

```
'''
```

### 1.2.1

Напишите функцию `sum_range(start, end)`, которая суммирует Все целые числа от значения «start» до Величины «end» Включительно. Если пользователь задаст первое число большее чем Второе, просто поменяйте их местами.

```
'''
```

```
def sum_range(start, end):  
    if start > end:  
        end, start = start, end  
    return sum(range (start, end + 1))
```

```
print(sum_range(2, 12))  
print(sum_range(-4, 4))  
print(sum_range(3, 2))
```

**Program output:**

77

0

5

```
'''
```

### 1.2.2

Напишите рекурсивную функцию Вычисления факториала на языке Python

```
'''
```

```
def fact(num):  
    if num == 0:  
        return 1  
    else:  
        return num * fact(num-1)  
print(fact(5))
```

Program output:

120

```
'''
```

1.2.3

Напишите функции в Python, которая вычисляет Евклидово расстояние между двумя массивами NumPy.

```
'''
```

```
import numpy as np
def euclidian_distance(v1, v2):
    return sum((x-y)**2 for x,y in zip(v1, v2))**0.5
x = np.array([0,0,0])
y = np.array([3,3,3])
print(euclidian_distance(x,y))
```

**Program output:**

5.196152422706632

```
'''
```

#### 1.2.4

Напишите 4 функции в Python, которые рассчитывают квадрат Евклидова расстояния, Взвешенное евклидово расстояние, Хеммингово расстояние и расстояние Чебышева между двумя массивами NumPy.

```
'''
```

```
import numpy as np
def sqr_euclidean_distance(v1, v2):
    return sum((x - y) ** 2 for x, y in zip(v1, v2))
def weighted_euclidean_distance(v1, v2, w):
    return sum((x - y) ** 2 * s for x, y, s in zip(v1, v2, w)) ** 0.5
def manhattan_distance(v1, v2):
    return sum(abs(x - y) for (x, y) in zip(v1, v2))
def chebyshev_distance(v1, v2):
    return max(abs(x - y) for (x, y) in zip(v1, v2))

x = np.array([0,0,0])
y = np.array([3,3,3])
w = np.array([0,0,1])
print(sqr_euclidean_distance(x,y))
print(weighted_euclidean_distance(x,y,w))
print(manhattan_distance(x,y))
print(chebyshev_distance(x,y))
```

**Program output:**

27

3.0

9

3

```
'''
```

### 1.2.5

В Python есть Встроенные функции для Вычисления расстояний между Векторами. Мы будем использовать NumPy для расчета расстояния для двух точек, поскольку ранее рассмотренные структуры данных могут быть переведены в NumPy массив с помощью специальных функций. Например, для серий это будет выглядеть следующим образом: `seriesName.to_numpy()`.

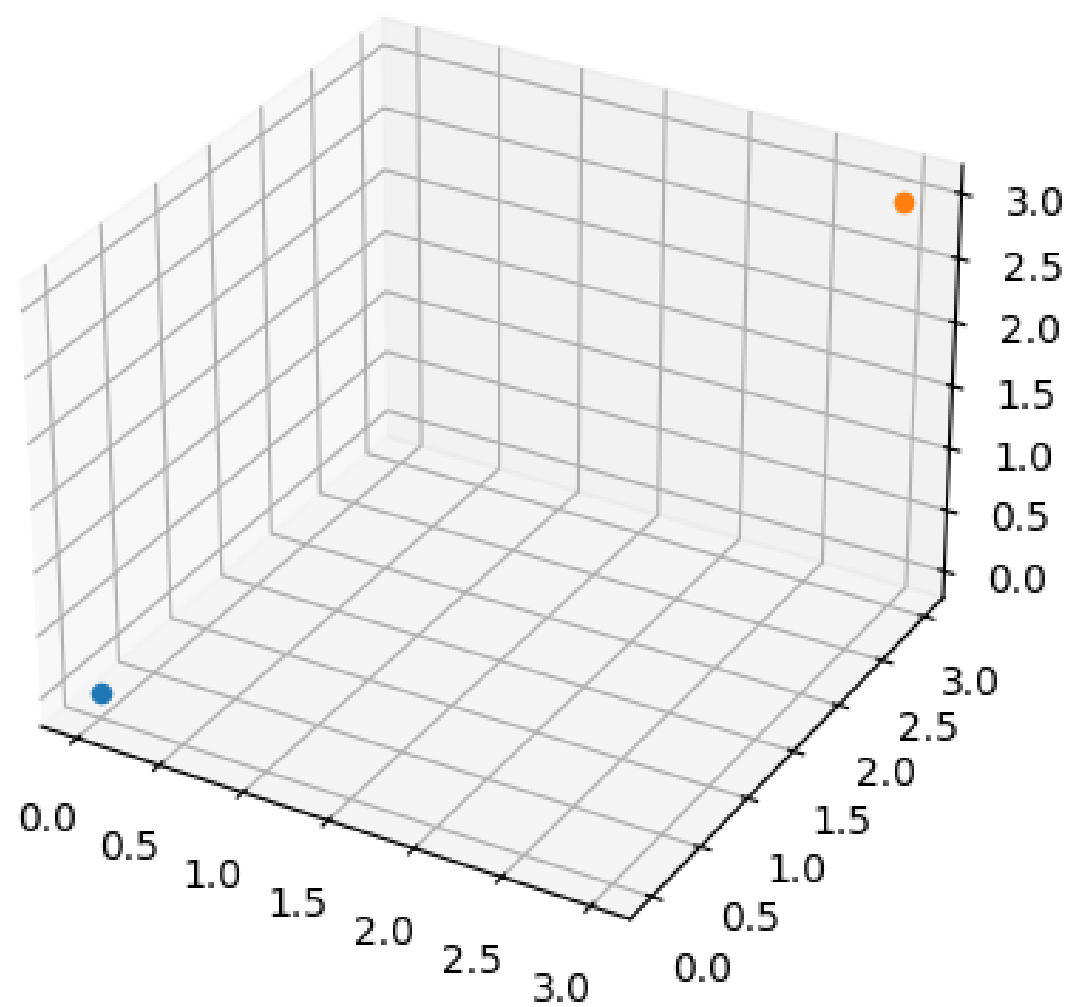
Для удобства Визуализации и анализа результатов в дальнейших расчетах будем использовать 2 точки в 3-х мерном пространстве

```
'''
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
```

```
ax.scatter(0, 0, 0)
ax.scatter(3, 3, 3)
plt.savefig("1.2.5.1.png")
```

Program output:



```
'''
```

### 1.2.6

Рассчитать расстояния между двумя точками с использованием методов определения расстояний, представленных выше.

```
'''
```

```
import numpy as np
x = np.array([0,0,0])
y = np.array([3,3,3])
w = np.array([0,0,1])
#Расстояние Евклида
print(np.linalg.norm(x-y))
#Квадрат Евклидова расстояния
print(np.linalg.norm(x-y) ** 2)
#Расстояние Чебышева
print(np.linalg.norm(x-y,ord=np.inf))
#Расстояние Хемминга
print(np.linalg.norm(x-y,ord=1))
```

#### Program output:

5.196152422706632

27.0

3.0

9.0

```
'''
```

### 1.3.1

Задайте 4 точки в трехмерном пространстве, рассчитайте между ними расстояния по описанным в примере выше метрикам. Отобразите точки в трехмерном пространстве.

```
'''
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

a = np.random.randint(5, size=3)
b = np.random.randint(5, size=3)
c = np.random.randint(5, size=3)
d = np.random.randint(5, size=3)

print("Расстояние между 1 и 2 точкой:")
print(np.linalg.norm(a-b))
print(np.linalg.norm(a-b) ** 2)
print(np.linalg.norm(a-b, ord = np.inf))
print(np.linalg.norm(a-b, ord=1))

print("\nРасстояние между 1 и 3 точкой:")
print(np.linalg.norm(a-c))
print(np.linalg.norm(a-c) ** 2)
print(np.linalg.norm(a-c, ord = np.inf))
print(np.linalg.norm(a-c, ord=1))

print("\nРасстояние между 1 и 4 точкой:")
print(np.linalg.norm(a-d))
print(np.linalg.norm(a-d) ** 2)
print(np.linalg.norm(a-d, ord = np.inf))
print(np.linalg.norm(a-d, ord=1))

print("\nРасстояние между 2 и 3 точкой:")
```



```

print(np.linalg.norm(b-c))
print(np.linalg.norm(b-c) ** 2)
print(np.linalg.norm(b-c, ord = np.inf))
print(np.linalg.norm(b-c, ord=1))

print("\nРасстояние между 2 и 4 точкой:")
print(np.linalg.norm(b-d))
print(np.linalg.norm(b-d) ** 2)
print(np.linalg.norm(b-d, ord = np.inf))
print(np.linalg.norm(b-d, ord=1))

print("\nРасстояние между 3 и 4 точкой:")
print(np.linalg.norm(d-c))
print(np.linalg.norm(d-c) ** 2)
print(np.linalg.norm(d-c, ord = np.inf))
print(np.linalg.norm(d-c, ord=1))

ax.scatter(a[0],a[1],a[2], marker = "^")
ax.scatter(b[0],b[1],b[2], marker = "v")
ax.scatter(c[0],c[1],c[2], marker = "<")
ax.scatter(d[0],d[1],d[2], marker = ">")
plt.savefig("1.3.1.1.png")

```

#### Program output:

Расстояние между 1 и 2 точкой:

```

3.0
9.0
2.0
5.0

```

Расстояние между 1 и 3 точкой:

```

4.242640687119285
17.999999999999996
4.0
6.0

```

Расстояние между 1 и 4 точкой:

3.0

9.0

3.0

3.0

Расстояние между 2 и 3 точкой:

3.3166247903554

11.0

3.0

5.0

Расстояние между 2 и 4 точкой:

2.449489742783178

5.999999999999999

2.0

4.0

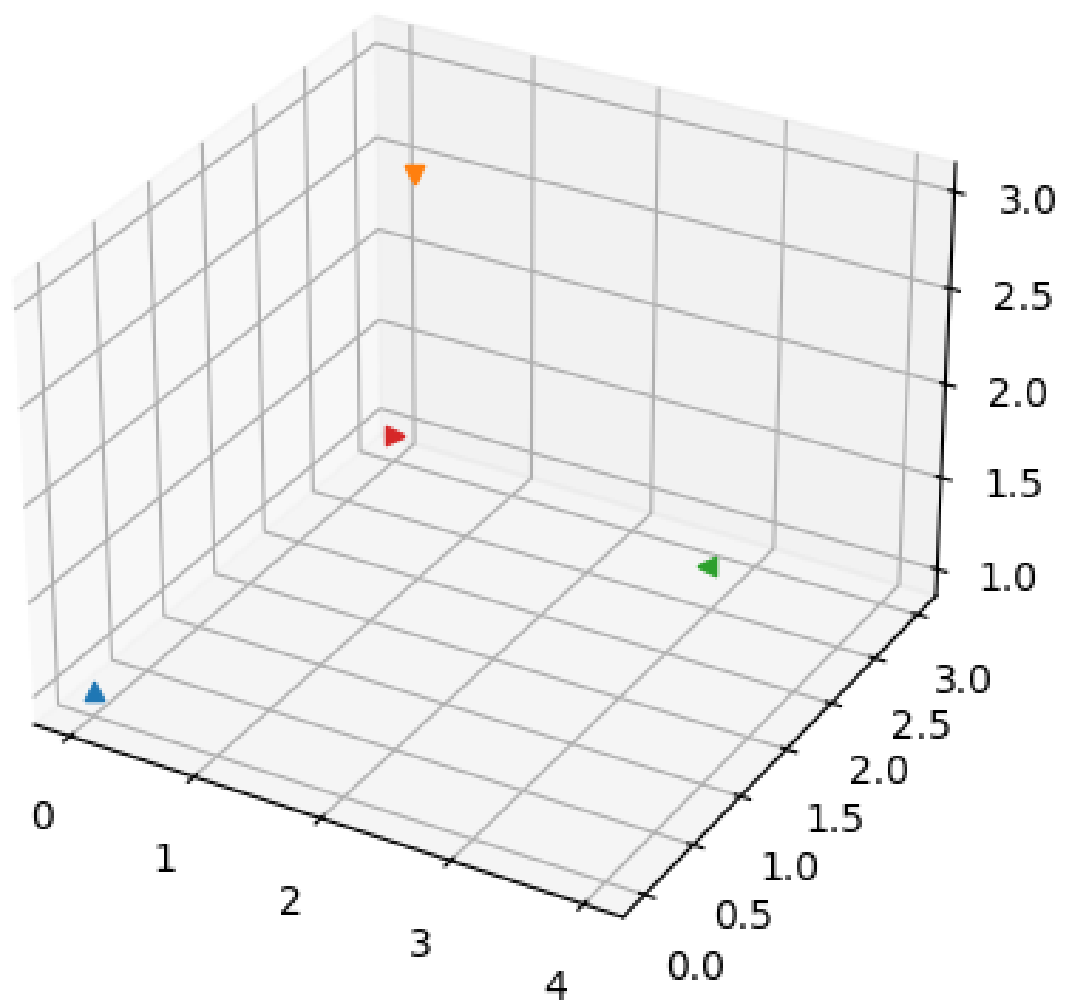
Расстояние между 3 и 4 точкой:

4.58257569495584

21.0

4.0

7.0



```
'''
```

### 1.3.2

Создать 5x5 матрицу со значениями в строках от 0 до 4. Для создания необходимо использовать функцию `arange`.

```
'''
```

```
import numpy as np
Z = np.zeros((5,5))
Z += np.arange(5)
print(Z)
```

**Program output:**

```
[[0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]]
```

```
'''
```

### 2.2.1

В примере показано создание 2d-массива со значениями x и y. Список target содержит возможные выходные классы (часто называемые метками). Далее происходит обучение классификатора k-ближайших соседей по исходным данным. Далее производится прогноз принадлежности к классам для двух точек данных.

```
'''
```

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
```

```
#данные
```

```
X = np.array([[ -1,-1], [-2,-1], [-3,-2], [ 1,1], [ 2,1], [ 3,2]])
target = [0, 0, 0, 1, 1, 1]
```

```
#обучаем модель k-ближайших соседей к данным
```

```
K = 3
```

```
model = KNeighborsClassifier(n_neighbors=K)
```

```
model.fit(X, target)
```

```
print(model)
```

```
#делаем прогноз
```

```
print('(-2,-2) is class')
```

```
print(model.predict([[-2,-2]]))
```

```
print('(1,3) is class')
```

```
print(model.predict([[1,3]]))
```

**Program output:**

```
KNeighborsClassifier(n_neighbors=3)
```

```
(-2,-2) is class
```

```
[0]
```

```
(1,3) is class
```

```
[1]
```

```
'''
```

### 2.2.2

Далее приведем более наглядный пример. Будет построена граница решения для каждого класса. В качестве данных будем использовать уже знакомый нам и Встроенный в библиотеку `sklearn` набор данных ирисов Фишера. Этот набор данных стал уже классическим, и часто используется в литературе для иллюстрации работы различных статистических алгоритмов. Датасет содержит наблюдения за 150 разными цветками ирисов, данные по каждому цветку расположены в строках. В столбцах записаны длина и ширина чашелистика, длина и ширина лепестка, вид ириса.

```
'''
```

```
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
iris = sns.load_dataset('iris')
print(iris)
```

Program output:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

[150 rows x 5 columns]

```
'''
```

### 2.2.3

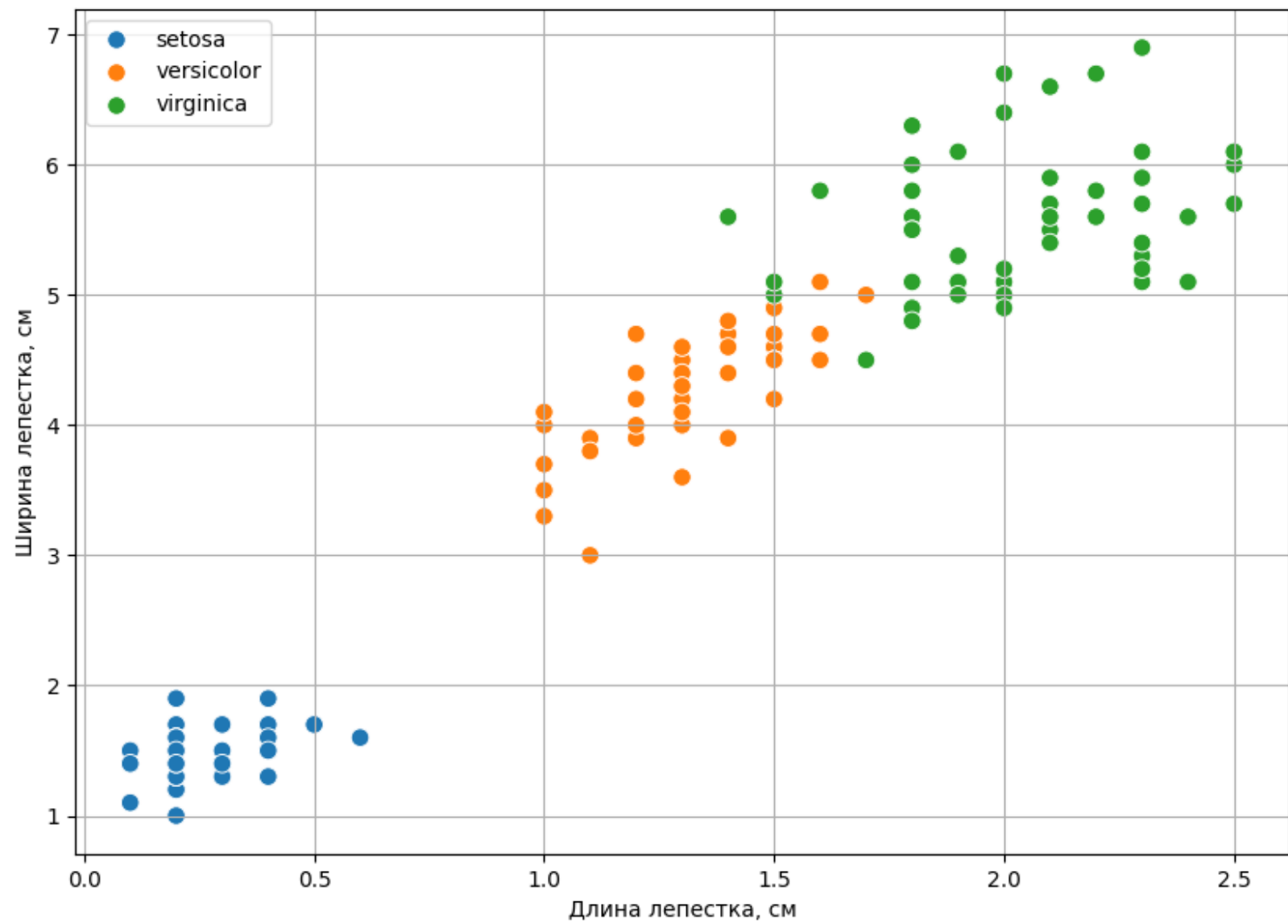
Покажем на графиках зависимости ширины лепестка от его длины, а также аналогичный график зависимость для длины и ширины чашелистика. Разные виды цветков отмечены разными цветами.

```
'''
```

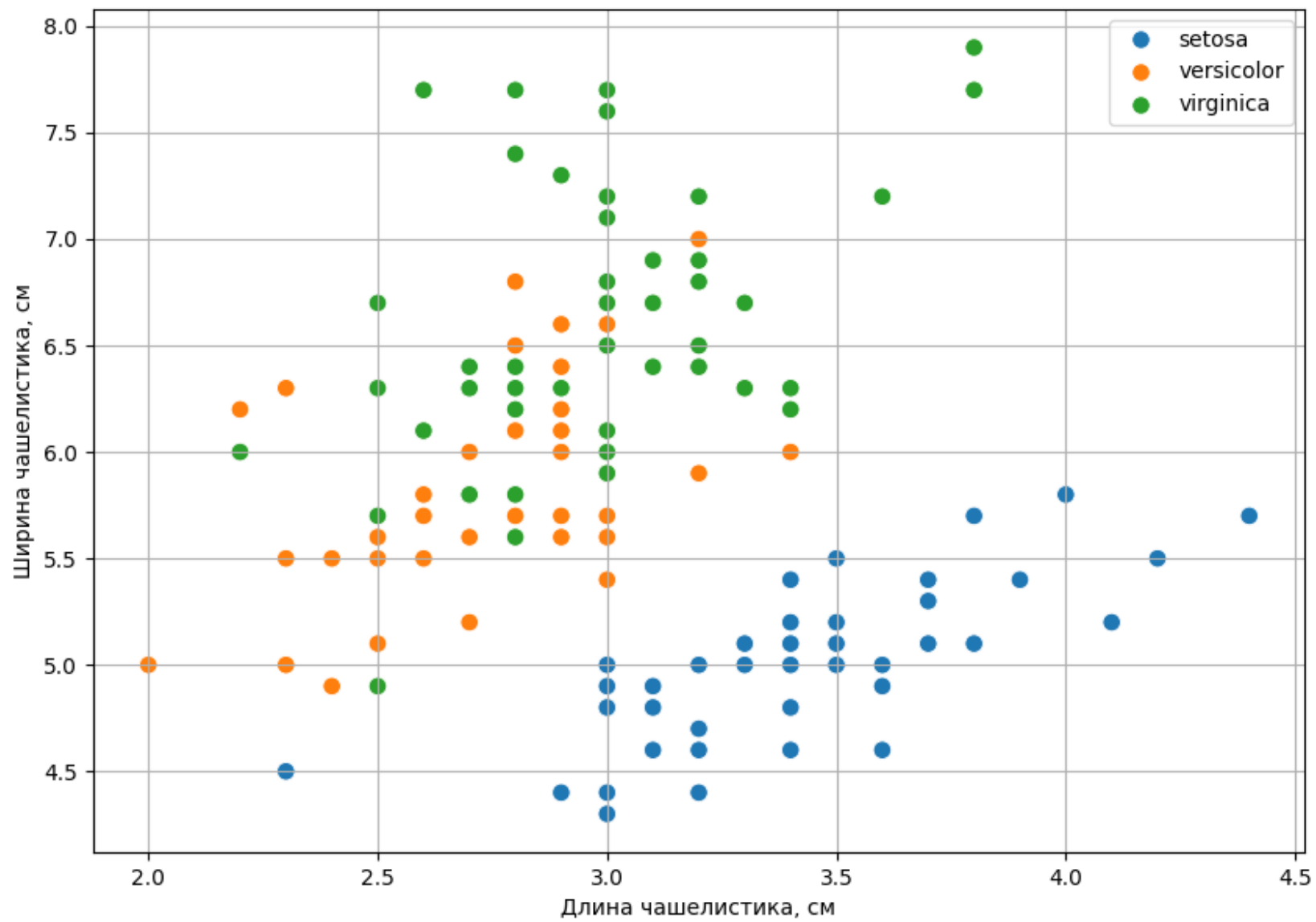
```
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
plt.figure(figsize=(10, 7))
iris = sns.load_dataset('iris')

sns.scatterplot(
    data=iris,
    x = 'petal_width', y = 'petal_length',
    hue='species',
    s=70)
plt.xlabel('Длина лепестка, см')
plt.ylabel('Ширина лепестка, см')
plt.legend()
plt.grid()
plt.savefig("2.2.3.1.png")
plt.clf()
plt.subplot()
sns.scatterplot(
    data=iris,
    x = 'sepal_width', y = 'sepal_length',
    hue='species',
    s=70)
plt.xlabel('Длина чашелистика, см')
plt.ylabel('Ширина чашелистика, см')
plt.legend()
plt.grid()
plt.savefig("2.2.3.2.png")
```

Program output:







```
...
```

#### 2.2.4

Из графиков видно, что в первом случае классы визуально хорошо отделены друг от друга, хотя два класса имеют небольшое пересечение. Во втором случае разделить два класса между собой уже намного труднее.

Далее разделим датасет на обучающую и тестовую выборки в соотношении 80:20. Обучающая выборка (training sample) – выборка, по которой производится настройка (оптимизация параметров) модели зависимости. Тестовая (или контрольная) выборка (test sample) – выборка, по которой оценивается качество построенной модели.

```
...
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
iris = sns.load_dataset('iris')
```

```
x_train, x_test, y_train, y_test = train_test_split(iris.iloc[:, :-1], iris.iloc[:, -1], test_size=0.20)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
print(x_train.head())
print(y_train.head())
```

```
model = KNeighborsClassifier(n_neighbors=3)
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_test)
print(y_pred)
```

```
plt.figure(figsize=(10,7))
sns.scatterplot(
    data=iris,
    x = 'petal_width', y = 'petal_length',
    hue='species',
    s=70)
```

```
plt.xlabel('Длина лепестка, см')
```

```

plt.ylabel('Ширина лепестка, см')
plt.legend(loc=2)
plt.grid()

for i in range(len(y_test)):
    if np.array(y_test)[i] != y_pred[i]:
        plt.scatter(x_test.iloc[i,3],x_test.iloc[i,2], color='red', s=150 )

plt.savefig("2.2.4.png")

from sklearn.metrics import accuracy_score
print(f'accuracy: {accuracy_score(y_test, y_pred):.3f}')

```

#### Program output:

	sepal_length	sepal_width	petal_length	petal_width
70	5.9	3.2	4.8	1.8
73	6.1	2.8	4.7	1.2
136	6.3	3.4	5.6	2.4
39	5.1	3.4	1.5	0.2
97	6.2	2.9	4.3	1.3

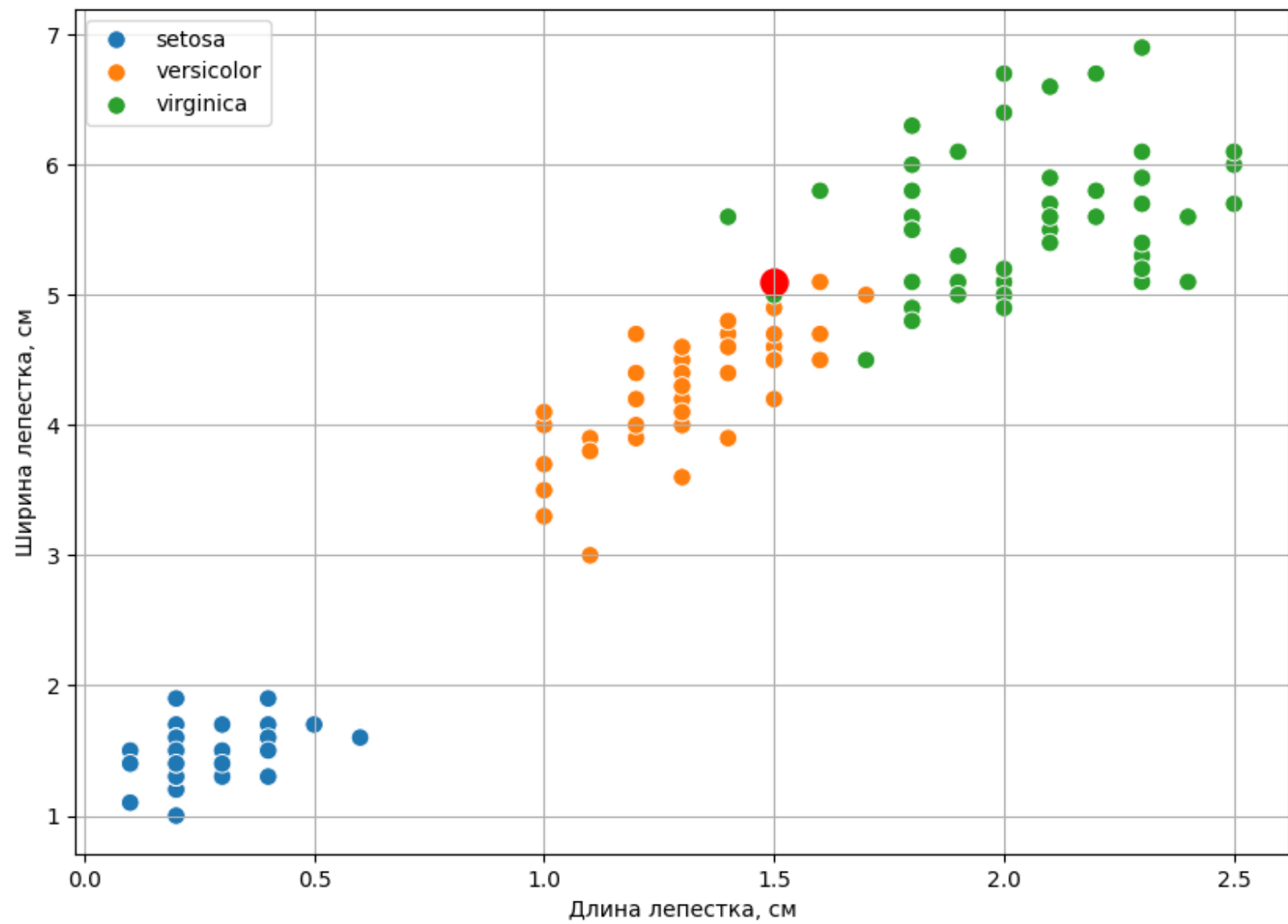
70	versicolor
73	versicolor
136	virginica
39	setosa
97	versicolor

Name: species, dtype: object

```

['setosa' 'setosa' 'versicolor' 'versicolor' 'setosa' 'virginica'
 'virginica' 'versicolor' 'virginica' 'virginica' 'versicolor' 'setosa'
 'versicolor' 'versicolor' 'versicolor' 'setosa' 'setosa' 'virginica'
 'virginica' 'setosa' 'setosa' 'virginica' 'setosa' 'virginica'
 'versicolor' 'virginica' 'setosa' 'versicolor' 'setosa' 'setosa']
accuracy: 0.967

```



```
...
```

### 2.3.1

Для предыдущего примера поэкспериментируйте с параметрами классификатора:

1. Установите другое количество ближайших соседей ( $k = 1, 5, 10$ ).
2. Установите размер тестовой выборки 15% от всего датасета.
3. Постройте графики и оцените качество моделей, проанализируйте результаты.

```
...
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
iris = sns.load_dataset('iris')

cnt=1
k = [1, 5, 10]
for a in k:
    print(f"Для ближайших соседей = {a}:")
    x_train, x_test, y_train, y_test = train_test_split(iris.iloc[:, :-1], iris.iloc[:, -1], test_size=0.15)
    x_train.shape, x_test.shape, y_train.shape, y_test.shape
    print(x_train.head())
    print(y_train.head())

    model = KNeighborsClassifier(n_neighbors=a)
    model.fit(x_train, y_train)

    y_pred = model.predict(x_test)
    print(y_pred)

    plt.figure(figsize=(10,7))
    sns.scatterplot(
        data=iris,
        x = 'petal_width', y = 'petal_length',
        hue='species',
        s=70)
```

```

plt.xlabel('Длина лепестка, см')
plt.ylabel('Ширина лепестка, см')
plt.legend(loc=2)
plt.grid()

for i in range(len(y_test)):
    if np.array(y_test)[i] != y_pred[i]:
        plt.scatter(x_test.iloc[i,3],x_test.iloc[i,2], color='red', s=150 )

plt.savefig(f"2.3.1.{cnt}.png")
cnt = cnt+1
from sklearn.metrics import accuracy_score
print(f'accuracy: {accuracy_score(y_test, y_pred):.3}')

```

#### Program output:

Для ближайших соседей = 1:

	sepal_length	sepal_width	petal_length	petal_width
36	5.5	3.5	1.3	0.2
37	4.9	3.6	1.4	0.1
77	6.7	3.0	5.0	1.7
56	6.3	3.3	4.7	1.6
125	7.2	3.2	6.0	1.8

36        setosa

37        setosa

77        versicolor

56        versicolor

125       virginica

Name: species, dtype: object

```

['virginica' 'versicolor' 'virginica' 'virginica' 'versicolor' 'virginica'
 'virginica' 'virginica' 'virginica' 'versicolor' 'setosa' 'virginica'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica'
 'virginica' 'virginica' 'setosa' 'versicolor']

```

accuracy: 0.87

Для ближайших соседей = 5:

	sepal_length	sepal_width	petal_length	petal_width
115	6.4	3.2	5.3	2.3
127	6.1	3.0	4.9	1.8
62	6.0	2.2	4.0	1.0
31	5.4	3.4	1.5	0.4
40	5.0	3.5	1.3	0.3

115 virginica

127 virginica

62 versicolor

31 setosa

40 setosa

Name: species, dtype: object

```
['virginica' 'versicolor' 'versicolor' 'setosa' 'versicolor' 'versicolor'
 'virginica' 'virginica' 'setosa' 'versicolor' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'versicolor' 'virginica' 'virginica'
 'versicolor' 'setosa' 'virginica' 'setosa' 'virginica']
```

accuracy: 0.957

Для ближайших соседей = 10:

	sepal_length	sepal_width	petal_length	petal_width
12	4.8	3.0	1.4	0.1
43	5.0	3.5	1.6	0.6
148	6.2	3.4	5.4	2.3
60	5.0	2.0	3.5	1.0
78	6.0	2.9	4.5	1.5

12 setosa

43 setosa

148 virginica

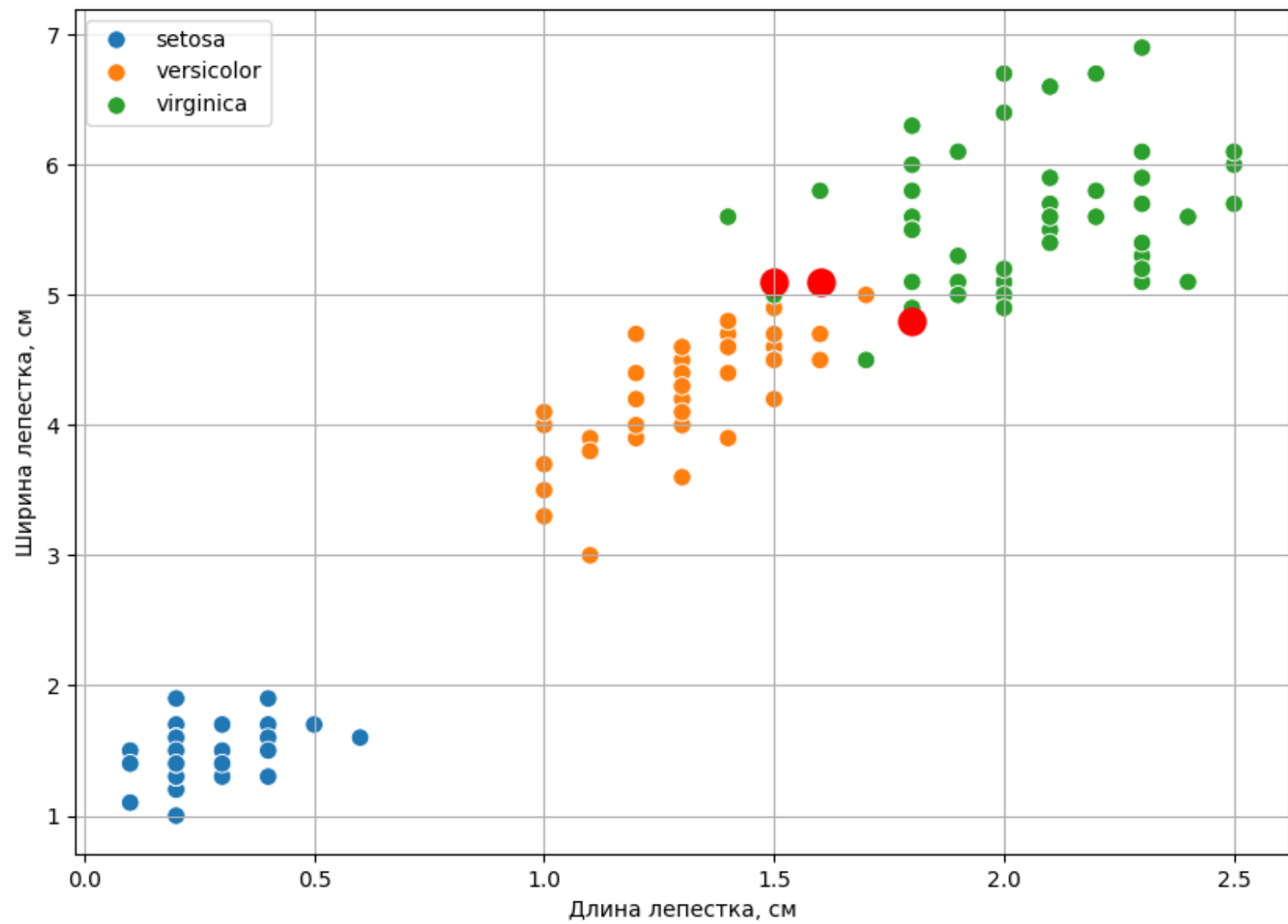
60 versicolor

78 versicolor

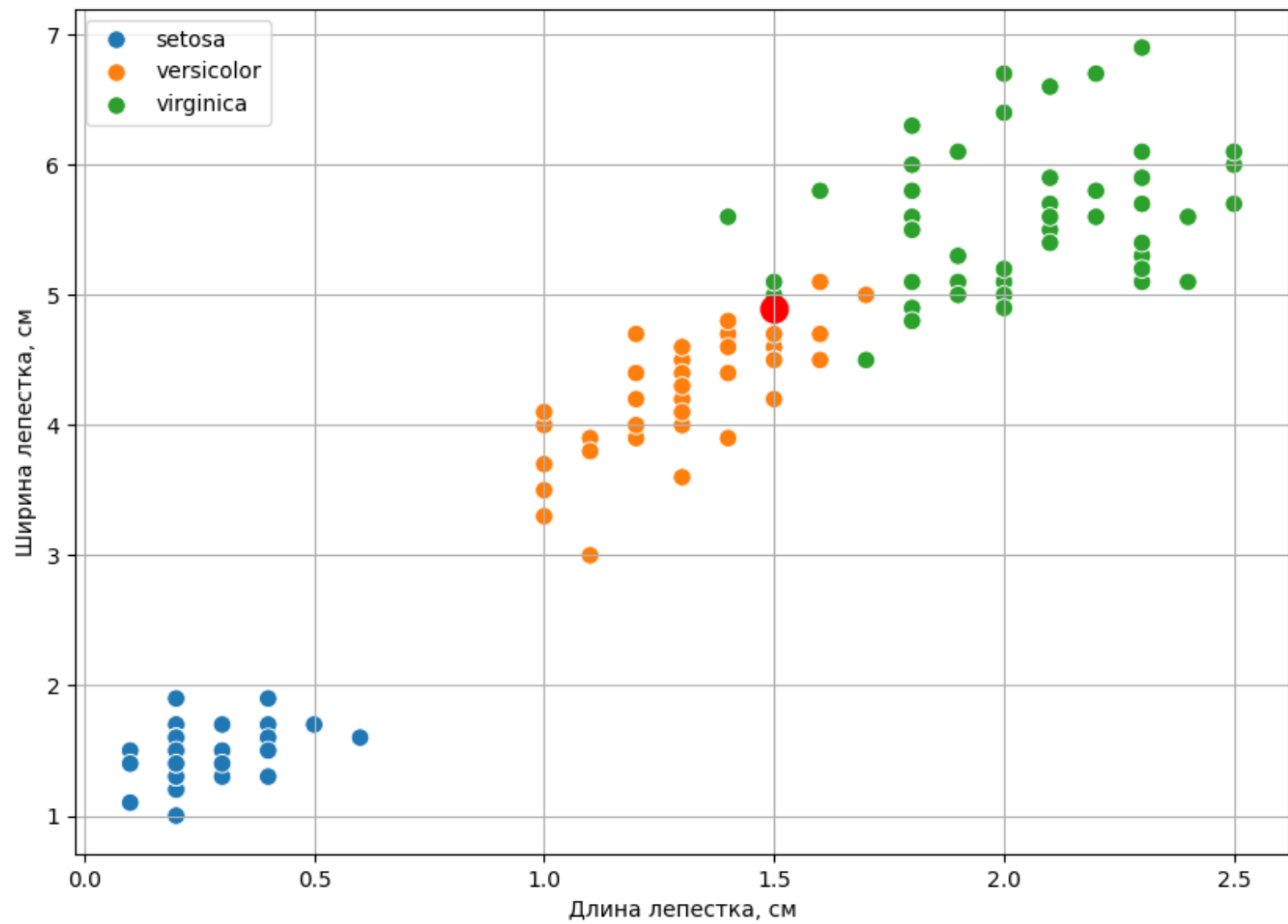
Name: species, dtype: object

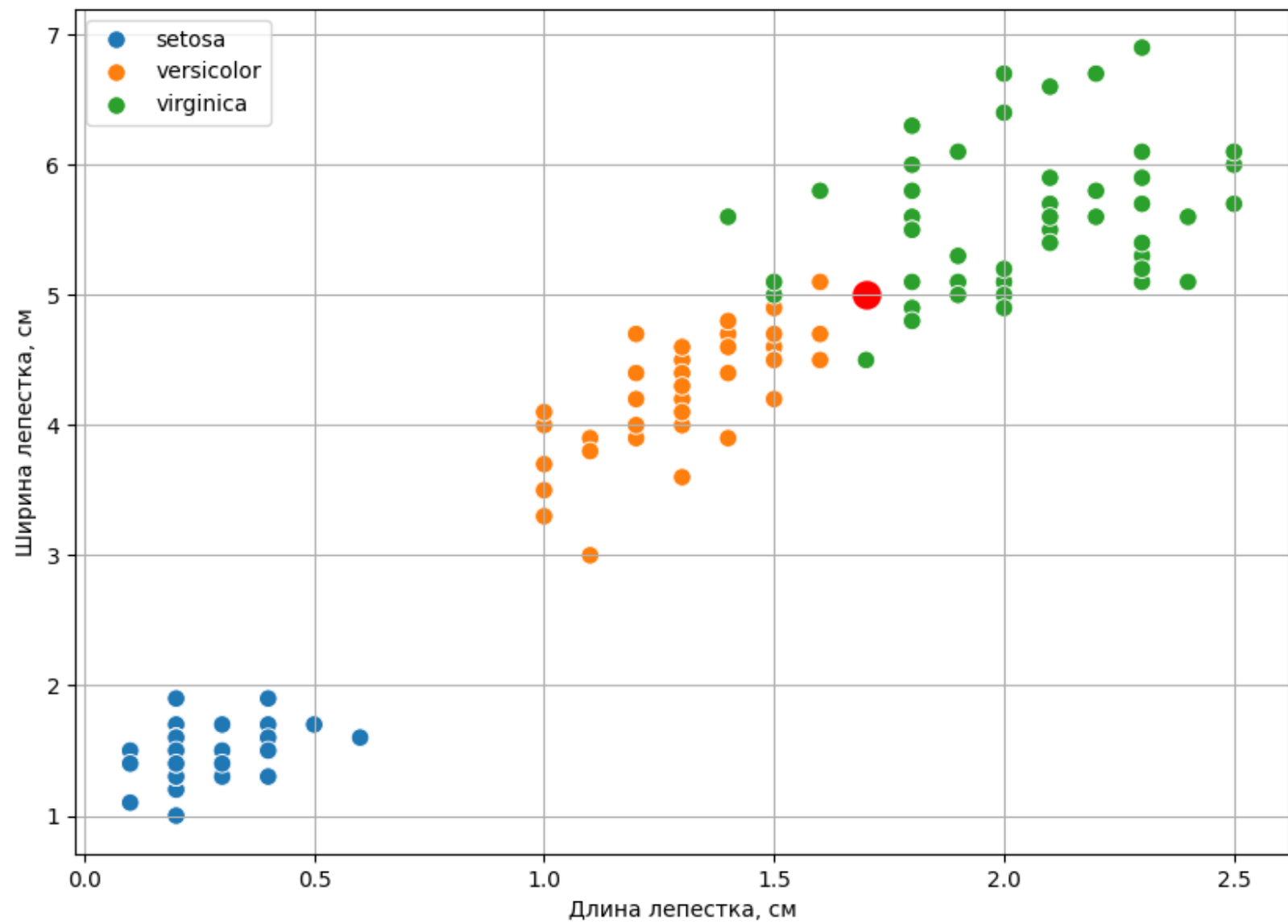
```
['versicolor' 'virginica' 'versicolor' 'setosa' 'virginica' 'setosa'
 'virginica' 'virginica' 'setosa' 'virginica' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'setosa' 'versicolor'
 'versicolor' 'virginica' 'versicolor' 'virginica' 'virginica']
```

accuracy: 0.957









```
'''
```

### 3.2.1

Дан порядковый категориальный признак (например, Высокий, средний, низкий). Выполнить его кодировку. Для решения задачи можно использовать метод replace фрейма данных pandas для преобразования строковых меток в числовые эквиваленты

```
'''
```

```
import pandas as pd
```

```
dataframe = pd.DataFrame({"оценка":["низкая", 'низкая', 'средняя', 'средняя', 'Высокая']})
```

```
scale_mapper = {"низкая":1, 'средняя':2, 'Высокая':3}
```

```
print(dataframe["оценка"].replace(scale_mapper))
```

**Program output:**

```
0    1
```

```
1    1
```

```
2    2
```

```
3    2
```

```
4    3
```

```
Name: оценка, dtype: int64
```

```
'''
```

### 3.2.2

Дан словарь, и требуется его конвертировать в матрицу признаков. Для решения задачи можно задействовать класс-Векторизатор словаря DictVectorizer:

```
'''
```

```
from sklearn.feature_extraction import DictVectorizer
```

```
data_dict = [{"красный": 2, "синий":4},  
             {"красный": 4, "синий":3},  
             {"красный": 1, "желтый":2},  
             {"красный": 2, "желтый":2}]
```

```
dictvectorizer = DictVectorizer(sparse=False)  
features = dictvectorizer.fit_transform(data_dict)  
print(features)
```

**Program output:**

```
[[0. 2. 4.]  
 [0. 4. 3.]  
 [2. 1. 0.]  
 [2. 2. 0.]]
```

```
'''
```

### 3.3.2

Определите набор признаков человека, по аналогии из РТ 1, – например, цвет глаз и конвертируйте его в матрицу признаков.

```
'''
```

```
from sklearn.feature_extraction import DictVectorizer
```

```
data_dict = [{"карий": 2, "голубой":4, "серый":3},  
             {"карий": 1, "зеленый":1, "серый":1},  
             {"зеленый": 2, "голубой":3, "серый":3},  
             {"карий": 3, "голубой":4, "зеленый":4}]
```

```
dictvectorizer = DictVectorizer(sparse=False)  
features = dictvectorizer.fit_transform(data_dict)  
print(features)
```

**Program output:**

```
[[4. 0. 2. 3.]  
 [0. 1. 1. 1.]  
 [3. 2. 0. 3.]  
 [4. 4. 3. 0.]]
```