DS 520/CS 520 – Lecture 1

DS-520-50: Data Analy: Conc/Tech
2024 Fall
MONMOUTH CAMPUS

T 7:30 PM - 10:20 PM
9/3/2024 - 12/9/2024
Howard Hall, 206 LECTURE

Arup Das

adas@Monmouth.edu

Disclaimer:

- The views expressed are solely those of the presenter and not affiliated with any other party.
- This presentation is free of copyright violations, and external sources have been appropriately credited.
- T**he content within this presentation is legally protected; unauthorized reproduction, including photography, will result in legal action.**
- **This material is not intended for distribution and must remain solely within the confines of this class. Do not distribute slides or assignments to other students**
- **Using cameras to take screenshots or photographs of the slides is strictly prohibited**.

Course Logistics

Introduction to Machine Learning using Python

Master Syllabus

Course Code: DS-520/CS-520
Course Title: Introduction to Machine Learning using Python
Credits: 3
Professor: Arup Das , email: adas@monmouth.edu

Catalog Description:
This course is designed to introduce students to the discipline of data analytics and teach them basic algorithms, methodologies, and techniques for data analysis. It presents the main steps of a data analysis process, including data gathering and collection, exploratory data analysis, data mining algorithms, and evaluation methodologies. **The most important data mining techniques are introduced: classification, regression, and clustering analysis**. The students will gain **familiarity with Python data analysis libraries**.
.

Prerequisites: Prerequisite(s): DS-501, DS-502, and DS-504

Introduction to Machine Learning using Python

Course Objectives

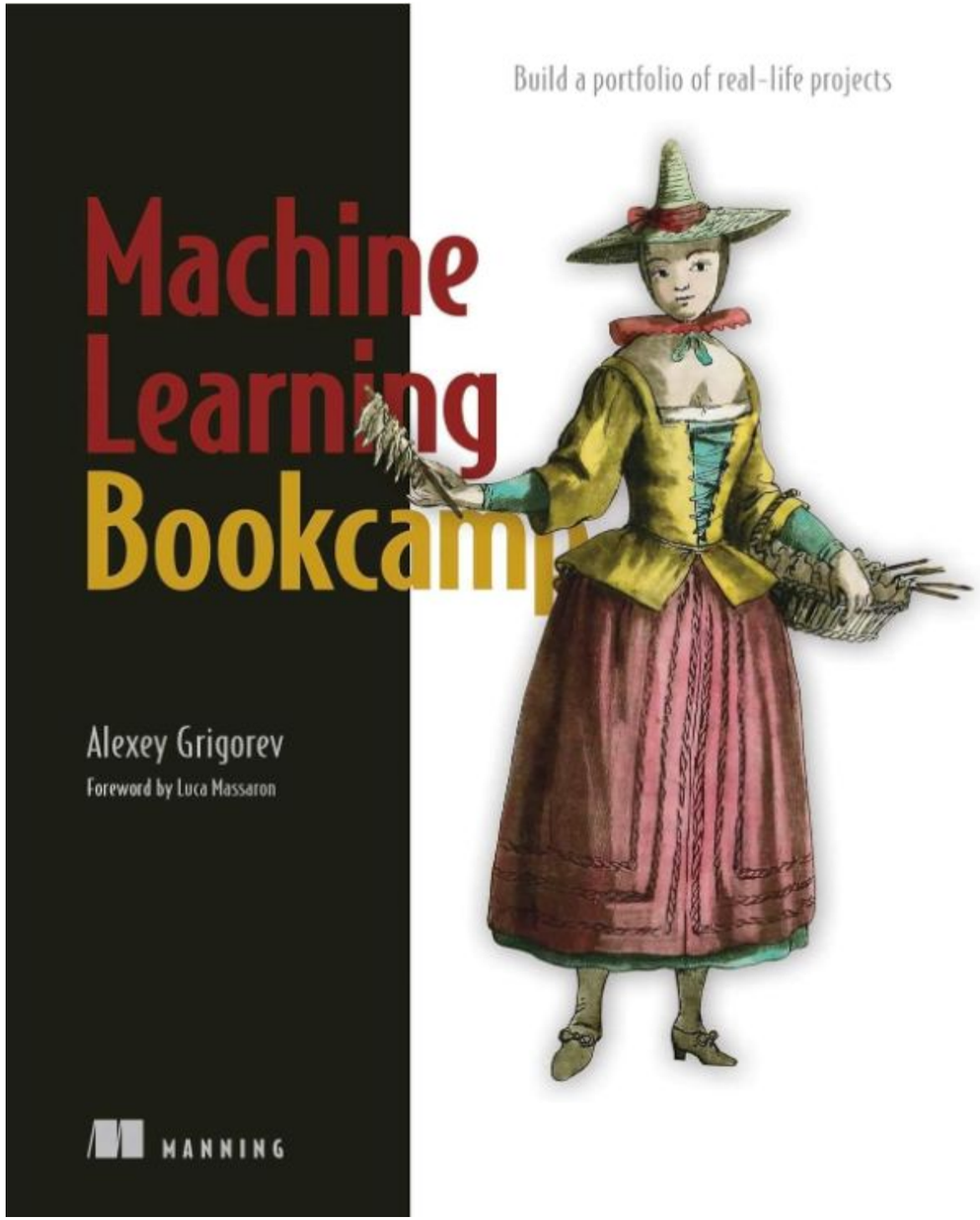
By the end of this course, students will master the core concepts of data analytics and machine learning and develop practical skills in using Python and its libraries for data analysis, model building, and applying critical algorithms like classification, regression, and clustering to real-world problems. They will also learn to conduct exploratory data analysis (EDA), feature engineering, and efficiently implement machine learning workflows and pipelines.

Assessable Learning Outcomes:

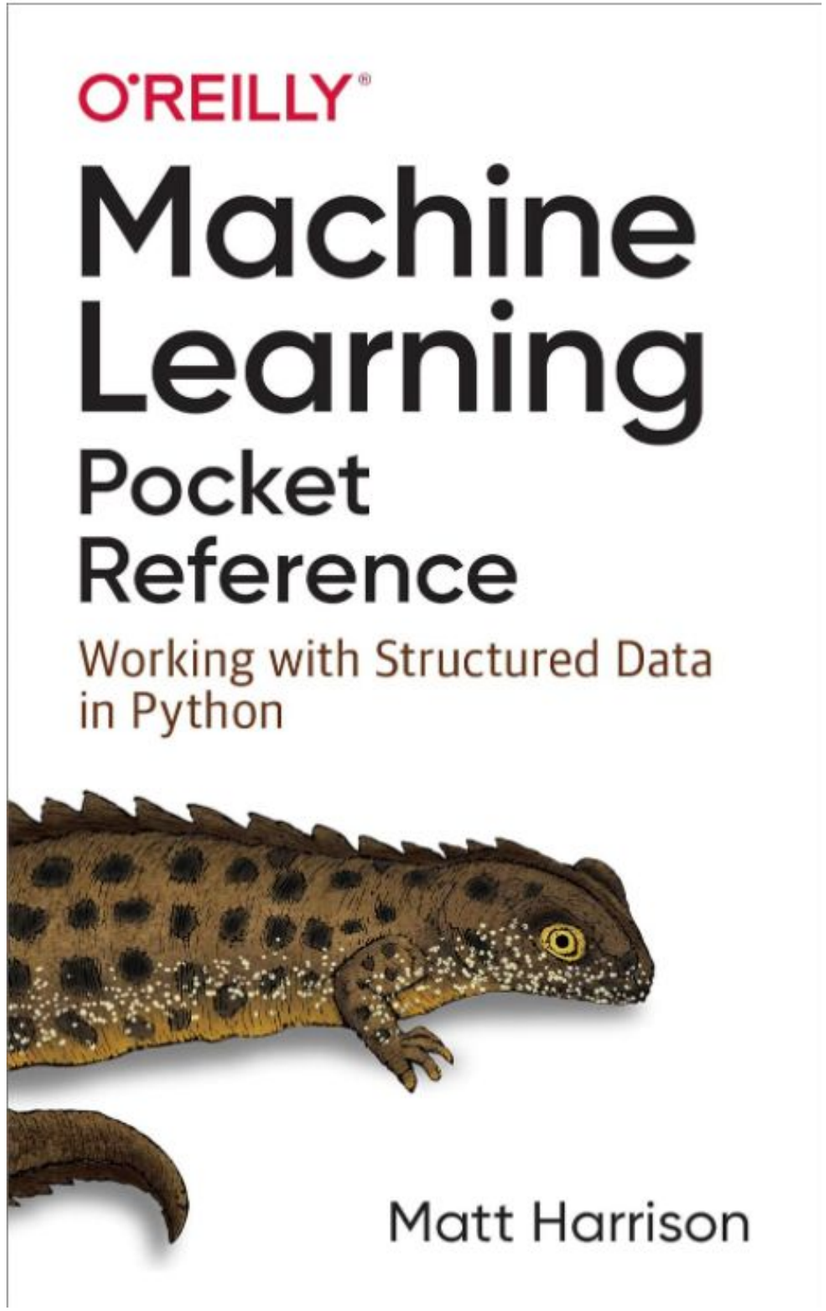
- Master the basics of Python for machine learning, including **data structures and libraries**.
- Conduct exploratory data analysis using Python to uncover insights from data.
- Implement **regression and classification models** and evaluate their performance.
- Understand and apply techniques for **feature engineering, model selection, and tuning**.
- Work with unsupervised learning methods such as **clustering and dimensionality reduction**.

Required Textbook

Book1: Machine Learning Book camp: Build a portfolio of real-life projects- ISBN - 978-1617296819

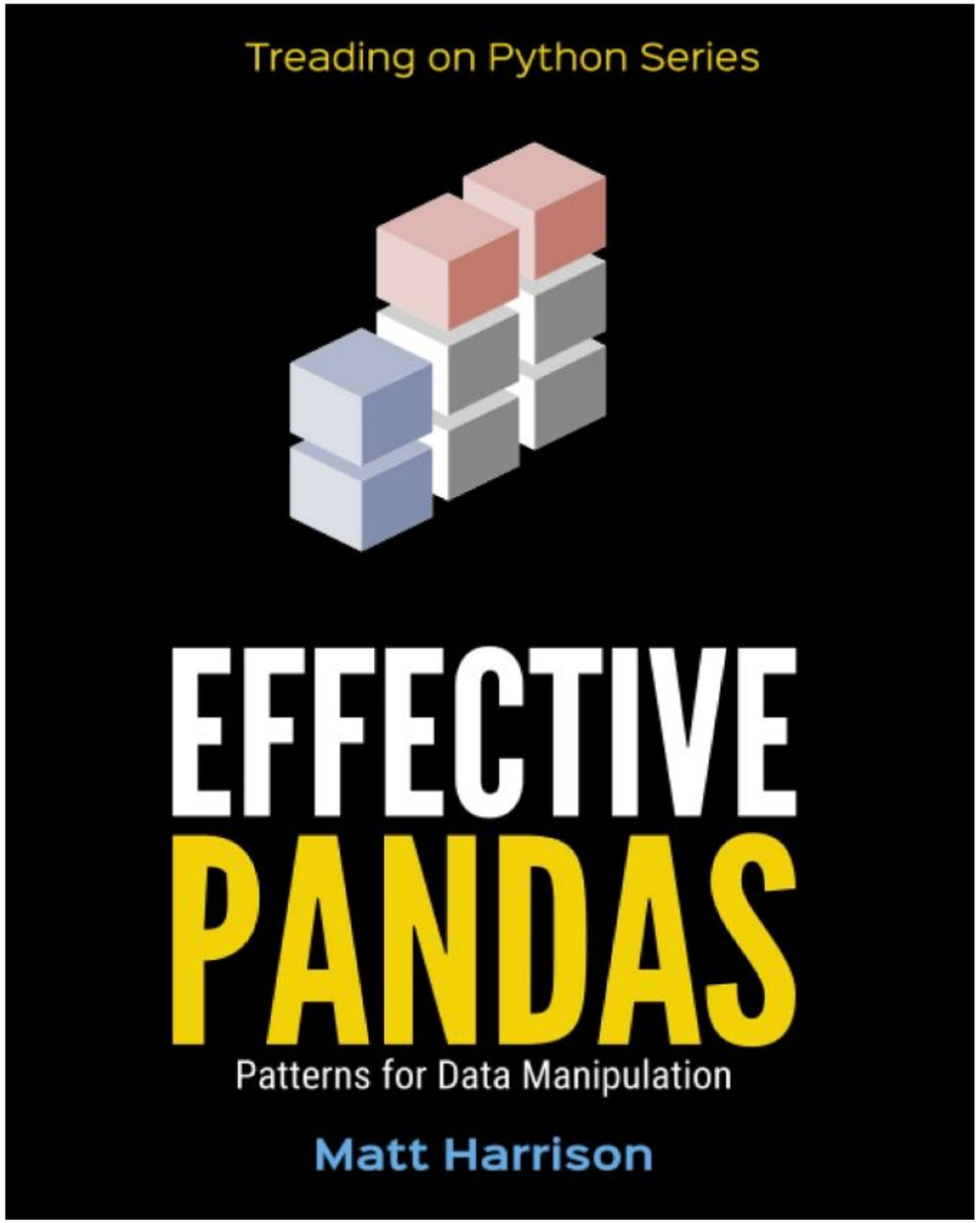


Machine Learning Pocket Reference: Working with Structured Data in Python –
ISBN
978-1492047544



Optional:

Effective Pandas: Patterns for Data Manipulation (Optional) – ISBN - 979-8772692936



Week 1: Python for Machine Learning Refresher

- Variables, data structures (List, Tuple, Set, Dictionary)
- Loops (for, while), Numpy Arrays, Pandas Series and DataFrames
- Accessing and modifying data, Python libraries (Plotly, tqdm)
- Google Colab functions, magic functions, Python notebook structure
- Introduction to building applications using Streamlit and Gradio

Week 2 & 3: Machine Learning Workflow and Exploratory Data Analysis (EDA)

- Machine Learning workflow overview
- Types of Machine Learning: Supervised, Unsupervised, and Reinforcement Learning
- Fundamentals of EDA: Data variables, categorical vs. numerical data
- Univariate, Bi-variate, and Multivariate analysis
- In-class walkthrough of an EDA notebook **(HW 1 distributed)**

Week 4 & 5: Machine Learning for Regression

- Application of Regression (Business Use Cases)
- Exploratory Data Analysis, Handling Missing Values, Target Variable Analysis
- Implementing Price Prediction using Regression, RMSE
- Simple Feature Engineering, Handling Categorical Variables, Regularization
- In-class walkthrough of a Python notebook **(HW 2 distributed)**

Week 6 & 7: Machine Learning for Classification

- Application of Classification (Business Use Cases)
- Data Preparation, EDA, and Feature Engineering for Classification
- Logistic Regression, Model Interpretation
- Evaluation metrics for Classification: Confusion Matrix, Precision, Recall, ROC Curve, AUC Score
- **(HW 3 Distributed)**

Week 8: Quiz 1

- **Covers material from Weeks 1-7**

Week 9 & 10: Feature Selection, Model Selection, and Tuning

- Feature engineering and cross-validation
- Oversampling and under-sampling techniques, Regularization models
- Building ML pipelines, Hyperparameter tuning (Random Search CV)
- **(HW 4 Distributed)**

Week 11 & 12: Unsupervised Learning

- K-means clustering, Silhouette Coefficient for K-means
- Hierarchical clustering, Dimensionality Reduction techniques (PCA, t-SNE)
- **HW 5 Distributed**

Week 13: AI Certification Preparation

- Discussion on AI certifications: AWS, Google, Microsoft
- Preparing for certification exams

Week 14: Quiz 2 and Course Wrap-Up

- **Quiz covering material from Weeks 9-13**

Fall 2024 Academic Calendar

September
Tuesday, Sept. 3 – Classes Begin
Tuesday, Sept. 3 – Tuesday, Sept. 10 – Late Registration / Drop – Add / Leave of Absence
Friday, Sept. 27 – “W” Deadline for “A” Session

October
Saturday, Oct. 12 – Tuesday, Oct. 15 – **Fall Holiday (non-weekend students)**
Tuesday, Oct. 22 – Undergraduate Midterm Grades Due
Tuesday, Oct. 22 – Session “A” Classes End
Wednesday, Oct. 23 – Session “B” Classes Begin
Wednesday, Oct. 30 – Pattern B add/drop
Thursday, Oct. 31 – “W” Deadline

November
Monday, Nov. 18 – “W” Deadline for “B” Session Classes
Wednesday, Nov. 27 – Sunday, Dec. 1 – **Thanksgiving Holiday**

December
Monday, Dec. 9 – Thirteenth Week Ends
Tuesday, Dec. 10 – Reading Day
Wednesday, Dec. 11 – Tuesday, Dec. 17 – Fourteenth Week Adjusted Schedule
Friday, Dec. 20 – End of Final Grading Period

Date	Week	Class Format/Location/Time	Topics	Readings Required (Due before class)	Assignment/Quiz
September 3,2024	Week_1	On-Premise/Howard Hall, 206 LECTURE/7:30 PM-10:20PM	Python for Machine Learning Refresher		
September 10,2024	Week_2	On-Premise/Howard Hall, 206 LECTURE/7:30 PM-10:20PM	Machine Learning Workflow and Exploratory Data Analysis (EDA)	Book 1 – Chapter 1, Chapter 2 (Pages 22 – Page 29)	
September 17,2024	Week_3	On-Premise/Howard Hall, 206 LECTURE/7:30 PM-10:20PM	Machine Learning Workflow and Exploratory Data Analysis (EDA)	Book 1 – Chapter 2 (Page 32- 63)	**Project 1 Distributed - Due Sep 27,2024**
September 24, 2024	Week_4	On-Premise/Howard Hall, 206 LECTURE/7:30 PM-10:20 PM	Machine Learning for Regression	Book 1 – Chapter 2 (Page 32- 63)	
October 1, 2024	Week_5	On-Premise/Howard Hall, 206 LECTURE/7:30 PM-10:20 PM	Machine Learning for Regression	Book 1 – Chapter 3 (Page 65- 110)	**Project 2 Distributed - Due Oct 12, 2024**
October 6, 2024	Week_6	On-Premise/Howard Hall, 206 LECTURE/7:30 PM-10:20 PM	Machine Learning for Classification	Book 1 – Chapter 4 (Page 113- 145)/Chapter 6	
October 22, 2024	Week_7	Zoom remote/8:00pm – 10:30 pm	Machine Learning for Classification	Book 1 – Chapter 4 (Page 113- 145), /Chapter 6	**Project 3 Distributed – Due Nov 1, 2024**
October 29,2024	Week_8	Zoom remote/8:00pm – 10:30 pm	Quiz 1 (Cover materials from Week 1-7)	Book 1 – Chapter 3 (Pages 88- 92)	**Quiz 1 – Open Book/Open Notes**
November 5, 2024	Week_9	Zoom remote/8:00pm – 10:30 pm	Feature Selection, Model Selection and Tuning	Book 1 – Chapter 4 (Pages 147- 151)	
November 12, 2024	Week_10	Zoom remote/8:00pm – 10:30 pm	Feature Selection, Model Selection, and Tuning	Book 1 – Chapter 4 (Pages 147- 151)	**Project 4 Distributed – Due Nov 22, 2024**
November 19,2024	Week_11	Zoom remote/8:00pm – 10:30 pm	Unsupervised Learning	Professor Lecture Notes	
November 26, 2024	Week_12	Zoom remote/8:00 pm – 10:30 pm	Unsupervised Learning	Professor Lecture Notes	**Project 5 Distributed – Due Dec 6, 2024**
December 3, 2024	Week_13	Zoom remote/8:00 pm – 10:30 pm	AI Certifications Overview or additional topics spill over from preceding weeks	Professor Lecture Notes	**Quiz 2 Distributed**
December 9, 2024 – Last day of class	Week_14	Quiz 2 Due	Quiz 2 (Covers materials from Weeks 9 -12) and Course wrap-up		**Quiz 2 Due Dec 8, 2024 before midnight EST**

Course Logistics

1. OneDrive link for professor notes and assignments/quiz
2. Check your Monmouth email for announcements
3. Check your Monmouth calendar for Zoom links for office hours and remote lectures
4. My contact information: adas@monmouth.edu, Cell # 917-523-7683
4. Office hours (zoom only) – Friday (EST) 7-7:30 pm EST
5. Assignment submission to professoraruprdas@gmail.com (Notation for files: Assignment_1_Name_of_Student), Colab notebooks ipynb file and html file, all presentation in ppt format.
6. Quiz submission to professoraruprdas@gmail.com (Notation for file : Quiz_1_Name_of_Student. doc , Quiz_2_Name_of_Student.doc)

Methods of Evaluation

Projects – 5 hands-on projects – 50%
Quiz 1 – 25%
Quiz 2 – 25%

Letter Grade	Percentage Points
A	100-93
A-	92-90
B+	89-87
B	86-83
B-	82-80
C+	79-77
C	76-73
C-	72-70
F	69-0

Academic Honesty

Everything you turn in for grading must be your work. Academic dishonesty subverts the University's mission and undermines the student's intellectual growth. Therefore, we will not tolerate violations of the code of academic honesty. Penalties for such violations include suspension or dismissal and are elaborated upon in the Student Handbook.

A guide to plagiarism can be found at http://www.plagiarism.org/.

Week 2 Deliverables

Next Lecture Deliverables & Lecture_2

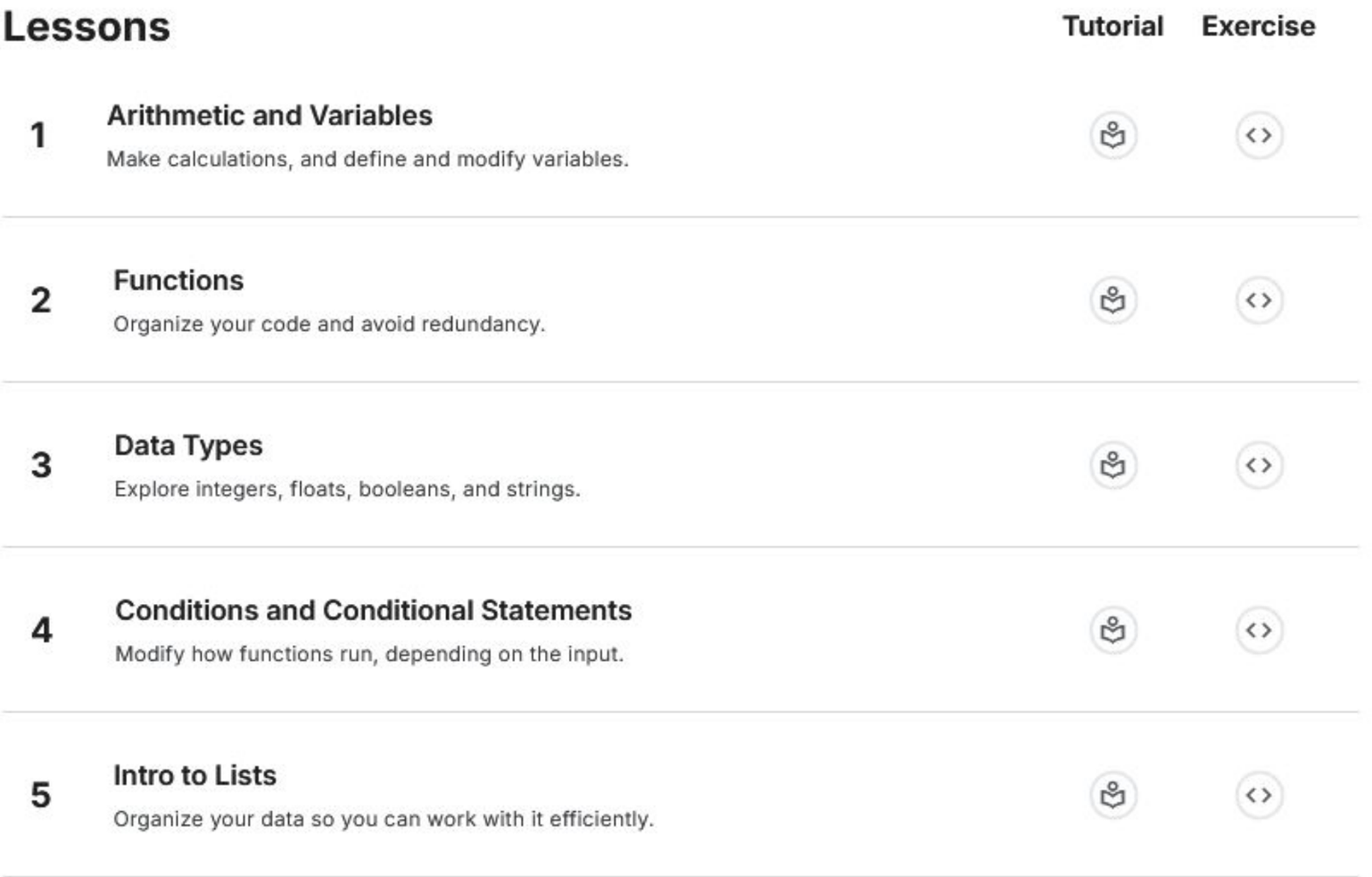
1. Review Lecture 1 Notes and class notebooks, video links - (All in yellow please review links and videos)
2. Complete readings Book 1 – Chapter 1, Chapter 2 (Pages 22 – Page 29)
3. Start working on Learning and Badges
4. **Lecture 2 - Exploratory Data Analysis**
5. **Assignment 1 will be distributed on Sep 17, 2024 and due on Sep 27, 2024 - Please follow the assignment submission guidelines outlined in the logistics slides**

Learning and Badges (Add to Linkedin)

Intro to Programming

Get started with Python, if you have no coding experience.

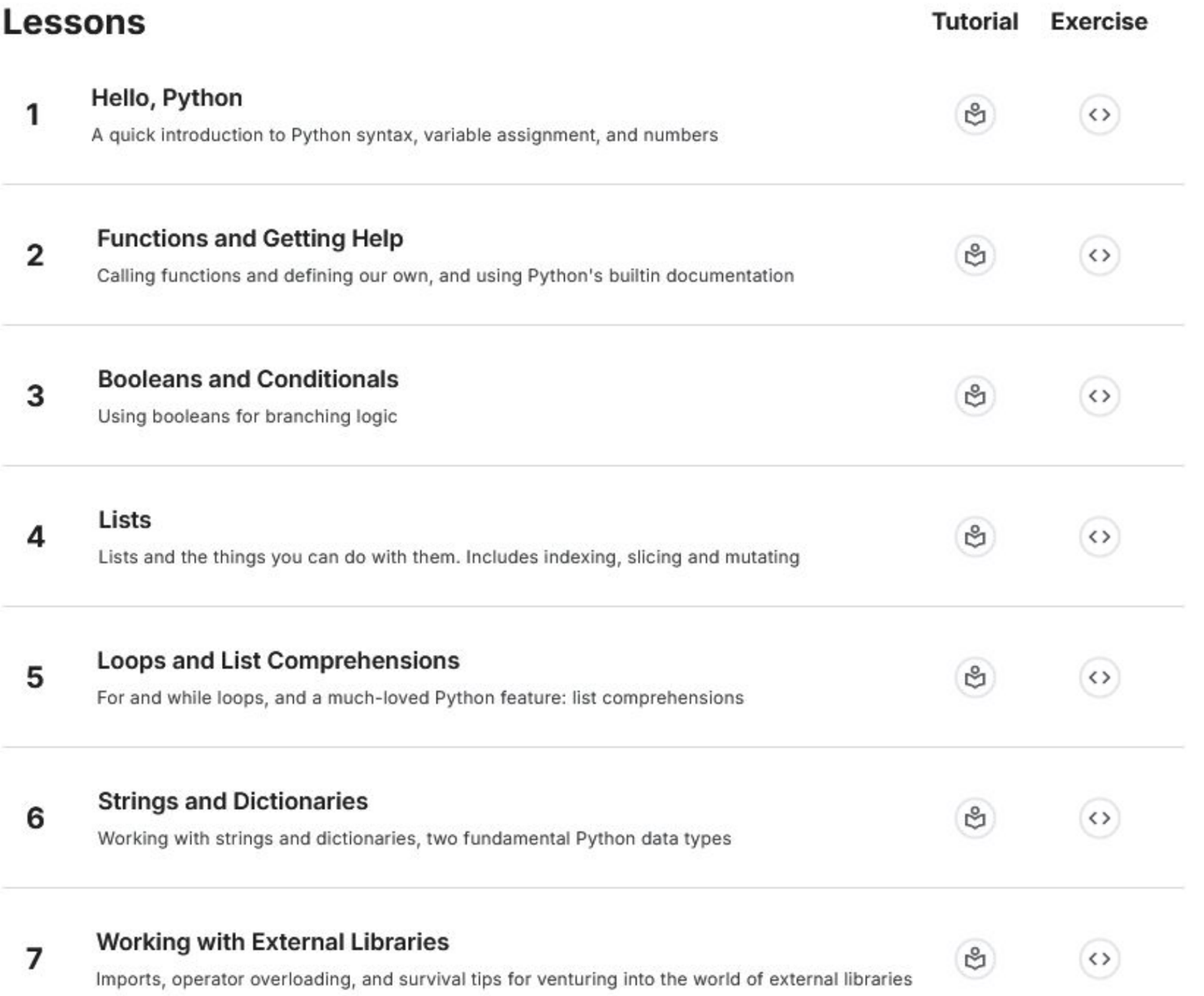
https://www.kaggle.com/learn/intro-to-programming



Python

Learn the most important language for data science.

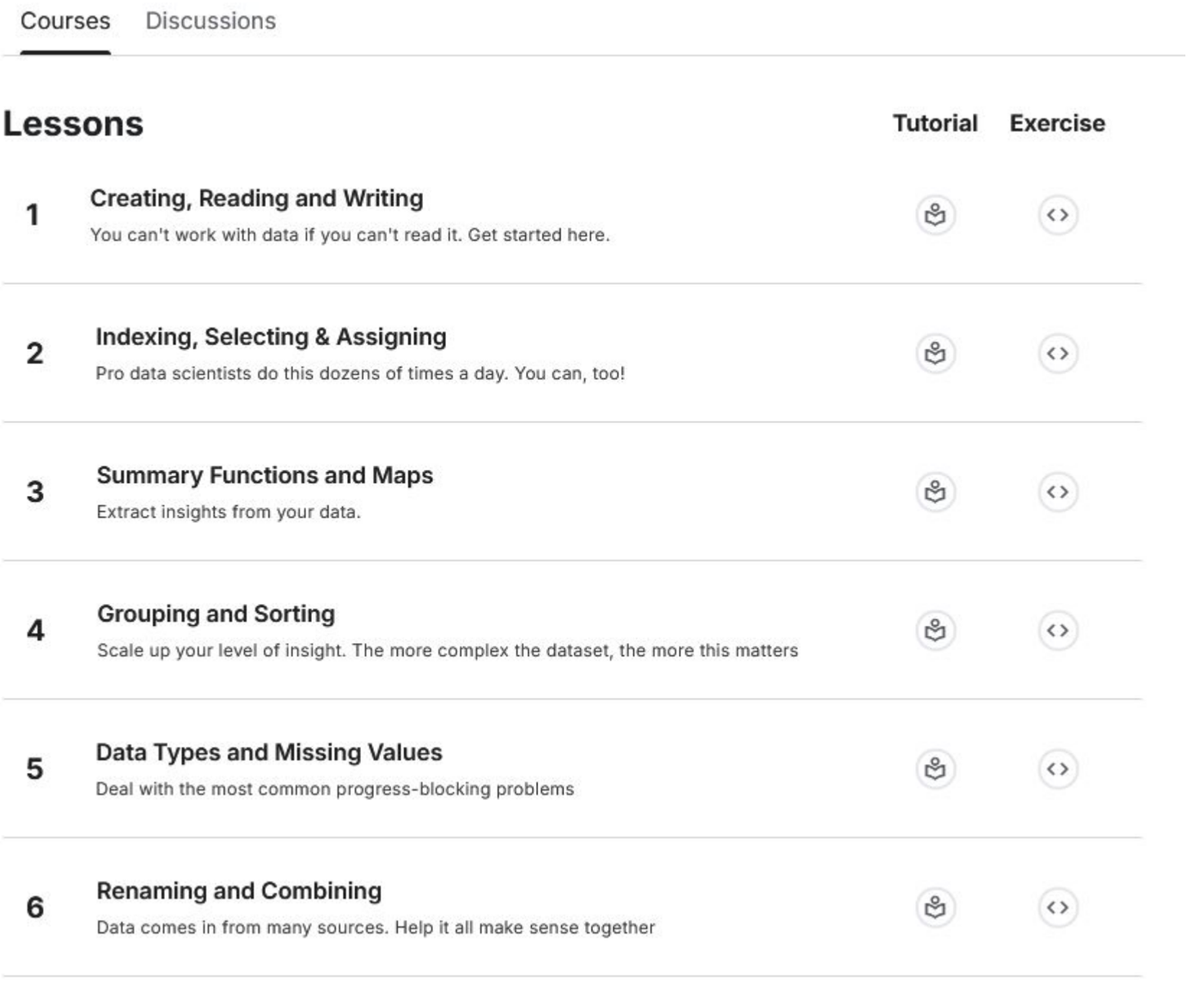
https://www.kaggle.com/learn/python



Pandas

Solve short hands-on challenges to perfect your data manipulation skills..

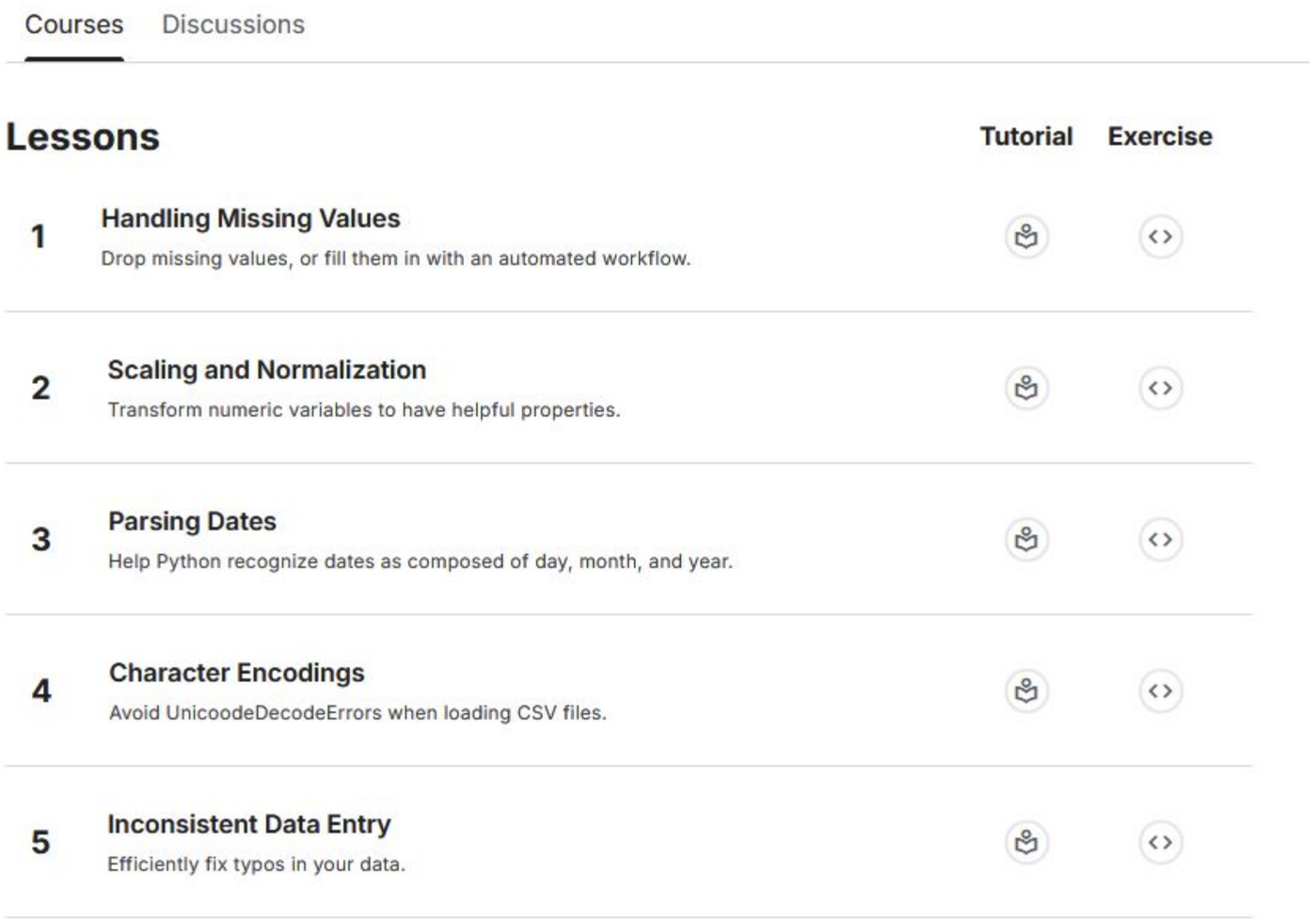
https://www.kaggle.com/learn/pandas



Data Cleaning

Master efficient workflows for cleaning real-world, messy data.

https://www.kaggle.com/learn/data-cleaning



Python for Machine Learning

Topics

1. How to setup Google colab, Google Drive, How to load datasets into google colab, Google Colab functions, magic functions, and Typical Python ML notebook structure
2. Variables & Data structures (List, Tuple, Set, Dictionary)
3. Loops & Conditional statements
4. User-defined functions & Lambda functions
5. Numpy Arrays, Pandas Series & DataFrames, Pandas Functions, How to access Pandas data frame using indexes
6. Key Python libraries for ML – pandas, numpy, sci-kit learn, Plotly, seaborn, tqdm , OS functions
7. Streamlit & Gradio

Python & IDE Setup

What is Python

Python is a **high-level**, general-purpose programming language known for its **simplicity** and **readability**. Key characteristics include:

- **Interpreted language**: Python code is executed line by line, facilitating easier debugging and rapid development.
- **Dynamic typing**: Variables don't require explicit type declarations, enhancing code flexibility.
- **Extensive standard library**: Python comes with a comprehensive set of built-in modules and functions, reducing the need for external dependencies.
- **Cross-platform compatibility**: Python runs on various operating systems, including Windows, macOS, and Linux.
- **Object-oriented**: Python **supports object-oriented programming principles**, allowing for modular and reusable code
- Case Sensitive and Zero indexing .

Python's versatility extends beyond machine learning, with applications in web development, automation, data analysis, and scientific computing. Its combination of simplicity, power, and extensive ecosystem has made it a preferred choice for both beginners and experienced developers in the machine learning field.

Low-level languages like Assembly are closer to machine code and provide direct hardware control, while high-level languages like Python are more abstracted and easier for humans to read and write. For example, Assembly requires detailed knowledge of processor architecture and uses mnemonics like "MOV" and "ADD", whereas Python allows natural language-like syntax such as "print('Hello world')" to accomplish tasks.

Python vs. R

1. Python and R are both very popular among data scientists
2. Proficiency in concepts is more important than language choice
3. R is a little better if you're only doing statistics work
4. If you need to write broad programs, python is the clear winner

Why Python is Used in Machine Learning

1. **Rich ecosystem of libraries**: Python offers a vast array of specialized libraries for machine learning and data science, including:
   - **NumPy** for numerical computing
   - **Pandas** for data manipulation and analysis
   - **Scikit-learn** for traditional machine learning algorithms
   - **TensorFlow** and **PyTorch** for deep learning
   - **Matplotlib** and **Seaborn** for data visualization
2. **Ease of use**: Python's simple and readable syntax makes it accessible to beginners and allows for rapid prototyping of machine learning models.
3. **Flexibility**: Python supports multiple programming paradigms, including object-oriented, functional, and procedural styles, making it adaptable to various machine learning approaches.
4. **Community support**: A large and active community contributes to Python's extensive documentation, tutorials, and resources for machine learning practitioners.
5. **Integration capabilities**: Python easily integrates with other languages and tools commonly used in data science and machine learning workflows.
6. **Performance optimization**: While Python is interpreted, its scientific computing libraries are often implemented in C or Fortran, providing high performance for computationally intensive tasks.

Programming Environment – IDE

What is an IDE?

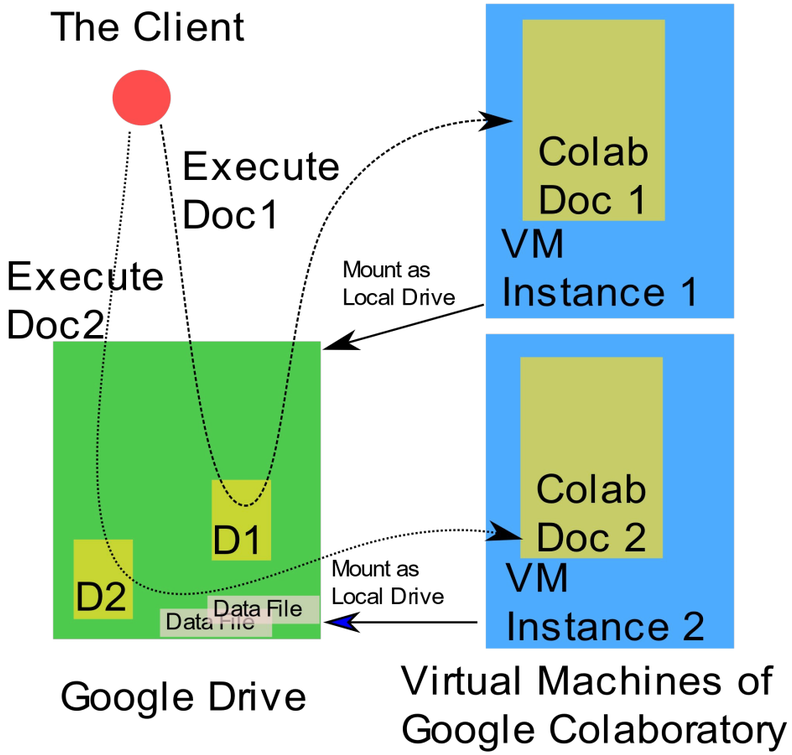
An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE typically consists of:

- **Source Code Editor:** Where you write your code.
- **Build Automation Tools:** To compile and run your code.
- **Debugger:** To test and debug your code.
- **Other Features:** Version control, syntax highlighting, and autocompletion.

Feature	Google Colab	Anaconda Jupyter	Spyder
Platform	Cloud-based	Desktop (Windows, macOS, Linux)	Desktop (Windows, macOS, Linux)
Installation Required	No (Access via web browser)	Yes (Comes with Anaconda distribution)	Yes (Comes with Anaconda distribution)
Primary Language Support	Python	Python, R	Python
Collaboration	Excellent (Real-time collaboration, sharing)	Limited (Notebook sharing via files)	Limited (Project sharing via files)
Resource Management	Google-hosted resources, limited by quotas	Local machine resources	Local machine resources
Ease of Setup	Very easy (Just sign in with Google account)	Moderate (Install Anaconda, configure Jupyter)	Moderate (Install Anaconda, configure Spyder)
File Management	Google Drive integration, local file upload	Local file system	Local file system
Support for GPUs	Yes (Free and paid options)	No (Local machine only)	No (Local machine only)
Debugging Tools	Basic (Print statements, runtime errors)	Basic (Can add debugging support with extensions)	Advanced (Integrated debugger)

Setting up Programming Environment – IDE

1. Google Collab – Platform we will use to write and execute python code
2. https://colab.research.google.com/
3. https://web.eecs.umich.edu/~justincj/teaching/eecs442/WI2021/colab.html
4. https://www.marqo.ai/blog/getting-started-with-google-colab-a-beginners-guide
5. https://www.geeksforgeeks.org/how-to-use-google-colab/



- **Setup a gmail address specifically for ML work : for example arup_das_ml_2024@gmail.com**
- **Once you setup this gmail you well a google drive account as well**
- **Setup a GitHub account and connect to your Gmail account**

https://cheng-lin-li.github.io/2018/04/04/Google_Colaboratory.html

Sample Python Code for Machine Learning_Notebook_1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate some sample data
np.random.seed(0)
X = np.linspace(0, 10, 100).reshape(-1, 1)
y = 2 * X + 1 + np.random.randn(100, 1)

# Create a Linear Regression instance
model = LinearRegression()

# Fit the model to the data
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Plot the results
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, y_pred, color='red', label='Linear regression')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()

# Print the model coefficients
print(f"Slope (coefficient): {model.coef_[0][0]:.2f}")
print(f"Intercept: {model.intercept_[0]:.2f}")
```

Connecting Google Colab notebook to Google Drive to load data for ML _Notebook2

Step 1: Mount Google Drive

```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

Explanation: Mount Google Drive: The drive.mount() function mounts your Google Drive to the Colab environment. When you run this cell, you will be prompted to authorize access to your Google Drive account. Once authorized, your Google Drive will be accessible under the /content/drive directory.

Step 2: Access Files in Google Drive

```
import os

# List files in a specific directory in your Google Drive
directory = '/content/drive/My Drive'
files = os.listdir(directory)
print(files)

# Read a file from Google Drive
file_path = '/content/drive/My Drive/your_file.txt'  # Replace with your file's path

with open(file_path, 'r') as file:
    content = file.read()
    print(content)
```

After mounting, you can access your Google Drive files just like any other directory.

Connecting Google Colab notebook to Google Drive to load data for ML

Step 3: Save Files to Google Drive

```
# Save a file to Google Drive
save_path = '/content/drive/My Drive/your_file.txt'  # Replace with your
desired file path
with open(save_path, 'w') as file:
    file.write('Hello, Google Drive!')
```

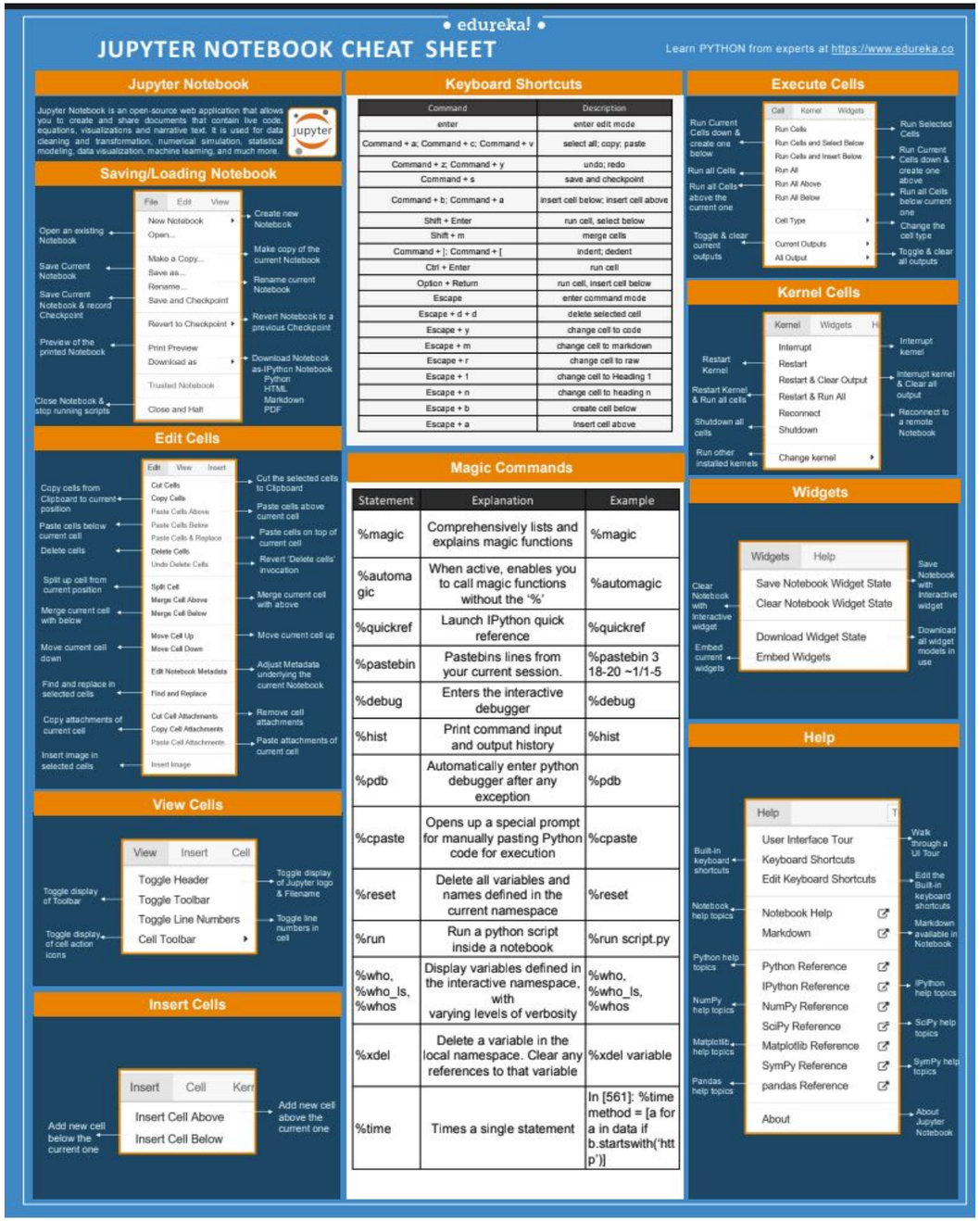
Explanation: Mount Google Drive: The drive.mount() function mounts your Google Drive to the Colab environment. When you run this cell, you will be prompted to authorize access to your Google Drive account. Once authorized, your Google Drive will be accessible under the /content/drive directory.

Summary:

- **Mount Google Drive:** This step connects your Google Drive to the Colab environment.
- **Access Files:** You can access files in Google Drive using standard file operations.
- **Save Files:** Similarly, you can save files directly to Google Drive.

Introduction to Python Notebooks

Notebook Cheat Sheet - pdf in lecture folder



edureka!

JUPYTER NOTEBOOK CHEAT SHEET

Learn PYTHON from experts at https://www.edureka.co

Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. It is used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Saving/Loading Notebook

File Edit View

- New Notebook — Create new Notebook
- Open... — Open an existing Notebook
- Make a Copy... — Make copy of the current Notebook
- Save as... — Save Current Notebook
- Rename... — Rename current Notebook
- Save and Checkpoint — Save Current Notebook & record Checkpoint
- Revert to Checkpoint — Revert Notebook to a previous Checkpoint
- Print Preview — Preview of the printed Notebook
- Download as — Download Notebook as-IPython Notebook, Python, HTML, Markdown, PDF
- Trusted Notebook
- Close and Halt — Close Notebook & stop running scripts

Edit Cells

Edit View Insert

- Cut Cells — Cut the selected cells to Clipboard
- Copy Cells — Copy cells from Clipboard to current position
- Paste Cells Above — Paste cells above current cell
- Paste Cells Below — Paste cells below current cell
- Paste Cells & Replace — Paste cells on top of current cell
- Delete Cells — Delete cells
- Undo Delete Cells — Revert 'Delete cells' invocation
- Split Cell — Split up cell from current position
- Merge Cell Above — Merge current cell with above
- Merge Cell Below — Merge current cell with below
- Move Cell Up — Move current cell up
- Move Cell Down — Move current cell down
- Edit Notebook Metadata — Adjust Metadata underlying the current Notebook
- Find and Replace — Find and replace in selected cells
- Cut Cell Attachments — Remove cell attachments
- Copy Cell Attachments — Copy attachments of current cell
- Paste Cell Attachments — Paste attachments of current cell
- Insert Image — Insert image in selected cells

View Cells

View Insert Cell

- Toggle Header — Toggle display of Jupyter logo & Filename
- Toggle Toolbar — Toggle display of Toolbar
- Toggle Line Numbers — Toggle line numbers in cell
- Cell Toolbar — Toggle display of cell action icons

Insert Cells

Insert Cell Kern

- Insert Cell Above — Add new cell above the current one
- Insert Cell Below — Add new cell below the current one

Keyboard Shortcuts

Command	Description
enter	enter edit mode
Command + a; Command + c; Command + v	select all; copy; paste
Command + z; Command + y	undo; redo
Command + s	save and checkpoint
Command + b; Command + a	insert cell below; insert cell above
Shift + Enter	run cell, select below
Shift + m	merge cells
Command +]; Command + [	indent; dedent
Ctrl + Enter	run cell
Option + Return	run cell, insert cell below
Escape	enter command mode
Escape + d + d	delete selected cell
Escape + y	change cell to code
Escape + m	change cell to markdown
Escape + r	change cell to raw
Escape + 1	change cell to Heading 1
Escape + n	change cell to heading n
Escape + b	create cell below
Escape + a	insert cell above

Magic Commands

Statement	Explanation	Example
%magic	Comprehensively lists and explains magic functions	%magic
%automagic	When active, enables you to call magic functions without the '%'	%automagic
%quickref	Launch IPython quick reference	%quickref
%pastebin	Pastebins lines from your current session.	%pastebin 3 18-20 ~1/1-5
%debug	Enters the interactive debugger	%debug
%hist	Print command input and output history	%hist
%pdb	Automatically enter python debugger after any exception	%pdb
%cpaste	Opens up a special prompt for manually pasting Python code for execution	%cpaste
%reset	Delete all variables and names defined in the current namespace	%reset
%run	Run a python script inside a notebook	%run script.py
%who, %who_ls, %whos	Display variables defined in the interactive namespace, with varying levels of verbosity	%who, %who_ls, %whos
%xdel	Delete a variable in the local namespace. Clear any references to that variable	%xdel variable
%time	Times a single statement	In [561]: %time method = [a for a in data if b.startswith('http')]

Execute Cells

Cell Kernel Widgets

- Run Cells
- Run Cells and Select Below
- Run Cells and Insert Below
- Run All
- Run All Above
- Run All Below
- Cell Type
- Current Outputs
- All Output

Run Current Cells down & create one below — Run Selected Cells

Run all Cells — Run Current Cells down & create one above

Run all Cells above the current one — Run all Cells below current one

Toggle & clear current outputs — Change the cell type

Toggle & clear all outputs

Kernel Cells

Kernel Widgets

- Interrupt
- Restart
- Restart & Clear Output
- Restart & Run All
- Reconnect
- Shutdown
- Change kernel

Restart Kernel — Interrupt kernel

Restart Kernel & Run all cells — Interrupt kernel & Clear all output

Shutdown all cells — Reconnect to a remote Notebook

Run other installed kernels

Widgets

Widgets Help

- Save Notebook Widget State
- Clear Notebook Widget State
- Download Widget State
- Embed Widgets

Clear Notebook with Interactive widget — Save Notebook with Interactive widget

Embed current widgets — Download all widget models in use

Help

Help

- User Interface Tour — Walk through a UI Tour
- Keyboard Shortcuts — Built-in keyboard shortcuts
- Edit Keyboard Shortcuts — Edit the Built-in keyboard shortcuts
- Notebook Help — Notebook help topics
- Markdown — Markdown available in Notebook
- Python Reference — Python help topics
- IPython Reference — IPython help topics
- NumPy Reference — NumPy help topics
- SciPy Reference — SciPy help topics
- Matplotlib Reference — Matplotlib help topics
- SymPy Reference — SymPy help topics
- pandas Reference — Pandas help topics
- About — About Jupyter Notebook

https://www.tutorialspoint.com/google_colab/google_colab_tutorial.pdf

https://www.datacamp.com/tutorial/tutorial-google-colab-for-data-scientists

https://www.datacamp.com/cheat-sheet

Google Colab - Magic and OS commands (UNIX Commands)

Google Colab magic commands are special commands that enhance the functionality of Jupyter notebooks by allowing users to perform various tasks more efficiently. These commands are specific to the IPython kernel, which is used by Jupyter and Google Colab. They come in two types: line magics and cell magics.

https://www.tutorialspoint.com/google_colab/google_colab_magics.htm

Google Colab provides several ways to execute system commands and interact with the underlying operating system. Here are some key methods for running OS commands in Google Colab:

Using the Exclamation Mark (!)

The simplest way to run a single system command is by prefixing it with an exclamation mark:

```
!ls
!pwd
```

https://www.tutorialspoint.com/google_colab/google_colab_invoking_system_commands.htm

Google Colab - Importing Data

Cheatography

Importing Data in Python I Cheat Sheet
by issambd via cheatography.com/88527/cs/20287/

Importing Text Files I

`open(file_name, 'r')`	open the file
`file.read()`	read the file
`file.close()`	close the file
`file.closed()`	check if the file is closed

It is a good practice to close the file after reading it when using 'open'

Importing Text Files II

`with open(file_name) as file :`	open the file
`file.read()`	read the file
file.readline()	read line by line

When using the 'with' statement there is no need to close the file

Importing Flat Files with Numpy I

`import numpy as np`	import numpy
`np.loadtxt(file_name, delimiter= ' ')`	importing the file
`skiprows=1`	argument to skip a specific row
`usecols=[0, 2]`	argument to only show specific columns
`dtype = str'	argument to import the data as string

loadtxt only works with numeric data

Importing Flat Files with Numpy II

`import numpy as np`	import numpy
`np.recfromcsv(file, delimiter=",", names=True, dtype=None)`	open the file
`np.genfromtxt(file, delimiter=',', names=True, dtype=None)`	open the file

with the functions recfromcsv() and genfromtxt() we are able to import data with different types

Importing Stata Files

`import pandas as pd`	importing pandas
`df = pd.read_stata('disarea.dta')`	reading the stata file

Importing Flat Files With Pandas

`import pandas as pd`	import pandas
`pd.read_csv(file)`	open csv file
`nrows=5`	argument for the number of rows to load
`header=None`	argument for no header
`sep='\t'`	argument to set delimiter
`comment='#'`	argument takes characters that comments occur after in the file
`na_values='Nothing'`	argument to recognize a string as a NaN Value

Import pickled files

`import pickle`	import the library
`with open(file_name, 'rb') as file :`	open file
`pickle.load(file)`	read file

Importing Spreadsheet Files

`import pandas as pd`	importing pandas
`pd.ExcelFile(file)`	opening the file
`xl.sheet_names`	exporting the sheet names
`xl.parse(sheet_name/index)`	loading a sheet to a dataframe
`skiprows=[index]`	skipping a specific row
`names=[List of Names]`	naming the sheet's columns
`usecols=[0,]`	parse spesific columns

skiprows, names and useclos are all arguments of the function parse()

Importing SAS Files

`from sas7bdat import SAS7BDAT`	importing sas7bdat library
`import pandas as pd`	importing pandas
`with SAS7BDAT('file_name') as file:`	opening the file
`file.to_data_frame()`	loading the file as dataframe

Introduction to Python Data Types- Notebook 3

Python Variables

What is a Python Variable?

In Python, a variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name. Variables in Python are created when you assign a value to a name using the = operator.

```
x = 10
name = "Alice"
```

In the example above, x is a variable that stores the integer value 10, and the name is a variable that stores the string value "Alice." Variables in Python do not need an explicit declaration to reserve memory space; the declaration happens automatically when you assign a value to a variable.

Python Main Data Types

Here's a table explaining variables and the four main data types in Python: list, tuple, set, and dictionary.

Concept	Description
Variables	Variables in Python are used to store data that can be referenced and manipulated later in the program. A variable is created when you assign a value to it using the `=` operator.
List	A list is an ordered, mutable collection of items in Python. Lists allow duplicate elements and can store elements of different data types. Lists are defined using square brackets `[ ]`. **Example:** `my_list = [1, 2, 3, 'apple']`
Tuple	A tuple is an ordered, immutable collection of items in Python. Once a tuple is created, its elements cannot be changed. Like lists, tuples can store elements of different data types, but they are defined using parentheses `( )`. **Example:** `my_tuple = (1, 2, 3, 'apple')`
Set	A set is an unordered, mutable collection of unique items in Python. Sets do not allow duplicate elements, and their elements can be of different data types. Sets are defined using curly braces `{ }`. **Example:** `my_set = {1, 2, 3, 'apple'}`
Dictionary	A dictionary is an unordered, mutable collection of key-value pairs in Python. Each key must be unique, and each key is associated with a value. Dictionaries are defined using curly braces `{ }`, with keys and values separated by a colon `:`. **Example:** `my_dict = {'name': 'Alice', 'age': 25}`

Types of Python Variables

Variable Type	Description	Example	Explanation
Integer (int)	Stores whole numbers without a fractional component	x = 10	x stores the integer value 10
Float (float)	Stores floating-point numbers, which are numbers with a decimal point	y = 3.14	y stores the floating-point number 3.14
String (str)	Stores sequences of characters (text)	name = "Alice"	name stores the string "Alice"
Boolean (bool)	Stores one of two values: True or False	is_active = True	is_active stores the boolean value True
List (list)	Stores an ordered collection of items that can be of different types	numbers = [1, 2, 3, 4, 5]	numbers stores a list of integers
Tuple (tuple)	Stores an ordered collection of items that cannot be modified (immutable)	coordinates = (10, 20)	coordinates stores a tuple with two integers
Dictionary (dict)	Stores a collection of key-value pairs	student = {"name": "Alice", "age": 21}	student stores a dictionary with keys "name" and "age"
Set (set)	Stores an unordered collection of unique items	unique_numbers = {1, 2, 3, 4, 5}	unique_numbers stores a set of integers, each number appearing only once
NoneType (None)	Represents the absence of a value or a null value	result = None	result stores the value None, which is used to indicate no value or null
Complex (complex)	Stores complex numbers, which have a real and an imaginary part	z = 2 + 3j	z stores a complex number where 2 is the real part and 3j is the imaginary part

Summary of Python Variables

Summary of Python Variable Types

- **Integer** (`int`): For storing whole numbers without a decimal point.
- **Float** (`float`): For storing numbers with a decimal point.
- **String** (`str`): For storing text.
- **Boolean** (`bool`): For storing `True` or `False` values.
- **List** (`list`): For storing ordered collections of items that can be changed (mutable).
- **Tuple** (`tuple`): For storing ordered collections of items that cannot be changed (immutable).
- **Dictionary** (`dict`): For storing key-value pairs, where each key is unique.
- **Set** (`set`): For storing unordered collections of unique items.
- **NoneType** (`None`): For representing the absence of a value.
- **Complex** (`complex`): For storing complex numbers with real and imaginary parts.

Python Variables

https://www.geeksforgeeks.org/python-variables/

0 Indexing in Python

In Python, and many other programming languages, **0-indexing** is a way of numbering the elements in a sequence (like a list, tuple, or string) starting from 0 instead of 1.

What is 0-Indexing?

When we say that Python uses 0-indexing, it means that the first element of a sequence has an index of 0, the second element has an index of 1, the third element has an index of 2, and so on.

Why Use 0-Indexing?

1. **Historical Reasons**: 0-indexing has its roots in the way computer memory is addressed. The offset from the start of an array or list is 0 for the first element, 1 for the second element, etc. This is a concept that has been carried over from low-level languages like C.
2. **Consistency with Slices**: When you slice a sequence in Python, the start index is inclusive, and the end index is exclusive. For example, `my_list[1:3]` includes elements at indices 1 and 2 but not 3. With 0-indexing, this slicing behavior becomes consistent and intuitive.

```
my_list = ['apple', 'banana', 'cherry', 'date']
```

Here's how the elements are indexed:

Element	Index
'apple'	0
'banana'	1
'cherry'	2
'date'	3

Why is 0-Indexing Important?

Understanding 0-indexing is crucial because:

1. **Prevents Off-by-One Errors**: These are common mistakes in programming where the code accidentally refers to the wrong element due to incorrect indexing.
2. **Helps in Looping and Slicing**: When iterating over lists or creating slices, knowing that indexing starts at 0 ensures that your loops and slices behave as expected.

In summary, 0-indexing is a foundational concept in Python that affects how you interact with sequences like lists, tuples, and strings.

If you want to access the first element, you would write:

```
print(my_list[0])  # Output: 'apple'
```

to access the third element:

```
print(my_list[2])  # Output: 'cherry'
```

Accessing the Last Element

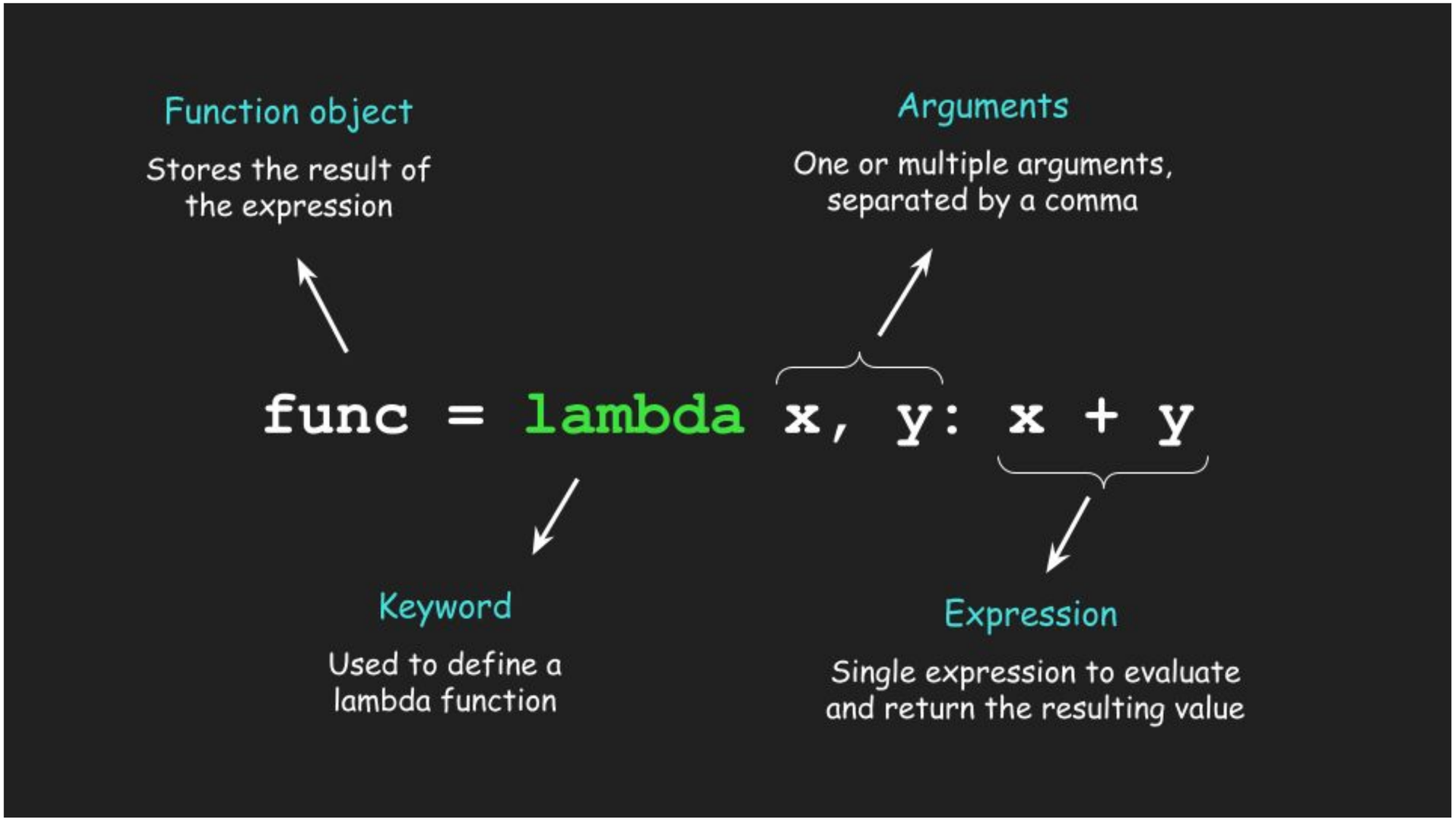
You can access the last element of a sequence using a negative index, where `-1` refers to the last element, `-2` to the second-to-last, and so on.

```
print(my_list[-1])  # Output: 'date'
print(my_list[-2])  # Output: 'cherry'
```

Looping & Conditional Statements- Notebook 4

Functions (User Defined and Lambda)- Notebook 5

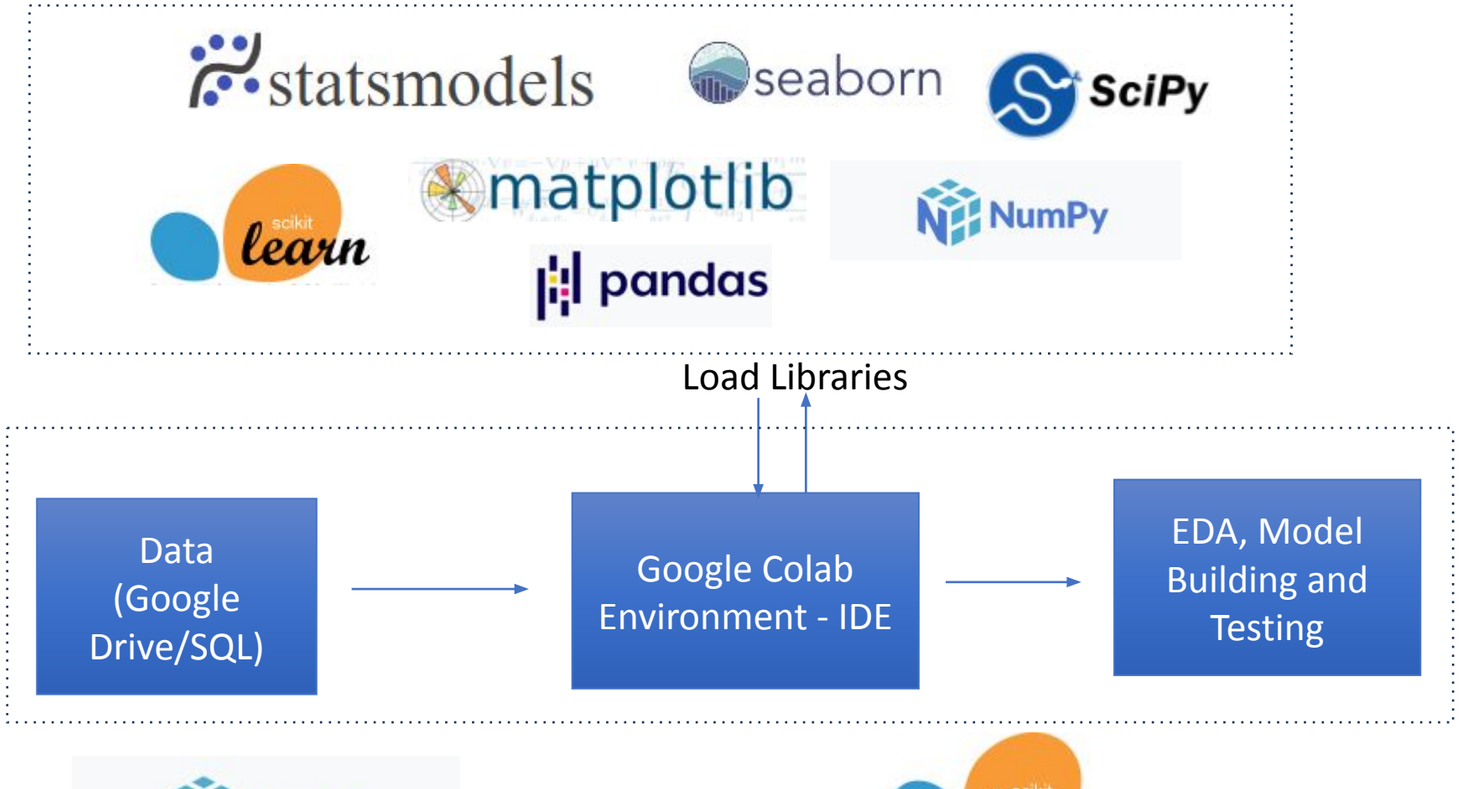
Lambda Functions



OS Module - Notebook 6

Python for Data Sciences_Notebook 7

ML Libraries



NumPy: NumPy stands for Numerical Python and it is a core scientific computing library in Python. It provides efficient multi-dimensional array objects and various operations to work with these array objects.

Scikit-Learn: Also known as Sklearn provides advanced analytics tools combined with complex machine learning capabilities. This allows you to build more sophisticated models, performing more complex and multivariate regressions, as well as data preprocessing.

Pandas: It offers a plenty of tools to manipulate, analyze, and even represent data structures and complex datasets. This includes time series and more complex data structures such as merging, pivoting, and slicing tables to create new views and perspectives on existing sets.

Matplotlib: It is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Seaborn: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Videos to Watch

Learn Python - Full Course for Beginners [Tutorial]

https://www.youtube.com/watch?v=rfscVS0vtbw&list=PLWKjhJtqVAbnupwRFOq9zGOWjdvPRtCmO

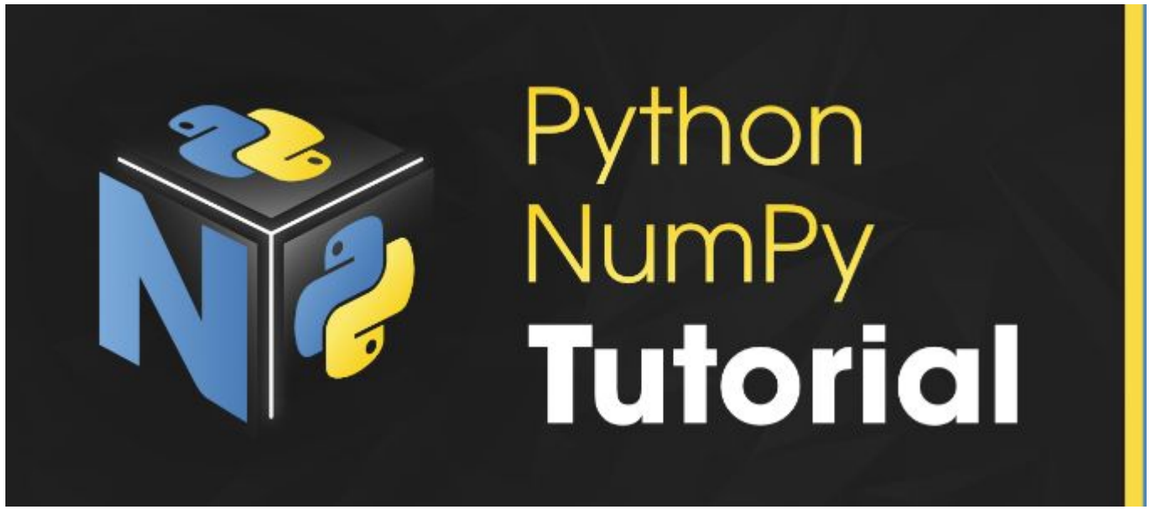
Python for Beginners – Full Course [Programming Tutorial]

https://www.youtube.com/watch?v=eWRfhZUzrAc&list=PLWKjhJtqVAbnqBxcdjVGgT3uVR10bzTEB

Numpy Tutorial

NumPy is a general-purpose array-processing Python library which provides handy methods/functions for working **n-dimensional arrays**. NumPy is a short form for "**Numerical Python**". It provides various computing tools such as comprehensive mathematical functions, and linear algebra routines.

- NumPy provides both the **flexibility of Python** and the speed of **well-optimized** compiled C code.
- Its **easy-to-use syntax** makes it highly accessible and productive for programmers from any background.



https://www.geeksforgeeks.org/numpy-tutorial/

Pandas Tutorial

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

https://www.geeksforgeeks.org/pandas-tutorial/

Cheatsheets - Uploaded into Lecture Folder

Data Science Cheat Sheet

NumPy

KEY

We'll use shorthand in this cheat sheet

`arr` - A numpy Array object

IMPORTS

Import these to start

```
import numpy as np
```

IMPORTING/EXPORTING

- `np.loadtxt('file.txt')` - From a text file
- `np.genfromtxt('file.csv',delimiter=',')` - From a CSV file
- `np.savetxt('file.txt',arr,delimiter=' ')` - Writes to a text file
- `np.savetxt('file.csv',arr,delimiter=',')` - Writes to a CSV file

CREATING ARRAYS

- `np.array([1,2,3])` - One dimensional array
- `np.array([(1,2,3),(4,5,6)])` - Two dimensional array
- `np.zeros(3)` - 1D array of length 3 all values 0
- `np.ones((3,4))` - 3x4 array with all values 1
- `np.eye(5)` - 5x5 array of 0 with 1 on diagonal (Identity matrix)
- `np.linspace(0,100,6)` - Array of 6 evenly divided values from 0 to 100
- `np.arange(0,10,3)` - Array of values from 0 to less than 10 with step 3 (eg `[0,3,6,9]`)
- `np.full((2,3),8)` - 2x3 array with all values 8
- `np.random.rand(4,5)` - 4x5 array of random floats between 0-1
- `np.random.rand(6,7)*100` - 6x7 array of random floats between 0-100
- `np.random.randint(5,size=(2,3))` - 2x3 array with random ints between 0-4

INSPECTING PROPERTIES

- `arr.size` - Returns number of elements in `arr`
- `arr.shape` - Returns dimensions of `arr` (rows, columns)
- `arr.dtype` - Returns type of elements in `arr`
- `arr.astype(dtype)` - Convert `arr` elements to type `dtype`
- `arr.tolist()` - Convert `arr` to a Python list
- `np.info(np.eye)` - View documentation for `np.eye`

COPYING/SORTING/RESHAPING

- `np.copy(arr)` - Copies `arr` to new memory
- `arr.view(dtype)` - Creates view of `arr` elements with type `dtype`
- `arr.sort()` - Sorts `arr`
- `arr.sort(axis=0)` - Sorts specific axis of `arr`
- `two_d_arr.flatten()` - Flattens 2D array `two_d_arr` to 1D
- `arr.T` - Transposes `arr` (rows become columns and vice versa)
- `arr.reshape(3,4)` - Reshapes `arr` to 3 rows, 4 columns without changing data
- `arr.resize((5,6))` - Changes `arr` shape to 5x6 and fills new values with 0

ADDING/REMOVING ELEMENTS

- `np.append(arr,values)` - Appends `values` to end of `arr`
- `np.insert(arr,2,values)` - Inserts `values` into `arr` before index 2
- `np.delete(arr,3,axis=0)` - Deletes row on index 3 of `arr`
- `np.delete(arr,4,axis=1)` - Deletes column on index 4 of `arr`

COMBINING/SPLITTING

- `np.concatenate((arr1,arr2),axis=0)` - Adds `arr2` as rows to the end of `arr1`
- `np.concatenate((arr1,arr2),axis=1)` - Adds `arr2` as columns to end of `arr1`
- `np.split(arr,3)` - Splits `arr` into 3 sub-arrays
- `np.hsplit(arr,5)` - Splits `arr` horizontally on the 5th index

INDEXING/SLICING/SUBSETTING

- `arr[5]` - Returns the element at index 5
- `arr[2,5]` - Returns the 2D array element on index `[2][5]`
- `arr[1]=4` - Assigns array element on index 1 the value 4
- `arr[1,3]=10` - Assigns array element on index `[1][3]` the value 10
- `arr[0:3]` - Returns the elements at indices `0,1,2` (On a 2D array: returns rows `0,1,2`)
- `arr[0:3,4]` - Returns the elements on rows `0,1,2` at column 4
- `arr[:2]` - Returns the elements at indices `0,1` (On a 2D array: returns rows `0,1`)
- `arr[:,1]` - Returns the elements at index 1 on all rows
- `arr<5` - Returns an array with boolean values
- `(arr1<3) & (arr2>5)` - Returns an array with boolean values
- `~arr` - Inverts a boolean array
- `arr[arr<5]` - Returns array elements smaller than 5

SCALAR MATH

- `np.add(arr,1)` - Add 1 to each array element
- `np.subtract(arr,2)` - Subtract 2 from each array element
- `np.multiply(arr,3)` - Multiply each array element by 3
- `np.divide(arr,4)` - Divide each array element by 4 (returns `np.nan` for division by zero)
- `np.power(arr,5)` - Raise each array element to the 5th power

VECTOR MATH

- `np.add(arr1,arr2)` - Elementwise add `arr2` to `arr1`
- `np.subtract(arr1,arr2)` - Elementwise subtract `arr2` from `arr1`
- `np.multiply(arr1,arr2)` - Elementwise multiply `arr1` by `arr2`
- `np.divide(arr1,arr2)` - Elementwise divide `arr1` by `arr2`
- `np.power(arr1,arr2)` - Elementwise raise `arr1` raised to the power of `arr2`
- `np.array_equal(arr1,arr2)` - Returns `True` if the arrays have the same elements and shape
- `np.sqrt(arr)` - Square root of each element in the array
- `np.sin(arr)` - Sine of each element in the array
- `np.log(arr)` - Natural log of each element in the array
- `np.abs(arr)` - Absolute value of each element in the array
- `np.ceil(arr)` - Rounds up to the nearest int
- `np.floor(arr)` - Rounds down to the nearest int
- `np.round(arr)` - Rounds to the nearest int

STATISTICS

- `np.mean(arr,axis=0)` - Returns mean along specific axis
- `arr.sum()` - Returns sum of `arr`
- `arr.min()` - Returns minimum value of `arr`
- `arr.max(axis=0)` - Returns maximum value of specific axis
- `np.var(arr)` - Returns the variance of array
- `np.std(arr,axis=1)` - Returns the standard deviation of specific axis
- `arr.corrcoef()` - Returns correlation coefficient of array

Data Analysis with PANDAS

CHEAT SHEET

Created By: Arianne Colton and Sean Chen

DATA STRUCTURES

SERIES (1D)

One-dimensional array-like object containing an array of data (of any **NumPy** data type) and an associated array of data labels, called its **"index"**. If index of data is not specified, then a default one consisting of the integers 0 through N-1 is created.

Create Series	`series1 = pd.Series ([1, 2], index = ['a', 'b'])` `series1 = pd.Series(dict1)` *
Get Series Values	`series1.values`
Get Values by Index	`series1['a']` `series1[['b','a']]`
Get Series Index	`series1.index`
Get Name Attribute (None is default)	`series1.name` `series1.index.name`
** Common Index Values are Added	`series1 + series2`
Unique But Unsorted	`series2 = series1.unique()`

* Can think of Series as a fixed-length, ordered dict. Series can be substituted into many functions that expect a dict.

** Auto-align differently-indexed data in arithmetic operations

DATAFRAME (2D)

Tabular data structure with ordered collections of columns, each of which can be different value type.
Data Frame (DF) can be thought of as a dict of Series.

Create DF
(from a dict of equal-length lists or NumPy arrays)
	`dict1 = {'state': ['Ohio', 'CA'], 'year': [2000, 2010]}`
	`df1 = pd.DataFrame(dict1)`
	# columns are placed in sorted order
	`df1 = pd.DataFrame(dict1, index = ['row1', 'row2'])`
	# specifying index
	`df1 = pd.DataFrame(dict1, columns = ['year', 'state'])`
	# columns are placed in your given order

* Create DF (from nested dict of dicts) The inner keys as row indices	`dict1 = {'col1': {'row1': 1, 'row2': 2}, 'col2': {'row1': 3, 'row2': 4} }` `df1 = pd.DataFrame(dict1)`

Get Columns and Row Names	`df1.columns` `df1.index`
Get Name Attribute (None is default)	`df1.columns.name` `df1.index.name`
Get Values	`df1.values` # returns the data as a 2D ndarray, the dtype will be chosen to accomandate all of the columns
** Get Column as Series	`df1['state'] or df1.state`
** Get Row as Series	`df1.ix['row2'] or df1.ix[1]`
Assign a column that doesn't exist will create a new column	`df1['eastern'] = df1.state == 'Ohio'`
Delete a column	`del df1['eastern']`
Switch Columns and Rows	`df1.T`

* Dicts of Series are treated the same as Nested dict of dicts.

** Data returned is a 'view' on the underlying data, NOT a copy. Thus, any in-place modifications to the data will be reflected in df1.

PANEL DATA (3D)

Create Panel Data : (Each item in the Panel is a DF)

```
import pandas.datareader.data as web
panel1 = pd.Panel({stk : web.get_data_
yahoo(stk, '1/1/2000', '1/1/2010')
for stk in ['AAPL', 'IBM']})
# panel1 Dimensions : 2 (item) * 861 (major) * 6 (minor)
```

Stacked DF form : (Useful way to represent panel data)

```
panel1 = panel1.swapaxes('item', 'minor')
panel1.ix[:, '6/1/2003', :].to_frame() *
```

=> Stacked DF (with **hierarchical indexing** **) :

		Open	High	Low	Close	Volume	Adj-Close
major	minor						
2003-06-01	AAPL						
	IBM						
2003-06-02	AAPL						
	IBM						

DATA STRUCTURES CONTINUED

* DF has a "to_panel()" method which is the inverse of "to_frame()".

** Hierarchical indexing makes N-dimensional arrays unnecessary in a lot of cases. Aka prefer to use Stacked DF, not Panel data.

INDEX OBJECTS

Immutable objects that hold the axis labels and other metadata (i.e. axis name)

- i.e. Index, MultiIndex, DatetimeIndex, PeriodIndex
- Any sequence of labels used when constructing Series or DF internally converted to an Index.
- Can functions as fixed-size set in additional to being array-like.

HIERARCHICAL INDEXING

Multiple index levels on an axis : A way to work with higher dimensional data in a lower dimensional form.

MultiIndex :

```
series1 = Series(np.random.randn(6),index =
[['a', 'a', 'a', 'b', 'b', 'b'], [1, 2, 3,
1, 2, 3]])
series1.index.names = ['key1', 'key2']
```

Series Partial Indexing	`series1['b']` # Outer Level `series1[:, 2]` # Inner Level

DF Partial Indexing	`df1['outerCol3','InnerCol2']`
	Or
	`df1['outerCol3']['InnerCol2']`

Swaping and Sorting Levels

Swap Level (level interchanged) *	`swapSeries1 = series1.` `swaplevel('key1', 'key2')`
Sort Level	`series1.sortlevel(1)` # sorts according to first inner level
Common Ops : Swap and Sort **	`series1.swaplevel(0, 1).sortlevel(0)` # the order of rows also change

* The order of the rows do not change. Only the two levels got swapped.

** Data selection performance is much better if the index is sorted starting with the outermost level, as a result of calling `sortlevel(0)` or `sort_index()`.

Summary Statistics by Level

Most stats functions in DF or Series have a "level" option that you can specify the level you want on an axis.

Sum rows (that have same 'key2' value)	`df1.sum(level = 'key2')`
Sum columns ..	`df1.sum(level = 'col3', axis = 1)`

- Under the hood, the functionality provided here utilizes panda's **'groupby'**.

DataFrame's Columns as Indexes

DF's "set_index" will create a new DF using one or more of its columns as the index.

New DF using columns as index	`df2 = df1.set_index(['col3', 'col4'])` * ‡ # col3 becomes the outermost index, col4 becomes inner index. Values of col3, col4 become the index values.

* "reset_index" does the opposite of "set_index", the hierarchical index are moved into columns.

‡ By default, 'col3' and 'col4' will be removed from the DF, though you can leave them by option : `'drop = False'`.

MISSING DATA

Python	`NaN - np.nan` (not a number)
Pandas *	`NaN` or python built-in `None` mean missing/NA values

* Use `pd.isnull()`, `pd.notnull()` or `series1/df1.isnull()` to detect missing data.

FILTERING OUT MISSING DATA

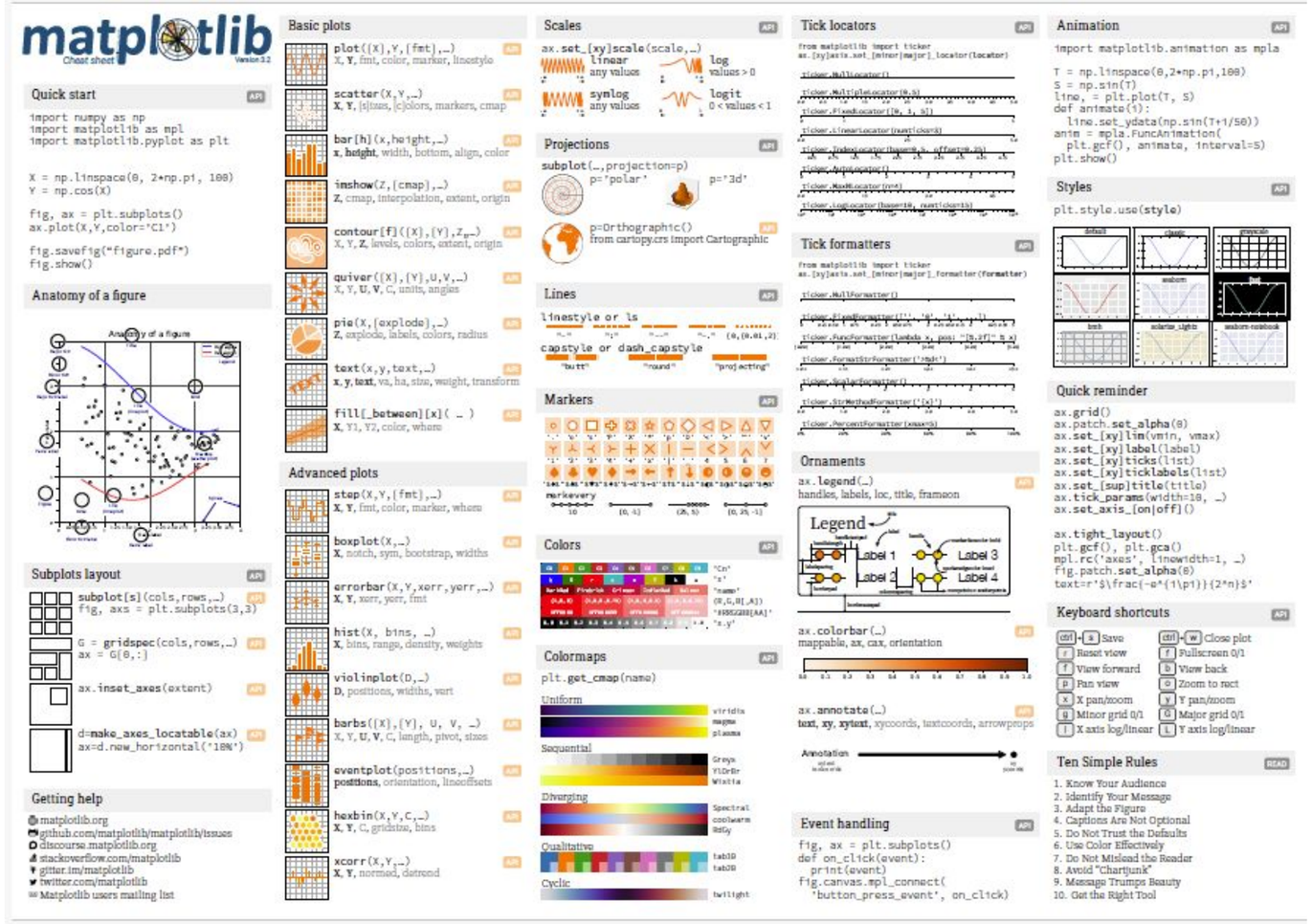
`dropna()` returns with ONLY non-null data, source data NOT modified.

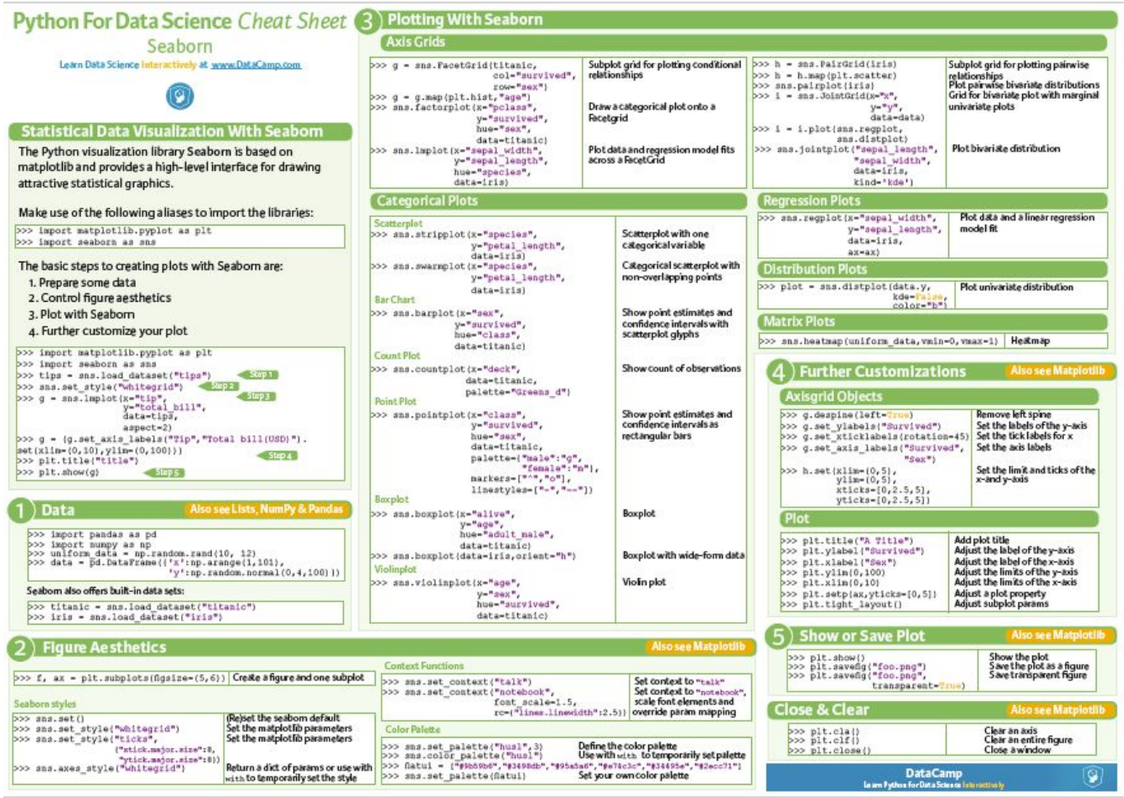
`df1.dropna()`	# drop any row containing missing value
`df1.dropna(axis = 1)` containing missing values	# drop any column
`df1.dropna(how = 'all')`	# drop row that are all missing
`df1.dropna(thresh = 3)` < 3 number of observations	# drop any row containing

FILLING IN MISSING DATA

`df2 = df1.fillna(0)`	# fill all missing data with 0
`df1.fillna(inplace = True)`	# modify in-place
Use a different fill value for each column :	
`df1.fillna({'col1' : 0, 'col2' : -1})`	
Only forward fill the 2 missing values in front :	
`df1.fillna(method = 'ffill', limit = 2)`	
i.e. for column1, if row 3-6 are missing. so 3 and 4 get filled with the value from 2, NOT 5 and 6.	

Cheatsheets - Uploaded into Lecture Folder





Cheatsheets - Uploaded into Lecture Folder

datacamp plotly

Data Visualization with Plotly Express in Python

Learn Plotly online at www.DataCamp.com

> What is plotly?

Plotly Express is a high-level data visualization package that allows you to create interactive plots with very little code. It is built on top of Plotly Graph Objects, which provides a lower-level interface for developing custom visualizations.

> Interactive controls in Plotly

Plotly plots have interactive controls shown in the top-right of the plot. The controls allow you to do the following:

- **Download plot as a png:** Save your interactive plot as a static PNG.
- **Zoom:** Zoom in on a region of interest in the plot.
- **Pan:** Move around in the plot.
- **Box Select:** Select a rectangular region of the plot to be highlighted.
- **Lasso Select:** Draw a region of the plot to be highlighted.
- **Autoscale:** Zoom to a "best" scale.
- **Reset axes:** Return the plot to its original state.
- **Toggle Spike Lines:** Show or hide lines to the axes whenever you hover over data.
- **Show closest data on hover:** Show details for the nearest data point to the cursor.
- **Compare data on hover:** Show the nearest data point to the x-coordinate of the cursor.

> Plotly Express code pattern

The code pattern for creating plots is to call the plotting function, passing a data frame as the first argument. The x argument is a string naming the column to be used on the x-axis. The y argument can either be a string or a list of strings naming column(s) to be used on the y-axis.

```
px.plotting_fn(dataframe, # Dataframe being visualized
               x=["column-for-x-axis"], # Accepts a string or a list of strings
               y=["columns-for-y-axis"], # Accepts a string or a list of strings
               title="Overall plot title", # Accepts a string
               xaxis_title="X-axis title", # Accepts a string
               yaxis_title="Y-axis title", # Accepts a string
               width=width_in_pixels, # Accepts an integer
               height=height_in_pixels) # Accepts an integer
```

> Common plot types

Import plotly

```
# import plotly express as px
import plotly.express as px
```

Scatter plots

```
# Create a scatterplot on a DataFrame named clinical_data
px.scatter(clinical_data, x="experiment_1", y="experiment_2")
```

Set the `size` argument to the name of a numeric column to control the size of the points and create a bubble plot.

Line plots

```
# Create a lineplot on a DataFramed named stock_data
px.line(stock_data, x="date", y=["FB", "AMZN"])
```

Set the `line_dash` argument to the name of a categorical column to have dashes or dots for different lines.

Bar plots

```
# Create a barplot on a DataFramed named commodity_data
px.bar(commodity_data, x="nation", y=["gold", "silver", "bronze"],
       color_discrete_map={"gold": "yellow",
                           "silver": "grey",
                           "bronze": "brown"})
```

Swap the `x` and `y` arguments to draw horizontal bars.

Histograms

```
# Create a histogram on a DataFramed named bill_data
px.histogram(bill_data, x="total_bill")
```

Set the `nbins` argument to control the number of bins shown in the histogram.

Heatmaps

```
# Create a heatmap on a DataFramed named iris_data
px.imshow(iris_data.corr(numeric_only=True),
          zmin=-1, zmax=1, color_continuous_scale='rdbu')
```

Set the `text_auto` argument to `True` to display text values for each cell.

> Customizing plots in plotly

The code pattern for customizing a plot is to save the figure object returned from the plotting function, call its `.update_traces()` method, then call its `.show()` method to display it.

```
# Create a plot with plotly (can be of any type)
fig = px.some_plotting_function()
# Customize and show it with .update_traces() and .show()
fig.update_traces()
fig.show()
```

Customizing markers in Plotly

When working with visualizations like scatter plots, lineplots, and more, you can customize markers according to certain properties. These include:

- `size`: set the marker size
- `color`: set the marker color
- `opacity`: set the marker transparency
- `line`: set the width and color of a border
- `symbol`: set the shape of the marker

```
# In this example, we're updating a scatter plot named fig_sct
fig_sct.update_traces(marker={"size": 24,
                              "color": "magenta",
                              "opacity": 0.5,
                              "line": {"width": 2, "color": "cyan"},
                              "symbol": "square"})
fig_sct.show()
```

Customizing lines in Plotly

When working with visualizations that contain lines, you can customize them according to certain properties. These include:

- `color`: set the line color
- `dash`: set the dash style (`"solid"`, `"dot"`, `"dash"`, `"longdash"`, `"dashdot"`, `"longdashdot"`)
- `shape`: set how values are connected (`"linear"`, `"spline"`, `"hv"`, `"vh"`, `"hvh"`,`"vhv"`)
- `width`: set the line width

```
# In this example, we're updating a scatter plot named fig_ln
fig_ln.update_traces(patch={"line": {"dash": "dot",
                                     "shape": "spline",
                                     "width": 6}})
fig_ln.show()
```

Customizing bars in Plotly

When working with barplots and histograms, you can update the bars themselves according to the following properties:

- `size`: set the marker size
- `color`: set the marker color
- `opacity`: set the marker transparency
- `line`: set the width and color of a border
- `symbol`: set the shape of the marker

```
# In this example, we're updating a scatter plot named fig_bar
fig_bar.update_traces(marker={"color": "magenta",
                              "opacity": 0.5,
                              "line": {"width": 2, "color": "cyan"}})
fig_bar.show()
```

```
# In this example, we're updating a histogram named fig_hst
fig_hst.update_traces(marker={"color": "magenta",
                              "opacity": 0.5,
                              "line": {"width": 2, "color": "cyan"}})
fig_hst.show()
```

Learn Data Skills Online at www.DataCamp.com

datacamp

Coding Template Best Practices

Module 1: Project Description: Write a description of the project and the problem you are trying to solve. (Text Block)

Module 2: Data dictionary. Please write down the description of all columns, including the type of variables they are – Text, Numerical, and Categorical (Text Block)

Module 3: Load all python libraries (Code block)

For example: from pandas load pd # describe each function if possible

Module 4: Load data set # data set can be loaded using api, local drive and google URL method

Load the data set into a data frame called dforiginal # this is a pandas dataframe

dforiginal = pd.read_xxxx # depends on the file type and method

Module 5: Make a copy of the original data frame for analysis and model building

df=dforiginal.copy() # the df pandas data frame is now our dataset we will work with. Note that we have kept an original copy if we make errors in the df data frame. Note that this will consume more memory

Module 6: Description and data analysis of the load data frame

1.Data types – df.dtypes

2.Data shape – df.shape

3.Data descriptions (mean, median, count, min, max) – df.describe() # only works on numerical columns .