

PROGRAM:

In [1]: import pandas as pd

In [2]: df=pd.read_csv("student.csv")

In [3]: df

In [6]: df.head(2)

In [7]: df.tail(2)

OUTPUT:

Out [3]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	NaN	2.0	1
2	102	NaN	65.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	NaN	0

Out [6]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	NaN	2.0	1

Out [7]: Reg. no. M1 M2 M3 result

4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	NaN	0

PROGRAM:

In [1]: import pandas as pd

In [2]: df=pd.read_csv("student.csv")

In [3]: df

In [6]: df.describe()

In [7]: df.info()

In [8]: df.isnull().sum()

In [9]: df.dtypes

In [10]: df.shape

In [11]: df1=df.fillna("n")

In [12]: df1

In [13]: df2=df.fillna(5)

In [14]: df2

dictionary

In [15]: df1=df.fillna({'M1':'chol','M2':'fbs', 'M3':2})

In [16]: df1.isnull().sum()

df1

Carry forward

In [17]: df1=df.fillna(method="ffill")

df1

Backward fill

In [18]: df1=df.fillna(method="bfill")

df1

```
# Fill avg value
```

```
In [19]: df1=df.interpolate()
```

```
df1
```

```
# Drop NA row/column
```

```
In [20]: df1=df.dropna()
```

```
df1
```

OUTPUT:

Out [3]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	NaN	2.0	1
2	102	NaN	65.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	NaN	0

Out [6]: Reg. no. M1 M2 M3 result

count	6.000000	5.000000	5.000000	5.000000	6.000000
mean	102.500000	292.000000	235.000000	47.400000	0.500000
std	1.870829	331.910379	419.85295	29.12559	0.547723
min	100.000000	1.000000	5.000000	2.000000	0.000000
25%	101.250000	67.000000	34.000000	45.000000	0.000000
50%	102.500000	84.000000	65.000000	45.000000	0.500000
75%	103.750000	654.000000	87.000000	67.000000	1.000000
max	105.000000	654.000000	984.000000	78.000000	1.000000

Out [7]:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 6 entries, 0 to 5

Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	Reg. no.	6 non-null	int64
1	M1	5 non-null	float64
2	M2	5 non-null	float64
3	M3	5 non-null	float64
4	result	6 non-null	int64

Out [8]: Reg. no. 0

```
M1      1  
M2      1  
M3      1  
result   0  
dtype: int64
```

Out [10]: (6, 5)

Out [12]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	n	2.0	1
2	102	n	65.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	n	0

Out [14]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	5.0	2.0	1
2	102	5.0	65.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	5.0	0

Out [16]: Reg. no. 0

```
M1      0  
M2      0  
M3      0  
result   0  
dtype: int64
```

Out [17]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	984.0	2.0	1
2	102	654.0	65.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	78.0	0

Out [18]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	65.0	2.0	1
2	102	84.0	65.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	NaN	0

Out [19]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
1	101	654.0	524.5	2.0	1
2	102	369.0	65.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1
5	105	67.0	34.0	78.0	0

Out [20]: Reg. no. M1 M2 M3 result

0	100	654.0	984.0	45.0	0
3	103	84.0	87.0	67.0	1
4	104	1.0	5.0	78.0	1

PROGRAM:

In [1]: import pandas as pd

In [2]: data = pd.DataFrame({"A": [1,2,3,4,5,6],
"B": [7,8,9,10,11,12],
"C": [0,0,0,0,0,0],
"D": [21,54,32,85,55,2]})

In [13]: data

In [4]: from sklearn.feature_selection import VarianceThreshold

In [5]: var_thres = VarianceThreshold(threshold = 0)

In [6]: var_thres.fit(data)

In [7]: var_thres.get_support()

In [8]: data.columns[var_thres.get_support()]

In [9]: constant_columns = [column for column in data.columns if column not in data.columns[var_thres.get_support()]]

In [10]: print(len(constant_columns))

1

In [11]: for feature in constant_columns: print(feature)

C

In [12]: data.drop(constant_columns, axis = 1)

OUTPUT:

Out [13]: A B C D

```
0 1 7 0 21
1 2 8 0 54
2 3 9 0 32
3 4 10 0 85
4 5 11 0 35
5 6 12 0 2
```

Out [6]: VarianceThreshold(threshold = 0)

Out [7]: array([True, True, False, True])

Out [8]: Index(['A', 'B', 'D'], dtype='object')

Out[12]: A B D

```
0 1 7 21
1 2 8 54
2 3 9 32
3 4 10 85
4 5 11 35
5 6 12 2
```

PROGRAM:

In [1]: import pandas as pd

In [1]: df=pd.read_csv("diabetes.csv")

df

In [2]: from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

In [4]: df['Outcome'].value_counts()

In [5]: X = df.drop(columns=['Outcome'], axis=1)

Y = df['Outcome']

In [6]: print(X)

	Pregnancies	Glucose	BP	SkinThickness	Insulin	BMI	DPF	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

[768 rows x 8 columns]

In [7]: print(Y)

0 1

1 0

2 1

3 0

4 1

...

```
763 0  
764 0  
765 0  
766 1  
767 0
```

Name: Outcome, Length: 768, dtype: int64

```
In [8]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y,  
random_state=2)
```

```
In [9]: print(X.shape, X_train.shape, X_test.shape)
```

(768, 8) (614, 8) (154, 8)

```
In [10]: model = LogisticRegression(max_iter=1000)
```

```
In [11]: model.fit(X_train, Y_train)
```

```
In [12]: from sklearn.metrics import accuracy_score
```

```
In [13]: X_train_prediction = model.predict(X_train)
```

```
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

```
print(training_data_accuracy)
```

0.7882736156351792

```
In [14]: print('Accuracy on Training data :', round(training_data_accuracy*100, 2), '%')
```

Accuracy on Training data : 78.83 %

```
In [15]: X_test_prediction = model.predict(X_test)
```

```
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
```

```
print(test_data_accuracy)
```

0.7597402597402597

```
In [16]: print('Accuracy on Training data :', round(test_data_accuracy*100, 2), '%')
```

Accuracy on Training data : 75.97 %

```
In [15]: X_test_prediction = model.predict(X_test)
```

```
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
```

```
print(test_data_accuracy)
```

0.7597402597402597

```
In [16]: print('Accuracy on Training data :', round(test_data_accuracy*100, 2), '%')
```

Accuracy on Training data : 75.97 %

```
In [20]: import seaborn as sns
```

```
    sns.heatmap(cf_matrix, annot=True)
```

```
In [21]: from sklearn.metrics import precision_score
```

```
    precision_train = precision_score(Y_train, X_train_prediction)
```

```
    print(precision_train)
```

```
0.7530120481927711
```

```
In [22]: precision_test = precision_score(Y_test, X_test_prediction)
```

```
    print('test data Precision = ', precision_test)
```

```
    print(test data Precision)
```

```
0.717948717948718
```

```
In [23]: from sklearn.metrics import recall_score
```

```
    recall_train = recall_score(Y_train, X_train_prediction)
```

```
    print('training data Recall = ', recall_train)
```

```
0.5841121495327103
```

```
In [24]: recall_test = recall_score(Y_test, X_test_prediction)
```

```
    print('test data Recall = ', recall_test)
```

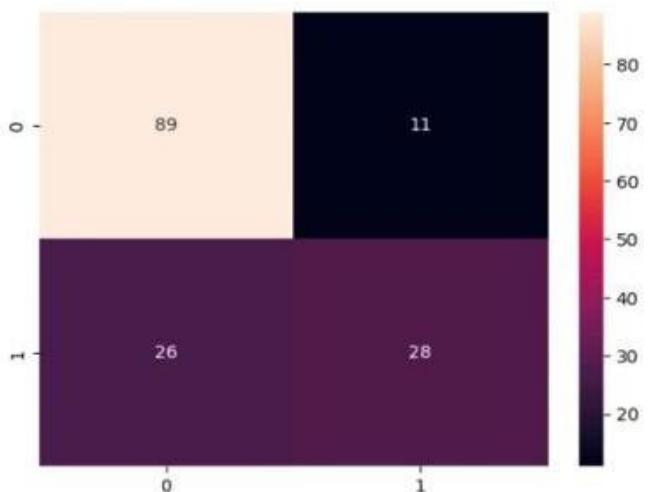
```
0.5185185185185185
```

```
In [25]: from sklearn.metrics import f1_score
```

```
    f1_score_train = f1_score(Y_train, X_train_prediction)
```

```
    print('training data F1 Score = ', f1_score_train)
```

```
0.6589473684210526
```



```
In [26]: f1_score_test = f1_score(Y_test, X_test_prediction)
```

```
print('Test data F1 Score = ', f1_score_test)
```

```
0.6021505376344086
```

```
In [27]: def precision_recall_f1_score(true_labels, pred_labels):
```

```
    precision_value = precision_score(true_labels, pred_labels)
```

```
    recall_value = recall_score(true_labels, pred_labels)
```

```
    f1_score_value = f1_score(true_labels, pred_labels)
```

```
    print('Precision - ', precision_value)
```

```
    print('Recall - ', recall_value)
```

```
    print('F1 Score - ', f1_score_value)
```

```
In [28]: precision_recall_f1_score(Y_train, X_train_prediction)
```

```
Precision = 0.7530120481927711
```

```
Recall = 0.5841121495327103
```

```
F1 Score = 0.6589473684210526
```

```
In [29]: precision_recall_f1_score(Y_test, X_test_prediction)
```

```
Precision = 0.717948717948718
```

```
Recall = 0.5185185185185185
```

```
F1 Score = 0.6021505376344086
```

OUTPUT:

Out [1]:

	Pregnancies	Glucose	BP	SkinThickness	Insulin	BMI	DPF	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	1
...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

[768 rows x 9 columns]

Out [4]:

0	500
1	268

Name: Outcome, dtype: int64

Out [11]: LogisticRegression(max_iter=1000)

Out [20]: <AxesSubplot: >

PROGRAM:

In [1]: import pandas as pd

In [24]: df = pd.read_csv("titanic.csv")

df.head()

In [25]: df.drop(['PassengerId', 'Name', 'SibSp', 'Parch', 'Ticket', 'Cabin', 'Embarked'],

axis='columns', inplace=True)

df.head()

In [26]: inputs = df.drop('Survived', axis='columns')

target = df.Survived

In [27]: dummies = pd.get_dummies(inputs.Sex)

dummies.head()

In [28]: inputs = pd.concat([inputs, dummies], axis='columns')

inputs.head(3)

In [29]: inputs.drop(['Sex', 'male'], axis='columns', inplace=True)

inputs.head(3)

In [30]: inputs.columns[inputs.isna().any()]

In [10]: inputs.Age[:10]

In [31]: inputs.Age = inputs.Age.fillna(inputs.Age.mean())

inputs.head()

In [12]: from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(inputs, target, test_size=0.3)

In [13]: from sklearn.naive_bayes import GaussianNB

model = GaussianNB()

In [14]: model.fit(X_train, Y_train)

In [15]: model.score(X_test, Y_test)

```
In [16]: X_test[:10]
```

```
In [17]: y_test[:10]
```

```
In [18]: model.predict(X_test[:10])
```

```
In [19]: model.predict_proba(X_test[:10])  
In [34]: from sklearn.model_selection import cross_val_score  
cross_val_score(GaussianNB(), X_train, Y_train, cv=5)
```

OUTPUT:

Out [24]:

	PassengerId	Name	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
0	1	Mr.Owen	3	male	22.0	1	0	A/521171	7.2500	NaN	S	0
1	2	Mrs. John	1	female	38.0	1	0	PC17599	71.2833	C85	C	1
2	3	Miss. Laina	3	female	26.0	0	0	ST/O2.322	7.9250	NaN	S	1
3	4	Mrs. Jacques	1	female	35.0	1	0	113803	53.1000	C123	S	1
4	5	Mr. William	3	male	35.0	0	0	373450	8.0500	NaN	S	0

Out [25]: Pclass Sex Age Fare Survived

0	3	male	22.0	7.2500	0
1	1	female	38.0	71.2833	1
2	3	female	26.0	7.9250	1
3	1	female	35.0	53.1000	1
4	3	male	35.0	8.0500	0

Out [27]: female male

0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

Out [28]: Pclass Sex Age Fare female male

0	3	male	22.0	7.2500	0	1
1	1	female	38.0	71.2833	1	0
2	3	female	26.0	7.9250	1	0

Out [29]: Pclass Age Fare female

0	3	22.0	7.2500	0
1	1	38.0	71.2833	1
2	3	26.0	7.9250	1

Out [30]:

Index(['Age'],dtype='object')

Out [10]: 0 22.0

1 38.0
2 26.0
3 35.0
4 35.0
5 NaN
6 54.0
7 2.0
8 27.0
9 14.0

Name: Age, dtype: float64

Out [31]:

	Pclass	Age	Fare	female
--	--------	-----	------	--------

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

Out [14]: GaussianNB(priors=None,var_smoothing=1e-09)

Out [15]: 0.7835820895522388

Out [16]:

	Pclass	Age	Fare	female
--	--------	-----	------	--------

309	1	30.0000	56.9292	1
839	1	29.6991	29.7000	0
872	1	33.0000	5.0000	0
235	3	29.6991	7.5500	1
411	3	29.6991	6.8583	1
32	3	29.6991	7.7500	1
562	2	28.0000	13.5000	0

```
193    2    3.0000  26.0000  0
832    3   29.6991  7.2292  0
250    3   29.6991  7.2500  0
```

Out [17]: 309 1

```
839  1
872  0
235  0
411  0
32   1
562  0
193  1
832  0
250  0
```

Name: survived, dtype: int64

Out [18]: array([1, 0, 0, 0, 1, 0, 1, 0, 0, 0], dtype=int64)

Out [19]: array([[0.0045592, 0.9954408,

```
[0.8138202, 0.1861797],
[0.8653257, 0.1346742],
[0.9234707, 0.0765292],
[0.1018866, 0.8981134],
[0.9903305, 0.0096695],
[0.9984926, 0.0015073],
[0.9792378, 0.0207621],
[0.8516962, 0.0483037],
[0.9631673, 0.0368326]])
```

Out [34]: array([0.75396825, 0.784, 0.76612903, 0.82258065, 0.77419355])

PROGRAM:

In [1]: import pandas as pd

```
df = pd.read_csv("spam.csv")
df.head()
```

In [2]: df.groupby('Category').describe()

In [3]: df['spam']=df['Category'].apply(lambda x: 1 if x=='spam' else 0)
df.head()

In [4]: from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(df.Message, df.spam, test_size=0.25)
```

In [5]: from sklearn.feature_extraction.text import CountVectorizer

```
v = CountVectorizer()
X_train_count = v.fit_transform(X_train.values)
X_train_count.toarray()[:2]
```

In [6]: from sklearn.naive_bayes import MultinomialNB

```
model = MultinomialNB()
model.fit(X_train_count, y_train)
```

In [8]: X_test_count = v.transform(X_test)

```
model.score(X_test_count, y_test)
```

In [9]: from sklearn.pipeline import Pipeline

```
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])
```

In [10]: clf.fit(X_train, y_train)

In [11]: clf.score(X_test, y_test)

In [12]: clf.predict(emails)

OUTPUT:

```
Out [1]:      Category          Message
            0    ham    Go until jurong point, crazy. Available only ...
            1    ham          Ok lar... Joking wif u oni...
            2   spam  Free entry in 2 a wkly comp to win FA Cup fina...
            3    ham    U dun say so early hor... U c already then say...
            4    ham    Nah I don't think he goes to usf, he lives aro...
```

```
Out [2]:          Message
                  count  unique          top      freq
Category
    ham    4825    4516  Sorry, I'll call later  30
    spam     747    641  Please call our customer service representativ...  4
```

```
Out [3]:      Category          Message      spam
            0    ham    Go until jurong point, crazy. Available only ...  0
            1    ham          Ok lar... Joking wif u oni...  0
            2   spam  Free entry in 2 a wkly comp to win FA Cup fina...  1
            3    ham    U dun say so early hor... U c already then say...  0
            4    ham    Nah I don't think he goes to usf, he lives aro...  0
```

```
Out [5]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
Out [6]: MultinomialNB()
```

```
Out [8]: 0.9877961234745154
```

```
Out [10]: Pipeline(steps=[('vectorizer', CountVectorizer()), ('nb', MultinomialNB())])
```

```
Out [11]: 0.9877961234745154
```

```
Out [12]: array([0, 1], dtype=int64)
```

PROGRAM:

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

```
    import numpy as np
```

```
    import pgmpy
```

```
In [2]: from pgmpy.estimators import MaximumLikelihoodEstimator
```

```
    from pgmpy.models import BayesianModel
```

```
    from pgmpy.inference import VariableElimination
```

```
In [3]: hd = pd.read_csv('heart.csv')
```

```
hd = hd.replace('?', np.nan)
```

```
In [9]: model = BayesianModel([('age', 'target'), ('sex', 'target'), ('trestbps', 'target'), ('cp', 'target'),  
    ('target', 'restecg'), ('target', 'chol'), ('target', 'fbs'), ('target', 'thalach'), ('target', 'exang'),  
    ('target', 'oldpeak'), ('target', 'slope'), ('target', 'ca'), ('target', 'thal')])
```

```
In [5]: print('Learning CPD using Maximum likelihood estimators')
```

```
model.fit(hd, estimator=MaximumLikelihoodEstimator)
```

```
# Learning CPD using Maximum likelihood estimators (No output shown in manual)
```

```
# Inferencing with Bayesian Network
```

```
In [6]: hd_infer = VariableElimination(model)
```

```
In [10]: print('1. Probability of heartdesease given evidence = restecg:1')
```

```
q1 = hd_infer.query(variables=['target'], evidence={'restecg': 1})
```

```
print(q1)
```

```
In [8]: print('2. Probability of heartdesease given evidence = cp:2')
```

```
q2 = hd_infer.query(variables=['target'], evidence={'cp': 2})
```

```
print(q2)
```

OUTPUT:

Out [3]:

	Age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	2	1	2	0

[303 rows x 14 columns]

Out [10]: 1. Probability of heartdesease given evidence = restecg:1

```
+-----+-----+
| target | phi(target)|
+-----+-----+
| target(0)| 0.4031 |
| target(1)| 0.5969 |
+-----+-----+
```

Out [8]: 2. Probability of heartdesease given evidence = cp:2

```
+-----+-----+
| target | phi(target) |
+-----+-----+
| target(0)| 0.4862 |
| target(1)| 0.5138 |
+-----+-----+
```

PROGRAM:

```
In [1]: #Imported libraries and dataset
    from sklearn import datasets
    from sklearn.cluster import KMeans
    from sklearn.utils import shuffle
    import numpy as np
    import pandas as pd
```

```
In [2]: #Loading data-set for EM algorithm
    iris = datasets.load_iris()
    X = pd.DataFrame(iris.data)
    Y = pd.DataFrame(iris.target)
    X.head()      #(Output not in manual, but helpful for context)
    Y.head()      #(Output not in manual, but helpful for context)
```

```
In [3]: #Defining EM Model
    from sklearn.mixture import GaussianMixture
    model2=GaussianMixture(n_components=3,random_state=3425)
    #Training of the model
    model2.fit(X)
```

```
In [4]: #Predicting classes for our data
    uu=model2.predict(X)
    #Accuracy of EM Model
    from sklearn.metrics import confusion_matrix
    cm=confusion_matrix(Y,uu)
    print(cm)
    from sklearn.metrics import accuracy_score
    print(accuracy_score(Y,uu))
```

OUTPUT:

Out [3]: GaussianMixture(n_components=3, random_state=3425)

Out [4]:

[[0 50 0]

[45 5 0]

[0 0 50]]

0.3333333333333333

PROGRAM:

In [1]: import pandas as pd

```
from sklearn.datasets import load_iris  
iris=load_iris()
```

In [3]: iris.feature_names

In [4]: iris.target_names

In [5]: df=pd.DataFrame(iris.data,columns=iris.feature_names)

In [6]: df.head()

In [7]: df['target']=iris.target

```
df.head()
```

In [8]: df[df.target==1].head()

In [9]: df[df.target==2].head()

In [18]: df['flower_name']=df.target.apply(lambda x: iris.target_names[x])

In [19]: df.head()

In [20]: df0=df[:50]

```
df1=df[50:100]
```

```
df2=df[100:]
```

In [21]: import matplotlib.pyplot as plt

```
%matplotlib inline
```

In [31]: plt.xlabel('Sepal Length')

```
plt.ylabel('Sepal Width')
```

```
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color="green",marker='+')
```

```
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],color="blue",marker='.'
```

In [32]: plt.xlabel('Petal Length')

```
plt.ylabel('Petal Width')
```

```
plt.scatter(df0['petal length(cm)'],df0['petal width (cm)'],color="green",marker='+')
```

```
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color="blue",marker='.'
```

```
In [23]: from sklearn.model_selection import train_test_split
```

```
X=df.drop(['target','flower_name'],axis='columns')  
Y=df.target  
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1)
```

```
In [24]: len(X_train)
```

```
In [26]: len(X_test)
```

```
In [34]: from sklearn.neighbors import KNeighborsClassifier
```

```
knn=KNeighborsClassifier(n_neighbors=10)
```

```
In [35]: knn.fit(X_train,y_train)
```

```
In [36]: knn.score(X_test,y_test)
```

```
In [37]: knn.predict([[4.8, 3.0, 1.5, 0.3]])
```

```
In [38]: from sklearn.metrics import confusion_matrix
```

```
y_pred=knn.predict(X_test)  
cm=confusion_matrix(y_test,y_pred)
```

```
In [39]: cm
```

```
In [40]: import seaborn as sns
```

```
import matplotlib.pyplot as plt  
plt.figure(figsize=(7,5))  
sns.heatmap(cm, annot=True)  
plt.xlabel('Predicted')  
plt.ylabel('Truth')
```

```
In [41]: from sklearn.metrics import classification_report
```

```
In [42]: print(classification_report(y_test,y_pred))
```

OUTPUT:

Out [3]: ['sepal length (cm)',

'sepal width (cm)',

'petal length (cm)',

'petal width (cm)']

Out [4]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

Out [6]: sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Out [7]: sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) target

0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Out [8]: sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) target

50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

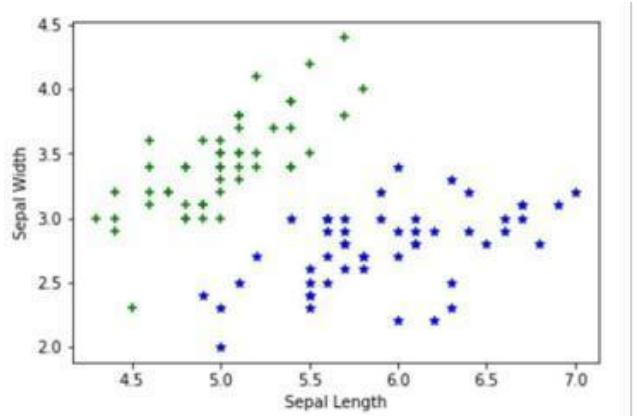
Out [9]: sepal length(cm) sepal width(cm) petal length(cm) petal width(cm) target

100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

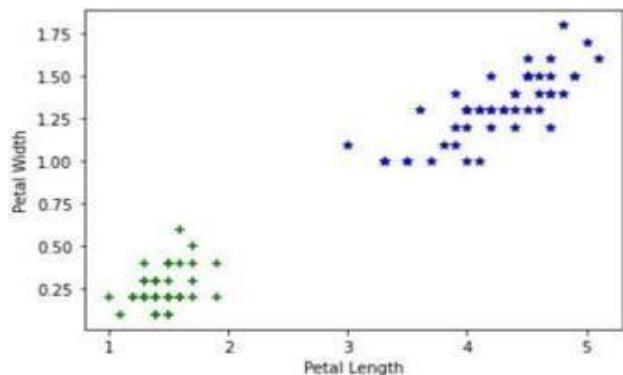
Out [19]: sepal length(cm) sepal width(cm) petal length(cm) petal width(cm) target flower_name

0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Out [31]: <matplotlib.collections.PathCollection at 0x2388bc49cb0>



Out [32]: <matplotlib.collections.PathCollection at 0x2388bc6c1f0>



Out [24]: 120

Out [26]: 30

Out [35]: KNeighborsClassifier(n_neighbors=10)

Out [36]: 0.9666666666666667

Out [37]: array([0])

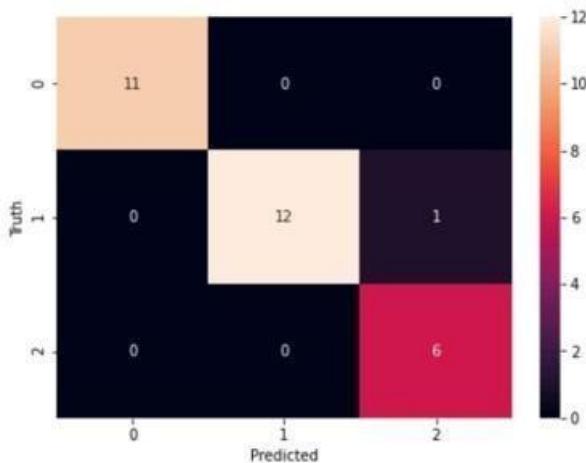
Out [39]:

array([[11, 0, 0],

[0, 12, 1],

[0, 0, 6]], dtype=int64)

Out[40]: <AxesSubplot: xlabel='Predicted', ylabel='Truth'>



	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	0.92	0.96	13
2	0.86	1.00	0.92	6
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

PROGRAM:

In [1]: import pandas as pd

In [2]: df=pd.read_csv("salaries.csv")

```
df.head()
```

In [3]: inputs=df.drop('salary_more_then_100k',axis='columns')

In [4]: target = df['salary_more_then_100k']

In [5]: from sklearn.preprocessing import LabelEncoder

```
le_company = LabelEncoder()
```

```
le_job = LabelEncoder()
```

```
le_degree = LabelEncoder()
```

In [6]: inputs['company_n']=le_company.fit_transform(inputs['company'])

```
inputs['job_n']=le_job.fit_transform(inputs['job'])
```

```
inputs['degree_n']=le_degree.fit_transform(inputs['degree'])
```

In [7]: inputs

In [8]: inputs_n=inputs.drop(['company','job','degree'],axis='columns')

In [9]: inputs_n

In [10]: target

In [11]: from sklearn import tree

```
model = tree.DecisionTreeClassifier()
```

In[12]: model.fit(inputs_n, target)

In[13]: model.score(inputs_n, target)

In [14]: print("Is salary of Google, Computer Engineer, Bachelors degree > 100 k ?")

```
model.predict([[2,1,0]])
```

In [15]: print("Is salary of Google, Computer Engineer, Masters degree > 100 k ?")

```
model.predict([[2,1,1]])
```

OUTPUT:

```
Out [2]:   company      job      degree  salary_more_then_100k
          0  Google sales executive    bachelors      0
          1  Google sales executive    masters       0
          2  Google business manager  bachelors      1
          3  Google business manager  masters       1
          4  Google computer programmer bachelors      0
```

Out [7]:

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc phar	sales executive	masters	0	2	1
7	abc phar	computer programmer	bachelors	0	1	0
8	abc phar	business manager	bachelors	0	0	0
9	abc phar	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0

Out [9]: company_n job_n degree_n

0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1

Out [10]:

0	0
1	0
2	1
3	1

Name: salary_more_then_100k, dtype: int64

Out [12]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

Out [13]: 1.0

Out [14]: array([0], dtype=int64)

Out [15]: array([1], dtype=int64)

PROGRAM:

In [1]: import numpy as np

```
X = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)  
y = np.array([[92], [86], [89]], dtype=float)  
X = X/np.amax(X, axis=0) # scale X  
y = y/100 # Maximum test score is 100
```

```
class NeuralNetwork(object):  
  
    def __init__(self):  
        self.inputSize = 2  
        self.outputSize = 1  
        self.hiddenSize = 3  
        self.W1 = np.random.randn(self.inputSize, self.hiddenSize)  
        self.W2 = np.random.randn(self.hiddenSize, self.outputSize)
```

```
    def feedForward(self, X):  
        self.z = np.dot(X, self.W1)  
        self.z2 = self.sigmoid(self.z)  
        self.z3 = np.dot(self.z2, self.W2)  
        output = self.sigmoid(self.z3)  
        return output
```

```
    def sigmoid(self, s, deriv=False):  
        if (deriv == True):  
            return s * (1 - s)  
        return 1/(1 + np.exp(-s))
```

```
    def backward(self, X, y, output):  
        self.output_error = y - output
```

```
self.output_delta = self.output_error * self.sigmoid(output, deriv=True)
self.z2_error = self.output_delta.dot(self.W2.T)
self.z2_delta = self.z2_error * self.sigmoid(self.z2, deriv=True)
self.W1 += X.T.dot(self.z2_delta)
self.W2 += self.z2.T.dot(self.output_delta)

def train(self, X, y):
    output = self.feedForward(X)
    self.backward(X, y, output)
    NN = NeuralNetwork()
    for i in range(1000):
        if (i % 100 == 0):
            print("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))
        NN.train(X, y)
        print("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))
        print("Input: " + str(X))
        print("Actual Output: " + str(y))
        print("Predicted Output: " + str(NN.feedForward(X)))
```

OUTPUT:

Out [1]:

Loss: 0.024912562840868836

Loss: 0.020568438888811255

Loss: 0.01750275817865184

Loss: 0.014471137378264548

Loss: 0.012306367468437503

Loss: 0.010729793160898522

Loss: 0.00910854422123443

Loss: 0.007914369791039867

Loss: 0.006921107548652352

Loss: 0.006684151847431011

Loss: 0.00632009228555556

Input: [[0.22222222 1.]

[0.11111111 0.55555556]

[0.33333333 0.66666667]]

Actual Output: [[0.92]

[0.86]

[0.89]]

Predicted Output: [[0.85079209]

[0.80977176]

[0.7961218]]

PROGRAM:

In [1]: from sklearn.datasets import load_breast_cancer

In [2]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.svm import SVC
import pandas as pd

In [3]: cancer = load_breast_cancer()

```
X_data = pd.DataFrame(cancer.data, columns = cancer.feature_names)  
y_target = cancer.target  
a_target = cancer.target_names
```

In [4]: df = pd.DataFrame(cancer.data, columns = list(a.feature_names))

```
df['diagnosis'] = a.target  
df.head()
```

In [5]: X_train, X_test, y_train, y_test = train_test_split(X_data, y_target,
stratify=y_target, random_state=42)
print(f'X train shape: {X_train.shape}')
print(f'X test shape: {X_test.shape}')
print(f'y train shape: {y_train.shape}')
print(f'y test shape: {y_test.shape}')

In [6]: svm = SVC(kernel='linear')

```
svm.fit(X_train, y_train)
```

In [7]: print(f'Accuracy on training subset is: {svm.score(X_train, y_train):.3f}')

```
print(f'Accuracy on test subset is: {svm.score(X_test, y_test):.3f}')
```

In [8]: `scaler = StandardScaler()`

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

In [9]: `svm = SVC(kernel='linear')`

```
svm.fit(X_train_scaled, y_train)
```

In [10]: `print(f'Accuracy on training subset is: {svm.score(X_train_scaled, y_train):.3f}')`

```
print(f'Accuracy on test subset is: {svm.score(X_test_scaled, y_test):.3f}')
```

OUTPUT:

Out [3]: array(['malignant', 'benign'], dtype='<U9')

Out [4]:

mean radius mean texture mean perimeter mean area mean smoothness mean compactness mean concavity mean concave points mean symmetry mean fractal dimension worst radius worst texture worst perimeter worst area worst smoothness worst compactness worst concavity worst concave points worst symmetry worst fractal dimension

0	17.99	10.38	122.80	1001.00	0.11840	0.27760	0.30010
	0.14710	0.24190	0.07871	25.38	17.33	184.60	2019.0
	0.16220	0.66560	0.71190	0.26540	0.46010		0.11890
1	20.57	17.77	132.90	1326.00	0.08474	0.07864	0.08690
	0.07017	0.18120	0.05667	24.99	23.41	158.80	1956.0
	0.12380	0.18660	0.24160	0.18600	0.27500		0.08902
2	19.69	21.25	130.00	1203.00	0.10960	0.15990	0.19740
	0.12790	0.20690	0.05999	23.57	25.53	152.50	1709.0
	0.14440	0.42450	0.45040	0.24300	0.36130		0.08758
3	11.42	20.38	77.58	386.10	0.14250	0.28390	0.24140
	0.10520	0.25970	0.09744	14.91	26.50	98.87	567.7
	0.20980	0.86630	0.68690	0.25750	0.66380		0.17300

Out[6]:

X train shape: (426, 30)

X test shape: (143, 30)

y train shape: (426,) y

test shape: (143,)

svm.fit(X_train,y_train)

Accuracy on training subset is: 0.962

Accuracy on test subset is: 0.965

Accuracy on training subset is: 0.990

Accuracy on test subset is: 0.986

PROGRAM:

In [1]: import pandas as pd

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model
import LogisticRegression
```

In [2]: data = pd.read_csv('spam.csv')

```
data.head()
```

In [23]: data['Message'].value_counts()

In [39]: from sklearn import preprocessing

```
le = preprocessing.LabelEncoder()
le.fit_transform(data['Category'])
```

In [40]: data['Category'] = le.fit_transform(data['Category'])

```
data = data.drop(columns='Message', axis=1)
```

In [41]: y = data['Message']

In [42]: x

In [43]: y

In [49]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, shuffle=True)

In [50]: print(x.shape, x_train.shape, x_test.shape)

In [61]: model = LogisticRegression(max_iter=1000)

```
In [62]: model.fit(x_train, y_train)
```

```
In [63]: from sklearn.metrics import accuracy_score  
x_train_prediction = model.predict(x_train)  
trained_data_accuracy = accuracy_score(y_train, x_train_prediction)  
print(trained_data_accuracy)
```

```
In [65]: print('Accuracy on training data :', round(trained_data_accuracy*100, 2), '%')
```

```
In [67]: x_test_prediction = model.predict(x_test)  
test_data_accuracy = accuracy_score(y_test, x_test_prediction)  
print(test_data_accuracy)
```

```
In [68]: print('Accuracy on test data :', round(test_data_accuracy*100, 2), '%')
```

OUTPUT:

Out [22]:	Category	Message
0	ham	Go until jurong point, crazy. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Out [23]:

Sorry, I'll call later

30

I can't pick the phone right now. Pls send a message

12

OK.....

.

Say this slowly.? GOD,I LOVE YOU & I NEED YOU,CLEAN MY HEART WITH YOUR BLOOD.Send this to Ten special people & u c mira cle tomorrow, do it,pls.pls do it...

Haha, my friend Tyler literally just asked if you could get him a dubsack

Try neva mate!!

4

Ur cash-balance is currently 500 pounds - to maximize ur cash-in now send GO to 86688 only 150p/msg. CC: 08718720201 PO BOX 11 OXR W1A OER. T&Cs apply.

4

Its just the effect of irritation. Just ignore it

1

Booked ticket for pongal

1

Name: Message, Length: 5157, dtype: int64

Out [42]: Category

```
0  0
1  0
2  1
3  0
4  0
...
5567 1
5568 0
5569 0
5570 0
5571 0
```

5572 rows x 1 columns

Out [43]:

```
0    1080
1    3126
2    999
3    4121
4    2781
...
5567  4596
5568  4025
5569  3932
5570  4187
5571  3947
```

Name: Message, Length: 5572, dtype: int32

(5572, 1) (4457, 1) (1115, 1)

Out [62]: LogisticRegression(max_iter=1000)

0.006282700874916982

Accuracy on training data: 0.63 %

0.006278026905829596

Accuracy on test data: 0.63 %

PROGRAM:

In [1]: import pandas as pd

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [2]: data = pd.read_csv('diabetes.csv')

```
data.head()
```

In [6]: data['Outcome'].value_counts()

In [7]: x = data.drop(columns='Outcome', axis=1)

In [9]: y = data['Outcome']

In [10]: x

In [11]: y

In [12]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y,
random_state=2)

In [17]: print(x_train.shape, x_test.shape)

In [18]: from sklearn.metrics import accuracy_score

In [20]: model = LogisticRegression(max_iter=1000)

```
model.fit(x_train, y_train)
x_train_prediction = model.predict(x_train)
trained_data_accuracy = accuracy_score(y_train, x_train_prediction)
print(trained_data_accuracy)
```

In [21]: print('Accuracy on training data :', round(trained_data_accuracy*100, 2), '%')

In [22]: x_test_prediction = model.predict(x_test)

```
test_data_accuracy = accuracy_score(y_test, x_test_prediction)
print(test_data_accuracy)
```

In [23]: print('Accuracy on test data :', round(test_data_accuracy*100, 2), '%')

OUTPUT:

	Out [3]:	Preg	Glucose	BP	SkinThick	Insulin	BMI	DiabPed	Age	Outcome
0		6	148	72	35	0	33.6	0.627	50	1
1		1	85	66	29	0	26.6	0.351	31	2
2		1	83	64	0	0	23.3	0.672	32	1
3		1	89	66	23	94	28.1	0.167	21	0
4		0	137	40	35	168	43.1	2.288	33	1

Out [6]:

```
0 500  
1 268
```

Name: Outcome, dtype: int64 Out[10]:

	Preg	Glucose	BP	SkinThickness	Insulin	BMI	DiabPediFun	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	64	0	0	0	23.3	0.672	32
3	1	89	66	23	1	23.4	0.768	
4	0	137	40	35	168	43.1		2.2833
..
763	10	101	76	48	180	32.9		0.171 63
764	2	122	70	27	0	36.8		0.340 27
765	5	121	72	23	112	26.2		0.245 30
766	1	126	60	0	0	30.1		0.349 47
767	1	93	70	31	0	30.4		0.315 23

[768 rows x 8 columns]

Out [11]:

0 1

1 0

2 1

3 0

4 1

763 0

764 0

765 0

766 1

767 0

Name: Outcome, Length: 768, dtype: int64

(614, 8) (154, 8)

0.7882736156351792

Accuracy on training data: 78.83 %

0.7597402597402597

Accuracy on test data: 75.97 %

PROGRAM:

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

```
In [2]: # Loading the data from csv file to a Pandas DataFrame  
customer_data = pd.read_csv('Mall_Customers.csv')
```

```
In [3]: #First 5 rows in the dataframe  
customer_data.head()
```

```
In [4]: #Finding the number of rows and columns  
customer_data.shape
```

```
In [5]: #Getting some informations about the dataset  
customer_data.info()
```

```
In [6]: #checking for missing values  
Customer_data.isnull().sum()
```

```
In [7]: X = customer_data.iloc[:,[3,4]].values
```

```
In [8]: print(X)
```

```
In [9]: #finding wcss value for different number of clusters
```

```
wcss = []  
for i in range(1,11):  
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)  
    kmeans.fit(X)  
    wcss.append(kmeans.inertia_)
```

```
In [10]: #plot on elbow graph
```

```
    sns.set()  
    plt.plot(range(1,11), wcss)  
    plt.title('The Elbow Point  
Graph')  
    plt.xlabel('Number of Clusters')  
    plt.ylabel('WCSS')  
    plt.show()
```

```
In [11]: kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)
```

```
Y = kmeans.fit_predict(X)      print(Y)
```

OUTPUT:

Out [3]: CustomerID Gender Age Annual Income (k\$) Spending Score (1-100)

0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Out [4]: (200, 5)

Out [5]:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 200 entries, 0 to 199 Data

columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	CustomerID	200	non-null int64
1	Gender	200	non-null object
2	Age	200	non-null int64
3	Annual Income (k\$)	200	non-null int64
4	Spending Score (1-100)	200	non-null int64

dtypes: int64(4), object(1) memory usage: 7.9 KB

Out [6]:

CustomerID

Gender 0

Age 0

Annual Income (k\$) 0

Spending Score 0

dtype: int64

Out [8]:

```
[[ 15  39]
 [ 15  81]
 [ 16  6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18  6]
 [ 18  94]
 [ 19  3]
 [ 19  82]
 [ 20  13]
 [ 20  75]
 [ 21  66]
 [ 21  55]
 [ 23  99]
 [ 23  15]
 [ 24  35]
 [ 24  73]
 [ 25  57]
 [ 25  73]
 [ 26  32]
 [ 26  82]
 [ 27  33]
 [ 30  4]]
... (array continues for 200 rows) ...
```