ISA 419: Data-Driven Security

04: A Short Introduction to Pandas

Fadel M. Megahed, PhD

EProfessor
Farmer School of Business
Miami University

@FadelMegahed
fmegahed
fmegahed@miamioh.edu
Automated Scheduler for Office Hours

Spring 2025

Quick Refresher of Last Class

✅ Understand the anatomy of a Python function, use arguments correctly, and construct your first function.

✅ Utilize built-in and anonymous functions (`map`, `lambda`, `filter`).

✅ Analyze your second dataset.

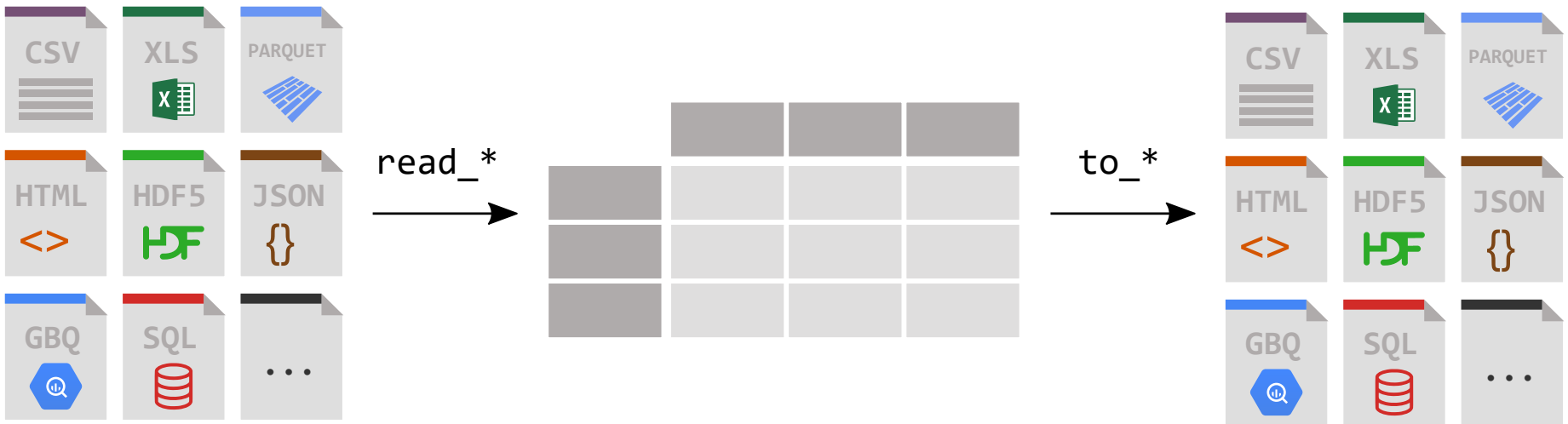
Learning Objectives for Today's Class

- Describe and create data frames
- Import data using pandas
- Utilize the `head()`, `tail()`, `shape()`, `info()`, `dtypes()`, `count()`, `value_count()`, `describe()`, `isnull().sum()` functions to quickly explore and summarize your dataset
- Use the `loc[]` and `iloc[]` to select rows and columns
- Use `assign()` to create new columns on your existing data frame.

Describing and Creating Data Frames

What is Pandas?

- Along with polars, it is one of the most popular data analysis and manipulation libraries in Python, designed to make working with 'relational' or 'labeled' data both easy and intuitive.
- pandas supports the integration with many file formats or data sources out of the box (csv, excel, sql, json, parquet,...). Importing data from each of these data sources is provided by function with the prefix `read_*`. Similarly, the `to_*` methods are used to store data.



Source: The content and image are from the *Getting Started with Pandas* description in pandas.pydata.org. The image can be directly accessed here.

What is a Data Frame?

- A data frame is a two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).
- It is a primary data structure in pandas.
- It is similar to a spreadsheet or SQL table, or a dictionary of Series objects.
- It is generally the most commonly used pandas object.

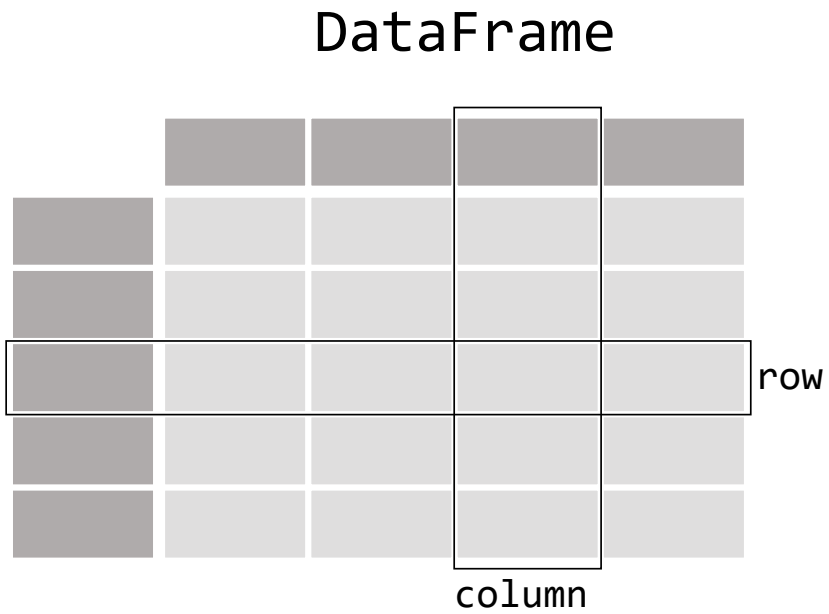
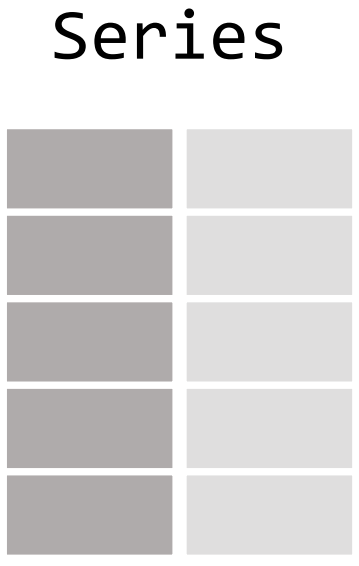


Image Source: The image is from the *Getting Started with Pandas* description in pandas.pydata.org. The image can be directly accessed here.

What is a Series?

- Each column in a `DataFrame` is a `Series`.
- When **selecting a single column of a pandas** `DataFrame`, the **result** is a pandas `Series`. To select the column, use the *column label* in between square brackets `[]`.
- A Series is a one-dimensional data structure capable of holding data of any type (integer, string, float, python objects, etc.); it is **somewhat similar** to a list/dictionary.



Source: The content and image are from the *Getting Started with Pandas* tutorial titled: What kind of data does pandas handle? The image can be directly accessed here.

Creating a Series

We can easily create a `Series` from a list, or a dictionary.

From a List

```
import pandas as pd

# Creating a Series from a list
lst = [3, 5, 7, 9]
s1 = pd.Series(lst)
print(s1)
```

```
## 0    3
## 1    5
## 2    7
## 3    9
## dtype: int64
```

From a Dictionary

```
import pandas as pd

# Creating a Series from a dictionary
d = {'a': 3, 'b': 5, 'c': 7, 'd': 9}
s2 = pd.Series(d)
print(s2)
```

```
## a    3
## b    5
## c    7
## d    9
## dtype: int64
```

What are the differences between the two created series?

Edit me to answer the question.

Creating a Data Frame

- There are several ways, where we can create a `DataFrame` in pandas. These include:

From a Dictionary of Lists

```
import pandas as pd

# DataFrame from a dictionary of lists
l1 = [1, 5]
l2 = [2, 10]
data = {'a': l1, 'b': l2, 'c': [20, 30]}
df1 = pd.DataFrame(data)
print(df1)
```

```
##    a   b   c
## 0  1   2  20
## 1  5  10  30
```

From a List of Dictionaries

```
import pandas as pd

# DataFrame from a list of dictionaries
dict1 = {'a': 1, 'b': 2}
dict2 = {'a': 5, 'b': 10, 'c': 20}
data = [dict1, dict2]
df2 = pd.DataFrame(data)
print(df2)
```

```
##    a   b     c
## 0  1   2   NaN
## 1  5  10  20.0
```

Note the highlighted row in the second code chunk. What happened there?

Edit me to answer the question.

Class Activity: Creating a Data Frame

04:00

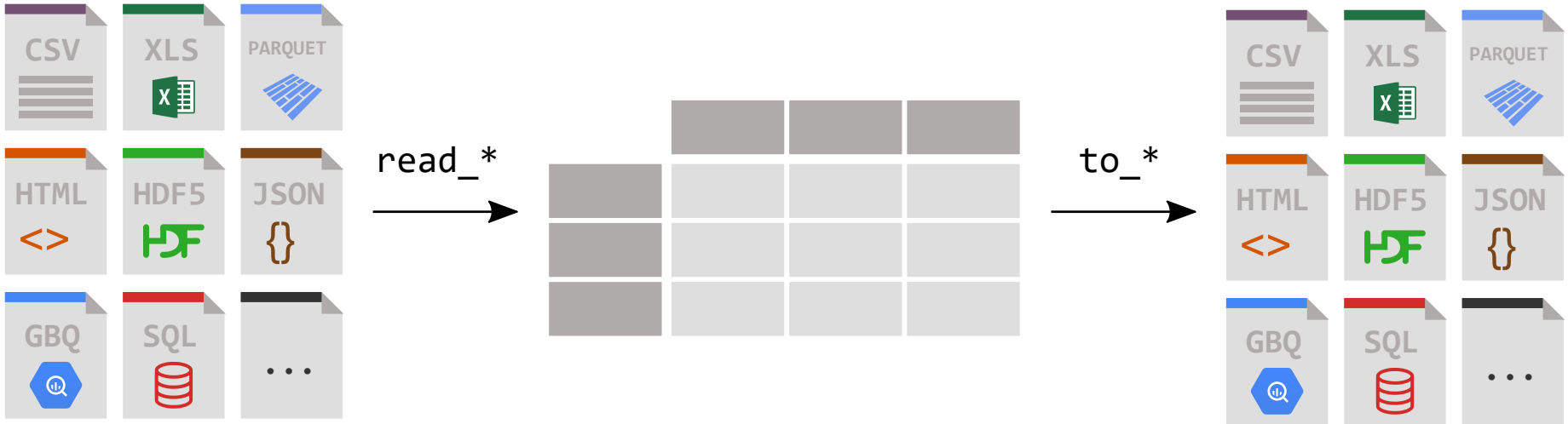
Task | Comments | Your Answer

- Use Google Colab to identify the correct piece of code that can be used to create two data frames: (a) a data frame from a single list, and (b) a data frame from two lists.

```
import pandas as pd

my_lst = [1, 5, 2, 10]
df3 = pd.DataFrame(my_lst)

# A data frame from two lists
l1 = [1, 5]
l2 = [2, 10]

# which of these produces the expected result,
# where l1 and l2 are the columns of the data frame?
df4a = pd.DataFrame(l1, l2)
df4b = pd.DataFrame((l1, l2))
df4c = pd.DataFrame([l1, l2])
df4d = pd.DataFrame(zip(l1,l2))
```

Importing Data Using Pandas

Importing Data Using Pandas

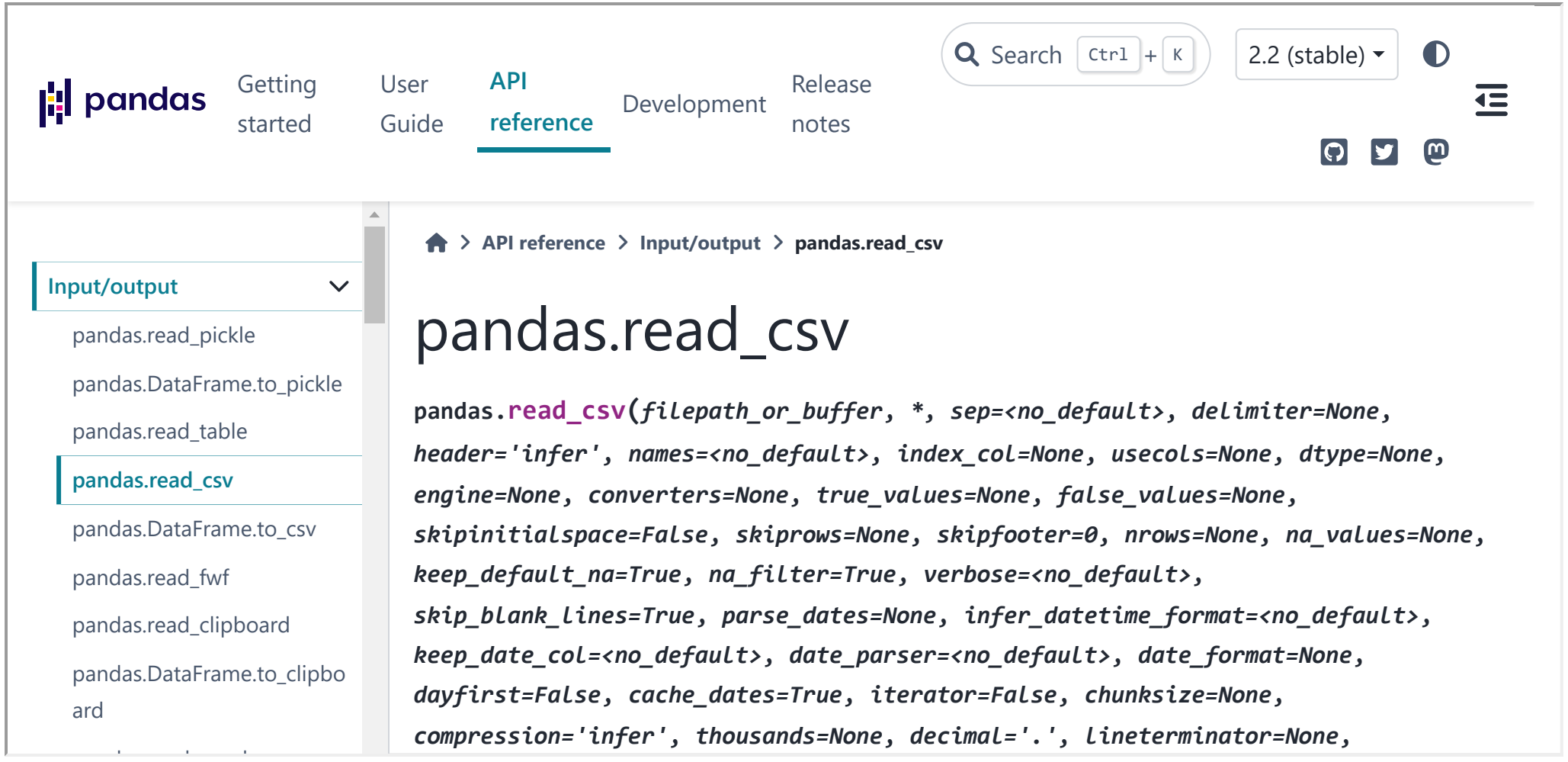


Source: The image is from the *Getting Started with Pandas* tutorial available at pandas.pydata.org. The image can be directly accessed here.

Things to Consider while Importing Data with Pandas

- **Check the file extension:** Your **choice of function** will depend on the file extension.
- **Check the file path:**
  - **Local file:**
    - If the file is in the **same directory** as your code, you can simply provide the **file name**.
    - **Otherwise**, you will need to provide the **path** to the file.
  - **Remote file:** You will need to provide the **URL to the file**.
- **Inspect the file:** to know which **default parameter values must be changed**. Things to consider:
  - Delimiter (in the case of .txt files)
  - Header (does it exist? if not, what should be used as column names?)
  - Index Column (if it exists, what is it? if not, what should be used as the index?)
  - Encoding (to be changed if file is not in English, or has names such as: François, José, Weiß, etc.)
  - Date Format (if the file has dates, what is the format?)
  - etc.

Importing Data Using Pandas: CSVs



Source: The content is from the `pandas.read_csv()` documentation available at pandas.pydata.org.

Importing Data Using Pandas: CSVs [Example 1]

Here is how you can import the **Simulated Equifax Breach Dataset** from a local file.

```
import pandas as pd

equifax_df1 = pd.read_csv('../../data/simulated_equifax_breach_data.csv')
equifax_df1.head()
```

```
##                Name  ... Driver License Number
## 0  Matthew Jenkins  ...              OA648128
## 1   James Williams  ...              BB738847
## 2   William Patton  ...              EL391232
## 3       Eric Wells  ...              DZ520550
## 4   William Walker  ...              GV282163
##
## [5 rows x 6 columns]
```

Importing Data Using Pandas: CSVs [Example 2]

Here is how you can import the **Simulated Equifax Breach Dataset** from a remote file.

```
import pandas as pd

equifax_df2 = pd.read_csv('https://raw.githubusercontent.com/fmegahed/isa419/main/data/simul
equifax_df2.head()
```

```
##                Name  ... Driver License Number
## 0   Matthew Jenkins  ...               OA648128
## 1    James Williams  ...               BB738847
## 2    William Patton  ...               EL391232
## 3        Eric Wells  ...               DZ520550
## 4    William Walker  ...               GV282163
##
## [5 rows x 6 columns]
```

Importing Data Using Pandas: XML [Example 3]

Here is how you can import the **open threat feeds** from the ICS Sans API.

```
open_threats = pd.read_xml('https://isc.sans.edu/api/threatfeeds/')
open_threats.head()
```

```
##               type  ... frequency
## 0          adscore  ...         0
## 1           alltor  ...      3600
## 2  alphastrikelabs  ...         0
## 3        anthropic  ...      3600
## 4            arbor  ...         0
##
## [5 rows x 6 columns]
```

Importing Data Using Pandas: JSON [Example 4]

Here is how you can import the *Scapy Python Package Issues* from the GitHub API. Note that *Scapy* is a powerful interactive packet manipulation package in Python (often used in *Pen Testing* and *Forensic Investigations*).

```
import pandas as pd

# Reading JSON data
scapy_issues = pd.read_json('https://api.github.com/repos/secdev/scapy/issues')
scapy_issues.head()
```

```
##                                                 url  ...
## 0  https://api.github.com/repos/secdev/scapy/issu...  ...
## 1  https://api.github.com/repos/secdev/scapy/issu...  ...  {'url': 'https://api.github.co
## 2  https://api.github.com/repos/secdev/scapy/issu...  ...
## 3  https://api.github.com/repos/secdev/scapy/issu...  ...  {'url': 'https://api.github.co
## 4  https://api.github.com/repos/secdev/scapy/issu...  ...
##
## [5 rows x 32 columns]
```

Exploring and Summarizing Data

Exploring and Summarizing Data

Function	Description	Usage
`head()`	Returns the first `n` rows (default = 5) for the object based on position.	`equifax_df2.head()`
`tail()`	Returns the last `n` rows (default = 5) for the object based on position.	`equifax_df2.tail()`
`shape`	Returns the number of rows and columns in the DataFrame.	`equifax_df2.shape`
`info()`	Prints information about the DataFrame, including the number of entries, the column names, and the data types.	`equifax_df2.info()`
`dtypes`	Returns the data types of the columns.	`equifax_df2.dtypes`

Exploring and Summarizing Data (Cont.)

Function	Description	Usage
`count()`	Returns the number of non-null values in each DataFrame column.	`equifax_df2.count()`
`value_counts()`	Returns the frequency counts for each unique value in a column. **Note: Applied per column.**	`equifax_df2['Name'].value_counts()`
`nunique()`	Returns the number of unique values in a column. **Note: Applied per column.**	`equifax_df2['Name'].nunique()`
`describe()`	Provides descriptive statistics for numerical columns, but can also be used for object columns with `include='all'` parameter.	`equifax_df2.describe()`
`isnull().sum()`	Returns the count of missing values (`NaN`) in each column.	`equifax_df2.isnull().sum()`

Explore and Summarize the Equifax Breach Dataset

05:00

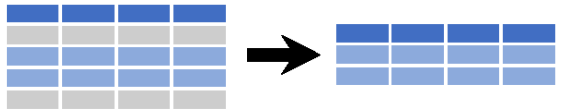
Task | Task 1 | Task 2 | Task 3 | Task 4 | Task 5

- Download the `simulated_equifax_breach_data.csv` file from Canvas.
- Load the dataset into Google Colab.
- Using **pandas**, answer the questions in the next tabs. Note the first three are identical to what we did in the previous class

Subsetting Data in Pandas

Subsetting Observations/Rows in Pandas

Subset Observations - rows



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)` Randomly select n rows.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

Subset Variables - columns

`df[['width', 'length', 'species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use `df.loc[]` and `df.iloc[]` to select only rows, only columns or both.
Use `df.at[]` and `df.iat[]` to access a single value by row and column.
First index selects rows, second index columns.

`df.iloc[10:20]`
Select rows 10-20.

`df.iloc[:, [1, 2, 5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[:, 'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.loc[df['a'] > 10, ['a', 'c']]`
Select rows meeting logical condition, and only the specific columns .

`df.iat[1, 2]` Access single value by index

`df.at[4, 'A']` Access single value by label

Source: The image is cropped from the Pandas Cheat Sheet.

Common Issues with Subsetting

- A common issue when **subsetting rows is that the index is not reset**, i.e., the index of the original data frame is retained. This **can be problematic in the following subsequent scenarios:**
  - When you want to subset the data frame using the `iloc[]` or `loc[]` functions.
  - When you want to access the data frame using `.iat[]`.

To **overcome this issue**, use the **method `.reset_index(drop=True)`** to reset the index of the data frame.

Common Issues with Subsetting (Cont.)

- The `query()` method **cannot handle column names with spaces**.
- To **overcome this issue**, we can do the following:
  - first, use the **method `.rename()`** to rename the columns of the data frame;
  - then, use the **method `.query()`** to subset the data frame.

Example: `df.rename(columns = {'old name': 'new_name'}, inplace = True)`

OR

- Use the method **`.query()`** to subset the data frame, but use **backticks** to enclose the column names with spaces.

Example: `df.query('` old name ` > 5')`

OR

- Use `clean_names()` from the pyjanitor package to clean the column names, and then use, `query()` to subset the data frame.

Creating New Columns in Pandas

Creating New Columns in Pandas using `assign()`

- The `assign()` method is used to create new columns on your existing data frame.
- It returns a new data frame with the new columns added to the original data frame.
- The method takes the form: `df.assign(new_column_name = new_column_values)`.
- The `new_column_values` can be a single value, a list, or a series.
- The method can also be used to create multiple columns at once.

Demo: Creating New Columns using `assign()`

In class, we will use `assign()` combined with `lambda` to create several new columns.

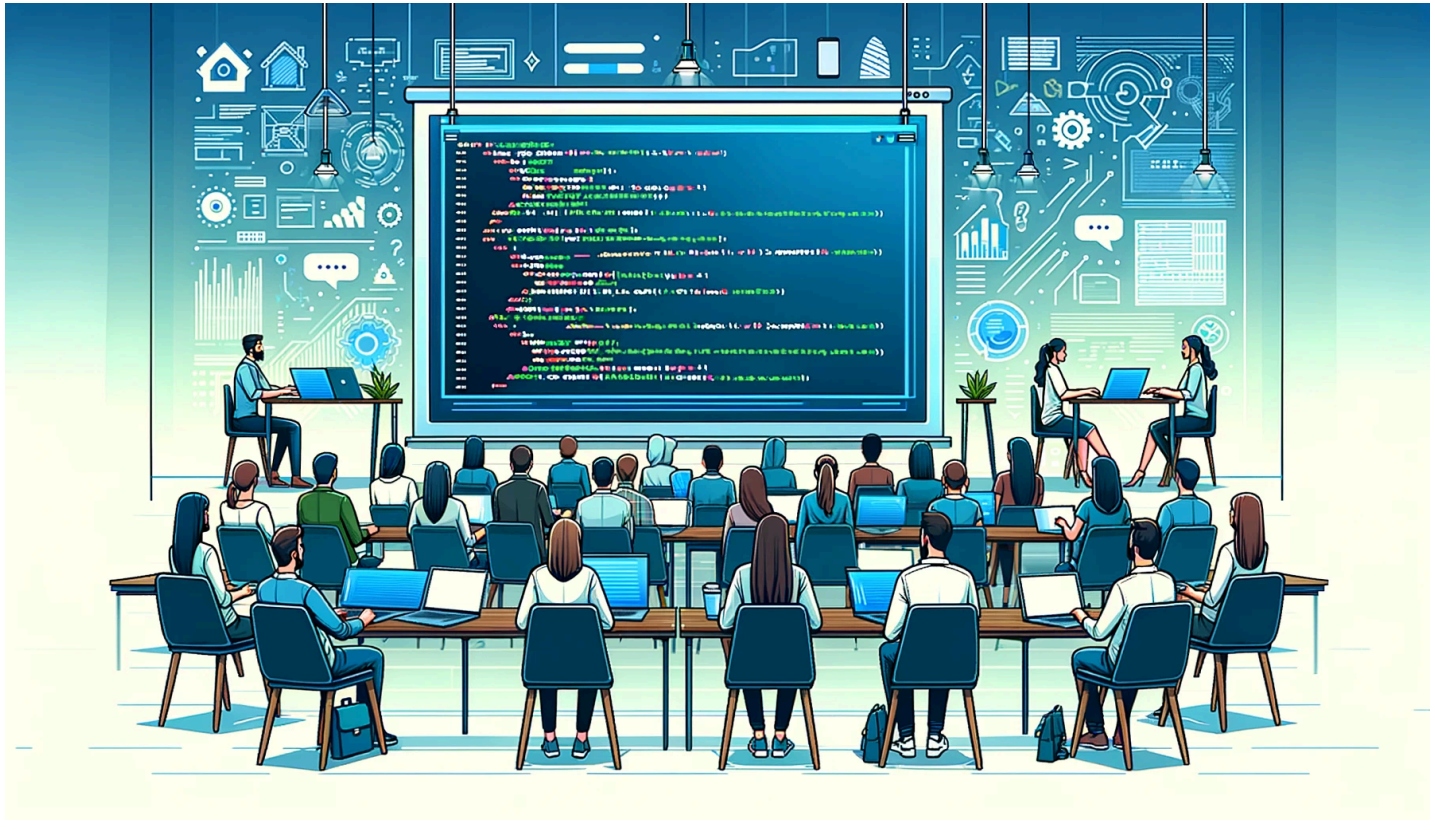


Image Source: The image was generated using DALL·E on 2024-02-07.

Bonus Tip: We can use `.reindex(columns = [some_list])` to reorder/drop columns.

Recap

Summary of Main Points

By now, you should be able to do the following:

- Describe and create data frames
- Import data using pandas
- Utilize the `head()`, `tail()`, `shape()`, `info()`, `dtypes()`, `count()`, `value_count()`, `describe()`, `isnull().sum()` functions to quickly explore and summarize your dataset
- Use the `loc[]` and `iloc[]` to select rows and columns
- Use `assign()` to create new columns on your existing data frame.

 Review and Clarification 

- **Class Notes**: Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording**: The recording of today's class will be made available on Canvas approximately 3-4 hours after the end of class.
- **Questions**: Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.