HO-2

November 20, 2018

1 HO-2 Report

1.0.1 Irving Hernández Gallegos

1.0.2 Objective

Apply descriptive statistics to explore data collections and compute quantitative descriptions. To describe the sample data and to be able to infer any conclusion, we should go through several steps: data preparation and descriptive analytics. http://vargas-solar.com/data-centric-smart-everything/hands-on/exploring-data-collections-using-descriptive-statistics/

1.1 Playing with the Education and training data

Let us consider a public database called the "Adult" dataset, hosted on the UCI's Machine Learning Repository. It contains approximately 32,000 observations concerning different financial parameters related to the US population: age, sex, marital (marital status of the individual), country, income (Boolean variable: whether the person makes more than $50,000 per annum), education (the highest level of education achieved by the individual), occupation, capital gain, etc.

We will show that we can explore the data by asking questions like: "Are men more likely to become high-income professionals than women, i.e., to receive an income of over $50,000 per annum?"

```
In [1]: file = open('files/adult.data', 'r')
```

```
In [2]: def file_read_from_head(fname, nlines):
                from itertools import islice
                with open(fname) as f:
                        for line in islice(f, nlines):
                                print(line)
        file_read_from_head('files/adult.data',3)
```

```
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 21

50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White,

38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners, Not-in-family, White, Male, 0, 0,
```

```
In [3]: def chr_int(a):
            if a.isdigit():
                return int(a)
            else:
                return 0

        data=[]
        for line in file:
            data1=line.split(', ')
            if len(data1)==15:
                data.append([chr_int(data1[0]),data1[1],chr_int(data1[2]),data1[3],chr_int(data1
                    data1[7],data1[8],data1[9],chr_int(data1[10]),chr_int(data1[11]),chr_int(dat
                    data1[14]])
```

```
In [4]: print (data[0:3])
```

```
[[39, 'State-gov', 77516, 'Bachelors', 13, 'Never-married', 'Adm-clerical', 'Not-in-family', 'Wh
```

Q1. What is the obtained result? What did you ask for in the previous command? Explain It reads the file and creates a two dimensional list with the items in its content, later we print rows in the given range

```
In [5]: %matplotlib inline
        import pandas as pd

        df = pd.DataFrame(data) #  Two-dimensional size-mutable, potentially heterogeneous tabul

        df.columns = ['age', 'type_employer', 'fnlwgt', 'education',
                        "education_num","marital", "occupation", "relationship", "race","sex",
                        "capital_gain", "capital_loss", "hr_per_week","country","income"]
        df.head()
```

```
Out[5]:    age     type_employer  fnlwgt  education  education_num  \
        0   39          State-gov   77516  Bachelors             13
        1   50  Self-emp-not-inc   83311  Bachelors             13
        2   38           Private  215646    HS-grad              9
        3   53           Private  234721       11th              7
        4   28           Private  338409  Bachelors             13

                      marital         occupation   relationship   race     sex  \
        0       Never-married       Adm-clerical  Not-in-family  White    Male
        1  Married-civ-spouse    Exec-managerial        Husband  White    Male
        2            Divorced  Handlers-cleaners  Not-in-family  White    Male
        3  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
        4  Married-civ-spouse     Prof-specialty           Wife  Black  Female

           capital_gain  capital_loss  hr_per_week        country   income
```

```
0          2174            0          40  United-States  <=50K\n
1             0            0          13  United-States  <=50K\n
2             0            0          40  United-States  <=50K\n
3             0            0          40  United-States  <=50K\n
4             0            0          40           Cuba  <=50K\n
```

Q2. Describe and explain the result. Using pandas library, list data was translated into a dataframe, header names were added manually.

```
In [6]: df.tail()
```

```
Out[6]:        age type_employer  fnlwgt   education  education_num  \
        32556   27       Private  257302  Assoc-acdm             12
        32557   40       Private  154374     HS-grad              9
        32558   58       Private  151910     HS-grad              9
        32559   22       Private  201490     HS-grad              9
        32560   52  Self-emp-inc  287927     HS-grad              9

                          marital         occupation relationship   race     sex  \
        32556  Married-civ-spouse       Tech-support         Wife  White  Female
        32557  Married-civ-spouse  Machine-op-inspct      Husband  White    Male
        32558             Widowed       Adm-clerical    Unmarried  White  Female
        32559       Never-married       Adm-clerical    Own-child  White    Male
        32560  Married-civ-spouse    Exec-managerial         Wife  White  Female

               capital_gain  capital_loss  hr_per_week        country   income
        32556             0             0           38  United-States  <=50K\n
        32557             0             0           40  United-States   >50K\n
        32558             0             0           40  United-States  <=50K\n
        32559             0             0           20  United-States  <=50K\n
        32560         15024             0           40  United-States   >50K\n
```

Q3. Describe and explain the result. Compare with the previous one. Applying head and tail function we disply first 5 items and last 5 items respectively in dataframe.

```
In [7]: df.shape
```

```
Out[7]: (32561, 15)
```

Q4. Describe an explain the result. Shape attribute returns number of rows and columns in dataframe

```
In [8]: counts = df.groupby('country').size()

        print (counts)
        # also: df.outcome.value_counts()
```

```
country
?                              583
Cambodia                        19
Canada                         121
China                           75
Columbia                        59
Cuba                            95
Dominican-Republic              70
Ecuador                         28
El-Salvador                    106
England                         90
France                          29
Germany                        137
Greece                          29
Guatemala                       64
Haiti                           44
Holand-Netherlands               1
Honduras                        13
Hong                            20
Hungary                         13
India                          100
Iran                            43
Ireland                         24
Italy                           73
Jamaica                         81
Japan                           62
Laos                            18
Mexico                         643
Nicaragua                       34
Outlying-US(Guam-USVI-etc)      14
Peru                            31
Philippines                    198
Poland                          60
Portugal                        37
Puerto-Rico                    114
Scotland                        12
South                           80
Taiwan                          51
Thailand                        18
Trinadad&Tobago                 19
United-States                29170
Vietnam                         67
Yugoslavia                      16
dtype: int64
```

Q.5 How many items are there for USA? and for Mexico? Mexico: 643 USA: 29170

```
In [9]: counts = df.groupby('age').size() # grouping by age
        print (counts)

        srt = counts.sort_values(ascending=True)
        srt.tail(1)
```

```
age
17    395
18    550
19    712
20    753
21    720
22    765
23    877
24    798
25    841
26    785
27    835
28    867
29    813
30    861
31    888
32    828
33    875
34    886
35    876
36    898
37    858
38    827
39    816
40    794
41    808
42    780
43    770
44    724
45    734
46    737
     ...
60    312
61    300
62    258
63    230
64    208
65    178
66    150
67    151
68    120
69    108
```

```
70     89
71     72
72     67
73     64
74     51
75     45
76     46
77     29
78     23
79     22
80     22
81     20
82     12
83      6
84     10
85      3
86      1
87      1
88      3
90     43
Length: 73, dtype: int64
```

```
Out[9]: age
        36    898
        dtype: int64
```

6.What is the age of the most represented people? people who is 36 is the largest group

-- Let us split people according to their gender into two groups: men and women. If we focus on high-income professionals separated by sex, we can do:

```
In [10]: ml = df[(df.sex == 'Male')] # grouping by sex
         ml.shape
         ml1 = df[(df.sex == 'Male')&(df.income=='>50K\n')]
         ml1.shape
```

```
Out[10]: (6662, 15)
```

```
In [11]: fm =df[(df.sex == 'Female')]
         fm.shape
         fm1 =df[(df.sex == 'Female')&(df.income=='>50K\n')]
         fm1.shape
```

```
Out[11]: (1179, 15)
```

```
In [12]: df1=df[(df.income=='>50K\n')]
```

```
print ('The rate of people with high income is: ', int(len(df1)/float(len(df))*100), '%
print ('The rate of men with high income is: ', int(len(ml1)/float(len(ml))*100), '%.'
print ('The rate of women with high income is: ', int(len(fm1)/float(len(fm))*100), '%.
```

```
The rate of people with high income is:  24 %.
The rate of men with high income is:  30 %.
The rate of women with high income is:  10 %.
```

Q7. Describe an explain the result. New dataframes were created for every subset based on sex and income higher than 50K per year. In order to answer the initial question: *"Are men more likely to become high-income professionals than women, i.e., to receive an income of over $50,000 per annum?"* Percentage of people (any sex), male and female whose income is higher than 50K was calculated. Figures show that male group is more likely to earn higher income than female by 20 percentage points.

Quantitative: exploratory data analysis is a way to make preliminary assessments about the population distribution of the variable. The characteristics of the population distribution of a quantitative variable are its mean, deviation, histograms,outliers, etc.

```
In [13]: print ('The average age of men is: ', ml['age'].mean(), '.' )
         print ('The average age of women is: ', fm['age'].mean(), '.')
```

```
The average age of men is:  39.43354749885268 .
The average age of women is:  36.85823043357163 .
```

```
In [14]: print ('The average age of high-income men is: ', ml1['age'].mean(), '.' )
         print ('The average age of high-income women is: ', fm1['age'].mean(), '.')
```

```
The average age of high-income men is:  44.62578805163614 .
The average age of high-income women is:  42.125530110262936 .
```

Q8. Describe and explain the result. Mean function was applied to age column from male/female dataframes. The result show average income from every group mentioned above. We can notice that high income male population earn more than female population, while average men age is higher in whole population sample

Sample population and Whole population analytics

```
In [15]: ml_mu = ml['age'].mean()
         fm_mu = fm['age'].mean()
         ml_var = ml['age'].var()
         fm_var = fm['age'].var()
         ml_std = ml['age'].std()
         fm_std = fm['age'].std()

         print ('Statistics of age for men: mu:', ml_mu, 'var:', ml_var, 'std:', ml_std)
         print ('Statistics of age for women: mu:', fm_mu, 'var:', fm_var, 'std:', fm_std)
```

```
Statistics of age for men: mu: 39.43354749885268 var: 178.77375174530096 std: 13.37063019252649
Statistics of age for women: mu: 36.85823043357163 var: 196.3837063948037 std: 14.01369709943824
```

```
In [16]: ml_mu_hr = ml['hr_per_week'].mean()
         fm_mu_hr = fm['hr_per_week'].mean()
         ml_var_hr = ml['hr_per_week'].var()
         fm_var_hr = fm['hr_per_week'].var()
         ml_std_hr = ml['hr_per_week'].std()
         fm_std_hr = fm['hr_per_week'].std()

         print ('Statistics of hours per week for men: mu:', ml_mu_hr, 'var:', ml_var_hr, 'std:'
         print ('Statistics of hours per week for women: mu:', fm_mu_hr, 'var:', fm_var_hr, 'std
```

```
Statistics of hours per week for men: mu: 42.42808627810923 var: 146.88846717142022 std: 12.1197
Statistics of hours per week for women: mu: 36.410361154953115 var: 139.50679700047252 std: 11.8
```

Q9. Describe an explain the result. Additionally to mean value we can get variance and standar deviation from population samples. Male population work significally more hours per week than its female counterpart. Also, work hours are more spread in male population.

```
In [17]: ml_median= ml['age'].median()
         fm_median= fm['age'].median()

         print ("Median age per men and women: ", ml_median, fm_median)
```

```
Median age per men and women:  38.0 35.0
```

```
In [18]: ml_median_age= ml1['age'].median()
         fm_median_age= fm1['age'].median()

         print ("Median age per men and women with high-income: ", ml_median_age, fm_median_age)
```

```
Median age per men and women with high-income:  44.0 41.0
```

```
In [19]: ml_median_hr= ml['hr_per_week'].median()
         fm_median_hr= fm['hr_per_week'].median()
         print ("Median hours per week per men and women: ", ml_median_hr, fm_median_hr)
```

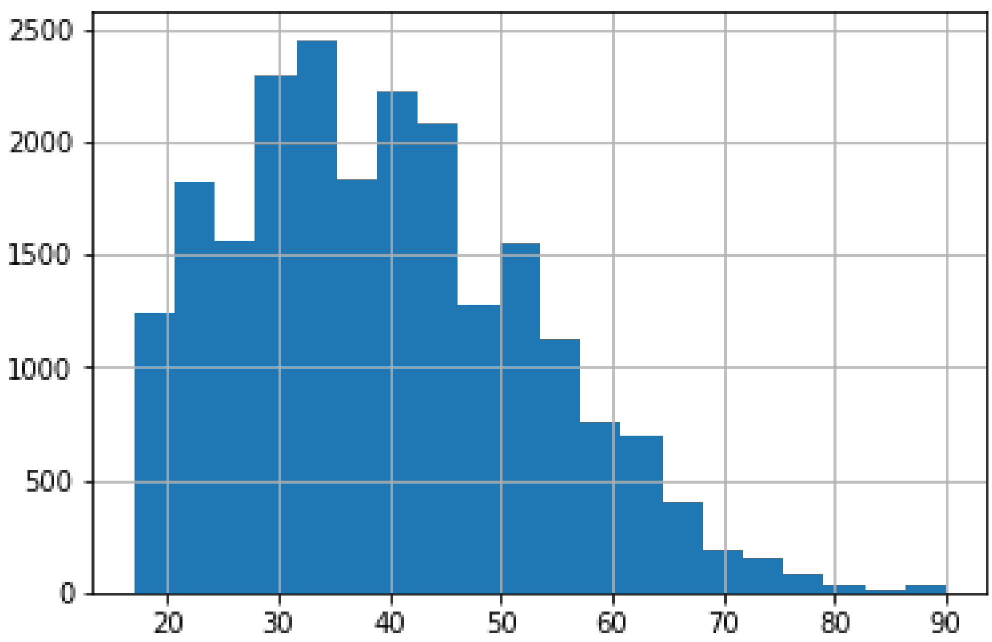
```
Median hours per week per men and women:  40.0 40.0
```

Q10. Describe an explain the result. Interestingly we can see that median work hour for both sexes are the same; 40hrs.

```
In [20]: import matplotlib.pyplot as plt
         ml_age=ml['age']
         ml_age.hist(normed=0, histtype='stepfilled', bins=20)
```

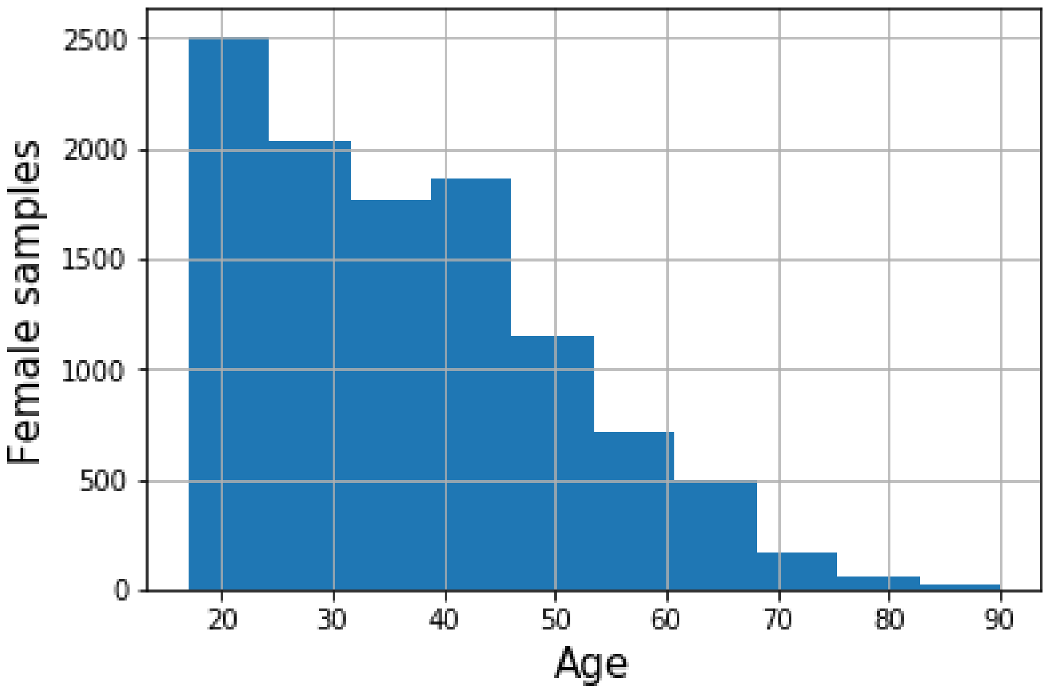
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1906b782b0>
```



Q10.1 Show the graphics and and explain the result. Above plot shows an age histogram for male population, we can see a bell like distribution with where density is inclined towards younger ages.

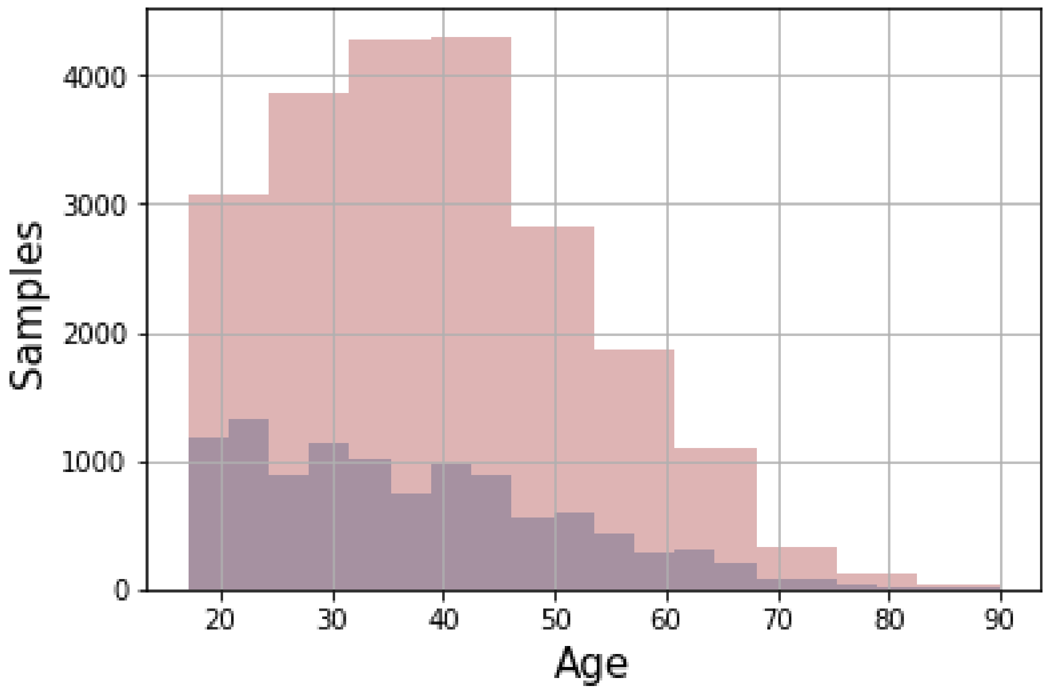
```
In [21]: fm_age=fm['age']
         fm_age.hist(normed=0, histtype='stepfilled', bins=10)
         plt.xlabel('Age',fontsize=15)
         plt.ylabel('Female samples',fontsize=15)
         plt.show()
```



Q11. Show the graphics and an explain the result. Similarly to male's plot, there's a tendency for younger ages nevertheless this plot can be described by a slope function

```
In [22]: import seaborn as sns
         fm_age.hist(normed=0, histtype='stepfilled', alpha=.5, bins=20)   # default number of b
         ml_age.hist(normed=0, histtype='stepfilled', alpha=.5, color=sns.desaturate("indianred"
         plt.xlabel('Age',fontsize=15)
         plt.ylabel('Samples',fontsize=15)
         plt.show()
```

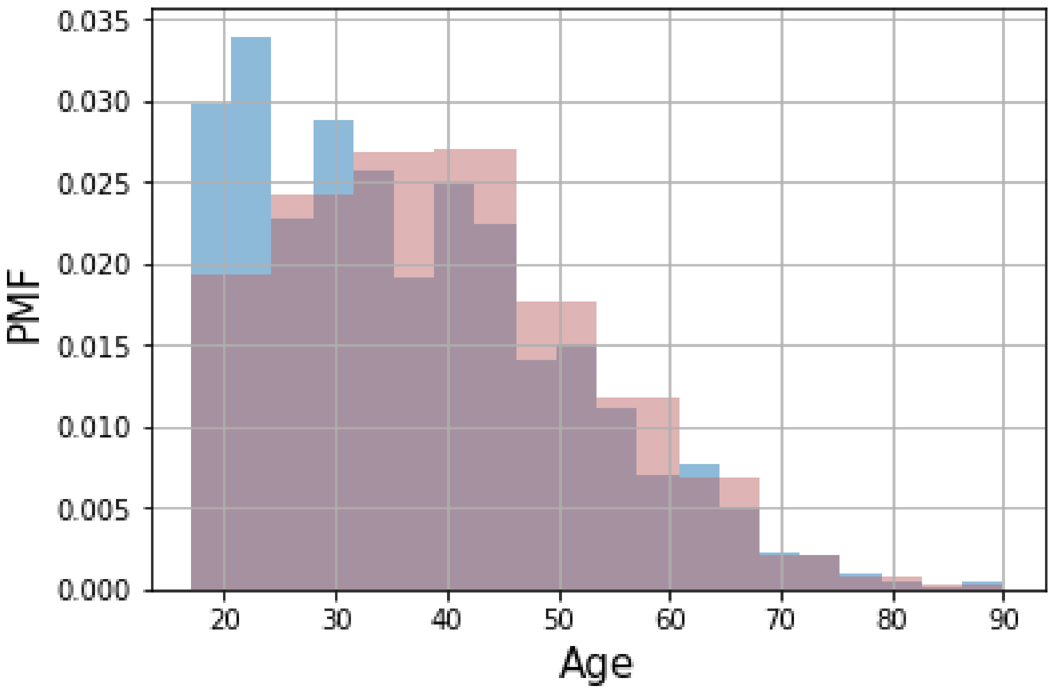
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



Q12. Show the graphics and an explain the result. Above plot shows a normalized histogram called Probability Mass Function (PMF) which is obtained by dividing/normalizing by n, the number of samples.

```
In [23]: fm_age.hist(normed=1, histtype='stepfilled', alpha=.5, bins=20)   # default number of b
         ml_age.hist(normed=1, histtype='stepfilled', alpha=.5, color=sns.desaturate("indianred"
         plt.xlabel('Age',fontsize=15)
         plt.ylabel('PMF',fontsize=15)
         plt.show()
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```

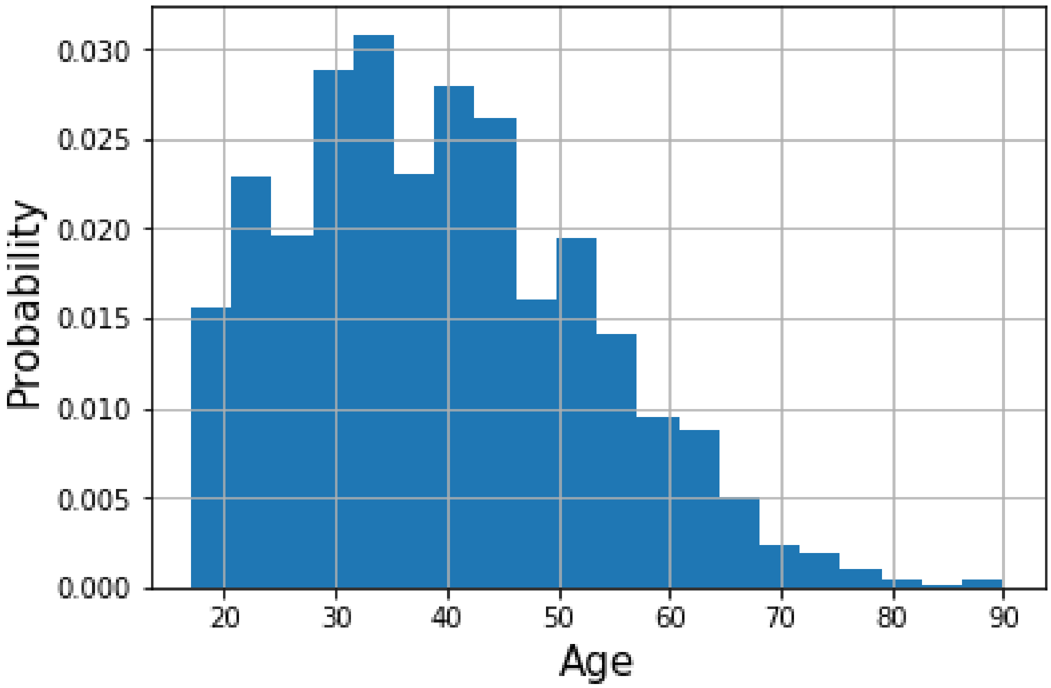


Q13. Show the graphics and an explain the result. Cumulative Distribution Function (CDF), describes the probability that a real-valued random variable X with a given proba- bility distribution will be found to have a value less than or equal to x. Above plot shows both CDF for male (red) and female (blue)

1.1.1 Data Distributions

```
In [24]: ml_age.hist(normed=1, histtype='stepfilled', bins=20)

         plt.xlabel('Age',fontsize=15)
         plt.ylabel('Probability',fontsize=15)
         plt.show()
```

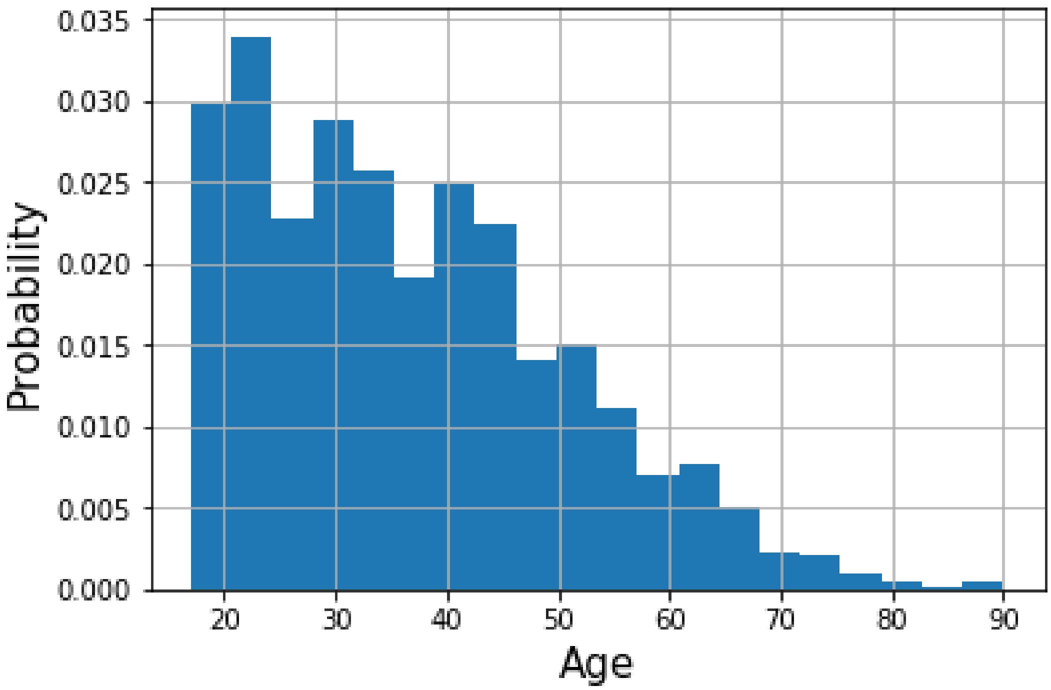
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



Q14. Show the graphics and explain the results *Probability Mass Function (PMF) for male*; Histogram was normalized by n=1 where n is the number of samples

```
In [25]: fm_age.hist(normed=1, histtype='stepfilled', bins=20)

         plt.xlabel('Age',fontsize=15)
         plt.ylabel('Probability',fontsize=15)
         plt.show()
```

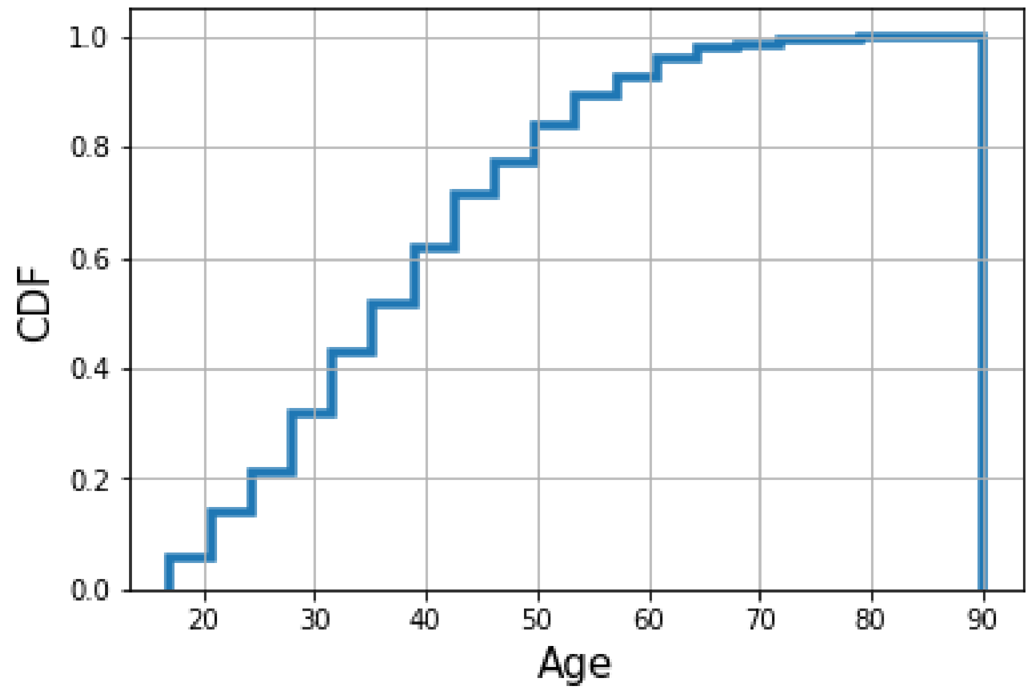
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



Q15. Show the graphics and explain the results *Probability Mass Function (PMF) for female*; Histogram was normalized by n=1 where n is the number of samples

```
In [26]: ml_age.hist(normed=1, histtype='step', cumulative=True, linewidth=3.5, bins=20)

         plt.xlabel('Age',fontsize=15)
         plt.ylabel('CDF',fontsize=15)
         plt.show()
```

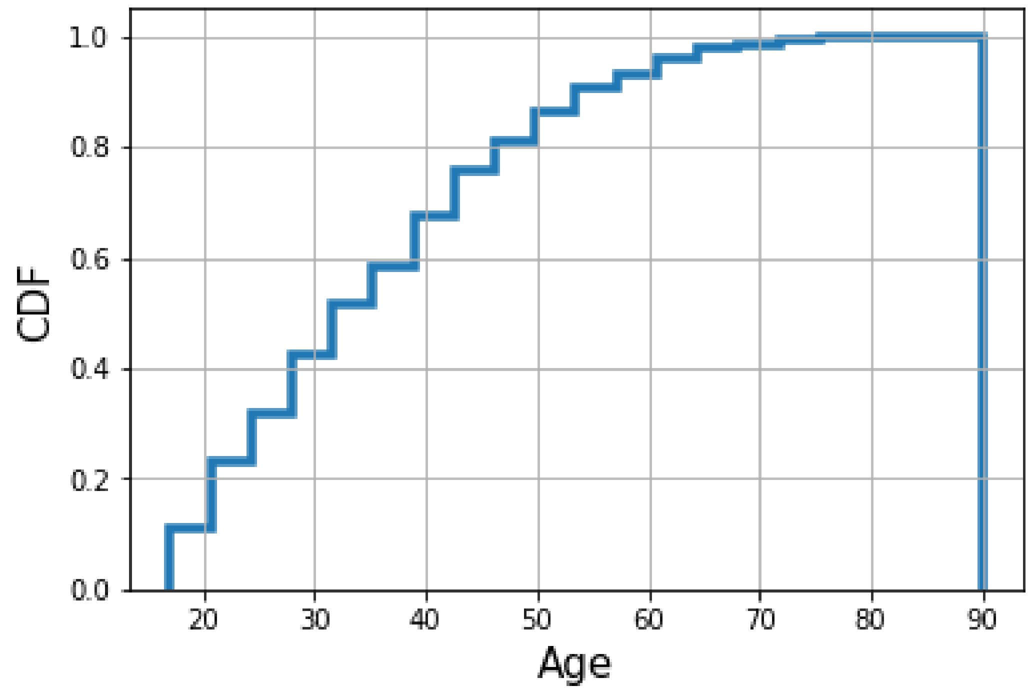
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



Q16. Show the graphics and an explain the result. Cumulative Distribution Function (CDF) is shown. Probability that a real-valued random variable X with a given probability distribution will be found to have a value less than or equal to x

```
In [27]: fm_age.hist(normed=1, histtype='step', cumulative=True, linewidth=3.5, bins=20)

         plt.xlabel('Age',fontsize=15)
         plt.ylabel('CDF',fontsize=15)
         plt.show()
```

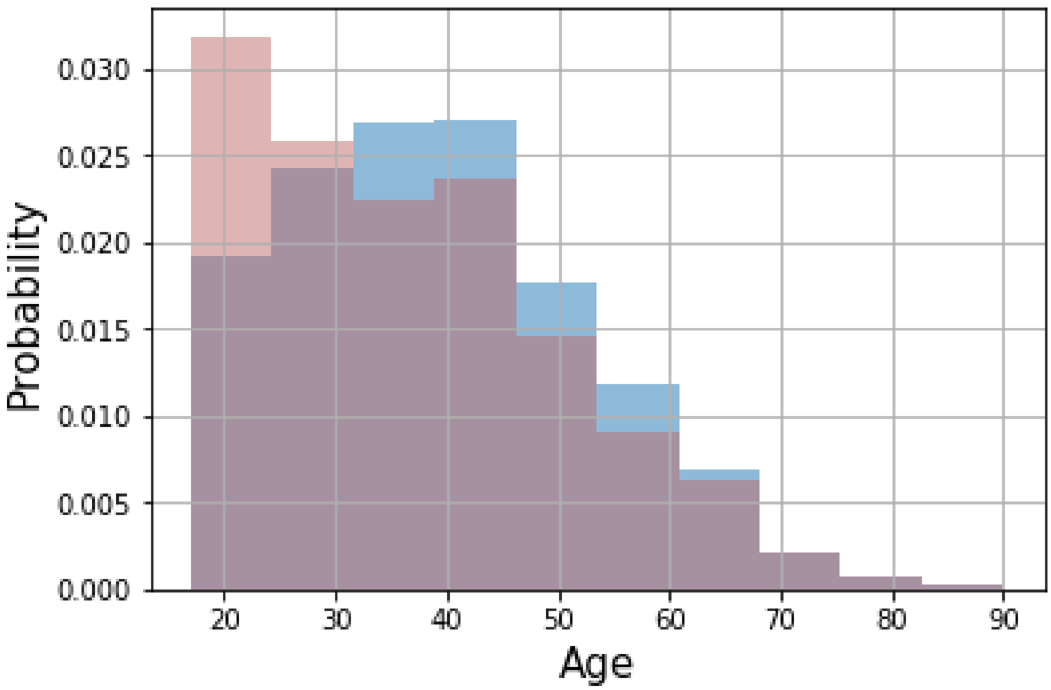
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



Q17. Show the graphics and an explain the result. Cumulative Distribution Function (CDF) for females. The main difference we can notice against the male CDF is that the slop rises faster.

```
In [28]: ml_age.hist(bins=10, normed=1, histtype='stepfilled', alpha=.5)    # default number of b
         fm_age.hist(bins=10, normed=1, histtype='stepfilled', alpha=.5, color=sns.desaturate("i
         plt.xlabel('Age',fontsize=15)
         plt.ylabel('Probability',fontsize=15)
         plt.show()
```

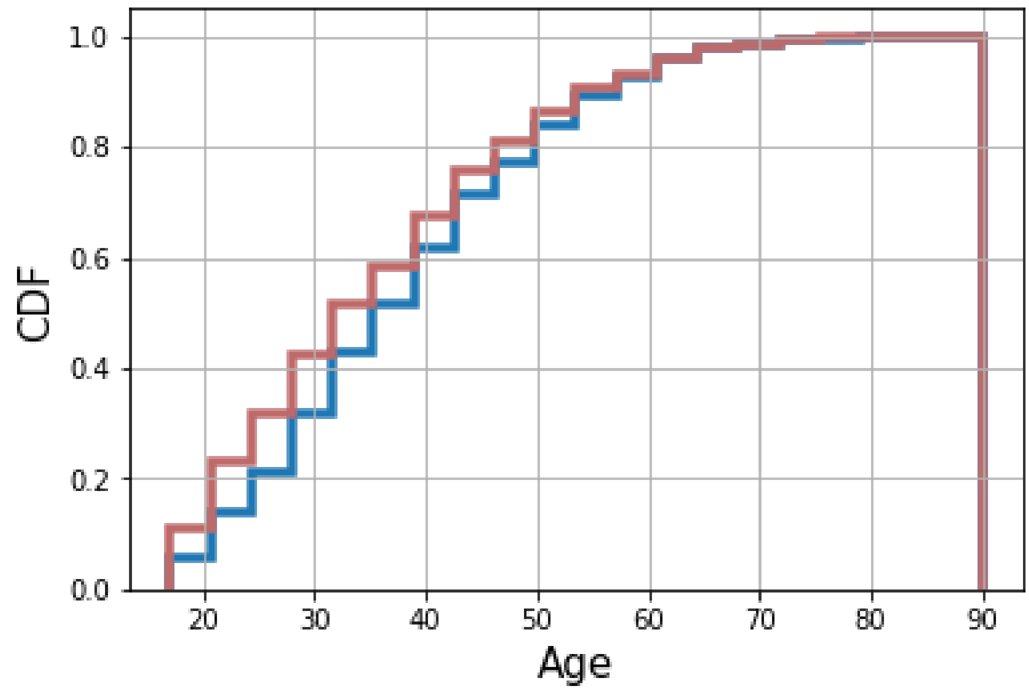
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



Q18. Show the graphics and an explain the results Both histogram plots can be merged and be differenciated by color

```
In [29]: ml_age.hist(normed=1, histtype='step', cumulative=True,  linewidth=3.5, bins=20)
         fm_age.hist(normed=1, histtype='step', cumulative=True,  linewidth=3.5, bins=20, color=
         plt.xlabel('Age',fontsize=15)
         plt.ylabel('CDF',fontsize=15)
         plt.show()
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



Q19. Show the graphics and an explain the result Both comulative Distribution Funtion plots are shown together

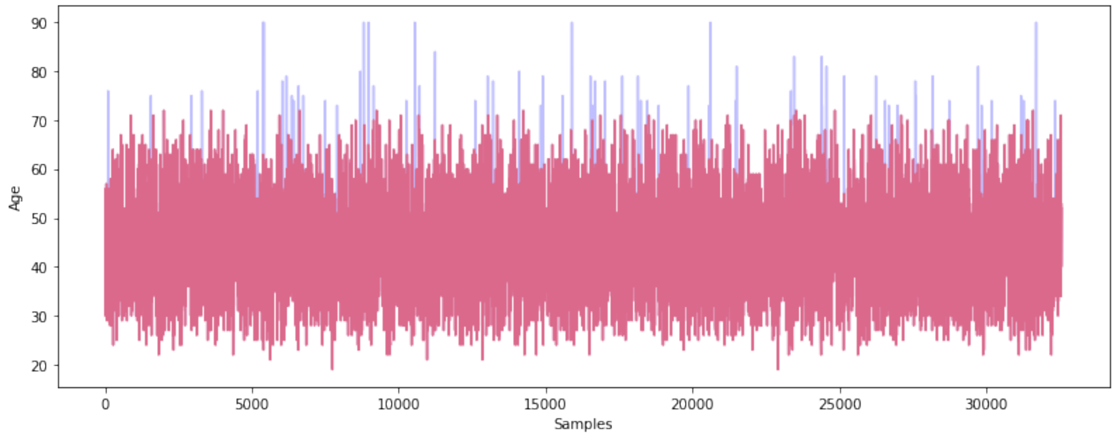
```
In [30]: print ("The mean sample difference is ", ml_age.mean() - fm_age.mean())

The mean sample difference is  2.5753170652810553
```

Q20. Explain the result. Male mean age is 2.57 years older than females in a working place. This matches the distribution results, which female histogram was more populated towards the younger side of the plot

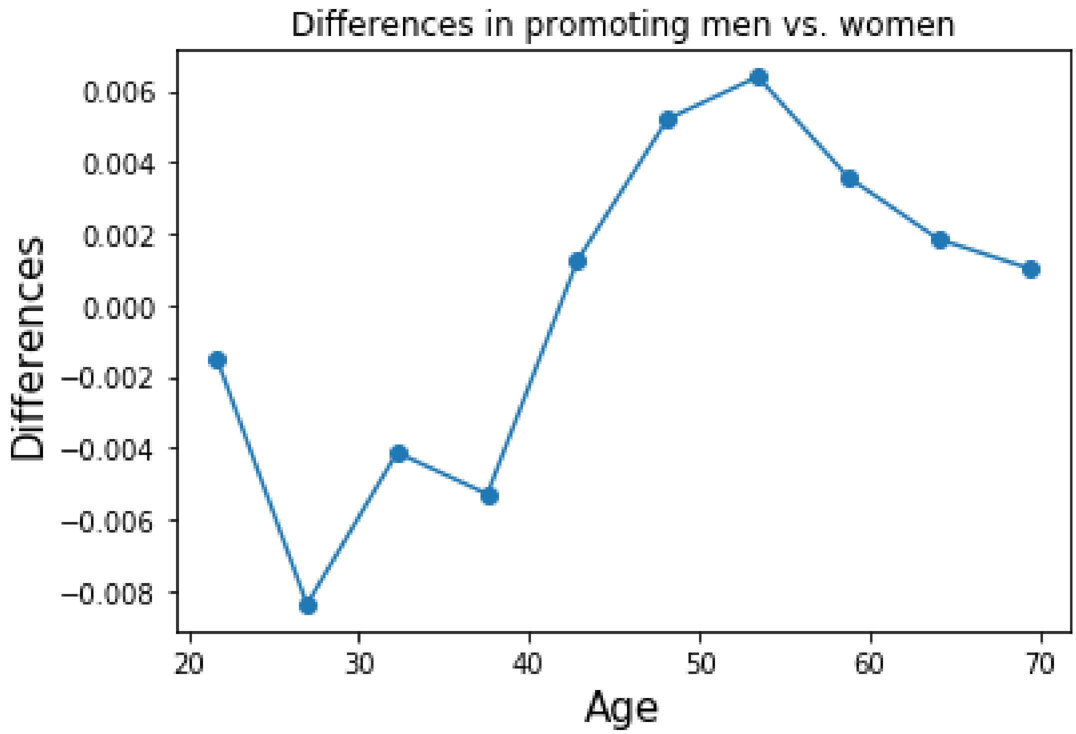
1.1.2 Examples of Outliners

```
In [31]: df['age'].median()

Out[31]: 37.0

In [32]: len(df[(df.income == '>50K\n') & (df['age'] < df['age'].median() - 15)])

Out[32]: 5

In [33]: len(df[(df.income == '>50K\n') & (df['age'] > df['age'].median() + 35)])

Out[33]: 69
```

```
In [34]: df2 = df.drop(df.index[(df.income=='>50K\n') & (df['age']>df['age'].median() +35) & (df

         df2.shape

Out[34]: (32492, 15)

In [35]: ml1_age=ml1['age']
         fm1_age=fm1['age']

In [36]: ml2_age = ml1_age.drop(ml1_age.index[(ml1_age >df['age'].median()+35) & (ml1_age>df['ag
         fm2_age = fm1_age.drop(fm1_age.index[(fm1_age > df['age'].median()+35) & (fm1_age > df[

In [37]: mu2ml = ml2_age.mean()
         std2ml = ml2_age.std()
         md2ml = ml2_age.median()

         # Computing the mean, std, median, min and max for the high-income male population

         print ("Men statistics: Mean:", mu2ml, "Std:", std2ml, "Median:", md2ml, "Min:", ml2_ag

Men statistics: Mean: 44.317982123920615 Std: 10.019749857171412 Median: 44.0 Min: 19 Max: 72


In [38]: mu3ml = fm2_age.mean()
         std3ml = fm2_age.std()
         md3ml = fm2_age.median()

         # Computing the mean, std, median, min and max for the high-income female population
         print ("Women statistics: Mean:", mu2ml, "Std:", std2ml, "Median:", md2ml, "Min:", fm2_

Women statistics: Mean: 44.317982123920615 Std: 10.019749857171412 Median: 44.0 Min: 19 Max: 72


In [39]: print ('The mean difference with outliers is: %4.2f.'% (ml_age.mean() - fm_age.mean()))
         print ("The mean difference without outliers is: %4.2f."% (ml2_age.mean() - fm2_age.mea

The mean difference with outliers is: 2.58.
The mean difference without outliers is: 2.44.


In [40]: plt.figure(figsize=(13.4,5))

         df.age[(df.income == '>50K\n')].plot(alpha=.25, color='blue')
         df2.age[(df2.income == '>50K\n')].plot(alpha=.45,color='red')

         plt.ylabel('Age')
         plt.xlabel('Samples')

Out[40]: Text(0.5, 0, 'Samples')
```




```
In [41]: import numpy as np

         countx,divisionx = np.histogram(ml2_age, normed=True)
         county,divisiony = np.histogram(fm2_age, normed=True)

In [42]: import matplotlib.pyplot as plt

         val = [(divisionx[i]+divisionx[i+1])/2 for i in range(len(divisionx)-1)]

         plt.plot(val, countx-county,'o-')
         plt.title('Differences in promoting men vs. women')
         plt.xlabel('Age',fontsize=15)
         plt.ylabel('Differences',fontsize=15)
         plt.show()
```




```
In [43]: print ("Remember:\n We have the following mean values for men, women and the difference
         print ("For high-income: ", ml1_age.mean(), fm1_age.mean(), ml1_age.mean()- fm1_age.mea
         print ("After cleaning: ", ml2_age.mean(), fm2_age.mean(), ml2_age.mean()- fm2_age.mean

         print ("\nThe same for the median:")
         print (ml_age.median(), fm_age.median(), ml_age.median()- fm_age.median()) # The differ
         print (ml1_age.median(), fm1_age.median(), ml1_age.median()- fm1_age.median()) # The di
         print (ml2_age.median(), fm2_age.median(), ml2_age.median()- fm2_age.median()), # The d
```

```
Remember:
 We have the following mean values for men, women and the difference:
Originally:  39.43354749885268 36.85823043357163 2.5753170652810553
For high-income:  44.62578805163614 42.125530110262936 2.5002579413732064
After cleaning:  44.317982123920615 41.877028181041844 2.440953942878771

The same for the median:
38.0 35.0 3.0
44.0 41.0 3.0
44.0 41.0 3.0
```

```
Out[43]: (None,)
```

```
In [44]: def skewness(x):
             res=0
```

```
        m=x.mean()
        s=x.std()
        for i in x:
            res+=(i-m)*(i-m)*(i-m)
        res/=(len(x)*s*s*s)
        return res

    print ("The skewness of the male population is:", skewness(ml2_age))
    print ("The skewness of the female population is:", skewness(fm2_age))
```

```
The skewness of the male population is: 0.2664443838432819
The skewness of the female population is: 0.38633352491285977
```

Q20. Explain the result Skewness defines the extent to which a distribution differs from a normal distribution. In both cases for male and female, value is > 0 which means that there is more weight in the left tail of the distribution given value (younger ages).

```
In [45]: def pearson(x):
             return 3*(x.mean()-x.median())/x.std()

         print ("The Pearson's coefficient of the male population is:", pearson(ml2_age))
         print ("The Pearson's coefficient of the female population is:", pearson(fm2_age))
```

```
The Pearson's coefficient of the male population is: 0.0952066054901639
The Pearson's coefficient of the female population is: 0.2621531209596965
```

```
In [46]: #ml1 = df[(df.sex == 'Male')&(df.income=='>50K\n')]

         ml2 = ml1.drop(ml1.index[(ml1['age']>df['age'].median() +35)&(ml1['age']> df['age'].med

         fm2 = fm1.drop(fm1.index[(fm1['age']> df['age'].median() + 35)& (fm1['age']> df['age'].

         print (ml2.shape, fm2.shape)
```

```
(6601, 15) (1171, 15)
```

```
In [47]: print ("Men grouped in 3 categories:")
         print ("Young:",int(round(100*len(ml2_age[ml2_age<41])/float(len(ml2_age.index)))),"%."
         print ("Elder:", int(round(100*len(ml2_age[ml2_age >44])/float(len(ml2_age.index)))),"%
         print ("Average age:", int(round(100*len(ml2_age[(ml2_age>40) & (ml2_age< 45)])/float(l
```

```
Men grouped in 3 categories:
Young: 38 %.
Elder: 48 %.
Average age: 14 %.
```

```
In [48]: print ("Women grouped in 3 categories:")
         print ("Young:",int(round(100*len(fm2_age[fm2_age <41])/float(len(fm2_age.index)))),"%.
         print ("Elder:", int(round(100*len(fm2_age[fm2_age >44])/float(len(fm2_age.index)))),"%
         print ("Average age:", int(round(100*len(fm2_age[(fm2_age>40) & (fm2_age< 45)])/float(l
```

```
Women grouped in 3 categories:
Young: 48 %.
Elder: 37 %.
Average age: 15 %.
```

```
In [49]: print ("The male mean:", ml2_age.mean())
         print ("The female mean:", fm2_age.mean())
```

```
The male mean: 44.317982123920615
The female mean: 41.877028181041844
```

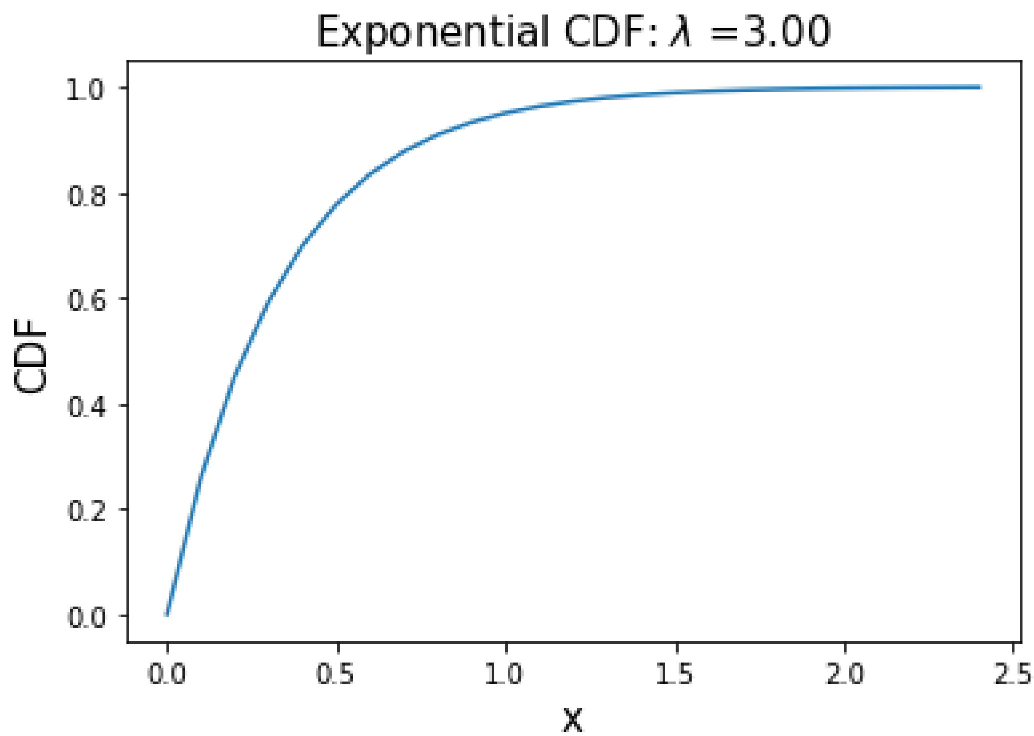
```
In [50]: ml2_young = len(ml2_age[(ml2_age<41)])/float(len(ml2_age.index))
         fm2_young  = len(fm2_age[(fm2_age<41)])/float(len(fm2_age.index))
         print ("The relative risk of female early promotion is: ", 100*(1-ml2_young/fm2_young))
```

```
The relative risk of female early promotion is:  21.125440082163816
```

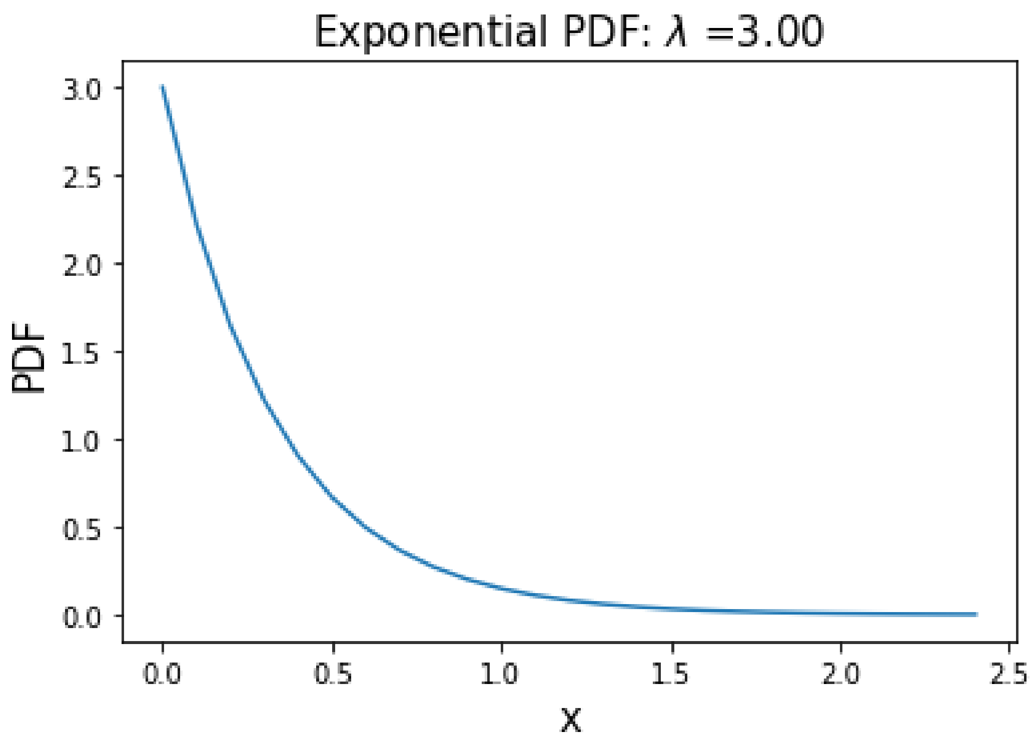
```
In [51]: ml2_elder = len(ml2_age[(ml2_age>44)])/float(len(ml2_age.index))
         fm2_elder  = len(fm2_age[(fm2_age>44)])/float(len(fm2_age.index))
         print ("The relative risk of male late promotion is: ", 100*ml2_elder/fm2_elder)
```

```
The relative risk of male late promotion is:  128.9715708971242
```

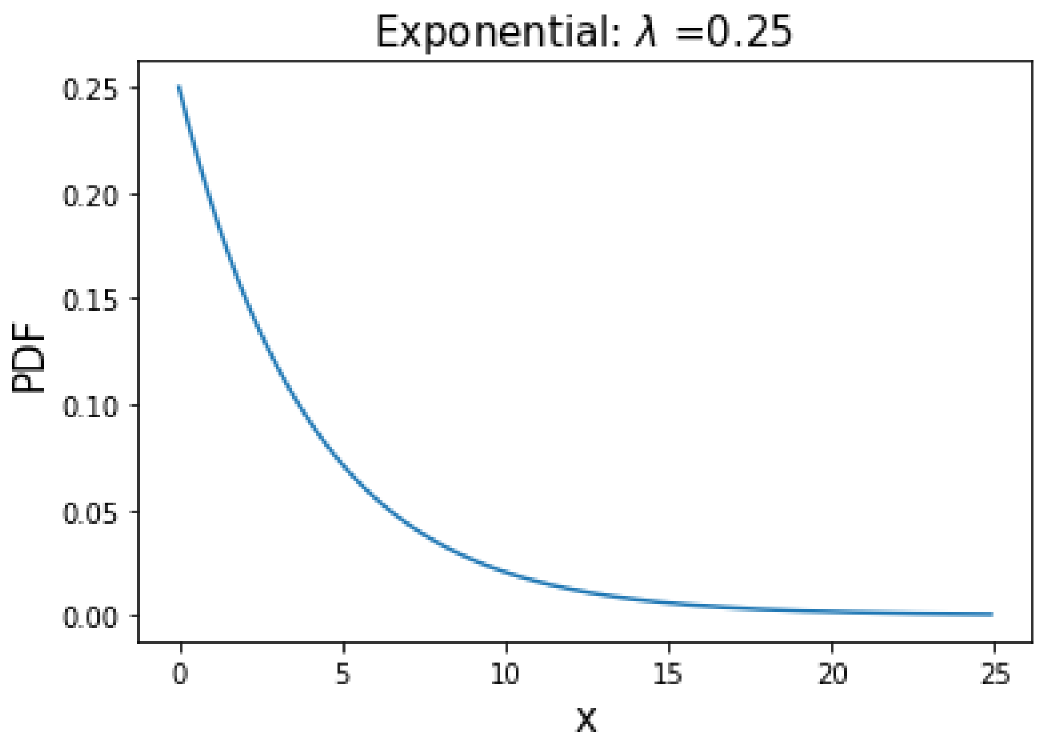
```
In [52]: l = 3
         x=np.arange(0,2.5,0.1)
         y= 1- np.exp(-l*x)

         plt.plot(x,y,'-')
         plt.title('Exponential CDF: $\lambda$ =%.2f'% l ,fontsize=15)
         plt.xlabel('x',fontsize=15)
         plt.ylabel('CDF',fontsize=15)
         plt.show()
```



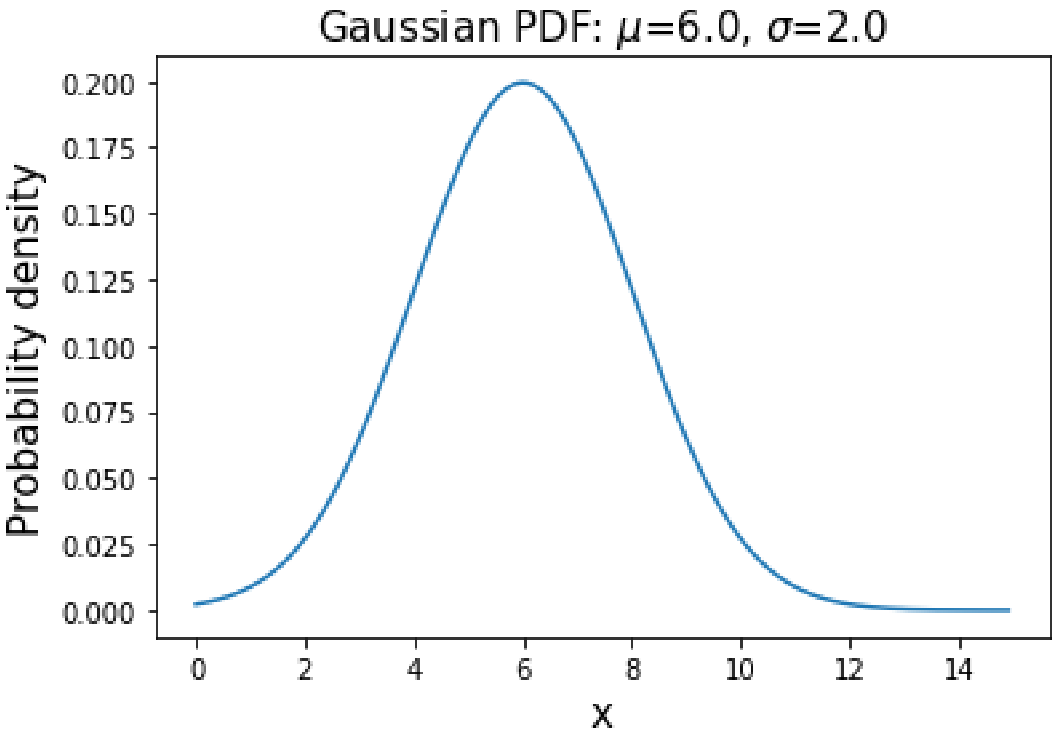
In [53]:
```
from __future__ import division
import scipy.stats as stats

#i
l = 3

x=np.arange(0,2.5,0.1)
y= l * np.exp(-l*x)

plt.plot(x,y,'-')
plt.title('Exponential PDF: $\lambda$ =%.2f'% l, fontsize=15)
plt.xlabel('x', fontsize=15)
plt.ylabel('PDF', fontsize=15)
plt.show()
```



In [54]:
```
l = 0.25

x=np.arange(0,25,0.1)
y= l * np.exp(-l*x)

plt.plot(x,y,'-')
plt.title('Exponential: $\lambda$ =%.2f' %l ,fontsize=15)
plt.xlabel('x',fontsize=15)
plt.ylabel('PDF',fontsize=15)
plt.show()
```



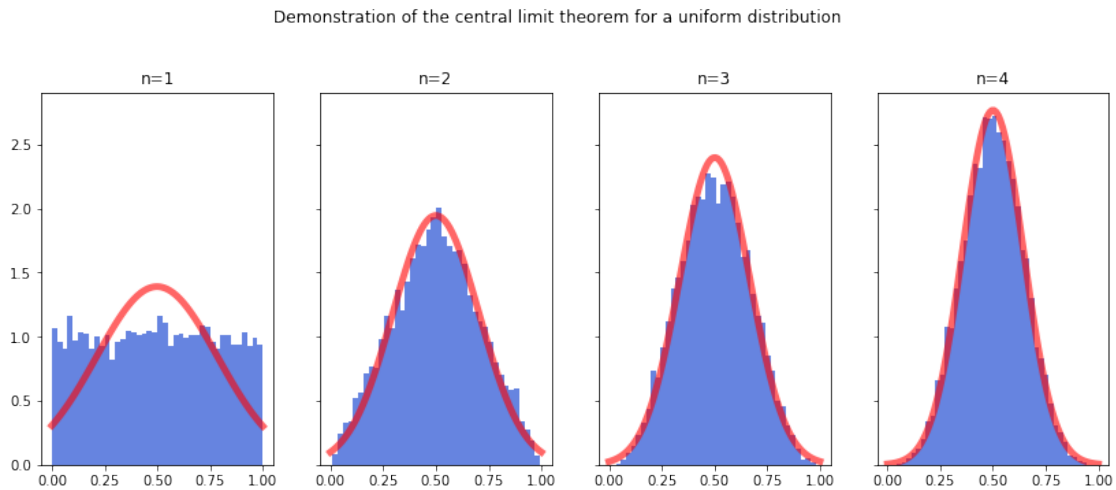
In [55]:
```
u=6 # mean
s=2 # standard deviation

x=np.arange(0,15,0.1)

y=(1/(np.sqrt(2*np.pi*s*s)))*np.exp(-(((x-u)**2)/(2*s*s)))

plt.plot(x,y,'-')
plt.title('Gaussian PDF: $\mu$=%.1f, $\sigma$=%.1f'%(u,s),fontsize=15)
plt.xlabel('x',fontsize=15)
plt.ylabel('Probability density',fontsize=15)
plt.show()
```



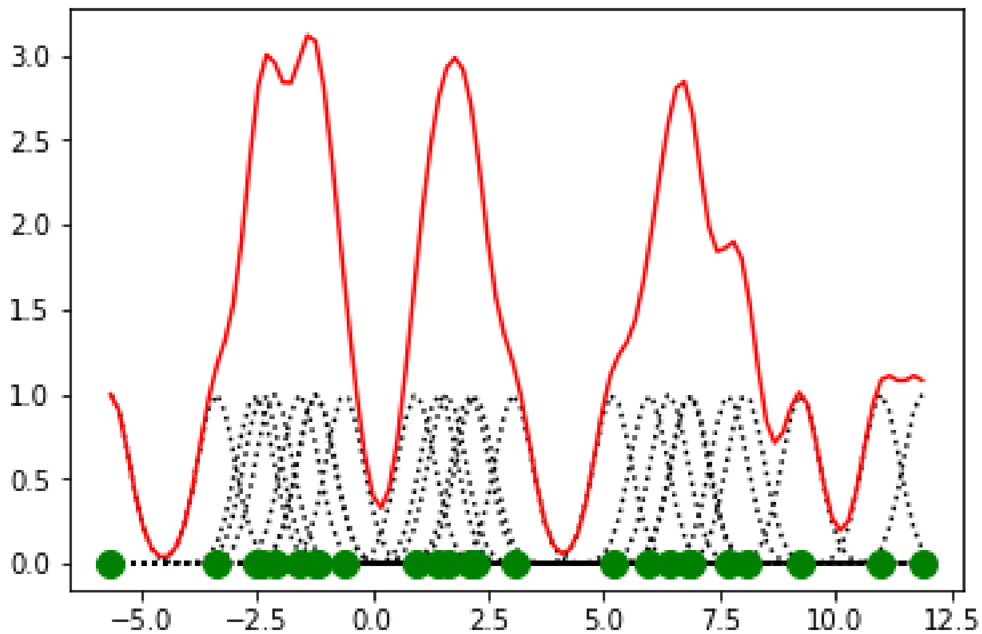

```
In [56]: fig, ax = plt.subplots(1, 4, sharey=True, squeeze=True, figsize=(14, 5))
         x = np.linspace(0, 1, 100)
         for i in range(4):
             f = np.mean(np.random.random((10000, i+1)), 1)
             m, s = np.mean(f), np.std(f, ddof=1)
             fn = (1/(s*np.sqrt(2*np.pi)))*np.exp(-(x-m)**2/(2*s**2))  # normal pdf
             ax[i].hist(f, 40, normed=True, color=[0, 0.2, .8, .6])
             ax[i].set_title('n=%d' %(i+1))
             ax[i].plot(x, fn, color=[1, 0, 0, .6], linewidth=5)
         plt.suptitle('Demonstration of the central limit theorem for a uniform distribution', y
         plt.show()
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
```

```
alternative="'density'", removal="3.1")
```

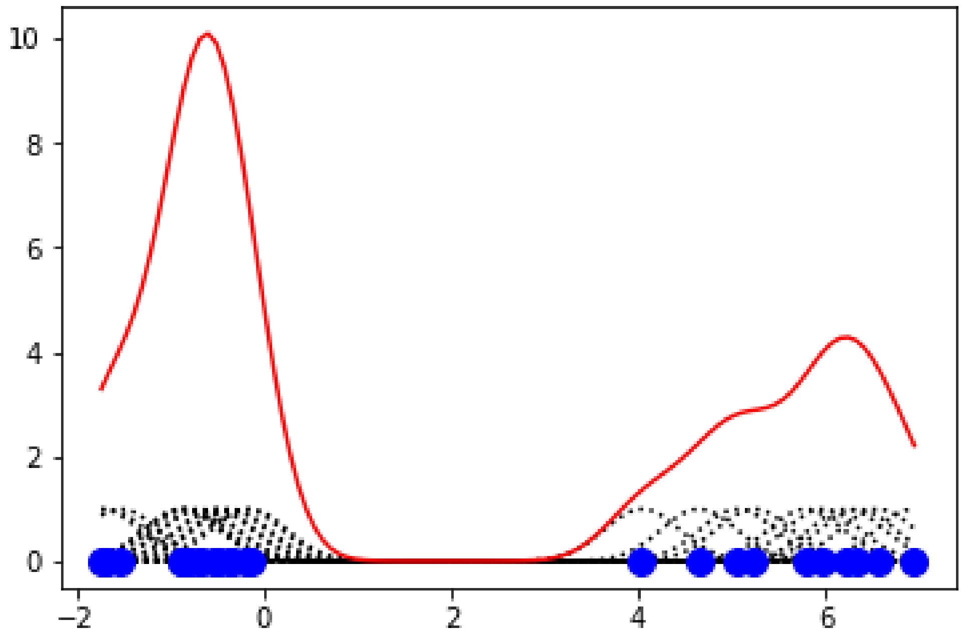



```
In [57]: from scipy.stats.distributions import norm

         # Some random data
         y = np.random.random(15) * 10
         x = np.linspace(0, 10,100)

         x1 = np.random.normal(-1, 2, 15) # parameters: (loc=0.0, scale=1.0, size=None)
         x2 = np.random.normal(6, 3, 10)

         y = np.r_[x1, x2]
         # r_ Translates slice objects to concatenation along the first axis.
         x = np.linspace(min(y), max(y), 100)

         # Smoothing parameter
         s = 0.4

         # Calculate the kernels
         kernels = np.transpose([norm.pdf(x, yi, s) for yi in y])
         plt.plot(x, kernels, 'k:')
         plt.plot(x, kernels.sum(1), 'r')
         plt.plot(y, np.zeros(len(y)), 'go', ms=10)

Out[57]: [<matplotlib.lines.Line2D at 0x7f18fc10a0f0>]
```



In [58]:
```
from scipy.stats import kde
import numpy as np

x1 = np.random.normal(-1, 0.5, 15) # random array of 15 elements from -1 to 0.5

# parameters: (loc=0.0, scale=1.0, size=None)

x2 = np.random.normal(6, 1, 10) # random array of 10 elements from 6 to 1
y = np.r_[x1, x2] # reborujar ambos arreglos

# r_ Translates slice objects to concatenation along the first axis.

x = np.linspace(min(y), max(y), 100) # ordered linear array from -1.7 to 6.94...
s = 0.4   # Smoothing parameter

kernels = np.transpose([norm.pdf(x, yi, s) for yi in y]) # calculates probability densi

# Calculate the kernels
density = kde.gaussian_kde(y) # perform Kernel Density Estimation

plt.plot(x, kernels, 'k:') # plots PDF against x; dotted wave line
plt.plot(x, kernels.sum(1), 'r') # plots PDF + 1 against x; red solid line
plt.plot(y, np.zeros(len(y)), 'bo', ms=10) # plots an array of zeros with size y, agai
```

Out[58]: [<matplotlib.lines.Line2D at 0x7f18fbd495f8>]

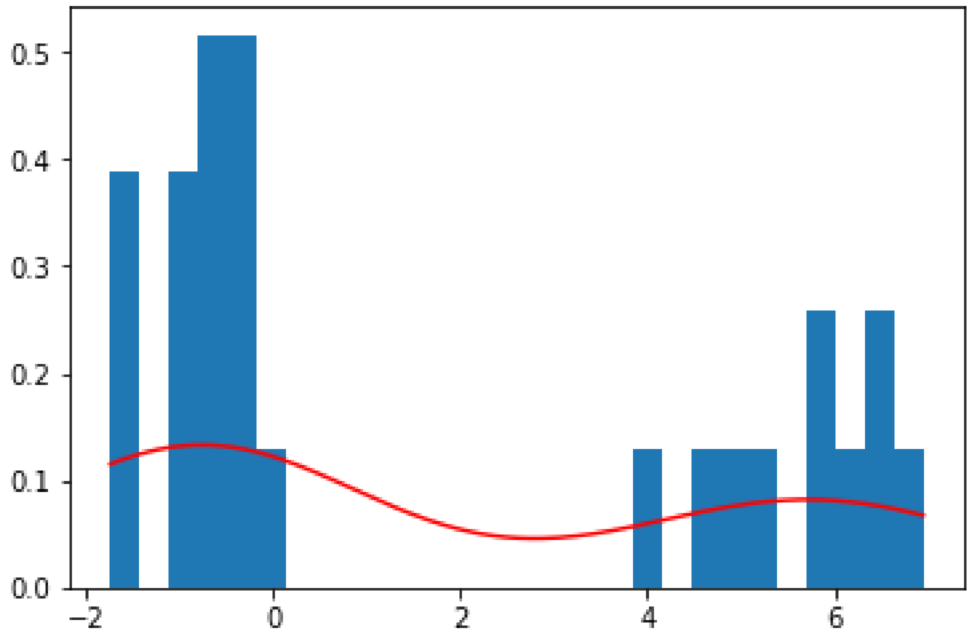


Q21 What does the figure show? plots PDF against x; dotted wave line plots PDF + 1 against x; red solid line plots an array of zeros with size y, against y; blue dots

```
In [59]: xgrid = np.linspace(x.min(), x.max(), 200)
         plt.hist(y, bins=28, normed=True)
         plt.plot(xgrid, density(xgrid), 'r-')
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```

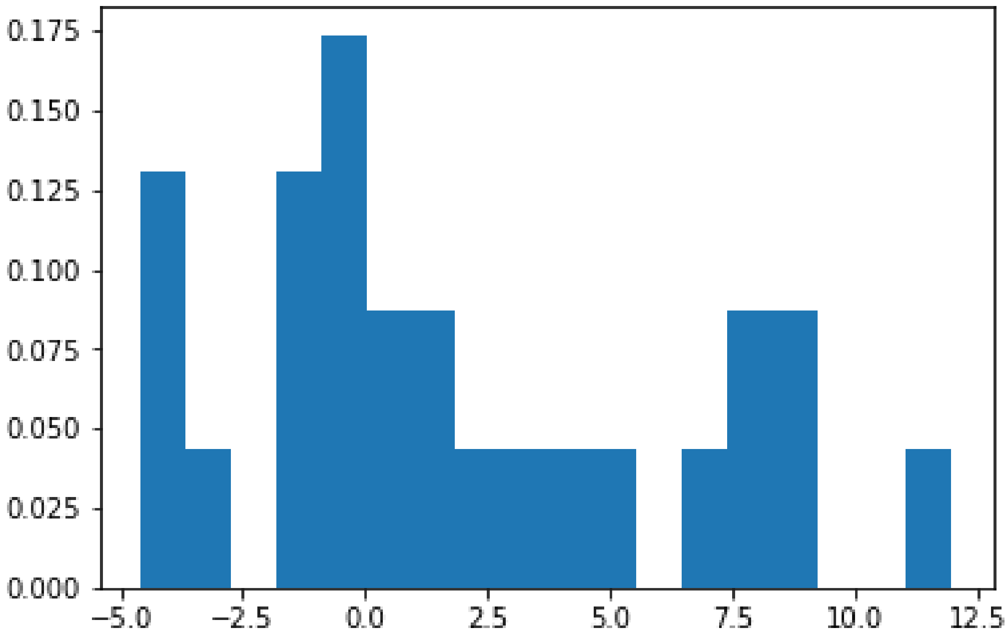
```
Out[59]: [<matplotlib.lines.Line2D at 0x7f18fc073e10>]
```




```
In [60]: # Create a bi-modal distribution with a mixture of Normals.

         x1 = np.random.normal(-1, 2, 15) # parameters: (loc=0.0, scale=1.0, size=None)
         x2 = np.random.normal(6, 3, 10)

         # Append by row
         x = np.r_[x1, x2]

         # r_ Translates slice objects to concatenation along the first axis.
         plt.hist(x, bins=18, normed=True)
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```

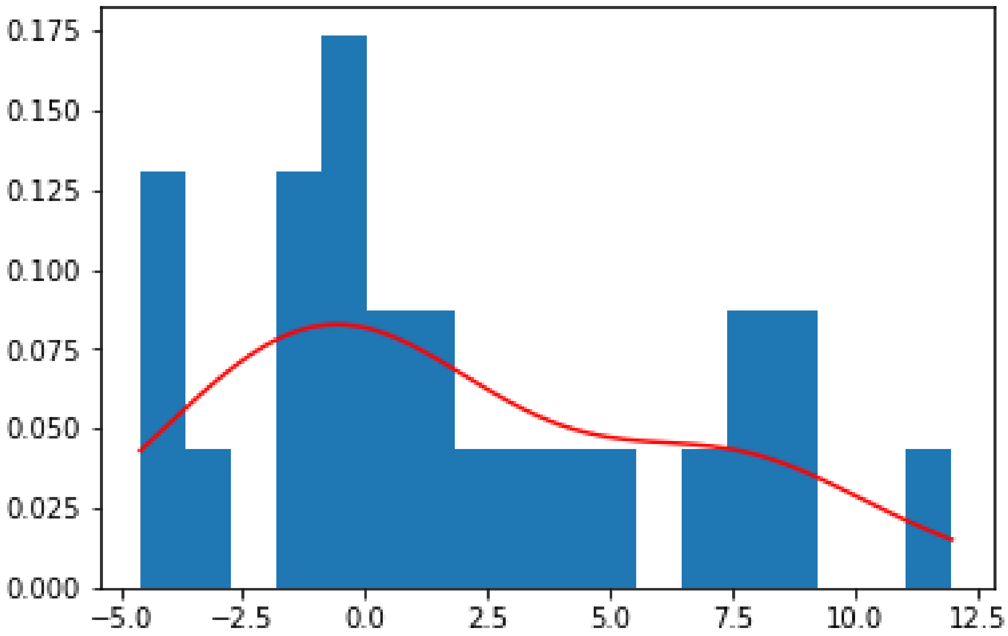
```
Out[60]: (array([0.13019574, 0.04339858, 0.        , 0.13019574, 0.17359432,
                 0.08679716, 0.08679716, 0.04339858, 0.04339858, 0.04339858,
                 0.04339858, 0.        , 0.04339858, 0.08679716, 0.08679716,
                 0.        , 0.        , 0.04339858]),
          array([-4.58989212, -3.66820298, -2.74651384, -1.82482471, -0.90313557,
                  0.01855357,  0.94024271,  1.86193185,  2.78362099,  3.70531012,
                  4.62699926,  5.5486884 ,  6.47037754,  7.39206668,  8.31375581,
                  9.23544495, 10.15713409, 11.07882323, 12.00051237]),
          <a list of 18 Patch objects>)
```




```
In [61]: density = kde.gaussian_kde(x)
         xgrid = np.linspace(x.min(), x.max(), 200)
         plt.hist(x, bins=18, normed=True)
         plt.plot(xgrid, density(xgrid), 'r-')
```

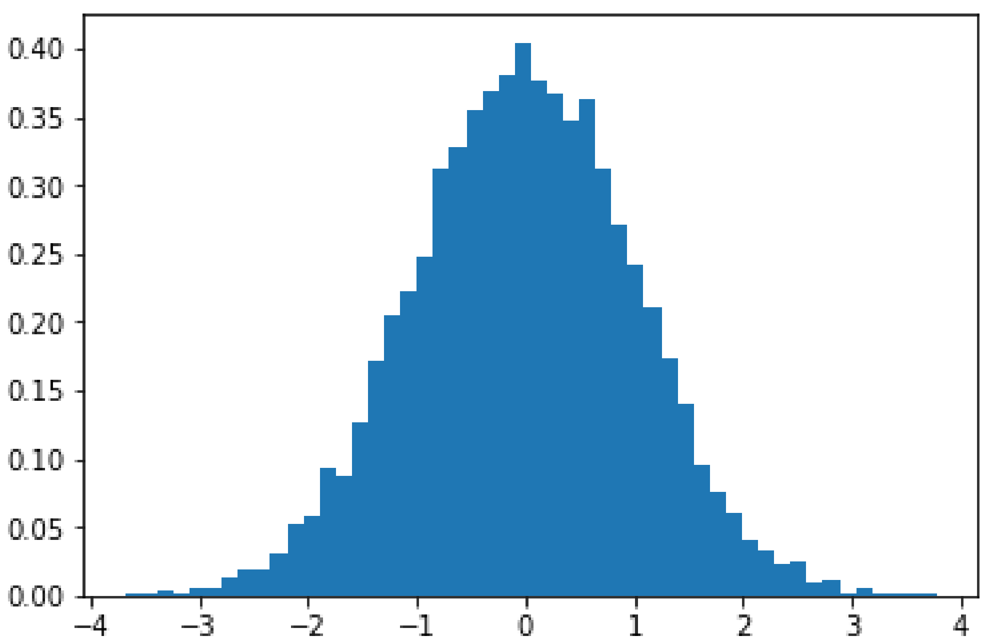
```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```

```
Out[61]: [<matplotlib.lines.Line2D at 0x7f18f811cd30>]
```




```
In [62]: x = np.random.normal(0.0, 1.0, 10000)
         a = plt.hist(x,50,normed='True')
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6499: Matplotlib
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' in
  alternative="'density'", removal="3.1")
```



```
In [63]: print ('The empirical mean of the sample is ', x.mean())
```

```
The empirical mean of the sample is  -0.013098606336531832
```

```
In [64]: NTs=200
         mu=0.0
         var=1.0
         err = 0.0
         NPs=1000
         for i in range(NTs):
             x = np.random.normal(mu, var, NPs) # random array from mu=0 to var=1, size NPs=1000
             err += (x.mean()-mu)**2 # get sum of square mean of every value in array

         print ('MSE: ', err/NTs) # calculate mean squared error (MSE)
```

```
MSE:  0.001015315794121856
```

Q22. What did you obtain as result? MSE: The mean squared error tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them.

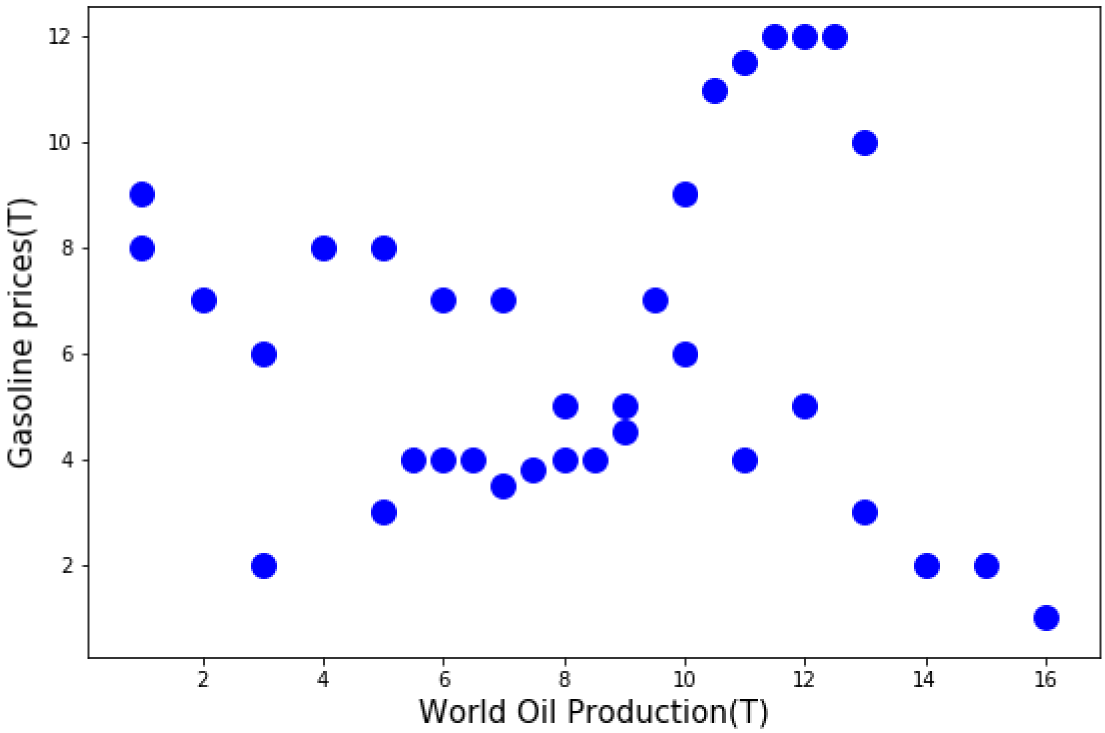
```
In [65]: def Cov(X, Y):
             def _get_dvis(V):
                 return [v - np.mean(V) for v in V]
             dxis = _get_dvis(X)
             dyis = _get_dvis(Y)
             return np.sum([x * y for x, y in zip(dxis, dyis)])/len(X)


         X = [5, -1, 3.3, 2.7, 12.2]
         X= np.array(X)
         Y = [10, 12, 8, 9, 11]

         print ("Cov(X, X) = %.2f" % Cov(X, X))
         print ("Var(X) = %.2f" % np.var(X))

         print ("Cov(X, Y) = %.2f" % Cov(X, Y))
```

```
Cov(X, X) = 18.89
Var(X) = 18.89
Cov(X, Y) = 0.18
```

```
In [66]: MAXN=100
         MAXN=40

         X=np.array([[1,9],[3, 2], [5,3],[5.5,4],[6,4],[6.5,4],[7,3.5],[7.5,3.8],[8,4],
         [8.5,4],[9,4.5],[9.5,7],[10,9],[10.5,11],[11,11.5],[11.5,12],[12,12],[12.5,12],[13,10]]
```

```
In [67]: plt.subplot(1,2,1)
         plt.scatter(X[:,0],X[:,1],color='b',s=120, linewidths=2,zorder=10)
         plt.xlabel('Economic growth(T)',fontsize=15)
         plt.ylabel('Stock market returns(T)',fontsize=15)
         plt.gcf().set_size_inches((20,6))

         X=np.array([[1,8],[2, 7], [3,6],[4,8],[5,8],[6,7],[7,7],[8,5],[9,5],[10,6],[11,4],[12,5

         plt.subplot(1,2,1)
         plt.scatter(X[:,0],X[:,1],color='b',s=120, linewidths=2,zorder=10)
         plt.xlabel('World Oil Production(T)',fontsize=15)
         plt.ylabel('Gasoline prices(T)',fontsize=15)
         plt.gcf().set_size_inches((20,6))
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/figure.py:98: MatplotlibDeprec
Adding an axes using the same arguments as a previous axes currently reuses the earlier instance
  "Adding an axes using the same arguments as a previous axes "
```

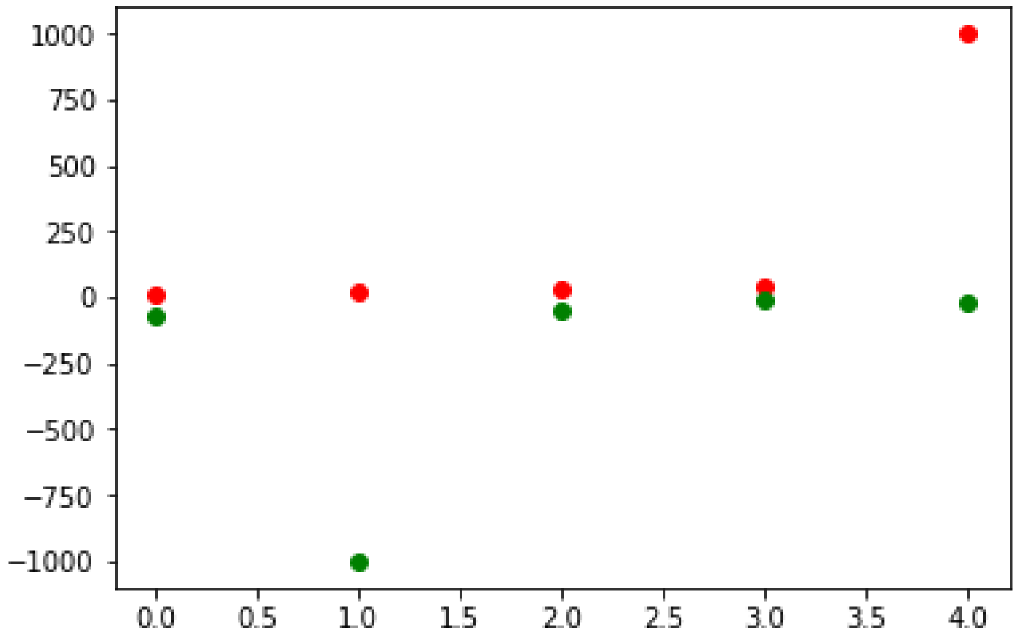


In [68]:
```
def Corr(X, Y):
    assert len(X) == len(Y)
    return Cov(X, Y) / np.prod([np.std(V) for V in [X, Y]])

print ("Corr(X, X) = %.5f" % Corr(X, X))

Y=np.random.random(len(X))

print ("Corr(X, Y) = %.5f" % Corr(X, Y))
```

```
Corr(X, X) = 2.00000
Corr(X, Y) = -0.02126
```

In [79]:
```
def list2rank(l):
#l is a list of numbers
# returns a list of 1-based index; mean when multiple instances
    return [np.mean([i+1 for i, sorted_el in enumerate(sorted(l)) if sorted_el == el])

l = [7, 1, 2, 5]
print ("ranks: ", list2rank(l))

def spearmanRank(X, Y):
    # X and Y are same-length lists
    print (list2rank(X))
    print (list2rank(Y))
    return Corr(list2rank(X), list2rank(Y))

X = [10, 20, 30, 40, 1000]
Y = [-70, -1000, -50, -10, -20]
plt.plot(X,'ro')
plt.plot(Y,'go')

print ("Pearson rank coefficient: %.2f" % Corr(X, Y))
print ("Spearman rank coefficient: %.2f" % spearmanRank(X, Y))
```

```
ranks:  [4.0, 1.0, 2.0, 3.0]
Pearson rank coefficient: 0.28
[1.0, 2.0, 3.0, 4.0, 5.0]
[2.0, 1.0, 3.0, 5.0, 4.0]
Spearman rank coefficient: 0.80
```

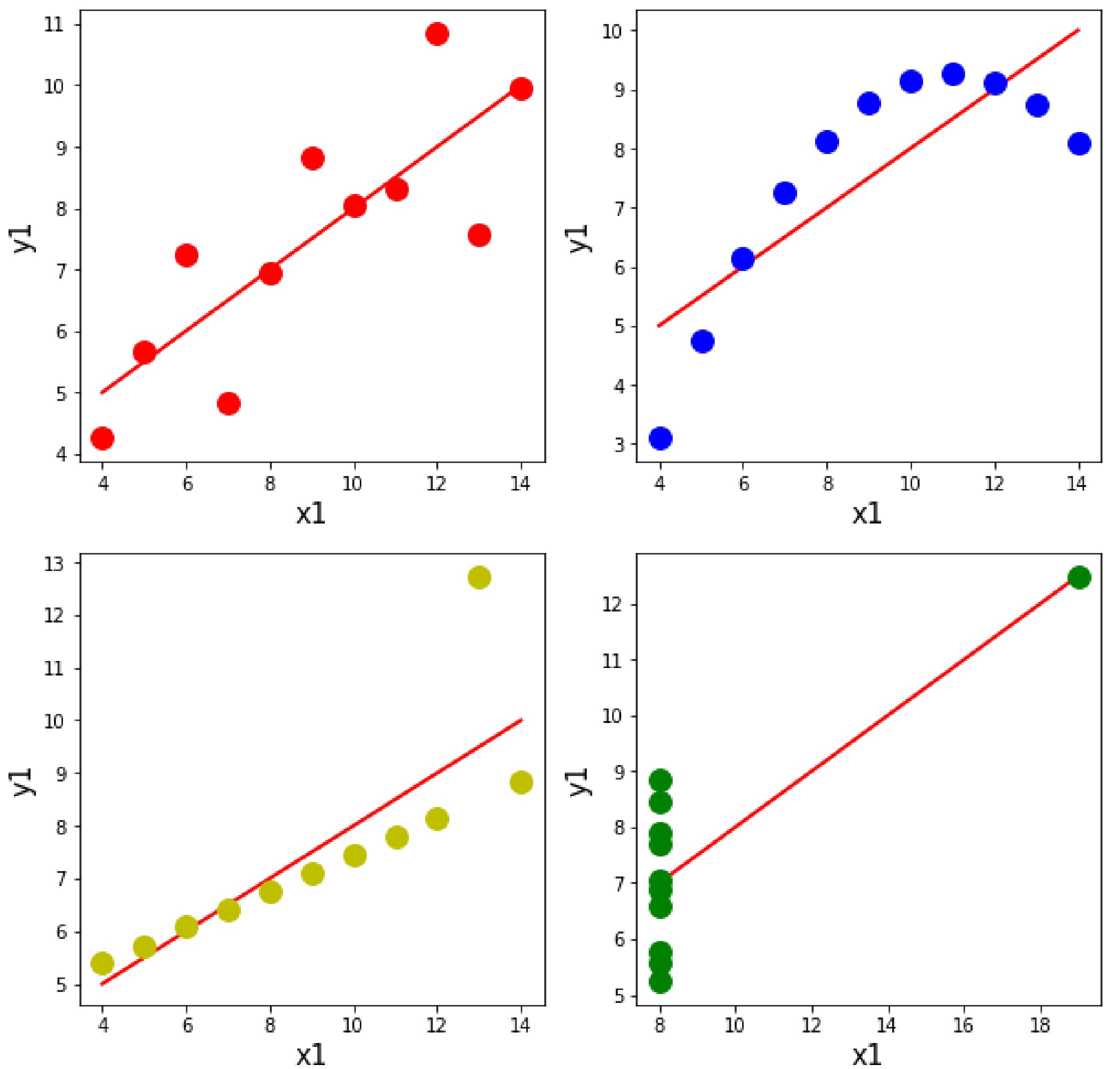


1.1.3 Exercise: Obtain for the Anscombe's quartet [2] given in the figures bellow, the different estimators (mean, variance, covariance for each pair, Pearson's correlation and Spearman's rank correlation.

In [116]:
```
from scipy import stats
# Anscombe's quartet comprises four datasets that have nearly identical simple descrip
X=np.array([[10.0, 8.04,10.0, 9.14, 10.0, 7.46, 8.0, 6.58],
[8.0,6.95, 8.0, 8.14, 8.0, 6.77, 8.0, 5.76],
[13.0,7.58,13.0,8.74,13.0,12.74,8.0,7.71],
[9.0,8.81,9.0,8.77,9.0,7.11,8.0,8.84],
[11.0,8.33,11.0,9.26,11.0,7.81,8.0,8.47],
[14.0,9.96,14.0,8.10,14.0,8.84,8.0,7.04],
[6.0,7.24,6.0,6.13,6.0,6.08,8.0,5.25],
[4.0,4.26,4.0,3.10,4.0,5.39,19.0,12.50],
[12.0,10.84,12.0,9.13,12.0,8.15,8.0,5.56],
[7.0,4.82,7.0,7.26,7.0,6.42,8.0,7.91],
[5.0,5.68,5.0,4.74,5.0,5.73,8.0,6.89]])

def fit(x):
    return 3 + 0.5*x
xfit = np.array([np.amin(X[:,0]), np.amax(X[:,0])])

plt.subplot(2,2,1)
plt.scatter(X[:,0],X[:,1],color='r',s=120, linewidths=2,zorder=10)
plt.plot(X[:,0], X[:,1], 'ks', xfit, fit(xfit), 'r-', lw=2)
```

```
        plt.xlabel('x1',fontsize=15)
        plt.ylabel('y1',fontsize=15)

        plt.subplot(2,2,2)
        plt.scatter(X[:,2],X[:,3],color='b',s=120, linewidths=2,zorder=10)
        plt.plot(X[:,2], X[:,3], 'ks', xfit, fit(xfit), 'r-', lw=2)
        plt.xlabel('x1',fontsize=15)
        plt.ylabel('y1',fontsize=15)

        plt.subplot(2,2,3)
        plt.scatter(X[:,4],X[:,5],color='y',s=120, linewidths=2,zorder=10)
        plt.plot(X[:,4], X[:,5], 'ks', xfit, fit(xfit), 'r-', lw=2)
        plt.xlabel('x1',fontsize=15)
        plt.ylabel('y1',fontsize=15)

        xfit = np.array([np.amin(X[:,6]), np.amax(X[:,6])])

        plt.subplot(2,2,4)
        plt.scatter(X[:,6],X[:,7],color='g',s=120, linewidths=2,zorder=10)
        plt.plot(X[:,6], X[:,7], 'ks', xfit, fit(xfit), 'r-', lw=2)
        plt.xlabel('x1',fontsize=15)
        plt.ylabel('y1',fontsize=15)
        plt.gcf().set_size_inches((10,10))

        # mean, variance, covariance for each pair, Pearson's correlation and Spearman's rank
        pairs = (X[:,0], X[:,1]), (X[:,2], X[:,3]), (X[:,4], X[:,5]), (X[:,6], X[:,7])
        for x, y in pairs:
            print('mean=%1.2f, std=%1.2f, r=%1.2f, var=%1.2f, cov=%1.2d, pearson=%s, spearman=
```

```
mean=7.50, std=1.94, r=0.82, var=3.75, cov=04, pearson=(0.81642051634484, 0.002169628873078789),
mean=7.50, std=1.94, r=0.82, var=3.75, cov=04, pearson=(0.8162365060002427, 0.002178816236910803
mean=7.50, std=1.94, r=0.82, var=3.75, cov=04, pearson=(0.8162867394895981, 0.002176305279228030
mean=7.50, std=1.94, r=0.82, var=3.75, cov=04, pearson=(0.816521436888503, 0.002164602347197212
```


y1
x1
y1
x1
y1
x1
y1
x1