UNIT – III

Data Manipulation with Pandas :

Pandas is a Python library.

Pandas is used to analyze data.

What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called *cleaning* the data.

Where is the Pandas Codebase?

The source code for Pandas is located at this github repository https://github.com/pandas-dev/pandas

Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install pandas
```

If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

```
import pandas
```

Now Pandas is imported and ready to use.

Example

```
import pandas

mydataset = {
```

```
  'cars':                                   ["BMW", "Volvo", "Ford"],
  'passings':                                                   [3, 7, 2]
}

myvar                       =                   pandas.DataFrame(mydataset)

print(myvar)
```

Pandas as pd

Pandas is usually imported under the pd alias.

alias: In Python alias are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

```
import pandas as pd
```

Now the Pandas package can be referred to as pd instead of pandas.

Example

```
import pandas                   as                  pd

mydataset                       =                   {
  'cars':                                   ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar                       =                   pd.DataFrame(mydataset)

print(myvar)
```

Checking Pandas Version

The version string is stored under __version__ attribute.

Example

```
import pandas as pd

print(pd.__version__)
```

Pandas Series

What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

Example

Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

Example

Return the first value of the Series:

```
print(myvar[0])
```

Create Labels

With the index argument, you can name your own labels.

Example

Create your own labels:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

When you have created labels, you can access an item by referring to the label.

Example

Return the value of "y":

```
print(myvar["y"])
```

Pandas DataFrames

What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Example

Create a simple Pandas DataFrame:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
```

```
#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Result

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

Example

Return row 0:

```
#refer to the row index:
print(df.loc[0])
```

Result

```
calories    420
duration     50
Name: 0, dtype: int64
```

Named Indexes

With the index argument, you can name your own indexes.

Example

Add a list of names to give each row a name:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

Result

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

Locate Named Indexes

Use the named index in the loc attribute to return the specified row(s).

Example

Return "day2":

```
#refer to the named index:
print(df.loc["day2"])
```

Result

```
calories    380
```

```
duration    40
Name: 0, dtype: int64
```

Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example

Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd

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

print(df)
```

Aggregation in Pandas

Aggregation in pandas provides various functions that perform a mathematical or logical operation on our dataset and returns a summary of that function. Aggregation can be used to get a summary of columns in our dataset like getting sum, minimum, maximum, etc. from a particular column of our dataset. The function used for aggregation is agg(), the parameter is the function we want to perform.

Some functions used in the aggregation are:

Function Description:

- *sum()* *:Compute sum of column values*

- *min()* :Compute min of column values
- *max()* :Compute max of column values
- *mean()* :Compute mean of column
- *size()* :Compute column sizes
- *describe()* :Generates descriptive statistics
- *first()* :Compute first of group values
- *last()* :Compute last of group values
- *count()* :Compute count of column values
- *std()* :Standard deviation of column
- *var()* :Compute variance of column
- *sem()* :Standard error of the mean of column

Examples:

- The sum() function is used to calculate the sum of every value.

Examples:

- The sum() function is used to calculate the sum of every value.

- Python

```
df.sum()
```

Output:

```
Maths      24
English    20
Science    23
History    20
dtype: int64
```

- The describe() function is used to get a summary of our dataset

- Python

```
df.describe()
```

Output:

	Maths	English	Science	History
count	3.0	3.000000	3.000000	3.000000
mean	8.0	6.666667	7.666667	6.666667
std	1.0	3.055050	0.577350	2.081666
min	7.0	4.000000	7.000000	5.000000
25%	7.5	5.000000	7.500000	5.500000
50%	8.0	6.000000	8.000000	6.000000
75%	8.5	8.000000	8.000000	7.500000
max	9.0	10.000000	8.000000	9.000000

- We used agg() function to calculate the sum, min, and max of each column in our dataset.

- Python

```
df.agg(['sum', 'min', 'max'])
```

Output:

	Maths	English	Science	History
sum	24	20	23	20
min	7	4	7	5
max	9	10	8	9

Grouping in Pandas

Grouping is used to group data using some criteria from our dataset. It is used as split-apply-combine strategy.

- Splitting the data into groups based on some criteria.
- Applying a function to each group independently.
- Combining the results into a data structure.

Examples:

We use groupby() function to group the data on “Maths” value. It returns the object as result.

- Python

```
df.groupby(by=['Maths'])
```

Output:

<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000012581821388>

Applying groupby() function to group the data on “Maths” value. To view result of formed groups use first() function.

- Python

```
a = df.groupby('Maths')
a.first()
```

Output:

Maths	English	Science	History
7	6	8	5
8	10	7	6
9	4	8	9

First grouping based on “Maths” within each team we are grouping based on “Science”

- Python

```
b = df.groupby(['Maths', 'Science'])
b.first()
```

Output:

		English	History
Maths	Science		
7	8	6	5
8	7	10	6
9	8	4	9

Implementation on a Dataset

Here we are using a dataset of diamond information.

- Python

```
# import module
import numpy as np
import pandas as pd

# reading csv file
dataset = pd.read_csv("diamonds.csv")

# printing first 5 rows
print(dataset.head(5))
```

Output:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

- We group by using cut and get the sum of all columns.

- Python

```
dataset.groupby('cut').sum()
```

Output:

cut	carat	depth	table	price	x	y	z
Fair	1684.28	103107.1	95076.6	7017600	10057.50	9954.07	6412.26
Good	4166.10	305967.0	287955.9	19275009	28645.08	28703.75	17855.42
Ideal	15146.84	1329899.3	1205814.4	74513487	118691.07	118963.24	73304.61
Premium	12300.95	844901.1	810167.4	63221498	82385.88	81985.82	50297.49
Very Good	9742.70	746888.4	700226.2	48107623	69359.09	69713.45	43009.52

- Here we are grouping using cut and color and getting minimum value for all other groups.

- Python

```
dataset.groupby(['cut', 'color']).agg('min')
```

Output:

cut	color	carat	clarity	depth	table	price	x	y	z
Fair	D	0.25	I1	52.2	52.0	536	4.09	4.11	2.49
	E	0.22	I1	51.0	49.0	337	3.87	3.78	2.33
	F	0.25	I1	52.3	50.0	496	4.19	4.15	2.32
	G	0.23	I1	43.0	53.0	369	0.00	0.00	0.00
	H	0.33	I1	52.7	50.0	659	4.40	4.32	2.84
	I	0.41	I1	50.8	49.0	735	4.62	4.66	2.93
	J	0.30	I1	55.0	52.0	416	4.24	4.16	2.72
Good	D	0.23	I1	54.3	52.0	361	3.83	3.85	2.37
	E	0.23	I1	56.3	53.0	327	3.83	3.85	2.31
	F	0.23	I1	56.2	52.0	357	0.00	0.00	0.00
	G	0.23	I1	56.2	53.0	394	3.94	3.90	0.00
	H	0.25	I1	56.0	51.0	368	4.04	4.06	2.46
	I	0.30	I1	56.1	51.0	351	4.19	4.19	2.67
	J	0.28	I1	56.2	52.0	335	4.22	4.23	2.51
Ideal	D	0.20	I1	58.5	52.0	367	3.81	3.77	2.33
	E	0.20	I1	58.3	52.0	326	3.76	3.73	2.06
	F	0.23	I1	58.0	52.4	408	0.00	3.92	0.00
	G	0.23	I1	58.8	52.0	361	0.00	0.00	0.00
	H	0.23	I1	58.3	52.0	357	3.94	3.97	1.41
	I	0.23	I1	58.4	43.0	348	3.94	3.90	1.53
	J	0.23	I1	43.0	53.0	340	3.93	3.90	2.46
Premium	D	0.20	I1	58.0	52.0	367	0.00	0.00	0.00
	E	0.20	I1	58.0	52.0	326	3.79	3.75	2.24
	F	0.20	I1	58.0	51.0	342	3.73	3.71	0.00
	G	0.23	I1	58.0	52.0	382	3.95	3.92	0.00

	H	0.23	I1	58.0	51.0	368	0.00	0.00	0.00
	I	0.23	I1	58.0	52.0	334	3.97	3.94	0.00
	J	0.30	I1	58.0	54.0	363	4.22	4.21	2.59
Very Good	D	0.23	I1	57.5	52.0	357	3.86	3.85	2.35
	E	0.20	I1	57.7	44.0	352	3.74	3.71	2.25
	F	0.23	I1	56.9	52.0	357	3.84	3.86	2.36
	G	0.23	I1	57.1	52.0	354	3.88	3.92	2.36
	H	0.23	I1	56.8	52.0	337	0.00	0.00	0.00
	I	0.24	I1	57.5	52.0	336	3.95	3.98	2.46
	J	0.24	I1	57.6	51.6	336	3.94	3.96	2.48

- Here we are grouping using color and getting aggregate values like sum, mean, min, etc. for the price group.

- Python

```
# dictionary having key as group name of price and
# value as list of aggregation function
# we want to perform on group price
agg_functions = {
   'price':
   ['sum', 'mean', 'median', 'min', 'max', 'prod']
}

dataset.groupby(['color']).agg(agg_functions)
```

Output:

	price					
	sum	mean	median	min	max	prod
color						
D	21476439	3169.954096	1838.0	357	18693	inf
E	30142944	3076.752475	1739.0	326	18731	inf
F	35542866	3724.886397	2343.5	342	18791	inf
G	45158240	3999.135671	2242.0	354	18818	inf
H	37257301	4486.669196	3460.0	337	18803	inf
I	27608146	5091.874954	3730.0	334	18823	inf
J	14949281	5323.818020	4234.0	335	18710	inf

We can see that in the prod(product i.e. multiplication) column all values are inf, inf is the result of a numerical calculation that is mathematically infinite.