliverables

Submit a Python notebook containing:

- The code used to retrieve and prepare the data.
- The dataset code and a brief description of its contents.
- At least three well-labelled visualizations, using different chart types where appropriate.
- A short interpretation of the results.
- The implementation and a demonstration of the `EuroStatExplorer` class.

Important: The objective is not merely to produce charts. You must explain what each visualization communicates, why the chart type is appropriate, and what conclusions can reasonably be drawn from the data.