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.

In [2]:
```
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))
```

```
77
0
5
```

1.2.2 - Напишите рекурсивную функцию вычисления факториала на языке Python.

In [3]:
```
def fact(num):
    if num == 0:
        return 1
    else:
        return num * fact(num - 1)

print(fact(5))
```

```
120
```

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

In [4]:
```
import numpy as np

def euclidean_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(euclidean_distance(x, y))
```

```
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.

In [5]:
```
def sqr_euclidean_distance(v1, v2):
    return sum((x - y) ** 2 for x, y in zip(v1, v2))

def weight_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):
```

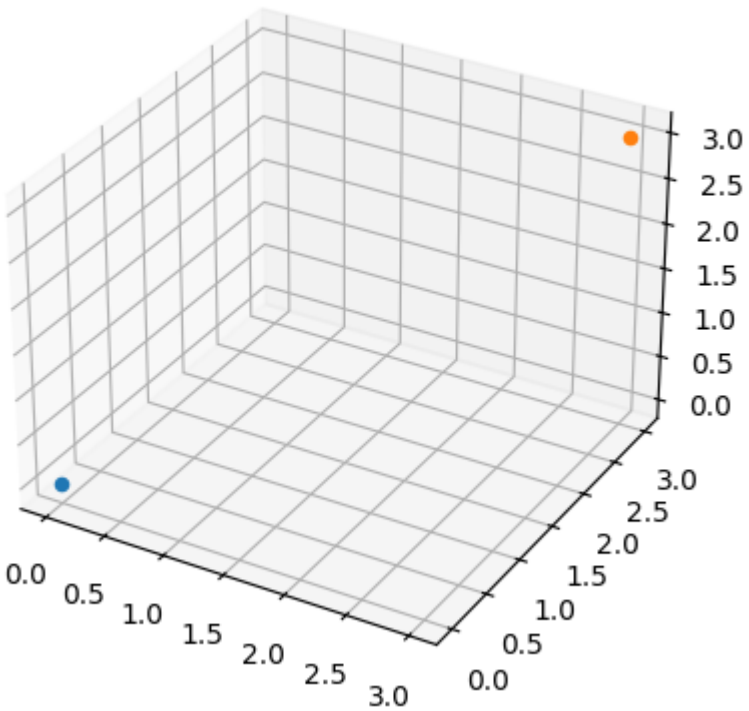
```
    return max(abs(x - y) for (x, y) in zip(v1, v2))

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(weight_euclidean_distance(x,y,w))
print(manhattan_distance(x,y))
print(chebyshev_distance(x,y))
```

```
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 помошью специальных функциий. Например, для серий это будет выглядеть следующим образом: seriesName.to_numpy().

Для удобства визуализации и анализа результатов в дальнейших расчетах будм использовать 2 точки в 3-х мерном пространстве:

In [6]:
```
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.show()
```



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

In [7]:
```
print(np.linalg.norm(x-y)) # Евклид
print(np.linalg.norm(x-y) ** 2) # Квадрат Евклида
print(np.linalg.norm(x-y, ord=np.inf)) # Чебышев
print(np.linalg.norm(x-y, ord=1)) # Хемминг
```

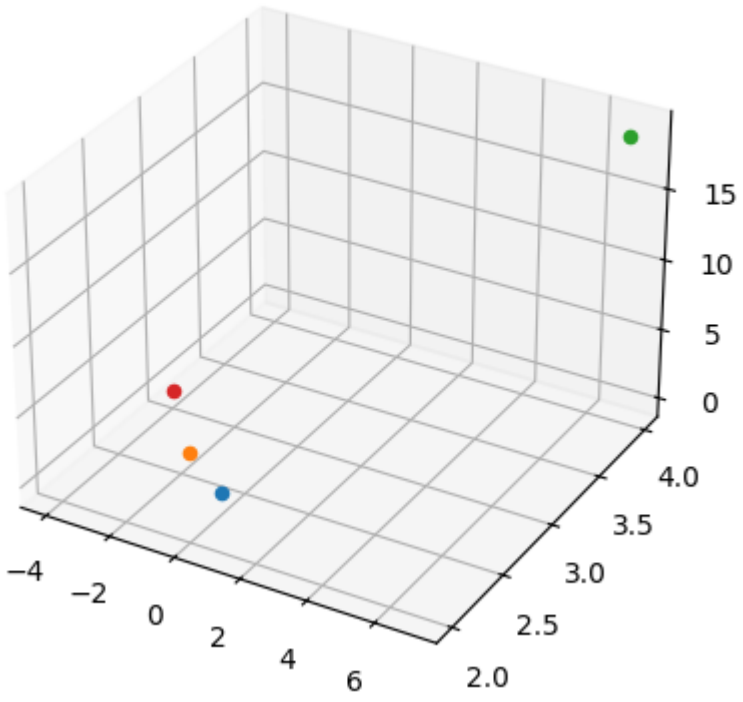
```
5.196152422706632
27.0
3.0
9.0
```

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

In [8]:
```
a = np.array([1, 2, 3])
b = np.array([0, 2, 5])
c = np.array([7, 4, 19])
d = np.array([-4, 3, 0])

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), end='\n \n')

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), end='\n \n')

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), end='\n \n')

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), end='\n \n')

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), end='\n \n')

print(np.linalg.norm(c-d))
print(np.linalg.norm(c-d) ** 2)
print(np.linalg.norm(c-d, ord=np.inf))
print(np.linalg.norm(c-d, ord=1), end='\n \n')

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

ax.scatter(a[0], a[1], a[2])
ax.scatter(b[0], b[1], b[2])
ax.scatter(c[0], c[1], c[2])
```

```
ax.scatter(d[0], d[1], d[2])
plt.show()
```

```
2.23606797749979
5.000000000000001
2.0
3.0

17.204650534085253
296.0
16.0
24.0

5.916079783099616
35.0
5.0
9.0

15.7797338380595
248.99999999999997
14.0
23.0

6.48074069840786
42.0
5.0
10.0

21.97726097583591
483.0
19.0
31.0
```



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

In [9]:
```
Z = np.zeros((5,5))
Z += np.arange(5)
print(Z)
```

```
[[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 обучене классификатора k-ближайших соседей по исходным данным. Далее производится прогноз принадлежности к классам для двух точек данных.

In [10]:
```
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', model.predict([[-2,-2]]))
print( '(1,3) is class', model.predict([[1,3]]))
```

```
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 стобцах записаны длина и ширина чашелистика, длина и ширина лепестка, вид ириса

In [11]:
```
import seaborn as sns

iris = sns.load_dataset('iris')
iris
```

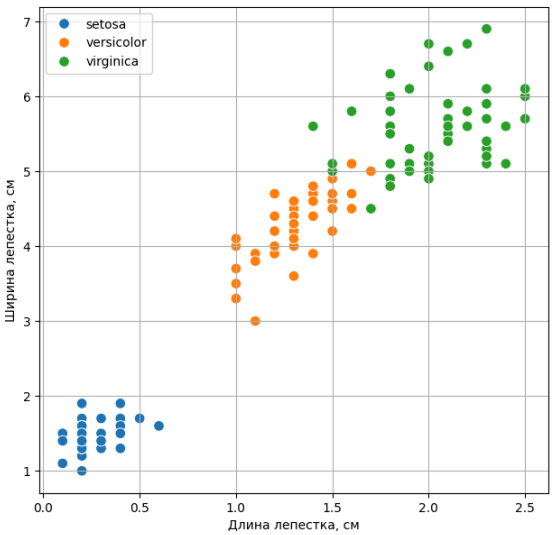
Out[11]:

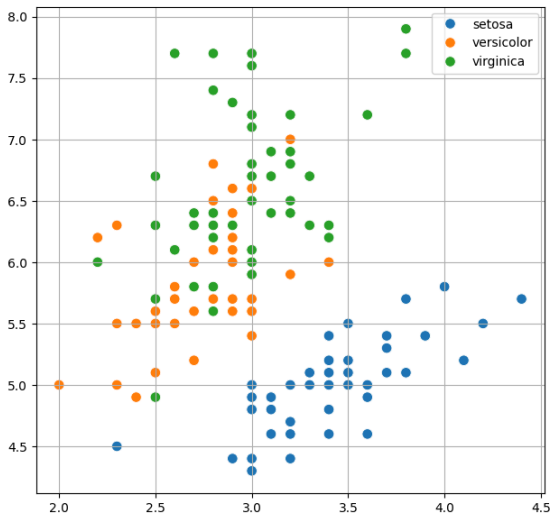
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148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

150 rows × 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.

In [12]:
```
# Объявляем фигуру из двух графиков и ее размер
plt.figure(figsize=(16, 7))

# Левый график
plt.subplot(121)
sns.scatterplot(
    data=iris, # из этой таблицы нарисовать точки
    x='petal_width', y='petal_length', # с этими координатами
    hue='species', # для которых определить цвет согласно этому столбцу
    s = 70 # размер точек
)
plt.xlabel('Длина лепестка, см')
plt.ylabel('Ширина лепестка, см')
plt.legend() # добавить легенду
plt.grid() # добавить сетку

# аналогично чашелистики
plt.subplot(122)
sns.scatterplot(
    data=iris,
    x='sepal_width', y='sepal_length',
    hue='species',
    s = 70
)
plt.xlabel('')
plt.ylabel('')
plt.legend()
plt.grid()
```





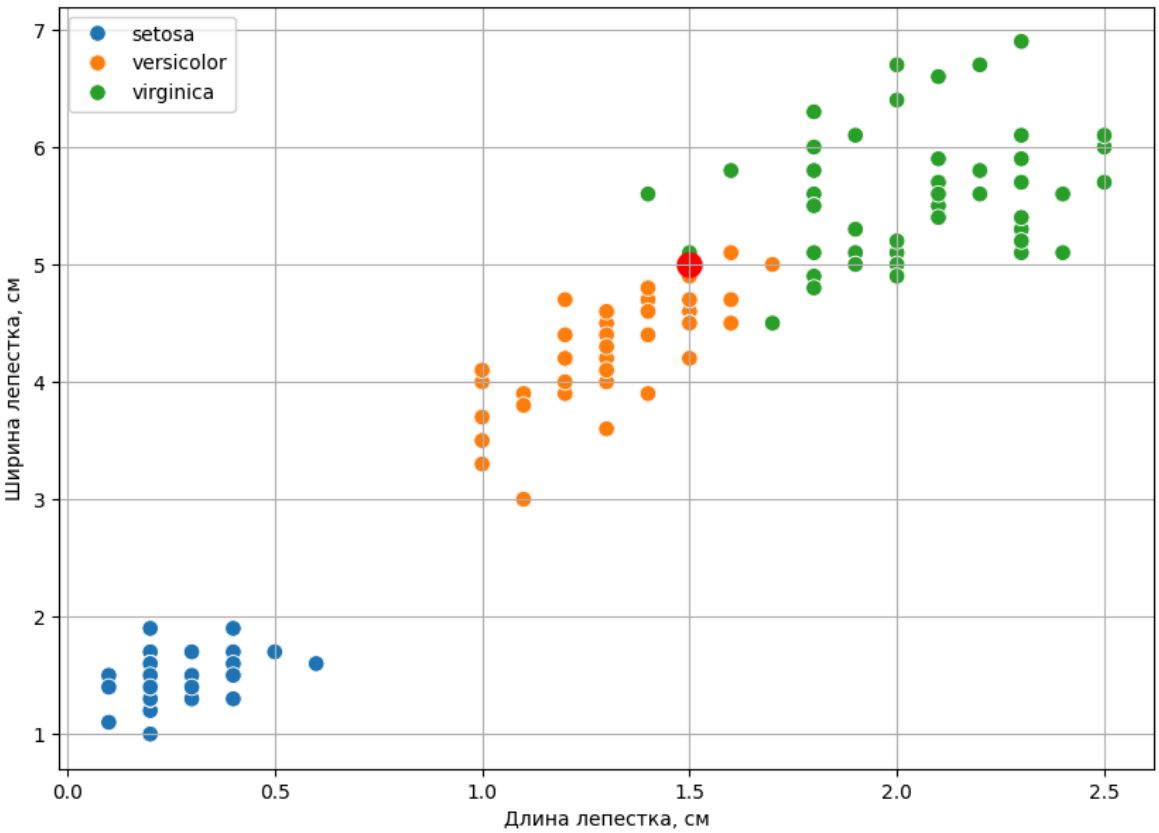
2.2.4 - Из графиков видно, что в первом случае классы визуально хорошо отделимы друг от друга, хотя два классе имеют небольшое пересечение. Во втором случае разделить два класса между собой уже намного труднее.

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.

In [13]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    #поскольку iris это pandas-таблица, для нее нужно указывать iloc
    iris.iloc[:, :-1], # берем все колонки кроме последней в признаки
    iris.iloc[:, -1], # последнюю в целевую переменную (класс)
    test_size = 0.20 # размер тестовой выборки 20%
)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print(X_train.head())
print(y_train.head())

# Обучим метод трех ближайших соседей
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
# Получим предсказания модели
y_pred = model.predict(X_test)
print(y_pred)

# Покажем на графике, что отражает полученное число.
# Красным цветом обозначены точки, для которых классификация сработала неправиль
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()

# Перебираем все объекты из теста
for i in range(len(y_test)):
    # Если предсказание неправильное
    if np.array(y_test)[i] != y_pred[i]:
        # то подсвечиваем точку красным
        plt.scatter(X_test.iloc[i, 3], X_test.iloc[i, 2], color='red', s=150)

# качество модели (доля правильно классифицированных точек)
from sklearn.metrics import accuracy_score
print(f'accuracy: {accuracy_score(y_test, y_pred) :.3}')
```

```
(120, 4) (30, 4) (120,) (30,)
     sepal_length  sepal_width  petal_length  petal_width
56            6.3          3.3           4.7          1.6
63            6.1          2.9           4.7          1.4
17            5.1          3.5           1.4          0.3
124           6.7          3.3           5.7          2.1
33            5.5          4.2           1.4          0.2
56     versicolor
63     versicolor
17         setosa
124     virginica
33         setosa
Name: species, dtype: object
['versicolor' 'virginica' 'versicolor' 'setosa' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa'
 'setosa' 'setosa' 'versicolor' 'setosa' 'versicolor' 'virginica'
 'virginica' 'setosa' 'versicolor' 'setosa' 'versicolor' 'virginica'
 'virginica' 'virginica' 'virginica' 'setosa' 'setosa' 'setosa']
accuracy: 0.967
```

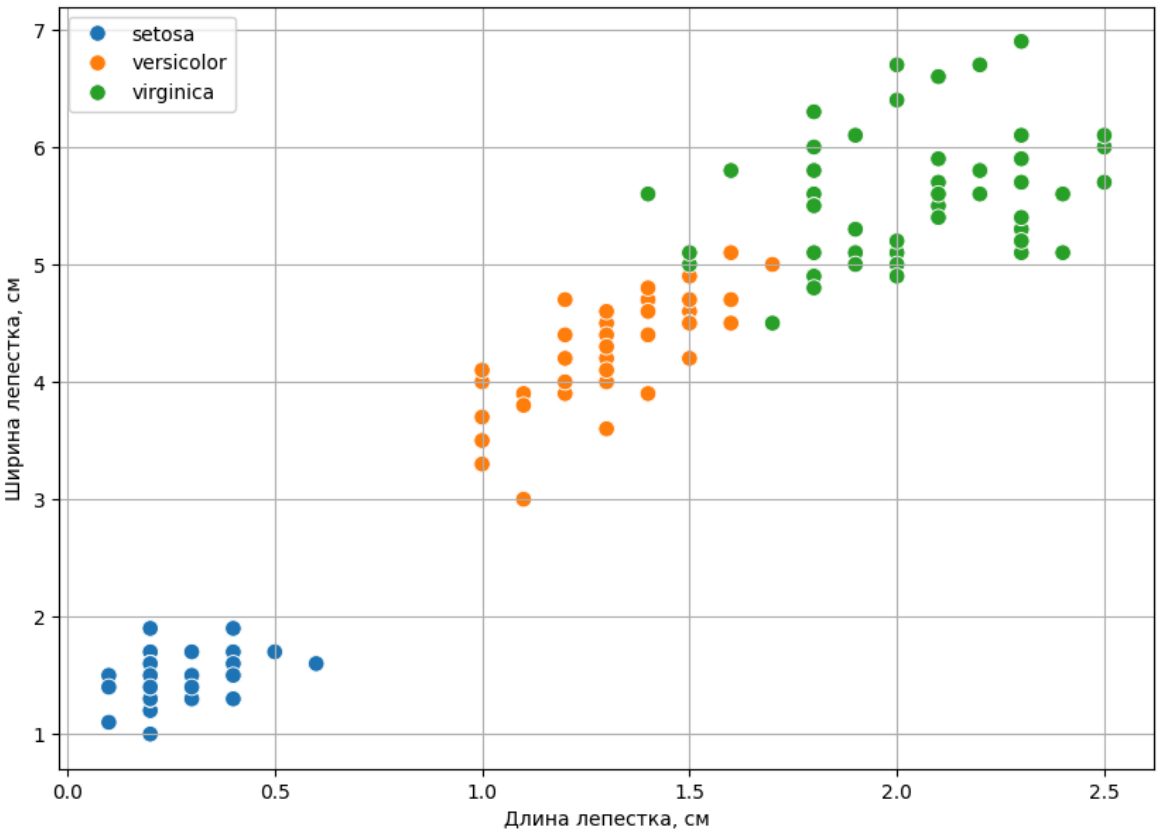


2.3.1 - Для предыдущего примера поэкспериментируйте с параметрами классификатора:

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.

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(
    #поскольку iris это pandas-таблица, для нее нужно указывать iloc
    iris.iloc[:, :-1], # берем все колонки кроме последней в признаки
    iris.iloc[:, -1], # последнюю в целевую переменную (класс)
    test_size = 0.2 # размер тестовой выборки 20%
)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print(X_train.head())
print(y_train.head())

# Обучим метод трех ближайших соседей
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
# Получим предсказания модели
y_pred = model.predict(X_test)
print(y_pred)

# Покажем на графике, что отражает полученное число.
# Красным цветом обозначены точки, для которых классификация сработала неправиль
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()

# Перебираем все объекты из теста
for i in range(len(y_test)):
    # Если предсказание неправильное
    if np.array(y_test)[i] != y_pred[i]:
        # то подсвечиваем точку красным
        plt.scatter(X_test.iloc[i, 3], X_test.iloc[i, 2], color='red', s=150)

# качество модели (доля правильно классифицированных точек)
from sklearn.metrics import accuracy_score
print(f'accuracy: {accuracy_score(y_test, y_pred) :.3}')
```

```
(120, 4) (30, 4) (120,) (30,)
     sepal_length  sepal_width  petal_length  petal_width
71            6.1          2.8           4.0          1.3
53            5.5          2.3           4.0          1.3
3             4.6          3.1           1.5          0.2
143           6.8          3.2           5.9          2.3
74            6.4          2.9           4.3          1.3
71     versicolor
53     versicolor
3          setosa
143     virginica
74     versicolor
Name: species, dtype: object
['versicolor' 'setosa' 'setosa' 'virginica' 'setosa' 'setosa' 'versicolor'
 'setosa' 'versicolor' 'setosa' 'virginica' 'virginica' 'setosa'
 'versicolor' 'virginica' 'setosa' 'setosa' 'versicolor' 'setosa' 'setosa'
 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa' 'setosa' 'setosa'
 'virginica' 'virginica' 'setosa']
accuracy: 1.0
```



- При увеличении кол-ва ближайших соседей количество ошибочных предсказаний уменьшается
- При уменьшении размера тестовой выборки должно наблюдаться уменьшение количества неправильных предсказаний, однако зависимость незаметна

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

In [25]:
```
# загрузить библиотеку
import pandas as pd

#Создать признаки
dataframe = pd.DataFrame({"оценка": ["низкая", "низкая", "средняя", "средняя", "
# Создать словарь преобразования шкалы
scale_mapper = {"низкая":1,"средняя":2,"высокая":3}

# Заменить значения признаков значениями словаря
dataframe["оценка"].replace(scale_mapper)
```

Out[25]:
```
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:

In [26]:
```
from sklearn.feature_extraction import DictVectorizer
# создать словарь
data_dict = [{"красный": 2, "синий": 4},
             {"красный": 4, "синий": 3},
             {"красный": 1, "желтый": 2},
             {"красный": 2, "желтый": 2}]
# Создать векторизатор словаря
dictvectorizer = DictVectorizer(sparse=False)
# Конвертировать словарь в матрицу признаков
features = dictvectorizer.fit_transform(data_dict)
features
```

Out[26]:
```
array([[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.

In [27]:
```
dataframe = pd.DataFrame({"цвет глаз": ["карий", "карий", "голубой", "зеленый",

scale_mapper = {"карий":1, "голубой":2, "зеленый":3, "серый":4, "черный":5}

dataframe["цвет глаз"].replace(scale_mapper)
```

Out[27]:
```
0    1
1    1
2    2
3    3
4    1
5    4
6    5
7    2
Name: цвет глаз, dtype: int64
```