Pandas:

Pandas is mainly used for data cleaning/wrangling/munging or ETL (extract,transform,load) operations. Pandas store data which is mainly in the form of data-frames. The pandas data frame is similar to MS-excel (it has rows and columns)

Reading data

In [2]:

```
import pandas as pd
```

In [4]:

```
#pandas has capbility to read data from common file formats like excel,csv,j
#importing data from csv format:
cols = ['preg_count','glucose','BP','skin_thick','insulin','BMI','pedigree',
df_csv = pd.read_csv(r"C:\Users\admin\Desktop\Pandas_sample_data - Copy.csv"
print (df_csv.columns)
df_csv.head()
```

```
Index(['preg_count', 'glucose', 'BP', 'skin_thick', 'insulin', 'BMI',
       'pedigree', 'age', 'class', 'names'],
      dtype='object')
```

Out[4]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age	class	names
1	6.0	148	70.0	20.0	0.0	28.2	0.526	38	1	xyz
2	NaN	85	71.0	21.0	0.0	29.2	1.526	39	1	pqr
3	8.0	69	72.0	22.0	0.0	30.2	2.526	40	0	abc
4	1.0	156	73.0	23.0	0.0	31.2	3.526	41	1	an
5	7.0	123	74.0	24.0	45.0	32.2	4.526	42	1	asdg

In [26]:

```python
#Importing data from excel format:
#The below is the sample data we will be working on, you can download the fi
df_excel = pd.read_excel(r"C:\Users\admin\Desktop\Pandas_sample_data - Copy.
df_excel.head(30) #have a look at complete data set
```

Out[26]:

```
    preg_count  glucose    BP  skin_thick  insulin   BMI  pedigree  age  class  gender  names
 1         6.0    148.0  70.0        20.0      0.0  28.2     0.526   38      1       M    NaN
 2         NaN     85.0  71.0        21.0      0.0   NaN     1.526   39      1       F    NaN
 3         8.0   1000.0  72.0        22.0      0.0  30.2     2.526  150      0       M    NaN
 4         1.0    156.0  73.0        23.0      0.0  31.2     3.526   41      1       M    NaN
 5         7.0    123.0  74.0        24.0     45.0  32.2     4.526   42      1       M    NaN
 6         5.0     78.0  75.0        25.0     15.0  33.2     5.526   43      0       F    NaN
 7         NaN    159.0  76.0        26.0     26.0  34.2     6.526   44      1       F    NaN
 8         NaN    162.0  77.0        27.0    124.0  35.2     7.526   45      1       F    NaN
 9         1.0     87.0  78.0        28.0      3.0  36.2     8.526   46      0       M    NaN
10         2.0     96.0  79.0         NaN      4.0  37.2     9.526   47      1       M    NaN
11         3.0     85.0  80.0        30.0      5.0  38.2    10.526   48      1       M    NaN
12         1.0    156.0  81.0        31.0      6.0  39.2    11.526   49      0       F    NaN
13         2.0    123.0   NaN        32.0      7.0  40.2    12.526   50      1       M    NaN
14         9.0    114.0  83.0         NaN      8.0  41.2    13.526   51      1       F    NaN
15         5.0    125.0  84.0        34.0      2.0  42.2    14.526   52      0       F    NaN
16         NaN     85.0  85.0        35.0     61.0  43.2    15.526   53      1       M    NaN
17         6.0     87.0  86.0        36.0      3.0  44.2    16.526   54      1       M    NaN
18         3.0      NaN  87.0        37.0      7.0  45.2    17.526   55      0       M    NaN
19         2.0     76.0  88.0        38.0      5.0  46.2    18.526   56      1       M    NaN
20         1.0     72.0  89.0        39.0      6.0  47.2    19.526   57      1       F    NaN
21         7.0     74.0   NaN        40.0      9.0  48.2    20.526   58      0       M    NaN
22         NaN    157.0  91.0        41.0      0.0  49.2    21.526   59      1       F    NaN
23         9.0    142.0  92.0        42.0      0.0  50.2    22.526   60      1       M    NaN
24         8.0    125.0  93.0         NaN      2.0  51.2    23.526   61      0       M    NaN
25         6.0    158.0  94.0        44.0      8.0  52.2    24.526   62      1       M    NaN
26         NaN    178.0  95.0        45.0      4.0  53.2    25.526   63      1       M    NaN
27         1.0    159.0  96.0        46.0      6.0  54.2    26.526   64      1       M    NaN
28         7.0    147.0  97.0        47.0      1.0  55.2    27.526   65      0       F    NaN
29         2.0    167.0  98.0        48.0      3.0  56.2    28.526   66      1       F    NaN
30         5.0     83.0  99.0         NaN      NaN  57.2    29.526   67      1       M    NaN
```

In [5]:

```
#Reading data from dictionary(Json format):
df_dict = {'preg_count':[10,12,9,6,15],'glucose':[123,145,126,120,110],'BP':
pd.DataFrame.from_dict(df_dict)
# df_dict['BP'][:3]
```

Out[5]:

	preg_count	glucose	BP
0	10	123	70
1	12	145	72
2	9	126	75
3	6	120	77
4	15	110	79

Exploring data

In [6]:

```
df_excel.dtypes #returns the data-type of each field
```

Out[6]:

```
preg_count    float64
glucose       float64
BP            float64
skin_thick    float64
insulin       float64
BMI           float64
pedigree      float64
age             int64
class           int64
gender         object
names         float64
dtype: object
```

In [7]:

```
df_excel.info() #returns no. of not-null values and data-types in each field
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30 entries, 1 to 30
Data columns (total 11 columns):
preg_count    24 non-null float64
glucose       29 non-null float64
BP            28 non-null float64
skin_thick    26 non-null float64
insulin       29 non-null float64
BMI           29 non-null float64
pedigree      30 non-null float64
age           30 non-null int64
class         30 non-null int64
gender        30 non-null object
names         0 non-null float64
dtypes: float64(8), int64(2), object(1)
memory usage: 2.8+ KB
```

```
In [8]:  df_excel['preg_count'].value_counts().sort_index() #gives the value count so
```

```
Out[8]: 1.0    5
        2.0    4
        3.0    2
        5.0    3
        6.0    3
        7.0    3
        8.0    2
        9.0    2
        Name: preg_count, dtype: int64
```

```
In [9]:  df_excel.describe() #returns all the stastistical parameters at once
```

Out[9]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	ag
count	24.000000	29.000000	28.000000	26.000000	29.000000	29.000000	30.000000	30.0000
mean	4.458333	151.965517	84.392857	33.884615	12.413793	43.165517	15.026000	56.1666
std	2.812691	166.751629	9.048590	8.769615	25.415852	8.575225	8.803408	19.6470
min	1.000000	72.000000	70.000000	20.000000	0.000000	28.200000	0.526000	38.0000
25%	2.000000	85.000000	76.750000	26.250000	2.000000	36.200000	7.776000	46.2500
50%	5.000000	125.000000	84.500000	34.500000	5.000000	43.200000	15.026000	53.5000
75%	7.000000	157.000000	92.250000	40.750000	8.000000	50.200000	22.276000	60.7500
max	9.000000	1000.000000	99.000000	48.000000	124.000000	57.200000	29.526000	150.0000

Stastistical Operations:

```
In [10]:  df_excel.mean() #average of all the values belonging to a particular attribu
```

```
Out[10]: preg_count      4.458333
         glucose       151.965517
         BP             84.392857
         skin_thick     33.884615
         insulin        12.413793
         BMI            43.165517
         pedigree       15.026000
         age            56.166667
         class           0.700000
         names                NaN
         dtype: float64
```

In [11]:

```
df_excel.mean(axis=1).head() #for axis=1, i.e across rows
```

Out[11]:

```
1     34.636222
2     31.218000
3    142.747333
4     36.636222
5     39.191778
dtype: float64
```

In [12]:

```
df_excel.var() #it is the square of the standard deviation of values from me
#distributed from mean
```

Out[12]:

```
preg_count        7.911232
glucose       27806.105911
BP               81.876984
skin_thick       76.906154
insulin         645.965517
BMI              73.534483
pedigree         77.500000
age             386.005747
class             0.217241
names                  NaN
dtype: float64
```

In [13]:

```
df_excel.std() #how much the values are deviating from the mean (small value
# a large deviation)
```

Out[13]:

```
preg_count      2.812691
glucose       166.751629
BP              9.048590
skin_thick      8.769615
insulin        25.415852
BMI             8.575225
pedigree        8.803408
age            19.647029
class           0.466092
names                NaN
dtype: float64
```

In [14]:

```
df_excel.skew() #its a measure of symmetry along the normal distribution
#positive value shows that it is skewed to right side of mean
#negative value shows that it is skewed to left side of mean
```

Out[14]:

```
preg_count    1.374773e-01
glucose       5.013250e+00
BP            3.014098e-02
skin_thick    4.020445e-03
insulin       3.551874e+00
BMI          -2.542836e-02
pedigree     -1.403646e-15
age           3.917756e+00
class        -9.195004e-01
names                  NaN
dtype: float64
```

In [15]:

```
df_excel.kurtosis()
#Kurtosis is a measure of the flatness or peakedness of a distribution compa
```

Out[15]:

```
preg_count    -1.469797
glucose       26.272521
BP            -1.281966
skin_thick    -1.263220
insulin       13.833833
BMI           -1.160735
pedigree      -1.200000
age           18.852659
class         -1.242126
names               NaN
dtype: float64
```

In [16]:

```
df_excel.min() #returns the min value of each attribute or field
```

Out[16]:

```
preg_count        1
glucose          72
BP               70
skin_thick       20
insulin           0
BMI            28.2
pedigree      0.526
age              38
class             0
gender            F
names          None
dtype: object
```

In [17]:

```
df_excel.max() #returns the max value of each attribute or field
```

Out[17]:

```
preg_count         9
glucose         1000
BP                99
skin_thick        48
insulin          124
BMI             57.2
pedigree      29.526
age              150
class              1
gender             M
names           None
dtype: object
```

In [18]:

```
df_excel.median() #return the middle value for each attribute or field
```

Out[18]:

```
preg_count      5.000
glucose       125.000
BP             84.500
skin_thick     34.500
insulin         5.000
BMI            43.200
pedigree       15.026
age            53.500
class           1.000
names             NaN
dtype: float64
```

In [19]:

```
df_excel.corr() #the correlation shows, how much the values are dependent on
#increases so does the salary of the person. so we can say they are strongly
#So correlation is the measure of how much the value varies with the change
```

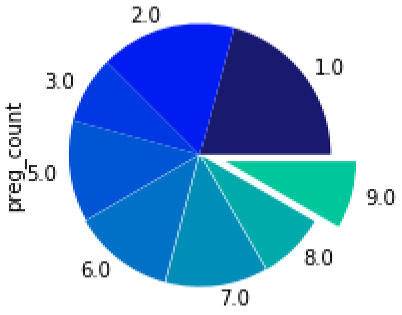
Out[19]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age
preg_count	1.000000	0.265195	0.016389	-0.037203	0.130925	0.050985	0.050985	0.30263
glucose	0.265195	1.000000	-0.222666	-0.223293	-0.074500	-0.263007	-0.228810	0.89689
BP	0.016389	-0.222666	1.000000	1.000000	-0.214819	1.000000	1.000000	0.17067
skin_thick	-0.037203	-0.223293	1.000000	1.000000	-0.212941	1.000000	1.000000	0.13415
insulin	0.130925	-0.074500	-0.214819	-0.212941	1.000000	-0.254070	-0.214212	-0.18821
BMI	0.050985	-0.263007	1.000000	1.000000	-0.254070	1.000000	1.000000	0.13363
pedigree	0.050985	-0.228810	1.000000	1.000000	-0.214212	1.000000	1.000000	0.17394
age	0.302632	0.896899	0.170679	0.134155	-0.188211	0.133633	0.173948	1.00000
class	-0.152380	-0.271347	0.072449	0.068580	0.199142	0.068021	0.029414	-0.27677
names	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Data Visualising:

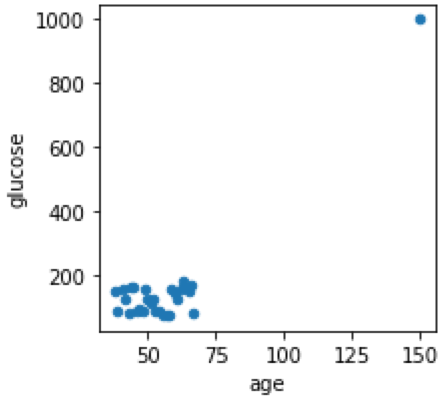
In [20]:
```
#It is actually required to present what lies within the data to anyone by u
%matplotlib inline
x = df_excel['preg_count'].value_counts().sort_index()
colors = ['#191970', '#001CF0', '#0038E2', '#0055D4', '#0071C6', '#008DB8',
'#00C69C']
explode = (0, 0, 0,0,0.01, 0.01, 0.015, 0.2)
x.plot(kind='pie',colors= colors, explode=explode,fontsize =10, figsize=(3,3
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x843353d668>
```



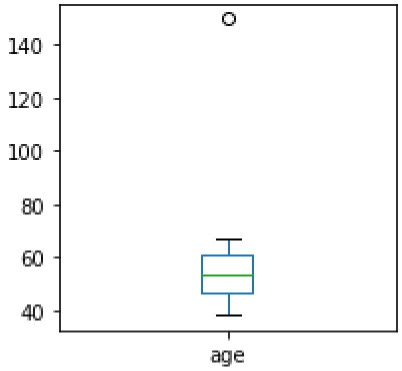
In [21]:
```
df_excel.plot.scatter('age','glucose',figsize=(3,3))
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x84335c9a58>
```



In [23]:
```
df_excel['age'].plot.box(figsize=(3,3))
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x84336b8208>
```



Data Cleaning:

Uncleaned data always gives the inaccurate model. It is meaningless to spend time on modelling if data is uncleaned. So clean data is critical for training models. Data scientist spend max of time in cleaning data. The uncleaned data can have:

1) Missing values 2) Infinite values 3) outliers 4) Errorneous values

we apply various methods to clean data:

identifying the missing values:

In [25]:

```
df_excel.count() #gives the count of number of not null values
```

Out[25]:

```
preg_count    24
glucose       29
BP            28
skin_thick    26
insulin       29
BMI           29
pedigree      30
age           30
class         30
gender        30
names          0
dtype: int64
```

In [28]:

```
pd.isna(df_excel).head(3) #returns the boolean result. The values which are
```

Out[28]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age	class	gender	names
1	False	False	False	False	False	False	False	False	False	False	True
2	True	False	False	False	False	True	False	False	False	False	True
3	False	False	False	False	False	False	False	False	False	False	True

Handling the missing values:

In [37]:

```
#It is very important to clean the data and remove the missing values or rep
df_excel.dropna(how='any',axis = 0)  #drop the rows with atleast one missing
df_excel.dropna(how='all',axis = 0)  #drop the rows with all missing values
df_excel.dropna(how='any', axis = 1)   #drop the columns with atleast one mi
df_excel.dropna(how='all', axis = 1)   #drop the columns with all missing va
#removed as it contained complete null values)
```

Out[37]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age	class	gender
1	6.0	148.0	70.0	20.0	0.0	28.2	0.526	38	1	M
2	NaN	85.0	71.0	21.0	0.0	NaN	1.526	39	1	F
3	8.0	1000.0	72.0	22.0	0.0	30.2	2.526	150	0	M
4	1.0	156.0	73.0	23.0	0.0	31.2	3.526	41	1	M
5	7.0	123.0	74.0	24.0	45.0	32.2	4.526	42	1	M
6	5.0	78.0	75.0	25.0	15.0	33.2	5.526	43	0	F
7	NaN	159.0	76.0	26.0	26.0	34.2	6.526	44	1	F
8	NaN	162.0	77.0	27.0	124.0	35.2	7.526	45	1	F
9	1.0	87.0	78.0	28.0	3.0	36.2	8.526	46	0	M
10	2.0	96.0	79.0	NaN	4.0	37.2	9.526	47	1	M
11	3.0	85.0	80.0	30.0	5.0	38.2	10.526	48	1	M
12	1.0	156.0	81.0	31.0	6.0	39.2	11.526	49	0	F
13	2.0	123.0	NaN	32.0	7.0	40.2	12.526	50	1	M
14	9.0	114.0	83.0	NaN	8.0	41.2	13.526	51	1	F
15	5.0	125.0	84.0	34.0	2.0	42.2	14.526	52	0	F
16	NaN	85.0	85.0	35.0	61.0	43.2	15.526	53	1	M
17	6.0	87.0	86.0	36.0	3.0	44.2	16.526	54	1	M
18	3.0	NaN	87.0	37.0	7.0	45.2	17.526	55	0	M
19	2.0	76.0	88.0	38.0	5.0	46.2	18.526	56	1	M
20	1.0	72.0	89.0	39.0	6.0	47.2	19.526	57	1	F
21	7.0	74.0	NaN	40.0	9.0	48.2	20.526	58	0	M
22	NaN	157.0	91.0	41.0	0.0	49.2	21.526	59	1	F
23	9.0	142.0	92.0	42.0	0.0	50.2	22.526	60	1	M
24	8.0	125.0	93.0	NaN	2.0	51.2	23.526	61	0	M
25	6.0	158.0	94.0	44.0	8.0	52.2	24.526	62	1	M
26	NaN	178.0	95.0	45.0	4.0	53.2	25.526	63	1	M
27	1.0	159.0	96.0	46.0	6.0	54.2	26.526	64	1	M
28	7.0	147.0	97.0	47.0	1.0	55.2	27.526	65	0	F
29	2.0	167.0	98.0	48.0	3.0	56.2	28.526	66	1	F
30	5.0	83.0	99.0	NaN	NaN	57.2	29.526	67	1	M

Filling or replacing the missing values:

In [38]:

```python
#filling the missing values by specified values
df_excel.fillna(value = 5).head()
```

Out[38]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age	class	gender	names
1	6.0	148.0	70.0	20.0	0.0	28.2	0.526	38	1	M	5.0
2	5.0	85.0	71.0	21.0	0.0	5.0	1.526	39	1	F	5.0
3	8.0	1000.0	72.0	22.0	0.0	30.2	2.526	150	0	M	5.0
4	1.0	156.0	73.0	23.0	0.0	31.2	3.526	41	1	M	5.0
5	7.0	123.0	74.0	24.0	45.0	32.2	4.526	42	1	M	5.0

In [42]:

```python
# Replacing the missing values by some statistical value (mean,median,mode)
df_excel['preg_count'] = df_excel['preg_count'].fillna(df_excel['preg_count'
df_excel['preg_count'].head() #you can notice that here the missing values i
```

Out[42]:

```
1    6.000000
2    4.458333
3    8.000000
4    1.000000
5    7.000000
Name: preg_count, dtype: float64
```

There might be certain values which are lying outside the valid range, for eg: in age column, the age = 150 is an outlier. such values actually deflects our mean. hence these values are to be replaced by other value such that it don't impact our data set.

In [48]:

```python
df_excel['age'].loc[df_excel['age']>100] = df_excel['age'].mean()
df_excel['age'].head()
```

Out[48]:

```
1    38.000000
2    39.000000
3    56.166667
4    41.000000
5    42.000000
Name: age, dtype: float64
```

Data Selection and Slicing

In [49]:
```python
df_excel.head(3) #to select the first 3 rows of data
```

Out[49]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age	class	gender	names
1	6.000000	148.0	70.0	20.0	0.0	28.2	0.526	38.000000	1	M	Na
2	4.458333	85.0	71.0	21.0	0.0	NaN	1.526	39.000000	1	F	Na
3	8.000000	1000.0	72.0	22.0	0.0	30.2	2.526	56.166667	0	M	Na

In [50]:
```python
df_excel.tail(3) #to select the last 3 rows of data
```

Out[50]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age	class	gender	names
28	7.0	147.0	97.0	47.0	1.0	55.2	27.526	65.0	0	F	NaN
29	2.0	167.0	98.0	48.0	3.0	56.2	28.526	66.0	1	F	NaN
30	5.0	83.0	99.0	NaN	NaN	57.2	29.526	67.0	1	M	NaN

In [53]:
```python
#Selection by label (loc)
df_excel.loc[1] #it will return the first row based on the serial number Loc
```

Out[53]:
```
preg_count        6
glucose         148
BP               70
skin_thick       20
insulin           0
BMI            28.2
pedigree      0.526
age              38
class             1
gender            M
names           NaN
Name: 1, dtype: object
```

In [54]:
```python
#Selection by index (iloc)
df_excel.iloc[0] #this will select the first row on the basis of index given
```

Out[54]:
```
preg_count        6
glucose         148
BP               70
skin_thick       20
insulin           0
BMI            28.2
pedigree      0.526
age              38
class             1
gender            M
names           NaN
Name: 1, dtype: object
```

In [ ]:
```python
#In the above case both represents the same row, but one selects on the basi
#on the basic of index location (iloc)
```

In [58]:

```python
#Filtering the data
df_excel[df_excel['age'] > 45].head() # filters out only those values where
```

Out[58]:

	preg_count	glucose	BP	skin_thick	insulin	BMI	pedigree	age	class	gender	name
3	8.0	1000.0	72.0	22.0	0.0	30.2	2.526	56.166667	0	M	Na
9	1.0	87.0	78.0	28.0	3.0	36.2	8.526	46.000000	0	M	Na
10	2.0	96.0	79.0	NaN	4.0	37.2	9.526	47.000000	1	M	Na
11	3.0	85.0	80.0	30.0	5.0	38.2	10.526	48.000000	1	M	Na
12	1.0	156.0	81.0	31.0	6.0	39.2	11.526	49.000000	0	F	Na

Reading data from a dictionary:

In [60]:

```python
dict_ = {'Names':['john','mike','smith','jack','tom'],'Age':[20,18,22,24,25]
dict_x = pd.DataFrame(dict_)
```

In [61]:

```python
#Applying string operations on a dataframe:
dict_x['Names'].str.upper() #converts each string into upper case
```

Out[61]:

```
0     JOHN
1     MIKE
2    SMITH
3     JACK
4      TOM
Name: Names, dtype: object
```

In [64]:

```python
dict_x['Names'] = dict_x['Names'].str.capitalize() #capitalize the first cha
dict_x['Names']
```

Out[64]:

```
0     John
1     Mike
2    Smith
3     Jack
4      Tom
Name: Names, dtype: object
```

In [ ]:

```python

```