Data Imputation (deal with missing values)

- It is a method for retaining the majority of datasets data and information by substituting missing data with a different value
- if only few of the values are missing ,we can perform imputation
- there are different methods for imputation
  - Using the mean value
  - Using the median value
  - Using the most frequent value
  - filling missing value with a constant
- Sem exams and electric bills

In [1]:

```
from sklearn.impute import SimpleImputer
```

In [6]:

```
import numpy as np
import pandas as pd
di={
    "a":pd.Series([12,34,45,np.nan,56],index=[1,2,3,4,5]),
    "b":pd.Series([90,89,78,89],index=[1,3,4,5]),
    "c":pd.Series([13,45,35,35],index=[1,2,3,4])
}

df=pd.DataFrame(di)
df
```

Out[6]:

	a	b	c
1	12.0	90.0	13.0
2	34.0	NaN	45.0
3	45.0	89.0	35.0
4	NaN	78.0	35.0
5	56.0	89.0	NaN

In [7]:

```
si=SimpleImputer(strategy="median")
si.fit_transform(df)
```

Out[7]:

```
array([[12. , 90. , 13. ],
       [34. , 89. , 45. ],
       [45. , 89. , 35. ],
       [39.5, 78. , 35. ],
       [56. , 89. , 35. ]])
```

In [8]:

```
df.median()
```

Out[8]:

```
a    39.5
b    89.0
c    35.0
dtype: float64
```

In [9]:

```
sm=SimpleImputer(strategy='mean')
sm.fit_transform(df)
```

Out[9]:

```
array([[12.  , 90.  , 13.  ],
       [34.  , 86.5 , 45.  ],
       [45.  , 89.  , 35.  ],
       [36.75, 78.  , 35.  ],
       [56.  , 89.  , 32.  ]])
```

In [10]:

```
df.mean()
```

Out[10]:

```
a    36.75
b    86.50
c    32.00
dtype: float64
```

In [11]:

```
sf=SimpleImputer(strategy="most_frequent")
sf.fit_transform(df)
```

```
C:\Users\meena\anaconda3\lib\site-packages\sklearn\impute\_base.py:49: Fut
ureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), th
e default behavior of `mode` typically preserves the axis it acts along. I
n SciPy 1.11.0, this behavior will change: the default value of `keepdims`
will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to
True or False to avoid this warning.
  mode = stats.mode(array)
```

Out[11]:

```
array([[12., 90., 13.],
       [34., 89., 45.],
       [45., 89., 35.],
       [12., 78., 35.],
       [56., 89., 35.]])
```

In [12]:

```
si=SimpleImputer(strategy="constant",fill_value=-1)
si.fit_transform(df)
```

Out[12]:

```
array([[12., 90., 13.],
       [34., -1., 45.],
       [45., 89., 35.],
       [-1., 78., 35.],
       [56., 89., -1.]])
```

Task

- load titanic dataset
- clean data by using SimpleImputer

Feature scaling

- Feature scaling is a preprocessing technique that involves transforming the values of features or variables in a dataset to a similar scale
- real world datasets contains features that are varying in agnitute,units or range
- Inorder to convert these features on the samescale, we need to perform feature scaling

In [ ]:

```
[10kg, 100tons, 800gms, 70gms]   ---> weights

kgs

10+100+800+70
```

Scaling techniques

- Standardizing Data
- Data Range
- Normalizing Data
- Robust scaling

Standardizing data

- Converting raw data into standard format to make it easier to understand
- The standard format refers to data that has 0 mean and unit variance(i.e standard deviation=1). the process of converting data into this format is called Data Standardization
- Improves the performance of model
- Standardization rescales data to have mean=0 and standard deviation of 1
- the formula for this is (x-mean)/standard deviation

In [13]:

```
adv=pd.read_csv("Advertising.csv")
adv
```

Out[13]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

In [14]:

```
adv.head()
```

Out[14]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

In [15]:

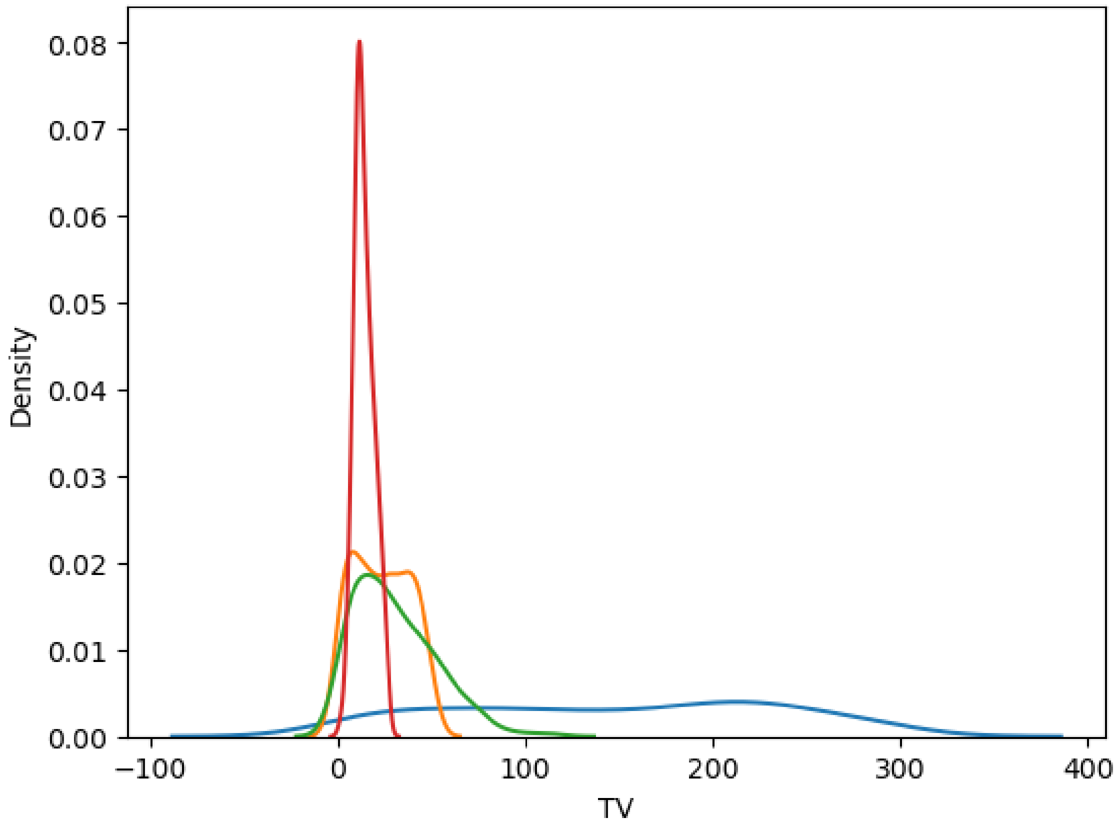
```
import seaborn as sns
```

In [17]:

```
sns.kdeplot(adv["TV"])  # kernel density plot to display how data is distributed as area
sns.kdeplot(adv["radio"])
sns.kdeplot(adv["newspaper"])
sns.kdeplot(adv["sales"])
```

Out[17]:

```
<AxesSubplot:xlabel='TV', ylabel='Density'>
```



In [18]:

```
adv["TV"][0]
```

Out[18]:

```
230.1
```

In [21]:

```
# std_data=(x-mean)/std(x)

(adv["TV"][0]-adv["TV"].mean())/adv["TV"].std()
```

Out[21]:

```
0.9674245973763037
```

In [20]:

```
adv["TV"].mean()
```

Out[20]:

```
147.0425
```

In [22]:

```
from sklearn.preprocessing import scale
```

In [25]:

```
scl=scale(adv)
scl
```

Out[25]:

```
array([[ 9.69852266e-01,  9.81522472e-01,  1.77894547e+00,
         1.55205313e+00],
       [-1.19737623e+00,  1.08280781e+00,  6.69578760e-01,
        -6.96046111e-01],
       [-1.51615499e+00,  1.52846331e+00,  1.78354865e+00,
        -9.07405869e-01],
       [ 5.20496822e-02,  1.21785493e+00,  1.28640506e+00,
         8.60330287e-01],
       [ 3.94182198e-01, -8.41613655e-01,  1.28180188e+00,
        -2.15683025e-01],
       [-1.61540845e+00,  1.73103399e+00,  2.04592999e+00,
        -1.31091086e+00],
       [-1.04557682e+00,  6.43904671e-01, -3.24708413e-01,
        -4.27042783e-01],
       [-3.13436589e-01, -2.47406325e-01, -8.72486994e-01,
        -1.58039455e-01],
       [-1.61657614e+00, -1.42906863e+00, -1.36042422e+00,
        -1.77205942e+00],
```

In [27]:

```
scl_data=pd.DataFrame(scl,columns=adv.columns)
scl_data
```

Out[27]:

	TV	radio	newspaper	sales
0	0.969852	0.981522	1.778945	1.552053
1	-1.197376	1.082808	0.669579	-0.696046
2	-1.516155	1.528463	1.783549	-0.907406
3	0.052050	1.217855	1.286405	0.860330
4	0.394182	-0.841614	1.281802	-0.215683
...	...	...	...	...
195	-1.270941	-1.321031	-0.771217	-1.234053
196	-0.617035	-1.240003	-1.033598	-0.830548
197	0.349810	-0.942899	-1.111852	-0.234898
198	1.594565	1.265121	1.640850	2.205347
199	0.993206	-0.990165	-1.005979	-0.119610

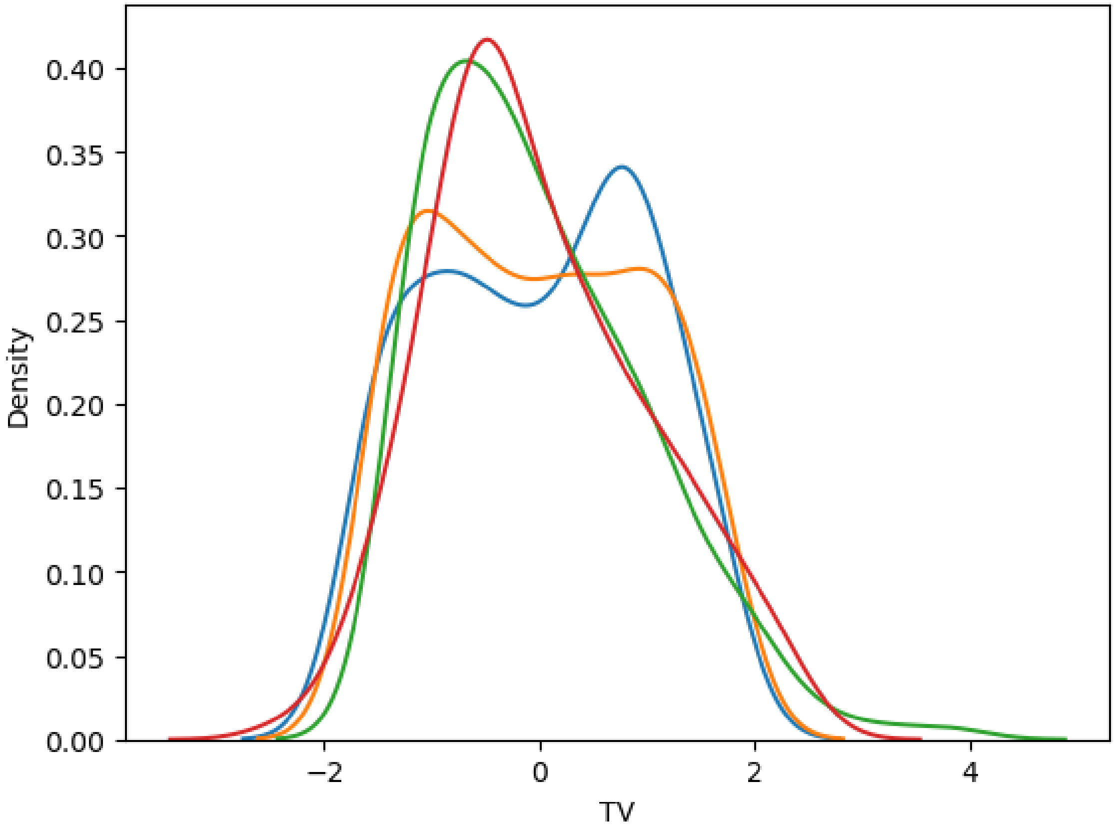
200 rows × 4 columns

In [29]:

```
sns.kdeplot(scl_data["TV"])
sns.kdeplot(scl_data["radio"])
sns.kdeplot(scl_data["newspaper"])
sns.kdeplot(scl_data["sales"])
```

Out[29]:

```
<AxesSubplot:xlabel='TV', ylabel='Density'>
```



In [31]:

```
adv.mean()
```

Out[31]:

```
TV           147.0425
radio         23.2640
newspaper     30.5540
sales         14.0225
dtype: float64
```

In [32]:

```
adv.std()
```

Out[32]:

```
TV           85.854236
radio        14.846809
newspaper    21.778621
sales         5.217457
dtype: float64
```

In [33]:

```
scl_data.mean().round(3)
```

Out[33]:

```
TV           0.0
radio       -0.0
newspaper    0.0
sales       -0.0
dtype: float64
```

In [34]:

```
scl_data.std().round(3)
```

Out[34]:

```
TV           1.003
radio        1.003
newspaper    1.003
sales        1.003
dtype: float64
```

Data range

- Scale the data by compressing it into a fixed range
- one of the biggest use case for this is compressing data into the range[0,1]
- MinMaxScaler

In [35]:

```
adv.head()
```

Out[35]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

In [36]:

```
from sklearn.preprocessing import MinMaxScaler
```

In [37]:

```
mnscl=MinMaxScaler()
```

In [38]:

```
mnscale=mnscl.fit_transform(adv)
```

In [43]:

```
mnscale
```

Out[43]:

```
array([[0.77578627, 0.76209677, 0.60598065, 0.80708661],
       [0.1481231 , 0.79233871, 0.39401935, 0.34645669],
       [0.0557998 , 0.92540323, 0.60686016, 0.30314961],
       [0.50997633, 0.83266129, 0.51187335, 0.66535433],
       [0.60906324, 0.21774194, 0.51099384, 0.44488189],
       [0.02705445, 0.9858871 , 0.65699208, 0.22047244],
       [0.19208657, 0.66129032, 0.20404573, 0.4015748 ],
       [0.4041258 , 0.39516129, 0.09938434, 0.45669291],
       [0.02671627, 0.04233871, 0.00615655, 0.12598425],
       [0.67331755, 0.05241935, 0.18381706, 0.35433071],
       [0.2211701 , 0.11693548, 0.21020229, 0.27559055],
       [0.72370646, 0.48387097, 0.03254178, 0.62204724],
       [0.07811972, 0.70766129, 0.5769569 , 0.2992126 ],
       [0.32735881, 0.15322581, 0.06068602, 0.31889764],
       [0.68785932, 0.66330645, 0.40193492, 0.68503937],
       [0.65843761, 0.96169355, 0.46262093, 0.81889764],
       [0.22691917, 0.73790323, 1.        , 0.42913386],
       [0.94927291, 0.7983871 , 0.48812665, 0.8976378 ],
```

In [39]:

```
mnscale.min()
```

Out[39]:

```
0.0
```

In [40]:

```
mnscale.max()
```

Out[40]:

```
1.0
```

In [41]:

```
adv.min()
```

Out[41]:

```
TV           0.7
radio        0.0
newspaper    0.3
sales        1.6
dtype: float64
```

In [42]:

```
adv.max()
```

Out[42]:

```
TV           296.4
radio         49.6
newspaper    114.0
sales         27.0
dtype: float64
```

Normalizing data

- want to scale the individual data observations(i.e rows)
- Used in classification probelem and data mining
- when clustering data we need to apply L2 normalization to each row
- L2 normalization applied to particular row of a data array
- L2 norm of a row is the squareroot of the sum of the squared values for the row

In [44]:

```
home=pd.read_csv("HomeBuyer.csv")
```

In [45]:

```
home
```

...

In [47]:

```
from sklearn.preprocessing import Normalizer
```

In [48]:

```
norm=Normalizer()
```

In [49]:

```
nor_data=norm.fit_transform(home)
```

In [50]:

```
nor_data
```

...

In [51]:

```
nor_data.min()
```

Out[51]:

```
0.0
```

In [52]:

```
nor_data.max()
```

Out[52]:

```
0.9999999807797668
```

Robust scaling

- Deal with outliers (data point which is significantly further away from the other data points
- Robustly scale the data i.e avoid being affected by the outliers
- Scaling by using data's mean and interquartile range(IQR)
- Here mean affected but median remains same
- Subtract the median fro each data value then scale to the IQR

In [53]:

```
adv.head()
```

Out[53]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

In [54]:

```
from sklearn.preprocessing import RobustScaler
```

In [55]:

```
rscl=RobustScaler()
```

In [56]:

```
rscl=rscl.fit_transform(adv)
```

In [57]:

```
rscl_data=pd.DataFrame(rscl,columns=adv.columns)
rscl_data
```

Out[57]:

	TV	radio	newspaper	sales
0	0.556248	0.561205	1.343122	1.309609
1	-0.728626	0.617702	0.598145	-0.355872
2	-0.917619	0.866290	1.346213	-0.512456
3	0.012115	0.693032	1.012365	0.797153
4	0.214953	-0.455744	1.009274	0.000000
...	...	...	...	...
195	-0.772240	-0.723164	-0.369397	-0.754448
196	-0.384562	-0.677966	-0.545595	-0.455516
197	0.188647	-0.512241	-0.598145	-0.014235
198	0.926618	0.719397	1.250386	1.793594
199	0.570093	-0.538606	-0.527048	0.071174

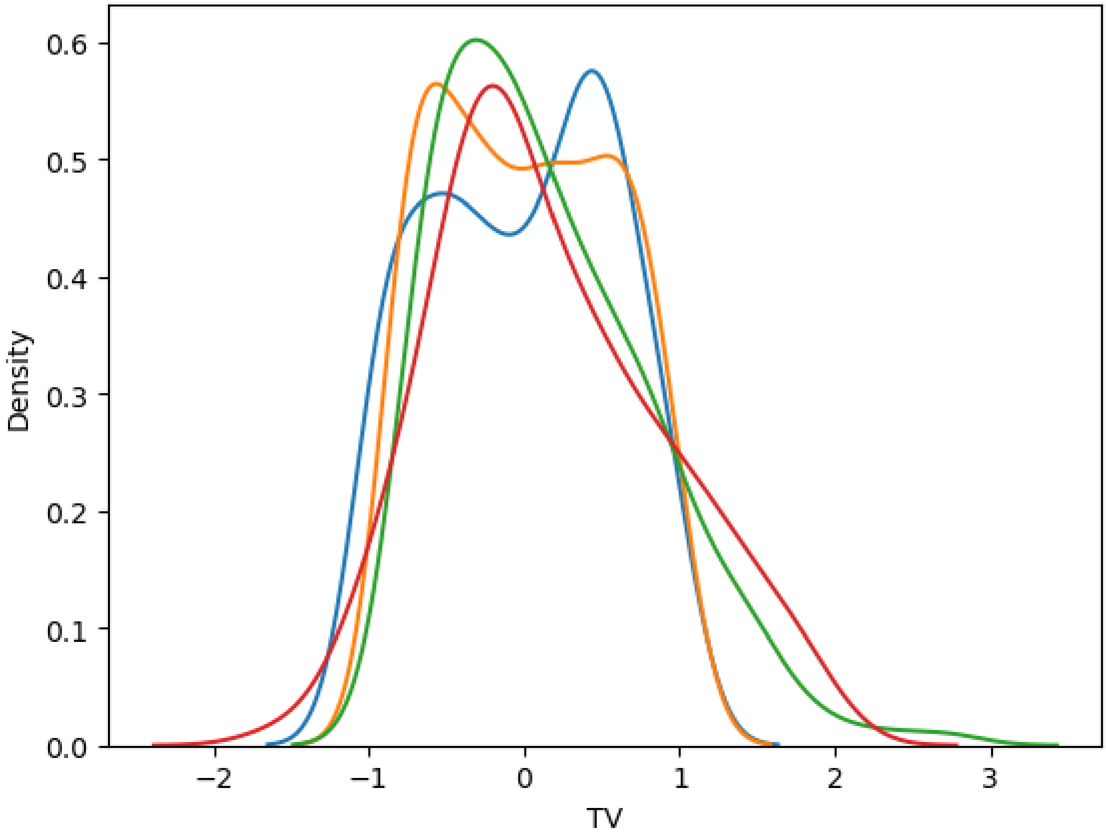
200 rows × 4 columns

In [59]:

```
sns.kdeplot(rscl_data["TV"])
sns.kdeplot(rscl_data["radio"])
sns.kdeplot(rscl_data["newspaper"])
sns.kdeplot(rscl_data["sales"])
```

Out[59]:

```
<AxesSubplot:xlabel='TV', ylabel='Density'>
```



In [ ]:

```

```