Today topics:

- group df
- concate
- sort
- merge/join

Data Preprocess:

- standardization(standard scaler)
- Robost scaling
- Range scaling(MinMaxScaler)
- Normalization(L2 Norm)

Grouping:

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

In [2]:
```
df=pd.read_csv("weather.csv")
df.head()
```

Out[2]:

	s.no	est	temparature	winspeed	humidity	event	city
0	1	1/2/2019	30	12	12.50	rain	guntur
1	1	1/2/2019	33	14	14.23	rain	vijayawada
2	2	1/3/2019	43	13	43.13	rain	vizag
3	3	1/4/2019	55	8	55.80	fullair	guntur
4	4	1/5/2019	66	10	66.10	cold	vijayawada

In [3]:
```
df['city'].unique()
```

Out[3]:
```
array(['guntur', 'vijayawada', 'vizag'], dtype=object)
```

In [4]:
```
g=df.groupby('city')
g
```

Out[4]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000023EC2C04860>
```

In [6]:
```
for i,j in g:
    print(i)
    print(j)
```

```
guntur
   s.no        est  temparature  winspeed  humidity    event    city
0     1   1/2/2019           30        12     12.50     rain  guntur
3     3   1/4/2019           55         8     55.80  fullair  guntur
6     6   1/7/2019           76        17     76.17  fullair  guntur
9     9  1/10/2019           90        11     90.11     cold  guntur
vijayawada
    s.no        est  temparature  winspeed  humidity    event        city
1      1   1/2/2019           33        14     14.23     rain  vijayawada
4      4   1/5/2019           66        10     66.10     cold  vijayawada
7      7   1/8/2019           89        23     89.23     cold  vijayawada
10    10  1/11/2019           65        18     65.18  fullair  vijayawada
vizag
    s.no        est  temparature  winspeed  humidity    event   city
2      2   1/3/2019           43        13     43.13     rain  vizag
5      5   1/6/2019           34        15     34.15     cold  vizag
8      8   1/9/2019           23         9     23.90  fullair  vizag
11    11  1/12/2019           78        20     78.20  fullair  vizag
```

In [8]:
```
for i,j in g:
    if(i=='guntur'):
        guntur=pd.DataFrame(j)
    elif(i=="vizag"):
        vizag=pd.DataFrame(j)
    elif(i=="vijayawada"):
        vij=pd.DataFrame(j)
```

In [10]:
```
vizag
vij
```

Out[10]:

	s.no	est	temparature	winspeed	humidity	event	city
1	1	1/2/2019	33	14	14.23	rain	vijayawada
4	4	1/5/2019	66	10	66.10	cold	vijayawada
7	7	1/8/2019	89	23	89.23	cold	vijayawada
10	10	1/11/2019	65	18	65.18	fullair	vijayawada

In [11]:
```
g=df.groupby('city')
g.get_group('vijayawada')
```

Out[11]:

	s.no	est	temparature	winspeed	humidity	event	city
1	1	1/2/2019	33	14	14.23	rain	vijayawada
4	4	1/5/2019	66	10	66.10	cold	vijayawada
7	7	1/8/2019	89	23	89.23	cold	vijayawada
10	10	1/11/2019	65	18	65.18	fullair	vijayawada

sorting

In [12]:
```
#asending order
df.sort_values(['temparature'])
```

Out[12]:

	s.no	est	temparature	winspeed	humidity	event	city
8	8	1/9/2019	23	9	23.90	fullair	vizag
0	1	1/2/2019	30	12	12.50	rain	guntur
1	1	1/2/2019	33	14	14.23	rain	vijayawada
5	5	1/6/2019	34	15	34.15	cold	vizag
2	2	1/3/2019	43	13	43.13	rain	vizag
3	3	1/4/2019	55	8	55.80	fullair	guntur
10	10	1/11/2019	65	18	65.18	fullair	vijayawada
4	4	1/5/2019	66	10	66.10	cold	vijayawada
6	6	1/7/2019	76	17	76.17	fullair	guntur
11	11	1/12/2019	78	20	78.20	fullair	vizag
7	7	1/8/2019	89	23	89.23	cold	vijayawada
9	9	1/10/2019	90	11	90.11	cold	guntur

In [13]:
```
#decending order
df.sort_values(['temparature'],ascending=False)
```

Out[13]:

	s.no	est	temparature	winspeed	humidity	event	city
9	9	1/10/2019	90	11	90.11	cold	guntur
7	7	1/8/2019	89	23	89.23	cold	vijayawada
11	11	1/12/2019	78	20	78.20	fullair	vizag
6	6	1/7/2019	76	17	76.17	fullair	guntur
4	4	1/5/2019	66	10	66.10	cold	vijayawada
10	10	1/11/2019	65	18	65.18	fullair	vijayawada
3	3	1/4/2019	55	8	55.80	fullair	guntur
2	2	1/3/2019	43	13	43.13	rain	vizag
5	5	1/6/2019	34	15	34.15	cold	vizag
1	1	1/2/2019	33	14	14.23	rain	vijayawada
0	1	1/2/2019	30	12	12.50	rain	guntur
8	8	1/9/2019	23	9	23.90	fullair	vizag

In [14]:
```
df.sort_values(['temparature','winspeed'],ascending=[False,True])
```

Out[14]:

	s.no	est	temparature	winspeed	humidity	event	city
9	9	1/10/2019	90	11	90.11	cold	guntur
7	7	1/8/2019	89	23	89.23	cold	vijayawada
11	11	1/12/2019	78	20	78.20	fullair	vizag
6	6	1/7/2019	76	17	76.17	fullair	guntur
4	4	1/5/2019	66	10	66.10	cold	vijayawada
10	10	1/11/2019	65	18	65.18	fullair	vijayawada
3	3	1/4/2019	55	8	55.80	fullair	guntur
2	2	1/3/2019	43	13	43.13	rain	vizag
5	5	1/6/2019	34	15	34.15	cold	vizag
1	1	1/2/2019	33	14	14.23	rain	vijayawada
0	1	1/2/2019	30	12	12.50	rain	guntur
8	8	1/9/2019	23	9	23.90	fullair	vizag

concat

In [15]:
```
d={"city":['Guntur','Vijayawada','Vizag'],"count":[120,150,130],"area":[300.23,5(
df1=pd.DataFrame(d)
df1
```

Out[15]:

	city	count	area
0	Guntur	120	300.23
1	Vijayawada	150	500.43
2	Vizag	130	450.67

In [16]:
```
d2={"city":['Tirupati','Srikakulam','anathapur'],"count":[150,170,120],"area":[2(
df2=pd.DataFrame(d2)
df2
```

Out[16]:

	city	count	area
0	Tirupati	150	200.23
1	Srikakulam	170	400.43
2	anathapur	120	350.67

In [17]:
```
total_df=pd.concat([df1,df2])
total_df
```

Out[17]:

	city	count	area
0	Guntur	120	300.23
1	Vijayawada	150	500.43
2	Vizag	130	450.67
0	Tirupati	150	200.23
1	Srikakulam	170	400.43
2	anathapur	120	350.67

In [18]:
```
total_df=pd.concat([df1,df2],ignore_index=True)
total_df
```

Out[18]:

	city	count	area
0	Guntur	120	300.23
1	Vijayawada	150	500.43
2	Vizag	130	450.67
3	Tirupati	150	200.23
4	Srikakulam	170	400.43
5	anathapur	120	350.67

In [19]:
```
total_df=pd.concat([df1,df2],keys=['a','b'])
total_df
```

Out[19]:

		city	count	area
a	**0**	Guntur	120	300.23
	1	Vijayawada	150	500.43
	2	Vizag	130	450.67
b	**0**	Tirupati	150	200.23
	1	Srikakulam	170	400.43
	2	anathapur	120	350.67

In [20]:
```
total_df.loc['a']
```

Out[20]:

	city	count	area
0	Guntur	120	300.23
1	Vijayawada	150	500.43
2	Vizag	130	450.67

In [21]:
```
total_df=pd.concat([df1,df2],axis=1)
total_df
```

Out[21]:

	city	count	area	city	count	area
0	Guntur	120	300.23	Tirupati	150	200.23
1	Vijayawada	150	500.43	Srikakulam	170	400.43
2	Vizag	130	450.67	anathapur	120	350.67

merge/join

In [69]:
```
d={"sno":[101,102,103,104],"sname":["a","b","c","d"]}
df1=pd.DataFrame(d)
df1
```

Out[69]:

	sno	sname
0	101	a
1	102	b
2	103	c
3	104	d

In [71]:
```
f={"sno":[101,102,103,104,105,110],"address":["ab","bc","cd","dd",'ee','ff']}
df2=pd.DataFrame(f)
df2
```

Out[71]:

	sno	address
0	101	ab
1	102	bc
2	103	cd
3	104	dd
4	105	ee
5	110	ff

In [75]:
```
s=pd.merge(df1,df2,on='sno',how='right')
s
```

Out[75]:

	sno	sname	address
0	101	a	ab
1	102	b	bc
2	103	c	cd
3	104	d	dd
4	105	NaN	ee
5	110	NaN	ff

In [ ]:
```

```

In [22]:
```
s=pd.DataFrame({"sno":[101,102,103,104],"sname":["a","b","c","d"]})
s
```

Out[22]:

	sno	sname
0	101	a
1	102	b
2	103	c
3	104	d

In [30]:
```
s2=pd.DataFrame({'sno':[101,102,103,110],"saddress":["srikakulam","kadapa","Tuni
s2
```

Out[30]:

	sno	saddress
0	101	srikakulam
1	102	kadapa
2	103	Tuni
3	110	hyd

In [31]:
```
merge_df=pd.merge(s,s2,on='sno')
merge_df
```

Out[31]:

	sno	sname	saddress
0	101	a	srikakulam
1	102	b	kadapa
2	103	c	Tuni

In [32]:
```
merge_df=pd.merge(s,s2,on='sno',how='outer')
merge_df
```

Out[32]:

	sno	sname	saddress
0	101	a	srikakulam
1	102	b	kadapa
2	103	c	Tuni
3	104	d	NaN
4	110	NaN	hyd

In [33]:
```
merge_df=pd.merge(s,s2,on='sno',how='inner')
merge_df
```

Out[33]:

	sno	sname	saddress
0	101	a	srikakulam
1	102	b	kadapa
2	103	c	Tuni

In [34]:
```
merge_df=pd.merge(s,s2,on='sno',how='right')
merge_df
```

Out[34]:

	sno	sname	saddress
0	101	a	srikakulam
1	102	b	kadapa
2	103	c	Tuni
3	110	NaN	hyd

In [35]:
```
merge_df=pd.merge(s,s2,on='sno',how='left')
merge_df
```

Out[35]:

	sno	sname	saddress
0	101	a	srikakulam
1	102	b	kadapa
2	103	c	Tuni
3	104	d	NaN

In [37]:
```
merge_df.sort_values('sno')
```

Out[37]:

	sno	sname	saddress
0	101	a	srikakulam
1	102	b	kadapa
2	103	c	Tuni
3	104	d	NaN

DataPreprocessing:

- Data preprocessing is used for improve the quality of data
- Problems in data:
  - insufficient data

  - Too much data
  - Missing data
  - Duplicate data
  - outliers
- Outliers: In general term it is a data point that is significantly further away from the other data points

- Standardization(Standard Scaler)
- RobostScalling
- Data Range(MinMaxScaler)
- Normalization

Standardize data:

- SD is a useful technique to transform the attributes with a gaussian distribution
- when data can take a any range of values it makes difficult to interpret. so, datascientists will convert the data into standard format

0=> mean 1=>standara diviation

In [38]:
```
df=pd.read_csv("Advertisement.csv")    # d://folder1/filename.csv
df.head()
```

Out[38]:

	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 [40]:
```
df['TV'].min()
```

Out[40]:
```
0.7
```

In [41]:
```
df['TV'].max()
```

Out[41]:
```
296.4
```

In [42]:
```
df.mean()
```

Out[42]:
```
TV           147.0425
radio         23.2640
newspaper     30.5540
sales         14.0225
dtype: float64
```

In [43]:
```
df.median()
```

Out[43]:
```
TV           149.75
radio         22.90
newspaper     25.75
sales         12.90
dtype: float64
```

In [44]:
```
df.std()
```

Out[44]:
```
TV           85.854236
radio        14.846809
newspaper    21.778621
sales         5.217457
dtype: float64
```

In [45]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
TV           200 non-null float64
radio        200 non-null float64
newspaper    200 non-null float64
sales        200 non-null float64
dtypes: float64(4)
memory usage: 6.3 KB
```

In [47]:
```
df.isnull().sum()
```

Out[47]:
```
TV           0
radio        0
newspaper    0
sales        0
dtype: int64
```

In [51]:
```
df['TV'].isna().sum()
```

Out[51]:
```
0
```

In [50]:
```
df.describe()
```

Out[50]:

	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

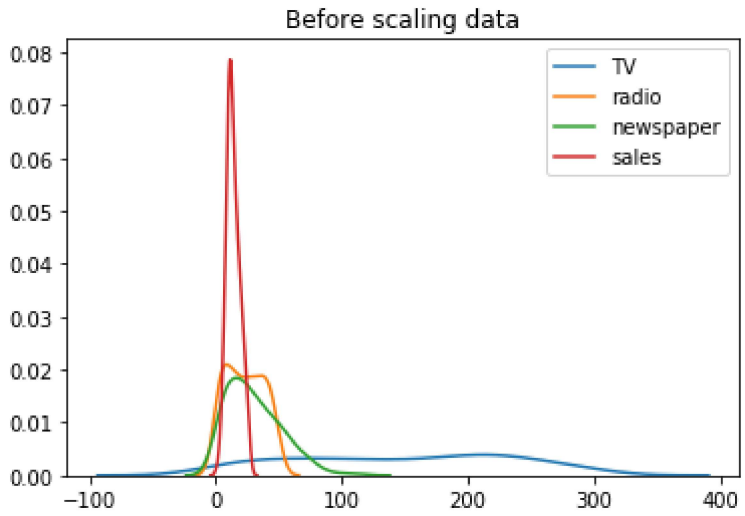
Visulize the data using KDE

Keranal Dencity Estimate plot is used for display the multiple samples data into one graph.

In [52]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [53]:
```
plt.title("Before scaling data")
sns.kdeplot(df['TV'])
sns.kdeplot(df['radio'])
sns.kdeplot(df['newspaper'])
sns.kdeplot(df['sales'])
```

Out[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23ec764c2b0>
```



In [54]:
```
from sklearn.preprocessing import StandardScaler
std=StandardScaler()
std_data=std.fit_transform(df)
std_data
```

...

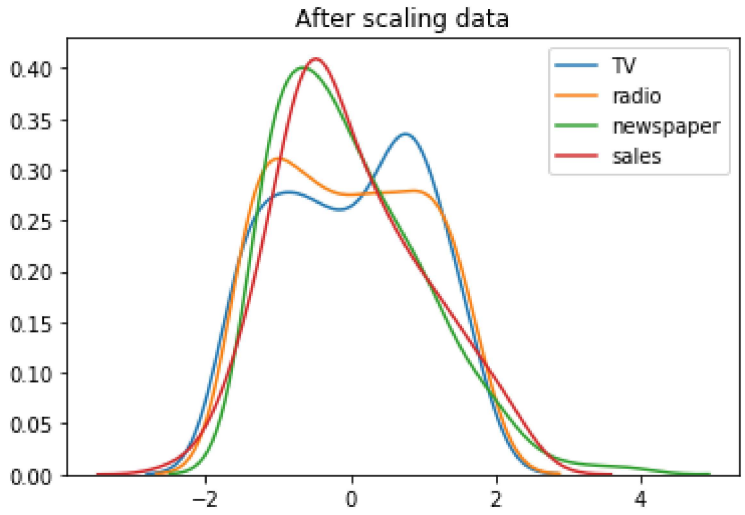
In [55]:
```
std_data_df=pd.DataFrame(std_data,columns=df.columns)
std_data_df.head()
```

Out[55]:

	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

In [56]:
```
plt.title("After scaling data")
sns.kdeplot(std_data_df['TV'])
sns.kdeplot(std_data_df['radio'])
sns.kdeplot(std_data_df['newspaper'])
sns.kdeplot(std_data_df['sales'])
```

Out[56]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23ec83849e8>
```



Robost Scalling

Robost scalling also used for scale the outliers, it scale using median and Inter Quartail Range(IQR)

In [57]:
```
from sklearn.preprocessing import RobustScaler
rb=RobustScaler()
rb_data=rb.fit_transform(df)
rb_data
```

...

In [58]:
```
rb_data_df=pd.DataFrame(rb_data,columns=df.columns)
rb_data_df
```

...

In [59]:
```
plt.title("After scaling data in robost scaler")
sns.kdeplot(rb_data_df['TV'])
sns.kdeplot(rb_data_df['radio'])
sns.kdeplot(rb_data_df['newspaper'])
sns.kdeplot(rb_data_df['sales'])
```

...

Range scaling(Min Max Scaller)

By using MinMaxScaller we can provide custom range for scaling the data

In [62]:
```
from sklearn.preprocessing import MinMaxScaler
m=MinMaxScaler(feature_range=(-1,1))
m_data=m.fit_transform(df)
m_data
```

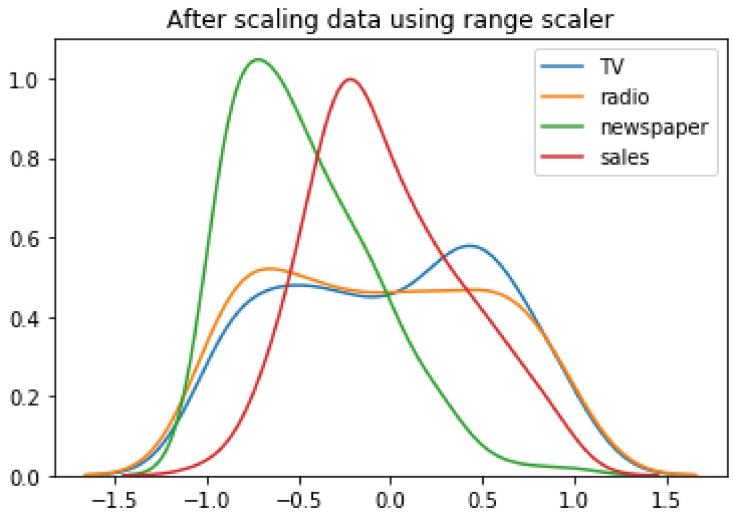
...

In [63]:
```
m_data_df=pd.DataFrame(m_data,columns=df.columns)
m_data_df
```

...

In [64]:
```
plt.title("After scaling data using range scaler")
sns.kdeplot(m_data_df['TV'])
sns.kdeplot(m_data_df['radio'])
sns.kdeplot(m_data_df['newspaper'])
sns.kdeplot(m_data_df['sales'])
```

Out[64]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23ec83c3470>
```



Normalization:

- upto now we are scale by using features(columns)
- In certain cases we want to scale the individual data observation(rows)
- when clustering data we need to apply normalization

In [65]:
```
from sklearn.preprocessing import Normalizer
n=Normalizer()
n_data=n.fit_transform(df)
n_data
```

...

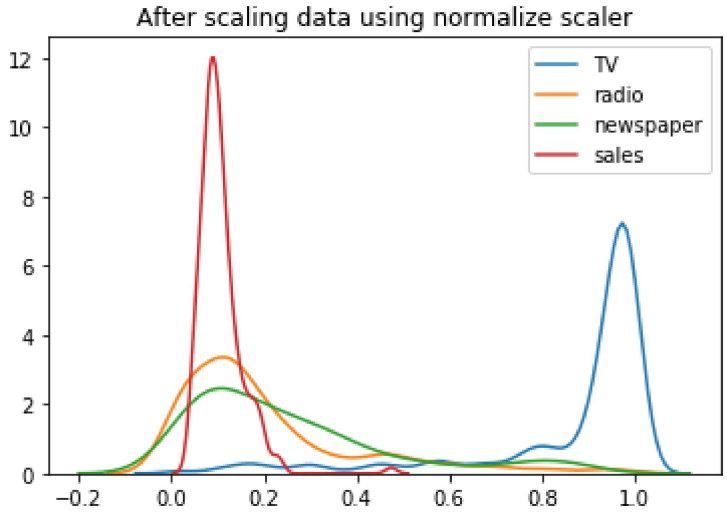
In [67]:
```
n_data_df=pd.DataFrame(n_data,columns=df.columns)
n_data_df.head()
```

Out[67]:

	TV	radio	newspaper	sales
0	0.942116	0.154767	0.283331	0.090486
1	0.591135	0.522059	0.599106	0.138153
2	0.201426	0.537527	0.811561	0.108911
3	0.898632	0.244974	0.346997	0.109734
4	0.947881	0.056621	0.306174	0.067631

In [68]:
```
plt.title("After scaling data using normalize scaler")
sns.kdeplot(n_data_df['TV'])
sns.kdeplot(n_data_df['radio'])
sns.kdeplot(n_data_df['newspaper'])
sns.kdeplot(n_data_df['sales'])
```

Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x23ec84b6da0>



In [ ]: