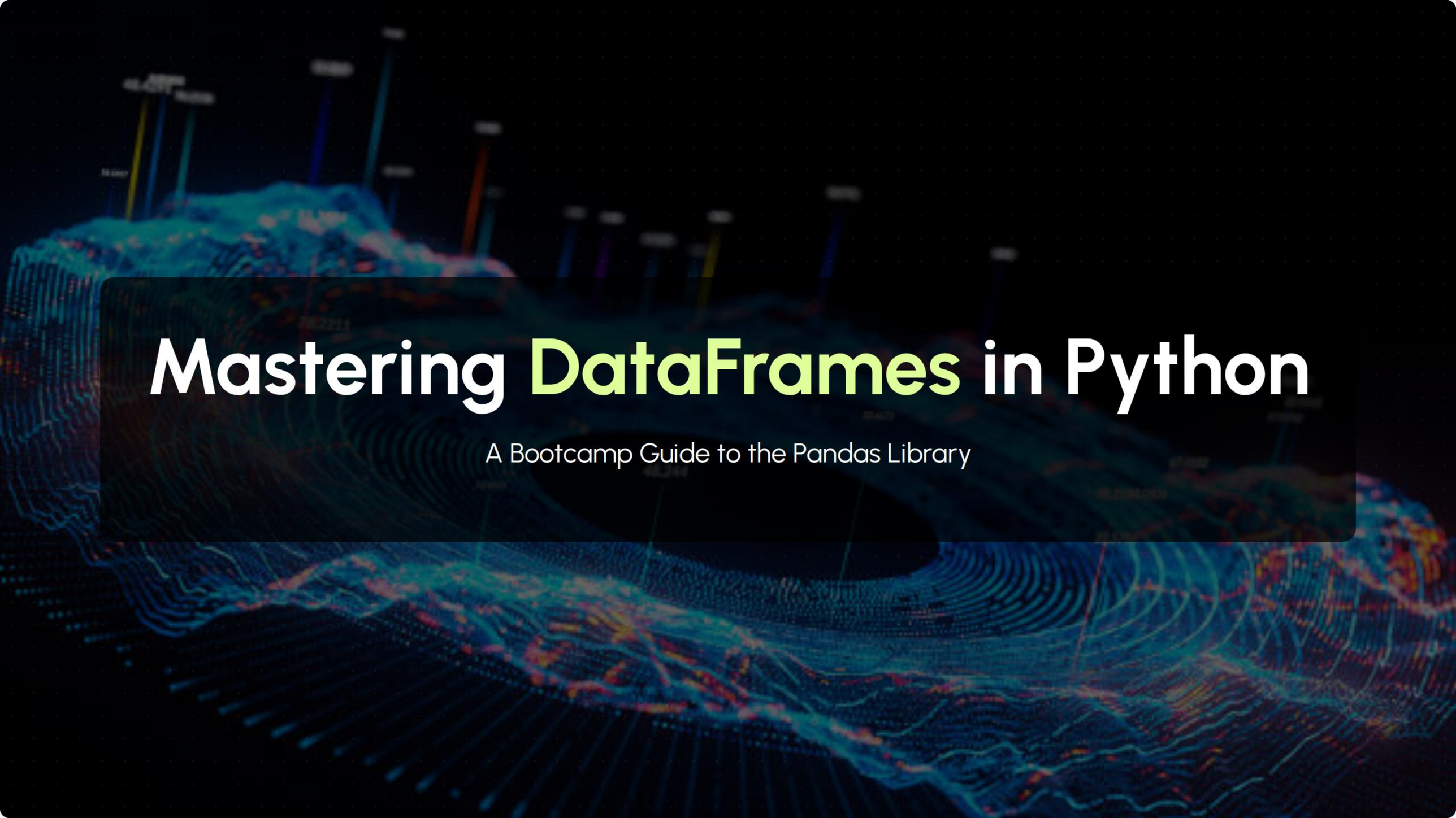
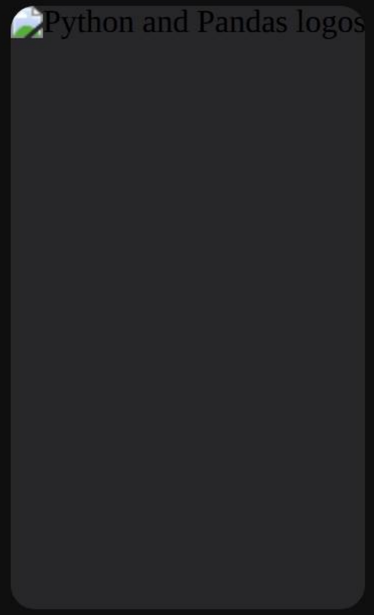

Mastering DataFrames in Python
A Bootcamp Guide to the Pandas Library

What is Data Analysis?

- **Inspecting:** Looking at your data to understand its structure.
- **Cleaning:** Fixing errors, handling missing values, and formatting.
- **Transforming:** Changing the data's shape or content to suit your needs.
- **Modeling & Aggregating:** Summarizing data to find insights.
- **Visualizing:** Turning your findings into charts and graphs.

Why not Excel? Scalability, Reproducibility, and Power.

The Python Data Science Stack



What is Pandas?

Pandas is the "spreadsheet" of Python.

It's the single most important tool for data analysis and manipulation in Python, built on top of NumPy.

What is a DataFrame?

The Core Concept

A DataFrame is a 2-dimensional, size-mutable, and labeled data structure.

Think of it as:

- ✓ An Excel spreadsheet
- ✓ A SQL table
- ✓ A dictionary of Series objects

Visual Analogy

```
  (Index)   | Name      | Age | City
+-----------+-----------+-----+-----------+
|   0       | Alice     | 25  | New York  |
|   1       | Bob       | 30  | London    |
|   2       | Charlie   | 22  | Paris     |
+-----------+-----------+-----+-----------+
    (Labels)    (Columns of data)
```

The Two Core Pandas Objects

Series (1D)

A single column. It's like a 1D array or a list with labels (an index).

```
0    Alice
1    Bob
2    Charlie
Name: Name, dtype: object
```

+

+

DataFrame (2D)

A collection of Series (columns) that share the same index.

```
    Name   Age
0  Alice   25
1  Bob     30
2  Charlie 22
```

Module 1

Creating DataFrames

Creation: From a Dictionary

Method 1: From Dictionary

The most common way to create a small DataFrame. Keys become column headers, values become column data.

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 22]
}

df = pd.DataFrame(data)
```

Result:

```
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   22
```

Creation: From a List of Dictionaries

Method 2: List of Dicts

Very useful when your data comes from a JSON API. Each dictionary is a row.

```
data = [
    {'Name': 'Alice', 'Age': 25},
    {'Name': 'Bob', 'Age': 30},
    {'Name': 'Charlie', 'Age': 22}
]

df = pd.DataFrame(data)
```

Result:

```
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   22
```

Creation: From a CSV File

Method 3: From a File

This is the most practical and common way to load data in the real world.

```
# Assuming you have a file 'my_data.csv'
# in the same directory

df = pd.read_csv('my_data.csv')

# You can also read from Excel!
# df = pd.read_excel('my_data.xlsx')
```

Result:

Your data is now loaded into a DataFrame named `df`, ready for analysis!

Module 2

Inspecting Your Data

You've loaded your data. Now what?

Inspection: `df.head()` & `df.tail()`

See the Start and End

Get a quick look at the first or last 5 rows. Essential for a "sanity check."

```
# Show the first 5 rows
df.head()

# Show the last 3 rows
df.tail(3)
```

Result (from `df.head()`):

	PassengerId	Survived	Pclass	Name	Sex	Age
0	1	0	3	Braund, Mr...	male	22.0
1	2	1	1	Cumings, Mrs...	female	38.0
2	3	1	3	Heikkinen, ...	female	26.0
3	4	1	1	Futrelle, Mrs...	female	35.0
4	5	0	3	Allen, Mr...	male	35.0

Inspection: `df.info()`

The Most Important Method

Provides a technical summary:

- ✓ Number of rows (entries)
- ✓ Column names
- ✓ Number of non-null values (CRITICAL for cleaning)
- ✓ Data types (dtype) of each column
- ✓ Memory usage

```
df.info()
```

Result:

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Inspection: `df.shape` & `df.columns`

How big is it? What's in it?

`df.shape` is an attribute (not a method, no `()`) that gives you a tuple of `(rows, columns)`.

`df.columns` gives you a list of all column names.

```
# Get the shape
print(df.shape)

# Get the column names
print(df.columns)
```

Result:

```
(891, 12)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Inspection: `df.describe()`

Statistical Summary

Gives you a quick statistical summary (count, mean, std, min, max, quartiles) for all numerical columns.

Instantly reveals scale, outliers, and distribution.

```
# Get statistical summary
df.describe()
```

Result (abbreviated):

	PassengerId	Survived	Pclass	Age
count	891.000000	891.000000	891.000000	714.000000
mean	446.000000	0.383838	2.308642	29.699118
std	257.353842	0.486592	0.836071	14.526497
min	1.000000	0.000000	1.000000	0.420000
25%	223.500000	0.000000	2.000000	20.125000
50%	446.000000	0.000000	3.000000	28.000000
75%	668.500000	1.000000	3.000000	38.000000
max	891.000000	1.000000	3.000000	80.000000

Module 3

Selecting Data (Subsetting)

How to get the exact data you want.

Selecting Columns

One Column (returns a Series)

```
# Get the 'Name' column
df['Name']
```

Multiple Columns (returns a DataFrame)

Note the double square brackets `[[ ... ]]`

```
# Get 'Name' and 'Age' columns
df[['Name', 'Age']]
```

Result (One Column):

```
0      Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley
2     Heikkinen, Miss. Laina
...
Name: Name, dtype: object
```

Result (Multiple Columns):

```
                          Name   Age
0      Braund, Mr. ...   22.0
1    Cumings, Mrs...   38.0
2     Heikkinen, ...   26.0
...
```

Selecting Rows: `.loc` (Label-Based)

Using the Index Label

`.loc` is the primary way to select data by its label. The index is a label (even if it's a number like 0, 1, 2).

```
# Get the row with index label 0
df.loc[0]

# Get rows with index labels 0, 1, and 2
df.loc[[0, 1, 2]]

# Get rows 0 to 3 (inclusive!)
df.loc[0:3]
```

Result (from `df.loc[0]`):

```
PassengerId                     1
Survived                        0
Pclass                          3
Name            Braund, Mr. Owen Harris
Sex                          male
Age                          22.0
...
Name: 0, dtype: object
```

Selecting Rows: `.iloc` (Integer-Based)

Using the Index Position

`.iloc` selects data by its integer position (like a standard Python list).

```
# Get the first row (at position 0)
df.iloc[0]

# Get the first three rows
df.iloc[[0, 1, 2]]

# Get rows 0 up to (but not including) 3
df.iloc[0:3]
```

Result (from `df.iloc[0]`):

```
PassengerId                     1
Survived                        0
Pclass                          3
Name            Braund, Mr. Owen Harris
Sex                          male
Age                          22.0
...
Name: 0, dtype: object
```

`.loc` vs `.iloc`

Slicing `loc[0:3]` includes 3. Slicing `iloc[0:3]` excludes 3.

The Powerhouse: Boolean Indexing

This is the way to filter data.

1. Create a boolean Series (a condition):

```
condition = df['Age'] > 25
```

2. Pass that Series into the DataFrame:

```
# This gives you a new DataFrame
# containing only the rows where
# the condition was True.
df[condition]

# Or, all in one line:
df[df['Age'] > 25]
```

Result (from `condition.head()`):

```
0    False
1     True
2     True
3     True
4     True
Name: Age, dtype: bool
```

Result (from `df[df['Age'] > 25].head(2)`):

	PassengerId	Survived	Pclass	Name	Sex	Age
1	2	1	1	Cumings, Mrs...	female	38.0
2	3	1	3	Heikkinen, ...	female	26.0

Combining Conditions

Use `&` (AND) and `|` (OR)

You must use `&` and `|`, not `and` or `or`.

You must wrap each condition in parentheses `( )`.

```
# Condition 1: Age > 25
cond1 = df['Age'] > 25

# Condition 2: Sex is 'female'
cond2 = df['Sex'] == 'female'

# Get all females over 25
df[cond1 & cond2]

# All in one line:
df[(df['Age'] > 25) & (df['Sex'] == 'female')]
```

Result (from one-liner):

```
    PassengerId  Survived  Pclass   Name           Sex   Age
1             2         1       1   Cumings, Mrs... female  38.0
2             3         1       3   Heikkinen, ...  female  26.0
3             4         1       1   Futrelle, Mrs... female  35.0
...
```

Module 4

Manipulating & Cleaning Data

Creating a New Column

Simple Assignment

Create a new column just by assigning values to it. It's that simple.

```
# Create a 'AgeInDays' column
df['AgeInDays'] = df['Age'] * 365

# Create a 'FamilySize' column
df['FamilySize'] = df['SibSp'] + df['Parch']
```

Result (with 'FamilySize'):

```
    Name            SibSp  Parch  FamilySize
0   Braund, Mr...       1      0           1
1   Cumings, Mrs...     1      0           1
2   Heikkinen, ...      0      0           0
3   Futrelle, Mrs...    1      0           1
4   Allen, Mr...        0      0           0
...
```

Modifying with `.apply()`

Perform Complex Operations

`.apply()` lets you run a custom function on every value in a column.

Often used with lambda (anonymous) functions.

```
# Get the first name from the 'Name'
# (This is a bit complex, just see the pattern)
df['FirstName'] = df['Name'].apply(
    lambda x: x.split(',')[1].split('.')[0].strip()
)

# Convert 'Sex' to 0s and 1s
df['Sex_numeric'] = df['Sex'].apply(
    lambda x: 1 if x == 'female' else 0
)
```

Result (from 'Sex_numeric'):

```
    Name              Sex  Sex_numeric
0   Braund, Mr...      male            0
1   Cumings, Mrs...  female            1
2   Heikkinen, ...   female            1
3   Futrelle, Mrs... female            1
4   Allen, Mr...       male            0
...
```

Handling Missing Data: Finding It

Use `.isnull()` or `.isna()`

These methods return a DataFrame of `True`/`False` values.

Chain it with `.sum()` to get a count of missing values per column. (This is a key trick!)

```
# Get a DataFrame of True/False
df.isnull()

# Get a count of missing values per column
df.isnull().sum()
```

Result (from `df.isnull().sum()`):

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

We can see 'Age' is missing 177 values and 'Cabin' is missing 687.

Handling Missing Data: Dropping It

`df.dropna()`

The "easy" way out. Be careful, you might lose a lot of data!

```
# Drop any ROW that has AT LEAST ONE missing value
df.dropna()

# Drop any COLUMN that has missing values
# (axis=1 means columns)
df.dropna(axis=1)

# Only drop rows where 'Age' is missing
df.dropna(subset=['Age'])
```

Note: These operations are not "in-place" by default. They return a new DataFrame. To change the original, use `inplace=True`.

Before: (891, 12)

After `df.dropna()`: (183, 12)

We lost 708 rows! Maybe dropping isn't the best idea.

Handling Missing Data: Filling It

`df.fillna()`

A much safer and more common approach. Fill missing values with something else.

```
# Fill missing 'Age' with the mean age
mean_age = df['Age'].mean()
df['Age'].fillna(mean_age, inplace=True)

# Fill missing 'Embarked' with the mode
mode_embarked = df['Embarked'].mode()[0]
df['Embarked'].fillna(mode_embarked, inplace=True)

# Fill missing 'Cabin' with a placeholder
df['Cabin'].fillna('Unknown', inplace=True)
```

After `df.isnull().sum()`:

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
...
Cabin          0
Embarked       0
dtype: int64
```

Renaming Columns

`df.rename()`

Useful for cleaning up messy column names from a file.

```
# Rename specific columns using a dictionary
df.rename(columns={
    'Pclass': 'PassengerClass',
    'SibSp': 'SiblingSpouseCount'
}, inplace=True)
```

A common trick is to rename all columns to lowercase and replace spaces with underscores `_` for easier access.

Result (from `df.columns`):

```
Index(['PassengerId', 'Survived',
       'PassengerClass', 'Name', 'Sex', 'Age',
       'SiblingSpouseCount', 'Parch', 'Ticket', 'Fare',
       'Cabin', 'Embarked'],
      dtype='object')
```

Module 5

Combining DataFrames

Combining: `pd.concat()` (Stacking)

DataFrame A

	Name	Age
0	Alice	25
1	Bob	30

DataFrame B

	Name	Age
0	Charlie	22
1	David	40

Result (axis=0)

`pd.concat([df_a, df_b])`

	Name	Age
0	Alice	25
1	Bob	30
0	Charlie	22
1	David	40

Combining: `pd.merge()` (Joining)

The SQL-style JOIN

This is for combining DataFrames that share a common key (or keys).

```
# DataFrame with user info
df_users = pd.DataFrame({
    'user_id': [1, 2, 3],
    'name': ['Alice', 'Bob', 'Charlie']
})

# DataFrame with their orders
df_orders = pd.DataFrame({
    'order_id': [101, 102, 103],
    'user_id': [1, 3, 1],
    'product': ['Apple', 'Book', 'Banana']
})

# Merge them on the 'user_id' key
pd.merge(df_users, df_orders, on='user_id')
```

Result:

```
   user_id     name  order_id   product
0        1    Alice       101     Apple
1        1    Alice       103    Banana
2        3  Charlie       102      Book
```

Notice Bob (user_id 2) is gone. This is an INNER join by default.

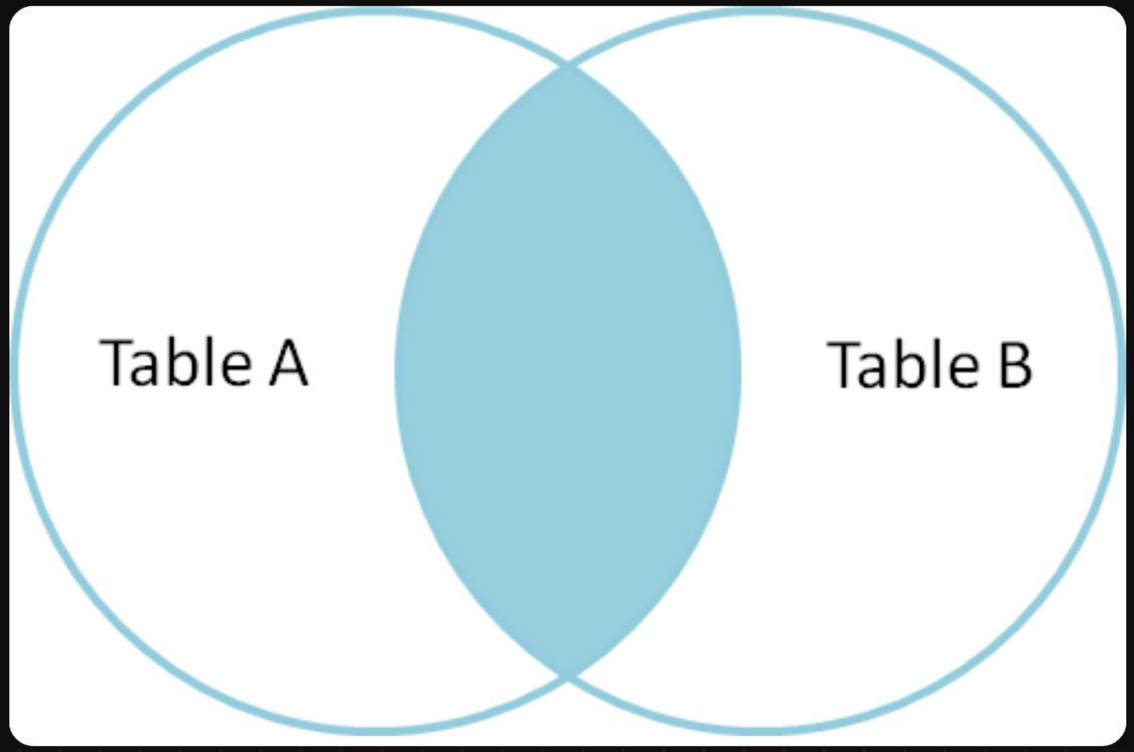
Merge Types (`how`)

How to Join?

You can control the join logic using the `how` parameter:

- ✓ `'inner'` (default): Only keep matching keys from both.
- ✓ `'left'`: Keep all keys from the left DataFrame.
- ✓ `'right'`: Keep all keys from the right DataFrame.
- ✓ `'outer'`: Keep all keys from both DataFrames.

```
# Keep all users, even if they have no orders
pd.merge(df_users, df_orders,
         on='user_id', how='left')
```



An INNER join only includes the overlapping data.

Module 6

Analysis & Aggregation

This is the payoff: getting answers from your data.

The `groupby()` Operation

1. Split

Group the DataFrame into smaller chunks based on a category (e.g., 'Sex' or 'Pclass').

2. Apply

Apply an aggregation function (like `.sum()`, `.mean()`, `.count()`) to each chunk.

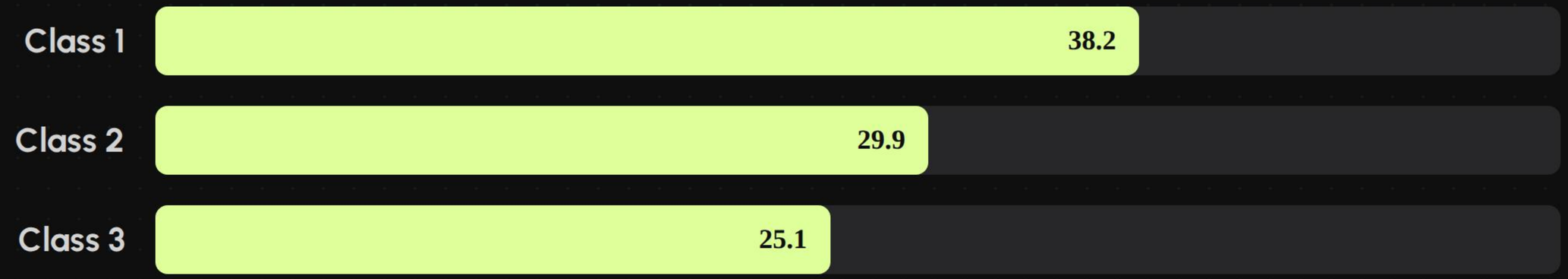
3. Combine

Combine the results from each chunk back into a new, smaller DataFrame.

`groupby()` in Practice

```
# Get the average age for each 'PassengerClass'
df.groupby('PassengerClass')['Age'].mean()

# Get the total survival count for each 'Sex'
df.groupby('Sex')['Survived'].sum()
```

Average Age by Passenger Class



The `groupby()` operation allows us to quickly see that 1st class passengers were, on average, older than 2nd and 3rd class passengers.

Useful Methods: `.value_counts()` & `.sort_values()`

`.value_counts()`

The fastest way to get frequency counts for a column. A `groupby().count()` shortcut.

```
# How many passengers of each sex?
df['Sex'].value_counts()
```

Result:

```
male      577
female    314
Name: Sex, dtype: int64
```

`.sort_values()`

Sort your DataFrame by one or more columns.

```
# Sort by Age, from oldest to youngest
df.sort_values(by='Age', ascending=False)
```

Result (abbreviated):

```
     PassengerId   Name          Sex    Age
630          631   Barkworth, Mr... male   80.0
851          852   Svensson, Mr...  male   74.0
...
```

Module 7

Visualization & Saving

Pandas has a built-in `.plot()` method (which uses Matplotlib) for quick and easy visualizations.

```
# Quick histogram of the 'Age' column
df['Age'].plot(kind='hist', bins=20)

# Quick bar chart of survival counts
df['Survived'].value_counts().plot(kind='bar')

# Quick line chart (if data is time-series)
# stock_df['Price'].plot(kind='line')
```

Saving Your Work

Export to a File

After all your hard work, save your cleaned and analyzed data to a new file.

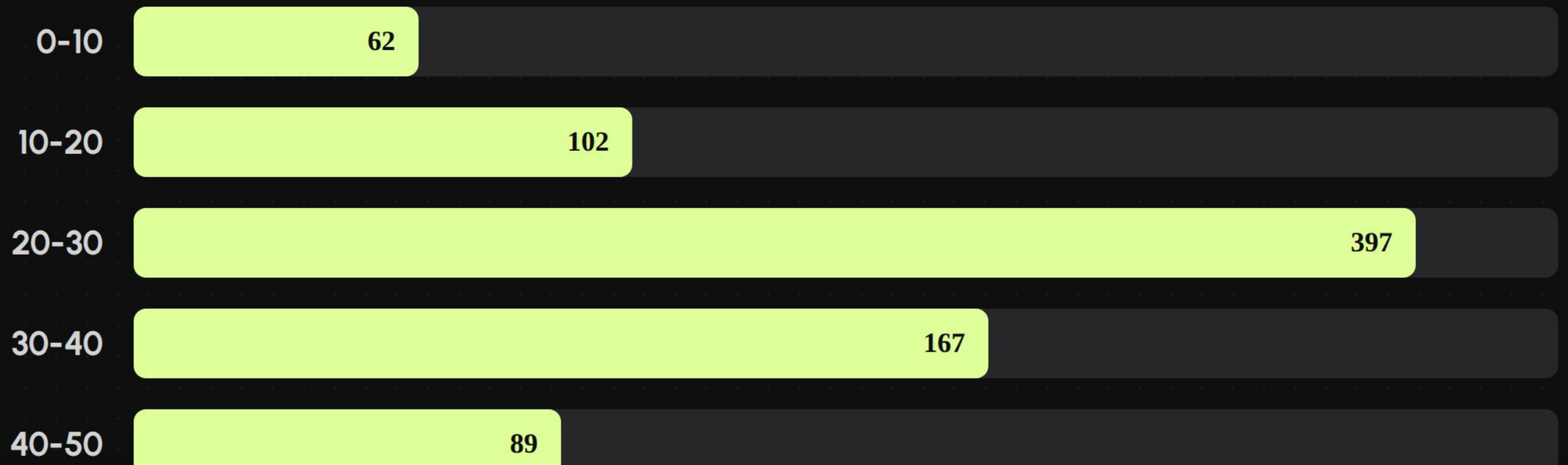
```
# Save to a new CSV file
# index=False is VERY important!
# (Prevents saving the pandas index
# as a new column)
df.to_csv('cleaned_titanic_data.csv', index=False)

# You can also save to Excel
df.to_excel('cleaned_titanic_data.xlsx', index=False)
```

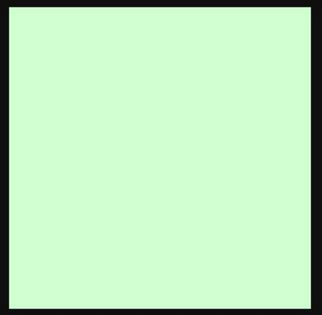
Your work is now saved!

Let's combine what we've learned:

1. Load 'titanic.csv'
2. Fill missing 'Age' values with the mean
3. Create a 'FamilySize' column
4. Find the survival rate for males vs. females (groupby)
5. Plot the 'Age' distribution (histogram)

Age Distribution (Histogram)





Questions?

Next Steps: Advanced Pandas, Seaborn/Plotly, and Machine Learning!

Image Sources

https://i.ytimg.com/vi/9PMELyEljuo/hq720.jpg?sqp=-oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAAGAElAADIQj0AgKJD&rs=AOn4CLApA_MgRyMmgseqVHIEXIR3Yw2UJg

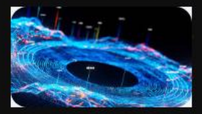
Source: www.youtube.com

https://blog.codinghorror.com/content/images/2025/03/6a0120a85dcdae970b012877702708970c-pi.png

Source: blog.codinghorror.com

https://media.istockphoto.com/id/1494107968/vector/question-and-answer-bubble-like-qa-icon.jpg?s=612x612&w=0&k=20&c=58SOMMQca4o5a106W7r3IYe2lGWMfQ5MilrqnIz6Cw8=

Source: www.istockphoto.com



https://media.istockphoto.com/id/2153711283/photo/abstract-data-flow-background.jpg?s=612x612&w=0&k=20&c=tKBt5lPfz6smBlMDoDtHsZy02CY9jgzBksDh7FWiumM=

Source: www.istockphoto.com