Реализуем алгоритм kNN с помощью Python.

In [1]:

```
import numpy as np
from sklearn import model_selection
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from scipy.stats import rankdata
```

Загрузим один из "игрушечных" датасетов из sklearn.

In [2]:

```
X, y = load_iris(return_X_y=True)

# Для наглядности возьмем только первые два признака (всего в датасете их 4)
X = X[:, :2]
```

Разделим выборку на обучающую и тестовую

In [3]:

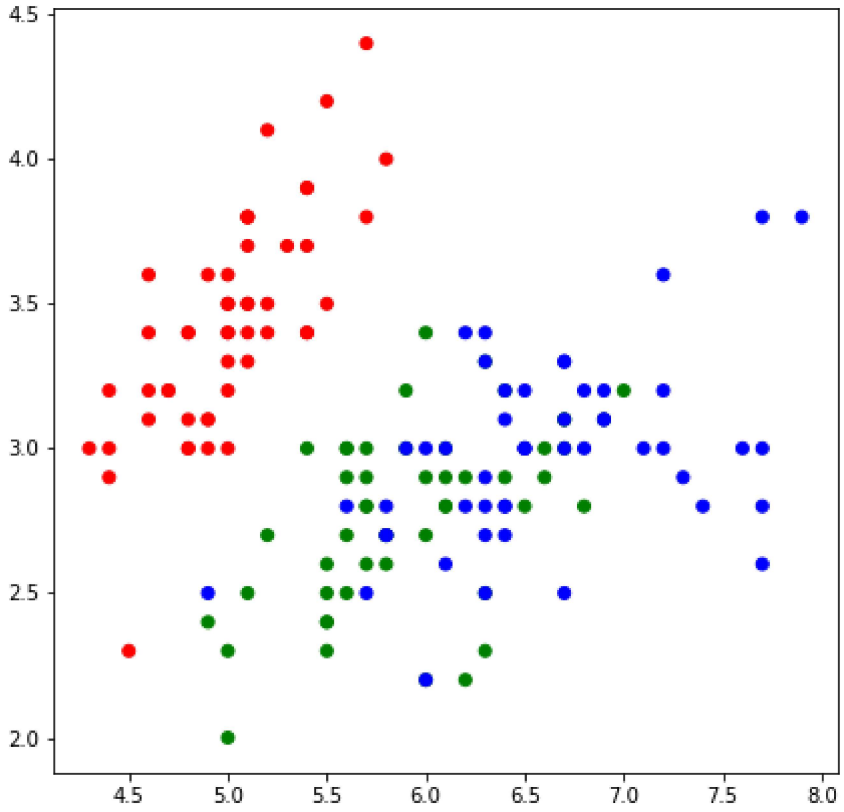
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.2, ra
```

In [4]:

```
cmap = ListedColormap(['red', 'green', 'blue'])
plt.figure(figsize=(7, 7))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap)
```

Out[4]:

```
<matplotlib.collections.PathCollection at 0x166db5f2188>
```



Используем евклидову метрику. Реализуем функцию для ее подсчета.

In [5]:

```
def e_metrics(x1, x2):

    distance = 0
    for i in range(len(x1)):
        distance += np.square(x1[i] - x2[i])

    return np.sqrt(distance)
```

Реализуем алгоритм поиска k ближайших соседей.

In [6]:

```
def knn(x_train, y_train, x_test, k):

    answers = []

    for x in x_test:
        test_distances = []

        for i in range(len(x_train)):

            # расчет расстояния от классифицируемого объекта до
            # объекта обучающей выборки
            distance = e_metrics(x, x_train[i])

            # Записываем в список значение расстояния и ответа на объекте обучающей выборки
            test_distances.append((distance, y_train[i]))

        # создаем словарь со всеми возможными классами
        classes = {class_item: 0 for class_item in set(y_train)}
        sorted_distance = sorted(test_distances)[0:k]

        sorted_rank = rankdata(sorted_distance, method='min')

        # Сортируем список и среди первых k элементов подсчитаем частоту появления разных к
        for d in sorted_distance:
            #dist = sorted_distance[d][0]
            #class_label = int(sorted_rank[d])
            #print(dist, class_label)
            classes[d[1]] += 1/(d[0] + 1e-3)


        # Записываем в список ответов наиболее часто встречающийся класс
        answers.append(sorted(classes, key=classes.get)[-1])
    return answers
```

Напишем функцию для вычисления точности

In [7]:

```
def accuracy(pred, y):
    return (sum(pred == y) / len(y))
```

Проверим работу алгоритма при различных k

In [8]:

```
k = 1

y_pred = knn(X_train, y_train, X_test, k)

print(f'Точность алгоритма при k = {k}: {accuracy(y_pred, y_test):.3f}')
```

```
Точность алгоритма при k = 1: 0.667
```

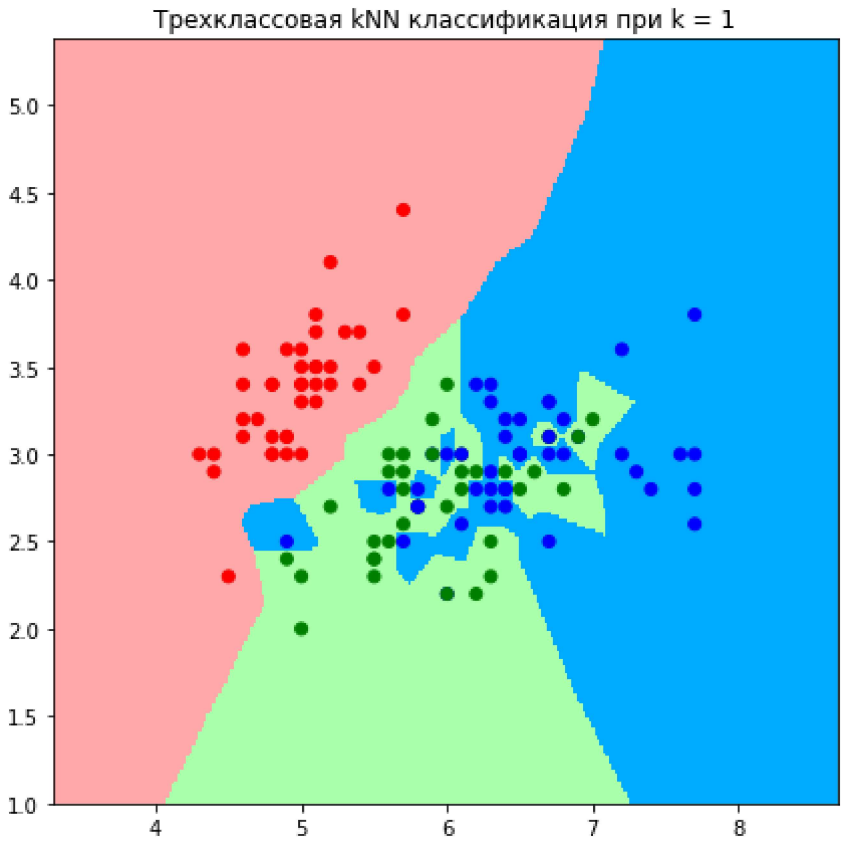
Построим график распределения классов.

In [9]:

```
def get_graph(X_train, y_train, k):
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA','#00AAFF'])

    h = .02

    # Расчет пределов графика
    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
    y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Получим предсказания для всех точек
    Z = knn(X_train, y_train, np.c_[xx.ravel(), yy.ravel()], k)

    # Построим график
    Z = np.array(Z).reshape(xx.shape)
    plt.figure(figsize=(7,7))
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Добавим на график обучающую выборку
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(f"Трехклассовая kNN классификация при k = {k}")
    plt.show()
```

In [10]:

```
get_graph(X_train, y_train, k)
```



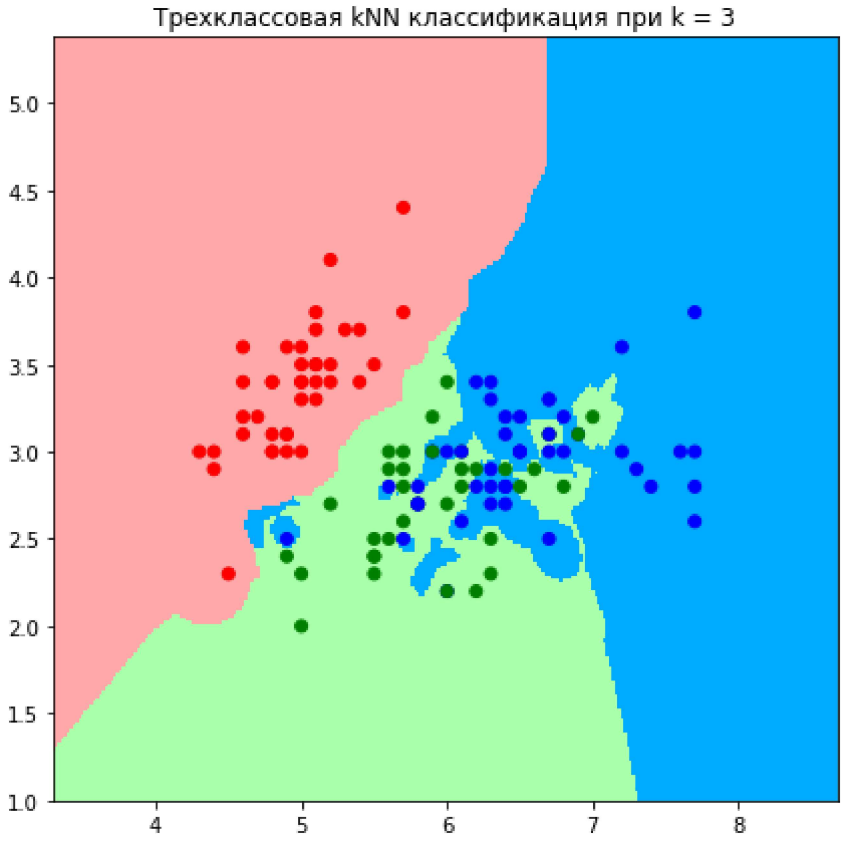
In [11]:

```
k = 3

y_pred = knn(X_train, y_train, X_test, k)

print(f'Точность алгоритма при k = {k}: {accuracy(y_pred, y_test):.3f}')
```

```
Точность алгоритма при k = 3: 0.733
```

In [12]:

```
get_graph(X_train, y_train, k)
```



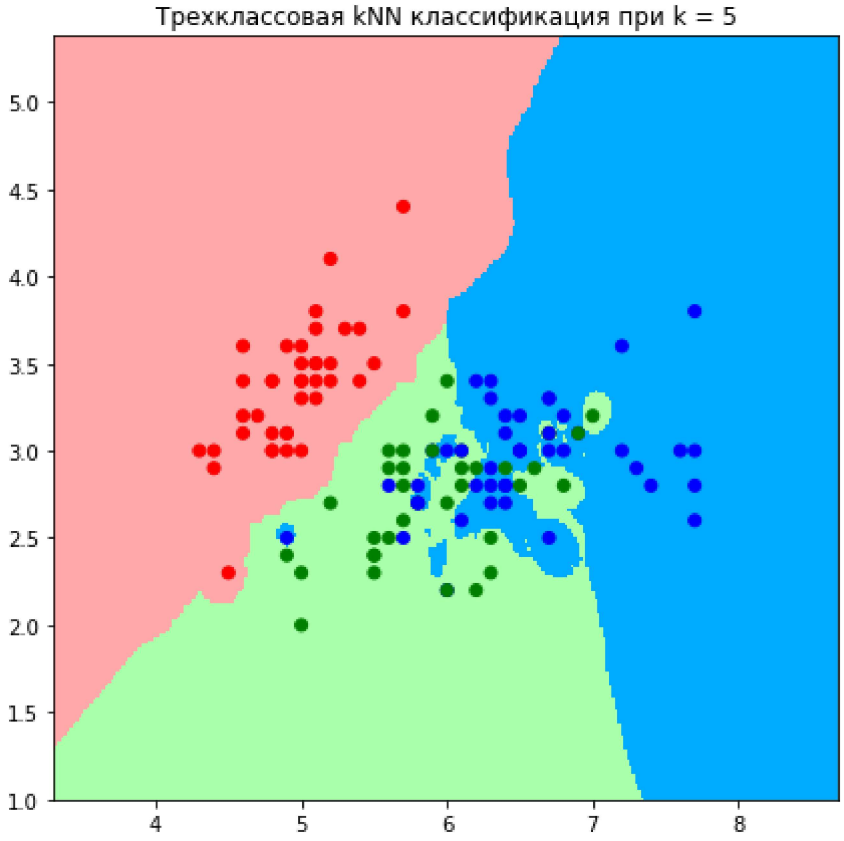
In [13]:

```
k = 5

y_pred = knn(X_train, y_train, X_test, k)

print(f'Точность алгоритма при k = {k}: {accuracy(y_pred, y_test):.3f}')
```

```
Точность алгоритма при k = 5: 0.833
```

In [14]:

```
get_graph(X_train, y_train, k)
```



При увеличении k мы на графиках наблюдаем, как алгоритм меньше концентрируется на выбросах, однако, точность на тестовой выборке при этом увеличивается.

Рассматриваемый метод, несмотря на положительные стороны в виде легкости интерпретации, простоты и удобства использования, обладает некоторыми минусами, в частности, он плохо работает на датасетах с большим количеством признаков.

Например, если мы имеем три объекта, при этом второй отличается от первого только значением одного признака, но значительно, а третий отличается от первого незначительно в каждом признаке, расстояния от первого объекта до второго и третьего могут совпадать. Несущественные различия в каждом признаке могут иметь большее значение, чем большое различие в одном признаке. Такое поведение в ряде случаев будет нежелательным.

Второй пример - случай, когда количество признаком сравнимо с количеством объектов. В этом случае может возникнуть ситуация, когда расстояния между любыми двумя объектами почти одинаковы. В двумерном пространстве (на плоскости) три точки могут располагаться по вершинам равностороннего треугольника, при этом расстояния между ними будут равны; в трехмерном пространстве то же самое

справедливо для четырех точек на вершинах тетраэдра - расстояние между любыми двумя точками будет одинаково. В общем случае это означает, что в -мерном пространстве можно выбрать точку так, чтобы расстояние между любыми двумя точками было одинаковым.

Третий пример - так называемое "проклятие размерности". Суть его заключается в том, что при наличии бинарных признаков в пространстве признаков будет возможно различных объектов вида , и размер обучающей выборки, необходимый, чтобы описать все пространство объектов (то есть все возможные комбинации таких признаков) также будет порядка . Чтобы покрыть не все пространство, а долю объектов (то есть долю объема) нужно будет описать гиперкуб с длиной ребра . Например, в 10-мерном пространстве признаков чтобы покрыть 1% объема нужно взять гиперкуб с длиной , то есть взять окрестность длиной больше половины ребра исходного пространства. Чем больше признаков, тем меньше будет область, которая покрывается во время поиска на заданном расстоянии. Таким образом, при сохранении требований по точности нахождения объекта в пространстве, количество требуемых данных для этого при увеличении количества признаков растет экспоненциально (подробнее про это явление см. в доп. материалах).