1. **Write a script that inputs a number of seconds from the user. Calculate the number of hours, minutes, and remaining seconds. Print them separated by " - ". For example, if the user types 3750 seconds as input, the script should print: 1 - 2 - 30 Assume that the user enters a number of seconds that is higher than 3600. Use both the floor division and the remainder operation to calculate the number of hours, minutes, and seconds.**

In [5]:

```
import time

numSeconds = int(input("Enter the number of seconds : "))
ans = time.strftime("%H - %M - %S", time.gmtime(numSeconds))
print(ans)
```

```
Enter the number of seconds : 3750
01 - 02 - 30
```

1. **Starting with 200 bacteria, the growth in the number of bacteria after n hours is calculated as follows: B = 200 × 2n. Print the number of bacteria after 0, 5, 10, and 15 hours in table format as shown below. Use the tab escape sequence to achieve the two-column output.**

In [9]:

```
print("Time \t Bacteria")
for i in range(0, 20, 5):
    print(i, " \t ", (200 + (200*2*i)))
```

```
Time    Bacteria
0      200
5      2200
10      4200
15      6200
```

1. **During the flu season, it is important to keep track of the number of infected patients. Write a script in which the user can input the number of reported infections per day over one week. Calculate the total, average, smallest, and largest of these values. Write your script using a loop structure.**

In [18]:

```
import numpy as np

numWeeks = int(input("Enter the number of weeks : "))
total_records = np.empty(shape=(numWeeks, 4))
for i in range(numWeeks):
    infection_count = np.empty(7)
    for j in range(1, 8):
        temp = int(input(f"Enter the infection for day {j} : "))
        infection_count[j-1] = temp
    record = np.array([infection_count.sum(), infection_count.mean(), infection_count.min
(), infection_count.max()])
    total_records[i] = record
print(np.matrix(total_records))
```

```
Enter the number of weeks : 1
Enter the infection for day 1 : 1
Enter the infection for day 2 : 2
Enter the infection for day 3 : 3
Enter the infection for day 4 : 4
Enter the infection for day 5 : 5
Enter the infection for day 6 : 6
Enter the infection for day 7 : 7
Total    Average    Smallest    Largest
[[28.   4.   1.   7.]]
```

1. **Calculate the mean, median, and mode of the temperatures measured in Sidney on the first 9 days of**

1. **Calculate the mean, median, and mode of the temperatures measured in Sidney on the first 9 days of February (in °C): 19.5, 19.5, 21.6, 20.2, 19.7, 20.2, 18.6, 17.2 and 19.5.**

In [20]:

```
lis = [float(item) for item in input("Enter the list of temperatures measured").split()]
temp_arr = np.array(lis)
print("Mean : ", np.mean(temp_arr))
print("Median : ", np.median(temp_arr))
vals, count = np.unique(temp_arr, return_counts=True)
index = np.argmax(count)
print("Mode : ", vals[index])
```

```
Enter the list of temperatures measured19.5 19.5 21.6 20.2 19.7 20.2 18.6 17.2 19.5
Mean :  19.555555555555557
Median :  19.5
Mode :  19.5
```

1. **Rolling a Six-Sided Die 6,000,000 Times If randrange truly produces integers at random, every number in its range has an equal probability (or chance or likelihood) of being returned each time we call it. To show that the die faces 1–6 occur with equal likelihood, the following script simulates 6,000,000 die rolls. When you run the script, each die face should occur approximately 1,000,000 times, as in the sample output.**

In [24]:

```
import random

rand_count = {}

for i in range(1, 7):
    rand_count[i] = 0
for i in range(6000000):
    rand_count[random.randrange(1, 7)] += 1
print(rand_count)
```

```
{1: 1000306, 2: 999257, 3: 998832, 4: 1000631, 5: 1000575, 6: 1000399}
```

1. **During the first 20 days of the COVID-19 pandemic, the number of newly infected patients per day in a country were recorded. Place the following numbers in a list: 174, 335, 278, 214, 422, 513, 737, 672, 489, 412, 1301, 1105, 1123, 1376, 1502, 894, 665, 1704, 1656, 1342 Use the built-in functions in the statistics module to display the following statistics concerning the infection rate: minimum, maximum, range, mean, median, variance, and standard deviation.**

In [27]:

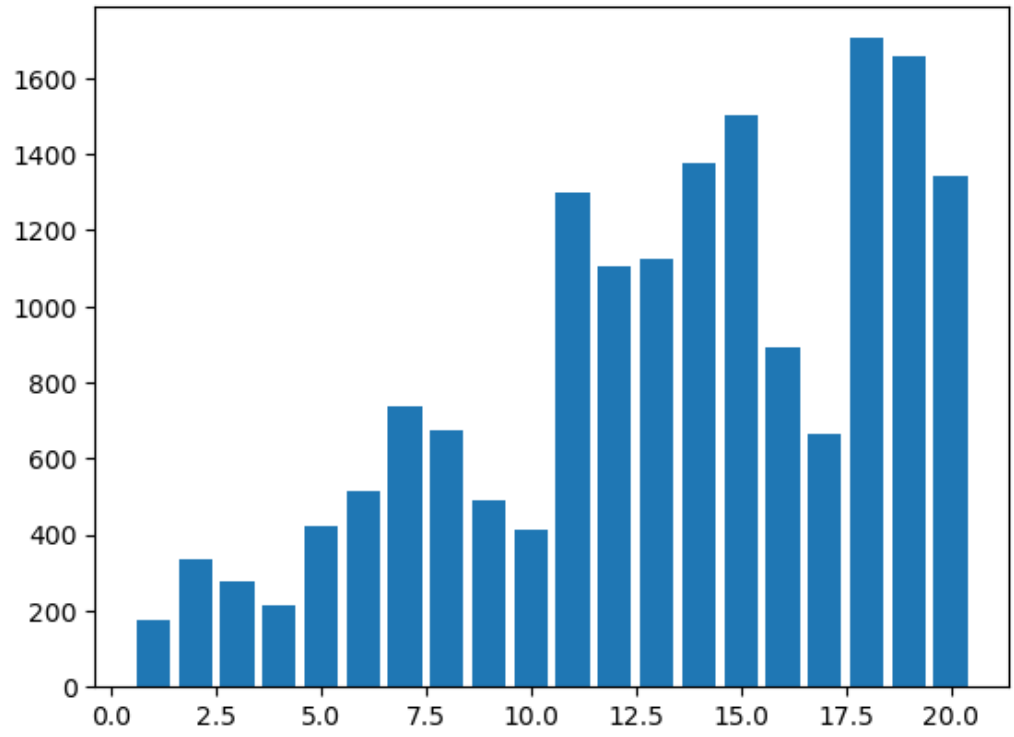
```
infected_count = [174, 335, 278, 214, 422, 513, 737, 672, 489, 412, 1301, 1105, 1123, 137
6, 1502, 894, 665, 1704, 1656, 1342]
print("Minimum : ", np.min(infected_count))
print("Maximum : ", np.max(infected_count))
print("Range : ", np.max(infected_count) - np.min(infected_count))
print("Mean : ", np.mean(infected_count))
print("Median : ", np.median(infected_count))
print("Variance : ", np.var(infected_count))
print("Standard Deviation : ", np.std(infected_count))
```

```
Minimum :  174
Maximum :  1704
Range :  1530
Mean :  845.7
Median :  704.5
Variance :  240939.70999999996
Standard Deviation :  490.85609907589003
```

1. **Using the created list above display a bar chart showing the number of newly infected patients for each day.**

In [28]:

```
import matplotlib.pyplot as plt

X = np.array(list(range(1, len(infected_count)+1)))
Y = np.array(infected_count)

plt.bar(X, Y)
plt.show()
```



1. **Calculate the sum of the two values. Each die has a value from 1 to 6, so the sum of the values will vary from 2 to 12, with 7 being the most frequent sum, and 2 and 12 the least frequent.**

In [33]:

```
from collections import defaultdict
val = {}
val = defaultdict(lambda : 0, val)
for i in range(6000000):
    val[random.randrange(1, 7) + random.randrange(1, 7)] += 1
print(dict(val))
```

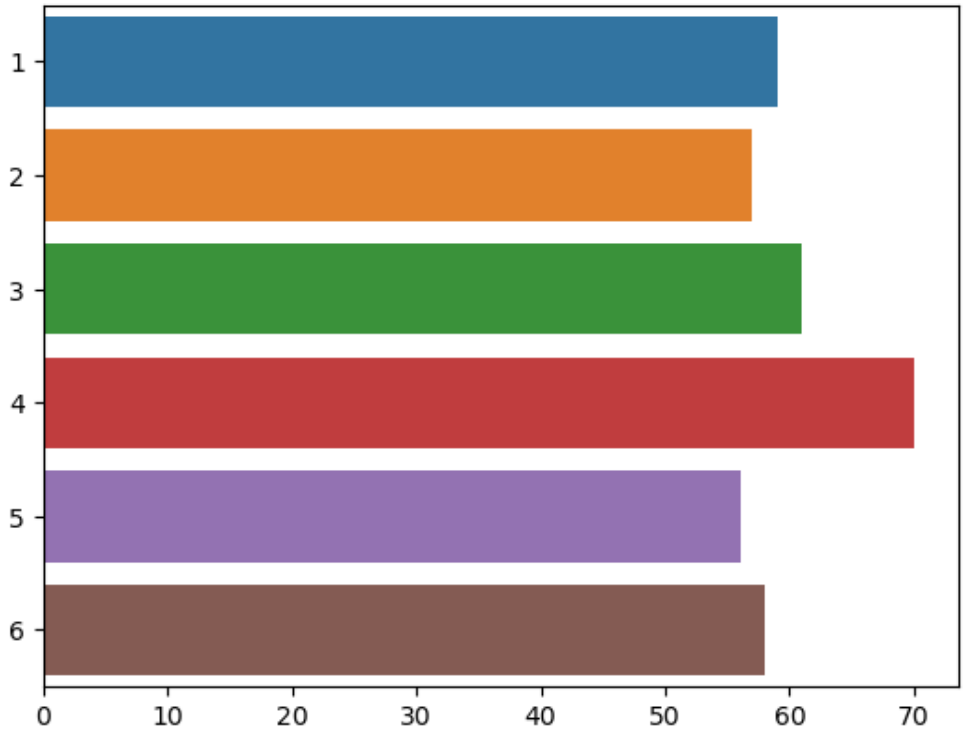
```
{7: 1000355, 8: 833226, 6: 834654, 10: 500875, 9: 666792, 4: 498984, 3: 333815, 5: 665733,
2: 166363, 11: 332919, 12: 166284}
```

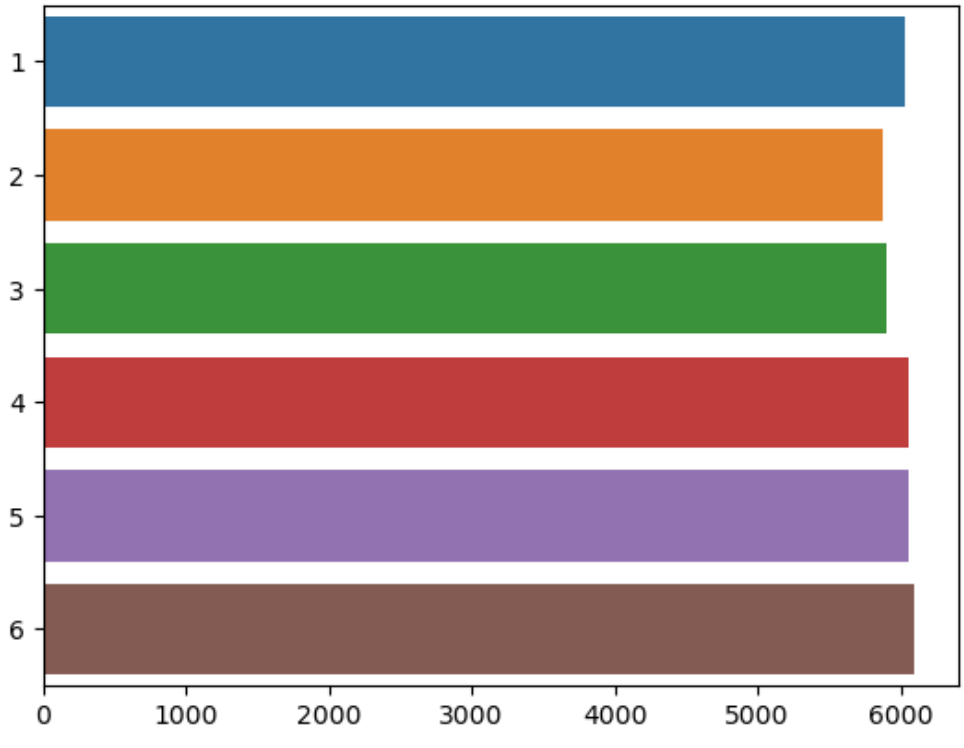
1. **Use a command-line argument to obtain the number of rolls. Display a bar plot summarising the roll frequencies. The following screen captures show the final bar plots for sample executions of 360, 36,000 and 36,000,000 rolls. Use the Seaborn barplot function's optional orient keyword argument to specify a horizontal bar plot.**

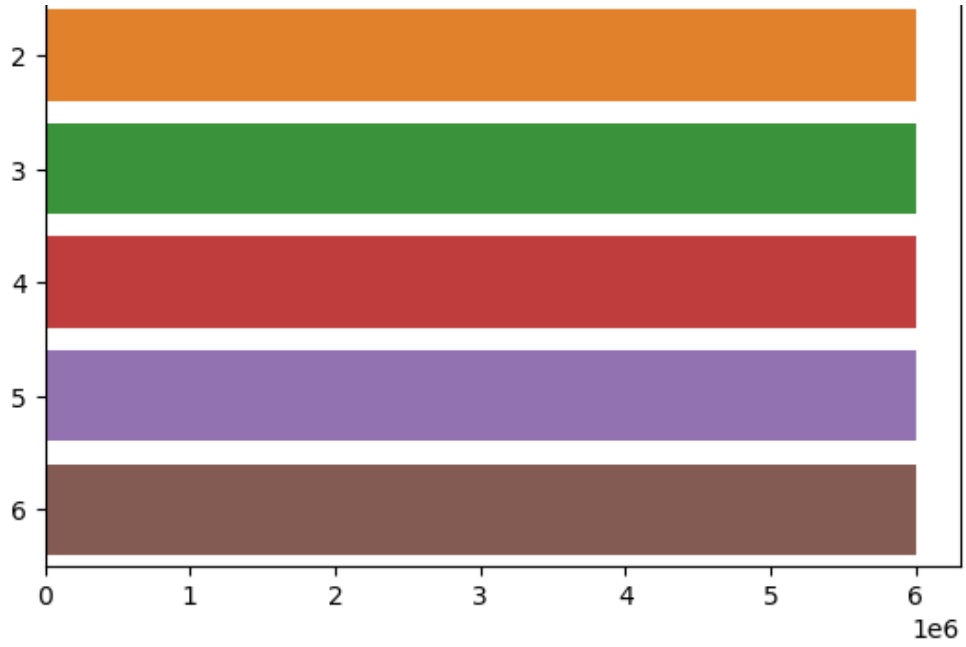
In [44]:

```
import seaborn as sns
rolls = {}
rolls = defaultdict(lambda : 0, rolls)
for i in range(361):
    rolls[random.randrange(1, 7)] += 1

roll = list(dict(rolls).keys())
count = [rolls[i] for i in roll]
sns.barplot(y = roll, x = count, orient = "h")
plt.show()

rolls.clear()
```

```
for i in range(36001):
    rolls[random.randrange(1, 7)] += 1

roll = list(dict(rolls).keys())
count = [rolls[i] for i in roll]
sns.barplot(y = roll, x = count, orient = "h")
plt.show()

rolls.clear()
for i in range(36000001):
    rolls[random.randrange(1, 7)] += 1

roll = list(dict(rolls).keys())
count = [rolls[i] for i in roll]
sns.barplot(y = roll, x = count, orient = "h")
plt.show()
```







1. **(Element-Wise array Multiplication) Create a 10-by-10 array containing the number 4 in each position. Create a second 10-by-10 array containing the numbers 1 to 100 in ascending order. Multiply the first array by the second array.**

In [3]:

```
import numpy as np
arr1 = np.full((10, 10), 4)
arr2 = np.arange(1,101).reshape(10,10)
```

In [7]:

```
mult_res = arr1 * arr2
mult_res
```

Out[7]:

```
array([[  4,   8,  12,  16,  20,  24,  28,  32,  36,  40],
       [ 44,  48,  52,  56,  60,  64,  68,  72,  76,  80],
       [ 84,  88,  92,  96, 100, 104, 108, 112, 116, 120],
       [124, 128, 132, 136, 140, 144, 148, 152, 156, 160],
       [164, 168, 172, 176, 180, 184, 188, 192, 196, 200],
       [204, 208, 212, 216, 220, 224, 228, 232, 236, 240],
       [244, 248, 252, 256, 260, 264, 268, 272, 276, 280],
       [284, 288, 292, 296, 300, 304, 308, 312, 316, 320],
       [324, 328, 332, 336, 340, 344, 348, 352, 356, 360],
       [364, 368, 372, 376, 380, 384, 388, 392, 396, 400]])
```

1. **(array from List of Lists) Create an array from a list of two lists. The first list con[1]sists of the even numbers counting down from 10 to 0 and the second counting up from 0 to 10.**

In [10]:

```
arr = np.array([list(range(10, -1, -2)), list(range(0, 11, 2))])
print(arr)
```

```
[[10  8  6  4  2  0]
 [ 0  2  4  6  8 10]]
```

1. **(Flattening arrays with flatten vs. ravel) Using the array created in Exercise 7.4, first, flatten the array with the method flatten. Change the second element of the new array to 10. Compare the new and the original array. Second, flatten the array using the method ravel and perform the same comparison.**

In [12]:

```
# Using flatten
a = arr.flatten()
print(a)
a[1] = 10
print("a is \n", a)
print("arr is \n", arr)
```

```
[10  8  6  4  2  0  0  2  4  6  8 10]
a is
 [10 10  6  4  2  0  0  2  4  6  8 10]
arr is
 [[10  8  6  4  2  0]
 [ 0  2  4  6  8 10]]
```

In [13]:

```
# Using ravel
a = np.ravel(arr)
print(a)
a[1] = 10
print("a is \n", a)
print("arr is \n", arr)
```

```
[10  8  6  4  2  0  0  2  4  6  8 10]
a is
 [10 10  6  4  2  0  0  2  4  6  8 10]
arr is
 [[10 10  6  4  2  0]
 [ 0  2  4  6  8 10]]
```

1. **Loading the Titanic Dataset via a URL**

In [23]:

```
import pandas as pd

titanic_df = pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")
```

1. **Viewing Some of the Rows in the Titanic Dataset**

In [ ]:

```

```

In [24]:

```
titanic_df.head()
```

Out[24]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

1. **Simple Data Analysis with the Titanic Disaster Dataset use describe**

In [25]:

```
titanic_df.describe()
```

Out[25]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

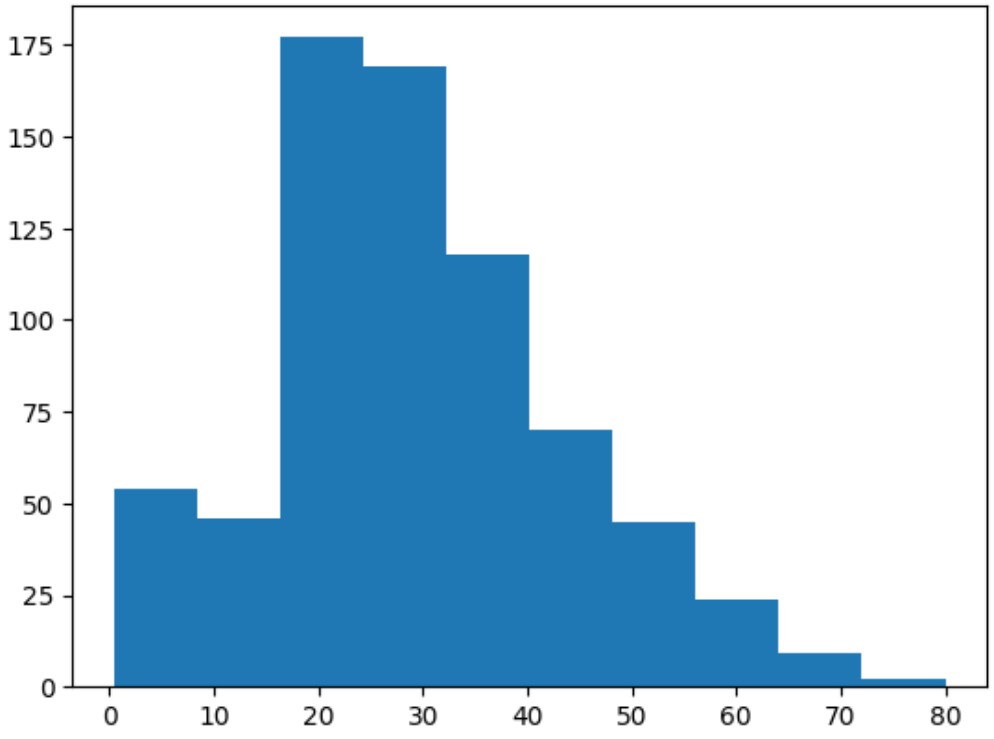
1. **Draw Passenger Age Histogram**

In [28]:

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.hist(titanic_df['Age'])
```

Out[28]:

```
(array([ 54.,  46., 177., 169., 118.,  70.,  45.,  24.,   9.,   2.]),
 array([ 0.42 ,  8.378, 16.336, 24.294, 32.252, 40.21 , 48.168, 56.126,
        64.084, 72.042, 80.   ]),
 <BarContainer object of 10 artists>)
```



1. **(Working with the Iris.csv Dataset in Pandas) Another popular dataset for ma[1]chine-learning novices is the Iris dataset, which contains 150 records of information about three Iris plant species. Like this diamonds dataset, the Iris dataset is available from various online sources, including Kaggle. Investigate the Iris dataset's columns, then perform the following tasks to study and analyze the dataset:**

a) Download Iris.csv from one of the dataset repositories

b) Load the dataset into a pandas DataFrame with the following statement, which uses the first column of each record as the row index: df = pd.read_csv('Iris.csv', index_col=0)

c) Display the DataFrame's head.

d) Display the DataFrame's tail.

e) Use the DataFrame method describe to calculate the descriptive statistics for the numerical data columns—SepalLengthCm, SepalWidthCm, PetalLengthCm and PetalWidthCm.

f) Pandas has many built-in graphing capabilities. Execute the %matplotlib magiic to enable Matplotlib support in IPython. Then, to view histograms of each numerical data column, call your DataFrame's hist method.

In [31]:

```
iris_df = pd.read_csv('iris.csv')
```

In [32]:

```
iris_df
```

Out[32]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

150 rows × 5 columns

In [33]:

```
iris_df.head()
```

Out[33]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

In [34]:

```
iris_df.tail()
```

Out[34]:

	sepal.length	sepal.width	petal.length	petal.width	variety

	sepal.length	sepal.width	petal.length	petal.width	variety
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

In [35]:

```
iris_df.describe()
```

Out[35]:

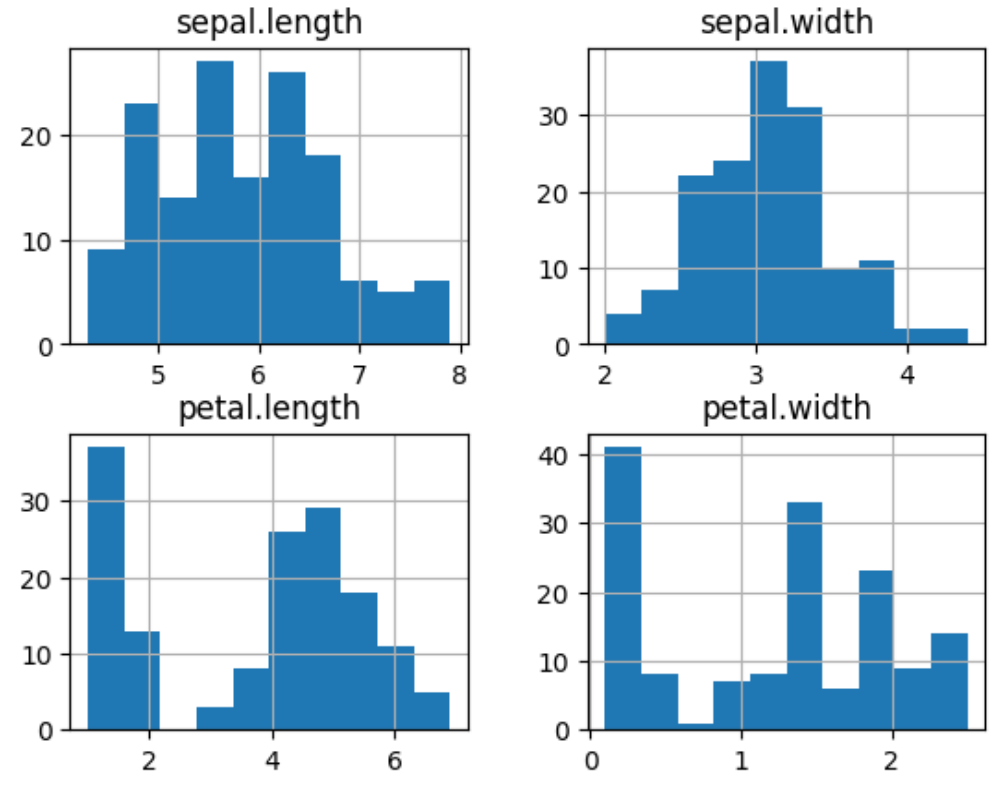
	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [36]:

```
iris_df.hist()
```

Out[36]:

```
array([[<AxesSubplot: title={'center': 'sepal.length'}>,
        <AxesSubplot: title={'center': 'sepal.width'}>],
       [<AxesSubplot: title={'center': 'petal.length'}>,
        <AxesSubplot: title={'center': 'petal.width'}>]], dtype=object)
```



1. **Perform the following tasks to study and analyze the diamonds dataset:**

a) Download diamonds.csv from one of the dataset repositories.

b) Load the dataset into a pandas DataFrame with the following statement, which uses the first column of each record as the row index: df = pd.read_csv('diamonds.csv', index_col=0)

c) Display the first seven rows of the DataFrame

d) Display the last seven rows of the DataFrame.

e) Use the DataFrame method describe (which looks only at the numerical columns) to calculate the descriptive statistics for the numerical columns—carat, depth, table, price, x, y and z.

f) Use Series method describe to calculate the descriptive statistics for the categorical data (text) columns—cut, color and clarity.

g) What are the unique category values (use the Series method unique)?

h) Pandas has many built-in graphing capabilities. Execute the %matplotlib magic to enable Matplotlib support in IPython. Then, to view histograms of each numerical data column, call your DataFrame's hist method. The following figure shows the results for the DataFrame's seven numerical column

In [38]:

```
diamonds_df = pd.read_csv("diamonds.csv", index_col = 0)
```

In [39]:

```
diamonds_df.head(7)
```

Out[39]:

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47

In [40]:

```
diamonds_df.tail(7)
```

Out[40]:

	carat	cut	color	clarity	depth	table	price	x	y	z
53934	0.70	Very Good	E	VS2	61.2	59.0	2757	5.69	5.72	3.49
53935	0.72	Premium	D	SI1	62.7	59.0	2757	5.69	5.73	3.58
53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

In [41]:

```
diamonds_df.describe()
```

Out[41]:

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

In [42]:

```
pd.Series.describe(diamonds_df['cut'])
```

Out[42]:

```
count      53940
unique         5
top        Ideal
freq       21551
Name: cut, dtype: object
```

In [44]:

```
pd.Series.describe(diamonds_df['color'])
```

Out[44]:

```
count      53940
unique         7
top            G
freq       11292
Name: color, dtype: object
```

In [45]:

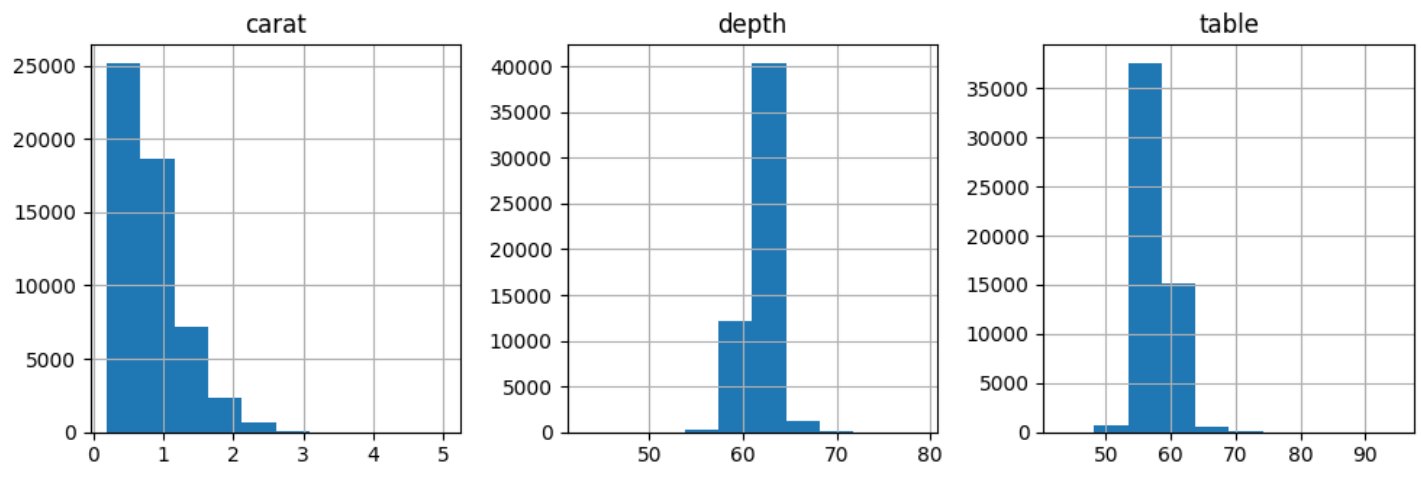
```
pd.Series.describe(diamonds_df['clarity'])
```

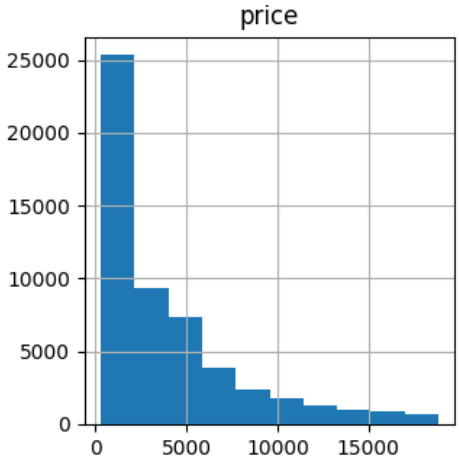
Out[45]:

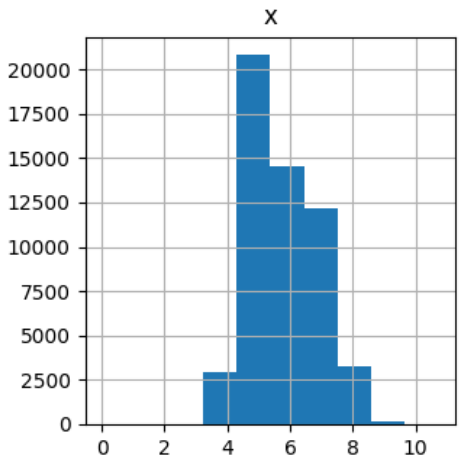
```
count      53940
unique         8
top          SI1
freq       13065
Name: clarity, dtype: object
```

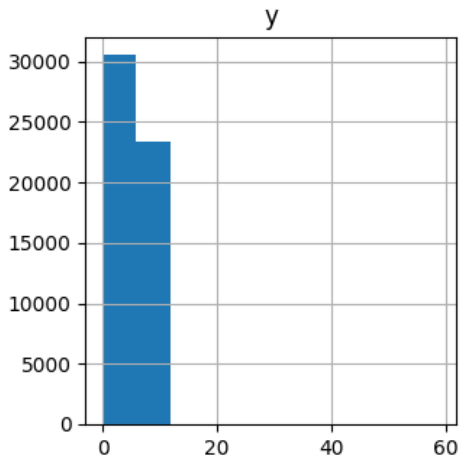
In [48]:

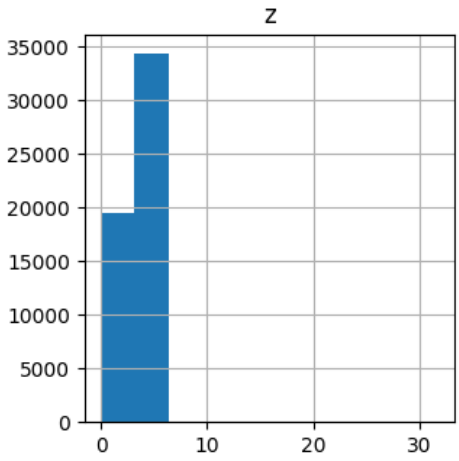
```
diamonds_df.hist(figsize=(10, 10))
plt.tight_layout()
plt.show()
```




price
25000
20000
15000
10000
5000
0
0
5000
10000
15000


x
20000
17500
15000
12500
10000
7500
5000
2500
0
0
2
4
6
8
10


y
30000
25000
20000
15000
10000
5000
0
0
20
40
60


z
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30

In [ ]: