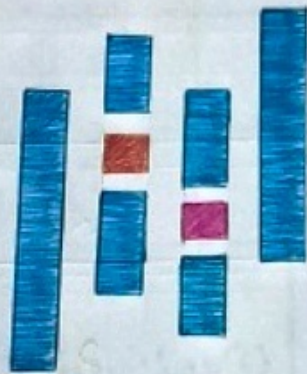


# Important Pandas Methods For Data Science

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.

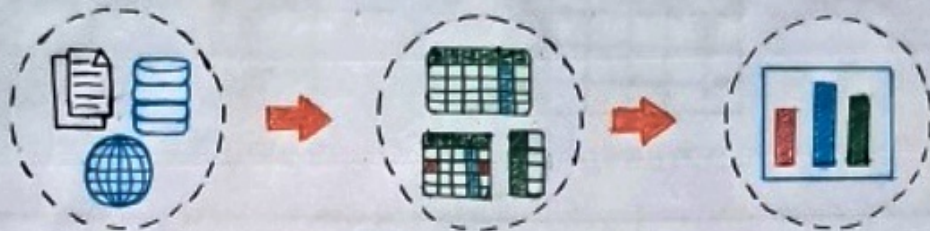


# pandas



## Pandas setup

Pandas is open source package 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_ob-  
ject)
```

Reads from a SQL table/  
database

```
pd.read_json(json_string)
```

Reads from a JSON format-  
ted string, URL or file



## Write

Write data to CSV, Excel, JSON, HTML.

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

## Inspect Data

View stats, samples and summary of the data.



`describe()` →

count 100  
mean 5  
std 10  
min 1

<code>df.head(n)</code>	First n rows
<code>df.tail(n)</code>	Last n rows
<code>df.shape</code>	Number of rows and columns
<code>df.info()</code>	Index, Datatype and Memory information
<code>df.describe()</code>	Summary statistics for numerical columns
<code>s.value_counts(dropna=False)</code>	(Series) Views unique values and counts
<code>df.sample(n)</code>	Randomly select n rows.
<code>df.nlargest(n, 'col-1')</code>	Select and order top n entries for column
<code>df.nsmallest(n, 'col-1')</code>	Select and order bottom n entries.
<code>df.quantile([0.25, 0.75])</code>	Quantiles of each object



## Select

Select data by index, by label, get subset.



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

## Add rows / columns

Add new values to existing DataFrame.

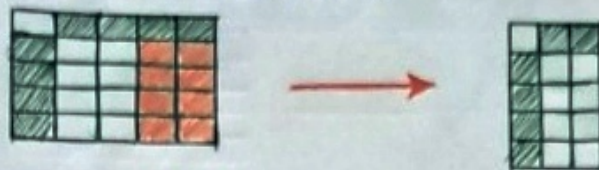


<code>df['new col'] = df['col'] * 100</code>	Add new column based on other column
<code>df['new col'] = False</code>	Add new column single value
<code>df.loc[-1] = [1, 2, 3]</code>	Add new row at the end of DataFrame
<code>df.append(df2, ignore_index = True)</code>	add rows from DataFrame to existing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



## Drop rows/columns/nan

Drop data from DataFrame.



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

## Sort values / index

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

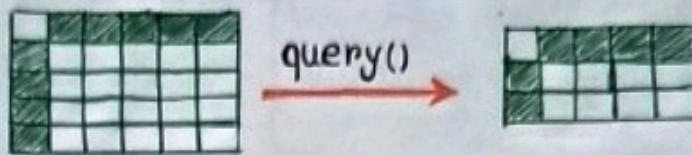


<code>df.sort_values(by='col-1', ascending=False)</code>	Sort values by column, ascending order
<code>df.sort_values(by=['col-1', 'col-2'])</code>	Sort values by columns
<code>df.sort_index(ascending=False)</code>	Sort object by labels (along an axis) in descending order
<code>df.sort_values(by=[('col-1', 'col-2')])</code>	Sort multiple levels
<code>df.reset_index()</code>	Reset the index of the DataFrame, moving index to columns



## Filter

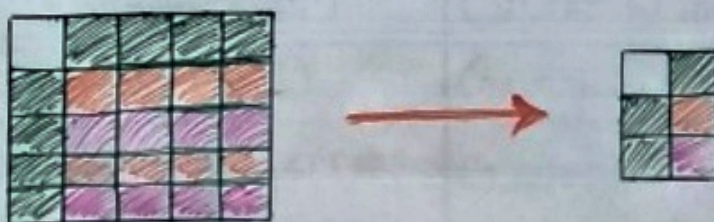
Filter data based on multiple criteria.



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

## Group by

Group by and summarize data.





## Group by

<code>df.groupby('col-1')</code>	Group by single column-return Pandas.core.groupby.DataFrameGroupBy
<code>df.groupby(['col-1', 'col-2'])</code>	Group by multiple columns
<code>df.groupby('col-1').groups</code>	View groups
<code>df.groupby('col-1').get_group(1)</code>	Get group
<code>df.groupby('col-1').count()</code>	Get count per groups
<code>df.groupby('col-1').agg([np. sum, np.mean])</code>	Apply multiple agg functions on group
<code>df.groupby('col-1').filter (lambda x: len(x) &gt;= 5)</code>	Filter groups
<code>df.groupby('col-1').agg('count')</code>	Aggregate group using function.
<code>df.groupby('col-1').rank(method ='dense')</code>	Compute numerical data ranks (1 through n) along axis.

## Convert

Convert to date, string, numeric.

String → date  
String → number

<code>df['points'].astype(str)</code>	Convert to string
<code>df['col-1'].astype('int64')</code>	Convert to int64
<code>df['col-1'].astype(float)</code>	Convert to float
<code>Pd.to_numeric(df['col-1'], errors= 'coerce')</code>	Convert to numeric

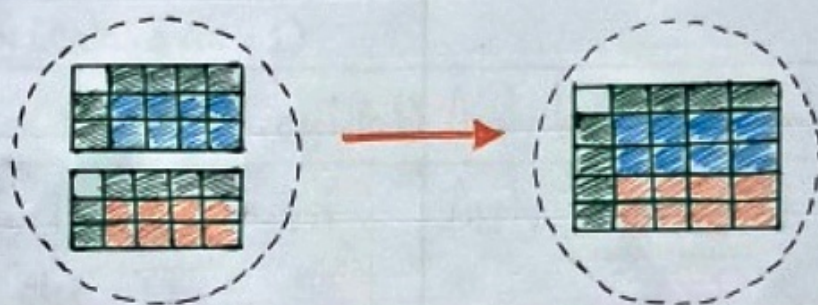


## Convert

<code>pd.to_datetime(df['date'], format = '%Y-%m-%d')</code>	convert string to date
<code>pd.DataFrame(df['values']. tolist(), columns = ['col.1', 'col.1'])</code>	split column list to multiple Columns
<code>df['col.1'].apply(pd.Series)</code>	Expand Series of dictionaries

## Merge & Concat

Merging, joining and concatenating 2 and more DataFrames.

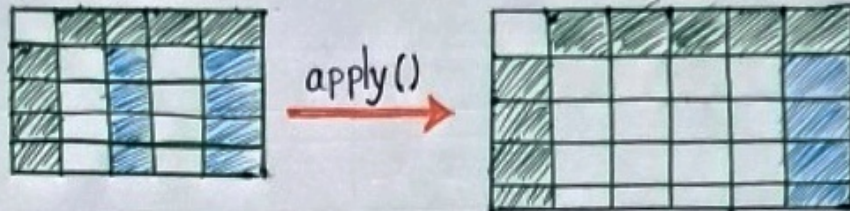


<code>df1.append(df2)</code>	Adds the rows in df1 to the end of df2 (columns should be identical)
<code>pd.concat([df1, df2], axis = 1)</code>	Adds the columns in df1 to the end of df2 (rows should be identical)
<code>df1.join(df2, on = col1, how = 'inner')</code>	SQL-style joins the columns in df1 with 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.



```
df.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(lambda x: x*-1 if  
x < 0 else x)
```

Apply lambda

