In [1]:
```
import numpy as np
import pandas as pd
```

Create A Series

- Series Is One-Dimensional Array Of Data
- It Can Hold Data Of Any Type: String, Integer, Float, Dict, List, Boolean, ...
- Pandas Series Like A Column In A Table

In [3]:
```
data = [1, 2, 3, 4]
series = pd.Series(data)
series
```

Out[3]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

In [4]:
```
data = ["Osama", "Mohamed", "Khaled", "Mahmoud"]
series = pd.Series(data)
series
```

Out[4]:
```
0      Osama
1    Mohamed
2     Khaled
3    Mahmoud
dtype: object
```

You Can Specify The Index Of the Rows

In [6]:
```
series = pd.Series(data, index=["N1", "N2", "N3", "N4"])
series
```

Out[6]:
```
N1      Osama
N2    Mohamed
N3     Khaled
N4    Mahmoud
dtype: object
```

DataFrame

- Pandas DataFrame Is A 2-Dimensional Data Structure
- Like 2 Dimensional Or A Table With Rows And Columns

In [8]:
```
data = {
    "Calories": [420, 380, 390],
    "Duration": [50, 40, 45],
}

# Load The Data Into A DataFrame Object
df = pd.DataFrame(data)
df
```

Out[8]:

	Calories	Duration
0	420	50
1	380	40
2	390	45

In [9]:
```
data = [
    [1, 2, 3],
    [4, 5 ,6],
    [7, 8, 9],
]

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

Out[9]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Rename DataFrame Columns

In [11]:
```
cols = ["Column 1", 'Column 2', "Column 3"]
df.columns = cols
display(df)
```

	Column 1	Column 2	Column 3
0	1	2	3
1	4	5	6
2	7	8	9

Rename DataFrame Index

In [13]:
```
index = ["Row 1", 'Row 2', 'Row 3']
df.index = index
df
```

Out[13]:

	Column 1	Column 2	Column 3
Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

Pandas Dtypes

- Bool: Represents Numerical Datatypes With A True & False Values
- Int: Represents Numerical Datatypes With Ineger Values
- Float: Represents Numerical Datatypes With Continuous Values
- Category: Represents categorical Datatypes
- Object: Is A Mix Of Categorical & Numerical Datatypes, Can Carry Any Python Object

Get Datatypes Of All Columns

In [16]:
```
df.dtypes
```

Out[16]:
```
Column 1    int64
Column 2    int64
Column 3    int64
dtype: object
```

Get Datatype Of One Column

In [18]:
```
df['Column 1'].dtype
```

Out[18]:
```
dtype('int64')
```

Change Datatype Of Group Of Columns

In [20]:
```
columns = ["Column 1", "Column 2", "Column 3"]
df[cols] = df[cols].astype("category")
df.dtypes
```

Out[20]:
```
Column 1    category
Column 2    category
Column 3    category
dtype: object
```

Create DataFrame For Testing

In [22]:
```
data = np.random.randint(0, 20, (10, 3))
df = pd.DataFrame(data)
df
```

Out[22]:

	0	1	2
0	10	13	15
1	4	18	2
2	13	17	0
3	5	7	8
4	4	15	8
5	9	5	9
6	15	14	6
7	15	8	17
8	4	7	19
9	8	7	8

Get The First Rows Of DataFrame

- Get The First 5 Rows On The DataFrame

In [24]:
```
df.head()
```

Out[24]:

	0	1	2
0	10	13	15
1	4	18	2
2	13	17	0
3	5	7	8
4	4	15	8

Get The Last Rows Of DataFrame

- Get The Last 5 Rows On The DataFrame

In [26]:
```
df.tail()
```

Out[26]:

	0	1	2
5	9	5	9
6	15	14	6
7	15	8	17
8	4	7	19
9	8	7	8

Get Specific N Rows On The Head

In [28]:
```
df.head(3)
```

Out[28]:

	0	1	2
0	10	13	15
1	4	18	2
2	13	17	0

Get Specific N Rows On The Tail

In [30]:
```
df.tail(3)
```

Out[30]:

	0	1	2
7	15	8	17
8	4	7	19
9	8	7	8

Get Statistical Measures About Numerical Columns

- Method Retuens Description Of The Data In The DataFrame
- If The DataFrame COntains Numerical Data, The Description Contains These Information For Each Column:

- Count: The Number Of Non-Empty Values
- Mean: The Average value
- Std: The Standard Deviation
- Min: The Minimum Value
- 25%: The 25% Percentile*
- 50%: The 50% Percentile*
- 75%: The 75% Percentile*
- Max: The Maximum Value

In [32]:
```
df.describe()
```

Out[32]:

	0	1	2
count	10.000000	10.000000	10.000000
mean	8.700000	11.100000	9.200000
std	4.473378	4.794673	6.160808
min	4.000000	5.000000	0.000000
25%	4.250000	7.000000	6.500000
50%	8.500000	10.500000	8.000000
75%	12.250000	14.750000	13.500000
max	15.000000	18.000000	19.000000

Get DataFrame Rows Names

In [34]:
```
df.index
```

Out[34]:
```
RangeIndex(start=0, stop=10, step=1)
```

Get DataFrame Columns Names

In [36]:
```
df.columns
```

Out[36]:
```
RangeIndex(start=0, stop=3, step=1)
```

Get Unique Values

In [38]:
```
df[0].unique()
```

Out[38]:
```
array([10,  4, 13,  5,  9, 15,  8])
```

Get Unique Values Numbers

In [40]:
```
df[0].nunique()
```

Out[40]:
```
7
```

Get Max Value

In [42]:
```
df[0].max()
```

Out[42]:
```
15
```

Get Min Values

In [44]:
```
df[0].min()
```

Out[44]:
```
4
```

Get Element Whose Value Is Max

- Returns A Series With The Index Of The Maximum Value For Each Column
- Returns A Series With The Index Of The Maximum Value For Each Row

In [46]:
```
df.idxmax()
```

Out[46]:
```
0    6
1    1
2    8
dtype: int64
```

Get Element Whose Value Is Min

- Rerurns A Series With The Index Of Minimum Value For each Column
- By Sepcifying The Column Axis (axis="columns"), The idxmin() Returns A Series With The Index Of The Minimum Value For Each Row
- Syntax: DataFrame.idxmin(axis)

In [48]:
```
df.idxmin()
```

Out[48]:
```
0    1
1    5
2    2
dtype: int64
```

Get The DataFrame Basic Information

- Returns A Summary Of The DataFrame
- The Summary Contains The Number Of Columns, Column Lables, Column Data Types, Memory Usage, Range Index, And The Number Of Cells In Each Columns (non-null values)

In [50]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       10 non-null     int32
 1   1       10 non-null     int32
 2   2       10 non-null     int32
dtypes: int32(3)
memory usage: 252.0 bytes
```

Get Max Value Of All Columns

In [52]:
```
df.max()
```

Out[52]:
```
0    15
1    18
2    19
dtype: int32
```

Get The Unique Value Frequency

In [54]:
```
d = df[0].value_counts()
d
```

Out[54]:
```
0
4     3
15    2
10    1
13    1
5     1
9     1
8     1
Name: count, dtype: int64
```

Summation

In [56]:
```
df[0].sum()
```

Out[56]:
```
87
```

In [57]:
```
df[1].sum()
```

Out[57]:
```
111
```

In [58]:
```
df[2].sum()
```

Out[58]:
```
92
```

Convert Pandas Series Into Numpy 1-D Array

In [60]:
```
df[0].values
```

Out[60]:
```
array([10,  4, 13,  5,  4,  9, 15, 15,  4,  8])
```

In [61]:
```
df[1].values
```

Out[61]:
```
array([13, 18, 17,  7, 15,  5, 14,  8,  7,  7])
```

Convert Pandas DataFrame Into Numpy 2-D Array

In [63]:
```
df.values
```

Out[63]:
```
array([[10, 13, 15],
       [ 4, 18,  2],
       [13, 17,  0],
       [ 5,  7,  8],
       [ 4, 15,  8],
       [ 9,  5,  9],
       [15, 14,  6],
       [15,  8, 17],
       [ 4,  7, 19],
       [ 8,  7,  8]])
```

Replace A Single Value

In [65]:
```
data = np.arange(20).reshape(5, 4)
df = pd.DataFrame(data)
df
```

Out[65]:

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

In [66]:
```
df.replace(0, "Zero")
```

Out[66]:

	0	1	2	3
0	Zero	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

In [67]:
```
df.replace(10, "Ten")
```

Out[67]:

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	Ten	11
3	12	13	14	15
4	16	17	18	19

Replace Multiple Values Using Dictionary

In [69]:
```
df.replace({
    0 : "Zero",
    5 : "Five",
    10: "Ten",
})
```

Out[69]:

	0	1	2	3
0	Zero	1	2	3
1	4	Five	6	7
2	8	9	Ten	11
3	12	13	14	15
4	16	17	18	19

Replace Multiple Values By One Value

In [71]:
```
df.replace({0, 1, 2, 3}, "Value")
```

Out[71]:

	0	1	2	3
0	Value	Value	Value	Value
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

Data Manipulating (Mapping)

- Used To Substitute

Data Before Mapping

In [74]:
```
cols = ["col1", "col2", "col3", "col4"]
df.columns = cols
df
```

Out[74]:

	col1	col2	col3	col4
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

Data After mapping

In [76]:
```
df.col2 = df.col2.map({5: "Fives", 1: "Ones"})
df
```

Out[76]:

	col1	col2	col3	col4
0	0	Ones	2	3
1	4	Fives	6	7
2	8	NaN	10	11
3	12	NaN	14	15
4	16	NaN	18	19

Sorting

Ascending Sorting

In [79]:
```
# Sort col1 On The DataFrame
sorted_df = df.sort_values(by="col1")
sorted_df
```

Out[79]:

	col1	col2	col3	col4
0	0	Ones	2	3
1	4	Fives	6	7
2	8	NaN	10	11
3	12	NaN	14	15
4	16	NaN	18	19

In [80]:
```
sored_df = df.sort_values(by="col3")
sorted_df
```

Out[80]:

	col1	col2	col3	col4
0	0	Ones	2	3
1	4	Fives	6	7
2	8	NaN	10	11
3	12	NaN	14	15
4	16	NaN	18	19

Descending Sort

In [82]:
```
sorted_df = df.sort_values(by = "col1", ascending=False)
sorted_df
```

Out[82]:

	col1	col2	col3	col4
4	16	NaN	18	19
3	12	NaN	14	15
2	8	NaN	10	11
1	4	Fives	6	7
0	0	Ones	2	3

Apply Method

In [84]:
```
# First Print The DataFrame Before Applying The Method
df
```

Out[84]:

	col1	col2	col3	col4
0	0	Ones	2	3
1	4	Fives	6	7
2	8	NaN	10	11
3	12	NaN	14	15
4	16	NaN	18	19

In [85]:
```
# Then Make Method, And Apply This To The DataFrame Elements
def duplicate(x):
    return x*2
df["col1"].apply(duplicate)
```

Out[85]:
```
0     0
1     8
2    16
3    24
4    32
Name: col1, dtype: int64
```

You Can Use Lambda Function

In [87]:
```
df['col1'].apply(lambda x: x*3)
```

Out[87]:
```
0     0
1    12
2    24
3    36
4    48
Name: col1, dtype: int64
```

Indexing & Slicing

Indexing

- Means Accesing One Element In Series Or DataFrame, Using Its Index Or Name
- There Are Two Ways To Apply Indexing
  1. Using The Element's Index
  2. Using The Element's Name

Slicing

- Means Accessing Many Elements In Series Or DataFrame, By Specifying A Range Of Indicies Or Name
- There Are Two Ways To Apply Slicing:
  1. Using Range Of Element's Indicies
  2. Using Range Of Element's Names

In [89]:
```
data = np.arange(20).reshape(10, 2)
df = pd.DataFrame(data)
cols = ["Column1", "Column2"]
df.columns = cols
df
```

Out[89]:

	Column1	Column2
0	0	1
1	2	3
2	4	5
3	6	7
4	8	9
5	10	11
6	12	13
7	14	15
8	16	17
9	18	19

Series Indexing

In [91]:
```
# Print The Value Of The Given Index
df.Column1.iloc[6]
```

Out[91]:
```
12
```

In [92]:
```
# Print The Value Of The Given Index
df.Column2.iloc[5]
```

Out[92]:
```
11
```

Matrix Indexing

In [94]:
```
df.iloc[2, 1]
```

Out[94]:
```
5
```

Series Slicing

In [96]:
```
df.Column1.iloc[0:2]
```

Out[96]:
```
0    0
1    2
Name: Column1, dtype: int32
```

Matrix Slicing

In [98]:
```
# Get Rows Starting From Index 5 To End, And Get All Columns
df.iloc[5:, :]
```

Out[98]:

	Column1	Column2
5	10	11
6	12	13
7	14	15
8	16	17
9	18	19

In [99]:
```
# Get All Rows, And Get First Column
df.iloc[:, 0:1]
```

Out[99]:

	Column1
0	0
1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	18

Indexing & Slicing Using Names

Series Indexing

In [102…
```
df.Column1.loc[3]
```

Out[102… 6

Matrix Indexing

In [104…
```
df.loc[2, "Column1"]
```

Out[104… 4

Series Slicing

In [106…
```
df.Column1.loc[2:3]
```

Out[106…
```
2    4
3    6
Name: Column1, dtype: int32
```

Matric Slicing

In [108…
```
df.loc[:3, "Column1": "Column2"]
```

Out[108…

	Column1	Column2
0	0	1
1	2	3
2	4	5
3	6	7

Inserting & Dropping DataFrame Columns & Rows

Create New DataFrame

In [111…
```
data = np.arange(18).reshape(6, 3)
df = pd.DataFrame(data)
cols = ["col1", "col2", "col3"]
df.columns = cols
df
```

Out[111…

	col1	col2	col3
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11
4	12	13	14
5	15	16	17

Insert New Column

- Method Allows Us To Insert New Column To An Existing DataFrame
- Syntax: insert(location, column, value, allow_duplicate)

In [113…

```
new_col = df.col1 + df.col2
df.insert(3, "new", new_col)
df
```

Out[113…

	col1	col2	col3	new
0	0	1	2	1
1	3	4	5	7
2	6	7	8	13
3	9	10	11	19
4	12	13	14	25
5	15	16	17	31

Another Way To Insert New Column

In [115…

```
df["new_column"] = df.col1 + df.col2
df
```

Out[115…

	col1	col2	col3	new	new_column
0	0	1	2	1	1
1	3	4	5	7	7
2	6	7	8	13	13
3	9	10	11	19	19
4	12	13	14	25	25
5	15	16	17	31	31

Drop Column

Drop One Column

In [118…

```
df.drop('new', axis=1)
```

Out[118…

	col1	col2	col3	new_column
0	0	1	2	1
1	3	4	5	7
2	6	7	8	13
3	9	10	11	19
4	12	13	14	25
5	15	16	17	31

Drop Many Columns

In [120…

```
df.drop(["col1", "col2"], axis=1)
```

Out[120…

	col3	new	new_column
0	2	1	1
1	5	7	7
2	8	13	13
3	11	19	19
4	14	25	25
5	17	31	31

Insert New Rows

- Syntax: append(other, ignore_index, verify_integrity, sort)
- ignore_index: If True --> The Original Indexs Are Ignored And Will Be Replaced By 0, 1, 2, ..
- verify_integrity --> If True You Will Get An Error If You Have Two Or More Rows With The Same Index
- sort --> If True Sorts Columns

In [122…

```
new_row = {"col1": 1, "col2": 222, "col3": 333}
new_row2 = {"col1": 10, "col2": 20, "col3": 30, "new": 40, "new_column": 50}
x = df._append(new_row, ignore_index = True)
y = df._append(new_row2, ignore_index = True)
display(x)
```

	col1	col2	col3	new	new_column
0	0	1	2	1.0	1.0
1	3	4	5	7.0	7.0
2	6	7	8	13.0	13.0
3	9	10	11	19.0	19.0
4	12	13	14	25.0	25.0
5	15	16	17	31.0	31.0
6	1	222	333	NaN	NaN

In [123…

```
display(y)
```

	col1	col2	col3	new	new_column
0	0	1	2	1	1
1	3	4	5	7	7
2	6	7	8	13	13
3	9	10	11	19	19
4	12	13	14	25	25
5	15	16	17	31	31
6	10	20	30	40	50

Drop Rows

Drop One Row

In [126…

```
# Drop Row With Index 2
df.drop(2, axis=0)
```

Out[126…

	col1	col2	col3	new	new_column
0	0	1	2	1	1
1	3	4	5	7	7
3	9	10	11	19	19
4	12	13	14	25	25
5	15	16	17	31	31

Drop Many Rows

In [128…
```
df.drop([1, 3], axis=0)
```

Out[128…

	col1	col2	col3	new	new_column
0	0	1	2	1	1
2	6	7	8	13	13
4	12	13	14	25	25
5	15	16	17	31	31

Null Values

- Null Values Means Missing Values, Which Means That An Element Doesn't Have A Value, Or Have A Value Of None Or Nan
- Null Values Occur Due To Problems During Gathering Data For Example A Client Forget Or To Enter His Age

In [130…
```
# Check For Null Values
null = df.isnull()
pd.DataFrame(null.sum()).T
```

Out[130…

	col1	col2	col3	new	new_column
0	0	0	0	0	0

Create New DataFrame With A Null (NaN) Values

In [154…
```
# Create New DataFrame With Null Values
data = np.array([
    [np.nan, 2, 3, np.nan],
    [10, 20, 30, np.nan],
    [10, 20, 30, 40],
    [10, 20, 65, 60],
    [10, 80, 100, 60],
    [np.nan, 10, np.nan, np.nan],
])

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

Out[154…

	0	1	2	3
0	NaN	2.0	3.0	NaN
1	10.0	20.0	30.0	NaN
2	10.0	20.0	30.0	40.0
3	10.0	20.0	65.0	60.0
4	10.0	80.0	100.0	60.0
5	NaN	10.0	NaN	NaN

Check For Null (NaN) Values

In [157…
```
null = df.isnull()
pd.DataFrame(null.sum()).T
```

Out[157…

	0	1	2	3
0	2	0	1	3

In [159…
```
# Print The Null Values As Boolean Values
null
```

Out[159…

	0	1	2	3
0	True	False	False	True
1	False	False	False	True
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
5	True	False	True	True

Handle Null Values

- There Are Three Options To Handle Missing Values:

1. Drop Rows That Contains Null Values
2. Drop Columns That Contains Null Values
3. replace Null Values With Mean, Median, Or Mode

Drop All Rows

Remove ALl Rows With Null Values From The DataFrame

In [164…
```
df.dropna()
```

Out[164…

	0	1	2	3
2	10.0	20.0	30.0	40.0
3	10.0	20.0	65.0	60.0
4	10.0	80.0	100.0	60.0

Drop Rows In Specific Columns

In [167…
```
df.dropna(subset=[3])
```

Out[167…

	0	1	2	3
2	10.0	20.0	30.0	40.0
3	10.0	20.0	65.0	60.0
4	10.0	80.0	100.0	60.0

Drop Columns

In [170…
```
df
```

Out[170…

	0	1	2	3
0	NaN	2.0	3.0	NaN
1	10.0	20.0	30.0	NaN
2	10.0	20.0	30.0	40.0
3	10.0	20.0	65.0	60.0
4	10.0	80.0	100.0	60.0
5	NaN	10.0	NaN	NaN

Drop All Columns That Contains Null Values

In [172…
```
df.dropna(axis=1)
```

Out[172…

	1
0	2.0
1	20.0
2	20.0
3	20.0
4	80.0
5	10.0

Drop Specific Columns

In [180…
```
df.drop([0], axis=1)
```

Out[180…

	1	2	3
0	2.0	3.0	NaN
1	20.0	30.0	NaN
2	20.0	30.0	40.0
3	20.0	65.0	60.0
4	80.0	100.0	60.0
5	10.0	NaN	NaN

In [182…
```
df.drop([2], axis=1)
```

Out[182…

	0	1	3
0	NaN	2.0	NaN
1	10.0	20.0	NaN
2	10.0	20.0	40.0
3	10.0	20.0	60.0
4	10.0	80.0	60.0
5	NaN	10.0	NaN

Rename The Columns Name

In [191…
```
cols = ["col1", "col2", "col3", "col4"]
df.columns = cols
df
```

Out[191…

	col1	col2	col3	col4
0	NaN	2.0	3.0	NaN
1	10.0	20.0	30.0	NaN
2	10.0	20.0	30.0	40.0
3	10.0	20.0	65.0	60.0
4	10.0	80.0	100.0	60.0
5	NaN	10.0	NaN	NaN

Replace Rows Null Values

In [194…
```
mean = df.col1.mean()
df.col1 = df.col1.fillna(value=mean)
df
```

Out[194…

	col1	col2	col3	col4
0	10.0	2.0	3.0	NaN
1	10.0	20.0	30.0	NaN
2	10.0	20.0	30.0	40.0
3	10.0	20.0	65.0	60.0
4	10.0	80.0	100.0	60.0
5	10.0	10.0	NaN	NaN

In [ ]:
```
```