Module 1 Activity

January 26, 2025

#

Module 1 Activity

#

Assigned at the start of Module 1

#

Due at the end of Module 1

1 Weekly Discussion Forum Participation

Each week, you are required to participate in the module's discussion forum. The discussion forum consists of the week's Module Activity, which is released at the beginning of the module. You must complete/attempt the activity before you can post about the activity and anything that relates to the topic.

1.1 Grading of the Discussion

1.1.1 1. Initial Post:

Create your thread by **Day 5 (Saturday night at midnight, PST).**

1.1.2 2. Responses:

Respond to at least two other posts by **Day 7 (Monday night at midnight, PST).**

1.2 Grading Criteria:

Your participation will be graded as follows:

1.2.1 Full Credit (100 points):

- Submit your initial post by **Day 5.**
- Respond to at least two other posts by **Day 7.**

1.2.2 Half Credit (50 points):

- If your initial post is late but you respond to two other posts.
- If your initial post is on time but you fail to respond to at least two other posts.

1.2.3 No Credit (0 points):

- If both your initial post and responses are late.
- If you fail to submit an initial post and do not respond to any others.

1.3 Additional Notes:

- **Late Initial Posts:** Late posts will automatically receive half credit if two responses are completed on time.
- **Substance Matters:** Responses must be thoughtful and constructive. Comments like "Great post!" or "I agree!" without further explanation will not earn credit.
- **Balance Participation:** Aim to engage with threads that have fewer or no responses to ensure a balanced discussion.

1.4 Avoid:

- A number of posts within a very short time-frame, especially immediately prior to the posting deadline.
- Posts that complement another post, and then consist of a summary of that.

1.5 General Overview of Data

[816]:
```
import pandas as pd
from sklearn.datasets import load_iris
from tabulate import tabulate

iris = load_iris()

iris_df = pd.DataFrame(
    data=iris.data,
    columns=iris.feature_names
)
iris_df['target'] = iris.target

iris_df['target_name'] = [iris.target_names[x] for x in iris_df['target']]

top_5_of_each_species = iris_df.groupby('target_name').head(3).
  ↪reset_index(drop=True)

print(tabulate(top_5_of_each_species, headers='keys', tablefmt='grid'))
```

```
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|    |   sepal length (cm) |   sepal width (cm) |   petal length (cm) |   petal
width (cm) |   target | target_name   |
+====+=====================+====================+=====================+=========
===========+==========+===============+
|  0 |                 5.1 |                3.5 |                 1.4 |
0.2 |        0 | setosa        |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  1 |                 4.9 |                3   |                 1.4 |
0.2 |        0 | setosa        |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  2 |                 4.7 |                3.2 |                 1.3 |
0.2 |        0 | setosa        |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  3 |                 7   |                3.2 |                 4.7 |
1.4 |        1 | versicolor    |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  4 |                 6.4 |                3.2 |                 4.5 |
1.5 |        1 | versicolor    |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  5 |                 6.9 |                3.1 |                 4.9 |
1.5 |        1 | versicolor    |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  6 |                 6.3 |                3.3 |                 6   |
2.5 |        2 | virginica     |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  7 |                 5.8 |                2.7 |                 5.1 |
1.9 |        2 | virginica     |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
|  8 |                 7.1 |                3   |                 5.9 |
2.1 |        2 | virginica     |
+----+---------------------+--------------------+---------------------+---------
-----------+----------+---------------+
```

2 1. Module 1

Python has four inbuilt data structures as follows: 1. Lists 2. Dictionary 3. Tuple 4. Set

For each of these data structures use the Iris data set and provide the following:

3 (a) Lists

- Define a “list” as comprehensively and accurately as you would find in a dictionary. Provide a clear and concise explanation, covering its essential meaning, characteristics, and any relevant context.
- Create a list in Python using the first observation, all features and label from the Iris data set
- Show the output with a print statement
- Add the second observation of the Iris data set to your list
- Show the output of your list with a print statement

3.1 A list is a built-in data structure that can store multiple items in a single variable. They are ordered and each element can be accessed by its index. Lists allow different types of data elements and duplicates. They are mutable, meaning their elements can be modified, replaced, or removed, and new elements can be added at any position. Their size grows or shrinks automatically, making them dynamic.

[817]:
```python
## import libraries
import pandas as pd
## import iris dataset
from sklearn.datasets import load_iris
```

[818]:
```python
iris = load_iris()

# create a list using the first observation
list_of_observations = [[[feature_value for feature_value in iris.data[0]],␣
  ↪iris.target[0]]]
print("Initial list with first observation: ")
for index, feature_value in enumerate(list_of_observations[0][0]):
    # print the feature name and the value of the feature
    print(f"  • {iris.feature_names[index]}: {feature_value}")
# print the target name (the species of the iris)
print(f"  • {iris.target_names[list_of_observations[0][1]]}")

# add the second observation to the list
list_of_observations.append([list(iris.data[1]), iris.target[1]])

print("\nUpdated list after adding second observation: ")
for observation_index, current_observation in enumerate(list_of_observations):
    """ print the updated list, which includes reprinting the first and now
    printing the second observation """

    print(f"\nObservation {observation_index + 1}:")
    for feature_index, feature_value in enumerate(current_observation[0]):
        print(f"  • {iris.feature_names[feature_index]}: {feature_value}")
    print(f"  • {iris.target_names[current_observation[1]]}")
print("\n")

""" print out the pure list of observations, without adding any additional␣
↪information
so you can see the structure of the list """
print("List of observations, without organizing info:")
print(list_of_observations)
```

```
Initial list with first observation:
  • sepal length (cm): 5.1
  • sepal width (cm): 3.5
  • petal length (cm): 1.4
  • petal width (cm): 0.2
  • setosa

Updated list after adding second observation:

Observation 1:
  • sepal length (cm): 5.1
  • sepal width (cm): 3.5
  • petal length (cm): 1.4
  • petal width (cm): 0.2
  • setosa

Observation 2:
  • sepal length (cm): 4.9
  • sepal width (cm): 3.0
  • petal length (cm): 1.4
  • petal width (cm): 0.2
  • setosa


List of observations, without organizing info:
[[[np.float64(5.1), np.float64(3.5), np.float64(1.4), np.float64(0.2)],
np.int64(0)], [[np.float64(4.9), np.float64(3.0), np.float64(1.4),
np.float64(0.2)], np.int64(0)]]
```

4 (b) Dictionary

- Define a "dictionary" as comprehensively and accurately as you would find in a dictionary. Provide a clear and concise explanation, covering its essential meaning, characteristics, and any relevant context.
- Create a dictionary in Python using the classes of Setosa, Versicolor and Virginica, allow elements (2 observations per class) to be added
- Show the output with a print statement
- Add a third observation of the Iris data set for each class
- Show the output of your with a print statement

4.1 A dictionary stores information as key:value pairs. Their order is based on how they were inserted. Keys are unique identifiers, meaning that they cannot be duplicated or mutated. Values are the data associated with a key, and they can be modified or of any type.

4.2 Creating a dictionary with 2 observations per target

[819]:
```
# create a dictionary, adding 2 observations per unique target, focusing on␣
↪setosa, versicolor, and virginica
dictionary_of_observations = {}
# iterate through each unique target
for iris_target in set(iris.target):
    # create a dictionary key for the target
    dictionary_of_observations[iris_target] = []

    if (iris_target > 0):
        # print a blank line between each target
        print(" ")

    print(f"\nObservations for target {iris.target_names[iris_target]}:")

    # create a count to only add 2 observations per target
    count = 0

    for observation_index, current_target in enumerate(iris.target):
        # only add the observation if the target matches the current target
        if current_target == iris_target:
            # add the observation to the dictionary
            dictionary_of_observations[iris_target].append([list(iris.
↪data[observation_index]), current_target])
            count += 1

            print(f"\nObservation {count}:")

            for feature_index, current_feature_value in␣
↪enumerate(dictionary_of_observations[iris_target][-1][0]):
                # don't add a comma after the last feature value
                if feature_index ==␣
↪len(dictionary_of_observations[iris_target][-1][0]) - 1:
                    print(f"{iris.feature_names[feature_index]}:␣
↪{current_feature_value}")
                else:
                    print(f"{iris.feature_names[feature_index]}:␣
↪{current_feature_value}", end=", ")

        if count == 2:
            break
```

```
Observations for target setosa:

Observation 1:
sepal length (cm): 5.1, sepal width (cm): 3.5, petal length (cm): 1.4, petal
width (cm): 0.2

Observation 2:
sepal length (cm): 4.9, sepal width (cm): 3.0, petal length (cm): 1.4, petal
width (cm): 0.2


Observations for target versicolor:

Observation 1:
sepal length (cm): 7.0, sepal width (cm): 3.2, petal length (cm): 4.7, petal
width (cm): 1.4

Observation 2:
sepal length (cm): 6.4, sepal width (cm): 3.2, petal length (cm): 4.5, petal
width (cm): 1.5


Observations for target virginica:

Observation 1:
sepal length (cm): 6.3, sepal width (cm): 3.3, petal length (cm): 6.0, petal
width (cm): 2.5

Observation 2:
sepal length (cm): 5.8, sepal width (cm): 2.7, petal length (cm): 5.1, petal
width (cm): 1.9
```

4.3 Dictionary of observations, without special formatting

[820]:
```
""" print out the pure dictionary of observations, without adding any␣
 ↪additional information
so you can see the structure of the dictionary """
print("\nDictionary of observations, without organizing info:")
print(dictionary_of_observations)
```

```
Dictionary of observations, without organizing info:
{np.int64(0): [[[np.float64(5.1), np.float64(3.5), np.float64(1.4),
np.float64(0.2)], np.int64(0)], [[np.float64(4.9), np.float64(3.0),
np.float64(1.4), np.float64(0.2)], np.int64(0)]], np.int64(1):
[[[np.float64(7.0), np.float64(3.2), np.float64(4.7), np.float64(1.4)],
np.int64(1)], [[np.float64(6.4), np.float64(3.2), np.float64(4.5),
np.float64(1.5)], np.int64(1)]], np.int64(2): [[[np.float64(6.3),
np.float64(3.3), np.float64(6.0), np.float64(2.5)], np.int64(2)],
[[np.float64(5.8), np.float64(2.7), np.float64(5.1), np.float64(1.9)],
np.int64(2)]]}
```

4.4 Adding a 3rd observation to the dictionary for each target

[821]:
```
# add a third observation to the dictionary for each target
for iris_target in dictionary_of_observations:
    # add the third observation to the dictionary that corresponds to the target
    for observation_index, current_target in enumerate(iris.target):
        # skip the observation if it's already in the dictionary
        if current_target == iris_target and [list(iris.
 ↪data[observation_index]),
                                              current_target] not in␣
 ↪dictionary_of_observations[iris_target]:
            dictionary_of_observations[iris_target].append([list(iris.
 ↪data[observation_index]), current_target])
            break

print("\nUpdated dictionary after adding third observation:")
for iris_target in dictionary_of_observations:

    print(f"\nObservations for target {iris.target_names[iris_target]}:")
    for observation_index, observation in␣
 ↪enumerate(dictionary_of_observations[iris_target]):
        print(f"Observation {observation_index + 1}:")
        for feature_index, current_target in enumerate(observation[0]):
            if feature_index ==␣
 ↪len(dictionary_of_observations[iris_target][-1][0]) - 1:
                print(f"{iris.feature_names[feature_index]}: {current_target}")
            else:
                print(f"{iris.feature_names[feature_index]}: {current_target}",␣
 ↪end=", ")
```

```
Updated dictionary after adding third observation:

Observations for target setosa:
Observation 1:
sepal length (cm): 5.1, sepal width (cm): 3.5, petal length (cm): 1.4, petal
width (cm): 0.2
Observation 2:
sepal length (cm): 4.9, sepal width (cm): 3.0, petal length (cm): 1.4, petal
width (cm): 0.2
Observation 3:
sepal length (cm): 4.7, sepal width (cm): 3.2, petal length (cm): 1.3, petal
width (cm): 0.2
```

```
Observations for target versicolor:
Observation 1:
sepal length (cm): 7.0, sepal width (cm): 3.2, petal length (cm): 4.7, petal
width (cm): 1.4
Observation 2:
sepal length (cm): 6.4, sepal width (cm): 3.2, petal length (cm): 4.5, petal
width (cm): 1.5
Observation 3:
sepal length (cm): 6.9, sepal width (cm): 3.1, petal length (cm): 4.9, petal
width (cm): 1.5

Observations for target virginica:
Observation 1:
sepal length (cm): 6.3, sepal width (cm): 3.3, petal length (cm): 6.0, petal
width (cm): 2.5
Observation 2:
sepal length (cm): 5.8, sepal width (cm): 2.7, petal length (cm): 5.1, petal
width (cm): 1.9
Observation 3:
sepal length (cm): 7.1, sepal width (cm): 3.0, petal length (cm): 5.9, petal
width (cm): 2.1
```

4.5 Printing out dictionary without formatting

[822]:
```python
""" print out the updated dictionary of observations, without adding any␣
↪additional information
so you can see the structure of the dictionary """
print("\nUpdated dictionary of observations, without organizing info:")
print(dictionary_of_observations)
```

```
Updated dictionary of observations, without organizing info:
{np.int64(0): [[[np.float64(5.1), np.float64(3.5), np.float64(1.4),
np.float64(0.2)], np.int64(0)], [[np.float64(4.9), np.float64(3.0),
np.float64(1.4), np.float64(0.2)], np.int64(0)], [[np.float64(4.7),
np.float64(3.2), np.float64(1.3), np.float64(0.2)], np.int64(0)]], np.int64(1):
[[[np.float64(7.0), np.float64(3.2), np.float64(4.7), np.float64(1.4)],
np.int64(1)], [[np.float64(6.4), np.float64(3.2), np.float64(4.5),
np.float64(1.5)], np.int64(1)], [[np.float64(6.9), np.float64(3.1),
np.float64(4.9), np.float64(1.5)], np.int64(1)]], np.int64(2):
[[[np.float64(6.3), np.float64(3.3), np.float64(6.0), np.float64(2.5)],
np.int64(2)], [[np.float64(5.8), np.float64(2.7), np.float64(5.1),
np.float64(1.9)], np.int64(2)], [[np.float64(7.1), np.float64(3.0),
np.float64(5.9), np.float64(2.1)], np.int64(2)]]}
```

5 (c) Tuple

- Define a “tuple” ” as comprehensively and accurately as you would find in a dictionary. Provide a clear and concise explanation, covering its essential meaning, characteristics, and any relevant context.
- Create a tuple in Python using the classes of Setosa, Versicolor and Virginica, allow elements (start with 2 observations and 2 features per class) to be added
- Show the output with a print statement
- Add a third observation and a third feature of the Iris data set for each class
- Show the output of your with a print statement

5.1 Tuples is an ordered dataset which can store multiple items in a variable. Each item is seperated by commas and can be of different data types. It cannot be changed, so once it has been created, elements cannot be added, removed, or replaced. Elements are accessed using indexing.

5.2 Creating a tuple with 2 obs per class and 2 features per class

[823]:
```python
import numpy as np

# create a tuple, adding 2 observations per unique target, and 2 features per␣
↪observation
tuple_of_observations = ()
# iterate through each unique target
for iris_target in set(iris.target):
    # create a list to store the observations
    observations = []
    print(f"\nObservations for target {iris.target_names[iris_target]}:")

    # create a count to only add 2 observations per target
    count = 0

    for index, current_target in enumerate(iris.target):
        # only add the observation if the target matches the current target
        if current_target == iris_target:
            # add the observation to the list, only including the first 2␣
↪features
            observations.append([iris.data[index][:2], current_target])
            count += 1

            # print the observation
            print(f"\nObservation {count}:")
            for feature_index, feature_value in enumerate(observations[-1][0]):
                print(f"  • {iris.feature_names[feature_index]}:␣
↪{feature_value}")

        if count == 2:
            break
    # add the list of observations to the tuple
    tuple_of_observations += (observations,)
```

```
Observations for target setosa:

Observation 1:
  • sepal length (cm): 5.1
  • sepal width (cm): 3.5

Observation 2:
  • sepal length (cm): 4.9
  • sepal width (cm): 3.0

Observations for target versicolor:

Observation 1:
  • sepal length (cm): 7.0
  • sepal width (cm): 3.2

Observation 2:
  • sepal length (cm): 6.4
  • sepal width (cm): 3.2

Observations for target virginica:

Observation 1:
  • sepal length (cm): 6.3
  • sepal width (cm): 3.3

Observation 2:
  • sepal length (cm): 5.8
  • sepal width (cm): 2.7
```

5.3 Printing pure tuple to see current data structure

[824]:
```
""" print out the pure tuple of observations, without adding any additional␣
 ↪information
so you can see the structure of the tuple """
print("\nTuple of observations, without organizing info:")
print(tuple_of_observations)
```

```
Tuple of observations, without organizing info:
([[array([5.1, 3.5]), np.int64(0)], [array([4.9, 3. ]), np.int64(0)]],
[[array([7. , 3.2]), np.int64(1)], [array([6.4, 3.2]), np.int64(1)]],
[[array([6.3, 3.3]), np.int64(2)], [array([5.8, 2.7]), np.int64(2)]])
```

5.4 Adding 3rd obs to the tuple for each target and a 3rd feature to each tuple

[825]:
```
# add a third observation to the tuple for each target. convert the tuple to a␣
↪list to add the observation
temp_observations_list = list(tuple_of_observations)

for iris_target in range(len(temp_observations_list)):
    # add the third observation to the list that corresponds to the target
    for observation_index in range(len(temp_observations_list[iris_target])):
        # skip the observation if it's already in the list
        if len(temp_observations_list[iris_target][observation_index][0]) == 2:
            # add the third feature to the observation
            third_feature = iris.data[np.where(iris.target ==␣
↪iris_target)][0][2]
            # convert the tuple to a list to add the third feature
            temp_observations_list[iris_target][observation_index][0] =␣
↪list(temp_observations_list
                                                                          ␣
↪[iris_target][observation_index][0])
            # add the third feature to the observation
            temp_observations_list[iris_target][observation_index][0].
↪append(third_feature)

    count = len(temp_observations_list[iris_target])

    for index, current_target in enumerate(iris.target):
        if count >= 3:
            break

        if current_target == iris_target:
            # add the observation to the list, only including the first 2␣
↪features
            if not any(np.allclose(iris.data[index][:2], obs[0][:2]) for obs in␣
↪temp_observations_list[iris_target]):
                temp_observations_list[iris_target].append([list(iris.
↪data[index][:3]), current_target])
                count += 1

# convert the list back to a tuple
tuple_of_observations = tuple(temp_observations_list)

print("Updated tuple after adding third observation:")
for iris_target in range(len(tuple_of_observations)):
    print(f"\nObservations for target {iris.target_names[iris_target]}:")
    for observation_index, observation in␣
↪enumerate(tuple_of_observations[iris_target]):
        print(f"\nObservation {observation_index + 1}:")
        for feature_index, feature_value in enumerate(observation[0]):
            #print(f"  • {iris.feature_names[feature_index]}: {feature_value}")
            if feature_index ==␣
↪len(dictionary_of_observations[iris_target][-1][0]) - 1:
                print(f"{iris.feature_names[feature_index]}: {feature_value}")
            else:
                print(f"{iris.feature_names[feature_index]}: {feature_value}",␣
↪end=", ")
```

```
Updated tuple after adding third observation:

Observations for target setosa:

Observation 1:
sepal length (cm): 5.1, sepal width (cm): 3.5, petal length (cm): 1.4,
Observation 2:
sepal length (cm): 4.9, sepal width (cm): 3.0, petal length (cm): 1.4,
Observation 3:
sepal length (cm): 4.7, sepal width (cm): 3.2, petal length (cm): 1.3,
Observations for target versicolor:

Observation 1:
sepal length (cm): 7.0, sepal width (cm): 3.2, petal length (cm): 4.7,
Observation 2:
sepal length (cm): 6.4, sepal width (cm): 3.2, petal length (cm): 4.7,
Observation 3:
sepal length (cm): 6.9, sepal width (cm): 3.1, petal length (cm): 4.9,
Observations for target virginica:

Observation 1:
sepal length (cm): 6.3, sepal width (cm): 3.3, petal length (cm): 6.0,
Observation 2:
sepal length (cm): 5.8, sepal width (cm): 2.7, petal length (cm): 6.0,
Observation 3:
sepal length (cm): 7.1, sepal width (cm): 3.0, petal length (cm): 5.9,
```

5.5 Printing out tuple without formatting

[826]:
```
""" print out the updated tuple of observations, without adding any additional␣
↪information
so you can see the structure of the tuple """
print("\nUpdated tuple of observations, without organizing info:")
print(tuple_of_observations)
```

```
Updated tuple of observations, without organizing info:
([[[np.float64(5.1), np.float64(3.5), np.float64(1.4)], np.int64(0)],
[[np.float64(4.9), np.float64(3.0), np.float64(1.4)], np.int64(0)],
[[np.float64(4.7), np.float64(3.2), np.float64(1.3)], np.int64(0)]],
[[[np.float64(7.0), np.float64(3.2), np.float64(4.7)], np.int64(1)],
[[np.float64(6.4), np.float64(3.2), np.float64(4.7)], np.int64(1)],
[[np.float64(6.9), np.float64(3.1), np.float64(4.9)], np.int64(1)]],
[[[np.float64(6.3), np.float64(3.3), np.float64(6.0)], np.int64(2)],
[[np.float64(5.8), np.float64(2.7), np.float64(6.0)], np.int64(2)],
[[np.float64(7.1), np.float64(3.0), np.float64(5.9)], np.int64(2)]]])
```

6 (d) Set

- Define a "set" " as comprehensively and accurately as you would find in a dictionary. Provide a clear and concise explanation, covering its essential meaning, characteristics, and any relevant context.
- Create a set (1st set) in Python using the first observation of the Setosa class allowing elements to be added
- Show the output with a print statement
- Create another set (2nd set) by adding the second observation of the Setosa class
- Show the output of the following operations with the 1st set and 2nd set with a print statement
  - Difference
  - Intersection
  - Union

6.1 Sets is an unordered collection which can store multiple items in a single variable. It cannot be modified and is unindexed, It does not contain duplicate elements.

6.2 Create 2 sets

[827]:
```python
set_of_observations = {tuple(iris.data[0])}
print("First set with first observation:")
for index, value in enumerate(iris.data[0]):
    print(f"  • {iris.feature_names[index]}: {value}")

# create a second set with the second observation
second_set_of_observations = {tuple(iris.data[1])}

print("\nSecond set with second observation:")
for index, value in enumerate(iris.data[1]):
    print(f"  • {iris.feature_names[index]}: {value}")
```

```
First set with first observation:
  • sepal length (cm): 5.1
  • sepal width (cm): 3.5
  • petal length (cm): 1.4
  • petal width (cm): 0.2

Second set with second observation:
  • sepal length (cm): 4.9
  • sepal width (cm): 3.0
  • petal length (cm): 1.4
  • petal width (cm): 0.2
```

6.3 Printing results of set operations

[828]:
```python
# print difference between the two sets
print("\nDifference between the two sets:")
difference = set_of_observations.difference(second_set_of_observations)
if len(difference) == 0:
    print("  • No difference")
else:
    for feature_index, value in enumerate(difference.pop()):
        print(f"  • {iris.feature_names[feature_index]}: {value}")

# print intersection of the two sets
print("\nIntersection of the two sets:")
intersection = set_of_observations.intersection(second_set_of_observations)
if len(intersection) == 0:
    print("  • No intersection")
else:
    for feature_index, value in enumerate(observation):
        print(f"  • {iris.feature_names[feature_index]}: {value}")

# print union of the two sets
print("\nUnion of the two sets:")
union = set_of_observations.union(second_set_of_observations)
for observation in union:
    for feature_index, value in enumerate(observation):
        print(f"  • {iris.feature_names[feature_index]}: {value}")
```

```
Difference between the two sets:
  • sepal length (cm): 5.1
  • sepal width (cm): 3.5
  • petal length (cm): 1.4
  • petal width (cm): 0.2

Intersection of the two sets:
  • No intersection

Union of the two sets:
  • sepal length (cm): 5.1
  • sepal width (cm): 3.5
  • petal length (cm): 1.4
  • petal width (cm): 0.2
  • sepal length (cm): 4.9
  • sepal width (cm): 3.0
```

- `petal length (cm): 1.4`
- `petal width (cm): 0.2`

7 References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. MIT Press and McGraw-Hill, 2009. ISBN-13: 978-0-262-03384-8 [2] Thomas H. Cormen. The clrscode and clrscode3e packages for LaTeX2e, Retrived Jan 2010, http://www.cs.dartmouth.edu/ thc/clrscode/ [3] simpleilearn, Top 90+ Data Science Interview Questions and Answers: Basic to Technical, https://www.simplilearn.com/tutorials/data-science-tutorial/data-science-interview-questions [4] simpleilearn, 22 Artificial Intelligence Interview Questions to Prepare, https://www.simplilearn.com/artificial-intelligence-ai-interview-questions-and-answers-article
[5] Python, Data Structures, https://docs.python.org/3/tutorial/datastructures.html