HyperionDev

Datasets and Dataframes

August 2024

Data Science Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Data Science Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Learning Outcomes

- **Describe** the role and benefits of Python packages like Numpy and Pandas in data science tasks, including the use cases for Numpy arrays and Pandas dataframes.
- **Apply** Pandas functions to create and manipulate data frames from datasets, performing data exploration and transformation tasks.

HyperionDev

Learning Outcomes

❖ **Create** and run a Jupyter Notebook to document and present data analysis workflows, including reading, exploring, and manipulating datasets using Pandas.

Problem Statement

Data science students are often required to handle, manipulate, and analyse large datasets efficiently. However, working with raw data using basic Python structures like lists and dictionaries can be cumbersome and inefficient, especially when dealing with complex operations or large volumes of data. Additionally, managing and presenting your work can be challenging without the right tools.

- ❖ How can we read datasets into our code more efficiently, and represent them using more appropriate data structures?
- ❖ How can we explore and manipulate our datasets using built-in functions in Python?

Lecture Overview

→ Python Packages
  ↳ Numpy
  ↳ Pandas

→ Dataframes in Pandas

→ Jupyter Notebooks

HyperionDev

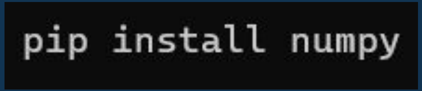
Python packages for data science

NumPy

HyperionDev

Python packages for data science: NumPy

- **NumPy** - stands for **Num**erical **Py**thon

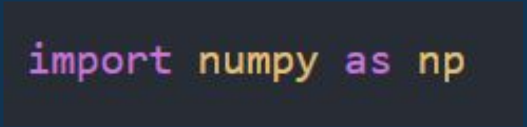


```
pip install numpy
```

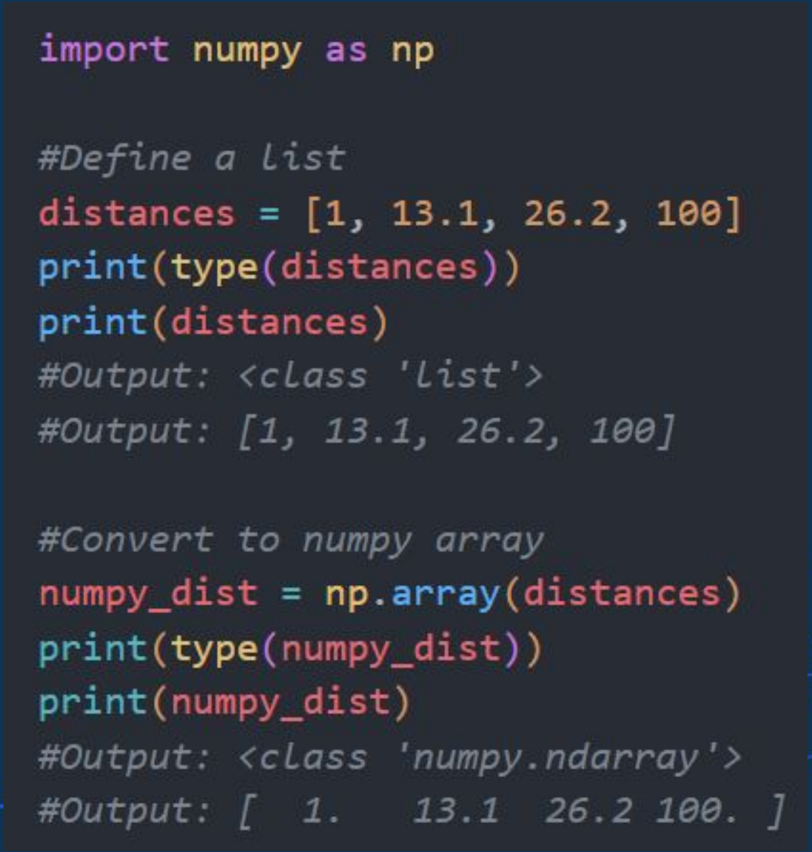
- Numpy arrays differ from python lists
  - Elements of same datatype
  - Can handle arithmetic operations
  - Preferred for larger chunks of data
  - Requires proper modules to perform operations on them

- Array oriented programming, NumPy has smaller memory consumption, better runtime, and ease of data manipulation

Using NumPy



```python
import numpy as np
```

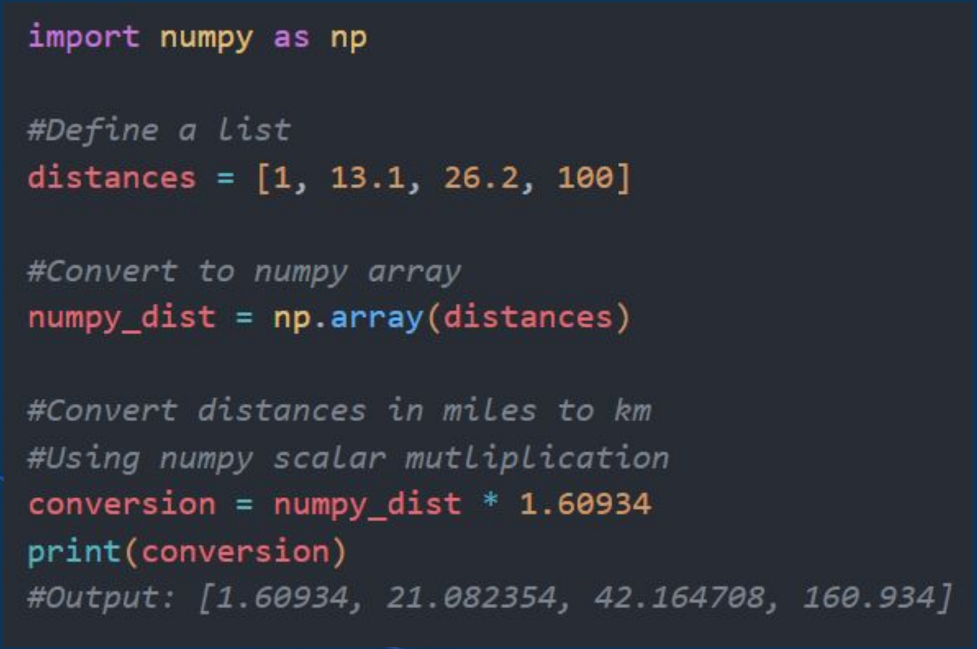


```python
import numpy as np

#Define a list
distances = [1, 13.1, 26.2, 100]
print(type(distances))
print(distances)
#Output: <class 'list'>
#Output: [1, 13.1, 26.2, 100]

#Convert to numpy array
numpy_dist = np.array(distances)
print(type(numpy_dist))
print(numpy_dist)
#Output: <class 'numpy.ndarray'>
#Output: [  1.   13.1  26.2 100. ]
```

Using NumPy

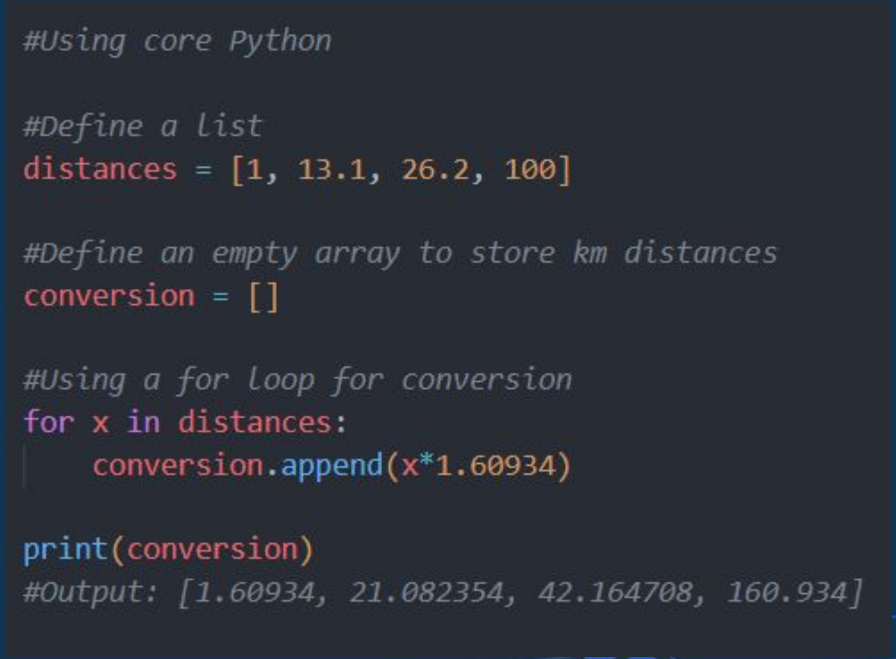


```python
import numpy as np

#Define a list
distances = [1, 13.1, 26.2, 100]

#Convert to numpy array
numpy_dist = np.array(distances)

#Convert distances in miles to km
#Using numpy scalar mutliplication
conversion = numpy_dist * 1.60934
print(conversion)
#Output: [1.60934, 21.082354, 42.164708, 160.934]
```



```python
#Using core Python

#Define a list
distances = [1, 13.1, 26.2, 100]

#Define an empty array to store km distances
conversion = []

#Using a for loop for conversion
for x in distances:
    conversion.append(x*1.60934)

print(conversion)
#Output: [1.60934, 21.082354, 42.164708, 160.934]
```

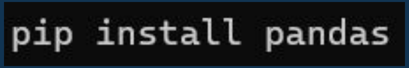
Python packages for data science

Pandas

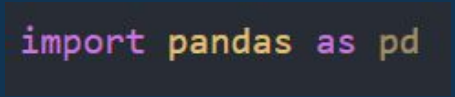
HyperionDev

Pandas

- **Pandas**, built on NumPy, is a Python module that contains high-level data structures and tools designed for fast and easy data analysis.

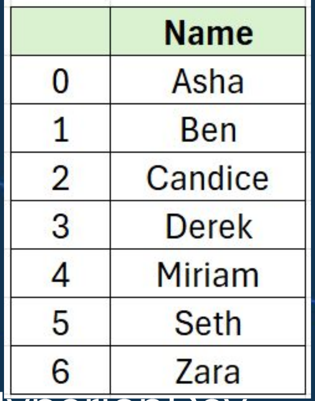


```
pip install pandas
```

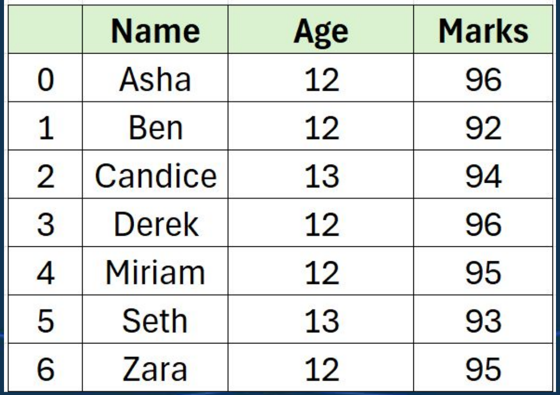


```
import pandas as pd
```

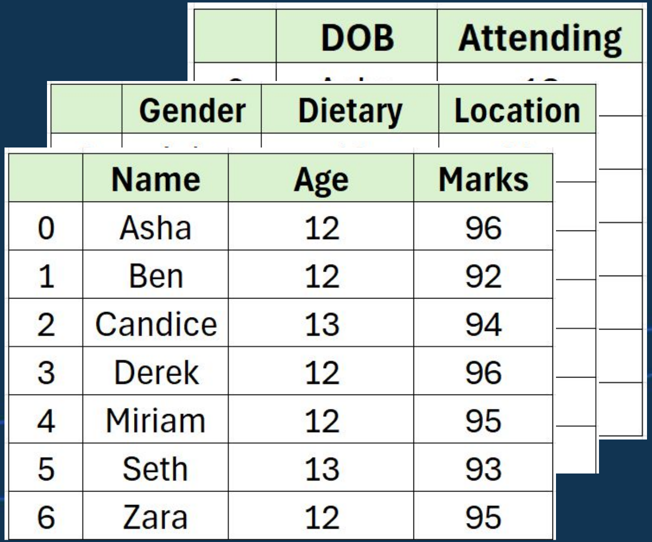
- Fundamental pandas data structures
  - Series (1-dimensional labelled array, can hold any data type)
  - DataFrame (2-dimensional)
  - Panel (pandas -> **pan**el **da**ta)



	Name
0	Asha
1	Ben
2	Candice
3	Derek
4	Miriam
5	Seth
6	Zara



	Name	Age	Marks
0	Asha	12	96
1	Ben	12	92
2	Candice	13	94
3	Derek	12	96
4	Miriam	12	95
5	Seth	13	93
6	Zara	12	95


DOB
Attending
Gender
Dietary
Location

	Name	Age	Marks
0	Asha	12	96
1	Ben	12	92
2	Candice	13	94
3	Derek	12	96
4	Miriam	12	95
5	Seth	13	93
6	Zara	12	95

Datasets

HyperionDev

Datasets

- A dataset is a structured collection of information relevant to a specific investigation or project

- In data science, they provide the raw material for analysis and modeling. Understanding different dataset formats ensures you can work with data from various sources (databases, online repositories, etc.).

- With the help of Pandas DataFrames, we can effortlessly manipulate data to suit our needs.

DataFrames

HyperionDev

DataFrames

- A DataFrame is the way the Pandas library in Python represents tabular data. It's like a powerful *spreadsheet* within your code.

- Rows: Each row represents a single observation or data point (e.g., a person, a product, a transaction).

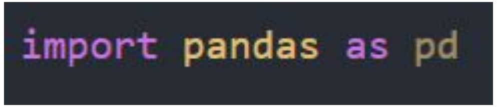
- Columns: Each column represents a variable or feature (e.g., height, price, date). Data within a column usually shares the same data type (numbers, text, etc.).

Pandas DataFrame

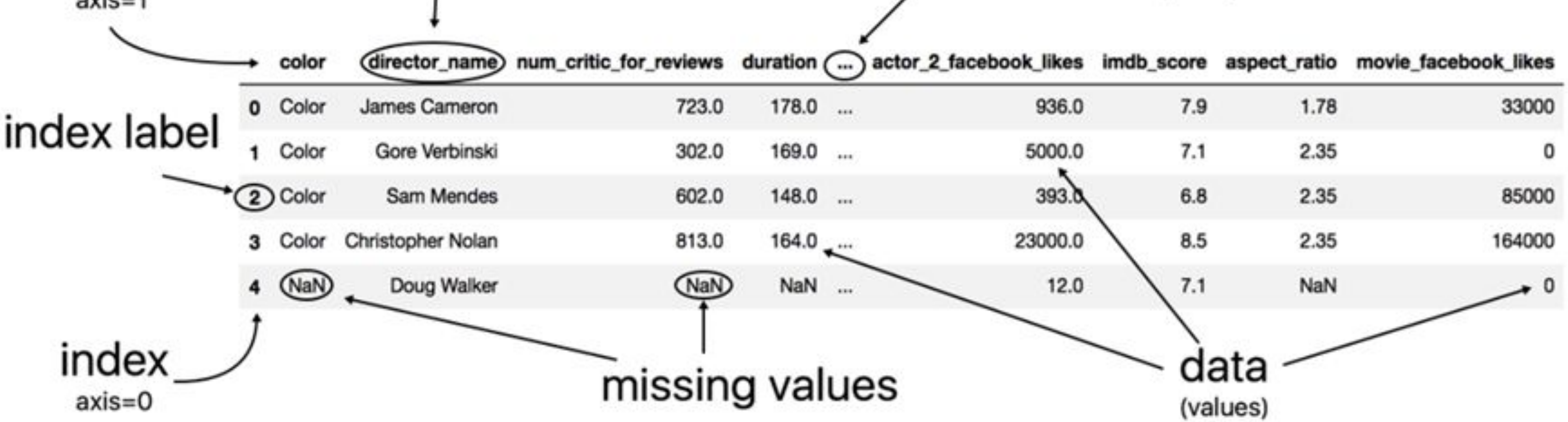
HyperionDev

Pandas DataFrame

- ❖ The pandas' library documentation defines a DataFrame as a "two-dimensional, size-mutable, with labelled rows and columns."



```
import pandas as pd
```



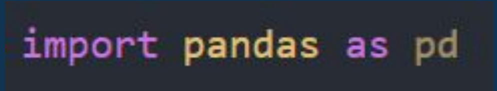
	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

Anatomy of a DataFrame

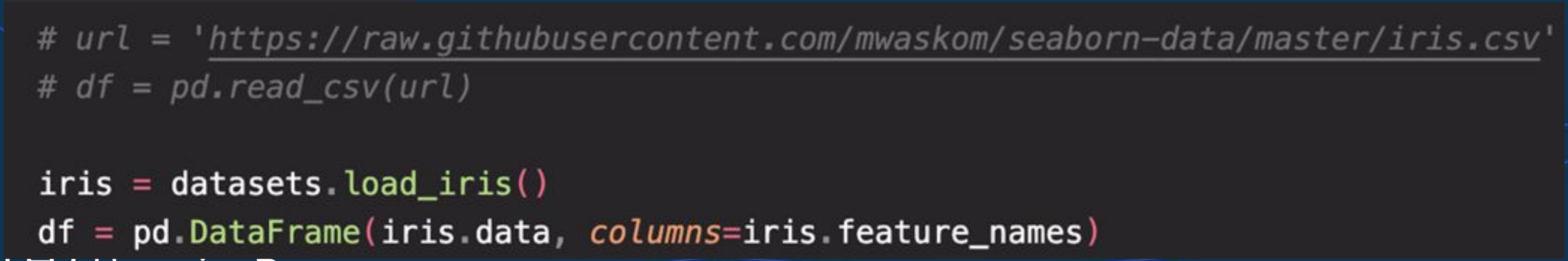
Pandas DataFrame

- Pandas provides functions like **pd.read_csv()**, **pd.read_excel(), pd.read_sql()**, to bring your data directly into your coding environment as DataFrames.

- This is where you start turning your raw data into something easily workable.



```
import pandas as pd
```

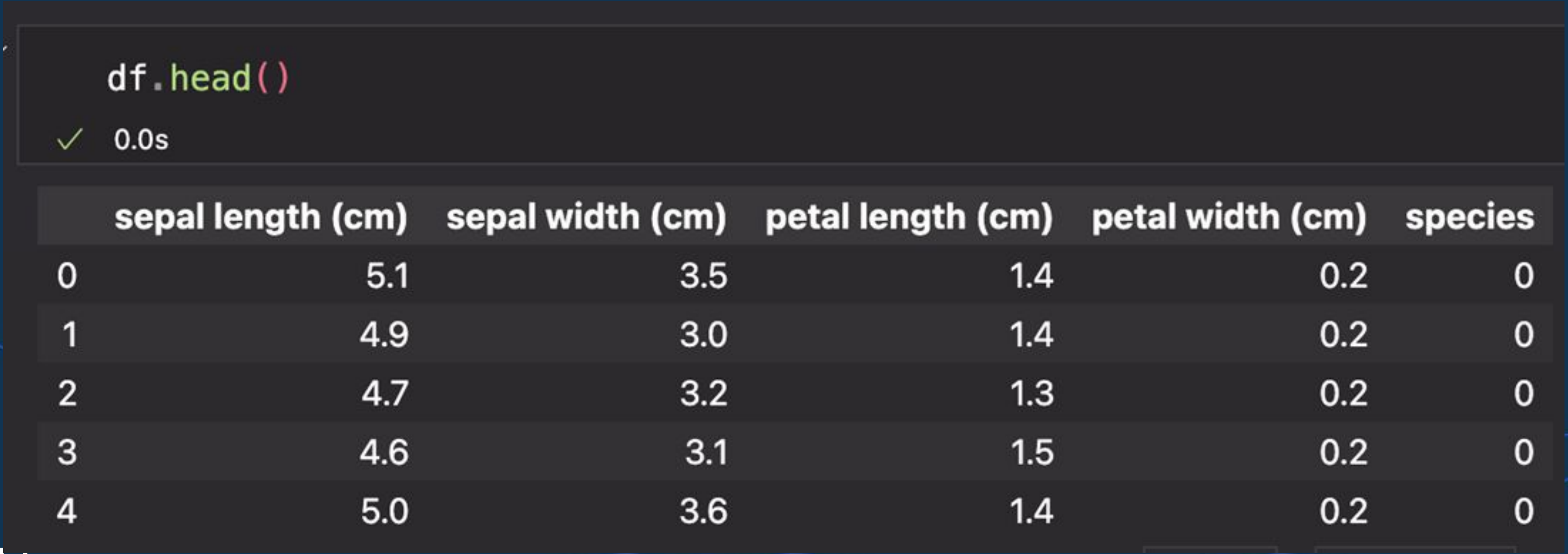


```
# url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv'
# df = pd.read_csv(url)

iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

Exploring datasets

- **df.head(), df.tail()**: Peek at the top and bottom rows for initial understanding



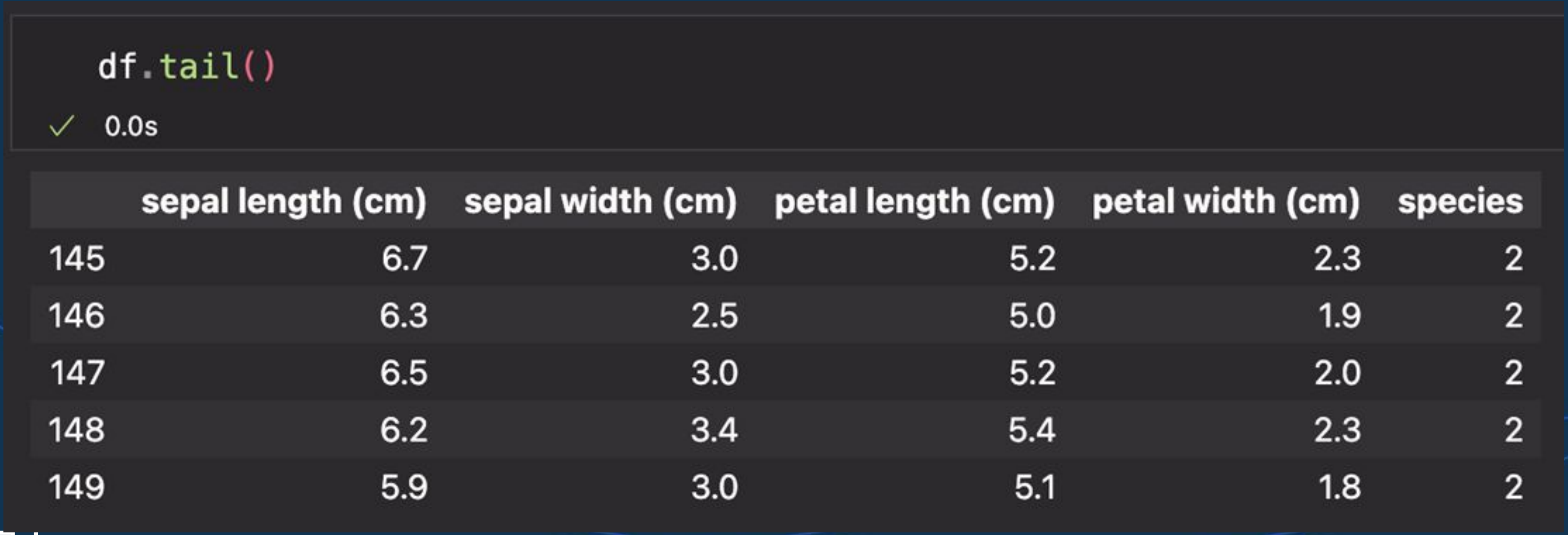
```
df.head()
```

✓ 0.0s

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Exploring datasets

- **df.head(), df.tail()**: Peek at the top and bottom rows for initial understanding



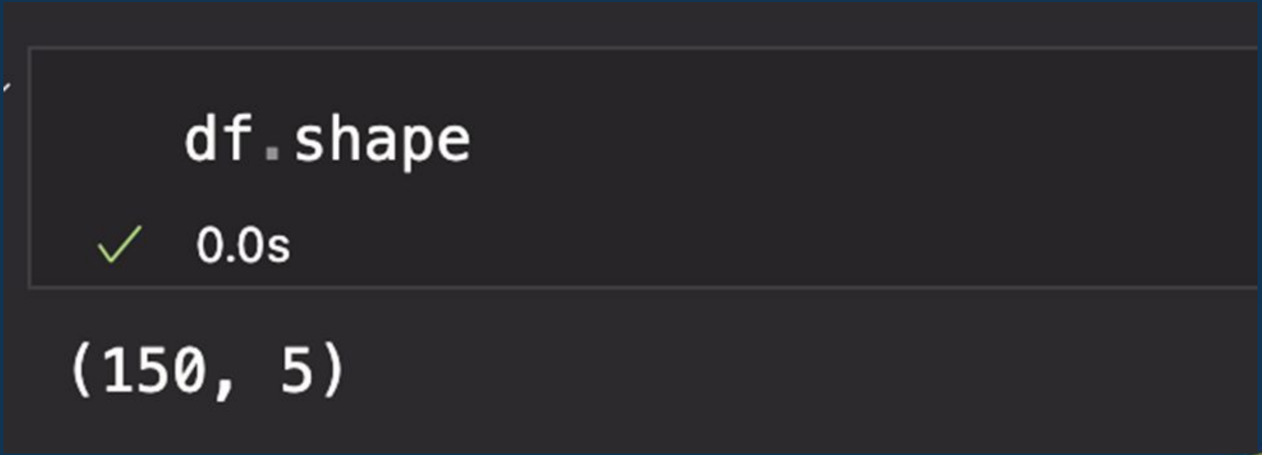
```
df.tail()
```

✓ 0.0s

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

Exploring datasets

- **df.shape:** Tells you the dimensions (rows, columns) of your data.



```
df.shape
```

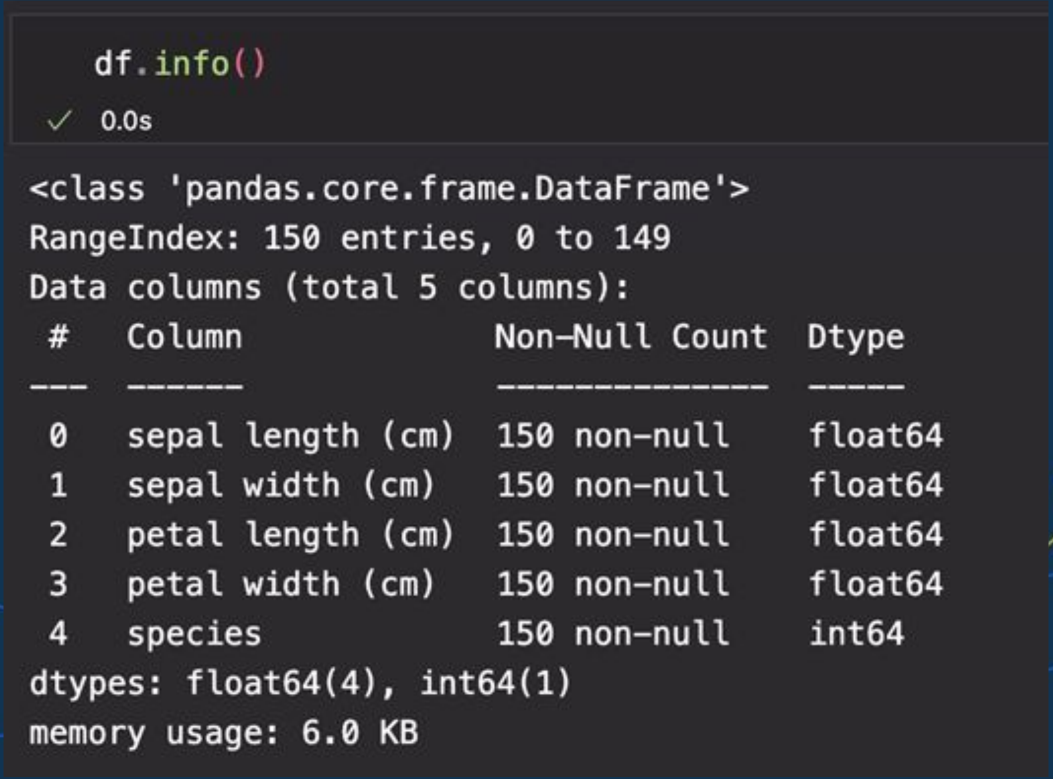
✓ 0.0s

```
(150, 5)
```

HyperionDev

Exploring datasets

- **df.info():** Gives the data types of each column, and if columns have missing values



```
df.info()
```

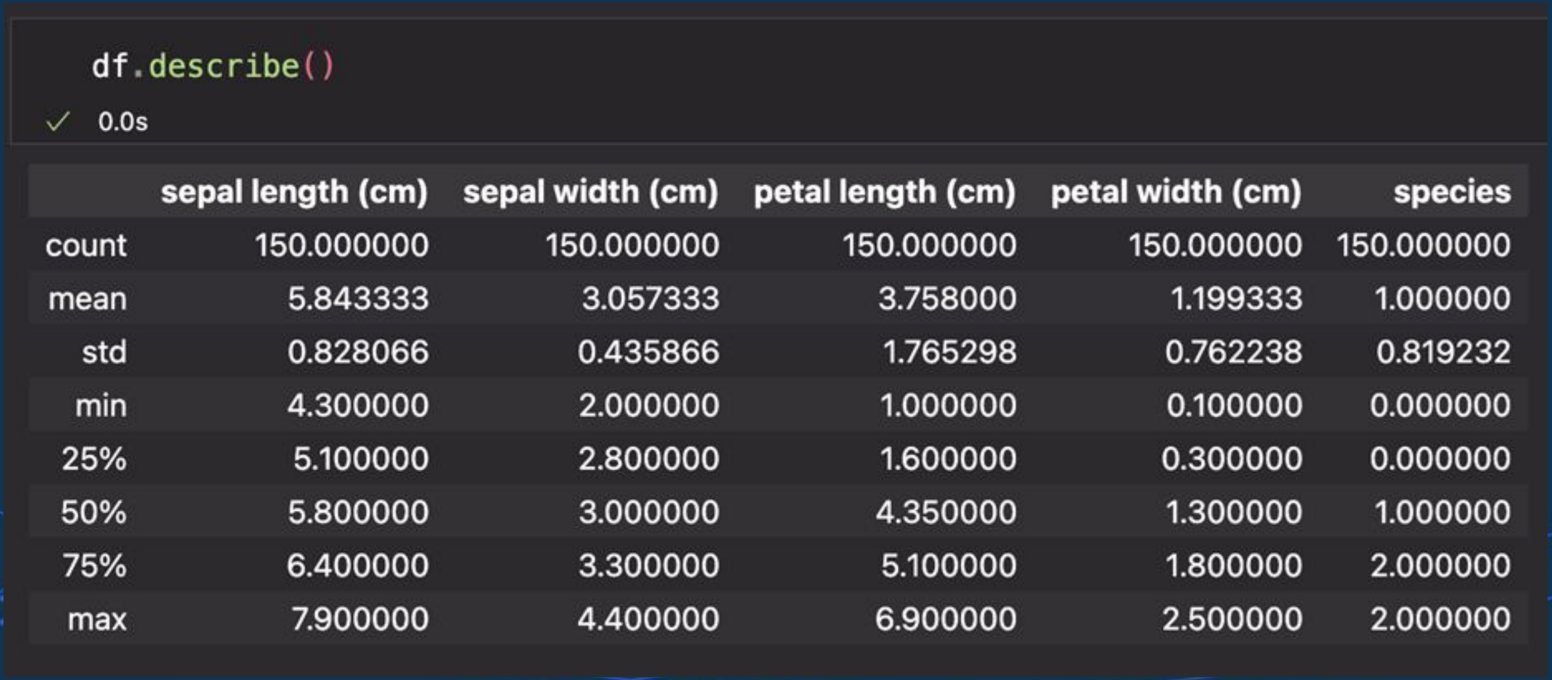
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

HyperionDev

Exploring datasets

- **df.describe():** Quick summary statistics for numerical columns.



```
df.describe()
```

✓ 0.0s

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

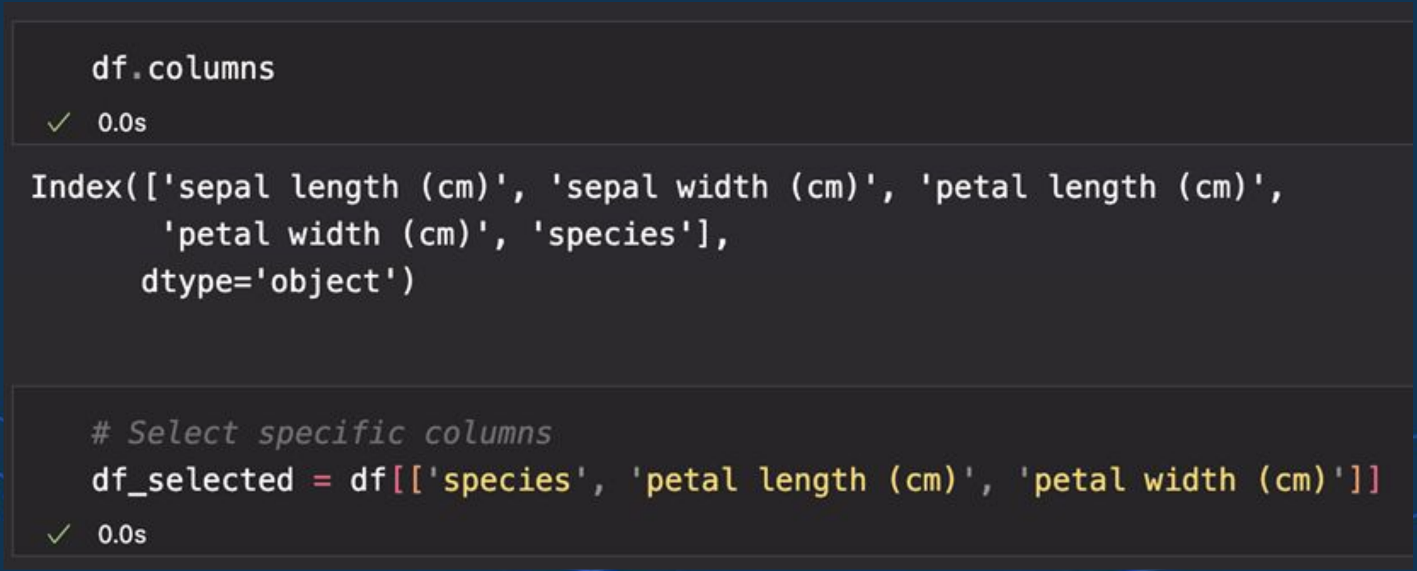
Manipulating Data

HyperionDev

Manipulating Data

- **Selecting Columns**: You often work with a **subset of features.**

- Using df[['column1', 'column2']] gets you only specific columns.



```
df.columns
```

✓ 0.0s

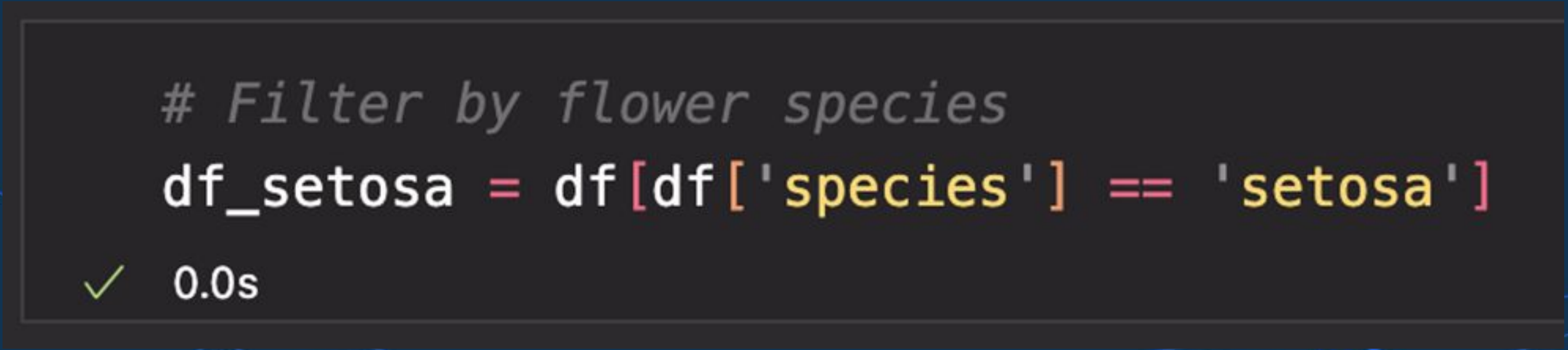
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'species'],
      dtype='object')
```

```
# Select specific columns
df_selected = df[['species', 'petal length (cm)', 'petal width (cm)']]
```

✓ 0.0s

Manipulating Data

- **Filtering Rows:** Focus on specific subsets meeting certain conditions, e.g., df[df['species'] == 'setosa’]

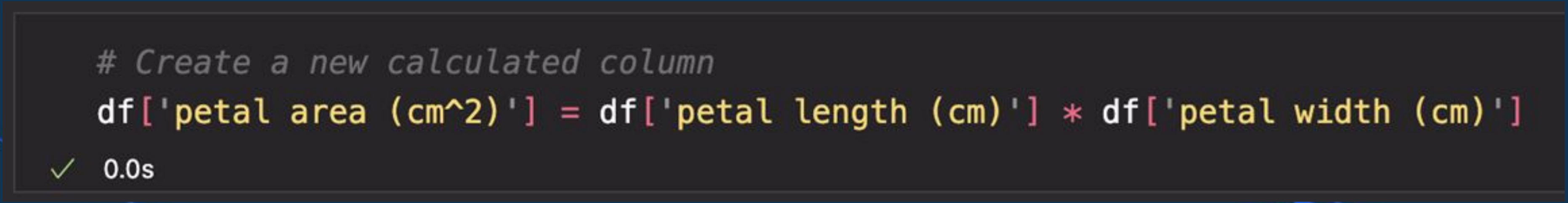


```
# Filter by flower species
df_setosa = df[df['species'] == 'setosa']
```

```
✓  0.0s
```

Manipulating Data

- **Creating New Columns:** Derived features, e.g., calculating area from length and width.

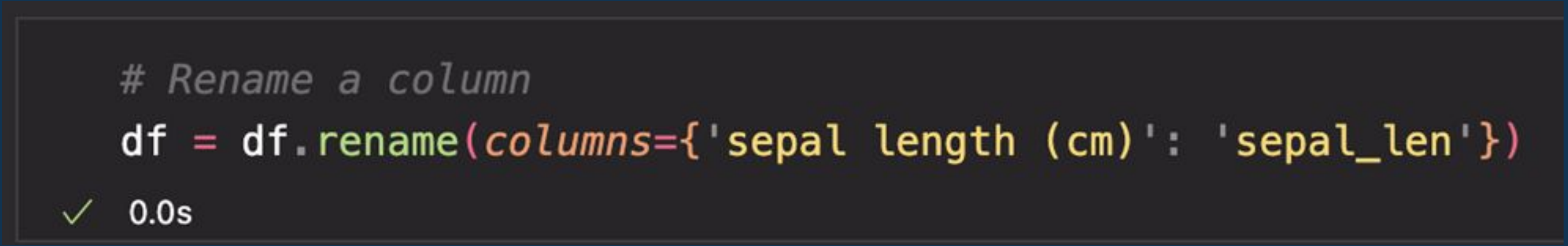


```
# Create a new calculated column
df['petal area (cm^2)'] = df['petal length (cm)'] * df['petal width (cm)']
```

✓ 0.0s

Manipulating Data

- **Renaming/Dropping:** Improve clarity or get rid of unneeded data.



```
# Rename a column
df = df.rename(columns={'sepal length (cm)': 'sepal_len'})
```

```
✓ 0.0s
```

- Data manipulation gives you a **highly customized DataFrame** focused on your exact analysis needs.

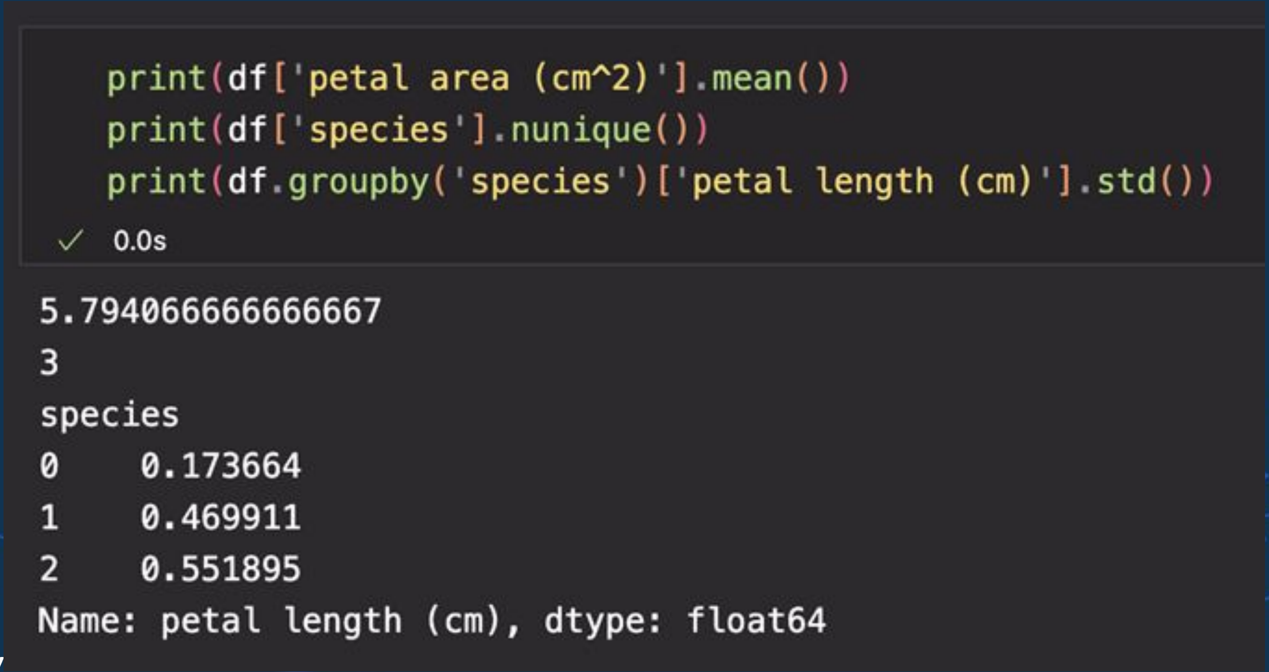
Built-in Methods

- ❖ Pandas offers a toolbox of functions for calculations:
  - ➢ **mean()** - Computes the mean for each column.
  - ➢ **min()** - Computes the minimum for each column.
  - ➢ **max()** - Computes the maximum for each column.
  - ➢ **std()** - Computes the standard deviation for each column.
  - ➢ **var()** - Computes the variance for each column.
  - ➢ **unique()** - Computes the number of unique values in each column.

- ❖ This is the start of understanding the characteristics of your data.

Grouping and Aggregation

- df.groupby(): Divide your data **based on categories** in a column (e.g., group by species).



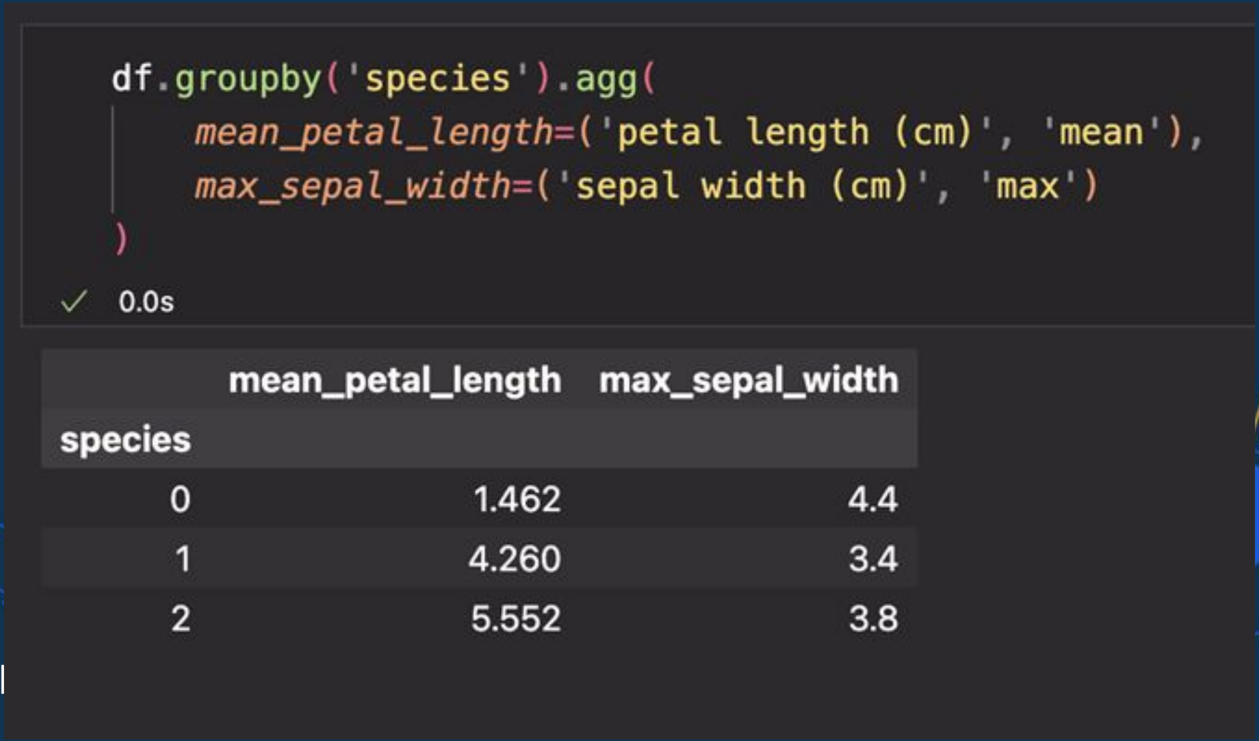
```python
print(df['petal area (cm^2)'].mean())
print(df['species'].nunique())
print(df.groupby('species')['petal length (cm)'].std())
```

✓ 0.0s

```
5.794066666666667
3
species
0    0.173664
1    0.469911
2    0.551895
Name: petal length (cm), dtype: float64
```

Grouping and Aggregation

- **.agg()**: Apply calculations within each group (e.g., average length, maximum width).



```python
df.groupby('species').agg(
    mean_petal_length=('petal length (cm)', 'mean'),
    max_sepal_width=('sepal width (cm)', 'max')
)
```

✓ 0.0s

	mean_petal_length	max_sepal_width
species		
0	1.462	4.4
1	4.260	3.4
2	5.552	3.8

Jupyter Notebook

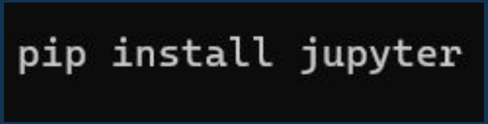
HyperionDev

Jupyter Notebook

- ❖ An interactive environment perfect for data science work. They let you combine code, the results of the code (output), and explanatory text (like in a scientific report).

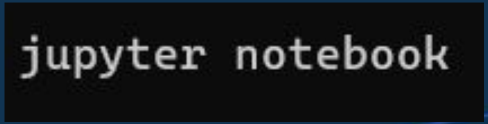
- ❖ This fosters clear data exploration and storytelling, all in one place

Installation

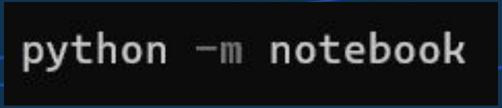


```
pip install jupyter
```

Running



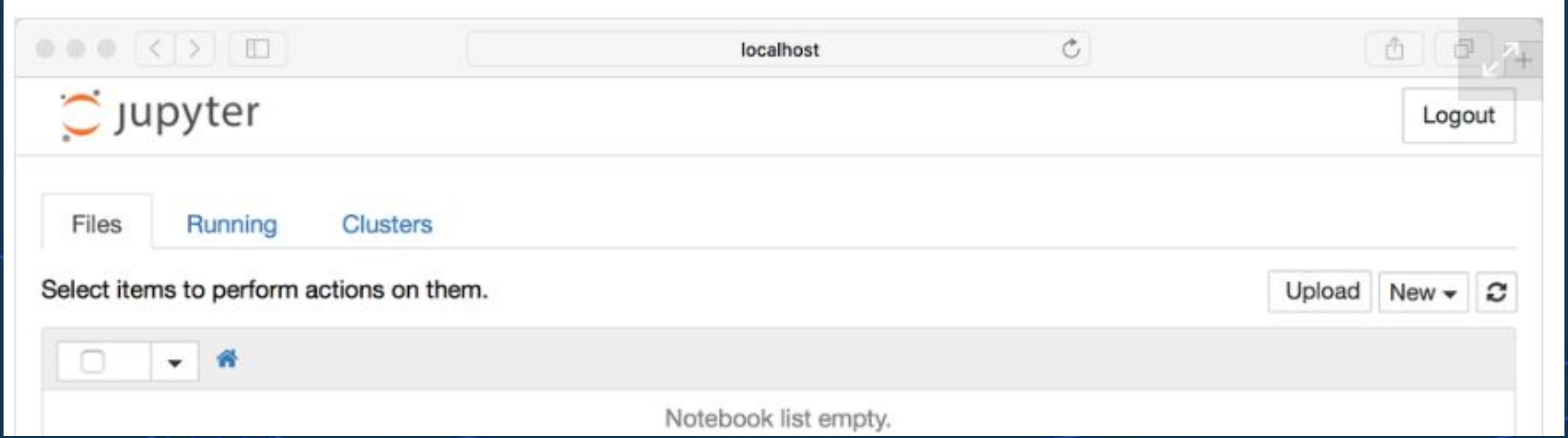
```
jupyter notebook
```



```
python -m notebook
```

Jupyter Notebook

Starting the Jupyter Notebook Server, default browser goes to the URL http://localhost:8888/tree

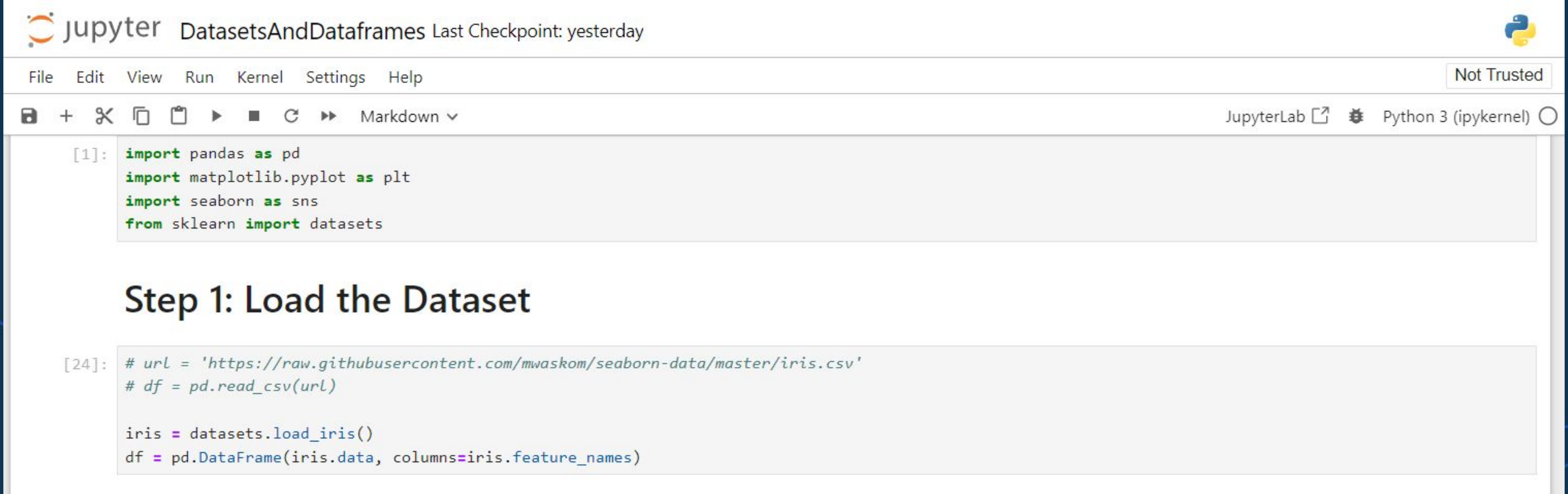


Select items to perform actions on them.

Notebook list empty.

Jupyter Notebook

Creating, naming notebook, code, markdown


jupyter DatasetsAndDataframes Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Not Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
[1]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn import datasets
```

Step 1: Load the Dataset

```
[24]: # url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv'
      # df = pd.read_csv(url)

      iris = datasets.load_iris()
      df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

HyperionDev

Questions and Answers

HyperionDev

Thank you for attending

HyperionDev