Vincent Roy

ID : 119244546

In [1]:
```
# import all of the required libraries for the assignnement
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats  as stats
import math
import random

%matplotlib inline
```

Assignment 1

General Instructions

- The Python standard library is not enough to solve these questions. You will need to import appropriate libraries for each task. Generally, you might import and use any library you wish unless otherwise stated.
- Where detail instructions like variable or function names, required libraries, and etc are not given by the question, feel free to do it the way you would like to.
- After each question, add the needed number of new cells and place your answers inside the cells.
- When you are required to explain or answer in text format open a Markdown cell and enter your answer in it.
- Do not remove or modify the original cells provided by the instructor.
- In the following cell you are provided with some extra possibilities, like colors RED, OKBLUE, or text styles like BOLD or UNDERLINE that you can use to produce text in the output of your codes. For example, to output text in red you can type the following code:

  ```
  print(bcolors.RED + "your text" + bcolors.ENDC)
  ```

- Comment your code whenever needed using # sign at the beginning of the row.
- For the last question you may need to do some online research since not all the details needed are provided in the question. This especially helps you develop some search skills for coding in Python which is inevitable due to the inconsistent syntax of Python.
- Do not hesitate to communicate your questions to the TA's or instructors. Good luck!

In [2]:
```
# The following piece of code gives the opportunity to show multiple outputs
# in one cell:
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Colorful outputs
class bcolors:
    RED       = '\033[91m'
    OKBLUE    = '\033[94m'
    BOLD      = '\033[1m'
    UNDERLINE = '\033[4m'
    ENDC      = '\033[0m'
```

Question 1

Write a piece of program that takes as input 2 lists called `list1` and `list2`. `list1` contains 5 first names (strings) and `list2` has 3 first names. `list1` and `list2` may or may not have common names. Then, it returns a third list called `set_difference` which contains those names in `list1` that are not in `list2`.

Example

list1 = $\{\text{John, Michael, Vanessa, Ahmed, Tiffany}\}$
list2 = $\{\text{Cyrus, Vanessa}\}$
Expected Output : $\{\text{John, Michael, Ahmed, Tiffany}\}$

Answer 1

```
In [3]: # define the two lists
        list1 = ['John', 'Michael','Vanessa','Ahmed','Tiffany']
        list2 = ['Cyrus','Vanessa']
```

```
In [4]: def difference(list1,list2):
            """
            This function takes to lists as arguments and then identifes the names in list
        1 that are not in list2
            """

            # result list
            result = []

            # for each element in list1 verify if it is not in list 2 and if so add to the
        result
            for value in list1:
                if value not in list2:
                    result.append(value)

            return result
```

```
In [5]: set_difference = difference(list1,list2)
        print('The unique elements of list1 with respect to list2 are : '+ str(set_differe
        nce))
```

```
        The unique elements of list1 with respect to list2 are : ['John', 'Michael', 'Ah
        med', 'Tiffany']
```

Question 2

Write a Python **function** that takes two **positive integers** and returns their **greatest common divisor**. In case you pass a negative integer to the function it must return the following string: "This function takes only positive integers!"

Answer 2

In [6]:
```
def greatDiv(a, b):
    """
    This function claculates the greatest comon divisor according to Euclid's algo
rithm which is a recursive algorithm
    """

    if (a<0) or (b<0):

        return 'This function takes only positive integers!'

    if b == 0:
        return a

    # call recursively the function with one of the inputs as the modulo of a to b
    else:
        return greatDiv(b, a % b)
```

In [7]:
```
print('The greatest common divisor is : ' + str(greatDiv(45,9)))
```

```
The greatest common divisor is : 9
```

In [8]:
```
print('The greatest common divisor is : ' + str(greatDiv(45,-9)))
```

```
The greatest common divisor is : This function takes only positive integers!
```

Question 3

Write a function that prints all the prime numbers in the interval $[0, p]$, where p is a parameter to be passed to the function.

Answer 3

In [9]:
```
def isPrime(number):
    """
    This function identifies if the number is a prime number
    """

    # for every number from 2 to the number passed into the function check to see
if there is a remainder
    for i in range(2,number):

        # if one of the numbers when divided by are passed in number has no remain
der then the number is not prime
        if ((number%i)==0):


            return False

        else:

            pass

    # if all numbers have no remainder with the passed in value then we have ident
ified the prime number
    return True


def primeNumbers(p):
    """
    This function scans all of the numbers from 2 to p and checks to see if it is
prime. If it is a prime number then
    it is added to the result list of prime numbers
    """

    result = []

    # scan all the numbers from 2 to p to see if it is a prime number
    for number in range(2,p+1):

        if isPrime(number):

            result.append(number)

    return result
```

In [10]:
```
print('The following is the list of prime numbers from o to 45')
print(primeNumbers(45))
```

```
The following is the list of prime numbers from o to 45
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43]
```

Question 4

1. Set seed by the initial value 1231 and define the following variables (Python objects) with the shown assigned values:
   - `size` $\longleftarrow 1000$,
   - `n` $\longleftarrow 700$,
   - `p` $\longleftarrow 0.3$
2. Randomly generate `size` number of itegers in $(0, 200)$ and save them as `col1`.
3. Randomly generate `size` number of values according to $\mathrm{Unif}[0, 1]$ and call it `col2`.
4. Randomly generate `2*size` numbers from $\mathrm{Binom}($ `n`,`p` $)$ and randomly select `size` number of them and save as `col3`.
5. Define the following functions
   - $\mathtt{funct1}(x) = \ln x$
   - $\mathtt{funct2}(x) = \frac{10\exp(x)}{1+\exp(x)}$
   - $\mathtt{funct3}(x) = \frac{350}{100\sqrt{2\pi}}.\exp\left(-\frac{x^2}{20000}\right)$
6. Define
   - `col4`$=\mathtt{funct1}($ `col1` $)$,
   - `col5`$=\mathtt{funct2}($ `col2` $)$, and
   - `col6`$=\mathtt{funct3}($ `col3` $)$;
7. Randomly generate `size` number of genders from the set $\{\mathrm{Female}, \mathrm{Male}\}$ and save them as `col7`.
8. Construct a **data frame** with 7 columns `col1` to `col7` and call it `mydata`.
9. Describe the dataset using the descriptive statistics discussed in the class.
10. Use the appropriate visualisation tool to visualize each variable in the dataset.
11. Using **loops** scatterplot every pair of columns versus each other if appropriate.

Answer 4-1

In [11]:
```
random.seed(1231)

size = 1000
n = 700
p = 0.3
```

Answer 4-2

Note that I changed the value range 1 to 200, since a 0 will cause problems with natural log calculations in col4

In [12]:
```
# generate array of length size with random integers from 0 to 200 as a numpy arra
y
col1 = np.array([random.randint(1,200) for x in range(size)])
```

Answer 4-3

In [13]:
```
# generate array of length size with floats from 0 to 1 as numpy array
col2 = np.array([random.uniform(0,1) for x in range(size)])
```

Answer 4-4

In [14]:
```
# genrate array of length size*2 with number from a binomial distribution with par
ameter n amd p as numpy array
temp = stats.binom.rvs(n=n,p=p,size=size*2)

# smaple the temp array above
col3 = np.random.choice(a=temp,size=size)
```

Answer 4-5

In [15]:
```
def funct1(vector):
    """
    Function that returns the natural log of an array
    """

    result = np.log(vector)

    return result

def funct2(vector):
    """
    Function that returns 10*exp(x)/(1+exp(x))
    """

    result = (10*np.exp(vector))/(1+np.exp(vector))

    return result


def funct3(vector):
    """
    Function that returns 350/(100*2pi**0.5)*exp(-x**2/2000)
    """

    result = (350/(100*math.sqrt(2*math.pi)))*np.exp(-1*np.power(vector,2)/2000)

    return result
```

Answer 4-6

In [16]:
```
col4 = funct1(col1)

col5 = funct2(col2)

col6 = funct3(col3)
```

Answer 4-7

In [17]:
```
col7 = np.array([random.choice(['Female','Male']) for x in range(size)])
```

Answer 4-8

```
In [18]: mydata = pd.DataFrame({'col1':col1,'col2':col2,'col3':col3,'col4':col4,'col5':col5
         ,'col6':col6,'col7':col7})
         mydata.head()
```

Out[18]:

	col1	col2	col3	col4	col5	col6	col7
0	97	0.382249	198	4.574711	5.944154	4.284870e-09	Female
1	148	0.864731	211	4.997212	7.036482	3.001690e-10	Male
2	66	0.710638	189	4.189655	6.705421	2.444902e-08	Male
3	9	0.228006	206	2.197225	5.567558	8.513689e-10	Male
4	107	0.048694	225	4.672829	5.121712	1.418728e-11	Female

Answer 4-9

The mean, standard deviation, min and max are calculated for each column

```
In [19]: # mean of each column
         mydata.mean(axis=0)

Out[19]: col1    1.009460e+02
         col2    4.912322e-01
         col3    2.104350e+02
         col4    4.305920e+00
         col5    6.182041e+00
         col6    4.538031e-09
         dtype: float64
```

```
In [20]: # standard deviation of each column
         mydata.std(axis=0)

Out[20]: col1    5.882928e+01
         col2    2.798917e-01
         col3    1.230598e+01
         col4    9.880623e-01
         col5    6.514875e-01
         col6    1.988466e-08
         dtype: float64
```

```
In [21]: # minimum of each column
         mydata.min(axis=0)

Out[21]: col1              1
         col2     0.00352653
         col3            174
         col4              0
         col5        5.00882
         col6    1.76036e-14
         col7         Female
         dtype: object
```

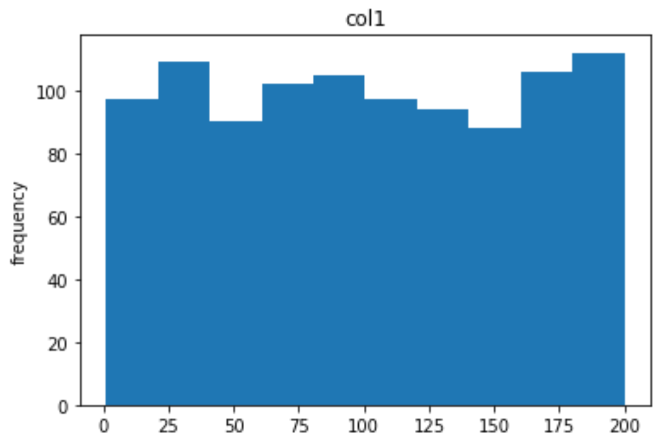
```
In [22]: # maximum of each column
         mydata.max(axis=0)
```

```
Out[22]: col1            200
         col2       0.999766
         col3            253
         col4        5.29832
         col5        7.31013
         col6    3.72073e-07
         col7           Male
         dtype: object
```

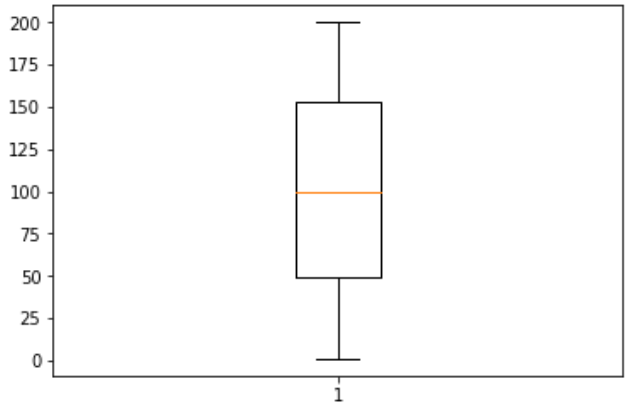
Answer 4-10

Histogram and boxplot for col1

```
In [23]: plt.hist(mydata['col1'])
         plt.title('col1')
         plt.ylabel('frequency');
```

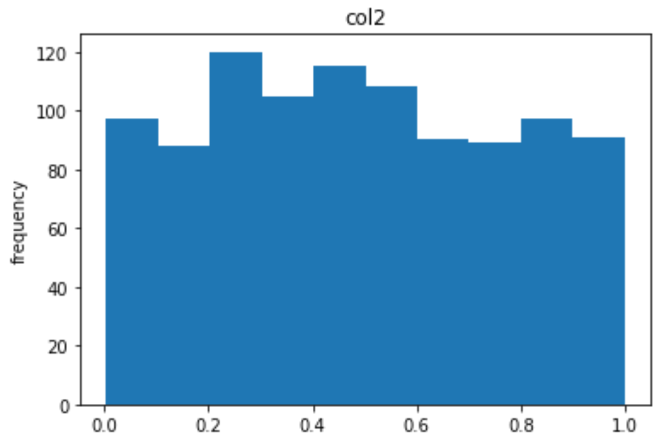


```
In [24]: plt.boxplot(mydata['col1']);
```

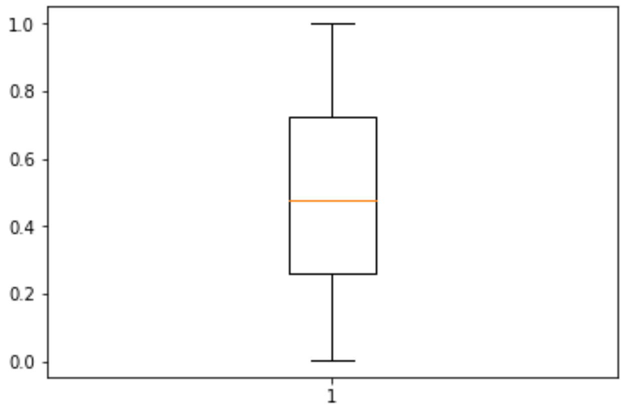


Histogram and boxplot for col2

In [25]:
```
plt.hist(mydata['col2'])
plt.title('col2')
plt.ylabel('frequency');
```

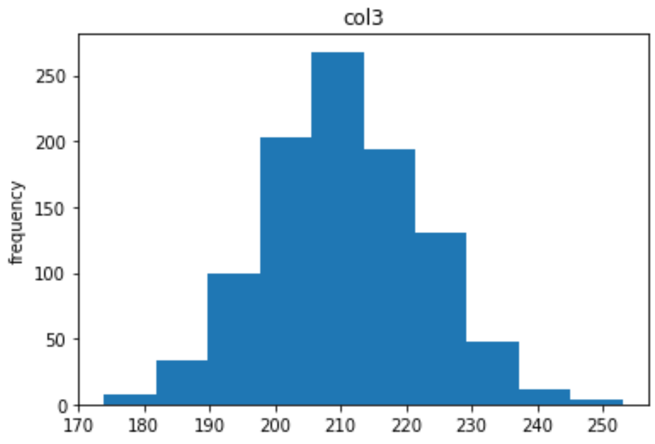


In [26]:
```
plt.boxplot(mydata['col2']);
```

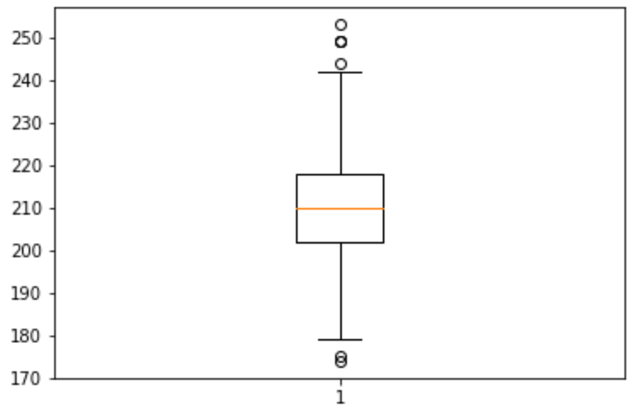


Histogram and boxplot for col3

In [27]:
```
plt.hist(mydata['col3'])
plt.title('col3')
plt.ylabel('frequency');
```

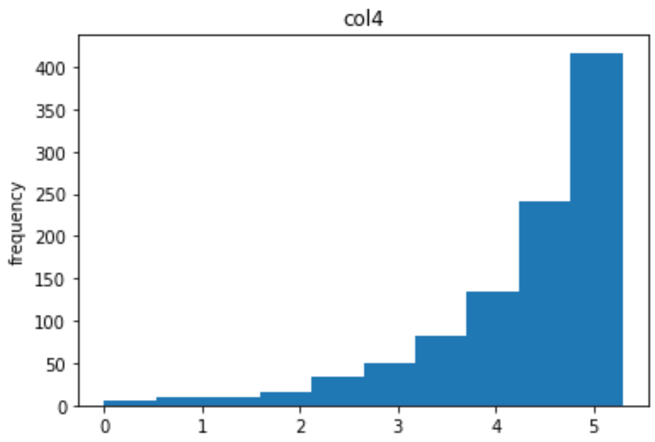


In [28]:
```
plt.boxplot(mydata['col3']);
```

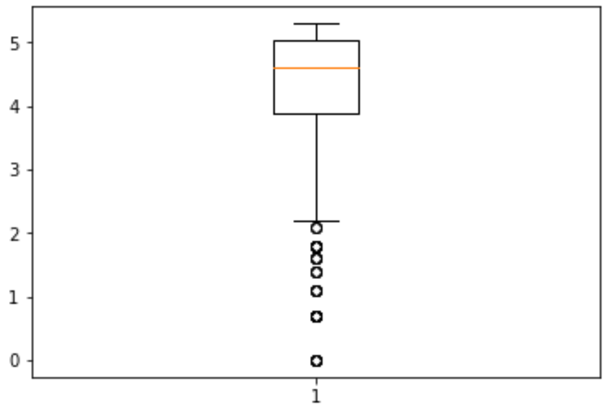


Histogram and boxplot for col4

In [29]:
```
plt.hist(mydata['col4'])
plt.title('col4')
plt.ylabel('frequency');
```

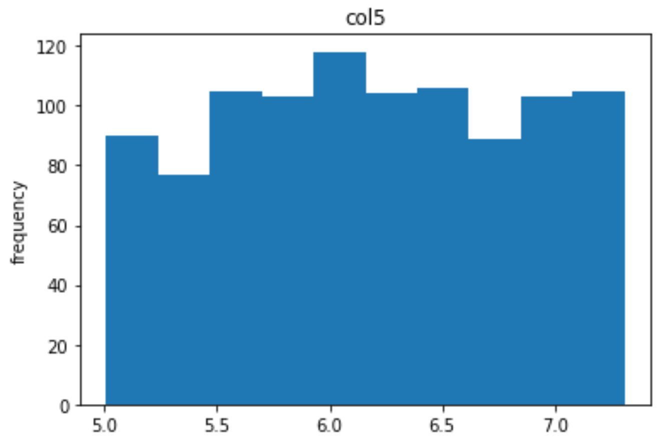


In [30]:
```
plt.boxplot(mydata['col4']);
```

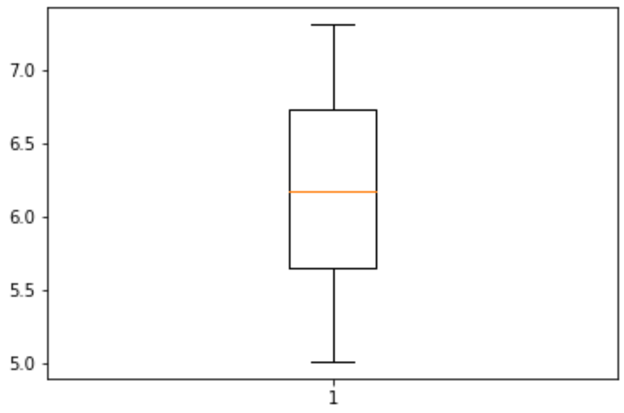


Histogram and boxplot for col5

```
In [31]: plt.hist(mydata['col5'])
         plt.title('col5')
         plt.ylabel('frequency');
```

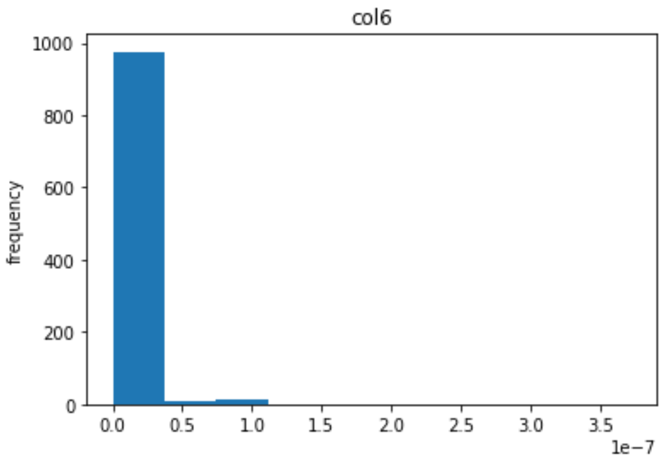


```
In [32]: plt.boxplot(mydata['col5']);
```

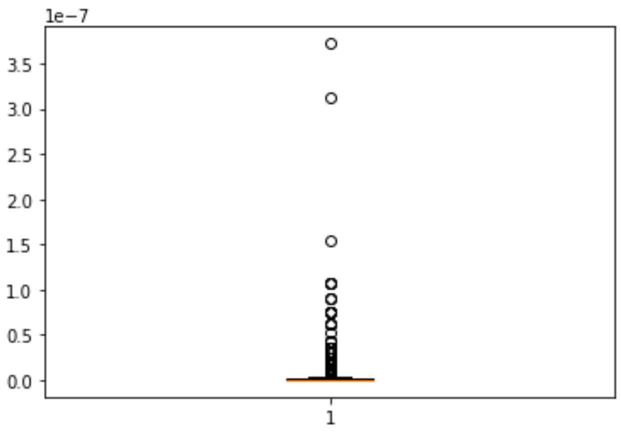


Histogram and boxplot for col6

```
In [33]: plt.hist(mydata['col6'])
         plt.title('col6')
         plt.ylabel('frequency');
```

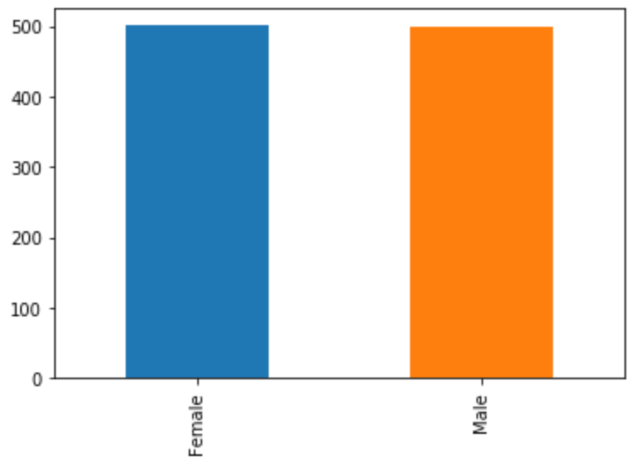


```
In [34]: plt.boxplot(mydata['col6']);
```



Histogram for col7

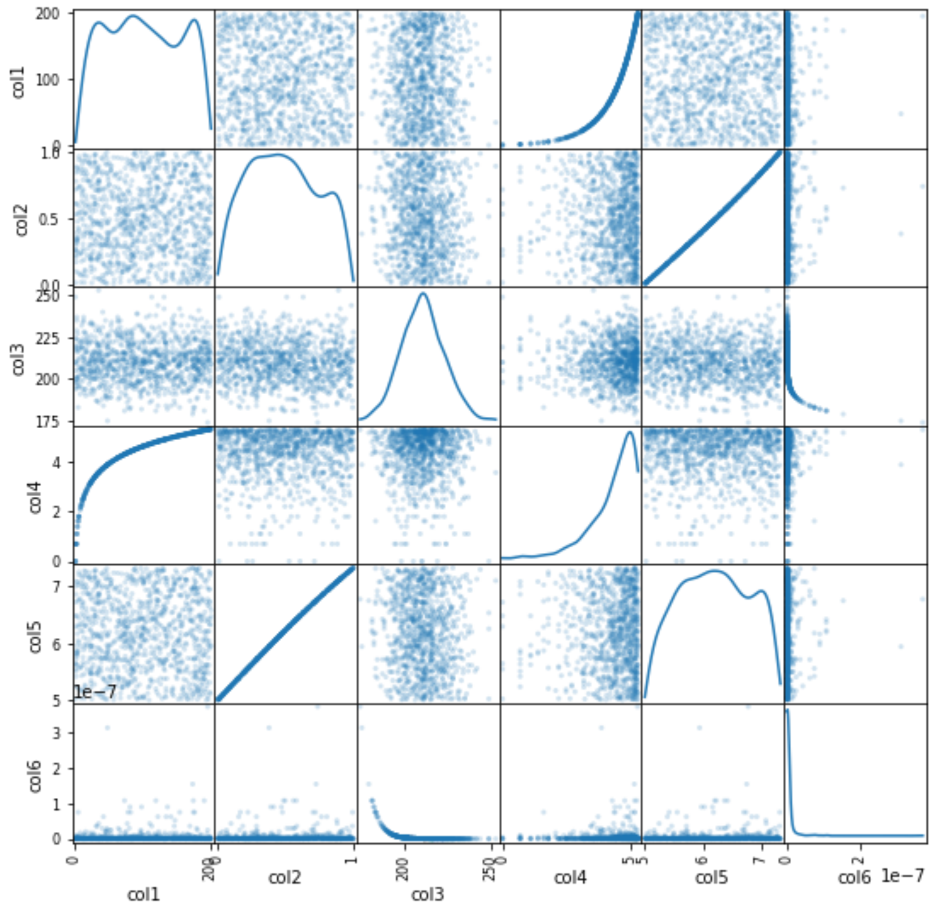
```
In [35]: mydata['col7'].value_counts().plot(kind='bar')
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1a229f2278>
```



Answer 4-11

The following figur ia a matrix scatterplot of all the columns

```
In [36]: # matrix scatterplot of all the columns
         pd.plotting.scatter_matrix(mydata, alpha=0.2, figsize=(9, 9), diagonal='kde');
```



Question 5

1. Assume that $X \sim N(\mu, \sigma^2)$ with $\mu = 55, \sigma = 15$. Randomly generate a set of $1000, 000$ values for X according to the given distribution and call it set D.
2. Pretend that D is your whole population. Choose a sample of 1000 values from D.
3. Plot an approximate density distribution function using the selected sample.
4. Using a loop repeat the second step 30 times. For each sample estimate the population mean and save each estimate. Calculate the *mean squared error* of your estimated means.
5. Plot the histogram of all 30 saved sample means (**Only sample means**). According to the histogram, what is the *sampling distribution* of the mean?
6. If instead of 500 times, we resample over and over for a large number of times, how does the sample mean change?

Answer 5-1

1000000 values from the normal distribution

```
In [37]: D = stats.norm.rvs(loc=55,scale=15,size=1000000)
```

Answer 5-2

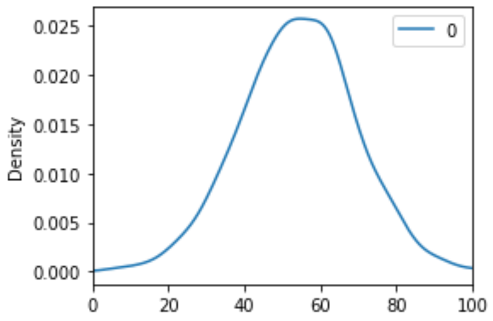
1000 randomly selected points from the whole population D.

```
In [38]: sample = np.random.choice(a=D,size=1000)
```

Answer 5-3

The distribution plot of the sample

```
In [39]: plot = pd.DataFrame(sample).plot(kind="density",figsize=(4,3),xlim=(0,100))
```



Answer 5-4

The following code repeats the sampling process 30 times. Each time the mean is added to a vector of means. The mean square error is calaculated for all of the sample means

```
In [40]: means = []

         mse = 0

         numberSamples = 30

         for i in range(numberSamples):
             sample = np.random.choice(a=D,size=1000)

             sample_mean = sample.mean()

             means.append(sample_mean)

             mse = mse + (55 - sample_mean)**2


         tempMean= np.array(means).mean()
         print('The mean of the sample ' + str(tempMean))

         mse = mse/numberSamples
         print('The MSE is '+str(mse))
```

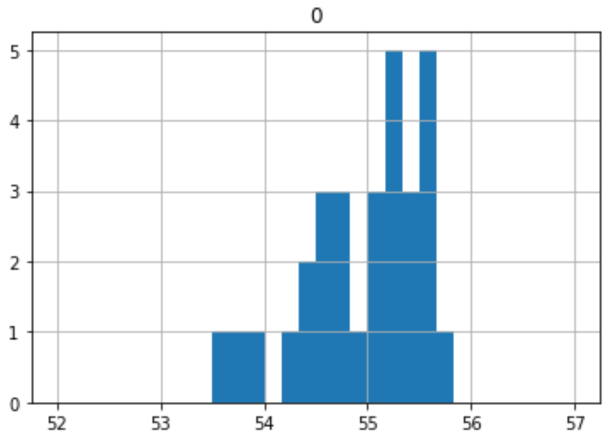
```
The mean of the sample 54.945537750497415
The MSE is 0.31909849778173294
```

Answer 5-5

From the histogram plot of all the sample means we can sse that the distribution is normal (ie gaussian)

We can see from the plot that the mean of the sample is close to the actual mean of 55.

```
In [41]: hist = pd.DataFrame(means).hist(bins=30,range=(52,57),figsize=(6,4))
```



Answer 5-6

I assumed here that you meant 30 times and not 500.

If we sample for lets say 1000 times it is observed that the MSE approches a constant value. That is that the MSE reduces when we increase the number of iterations. For around 1000 iterations we obtan a steady value around 0.2272 for the MSE.

The mean therefore approachs the actual mean of 55 of the distubution

Question 6

Suppose that we would like to model the event of flipping a coin for 15 times. If the probability of getting heads equals 0.6, then answer the following questions.
Do not forget to include your codes.

1. Which distribution is it and what are the parameters of the distribution? Is it a discrete or continuous distribution?
2. What is the probability of obtaining 10 heads? Explain how to calculate it.
3. What is the probability of getting more than or equal to 10 heads? What about less than or equal to 10 head? What should be the summation of these two probabilities and why?
4. Find the expected value of obtained number of heads in each trial? (**Each trial consists of 15 tosses**)
5. How probable is it to get (H,H,H,T,T,H,T,T,H,T,H,H,H,T,H)?
6. Find the first, second, and the third quantiles.
7. Repeat the trial 10 times and estimate the mean each time. Using `pandas.crosstab` build the frequency table of the results. Plot the histogram of these ten estimates.
8. Now, gradually increase the number of trials from 100 to 1000. (start from 100 and add 50 each time to reach 1000.) Plot the histogram for the sample mean each time. How does the sampling distribun of mean is changing?

Answer 6-1

It is a binomial distribution and the parameters are n = 15 and p = 0.6 (ie P(X=H) = 0.6 and P(X=T) = 0.4). It is a discrete distribution.

Answer 6-2

The probability of getting exactly 10 heads P(X=10) is given by the pmf function below
Luckily the stats library has a PMF fuction. Since calculating factorials is a bit of a pain

```
In [42]: # probability of getting exactly 10 heads P(X=10)
         pr_equal_10 = stats.binom.pmf(k=10, n=15, p=0.6)
         print('P(X=10) is '+str(pr_equal_10))
```

```
P(X=10) is 0.18593784476467232
```

Answer 6-3

To answer P(X>=10) more than or equal to 10 heads we must first calculate the P(X<=10) less than or equal to 10 heads

```
In [43]: pr_lessEqual_10 = stats.binom.cdf(k=10,n=15, p=0.6)
         print('P(X<=10) is '+ str(pr_lessEqual_10))
```

```
P(X<=10) is 0.782722294349824
```

The probability of P(X>=10) = 1 - P(X<=10) + P(X=10). Note that we must add P(X=10) since we want the probability of P(X>=10)

```
In [44]: pr_greaterEqual_10 = 1 - pr_lessEqual_10 + pr_equal_10
         print('P(X>=10) is '+ str(pr_greaterEqual_10))
```

```
P(X>=10) is 0.40321555041484836
```

Answer 6-4

The expected number of heads in each trial is E(X) = np = 15*0.6

```
In [45]: expected_heads = 15*0.6
         print(str(expected_heads)+' heads are expected for each trial')
```

```
9.0 heads are expected for each trial
```

Answer 6-5 approach

We know that the probability of Heads is Pr(X=H) = 0.6 Thus the probability of Tails is Pr(X=T) = 1 - 0.6 = 0.4

Since all of the coin flips are independant the probability of getting :

Pr(H,H,H,T,T,H,T,T,H,T,H,H,H,T,H) = (Pr(X=H) **number of heads) * (Pr(X=T)** number of tails)

Note ** is to the power of n

in the set there are 9 heads in the set there are 6 tails

In [46]:
```
pr_HHHTTHTTHTHHHTH_approach2 = (0.6 ** 9)*(0.4 ** 6)
print('The probability of Pr(H,H,H,T,T,H,T,T,H,T,H,H,H,T,H) is '+str(pr_HHHTTHTTHT
HHHTH_approach2) )
```

```
The probability of Pr(H,H,H,T,T,H,T,T,H,T,H,H,H,T,H) is 4.127824281600001e-05
```

Answer 6-6

The first quantile corresponds to number of H where the cumulative probability is 25% of the CDF.

The second quantile corresponds to number of H where the cumulative probability is 50% of the CDF.

The thrid quantile corresponds to number of H where the cumulative probability is 75% of the CDF.

The stats library has function called ppf() that calculates the number of heads that corresponds to cumulative proability.

In [47]:
```
quant1 = stats.binom.ppf(q=0.25,n=15, p=0.6)
quant2 = stats.binom.ppf(q=0.50,n=15, p=0.6)
quant3 = stats.binom.ppf(q=0.75,n=15, p=0.6)

print('The first qunatile is ' + str(quant1) + ' number of heads' )
print('The second qunatile is ' + str(quant2) + ' number of heads' )
print('The third qunatile is ' + str(quant3) + ' number of heads' )
```

```
The first qunatile is 8.0 number of heads
The second qunatile is 9.0 number of heads
The third qunatile is 10.0 number of heads
```

Answer 6-7

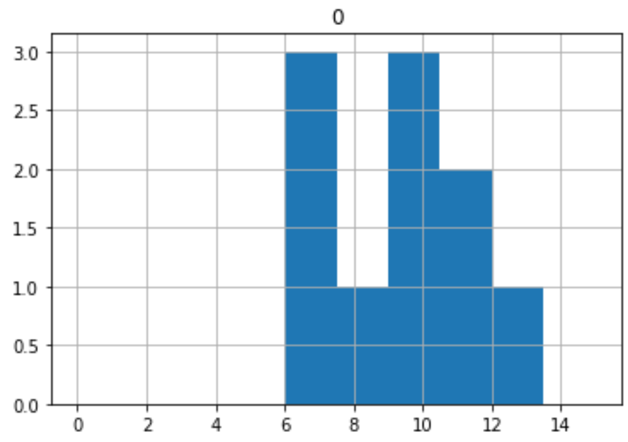
In [48]:
```
tenTrialValues = stats.binom.rvs(n=15, p=0.6, size=10)
meanTenTrialValues = tenTrialValues.mean()

print('Here are the the number of heads for ten trials'+str(tenTrialValues))
print('Here is the mean of the number of hrads for the 10 trails ' + str(meanTenTr
ialValues))
```

```
Here are the the number of heads for ten trials[10 11  9 13  7 11  9  7  8  6]
Here is the mean of the number of hrads for the 10 trails 9.1
```

In [49]:
```
print('Here is the frequency table for the reults of the ten trials')
print(pd.crosstab(index="counts", columns= tenTrialValues))
```

```
Here is the frequency table for the reults of the ten trials
col_0   6   7   8   9   10  11  13
row_0
counts   1   2   1   2   1   2   1
```

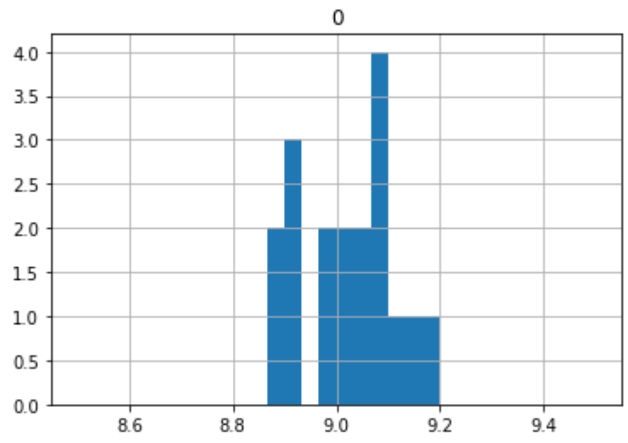
In [50]:
```
hist = pd.DataFrame(tenTrialValues).hist(range=(0,15))
```



Answer 6-8

By looking at the histograms generated below, we can see an increasing trial size make the distribution look like **normal distribution** with a mean of 9

In [51]:
```
trialMeansVector = []

for trialSize in range(100,1000,50):
    trialValues = stats.binom.rvs(n=15, p=0.6, size=trialSize)
    meanTrialValue = trialValues.mean()

    trialMeansVector.append(meanTrialValue)

    #hist = pd.DataFrame(trialMeansVector).hist(bins=100, range=(8.5,9.5))

hist = pd.DataFrame(trialMeansVector).hist(bins=30, range=(8.5,9.5))
```



Question 7

Assume that for a study we want to sample people from the Montreal population. The target of the study is a particular disease. If the probability of sampling a person with this disease equals 0.007.

1. How many people we need to sample totally in order to get exactly 73 patients with the disease?
2. How many people we need to sample totally in order to have at least 73 patients with the target disease?
3. What distribution is it and what are its parameters?
4. Callculate the expected value, variance, as well as the first, second and the third quantiles.

Answer 7-1

We wish to calculate the value of n (number of people sampled) to get an expected value of 73

This is a binomimal distribution, since it is binary outcome (diasease or no disease) of multiple independant events (patients)

Thus E(X) = n * p

p = 0.007 n = ? number we are trying to solve E(X) = 73

n = E(X) / p

```
In [52]: sample_73 = 73 / 0.007
         print('Number of people to sample exactly to get 73 patients with disease is ' + s
         tr(sample_73))
```

```
Number of people to sample exactly to get 73 patients with disease is 10428.5714
28571428
```

Answer 7-2

I am assuming that we want here to identify whole people to get at least 73 people with the disease.

Thus we need to round upwards the number of people, since a fraction of a person does not really exist.

Thus we need sample **10429 people**

Answer 7-3

As mentioned earlier, this is a binomimal distribution, since it is binary outcome (diasease or no disease) of multiple independant events (patients).

The parameters are p = 0.007 n = 10429

```
In [53]: p = 0.007
         n = 10429
```

Answer 7-4

In [54]:
```
expected = n * p
print('The expected value is ' + str(expected))

variance = n*p*(1-p)
print('The variance is ' + str(variance))


quant1 = stats.binom.ppf(q=0.25,n=n, p=p)
quant2 = stats.binom.ppf(q=0.50,n=n, p=p)
quant3 = stats.binom.ppf(q=0.75,n=n, p=p)


print('The first qunatile is ' + str(quant1) + ' people' )
print('The second qunatile is ' + str(quant2) + ' people' )
print('The third qunatile is ' + str(quant3) + ' peolple' )
```

```
The expected value is 73.003
The variance is 72.491979
The first qunatile is 67.0 people
The second qunatile is 73.0 people
The third qunatile is 79.0 peolple
```

Question 8

1. Generate 1000 data points according to the exponential distribution with parameter $\lambda = 1.2$.
2. Estimate the mean of the sample.
3. Repeat the first step but each time increase the sample size up to $1000, 000$. Calculate the sample mean each time. Scatterplot the mean versus the sample size for each repetition. Do you see any trend in the plot sample means? Can you guess the limit of the sample mean due to the plot?

Answer 8-1

The following code produces a vector for all of the sample sizes from 1000 to 1000000 in 10000 increments and calculates the means of each sample. Then the code section plots the mean vs sample size.

Answer 8-2

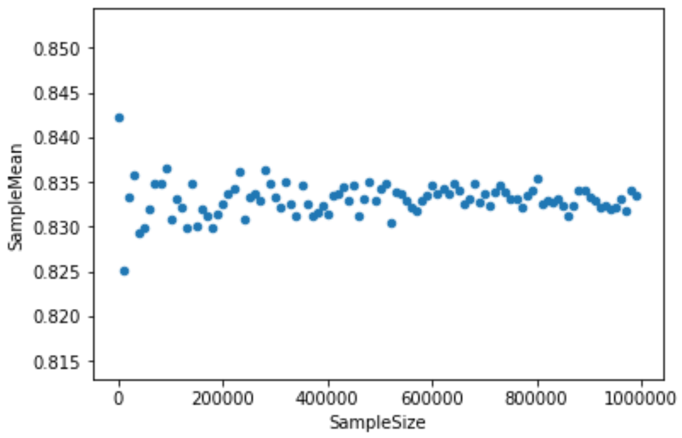
The first esimte of the mean for a sample size of 1000 is 0.8437

Answer 8-3

The scatter plot indicates that the mean tends to 0.8333 which corresponds to 1/lamda, hence 1/1.2.

In [55]:
```
# set the parameters for the sample sizes
startSampleSize = 1000
endSampleSize = 1000000
sampleIncrement = 10000

# create the sample size vector
sampleSizeVector = list(range(startSampleSize,endSampleSize,sampleIncrement))

# create an empty vector
sampleMeanVector = []

# calculate the mean for each sample
for sampleSize in sampleSizeVector:

    # create the sample
    sample = stats.expon.rvs(scale=1/1.2,size=sampleSize)

    # calculate the mean and add to the sample mean result list
    sampleMeanVector.append(sample.mean())

# create a dataframe from the result and plot the scatter plot
data = pd.DataFrame({'SampleSize':sampleSizeVector,'SampleMean':sampleMeanVector})
data.plot.scatter('SampleSize','SampleMean')
```

Out[55]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a224b1080>
```



Question 9

Note: For this question you may need to **Google** in order to find commands necessary for some parts of the question.

1. Import `sklearn` library and from it import `datasets`. Aslso, set seed as in **Question 4**.
2. From `sklearn.datasets` load the dataset called "Boston". Read the documentaion of `sklearn.datasets` in order to understand the structure of datasets built in the library.
3. From the dataset `boston` extract the part called `data`. (`boston` is in the form of **dictionary** and includes different parts. You need to extract only the part called `data`)
4. Find the mean and standard deviation of each variable (=feature, column) in the data.
5. Assume the data you have are the whole population. Randomly sample 300 entries from the 11-th variable. Using this sample estimate estimate the population mean (whose true value is already calculated).
6. Provide a confidence interval with 95% of confidence level. (To find the corresponding z-value you can use `stats.norm.ppf()`)
7. If you repeat Steps 5 and 6 above 20 times, how many of these 20 confidence intervals do you expect to include the true mean? Why?
8. Scatterplot the estimated means with their 95% margins of error, as well as the true value of the mean. How many of the error margins include the true mean? Does it match with your answer to the previous step? If not, what is the reason in your opinion?

Answer 9-1

In [56]:
```
from sklearn.datasets import load_boston
```

Answer 9-2

In [57]:
```
boston = load_boston()
```

Answer 9-3

In [58]:
```
boston_data = boston['data']
```

Answer 9-4

Mean and standard deviation of each variable

In [59]:
```
meanOfColumns = boston_data.mean(axis=0)
print('Means of the columns')
print(meanOfColumns)
```

```
Means of the columns
[3.61352356e+00 1.13636364e+01 1.11367787e+01 6.91699605e-02
 5.54695059e-01 6.28463439e+00 6.85749012e+01 3.79504269e+00
 9.54940711e+00 4.08237154e+02 1.84555336e+01 3.56674032e+02
 1.26530632e+01]
```

```
In [60]: stdOfColumns = boston_data.std(axis=0)
         print('Standard deviations of the columns')
         print(stdOfColumns)
```

```
Standard deviations of the columns
[8.59304135e+00 2.32993957e+01 6.85357058e+00 2.53742935e-01
 1.15763115e-01 7.01922514e-01 2.81210326e+01 2.10362836e+00
 8.69865112e+00 1.68370495e+02 2.16280519e+00 9.12046075e+01
 7.13400164e+00]
```

Answer 9-5

```
In [61]: # the index of the 11th column is 10
         bostonSample = np.random.choice(a=boston_data[:,10],size=300)
```

```
In [62]: bostonSampleMean = bostonSample.mean()
         bostonSampleStd = bostonSample.std()
```

```
In [63]: print('The mean of the sample is : ' +str(bostonSampleMean))
         print('The std of the sample is : ' +str(bostonSampleStd))
```

```
The mean of the sample is : 18.414333333333333
The std of the sample is : 2.21534674988414
```

```
In [64]: realBostonMean = meanOfColumns[10]

         print('The real mean of the whole population is : ' + str(realBostonMean))
```

```
The real mean of the whole population is : 18.455533596837967
```

Answer 9-6

1 - alpha = .95

alpha = .05

alpha/2 = .025

interval = Z alpha/2 * std / n ** 0.5

```
In [65]: Z_alpha_2 = stats.norm.ppf(q=.975,loc=bostonSampleMean,scale=bostonSampleStd)
         interval = (Z_alpha_2 * bostonSampleStd) / (math.sqrt(300))
         print('The 95% confidence interval is : ' + str(interval))
```

```
The 95% confidence interval is : 2.910605654364227
```

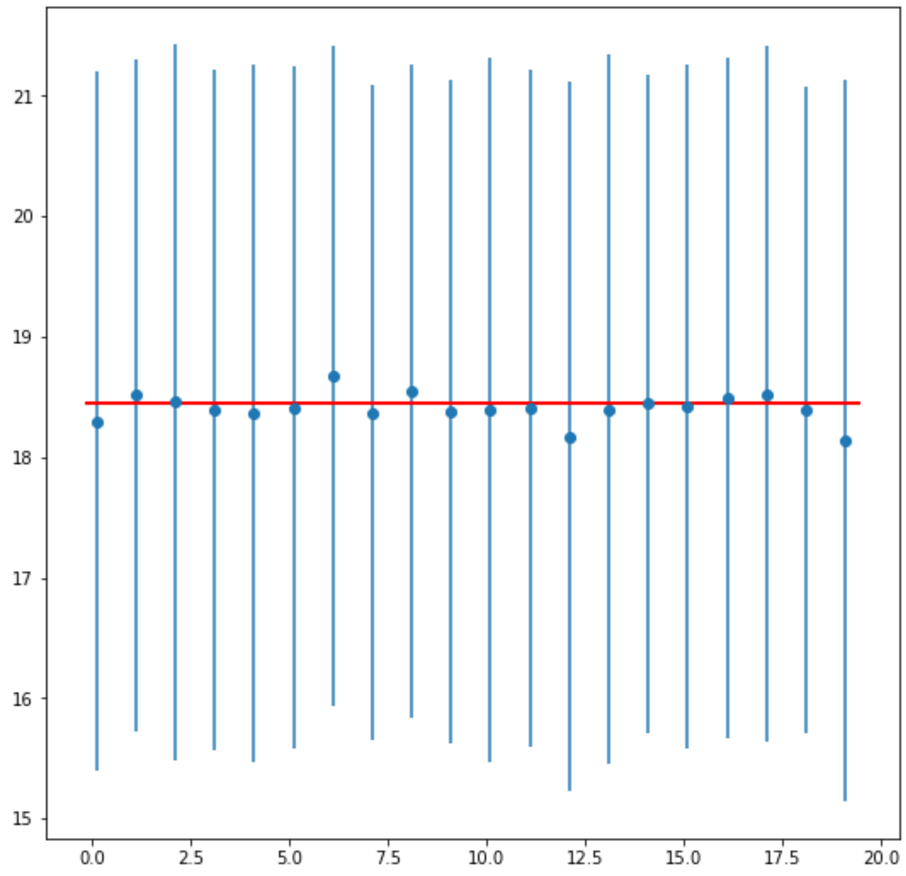
Answer 9-7

The 95% confidence interval calculated in the sample above 2.787807617078 is relatively larger than the real standard deviation of the whole population 2.16280519. Hence, I would expect that all of the confdence intervals will include the true mean.

Answer 9-8

In [66]:
```
bostonSampleMeansVector = []
bostonConfIntervalVector = []

for i in range(20):

    bostonSample = np.random.choice(a=boston_data[:,10],size=300)

    bostonSampleMean = bostonSample.mean()
    bostonSampleStd = bostonSample.std()

    Z_alpha_2 = stats.norm.ppf(q=.975,loc=bostonSampleMean,scale=bostonSampleStd)


    interval = (Z_alpha_2 * bostonSampleStd) / (math.sqrt(300))

    bostonSampleMeansVector.append(bostonSampleMean)


    bostonConfIntervalVector.append([bostonSampleMean+interval,bostonSampleMean-in
terval])
```

```
In [67]: plt.figure(figsize=(9,9))

         plt.errorbar(x=np.arange(0.1, 20, 1),
                      y=bostonSampleMeansVector,
                      yerr=[(top-bot)/2 for top,bot in bostonConfIntervalVector],
                      fmt='o')

         plt.hlines(xmin=-0.2, xmax=19.5,
                    y=realBostonMean,
                    linewidth=2.0,
                    color="red");
```



As expected the real mean is always within the confdence intervals of the samples