Importing the necesssary libraries

```
import pandas as pd     #'pandas' refrenced as 'pd'
import numpy as np      #'numpy' refrenced as 'np'
```

Creating data for using pandas libraries

```
# Creating Data frame using dictionary

data = {"Name":['Rishi','Harshit','kartik','yash'],
       "Subject":['physics','chemistry' , 'physics','computer
science'],
       "college":['poornima','biet','lord','jecrc']}

data
```

```
{'Name': ['Rishi', 'Harshit', 'kartik', 'yash'],
 'Subject': ['physics', 'chemistry', 'physics', 'computer science'],
 'college': ['poornima', 'biet', 'lord', 'jecrc']}
```

Using Pandas library to read the data

```
df = pd.DataFrame(data)       #calling 'pd' for reading the data
df
```

```
      Name           Subject   college
0    Rishi           physics  poornima
1  Harshit         chemistry      biet
2   kartik           physics      lord
3     yash  computer science     jecrc
```

```
type(df)       #  determine the type of an object in 'df'
```

```
pandas.core.frame.DataFrame
```

```
df['Name']     #return a Series containing all the values in the
'Name' column.
```

```
0      Rishi
1    Harshit
2     kartik
3       yash
Name: Name, dtype: object
```

```
type(df['Name'])     # determine the type of an series in a dataframe
```

```
pandas.core.series.Series
```

Accessing external data using Pandas library

```
df = pd.read_csv(r"E:\used bike\Used_Bikes.csv")      #Accessing a
external dataset using python
df.head()      # Display top 5 records of data
```

```
                                bike_name     price       city
kms_driven  \
0    TVS Star City Plus Dual Tone 110cc   35000.0  Ahmedabad
17654.0
1           Royal Enfield Classic 350cc  119900.0      Delhi
11000.0
2                  Triumph Daytona 675R  600000.0      Delhi
110.0
3                  TVS Apache RTR 180cc   65000.0  Bangalore
16329.0
4  Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0  Bangalore
10000.0

         owner  age  power          brand
0  First Owner  3.0  110.0            TVS
1  First Owner  4.0  350.0  Royal Enfield
2  First Owner  8.0  675.0        Triumph
3  First Owner  4.0  180.0            TVS
4  First Owner  3.0  150.0         Yamaha
```

```
df.info()    ## we want to take information from list or table
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32648 entries, 0 to 32647
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   bike_name   32648 non-null  object
 1   price       32648 non-null  float64
 2   city        32648 non-null  object
 3   kms_driven  32648 non-null  float64
 4   owner       32648 non-null  object
 5   age         32648 non-null  float64
 6   power       32648 non-null  float64
 7   brand       32648 non-null  object
dtypes: float64(4), object(4)
memory usage: 2.0+ MB
```

```
df.shape     #'shape' function represents the dimensionality of the
DataFrame.
```

```
(32648, 8)
```

```
# data types in data frame

df.dtypes
```

```
bike_name      object
price         float64
city           object
kms_driven    float64
owner          object
age           float64
power         float64
brand          object
dtype: object
```

Retreating data from the dataframe

```
df.head()    # By default, it returns the first 5 rows.
```

```
                                bike_name     price       city
kms_driven  \
0    TVS Star City Plus Dual Tone 110cc   35000.0  Ahmedabad
17654.0
1           Royal Enfield Classic 350cc  119900.0      Delhi
11000.0
2                  Triumph Daytona 675R  600000.0      Delhi
110.0
3                  TVS Apache RTR 180cc   65000.0  Bangalore
16329.0
4  Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0  Bangalore
10000.0

         owner  age  power           brand
0  First Owner  3.0  110.0             TVS
1  First Owner  4.0  350.0  Royal Enfield
2  First Owner  8.0  675.0         Triumph
3  First Owner  4.0  180.0             TVS
4  First Owner  3.0  150.0          Yamaha
```

```
df.tail()    # By default, it returns the last 5 rows.
```

```
                        bike_name    price    city  kms_driven
owner  \
32643     Hero Passion Pro 100cc  39000.0   Delhi     22000.0  First
Owner
32644       TVS Apache RTR 180cc  30000.0  Karnal      6639.0  First
Owner
32645   Bajaj Avenger Street 220  60000.0   Delhi     20373.0  First
Owner
32646  Hero Super Splendor 125cc  15600.0  Jaipur     84186.0  First
```

```
Owner
32647         Bajaj Pulsar 150cc  22000.0    Pune     60857.0  First
Owner

        age  power  brand
32643   4.0  100.0   Hero
32644   9.0  180.0    TVS
32645   6.0  220.0  Bajaj
32646  16.0  125.0   Hero
32647  13.0  150.0  Bajaj
```

```
df.head(10)      # returns the 'n' number of rows given in bracket.
```

```
                                  bike_name     price       city
kms_driven  \
0        TVS Star City Plus Dual Tone 110cc   35000.0  Ahmedabad
17654.0
1               Royal Enfield Classic 350cc  119900.0      Delhi
11000.0
2                     Triumph Daytona 675R  600000.0      Delhi
110.0
3                     TVS Apache RTR 180cc   65000.0  Bangalore
16329.0
4      Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0  Bangalore
10000.0
5                         Yamaha FZs 150cc   53499.0      Delhi
25000.0
6             Honda CB Hornet 160R  ABS DLX   85000.0      Delhi
8200.0
7       Hero Splendor Plus Self Alloy 100cc   45000.0      Delhi
12645.0
8         Royal Enfield Thunderbird X 350cc  145000.0  Bangalore
9190.0
9  Royal Enfield Classic Desert Storm 500cc   88000.0      Delhi
19000.0

          owner  age  power          brand
0   First Owner  3.0  110.0            TVS
1   First Owner  4.0  350.0  Royal Enfield
2   First Owner  8.0  675.0        Triumph
3   First Owner  4.0  180.0            TVS
4   First Owner  3.0  150.0         Yamaha
5   First Owner  6.0  150.0         Yamaha
6   First Owner  3.0  160.0          Honda
7   First Owner  3.0  100.0           Hero
8   First Owner  3.0  350.0  Royal Enfield
9  Second Owner  7.0  500.0  Royal Enfield
```

```
df['city']    ## data type is series
```

```
0        Ahmedabad
1            Delhi
2            Delhi
3        Bangalore
4        Bangalore
           ...
32643        Delhi
32644       Karnal
32645        Delhi
32646       Jaipur
32647         Pune
Name: city, Length: 32648, dtype: object
```

```
df[['brand','age','price']]    #retreving the given series from the dataframe
```

```
                brand   age     price
0                 TVS   3.0   35000.0
1      Royal Enfield   4.0  119900.0
2             Triumph   8.0  600000.0
3                 TVS   4.0   65000.0
4              Yamaha   3.0   80000.0
...               ...   ...       ...
32643           Hero   4.0   39000.0
32644             TVS   9.0   30000.0
32645          Bajaj   6.0   60000.0
32646           Hero  16.0   15600.0
32647          Bajaj  13.0   22000.0

[32648 rows x 3 columns]
```

```
df[['bike_name','price']]      # Used to call acc. to the need of task we want to perform
```

```
                                    bike_name     price
0        TVS Star City Plus Dual Tone 110cc   35000.0
1               Royal Enfield Classic 350cc  119900.0
2                     Triumph Daytona 675R  600000.0
3                     TVS Apache RTR 180cc   65000.0
4      Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0
...                                     ...       ...
32643               Hero Passion Pro 100cc   39000.0
32644                 TVS Apache RTR 180cc   30000.0
32645             Bajaj Avenger Street 220   60000.0
32646            Hero Super Splendor 125cc   15600.0
32647                   Bajaj Pulsar 150cc   22000.0

[32648 rows x 2 columns]
```

Loc and Iloc

The loc and iloc functions in pandas are used for accessing data in a DataFrame by labels and integer-location-based indexing, respectively.

```
## loc and iloc

df.loc[0:100]         # loc is print till ending point of input
# means the value will be +1 of given data
```

```
                                bike_name     price           city  \
0      TVS Star City Plus Dual Tone 110cc   35000.0      Ahmedabad
1              Royal Enfield Classic 350cc  119900.0          Delhi
2                    Triumph Daytona 675R  600000.0          Delhi
3                    TVS Apache RTR 180cc   65000.0      Bangalore
4    Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0      Bangalore
..                                     ...       ...            ...
96             Suzuki Gixxer Fi 150cc ABS   96700.0       Vadodara
97               Hero Splendor plus 100cc   31900.0          Delhi
98                     Bajaj Pulsar 150cc   35000.0  Visakhapatnam
99                         KTM Duke 390cc  240000.0      Bangalore
100                        Honda CBR 250R   68000.0          Delhi

     kms_driven        owner   age  power           brand
0       17654.0  First Owner   3.0  110.0             TVS
1       11000.0  First Owner   4.0  350.0  Royal Enfield
2         110.0  First Owner   8.0  675.0        Triumph
3       16329.0  First Owner   4.0  180.0             TVS
4       10000.0  First Owner   3.0  150.0          Yamaha
..          ...          ...   ...    ...             ...
96       5100.0  First Owner   1.0  150.0         Suzuki
97       9000.0  First Owner   5.0  100.0           Hero
98      44480.0  First Owner  13.0  150.0          Bajaj
99      11000.0  First Owner   3.0  390.0            KTM
100     16000.0  First Owner   7.0  250.0          Honda

[101 rows x 8 columns]
```

```
df.loc[0:100,['bike_name','age','price']]
#retriving loc from multiple series
```

```
                                bike_name   age     price
0      TVS Star City Plus Dual Tone 110cc   3.0   35000.0
1              Royal Enfield Classic 350cc   4.0  119900.0
2                    Triumph Daytona 675R   8.0  600000.0
3                    TVS Apache RTR 180cc   4.0   65000.0
4    Yamaha FZ S V 2.0 150cc-Ltd. Edition   3.0   80000.0
..                                     ...   ...       ...
96             Suzuki Gixxer Fi 150cc ABS   1.0   96700.0
97               Hero Splendor plus 100cc   5.0   31900.0
```

```
98                      Bajaj Pulsar 150cc  13.0   35000.0
99                          KTM Duke 390cc   3.0  240000.0
100                         Honda CBR 250R   7.0   68000.0

[101 rows x 3 columns]
```

```
df.iloc[0:100,0:3]           #iloc = loc is print till n-1 point of
input
```

```
                                bike_name     price           city
0     TVS Star City Plus Dual Tone 110cc   35000.0      Ahmedabad
1            Royal Enfield Classic 350cc  119900.0          Delhi
2                  Triumph Daytona 675R  600000.0          Delhi
3                  TVS Apache RTR 180cc   65000.0      Bangalore
4   Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0      Bangalore
..                                   ...       ...            ...
95                   Bajaj Pulsar 150cc   60000.0           Pune
96            Suzuki Gixxer Fi 150cc ABS   96700.0       Vadodara
97              Hero Splendor plus 100cc   31900.0          Delhi
98                   Bajaj Pulsar 150cc   35000.0  Visakhapatnam
99                       KTM Duke 390cc  240000.0      Bangalore

[100 rows x 3 columns]
```

```
df.loc[0:10]   # We wanted a 10 data values but it will give a 11 data
values
```

```
                                    bike_name     price       city
kms_driven  \
0         TVS Star City Plus Dual Tone 110cc   35000.0  Ahmedabad
17654.0
1                Royal Enfield Classic 350cc  119900.0      Delhi
11000.0
2                      Triumph Daytona 675R  600000.0      Delhi
110.0
3                      TVS Apache RTR 180cc   65000.0  Bangalore
16329.0
4       Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0  Bangalore
10000.0
5                          Yamaha FZs 150cc   53499.0      Delhi
25000.0
6              Honda CB Hornet 160R  ABS DLX   85000.0      Delhi
8200.0
7        Hero Splendor Plus Self Alloy 100cc   45000.0      Delhi
12645.0
8          Royal Enfield Thunderbird X 350cc  145000.0  Bangalore
9190.0
9   Royal Enfield Classic Desert Storm 500cc   88000.0      Delhi
19000.0
10                  Yamaha YZF-R15 2.0 150cc   72000.0  Bangalore
```

```
20000.0
```

```
            owner  age  power           brand
0    First Owner  3.0  110.0             TVS
1    First Owner  4.0  350.0  Royal Enfield
2    First Owner  8.0  675.0         Triumph
3    First Owner  4.0  180.0             TVS
4    First Owner  3.0  150.0          Yamaha
5    First Owner  6.0  150.0          Yamaha
6    First Owner  3.0  160.0           Honda
7    First Owner  3.0  100.0            Hero
8    First Owner  3.0  350.0  Royal Enfield
9   Second Owner  7.0  500.0  Royal Enfield
10   First Owner  7.0  150.0          Yamaha
```

```
df.iloc[0:10]   #Data given as we required
```

```
                                      bike_name     price       city
kms_driven  \
0        TVS Star City Plus Dual Tone 110cc   35000.0  Ahmedabad
17654.0
1                Royal Enfield Classic 350cc  119900.0      Delhi
11000.0
2                      Triumph Daytona 675R  600000.0      Delhi
110.0
3                      TVS Apache RTR 180cc   65000.0  Bangalore
16329.0
4      Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0  Bangalore
10000.0
5                           Yamaha FZs 150cc   53499.0      Delhi
25000.0
6              Honda CB Hornet 160R  ABS DLX   85000.0      Delhi
8200.0
7       Hero Splendor Plus Self Alloy 100cc   45000.0      Delhi
12645.0
8         Royal Enfield Thunderbird X 350cc  145000.0  Bangalore
9190.0
9  Royal Enfield Classic Desert Storm 500cc   88000.0      Delhi
19000.0
```

```
            owner  age  power           brand
0    First Owner  3.0  110.0             TVS
1    First Owner  4.0  350.0  Royal Enfield
2    First Owner  8.0  675.0         Triumph
3    First Owner  4.0  180.0             TVS
4    First Owner  3.0  150.0          Yamaha
5    First Owner  6.0  150.0          Yamaha
6    First Owner  3.0  160.0           Honda
7    First Owner  3.0  100.0            Hero
```

```
8   First Owner  3.0  350.0  Royal Enfield
9  Second Owner  7.0  500.0  Royal Enfield
```

```
df.loc[500:560,['brand','power','age']]    #data can be retreaved from
any indexed value as required
```

```
                brand  power  age
500  Harley-Davidson  750.0  5.0
501            Bajaj  100.0  7.0
502            Bajaj  150.0  4.0
503     Royal Enfield  350.0  4.0
504            Bajaj  160.0  4.0
..               ...    ...  ...
556          Benelli  300.0  6.0
557  Harley-Davidson  750.0  7.0
558              TVS  160.0  5.0
559            Bajaj  150.0  5.0
560           Suzuki  150.0  3.0

[61 rows x 3 columns]
```

```
df.iloc[500:561,4:7]     #retriving rows and columns
```

```
            owner  age  power
500  Second Owner  5.0  750.0
501   First Owner  7.0  100.0
502   First Owner  4.0  150.0
503   First Owner  4.0  350.0
504   First Owner  4.0  160.0
..            ...  ...    ...
556   First Owner  6.0  300.0
557   First Owner  7.0  750.0
558   First Owner  5.0  160.0
559   First Owner  5.0  150.0
560   First Owner  3.0  150.0

[61 rows x 3 columns]
```

```
## data is filtering with conditions
```

```
df[df['price']<=50000]
```

```
                                    bike_name    price         city
kms_driven  \
0       TVS Star City Plus Dual Tone 110cc  35000.0    Ahmedabad
17654.0
7      Hero Splendor Plus Self Alloy 100cc  45000.0        Delhi
12645.0
13                     Bajaj Discover 100M  29499.0        Delhi
20000.0
14                     Bajaj Discover 125M  29900.0        Delhi
```

```
20000.0
17                  Suzuki Gixxer SF 150cc  48000.0       Mumbai
24725.0
...                                    ...      ...          ...
...
32642                    Hero Passion 100cc  15000.0  Perumbavoor
35000.0
32643                Hero Passion Pro 100cc  39000.0        Delhi
22000.0
32644                  TVS Apache RTR 180cc  30000.0       Karnal
6639.0
32646             Hero Super Splendor 125cc  15600.0       Jaipur
84186.0
32647                    Bajaj Pulsar 150cc  22000.0         Pune
60857.0

               owner   age  power   brand
0        First Owner   3.0  110.0     TVS
7        First Owner   3.0  100.0    Hero
13       First Owner   8.0  100.0   Bajaj
14       First Owner   7.0  125.0   Bajaj
17       First Owner   5.0  150.0  Suzuki
...              ...   ...    ...     ...
32642  Second Owner  19.0  100.0    Hero
32643   First Owner   4.0  100.0    Hero
32644   First Owner   9.0  180.0     TVS
32646   First Owner  16.0  125.0    Hero
32647   First Owner  13.0  150.0   Bajaj

[19114 rows x 8 columns]
```

```
df[df['age']<=2]    # data is filtering with conditions of age
```

```
                                  bike_name     price        city  \
22       Hero Splendor iSmart Plus IBS 110cc   46500.0       Delhi
29                    Honda X-Blade 160CC ABS   81200.0      Mettur
38         Royal Enfield Thunderbird X 500cc  190500.0  Samastipur
39                           KTM RC 200cc ABS  179000.0   Bangalore
43               Bajaj Avenger Street 220 ABS  110000.0   Bangalore
...                                       ...       ...         ...
8796                     Hero Xtreme 200R ABS   85000.0   Bangalore
8836   Royal Enfield Thunderbird X 350cc ABS  170200.0      Mumbai
8883  TVS Apache RTR 200 4V ABS Race Edition  120000.0      Mumbai
9187           Harley-Davidson Street 750 ABS  520000.0      Jaipur
9322                TVS Radeon 110cc Drum SBT   58000.0       Delhi

      kms_driven        owner  age  power            brand
22        3500.0  First Owner  2.0  110.0             Hero
29        9100.0  First Owner  2.0  160.0            Honda
38        4550.0  First Owner  2.0  500.0    Royal Enfield
```

```
39        3400.0  First Owner  2.0  200.0              KTM
43        2550.0  First Owner  2.0  220.0            Bajaj
...          ...          ...  ...    ...              ...
8796        40.0  First Owner  2.0  200.0             Hero
8836      1000.0  First Owner  2.0  350.0    Royal Enfield
8883      4442.0  First Owner  2.0  200.0              TVS
9187       399.0  First Owner  2.0  750.0  Harley-Davidson
9322      4020.0  First Owner  2.0  110.0              TVS

[395 rows x 8 columns]
```

```
# for count the values
df['brand'].value_counts()
```

```
brand
Bajaj              11213
Hero                6368
Royal Enfield       4178
Yamaha              3916
Honda               2108
Suzuki              1464
TVS                 1247
KTM                 1077
Harley-Davidson      737
Kawasaki              79
Hyosung               64
Benelli               56
Mahindra              55
Triumph               26
Ducati                22
BMW                   16
Jawa                  10
MV                     4
Indian                 3
Ideal                  2
Rajdoot                1
LML                    1
Yezdi                  1
Name: count, dtype: int64
```

```
df['city'].value_counts()    # for count the values of a series
```

```
city
Delhi            7318
Bangalore        2723
Mumbai           2591
Hyderabad        2160
Pune             1724
                 ...
Surendranagar       1
```

```
Khandela             1
Mohammadabad         1
Shimla               1
Sidhi                1
Name: count, Length: 443, dtype: int64
```

```
df['city'].value_counts().keys()        # this is use only for data
names
```

```
Index(['Delhi', 'Bangalore', 'Mumbai', 'Hyderabad', 'Pune', 'Chennai',
       'Lucknow', 'Jaipur', 'Ghaziabad', 'Ahmedabad',
       ...
       'Raigarh', 'Chikkaballapur', 'Kasba', 'Marandahalli', 'Badaun',
       'Surendranagar', 'Khandela', 'Mohammadabad', 'Shimla',
'Sidhi'],
      dtype='object', name='city', length=443)
```

```
df['city'].value_counts().values       # this is use for data numbers
or values
```

```
array([7318, 2723, 2591, 2160, 1724, 1619, 1294, 1007,  938,  905,
776,
        651,  649,  640,  635,  625,  621,  621,  617,  611,  609,
608,
        608,  100,   97,   94,   87,   75,   57,   52,   46,   36,
33,
         30,   29,   27,   27,   25,   23,   23,   22,   21,   19,
18,
         17,   16,   15,   14,   13,   13,   13,   12,   12,   12,
12,
         12,   11,   11,   11,   11,   10,   10,   10,   10,   10,
9,
          9,    9,    9,    8,    8,    8,    8,    8,    8,    7,
7,
          7,    7,    7,    7,    7,    7,    7,    7,    6,    6,
6,
          6,    6,    6,    6,    5,    5,    5,    5,    5,    5,
5,
          5,    5,    5,    5,    5,    5,    5,    5,    5,    5,
4,
          4,    4,    4,    4,    4,    4,    4,    4,    4,    4,
4,
          4,    4,    4,    4,    4,    4,    4,    4,    4,    4,
4,
          4,    4,    4,    4,    4,    4,    4,    4,    3,    3,
3,
          3,    3,    3,    3,    3,    3,    3,    3,    3,    3,
3,
          3,    3,    3,    3,    3,    3,    3,    3,    3,    3,
3,
```

```
                 3,      3,      3,      3,      3,      3,      3,      3,      3,      3,
3,
                 3,      3,      3,      3,      3,      3,      3,      3,      3,      3,
3,
                 3,      3,      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,      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,
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,      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,
                 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,      1,      1,      1,      1,
```

```
1,
          1,    1,    1], dtype=int64)
```

```
df['city'].unique()
#'unique()' function in pandas is used to find the unique values in a
column.
```

```
array(['Ahmedabad', 'Delhi', 'Bangalore', 'Mumbai', 'Kalyan',
'Faridabad',
       'Mettur', 'Hyderabad', 'Kaithal', 'Gurgaon', 'Pune', 'Noida',
       'Nashik', 'Kochi', 'Allahabad', 'Samastipur', 'Nadiad',
'Lucknow',
       'Jaipur', 'Karnal', 'Gorakhpur', 'Vidisha', 'Hosur',
'Bagalkot',
       'Baripara', 'Agra', 'Dharwad', 'Vadodara', 'Jalandhar',
'Surat',
       'Chennai', 'Navi Mumbai', 'Gandhidham', 'Visakhapatnam',
       'Thrissur', 'Kolkata', 'Ernakulam', 'Barasat', 'Ghaziabad',
       'Bhubaneshwar', 'Amritsar', 'Bhopal', 'Hamirpur(hp)',
'Kottayam',
       'Arrah', 'Patiala', 'Ranga Reddy', 'Mandi', 'Ludhiana',
'Mandya',
       'Siliguri', 'Aurangabad', 'Kanpur', 'Bhilwara', 'Meerut',
'Rewari',
       'Ahmednagar', 'Wardha', 'Chandigarh', 'Ranchi', 'Panvel',
'Thane',
       'Jabalpur', 'Kota', 'Rohtak', 'Rajkot', 'Varanasi', '24
Pargana',
       'Banka', 'Nagpur', 'Banki', 'Pali', 'Chhatarpur', 'Katihar',
       'Mohali', 'Rudrapur', 'Coimbatore', 'Jajpur', 'Mysore',
'Adoni',
       'Bikaner', 'Malout', 'Jammu', 'Rajnandgaon', 'Unnao',
'Godhara',
       'Kolhapur', 'Satara', 'Siwan', 'Dadra & Nagar Haveli',
'Bhiwani',
       'Koppal', 'Nizamabad', 'Madurai', 'Ujjain', 'Palakkad',
       'Tiruvallur', 'Panchkula', 'Nanjangud', 'Jhansi', 'Sonipat',
       'Puttur', 'Hoshiarpur', 'Gohana', 'Gautam Buddha Nagar',
       'Durgapur', 'Palwal', 'Chatrapur', 'Howrah', 'Jind', 'Hubli',
       'Panipat', 'Bharatpur', 'Vellore', 'Ambala', 'Guwahati',
'Gangtok',
       'Rajahmundry', 'Tiruchirappalli', 'Belgaum', 'Balaghat',
'Jatani',
       'Asansol', 'Bilaspur', 'Thanjavur', 'Raigarh(mh)', 'Mandi
Dabwali',
       'Basti', 'Bolpur', 'Aligarh', 'Balrampur', 'Ratnagiri',
'Muktsar',
       'Baran', 'Haldwani', 'Thiruvananthapuram', 'Indore', 'Buxar',
       'Chaksu', 'Haridwar', 'Bharuch', 'Muvattupuzha', 'Patna',
       'Simdega', 'Singhbhum', 'Bardhaman', 'Pathankot', 'Kharar',
```

```
       'Silchar', 'Jhalawar', 'Roorkee', 'Saharanpur', 'Solapur',
       'Gwalior', 'Alibag', 'Katni', 'Khedbrahma', 'Valsad', 'Satna',
       'Hooghly', 'Gurdaspur', 'Dadri', 'Amravati', 'Durg', 'Mehsana',
       'Lansdowne', 'Cuttack', 'Jaisalmer', 'Hanumangarh',
'Dungarpur',
       'Sri Ganganagar', 'Margao', 'Chinsurah', 'Bhatinda',
'Sibsagar',
       'Khalilabad', 'Dehradun', 'Anand', 'Sambalpur', 'Ankleshwar',
       'Purnia', 'Tiruverkadu', 'Bahadurgarh', 'Udaipur', 'Jodhpur',
       'Sheikhpura', 'Pondicherry', 'Sirsa', 'Godavari', 'Ajmer',
       'Moradabad', 'Raipur', 'Navsari', 'Herbertpur', 'Jamshedpur',
       'Ramanagar', 'Berhampur', 'Vijayawada', 'Murad Nagar',
       'Chandrapur', 'Jamtara', 'Uppidamangalam', 'Nalagarh', 'Una',
       'Chakan', 'Idukki', 'Shivpuri', 'Arkalgud', 'Bidar',
'Rupnagar',
       'Deoghar', 'Kanchipuram', 'Vapi', 'Medak', 'Kasargode',
'Dhanbad',
       'Dakshina Kannada', 'Ganaur', 'Jamalpur', 'Amraoti',
'Mangalore',
       'Deolali', 'Gandhinagar', 'Chitradurga', 'Chinchwad',
'Jhajjar',
       'Jagdalpur', 'Ranoli', 'Raiwala', 'Guntur', 'Badarpur',
'Adalaj',
       'Alipore', 'Bhawani Mandi', 'Mughalsarai', 'Kollam',
'Farukhabad',
       'Thiruvallur', 'Udaipurwati', 'Rasra', 'Latur', 'Krishna',
       'Gangaikondan', 'Warangal', 'Uluberia', 'Poonamallee',
'Nagaon',
       'Hissar', 'Kanyakumari', 'Morbi', 'Bankura', 'Virar',
'Tikamgarh',
       'Sultanpur', 'Tirunelveli', 'Bihar Shariff', 'Goa-panaji',
       'Ganganagar', 'Kolar', 'Bahadurpur', 'Batala', 'Budhlada',
       'Muzaffarnagar', 'Adyar', 'Calicut', 'Raigarh', 'Sonepat',
       'Chikkaballapur', 'Kasba', 'Bulandshahr', 'Burdwan', 'Anjar',
       'Marandahalli', 'Badaun', 'Namakkal', 'Puri', 'Alwar',
       'Surendranagar', 'Khandela', 'Kullu', 'Mohammadabad',
'Sangareddy',
       'Ghazipur', 'Shimla', 'Azamgarh', 'Chenani', 'Kanpur Nagar',
       'Trivandrum', 'Secunderabad', 'Kurukshetra', 'Dhariawad',
       'Bargarh', 'Gadarpur', 'Chikamaglur', 'Karim Nagar', 'Kotdwar',
       'Jalaun', 'Parola', 'Bareilly', 'Salem', 'Indi', 'Muzaffarpur',
       'Nayagarh', 'Jalgaon', 'Ambikapur', 'Udupi', 'Junagadh',
       'Dibrugarh', 'Faridkot', 'Naraingarh', 'Karwar',
       'Sant Kabir Nagar', 'Viramgam', 'Manali', 'Gadwal', 'Honavar',
       'Mathura', 'Khandwa', 'Solan', 'Sitapur', 'Betul', 'Anantapur',
       'Sholapur', 'Pinjore', 'Qadian', 'Sangrur', 'Jorhat',
'Palanpur',
       'Narnaul', 'Palamu', 'Falakata', 'Ferozepur', 'Porbandar',
       'Dwarka', 'Rangpo', 'Cannanore (kannur)', 'Churu', 'Baghpat',
```

```
       'Jhumri Tilaiya', 'Naihati', 'Virudhunagar', 'Dharmavaram',
       'Darbhanga', 'Nawanshahr', 'Sangli', 'Suri', 'Yamuna Nagar',
       'Vasai', 'Aluva', 'Sirsi', 'Bijapur', 'Krishnagar', 'Bhiwadi',
       'Bellary', 'Erode', 'Aquem', 'Nellore', 'Udhampur', 'Dhamtari',
       'Vandalur', 'Motihari', 'Dharwar', 'Shimoga', 'Jhunjhunu',
       'Bijnor', 'Yemmiganur', 'Bokaro', 'Kurnool', 'Srinagar',
'Ranip',
       'Davanagere', 'Rajouri', 'Begusarai', 'Goregaon', 'Bally',
       'Kachchh', 'Nagaur', 'Anekal', 'Mansa', 'Nanded',
'Dharamasala',
       'Chhindwara', 'Jamnagar', 'Zirakpur', 'Abohar', 'Barabanki',
       'Nabha', 'Kadapa', 'Perumbavoor', 'Sundargarh', 'Nazira',
       'Pratapgarh', 'Dharmapuri', 'Thangadh', 'Lonavala',
'Vizianagaram',
       'Kathua', 'Deesa', 'Tiruppur', 'Gadchiroli', 'Gangaghat',
'Bhuj',
       'Vastral', 'Phagwara', 'Kheda', 'Swaimadhopur', 'Kharagpur',
       'Jobner', 'Gondia', 'Bundi', 'Hamirpur', 'Dongargaon',
       'Mubarakpur', 'Tumkur', 'Sanand', 'Kartarpur', 'Bhavnagar',
       'Farrukhabad', 'Kadi', 'Seppa', 'Challakere', 'Dhubri',
'Deoria',
       'Akot', 'Alappuzha', 'Bhiwandi', 'Shillong', 'Osmanabad',
'Kendua',
       'Uran', 'Jaunpur', 'Hisar', 'Bodhan', 'Bhubaneswar', 'Raiganj',
       'Bhilai Nagar', 'Baloda', 'Anantnag', 'Berhampore', 'Silvasa',
       'Hospet', 'Palai', 'Sidhi'], dtype=object)
```

```
df['city'].nunique()
#The 'nunique()' function in pandas returns the number of unique
values in a specified column.
```

```
443
```

```
# this all are the condition which can be used in the 'and' operator

filter_df = df[(df['price']<=85000) & (df['age']<=2) &
(df['owner']=='First Owner')]
filter_df
```

```
                                bike_name    price       city
kms_driven  \
22    Hero Splendor iSmart Plus IBS 110cc  46500.0      Delhi
3500.0
29                Honda X-Blade 160CC ABS  81200.0     Mettur
9100.0
47             Mahindra Centuro NXT 110cc  28000.0     Jaipur
45000.0
95                     Bajaj Pulsar 150cc  60000.0       Pune
2000.0
121   Hero Splendor iSmart Plus IBS 110cc  48672.0  Ernakulam
```

```
608.0
...                                    ...      ...        ...
...
8335               Bajaj Pulsar 150cc Neon  72000.0       Pune
3162.0
8531                  Hero Xtreme 200R ABS  85000.0  Bangalore
40.0
8536                Hero Glamour i3s 125cc  75000.0    Gurgaon
7488.0
8796                  Hero Xtreme 200R ABS  85000.0  Bangalore
40.0
9322             TVS Radeon 110cc Drum SBT  58000.0      Delhi
4020.0

            owner  age  power     brand
22    First Owner  2.0  110.0      Hero
29    First Owner  2.0  160.0     Honda
47    First Owner  2.0  110.0  Mahindra
95    First Owner  2.0  150.0     Bajaj
121   First Owner  2.0  110.0      Hero
...           ...  ...    ...       ...
8335  First Owner  2.0  150.0     Bajaj
8531  First Owner  2.0  200.0      Hero
8536  First Owner  2.0  125.0      Hero
8796  First Owner  2.0  200.0      Hero
9322  First Owner  2.0  110.0       TVS

[120 rows x 8 columns]
```

```
filter_df[filter_df['city']=="Jaipur"]   #Returns a new DataFrame with
only the rows where the condition is True.
```

```
                          bike_name    price    city  kms_driven
owner  \
47       Mahindra Centuro NXT 110cc  28000.0  Jaipur     45000.0
First Owner
2185  Bajaj Platina 110 H Gear Disc  43500.0  Jaipur     18346.0
First Owner
3942                Honda Livo 110cc  52000.0  Jaipur     12322.0
First Owner
4525                Honda Livo 110cc  52000.0  Jaipur     12322.0
First Owner
4749                Honda Livo 110cc  53214.0  Jaipur     15492.0
First Owner
4781                Honda Livo 110cc  53214.0  Jaipur     15492.0
First Owner
7311           TVS Apache RTR 180cc  70000.0  Jaipur    207102.0
First Owner

      age  power     brand
```

```
47    2.0  110.0  Mahindra
2185  2.0  110.0     Bajaj
3942  2.0  110.0     Honda
4525  2.0  110.0     Honda
4749  2.0  110.0     Honda
4781  2.0  110.0     Honda
7311  2.0  180.0       TVS
```

```
df['brand'].value_counts()     # this is used for data numbers or values of series
```

```
brand
Bajaj               11213
Hero                 6368
Royal Enfield        4178
Yamaha               3916
Honda                2108
Suzuki               1464
TVS                  1247
KTM                  1077
Harley-Davidson       737
Kawasaki               79
Hyosung                64
Benelli                56
Mahindra               55
Triumph                26
Ducati                 22
BMW                    16
Jawa                   10
MV                      4
Indian                  3
Ideal                   2
Rajdoot                 1
LML                     1
Yezdi                   1
Name: count, dtype: int64
```

```
df[df['brand']=="Royal Enfield"]    #Used to filter rows in a DataFrame df where the value in the 'brand' column is equal to "Royal Enfield".
```

```
                                      bike_name     price  \
1                   Royal Enfield Classic 350cc  119900.0
8              Royal Enfield Thunderbird X 350cc  145000.0
9      Royal Enfield Classic Desert Storm 500cc   88000.0
23           Royal Enfield Classic Chrome 500cc  121700.0
36                  Royal Enfield Classic 350cc   98800.0
...                                         ...       ...
32601               Royal Enfield Classic 350cc   95500.0
32614        Royal Enfield Bullet Electra 350cc  105000.0
```

```
32633                Royal Enfield Classic 350cc   87000.0
32634           Royal Enfield Thunderbird 350cc   70000.0
32639                Royal Enfield Classic 350cc   95500.0

                        city  kms_driven         owner   age  power  \
1                      Delhi     11000.0   First Owner   4.0  350.0
8                  Bangalore      9190.0   First Owner   3.0  350.0
9                      Delhi     19000.0  Second Owner   7.0  500.0
23                    Kalyan     24520.0   First Owner   5.0  500.0
36                     Kochi     39000.0   First Owner   5.0  350.0
...                      ...         ...           ...   ...    ...
32601                  Delhi     18000.0   First Owner   8.0  350.0
32614                  Delhi     20000.0   First Owner   4.0  350.0
32633  Gautam Buddha Nagar      16336.0   First Owner   7.0  350.0
32634                Mumbai     13858.0  Second Owner  11.0  350.0
32639                  Delhi     18000.0   First Owner   8.0  350.0

                brand
1      Royal Enfield
8      Royal Enfield
9      Royal Enfield
23     Royal Enfield
36     Royal Enfield
...              ...
32601  Royal Enfield
32614  Royal Enfield
32633  Royal Enfield
32634  Royal Enfield
32639  Royal Enfield

[4178 rows x 8 columns]
```

```
df.duplicated().sum()
# used to identify and count the number of duplicate rows in a
DataFrame 'df'.
```

```
25324
```

```
df.drop_duplicates(inplace=True)
#Removes duplicate rows from the DataFrame 'df' and modifies the
DataFrame in place without returning a new DataFrame.

df.shape   # number of rows and columns in a dataset
```

```
(7324, 8)
```

```
## column remove and adding columns
df.drop('bike_name',axis='columns')   # to remove sinlge column from
the dataframe
```

```
         price       city  kms_driven        owner  age  power  \
0      35000.0  Ahmedabad     17654.0  First Owner  3.0  110.0
1     119900.0      Delhi     11000.0  First Owner  4.0  350.0
2     600000.0      Delhi       110.0  First Owner  8.0  675.0
3      65000.0  Bangalore     16329.0  First Owner  4.0  180.0
4      80000.0  Bangalore     10000.0  First Owner  3.0  150.0
...        ...        ...         ...          ...  ...    ...
9362   25000.0      Delhi     48587.0  First Owner  8.0  150.0
9369   35000.0  Bangalore     60000.0  First Owner  9.0  220.0
9370  450000.0    Jodhpur      3430.0  First Owner  4.0  750.0
9371  139000.0  Hyderabad     21300.0  First Owner  4.0  400.0
9372   80000.0  Hyderabad      7127.0  First Owner  5.0  220.0

                brand
0                 TVS
1       Royal Enfield
2             Triumph
3                 TVS
4              Yamaha
...               ...
9362            Hero
9369           Bajaj
9370  Harley-Davidson
9371           Bajaj
9372           Bajaj

[7324 rows x 7 columns]
```

```
df.drop(['bike_name','kms_driven'],axis='columns')   # inplace=True

# feature , column , variable
```

```
         price       city        owner  age  power             brand
0      35000.0  Ahmedabad  First Owner  3.0  110.0               TVS
1     119900.0      Delhi  First Owner  4.0  350.0    Royal Enfield
2     600000.0      Delhi  First Owner  8.0  675.0           Triumph
3      65000.0  Bangalore  First Owner  4.0  180.0               TVS
4      80000.0  Bangalore  First Owner  3.0  150.0             Yamaha
...        ...        ...          ...  ...    ...               ...
9362   25000.0      Delhi  First Owner  8.0  150.0              Hero
9369   35000.0  Bangalore  First Owner  9.0  220.0             Bajaj
9370  450000.0    Jodhpur  First Owner  4.0  750.0  Harley-Davidson
9371  139000.0  Hyderabad  First Owner  4.0  400.0             Bajaj
9372   80000.0  Hyderabad  First Owner  5.0  220.0             Bajaj

[7324 rows x 6 columns]
```

```
## adding new columns with scaler value
df['demo'] = "upflairs"
```

```
df['kms_driven'] - 50
#To subtract a constant value (like 50) from all elements in the
'kms_driven' column of a DataFrame 'df'.
```

```
0       17604.0
1       10950.0
2          60.0
3       16279.0
4        9950.0
         ...
9362    48537.0
9369    59950.0
9370     3380.0
9371    21250.0
9372     7077.0
Name: kms_driven, Length: 7324, dtype: float64
```

```
df['reduced_kms_driven'] = df['kms_driven'] - 50
# derived a new column from existing column

df['bike_name'].dtype
```

```
dtype('O')
```

```
df['bike_name_with_brand'] = df['bike_name']+" " +df['brand']
#Creating a new column by adding previous columns
df['bike_name_with_brand']
```

```
0               TVS Star City Plus Dual Tone 110cc TVS
1            Royal Enfield Classic 350cc Royal Enfield
2                         Triumph Daytona 675R Triumph
3                             TVS Apache RTR 180cc TVS
4          Yamaha FZ S V 2.0 150cc-Ltd. Edition Yamaha
                              ...
9362                    Hero Hunk Rear Disc 150cc Hero
9369                         Bajaj Avenger 220cc Bajaj
9370    Harley-Davidson Street 750 ABS Harley-Davidson
9371                       Bajaj Dominar 400 ABS Bajaj
9372                    Bajaj Avenger Street 220 Bajaj
Name: bike_name_with_brand, Length: 7324, dtype: object
```

```
df.dtypes  # object or string
```

```
bike_name                object
price                   float64
city                     object
kms_driven              float64
owner                    object
age                     float64
power                   float64
brand                    object
```

```
demo                     object
reduced_kms_driven      float64
bike_name_with_brand     object
dtype: object
```

```
df.drop(['demo','reduced_kms_driven','bike_name_with_brand'],axis=1,in
place=True)
# 1 == column
# 0 == row

# categorical object and numerical
df[['bike_name','city','owner','brand']]
```

```
                                  bike_name       city        owner  \
0       TVS Star City Plus Dual Tone 110cc  Ahmedabad  First Owner
1             Royal Enfield Classic 350cc      Delhi  First Owner
2                   Triumph Daytona 675R      Delhi  First Owner
3                   TVS Apache RTR 180cc  Bangalore  First Owner
4     Yamaha FZ S V 2.0 150cc-Ltd. Edition  Bangalore  First Owner
...                                    ...        ...          ...
9362            Hero Hunk Rear Disc 150cc      Delhi  First Owner
9369                 Bajaj Avenger 220cc  Bangalore  First Owner
9370       Harley-Davidson Street 750 ABS    Jodhpur  First Owner
9371               Bajaj Dominar 400 ABS  Hyderabad  First Owner
9372            Bajaj Avenger Street 220  Hyderabad  First Owner

                brand
0                 TVS
1       Royal Enfield
2             Triumph
3                 TVS
4              Yamaha
...               ...
9362            Hero
9369           Bajaj
9370  Harley-Davidson
9371           Bajaj
9372           Bajaj

[7324 rows x 4 columns]
```

```
#This function filters the columns of 'df' based on the data type
specified in the 'include' parameter.
cat_col = df.select_dtypes(include='O')   # 'O'  ==> object
cat_col.head()
```

```
                               bike_name       city        owner
brand
0    TVS Star City Plus Dual Tone 110cc  Ahmedabad  First Owner
TVS
1           Royal Enfield Classic 350cc      Delhi  First Owner  Royal
```

```
Enfield
2                  Triumph Daytona 675R      Delhi  First Owner
Triumph
3                  TVS Apache RTR 180cc  Bangalore  First Owner
TVS
4  Yamaha FZ S V 2.0 150cc-Ltd. Edition  Bangalore  First Owner
Yamaha
```

```
#This function filters the columns of 'df' based on the data type
specified in the exclude parameter.
num_col = df.select_dtypes(exclude='O')   # 'O'  ==> object
num_col.head()
```

```
      price  kms_driven  age  power
0   35000.0     17654.0  3.0  110.0
1  119900.0     11000.0  4.0  350.0
2  600000.0       110.0  8.0  675.0
3   65000.0     16329.0  4.0  180.0
4   80000.0     10000.0  3.0  150.0
```

```
cat_col.shape  # dimension of array in row/columns
```

```
(7324, 4)
```

```
num_col.shape  # dimension of array in row/columns
```

```
(7324, 4)
```

```
combined_df = pd.concat([num_col , cat_col],axis=1)
# This function concatenates DataFrames 'num_col' and 'cat_col' along
the columns axis('axis'=1).
combined_df.head()
# 'combined_df' is assigned
# '.head()' is used to display the first few rows of the resulting
concatenated DataFrame.
```

```
      price  kms_driven  age  power
bike_name  \
0   35000.0     17654.0  3.0  110.0    TVS Star City Plus Dual Tone
110cc
1  119900.0     11000.0  4.0  350.0           Royal Enfield Classic
350cc
2  600000.0       110.0  8.0  675.0                    Triumph Daytona
675R
3   65000.0     16329.0  4.0  180.0                    TVS Apache RTR
180cc
4   80000.0     10000.0  3.0  150.0  Yamaha FZ S V 2.0 150cc-Ltd.
Edition

        city        owner          brand
0  Ahmedabad  First Owner            TVS
```

```
1      Delhi  First Owner  Royal Enfield
2      Delhi  First Owner        Triumph
3  Bangalore  First Owner            TVS
4  Bangalore  First Owner         Yamaha
```

```
# pd.merge()  # join
```

```
df['owner'].value_counts()
```

```
owner
First Owner             6642
Second Owner             588
Third Owner               84
Fourth Owner Or More      10
Name: count, dtype: int64
```

```
dt = {'First Owner':1,'Second Owner':2,'Third Owner':3,'Fourth Owner
Or More':4}
dt
```

```
#giving a number to various onwer type given in 'owner' column.
```

```
{'First Owner': 1,
 'Second Owner': 2,
 'Third Owner': 3,
 'Fourth Owner Or More': 4}
```

```
# displaying replaced value for 'owner'
for owner in df['owner']:
    print(dt[owner])
```

```
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
2
1
```

```
1
1
1
2
1
1
1
2
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
```

```
1
2
1
1
1
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
2
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
3
2
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

1
3
1
1
2
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1

```
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
1
1
3
1
1
1
1
1
1
1
1
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
2
1
```

```
1
1
1
1
1
2
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
2
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
1
1
1
1
1
3
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
2
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
```

```
1
1
1
2
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
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
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
3
1
1
1
1
2
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
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
2
1
1
1
1
3
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
1
1
1
```

```
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
3
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
2
2
```

```
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
1
1
1
1
1
2
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
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
1
2
1
1
1
1
2
1
1
1
1
1
1
2
1
1
```

```
1
1
1
1
1
1
1
3
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
2
```

```
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
2
2
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
```

```
1
1
1
1
1
1
1
2
2
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
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
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
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
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
2
1
2
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
1
2
1
1
1
1
2
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
3
1
2
1
1
1
1
```

```
1
1
1
1
2
1
1
1
1
1
1
1
1
3
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
1
1
```

```
1
2
1
1
1
1
1
1
1
1
2
1
2
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
1
2
1
1
1
1
1
1
1
2
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
1
2
1
1
1
2
2
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
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
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
1
2
2
1
3
4
2
2
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
3
1
2
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
3
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
```

```
1
1
2
1
1
2
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
1
2
1
1
3
1
1
1
1
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
2
1
2
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
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
```

```
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
3
2
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
```

```
1
1
1
1
1
1
1
2
1
2
3
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
2
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
1
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
2
2
2
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
2
1
1
2
1
1
1
1
2
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
3
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
2
1
1
1
1
1
1
2
1
1
1
2
1
1
1
1
1
1
1
2
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
1
1
1
1
```

```
1
1
1
1
2
1
1
1
1
1
2
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
1
1
3
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
1
1
3
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
4
1
1
1
2
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
```

```
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
```

```
1
1
1
1
1
1
1
2
1
2
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
1
2
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
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
```

```
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

2
1
1
1
1
1
1
1
1
1
1
1
2
2
1
2
1
1
2
1
2
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1

```
1
1
1
2
1
1
1
2
1
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
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
4
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
1
1
1
1
1
1
3
1
1
```

```
1
1
2
2
2
1
1
2
1
1
1
1
1
1
1
1
1
1
1
3
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
2
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
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
1
1
2
2
1
2
1
3
1
1
2
1
1
1
2

```
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
1
1
1
2
1
1
2
1
1
1
1
1
2
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
4
2
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
1
1
1
1
1
1
2
1
1
1
1
2
2
2
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
1
1
1
1
2
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
1
2
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
2
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
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
1
1
2
2
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
2
1
3
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
3
2
2
1
1
1
1
```

```
1
1
1
2
1
1
1
1
1
1
1
2
3
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
1
1
1
1
1
1
3
1
2
1
1
1
1
2
1
1
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
2
3
3
2
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
```

```
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
2
1
3
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
1
1
1
1
1
2
1
1
1
3
1
1
1
1
1
1
2
2
1
1
1
1
1
1
```

```
1
2
1
1
2
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
2
1
1
1
1
1
1
2
1
1
2
2
1
1
1
1
1
3
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
```

```
1
3
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
2
1
1
1
2
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
1
1
2
1
1
```

```
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
```

```
1
1
1
1
2
1
2
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
1
1
4
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
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
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
```

```
3
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
1
1
1
1
1
1
1
1
1
1
3
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
2
1
1
1
2
1
```

```
1
1
1
2
1
1
1
1
1
2
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
```

```
4
1
2
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
1
1
2
1
1
1
2
3
1
1
```

```
1
1
1
1
1
1
1
2
1
1
1
1
1
2
1
1
1
1
1
1
2
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
2
1
2
1
1
1
1
1
2
1
```

```
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
3
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
1
1
1
3
1
1
1
1
1
1
2
1
1
1
2
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
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
1
1
1
1
3
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
1
1
1
2
2
1
1
1
1
1
1
1
1
3
2
```

2
3
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
2
3
1
1
1
1
2
1
2
1
1
2
1
1
3
1
1
1
1
1
1
1

```
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
2
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
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
1
3
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
1
2
2
1
1
1
1
1
2
4
1
2
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
3
1
```

```
1
1
1
1
1
2
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
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
3
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
3
1
1
2
1
1
1
2
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
1
1
1
2
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
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
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
1
2
1
1
2
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
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
1
1
1
1
1
1
1
1
2
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
2
```

```
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
3
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
2
1
2
1
1
1
2
```

```
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
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
4
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1
1
4
1
1
1
1
2
1
1
1
```

```
1
3
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
3
1
1
1
3
2
1
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
2
1
1
1
2
1
2
1
1
1
1
```

```
1
1
1
1
2
1
1
1
2
1
1
1
1
1
2
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
1
1
3
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
2
1
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
1
1
1
1
2
1
1
2
1
2
1
1
1
1
1
```

1
1
2
1
1
1
2
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
1
2
2
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
1
1
3
2
1
1
1
1
1
2
3
2
1
3
1
1
1
1
3
1
1
1
1
1
1
1
1
1
3
1
1
1
1
```

```
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
3
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
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
2
1
1
1
1
1
1
1
2
1
```

```
1
1
1
1
1
2
2
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
3
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
2
1
2
1
1
```

```
1
1
1
1
1
1
2
1
1
1
1
2
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
2
1
1
2
1
1
1
3
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
1
1
2
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
```

```
1
1
1
3
2
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
1
1
1
1
```

```
1
1
1
1
1
3
2
1
1
1
1
1
1
1
1
1
1
1
2
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
3
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
```

```
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
2
1
1
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
2
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
1
1
1
1
1
1
2
2
2
1
1
1
1
1
1
1
1
1
1
3
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
3
1
1
1
1
1
1
2
```

```
1
1
1
1
1
1
1
1
1
1
1
3
1
1
2
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
3
1
1
2
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
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
3
3
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

2
1
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1
2
2
3
1
1
1
1
1
1
1

```
1
2
1
1
1
2
1
1
1
1
1
2
1
1
1
1
1
1
3
1
2
1
1
1
1
1
2
1
1
2
1
1
1
1
1
1
1
1
1
3
1
1
1
1
2
1
1
1
```

2
1
1
1
1
2
1
1
1
1
1
1
2
1
1
1
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
1
1
1

```
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
1
1
1
3
1
1
1
1
1
1
1
1
1
1
1
2
1
1
2
2
1
2
1
1
2
1
2
2
1
2
```

```
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
3
2
1
2
1
1
1
1
1
1
1
1
1
2
1
1
1
1
1
1
```

```
1
1
1
1
1
1
2
1
2
2
2
2
2
2
2
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
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
4
2
1
1
1
1
1
2
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
1
1
1
1
1
1
1
1
1
1
1
1
1
3
1
2
2
1
2
1
1
1
2
1
1
1
```

```
2
1
2
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
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
1
1
1
1
1
1
1
1
1
1
3
1
1
1
1
1
1
1
1
```

```
df['owner2'] = df['owner'].map(dt)
```

```
df['owner'] = df['owner'].map(dt)
```

missing value handling

```
import numpy as np
import  pandas as pd
```

```
# Creating a dataset
data = {'A':[2,4,5,np.nan,78,np.nan,8],
       'B':[12,14,np.nan,78,np.nan,8,55],
       'C':[np.nan,4,5,78,np.nan,8,63],
        'D':[2,4,5,500,78,11,8]}
data
```

```
{'A': [2, 4, 5, nan, 78, nan, 8],
 'B': [12, 14, nan, 78, nan, 8, 55],
 'C': [nan, 4, 5, 78, nan, 8, 63],
 'D': [2, 4, 5, 500, 78, 11, 8]}
```

```
data
```

```
{'A': [2, 4, 5, nan, 78, nan, 8],
 'B': [12, 14, nan, 78, nan, 8, 55],
 'C': [nan, 4, 5, 78, nan, 8, 63],
 'D': [2, 4, 5, 500, 78, 11, 8]}
```

```
sample = pd.DataFrame(data)      # using pandas to represent data
sample
```

```
      A     B     C    D
0   2.0  12.0   NaN    2
1   4.0  14.0   4.0    4
2   5.0   NaN   5.0    5
3   NaN  78.0  78.0  500
4  78.0   NaN   NaN   78
5   NaN   8.0   8.0   11
6   8.0  55.0  63.0    8
```

```
sample.isnull().sum()   #isnull = null/missing values
#command used for DataFrame to check for missing values.
```

```
A    2
B    2
C    2
D    0
dtype: int64
```

```
sample.isnull().sum().sum()
#calculating overall number of missing values in a data.
```

```
6
```

```
sample
```

```
      A     B     C    D
0   2.0  12.0   NaN    2
1   4.0  14.0   4.0    4
2   5.0   NaN   5.0    5
3   NaN  78.0  78.0  500
```

```
4  78.0   NaN   NaN   78
5   NaN   8.0   8.0   11
6   8.0  55.0  63.0    8
```

```
sample.dropna()  #use to drop the rows with a missing values
# but its not ideal for small data because rows like this can make
data less useful.
```

```
     A     B     C  D
1  4.0  14.0   4.0  4
6  8.0  55.0  63.0  8
```

```
sample.dropna(axis = 1)  #dropping columns of missing values
```

```
     D
0    2
1    4
2    5
3  500
4   78
5   11
6    8
```

fill the value of missing

missing value imputation

```
sample
```

```
      A     B     C    D
0   2.0  12.0   NaN    2
1   4.0  14.0   4.0    4
2   5.0   NaN   5.0    5
3   NaN  78.0  78.0  500
4  78.0   NaN   NaN   78
5   NaN   8.0   8.0   11
6   8.0  55.0  63.0    8
```

```
sample['A'].fillna(0)
# fill the missing values in column 'A' of the 'sample' DataFrame with
'0'.
```

```
0     2.0
1     4.0
2     5.0
3     0.0
4    78.0
5     0.0
```

```
6     8.0
Name: A, dtype: float64
```

```
sample.fillna(0) #replace missing values with '0'.
```

```
      A     B     C    D
0   2.0  12.0   0.0    2
1   4.0  14.0   4.0    4
2   5.0   0.0   5.0    5
3   0.0  78.0  78.0  500
4  78.0   0.0   0.0   78
5   0.0   8.0   8.0   11
6   8.0  55.0  63.0    8
```

```
sample['A'].mean()  #mean value of column 'A'.
```

```
19.4
```

```
sample['A'].median()    #median value of column 'A'.
```

```
5.0
```

```
sample['A'].fillna(sample['A'].median())
#fill the missing values in column 'A' of the 'sample' DataFrame with
the median value of that column.
```

```
0     2.0
1     4.0
2     5.0
3     5.0
4    78.0
5     5.0
6     8.0
Name: A, dtype: float64
```

```
sample['A'].fillna(sample['A'].mean())
#fill the missing values in column 'A' of the 'sample' DataFrame with
the mean value of that column.
```

```
0     2.0
1     4.0
2     5.0
3    19.4
4    78.0
5    19.4
6     8.0
Name: A, dtype: float64
```

group by operation

```
#write a program to find out minimum price of each and every brand

df = pd.read_csv(r"E:\used bike\Used_Bikes.csv")
df
```

```
                                    bike_name     price       city
kms_driven  \
0        TVS Star City Plus Dual Tone 110cc   35000.0  Ahmedabad
17654.0
1               Royal Enfield Classic 350cc  119900.0      Delhi
11000.0
2                     Triumph Daytona 675R  600000.0      Delhi
110.0
3                     TVS Apache RTR 180cc   65000.0  Bangalore
16329.0
4      Yamaha FZ S V 2.0 150cc-Ltd. Edition   80000.0  Bangalore
10000.0
...                                     ...       ...        ...
...
32643                Hero Passion Pro 100cc   39000.0      Delhi
22000.0
32644                  TVS Apache RTR 180cc   30000.0     Karnal
6639.0
32645              Bajaj Avenger Street 220   60000.0      Delhi
20373.0
32646             Hero Super Splendor 125cc   15600.0     Jaipur
84186.0
32647                   Bajaj Pulsar 150cc   22000.0       Pune
60857.0

              owner   age  power           brand
0      First Owner   3.0  110.0             TVS
1      First Owner   4.0  350.0  Royal Enfield
2      First Owner   8.0  675.0        Triumph
3      First Owner   4.0  180.0             TVS
4      First Owner   3.0  150.0          Yamaha
...            ...   ...    ...             ...
32643  First Owner   4.0  100.0           Hero
32644  First Owner   9.0  180.0             TVS
32645  First Owner   6.0  220.0           Bajaj
32646  First Owner  16.0  125.0            Hero
32647  First Owner  13.0  150.0           Bajaj

[32648 rows x 8 columns]
```

```
df['brand'].nunique()   # Number of Unique values
```

```
23
```

```
grouped = df.groupby('brand')
#group the rows of a DataFrame by the values in the 'brand' column.

grouped   #won't show the data but the structure
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at
0x000001F39534D040>
```

```
df['brand'].value_counts()    #count of value
```

```
brand
Bajaj              11213
Hero                6368
Royal Enfield       4178
Yamaha              3916
Honda               2108
Suzuki              1464
TVS                 1247
KTM                 1077
Harley-Davidson      737
Kawasaki              79
Hyosung               64
Benelli               56
Mahindra              55
Triumph               26
Ducati                22
BMW                   16
Jawa                  10
MV                     4
Indian                 3
Ideal                  2
Rajdoot                1
LML                    1
Yezdi                  1
Name: count, dtype: int64
```

```
grouped.get_group('Bajaj')
#retrieves all rows from the DataFrame that belong to the group
labeled 'Bajaj'.
```

```
                      bike_name     price       city  kms_driven  \
12           Bajaj Pulsar NS200   78000.0  Bangalore      9900.0
13          Bajaj Discover 100M   29499.0      Delhi     20000.0
14          Bajaj Discover 125M   29900.0      Delhi     20000.0
15       Bajaj Pulsar NS200 ABS   90000.0  Bangalore     11574.0
16       Bajaj Pulsar RS200 ABS  120000.0  Bangalore     23000.0
...                         ...       ...        ...         ...
32632  Bajaj Avenger Street 220   55005.0    Godhara      6600.0
32637         Bajaj Pulsar 150cc   25000.0      Delhi     32588.0
32641       Bajaj Avenger 220cc   41000.0      Delhi     20245.0
32645  Bajaj Avenger Street 220   60000.0      Delhi     20373.0
```

```
32647        Bajaj Pulsar 150cc   22000.0       Pune     60857.0
```

```
               owner   age  power  brand
12       First Owner   4.0  200.0  Bajaj
13       First Owner   8.0  100.0  Bajaj
14       First Owner   7.0  125.0  Bajaj
15       First Owner   3.0  200.0  Bajaj
16       First Owner   3.0  200.0  Bajaj
...              ...   ...    ...    ...
32632    First Owner   5.0  220.0  Bajaj
32637    First Owner   9.0  150.0  Bajaj
32641  Second Owner  11.0  220.0  Bajaj
32645    First Owner   6.0  220.0  Bajaj
32647    First Owner  13.0  150.0  Bajaj

[11213 rows x 8 columns]
```

```
grouped['price'].min()  #tells the cheapest price of every brand
```

```
brand
BMW                255000.0
Bajaj                6400.0
Benelli            110700.0
Ducati             380000.0
Harley-Davidson    250000.0
Hero                 5000.0
Honda                10000.0
Hyosung            120000.0
Ideal              100000.0
Indian             700000.0
Jawa               146000.0
KTM                 55000.0
Kawasaki           110000.0
LML                  4400.0
MV                 950000.0
Mahindra            17800.0
Rajdoot             75000.0
Royal Enfield       33500.0
Suzuki               8000.0
TVS                  5800.0
Triumph            500000.0
Yamaha               9400.0
Yezdi               68000.0
Name: price, dtype: float64
```

```
grouped[['price']].min()   #A better representation for above cell
```

```
                    price
brand
BMW              255000.0
```

```
Bajaj              6400.0
Benelli          110700.0
Ducati           380000.0
Harley-Davidson  250000.0
Hero               5000.0
Honda             10000.0
Hyosung          120000.0
Ideal            100000.0
Indian           700000.0
Jawa             146000.0
KTM               55000.0
Kawasaki         110000.0
LML                4400.0
MV               950000.0
Mahindra          17800.0
Rajdoot           75000.0
Royal Enfield     33500.0
Suzuki             8000.0
TVS                5800.0
Triumph          500000.0
Yamaha             9400.0
Yezdi             68000.0
```

```
grouped[['price']].max()  #maximum priced bike of every brands
```

```
                    price
brand
BMW               1800000.0
Bajaj              195000.0
Benelli            785000.0
Ducati            1500000.0
Harley-Davidson   1100000.0
Hero               104000.0
Honda              800000.0
Hyosung            493500.0
Ideal              100000.0
Indian            1900000.0
Jawa               223000.0
KTM                860000.0
Kawasaki          1100000.0
LML                  4400.0
MV                1500000.0
Mahindra           175000.0
Rajdoot             75000.0
Royal Enfield      285000.0
Suzuki            1260000.0
TVS                224000.0
Triumph           1300000.0
Yamaha            1550000.0
Yezdi               68000.0
```

quiz find out average price of every owner using group by

```
grouped = df.groupby('owner')

grouped.value_counts()
```

```
owner         bike_name                  price    city        kms_driven
age   power   brand
First Owner   Bajaj Avenger Street 220   70000.0  Mumbai      8000.0
5.0   220.0   Bajaj      620
              Bajaj Pulsar 150cc         10000.0  Ghaziabad   64955.0
15.0  150.0   Bajaj      620
                                         14227.0  Bhopal      36000.0
15.0  150.0   Bajaj      620
              Hero CBZ Xtreme 150cc      11900.0  Noida       34968.0
14.0  150.0   Hero       620
              Hero CD Deluxe 100cc       18000.0  Chennai     22824.0
8.0   100.0   Hero       620

...
Third Owner   Yamaha YZF-R15 150cc       33000.0  Jhajjar     57000.0
10.0  150.0   Yamaha       1
                                         39000.0  Bangalore   29570.0
12.0  150.0   Yamaha       1
              Yamaha YZF-R15 2.0 150cc   54000.0  Azamgarh    40000.0
8.0   150.0   Yamaha       1
                                         55000.0  Faridabad   28864.0
8.0   150.0   Yamaha       1
              Yezdi Classic 250cc        68000.0  Ahmedabad   23.0
39.0  250.0   Yezdi        1
Name: count, Length: 7324, dtype: int64
```

```
grouped[['price']].mean()
```

```
                              price
owner
First Owner           69512.420037
Fourth Owner Or More  61332.500000
Second Owner          53552.263651
Third Owner           81431.916667
```

```
grouped = df.groupby('brand')

grouped['price'].agg(min_prices = 'min', max_price = 'max', mean_price
= 'mean')
# perform multiple aggregation operations (minimum, maximum, and mean)
```

```
                  min_prices  max_price    mean_price
brand
BMW                 255000.0  1800000.0  5.987500e+05
Bajaj                 6400.0   195000.0  4.833127e+04
```

Benelli	110700.0	785000.0	2.942000e+05
Ducati	380000.0	1500000.0	9.355455e+05
Harley-Davidson	250000.0	1100000.0	4.529988e+05
Hero	5000.0	104000.0	2.382945e+04
Honda	10000.0	800000.0	5.923047e+04
Hyosung	120000.0	493500.0	2.491678e+05
Ideal	100000.0	100000.0	1.000000e+05
Indian	700000.0	1900000.0	1.100000e+06
Jawa	146000.0	223000.0	1.855000e+05
KTM	55000.0	860000.0	1.746697e+05
Kawasaki	110000.0	1100000.0	4.116246e+05
LML	4400.0	4400.0	4.400000e+03
MV	950000.0	1500000.0	1.325000e+06
Mahindra	17800.0	175000.0	7.250709e+04
Rajdoot	75000.0	75000.0	7.500000e+04
Royal Enfield	33500.0	285000.0	9.856207e+04
Suzuki	8000.0	1260000.0	4.594683e+04
TVS	5800.0	224000.0	4.429915e+04
Triumph	500000.0	1300000.0	8.274230e+05
Yamaha	9400.0	1550000.0	5.706896e+04
Yezdi	68000.0	68000.0	6.800000e+04