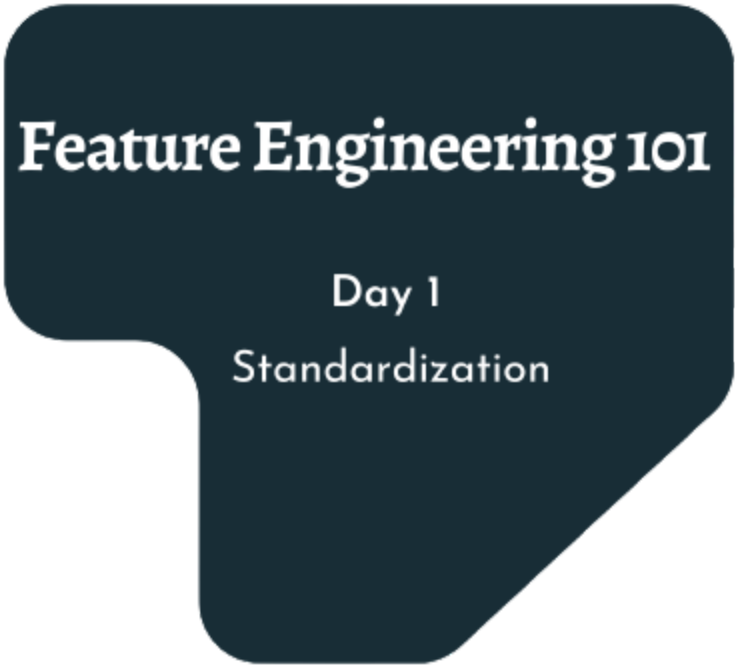
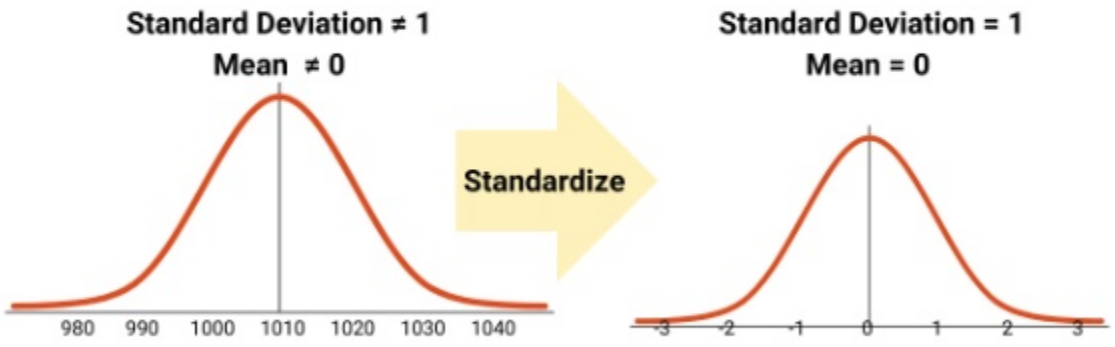


Standardization in Machine Learning



In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('Social_Network_Ads.csv')
```

In [3]:
```
df=df.iloc[:,2:]
```

In [4]:
```
df
```

Out[4]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

400 rows × 3 columns

In [5]:
```
df.describe()
```

Out[5]:

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

Now Train Test Spilit

In [6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df.drop('Purchased', axis=1),
                                                    df['Purchased'],
```

```
                                                    test_size = 0.3,
                                                    random_state=0)
X_train.shape, X_test.shape
```

Out[6]:
```
((280, 2), (120, 2))
```

Standerdscaler

In [7]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# fit the scaler to the train set, it will learn the parameters
scaler.fit(X_train)

# transform train and test sets
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [8]:
```
scaler.mean_
```

Out[8]:
```
array([3.78642857e+01, 6.98071429e+04])
```

In [9]:
```
X_train
```

Out[9]:

	Age	EstimatedSalary
92	26	15000
223	60	102000
234	38	112000
232	40	107000
377	42	53000
...	...	...
323	48	30000
192	29	43000
117	36	52000
47	27	54000
172	26	118000

280 rows × 2 columns

In [10]:
```
X_train_scaled
```

Out[10]:
```
array([[-1.1631724 , -1.5849703 ],
       [ 2.17018137,  0.93098672],
       [ 0.0133054 ,  1.22017719],
       [ 0.20938504,  1.07558195],
       [ 0.40546467, -0.48604654],
       [-0.28081405, -0.31253226],
       [ 0.99370357, -0.8330751 ],
       [ 0.99370357,  1.8563962 ],
```

```
[ 0.0133054 ,  1.24909623],
[-0.86905295,  2.26126285],
[-1.1631724 , -1.5849703 ],
[ 2.17018137, -0.80415605],
[-1.35925203, -1.46929411],
[ 0.40546467,  2.2901819 ],
[ 0.79762394,  0.75747245],
[-0.96709276, -0.31253226],
[ 0.11134522,  0.75747245],
[-0.96709276,  0.55503912],
[ 0.30742485,  0.06341534],
[ 0.69958412, -1.26686079],
[-0.47689368, -0.0233418 ],
[-1.7514113 ,  0.3526058 ],
[-0.67297331,  0.12125343],
[ 0.40546467,  0.29476771],
[-0.28081405,  0.06341534],
[-0.47689368,  2.2901819 ],
[ 0.20938504,  0.03449629],
[ 1.28782302,  2.20342476],
[ 0.79762394,  0.26584866],
[-0.28081405,  0.15017248],
[ 0.0133054 , -0.54388463],
[-0.18277423,  0.15017248],
[-0.08473441,  0.23692961],
[ 0.0133054 , -0.25469417],
[ 2.17018137,  1.104501  ],
[-1.7514113 ,  0.3526058 ],
[ 1.87606192,  0.12125343],
[ 0.40546467, -0.13901799],
[-1.1631724 ,  0.29476771],
[ 0.79762394,  1.36477242],
[-0.28081405, -0.25469417],
[-1.65337148, -0.05226085],
[-0.96709276, -0.74631796],
[ 0.30742485,  0.49720103],
[-0.08473441, -1.06442747],
[-1.06513258,  0.58395817],
[ 0.11134522, -0.80415605],
[-0.96709276,  1.53828669],
[-0.67297331,  1.39369146],
[-1.26121221,  0.49720103],
[-0.28081405,  0.03449629],
[-0.08473441,  0.00557724],
[-0.28081405, -0.89091319],
[ 0.89566375, -1.35361793],
[-0.28081405,  2.2323438 ],
[ 0.99370357,  1.97207239],
[-1.1631724 ,  0.46828198],
[-1.26121221,  0.26584866],
[ 1.38586284,  1.97207239],
[ 1.28782302, -1.35361793],
[-0.28081405, -0.28361322],
[-0.47689368,  1.24909623],
[-0.77101313,  1.07558195],
[ 0.99370357, -1.06442747],
[ 0.30742485,  0.29476771],
[ 0.99370357,  0.75747245],
[-0.67297331, -1.49821316],
[-0.67297331,  0.03449629],
[ 0.50350449,  1.71180097],
[ 2.07214155,  0.17909152],
[-1.94749093, -0.74631796],
[-0.18277423,  1.39369146],
[ 0.40546467,  0.58395817],
[ 0.89566375, -1.1511846 ],
```

```
[-1.1631724 , -0.775237  ],
[ 0.20938504,  0.23692961],
[ 0.79762394, -0.31253226],
[ 2.07214155, -0.80415605],
[ 0.79762394,  0.12125343],
[-0.28081405,  0.61287722],
[-0.96709276, -0.31253226],
[ 0.20938504, -0.37037036],
[ 2.07214155,  2.11666762],
[ 1.87606192, -1.26686079],
[ 1.38586284, -0.91983223],
[ 0.89566375,  1.24909623],
[ 1.48390265,  2.11666762],
[-0.28081405, -1.23794174],
[ 1.97410174,  0.90206768],
[ 0.69958412, -0.71739891],
[-1.45729185,  0.3526058 ],
[ 0.79762394, -1.35361793],
[ 0.40546467, -0.13901799],
[-0.96709276,  0.41044389],
[ 0.0133054 , -0.31253226],
[-1.1631724 ,  0.41044389],
[-0.86905295, -1.2090227 ],
[-0.08473441,  0.03449629],
[-1.55533166, -0.42820845],
[ 0.99370357, -1.00658937],
[ 1.09174339, -1.2090227 ],
[ 0.0133054 , -0.13901799],
[-1.06513258, -1.52713221],
[ 0.79762394, -1.2090227 ],
[ 0.99370357,  2.05882953],
[-1.1631724 , -1.52713221],
[-0.28081405,  0.78639149],
[ 0.11134522, -0.31253226],
[-1.35925203, -1.23794174],
[-0.5749335 , -1.49821316],
[ 0.79762394,  0.52612008],
[-0.28081405, -0.34145131],
[ 1.7780221 , -0.28361322],
[ 0.89566375, -1.03550842],
[ 0.20938504,  0.06341534],
[-0.5749335 ,  0.87314863],
[-1.84945111, -1.41145602],
[-1.26121221,  0.58395817],
[-0.28081405,  0.52612008],
[-0.96709276, -1.09334651],
[ 1.1897832 , -1.44037507],
[ 0.20938504, -0.31253226],
[ 1.1897832 , -0.74631796],
[-0.28081405,  0.06341534],
[ 0.20938504,  2.08774857],
[ 0.79762394, -1.09334651],
[ 0.11134522,  0.03449629],
[-1.7514113 ,  0.12125343],
[-0.86905295,  0.15017248],
[-0.67297331,  0.17909152],
[ 0.89566375, -1.29577984],
[ 0.20938504, -0.25469417],
[-0.37885386,  1.22017719],
[ 0.0133054 ,  0.29476771],
[ 0.40546467,  0.15017248],
[ 0.89566375, -0.65956082],
[ 0.11134522,  0.15017248],
[-1.84945111, -1.29577984],
[-0.08473441,  0.29476771],
[-0.18277423, -0.28361322],
```

```
[ 0.30742485, -0.51496559],
[-0.18277423,  1.59612479],
[ 0.99370357, -1.18010365],
[-0.18277423,  1.62504383],
[ 1.28782302,  1.8563962 ],
[-1.06513258, -0.37037036],
[ 0.0133054 ,  0.03449629],
[ 0.11134522, -0.25469417],
[-1.55533166, -1.23794174],
[-0.47689368, -0.28361322],
[ 0.99370357,  0.12125343],
[ 1.97410174, -1.35361793],
[ 1.48390265,  0.06341534],
[-0.5749335 ,  1.36477242],
[ 1.58194247,  0.00557724],
[-0.77101313,  0.29476771],
[ 1.97410174,  0.7285534 ],
[-1.1631724 , -0.51496559],
[ 0.69958412,  0.26584866],
[-1.35925203, -0.42820845],
[ 0.20938504,  0.15017248],
[-0.47689368, -1.2090227 ],
[ 0.6015443 ,  2.00099143],
[-1.55533166, -1.49821316],
[-0.47689368, -0.54388463],
[ 0.50350449,  1.82747716],
[-1.35925203, -1.09334651],
[ 0.79762394, -1.38253697],
[-0.28081405, -0.42820845],
[ 1.58194247,  0.98882482],
[ 0.99370357,  1.42261051],
[-0.28081405, -0.48604654],
[-0.08473441,  2.14558666],
[-1.45729185, -0.11009894],
[-0.08473441,  1.94315334],
[-0.67297331, -0.34145131],
[-0.47689368, -0.8330751 ],
[ 0.69958412, -1.38253697],
[-0.77101313, -1.5849703 ],
[-1.84945111, -1.46929411],
[ 1.09174339,  0.12125343],
[ 0.11134522,  1.50936765],
[-0.28081405,  0.09233438],
[ 0.11134522,  0.03449629],
[-1.35925203, -1.35361793],
[ 0.30742485,  0.06341534],
[-0.86905295,  0.38152485],
[ 1.58194247, -1.26686079],
[-0.28081405, -0.74631796],
[-0.08473441,  0.15017248],
[-0.86905295, -0.65956082],
[-0.67297331, -0.05226085],
[ 0.40546467, -0.45712749],
[-0.77101313,  1.88531525],
[ 1.38586284,  1.27801528],
[ 1.1897832 , -0.97767033],
[ 1.7780221 ,  1.82747716],
[-0.86905295, -0.25469417],
[-0.77101313,  0.55503912],
[-1.1631724 , -1.55605125],
[-0.47689368, -1.12226556],
[ 0.30742485,  0.06341534],
[-0.18277423, -1.06442747],
[ 1.67998229,  1.59612479],
[ 0.99370357,  1.76963906],
[ 0.30742485,  0.03449629],
```

```
[-0.77101313, -0.22577513],
[-0.08473441,  0.06341534],
[ 0.30742485, -0.19685608],
[ 1.97410174, -0.65956082],
[-0.77101313,  1.33585337],
[-1.7514113 , -0.60172273],
[-0.08473441,  0.12125343],
[ 0.30742485, -0.31253226],
[ 1.09174339,  0.55503912],
[-0.96709276,  0.26584866],
[ 1.48390265,  0.3526058 ],
[ 0.20938504, -0.37037036],
[ 2.17018137, -1.03550842],
[-0.28081405,  1.104501  ],
[-1.65337148,  0.06341534],
[ 0.0133054 ,  0.03449629],
[ 0.11134522,  1.04666291],
[-0.08473441, -0.37037036],
[-1.1631724 ,  0.06341534],
[-0.28081405, -1.35361793],
[ 1.58194247,  1.104501  ],
[-0.77101313, -1.52713221],
[ 0.11134522,  1.8563962 ],
[-0.86905295, -0.775237  ],
[-0.47689368, -0.775237  ],
[-0.28081405, -0.91983223],
[ 0.30742485, -0.71739891],
[ 0.30742485,  0.06341534],
[ 0.11134522,  1.8563962 ],
[-1.06513258,  1.94315334],
[-1.65337148, -1.55605125],
[-1.1631724 , -1.09334651],
[-0.67297331, -0.11009894],
[ 0.11134522,  0.09233438],
[ 0.30742485,  0.26584866],
[ 0.89566375, -0.57280368],
[ 0.30742485, -1.1511846 ],
[-0.08473441,  0.67071531],
[ 2.17018137, -0.68847986],
[-1.26121221, -1.38253697],
[-0.96709276, -0.94875128],
[ 0.0133054 , -0.42820845],
[-0.18277423, -0.45712749],
[-1.7514113 , -0.97767033],
[ 1.7780221 ,  0.98882482],
[ 0.20938504, -0.37037036],
[ 0.40546467,  1.104501  ],
[-1.7514113 , -1.35361793],
[ 0.20938504, -0.13901799],
[ 0.89566375, -1.44037507],
[-1.94749093,  0.46828198],
[-0.28081405,  0.26584866],
[ 1.87606192, -1.06442747],
[-0.37885386,  0.06341534],
[ 1.09174339, -0.89091319],
[-1.06513258, -1.12226556],
[-1.84945111,  0.00557724],
[ 0.11134522,  0.26584866],
[-1.1631724 ,  0.32368675],
[-1.26121221,  0.29476771],
[-0.96709276,  0.43936294],
[ 1.67998229, -0.89091319],
[ 1.1897832 ,  0.52612008],
[ 1.09174339,  0.52612008],
[ 1.38586284,  2.31910094],
[-0.28081405, -0.13901799],
```

```
       [ 0.40546467, -0.45712749],
       [-0.37885386, -0.775237  ],
       [-0.08473441, -0.51496559],
       [ 0.99370357, -1.1511846 ],
       [-0.86905295, -0.775237  ],
       [-0.18277423, -0.51496559],
       [-1.06513258, -0.45712749],
       [-1.1631724 ,  1.39369146]])
```

In [11]:
```
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

In [12]:
```
X_test_scaled
```

Out[12]:

	Age	EstimatedSalary
0	-0.771013	0.497201
1	0.013305	-0.572804
2	-0.280814	0.150172
3	-0.771013	0.265849
4	-0.280814	-0.572804
...	...	...
115	1.091743	-0.139018
116	0.699584	1.769639
117	-0.672973	0.555039
118	0.797624	0.352606
119	0.895664	-0.543885

120 rows × 2 columns

In [13]:
```
X_train_scaled
```

Out[13]:

	Age	EstimatedSalary
0	-1.163172	-1.584970
1	2.170181	0.930987
2	0.013305	1.220177
3	0.209385	1.075582
4	0.405465	-0.486047
...	...	...
275	0.993704	-1.151185
276	-0.869053	-0.775237
277	-0.182774	-0.514966
278	-1.065133	-0.457127
279	-1.163172	1.393691

280 rows × 2 columns

In [14]:
```
np.round(X_train.describe(), 1)
```

Out[14]:

	Age	EstimatedSalary
count	280.0	280.0
mean	37.9	69807.1
std	10.2	34641.2
min	18.0	15000.0
25%	30.0	43000.0
50%	37.0	70500.0
75%	46.0	88000.0
max	60.0	150000.0

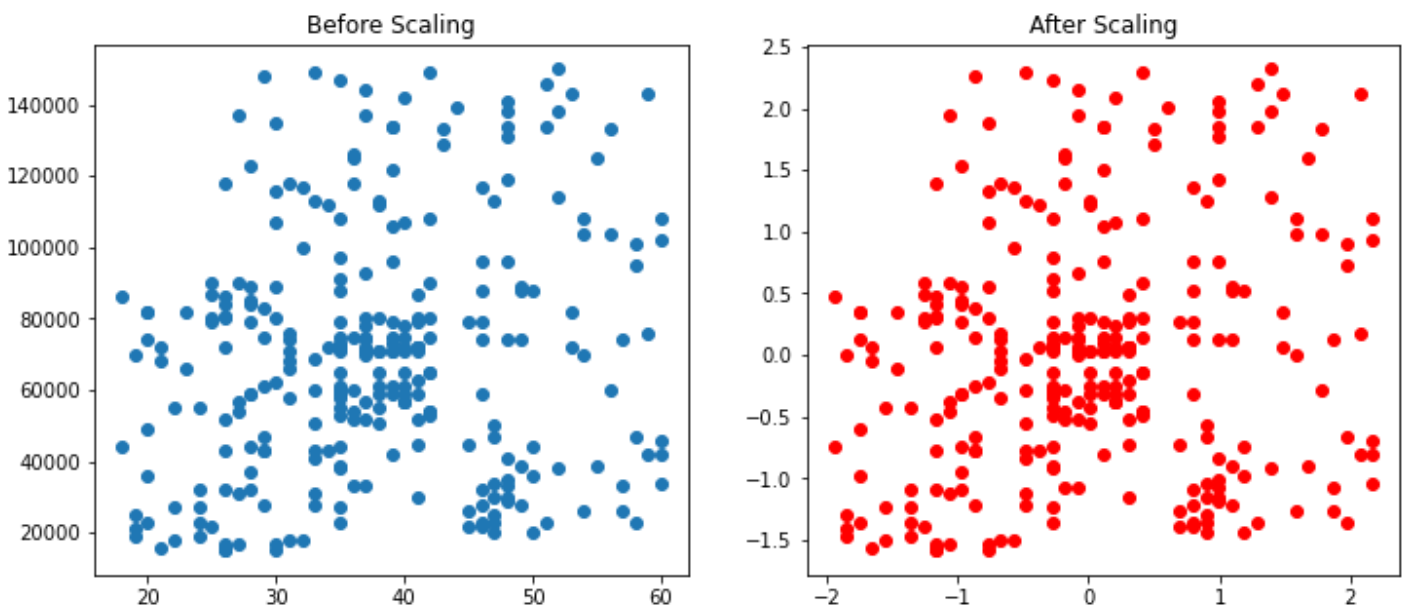
In [15]:
```
np.round(X_train_scaled.describe(), 1)
```

Out[15]:

	Age	EstimatedSalary
count	280.0	280.0
mean	0.0	0.0
std	1.0	1.0
min	-1.9	-1.6
25%	-0.8	-0.8
50%	-0.1	0.0
75%	0.8	0.5
max	2.2	2.3

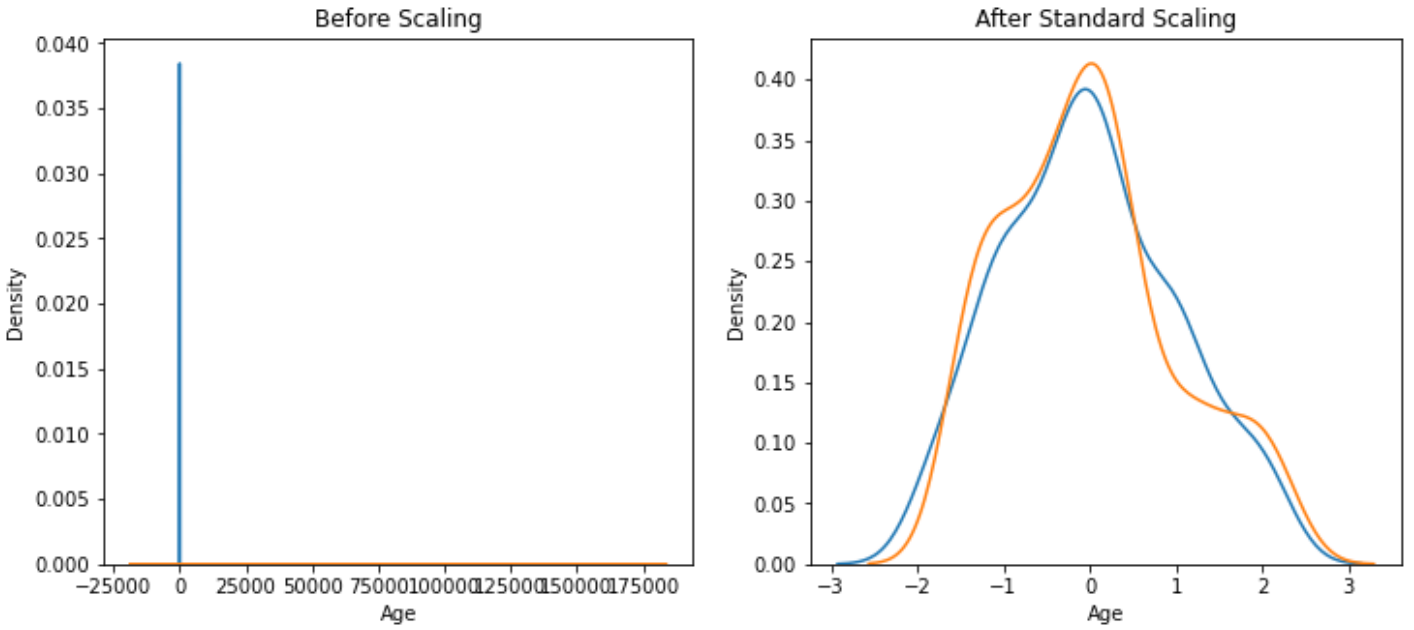
Now set the scaling

In [16]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

ax1.scatter(X_train['Age'], X_train['EstimatedSalary'])
ax1.set_title("Before Scaling")
ax2.scatter(X_train_scaled['Age'], X_train_scaled['EstimatedSalary'],color='red')
ax2.set_title("After Scaling")
plt.show()
```



In [17]:

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Before Scaling')
sns.kdeplot(X_train['Age'], ax=ax1)
sns.kdeplot(X_train['EstimatedSalary'], ax=ax1)

# after scaling
ax2.set_title('After Standard Scaling')
sns.kdeplot(X_train_scaled['Age'], ax=ax2)
sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2)
plt.show()
```

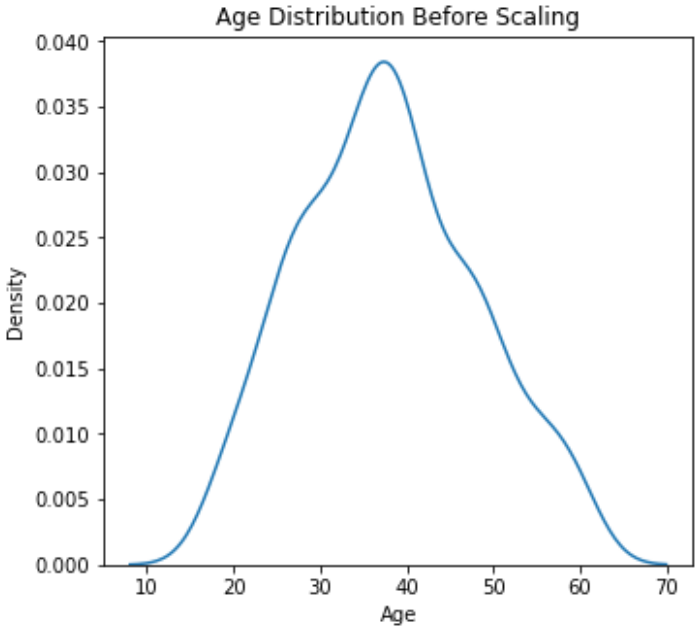


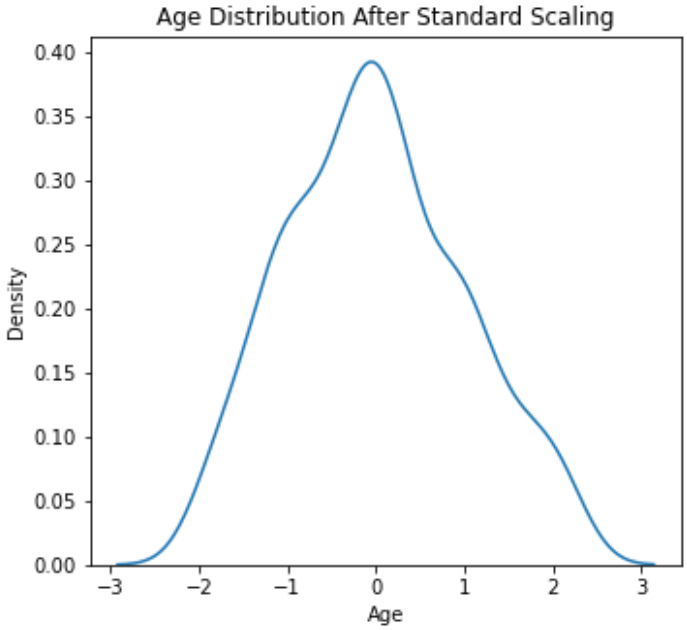
Comparison of Distributions

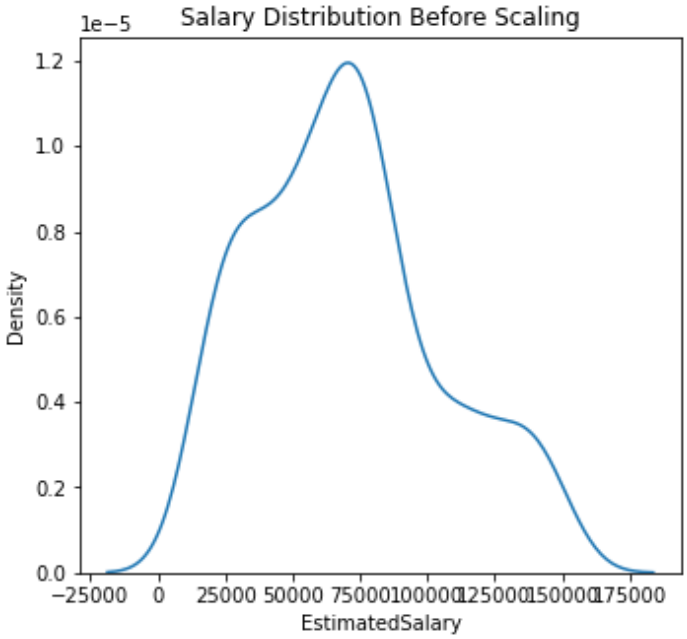
In [18]:

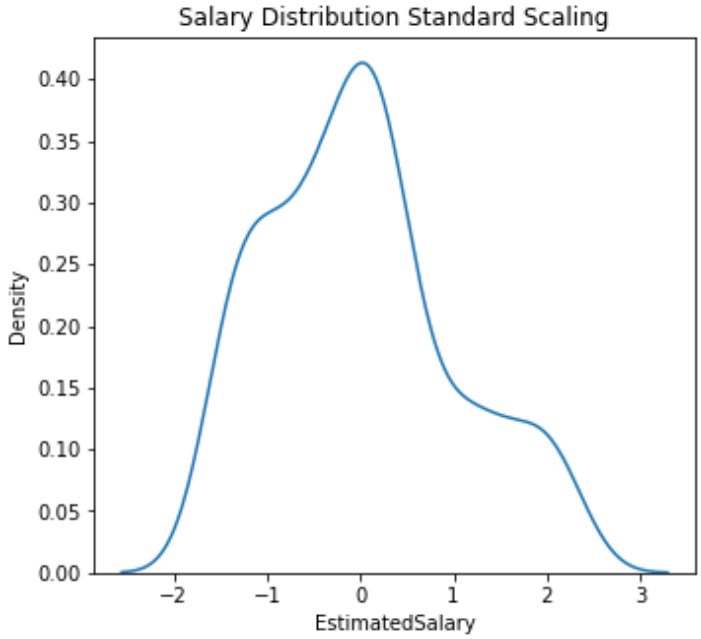
```
#Age  Dist.
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Age Distribution Before Scaling')
sns.kdeplot(X_train['Age'], ax=ax1)
```

```
# after scaling
ax2.set_title('Age Distribution After Standard Scaling')
sns.kdeplot(X_train_scaled['Age'], ax=ax2)
plt.show()

#Salry Dist.
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Salary Distribution Before Scaling')
sns.kdeplot(X_train['EstimatedSalary'], ax=ax1)

# after scaling
ax2.set_title('Salary Distribution Standard Scaling')
sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2)
plt.show()
```









Importance of Scaling

From Logistis Regression

In [19]:
```
from sklearn.linear_model import LogisticRegression
```

In [20]:
```
#Creat lr function
lr = LogisticRegression()
lr_scaled = LogisticRegression()
```

In [21]:
```
#Fit the train and scaled datsset of LR
lr.fit(X_train,Y_train)
lr_scaled.fit(X_train_scaled,Y_train)
```

Out[21]:
```
LogisticRegression()
```

In [22]:
```
#Now it's time to prediction
y_pred = lr.predict(X_test)
y_pred_scaled = lr_scaled.predict(X_test_scaled)
```

In [23]:
```
#Findout Acc. Score
from sklearn.metrics import accuracy_score
```

In [24]:
```
print("Actual",accuracy_score(Y_test,y_pred))
print("Scaled",accuracy_score(Y_test,y_pred_scaled))
```

```
Actual 0.6583333333333333
Scaled 0.8666666666666667
```

From DecisionTreeClassifier

In [25]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [26]:
```
#Creat DT function
dt = DecisionTreeClassifier()
dt_scaled = DecisionTreeClassifier()
```

In [27]:
```
#Fit the train and scaled datsset of LR
dt.fit(X_train,Y_train)
dt_scaled.fit(X_train_scaled,Y_train)
```

Out[27]:
```
DecisionTreeClassifier()
```

In [28]:
```
#Now it's time to prediction
y_pred = dt.predict(X_test)
y_pred_scaled = dt_scaled.predict(X_test_scaled)
```

In [29]:
```
#Findout Acc. Score
from sklearn.metrics import accuracy_score
```

In [30]:
```
#It's better than LogisticsRegression
print("Actual",accuracy_score(Y_test,y_pred))
print("Scaled",accuracy_score(Y_test,y_pred_scaled))
```

```
Actual 0.875
Scaled 0.8666666666666667
```

In [31]:
```
#See the standrization our data
df.describe()
```

Out[31]:

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

In [32]:
```
#Now let's check the model.

#Dummy values
df = df.append(pd.DataFrame({'Age':[10,85,70],'EstimatedSalary':[2500,450000,250000],'Purc
```

In [33]:
```
df
```

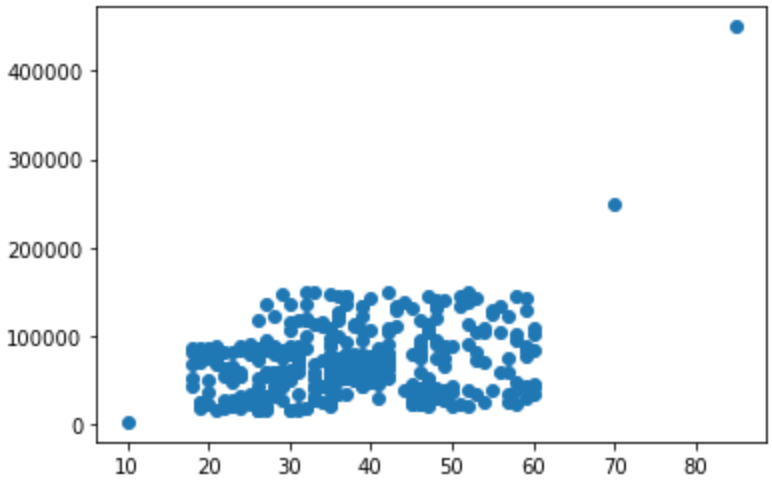
Out[33]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
398	36	33000	0
399	49	36000	1
400	10	2500	0
401	85	450000	1
402	70	250000	1

403 rows × 3 columns

In [34]:
```
plt.scatter(df['Age'], df['EstimatedSalary'])
```

Out[34]:
```
<matplotlib.collections.PathCollection at 0x21a2f4d2790>
```



In [35]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df.drop('Purchased', axis=1),
                                                    df['Purchased'],
                                                    test_size = 0.3,
                                                    random_state=0)
X_train.shape, X_test.shape
```

Out[35]:
```
((282, 2), (121, 2))
```

In [36]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# fit the scaler to the train set, it will learn the parameters
scaler.fit(X_train)

# transform train and test sets
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [37]:
```
scaler.mean_
```

Out[37]:
```
array([3.81489362e+01, 6.98528369e+04])
```

In [38]:
```
X_train
```

Out[38]:

	Age	EstimatedSalary
179	31	34000
219	59	143000
302	37	137000
49	31	89000
241	38	59000
...	...	...
323	48	30000
192	29	43000
117	36	52000

	Age	EstimatedSalary
47	27	54000
172	26	118000

282 rows × 2 columns

In [39]:
```
X_train_scaled
```

Out[39]:
```
array([[-6.60321806e-01, -8.84542567e-01],
       [ 1.92593860e+00,  1.80464881e+00],
       [-1.06123147e-01,  1.65661993e+00],
       [-6.60321806e-01,  4.72388862e-01],
       [-1.37567043e-02, -2.67755554e-01],
       [ 1.55647283e+00,  1.48391956e+00],
       [-1.12215402e+00, -1.35330070e+00],
       [ 8.17541284e-01, -4.89798878e-01],
       [ 1.37173994e+00, -8.84542567e-01],
       [ 1.00227417e+00,  1.75530585e+00],
       [ 1.74120572e+00,  1.28654772e+00],
       [ 2.01830504e+00,  7.93118109e-01],
       [-1.30688691e+00, -1.25461477e+00],
       [ 7.25174841e-01,  4.47717381e-01],
       [-3.83222477e-01,  1.11384736e+00],
       [-9.37421136e-01, -2.67755554e-01],
       [ 1.00227417e+00, -7.61185164e-01],
       [-7.52688249e-01,  2.25674057e-01],
       [ 8.17541284e-01,  9.16475511e-01],
       [ 6.32808398e-01, -1.08191441e+00],
       [-1.37567043e-02, -4.89798878e-01],
       [-2.90856034e-01, -4.89798878e-01],
       [-6.60321806e-01,  1.02316654e-01],
       [ 3.55709068e-01,  2.50345537e-01],
       [-1.98489591e-01, -4.89798878e-01],
       [-4.75588920e-01,  1.95267769e+00],
       [ 6.32808398e-01,  2.25674057e-01],
       [ 2.63342625e-01,  5.29736931e-02],
       [ 3.55709068e-01,  1.95267769e+00],
       [-2.90856034e-01,  1.26988135e-01],
       [ 3.55709068e-01, -4.15784437e-01],
       [-1.98489591e-01,  1.26988135e-01],
       [-1.67635268e+00,  2.99688498e-01],
       [-1.37567043e-02, -2.18412593e-01],
       [ 7.25174841e-01,  6.45089226e-01],
       [-1.67635268e+00,  2.99688498e-01],
       [ 6.32808398e-01, -6.13156281e-01],
       [ 3.55709068e-01, -1.19726671e-01],
       [-2.90856034e-01,  5.21731823e-01],
       [ 9.09907727e-01,  1.02316654e-01],
       [ 1.27937350e+00, -1.20527181e+00],
       [-1.58398624e+00, -4.57122290e-02],
       [-9.37421136e-01, -6.37827761e-01],
       [ 9.09907727e-01, -9.09214047e-01],
       [-1.06123147e-01, -9.09214047e-01],
       [-1.02978758e+00,  4.97060342e-01],
       [ 1.00227417e+00, -1.19726671e-01],
       [-9.37421136e-01,  1.31121920e+00],
       [-6.60321806e-01,  1.18786180e+00],
       [-1.21452047e+00,  4.23045901e-01],
       [-2.90856034e-01,  2.83022126e-02],
       [ 1.18700706e+00, -1.15592885e+00],
       [ 2.63342625e-01,  5.29736931e-02],
       [ 7.86097388e-02, -2.67755554e-01],
```

```
[ 1.00227417e+00,  3.98374420e-01],
[ 2.01830504e+00, -6.87170722e-01],
[-1.12215402e+00,  3.98374420e-01],
[ 8.17541284e-01,  1.06450439e+00],
[-1.21452047e+00,  2.25674057e-01],
[-1.98489591e-01, -9.09214047e-01],
[-9.37421136e-01,  4.72388862e-01],
[ 3.55709068e-01, -1.44398151e-01],
[-4.75588920e-01,  1.06450439e+00],
[-7.52688249e-01,  9.16475511e-01],
[ 1.74120572e+00, -9.09214047e-01],
[-1.06123147e-01,  2.01002576e-01],
[-1.37567043e-02,  1.03983291e+00],
[-6.60321806e-01, -1.27928625e+00],
[-6.60321806e-01,  2.83022126e-02],
[-1.06123147e-01,  1.26988135e-01],
[ 2.63342625e-01, -1.69069632e-01],
[-1.86108557e+00, -6.37827761e-01],
[ 9.09907727e-01,  1.21253328e+00],
[ 2.01830504e+00, -8.84542567e-01],
[ 8.17541284e-01, -9.83228489e-01],
[-1.12215402e+00, -6.62499242e-01],
[ 2.63342625e-01,  2.50345537e-01],
[ 7.25174841e-01, -2.67755554e-01],
[ 2.01830504e+00,  9.41146992e-01],
[ 1.83357216e+00, -1.15592885e+00],
[-2.90856034e-01, -2.43084073e-01],
[-9.37421136e-01, -2.67755554e-01],
[ 1.37173994e+00,  1.80464881e+00],
[-2.90856034e-01, -3.17098515e-01],
[-1.37567043e-02,  2.83022126e-02],
[ 1.92593860e+00,  4.47717381e-01],
[ 3.55709068e-01,  3.63073206e-03],
[ 8.17541284e-01, -1.15592885e+00],
[-2.90856034e-01, -1.05724293e+00],
[ 1.83357216e+00,  7.68446628e-01],
[ 8.17541284e-01,  8.67132550e-01],
[ 9.09907727e-01,  6.45089226e-01],
[-1.39925335e+00,  2.99688498e-01],
[ 7.25174841e-01, -1.15592885e+00],
[ 1.70976182e-01, -2.18412593e-01],
[-9.37421136e-01,  3.49031459e-01],
[ 1.37173994e+00,  5.29736931e-02],
[-1.12215402e+00,  3.49031459e-01],
[-8.45054693e-01, -1.03257145e+00],
[-1.06123147e-01,  2.83022126e-02],
[-1.49161979e+00, -3.66441476e-01],
[ 6.32808398e-01, -9.33885528e-01],
[-2.90856034e-01, -3.66441476e-01],
[ 1.74120572e+00,  1.02316654e-01],
[-1.02978758e+00, -1.30395774e+00],
[ 7.25174841e-01, -1.03257145e+00],
[-1.06123147e-01, -1.93741112e-01],
[-1.12215402e+00, -1.30395774e+00],
[ 9.09907727e-01,  1.58260549e+00],
[ 1.09464061e+00, -6.37827761e-01],
[-1.30688691e+00, -1.05724293e+00],
[-5.67955363e-01, -1.27928625e+00],
[-2.90856034e-01,  1.26988135e-01],
[-2.90856034e-01, -2.92427034e-01],
[ 8.17541284e-01, -8.84542567e-01],
[ 1.18700706e+00,  1.87866325e+00],
[ 1.70976182e-01,  5.29736931e-02],
[-5.67955363e-01,  7.43775148e-01],
[-1.76871912e+00, -1.20527181e+00],
[-1.21452047e+00,  4.97060342e-01],
```

```
[-2.90856034e-01,  4.47717381e-01],
[-9.37421136e-01, -9.33885528e-01],
[-2.60001711e+00, -1.66169420e+00],
[ 1.70976182e-01, -2.67755554e-01],
[-1.37567043e-02, -4.65127398e-01],
[-2.90856034e-01,  5.29736931e-02],
[ 1.70976182e-01,  1.77997733e+00],
[ 2.63342625e-01,  4.23045901e-01],
[ 7.86097388e-02,  2.83022126e-02],
[-1.67635268e+00,  1.02316654e-01],
[-8.45054693e-01,  1.26988135e-01],
[-6.60321806e-01,  1.51659615e-01],
[ 8.17541284e-01, -1.10658589e+00],
[ 1.92593860e+00,  1.51659615e-01],
[-3.83222477e-01,  1.03983291e+00],
[-1.37567043e-02,  2.50345537e-01],
[-2.90856034e-01, -2.18412593e-01],
[-1.76871912e+00, -1.10658589e+00],
[-1.06123147e-01,  2.50345537e-01],
[-2.90856034e-01,  1.76331096e-01],
[ 2.63342625e-01, -4.40455917e-01],
[ 2.63342625e-01, -4.40455917e-01],
[ 7.86097388e-02,  6.45089226e-01],
[ 9.09907727e-01, -1.00789997e+00],
[-1.98489591e-01,  1.38523364e+00],
[-1.37567043e-02, -4.89798878e-01],
[-1.02978758e+00, -3.17098515e-01],
[ 1.27937350e+00, -7.85856644e-01],
[ 7.86097388e-02, -2.18412593e-01],
[-1.49161979e+00, -1.05724293e+00],
[-4.75588920e-01, -2.43084073e-01],
[-1.06123147e-01,  2.25674057e-01],
[ 5.40441955e-01,  1.70596289e+00],
[-1.37567043e-02, -2.18412593e-01],
[-5.67955363e-01,  1.16319032e+00],
[ 7.25174841e-01, -7.11842203e-01],
[-7.52688249e-01,  2.50345537e-01],
[ 1.83357216e+00,  6.20417745e-01],
[-1.12215402e+00, -4.40455917e-01],
[ 1.83357216e+00, -7.85856644e-01],
[-1.30688691e+00, -3.66441476e-01],
[ 1.70976182e-01,  1.26988135e-01],
[-4.75588920e-01, -1.03257145e+00],
[ 1.64883927e+00, -2.43084073e-01],
[-1.49161979e+00, -1.27928625e+00],
[-4.75588920e-01, -4.65127398e-01],
[ 4.48075511e-01,  1.55793400e+00],
[-1.30688691e+00, -9.33885528e-01],
[ 7.25174841e-01, -1.18060033e+00],
[ 1.27937350e+00,  1.68129141e+00],
[-1.98489591e-01, -2.43084073e-01],
[ 1.74120572e+00, -2.43084073e-01],
[-2.90856034e-01, -4.15784437e-01],
[-1.06123147e-01,  1.82932029e+00],
[-1.39925335e+00, -9.50551900e-02],
[ 7.86097388e-02,  1.26988135e-01],
[-6.60321806e-01, -2.92427034e-01],
[-4.75588920e-01, -7.11842203e-01],
[ 6.32808398e-01, -1.18060033e+00],
[-7.52688249e-01, -1.35330070e+00],
[-1.76871912e+00, -1.25461477e+00],
[ 1.00227417e+00,  1.02316654e-01],
[-1.37567043e-02, -1.19726671e-01],
[-2.90856034e-01,  7.76451736e-02],
[ 1.70976182e-01, -2.43084073e-01],
[ 9.09907727e-01, -9.09214047e-01],
```

```
[-1.30688691e+00, -1.15592885e+00],
[ 2.63342625e-01,  5.29736931e-02],
[-8.45054693e-01,  3.24359979e-01],
[ 7.86097388e-02,  7.76451736e-02],
[-2.90856034e-01, -6.37827761e-01],
[-1.98489591e-01,  7.19103667e-01],
[-8.45054693e-01, -5.63813320e-01],
[-6.60321806e-01, -4.57122290e-02],
[ 3.55709068e-01, -3.91112956e-01],
[-7.52688249e-01,  1.60727697e+00],
[ 1.27937350e+00,  1.08917587e+00],
[ 1.09464061e+00, -8.35199606e-01],
[ 1.64883927e+00,  1.55793400e+00],
[-8.45054693e-01, -2.18412593e-01],
[-7.52688249e-01,  4.72388862e-01],
[-1.12215402e+00, -1.32862922e+00],
[-4.75588920e-01, -9.58557008e-01],
[ 4.48075511e-01,  1.45924808e+00],
[ 4.32746612e+00,  9.37879333e+00],
[ 1.55647283e+00,  1.36056216e+00],
[ 9.09907727e-01,  1.50859104e+00],
[ 2.63342625e-01,  2.83022126e-02],
[-7.52688249e-01, -1.93741112e-01],
[-1.06123147e-01,  5.29736931e-02],
[ 7.86097388e-02,  1.76331096e-01],
[ 1.46410639e+00, -1.08191441e+00],
[-7.52688249e-01,  1.13851884e+00],
[-1.67635268e+00, -5.14470359e-01],
[-1.06123147e-01,  1.02316654e-01],
[ 2.63342625e-01, -2.67755554e-01],
[ 1.00227417e+00,  4.72388862e-01],
[-9.37421136e-01,  2.25674057e-01],
[ 1.37173994e+00,  2.99688498e-01],
[ 1.70976182e-01, -3.17098515e-01],
[ 1.70976182e-01,  2.83022126e-02],
[-2.90856034e-01,  9.41146992e-01],
[-1.58398624e+00,  5.29736931e-02],
[ 2.01830504e+00,  3.24359979e-01],
[ 7.86097388e-02,  8.91804031e-01],
[ 1.46410639e+00,  3.63073206e-03],
[-1.12215402e+00,  5.29736931e-02],
[-2.90856034e-01, -1.15592885e+00],
[-1.37567043e-02,  1.06450439e+00],
[-7.52688249e-01, -1.30395774e+00],
[ 7.86097388e-02,  1.58260549e+00],
[-8.45054693e-01, -6.62499242e-01],
[-4.75588920e-01, -6.62499242e-01],
[-2.90856034e-01, -7.85856644e-01],
[ 2.63342625e-01, -6.13156281e-01],
[ 2.63342625e-01,  5.29736931e-02],
[ 7.86097388e-02,  1.58260549e+00],
[-1.02978758e+00,  1.65661993e+00],
[-1.58398624e+00, -1.32862922e+00],
[-1.12215402e+00, -9.33885528e-01],
[-6.60321806e-01, -9.50551900e-02],
[ 7.25174841e-01, -9.33885528e-01],
[ 2.63342625e-01,  2.25674057e-01],
[ 1.92593860e+00, -1.00789997e+00],
[ 2.63342625e-01, -9.83228489e-01],
[-1.06123147e-01,  5.71074784e-01],
[ 1.92593860e+00,  1.48391956e+00],
[-1.21452047e+00, -1.18060033e+00],
[-9.37421136e-01, -8.10528125e-01],
[-1.37567043e-02, -3.66441476e-01],
[-1.98489591e-01, -3.91112956e-01],
[-1.67635268e+00, -8.35199606e-01],
```

```
       [ 1.64883927e+00,  8.42461070e-01],
       [ 1.70976182e-01, -3.17098515e-01],
       [ 3.55709068e-01,  9.41146992e-01],
       [-1.67635268e+00, -1.15592885e+00],
       [ 1.70976182e-01, -1.19726671e-01],
       [ 8.17541284e-01, -1.22994329e+00],
       [-1.86108557e+00,  3.98374420e-01],
       [-2.90856034e-01,  2.25674057e-01],
       [ 7.86097388e-02,  2.83022126e-02],
       [-3.83222477e-01,  5.29736931e-02],
       [ 9.09907727e-01, -8.59871086e-01],
       [-1.02978758e+00, -9.58557008e-01],
       [-1.76871912e+00,  3.63073206e-03],
       [ 7.86097388e-02,  2.25674057e-01],
       [-1.12215402e+00,  2.75017018e-01],
       [-1.21452047e+00,  2.50345537e-01],
       [-9.37421136e-01,  3.73702940e-01],
       [ 1.55647283e+00, -7.61185164e-01],
       [ 1.09464061e+00,  4.47717381e-01],
       [ 1.00227417e+00,  4.47717381e-01],
       [ 1.27937350e+00,  1.97734917e+00],
       [-2.90856034e-01, -1.19726671e-01],
       [ 3.55709068e-01, -3.91112956e-01],
       [-3.83222477e-01, -6.62499242e-01],
       [-1.06123147e-01, -4.40455917e-01],
       [ 9.09907727e-01, -9.83228489e-01],
       [-8.45054693e-01, -6.62499242e-01],
       [-1.98489591e-01, -4.40455917e-01],
       [-1.02978758e+00, -3.91112956e-01],
       [-1.12215402e+00,  1.18786180e+00]])
```

In [40]:

```
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

In [41]:

```
print(X_train_scaled)
print(X_test_scaled)
```

```
          Age  EstimatedSalary
0   -0.660322        -0.884543
1    1.925939         1.804649
2   -0.106123         1.656620
3   -0.660322         0.472389
4   -0.013757        -0.267756
..        ...              ...
277  0.909908        -0.983228
278 -0.845055        -0.662499
279 -0.198490        -0.440456
280 -1.029788        -0.391113
281 -1.122154         1.187862

[282 rows x 2 columns]
          Age  EstimatedSalary
0   -0.290856        -1.180600
1    0.355709         0.126988
2   -0.567955         1.163190
3   -0.937421         0.423046
4   -1.306887        -0.292427
..        ...              ...
116 -0.290856        -1.106586
117 -0.383222        -1.106586
118 -0.290856        -0.489799
119 -0.106123        -0.317099
120  0.817541        -0.514470
```

```
[121 rows x 2 columns]
```

In [42]:

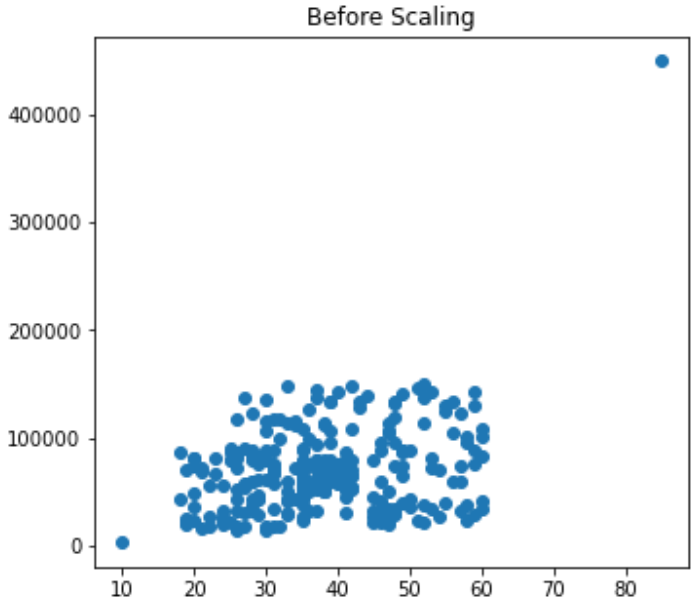
```
print(np.round(X_train.describe(), 1))
print(np.round(X_train_scaled.describe(), 1))
```

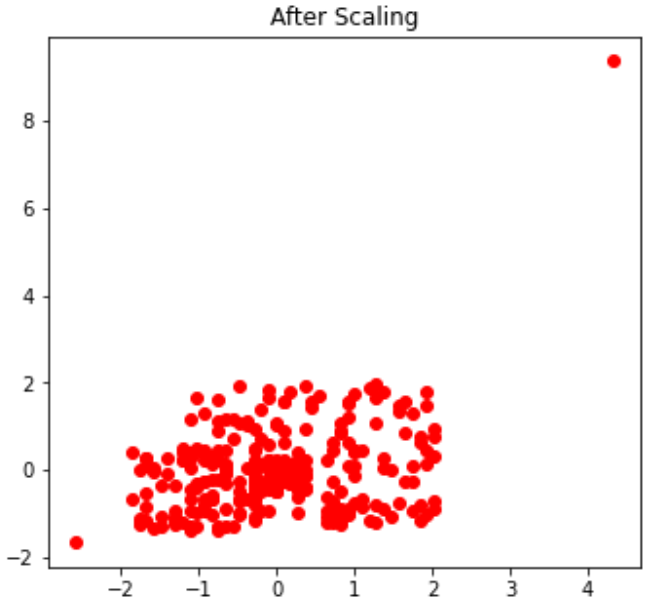
```
         Age  EstimatedSalary
count  282.0            282.0
mean    38.1          69852.8
std     10.8          40604.7
min     10.0           2500.0
25%     30.0          43000.0
50%     37.0          68000.0
75%     46.0          86750.0
max     85.0         450000.0
         Age  EstimatedSalary
count  282.0            282.0
mean     0.0             -0.0
std      1.0              1.0
min     -2.6             -1.7
25%     -0.8             -0.7
50%     -0.1             -0.0
75%      0.7              0.4
max      4.3              9.4
```

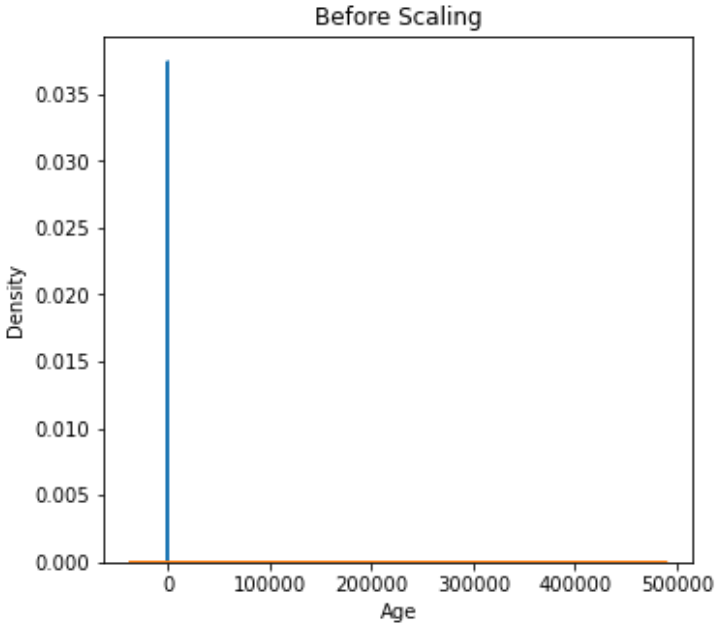
In [43]:

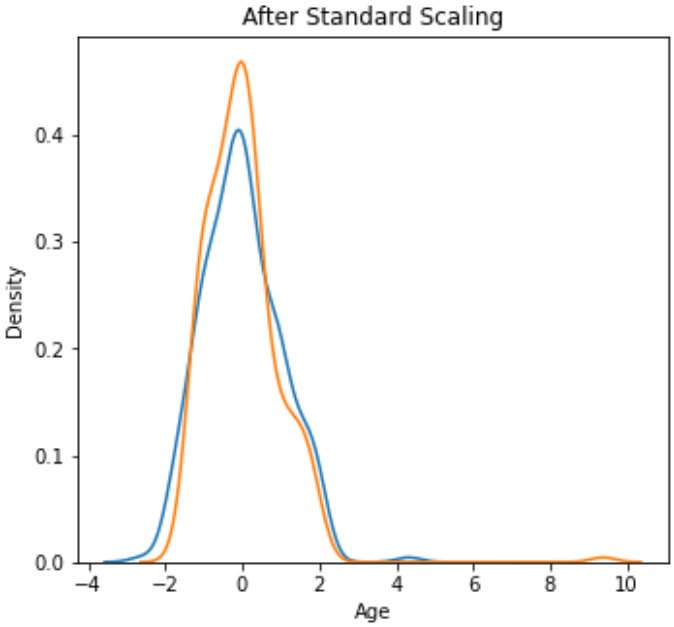
```
#A
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))
ax1.scatter(X_train['Age'], X_train['EstimatedSalary'])
ax1.set_title("Before Scaling")
ax2.scatter(X_train_scaled['Age'], X_train_scaled['EstimatedSalary'],color='red')
ax2.set_title("After Scaling")
plt.show()

#B
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))
# before scaling
ax1.set_title('Before Scaling')
sns.kdeplot(X_train['Age'], ax=ax1)
sns.kdeplot(X_train['EstimatedSalary'], ax=ax1)

# after scaling
ax2.set_title('After Standard Scaling')
sns.kdeplot(X_train_scaled['Age'], ax=ax2)
sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2)
plt.show()

#C
#Age  Dist.
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Age Distribution Before Scaling')
sns.kdeplot(X_train['Age'], ax=ax1)

# after scaling
ax2.set_title('Age Distribution After Standard Scaling')
sns.kdeplot(X_train_scaled['Age'], ax=ax2)
plt.show()

#Salry Dist.
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

# before scaling
ax1.set_title('Salary Distribution Before Scaling')
```

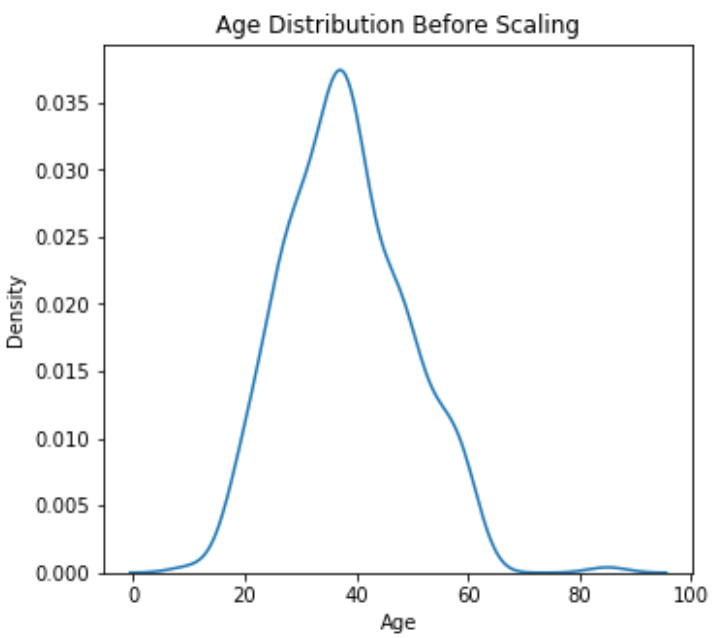
```
sns.kdeplot(X_train['EstimatedSalary'], ax=ax1)

# after scaling
ax2.set_title('Salary Distribution Standard Scaling')
sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2)
plt.show()
```

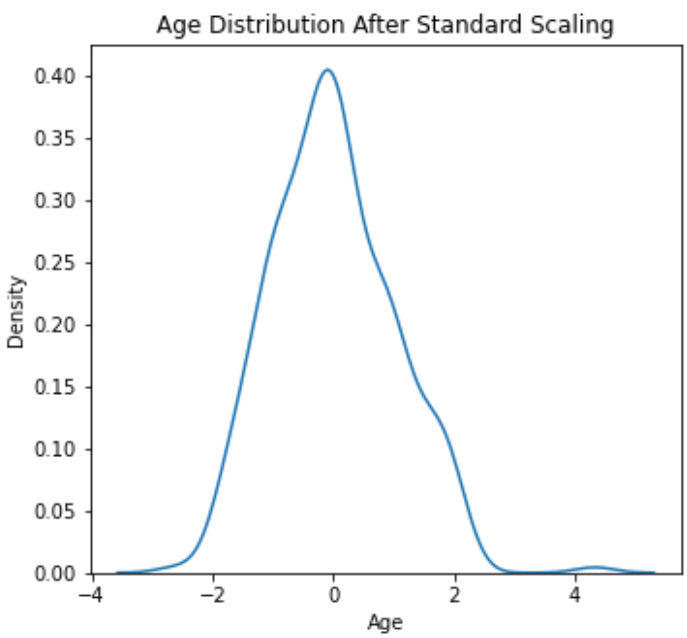


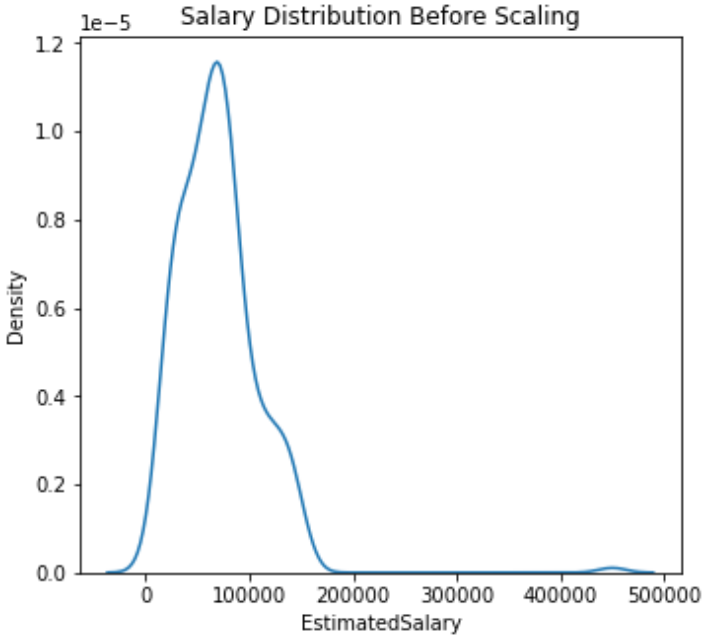


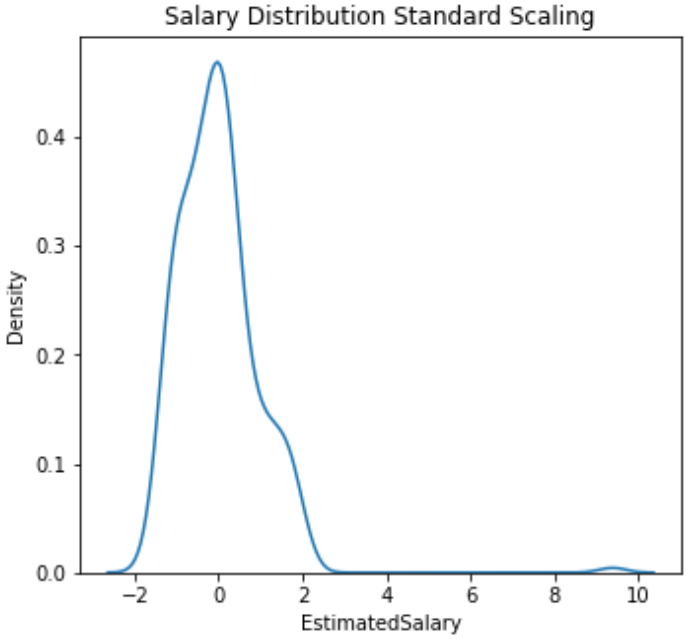












In [44]:

```
#From Logistics regression

from sklearn.linear_model import LogisticRegression

#Creat lr function
lr = LogisticRegression()
lr_scaled = LogisticRegression()

#Fit the train and scaled datsset of LR
lr.fit(X_train,Y_train)
lr_scaled.fit(X_train_scaled,Y_train)

#Now it's time to prediction
y_pred = lr.predict(X_test)
y_pred_scaled = lr_scaled.predict(X_test_scaled)

#Findout Acc. Score
from sklearn.metrics import accuracy_score

print("Actual",accuracy_score(Y_test,y_pred))
print("Scaled",accuracy_score(Y_test,y_pred_scaled))
```

```
Actual 0.6446280991735537
```

```
Scaled 0.8512396694214877
```

In [45]:

```python
#From DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier

#Creat lr function
dt = DecisionTreeClassifier()
dt_scaled = DecisionTreeClassifier()

#Fit the train and scaled datsset of LR
dt.fit(X_train,Y_train)
dt_scaled.fit(X_train_scaled,Y_train)

#Now it's time to prediction
y_pred = dt.predict(X_test)
y_pred_scaled = dt_scaled.predict(X_test_scaled)

#Findout Acc. Score
from sklearn.metrics import accuracy_score

print("Actual",accuracy_score(Y_test,y_pred))
print("Scaled",accuracy_score(Y_test,y_pred_scaled))
```

```
Actual 0.8760330578512396
Scaled 0.8760330578512396
```

📖 Conclusion

👉We observed without Outliers:

Our acc. score for with **LR** is: **Acctual=65 and Scaled=86**

Our accuracy score with **DT** is **Acctual=87 and Scaled=87**.

👉We saw this with Outliers:

Our accuracy score with **LR** is: **Acctual=64 and Scaled=85**.

Our accuracy score with **DT** is **Acctual=87 and Scaled=87**.

👉Outliers' Effects on Data:

1. Lowers quality. Outliers caused by measurement error imply poor data quality.
2. It skews the data's mean.
3. This leads to incorrect analyses and misleading insights.

👉Outlier detection method:

1. Domain Knowledge

2. Locate the Z-score

3. Interquartile Range

4. Boxplot 5. Scatter plot

6. Histogram

7. Clustering techniques Isolation Forest

9. The Local Outlier Factor

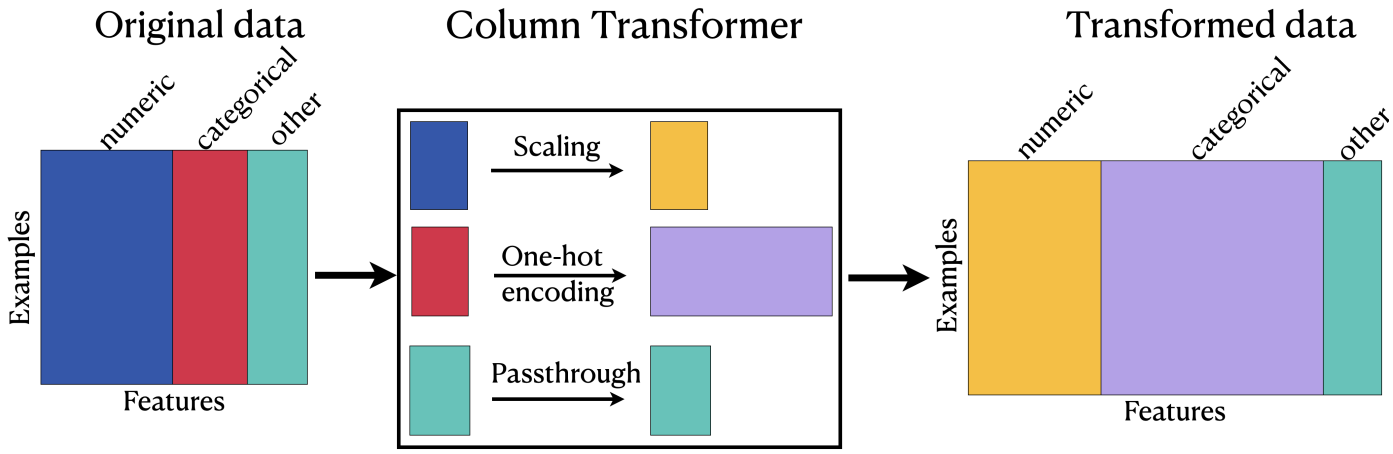
10.Minimum Covariance Determinant (MCD)

In [ ]:

Feature Engineering 101

Day 2

Column Transformer



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

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
```

In [2]:
```
df = pd.read_csv('covid_toy.csv')
```

In [3]:
```
df
```

Out[3]:

	age	gender	fever	cough	city	has_covid
0	60	Male	103.0	Mild	Kolkata	No
1	27	Male	100.0	Mild	Delhi	Yes
2	42	Male	101.0	Mild	Delhi	No
3	31	Female	98.0	Mild	Kolkata	No
4	65	Female	101.0	Mild	Mumbai	No
...	...	...	...	...	...	...
95	12	Female	104.0	Mild	Bangalore	No
96	51	Female	101.0	Strong	Kolkata	Yes
97	20	Female	101.0	Mild	Bangalore	No
98	5	Female	98.0	Strong	Mumbai	No
99	10	Female	98.0	Strong	Kolkata	Yes

100 rows × 6 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
age           0
gender        0
fever        10
cough         0
city          0
has_covid     0
dtype: int64
```

In [5]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(df.drop(columns=['has_covid']),df['has_co
                                                 test_size=0.3)
```

In [6]:
```
X_train
```

Out[6]:

	age	gender	fever	cough	city
39	50	Female	103.0	Mild	Kolkata
92	82	Female	102.0	Strong	Kolkata
11	65	Female	98.0	Mild	Mumbai
52	47	Female	100.0	Strong	Bangalore
28	16	Male	104.0	Mild	Kolkata
...	...	...	...	...	...
32	34	Female	101.0	Strong	Delhi
15	70	Male	103.0	Strong	Kolkata
19	42	Female	NaN	Strong	Bangalore
12	25	Female	99.0	Strong	Kolkata
33	26	Female	98.0	Mild	Kolkata

70 rows × 5 columns

Long method

In [7]:
```
# adding simple imputer to fever col
si = SimpleImputer()
X_train_fever = si.fit_transform(X_train[['fever']])

# also the test data
X_test_fever = si.fit_transform(X_test[['fever']])

X_train_fever.shape
```

Out[7]:
```
(70, 1)
```

In [8]:
```
# Ordinalencoding -> cough
oe = OrdinalEncoder(categories=[['Mild','Strong']])
X_train_cough = oe.fit_transform(X_train[['cough']])

# also the test data
X_test_cough = oe.fit_transform(X_test[['cough']])

X_train_cough.shape
```

Out[8]:
```
(70, 1)
```

In [9]:
```
print('City')
print(df['city'].value_counts())
print('Gender')
print(df['gender'].value_counts())
```

```
City
Kolkata      32
Bangalore    30
Delhi        22
Mumbai       16
Name: city, dtype: int64
Gender
Female    59
Male      41
Name: gender, dtype: int64
```

In [10]:
```
# Ordinalencoding -> cough
oe = OrdinalEncoder(categories=[['Mild','Strong']])
X_train_cough = oe.fit_transform(X_train[['cough']])

# also the test data
X_test_cough = oe.fit_transform(X_test[['cough']])

X_train_cough.shape
```

Out[10]:
```
(70, 1)
```

In [11]:
```
# OneHotEncoding -> gender,city
ohe = OneHotEncoder(drop='first',sparse=False)
X_train_gender_city = ohe.fit_transform(X_train[['gender','city']])
```

```
# also the test data
X_test_gender_city = ohe.fit_transform(X_test[['gender','city']])

X_train_gender_city.shape
```

Out[11]:
```
(70, 4)
```

In [12]:
```
# Extracting Age
X_train_age = X_train.drop(columns=['gender','fever','cough','city']).values

# also the test data
X_test_age = X_test.drop(columns=['gender','fever','cough','city']).values

X_train_age.shape
```

Out[12]:
```
(70, 1)
```

In [13]:
```
X_train_transformed = np.concatenate((X_train_age,X_train_fever,X_train_gender_city,X_trai
# also the test data
X_test_transformed = np.concatenate((X_test_age,X_test_fever,X_test_gender_city,X_test_cou

X_train_transformed.shape
```

Out[13]:
```
(70, 7)
```

Column TransFromer

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression

# Define the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['numerical_feature_1',
'numerical_feature_2']),
        ('cat', OneHotEncoder(), ['categorical_feature'])
    ])

# Define the pipeline
pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])

# Fit the pipeline to the training data
pipe.fit(X_train, y_train)

# Use the pipeline to predict on the test data
y_pred = pipe.predict(X_test)
```

In [14]:
```
from sklearn.compose import ColumnTransformer
```

In [15]:
```
transformer = ColumnTransformer(transformers=[
    ('tnf1',SimpleImputer(),['fever']),
    ('tnf2',OrdinalEncoder(categories=[['Mild','Strong']]),['cough']),
    ('tnf3',OneHotEncoder(sparse=False,drop='first'),['gender','city'])
],remainder='passthrough')
```

In [16]:
```
transformer.fit_transform(X_train)
```

Out[16]:
```
array([[103.  ,   0.  ,   0.  ,   0.  ,   1.  ,   0.  ,
         50.  ],
       [102.  ,   1.  ,   0.  ,   0.  ,   1.  ,   0.  ,
         82.  ],
       [ 98.  ,   0.  ,   0.  ,   0.  ,   0.  ,   1.  ,
         65.  ],
       [100.  ,   1.  ,   0.  ,   0.  ,   0.  ,   0.  ,
         47.  ],
       [104.  ,   0.  ,   1.  ,   0.  ,   1.  ,   0.  ,
         16.  ],
       [101.  ,   0.  ,   0.  ,   0.  ,   0.  ,   1.  ,
         81.  ],
       [104.  ,   0.  ,   0.  ,   0.  ,   1.  ,   0.  ,
         17.  ],
```

```
[101.     ,   0.     ,   0.     ,   0.     ,   0.     ,   1.     ,
  19.     ],
[103.     ,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
  83.     ],
[100.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
   5.     ],
[104.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
  18.     ],
[100.     ,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
  27.     ],
[101.     ,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
  19.     ],
[101.     ,   1.     ,   1.     ,   0.     ,   0.     ,   0.     ,
  47.     ],
[102.     ,   0.     ,   1.     ,   0.     ,   0.     ,   1.     ,
  74.     ],
[102.     ,   1.     ,   1.     ,   1.     ,   0.     ,   0.     ,
  20.     ],
[104.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
  25.     ],
[100.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
  10.     ],
[102.     ,   0.     ,   0.     ,   1.     ,   0.     ,   0.     ,
  49.     ],
[ 99.     ,   1.     ,   0.     ,   1.     ,   0.     ,   0.     ,
  59.     ],
[101.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
   8.     ],
[104.     ,   1.     ,   0.     ,   0.     ,   1.     ,   0.     ,
  54.     ],
[ 98.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
  31.     ],
[104.     ,   1.     ,   0.     ,   1.     ,   0.     ,   0.     ,
  34.     ],
[ 99.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
  72.     ],
[103.     ,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
  60.     ],
[100.90625,   0.     ,   1.     ,   0.     ,   0.     ,   1.     ,
  23.     ],
[ 98.     ,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
  24.     ],
[104.     ,   1.     ,   0.     ,   0.     ,   0.     ,   0.     ,
  56.     ],
[102.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
  64.     ],
[ 98.     ,   1.     ,   0.     ,   0.     ,   0.     ,   1.     ,
   5.     ],
[104.     ,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
  51.     ],
[100.     ,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
  55.     ],
[ 98.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
  73.     ],
[100.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
  80.     ],
[100.     ,   1.     ,   0.     ,   0.     ,   1.     ,   0.     ,
  11.     ],
[ 99.     ,   0.     ,   0.     ,   0.     ,   0.     ,   1.     ,
  60.     ],
[102.     ,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
   5.     ],
[ 98.     ,   1.     ,   1.     ,   0.     ,   0.     ,   1.     ,
  23.     ],
[100.     ,   1.     ,   0.     ,   0.     ,   0.     ,   0.     ,
  19.     ],
```

```
       [ 99.     ,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
         65.     ],
       [101.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         83.     ],
       [100.90625,   1.     ,   0.     ,   0.     ,   0.     ,   1.     ,
         34.     ],
       [103.     ,   0.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         73.     ],
       [100.     ,   1.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         13.     ],
       [100.90625,   1.     ,   1.     ,   0.     ,   1.     ,   0.     ,
         71.     ],
       [101.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         38.     ],
       [ 99.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         22.     ],
       [100.90625,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
         82.     ],
       [ 99.     ,   1.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         49.     ],
       [104.     ,   1.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         75.     ],
       [102.     ,   1.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         82.     ],
       [101.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         20.     ],
       [ 98.     ,   1.     ,   1.     ,   0.     ,   1.     ,   0.     ,
         34.     ],
       [ 98.     ,   1.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         40.     ],
       [101.     ,   1.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         68.     ],
       [104.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         12.     ],
       [101.     ,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
         15.     ],
       [103.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         16.     ],
       [ 98.     ,   1.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         10.     ],
       [101.     ,   0.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         49.     ],
       [ 99.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
         65.     ],
       [100.90625,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
         38.     ],
       [104.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
         51.     ],
       [ 98.     ,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
         83.     ],
       [101.     ,   1.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         34.     ],
       [103.     ,   1.     ,   1.     ,   0.     ,   1.     ,   0.     ,
         70.     ],
       [100.90625,   1.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         42.     ],
       [ 99.     ,   1.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         25.     ],
       [ 98.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         26.     ]])
```

In [17]:

```
transformer.transform(X_test)
```

Out[17]:

```
array([[100.90625,   0.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         75.     ],
```

```
       [ 99.     ,   0.     ,   0.     ,   0.     ,   0.     ,   1.     ,
         14.     ],
       [101.     ,   1.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         51.     ],
       [101.     ,   1.     ,   1.     ,   0.     ,   0.     ,   0.     ,
         14.     ],
       [ 98.     ,   1.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         71.     ],
       [100.90625,   1.     ,   0.     ,   0.     ,   0.     ,   1.     ,
         20.     ],
       [103.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         48.     ],
       [103.     ,   1.     ,   1.     ,   0.     ,   0.     ,   0.     ,
         46.     ],
       [100.     ,   0.     ,   1.     ,   0.     ,   0.     ,   0.     ,
         11.     ],
       [100.     ,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
         27.     ],
       [ 98.     ,   1.     ,   1.     ,   0.     ,   0.     ,   0.     ,
         12.     ],
       [ 98.     ,   1.     ,   0.     ,   0.     ,   0.     ,   1.     ,
         81.     ],
       [ 99.     ,   1.     ,   1.     ,   0.     ,   0.     ,   0.     ,
         66.     ],
       [ 98.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         64.     ],
       [104.     ,   0.     ,   1.     ,   0.     ,   0.     ,   1.     ,
         42.     ],
       [101.     ,   0.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         64.     ],
       [101.     ,   0.     ,   1.     ,   1.     ,   0.     ,   0.     ,
         42.     ],
       [ 98.     ,   1.     ,   0.     ,   0.     ,   0.     ,   1.     ,
         69.     ],
       [102.     ,   1.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         33.     ],
       [100.90625,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         84.     ],
       [104.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
          6.     ],
       [100.     ,   0.     ,   1.     ,   0.     ,   1.     ,   0.     ,
         27.     ],
       [102.     ,   0.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         69.     ],
       [ 98.     ,   0.     ,   0.     ,   1.     ,   0.     ,   0.     ,
         80.     ],
       [100.90625,   1.     ,   1.     ,   0.     ,   1.     ,   0.     ,
         79.     ],
       [103.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         69.     ],
       [104.     ,   0.     ,   1.     ,   0.     ,   0.     ,   1.     ,
         44.     ],
       [100.     ,   0.     ,   0.     ,   0.     ,   1.     ,   0.     ,
         19.     ],
       [102.     ,   1.     ,   0.     ,   0.     ,   0.     ,   0.     ,
         24.     ],
       [101.     ,   0.     ,   0.     ,   0.     ,   0.     ,   1.     ,
         65.     ]])
```

In [18]:

```
print(transformer.fit_transform(X_train).shape)
print(transformer.transform(X_test).shape)
```

```
(70, 7)
(30, 7)
```

In [19]:
```
print('X_Train')
print(X_train)

print('X_Test')
print(X_test)
```

```
X_Train
    age  gender  fever   cough       city
39   50  Female  103.0    Mild    Kolkata
92   82  Female  102.0  Strong    Kolkata
11   65  Female   98.0    Mild     Mumbai
52   47  Female  100.0  Strong  Bangalore
28   16    Male  104.0    Mild    Kolkata
..  ...     ...    ...     ...        ...
32   34  Female  101.0  Strong      Delhi
15   70    Male  103.0  Strong    Kolkata
19   42  Female    NaN  Strong  Bangalore
12   25  Female   99.0  Strong    Kolkata
33   26  Female   98.0    Mild    Kolkata

[70 rows x 5 columns]
X_Test
    age  gender  fever   cough       city
10   75  Female    NaN    Mild      Delhi
80   14  Female   99.0    Mild     Mumbai
96   51  Female  101.0  Strong    Kolkata
6    14    Male  101.0  Strong  Bangalore
22   71  Female   98.0  Strong    Kolkata
7    20  Female    NaN  Strong     Mumbai
79   48  Female  103.0    Mild    Kolkata
89   46    Male  103.0  Strong  Bangalore
78   11    Male  100.0    Mild  Bangalore
42   27    Male  100.0    Mild      Delhi
20   12    Male   98.0  Strong  Bangalore
61   81  Female   98.0  Strong     Mumbai
48   66    Male   99.0  Strong  Bangalore
18   64  Female   98.0    Mild  Bangalore
64   42    Male  104.0    Mild     Mumbai
9    64  Female  101.0    Mild      Delhi
2    42    Male  101.0    Mild      Delhi
84   69  Female   98.0  Strong     Mumbai
27   33  Female  102.0  Strong      Delhi
5    84  Female    NaN    Mild  Bangalore
59    6  Female  104.0    Mild    Kolkata
93   27    Male  100.0    Mild    Kolkata
65   69  Female  102.0    Mild  Bangalore
23   80  Female   98.0    Mild      Delhi
94   79    Male    NaN  Strong    Kolkata
16   69  Female  103.0    Mild    Kolkata
49   44    Male  104.0    Mild     Mumbai
26   19  Female  100.0    Mild    Kolkata
60   24  Female  102.0  Strong  Bangalore
4    65  Female  101.0    Mild     Mumbai
```

In [20]:
```
print(X_train.shape)
print(X_test.shape)
```

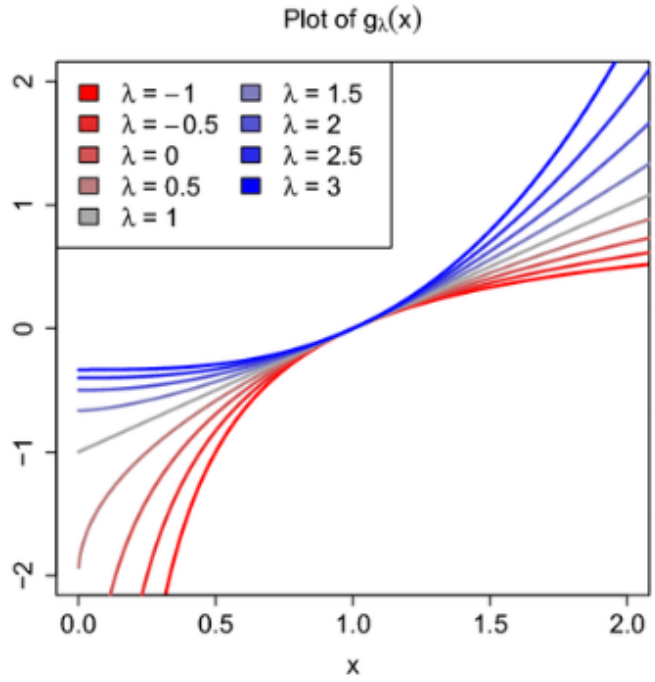
```
(70, 5)
(30, 5)
```

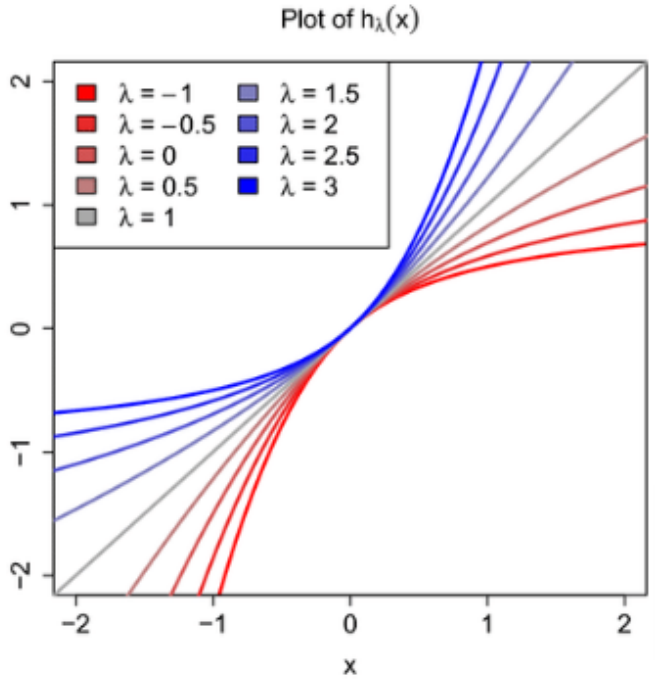
In [ ]:
```
```

Feature Engineering 101

Topic - 3

Power Transformer





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

import scipy.stats as stats


from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.preprocessing import PowerTransformer
```

In [2]:
```
df = pd.read_csv('concrete_data.csv')
```

In [3]:
```
df.sample(5)
```

Out[3]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
964	143.7	170.2	132.6	191.6	8.5	814.1	805.3	28	29.87
536	393.0	0.0	0.0	192.0	0.0	940.6	785.6	28	39.60
919	313.0	0.0	0.0	178.0	8.0	1000.0	822.0	28	25.10
795	525.0	0.0	0.0	189.0	0.0	1125.0	613.0	180	61.92
232	213.7	98.1	24.5	181.7	6.9	1065.8	785.4	56	50.77

In [4]:
```
df.shape
```

Out[4]:
```
(1030, 9)
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

In [6]:
```
df.describe()
```

Out[6]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000

In [7]:
```
X = df.drop(columns=['Strength'])
y = df.iloc[:,-1]
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=52)
```

Applying Regression without any transformation

In [9]:
```
lr = LinearRegression()

lr.fit(X_train,y_train)

y_pred = lr.predict(X_test)

r2_score(y_test,y_pred)
```

Out[9]:
```
0.601364413277667
```

In [10]:
```
# Cross checking with cross val score
lr = LinearRegression()
np.mean(cross_val_score(lr,X,y,scoring='r2'))
```

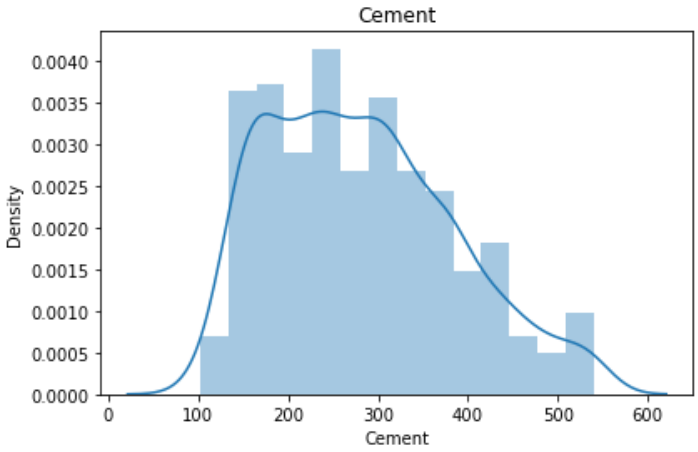
Out[10]:
```
0.4609940491662866
```

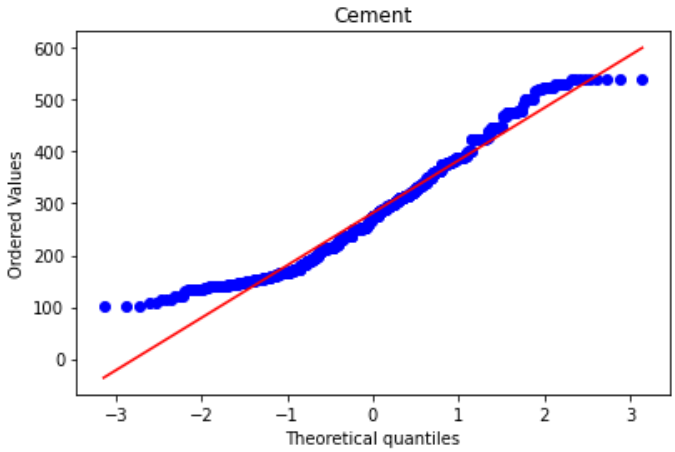
In [11]:
```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\programdata\anaconda3\lib\site-packages (0.1
1.2)
Requirement already satisfied: pandas>=0.23 in c:\programdata\anaconda3\lib\site-packages
(from seaborn) (1.3.4)
Requirement already satisfied: numpy>=1.15 in c:\programdata\anaconda3\lib\site-packages
(from seaborn) (1.20.3)
Requirement already satisfied: matplotlib>=2.2 in c:\programdata\anaconda3\lib\site-packag
es (from seaborn) (3.4.3)
Requirement already satisfied: scipy>=1.0 in c:\programdata\anaconda3\lib\site-packages (f
rom seaborn) (1.7.1)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages
(from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages
(from matplotlib>=2.2->seaborn) (8.4.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\anaconda3\lib\site-p
ackages (from matplotlib>=2.2->seaborn) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\programdata\anaconda3\lib\site-packa
ges (from matplotlib>=2.2->seaborn) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-pack
ages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from cyc
ler>=0.10->matplotlib>=2.2->seaborn) (1.16.0)
Requirement already satisfied: pytz>=2017.3 in c:\programdata\anaconda3\lib\site-packages
(from pandas>=0.23->seaborn) (2021.3)
WARNING: Ignoring invalid distribution -oblib (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oblib (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oblib (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oblib (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oblib (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oblib (c:\programdata\anaconda3\lib\site-packages)
```

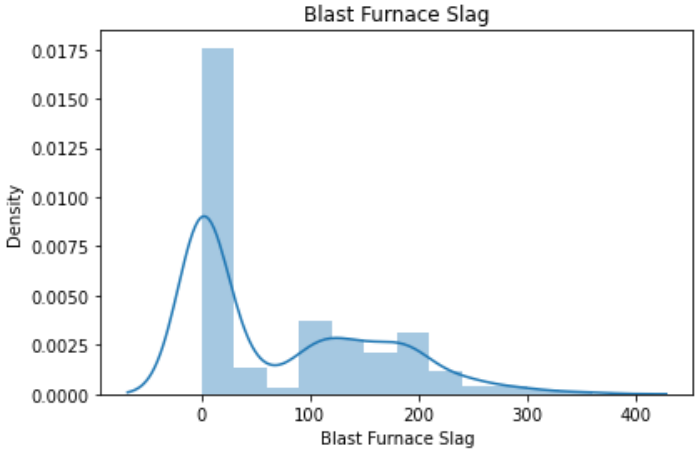
In [12]:
```
# Plotting the distplots without any transformation

for col in X_train.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    stats.probplot(X_train[col], dist="norm", plot=plt)
    plt.title(col)

    plt.show()
```

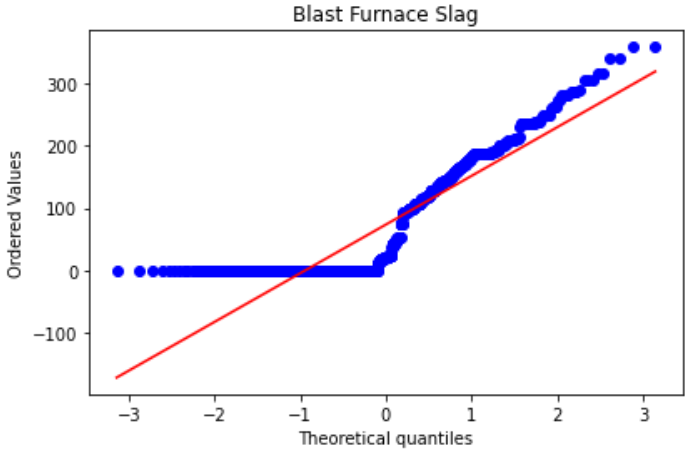
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



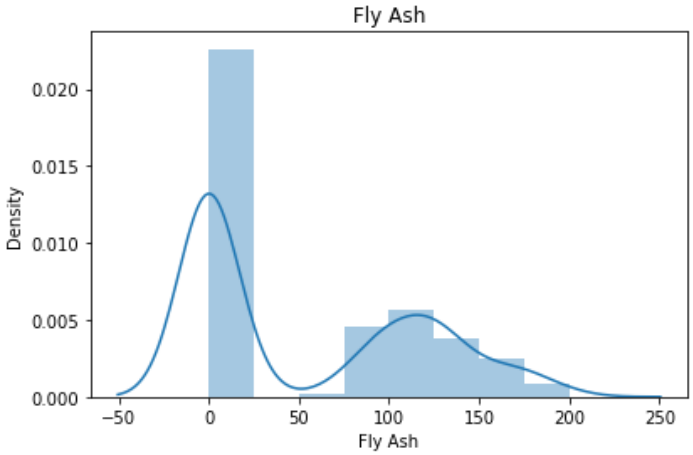


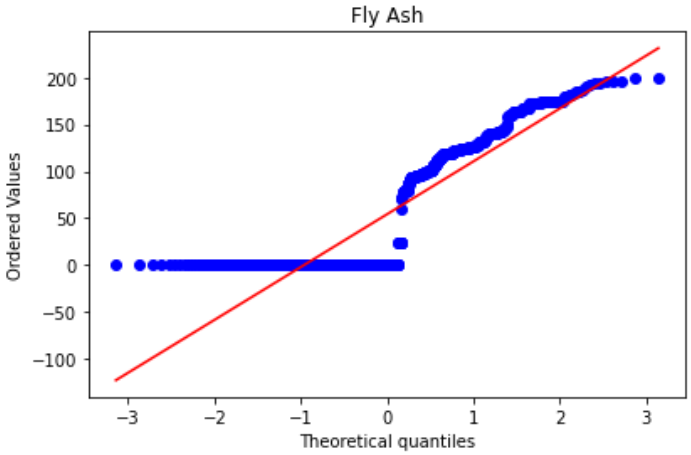

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



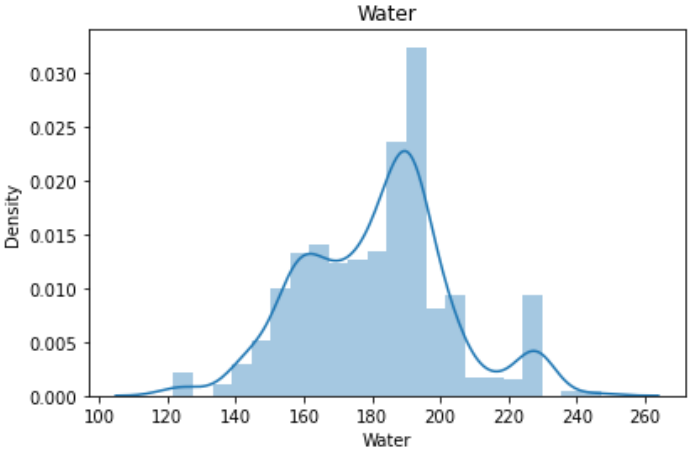


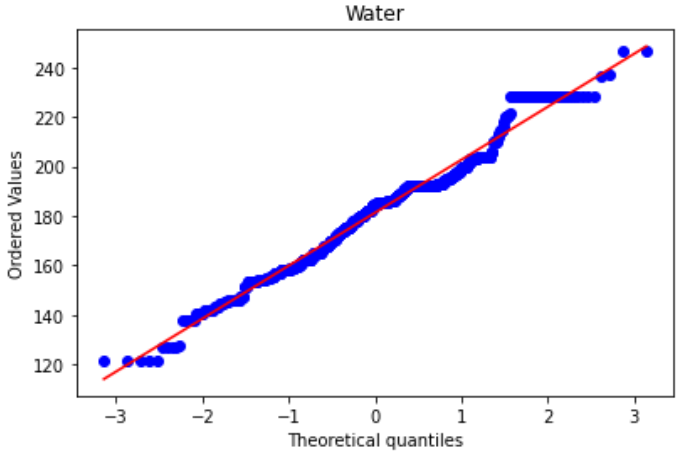

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



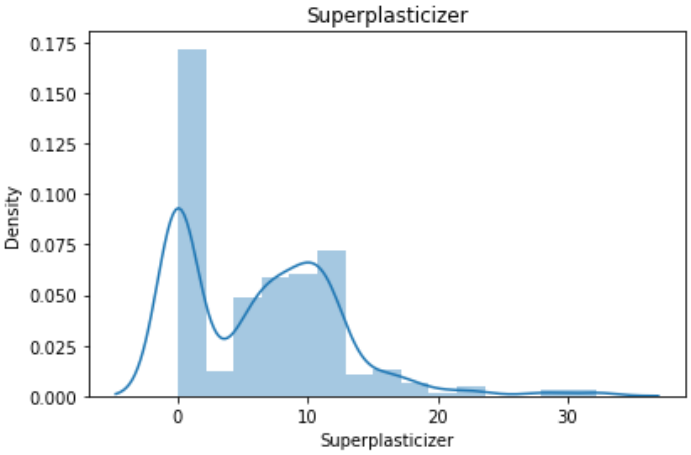


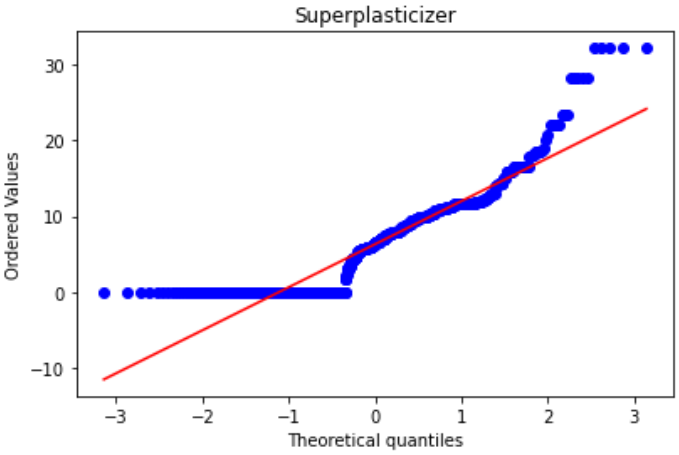

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



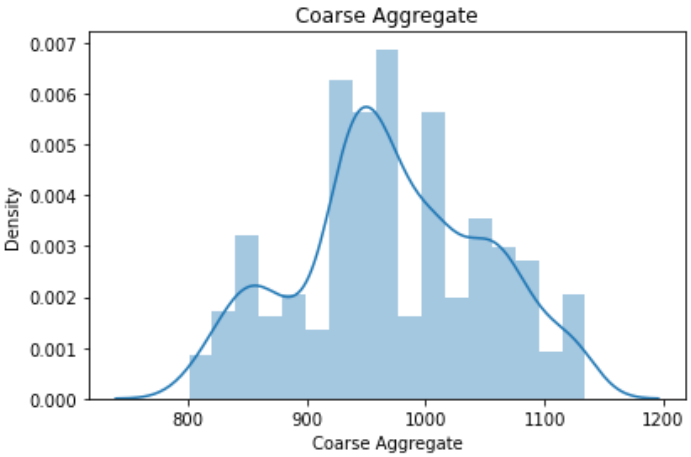


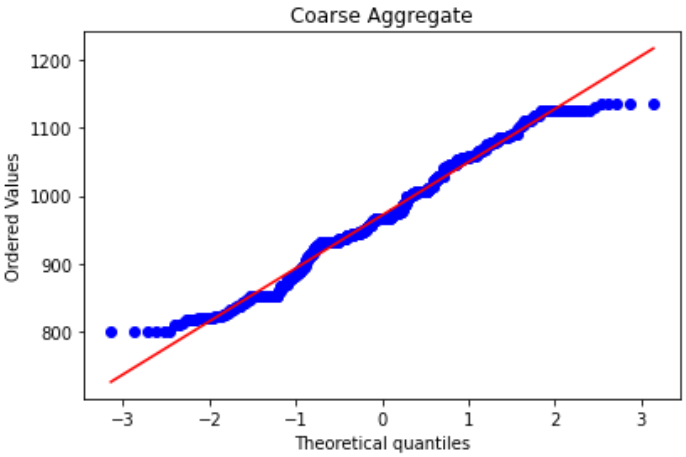

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



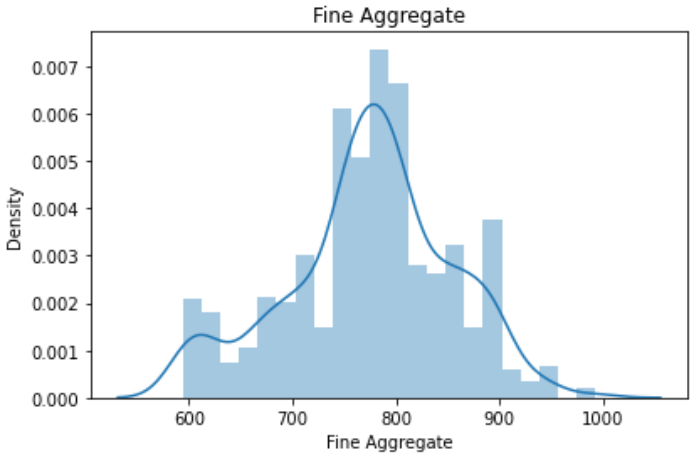


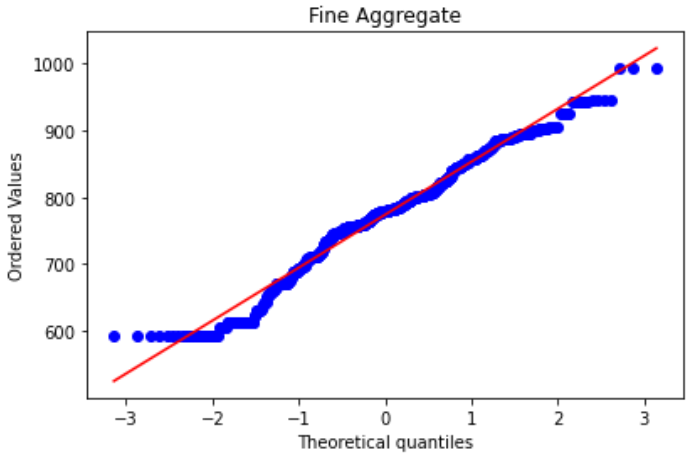

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



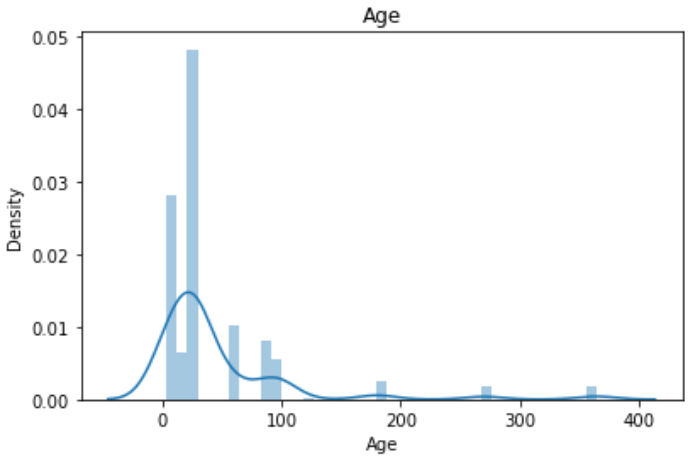


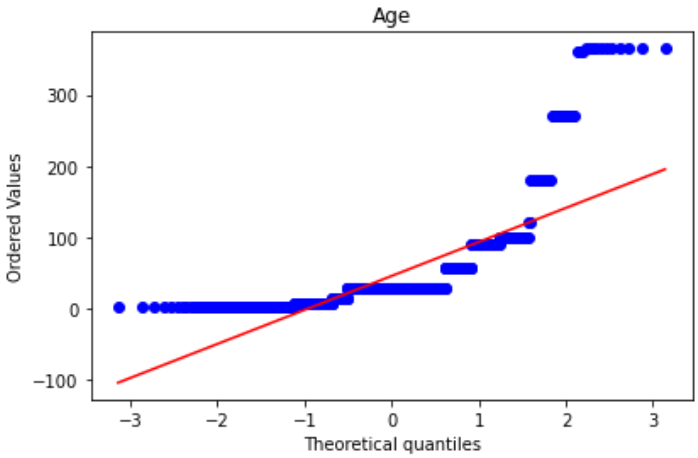

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```






```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





Applying Box-Cox Transform

In [13]:
```
pt = PowerTransformer(method='box-cox')
```

In [14]:
```
X_train_transformed = pt.fit_transform(X_train+0.000001)
X_test_transformed = pt.transform(X_test+0.000001)

pd.DataFrame({'cols':X_train.columns,'box_cox_lambdas':pt.lambdas_})
```

Out[14]:

	cols	box_cox_lambdas
0	Cement	0.192177
1	Blast Furnace Slag	0.023543
2	Fly Ash	-0.033365
3	Water	0.729294
4	Superplasticizer	0.102799
5	Coarse Aggregate	0.944492
6	Fine Aggregate	1.912493
7	Age	0.050675

In [15]:
```
# Applying linear regression on transformed data

lr = LinearRegression()
lr.fit(X_train_transformed,y_train)

y_pred2 = lr.predict(X_test_transformed)

r2_score(y_test,y_pred2)
```

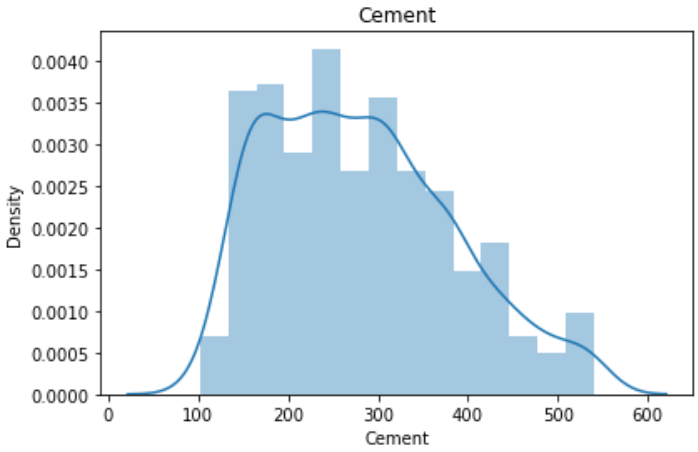
Out[15]:
```
0.8059395299868048
```

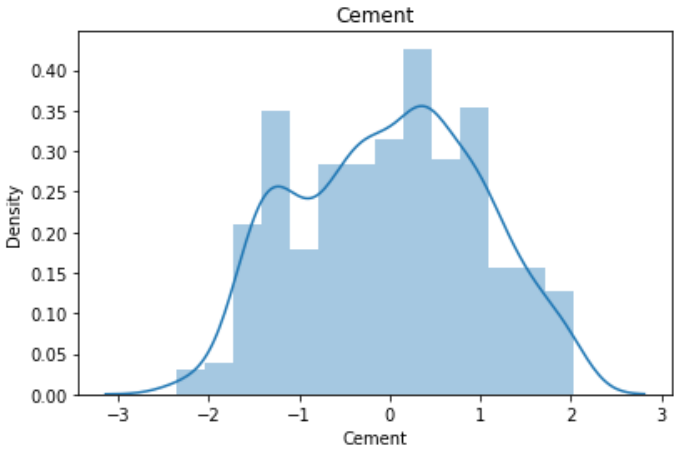
In [16]:
```
# Using cross val score

pt = PowerTransformer(method='box-cox')
X_transformed = pt.fit_transform(X+0.0000001)

lr = LinearRegression()
np.mean(cross_val_score(lr,X_transformed,y,scoring='r2'))
```

Out[16]:
```
0.6658537942219862
```

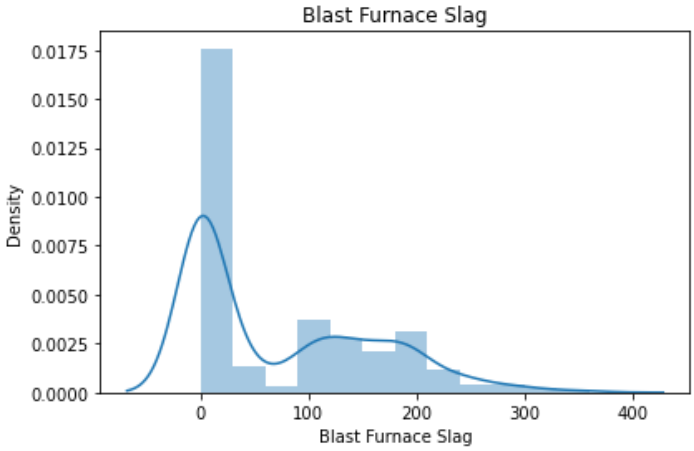
In [17]:
```
# Before and after comparision for Box-Cox Plot
X_train_transformed = pd.DataFrame(X_train_transformed,columns=X_train.columns)

for col in X_train_transformed.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    sns.distplot(X_train_transformed[col])
    plt.title(col)



    plt.show()
```

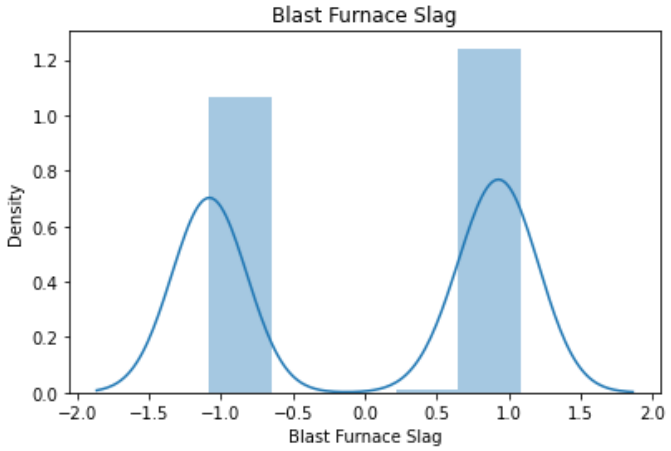
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



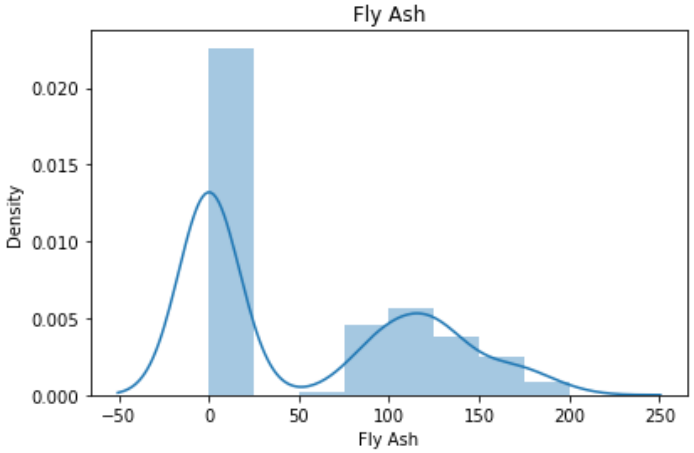


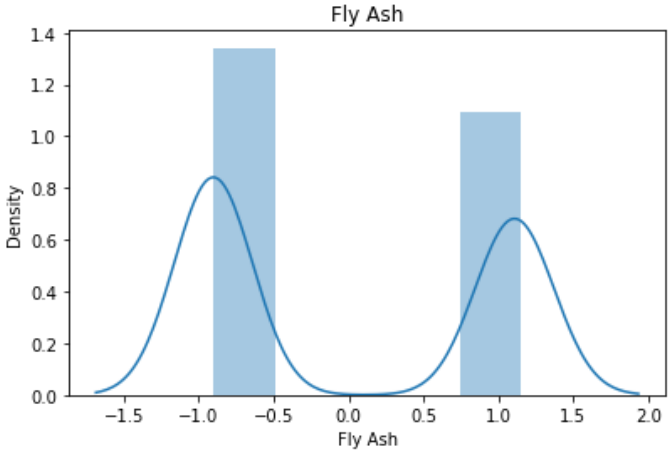

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



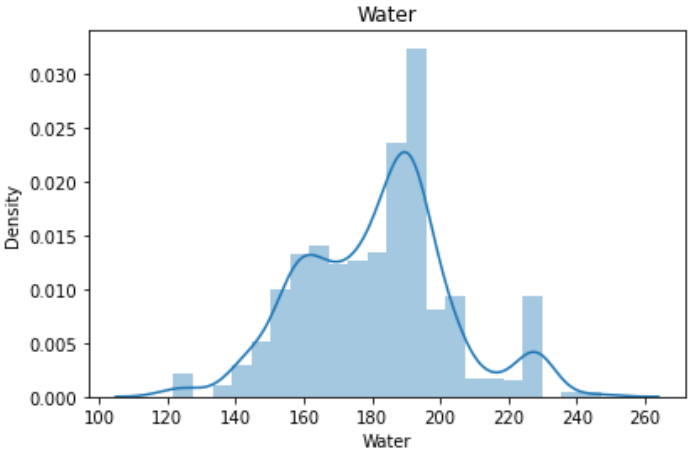


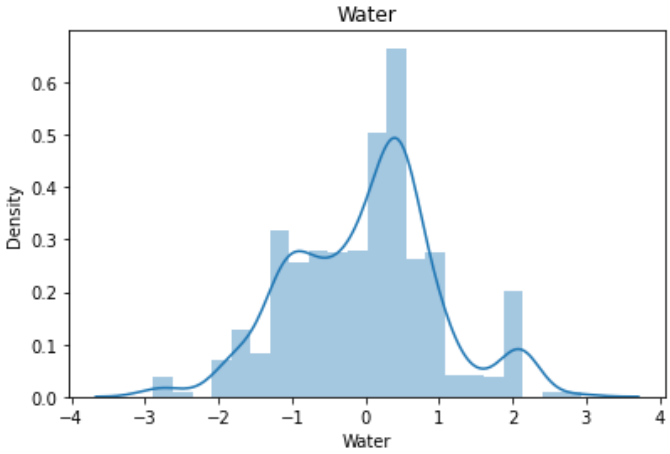

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



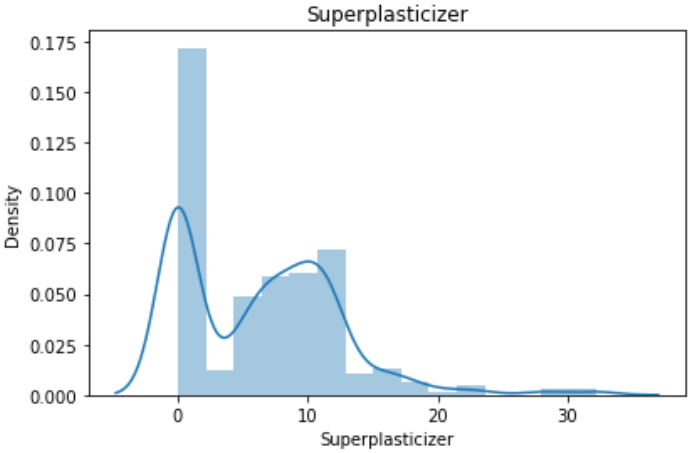


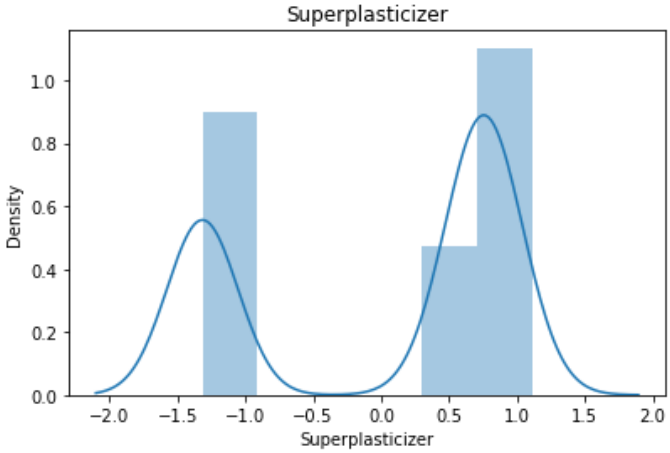
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



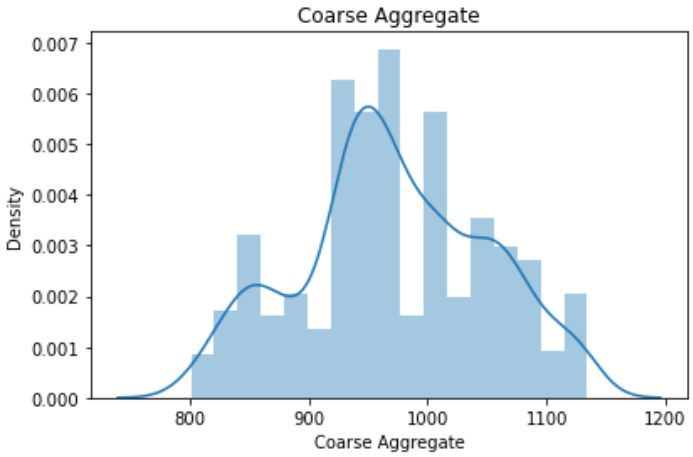


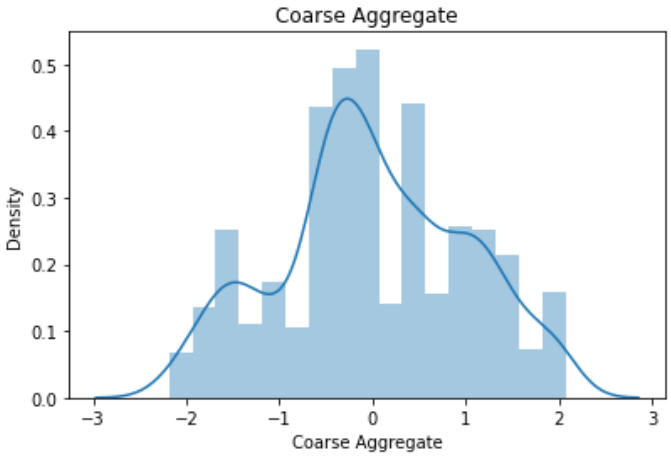

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



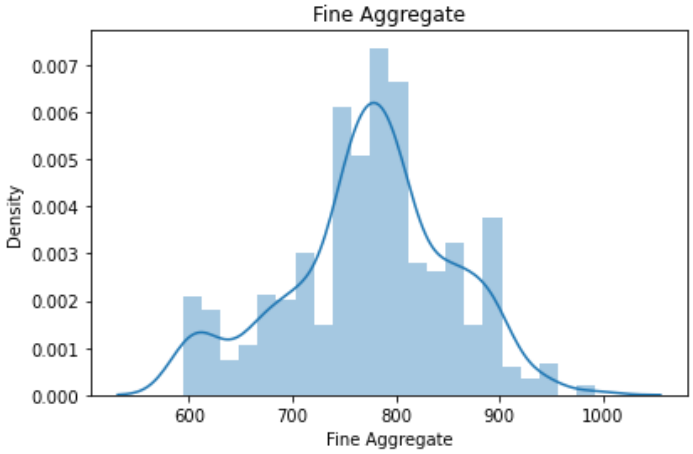


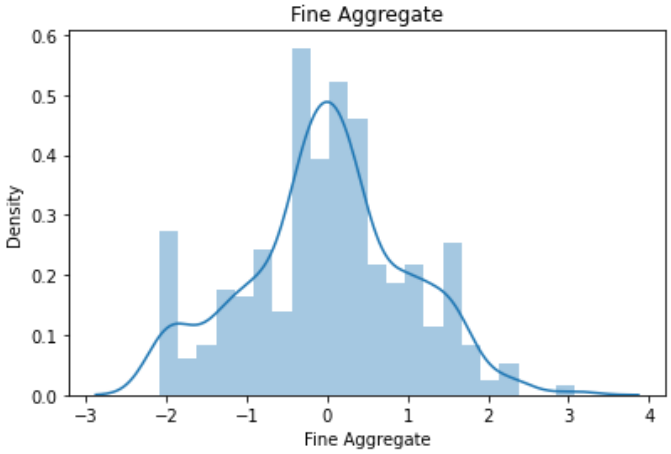

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



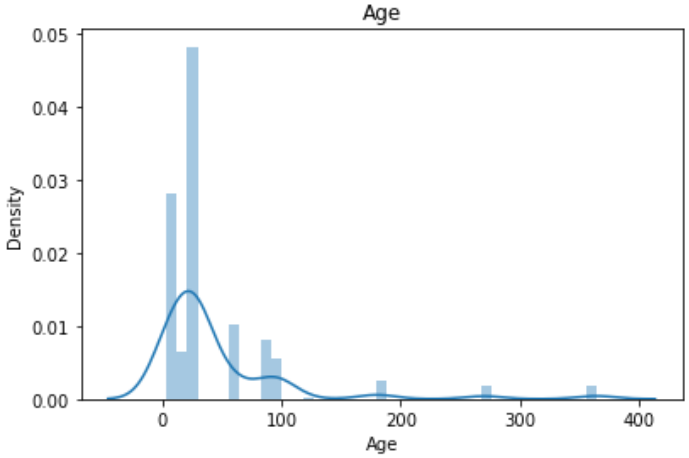


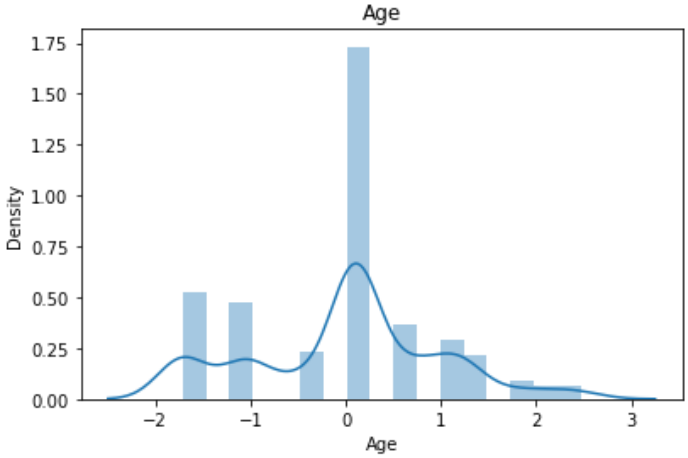

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```






```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





Apply Yeo-Johnson transform

In [18]:
```
pt1 = PowerTransformer()
```

In [19]:
```
X_train_transformed2 = pt1.fit_transform(X_train)
X_test_transformed2 = pt1.transform(X_test)

lr = LinearRegression()
lr.fit(X_train_transformed2,y_train)

y_pred3 = lr.predict(X_test_transformed2)

print(r2_score(y_test,y_pred3))

pd.DataFrame({'cols':X_train.columns,'Yeo_Johnson_lambdas':pt1.lambdas_})
```

```
0.8096460862674353
```

Out[19]:

	cols	Yeo_Johnson_lambdas
0	Cement	0.189513
1	Blast Furnace Slag	0.010273
2	Fly Ash	-0.140102
3	Water	0.727681
4	Superplasticizer	0.271741
5	Coarse Aggregate	0.944526
6	Fine Aggregate	1.913745
7	Age	0.005244

In [20]:
```
# applying cross val score

pt = PowerTransformer()
X_transformed2 = pt.fit_transform(X)

lr = LinearRegression()
np.mean(cross_val_score(lr,X_transformed2,y,scoring='r2'))
```

Out[20]:
```
0.6834625134285743
```

In [21]:
```
X_train_transformed2 = pd.DataFrame(X_train_transformed2,columns=X_train.columns)
```

In [22]:
```
X_train_transformed2
```

Out[22]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	-0.009023	0.904728	1.024625	-0.137365	0.776304	-0.786372	-0.407202	0.104565
1	-0.604728	-1.052106	1.132351	-1.035647	0.529274	1.450927	0.282969	1.248846
2	-0.603517	-1.052106	1.124763	-0.030435	0.247086	1.097752	0.021980	-1.696745
3	1.636626	-1.052106	-0.900126	2.111781	-1.233985	-0.504527	-2.085586	2.159262

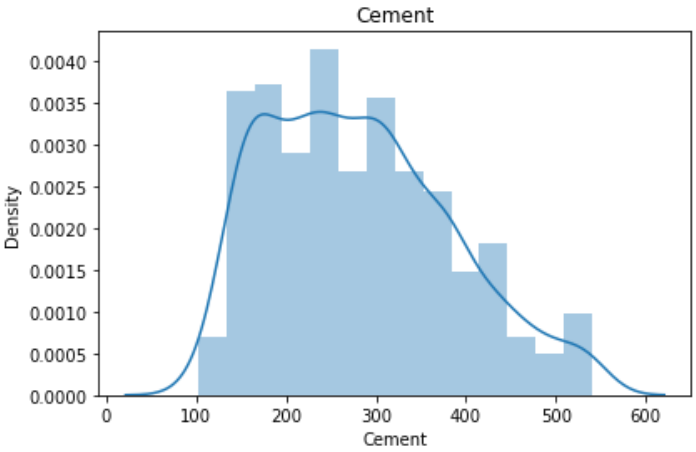
	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
4	0.938188	-1.052106	-0.900126	0.425694	0.388456	0.532615	-0.584531	0.104565
...	...	...	...	...	...	...	...	...
819	-1.697137	1.141106	1.060729	0.825766	0.291024	-1.595598	0.069329	0.104565
820	0.847477	1.125678	-0.900126	-0.753743	0.934141	-0.341994	-0.273968	-1.696745
821	1.142699	0.834321	-0.900126	-1.598327	0.915241	-0.315130	0.991109	0.723317
822	0.271409	-1.052106	-0.900126	0.186935	-1.233985	1.242641	-0.108751	0.104565
823	0.794462	-1.052106	1.168879	0.567189	0.876763	-2.183395	0.010493	0.104565

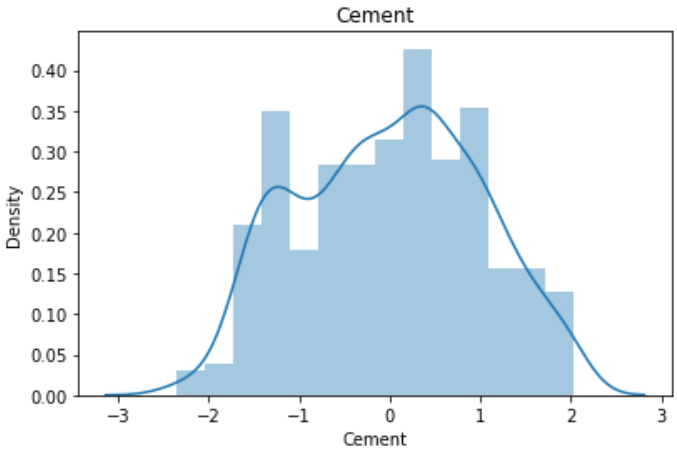
824 rows × 8 columns

In [23]:

```
# Before and after comparision for Yeo-Johnson

for col in X_train_transformed2.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    sns.distplot(X_train_transformed2[col])
    plt.title(col)

    plt.show()
```

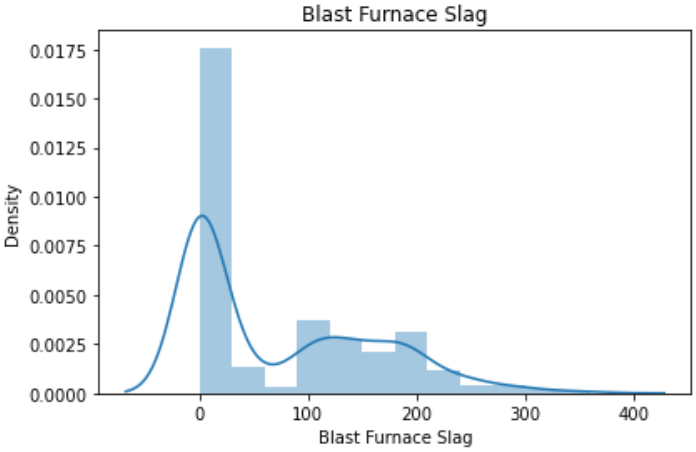
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

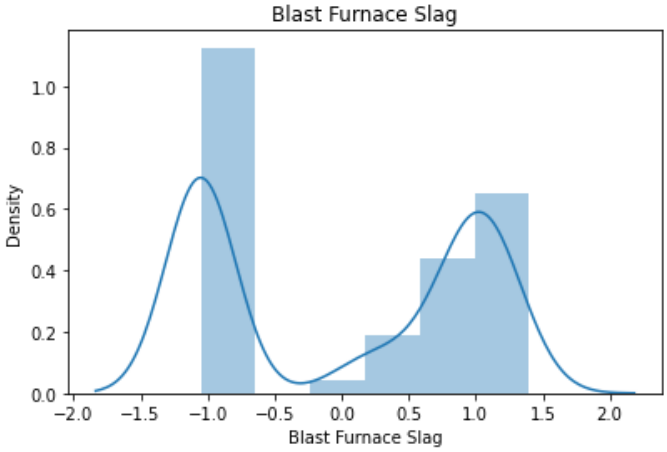




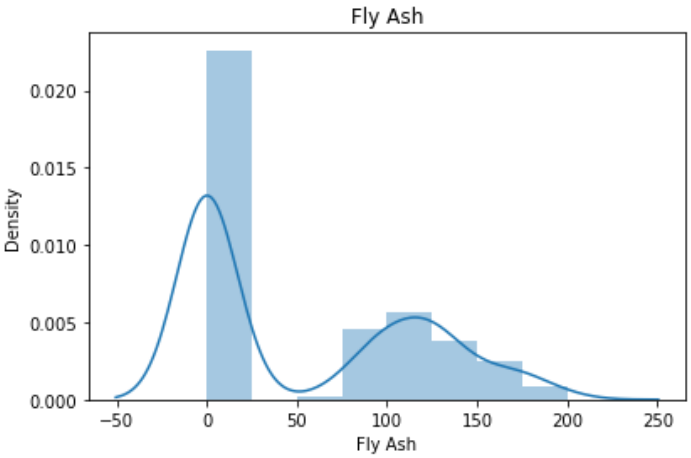

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
```

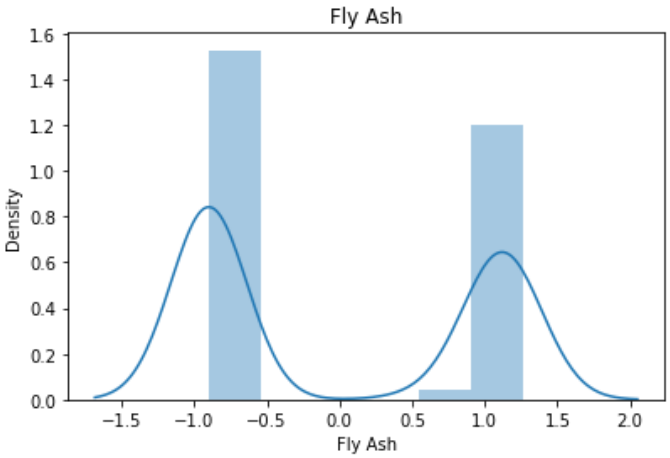
```
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



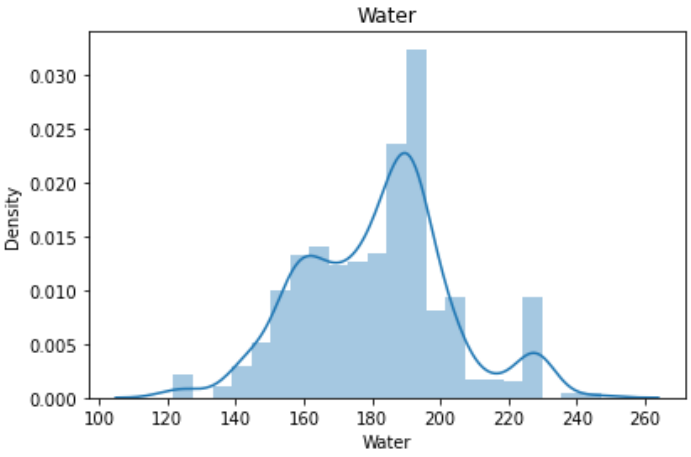


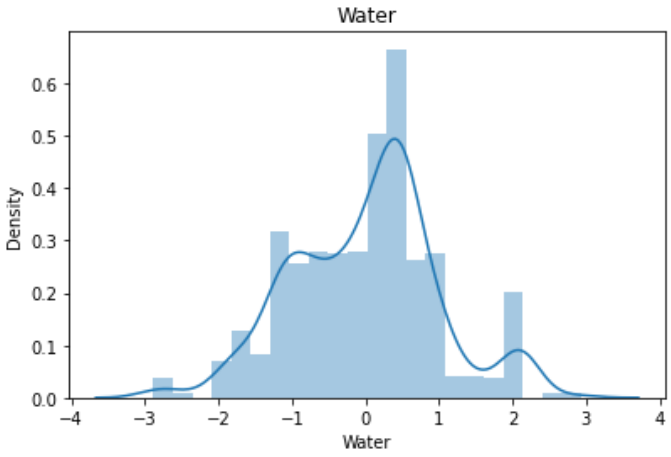

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



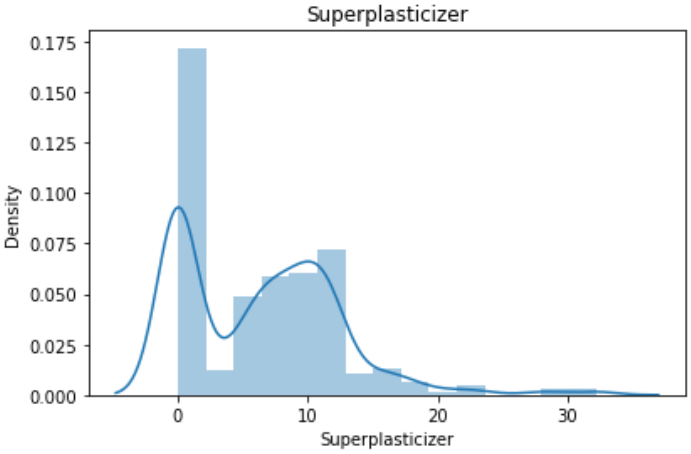


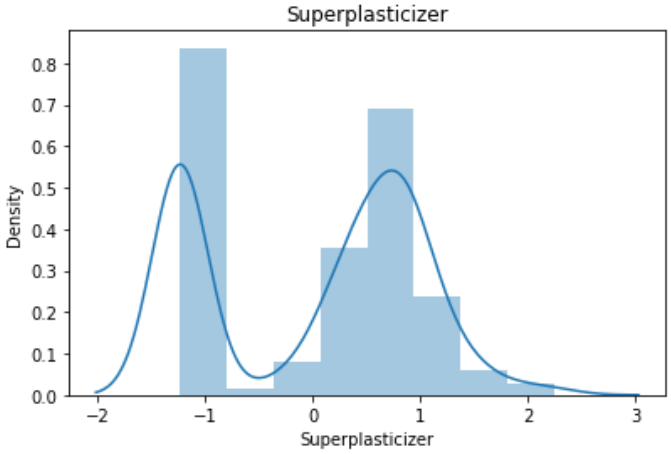

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



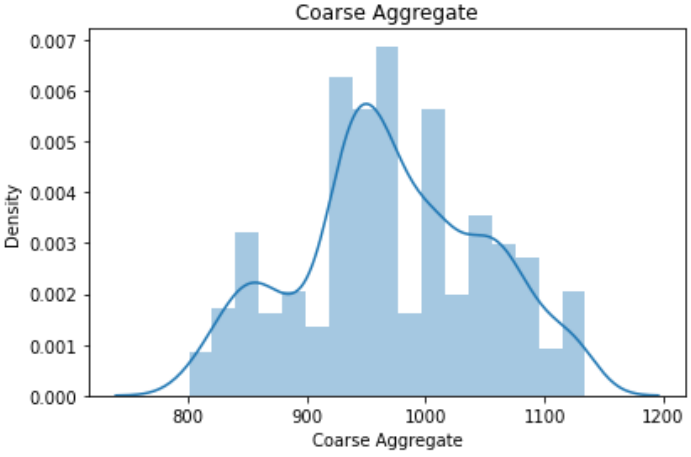


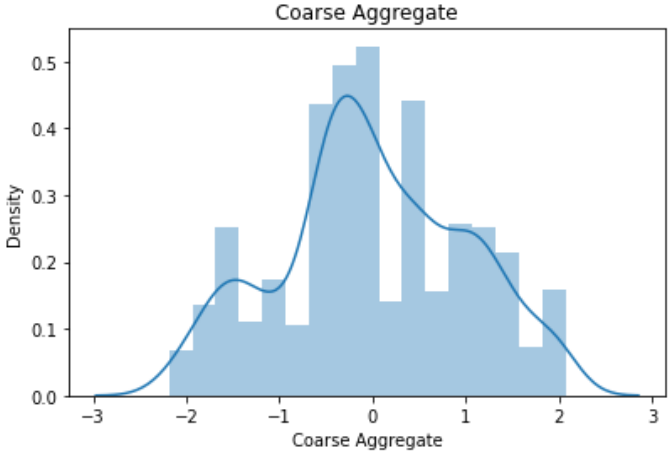

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



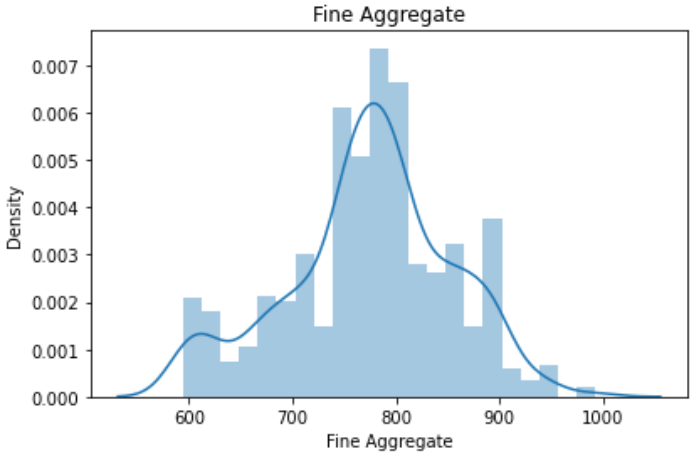


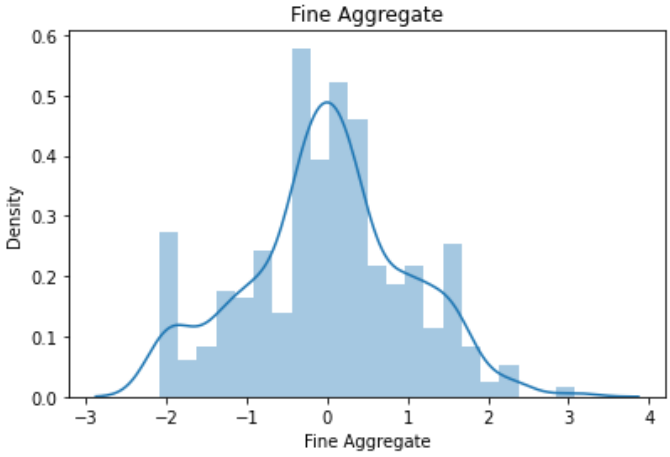

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



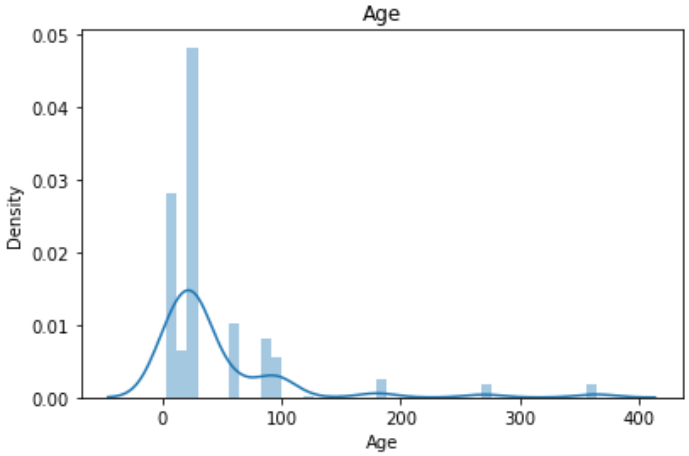


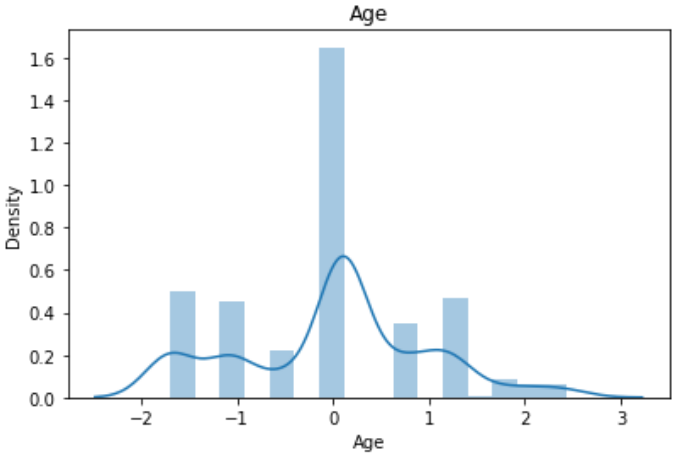
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

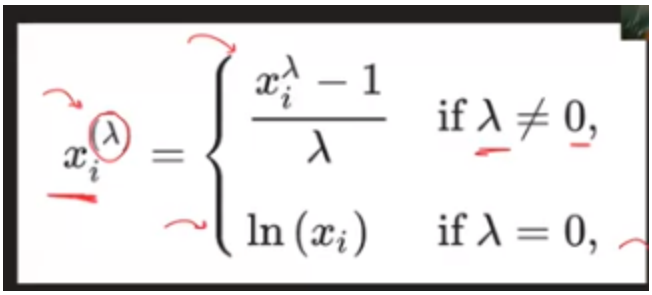





```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```







$$x_i^{(\lambda)} = \begin{cases} \dfrac{x_i^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(x_i) & \text{if } \lambda = 0, \end{cases}$$

In [24]:

```
# Side by side Lambdas
pd.DataFrame({'cols':X_train.columns,'box_cox_lambdas':pt.lambdas_,'Yeo_Johnson_lambdas':p
```

Out[24]:

	cols	box_cox_lambdas	Yeo_Johnson_lambdas
0	Cement	0.169544	0.189513
1	Blast Furnace Slag	0.016633	0.010273
2	Fly Ash	-0.136480	-0.140102
3	Water	0.808438	0.727681
4	Superplasticizer	0.264160	0.271741
5	Coarse Aggregate	1.129395	0.944526
6	Fine Aggregate	1.830763	1.913745
7	Age	0.001771	0.005244

In [ ]:

Feature Engineering 101

Topic - 4

Function Transformer

Sample Code

```python
from sklearn.base import BaseEstimator, TransformerMixin

class MultiplyTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, factor=1.0):
        self.factor = factor

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X * self.factor
```

Function Transformer

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

import scipy.stats as stats

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import ColumnTransformer
```

In [2]:
```python
df = pd.read_csv('train.csv',usecols=['Age','Fare','Survived'])
```

In [3]:
```python
df
```

Out[3]:

	Survived	Age	Fare
0	0	22.0	7.2500
1	1	38.0	71.2833
2	1	26.0	7.9250
3	1	35.0	53.1000
4	0	35.0	8.0500
...	...	...	...
886	0	27.0	13.0000
887	1	19.0	30.0000

	Survived	Age	Fare
888	0	NaN	23.4500
889	1	26.0	30.0000
890	0	32.0	7.7500

891 rows × 3 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Survived      0
Age         177
Fare          0
dtype: int64
```

In [5]:
```
df['Age'].fillna(df['Age'].mean(),inplace=True)
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Survived    0
Age         0
Fare        0
dtype: int64
```

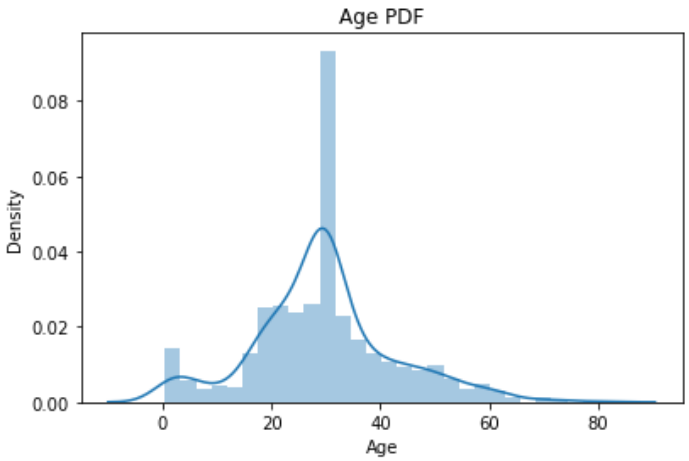
In [7]:
```
X = df.iloc[:,1:3]
y = df.iloc[:,0]
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

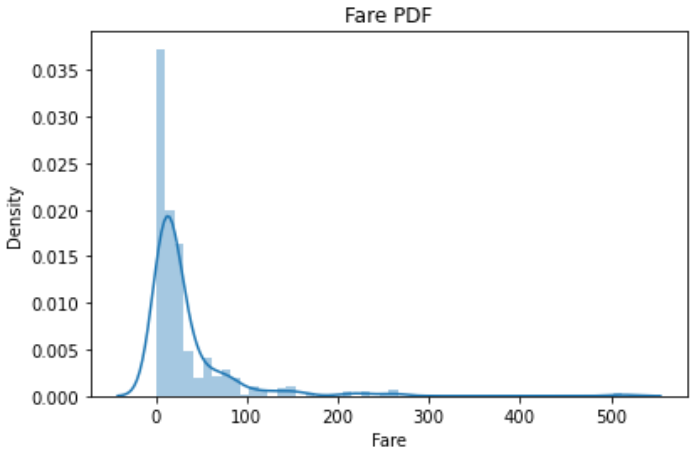
In [9]:
```
#Age>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
plt.figure(figsize=(14,4))
plt.subplot(121)
sns.distplot(X_train['Age'])
plt.title('Age PDF')

plt.subplot(122)
stats.probplot(X_train['Age'], dist="norm", plot=plt)
plt.title('Age QQ Plot')

plt.show()

#Fare >>>>>>>>>>>>>>>>>>>>>>>>>>>>
plt.figure(figsize=(14,4))
plt.subplot(121)
sns.distplot(X_train['Fare'])
plt.title('Fare PDF')

plt.subplot(122)
stats.probplot(X_train['Fare'], dist="norm", plot=plt)
plt.title('Fare QQ Plot')

plt.show()
```

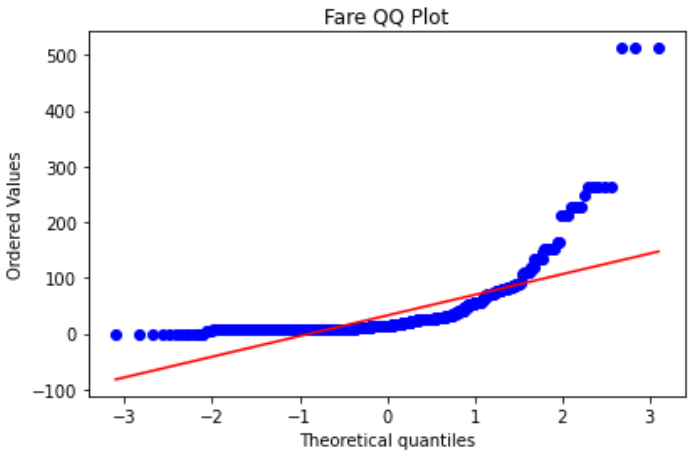
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
```

```
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





In [10]:
```
#Now the classifire time

clf = LogisticRegression()
clf2 = DecisionTreeClassifier()
```

In [11]:
```
#Predict nad Fit

clf.fit(X_train,y_train)
clf2.fit(X_train,y_train)

y_pred = clf.predict(X_test)
y_pred1 = clf2.predict(X_test)
```

In [12]:
```
#Acc. Score

print("Accuracy LR",accuracy_score(y_test,y_pred))
print("Accuracy DT",accuracy_score(y_test,y_pred1))
```

```
Accuracy LR 0.6480446927374302
Accuracy DT 0.664804469273743
```

Now use Log1 TransforMer

In [13]:
```
trf = FunctionTransformer(func=np.log1p)
```

In [14]:
```
X_train_transformed = trf.fit_transform(X_train)
X_test_transformed = trf.transform(X_test)
```

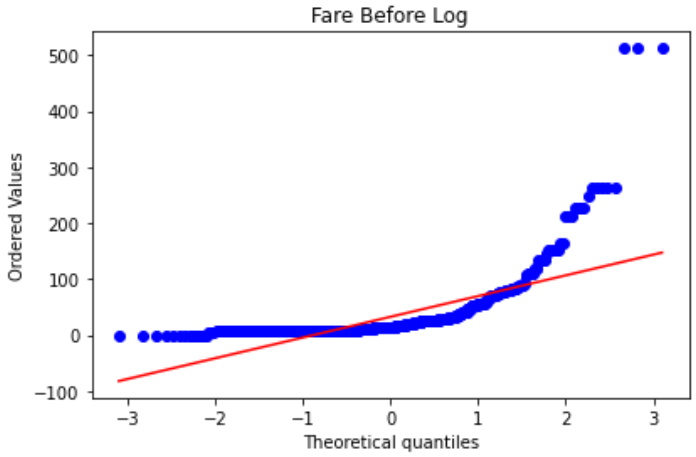
In [15]:
```
clf = LogisticRegression()
clf2 = DecisionTreeClassifier()

clf.fit(X_train_transformed,y_train)
clf2.fit(X_train_transformed,y_train)

y_pred = clf.predict(X_test_transformed)
y_pred1 = clf2.predict(X_test_transformed)

print("Accuracy LR",accuracy_score(y_test,y_pred))
print("Accuracy DT",accuracy_score(y_test,y_pred1))
```

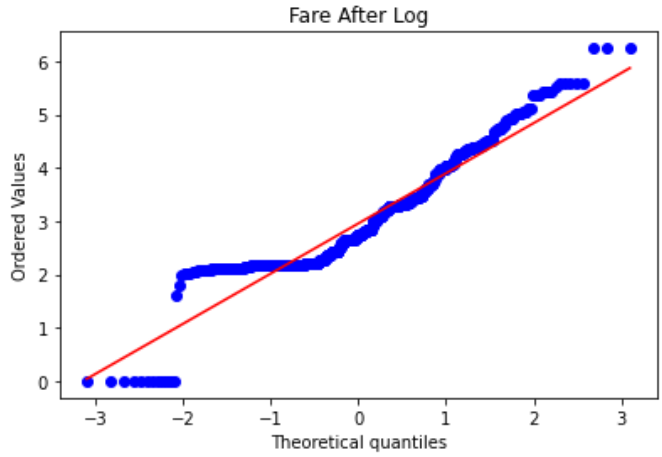
```
Accuracy LR 0.6815642458100558
Accuracy DT 0.6815642458100558
```

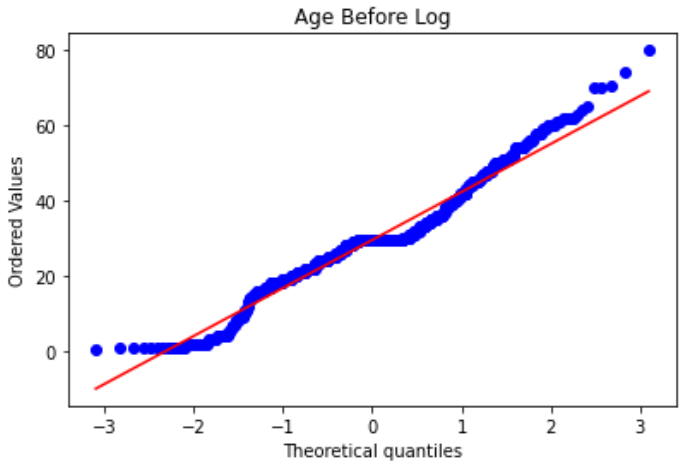
In [16]:
```
#With cross val score

X_transformed = trf.fit_transform(X)

clf = LogisticRegression()
clf2 = DecisionTreeClassifier()

print("LR",np.mean(cross_val_score(clf,X_transformed,y,scoring='accuracy',cv=10)))
print("DT",np.mean(cross_val_score(clf2,X_transformed,y,scoring='accuracy',cv=10)))
```

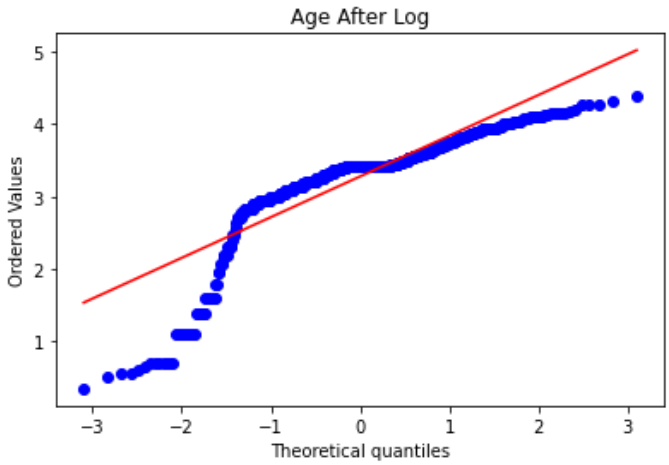
```
LR 0.678027465667915
DT 0.661123595505618
```

In [17]:
```
#Fare >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
plt.figure(figsize=(14,4))

plt.subplot(121)
stats.probplot(X_train['Fare'], dist="norm", plot=plt)
plt.title('Fare Before Log')

plt.subplot(122)
stats.probplot(X_train_transformed['Fare'], dist="norm", plot=plt)
plt.title('Fare After Log')

plt.show()

#Age >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
plt.figure(figsize=(14,4))

plt.subplot(121)
stats.probplot(X_train['Age'], dist="norm", plot=plt)
plt.title('Age Before Log')

plt.subplot(122)
stats.probplot(X_train_transformed['Age'], dist="norm", plot=plt)
plt.title('Age After Log')

plt.show()
```









Now use Log TransforMer

In [18]:
```
trf2 = ColumnTransformer([('log',FunctionTransformer(np.log1p),['Fare'])],remainder='passt

X_train_transformed2 = trf2.fit_transform(X_train)
X_test_transformed2 = trf2.transform(X_test)
```

In [19]:
```
clf = LogisticRegression()
clf2 = DecisionTreeClassifier()

clf.fit(X_train_transformed2,y_train)
clf2.fit(X_train_transformed2,y_train)

y_pred = clf.predict(X_test_transformed2)
y_pred2 = clf2.predict(X_test_transformed2)

print("Accuracy LR",accuracy_score(y_test,y_pred))
print("Accuracy DT",accuracy_score(y_test,y_pred2))
```

```
Accuracy LR 0.6703910614525139
Accuracy DT 0.6871508379888268
```

In [20]:
```
X_transformed2 = trf2.fit_transform(X)

clf = LogisticRegression()
clf2 = DecisionTreeClassifier()

print("LR",np.mean(cross_val_score(clf,X_transformed2,y,scoring='accuracy',cv=10)))
print("DT",np.mean(cross_val_score(clf2,X_transformed2,y,scoring='accuracy',cv=10)))
```

```
LR 0.6712609238451936
DT 0.6610736579275904
```

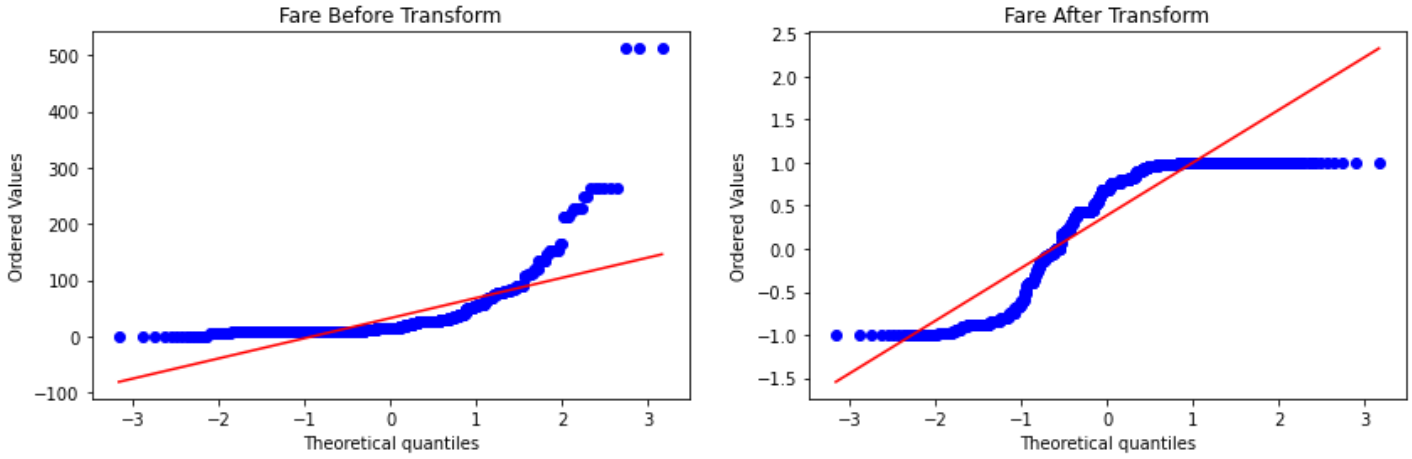
In [21]:

```
def apply_transform(transform):
    X = df.iloc[:,1:3]
    y = df.iloc[:,0]

    trf = ColumnTransformer([('log',FunctionTransformer(transform),['Fare'])],remainder='p

    X_trans = trf.fit_transform(X)

    clf = LogisticRegression()

    print("Accuracy",np.mean(cross_val_score(clf,X_trans,y,scoring='accuracy',cv=10)))

    plt.figure(figsize=(14,4))

    plt.subplot(121)
    stats.probplot(X['Fare'], dist="norm", plot=plt)
    plt.title('Fare Before Transform')

    plt.subplot(122)
    stats.probplot(X_trans[:,0], dist="norm", plot=plt)
    plt.title('Fare After Transform')

    plt.show()
```

In [22]:

```
apply_transform(np.sin)
```

```
Accuracy 0.6195131086142323
```



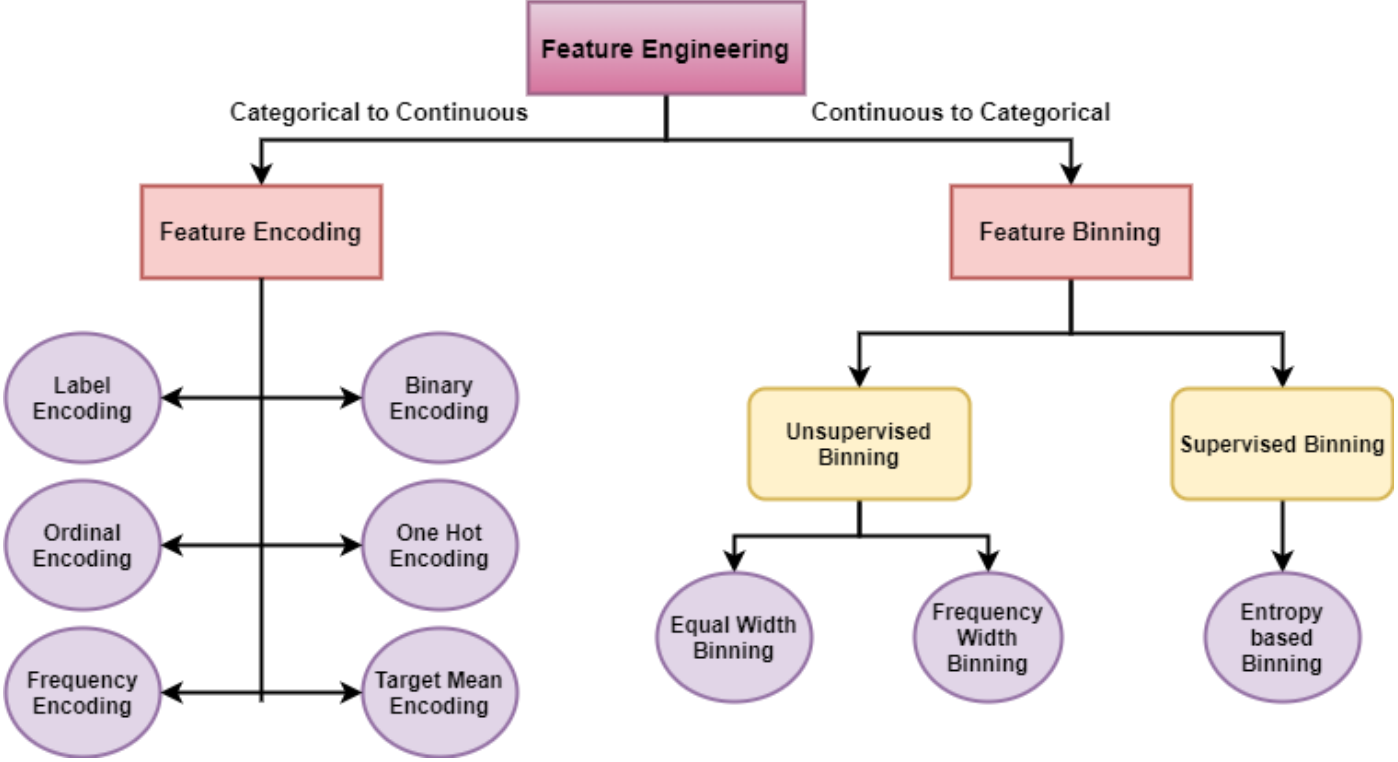
In [ ]:

Feature Engineering 101

Topic - 5

Encoding's

1. Ordinal
2. One-Hot
3. Binary
4. Target
5. Helmert
6. Leave One Out



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

df = pd.read_csv('cars.csv')
```

In [2]:
```
df
```

Out[2]:

	brand	km_driven	fuel	owner	selling_price
0	Maruti	145500	Diesel	First Owner	450000
1	Skoda	120000	Diesel	Second Owner	370000
2	Honda	140000	Petrol	Third Owner	158000
3	Hyundai	127000	Diesel	First Owner	225000
4	Maruti	120000	Petrol	First Owner	130000
...	...	...	...	...	...
8123	Hyundai	110000	Petrol	First Owner	320000
8124	Hyundai	119000	Diesel	Fourth & Above Owner	135000
8125	Maruti	120000	Diesel	First Owner	382000
8126	Tata	25000	Diesel	First Owner	290000
8127	Tata	25000	Diesel	First Owner	290000

8128 rows × 5 columns

In [3]:
```
print(df.isnull().sum())
```

```
brand          0
km_driven      0
fuel           0
owner          0
```

```
selling_price    0
dtype: int64
```

In [4]:
```
print(df['owner'].value_counts())
print(df['fuel'].value_counts())
```

```
First Owner             5289
Second Owner            2105
Third Owner              555
Fourth & Above Owner     174
Test Drive Car             5
Name: owner, dtype: int64
Diesel    4402
Petrol    3631
CNG         57
LPG         38
Name: fuel, dtype: int64
```

In [5]:
```
print(df['brand'].value_counts())
```

```
Maruti           2448
Hyundai          1415
Mahindra          772
Tata              734
Toyota            488
Honda             467
Ford              397
Chevrolet         230
Renault           228
Volkswagen        186
BMW               120
Skoda             105
Nissan             81
Jaguar             71
Volvo              67
Datsun             65
Mercedes-Benz      54
Fiat               47
Audi               40
Lexus              34
Jeep               31
Mitsubishi         14
Force               6
Land                6
Isuzu               5
Kia                 4
Ambassador          4
Daewoo              3
MG                  3
Ashok               1
Opel                1
Peugeot             1
Name: brand, dtype: int64
```

Now apply OneHot encoding with pandas

In [6]:
```
pd.get_dummies(df,columns=['fuel','owner'])
```

Out[6]:

	brand	km_driven	selling_price	fuel_CNG	fuel_Diesel	fuel_LPG	fuel_Petrol	owner_First Owner	owner_Fourth & Above Owner	ow

	brand	km_driven	selling_price	fuel_CNG	fuel_Diesel	fuel_LPG	fuel_Petrol	owner_First Owner	owner_Fourth & Above Owner	ow
0	Maruti	145500	450000	0	1	0	0	1	0	
1	Skoda	120000	370000	0	1	0	0	0	0	
2	Honda	140000	158000	0	0	0	1	0	0	
3	Hyundai	127000	225000	0	1	0	0	1	0	
4	Maruti	120000	130000	0	0	0	1	1	0	
...	...	...	...	...	...	...	...	...	...	
8123	Hyundai	110000	320000	0	0	0	1	1	0	
8124	Hyundai	119000	135000	0	1	0	0	0	1	
8125	Maruti	120000	382000	0	1	0	0	1	0	
8126	Tata	25000	290000	0	1	0	0	1	0	
8127	Tata	25000	290000	0	1	0	0	1	0	

8128 rows × 12 columns

K-1 OneHot Encoding

In [7]:
```
#K-1 means we have drop the two column for this table. for reasone this step is ""Multico

pd.get_dummies(df,columns=['fuel','owner'],drop_first=True)
```

Out[7]:

	brand	km_driven	selling_price	fuel_Diesel	fuel_LPG	fuel_Petrol	owner_Fourth & Above Owner	owner_Second Owner	owner_Test Drive Car
0	Maruti	145500	450000	1	0	0	0	0	0
1	Skoda	120000	370000	1	0	0	0	1	0
2	Honda	140000	158000	0	0	1	0	0	0
3	Hyundai	127000	225000	1	0	0	0	0	0
4	Maruti	120000	130000	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
8123	Hyundai	110000	320000	0	0	1	0	0	0
8124	Hyundai	119000	135000	1	0	0	1	0	0
8125	Maruti	120000	382000	1	0	0	0	0	0
8126	Tata	25000	290000	1	0	0	0	0	0
8127	Tata	25000	290000	1	0	0	0	0	0

8128 rows × 10 columns

OneHotEncoding using Sklearn

In [8]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(df.iloc[:,0:4],df.iloc[:,-1],test_size=0.
```

In [9]:
```
X_train.head()
```

Out[9]:

	brand	km_driven	fuel	owner
6310	Maruti	131111	Diesel	First Owner
2219	Maruti	29000	Petrol	First Owner
4600	Maruti	85000	Diesel	Second Owner
2167	Maruti	90000	Petrol	Second Owner
5272	Hyundai	80000	Diesel	Second Owner

In [10]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [11]:
```
ohe = OneHotEncoder(drop='first',sparse=False,dtype=np.int32)
```

In [12]:
```
X_train_new = ohe.fit_transform(X_train[['fuel','owner']])
X_test_new = ohe.fit_transform(X_test[['fuel','owner']])
```

In [13]:
```
print(X_train_new.shape)
print(X_test_new.shape)
```

```
(5689, 7)
(2439, 7)
```

Now merge X_train_new and X_train table

In [14]:
```
np.hstack((X_train[['brand','km_driven']].values,X_train_new))
```

Out[14]:
```
array([['Maruti', 131111, 1, ..., 0, 0, 0],
       ['Maruti', 29000, 0, ..., 0, 0, 0],
       ['Maruti', 85000, 1, ..., 1, 0, 0],
       ...,
       ['Tata', 15000, 0, ..., 0, 0, 0],
       ['Maruti', 32500, 1, ..., 1, 0, 0],
       ['Isuzu', 121000, 1, ..., 0, 0, 0]], dtype=object)
```

Now apply OneHot Encoding catagories cloumn(Brand)

In [15]:
```
counts = df['brand'].value_counts()
```

In [16]:
```
df['brand'].nunique()
threshold = 100
```

In [17]:
```
repl = counts[counts <= threshold].index
```

In [18]:
```
pd.get_dummies(df['brand'].replace(repl, 'uncommon')).sample(50)
```

Out[18]:

	BMW	Chevrolet	Ford	Honda	Hyundai	Mahindra	Maruti	Renault	Skoda	Tata	Toyota	Volkswagen	un
4309	0	0	0	0	0	0	1	0	0	0	0	0	
7010	0	0	0	0	0	0	1	0	0	0	0	0	
10	0	0	0	0	0	0	0	1	0	0	0	0	
5309	0	0	0	0	0	0	0	1	0	0	0	0	
5737	0	0	0	0	0	1	0	0	0	0	0	0	
1502	0	0	0	0	0	0	1	0	0	0	0	0	
7096	0	0	1	0	0	0	0	0	0	0	0	0	
6758	0	0	0	0	0	0	0	1	0	0	0	0	
950	0	0	0	0	1	0	0	0	0	0	0	0	
5624	0	0	0	0	0	0	1	0	0	0	0	0	
736	0	0	0	0	0	0	0	0	0	0	1	0	
4334	1	0	0	0	0	0	0	0	0	0	0	0	
1417	0	0	0	0	1	0	0	0	0	0	0	0	
3088	0	0	0	0	0	0	1	0	0	0	0	0	
172	0	0	1	0	0	0	0	0	0	0	0	0	
6716	0	0	0	0	1	0	0	0	0	0	0	0	
5323	0	0	0	0	0	0	0	0	0	0	1	0	
87	0	0	0	0	0	0	1	0	0	0	0	0	
3132	0	0	0	0	0	1	0	0	0	0	0	0	
6994	0	0	0	0	0	0	1	0	0	0	0	0	
3704	0	0	0	0	1	0	0	0	0	0	0	0	
4783	0	0	0	0	0	0	0	0	0	1	0	0	
1136	0	0	0	0	0	0	0	0	0	0	0	0	
7272	0	0	0	0	0	0	0	0	0	1	0	0	
265	0	0	0	0	0	0	0	0	0	0	1	0	
3477	0	0	1	0	0	0	0	0	0	0	0	0	
2820	0	0	0	0	0	0	1	0	0	0	0	0	
1495	0	0	0	0	0	1	0	0	0	0	0	0	
2598	0	0	0	0	1	0	0	0	0	0	0	0	
1691	0	0	0	0	0	0	0	0	0	0	0	0	
3421	0	0	0	0	1	0	0	0	0	0	0	0	
7319	0	0	0	0	0	1	0	0	0	0	0	0	
2832	0	0	0	0	0	0	0	0	0	0	0	0	
1581	0	0	0	0	1	0	0	0	0	0	0	0	

	BMW	Chevrolet	Ford	Honda	Hyundai	Mahindra	Maruti	Renault	Skoda	Tata	Toyota	Volkswagen	un
6799	0	0	0	0	1	0	0	0	0	0	0	0	
1158	0	0	0	0	0	0	0	1	0	0	0	0	
183	0	0	0	0	1	0	0	0	0	0	0	0	
2067	0	0	0	0	0	0	0	0	0	0	0	1	
7194	0	0	0	0	1	0	0	0	0	0	0	0	
5327	0	0	0	0	1	0	0	0	0	0	0	0	
5192	0	0	0	0	1	0	0	0	0	0	0	0	
5608	0	0	0	0	0	0	0	0	0	0	1	0	
1036	0	0	0	0	0	0	1	0	0	0	0	0	
7077	0	0	0	0	0	0	0	0	0	0	0	0	
6291	0	0	0	0	0	0	1	0	0	0	0	0	
7348	0	0	0	0	1	0	0	0	0	0	0	0	
7267	0	0	0	0	0	1	0	0	0	0	0	0	
3383	0	0	0	0	0	0	1	0	0	0	0	0	
7216	0	0	0	0	1	0	0	0	0	0	0	0	
5415	0	0	0	0	0	0	0	0	0	1	0	0	

What is Multi-Collinearity?

Multicollinearity is a phenomenon that occurs when two or more independent variables in a multiple regression model are highly correlated with each other. This means that they are measuring the same or similar information, and this can lead to unstable and unreliable coefficient estimates.

When multicollinearity is present, the coefficients of the independent variables may change erratically in response to small changes in the model or data. This makes it difficult to interpret the importance of each independent variable and can lead to incorrect

conclusions about the relationship between the variables.

Multicollinearity can be detected by calculating the correlation matrix of the independent variables and looking for high correlation coefficients. Variance Inflation Factor (VIF) can also be used to detect multicollinearity. A VIF value greater than 5 is an indication of high multicollinearity.

To address multicollinearity, one can remove one of the highly correlated independent variables from the model, or combine them into a single variable. Another approach is to use principal component analysis to extract a new set of uncorrelated variables from the original correlated set.

In [ ]:

Feature Engineering 101

Topic - 6

Pipeline

PipiLine in Machine Learning

Without using PipeLine

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

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
df = pd.read_csv('train.csv')
```

In [3]:
```
df.sample(5)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
559	560	1	3	de Messemaeker, Mrs. Guillaume Joseph (Emma)	female	36.0	1	0	345572	17.40	NaN	S
419	420	0	3	Van Impe, Miss. Catharina	female	10.0	0	2	345773	24.15	NaN	S
153	154	0	3	van Billiard, Mr. Austin Blyler	male	40.5	0	2	A/5. 851	14.50	NaN	S
741	742	0	1	Cavendish, Mr. Tyrell William	male	36.0	1	0	19877	78.85	C46	S

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
167	168	0	3	Skoog, Mrs. William (Anna Bernhardina Karlsson)	female	45.0	1	4	347088	27.90	NaN	S

In [4]:
```
#selcet the important column
df.drop(columns=['PassengerId','Name','Ticket','Cabin'],inplace=True)
```

In [5]:
```
df.sample(5)
```

Out[5]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
597	0	3	male	49.0	0	0	0.0000	S
533	1	3	female	NaN	0	2	22.3583	C
277	0	2	male	NaN	0	0	0.0000	S
152	0	3	male	55.5	0	0	8.0500	S
82	1	3	female	NaN	0	0	7.7875	Q

Step 1 - Train-Test-Split

In [6]:
```
X_train,X_test,y_train,y_test = train_test_split(df.drop(columns=['Survived']),
                                                 df['Survived'],
                                                 test_size=0.2,
                                                random_state=42)
```

In [7]:
```
print(X_train.head(2))
```

```
     Pclass   Sex   Age  SibSp  Parch  Fare Embarked
331       1  male  45.5      0      0  28.5        S
733       2  male  23.0      0      0  13.0        S
```

In [8]:
```
print(y_train.head())
```

```
331    0
733    0
382    0
704    0
813    0
Name: Survived, dtype: int64
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

In [10]:
```
# Applying imputation

si_age = SimpleImputer()
si_embarked = SimpleImputer(strategy='most_frequent')

X_train_age = si_age.fit_transform(X_train[['Age']])
X_train_embarked = si_embarked.fit_transform(X_train[['Embarked']])

X_test_age = si_age.transform(X_test[['Age']])
X_test_embarked = si_embarked.transform(X_test[['Embarked']])
```

In [11]:
```
print(X_train_embarked)
```

```
[['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['C']
 ['S']
 ['C']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['C']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['C']
 ['C']
 ['S']
 ['S']
 ['C']
 ['S']
 ['C']
 ['S']
 ['Q']
 ['Q']
 ['S']
 ['S']
 ['S']
 ['C']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
 ['S']
```

['S']
['S']
['C']
['Q']
['C']
['S']
['S']
['S']
['S']
['C']
['Q']
['S']
['S']
['C']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['Q']
['C']
['C']
['S']

['S']
['Q']
['C']
['C']
['S']
['S']
['C']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['S']
['Q']
['C']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['Q']
['S']
['C']
['S']
['C']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['S']
['S']

['S']
['Q']
['S']
['C']
['Q']
['S']
['S']
['S']
['S']
['C']
['C']
['Q']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['C']
['S']
['S']
['C']
['S']
['S']
['S']
['C']
['S']
['Q']
['C']
['C']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['C']
['S']
['C']
['Q']
['C']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['C']
['S']
['Q']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['S']
['S']
['C']
['S']

['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['Q']
['S']
['S']
['S']
['S']
['C']
['S']
['C']
['Q']
['Q']
['C']
['Q']
['S']
['C']
['S']
['C']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['C']
['S']
['C']
['C']
['Q']
['S']
['S']
['S']
['S']
['S']
['C']
['Q']
['S']

['S']
['C']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['Q']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['C']
['S']
['S']
['S']
['S']
['C']
['S']
['Q']
['C']
['S']
['S']
['S']
['S']
['Q']
['C']
['S']
['S']
['S']

['S']
['Q']
['Q']
['S']
['C']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['S']
['C']
['S']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']

['S']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['Q']
['S']
['S']
['C']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']

['S']
['S']
['S']
['C']
['Q']
['C']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['C']
['S']
['S']
['C']
['S']
['C']
['S']
['S']
['S']
['S']
['C']
['S']
['C']
['S']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['Q']
['Q']
['S']
['Q']

['Q']
['Q']
['S']
['S']
['S']
['C']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['C']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['C']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['C']

['S']
['S']
['S']
['S']
['Q']
['Q']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['Q']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['Q']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['C']
['C']
['C']
['S']
['S']
['S']
['S']
['S']
['S']
['S']
['Q']
['Q']
['S']
['S']
['S']
['S']
['C']
['S']
['S']
['S']
['S']

```
['S']
['S']]
```

One-Hot Encoding with Sex and Embarked

In [12]:
```
ohe_sex = OneHotEncoder(sparse=False,handle_unknown='ignore')
ohe_embarked = OneHotEncoder(sparse=False,handle_unknown='ignore')

X_train_sex = ohe_sex.fit_transform(X_train[['Sex']])
X_train_embarked = ohe_embarked.fit_transform(X_train_embarked)

X_test_sex = ohe_sex.transform(X_test[['Sex']])
X_test_embarked = ohe_embarked.transform(X_test_embarked)
```

In [13]:
```
X_train_embarked
```

Out[13]:
```
array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]])
```

In [14]:
```
X_train.head(2)
```

Out[14]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
331	1	male	45.5	0	0	28.5	S
733	2	male	23.0	0	0	13.0	S

In [15]:
```
X_train_rem = X_train.drop(columns=['Sex','Age','Embarked'])
```

In [16]:
```
X_test_rem = X_test.drop(columns=['Sex','Age','Embarked'])
```

Now concatenet the X_train_embarked column and X_train column and also same on X_test

In [17]:
```
X_train_transformed = np.concatenate((X_train_rem,X_train_age,X_train_sex,X_train_embarked
X_test_transformed = np.concatenate((X_test_rem,X_test_age,X_test_sex,X_test_embarked),axi
```

In [18]:
```
X_test_transformed.shape
```

Out[18]:
```
(179, 10)
```

In [19]:
```
clf = DecisionTreeClassifier()
clf.fit(X_train_transformed,y_train)
```

Out[19]:
```
DecisionTreeClassifier()
```

In [20]:
```
y_pred = clf.predict(X_test_transformed)
```

```
y_pred
```

Out[20]:
```
array([0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1], dtype=int64)
```

In [21]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[21]:
```
0.776536312849162
```

In [22]:
```
import pickle
```

In [23]:
```
pickle.dump(ohe_sex,open('ohe_sex.pkl','wb'))
pickle.dump(ohe_embarked,open('ohe_embarked.pkl','wb'))
pickle.dump(clf,open('clf.pkl','wb'))
```

In [ ]:
```

```

PipiLine in Machine Learning

Pred. without PipeLine

In [1]:
```
import pickle
import numpy as np
```

In [2]:
```
ohe_sex = pickle.load(open('ohe_sex.pkl','rb'))
ohe_embarked = pickle.load(open('ohe_embarked.pkl','rb'))
clf = pickle.load(open('clf.pkl','rb'))
```

In [3]:
```
# Process Flow >>>>>>> Pclass/gender/age/SibSp/Parch/Fare/Embarked

test_input = np.array([2, 'male', 31.0, 0, 0, 10.5, 'S'],dtype=object).reshape(1,7)
```

In [4]:
```
test_input
```

Out[4]:
```
array([[2, 'male', 31.0, 0, 0, 10.5, 'S']], dtype=object)
```

In [5]:
```
test_input_sex = ohe_sex.transform(test_input[:,1].reshape(1,1))
```

In [6]:
```
test_input_sex
```

Out[6]:
```
array([[0., 1.]])
```

In [7]:
```
test_input_embarked = ohe_embarked.transform(test_input[:,-1].reshape(1,1))
```

In [8]:
```
test_input_embarked
```

Out[8]:
```
array([[0., 0., 1.]])
```

In [ ]:
```

```

Step by Step apply Pipeline

With PipeLine

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

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline,make_pipeline
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
df = pd.read_csv('train.csv')
```

In [3]:
```
df.sample(5)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
760	761	0	3	Garfirth, Mr. John	male	NaN	0	0	358585	14.5000	NaN	S
423	424	0	3	Danbom, Mrs. Ernst Gilbert (Anna Sigrid Maria ...	female	28.0	1	1	347080	14.4000	NaN	S
210	211	0	3	Ali, Mr. Ahmed	male	24.0	0	0	SOTON/O.Q. 3101311	7.0500	NaN	S
503	504	0	3	Laitinen, Miss. Kristina Sofia	female	37.0	0	0	4135	9.5875	NaN	S
386	387	0	3	Goodwin, Master. Sidney Leonard	male	1.0	5	2	CA 2144	46.9000	NaN	S

In [4]:
```
#selcet the important column
df.drop(columns=['PassengerId','Name','Ticket','Cabin'],inplace=True)
```

In [5]:
```
df.sample(5)
```

Out[5]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
700	1	1	female	18.0	1	0	227.5250	C

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
621	1	1	male	42.0	1	0	52.5542	S
520	1	1	female	30.0	0	0	93.5000	S
60	0	3	male	22.0	0	0	7.2292	C
462	0	1	male	47.0	0	0	38.5000	S

Step 1 - Train-Test-Split

In [6]:
```
X_train,X_test,y_train,y_test = train_test_split(df.drop(columns=['Survived']),
                                                 df['Survived'],
                                                 test_size=0.2,
                                                random_state=42)
```

In [7]:
```
print(X_train.head(2))
```

```
     Pclass   Sex   Age  SibSp  Parch  Fare Embarked
331       1  male  45.5      0      0  28.5        S
733       2  male  23.0      0      0  13.0        S
```

In [8]:
```
print(y_train.head())
```

```
331    0
733    0
382    0
704    0
813    0
Name: Survived, dtype: int64
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

Step.2 Imputation Transformer

In [24]:
```
# imputation transformer
trf1 = ColumnTransformer([
    ('impute_age',SimpleImputer(),[2]),
    ('impute_embarked',SimpleImputer(strategy='most_frequent'),[6])
],remainder='passthrough')
```

Step.3 One-Hot Encoding

In [25]:
```
trf2 = ColumnTransformer([
```

```
    ('ohe_sex_embarked',OneHotEncoder(sparse=False,handle_unknown='ignore'),[1,6])
],remainder='passthrough')
```

Step.4 Scaling

In [26]:
```
trf3 = ColumnTransformer([
    ('scale',MinMaxScaler(),slice(0,10))
])
```

Step.5 Feature Selection

In [27]:
```
trf4 = SelectKBest(score_func=chi2,k=8)
```

Step.6 Train the Model

In [28]:
```
trf5 = DecisionTreeClassifier()
```

Step.7 Create PipeLine

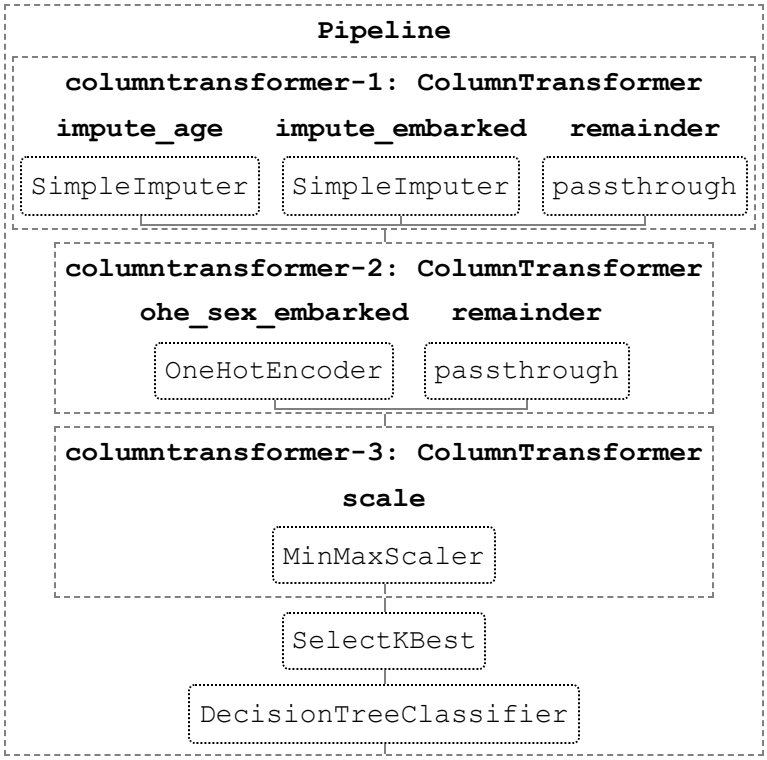
In [29]:
```
pipe = Pipeline([
    ('trf1',trf1),
    ('trf2',trf2),
    ('trf3',trf3),
    ('trf4',trf4),
    ('trf5',trf5)
])
```

Step7.1 Create Pipeline with Make_PipeLine

In [30]:
```
pipe = make_pipeline(trf1,trf2,trf3,trf4,trf5)
```

In [31]:
```
pipe.fit(X_train,y_train)
```

Out[31]:



In [18]:
```
pipe.named_steps
```

Out[18]:
```
{'columntransformer-1': ColumnTransformer(remainder='passthrough',
                  transformers=[('impute_age', SimpleImputer(), [2]),
                                ('impute_embarked',
                                 SimpleImputer(strategy='most_frequent'),
                                 [6])]),
 'columntransformer-2': ColumnTransformer(remainder='passthrough',
                  transformers=[('ohe_sex_embarked',
                                 OneHotEncoder(handle_unknown='ignore',
                                               sparse=False),
                                 [1, 6])]),
 'columntransformer-3': ColumnTransformer(transformers=[('scale', MinMaxScaler(), slice(0,
10, None))]),
 'selectkbest': SelectKBest(k=8, score_func=<function chi2 at 0x000002836E0DED30>),
 'decisiontreeclassifier': DecisionTreeClassifier()}
```

In [19]:
```
# Display Pipeline

from sklearn import set_config
set_config(display='diagram')
```

In [20]:
```
# Predict
y_pred = pipe.predict(X_test)
```

In [21]:
```
y_pred
```

Out[21]:
```
array([1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0], dtype=int64)
```

In [22]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[22]:
```
0.6256983240223464
```

Cross Validation using Pipeline

In [23]:
```
# cross validation using cross_val_score
from sklearn.model_selection import cross_val_score
cross_val_score(pipe, X_train, y_train, cv=5, scoring='accuracy').mean()
```

Out[23]:
```
0.6391214419383433
```

In [ ]:

Handling Mixed Data in Machine Learning

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

In [2]:
```
df = pd.read_csv('titanic.csv')
```

In [3]:
```
df.head()
```

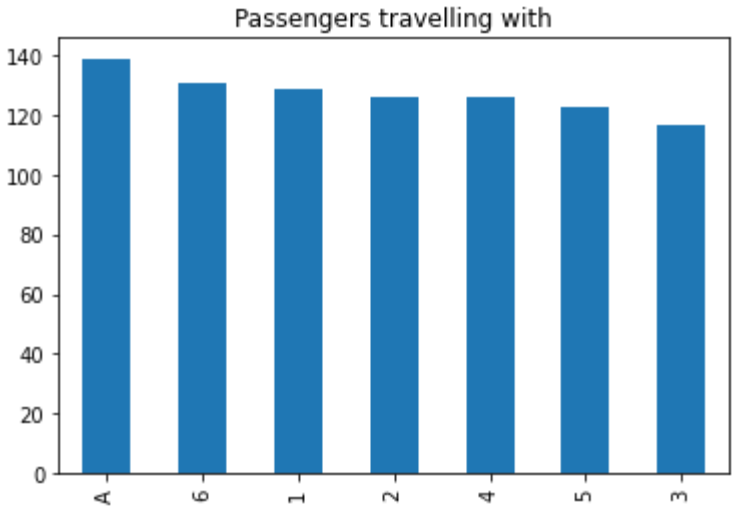
Out[3]:

	Cabin	Ticket	number	Survived
0	NaN	A/5 21171	5	0
1	C85	PC 17599	3	1
2	NaN	STON/O2. 3101282	6	1
3	C123	113803	3	1
4	NaN	373450	A	0

In [4]:
```
df['number'].unique()
```

Out[4]:
```
array(['5', '3', '6', 'A', '2', '1', '4'], dtype=object)
```

In [5]:
```
import matplotlib.pylab as plt
fig = df['number'].value_counts().plot.bar()
fig.set_title('Passengers travelling with')
plt.show()
```



Extract numerical part and Catagorical part

In [6]:
```
df['number_numerical'] = pd.to_numeric(df["number"],errors='coerce',downcast='integer')

df['number_categorical'] = np.where(df['number_numerical'].isnull(),df['number'],np.nan)

df.head()
```

Out[6]:

	Cabin	Ticket	number	Survived	number_numerical	number_categorical
0	NaN	A/5 21171	5	0	5.0	NaN
1	C85	PC 17599	3	1	3.0	NaN
2	NaN	STON/O2. 3101282	6	1	6.0	NaN

	Cabin	Ticket	number	Survived	number_numerical	number_categorical
3	C123	113803	3	1	3.0	NaN
4	NaN	373450	A	0	NaN	A

In [7]:
```
df['Cabin'].unique()
```

Out[7]:
```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148'], dtype=object)
```

In [8]:
```
df['Ticket'].unique()
```

Out[8]:
```
array(['A/5 21171', 'PC 17599', 'STON/O2. 3101282', '113803', '373450',
       '330877', '17463', '349909', '347742', '237736', 'PP 9549',
       '113783', 'A/5. 2151', '347082', '350406', '248706', '382652',
       '244373', '345763', '2649', '239865', '248698', '330923', '113788',
       '347077', '2631', '19950', '330959', '349216', 'PC 17601',
       'PC 17569', '335677', 'C.A. 24579', 'PC 17604', '113789', '2677',
       'A./5. 2152', '345764', '2651', '7546', '11668', '349253',
       'SC/Paris 2123', '330958', 'S.C./A.4. 23567', '370371', '14311',
       '2662', '349237', '3101295', 'A/4. 39886', 'PC 17572', '2926',
       '113509', '19947', 'C.A. 31026', '2697', 'C.A. 34651', 'CA 2144',
       '2669', '113572', '36973', '347088', 'PC 17605', '2661',
       'C.A. 29395', 'S.P. 3464', '3101281', '315151', 'C.A. 33111',
       'S.O.C. 14879', '2680', '1601', '348123', '349208', '374746',
       '248738', '364516', '345767', '345779', '330932', '113059',
       'SO/C 14885', '3101278', 'W./C. 6608', 'SOTON/OQ 392086', '343275',
       '343276', '347466', 'W.E.P. 5734', 'C.A. 2315', '364500', '374910',
       'PC 17754', 'PC 17759', '231919', '244367', '349245', '349215',
       '35281', '7540', '3101276', '349207', '343120', '312991', '349249',
       '371110', '110465', '2665', '324669', '4136', '2627',
       'STON/O 2. 3101294', '370369', 'PC 17558', 'A4. 54510', '27267',
       '370372', 'C 17369', '2668', '347061', '349241',
       'SOTON/O.Q. 3101307', 'A/5. 3337', '228414', 'C.A. 29178',
       'SC/PARIS 2133', '11752', '7534', 'PC 17593', '2678', '347081',
       'STON/O2. 3101279', '365222', '231945', 'C.A. 33112', '350043',
       '230080', '244310', 'S.O.P. 1166', '113776', 'A.5. 11206',
       'A/5. 851', 'Fa 265302', 'PC 17597', '35851', 'SOTON/OQ 392090',
       '315037', 'CA. 2343', '371362', 'C.A. 33595', '347068', '315093',
       '363291', '113505', 'PC 17318', '111240', 'STON/O 2. 3101280',
       '17764', '350404', '4133', 'PC 17595', '250653', 'LINE',
       'SC/PARIS 2131', '230136', '315153', '113767', '370365', '111428',
       '364849', '349247', '234604', '28424', '350046', 'PC 17610',
       '368703', '4579', '370370', '248747', '345770', '3101264', '2628',
       'A/5 3540', '347054', '2699', '367231', '112277',
       'SOTON/O.Q. 3101311', 'F.C.C. 13528', 'A/5 21174', '250646',
```

'367229', '35273', 'STON/O2. 3101283', '243847', '11813',
'W/C 14208', 'SOTON/OQ 392089', '220367', '21440', '349234',
'19943', 'PP 4348', 'SW/PP 751', 'A/5 21173', '236171', '347067',
'237442', 'C.A. 29566', 'W./C. 6609', '26707', 'C.A. 31921',
'28665', 'SCO/W 1585', '367230', 'W./C. 14263',
'STON/O 2. 3101275', '2694', '19928', '347071', '250649', '11751',
'244252', '362316', '113514', 'A/5. 3336', '370129', '2650',
'PC 17585', '110152', 'PC 17755', '230433', '384461', '110413',
'112059', '382649', 'C.A. 17248', '347083', 'PC 17582', 'PC 17760',
'113798', '250644', 'PC 17596', '370375', '13502', '347073',
'239853', 'C.A. 2673', '336439', '347464', '345778', 'A/5. 10482',
'113056', '349239', '345774', '349206', '237798', '370373',
'19877', '11967', 'SC/Paris 2163', '349236', '349233', 'PC 17612',
'2693', '113781', '19988', '9234', '367226', '226593', 'A/5 2466',
'17421', 'PC 17758', 'P/PP 3381', 'PC 17485', '11767', 'PC 17608',
'250651', '349243', 'F.C.C. 13529', '347470', '29011', '36928',
'16966', 'A/5 21172', '349219', '234818', '345364', '28551',
'111361', '113043', 'PC 17611', '349225', '7598', '113784',
'248740', '244361', '229236', '248733', '31418', '386525',
'C.A. 37671', '315088', '7267', '113510', '2695', '2647', '345783',
'237671', '330931', '330980', 'SC/PARIS 2167', '2691',
'SOTON/O.Q. 3101310', 'C 7076', '110813', '2626', '14313',
'PC 17477', '11765', '3101267', '323951', 'C 7077', '113503',
'2648', '347069', 'PC 17757', '2653', 'STON/O 2. 3101293',
'349227', '27849', '367655', 'SC 1748', '113760', '350034',
'3101277', '350052', '350407', '28403', '244278', '240929',
'STON/O 2. 3101289', '341826', '4137', '315096', '28664', '347064',
'29106', '312992', '349222', '394140', 'STON/O 2. 3101269',
'343095', '28220', '250652', '28228', '345773', '349254',
'A/5. 13032', '315082', '347080', 'A/4. 34244', '2003', '250655',
'364851', 'SOTON/O.Q. 392078', '110564', '376564', 'SC/AH 3085',
'STON/O 2. 3101274', '13507', 'C.A. 18723', '345769', '347076',
'230434', '65306', '33638', '113794', '2666', '113786', '65303',
'113051', '17453', 'A/5 2817', '349240', '13509', '17464',
'F.C.C. 13531', '371060', '19952', '364506', '111320', '234360',
'A/S 2816', 'SOTON/O.Q. 3101306', '113792', '36209', '323592',
'315089', 'SC/AH Basle 541', '7553', '31027', '3460', '350060',
'3101298', '239854', 'A/5 3594', '4134', '11771', 'A.5. 18509',
'65304', 'SOTON/OQ 3101317', '113787', 'PC 17609', 'A/4 45380',
'36947', 'C.A. 6212', '350035', '315086', '364846', '330909',
'4135', '26360', '111427', 'C 4001', '382651', 'SOTON/OQ 3101316',
'PC 17473', 'PC 17603', '349209', '36967', 'C.A. 34260', '226875',
'349242', '12749', '349252', '2624', '2700', '367232',
'W./C. 14258', 'PC 17483', '3101296', '29104', '2641', '2690',
'315084', '113050', 'PC 17761', '364498', '13568', 'WE/P 5735',
'2908', '693', 'SC/PARIS 2146', '244358', '330979', '2620',
'347085', '113807', '11755', '345572', '372622', '349251',
'218629', 'SOTON/OQ 392082', 'SOTON/O.Q. 392087', 'A/4 48871',
'349205', '2686', '350417', 'S.W./PP 752', '11769', 'PC 17474',
'14312', 'A/4. 20589', '358585', '243880', '2689',
'STON/O 2. 3101286', '237789', '13049', '3411', '237565', '13567',
'14973', 'A./5. 3235', 'STON/O 2. 3101273', 'A/5 3902', '364848',
'SC/AH 29037', '248727', '2664', '349214', '113796', '364511',
'111426', '349910', '349246', '113804', 'SOTON/O.Q. 3101305',
'370377', '364512', '220845', '31028', '2659', '11753', '350029',
'54636', '36963', '219533', '349224', '334912', '27042', '347743',
'13214', '112052', '237668', 'STON/O 2. 3101292', '350050',
'349231', '13213', 'S.O./P.P. 751', 'CA. 2314', '349221', '8475',
'330919', '365226', '349223', '29751', '2623', '5727', '349210',
'STON/O 2. 3101285', '234686', '312993', 'A/5 3536', '19996',
'29750', 'F.C. 12750', 'C.A. 24580', '244270', '239856', '349912',
'342826', '4138', '330935', '6563', '349228', '350036', '24160',
'17474', '349256', '2672', '113800', '248731', '363592', '35852',
'348121', 'PC 17475', '36864', '350025', '223596', 'PC 17476',
'PC 17482', '113028', '7545', '250647', '348124', '34218', '36568',
'347062', '350048', '12233', '250643', '113806', '315094', '36866',

```
'236853', 'STON/O2. 3101271', '239855', '28425', '233639',
'349201', '349218', '16988', '376566', 'STON/O 2. 3101288',
'250648', '113773', '335097', '29103', '392096', '345780',
'349204', '350042', '29108', '363294', 'SOTON/O2 3101272', '2663',
'347074', '112379', '364850', '8471', '345781', '350047',
'S.O./P.P. 3', '2674', '29105', '347078', '383121', '36865',
'2687', '113501', 'W./C. 6607', 'SOTON/O.Q. 3101312', '374887',
'3101265', '12460', 'PC 17600', '349203', '28213', '17465',
'349244', '2685', '2625', '347089', '347063', '112050', '347087',
'248723', '3474', '28206', '364499', '112058', 'STON/O2. 3101290',
'S.C./PARIS 2079', 'C 7075', '315098', '19972', '368323', '367228',
'2671', '347468', '2223', 'PC 17756', '315097', '392092', '11774',
'SOTON/O2 3101287', '2683', '315090', 'C.A. 5547', '349213',
'347060', 'PC 17592', '392091', '113055', '2629', '350026',
'28134', '17466', '233866', '236852', 'SC/PARIS 2149', 'PC 17590',
'345777', '349248', '695', '345765', '2667', '349212', '349217',
'349257', '7552', 'C.A./SOTON 34068', 'SOTON/OQ 392076', '211536',
'112053', '111369', '370376'], dtype=object)
```

In [9]:

```
df['cabin_num'] = df['Cabin'].str.extract('(\d+)') # captures numerical part
df['cabin_cat'] = df['Cabin'].str[0] # captures the first letter

df.head()
```

Out[9]:

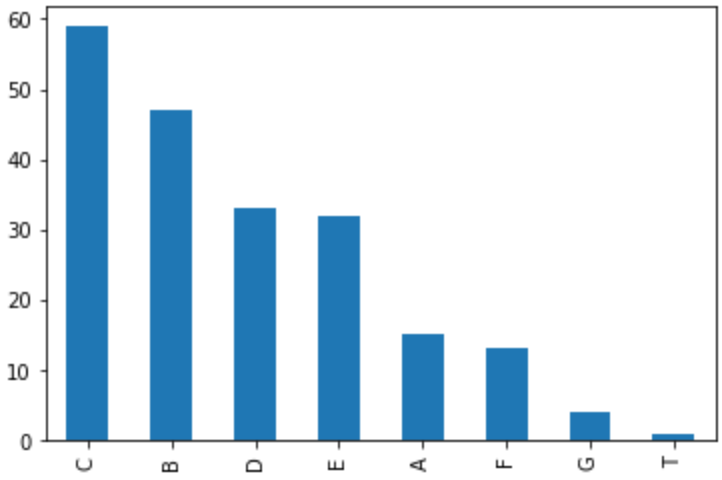
	Cabin	Ticket	number	Survived	number_numerical	number_categorical	cabin_num	cabin_cat
0	NaN	A/5 21171	5	0	5.0	NaN	NaN	NaN
1	C85	PC 17599	3	1	3.0	NaN	85	C
2	NaN	STON/O2. 3101282	6	1	6.0	NaN	NaN	NaN
3	C123	113803	3	1	3.0	NaN	123	C
4	NaN	373450	A	0	NaN	A	NaN	NaN

In [10]:

```
df['cabin_cat'].value_counts().plot(kind='bar')
```

Out[10]:

```
<AxesSubplot:>
```



In [11]:

```
# extract the last bit of ticket as number
df['ticket_num'] = df['Ticket'].apply(lambda s: s.split()[-1])
df['ticket_num'] = pd.to_numeric(df['ticket_num'],
                                 errors='coerce',
                                 downcast='integer')

# extract the first part of ticket as category
```

```
df['ticket_cat'] = df['Ticket'].apply(lambda s: s.split()[0])
df['ticket_cat'] = np.where(df['ticket_cat'].str.isdigit(), np.nan,
                            df['ticket_cat'])

df.head(20)
```

Out[11]:

	Cabin	Ticket	number	Survived	number_numerical	number_categorical	cabin_num	cabin_cat	ticket_num
0	NaN	A/5 21171	5	0	5.0	NaN	NaN	NaN	21171.0
1	C85	PC 17599	3	1	3.0	NaN	85	C	17599.0
2	NaN	STON/O2. 3101282	6	1	6.0	NaN	NaN	NaN	3101282.0
3	C123	113803	3	1	3.0	NaN	123	C	113803.0
4	NaN	373450	A	0	NaN	A	NaN	NaN	373450.0
5	NaN	330877	2	0	2.0	NaN	NaN	NaN	330877.0
6	E46	17463	2	0	2.0	NaN	46	E	17463.0
7	NaN	349909	5	0	5.0	NaN	NaN	NaN	349909.0
8	NaN	347742	1	1	1.0	NaN	NaN	NaN	347742.0
9	NaN	237736	A	1	NaN	A	NaN	NaN	237736.0
10	G6	PP 9549	1	1	1.0	NaN	6	G	9549.0
11	C103	113783	1	1	1.0	NaN	103	C	113783.0
12	NaN	A/5. 2151	3	0	3.0	NaN	NaN	NaN	2151.0
13	NaN	347082	3	0	3.0	NaN	NaN	NaN	347082.0
14	NaN	350406	5	0	5.0	NaN	NaN	NaN	350406.0
15	NaN	248706	3	1	3.0	NaN	NaN	NaN	248706.0
16	NaN	382652	3	0	3.0	NaN	NaN	NaN	382652.0
17	NaN	244373	2	1	2.0	NaN	NaN	NaN	244373.0
18	NaN	345763	5	0	5.0	NaN	NaN	NaN	345763.0
19	NaN	2649	4	1	4.0	NaN	NaN	NaN	2649.0

In [12]:
```
df['ticket_cat'].unique()
```

Out[12]:
```
array(['A/5', 'PC', 'STON/O2.', nan, 'PP', 'A/5.', 'C.A.', 'A./5.',
       'SC/Paris', 'S.C./A.4.', 'A/4.', 'CA', 'S.P.', 'S.O.C.', 'SO/C',
       'W./C.', 'SOTON/OQ', 'W.E.P.', 'STON/O', 'A4.', 'C', 'SOTON/O.Q.',
       'SC/PARIS', 'S.O.P.', 'A.5.', 'Fa', 'CA.', 'LINE', 'F.C.C.', 'W/C',
       'SW/PP', 'SCO/W', 'P/PP', 'SC', 'SC/AH', 'A/S', 'A/4', 'WE/P',
       'S.W./PP', 'S.O./P.P.', 'F.C.', 'SOTON/O2', 'S.C./PARIS',
       'C.A./SOTON'], dtype=object)
```

In [13]:
```
df['ticket_num'].unique()
```

Out[13]:
```
array([2.117100e+04, 1.759900e+04, 3.101282e+06, 1.138030e+05,
       3.734500e+05, 3.308770e+05, 1.746300e+04, 3.499090e+05,
       3.477420e+05, 2.377360e+05, 9.549000e+03, 1.137830e+05,
       2.151000e+03, 3.470820e+05, 3.504060e+05, 2.487060e+05,
       3.826520e+05, 2.443730e+05, 3.457630e+05, 2.649000e+03,
```

2.398650e+05, 2.486980e+05, 3.309230e+05, 1.137880e+05,
3.470770e+05, 2.631000e+03, 1.995000e+04, 3.309590e+05,
3.492160e+05, 1.760100e+04, 1.756900e+04, 3.356770e+05,
2.457900e+04, 1.760400e+04, 1.137890e+05, 2.677000e+03,
2.152000e+03, 3.457640e+05, 2.651000e+03, 7.546000e+03,
1.166800e+04, 3.492530e+05, 2.123000e+03, 3.309580e+05,
2.356700e+04, 3.703710e+05, 1.431100e+04, 2.662000e+03,
3.492370e+05, 3.101295e+06, 3.988600e+04, 1.757200e+04,
2.926000e+03, 1.135090e+05, 1.994700e+04, 3.102600e+04,
2.697000e+03, 3.465100e+04, 2.144000e+03, 2.669000e+03,
1.135720e+05, 3.697300e+04, 3.470880e+05, 1.760500e+04,
2.661000e+03, 2.939500e+04, 3.464000e+03, 3.101281e+06,
3.151510e+05, 3.311100e+04, 1.487900e+04, 2.680000e+03,
1.601000e+03, 3.481230e+05, 3.492080e+05, 3.747460e+05,
2.487380e+05, 3.645160e+05, 3.457670e+05, 3.457790e+05,
3.309320e+05, 1.130590e+05, 1.488500e+04, 3.101278e+06,
6.608000e+03, 3.920860e+05, 3.432750e+05, 3.432760e+05,
3.474660e+05, 5.734000e+03, 2.315000e+03, 3.645000e+05,
3.749100e+05, 1.775400e+04, 1.775900e+04, 2.319190e+05,
2.443670e+05, 3.492450e+05, 3.492150e+05, 3.528100e+04,
7.540000e+03, 3.101276e+06, 3.492070e+05, 3.431200e+05,
3.129910e+05, 3.492490e+05, 3.711100e+05, 1.104650e+05,
2.665000e+03, 3.246690e+05, 4.136000e+03, 2.627000e+03,
3.101294e+06, 3.703690e+05, 1.755800e+04, 5.451000e+04,
2.726700e+04, 3.703720e+05, 1.736900e+04, 2.668000e+03,
3.470610e+05, 3.492410e+05, 3.101307e+06, 3.337000e+03,
2.284140e+05, 2.917800e+04, 2.133000e+03, 1.175200e+04,
7.534000e+03, 1.759300e+04, 2.678000e+03, 3.470810e+05,
3.101279e+06, 3.652220e+05, 2.319450e+05, 3.311200e+04,
3.500430e+05, 2.300800e+05, 2.443100e+05, 1.166000e+03,
1.137760e+05, 1.120600e+04, 8.510000e+02, 2.653020e+05,
1.759700e+04, 3.585100e+04, 3.920900e+05, 3.150370e+05,
2.343000e+03, 3.713620e+05, 3.359500e+04, 3.470680e+05,
3.150930e+05, 3.632910e+05, 1.135050e+05, 1.731800e+04,
1.112400e+05, 3.101280e+06, 1.776400e+04, 3.504040e+05,
4.133000e+03, 1.759500e+04, 2.506530e+05,   nan,
2.131000e+03, 2.301360e+05, 3.151530e+05, 1.137670e+05,
3.703650e+05, 1.114280e+05, 3.648490e+05, 3.492470e+05,
2.346040e+05, 2.842400e+04, 3.500460e+05, 1.761000e+04,
3.687030e+05, 4.579000e+03, 3.703700e+05, 2.487470e+05,
3.457700e+05, 3.101264e+06, 2.628000e+03, 3.540000e+03,
3.470540e+05, 2.699000e+03, 3.672310e+05, 1.122770e+05,
3.101311e+06, 1.352800e+04, 2.117400e+04, 2.506460e+05,
3.672290e+05, 3.527300e+04, 3.101283e+06, 2.438470e+05,
1.181300e+04, 1.420800e+04, 3.920890e+05, 2.203670e+05,
2.144000e+04, 3.492340e+05, 1.994300e+04, 4.348000e+03,
7.510000e+02, 2.117300e+04, 2.361710e+05, 3.470670e+05,
2.374420e+05, 2.956600e+04, 6.609000e+03, 2.670700e+04,
3.192100e+04, 2.866500e+04, 1.585000e+03, 3.672300e+05,
1.426300e+04, 3.101275e+06, 2.694000e+03, 1.992800e+04,
3.470710e+05, 2.506490e+05, 1.175100e+04, 2.442520e+05,
3.623160e+05, 1.135140e+05, 3.336000e+03, 3.701290e+05,
2.650000e+03, 1.758500e+04, 1.101520e+05, 1.775500e+04,
2.304330e+05, 3.844610e+05, 1.104130e+05, 1.120590e+05,
3.826490e+05, 1.724800e+04, 3.470830e+05, 1.758200e+04,
1.776000e+04, 1.137980e+05, 2.506440e+05, 1.759600e+04,
3.703750e+05, 1.350200e+04, 3.470730e+05, 2.398530e+05,
2.673000e+03, 3.364390e+05, 3.474640e+05, 3.457780e+05,
1.048200e+04, 1.130560e+05, 3.492390e+05, 3.457740e+05,
3.492060e+05, 2.377980e+05, 3.703730e+05, 1.987700e+04,
1.196700e+04, 2.163000e+03, 3.492360e+05, 3.492330e+05,
1.761200e+04, 2.693000e+03, 1.137810e+05, 1.998800e+04,
9.234000e+03, 3.672260e+05, 2.265930e+05, 2.466000e+03,
1.742100e+04, 1.775800e+04, 3.381000e+03, 1.748500e+04,
1.176700e+04, 1.760800e+04, 2.506510e+05, 3.492430e+05,
1.352900e+04, 3.474700e+05, 2.901100e+04, 3.692800e+04,

1.696600e+04, 2.117200e+04, 3.492190e+05, 2.348180e+05,
3.453640e+05, 2.855100e+04, 1.113610e+05, 1.130430e+05,
1.761100e+04, 3.492250e+05, 7.598000e+03, 1.137840e+05,
2.487400e+05, 2.443610e+05, 2.292360e+05, 2.487330e+05,
3.141800e+04, 3.865250e+05, 3.767100e+04, 3.150880e+05,
7.267000e+03, 1.135100e+05, 2.695000e+03, 2.647000e+03,
3.457830e+05, 2.376710e+05, 3.309310e+05, 3.309800e+05,
2.167000e+03, 2.691000e+03, 3.101310e+06, 7.076000e+03,
1.108130e+05, 2.626000e+03, 1.431300e+04, 1.747700e+04,
1.176500e+04, 3.101267e+06, 3.239510e+05, 7.077000e+03,
1.135030e+05, 2.648000e+03, 3.470690e+05, 1.775700e+04,
2.653000e+03, 3.101293e+06, 3.492270e+05, 2.784900e+04,
3.676550e+05, 1.748000e+03, 1.137600e+05, 3.500340e+05,
3.101277e+06, 3.500520e+05, 3.504070e+05, 2.840300e+04,
2.442780e+05, 2.409290e+05, 3.101289e+06, 3.418260e+05,
4.137000e+03, 3.150960e+05, 2.866400e+04, 3.470640e+05,
2.910600e+04, 3.129920e+05, 3.492220e+05, 3.941400e+05,
3.101269e+06, 3.430950e+05, 2.822000e+04, 2.506520e+05,
2.822800e+04, 3.457730e+05, 3.492540e+05, 1.303200e+04,
3.150820e+05, 3.470800e+05, 3.424400e+04, 2.003000e+03,
2.506550e+05, 3.648510e+05, 3.920780e+05, 1.105640e+05,
3.765640e+05, 3.085000e+03, 3.101274e+06, 1.350700e+04,
1.872300e+04, 3.457690e+05, 3.470760e+05, 2.304340e+05,
6.530600e+04, 3.363800e+04, 1.137940e+05, 2.666000e+03,
1.137860e+05, 6.530300e+04, 1.130510e+05, 1.745300e+04,
2.817000e+03, 3.492400e+05, 1.350900e+04, 1.746400e+04,
1.353100e+04, 3.710600e+05, 1.995200e+04, 3.645060e+05,
1.113200e+05, 2.343600e+05, 2.816000e+03, 3.101306e+06,
1.137920e+05, 3.620900e+04, 3.235920e+05, 3.150890e+05,
5.410000e+02, 7.553000e+03, 3.102700e+04, 3.460000e+03,
3.500600e+05, 3.101298e+06, 2.398540e+05, 3.594000e+03,
4.134000e+03, 1.177100e+04, 1.850900e+04, 6.530400e+04,
3.101317e+06, 1.137870e+05, 1.760900e+04, 4.538000e+04,
3.694700e+04, 6.212000e+03, 3.500350e+05, 3.150860e+05,
3.648460e+05, 3.309090e+05, 4.135000e+03, 2.636000e+04,
1.114270e+05, 4.001000e+03, 3.826510e+05, 3.101316e+06,
1.747300e+04, 1.760300e+04, 3.492090e+05, 3.696700e+04,
3.426000e+04, 2.268750e+05, 3.492420e+05, 1.274900e+04,
3.492520e+05, 2.624000e+03, 2.700000e+03, 3.672320e+05,
1.425800e+04, 1.748300e+04, 3.101296e+06, 2.910400e+04,
2.641000e+03, 2.690000e+03, 3.150840e+05, 1.130500e+05,
1.776100e+04, 3.644980e+05, 1.356800e+04, 5.735000e+03,
2.908000e+03, 6.930000e+02, 2.146000e+03, 2.443580e+05,
3.309790e+05, 2.620000e+03, 3.470850e+05, 1.138070e+05,
1.175500e+04, 3.455720e+05, 3.726220e+05, 3.492510e+05,
2.186290e+05, 3.920820e+05, 3.920870e+05, 4.887100e+04,
3.492050e+05, 2.686000e+03, 3.504170e+05, 7.520000e+02,
1.176900e+04, 1.747400e+04, 1.431200e+04, 2.058900e+04,
3.585850e+05, 2.438800e+05, 2.689000e+03, 3.101286e+06,
2.377890e+05, 1.304900e+04, 3.411000e+03, 2.375650e+05,
1.356700e+04, 1.497300e+04, 3.235000e+03, 3.101273e+06,
3.902000e+03, 3.648480e+05, 2.903700e+04, 2.487270e+05,
2.664000e+03, 3.492140e+05, 1.137960e+05, 3.645110e+05,
1.114260e+05, 3.499100e+05, 3.492460e+05, 1.138040e+05,
3.101305e+06, 3.703770e+05, 3.645120e+05, 2.208450e+05,
3.102800e+04, 2.659000e+03, 1.175300e+04, 3.500290e+05,
5.463600e+04, 3.696300e+04, 2.195330e+05, 3.492240e+05,
3.349120e+05, 2.704200e+04, 3.477430e+05, 1.321400e+04,
1.120520e+05, 2.376680e+05, 3.101292e+06, 3.500500e+05,
3.492310e+05, 1.321300e+04, 2.314000e+03, 3.492210e+05,
8.475000e+03, 3.309190e+05, 3.652260e+05, 3.492230e+05,
2.975100e+04, 2.623000e+03, 5.727000e+03, 3.492100e+05,
3.101285e+06, 2.346860e+05, 3.129930e+05, 3.536000e+03,
1.999600e+04, 2.975000e+04, 1.275000e+04, 2.458000e+04,
2.442700e+05, 2.398560e+05, 3.499120e+05, 3.428260e+05,
4.138000e+03, 3.309350e+05, 6.563000e+03, 3.492280e+05,

```
       3.500360e+05, 2.416000e+04, 3.492560e+05, 2.672000e+03,
       1.138000e+05, 2.487310e+05, 3.635920e+05, 3.585200e+04,
       3.481210e+05, 1.747500e+04, 3.686400e+04, 3.500250e+05,
       2.235960e+05, 1.747600e+04, 1.748200e+04, 1.130280e+05,
       7.545000e+03, 2.506470e+05, 3.481240e+05, 3.421800e+04,
       3.656800e+04, 3.470620e+05, 3.500480e+05, 1.223300e+04,
       2.506430e+05, 1.138060e+05, 3.150940e+05, 3.686600e+04,
       2.368530e+05, 3.101271e+06, 2.398550e+05, 2.842500e+04,
       2.336390e+05, 3.492010e+05, 3.492180e+05, 1.698800e+04,
       3.765660e+05, 3.101288e+06, 2.506480e+05, 1.137730e+05,
       3.350970e+05, 2.910300e+04, 3.920960e+05, 3.457800e+05,
       3.492040e+05, 3.500420e+05, 2.910800e+04, 3.632940e+05,
       3.101272e+06, 2.663000e+03, 3.470740e+05, 1.123790e+05,
       3.648500e+05, 8.471000e+03, 3.457810e+05, 3.500470e+05,
       3.000000e+00, 2.674000e+03, 2.910500e+04, 3.470780e+05,
       3.831210e+05, 3.686500e+04, 2.687000e+03, 1.135010e+05,
       6.607000e+03, 3.101312e+06, 3.748870e+05, 3.101265e+06,
       1.246000e+04, 1.760000e+04, 3.492030e+05, 2.821300e+04,
       1.746500e+04, 3.492440e+05, 2.685000e+03, 2.625000e+03,
       3.470890e+05, 3.470630e+05, 1.120500e+05, 3.470870e+05,
       2.487230e+05, 3.474000e+03, 2.820600e+04, 3.644990e+05,
       1.120580e+05, 3.101290e+06, 2.079000e+03, 7.075000e+03,
       3.150980e+05, 1.997200e+04, 3.683230e+05, 3.672280e+05,
       2.671000e+03, 3.474680e+05, 2.223000e+03, 1.775600e+04,
       3.150970e+05, 3.920920e+05, 1.177400e+04, 3.101287e+06,
       2.683000e+03, 3.150900e+05, 5.547000e+03, 3.492130e+05,
       3.470600e+05, 1.759200e+04, 3.920910e+05, 1.130550e+05,
       2.629000e+03, 3.500260e+05, 2.813400e+04, 1.746600e+04,
       2.338660e+05, 2.368520e+05, 2.149000e+03, 1.759000e+04,
       3.457770e+05, 3.492480e+05, 6.950000e+02, 3.457650e+05,
       2.667000e+03, 3.492120e+05, 3.492170e+05, 3.492570e+05,
       7.552000e+03, 3.406800e+04, 3.920760e+05, 2.115360e+05,
       1.120530e+05, 1.113690e+05, 3.703760e+05])
```

<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

Handling Date and Time variable in Machine Learning

In [14]:

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

In [15]:

```
date = pd.read_csv('orders.csv')
time = pd.read_csv('messages.csv')
```

In [16]:

```
date.head()
```

Out[16]:

	date	product_id	city_id	orders
0	2019-12-10	5628	25	3
1	2018-08-15	3646	14	157
2	2018-10-23	1859	25	1
3	2019-08-17	7292	25	1

	date	product_id	city_id	orders
4	2019-01-06	4344	25	3

In [17]:
```
time.head()
```

Out[17]:

	date	msg
0	2013-12-15 00:50:00	ищу на сегодня мужика 37
1	2014-04-29 23:40:00	ПАРЕНЬ БИ ИЩЕТ ДРУГА СЕЙЧАС!! СМС ММС 0955532826
2	2012-12-30 00:21:00	Днепр.м 43 позн.с д/ж *.o 067.16.34.576
3	2014-11-28 00:31:00	КИЕВ ИЩУ Д/Ж ДО 45 МНЕ СЕЙЧАС СКУЧНО 093 629 9...
4	2013-10-26 23:11:00	Зая я тебя никогда не обижу люблю тебя!) Даше

In [18]:
```
print(date.info())
print(time.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   date        1000 non-null   object
 1   product_id  1000 non-null   int64 
 2   city_id     1000 non-null   int64 
 3   orders      1000 non-null   int64 
dtypes: int64(3), object(1)
memory usage: 31.4+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   date    1000 non-null   object
 1   msg     1000 non-null   object
dtypes: object(2)
memory usage: 15.8+ KB
None
```

Working with dates

In [19]:
```
# Converting to datetime datatype
date['date'] = pd.to_datetime(date['date'])
```

In [20]:
```
date.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   date        1000 non-null   datetime64[ns]
 1   product_id  1000 non-null   int64         
 2   city_id     1000 non-null   int64         
```

```
 3   orders      1000 non-null   int64
dtypes: datetime64[ns](1), int64(3)
memory usage: 31.4 KB
```

In [21]:

```
#Extract year----
date['date_year'] = date['date'].dt.year

#Extract month----
date['date_month_no'] = date['date'].dt.month

#Extract month name----
date['date_month_name'] = date['date'].dt.month_name()

#Extract day----
date['date_day'] = date['date'].dt.day

#Extract day of week----
date['date_dow'] = date['date'].dt.dayofweek

#Extract day name----
date['date_dow_name'] = date['date'].dt.day_name()

#Extract date is weekend?----
date['date_is_weekend'] = np.where(date['date_dow_name'].isin(['Sunday', 'Saturday']), 1,0

#Extract date week----
date['date_week'] = date['date'].dt.week

#Extract quarter----
date['quarter'] = date['date'].dt.quarter

#Extract semester----
date['semester'] = np.where(date['quarter'].isin([1,2]), 1, 2)
```

```
C:\Users\HP\AppData\Local\Temp/ipykernel_75188/3985301858.py:23: FutureWarning: Series.dt.
weekofyear and Series.dt.week have been deprecated.  Please use Series.dt.isocalendar().we
ek instead.
  date['date_week'] = date['date'].dt.week
```

In [22]:

```
date.drop(columns=['product_id','city_id','orders']).head()
```

Out[22]:

	date	date_year	date_month_no	date_month_name	date_day	date_dow	date_dow_name	date_is_weekend	dat
0	2019-12-10	2019	12	December	10	1	Tuesday	0	
1	2018-08-15	2018	8	August	15	2	Wednesday	0	
2	2018-10-23	2018	10	October	23	1	Tuesday	0	
3	2019-08-17	2019	8	August	17	5	Saturday	1	
4	2019-01-06	2019	1	January	6	6	Sunday	1	

Working with Times

In [23]:

```
import datetime
```

```
today = datetime.datetime.today()

today
```

Out[23]:
```
datetime.datetime(2023, 1, 29, 23, 41, 23, 626953)
```

In [24]:
```
today - date['date']
```

Out[24]:
```
0     1146 days 23:41:23.626953
1     1628 days 23:41:23.626953
2     1559 days 23:41:23.626953
3     1261 days 23:41:23.626953
4     1484 days 23:41:23.626953
                 ...
995   1574 days 23:41:23.626953
996   1515 days 23:41:23.626953
997   1363 days 23:41:23.626953
998   1428 days 23:41:23.626953
999   1202 days 23:41:23.626953
Name: date, Length: 1000, dtype: timedelta64[ns]
```

In [25]:
```
(today - date['date']).dt.days
```

Out[25]:
```
0      1146
1      1628
2      1559
3      1261
4      1484
       ...
995    1574
996    1515
997    1363
998    1428
999    1202
Name: date, Length: 1000, dtype: int64
```

In [26]:
```
# Months passed

np.round((today -date['date']) / np.timedelta64(1, 'M'),0)
```

Out[26]:
```
0      38.0
1      54.0
2      51.0
3      41.0
4      49.0
       ...
995    52.0
996    50.0
997    45.0
998    47.0
999    40.0
Name: date, Length: 1000, dtype: float64
```

In [27]:
```
time.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    1000 non-null   object
```

```
 1   msg     1000 non-null   object
dtypes: object(2)
memory usage: 15.8+ KB
```

In [28]:
```
# Converting to datetime datatype
time['date'] = pd.to_datetime(time['date'])
```

In [29]:
```
time['hour'] = time['date'].dt.hour
time['min'] = time['date'].dt.minute
time['sec'] = time['date'].dt.second

time.head()
```

Out[29]:

	date	msg	hour	min	sec
0	2013-12-15 00:50:00	ищу на сегодня мужика 37	0	50	0
1	2014-04-29 23:40:00	ПАРЕНЬ БИ ИЩЕТ ДРУГА СЕЙЧАС!! СМС ММС 0955532826	23	40	0
2	2012-12-30 00:21:00	Днепр.м 43 позн.с д/ж *.o 067.16.34.576	0	21	0
3	2014-11-28 00:31:00	КИЕВ ИЩУ Д/Ж ДО 45 МНЕ СЕЙЧАС СКУЧНО 093 629 9...	0	31	0
4	2013-10-26 23:11:00	Зая я тебя никогда не обижу люблю тебя!) Даше	23	11	0

In [30]:
```
#Extract time part
time['time'] = time['date'].dt.time

time.head()
```

Out[30]:

	date	msg	hour	min	sec	time
0	2013-12-15 00:50:00	ищу на сегодня мужика 37	0	50	0	00:50:00
1	2014-04-29 23:40:00	ПАРЕНЬ БИ ИЩЕТ ДРУГА СЕЙЧАС!! СМС ММС 0955532826	23	40	0	23:40:00
2	2012-12-30 00:21:00	Днепр.м 43 позн.с д/ж *.o 067.16.34.576	0	21	0	00:21:00
3	2014-11-28 00:31:00	КИЕВ ИЩУ Д/Ж ДО 45 МНЕ СЕЙЧАС СКУЧНО 093 629 9...	0	31	0	00:31:00
4	2013-10-26 23:11:00	Зая я тебя никогда не обижу люблю тебя!) Даше	23	11	0	23:11:00

In [31]:
```
#Time diff.
today - time['date']
```

Out[31]:
```
0      3332 days 22:51:23.626953
1      3197 days 00:01:23.626953
2      3682 days 23:20:23.626953
3      2984 days 23:10:23.626953
4      3382 days 00:30:23.626953
                  ...
995    3971 days 22:51:23.626953
996    3293 days 00:27:23.626953
997    3758 days 00:04:23.626953
998    3874 days 00:07:23.626953
999    3146 days 00:16:23.626953
Name: date, Length: 1000, dtype: timedelta64[ns]
```

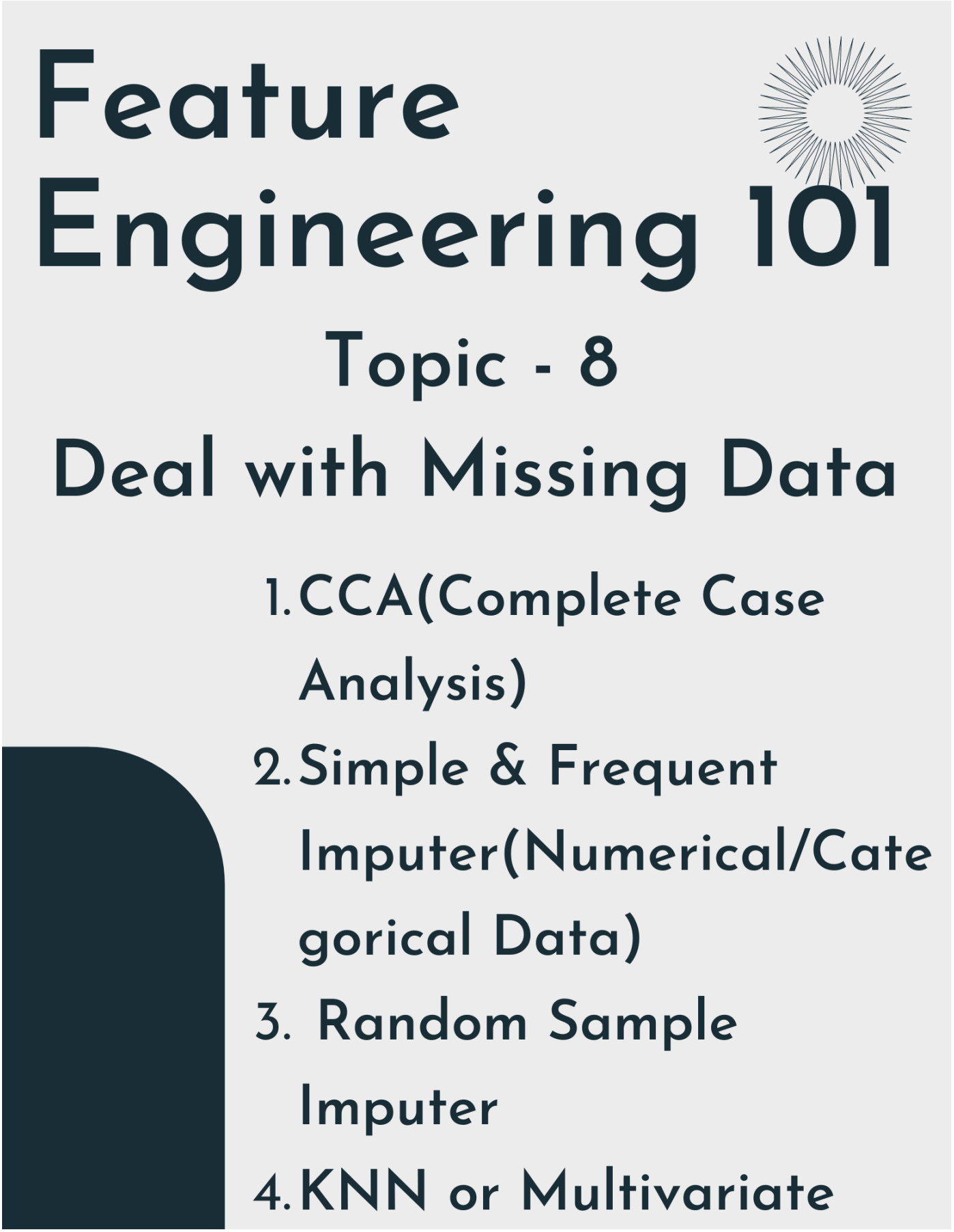
In [32]:
```
# in seconds

(today - time['date'])/np.timedelta64(1,'s')
```

Out[32]:
```
0      2.879671e+08
1      2.762209e+08
2      3.182088e+08
3      2.579010e+08
4      2.922066e+08
           ...
995    3.431767e+08
996    2.845168e+08
997    3.246915e+08
998    3.347140e+08
999    2.718154e+08
Name: date, Length: 1000, dtype: float64
```

In [33]:
```
# in hours

(today - time['date'])/np.timedelta64(1,'h')
```

Out[33]:
```
0      79990.856563
1      76728.023230
2      88391.339896
3      71639.173230
4      81168.506563
           ...
995    95326.856563
996    79032.456563
997    90192.073230
998    92976.123230
999    75504.273230
Name: date, Length: 1000, dtype: float64
```

In [ ]:
```

```



CCA (Complete Case Analysis)

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('data_science_job.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	enrollee_id	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	m
0	8949	city_103	0.920	Male	Has relevent experience	no_enrollment	Graduate	
1	29725	city_40	0.776	Male	No relevent experience	no_enrollment	Graduate	
2	11561	city_21	0.624	NaN	No relevent experience	Full time course	Graduate	
3	33241	city_115	0.789	NaN	No relevent experience	NaN	Graduate	B
4	666	city_162	0.767	Male	Has relevent experience	no_enrollment	Masters	

In [4]:
```
df.isnull().mean()*100
```

Out[4]:
```
enrollee_id              0.000000
city                     0.000000
city_development_index   2.500261
gender                  23.530640
relevent_experience      0.000000
enrolled_university      2.014824
education_level          2.401086
major_discipline        14.683161
experience               0.339284
company_size            30.994885
company_type            32.049274
training_hours           3.998330
target                   0.000000
dtype: float64
```

In [5]:
```
df.shape
```

Out[5]:
```
(19158, 13)
```

In [6]:
```
cols = [var for var in df.columns if df[var].isnull().mean() < 0.05 and df[var].isnull().m
cols
```

Out[6]:
```
['city_development_index',
 'enrolled_university',
 'education_level',
 'experience',
 'training_hours']
```

In [7]:
```
df[cols].sample(5)
```

Out[7]:

	city_development_index	enrolled_university	education_level	experience	training_hours
2179	0.899	no_enrollment	Graduate	8.0	54.0
11724	0.884	no_enrollment	Graduate	20.0	NaN

	city_development_index	enrolled_university	education_level	experience	training_hours
7189	0.897	no_enrollment	Masters	16.0	156.0
748	0.920	no_enrollment	Graduate	20.0	55.0
13129	0.920	no_enrollment	Graduate	20.0	3.0

In [8]:
```
df['education_level'].value_counts()
```

Out[8]:
```
Graduate          11598
Masters            4361
High School        2017
Phd                 414
Primary School      308
Name: education_level, dtype: int64
```

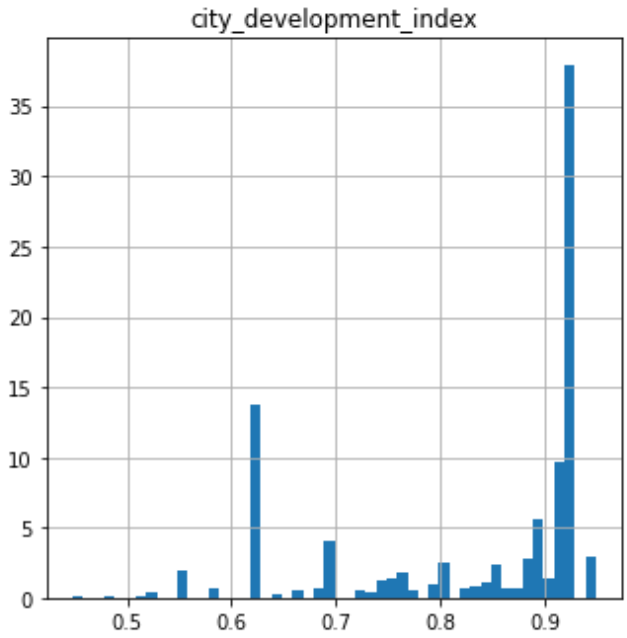
In [9]:
```
len(df[cols].dropna()) / len(df)
```

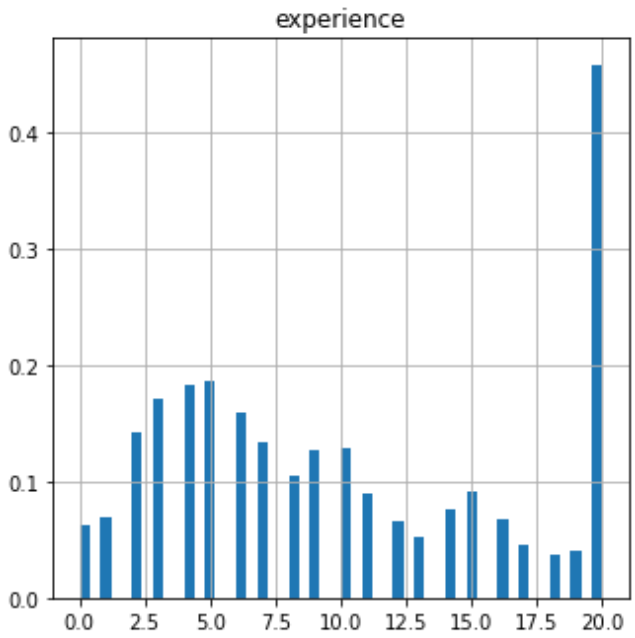
Out[9]:
```
0.8968577095730244
```

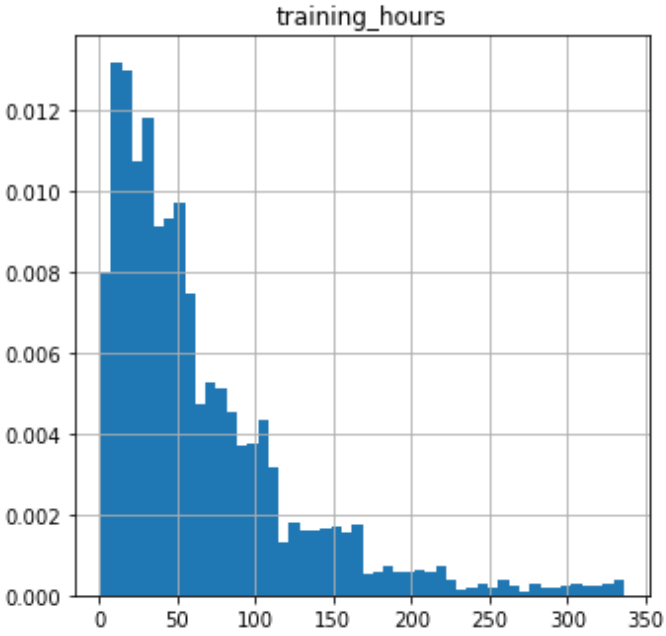
In [10]:
```
new_df = df[cols].dropna()
df.shape, new_df.shape
```

Out[10]:
```
((19158, 13), (17182, 5))
```

In [11]:
```
new_df.hist(bins=50, density=True, figsize=(12, 12))
plt.show()
```

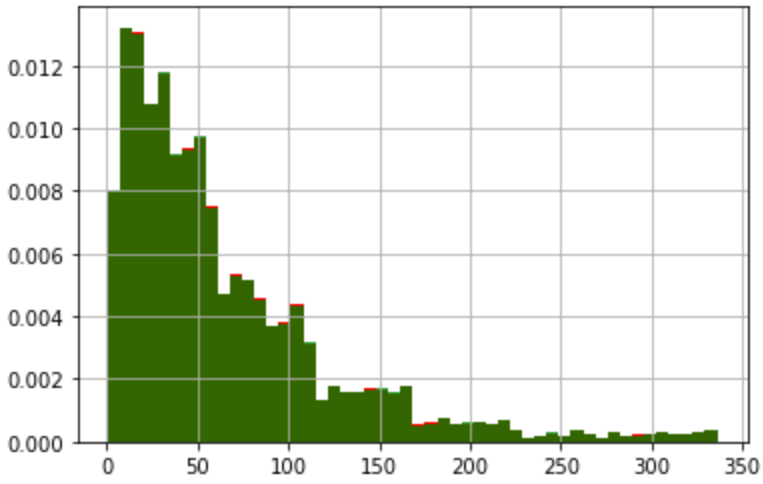






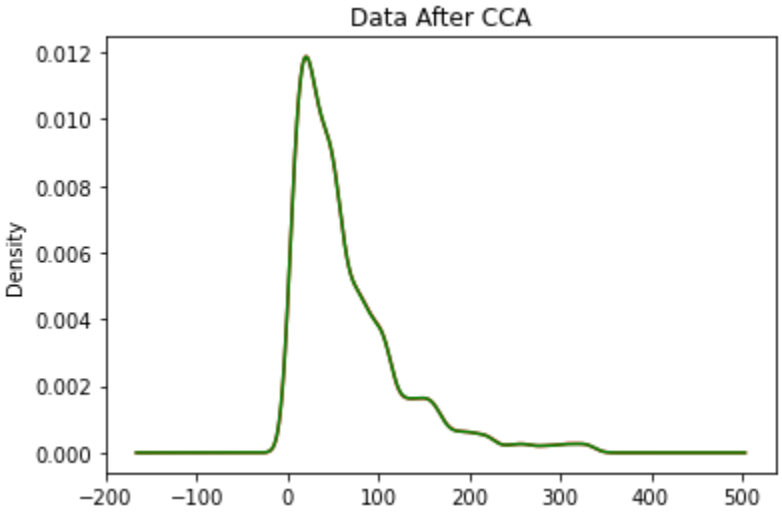
In [12]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
df['training_hours'].hist(bins=50, ax=ax, density=True, color='red')

# data after cca, the argument alpha makes the color transparent, so we can
# see the overlay of the 2 distributions
new_df['training_hours'].hist(bins=50, ax=ax, color='green', density=True, alpha=0.8)
plt.show()
```



In [13]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
df['training_hours'].plot.density(color='red')

# Data After CCA
new_df['training_hours'].plot.density(color='green')
plt.title('Data After CCA')
plt.show()
```

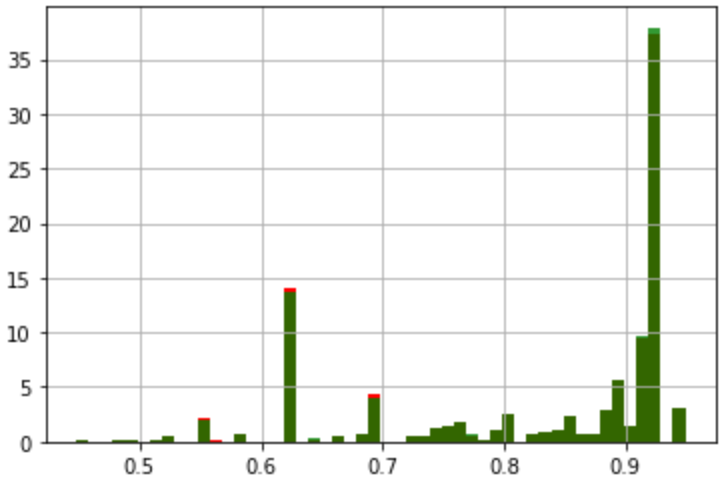


In [14]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
df['city_development_index'].hist(bins=50, ax=ax, density=True, color='red')

# data after cca, the argument alpha makes the color transparent, so we can
# see the overlay of the 2 distributions
new_df['city_development_index'].hist(bins=50, ax=ax, color='green', density=True, alpha=0
```

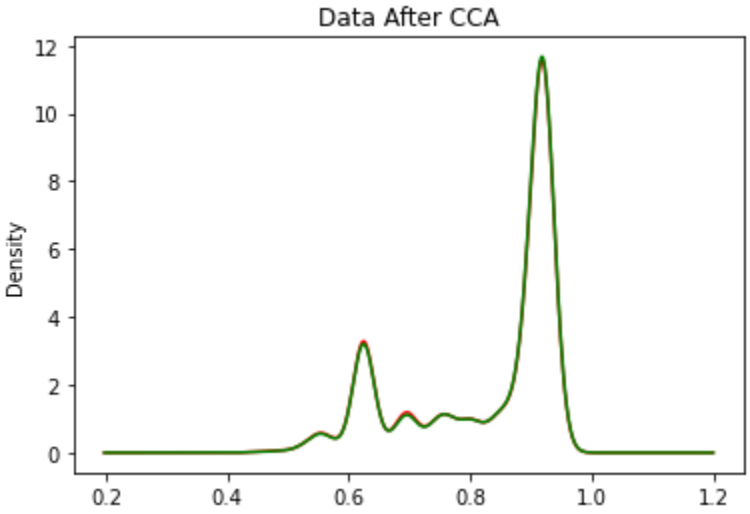
Out[14]:

```
<AxesSubplot:>
```



In [15]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
df['city_development_index'].plot.density(color='red')

# data after cca
new_df['city_development_index'].plot.density(color='green')
plt.title('Data After CCA')
plt.show()
```

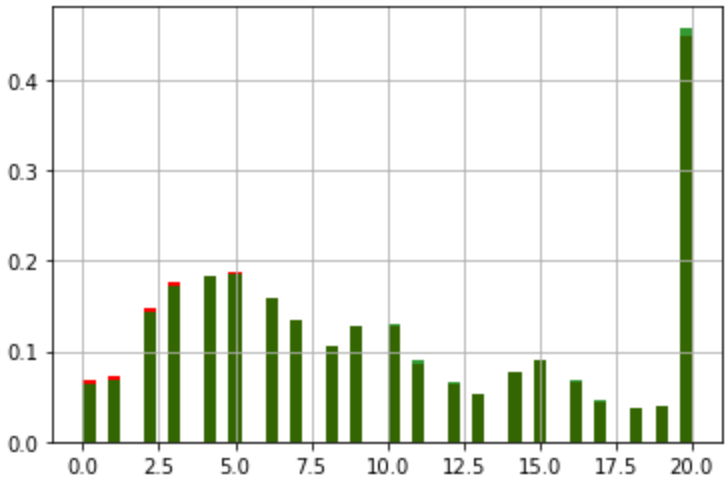


In [16]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
df['experience'].hist(bins=50, ax=ax, density=True, color='red')

# data after cca, the argument alpha makes the color transparent, so we can
# see the overlay of the 2 distributions
new_df['experience'].hist(bins=50, ax=ax, color='green', density=True, alpha=0.8)
```

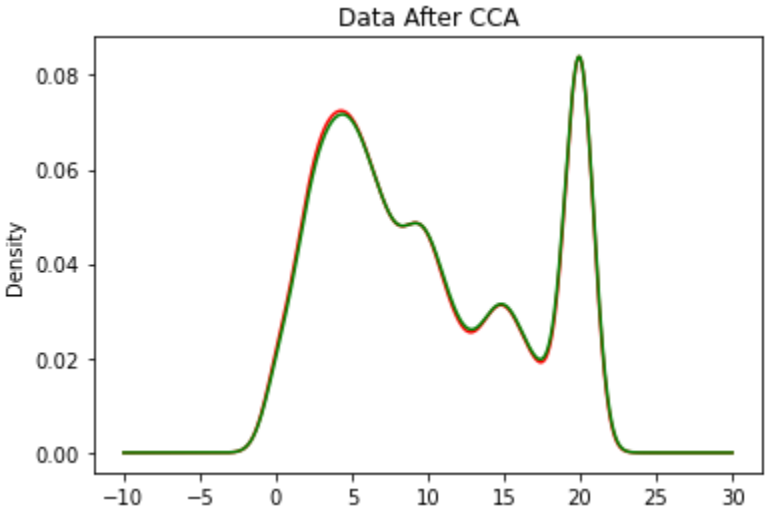
Out[16]:

```
<AxesSubplot:>
```



In [17]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
df['experience'].plot.density(color='red')

# data after cca
new_df['experience'].plot.density(color='green')
plt.title('Data After CCA')
plt.show()
```



In [18]:

```
temp = pd.concat([
            # percentage of observations per category, original data
            df['enrolled_university'].value_counts() / len(df),

            # percentage of observations per category, cca data
            new_df['enrolled_university'].value_counts() / len(new_df)
        ],
        axis=1)

# add column names
temp.columns = ['original', 'cca']

temp
```

Out[18]:

	original	cca
no_enrollment	0.721213	0.735188

	original	cca
Full time course	0.196106	0.200733
Part time course	0.062533	0.064079

In [19]:

```
temp = pd.concat([
            # percentage of observations per category, original data
            df['education_level'].value_counts() / len(df),

            # percentage of observations per category, cca data
            new_df['education_level'].value_counts() / len(new_df)
        ],
        axis=1)

# add column names
temp.columns = ['original', 'cca']

temp
```

Out[19]:

	original	cca
Graduate	0.605387	0.619835
Masters	0.227633	0.234082
High School	0.105282	0.107380
Phd	0.021610	0.022116
Primary School	0.016077	0.016587

In [ ]:

Handling Missing Categorical Data (frequent-value-imputation)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('train1.csv',usecols=['GarageQual','FireplaceQu','SalePrice'])
```

In [3]:
```
df.head()
```

Out[3]:

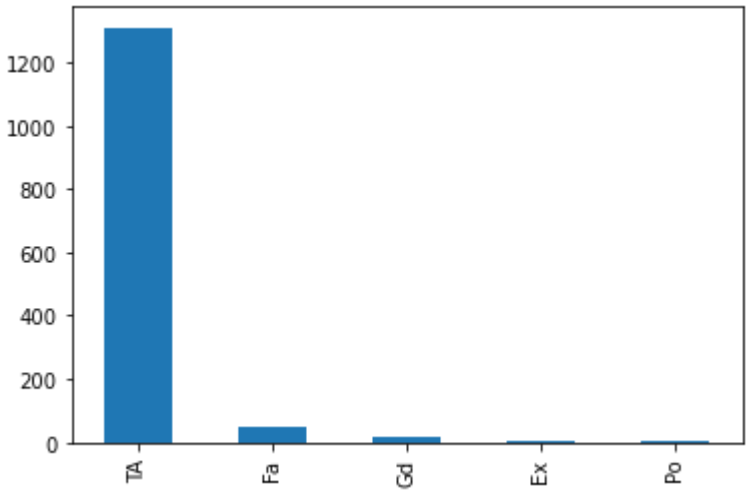
	FireplaceQu	GarageQual	SalePrice
0	NaN	TA	208500
1	TA	TA	181500
2	TA	TA	223500
3	Gd	TA	140000
4	TA	TA	250000

In [4]:
```
df.isnull().mean()*100
```

Out[4]:
```
FireplaceQu    47.260274
GarageQual      5.547945
SalePrice       0.000000
dtype: float64
```

In [5]:
```
df['GarageQual'].value_counts().plot(kind='bar')
```

Out[5]:
```
<AxesSubplot:>
```

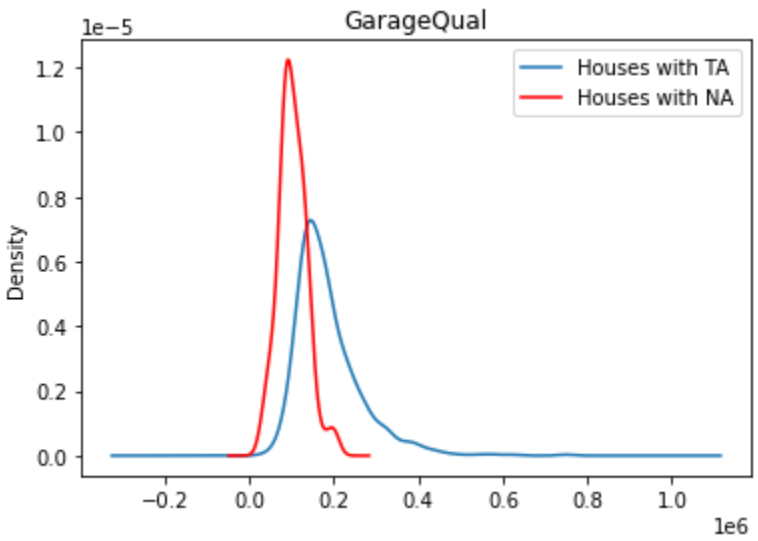


In [6]:
```
df['GarageQual'].mode()
```

Out[6]:
```
0    TA
dtype: object
```

In [7]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

df[df['GarageQual']=='TA']['SalePrice'].plot(kind='kde', ax=ax)

df[df['GarageQual'].isnull()]['SalePrice'].plot(kind='kde', ax=ax, color='red')

lines, labels = ax.get_legend_handles_labels()
labels = ['Houses with TA', 'Houses with NA']
ax.legend(lines, labels, loc='best')

plt.title('GarageQual')
```

Out[7]:
```
Text(0.5, 1.0, 'GarageQual')
```

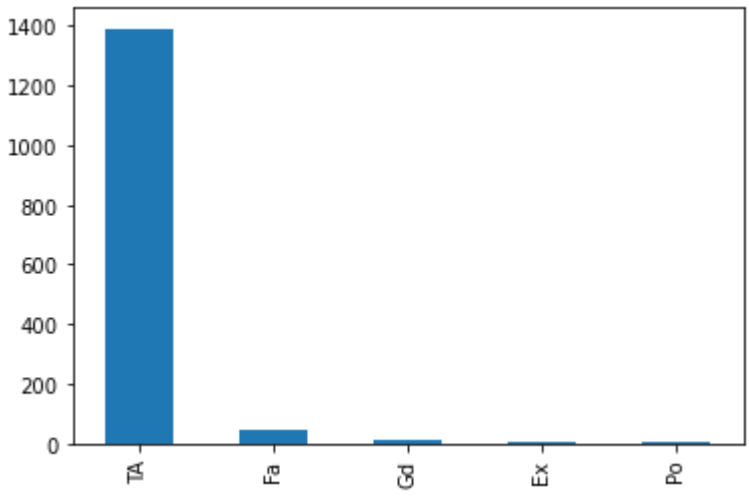


In [8]:
```
temp = df[df['GarageQual']=='TA']['SalePrice']
```

In [9]:
```
df['GarageQual'].fillna('TA', inplace=True)
```

In [10]:
```
df['GarageQual'].value_counts().plot(kind='bar')
```

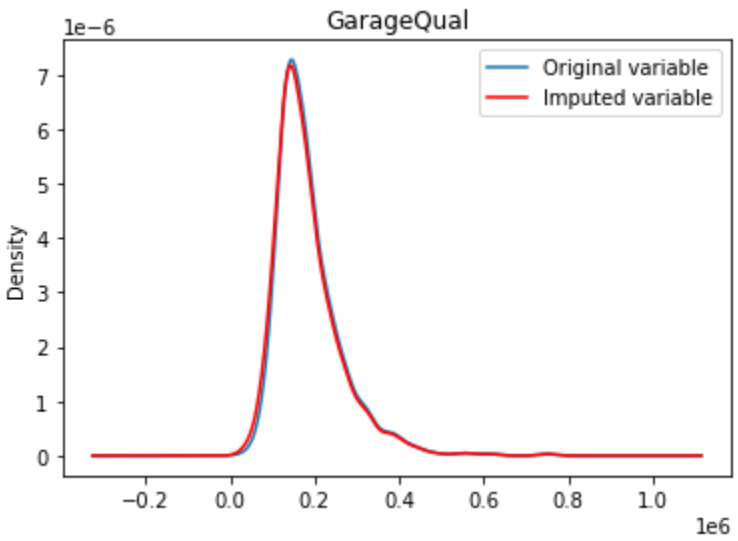
Out[10]:
```
<AxesSubplot:>
```



In [11]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
```

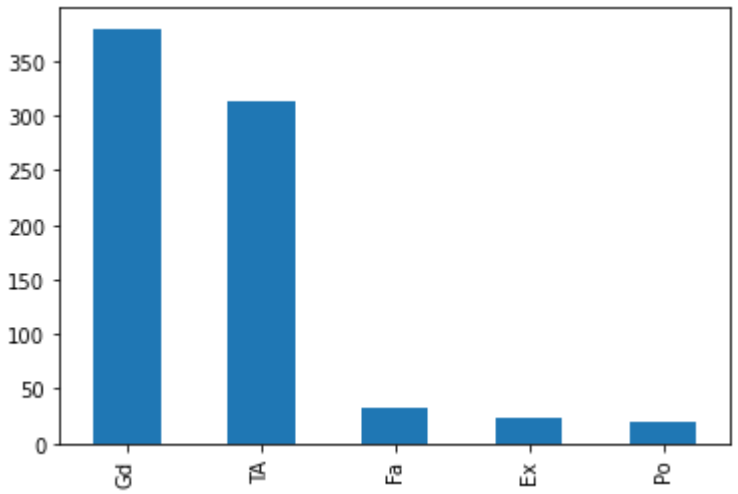
```
temp.plot(kind='kde', ax=ax)

# distribution of the variable after imputation
df[df['GarageQual'] == 'TA']['SalePrice'].plot(kind='kde', ax=ax, color='red')

lines, labels = ax.get_legend_handles_labels()
labels = ['Original variable', 'Imputed variable']
ax.legend(lines, labels, loc='best')

# add title
plt.title('GarageQual')
```

Out[11]:
```
Text(0.5, 1.0, 'GarageQual')
```



In [12]:
```
df['FireplaceQu'].value_counts().plot(kind='bar')
```

Out[12]:
```
<AxesSubplot:>
```



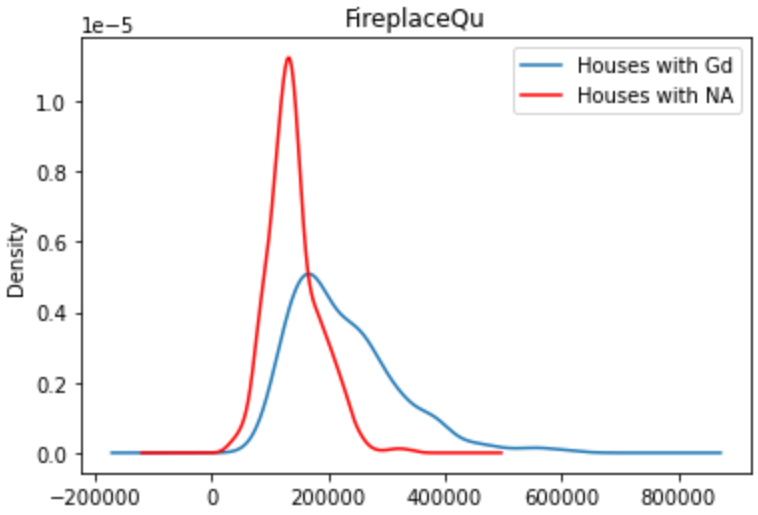
In [13]:
```
df['FireplaceQu'].mode()
```

Out[13]:
```
0    Gd
dtype: object
```

In [14]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

df[df['FireplaceQu']=='Gd']['SalePrice'].plot(kind='kde', ax=ax)
```

```
df[df['FireplaceQu'].isnull()]['SalePrice'].plot(kind='kde', ax=ax, color='red')

lines, labels = ax.get_legend_handles_labels()
labels = ['Houses with Gd', 'Houses with NA']
ax.legend(lines, labels, loc='best')

plt.title('FireplaceQu')
```

Out[14]:
```
Text(0.5, 1.0, 'FireplaceQu')
```

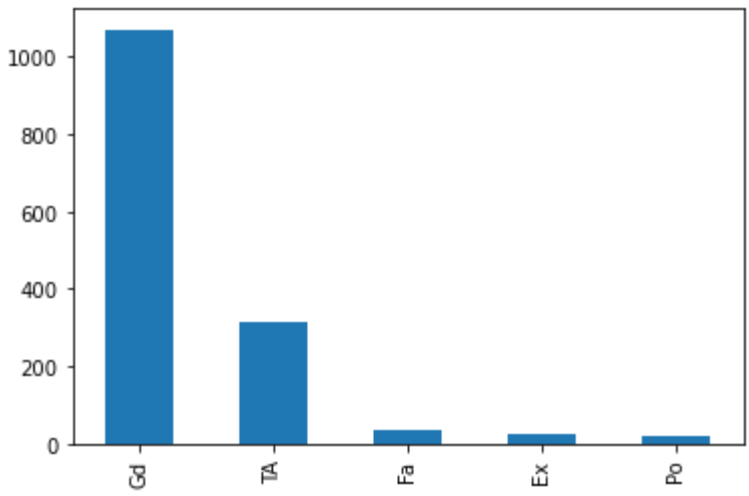


In [15]:
```
temp = df[df['FireplaceQu']=='Gd']['SalePrice']
```

In [16]:
```
df['FireplaceQu'].fillna('Gd', inplace=True)
```

In [17]:
```
df['FireplaceQu'].value_counts().plot(kind='bar')
```

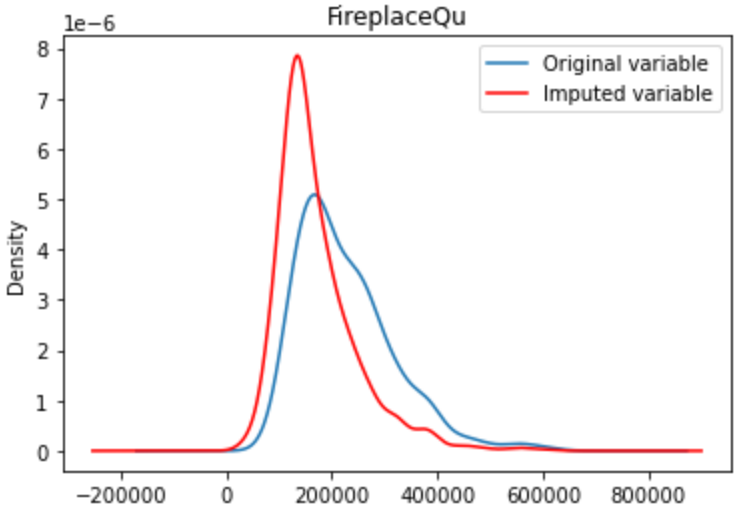
Out[17]:
```
<AxesSubplot:>
```



In [18]:
```
fig = plt.figure()
ax = fig.add_subplot(111)


temp.plot(kind='kde', ax=ax)

# distribution of the variable after imputation
df[df['FireplaceQu'] == 'Gd']['SalePrice'].plot(kind='kde', ax=ax, color='red')
```

```
lines, labels = ax.get_legend_handles_labels()
labels = ['Original variable', 'Imputed variable']
ax.legend(lines, labels, loc='best')

# add title
plt.title('FireplaceQu')
```

Out[18]:
```
Text(0.5, 1.0, 'FireplaceQu')
```



In [19]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(df.drop(columns=['SalePrice']),df['SalePr
```

In [20]:
```
from sklearn.impute import SimpleImputer
```

In [21]:
```
imputer = SimpleImputer(strategy='most_frequent')
```

In [22]:
```
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_train)
```

In [23]:
```
imputer.statistics_
```

Out[23]:
```
array(['Gd', 'TA'], dtype=object)
```

In [ ]:

Missing Indicator (automatically-select-imputer-parameters)

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

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
df = pd.read_csv('train.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [4]:
```
df.drop(columns=['PassengerId','Name','Ticket','Cabin'],inplace=True)
```

In [5]:
```
#devide the columns
X = df.drop(columns=['Survived'])
y = df['Survived']
```

In [6]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [7]:
```
X_train.head()
```

Out[7]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
30	1	male	40.0	0	0	27.7208	C
10	3	female	4.0	1	1	16.7000	S
873	3	male	47.0	0	0	9.0000	S
182	3	male	9.0	4	2	31.3875	S
876	3	male	20.0	0	0	9.8458	S

In [8]:
```
y_train.head()
```

Out[8]:
```
30     0
10     1
873    0
182    0
876    0
Name: Survived, dtype: int64
```

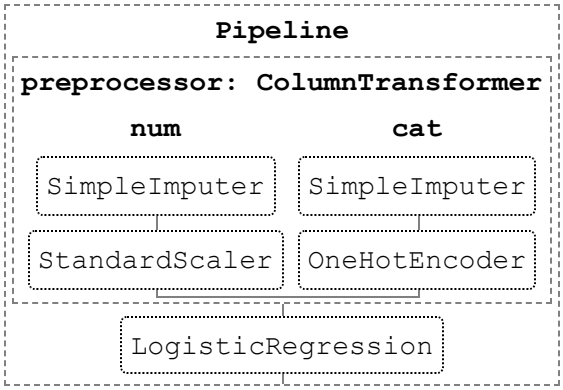
In [9]:
```
numerical_features = ['Age', 'Fare']
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_features = ['Embarked', 'Sex']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe',OneHotEncoder(handle_unknown='ignore'))
])
```

In [10]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)
```

In [11]:
```
clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])
```

In [12]:
```
from sklearn import set_config

set_config(display='diagram')
clf
```

Out[12]:



In [13]:
```
param_grid = {
    'preprocessor__num__imputer__strategy': ['mean', 'median'],
    'preprocessor__cat__imputer__strategy': ['most_frequent', 'constant'],
    'classifier__C': [0.1, 1.0, 10, 100]
}

grid_search = GridSearchCV(clf, param_grid, cv=10)
```

In [14]:
```
grid_search.fit(X_train, y_train)

print(f"Best params:")
print(grid_search.best_params_)
```

```
Best params:
{'classifier__C': 1.0, 'preprocessor__cat__imputer__strategy': 'most_frequent', 'preproces
sor__num__imputer__strategy': 'mean'}
```

In [15]:
```
print(f"Internal CV score: {grid_search.best_score_:.3f}")
```

```
Internal CV score: 0.788
```

In [16]:
```
import pandas as pd

cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results = cv_results.sort_values("mean_test_score", ascending=False)
cv_results[['param_classifier__C','param_preprocessor__cat__imputer__strategy','param_prep
```

Out[16]:

	param_classifier__C	param_preprocessor__cat__imputer__strategy	param_preprocessor__num__imputer__strategy	mea
4	1.0	most_frequent	mean	
5	1.0	most_frequent	median	
6	1.0	constant	mean	
7	1.0	constant	median	
8	10	most_frequent	mean	
9	10	most_frequent	median	
10	10	constant	mean	
11	10	constant	median	
12	100	most_frequent	mean	
13	100	most_frequent	median	
14	100	constant	mean	

	param_classifier__C	param_preprocessor__cat__imputer__strategy	param_preprocessor__num__imputer__strategy	mea
15	100	constant	median	
0	0.1	most_frequent	mean	
1	0.1	most_frequent	median	
2	0.1	constant	mean	
3	0.1	constant	median	

In [ ]:

Imputing Numerical Data (Man Median Imputation)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
```

In [3]:
```
df = pd.read_csv('titanic_toy.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	Age	Fare	Family	Survived
0	22.0	7.2500	1	0
1	38.0	71.2833	1	1
2	26.0	7.9250	0	1
3	35.0	53.1000	1	1
4	35.0	8.0500	0	0

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Age       714 non-null    float64
 1   Fare      846 non-null    float64
 2   Family    891 non-null    int64  
 3   Survived  891 non-null    int64  
dtypes: float64(2), int64(2)
memory usage: 28.0 KB
```

In [6]:
```
df.isnull().mean()
```

Out[6]:
```
Age         0.198653
Fare        0.050505
Family      0.000000
Survived    0.000000
dtype: float64
```

In [7]:
```
X = df.drop(columns=['Survived'])
y = df['Survived']
```

In [8]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [9]:
```
X_train.shape, X_test.shape
```

Out[9]:
```
((712, 3), (179, 3))
```

In [10]:
```
X_train.isnull().mean()
```

Out[10]:
```
Age       0.207865
Fare      0.050562
Family    0.000000
dtype: float64
```

In [11]:
```
mean_age = X_train['Age'].mean()
median_age = X_train['Age'].median()

mean_fare = X_train['Fare'].mean()
median_fare = X_train['Fare'].median()
```

In [12]:
```
X_train['Age_median'] = X_train['Age'].fillna(median_age)
X_train['Age_mean'] = X_train['Age'].fillna(mean_age)

X_train['Fare_median'] = X_train['Fare'].fillna(median_fare)
X_train['Fare_mean'] = X_train['Fare'].fillna(mean_fare)
```

```
C:\Users\HP\AppData\Local\Temp/ipykernel_69128/2444989457.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['Age_median'] = X_train['Age'].fillna(median_age)
C:\Users\HP\AppData\Local\Temp/ipykernel_69128/2444989457.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['Age_mean'] = X_train['Age'].fillna(mean_age)
C:\Users\HP\AppData\Local\Temp/ipykernel_69128/2444989457.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['Fare_median'] = X_train['Fare'].fillna(median_fare)
C:\Users\HP\AppData\Local\Temp/ipykernel_69128/2444989457.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['Fare_mean'] = X_train['Fare'].fillna(mean_fare)
```

In [13]:
```
X_train.sample(5)
```

Out[13]:

	Age	Fare	Family	Age_median	Age_mean	Fare_median	Fare_mean
154	NaN	7.3125	0	28.75	29.785904	7.3125	7.3125
580	25.0	30.0000	2	25.00	25.000000	30.0000	30.0000
747	30.0	13.0000	0	30.00	30.000000	13.0000	13.0000

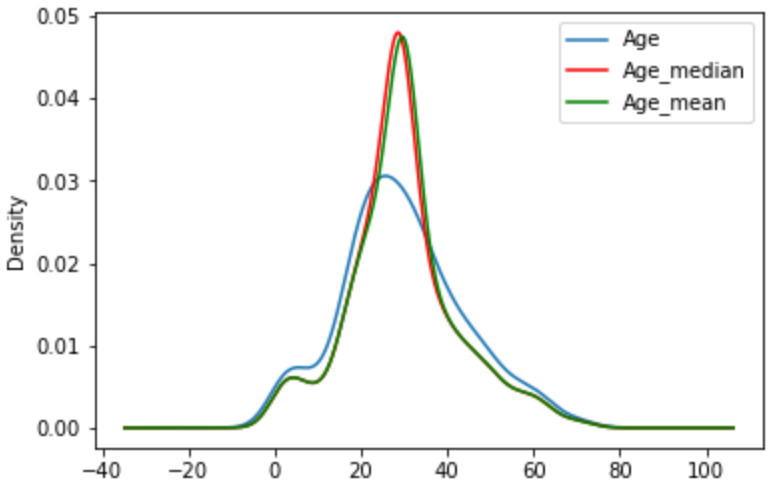
	Age	Fare	Family	Age_median	Age_mean	Fare_median	Fare_mean
292	36.0	12.8750	0	36.00	36.000000	12.8750	12.8750
435	14.0	120.0000	3	14.00	14.000000	120.0000	120.0000

In [14]:
```
print('Original Age variable variance: ', X_train['Age'].var())
print('Age Variance after median imputation: ', X_train['Age_median'].var())
print('Age Variance after mean imputation: ', X_train['Age_mean'].var())

print('Original Fare variable variance: ', X_train['Fare'].var())
print('Fare Variance after median imputation: ', X_train['Fare_median'].var())
print('Fare Variance after mean imputation: ', X_train['Fare_mean'].var())
```

```
Original Age variable variance:  204.3495133904614
Age Variance after median imputation:  161.9895663346054
Age Variance after mean imputation:  161.81262452718673
Original Fare variable variance:  2448.197913706318
Fare Variance after median imputation:  2340.0910219753637
Fare Variance after mean imputation:  2324.2385256705547
```

In [15]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

# original variable distribution
X_train['Age'].plot(kind='kde', ax=ax)

# variable imputed with the median
X_train['Age_median'].plot(kind='kde', ax=ax, color='red')

# variable imputed with the mean
X_train['Age_mean'].plot(kind='kde', ax=ax, color='green')

# add legends
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

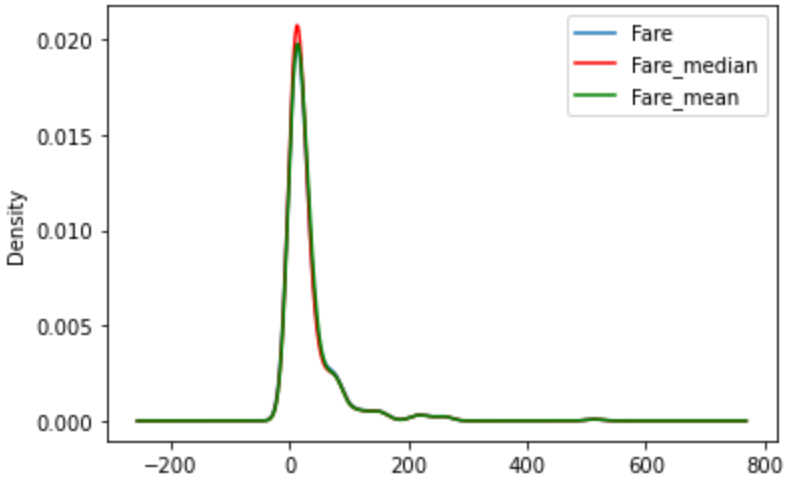
Out[15]:
```
<matplotlib.legend.Legend at 0x281a01a5850>
```



In [16]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

# original variable distribution
X_train['Fare'].plot(kind='kde', ax=ax)

# variable imputed with the median
```

```
X_train['Fare_median'].plot(kind='kde', ax=ax, color='red')

# variable imputed with the mean
X_train['Fare_mean'].plot(kind='kde', ax=ax, color='green')

# add legends
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[16]:
```
<matplotlib.legend.Legend at 0x281a09eff10>
```



In [17]:
```
X_train.cov()
```

Out[17]:

	Age	Fare	Family	Age_median	Age_mean	Fare_median	Fare_mean
Age	204.349513	70.719262	-6.498901	204.349513	204.349513	64.858859	66.665205
Fare	70.719262	2448.197914	17.258917	57.957599	55.603719	2448.197914	2448.197914
Family	-6.498901	17.258917	2.735252	-5.112563	-5.146106	16.476305	16.385048
Age_median	204.349513	57.957599	-5.112563	161.989566	161.812625	53.553455	55.023037
Age_mean	204.349513	55.603719	-5.146106	161.812625	161.812625	51.358000	52.788341
Fare_median	64.858859	2448.197914	16.476305	53.553455	51.358000	2340.091022	2324.238526
Fare_mean	66.665205	2448.197914	16.385048	55.023037	52.788341	2324.238526	2324.238526

In [18]:
```
X_train.corr()
```

Out[18]:

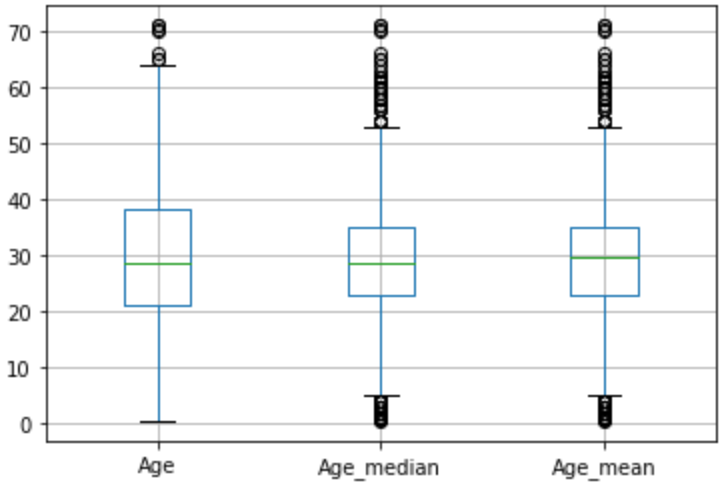
	Age	Fare	Family	Age_median	Age_mean	Fare_median	Fare_mean
Age	1.000000	0.092644	-0.299113	1.000000	1.000000	0.087356	0.090156
Fare	0.092644	1.000000	0.208268	0.091757	0.088069	1.000000	1.000000
Family	-0.299113	0.208268	1.000000	-0.242883	-0.244610	0.205942	0.205499
Age_median	1.000000	0.091757	-0.242883	1.000000	0.999454	0.086982	0.089673
Age_mean	1.000000	0.088069	-0.244610	0.999454	1.000000	0.083461	0.086078
Fare_median	0.087356	1.000000	0.205942	0.086982	0.083461	1.000000	0.996607
Fare_mean	0.090156	1.000000	0.205499	0.089673	0.086078	0.996607	1.000000

In [19]:

```
X_train[['Age', 'Age_median', 'Age_mean']].boxplot()
```

Out[19]:

```
<AxesSubplot:>
```

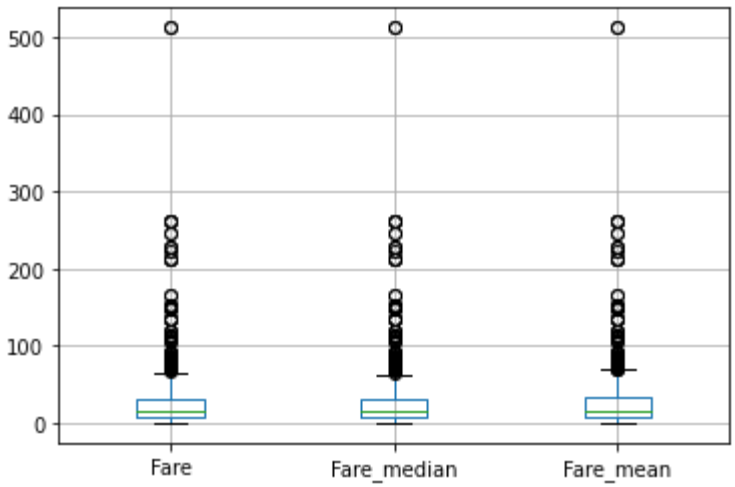


In [20]:

```
X_train[['Fare', 'Fare_median', 'Fare_mean']].boxplot()
```

Out[20]:

```
<AxesSubplot:>
```



Using Sklearn

In [21]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [22]:

```
imputer1 = SimpleImputer(strategy='median')
imputer2 = SimpleImputer(strategy='mean')
```

In [23]:

```
trf = ColumnTransformer([
    ('imputer1',imputer1,['Age']),
    ('imputer2',imputer2,['Fare'])
],remainder='passthrough')
```

In [24]:

```
trf.fit(X_train)
```

Out[24]:

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('imputer1', SimpleImputer(strategy='median'),
                                 ['Age']),
                                ('imputer2', SimpleImputer(), ['Fare'])])
```

In [25]:
```
trf.named_transformers_['imputer1'].statistics_
```

Out[25]:
```
array([28.75])
```

In [26]:
```
trf.named_transformers_['imputer2'].statistics_
```

Out[26]:
```
array([32.61759689])
```

In [27]:
```
X_train = trf.transform(X_train)
X_test = trf.transform(X_test)
```

In [28]:
```
X_train
```

Out[28]:
```
array([[ 40.    ,  27.7208,   0.    ],
       [  4.    ,  16.7   ,   2.    ],
       [ 47.    ,   9.    ,   0.    ],
       ...,
       [ 71.    ,  49.5042,   0.    ],
       [ 28.75  , 221.7792,   0.    ],
       [ 28.75  ,  25.925 ,   0.    ]])
```

In [ ]:
```

```

Imputing Numerical Data (Arbitrary-Value-Imputation)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
```

In [3]:
```
df = pd.read_csv('titanic_toy.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	Age	Fare	Family	Survived
0	22.0	7.2500	1	0
1	38.0	71.2833	1	1
2	26.0	7.9250	0	1
3	35.0	53.1000	1	1
4	35.0	8.0500	0	0

In [5]:
```
df.isnull().mean()
```

Out[5]:
```
Age         0.198653
Fare        0.050505
Family      0.000000
Survived    0.000000
dtype: float64
```

In [6]:
```
X = df.drop(columns=['Survived'])
y = df['Survived']
```

In [7]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [8]:
```
X_train['Age_99'] = X_train['Age'].fillna(99)
X_train['Age_minus1'] = X_train['Age'].fillna(-1)

X_train['Fare_999'] = X_train['Fare'].fillna(999)
X_train['Fare_minus1'] = X_train['Fare'].fillna(-1)
```

```
C:\Users\HP\AppData\Local\Temp/ipykernel_79272/3652012184.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  X_train['Age_99'] = X_train['Age'].fillna(99)
C:\Users\HP\AppData\Local\Temp/ipykernel_79272/3652012184.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_train['Age_minus1'] = X_train['Age'].fillna(-1)
C:\Users\HP\AppData\Local\Temp/ipykernel_79272/3652012184.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_train['Fare_999'] = X_train['Fare'].fillna(999)
C:\Users\HP\AppData\Local\Temp/ipykernel_79272/3652012184.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_train['Fare_minus1'] = X_train['Fare'].fillna(-1)
```

In [9]:

```
print('Original Age variable variance: ', X_train['Age'].var())
print('Age Variance after 99 wala imputation: ', X_train['Age_99'].var())
print('Age Variance after -1 wala imputation: ', X_train['Age_minus1'].var())

print('Original Fare variable variance: ', X_train['Fare'].var())
print('Fare Variance after 999 wala imputation: ', X_train['Fare_999'].var())
print('Fare Variance after -1 wala imputation: ', X_train['Fare_minus1'].var())
```

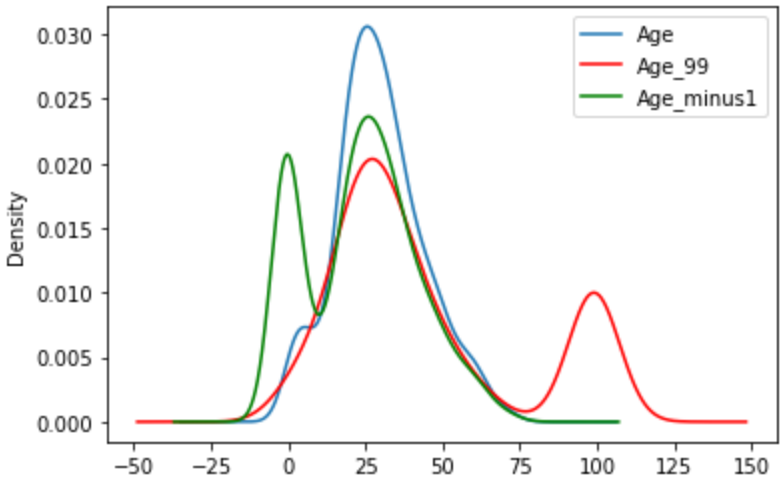
```
Original Age variable variance:  204.3495133904614
Age Variance after 99 wala imputation:  951.7275570187172
Age Variance after -1 wala imputation:  318.0896202624484
Original Fare variable variance:  2448.197913706318
Fare Variance after 999 wala imputation:  47219.20265217623
Fare Variance after -1 wala imputation:  2378.5676784883503
```

In [10]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original variable distribution
X_train['Age'].plot(kind='kde', ax=ax)

# variable imputed with the median
X_train['Age_99'].plot(kind='kde', ax=ax, color='red')

# variable imputed with the mean
X_train['Age_minus1'].plot(kind='kde', ax=ax, color='green')

# add legends
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[10]:

```
<matplotlib.legend.Legend at 0x28930f0acd0>
```

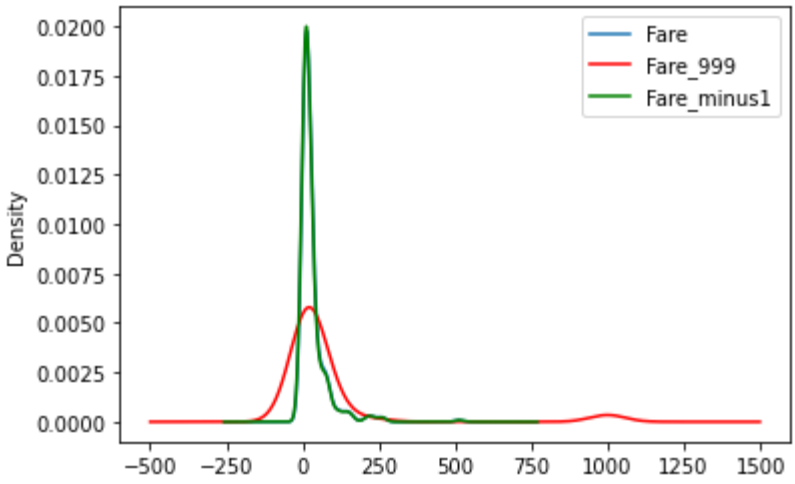


In [11]:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original variable distribution
X_train['Fare'].plot(kind='kde', ax=ax)

# variable imputed with the median
X_train['Fare_999'].plot(kind='kde', ax=ax, color='red')

# variable imputed with the mean
X_train['Fare_minus1'].plot(kind='kde', ax=ax, color='green')

# add legends
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[11]:

```
<matplotlib.legend.Legend at 0x28931765760>
```



In [12]:

```
X_train.cov()
```

Out[12]:

	Age	Fare	Family	Age_99	Age_minus1	Fare_999	Fare_minus1
Age	204.349513	70.719262	-6.498901	204.349513	204.349513	162.793430	63.321188
Fare	70.719262	2448.197914	17.258917	-101.671097	125.558364	2448.197914	2448.197914
Family	-6.498901	17.258917	2.735252	-7.387287	-4.149246	11.528625	16.553989
Age_99	204.349513	-101.671097	-7.387287	951.727557	-189.535540	-159.931663	-94.317400
Age_minus1	204.349513	125.558364	-4.149246	-189.535540	318.089620	257.379887	114.394141

	Age	Fare	Family	Age_99	Age_minus1	Fare_999	Fare_minus1
Fare_999	162.793430	2448.197914	11.528625	-159.931663	257.379887	47219.202652	762.474982
Fare_minus1	63.321188	2448.197914	16.553989	-94.317400	114.394141	762.474982	2378.567678

In [13]:
```
X_train.corr()
```

Out[13]:

	Age	Fare	Family	Age_99	Age_minus1	Fare_999	Fare_minus1
Age	1.000000	0.092644	-0.299113	1.000000	1.000000	0.051179	0.084585
Fare	0.092644	1.000000	0.208268	-0.066273	0.142022	1.000000	1.000000
Family	-0.299113	0.208268	1.000000	-0.144787	-0.140668	0.032079	0.205233
Age_99	1.000000	-0.066273	-0.144787	1.000000	-0.344476	-0.023857	-0.062687
Age_minus1	1.000000	0.142022	-0.140668	-0.344476	1.000000	0.066411	0.131514
Fare_999	0.051179	1.000000	0.032079	-0.023857	0.066411	1.000000	0.071946
Fare_minus1	0.084585	1.000000	0.205233	-0.062687	0.131514	0.071946	1.000000

Using Sklearn

In [14]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [15]:
```
imputer1 = SimpleImputer(strategy='constant',fill_value=99)
imputer2 = SimpleImputer(strategy='constant',fill_value=999)
```

In [16]:
```
trf = ColumnTransformer([
    ('imputer1',imputer1,['Age']),
    ('imputer2',imputer2,['Fare'])
],remainder='passthrough')
```

In [17]:
```
trf.fit(X_train)
```

Out[17]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('imputer1',
                                 SimpleImputer(fill_value=99,
                                               strategy='constant'),
                                 ['Age']),
                                ('imputer2',
                                 SimpleImputer(fill_value=999,
                                               strategy='constant'),
                                 ['Fare'])])
```

In [18]:
```
trf.named_transformers_['imputer1'].statistics_
```

Out[18]:
```
array([99.])
```

In [19]:
```
trf.named_transformers_['imputer2'].statistics_
```

Out[19]:
```
array([999.])
```

In [20]:
```
X_train = trf.transform(X_train)
X_test = trf.transform(X_test)
```

In [21]:
```
X_train
```

Out[21]:
```
array([[ 40.    ,  27.7208,   0.    ],
       [  4.    ,  16.7   ,   2.    ],
       [ 47.    ,   9.    ,   0.    ],
       ...,
       [ 71.    ,  49.5042,   0.    ],
       [ 99.    , 221.7792,   0.    ],
       [ 99.    ,  25.925 ,   0.    ]])
```

In [ ]:
```

```

Random Sample Imputation

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

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('train.csv',usecols=['Age','Fare','Survived'])
```

In [3]:
```
df.head()
```

Out[3]:

	Survived	Age	Fare
0	0	22.0	7.2500
1	1	38.0	71.2833
2	1	26.0	7.9250
3	1	35.0	53.1000
4	0	35.0	8.0500

In [4]:
```
df.isnull().mean() * 100
```

Out[4]:
```
Survived     0.00000
Age         19.86532
Fare         0.00000
dtype: float64
```

In [5]:
```
X = df.drop(columns=['Survived'])
y = df['Survived']
```

In [6]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [7]:
```
X_train
```

Out[7]:

	Age	Fare
30	40.0	27.7208
10	4.0	16.7000
873	47.0	9.0000
182	9.0	31.3875
876	20.0	9.8458
...	...	...
534	30.0	8.6625

	Age	Fare
584	NaN	8.7125
493	71.0	49.5042
527	NaN	221.7792
168	NaN	25.9250

712 rows × 2 columns

In [8]:
```
X_train['Age_imputed'] = X_train['Age']
X_test['Age_imputed'] = X_test['Age']
```

```
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/1230362693.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_train['Age_imputed'] = X_train['Age']
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/1230362693.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_test['Age_imputed'] = X_test['Age']
```

In [9]:
```
X_test.tail()
```

Out[9]:

	Age	Fare	Age_imputed
89	24.0	8.0500	24.0
80	22.0	9.0000	22.0
846	NaN	69.5500	NaN
870	26.0	7.8958	26.0
251	29.0	10.4625	29.0

In [10]:
```
X_train['Age_imputed'][X_train['Age_imputed'].isnull()] = X_train['Age'].dropna().sample(X
X_test['Age_imputed'][X_test['Age_imputed'].isnull()] = X_train['Age'].dropna().sample(X_t
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:8870: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
```

In [11]:
```
X_train['Age'].dropna().sample(X_train['Age'].isnull().sum()).values
```

Out[11]:
```
array([54.  , 50.  , 30.  , 59.  , 40.  , 22.  ,  4.  , 54.  , 12.  ,
       31.  , 45.  , 62.  , 40.5 , 20.  , 36.  , 48.  , 16.  , 44.  ,
       30.  , 36.  , 48.  , 14.  , 20.  , 24.  , 36.  , 27.  , 29.  ,
       14.  , 54.  , 23.  , 56.  , 50.  , 13.  , 11.  , 25.  , 30.  ,
```

```
 9.  , 16.  , 25.  , 45.  , 16.  , 65.  , 49.  , 70.5 , 50.  ,
42.  , 16.  , 19.  , 58.  , 26.  , 40.5 , 24.  , 34.  , 45.  ,
24.  , 24.  , 30.  , 32.  , 47.  , 14.5 , 29.  , 28.  , 30.  ,
17.  , 22.  , 62.  , 38.  , 22.  ,  0.42, 21.  , 33.  ,  2.  ,
32.5 , 18.  , 60.  , 21.  , 18.  ,  4.  , 28.5 , 70.  , 28.  ,
58.  , 34.  , 52.  , 44.  , 38.  ,  2.  , 21.  , 34.  , 18.  ,
54.  , 25.  , 33.  , 34.  , 62.  , 59.  ,  7.  , 28.  ,  0.75,
33.  , 20.  , 50.  , 32.  , 16.  , 23.5 , 36.  , 25.  , 28.  ,
32.  , 20.  , 17.  , 17.  , 25.  , 33.  , 27.  , 19.  , 51.  ,
39.  , 28.  , 32.  , 34.  , 22.  , 23.  , 35.  , 35.  , 22.  ,
23.  , 24.  , 35.  , 24.  , 45.  , 27.  , 27.  , 22.  , 42.  ,
21.  , 47.  , 45.5 , 41.  , 51.  , 37.  , 24.  , 39.  , 48.  ,
27.  , 30.  , 18.  ,  9.  ])
```

In [12]:
```
X_train['Age'].isnull().sum()
```

Out[12]:
```
148
```

In [13]:
```
X_train
```

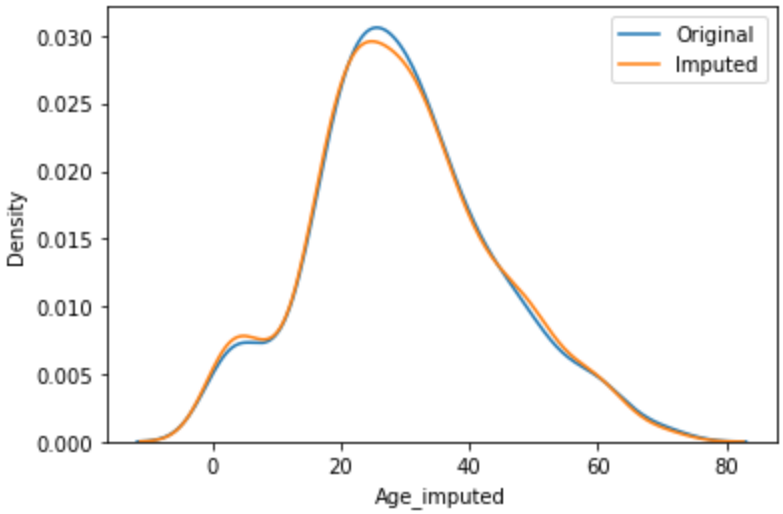
Out[13]:

	Age	Fare	Age_imputed
30	40.0	27.7208	40.0
10	4.0	16.7000	4.0
873	47.0	9.0000	47.0
182	9.0	31.3875	9.0
876	20.0	9.8458	20.0
...	...	...	...
534	30.0	8.6625	30.0
584	NaN	8.7125	59.0
493	71.0	49.5042	71.0
527	NaN	221.7792	34.0
168	NaN	25.9250	49.0

712 rows × 3 columns

In [14]:
```
sns.distplot(X_train['Age'],label='Original',hist=False)
sns.distplot(X_train['Age_imputed'],label = 'Imputed',hist=False)

plt.legend()
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



In [15]:
```
print('Original variable variance: ', X_train['Age'].var())
print('Variance after random imputation: ', X_train['Age_imputed'].var())
```

```
Original variable variance:  204.3495133904614
Variance after random imputation:  208.142956065796
```

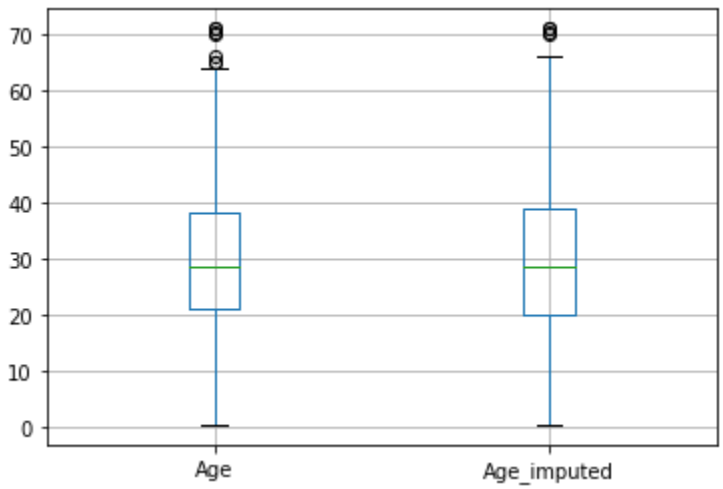
In [16]:
```
X_train[['Fare', 'Age', 'Age_imputed']].cov()
```

Out[16]:

	Fare	Age	Age_imputed
Fare	2368.246832	71.512440	58.923994
Age	71.512440	204.349513	204.349513
Age_imputed	58.923994	204.349513	208.142956

In [17]:
```
X_train[['Age', 'Age_imputed']].boxplot()
```

Out[17]:
```
<AxesSubplot:>
```



In [19]:
```
data = pd.read_csv('house-train.csv',usecols=['GarageQual','FireplaceQu', 'SalePrice'])
```

In [20]:
```
data.head()
```

Out[20]:

	FireplaceQu	GarageQual	SalePrice

	FireplaceQu	GarageQual	SalePrice
0	NaN	TA	208500
1	TA	TA	181500
2	TA	TA	223500
3	Gd	TA	140000
4	TA	TA	250000

In [21]:
```
data.isnull().mean() * 100
```

Out[21]:
```
FireplaceQu    47.260274
GarageQual      5.547945
SalePrice       0.000000
dtype: float64
```

In [22]:
```
X = data
y = data['SalePrice']
```

In [23]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [24]:
```
X_train['GarageQual_imputed'] = X_train['GarageQual']
X_test['GarageQual_imputed'] = X_test['GarageQual']

X_train['FireplaceQu_imputed'] = X_train['FireplaceQu']
X_test['FireplaceQu_imputed'] = X_test['FireplaceQu']
```

```
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/3838090268.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['GarageQual_imputed'] = X_train['GarageQual']
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/3838090268.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test['GarageQual_imputed'] = X_test['GarageQual']
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/3838090268.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['FireplaceQu_imputed'] = X_train['FireplaceQu']
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/3838090268.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test['FireplaceQu_imputed'] = X_test['FireplaceQu']
```

In [25]:
```
X_train.sample(5)
```

Out[25]:

	FireplaceQu	GarageQual	SalePrice	GarageQual_imputed	FireplaceQu_imputed
745	TA	TA	299800	TA	TA
300	Gd	TA	157000	TA	Gd
695	TA	TA	176000	TA	TA
1170	Po	TA	171000	TA	Po
1110	TA	TA	188000	TA	TA

In [26]:

```
X_train['GarageQual_imputed'][X_train['GarageQual_imputed'].isnull()] = X_train['GarageQua
X_test['GarageQual_imputed'][X_test['GarageQual_imputed'].isnull()] = X_train['GarageQual'

X_train['FireplaceQu_imputed'][X_train['FireplaceQu_imputed'].isnull()] = X_train['Firepla
X_test['FireplaceQu_imputed'][X_test['FireplaceQu_imputed'].isnull()] = X_train['Fireplace
```

```
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/856878696.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_train['GarageQual_imputed'][X_train['GarageQual_imputed'].isnull()] = X_train['GarageQ
ual'].dropna().sample(X_train['GarageQual'].isnull().sum()).values
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:8870: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/856878696.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_test['GarageQual_imputed'][X_test['GarageQual_imputed'].isnull()] = X_train['GarageQua
l'].dropna().sample(X_test['GarageQual'].isnull().sum()).values
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/856878696.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_train['FireplaceQu_imputed'][X_train['FireplaceQu_imputed'].isnull()] = X_train['Firep
laceQu'].dropna().sample(X_train['FireplaceQu'].isnull().sum()).values
C:\Users\HP\AppData\Local\Temp/ipykernel_80384/856878696.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  X_test['FireplaceQu_imputed'][X_test['FireplaceQu_imputed'].isnull()] = X_train['Firepla
ceQu'].dropna().sample(X_test['FireplaceQu'].isnull().sum()).values
```

In [27]:

```
temp = pd.concat(
        [
            X_train['GarageQual'].value_counts() / len(X_train['GarageQual'].dropna()),
            X_train['GarageQual_imputed'].value_counts() / len(X_train)
        ],
        axis=1)

temp.columns = ['original', 'imputed']
```

In [28]:
```
temp
```

Out[28]:

	original	imputed
TA	0.951043	0.952055
Fa	0.037171	0.035959
Gd	0.009973	0.010274
Po	0.000907	0.000856
Ex	0.000907	0.000856

In [29]:
```
temp = pd.concat(
        [
            X_train['FireplaceQu'].value_counts() / len(X_train['FireplaceQu'].dropna()),
            X_train['FireplaceQu_imputed'].value_counts() / len(df)
        ],
        axis=1)

temp.columns = ['original', 'imputed']

temp
```

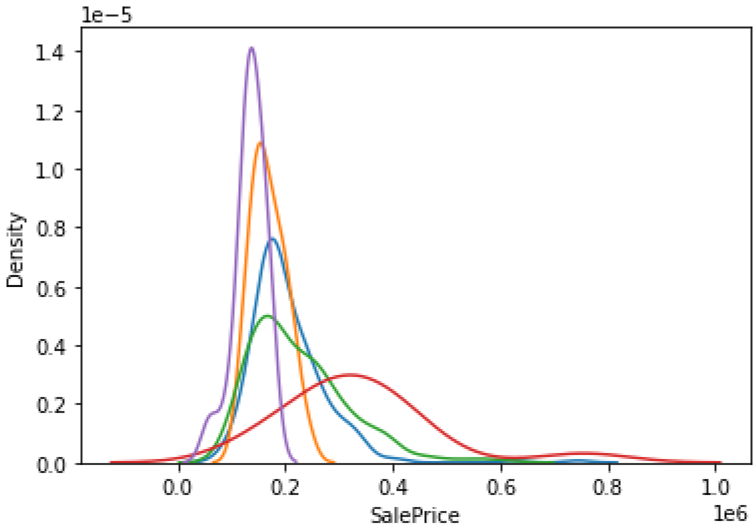
Out[29]:

	original	imputed
Gd	0.494272	0.647587
TA	0.412439	0.543210
Fa	0.040917	0.052750
Po	0.027823	0.035915
Ex	0.024550	0.031425

In [30]:
```
for category in X_train['FireplaceQu'].dropna().unique():
    sns.distplot(X_train[X_train['FireplaceQu'] == category]['SalePrice'],hist=False,label
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
```

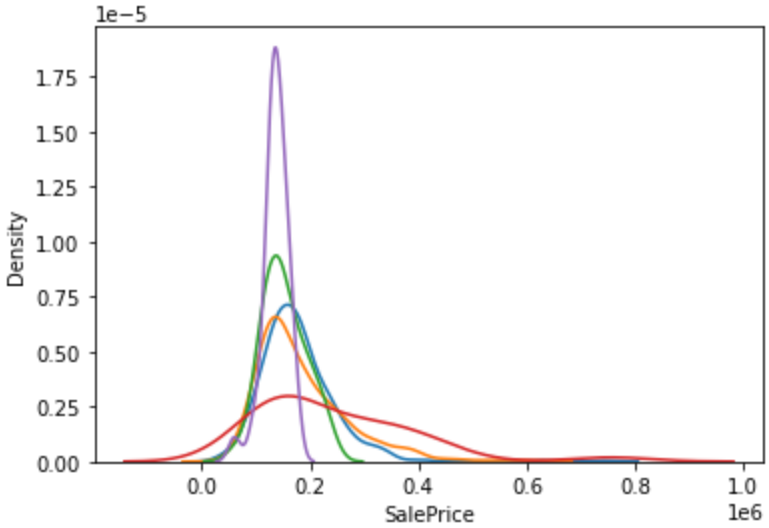
```
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



In [31]:

```python
for category in X_train['FireplaceQu_imputed'].dropna().unique():
    sns.distplot(X_train[X_train['FireplaceQu_imputed'] == category]['SalePrice'],hist=Fal
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```


1e−5
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
0.0
0.2
0.4
0.6
0.8
1.0
1e6
SalePrice

In [ ]:

KNN Imputer

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

from sklearn.model_selection import train_test_split

from sklearn.impute import KNNImputer,SimpleImputer
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score
```

In [2]:
```
df = pd.read_csv('train.csv')[['Age','Pclass','Fare','Survived']]
```

In [3]:
```
df.head()
```

Out[3]:

	Age	Pclass	Fare	Survived
0	22.0	3	7.2500	0
1	38.0	1	71.2833	1
2	26.0	3	7.9250	1
3	35.0	1	53.1000	1
4	35.0	3	8.0500	0

In [4]:
```
df.isnull().mean() * 100
```

Out[4]:
```
Age         19.86532
Pclass       0.00000
Fare         0.00000
Survived     0.00000
dtype: float64
```

In [5]:
```
X = df.drop(columns=['Survived'])
y = df['Survived']
```

In [6]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [7]:
```
X_train.head()
```

Out[7]:

	Age	Pclass	Fare
30	40.0	1	27.7208
10	4.0	3	16.7000
873	47.0	3	9.0000
182	9.0	3	31.3875
876	20.0	3	9.8458

In [8]:
```
knn = KNNImputer(n_neighbors=3,weights='distance')

X_train_trf = knn.fit_transform(X_train)
X_test_trf = knn.transform(X_test)
```

In [9]:
```
lr = LogisticRegression()

lr.fit(X_train_trf,y_train)

y_pred = lr.predict(X_test_trf)

accuracy_score(y_test,y_pred)
```

Out[9]:
```
0.7150837988826816
```

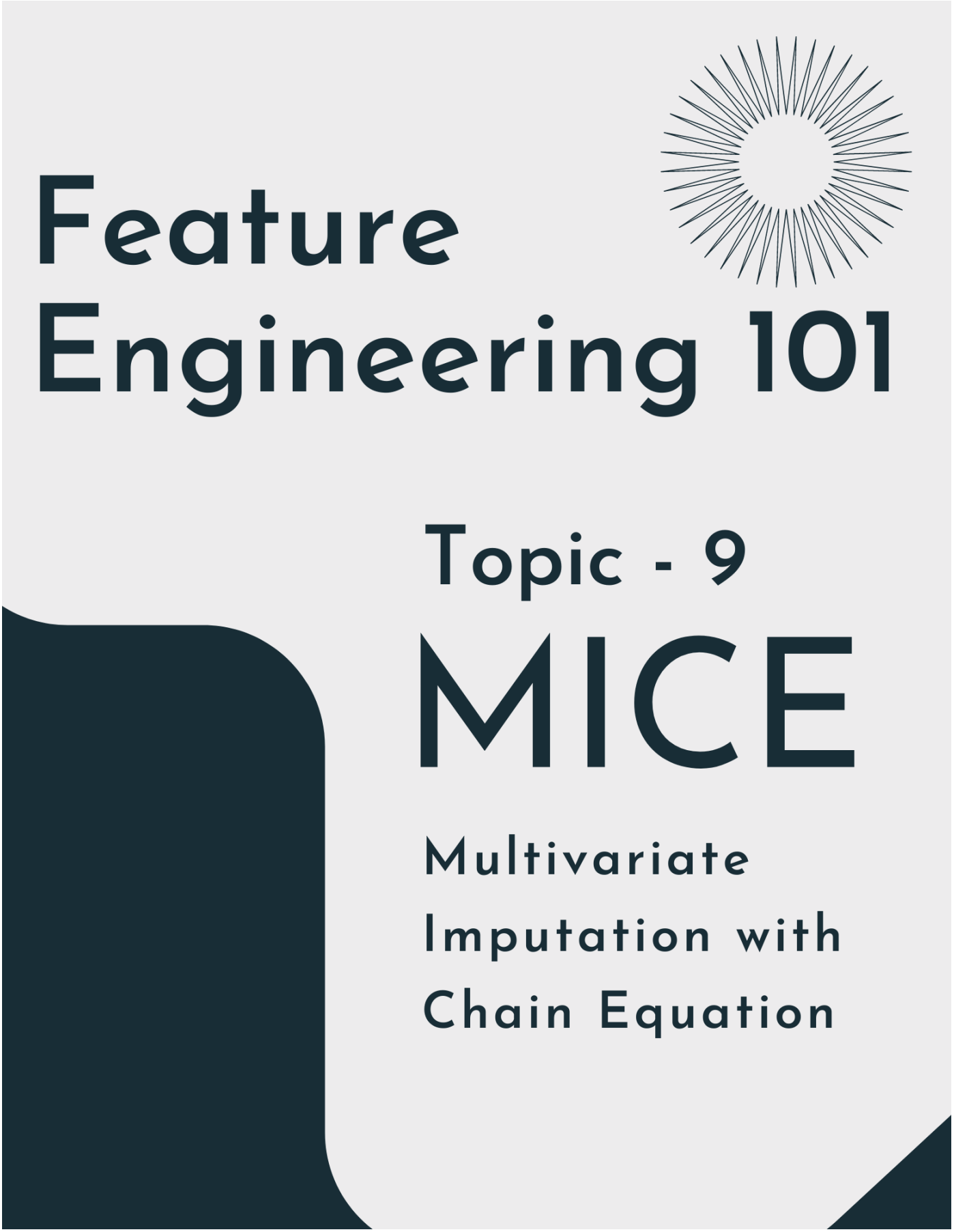
In [10]:
```
# Comparision with Simple Imputer --> mean

si = SimpleImputer()

X_train_trf2 = si.fit_transform(X_train)
X_test_trf2 = si.transform(X_test)
```

In [11]:
```
lr = LogisticRegression()

lr.fit(X_train_trf2,y_train)

y_pred2 = lr.predict(X_test_trf2)

accuracy_score(y_test,y_pred2)
```

Out[11]:
```
0.6927374301675978
```

In [ ]:

In [ ]:



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

from sklearn.linear_model import LinearRegression
```

In [2]:

```
df = np.round(pd.read_csv('50_Startups.csv')[['R&D Spend','Administration','Marketing Spen
np.random.seed(9)
df = df.sample(5)
df
```

Out[2]:

	R&D Spend	Administration	Marketing Spend	Profit
21	8.0	15.0	30.0	11.0
37	4.0	5.0	20.0	9.0
2	15.0	10.0	41.0	19.0
14	12.0	16.0	26.0	13.0
44	2.0	15.0	3.0	7.0

In [3]:

```
df = df.iloc[:,0:-1]
df
```

Out[3]:

	R&D Spend	Administration	Marketing Spend
21	8.0	15.0	30.0
37	4.0	5.0	20.0
2	15.0	10.0	41.0
14	12.0	16.0	26.0
44	2.0	15.0	3.0

In [4]:

```
df.iloc[1,0] = np.NaN
df.iloc[3,1] = np.NaN
df.iloc[-1,-1] = np.NaN
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1732: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:723: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value, self.name)
```

In [5]:

```
df.head()
```

Out[5]:

	R&D Spend	Administration	Marketing Spend
21	8.0	15.0	30.0
37	NaN	5.0	20.0
2	15.0	10.0	41.0
14	12.0	NaN	26.0

	R&D Spend	Administration	Marketing Spend
44	2.0	15.0	NaN

In [6]:
```
# Step 1 - Impute all missing values with mean of respective col

df0 = pd.DataFrame()

df0['R&D Spend'] = df['R&D Spend'].fillna(df['R&D Spend'].mean())
df0['Administration'] = df['Administration'].fillna(df['Administration'].mean())
df0['Marketing Spend'] = df['Marketing Spend'].fillna(df['Marketing Spend'].mean())
```

In [7]:
```
# 0th Iteration
df0
```

Out[7]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.00	30.00
37	9.25	5.00	20.00
2	15.00	10.00	41.00
14	12.00	11.25	26.00
44	2.00	15.00	29.25

In [8]:
```
# Remove the col1 imputed value
df1 = df0.copy()

df1.iloc[1,0] = np.NaN

df1
```

Out[8]:

	R&D Spend	Administration	Marketing Spend
21	8.0	15.00	30.00
37	NaN	5.00	20.00
2	15.0	10.00	41.00
14	12.0	11.25	26.00
44	2.0	15.00	29.25

In [9]:
```
# Use first 3 rows to build a model and use the last for prediction

X = df1.iloc[[0,2,3,4],1:3]
X
```

Out[9]:

	Administration	Marketing Spend
21	15.00	30.00
2	10.00	41.00
14	11.25	26.00
44	15.00	29.25

In [10]:
```
y = df1.iloc[[0,2,3,4],0]
y
```

Out[10]:
```
21     8.0
2     15.0
14    12.0
44     2.0
Name: R&D Spend, dtype: float64
```

In [11]:
```
lr = LinearRegression()
lr.fit(X,y)
lr.predict(df1.iloc[1,1:].values.reshape(1,2))
```

Out[11]:
```
array([23.14158651])
```

In [12]:
```
df1.iloc[1,0] = 23.14
```

In [13]:
```
df1
```

Out[13]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.00	30.00
37	23.14	5.00	20.00
2	15.00	10.00	41.00
14	12.00	11.25	26.00
44	2.00	15.00	29.25

In [14]:
```
# Remove the col2 imputed value

df1.iloc[3,1] = np.NaN

df1
```

Out[14]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.0	30.00
37	23.14	5.0	20.00
2	15.00	10.0	41.00
14	12.00	NaN	26.00
44	2.00	15.0	29.25

In [15]:
```
# Use last 3 rows to build a model and use the first for prediction
X = df1.iloc[[0,1,2,4],[0,2]]
X
```

Out[15]:

	R&D Spend	Marketing Spend
21	8.00	30.00
37	23.14	20.00

	R&D Spend	Marketing Spend
2	15.00	41.00
44	2.00	29.25

In [16]:
```
y = df1.iloc[[0,1,2,4],1]
y
```

Out[16]:
```
21    15.0
37     5.0
2     10.0
44    15.0
Name: Administration, dtype: float64
```

In [17]:
```
lr = LinearRegression()
lr.fit(X,y)
lr.predict(df1.iloc[3,[0,2]].values.reshape(1,2))
```

Out[17]:
```
array([11.06331285])
```

In [18]:
```
df1.iloc[3,1] = 11.06
```

In [19]:
```
df1
```

Out[19]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.00	30.00
37	23.14	5.00	20.00
2	15.00	10.00	41.00
14	12.00	11.06	26.00
44	2.00	15.00	29.25

In [20]:
```
# Remove the col3 imputed value
df1.iloc[4,-1] = np.NaN

df1
```

Out[20]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.00	30.0
37	23.14	5.00	20.0
2	15.00	10.00	41.0
14	12.00	11.06	26.0
44	2.00	15.00	NaN

In [21]:
```
# Use last 3 rows to build a model and use the first for prediction
X = df1.iloc[0:4,0:2]
X
```

Out[21]:

	R&D Spend	Administration
21	8.00	15.00
37	23.14	5.00
2	15.00	10.00
14	12.00	11.06

In [22]:
```
y = df1.iloc[0:4,-1]
y
```

Out[22]:
```
21    30.0
37    20.0
2     41.0
14    26.0
Name: Marketing Spend, dtype: float64
```

In [23]:
```
lr = LinearRegression()
lr.fit(X,y)
lr.predict(df1.iloc[4,0:2].values.reshape(1,2))
```

Out[23]:
```
array([31.56351448])
```

In [24]:
```
df1.iloc[4,-1] = 31.56
```

In [25]:
```
# After 1st Iteration
df1
```

Out[25]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.00	30.00
37	23.14	5.00	20.00
2	15.00	10.00	41.00
14	12.00	11.06	26.00
44	2.00	15.00	31.56

In [26]:
```
# Subtract 0th iteration from 1st iteration

df1 - df0
```

Out[26]:

	R&D Spend	Administration	Marketing Spend
21	0.00	0.00	0.00
37	13.89	0.00	0.00
2	0.00	0.00	0.00
14	0.00	-0.19	0.00
44	0.00	0.00	2.31

In [27]:
```
df2 = df1.copy()
```

```
df2.iloc[1,0] = np.NaN

df2
```

Out[27]:

	R&D Spend	Administration	Marketing Spend
21	8.0	15.00	30.00
37	NaN	5.00	20.00
2	15.0	10.00	41.00
14	12.0	11.06	26.00
44	2.0	15.00	31.56

In [28]:
```
X = df2.iloc[[0,2,3,4],1:3]
y = df2.iloc[[0,2,3,4],0]

lr = LinearRegression()
lr.fit(X,y)
lr.predict(df2.iloc[1,1:].values.reshape(1,2))
```

Out[28]:
```
array([23.78627207])
```

In [29]:
```
df2.iloc[1,0] = 23.78
```

In [30]:
```
df2.iloc[3,1] = np.NaN
X = df2.iloc[[0,1,2,4],[0,2]]
y = df2.iloc[[0,1,2,4],1]

lr = LinearRegression()
lr.fit(X,y)
lr.predict(df2.iloc[3,[0,2]].values.reshape(1,2))
```

Out[30]:
```
array([11.22020174])
```

In [31]:
```
df2.iloc[3,1] = 11.22
```

In [32]:
```
df2.iloc[4,-1] = np.NaN

X = df2.iloc[0:4,0:2]
y = df2.iloc[0:4,-1]

lr = LinearRegression()
lr.fit(X,y)
lr.predict(df2.iloc[4,0:2].values.reshape(1,2))
```

Out[32]:
```
array([38.87979054])
```

In [33]:
```
df2.iloc[4,-1] = 31.56
```

In [34]:
```
df2
```

Out[34]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.00	30.00
37	23.78	5.00	20.00
2	15.00	10.00	41.00
14	12.00	11.22	26.00
44	2.00	15.00	31.56

In [35]:
```
df2 - df1
```

Out[35]:

	R&D Spend	Administration	Marketing Spend
21	0.00	0.00	0.0
37	0.64	0.00	0.0
2	0.00	0.00	0.0
14	0.00	0.16	0.0
44	0.00	0.00	0.0

In [36]:
```
df3 = df2.copy()

df3.iloc[1,0] = np.NaN

df3
```

Out[36]:

	R&D Spend	Administration	Marketing Spend
21	8.0	15.00	30.00
37	NaN	5.00	20.00
2	15.0	10.00	41.00
14	12.0	11.22	26.00
44	2.0	15.00	31.56

In [37]:
```
X = df3.iloc[[0,2,3,4],1:3]
y = df3.iloc[[0,2,3,4],0]

lr = LinearRegression()
lr.fit(X,y)
lr.predict(df3.iloc[1,1:].values.reshape(1,2))
```

Out[37]:
```
array([24.57698058])
```

In [38]:
```
df3.iloc[1,0] = 24.57
```

In [39]:
```
df3.iloc[3,1] = np.NaN
X = df3.iloc[[0,1,2,4],[0,2]]
y = df3.iloc[[0,1,2,4],1]

lr = LinearRegression()
```

```
lr.fit(X,y)
lr.predict(df3.iloc[3,[0,2]].values.reshape(1,2))
```

Out[39]:
```
array([11.37282844])
```

In [40]:
```
df3.iloc[3,1] = 11.37
```

In [41]:
```
df3.iloc[4,-1] = np.NaN

X = df3.iloc[0:4,0:2]
y = df3.iloc[0:4,-1]

lr = LinearRegression()
lr.fit(X,y)
lr.predict(df3.iloc[4,0:2].values.reshape(1,2))
```

Out[41]:
```
array([45.53976417])
```

In [42]:
```
df3.iloc[4,-1] = 45.53
```

In [43]:
```
df2.iloc[3,1] = 11.22
```

In [44]:
```
df3
```

Out[44]:

	R&D Spend	Administration	Marketing Spend
21	8.00	15.00	30.00
37	24.57	5.00	20.00
2	15.00	10.00	41.00
14	12.00	11.37	26.00
44	2.00	15.00	45.53

In [45]:
```
df3 - df2
```

Out[45]:

	R&D Spend	Administration	Marketing Spend
21	0.00	0.00	0.00
37	0.79	0.00	0.00
2	0.00	0.00	0.00
14	0.00	0.15	0.00
44	0.00	0.00	13.97

In [ ]:
```

```

Feature Engineering 101

Topic - 10

Outliers

Outliers Detection Method

1. Z-Score
2. IQR
3. Winsorization or Percentile

The concept of outliers: What is it?

Outliers are data points that are significantly different from the majority of the other data points in a dataset. In machine learning, they can have a significant impact on the results of a model if they are not detected and handled appropriately. Outliers can be due to measurement errors, errors in data collection, or they can be genuine examples that are not representative of the population.

Z-Score

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('placement.csv')
```

In [3]:
```
df.shape
```

Out[3]:
```
(1000, 3)
```

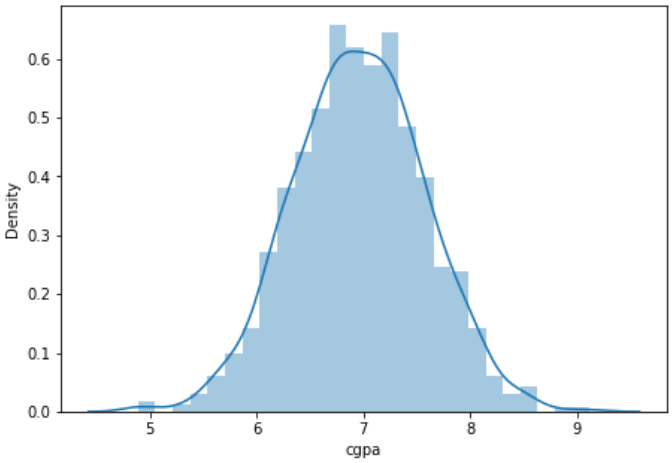
In [4]:
```
df.sample(5)
```

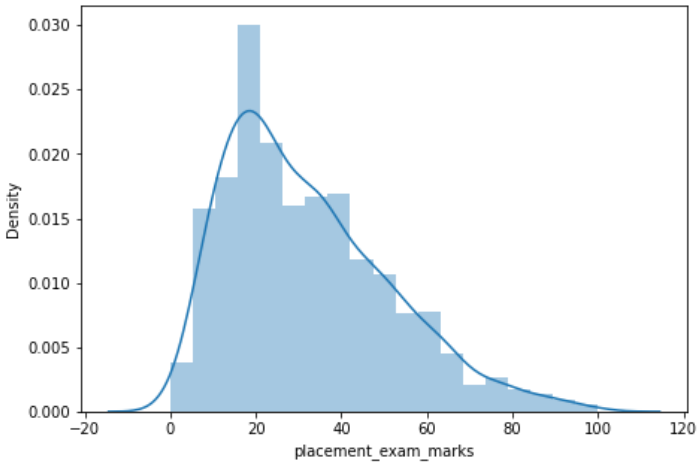
Out[4]:

	cgpa	placement_exam_marks	placed
719	7.17	26.0	0
457	6.58	20.0	0
542	7.06	22.0	0
733	7.07	10.0	0
770	7.33	67.0	1

In [5]:
```
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['cgpa'])

plt.subplot(1,2,2)
sns.distplot(df['placement_exam_marks'])

plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





In [6]:
```
df['placement_exam_marks'].skew()
```

Out[6]:
```
0.8356419499466834
```

In [7]:
```
print("Mean value of cgpa",df['cgpa'].mean())
print("Std value of cgpa",df['cgpa'].std())
print("Min value of cgpa",df['cgpa'].min())
print("Max value of cgpa",df['cgpa'].max())
```

```
Mean value of cgpa 6.96124000000001
Std value of cgpa 0.6158978751323894
Min value of cgpa 4.89
Max value of cgpa 9.12
```

In [8]:
```
# Finding the boundary values
print("Highest allowed",df['cgpa'].mean() + 3*df['cgpa'].std())
print("Lowest allowed",df['cgpa'].mean() - 3*df['cgpa'].std())
```

```
Highest allowed 8.808933625397177
Lowest allowed 5.113546374602842
```

In [9]:
```
# Finding the outliers
df[(df['cgpa'] > 8.80) | (df['cgpa'] < 5.11)]
```

Out[9]:

	cgpa	placement_exam_marks	placed
485	4.92	44.0	1
995	8.87	44.0	1
996	9.12	65.0	1
997	4.89	34.0	0
999	4.90	10.0	1

Trimming

In [10]:
```
# Trimming

new_df = df[(df['cgpa'] < 8.80) & (df['cgpa'] > 5.11)]
new_df
```

Out[10]:

	cgpa	placement_exam_marks	placed
0	7.19	26.0	1
1	7.46	38.0	1
2	7.54	40.0	1
3	6.42	8.0	1
4	7.23	17.0	0
...	...	...	...
991	7.04	57.0	0
992	6.26	12.0	0
993	6.73	21.0	1
994	6.48	63.0	0
998	8.62	46.0	1

995 rows × 3 columns

In [11]:
```
# Approach 2

# Calculating the Zscore

df['cgpa_zscore'] = (df['cgpa'] - df['cgpa'].mean())/df['cgpa'].std()
```

In [12]:
```
df.head()
```

Out[12]:

	cgpa	placement_exam_marks	placed	cgpa_zscore
0	7.19	26.0	1	0.371425
1	7.46	38.0	1	0.809810
2	7.54	40.0	1	0.939701
3	6.42	8.0	1	-0.878782
4	7.23	17.0	0	0.436371

In [13]:
```
df[df['cgpa_zscore'] > 3]
```

Out[13]:

	cgpa	placement_exam_marks	placed	cgpa_zscore
995	8.87	44.0	1	3.099150
996	9.12	65.0	1	3.505062

In [14]:
```
df[df['cgpa_zscore'] < -3]
```

Out[14]:

	cgpa	placement_exam_marks	placed	cgpa_zscore
485	4.92	44.0	1	-3.314251
997	4.89	34.0	0	-3.362960
999	4.90	10.0	1	-3.346724

In [15]:
```
df[(df['cgpa_zscore'] > 3) | (df['cgpa_zscore'] < -3)]
```

Out[15]:

	cgpa	placement_exam_marks	placed	cgpa_zscore
485	4.92	44.0	1	-3.314251
995	8.87	44.0	1	3.099150
996	9.12	65.0	1	3.505062
997	4.89	34.0	0	-3.362960
999	4.90	10.0	1	-3.346724

In [16]:
```
# Trimming
new_df = df[(df['cgpa_zscore'] < 3) & (df['cgpa_zscore'] > -3)]
```

In [17]:
```
new_df
```

Out[17]:

	cgpa	placement_exam_marks	placed	cgpa_zscore
0	7.19	26.0	1	0.371425
1	7.46	38.0	1	0.809810
2	7.54	40.0	1	0.939701
3	6.42	8.0	1	-0.878782
4	7.23	17.0	0	0.436371
...	...	...	...	...
991	7.04	57.0	0	0.127878
992	6.26	12.0	0	-1.138565
993	6.73	21.0	1	-0.375452
994	6.48	63.0	0	-0.781363
998	8.62	46.0	1	2.693239

995 rows × 4 columns

In [18]:
```
new_df['cgpa'].describe()
```

Out[18]:
```
count    995.000000
mean       6.963357
std        0.600082
min        5.230000
25%        6.550000
50%        6.960000
75%        7.365000
max        8.620000
Name: cgpa, dtype: float64
```

Capping

In [19]:
```
upper_limit = df['cgpa'].mean() + 3*df['cgpa'].std()
lower_limit = df['cgpa'].mean() - 3*df['cgpa'].std()
```

In [20]:
```
upper_limit
```

Out[20]:
```
8.808933625397177
```

In [21]:
```
lower_limit
```

Out[21]:
```
5.113546374602842
```

In [22]:
```
#Capping fun
df['cgpa'] = np.where(
    df['cgpa']>upper_limit,
    upper_limit,
    np.where(
        df['cgpa']<lower_limit,
        lower_limit,
```

```
        df['cgpa']
    )
)
```

In [23]:
```
df.shape
```

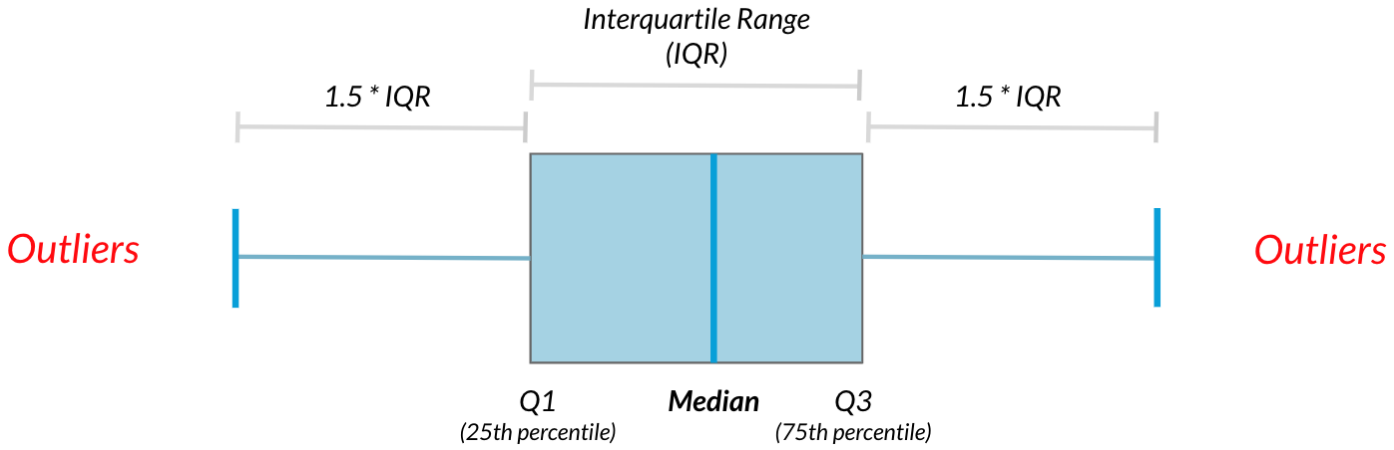
Out[23]:
```
(1000, 4)
```

In [24]:
```
df['cgpa'].describe()
```

Out[24]:
```
count    1000.000000
mean        6.961499
std         0.612688
min         5.113546
25%         6.550000
50%         6.960000
75%         7.370000
max         8.808934
Name: cgpa, dtype: float64
```

In [ ]:
```

```

IQR (Inter-quartile range)



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

In [2]:
```
df = pd.read_csv('placement.csv')
```

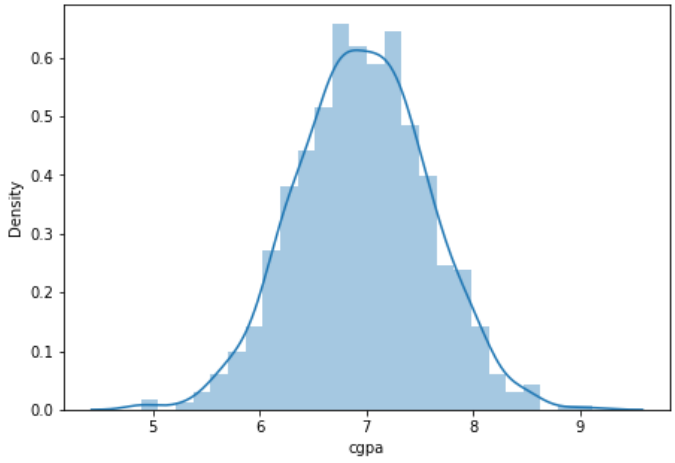
In [3]:
```
df.head()
```

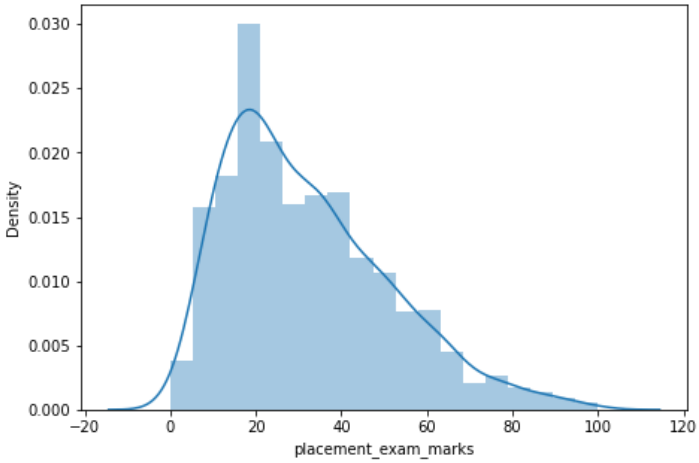
Out[3]:

	cgpa	placement_exam_marks	placed
0	7.19	26.0	1
1	7.46	38.0	1
2	7.54	40.0	1
3	6.42	8.0	1
4	7.23	17.0	0

In [4]:
```
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['cgpa'])

plt.subplot(1,2,2)
sns.distplot(df['placement_exam_marks'])

plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





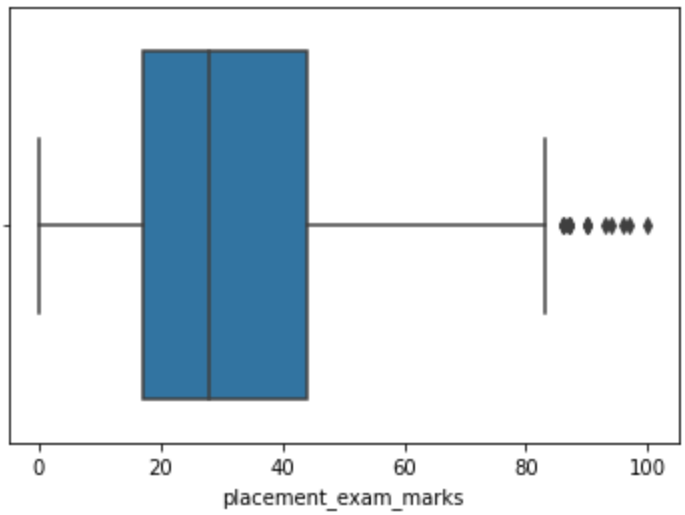
In [5]:
```
df['placement_exam_marks'].describe()
```

Out[5]:
```
count    1000.000000
mean       32.225000
std        19.130822
min         0.000000
25%        17.000000
50%        28.000000
75%        44.000000
max       100.000000
Name: placement_exam_marks, dtype: float64
```

In [6]:
```
sns.boxplot(df['placement_exam_marks'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
```

Out[6]:
```
<AxesSubplot:xlabel='placement_exam_marks'>
```



In [7]:
```
# Finding the IQR
percentile25 = df['placement_exam_marks'].quantile(0.25)
percentile75 = df['placement_exam_marks'].quantile(0.75)
```

In [8]:
```
percentile75
```

Out[8]:
```
44.0
```

In [9]:
```
iqr = percentile75 - percentile25
```

In [10]:
```
iqr
```

Out[10]:
```
27.0
```

In [11]:
```
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr
```

In [12]:
```
print("Upper limit",upper_limit)
print("Lower limit",lower_limit)
```

```
Upper limit 84.5
Lower limit -23.5
```

Finding Outliers

In [13]:
```
df[df['placement_exam_marks'] > upper_limit]
```

Out[13]:

	cgpa	placement_exam_marks	placed
9	7.75	94.0	1
40	6.60	86.0	1
61	7.51	86.0	0
134	6.33	93.0	0
162	7.80	90.0	0
283	7.09	87.0	0
290	8.38	87.0	0
311	6.97	87.0	1
324	6.64	90.0	0
630	6.56	96.0	1
685	6.05	87.0	1
730	6.14	90.0	1
771	7.31	86.0	1
846	6.99	97.0	0
917	5.95	100.0	0

In [14]:
```
df[df['placement_exam_marks'] < lower_limit]
```

Out[14]:

	cgpa	placement_exam_marks	placed

Trimming

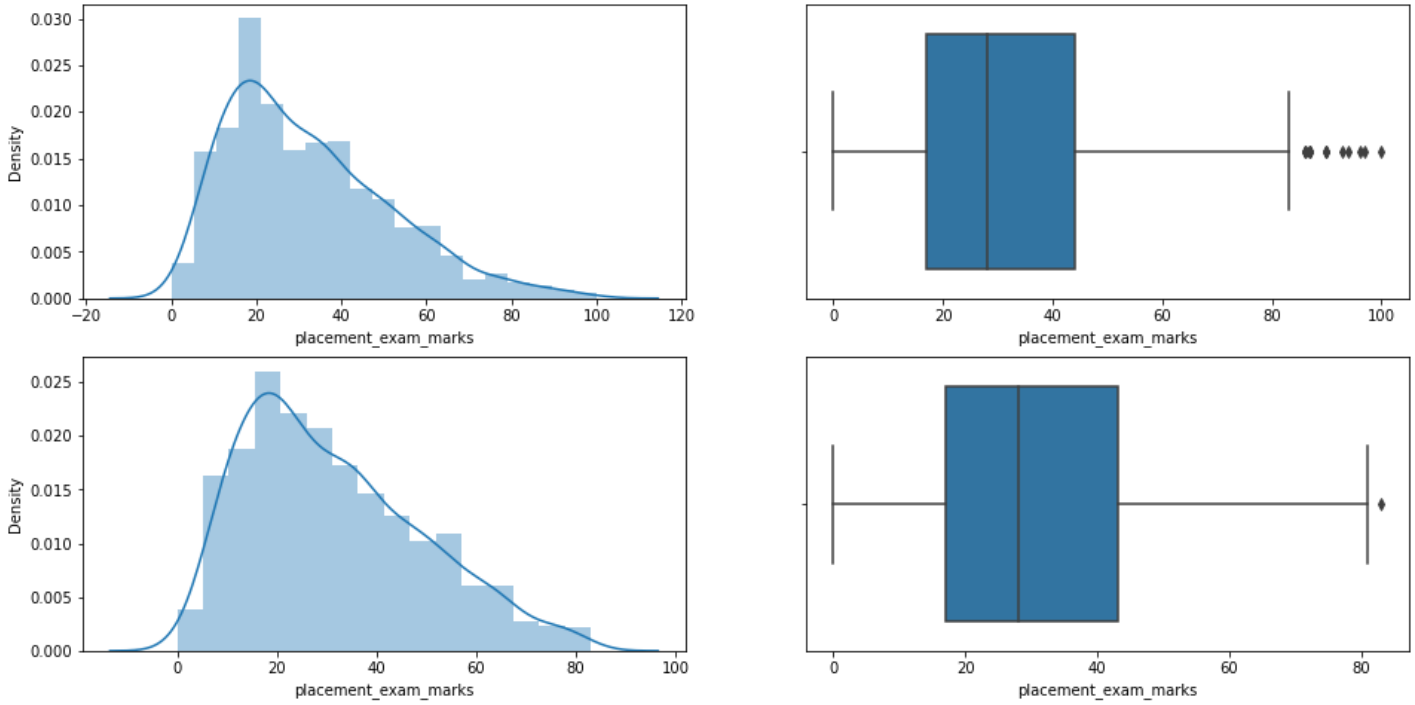
In [15]:
```
new_df = df[df['placement_exam_marks'] < upper_limit]
```

In [16]:
```
new_df.shape
```

Out[16]:
```
(985, 3)
```

In [17]:
```
# Comparing

plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(df['placement_exam_marks'])

plt.subplot(2,2,2)
sns.boxplot(df['placement_exam_marks'])

plt.subplot(2,2,3)
sns.distplot(new_df['placement_exam_marks'])

plt.subplot(2,2,4)
sns.boxplot(new_df['placement_exam_marks'])

plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
```



Capping

In [20]:
```
new_df_cap = df.copy()

new_df_cap['placement_exam_marks'] = np.where(
    new_df_cap['placement_exam_marks'] > upper_limit,
    upper_limit,
    np.where(
        new_df_cap['placement_exam_marks'] < lower_limit,
        lower_limit,
        new_df_cap['placement_exam_marks']
    )
)
```

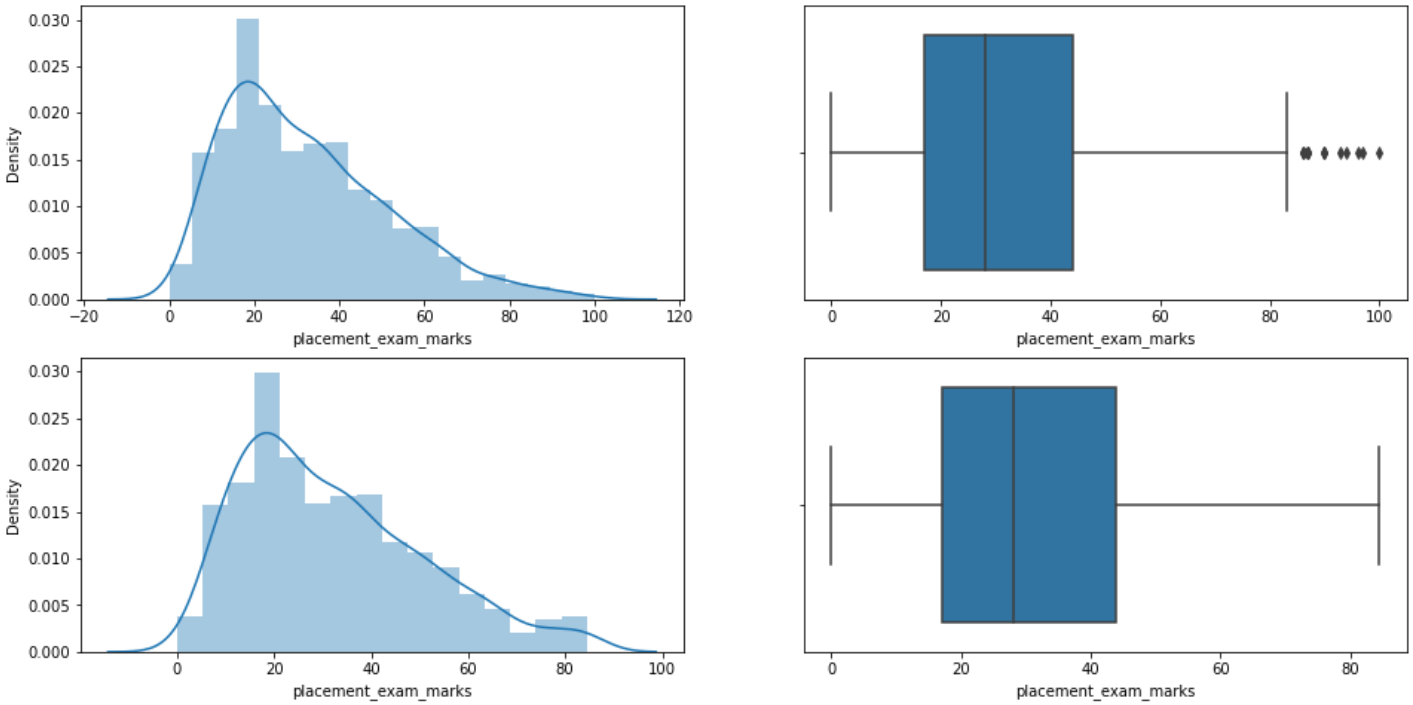
In [22]:
```
new_df_cap.shape
```

Out[22]:
```
(1000, 3)
```

In [23]:
```
# Comparing

plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(df['placement_exam_marks'])

plt.subplot(2,2,2)
sns.boxplot(df['placement_exam_marks'])

plt.subplot(2,2,3)
sns.distplot(new_df_cap['placement_exam_marks'])

plt.subplot(2,2,4)
sns.boxplot(new_df_cap['placement_exam_marks'])

plt.show()
```

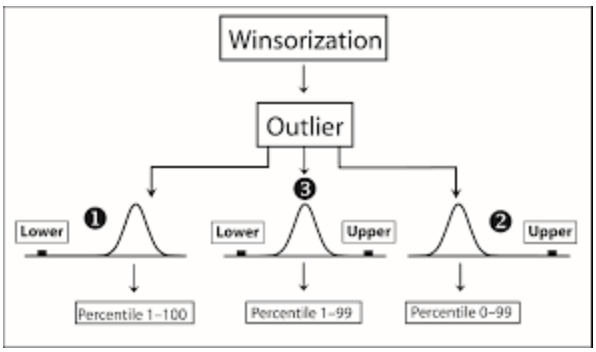
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
```

```
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
```



In [ ]:

Winsorization or Persentile



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

In [2]:
```
df = pd.read_csv('weight-height.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

In [4]:
```
df.shape
```

Out[4]:
```
(10000, 3)
```

In [5]:
```
df['Height'].describe()
```

Out[5]:
```
count    10000.000000
mean        66.367560
std          3.847528
min         54.263133
25%         63.505620
50%         66.318070
75%         69.174262
max         78.998742
Name: Height, dtype: float64
```

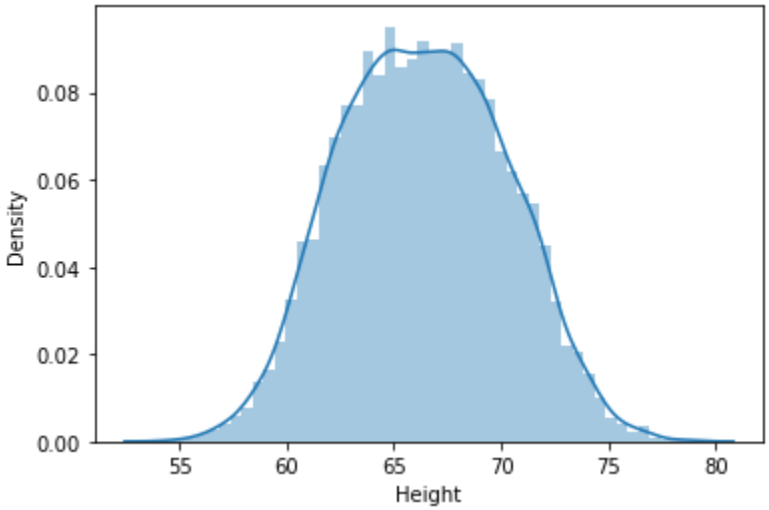
In [6]:
```
import seaborn as sns
```

In [7]:
```
sns.distplot(df['Height'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
```

```
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

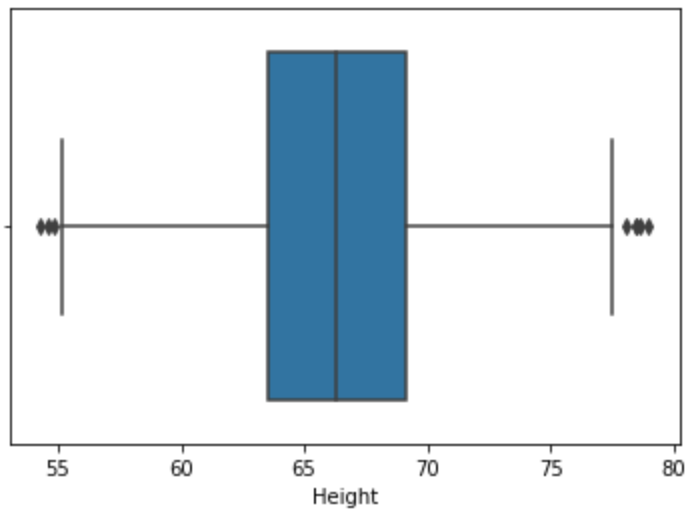
Out[7]:
```
<AxesSubplot:xlabel='Height', ylabel='Density'>
```



In [8]:
```
sns.boxplot(df['Height'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
```

Out[8]:
```
<AxesSubplot:xlabel='Height'>
```



In [9]:
```
upper_limit = df['Height'].quantile(0.99)
upper_limit
```

Out[9]:
```
74.7857900583366
```

In [10]:
```
lower_limit = df['Height'].quantile(0.01)
lower_limit
```

Out[10]:
```
58.13441158671655
```

In [11]:
```
new_df = df[(df['Height'] <= 74.78) & (df['Height'] >= 58.13)]
```

In [12]:
```
new_df['Height'].describe()
```

Out[12]:
```
count    9799.000000
mean       66.363507
std         3.644267
min        58.134496
25%        63.577147
50%        66.317899
75%        69.119859
max        74.767447
Name: Height, dtype: float64
```

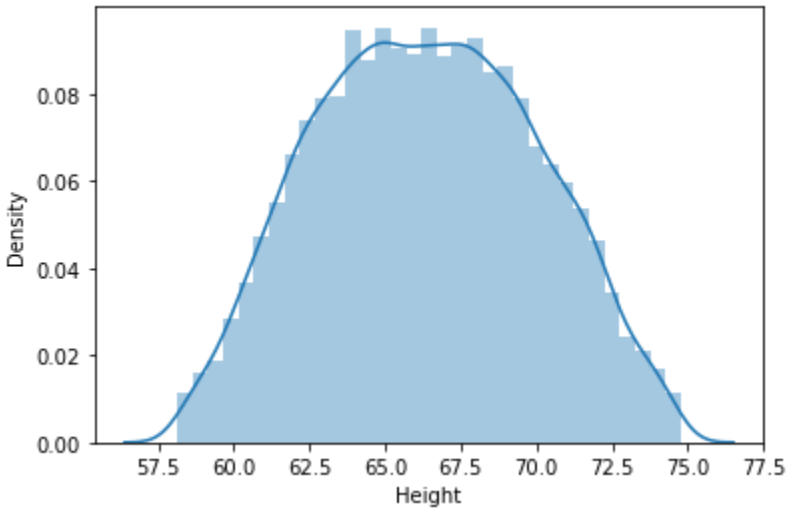
In [13]:
```
df['Height'].describe()
```

Out[13]:
```
count    10000.000000
mean        66.367560
std          3.847528
min         54.263133
25%         63.505620
50%         66.318070
75%         69.174262
max         78.998742
Name: Height, dtype: float64
```

In [14]:
```
sns.distplot(new_df['Height'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

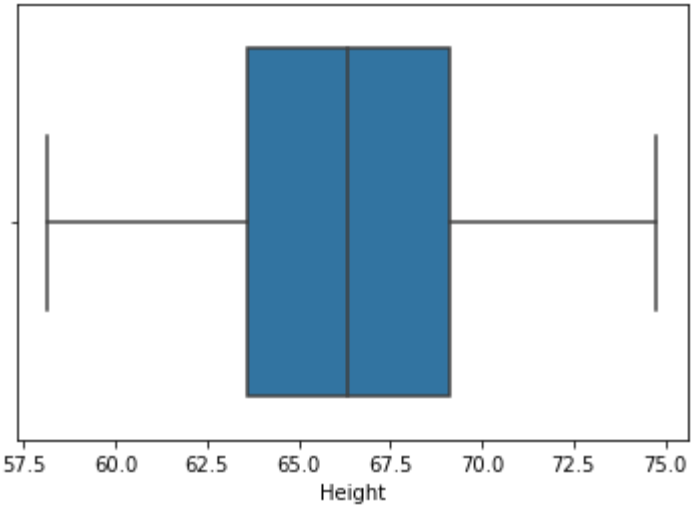
Out[14]:
```
<AxesSubplot:xlabel='Height', ylabel='Density'>
```



In [15]:
```
sns.boxplot(new_df['Height'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
```

Out[15]:
```
<AxesSubplot:xlabel='Height'>
```



In [16]:
```
# Capping --> Winsorization
df['Height'] = np.where(df['Height'] >= upper_limit,
        upper_limit,
        np.where(df['Height'] <= lower_limit,
        lower_limit,
        df['Height']))
```

In [17]:
```
df.shape
```

Out[17]:
```
(10000, 3)
```

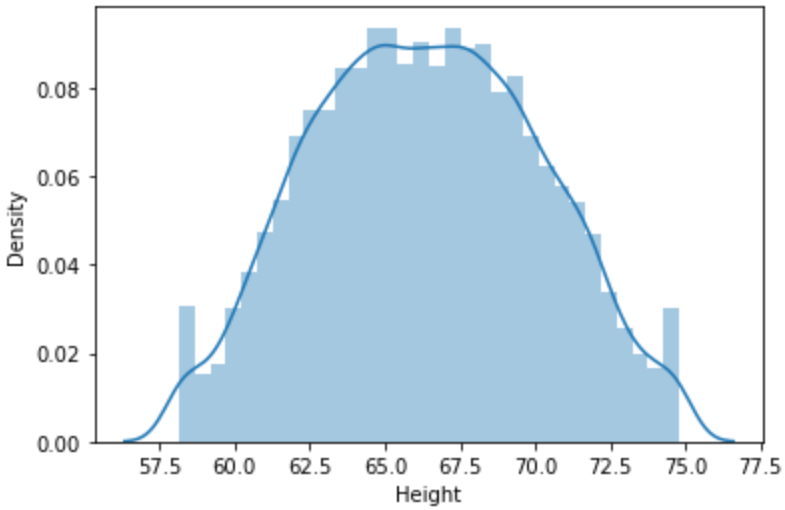
In [18]:
```
df['Height'].describe()
```

Out[18]:
```
count    10000.000000
mean        66.366281
std          3.795717
min         58.134412
25%         63.505620
50%         66.318070
75%         69.174262
max         74.785790
Name: Height, dtype: float64
```

In [19]:
```
sns.distplot(df['Height'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

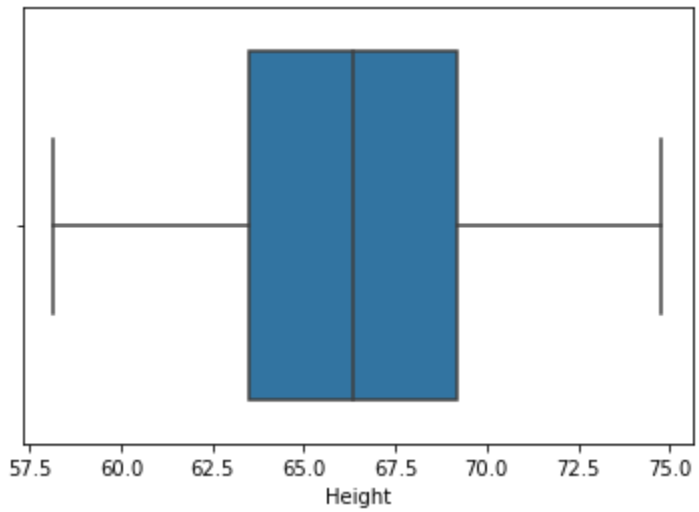
Out[19]:
```
<AxesSubplot:xlabel='Height', ylabel='Density'>
```



In [20]:
```
sns.boxplot(df['Height'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:xlabel='Height'>
```

Out[20]:

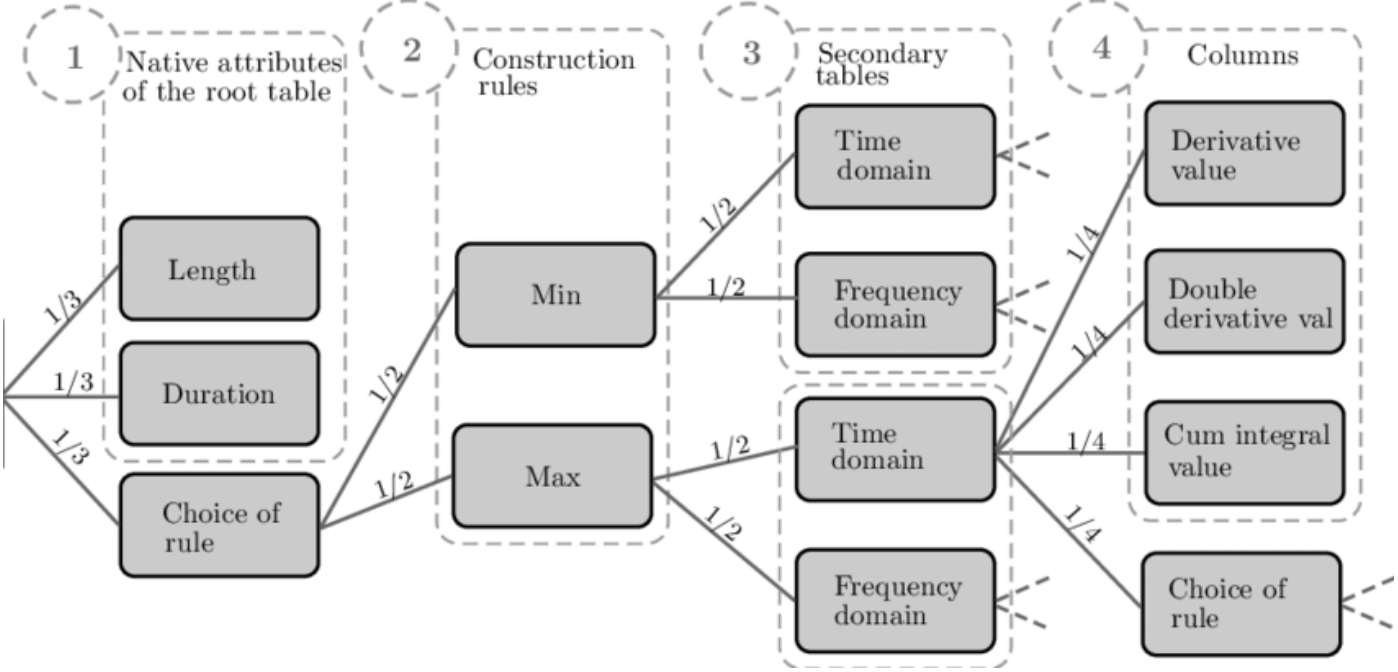


In [ ]:

Feature Engineering 101

Topic - 11

Feature Construction



Feature Construction

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

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

import seaborn as sns
```

In [2]:
```
df = pd.read_csv('train.csv')[['Age','Pclass','SibSp','Parch','Survived']]
```

In [3]:
```
df.head()
```

Out[3]:

	Age	Pclass	SibSp	Parch	Survived
0	22.0	3	1	0	0
1	38.0	1	1	0	1
2	26.0	3	0	0	1
3	35.0	1	1	0	1
4	35.0	3	0	0	0

In [4]:
```
df.dropna(inplace=True)
```

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

Out[5]:

	Age	Pclass	SibSp	Parch	Survived
0	22.0	3	1	0	0
1	38.0	1	1	0	1

	Age	Pclass	SibSp	Parch	Survived
2	26.0	3	0	0	1
3	35.0	1	1	0	1
4	35.0	3	0	0	0

In [6]:
```
X = df.iloc[:,0:4]
y = df.iloc[:,-1]
```

In [7]:
```
X.head()
```

Out[7]:

	Age	Pclass	SibSp	Parch
0	22.0	3	1	0
1	38.0	1	1	0
2	26.0	3	0	0
3	35.0	1	1	0
4	35.0	3	0	0

In [8]:
```
np.mean(cross_val_score(LogisticRegression(),X,y,scoring='accuracy',cv=20))
```

Out[8]:
```
0.6933333333333332
```

Applying Feature Construction

In [9]:
```
X['Family_size'] = X['SibSp'] + X['Parch'] + 1
```

In [10]:
```
X.head()
```

Out[10]:

	Age	Pclass	SibSp	Parch	Family_size
0	22.0	3	1	0	2
1	38.0	1	1	0	2
2	26.0	3	0	0	1
3	35.0	1	1	0	2
4	35.0	3	0	0	1

In [11]:
```
def myfunc(num):
    if num == 1:
        #alone
        return 0
    elif num >1 and num <=4:
        # small family
        return 1
    else:
```

```
        # large family
        return 2
```

In [12]:
```
myfunc(4)
```

Out[12]:
```
1
```

In [13]:
```
X['Family_type'] = X['Family_size'].apply(myfunc)
```

In [14]:
```
X.head()
```

Out[14]:

	Age	Pclass	SibSp	Parch	Family_size	Family_type
0	22.0	3	1	0	2	1
1	38.0	1	1	0	2	1
2	26.0	3	0	0	1	0
3	35.0	1	1	0	2	1
4	35.0	3	0	0	1	0

In [15]:
```
X.drop(columns=['SibSp','Parch','Family_size'],inplace=True)
```

In [16]:
```
X.head()
```

Out[16]:

	Age	Pclass	Family_type
0	22.0	3	1
1	38.0	1	1
2	26.0	3	0
3	35.0	1	1
4	35.0	3	0

In [17]:
```
np.mean(cross_val_score(LogisticRegression(),X,y,scoring='accuracy',cv=20))
```

Out[17]:
```
0.7003174603174602
```

Feature Splitting

In [1]:

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

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

import seaborn as sns
```

In [2]:
```
df = pd.read_csv('train.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [4]:
```
df['Name']
```

Out[4]:
```
0                                Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                 Heikkinen, Miss. Laina
3           Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                               Allen, Mr. William Henry
                             ...
886                             Montvila, Rev. Juozas
887                      Graham, Miss. Margaret Edith
888          Johnston, Miss. Catherine Helen "Carrie"
889                              Behr, Mr. Karl Howell
890                                Dooley, Mr. Patrick
Name: Name, Length: 891, dtype: object
```

In [5]:
```
df['Name'].str.split(', ', expand=True)[1].str.split('.', expand=True)[0]
```

Out[5]:
```
0        Mr
1       Mrs
2      Miss
3       Mrs
4        Mr
       ...
886     Rev
887    Miss
888    Miss
889      Mr
```

```
890      Mr
Name: 0, Length: 891, dtype: object
```

In [6]:
```
df['Title'] = df['Name'].str.split(', ', expand=True)[1].str.split('.', expand=True)[0]
```

In [7]:
```
df['Name'].str.split(', ', expand=True)[1].str.split('.', expand=True)[0]
```

Out[7]:
```
0        Mr
1       Mrs
2      Miss
3       Mrs
4        Mr
       ...
886     Rev
887    Miss
888    Miss
889      Mr
890      Mr
Name: 0, Length: 891, dtype: object
```

In [8]:
```
df[['Title','Name']]
```

Out[8]:

	Title	Name
0	Mr	Braund, Mr. Owen Harris
1	Mrs	Cumings, Mrs. John Bradley (Florence Briggs Th...
2	Miss	Heikkinen, Miss. Laina
3	Mrs	Futrelle, Mrs. Jacques Heath (Lily May Peel)
4	Mr	Allen, Mr. William Henry
...	...	...
886	Rev	Montvila, Rev. Juozas
887	Miss	Graham, Miss. Margaret Edith
888	Miss	Johnston, Miss. Catherine Helen "Carrie"
889	Mr	Behr, Mr. Karl Howell
890	Mr	Dooley, Mr. Patrick

891 rows × 2 columns

In [9]:
```
(df.groupby('Title').mean()['Survived']).sort_values(ascending=False)
```

Out[9]:
```
Title
the Countess    1.000000
Mlle            1.000000
Sir             1.000000
Ms              1.000000
Lady            1.000000
Mme             1.000000
Mrs             0.792000
Miss            0.697802
Master          0.575000
Col             0.500000
Major           0.500000
Dr              0.428571
```

```
Mr              0.156673
Jonkheer        0.000000
Rev             0.000000
Don             0.000000
Capt            0.000000
Name: Survived, dtype: float64
```

In [10]:

```
df['Is_Married'] = 0
df['Is_Married'].loc[df['Title'] == 'Mrs'] = 1
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1732: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

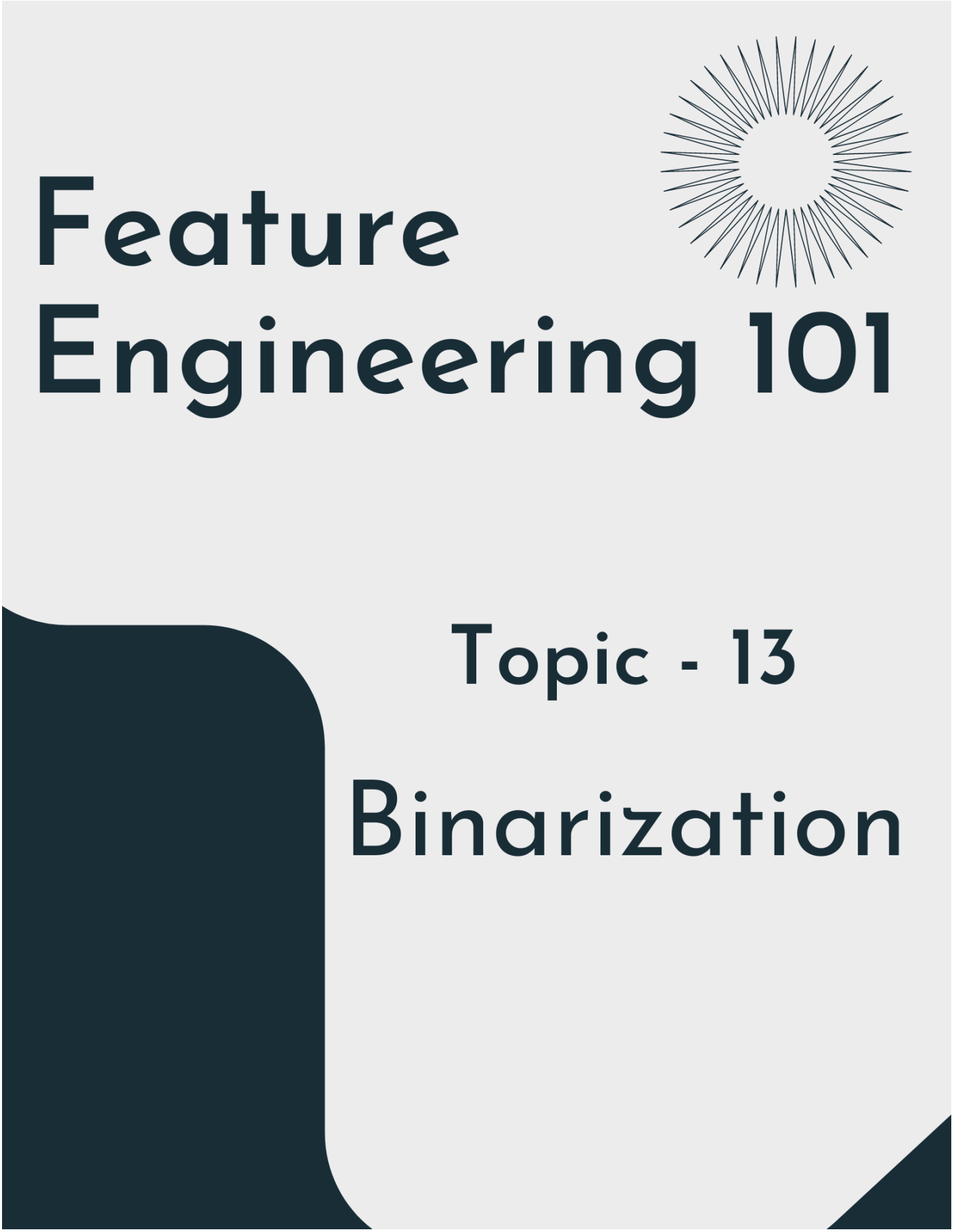
In [11]:

```
df['Is_Married']
```

Out[11]:

```
0      0
1      1
2      0
3      1
4      0
      ..
886    0
887    0
888    0
889    0
890    0
Name: Is_Married, Length: 891, dtype: int64
```

In [ ]:



Binarization

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

In [2]:
```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score

from sklearn.preprocessing import KBinsDiscretizer
from sklearn.compose import ColumnTransformer
```

In [3]:
```
df = pd.read_csv('train.csv',usecols=['Age','Fare','Survived'])
```

In [4]:
```
df.dropna(inplace=True)
```

In [5]:
```
df.shape
```

Out[5]:
```
(714, 3)
```

In [6]:
```
df.head()
```

Out[6]:

	Survived	Age	Fare
0	0	22.0	7.2500
1	1	38.0	71.2833
2	1	26.0	7.9250
3	1	35.0	53.1000
4	0	35.0	8.0500

In [7]:
```
X = df.iloc[:,1:]
y = df.iloc[:,0]
```

In [8]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [9]:
```
X_train.head(2)
```

Out[9]:

	Age	Fare
328	31.0	20.5250
73	26.0	14.4542

In [10]:
```
clf = DecisionTreeClassifier()
```

In [11]:
```
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

In [12]:
```
accuracy_score(y_test,y_pred)
```

Out[12]:
```
0.6433566433566433
```

In [13]:
```
np.mean(cross_val_score(DecisionTreeClassifier(),X,y,cv=10,scoring='accuracy'))
```

Out[13]:
```
0.6289319248826291
```

In [14]:
```
kbin_age = KBinsDiscretizer(n_bins=15,encode='ordinal',strategy='quantile')
kbin_fare = KBinsDiscretizer(n_bins=15,encode='ordinal',strategy='quantile')
```

In [15]:
```
trf = ColumnTransformer([
    ('first',kbin_age,[0]),
    ('second',kbin_fare,[1])
])
```

In [16]:
```
X_train_trf = trf.fit_transform(X_train)
X_test_trf = trf.transform(X_test)
```

In [17]:
```
trf.named_transformers_['first'].bin_edges_
```

Out[17]:
```
array([array([ 0.42,  6.  , 16.  , 19.  , 21.  , 23.  , 25.  , 28.  , 30.  ,
              32.  , 35.  , 38.  , 42.  , 47.  , 54.  , 80.  ])             ],
      dtype=object)
```

In [18]:
```
trf.named_transformers_['first'].bin_edges_
```

Out[18]:
```
array([array([ 0.42,  6.  , 16.  , 19.  , 21.  , 23.  , 25.  , 28.  , 30.  ,
              32.  , 35.  , 38.  , 42.  , 47.  , 54.  , 80.  ])             ],
      dtype=object)
```

In [19]:
```
output = pd.DataFrame({
    'age':X_train['Age'],
    'age_trf':X_train_trf[:,0],
    'fare':X_train['Fare'],
    'fare_trf':X_train_trf[:,1]
})
```

In [20]:
```
output['age_labels'] = pd.cut(x=X_train['Age'],
                                        bins=trf.named_transformers_['first'].bin_edges_[0].to
output['fare_labels'] = pd.cut(x=X_train['Fare'],
                                        bins=trf.named_transformers_['second'].bin_edges_[0].t
```

In [21]:
```
output.sample(5)
```

Out[21]:

	age	age_trf	fare	fare_trf	age_labels	fare_labels
821	27.0	6.0	8.6625	4.0	(25.0, 28.0]	(8.158, 10.5]
230	35.0	10.0	83.4750	13.0	(32.0, 35.0]	(76.292, 108.9]
784	25.0	6.0	7.0500	0.0	(23.0, 25.0]	(0.0, 7.25]
660	50.0	13.0	133.6500	14.0	(47.0, 54.0]	(108.9, 512.329]

	age	age_trf	fare	fare_trf	age_labels	fare_labels
621	42.0	12.0	52.5542	12.0	(38.0, 42.0]	(51.479, 76.292]

In [22]:
```
clf = DecisionTreeClassifier()
clf.fit(X_train_trf,y_train)
y_pred2 = clf.predict(X_test_trf)
```

In [23]:
```
accuracy_score(y_test,y_pred2)
```

Out[23]:
```
0.6363636363636364
```

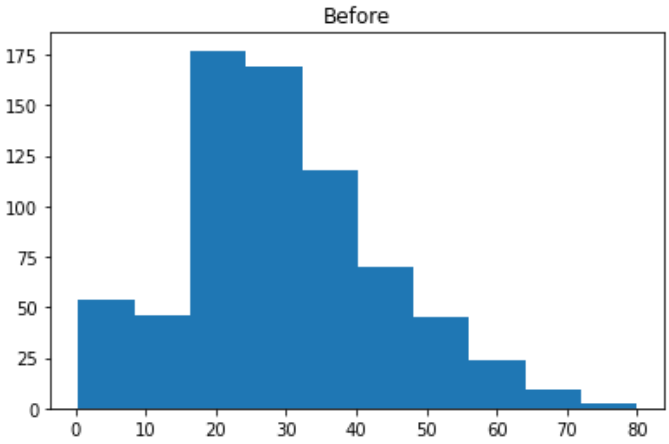
In [24]:
```
X_trf = trf.fit_transform(X)
np.mean(cross_val_score(DecisionTreeClassifier(),X,y,cv=10,scoring='accuracy'))
```

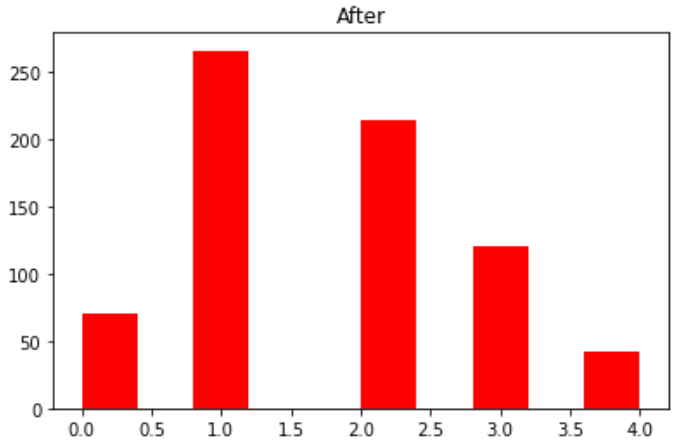
Out[24]:
```
0.6303012519561815
```

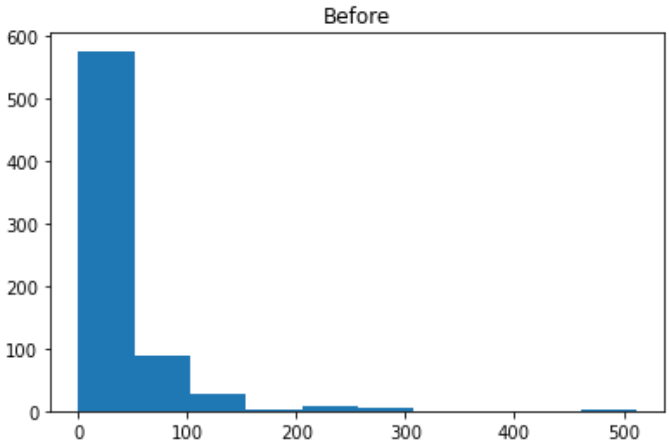
In [25]:
```
def discretize(bins,strategy):
    kbin_age = KBinsDiscretizer(n_bins=bins,encode='ordinal',strategy=strategy)
    kbin_fare = KBinsDiscretizer(n_bins=bins,encode='ordinal',strategy=strategy)

    trf = ColumnTransformer([
        ('first',kbin_age,[0]),
        ('second',kbin_fare,[1])
    ])

    X_trf = trf.fit_transform(X)
    print(np.mean(cross_val_score(DecisionTreeClassifier(),X,y,cv=10,scoring='accuracy')))

    plt.figure(figsize=(14,4))
    plt.subplot(121)
    plt.hist(X['Age'])
    plt.title("Before")

    plt.subplot(122)
    plt.hist(X_trf[:,0],color='red')
    plt.title("After")

    plt.show()

    plt.figure(figsize=(14,4))
    plt.subplot(121)
    plt.hist(X['Fare'])
    plt.title("Before")

    plt.subplot(122)
    plt.hist(X_trf[:,1],color='red')
    plt.title("Fare")

    plt.show()
```

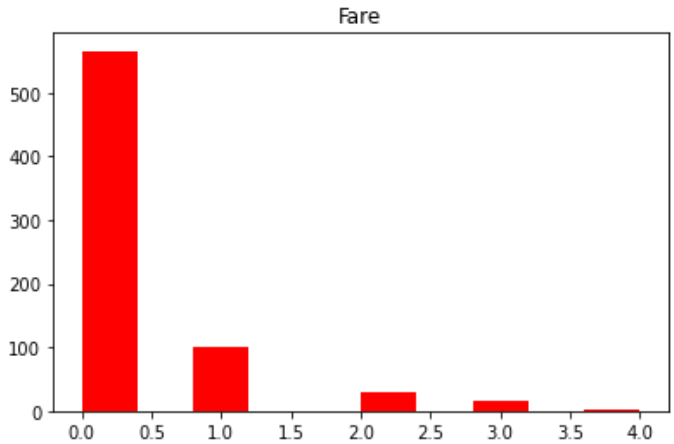
In [26]:
```
discretize(5,'kmeans')
```

```
0.6288928012519561
```


Before
175
150
125
100
75
50
25
0
0
10
20
30
40
50
60
70
80


After
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Before
600
500
400
300
200
100
0
0
100
200
300
400
500


Fare
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

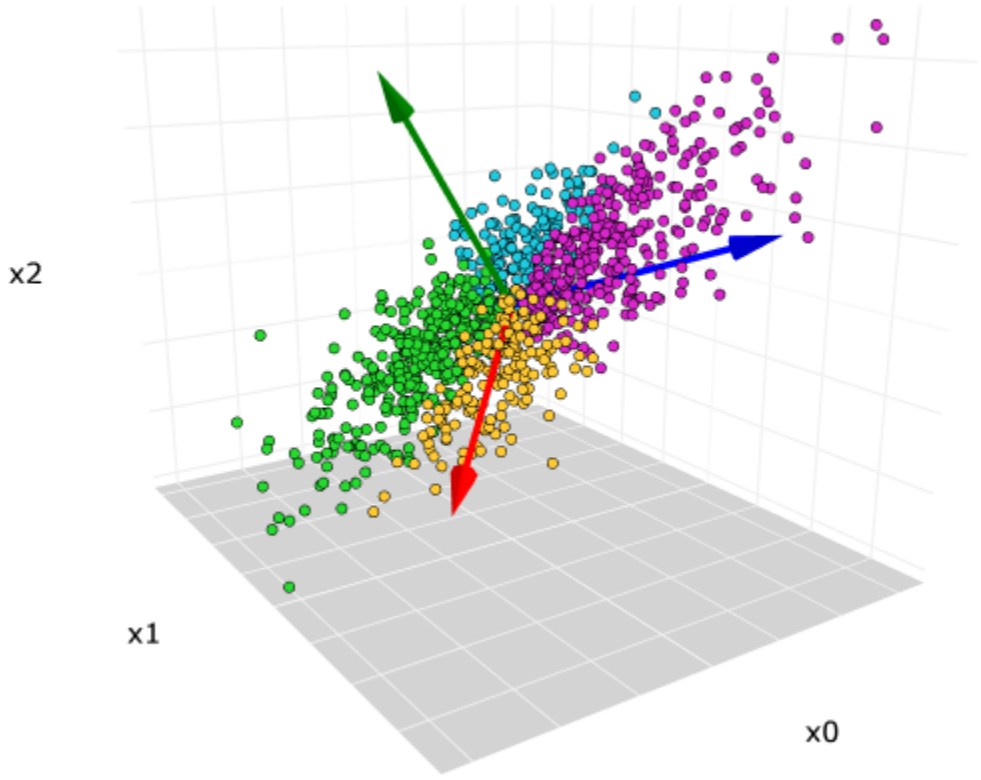
In [ ]:

Feature Engineering 101

Topic - 14

PCA

Principal Component Analysis



PCA (Principal Component Analysis)

In [1]:

```
import numpy as np
import pandas as pd

np.random.seed(23)

mu_vec1 = np.array([0,0,0])
cov_mat1 = np.array([[1,0,0],[0,1,0],[0,0,1]])
class1_sample = np.random.multivariate_normal(mu_vec1, cov_mat1, 20)

df = pd.DataFrame(class1_sample,columns=['feature1','feature2','feature3'])
df['target'] = 1

mu_vec2 = np.array([1,1,1])
cov_mat2 = np.array([[1,0,0],[0,1,0],[0,0,1]])
class2_sample = np.random.multivariate_normal(mu_vec2, cov_mat2, 20)

df1 = pd.DataFrame(class2_sample,columns=['feature1','feature2','feature3'])

df1['target'] = 0

df = df.append(df1,ignore_index=True)

df = df.sample(40)
```

In [2]:

```
df.head()
```

Out[2]:

	feature1	feature2	feature3	target
2	-0.367548	-1.137460	-1.322148	1

	feature1	feature2	feature3	target
34	0.177061	-0.598109	1.226512	0
14	0.420623	0.411620	-0.071324	1
11	1.968435	-0.547788	-0.679418	1
12	-2.506230	0.146960	0.606195	1

In [3]:

```
import plotly.express as px
#y_train_trf = y_train.astype(str)
fig = px.scatter_3d(df, x=df['feature1'], y=df['feature2'], z=df['feature3'],
                    color=df['target'].astype('str'))
fig.update_traces(marker=dict(size=12,
                              line=dict(width=2,
                                        color='DarkSlateGrey')),
                  selector=dict(mode='markers'))

fig.show()
```

In [4]:

```
# Step 1 - Apply standard scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

df.iloc[:,0:3] = scaler.fit_transform(df.iloc[:,0:3])
```

In [5]:

```
# Step 2 - Find Covariance Matrix
covariance_matrix = np.cov([df.iloc[:,0],df.iloc[:,1],df.iloc[:,2]])
print('Covariance Matrix:\n', covariance_matrix)
```

```
Covariance Matrix:
 [[1.02564103 0.20478114 0.080118  ]
 [0.20478114 1.02564103 0.19838882]
 [0.080118   0.19838882 1.02564103]]
```

In [6]:
```
# Step 3 - Finding EV and EVs
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
```

In [7]:
```
eigen_values
```

Out[7]:
```
array([1.3536065 , 0.94557084, 0.77774573])
```

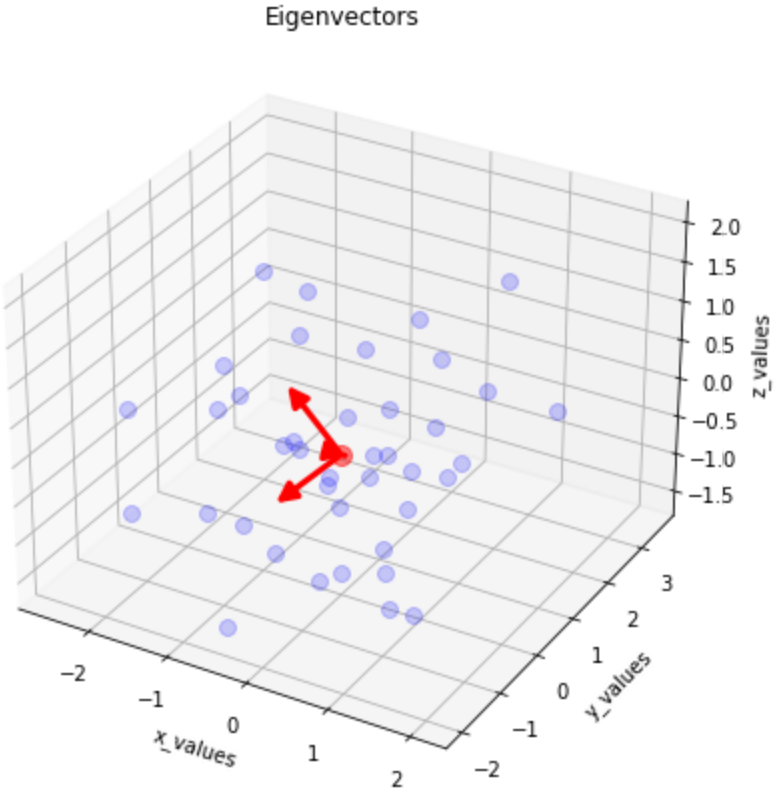
In [8]:
```
eigen_vectors
```

Out[8]:
```
array([[-0.53875915, -0.69363291,  0.47813384],
       [-0.65608325, -0.01057596, -0.75461442],
       [-0.52848211,  0.72025103,  0.44938304]])
```

In [9]:
```
%pylab inline

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d
from matplotlib.patches import FancyArrowPatch


class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)

fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(111, projection='3d')

ax.plot(df['feature1'], df['feature2'], df['feature3'], 'o', markersize=8, color='blue', a
ax.plot([df['feature1'].mean()], [df['feature2'].mean()], [df['feature3'].mean()], 'o', ma
for v in eigen_vectors.T:
    a = Arrow3D([df['feature1'].mean(), v[0]], [df['feature2'].mean(), v[1]], [df['feature
    ax.add_artist(a)
ax.set_xlabel('x_values')
ax.set_ylabel('y_values')
ax.set_zlabel('z_values')

plt.title('Eigenvectors')

plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
C:\Users\HP\AppData\Local\Temp/ipykernel_88008/3713440988.py:16: MatplotlibDeprecationWarn
ing:
```

```
The M attribute was deprecated in Matplotlib 3.4 and will be removed two minor releases la
ter. Use self.axes.M instead.
```



In [10]:
```
pc = eigen_vectors[0:2]
pc
```

Out[10]:
```
array([[-0.53875915, -0.69363291,  0.47813384],
       [-0.65608325, -0.01057596, -0.75461442]])
```

In [11]:
```
transformed_df = np.dot(df.iloc[:,0:3],pc.T)
# 40,3 - 3,2
new_df = pd.DataFrame(transformed_df,columns=['PC1','PC2'])
new_df['target'] = df['target'].values
new_df.head()
```

Out[11]:

	PC1	PC2	target
0	0.599433	1.795862	1
1	1.056919	-0.212737	0
2	-0.271876	0.498222	1
3	-0.621586	0.023110	1
4	1.567286	1.730967	1

In [12]:
```
new_df['target'] = new_df['target'].astype('str')
fig = px.scatter(x=new_df['PC1'],
                 y=new_df['PC2'],
                 color=new_df['target'],
                 color_discrete_sequence=px.colors.qualitative.G10
                )

fig.update_traces(marker=dict(size=12,
```