PANDAS

- Python package named Pandas is used for data manipulation and even visualization.
  Pandas is built on top of numpy and matplotlib.

  In pandas data is represented as Dataframe object. Here every column has same data type

- Inspecting a Dataframe
  Let's have data employees save as dataframe

  1) .head() → returns first few rows
     eg employees.head()

  2) .info() → shows information on each of the columns, such as data type and number of missing values.

  3) .shape → .shape component returns number of rows and columns
  4) .describe() → calculates few summary statistics for each column.

- 3 components of Dataframe

  1) .values :- returns a two dimensional NumPy array of values
  2) .columns :- An index of columns : the column names
  3) .index :- An index for rows : either row numbers or names.

- Add column
  1) add new column analyse with value salary /age
  employee ['analyse'] = employee['salary'] / employee['age']

Sorting & Subnetting

1) Some employee df with respect to column name

```
ans = employee . sort_values ('name')
```

2) in descending order

```
ans = employee . sort_values ('name' , ascending = False)
```

3) sort name (ascending) & region (descending)

```
ans = employee . sort_values ( ['name', 'region'], ascending =[True ,False])
```

Subnetting

4) Select name column only

```
ans = employee ['name']
print (ans . head())
```

5) Select n columns

```
ans = employee [ [" name", "region", "salary"] ]
```

Subnetting rows

6) Select rows with age > 45

```
ans = employee [employee ['age'] > 45]
```

7) Select rows with age > 45 and region India

```
ans = employee [(employee ['age'] > 45) & (employee ['region'] == "India")]
```

8) Select rows with region Asia , europe

```
ans = employee [( employee ['region']. isin (['Asia', 'Europe']))]
```

Aggregating data

Now lets have sales dataframe

1) Get mean of weekly sales

Sales ['weekly-sales'] . mean()

Similarly :- . median () , . mode () , . min () , . max ()
. var () , . std () , . sum () , . quantile ()

2) apply custom functions eg apply IQR on temperature
. agg

```
def iqr ( columns ):
    return columns . quantile (0.75) - columns . quantile (0.25)
```

- Sales ['temperature'] .agg (iqr)

apply on 2 columns

Sales [['temperature', temperature_day']] . agg (iqr)

apply 2 different functions

Sales ['temperature'] . agg ([iqr, pcr , np.mean , np.median])

3) using cummulative functions

1) cumsum ()   2) cum min ()   3) cumprod ()

eg Sales ['weekly-sales'] . cumsum () .

COUNTING

1) remove duplicates , eg remove duplicate stores.

Sales . drop _ duplicates (subset = ["store" , "department"])

2) Count number of store of each type

Store ['type'] . value _ counts ()

- in sorted form Store ['type'] . value _counts (sort = True)
- get proportions of store of each type Store ['type'] . value _counts (normalize = True)

Grouping

1) subset by type A store & find total weekly sales.

```
Sales_A = Sales [ Sales [' type'] = = "A"] ["weekly_sales"]. sum ()
```

again for type B , type C

using group by to find for all types

```
Sales_by_type = Sales . groupby ( "type" ) ["weekly_sales"] . Sum ()
```

2) get multiple columns & apply multiple function

```
Sales . groupby ('type') [["dob", "salary"]] . agg ([np.min , np.max , np. mean])
```

Pivot tables

Pivot tables are standard way of aggregating data in spreadsheets . In pandas pivot tables are essentially just another way of performing grouped calculations . That is , the .pivot_table () method is just alternative of .groupby()

default function applied is mean

1) get mean by type

```
Sales . groupby ('type') ['weekly_sales'] . mean ()
    ≡ Sales. pivot_table (values = "weekly_sales", inden = 'type')
```

2) apply other functions

```
Sales . pivot_tables (values = "weekly_sales", inden = 'type' , aggfunc = [np.mean , np.median])
```

3) group by 2 values eg for dogs → by color , breed

```
dogs . pivot_tables (values = "weight" , inden = "color" , columns = "breed")
```

4) fill empty values with 0 & find sum of all non-zero columns as a seperate column

```
Sales . pivot_tables (values = "weekly_sales" , inden = type" , fill_value = 0 ,
        margin = True)
```

Slicing & Indexing

• Explicit Index

1) Set a column as index → dogs.set_index("name")
   or dogs.set_index(["name", breed"])

2) Removing an index → dogs.realt_index()
   # to drop it → dogs.reset_index(drop=True)

3) Subsetting becomes easier after indexing
   eg if name is index → dogs.log[['Bello', 'Stello']]

index value don't need to be unique.

4) Sort by index value
   dog.sort_index()
   # if 2 indexes → dog.sort_index(level=['color', 'breed'], ascending=[True, False])

• Slicing & Subsetting

1) if indexes are normal → breeds[2:5] , breed[:3] , breed[:, 3:5]
   ↑
   5 excluded.

2) if indexed by name → dogs.loc["Chow": "Poodle"]
   ↑
   included

3) Slicing columns → dogs.loc[:, "name":"height_cm"]

4) Subsetting indexed one with row/column number
   dogs.iloc[2:5, 1:4]

Working with Pivot table

- if pivot has index = "breed"

  so now can use loc

  ```
  dogs .loc ["Chow Chow" : "Poodle"]
  ```

Can also find mean about a axis

```
dogs. mean (axis ="index")
dogs . mean (axis = "columns")
```

Analyzing the data

1) Histograms

make histogram for heights

⇒

```
import matplotlib . pyplot as plt.
dogs ["height_cm"] . hist ()
plt .show()
```

- adjust no. of bars.

  ```
  dogs ["height _cm"] . hist (bins = 20)
  ```

- 2 histograms together

  alpha = opacity .

  ```
  dag [dog ["breed"] == "Poodle"] ["height_cm"] . hist (alpha = 0.7)
  dog [ dog ["breed"] == "Chow Chow"] ["height_cm"] . hist (alpha =0.7)
  plt .legend (["F", "M"])
  plt . show ()
  ```

Bar Plot

```
avg_weight_by_breed = dog.groupby("breed")["weight_kg"].mean()
avg_weight_by_breed.plot(kind="bar", title="Weight by Dog Breed")
plt.show()
```

Line Plot (betn 2 columns)

```
dog.plot(x="date", y="weight_kg", kind="line")
plt.show()
```

Scatter plot

```
dog.plot(x="height_cm", y="weight_cm", kind="scatter")
plt.show()
```

Missing Data

in Pandas missing value is denoted as NaN (not a number)

1) detecting missing datas

```
dogs.isna()
```

returns True for missing data

2) detect columns having any missing data

```
dogs.isna().any()

dogs.isna().sum()          # count of missing data in columns.
```

3) Removing Missing data

```
dogs.dropna()
```

drop rows with missing data.

4) Replace missing data with a value

```
dogs.fillna(0)
```

Creating Dataframes

1)
```
List_of_dict = [
    {"name" : "Ginger" , "breed = "Dach" }
    {"name" : " Scout" , "breed" : "Dalmation" {
]

new dogs = pd.DataFrame (list_of_dict)
```

2)
```
dict_of_list = {
    "name" : ["Ginger", "Scout"],
    "breed" : ['Dach", 'Dalmation"]
}
new_dogs = pd.DataFrame (dict_of_list)
```

Reading and writing CSV

1)
```
dogs = pd.read_csv ("new_dogs.csv")
```

2) Dataframe to CSV

```
dogs.to_csv ( "new_dogs.csv")
```