

Import & Export Data (CSV, Excel, JSON)

◆ Reminder

- Yesterday: we created DataFrames manually (`dict` → `DataFrame`).
 - Today: data usually comes from **files** (CSV, Excel, JSON) or **APIs**.
-

◆ Data Formats

Notes

- **CSV** → Most common, simple, readable.
- **Excel** → Business world, multiple sheets.
- **JSON** → API/web services, nested structures.

Examples

students.csv

```
Name, Age, City  
Alice, 25, Nairobi  
Bob, 30, Mombasa
```

students.xlsx → same as above, saved as Excel.

students.json

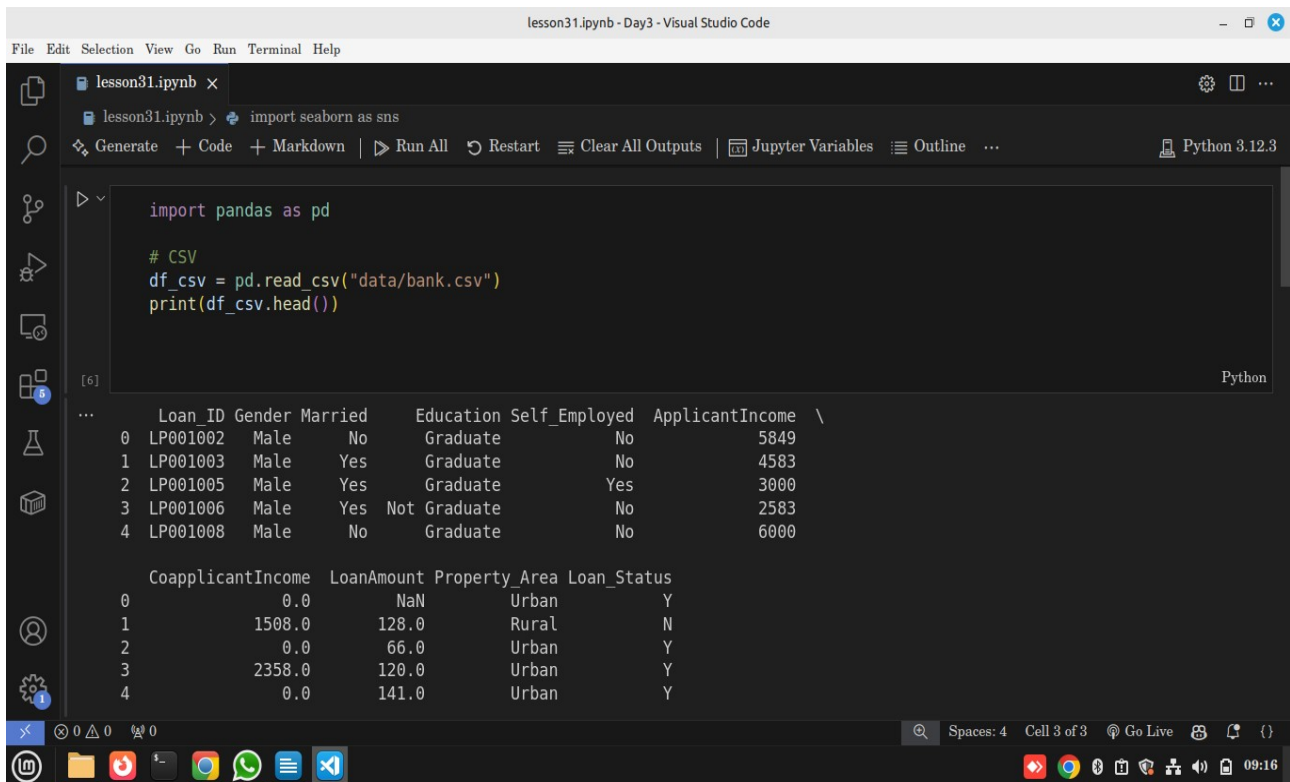
```
[  
  {"Name": "Alice", "Age": 25, "City": "Nairobi"},  
  {"Name": "Bob", "Age": 30, "City": "Mombasa"}  
]
```

Use in Data Science:

- CSV → Kaggle datasets.
 - Excel → corporate records.
 - JSON → web APIs (Twitter, Weather).
-

◆ Importing Data

Practical – Reading Different Formats



The screenshot shows a Jupyter Notebook interface within Visual Studio Code. The notebook is titled 'lesson31.ipynb - Day3 - Visual Studio Code'. The code in the notebook is as follows:

```
import pandas as pd

# CSV
df_csv = pd.read_csv("data/bank.csv")
print(df_csv.head())
```

The output of the code is displayed below the code cell. It shows the first five rows of the 'bank.csv' file. The columns are: Loan_ID, Gender, Married, Education, Self_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Property_Area, and Loan_Status.

	Loan_ID	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Property_Area	Loan_Status
0	LP001002	Male	No	Graduate	No	5849	0.0	NaN	Urban	Y
1	LP001003	Male	Yes	Graduate	No	4583	1508.0	128.0	Rural	N
2	LP001005	Male	Yes	Graduate	Yes	3000	0.0	66.0	Urban	Y
3	LP001006	Male	Yes	Not Graduate	No	2583	2358.0	120.0	Urban	Y
4	LP001008	Male	No	Graduate	No	6000	0.0	141.0	Urban	Y

- `pd.read_csv()` → loads CSV into DataFrame.
- `pd.read_excel()` → needs `openpyxl` installed.
- `pd.read_json()` → parses JSON into DataFrame.

Use in Data Science

These are **entry points** for every project — from raw file → DataFrame → analysis.

◆ Exporting Data

Practical – Saving Data

```
# Save to CSV
df_csv.to_csv("output.csv", index=False)

# Save to Excel
df_csv.to_excel("output.xlsx", index=False)

# Save to JSON
df_csv.to_json("output.json", orient="records")
```

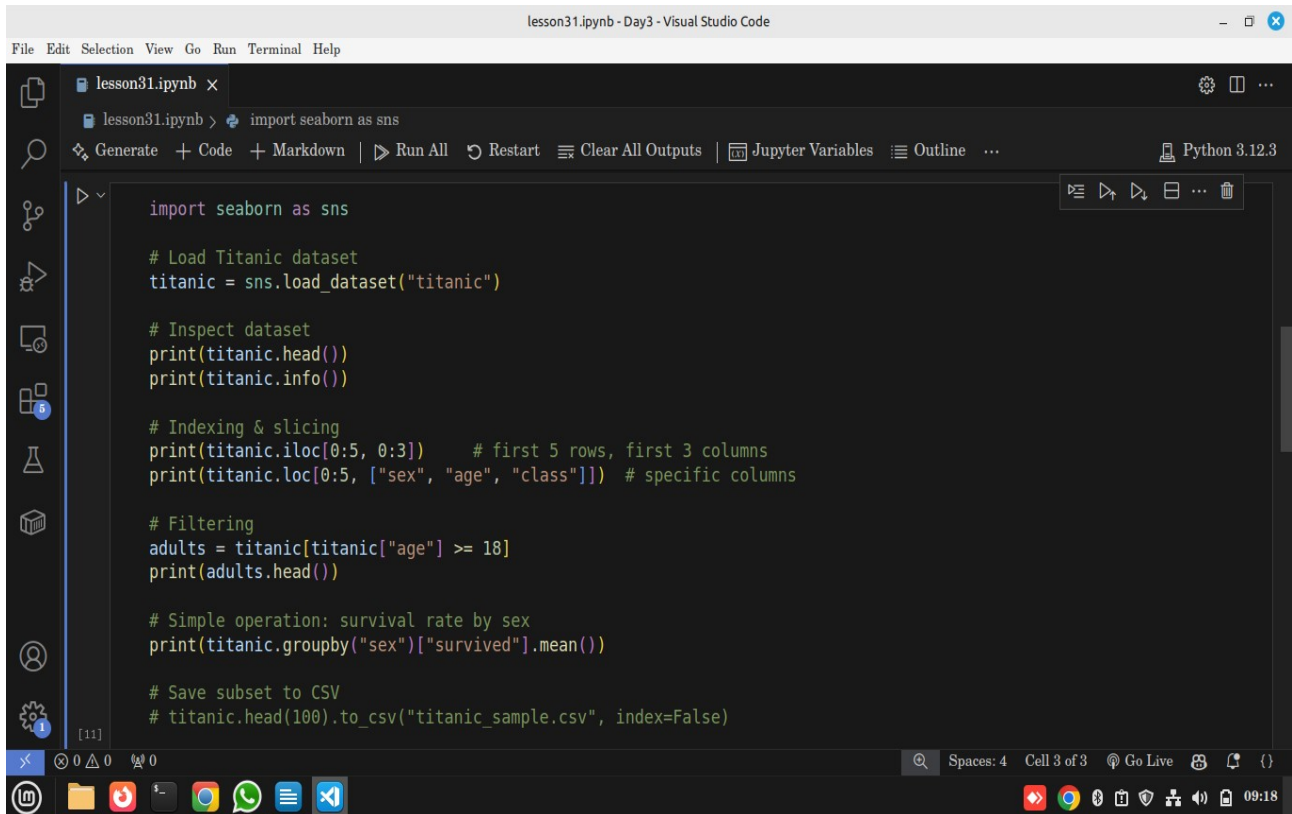
Explanation

- `index=False` → avoids extra column.
- `orient="records"` → JSON-friendly format.

Use in Data Science

Exporting lets you share cleaned/processed data with **colleagues, dashboards, or ML pipelines**.

◆ Hands-On: Titanic Dataset



```
import seaborn as sns

# Load Titanic dataset
titanic = sns.load_dataset("titanic")

# Inspect dataset
print(titanic.head())
print(titanic.info())

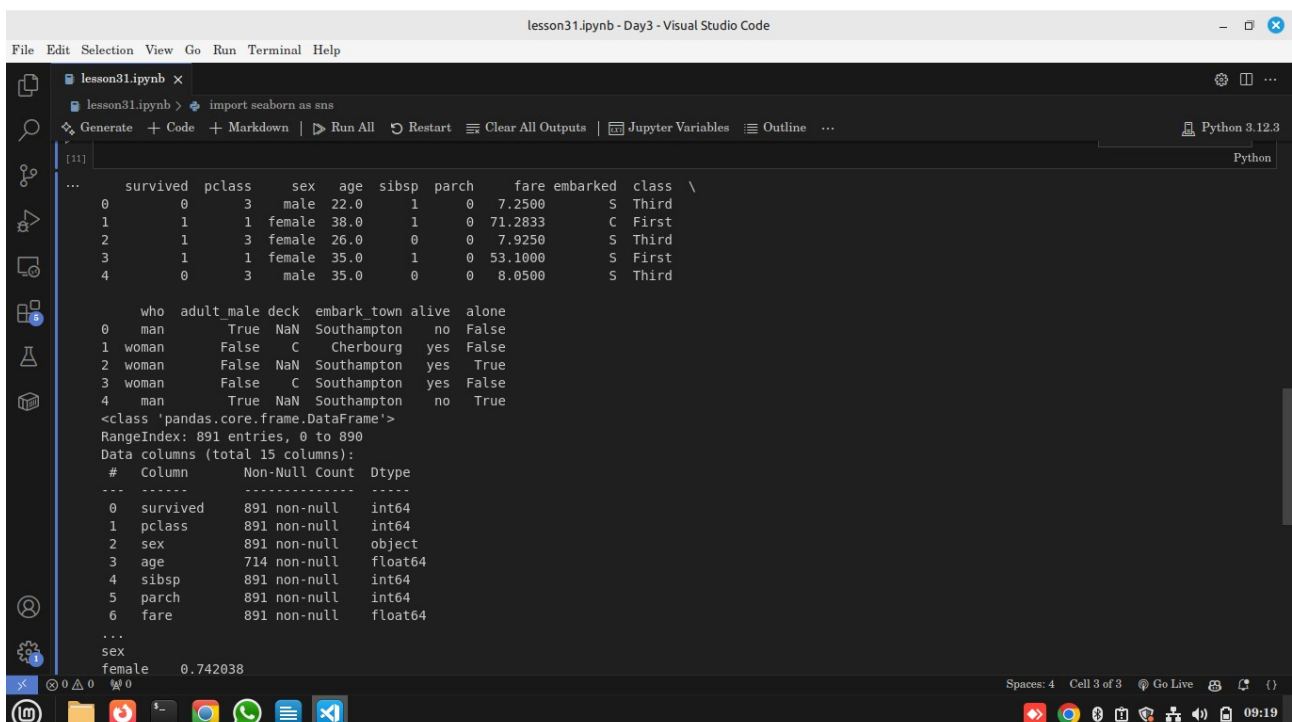
# Indexing & slicing
print(titanic.iloc[0:5, 0:3]) # first 5 rows, first 3 columns
print(titanic.loc[0:5, ["sex", "age", "class"]]) # specific columns

# Filtering
adults = titanic[titanic["age"] >= 18]
print(adults.head())

# Simple operation: survival rate by sex
print(titanic.groupby("sex")["survived"].mean())

# Save subset to CSV
# titanic.head(100).to_csv("titanic_sample.csv", index=False)
```

Sample Output



```
[11]:
...   survived  pclass    sex  age  sibsp  parch  fare embarked  class \
0         0      3   male  22.0    1     0   7.2500    S   Third
1         1      1  female  38.0    1     0  71.2833    C   First
2         1      3  female  26.0    0     0   7.9250    S   Third
3         1      1  female  35.0    1     0  53.1000    S   First
4         0      3   male  35.0    0     0   8.0500    S   Third

   who  adult_male  deck  embark_town  alive  alone
0  man         True   NaN  Southampton    no   False
1 woman        False    C  Cherbourg    yes   False
2 woman        False   NaN  Southampton    yes   True
3 woman        False    C  Southampton    yes   False
4  man         True   NaN  Southampton    no    True
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
#   Column      Non-Null Count  Dtype
---  -
0   survived    891 non-null      int64
1   pclass      891 non-null      int64
2   sex         891 non-null      object
3   age         714 non-null      float64
4   sibsp       891 non-null      int64
5   parch       891 non-null      int64
6   fare        891 non-null      float64
...
sex
female    0.742038
```

Explanation

- `iloc` → integer-based indexing.
- `loc` → label-based indexing.
- Filtering (e.g., $\text{age} \geq 18$) → data wrangling basics.
- `.groupby("sex")` → real analysis: shows women had higher survival rates.
- `.to_csv()` → saves results for later.

Use in Data Science

This shows how **imported data** can immediately be **inspected, filtered, and analyzed** — bridging file formats with real analysis.



Reflection

- Which format is most common? (**CSV**).
- Why use JSON? (**APIs / real-time data**).
- Why is Excel risky? (**Formatting errors, multiple sheets**).
- How does indexing/slicing help? (**Lets us zoom into relevant data quickly**).