**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Основы машинного обучения»**

**Тема: Изучение и предобработка данных**

**Вариант 1**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Изучить способы анализа предоставленных данных с помощью языка Python и библиотек Pandas, NumPy, Seaborn, Matplotlib, Scikit-learn. Научиться с помощью данных библиотек анализировать датафрейм.

**Задание.**

При выполнении лабораторной работы также будет оцениваться количество инструкций, которые были использованы для получения результата.

1. Изучение набора данных iris.csv с использованием Pandas и Seaborn:
   1. Загрузить данные из файла как Pandas DataFrame
   2. Вызвав у датафрейма метод head, проверить корректность загруженных данных
   3. Вызвав у датафрейма метод describe, получить характеристики. Опишите полученный результат.
   4. Видоизмените полученный датафрейм таким образом, чтобы метка классов были следующими: 0 - Iris-setosa, 1 - Iris-versicolor, 2 - Iris-virginica. Сохраните полученный датафрейм в отдельный файл формата csv.
   5. Визуально оцените набор данных, построив изображение, содержащее графики ядерной оценки плотности каждого признака (кроме признака класса), диаграмму рассеяния и двумерную ядерную оценку плотности для каждых признаков. Наблюдения разных классов должны быть выделены отдельным цветом (рекомендуемая палитра ‘tab10’ или ‘Set1’). Пример построения: <https://seaborn.pydata.org/examples/pair_grid_with_kde.html> . Опишите полученный график, что на нем изображено, какие выводы о данных можно сделать.
   6. На одном изображении постройте гистограммы распределения для каждого признака (для построения нескольких диаграмм на одном изображении, необходимо создать subplot из matplotlib, и для каждой диаграммы задать параметр ax, указав нужную ячейку. subplot возвращает два параметра: саму фигуру с изображением и список ячеек. Например, изображение с 4 ячейками записанных в ряд: fig, axs = plt.subplots(1,4). Указание ячейки в параметре диаграммы делается следующим образом: ax=axs[0]). Затем последовательно модифицируйте изображение:
      1. Постройте гистограммы для разного количества столбцов: 5,10,15,20,30. Выберите на ваш взгляд такое количество столбцов, который лучшие образом описывает форму распределения признаков.
      2. Сделайте на каждой гистограмме разделение по цвету согласно классу. Проведите это в двух режимах, когда гистограммы накладываются/суммируются и когда пересекаются. Далее используйте режим с пересечением.
      3. Постройте гистограммы, чтобы вместо столбцов изображались ступеньки.
      4. Добавьте на гистограммы график ядерной оценки плотности.
2. Изучение набора данных iris.csv с использованием NumPy:
   1. Загрузите данные из файла как массив NumPy
   2. Выведите первые 10 наблюдений набора данных.
   3. Рассчитайте характеристики полученные методом describe в п. 1.3 с использованием методов NumPy
3. Изучение набора данных вашего варианта:
   1. Оцените и опишите набор данных вашего варианта с использованием методов в п. 1
4. Преобразование данных:
   1. Получите из датафрейма из п. 1.4 столбец с названием классов. Используя LabelEncoder и OneHotEncoder получите различные способы кодирования меток класса. В чем различия полученных кодировок?
   2. Для датафрейма из п. 1.4, получите все столбцы признаков (столбцы не содержащие метки классов). Преобразуйте полученные столбцы в массив NumPy.
   3. Для массива NumPy из п. 4.2 примените StandardScaler, MinMaxScaler, MaxAbsScaler и RobustScaler. Для каждого из результатов постройте гистограммы по каждому признаку без разделения по классам. В чем различия между такими преобразованиями данных?
   4. Согласно варианту, самостоятельно реализуйте StandardScaler или MinMaxScaler с использованием NumPy. Проверьте корректность работы на вашем наборе данных, сравните результаты между вашей реализацией и реализацией из Sklearn, а также рассчитав минимальное, максимальное, среднее значение и дисперсию, после преобразования.
   5. Для датафрейма из п. 1.4 получите новый, который содержит только классы Iris-versicolor и Iris-virginica, признаки “sepal length (cm)” и “petal length (cm)”, и наблюдения, для которых значения признака “sepal width (cm)” лежат между квантилями 25% и 75%.
5. Понижение размерности:
   1. Для набора данных iris.csv примените понижение размерности до 2, используя PCA и TSNE из Sklearn. Для каждого из результатов постройте диаграмму рассеяния с выделением разным цветом наблюдений разных классов. Объясните полученные результаты.

**Выполнение работы.**

1. Изучим данные из iris.csv, используя Pandas и Seaborn
   1. Загрузим данные из файла как Pandas DataFrame (*read\_csv*) (см. листинг 1.1).

Листинг 1.1 – Загрузка датасета

| df = pd.read\_csv("iris.csv") |
| --- |

* 1. Вызовем у датафрейма метод *head* и проверим корректность загруженных данных (см. листинг 1.2). Команда выведет первые 5 строк датафрейма. Вывод метода см. в таблице 1.1

Листинг 1.2 – Вызов *head*

| df.head() |
| --- |

Таблица 1.1 – Вывод *head*

| index | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
| --- | --- | --- | --- | --- | --- |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

* 1. Вызвав у датафрейма метод *describe*, получим характеристики (см. листинг 1.3). Результаты выполнения метода представлены в таблице 1.2.

Листинг 1.3 – Вызов *describe*

| df.describe() |
| --- |

Таблица 1.2 – вывод *describe*

|  | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
| --- | --- | --- | --- | --- | --- |
| count | 150.000000 | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.057333 | 3.758000 | 1.199333 | 1.000000 |
| std | 0.828066 | 0.435866 | 1.765298 | 0.762238 | 0.819232 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 | 0.000000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 | 0.000000 |
| 50% | 5.800000 | 3.000000 | 4.350000 | 1.300000 | 1.000000 |
| 75% | 6.400000 | 3.300000 | 5.100000 | 1.800000 | 2.000000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 | 2.000000 |

Была выведена информация для каждого признака в датафрейме (см. таблицу 1.2).

В строках хранятся следующие данные:

* count – количество непропущенных элементов.
* mean – среднее значение.
* min – минимальное значение.
* 25% – значение нижний квартили.
* 50% – значение медианы.
* 75% – значение верхней квартили.
* max – максимальное значение.
  1. Видоизменим полученный датафрейм с помощью метода *replace* таким образом, чтобы метки классов были следующими: 0 - Iris-setosa, 1 - Iris-versicolor, 2 - Iris-virginica (изменим значение признака target) и cохраним полученный датафрейм в отдельный файл new\_iris.csv с помощью метода *to\_csv* (см. листинг 1.4). Проверим получившийся датафрейм с помощью *head* (см. таблицу 1.3).

Листинг 1.4 – Изменение и сохранение датафрейма.

| df.replace(  {"target":  {0: "Iris-setosa",  1: "Iris-versicolor",  2: "Iris-virginica"}},  inplace=True)  df.to\_csv("new\_iris.csv", index=False) |
| --- |

Таблица 1.3 – Измененный датафрейм.

| index | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
| --- | --- | --- | --- | --- | --- |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

* 1. Визуально оценим набор данных, построив изображение, содержащее графики ядерной оценки плотности каждого признака, диаграмму рассеяния и двумерную ядерную оценку плотности для каждого признаков (см. рисунок 1.1). Используем для этого библиотеку Seaborn (см. листинг 1.5)

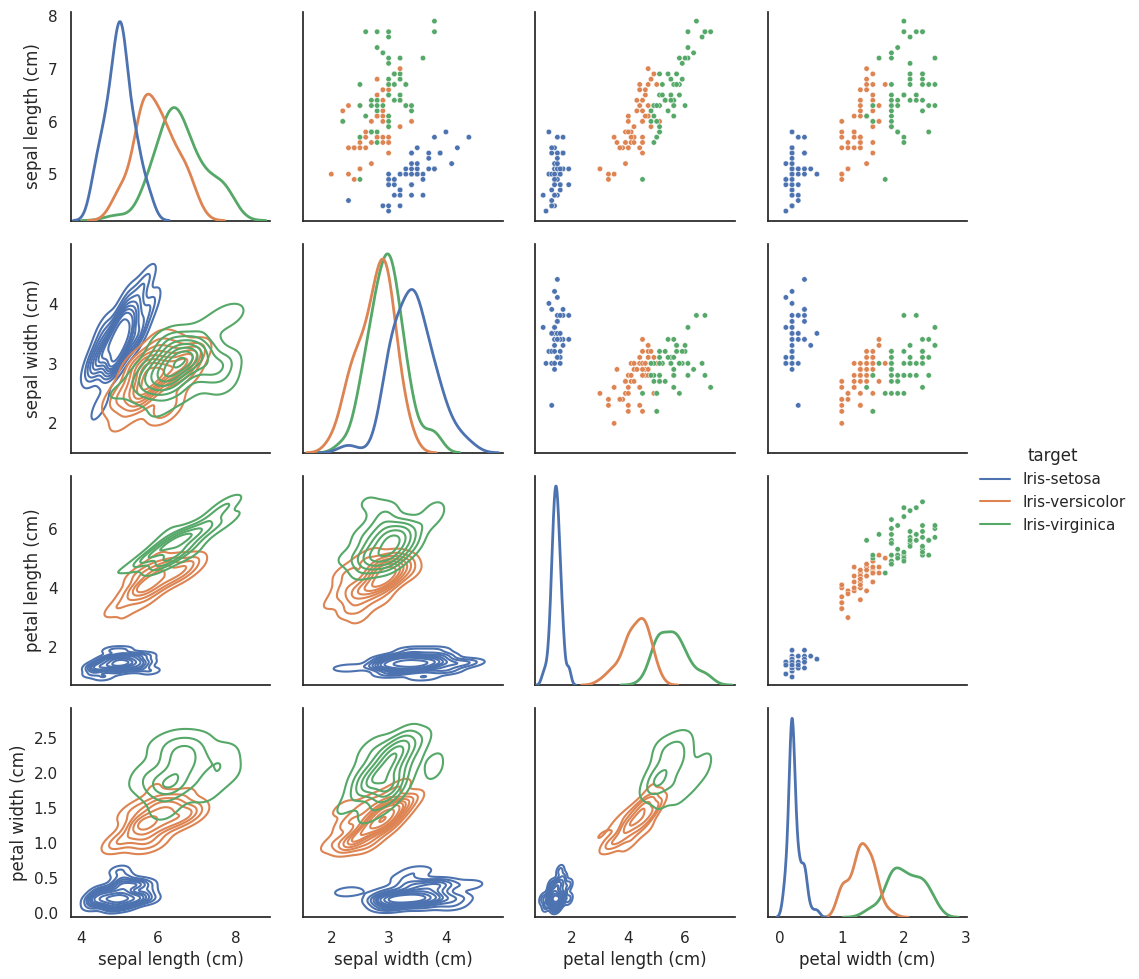


Рисунок 1.1 – Графики iris.csv

Листинг 1.5 – Рисование и вывод графиков.

| sns.set\_theme(style="white")  sns.color\_palette("tab10")  g = sns.PairGrid(df, diag\_sharey=False, hue="target")  g.map\_upper(sns.scatterplot, s=15)  g.map\_lower(sns.kdeplot)  g.map\_diag(sns.kdeplot, lw=2)  g.add\_legend() |
| --- |

Каждый класс на графиках имеет свой цвет в соответствии с палитрой «tab10». Для реализации разных цветов в атрибут *hue* передается *target* (столбец классов). С помощью метода *scatterplot* на правой верхней части сетки рисуется диаграмма рассеивания, с помощью метода *kdeplot* на левой нижней части и по диагонали сети рисуется двумерная ядерная оценка плотности и ядерная оценка плотности. Из графиков (см. рисунок 1.1) можно заметить, что по всем признакам классы «Iris-versicolor» и «Iris-virginica» смешиваются, в отличии от класса «Iris-setosa». Самое сильное смешивание классов проявляется у признака «sepal width (cm)» .

* 1. На одном изображении построим гистограммы распределения для каждого признака.
     1. Построим гистограммы для разного количества столбцов: 5, 10, 15, 20, 30 (см. рисунок 1.2). Из получившихся рисунков выберем наиболее лучше описывающий форму распределения признаков. Для рисования воспользуемся методом *histplot* и библиотеками Seaborne и Matplotlib (см. листинг 1.6).

Листинг 1.6 – Рисование и вывод гистограмм.

| bins = [5, 10, 15, 20, 30]  fig, axs = plt.subplots(  len(bins),  len(df.columns[:-1]),  figsize=(20, 5\*len(bins)))  for k in range(len(bins)):  for i, column in enumerate(df.columns[:-1]):  sns.histplot(  df,  x=column,  bins=bins[k],  ax=axs[k][i]) |
| --- |

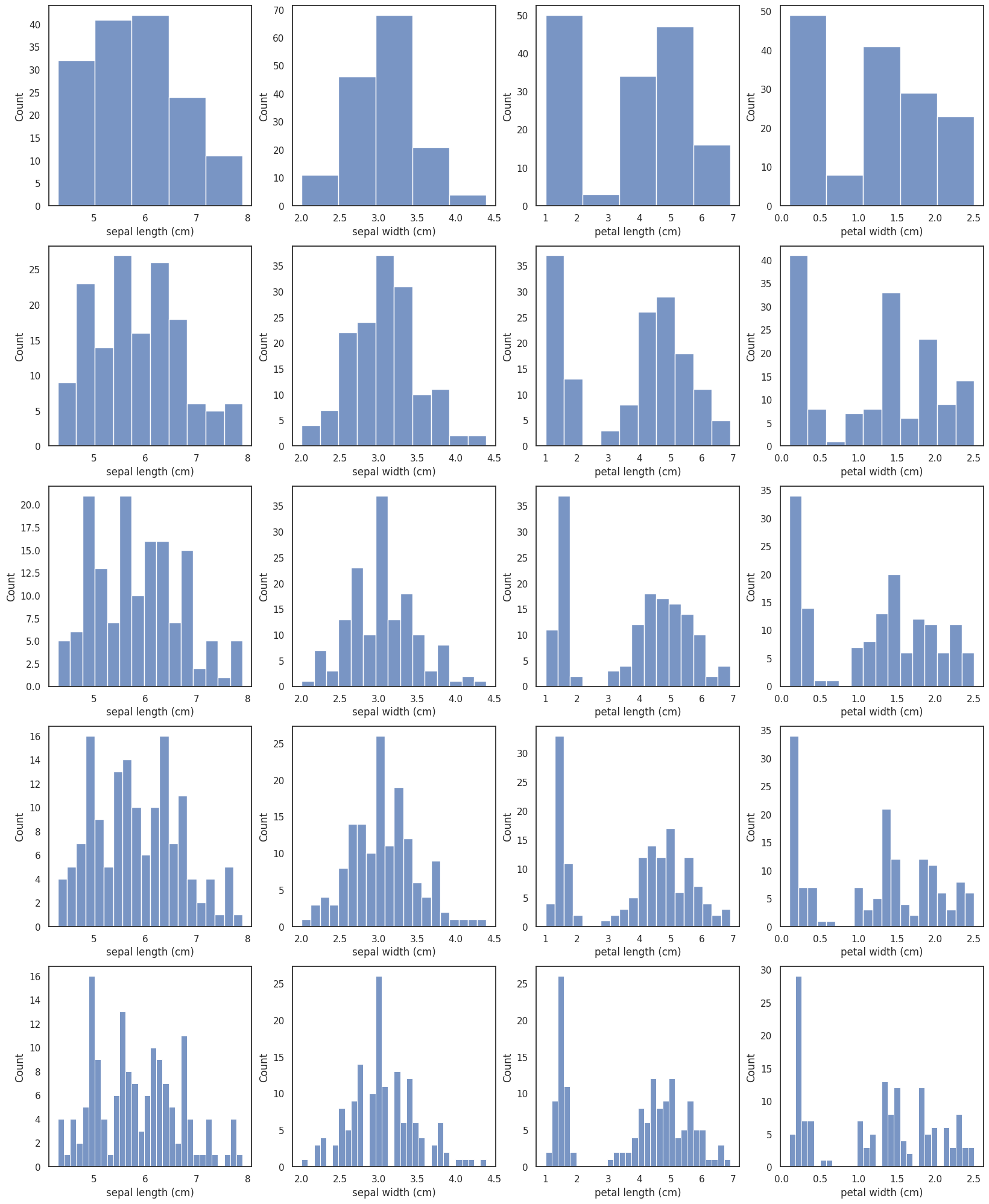


Рисунок 1.2 – Графики гистограмм (построчно, сверху вниз: 5, 10, 15, 20, 30 столбцов)

Из полученных графиков гистограмм (см. рисунок 1.2) можно увидеть, что вариант с 20 столбцами лучше всего описывает распределения всех признаков, т.к. количество пустых промежутков оптимально мало, а пики распределений наглядно видно. Следовательно, далее для рисования гистограмм будет использоваться 20 столбцов.

* + 1. Распределим на гистограмме классы по цветам и рассмотрим режимы, когда они накладываются (см. рисунок 1.3) и пересекаются (см. рисунок 1.4)

Листинг 1.7 – Рисование гистограмм с наложением.

| fig, axs = plt.subplots(1, len(df.columns[:-1]), figsize=(20, 5))  for i, column in enumerate(df.columns[:-1]):  sns.histplot(df, x=column, hue="target", bins=20, ax=axs[i],  multiple="stack") |
| --- |

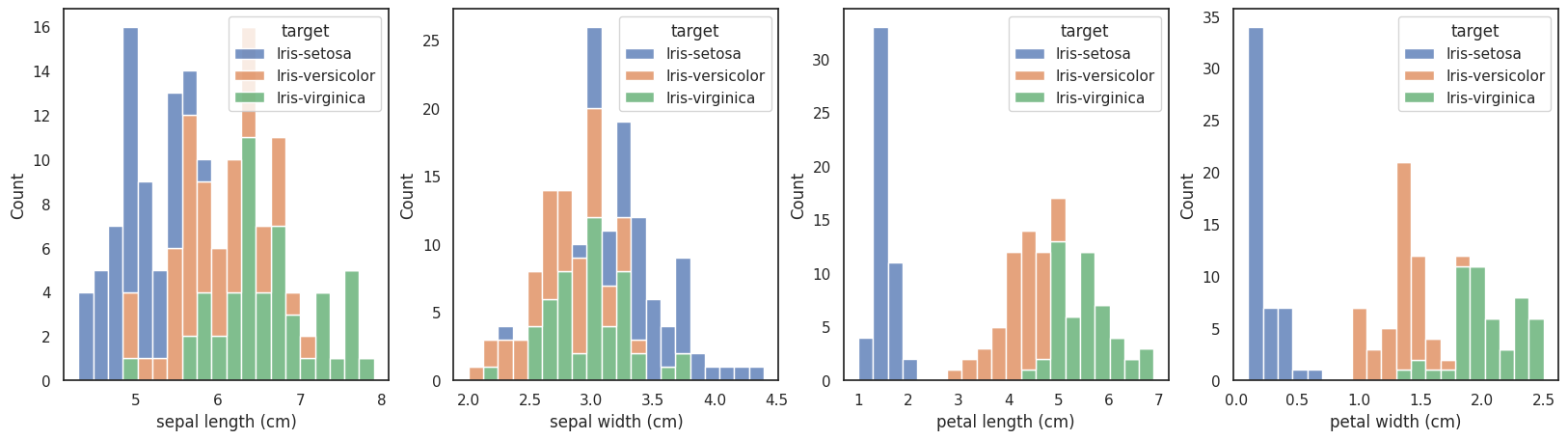


Рисунок 1.3 – Гистограммы с наложением.

Листинг 1.8 – Рисование гистограмм с пересечением.

| fig, axs = plt.subplots(1, len(df.columns[:-1]), figsize=(20, 5))  for i, column in enumerate(df.columns[:-1]):  sns.histplot(df, x=column, hue="target", bins=20, ax=axs[i]) |
| --- |

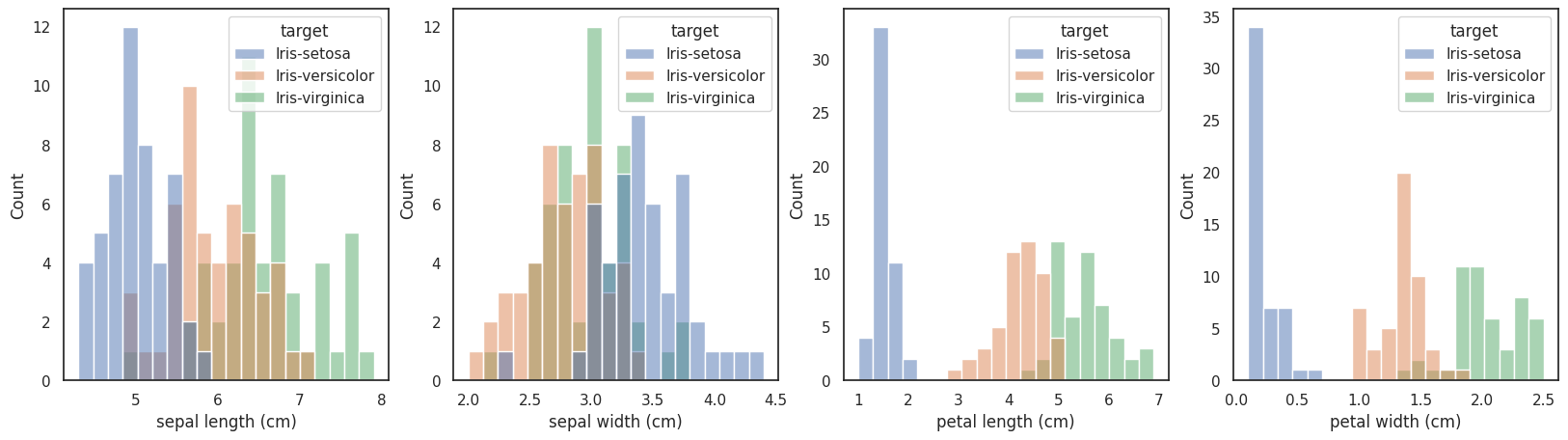


Рисунок 1.4 – Гистограммы с пересечением.

Для реализации наложения гистограмм (см. листинг 1.7 и 1.8) друг на друга в атрибут *multiple* внесено значение *stack*. Для пересечения достаточно использовать атрибут *hue*.

Из графиков пересечения гистограмм (см. рисунок 1.4), можно сделать вывод о схожести смешивания классов полученных графиков с графиками ядерной оценки плотности по каждому признаку.

* + 1. Построим гистограммы, чтобы вместо столбцов изображались ступеньки (см. рисунок 1.5).

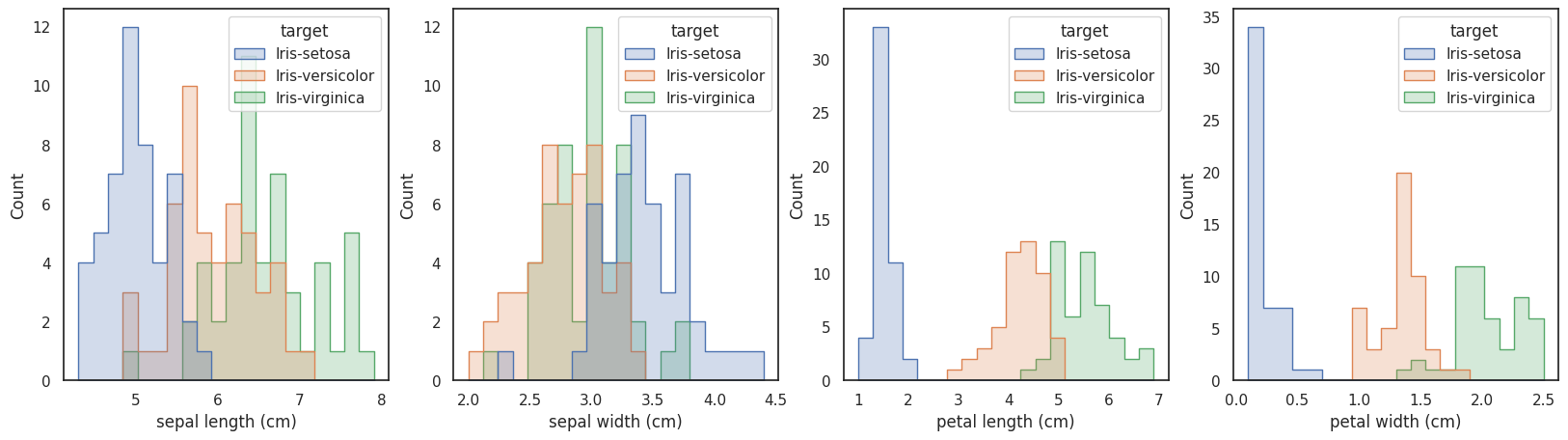


Рисунок 1.5 – Гистограммы в виде ступенек.

Листинг 1.9 – Рисование гистограмм в виде ступенек.

| fig, axs = plt.subplots(1, len(df.columns[:-1]), figsize=(20, 5))  for i, column in enumerate(df.columns[:-1]):  sns.histplot(df, x=column, hue="target", bins=20, ax=axs[i],  element="step") |
| --- |

Графики гистограмм приведены в ступенчатый вид с помощью передачи в *element* значения step (см. листинг 1.9). Воспользуемся графиками ступенчатых гистограмм для дальнейшего построения на их основе графиков оценки плотности.

* + 1. Добавим на гистограммы график ядерной оценки плотности (см. рисунок 1.6).

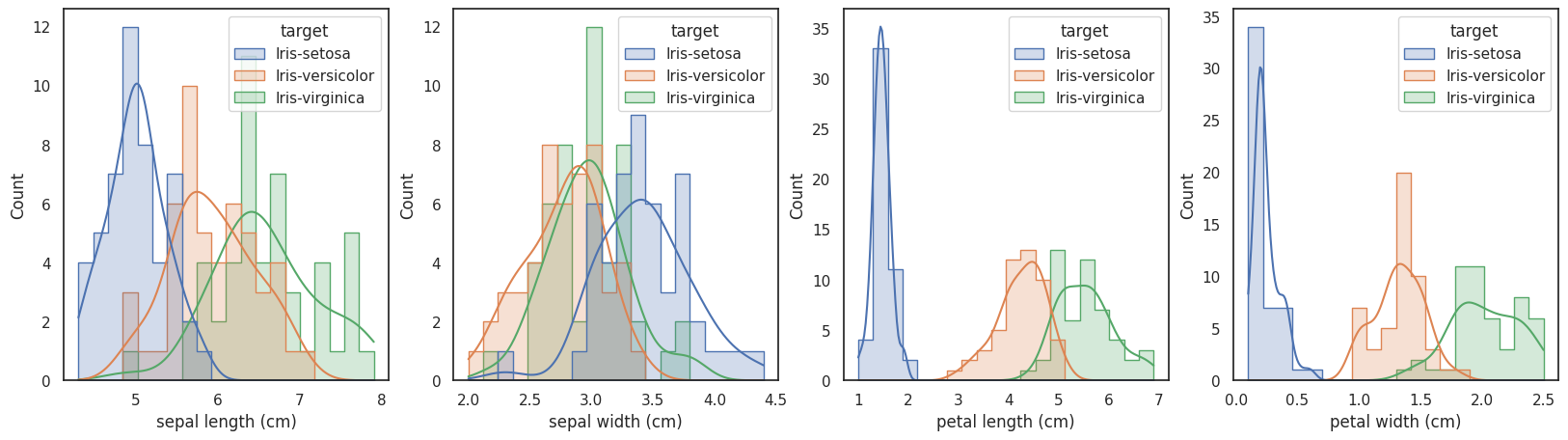


Рисунок 1.6 – Гистограммы с графиком ядерной оценки плотности.

Листинг 1.10 – Рисование гистограмм с графиком ядерной оценки плотности.

| fig, axs = plt.subplots(1, len(df.columns[:-1]), figsize=(20, 5))  for i, column in enumerate(df.columns[:-1]):  sns.histplot(df, x=column, hue="target", bins=20,  ax=axs[i], kde=True, element="step") |
| --- |

График ядерной оценки плотности был добавлен к гистограмме с помощью изменения атрибута *kde* на *True* (см. листинг 1.10). Из рисунка 1.6 видно, что построенные с учетом гистограмм графики совпадают с графиками ядерной оценки плотности из рисунка 1.1. Таким образом, можно сказать, что графики плотности дают более четкое понимание о распределении признаков, чем гистограммы, так как оценка плотности точнее показывает максимумы и колебания на графике.

1. Изучим набор данных iris.csv с использованием NumPy
   1. Загрузим данные из файла как массив NumPy. Для этого воспользуемся методом *genfromtxt* (см. листинг 2.1)

Листинг 2.1 – Загрузка датасета в виде массива NumPy.

| df\_arr = np.genfromtxt("iris.csv", delimiter=",", dtype=float, filling\_values=0, names=True,  usecols=(1, 2, 3, 4, 5)) |
| --- |

Для корректной загрузки следующим атрибутам были заданы значения:

* *delimeter = “,”* – данные разделяются по запятой.
* *dtype = float* – определение типа данных для загружаемых вещественных чисел.
* *filling\_values = 0* – все пропущенные значения заполняются нулями.
* *names = True* – для считывания имен признаков.
* *usecols = (1, 2, 3, 4, 5)* – какие стобцы загружаются из файла.
  1. Выведем первые 10 наблюдений набора загруженных данных с помощью среза массива (см. листинг 2.2). Результат вывода запишем в таблицу 2.1.

Листинг 2.2 – Вывод первых 10 строк.

| df\_arr[:10] |
| --- |

Таблица 2.1 – Результат вывода данных.

| sepal\_length\_cm | sepal\_width\_cm | petal\_length\_cm | petal\_width\_cm | target |
| --- | --- | --- | --- | --- |
| 5.1 | 3.5 | 1.4 | 0.2 | 0.0 |
| 4.9 | 3.0 | 1.4 | 0.2 | 0.0 |
| 4.7 | 3.2 | 1.3 | 0.2 | 0.0 |
| 4.6 | 3.1 | 1.5 | 0.2 | 0.0 |
| 5.0 | 3.6 | 1.4 | 0.2 | 0.0 |
| 5.4 | 3.9 | 1.7 | 0.4 | 0.0 |
| 4.6 | 3.4 | 1.4 | 0.3 | 0.0 |
| 5.0 | 3.4 | 1.5 | 0.2 | 0.0 |
| 4.4 | 2.9 | 1.4 | 0.2 | 0.0 |
| 4.9 | 3.1 | 1.5 | 0.1 | 0.0 |

По таблице 2.1 можно подтвердить, что данные были загружены корректно.

* 1. Рассчитаем характеристики, полученные методом *describe* в пункте. 1.3 с использованием методов NumPy (см. листинг 2.3). Полученный результат вывода запишем в таблицу 2.2.

Листинг 2.3 – Вывод характеристики массива NumPy.

| numeric\_data = df\_arr.view((float, len(df\_arr.dtype.names)))  print("\t"+ "\t".join(df\_arr.dtype.names))  print("count\t" + "".join([f"{np.count\_nonzero(~np.isnan(numeric\_data[:, i:i+1])):.6f}\t"  for i in range(len(df\_arr.dtype.names))]))  print("mean\t" + "".join([f"{np.mean(numeric\_data[:, i:i+1]):.6f}\t"  for i in range(len(df\_arr.dtype.names))]))  print("std\t" + "".join([f"{np.std(numeric\_data[:, i:i+1]):.6f}\t"  for i in range(len(df\_arr.dtype.names))]))  print("min\t" + "".join([f"{np.amin(numeric\_data[:, i:i+1]):.6f}\t"  for i in range(len(df\_arr.dtype.names))]))  print("25%\t" + "".join([f"{np.percentile(numeric\_data[:, i:i+1], 25):.6f}\t"  for i in range(len(df\_arr.dtype.names))]))  print("50%\t" + "".join([f"{np.percentile(numeric\_data[:, i:i+1], 50):.6f}\t"  for i in range(len(df\_arr.dtype.names))]))  print("75%\t" + "".join([f"{np.percentile(numeric\_data[:, i:i+1], 75):.6f}\t"  for i in range(len(df\_arr.dtype.names))]))  print("max\t" + "".join([f"{np.amax(numeric\_data[:, i:i+1]):.6f}\t"  for i in range(len(df\_arr.dtype.names))])) |
| --- |

Таблица 2.2 – Результат вывода характеристики.

|  | sepal\_length\_cm | sepal\_width\_cm | petal\_length\_cm | petal\_width\_cm | target |
| --- | --- | --- | --- | --- | --- |
| count | 150.000000 | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.057333 | 3.758000 | 1.199333 | 1.000000 |
| std | 0.828066 | 0.435866 | 1.765298 | 0.762238 | 0.819232 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 | 0.000000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 | 0.000000 |
| 50% | 5.800000 | 3.000000 | 4.350000 | 1.300000 | 1.000000 |
| 75% | 6.400000 | 3.300000 | 5.100000 | 1.800000 | 2.000000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 | 2.000000 |

Полученные данные для *count, mean, 25%, 50%, 75%, max* полностью совпали с выводом *describe* (см. таблицу 1.2), значение для *std* имеют небольшие отклонения.

1. Изучим набор данных варианта 1.
   1. Загрузим данные из файла как Pandas DataFrame (*read\_csv*) (см. листинг 3.1)

Листинг 3.1 – Загрузка датасета

| df\_var1 = pd.read\_csv("iris.csv") |
| --- |

* 1. Вызовем у датафрейма метод *head* и проверим корректность загруженных данных (см. листинг 3.2/ Вывод метода см. в таблице 3.1

Листинг 3.2 – Вызов *head*

| df\_var1.head() |
| --- |

Таблица 3.1 – Вывод *head*

|  | first | second | third | fourth | label |
| --- | --- | --- | --- | --- | --- |
| 0 | 0.342494 | -4.170293 | -0.427134 | 5.285126 | 0 |
| 1 | 2.506861 | 1.536588 | 3.311512 | 3.455109 | 1 |
| 2 | -0.723178 | -2.866860 | -1.778395 | 3.604657 | 0 |
| 3 | 2.413882 | 3.921118 | 2.923352 | 2.768869 | 1 |
| 4 | -0.377760 | -2.471150 | -2.796839 | 3.968686 | 0 |

* 1. Вызвав у датафрейма метод *describe*, получим характеристики (см. листинг 3.3). Результаты выполнения метода представлены в таблице 3.2.

Листинг 3.3 – Вызов *describe*

| df\_var1.describe() |
| --- |

Таблица 3.2 – вывод *describe*

|  | first | second | third | fourth | label |
| --- | --- | --- | --- | --- | --- |
| count | 200.000000 | 200.000000 | 200.000000 | 200.000000 | 200.000000 |
| mean | -0.079295 | 0.056456 | -0.078393 | 3.019419 | 0.500000 |
| std | 3.199506 | 3.026384 | 3.285926 | 1.055298 | 0.501255 |
| min | -6.200980 | -4.808300 | -6.728692 | 0.523930 | 0.000000 |
| 25% | -3.150403 | -2.655759 | -3.176466 | 2.350288 | 0.000000 |
| 50% | 0.708355 | -0.549150 | -0.120588 | 2.986238 | 0.500000 |
| 75% | 2.775412 | 2.894577 | 2.872863 | 3.763799 | 1.000000 |
| max | 5.443596 | 5.361941 | 6.107679 | 6.353969 | 1.000000 |

Была выведена информация для каждого признака в датафрейме (см. таблицу 3.2). По ней можно сказать, что датафрейм хранит по 200 значений для каждого признака. Также видно, что наблюдения для классов 1 и 0 распределены примерно поровну.

* 1. Видоизменим полученный датафрейм с помощью метода *replace* таким образом, чтобы метки классов были следующими: 0 - Zero, 1 - One и cохраним полученный датафрейм в отдельный файл new\_lab1\_var1.csv с помощью метода *to\_csv* (см. листинг 1.4). Проверим получившийся датафрейм с помощью *head* (см. таблицу 1.3).

Листинг 3.4 – Изменение и сохранение датафрейма.

| df\_var1.replace(  {"label":  {0: "Zero",  1: "One"}},  inplace=True)  df\_var1.to\_csv("new\_lab1\_var1.csv", index=False) |
| --- |

Таблица 3.3 – Измененный датафрейм.

|  | first | second | third | fourth | label |
| --- | --- | --- | --- | --- | --- |
| 0 | 0.342494 | -4.170293 | -0.427134 | 5.285126 | Zero |
| 1 | 2.506861 | 1.536588 | 3.311512 | 3.455109 | One |
| 2 | -0.723178 | -2.866860 | -1.778395 | 3.604657 | Zero |
| 3 | 2.413882 | 3.921118 | 2.923352 | 2.768869 | One |
| 4 | -0.377760 | -2.471150 | -2.796839 | 3.968686 | Zero |

* 1. Визуально оценим набор данных, построив изображение, содержащее графики ядерной оценки плотности каждого признака, диаграмму рассеяния и двумерную ядерную оценку плотности для каждого признаков (см. рисунок 3.1). Используем для этого библиотеку Seaborn (см. листинг 3.5)

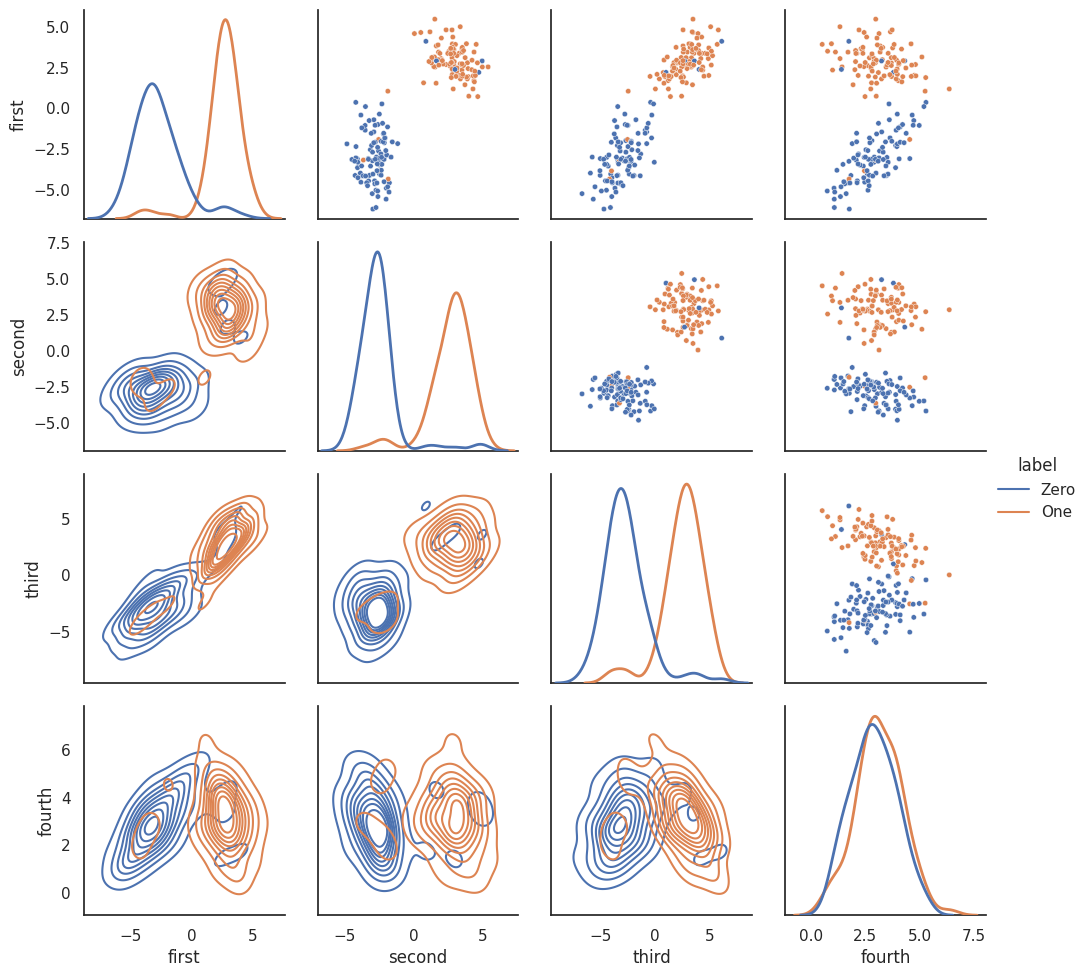


Рисунок 3.1 – Графики lab1\_var1.csv

Листинг 3.5 – Рисование и вывод графиков.

| sns.set\_theme(style="white")  sns.color\_palette("tab10")  g = sns.PairGrid(df\_var1, diag\_sharey=False, hue="label")  g.map\_upper(sns.scatterplot, s=15)  g.map\_lower(sns.kdeplot)  g.map\_diag(sns.kdeplot, lw=2)  g.add\_legend() |
| --- |

Из графиков (см. рисунок 3.1) можно заметить, что по всем признакам классы «Zero» и «One» имеют выбросы точек из скопления одного класса в другой. Графики ядерной оценки плотности для признака «fourth» для двух классов являются практически эквивалентными.

* 1. На одном изображении построим гистограммы распределения для каждого признака.
     1. Сделаем на каждой гистограмме разделение по классам разными цветами, отображение ступенек вместо столбцов (см. рисунок 3.2) и для сравнения добавим график ядерной оценки плотности к гистограммам (см. рисунок 3.3). Код для реализации рисования графиков можно увидеть в листинге 3.6 и 3.7.

Листинг 3.6 – Рисование и вывод ступенчатых гистограмм.

| fig, axs = plt.subplots(1, len(df\_var1.columns[:-1]), figsize=(20, 5))  for i, column in enumerate(df\_var1.columns[:-1]):  sns.histplot(df\_var1, x=column, hue="label", bins=20, ax=axs[i], element="step") |
| --- |

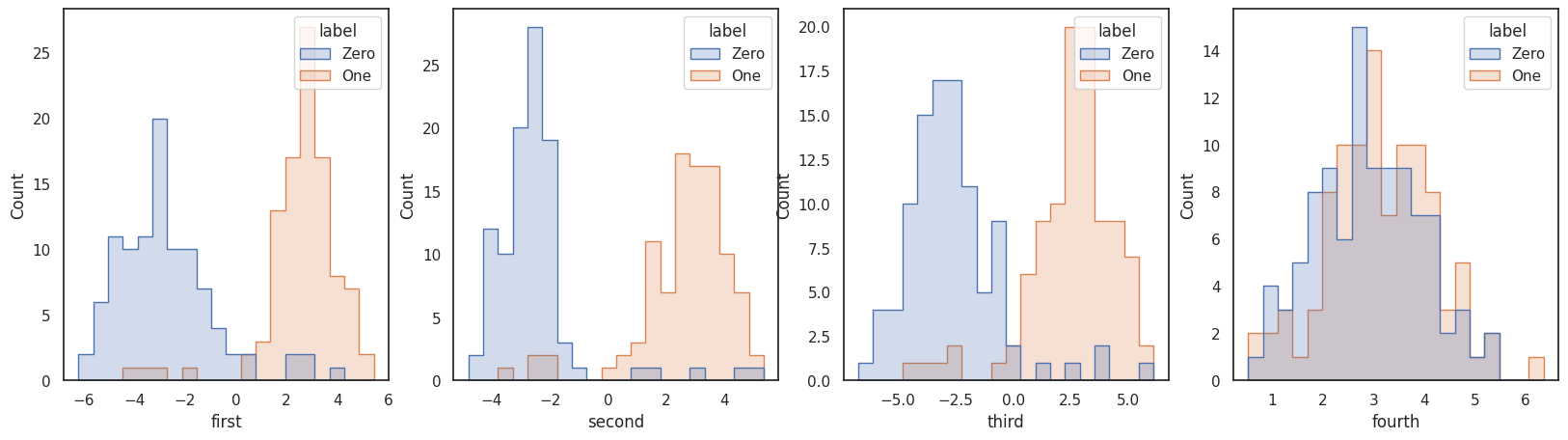


Рисунок 3.2 – Графики ступенчатых гистограмм.

Листинг 3.7 – Рисование гистограмм с графиком ядерной оценки плотности.

| fig, axs = plt.subplots(1, len(df\_var1.columns[:-1]), figsize=(20, 5))  for i, column in enumerate(df\_var1.columns[:-1]):  sns.histplot(df\_var1, x=column, hue="label", bins=20, ax=axs[i], kde=True, element="step") |
| --- |

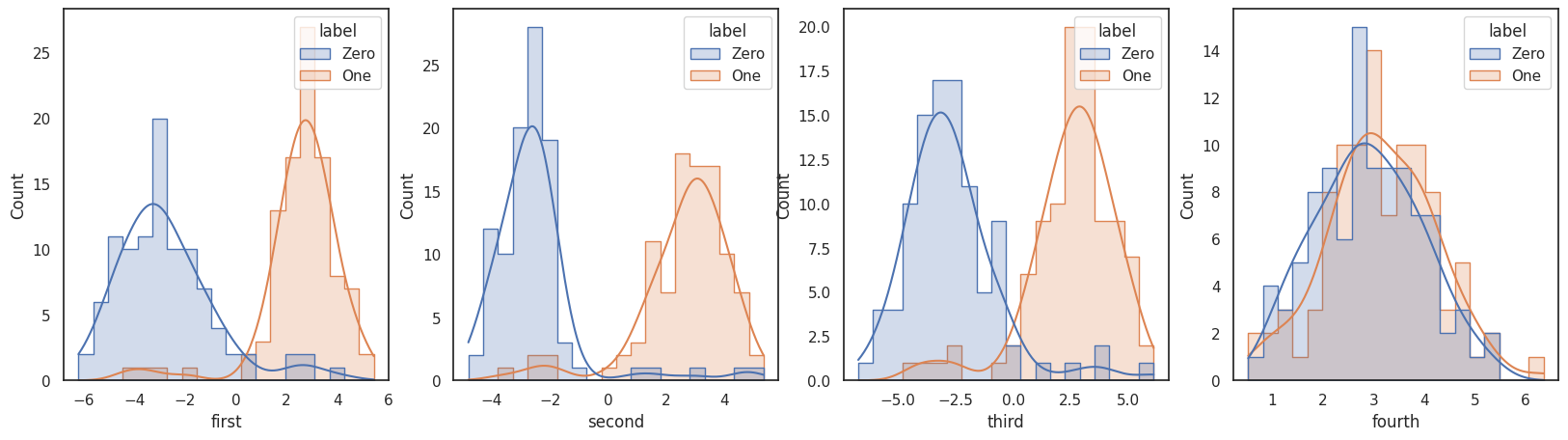


Рисунок 3.3 – Гистограммы с графиком ядерной оценки плотности.

Из рисунка 3.3 видно, что построенные графики плотности с учётом гистограмм совпадают с графиками ядерной оценки плотности из рисунка 3.1. Таким образом, можно сделать вывод, что признак «fourth» не годится для классификации данных с помощью «Zero» и «One» из-за схожести двух графиков плотности.

1. Преобразуем данные полученные в new\_iris.csv (cм. листинг 1.4)
   1. Закодируем классы используя *LabelEncoder* и *OneHotEncoder* (см. листинг 4.1 и 4.2). Результат внесем в таблицу 4.1 и 4.2.

Листинг 4.1 – Кодирование *LabelEncoder.*

| new\_df = pd.read\_csv("new\_iris.csv")  le = LabelEncoder()  new\_df["target"] = le.fit\_transform(new\_df["target"])  new\_df |
| --- |

Таблица 4.1 – Результат кодирования *LabelEncoder.*

| Iris-setosa | 0 |
| --- | --- |
| Iris-versicolor | 1 |
| Iris-virginica | 2 |

Листинг 4.2 – Кодирование *OneHotEncoder.*

| new\_df = pd.read\_csv("new\_iris.csv")  ohe = OneHotEncoder()  ohe\_df = pd.DataFrame(ohe.fit\_transform(new\_df[["target"]]).toarray())  new\_df.drop("target", axis=1, inplace=True)  new\_df = pd.concat([new\_df, ohe\_df], axis=1)  new\_df |
| --- |

Таблица 4.2 – Результат кодирования *OneHotEncoder.*

| Iris-setosa | 1 | 0 | 0 |
| --- | --- | --- | --- |
| Iris-versicolor | 0 | 1 | 0 |
| Iris-virginica | 0 | 0 | 1 |

Таким образом, *LabelEncoder* кодирует каждую метку класса числовым значением по порядку, из-за чего не всегда ясна связь данных меток классов. *OneHotEncoder*, кодирует каждую метку класса вектором, состоящим из 0 и 1, где каждый столбец говорит о принадлежности признака к классу (да - 1, нет - 0), что дает возможности избежать проблем связи в случае *LabelEncoder*.

* 1. Преобразуем датафрейм new\_iris.csv из пункта 1.4 в массив NumPy (см. листинг 4.3)

Листинг 4.3 – Загрузка датасета в виде массива NumPy.

| changed\_df = pd.read\_csv("changed\_iris.csv")  attributes = ["sepal length (cm)", "sepal width (cm)", "petal length (cm)", "petal width (cm)"]  data = new\_df[attributes].to\_numpy() |
| --- |

Для преобразования только нужных признаков был создан массив *attributes* и использован метод *to\_numpy*.

* 1. Для массива NumPy из п. 4.2 применим *StandardScaler, MinMaxScaler, MaxAbsScaler* и *RobustScaler* (см. листинг 4.4, 4.5, 4.6, 4.7, 4.8). Для каждого из результатов построим гистограммы по каждому признаку без разделения по классам (см. рисунки 4.1, 4.2, 4.3, 4.4, 4.5).

Листинг 4.4 – Рисование гистограмм без нормирования.

| fig, axs = plt.subplots(1, data.shape[1], figsize=(20, 5))  for i in range(data.shape[1]):  sns.histplot(data[:, i:i+1], bins=20, ax=axs[i], legend=False).set(title=attributes[i]) |
| --- |

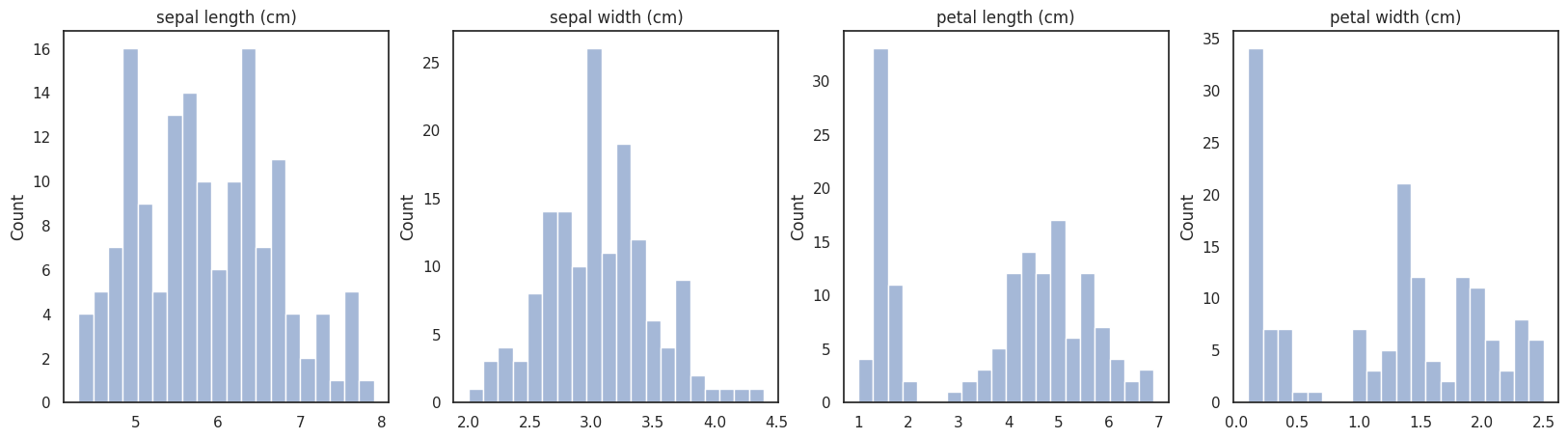


Рисунок 4.1 – Гистограммы без нормирования.

Листинг 4.5 – Рисование гистограмм с *StandartScaler*.

| scaler = StandardScaler()  scale\_data = scaler.fit\_transform(data)  fig, axs = plt.subplots(1, scale\_data.shape[1], figsize=(20, 5))  for i in range(scale\_data.shape[1]):  sns.histplot(scale\_data[:, i:i+1], bins=20,ax=axs[i], legend=False).set(title=attributes[i]) |
| --- |

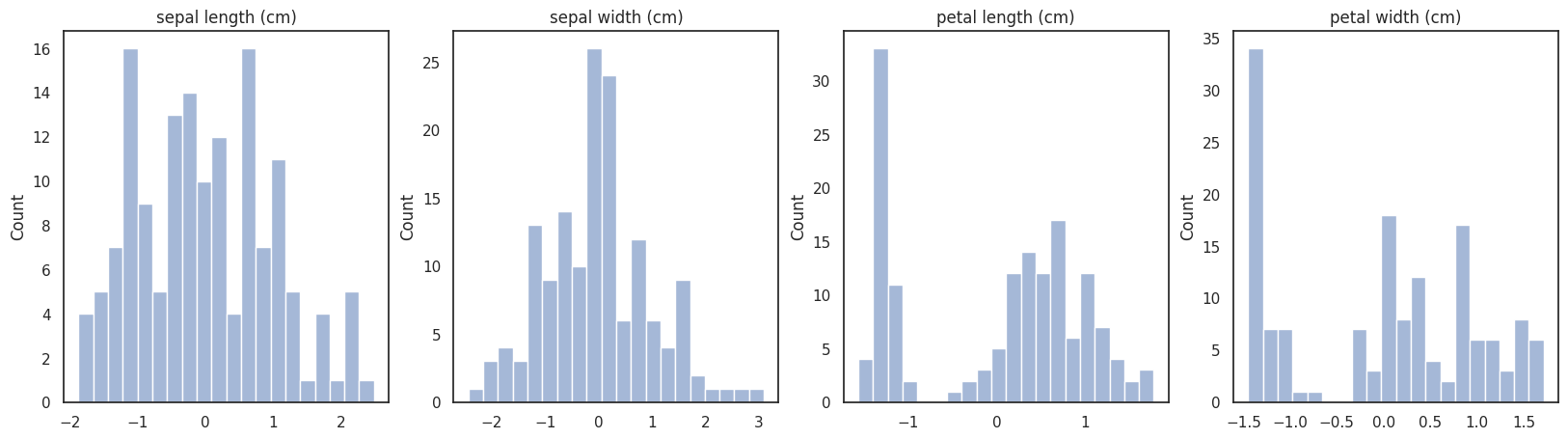


Рисунок 4.2 – Гистограммы с *StandartScaler.*

Листинг 4.6 – Рисование гистограмм с *MinMaxScaler*.

| scaler = MinMaxScaler()  scale\_data = scaler.fit\_transform(data)  fig, axs = plt.subplots(1, scale\_data.shape[1], figsize=(20, 5))  for i in range(scale\_data.shape[1]):  sns.histplot(scale\_data[:, i:i+1], bins=20, ax=axs[i], legend=False).set(title=attributes[i]) |
| --- |

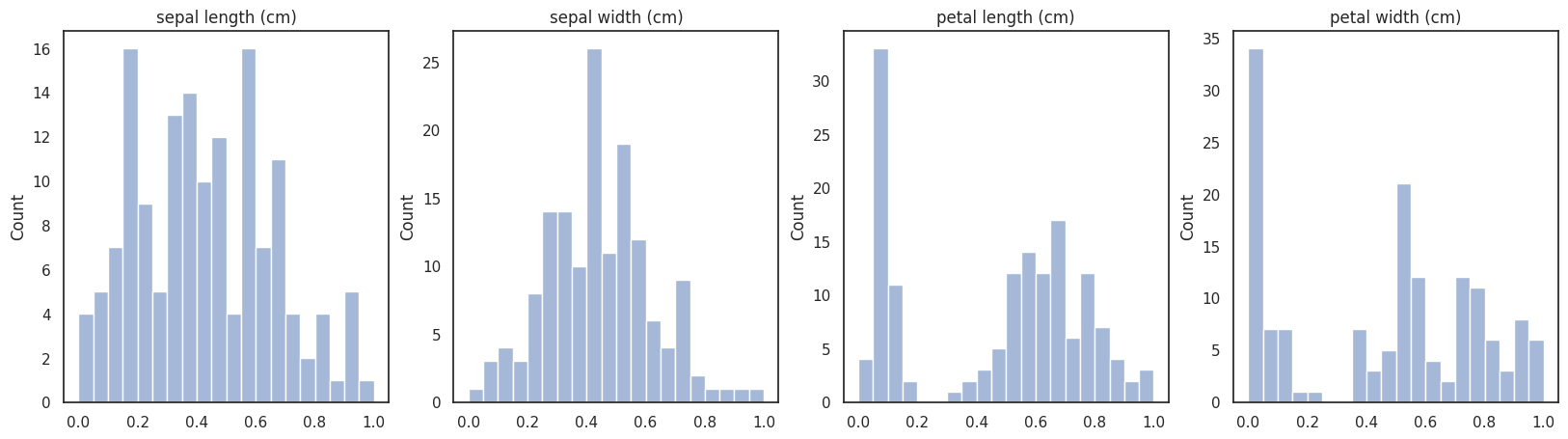


Рисунок 4.2 – Гистограммы с *MinMaxScaler.*

Листинг 4.7 – Рисование гистограмм с *MaxAbsScaler*.

| scaler = MaxAbsScaler()  scale\_data = scaler.fit\_transform(data)  fig, axs = plt.subplots(1, scale\_data.shape[1], figsize=(20, 5))  for i in range(scale\_data.shape[1]):  sns.histplot(scale\_data[:, i:i+1], bins=20, ax=axs[i], legend=False).set(title=attributes[i]) |
| --- |

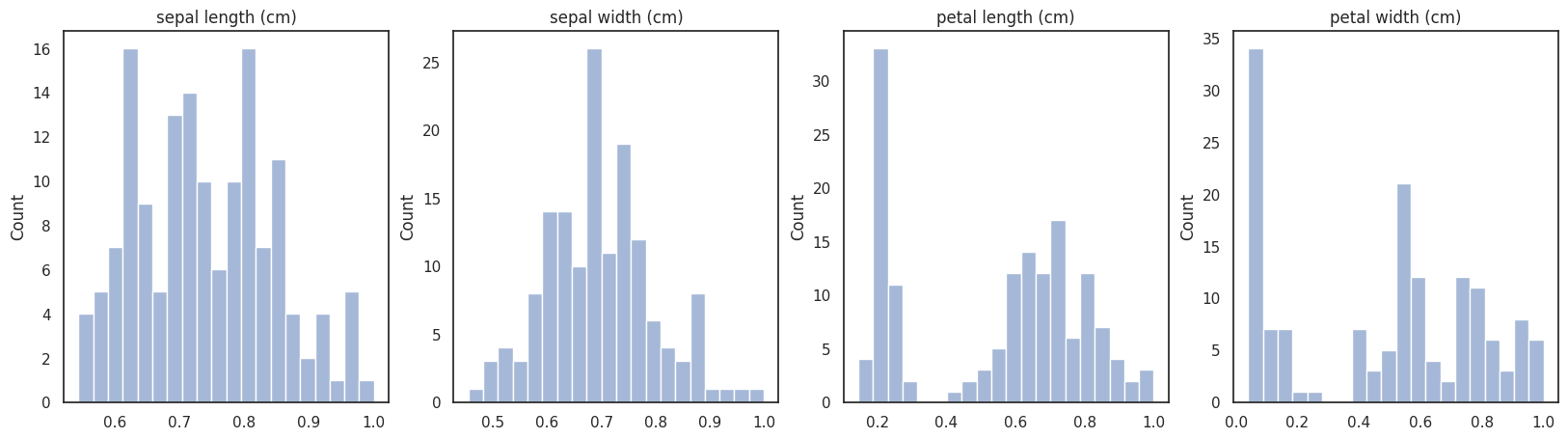
**

Рисунок 4.2 – Гистограммы с *MaxAbsScaler.*

Листинг 4.8 – Рисование гистограмм с *RobustScaler*.

| scaler = RobustScaler()  scale\_data = scaler.fit\_transform(data)  fig, axs = plt.subplots(1, scale\_data.shape[1], figsize=(20, 5))  for i in range(scale\_data.shape[1]):  sns.histplot(scale\_data[:, i:i+1], bins=20, ax=axs[i], legend=False).set(title=attributes[i]) |
| --- |

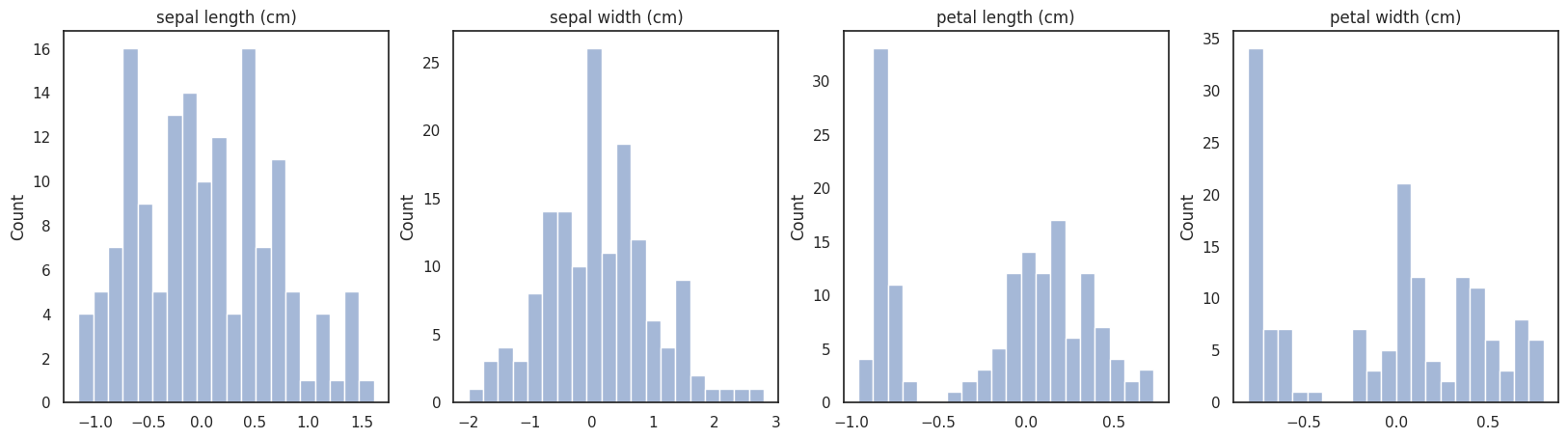
**

Рисунок 4.2 – Гистограммы с *RobustScaler.*

При нормировании в листинге 4.5 использовался StandardScaler. Данная нормировка преобразует данные так, чтобы среднее значением стало равным или приближенным нулю, а стандартное отклонение равным 1. Такое преобразование можно охарактеризовать формулой 4.1.

(4.1)

Где E(X) – мат. ожидание, а σX – стандартное отклонение.

При нормировании в листинге 4.6 использовался MinMaxScaler. Данная нормировка преобразует данные так, чтобы значения находились в определённом диапазоне (обычно от 0 до 1). Такое преобразование можно охарактеризовать формулой 4.2.

(4.2)

Где min(X) – минимальное, а max(X) – максимальное значения.

При нормировании в листинге 4.7 использовался MaxAbsScaler. Данная нормировка преобразует данные так, чтобы значения находились в диапозоне от -1 до 1. Такое преобразование можно охарактеризовать формулой 4.3.

(4.3)

Где max(|X|) – максимальное значение по модулю.

При нормировании в листинге 4.8 использовался RobustScaler. Данная нормировка преобразует данные так, чтобы происходило центрирование и масштабирование на основе медианы и интерквартильного размаха. Такое преобразование можно охарактеризовать формулой 4.4.

(4.4)

Q1, Q2, Q3 – 1, 2 и 3 квартили.

Из рисунков 4.1, 4.2, 4.3, 4.4, 4.5 можно сделать вывод, что внешний вид распределения не изменился, но изменилась ось X в зависимости от примененной нормировки.

* 1. Согласно варианту, реализуем MinMaxScaler с использованием NumPy. Код приведен в листинге 4.9.

Листинг 4.9 – Реализация нормирования *MinMaxScaler*.

| def CustomMinMaxScaler(dataframe, feature\_min, feature\_max):  dataframe\_scaled = np.zeros(dataframe.shape)  for i in range(dataframe.shape[1]):  current\_column = dataframe[:, i]  dataframe\_std = (dataframe - min(current\_column)) / (max(current\_column) - min(current\_column))  column\_scaled = dataframe\_std[:, i] \* (feature\_max - feature\_min) + feature\_min  dataframe\_scaled[:, i] = column\_scaled  return dataframe\_scaled |
| --- |

Функция CustomMinMaxScaler (см. листинг 4.9) принимает на вход массив данных *dataframe*, минимальное и максимальное значения *feature\_min* и *feature\_max*, используемые для масштабирования данных. Внутри функции создаётся *dataframe\_scaled*, который изначально заполняется нулями. В него вносятся нормированные данные. В цикле по всем столбцам с признаками функция вычисляет значение для каждого элемента в столбце путем вычитания минимального значения столбца из *dataframe* и деления на разницу между максимальным и минимальным значениями столбца. Полученный столбец умножается на разницу между *feature\_max* и *feature\_min*, а затем добавляется *feature\_min* для восстановления оригинального масштаба данных. Затем вносим вычисленный столбец в *dataframe\_scaled* и возвращаем его. Для сравнения двух функций выпишем результат первых пяти строк в таблицы (см. таблица 4.3 и 4.4).

Таблица 4.3 – Результат *MinMaxScaler*

| 0.56193323 | 0.06273273 | 0.49091428 | 0.81666626 |
| --- | --- | --- | --- |
| 0.74780234 | 0.62386802 | 0.78216842 | 0.50277178 |
| 0.47041666 | 0.19089415 | 0.38564611 | 0.52842311 |
| 0.73981761 | 0.85832953 | 0.75192933 | 0.3850641 |
| 0.50008003 | 0.2298028 | 0.30630564 | 0.59086327 |

Таблица 4.4 – Результат *CustomMinMaxScaler*

| 0.56193323 | 0.06273273 | 0.49091428 | 0.81666626 |
| --- | --- | --- | --- |
| 0.74780234 | 0.62386802 | 0.78216842 | 0.50277178 |
| 0.47041666 | 0.19089415 | 0.38564611 | 0.52842311 |
| 0.73981761 | 0.85832953 | 0.75192933 | 0.3850641 |
| 0.50008003 | 0.2298028 | 0.30630564 | 0.59086327 |

Данные полученные с помощью *CustomMinMaxScaler* и *MinMaxScaler* совпали, значит реализованная функция правильно нормирует датафрейм. Рассчитаем минимальное, максимальное, среднее значение и дисперсию, после преобразования (см. листинг 4.10). Рассчитанные результаты можно увидеть в таблице 4.5.

Листинг 4.10 – Вычисление характеристик.

| print("min\t" + "".join([f"{np.amin(scale\_data\_my[:, i:i+1]):.6f}\t" for i in range(len(scale\_data\_my[0]))]))  print("max\t" + "".join([f"{np.amax(scale\_data\_my[:, i:i+1]):.6f}\t" for i in range(len(scale\_data\_my[0]))]))  print("mean\t" + "".join([f"{np.mean(scale\_data\_my[:, i:i+1]):.6f}\t" for i in range(len(scale\_data\_my[0]))]))  print("var\t" + "".join([f"{np.var(scale\_data\_my[:, i:i+1]):.6f}\t" for i in range(len(scale\_data\_my[0]))])) |
| --- |

Таблица 4.5 – Вычисленные характеристики.

| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| --- | --- | --- | --- | --- |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| mean | 0.525711 | 0.478332 | 0.518082 | 0.428040 |
| var | 0.075118 | 0.088107 | 0.065201 | 0.032601 |

По данным из таблицы 4.5, можно подтвердить, что был применена нормировка MinMaxScaler со значениями диапазона 0 и 1, так как минимальное значение равно 0, а максимальное – 1 у всех признаков.

* 1. Для датафрейма из пункта 1.4 получим новый, который содержит только классы «Iris-versicolor» и «Iris-virginica», признаки «sepal length (cm)» и «petal length (cm)», и наблюдения, для которых значения признака «sepal width (cm)» лежат между квантилями 25% и 75% (см. листинг 4.11).

Листинг 4.11 – Получение нового датафрейма.

| new\_df = pd.read\_csv("new\_iris.csv")  new\_df = new\_df[(new\_df["target"] == "Iris-versicolor") | (new\_df["target"] == "Iris-virginica")]  new\_df = new\_df[["sepal length (cm)", "sepal width (cm)", "petal length (cm)"]]  new\_df = new\_df[(new\_df["sepal width (cm)"] >= new\_df["sepal width (cm)"].quantile(0.25)) & (new\_df["sepal width (cm)"] <= new\_df["sepal width (cm)"].quantile(0.75))] |
| --- |

Для получение нового датафрейма со всеми условиями, во 2-й строке листинга 4.11 были отфильтрованы данные, чтобы остались только строки, где значение в столбце «target» равно «Iris-versicolor» или «Iris-virginica». В строке 3 были оставлены только данные столбцов «sepal length (cm)», «sepal width (cm)», «petal length (cm)». В строке 4 было проведена фильтрация данных по значению «sepal width (cm)», оставляя только строки, где значение этого столбца находится между 25% и 75% перцентилем этого столбца.

1. Понижение размерности.
   1. Для набора данных iris.csv применим понижение размерности до 2-й, используя PCA и t-SNE из Sklearn (см. листинг 5.1 и 5.2). Для каждого из результатов построим диаграмму рассеяния с выделением разным цветом наблюдений разных классов (см. рисунок 5.1 и 5.2).

Листинг 5.1 – Понижение размерности PCA.

| df = pd.read\_csv("iris.csv")  pca = PCA(n\_components=2)  new\_df\_pca = pca.fit\_transform(df)  new\_df\_pca = pd.DataFrame(new\_df\_pca, columns=["first", "second"])  new\_df\_pca.insert(loc=new\_df\_pca.shape[1], column="target", value=df["target"])  sns.scatterplot(new\_df\_pca, x="first", y="second", hue="target", palette='tab10') |
| --- |

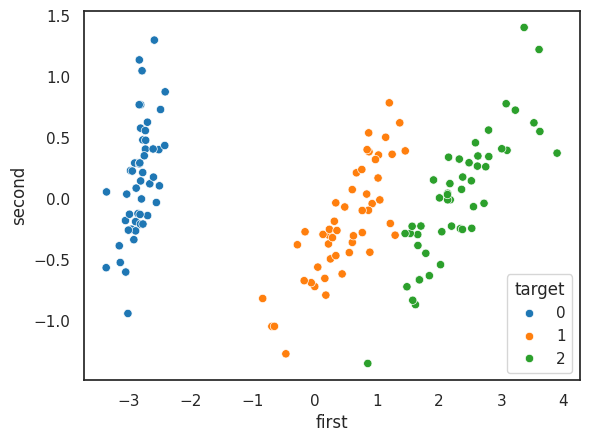


Рисунок 5.1 – Диаграмма рассеивания после PCA.

Листинг 5.2 – Понижение размерности t-SNE.

| df = pd.read\_csv("iris.csv")  pca = PCA(n\_components=2)  new\_df\_pca = pca.fit\_transform(df)  new\_df\_pca = pd.DataFrame(new\_df\_pca, columns=["first", "second"])  new\_df\_pca.insert(loc=new\_df\_pca.shape[1], column="target", value=df["target"])  sns.scatterplot(new\_df\_pca, x="first", y="second", hue="target", palette='tab10') |
| --- |

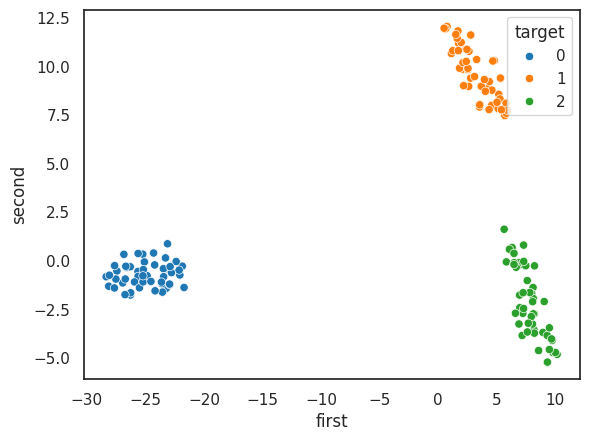


Рисунок 5.2 – Диаграмма рассеивания после t-SNE.

Алгоритмы PCA и t-SNE предназначены для выделения значимых признаков, отброса незначительных, устранение шума и визуализации многомерных пространств. Идея алгоритма PCA заключается в нахождении таких осей, на долю которых приходятся самые большие величины дисперсии в наборе данных. Такие оси называются главными компонентами. Данный алгоритм используют для устранения шумов и для линейной разделимости классов. Алгоритм t-SNE служит для понижения размерности, чтобы впоследствии визуализировать данные и использует одностороннее нелинейное преобразование.

Результаты работы двух алгоритмов различаются, потому что PCA стремится сохранить глобальную структуру данных, а t-SNE ориентирован на сохранение локальной структуры.

**Вывод.**

В ходе лабораторной работы были изучены способы анализа данных с применением библиотек Pandas и NumPy, с помощью которых считывались датафреймы и вычислялись характеристики их признаков. Для представления данных в виде графиков, диаграмм и гистограмм были использованы Seaborn и Matplotlib. По графикам был проведен анализ и сравнение. Было выяснено, что данные могут смешиваться и рассеиваться. Для преобразований датафреймов была использована библиотека Scikit-learn, с ее помощью данные были нормированы, закодированы, а их размерность была понижена. Также были изучены виды номировок: StandardScaler, MinMaxScaler, MaxAbsScaler и RobustScaler. Были изучены два вида кодировок – LabelEncoder и OneHotEncoder. Были изучены два метода понижения размерности исходного датафрейма: PCA и t-SNE.