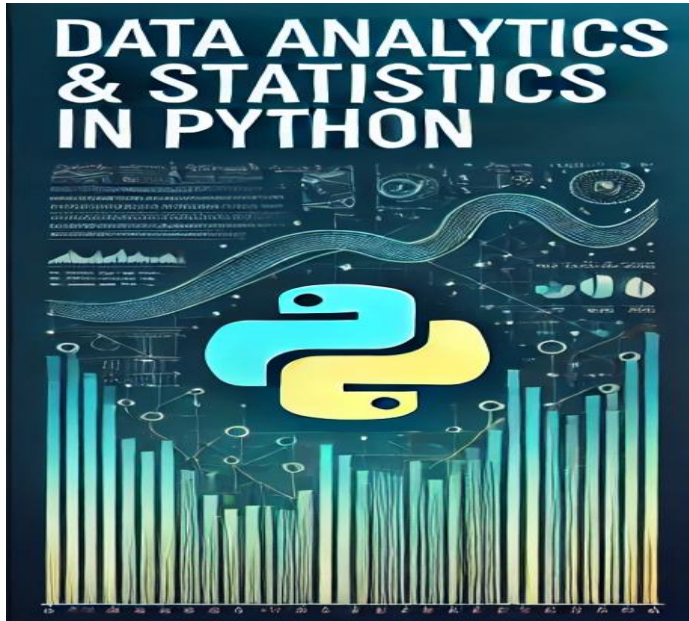


# Data Analytics & Statistics in Python

## Session 1: Course Introduction



*Learning data-driven decision-making with Python*

---

**Instructor:** Hamed Ahmadinia, Ph.D.

**Email:**

[hamed.ahmadinia@metropolia.fi](mailto:hamed.ahmadinia@metropolia.fi)

# Introducing Myself

- *Name:* Hamed Ahmadinia
- *Academic Background:*
  - Ph.D. in Information Studies
  - Master of Science in Information & Knowledge Management
  - Master degree in Business Administration – specialisation in Finance
  - Bachelor degree in Tax Accounting
- *Latest Research:*
  - Mobile Futures Project (data analysis and interpretation of quantitative data related to trust in information and the labor market integration).
  - Expertise in Analysing data with SPSS, Smart Pls, Data analysing with Python.

# Understanding Data Analytics

- *Definition:* The process of analyzing raw data to extract meaningful insights that guide decision-making is a core aspect of data analytics. It involves examining entire datasets to provide essential information to users (Vohra & Patil, 2021).
- *Purpose:* Make decisions patterns of data to make informed decisions rather than relying on assumptions.
- *Types of Data Analytics:*
  1. *Descriptive Analytics:* What happened? (Summarizes past trends)
  2. *Diagnostic Analytics:* Why did it happen? (Explains underlying reasons)
  3. *Predictive Analytics:* What will happen? (Forecasts future outcomes)
  4. *Prescriptive Analytics:* What should we do? (Recommends the best course of action)

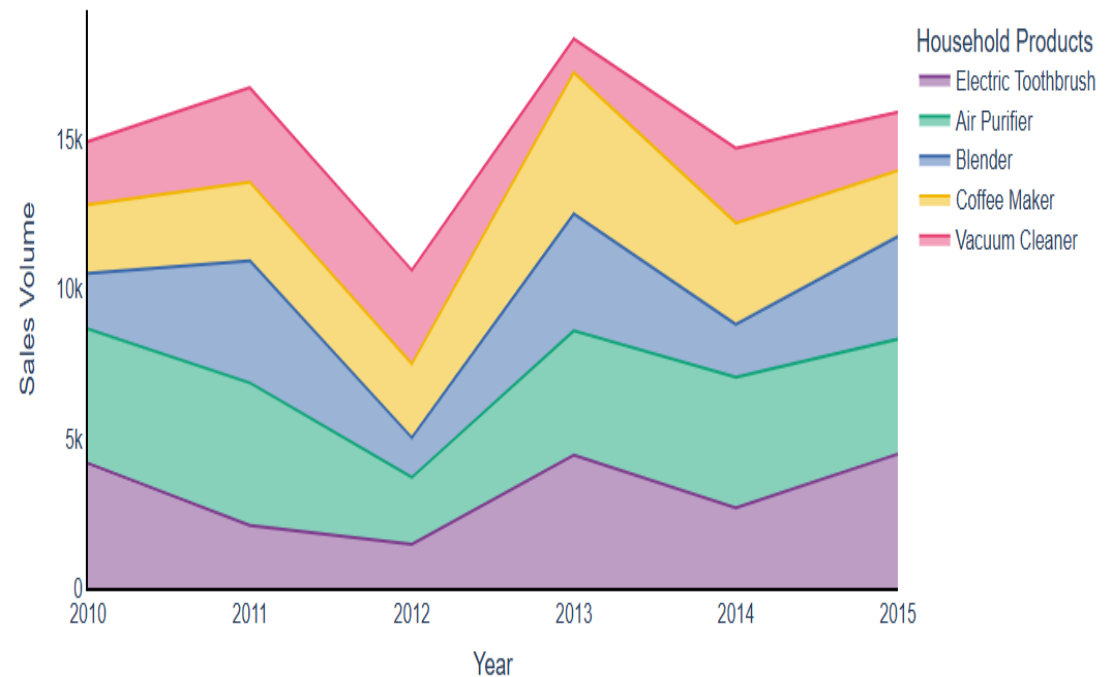
# Applications of Data Analytics:

## Business

- **Business:**

- Analyze customer purchases for targeted marketing.
- Optimize supply chain performance.
- Improve customer retention with predictive models.

Philips Household Products Sales for Targeted Marketing (2010-2015)

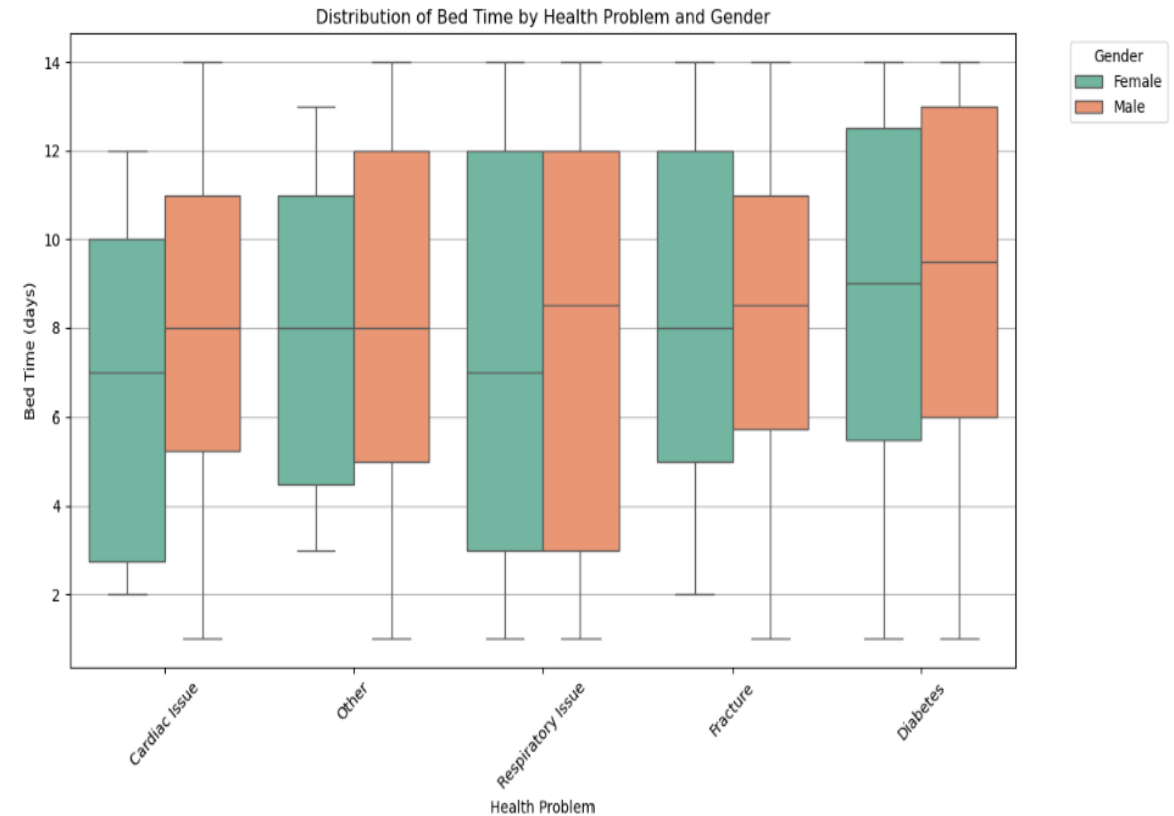


# Applications of Data Analytics:

## Healthcare

- **Healthcare:**

- Predict disease outbreaks using medical data.
- Create personalized treatment plans.
- Optimize resource allocation in hospitals.

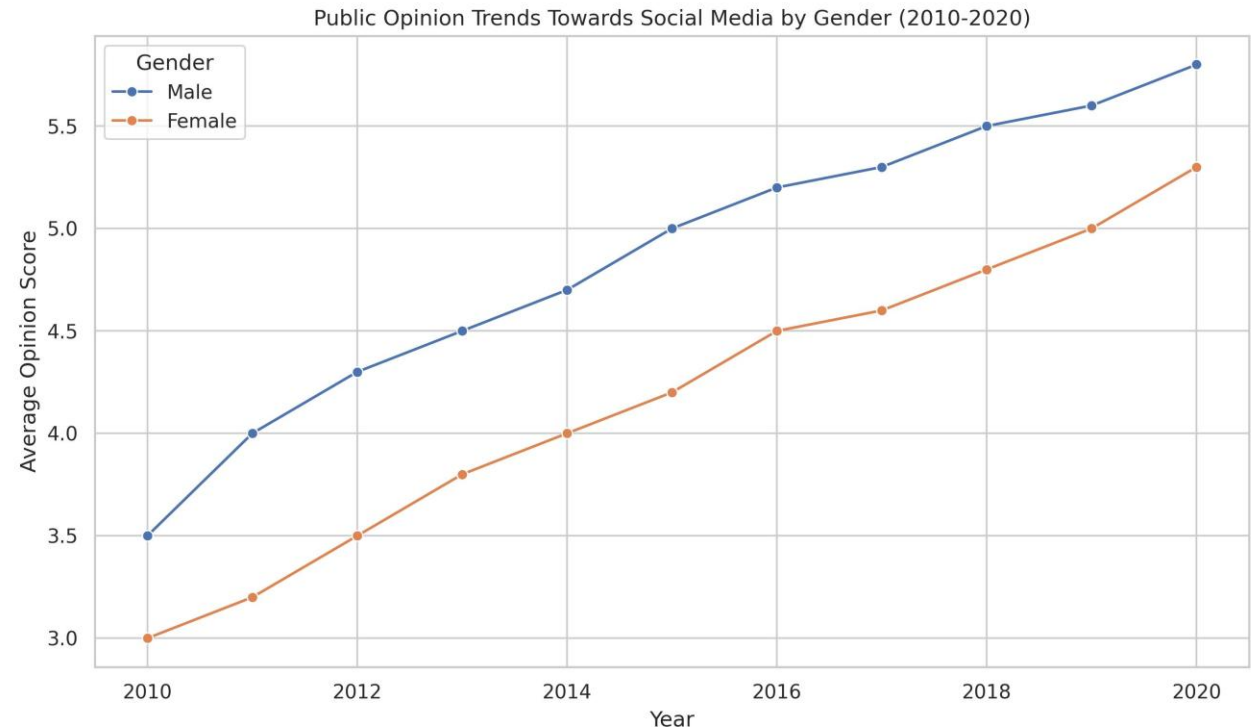


# Applications of Data Analytics:

## Social Science

- **Social Science:**

- Track public opinion trends.
- Measure the effectiveness of policies using social media data.

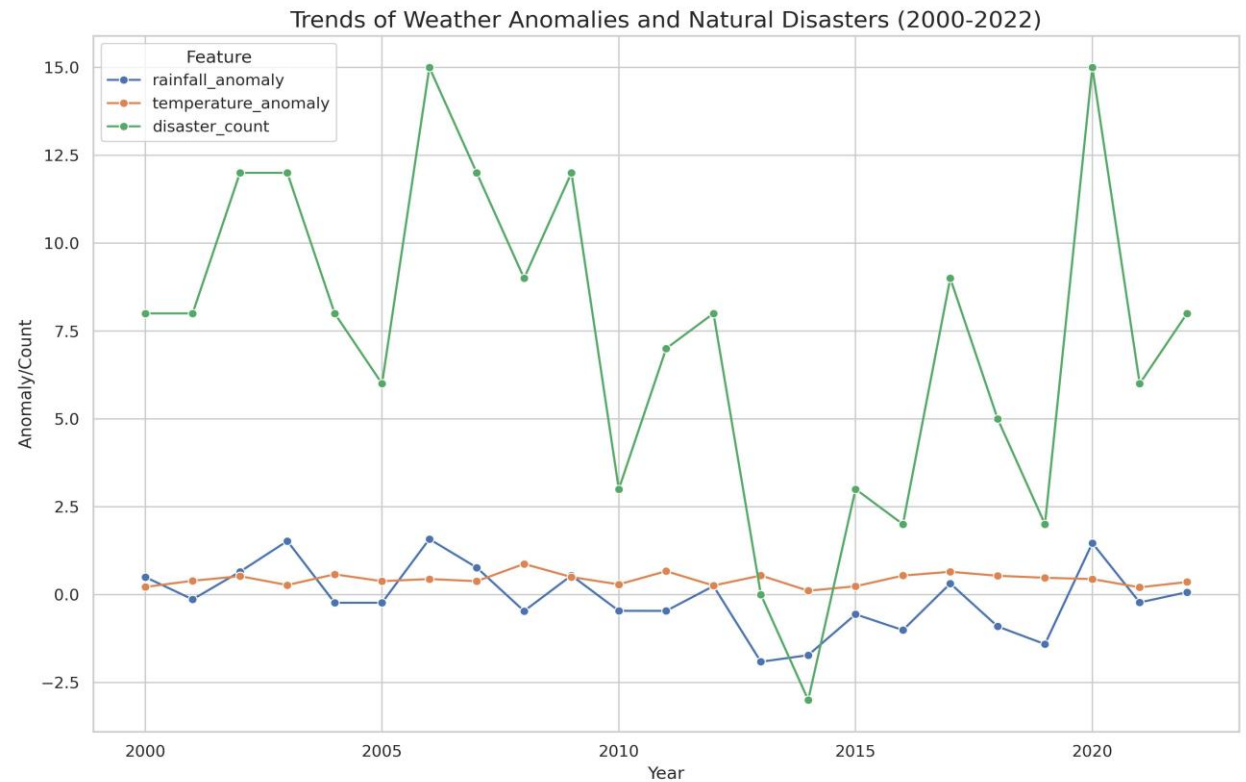


# Applications of Data Analytics:

## Environment

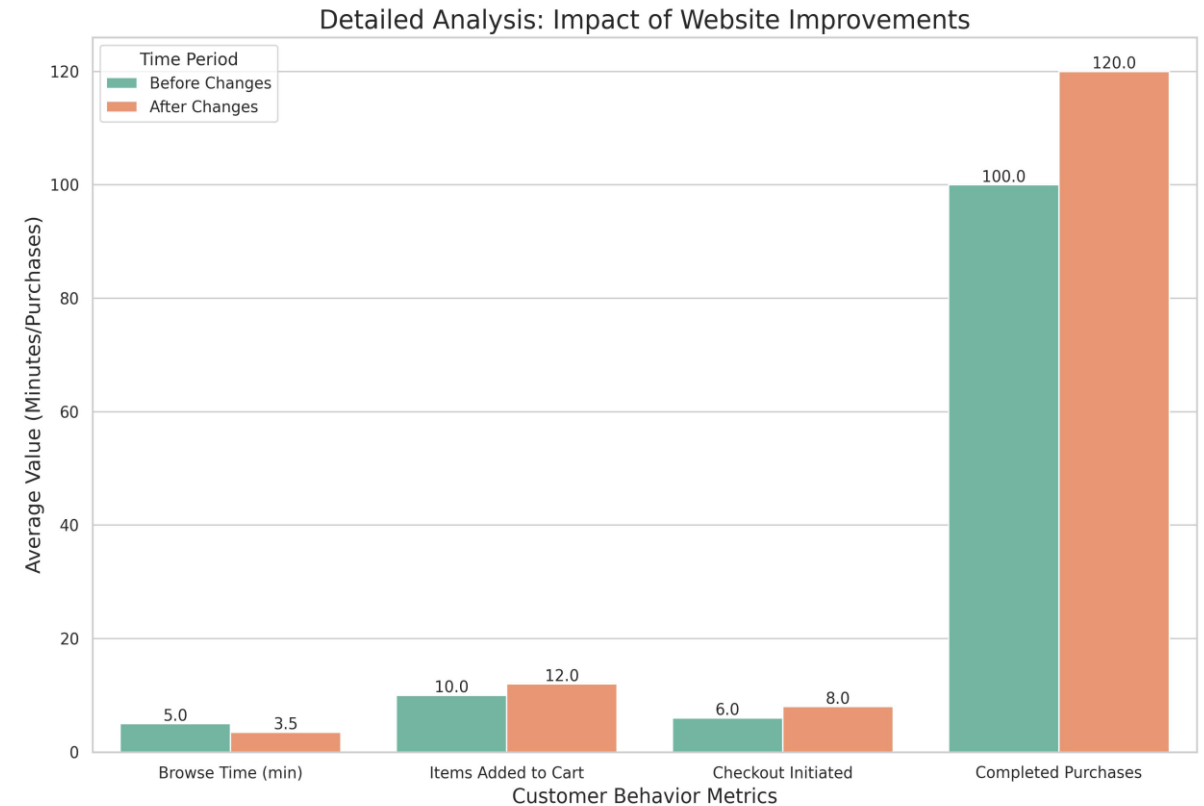
- **Environmental Science:**

- Monitor climate change using satellite and sensor data. (wind speed, temperature, and relative humidity)
- Predict natural disasters based on historical weather patterns.



# Example Use Case: E-Commerce

- **Problem:** Many customers leave items in their cart without completing the purchase.
- **Data Analysis:** Studied customer behavior, such as how long they browse and what items they add to their cart.
- **Result:** Improved the website based on the findings, leading to a 20% increase in purchases.





# Data Analytics in Leading Companies

- **Amazon:**
  - Uses data to personalize recommendations and optimize inventory management.
- **Google:**
  - Leverages data analytics for targeted advertising and search engine improvements.
- **Netflix:**
  - Utilizes data to recommend shows and movies based on user preferences.
- **Facebook (Meta):**
  - Analyzes user data to improve ad targeting and engagement.
- **Tesla:**
  - Uses real-time data from vehicles to improve autonomous driving systems.



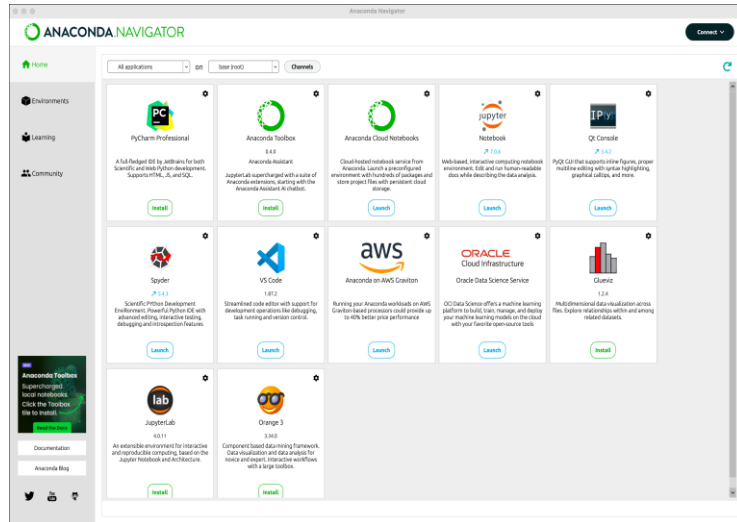
# Course Overview

- ***Purpose:*** To gain practical skills in data analytics and statistics using Python.
- ***Key Takeaways:***
  - Understand data types, control flows, and statistical methods.
  - Explore real-world datasets with Python libraries like pandas, matplotlib, seaborn, ...
  - Create effective data visualisations and interpret results.
- ***Course Focus:***
  - Reviewing important statistical concepts.
  - Hands-on coding in Jupyter Notebook.

# Course Structure

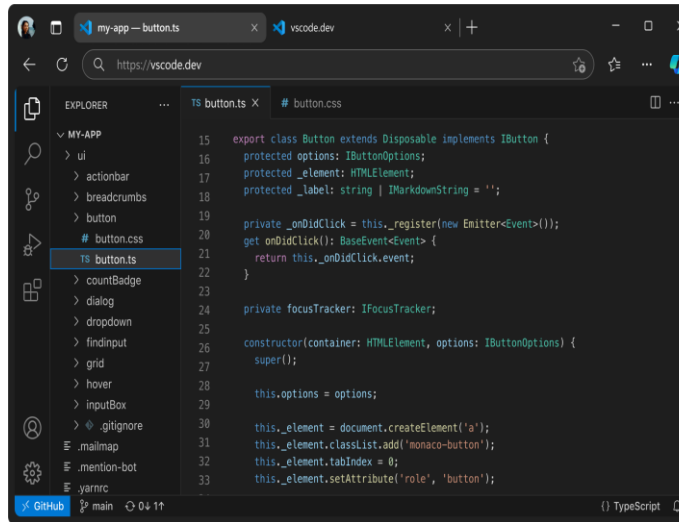
- *Objective:* Learn data analytics, statistics, and Python programming.
- *Duration:* 8 sessions (2 hours each).
- *Assessment:*
  - Hands-on exercises (30%).
  - Final project (70%).
- *Learning Approach:*
  - Real-world datasets and examples.
  - Collaborative learning environment.

# Recommended Tools



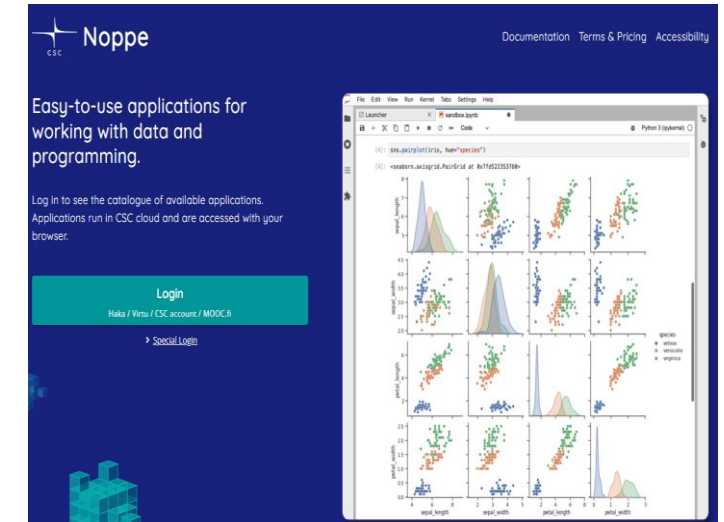
## Anaconda

Python distribution with essential libraries



## Visual Studio Code

Code editor



## Noppe

A user-friendly tool for data and programming

# Recommended Course Materials

- **Books:**

- *Python Data Science Handbook*

- Author: Jake VanderPlas

- *Practical Statistics for Data Scientists*

- Authors: Andrew Bruce & Peter Bruce

- *Python for Everybody Exploring Data Using Python 3*

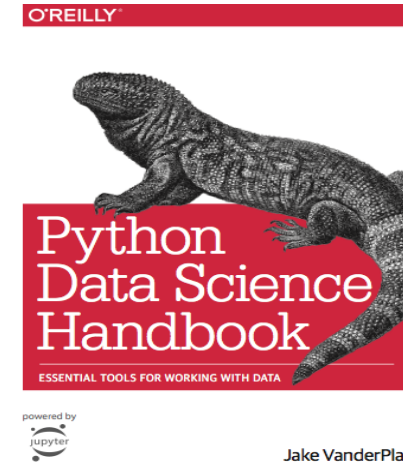
- Author: Charles Severance

- **Online Documentation:**

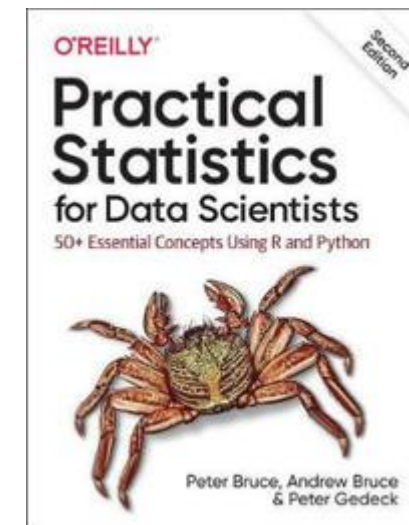
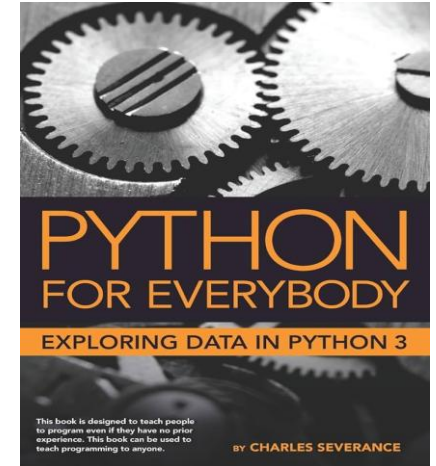
- Pandas: <https://pandas.pydata.org/>

- Matplotlib: <https://matplotlib.org/>

- **Other sources:** Kaggle, Github, ...



Jake VanderPlas



# Datasets Used in This Course

- ***Adult Income Dataset***: Demographic data for income classification ([Dataset Link](#)).
- ***Global Bike Sales Dataset***: Time-series sales data ([Dataset Link](#)).
- ***Medical Examination Dataset***: Health examination data ([Dataset Link](#)).
- ***Forum Pageviews Dataset***: Forum traffic analytics ([Dataset Link](#)).
- ***Sea Level Rise Dataset***: Environmental data tracking sea level changes ([Dataset Link](#)).
- ***World Happiness Report Dataset (2024 - Yearly Updated)***: Global happiness index data ([Dataset Link](#)).
- ***Spotify Streaming History Dataset***: Music streaming history data ([Dataset Link](#)).

# Concepts of Today

- **Recap of Python operations:**
  - **Common Operations:** *Arithmetic (+, -, \*, /, //, \*\*, %), Comparison (==, !=, <, >, <=, >=), Boolean (AND, OR, NOT), Assignment Shortcuts (+=, -=, \*=, /=).*
  - **Data Types:** Strings, Lists, Sets, Dictionaries.
  - **Control Flows:** *if/else, loops (for, while).*
  - **Functions:** Structure, recursion, common issues.
  - **File Handling:** Reading and writing to files (open(), modes like "r", "w", etc.)



# Common Python Operations

- **Purpose:** Python operations are crucial for data transformations and filtering.
- These operations form the foundation of most data analysis tasks.

Operation	Symbol	Description	Example	Output
Addition	+	Adds two values.	2 + 3	5
Division	/	Divides and returns a float.	10 / 4	2.5
Equal	==	Checks if two values are equal.	5 == 5	True
Exponentiation	**	Raises a number to the power of another.	2 ** 3	8
Floor Division	//	Performs division and returns the whole number.	10 // 3	3
Greater Than	>	Checks if one value is greater than another.	7 > 10	False
Greater or Equal	>=	Checks if a value is greater than or equal to another.	6 >= 3	True
Less Than	<	Checks if one value is less than another.	3 < 5	True
Less or Equal	<=	Checks if a value is less than or equal to another.	5 <= 5	True
Modulo	%	Returns the remainder of division.	10 % 3	1
Multiplication	*	Multiplies values.	4 * 3	12
Not Equal	!=	Checks if two values are not equal.	5 != 3	True
Subtraction	-	Subtracts one number from another.	5 - 2	3



# Data Types in Data Analytics

- **Why Data Types Matter:**

Data types help store and organize different types of information. Knowing data types helps you handle and analyze data efficiently.

- **Key Data Types in Python:**

1. **Int (Integer):** Represents whole numbers (e.g., 1, 42, -7).
2. **Float (Floating-point Number):** Represents decimal numbers (e.g., 3.14, -0.5).
3. **Strings** – Text data like names and descriptions (e.g., "Hello", "123").
4. **Lists (Arrays)** – Ordered collections of items (e.g., [1, 2, 3], ["apple", "banana"]).
5. **Sets** – Collections of unique, unordered items (e.g., {1, 2, 3}).
6. **Dictionaries** – Key-value pairs (e.g., {"name": "Alice", "age": 30}).

# Strings – Handling Text in Python

- **Definition:** Strings store text data (e.g., words and sentences).
- **Examples of String Operations:**
  - **Concatenation** (combine strings): "Hello " + "World" → "Hello World"
  - **Slicing** (get parts of a string): "apple"[1:4] → "ppl"
  - **Reversing:** "apple"[::-1] → "elppa"
- **Useful String Methods:**
  - **split()** – Splits text into a list of words.
  - **strip()** – Removes unnecessary spaces or characters.
  - **join()** – Combines words into a single string.

# Lists – Organizing Data in Order

- **Definition:** Lists store collections of items in a specific order.
- **Adding to a List:**
  - `append(item)` – Adds to the end of the list.
  - `insert(index, item)` – Adds at a specific position.
- **Removing from a List:**
  - `remove(value)` – Removes the first matching value.
  - `pop(index)` – Removes the item at a specific index.
- **Sorting Lists:**
  - `sorted(list)` – Returns a new sorted list without changing the original.
  - `.sort()` – Sorts the list directly.
- **List Comprehension: A shortcut for creating lists.**  
Example: `[x for x in range(5)]` → `[0, 1, 2, 3, 4]`

# Sets – Unique Collections

- **Definition:** Sets store unique, unordered items (no duplicates).
- **Why Use Sets:** Great for checking membership and ensuring no duplicates.
- **Common Operations:**
  - `add(item)` – Adds an item to the set.
  - `remove(item)` – Removes an item.
  - `len(set)` – Returns the number of unique items.

- **Example:**

```
my_set = set([1, 1, 2, 3])  
print(my_set)  # Output: {1, 2, 3}
```

# Dictionaries – Key-Value Data Storage

- **Definition:** Dictionaries store data as key-value pairs (like labels with information).
- **Example:**

```
my_dict = {'name': 'Alice', 'age': 25}
```

Key: 'name' → Value: 'Alice'

- **Common Operations:**
  - my\_dict[key] – Accesses the value for a key.
  - dict.keys() – Lists all keys.
  - dict.values() – Lists all values.
  - pop(key) – Removes a key-value pair.
- **Practical Use:** Contact lists, customer records, etc.

# Control Flows in Python

- **What are Control Flows?**

- Determine the flow of execution based on conditions.
- Allow for loops and conditional logic.

- **Types of Control Flows:**

- **if:** Executes when the condition is True.
- **elif:** Adds more conditions after if.
- **else:** Executes when no conditions are met.

```
score = 85
if score >= 90:
    print("Grade: A")
elif score >= 75:
    print("Grade: B")
else:
    print("Grade: C")
```

# Loops in Python

- **For Loops:**

- Iterate over a sequence (e.g., list, string).

```
sequence = ["apple", "banana", "cherry"]
for item in sequence:
    print(item)
```

- **Control Statements:**

- **break:** Exit the loop early.
- **continue:** Skip the rest of the loop body and continue with the next iteration

- **While Loops:**

- Repeats execution while the condition remains True.

```
count = 1
while count <= 5:
    print("Count:", count)
    count += 1
```

```
sequence = ["apple", "banana", "cherry"]
for item in sequence:
    if item == "banana":
        break # Exits the loop when "banana" is found
    print(item)
```

```
sequence = ["apple", "banana", "cherry"]
for item in sequence:
    if item == "banana":
        continue # Skips "banana" and moves to the next item
    print(item)
```

# Understanding Functions in Python

- **What is a Function?**

- A reusable block of code that performs a specific task.
- Helps avoid repetitive code and makes debugging easier.
- Functions can take inputs (parameters) and return outputs.

- **Key Components:**

- **Function Name:** Identifies the function.
- **Parameters:** Inputs passed into the function.
- **Return Statement:** Sends back the result.

```
# Function to greet a user
def greet_user(name):
    return f"Hello, {name}!"

print(greet_user("Alice")) # Output: Hello, Alice!
```

The name parameter is used inside the function to personalise the greeting.



# Recursion, Global Variables, and Common Issues

- **Recursion:**

- A function that calls itself to break a problem into smaller steps.

```
# Simple countdown function
def countdown(n):
    if n == 0:
        return "Done!"
    else:
        return countdown(n - 1)
print(countdown(3)) # Output: Done!
```

- **Global vs Local Variables:**

- **Global Variable:** Can be accessed and modified anywhere in the program.
- **Local Variable:** Exists only inside the function.

```
x = 10 # Global variable

def update_value():
    global x
    x = 5 # Changes global x
update_value()
print(x) # Output: 5
```

# Input in Python

- **What is Input?**


- Allows users to enter data into the program.
- Returns user input as a string.

```
# Simple input example  
data = input("Enter your name: ")  
print(f"Hello, {data}!")
```

Enter your name: ↑↓ for history. Search history with c-↑/c-↓

## Explanation:

- Input is always stored as a string, even if the user enters a number.
- To handle numbers, you must convert the input to an integer or float.



```
[3]: # Simple input example  
data = input("Enter your name: ")  
print(f"Hello, {data}!")
```

Enter your name: Alex  
Hello, Alex!

# Understanding Type Conversion in Python

- **Int to Float/String:** `float(3) → 3.0`, `str(3) → "3"`
- **Float to Int/String:** `int(3.9) → 3`, `str(3.9) → "3.9"`
- **String to Int/Float (if numeric):** `int("5") → 5`, `float("5.5") → 5.5`
- **String to List/Set:** `list("abc") → ['a', 'b', 'c']`, `set("abc") → {'a', 'b', 'c'}`
- **List to String:** `str([1, 2]) → "[1, 2]"`
- **Set to List:** `list({1, 2}) → [1, 2]`
- **List to Dictionary Keys:** `dict.fromkeys(["name", "age"]) → {'name': None, 'age': None}` \*\*

\*\* The `dict.fromkeys()` function creates a dictionary using the elements from a list (or any iterable) as keys. The values are set to `None` by default unless specified.

# Formatting Output

- **Using f-strings:**

- Format numbers directly into print statements.

```
python

name = "Alice"
age = 25
print(f"{name} is {age} years old.")
```

- **Line Breaks and Tabs:**

- \n → New Line
- \t → Tab Space

```
python

print("Line 1\nLine 2")
print("Tabbed:\tHello")
```

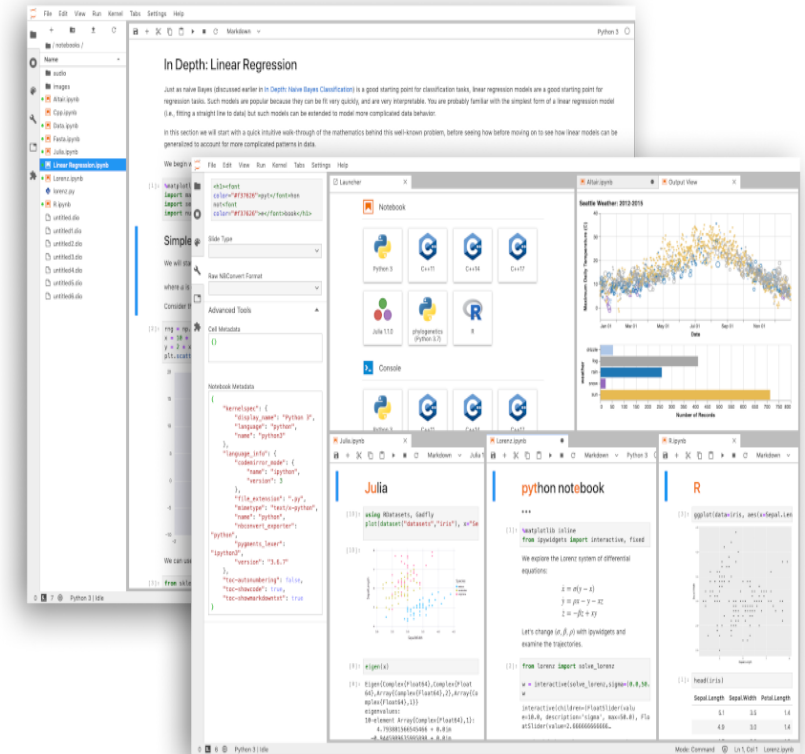
# Reading and writing data to files

- **Importance:** Essential for data analysis to read/write text, numerical, or mixed data.
- **File Operations in Python:**
  - **open( \_\_\_\_, \_\_ ) as file:** Opens a file with the specified mode:
    - "r": Read mode (default) – reads file content.
    - "w": Write mode – overwrites existing content.
    - "a": Append mode – adds new content without overwriting.
    - "r+": Read/Write mode – reads and updates without truncating.
    - "w+": Read/Write mode – writes and truncates.
- **Common File Methods:**
  - **file.read():** Reads the entire file content as a string.
  - **file.write():** Writes data to the file.
  - **file.readline():** Reads a single line from the file.
  - **file.readlines():** Reads all lines and returns them as a list.
  - **file.writelines():** Writes a list of strings to the file.

# Notebook Review

Walk through how to apply key Python concepts in a Jupyter Notebook:

- Common Operations
- Data Types
- Control Flows
- Functions
- Input and Output Formatting
- Debugging
- File Handling



# Kahoot Quiz Time!

# Kahoot!

*Let's Test Our Knowledge!*



# Hands-on Exercise

**Form groups (2–3 members).**

- Download *Hands-on Exercise #1* from the course page.
- Complete the coding tasks and discuss your solutions.
- Don't forget to add the names of your group members to the file.
- Submit your completed *Hands-on Exercise* to the course Moodle page or send it to the teacher's email address.





# Reference

- Vohra, M., & Patil, B. (2021). A Walk Through the World of Data Analytics. , 19-27. <https://doi.org/10.4018/978-1-7998-3053-5.ch002>.
- VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data. O'Reilly Media. Available at <https://jakevdp.github.io/PythonDataScienceHandbook/>
- Severance, C. (2016). Python for everybody: Exploring data using Python 3. Charles Severance. Available at <https://www.py4e.com/html3/>
- McKinney, W. (2017). *Python for data analysis: Data wrangling with pandas, NumPy, and Jupyter*. O'Reilly Media.