The Engineering World #DataScience 1 & 2

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 FILTERING AND SELECTING DATA WITH PANDAS

```
In [32]: import numpy as np
         import pandas as pd
         from pandas import Series, DataFrame
```

1.0.1 Selecting and retriving data

```
In [33]: series_obj = Series(np.arange(8), index = ['row 1', 'row 2', 'row 3', 'row 4', 'row 5',

In [34]: series_obj

Out[34]: row 1    0
         row 2    1
         row 3    2
         row 4    3
         row 5    4
         row 6    5
         row 7    6
         row 8    7
         dtype: int64

In [35]: series_obj['row 7']

Out[35]: 6

In [36]: series_obj[[0,7]]

Out[36]: row 1    0
         row 8    7
         dtype: int64

In [37]: np.random.seed(25)
         DF_obj = DataFrame(np.random.rand(64) .reshape(8,8), index = ['row 1', 'row 2', 'row 3'
```

```
In [38]: DF_obj

Out[38]:        column 1  column 2  column 3  column 4  column 5  column 6  column 7  \
         row 1  0.870124  0.582277  0.278839  0.185911  0.411100  0.117376  0.684969
         row 2  0.556229  0.367080  0.402366  0.113041  0.447031  0.585445  0.161985
         row 3  0.326051  0.699186  0.366395  0.836375  0.481343  0.516502  0.383048
         row 4  0.514244  0.559053  0.034450  0.719930  0.421004  0.436935  0.281701
         row 5  0.669612  0.456069  0.289804  0.525819  0.559242  0.745284  0.828346
         row 6  0.077140  0.644862  0.309258  0.524254  0.958092  0.883201  0.295432
         row 7  0.088702  0.641717  0.132421  0.766486  0.076742  0.331044  0.679852
         row 8  0.655146  0.602120  0.719055  0.415219  0.396542  0.825139  0.712552

                column 8
         row 1  0.437611
         row 2  0.520719
         row 3  0.997541
         row 4  0.900274
         row 5  0.823694
         row 6  0.512376
         row 7  0.509213
         row 8  0.097937
```

```
In [39]: DF_obj.loc[['row 2', 'row 2'], ['column 5', 'column 2']]

Out[39]:        column 5  column 2
         row 2  0.447031   0.36708
         row 2  0.447031   0.36708
```

1.0.2 Data Slicing

```
In [40]: series_obj['row 3':'row 7']

Out[40]: row 3    2
         row 4    3
         row 5    4
         row 6    5
         row 7    6
         dtype: int64
```

1.0.3 Comparing with Scalars

```
In [41]: DF_obj < .2

Out[41]:        column 1  column 2  column 3  column 4  column 5  column 6  column 7  \
         row 1     False     False     False      True     False      True     False
         row 2     False     False     False      True     False     False      True
         row 3     False     False     False     False     False     False     False
         row 4     False     False      True     False     False     False     False
         row 5     False     False     False     False     False     False     False
```

```
row 6     True    False    False    False    False    False    False
row 7     True    False     True    False     True    False    False
row 8    False    False    False    False    False    False    False

       column 8
row 1     False
row 2     False
row 3     False
row 4     False
row 5     False
row 6     False
row 7     False
row 8      True
```

1.0.4 FIltering with scalars

```
In [42]: series_obj[series_obj > 6]
```

```
Out[42]: row 8    7
         dtype: int64
```

1.0.5 Setting values with scalars

```
In [43]: series_obj ['row 1', 'row 5', 'row 7', 'row 8'] = 8
```

```
In [44]: series_obj
```

```
Out[44]: row 1    8
         row 2    1
         row 3    2
         row 4    3
         row 5    8
         row 6    5
         row 7    8
         row 8    8
         dtype: int64
```

```
In [45]:  DF_obj ['row 1', 'row 5', 'row 8'] = 8
```

```
In [46]: DF_obj
```

```
Out[46]:        column 1  column 2  column 3  column 4  column 5  column 6  column 7  \
         row 1  0.870124  0.582277  0.278839  0.185911  0.411100  0.117376  0.684969
         row 2  0.556229  0.367080  0.402366  0.113041  0.447031  0.585445  0.161985
         row 3  0.326051  0.699186  0.366395  0.836375  0.481343  0.516502  0.383048
         row 4  0.514244  0.559053  0.034450  0.719930  0.421004  0.436935  0.281701
         row 5  0.669612  0.456069  0.289804  0.525819  0.559242  0.745284  0.828346
         row 6  0.077140  0.644862  0.309258  0.524254  0.958092  0.883201  0.295432
         row 7  0.088702  0.641717  0.132421  0.766486  0.076742  0.331044  0.679852
```

```
row 8  0.655146  0.602120  0.719055  0.415219  0.396542  0.825139  0.712552

       column 8  (row 1, row 5, row 8)
row 1  0.437611                      8
row 2  0.520719                      8
row 3  0.997541                      8
row 4  0.900274                      8
row 5  0.823694                      8
row 6  0.512376                      8
row 7  0.509213                      8
row 8  0.097937                      8
```

2 TREATING MISSING VALUES

In [47]:
```python
missing =np.NaN
series_obj = Series(['row 1', 'row 2', missing, 'row 4', 'row 5', missing, 'row 6'])
series_obj
```

Out[47]:
```
0    row 1
1    row 2
2      NaN
3    row 4
4    row 5
5      NaN
6    row 6
dtype: object
```

In [48]:
```python
series_obj
```

Out[48]:
```
0    row 1
1    row 2
2      NaN
3    row 4
4    row 5
5      NaN
6    row 6
dtype: object
```

In [49]:
```python
series_obj.isnull()
```

Out[49]:
```
0    False
1    False
2     True
3    False
4    False
5     True
6    False
dtype: bool
```

2.0.1 Filling on the missing values

In [50]:
```
np.random.seed(25)
DF_obj = DataFrame(np.random.randn(36) .reshape(6, 6))
DF_obj
```

Out[50]:

	0	1	2	3	4	5
0	0.228273	1.026890	-0.839585	-0.591182	-0.956888	-0.222326
1	-0.619915	1.837905	-2.053231	0.868583	-0.920734	-0.232312
2	2.152957	-1.334661	0.076380	-1.246089	1.202272	-1.049942
3	1.056610	-0.419678	2.294842	-2.594487	2.822756	0.680889
4	-1.577693	-1.976254	0.533340	-0.290870	-0.513520	1.982626
5	0.226001	-1.839905	1.607671	0.388292	0.399732	0.405477

In [51]:
```
DF_obj.loc[3:5, 0] = missing
DF_obj.loc[1:4, 5] = missing
```

In [52]:
```
DF_obj
```

Out[52]:

	0	1	2	3	4	5
0	0.228273	1.026890	-0.839585	-0.591182	-0.956888	-0.222326
1	-0.619915	1.837905	-2.053231	0.868583	-0.920734	NaN
2	2.152957	-1.334661	0.076380	-1.246089	1.202272	NaN
3	NaN	-0.419678	2.294842	-2.594487	2.822756	NaN
4	NaN	-1.976254	0.533340	-0.290870	-0.513520	NaN
5	NaN	-1.839905	1.607671	0.388292	0.399732	0.405477

In [53]:
```
filled_DF = DF_obj.fillna(0)
```

In [54]:
```
filled_DF
```

Out[54]:

	0	1	2	3	4	5
0	0.228273	1.026890	-0.839585	-0.591182	-0.956888	-0.222326
1	-0.619915	1.837905	-2.053231	0.868583	-0.920734	0.000000
2	2.152957	-1.334661	0.076380	-1.246089	1.202272	0.000000
3	0.000000	-0.419678	2.294842	-2.594487	2.822756	0.000000
4	0.000000	-1.976254	0.533340	-0.290870	-0.513520	0.000000
5	0.000000	-1.839905	1.607671	0.388292	0.399732	0.405477

In [55]:
```
filled_DF = DF_obj.fillna({0:0.1, 5:1.25})
filled_DF
```

Out[55]:

	0	1	2	3	4	5
0	0.228273	1.026890	-0.839585	-0.591182	-0.956888	-0.222326
1	-0.619915	1.837905	-2.053231	0.868583	-0.920734	1.250000
2	2.152957	-1.334661	0.076380	-1.246089	1.202272	1.250000
3	0.100000	-0.419678	2.294842	-2.594487	2.822756	1.250000
4	0.100000	-1.976254	0.533340	-0.290870	-0.513520	1.250000
5	0.100000	-1.839905	1.607671	0.388292	0.399732	0.405477

```
In [56]: filled_DF = DF_obj.fillna(method = 'ffill')
```

```
In [57]: filled_DF
```

```
Out[57]:           0         1         2         3         4         5
         0  0.228273  1.026890 -0.839585 -0.591182 -0.956888 -0.222326
         1 -0.619915  1.837905 -2.053231  0.868583 -0.920734 -0.222326
         2  2.152957 -1.334661  0.076380 -1.246089  1.202272 -0.222326
         3  2.152957 -0.419678  2.294842 -2.594487  2.822756 -0.222326
         4  2.152957 -1.976254  0.533340 -0.290870 -0.513520 -0.222326
         5  2.152957 -1.839905  1.607671  0.388292  0.399732  0.405477
```

2.0.2 Counting missing values

```
In [58]: np.random.seed(25)
         DF_obj = DataFrame(np.random.randn(36) .reshape(6, 6))
         DF_obj.loc[3:5, 0] = missing
         DF_obj.loc[1:4, 5] = missing
         DF_obj
```

```
Out[58]:           0         1         2         3         4         5
         0  0.228273  1.026890 -0.839585 -0.591182 -0.956888 -0.222326
         1 -0.619915  1.837905 -2.053231  0.868583 -0.920734       NaN
         2  2.152957 -1.334661  0.076380 -1.246089  1.202272       NaN
         3       NaN -0.419678  2.294842 -2.594487  2.822756       NaN
         4       NaN -1.976254  0.533340 -0.290870 -0.513520       NaN
         5       NaN -1.839905  1.607671  0.388292  0.399732  0.405477
```

```
In [59]: DF_obj.isnull().sum()
```

```
Out[59]: 0    3
         1    0
         2    0
         3    0
         4    0
         5    4
         dtype: int64
```

2.0.3 Filtering out missing values

```
In [60]: DF_no_NaN = DF_obj.dropna(axis = 1)
```

```
In [61]: DF_no_NaN
```

```
Out[61]:           1         2         3         4
         0  1.026890 -0.839585 -0.591182 -0.956888
         1  1.837905 -2.053231  0.868583 -0.920734
         2 -1.334661  0.076380 -1.246089  1.202272
         3 -0.419678  2.294842 -2.594487  2.822756
         4 -1.976254  0.533340 -0.290870 -0.513520
         5 -1.839905  1.607671  0.388292  0.399732
```

```
In [62]: DF_obj.dropna (how = 'all')
```

Out[62]:

	0	1	2	3	4	5
0	0.228273	1.026890	-0.839585	-0.591182	-0.956888	-0.222326
1	-0.619915	1.837905	-2.053231	0.868583	-0.920734	NaN
2	2.152957	-1.334661	0.076380	-1.246089	1.202272	NaN
3	NaN	-0.419678	2.294842	-2.594487	2.822756	NaN
4	NaN	-1.976254	0.533340	-0.290870	-0.513520	NaN
5	NaN	-1.839905	1.607671	0.388292	0.399732	0.405477

The Engineering World #DataScience 3 & 4

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 HOW TO REMOVE DUPLICATE DATA

1.0.1 Remove duplicate data

```
In [1]: import numpy as np
        import pandas as pd
        from pandas import Series, DataFrame

In [2]: DF_obj = DataFrame({'Roll': [1, 2, 3, 4, 5, 6, 7, 8, 3, 3, 5, 5, 5],
                            'Name': ['Akkal', 'Janak', 'Laxman', 'Dinesh', 'Amin', 'Bikash', 'Sum
                            'Marks': [10, 20, 30, 40, 50, 60, 70, 80, 30, 30, 50, 50, 50]})

In [3]: DF_obj

Out[3]:     Marks    Name  Roll
        0      10   Akkal     1
        1      20   Janak     2
        2      30  Laxman     3
        3      40  Dinesh     4
        4      50    Amin     5
        5      60  Bikash     6
        6      70   Sumil     7
        7      80   Kiran     8
        8      30  Laxman     3
        9      30  Laxman     3
        10     50    Amin     5
        11     50    Amin     5
        12     50    Amin     5

In [4]: DF_obj.duplicated()

Out[4]: 0     False
        1     False
```

```
2     False
3     False
4     False
5     False
6     False
7     False
8      True
9      True
10     True
11     True
12     True
dtype: bool
```

In [5]: `DF_obj.drop_duplicates()`

Out[5]:

	Marks	Name	Roll
0	10	Akkal	1
1	20	Janak	2
2	30	Laxman	3
3	40	Dinesh	4
4	50	Amin	5
5	60	Bikash	6
6	70	Sumil	7
7	80	Kiran	8

In [6]: `DF_obj.drop_duplicates(['Marks'])`

Out[6]:

	Marks	Name	Roll
0	10	Akkal	1
1	20	Janak	2
2	30	Laxman	3
3	40	Dinesh	4
4	50	Amin	5
5	60	Bikash	6
6	70	Sumil	7
7	80	Kiran	8

In [7]: `DF_obj`

Out[7]:

	Marks	Name	Roll
0	10	Akkal	1
1	20	Janak	2
2	30	Laxman	3
3	40	Dinesh	4
4	50	Amin	5
5	60	Bikash	6
6	70	Sumil	7
7	80	Kiran	8
8	30	Laxman	3

```
9       30  Laxman     3
10      50    Amin     5
11      50    Amin     5
12      50    Amin     5
```

2 CONCATINATING AND TRANSFORMING DATA

2.0.1 Concatinating Data

```
In [8]: DF_obj = pd.DataFrame(np.arange(36).reshape(6,6))
```

```
In [9]: DF_obj
```

```
Out[9]:     0   1   2   3   4   5
        0   0   1   2   3   4   5
        1   6   7   8   9  10  11
        2  12  13  14  15  16  17
        3  18  19  20  21  22  23
        4  24  25  26  27  28  29
        5  30  31  32  33  34  35
```

```
In [10]: DF_obj_2 = pd.DataFrame(np.arange(15).reshape(5,3))
```

```
In [11]: DF_obj_2
```

```
Out[11]:     0   1   2
         0   0   1   2
         1   3   4   5
         2   6   7   8
         3   9  10  11
         4  12  13  14
```

```
In [12]: pd.concat([DF_obj,DF_obj_2], axis = 1)
```

```
Out[12]:     0   1   2   3   4   5     0     1     2
         0   0   1   2   3   4   5   0.0   1.0   2.0
         1   6   7   8   9  10  11   3.0   4.0   5.0
         2  12  13  14  15  16  17   6.0   7.0   8.0
         3  18  19  20  21  22  23   9.0  10.0  11.0
         4  24  25  26  27  28  29  12.0  13.0  14.0
         5  30  31  32  33  34  35   NaN   NaN   NaN
```

```
In [13]: pd.concat([DF_obj,DF_obj_2])
```

```
Out[13]:     0   1   2     3     4     5
         0   0   1   2   3.0   4.0   5.0
         1   6   7   8   9.0  10.0  11.0
         2  12  13  14  15.0  16.0  17.0
         3  18  19  20  21.0  22.0  23.0
```

```
4  24  25  26  27.0  28.0  29.0
5  30  31  32  33.0  34.0  35.0
0   0   1   2   NaN   NaN   NaN
1   3   4   5   NaN   NaN   NaN
2   6   7   8   NaN   NaN   NaN
3   9  10  11   NaN   NaN   NaN
4  12  13  14   NaN   NaN   NaN
```

2.0.2 Trandforming Data

Dropping data

```
In [14]: DF_obj.drop([0,2])
```

```
Out[14]:     0   1   2   3   4   5
         1   6   7   8   9  10  11
         3  18  19  20  21  22  23
         4  24  25  26  27  28  29
         5  30  31  32  33  34  35
```

```
In [15]: DF_obj.drop([0,2], axis = 1)
```

```
Out[15]:     1   3   4   5
         0   1   3   4   5
         1   7   9  10  11
         2  13  15  16  17
         3  19  21  22  23
         4  25  27  28  29
         5  31  33  34  35
```

Adding Data

```
In [16]: series_obj = Series(np.arange(6))
         series_obj.name = 'aded_variables'
         series_obj
```

```
Out[16]: 0    0
         1    1
         2    2
         3    3
         4    4
         5    5
         Name: aded_variables, dtype: int64
```

```
In [17]: variable_added = DataFrame.join(DF_obj,series_obj)
```

```
In [18]: variable_added
```

Out[18]:

	0	1	2	3	4	5	aded_variables
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1
2	12	13	14	15	16	17	2
3	18	19	20	21	22	23	3
4	24	25	26	27	28	29	4
5	30	31	32	33	34	35	5

In [19]:
```
added_datatable = variable_added.append(variable_added, ignore_index = False)
```

In [20]:
```
added_datatable
```

Out[20]:

	0	1	2	3	4	5	aded_variables
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1
2	12	13	14	15	16	17	2
3	18	19	20	21	22	23	3
4	24	25	26	27	28	29	4
5	30	31	32	33	34	35	5
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1
2	12	13	14	15	16	17	2
3	18	19	20	21	22	23	3
4	24	25	26	27	28	29	4
5	30	31	32	33	34	35	5

In [21]:
```
added_datatable = variable_added.append(variable_added, ignore_index = True)
added_datatable
```

Out[21]:

	0	1	2	3	4	5	aded_variables
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1
2	12	13	14	15	16	17	2
3	18	19	20	21	22	23	3
4	24	25	26	27	28	29	4
5	30	31	32	33	34	35	5
6	0	1	2	3	4	5	0
7	6	7	8	9	10	11	1
8	12	13	14	15	16	17	2
9	18	19	20	21	22	23	3
10	24	25	26	27	28	29	4
11	30	31	32	33	34	35	5

Sorting data

In [22]:
```
DF_sorted = DF_obj.sort_values(by = [5], ascending = [False])
```

In [23]:
```
DF_sorted
```

Out[23]:

	0	1	2	3	4	5
5	30	31	32	33	34	35
4	24	25	26	27	28	29
3	18	19	20	21	22	23
2	12	13	14	15	16	17
1	6	7	8	9	10	11
0	0	1	2	3	4	5

In [24]:
```
DF_sorted = DF_obj.sort_values(by = [5], ascending = [True])
```

In [25]:
```
DF_sorted
```

Out[25]:

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	6	7	8	9	10	11
2	12	13	14	15	16	17
3	18	19	20	21	22	23
4	24	25	26	27	28	29
5	30	31	32	33	34	35

In [26]:
```
DF_sorted = DF_obj.sort_values(by = [5])
DF_sorted
```

Out[26]:

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	6	7	8	9	10	11
2	12	13	14	15	16	17
3	18	19	20	21	22	23
4	24	25	26	27	28	29
5	30	31	32	33	34	35

The Engineering World #DataScience 5 & 6

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 GROUPING AND AGGREGATE DATA

```
In [1]: import numpy as np
        import pandas as pd
        from pandas import Series, DataFrame
```

1.0.1 Grouping data by column index

```
In [2]: address = 'mtcars.csv'

In [3]: cars = pd.read_csv(address)

In [4]: cars.columns = ['car_name', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am'

In [5]: cars.head()

Out[5]:             car_name   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
        0          Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
        1      Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
        2         Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
        3     Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
        4  Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

           carb
        0     4
        1     4
        2     1
        3     1
        4     2

In [6]: cars_groups = cars.groupby(cars['cyl'])

In [7]: cars_groups.mean()
```

```
Out[7]:           mpg        disp          hp      drat        wt       qsec  \
        cyl
        4    26.663636  105.136364   82.636364  4.070909  2.285727  19.137273
        6    19.742857  183.314286  122.285714  3.585714  3.117143  17.977143
        8    15.100000  353.100000  209.214286  3.229286  3.999214  16.772143

                   vs        am      gear      carb
        cyl
        4    0.909091  0.727273  4.090909  1.545455
        6    0.571429  0.428571  3.857143  3.428571
        8    0.000000  0.142857  3.285714  3.500000
```

2 LINE, BAR AND PIE PLOTS

```
In [8]: import numpy as np
        import pandas as pd
        from pandas import Series, DataFrame
        from numpy.random import randn
        import matplotlib.pyplot as plt
        from matplotlib import rcParams
        import seaborn as sb
```

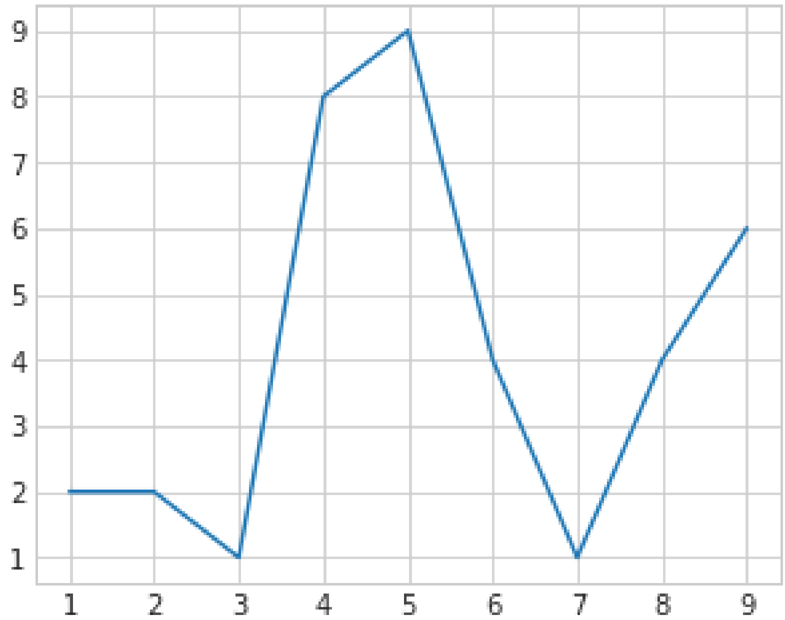
```
In [9]: %matplotlib inline
        rcParams['figure.figsize'] = 5, 4
        sb.set_style('whitegrid')
```

2.0.1 Creating a line chart from a list object

2.0.2 Plotting line chart in matplotlib

```
In [10]: x = range(1,10)
         y = [2, 2, 1, 8, 9, 4, 1, 4, 6]
         plt.plot(x,y)
```

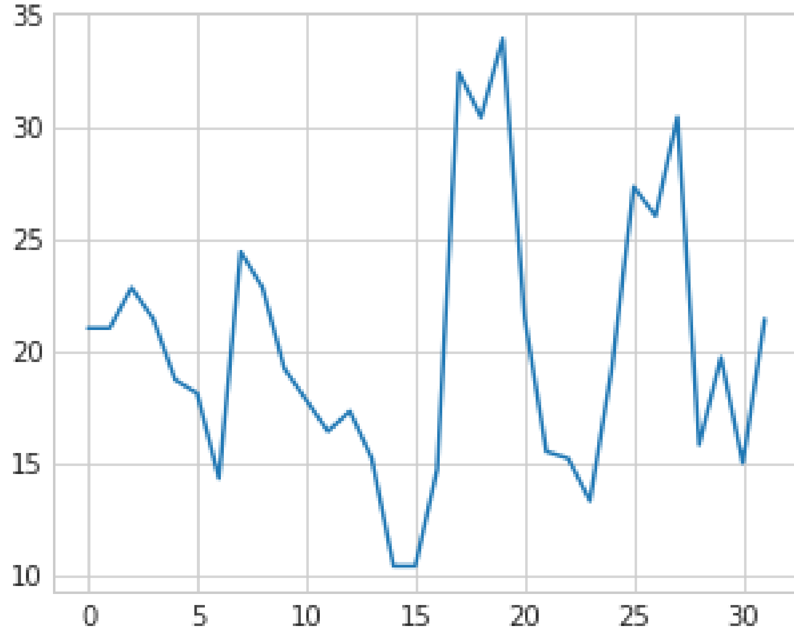
```
Out[10]: [<matplotlib.lines.Line2D at 0x7fda1481a240>]
```



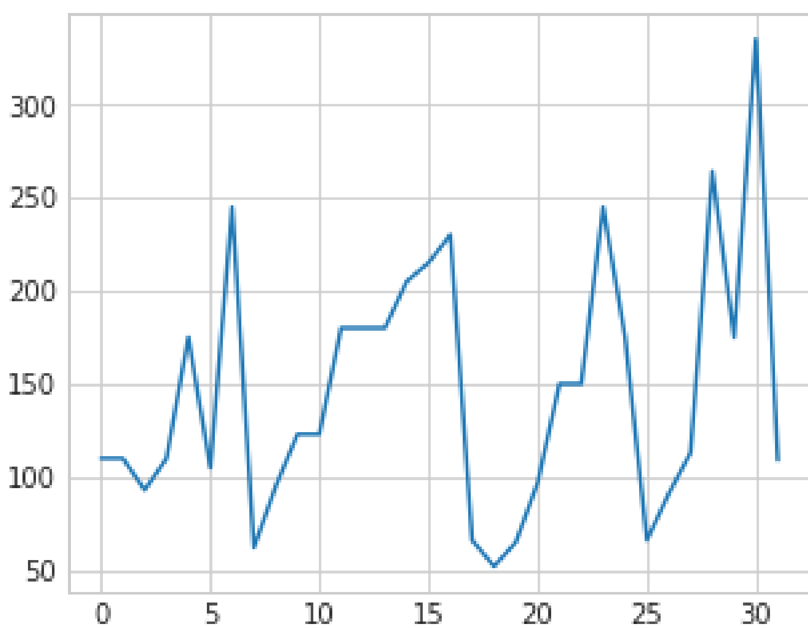
2.0.3 Plotting a line chart from a Pandas object

```
In [11]: address = 'mtcars.csv'
         cars = pd.read_csv(address)
         cars.columns = ['car_name', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
         mpg = cars['mpg']
         mpg.plot()
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda147c9710>
```

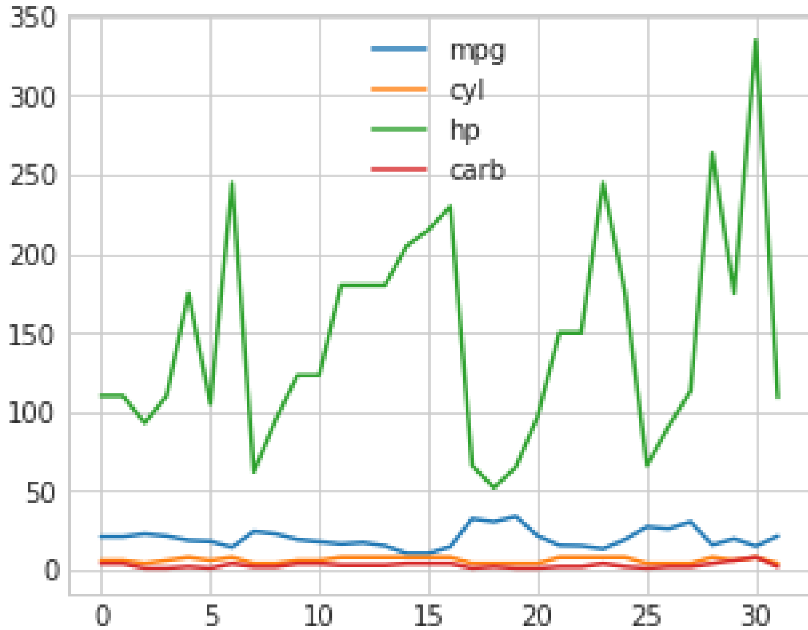



```
In [12]: hp = cars['hp']
         hp.plot()

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda147b4160>
```



```
In [13]: df = cars[['mpg', 'cyl', 'hp', 'carb']]
         df.plot()
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda147eb898>
```

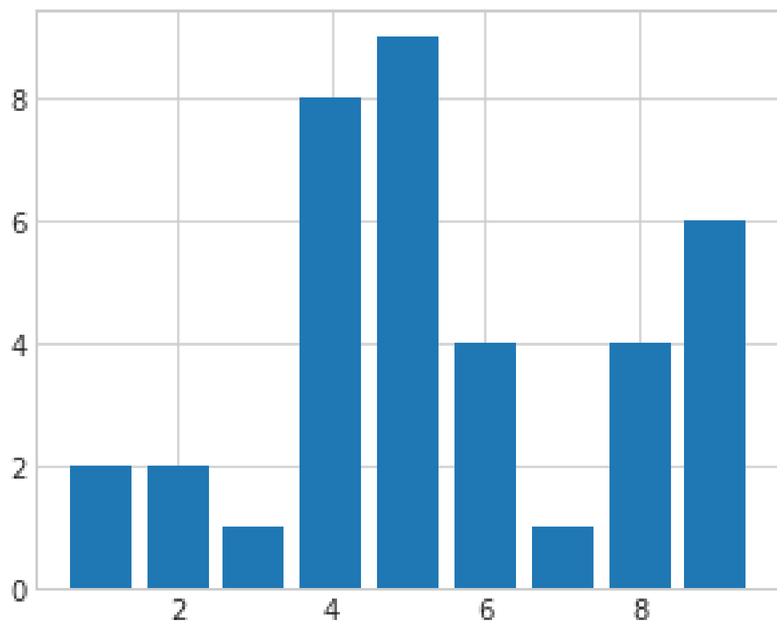


2.0.4 Creating Bar Chart

2.0.5 Creating a bar chart from a list

```
In [14]: plt.bar(x,y)
```

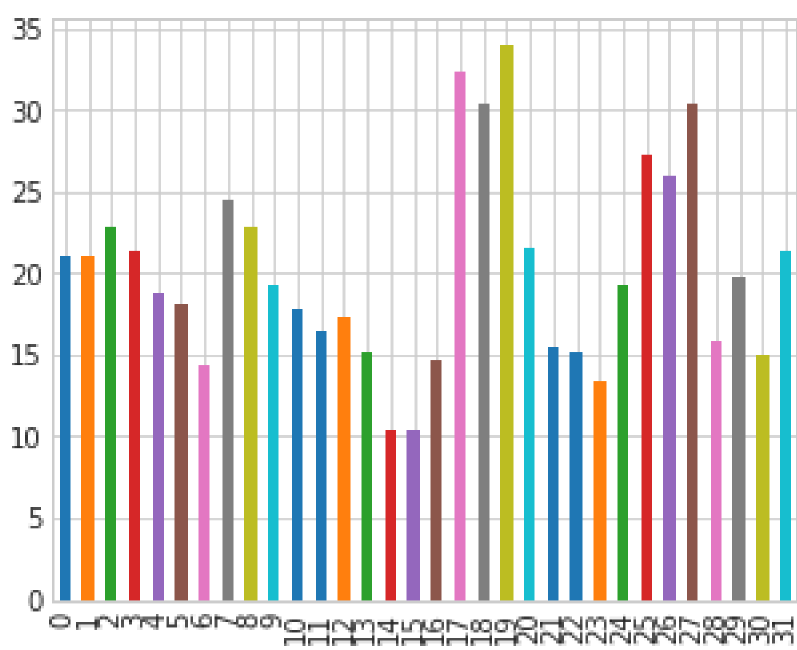
```
Out[14]: <Container object of 9 artists>
```



2.0.6 Creating bar chart from a pandas objects

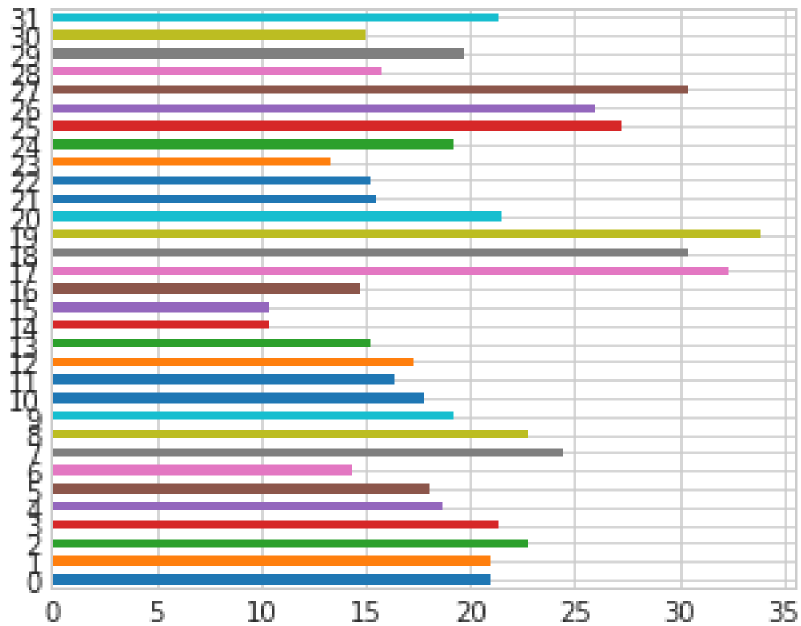
```
In [15]: mpg.plot(kind = 'bar')
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda4c654940>
```



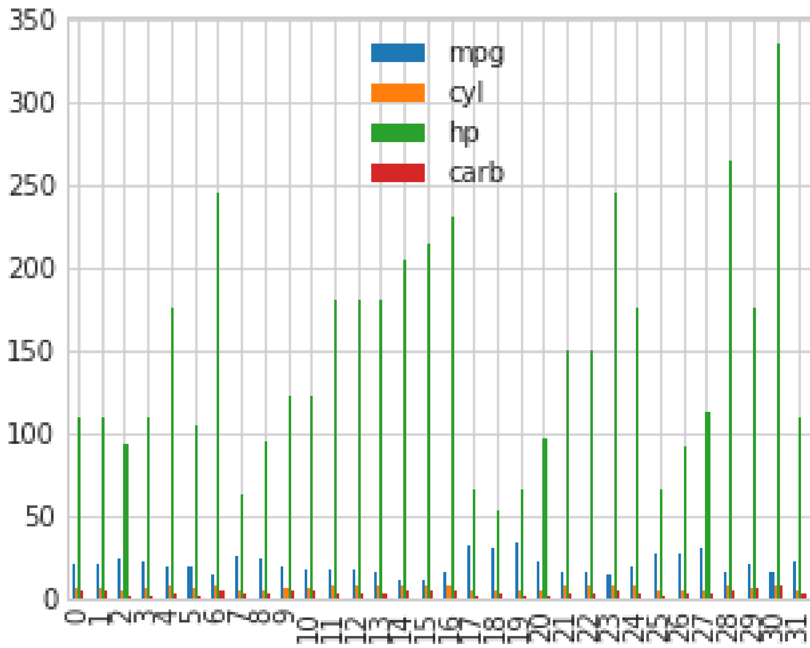
```
In [16]: mpg.plot(kind = 'barh')
```

```
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda14530278>
```




```
In [17]: df.plot(kind = 'bar')
```

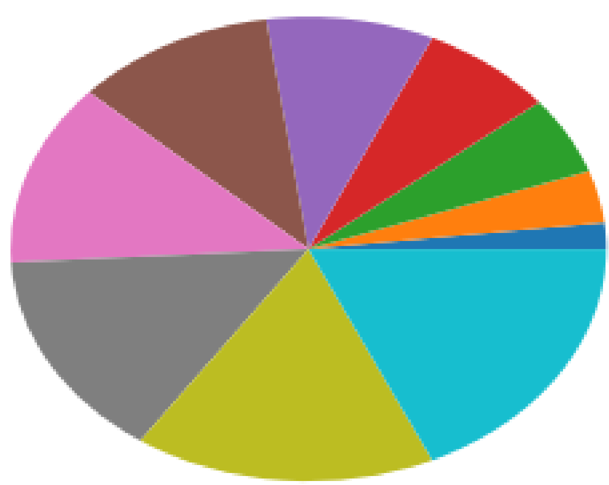
```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda1441bcf8>
```



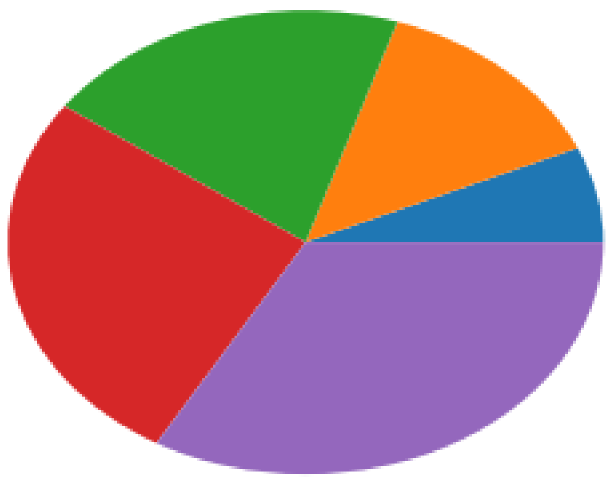
2.0.7 Creating a pie chart

```
In [18]: x = [1,2,3,4,5,6,7,8,9,10]
         plt.pie(x)
```

```
Out[18]: ([<matplotlib.patches.Wedge at 0x7fda141deef0>,
           <matplotlib.patches.Wedge at 0x7fda141ef400>,
           <matplotlib.patches.Wedge at 0x7fda141ef940>,
           <matplotlib.patches.Wedge at 0x7fda141efe80>,
           <matplotlib.patches.Wedge at 0x7fda141f6400>,
           <matplotlib.patches.Wedge at 0x7fda141f6940>,
           <matplotlib.patches.Wedge at 0x7fda141f6e80>,
           <matplotlib.patches.Wedge at 0x7fda14180400>,
           <matplotlib.patches.Wedge at 0x7fda14180940>,
           <matplotlib.patches.Wedge at 0x7fda14180e80>],
          [Text(1.09821,0.0627977,''),
           Text(1.07141,0.249146,''),
           Text(0.957821,0.540906,''),
           Text(0.671713,0.871092,''),
           Text(0.156546,1.0888,''),
           Text(-0.513334,0.972876,''),
           Text(-1.03603,0.369654,''),
           Text(-0.957821,-0.540906,''),
           Text(-0.0941326,-1.09596,''),
           Text(0.925379,-0.594705,'')])
```



```
In [19]: y = [1,2,3,4,5]
         plt.pie(y)

Out[19]: ([<matplotlib.patches.Wedge at 0x7fda14144f28>,
           <matplotlib.patches.Wedge at 0x7fda1414d438>,
           <matplotlib.patches.Wedge at 0x7fda1414d978>,
           <matplotlib.patches.Wedge at 0x7fda1414deb8>,
           <matplotlib.patches.Wedge at 0x7fda14157438>],
          [Text(1.07596,0.228703,''),
           Text(0.736044,0.817459,''),
           Text(-0.339919,1.04616,''),
           Text(-1.07596,-0.228703,''),
           Text(0.55,-0.952628,'')])
```

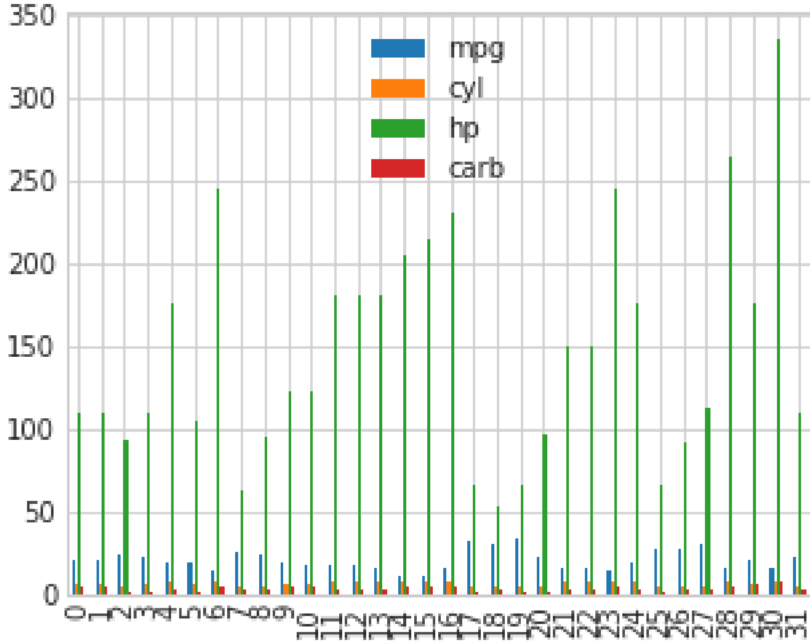


2.0.8 Saving a Plot

```
In [20]: plt.savefig('pie_chart.png')
         plt.show()

<matplotlib.figure.Figure at 0x7fda4c6b4e48>

In [21]: %pwd

Out[21]: '/home/akkal/Documents/DS_PYTHON/TheEngineeringWorld'
```

df.plot(kind = 'bar') plt.savefig('bar_chart.png') plt.show()

In [22]:
```
df.plot(kind = 'bar')
plt.savefig('bar_chart.png')
plt.show()
```



The Engineering World #DataScience 7 & 8

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 TITLE AND MARKERS IN PLOT ELEMENTS

1.0.1 Defining Elements of a Plot

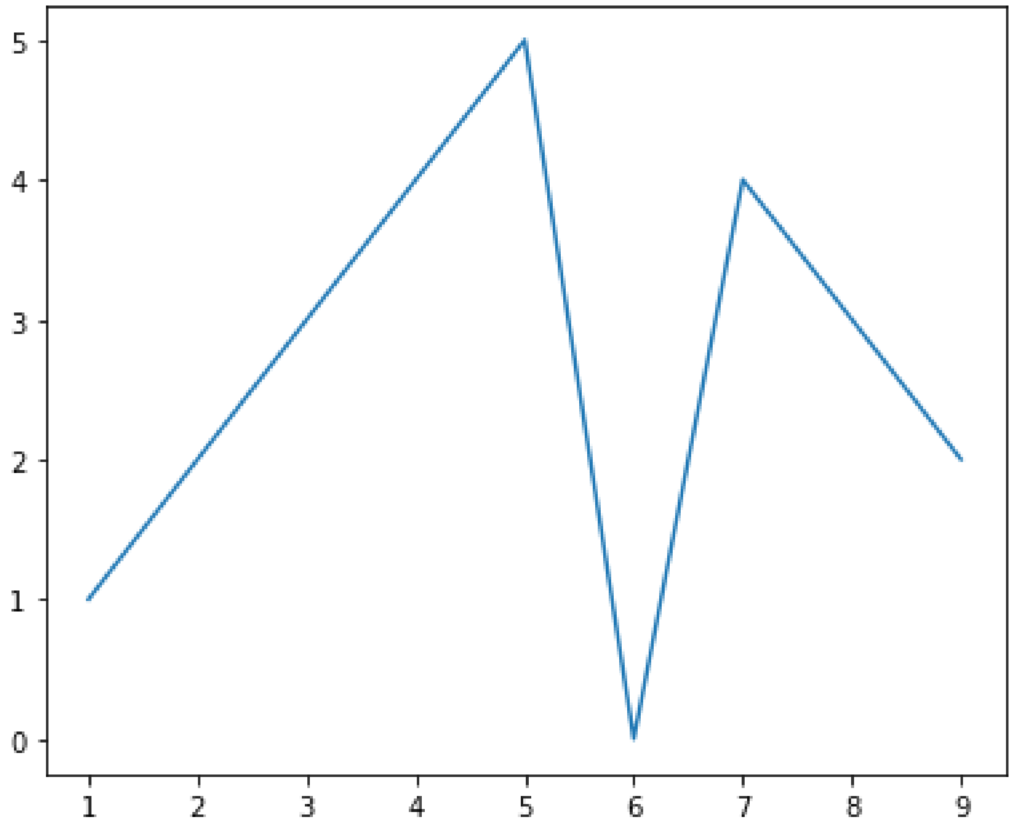
In [1]:
```
import numpy as np
from numpy.random import randn
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
from matplotlib import rcParams
```

In [2]:
```
%matplotlib inline
rcParams ['figure.figsize'] = 5,4
```

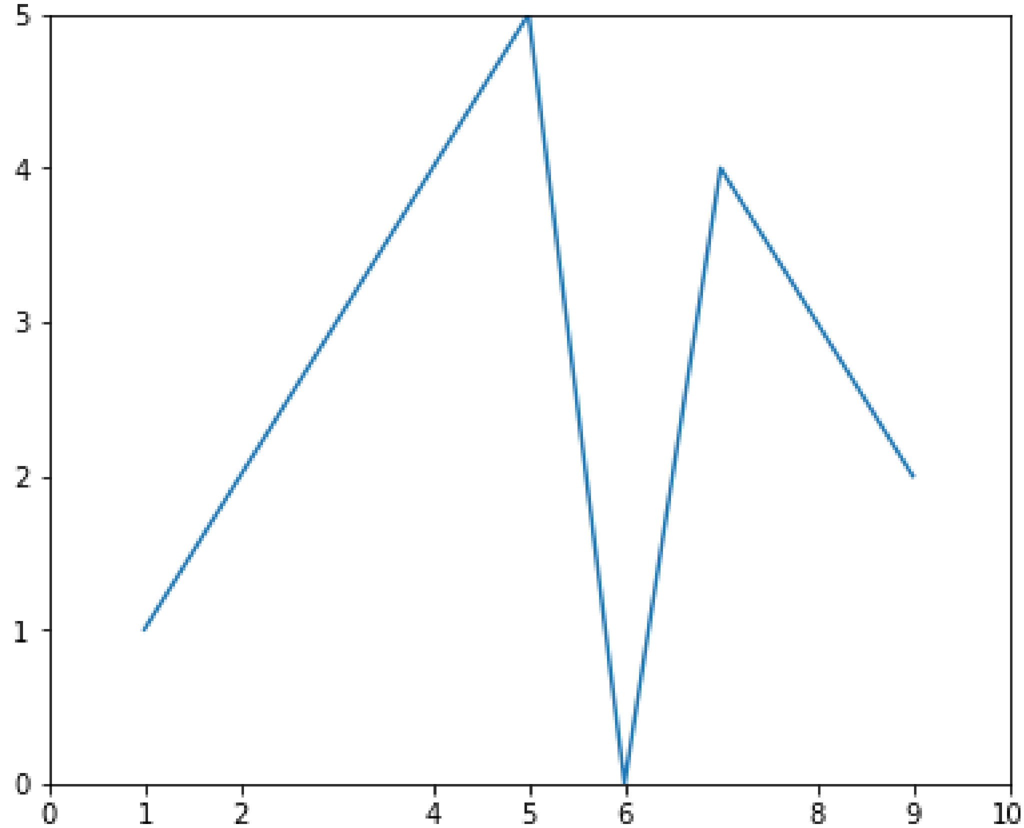
1.0.2 Defining Axes, Ticks, and Grids

In [3]:
```
x = range(1,10)
y = [1,2,3,4,5,0,4,3,2]
fig = plt.figure()
ax = fig.add_axes([.1, .1, 1, 1]) #add axes in plot
ax.plot(x,y)
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x7f21f10ab0f0>]
```

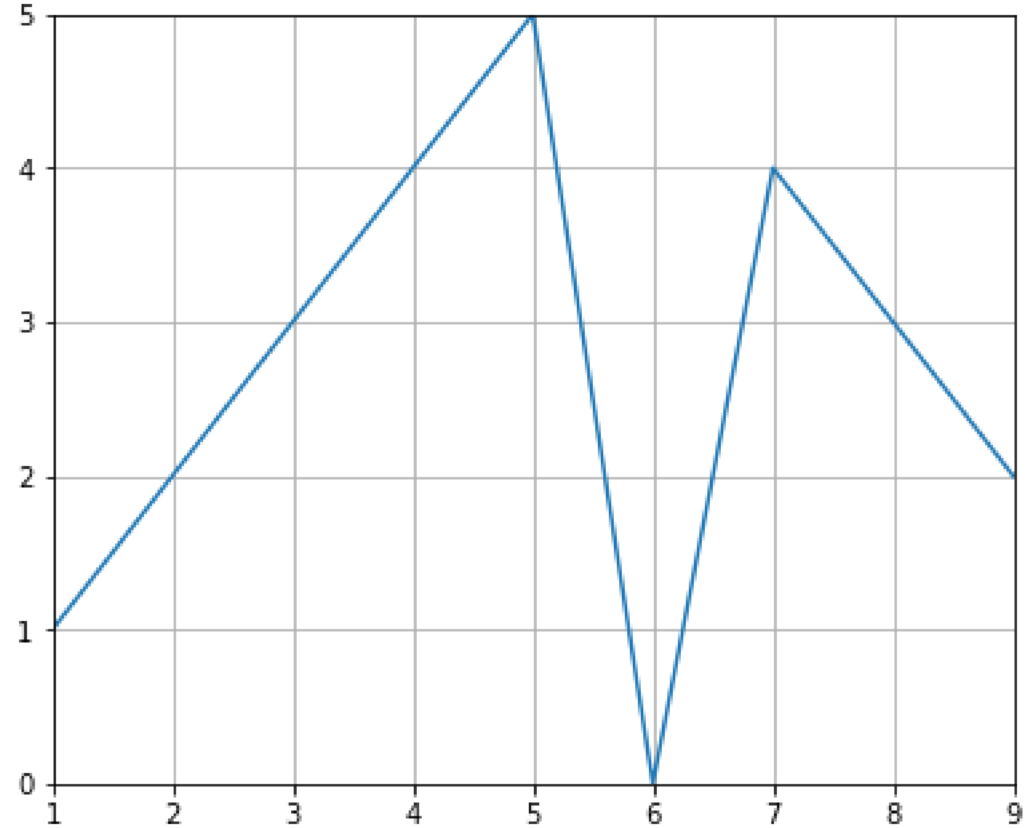



```
In [4]: ax.plot(x,y)

Out[4]: [<matplotlib.lines.Line2D at 0x7f2206fa4940>]

In [5]: fig = plt.figure()
        ax = fig.add_axes([.1, .1, 1, 1])
        ax.set_xlim([1,9]) #define x,y limit in plot
        ax.set_ylim([0,5])
        ax.set_xticks([0,1,2,4,5,6,8,9,10])
        ax.set_yticks([0,1,2,3,4,5])
        ax.plot(x,y)

Out[5]: [<matplotlib.lines.Line2D at 0x7f21e6648940>]
```



In [6]:
```
fig = plt.figure()
ax = fig.add_axes([.1, .1, 1, 1])
ax.set_xlim([1,9])
ax.set_ylim([0,5])
ax.grid() #make a grid in plot
ax.plot(x,y)
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x7f21f109d630>]
```

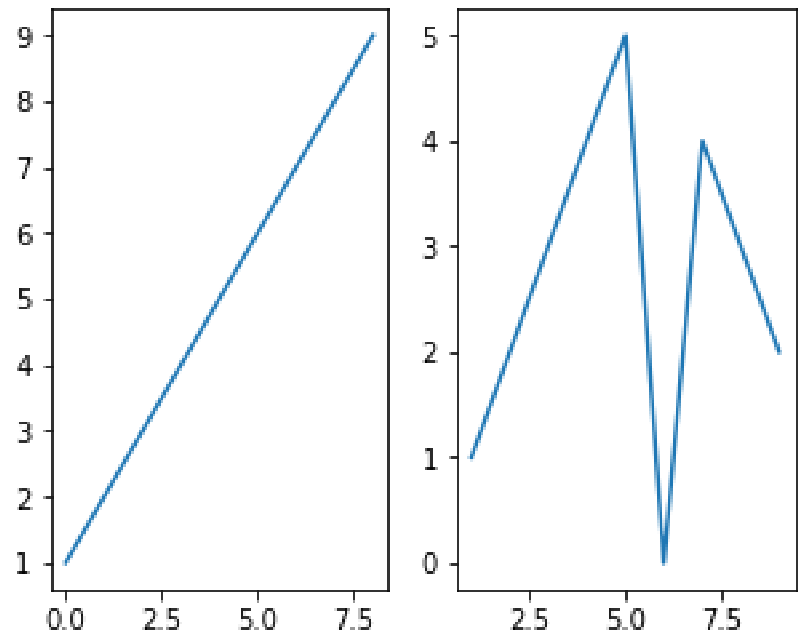


1.0.3 Generate multiplots in one figure with subplots

```
In [7]: fig = plt.figure()
        fig, (ax1, ax2) = plt.subplots(1,2)
        ax1.plot(x)
        ax2.plot(x,y)
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x7f21e6546400>]
```

```
<matplotlib.figure.Figure at 0x7f21e66e8978>
```



2 PLOT FORMATING

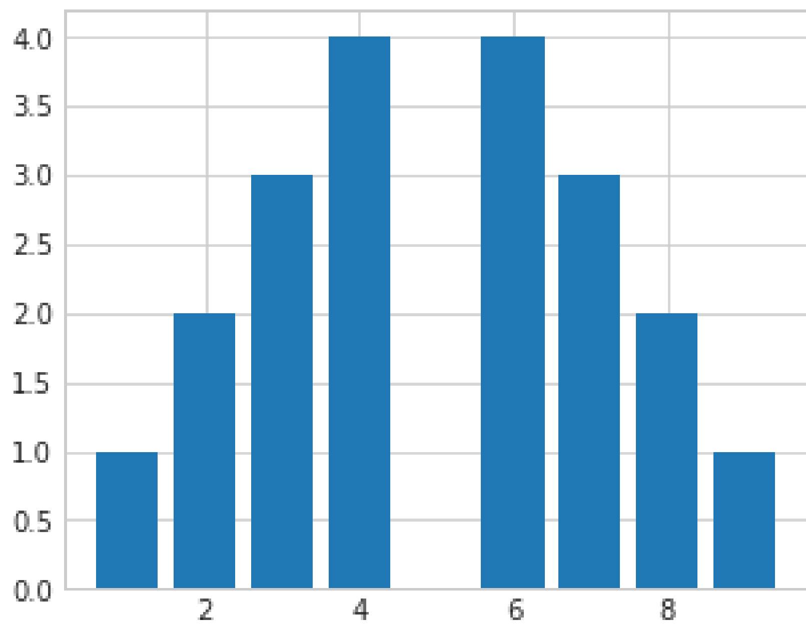
2.0.1 Plotformating

```
In [8]: import seaborn as sb
```

```
In [9]: %matplotlib inline
        rcParams['figure.figsize'] = 5,4
        sb.set_style('whitegrid')
```

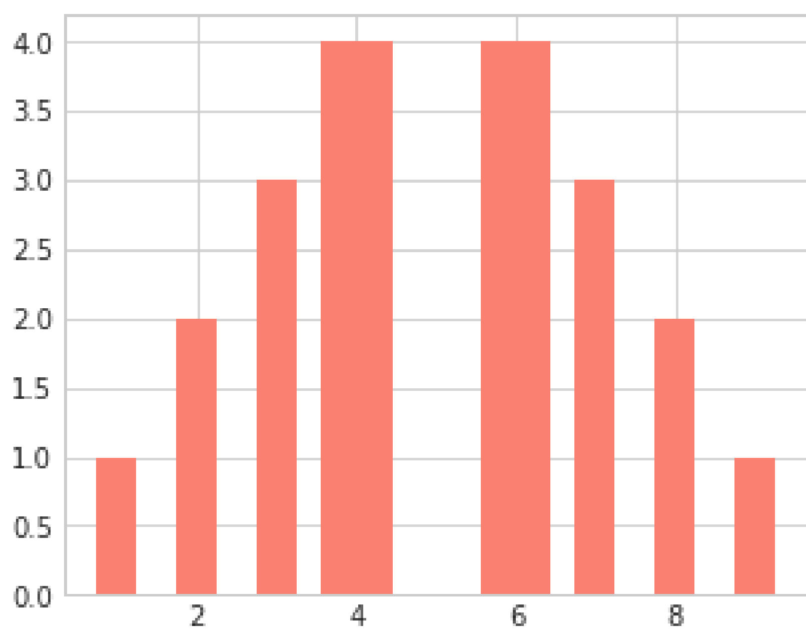
2.0.2 Defining plot color

```
In [10]: x = range(1,10)
         y = [1,2,3,4,0,4,3,2,1]
         plt.bar(x,y)

Out[10]: <Container object of 9 artists>
```

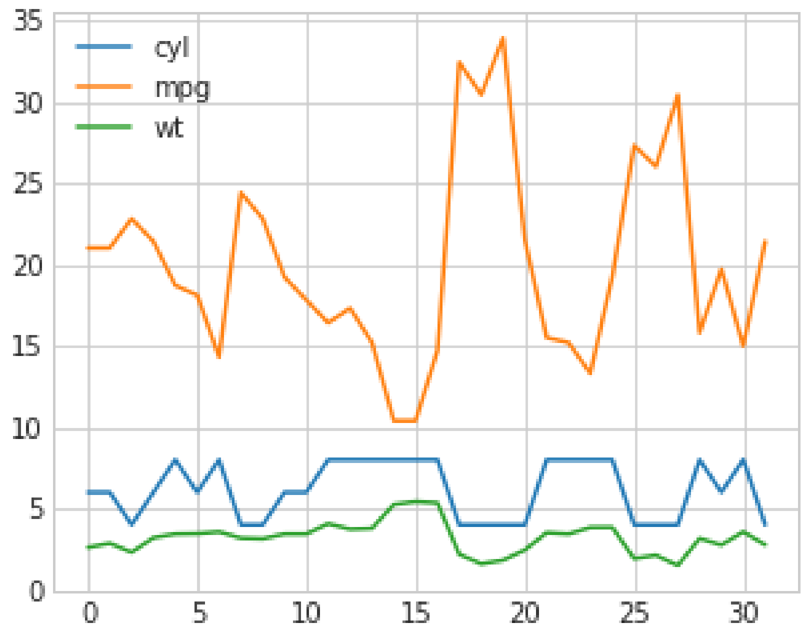


In [11]:
```
wide = [0.5, 0.5, 0.5, 0.9, 0.9, 0.9, 0.5, 0.5, 0.5]
color = ['salmon'] #costom color define
plt.bar(x,y, width = wide, color = color, align = 'center')
```

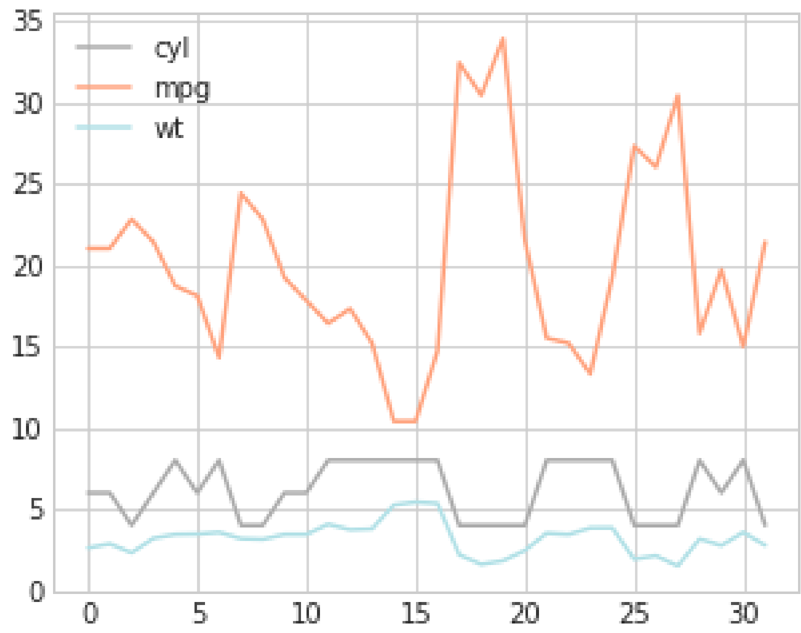
Out[11]:
```
<Container object of 9 artists>
```



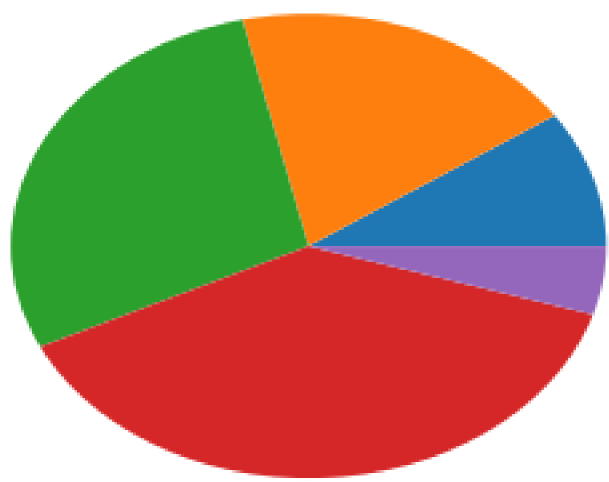
```
In [12]: address = 'mtcars.csv'
         cars = pd.read_csv(address)
         cars.columns = ['car_name', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
         df = cars[['cyl', 'mpg', 'wt']]
         df.plot()

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f21dce82940>
```



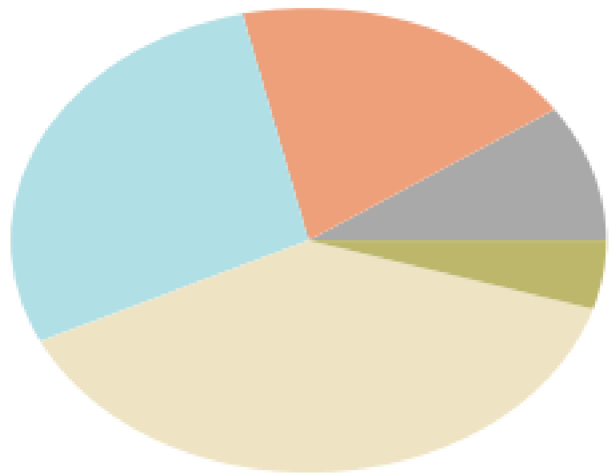

```
In [13]: color_theme =['darkgray', 'lightsalmon', 'powderblue'] #define color type
         df.plot(color = color_theme)

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f21dceb94a8>
```




```
In [14]: z = [1,2,3,4,0.5]
         plt.pie(z)
         plt.show()
```

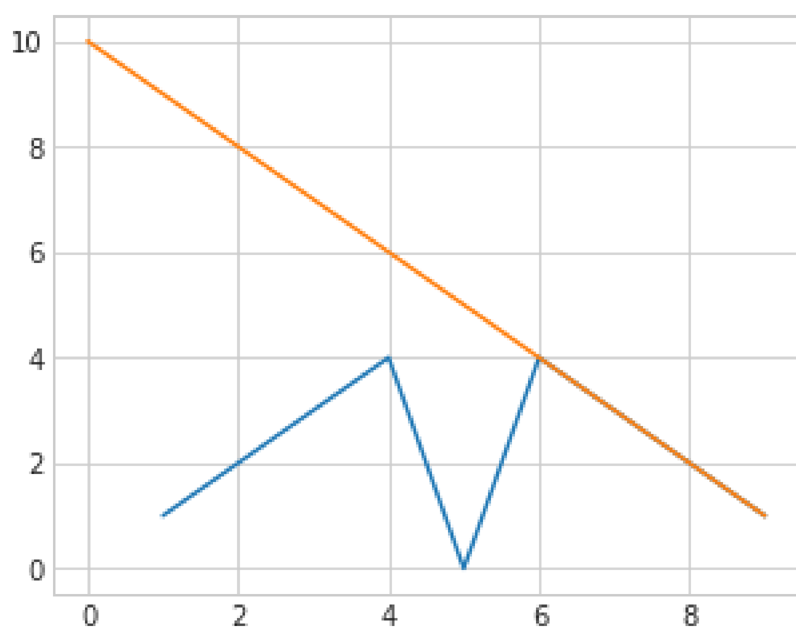


```
In [15]: color_theme = ['#A9A9A9', '#EEA07A', '#B0E0E6', '#EEE4C4', '#BDB76B']
         plt.pie(z, colors = color_theme)
         plt.show()
```



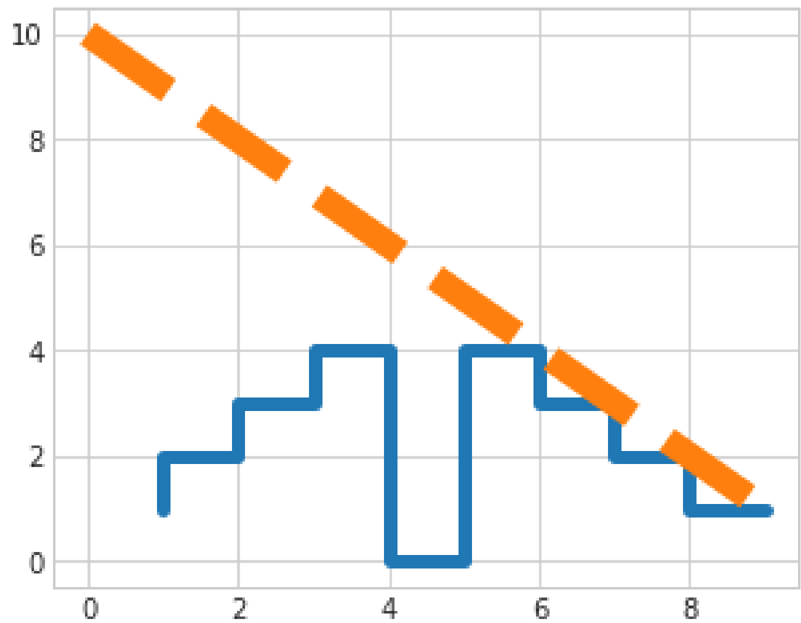
2.0.3 Costomizing line styles

```
In [16]: x1 = range(0,10)
         y1 = [10,9,8,7,6,5,4,3,2,1]

         plt.plot(x,y)
         plt.plot(x1,y1)

Out[16]: [<matplotlib.lines.Line2D at 0x7f21dcd0be80>]
```



```
In [17]: x1 = range(0,10)
         y1 = [10,9,8,7,6,5,4,3,2,1]

         plt.plot(x,y, ls = 'steps', lw = 5) #line size
         plt.plot(x1,y1, ls ='--', lw = 10)
```

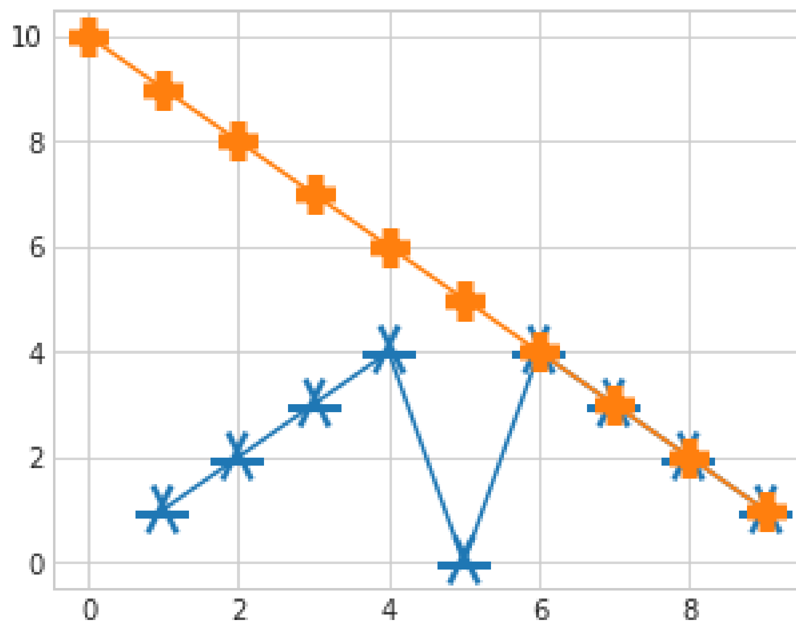
```
Out[17]: [<matplotlib.lines.Line2D at 0x7f21dcd49b38>]
```



2.0.4 Setting plot markers

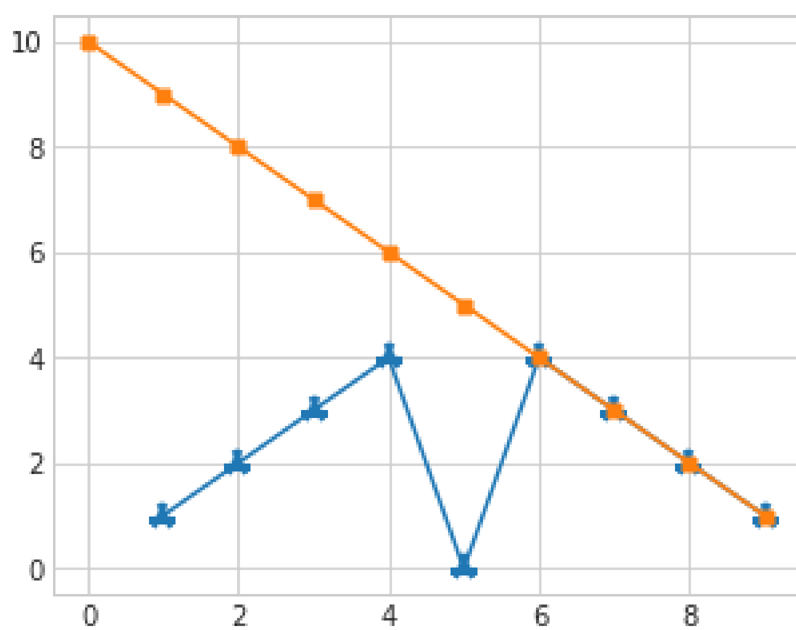
```
In [18]: x1 = range(0,10)
         y1 = [10,9,8,7,6,5,4,3,2,1]

         plt.plot(x,y, marker = '1', mew = 20) #define marker points in plot
         plt.plot(x1,y1, marker = '+', mew = 15)
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x7f21dcc5eb00>]
```




```
In [19]: x1 = range(0,10)
         y1 = [10,9,8,7,6,5,4,3,2,1]

         plt.plot(x,y, marker = '1', mew = 10)
         plt.plot(x1,y1, marker = '+', mew = 5)
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x7f21dcc7a4a8>]
```



The Engineering World #DataScience 9 & 10

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 CREATING LABELS AND ANNOTATIONS

1.0.1 Creating labels and annotations

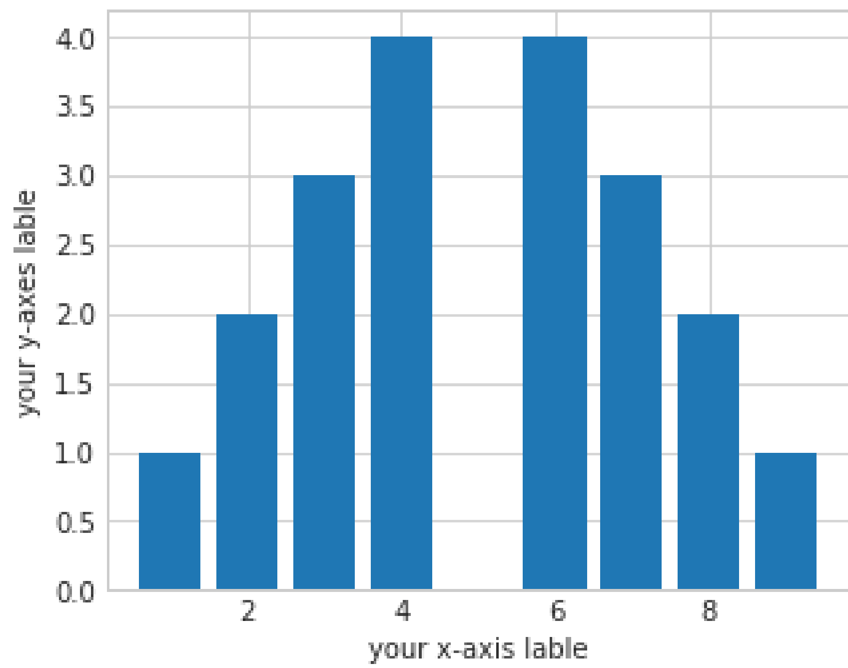
```
In [1]: import numpy as np
        from numpy.random import randn
        import pandas as pd
        from pandas import Series, DataFrame
        import matplotlib.pyplot as plt
        from matplotlib import rcParams
        import seaborn as sb
```

```
In [2]: %matplotlib inline
        rcParams ['figure.figsize'] = 5,4
        sb.set_style ('whitegrid')
```

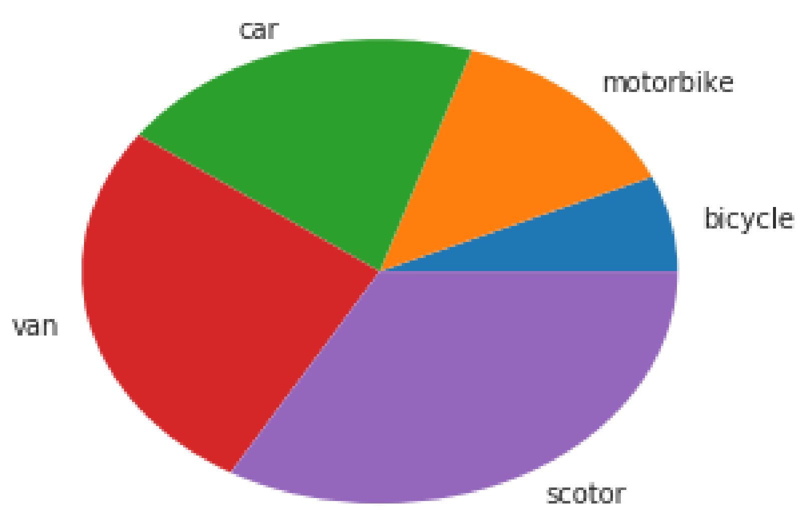
1.0.2 Labeling plot features

1.0.3 The functional method

```
In [3]: x = range(1,10)
        y = [1,2,3,4,0,4,3,2,1]

        plt.bar(x,y)
        plt.xlabel('your x-axis lable') #define axis lable in a graph side
        plt.ylabel('your y-axes lable')

Out[3]: Text(0,0.5,'your y-axes lable')
```



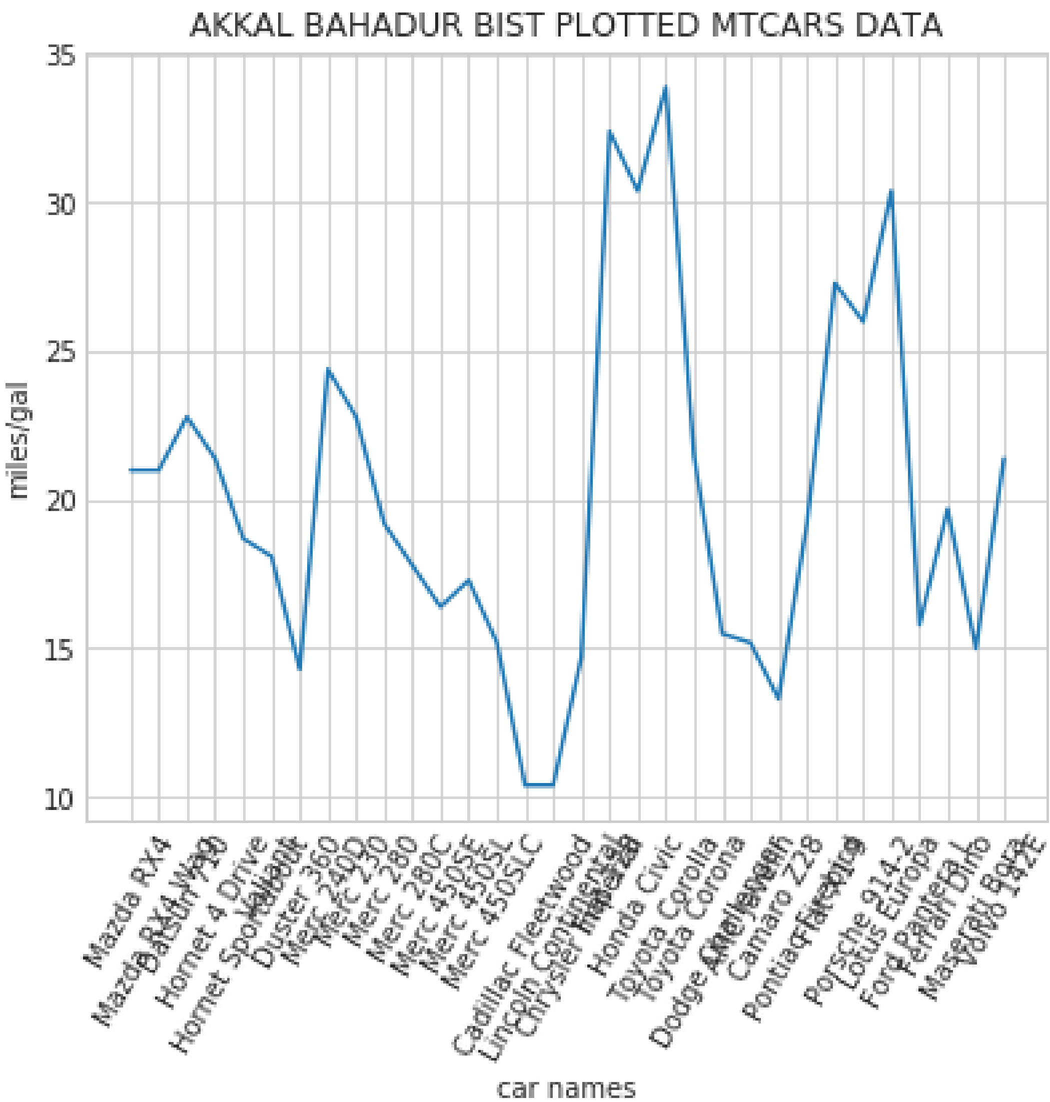

```
In [4]: z = [1,2,3,4,5]
        veh_type = ['bicycle', 'motorbike', 'car', 'van', 'scotor']
        plt.pie(z, labels = veh_type)
        plt.show()
```



1.0.4 The object-oriented method

In [5]:
```
address = 'mtcars.csv'
cars = pd.read_csv(address)
cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
mpg = cars.mpg

fig = plt.figure()

ax = fig.add_axes([.1,.1,1,1])
mpg.plot()

ax.set_xticks(range(32))
ax.set_xticklabels(cars.car_names, rotation = 60, fontsize = 'medium') #label name defin
ax.set_title('AKKAL BAHADUR BIST PLOTTED MTCARS DATA') #plot title name
ax.set_xlabel('car names') #xlabel name
ax.set_ylabel('miles/gal') #ylabel name
```

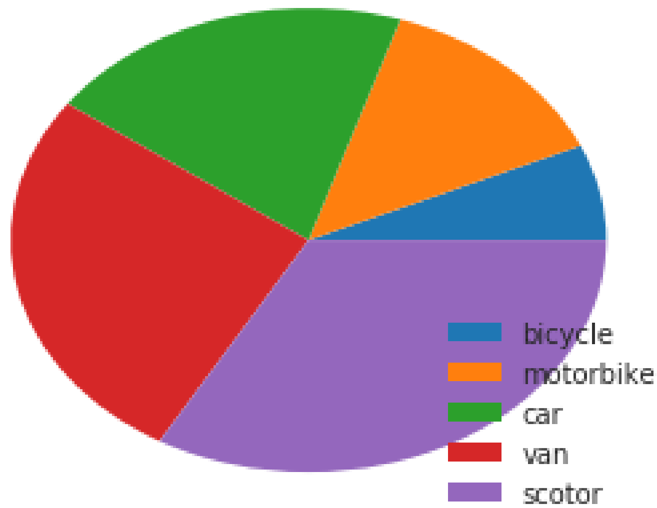
Out[5]:
```
Text(0,0.5,'miles/gal')
```



1.0.5 Adding a lagend to yur plot

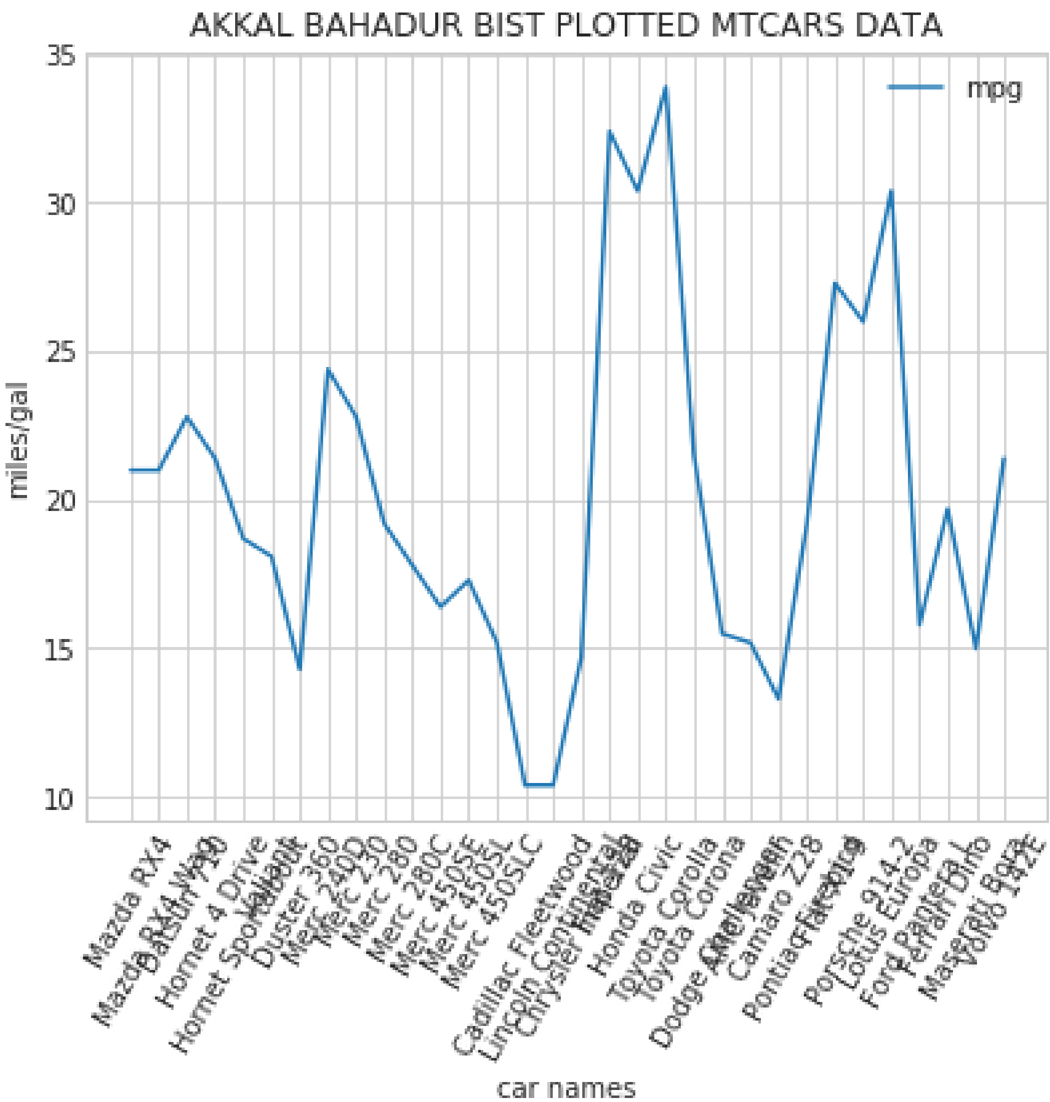
1.0.6 The functional method

```
In [6]: plt.pie(z)
        plt.legend(veh_type, loc = 'best') #lagend in pie plot
        plt.show()
```



1.0.7 The object-oriented method

```
In [7]: fig = plt.figure()
        ax = fig.add_axes([.1,.1,1,1])
        mpg.plot()

        ax.set_xticks(range(32))

        ax.set_xticklabels(cars.car_names, rotation = 60, fontsize = 'medium')
        ax.set_title('AKKAL BAHADUR BIST PLOTTED MTCARS DATA')
        ax.set_xlabel('car names')
        ax.set_ylabel('miles/gal')
        ax.legend (loc = 'best') #define legend in plot

Out[7]: <matplotlib.legend.Legend at 0x7f82142a2320>
```



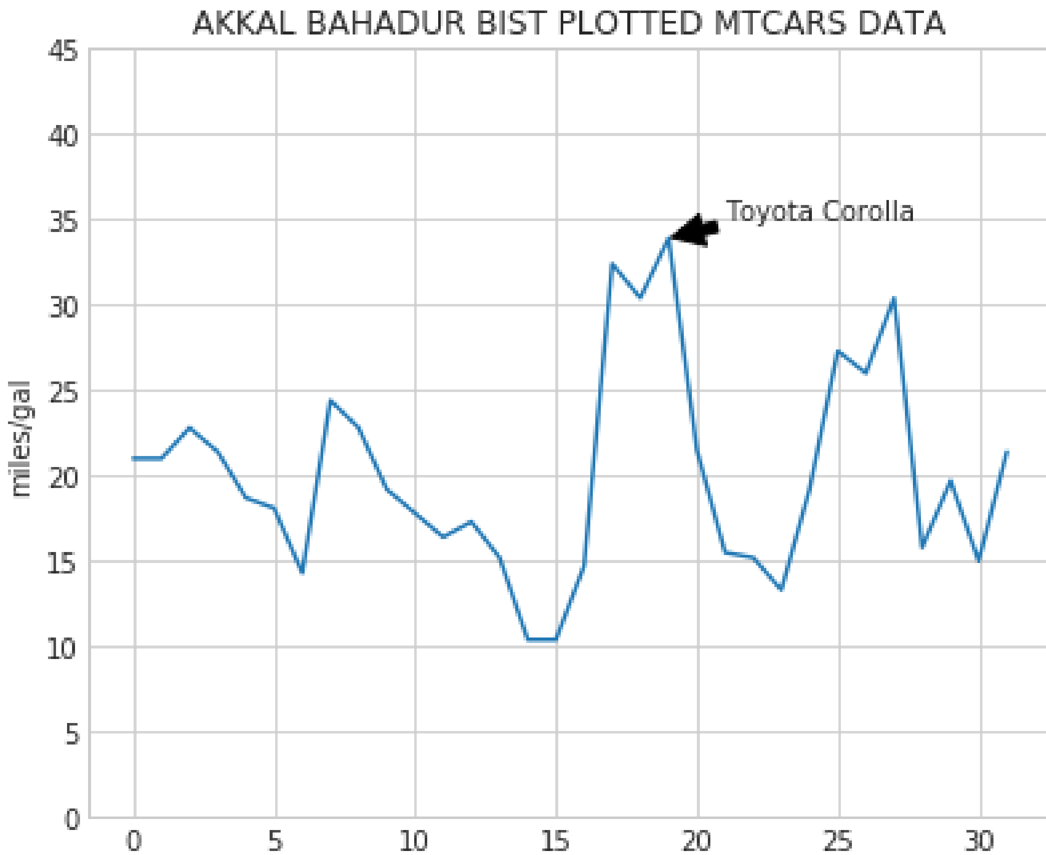
1.0.8 Annotating your plot

```
In [8]: mpg.max() #show data status in user required
```

```
Out[8]: 33.9
```

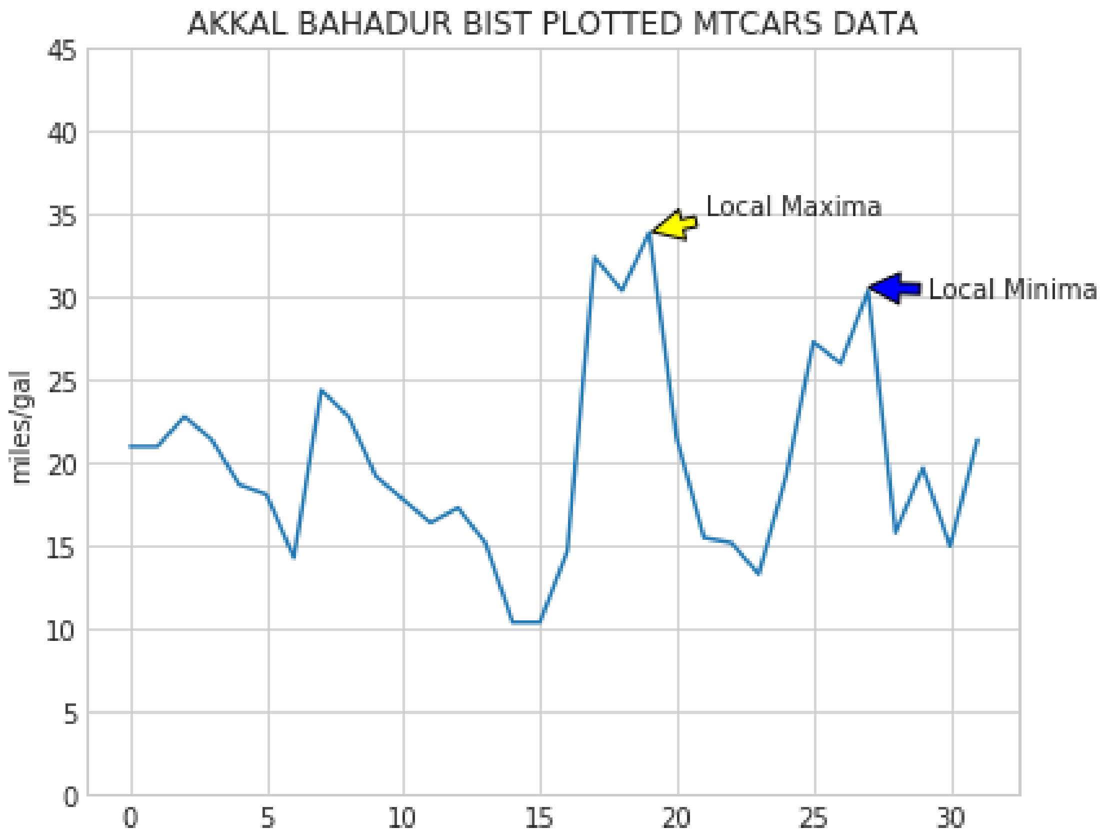
```
In [9]: fig = plt.figure()
        ax = fig.add_axes([.1,.1,1,1])
        mpg.plot()

        ax.set_title('AKKAL BAHADUR BIST PLOTTED MTCARS DATA')
        ax.set_ylabel('miles/gal')
        ax.set_ylim([0,45])
        ax.annotate('Toyota Corolla', xy = (19,33.9), xytext = (21,35), arrowprops = dict(faceco
```

Out[9]: Text(21,35,'Toyota Corolla')




```
In [10]: fig = plt.figure()
         ax = fig.add_axes([.1,.1,1,1])
         mpg.plot()

         ax.set_title('AKKAL BAHADUR BIST PLOTTED MTCARS DATA')
         ax.set_ylabel('miles/gal')
         ax.set_ylim([0,45])
         ax.annotate('Local Maxima', xy = (19,33.9), xytext = (21,35), arrowprops = dict(facecol
         ax.annotate('Local Minima', xy = (26.9,30.6), xytext = (29.2,30), arrowprops = dict(fac
```

Out[10]: Text(29.2,30,'Local Minima')



2 TIME SERIES DATA VISUALIZATION

2.0.1 Creating visualization from a time series data

2.0.2 The simplest time series plot

```
In [11]: address = 'DATA.csv'
         df = pd.read_csv(address, index_col = 'Year', parse_dates = True)
         df.head
```

```
Out[11]: <bound method NDFrame.head of                    Wildlife       WT    PE  NU   RED arrest_d
         Year
         2011-01-01        Tiger   15.000   0.0   0  EN           Parsa   Winter
         2011-01-01   Rhinoceros    1.000   0.0   0  VU         Syanjha   Winter
         2011-01-01      Leopard    0.000   1.0   0  VU        Lalitpur  Spring
         2011-01-01      Leopard    0.000   1.0   0  VU           Kavre  Spring
         2011-01-01        Tiger    8.800   1.0   0  EN       Makwanpur  Spring
         2011-01-01   Rhinoceros    0.000  12.0   0  VU       Kathmandu  Spring
         2011-01-01   Rhinoceros    1.000   0.0   0  VU        Lalitpur  Spring
         2011-01-01   Rhinoceros    1.000   0.0   0  VU       Kathmandu  Spring
         2011-01-01   Rhinoceros    1.000   0.0   0  VU       Kathmandu  Summer
```

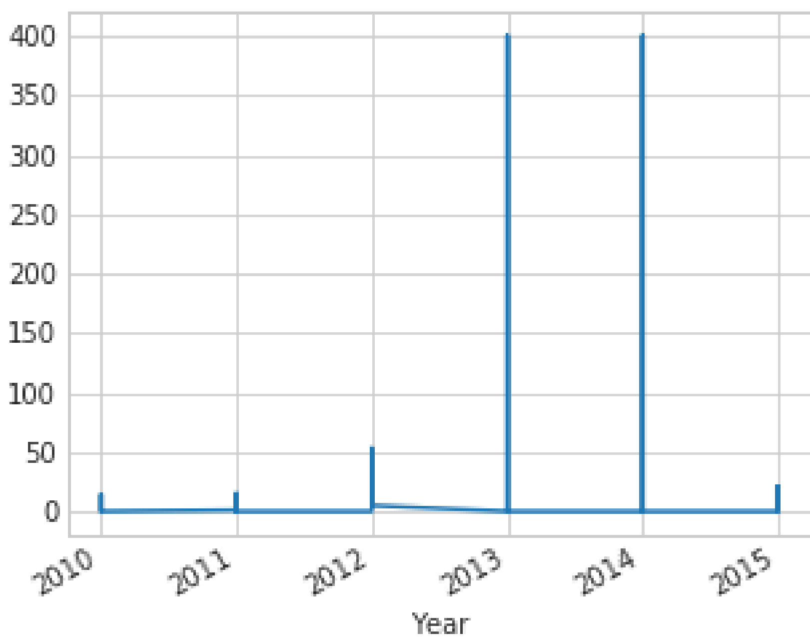
2011-01-01	Leopard	4.500	1.0	0	VU	Tanahun	Summer
2011-01-01	Red panda	0.000	2.0	0	EN	Kathmandu	Autumn
2012-01-01	Bear	0.073	0.0	0	VU	Kathmandu	Winter
2012-01-01	Leopard	0.000	2.0	0	VU	Kathmandu	Winter
2012-01-01	Leopard	0.000	1.0	0	VU	Kathmandu	Winter
2012-01-01	Bear	0.000	2.0	0	VU	Kathmandu	Winter
2012-01-01	Musk Deer	0.000	1.0	0	EN	Kathmandu	Spring
2012-01-01	Leopard	0.000	2.0	0	VU	Kavre	Spring
2012-01-01	Rhinoceros	1.000	0.0	0	VU	Kathmandu	Spring
2012-01-01	Rhinoceros	1.000	0.0	0	VU	Kaski	Spring
2012-01-01	elephant	0.000	2.0	0	EN	Kailali	Summer
2012-01-01	Bear	0.185	0.0	0	VU	Kathmandu	Summer
2012-01-01	Tiger	10.000	0.0	0	EN	Makwanpur	Summer
2012-01-01	Bear	2.000	0.0	0	VU	Kathmandu	Summer
2012-01-01	Leopard	0.000	2.0	0	VU	Lalitpur	Summer
2012-01-01	Tiger	0.000	1.0	0	EN	Kathmandu	Autumn
2012-01-01	Red panda	0.000	2.0	0	EN	Kathmandu	Autumn
2013-01-01	Leopard	0.000	1.0	0	VU	Sunsari	Winter
2013-01-01	Tiger	0.000	12.0	0	EN	Kathmandu	Spring
2013-01-01	Pangolin	0.000	1.0	0	EN	Kathmandu	Winter
2013-01-01	Red panda	0.000	1.0	0	EN	Nuwakot	Winter
...	...	...	...	..	...	...	...
2014-01-01	Elephant	21.000	0.0	0	EN	Lalitpur	FALSE
2013-01-01	Rhinoceros	0.000	1.0	0	VU	Mahottari	Spring
2015-01-01	Red Panda	0.000	2.0	0	EN	Bhaktapur	FALSE
2013-01-01	Pangolin	9.000	9.0	0	EN	Sindhupalchok	Spring
2013-01-01	Pangolin	80.000	80.0	0	EN	Sindhupalchok	Spring
2013-01-01	Tiger	1.240	0.0	0	EN	Bardiya	Spring
2013-01-01	Tiger	400.000	8.0	0	EN	Nuwakot	Spring
2013-01-01	Pangolin	40.000	40.0	0	EN	Darchula	Spring
2013-01-01	Red Panda	0.000	3.0	0	EN	Makwanpur	Summer
2013-01-01	Sambar deer	1.000	0.0	0	VU	Sindhupalchok	Summer
2013-01-01	Pangolin	48.000	48.0	0	EN	Bhaktapur	Summer
2013-01-01	Tiger	0.000	3.0	0	EN	Dang	Summer
2013-01-01	Leopard	0.000	3.0	0	VU	Dang	Summer
2013-01-01	Pangolin	40.000	40.0	0	EN	Dang	Summer
2013-01-01	Spotted deer	3.000	0.0	0	LC	Banke	Summer
2013-01-01	Python	0.000	1.0	0	VU	Kathmandu	Summer
2010-01-01	Pangolin	14.000	14.0	0	EN	Kathmandu	FALSE
2015-01-01	Leopard	2.445	1.0	0	VU	Surkhet	Spring
2010-01-01	Pangolin	10.000	10.0	0	EN	Sindhupalchok	FALSE
2011-01-01	Bear	0.000	5.0	0	VU	Sankhuwasabha	Summer
2011-01-01	Red Panda	0.000	2.0	0	EN	Kathmandu	Summer
2012-01-01	Bear	0.000	5.0	0	VU	Sankhuwasabha	Autumn
2011-01-01	Tiger	0.000	1.0	0	EN	Kathmandu	Spring
2011-01-01	Leopard	0.000	2.0	0	VU	Dolakha	Summer
2011-01-01	Tiger	7.500	0.0	0	EN	Bara	Autumn
2012-01-01	Rhinoceros	1.000	0.0	0	VU	Kathmandu	Winter

```
2012-01-01      Pangolin    4.000   4.0    0    EN            Ilam  Spring
2012-01-01          Bear    0.182   0.0    0    VU       Kathmandu  Summer
2011-01-01     Red Panda    0.000   1.0    0    EN       Kathmandu  Summer
2012-01-01    Rhinoceros    0.000   3.0    0    VU         Chitwan  Summer

             Volume
Year
2011-01-01   15.000
2011-01-01    1.000
2011-01-01    1.000
2011-01-01    1.000
2011-01-01    9.800
2011-01-01   12.000
2011-01-01    1.000
2011-01-01    1.000
2011-01-01    1.000
2011-01-01    5.500
2011-01-01    2.000
2012-01-01    0.073
2012-01-01    2.000
2012-01-01    1.000
2012-01-01    2.000
2012-01-01    1.000
2012-01-01    2.000
2012-01-01    1.000
2012-01-01    1.000
2012-01-01    2.000
2012-01-01    0.185
2012-01-01   10.000
2012-01-01    2.000
2012-01-01    2.000
2012-01-01    1.000
2012-01-01    2.000
2013-01-01    1.000
2013-01-01   12.000
2013-01-01    1.000
2013-01-01    1.000
...             ...
2014-01-01   21.000
2013-01-01    1.000
2015-01-01    2.000
2013-01-01   18.000
2013-01-01  160.000
2013-01-01    1.240
2013-01-01  408.000
2013-01-01   80.000
2013-01-01    3.000
2013-01-01    1.000
```

```
2013-01-01   96.000
2013-01-01    3.000
2013-01-01    3.000
2013-01-01   80.000
2013-01-01    3.000
2013-01-01    1.000
2010-01-01   28.000
2015-01-01    3.445
2010-01-01   20.000
2011-01-01    5.000
2011-01-01    2.000
2012-01-01    5.000
2011-01-01    1.000
2011-01-01    2.000
2011-01-01    7.500
2012-01-01    1.000
2012-01-01    8.000
2012-01-01    0.182
2011-01-01    1.000
2012-01-01    3.000

[444 rows x 8 columns]>
```

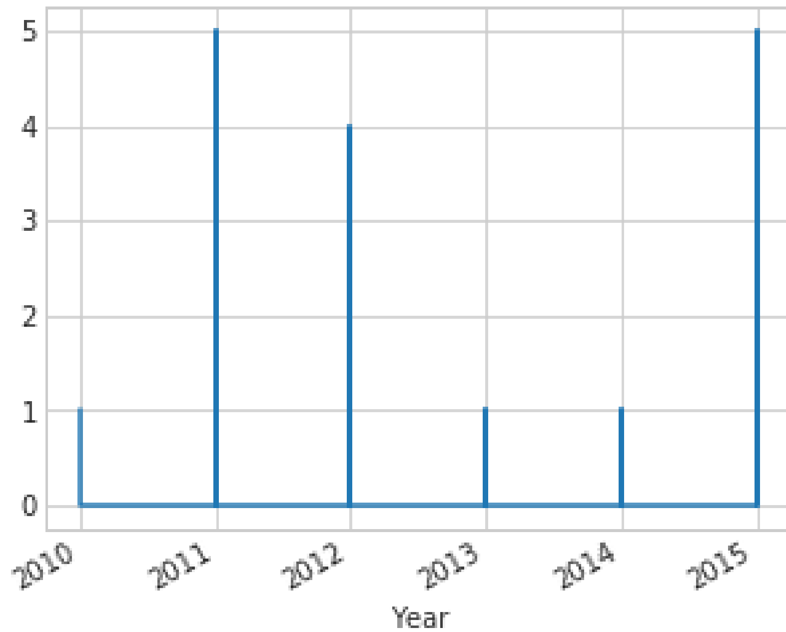
```
In [12]: df['WT'].plot()
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8214bb3b70>
```



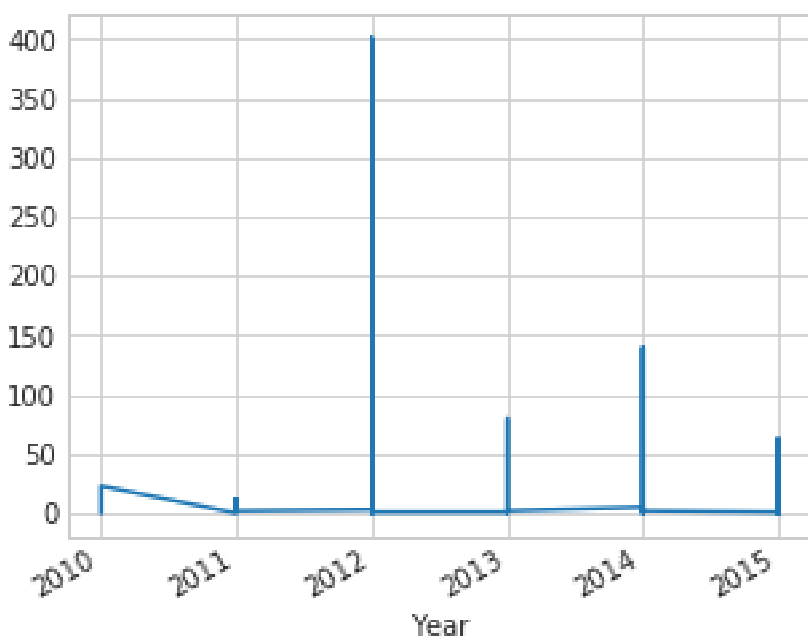
In [13]: df['NU'].plot()

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8214baca20>



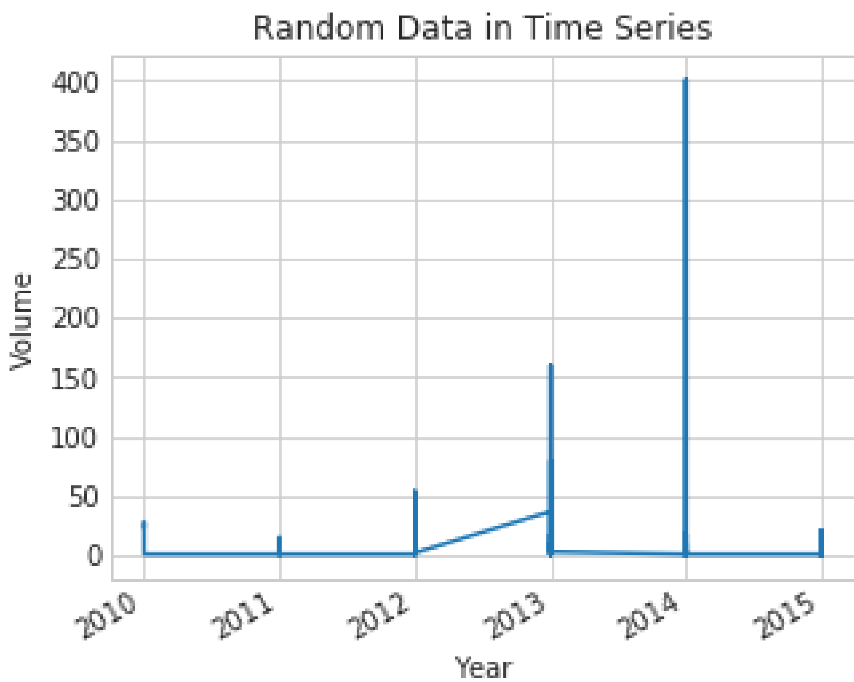
In [14]: df['PE'].plot()

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f821410f4e0>



```
In [15]: df2 = df.sample(n=100, random_state=25,axis=0)
         plt.xlabel('Year')
         plt.ylabel('Volume')
         plt.title('Random Data in Time Series')
         df2['Volume'].plot()
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8214b6eac8>
```




```
In [ ]:
```

The Engineering World #DataScience 11

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 HISTOGRAMS, BOX PLOTS AND SCATTER PLOTS

1.0.1 Constructing Histogram, Box plots and Scatter plots

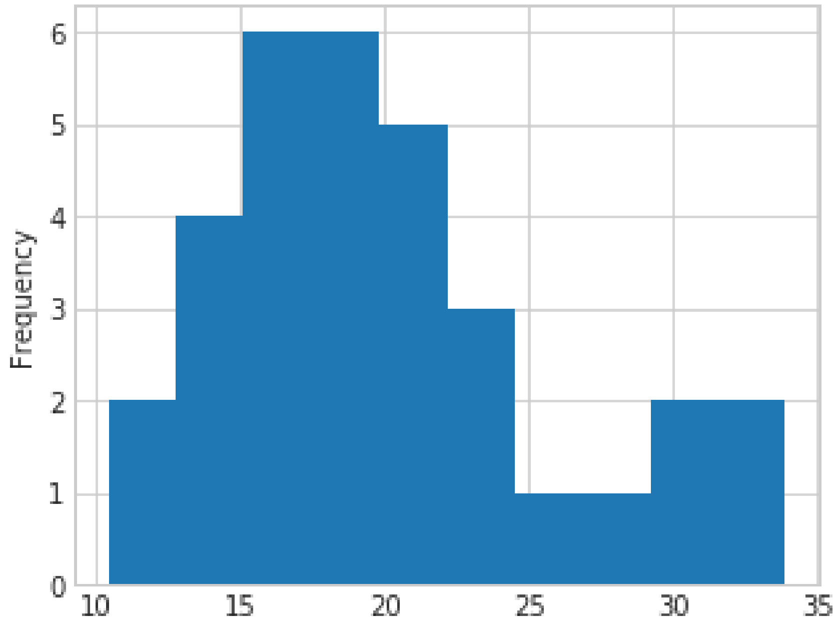
```
In [1]: import numpy as np
        from numpy.random import randn

        import pandas as pd
        from pandas import Series, DataFrame

        import matplotlib.pyplot as plt
        from matplotlib import rcParams

        import seaborn as sb
```

```
In [2]: %matplotlib inline
        rcParams ['figure.figsize'] = 5,4
        sb.set_style ('whitegrid')
```

1.0.2 Eyballing dataset distributions with histograms

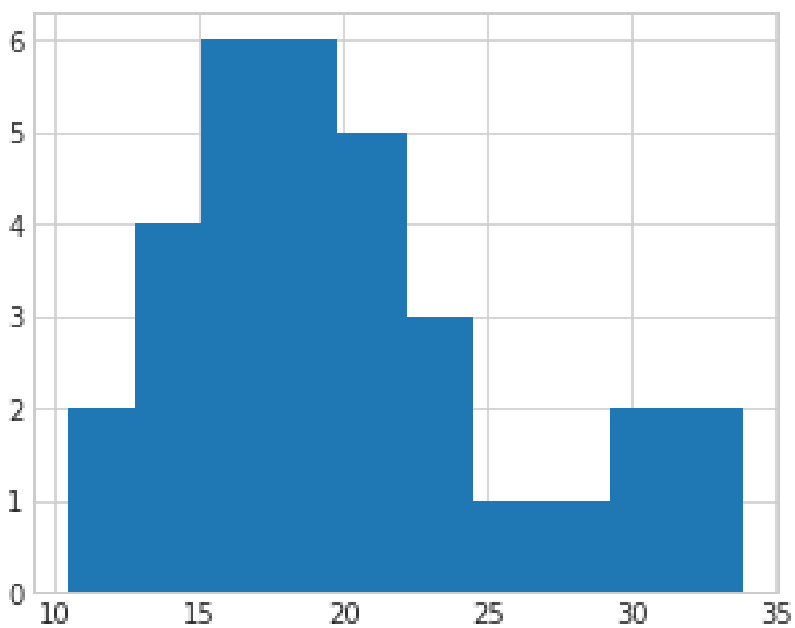
```
In [3]: address = 'mtcars.csv'
        cars = pd.read_csv(address)
        cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
        cars.index = cars.car_names
        mpg = cars['mpg']
        mpg.plot(kind = 'hist')
```

```
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc6fae9bfd0>
```



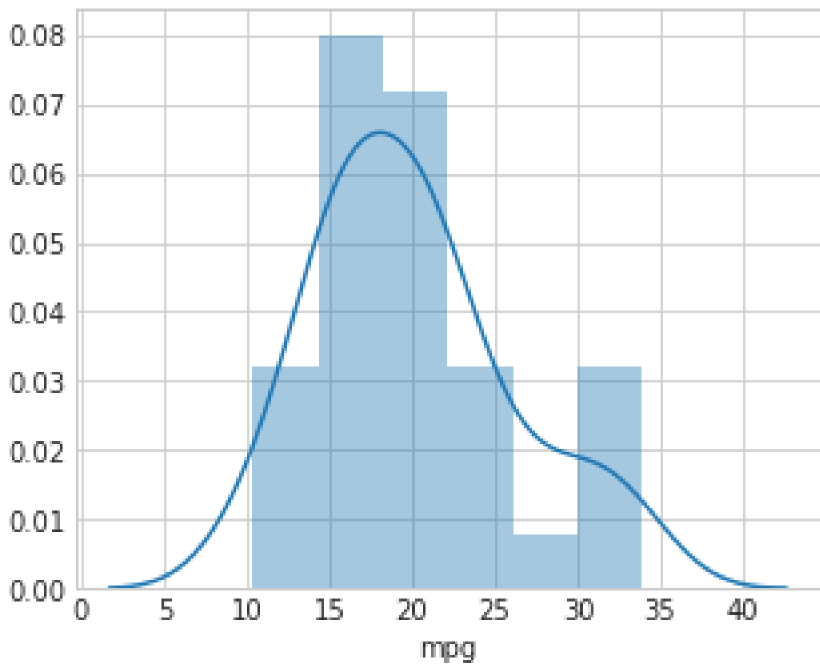

```
In [4]: plt.hist(mpg)
        plt.plot()
```

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



```
In [5]: sb.distplot(mpg) #shows the distribution line
```

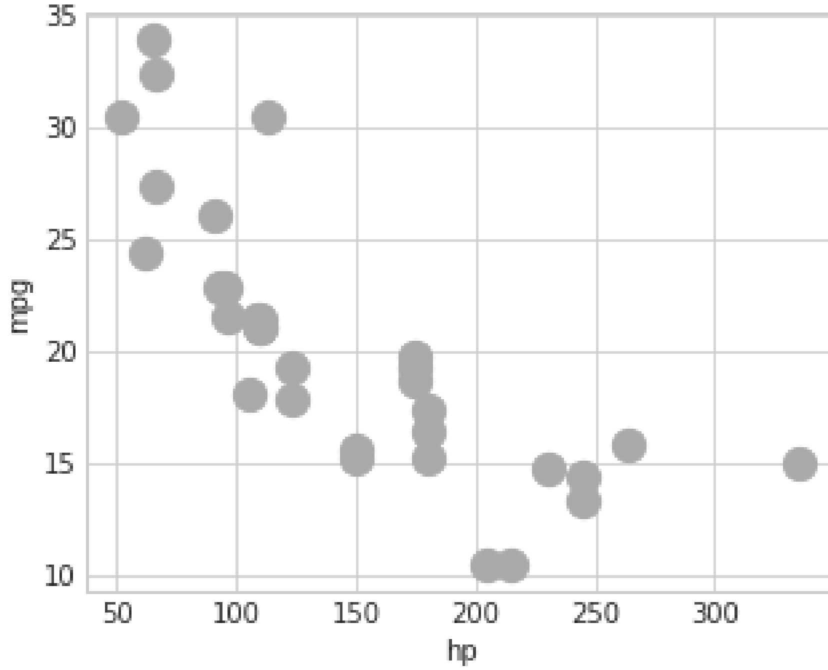
```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc6f0459470>
```



1.0.3 Seeing scatterplots in action

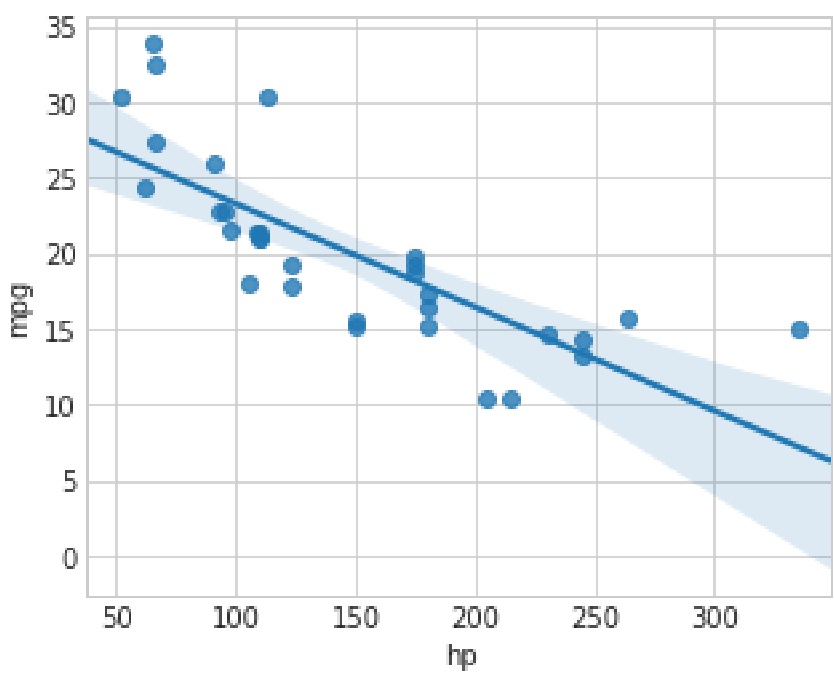
```
In [6]: cars.plot(kind='scatter', x = 'hp', y = 'mpg', c = ['darkgray'], s=150)
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc6f03b5c50>
```




```
In [7]: sb.regplot(x = 'hp', y = 'mpg', data = cars, scatter = True) #draw line in scatter point
```

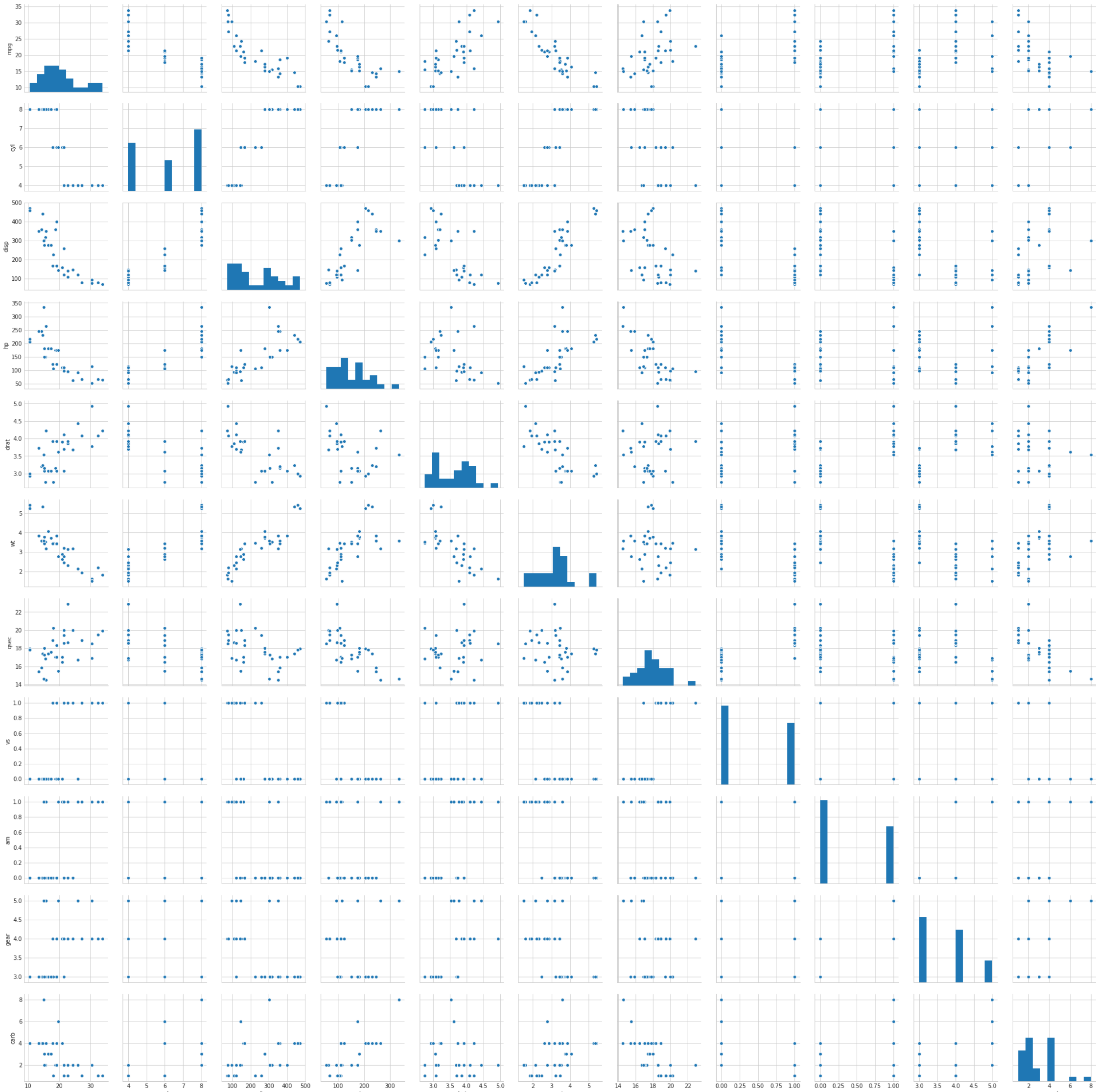
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc6f0364cf8>



1.0.4 Generating a scatter plot matrix

```
In [8]: sb.pairplot(cars)
```

```
Out[8]: <seaborn.axisgrid.PairGrid at 0x7fc6faf3e550>
```



```
In [9]: cars_df = pd.DataFrame((cars.ix[:,(1,3,4,6)].values), columns = ['mpg', 'disp', 'hp', 'w
        cars_target = cars.ix[:,9].values
        target_names = [0,1]

        cars_df['group'] = pd.Series(cars_target, dtype = "category")
        sb.pairplot(cars_df, hue = 'group', palette = 'hls')
```

```
/home/akkal/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning:
.ix is deprecated. Please use
```

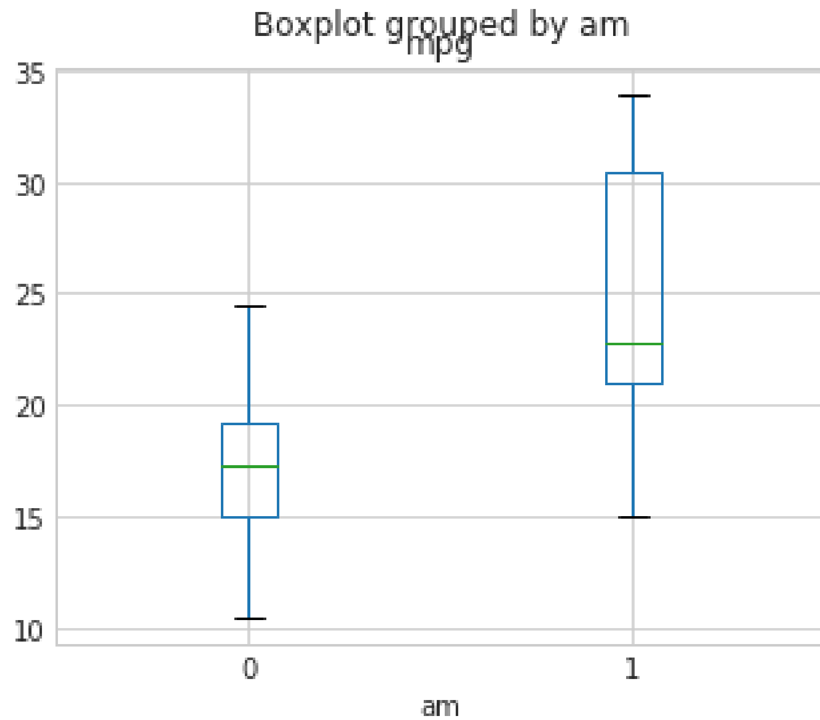
```
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  """Entry point for launching an IPython kernel.
```

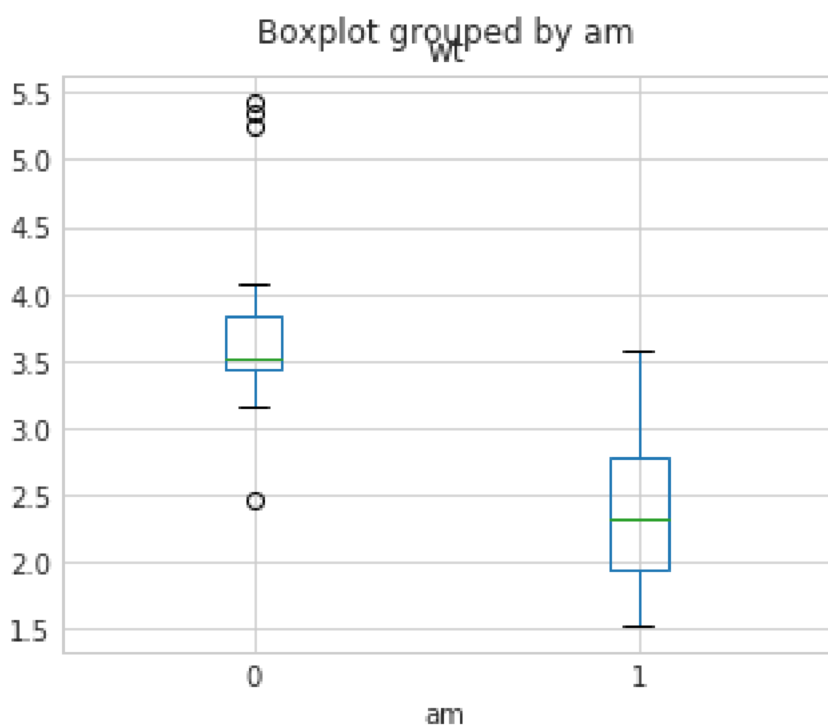
```
Out[9]: <seaborn.axisgrid.PairGrid at 0x7fc6ecd2fba8>
```

1.0.5 Building Box Plots

```
In [10]: cars.boxplot(column = 'mpg', by = 'am')
         cars.boxplot(column = 'wt', by = 'am')
```

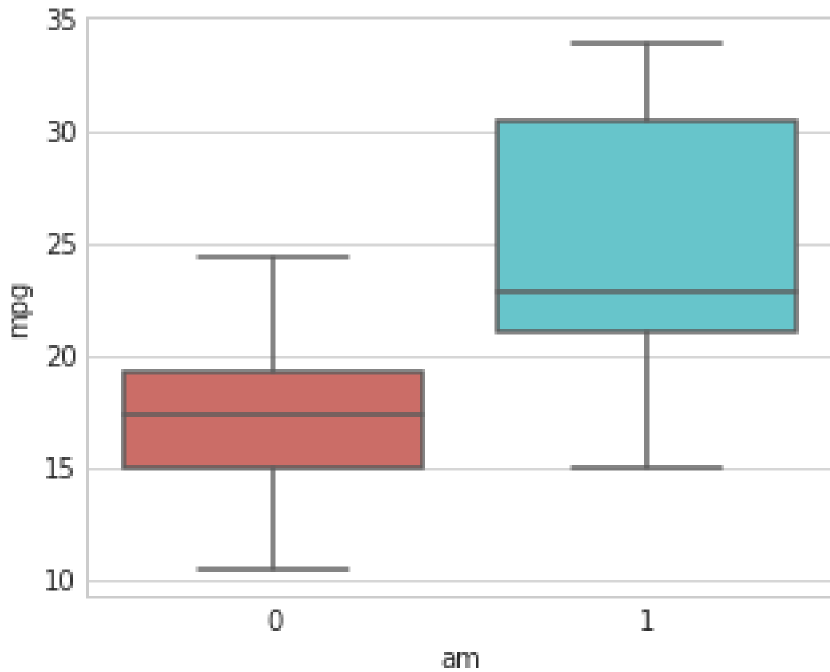
```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc6e48ee668>
```


Boxplot grouped by am
mpg
35
30
25
20
15
10
0
1
am


Boxplot grouped by am
wt
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0
1
am

```
In [11]: sb.boxplot(x = 'am', y = 'mpg', data = cars, palette = 'hls')
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc6e52a0978>
```



The Engineering World #DataScience 12 & 13

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 USING NUMPY TO PERFORM ARITHMETIC OPERATION ON DATA

```
In [1]: import numpy as np
        from numpy.random import randn

In [2]: np.set_printoptions(precision = 2)
```

1.0.1 Creating Arrays

1.0.2 Creating array using a list

```
In [3]: a = np.array([1,2,3,4,5,6])
        a

Out[3]: array([1, 2, 3, 4, 5, 6])

In [4]: b = np.array([[10,20,30,40,50], [60,70,80,90,100]])
        b

Out[4]: array([[ 10,  20,  30,  40,  50],
               [ 60,  70,  80,  90, 100]])
```

1.0.3 Creating arrays vis assignment

```
In [5]: np.random.seed(25)
        c = 35*np.random.randn(6)
        c

Out[5]: array([  7.99,  35.94, -29.39, -20.69, -33.49,  -7.78])

In [6]: d = np.arange(1,35)
        d

Out[6]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
               18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34])
```

1.0.4 Performing arithmetic on array

```
In [7]: a*10

Out[7]: array([10, 20, 30, 40, 50, 60])

In [8]: c+a

Out[8]: array([  8.99,  37.94, -26.39, -16.69, -28.49,  -1.78])

In [9]: c-a

Out[9]: array([  6.99,  33.94, -32.39, -24.69, -38.49, -13.78])

In [10]: c*a

Out[10]: array([   7.99,   71.88,  -88.16,  -82.77, -167.46,  -46.69])

In [11]: c/a

Out[11]: array([ 7.99, 17.97, -9.8 , -5.17, -6.7 , -1.3 ])
```

1.0.5 Multiplying matrices and basic algebra

```
In [12]: aa = np.array([[1.,2.,3.,4.,5.], [10.,20.,30.,40.,50.], [100.,200.,300.,400.,500.]])
         aa

Out[12]: array([[  1.,   2.,   3.,   4.,   5.],
                [ 10.,  20.,  30.,  40.,  50.],
                [100., 200., 300., 400., 500.]])

In [13]: bb = np.array([[0.,1.,2.,3.,4.], [00.,11.,22.,33.,44.], [100.,200.,300.,400.,500.]])
         bb

Out[13]: array([[  0.,   1.,   2.,   3.,   4.],
                [  0.,  11.,  22.,  33.,  44.],
                [100., 200., 300., 400., 500.]])

In [14]: aa *bb

Out[14]: array([[0.00e+00, 2.00e+00, 6.00e+00, 1.20e+01, 2.00e+01],
                [0.00e+00, 2.20e+02, 6.60e+02, 1.32e+03, 2.20e+03],
                [1.00e+04, 4.00e+04, 9.00e+04, 1.60e+05, 2.50e+05]])

In [15]: a1 = np.array([[1,2,3], [4,5,6], [7,8,9]])
         a1

Out[15]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

```
In [16]: b1 = np.array([[10,20,30],[40,50,60],[70,80,90]])
         b1
```

```
Out[16]: array([[10, 20, 30],
                [40, 50, 60],
                [70, 80, 90]])
```

```
In [17]: np.dot(a1,b1)
```

```
Out[17]: array([[ 300,  360,  420],
                [ 660,  810,  960],
                [1020, 1260, 1500]])
```

2 DESCRIPTIVE STATISTICS

2.0.1 Generating summary statistics using pandas and scipy

```
In [18]: import numpy as np
         import pandas as pd
         from pandas import Series, DataFrame

         import scipy
         from scipy import stats
```

```
In [19]: address = 'mtcars.csv'
         cars = pd.read_csv(address)
         cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'a
         cars.head()
```

```
Out[19]:             car_names   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
         0           Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
         1       Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
         2          Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
         3      Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
         4   Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

            carb
         0     4
         1     4
         2     1
         3     1
         4     2
```

2.0.2 Looking at summary statistics that describe a variable's numeric values

```
In [20]: cars.sum()
```

```
Out[20]: car_names    Mazda RX4Mazda RX4 WagDatsun 710Hornet 4 Drive...
         mpg                                                      642.9
```

```
cyl                                                               198
disp                                                           7383.1
hp                                                               4694
drat                                                           115.09
wt                                                            102.952
qsec                                                           571.16
vs                                                                 14
am                                                                 13
gear                                                              118
carb                                                               90
dtype: object
```

In [21]:
```
cars.sum(axis=1)
```

Out[21]:
```
0     328.980
1     329.795
2     259.580
3     426.135
4     590.310
5     385.540
6     656.920
7     270.980
8     299.570
9     350.460
10    349.660
11    510.740
12    511.500
13    509.850
14    728.560
15    726.644
16    725.695
17    213.850
18    195.165
19    206.955
20    273.775
21    519.650
22    506.085
23    646.280
24    631.175
25    208.215
26    272.570
27    273.683
28    670.690
29    379.590
30    694.710
31    288.890
dtype: float64
```

In [22]:
```
cars.median()
```

```
Out[22]: mpg      19.200
         cyl       6.000
         disp    196.300
         hp      123.000
         drat      3.695
         wt        3.325
         qsec     17.710
         vs        0.000
         am        0.000
         gear      4.000
         carb      2.000
         dtype: float64
```

```
In [23]: cars.mean()
```

```
Out[23]: mpg      20.090625
         cyl       6.187500
         disp    230.721875
         hp      146.687500
         drat      3.596563
         wt        3.217250
         qsec     17.848750
         vs        0.437500
         am        0.406250
         gear      3.687500
         carb      2.812500
         dtype: float64
```

```
In [24]: cars.max()
```

```
Out[24]: car_names    Volvo 142E
         mpg                33.9
         cyl                   8
         disp                472
         hp                  335
         drat               4.93
         wt                5.424
         qsec               22.9
         vs                    1
         am                    1
         gear                  5
         carb                  8
         dtype: object
```

```
In [25]: mpg = cars.mpg
         mpg.idxmax()
```

```
Out[25]: 19
```

2.0.3 Looking at summary statistics that describe variable distribution

In [26]:
```
cars.std()
```

Out[26]:
```
mpg       6.026948
cyl       1.785922
disp    123.938694
hp       68.562868
drat      0.534679
wt        0.978457
qsec      1.786943
vs        0.504016
am        0.498991
gear      0.737804
carb      1.615200
dtype: float64
```

In [27]:
```
cars.var()
```

Out[27]:
```
mpg        36.324103
cyl         3.189516
disp    15360.799829
hp       4700.866935
drat        0.285881
wt          0.957379
qsec        3.193166
vs          0.254032
am          0.248992
gear        0.544355
carb        2.608871
dtype: float64
```

In [28]:
```
gear = cars.gear
gear.value_counts()
```

Out[28]:
```
3    15
4    12
5     5
Name: gear, dtype: int64
```

In [29]:
```
cars.describe()
```

Out[29]:
```
             mpg        cyl        disp          hp       drat         wt  \
count  32.000000  32.000000   32.000000   32.000000  32.000000  32.000000
mean   20.090625   6.187500  230.721875  146.687500   3.596563   3.217250
std     6.026948   1.785922  123.938694   68.562868   0.534679   0.978457
min    10.400000   4.000000   71.100000   52.000000   2.760000   1.513000
25%    15.425000   4.000000  120.825000   96.500000   3.080000   2.581250
50%    19.200000   6.000000  196.300000  123.000000   3.695000   3.325000
```

```
75%    22.800000   8.000000  326.000000  180.000000   3.920000   3.610000
max    33.900000   8.000000  472.000000  335.000000   4.930000   5.424000

            qsec         vs         am       gear     carb
count  32.000000  32.000000  32.000000  32.000000  32.0000
mean   17.848750   0.437500   0.406250   3.687500   2.8125
std     1.786943   0.504016   0.498991   0.737804   1.6152
min    14.500000   0.000000   0.000000   3.000000   1.0000
25%    16.892500   0.000000   0.000000   3.000000   2.0000
50%    17.710000   0.000000   0.000000   4.000000   2.0000
75%    18.900000   1.000000   1.000000   4.000000   4.0000
max    22.900000   1.000000   1.000000   5.000000   8.0000
```

The Engineering World #DataScience 14 & 15

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 PEARSON CORRELATION-PARAMETRIC METHODS

1.0.1 Starting with parametric method in pandas and scipy

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from pylab import rcParams
        import seaborn as sb
        import scipy
        from scipy.stats.stats import pearsonr
```

```
In [2]: %matplotlib inline
        rcParams ['figure.figsize'] = 5,4
        sb.set_style ('whitegrid')
```

1.0.2 The Person Correlation

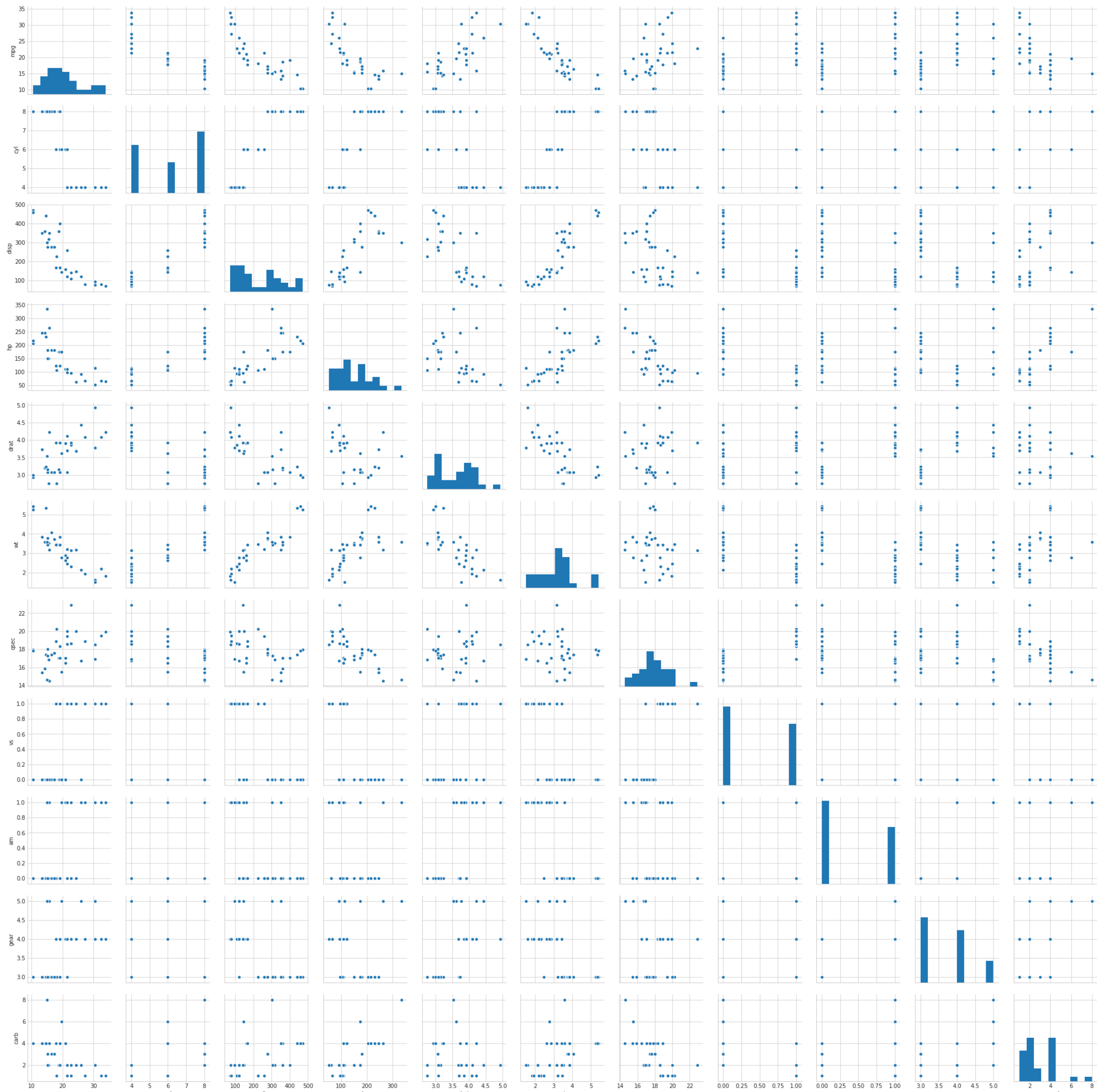
```
In [3]: address = 'mtcars.csv'
        cars = pd.read_csv(address)
        cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
        cars.head()
```

```
Out[3]:           car_names   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
        0         Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
        1     Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
        2        Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
        3    Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
        4 Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

           carb
        0     4
```

```
1    4
2    1
3    1
4    2
```

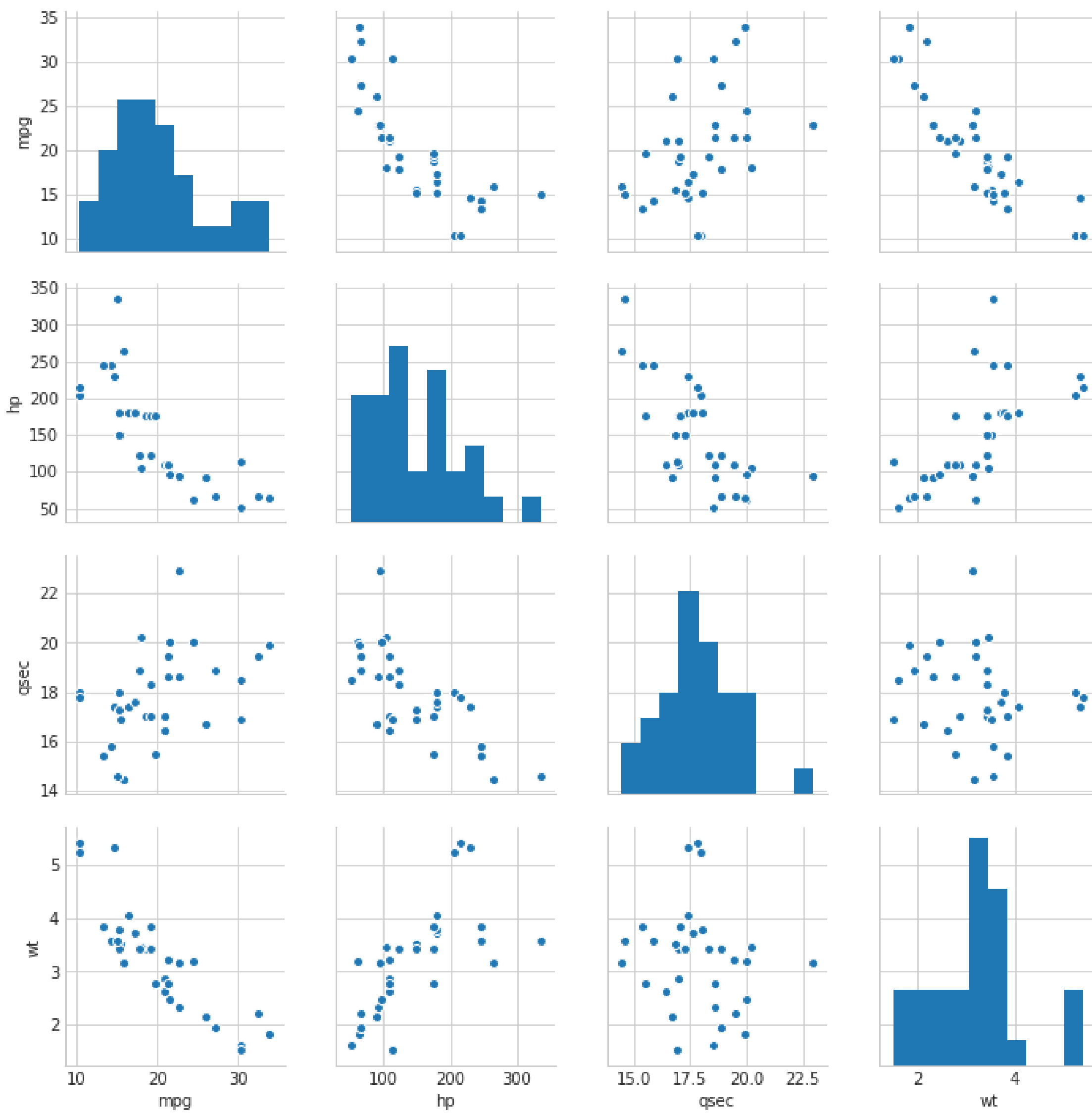
In [4]: sb.pairplot(cars)

```
Out[4]: <seaborn.axisgrid.PairGrid at 0x7f2423a317f0>
```



In [5]:
```
X = cars[['mpg','hp','qsec','wt']]
sb.pairplot(X)
```

```
Out[5]: <seaborn.axisgrid.PairGrid at 0x7f2416258048>
```



In [6]: X

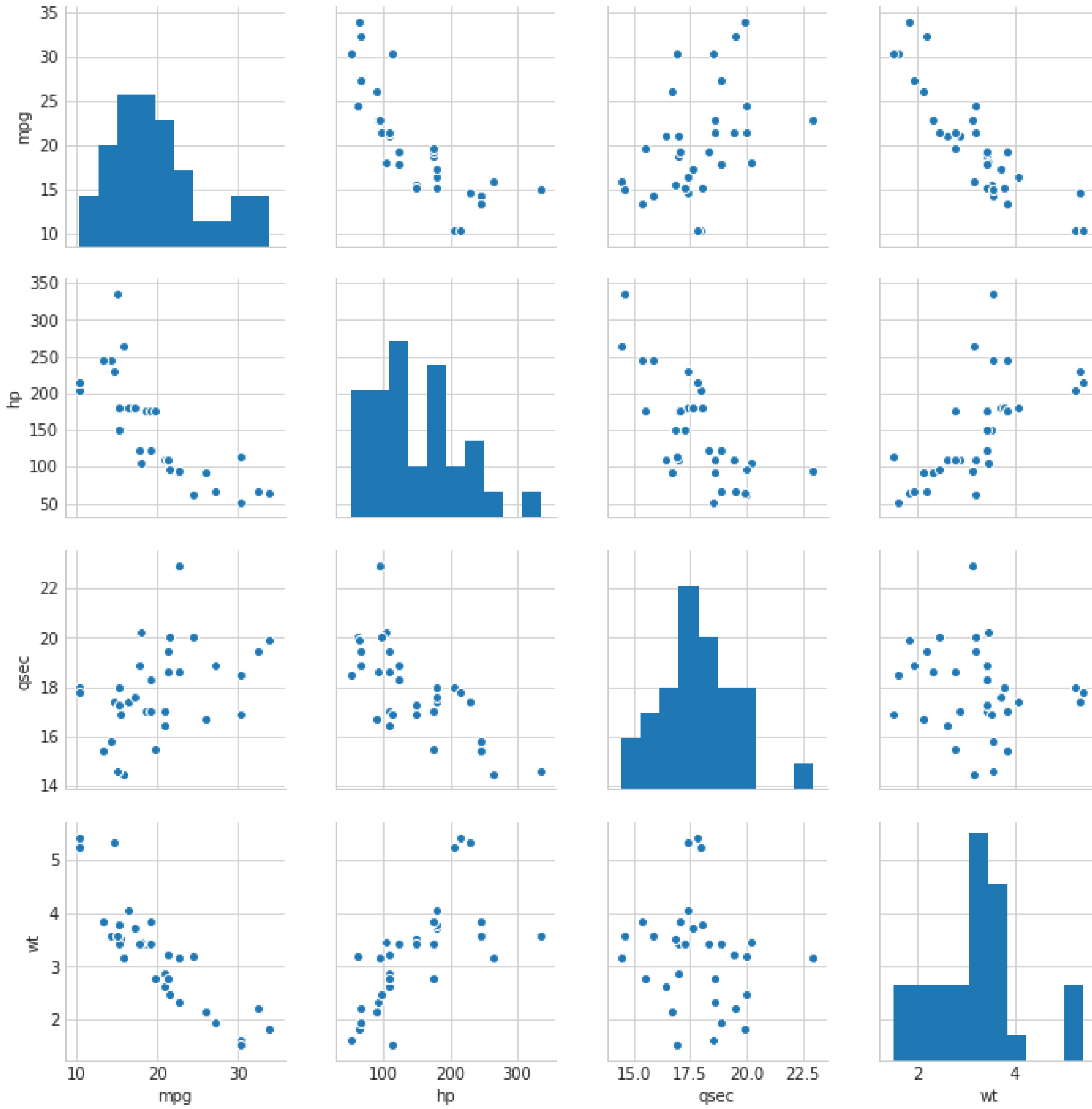
Out[6]:

	mpg	hp	qsec	wt
0	21.0	110	16.46	2.620
1	21.0	110	17.02	2.875
2	22.8	93	18.61	2.320
3	21.4	110	19.44	3.215
4	18.7	175	17.02	3.440
5	18.1	105	20.22	3.460
6	14.3	245	15.84	3.570
7	24.4	62	20.00	3.190
8	22.8	95	22.90	3.150
9	19.2	123	18.30	3.440

```
10  17.8  123  18.90  3.440
11  16.4  180  17.40  4.070
12  17.3  180  17.60  3.730
13  15.2  180  18.00  3.780
14  10.4  205  17.98  5.250
15  10.4  215  17.82  5.424
16  14.7  230  17.42  5.345
17  32.4   66  19.47  2.200
18  30.4   52  18.52  1.615
19  33.9   65  19.90  1.835
20  21.5   97  20.01  2.465
21  15.5  150  16.87  3.520
22  15.2  150  17.30  3.435
23  13.3  245  15.41  3.840
24  19.2  175  17.05  3.845
25  27.3   66  18.90  1.935
26  26.0   91  16.70  2.140
27  30.4  113  16.90  1.513
28  15.8  264  14.50  3.170
29  19.7  175  15.50  2.770
30  15.0  335  14.60  3.570
31  21.4  109  18.60  2.780
```

In [7]:
```
sb.pairplot(X)
#histogram represent Normally distributed
#cluster point represent linearly distrinuted
```

Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x7f2412697550>
```



1.0.3 Using cipy to calculate the pearson correlation coefficient

In [8]:
```
mpg = cars['mpg']
hp = cars['hp']
qsec = cars['qsec']
wt = cars['wt']
```

In [9]:
```
pearsonr_coefficient, p_value = pearsonr(mpg, hp)
print ('PearsonR Correlation Coefficient %0.3f' % (pearsonr_coefficient))
```

```
PearsonR Correlation Coefficient -0.776
```

```
In [10]: pearsonr_coefficient, p_value = pearsonr(mpg, qsec)
         print ('PearsonR Correlation Coefficient %0.3f' % (pearsonr_coefficient))
```

```
PearsonR Correlation Coefficient 0.419
```

```
In [11]: pearsonr_coefficient, p_value = pearsonr(mpg, wt)
         print ('PearsonR Correlation Coefficient %0.3f' % (pearsonr_coefficient))
```

```
PearsonR Correlation Coefficient -0.868
```

```
In [12]: corr = X.corr()
```

```
In [13]: corr
```

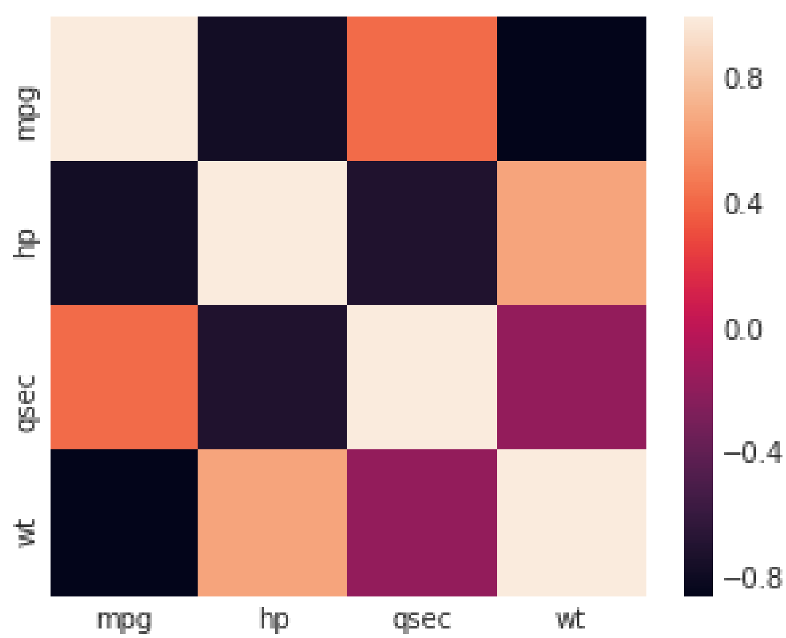
Out[13]:

	mpg	hp	qsec	wt
mpg	1.000000	-0.776168	0.418684	-0.867659
hp	-0.776168	1.000000	-0.708223	0.658748
qsec	0.418684	-0.708223	1.000000	-0.174716
wt	-0.867659	0.658748	-0.174716	1.000000

1.0.4 Using pandas to calculate the pearson correlation coefficient

```
In [14]: sb.heatmap(corr, xticklabels = corr.columns.values, yticklabels = corr.columns.values)
```

```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2411729320>
```



1.0.5 Using Seaborn to visualize the pearson correlation coefficient

2 SPEARNAM'S RANK CORRELATION AND CHI-SQUARE TABLE TEST

2.0.1 Non-parametric methods using pandas and scipy

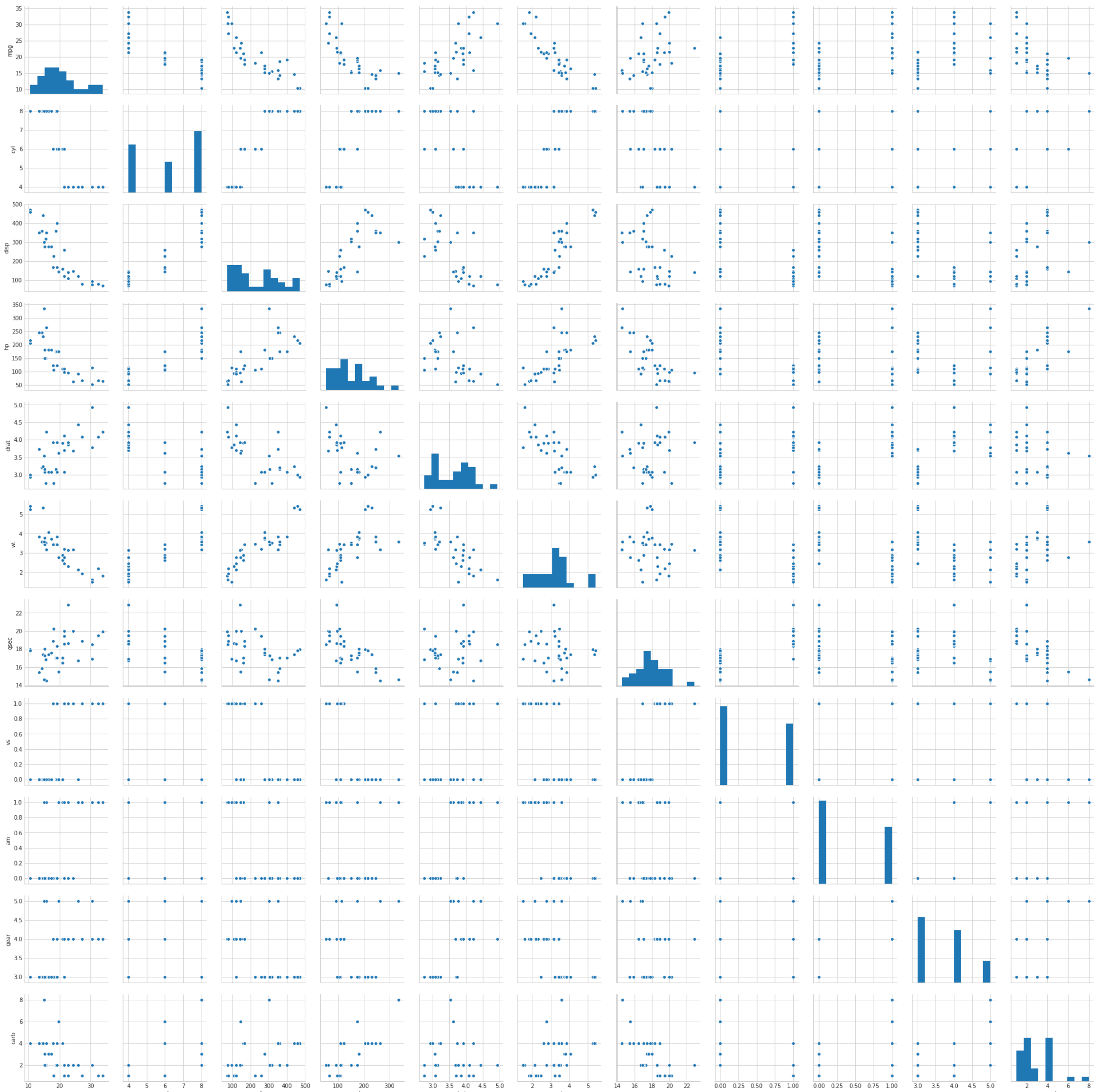
2.0.2 The Spearman Rank Correlation

In [15]: `cars.head()`

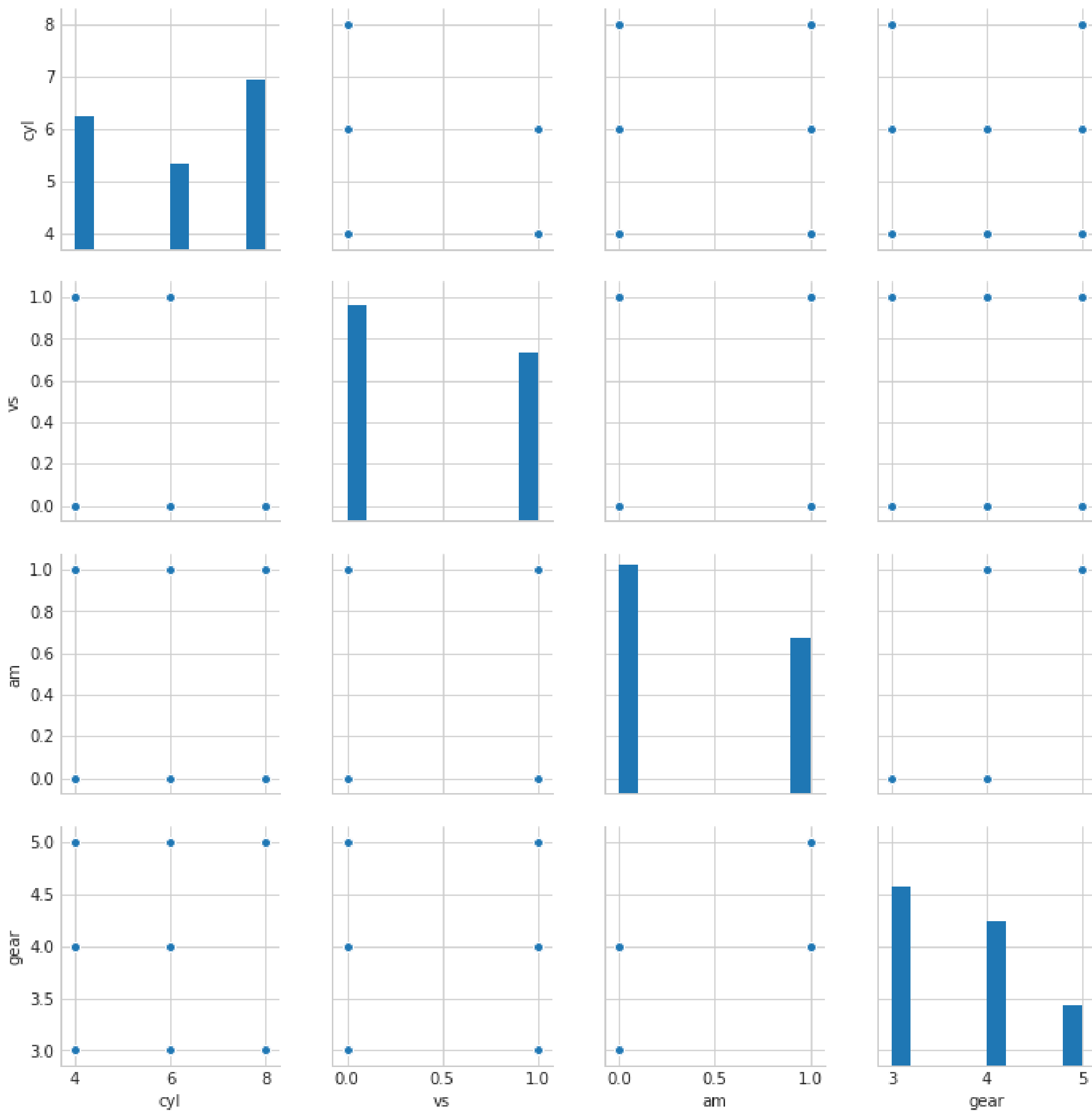
```
Out[15]:            car_names   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
         0          Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
         1      Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
         2         Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
         3     Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
         4  Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

            carb
         0     4
         1     4
         2     1
         3     1
         4     2
```

In [16]: `sb.pairplot(cars)`

Out[16]: `<seaborn.axisgrid.PairGrid at 0x7f241067a3c8>`




```
In [17]: X = cars[['cyl', 'vs', 'am', 'gear']]
         sb.pairplot(X)

Out[17]: <seaborn.axisgrid.PairGrid at 0x7f2409cb32b0>
```



In [18]:
```
cyl = cars['cyl']
vs = cars['vs']
am = cars['am']
gear = cars['gear']

pearsonr_coefficient, p_value = pearsonr(cyl, vs)
print ('PearsonR Correlation Coefficient %0.3f' % (pearsonr_coefficient))
```

```
PearsonR Correlation Coefficient -0.811
```

In [19]:
```
pearsonr_coefficient, p_value = pearsonr(cyl, am)
print ('PearsonR Correlation Coefficient %0.3f' % (pearsonr_coefficient))
```

```
PearsonR Correlation Coefficient -0.523
```

```
In [20]: pearsonr_coefficient, p_value = pearsonr(cyl, gear)
         print ('PearsonR Correlation Coefficient %0.3f' % (pearsonr_coefficient))
```

```
PearsonR Correlation Coefficient -0.493
```

2.0.3 Chi-squar test for independence

```
In [21]: table = pd.crosstab(cyl, am) #select table value

         from scipy.stats import chi2_contingency #import chi2 library
         chi2, p, dof, expected = chi2_contingency(table.values) #calculate chi2 value
         print ('Chi-square Statistic %0.3f p_value %0.3f' % (chi2, p))
```

```
Chi-square Statistic 8.741 p_value 0.013
```

```
In [22]: table = pd.crosstab(cars['cyl'],cars['vs'])

         from scipy.stats import chi2_contingency
         chi2, p, dof, expected = chi2_contingency(table.values)
         print ('Chi-square Statistic %0.3f p_value %0.3f' % (chi2, p))
```

```
Chi-square Statistic 21.340 p_value 0.000
```

```
In [23]: table = pd.crosstab(cars['cyl'],cars['gear'])

         from scipy.stats import chi2_contingency
         chi2, p, dof, expected = chi2_contingency(table.values)
         print ('Chi-square Statistic %0.3f p_value %0.3f' % (chi2, p))
```

```
Chi-square Statistic 18.036 p_value 0.001
```

```
In [24]: table = pd.crosstab(cars['cyl'],cars['am'])

         from scipy.stats import chi2_contingency
         chi2, p, dof, expected = chi2_contingency(table.values)
         print ('Chi-square Statistic %0.3f p_value %0.3f' % (chi2, p))
```

```
Chi-square Statistic 8.741 p_value 0.013
```

```
In [25]: table = pd.crosstab(cars['gear'],cars['vs'])

         from scipy.stats import chi2_contingency
         chi2, p, dof, expected = chi2_contingency(table.values)
         print ('Chi-square Statistic %0.3f p_value %0.3f' % (chi2, p))
```

```
Chi-square Statistic 12.224 p_value 0.002
```

The Engineering World #DataScience 16 & 17

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 SCALING AND DISTRIBUTION OF DATA

1.0.1 Transforming dataset distributions

```
In [1]: import numpy as np
        import pandas as pd
        from pandas import Series, DataFrame
        import matplotlib.pyplot as plt

        from pylab import rcParams
        import seaborn as sb

        import scipy

        import sklearn
        from sklearn import preprocessing
        from sklearn.preprocessing import scale
        from scipy.stats.stats import pearsonr
```

```
In [2]: %matplotlib inline
        rcParams ['figure.figsize'] = 5,4
        sb.set_style ('whitegrid')
```

1.0.2 Normalizing an transform features with MinMaxScalar() and fit_transform

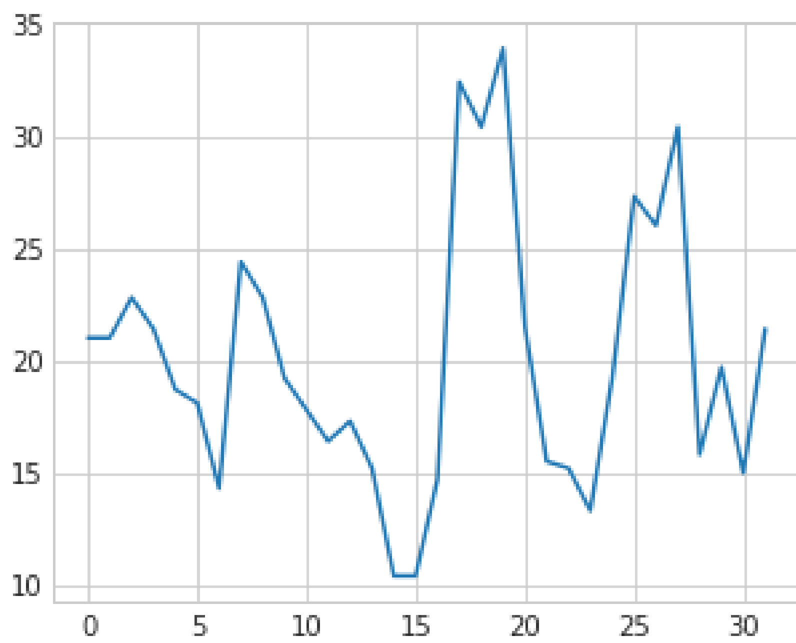
```
In [3]: address = 'mtcars.csv'
        cars = pd.read_csv(address)
        cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
        cars.head()
```

```
Out[3]:            car_names   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
        0          Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
```

```
1       Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
2          Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
3      Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
4   Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

   carb
0     4
1     4
2     1
3     1
4     2
```

In [4]:
```
mpg = cars.mpg
plt.plot(mpg)
```

Out[4]: [<matplotlib.lines.Line2D at 0x7f6bbd0b6f60>]

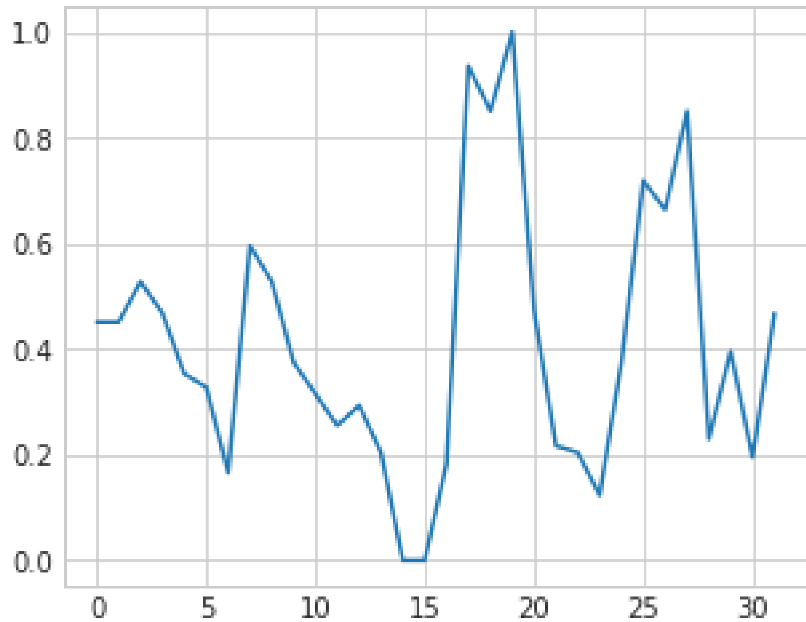


In [5]:
```
cars[['mpg']].describe()
```

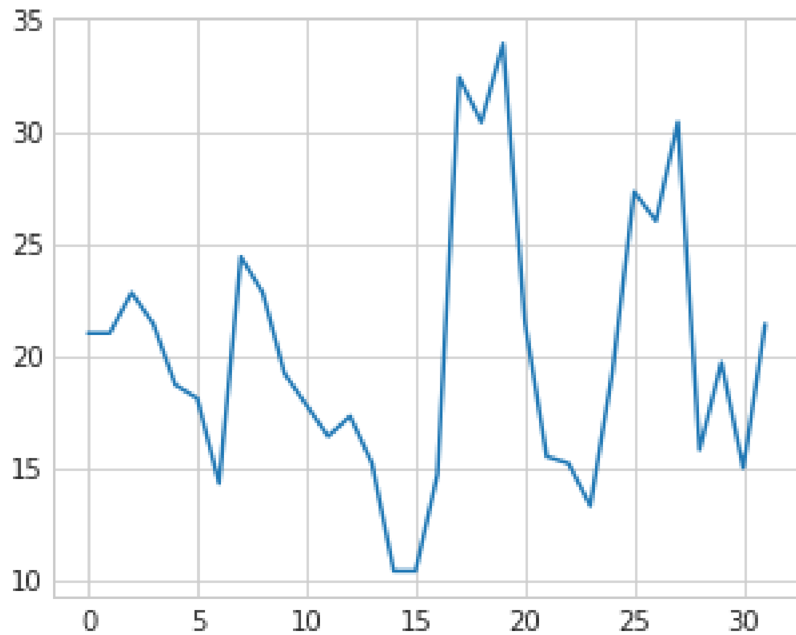
Out[5]:

	mpg
count	32.000000
mean	20.090625
std	6.026948
min	10.400000
25%	15.425000
50%	19.200000
75%	22.800000
max	33.900000

```
In [6]: mpg_matrix = mpg.values.reshape(-1,1) #scaled
        scaled = preprocessing.MinMaxScaler()

        scaled_mpg = scaled.fit_transform(mpg_matrix)
        plt.plot(scaled_mpg)

Out[6]: [<matplotlib.lines.Line2D at 0x7f6bbcfed978>]
```

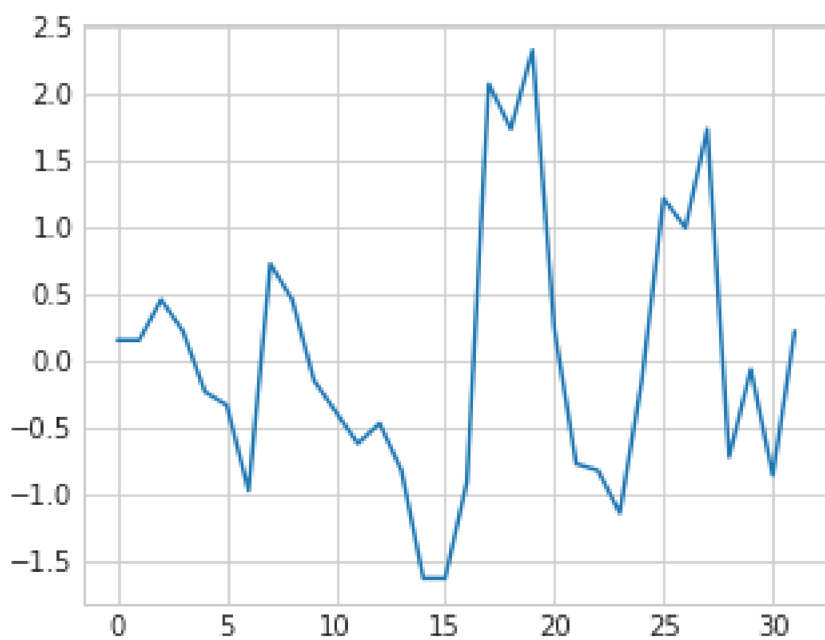



```
In [7]: standardized_mpg = scale(mpg, axis = 0, with_mean = False, with_std = False)
        plt.plot(standardized_mpg)

Out[7]: [<matplotlib.lines.Line2D at 0x7f6bbcfd9940>]
```




```
In [8]: standardized_mpg = scale(mpg)
        plt.plot(standardized_mpg)
```

```
Out[8]: [<matplotlib.lines.Line2D at 0x7f6bbcf41518>]
```



1.0.3 Using scale() to scale your features

2 INTRODUCTION TO MACHINE LEARNING

The Engineering World #DataScience 18 & 19

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 FACTOR ANALYSIS

1.0.1 Explanatory Factor Analysis

```
In [1]: import numpy as np
        import pandas as pd
        import sklearn
        from sklearn.decomposition import FactorAnalysis
        from sklearn import datasets
```

1.0.2 Factor analysis on iris data sets

```
In [2]: iris = datasets.load_iris()
        X = iris.data
        variable_names =iris.feature_names
        X[0:10,]

Out[2]: array([[5.1, 3.5, 1.4, 0.2],
               [4.9, 3. , 1.4, 0.2],
               [4.7, 3.2, 1.3, 0.2],
               [4.6, 3.1, 1.5, 0.2],
               [5. , 3.6, 1.4, 0.2],
               [5.4, 3.9, 1.7, 0.4],
               [4.6, 3.4, 1.4, 0.3],
               [5. , 3.4, 1.5, 0.2],
               [4.4, 2.9, 1.4, 0.2],
               [4.9, 3.1, 1.5, 0.1]])

In [3]: factor = FactorAnalysis().fit(X)

In [4]: pd.DataFrame(factor.components_,columns = variable_names)
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.707227	-0.153147	1.653151	0.701569
1	0.114676	0.159763	-0.045604	-0.014052
2	-0.000000	0.000000	0.000000	0.000000
3	-0.000000	0.000000	0.000000	-0.000000

2 PRINCIPAL COMPONENT ANALYSIS AND SINGULAR VALUE DECOMPOSITION

2.0.1 Principal Component Analysis(PCA)

```
In [5]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import pylab as plt
        import seaborn as sb
        from IPython.display import Image
        from IPython.core.display import HTML
        from pylab import rcParams
        import sklearn
        from sklearn import decomposition
        from sklearn.decomposition import PCA
        from sklearn import datasets
```

```
In [6]: %matplotlib inline
        rcParams['figure.figsize'] = 5, 4
        sb.set_style('whitegrid')
```

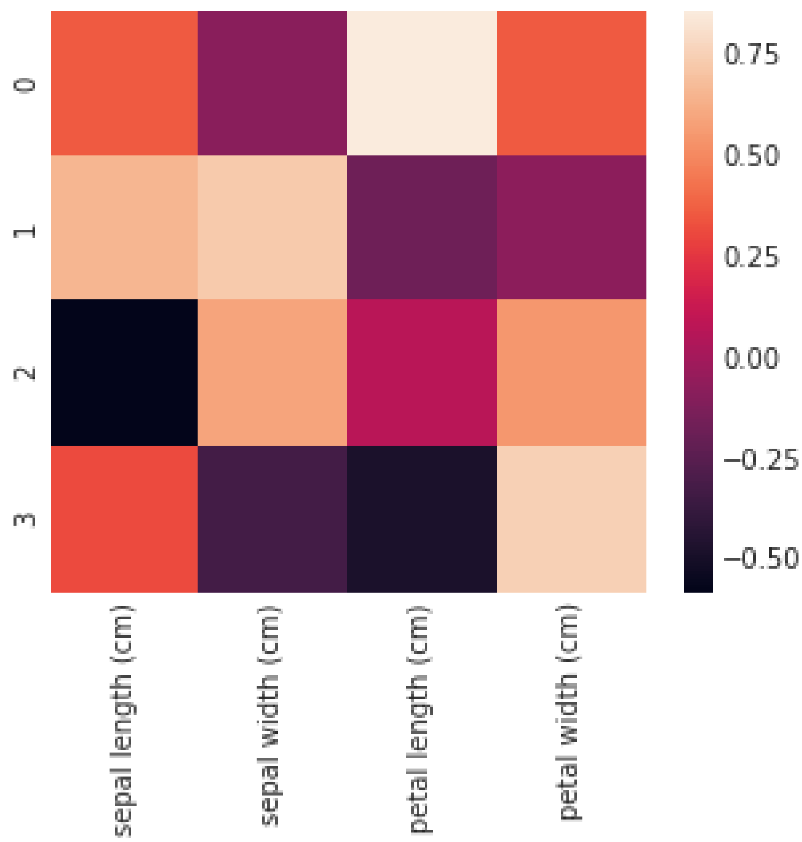
2.0.2 PCA on the iris dataset

```
In [7]: iris = datasets.load_iris()
        X = iris.data
        variable_names = iris.feature_names
        X[0:10,]
```

```
Out[7]: array([[5.1, 3.5, 1.4, 0.2],
               [4.9, 3. , 1.4, 0.2],
               [4.7, 3.2, 1.3, 0.2],
               [4.6, 3.1, 1.5, 0.2],
               [5. , 3.6, 1.4, 0.2],
               [5.4, 3.9, 1.7, 0.4],
               [4.6, 3.4, 1.4, 0.3],
               [5. , 3.4, 1.5, 0.2],
               [4.4, 2.9, 1.4, 0.2],
               [4.9, 3.1, 1.5, 0.1]])
```

```
In [8]: pca = decomposition.PCA()
        iris_pca = pca.fit_transform(X)
        pca.explained_variance_ratio_
```

```
Out[8]: array([0.92461621, 0.05301557, 0.01718514, 0.00518309])

In [9]: pca.explained_variance_ratio_.sum()

Out[9]: 1.0

In [10]: comps = pd.DataFrame (pca.components_, columns = variable_names)
         comps

Out[10]:    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
         0           0.361590         -0.082269           0.856572          0.358844
         1           0.656540          0.729712          -0.175767         -0.074706
         2          -0.580997          0.596418           0.072524          0.549061
         3           0.317255         -0.324094          -0.479719          0.751121

In [11]: sb.heatmap(comps)

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2fddf3ad30>
```



The Engineering World #DataScience 20 & 21

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 OUTLIER ANALYSIS DETECTION WITH UNIVARIATE METHOD USING TUKEY BOXPLOTS

1.0.1 Extreme value analysis using Univariate Methods

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from pylab import rcParams
```

```
In [2]: %matplotlib inline
        rcParams['figure.figsize'] = 5, 4
```

```
In [3]: df = pd.read_csv('iris_data_nepal.csv', header = None, sep = ',')
        df.columns = ['Special Length', 'Special Width', 'Petal Length', 'Petal Width', 'Species
        X = df.iloc[:,0:4].values
        y = df.iloc[:,4].values
        df[:5]
```

```
        ---------------------------------------------------------------------------

        UnicodeDecodeError                        Traceback (most recent call last)

        pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_tokens()


        pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_with_dtype()


        pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._string_convert()
```

```
    pandas/_libs/parsers.pyx in pandas._libs.parsers._string_box_utf8()


    UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start by


During handling of the above exception, another exception occurred:


    UnicodeDecodeError                       Traceback (most recent call last)

    <ipython-input-3-815e685eb497> in <module>()
----> 1 df = pd.read_csv('iris_data_nepal.csv', header = None, sep = ',')
      2 df.columns = ['Special Length', 'Special Width', 'Petal Length', 'Petal Width', 'Spe
      3 X = df.iloc[:,0:4].values
      4 y = df.iloc[:,4].values
      5 df[:5]


    ~/anaconda3/lib/python3.6/site-packages/pandas/io/parsers.py in parser_f(filepath_or_buf
    707                     skip_blank_lines=skip_blank_lines)
    708
--> 709         return _read(filepath_or_buffer, kwds)
    710
    711     parser_f.__name__ = name


    ~/anaconda3/lib/python3.6/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer
    453
    454     try:
--> 455         data = parser.read(nrows)
    456     finally:
    457         parser.close()


    ~/anaconda3/lib/python3.6/site-packages/pandas/io/parsers.py in read(self, nrows)
   1067                 raise ValueError('skipfooter not supported for iteration')
   1068
-> 1069         ret = self._engine.read(nrows)
   1070
   1071         if self.options.get('as_recarray'):


    ~/anaconda3/lib/python3.6/site-packages/pandas/io/parsers.py in read(self, nrows)
   1837     def read(self, nrows=None):
   1838         try:
-> 1839             data = self._reader.read(nrows)
```

```
   1840         except StopIteration:
   1841             if self._first_chunk:


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.read()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_low_memory()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_rows()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_column_data()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_tokens()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_with_dtype()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._string_convert()


    pandas/_libs/parsers.pyx in pandas._libs.parsers._string_box_utf8()


    UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start by
```

1.0.2 Identifying Outlier from Tukey boxplots

```
In [ ]: df.boxplot(return_type = 'dict')
        plt.plot()
```

```
In [ ]: Sepal_Width = X[:,1]
        iris_outliers = (Sepal_Width > 4)
        df(iris_outliers)
```

```
In [ ]: Sepal_Width = X[:,1]
        iris_outliers = (Sepal_Width < -.25)
        df(iris_outliers)
```

1.0.3 Applying Tukey outlier labeling

```
In [ ]: pd.options.display.float_format = '{:.1f}'.format
        X_df = pd.DataFrame(X)
        print x_df.describe()
```

2 MULTIVARIATE OUTLIER ANALYSIS DETECTION

2.0.1 Visually inspecting boxplots

```
In [ ]: df = pd.read_csv('iris_data_nepal.csv', header = None, sep = ',')
        df.columns = ['Special Length', 'Special Width', 'Petal Length', 'Petal Width', 'Species
        X = df.iloc[:,0:4].values
        y = df.iloc[:,4].values
        df[:5]
```

2.0.2 Looking at the scatterplot matrix

```
In [ ]: sb.boxplot(x ='Species', y = 'Sepal Length', data = df, palatte = 'hls')
```

```
In [ ]: sb.pairplot(df, hue = 'Species', platte = 'hls')
```

The Engineering World #DataScience 22 & 23

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 DBSCAN CLUSTERING FOR IDENTIFYING OUTLIER

1.0.1 DBSCAN clustering for identifying outliers

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from pylab import rcParams

In [2]: %matplotlib inline
        rcParams['figure.figsize'] = 5, 4
```

The Engineering World #DataScience 24 & 25

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 K-MEANS METHOD FOR CLUSTERING

```
In [5]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from pylab import rcParams

        import sklearn
        from sklearn.cluster import KMeans
        from mpl_toolkits.mplot3d import Axes3D
        from sklearn.preprocessing import scale

        import sklearn.metrics as sm
        from sklearn.metrics import confusion_matrix, classification_report
```

```
In [6]: %matplotlib inline
        rcParams['figure.figsize'] = 7, 4
```

```
In [7]: iris = datasets.load_iris()
        X = scale(iris.data)
        Y = pd.DataFrame(iris.target)
        variable_names = iris.feature_names
        X[0:10,]
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-7-c0b168f8d8bb> in <module>()
    ----> 1 iris = datasets.load_iris()
          2 X = scale(iris.data)
```

```
3 Y = pd.DataFrame(iris.target)
4 variable_names = iris.feature_names
5 X[0:10,]


NameError: name 'datasets' is not defined
```

1.0.1 Building and Running your model

```
In [ ]: clustering = KMeans(n_clusters = 3, random_state = 5)
        clustering.fit(X)
```

1.0.2 Plotting your model output

```
In [ ]: iris_df = pd.DataFrame(iris.data)
        iris_df.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']
        Y.columns = ['Targets']
```

```
In [ ]: color_theme = np.array(['darkgray', 'lightsalmon', 'powderblue'])

        plt.subplot(1,2,1)
        plt.scatter(x = iris_df.Petal_Length, y = iris_df.Petal_Width, c = color_theme[iris.targ
        plt.title('Ground Truth Classification')

        plt.subplot(1,2,2)
        plt.scatter(x = iris_df.Petal_Length, y = iris_df.Petal_Width, c = color_theme[clusterin
        plt.title('K-Means Classification')
```

```
In [ ]: relabel = np.choose(clustering.labels_, [2,0,1]).astype(np.int64)
        plt.subplot(1,2,1)
        plt.scatter(x = iris_df.Petal_Length, y = iris_df.Petal_Width, c = color_theme[iris.targ
        plt.title('Ground Truth Classification')

        plt.subplot(1,2,2)
        plt.scatter(x = iris_df.Petal_Length, y = iris_df.Petal_Width, c = color_theme[clusterin
        plt.title('K-Means Classification')
```

1.0.3 Evaluate your clustering result

```
In [ ]: print (classification_report(Y,relabel))
```

2 HIERARCHICAL CLUSTERING

```
In [ ]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from pylab import rcParams
```

```
import scipy
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist

import seaborn as sb

import sklearn
from sklearn.cluster import AgglomerativeClustering
import sklearn.metrics as sm
```

```
In [ ]: np.set_printoptions(precision=4, suppress=True)
        plt.Figure(figsize=(10,3))
        %matplotlib inline
        plt.style.use('seaborn-whitegrid')
```

```
In [ ]: address = 'mtcars.csv'
        cars = pd.read_csv(address)
        cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
        X = cars.ix[:,(1,3,4,6)].values
        Y = cars.ix[:,(9)].values
```

2.0.1 Using scipy to generate dendrogram

```
In [ ]: Z = linkage(X, 'ward')
```

```
In [ ]: dendrogram(Z, truncate_mode='lastp', p = 12, leaf_rotation=45, leaf_font_size=15, show_c
        plt.title('Truncate Hierarchical Clustering Dendrogram')
        plt.xlabel("cluster Size")
        plt.ylabel('Distance')
        plt.axhline(y = 500)
        plt.axhline(y = 150)
        plt.show()
```

2.0.2 Generate Hierarchical Clusters

```
In [ ]: k = 2
        Hclustering = AgglomerativeClustering(n_clusters=k, affinity='euclidean', linkage= 'ward
        Hclustering.fit(X)
        sm.accuracy_score(Y, Hclustering.labels_)
```

```
In [ ]: Hclustering = AgglomerativeClustering(n_clusters=k, affinity='euclidean', linkage= 'comp
        Hclustering.fit(X)
        sm.accuracy_score(Y, Hclustering.labels_)
```

```
In [ ]: Hclustering = AgglomerativeClustering(n_clusters=k, affinity='euclidean', linkage= 'aver
        Hclustering.fit(X)
        sm.accuracy_score(Y, Hclustering.labels_)
```

```
In [ ]: Hclustering = AgglomerativeClustering(n_clusters=k, affinity='manhattan', linkage= 'aver
        Hclustering.fit(X)
        sm.accuracy_score(Y, Hclustering.labels_)
```

The Engineering World #DataScience 26

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 K-NEAREST NEIGHBOR CLASSIFICATION

```
In [ ]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from pylab import rcParams

        import scipy
        import urllib
        import sklearn

        from sklearn.neighbors import KNeighborsClassifier
        from sklearn import neighbors
        from sklearn import preprocessing
        from sklearn.cross_validation import train_test_split
        from sklearn import metrics
```

```
In [ ]: np.set_printoptions(precision=4, suppress=True)
        %matplotlib inline
        rcParams['figure.figsize'] = 7, 4
        plt.style.use('seaborn-whitegrid')
```

1.0.1 Setting your data into test and traines datasets

```
In [ ]: address = 'mtcars.csv'
        cars = pd.read_csv(address)
        cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
        X_prime = cars.ix[:,(1,3,4,5)].values
        Y = cars.ix[:,(9)].values
```

```
In [ ]: X = preprocessing.scale(X_prime)
```

```
In [ ]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = .33, random_state =
```

1.0.2 Building and training your model with training data

```
In [ ]: clf = neighbors.KNeighborsClassifier()
        clf.fit(X_train, Y_train)
        print(clf)
```

1.0.3 Evoluting your model's prodiction against the test dataset

```
In [ ]: y_expect = Y_test
        y_pred = clf.predict(X_test)
        print(metrics.classification_report(y_expect,y_pred))
```

```
In [ ]: address = 'Advertising2.csv'
        cars = pd.read_csv(address)
        plt.plot(cars)
```

```
In [ ]: print(cars)
```

```
In [ ]: plt.hist(cars)
```

```
In [ ]: plt.bar(cars)
```

The Engineering World #DataScience 27 & 28

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 NETWORK ANALYSIS

1.0.1 Network analysis use cases

Social media marketing analysis, Infrastructure system design, Financial risk management, Public health management

1.0.2 Network

A body of connected data that's evaluated during graph analysis

1.0.3 Graph

A data visualization schematic depicting the data that comparises a network

1.0.4 Network analysis vocablary

Nodes: the vertices around which a graph is formed

Edges: the lines that connect vertices within a graph

Directed graph(aka digraph): a graph where there is a direction assign to each edge that connects a node

Directed edge: an edge feature that has been assign a direction between nodes

Undirected graph: a graph where all edges are bidirectional

Undirected eddge: a bidirectional edge feature

Graph size: the number of edge in a graph

Graph order: number of vertices is a graph

Degree: the number of edges connected to a vertex, with loops counted twice

1.0.5 Graph generator

The functions that generates graphs

graph generator has most important application is Synthetic veriation of A particular graph

Type of graph generators Graph drawing algorithms
Network analysis algorithms
Algorithmic routing for graphs
Graph search algorithms
Subgraphs algorithms

2 GRAPH OBJECT NETWORK ANALYSIS

You + Machine Learning = Scientific Discovery

2.0.1 Working with Graph objects

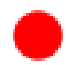
```
In [1]: import numpy as np
        import pandas as pd
        from pylab import rcParams
        import seaborn as sb
        import matplotlib.pyplot as plt
        import networkx as nx
```

```
In [2]: %matplotlib inline
        rcParams ['figure.figsize'] = 5,4
        sb.set_style ('whitegrid')
```

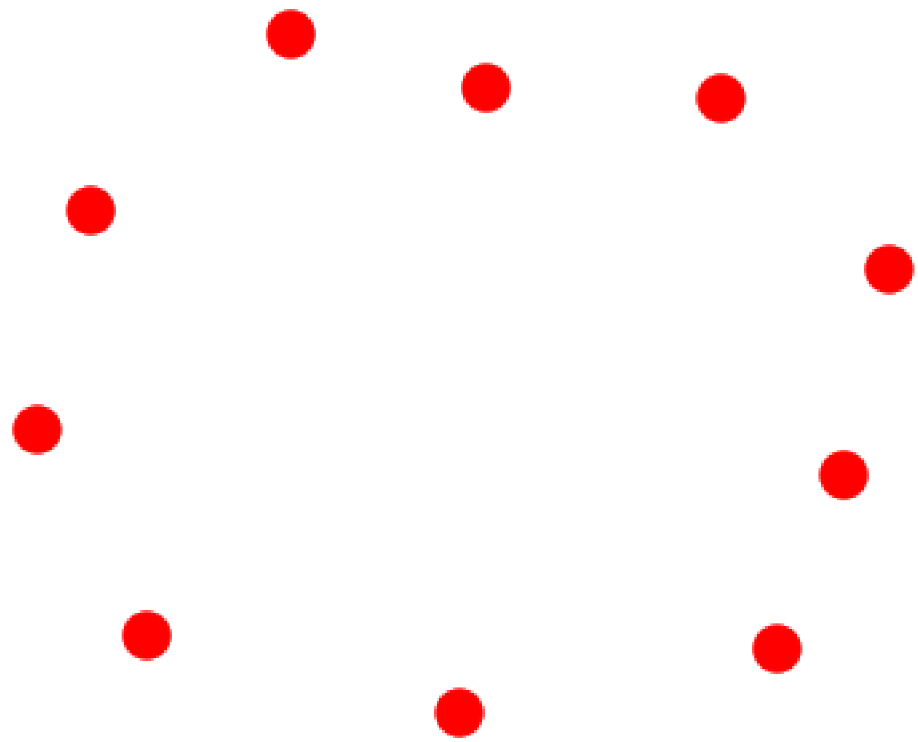
2.0.2 Creating graph objects

```
In [3]: G = nx.Graph() #empty graph drawing
        nx.draw(G)
```

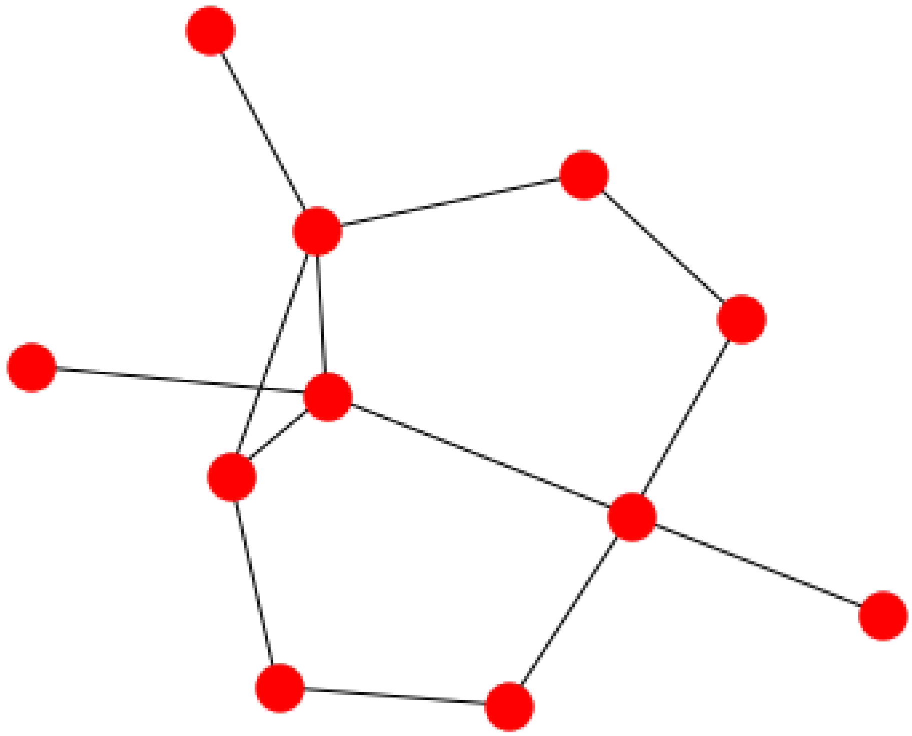
```
In [4]: G.add_node(1)
        nx.draw(G)
```



```
In [5]: G.add_nodes_from([1,2,3,4,5,6,7,8,9,10])
        nx.draw(G)
```

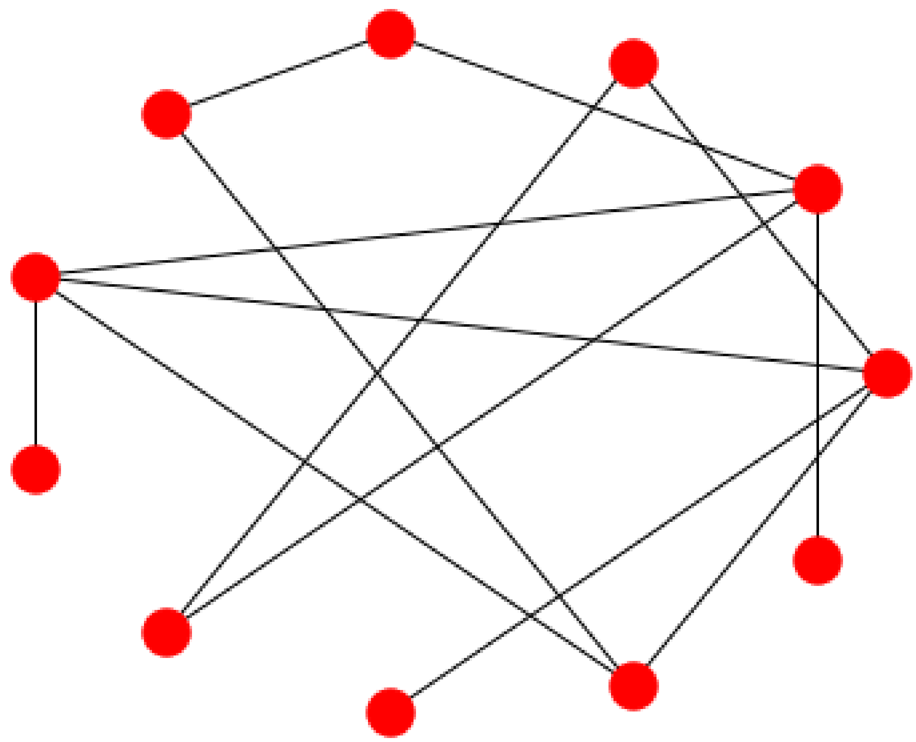


In [6]:
```
G.add_edges_from([(2,4),(2,6),(2,8),(1,3),(1,10),(2,12),(1,6),(3,8),(1,10),(5,10),(5,4),
nx.draw(G)
```

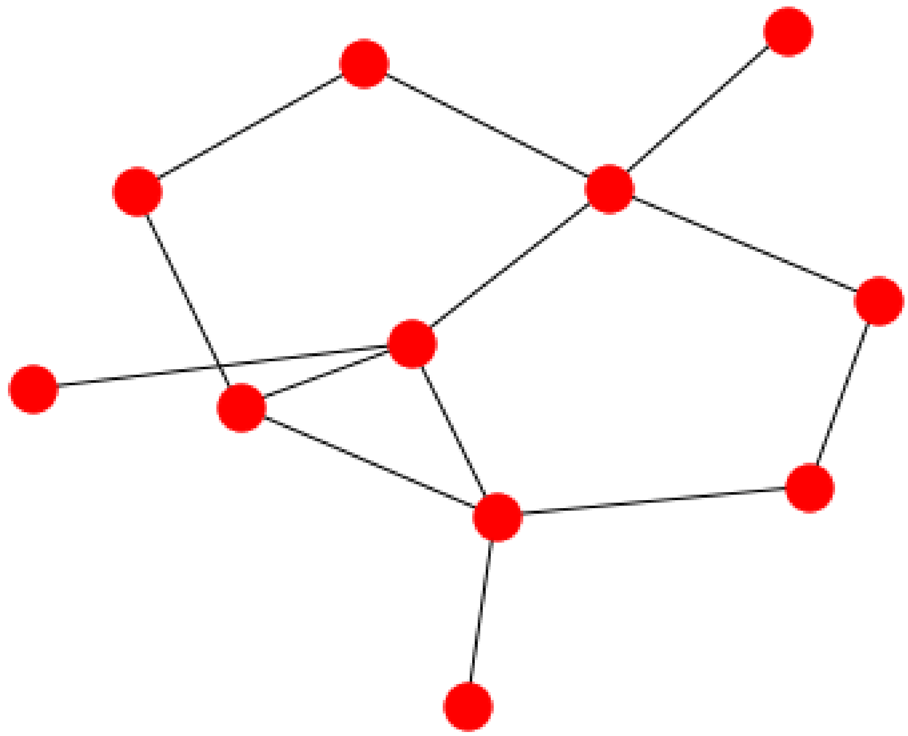


2.0.3 The basics about drawing graph objects

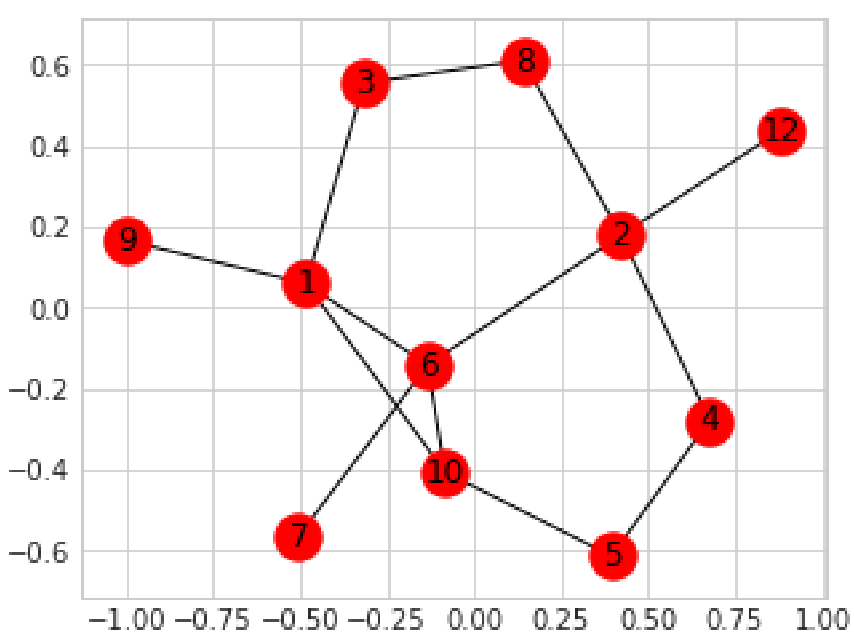
```
In [7]: nx.draw_circular(G)
```



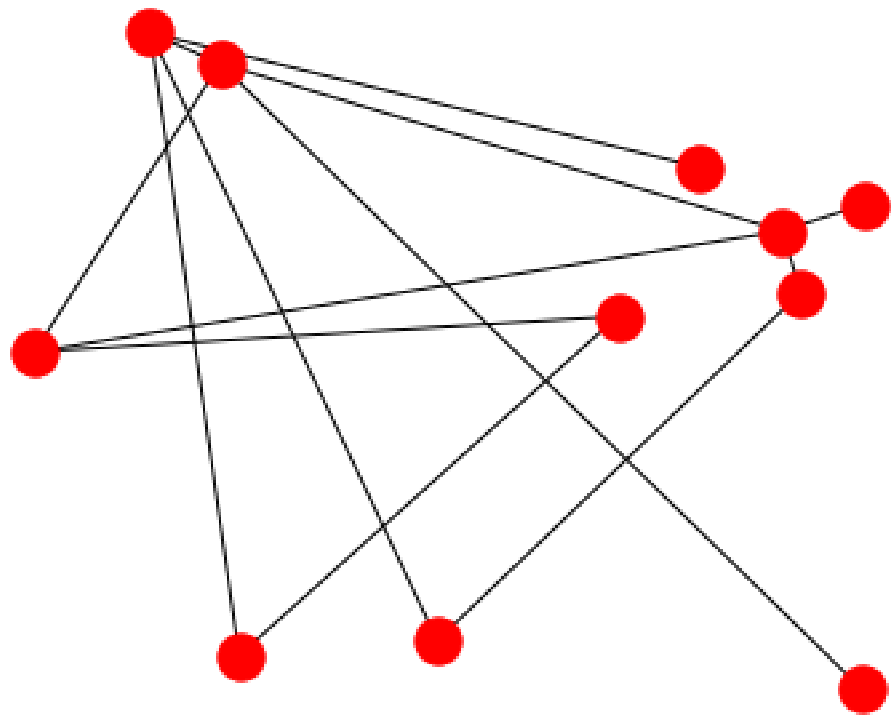
```
In [8]: nx.draw_kamada_kawai(G)
```



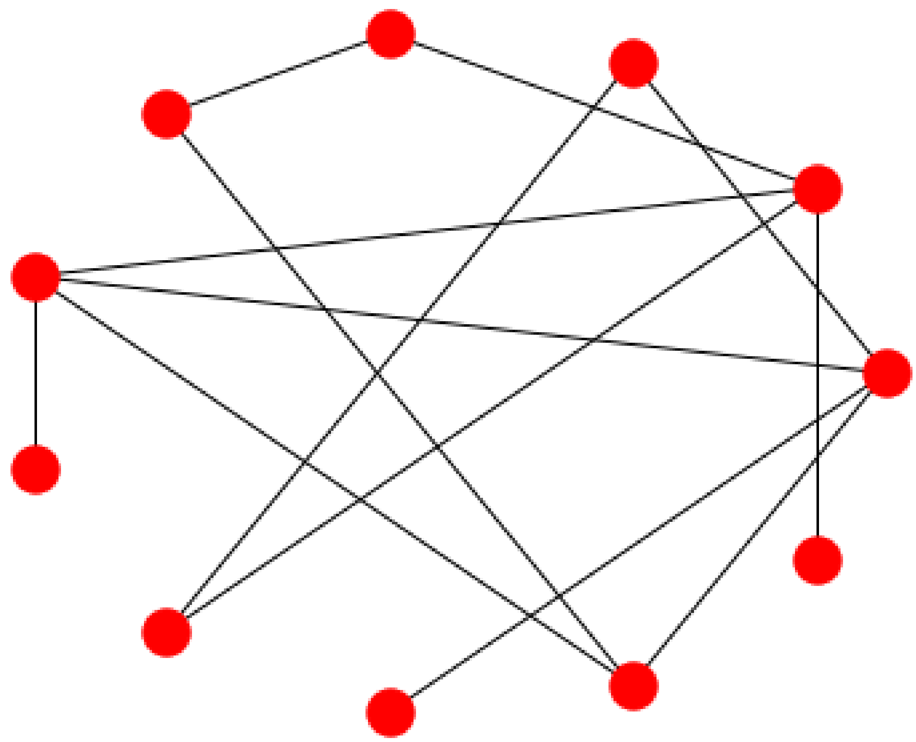
```
In [9]: nx.draw_networkx(G)
```



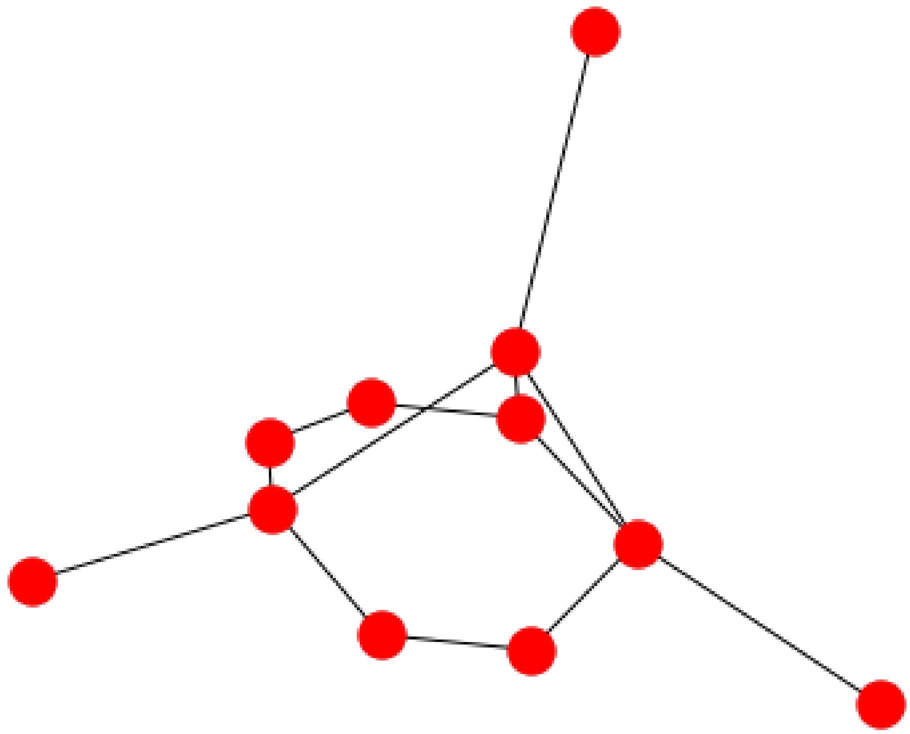
```
In [10]: nx.draw_random(G)
```



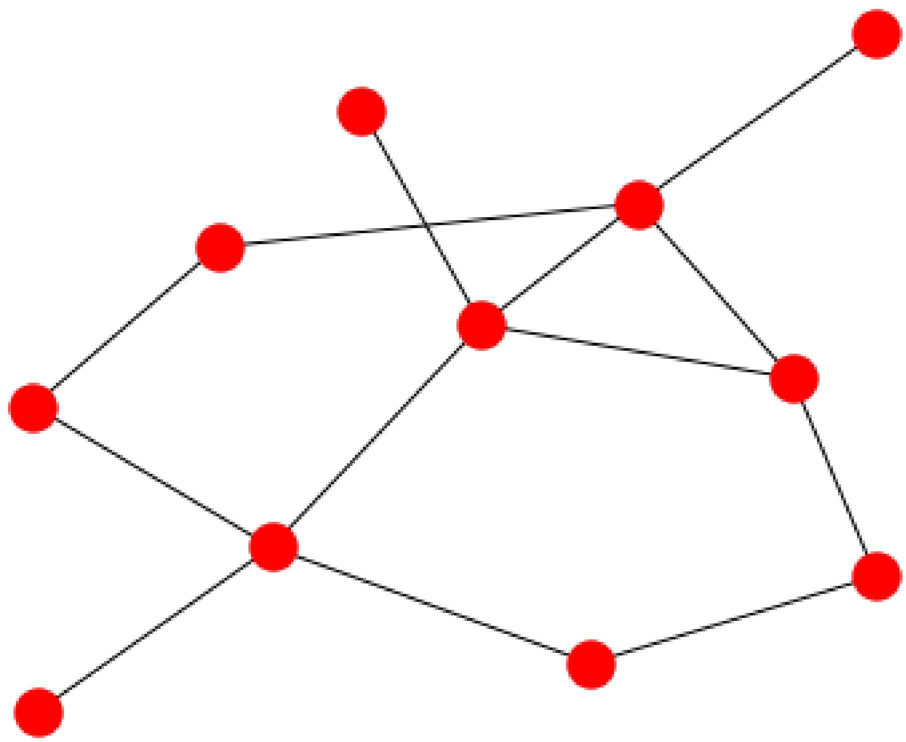
```
In [11]: nx.draw_shell(G)
```



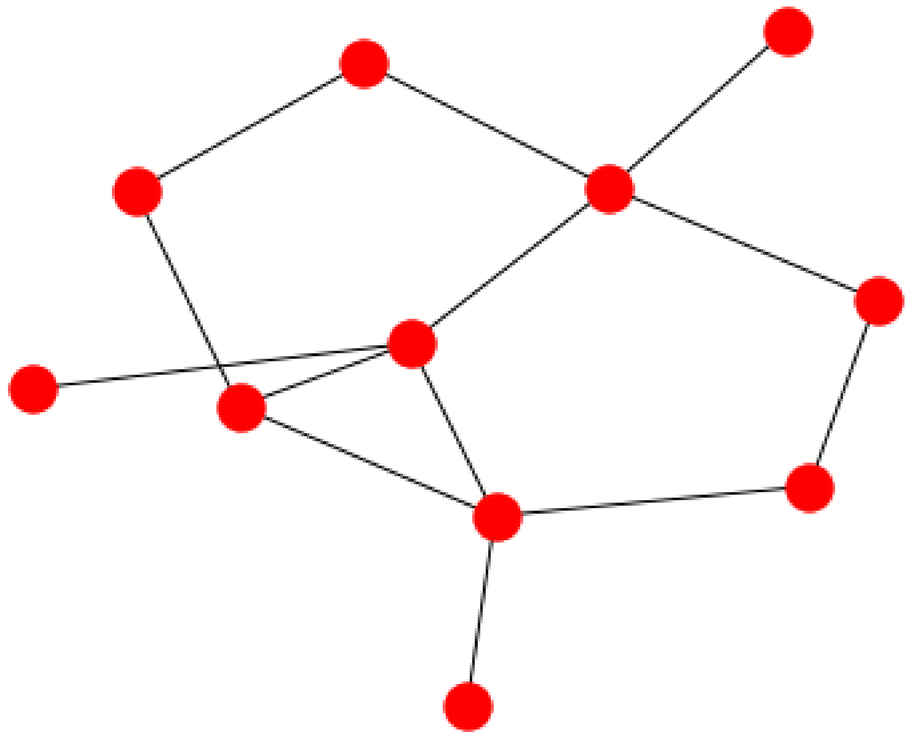
```
In [12]: nx.draw_spectral(G)
```



```
In [13]: nx.draw_spring(G)
```

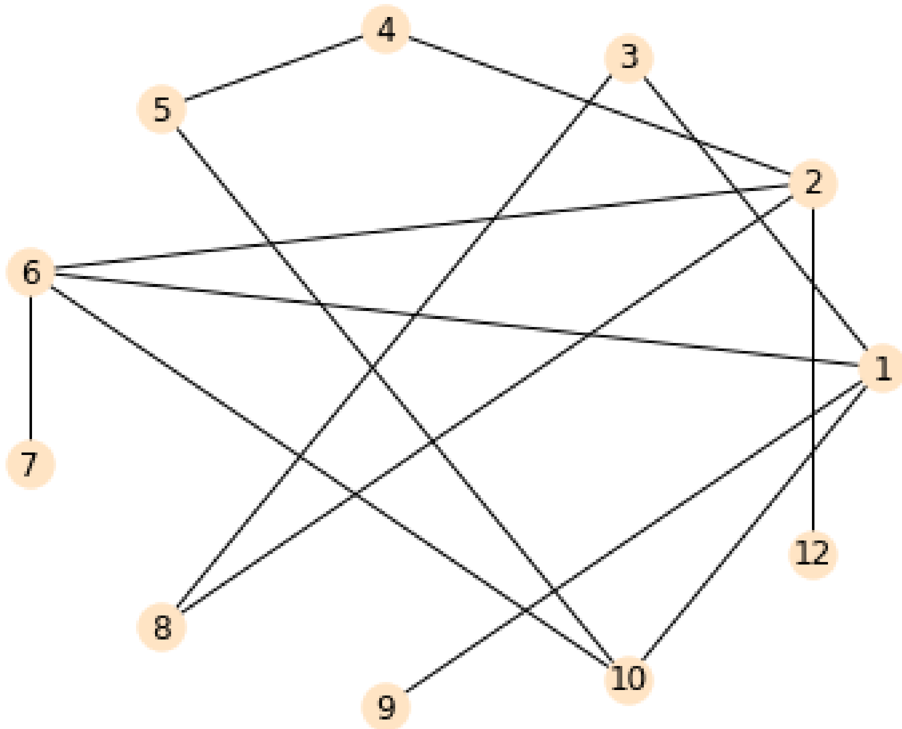


```
In [14]: nx.draw_kamada_kawai(G)
```

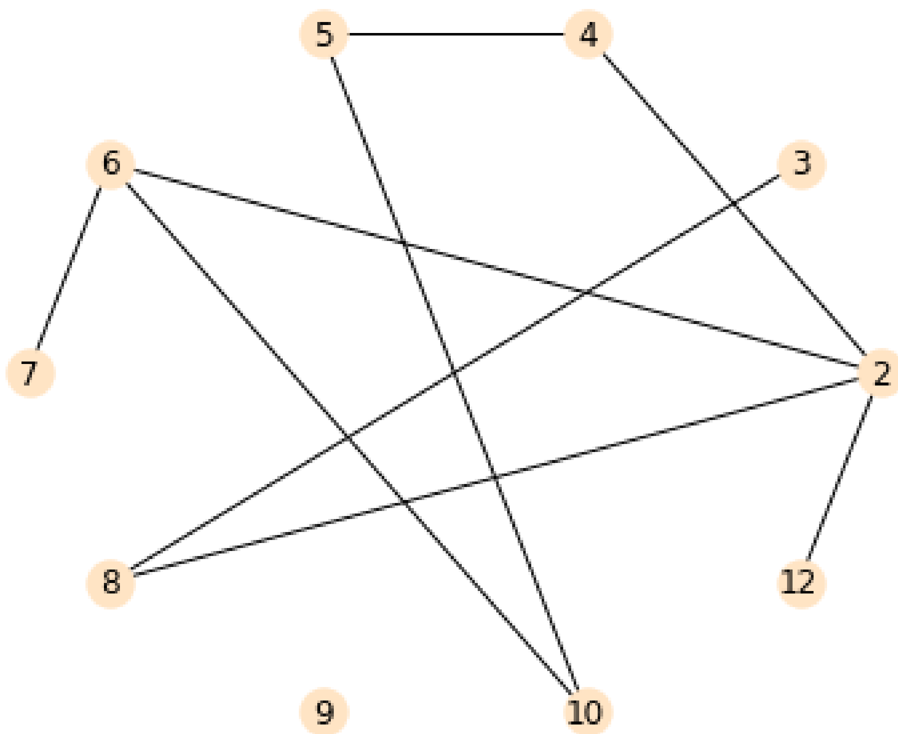


2.0.4 Labeling and coloring your graph plots

```
In [15]: nx.draw_circular(G, node_color = 'bisque', with_labels = True) #add node color and labe
```




```
In [16]: G.remove_node(1) #remove some nodes from graph plots
         nx.draw_circular(G, node_color = 'bisque', with_labels = True) #add node color and labe
```



2.0.5 Identify graph proporties

```
In [17]: sum_stats = nx.info(G)
         print (sum_stats)
```

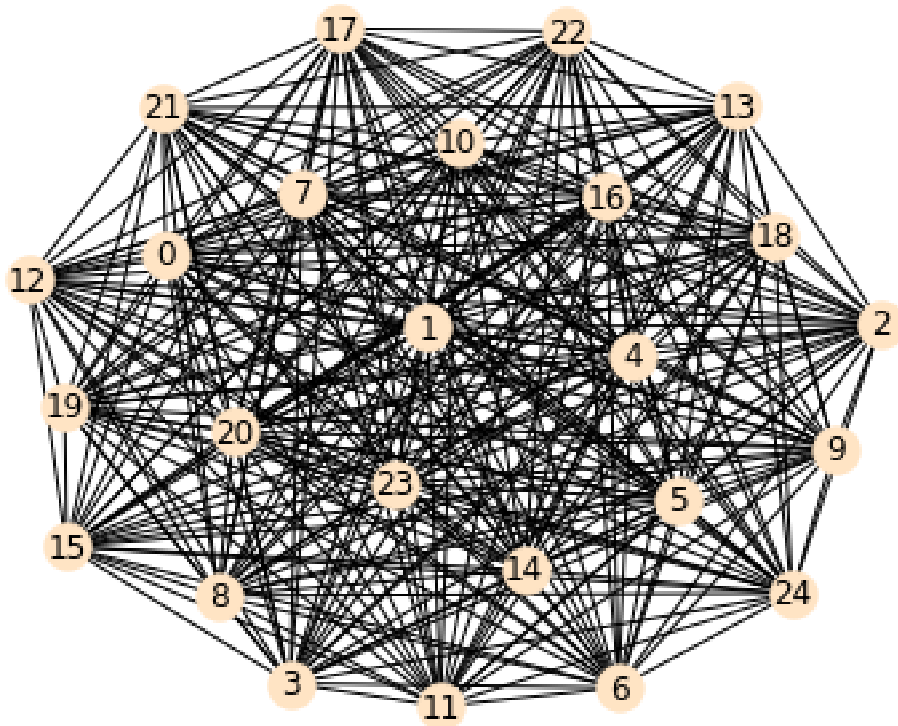
```
Name:
Type: Graph
Number of nodes: 10
Number of edges: 9
Average degree:   1.8000
```

```
In [18]: print (nx.degree(G))
```

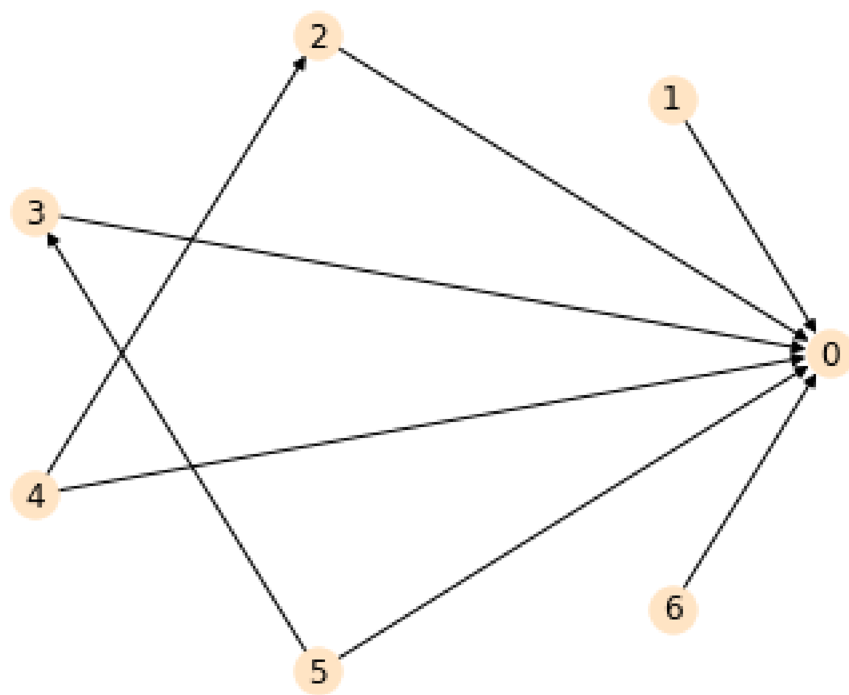
```
[(2, 4), (3, 1), (4, 2), (5, 2), (6, 3), (7, 1), (8, 2), (9, 0), (10, 2), (12, 1)]
```

2.0.6 Using graph generator

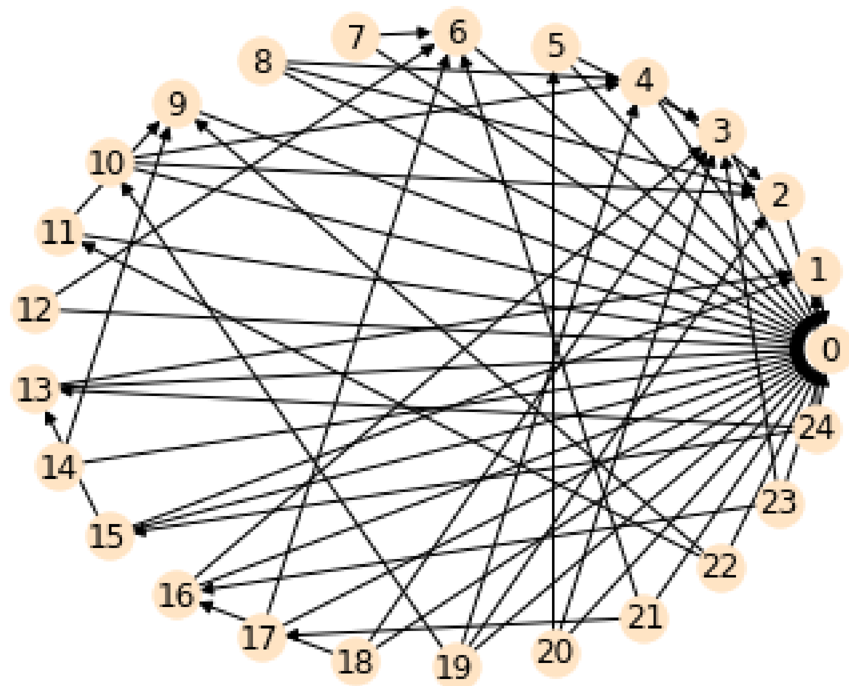
```
In [19]: G = nx.complete_graph(25)
         nx.draw(G, node_color='bisque', with_labels=True)
```



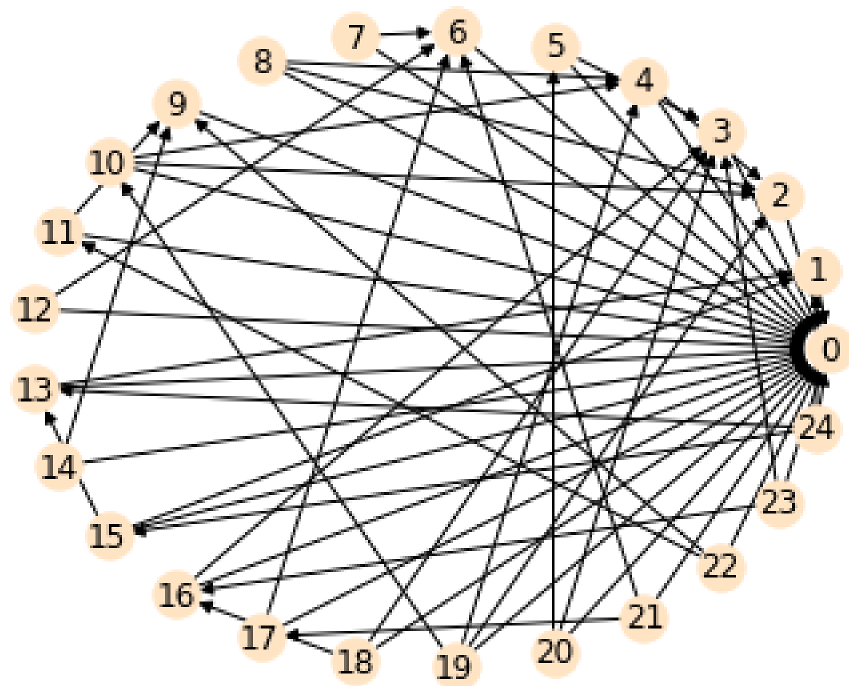
In [20]:
```
G = nx.gnc_graph(7, seed=25) #add a direction arrow in the graph
nx.draw_circular(G, node_color='bisque', with_labels=True)
```



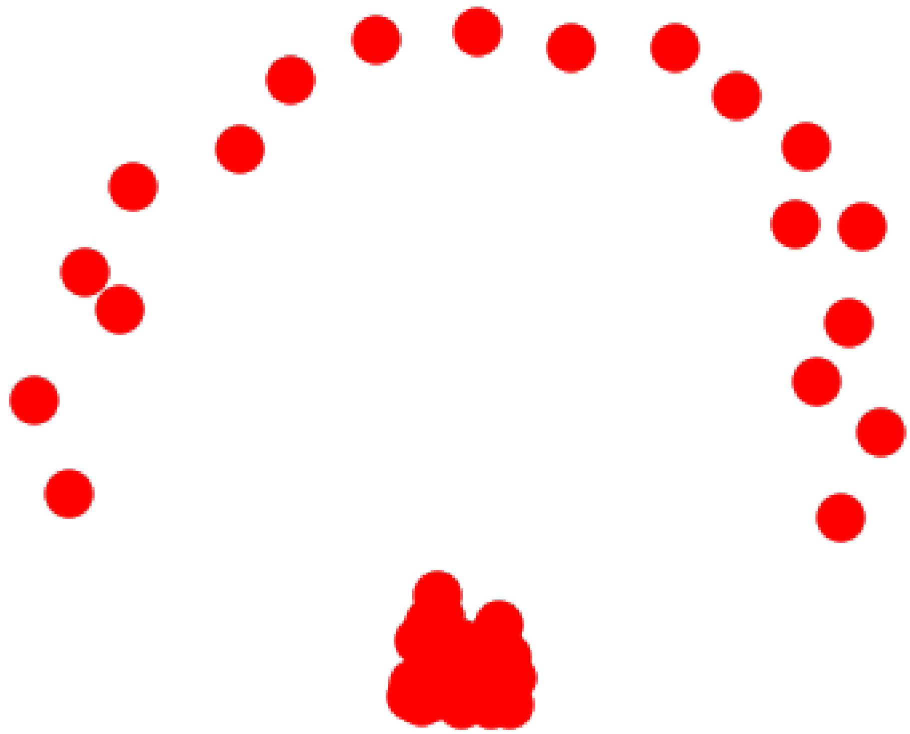

```
In [21]: G = nx.gnc_graph(25, seed=25)
         nx.draw_circular(G, node_color='bisque', with_labels=True)
```



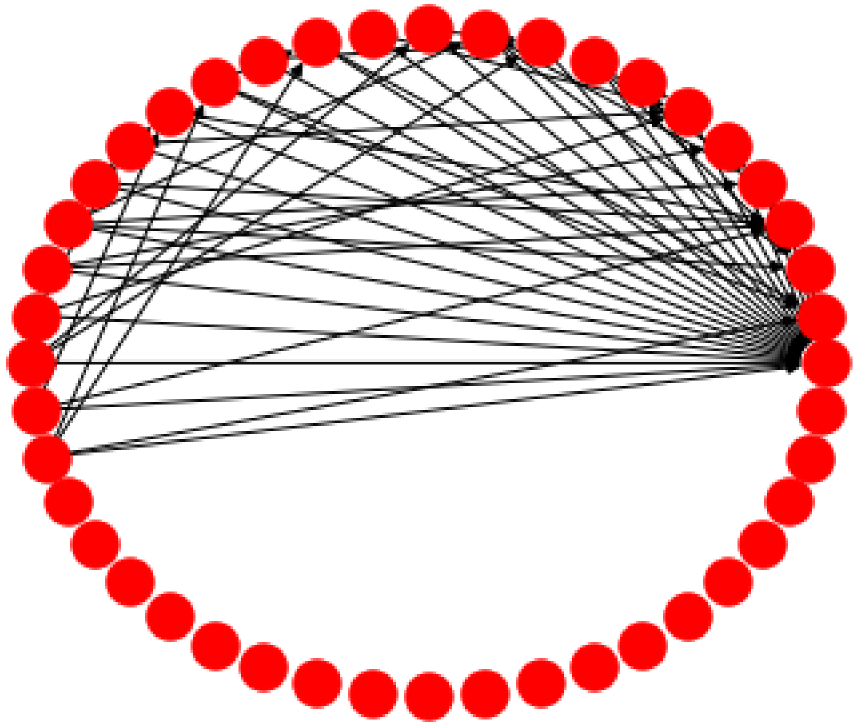

```
In [22]: ego_G = nx.ego_graph(G, 15, radius=5)
         nx.draw_circular(G, node_color='bisque', with_labels=True)
```



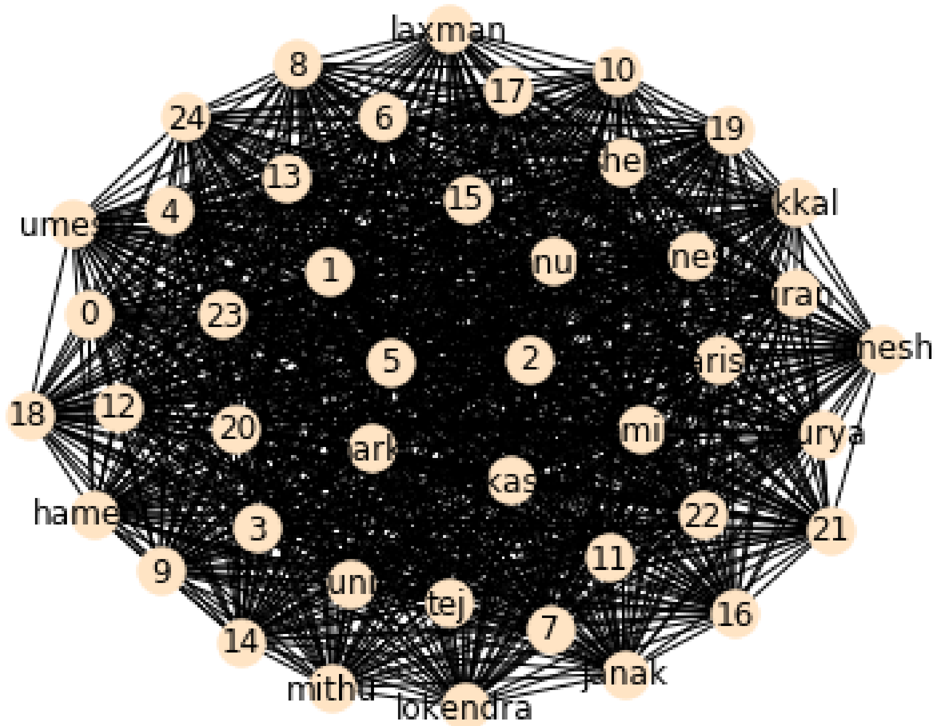
In [23]:
```
G.add_nodes_from(['akkal','janak','laxman','bikash','dinesh','amin','sunil','kiran','ha
nx.draw(G)
```



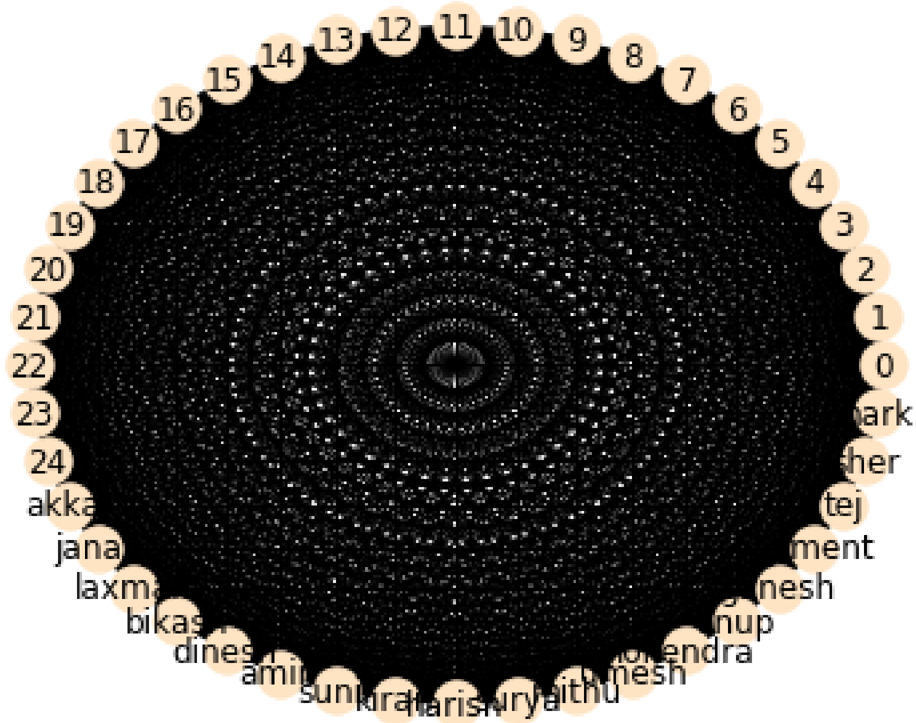
```
In [24]: nx.draw_circular(G)
```



```
In [25]: G = nx.complete_graph(G)
         nx.draw(G, node_color='bisque', with_labels=True)
```




```
In [26]: ego_G = nx.ego_graph(G, 5, radius=5)
         nx.draw_circular(G, node_color='bisque', with_labels=True)
```


11
10
12
9
13
8
14
7
15
6
16
5
17
4
18
3
19
2
20
1
21
0
22
23
24
tej

The Engineering World #DataScience 28 & 29

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 DIRECTED NETWORK ANALYSIS

1.1 Simulating Social Network(Directed Network Analysis)

```
In [1]: import numpy as np
        import pandas as pd
        from pylab import rcParams
        import seaborn as sb
        import matplotlib.pyplot as plt
        import networkx as nx
```

```
In [2]: %matplotlib inline
        rcParams ['figure.figsize'] = 5,4
        sb.set_style ('whitegrid')
```

1.1.1 Generating a grapg object and edgelist

```
In [3]: DG = nx.gn_graph(11, seed=25)
        for line in nx.generate_edgelist(DG, data=False):
            print (line)
```

```
1 0
2 0
3 2
4 3
5 0
6 4
7 3
8 0
9 8
10 1
```

```
In [4]: print (DG.node[0])
```

```
{}
```

```
In [5]: print (DG.node[5])
```

```
{}
```

1.1.2 Assigning attributes to nodes

```
In [6]: DG.node[0]['name']='Alice'
```

```
In [7]: print (DG.node[0])
```

```
{'name': 'Alice'}
```

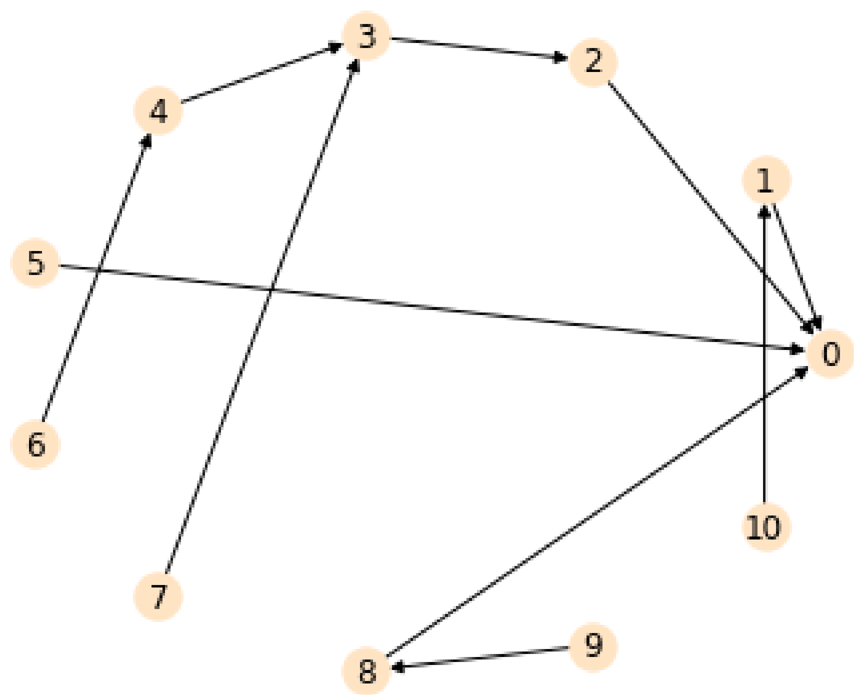
```
In [8]: DG.node[0]['name']='Alice'
        DG.node[1]['name']='Akkal'
        DG.node[2]['name']='Janak'
        DG.node[3]['name']='Laxman'
        DG.node[4]['name']='Bikash'
        DG.node[5]['name']='Dinesh'
        DG.node[6]['name']='Amin'
        DG.node[7]['name']='Sunil'
        DG.node[8]['name']='Kiran'
        DG.node[9]['name']='Surya'
```

```
In [9]: DG.add_nodes_from([(0,{'age':25}),(1,{'age':31}),(2,{'age':18}),(3,{'age':47}),(4,{'age'
        print(DG.node[1])
```

```
{'name': 'Akkal', 'age': 31}
```

```
In [10]: DG.node[0]['name']='M'
         DG.node[1]['name']='M'
         DG.node[2]['name']='F'
         DG.node[3]['name']='M'
         DG.node[4]['name']='F'
         DG.node[5]['name']='F'
         DG.node[6]['name']='M'
         DG.node[7]['name']='F'
         DG.node[8]['name']='M'
         DG.node[9]['name']='F'
```

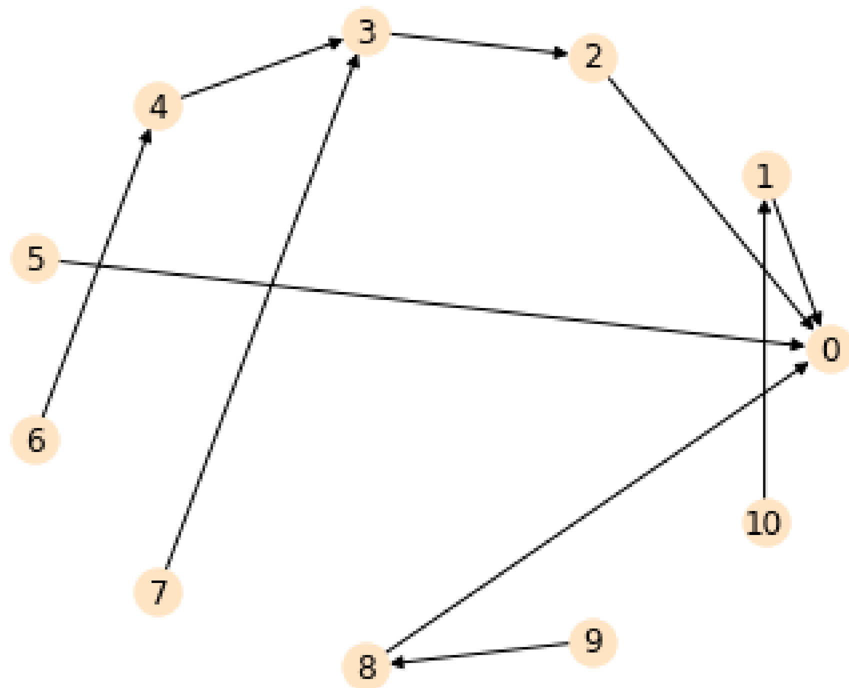
```
In [11]: nx.draw_circular(DG, node_color = 'bisque', with_labels = True)
```



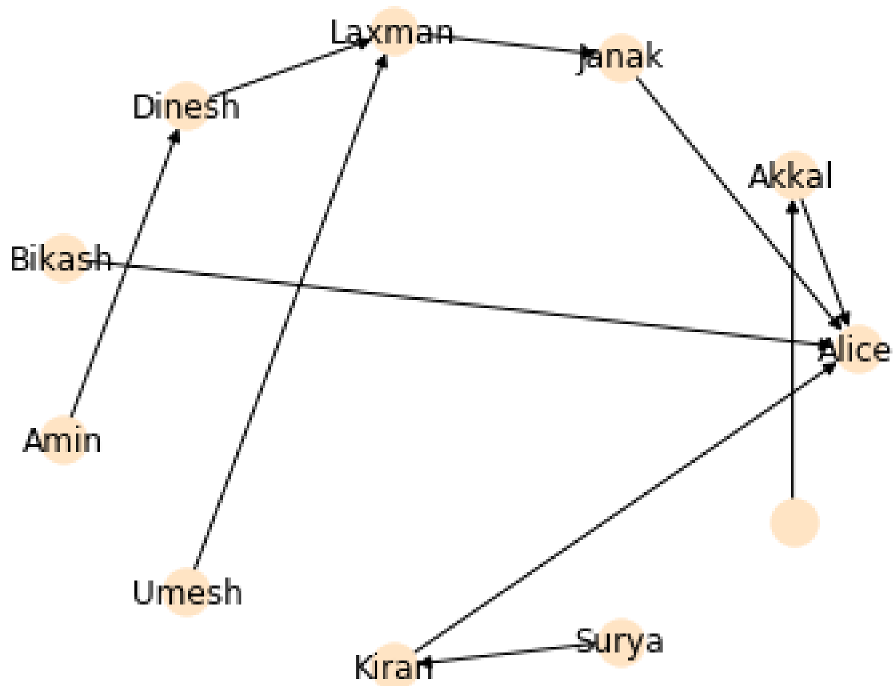
1.1.3 Visualize your network graph

```
In [12]: labeldict = {0:'Alice', 1:'Akkal', 2:'Janak', 3:'Laxman', 4:'Dinesh', 5:'Bikash', 6:'Am
```

```
In [13]: nx.draw_circular(DG, node_color = 'bisque', with_labels = True)
```

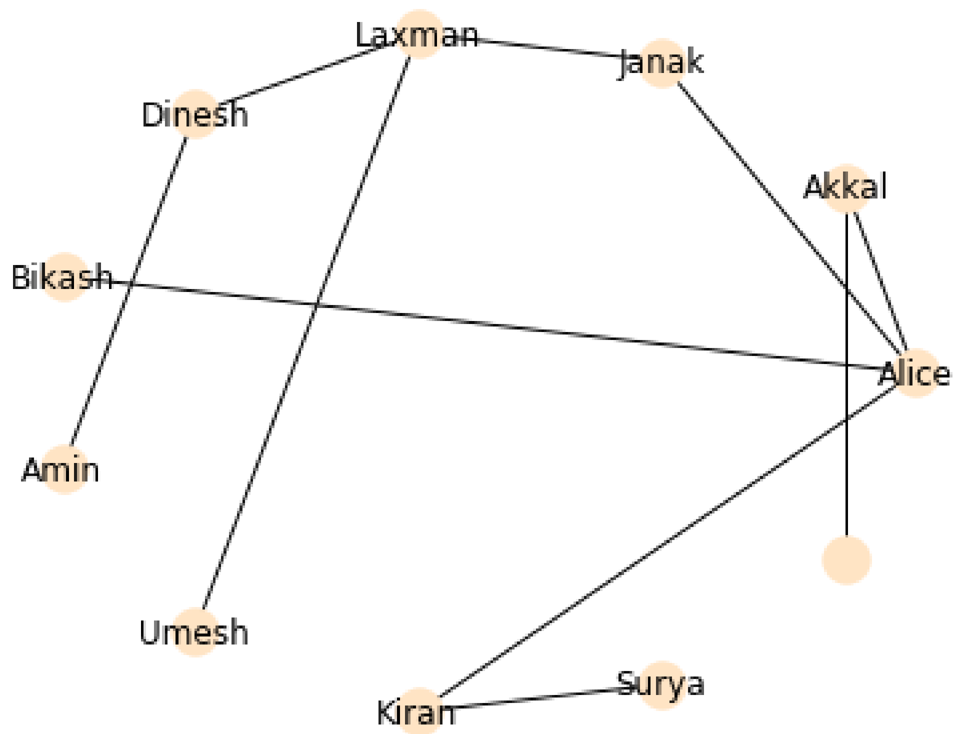



```
In [14]: nx.draw_circular(DG, labels = labeldict, node_color = 'bisque', with_labels = True) #ad
```




```
In [15]: G = DG.to_undirected()
```

```
In [16]: nx.draw_circular(G, labels = labeldict, node_color = 'bisque', with_labels = True)
```



2 NETWRK ANALYSIS GRAPH INSPECTION AND STATES ON NODES

2.0.1 Analyzing a Social Network

```
In [17]: DG = nx.gn_graph(7, seed=25)
         for line in nx.generate_edgelist(DG, data=False):
             print(line)

         DG.node[0]['name']='Alice'
         DG.node[1]['name']='Akkal'
         DG.node[2]['name']='Janak'
         DG.node[3]['name']='Laxman'
         DG.node[4]['name']='Bikash'
         DG.node[5]['name']='Dinesh'
         DG.node[6]['name']='Amin'
```

```
1 0
2 0
3 2
```

```
4 3
5 0
6 4
```

```
In [18]: G = DG.to_undirected()
```

```
In [19]: print (nx.info(DG))
```

```
Name:
Type: DiGraph
Number of nodes: 7
Number of edges: 6
Average in degree:   0.8571
Average out degree:   0.8571
```

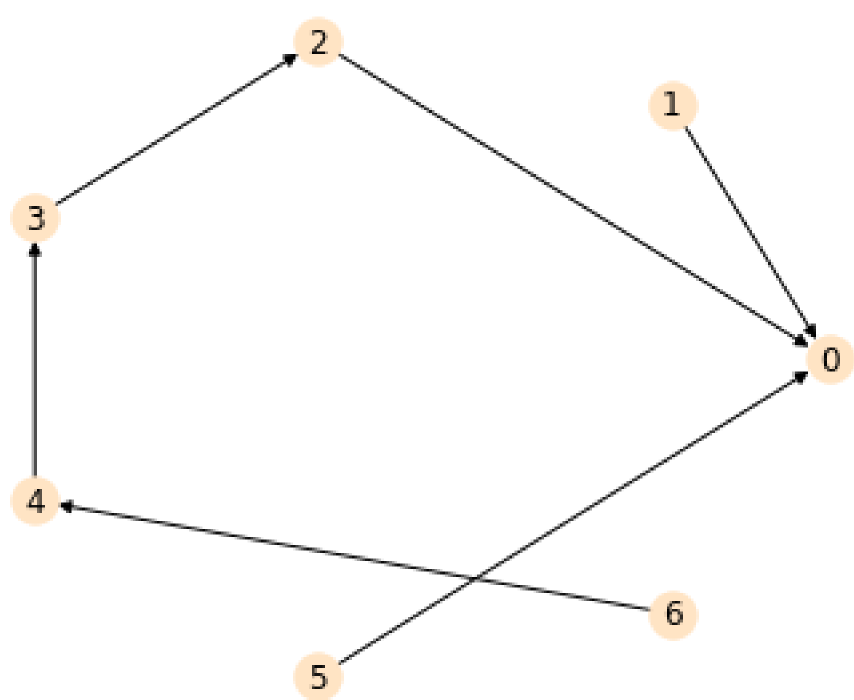
Considering degree in a social network

```
In [20]: DG.degree()
```

```
Out[20]: DiDegreeView({0: 3, 1: 1, 2: 2, 3: 2, 4: 2, 5: 1, 6: 1})
```

Identifying successor nodes

```
In [21]: nx.draw_circular(DG, node_color = 'bisque', with_labels = True) #add labels in graph pl
```



```
In [22]: DG.successors(3)

Out[22]: <dict_keyiterator at 0x7f77d5bf62c8>

In [23]: DG.neighbors(4)

Out[23]: <dict_keyiterator at 0x7f77d5be4318>

In [24]: G.neighbors(4)

Out[24]: <dict_keyiterator at 0x7f77d5be4b38>
```

The Engineering World #DataScience 30 & 31

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 LINEAR REGRESSION FOR MACHINE LEARNING

1.0.1 Linear Regression

```
In [1]: import numpy as np
        import pandas as pd
        from pylab import rcParams
        import seaborn as sb
        import matplotlib.pyplot as plt

        import sklearn
        from sklearn.linear_model import LinearRegression
        from sklearn.preprocessing import scale
        from collections import Counter
```

```
In [2]: %matplotlib inline
        rcParams ['figure.figsize'] = 5,4
        sb.set_style ('whitegrid')
```

1.0.2 (Multiple)Linear Regression on the enrollment data

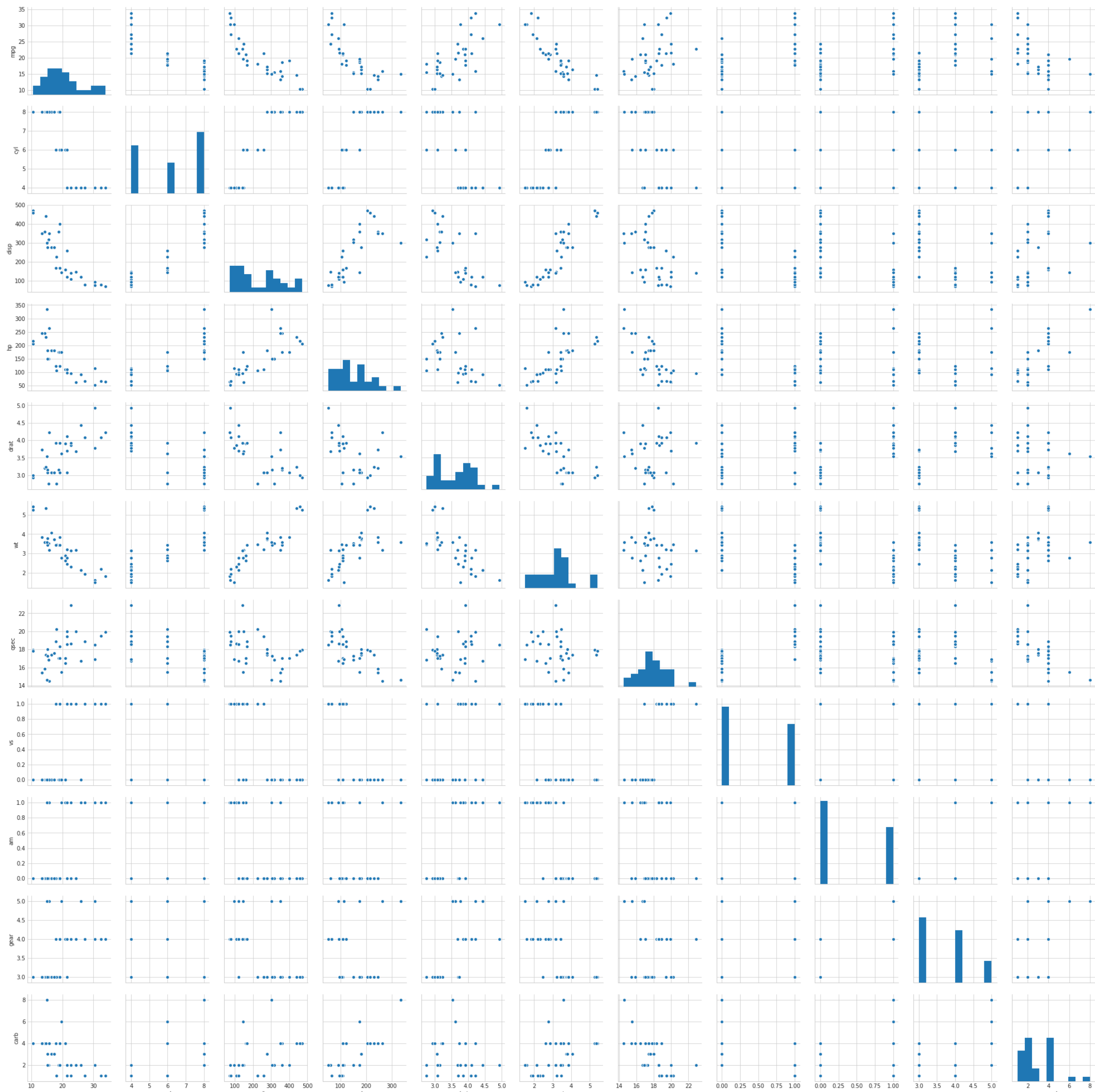
```
In [3]: address = 'mtcars.csv'
        cars = pd.read_csv(address)
        cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am
        cars.head()
```

```
Out[3]:            car_names   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
        0          Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
        1      Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
        2         Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
        3     Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
        4  Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3
```

```
   carb
0     4
1     4
2     1
3     1
4     2
```

```
In [4]: sb.pairplot(cars)
```

```
Out[4]: <seaborn.axisgrid.PairGrid at 0x7f1f98314c50>
```



```
In [5]: print(cars.corr())
```

```
           mpg       cyl      disp        hp      drat        wt      qsec  \
mpg   1.000000 -0.852162 -0.847551 -0.776168  0.681172 -0.867659  0.418684
cyl  -0.852162  1.000000  0.902033  0.832447 -0.699938  0.782496 -0.591242
disp -0.847551  0.902033  1.000000  0.790949 -0.710214  0.887980 -0.433698
hp   -0.776168  0.832447  0.790949  1.000000 -0.448759  0.658748 -0.708223
drat  0.681172 -0.699938 -0.710214 -0.448759  1.000000 -0.712441  0.091205
wt   -0.867659  0.782496  0.887980  0.658748 -0.712441  1.000000 -0.174716
qsec  0.418684 -0.591242 -0.433698 -0.708223  0.091205 -0.174716  1.000000
vs    0.664039 -0.810812 -0.710416 -0.723097  0.440278 -0.554916  0.744535
am    0.599832 -0.522607 -0.591227 -0.243204  0.712711 -0.692495 -0.229861
gear  0.480285 -0.492687 -0.555569 -0.125704  0.699610 -0.583287 -0.212682
carb -0.550925  0.526988  0.394977  0.749812 -0.090790  0.427606 -0.656249

            vs        am      gear      carb
mpg   0.664039  0.599832  0.480285 -0.550925
cyl  -0.810812 -0.522607 -0.492687  0.526988
disp -0.710416 -0.591227 -0.555569  0.394977
hp   -0.723097 -0.243204 -0.125704  0.749812
drat  0.440278  0.712711  0.699610 -0.090790
wt   -0.554916 -0.692495 -0.583287  0.427606
qsec  0.744535 -0.229861 -0.212682 -0.656249
vs    1.000000  0.168345  0.206023 -0.569607
am    0.168345  1.000000  0.794059  0.057534
gear  0.206023  0.794059  1.000000  0.274073
carb -0.569607  0.057534  0.274073  1.000000
```

In [6]:
```
cars_data = cars.ix[:,(2,3)].values
cars_target = cars.ix[:,1].values
cars_data_names = ['hp', 'am']
X,Y = scale(cars_data), cars_target
```

```
/home/akkal/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  """Entry point for launching an IPython kernel.
```

1.0.3 Checking for missing values

In [7]:
```
missing_values = X == np.NAN
X[missing_values == True]
```

Out[7]:
```
array([], dtype=float64)
```

```
In [8]: LinReg = LinearRegression(normalize=True)
        LinReg.fit(X,Y)
        print(LinReg.score(X,Y))

0.7595657755568964
```

2 LOGESTIC REGRESSION FOR MACHINE LEARNING

2.0.1 Logestic Regression

```
In [9]: import numpy as np
        import pandas as pd
        from pandas import Series, DataFrame
        from pylab import rcParams
        import scipy
        from scipy.stats import spearmanr
        import seaborn as sb
        import matplotlib.pyplot as plt

        import sklearn
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import train_test_split
        from sklearn import metrics
        from sklearn.preprocessing import scale
        from sklearn import preprocessing

In [10]: %matplotlib inline
         rcParams ['figure.figsize'] = 5,4
         sb.set_style ('whitegrid')
```

2.0.2 Logestic regression on mtcars

```
In [11]: address = 'mtcars.csv'
         cars = pd.read_csv(address)
         cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'a
         cars.head()

Out[11]:             car_names   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
         0           Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
         1       Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
         2          Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
         3      Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
         4   Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

            carb
         0     4
         1     4
```

```
2     1
3     1
4     2
```

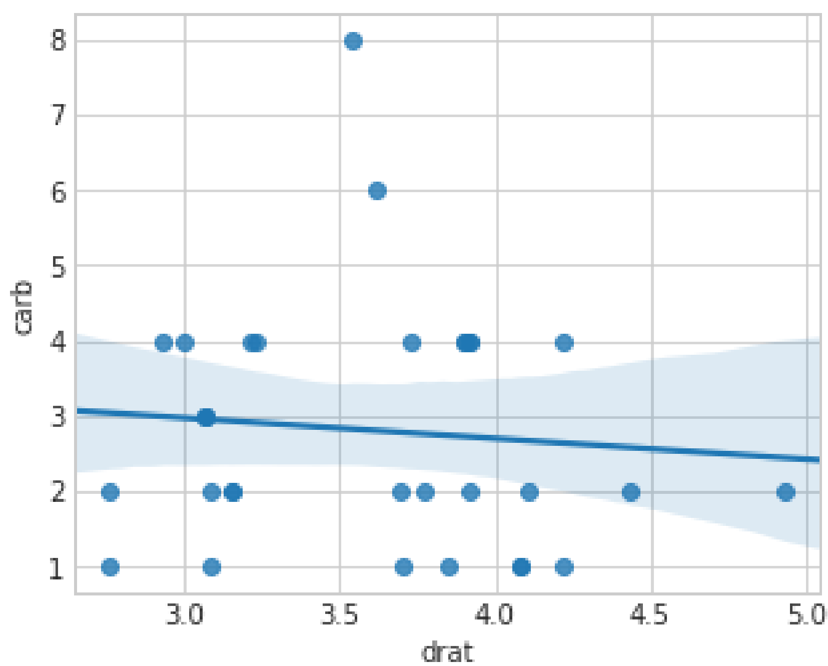
```
In [12]: cars_data = cars.ix[:,(5,11)].values
         cars_data_names = ['drat', 'carb']

         y = cars.ix[:,9].values
```

```
/home/akkal/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  """Entry point for launching an IPython kernel.
```

2.0.3 Checking for independence between features

```
In [13]: sb.regplot(x = 'drat', y = 'carb', data = cars, scatter=True)
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f5a0eea20>
```



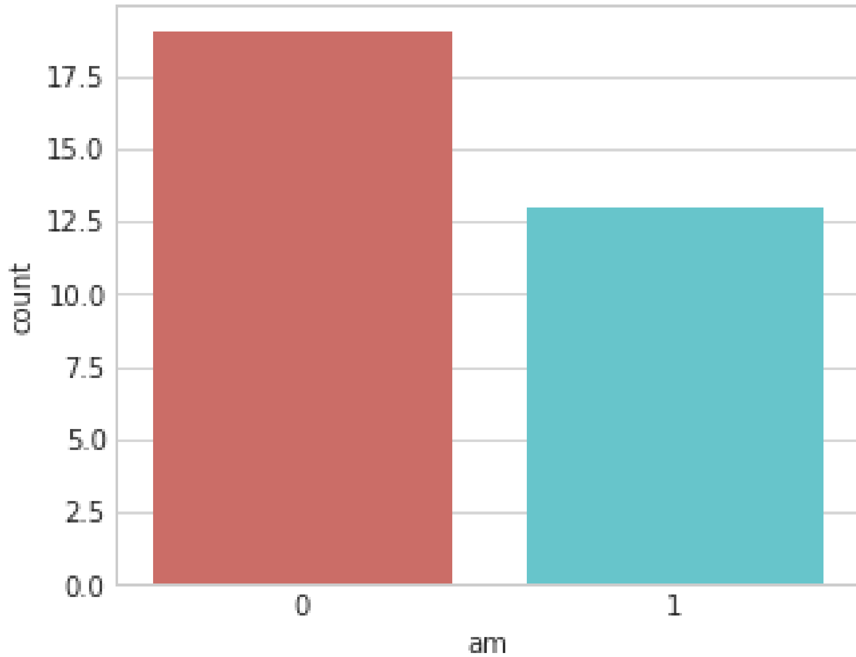
```
In [14]: drat = cars['drat']
         carb = cars['carb']
         spearmanr_coefficient, p_value = spearmanr(drat, carb)
         print('spearmanr Rand Correlation Coefficient %0.3f' % (spearmanr_coefficient))

spearmanr Rand Correlation Coefficient -0.125
```

2.0.4 Checking for missing values

```
In [15]: cars.isnull().sum()

Out[15]: car_names    0
         mpg          0
         cyl          0
         disp         0
         hp           0
         drat         0
         wt           0
         qsec         0
         vs           0
         am           0
         gear         0
         carb         0
         dtype: int64
```

2.0.5 Checking that your binary or ordinal

```
In [16]: sb.countplot(x = 'am', data = cars, palette='hls')

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f59bf3828>
```



2.0.6 Checking that yur data size is sufficient

```
In [17]: cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
car_names    32 non-null object
mpg          32 non-null float64
cyl          32 non-null int64
disp         32 non-null float64
hp           32 non-null int64
drat         32 non-null float64
wt           32 non-null float64
qsec         32 non-null float64
vs           32 non-null int64
am           32 non-null int64
gear         32 non-null int64
carb         32 non-null int64
dtypes: float64(5), int64(6), object(1)
memory usage: 3.1+ KB
```

2.0.7 Deploying and evaluating yur model

```
In [18]: X = scale(cars_data)
```

```
In [19]: LogReg = LogisticRegression()
         LogReg.fit(X,y)
         print(LogReg.score(X,y))
```

```
0.8125
```

```
In [20]: y_pred = LogReg.predict(X)
         from sklearn.metrics import classification_report
         print (classification_report(y,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.88      0.79      0.83        19
          1       0.73      0.85      0.79        13

avg / total       0.82      0.81      0.81        32
```

The Engineering World #DataScience 32

May 31, 2018

AKKAL BAHADUR BIST
DATA SCIENTIST AT
KATHMANDU INSTITUTE OF APPLIED SCIENCES (KIAS)
Center for Conservation Biology (CCB)

1 NAIVE BAYES CLASSIFIERS

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

import sklearn
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

```
/home/akkal/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWar
  "This module will be removed in 0.20.", DeprecationWarning)
```

1.1 Naive Bayes

1.1.1 Using naive bayes to predict spam