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 [1]:
```
# 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 : $\{\text{John, Michael, Ahmed, Tiffany}\}$

In [2]:
```
list1 = ['John', 'Michael', 'Vanessa', 'Ahmed', 'Tiffany']
list2 = ['Cyrus','Vanessa']

#using set
set_difference = list(set(list1) - set(list2))
print(set_difference)

# iterating throgh elements
set_difference1 = [i for i in list1 + list2 if i in list1 and i not in list2]
print(set_difference1)

#using pandas or numpy
```

```
['Ahmed', 'John', 'Tiffany', 'Michael']
['John', 'Michael', 'Ahmed', '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!"

In [3]:
```
def function_gcd(x, y):
    if x > 0 and y > 0:
        print("Inputs are positive integers")
        while(y):
            x, y = y, x % y
        return x
    else:
        print("This function takes only positive integers!")

ip1 = int(input("Enter first number: "))
ip2 = int(input("Enter 2nd number: "))
print("The  greatest common divisor between " + str(ip1) + " and " + str(ip2) +
```

```
Enter first number: 45
Enter 2nd number: 15
Inputs are positive integers
The  greatest common divisor between 45 and 15 is: 15
```

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.

In [4]:
```
#prime numbers in the interval  [0,p] ,
def function_prime(p):
    for num in range(0,p + 1):
        if num > 1:
            for i in range(2,num):
                if (num % i) == 0:
                    break
            else:
                print(num)

p = int(input("Enter upper range: "))
function_prime(p)
```

```
Enter upper range: 12
2
3
5
7
11
```

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 $\{\text{Female}, \text{Male}\}$ 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.

In [5]:
```
import random
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
```

In [6]:
```
#1.Set seed by the initial value  1231  and define the following variables (Pytho
# size  ⟵1000 ,n  ⟵700 ,p  ⟵0.3

random.seed(1234)
size = 1000
n = 700
p = 0.3
```

In [7]:
```
#2.Randomly generate size number of integers in  (0,200)  and save them as col1
col1 = [random.randint(1,200) for x in range(size)] #from (1,200) instead as 0 wi
#print(col1)
```

In [8]:
```
#3.Randomly generate size number of values according to  Unif[0,1]  and call it c
col2 = [random.uniform(0,1) for x in range(size)]
#print(col2)
```

In [9]:
```
#4.Randomly generate 2*size numbers from  Binom(  n,p  )  and randomly select siz
col3 = [random.choice(np.random.binomial(n, p, 2*size)) for x in range(size)]
#np.random.binomial
```

In [10]:
```
#5.Define the following functions

#funct1(𝑥)=ln𝑥
def funct1(list1):
    retList = []
    for x in list1:
        retList.append(math.log(x))
    return retList
#math.log(0) is error

#funct2(𝑥)=10exp(𝑥)1+exp(𝑥)
def funct2(list2):
    retList = []
    for x in list2:
        retList.append((10 * math.exp(x)) / (1+math.exp(x)))
    return retList

#funct3(𝑥)=350/100**2𝜋√.exp(−𝑥220000)
def funct3(list3):
    retList = []
    for x in list3:
        retList.append((350/(100*math.sqrt(2*math.pi)))*(math.exp(-(math.pow(x,2
    return retList
```

In [11]:
```
#6.Define
col4 = funct1(col1)
col5 = funct2(col2)
col6 = funct3(col3)
```

In [12]:
```
#7.Randomly generate size number of genders from the set  {Female, Male}  and sa
col7 = [random.choice(['Female', 'Male']) for x in range(size)]
```

In [13]:
```
#8.Construct a data frame with  7  columns col1 to col7 and call it mydata.
data = {'col_1': col1, 'col_2': col2,'col_3':col3,'col_4':col4,'col_5':col5,'col_
mydata = pd.DataFrame.from_dict(data)
print(mydata.head())
```

```
   col_1     col_2  col_3     col_4     col_5     col_6   col_7
0    200  0.423387    204  5.298317  6.042935  0.174300    Male
1    113  0.625016    192  4.727388  6.513584  0.221048  Female
2     30  0.701010    201  3.401197  6.684116  0.185217  Female
3      2  0.655179    192  0.693147  6.581767  0.221048    Male
4     24  0.379352    210  3.178054  5.937169  0.153943  Female
```

In [14]:
```
mydata.describe()
```

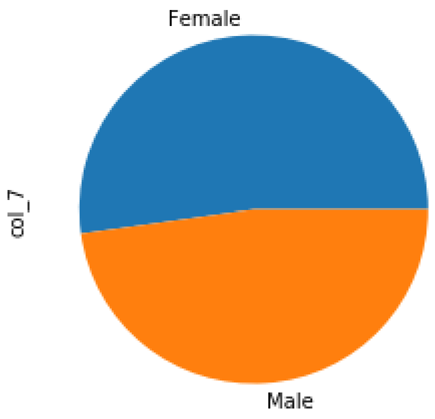
Out[14]:

	col_1	col_2	col_3	col_4	col_5	col_6
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	101.988000	0.504844	210.079000	4.338802	6.212889	0.157324
std	57.368083	0.285791	11.809751	0.935959	0.664946	0.038774
min	1.000000	0.004128	169.000000	0.000000	5.010319	0.072903
25%	52.000000	0.261979	202.000000	3.951244	5.651227	0.129720
50%	104.000000	0.515328	210.000000	4.644391	6.260545	0.153943
75%	151.000000	0.748347	218.000000	5.017280	6.788184	0.181523
max	200.000000	0.999477	243.000000	5.298317	7.309558	0.334799

9.Describe the dataset using the descriptive statistics discussed in the class.

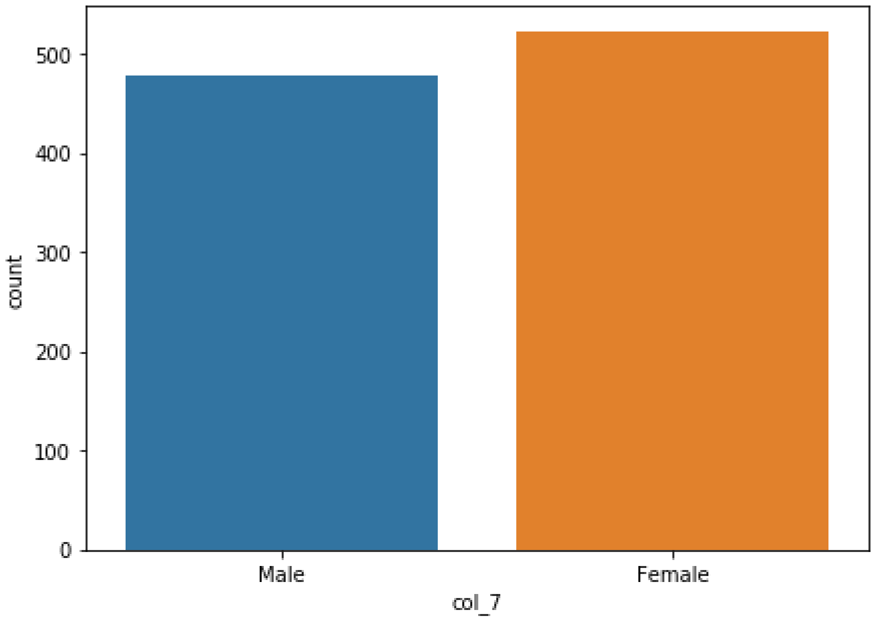
Ans: There are total 1000 observations with 7 variables in the dataset. outof this col_1 to col_6 are continuous & col_7 is categorical. col_1: the range is in between 1 to 200. The mean value is 101.988 and median is 104. col_2: the range is in between 0.004128 to 0.999477. The mean value is 0.504844 and median is 0.515328. col_3: the range is in between 170 to 251. The mean value is 4.338802 and median is 210. col_4: the range is in between 0 to 5.298317. The mean value is 101.988 and median is 4.644391. col_5: the range is in between 5.010319 to 7.309558. The mean value is 6.212889 and median is 6.260545. col_6: the range is in between 0.059831 to 0.329172. The mean value is 0.157877 and median is 0.153943.

In [15]:
```
#10.Use the appropriate visualisation tool to visualize each variable in the datc

mydata['col_7'].value_counts().plot(kind='pie');
```



In [16]:
```
fig, ax = plt.subplots(figsize=[7, 5])
sns.countplot(x = "col_7", data = mydata)
```

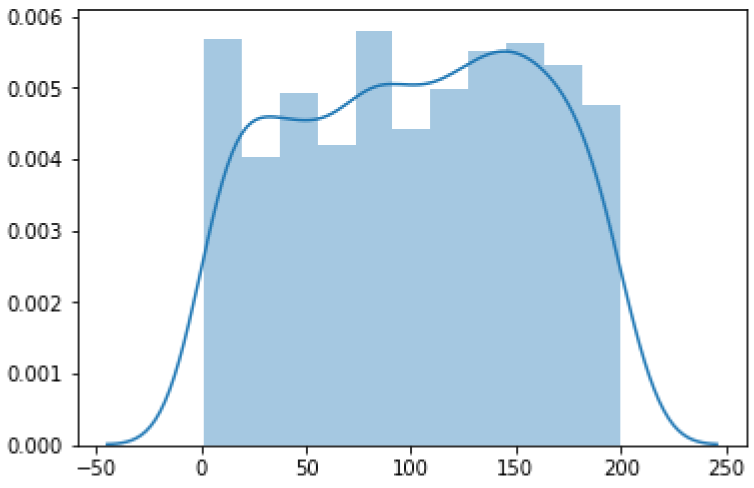
Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x54145ef860>
```



In [17]:
```
sns.distplot(mydata[['col_1']])
```

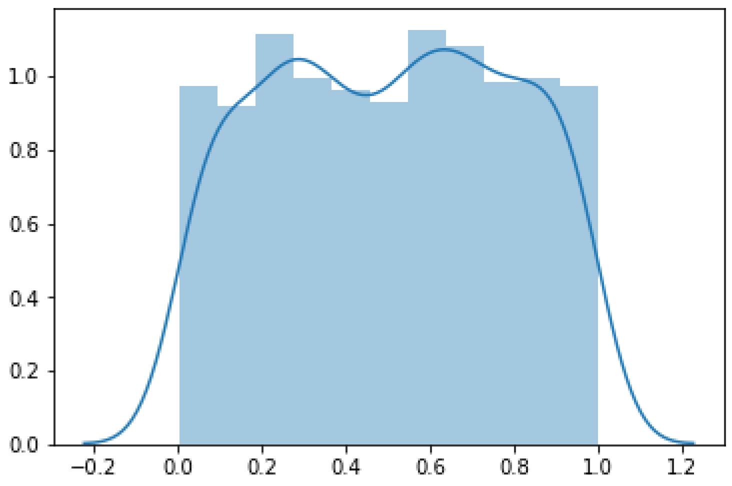
```
C:\Users\mana\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarn
ing: Using a non-tuple sequence for multidimensional indexing is deprecated; us
e `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpret
ed as an array index, `arr[np.array(seq)]`, which will result either in an erro
r or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x541461e828>
```



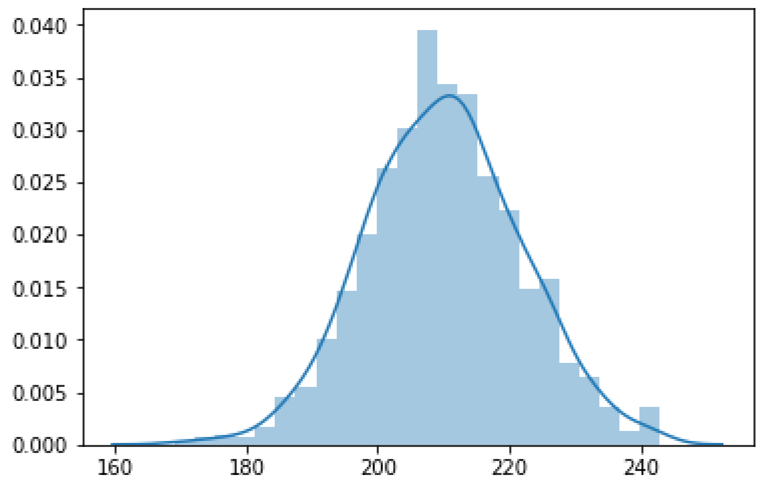
In [18]:
```
sns.distplot(mydata[['col_2']])
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x54145bdc50>
```



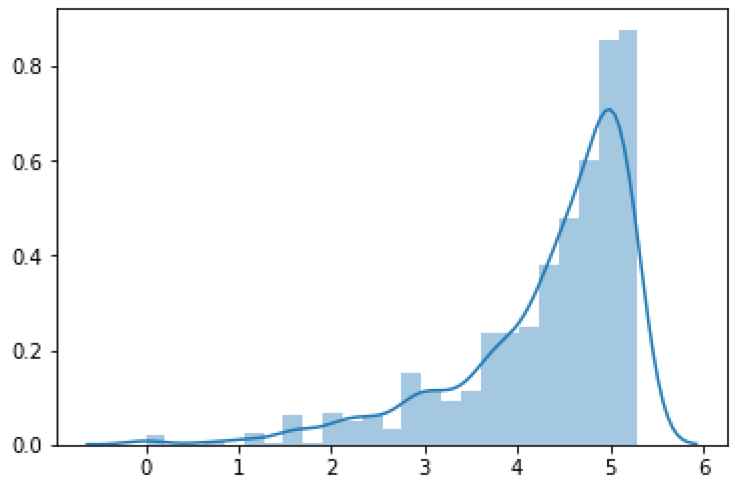
```
In [19]: sns.distplot(mydata[['col_3']])
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x54159c1048>
```



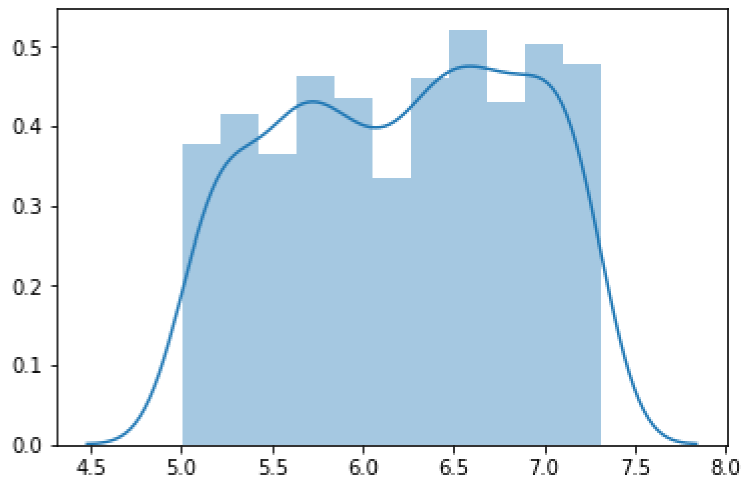
```
In [20]: sns.distplot(mydata[['col_4']])
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x5415a61358>
```



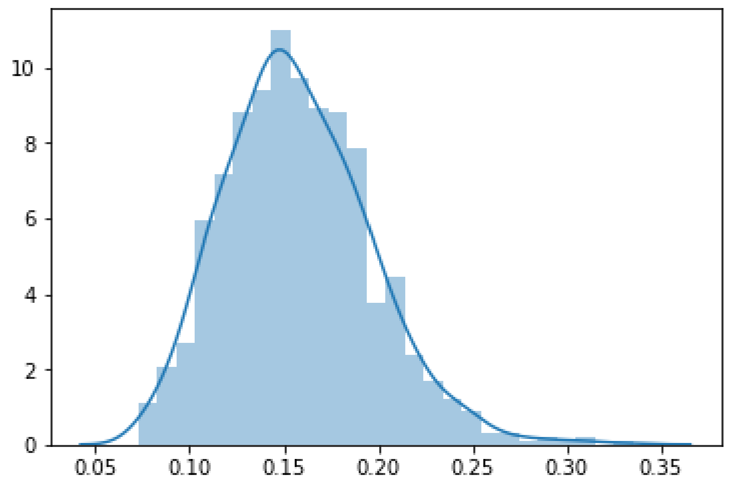
In [21]:
```
sns.distplot(mydata[['col_5']])
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x5415aba9b0>
```

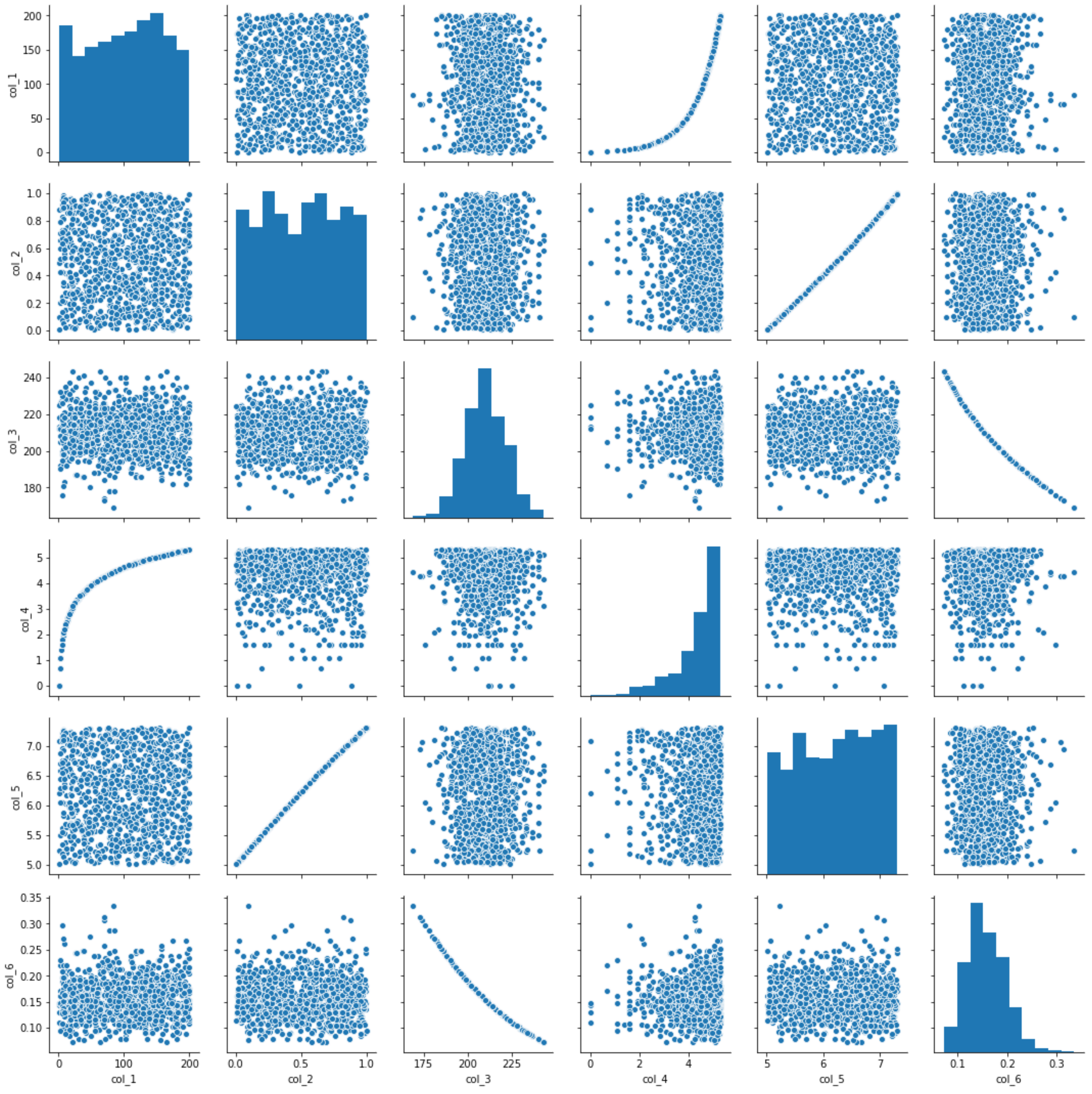


In [22]:
```
sns.distplot(mydata[['col_6']])
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x5415b45ba8>
```



In [55]:
```
#11.Using loops scatterplot every pair of columns versus each other if appropria
g = sns.pairplot(mydata, kind = "scatter")
```



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?

In [24]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats  as stats
import math
```

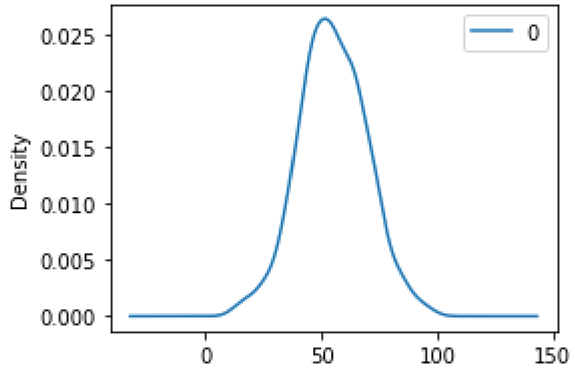
In [25]:
```
#1.Assume that  X∼N(μ,σ2)  with  μ=55,σ=15 . Randomly generate a set of  1000,00
#according to the given distribution and call it set  D .
np.random.seed(123)
sample_size = 1000

#1.Assume that  X∼N(μ,σ2) (normal distribution) with  μ=55,σ=15
#Randomly generate a set of 1000,000 values for X according to the given distrib
mu = 55
sigma = 15
D = stats.norm.rvs(size=1000000,  # Generate 1000000 numbers
                   loc = mu,
                   scale = sigma)
print(D)
```

```
[38.71554095 69.9601817  59.24467747 ... 42.28479327 43.92714079
 35.60840992]
```

In [26]:
```
#2.Pretend that  D  is your whole population. Choose a sample of  1000  values f
sample_s = np.random.choice(a=D,size=sample_size)
print(sample_s)
```

```
[56.58120253 67.92922894 55.19918441 48.24473337 35.69247838 78.10249866
 50.29612829 60.31423474 53.7244298  46.97961229 32.60578552 37.96654239
 49.36395733 55.74081049 49.79119217 41.10328683 64.8064774  50.06357603
 50.75309315 52.06628662 53.98728573 62.7926766  73.6998135  58.98097269
 42.90351024 61.41610725 47.89914735 49.19823266 47.45764815 36.74050014
 45.27422334 83.34162858 56.39933875 59.76341953 52.4585949  49.95330971
 58.32950112 20.40094716 52.91253396 66.53634452 96.73211993 55.07106865
 49.22374974 82.97484171 55.61467833 70.55124848 76.52047979 54.30046746
 52.1861275  80.93806482 33.55043422 43.38513804 39.03129535 62.10964126
 51.56059891 54.37953996 43.1788329  41.72115479 66.07901406 39.85194871
 32.93207613 29.61255604 68.72125704 52.85378182 62.7817657  54.91049742
 50.00770672 58.67689693 71.89341546 54.55913456 74.45351591 46.18989279
 66.75578129 51.05960582 49.13543699 56.53624921 92.28927373 78.24882167
 67.4418511  37.71002125 26.70126418 54.09381833 69.66446403 45.16110784
 72.83592296 49.96027178 45.18404519 36.35882727 40.89710783 76.35415947
 50.44270878 47.16621367 46.00735512 67.01783488 46.13705889 60.4744039
 63.88778614 48.25135105 92.59635301 45.18348128 40.78011505 92.48292199
 51.24165356 57.40696612 63.5440229  34.42790585 76.05298752 59.99393495
 55.4897578  56.08302341 70.44555418 58.96878648 46.37685015 70.53364198
```

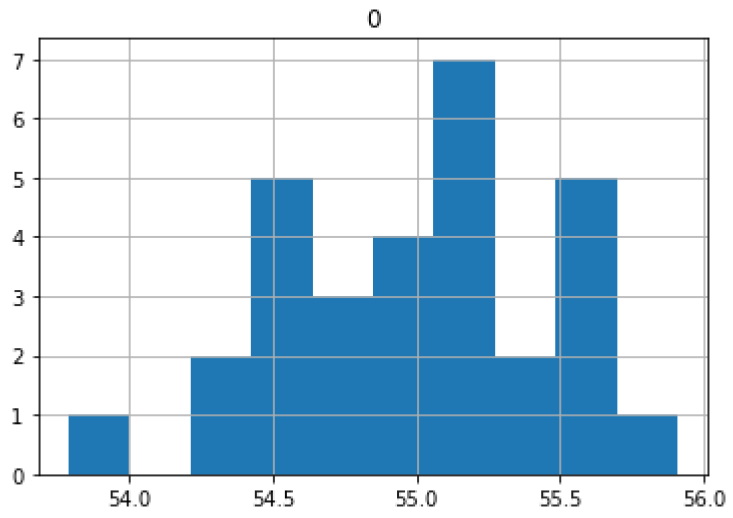
In [27]:
```
#3.Plot an approximate density distribution function using the selected sample.
plot = pd.DataFrame(sample_s).plot(kind="density", figsize=(4,3))
```



In [28]:
```
#4.Using a loop repeat the second step  30  times. For each sample estimate the p
sample_means = []
for x in range(30):
    sample_s = np.random.choice(a=D,size=sample_size)
    sample_means.append(sample_s.mean())

# Calculate the mean squared error of your estimated means.
#mean of the population call this Y_mean
Y_mean = stats.norm.mean(loc=mu, scale=sigma)
print(Y_mean)
#print(sample_means)
samplemeanerr_square = (sample_means - Y_mean) ** 2
MSE = sum(samplemeanerr_square)/len(samplemeanerr_square)
print("Mean squared error of the estimated means is :", MSE)
```

```
55.0
Mean squared error of the estimated means is : 0.22740779024496535
```

In [29]:
```
#5.Plot the histogram of all  30  saved sample means (Only sample means)
hist = pd.DataFrame(sample_means).hist()
```



6.If instead of 500 times, we resample over and over for a large number of times, how does the sample mean change?

Ans: If we resample over and over for a large number of times, the sample means are normally distributed around the population mean.

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?

1.Which distribution is it and what are the parameters of the distribution? Is it a discrete or continuous distribution?

Ans:Fliping a coin, possible outcomes are head/tail. Therefore it is discrete. As it is taking only 2 values (Bernoulli Distribution), however a random variable X is said to be binomial if it represents the number of successes in n independent Bernoulli trials. It is denoted by X ~ Binom(n; p), Mean: E(X) = np, Variance: Var(X) = np(1 − p)

```
In [30]: import random
         import numpy as np
         import pandas as pd
         import math
         from scipy.stats import binom
```

2.What is the probability of obtaining 10 heads? Explain how to calculate it.

Ans:p(1 head) = 0.6, q(1tail) = 1-0.6 = 0.4 P(k out of n) = (n!/k!(n-k)!) * (p ** k) * ((1-p) ** (n-k)) where n = no of trials, p = probability of success

In [31]:
```
#2.What is the probability of obtaining  10  heads?
n = 15
p = 0.6
k= 10
print("the probability of obtaining  10  heads is :", binom.pmf(k,n,p))
```

```
the probability of obtaining  10  heads is : 0.18593784476467232
```

In [32]:
```
#3.What is the probability of getting more than or equal to  10  heads? What abou
# What should be the summation of these two probabilities and why?

#probability of getting more than or equal to  10  heads
k = np.arange(10,16)
binom10_moreorequal = binom.pmf(k, n, p)
p10_moreorequal = np.sum(binom10_moreorequal)
print("probability of getting more than or equal to  10  heads is :", p10_moreor

#probability of getting less than or equal to  10  head
k = 10
p10_lessorequal = binom.cdf(k,n,p)
print("probability of getting less than or equal to  10  heads is :", p10_lessor

#summation of these two probabilities
print("summation of these two probabilities is:", p10_moreorequal + p10_lessorequ

#probability of getting less than 10  heads
k = np.arange(0,10)
binom10_less = binom.pmf(k, n, p)
p10_less = np.sum(binom10_less)
print("probability of getting less than 10  heads is :", p10_less)
print("Total Probability: ", p10_less+p10_moreorequal)
```

```
probability of getting more than or equal to  10  heads is : 0.4032155504148489
probability of getting less than or equal to  10  heads is : 0.782722294349824
summation of these two probabilities is: 1.185937844764673
probability of getting less than 10  heads is : 0.5967844495851539
Total Probability:  1.0000000000000027
```

In [33]:
```
#4.Find the expected value of obtained number of heads in each trial? (Each trial
print(binom.mean(n, p))
expect_val= binom.expect(args=(n, p))
print("expected value of obtained number of heads in each trial", expect_val )
```

```
9.0
expected value of obtained number of heads in each trial 9.000000000000025
```

In [34]:
```
#5.How probable is it to get (H,H,H,T,T,H,T,T,H,T,H,H,H,T,H)?
p_h = 0.6
p_t= 1- 0.6
print("probable is it to get (H,H,H,T,T,H,T,T,H,T,H,H,H,T,H)",p_h*p_h*p_h*p_t*p_
```

```
probable is it to get (H,H,H,T,T,H,T,T,H,T,H,H,H,T,H) 4.1278242815999987e-05
```

In [35]:
```
#6.Find the first, second, and the third quantiles.
n = 15
p = 0.6
print("The first quantile is :",binom.ppf(0.25, n, p))
print("The second quantile is :",binom.ppf(0.5, n, p))
print("The third quantile is :",binom.ppf(0.75, n, p))
```

```
The first quantile is : 8.0
The second quantile is : 9.0
The third quantile is : 10.0
```

In [36]:
```
#7.Repeat the trial  10  times and estimate the mean each time. Using pandas.cros
#Plot the histogram of these ten estimates

sample_sizes = []
samplemeans = []
for i in range(1,11):
    sample_sizes.append(i) #appending sample size which is 1
    data_binom = binom.rvs(n=15,p=0.6,size=1)
    samplemean = data_binom.mean()
    samplemeans.append(samplemean) # appending sample means
print(samplemeans)
data1 = pd.DataFrame.from_dict({'sample_size': sample_sizes,'samplemean': samplem
pd.crosstab(data1.sample_size,data1.samplemean)
```

```
[8.0, 12.0, 8.0, 12.0, 9.0, 9.0, 9.0, 9.0, 8.0, 12.0]
```

Out[36]:

samplemean	8.0	9.0	12.0
sample_size			
1	1	0	0
2	0	0	1
3	1	0	0
4	0	0	1
5	0	1	0
6	0	1	0
7	0	1	0
8	0	1	0
9	1	0	0
10	0	0	1

In [37]:
```
#8.Now, gradually increase the number of trials from  100  to  1000 . (start fror
#Plot the histogram for the sample mean each time. How does the sampling distribu
max_sample_size = 1000
sample_size1 = 50
sample_sizes = []
samplemeans1 = []
#for i in range(100,1000+50,50):
for i in range(100,max_sample_size + sample_size1,sample_size1):
    sample_sizes.append(i) #appending sample size which is 1
    data_binom = binom.rvs(n=15,p=0.6,size=i)
    sample_mean = data_binom.mean()
    samplemeans1.append(sample_mean) # appending sample means
print(samplemeans1)
print(sample_sizes)
```

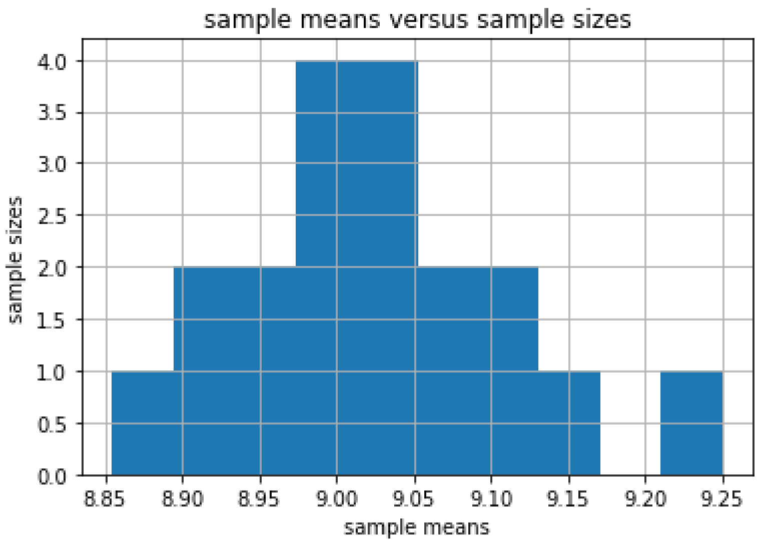
```
[9.25, 9.013333333333334, 8.855, 8.932, 9.036666666666667, 9.06857142857143, 8.
9775, 9.117777777777778, 8.916, 9.136363636363637, 8.938333333333333, 9.0430769
23076923, 9.117142857142857, 8.950666666666667, 9.0325, 8.98235294117647, 9.005
555555555556, 8.993684210526316, 9.086]
[100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 85
0, 900, 950, 1000]
```

In [38]:
```
##Plot the histogram for the sample mean each time. How does the sampling distril
#plt.hist(samplemeans1, normed=True, bins=5)
hist = pd.DataFrame(samplemeans1).hist()
plt.title('sample means versus sample sizes')
plt.xlabel('sample means')
plt.ylabel('sample sizes')
plt.show()
```

Out[38]:
```
Text(0.5, 1.0, 'sample means versus sample sizes')
```

Out[38]:
```
Text(0.5, 0, 'sample means')
```

Out[38]:
```
Text(0, 0.5, 'sample sizes')
```



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.

The target of the study is a particular disease (either having disease or not: success or failuer, each patients has the disease or not)(therefore it's a Bernoulli distribution). Based on this we can consider binomial distibution (as summing up the Bernoulli) to calculate further.

Probability of sampling a person with this disease p(X)= 0.007. We need the sample size (parameter n which is unknown) to calculate the results. We can try considering some probable value for this.

Q.1. How many people we need to sample totally in order to get exactly 73 patients with the disease?

Ans: If we consider p(X) = 1 , then we can conclude that exactly 73 patients have the disease. However in reality we can never reach 1, therefore we can consider aproximating it to 1, using various 'n' values.

In [39]:
```
import random
import numpy as np
import pandas as pd
import math
from scipy.stats import binom

#n= ?
p = 0.007
k = 73
#we can start considering the expected value np = 73
n = 73/p
print(n)
for i in range (10428,0,-1):
    if binom.cdf(k,i,p) == 0.9999999999999999:
        break

print("approximate value for n(sample size) is: ", i)
```

```
10428.571428571428
approximate value for n(sample size) is:  3431
```

In [40]:
```
#2.How many people we need to sample totally in order to have at least  73  patie
#atleast 73 means it may be 73, 74, 75 ...(summing it up)
```

Q3.What distribution is it and what are its parameters

Ans: The target of the study is a particular disease (either having disease or not: success or failuer, each patients has the disease or not)(therefore it's a Bernoulli distribution). Based on this we can consider binomial distibution (as summing up the Bernoulli) to calculate further.

Parameters are

p= 0.007

k= 73

n = ?is unknown

In [41]:
```
#4.Callculate the expected value, variance, as well as the first, second and the
n = 3431
p = 0.007
print("expected value :", binom.expect(args=(n , p)))
print("Variance :", binom.var(n , p))
print("The first quantile is :",binom.ppf(0.25, n, p))
print("The second quantile is :",binom.ppf(0.5, n, p))
print("The third quantile is :",binom.ppf(0.75, n, p))
```

```
expected value : 24.016999999963794
Variance : 23.848881
The first quantile is : 21.0
The second quantile is : 24.0
The third quantile is : 27.0
```

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?

In [42]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats  as stats
import random
```

In [43]:
```
#1.Generate  1000  data points according to the exponential distribution with par
sample_size = 1000
sample_means = []

sample_scale = 1/1.2
print(sample_scale)
sample_datapoint = stats.expon.rvs(scale=sample_scale,loc=0,size=sample_size)

```

```
0.8333333333333334
```

In [44]:
```
#2.Estimate the mean of the sample.
sample_mean = sample_datapoint.mean()
print(sample_mean)
```

```
0.8293361809632063
```

In [45]:
```
#3.Repeat the first step but each time increase the sample size up to  1000,000
#Scatterplot the mean versus the sample size for each repetition.

max_sample_size = 1000000
sample_sizes = []
samplemeans = []
#for i in range(1000,1000000+1000,1000):
for i in range(sample_size,max_sample_size + sample_size,sample_size):
    sample_sizes.append(i) #appending sample sizes

    sample_dp = stats.expon.rvs(scale=sample_scale,loc=0,size=i)
    samplemean = sample_dp.mean()
    samplemeans.append(samplemean) # appending sample means
#print(sample_sizes)
#print(samplemeans)
```

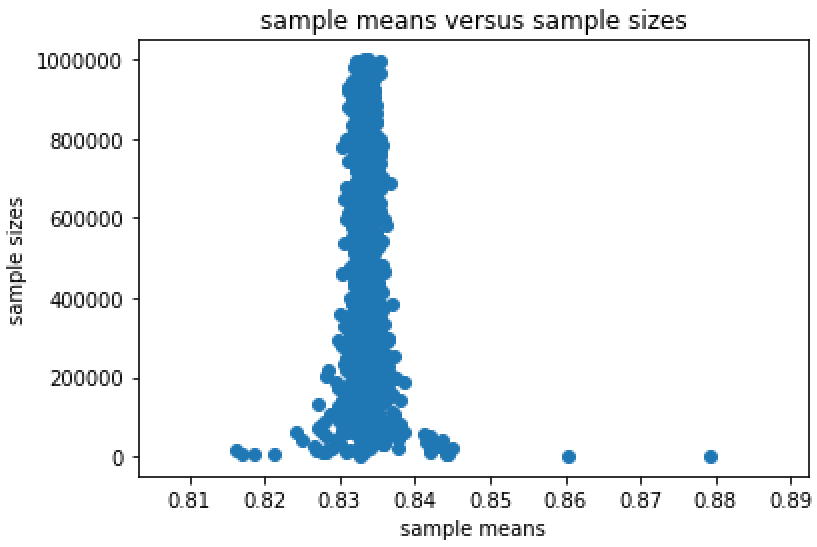
In [46]:
```
#Scatterplot the mean versus the sample size for each repetition
plt.scatter(samplemeans, sample_sizes)
plt.title('sample means versus sample sizes')
plt.xlabel('sample means')
plt.ylabel('sample sizes')
```

Out[46]:
```
<matplotlib.collections.PathCollection at 0x54161d4c18>
```

Out[46]:
```
Text(0.5, 1.0, 'sample means versus sample sizes')
```

Out[46]:
```
Text(0.5, 0, 'sample means')
```

Out[46]:
```
Text(0, 0.5, 'sample sizes')
```



Q.Do you see any trend in the plot sample means? Can you guess the limit of the sample mean due to the plot?

Ans:The sample mean remain similar by increasing the sample size (by increasing to a large number). the limit of the sample mean is in between 0.82 to 0.85

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?

In [47]:
```
import numpy as np
import pandas as pd
import random
import scipy.stats as stats
import math
```

In [48]:
```
#1.Import sklearn library and from it import datasets. Aslso, set seed as in Ques
#2.From sklearn.datasets load the dataset called "Boston". Read the documentaion
# understand the structure of datasets built in the library.

from sklearn.datasets import load_boston
np.random.seed(1234)

boston = load_boston()
print(boston.data.shape)
```

```
(506, 13)
```

In [49]:
```
#3.From the dataset boston extract the part called data. (boston is in the form c
# You need to extract only the part called data)
boston_data = pd.DataFrame(boston.data, columns = boston.feature_names)
print(boston_data.head())
#boston['MEDV'] = dataset.target
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

In [50]:
```
#4.Find the mean and standard deviation of each variable (=feature, column) in tl
boston_data.describe()

mean_boston = boston_data.mean()
#print(boston_data.mean())
#print(boston_data.std())
```

Out[50]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

In [51]:
```
#5.Assume the data you have are the whole population. Randomly sample  300  entr
# Using this sample estimate estimate the population mean (whose true value is a
sample_size = 300
# 11th Variable is ['PTRATIO']
boston_PRTATIO = boston_data['PTRATIO']
sample_PRTATIO = np.random.choice(a= boston_PRTATIO, size = sample_size)
samplemean_PRTATIO = sample_PRTATIO.mean()
true_mean_PTRATIO = boston_PRTATIO.mean()
print('true mean for boston[PTRATIO] is: ', true_mean_PTRATIO)
print('Sample mean for boston[PTRATIO] is: ', sample_mean)
```

```
true mean for boston[PTRATIO] is:  18.455533596837967
Sample mean for boston[PTRATIO] is:  0.8293361809632063
```

In [52]:
```
#6.Provide a confidence interval with  95 % of confidence level.
#(To find the corresponding z-value you can use stats.norm.ppf())

z_critical = stats.norm.ppf(q = 0.975)
print("z-critical value: ", z_critical)

boston_PRTATIO_stdev = boston_PRTATIO.std()
margin_of_error = z_critical * (boston_PRTATIO_stdev/math.sqrt(sample_size))

confidence_interval = (samplemean_PRTATIO - margin_of_error,
                       samplemean_PRTATIO + margin_of_error)
print("95% Confidence interval for the mean: " , confidence_interval)
```

```
z-critical value:  1.959963984540054
95% Confidence interval for the mean:  (18.221351186364597, 18.711315480302066)
```

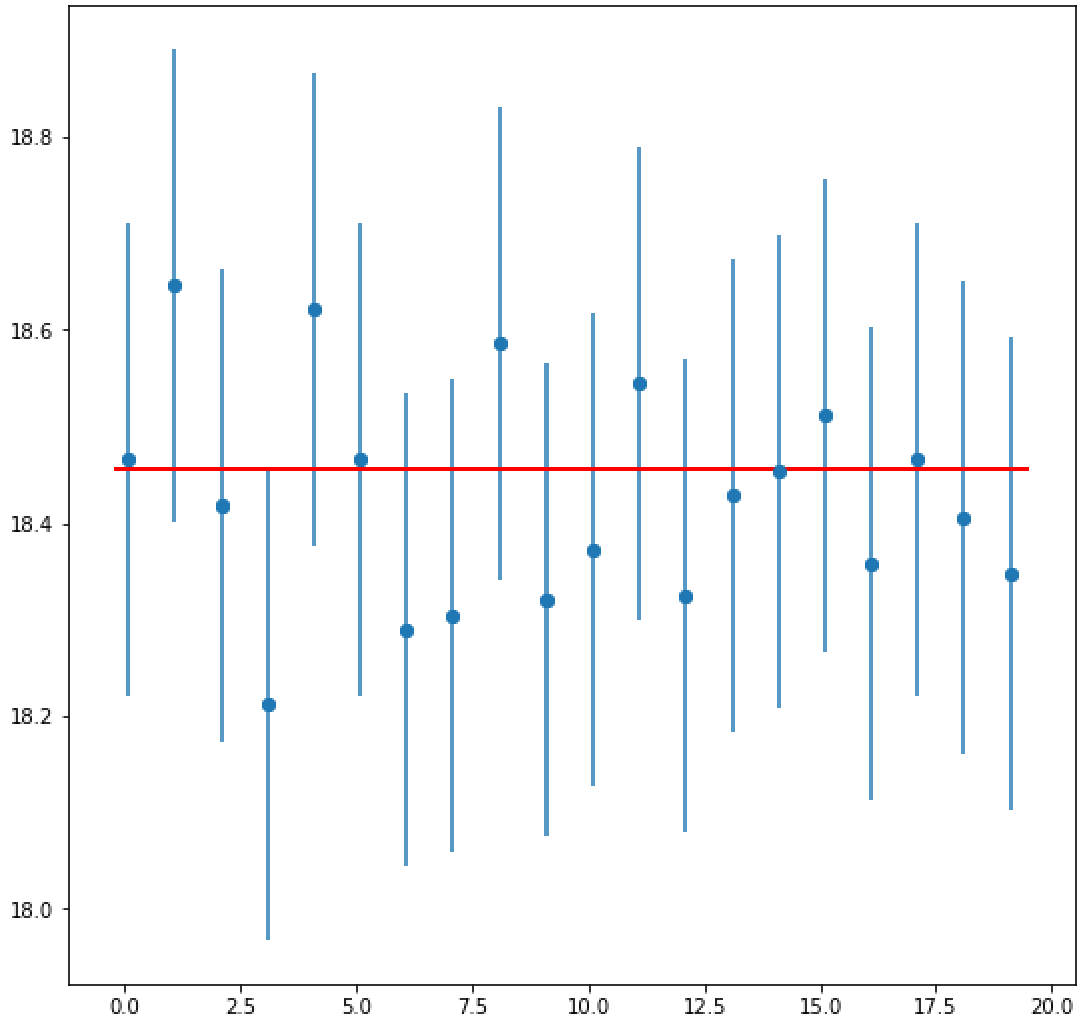
In [53]:

```
#7.If you repeat Steps 5 and 6 above  20  times, how many of these  20  confiden
# include the true mean? Why?
np.random.seed(1234)
intervals = []
samplemeans_PRTATIO = []

for sample in range(20):  # We want 20 different confidence intervals
    sample1 = np.random.choice(a= boston_PRTATIO, size = sample_size)
    samplemean1 = sample1.mean()
    samplemeans_PRTATIO.append(samplemean1)

    z_critical1 = stats.norm.ppf(q = 0.975)
    boston_PRTATIO_stdev1 = boston_PRTATIO.std()
    margin_of_error1 = z_critical1 * (boston_PRTATIO_stdev1/math.sqrt(sample_siz

    confidence_interval1 = (samplemean1 - margin_of_error1,
                            samplemean1 + margin_of_error1)
    intervals.append(confidence_interval1)

print('No of confidence intervals',len(intervals))
print(intervals)
```

```
No of confidence intervals 20
[(18.221351186364597, 18.711315480302066), (18.402017853031264, 18.891982146968
733), (18.1733511863646, 18.663315480302067), (17.968684519697934, 18.458648813
635403), (18.377017853031266, 18.866982146968734), (18.22201785303126, 18.71198
214696873), (18.04468451969793, 18.5346488136354), (18.060017853031265, 18.5499
82146968734), (18.342017853031265, 18.831982146968734), (18.075684519697933, 1
8.565648813635402), (18.127017853031266, 18.616982146968734), (18.300351186364
6, 18.79031548030207), (18.080684519697932, 18.5706488136354), (18.183017853031
263, 18.67298214696873), (18.20768451969793, 18.6976488136354), (18.26568451969
7928, 18.755648813635396), (18.11268451969793, 18.602648813635398), (18.2213511
86364597, 18.711315480302066), (18.161017853031264, 18.650982146968733), (18.10
3351186364595, 18.593315480302063)]
```

In [54]:
```python
#8.Scatterplot the estimated means with their  95 % margins of error, as well as
import matplotlib.pyplot as plt

plt.figure(figsize=(9,9))
plt.errorbar(x=np.arange(0.1, 20, 1),
             y=samplemeans_PRTATIO,
             yerr=[(top-bot)/2 for top,bot in intervals],
             fmt='o')
plt.hlines(xmin=-0.2, xmax=19.5,
           y=18.455533596837967,
           linewidth=2.0,
           color="red");
plt.scatter(x=np.arange(0.1, 20, 1), y=samplemeans_PRTATIO)
```

Out[54]:
```
<Figure size 648x648 with 0 Axes>
```
Out[54]:
```
<ErrorbarContainer object of 3 artists>
```
Out[54]:
```
<matplotlib.collections.LineCollection at 0x54187a2128>
```
Out[54]:
```
<matplotlib.collections.PathCollection at 0x54187a20b8>
```



Q.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

Ans: out of 20, 19 error margins mostly include the true mean. One just touched the boundary. Therefore it matches our answer in the previous question