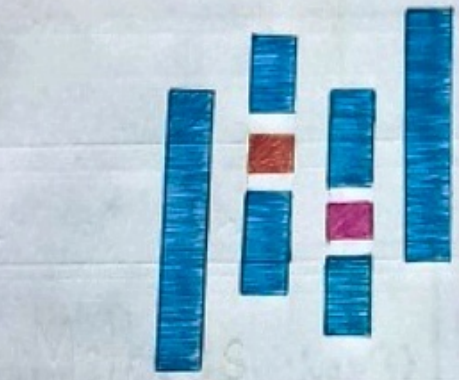
Important Pandas Methods For Data Science

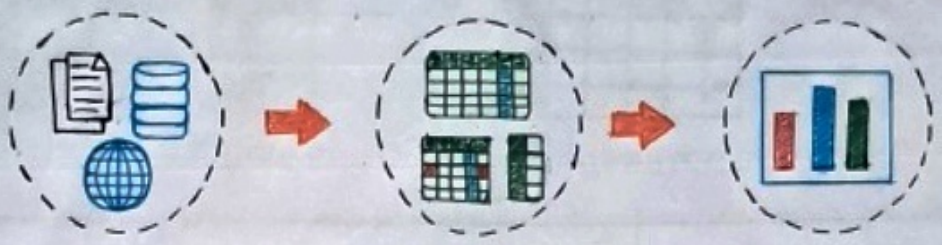
Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.



pandas

Pandas setup

Pandas is open source pakage for data science / data analysis.



pip install pandas	Install Pandas library
import pandas as pd	Import Pandas
df = (pd.melt(df) .rename(columns={'variable':'var'}) .query('val >= 200'))	Method Chaining
df[(df['col_1']=='a')&(df['col_2']>=10)]	Pandas syntax

Data Structure

Pandas has two data structures: Series (1 dimension) and DataFrame (multi dimensional).

Series

DataFrame

`S = pd.Series(['a','b','c'], index=[0,1,2])`	Create new Series
`df = pd.DataFrame({'col_1':[11,12,13], 'col_2':[21,22,23], 'col_3':[31,32,33]}, index=[0,1,2])`	Create new DataFrame

Read

Import data from CSV, Excel, JSON, SQL, HTML, Web.

`pd.read_csv(filename)`	From a CSV file
`pd.read_csv(filename, header=None, nrows=5)`	From a csv file with Parameters
`Pd.read_excel(filename)`	From an Excel file
`Pd.read_sql(query, connection_object)`	Reads from a SQL table/ database
`pd.read_json(json_string)`	Reads from a JSON formatted string, URL or file

Write

Write data to CSV, Excel, JSON, HTML.

df.to_csv (filename)	Writesto a CSV file
df.to_excel(filename)	Writes to a Excel file
df.to_json (filename)	Writesto a file in JSON format
df.to_html (filename)	Saves as an HTML table

Inspect Data

View stats, samples and summary of the data.

describe() →

count	100
mean	5
std	10
min	1

df.head (n)	First n rows
df.tail (n)	Last n rows
df. shape	Number of rows and columns
df. info()	Index, Datatype and Memory information
df. describe()	Summary Statistics for numerical columns
s. value_counts(dropna=False)	(Series) Views unique values and counts
df. sample (n)	Randomly select n rows.
df.nlargest(n, 'col_1')	Select and order top n entries for column
df.nsmallest(n, 'col_1')	Select and order bottom n entries.
df.quantile([0.25, 0.75])	Quantiles of each object

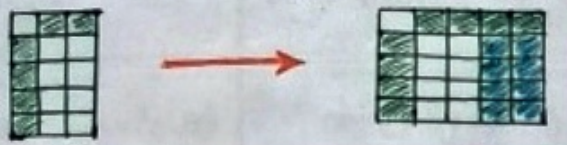
Select

Select data by index, by label, get subset.

S.loc[0]	(Series) Select by index
S.iloc[0]	(Series) Select by position
df['col_1']	Get single column as series
df[['col_1', 'col_2']]	Get multiple columns as a DataFrame
df.iloc[0, :]	Select first row from DataFrame
df.iloc[0, 0]	First element of first column
df.loc[df['col_1'] > 10, ['col_1', 'col_2']]	Select rows meeting logical condition, and only the specific columns
df.iat[1, 2]	Access single value by index
df.at[3, 'col_2']	Access single value by label

Add rows / columns

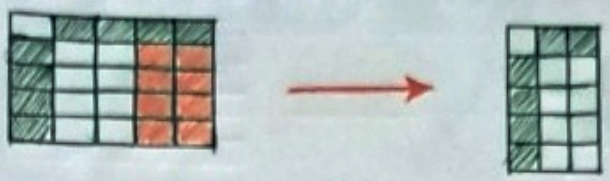
Add new values to existing DataFrame.



df['new col'] = df['col'] * 100	Add new column based on other column
df['new col'] = False	Add new column single value
df.loc[-1] = [1, 2, 3]	Add new row at the end of DataFrame
df.append(df2, ignore_index = True)	add rows from DataFrame to existing DataFrame

Drop rows/columns/nan

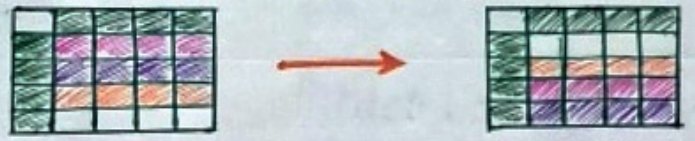
Drop data from DataFrame.



S.drop([0,1])	(Series) Drop values from Series by index(row axis)
df.drop('col_1', axis=1)	Drop column by name col_1 (column axis)
df.dropna()	Drop all rows that contain null values
df.dropna(axis=1)	Drop all columns that contain null values
df.dropna(axis=1, thresh=n)	Drop all rows have less than n non null values

Sort values / index

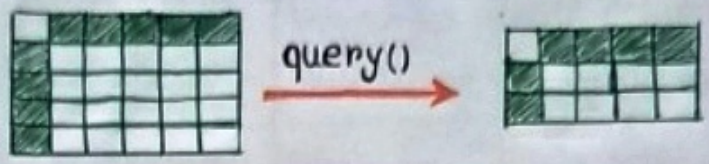
Sort and rank values/index by one or multiple criteria.



df.sort_values(by='col_1', ascending=False)	Sort values by column, ascending order
df.sort_values(by=['col_1', 'col_2'])	Sort values by columns
df.sort_index(ascending=False)	Sort object by labels (along an axis) in descending order
df.sort_values(by=[('col_1', 'col_2')])	Sort multiple levels
df.reset_index()	Reset the index of the DataFrame, moving index to columns

Filter

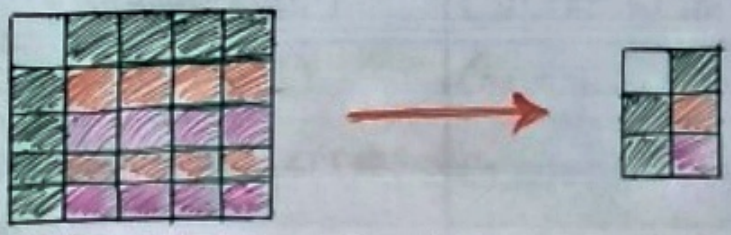
Filter data based on multiple criteria.



df[df['col_1']>100]	Values greater than x
df[(df['col_1']=='a')&(df['col_2']>=10)]	Filter Multiple Conditions - & - and; | - or
df[df['date']>'2022-02-22']	Date filtering
df[df['date'].dt.month==2]	Filter with dt attributes
df[df['col_1'].str.contains('pan*', regex=True)]	Filter by regex
df[df['col_1'].isin(['pan', 'das'])]	Filter based on list of values
df.query('col_1>100')	Filter by queries
df.query('col_1>100 and col2=0')	Filter by multiple queries

Group by

Group by and summarize data.



Group by

df.groupby('col_1')	Group by single column-return Pandas.core.groupby.DataFrameGroupBy
df.groupby(['col_1', 'col_2'])	Group by multiple columns
df.groupby('col_1').groups	View groups
df.groupby('col_1').get_group(1)	Get group
df.groupby('col_1').count()	Get count per groups
df.groupby('col_1').agg([np.sum, np.mean])	Apply multiple agg functions on group
df.groupby('col_1').filter(lamba x : len(x)>=5)	Filter groups
df.groupby('col_1').agg('count')	Aggregate group using function.
df.groupby('col_1').rank(method='dense')	Compute numerical data ranks (1 through n) along axis.

Convert

Convert to date, string, numeric.

String ⟶ date

String ⟶ number

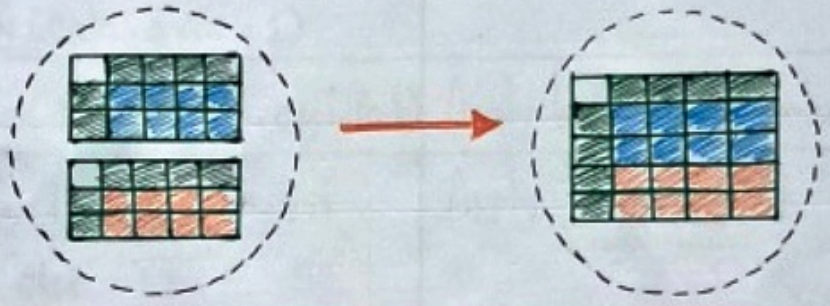
df['points'].astype(str)	Convert to string
df['col_1'].astype('int64')	Convert to int64
df['col_1'].astype(float)	Convert to float
Pd.to_numeric(df['col_1'], errors='coerce')	Convert to numeric

Convert

`Pd.to_datetime(df['date'], format = '%y-%m-%d')`	convert string to date
`Pd.to_DataFrame(df['values'].tolist(), columns = ['col_1','col_1'])`	split column list to multiple columns
`df['col_1'].apply(Pd.series)`	Expand Series of dictionnaries

Merge & Concat

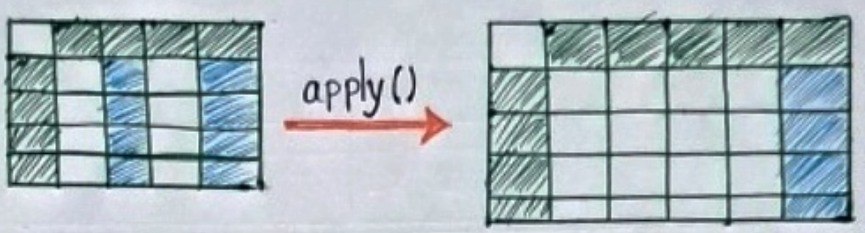
Merging, joining and concatenating 2 and more DataFrames.



`df1.append(df2)`	Adds the rows in df1 to the end of df2 (columns should be identical)
`pd.concat([df1,df2], axis = 1)`	Adds the columns in df1 to the end of df2 (rows should be identical)
`df1.join(df2,on=col1, how='inner')`	SQL-style joins the columns in df1 with the columns on df2 where the rows for col have identical values. how can be one of 'left', 'right', 'outer', 'inner'

Apply

Applying functions to a column or DataFrame; lambda functions.



`def calc(x):`
`  return x + 1`
`df.apply(calc, axis = 1)`
	Apply function to DataFrame

`df[['col_1','col_2']].apply(calc)`	Apply to multiple columns
`df.apply(lamba x: x*-1 if x < 0 else x)`	Apply lamba