CSN-373: Probability Theory for Computer Engineers

Statistical Analysis with Python Programming

Pratyush Kumar
20114076

Problem Statement 1

Ques 1.

Using Pandas for importing dataset.

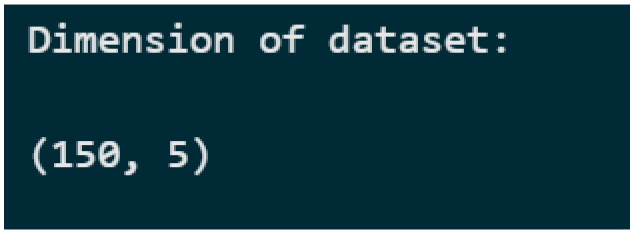
```
data = pd.read_csv('iris.data.csv', header=None, names=['Sepal length','Sepal width','Petal length','Petal width','Species'])
data
```

Ques 2.

Dimensions of dataset:

```
print("Dimension of dataset: ")
data.shape
```

Output:

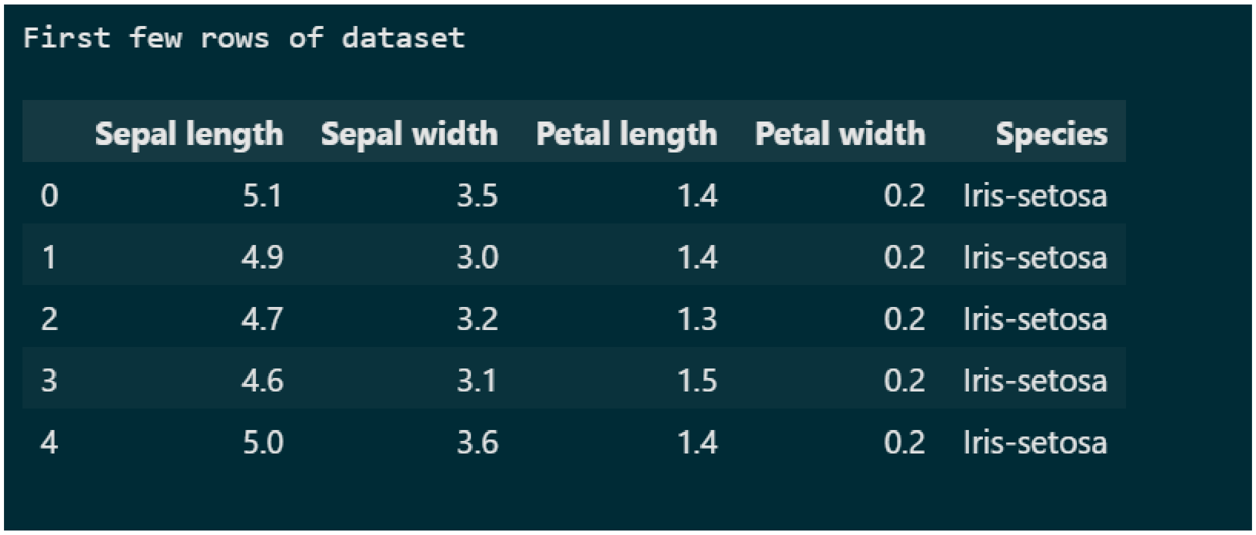


```
Dimension of dataset:

(150, 5)
```

First Few Rows:

```
print("First few rows of dataset")
data.head()
```

Output:



```
First few rows of dataset
```

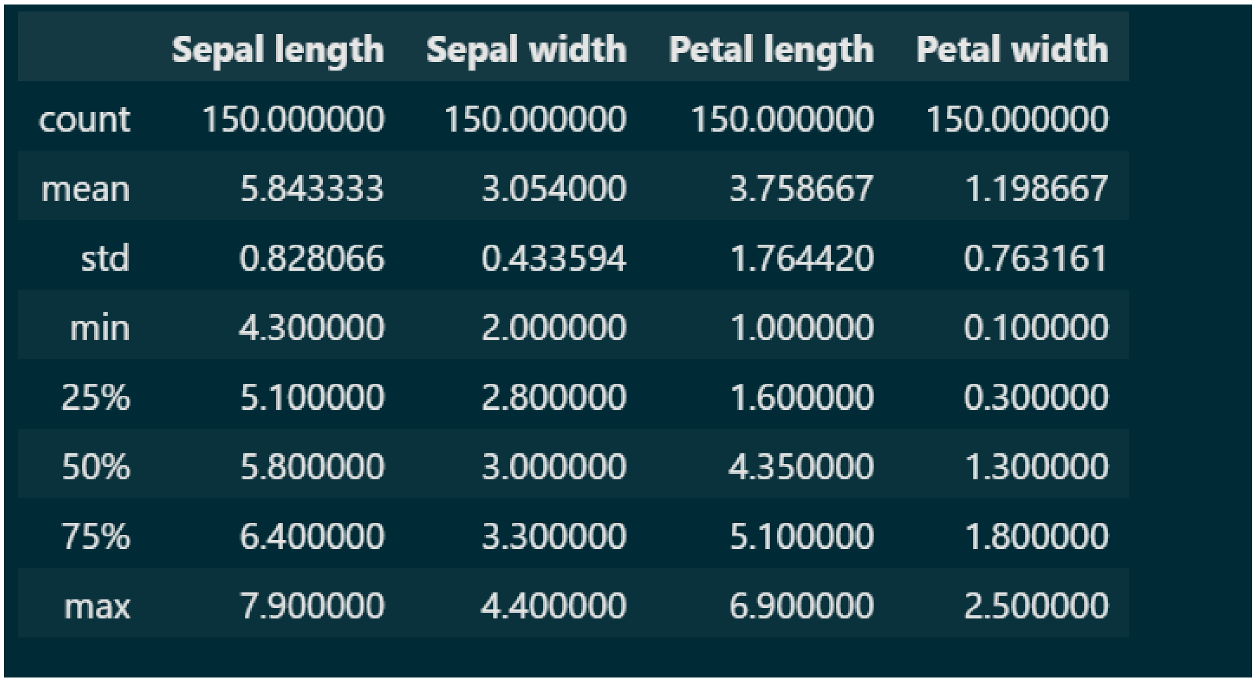
	Sepal length	Sepal width	Petal length	Petal width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Ques 3.

Basic Stats:

```
data.describe()
```

Output:

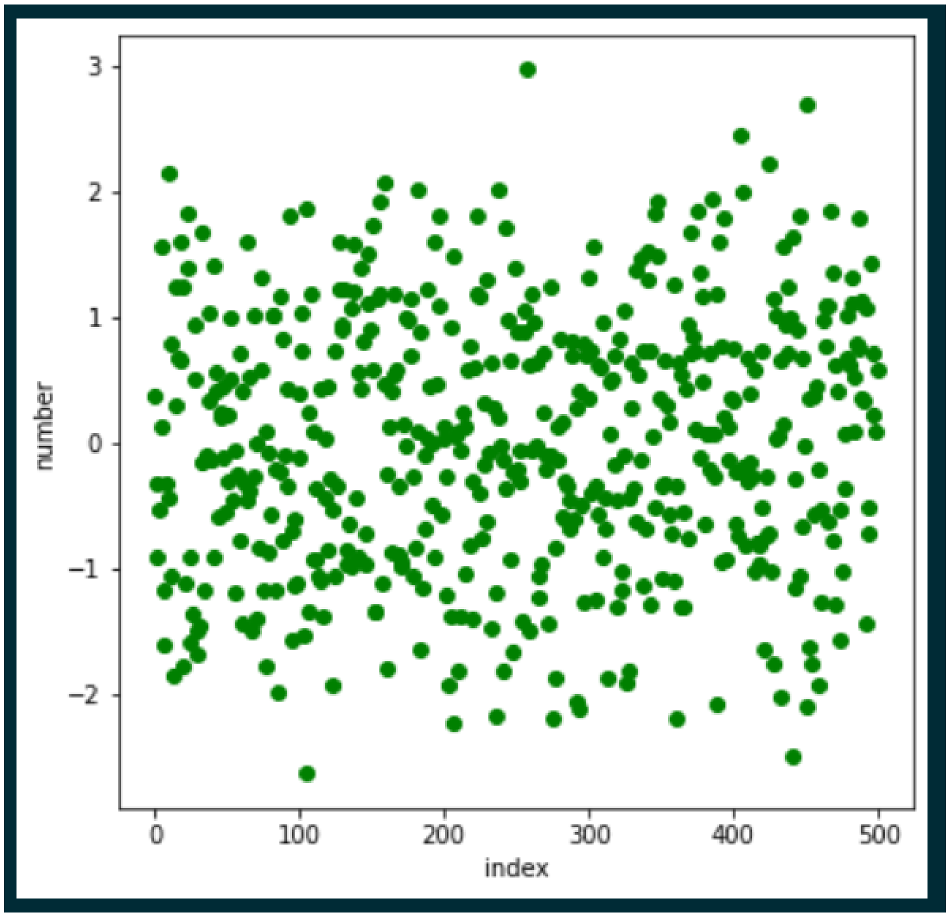


	Sepal length	Sepal width	Petal length	Petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

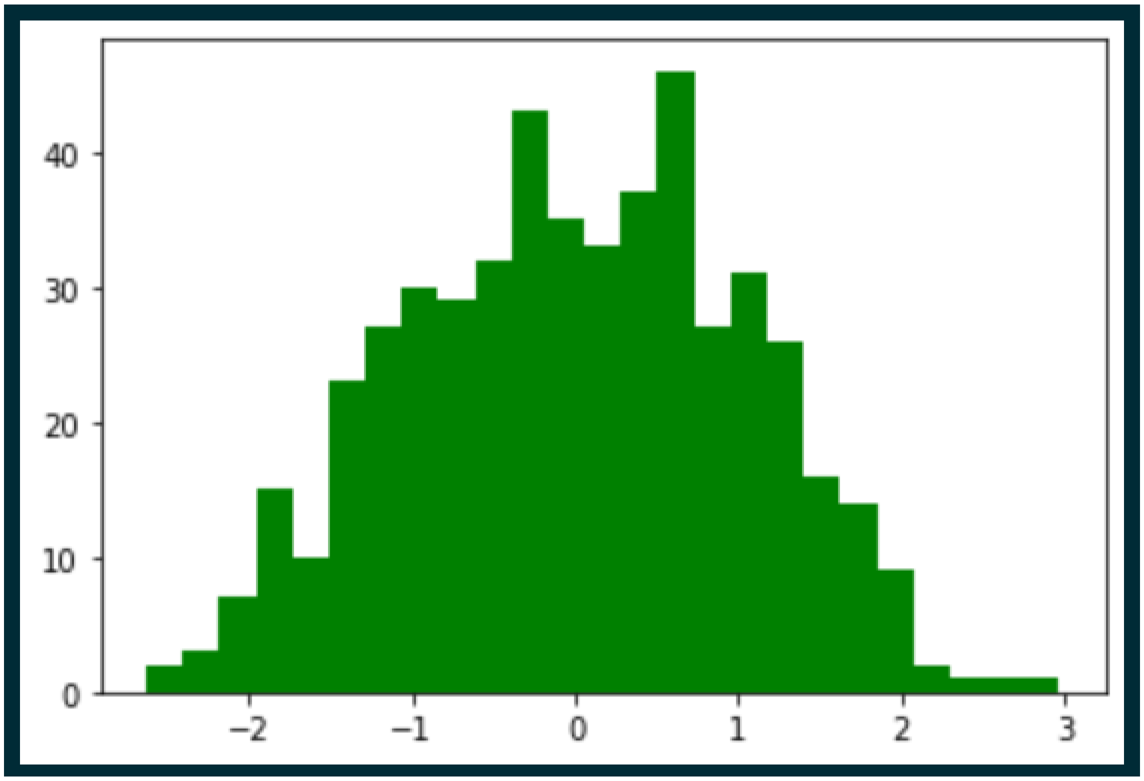
Ques 4.
We require 500 random numbers following Normal distribution between -3 and 3, which implies mean = 0.
We know that 99.7% of our values lie between [mean - 3std. , mean + 3std.], implies 3std. = 3. Hence Standard deviation(Std.) = 1.
We simulate ‘random.normal(0,1)‘ 700 times and pick the first 500 numbers which are in the range of -3 to +3.

Code:
```
random_numbers = np.random.normal(0, 1, 1000)

numbers_inrange = []

#selecting 500 numbers in range -3 to 3 from the generated 600.
for rn in random_numbers:
    if len(numbers_inrange) == 500:
        break
    if rn>=-3 and rn<=3:
        numbers_inrange.append(rn)
```

Scatter Plot:



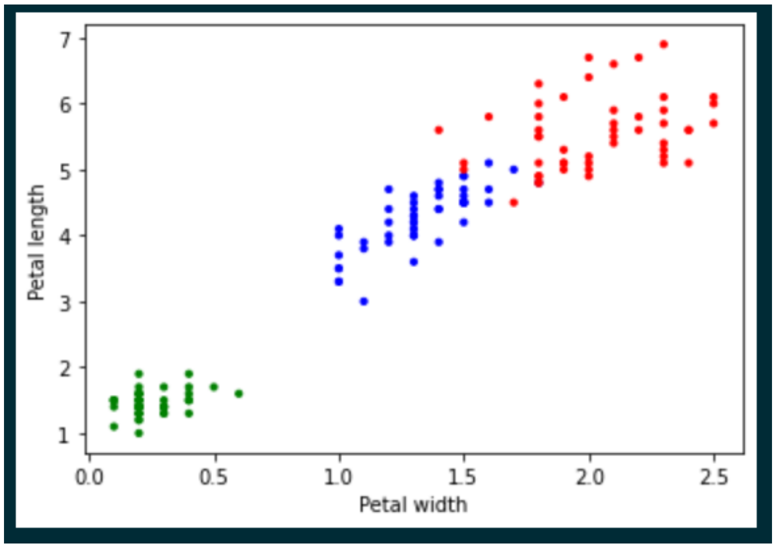
Histogram:



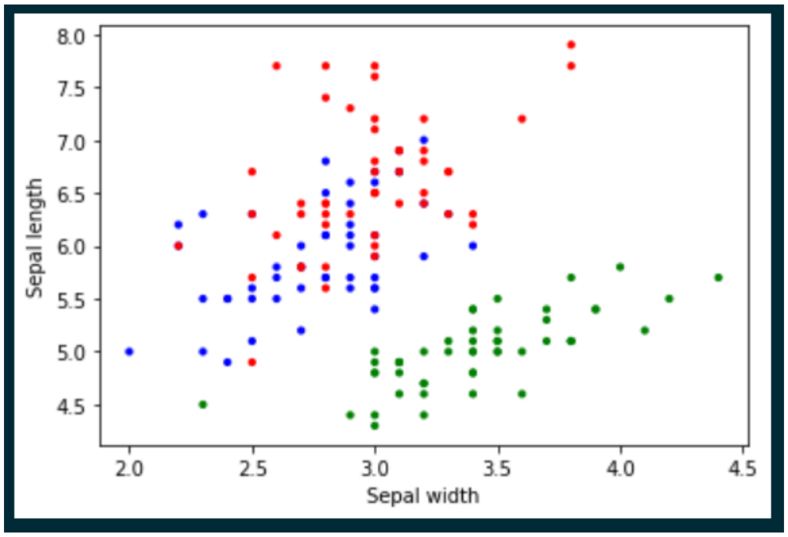
Ques 5.

Scatter Plots for all 4 RV's:

1. Petal Length vs Petal Width

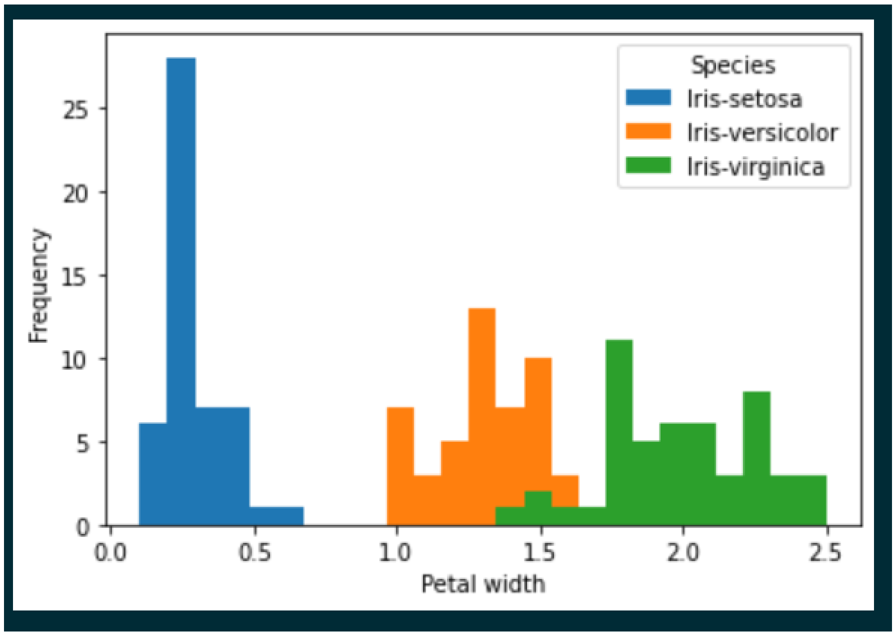


2. Sepal Length vs Sepal Width

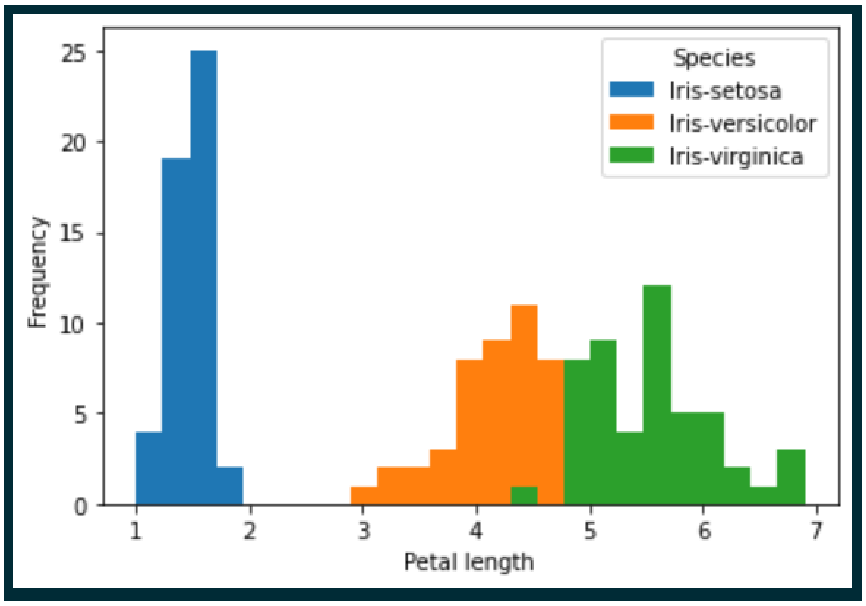


Histograms for all 4 RV's:

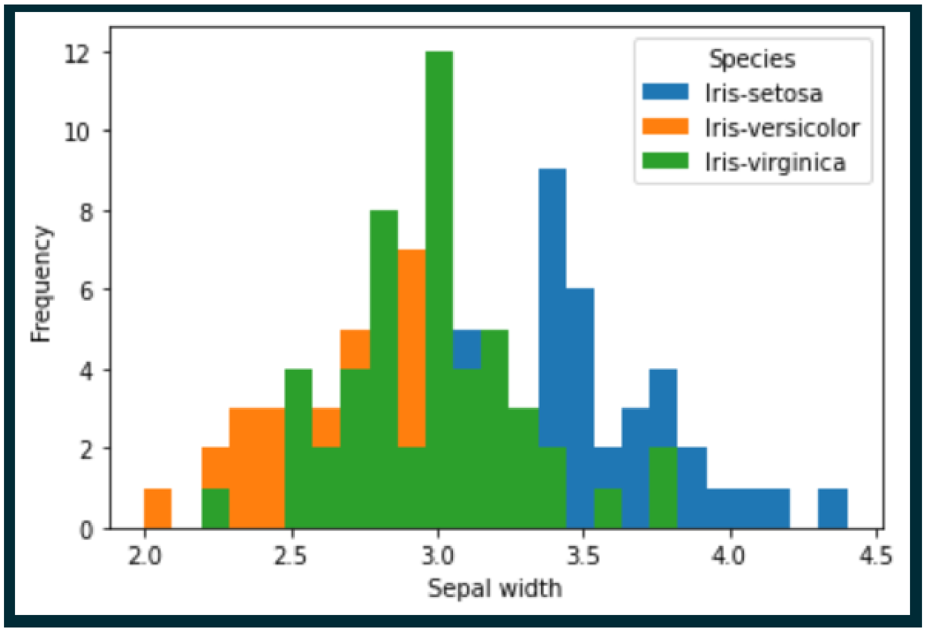
1. Petal Width



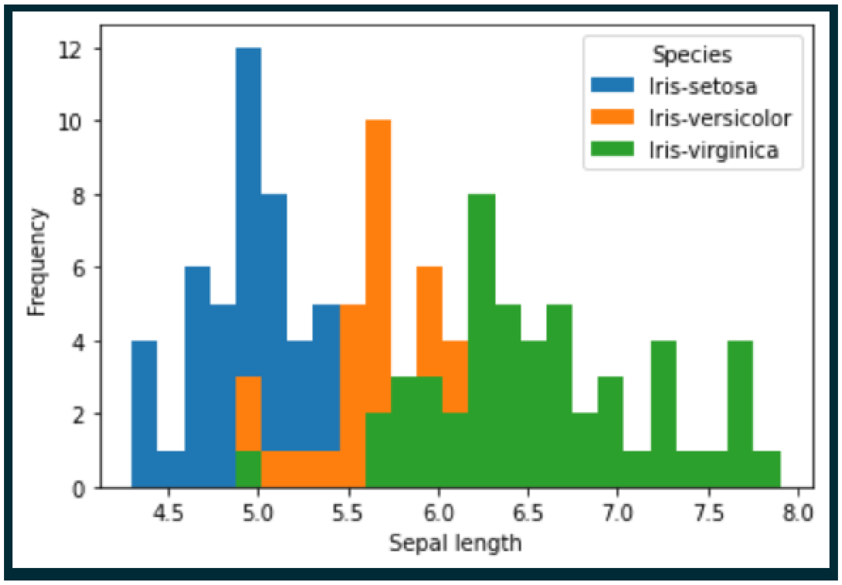
2. Petal Length



3. Sepal Width



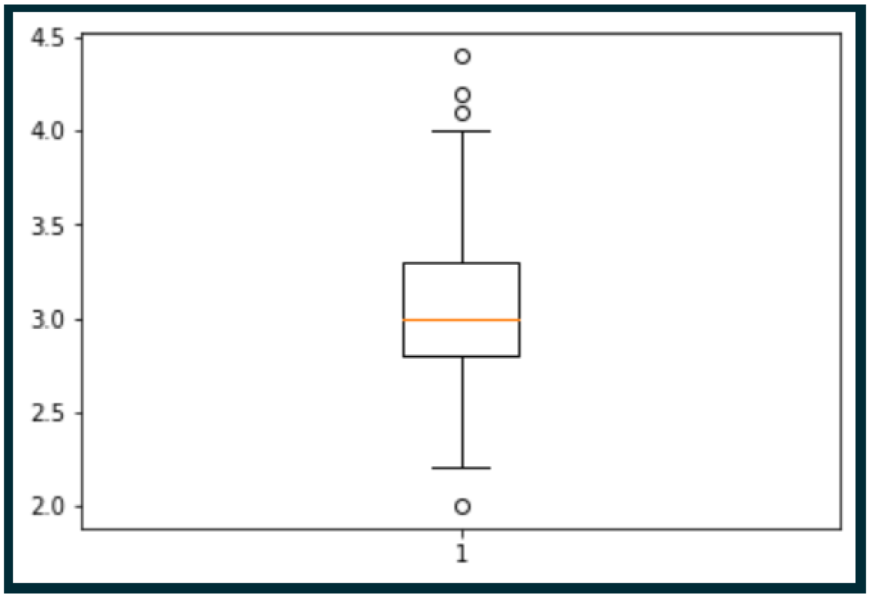
4. Sepal Length



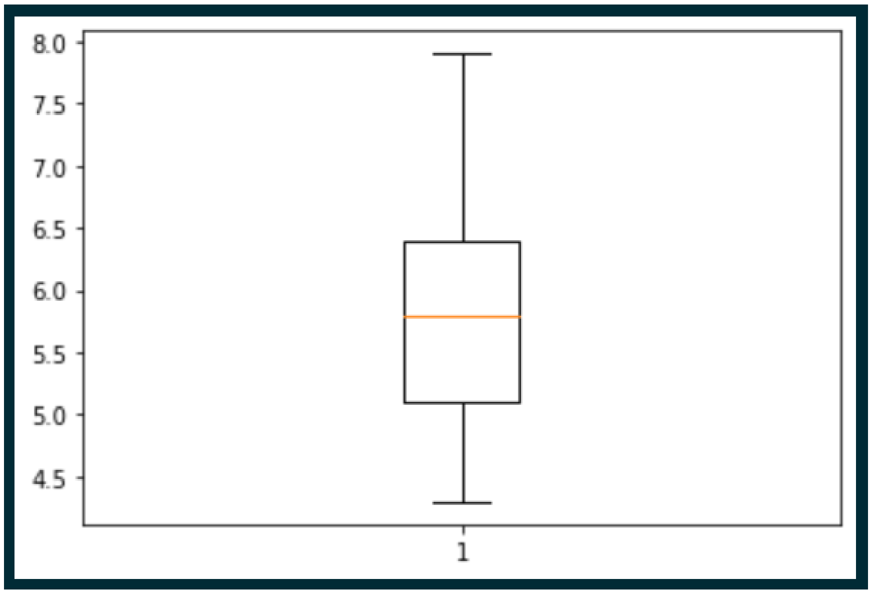
Ques 6.

Box Plots for all 4 RV’s:

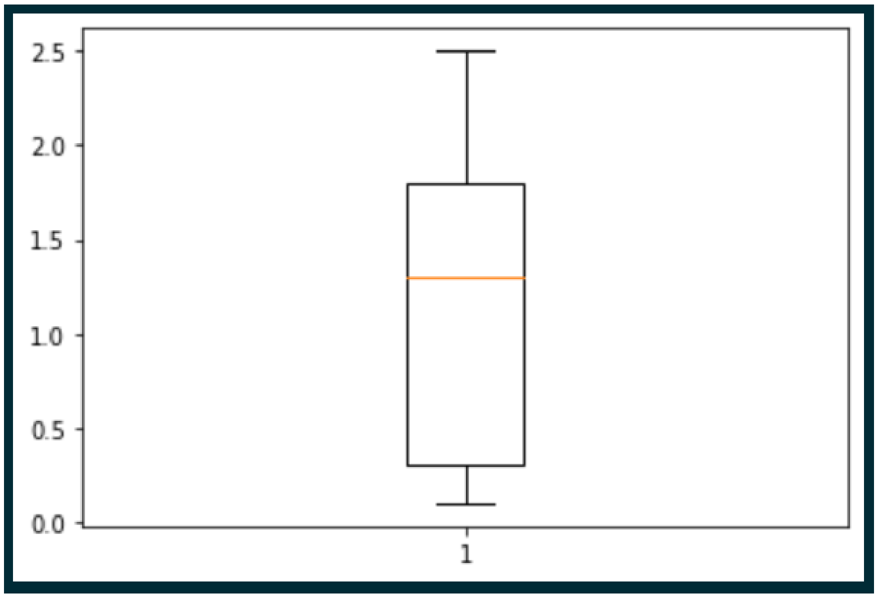
1. Sepal Width



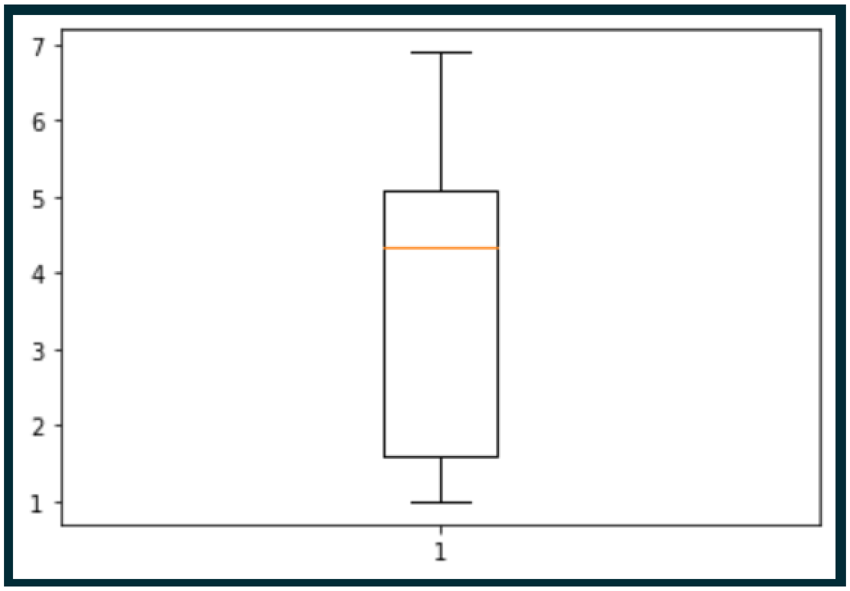
2. Sepal Length



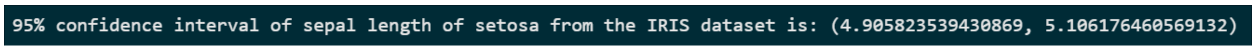
3. Petal Width



4. Petal Length



Ques 7.
Here, we use t_{n-1} distribution because we don't have population standard deviation.
For n=150, alpha = 0.05,
95% C.I. = (Mean - $t_{149,0.025}$*(s/root(n)),Mean + $t_{149,0.025}$*(s/root(n)))

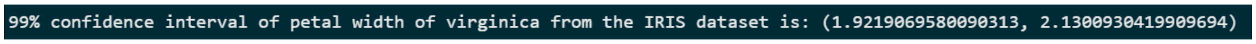
Code:
```
dataSetosa =  data[data['Species'] == 'Iris-setosa']

#calculating 95% confidence interval
ci_95 = st.t.interval(alpha=0.95, df=len(dataSetosa['Sepal length'])-1,
            loc=np.mean(dataSetosa['Sepal length']),
            scale=st.sem(dataSetosa['Sepal length']))
print("95% confidence interval of sepal length of setosa from the IRIS dataset is: ",
ci_95)
```
Output:

```
95% confidence interval of sepal length of setosa from the IRIS dataset is: (4.905823539430869, 5.106176460569132)
```

Ques 8.
Here, we use t_{n-1} distribution because we don't have population standard deviation.
For n=150, alpha = 0.01,
99% C.I. = (Mean - $t_{149,0.005}$*(s/root(n)),Mean + $t_{149,0.005}$*(s/root(n)))

Code:

```
dataVirginica =  data[data[‘Species’] ==  ‘Iris-virginica’]

#calculating 99% confidence interval
ci_99 = st.t.interval(alpha=0.99, df=len(dataVirginica[‘Petal width’])-1,
            loc=np.mean(dataVirginica[‘Petal width’]),
            scale=st.sem(dataVirginica[‘Petal width’]))
print(“99% confidence interval of petal width of virginica from the IRIS dataset is: ”,
ci_99)
```

Output:



```
99% confidence interval of petal width of virginica from the IRIS dataset is: (1.9219069580090313, 2.1300930419909694)
```

Ques 9.

Function for testing hypothesis:

```
#t-test
def check_null_hypothesis(d1, d2, alpha):
   #means
   x1 = np.mean(d1)
   x2 = np.mean(d2)
   #standard deviations
   std1 = np.std(d1)
   std2 = np.std(d2)
   #lengths
   len1 = len(d1)
   len2 = len(d2)

   degree = min(len1-1,len2-1)

   t = (x1 - x2)/(sqrt((std1*std1)/len1 + (std2*std2)/len2))

   upper_bound = st.t.ppf(0.025, df=degree)
   lower_bound = st.t.ppf(1 - 0.025, df=degree)

   if t < 0:
    return t > upper_bound
  else:
    return t < lower_bound
```

Null Hypothesis(H0): $\text{mean}_{\text{versicolor\_sepal\_width}} = \text{mean}_{\text{virginica\_sepal\_width}}$
Alternative Hypothesis(H1): $\text{mean}_{\text{versicolor\_sepal\_width}} != \text{mean}_{\text{virginica\_sepal\_width}}$

Code for Hypothesis Testing:

```
print(st.ttest_ind(a = dataVirginica['Sepal width'], b = dataVersicolor['Sepal width'],
equal_var=False))
```

Output:

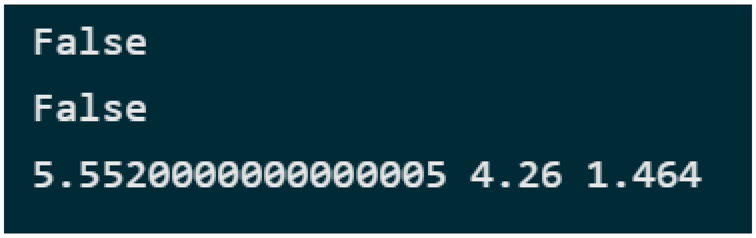
```
Ttest_indResult(statistic=3.2057607502218186, pvalue=0.001819483482104968)
Significance Level: 0.05
Is Null hypothesis true? : False
Null Hypothesis is Rejected
The means of sepal width between versicolor and virginica are different
```

Ques 10.
Null Hypothesis($H0_1$): $\text{Mean}_{\text{setosa\_petal\_length}} = \text{Mean}_{\text{virginica\_petal\_length}}$
Null Hypothesis($H0_2$): $\text{Mean}_{\text{setosa\_petal\_length}} = \text{Mean}_{\text{versicolor\_petal\_length}}$

Code for Hypothesis Testing:

```
print(check_null_hypothesis(dataVirginica['Petal length'],dataSetosa['Petal
length'],alpha))
print(check_null_hypothesis(dataVersicolor['Petal length'],dataSetosa['Petal
length'],alpha))
print(np.mean(dataVirginica['Petal length']),np.mean(dataVersicolor['Petal
length']),np.mean(dataSetosa['Petal length']))
```

Output:



```
False
False
5.5520000000000005 4.26 1.464
```

Conclusion:

The mean of setosa is less than virginica and versicolor. Also the null hypothesis can be rejected for both cases. Hence, it is true that the petal length of class setosa is shorter than the petal length of other classes (virginica and versicolor)

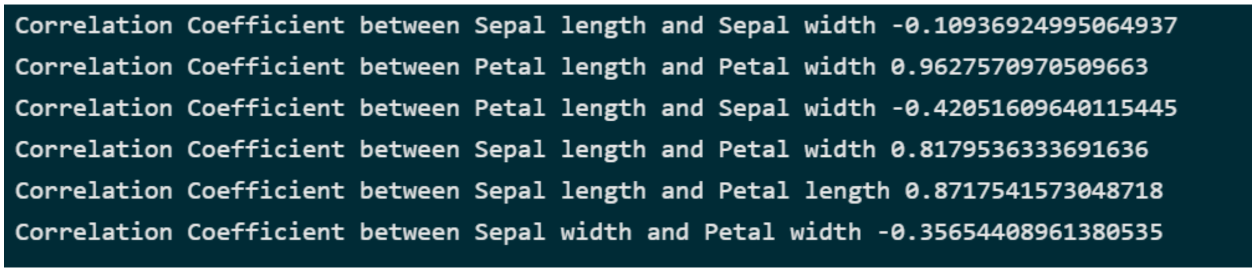
Ques 11.
Finding correlation coefficient for all pair of RVs:

Code:

```
#correlations between different RV's
print( "Correlation Coefficient between Sepal length and Sepal
width ",np.corrcoef(data[ 'Sepal length '], data[ 'Sepal width '])[0][1]) #negatively
correlated
print( "Correlation Coefficient between Petal length and Petal
width ",np.corrcoef(data[ 'Petal length '], data[ 'Petal width '])[0][1]) #strongly
correalted positively
print( "Correlation Coefficient between Petal length and Sepal
width ",np.corrcoef(data[ 'Petal length '], data[ 'Sepal width '])[0][1]) #negatively
correlated
print( "Correlation Coefficient between Sepal length and Petal
width ",np.corrcoef(data[ 'Sepal length '], data[ 'Petal width '])[0][1]) #positively
correalted (strong linear)
print( "Correlation Coefficient between Sepal length and Petal
length ",np.corrcoef(data[ 'Sepal length '], data[ 'Petal length '])[0][1]) #positively
correalted (strong linear)
```

```
print("Correlation Coefficient between Sepal width and Petal
width",np.corrcoef(data['Petal width'], data['Sepal width'])[0][1])  #negatively
correlated
```

Output:



```
Correlation Coefficient between Sepal length and Sepal width -0.10936924995064937
Correlation Coefficient between Petal length and Petal width 0.9627570970509663
Correlation Coefficient between Petal length and Sepal width -0.42051609640115445
Correlation Coefficient between Sepal length and Petal width 0.8179536333691636
Correlation Coefficient between Sepal length and Petal length 0.8717541573048718
Correlation Coefficient between Sepal width and Petal width -0.35654408961380535
```

Conclusion:

```
Sepal length and width are weakly negatively correlated.
Petal length and sepal width are negatively correlated.
Petal and sepal width are negatively correlated.
Petal length and width are strongly correlated postively.
Sepal length and petal width are positively correlated.
Sepal and petal length are positively correlated.
```

Ques 12.

Null Hypothesis(H0): correlation_coefficient$_{\text{(Sepal\_Length, Sepal\_Width)}}$ = 0
Alternative Hypothesis(H1): correlation_coefficient$_{\text{(Sepal\_Length, Sepal\_Width)}}$!= 0

Code for Hypothesis Testing:

```
print("Null Hypothesis: Correlation Coefficient between Sepal Length and Sepal Width
= 0")



#correlation coefficients
correlation = np.corrcoef(data['Sepal length'], data['Sepal width'])[0][1]
print("Correlation coefficient: ",correlation) #negatively correlated
```

```
#test of correlation
length = len(data['Sepal length'])
t = (correlation*sqrt(length))/sqrt(1-correlation*correlation)
degree = len(data['Sepal length'])-2

upper_bound = st.t.ppf(0.025, df=degree)
lower_bound = st.t.ppf(1 - 0.025, df=degree)

if t < 0:
    if t > upper_bound:
        print("Null hypothesis true")
    else:
        print("Null hypothesis can be rejected")
else:
    if t < lower_bound:
        print("Null hypothesis true")
    else:
        print("Null hypothesis can be rejected")
```

Output:

```
Null Hypothesis: Correlation Coefficient between Sepal Length and Sepal Width = 0
Correlation coefficient: -0.10936924995064937
Null hypothesis true
```

Performing Linear Regression on dataset, taking Sepal Length as independent variable and Sepal Width as dependent variable.

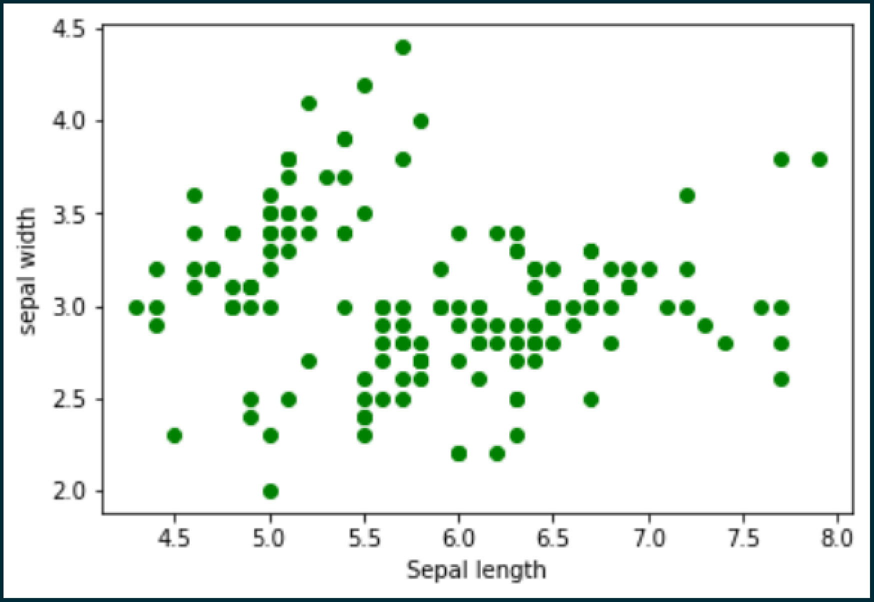
Code for Linear Regression:

```
model = LinearRegression()
X = np.array(data['Sepal length']).reshape((-1,1))
```

```
Y = np.array(data['Sepal width'])
model.fit(X, Y)
score = model.score(X, Y)
print("The score of the model of linear regression is: ", score)
```

Output:

```
The score of the model of linear regression is: 0.0119616328347677
```



Conclusion:

Hence, we can say that the model cannot be linearly expressed.

Ques 13.

Linear Regression using Petal Length as X(Independent Variable) and Petal Width as Y(Dependent Variable):

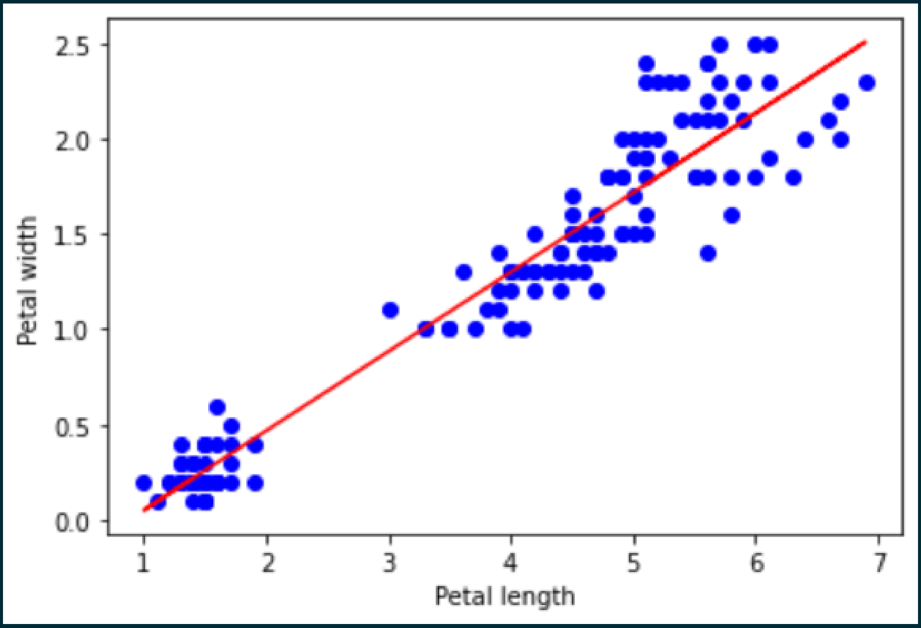
Code:

```
model = LinearRegression()
X = np.array(data['Petal length']).reshape((-1,1))
model.fit(X, np.array(data['Petal width']))
score = model.score(X, data['Petal width'])
```

```
print("The score of the model of linear regression is: ", score)

n = len(data['Petal width'])
x = data['Petal length']
y = data['Petal width']
x_mean = np.mean(x)
y_mean = np.mean(y)

Sxy = np.sum(x*y)- n*x_mean*y_mean
Sxx = np.sum(x*x)-n*x_mean*x_mean

b1 = Sxy/Sxx
b0 = y_mean-b1*x_mean
print('slope b1 is', b1)
print('intercept b0 is', b0)

y_pred = b1 * x + b0
```

Output:

```
The score of the model of linear regression is: 0.9269012279220037
slope b1 is 0.4164191322854009
intercept b0 is -0.3665140452167266

Text(0, 0.5, 'Petal width')
```



Scatter Plot with Best Fit Line:

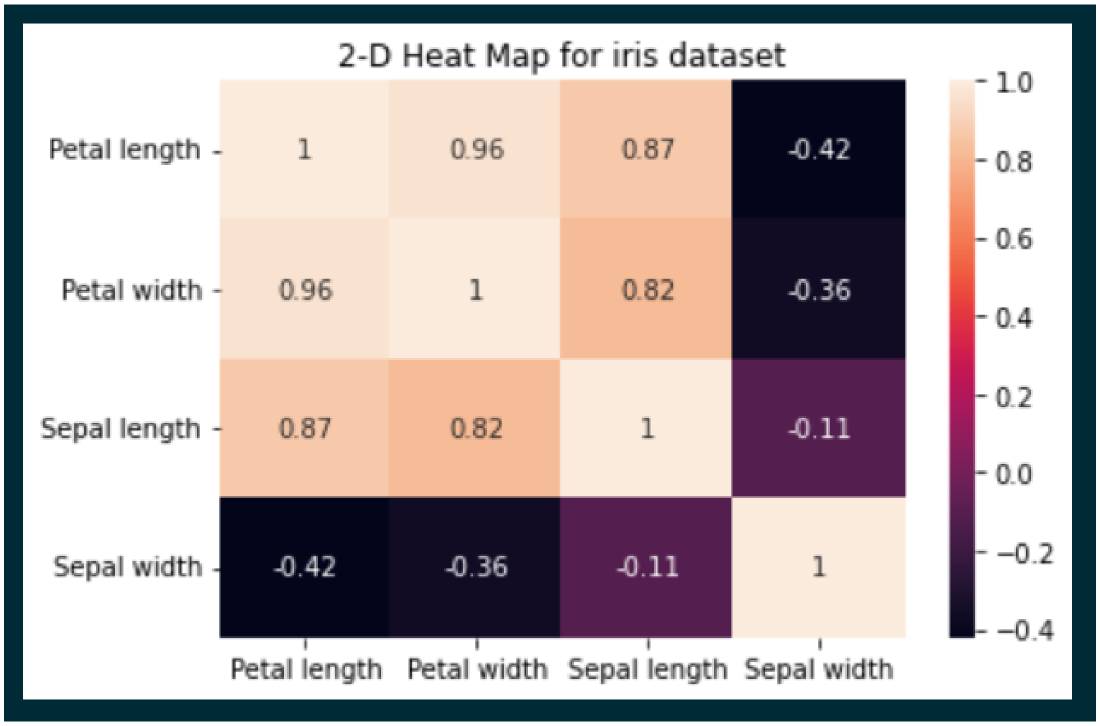
Ques 14.

Heatmap code:

```
data_hm = [[0 for _ in range(4)] for _ in range(4)]

properties = [data['Petal length'], data['Petal width'], data['Sepal length'],
data['Sepal width']]

for i in range(4):
    for j in range(4):
        data_hm[i][j] = np.corrcoef(properties[i], properties[j])[0][1]
```

```
sns.heatmap(data_hm, annot=True, xticklabels=['Petal length', 'Petal width', 'Sepal length', 'Sepal width'], yticklabels=['Petal length', 'Petal width', 'Sepal length', 'Sepal width'])
plt.title("2-D Heat Map for iris dataset")
plt.show()
```

Output:



Problem Statement 2

Monte Carlo Estimation

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. One of the basic examples of getting started with the Monte Carlo algorithm is the estimation of Pi.

Estimation of Pi

The idea is to simulate random (x, y) points in a 2-D plane with domain as a square of side 2r units centered on (0,0). Imagine a circle inside the same domain with the same radius r and inscribed into the square. We then calculate the ratio of number points that lie inside the circle and total number of generated points.

We know that the area of the square is $4r^2$ unit sq while that of the circle is πr^2. The ratio of these two areas is as follows :

$$\textit{Area of the circle/Area of the square} = \pi r^2/4r^2 = \pi/4$$

$\pi/4$ = (no. of points generated inside the circle/ total no. of points generated or no. of points generated inside the square)

We simply generate random (x, y) pairs and then check if $x^2 + y^2 <= 1$. If yes, we increment the number of points that appear inside the circle.
In randomized and simulation algorithms like Monte Carlo, the more the number of iterations, the more accurate the result is.

The Algorithm

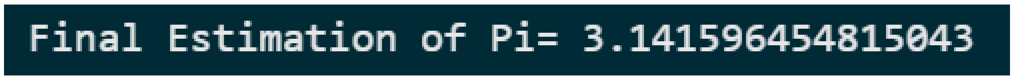
```
1. Initialize circle_points, square_points and interval to 0.
2. Generate random point x.
3. Generate random point y.
4. Calculate d = x*x + y*y.
5. If d <= 1, increment circle_points.
6. Increment square_points.
7. Increment interval.
8. If increment < NO_OF_ITERATIONS, repeat from 2.
9. Calculate pi = 4*(circle_points/square_points).
10. Terminate.
```

Code For the above algorithm

```
INTERVAL = 5000
ITERATIONS = 500000
```

```
cnt_circle = 0
cnt_square = 0

averages = []
for i in range(ITERATIONS):
    pi = 0
    for j in range(INTERVAL):

        # Randomly generated x and y values from a
        # uniform distribution
        # Range of x and y values is -1 to 1
        rand_x = random.uniform(-1, 1)
        rand_y = random.uniform(-1, 1)

        # Distance between (x, y) from the origin
        origin_dist = rand_x**2 + rand_y**2

        if origin_dist <= 1:
            cnt_circle += 1

        cnt_square += 1

        pi = 4 * cnt_circle / cnt_square
    #print("Estmated pi value after", (i+1), "th iteration is", pi)
    averages.append(pi)
print("Final Estimation of Pi=", sum(averages)/len(averages))
```

Estimated Value of pi:



```
Final Estimation of Pi= 3.141596454815043
```