In [ ]:

```
Recap
-----
numpy ->array
           --> vector 1D ;  --> matrix 2D;  ndarray ->tensor(<slice>,row,cols)
                                ->rowxcol
           -->computation ; index; slicing ; reshape
           -->numpy functions()
|
pandas -->Series -->DataFrame (Table = Row X Cols)
                    |->input_ our structure(dict,list)
                    |->input_ file
                    |->df.loc[index] - row ; df['column']
                              |->slicing
                    |-> we can combine row with column
                    |-> we can do filter
                    |-> filter(),group(),query(),head(),tail(),sample(),shape,....
                    |-> omit duplication ; replace NaN value

|
matplotlib -> Visualizaion --> plot() => line style; scatter ->data point ; bar;heatmap;hi
                               pie ; boxplot
                             ------------------//own set of attributes(keyword args)

data analysis
===============
[Load the data(raw)] =>[structured Data(DataFrame)] =>[Data pre-processing] =>[Cleaned Dat
                                                        - no duplicate item
                                                        - no NaN
==>(A) ==>[ EDA ] - data relationship =>[Visualization]
                                                    //DataAnalysis
                                          |
                                       Model - ML
```

In [ ]:
```
Statistics
-----------
 |->elements
     |->1.population - Overall Data
        2.sample     - from population - select certain part of data
        3.variable   - characterstics
              | --> numerical (Quantitative)
                       |->discreate
                       |->continuous
              | -->  categorical Data(String)

     |->4. Types
             |---------> Descriptive statistics => df.describe()
             |---------> Inferential statistics

Descriptive Statistics
-------------------------
Collection ->classification ->Summarization =>Results:[chart;table]

Inferential statistics
----------------------
from sample data =>comparision ->estimation ->relationship ->Hypothesis testing ->predicti
=>results [ probability_score]

Descriptive Statistics
----------------------
mean  => Average of all values
median => central value of sample set
mode  ==> repeated value
range_ ==>  max() - min()

1 2 3 4  4 5 2 4 10 9 12 8
X x x x  x --- x   x  x  x    X
           |
           5+2 => 7
                  - = 3.5
                  2


1 2 3 4  4 5 2 4 10 9 12 8  => mode is 4
       ---     -
```

In [1]:
```
import statistics
d =[1,2,3,4,4,5,2,4,10,9,12,8]
statistics.median(d)
```

Out[1]: 4.0

In [2]:
```
statistics.mode(d)
```

Out[2]: 4

In [3]:
```
statistics.mean(d)
```

Out[3]: 5.333333333333333

In [ ]:
```
variance
 |->population variance  |->sample variance
```

In [4]:
```
statistics.mean([90,92,95])
```

Out[4]:
```
92.33333333333333
```

In [ ]:
```
S1 => 90 - 92.33  => 2.33
S2 => 92 - 92.33  => 0.33
S3 => 95 - 92.33  => 2.67
                     -------
                      5.33
                      ------

permutation & combination
--------------------------
DATA => A B C D

permutation
AB BA  CA DA
AC BC  CB DB
AD BD  CD DC
--------------// 12


combination
AB -  CA DA
- BC  - DB
- -  - DC
--------------// 6

InterQuartileRange(IQR)
|-> step 1: load the data in ascending order
    step 2: median
    step 3: Quartile
    ------------------//IQR
```

In [6]:
```
from itertools import permutations

count = 0
data = [1,2,3,4]

perm = permutations(data,2)
for v in list(perm):
    print(v)
    count += 1
```

```
(1, 2)
(1, 3)
(1, 4)
(2, 1)
(2, 3)
(2, 4)
(3, 1)
(3, 2)
(3, 4)
(4, 1)
(4, 2)
(4, 3)
```

In [7]:
```
count
```

Out[7]:
```
12
```

In [ ]:
```
## Sample selection
df.sample(n=20) <== simple random sampling

1. simple random sampling  => df.sample(n=20)
2. systemmatic sampling index ->(np.arange(0,len(df),step=<Count>))
3. cluster sampling => split data into multiple groups(cluster)
         n=5 - 5 cluster
      dataset['cluster_id'] = np.repeat()
                                          |__[range(1,n+1),len(dataset)/n]
    index=0
    n=5
    for var in range(0,len(dataset)):
        if dataset['cluster_id'].loc[var] % 2 == 0:
            index.append(var)
```

In [ ]:
```
Statistical Tool and technique used in data analysis are based on probability
probability - measures - how event to scale from 0 to 1

0 - notHappening(failed)  1 - Happening(success)
probability distribution
------------------------//How random variable is distributed

1.Discreate probability distribution                    2. continuous probability distribut

x = [1,2,3]                                                  [ 1 <= x <= 2 ]

 ->binomial distribution                                    -> Normal distribution
 ->poisson distribution
                                                    lies btn 0 and 1.83
p = success                                                 z = 1.83
q = failed                                                              1.8 - left
                                                                        0 03 - right z=.0966
p + q = 1
```

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
```

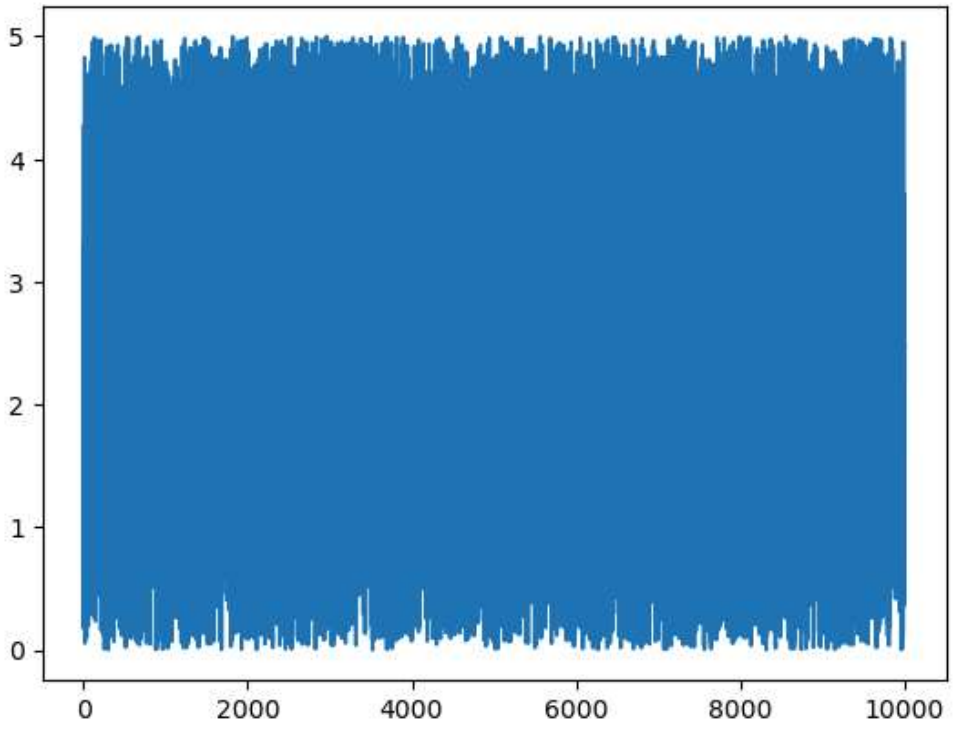
In [9]:
```
data = stats.uniform.rvs(size=10000,loc=0,scale=5)
```

In [10]:
```
data
```

Out[10]:
```
array([2.26239418, 2.17046602, 2.43475815, ..., 0.76463151, 2.49992663,
       2.46975861])
```

In [12]:
```
df = pd.DataFrame(data)
plt.plot(df[0])
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x24489300490>]
```

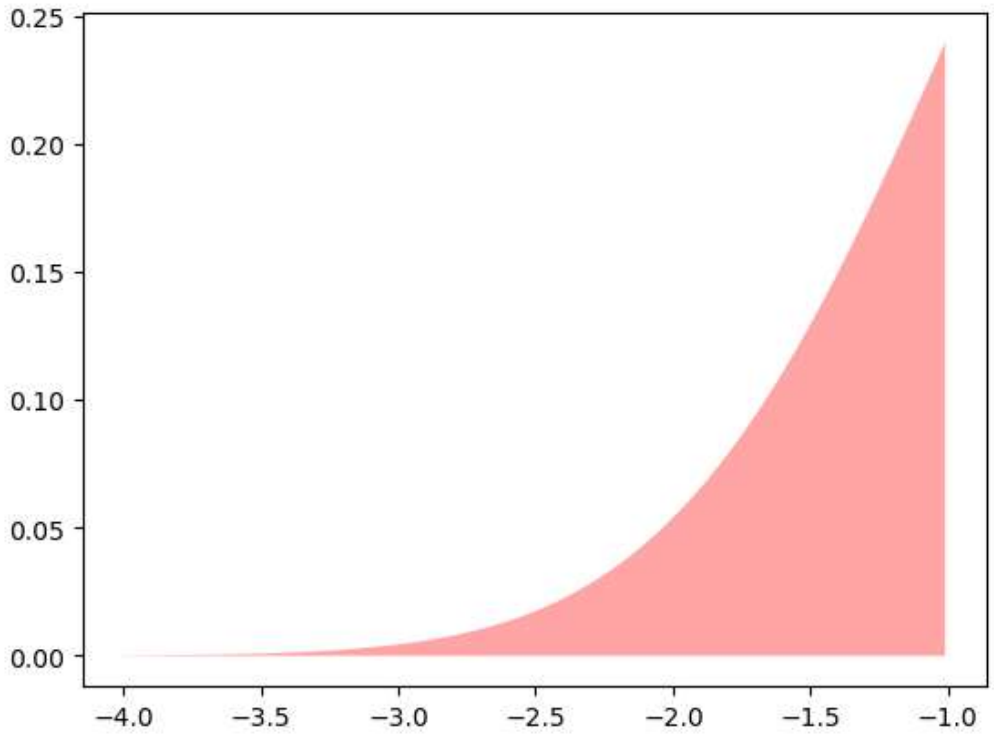


In [16]:
```
p_minus1 = stats.norm.cdf(x=-1,loc=0,scale=1)
p_over1  = 1-stats.norm.cdf(x=1,loc=0,scale=1)
btn_p = 1 - (p_minus1 + p_over1)

print(p_minus1,p_over1,btn_p)
```

```
0.15865525393145707 0.15865525393145707 0.6826894921370859
```

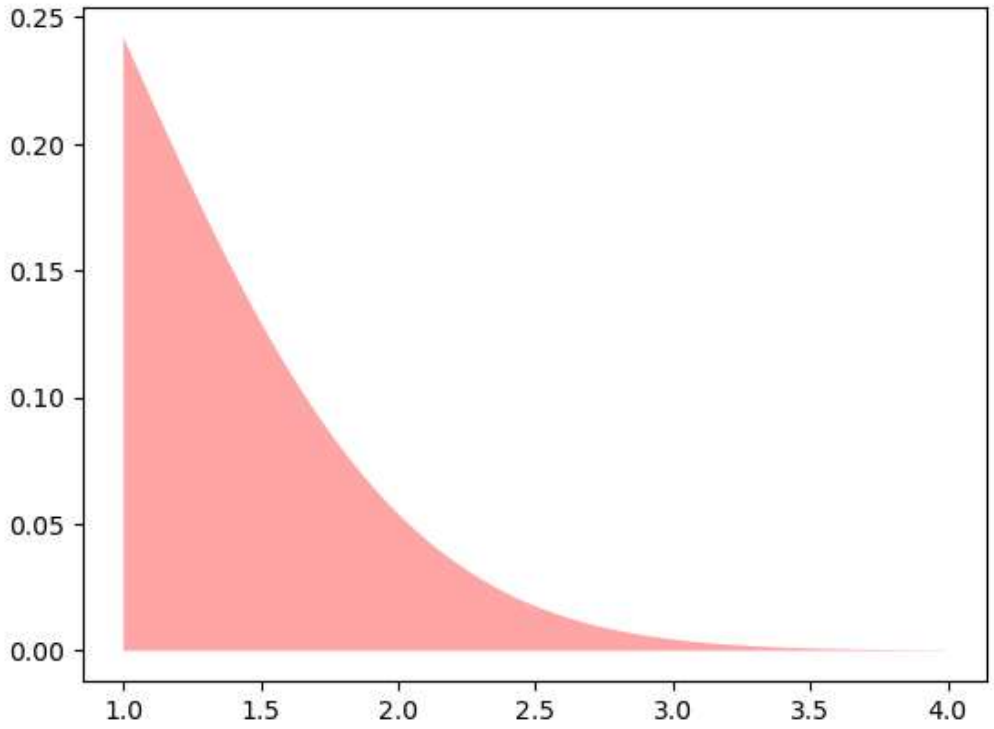
In [20]:
```
#help(plt.fill_between)

plt.fill_between(x=np.arange(-4,-1,0.01),
                 y1=stats.norm.pdf(np.arange(-4,-1,0.01)),
                 facecolor='red',
                 alpha=0.35)
```

Out[20]: <matplotlib.collections.PolyCollection at 0x24492c28f90>



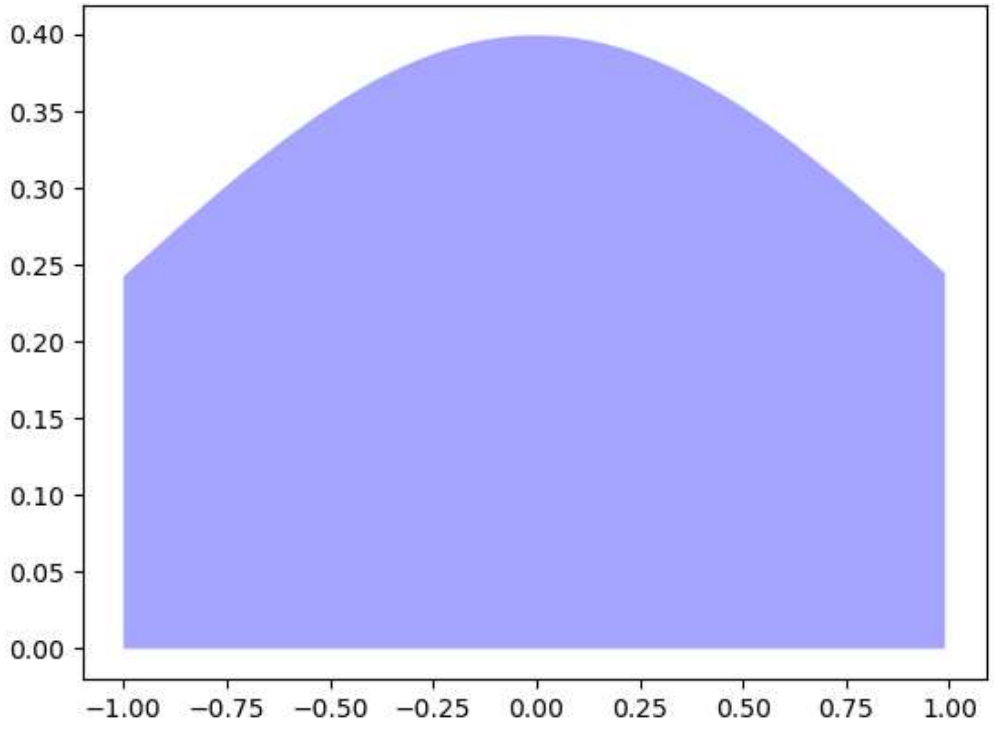
In [21]:
```
#help(plt.fill_between)

plt.fill_between(x=np.arange(1,4,0.01),
                 y1=stats.norm.pdf(np.arange(1,4,0.01)),
                 facecolor='red',
                 alpha=0.35)
```

Out[21]: <matplotlib.collections.PolyCollection at 0x24492c891d0>



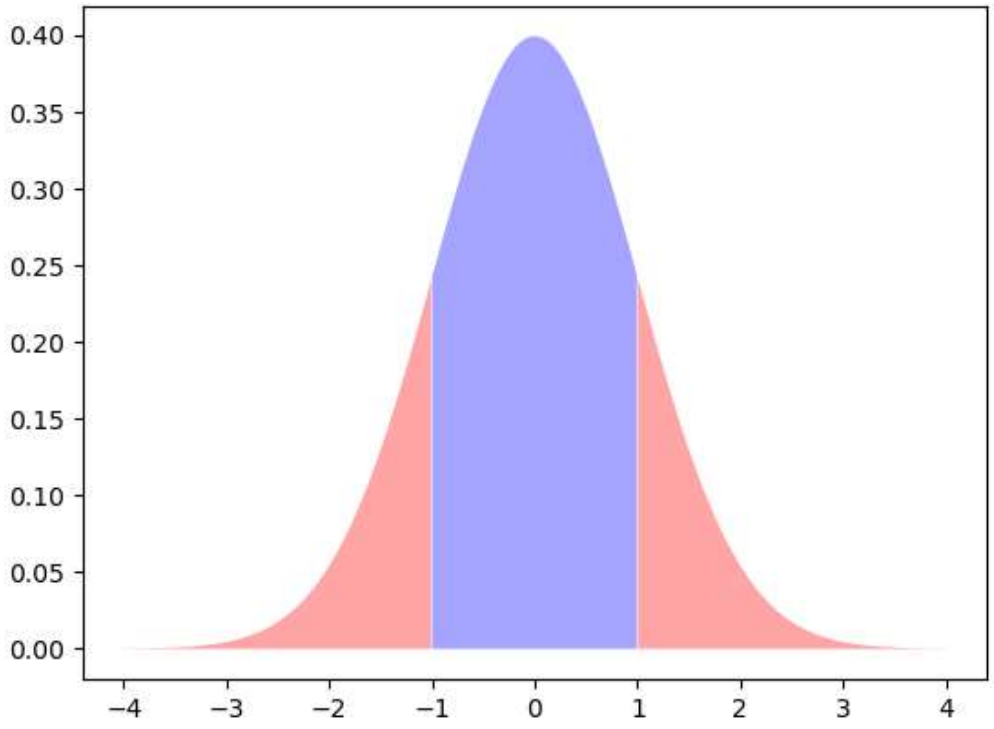
In [22]:
```
#help(plt.fill_between)

plt.fill_between(x=np.arange(-1,1,0.01),
                 y1=stats.norm.pdf(np.arange(-1,1,0.01)),
                 facecolor='blue',
                 alpha=0.35)
```

Out[22]: <matplotlib.collections.PolyCollection at 0x24492ceee90>



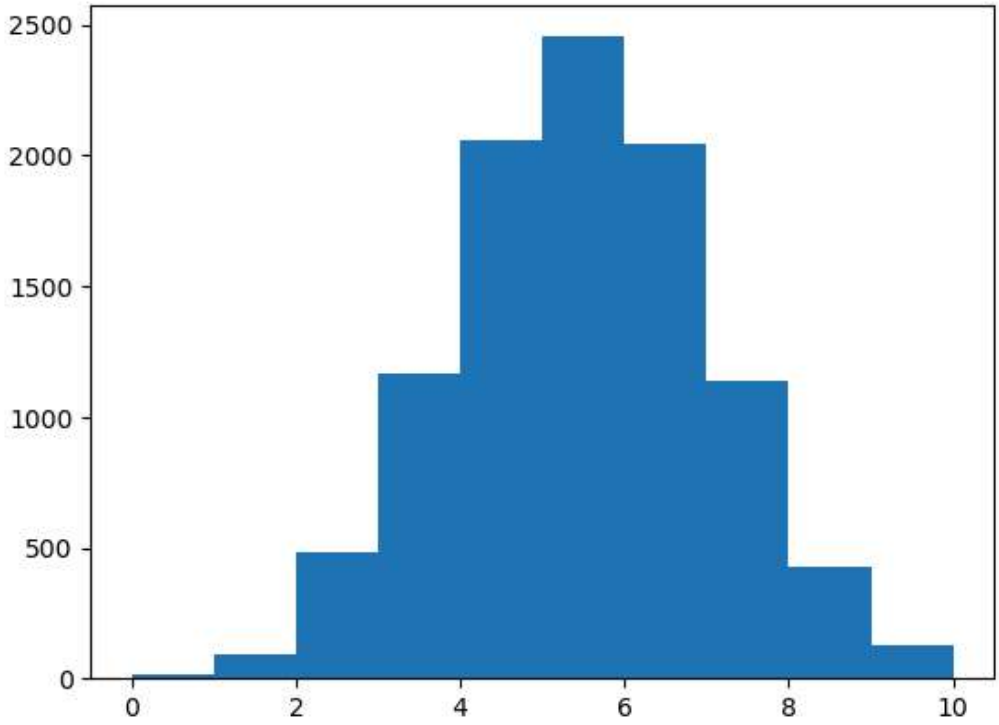
In [23]:
```
plt.fill_between(x=np.arange(-4,-1,0.01),
                 y1=stats.norm.pdf(np.arange(-4,-1,0.01)),
                 facecolor='red',
                 alpha=0.35)

plt.fill_between(x=np.arange(1,4,0.01),
                 y1=stats.norm.pdf(np.arange(1,4,0.01)),
                 facecolor='red',
                 alpha=0.35)

plt.fill_between(x=np.arange(-1,1,0.01),
                 y1=stats.norm.pdf(np.arange(-1,1,0.01)),
                 facecolor='blue',
                 alpha=0.35)
```

Out[23]:
```
<matplotlib.collections.PolyCollection at 0x24492ec5e50>
```



In [28]:
```
d = stats.binom.rvs(n=10,p=0.5,size=10000)
df = pd.DataFrame(d)
plt.hist(df[0])
```

Out[28]:
```
(array([  13.,   90.,  484., 1169., 2061., 2453., 2045., 1135.,  426.,
         124.]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]),
 <BarContainer object of 10 artists>)
```



In [ ]:
```
population                          sample
- parameters                        - statistics
      |______measurements_____________|

- size infinite                     finite collection (countable)
  (not countable )

1.Null Hypothesis Ho          -------> there is no difference
2.Alternate Hypothesis H1     -------> not equal ; 1Set is greater than 2set ; 2One is grea
  |-> critical region

3. Test
   ZTest tTest .,fTest, chiSquare Test ANOVA Test
    |      |_ <30
    >30
```

In [ ]:

```
Gender | dept |Age | Weight | Height
M      | sales| 45 | 66     | 160cm
F      |  prod| 55
M      | HR
F      | sales
-------------------------------------//sample data(sampling)

Gender - One Categorical data

|    | |
| || | |
|-----------
   M  F

Two categorical data
- chiSquare Test


Only one continuous variable
|
TTest


Two categorical data(more than two different categorical data(ex: sales,prod,HR,QA..))
ANOVA Test
```

In [ ]:
```
EDA  Exploratory Data Analysis
====
 |-> To understand - structure of a dataset
     -------------------------------------
       |->Summarize the key characterstics - Visualization

 1. Univariate
 2. Bivariate
 3. Mulitvariate

[marketingCost][productName] [productCost]

1. Univariate
[marketingCost]
[productName]
[productCost]

Visualization: plot,scatter,bar,pie


2. Bivariate
[productName][productCost]  => Visualization : bar ; pie
  pA            1000
  pB            1200
  pC             650

[marketingCost] [productCost] => Visualization: plot ; scatter
  5000            1000

3. Multivariate
 [marketingCost] [productName] [productCost] => Visualization: Heatmap (import seaborn as
                                                sns.heatmap(data,annot=True)
```

In [30]:
```
# load all the libs
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [33]:
```
load_data = pd.read_csv('C:\\Users\\Raja\\Downloads\\used_cars_data.csv')
load_data.shape
```

Out[33]: (7253, 14)

In [34]:
```
df = load_data.sample(n=35) # Appling -> random sampling
df.shape
```

Out[34]: (35, 14)

In [35]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 35 entries, 2476 to 3791
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   S.No.              35 non-null     int64  
 1   Name               35 non-null     object 
 2   Location           35 non-null     object 
 3   Year               35 non-null     int64  
 4   Kilometers_Driven  35 non-null     int64  
 5   Fuel_Type          35 non-null     object 
 6   Transmission       35 non-null     object 
 7   Owner_Type         35 non-null     object 
 8   Mileage            35 non-null     object 
 9   Engine             35 non-null     object 
 10  Power              35 non-null     object 
 11  Seats              35 non-null     float64
 12  New_Price          6 non-null      object 
 13  Price              30 non-null     float64
dtypes: float64(2), int64(3), object(9)
memory usage: 4.1+ KB
```

In [36]:
```
df.columns
```

Out[36]:
```
Index(['S.No.', 'Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Seats',
       'New_Price', 'Price'],
      dtype='object')
```

In [37]:
```
# dropped unwanted columns
df.drop(columns=['S.No.','Seats','Owner_Type','Owner_Type','Power'],inplace=True)
```

In [38]:
```
df.columns
```

Out[38]:
```
Index(['Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Mileage', 'Engine', 'New_Price', 'Price'],
      dtype='object')
```

In [39]:
```
df.shape
```

Out[39]:
```
(35, 10)
```

In [40]:
```
# Check any duplicate items are exists
df.duplicated().any()
```

Out[40]:
```
False
```

In [41]:
```
# another way
df.duplicated().sum()
```

Out[41]:
```
0
```

In [42]:
```
# check any empty values are exists
df.isnull().sum()
```

Out[42]:
```
Name                 0
Location             0
Year                 0
Kilometers_Driven    0
Fuel_Type            0
Transmission         0
Mileage              0
Engine               0
New_Price           29
Price                5
dtype: int64
```

In [43]:
```
df['New_Price']
```

Out[43]:
```
2476           NaN
6000           NaN
590            NaN
1815           NaN
4193           NaN
4722           NaN
2101           NaN
6568           NaN
5883           NaN
5518    39.27 Lakh
2090     6.08 Lakh
1947           NaN
6588     4.75 Lakh
6896           NaN
4992           NaN
2691           NaN
2812     7.14 Lakh
1426           NaN
6562           NaN
2582           NaN
4857           NaN
899            NaN
4494           NaN
4742           NaN
3455           NaN
481            NaN
6396     29.9 Lakh
1382           NaN
5507     7.91 Lakh
2211           NaN
3946           NaN
2790           NaN
4611           NaN
3717           NaN
3791           NaN
Name: New_Price, dtype: object
```

In [45]:
```
# replace NaN value
df.fillna(value=df.isnull().sum()/len(df)*100,inplace=True)
```

In [46]:
```
# check any empty values are exists
df.isnull().sum()
```

Out[46]:
```
Name                0
Location            0
Year                0
Kilometers_Driven   0
Fuel_Type           0
Transmission        0
Mileage             0
Engine              0
New_Price           0
Price               0
dtype: int64
```

In [47]:
```
df.columns
```

Out[47]:
```
Index(['Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Mileage', 'Engine', 'New_Price', 'Price'],
      dtype='object')
```

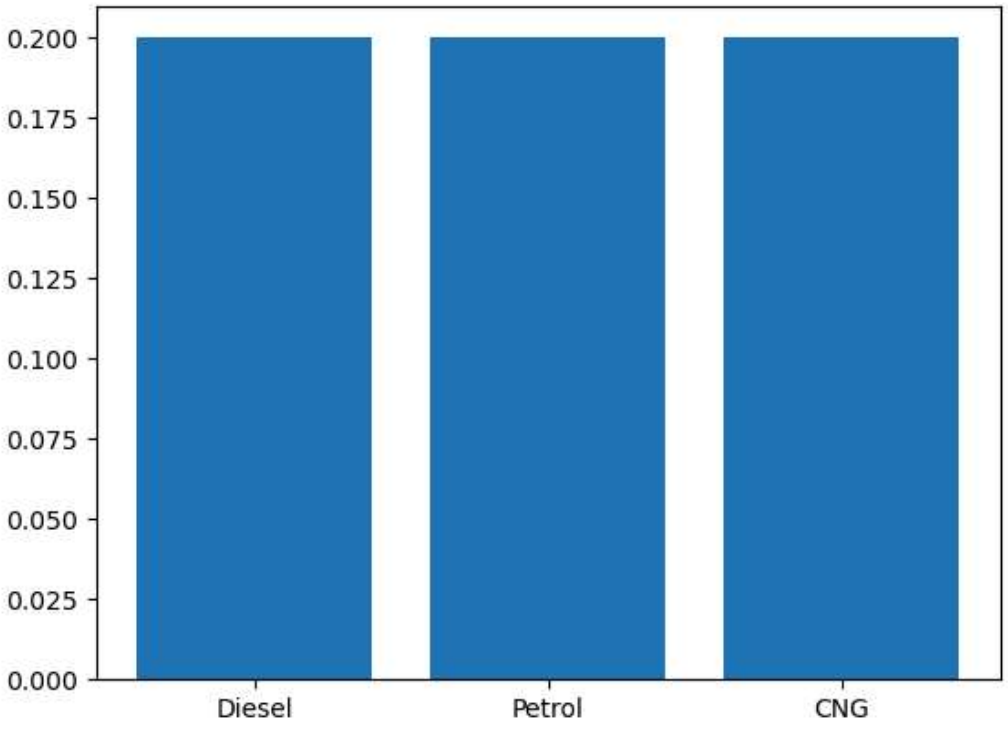
In [48]:
```
# renamed the columns
df.rename(columns={'Kilometers_Driven':'km','Transmission':'trans'},inplace=True)
```

In [49]:
```
df.columns
```

Out[49]:
```
Index(['Name', 'Location', 'Year', 'km', 'Fuel_Type', 'trans', 'Mileage',
       'Engine', 'New_Price', 'Price'],
      dtype='object')
```

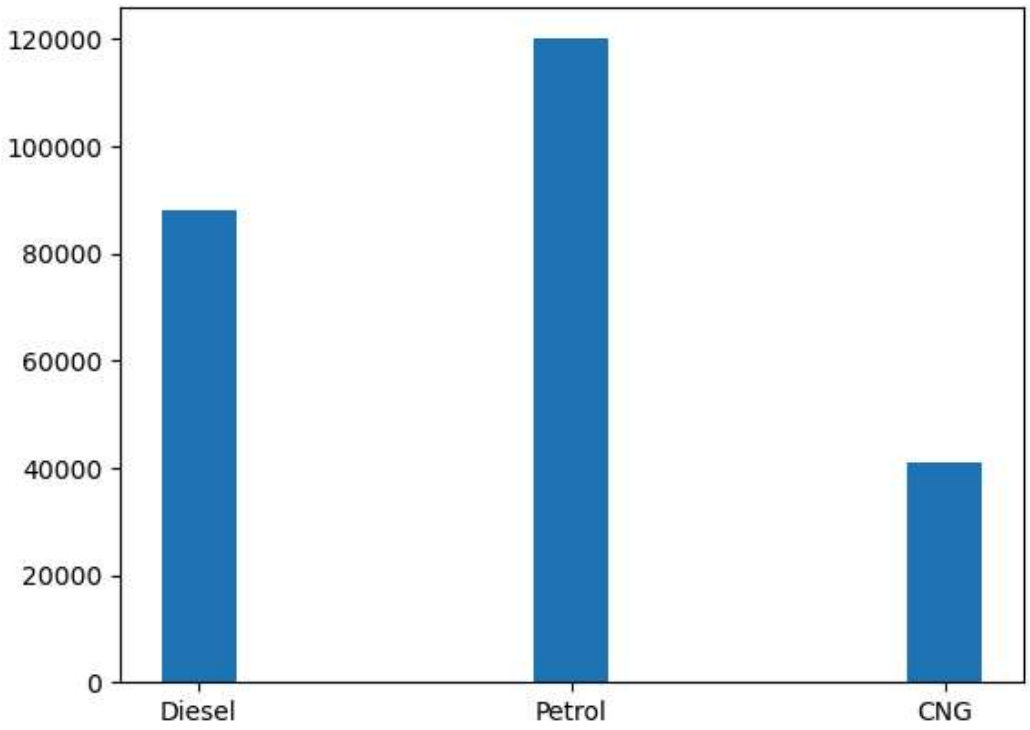
In [52]:
```
plt.bar(df['Fuel_Type'],height=0.2)
```

Out[52]:
```
<BarContainer object of 35 artists>
```



In [53]:
```
plt.bar(df['Fuel_Type'],df['km'],width=0.2)
```

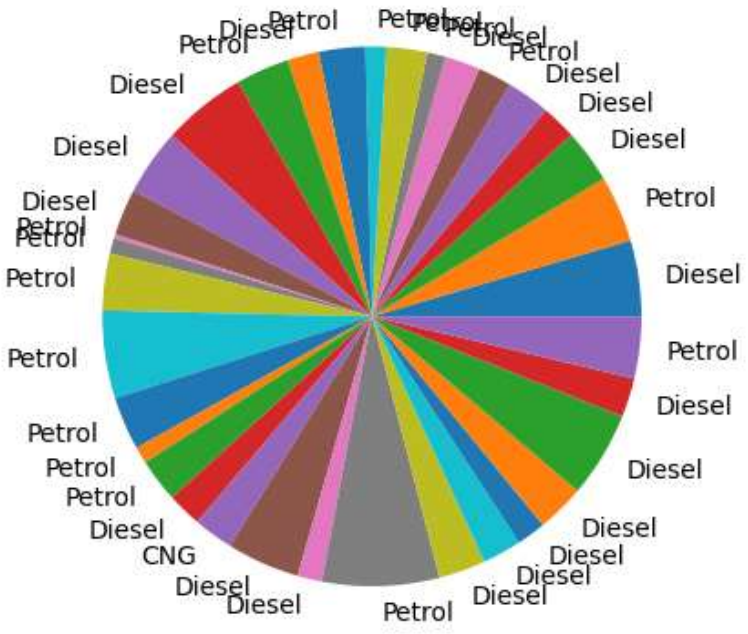
Out[53]:
```
<BarContainer object of 35 artists>
```



In [54]:
```
plt.pie(df['km'],labels=df['Fuel_Type'])
```

```
Out[54]: ([<matplotlib.patches.Wedge at 0x24494e796d0>,
           <matplotlib.patches.Wedge at 0x24494e50b90>,
           <matplotlib.patches.Wedge at 0x24494ecc590>,
           <matplotlib.patches.Wedge at 0x24494eb7950>,
           <matplotlib.patches.Wedge at 0x24494ef7590>,
           <matplotlib.patches.Wedge at 0x24494f01190>,
           <matplotlib.patches.Wedge at 0x24494f0be90>,
           <matplotlib.patches.Wedge at 0x24493f5ac50>,
           <matplotlib.patches.Wedge at 0x24494ef7a10>,
           <matplotlib.patches.Wedge at 0x24494f18f90>,
           <matplotlib.patches.Wedge at 0x24494f22750>,
           <matplotlib.patches.Wedge at 0x24494f2bf50>,
           <matplotlib.patches.Wedge at 0x24494f35850>,
           <matplotlib.patches.Wedge at 0x24494f3e7d0>,
           <matplotlib.patches.Wedge at 0x24494f54110>,
           <matplotlib.patches.Wedge at 0x24494f5e150>,
           <matplotlib.patches.Wedge at 0x24494f68510>,
           <matplotlib.patches.Wedge at 0x24494f72190>,
           <matplotlib.patches.Wedge at 0x24494f77450>,
           <matplotlib.patches.Wedge at 0x24494f88c90>,
           <matplotlib.patches.Wedge at 0x24494f962d0>,
           <matplotlib.patches.Wedge at 0x24494f9fe90>,
           <matplotlib.patches.Wedge at 0x24494fa9650>,
           <matplotlib.patches.Wedge at 0x24494fbb250>,
           <matplotlib.patches.Wedge at 0x24494fccf90>,
           <matplotlib.patches.Wedge at 0x24494fd3010>,
           <matplotlib.patches.Wedge at 0x24494fe41d0>,
           <matplotlib.patches.Wedge at 0x24494fee990>,
           <matplotlib.patches.Wedge at 0x24494ff7fd0>,
           <matplotlib.patches.Wedge at 0x24495001b50>,
           <matplotlib.patches.Wedge at 0x2449500f550>,
           <matplotlib.patches.Wedge at 0x2449501cf90>,
           <matplotlib.patches.Wedge at 0x24495026490>,
           <matplotlib.patches.Wedge at 0x24495034590>,
           <matplotlib.patches.Wedge at 0x2449503e290>],
          [Text(1.0887855648339941, 0.15667161136951543, 'Diesel'),
           Text(1.0084956446954683, 0.4392454150361417, 'Petrol'),
           Text(0.8844380830696236, 0.6540407305482816, 'Diesel'),
           Text(0.7632786687966783, 0.7920894354553473, 'Diesel'),
           Text(0.6384999838259299, 0.8957219270813278, 'Diesel'),
           Text(0.5016374503743608, 0.9789585631587839, 'Petrol'),
           Text(0.3700508120930279, 1.0358872508479342, 'Diesel'),
           Text(0.2639592554802683, 1.0678602490243292, 'Petrol'),
           Text(0.14317677987341162, 1.0906422005887544, 'Petrol'),
           Text(0.01605248319112782, 1.099882865483138, 'Petrol'),
           Text(-0.12272228209358019, 1.093132764799292, 'Petrol'),
           Text(-0.2815037132400511, 1.063369954170261, 'Diesel'),
           Text(-0.44745102684076365, 1.0048818729478337, 'Petrol'),
           Text(-0.6896638968720437, 0.8569502373832844, 'Diesel'),
           Text(-0.9014655236441053, 0.6303649020060201, 'Diesel'),
           Text(-1.014390705635765, 0.4254544585731532, 'Diesel'),
           Text(-1.0503214301039396, 0.3268407769303196, 'Petrol'),
           Text(-1.0615962287257341, 0.2881205427512919, 'Petrol'),
           Text(-1.0908493621662865, 0.14158979151551102, 'Petrol'),
           Text(-1.0888019662123924, -0.1565575880372092, 'Petrol'),
           Text(-1.009447056941803, -0.43705450373097937, 'Petrol'),
           Text(-0.9433222206933222, -0.5658119722542281, 'Petrol'),
           Text(-0.8730936567417634, -0.6691094578298052, 'Petrol'),
           Text(-0.7656833714187607, -0.7897651389703144, 'Diesel'),
           Text(-0.6480564698286481, -0.8888322743427078, 'CNG'),
           Text(-0.4454176582234249, -1.0057848227840587, 'Diesel'),
           Text(-0.25419072007550503, -1.0702275822587906, 'Diesel'),
           Text(0.036496569422903184, -1.0993943789288534, 'Petrol'),
```

```
Text(0.36871458176908994, -1.0363636220896821, 'Diesel'),
Text(0.5308957707517064, -0.9634052525276949, 'Diesel'),
Text(0.6502229319966807, -0.8872486340962381, 'Diesel'),
Text(0.7719542087896671, -0.783636841612184, 'Diesel'),
Text(0.9429665365703049, -0.5664045470408974, 'Diesel'),
Text(1.0489797478371516, -0.33112156170733736, 'Diesel'),
Text(1.0925897277074523, -0.12746641482427887, 'Petrol')])
```



In [58]:
```
df.columns
```

Out[58]:
```
Index(['Name', 'Location', 'Year', 'km', 'Fuel_Type', 'trans', 'Mileage',
       'Engine', 'New_Price', 'Price'],
      dtype='object')
```

In [69]:
```
help(sns.pairplot)
```

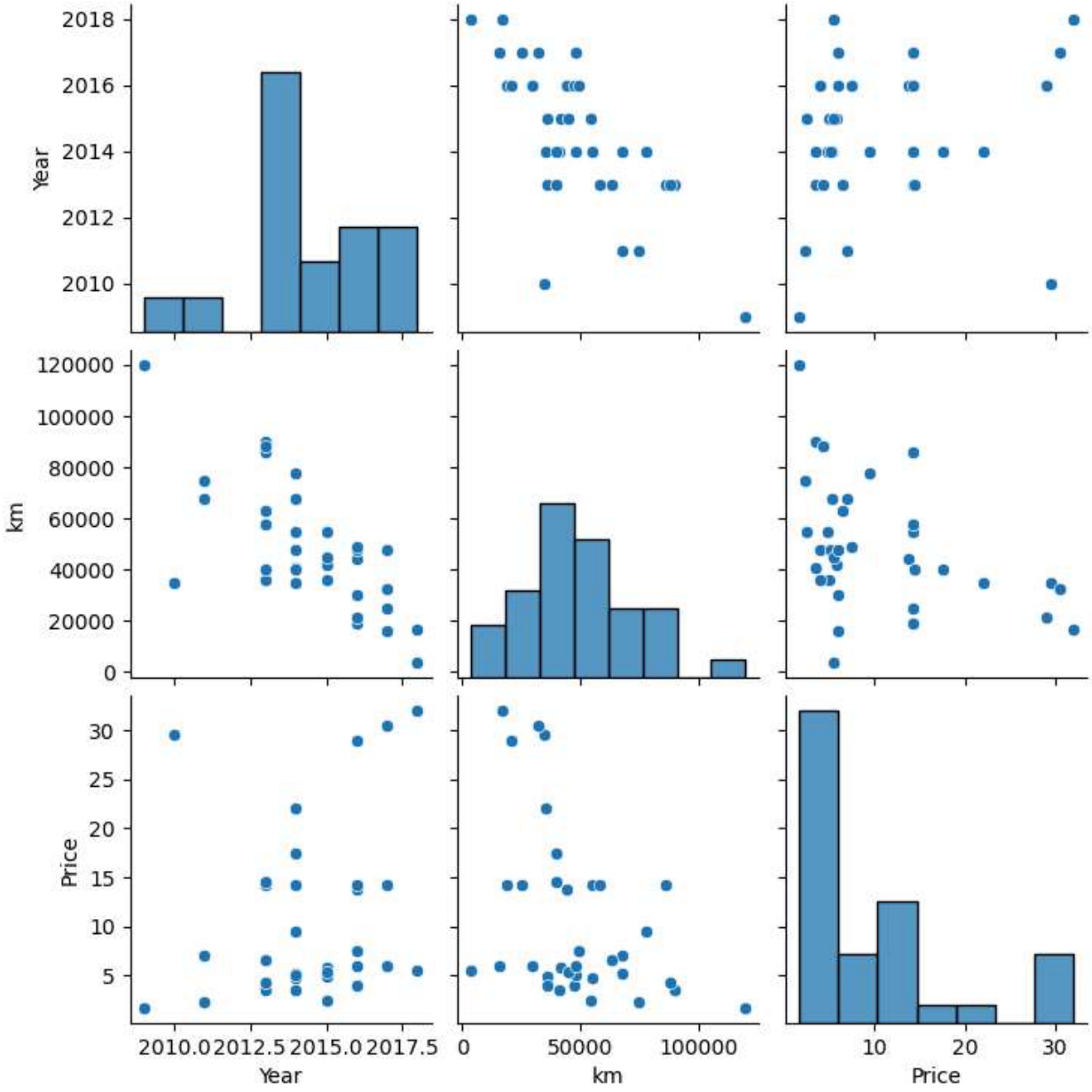
```
Help on function pairplot in module seaborn.axisgrid:

pairplot(data, *, hue=None, hue_order=None, palette=None, vars=None, x_vars=None, y_vars=
None, kind='scatter', diag_kind='auto', markers=None, height=2.5, aspect=1, corner=False,
dropna=False, plot_kws=None, diag_kws=None, grid_kws=None, size=None)
    Plot pairwise relationships in a dataset.

    By default, this function will create a grid of Axes such that each numeric
    variable in ``data`` will by shared across the y-axes across a single row and
    the x-axes across a single column. The diagonal plots are treated
    differently: a univariate distribution plot is drawn to show the marginal
    distribution of the data in each column.

    It is also possible to show a subset of variables or plot different
    variables on the rows and columns.

    This is a high-level interface for :class:`PairGrid` that is intended to
    make it easy to draw a few common styles. You should use :class:`PairGrid`
    directly if you need more flexibility.

    Parameters
    ----------
    data : `pandas.DataFrame`
        Tidy (long-form) dataframe where each column is a variable and
        each row is an observation.
    hue : name of variable in ``data``
        Variable in ``data`` to map plot aspects to different colors.
    hue_order : list of strings
        Order for the levels of the hue variable in the palette
    palette : dict or seaborn color palette
        Set of colors for mapping the ``hue`` variable. If a dict, keys
        should be values  in the ``hue`` variable.
    vars : list of variable names
        Variables within ``data`` to use, otherwise use every column with
        a numeric datatype.
    {x, y}_vars : lists of variable names
        Variables within ``data`` to use separately for the rows and
        columns of the figure; i.e. to make a non-square plot.
    kind : {'scatter', 'kde', 'hist', 'reg'}
        Kind of plot to make.
    diag_kind : {'auto', 'hist', 'kde', None}
        Kind of plot for the diagonal subplots. If 'auto', choose based on
        whether or not ``hue`` is used.
    markers : single matplotlib marker code or list
        Either the marker to use for all scatterplot points or a list of markers
        with a length the same as the number of levels in the hue variable so that
        differently colored points will also have different scatterplot
        markers.
    height : scalar
        Height (in inches) of each facet.
    aspect : scalar
        Aspect * height gives the width (in inches) of each facet.
    corner : bool
        If True, don't add axes to the upper (off-diagonal) triangle of the
        grid, making this a "corner" plot.
    dropna : boolean
        Drop missing values from the data before plotting.
    {plot, diag, grid}_kws : dicts
        Dictionaries of keyword arguments. ``plot_kws`` are passed to the
        bivariate plotting function, ``diag_kws`` are passed to the univariate
        plotting function, and ``grid_kws`` are passed to the :class:`PairGrid`
        constructor.
```

```
Returns
-------
grid : :class:`PairGrid`
    Returns the underlying :class:`PairGrid` instance for further tweaking.

See Also
--------
PairGrid : Subplot grid for more flexible plotting of pairwise relationships.
JointGrid : Grid for plotting joint and marginal distributions of two variables.

Examples
--------

.. include:: ../docstrings/pairplot.rst
```

In [71]:
```
sns.pairplot(data=df)
```

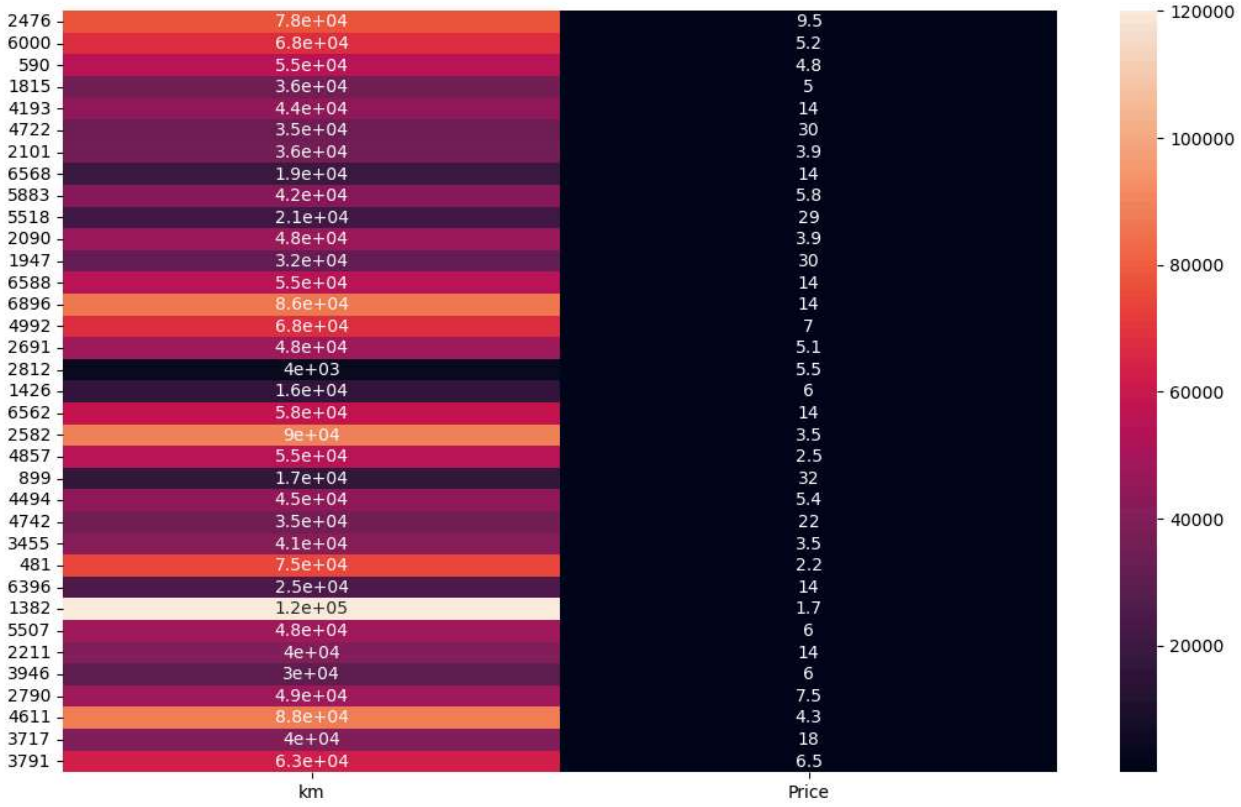
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figu
re layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[71]:
```
<seaborn.axisgrid.PairGrid at 0x2449a987910>
```



In [84]:
```
plt.figure(figsize=(13,8))
sns.heatmap(df[['km','Price']],annot=True)
```

Out[84]: <Axes: >



In [87]:
```
df.columns
```

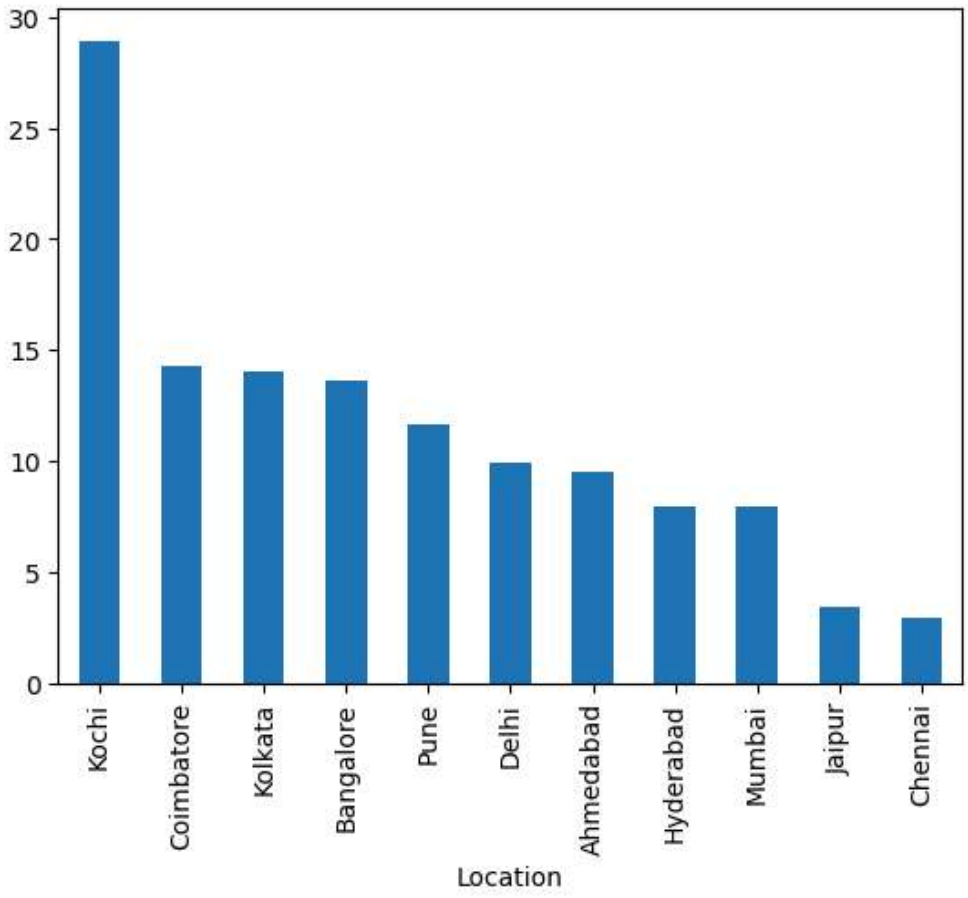
Out[87]:
```
Index(['Name', 'Location', 'Year', 'km', 'Fuel_Type', 'trans', 'Mileage',
       'Engine', 'New_Price', 'Price'],
      dtype='object')
```

In [89]:
```
df.groupby('Location')['Price'].mean()
```

Out[89]:
```
Location
Ahmedabad      9.517857
Bangalore     13.625000
Chennai        2.900000
Coimbatore    14.300000
Delhi          9.933333
Hyderabad      7.933929
Jaipur         3.415000
Kochi         28.950000
Kolkata       14.057143
Mumbai         7.918571
Pune          11.672500
Name: Price, dtype: float64
```

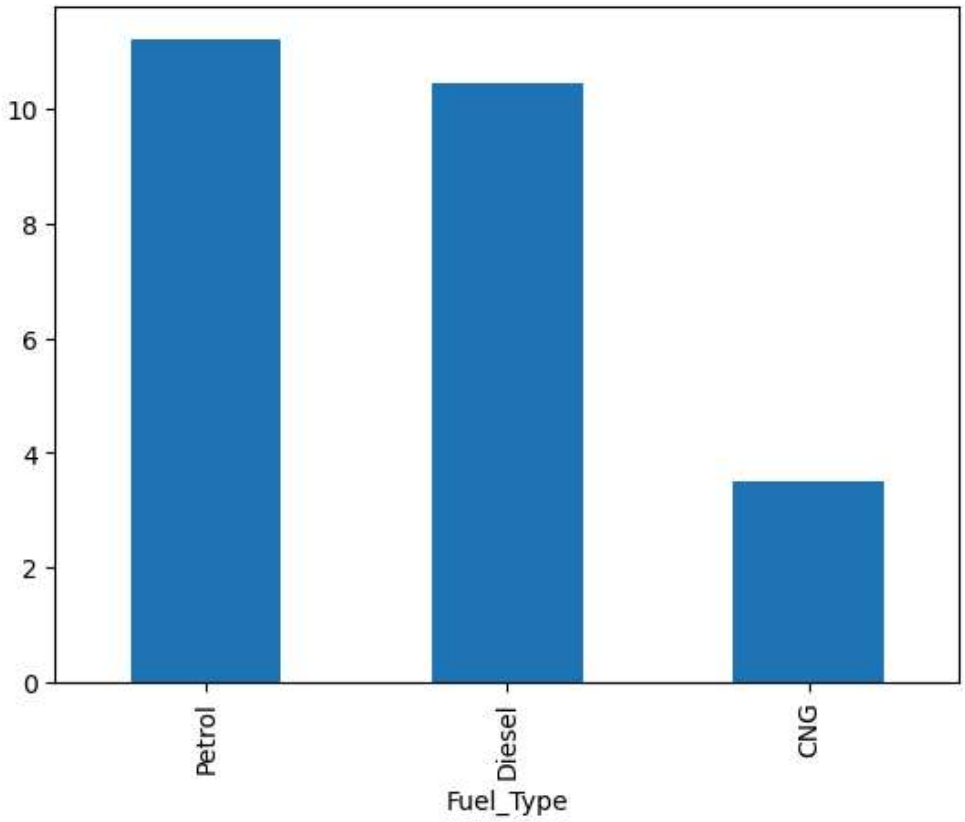
In [92]:
```
df.groupby('Location')['Price'].mean().sort_values(ascending=False).plot.bar()
```

Out[92]: <Axes: xlabel='Location'>



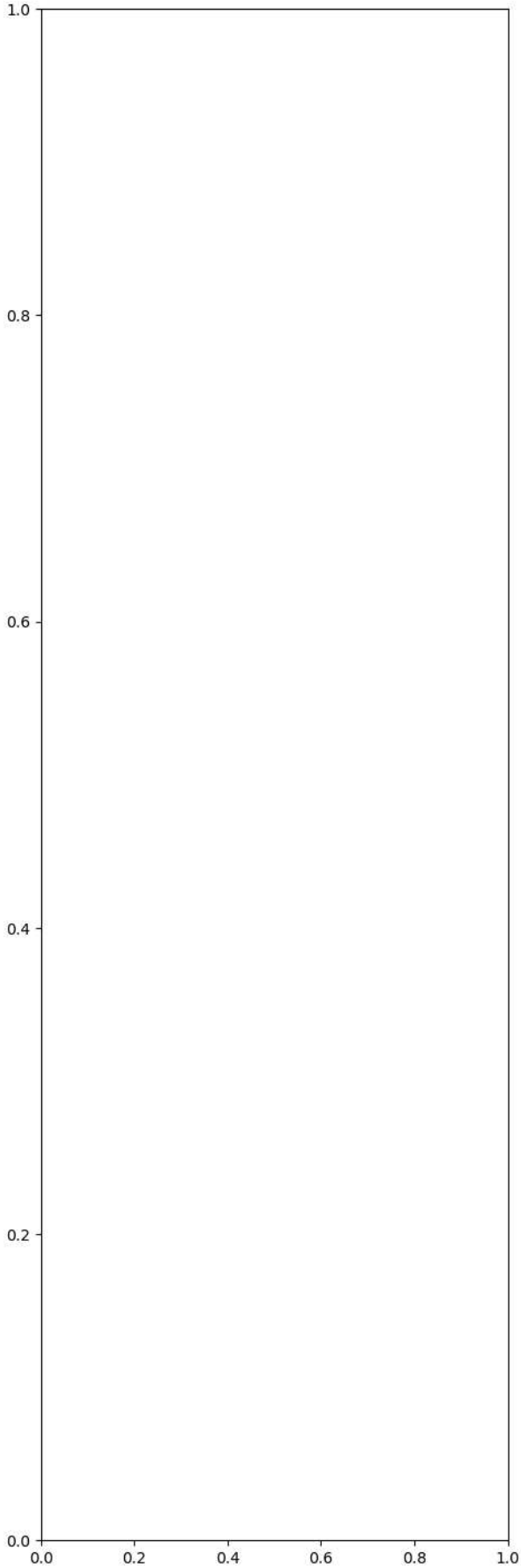
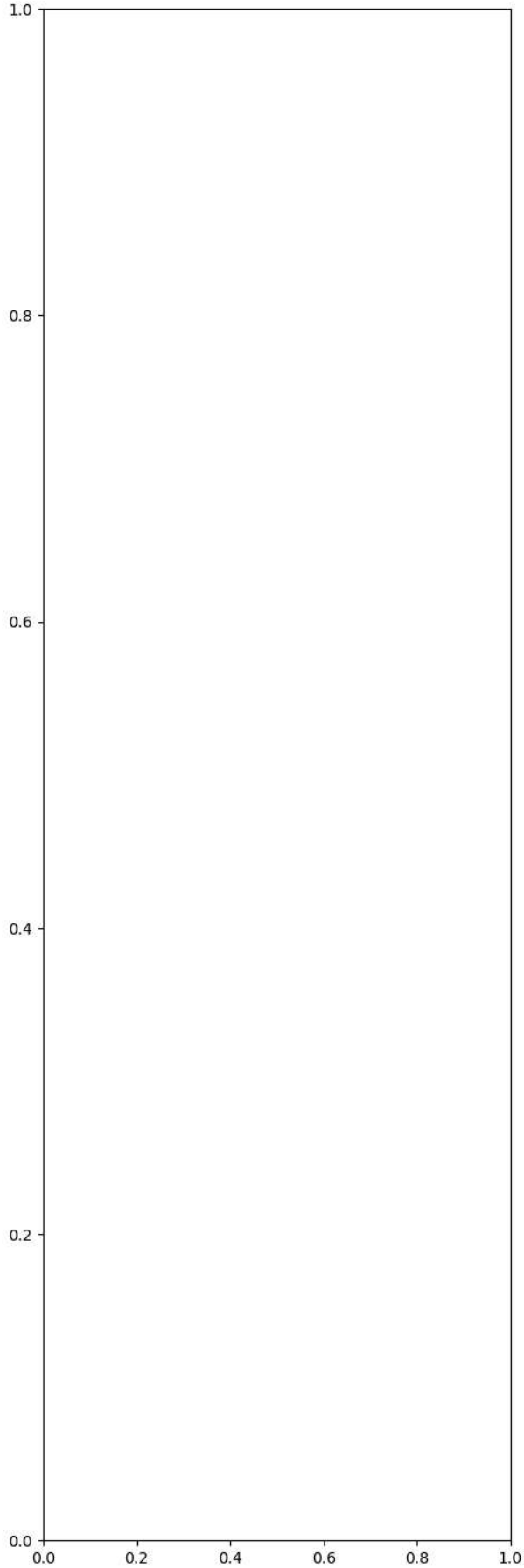
In [93]:
```
df.groupby('Fuel_Type')['Price'].mean().sort_values(ascending=False).plot.bar()
```

Out[93]: <Axes: xlabel='Fuel_Type'>



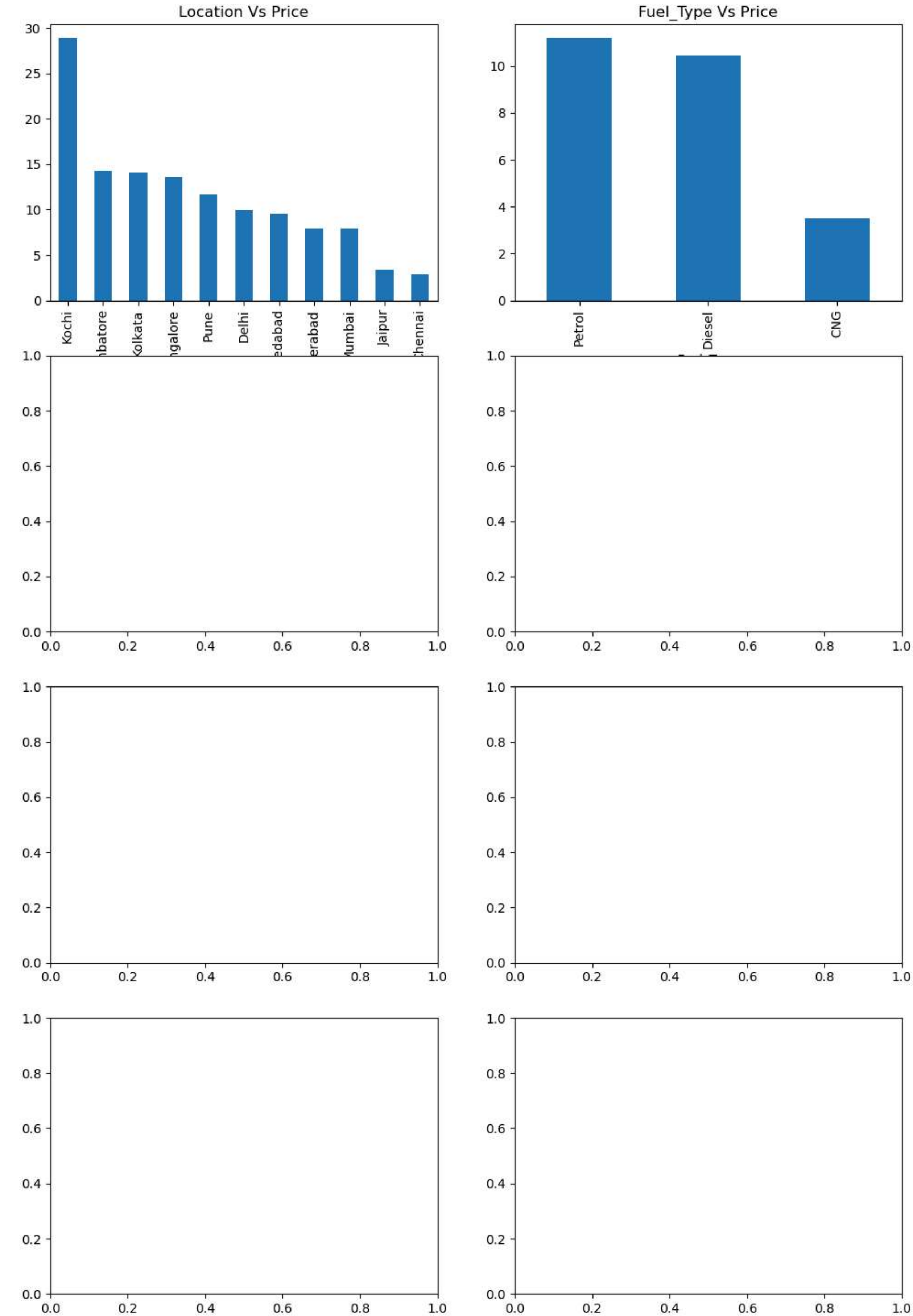
In [96]:
```
plt.subplots(1,2,figsize=(12,18))
```

Out[96]:
```
(<Figure size 1200x1800 with 2 Axes>,
 array([<Axes: >, <Axes: >], dtype=object))
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

In [107]:
```
figobj,axindex = plt.subplots(4,2,figsize=(12,18))
df.groupby('Location')['Price'].mean().sort_values(ascending=False).plot.bar(ax=axindex[0]
axindex[0][0].set_title("Location Vs Price")
df.groupby('Fuel_Type')['Price'].mean().sort_values(ascending=False).plot.bar(ax=axindex[0
axindex[0][1].set_title("Fuel_Type Vs Price")
```

Out[107]:
```
Text(0.5, 1.0, 'Fuel_Type Vs Price')
```


Location Vs Price
Kochi
Kolkata
Pune
Delhi
Mumbai
Jaipur
Fuel_Type Vs Price
Petrol
Diesel
CNG

In [108]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 35 entries, 2476 to 3791
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Name        35 non-null     object 
 1   Location    35 non-null     object 
 2   Year        35 non-null     int64  
 3   km          35 non-null     int64  
 4   Fuel_Type   35 non-null     object 
 5   trans       35 non-null     object 
 6   Mileage     35 non-null     object 
 7   Engine      35 non-null     object 
 8   New_Price   35 non-null     object 
 9   Price       35 non-null     float64
dtypes: float64(1), int64(2), object(7)
memory usage: 4.1+ KB
```

In [110]:
```
df.dtypes
```

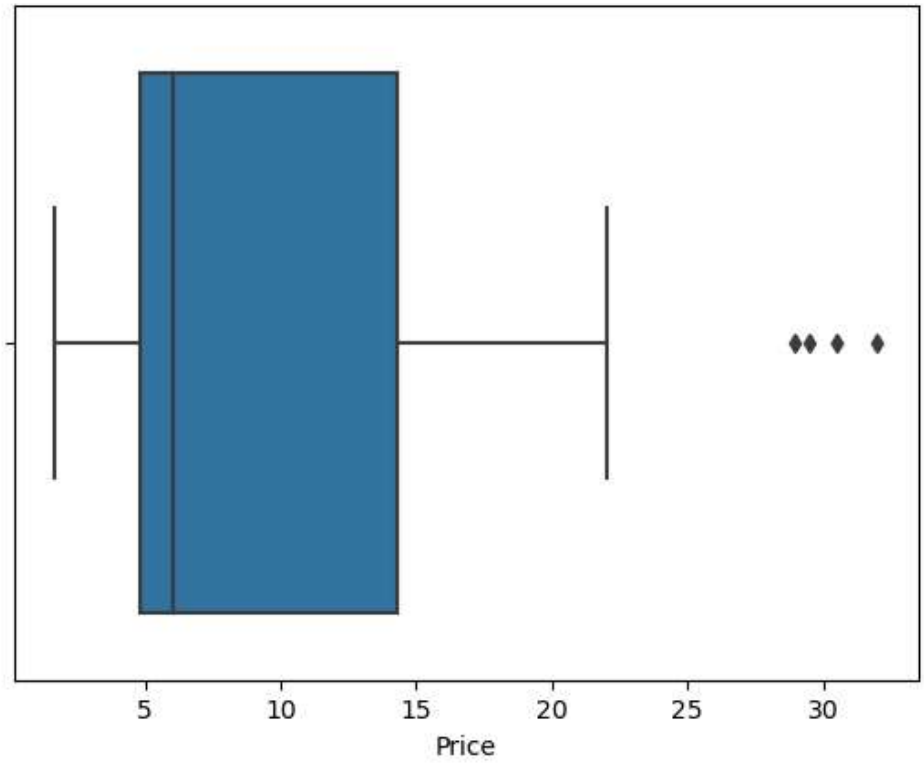
Out[110]:
```
Name          object
Location      object
Year           int64
km             int64
Fuel_Type     object
trans         object
Mileage       object
Engine        object
New_Price     object
Price        float64
dtype: object
```

In [113]:
```
df['Location'].value_counts()
```

Out[113]:
```
Location
Mumbai        6
Kolkata       5
Hyderabad     4
Coimbatore    4
Pune          4
Delhi         3
Bangalore     2
Ahmedabad     2
Chennai       2
Jaipur        2
Kochi         1
Name: count, dtype: int64
```

In [116]:
```
sns.boxplot(x=df['Price'])
```

Out[116]:
```
<Axes: xlabel='Price'>
```



In [125]:
```
d = pd.DataFrame({"K1":[10,12,13,100,15,16,10,11,300,14]})
Q1 = d['K1'].quantile(0.25)
Q3 = d['K1'].quantile(0.75)

IQR = Q3-Q1

lb = Q1 - 1.5 * IQR
hb = Q3 + 1.5 * IQR
r = d[(d['K1'] >= lb) & (d['K1'] <= hb)]
print(r)
```

```
   K1
0  10
1  12
2  13
4  15
5  16
6  10
7  11
9  14
```

In [120]:
```
help(d.quantile)
```

```
Help on method quantile in module pandas.core.frame:

quantile(q: 'float | AnyArrayLike | Sequence[float]' = 0.5, axis: 'Axis' = 0, numeric_o
nly: 'bool' = False, interpolation: 'QuantileInterpolation' = 'linear', method: "Litera
l['single', 'table']" = 'single') -> 'Series | DataFrame' method of pandas.core.frame.D
ataFrame instance
    Return values at the given quantile over requested axis.

    Parameters
    ----------
    q : float or array-like, default 0.5 (50% quantile)
        Value between 0 <= q <= 1, the quantile(s) to compute.
    axis : {0 or 'index', 1 or 'columns'}, default 0
        Equals 0 or 'index' for row-wise, 1 or 'columns' for column-wise.
    numeric_only : bool, default False
        Include only `float`, `int` or `boolean` data.

        .. versionchanged:: 2.0.0
            The default value of ``numeric_only`` is now ``False``.
```

In [126]:
```
help(stats.zscore)
```

```
Help on function zscore in module scipy.stats._stats_py:

zscore(a, axis=0, ddof=0, nan_policy='propagate')
    Compute the z score.

    Compute the z score of each value in the sample, relative to the
    sample mean and standard deviation.

    Parameters
    ----------
    a : array_like
        An array like object containing the sample data.
    axis : int or None, optional
        Axis along which to operate. Default is 0. If None, compute over
        the whole array `a`.
    ddof : int, optional
        Degrees of freedom correction in the calculation of the
        standard deviation. Default is 0.
    nan_policy : {'propagate', 'raise', 'omit'}, optional
        Defines how to handle when input contains nan. 'propagate' returns nan,
        'raise' throws an error, 'omit' performs the calculations ignoring nan
        values. Default is 'propagate'.  Note that when the value is 'omit',
        nans in the input also propagate to the output, but they do not affect
        the z-scores computed for the non-nan values.

    Returns
    -------
    zscore : array_like
        The z-scores, standardized by mean and standard deviation of
        input array `a`.

    See Also
    --------
    numpy.mean : Arithmetic average
    numpy.std : Arithmetic standard deviation
    scipy.stats.gzscore : Geometric standard score

    Notes
    -----
    This function preserves ndarray subclasses, and works also with
    matrices and masked arrays (it uses `asanyarray` instead of
    `asarray` for parameters).

    References
    ----------
    .. [1] "Standard score", *Wikipedia*,
           https://en.wikipedia.org/wiki/Standard_score. (https://en.wikipedia.org/wiki/S
tandard_score.)
    .. [2] Huck, S. W., Cross, T. L., Clark, S. B, "Overcoming misconceptions
           about Z-scores", Teaching Statistics, vol. 8, pp. 38-40, 1986

    Examples
    --------
    >>> import numpy as np
    >>> a = np.array([ 0.7972,  0.0767,  0.4383,  0.7866,  0.8091,
    ...                0.1954,  0.6307,  0.6599,  0.1065,  0.0508])
    >>> from scipy import stats
    >>> stats.zscore(a)
    array([ 1.1273, -1.247 , -0.0552,  1.0923,  1.1664, -0.8559,  0.5786,
            0.6748, -1.1488, -1.3324])

    Computing along a specified axis, using n-1 degrees of freedom
    (``ddof=1``) to calculate the standard deviation:
```

```
>>> b = np.array([[ 0.3148,  0.0478,  0.6243,  0.4608],
...               [ 0.7149,  0.0775,  0.6072,  0.9656],
...               [ 0.6341,  0.1403,  0.9759,  0.4064],
...               [ 0.5918,  0.6948,  0.904 ,  0.3721],
...               [ 0.0921,  0.2481,  0.1188,  0.1366]])
>>> stats.zscore(b, axis=1, ddof=1)
array([[-0.19264823, -1.28415119,  1.07259584,  0.40420358],
       [ 0.33048416, -1.37380874,  0.04251374,  1.00081084],
       [ 0.26796377, -1.12598418,  1.23283094, -0.37481053],
       [-0.22095197,  0.24468594,  1.19042819, -1.21416216],
       [-0.82780366,  1.4457416 , -0.43867764, -0.1792603 ]])

An example with `nan_policy='omit'`:

>>> x = np.array([[25.11, 30.10, np.nan, 32.02, 43.15],
...               [14.95, 16.06, 121.25, 94.35, 29.81]])
>>> stats.zscore(x, axis=1, nan_policy='omit')
array([[-1.13490897, -0.37830299,         nan, -0.08718406,  1.60039602],
       [-0.91611681, -0.89090508,  1.4983032 ,  0.88731639, -0.5785977 ]])
```

In [131]:

```
from scipy.stats import zscore
d = pd.DataFrame({"K1":[10,12,13,100,15,16,10,11,300,14]})

d['zscore'] = zscore(d['K1'])
d['zscore']

d[abs(d['zscore'] < 3)]
```

Out[131]:

	K1	zscore
0	10	-0.459342
1	12	-0.436432
2	13	-0.424977
3	100	0.571600
4	15	-0.402067
5	16	-0.390612
6	10	-0.459342
7	11	-0.447887
8	300	2.862581
9	14	-0.413522

In [135]:
```
d = pd.DataFrame({"K1":[10,12,13,100,15,16,10,11,300,14]})

L_p = d['K1'].quantile(0.01)
U_p = d['K1'].quantile(0.75)

d[(d['K1'] >= L_p) & (d['K1'] <=U_p)]
```

Out[135]:

	K1
0	10
1	12
2	13
4	15
6	10
7	11
9	14

In [136]:
```
df.describe()
```

Out[136]:

	Year	km	Price
count	35.000000	35.000000	35.000000
mean	2014.400000	48860.342857	10.602531
std	2.144761	24440.295882	8.690271
min	2009.000000	3958.000000	1.690000
25%	2013.000000	35082.500000	4.850000
50%	2014.000000	44846.000000	6.000000
75%	2016.000000	60610.000000	14.285714
max	2018.000000	120000.000000	32.000000

In [ ]:
```
General programming
--------------------
user ->input -> m/c -> output
                 ---
            Vs
Machine Learning
--------------------
user -> input & output -> m/c ->Algorithm
        ..............
```

In [ ]:
```
 person -> 100 books reading about dog details
           ===================================

    Q1 -> person  can  answer

          person -> 100 books + 5 books + .....
             |___________________| ............

1984 - Q: who is pm of india A: .....
           -----------------
2024 - Q:  ....              A: ......
```

In [ ]:
```
ML
 |->Supervised Learning
       |
       |-> Regression & Classification
 |-> Unsupervised Learning
       |-> Clustering & Association
 |-> Reinforcement Learning
```

In [ ]:
```
Regression - searches relationship among the variables
y=mx+c

      |            .
o/p   |       .
      |
      |
      ----------------
      |    input x1 x2 x3
```

In [138]:
```
from scipy import stats

x = [6,8,9,10,12,3,18,3,5,12,13,9,7,50]
y = [99,66,87,88,120,76,95,78,67,86,66,45,66,100]

stats.linregress(x,y)
```

Out[138]:
```
LinregressResult(slope=0.5948932774785557, intercept=74.34590065828846, rvalue=0.36672664
79140301, pvalue=0.19714135683995782, stderr=0.4356547035720552, intercept_stderr=7.11579
933508397)
```

In [139]:
```
slope,intercept,r,p,stderr = stats.linregress(x,y)
```

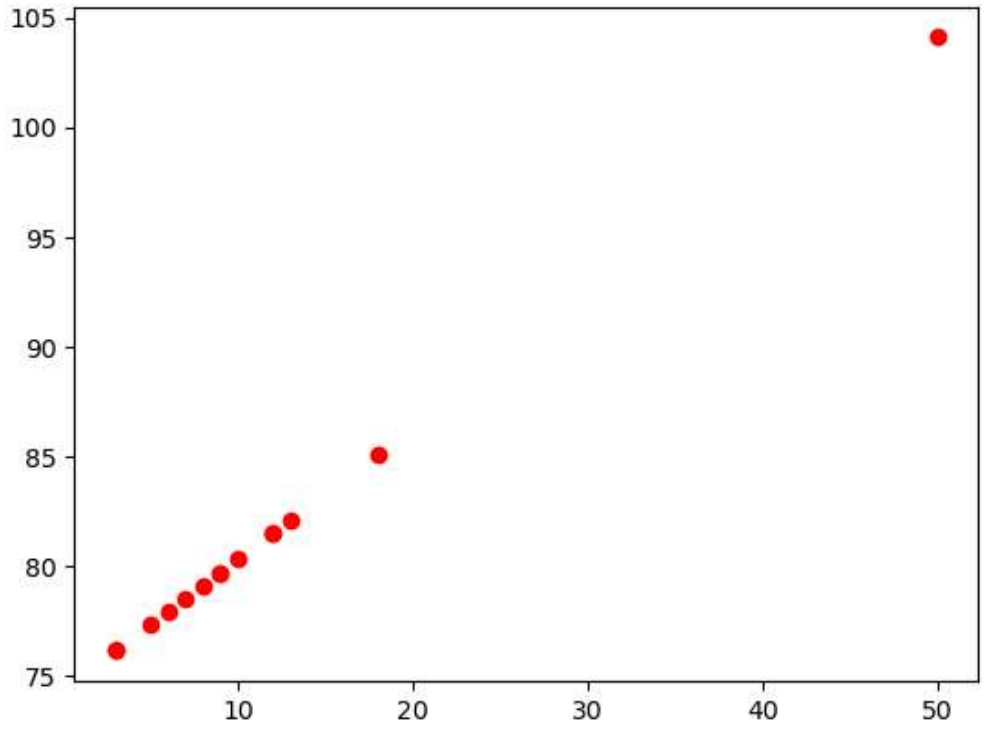
In [140]:
```
def fx(x):
    return slope * x + intercept
```

In [141]:
```
model = []
for var in x:
    r = fx(var)
    model.append(r)

model
```

Out[141]:
```
[77.91526032315979,
 79.10504687811691,
 79.69994015559546,
 80.29483343307402,
 81.48461998803113,
 76.13058049072413,
 85.05397965290246,
 76.13058049072413,
 77.32036704568124,
 81.48461998803113,
 82.07951326550968,
 79.69994015559546,
 78.51015360063835,
 104.09056453221623]
```

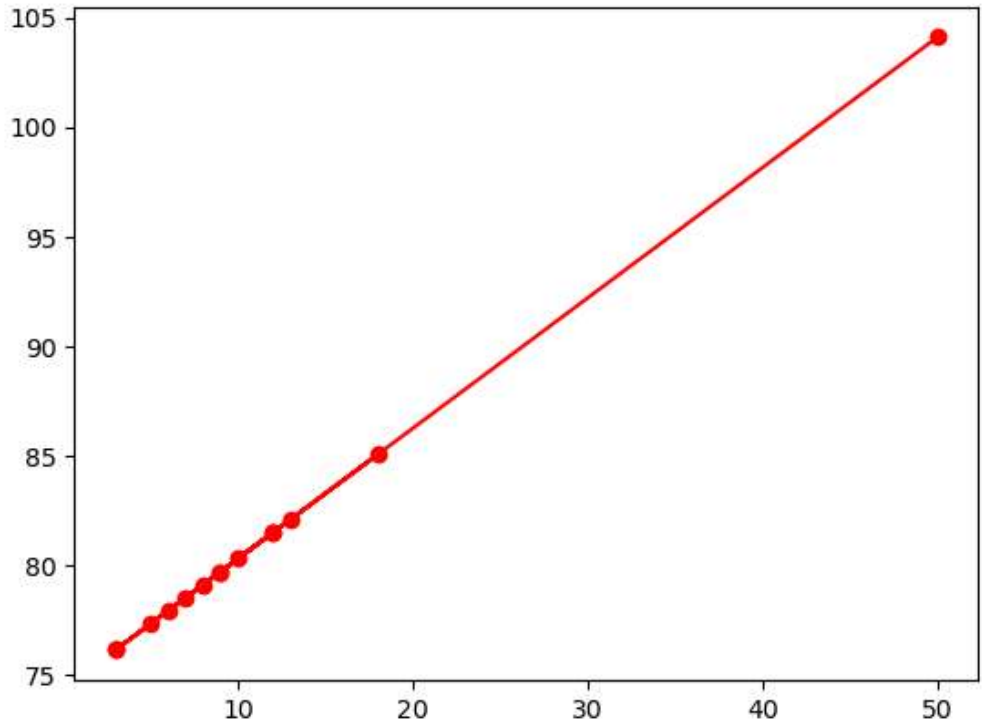
In [144]:
```
plt.scatter(x,model,c='r')
```

Out[144]:
```
<matplotlib.collections.PathCollection at 0x244a6ed0b50>
```



In [146]:
```
plt.plot(x,model,c='r',marker='o')
```

Out[146]:
```
[<matplotlib.lines.Line2D at 0x244a6d28710>]
```



In [147]:
```

```

Out[147]:
```
[77.91526032315979,
 79.10504687811691,
 79.69994015559546,
 80.29483343307402,
 81.48461998803113,
 76.13058049072413,
 85.05397965290246,
 76.13058049072413,
 77.32036704568124,
 81.48461998803113,
 82.07951326550968,
 79.69994015559546,
 78.51015360063835,
 104.09056453221623]
```

In [148]:
```
slope,intercept,r,p,stderr = stats.linregress(x,y)

def fx(x):
    return slope * x + intercept

fx(55) # input value is 55
```

Out[148]:
```
107.06503091960903
```

In [ ]:
```
pd.read_csv('E:\\emp.csv')
            ############ - BigData  ==> PySpark

python --------------- spark
           API
           |
          pyspark
             - dataframe <== immutable Vs pandas dataframe - mutable(inplce=True)
                ....

DataProcessing -->spark -> parallel cluster

    L=[10,20,30,40,50]
    |
    calculate sum of the list -> execution is done from local m/c

    Vs
    in pyspark
    L=[10,20,30,40,50]
    |
    calculate sum of the list -> execution is done from different m/c (cluster )
```

In [150]:
```
import seaborn as sn
sn.load_dataset('iris')
```

Out[150]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [151]:
```
from sklearn.model_selection import train_test_split
```

In [168]:
```
iris = sn.load_dataset('iris')
iris = iris[['petal_length','petal_width']]
X = iris['petal_length']
Y = iris['petal_width']
```

In [155]:
```
#help(train_test_split)
```

In [169]:
```
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.4,random_state=23)
```

In [170]:
```
x_train = np.array(x_train).reshape(-1,1)
x_test = np.array(x_test).reshape(-1,1)
```

In [171]:
```
from sklearn.linear_model import LinearRegression
```

In [161]:
```
print(LinearRegression)
```

```
<class 'sklearn.linear_model._base.LinearRegression'>
```

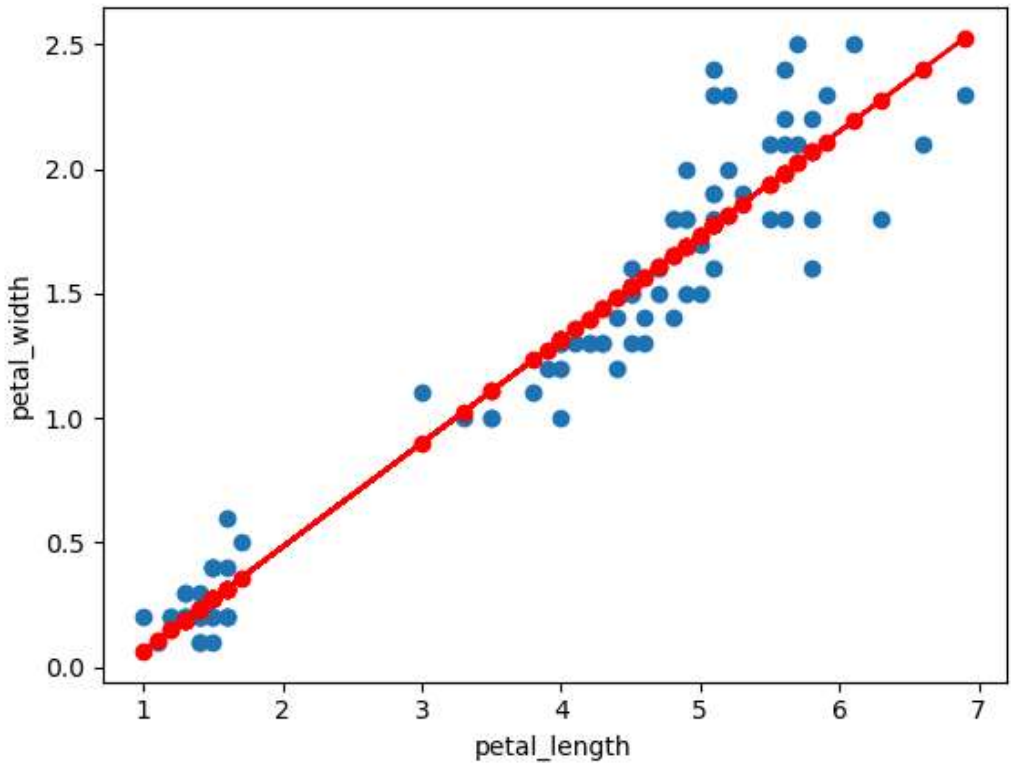
In [172]:
```
model_obj = LinearRegression()
```

In [173]:
```
model_obj.fit(x_train,y_train)
m = model_obj.intercept_
c = model_obj.coef_
```

In [174]:
```
y_pred_train = m * x_train + c
y_pred_train = model_obj.predict(x_train)
```

In [177]:
```
plt.scatter(x_train,y_train)
plt.plot(x_train,y_pred_train,color='r',marker='o')
plt.xlabel('petal_length')
plt.ylabel('petal_width')
```

Out[177]:
```
Text(0, 0.5, 'petal_width')
```



In [178]:
```
from sklearn.metrics import r2_score
r2_score(y_train,y_pred_train)
```

Out[178]:
```
0.9307444954156013
```

In [ ]:
```
LLM
 - transformer
 - torch
 - tensorflow
    |
    huggingface ->login ->create a token ->update token to local m/c in .env file
    |
    # download locally
    |
    from transformer import AutoModel...,AutoTokenizer
    model_name = <modelName/Path>

    # load the model and tokenizeer from Huggingface Model

    model = AutoModel....from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretraing(model_name)

    # after download ->use local path
    model = AutoModel....from_pretrained('/local_File_path/')
    tokenizer = AutoTokenizer.from_pretraing('local_file_path')

    |
    webframework - flask
      |
     127.0.0.1:5000/predict
    |
    import requests
    response = requests.post('127.0.0.1:5000/predict',json=data)
```