Pandas

---

python library for Data Processing / Data Analysis

Why Pandas not Numpy ?

1. numpy only works with homogenous data, specially numbers
2. pandas deals with categorical data and text data
3. easy to use
4. user freindly/ higher abstraction
5. API like library

user -> python -> c/c++/ modula3 -> byte code -> machine code -> kernel

API -> can be created uisng python -> using machine learning -> data -> complex processing
ex - text extraction from image

Pandas

Data Type / Data Structure in Pandas

1. `series` - a column table or a one dimensional array holding any data type
2. `DataFrame` - 2-D Data Structure / Data is aligned in a tabular fashion in rows and columns / collection of series
3. `Pannel` - 3-D Data, collection of Data Frame

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

Creating Series

```
s = pd.Series([10, 20, 30, 40, 50, 60])
```

```
n = np.array([10, 20, 30, 40, 50, 60])


print(n)    # row-vector
```

```
[10 20 30 40 50 60]
```

```
print(s)    # coloumn vector
```

```
0    10
1    20
2    30
3    40
4    50
5    60
dtype: int64
```

```
s = pd.Series(["A", 'B', "C", 'D', 'E', 'F'])


s
```

```
0    A
1    B
2    C
3    D
4    E
5    F
dtype: object
```

```
print(s[0])
```

```
A
```

```
# print(s[-1])


s = pd.Series([10, 20, 30, 40, 50, 60])
s.index = ["A", 'B', "C", 'D', 'E', 'F']
s    # manual indexing
```

```
A    10
B    20
C    30
D    40
E    50
F    60
dtype: int64
```

```
s['A']
```

```
10
```

```
s['D']
```

```
40
```

```
print(s['B' : 'D'])
```

```
B    20
C    30
D    40
dtype: int64
```

```
s = pd.Series([10, 20, 30, 40, 50, 60])
s.index = []
```

```
s[0]
```

```
10
```

```
s
```

```
A    10
B    20
C    30
D    40
E    50
F    60
dtype: int64
```

```
s+1
```

```
A    11
B    21
C    31
D    41
E    51
F    61
dtype: int64
```

```
s*10
```

```
A    100
B    200
C    300
D    400
E    500
F    600
dtype: int64
```

```
np.array([1, 2, 3, 4, 5]) * 10
```

```
array([10, 20, 30, 40, 50])
```

```
s.astype(float)
```

```
A    10.0
B    20.0
C    30.0
D    40.0
E    50.0
F    60.0
dtype: float64
```

```
s.min()
```

```
10
```

```
s.max()
```

```
60
```

```
s.median()
```

```
35.0
```

```
s.std()
```

```
18.708286933869708
```

```
s.describe()  # 5 point summary/ stats
```

```
count     6.000000
mean     35.000000
std      18.708287
min      10.000000
25%      22.500000
50%      35.000000
75%      47.500000
max      60.000000
dtype: float64
```

```
np.exp(s)
```

```
A    2.202647e+04
B    4.851652e+08
C    1.068647e+13
```

```
D    2.353853e+17
E    5.184706e+21
F    1.142007e+26
dtype: float64
```

```
np.sin(s)
```

```
A   -0.544021
B    0.912945
C   -0.988032
D    0.745113
E   -0.262375
F   -0.304811
dtype: float64
```

```
np.log(s)
```

```
A    2.302585
B    2.995732
C    3.401197
D    3.688879
E    3.912023
F    4.094345
dtype: float64
```

```
#boolean indexing
```

```
s
```

```
A    10
B    20
C    30
D    40
E    50
F    60
dtype: int64
```

```
s>30
```

```
A    False
B    False
C    False
D     True
E     True
F     True
dtype: bool
```

```
s[s<30] = 0
```

```
s
```

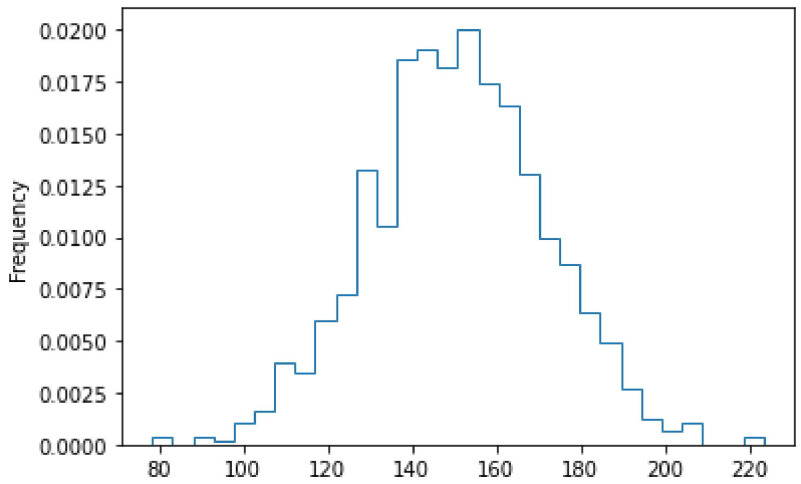
```
A     0
B     0
C    30
D    40
E    50
F    60
dtype: int64
```

```
s1 = pd.Series(np.random.normal(150, 20, 1000))
s1
```

```
0      131.743272
1      129.199743
2      148.013594
3      166.001403
4      137.011363
          ...
995    165.573985
996    138.967312
997    176.139847
998    147.439122
999    149.449656
Length: 1000, dtype: float64
```

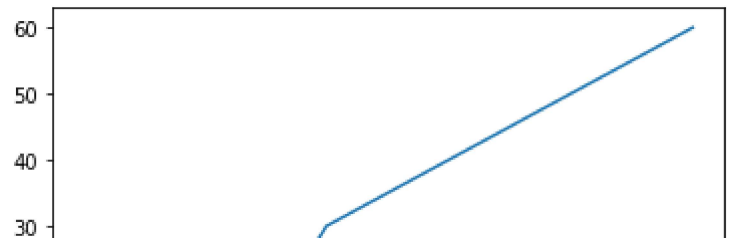
```
s1.plot(kind = 'hist', histtype = 'step', bins = 30, density=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68060c4690>
```




```
s.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6804d0c1d0>
```




```
s
```

```
A     0
B     0
C    30
D    40
E    50
F    60
dtype: int64
```

```
s.to_numpy()
```

```
array([ 0,  0, 30, 40, 50, 60])
```

```
s
```

```
A     0
B     0
C    30
D    40
E    50
F    60
dtype: int64
```

```
s2 = pd.Series([70, 76, 43, 56, 79, 88], index=['rahul', 'arun', 'varun', 'isha', 'amit', 'su
s2
```

```
rahul    70
arun     76
varun    43
isha     56
amit     79
sumit    88
dtype: int64
```

```
s2['isha']
```

```
56
```

```
s2[3]  # default indexing -- 0 based
```

```
56
```

Date Index

```
pd.date_range('from', 'to')
```

- used to generate date indexes
- ('month-day-year')

```
d = pd.date_range('09-01-2021', '09-10-2021')
print(d)
```

```
DatetimeIndex(['2021-09-01', '2021-09-02', '2021-09-03', '2021-09-04',
               '2021-09-05', '2021-09-06', '2021-09-07', '2021-09-08',
               '2021-09-09', '2021-09-10'],
              dtype='datetime64[ns]', freq='D')
```

```
a = np.random.randint(10, 50, 10)
```

```
a
```

```
array([37, 37, 10, 24, 36, 41, 41, 32, 49, 15])
```

```
sales = pd.Series(a, index= d)
```

```
sales
```

```
2021-09-01    37
2021-09-02    37
2021-09-03    10
2021-09-04    24
2021-09-05    36
2021-09-06    41
2021-09-07    41
2021-09-08    32
2021-09-09    49
2021-09-10    15
Freq: D, dtype: int64
```

```
sales['2021-09-08']
```

```
32
```

Create series using dictionary

```
height  = {
    'arun':144,
    'varun': 150,
    'amit':135,
    'sumit': 130,
    'kushal': 141,
    'rohit': 151,
    'rishi' : 148
}


h = pd.Series(height)


h
```

```
arun      144
varun     150
amit      135
sumit     130
kushal    141
rohit     151
rishi     148
dtype: int64
```

```
s = pd.Series(100, index= ["A", 'B', "C", 'D', 'E', 'F'], name = 'MLSeries')
s
```

```
A    100
B    100
C    100
D    100
E    100
F    100
Name: MLSeries, dtype: int64
```

```
s.name
```

```
'MLSeries'
```

Create a series with single value

- pass a scalar value
- broadcasting

we can also assign a name of series

```
s1 = pd.Series(100, index=['A', 'B', 'C', 'D', 'E'])
```

Data Frames

- collection of Series
- 2-D spreadsheet
- Two-dimensional, size-mutable,

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

```
df = pd.DataFrame({
    "name" : s2.index.to_list(),
    "gender" : pd.Categorical(["F", "M", "M", "F", "M", "M"]),
    "height": np.random.randint(120, 160, 6),
    "weight" : np.random.randint(40, 80, 6),
    "salary(k)" : np.random.randint(20, 60, 6)
}, index = ["A", "B", "C", "D", "E", "F"])

# pd.DataFrame( {dictionary} )

df
```

	name	gender	height	weight	salary(k)
A	rahul	F	159	64	24
B	arun	M	136	41	49

```
df['height']  #series
```

```
0    132
1    158
2    151
3    123
4    120
5    134
Name: height, dtype: int64
```

```
df.drop('gender', axis=1, inplace=True)

# row :  axis = 0  (by default)
# column  :  axis = 1
```

```
df
```

	name	height	weight	salary(k)
0	rahul	132	58	41
1	arun	158	48	52
2	varun	151	58	41
3	isha	123	60	41
4	amit	120	51	21
5	sumit	134	70	47

```
df.drop(2)
```

	name	height	weight	salary(k)
0	rahul	132	58	41
1	arun	158	48	52
3	isha	123	60	41
4	amit	120	51	21
5	sumit	134	70	47

data frame

1. store in new dataframe
2. inplace

Numerical Indexing

loc vs iloc

to select data on a DataFrame, Pandas loc and iloc are two top favorites. They are quick, fast, easy to read

1. loc is label-based, which means that you have to specify rows and columns based on their row and
2. iloc is integer position-based, so you have to specify rows and columns by their integer position

```
data = pd.DataFrame({
    'age': [10, 22, 13, 21, 12, 11, 17],
    'section': ['A', 'B', 'C', 'B', 'B', 'A', 'A'],
    'city': ['Gurgaon', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai'],
    'gender': ['M', 'F', 'F', 'M', 'M', 'M', 'F'],
    'favourite_color': ['red', np.NaN, 'yellow', np.NaN, 'black', 'green', 'red']
})
```

```
data
```

	age	section	city	gender	favourite_color
0	10	A	Gurgaon	M	red
1	22	B	Delhi	F	NaN
2	13	C	Mumbai	F	yellow
3	21	B	Delhi	M	NaN

```
data.iloc[4]  # default index
```

```
age                  12
section               B
city             Mumbai
gender                M
favourite_color     black
Name: 4, dtype: object
```

```
data1 = data.loc[data.age > 15]
data1
```

	age	section	city	gender	favourite_color
1	22	B	Delhi	F	NaN
3	21	B	Delhi	M	NaN
6	17	A	Mumbai	F	red

```
data1.iloc[2]
```

```
age                  17
section               A
city             Mumbai
gender                F
favourite_color     red
Name: 6, dtype: object
```

```
data
```

	age	section	city	gender	favourite_color

```
data.loc[2, 'city']
```

```
'Mumbai'
```

```
data.columns = ['A', "B", 'C', 'D', 'E']
data
```

	A	B	C	D	E
0	10	A	Gurgaon	M	red
1	22	B	Delhi	F	NaN
2	13	C	Mumbai	F	yellow
3	21	B	Delhi	M	NaN
4	12	B	Mumbai	M	black
5	11	A	Delhi	M	green
6	17	A	Mumbai	F	red

```
data.index = ['P', 'Q', 'R', 'S', 'T', 'U', 'V']
```

```
data
```

	A	B	C	D	E
P	10	A	Gurgaon	M	red
Q	22	B	Delhi	F	NaN
R	13	C	Mumbai	F	yellow
S	21	B	Delhi	M	NaN
T	12	B	Mumbai	M	black
U	11	A	Delhi	M	green
V	17	A	Mumbai	F	red

```
# slicing using loc
data.iloc[1:3]
```

	A	B	C	D	E
Q	22	B	Delhi	F	NaN

```
data.loc['P':'S']
```

	A	B	C	D	E
P	10	A	Gurgaon	M	red
Q	22	B	Delhi	F	NaN
R	13	C	Mumbai	F	yellow
S	21	B	Delhi	M	NaN

```
data = pd.DataFrame({
    'age' :     [ 10, 22, 13, 21, 12, 11, 17],
    'section' : [ 'A', 'B', 'C', 'B', 'B', 'A', 'A'],
    'city' :    [ 'Gurgaon', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai'],
    'gender' :  [ 'M', 'F', 'F', 'M', 'M', 'M', 'F'],
    'favourite_color' : [ 'red', np.NaN , 'yellow',np.NaN, 'black', 'green', 'red']
})

data
```

	age	section	city	gender	favourite_color
0	10	A	Gurgaon	M	red
1	22	B	Delhi	F	NaN
2	13	C	Mumbai	F	yellow
3	21	B	Delhi	M	NaN
4	12	B	Mumbai	M	black
5	11	A	Delhi	M	green
6	17	A	Mumbai	F	red

```
# rows where value of age is greater than or equal to 15
data.loc[data.age>15]
```

	age	section	city	gender	favourite_color
1	22	B	Delhi	F	NaN
3	21	B	Delhi	M	NaN

```
# finding all rows where age is greater than or equal to 12 and gender is male
data.loc[data.gender == 'M']
```

	age	section	city	gender	favourite_color
0	10	A	Gurgaon	M	red
3	21	B	Delhi	M	NaN
4	12	B	Mumbai	M	black
5	11	A	Delhi	M	green

```
# selct few columns with a condition
data.loc[(data.age > 12), ['age', 'city']]
```

	age	city
1	22	Delhi
2	13	Mumbai
3	21	Delhi
6	17	Mumbai

```
# update a column with a condition
data.loc[(data.age>15), ['gender']] = 'M'
```

```
data
```

	age	section	city	gender	favourite_color
0	10	A	Gurgaon	M	red
1	22	B	Delhi	M	NaN
2	13	C	Mumbai	F	yellow
3	21	B	Delhi	M	NaN
4	12	B	Mumbai	M	black
5	11	A	Delhi	M	green
6	17	A	Mumbai	M	red

```
# update multiple columns with a condititon
data.loc[ (data.age>20), ['section', 'city']] = ['S', 'Pune']
data
```

	age	section	city	gender	favourite_color
0	10	A	Gurgaon	M	red
1	22	S	Pune	M	NaN
2	13	C	Mumbai	F	yellow
3	21	S	Pune	M	NaN
4	12	B	Mumbai	M	black
5	11	A	Delhi	M	green
6	17	A	Mumbai	M	red

Concatenating Data Frames

```
# first dataframe

india_weather = pd.DataFrame(
                {
                    "city":["mumbai", "delhi", "banglore"],
                    "temperature" : [32, 45, 30],
                    "humidity": [80, 60, 78]
                })


# second data Frame

us_weather = pd.DataFrame({
    "city": ["new york", "chicago", "orlando"],
    "temperature" : [21, 14, 35],
    "humidity": [68, 65, 75]
})


us_weather
```

```
india_weather
```

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78

```
# column wise concatenate
# concatenating both india and US wether --> merging them

dfcol = pd.concat([india_weather, us_weather])
dfcol

#it will copy the index
```

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78
0	new york	21	68
1	chicago	14	65
2	orlando	35	75

```
# row-wise concatenate
dfrow = pd.concat([india_weather, us_weather], axis=1)
dfrow
```

	city	temperature	humidity	city	temperature	humidity
0	mumbai	32	80	new york	21	68
1	delhi	45	60	chicago	14	65
2	banglore	30	78	orlando	35	75

```
#axis = 0  -> coloumn [deafult]
#axis = 1 -> row
```

Merge DataFrames

```
# humidity dataframe

humidity_df = pd.DataFrame({
    "city": ["delhi", "mumbai", "banglore"],
    "humidity" : [68, 65, 75]
})

humidity_df
```

	city	humidity
0	delhi	68
1	mumbai	65
2	banglore	75

```
temperature_df = pd.DataFrame({
    "city": ["mumbai", "delhi", "banglore", "hyderabad"],
    "temperatur": [32, 45, 30, 40]
})

temperature_df
```

	city	temperatur
0	mumbai	32
1	delhi	45
2	banglore	30
3	hyderabad	40

```
df = pd.merge(temperature_df, humidity_df, on = "city")
df
```

	city	temperatur	humidity
0	mumbai	32	65
1	delhi	45	68
2	banglore	30	75

Inner Join -

[This is similar to the intersection of two sets] .

It returns a dataframe with only those rows that have common characteristics.

full join/ outer join -

[similar to Union of two Sets].

returns all those records which either have a match in the left or right dataframe.

```
df = pd.merge(temperature_df, humidity_df, on = "city", how = 'outer')
df
#how - inner -> intersection  (default)
#how - outer -> union
```

	city	temperatur	humidity
0	mumbai	32	65.0
1	delhi	45	68.0
2	banglore	30	75.0
3	hyderabad	40	NaN

Import csv and excel file

```
dfcsv = pd.read_csv('/content/dataset/weather_data_cities.csv')
dfcsv
```

	day	city	temperature	windspeed	event
0	1/1/2017	new york	32	6	Rain
1	1/2/2017	new york	36	7	Sunny
2	1/3/2017	new york	28	12	Snow
3	1/4/2017	new york	33	7	Sunny
4	1/1/2017	mumbai	90	5	Sunny
5	1/2/2017	mumbai	85	12	Fog
6	1/3/2017	mumbai	87	15	Fog
7	1/4/2017	mumbai	92	5	Rain

```
dfexc = pd.read_excel("/content/dataset/weather_data_cities.xlsx")
dfexc
```

	day	city	temperature	windspeed	event
0	2017-01-01	new york	32	6	Rain
1	2017-01-02	new york	36	7	Sunny
2	2017-01-03	new york	28	12	Snow
3	2017-01-04	new york	33	7	Sunny
4	2017-01-01	mumbai	90	5	Sunny
5	2017-01-02	mumbai	85	12	Fog
6	2017-01-03	mumbai	87	15	Fog
7	2017-01-04	mumbai	92	5	Rain
8	2017-01-01	paris	45	20	Sunny
9	2017-01-02	paris	50	13	Cloudy
10	2017-01-03	paris	54	8	Cloudy
11	2017-01-04	paris	42	10	Cloudy

Group-by

- Pandas dataframe.groupby() function is used to split the data into groups based on some criteri

- used for grouping the data according to the categories and apply a function to the categories.

```
g = dfexc.groupby('city')
g
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f68047e71d0>
```

```
for city, city_df in g:
    print(city)
    print(city_df)
```

```
mumbai
         day    city  temperature  windspeed  event
4 2017-01-01  mumbai           90          5  Sunny
5 2017-01-02  mumbai           85         12    Fog
6 2017-01-03  mumbai           87         15    Fog
7 2017-01-04  mumbai           92          5   Rain
new york
         day      city  temperature  windspeed  event
0 2017-01-01  new york           32          6   Rain
1 2017-01-02  new york           36          7  Sunny
2 2017-01-03  new york           28         12   Snow
3 2017-01-04  new york           33          7  Sunny
paris
          day   city  temperature  windspeed   event
8  2017-01-01  paris           45         20   Sunny
9  2017-01-02  paris           50         13  Cloudy
10 2017-01-03  paris           54          8  Cloudy
11 2017-01-04  paris           42         10  Cloudy
```

```
g.mean()
```

city	temperature	windspeed
mumbai	88.50	9.25
new york	32.25	8.00
paris	47.75	12.75

Saving and Serialising DataFrame

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

```
df
```

	city	temperatur	humidity
0	mumbai	32	65.0
1	delhi	45	68.0
2	banglore	30	75.0
3	hyderabad	40	NaN

```
df.to_csv("df_save.csv")
```

```
df.to_csv()
```

✓ 0s completed at 10:56 PM