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 3

Студент гр. 1303 Самохин К. А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Изучить способы анализа наборов данных при помощи ЯП Python и библиотек Pandas, Seabort, Matplotlib, NumPy, Scikit-learn. При помощи перечисленных библиотек провести анализ предоставленных датафреймов.

Задание.

1. Изучение набора данных iris.csv с использованием Pandas и Seaborn:
   - 1.1 Загрузить данные из файла как Pandas DataFrame
   - 1.2 Вызвав у датафрейма метод head, проверить корректность загруженных данных
   - 1.3 Вызвав у датафрейма метод describe, получить характеристики. Опишите полученный результат.
   - 1.4 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.
   - 1.5 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.
   - 1.6 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:
     - 1.6.1 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.
     - 1.6.2 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.

1.6.3 Постройте гистограммы, чтобы вместо столбцов изображались ступеньки.

1.6.4 Добавьте на гистограммы график ядерной оценки плотности.

2. Изучение набора данных iris.csv с использованием NumPy:
   - 2.1 Загрузите данные из файла как массив NumPy
   - 2.2 Выведите первые 10 наблюдений набора данных.
   - 2.3 Рассчитайте характеристики, полученные методом describe в п. 1.3 с использованием методов NumPy
3. Изучение набора данных вашего варианта:
   - 3.1 Оцените и опишите набор данных вашего варианта с использ ванием методов в п. 1
4. Преобразование данных:
   - 4.1 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?
   - 4.2 Для датафрейма из п. 1.4, получите все столбцы признаков (столбцы не содержащие метки классов). Преобразуйте полученные столбцы в массив NumPy.
   - 4.3 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?
   - 4.4 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.
   - 4.5 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:
   - 5.1 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. Изучение набора данных iris.csv с использованием Pandas и Seaborn.

1.1. Загрузим данные из файла iris.csv как Pandas DataFrame, используя метод *read_csv* (см. листинг 1.1).

Листинг 1.1 – Считывание датасета из файла iris.csv.

```
1  df = pd.read_csv('iris.csv')
```

1.2. Проверим корректность загруженных данных, вызвав у датафрейма метод *head* (см. листинг 1.2). Ко умолчанию данный метод выводит первые 5 строк. Вывод метода можно видеть в таблице 1.1.

Листинг 1.2 – Вызов метода *head.*

```
1  df.head()
```

Таблица 1.1 – Вывод метода *head.*

	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

1.3. Вызовем у датафрейма метод *describe* для получения характеристик (см. листинг 1.3). Вывод метода *describe* см. в таблице 1.2.

Листинг 1.3 – Вызов метода *describe.*

```
1  df.describe()
```

Таблица 1.2 – Вывод метода *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

Продолжение таблицы 1.2

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

Метод вывел информацию по каждому признаку из датафрейма (см. таблицу 1.2). В строке *count* представлено число непропущенных значений каждого столбца, в строке *mean* находится среднее значение по признаку, в строках *min* и *max* представлены соответственно минимальное и максимальное значения по столбцу, в строке 25% представлено значение первой квартили, в строке 50% выведено значение медианы, в строке 75% можно видеть значение верхней квартили.

1.4. Видоизменим датафрейм таким образом, чтобы метки классов были следующими: 0 - Iris-setosa, 1 - Iris-versicolor, 2 - Iris-virginica и запишем полученный датафрейм в отдельный файл формата csv. Замену значений столбца *target* будем проводить при помощи метода *replace* (см. листинг 1.4). Получившийся в результате замены датафрейм см. в таблице 1.3.

Листинг 1.4 – Замена значений в столбце датафрейма.

```
di = {0: 'Iris-setosa', 1: 'Iris-versicolor', 2: 'Iris-virginica'}

df['target'] = df['target'].replace(di)
df.to_csv("fixed_iris.csv", index=False)

df.head()
```

Таблица 1.3 – Вид изменённого датафрейма.

	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

Продолжение таблицы 1.3

3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

1.5. Визуально оценим набор данных, построив изображение, содержащее графики ядерной оценки плотности каждого признака, диаграмму рассеяния и двумерную ядерную оценку плотности для каждого признака (см. рисунок 1.1). Для рисования графиков будет использована библиотека Seaborn. (см. листинг 1.5).

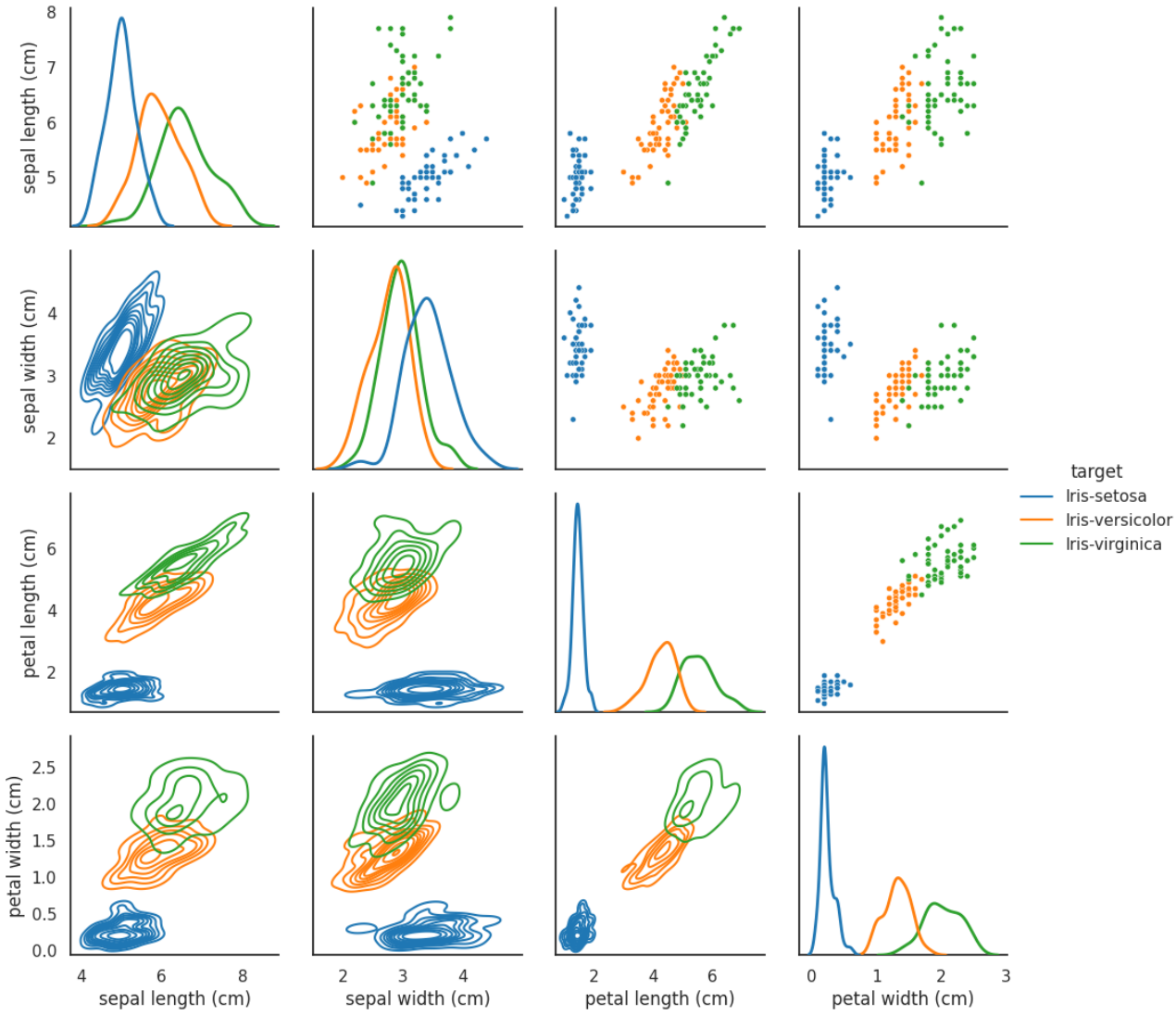


Рисунок 1.1 – Графики для iris.csv.

Листинг 1.5 – Рисование графиков.

```
1  g = sb.PairGrid(df, diag_sharey=False, hue='target',
palette='tab10')
```

Продолжение листинга 1.5

```
2  g.map_upper(sb.scatterplot, s=15)
3  g.map_lower(sb.kdeplot)
4  g.map_diag(sb.kdeplot, lw=2)
5  g.add_legend()
```

Для определения графиков каждого класса, все они на сетке графиков имеют свой цвет, согласно палитре «tab10». Для разделения по классам в атрибут *hue* передаётся строка с названием столбца классов. Метод *scatterplot* используется для рисования диаграмм рассеивания в правой верхней части сетки. Метод *kdeplot* используется для рисования графиков ядерной оценки плотности и двумерной ядерной оценки плотности по диагонали и в левом нижнем углу соответственно.

Из графиков (рисунок 1.1) несложно видеть, что классы «Iris-versicolor» и «Iris-verginica» смешиваются по всем признакам, однако класс «Iris-setosa» не смешивается ни с одним из них ни по одному признаку.

1.6. На одном изображении построим гистограммы распределения для каждого признака. Для построения нескольких графиков на одном изображении будет использован *subplot* из *matplotlib* (см. листинг 1.6). Полученные гистограммы см. на рисунке 1.2.

Листинг 1.6 – Построение гистограмм распределения без указания количества столбцов.

```
1  fig, axs = plt.subplots(1,4)
2  sb.histplot(df, x='sepal length (cm)', ax=axs[0])
3  sb.histplot(df, x='sepal width (cm)', ax=axs[1])
4  sb.histplot(df, x='petal length (cm)', ax=axs[2])
5  sb.histplot(df, x='petal width (cm)', ax=axs[3])
6  plt.show()
```

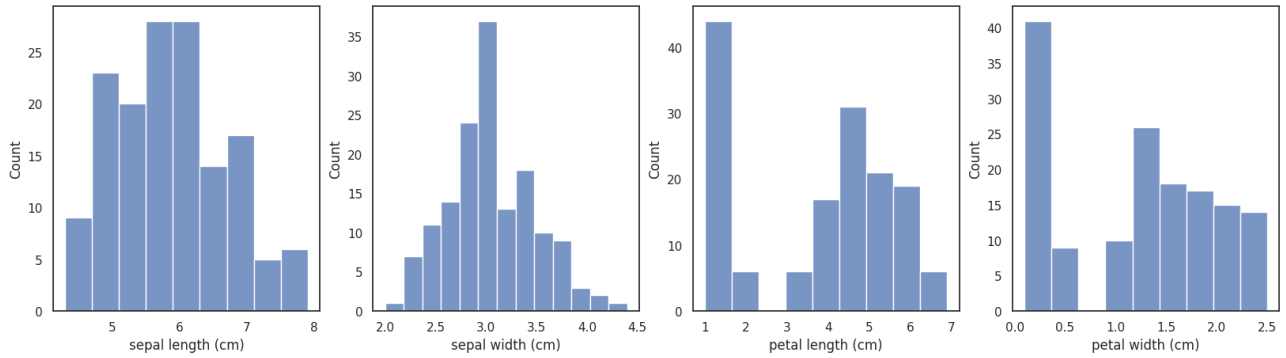
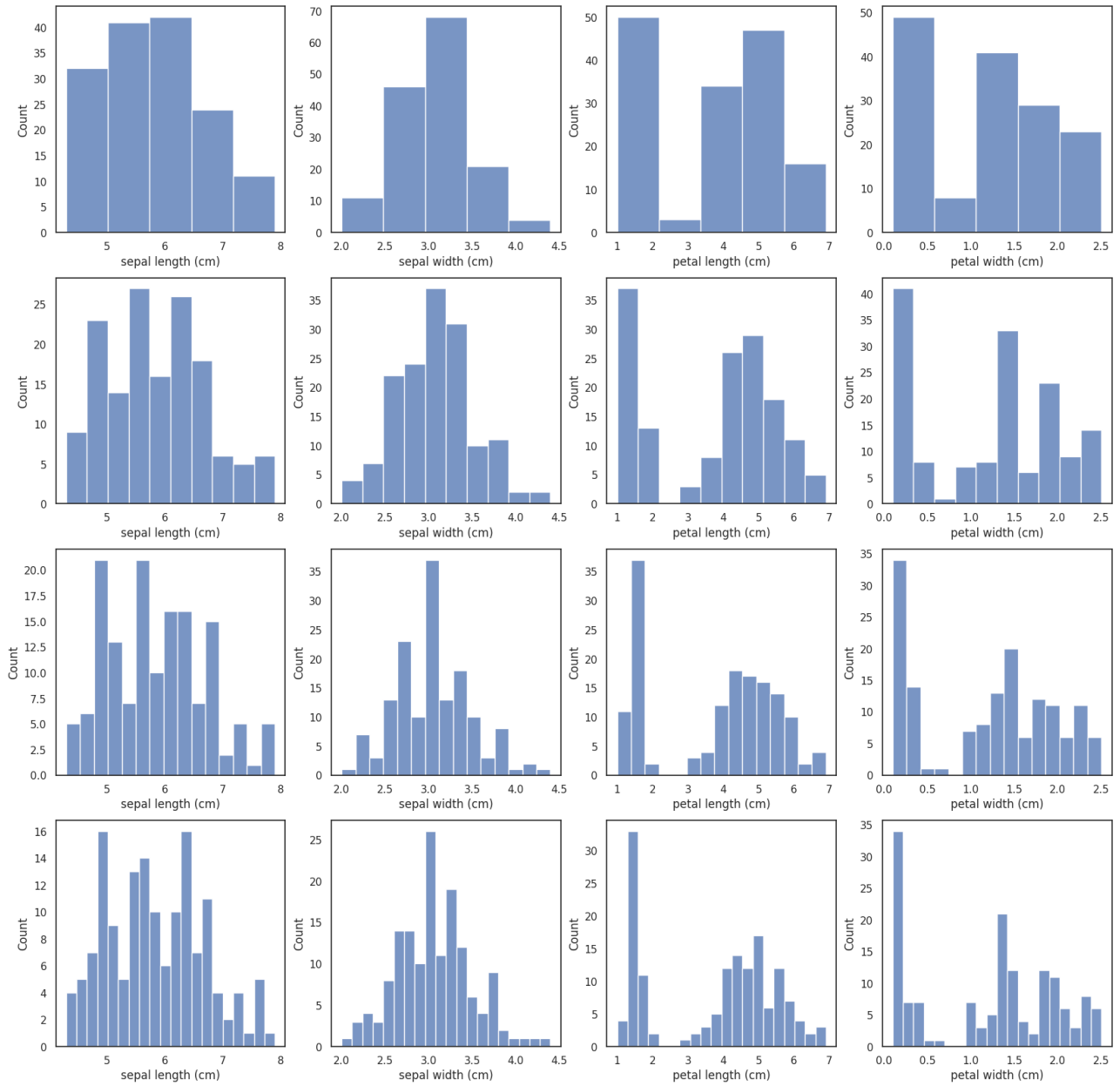


Рисунок 1.2 – Гистограммы распределения признаков.

1.6.1. Построим гистограммы для разного количества столбцов: 5,10,15,20,30 (см. рисунок 1.3). Для рисования графиков воспользуемся методом *histplot* и библиотеками *Seaborn* и *Matplotlob* (см. листинг 1.7).



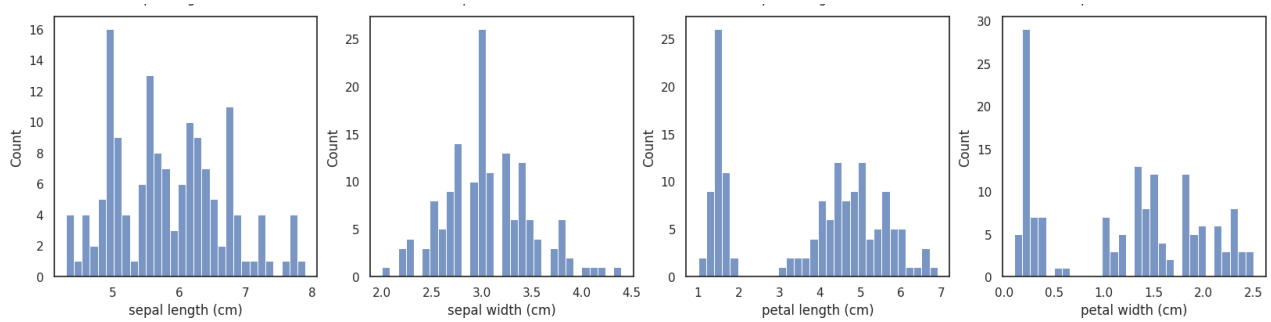


Рисунок 1.3 – Графики гистограмм на (5, 10, 15, 20, 30) столбцов.

Листинг 1.7 – Рисование графиков гистограмм.

```
bins = [5, 10, 15, 20, 30]

fig, axs = plt.subplots(5, 4, figsize=(20, 5*len(bins)))
for i in [5, 10, 15, 20, 30]:
    sb.histplot(df, bins=bins[i], x='sepal length (cm)', ax=axs[0])
    sb.histplot(df, bins=bins[i], x='sepal width (cm)', ax=axs[1])
    sb.histplot(df, bins=bins[i], x='petal length (cm)', ax=axs[2])
    sb.histplot(df, bins=bins[i], x='petal width (cm)', ax=axs[3])
    plt.show()
```

На полученных графиках гистограмм (см. рисунок 1.3) видно, что гистограммы с 20 столбцами достаточно точно показывают пики распределения всех четырёх параметров, при этом количество пустых промежутков оптимально мало. На основе вышесказанного приходим к решению, что в дальнейшем рисование гистограмм будет именно с 20 столбцами.

1.6.2. Сделаем на каждой гистограмме разделение по цветам, согласно классу. Рассмотрим режимы, когда гистограммы накладываются (см. рисунок 1.4) и пересекаются (см. рисунок 1.5).

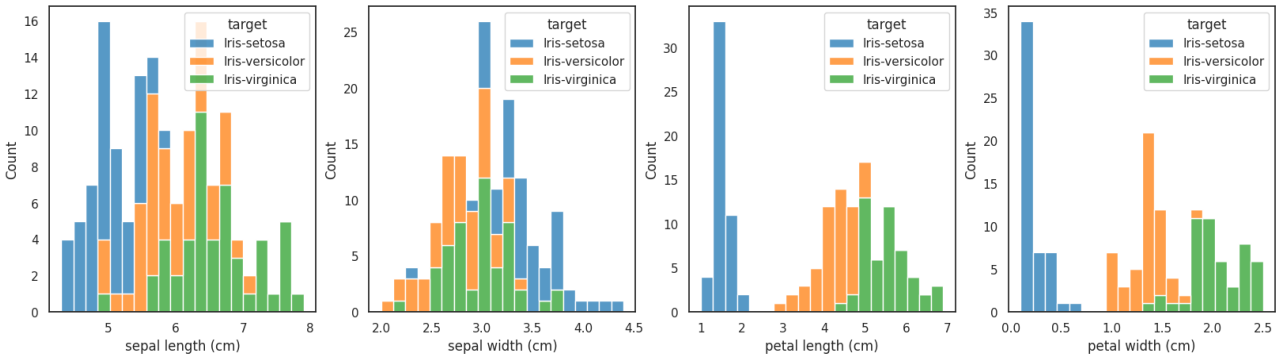


Рисунок 1.4 – Графики гистограмм с наложением.

Листинг 1.8 – Рисование графиков гистограмм с наложением.

```
1  fig, axs = plt.subplots(1,4)
2  sb.histplot(df, bins=20, hue='target', x='sepal length (cm)',
palette='tab10', multiple = 'stack', ax=axs[0])
3  sb.histplot(df, bins=20, hue='target', x='sepal width (cm)',
palette='tab10', multiple = 'stack', ax=axs[1])
4  sb.histplot(df, bins=20, hue='target', x='petal length (cm)',
palette='tab10', multiple = 'stack', ax=axs[2])
5  sb.histplot(df, bins=20, hue='target', x='petal width (cm)',
palette='tab10', multiple = 'stack', ax=axs[3])
6  plt.show()
```

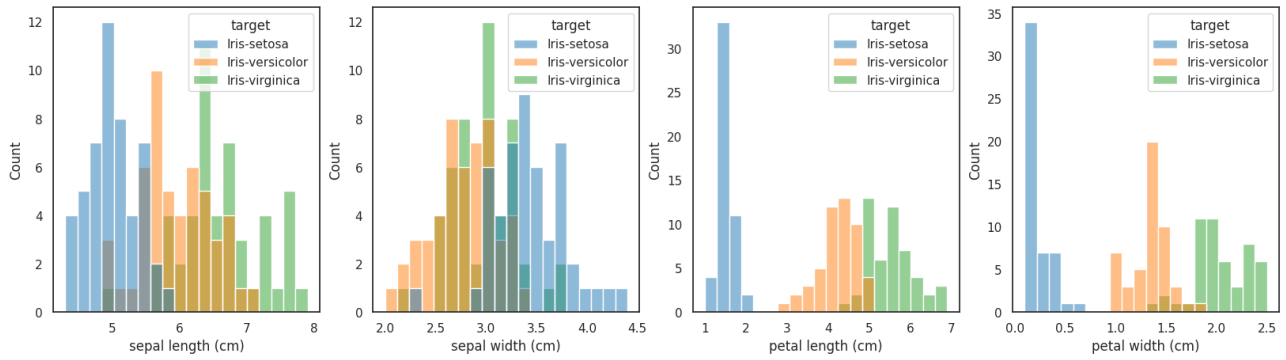


Рисунок 1.5 – Графики гистограмм с пересечением.

Листинг 1.9 – Рисование графиков гистограмм с пересечением.

```
1  fig, axs = plt.subplots(1,4)
2  sb.histplot(df, bins=20, hue='target', x='sepal length (cm)',
palette='tab10', ax=axs[0])
3  sb.histplot(df, bins=20, hue='target', x='sepal width (cm)',
palette='tab10', ax=axs[1])
4  sb.histplot(df, bins=20, hue='target', x='petal length (cm)',
palette='tab10', ax=axs[2])
5  sb.histplot(df, bins=20, hue='target', x='petal width (cm)',
palette='tab10', ax=axs[3])
6  plt.show()
```

Для построения графиков гистограмм с наложением (см. листинг 1.8) и с пересечением (см. листинг 1.9) использован метод *histplot*. Наложение

гистограмм реализуется путём добавления атрибута *multiple* со значением «stack».

1.6.3. Построим гистограммы, чтобы вместо столбцов отображались ступеньки (см. рисунок 1.6).

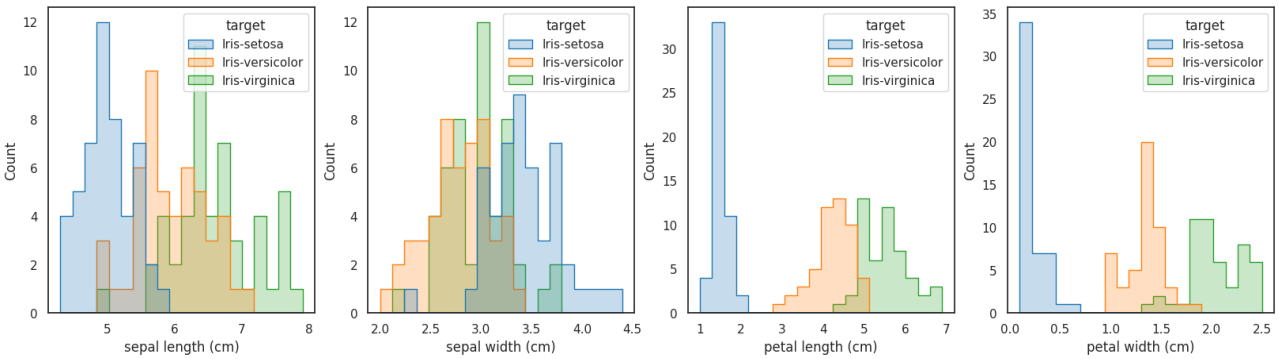


Рисунок 1.6 – Графики ступенчатых гистограмм.

Листинг 1.10 – Рисование ступенчатых гистограмм.

```
1  fig, axs = plt.subplots(1,4)
2  sb.histplot(df, bins=20, hue='target', x='sepal length (cm)',
palette='tab10', element = 'step', ax=axs[0])
3  sb.histplot(df, bins=20, hue='target', x='sepal width (cm)',
palette='tab10', element = 'step', ax=axs[1])
4  sb.histplot(df, bins=20, hue='target', x='petal length (cm)',
palette='tab10', element = 'step', ax=axs[2])
5  sb.histplot(df, bins=20, hue='target', x='petal width (cm)',
palette='tab10', element = 'step', ax=axs[3])
6  plt.show()
```

Графики гистограмм приедены к ступенчатому виду путём добавления атрибута *element* со значением «step» (см. листинг 1.10).

1.6.4. Добавим на гистограммы график ядерной оценки плотности (см. рисунок 1.7).

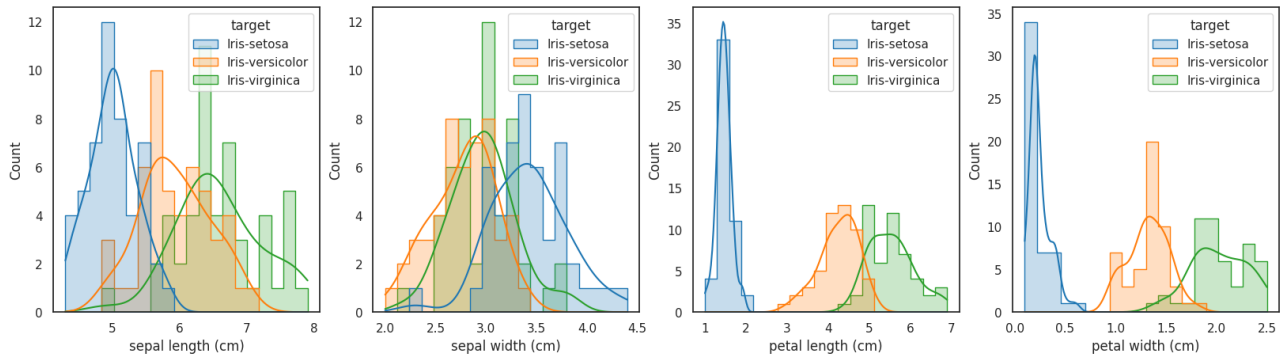


Рисунок 1.7 – Графики гистограмм с ядерной оценкой плотности.

Листинг 1.11 – Рисование гистограмм с ядерной оценкой плотности.

```
1  fig, axs = plt.subplots(1,4)
2  sb.histplot(df, bins=20, hue='target', x='sepal length (cm)',
palette='tab10', kde=True, element = 'step', ax=axs[0])
3  sb.histplot(df, bins=20, hue='target', x='sepal width (cm)',
palette='tab10', kde=True, element = 'step', ax=axs[1])
4  sb.histplot(df, bins=20, hue='target', x='petal length (cm)',
palette='tab10', kde=True, element = 'step', ax=axs[2])
5  sb.histplot(df, bins=20, hue='target', x='petal width (cm)',
palette='tab10', kde=True, element = 'step', ax=axs[3])
6  plt.show()
```

Графики плотности распределения были добавлены к гистограммам при помощи атрибута *kde* со значением *True* (см. листинг 1.11).

Из рисунка 1.7 видно, что полученные графики ядерной оценки плотности совпадают с аналогичными графиками на рисунке 1.1.

2. Изучение набора данных iris.csv с использованием NumPy.

2.1. Загрузим данные из файла как массив NumPy. Для этого будет использован метод *genfromtxt* (см. листинг 2.1).

Листинг 2.1 – Считывание данных из файла в массив NumPy.

```
1  data = np.genfromtxt('iris.csv', delimiter=',', dtype=float)[1::]
```

Чтобы все данные из файла были считаны корректно, в методе *genfrontxt* даны значения атрибутам *delimiter* и *dtype*. *delimeter* устанавливает разделитель, при помощи которого будет производиться деление строк файла на элементы. *dtype* устанавливает тип загружаемых данных, в нашем случае это *float*.

2.2. Выведем первые 10 наблюдений набора данных. Для этого будем использовать срез массива (см. листинг 2.2). Результат вывода см. в таблице 2.1.

Листинг 2.2 – Вывод первых 10 наблюдений.

```
1  data[:10:]
```

Таблица 2.1 – Результат вывода.

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
5.1	3.5	1.4	0.2	0.
4.9	3.0	1.4	0.2	0.
4.7	3.2	1.3	0.2	0.
4.6	3.1	1.5	0.2	0.
5.0	3.6	1.4	0.2	0.
5.4	3.9	1.7	0.4	0.
4.6	3.4	1.4	0.3	0.
5.0	3.4	1.5	0.2	0.
4.4	2.9	1.4	0.2	0.
4.9	3.1	1.5	0.1	0.

Из таблицы 2.1 видно, что все данные загружены корректно.

2.3. Рассчитаем характеристики, полученные методом *describe*, используя методы *NumPy* (см. листинг 2.3). Результат вычислений см. в таблице 2.2.

Листинг 2.3 – Вычисление характеристик массива *NumPy*.

```
1  count = [len(data)]*5
2  mean = np.mean(data, axis=0)
3  std = np.std(data, axis=0)
4  min = np.min(data, axis=0)
5  quartile1 = np.quantile(data, 0.25, axis=0)
6  median = np.quantile(data, 0.5, axis=0)
7  quartile2 = np.quantile(data, 0.75, axis=0)
```

Продолжение листинга 2.3

```
8  max = np.max(data, axis=0)
9  print("count\t", count[1::])
10 print("mean\t", mean[1::])
11 print("std\t", std[1::])
12 print("min\t", min[1::])
13 print("25%\t", quartile1[1::])
14 print("50%\t", median[1::])
15 print("75%\t", quartile2[1::])
16 print("max\t", max[1::])
```

Таблица 2.2 – Характеристики массива *NumPy.*

	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	0.825301	0.434411	1.759404	0.759693	0.816496
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

Параметры *count*, *mean*, *min*, *max*, *25%, 50%, 75%* полностью совпали с соответствующими значениями из метода *describe*. Параметр *std* незначительно отличается.

3. Изучим набор данных варианта 3.

3.1. Загрузим данные из файла lab1_var3.csv как Pandas DataFrame, используя метод *read_csv()* (см. листинг 3.1).

Листинг 3.1 – Считывание датасета из файла lab1_var3.csv.

```
1  var3DF = pd.read_csv('lab1_var3.csv')
```

3.2. Проверим корректность загруженных данных, вызвав у датафрейма метод *head()* (см. листинг 3.2). Ко умолчанию данный метод выводит первые 5 строк. Вывод метода можно видеть в таблице 3.1.

Листинг 3.2 – Вызов метода *head.*

```
1  var3DF.head()
```

Таблица 3.1 – Вывод метода *head.*

	1	2	3	4	label
0	5.782739	6.317180	-5.647817	-7.118864	0
1	5.835754	6.093211	-7.126608	5.134352	0
2	5.261540	6.246352	-6.197188	4.416334	0
3	-3.485119	-7.164230	-4.415422	-8.835990	1
4	3.374649	5.248284	-4.725906	-5.624475	0

3.3. Вызовем у датафрейма метод *describe()* для получения характеристик (см. листинг 3.3). Вывод метода *describe* см. в таблице 3.2.

Листинг 3.3 – Вызов метода *describe.*

```
1  var3DF.describe()
```

Таблица 3.2 – Вывод метода *describe.*

	1	2	3	4	label
count	200.000000	200.000000	200.000000	200.000000	200.000000
mean	3.107028	0.073528	-2.926605	0.083921	0.500000
std	5.353806	6.123719	5.239302	6.166536	0.501255
min	-8.656662	-9.608664	-9.152899	-8.974178	0.000000
25%	1.056465	-5.836700	-6.354661	-5.923682	0.000000
50%	5.782194	0.222078	-5.537275	0.139657	0.500000
75%	6.536680	6.092398	-0.907637	6.108283	1.000000
max	9.618610	8.653637	7.767355	9.082334	1.000000

Из таблицы 3.2 видно, что в датафрейме хранится 200 наблюдений и все они поделены поровну на 2 класса.

3.4. Видоизменим датафрейм таким образом, чтобы метки классов были следующими: 0 - Iris-setosa, 1 - Iris-versicolor и запишем полученный датафрейм

в отдельный файл формата csv. Замену значений столбца *target* будем проводить при помощи метода *replace* (см. листинг 3.4). Получившийся в результате замены датафрейм см. в таблице 3.3.

Листинг 3.4 – Замена значений в столбце датафрейма.

```
di = {0: 'Iris-setosa', 1: 'Iris-versicolor'}

var3DF['label'] = var3DF['label'].replace(di)
var3DF.to_csv("fixed_var3.csv", index=False)

var3DF.head()
```

Таблица 3.3 – Вид изменённого датафрейма.

	1	2	3	4	label
0	5.782739	6.317180	-5.647817	-7.118864	Iris-setosa
1	5.835754	6.093211	-7.126608	5.134352	Iris-setosa
2	5.261540	6.246352	-6.197188	4.416334	Iris-setosa
3	-3.485119	-7.164230	-4.415422	-8.835990	Iris-versicolor
4	3.374649	5.248284	-4.725906	-5.624475	Iris-setosa

3.5. Визуально оценим набор данных, построив изображение, содержащее графики ядерной оценки плотности каждого признака, диаграмму рассеяния и двумерную ядерную оценку плотности для каждого признака (см. рисунок 3.1). Для рисования графиков будет использована библиотека Seaborn. (см. листинг 3.5).

Листинг 3.5 – Рисование графиков.

```
g = sb.PairGrid(var3DF, diag_sharey=False, hue='label',
palette='tab10')
g.map_upper(sb.scatterplot, s=15)
g.map_lower(sb.kdeplot)
g.map_diag(sb.kdeplot, lw=2)
g.add_legend()
```

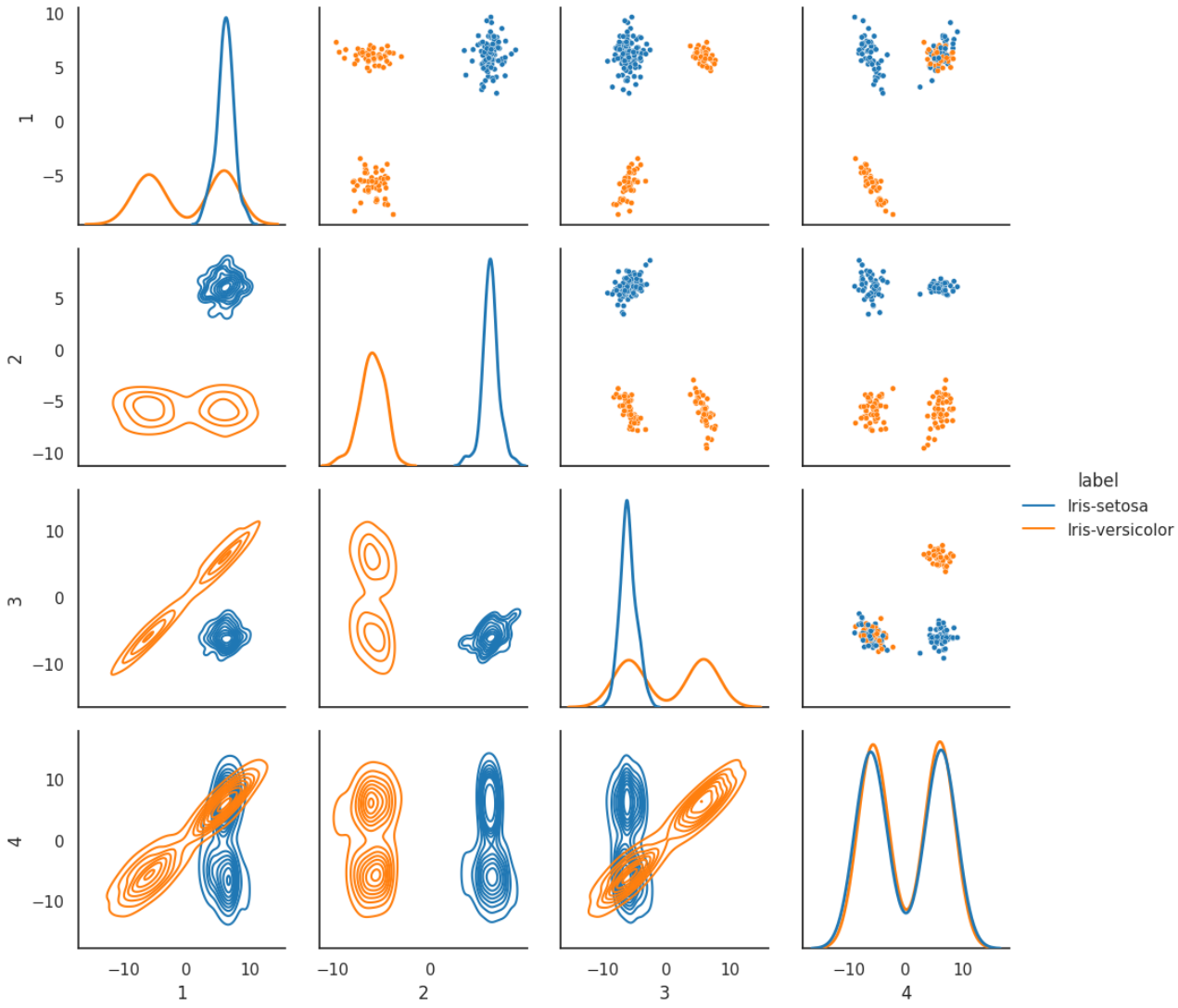


Рисунок 3.1 – Графики для lab1_var3.csv.

Из графиков (см. рисунок 3.1) видно, что классы частично смешиваются на графиках с признаками 1 и 4, а также 3 и 4. Также можно заметить, что графики ядерной оценки плотности обоих классов для признака 4 практически эквивалентны.

3.6. На одном изображении построим гистограммы распределения для каждого признака.

Так как в пунктах 1.6.1-1.6.4 был пройден весь «путь» по формированию требований к итоговому графику, здесь мы сразу построим гистограммы, объединённые с графиками ядерной оценки плотности для каждого признака (см. рисунок 3.2). Гистограммы будут содержать 20 столбцов, так как это значение ранее было принято за оптимальное. Код для рисования графиков см. в листинге 3.6.

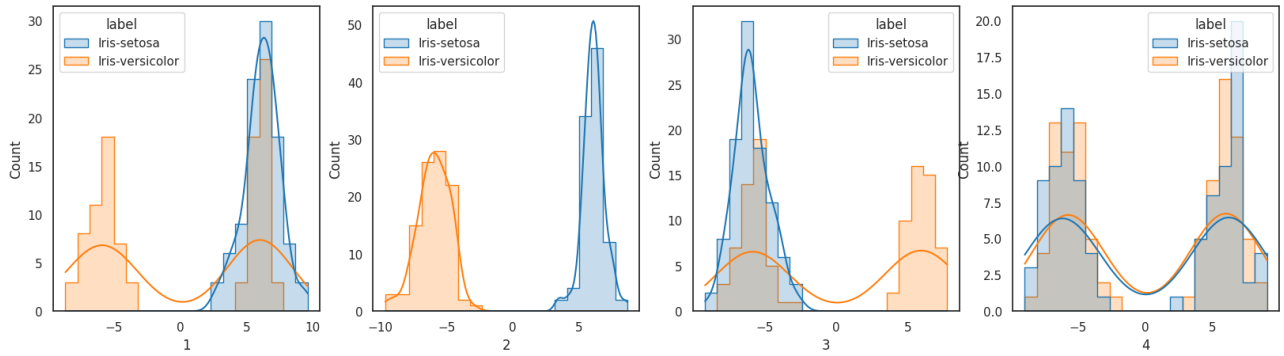


Рисунок 3.2 - Графики гистограмм с ядерной оценкой плотности.

Листинг 3.6 - Рисование гистограмм с ядерной оценкой плотности.

```
1  fig, axs = plt.subplots(1,4)
2  sb.histplot(var3DF, bins=20, hue='label', x='1', palette='tab10',
kde=True, element = 'step', ax=axs[0])
3  sb.histplot(var3DF, bins=20, hue='label', x='2', palette='tab10',
kde=True, element = 'step', ax=axs[1])
4  sb.histplot(var3DF, bins=20, hue='label', x='3', palette='tab10',
kde=True, element = 'step', ax=axs[2])
5  sb.histplot(var3DF, bins=20, hue='label', x='4', palette='tab10',
kde=True, element = 'step', ax=axs[3])
6  plt.show()
```

Из рисунка 3.2 видно, что полученные графики ядерной оценки плотности признаков совпадают с аналогичными графиками на рисунке 3.1. После детального рассмотрения графиков становится понятно, что признак 4 не подходит для классификации полученных данных по предложенным классам из-за слишком большой схожести графиков плотности распределения.

4. Преобразование данных из fixed_iris.csv (см. листинг 1.4).

4.1. Получим из датафрейма столбец с названиями классов. Используя *LabelEncoder* (см. листинг 4.1) и *OneHotEncoder* (см. листинг 4.2), закодируем названия классов. Результаты кодирования см. в таблицах 4.1 и 4.2.

Листинг 4.1 – Кодирование названий классов при помощи *LabelEncoder.*

```
1  labelencoder = slp.LabelEncoder()
2  irisDFLabelEncode = labelencoder.fit_transform(df[['target']])
3  irisDFLabelEncode = pd.DataFrame(irisDFLabelEncode)
4  DFlabel = df.drop("target", axis=1, inplace=False)
```

```
5  DFlabel.join(irisDFLabelEncode)
```

Таблица 4.1 – Результаты кодирования *LabelEncoder.*

Название класса	Код
Iris-setosa	0
Iris-versicolor	1
Iris-verginica	2

Листинг 4.2 – Кодирование названий классов при помощи *OneHotEncoder.*

```
1  onehotencoder = slp.OneHotEncoder()
2  irisDFHotEncode=onehotencoder.fit_transform(df[['target']]).toarray()
3  irisDFHotEncode = pd.DataFrame(irisDFHotEncode)
4  DFhot = df.drop("target", axis=1, inplace=False)
5  DFhot.join(irisDFHotEncode)
```

Таблица 4.2 – Результаты кодирования *OneHotEncoder*.

Название класса	0	1	2
Iris-setosa	1	0	0
Iris-versicolor	0	1	0
Iris-verginica	0	0	1

Из результатов кодирования (см. таблицы 4.1 и 4.2) видно, что *LabelEncoder* присваивает каждому уникальному значению одно числовое значение. Однако, в нашем случае, датафрейм содержит >2 классов, из-за чего данный метод кодирования плохо подходит, так как моделям тяжело подстраиваться под множество дискретных значений. В нашем случае намного лучше работает *OneHotEncoder*, который осуществляет кодирование при помощи векторов, состоящих из 0 и 1.

4.2. Из датафрейма, полученного в п 1.4 выделим все столбцы признаков и преобразуем их в массив *NumPy* (см. листинг 4.3).

Листинг 4.3 – Преобразование столбцов в массив *NumPy*.

```
1  tmpDF = df[['sepal length (cm)', 'sepal width (cm)', 'petal length
(cm)', 'petal width (cm)']]
2  tmpDF = tmpDF.to_numpy()
```

Перед преобразованием датафрейма в массив при помощи списка параметров были выбраны все необходимые столбцы, после чего к полученному датафрейму был применен метод *to_numpy*.

4.3. Для полученного в п. 4.2 массива применим *StandardScaler* (см. листинг 4.4), *MinMaxScaler* (см. листинг 4.5), *MaxAbsScaler* (см. листинг 4.6), *RobustScaler* (см. листинг 4.7). Для каждого результата построим гистограммы по всем признакам без разделения по классам (см. рисунки 4.1, 4.2, 4.3, 4.4).

Листинг 4.4 – Применение *StandardScaler*.

```
1  sscaler = slp.StandardScaler()
2  ssDF = sscaler.fit_transform(tmpDF)
3  fig, axs = plt.subplots(1,4)
4  sb.histplot(ssDF[:,0], bins=20, ax=axs[0])
5  sb.histplot(ssDF[:,1], bins=20, ax=axs[1])
6  sb.histplot(ssDF[:,2], bins=20, ax=axs[2])
7  sb.histplot(ssDF[:,3], bins=20, ax=axs[3])
8  plt.show()
```

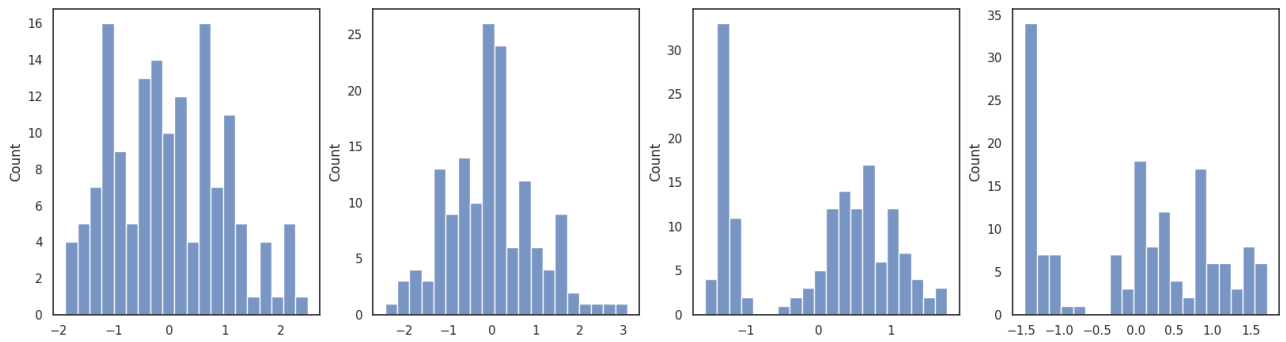


Рисунок 4.1 – Гистограммы распределения признаков после *StandardScaler*.

В листинге 4.4 представлено нормирование при помощи *StandardScaler*. Эта нормировка изменяет размер распределения значений так, чтобы среднее значение наблюдаемых значений было равно 0, а стандартное отклонение – 1. Математически такое преобразование можно представить формулой 4.1.

$$\hat{X} = \frac{X - \mathbb{E}X}{\sigma X} \quad (4.1)$$

Листинг 4.5 - Применение *MinMaxScaler*.

```
1  mmcaler = slp.MinMaxScaler()
2  mmDF = mmscaler.fit_transform(tmpDF)
3  fig, axs = plt.subplots(1,4)
4  sb.histplot(mmDF[:,0], bins=20, ax=axs[0])
5  sb.histplot(mmDF[:,1], bins=20, ax=axs[1])
6  sb.histplot(mmDF[:,2], bins=20, ax=axs[2])
7  sb.histplot(mmDF[:,3], bins=20, ax=axs[3])
8  plt.show()
```

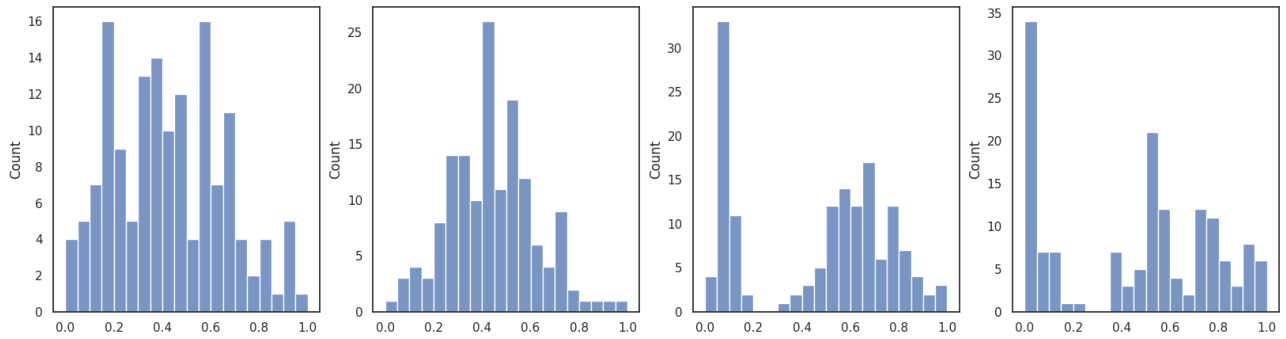


Рисунок 4.2 - Гистограммы распределения признаков после *MinMaxScaler*.

В листинге 4.5 приведено нормирование с использованием *MinMaxScaler*. Этот метод приводит все элементы из набора к значениям, лежащим в промежутке [0, 1], используя наименьшее и наибольшее значения из набора. Математически это преобразование представлено в формуле 4.2.

$$\hat{X} = \frac{X - \min(X)}{\max(X) - \min(X)} \quad (4.2)$$

Листинг 4.6 – Применение *MaxAbsScaler*.

```
1  mascaler = slp.MaxAbsScaler()
2  maDF = mascaler.fit_transform(tmpDF)
3  fig, axs = plt.subplots(1,4)
4  sb.histplot(maDF[:,0], bins=20, ax=axs[0])
5  sb.histplot(maDF[:,1], bins=20, ax=axs[1])
6  sb.histplot(maDF[:,2], bins=20, ax=axs[2])
7  sb.histplot(maDF[:,3], bins=20, ax=axs[3])
8  plt.show()
```

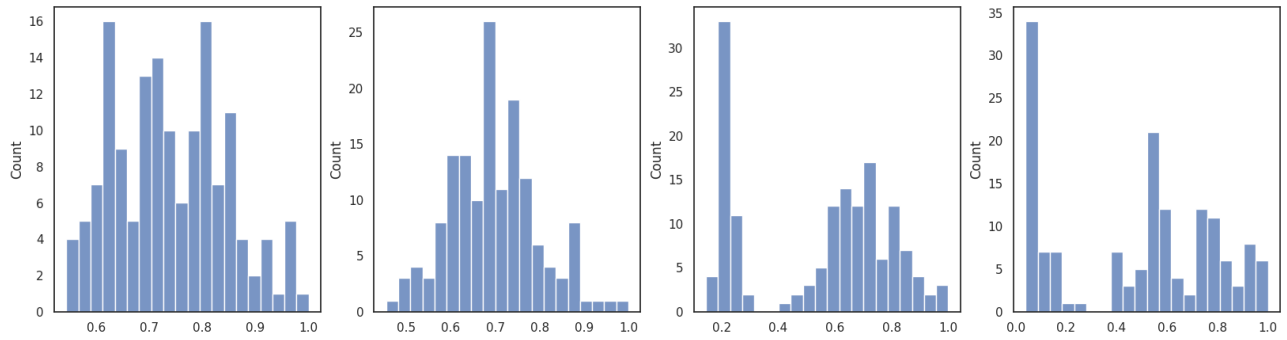


Рисунок 4.3 - Гистограммы распределения признаков после *MaxAbsScaler*.

Нормализацию с использованием *MaxAbsScaler* см. в листинге 4.6. Данный метод нормализации преобразует все элементы таким образом, чтобы они лежали в промежутке [-1, 1]. Выполняется это посредством деления каждого элемента набора на максимальное по модулю значение из этого набора. Математическое представление можно увидеть на формуле 4.3.

$$\hat{X} = \frac{X}{\max(|X|)} \tag{4.3}$$

Листинг 4.7 - Применение *RobustScaler*.

```
1  rscaler = slp.MaxAbsScaler()
2  rDF = rscaler.fit_transform(tmpDF)
3  fig, axs = plt.subplots(1,4)
4  sb.histplot(rDF[:,0], bins=20, ax=axs[0])
5  sb.histplot(rDF[:,1], bins=20, ax=axs[1])
6  sb.histplot(rDF[:,2], bins=20, ax=axs[2])
7  sb.histplot(rDF[:,3], bins=20, ax=axs[3])
8  plt.show()
```

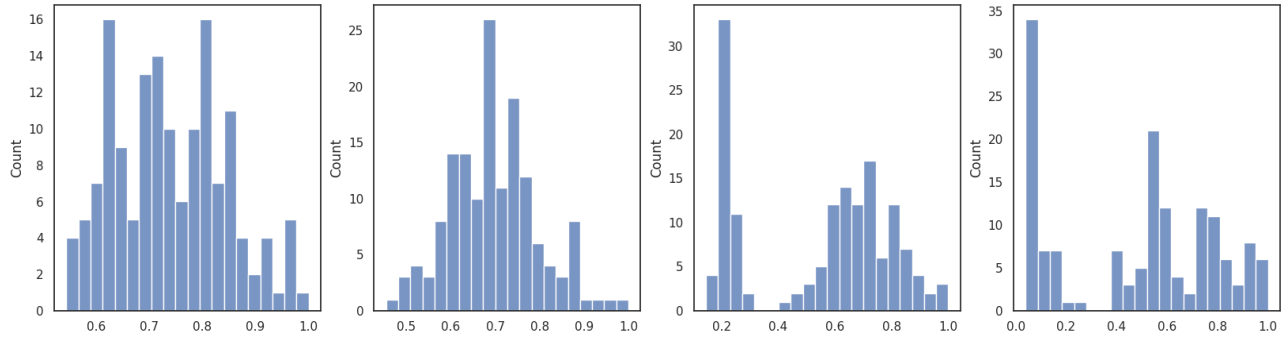


Рисунок 4.4 - Гистограммы распределения признаков после *RobustScaler*.

Нормализацию с использованием *RobustScaler* см. в листинге 4.7. Этот способ нормализации заключается в центрировании значений на основе медианы, 1 и 3 квартилей. Преобразование представлено на формуле 4.4.

$$\hat{X} = \frac{X - Q_2}{Q_3 - Q_1} \quad (4.4)$$

Из представленных наборов графиков видно, что форма гистограмм для каждого метода повторяется, ключевые отличия заключаются в значениях, находящихся на оси абсцисс.

4.4. Реализуем метод нормирования *MinMaxScaler* с использованием *NumPy*. Получившуюся функцию см. в листинге 4.8.

Листинг 4.8 – Функция *myMinMaxScaler*.

```
1  def myMinMaxScaler(arr):
2    min = np.min(arr, axis=0)
3    max = np.max(arr, axis=0)
4    res = []
5    for el in arr:
6       exp = []
7       for i in range(0,4):
8          exp.append((el[i] - min[i]) / (max[i] - min[i]))
9       res.append(exp)
10   res = np.array(res)
11   return res
```

Функция *myMinMaxScaler* принимает на вход *arr* - массив *NumPy*. В теле функции производится вычисление *max* и *min* по каждому параметру и создание результирующего списка *res*, после чего два цикла проходится по всем элементам из *arr*. Для каждого элемента вычисляется новое значение по формуле 4.2 и помещается в список *exp*, который, после заполнения значениями для каждого параметра рассматриваемого наблюдения, помещается в конец списка *res*. После окончания обработки всех элементов изначального массива список *res* преобразуется в массив *NumPy* при помощи метода *array*. Функция возвращает *res*.

Для подтверждения корректности написанной функции для нормализации выведем первые 5 строк из эталонного алгоритма (см. таблицу 4.1) и первые 5 строк массива, который вернула *myMinMaxScaler* (см. таблицу 4.2).

Таблица 4.1 – Результат работы *MinMaxScaler*.

0.22222222	0.62500000	0.06779661	0.04166667
0.16666667	0.41666667	0.06779661	0.04166667
0.11111111	0.50000000	0.05084746	0.04166667
0.08333333	0.45833333	0.08474576	0.04166667
0.19444444	0.66666667	0.06779661	0.04166667

Таблица 4.2 - Результат работы *myMinMaxScaler*.

0.22222222	0.62500000	0.06779661	0.04166667
0.16666667	0.41666667	0.06779661	0.04166667
0.11111111	0.50000000	0.05084746	0.04166667
0.08333333	0.45833333	0.08474576	0.04166667
0.19444444	0.66666667	0.06779661	0.04166667

Из таблиц 4.1 и 4.2 несложно видеть, что результаты работы функций абсолютно идентичны. Из этого делаем вывод, что написанная функция корректно выполняет нормирование набора данных.

Теперь вычислим минимальное, максимальное, среднее значения и дисперсию после преобразования (см листинг 4.9). Результат вычислений см. в таблице 4.3.

Листинг 4.9 – Вычисление характеристик набора после преобразования.

```
1  mean = np.mean(resDF, axis=0)
2  min = np.min(resDF, axis=0)
3  max = np.max(resDF, axis=0)
4  var = np.var(resDF, axis=0)
5  print("min\t", min)
6  print("max\t", max)
7  print("mean\t", mean)
8  print("var\t", var)
```

Таблица 4.3 – Результат вычисления характеристик после преобразования.

min	0.0000000	0.0000000	0.0000000	0.0000000
max	1.0000000	1.0000000	1.0000000	1.0000000
mean	0.4287037	0.44055556	0.46745763	0.45805556
var	0.05255573	0.03276265	0.08892567	0.10019668

Результаты, приведённые в таблице 4.3 полностью совпали с аналогичными результатами для результата работы *MinMaxScaler*, это говорит о том, что нормировка проведена успешно.

4.5. Для датафрейма из п. 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 4.10).

Листинг 4.10 – Преобразование датафрейма.

```
1  new_df = df[(df["target"] == "Iris-versicolor") | (df["target"] ==
"Iris-virginica")]
2  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))]
3  new_df = new_df[["sepal length (cm)", "petal length (cm)"]]
```

В начале преобразования выберем только те наблюдения, класс которых «Iris-versicilor» или «Iris-virginica». Далее выберем только те наблюдения, значение параметра «sepal width (cm)» которого находятся между 25% и 75%. После всех выше перечисленных действий выберем из полученного действия только столбцы с признаками «sepal length (cm)» и «petal length (cm)».

5. Понижение размерности.

5.1. Для набора данных iris.csv применим понижение размерности до 2, используя *PCA* и *TSNE* из *Sklearn* (см. листинги 5.1, 5.2). Для каждого из результатов построим диаграмму рассеяния с выделением разным цветом наблюдений разных классов (см. рисунки 5.1, 5.2).

Листинг 5.1 – Понижение размерности, используя *PCA*.

```
pca = PCA(n_components = 2)
pcaDF = pca.fit_transform(df)
pcaDF = pd.DataFrame(pcaDF, columns=["1_col", "2_col"])
pcaDF.insert(loc=pcaDF.shape[1], column="target", value=df["target"])

sb.scatterplot(pcaDF, x = "1_col", y = "2_col", hue="target",
palette="tab10")
```

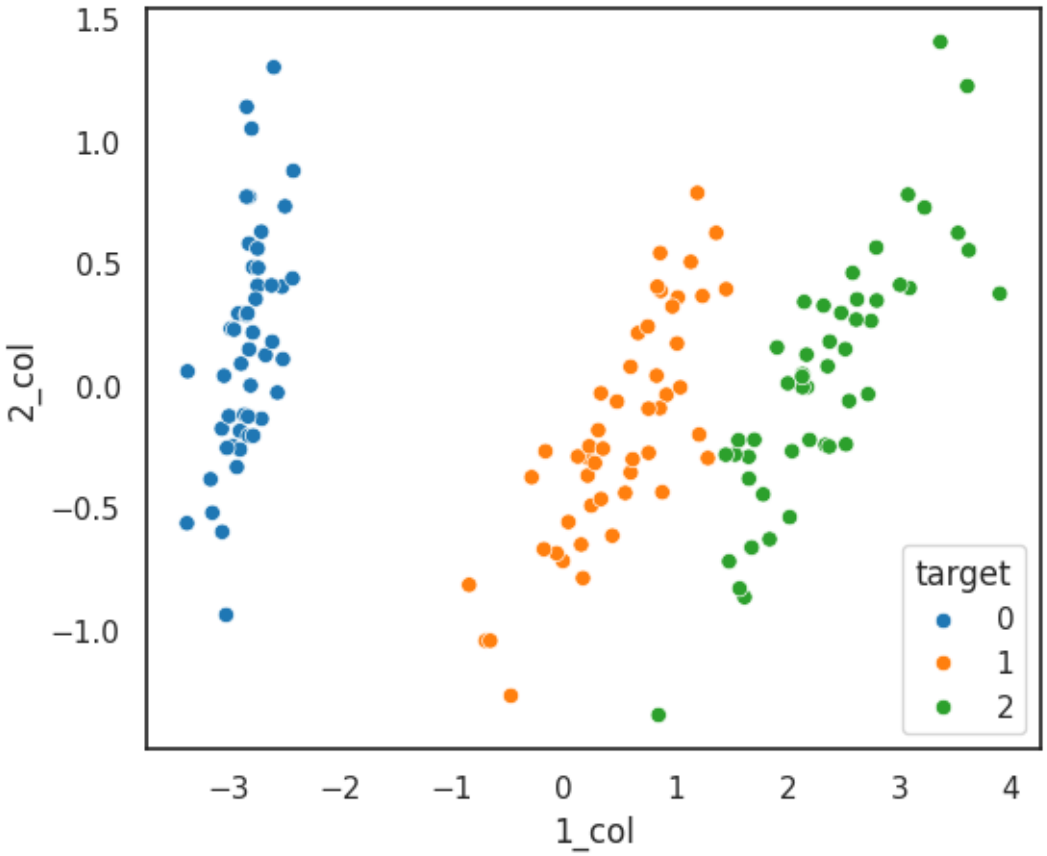


Рисунок 5.1 – Диаграмма рассеивания *PCA*.

Листинг 5.2 - Понижение размерности, используя *t-SNE*.

```
tsne = TSNE(n_components = 2)
tsneDF = tsne.fit_transform(df)
tsneDF = pd.DataFrame(tsneDF, columns=["1_col", "2_col"])
tsneDF.insert(loc=tsneDF.shape[1], column="target",
value=df["target"])
sb.scatterplot(tsneDF, x = "1_col", y = "2_col", hue="target",
palette="tab10")
```

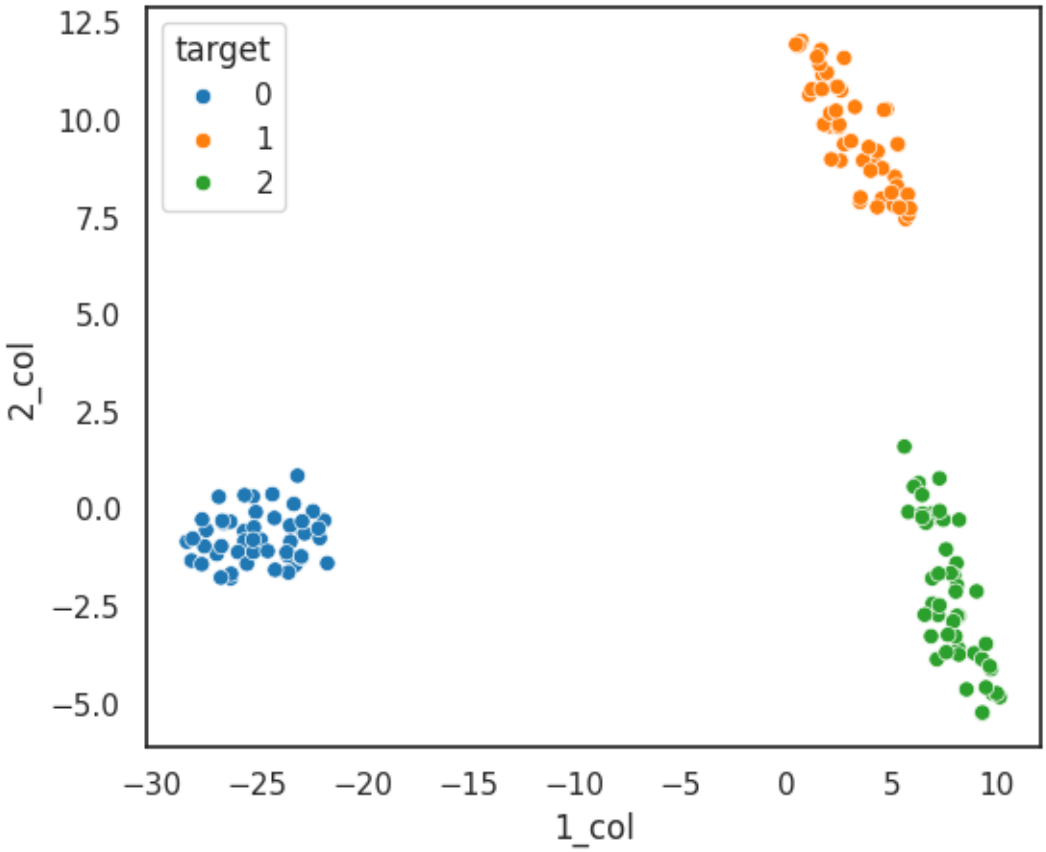


Рисунок 5.2 - Диаграмма рассеивания *t-SNE*.

Методы *PCA* и *t-SNE* используются для понижения размерности – уменьшения количества признаков, выделяя только наиболее значимые и отбрасывая незначительные. Понижение размерности также используется для устранения шумов и визуализации многомерных пространств.

Результат работы методов представлен на рисунках 5.1 и 5.2.

Можно видеть, что после преобразования *PCA* были устранены шумы, а точки одного класса сгруппировались, однако всё ещё есть случаи, когда отдельные точки одного цвета лежат очень близко к другим группам.

В свою очередь, понижение размерности методом *t-SNE* распределило точки так, что все классы чётко видны, и нет ни одной точки, «выпадающей» из своей группы.

Выводы.

В ходе выполнения лабораторной работы были изучены способы анализа и обработки данных. Использовались библиотеки *Pandas* и *NumPy*, позволяющие считывать датафреймы и вычислять числовые характеристики наборов данных.

Как способ графического представления данных были использованы гистограммы, графики ядерной оценки плотности и двумерной ядерной оценки плотности из библиотек *Seaborn* и *Matplotlib*. Также были изучены способы кодирования данных, такие как *LabelEncoder* и *OneHotEncoder*. Для нормировки наборов данных были использованы *StandardScaler*, *MinMaxScaler*, *MaxAbsScaler* и *RobustScaler*. Изучены два способа понижения размерности наборов данных: *PCA* и *t-SNE*.