

Lecture VII - Pandas and AI

Programming with Python

Dr. Tobias Vlček

Kühne Logistics University Hamburg - Fall 2025

Quick Recap of the last Lecture

What is NumPy?

- NumPy is a package for scientific computing in Python
- Provides multi-dimensional arrays and matrices
- Much faster than Python lists for numerical operations
- Operations are implemented in C and C++

...

Tip

NumPy arrays are stored in contiguous memory blocks, making operations very efficient.

Creating Arrays

- Core data structure is the `ndarray`
- Can create arrays from lists, tuples, or other data structures
- Special functions like:
 - `np.zeros()` for arrays of zeros
 - `np.random.rand()` for random values
 - `np.arange()` for evenly spaced values
 - `np.linspace()` for linearly spaced values

Working with Arrays

- Support for multi-dimensional operations
- Common operations:
 - Element-wise arithmetic (`+`, `-`, `*`, `/`)
 - Array indexing and slicing
 - Shape manipulation (`reshape`, `flatten`)
 - Sorting and transposing

...

💡 Tip

NumPy operations are vectorized, meaning they operate on entire arrays at once rather than element by element.

NumPy in Action I

Task: Complete the following task:

```
# TODO: Create an array with 10 evenly spaced numbers over the interval
from 0 to 73.
```

```
import numpy as np
# YOUR CODE HERE
```

...

i Note

Note, that you can always use the `help()` function to get more information about a function. But be sure to import the package first, otherwise you will get an error. To quit the help page, press `q`.

NumPy in Action II

Task: Complete the following task:

```
# TODO: Take the following 3x3 array and reduce it to a 1D array.
```

```
import numpy as np
array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# YOUR CODE HERE
```

Pandas Basics

What is Pandas?

- Pandas is a data manipulation and analysis library
- It provides data structures like DataFrames and Series
- Tools for data cleaning, analysis, and visualization
- It can also be used to work with Excel files!

How to install Pandas

- In the last lecture, we have installed it with `uv install pandas`
- Now, import the package `import pandas as pd`

...

Note

You can also use a different abbreviation, but `pd` is the most common one.

Creating DataFrames

- DataFrames behave quite similar to Numpy arrays
- But they have row and column labels

...

```
import pandas as pd
df = pd.DataFrame({ # DataFrame is created from a dictionary
    "Name": ["Tobias", "Robin", "Nils", "Nikolai"],
    "Kids": [2, 1, 0, 0],
    "City": ["Oststeinbek", "Oststeinbek", "Hamburg", "Lübeck"],
    "Salary": [3000, 3200, 4000, 2500]}) # print(df)
```

	Name	Kids	City	Salary
0	Tobias	2	Oststeinbek	3000
1	Robin	1	Oststeinbek	3200
2	Nils	0	Hamburg	4000
3	Nikolai	0	Lübeck	2500

Reading from CSV Files

```
df = pd.read_csv("supplementary/lec_08/employees.csv") # Reads the CSV file
print(df)
```

	Name	Age	Department	Position	Salary
0	Alice	30	HR	Manager	50000
1	Bob	25	IT	Developer	60000
2	Charlie	28	Finance	Analyst	55000
3	David	35	Marketing	Executive	52000
4	Eve	32	Sales	Representative	48000
5	Frank	29	IT	Developer	61000
6	Grace	31	HR	Assistant	45000
7	Hank	27	Finance	Analyst	53000
8	Ivy	33	Marketing	Manager	58000
9	Jack	26	Sales	Representative	47000
10	Kara	34	IT	Developer	62000
11	Leo	30	HR	Manager	51000
12	Mona	28	Finance	Analyst	54000
13	Nina	35	Marketing	Executive	53000
14	Oscar	32	Sales	Representative	49000
15	Paul	29	IT	Developer	63000
16	Quinn	31	HR	Assistant	46000
17	Rita	27	Finance	Analyst	52000
18	Sam	33	Marketing	Manager	59000
19	Tina	26	Sales	Representative	48000

20	Uma	34	IT	Developer	64000
21	Vince	30	HR	Manager	52000
22	Walt	28	Finance	Analyst	55000
23	Xena	35	Marketing	Executive	54000
24	Yara	32	Sales	Representative	50000
25	Zane	29	IT	Developer	65000
26	Anna	31	HR	Assistant	47000
27	Ben	27	Finance	Analyst	53000
28	Cathy	33	Marketing	Manager	60000
29	Dylan	26	Sales	Representative	49000
30	Ella	34	IT	Developer	66000
31	Finn	30	HR	Manager	53000
32	Gina	28	Finance	Analyst	56000
33	Hugo	35	Marketing	Executive	55000
34	Iris	32	Sales	Representative	51000
35	Jake	29	IT	Developer	67000
36	Kyla	31	HR	Assistant	48000
37	Liam	27	Finance	Analyst	54000
38	Mia	33	Marketing	Manager	61000
39	Noah	26	Sales	Representative	50000
40	Olive	34	IT	Developer	68000
41	Pete	30	HR	Manager	54000
42	Quincy	28	Finance	Analyst	57000
43	Rose	35	Marketing	Executive	56000
44	Steve	32	Sales	Representative	52000
45	Tara	29	IT	Developer	69000
46	Umar	31	HR	Assistant	49000
47	Vera	27	Finance	Analyst	55000
48	Will	33	Marketing	Manager	62000
49	Zara	26	Sales	Representative	51000

Basic Operations

- Use the `df.head()` method to display the first 5 rows
- Use the `df.tail()` method to display the last 5 rows

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
print(df.tail())
```

	Name	Age	Department	Position	Salary
45	Tara	29	IT	Developer	69000
46	Umar	31	HR	Assistant	49000
47	Vera	27	Finance	Analyst	55000
48	Will	33	Marketing	Manager	62000
49	Zara	26	Sales	Representative	51000

Information about the DataFrame

- Use `df.info()` to display information about a DataFrame

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Name        50 non-null    object
1   Age         50 non-null    int64
2   Department  50 non-null    object
3   Position    50 non-null    object
4   Salary      50 non-null    int64
dtypes: int64(2), object(3)
memory usage: 2.1+ KB
None
```

Statistics about a DataFrame

- Use `df.describe()` to display summary statistics
- Use the `df.index` attribute to access the index

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
print(df.describe())
```

	Age	Salary
count	50.000000	50.000000
mean	30.320000	54980.000000
std	2.958488	6175.957333
min	25.000000	45000.000000
25%	28.000000	50250.000000
50%	30.000000	54000.000000
75%	33.000000	59750.000000
max	35.000000	69000.000000

Filtering DataFrames

- Use `df['column_name']` to access a column
- Use the `df[df['column'] > value]` method to filter

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
df_high_salary = df[df['Salary'] >= 67000]
print(df_high_salary)
print(df_high_salary.iloc[2]["Name"]) #Access the third row and the "Name"
column
print(df_high_salary.loc[40]["Name"]) #Access the label 40 and the "Name"
column
```

	Name	Age	Department	Position	Salary
35	Jake	29	IT	Developer	67000
40	Olive	34	IT	Developer	68000
45	Tara	29	IT	Developer	69000
	Tara				
	Olive				

Filtering in Action

Task: Complete the following task:

```
# TODO: Load the employees.csv located in the git repository into a
DataFrame
# First, filter the DataFrame for employees with a manager position
# Then, print the average salary of the remaining employees
# Finally, print the name of the employee with the lowest salary
```

...

i Note

Note, that we can use the `mean()` method on the `Salary` column, as it is a numeric column. In addition, we can use the `min()` method on the `Salary` column to find the lowest salary.

Grouping DataFrames

Grouping

- Grouping is a powerful feature of Pandas
- Groups data by one or more columns
- And then perform operations
- Syntax is `df.groupby('column').method()`

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
df.groupby(['Position']).sum() # Sum per position
```

	Name	Age	Department	Salary
Position				
Analyst	CharlieHankMonaR- itaWaltBenGina- LiamQuincyVera	275	FinanceFinanceFi- nanceFinanceFinance- FinanceFina...	544000
Assistant	GraceQuinnAnnaKy- laUmar	155	HRHRHRHRHR	235000

		Name	Age	Department	Salary
Position					
Developer	Bob	Frank	Kara-	IT	645000
	Paul	Uma	Zane		
	Ella-	Jake	Olive		
Executive	David	Nina	Xe-	Marketing	270000
	na	Hugo	Rose		
Manager	Alice	Ivy	Leo	HR	560000
	Sam	Vince-	Cathy		
	Finn	Mia	Pete		
Representative	Eve	Jack	Oscar	Sales	495000
	Tina-	Yara	Dylan		
	Iris	Noah-	Steve		

Grouping Numeric Columns

- To prevent errors, we can select numeric columns first
- Afterwards, perform the operation on the selected columns
- Helps to avoid errors when grouping by non-numeric columns
- Or drop columns by `df.drop(columns=["column"])`

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
numeric_cols = df.select_dtypes(include=['number']).columns
print(df.groupby("Position")[numeric_cols].sum())
```

Position	Age	Salary
Analyst	275	544000
Assistant	155	235000
Developer	306	645000
Executive	175	270000
Manager	315	560000
Representative	290	495000

Grouping by Multiple Columns

- Group by multiple columns `['column1', 'column2']`
- You can use lists or tuples to specify multiple columns

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
df = df.drop(columns=["Name"])
```

```
# Max per position and department
df.groupby(['Position', 'Department']).max()
```

		Age	Salary
Position	Department		
Analyst	Finance	28	57000
Assistant	HR	31	49000
Developer	IT	34	69000
Executive	Marketing	35	56000
Manager	HR	30	54000
	Marketing	33	62000
Representative	Sales	32	52000

Grouping with Aggregations

- We can use different aggregation functions:
 - `sum()`: sum of the values
 - `mean()`: mean of the values
 - `max()`: maximum of the values
 - `min()`: minimum of the values
 - `count()`: count of the values

Pandas in Action

Task: Complete the following task:

```
# TODO: Load the employees.csv again into a DataFrame
# First, group by the "Position" column and count the employees per
position
# Then, group by the "Department" column and calculate the mean of all
other columns per department
df = pd.read_csv("supplementary/lec_08/employees.csv")
# Your code here
```

Combining DataFrames

Concatenating DataFrames

- `pd.concat()` to concatenate along shared columns

```
df1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
df2 = pd.DataFrame({"A": [7, 8, 9], "B": [10, 11, 12]})
df = pd.concat([df1, df2])
print(df)
```


	A	B
0	1	4
1	2	5
2	3	6
0	7	10
1	8	11
2	9	12

Joining DataFrames

- Use `pd.join()` to join DataFrames along columns
- Joining is done on the index by default!

```
df1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]}, index=['x', 'y', 'z'])
df2 = pd.DataFrame({"C": [7, 8, 9], "D": [10, 11, 12]}, index=['z', 'y', 'w'])
df = df1.join(df2)
print(df)
```

	A	B	C	D
x	1	4	NaN	NaN
y	2	5	8.0	11.0
z	3	6	7.0	10.0

Merging DataFrames on Columns

- `pd.merge(df_name, on='column', how='type')`
- merge DataFrames along shared columns
- `how` specifies the type of merge
 - `inner`: rows with matching keys in both DataFrames
 - `outer`: rows from both are kept, missing values are filled
 - `left`: rows from the left are kept, missing values are filled
 - `right`: rows from right are kept, missing values are filled

Outer Merge

```
df3 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
df4 = pd.DataFrame({"A": [2, 3, 4], "C": [7, 8, 9]})
df_merged = df3.merge(df4, on="A", how="outer")
print(df_merged)
```

	A	B	C
0	1	4.0	NaN
1	2	5.0	7.0
2	3	6.0	8.0
3	4	NaN	9.0

Merging in Action

Task: Complete the following task:

```
df1 = pd.DataFrame({
    "Name": ["John", "Alice", "Bob", "Carol"],
    "Department": ["Sales", "IT", "HR", "Sales"],
    "Salary": [50000, 60000, 55000, 52000]})
df2 = pd.DataFrame({
    "Name": ["Alice", "Bob", "Dave", "Eve"],
    "Position": ["Developer", "Manager", "Analyst", "Developer"],
    "Years": [5, 8, 3, 4]})

# TODO: Merge the two DataFrames on the "Name" column
# Try different types of merges (inner, outer, left, right)
# Observe and describe the differences in the results
```

Working with Excel Files

Reading Excel Files

- Read using the `pd.read_excel(file_path)` function
- Write using the `df.to_excel(file_path)` method

...

```
import pandas as pd
df = pd.read_csv("supplementary/lec_08/employees.csv")
df.to_excel("supplementary/lec_08/employees.xlsx", index=False)
```

...

Note

Note, that you likely need to install the `openpyxl` package to be able to write Excel files, as it handles the file format.

Advanced Excel file handling

We can also specify the sheet name when reading and writing

```
# Writes to the Employees sheet and does not include row indices
df.to_excel("supplementary/lec_08/employees.xlsx", sheet_name="Employees",
index=False)
```

...

```
# Reads from the Employees sheet
df = pd.read_excel("supplementary/lec_08/employees.xlsx",
sheet_name="Employees")
print(df.head())
```

	Name	Age	Department	Position	Salary
0	Alice	30	HR	Manager	50000
1	Bob	25	IT	Developer	60000
2	Charlie	28	Finance	Analyst	55000
3	David	35	Marketing	Executive	52000
4	Eve	32	Sales	Representative	48000

Excel in Action

Task: Complete the following task:

```
# TODO: Load the temperatures.xlsx file into a DataFrame
# Look at the first few rows of the DataFrame
# Then, print the average temperature per city
```

Melting DataFrames

Melting

- Sometimes, you want to transform a DataFrame
- Instead of wide format, you want long format
- This is useful for certain types of visualizations
- And when working with time series data

...

Question: Anybody ever heard of the terms?

Wide Format

For example, the following DataFrame is in wide format:

	Date	Hamburg	Los_Angeles	Tokyo
0	2024-03-01	12.0	18.2	14.8
1	2024-03-02	9.8	23.0	17.6
2	2024-03-03	7.6	20.3	16.0
3	2024-03-04	10.1	21.1	13.4
4	2024-03-05	11.2	18.5	15.1
..
87	2024-05-27	12.4	24.5	24.9
88	2024-05-28	17.8	20.6	22.3
89	2024-05-29	16.2	20.4	20.2
90	2024-05-30	15.5	20.7	21.7
91	2024-05-31	12.6	22.0	22.9

[92 rows x 4 columns]

Long Format

The melting process transforms it into the following long format:

	Date	City	Temperature
0	2024-03-01	Hamburg	12.0
1	2024-03-02	Hamburg	9.8
2	2024-03-03	Hamburg	7.6
3	2024-03-04	Hamburg	10.1
4	2024-03-05	Hamburg	11.2
..
271	2024-05-27	Tokyo	24.9
272	2024-05-28	Tokyo	22.3
273	2024-05-29	Tokyo	20.2
274	2024-05-30	Tokyo	21.7
275	2024-05-31	Tokyo	22.9

[276 rows x 3 columns]

How to melt DataFrames

- Use `pd.melt()` to transform from wide to long
- Parameters:
 - `id_vars`: columns to keep
 - `var_name`: name of the new column that will contain the names of the original columns
 - `value_name`: name of the new column that will contain the values of the original columns

...

```
df = pd.read_csv("supplementary/lec_08/employees.csv")
df = pd.melt(df, id_vars=['Position'], var_name='Variables',
             value_name='Values')
print(df)
```

	Position	Variables	Values
0	Manager	Name	Alice
1	Developer	Name	Bob
2	Analyst	Name	Charlie
3	Executive	Name	David
4	Representative	Name	Eve
..
195	Developer	Salary	69000
196	Assistant	Salary	49000
197	Analyst	Salary	55000
198	Manager	Salary	62000
199	Representative	Salary	51000

[200 rows x 3 columns]

Melting in Action

Task: Complete the following task:

```
# TODO: Load and transform the temperatures.xlsx file by melting it
# Expected output format:
#       Date      City  Temperature
# 0  2024-03-01   Hamburg         7.2
# 1  2024-03-01 Los_Angeles       18.5
# 2  2024-03-01    Tokyo        12.3
# Then, print the maximum temperature per city by grouping by the "City"
column
```

Programming with AI

Using AI to generate code

- Coding by hand is not the only way to generate code
- Most likely, a lot of you have already used ChatGPT

...

How do

Large Language

Models work?

Photo by Taylor Vick on Unsplash

Large Language Models (LLMs)

- Think of them like advanced pattern recognition systems
- They have “read” massive amounts of text
- Books, websites, articles, code, and more
- Text is broken into tokens, parts of words or punctuation
- Based on patterns, they can generate new text

Training LLMs

- Imagine learning a language by reading millions of books
- Learns patterns in how words and ideas connect via tokens
- Interconnected nodes with weights representing patterns
- During training, these weights are adjusted
- Once trained, applying them takes much less resources

Pattern Recognition

- Not like a search engine!
- When asked, it looks for relevant patterns it learned
- Like having a huge library in its “memory” to draw from
- It can find patterns between concepts and your question
- Knows only limited text at once (context window)

Probability based responses

- After each written token, it predicts “what should come next?”

- Like a advanced version of the word prediction on your phone
- Chooses the most likely next token based on training
- But can't actually "think" or "understand" like humans

Limitations

- No true understanding of cause and effect
- Sometimes makes mistakes or "hallucinates"
- Mostly only knows what it was trained on
- Can reflect biases present in training data
- No emotional understanding (but can simulate responses!)

Impact on Jobs

- Question: What do you think about their impact on jobs?
- Question: What are the implications for us?
- Question: Can we use them to our advantage?

(Current) Choices for Programmers

- [Github Copilot](#): Integrated into VS Code by Microsoft
- [Cursor](#): Fork of VS Code with AI assistance built in
- [Aider](#): Chat interface for AI to write code in the terminal
- [Zed](#): Lightweight IDE with AI features

...

Tip

Currently, [Zed](#) is my favorite one. But this might change in the future, as there is a lot of competition in this space.

Getting started with AI in Zed

- You will need to create an account (right top corner)
- Some free usage per month, after that you need to pay
- For us, the free plan should be more than enough
- If you need more prompts, create an account with [OpenRouter](#)
- Here you can get an API key and use their free models

Warning

If you use free models, be aware that your prompts are going to be used by the providers and are not private. But for learning and experimenting, this should be no issue.

Using Zed

- Open the folder with your tutorial files
- Create a new `.py` file

- Press `Ctrl + L` to open the chat

Asking for help

Task: Paste the following prompt in to the chat:

Can you please write me a small number guessing game in python? It should work for one player in the terminal. The player should guess a number between 1-10 and get hints about whether his guess was too large or too small. After 3 tries, end the game if he didn't succeed with a nice message.

...

Copy the generated code and paste it into your file.

More on Zed

- While working with Zed, it will suggest you code changes
- You can accept or reject them
- The rest you will learn by doing!

...

Note

And that's it for today's lecture!
You now have the basic knowledge to start working with tabular data and AI!.

Literature

Interesting Books

- Downey, A. B. (2024). Think Python: How to think like a computer scientist (Third edition). O'Reilly. [Link to free online version](#)
- Elter, S. (2021). Schrödinger programmiert Python: Das etwas andere Fachbuch (1. Auflage). Rheinwerk Verlag.

...

For more interesting literature to learn more about Python, take a look at the [literature list](#) of this course.