Тема практического занятия
«Визуализация с помощью библиотеки Seaborn»

1. Особенности Python Seaborn

Seaborn – популярная библиотека готовых шаблонов для статистической визуализации, написанная на бэкенде Matplotlib. Библиотека Python Seaborn используется для облегчения задачи визуализации данных. В Seaborn доступны различные цветовые палитры для создания привлекательных изображений.

В основном, Seaborn исправляет два недостатка Matplotlib:

1. Matplotlib можно освоить, но иногда сложно понять, какие настройки необходимы, чтобы сделать графики более привлекательными. Seaborn же предлагает множество настраиваемых функций и высокоуровневых интерфейсов для решения этой проблемы.
2. При работе с Pandas, Matplotlib непосредственно не подходит для работы с DataFrame, тогда как функции Seaborn фактически работают с DataFrame. Библиотека Matplotlib была выпущена на десятилетие раньше, чем библиотека Pandas, и потому не ориентирована на работу с объектами DataFrame библиотеки Pandas. Seaborn дополняет и расширяет Matplotlib. Seaborn построен поверх базовой библиотеки визуализации Python Matplotlib. Он должен служить дополнением, а не заменой.

Seaborn имеет некоторые важные функции. Вот некоторые из них:

- Встроенные функции для оформления графики matplotlib.
- Визуализация одномерных и двумерных данных.
- Подгонка и визуализация моделей линейной регрессии.
- Построение статистических данных временных рядов.
- Seaborn хорошо работает со структурами данных NumPy и Pandas.
- Он поставляется со встроенными функциями для оформления графики Matplotlib.

Отчет со вставленными комментариями можете присылать в виде выполненного программного кода с рисунками: **фамилия.ipynb** или отчета, выполненного в текстовом редакторе Microsoft Word: **фамилия.docs**.

В некоторых примерах есть задания – выделены красным шрифтом.

Содержание практического занятия по визуализации в Seaborn:

2. Импорт наборов данных и библиотек

2.1. Импорт библиотек

Начнем с импорта библиотек: Pandas – библиотеки для управления реляционными (в табличном формате) наборами данных. Seaborn очень удобен при работе с DataFrame – наиболее широко используемой структурой данных для анализа данных.

Следующая команда поможет вам импортировать Pandas:

```
import pandas as pd
```

Теперь импортируем библиотеку Matplotlib, которая помогает настраивать графики.

```
from matplotlib import pyplot as plt
```

Импортируем библиотеку Seaborn с помощью следующей команды:

```
import seaborn as sb
```

Импортируем библиотеку Numpy

```
import numpy as np
```

2.2. Импорт наборов данных

Библиотека Seaborn поставляется с несколькими важными наборами данных. Если Seaborn доступен, наборы данных загружаются автоматически. Вы можете использовать любой из этих наборов данных для обучения. С помощью следующей функции вы можете загрузить необходимый набор данных из 18, доступных в Seaborn: `load_dataset()`.

Чтобы просмотреть все доступные наборы данных в библиотеке Seaborn, вы можете использовать следующую команду с функцией `get_dataset_names()`: `sb.get_dataset_names()`.

Пример 1. Просмотрите типы доступных в Seaborn наборов данных, скопируйте их и вставьте в результат примера 1.

 # Пример 1

```
sb.get_dataset_names()
```

Пример 2. Импортируйте набор данных `'iris'`. Наборы данных загружаются как Pandas DataFrame по умолчанию.

 # Пример 2.

```
# а) Импорт набора данных 'iris'
data = sb.load_dataset('iris')
data.head()
```

```
# б) Импорт набора данных 'tips' - чаевые
df = sb.load_dataset('tips')
df.head()
```

DataFrame хранит данные в форме прямоугольных сеток, с помощью которых можно легко просматривать данные. Каждая строка прямоугольной сетки содержит значения экземпляра, а каждый столбец сетки представляет собой вектор, который содержит данные для определенной переменной. Это означает, что строки DataFrame не обязательно должны содержать значения одного и того же типа данных, они могут быть числовыми, символьными, логическими и т.д. DataFrames для Python поставляются с библиотекой Pandas и определяются как двумерные структуры данных с потенциально разными типами столбцов.

3. Визуализация в Seaborn: представление данных эффективным и простым способом

Библиотека Matplotlib отлично поддерживает настройку графиков и рисунков, но для ее использования необходимо знать, какие параметры нужно настроить для получения привлекательного и желаемого изображения. В отличие от Matplotlib, Seaborn поставляется с настроенными функциями и высокоуровневым интерфейсом для настройки и управления внешним видом фигур Matplotlib.

Пример 3. Постройте изображение с настройками по умолчанию в Matplotlib:

```
def sinplot(flip = 1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 5):
        plt.plot(x, np.sin(x + i*.5)*(7 - i)*flip)
sinplot()
plt.show()
```

Пример 4. Измените тот же график на значения по умолчанию в Seaborn, используя функцию `set()`:

```
def sinplot(flip = 1):
   x = np.linspace(0, 14, 100)
   for i in range(1, 5):
       plt.plot(x, np.sin(x + i*.5)*(7 - i)*flip)

sb.set()
sinplot()
plt.show()
```

Сравните полученные два рисунка по умолчанию в Matplotlib и Seaborn. Укажите разницу в графиках.

==========

По сути, Seaborn разбивает параметры Matplotlib на две группы:

- Стили фона
- Масштаб рисунка

4. Seaborn Figure Styles

Интерфейсом для манипулирования стилями является **`set_style()`**. С помощью этой функции вы можете установить «стиль» фона. Ниже представлены пять доступных «стилей»:

- `"darkgrid"` – темный фон, белая сетка
- `"whitegrid"` – белый фон, темная сетка
- `"dark"` – темный фон
- `"white"` – белый фон
- `"ticks"` – белый фон, обрамление рисунка

Попробуйте применить стили из вышеупомянутого списка. Стилем по умолчанию является `"darkgrid"`, который был использован в предыдущем примере.

Пример 5. Применим стиль `"whitegrid"`

```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 5):
        plt.plot(x, np.sin(x + i*.5)*(7 - i)*flip)
sb.set_style("whitegrid")
sinplot()
plt.show()
```

Укажите разницу между двумя рисунками (примеры 4 и 5).

Удаление осей сетки

В стилях `"white"` и `"ticks"` мы можем удалить верхнюю и правую оси (границы рисунка), используя функцию **`despine()`** (что не поддерживается в Matplotlib).

Пример 6.

```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 5):
        plt.plot(x, np.sin(x + i*.5)*(7 - i)*flip)
sb.set_style("white")
sinplot()
sb.despine()
```

```
plt.show()
```

Переопределение элементов

Если вы хотите настроить стили Seaborn, вы можете воспользоваться словарем параметров функции **set_style()**. Доступные параметры просматриваются с помощью функции **axes_style()**.

Пример 7. Выполните инструкцию:

```
sb.axes_style()
```

Изменение значений любого параметра изменит стиль графика.
Пример 8.

```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 5):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)

sb.set_style("darkgrid", {'axes.axisbelow': False}) # Изменили здесь
sinplot()
sb.despine()
plt.show()
```

Сравните два изображения: из примера 8 и примера 9. В чем отличие?
Пример 9.

```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 5):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)

sb.set_style("darkgrid", {'axes.axisbelow': True}) # Изменили здесь
sinplot()
sb.despine()
plt.show()
```

Уточнение: 'axes.axisbelow' может принимать два значения: False и True.

5. Цветовая палитра Seaborn

Цвет играет важную роль, чем любой другой аспект визуализации. Seaborn предоставляет функцию **`color_palette()`**, которую можно использовать для придания цвета графикам и для большего эстетического вида.

Синтаксис функции

```
seaborn.color_palette (palette = None, n_colors = None, desat = None)
```

Параметры для построения цветовой палитры:

`n_colors` – количество цветов в палитре.
Если `None`, значение по умолчанию будет зависеть от того, как указана палитра. По умолчанию значение **`n_colors`** составляет ? (выясним в примере 10) цветов.

`Desat` – пропорция для варьирования каждого цвета.
Приведем доступные палитры Seaborn:

- глубокий
- приглушенный
- яркий
- пастельный
- темный

Иногда трудно решить, какую палитру следует использовать для данного набора данных, не зная характеристик данных. Приведем различные способы использования типов **`color_palette():`**

- качественные палитры
- последовательные палитры
- расходящаяся палитра

Есть и другая функция **seaborn.palplot(),** которая работает с цветовой палитрой. Эта функция отображает цветовую палитру как горизонтальный массив. Узнаем больше о **seaborn.palplot ()** из следующих примеров.

5.1. Качественные цветовые палитры

Качественные или категориальные палитры лучше всего подходят для построения категориальных данных.

Пример 10.

```
current_palette = sb.color_palette()
sb.palplot(current_palette)
plt.show()
```

Мы не передали никаких параметров в **color_palette().**

Сколько цветов по умолчанию отобразились?

Примечание. Вы можете увидеть желаемое количество цветов, передав значение в параметр **n_colors**. Здесь **palplot()** используется для горизонтального построения массива цветов.

5.2. Последовательные цветовые палитры

Последовательные палитры подходят для отображения распределения данных в диапазоне от относительно более низких значений до более высоких значений в пределах диапазона.

При добавлении дополнительного символа ‘s’ к цвету, переданному параметру цвета, будет построен график «Последовательная палитра».

Пример 11. а) Выведите рисунок последовательной палитры.

```
current_palette = sb.color_palette()
sb.palplot(sb.color_palette("Greens"))
plt.show()
```

б) **Выберите другой цвет и приведите изображение последовательной палитры.**

5.3. Расходящаяся палитра

Расходящиеся палитры используют два разных цвета. Каждый цвет представляет изменение значения в пределах от общей точки в любом направлении.

Предположим, что данные располагаются в диапазоне от -1 до 1. Значения от -1 до 0 принимают один цвет, а от 0 до +1 – другой цвет.

По умолчанию значения центрированы от нуля. Вы можете управлять ими с помощью параметра `center`, передавая значение.

Пример 12. а) Выведите рисунок расходящейся палитры.

```
current_palette = sb.color_palette()
sb.palplot(sb.color_palette("BrBG", 7))
plt.show()
```

б) Приведите рисунок расходящейся палитры с другим сочетанием цветов.

5.4. Настройка цветовой палитры по умолчанию

У функций **color_palette()** есть сопутствующий элемент **set_palette()**. Аргументы одинаковы как для **set_palette(),** так и для **color_palette().** Параметры Matplotlib по умолчанию настроены так, что палитра используется для всех графиков.

Пример 13.

```
def sinplot(flip = 1):
   x = np.linspace(0, 14, 100)
   for i in range(1, 5):
       plt.plot(x, np.sin(x + i * .5) * (7 - i)*flip)

sb.set_style("white")
sb.set_palette("husl")
sinplot()
plt.show()
```

6. Пройдемся по основным видам графиков

6.1. Построение одномерного распределения

Распределение данных – это главное, что нужно знать при анализе данных. Здесь мы увидим, как seaborn помогает в понимании одномерного распределения данных.

Пример 14.

```
sb.set()
# Набор данных 'чаевые'
tips = sb.load_dataset('tips')
ax = sb.scatterplot(x = 'total_bill', y='tip', data = tips)
plt.show()
```

Как вы думаете есть ли зависимость между параметрами `tip` и `total_bill`? Как Вы ее можете охарактеризовать?

6.2. Гистограмма

Когда мы занимаемся наукой о данных, одной из самых распространенных задач является визуализация распределения данных.

Часто мы хотим понять, как наши данные распределяются в рамках исследовательского анализа данных.

Иногда мы исследуем данные, чтобы выяснить, как они структурированы (например, когда мы впервые получаем набор данных).

В других случаях нам нужно изучить распределения данных, чтобы ответить на вопрос или подтвердить какую-либо гипотезу о данных.

Изучение распределений данных также очень распространено в машинном обучении, поскольку многие методы машинного обучения предполагают, что данные распределяются определенным образом
Существует два основных способа изучения распределения данных: **гистограмма** и **график плотности**.

Гистограмма является наиболее распространенным инструментом для изучения распределения данных. В типичной гистограмме мы сопоставляем числовую переменную с осью x.

Затем ось x делится на несколько «интервалов», например, может быть интервал от 10 до 20, следующий интервал от 20 до 30, следующий от 30 до 40 и так далее.

Когда мы создаем гистограмму (или используем программное обеспечение для создания гистограммы), мы подсчитываем количество наблюдений в каждой ячейке.

Затем мы строим полосу для каждого бункера. Длина полосы соответствует количеству записей, находящихся в этом интервале по оси x.

Таким образом, гистограмма представляет распределение данных, формирует ячейки вдоль диапазона изменения данных, а затем рисует столбцы, чтобы показать количество наблюдений, попадающих в каждую ячейку. В конечном итоге гистограмма содержит группу столбцов, которые показывают «высоту» данных (то есть количество данных) для различных значений нашей числовой переменной. Гистограмма показывает нам, как распределяется переменная.

График плотности – другой основной инструмент для оценки распределения данных. Один из наиболее распространенных методов построения графиков называется «оценка плотности ядра». График, который мы создаем при использовании оценки ядерной плотности, называется «графиком ядерной оценки плотности». Они также известны как «графики KDE».

Графики KDE (то есть графики плотности) очень похожи на гистограммы с точки зрения того, как мы их используем. Мы используем графики плотности, чтобы оценить, как распределяется числовая переменная.

Основное отличие состоит в том, что графики KDE используют плавную линию для отображения распределения, тогда как гистограммы используют столбики. Итак, графики KDE показывают плотность, тогда как гистограммы показывают количество.

Разберемся с гистограммами и графиками KDE в контексте Seaborn.

Seaborn имеет две разные функции для визуализации одномерного распределения данных – `seaborn.kdeplot()` и `seaborn.distplot()`. Будем использовать распространенное соглашение о вызове функций: `sb.distplot()` и `sb.kdeplot()`.

Функция **distplot()** предоставляет наиболее удобный способ быстро взглянуть на одномерное распределение. Эта функция строит комбинированный рисунок, который содержит гистограмму и график оценки плотности ядра данных.

Вы можете использовать функцию `distplot` для создания диаграммы только с гистограммой или только с графиком KDE.

Функция `sb.distplot()` имеет около десятка параметров, которые вы можете использовать. Однако большинство из них вам не понадобится. Сосредоточимся на нескольких наиболее распространенных параметрах для `sb.distplot()`.

Параметры:

– `color` – меняет цвет графика KDE и гистограммы;

– `kde` – включает и выключает график KDE в выводе.

Этот параметр принимает в качестве аргумента логическое значение (`True` или `False`). По умолчанию для параметра `kde` установлено значение `kde = True`. Это означает, что по умолчанию функция `sb.distplot` будет включать оценку плотности ядра вашей входной переменной.

Если вы вручную установите `kde = False`, тогда функция удалит график KDE.

– `hist` – определяет, будет ли гистограмма отображаться на выходе.

По умолчанию установлено значение `hist = True`, что означает, что по умолчанию выходной график будет включать гистограмму входной переменной. Если вы установите `hist = False`, функция удалит гистограмму из вывода.

– `bins` – отвечает за количество интервалов в выходной гистограмме.

Если вы не установите значение для параметра `bins`, функция автоматически вычислит соответствующее количество интервалов.

Примеры визуализации распределений с помощью Seaborn

Пример 15. Поиграем с параметрами:

```
df = sb.load_dataset('iris')
sb.distplot(df['petal_length'],kde = False)
plt.show()
```

Прокомментируйте программный код и полученный рисунок.

Примечание. Оценка плотности ядра (KDE) – это способ оценки функции плотности вероятности непрерывной случайной величины. Используется для **непараметрического анализа**.

Пример 16. Играем с параметрами:

```
df = sb.load_dataset('iris')
sb.distplot(df['petal_length'], hist=False)
plt.show()
```

Прокомментируйте программный код и полученный рисунок.

6.3. Параметрическое распределение

distplot() используется для визуализации параметрического распределения набора данных.

Пример 17. Продолжаем играть с параметрами:

а) **Прокомментируйте программный код и полученный рисунок.**

```
df = sb.load_dataset('iris')
sb.distplot(df['petal_length'])
plt.show()
```

б) **Сравните два рисунка а) и б).**

```
df = sb.load_dataset('iris')
sb.distplot(df['petal_length'], hist_kws = {"alpha": 1})
plt.show()
```

Примечание. Вы, наверное, заметили, что по умолчанию гистограмма на графике а) более прозрачна. Это настройка по умолчанию. Как сделать цвет более непрозрачным?

Для этого нужно использовать параметр `alpha` из matplotlib. Причем, для этого необходимо использовать параметр `hist_kws` из `sb.distplot` для доступа к базовому параметру matplotlib.

Код `hist_kws = {"alpha": 1}` обращается к параметру `alpha` из matplotlib и устанавливает `alpha` равным 1.

6.4. Построение двумерного распределения

Двумерное распределение используется для определения связи между двумя переменными. Это в основном касается отношений между двумя переменными и того, как одна переменная ведет себя по отношению к другой.

Лучший способ проанализировать двумерное распределение – использовать функцию **`jointplot()`**.

`Jointplot` создает многопанельную фигуру, которая проецирует двумерные отношения между двумя переменными, а также одномерное распределение каждой переменной по отдельным осям.

Scatter Plot - диаграмма рассеяния является наиболее удобным способом визуализации распределения, в котором каждое наблюдение представлено в двухмерном графике через оси x и y.

Пример 18.

```
df = sb.load_dataset('iris')
sb.jointplot(x = 'petal_length', y = 'petal_width', data = df)
plt.show()
```

Между какими переменными рассматривается взаимосвязь? По вашему мнению, существует ли связь между переменными и какая?

6.5. Визуализация парных отношений

Наборы данных, как правило, содержат много переменных. В таких случаях анализируют отношение между каждыми переменными. Построение двумерного распределения на практике для всех комбинаций переменных – очень сложный и длительный процесс.

Чтобы построить несколько попарных двумерных распределений в наборе данных, можно использовать функцию **pairplot()**.

Синтаксис функции pairplot()

```
seaborn.pairplot(data,…)
```

параметры:

data – dataframe;

hue – переменная для отображения разных категорий данных разными цветами;

palette – набор цветов для отображения переменной;

diag_kind – тип графика для диагонального расположения;

kind – тип графика для внедиагонального расположения.

Значения которые могут принимать **diag_kind** и **kind** – {`'scatter'`, `'reg'`, `'hist'`, `'kde'`}.

Вышеупомянутые – часто используемые параметры.

За исключением параметра «`data`», все остальные параметры являются необязательными.

Пример 19.

```
df = sb.load_dataset('iris')
sb.set_style("ticks")
sb.pairplot(df, hue = 'species', diag_kind = "kde", kind = "scat
ter", palette = "husl")
plt.show()
```

Понаблюдайте за изменениями в каждом рисунке. Графики представлены в матричном формате, где имя строки представляет ось x, а имя столбца – ось y.

Диагональные графики представляют собой графики плотности ядра, другие графики представляют собой графики рассеяния.

Пример 20. По тем же данным постройте другую матрицу точечной диаграммы.

```
iris = sb.load_dataset("iris")
ax = sb.pairplot(iris)
plt.show()
```

Опишите различие в изображениях рисунка 19 и 20.

6.6. Категориальные точечные диаграммы

В предыдущих примерах мы узнали о диаграммах, которые используются для анализа непрерывных переменных. Эти графики не подходят, когда исследуемая переменная является категориальной.

Когда одна или обе исследуемые переменные являются категориальными, необходимо использовать графики `stripplot()`, `swarmplot()` и т.д.

`stripplot()` используется, когда одна из изучаемых переменных является категориальной. Он представляет данные в отсортированном порядке по любой из осей.

Пример 21.

```
df = sb.load_dataset('iris')
sb.stripplot(x = "species", y = "petal_length", data = df, jitte
r = False)
plt.show()
```

Рассмотрите график.

На полученном графике ясно видно разницу в **длине лепестка** у каждого вида. Но главная проблема с полученным графиком рассеяния состоит в том, что точки на графике рассеяния перекрываются.

Для обработки такого сценария можно используем параметр «`jitter`».

`jitter` – параметр, который будет регулировать позиции точек вдоль категориальной оси.

Пример 22.

```
df = sb.load_dataset('iris')
sb.stripplot(x = "species", y = "petal_length", data = df, jitte
r = True)
plt.show()
```

На полученном рисунке распределение точек видно лучше.

Swarmplot ()

Другой вариант, который можно использовать в качестве альтернативы `jitter`, – это функция **swarmplot()**, которая позиционирует каждую точку графика рассеяния на категориальной оси и таким образом избегает перекрывающихся точек.

Пример 23.

```
df = sb.load_dataset('iris')
sb.swarmplot(x = "species", y = "petal_length", data = df)
plt.show()
```

Приведите вид полученного изображния.

6.7. Графики для статистических распределений

С гистограммами мы уже знакомы. В seaborn построение гистограмм реализуется функцией `barplot()`:

Пример 24.

```
df = sb.load_dataset('iris')
sb. barplot(x = "species", y = "petal_length", data = df)
plt.show()
```

Приведите рисунок с гистограммой.

Коробка с усами

Boxplot – это удобный способ визуализации распределения данных по квартилям.

Коробчатые диаграммы обычно имеют вертикальные линии, идущие от коробок, которые называются усами. Эти усы указывают на изменчивость за пределами верхнего и нижнего квартилей, поэтому график boxplot также называются диаграммой «**коробка с усами**». Любые выбросы в данных представлены в виде отдельных точек.

Пример 25. а) По тем же данным постройте коробчатую диаграмму

```
df = sb.load_dataset('iris')
sb. boxplot(x = "species", y = "petal_length", data = df)
plt.show()
```

Примечание. Полоса посередине каждой коробки – это медиана. Усы, исходящие из каждой коробки, указывают доверительный интервал. Точки на графике указывают на выброс.

Укажите примерно величины медиан для каждого типа ириса и концы доверительных интервалов для параметра `"petal_length"`.

б) Постройте boxplot с другим расположением коробок

```
df = sb.load_dataset('iris')
sb.boxplot(data = df, orient = "h")
plt.show()
```

Сюжеты для скрипки

Графики для скрипки представляют собой комбинацию области, похожей на скрипку, с оценками плотности ядра. Таким образом, эти графики легко анализировать и понимать распределение данных.

Давайте воспользуемся параметром `total_bill` набора данных tips. Этот набор данных содержит информацию, касающуюся клиентов в ресторане.

Пример 26.

```
df = sb.load_dataset('tips')
sb.violinplot(x = "day", y = "total_bill", data=df)
plt.show()
```

Примечание. Значения квартилей и усов из коробочного графика показаны внутри скрипки. Поскольку в скрипке используется KDE, более широкая часть скрипки указывает на более высокую плотность, а узкая область представляет относительно более низкую плотность. Межквартильный диапазон в коробчатом графике и более высокая плотность в kde попадают в одну и ту же область каждой категории скрипичного сюжета.

На полученном графике показано распределение `total_bill` по четырем дням недели. Если мы хотим увидеть, как параметр `total_bill` ведет себя по отношению к полу, давайте рассмотрим его в следующем примере.

Пример 27.

```
df = sb.load_dataset('tips')
sb.violinplot(x = "day", y = "total_bill", hue = 'sex', data = d
f)
plt.show()
```

Рассмотрите рисунок и сформулируйте отличие в посещении ресторана мужчинами и женщинами.

==========

В большинстве ситуаций мы имеем дело с оценками всего распределения данных. Для оценки центральной тенденции нам нужен конкретный способ анализа распределения. Среднее значение и медиана являются часто используемыми методами для оценки центральной тенденции распределения.

Рассмотрим графики, по которым можно оценить центральную тенденцию распределения.

Barplot() – гистограмма представляет собой оценку центральной тенденции. Для анализа будем использовать набор данных «Титаник».

Пример 28. Построим гистограмму среднего числа выживших мужчин и женщин в каждом классе кают.

```
df = sb.load_dataset('titanic')
sb.barplot(x = "sex", y = "survived", hue = "class", data = df)
plt.show()
```

Что следует из рисунка: 1) сравните число выживших женщин и мужчин; 2) сравните число выживших по классам кают.

Гистограмма позволяет оценить количество наблюдений в каждой категории. Для этого можно использовать функцию **countplot().**

Пример 29.

```
df = sb.load_dataset('titanic')
sb.countplot(x = "class", data = df, palette = "Blues")
plt.show()
```

Какой вывод можно сделать о количество пассажиров в первом, втором и третьем классе?.

Графики с точками

Графики с точками служат той же цели, что и столбчатые. Вместо столбика значение оценки представляется точкой на определенной высоте оси.

Пример 30.

```
df = sb.load_dataset('titanic')
sb.pointplot(x = "sex", y = "survived", hue = "class", data = df
)
plt.show()
```

Сравните информацию, выведенную в примере 28 и 30.

6.8. Функции для изображения моделей линейной регрессии

В большинстве случаев мы используем наборы данных, которые содержат несколько количественных переменных, и цель анализа часто состоит в том, чтобы связать эти переменные друг с другом. Это можно сделать через линии регрессии.

При построении регрессионных моделей мы часто проверяем **мультиколлинеарность,** т.е. видеть корреляция между всеми комбинациями непрерывных переменных, и предпринимаем необходимые действия для удаления мультиколлинеарности, если она существует.

В Seaborn есть две основные функции для визуализации линейных отношений, определенных посредством регрессии. Этими функциями являются **regplot()** и **lmplot()**.

Функция **regplot()** принимает переменные x и y в различных форматах, включая простые числовые массивы, объекты серии pandas или как ссылки на переменные в DataFrame pandas.

У функции **lmplot** параметр `data` выступает в качестве обязательного, а переменные x и y должны быть указаны в виде строк. Такой формат данных называется «длинные данные».

Пример 31. Построим regplot и затем lmplot по одним и тем же данным.

```
df = sb.load_dataset('tips')
sb.regplot(x = "total_bill", y = "tip", data = df)
sb.lmplot(x = "total_bill", y = "tip", data = df)
plt.show()
```

Сравните рисунки и укажите разницу в изображениях.

Пример 32. Построим линейную регрессию, когда одна из переменных принимает дискретные значения.

```
df = sb.load_dataset('tips')
sb.lmplot(x = "size", y = "tip", data = df)
plt.show()
```

Опишите расположение данных и линии регрессии.

==========

Для некоторых «хороших» данных модель линейной регрессии подбирается достаточно просто и хорошо описывает тенденцию в данных. Но в большинстве случаев данные являются нелинейными, и подбор хорошей модели линии регрессии является нетривиальной задачей.

В следующих примерах будем использовать набор данных anscombe для построения графиков регрессии.

Пример 33.

```
df = sb.load_dataset('anscombe')
sb.lmplot(x="x", y="y", data=df.query("dataset == 'I'"))
plt.show()
```

По вашему мнению, данные хорошо подходят для модели линейной регрессии?

Пример 34.

```
df = sb.load_dataset('anscombe')
sb.lmplot(x = "x", y = "y", data = df.query("dataset == 'II'"))
plt.show()
```

Оцените полученное изображение: а) хорошо ли ложатся точки на линию регрессии? б) большое ли отклонение точек данных от линии

регрессии? в) какой функцией, по вашему мнению, может быть описана регрессионная модель?

БЛАГОДАРЮ ЗА ПРОДЕЛАННУЮ РАБОТУ!!!