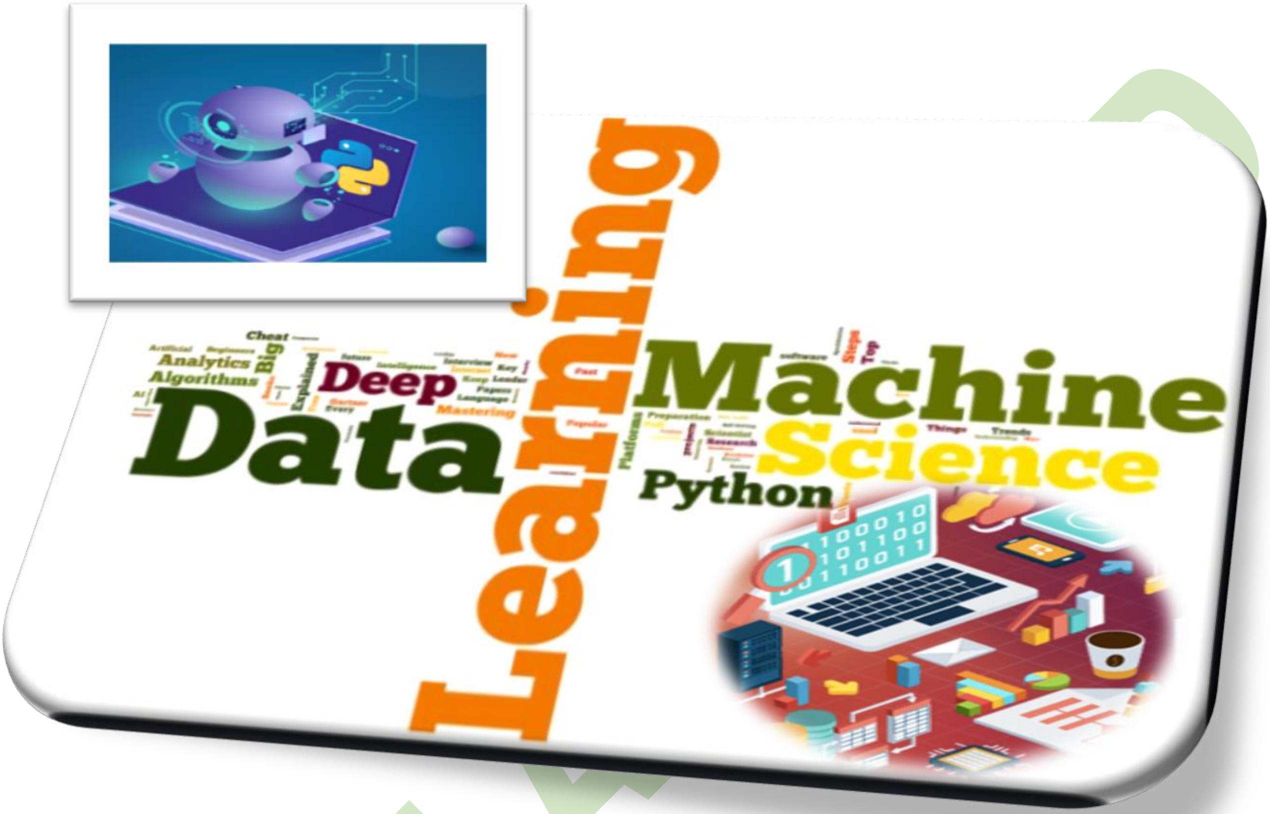
Data science and Machine learning with Python



Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.

Python for Data Analysis – Pandas (Continue)

We frequently find missing values in our data set. A quick method for imputing missing values is by filling the missing value with any random number. Not just missing values, you may find lots of outliers in your data set, which might require replacing. Let's see how can we replace values.

#Series function from pandas are used to create arrays

>>> data = pd.Series([1., -999., 2., -999., -1000., 3.])

>>> data

```
0        1.0
1     -999.0
2        2.0
3     -999.0
4    -1000.0
5        3.0
dtype: float64
```

#replace -999 with NaN values

>>> import numpy as np

>>> data.replace(-999, np.nan,inplace=True)

>>> data

```
0        1.0
1        NaN
2        2.0
3        NaN
4    -1000.0
5        3.0
dtype: float64
```

#We can also replace multiple values at once.

>>> data = pd.Series([1., -999., 2., -999., -1000., 3.])

>>> data.replace([-999,-1000],np.nan,inplace=True)

>>> data

```
0     1.0
1     NaN
2     2.0
3     NaN
4     NaN
5     3.0
dtype: float64
```

Now, let's learn how to rename column names and axis (row names).

>>> data = pd.DataFrame(np.arange(12).reshape((3, 4)),index=['Ohio', 'Colorado', 'New York'],columns=['one', 'two', 'three', 'four'])

>>> data

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
New York	8	9	10	11

#Using rename function

>>> data.rename(index = {'Ohio':'SanF'}, columns={'one':'one_p','two':'two_p'},inplace=True)

>>> data

	one_p	two_p	three	four
SanF	0	1	2	3
Colorado	4	5	6	7
New York	8	9	10	11

#You can also use string functions

>>> data.rename(index = str.upper, columns=str.title,inplace=True)

>>> data

	One_p	Two_p	Three	Four
SANF	0	1	2	3
COLORADO	4	5	6	7
NEW YORK	8	9	10	11

Next, we'll learn to categorize (bin) continuous variables.

>>> ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]

We'll divide the ages into bins such as 18-25, 26-35,36-60 and 60 and above.

#Understand the output - '(' means the value is included in the bin, '[' means the value is excluded

>>> bins = [18, 25, 35, 60, 100]

>>> cats = pd.cut(ages, bins)

>>> cats

```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35,
60], (25, 35]]
Length: 12
Categories (4, object): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

#To include the right bin value, we can do:

>>> pd.cut(ages,bins,right=False)

```
[[18, 25), [18, 25), [25, 35), [25, 35), [18, 25), ..., [25, 35), [60, 100), [35, 60), [35,
60), [25, 35)]
Length: 12
Categories (4, object): [[18, 25) < [25, 35) < [35, 60) < [60, 100)]
```

#Let's check how many observations fall under each bin

>>> pd.value_counts(cats)

```
(18, 25]      5
(35, 60]      3
(25, 35]      3
(60, 100]     1
dtype: int64
```

Also, we can pass a unique name to each label.

```
>>> bin_names = ['Youth', 'YoungAdult', 'MiddleAge', 'Senior']

>>> new_cats = pd.cut(ages, bins,labels=bin_names)

>>> pd.value_counts(new_cats)
```

Youth	5
MiddleAge	3
YoungAdult	3
Senior	1
dtype: int64	

#we can also calculate their cumulative sum

```
>>> pd.value_counts(new_cats).cumsum()
```

Youth	5
MiddleAge	3
YoungAdult	3
Senior	1
dtype: int64	

Let's proceed and learn about grouping data and creating pivots in pandas. It's an immensely important data analysis method which you'd probably have to use on every data set you work with.

```
>>> df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],

        'key2' : ['one', 'two', 'one', 'two', 'one'],

        'data1' : np.random.randn(5),

        'data2' : np.random.randn(5)})

>>> df
```

data1	data2	key1	key2
0	0.973599	0.001761	a
1	0.207283	-0.990160	a
2	1.099642	1.872394	b
3	0.939897	-0.241074	b
4	0.606389	0.053345	a

#calculate the mean of data1 column by key1

>>> grouped = df['data1'].groupby(df['key1'])

>>> grouped.mean()

```
key1
a     0.595757
b     1.019769
Name: data1, dtype: float64
```

Now, let's see how to slice the data frame.

>>> dates = pd.date_range('20130101',periods=6)

>>> df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))

>>> df

	A	B	C	D
2013-01-01	1.030816	-1.276989	0.837720	-1.490111
2013-01-02	-1.070215	-0.209129	0.604572	-1.743058
2013-01-03	1.524227	1.863575	1.291378	1.300696
2013-01-04	0.918203	-0.158800	-0.964063	-1.990779
2013-01-05	0.089731	0.114854	-0.585815	0.298772
2013-01-06	0.222260	0.435183	-0.045748	0.049898

#get first n rows from the data frame

```
>>> df[:3]
```

	A	B	C	D
2013-01-01	1.030816	-1.276989	0.837720	-1.490111
2013-01-02	-1.070215	-0.209129	0.604572	-1.743058
2013-01-03	1.524227	1.863575	1.291378	1.300696

#slice based on date range

```
>>> df['20130101':'20130104']
```

	A	B	C	D
2013-01-01	1.030816	-1.276989	0.837720	-1.490111
2013-01-02	-1.070215	-0.209129	0.604572	-1.743058
2013-01-03	1.524227	1.863575	1.291378	1.300696
2013-01-04	0.918203	-0.158800	-0.964063	-1.990779

#slicing based on column names

```
>>> df.loc[:,['A','B']]
```

	A	B
2013-01-01	1.030816	-1.276989
2013-01-02	-1.070215	-0.209129
2013-01-03	1.524227	1.863575
2013-01-04	0.918203	-0.158800
2013-01-05	0.089731	0.114854
2013-01-06	0.222260	0.435183

#slicing based on both row index labels and column names

>>> df.loc['20130102':'20130103',['A','B']]

	A	B
2013-01-02	-1.070215	-0.209129
2013-01-03	1.524227	1.863575

#slicing based on index of columns

>>> df.iloc[3] #returns 4th row (index is 3rd)

```
A     0.918203
B    -0.158800
C    -0.964063
D    -1.990779
Name: 2013-01-04 00:00:00, dtype: float64
```

#returns a specific range of rows

>>> df.iloc[2:4, 0:2]

	A	B
2013-01-03	1.524227	1.863575
2013-01-04	0.918203	-0.158800

#returns specific rows and columns using lists containing columns or row indexes

>>> df.iloc[[1,5],[0,2]]

	A	C
2013-01-02	-1.070215	0.604572
2013-01-06	0.222260	-0.045748

Similarly, we can do Boolean indexing based on column values as well. This helps in filtering a data set based on a pre-defined condition.

>>> df[df.A > 1]

	A	B	C	D
2013-01-01	1.030816	-1.276989	0.837720	-1.490111
2013-01-03	1.524227	1.863575	1.291378	1.300696

#we can copy the data set

>>> df2 = df.copy()

>>> df2['E']=['one', 'one','two','three','four','three']

>>> df2

	A	B	C	D	E
2013-01-01	1.030816	-1.276989	0.837720	-1.490111	one
2013-01-02	-1.070215	-0.209129	0.604572	-1.743058	one
2013-01-03	1.524227	1.863575	1.291378	1.300696	two
2013-01-04	0.918203	-0.158800	-0.964063	-1.990779	three
2013-01-05	0.089731	0.114854	-0.585815	0.298772	four
2013-01-06	0.222260	0.435183	-0.045748	0.049898	three

#select rows based on column values

>>> df2[df2['E'].isin(['two','four'])]

	A	B	C	D	E
2013-01-03	1.524227	1.863575	1.291378	1.300696	two
2013-01-05	0.089731	0.114854	-0.585815	0.298772	four

#select all rows except those with two and four

>>> df2[~df2['E'].isin(['two','four'])]

	A	B	C	D	E
2013-01-01	1.030816	-1.276989	0.837720	-1.490111	one
2013-01-02	-1.070215	-0.209129	0.604572	-1.743058	one
2013-01-04	0.918203	-0.158800	-0.964063	-1.990779	three
2013-01-06	0.222260	0.435183	-0.045748	0.049898	three

We can also use a query method to select columns based on a criterion. Let's see how!

#list all columns where A is greater than C

>>> df.query('A > C')

	A	B	C	D
2013-01-01	1.030816	-1.276989	0.837720	-1.490111
2013-01-03	1.524227	1.863575	1.291378	1.300696
2013-01-04	0.918203	-0.158800	-0.964063	-1.990779
2013-01-05	0.089731	0.114854	-0.585815	0.298772
2013-01-06	0.222260	0.435183	-0.045748	0.049898

#using OR condition

>>> df.query('A < B | C > A')

	A	B	C	D
2013-01-02	-1.070215	-0.209129	0.604572	-1.743058
2013-01-03	1.524227	1.863575	1.291378	1.300696
2013-01-05	0.089731	0.114854	-0.585815	0.298772
2013-01-06	0.222260	0.435183	-0.045748	0.049898

Pivot tables are extremely useful in analyzing data using a customized tabular format. I think, among other things, Excel is popular because of the pivot table option. It offers a super-quick way to analyze data.

#create a data frame

```
>>> data = pd.DataFrame({'group': ['a', 'a', 'a', 'b','b', 'b', 'c', 'c','c'],
            'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
>>> data
```

	group	ounces
0	a	4.0
1	a	3.0
2	a	12.0
3	b	6.0
4	b	7.5
5	b	8.0
6	c	3.0
7	c	5.0
8	c	6.0

#calculate means of each group

```
>>> data.pivot_table(values='ounces',index='group',aggfunc=np.mean)
```

```
group
a    6.333333
b    7.166667
c    4.666667
Name: ounces, dtype: float64
```

#calculate count by each group

>>> data.pivot_table(values='ounces',index='group',aggfunc='count')

```
group
a    3
b    3
c    3
Name: ounces, dtype: int64
```