EXERCISE - 1

```
import math

print(math.floor(3.7))
print(math.ceil(3.7))
print(math.sqrt(16))
print(math.isqrt(16))
print(math.gcd(54, 24))
```

```
3
4
4.0
4
6
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr.ndim)
print(arr.shape)
print(arr.size)
print(arr.sum())
print(arr.mean())
print(np.sort(arr))
print(np.sin(arr))
```

```
1
(5,)
5
15
3.0
[1 2 3 4 5]
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
```

```
from scipy import linalg
import numpy as np

matrix = np.array([[1, 2], [3, 4]])

print(linalg.det(matrix))
eigenvalues, eigenvectors = linalg.eig(matrix)
print(eigenvalues)
print(eigenvectors)
```

```
-2.0
[-0.37228132+0.j  5.37228132+0.j]
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

```
import numpy as np

lst = list(range(1, 13))

arr = np.array(lst)
print(arr)
print(arr.reshape(3, 4))
print(arr.reshape(2, 2, 3))
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

```
import numpy as np

print(np.eye(3))
print(np.zeros((3, 3)))
print(np.ones((2, 4)))
```

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

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

```
from scipy import linalg
import numpy as np

matrix = np.array([[1, 2], [3, 4]])

print(linalg.det(matrix))
```

```
-2.0
```

```
from scipy import linalg
import numpy as np

matrix = np.array([[1, 2], [3, 4]])

eigenvalues, eigenvectors = linalg.eig(matrix)
print(eigenvalues)
print(eigenvectors)
```

```
[-0.37228132+0.j  5.37228132+0.j]
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

```
import pandas as pd

s = pd.Series([10, 20, 30, 40, 50])
print(s)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

```
print(s.index)
print(s.values)
```

```
RangeIndex(start=0, stop=5, step=1)
[10 20 30 40 50]
```

```
import numpy as np
import pandas as pd

print(np.array([1, 2, 3]))
print(pd.Series([1, 2, 3]))
```

```
[1 2 3]
0    1
1    2
2    3
dtype: int64
```

```
import pandas as pd

s2 = pd.Series([100, 200, 300], index=['a', 'b', 'c'])
print(s2)
```

```
a    100
b    200
c    300
dtype: int64
```

```
print(s2['a'])
```

```
100
```

EXERCISE 3

```
import pandas as pd

df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print(df)
```

```
   A  B
0  1  3
1  2  4
```

import pandas as pd

csv_df = pd.read_csv('myfile.csv') print(csv_df)

```
from sklearn import datasets

iris = datasets.load_iris()
print(iris.data[:5])
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

```
import statistics as stats

data = [1, 2, 2, 3, 4, 5, 5, 5]

print(stats.mean(data))
print(stats.median(data))
print(stats.mode(data))
print(stats.variance(data))
print(stats.stdev(data))
```

```
3.375
3.5
5
2.5535714285714284
1.5979898086569353
```

```
import numpy as np

arr = np.array([[1, 2], [3, 4]])
reshaped = arr.reshape(4, 1)
print(reshaped)
```

```
[[1]
 [2]
 [3]
 [4]]
```

```
import numpy as np

arr = np.array([[1, 2], [3, 4]])
filtered = arr[arr > 2]
print(filtered)
```

```
[3 4]
```

```
import pandas as pd

df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
merged = pd.concat([df1, df2])
print(merged)
```

```
   A  B
0  1  3
1  2  4
0  5  7
1  6  8
```

```
import pandas as pd

df3 = pd.DataFrame({'A': [1, None, 3], 'B': [4, 5, None]})
print(df3.fillna(0))
```

```
     A    B
0  1.0  4.0
```

```
1  0.0  5.0
2  3.0  0.0
```

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

data = [1, 2, 3, 4, 5]
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(np.array(data).reshape(-1, 1))
print(data_scaled)
```

```
[[0.  ]
 [0.25]
 [0.5 ]
 [0.75]
 [1.  ]]
```

EXERCISE 4

```
from sklearn.decomposition import PCA
from sklearn import datasets

iris = datasets.load_iris()
pca = PCA(n_components=2)
pca_result = pca.fit_transform(iris.data)
print(pca_result[:5])
```

```
[[-2.68412563  0.31939725]
 [-2.71414169 -0.17700123]
 [-2.88899057 -0.14494943]
 [-2.74534286 -0.31829898]
 [-2.72871654  0.32675451]]
```

EXERCISE 5

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
y = iris.target

clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X, y)
print(clf.predict(X[:5]))
```

```
[0 0 0 0 0]
```

EXERCISE 6

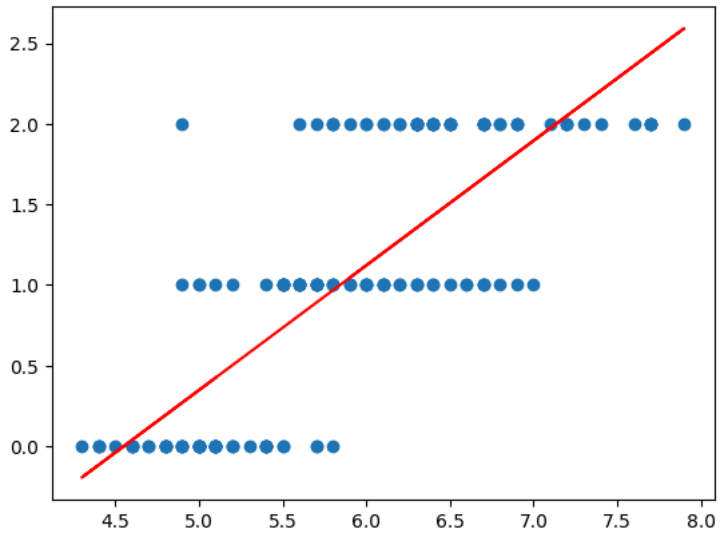
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
y = iris.target

for trees in [20, 50, 100, 200, 500]:
    clf = RandomForestClassifier(n_estimators=trees)
    clf.fit(X, y)
    print(f'Trees: {trees}, Score: {clf.score(X, y)}')
```

```
Trees: 20, Score: 0.9933333333333333
Trees: 50, Score: 1.0
Trees: 100, Score: 1.0
Trees: 200, Score: 1.0
Trees: 500, Score: 1.0
```

Double-click (or enter) to edit

EXERCISE 7

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, 0].reshape(-1, 1)
y = iris.target

model = LinearRegression()
model.fit(X, y)

plt.scatter(X, y)
plt.plot(X, model.predict(X), color='red')
plt.show()
```

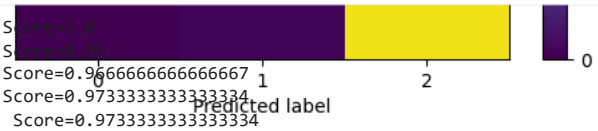


EXERCISE 8

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris.data
y = iris.target

clf = LogisticRegression(max_iter=200)
clf.fit(X, y)
y_pred = clf.predict(X)

cm = confusion_matrix(y, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

EXERCISE 9

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
y = iris.target

for k in [1, 3, 5, 7, 11]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X, y)
    print(f'k={k}, Score={knn.score(X, y)}')
```




```
k=1, Score=1.0
k=3, Score=0.98
k=5, Score=0.9666666666666667
k=7, Score=0.9733333333333334
k=11, Score=0.9733333333333334
```

EXERCISE 10

```
from sklearn.svm import SVC
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
y = iris.target

for kernel in ['linear', 'poly', 'rbf']:
    svc = SVC(kernel=kernel)
    svc.fit(X, y)
    print(f'Kernel={kernel}, Score={svc.score(X, y)}')
```

```
Kernel=linear, Score=0.9933333333333333
Kernel=poly, Score=0.9733333333333334
Kernel=rbf, Score=0.9733333333333334
```

EXERCISE 11

```
from sklearn.cluster import KMeans
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data

for k in [1, 3, 5]:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    print(f'k={k}, Inertia={kmeans.inertia_}')
```

```
k=1, Inertia=681.3705999999996
k=3, Inertia=78.85566582597727
k=5, Inertia=46.44618205128204
```