Data Management with Pandas

Agenda

01

CRUD Operations: **C**reate, **R**ead, **U**pdate, and **D**elete data.

02

Indexing and Filtering: Advanced selection with .loc, .iloc, and boolean logic.

03

DataFrame Reshaping: How to rename columns and reindex your DataFrame.

04

Data Export: Saving your work to CSV and Excel files for reporting

Data Management: CRUD

C – Create

R – Read

U – Update

D - Delete

Create

- pd.DataFrame())
- Create a new DataFrame
- Add new rows or columns.

Read

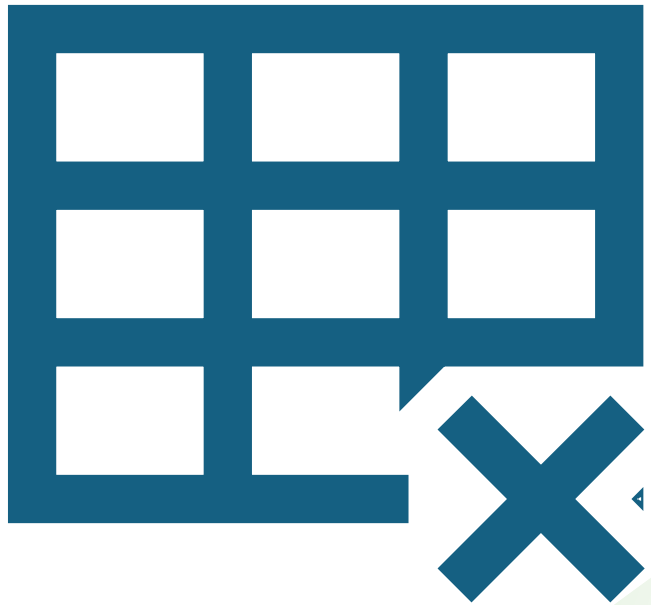
- Load data from a file (pd.read_csv()).
- Select and view data (df.head(), df['col'], .loc[]).

Update

Modify existing values in rows or columns

Delete

- Remove rows or columns (.drop()).



CRUD: **C**reate

- df = pd.DataFrame({'Name': ['Alice', 'Bob'], 'Score': [85, 90]})
- print(f"Original:\n{df}\n")

- # 1. Create (Add) a new column
- df['Grade'] = ['B', 'A']
- print(f"After adding 'Grade' column:\n{df}\n")

- # 2. Create (Add) a new row

- new_row = pd.DataFrame({'Name': ['Charlie'], 'Score': [78], 'Grade': ['C']})
- df = pd.concat([df, new_row], ignore_index=True)

- print(f"After adding 'Charlie':\n{df}")

CRUD: **R**ead

- "Read" means accessing the data you want to see. This includes loading the file and selecting subsets.

- df.head()

CRUD: **R**ead

- Read (Select) a single column #
- names = df['Name’]

- Read (Select) specific rows and columns # We will cover .loc[] and .iloc[] in detail next!

CRUD: **U**pdate

- Update operations modify existing data. The best way is using .loc or .iloc to target data

- df.loc[2, 'Score'] = 82
- df.loc[2, 'Grade'] = 'B'

- Update multiple values using a condition
- # Give everyone 5 bonus points
- df['Score'] = df['Score'] + 5

CRUD: **D**elete

- The df.drop() method is used to remove rows or columns.
- You must specify the axis:
- axis=0: Delete rows (default).
- axis=1: Delete columns.

Hands-On Code: CRUD Operations

- data = {
- 'student_id': [101, 102, 103, 104],
- 'name': ['Sanjay', 'Priya', 'Rohan', 'Meera'],
- 'subject': ['Math', 'Science', 'Math', 'English'],
- 'score': [85, 92, 78, 88]
- }
- df_students = pd.DataFrame(data)

Task 1 (Create):Add a new column Pass where all students are True.

Task 2 (Update):Rohan (index 2) got his score re-checked. Update his score to 81.

Task 3 (Delete):The subject column is no longer needed. Delete it.Task

4 (Read):Print the final DataFrame.

Indexing: .loc vs. .iloc

- **.loc[] (Label-based)**
- Selects data by **Index Name** and **Column Name**
- Syntax: df.loc[row_label, column_label]
- Slicing loc['A':'C'] is **inclusive** (includes 'C').

Hands on: loc

- Using df_students from last exercise
- loc: Get Priya (index 1) and Rohan (index 2)
- for 'name' and 'score' columns

Indexing: .loc vs. .iloc

- **.iloc[] (Integer-based)**
- Selects data by **Row Position** and **Column Position** (starting from 0).
- Syntax: df.iloc[row_position, column_position]
- Slicing iloc[0:2] is **exclusive** (does *not* include 2).

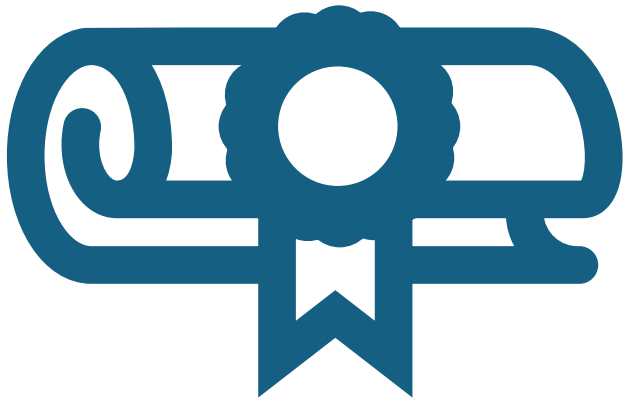
Hands on: iloc

- iloc: Get the same data using positions

- Rows 1 and 2, Columns 1 and 2

Advanced Filtering

- You can pass a "mask" of True/False values to select rows.
- Single Condition
- Multiple Conditions
- Filtering with a list using .isin()

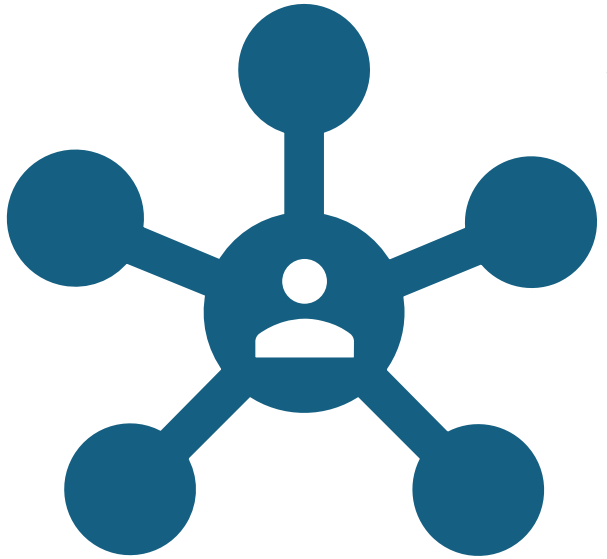


Advanced Filtering: Single Conditions

- Get all students with score > 85

- high_scorers = df_students[df_students['score'] > 85]
- print(f"High Scorers:\n{high_scorers}\n")

Multiple Conditions

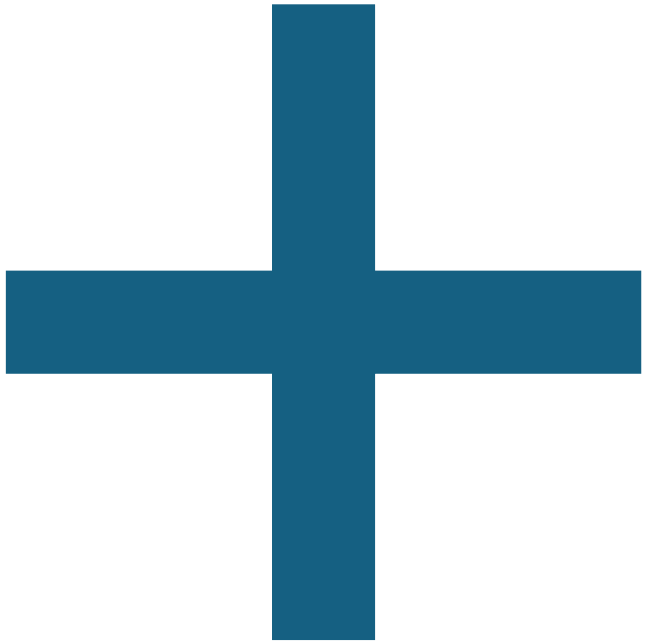


- Use & (AND) and | (OR)
- IMPORTANT: Use parentheses () around each condition
- Get Math students (index 0) with score > 80
- math_high = df_students[(df_students['name'] == 'Sanjay') & (df_students['score'] > 80)]

Filtering with a list using .isin()

Get only Sanjay and Meera

- names_list = ['Sanjay', 'Meera’]
- sanjay_meera = df_students[df_students['name'].isin(names_list)]
- print(f"Sanjay and Meera:\n{sanjay_meera}")



.apply()

- df.apply(func, axis=0)
- Apply a function along an axis of the DataFrame.

Reshaping: Rename Columns

Data cleaning often requires fixing column names.

To view all the column

- df_students.columns.to_list()

Rename specific columns

- Use a dictionary: {old_name: new_name}

Eg;

- df_renamed = df_students.rename(columns={ 'student_id': 'ID', 'score': 'Final_Score' })

Rename ALL columns at once

- df_new_cols.columns = ['ID', 'Student_Name', 'Score', 'Passed’]

- Must provide a name for every column

set_index()

Promotes a column to be the *new* index.

reset_index():

Turns the index back into a regular column.

Reshaping: Reindex

- Changes the row index labels. It adds NaN for any new labels and discards any old labels not included.

- Reorders
- Expands
- Shrinks

Hands-On Code

- **Task 1:**
- Rename the name column to Student_Name and the score column to Grade_Points.
- **Task 2:**
- Set the student_id column as the new index for the DataFrame.
- **Task 3:**
- After setting the index, print the final DataFrame.

Export Data (CSV & Excel)

- save and share your results.

Export to CSV

- index=False is VERY important!
- It stops Pandas from saving the index as a new column.

- df_final_indexed.to_csv('student_report.csv')

Export to Excel

- df_final_indexed.to_excel('student_report.xlsx', sheet_name='Final Grades')