Pandas - Also known as Advanced version of excel

Series - Data type

```
import numpy as np
import pandas as pd
```

Create a Series

```
label = ['a','b','c']
my_list = [10,20,30]
ar = np.array([10,20,30])
d = {'a':20, 'b':40, 'c':60}
```

```
pd.Series(data = my_list, index = label)
```

```
a    10
b    20
c    30
dtype: int64
```

```
pd.Series(ar)
```

```
0    10
1    20
2    30
dtype: int64
```

```
pd.Series(d)
```

```
a    20
b    40
c    60
dtype: int64
```

Using Index in Series

```
sales_q1 = pd.Series(data = [100,200,300,400], index = ['USA','AUS','IND','
sales_q1
```

```
USA    100
AUS    200
IND    300
EUR    400
dtype: int64
```

```
sales_q1["IND"]
```

```
300
```

```
sales_q1[3]
```

```
400
```

```
sales_q2 = pd.Series(data = [260,360,470,580], index = ['USA','AUS','IND','
sales_q2
```

```
USA    260
AUS    360
IND    470
JAP    580
dtype: int64
```

Operations

We can perform arithmatic operations if index of both series have same datatype and values

```
sales_q1 + sales_q2
```

```
AUS    560.0
EUR      NaN
IND    770.0
JAP      NaN
USA    360.0
dtype: float64
```

DataFrame

DF are work horse of Pandas. Bunch of series object that shares same index.

```
index = ['a','b','c','d','e']
col = ['w','x','y','z']
```

```
np.random.seed(42)
data = np.random.randint(-100,100,(5,4))
data
```

```
array([[  2,  79,  -8, -86],
       [  6, -29,  88, -80],
       [  2,  21, -26, -13],
       [ 16,  -1,   3,  51],
       [ 30,  49, -48, -99]])
```

```
df = pd.DataFrame(data,index,col)
df
```

	w	x	y	z
a	2	79	-8	-86
b	6	-29	88	-80
c	2	21	-26	-13
d	16	-1	3	51
e	30	49	-48	-99

Selection and Indexing

```
df['w']
```

```
a     2
b     6
c     2
d    16
e    30
Name: w, dtype: int64
```

```
type(df['w'])
```

pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool=False, fastpath: bool=False) -> None

One-dimensional ndarray with axis labels (including time series

Labels need not be unique but must be a hashable type. The obje supports both integer- and label-based indexing and provides a l methods for performing operations involving the index. Statisti methods from ndarray have been overridden to automatically exclu

```
df[['w','z']]
```

	w	z
a	2	-86
b	6	-80
c	2	-13
d	16	51
e	30	-99

```
type(df[['w','z']])
```

pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None, dtype: Dtype | None=None, copy: bool | None=None) -> None

Two-dimensional, size-mutable, potentially heterogeneous tabula

Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can l thought of as a dict-like container for Series objects. The pri pandas data structure.

```
df['new'] = df['w'] + df['y']
df
```

	w	x	y	z	new
a	2	79	-8	-86	-6
b	6	-29	88	-80	94
c	2	21	-26	-13	-24
d	16	-1	3	51	19
e	30	49	-48	-99	-18

```
df.drop('new',axis = 1,inplace = True) #it will not drop permamnently. So u
```

```
df
```

	w	x	y	z
a	2	79	-8	-86
b	6	-29	88	-80
c	2	21	-26	-13
d	16	-1	3	51
e	30	49	-48	-99

Work with Rows

```
df.loc['a']
```

```
w     2
x    79
y    -8
z   -86
Name: a, dtype: int64
```

```
type(df.loc['a'])
```

pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool=False, fastpath: bool=False) -> None

One-dimensional ndarray with axis labels (including time series

Labels need not be unique but must be a hashable type. The obje
supports both integer- and label-based indexing and provides a
methods for performing operations involving the index. Statisti
methods from ndarray have been overridden to automatically exclu

```
df.loc[['a','b','e']]
```

	w	x	y	z
a	2	79	-8	-86
b	6	-29	88	-80
e	30	49	-48	-99

```
df.iloc[0]
```

```
w     2
x    79
y    -8
z   -86
Name: a, dtype: int64
```

```
df.iloc[0:3]
```

	w	x	y	z
a	2	79	-8	-86
b	6	-29	88	-80
c	2	21	-26	-13

```
df.drop('a',axis = 0) #or df.drop('a') because default axis will be 0
```

	w	x	y	z
b	6	-29	88	-80
c	2	21	-26	-13
d	16	-1	3	51
e	30	49	-48	-99

```
df.loc[['a','c'],['w','x']]
```

	w	x
a	2	79
c	2	21

Conditional Selection

```
df>0
```

	w	x	y	z
a	True	True	False	False
b	True	False	True	False
c	True	True	False	False
d	True	False	True	True
e	True	True	False	False

```
df[df>0]
```

	w	x	y	z
a	2	79.0	NaN	NaN
b	6	NaN	88.0	NaN
c	2	21.0	NaN	NaN
d	16	NaN	3.0	51.0
e	30	49.0	NaN	NaN

```
df['x']>0
```

```
a     True
b    False
c     True
d    False
e     True
Name: x, dtype: bool
```

```
df[df['x']>0]
```

	w	x	y	z
a	2	79	-8	-86
c	2	21	-26	-13
e	30	49	-48	-99

```
df[df['x']>0]['y']
```

```
a    -8
c   -26
e   -48
Name: y, dtype: int64
```

```
df[(df['w']>0) & (df['y']>1)]
```

	w	x	y	z
b	6	-29	88	-80
d	16	-1	3	51

```
df.reset_index() #to make this change permanent, assign this to a variable
```

	index	w	x	y	z
0	a	2	79	-8	-86
1	b	6	-29	88	-80
2	c	2	21	-26	-13
3	d	16	-1	3	51
4	e	30	49	-48	-99

```
newindexlist = ['q','r','s','t','u']

df["newindex"] = newindexlist

df
```

	w	x	y	z	newindex
a	2	79	-8	-86	q
b	6	-29	88	-80	r
c	2	21	-26	-13	s
d	16	-1	3	51	t
e	30	49	-48	-99	u

```
df.set_index('newindex') #to make this change permanent, assign this to a v
```

newindex	w	x	y	z
q	2	79	-8	-86
r	6	-29	88	-80
s	2	21	-26	-13
t	16	-1	3	51
u	30	49	-48	-99

```
df
```

	w	x	y	z	newindex
a	2	79	-8	-86	q
b	6	-29	88	-80	r
c	2	21	-26	-13	s
d	16	-1	3	51	t
e	30	49	-48	-99	u

What if we have two index values with same word?

```
df['country'] = ['India','India','Germany','Japan','USA']
```

```
df
```

	w	x	y	z	newindex	country
a	2	79	-8	-86	q	India
b	6	-29	88	-80	r	India
c	2	21	-26	-13	s	Germany
d	16	-1	3	51	t	Japan
e	30	49	-48	-99	u	USA

```
df = df.set_index('country')
```

```
df
```

	w	x	y	z	newindex
country					
India	2	79	-8	-86	q
India	6	-29	88	-80	r
Germany	2	21	-26	-13	s
Japan	16	-1	3	51	t
USA	30	49	-48	-99	u

Checking which one it will pick..

```
df.loc['India']
```

	w	x	y	z	newindex
country					
India	2	79	-8	-86	q
India	6	-29	88	-80	r

Hence, it will chose both of them.

```
df.describe() #it rejects string or char column as it performs mathematical
```

	w	x	y	z
count	5.00000	5.000000	5.000000	5.000000
mean	11.20000	23.800000	1.800000	-45.400000
std	11.96662	42.109381	51.915316	63.366395
min	2.00000	-29.000000	-48.000000	-99.000000
25%	2.00000	-1.000000	-26.000000	-86.000000
50%	6.00000	21.000000	-8.000000	-80.000000
75%	16.00000	49.000000	3.000000	-13.000000
max	30.00000	79.000000	88.000000	51.000000

```
df.dtypes
```

```
w            int64
x            int64
y            int64
z            int64
newindex    object
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, India to USA
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   w         5 non-null      int64
 1   x         5 non-null      int64
 2   y         5 non-null      int64
 3   z         5 non-null      int64
 4   newindex  5 non-null      object
dtypes: int64(4), object(1)
memory usage: 412.0+ bytes
```

Reading CSV files

```
df = pd.read_csv('/content/Universities.csv')
```

```
df.head(5)
```

	Sector	University	Year	Completions	Geography
0	Private for-profit, 2-year	Pima Medical Institute-Las Vegas	2016	591	Nevada
1	Private for-profit, less-than 2-year	Healthcare Preparatory Institute	2016	28	Nevada
2	Private for-profit, less-than 2-year	Milan Institute-Las Vegas	2016	408	Nevada
3	Private for-profit, less-than 2-year	Utah College of Massage Therapy-Vegas	2016	240	Nevada
4	Public, 4-year or above	Western Nevada College	2016	960	Nevada

Step 1 :

Groupby the categorical column. example : Year, university, Geography

```
df.groupby('Year')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at
0x7b3f8c59be50>
```

Step 2 :

Perform an aggreggation function

```
df.groupby('Year').mean()
```

```
nput-54-bca8f719f7cd>:1: FutureWarning: The default value of numer
by('Year').mean()
letions
```

	5.078947
	6.150000
	8.809524
	7.250000
	9.860465

```
df.groupby('Year').std()
```

```
<ipython-input-55-17db71a5e5b5>:1: FutureWarning: The default val
  df.groupby('Year').std()
```

Year	Completions
2012	1036.433239
2013	1040.474782
2014	1150.355857
2015	1183.371791
2016	1235.952796

```
df.groupby('Year').max().sort_index()
```

Year	Sector	University	Completions	Geography
2012	Public, 4-year or above	Western Nevada College	5388	Nevada
2013	Public, 4-year or above	Western Nevada College	5278	Nevada
2014	Public, 4-year or above	Western Nevada College	5093	Nevada
2015	Public, 4-year or above	Western Nevada College	5335	Nevada
2016	Public, 4-year or above	Wongu University of Oriental Medicine	5367	Nevada

Groupby Multiple Colum

```
df.groupby(['Sector','Year']).var()
```

```
<ipython-input-57-5c51a5b2d424>:1: FutureWarning: The default val
  df.groupby(['Sector','Year']).var()
```

Sector	Year	Completions
Private for-profit, 2-year	2012	3.089531e+04
	2013	3.334096e+04
	2014	2.996696e+04
	2015	3.659573e+04
	2016	2.618345e+04
Private for-profit, 4-year or above	2012	8.612667e+03
	2013	3.008500e+03
	2014	5.253640e+04
	2015	3.650621e+04
	2016	2.439707e+04
Private for-profit, less-than 2-year	2012	2.157762e+04
	2013	1.966067e+04
	2014	1.487514e+04
	2015	2.202198e+04
	2016	2.268029e+04
Private not-for-profit, 2-year	2012	8.282450e+04
	2013	6.516050e+04
	2014	4.470050e+04
	2015	4.176050e+04
	2016	1.036800e+04
Private not-for-profit, 4-year or above	2012	9.849000e+03
	2013	8.933333e+03
	2014	5.114333e+03
	2015	1.085633e+04
	2016	5.977400e+04
Public, 2-year	2012	NaN
	2013	NaN

	2013	NaN
	2014	NaN
	2015	NaN
	2016	NaN
Public, 4-year or above	**2012**	4.266085e+06
	2013	4.228619e+06
	2014	4.559649e+06
	2015	4.913344e+06
	2016	5.107756e+06

Multiple Aggreggation function

```
df.groupby('Year').describe()
```

	Completions							
	count	mean	std	min	25%	50%	75%	max
Year								
2012	38.0	535.078947	1036.433239	13.0	114.25	229.5	420.50	5388.0
2013	40.0	526.150000	1040.474782	0.0	98.50	189.0	413.00	5278.0
2014	42.0	588.809524	1150.355857	0.0	104.50	203.5	371.75	5093.0
2015	44.0	597.250000	1183.371791	0.0	87.75	191.0	405.75	5335.0
2016	43.0	609.860465	1235.952796	0.0	90.00	208.0	414.00	5367.0