```
import pandas as pd
scores=[90,80,54,77,40,49,60,99,44,77,67,87,48,93,100,59,78,98,30,84]
grade=[]
result=[]
for i in scores:
  if i>=90:
    grade.append('A')
  elif i>=80:
    grade.append('B')
  elif i>=70:
    grade.append('C')
  elif i>=50:
    grade.append('D')
  else:
    grade.append('F')
for i in scores:
  if i>=50:
    result.append('Pass')
  else:
    result.append('Fail')
df=pd.DataFrame({'Student':[f"Student {i}" for i in range(1,21)],'Scores':scores,'Grade':grade,'Result':result})
print(df)
print("Overall performance:")
print("Average:",df['Scores'].mean())
print("Highest mark:",df['Scores'].max())
print("Lowest score:",df['Scores'].min())
```

```
        Student  Scores Grade Result
    0    Student 1      90     A   Pass
    1    Student 2      80     B   Pass
    2    Student 3      54     D   Pass
    3    Student 4      77     C   Pass
    4    Student 5      40     F   Fail
    5    Student 6      49     F   Fail
    6    Student 7      60     D   Pass
    7    Student 8      99     A   Pass
    8    Student 9      44     F   Fail
    9   Student 10      77     C   Pass
    10  Student 11      67     D   Pass
    11  Student 12      87     B   Pass
    12  Student 13      48     F   Fail
    13  Student 14      93     A   Pass
    14  Student 15     100     A   Pass
    15  Student 16      59     D   Pass
    16  Student 17      78     C   Pass
    17  Student 18      98     A   Pass
    18  Student 19      30     F   Fail
    19  Student 20      84     B   Pass
    Overall performance:
    Average: 70.7
    Highest mark: 100
    Lowest score: 30
```

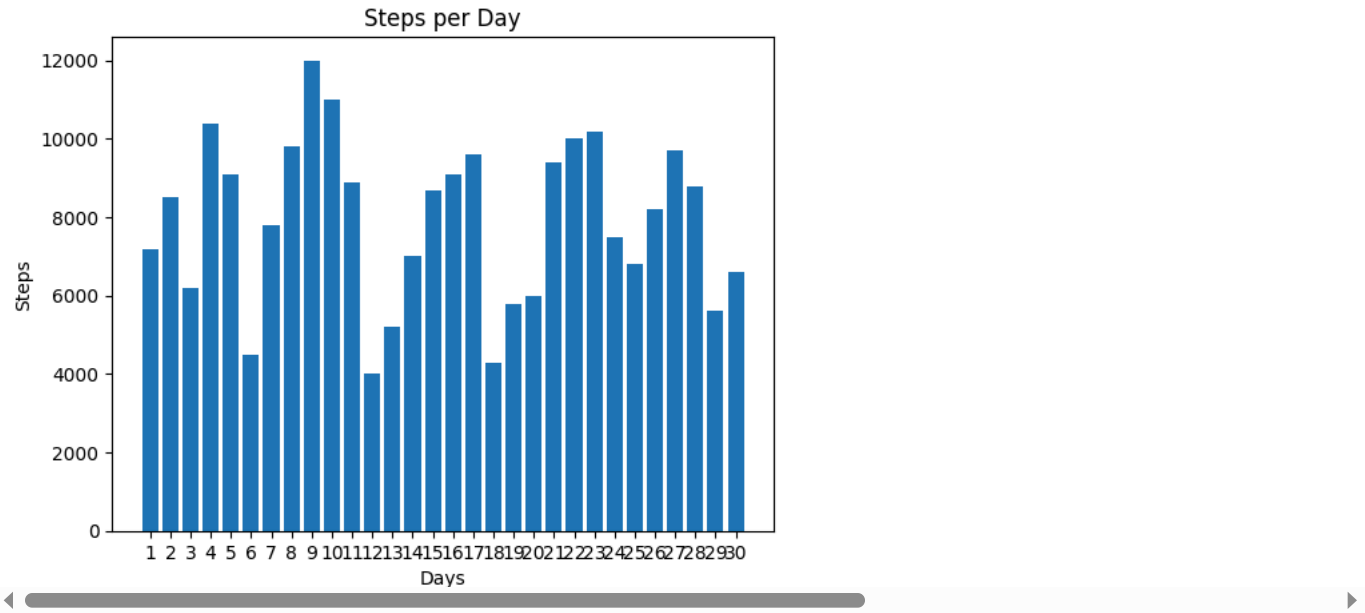
```
import numpy as np
A=np.array([[1,2],[3,4]])
b=np.array([5,6])
x=np.linalg.solve(A,b)
print(x)
```

```
[-4.   4.5]
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
steps=[ 7200, 8500, 6200, 10400, 9100, 4500, 7800,9800, 12000, 11000, 8900, 4000, 5200, 7000,8700, 9100, 9600, 4300, 5800, 6000, 9400,10000,
days=[f"{i}"for i in range(1,31)]
df=pd.DataFrame({'Days':days,'Steps':steps})
print(df)
print("Average steps per day:",df['Steps'].mean())
print("Maximum steps:",df['Steps'].max())
print("Minimum steps:",df['Steps'].min())
HA=[]
for i in range(len(steps)):
  if steps[i]>9000:
    HA.append(i+1)
print("High Activity Days:",HA)
```

```
plt.bar(days,steps)
plt.xlabel('Days')
plt.ylabel('Steps')
plt.title('Steps per Day')
plt.show()
```

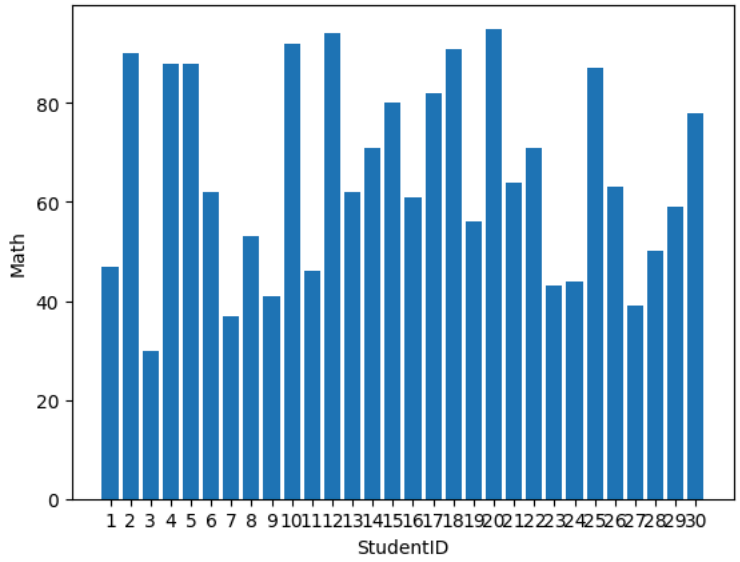
```
    Days  Steps
0     1   7200
1     2   8500
2     3   6200
3     4  10400
4     5   9100
5     6   4500
6     7   7800
7     8   9800
8     9  12000
9    10  11000
10   11   8900
11   12   4000
12   13   5200
13   14   7000
14   15   8700
15   16   9100
16   17   9600
17   18   4300
18   19   5800
19   20   6000
20   21   9400
21   22  10000
22   23  10200
23   24   7500
24   25   6800
25   26   8200
26   27   9700
27   28   8800
28   29   5600
29   30   6600
Average steps per day: 7930.0
Maximum steps: 12000
Minimum steps: 4000
High Activity Days: [4, 5, 8, 9, 10, 16, 17, 21, 22, 23, 27]
```

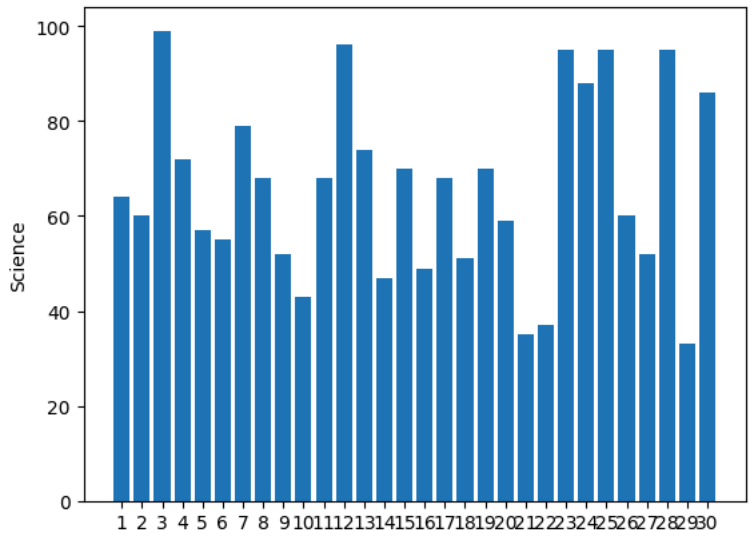


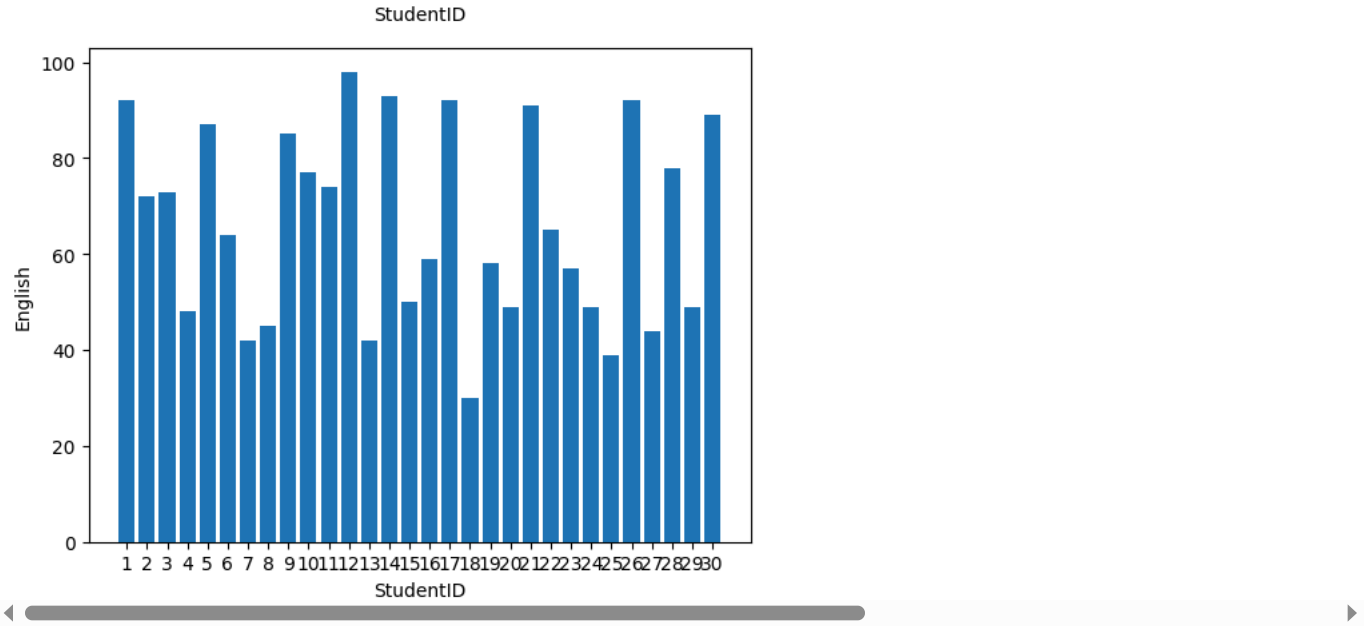

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
scores=np.random.randint(30,100,size=(30,3))
subjects=['Math','Science','English']
math=scores[:,0]
science=scores[:,1]
english=scores[:,2]
df=pd.DataFrame({'StudentID':[f"{i}" for i in range(1,31)],'Math':math,'Science':science,'English':english})
total=[]
for i in range(len(math)):
  total.append(math[i]+science[i]+english[i])
df['Total']=total
```

```
print(df)
print("Overall performance:")
print("Average:",df['Total'].mean())
print("Highest mark:",df['Total'].max())
print("Lowest score:",df['Total'].min())
max=0
TS=0
for i in range(len(total)):
  if total[i]>max:
    max=total[i]
    TS=i
print(f"Top performing student:Student {TS}")
plt.bar(df['StudentID'],df['Math'])
plt.xlabel('StudentID')
plt.ylabel('Math')
plt.show()
plt.bar(df['StudentID'],df['Science'])
plt.xlabel('StudentID')
plt.ylabel('Science')
plt.show()
plt.bar(df['StudentID'],df['English'])
plt.xlabel('StudentID')
plt.ylabel('English')
plt.show()
```

```
    StudentID  Math  Science  English  Total
0           1    47       64       92    203
1           2    90       60       72    222
2           3    30       99       73    202
3           4    88       72       48    208
4           5    88       57       87    232
5           6    62       55       64    181
6           7    37       79       42    158
7           8    53       68       45    166
8           9    41       52       85    178
9          10    92       43       77    212
10         11    46       68       74    188
11         12    94       96       98    288
12         13    62       74       42    178
13         14    71       47       93    211
14         15    80       70       50    200
15         16    61       49       59    169
16         17    82       68       92    242
17         18    91       51       30    172
18         19    56       70       58    184
19         20    95       59       49    203
20         21    64       35       91    190
21         22    71       37       65    173
22         23    43       95       57    195
23         24    44       88       49    181
24         25    87       95       39    221
25         26    63       60       92    215
26         27    39       52       44    135
27         28    50       95       78    223
28         29    59       33       49    141
29         30    78       86       89    253
Overall performance:
Average: 197.46666666666667
Highest mark: 288
Lowest score: 135
Top performing student:Student 11
```








```
import cmath
import pandas as pd

def imp(c, v, f):
  w = 2 * cmath.pi * f
  if c == 'R':
    return complex(v, 0)
  elif c == 'L':
    return complex(0, w * v)
  elif c == 'C':
    return complex(0, -1 / (w * v))
  else:
    return 'Invalid component'  # Return string indicating error

def tot(df, config):
  if config == 'Series':
    # Check if any impedance is invalid before summing
    if any(isinstance(z, str) for z in df['Impedance']):
      return "Invalid component in circuit"  # Handle invalid components
    return df['Impedance'].sum()
  else:
    # Check if any impedance is invalid before calculating
    if any(isinstance(z, str) for z in df['Impedance']):
      return "Invalid component in circuit"  # Handle invalid components
    return 1 / sum(1 / z for z in df['Impedance'])

f = float(input("Enter Frequency: "))
config = input("Enter the configuration(Series/Parallel): ")
n = int(input('No of components: '))
d = []
for i in range(n):
  c = input("Enter the component(R/L/C): ").upper()
  v = float(input("Enter the value: "))
  z = imp(c, v, f)
  d.append({'Component': f'{c}{i + 1}', 'Type': c, 'Value': v, 'Impedance': z})

df = pd.DataFrame(d)
pd.set_option("display.precision", 2)
print(df)
t = tot(df, config)

# Check if the total impedance is valid before formatting
if isinstance(t, str):
  print(t)  # Print the error message if impedance is invalid
else:
  print(f'Total Impedance: {t:3.2f}')  # Format as float if valid
```

```
Enter Frequency: 50
Enter the configuration(Series/Parallel): Series
No of components: 100
Enter the component(R/L/C): 0.1
Enter the value: 0.0001
```

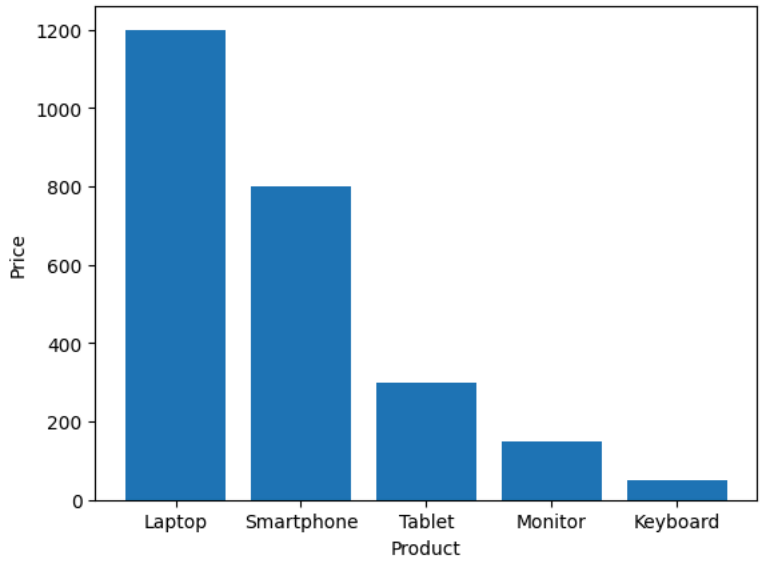
```
Enter the component(R/L/C): 100
Enter the value: 0.1
Enter the component(R/L/C): 0.001
Enter the value: 0.1
Enter the component(R/L/C): 100
Enter the value: 1000
Enter the component(R/L/C): 100
Enter the value: 0.01
Enter the component(R/L/C): 0.01
```

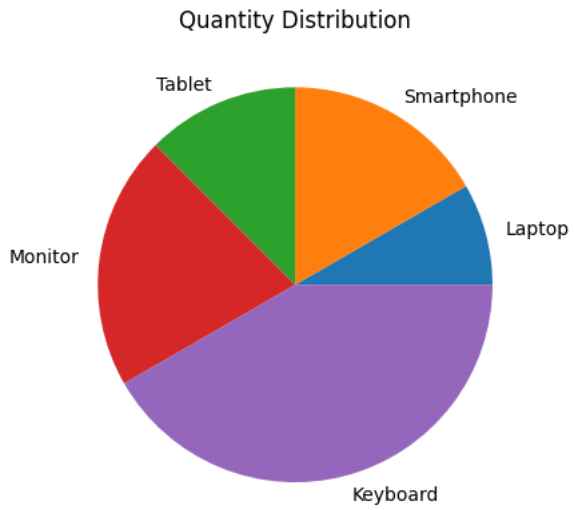
```
import numpy as np
stock_a = np.array([
    [100, 105, 95],
    [102, 108, 98],
    [101, 106, 97],
    [103, 107, 99],
    [104, 109, 100]
])
stock_b = np.array([
    [98, 104, 94],
    [99, 103, 95],
    [100, 102, 96],
    [101, 105, 97],
    [102, 106, 98]
])
sum=stock_a + stock_b
diff=stock_a - stock_b
print("Stock A:\n",stock_a)
print("\nStock B (5x3):\n",stock_b)
print("\nElement-wise Sum:\n",sum)
print("\nElement-wise Difference\n:",diff)
```

```
Stock A:
 [[100 105  95]
 [102 108  98]
 [101 106  97]
 [103 107  99]
 [104 109 100]]

Stock B (5x3):
 [[ 98 104  94]
 [ 99 103  95]
 [100 102  96]
 [101 105  97]
 [102 106  98]]

Element-wise Sum:
 [[198 209 189]
 [201 211 193]
 [201 208 193]
 [204 212 196]
 [206 215 198]]

Element-wise Difference
: [[2 1 1]
 [3 5 3]
 [1 4 1]
 [2 2 2]
 [2 3 2]]
```

```
import matplotlib.pyplot as plt
import pandas as pd # Import the pandas library

d={'Product':['Laptop','Smartphone','Tablet','Monitor','Keyboard'],'Price':[1200,800,300,150,50],'Quantity':[10, 20, 15, 25, 50]}
df=pd.DataFrame(d)
df['Total']=df['Price']*df['Quantity']
print("Average:",df['Total'].mean())
print("Highest Price:",df['Price'].max())
print("Lowest Price:",df['Price'].min())
plt.bar(df['Product'],df['Price'])
plt.xlabel('Product')
plt.ylabel('Price')
plt.show()
plt.pie(df['Quantity'],labels=df['Product'])
plt.title('Quantity Distribution')
plt.show()
```

```
Average: 7750.0
Highest Price: 1200
Lowest Price: 50
```






```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
sampling_rate = 50 # Hz
duration = 1 # in seconds
t = np.linspace(0, duration, int(sampling_rate * duration), endpoint=False)
def generate_wave(freq, amp, noise_level):
    signal = amp * np.sin(2 * np.pi * freq * t)
    noise = np.random.normal(0, noise_level, t.shape)
    return signal + noise
data={
    'Patient_ID':[f"{i}" for i in range(1,51)],
    'Time(s)':t,
    'Delta': generate_wave(2, 10, 10),
    'Theta': generate_wave(6, 6, 8),
    'Alpha': generate_wave(10, 4, 5),
    'Beta': generate_wave(20, 5, 3)
}
df = pd.DataFrame(data)
df.to_csv(r"C:\Users\DELL\Downloads\EEG-Sheet1.csv",index=False)
print(df)
print("\nStatistics (Mean, Variance, Max):")
print('Mean:\n',df[['Delta', 'Theta', 'Alpha', 'Beta']].mean())
print('Variance:\n',df[['Delta', 'Theta', 'Alpha', 'Beta']].var())
print('Max:\n',df[['Delta', 'Theta', 'Alpha', 'Beta']].max())
plt.bar(df['Patient_ID'],df['Delta'])
plt.xlabel('Patient_ID')
plt.ylabel('Delta')
```

```
plt.show()
plt.bar(df['Patient_ID'],df['Theta'])
plt.xlabel('Patient_ID')
plt.ylabel('Theta')
plt.show()
plt.bar(df['Patient_ID'],df['Alpha'])
plt.xlabel('Patient_ID')
plt.ylabel('Alpha')
plt.show()
plt.bar(df['Patient_ID'],df['Beta'])
plt.xlabel('Patient_ID')
plt.ylabel('Beta')
plt.show()
```

	Patient_ID	Time(s)	Delta	Theta	Alpha	Beta
0	1	0.00	1.982885	-2.330061	-7.191853	6.749905
1	2	0.02	4.292108	18.398283	8.490426	5.400751
2	3	0.04	3.088054	5.635519	-6.578515	-7.093901
3	4	0.06	-4.035047	6.887735	-6.416010	8.942764
4	5	0.08	1.430446	-4.141499	2.260762	0.694739
5	6	0.10	4.134830	-5.512797	2.011392	-1.952106
6	7	0.12	-0.546095	-4.324647	-3.308304	2.749439
7	8	0.14	-1.823390	1.653115	4.638489	-6.320114
8	9	0.16	2.767560	-0.847439	-9.344343	6.877714
9	10	0.18	21.056562	4.689771	-13.598626	-2.553744
10	11	0.20	-1.263658	9.916712	-1.244632	0.976565
11	12	0.22	17.373583	3.080091	-3.165511	2.622600
12	13	0.24	9.727563	-3.066649	-5.321551	-8.412900
13	14	0.26	2.653166	-3.996034	-6.550002	5.477689
14	15	0.28	-6.981932	-12.139756	0.290024	-1.432081
15	16	0.30	-5.553517	-8.908528	0.685757	0.592581
16	17	0.32	-7.079801	2.699077	8.365783	3.670773
17	18	0.34	-4.971568	-7.065856	-6.094424	-1.458730
18	19	0.36	-6.900049	11.973502	-4.542443	0.526201
19	20	0.38	-8.006375	6.449183	0.208377	-0.602757
20	21	0.40	5.329657	13.285820	-2.145203	4.775902
21	22	0.42	1.257286	15.640578	3.336306	-0.090595
22	23	0.44	6.510225	-3.495581	0.578162	-7.162187
23	24	0.46	8.596336	-25.153836	0.921728	3.600465
24	25	0.48	22.806115	-0.686029	-2.166193	-1.832426
25	26	0.50	-0.471009	-1.834312	0.130688	-1.178116
26	27	0.52	-11.423151	10.795051	5.581010	1.147064
27	28	0.54	-8.613556	-1.766639	9.722017	-3.141068
28	29	0.56	7.767478	5.110495	6.183991	10.543014
29	30	0.58	8.325818	5.153489	-7.593088	-5.610161
30	31	0.60	16.565374	-3.933056	-2.259638	3.820255
31	32	0.62	10.187269	-10.810092	4.337075	1.336271
32	33	0.64	21.293355	-13.919638	0.460054	-4.250619
33	34	0.66	4.754858	0.149991	12.246480	7.413288
34	35	0.68	25.118350	1.803639	-7.327190	1.106220
35	36	0.70	4.064663	2.195636	-0.916432	0.494479
36	37	0.72	16.352541	-3.912676	-5.034202	5.485635
37	38	0.74	12.545088	5.315838	1.143121	-6.131289
38	39	0.76	0.198710	-10.334200	-7.323793	-1.298052
39	40	0.78	-13.146992	-4.691628	-11.287639	-1.059758
40	41	0.80	-18.126692	-11.238708	-8.784582	-2.931810
41	42	0.82	2.169208	-0.392628	5.195581	2.173993
42	43	0.84	-18.184000	2.197133	-2.129031	-5.740176
43	44	0.86	-2.405726	8.323728	5.904216	4.840430
44	45	0.88	-25.404210	7.290374	-7.903133	-3.038777
45	46	0.90	-5.886987	10.397359	-0.370715	0.408768
46	47	0.92	2.476906	-2.474080	-0.909536	1.442742
47	48	0.94	3.192535	-0.265971	17.449840	-4.065494
48	49	0.96	-0.137320	-16.283683	-4.038653	6.599375
49	50	0.98	2.918386	-19.570091	-8.278454	-2.812214

```
Statistics (Mean, Variance, Max):
Mean:
 Delta    1.999517
Theta    -0.481080
Alpha    -1.033648
Beta      0.406011
dtype: float64
Variance:
 Delta    111.596092
Theta     78.985938
Alpha     39.911779
```