In [7]:
```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:
```
df=pd.read_csv("f:/dataset/classification/online_shop.csv")
df
```

Out[2]:

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

400 rows × 4 columns

In [3]:
```
X=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

In [8]:
```
model=KNeighborsClassifier()
model.fit(X,y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[8], line 2
      1 model=KNeighborsClassifier()
----> 2 model.fit(X,y)

File ~\anaconda3\Lib\site-packages\sklearn\neighbors\_classification.py:215, in KNeighbo
rsClassifier.fit(self, X, y)
    196 """Fit the k-nearest neighbors classifier from the training dataset.
    197
    198 Parameters
   (...)
    211     The fitted k-nearest neighbors classifier.
    212 """
    213 self._validate_params()
--> 215 return self._fit(X, y)

File ~\anaconda3\Lib\site-packages\sklearn\neighbors\_base.py:454, in NeighborsBase._fit
(self, X, y)
    452 if self._get_tags()["requires_y"]:
    453     if not isinstance(X, (KDTree, BallTree, NeighborsBase)):
--> 454         X, y = self._validate_data(
    455             X, y, accept_sparse="csr", multi_output=True, order="C"
    456         )
    458     if is_classifier(self):
    459         # Classification targets require a specific format
```

```
    460         if y.ndim == 1 or y.ndim == 2 and y.shape[1] == 1:

File ~\anaconda3\Lib\site-packages\sklearn\base.py:584, in BaseEstimator._validate_data
(self, X, y, reset, validate_separately, **check_params)
    582         y = check_array(y, input_name="y", **check_y_params)
    583     else:
--> 584         X, y = check_X_y(X, y, **check_params)
    585     out = X, y
    587 if not no_val_X and check_params.get("ensure_2d", True):

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1106, in check_X_y(X, y,
 accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, al
low_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
   1101         estimator_name = _check_estimator_name(estimator)
   1102     raise ValueError(
   1103         f"{estimator_name} requires y to be passed, but the target y is None"
   1104     )
-> 1106 X = check_array(
   1107     X,
   1108     accept_sparse=accept_sparse,
   1109     accept_large_sparse=accept_large_sparse,
   1110     dtype=dtype,
   1111     order=order,
   1112     copy=copy,
   1113     force_all_finite=force_all_finite,
   1114     ensure_2d=ensure_2d,
   1115     allow_nd=allow_nd,
   1116     ensure_min_samples=ensure_min_samples,
   1117     ensure_min_features=ensure_min_features,
   1118     estimator=estimator,
   1119     input_name="X",
   1120 )
   1122 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator=estima
tor)
   1124 check_consistent_length(X, y)

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:879, in check_array(arra
y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d,
 allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    877         array = xp.astype(array, dtype, copy=False)
    878     else:
--> 879         array = _asarray_with_order(array, order=order, dtype=dtype, xp=xp)
    880 except ComplexWarning as complex_warning:
    881     raise ValueError(
    882         "Complex data not supported\n{}\n".format(array)
    883     ) from complex_warning

File ~\anaconda3\Lib\site-packages\sklearn\utils\_array_api.py:185, in _asarray_with_ord
er(array, dtype, order, copy, xp)
    182     xp, _ = get_namespace(array)
    183 if xp.__name__ in {"numpy", "numpy.array_api"}:
    184     # Use NumPy API to support order
--> 185     array = numpy.asarray(array, order=order, dtype=dtype)
    186     return xp.asarray(array, copy=copy)
    187 else:

ValueError: could not convert string to float: 'Male'
```

Feature Encoding

- converting text feature into numeric form
- 2 techniques
  - label encoding

- one hot encoding

Label encoding

- using pandas
- using sklearn

In [9]:
```
#using pandas
df
```

Out[9]:

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

400 rows × 4 columns

In [11]:
```
df.Gender=df.Gender.map({'Female':0,'Male':1})
```

In [12]:
```
df
```

Out[12]:

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

400 rows × 4 columns

In [13]:
```
df=pd.read_csv("f:/dataset/classification/online_shop.csv")
df
```

Out[13]:

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

400 rows × 4 columns

In [14]:
```
#using sklearn
from sklearn.preprocessing import LabelEncoder
en=LabelEncoder()
df.Gender=en.fit_transform(df.Gender)
```

In [19]:
```
df
```

Out[19]:

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

400 rows × 4 columns

In [20]:
```
df=pd.read_csv("f:/dataset/classification/online_shop.csv")
df
```

Out[20]:

	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0

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

400 rows × 4 columns

In [25]:
```
#onehot encoding using pandas
df2=pd.get_dummies(df)
df2
```

Out[25]:

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

400 rows × 5 columns

In [43]:
```
#onehot encoding using sklearn
from sklearn.preprocessing import OneHotEncoder
en=OneHotEncoder()
en.fit_transform(df[['Gender']]).toarray()
```

Out[43]:
```
array([[0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
```

```
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
```

```
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
```

```
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
```

```
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
[1., 0.],
[0., 1.],
[1., 0.],
[1., 0.],
[0., 1.],
[0., 1.],
```

```
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
```

```
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

In [47]:
```
en=OneHotEncoder()
m=en.fit_transform(df[['Gender']])
print(m)
```

```
  (0, 1)        1.0
```

```
(1, 1)        1.0
(2, 0)        1.0
(3, 0)        1.0
(4, 1)        1.0
(5, 1)        1.0
(6, 0)        1.0
(7, 0)        1.0
(8, 1)        1.0
(9, 0)        1.0
(10, 0)       1.0
(11, 0)       1.0
(12, 1)       1.0
(13, 1)       1.0
(14, 1)       1.0
(15, 1)       1.0
(16, 1)       1.0
(17, 1)       1.0
(18, 1)       1.0
(19, 0)       1.0
(20, 1)       1.0
(21, 0)       1.0
(22, 1)       1.0
(23, 0)       1.0
(24, 1)       1.0
:     :
(375, 0)      1.0
(376, 0)      1.0
(377, 0)      1.0
(378, 1)      1.0
(379, 0)      1.0
(380, 1)      1.0
(381, 1)      1.0
(382, 0)      1.0
(383, 1)      1.0
(384, 0)      1.0
(385, 1)      1.0
(386, 0)      1.0
(387, 1)      1.0
(388, 1)      1.0
(389, 0)      1.0
(390, 1)      1.0
(391, 1)      1.0
(392, 0)      1.0
(393, 1)      1.0
(394, 0)      1.0
(395, 0)      1.0
(396, 1)      1.0
(397, 0)      1.0
(398, 1)      1.0
(399, 0)      1.0
```

In [48]:

```
en=OneHotEncoder(sparse_output=False)
m=en.fit_transform(df[['Gender']])
print(m)
```

```
[[0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
```

```
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
```

```
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
```

```
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
```

```
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
```

```
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
```

```
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]]
```

In [49]:
```
import sklearn
sklearn.__version__
```

Out[49]:
```
'1.2.2'
```

In [50]:
```
import pandas
pandas.__version__
```

Out[50]:
```
'1.5.3'
```

In [51]:
```
df=pd.read_csv("f:/dataset/classification/online_shop.csv")
en=LabelEncoder()
df.Gender=en.fit_transform(df.Gender)
```

In [52]:
```
df
```

Out[52]:

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

400 rows × 4 columns

In [53]:
```
X=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

In [56]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [55]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=1)
```

In [57]:
```
model=KNeighborsClassifier()
model.fit(X_train,y_train)
pred_train=model.predict(X_train)
pred_test=model.predict(X_test)
print("Train Score:",accuracy_score(y_train,pred_train))
print("Test Score:",accuracy_score(y_test,pred_test))
```

```
Train Score: 0.8966666666666666
Test Score: 0.75
```

In [69]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train_new=sc.fit_transform(X_train)
X_test_new=sc.transform(X_test)
model=KNeighborsClassifier(n_neighbors=7)
model.fit(X_train_new,y_train)
pred_train=model.predict(X_train_new)
pred_test=model.predict(X_test_new)
print("Train Score:",accuracy_score(y_train,pred_train))
print("Test Score:",accuracy_score(y_test,pred_test))
```

```
Train Score: 0.9266666666666666
Test Score: 0.89
```

In [ ]:

```
#new sample
g=
a=
s=

label encoding
scaling
pred
```