In [1]:
```
# importing libraries
import pandas as pd
import numpy as np
```

In [2]:
```
# reading data from a csv file
# data1=pd.read_csv\
#("C:/Users/B314-CSE-SYS27/Downloads/gapminder-FiveYearData.csv")
data1=pd.read_csv(r"C:\Users\B314-CSE-SYS27\Downloads\gapminder-FiveYearData
data1
```

Out[2]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106
...	...	...	...	...	...	...
1699	Zimbabwe	1987	9216418.0	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340.0	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948.0	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563.0	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143.0	Africa	43.487	469.709298

1704 rows × 6 columns

In [47]:
```
# to see the entire database
pd.set_option("display.max_rows", 1704)
df=pd.read_csv("C:/Users/B314-CSE-SYS27/Downloads/gapminder-FiveYearData.csv
df
```

Out[47]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8.425333e+06	Asia	28.80100	779.445314
1	Afghanistan	1957	9.240934e+06	Asia	30.33200	820.853030
2	Afghanistan	1962	1.026708e+07	Asia	31.99700	853.100710
3	Afghanistan	1967	1.153797e+07	Asia	34.02000	836.197138
4	Afghanistan	1972	1.307946e+07	Asia	36.08800	739.981106
5	Afghanistan	1977	1.488037e+07	Asia	38.43800	786.113360
6	Afghanistan	1982	1.288182e+07	Asia	39.85400	978.011439
7	Afghanistan	1987	1.386796e+07	Asia	40.82200	852.395945
8	Afghanistan	1992	1.631792e+07	Asia	41.67400	649.341395
9	Afghanistan	1997	2.222742e+07	Asia	41.76300	635.341351
10	Afghanistan	2002	2.526840e+07	Asia	42.12900	726.734055

In [4]:
```
# to print the first 5 records
data1.head()
```

Out[4]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106

In [5]:
```
df.head(7)
```

Out[5]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106
5	Afghanistan	1977	14880372.0	Asia	38.438	786.113360
6	Afghanistan	1982	12881816.0	Asia	39.854	978.011439

In [6]:
```
data1.shape
```

Out[6]:
```
(1704, 6)
```

In [7]:
```
data1.columns
```

Out[7]:
```
Index(['country', 'year', 'pop', 'continent', 'lifeExp', 'gdpPercap'], dty
pe='object')
```

In [8]:
```
data1.dtypes
```

Out[8]:
```
country       object
year           int64
pop          float64
continent     object
lifeExp      float64
gdpPercap    float64
dtype: object
```

In [9]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   country    1704 non-null   object 
 1   year       1704 non-null   int64  
 2   pop        1704 non-null   float64
 3   continent  1704 non-null   object 
 4   lifeExp    1704 non-null   float64
 5   gdpPercap  1704 non-null   float64
dtypes: float64(3), int64(1), object(2)
memory usage: 80.0+ KB
```

In [10]:
```
# to get a specific column and store it in another variable
country=data1['country']
country
```

Out[10]:
```
0             Afghanistan
1             Afghanistan
2             Afghanistan
3             Afghanistan
4             Afghanistan
5             Afghanistan
6             Afghanistan
7             Afghanistan
8             Afghanistan
9             Afghanistan
10            Afghanistan
11            Afghanistan
12                Albania
13                Albania
14                Albania
15                Albania
16                Albania
17                Albania
18                Albania
```

In [11]:
```
country.head()
```

Out[11]:
```
0    Afghanistan
1    Afghanistan
2    Afghanistan
3    Afghanistan
4    Afghanistan
Name: country, dtype: object
```

In [12]:
```
country.tail()
```

Out[12]:
```
1699    Zimbabwe
1700    Zimbabwe
1701    Zimbabwe
1702    Zimbabwe
1703    Zimbabwe
Name: country, dtype: object
```

In [13]:
```
# to print the last 5 records
data1.tail()
```

Out[13]:

	country	year	pop	continent	lifeExp	gdpPercap
1699	Zimbabwe	1987	9216418.0	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340.0	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948.0	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563.0	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143.0	Africa	43.487	469.709298

In [14]:
```
data1.tail(7)
```

Out[14]:

	country	year	pop	continent	lifeExp	gdpPercap
1697	Zimbabwe	1977	6642107.0	Africa	57.674	685.587682
1698	Zimbabwe	1982	7636524.0	Africa	60.363	788.855041
1699	Zimbabwe	1987	9216418.0	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340.0	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948.0	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563.0	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143.0	Africa	43.487	469.709298

In [15]:
```
# subset contains specific columns of the database
subset=data1[['country','continent']]
subset.head(3)
```

Out[15]:

	country	continent
0	Afghanistan	Asia
1	Afghanistan	Asia
2	Afghanistan	Asia

In [32]:
```
# subsetting columns
subset2=data1.loc[:,['country','continent']]
subset2.head(3)
```

Out[32]:

	country	continent
0	Afghanistan	Asia
1	Afghanistan	Asia
2	Afghanistan	Asia

In [18]:
```
subset3=data1.iloc[:,[0,3]]
subset3.head(3)
```

Out[18]:

	country	continent
0	Afghanistan	Asia
1	Afghanistan	Asia
2	Afghanistan	Asia

In [19]:
```
data1.loc[42,'country']
```

Out[19]:
```
'Angola'
```

In [23]:
```
# retrun the record at the specified location
data1.loc[3]
```

Out[23]:
```
country      Afghanistan
year                1967
pop          11537966.0
continent            Asia
lifeExp             34.02
gdpPercap      836.197138
Name: 3, dtype: object
```

In [21]:
```
data1.iloc[3]
```

Out[21]:
```
country      Afghanistan
year                1967
pop          11537966.0
continent            Asia
lifeExp             34.02
gdpPercap      836.197138
Name: 3, dtype: object
```

In [31]:
```
# subsetting rows
data1.loc[[1,3,5]]
```

Out[31]:

	country	year	pop	continent	lifeExp	gdpPercap
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
5	Afghanistan	1977	14880372.0	Asia	38.438	786.113360

In [24]:
```
# return the last record
data1.iloc[-1]
```

Out[24]:
```
country        Zimbabwe
year               2007
pop          12311143.0
continent        Africa
lifeExp           43.487
gdpPercap    469.709298
Name: 1703, dtype: object
```

In [33]:
```
# subsetting rows and columns
data1.loc[10:13,['country','year']]
```

Out[33]:

	country	year
10	Afghanistan	2002
11	Afghanistan	2007
12	Albania	1952
13	Albania	1957

In [27]:
```
data1.iloc[10:13,[0,1]]
```

Out[27]:

	country	year
10	Afghanistan	2002
11	Afghanistan	2007
12	Albania	1952

In [28]:
```
# returns the mean of the specified column
data1['year'].mean()
```

Out[28]:
```
1979.5
```

In [35]:
```
# grouped mean
data1.groupby('year')['lifeExp'].mean()
```

Out[35]:
```
year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

In [50]:
```
multi_group=\
data1.groupby(['year','continent'])['lifeExp','gdpPercap'].mean()
print(multi_group.head(10))
```

```
                  lifeExp     gdpPercap
year continent
1952 Africa     39.135500   1252.572466
     Americas   53.279840   4079.062552
     Asia       46.314394   5195.484004
     Europe     64.408500   5661.057435
     Oceania    69.255000  10298.085650
1957 Africa     41.266346   1385.236062
     Americas   55.960280   4616.043733
     Asia       49.318544   5787.732940
     Europe     66.703067   6963.012816
     Oceania    70.295000  11598.522455

C:\Users\B314-CSE-SYS27\AppData\Local\Temp\ipykernel_5504\2288589498.py:2:
FutureWarning: Indexing with multiple keys (implicitly converted to a tupl
e of keys) will be deprecated, use a list instead.
  data1.groupby(['year','continent'])['lifeExp','gdpPercap'].mean()
```

In [36]:
```
# to flatten the dataframe
flat_data=multi_group.reset_index()
flat_data.head(n=6)
```

Out[36]:

	year	continent	lifeExp	gdpPercap
0	1952	Africa	39.135500	1252.572466
1	1952	Americas	53.279840	4079.062552
2	1952	Asia	46.314394	5195.484004
3	1952	Europe	64.408500	5661.057435
4	1952	Oceania	69.255000	10298.085650
5	1957	Africa	41.266346	1385.236062

In [37]:
```
# grouped frequency counts
data1.groupby('continent')['country'].nunique()
```

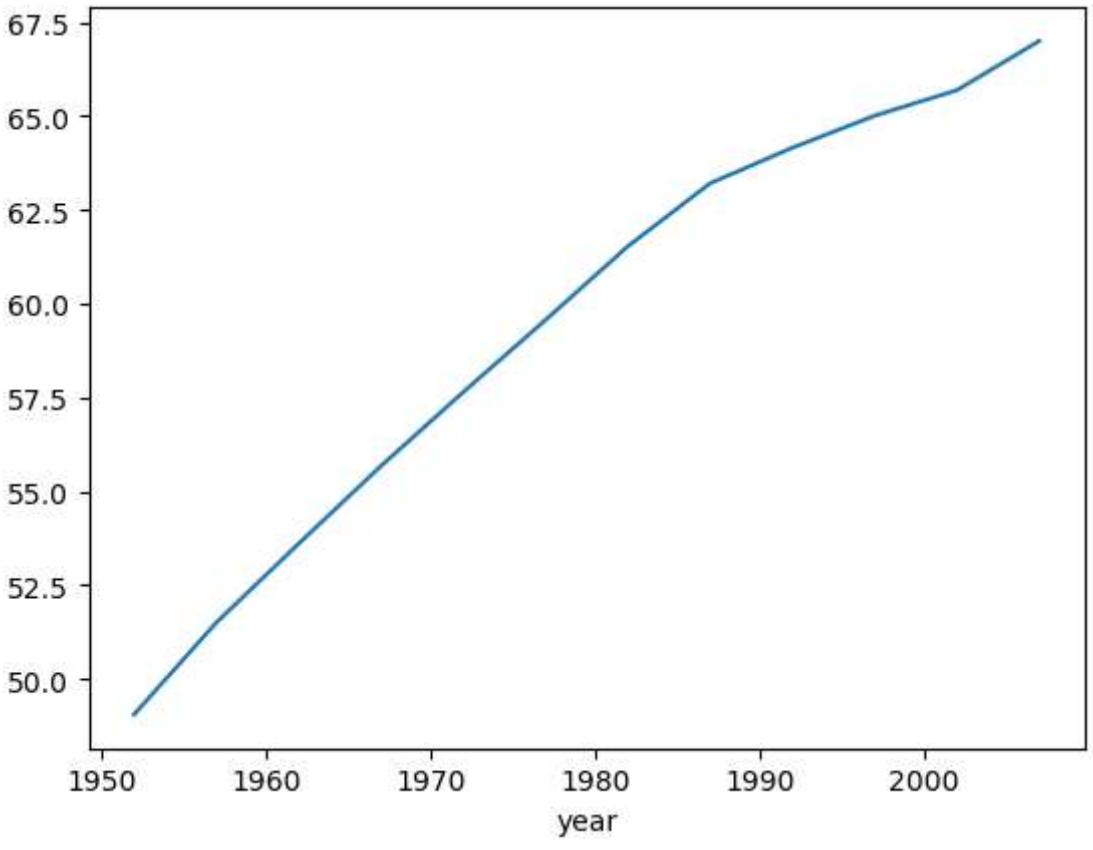
Out[37]:
```
continent
Africa      52
Americas    25
Asia        33
Europe      30
Oceania      2
Name: country, dtype: int64
```

In [38]:
```
# Basic plot
global_yearly_lifeExpectancy=data1.groupby('year')['lifeExp'].mean()
print(global_yearly_lifeExpectancy)
```

```
year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

In [39]:
```
# representatio of the plot
global_yearly_lifeExpectancy.plot()
```

Out[39]: <Axes: xlabel='year'>



In [40]:
```
# reading data from a tab delimited file
# data2=pd.read_csv\
#("C:/Users/B314-CSE-SYS27/Downloads/tabDelimitedFile.txt",sep="\t",header=N
data2=pd.read_csv\
("C:/Users/B314-CSE-SYS27/Downloads/tabDelimitedFile.txt", sep="\t")
data2
```

Out[40]:

	No.	Name	Dept.	Section
0	1	Laasya	CSE	D
1	2	Lekha	CSE	D
2	3	Madhuri	CSE	D
3	4	Swena	CSE	D
4	5	Madhu	CSE	D
5	6	Vibha	CSE	D

In [41]:
```
# reading data from an inbuilt dataset
from sklearn.datasets import load_iris
iris=load_iris()
iris
```

```
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
```

In [42]:
```
# returns the output labels
iris.target_names
```

Out[42]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [43]:
```
# returns the input features
iris.data
```

Out[43]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

In [44]:
```
iris.target
```

Out[44]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [45]:
```
# reading an excel file
data3=pd.read_excel(r"C:\Users\B314-CSE-SYS27\Downloads\ExcelFile.xlsx")
data3
```

Out[45]:

	No.	Name	Dept	Section
0	1	Laasya	CSE	D
1	2	Lekha	CSE	D
2	3	Madhuri	CSE	D
3	4	Swena	CSE	D

In [ ]:

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statistics as st
from scipy import stats as sct
```

Reading Data

In [3]:
```
ex_data = pd.read_excel(r'/content/IBM-313 Marks.xlsx')
ex_data
```

Out[3]:

	S.No.	midexam	miniproject	total_internal	endexam	total
0	1	5.00	20	25.00	12.0	37.00
1	2	11.05	20	31.05	26.0	57.05
2	3	8.10	20	28.10	14.0	42.10
3	4	6.00	10	16.00	13.0	29.00
4	5	11.35	20	31.35	17.0	48.35
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05
75	76	12.25	10	22.25	28.0	50.25
76	77	1.75	10	11.75	28.0	0.00
77	78	3.00	10	13.00	11.0	0.00
78	79	5.80	10	15.80	12.0	27.80

79 rows × 6 columns

In [4]:
```
#value at row 1 and column 1
ex_data.iloc[1,1]
```

Out[4]:
```
11.05
```

In [5]:
```
#all columns
print(ex_data.columns)
```

```
Index(['S.No.', 'midexam', 'miniproject', 'total_internal', 'endexam',
       'total'],
      dtype='object')
```

In [6]:
```
#dataframe shape
print(ex_data.shape)
```

```
(79, 6)
```

In [7]:
```
#total column
x = ex_data['total']
print(x)
```

```
0     37.00
1     57.05
2     42.10
3     29.00
4     48.35
      ...
74    42.05
75    50.25
76     0.00
77     0.00
78    27.80
Name: total, Length: 79, dtype: float64
```

Central Tendency

- mean
- median
- mode
- percentile
- range
- IQR
- Standard deviation
- Variance
- Skew

In [8]:
```
np.mean(x)
```

Out[8]:
```
46.90632911392405
```

In [9]:
```
x.mean()
```

Out[9]:
```
46.90632911392405
```

In [10]:
```
np.median(x)
```

Out[10]:
```
45.0
```

In [11]:
```
x.median()
```

Out[11]:
```
45.0
```

In [12]:
```
import statistics as st
st.mode(x)
```

Out[12]:
```
48.35
```

In [15]:
```
sct.mode(x)
```

Out[15]:
```
ModeResult(mode=0.0, count=2)
```

In [16]:
```
arr = np.array([1,2,3,4,5])
print(np.percentile(arr, 50))
```

```
3.0
```

In [17]:
```
r1 = max(x)
r1
```

Out[17]:
```
94.5
```

In [18]:
```
r2 = min(x)
r2
```

Out[18]:
```
0.0
```

In [19]:
```
range = r1 - r2
range
```

Out[19]:
```
94.5
```

In [20]:
```
Q3 = np.percentile(arr,75)
print(Q3)
```

```
4.0
```

In [21]:
```
Q1 = np.percentile(arr,25)
print(Q1)
```

```
2.0
```

In [22]:
```
IQR = Q3 - Q1
print(IQR)
```

```
2.0
```

In [23]:
```
lower_ex = Q1 - 1.5*IQR
lower_ex
```

Out[23]:
```
-1.0
```

In [24]:
```
upper_ex = Q3 + 1.5*IQR
upper_ex
```

Out[24]:
```
7.0
```

In [25]:
```
np.std(x)
```

Out[25]:
```
16.210536046955966
```

In [26]:
```
x.std()
```

Out[26]:
```
16.31411880088133
```

Difference between variance and numpy variance

numpy variance - population variance

variance - sample variance

In [27]:
```
np.var(x)
```

Out[27]:
```
262.7814789296587
```

In [28]:
```
x.var()
```

Out[28]:
```
266.1504722492697
```

In [29]:
```
from scipy.stats import skew
skew(x)
```

Out[29]:
```
0.10226407464884266
```

In [30]:
```
x.skew()
```

Out[30]:
```
0.10425411710908208
```

In [31]:
```
np.percentile(x, x.mean())
```

Out[31]:
```
43.79346835443038
```

In [32]:
```
np.percentile(x, x.median())
```

Out[32]:
```
43.5
```

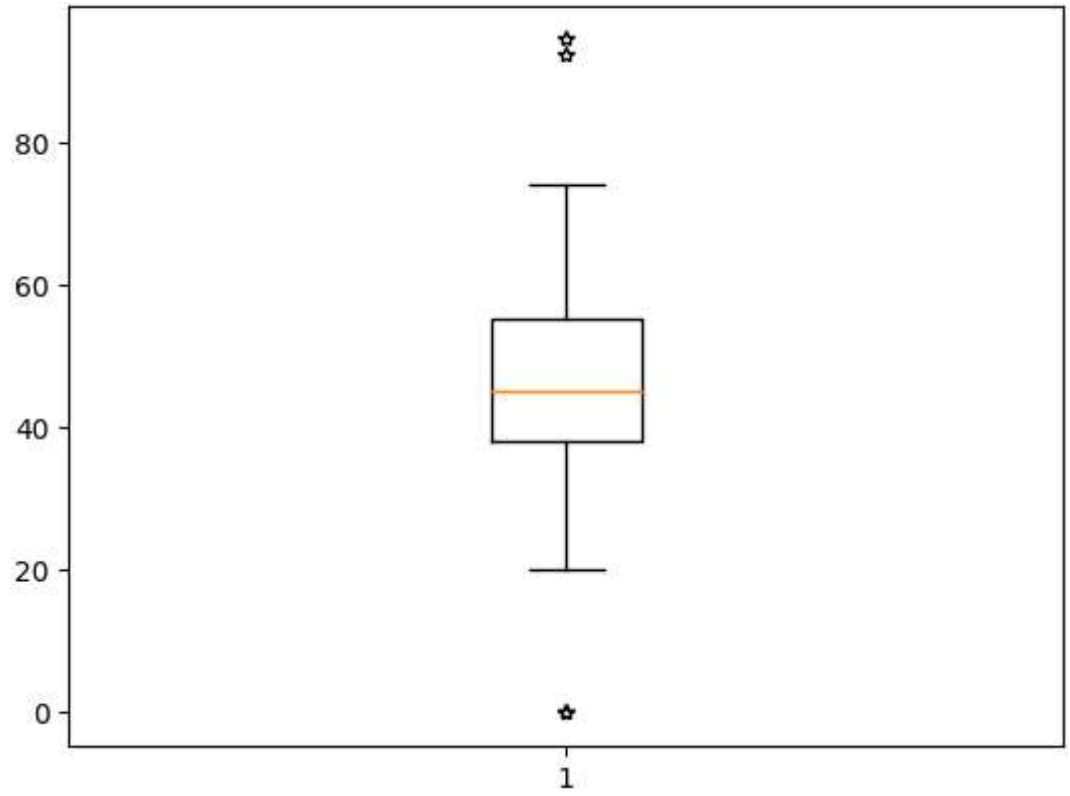
matplotlib.pyplot

-boxplot

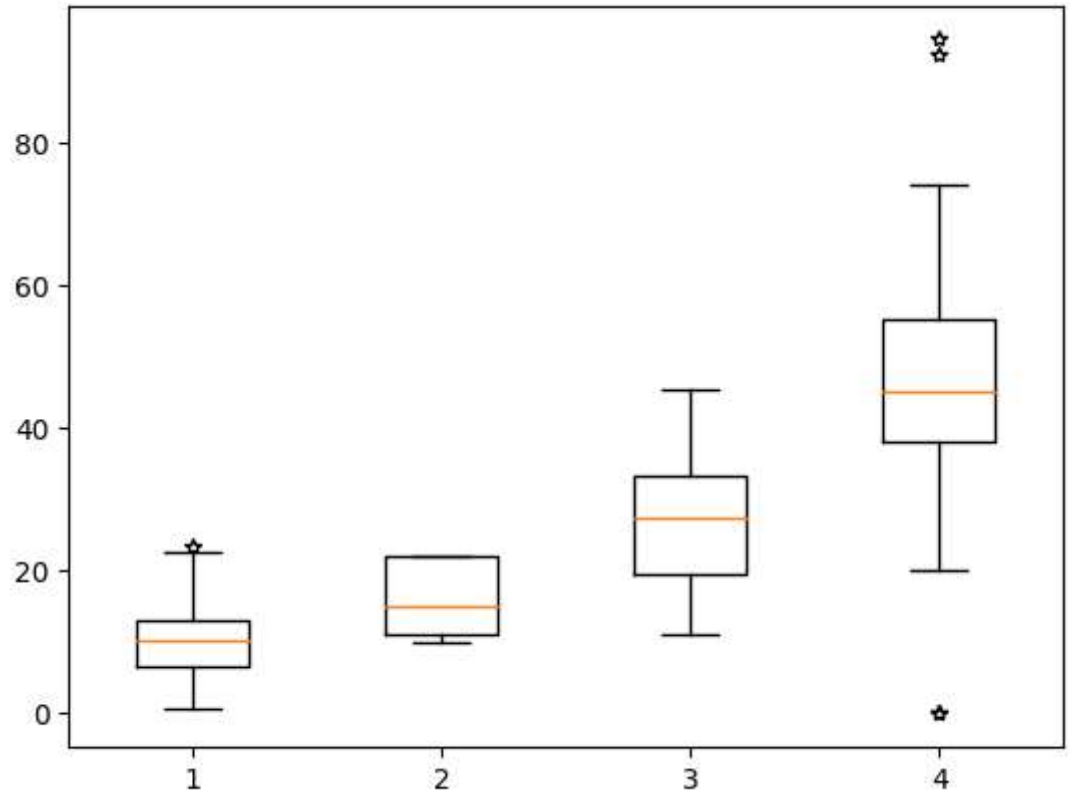
-seaborn.boxplot

-data.describe

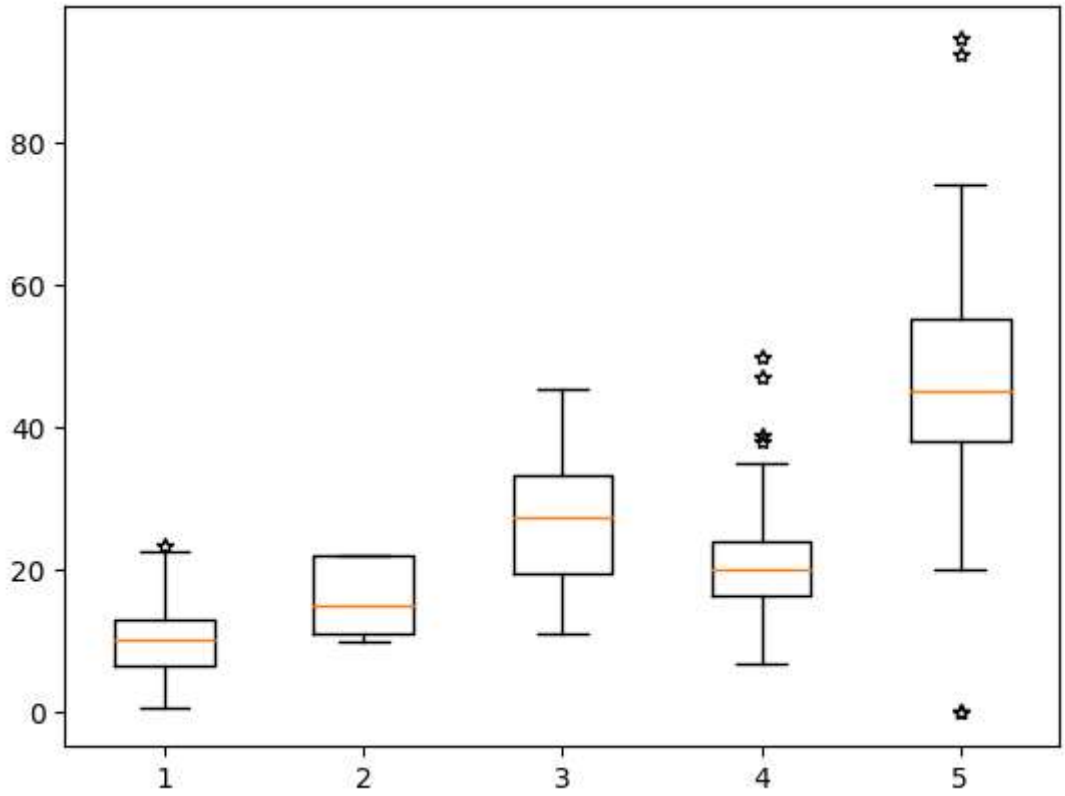
In [33]:
```
plt.boxplot(x, sym='*')
plt.show()
```



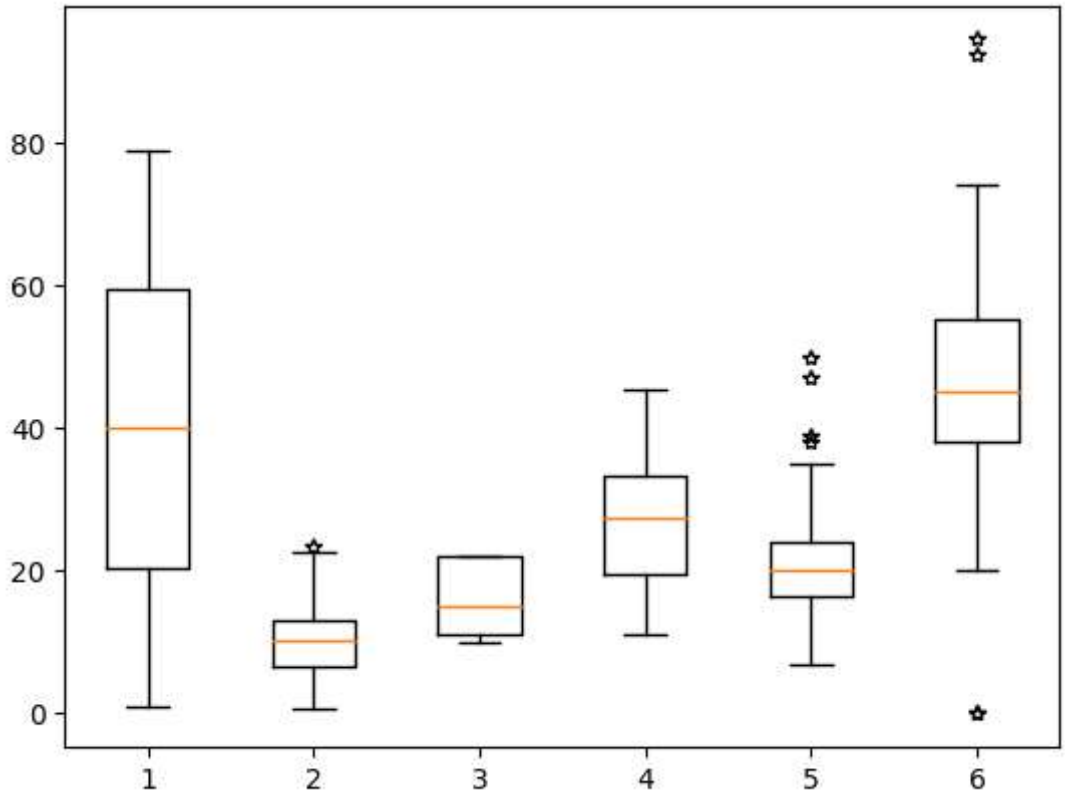
In [34]:
```
plt.boxplot(ex_data[['midexam', 'miniproject', 'total_internal','total']], sym='
plt.show()
```



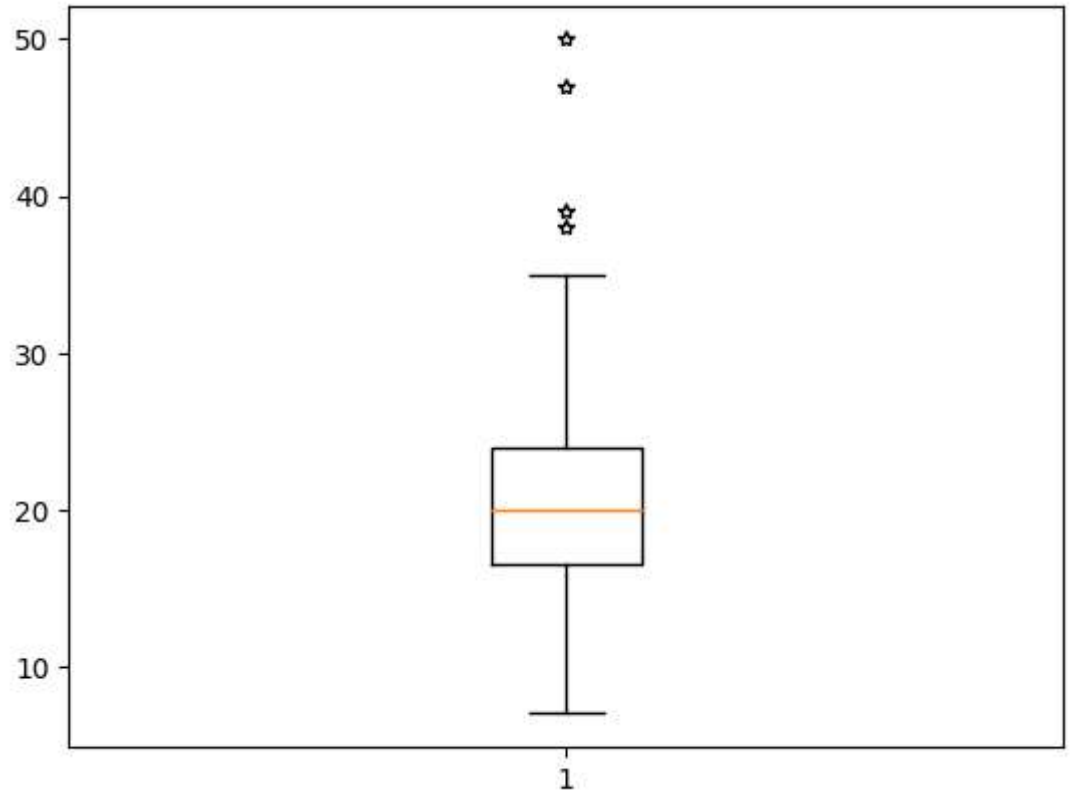
In [35]:
```
plt.boxplot(ex_data[['midexam', 'miniproject', 'total_internal','endexam','total
plt.show()
```



In [36]:
```
plt.boxplot(ex_data, sym ='*')
plt.show()
```



In [37]:
```
plt.boxplot(ex_data['endexam'], sym='*')
plt.show()
```

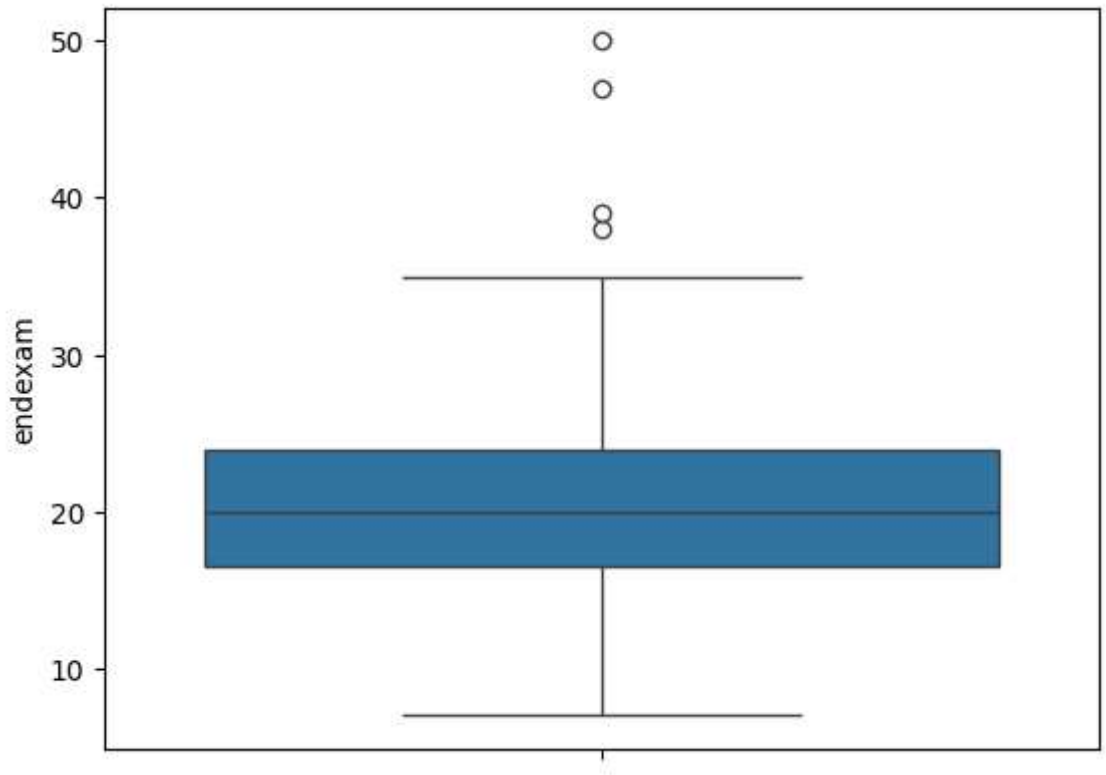


Seaborn Library and Matplotlib differences:

- Matplotlib uses numpy and pandas for plotting graphs, uses complex syntaxes.
- Seaborn uses matplotlib along with numpy and pandas for generating graphs, uses simpler sytaxes.

In [38]:
```
sns.boxplot(ex_data['endexam'])
```

Out[38]:
```
<Axes: ylabel='endexam'>
```



In [39]:
```
#return numerical description of the data in the DataFrame
ex_data.describe()
```

Out[39]:

	S.No.	midexam	miniproject	total_internal	endexam	total
count	79.000000	79.000000	79.000000	79.000000	79.000000	79.000000
mean	40.000000	10.178481	16.556962	26.735443	20.977848	46.906329
std	22.949219	4.961924	4.900934	8.504976	8.105493	16.314119
min	1.000000	0.700000	10.000000	11.200000	7.000000	0.000000
25%	20.500000	6.500000	11.000000	19.600000	16.500000	38.000000
50%	40.000000	10.300000	15.000000	27.500000	20.000000	45.000000
75%	59.500000	12.975000	22.000000	33.250000	24.000000	55.375000
max	79.000000	23.500000	22.000000	45.500000	50.000000	94.500000

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
ex_data = pd.read_excel(r'C:\Users\Vinni\Downloads\IBM-313 Marks.xlsx')
ex_data
```

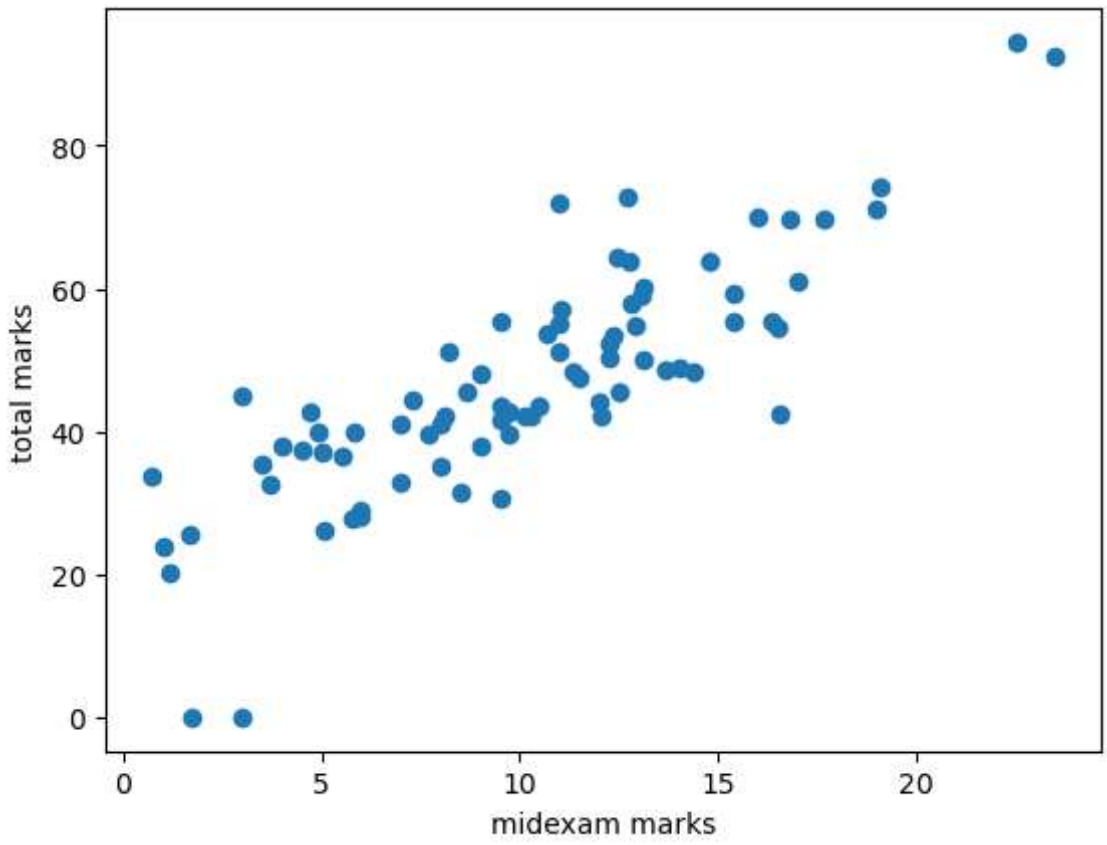
Out[2]:

	S.No.	midexam	miniproject	total_internal	endexam	total
0	1	5.00	20	25.00	12.0	37.00
1	2	11.05	20	31.05	26.0	57.05
2	3	8.10	20	28.10	14.0	42.10
3	4	6.00	10	16.00	13.0	29.00
4	5	11.35	20	31.35	17.0	48.35
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05
75	76	12.25	10	22.25	28.0	50.25
76	77	1.75	10	11.75	28.0	0.00
77	78	3.00	10	13.00	11.0	0.00
78	79	5.80	10	15.80	12.0	27.80

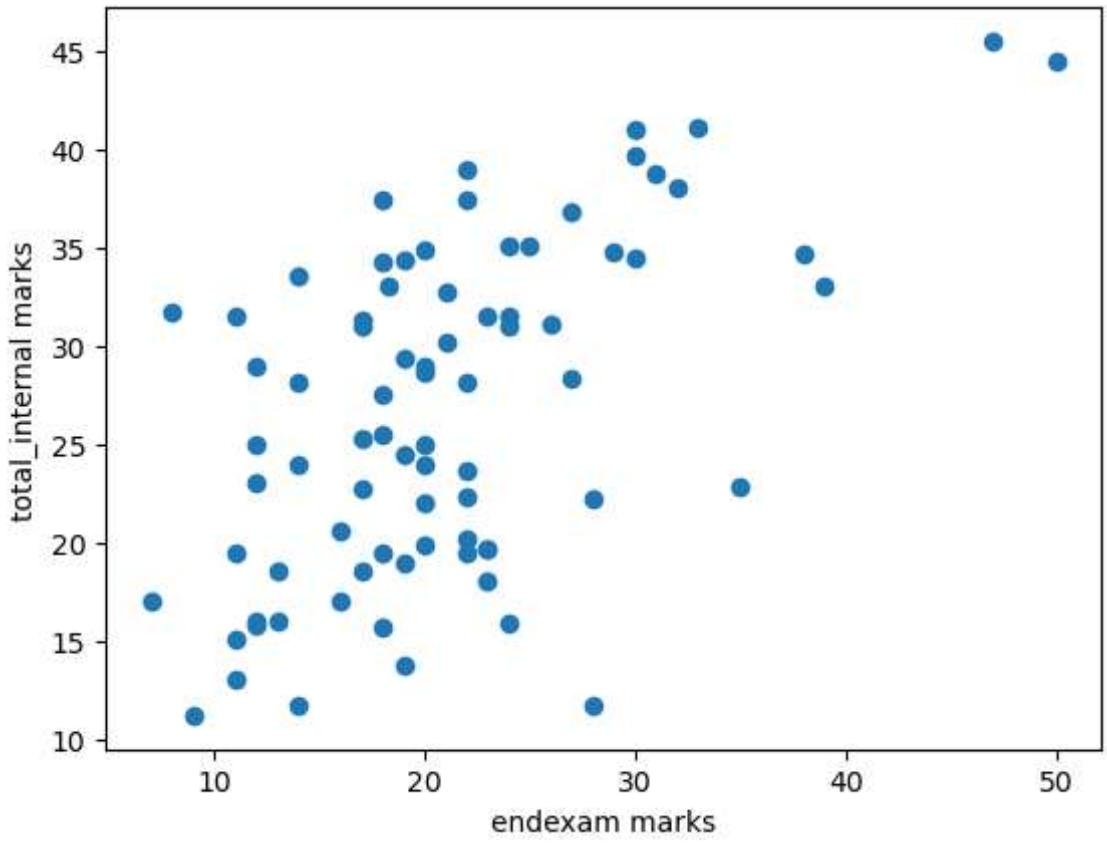
79 rows × 6 columns

Scatter Plot

In [3]:
```
plt.scatter(ex_data['midexam'] , ex_data['total'])
plt.xlabel('midexam marks')
plt.ylabel('total marks')
plt.show()
```



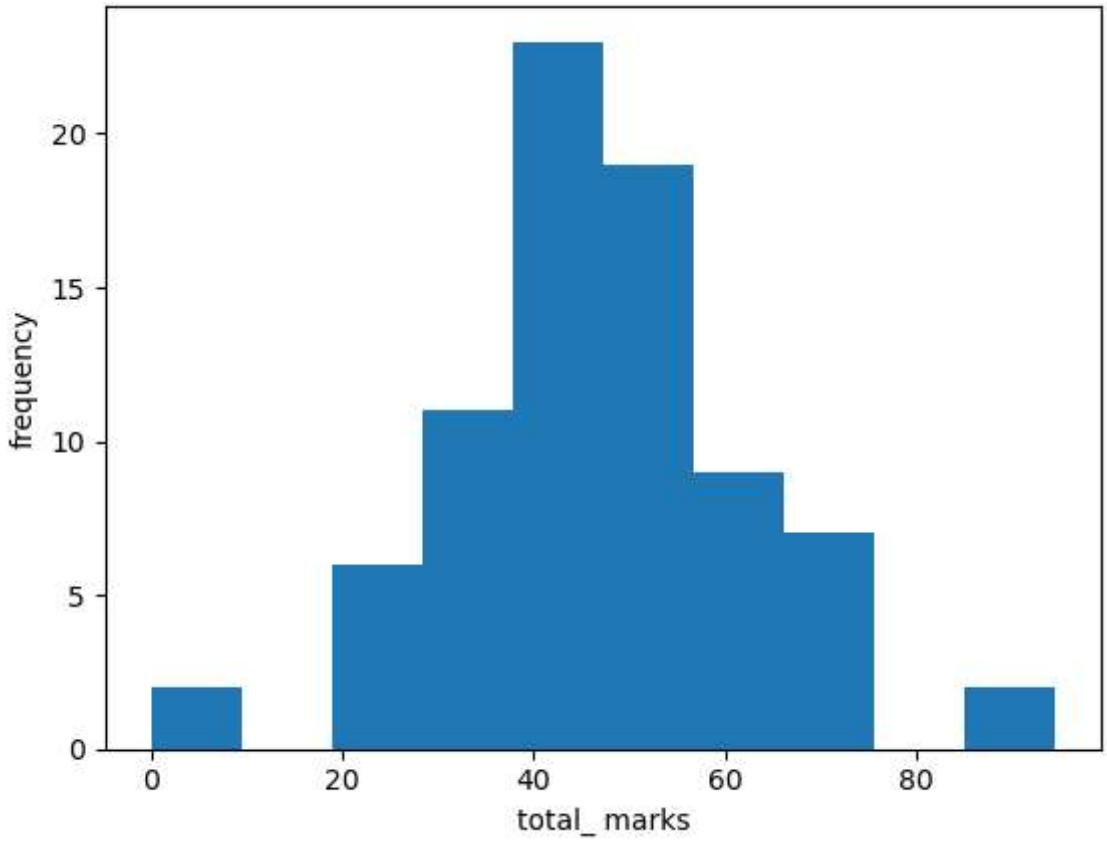
In [4]:
```
plt.scatter(ex_data['endexam'] , ex_data['total_internal'])
plt.xlabel('endexam marks')
plt.ylabel('total_internal marks')
plt.show()
```



Histogram

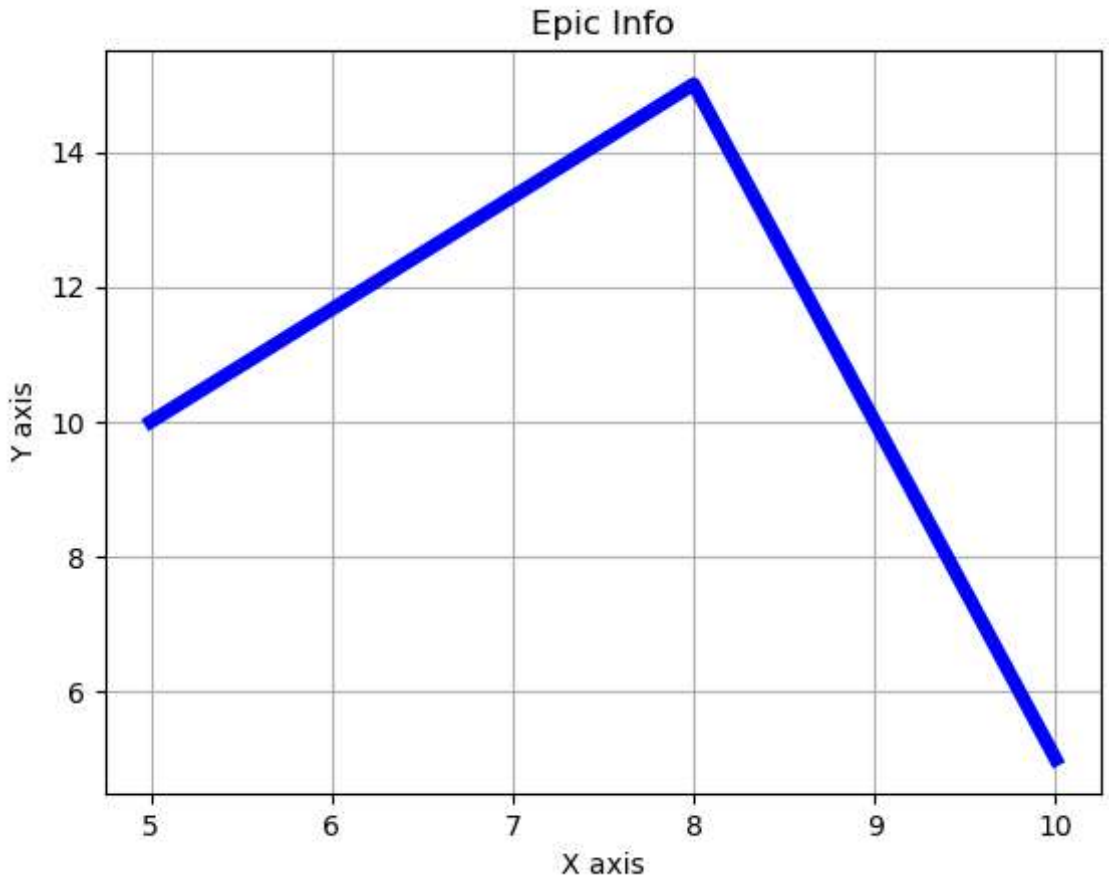
In [5]:
```
plt.hist(ex_data['total'])
plt.xlabel('total_ marks')
plt.ylabel('frequency')
```

Out[5]:
```
Text(0, 0.5, 'frequency')
```

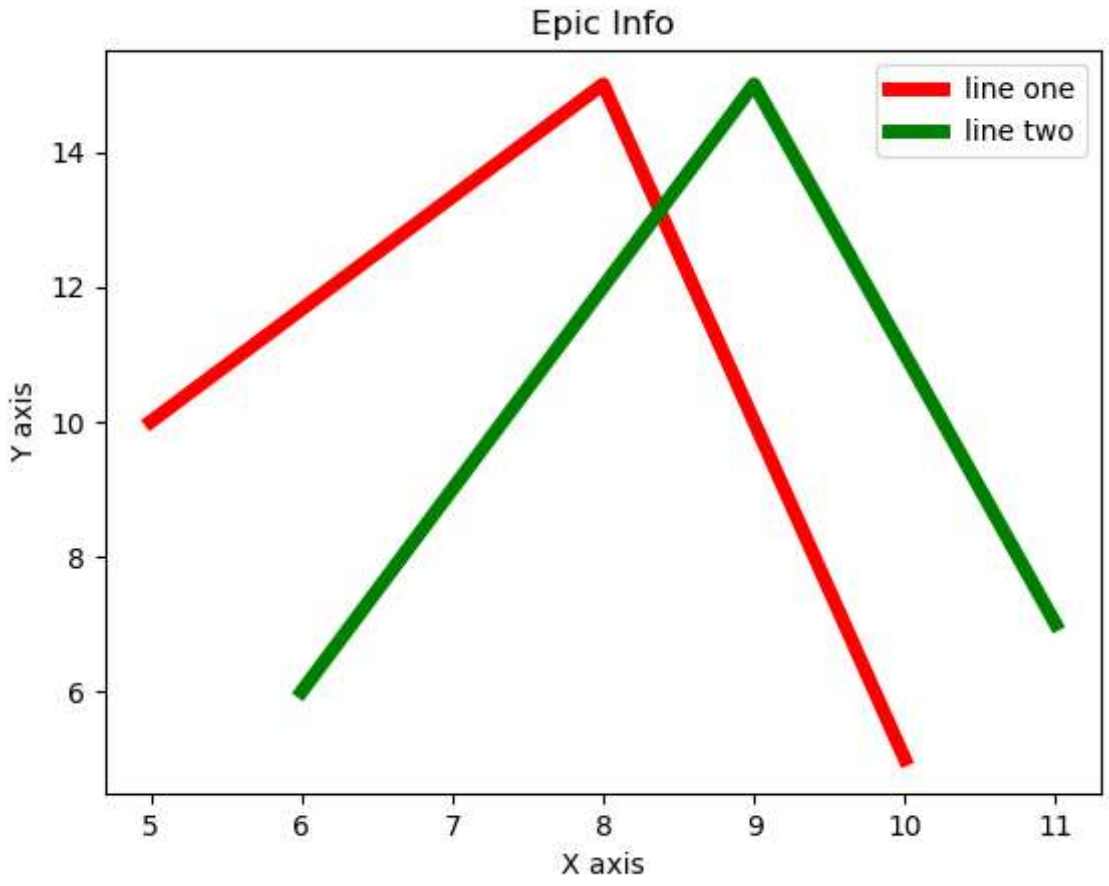


Line Graph

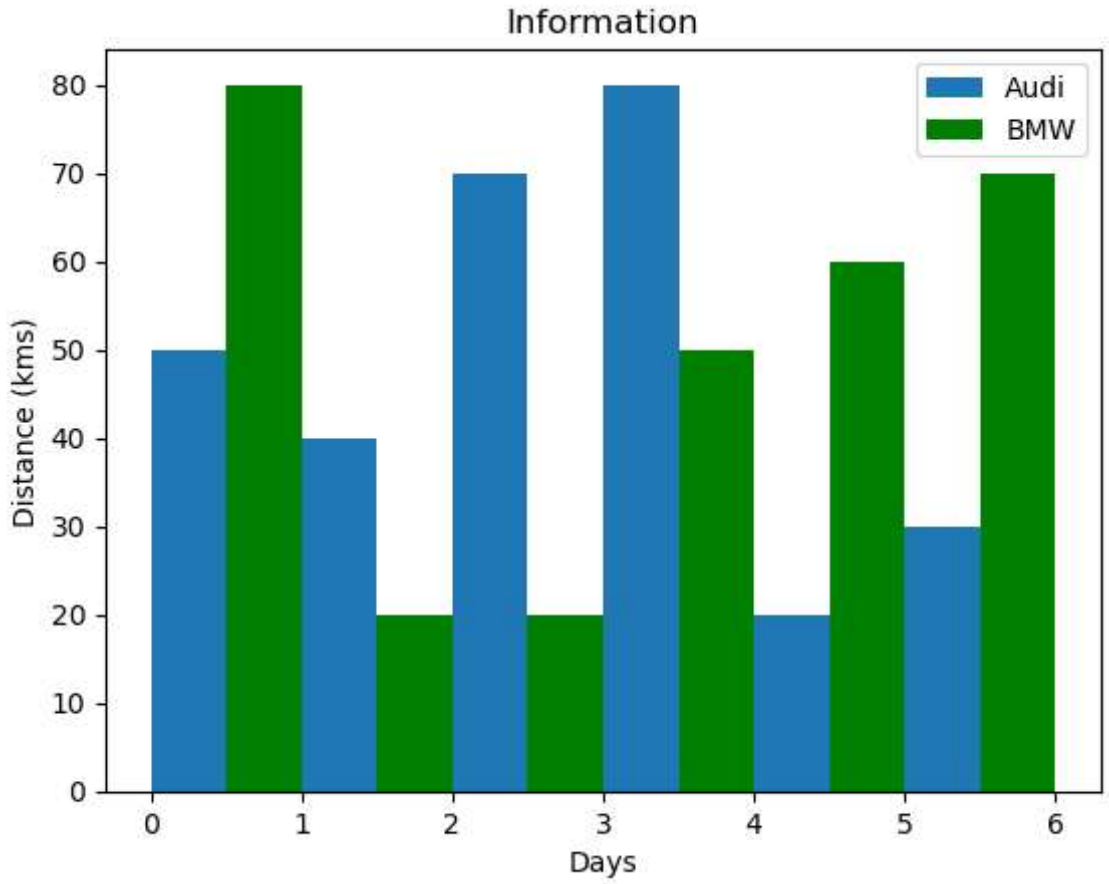
In [6]:
```
x = [5,8,10]
y = [10,15,5]
plt.plot(x,y,label='line one', linewidth=5, color='b')
plt.title('Epic Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.grid(True)
plt.show()
```



In [7]:
```
x = [5,8,10]
y = [10,15,5]
x2 = [6,9,11]
y2 = [6,15,7]
plt.plot(x,y,label='line one', linewidth=5, color='r')
plt.plot(x2,y2,label='line two',linewidth=5, color='g')
plt.title('Epic Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.legend()
plt.show()
```

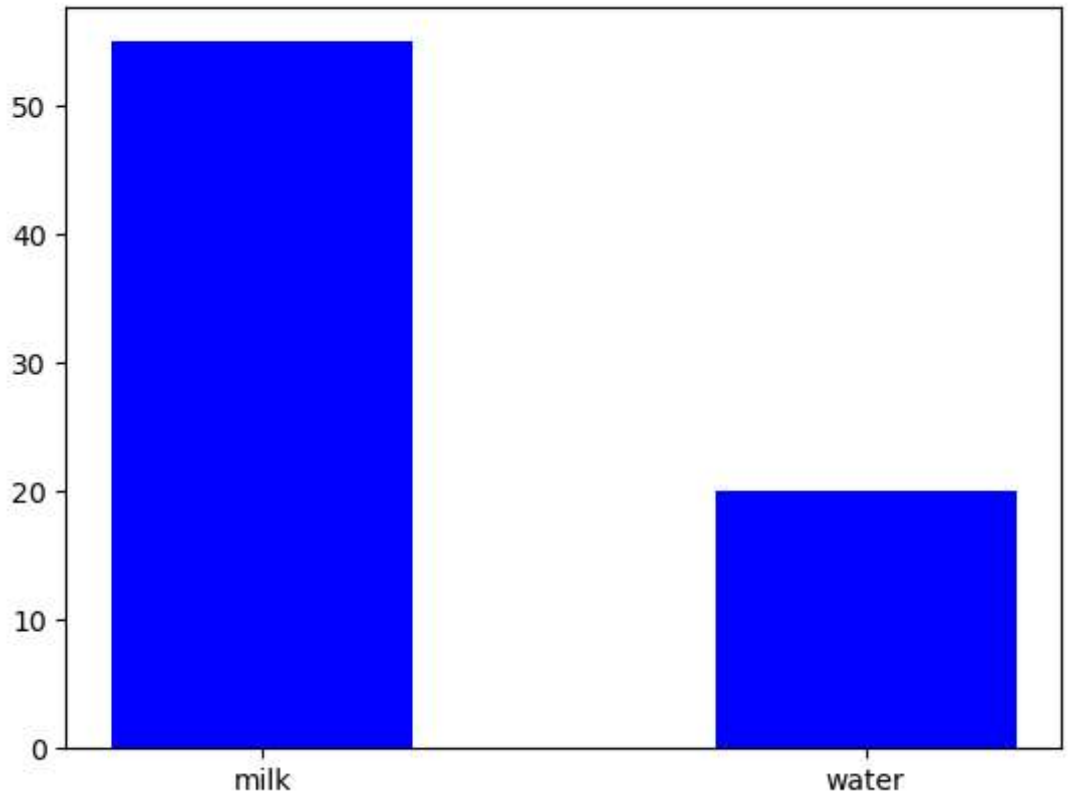


In [8]:
```
plt.bar([0.25,1.25,2.25,3.25,4.25,5.25],[50,40,70,80,20,30],
label="Audi",width=.5)
plt.bar([.75,1.75,2.75,3.75,4.75, 5.75],[80,20,20,50,60,70],
label="BMW", color='g',width=.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Information')
plt.show()
```

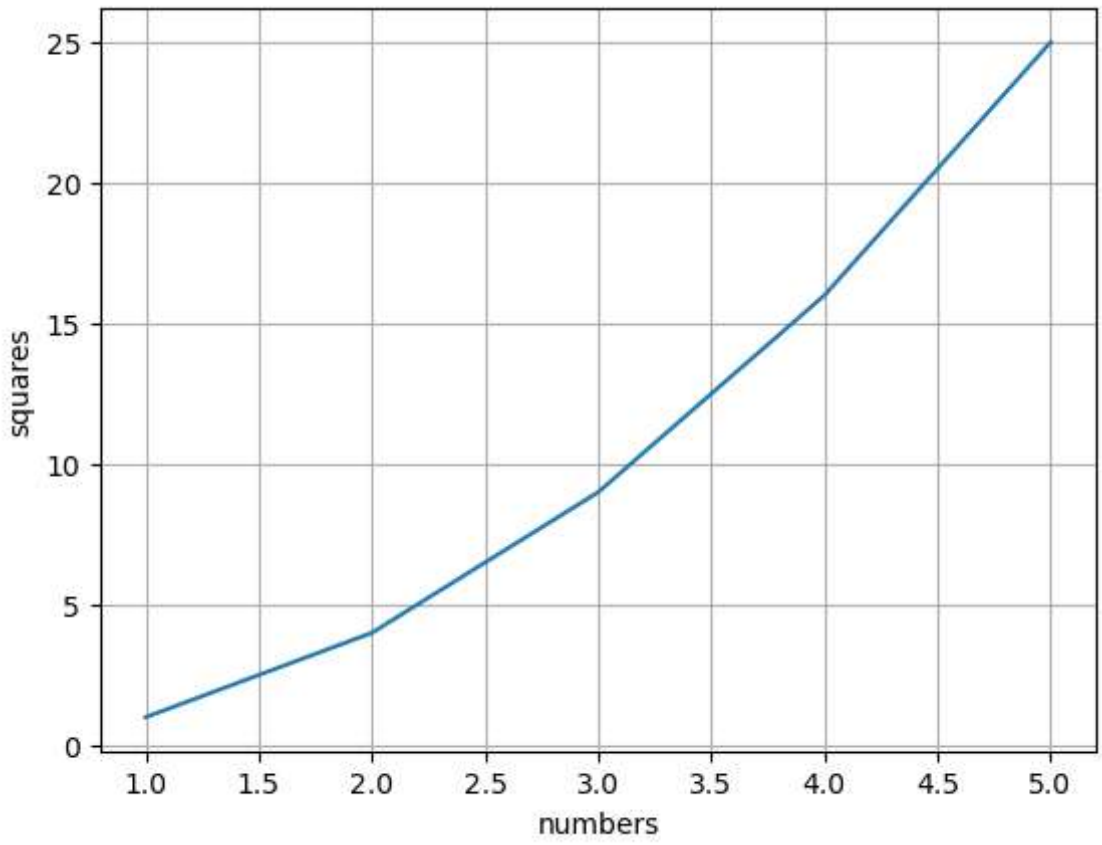


In [9]:
```
data = {'milk': 55, 'water': 20}
names = list(data.keys())
values = list(data.values())
plt.bar(names, values, color = 'b', width=0.5)
```

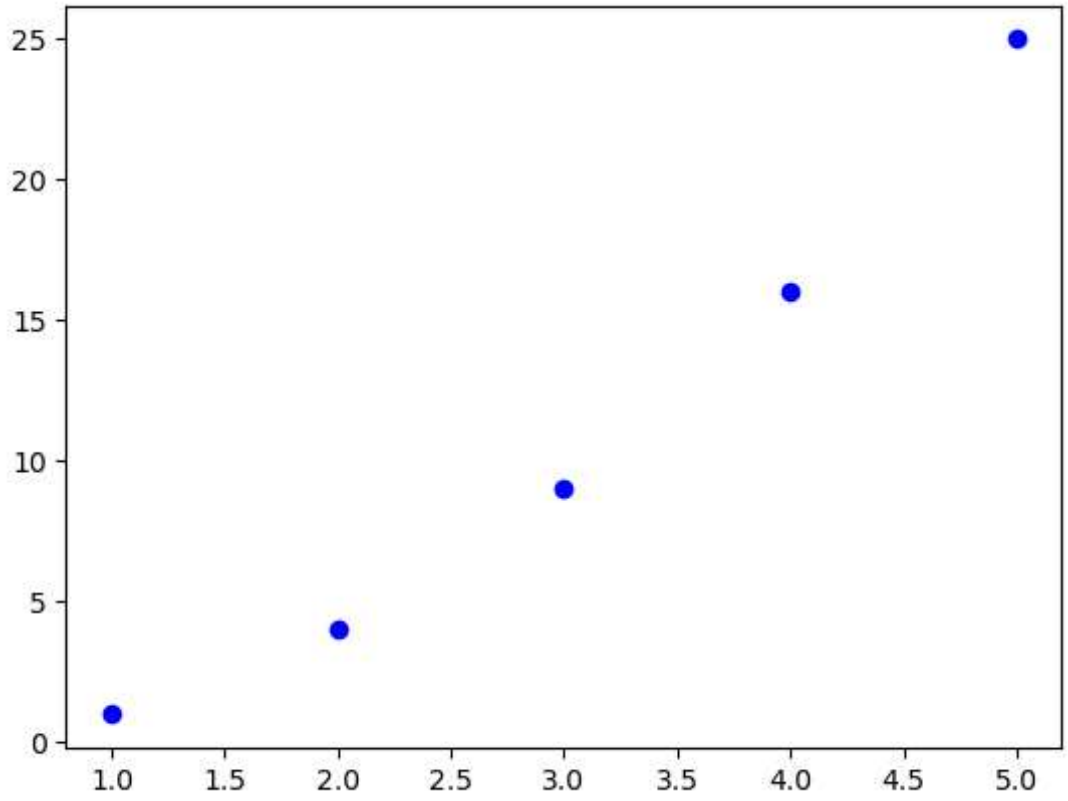
Out[9]:
```
<BarContainer object of 2 artists>
```



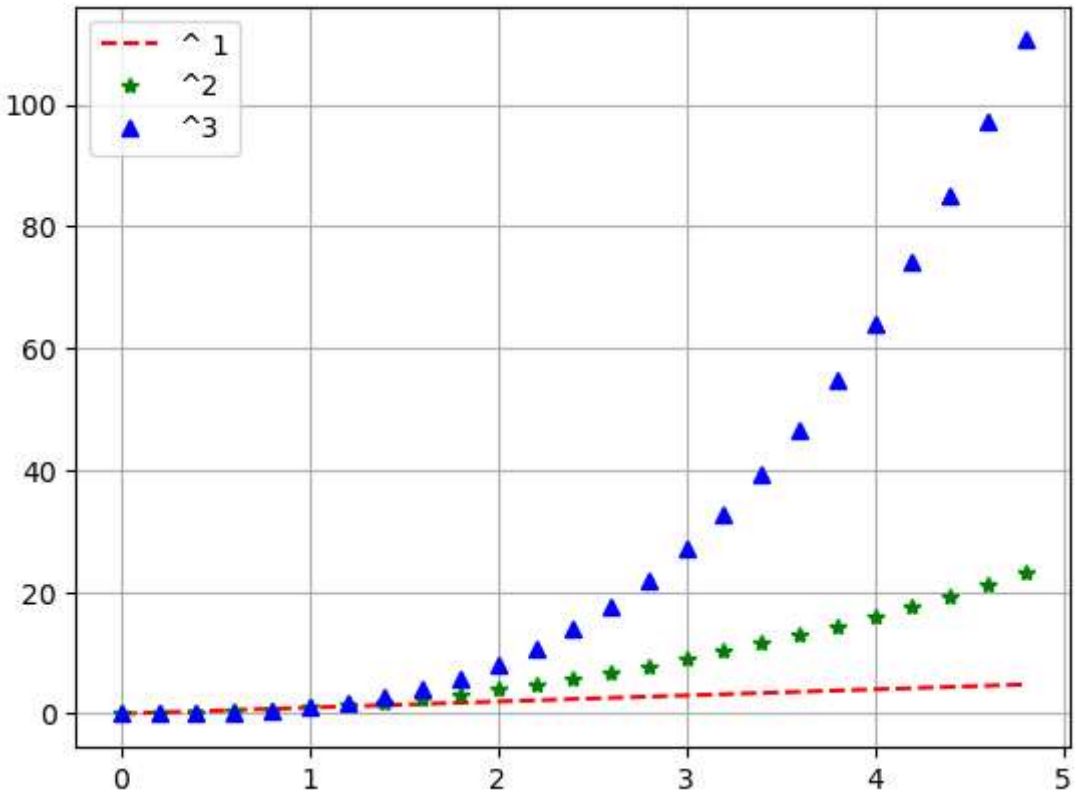
In [10]:
```
plt.plot([1, 2, 3, 4,5], [1, 4, 9, 16,25])
plt.ylabel('squares')
plt.xlabel('numbers')
plt.grid()
plt.show()
```



In [13]:
```
plt.plot([1, 2, 3, 4,5], [1, 4, 9, 16,25], 'bo')
plt.grid(False)
plt.savefig(r'D:\ML lab\myplot.png')
plt.savefig(r'D:\ML lab\myplot.pdf')
plt.show()
```



In [12]:
```
import numpy as np
t = np.arange(0., 5., 0.2)
plt.plot(t, t**1, 'r--', label='^ 1')#   'rs',   'g^')
plt.plot(t,t**2, 'g*', label='^2')
plt.plot(t, t**3, 'b^', label='^3')
plt.grid()
plt.legend()
plt.show()
```



- arange function used to generate a sequence of numbers within a specified range

In [14]:
```
t = np.arange(0., 5., 0.2)
print(t)
```

```
[0.  0.2 0.4 0.6 0.8 1.  1.2 1.4 1.6 1.8 2.  2.2 2.4 2.6 2.8 3.  3.2 3.4
 3.6 3.8 4.  4.2 4.4 4.6 4.8]
```

In [15]:
```
t**1
```

Out[15]:
```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4,
       2.6, 2.8, 3. , 3.2, 3.4, 3.6, 3.8, 4. , 4.2, 4.4, 4.6, 4.8])
```

In [16]:
```
t**2
```

Out[16]:
```
array([ 0.  ,  0.04,  0.16,  0.36,  0.64,  1.  ,  1.44,  1.96,  2.56,
        3.24,  4.  ,  4.84,  5.76,  6.76,  7.84,  9.  , 10.24, 11.56,
       12.96, 14.44, 16.  , 17.64, 19.36, 21.16, 23.04])
```

In [17]:
```
res = t**4
```

In [18]:
```
x = ['{:f}'.format(item) for item in res]
x
```

Out[18]:
```
['0.000000',
 '0.001600',
 '0.025600',
 '0.129600',
 '0.409600',
 '1.000000',
 '2.073600',
 '3.841600',
 '6.553600',
 '10.497600',
 '16.000000',
 '23.425600',
 '33.177600',
 '45.697600',
 '61.465600',
 '81.000000',
 '104.857600',
 '133.633600',
 '167.961600',
 '208.513600',
 '256.000000',
 '311.169600',
 '374.809600',
 '447.745600',
 '530.841600']
```

In [20]:
```
wine_data = pd.read_csv(r'C:\Users\Vinni\Downloads\wine-quality-white-and-red.csv'
wine_data
```

Out[20]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	white	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45
1	white	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49
2	white	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44
3	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40
4	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40
...	...	...	...	...	...	...	...	...	...	...	...
6492	red	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58
6493	red	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76
6494	red	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
6495	red	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
6496	red	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

6497 rows × 13 columns

In [21]:
```
bin_edges = np.arange(10, wine_data['residual sugar'].max() + 1, 20)
wine_data['residual sugar'].min()+1
```

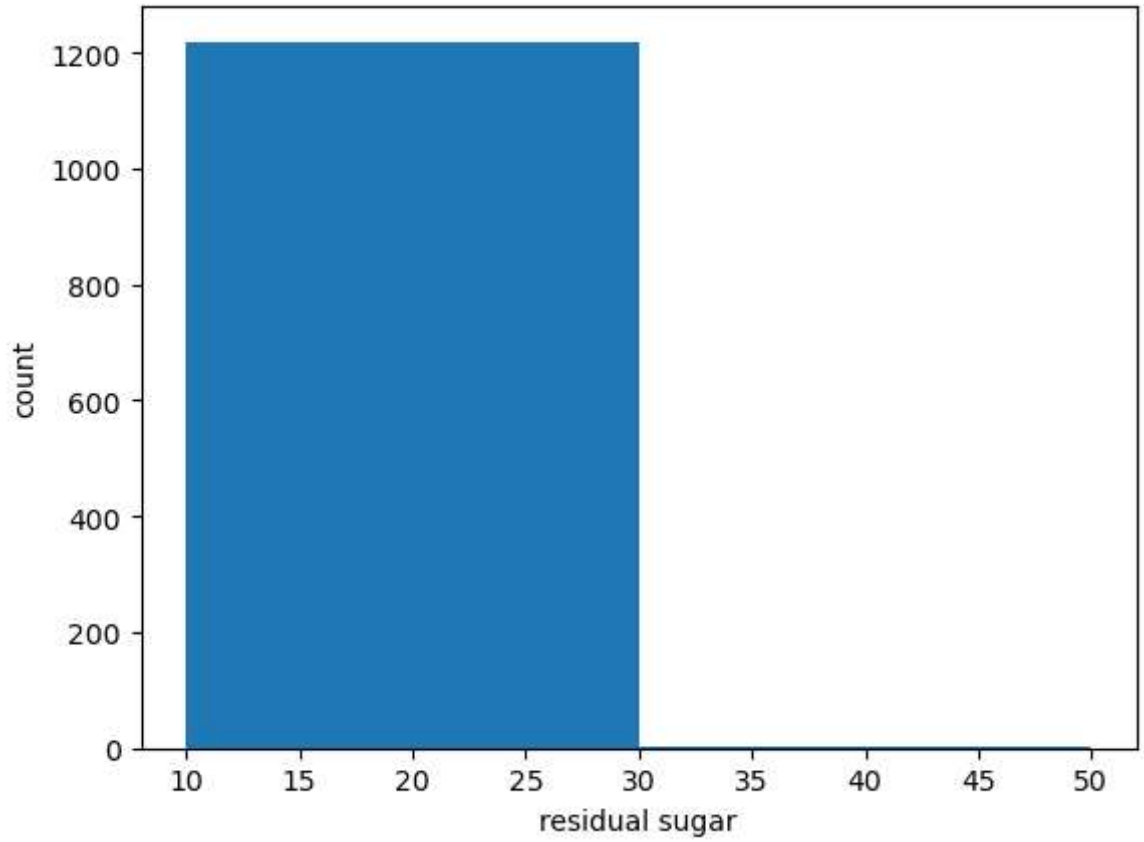
```
wine_data['residual sugar']
```

Out[21]:
```
0       20.7
1        1.6
2        6.9
3        8.5
4        8.5
        ...
6492     2.0
6493     2.2
6494     2.3
6495     2.0
6496     3.6
Name: residual sugar, Length: 6497, dtype: float64
```

In [22]:
```
bin_edges
```

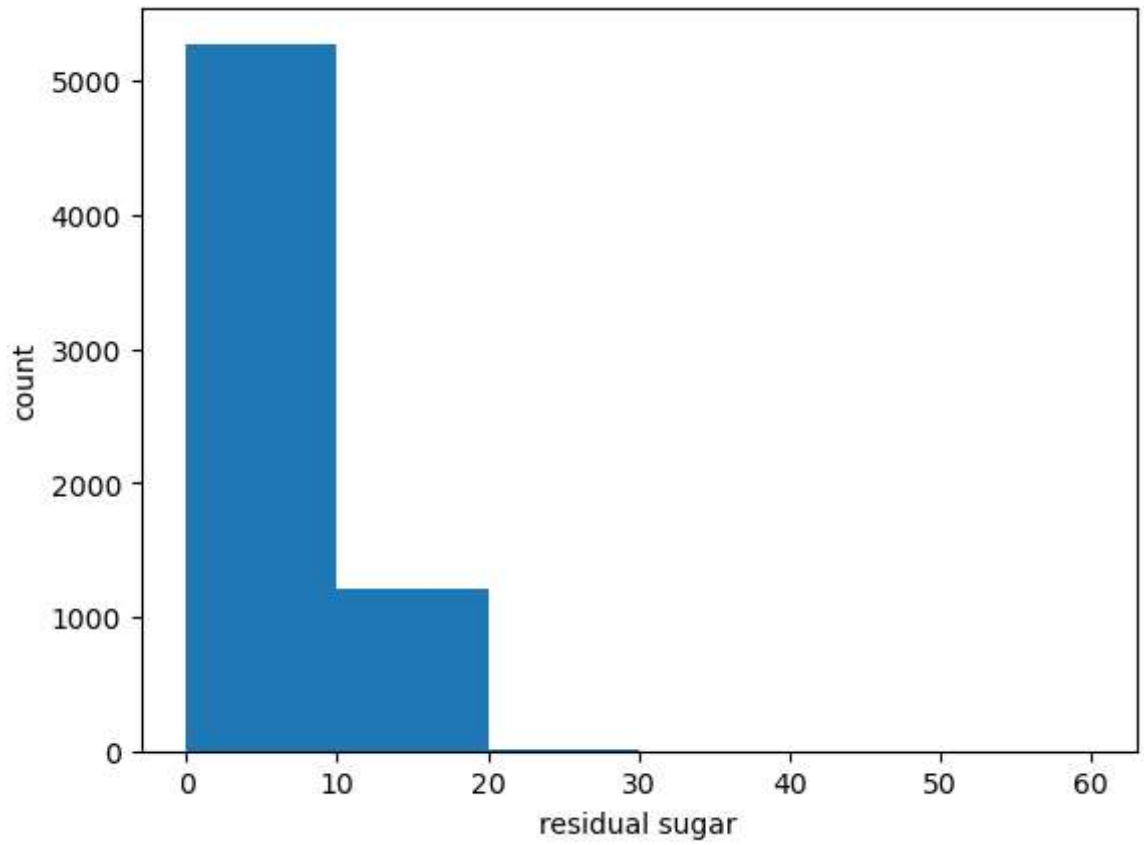
Out[22]:
```
array([10., 30., 50.])
```

In [23]:
```
fig = plt.hist(wine_data['residual sugar'], bins=bin_edges)
plt.xlabel('residual sugar')
plt.ylabel('count')
plt.show()
```

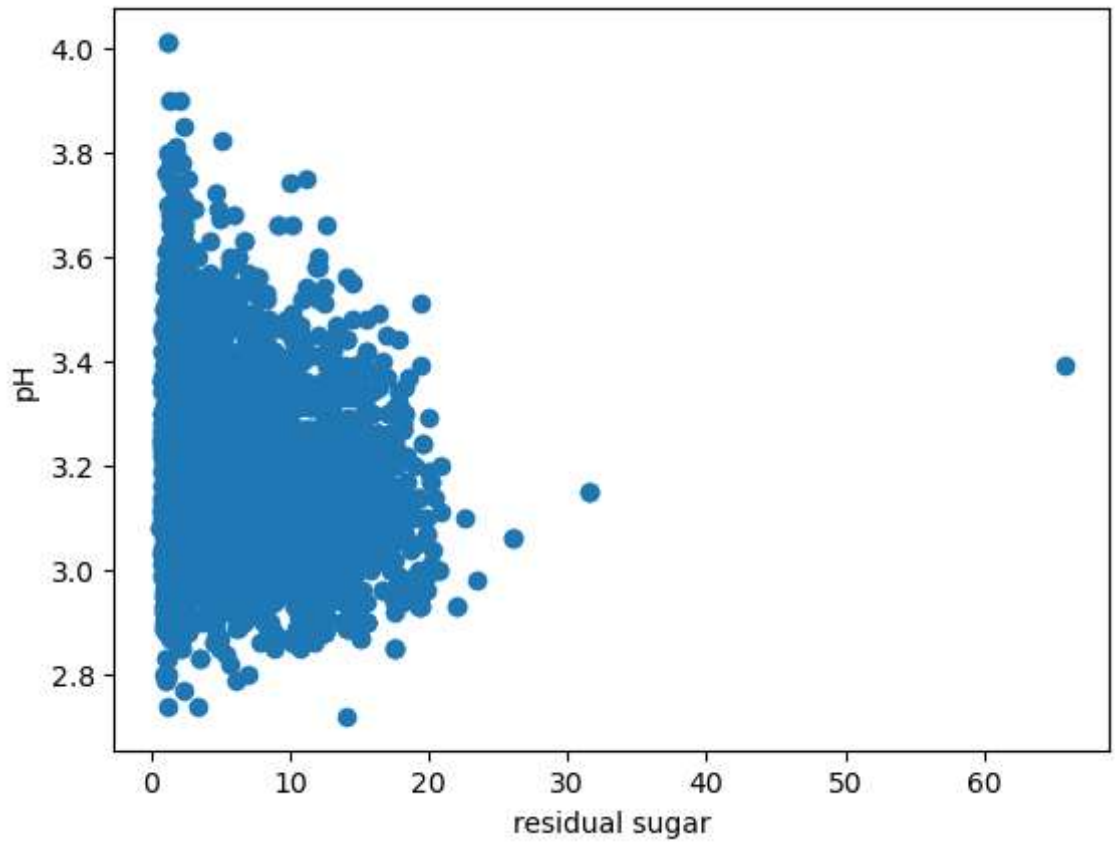


In [24]:
```
bin_edges2 = np.arange(0, wine_data['residual sugar'].max() + 1, 10)
```

In [25]:
```
fig2 = plt.hist(wine_data['residual sugar'], bins=bin_edges2)
plt.xlabel('residual sugar')
plt.ylabel('count')
plt.show()
```



In [26]:
```
fig3 = plt.scatter(wine_data['residual sugar'], wine_data['pH'])
plt.xlabel('residual sugar')
plt.ylabel('pH')
plt.show()
```



In [27]:
```
wine_data.columns
```

Out[27]:
```
Index(['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'],
      dtype='object')
```

Pair Plot

In [28]:
```
fig4 = sns.pairplot(data=df[['alcohol', 'pH', 'residual sugar', 'quality']],
                    hue='quality')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[28], line 1
----> 1 fig4 = sns.pairplot(data=df[['alcohol', 'pH', 'residual sugar', 'qualit
y']],
      2                     hue='quality')
      3 plt.show()

NameError: name 'df' is not defined
```

In [29]:
```
label_1 = LabelEncoder()
wine_data['type']=label_1.fit_transform(wine_data['type'])
wine_data
```

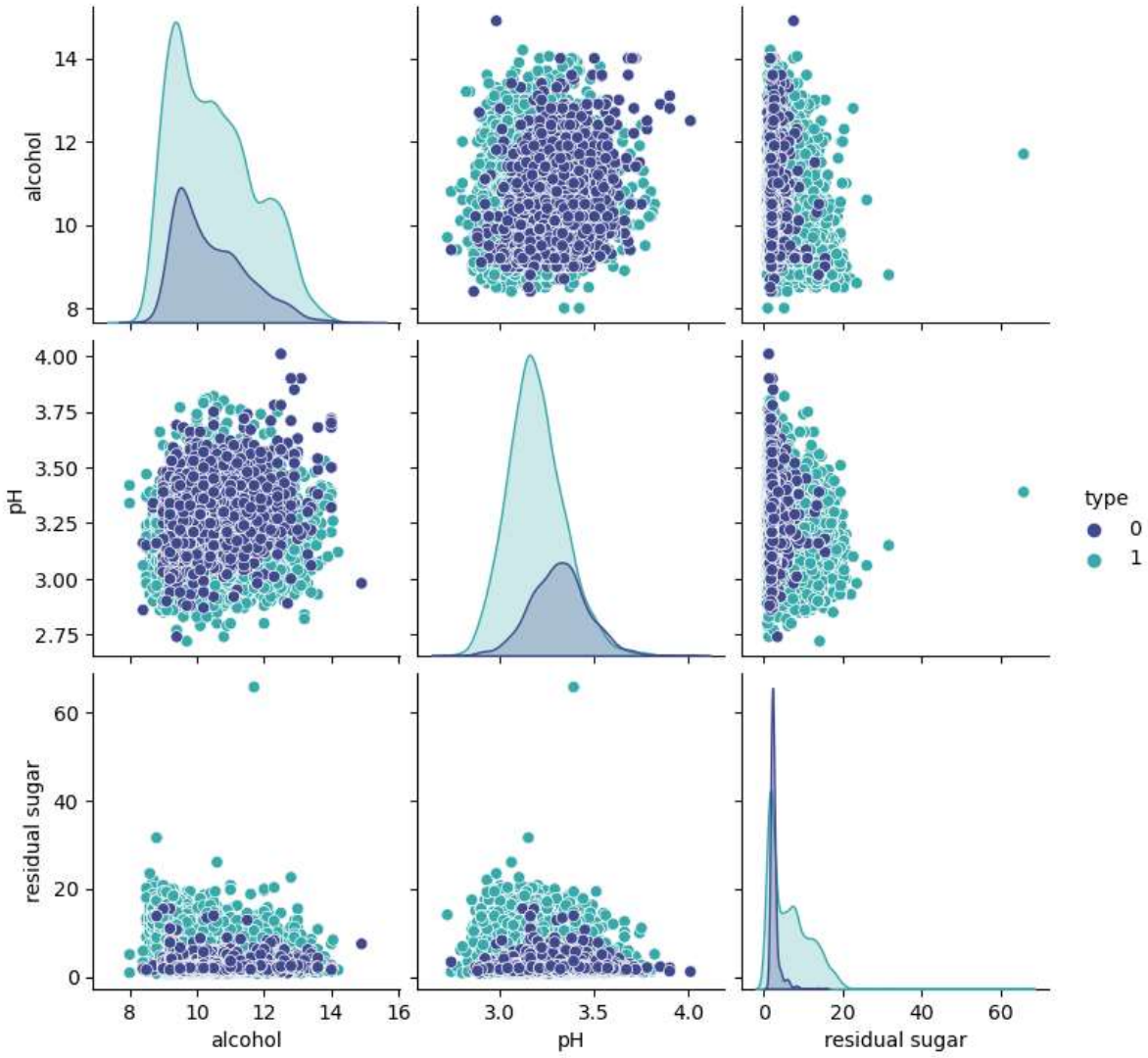
Out[29]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	1	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45
1	1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49
2	1	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44
3	1	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40
4	1	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40
...	...	...	...	...	...	...	...	...	...	...	...
6492	0	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58
6493	0	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76
6494	0	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
6495	0	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
6496	0	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

6497 rows × 13 columns

In [30]:
```
fig5 = sns.pairplot(data=wine_data[['alcohol', 'pH', 'residual sugar', 'type']],
                    hue='type', palette='mako')
for ax in fig5.axes.flat:
    ax.set_xlabel(ax.get_xlabel(), fontsize=10)
    ax.set_ylabel(ax.get_ylabel(), fontsize=10)
plt.show()
```

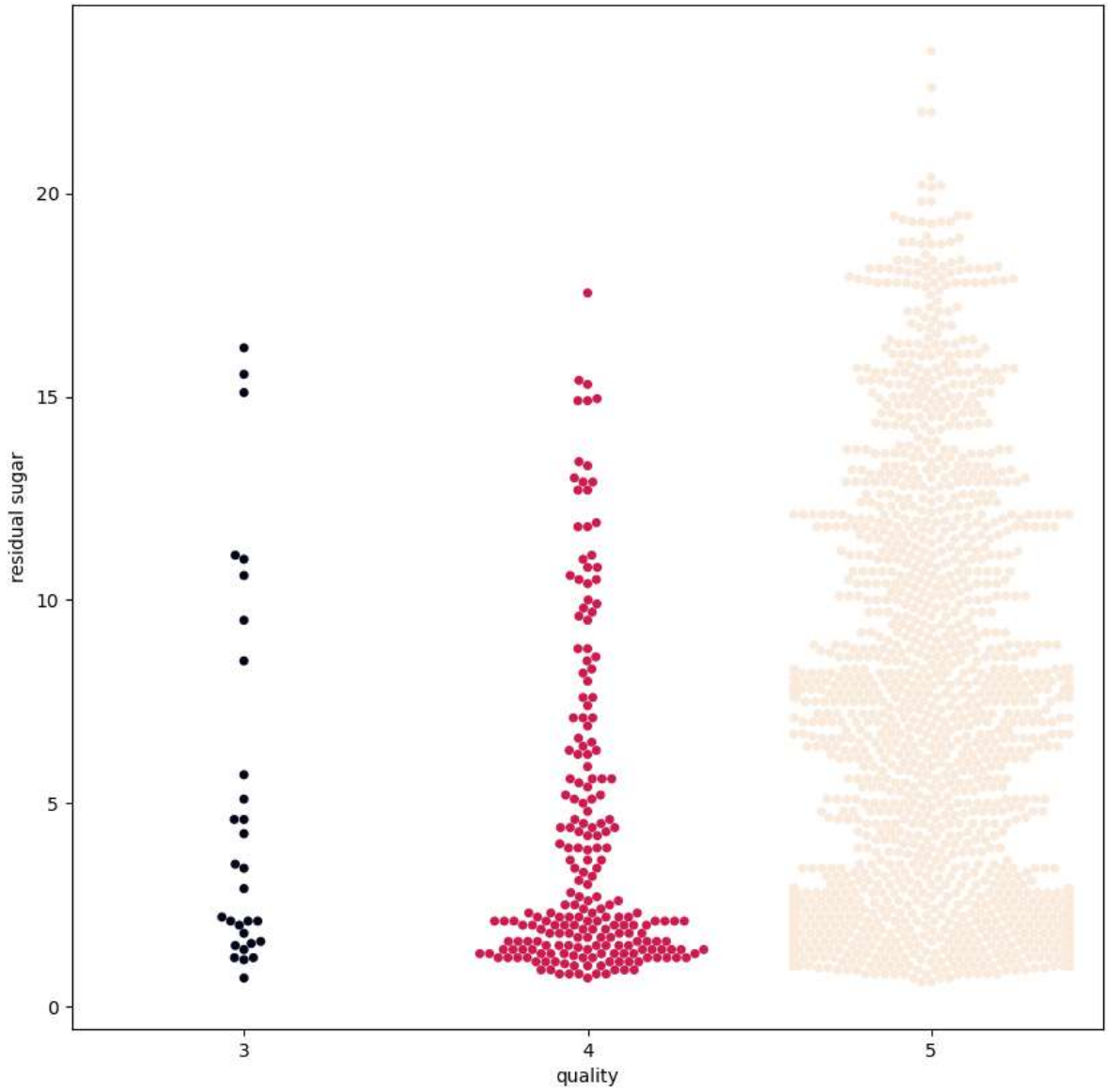
```
D:\Anaconda\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure lay
out has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Bee-Swarm

In [31]:
```
plt.figure(figsize=(10,10))
sns.swarmplot(x='quality', y='residual sugar', hue='quality',
              data=wine_data[wine_data['quality'] < 6], palette='rocket', legend=F
plt.show()
```

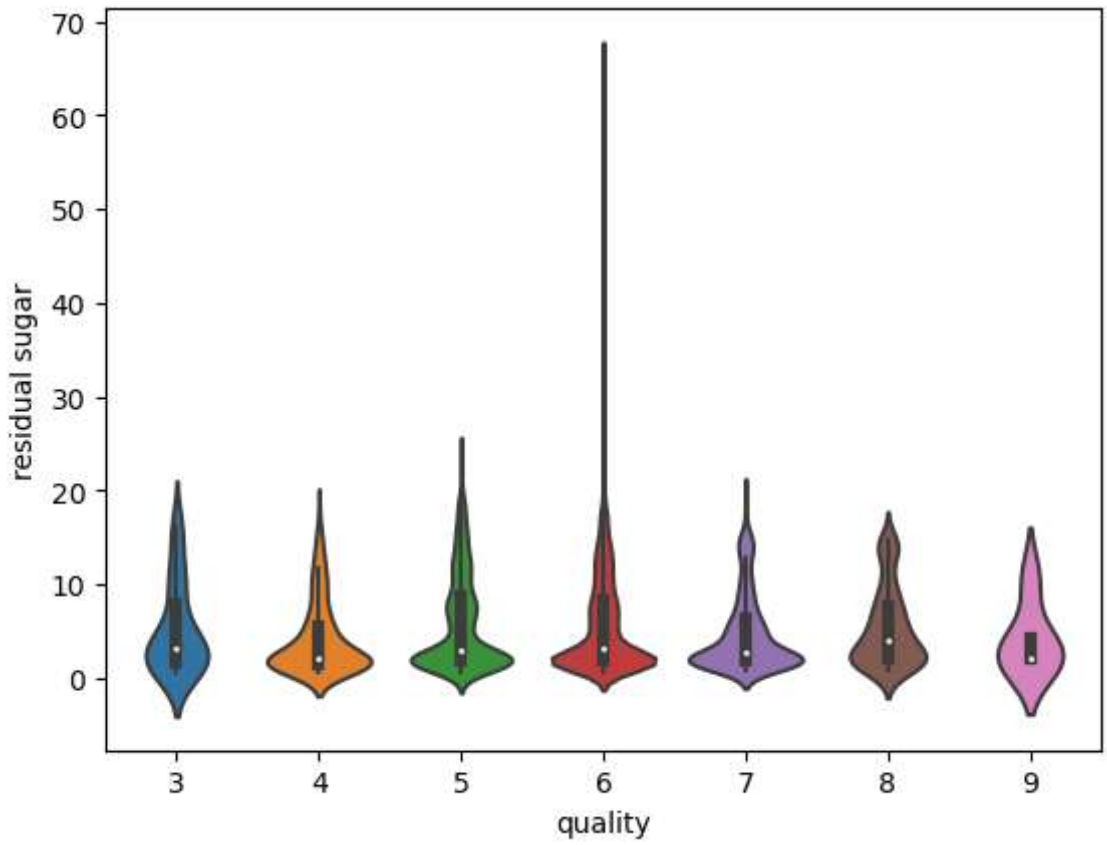
```
D:\Anaconda\Lib\site-packages\seaborn\categorical.py:3544: UserWarning: 31.6% of t
he points cannot be placed; you may want to decrease the size of the markers or us
e stripplot.
  warnings.warn(msg, UserWarning)
D:\Anaconda\Lib\site-packages\seaborn\categorical.py:3544: UserWarning: 37.4% of t
he points cannot be placed; you may want to decrease the size of the markers or us
e stripplot.
  warnings.warn(msg, UserWarning)
```



Violin-Plot

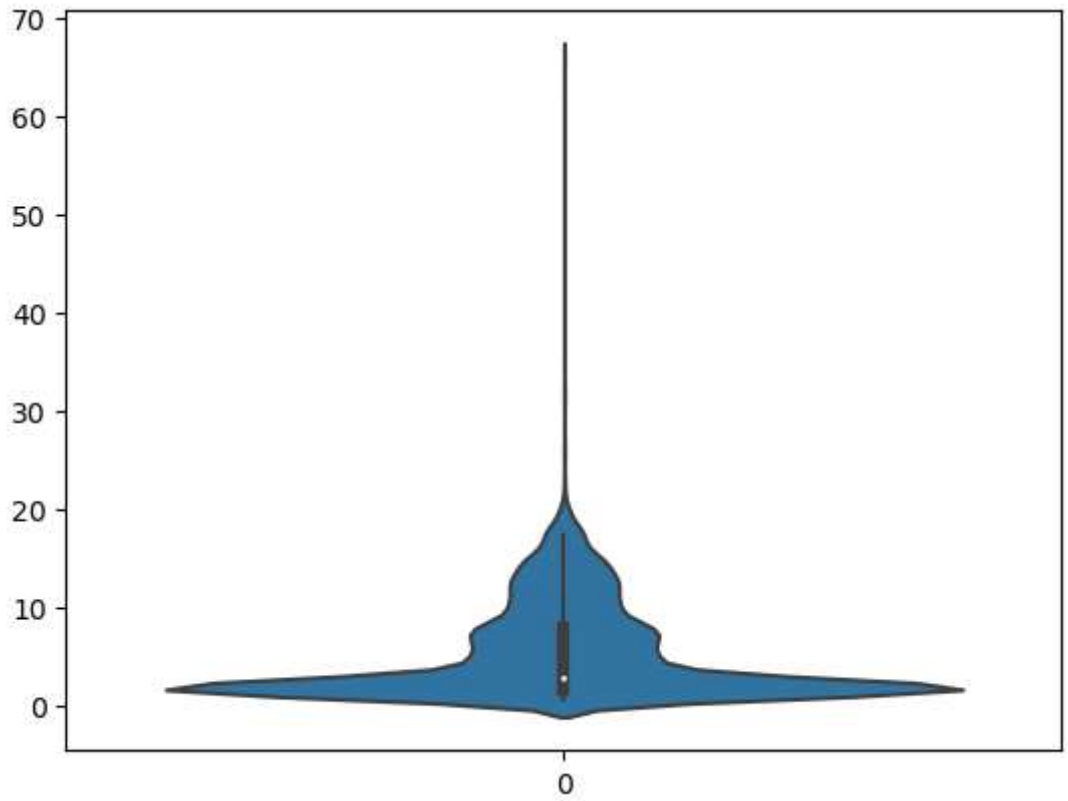
In [32]:
```
sns.violinplot(x=wine_data['quality'], y=wine_data['residual sugar'])
```

Out[32]:
```
<Axes: xlabel='quality', ylabel='residual sugar'>
```



In [33]:
```
sns.violinplot(wine_data['residual sugar'])
```

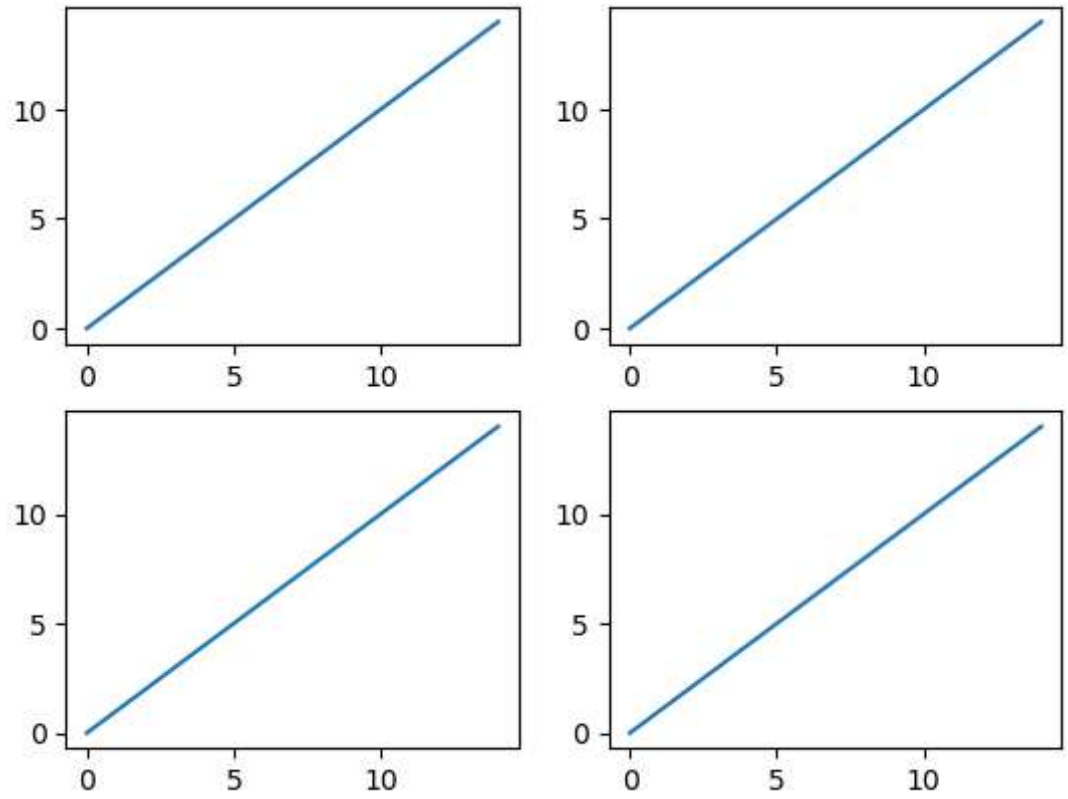
Out[33]: <Axes: >



Sub-Plot

Multiple plots in a single figure.

In [34]:
```
x = range(15)
y = range(15)
fig, ax = plt.subplots(nrows=2, ncols=2)
for row in ax:
    for col in row:
        col.plot(x, y)
plt.show()
```



In [ ]:

```
import pandas as pd
df = pd.read_csv('/content/Iris (2).csv')
df
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
df.shape
```

```
(150, 6)
```

```
#Transforming the class labels into integer representation
d = {
    'Iris-setosa' : 0,
    'Iris-versicolor': 1,
    'Iris-virginica' : 2
  }
df['Species']=df['Species'].map(d)
df['Species']
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

```
import numpy as np
np.unique(df['Species'])
```

```
array([0, 1, 2])
```

```
#extracting a column
y=df['Species']
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
```

```
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

```
print(type(y))
#Series-->It is a one-dimensional array holding data of any type.
```

```
<class 'pandas.core.series.Series'>
```

```
#y=y.values
y=df['Species'].values
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
print(type(y))
# the Series.values attribute returns an ndarray object containing the values of the given Series object.
```

```
<class 'numpy.ndarray'>
```

```
#to print all the columns in the dataframe
df.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
x = df.iloc[:,1:5]
x
#iloc--> index location
#x contains all the rows and their corresponding columns 1 to 4
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
print(type(x))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
x=x.values
```

```
print(type(x))
```

```
<class 'numpy.ndarray'>
```

```
x.shape
```

```
(150, 4)
```

```
x[:5]
#prints the rows 0 to 4 in x
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

The apply method offers a convenient way to manipulate pandas DataFrame entries along the column axis. We can use a regular Python or lambda function as input to the apply method. In this context, assume that our goal is to transform class labels from a string representation (e.g., "Iris-Setosa") to an integer representation (e.g., 0), which is a historical convention and a recommendation for compatibility with various machine learning tools.

```
df1=pd.read_csv('/content/Iris (2).csv')
df1['Species'] = df1['Species'].apply(lambda x: 0 if  x=='Iris-setosa' else 1\
                                  if x=='Iris-versicolor' else x)
df1
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Splitting the data into Test , Validation and Train Subsets

```
#getting all the indices of data
indices = np.arange(x.shape[0])   #x.shape[0] = 150
indices
#arange() --> returns an array with evenly spaced elements as per the interval.
```

```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149])
```

```
#randomising the indices
rng = np.random.RandomState(123)  #123 is the seed value
permuted_indices = rng.permutation(indices)
#A permutation is an arrangement of a set where order does matter.
permuted_indices
```

```
array([ 72, 112, 132,  88,  37, 138,  87,  42,   8,  90, 141,  33,  59,
       116, 135, 104,  36,  13,  63,  45,  28, 133,  24, 127,  46,  20,
        31, 121, 117,   4, 130, 119,  29,   0,  62,  93, 131,   5,  16,
        82,  60,  35, 143, 145, 142, 114, 136,  53,  19,  38, 110,  23,
```

```
             9,  86,  91,  89,  79, 101,  65, 115,  41, 124,  95,  21,  11,
           103,  74, 122, 118,  44,  51,  81, 149,  12, 129,  56,  50,  25,
           128, 146,  43,   1,  71,  54, 100,  14,   6,  80,  26,  70, 139,
            30, 108,  15,  18,  77,  22,  10,  58, 107,  75,  64,  69,   3,
            40,  76, 134,  34,  27,  94,  85,  97, 102,  52,  92,  99, 105,
             7,  48,  61, 120, 137, 125, 147,  39,  84,   2,  67,  55,  49,
            68, 140,  78, 144, 111,  32,  73,  47, 148, 113,  96,  57, 123,
           106,  83,  17,  98,  66, 126, 109])


#determining the size of train, validate and test subsets
train_size , valid_size = int(0.65*x.shape[0]) , int(0.15*x.shape[0])
test_size = x.shape[0] - (train_size + valid_size)
print(train_size , valid_size , test_size)
```

```
97 22 31
```

```
#assigning indices to each of the subsets
train_ind = permuted_indices[:train_size]
valid_ind = permuted_indices[train_size : train_size+valid_size]
test_ind = permuted_indices[train_size+valid_size : ]


train_ind
```

```
array([ 72, 112, 132,  88,  37, 138,  87,  42,   8,  90, 141,  33,  59,
       116, 135, 104,  36,  13,  63,  45,  28, 133,  24, 127,  46,  20,
        31, 121, 117,   4, 130, 119,  29,   0,  62,  93, 131,   5,  16,
        82,  60,  35, 143, 145, 142, 114, 136,  53,  19,  38, 110,  23,
         9,  86,  91,  89,  79, 101,  65, 115,  41, 124,  95,  21,  11,
       103,  74, 122, 118,  44,  51,  81, 149,  12, 129,  56,  50,  25,
       128, 146,  43,   1,  71,  54, 100,  14,   6,  80,  26,  70, 139,
        30, 108,  15,  18,  77,  22])
```

```
valid_ind
```

```
array([ 10,  58, 107,  75,  64,  69,   3,  40,  76, 134,  34,  27,  94,
        85,  97, 102,  52,  92,  99, 105,   7,  48])
```

```
test_ind
```

```
array([ 61, 120, 137, 125, 147,  39,  84,   2,  67,  55,  49,  68, 140,
        78, 144, 111,  32,  73,  47, 148, 113,  96,  57, 123, 106,  83,
        17,  98,  66, 126, 109])
```

```
#assigning instances to subsets
x_train , y_train = x[train_ind] , y[train_ind]
x_valid , y_valid = x[valid_ind] , y[valid_ind]
x_test , y_test = x[test_ind] , y[test_ind]
#x,y should be converted to ndarray before performing this step


print( x_train.shape , y_train.shape)
print( x_valid.shape , y_valid.shape)
print( x_test.shape , y_test.shape)
```

```
(97, 4) (97,)
(22, 4) (22,)
(31, 4) (31,)
```

```
#Splitting the data into subsets
from sklearn.model_selection import train_test_split
x_train , x_test, y_train , y_test = \
  train_test_split (x,y,test_size=0.2,shuffle=True,random_state=123,stratify=y)
```

Preprocessing Steps

Data in the real-world are rarely clean and homogeneous. Data can either be missing during data extraction or collection due to several reasons. Missing values need to be handled carefully because they reduce the quality of any of our performance matrix. It can also lead to wrong prediction or classification and can also cause a high bias for any given model being used. There are several options for handling missing values. However, the choice of what should be done is largely dependent on the nature of our data and the missing values. Below are some of the techniques:

```
import pandas as pd
table1 = pd.read_excel('/IBM-313 Marks - mis.xlsx')
table1.head(5)
```

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	NaN	20	NaN	12.0	NaN	NaN
1	2	11.05	20	31.05	26.0	57.05	NaN
2	3	NaN	20	NaN	14.0	NaN	NaN
3	4	6.00	10	16.00	13.0	29.00	NaN
4	5	11.35	20	31.35	17.0	48.35	NaN

Next steps: Generate code with `table1` View recommended plots

```
# to fill all 'NaN' in 'total_internal' column with 5
table1['total_internal'].fillna(5, inplace=False)
# The fillna() method replaces the NULL values with a specified value.
```

```
0      5.00
1     31.05
2      5.00
3     16.00
4     31.35
      ...
74    22.05
75    22.25
76    11.75
77    13.00
78    15.80
Name: total_internal, Length: 79, dtype: float64
```

```
table1.head(5)
# the table1 didnot change
```

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	NaN	20	NaN	12.0	NaN	NaN
1	2	11.05	20	31.05	26.0	57.05	NaN
2	3	NaN	20	NaN	14.0	NaN	NaN
3	4	6.00	10	16.00	13.0	29.00	NaN
4	5	11.35	20	31.35	17.0	48.35	NaN

```
# how many 'NaN' are present in each column
table1.isna().sum()
#Pandas dataframe.isna() function is used to detect missing values.
```

```
S.No.              0
midexam            2
miniproject        0
total_internal     2
endexam            2
total              2
Grace marks       79
dtype: int64
```

```
#output in the form of boolean
table1['midexam'].isna()
```

```
0      True
1     False
2      True
3     False
4     False
      ...
74    False
75    False
76    False
77    False
78    False
Name: midexam, Length: 79, dtype: bool
```

```
table1['midexam'].fillna(13,inplace=True)
table1.head(5)
#here changes are made in the original data also
```

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	13.00	20	NaN	12.0	NaN	NaN
1	2	11.05	20	31.05	26.0	57.05	NaN
2	3	13.00	20	NaN	14.0	NaN	NaN
3	4	6.00	10	16.00	13.0	29.00	NaN
4	5	11.35	20	31.35	17.0	48.35	NaN

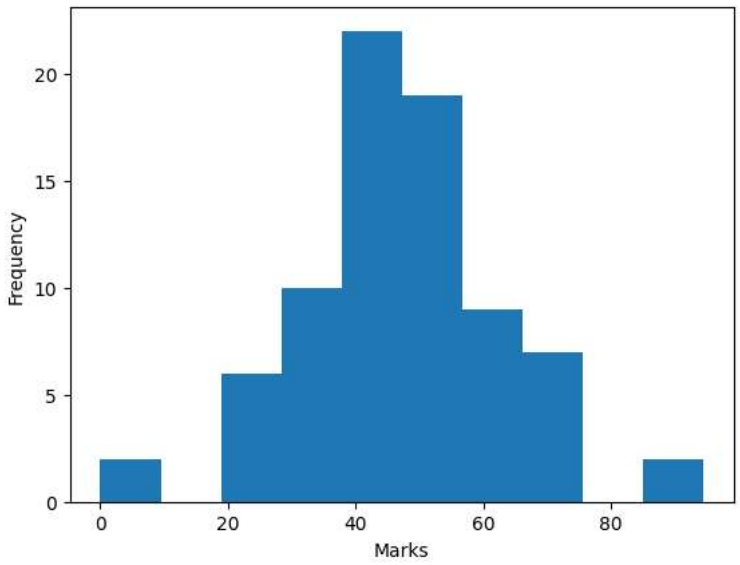
```
#filling all the 'NaN' with 2
table1.fillna(2, inplace=False)
```

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	13.00	20	2.00	12.0	2.00	2.0
1	2	11.05	20	31.05	26.0	57.05	2.0
2	3	13.00	20	2.00	14.0	2.00	2.0
3	4	6.00	10	16.00	13.0	29.00	2.0
4	5	11.35	20	31.35	17.0	48.35	2.0
...	...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05	2.0
75	76	12.25	10	22.25	28.0	50.25	2.0
76	77	1.75	10	11.75	2.0	0.00	2.0
77	78	3.00	10	13.00	2.0	0.00	2.0
78	79	5.80	10	15.80	12.0	27.80	2.0

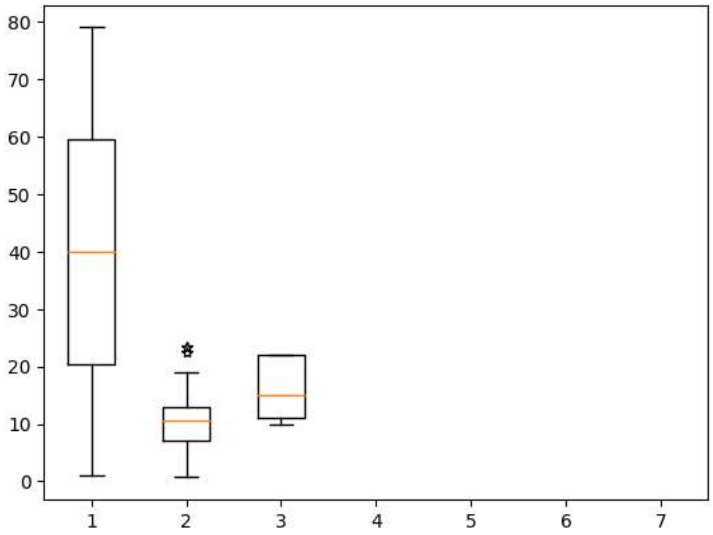
79 rows × 7 columns

```
#plot histogram for the 'total' column
import matplotlib.pyplot as plt
plt.hist(table1['total'])
plt.xlabel('Marks')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```




```
#boxplot for all columns
from matplotlib import pyplot as plt
plt.boxplot(table1,sym='*')
plt.show()
```




```
#'grace marks' column has all 'NaN' , so drop it
table1.drop(['Grace marks'], axis=1, inplace=True)
#axis=1 implies column
table1.head(5)
```

	S.No.	midexam	miniproject	total_internal	endexam	total
0	1	13.00	20	NaN	12.0	NaN
1	2	11.05	20	31.05	26.0	57.05
2	3	13.00	20	NaN	14.0	NaN
3	4	6.00	10	16.00	13.0	29.00
4	5	11.35	20	31.35	17.0	48.35

Next steps: Generate code with `table1` View recommended plots

```
#drop a paricular row, for that use axis=0
table1.drop([0],axis=0,inplace=True)
#this drops the 1st row
table1
```

	S.No.	midexam	miniproject	total_internal	endexam	total
1	2	11.05	20	31.05	26.0	57.05
2	3	13.00	20	NaN	14.0	NaN
3	4	6.00	10	16.00	13.0	29.00
4	5	11.35	20	31.35	17.0	48.35
5	6	11.00	20	31.00	24.0	55.00
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05
75	76	12.25	10	22.25	28.0	50.25
76	77	1.75	10	11.75	NaN	0.00
77	78	3.00	10	13.00	NaN	0.00
78	79	5.80	10	15.80	12.0	27.80

78 rows × 6 columns

Next steps: Generate code with `table1` View recommended plots

```
#get description of dataset
table1.describe()
```

	S.No.	midexam	miniproject	total_internal	endexam	total
count	78.000000	78.000000	78.000000	77.000000	76.000000	77.000000
mean	40.500000	10.307692	16.512821	26.740260	21.134868	47.097403
std	22.660538	4.962110	4.916824	8.612434	8.077277	16.477932
min	2.000000	0.700000	10.000000	11.200000	7.000000	0.000000
25%	21.250000	7.000000	10.500000	19.500000	17.000000	38.000000
50%	40.500000	10.600000	15.000000	27.500000	20.000000	45.500000
75%	59.750000	13.037500	22.000000	33.500000	24.000000	55.400000
max	79.000000	23.500000	22.000000	45.500000	50.000000	94.500000

```
# fill the 'NaN' in 'endexam' by mean
table1['endexam'].fillna(table1['endexam'].mean(), inplace=True)
table1
```

	S.No.	midexam	miniproject	total_internal	endexam	total
1	2	11.05	20	31.05	26.000000	57.05
2	3	13.00	20	NaN	14.000000	NaN
3	4	6.00	10	16.00	13.000000	29.00
4	5	11.35	20	31.35	17.000000	48.35
5	6	11.00	20	31.00	24.000000	55.00
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.000000	42.05
75	76	12.25	10	22.25	28.000000	50.25
76	77	1.75	10	11.75	21.134868	0.00
77	78	3.00	10	13.00	21.134868	0.00
78	79	5.80	10	15.80	12.000000	27.80

78 rows × 6 columns

```
#drop rows with missing values
table1.dropna(axis=0, inplace=False)
```

	S.No.	midexam	miniproject	total_internal	endexam	total
1	2	11.05	20	31.05	26.000000	57.05
3	4	6.00	10	16.00	13.000000	29.00
4	5	11.35	20	31.35	17.000000	48.35
5	6	11.00	20	31.00	24.000000	55.00
6	7	10.50	15	25.50	18.000000	43.50
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.000000	42.05
75	76	12.25	10	22.25	28.000000	50.25
76	77	1.75	10	11.75	21.134868	0.00
77	78	3.00	10	13.00	21.134868	0.00
78	79	5.80	10	15.80	12.000000	27.80

77 rows × 6 columns

```
#drop columns with missing value
table1.dropna(axis=1, inplace=False)
```

	S.No.	midexam	miniproject	endexam
1	2	11.05	20	26.000000
2	3	13.00	20	14.000000
3	4	6.00	10	13.000000
4	5	11.35	20	17.000000
5	6	11.00	20	24.000000
...	...	...	...	...
74	75	12.05	10	20.000000
75	76	12.25	10	28.000000
76	77	1.75	10	21.134868
77	78	3.00	10	21.134868
78	79	5.80	10	12.000000

78 rows × 4 columns

```
from sklearn.impute import SimpleImputer
import numpy as np
# SimpleImputer is a scikit-learn class which is helpful in handling the
# missing data in the predictive model dataset. It replaces the NaN values with
# a specified placeholder.
imputer=SimpleImputer(missing_values=np.nan, strategy='mean')
# fit_transform() preprocess the data model training
x= imputer.fit_transform(table1)
x[:5]
```

```
array([[ 2.        , 11.05      , 20.        , 31.05      , 26.        ,
        57.05      ],
       [ 3.        , 13.        , 20.        , 26.74025974, 14.        ,
        47.0974026 ],
       [ 4.        ,  6.        , 10.        , 16.        , 13.        ,
        29.        ],
       [ 5.        , 11.35      , 20.        , 31.35      , 17.        ,
        48.35      ],
       [ 6.        , 11.        , 20.        , 31.        , 24.        ,
        55.        ]])
```

```
# reading a file
df=pd.read_csv('/data encoding.csv')
df.head(5)
```

	Roll_no	gender	marks	placed
0	100	M	55	y
1	200	F	67	y
2	300	M	67	y
3	400	F	12	n
4	500	M	90	y

```
#Categorical Data --> Nominal Data
from sklearn.preprocessing import LabelEncoder
label_1=LabelEncoder()
df['gender']= label_1.fit_transform(df['gender'])
df
```

	Roll_no	gender	marks	placed
0	100	1	55	y
1	200	0	67	y
2	300	1	67	y
3	400	0	12	n
4	500	1	90	y
5	600	0	56	n
6	700	1	90	y

```
df['placed']=label_1.fit_transform(df['placed'])
df
```

	Roll_no	gender	marks	placed
0	100	1	55	1
1	200	0	67	1
2	300	1	67	1
3	400	0	12	0
4	500	1	90	1
5	600	0	56	0
6	700	1	90	1

```
df=pd.read_csv('/data encoding.csv')
```

```
# Normalization of data using MinMaxScaler
# MinMax Scaler shrinks the data within the given range, usually of 0 to 1.
# minimum of feature is made equal to 0 and the maximum of feature equal to 1.
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler()
df[["Roll_no", "marks"]]=scaling.fit_transform \
(df[["Roll_no", "marks"]])
df[["Roll_no", "marks"]]
```

	Roll_no	marks
0	0.000000	0.551282
1	0.166667	0.705128
2	0.333333	0.705128
3	0.500000	0.000000
4	0.666667	1.000000
5	0.833333	0.564103
6	1.000000	1.000000

```
df=pd.read_csv('/data encoding.csv')
```

```
# Standardization of data
# StandardScaler standardizes features by subtracting the mean value from the
# feature and then dividing the result by feature standard deviation.
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaled_data=scaler.fit_transform(df[["Roll_no","marks"]])
scaled_data
```

```
array([[-1.5       , -0.30320829],
       [-1.        ,  0.18658971],
       [-0.5       ,  0.18658971],
       [ 0.        , -2.05831779],
       [ 0.5       ,  1.12536921],
       [ 1.        , -0.26239179],
       [ 1.5       ,  1.12536921]])
```

```
df.head(2)
```

	Roll_no	gender	marks	placed
0	100	M	55	y
1	200	F	67	y

Next steps: Generate code with df | View recommended plots

One-hot Encoding for Categorical (Nominal) Features One hot encoding is a technique that we use to represent categorical variables as numerical values in a machine learning model.

```
pd.get_dummies(df)
```

	Roll_no	marks	gender_F	gender_M	placed_n	placed_y
0	100	55	0	1	0	1
1	200	67	1	0	0	1
2	300	67	0	1	0	1
3	400	12	1	0	1	0
4	500	90	0	1	0	1
5	600	56	1	0	1	0