PANDAS DATAFRAME

Pandas DataFrame(part 2)

- By Polu Sreekanth Reddy
- From SreeData

Pandas DataFrame

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

Creating a DataFrame

1. Creating a Empty Pandas DataFrame

In [1]:
```
import pandas as pd
df = pd.DataFrame()
print (df)
```

```
Empty DataFrame
Columns: []
Index: []
```

2. Creating a Pandas DataFrame

In [2]:
```
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
            'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

```
        0
0   Geeks
1     For
2   Geeks
3      is
4  portal
5     for
6   Geeks
```

3. Creating DataFrame from dict narray / lists

In [3]:
```
import pandas as pd

# intialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
print(df)
```

```
    Name  Age
0    Tom   20
1   nick   21
2  krish   19
3   jack   18
```

4. Create a DataFrame using List

In [4]:
```
import pandas as pd
# a list of strings
x = ['Python', 'Pandas','Numpy','matplot','seaborn']

# Calling DataFrame constructor on list
df = pd.DataFrame(x)
print(df)
```

```
         0
0   Python
1   Pandas
2    Numpy
3  matplot
4  seaborn
```

5. Create a DataFrame from Dict of Series

In [5]:
```
import pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),
   'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])}

d1 = pd.DataFrame(info)
print (d1)
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  4.0    4
e  5.0    5
f  6.0    6
g  NaN    7
h  NaN    8
```

Dealing with Rows

1. Row Selection

We can easily select, add, or delete any row at anytime. First of all, we will understand the row selection. Let's see how we can select a row using different ways that are as follows:

1. Selection by Label
2. selection by integer location

1.1. Selection by Label

In [6]:
```
import pandas as pd

data = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

df = pd.DataFrame(data)
print (df.loc['b'])
```

```
one    2.0
two    2.0
Name: b, dtype: float64
```

1.2. selection by integer location

The rows can also be selected by passing the integer location to an iloc function.

In [7]:
```
import pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

df = pd.DataFrame(info)
print (df.iloc[3])
```

```
one    4.0
two    4.0
Name: d, dtype: float64
```

2. Slice Rows

It is another method to select multiple rows using ':' operator

In [8]:
```
# importing the pandas library
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}
df = pd.DataFrame(info)
display(df[2:5])
```

	one	two
c	3.0	3
d	4.0	4
e	5.0	5

3. Addition of rows

We can easily add new rows to the DataFrame using append function. It adds the new rows at the end.

In [9]:
```
import pandas as pd
a = pd.DataFrame([10,20,30,45,60,70])
b = pd.DataFrame([10000,200000])
c = pd.concat([a,b], ignore_index = True)
display(c)
```

	0
0	10
1	20
2	30
3	45
4	60
5	70
6	10000
7	200000

In [10]:
```
dict = {'Name':['Martha', 'Tim', 'Rob', 'Georgia'],
        'Maths':[87, 91, 97, 95],
        'Science':[83, 99, 84, 76]
       }

df1 = pd.DataFrame(dict)
display(df1)

dict = {'Name':['Amy', 'Maddy'],
        'Maths':[89, 90],
        'Science':[93, 81]
       }

df2 = pd.DataFrame(dict)
display(df2)

df3 = pd.concat([df1, df2], ignore_index = True)
df3.reset_index()

display(df3)
```

	Name	Maths	Science
0	Martha	87	83
1	Tim	91	99
2	Rob	97	84
3	Georgia	95	76

	Name	Maths	Science
0	Amy	89	93
1	Maddy	90	81

	Name	Maths	Science
0	Martha	87	83
1	Tim	91	99
2	Rob	97	84
3	Georgia	95	76
4	Amy	89	93
5	Maddy	90	81

4.Deletion of rows

We can delete or drop any rows from a DataFrame using the index label. If in case, the label is duplicate then multiple rows will be deleted.

In [11]:
```
import pandas as pd

# Create DataFrame
dataFrame = pd.DataFrame([[10, 15], [20, 25], [30, 35], [40, 45]],index=['w', 'x', 'y', 'z'],columns=['a', 'b'])

# DataFrame
display(dataFrame)

# deleting a row
dataFrame = dataFrame.drop('w')
display(dataFrame)
```

	a	b
w	10	15
x	20	25
y	30	35
z	40	45

	a	b
x	20	25
y	30	35
z	40	45

Dealing with Columns

1. Column Selection

We can select any column from the DataFrame. Here is the code that demonstrates how to select a column from the DataFrame.

In [12]:
```
# importing the pandas library
import pandas as pd
df = pd.DataFrame([[10, 15], [20, 25], [30, 35], [40, 45]],index=['w', 'x', 'y', 'z'],columns=['a', 'b'])
d1 = pd.DataFrame(df)
print (d1 ['a'])
```

```
w    10
x    20
y    30
z    40
Name: a, dtype: int64
```

2. Column Addition

We can also add any new column to an existing DataFrame. The below code demonstrates how to add any new column to an existing DataFrame:

In [13]:
```
dict = {'Name':['Martha', 'Tim', 'Rob', 'Georgia'],
        'Maths':[87, 91, 97, 95],
        'Science':[83, 99, 84, 76]
       }

df1 = pd.DataFrame(dict)
display(df1)

df1['telugu'] = [85,74,100,97]
df1
```

	Name	Maths	Science
0	Martha	87	83
1	Tim	91	99
2	Rob	97	84
3	Georgia	95	76

Out[13]:

	Name	Maths	Science	telugu
0	Martha	87	83	85
1	Tim	91	99	74
2	Rob	97	84	100
3	Georgia	95	76	97

3.Column Deletion:

We can also delete any column from the existing DataFrame. This code helps to demonstrate how the column can be deleted from an existing DataFrame:

In [14]:
```
import pandas as pd

# Create DataFrame
dataFrame = pd.DataFrame([[10, 15], [20, 25], [30, 35], [40, 45]],index=['w', 'x', 'y', 'z'],columns=['a', 'b'])

# DataFrame
display(dataFrame)

# deleting a row
dataFrame = dataFrame.drop('a',axis=1)
display(dataFrame)
```

	a	b
w	10	15
x	20	25
y	30	35
z	40	45

	b
w	15
x	25
y	35
z	45

DataFrame Functions

1.Pandas DataFrame.apply()

In [15]:
```
import pandas as pd

df = pd.DataFrame({'A': [1, 2], 'B': [10, 20]})

def square(x):
    return x * x

df1 = df.apply(square)
display(df)
display(df1)
```

	A	B
0	1	10
1	2	20

	A	B
0	1	100
1	4	400

2. Pandas DataFrame.aggregate()

The main task of DataFrame.aggregate() function is to apply some aggregation to one or more column. Most frequently used aggregations are: sum,min,max

In [16]:
```
import pandas as pd

data = {
  "x": [50, 40, 30],
  "y": [300, 1112, 42]
}

df = pd.DataFrame(data)
x = df.agg(["sum",'min','max'])
print(x)
```

```
       x     y
sum  120  1454
min   30    42
max   50  1112
```

3.Pandas DataFrame.assign()

The assign() method adds a new column to an existing DataFrame.

In [17]:
```
import pandas as pd

data = {
  "age": [16, 14, 10],
  "qualified": [True, True, True]
}
df = pd.DataFrame(data)
display(df)
new_df = df.assign(name = ["Emil", "Tobias", "Linus"])
display(new_df)
```

	age	qualified
0	16	True
1	14	True
2	10	True

	age	qualified	name
0	16	True	Emil
1	14	True	Tobias
2	10	True	Linus

4.Pandas DataFrame.astype()

The astype() method returns a new DataFrame where the data types has been changed to the specified type

In [18]:
```
import pandas as pd

data = {
  "Duration": [50, 40, 45],
  "Pulse": [109, 117, 110],
  "Calories": [409.1, 479.5, 340.8]
}

df = pd.DataFrame(data)
df.info()
print("====================================")
newdf = df.astype('float64')
newdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------------------  ------
 0   Duration  3 non-null      int64
 1   Pulse     3 non-null      int64
 2   Calories  3 non-null      float64
dtypes: float64(1), int64(2)
memory usage: 204.0 bytes
====================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------------------  ------
 0   Duration  3 non-null      float64
 1   Pulse     3 non-null      float64
 2   Calories  3 non-null      float64
dtypes: float64(3)
memory usage: 204.0 bytes
```

1. Pandas DataFrame.count()

The Pandas count() is defined as a method that is used to count the number of non-NA cells for each column or row. It is also suitable to work with the non-floating data.

In [19]:
```
import pandas as pd

data = {
  "Duration": [50, 40, None, None, 90, 20],
  "Pulse": [109, 140, 110, 125, 138, 170]
}

df = pd.DataFrame(data)

df.count()
```

Out[19]:
```
Duration    4
Pulse       6
dtype: int64
```

6.Pandas DataFrame.truncate()

The truncate() method removes elements before and after the specified indexes or labels.

Use the axis='columns' parameter to remove specified columns.

In [20]:
```
import pandas as pd

data = {
  "age": [50, 40, 30, 40, 20, 10, 30],
  "qualified": [True, False, False, False, False, True, True]
}
df = pd.DataFrame(data)

newdf = df.truncate(before=3, after=5)

newdf
```

Out[20]:

	age	qualified
3	40	False
4	20	False
5	10	True

7.Pandas DataFrame.describe()

The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame. It analyzes both numeric and object series and also the DataFrame column sets of mixed data types.

In [21]:
```
import pandas as pd

data = [[10, 18, 11], [13, 15, 8], [9, 20, 3]]

df = pd.DataFrame(data,columns=['c1','c2','c3'])

df.describe()
```

Out[21]:

	c1	c2	c3
count	3.000000	3.000000	3.000000
mean	10.666667	17.666667	7.333333
std	2.081666	2.516611	4.041452
min	9.000000	15.000000	3.000000
25%	9.500000	16.500000	5.500000
50%	10.000000	18.000000	8.000000
75%	11.500000	19.000000	9.500000
max	13.000000	20.000000	11.000000

8.Pandas DataFrame.drop_duplicates()

The drop_duplicates() function performs common data cleaning task that deals with duplicate values in the DataFrame. This method helps in removing duplicate values from the DataFrame.

In [22]:
```
import pandas as pd
emp = {"Name": ["Parker", "Smith", "William", "Parker"], "Age": [21, 32, 29, 21]}
df = pd.DataFrame(emp)
display(df)
display(df.drop_duplicates())
```

	Name	Age
0	Parker	21
1	Smith	32
2	William	29
3	Parker	21

	Name	Age
0	Parker	21
1	Smith	32
2	William	29

9. Pandas DataFrame groupby()

The groupby() method allows you to group your data and execute functions on these groups.

In [23]:
```
import pandas as pd
technologies   = ({
    'Courses':["Spark","PySpark","Hadoop","Python","Pandas","Hadoop","Spark","Python","NA"],
    'Fee' :[22000,25000,23000,24000,26000,25000,25000,22000,1500],
    'Duration':['30days','50days','55days','40days','60days','35days','30days','50days','40days'],
    'Discount':[1000,2300,1000,1200,2500,None,1400,1600,0]
          })
df = pd.DataFrame(technologies)
display("Create DataFrame:\n", df)
```

```
'Create DataFrame:\n'
```

	Courses	Fee	Duration	Discount
0	Spark	22000	30days	1000.0
1	PySpark	25000	50days	2300.0
2	Hadoop	23000	55days	1000.0
3	Python	24000	40days	1200.0
4	Pandas	26000	60days	2500.0
5	Hadoop	25000	35days	NaN
6	Spark	25000	30days	1400.0
7	Python	22000	50days	1600.0
8	NA	1500	40days	0.0

In [24]:
```
df2 =df.groupby(['Courses', 'Duration']).sum()
display("Get sum of groupby multiple columns:\n", df2)
```

```
'Get sum of groupby multiple columns:\n'
```

Courses	Duration	Fee	Discount
Hadoop	**35days**	25000	0.0
	55days	23000	1000.0
NA	**40days**	1500	0.0
Pandas	**60days**	26000	2500.0
PySpark	**50days**	25000	2300.0
Python	**40days**	24000	1200.0
	50days	22000	1600.0
Spark	**30days**	47000	2400.0

2. Pandas DataFrame.head()

The head() returns the first n rows for the object based on position. If your object has the right type of data in it, it is useful for quick testing. This method is used for returning top n (by default value 5) rows of a data frame or series.

In [25]:
```
import pandas as pd
technologies   = ({
    'Courses':["Spark","PySpark","Hadoop","Python","Pandas","Hadoop","Spark","Python","NA"],
    'Fee' :[22000,25000,23000,24000,26000,25000,25000,22000,1500],
    'Duration':['30days','50days','55days','40days','60days','35days','30days','50days','40days'],
    'Discount':[1000,2300,1000,1200,2500,None,1400,1600,0]
          })
df = pd.DataFrame(technologies)
display("Create DataFrame:\n", df)
print()
print("first 5 rows of dataframe")
display(df.head())
```

'Create DataFrame:\n'

	Courses	Fee	Duration	Discount
0	Spark	22000	30days	1000.0
1	PySpark	25000	50days	2300.0
2	Hadoop	23000	55days	1000.0
3	Python	24000	40days	1200.0
4	Pandas	26000	60days	2500.0
5	Hadoop	25000	35days	NaN
6	Spark	25000	30days	1400.0
7	Python	22000	50days	1600.0
8	NA	1500	40days	0.0

first 5 rows of dataframe

	Courses	Fee	Duration	Discount
0	Spark	22000	30days	1000.0
1	PySpark	25000	50days	2300.0
2	Hadoop	23000	55days	1000.0
3	Python	24000	40days	1200.0
4	Pandas	26000	60days	2500.0

3. Pandas DataFrame.hist()

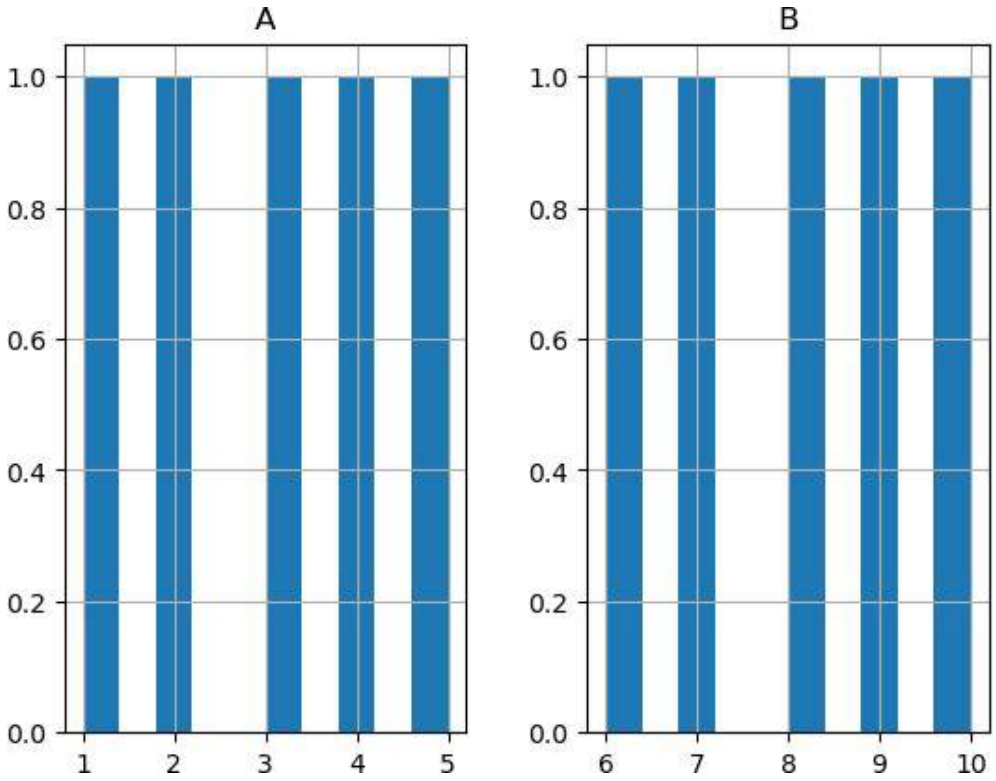
The hist() method in Pandas is used for plotting histograms to visually summarize the distribution of a dataset. A histogram represents the frequency distribution of numerical data by dividing the data range into bins and showing how many values fall into each bin.

In [26]:
```
import pandas as pd
import matplotlib.pyplot as plt

# sample DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [6, 7, 8, 9, 10]}

df = pd.DataFrame(data)

# plot histogram for all columns
hist_plot = df.hist()
plt.show()
```



4. Pandas DataFrame.join()

The join() method inserts column(s) from another DataFrame, or Series.

In [27]:
```
import pandas as pd

data1 = {
  "name": ["Sally", "Mary", "John"],
  "age": [50, 40, 30]
}

data2 = {
  "qualified": [True, False, False]
}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

newdf = df1.join(df2)
newdf
```

Out[27]:

	name	age	qualified
0	Sally	50	True
1	Mary	40	False
2	John	30	False

5. Pandas DataFrame.mean()

The mean() method returns a Series with the mean value of each column.

In [28]:
```
import pandas as pd

data = [[1, 1, 2], [6, 4, 2], [4, 2, 1], [4, 2, 3]]

df = pd.DataFrame(data)

df.mean()
```

Out[28]:
```
0    3.75
1    2.25
2    2.00
dtype: float64
```

6. Pandas DataFrame.mean()

The mode() method returns a Series with the mean value of each column.

In [29]:
```
import pandas as pd

data = [[1, 1, 2], [6, 4, 2], [4, 2, 1], [4, 2, 3]]

df = pd.DataFrame(data)

df.mode()
```

Out[29]:

	0	1	2
0	4	2	2

7. Pandas DataFrame.mean()

The mode() method returns a Series with the mean value of each column.

In [30]:
```
import pandas as pd

data = [[1, 1, 2], [6, 4, 2], [4, 2, 1], [4, 2, 3]]

df = pd.DataFrame(data)

df.median()
```

Out[30]:
```
0    4.0
1    2.0
2    2.0
dtype: float64
```

8. Pandas DataFrame merge()

The merge() method updates the content of two DataFrame by merging them together, using the specified method(s).

In [31]:
```
import pandas as pd

data1 = {
  "name": ["Sally", "Mary", "John"],
  "age": [50, 40, 30]
}

data2 = {
  "name": ["Sally", "Peter", "Micky"],
  "age": [77, 44, 22]
}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

newdf1 = df1.merge(df2, how='right')
newdf2 = df1.merge(df2,how='left')
```

In [32]:
```
newdf1
```

Out[32]:

	name	age
0	Sally	77
1	Peter	44
2	Micky	22

In [33]:
```
newdf2
```

Out[33]:

	name	age
0	Sally	50
1	Mary	40
2	John	30

9. Pandas DataFrame.query()

The query() method allows you to query the DataFrame.

The query() method takes a query expression as a string parameter, which has to evaluate to either True of False.

It returns the DataFrame where the result is True according to the query expression.

In [34]:
```
import pandas as pd

data = {
  "name": ["Sally", "Mary", "John"],
  "age": [50, 40, 30]
}

df = pd.DataFrame(data)

df.query('age > 35')
```

Out[34]:

	name	age
0	Sally	50
1	Mary	40

10. Pandas DataFrame.rename()

The rename() method allows you to change the row indexes, and the columns labels.

In [35]:
```
import pandas as pd

data = {
"age": [50, 40, 30],
"qualified": [True, False, False]
}
idx = ["Sally", "Mary", "John"]
df = pd.DataFrame(data, index=idx)

newdf = df.rename({"Sally": "Pete", "Mary": "Patrick", "John": "Paula"})

newdf
```

Out[35]:

	age	qualified
Pete	50	True
Patrick	40	False
Paula	30	False

11. Pandas Dataframe.sample()

The sample() method returns a specified number of random rows.

The sample() method returns 1 row if a number is not specified.

In [36]:
```
import pandas as pd

data = {
  "name": ["Sally", "Mary", "John"],
  "age": [50, 40, 30]
}

df = pd.DataFrame(data)
df.sample()
```

Out[36]:

	name	age
2	John	30

In [37]:
```
import pandas as pd

data = {
  "name": ["Sally", "Mary", "John"],
  "age": [50, 40, 30]
}

df = pd.DataFrame(data)

df.sample(2)
```

Out[37]:

	name	age
2	John	30
0	Sally	50

12. Pandas DataFrame.transpose()

The transpose() method transforms the columns into rows and the rows into columns.

In [38]:
```
import pandas as pd

data = {
  "age": [50, 40, 30, 40, 20, 10, 30],
  "qualified": [True, False, False, False, False, True, True]
}
df = pd.DataFrame(data)
display(df)

newdf = df.transpose()
display(newdf)
```

	age	qualified
0	50	True
1	40	False
2	30	False
3	40	False
4	20	False
5	10	True
6	30	True

	0	1	2	3	4	5	6
age	50	40	30	40	20	10	30
qualified	True	False	False	False	False	True	True

13. Pandas DataFrame.sort_index()

The sort_index() method sorts the DataFrame by the index

In [39]:
```
import pandas as pd

data = {
  "age": [50, 40, 30, 40, 20, 10, 30],
  "qualified": [True, False, False, False, False, True, True]
}

idx = ["Mary", "Sally", "Emil", "Tobias", "Linus", "John", "Peter"]
df = pd.DataFrame(data, index = idx)
display(df)

newdf = df.sort_index()
newdf
```

	age	qualified
Mary	50	True
Sally	40	False
Emil	30	False
Tobias	40	False
Linus	20	False
John	10	True
Peter	30	True

Out[39]:

	age	qualified
Emil	30	False
John	10	True
Linus	20	False
Mary	50	True
Peter	30	True
Sally	40	False
Tobias	40	False

Pandas Sorting Methods:

Pandas sort methods are the most primary way for learn and practice the basics of Data analysis by using Python. Data analysis is commonly done with Pandas, SQL, and spreadsheets. Pandas can handle a large amount of data and can offer the capabilities of highly performant data manipulations.

Pandas Sorting Methods are of two types:

1. DataFrame sort_Index
2. DataFrame sort_values

And Sorting can be Sorting a Column in the Ascending Order or Decending order

14. Pandas DataFrame.sort_values()

The sort_values() method sorts the DataFrame by the specified label.

In [40]:
```
import pandas as pd

data = {
  "age": [50, 40, 30, 40, 20, 10, 30],
  "qualified": [True, False, False, False, False, True, True] }
df = pd.DataFrame(data)
display(df)

newdf = df.sort_values(by='age')
newdf
```

	age	qualified
0	50	True
1	40	False
2	30	False
3	40	False
4	20	False
5	10	True
6	30	True

Out[40]:

	age	qualified
5	10	True
4	20	False
2	30	False
6	30	True
1	40	False
3	40	False
0	50	True

15. Pandas DataFrame.sum()

The sum() method adds all values in each column and returns the sum for each column.

By specifying the column axis (axis='columns'), the sum() method searches column-wise and returns the sum of each row.

In [41]:
```
import pandas as pd

data = [[10, 18, 11], [13, 15, 8], [9, 20, 3]]

df = pd.DataFrame(data)
display(df)
print("sum row wise")
print(df.sum(axis=1))
print()
print("sum column wise")
print(df.sum(axis=0))
```

	0	1	2
0	10	18	11
1	13	15	8
2	9	20	3

```
sum row wise
0    39
1    36
2    32
dtype: int64

sum column wise
0    32
1    53
2    22
dtype: int64
```

16. Pandas DataFrame.transform

The transform() method allows you to execute a function for each value of the DataFrame.

In [42]:
```
import pandas as pd

def multiply(x):
  return x * 10

data = {
  "for1": [2, 6, 3],
  "for5": [8, 20, 12]
}

df = pd.DataFrame(data)
display(df)

newdf = df.transform(multiply)
newdf
```

	for1	for5
0	2	8
1	6	20
2	3	12

Out[42]:

	for1	for5
0	20	80
1	60	200
2	30	120

Important Function that everyone should after after this :

- Between
- Pandas Styler
- unique and nunique
- Pipe
- Factorize
- Explode
- ExcelWriter
- Pandas option
- Mask
- Clip
- Cut and qcut

In [43]:
```
import pandas as pd

data = {
  "age": [50, 40, 30, 40, 20, 10, 30],
  "qualified": [True, False, False, False, False, True, True]
}
df = pd.DataFrame(data)
display(df)

newdf = df.where(df["age"] > 30)
display(newdf)
```

	age	qualified
0	50	True
1	40	False
2	30	False
3	40	False
4	20	False
5	10	True
6	30	True

	age	qualified
0	50.0	True
1	40.0	False
2	NaN	NaN
3	40.0	False
4	NaN	NaN
5	NaN	NaN
6	NaN	Na

25. Pandas DataFrame.where()

The where() method replaces the values of the rows where the condition evaluates to False.

The where() method is the opposite of the The mask() method.

26. Pandas DataFrame dropna() Method

The dropna() method removes the rows that contains NULL values.

The dropna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the dropna() method does the removing in the original DataFrame instead.

In [44]:
```
import pandas as pd

data = {
  "age": [50, 40, 30, 40, 20, 10, 30],
  "qualified": [True, False, False, False, False, True, True]
}

newdf = df.where(df["age"] > 30)
display(newdf)

display(newdf.dropna())
```

	age	qualified
0	50.0	True
a.	40.0	False
b.	NaN	NaN
c.	40.0	False
d.	NaN	NaN
e.	NaN	NaN
f.	NaN	NaN

	age	qualified
0	50.0	True
1	40.0	False
3	40.0	False

27.Pandas DataFrame ne()

The ne() method compares each value in a DataFrame to check if it is NOT equal to a specified value, or a value from a specified DataFrame objects, and returns a DataFrame with boolean True/False for each comparison.

In [45]:
```
import pandas as pd

df = pd.DataFrame([[10, 12, 2], [3, 4, 7]])
display(df)
print(df.ne(7))
```

	0	1	2
0	10	12	2
1	3	4	7

```
      0     1      2
0  True  True   True
1  True  True  False
```

28.Pandas DataFrame abs()

The abs() method returns a DataFrame with the absolute value of each value.

In [46]:
```
import pandas as pd

data = [[-50, 40, 30], [-1, 2, -2]]
df = pd.DataFrame(data)
display(df)

display(df.abs())
```

	0	1	2
0	-50	40	30
1	-1	2	-2

	0	1	2
0	50	40	30
1	1	2	2

29.Pandas DataFrame var()

The var() method calculates the standard deviation for each column.

By specifying the column axis (axis='columns'), the var() method searches column-wise and returns the standard deviation for each row.

In [47]:
```
import pandas as pd

data = [[10, 18, 11], [13, 15, 8], [9, 20, 3]]

df = pd.DataFrame(data)
display(df)

print("Row wise")
print(df.var(axis=1))
print()
print("Column wise")
print(df.var(axis=0))
```

	0	1	2
0	10	18	11
1	13	15	8
2	9	20	3

```
Row wise
0    19.000000
1    13.000000
2    74.333333
dtype: float64

Column wise
0     4.333333
1     6.333333
2    16.333333
dtype: float64
```

30.Pandas DataFrame prod()

The prod() method multiplies all values in each column and returns the product for each column.

By specifying the column axis (axis='columns'), the prod() method searches column-wise and returns the product of each row.

In [48]:
```
import pandas as pd

data = [[10, 18, 11], [13, 15, 8], [9, 20, 3]]

df = pd.DataFrame(data)
display(df)

print("Row wise")
print(df.prod(axis=1))
print()
print("Column wise")
print(df.prod(axis=0))
```

	0	1	2
0	10	18	11
1	13	15	8
2	9	20	3

```
Row wise
0    1980
1    1560
2     540
dtype: int64

Column wise
0    1170
1    5400
2     264
dtype: int64
```

31/Pandas DataFrame squeeze()

The squeeze() method converts a single column DataFrame into a Series.

In [49]:
```
import pandas as pd
data = {
  "age": [50, 40, 30, 40, 20, 10, 30]
}
df = pd.DataFrame(data)
s = df.squeeze()
s
```

Out[49]:
```
0    50
1    40
2    30
3    40
4    20
5    10
6    30
Name: age, dtype: int64
```