```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import plotly.express as px #for animation
```

```
df = pd.read_csv('/content/diamonds.csv')
# to show the first five rows
df.head()
# to show last 5 rows
#df.tail()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
df.describe()
```

	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	3.539635
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.703869
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

```
df.shape
```

```
(53940, 10)
```

```
df[['carat','cut','clarity','color']]
```

	carat	cut	clarity	color
0	0.23	Ideal	SI2	E
1	0.21	Premium	SI1	E
2	0.23	Good	VS1	E
3	0.29	Premium	VS2	I
4	0.31	Good	SI2	J
...	...	...	...	...
53935	0.72	Ideal	SI1	D
53936	0.72	Good	SI1	D
53937	0.70	Very Good	SI1	D
53938	0.86	Premium	SI2	H
53939	0.75	Ideal	SI2	D

53940 rows × 4 columns

```
#filtering observation;
#if we write single df below it will return just the boolean value
df[df['x']==0]
```

	carat	cut	color	clarity	depth	table	price	x	y	z
11182	1.07	Ideal	F	SI2	61.6	56.0	4954	0.0	6.62	0.0
11963	1.00	Very Good	H	VS2	63.3	53.0	5139	0.0	0.00	0.0
15951	1.14	Fair	G	VS1	57.5	67.0	6381	0.0	0.00	0.0
24520	1.56	Ideal	G	VS2	62.2	54.0	12800	0.0	0.00	0.0
26243	1.20	Premium	D	VVS1	62.1	59.0	15686	0.0	0.00	0.0
27429	2.25	Premium	H	SI2	62.8	59.0	18034	0.0	0.00	0.0
49556	0.71	Good	F	SI2	64.1	60.0	2130	0.0	0.00	0.0
49557	0.71	Good	F	SI2	64.1	60.0	2130	0.0	0.00	0.0

```
filtering_for_two_variable= df[(df['x']>0) & (df['y']>0)]
filtering_for_two_variable
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53932 rows × 10 columns

```
filtering_for_two_variable[filtering_for_two_variable['x']==0].shape
```

```
(0, 10)
```

```
df['x'].median()
```

```
5.7
```

```
df['cut'].value_counts()
# returning unique values
```

```
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64
```

```
pd.crosstab(df['cut'],df['color'])
```

color	D	E	F	G	H	I	J
cut							
Fair	163	224	312	314	303	175	119
Good	662	933	909	871	702	522	307
Ideal	2834	3903	3826	4884	3115	2093	896
Premium	1603	2337	2331	2924	2360	1428	808
Very Good	1513	2400	2164	2299	1824	1204	678

```
df.groupby('cut')['price'].median().sort_values()
```

```
cut
Ideal        1810.0
Very Good    2648.0
```

```
Good         3050.5
Premium      3185.0
Fair         3282.0
Name: price, dtype: float64
```

```
# define a list of values
a= [3,5,12,9,4]
np.mean(a)
```

```
6.6
```

```
#manually define a function that can find the average given a list
def avg_list(x):
    sum_list= sum(x)
    count_list= len(x)
    return sum_list/count_list
```

```
# calling the function
avg_list(a)
```

```
6.6
```

```
#manually define a function that can find the variance given a list
def variance_list(x):
    avg=avg_list(x)
    n=len(x)
    variance = sum((item-avg) ** 2 for item in x )/(n-1)
    return variance
```

```
variance_list(a)
```

```
14.299999999999999
```

```
attack =[15,20,10,35,30,35,100]
variance_list(attack)
```

```
916.6666666666666
```

```
def stdev_list(x):
  return np.sqrt(variance_list(x))
```

```
stdev_list(attack)
```

```
30.276503540974915
```

```
# writting error function
def variance_list(x):
    if len(x)< 2:
        raise ValueError('not valid data, Please redefine the list')
    avg=avg_list(x)
    n=len(x)
    variance = sum((item-avg) ** 2 for item in x )/(n-1)
    return variance
```

```
# z-score; x is list value as value
def Zscore_list(x,value):
    avg=avg_list(x)
    stdev_x=stdev_list(x)
    return (value-avg)/ stdev_x
```

```
Zscore_list(attack,30)
```

```
-0.1651445647689541
```

```
# define the function to sun elementwise sum
def elementwise_sum_list(list1,list2):
  if (len(list1)!= len(list2)):
     raise ValueError('Two lists must have SAME size')
  return [list1[i]+list2[i] for i in range(len(list1))]
```

```
a=[3,5,12,9,4]
d=[1,0,0,1,0]
elementwise_sum_list(a,d)
```

```
     [4, 5, 12, 10, 4]
```

```
sorted(a)
```

```
     [3, 4, 5, 9, 12]
```

```
dd= 10.4
int(dd)
```

```
     10
```

```
f=dd-int(dd)
f
```

```
     0.40000000000000036
```

```
def percentile_list(p,x):
  n=len(x)
  sorted_x= sorted(x)
  loc=(p/100) * (n+1)
  int_part = int(loc)
  fractional_part= loc-int_part

  return sorted_x[int_part -1] + fractional_part * (sorted_x[int_part]-sorted_x[int_part -1])



salary= [3850,3710,3755,3890,3880,3880,3920,3940,3950,4050,4130,4325]
percentile_list(80,salary)
```

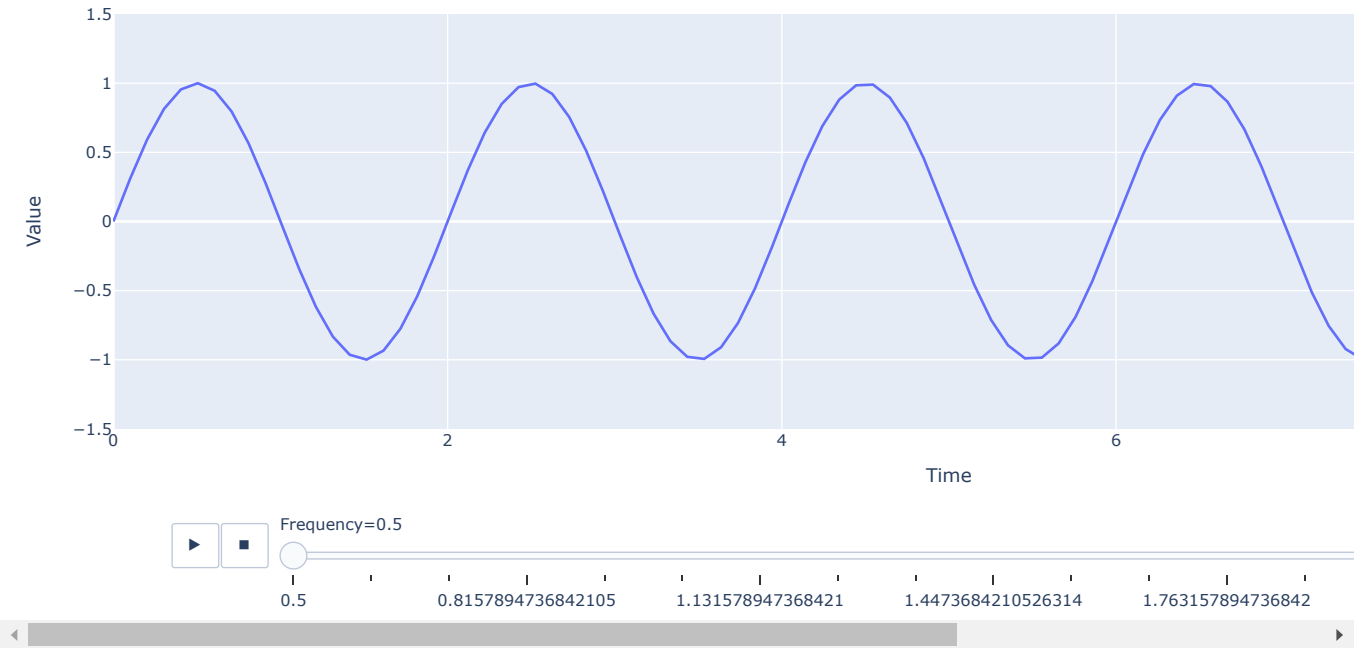
```
     4082.0
```

```
# defining sin function animation
# prepare data
time= np.linspace(0,10,100)
frequency=np.linspace(0.5,2.5,20)
# tranforming data into dataframe
data =pd.DataFrame(
    [{
        'Time': t,
        'Value': np.sin(2*np.pi*f*t),
        'Frequency': f
    } for f in frequency for t in time]

)
data.head()
```

	Time	Value	Frequency
0	0.00000	0.000000	0.5
1	0.10101	0.312033	0.5
2	0.20202	0.592908	0.5
3	0.30303	0.814576	0.5
4	0.40404	0.954902	0.5

```
fig= px.line(data,x='Time',y='Value',animation_frame='Frequency',
             range_y=[-1.5,1.5])
fig.show()
```




```
# moving bubble chart
#preparing the data
np.random.seed(42)
years= np.arange(2000,2021)
years
```

```
array([2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
       2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020])
```

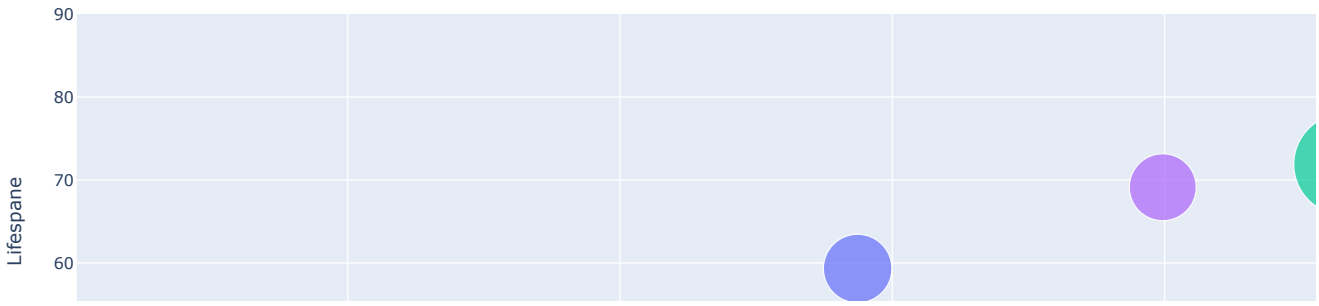
```
countries=['USA','FR','JP','UK']
```

```
data ={
    'Year': np.repeat(years, 4), # using years for 4 countries
    'Country': np.tile(countries,len(years)),
    'GDP': np.random.rand(len(years)* len(countries))* 50000 +10000,
    'Lifespane': np.random.rand(len(years)*len(countries))*30 +50,
    'Population': np.random.rand(len(years)* len(countries))*1000000+500000
}

df= pd.DataFrame(data)
df.head()
```

	Year	Country	GDP	Lifespane	Population
0	2000	USA	28727.005942	59.329470	5.407751e+05
1	2000	FR	57535.715320	59.755500	1.090893e+06
2	2000	JP	46599.697091	71.888185	1.177564e+06
3	2000	UK	39932.924210	69.126724	5.165878e+05
4	2001	USA	17800.932022	76.616382	1.012093e+06

```
fig= px.scatter(df,x='GDP',y='Lifespane',animation_frame='Year',
             animation_group='Country',size='Population', color='Country',
             hover_name='Country', size_max=60, range_x=[0,60000],range_y=[40,90])
fig.show()
```


90
80
70
60
Lifespane