Formative Assignment

Deadline: Tuesday 28 October 2025.

Your first assignment is submitted for feedback only, and does not form part of the assessment of passing or failing this course.

For this assignment, you must submit a completed workbook. There are five components to it.

Please email your assignment to onlinemarking@conted.ox.ac.uk along with a completed Declaration of Authorship Form. The Online Courses Office will confirm receipt of your assignment.

Name your file based on the naming convention: CourseTitle_Assignment1_Surname.pdf

In [1]:
```
## Let's import the libraries we are going to use later on
import numpy as np
import pandas as pd

# Let's set the precision for NumPy and Pandas
np.set_printoptions(precision=3)
pd.options.display.float_format = '{:,.2f}'.format
```

Exercises on Data Structures

1. Compute statistics on a Python list

Given the following list

```
l = [12, 45, 12, 999, 10, 8, 76, 20, 10, 10, 7, 70, 17]
```

Compute the mean, the median and the standard deviation of the values in the list, using only built-in functionalities of python (no imported libraries, no NumPy or Pandas)

In [2]:
```
l = [12, 45, 12, 999, 10, 8, 76, 20, 10, 10, 7, 70, 17]

# mean

def mean_function(x):
    total_sum = 0
    for current_item in l:
      total_sum = total_sum + current_item
    number_of_terms = len(l)
    mean_value = total_sum/number_of_terms
    return mean_value

result = mean_function(l)

print(f"Mean of {l} is : {result:.2f}")
```

```
Mean of [12, 45, 12, 999, 10, 8, 76, 20, 10, 10, 7, 70, 17] is : 99.69
```

In [3]:
```
l = [12, 45, 12, 999, 10, 8, 76, 20, 10, 10, 7, 70, 17]

def bubble_sort(x):
    for i in range(len(x)):
        for j in range(0, len(x) -1 -i):
            if x[j] > x[j + 1]:
                x[j], x[j + 1] = x[j + 1], x[j]

    return(x)

def median_function(x):
    values_list = list(set(x))
    item_index = len(values_list)/2
    median_value = 0
    number_of_terms = len(values_list)
    if len(values_list) % 2 == 0:
       median_value = (values_list[number_of_terms // 2 - 1] + x[number_of_terms // 2])/2
    else:
       median_value = values_list[number_of_terms // 2]

    return median_value
```

```
#median

result = median_function(l)

print(f"Median of {l} is : {result:.0f}")
```

```
Median of [12, 45, 12, 999, 10, 8, 76, 20, 10, 10, 7, 70, 17] is : 9
```

In [4]:
```
l = [12, 45, 12, 999, 10, 8, 76, 20, 10, 10, 7, 70, 17]

# standard deviation

def standard_dev_function(x):
    intermediate_value = 0
    count = 0
    average_value = mean_function(l)
    for i, current_value in enumerate(x):
        count = count + 1
        delta_1 = current_value - average_value
        average_value = average_value + delta_1 / count
        delta_2 = current_value - average_value
        intermediate_value = intermediate_value + delta_1 * delta_2

    if count < 0:
        return 0

    variance = intermediate_value / count

    return (variance ** 0.5)

result = standard_dev_function(l)

print(f"Standard Deviation of {l} is : {result:.2f}")
```

```
Standard Deviation of [12, 45, 12, 999, 10, 8, 76, 20, 10, 10, 7, 70, 17] is : 260.60
```

2A. Create a dictionary from a list of tuples (records)

Given the following list of tuples, each one holding a record

```
records = [
    ("author", 'title', 'publication_year', 'page_count'),
    ('J. R. R. Tolkien', 'The Fellowship of the Ring', 1954, 398),
    ('J. K. Rowling', 'Harry Potter and the Philosopher\'s stone', 1996, 223),
    ('Evelyn Waugh', 'Brideshead Revisited', 1945, 402),
    ('Philip K. Dick', 'Ubik', 1969, 202),
    ('Thomas Pynchon', "Gravity's Rainbow", 1973, 760),
    ('Stephen King', 'The Stand', 1978, 829)
]
```

Convert them into a list of dictionaries, using the first tuple as keys for all the dictionaries and all the other tuples as values, one per dictionary.
Expected result:

```
books = [{'author': 'J. R. R. Tolkien',
  'title': 'The Fellowship of the Ring',
  'publication_year': 1954,
  'page_count': 398},
 {'author': 'J. K. Rowling',
  'title': "Harry Potter and the Philosopher's stone",
  'publication_year': 1996,
  'page_count': 223},
 {'author': 'Evelyn Waugh',
  'title': 'Brideshead Revisited',
  'publication_year': 1945,
  'page_count': 402},
 {'author': 'Philip K. Dick',
  'title': 'Ubik',
  'publication_year': 1969,
  'page_count': 202},
 {'author': 'Thomas Pynchon',
  'title': "Gravity's Rainbow",
  'publication_year': 1973,
  'page_count': 760},
 {'author': 'Stephen King',
  'title': 'The Stand',
  'publication_year': 1978,
  'page_count': 829}]
```

In [5]:

```
records = [
    ('author', 'title', 'publication_year', 'page_count'),
    ('J. R. R. Tolkien', 'The Fellowship of the Ring', 1954, 398),
    ('J. K. Rowling', 'Harry Potter and the Philosopher\'s stone', 1996, 223),
    ('Evelyn Waugh', 'Brideshead Revisited', 1945, 402),
    ('Philip K. Dick', 'Ubik', 1969, 202),
    ('Thomas Pynchon', "Gravity's Rainbow", 1973, 760),
    ('Stephen King', 'The Stand', 1978, 829)
]

## Write your solution here

def populate_dict_function(r):
    columns = r[0]
    rows = r[1:]
    books = []
    for row in rows :
        book = {}
        for index, column in enumerate(columns):
           book[column] = row[index]
        books.append(book)
    return books

def print_dict_function(x):
    for item in x:
        for key, value in item.items():
            print(f"{key}: {value}")
        print("\n")

books = populate_dict_function(records)
print(f"Transform {records} to : \n")
print(books)
print("\n")
print_dict_function(books)
```

Transform [('author', 'title', 'publication_year', 'page_count'), ('J. R. R. Tolkien', 'The Fellowship of the Ring', 1954, 398), ('J. K. Rowling', "Harry Potter and the Philosopher's stone", 1996, 223), ('Evelyn Waugh', 'Brideshead Revisited', 1945, 402), ('Philip K. Dick', 'Ubik', 1969, 202), ('Thomas Pynchon', "Gravity's Rainbow", 1973, 760), ('Stephen King', 'The Stand', 1978, 829)] to :

[{'author': 'J. R. R. Tolkien', 'title': 'The Fellowship of the Ring', 'publication_year': 1954, 'page_count': 398}, {'author': 'J. K. Rowling', 'title': "Harry Potter and the Philosopher's stone", 'publication_year': 1996, 'page_count': 223}, {'author': 'Evelyn Waugh', 'title': 'Brideshead Revisited', 'publication_year': 1945, 'page_count': 402}, {'author': 'Philip K. Dick', 'title': 'Ubik', 'publication_year': 1969, 'page_count': 202}, {'author': 'Thomas Pynchon', 'title': "Gravity's Rainbow", 'publication_year': 1973, 'page_count': 760}, {'author': 'Stephen King', 'title': 'The Stand', 'publication_year': 1978, 'page_count': 829}]

author: J. R. R. Tolkien
title: The Fellowship of the Ring
publication_year: 1954
page_count: 398

author: J. K. Rowling
title: Harry Potter and the Philosopher's stone
publication_year: 1996
page_count: 223

author: Evelyn Waugh
title: Brideshead Revisited
publication_year: 1945
page_count: 402

author: Philip K. Dick
title: Ubik
publication_year: 1969
page_count: 202

author: Thomas Pynchon
title: Gravity's Rainbow
publication_year: 1973
page_count: 760

```
author: Stephen King
title: The Stand
publication_year: 1978
page_count: 829
```

2B. Dictionary manipulation

Write a function that takes the dictionary like `books` and return the author whose book has the highest page count.

Apply the function to `books` to verify that it works correctly

In [6]:
```python
## Write your solution here

sorted_by_page_count = sorted(books, key=lambda x:["page_count"])

print(f"Longest Book by Page Count: {list(sorted_by_page_count)[-1]}")
```

```
Longest Book by Page Count: {'author': 'Stephen King', 'title': 'The Stand', 'publication_year': 1978, 'page_count': 829}
```

Exercises on Numpy

For these exercises we will work on the IRIS dataset, a very popular multivariate dataset for data anaylisis, first introduced by Ronald Fisher in 1936.

See more about the IRIS dataset heare: https://archive.ics.uci.edu/ml/datasets/Iris

The data source is the file `iris.data` retrieved from the Web. It contains the data for this example in comma separated values (CSV) format. The number of columns is 5 and the number of rows is 150. The data set is composed of 50 samples from each of three species of Iris Flower: Iris Setosa, Iris Virginica and Iris Versicolor. Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species one from each other.

The variables are:

```
    sepal_length: Sepal length, in centimeters.
    sepal_width: Sepal width, in centimeters.
    petal_length: Petal length, in centimeters.
    petal_width: Petal width, in centimeters.
```

The three categories of Irises are: 'Iris Setosa', 'Iris Versicolor', and 'Iris Virginica'.

Let's first import the `iris` dataset as a 1-dimensional array of tuples.

In [7]:
```
# Input:
# Import iris keeping the text column intact
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris = np.genfromtxt(
    url, delimiter=',',
    names=['sepal length', 'sepal width', 'petal length', 'petal width', 'species'],
    dtype=[np.float64, np.float64, np.float64, np.float64, 'U15']
)
iris
```

Out[7]:
```
array([(5.1, 3.5, 1.4, 0.2, 'Iris-setosa'),
       (4.9, 3. , 1.4, 0.2, 'Iris-setosa'),
       (4.7, 3.2, 1.3, 0.2, 'Iris-setosa'),
       (4.6, 3.1, 1.5, 0.2, 'Iris-setosa'),
       (5. , 3.6, 1.4, 0.2, 'Iris-setosa'),
       (5.4, 3.9, 1.7, 0.4, 'Iris-setosa'),
       (4.6, 3.4, 1.4, 0.3, 'Iris-setosa'),
       (5. , 3.4, 1.5, 0.2, 'Iris-setosa'),
       (4.4, 2.9, 1.4, 0.2, 'Iris-setosa'),
       (4.9, 3.1, 1.5, 0.1, 'Iris-setosa'),
       (5.4, 3.7, 1.5, 0.2, 'Iris-setosa'),
       (4.8, 3.4, 1.6, 0.2, 'Iris-setosa'),
       (4.8, 3. , 1.4, 0.1, 'Iris-setosa'),
       (4.3, 3. , 1.1, 0.1, 'Iris-setosa'),
       (5.8, 4. , 1.2, 0.2, 'Iris-setosa'),
       (5.7, 4.4, 1.5, 0.4, 'Iris-setosa'),
       (5.4, 3.9, 1.3, 0.4, 'Iris-setosa'),
       (5.1, 3.5, 1.4, 0.3, 'Iris-setosa'),
       (5.7, 3.8, 1.7, 0.3, 'Iris-setosa'),
       (5.1, 3.8, 1.5, 0.3, 'Iris-setosa'),
       (5.4, 3.4, 1.7, 0.2, 'Iris-setosa'),
       (5.1, 3.7, 1.5, 0.4, 'Iris-setosa'),
       (4.6, 3.6, 1. , 0.2, 'Iris-setosa'),
       (5.1, 3.3, 1.7, 0.5, 'Iris-setosa'),
       (4.8, 3.4, 1.9, 0.2, 'Iris-setosa'),
       (5. , 3. , 1.6, 0.2, 'Iris-setosa'),
       (5. , 3.4, 1.6, 0.4, 'Iris-setosa'),
       (5.2, 3.5, 1.5, 0.2, 'Iris-setosa'),
       (5.2, 3.4, 1.4, 0.2, 'Iris-setosa'),
       (4.7, 3.2, 1.6, 0.2, 'Iris-setosa'),
       (4.8, 3.1, 1.6, 0.2, 'Iris-setosa'),
       (5.4, 3.4, 1.5, 0.4, 'Iris-setosa'),
       (5.2, 4.1, 1.5, 0.1, 'Iris-setosa'),
       (5.5, 4.2, 1.4, 0.2, 'Iris-setosa'),
       (4.9, 3.1, 1.5, 0.1, 'Iris-setosa'),
       (5. , 3.2, 1.2, 0.2, 'Iris-setosa'),
       (5.5, 3.5, 1.3, 0.2, 'Iris-setosa'),
       (4.9, 3.1, 1.5, 0.1, 'Iris-setosa'),
       (4.4, 3. , 1.3, 0.2, 'Iris-setosa'),
       (5.1, 3.4, 1.5, 0.2, 'Iris-setosa'),
```

```
(5. , 3.5, 1.3, 0.3, 'Iris-setosa'),
(4.5, 2.3, 1.3, 0.3, 'Iris-setosa'),
(4.4, 3.2, 1.3, 0.2, 'Iris-setosa'),
(5. , 3.5, 1.6, 0.6, 'Iris-setosa'),
(5.1, 3.8, 1.9, 0.4, 'Iris-setosa'),
(4.8, 3. , 1.4, 0.3, 'Iris-setosa'),
(5.1, 3.8, 1.6, 0.2, 'Iris-setosa'),
(4.6, 3.2, 1.4, 0.2, 'Iris-setosa'),
(5.3, 3.7, 1.5, 0.2, 'Iris-setosa'),
(5. , 3.3, 1.4, 0.2, 'Iris-setosa'),
(7. , 3.2, 4.7, 1.4, 'Iris-versicolor'),
(6.4, 3.2, 4.5, 1.5, 'Iris-versicolor'),
(6.9, 3.1, 4.9, 1.5, 'Iris-versicolor'),
(5.5, 2.3, 4. , 1.3, 'Iris-versicolor'),
(6.5, 2.8, 4.6, 1.5, 'Iris-versicolor'),
(5.7, 2.8, 4.5, 1.3, 'Iris-versicolor'),
(6.3, 3.3, 4.7, 1.6, 'Iris-versicolor'),
(4.9, 2.4, 3.3, 1. , 'Iris-versicolor'),
(6.6, 2.9, 4.6, 1.3, 'Iris-versicolor'),
(5.2, 2.7, 3.9, 1.4, 'Iris-versicolor'),
(5. , 2. , 3.5, 1. , 'Iris-versicolor'),
(5.9, 3. , 4.2, 1.5, 'Iris-versicolor'),
(6. , 2.2, 4. , 1. , 'Iris-versicolor'),
(6.1, 2.9, 4.7, 1.4, 'Iris-versicolor'),
(5.6, 2.9, 3.6, 1.3, 'Iris-versicolor'),
(6.7, 3.1, 4.4, 1.4, 'Iris-versicolor'),
(5.6, 3. , 4.5, 1.5, 'Iris-versicolor'),
(5.8, 2.7, 4.1, 1. , 'Iris-versicolor'),
(6.2, 2.2, 4.5, 1.5, 'Iris-versicolor'),
(5.6, 2.5, 3.9, 1.1, 'Iris-versicolor'),
(5.9, 3.2, 4.8, 1.8, 'Iris-versicolor'),
(6.1, 2.8, 4. , 1.3, 'Iris-versicolor'),
(6.3, 2.5, 4.9, 1.5, 'Iris-versicolor'),
(6.1, 2.8, 4.7, 1.2, 'Iris-versicolor'),
(6.4, 2.9, 4.3, 1.3, 'Iris-versicolor'),
(6.6, 3. , 4.4, 1.4, 'Iris-versicolor'),
(6.8, 2.8, 4.8, 1.4, 'Iris-versicolor'),
(6.7, 3. , 5. , 1.7, 'Iris-versicolor'),
(6. , 2.9, 4.5, 1.5, 'Iris-versicolor'),
(5.7, 2.6, 3.5, 1. , 'Iris-versicolor'),
(5.5, 2.4, 3.8, 1.1, 'Iris-versicolor'),
```

```
(5.5, 2.4, 3.7, 1. , 'Iris-versicolor'),
(5.8, 2.7, 3.9, 1.2, 'Iris-versicolor'),
(6. , 2.7, 5.1, 1.6, 'Iris-versicolor'),
(5.4, 3. , 4.5, 1.5, 'Iris-versicolor'),
(6. , 3.4, 4.5, 1.6, 'Iris-versicolor'),
(6.7, 3.1, 4.7, 1.5, 'Iris-versicolor'),
(6.3, 2.3, 4.4, 1.3, 'Iris-versicolor'),
(5.6, 3. , 4.1, 1.3, 'Iris-versicolor'),
(5.5, 2.5, 4. , 1.3, 'Iris-versicolor'),
(5.5, 2.6, 4.4, 1.2, 'Iris-versicolor'),
(6.1, 3. , 4.6, 1.4, 'Iris-versicolor'),
(5.8, 2.6, 4. , 1.2, 'Iris-versicolor'),
(5. , 2.3, 3.3, 1. , 'Iris-versicolor'),
(5.6, 2.7, 4.2, 1.3, 'Iris-versicolor'),
(5.7, 3. , 4.2, 1.2, 'Iris-versicolor'),
(5.7, 2.9, 4.2, 1.3, 'Iris-versicolor'),
(6.2, 2.9, 4.3, 1.3, 'Iris-versicolor'),
(5.1, 2.5, 3. , 1.1, 'Iris-versicolor'),
(5.7, 2.8, 4.1, 1.3, 'Iris-versicolor'),
(6.3, 3.3, 6. , 2.5, 'Iris-virginica'),
(5.8, 2.7, 5.1, 1.9, 'Iris-virginica'),
(7.1, 3. , 5.9, 2.1, 'Iris-virginica'),
(6.3, 2.9, 5.6, 1.8, 'Iris-virginica'),
(6.5, 3. , 5.8, 2.2, 'Iris-virginica'),
(7.6, 3. , 6.6, 2.1, 'Iris-virginica'),
(4.9, 2.5, 4.5, 1.7, 'Iris-virginica'),
(7.3, 2.9, 6.3, 1.8, 'Iris-virginica'),
(6.7, 2.5, 5.8, 1.8, 'Iris-virginica'),
(7.2, 3.6, 6.1, 2.5, 'Iris-virginica'),
(6.5, 3.2, 5.1, 2. , 'Iris-virginica'),
(6.4, 2.7, 5.3, 1.9, 'Iris-virginica'),
(6.8, 3. , 5.5, 2.1, 'Iris-virginica'),
(5.7, 2.5, 5. , 2. , 'Iris-virginica'),
(5.8, 2.8, 5.1, 2.4, 'Iris-virginica'),
(6.4, 3.2, 5.3, 2.3, 'Iris-virginica'),
(6.5, 3. , 5.5, 1.8, 'Iris-virginica'),
(7.7, 3.8, 6.7, 2.2, 'Iris-virginica'),
(7.7, 2.6, 6.9, 2.3, 'Iris-virginica'),
(6. , 2.2, 5. , 1.5, 'Iris-virginica'),
(6.9, 3.2, 5.7, 2.3, 'Iris-virginica'),
(5.6, 2.8, 4.9, 2. , 'Iris-virginica'),
```

```
       (7.7, 2.8, 6.7, 2. , 'Iris-virginica'),
       (6.3, 2.7, 4.9, 1.8, 'Iris-virginica'),
       (6.7, 3.3, 5.7, 2.1, 'Iris-virginica'),
       (7.2, 3.2, 6. , 1.8, 'Iris-virginica'),
       (6.2, 2.8, 4.8, 1.8, 'Iris-virginica'),
       (6.1, 3. , 4.9, 1.8, 'Iris-virginica'),
       (6.4, 2.8, 5.6, 2.1, 'Iris-virginica'),
       (7.2, 3. , 5.8, 1.6, 'Iris-virginica'),
       (7.4, 2.8, 6.1, 1.9, 'Iris-virginica'),
       (7.9, 3.8, 6.4, 2. , 'Iris-virginica'),
       (6.4, 2.8, 5.6, 2.2, 'Iris-virginica'),
       (6.3, 2.8, 5.1, 1.5, 'Iris-virginica'),
       (6.1, 2.6, 5.6, 1.4, 'Iris-virginica'),
       (7.7, 3. , 6.1, 2.3, 'Iris-virginica'),
       (6.3, 3.4, 5.6, 2.4, 'Iris-virginica'),
       (6.4, 3.1, 5.5, 1.8, 'Iris-virginica'),
       (6. , 3. , 4.8, 1.8, 'Iris-virginica'),
       (6.9, 3.1, 5.4, 2.1, 'Iris-virginica'),
       (6.7, 3.1, 5.6, 2.4, 'Iris-virginica'),
       (6.9, 3.1, 5.1, 2.3, 'Iris-virginica'),
       (5.8, 2.7, 5.1, 1.9, 'Iris-virginica'),
       (6.8, 3.2, 5.9, 2.3, 'Iris-virginica'),
       (6.7, 3.3, 5.7, 2.5, 'Iris-virginica'),
       (6.7, 3. , 5.2, 2.3, 'Iris-virginica'),
       (6.3, 2.5, 5. , 1.9, 'Iris-virginica'),
       (6.5, 3. , 5.2, 2. , 'Iris-virginica'),
       (6.2, 3.4, 5.4, 2.3, 'Iris-virginica'),
       (5.9, 3. , 5.1, 1.8, 'Iris-virginica')],
      dtype=[('sepal_length', '<f8'), ('sepal_width', '<f8'), ('petal_length', '<f8'), ('petal_width', '<f8'), ('species', '<
U15')])
```

3A. How to convert a 1d array of tuples to a 2d numpy array

Exercise: convert the `iris` 1D array to a numeric-only 2D array `iris_data` by omitting the species text field. Create a `iris_label` 1D array containing only the species text field. Keep the same indexing/order as in the original array.

In [8]:
```
import numbers
import numpy as np
```

```
## Write your solution here

def numerical_data_function(r):
   rows = []
   labels = []
   for item in r:
      row = [column for column in item if isinstance(column, numbers.Number)]
      rows.append(row)
      label = [column for column in item if isinstance(column, str)]
      labels.append(label[0])
   return (np.array(rows, dtype=float), np.array(labels, dtype=str))

def print_geometry_function(x):
    for item in x:
        print(f"[{', '.join(map(str,item))}]")

def print_labels_function(x):
    print(f"[{', '.join(map(str,x))}]")

(geometric_values, label_values) = numerical_data_function(iris)
print_geometry_function(geometric_values)
print_labels_function(label_values)
```

```
[5.1, 3.5, 1.4, 0.2]
[4.9, 3.0, 1.4, 0.2]
[4.7, 3.2, 1.3, 0.2]
[4.6, 3.1, 1.5, 0.2]
[5.0, 3.6, 1.4, 0.2]
[5.4, 3.9, 1.7, 0.4]
[4.6, 3.4, 1.4, 0.3]
[5.0, 3.4, 1.5, 0.2]
[4.4, 2.9, 1.4, 0.2]
[4.9, 3.1, 1.5, 0.1]
[5.4, 3.7, 1.5, 0.2]
[4.8, 3.4, 1.6, 0.2]
[4.8, 3.0, 1.4, 0.1]
[4.3, 3.0, 1.1, 0.1]
[5.8, 4.0, 1.2, 0.2]
[5.7, 4.4, 1.5, 0.4]
[5.4, 3.9, 1.3, 0.4]
[5.1, 3.5, 1.4, 0.3]
[5.7, 3.8, 1.7, 0.3]
[5.1, 3.8, 1.5, 0.3]
[5.4, 3.4, 1.7, 0.2]
[5.1, 3.7, 1.5, 0.4]
[4.6, 3.6, 1.0, 0.2]
[5.1, 3.3, 1.7, 0.5]
[4.8, 3.4, 1.9, 0.2]
[5.0, 3.0, 1.6, 0.2]
[5.0, 3.4, 1.6, 0.4]
[5.2, 3.5, 1.5, 0.2]
[5.2, 3.4, 1.4, 0.2]
[4.7, 3.2, 1.6, 0.2]
[4.8, 3.1, 1.6, 0.2]
[5.4, 3.4, 1.5, 0.4]
[5.2, 4.1, 1.5, 0.1]
[5.5, 4.2, 1.4, 0.2]
[4.9, 3.1, 1.5, 0.1]
[5.0, 3.2, 1.2, 0.2]
[5.5, 3.5, 1.3, 0.2]
[4.9, 3.1, 1.5, 0.1]
[4.4, 3.0, 1.3, 0.2]
[5.1, 3.4, 1.5, 0.2]
[5.0, 3.5, 1.3, 0.3]
```

```
[4.5, 2.3, 1.3, 0.3]
[4.4, 3.2, 1.3, 0.2]
[5.0, 3.5, 1.6, 0.6]
[5.1, 3.8, 1.9, 0.4]
[4.8, 3.0, 1.4, 0.3]
[5.1, 3.8, 1.6, 0.2]
[4.6, 3.2, 1.4, 0.2]
[5.3, 3.7, 1.5, 0.2]
[5.0, 3.3, 1.4, 0.2]
[7.0, 3.2, 4.7, 1.4]
[6.4, 3.2, 4.5, 1.5]
[6.9, 3.1, 4.9, 1.5]
[5.5, 2.3, 4.0, 1.3]
[6.5, 2.8, 4.6, 1.5]
[5.7, 2.8, 4.5, 1.3]
[6.3, 3.3, 4.7, 1.6]
[4.9, 2.4, 3.3, 1.0]
[6.6, 2.9, 4.6, 1.3]
[5.2, 2.7, 3.9, 1.4]
[5.0, 2.0, 3.5, 1.0]
[5.9, 3.0, 4.2, 1.5]
[6.0, 2.2, 4.0, 1.0]
[6.1, 2.9, 4.7, 1.4]
[5.6, 2.9, 3.6, 1.3]
[6.7, 3.1, 4.4, 1.4]
[5.6, 3.0, 4.5, 1.5]
[5.8, 2.7, 4.1, 1.0]
[6.2, 2.2, 4.5, 1.5]
[5.6, 2.5, 3.9, 1.1]
[5.9, 3.2, 4.8, 1.8]
[6.1, 2.8, 4.0, 1.3]
[6.3, 2.5, 4.9, 1.5]
[6.1, 2.8, 4.7, 1.2]
[6.4, 2.9, 4.3, 1.3]
[6.6, 3.0, 4.4, 1.4]
[6.8, 2.8, 4.8, 1.4]
[6.7, 3.0, 5.0, 1.7]
[6.0, 2.9, 4.5, 1.5]
[5.7, 2.6, 3.5, 1.0]
[5.5, 2.4, 3.8, 1.1]
[5.5, 2.4, 3.7, 1.0]
```

```
[5.8, 2.7, 3.9, 1.2]
[6.0, 2.7, 5.1, 1.6]
[5.4, 3.0, 4.5, 1.5]
[6.0, 3.4, 4.5, 1.6]
[6.7, 3.1, 4.7, 1.5]
[6.3, 2.3, 4.4, 1.3]
[5.6, 3.0, 4.1, 1.3]
[5.5, 2.5, 4.0, 1.3]
[5.5, 2.6, 4.4, 1.2]
[6.1, 3.0, 4.6, 1.4]
[5.8, 2.6, 4.0, 1.2]
[5.0, 2.3, 3.3, 1.0]
[5.6, 2.7, 4.2, 1.3]
[5.7, 3.0, 4.2, 1.2]
[5.7, 2.9, 4.2, 1.3]
[6.2, 2.9, 4.3, 1.3]
[5.1, 2.5, 3.0, 1.1]
[5.7, 2.8, 4.1, 1.3]
[6.3, 3.3, 6.0, 2.5]
[5.8, 2.7, 5.1, 1.9]
[7.1, 3.0, 5.9, 2.1]
[6.3, 2.9, 5.6, 1.8]
[6.5, 3.0, 5.8, 2.2]
[7.6, 3.0, 6.6, 2.1]
[4.9, 2.5, 4.5, 1.7]
[7.3, 2.9, 6.3, 1.8]
[6.7, 2.5, 5.8, 1.8]
[7.2, 3.6, 6.1, 2.5]
[6.5, 3.2, 5.1, 2.0]
[6.4, 2.7, 5.3, 1.9]
[6.8, 3.0, 5.5, 2.1]
[5.7, 2.5, 5.0, 2.0]
[5.8, 2.8, 5.1, 2.4]
[6.4, 3.2, 5.3, 2.3]
[6.5, 3.0, 5.5, 1.8]
[7.7, 3.8, 6.7, 2.2]
[7.7, 2.6, 6.9, 2.3]
[6.0, 2.2, 5.0, 1.5]
[6.9, 3.2, 5.7, 2.3]
[5.6, 2.8, 4.9, 2.0]
[7.7, 2.8, 6.7, 2.0]
```

```
[6.3, 2.7, 4.9, 1.8]
[6.7, 3.3, 5.7, 2.1]
[7.2, 3.2, 6.0, 1.8]
[6.2, 2.8, 4.8, 1.8]
[6.1, 3.0, 4.9, 1.8]
[6.4, 2.8, 5.6, 2.1]
[7.2, 3.0, 5.8, 1.6]
[7.4, 2.8, 6.1, 1.9]
[7.9, 3.8, 6.4, 2.0]
[6.4, 2.8, 5.6, 2.2]
[6.3, 2.8, 5.1, 1.5]
[6.1, 2.6, 5.6, 1.4]
[7.7, 3.0, 6.1, 2.3]
[6.3, 3.4, 5.6, 2.4]
[6.4, 3.1, 5.5, 1.8]
[6.0, 3.0, 4.8, 1.8]
[6.9, 3.1, 5.4, 2.1]
[6.7, 3.1, 5.6, 2.4]
[6.9, 3.1, 5.1, 2.3]
[5.8, 2.7, 5.1, 1.9]
[6.8, 3.2, 5.9, 2.3]
[6.7, 3.3, 5.7, 2.5]
[6.7, 3.0, 5.2, 2.3]
[6.3, 2.5, 5.0, 1.9]
[6.5, 3.0, 5.2, 2.0]
[6.2, 3.4, 5.4, 2.3]
[5.9, 3.0, 5.1, 1.8]
[Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-seto
sa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-s
etosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iri
s-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa,
Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setosa, Iris-setos
a, Iris-setosa, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-vers
icolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor,
Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-ver
sicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor,
Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-ver
sicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor,
Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-virginica, Iris-virg
inica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-vir
ginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-vi
```

```
rginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-v
irginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-
virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris
-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iris-virginica, Iri
s-virginica]
```

3B. Split the IRIS dataset by Label

Split the dataset in `iris_data` according to the labels `iris_labels`.

In [9]:

```
# Write your solution here
def create_dictionary_function(r):
    results = []
    labels = []
    for index, item in enumerate(r):
        row = [column for column in item if isinstance(column, numbers.Number)]
        label = [column for column in item if isinstance(column, str)]
        iris_dictionary = { "index" : index, "label" : label[0] , "values" : row }
        results.append(iris_dictionary)
    return np.array(results, dtype=dict)

def create_iris_label_set_function(r):
    rows = []
    labels = []
    for item in r:
        label = [column for column in item if isinstance(column, str)]
        labels.append(label[0])
    labels_array = np.array(labels, dtype=str)
    return list(set(labels_array))

def initialise_iris_dict_function(x):
    value_dict = {}
    for value in x:
        value_dict.update({ value : []})
    return value_dict


labels = create_iris_label_set_function(iris)
```

```
iris_dict = initialise_iris_dict_function(labels)

values_dict_list = create_dictionary_function(iris)

def loading_value(iris_dict, values_dict_array):
   for row in values_dict_array:
      iris_dict[row["label"]].append({"sepal_length": row['values'][0], "sepal_width" : row['values'][1], "petal_length" : row

loading_value(iris_dict, values_dict_list)

def print_dictionary_function(x):
   for key, values in x.items():
      print(f"{key}: ")
      for value in values:
         print(f" {value} ")

print_dictionary_function(iris_dict)
```

```
Iris-versicolor:
 {'sepal_length': np.float64(7.0), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(4.7), 'petal_width': np.float64
(1.4)}
 {'sepal_length': np.float64(6.4), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(4.5), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(6.9), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(4.9), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(5.5), 'sepal_width': np.float64(2.3), 'petal_length': np.float64(4.0), 'petal_width': np.float64
(1.3)}
 {'sepal_length': np.float64(6.5), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.6), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(5.7), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.5), 'petal_width': np.float64
(1.3)}
 {'sepal_length': np.float64(6.3), 'sepal_width': np.float64(3.3), 'petal_length': np.float64(4.7), 'petal_width': np.float64
(1.6)}
 {'sepal_length': np.float64(4.9), 'sepal_width': np.float64(2.4), 'petal_length': np.float64(3.3), 'petal_width': np.float64
(1.0)}
 {'sepal_length': np.float64(6.6), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(4.6), 'petal_width': np.float64
(1.3)}
 {'sepal_length': np.float64(5.2), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(3.9), 'petal_width': np.float64
(1.4)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(2.0), 'petal_length': np.float64(3.5), 'petal_width': np.float64
(1.0)}
 {'sepal_length': np.float64(5.9), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.2), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(6.0), 'sepal_width': np.float64(2.2), 'petal_length': np.float64(4.0), 'petal_width': np.float64
(1.0)}
 {'sepal_length': np.float64(6.1), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(4.7), 'petal_width': np.float64
(1.4)}
 {'sepal_length': np.float64(5.6), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(3.6), 'petal_width': np.float64
(1.3)}
 {'sepal_length': np.float64(6.7), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(4.4), 'petal_width': np.float64
(1.4)}
 {'sepal_length': np.float64(5.6), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.5), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(5.8), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(4.1), 'petal_width': np.float64
(1.0)}
 {'sepal_length': np.float64(6.2), 'sepal_width': np.float64(2.2), 'petal_length': np.float64(4.5), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(5.6), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(3.9), 'petal_width': np.float64
(1.1)}
```

{'sepal_length': np.float64(5.9), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(4.8), 'petal_width': np.float64(1.8)}
{'sepal_length': np.float64(6.1), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.0), 'petal_width': np.float64(1.3)}
{'sepal_length': np.float64(6.3), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(4.9), 'petal_width': np.float64(1.5)}
{'sepal_length': np.float64(6.1), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.7), 'petal_width': np.float64(1.2)}
{'sepal_length': np.float64(6.4), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(4.3), 'petal_width': np.float64(1.3)}
{'sepal_length': np.float64(6.6), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.4), 'petal_width': np.float64(1.4)}
{'sepal_length': np.float64(6.8), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.8), 'petal_width': np.float64(1.4)}
{'sepal_length': np.float64(6.7), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.0), 'petal_width': np.float64(1.7)}
{'sepal_length': np.float64(6.0), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(4.5), 'petal_width': np.float64(1.5)}
{'sepal_length': np.float64(5.7), 'sepal_width': np.float64(2.6), 'petal_length': np.float64(3.5), 'petal_width': np.float64(1.0)}
{'sepal_length': np.float64(5.5), 'sepal_width': np.float64(2.4), 'petal_length': np.float64(3.8), 'petal_width': np.float64(1.1)}
{'sepal_length': np.float64(5.5), 'sepal_width': np.float64(2.4), 'petal_length': np.float64(3.7), 'petal_width': np.float64(1.0)}
{'sepal_length': np.float64(5.8), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(3.9), 'petal_width': np.float64(1.2)}
{'sepal_length': np.float64(6.0), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(5.1), 'petal_width': np.float64(1.6)}
{'sepal_length': np.float64(5.4), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.5), 'petal_width': np.float64(1.5)}
{'sepal_length': np.float64(6.0), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(4.5), 'petal_width': np.float64(1.6)}
{'sepal_length': np.float64(6.7), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(4.7), 'petal_width': np.float64(1.5)}
{'sepal_length': np.float64(6.3), 'sepal_width': np.float64(2.3), 'petal_length': np.float64(4.4), 'petal_width': np.float64(1.3)}
{'sepal_length': np.float64(5.6), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.1), 'petal_width': np.float64(1.3)}
{'sepal_length': np.float64(5.5), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(4.0), 'petal_width': np.float64(1.3)}
{'sepal_length': np.float64(5.5), 'sepal_width': np.float64(2.6), 'petal_length': np.float64(4.4), 'petal_width': np.float64

(1.2)}
 {'sepal_length': np.float64(6.1), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.6), 'petal_width': np.float64
(1.4)}
 {'sepal_length': np.float64(5.8), 'sepal_width': np.float64(2.6), 'petal_length': np.float64(4.0), 'petal_width': np.float64
(1.2)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(2.3), 'petal_length': np.float64(3.3), 'petal_width': np.float64
(1.0)}
 {'sepal_length': np.float64(5.6), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(4.2), 'petal_width': np.float64
(1.3)}
 {'sepal_length': np.float64(5.7), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.2), 'petal_width': np.float64
(1.2)}
 {'sepal_length': np.float64(5.7), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(4.2), 'petal_width': np.float64
(1.3)}
 {'sepal_length': np.float64(6.2), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(4.3), 'petal_width': np.float64
(1.3)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(3.0), 'petal_width': np.float64
(1.1)}
 {'sepal_length': np.float64(5.7), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.1), 'petal_width': np.float64
(1.3)}
Iris-setosa:
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.5), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.9), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.7), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(1.3), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.6), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.6), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.4), 'sepal_width': np.float64(3.9), 'petal_length': np.float64(1.7), 'petal_width': np.float64
(0.4)}
 {'sepal_length': np.float64(4.6), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.3)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.4), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.9), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.1)}
 {'sepal_length': np.float64(5.4), 'sepal_width': np.float64(3.7), 'petal_length': np.float64(1.5), 'petal_width': np.float64

(0.2)}
 {'sepal_length': np.float64(4.8), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.6), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.8), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.1)}
 {'sepal_length': np.float64(4.3), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(1.1), 'petal_width': np.float64
(0.1)}
 {'sepal_length': np.float64(5.8), 'sepal_width': np.float64(4.0), 'petal_length': np.float64(1.2), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.7), 'sepal_width': np.float64(4.4), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.4)}
 {'sepal_length': np.float64(5.4), 'sepal_width': np.float64(3.9), 'petal_length': np.float64(1.3), 'petal_width': np.float64
(0.4)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.5), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.3)}
 {'sepal_length': np.float64(5.7), 'sepal_width': np.float64(3.8), 'petal_length': np.float64(1.7), 'petal_width': np.float64
(0.3)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.8), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.3)}
 {'sepal_length': np.float64(5.4), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.7), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.7), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.4)}
 {'sepal_length': np.float64(4.6), 'sepal_width': np.float64(3.6), 'petal_length': np.float64(1.0), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.3), 'petal_length': np.float64(1.7), 'petal_width': np.float64
(0.5)}
 {'sepal_length': np.float64(4.8), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.9), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(1.6), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.6), 'petal_width': np.float64
(0.4)}
 {'sepal_length': np.float64(5.2), 'sepal_width': np.float64(3.5), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.2), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.7), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(1.6), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.8), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(1.6), 'petal_width': np.float64
(0.2)}

```
 {'sepal_length': np.float64(5.4), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.4)}
 {'sepal_length': np.float64(5.2), 'sepal_width': np.float64(4.1), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.1)}
 {'sepal_length': np.float64(5.5), 'sepal_width': np.float64(4.2), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.9), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.1)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(1.2), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.5), 'sepal_width': np.float64(3.5), 'petal_length': np.float64(1.3), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.9), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.1)}
 {'sepal_length': np.float64(4.4), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(1.3), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.5), 'petal_length': np.float64(1.3), 'petal_width': np.float64
(0.3)}
 {'sepal_length': np.float64(4.5), 'sepal_width': np.float64(2.3), 'petal_length': np.float64(1.3), 'petal_width': np.float64
(0.3)}
 {'sepal_length': np.float64(4.4), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(1.3), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.5), 'petal_length': np.float64(1.6), 'petal_width': np.float64
(0.6)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.8), 'petal_length': np.float64(1.9), 'petal_width': np.float64
(0.4)}
 {'sepal_length': np.float64(4.8), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.3)}
 {'sepal_length': np.float64(5.1), 'sepal_width': np.float64(3.8), 'petal_length': np.float64(1.6), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(4.6), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.3), 'sepal_width': np.float64(3.7), 'petal_length': np.float64(1.5), 'petal_width': np.float64
(0.2)}
 {'sepal_length': np.float64(5.0), 'sepal_width': np.float64(3.3), 'petal_length': np.float64(1.4), 'petal_width': np.float64
(0.2)}
Iris-virginica:
 {'sepal_length': np.float64(6.3), 'sepal_width': np.float64(3.3), 'petal_length': np.float64(6.0), 'petal_width': np.float64
(2.5)}
```

```
 {'sepal_length': np.float64(5.8), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(5.1), 'petal_width': np.float64
(1.9)}
 {'sepal_length': np.float64(7.1), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.9), 'petal_width': np.float64
(2.1)}
 {'sepal_length': np.float64(6.3), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(5.6), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.5), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.8), 'petal_width': np.float64
(2.2)}
 {'sepal_length': np.float64(7.6), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(6.6), 'petal_width': np.float64
(2.1)}
 {'sepal_length': np.float64(4.9), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(4.5), 'petal_width': np.float64
(1.7)}
 {'sepal_length': np.float64(7.3), 'sepal_width': np.float64(2.9), 'petal_length': np.float64(6.3), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.7), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(5.8), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(7.2), 'sepal_width': np.float64(3.6), 'petal_length': np.float64(6.1), 'petal_width': np.float64
(2.5)}
 {'sepal_length': np.float64(6.5), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(5.1), 'petal_width': np.float64
(2.0)}
 {'sepal_length': np.float64(6.4), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(5.3), 'petal_width': np.float64
(1.9)}
 {'sepal_length': np.float64(6.8), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.5), 'petal_width': np.float64
(2.1)}
 {'sepal_length': np.float64(5.7), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(5.0), 'petal_width': np.float64
(2.0)}
 {'sepal_length': np.float64(5.8), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(5.1), 'petal_width': np.float64
(2.4)}
 {'sepal_length': np.float64(6.4), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(5.3), 'petal_width': np.float64
(2.3)}
 {'sepal_length': np.float64(6.5), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.5), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(7.7), 'sepal_width': np.float64(3.8), 'petal_length': np.float64(6.7), 'petal_width': np.float64
(2.2)}
 {'sepal_length': np.float64(7.7), 'sepal_width': np.float64(2.6), 'petal_length': np.float64(6.9), 'petal_width': np.float64
(2.3)}
 {'sepal_length': np.float64(6.0), 'sepal_width': np.float64(2.2), 'petal_length': np.float64(5.0), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(6.9), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(5.7), 'petal_width': np.float64
(2.3)}
 {'sepal_length': np.float64(5.6), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.9), 'petal_width': np.float64
```

(2.0)}
 {'sepal_length': np.float64(7.7), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(6.7), 'petal_width': np.float64
(2.0)}
 {'sepal_length': np.float64(6.3), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(4.9), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.7), 'sepal_width': np.float64(3.3), 'petal_length': np.float64(5.7), 'petal_width': np.float64
(2.1)}
 {'sepal_length': np.float64(7.2), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(6.0), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.2), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(4.8), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.1), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.9), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.4), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(5.6), 'petal_width': np.float64
(2.1)}
 {'sepal_length': np.float64(7.2), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.8), 'petal_width': np.float64
(1.6)}
 {'sepal_length': np.float64(7.4), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(6.1), 'petal_width': np.float64
(1.9)}
 {'sepal_length': np.float64(7.9), 'sepal_width': np.float64(3.8), 'petal_length': np.float64(6.4), 'petal_width': np.float64
(2.0)}
 {'sepal_length': np.float64(6.4), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(5.6), 'petal_width': np.float64
(2.2)}
 {'sepal_length': np.float64(6.3), 'sepal_width': np.float64(2.8), 'petal_length': np.float64(5.1), 'petal_width': np.float64
(1.5)}
 {'sepal_length': np.float64(6.1), 'sepal_width': np.float64(2.6), 'petal_length': np.float64(5.6), 'petal_width': np.float64
(1.4)}
 {'sepal_length': np.float64(7.7), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(6.1), 'petal_width': np.float64
(2.3)}
 {'sepal_length': np.float64(6.3), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(5.6), 'petal_width': np.float64
(2.4)}
 {'sepal_length': np.float64(6.4), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(5.5), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.0), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(4.8), 'petal_width': np.float64
(1.8)}
 {'sepal_length': np.float64(6.9), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(5.4), 'petal_width': np.float64
(2.1)}
 {'sepal_length': np.float64(6.7), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(5.6), 'petal_width': np.float64
(2.4)}
 {'sepal_length': np.float64(6.9), 'sepal_width': np.float64(3.1), 'petal_length': np.float64(5.1), 'petal_width': np.float64
(2.3)}

```
 {'sepal_length': np.float64(5.8), 'sepal_width': np.float64(2.7), 'petal_length': np.float64(5.1), 'petal_width': np.float64
(1.9)}
 {'sepal_length': np.float64(6.8), 'sepal_width': np.float64(3.2), 'petal_length': np.float64(5.9), 'petal_width': np.float64
(2.3)}
 {'sepal_length': np.float64(6.7), 'sepal_width': np.float64(3.3), 'petal_length': np.float64(5.7), 'petal_width': np.float64
(2.5)}
 {'sepal_length': np.float64(6.7), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.2), 'petal_width': np.float64
(2.3)}
 {'sepal_length': np.float64(6.3), 'sepal_width': np.float64(2.5), 'petal_length': np.float64(5.0), 'petal_width': np.float64
(1.9)}
 {'sepal_length': np.float64(6.5), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.2), 'petal_width': np.float64
(2.0)}
 {'sepal_length': np.float64(6.2), 'sepal_width': np.float64(3.4), 'petal_length': np.float64(5.4), 'petal_width': np.float64
(2.3)}
 {'sepal_length': np.float64(5.9), 'sepal_width': np.float64(3.0), 'petal_length': np.float64(5.1), 'petal_width': np.float64
(1.8)}
```

4. Compute statistics with Numpy

1. For each flower species compute the key statistics for `sepal width`:

- mean
- median
- standard deviation

Answer the following questions:

a) Which is the flower type with the largest mean value for `sepal_width`?

b) Which is the flower type with the smallest median value for `sepal_width`?

In [10]:
```
### Compute mean, median, and standard deviation for `sepal width` here:

def extract_sepal_width_function (iris_dict):
    iris_sepal_width_rows = []
    for species, irises in iris_dict.items():
        values = [iris['sepal_width'] for iris in irises]
        iris_sepal_width_rows.append({species: values})
```

```
    return iris_sepal_width_rows

def compute_sepal_width_stats_function(iris_dict):
    iris_stats = []
    for species, irises in iris_dict.items():
        values = [row['sepal_width'] for row in irises]
        iris_stat = {"species" : species, "mean": np.mean(values), "median" : np.median(values), "std" : np.std(values)}
        iris_stats.append(iris_stat)
    return iris_stats

iris_sepal_width_rows = extract_sepal_width_function(iris_dict)
iris_stats = compute_sepal_width_stats_function(iris_dict)

sorted_by_mean_desc = sorted(iris_stats, key=lambda iris_stat: iris_stat['mean'], reverse=True)
print(f"The species with the largest mean value for sepal width is {sorted_by_mean_desc[0]['species']}")

sorted_by_median_asc = sorted(iris_stats, key=lambda iris_stat: iris_stat['median'], reverse=False)
print(f"The species with the smallest median value for sepal width is {sorted_by_median_asc[0]['species']}")
```

```
The species with the largest mean value for sepal width is Iris-setosa
The species with the smallest median value for sepal width is Iris-versicolor
```

2. Compute the correlation matrix between `sepal_width` and `petal_length` for the three scpecies. Which is the species that shows the highest correlation among the two parameters?

In [11]:
```
### Compute the correlation coefficients here

def extract_petal_length_function (iris_dict):
    iris_petal_rows = []
    for species, irises in iris_dict.items():
        values = [iris['petal_length'] for iris in irises]
        iris_petal_rows.append({species: values})
    return iris_petal_rows

iris_petal_length_rows = extract_petal_length_function(iris_dict)


def compute_correlation_matrix_function(iris_sepal_width_rows, iris_petal_length_rows):
    correlations = {}
    for iris_sepal_row in iris_sepal_width_rows:
```

```
        current_key = list(iris_sepal_row.keys())[0]
        iris_petal_length_row = list(filter(lambda d: current_key in d, iris_petal_length_rows))

        width_values = [list(d.values())[0] for d in iris_sepal_width_rows]
        length_values = [list(d.values())[0] for d in iris_petal_length_rows]
        correlations[current_key] = np.corrcoef(width_values, length_values)[0, 1]
    return correlations

correlations_matrix = compute_correlation_matrix_function(iris_sepal_width_rows, iris_petal_length_rows)
sorted_correlations_matrix = dict(sorted(correlations_matrix.items(), key=lambda item: item[1], reverse=True))
highest_correlation_species = next(iter(sorted_correlations_matrix))

print(f"{highest_correlation_species} is the species that shows the highest correlation among sepal width and petal length.")
```

```
Iris-versicolor is the species that shows the highest correlation among sepal width and petal length.
```

5. How to create a new column from existing columns of a numpy array

Create a new column for the flower volume V in `iris`, where volume is computing according to this formula (the assumption here is that the shape of the iris flower is approximately conical):

$$Volume = \frac{\pi \times SepalLength^2 \times PetalLength}{3}$$

In [12]:

```
import math

pi_value = math.pi

# Input
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris_2d = np.genfromtxt(url, delimiter=',', dtype='object')

# Write your solution here

headers = ['volume', 'sepal length', 'sepal width', 'petal length', 'petal width', 'species']
iris_2d_with_volume_rows = []
for value in iris_2d:
    sepal_length = float(value[0].decode())
    petal_length = float(value[2].decode())
```

```
    volume = (pi_value * (sepal_length ** 2) * petal_length) / 3
    current_row = new_arr = np.insert(value, 0, volume)
    iris_2d_with_volume_rows.append(current_row)
# Print header
print(f"{headers[0]:<15}{headers[1]:<15}{headers[2]:<15}{headers[3]:<15}{headers[4]:<15}{headers[5]:<15}")
print("-" * 90)

for current_row in iris_2d_with_volume_rows:
    decoded = [x.decode() if isinstance(x, bytes) else x for x in current_row]
    print(f"{float(decoded[0]):<15.4f}{decoded[1]:<15}{decoded[2]:<15}{decoded[3]:<15}{decoded[4]:<15}{decoded[5]:<15}")
```

volume	sepal length	sepal width	petal length	petal width	species
38.1327	5.1	3.5	1.4	0.2	Iris-setosa
35.2005	4.9	3.0	1.4	0.2	Iris-setosa
30.0724	4.7	3.2	1.3	0.2	Iris-setosa
33.2381	4.6	3.1	1.5	0.2	Iris-setosa
36.6519	5.0	3.6	1.4	0.2	Iris-setosa
51.9117	5.4	3.9	1.7	0.4	Iris-setosa
31.0222	4.6	3.4	1.4	0.3	Iris-setosa
39.2699	5.0	3.4	1.5	0.2	Iris-setosa
28.3832	4.4	2.9	1.4	0.2	Iris-setosa
37.7148	4.9	3.1	1.5	0.1	Iris-setosa
45.8044	5.4	3.7	1.5	0.2	Iris-setosa
38.6039	4.8	3.4	1.6	0.2	Iris-setosa
33.7784	4.8	3.0	1.4	0.1	Iris-setosa
21.2990	4.3	3.0	1.1	0.1	Iris-setosa
42.2733	5.8	4.0	1.2	0.2	Iris-setosa
51.0352	5.7	4.4	1.5	0.4	Iris-setosa
39.6972	5.4	3.9	1.3	0.4	Iris-setosa
38.1327	5.1	3.5	1.4	0.3	Iris-setosa
57.8399	5.7	3.8	1.7	0.3	Iris-setosa
40.8564	5.1	3.8	1.5	0.3	Iris-setosa
51.9117	5.4	3.4	1.7	0.2	Iris-setosa
40.8564	5.1	3.7	1.5	0.4	Iris-setosa
22.1587	4.6	3.6	1.0	0.2	Iris-setosa
46.3039	5.1	3.3	1.7	0.5	Iris-setosa
45.8421	4.8	3.4	1.9	0.2	Iris-setosa
41.8879	5.0	3.0	1.6	0.2	Iris-setosa
41.8879	5.0	3.4	1.6	0.4	Iris-setosa
42.4743	5.2	3.5	1.5	0.2	Iris-setosa
39.6427	5.2	3.4	1.4	0.2	Iris-setosa
37.0122	4.7	3.2	1.6	0.2	Iris-setosa
38.6039	4.8	3.1	1.6	0.2	Iris-setosa
45.8044	5.4	3.4	1.5	0.4	Iris-setosa
42.4743	5.2	4.1	1.5	0.1	Iris-setosa
44.3488	5.5	4.2	1.4	0.2	Iris-setosa
37.7148	4.9	3.1	1.5	0.1	Iris-setosa
31.4159	5.0	3.2	1.2	0.2	Iris-setosa
41.1810	5.5	3.5	1.3	0.2	Iris-setosa
37.7148	4.9	3.1	1.5	0.1	Iris-setosa
26.3559	4.4	3.0	1.3	0.2	Iris-setosa

40.8564	5.1	3.4	1.5	0.2	Iris-setosa
34.0339	5.0	3.5	1.3	0.3	Iris-setosa
27.5675	4.5	2.3	1.3	0.3	Iris-setosa
26.3559	4.4	3.2	1.3	0.2	Iris-setosa
41.8879	5.0	3.5	1.6	0.6	Iris-setosa
51.7515	5.1	3.8	1.9	0.4	Iris-setosa
33.7784	4.8	3.0	1.4	0.3	Iris-setosa
43.5802	5.1	3.8	1.6	0.2	Iris-setosa
31.0222	4.6	3.2	1.4	0.2	Iris-setosa
44.1237	5.3	3.7	1.5	0.2	Iris-setosa
36.6519	5.0	3.3	1.4	0.2	Iris-setosa
241.1696	7.0	3.2	4.7	1.4	Iris-versicolor
193.0195	6.4	3.2	4.5	1.5	Iris-versicolor
244.2997	6.9	3.1	4.9	1.5	Iris-versicolor
126.7109	5.5	2.3	4.0	1.3	Iris-versicolor
203.5228	6.5	2.8	4.6	1.5	Iris-versicolor
153.1055	5.7	2.8	4.5	1.3	Iris-versicolor
195.3474	6.3	3.3	4.7	1.6	Iris-versicolor
82.9726	4.9	2.4	3.3	1.0	Iris-versicolor
209.8333	6.6	2.9	4.6	1.3	Iris-versicolor
110.4333	5.2	2.7	3.9	1.4	Iris-versicolor
91.6298	5.0	2.0	3.5	1.0	Iris-versicolor
153.1024	5.9	3.0	4.2	1.5	Iris-versicolor
150.7964	6.0	2.2	4.0	1.0	Iris-versicolor
183.1412	6.1	2.9	4.7	1.4	Iris-versicolor
118.2244	5.6	2.9	3.6	1.3	Iris-versicolor
206.8383	6.7	3.1	4.4	1.4	Iris-versicolor
147.7805	5.6	3.0	4.5	1.5	Iris-versicolor
144.4337	5.8	2.7	4.1	1.0	Iris-versicolor
181.1442	6.2	2.2	4.5	1.5	Iris-versicolor
128.0764	5.6	2.5	3.9	1.1	Iris-versicolor
174.9741	5.9	3.2	4.8	1.8	Iris-versicolor
155.8649	6.1	2.8	4.0	1.3	Iris-versicolor
203.6600	6.3	2.5	4.9	1.5	Iris-versicolor
183.1412	6.1	2.8	4.7	1.2	Iris-versicolor
184.4408	6.4	2.9	4.3	1.3	Iris-versicolor
200.7101	6.6	3.0	4.4	1.4	Iris-versicolor
232.4276	6.8	2.8	4.8	1.4	Iris-versicolor
235.0435	6.7	3.0	5.0	1.7	Iris-versicolor
169.6460	6.0	2.9	4.5	1.5	Iris-versicolor
119.0821	5.7	2.6	3.5	1.0	Iris-versicolor

120.3754	5.5	2.4	3.8	1.1	Iris-versicolor
117.2076	5.5	2.4	3.7	1.0	Iris-versicolor
137.3881	5.8	2.7	3.9	1.2	Iris-versicolor
192.2655	6.0	2.7	5.1	1.6	Iris-versicolor
137.4133	5.4	3.0	4.5	1.5	Iris-versicolor
169.6460	6.0	3.4	4.5	1.6	Iris-versicolor
220.9409	6.7	3.1	4.7	1.5	Iris-versicolor
182.8784	6.3	2.3	4.4	1.3	Iris-versicolor
134.6445	5.6	3.0	4.1	1.3	Iris-versicolor
126.7109	5.5	2.5	4.0	1.3	Iris-versicolor
139.3820	5.5	2.6	4.4	1.2	Iris-versicolor
179.2446	6.1	3.0	4.6	1.4	Iris-versicolor
140.9109	5.8	2.6	4.0	1.2	Iris-versicolor
86.3938	5.0	2.3	3.3	1.0	Iris-versicolor
137.9285	5.6	2.7	4.2	1.3	Iris-versicolor
142.8985	5.7	3.0	4.2	1.2	Iris-versicolor
142.8985	5.7	2.9	4.2	1.3	Iris-versicolor
173.0934	6.2	2.9	4.3	1.3	Iris-versicolor
81.7128	5.1	2.5	3.0	1.1	Iris-versicolor
139.4961	5.7	2.8	4.1	1.3	Iris-versicolor
249.3796	6.3	3.3	6.0	2.5	Iris-virginica
179.6614	5.8	2.7	5.1	1.9	Iris-virginica
311.4564	7.1	3.0	5.9	2.1	Iris-virginica
232.7543	6.3	2.9	5.6	1.8	Iris-virginica
256.6158	6.5	3.0	5.8	2.2	Iris-virginica
399.2085	7.6	3.0	6.6	2.1	Iris-virginica
113.1445	4.9	2.5	4.5	1.7	Iris-virginica
351.5725	7.3	2.9	6.3	1.8	Iris-virginica
272.6504	6.7	2.5	5.8	1.8	Iris-virginica
331.1490	7.2	3.6	6.1	2.5	Iris-virginica
225.6449	6.5	3.2	5.1	2.0	Iris-virginica
227.3340	6.4	2.7	5.3	1.9	Iris-virginica
266.3233	6.8	3.0	5.5	2.1	Iris-virginica
170.1172	5.7	2.5	5.0	2.0	Iris-virginica
179.6614	5.8	2.8	5.1	2.4	Iris-virginica
227.3340	6.4	3.2	5.3	2.3	Iris-virginica
243.3425	6.5	3.0	5.5	1.8	Iris-virginica
415.9919	7.7	3.8	6.7	2.2	Iris-virginica
428.4096	7.7	2.6	6.9	2.3	Iris-virginica
188.4956	6.0	2.2	5.0	1.5	Iris-virginica
284.1853	6.9	3.2	5.7	2.3	Iris-virginica

160.9166	5.6	2.8	4.9	2.0	Iris-virginica
415.9919	7.7	2.8	6.7	2.0	Iris-virginica
203.6600	6.3	2.7	4.9	1.8	Iris-virginica
267.9496	6.7	3.3	5.7	2.1	Iris-virginica
325.7203	7.2	3.2	6.0	1.8	Iris-virginica
193.2205	6.2	2.8	4.8	1.8	Iris-virginica
190.9345	6.1	3.0	4.9	1.8	Iris-virginica
240.2020	6.4	2.8	5.6	2.1	Iris-virginica
314.8630	7.2	3.0	5.8	1.6	Iris-virginica
349.8017	7.4	2.8	6.1	1.9	Iris-virginica
418.2758	7.9	3.8	6.4	2.0	Iris-virginica
240.2020	6.4	2.8	5.6	2.2	Iris-virginica
211.9727	6.3	2.8	5.1	1.5	Iris-virginica
218.2108	6.1	2.6	5.6	1.4	Iris-virginica
378.7389	7.7	3.0	6.1	2.3	Iris-virginica
232.7543	6.3	3.4	5.6	2.4	Iris-virginica
235.9127	6.4	3.1	5.5	1.8	Iris-virginica
180.9557	6.0	3.0	4.8	1.8	Iris-virginica
269.2282	6.9	3.1	5.4	2.1	Iris-virginica
263.2487	6.7	3.1	5.6	2.4	Iris-virginica
254.2711	6.9	3.1	5.1	2.3	Iris-virginica
179.6614	5.8	2.7	5.1	1.9	Iris-virginica
285.6922	6.8	3.2	5.9	2.3	Iris-virginica
267.9496	6.7	3.3	5.7	2.5	Iris-virginica
244.4452	6.7	3.0	5.2	2.3	Iris-virginica
207.8164	6.3	2.5	5.0	1.9	Iris-virginica
230.0693	6.5	3.0	5.2	2.0	Iris-virginica
217.3731	6.2	3.4	5.4	2.3	Iris-virginica
185.9100	5.9	3.0	5.1	1.8	Iris-virginica