Лабораторная работа

Упрощенная визуализация с помощью Seaborn

Цель работы:
- узнать почему Seaborn лучше, чем Matplotlib;
- научиться эффективно проектировать визуально привлекательные участки;
- научиться создавать проницательные фигуры.

Теоретическая часть

В отличие от *Matplotlib*, *Seaborn* не является самостоятельной библиотекой *Python*. Она построена на базе *Matplotlib* и обеспечивает более высокий уровень абстракции для создания визуально привлекательных статистических визуализаций. Отличительной особенностью *Seaborn* является возможность интеграции с *DataFrames* из библиотеки *pandas*.

С помощью *Seaborn* мы пытаемся сделать визуализацию центральной частью исследования и понимания данных. Внутри *Se*aborn работает на *DataFrames* и массивах, которые содержат полный набор данных. Это позволяет ему выполнять семантическое картирование и статистические агрегации, которые необходимы для отображения информативных визуализаций. *Seaborn* также может использоваться исключительно для изменения стиля и внешнего вида визуализаций *Matplotlib*.

Наиболее яркими чертами Seaborn являются следующие:

- Красивые нестандартные графикы с разными темами
- Встроенные цветовые палитры, которые могут быть использованы для выявления шаблонов в наборе данных
- Dataset-ориентированный интерфейс
- Высокоуровневая абстракция, позволяющая создавать сложные визуализации.

Преимущества Seaborn

Seaborn построен на вершине *Matplotlib*, а также обращается к некоторым из основных болевых точек работы с *Matplotlib*.

Работа с *DataFrames* с помощью *Matplotlib* добавляет некоторые неудобные накладные расходы. Например: простое исследование вашего набора данных может занять много времени, так как для построения графиков данных из *DataFrames* с помощью *Matplotlib* вам понадобятся некоторые дополнительные операции с данными.

Однако *Seaborn* построен для работы с *DataFrames* и полными массивами данных, что делает этот процесс более простым. Внутренне *Seaborn* выполняет необходимые семантические сопоставления и

статистическую агрегировку для построения информативных графиков. Ниже приведен пример построения графиков с использованием библиотеки *Seaborn*:

```
import seaborn as sns

import pandas as pd

sns.set(style="ticks")

data = pd.read_csv("data/salary.csv")

sns.relplot(x="Salary", y="Age", hue="Education",

style="Education", col="Gender", data=data)
```

Это создает следующий график:

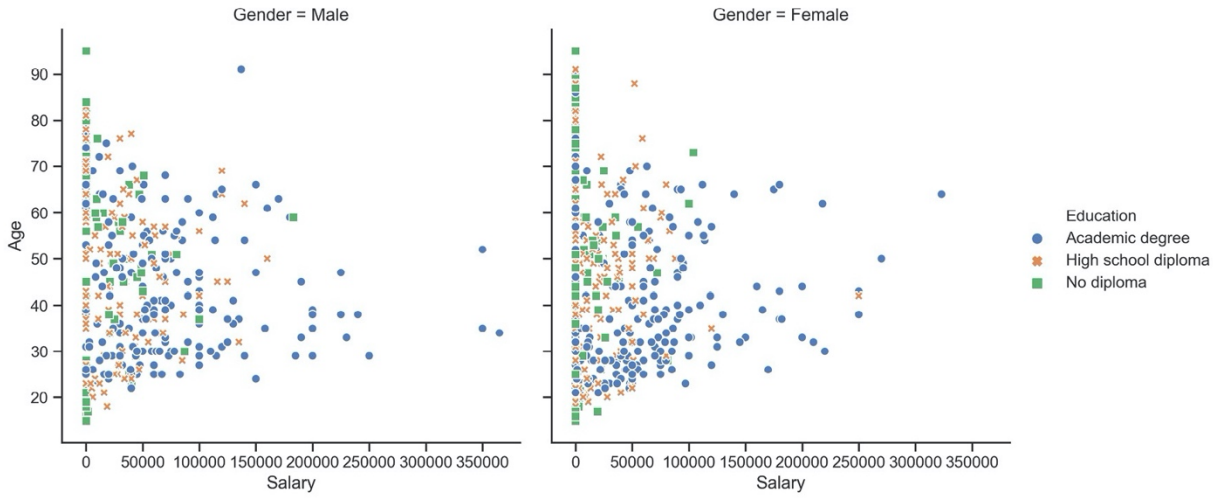

Рисунок 1: График отношений Seaborn

За кулисами Seaborn использует Matplotlib для рисования графиков. Несмотря на то, что многие задачи могут быть выполнены только с помощью Seaborn, дальнейшая настройка может потребовать использования Matplotlib. Мы предоставили только имена переменных в наборе данных и роли, которые они играют в графике. В отличие от Matplotlib, нет необходимости переводить переменные в параметры визуализации.

Другими "болевыми точками" являются параметры и конфигурации Matplotlib по умолчанию. Параметры по умолчанию в Seaborn обеспечивают лучшую визуализацию без дополнительных настроек. Мы подробно рассмотрим эти параметры по умолчанию в следующих темах.

Для пользователей, которые уже знакомы с Matplotlib, расширение в Seaborn является тривиальным, так как основные концепции в основном похожи.

Эстетика контрольного рисунка

Как мы уже упоминали, *Matplotlib* очень настраиваемый. Но это также приводит к тому, что сложно понять, какие настройки нужно настроить, чтобы получить визуально привлекательный график. В отличие от этого *Seaborn* предоставляет несколько настраиваемых тем и высокоуровневый интерфейс для управления внешним видом фигур *Matplotlib*.

Следующий фрагмент кода создает простой график в *Matplotlib*:

```
%matplotlib inline

import matplotlib.pyplot as plt

plt.figure()

x1 = [10, 20, 5, 40, 8]

x2 = [30, 43, 9, 7, 20]

plt.plot(x1, label=‘Group A’)

plt.plot(x2, label=‘Group B’)

plt.legend()

plt.show()
```

Вот как выглядит график с параметрами Matplotlib по умолчанию:

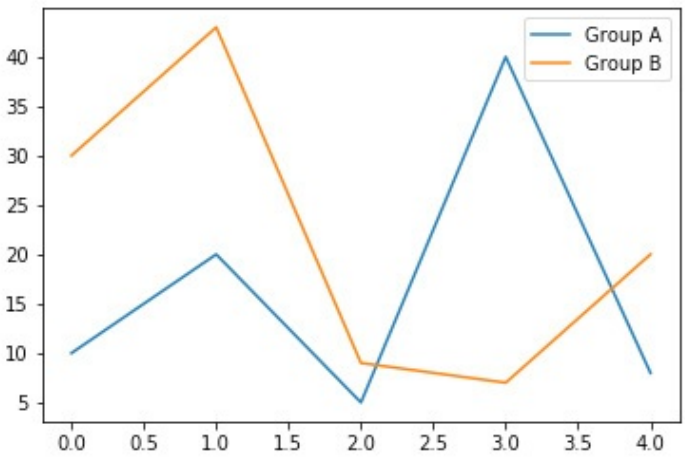

Рисунок 2: Линейный график Matplotlib

Чтобы переключиться на настройки Seaborn по умолчанию, просто вызовите функцию set():

```
%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

plt.figure()

x1 = [10, 20, 5, 40, 8]

x2 = [30, 43, 9, 7, 20]

plt.plot(x1, label='Group A')

plt.plot(x2, label='Group B')

plt.legend()

plt.show()
```

Получаем вот такой график:

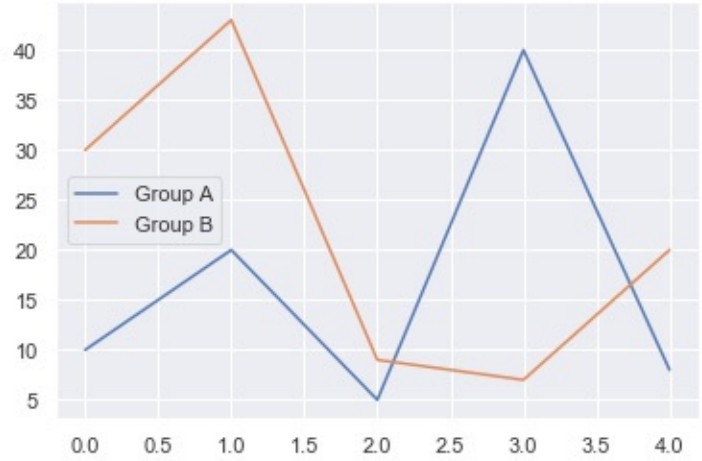

Рисунок 3: Линейный график Seaborn

Seaborn разделяет параметры Matplotlib на две группы. Первая группа содержит параметры для эстетики графика, вторая группа масштабирует различные элементы фигуры так, чтобы ее можно было легко использовать в различных контекстах, таких как визуализации, которые используются для презентаций, постеров и так далее.

Фигурные стили Seaborn

Чтобы контролировать стиль, Seaborn предоставляет два метода: **et_style(style, [rc])** и **axes_style(style, [rc])**.

seaborn.set_style(style, [rc]) задает эстетический стиль графиков.

Параметры:

- **style:** Словарь параметров или имя одного из следующих предварительно сконфигурированных наборов: darkgrid, whitegrid, dark, white или ticks
- **rc (опционально):** Отображение параметров для переопределения значений в заданных словарях стиля Seaborn

Пример:

```
%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style(“whitegrid”)

plt.figure()

x1 = [10, 20, 5, 40, 8]

x2 = [30, 43, 9, 7, 20]

plt.plot(x1, label=‘Group A’)

plt.plot(x2, label=‘Group B’)

plt.legend()

plt.show()
```

Это приводит к следующему графику:

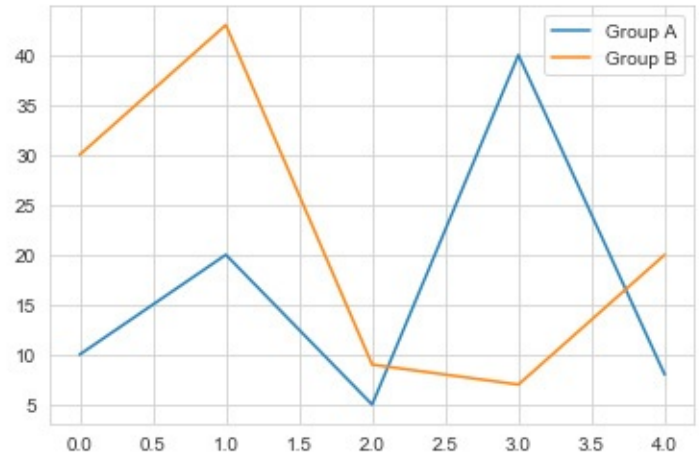

Рисунок 4: График Seaborn линейный график в стиле белой сетки

seaborn.axes_style(style, [rc]) возвращает словарь параметров для эстетического стиля сюжетов. Функция может быть использована в операционном задании для временного изменения стилевых параметров.

Параметры:

- **style:** словарь параметров или имя одного из следующих предварительно сконфигурированных наборов: darkgrid, whitegrid, dark, white или ticks
- **rc (опционально):** Отображение параметров для переопределения значений в заданных словарях стиля Seaborn.

Пример:

```
%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

plt.figure()

x1 = [10, 20, 5, 40, 8]

x2 = [30, 43, 9, 7, 20]
```

```
with sns.axes_style('dark'):

plt.plot(x1, label='Group A')

plt.plot(x2, label='Group B')

plt.legend()

plt.show()
```

Эстетика меняется только временно. Результат показан на следующей диаграмме:

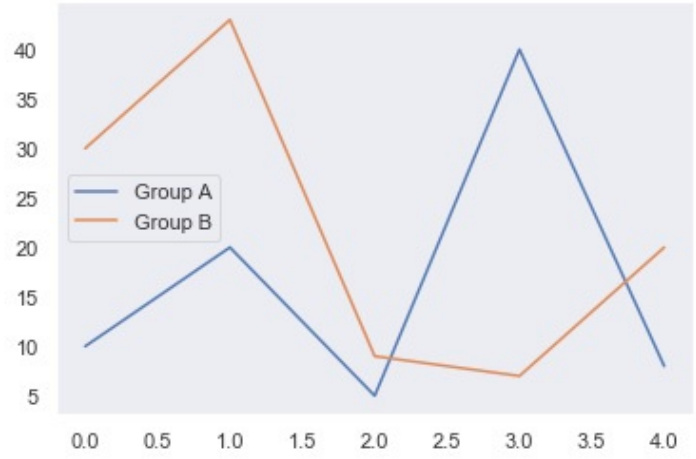

Рисунок 5: Seaborn линейный график с темными осями

Для дальнейшей настройки вы можете передать словарь параметров в аргумент **rc**. Вы можете переопределить только те параметры, которые являются частью определения стиля.

Удаление осевых шипов

Иногда бывает желательно удалить верхнюю и правую оси шипов.

seaborn.despine(fig=None, ax=None, top=True, right=True, left=False, bottom=False, offset=None, trim=False) удалить верхний и правый шип с графика.

Следующий код помогает удалить шипы осей:

```
%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns
```

```
sns.set_style(“white”)

plt.figure()

x1 = [10, 20, 5, 40, 8]

x2 = [30, 43, 9, 7, 20]

plt.plot(x1, label=‘Group A’)

plt.plot(x2, label=‘Group B’)

sns.despine()

plt.legend()

plt.show()
```

Это приводит к следующему графику:

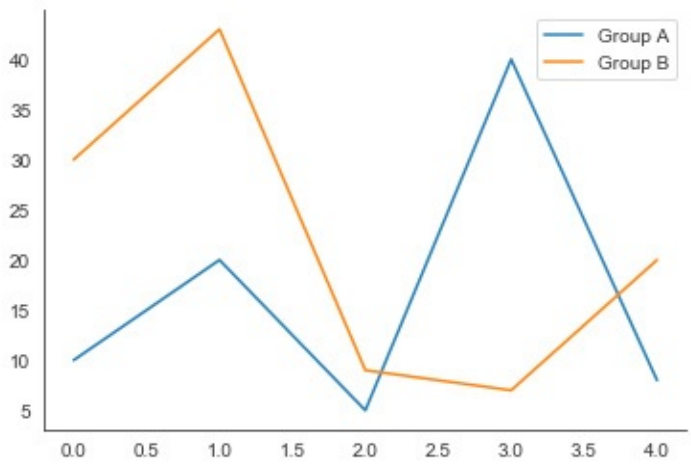

Рисунок 6: Линейный график Seaborn без осей

Контексты

Отдельный набор параметров управляет масштабом элементов графика. Это удобный способ использовать один и тот же код для создания графиков, которые подходят для использования в контекстах, где необходимы большие или маленькие графики. Для управления контекстом можно использовать две функции.

seaborn.set_context(context, [font_scale], [rc]) устанавливает параметры контекста для построения графиков. Это не меняет общий стиль графика, но влияет на такие вещи, как размер меток, линий и так далее. Базовым

контекстом является ноутбук, а другими контекстами являются бумага, лекция и плакат, которые являются версиями параметров ноутбука, масштабированных по 0.8, 1.3 и 1.6 соответственно.

Вот параметры:

- **Context:** Словарь параметров или имя одного из следующих предварительно сконфигурированных наборов: бумага, блокнот, разговор или плакат

- **font_scale (необязательно):** Масштабный коэффициент для независимого масштабирования размера элементов шрифта

- **rc (опционально):** Карта параметров

Следующий код помогает установить контекст:

```
%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_context(“poster”)

plt.figure()

x1 = [10, 20, 5, 40, 8]

x2 = [30, 43, 9, 7, 20]

plt.plot(x1, label=‘Group A’)

plt.plot(x2, label=‘Group B’)

plt.legend()

plt.show()
```

Предыдущий код генерирует следующий вывод:

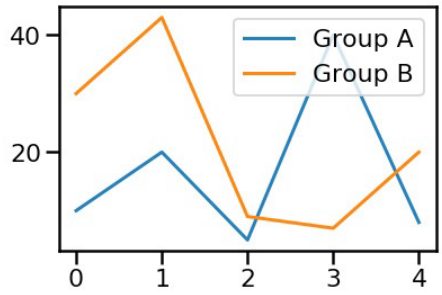

Рисунок 7: Линейный график Seaborn с контекстным меню

seaborn.plotting_context(context, [font_scale], [rc]) возвращает словарь параметров для масштабирования элементов рисунка. Эта функция может быть использована с оператором для временного изменения параметров контекста.

Вот параметры:

- **contex**: Словарь параметров или имя одного из следующих предварительно сконфигурированных наборов: бумага, блокнот, разговор или плакат

- **font_scale (необязательно):** Масштабный коэффициент для независимого масштабирования размера элементов шрифта

- **rc (опционально):** Отображение параметров для переопределения значений в заданных контекстных словарях Seaborn.

Практическая часть

Задание: Сравнение баллов IQ для различных тестовых групп с помощью блок-схемы.

В этом упражнении мы будем сравнивать оценки IQ между различными тестовыми группами, используя блок-график библиотеки Seaborn:

1. Давайте сравним показатели IQ среди различных тестовых групп с помощью библиотеки Seaborn: Для реализации данного задания откройте из папки Lesson04 блокнот с описанием **activity20_solution.ipynb** Jupyter. Перейдите по пути к этому файлу и наберите в командной строке терминала: jupyter-lab.2. Получить доступ к данным каждой группы в столбце, преобразовать их в список и назначить этот список переменным каждой соответствующей группы.

2. Импортируйте необходимые модули и включите их в JupyterNotebook.

```
%matplotlib inline

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

3. Используйте функцию pandas **read_csv()** для чтения данных, расположенных в папке данных:

```
mydata = pd.read_csv("./data/scores.csv")
```

4. Доступ к данным каждой тестовой группы в столбце .Преобразовать их в список с помощью метода tolist(). После преобразования данных каждой тестовой группы в список, присвойте этот список переменным каждой соответствующей тестовой группы:

```
group_a = mydata[mydata.columns[0]].tolist()

group_b = mydata[mydata.columns[1]].tolist()

group_c = mydata[mydata.columns[2]].tolist()

group_d = mydata[mydata.columns[3]].tolist()
```

5. Выведите переменные каждой группы на печать, чтобы проверить, преобразуются ли данные внутри нее в список. Это можно сделать с помощью функции print():

```
print(group_a)
```

Значения данных группы A показаны на следующем рисунке:

```
[118, 103, 125, 107, 111, 96, 104, 97, 96, 114, 96, 75, 114, 107, 87, 117, 117, 114, 117, 112, 107, 133, 94, 91, 118, 110, 117,
86, 143, 83, 106, 86, 98, 126, 109, 91, 112, 120, 108, 111, 107, 98, 89, 113, 117, 81, 113, 112, 84, 115, 96, 93, 128, 115, 13
8, 121, 87, 112, 110, 79, 100, 84, 115, 93, 108, 130, 107, 106, 106, 101, 117, 93, 94, 103, 112, 98, 103, 70, 139, 94, 110, 10
5, 122, 94, 94, 105, 129, 110, 112, 97, 109, 121, 106, 118, 131, 88, 122, 125, 93, 78]
```

Рисунок 8.1 Значения группы A

print(group_b)

Значения данных группы B показаны на следующем рисунке:

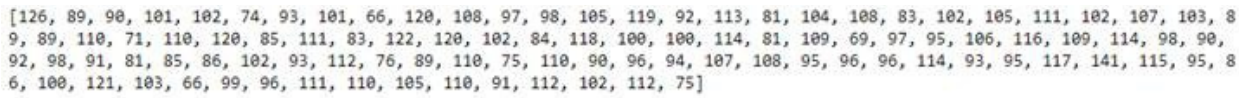

Рисунок 8.2 Значения группы B

print(group_c)

Значения данных группы C показаны на следующем рисунке:

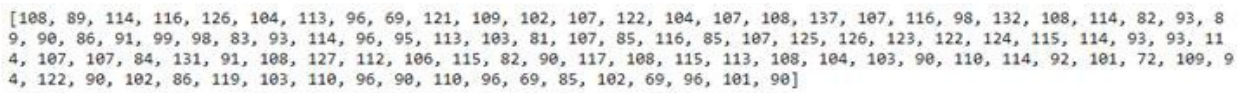

Рисунок 8.2 Значения группы C

print(group_d)

Значения данных группы D показаны на следующем рисунке:

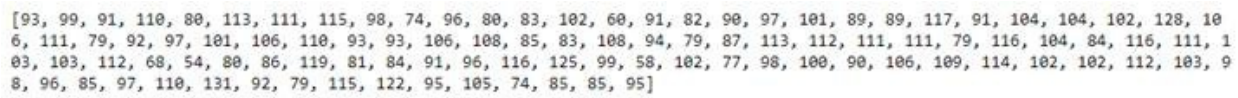

Рисунок 8.2 Значения группы D

6. После того, как у нас есть данные для каждой тестовой группы, нам нужно построить DataFrame из этих данных. Это можно сделать с помощью функции pd.DataFrame(), которую предоставляют pandas.

data = pd.DataFrame({‘Groups’: [‘Group A’] * len(group_a) + [‘Group B’] * len(group_b) + [‘Group C’] * len(group_c) + [‘Group D’] * len(group_d), ‘IQ score’: group_a + group_b + group_c + group_d})

7. Теперь, когда у нас есть DataFrame, нам нужно создать boxplot с помощью функции boxplot(), которая предоставляется Seaborn. В этой функции нам нужно указать заголовки для обеих осей вместе с DataFrame, который мы используем.Заголовок для оси x будет Groups, а заголовок для оси y - IQ. Что касается DataFrame, мы будем передавать данные в качестве параметра. Здесь данные - это DataFrame, который мы получили на предыдущем шаге.

plt.figure(dpi=150)

Set style

sns.set_style(‘whitegrid’)

```
# Create boxplot

sns.boxplot('Groups', 'IQ score', data=data)

# Despine

sns.despine(left=True, right=True, top=True)

# Add title

plt.title('IQ scores for different test groups')

# Show plot

plt.show()
```

Функция despine() помогает удалить верхний и правый позвоночники из графика. Здесь мы также удалили левый позвоночник. Используя функцию title(), мы установили заголовок для нашего графика. Функция show() помогает визуализировать график.

Из рисунка 8 можно сделать вывод, что при использовании бокс-графика IQ группы A лучше, чем у других групп.

8. После выполнения предыдущих шагов, окончательный вывод должен быть следующим:

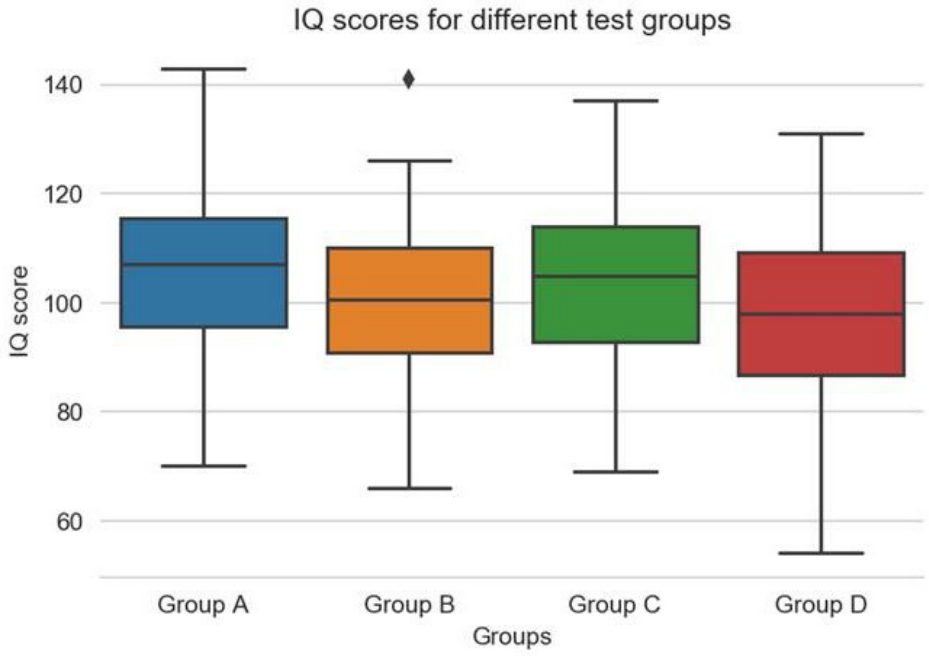

Рисунок 8: Оценки IQ групп

Цветовые палитры

Цвет является очень важным фактором для вашей визуализации. Цвет может выявить закономерности в данных, если используется эффективно, или скрыть закономерности, если используется плохо. Seaborn облегчает выбор и использование цветовых палитр, подходящих для Вашей задачи. Функция color_palette() предоставляет интерфейс для многих возможных способов генерации цветов.

seaborn.color_palette([palette], [n_colors], [desat]) возвращает список цветов, таким образом определяя палитру цветов.

Вот параметры:

- palette (необязательно): Имя палитры или Нет для возврата текущей палитры.
- n_colors (необязательно): Количество цветов в палитре. Если указанное количество цветов больше, чем количество цветов в палитре, то цвета будут циклически повторяться.
- desat (опционально): Доля каждого цвета в процентах.

С помощью функции set_palette() можно установить палитру для всех графиков. Эта функция принимает те же аргументы, что и color_palette(). В следующих разделах мы объясним, как палитры цветов делятся на различные группы.

Категорические цветовые палитры

Категорические палитры лучше всего подходят для различения дискретных данных, которые не имеют встроенного упорядочения. В Seaborn есть шесть тем по умолчанию: глубокая, приглушенная, яркая, пастельная, темная и слепая. Код и вывод для каждой темы приводится в следующем коде:

Практическая часть

```
import seaborn as sns

palette1 = sns.color_palette(“deep”)

sns.palplot(palette1)
```

На следующем рисунке показан вывод предыдущего кода:

Рисунок 9

palette2 = sns.color_palette(“muted”)

sns.palplot(palette2)

На следующем рисунке показан вывод предыдущего кода:

Рисунок 10

На следующем рисунке показана яркая цветовая палитра:

palette3 = sns.color_palette(“bright”)

sns.palplot(palette3)

Рисунок 11

На следующем рисунке показана пастельная цветовая палитра:

palette4 = sns.color_palette(“pastel”)

sns.palplot(palette4)

Рисунок 12

На следующем рисунке показана темная цветовая палитра:

palette5 = sns.color_palette(“dark”)

sns.palplot(palette5)

Рисунок 13

```
palette6 = sns.color_palette(“colorblind”)

sns.palplot(palette6)
```

На следующем рисунке показан вывод предыдущего кода:

Рисунок 14

Последовательные цветовые палитры

Последовательные цветовые палитры подходят, когда данные варьируются от относительно низких или неинтересных значений до относительно высоких или интересных значений. Следующие фрагменты кода вместе с их соответствующими выводами дают нам лучшее представление о последовательных цветовых палитрах:

```
custom_palette2 = sns.light_palette(“brown”)

sns.palplot(custom_palette2)
```

На следующем рисунке показан вывод предыдущего кода:

Рисунок 15

Предыдущую палитру можно также перевернуть, установив в следующем коде обратный параметр в значение True:

```
custom_palette3 = sns.light_palette(“brown”, reverse=True)

sns.palplot(custom_palette3)
```

На следующем рисунке показан вывод предыдущего кода:

Рисунок 16

Различающиеся цветовые палитры

Различающиеся цветовые палитры используются для данных, которые состоят из четко определенной средней точки. Упор делается на высокие и низкие значения. Например: если вы планируете какие-либо изменения популяции для определенного региона от некоторой базовой популяции, то лучше всего использовать расходящиеся цветовые карты, чтобы показать относительное увеличение и уменьшение популяции. Следующий фрагмент кода и вывод обеспечивает лучшее понимание расходящегося графика, в котором мы используем теплый шаблон, встроенный в Matplotlib:

```
custom_palette4 = sns.color_palette(“coolwarm”, 7)

sns.palplot(custom_palette4)
```

На следующем рисунке показан вывод предыдущего кода:

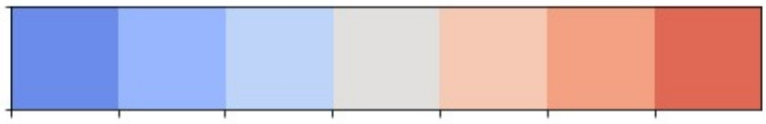

Рисунок 17

Вы можете использовать функцию diverging_palette() для создания пользовательских расходящихся палитр. В качестве параметров можно передавать два оттенка в градусах, а также общее количество палитр. Следующие фрагмент кода и вывод обеспечивают лучшее понимание:

```
custom_palette5 = sns.diverging_palette(440, 40, n=7)

sns.palplot(custom_palette5)
```

На следующем рисунке показан вывод предыдущего кода:

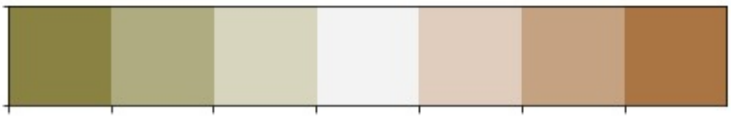

Рисунок 18

Задание для самостоятельной работы №1

Использование тепловых карт для поиска шаблонов в данных о пассажирах рейса

В этом задании мы будем использовать тепловую карту, чтобы найти закономерности в данных о пассажирах рейса:

1. С помощью pandas считывать данные, расположенные в подкаталоге. Данный набор данных содержит ежемесячные данные по пассажирам рейсов за период с 2001 по 2012 год.

2. Используйте тепловую карту для визуализации данных.

3. Используйте собственную цветную карту. Убедитесь, что наименьшее значение - самый темный цвет, а наибольшее - самый яркий.

4. После выполнения предыдущих шагов, ожидаемые результаты должны быть следующими:

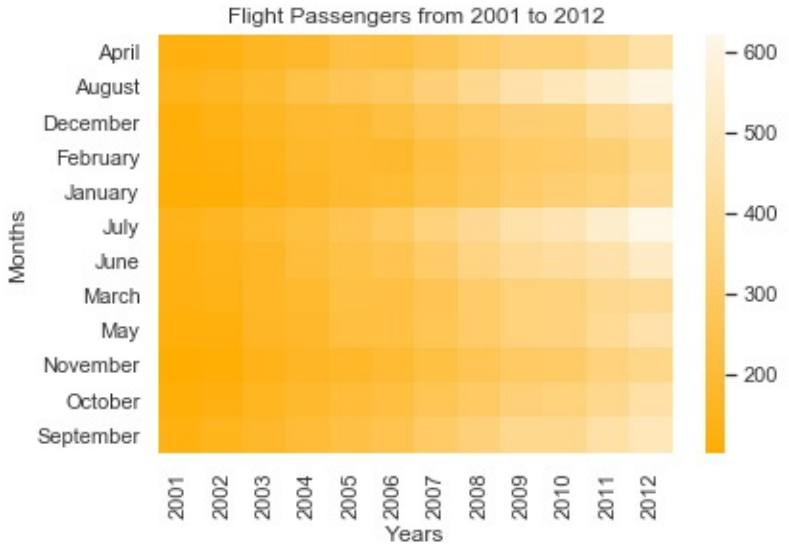

Рисунок 19 Тепловая карта данных о бортовых пассажирах

Интересные диаграммы в Seaborn

В предыдущей главе мы обсуждали различные сюжеты в Matplotlib, но осталось еще несколько визуализаций, которые мы хотим обсудить.

Столбчатые диаграммы

В последней главе мы уже объясняли, как создавать гистограммы с помощью Matplotlib. Создание барных графиков с подгруппами было довольно утомительным, но Seaborn предлагает очень удобный способ создания различных барных графиков. Они также могут быть использованы в Seaborn, чтобы представлять оценки центральной тенденции с высотой каждого прямоугольника и указывать на неопределенность вокруг этой оценки, используя столбики ошибок.

Следующий пример дает вам хорошее представление о том, как это работает:

```
import pandas as pd
```

```
import seaborn as sns

data = pd.read_csv(“data/salary.csv”)

sns.set(style=“whitegrid”)

sns.barplot(x=“Education”, y=“Salary”, hue=“District”, data=data)
```

Результат показан на следующей диаграмме:

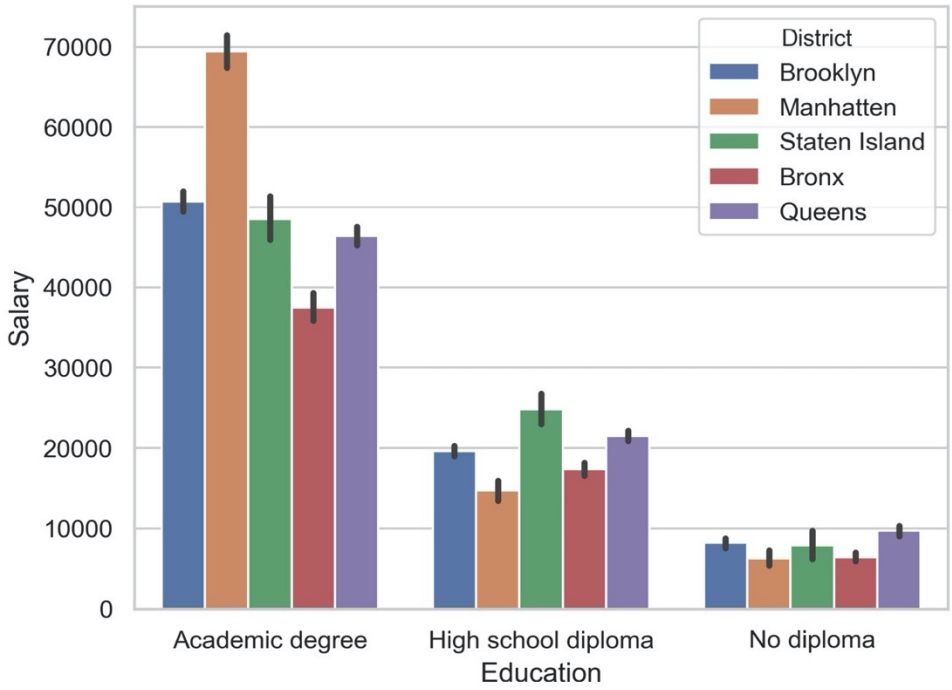

Рисунок 20 Столбчатая диаграмма Seaborn

Задание для самостоятельной работы №2

Пересмотр сравнения фильмов

В этом упражнении мы будем использовать сюжет бара для сравнения баллов фильмов. Вам будут предложены пять фильмов с оценкой "Гнилые помидоры". Томатометр - это процент от одобренных критиков Томатометра, которые дали положительный отзыв на фильм. Оценка зрительских симпатий - это процент пользователей, которые дали оценку 3,5 или выше из 5. Сравните эти два балла среди пяти фильмов:

1. Используйте pandas для чтения данных, расположенных в подпапках.
2. преобразовать данные в удобный для использования формат для функции построения гистограммы Seaborn.

3.использовать Seaborn для создания визуально привлекательного графика баров, который сравнивает две оценки для всех пяти фильмов.

4.после выполнения предыдущих шагов, ожидаемый результат должен выглядеть следующим образом:

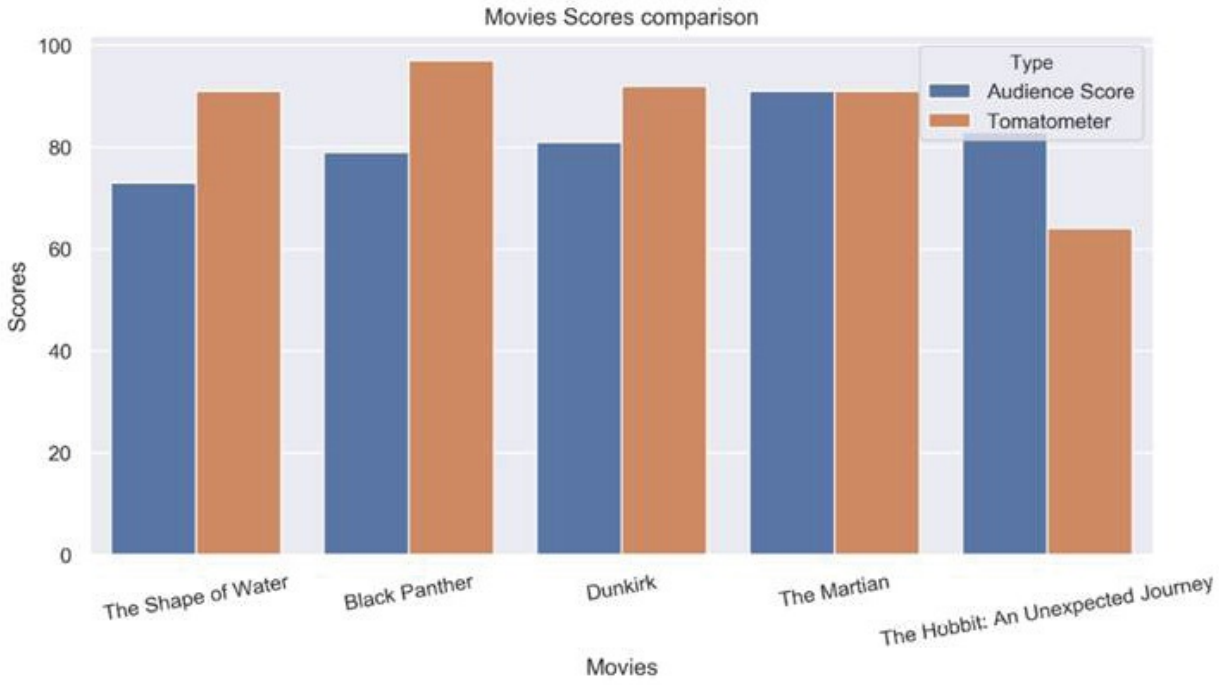

Рисунок 21 Сравнение кинобиографий

Диаграммы регрессии

Многие наборы данных содержат множество количественных переменных, и цель состоит в том, чтобы найти связь между этими переменными. Ранее мы упоминали о нескольких функциях, которые показывают совместное распределение двух переменных. Может оказаться полезным оценить отношения между двумя переменными. В данной теме мы рассмотрим только линейную регрессию, однако Seaborn при необходимости предоставляет более широкий спектр функциональных возможностей регрессии.

Для визуализации линейных зависимостей, определяемых с помощью линейной регрессии, Seaborn предлагает функцию regplot(). Следующий фрагмент кода дает простой пример:

```
import numpy as np

import seaborn as sns

x = np.arange(100)

y = x + np.random.normal(0, 5, size=100)
```

```
sns.regplot(x, y)
```

Функция regplot() рисует график рассеяния, линию регрессии и доверительный интервал 95% для этой регрессии, как показано на следующем рисунке:

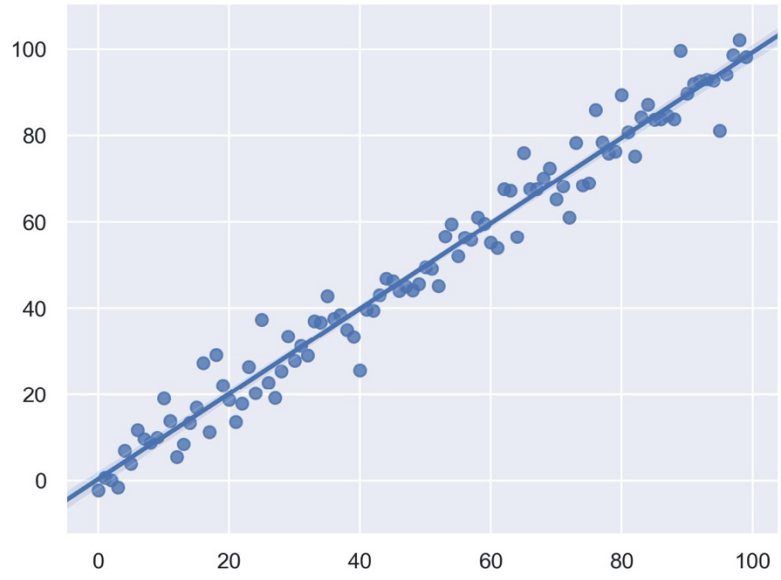

Рисунок 22 Seaborn диаграмма регрессии

Практическая часть

Линейная регрессия

Визуализируем линейную зависимость между максимальной продолжительностью жизни и массой тела в регрессионном графике с помощью функции regplot(), предоставляемой библиотекой Seaborn:

1. Для реализации этой активности откройте из папки Lesson04 JupyterNotebook активности25_solution.ipynb. Перейдите по пути к этому файлу и наберите в командной строке терминала: jupyter-lab.

2. Импортируйте необходимые модули и включите в JupyterNotebook планирование:

```
%matplotlib inline

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

3. Используйте функцию read_csv() pandas для чтения данных, расположенных в папке данных:

```
mydata = pd.read_csv("./data/anage_data.csv")
```

4. Фильтруйте данные так, чтобы в конечном итоге получить образцы, содержащие массу тела и максимальный срок службы. Рассматривайте только образцы для класса Маммалия и массу тела ниже 200 000. Эту препроцессию можно увидеть в следующем коде:

```
longevity = 'Maximum longevity (yrs)'
    mass = 'Body mass (g)'
    data = mydata[mydata['Class'] == 'Mammalia']

data = data[np.isfinite(data[longevity]) & np.isfinite(data[mass]) &
(data[mass] < 200000)]
```

5. После завершения препроцессирования нам необходимо построить график данных с помощью функции regplot(), предоставляемой библиотекой Seaborn. В следующем коде мы предоставили три параметра внутри функции regplot(). Первые два параметра - масса и продолжительность жизни, при этом данные о массе тела будут показаны по оси x, а данные о максимальной продолжительности жизни - по оси y. В третьем параметре нам необходимо предоставить DataFrame, называемый данными, которые мы получили на предыдущем шаге:

```
# Create figure

sns.set()

plt.figure(figsize=(10, 6), dpi=300)

# Create scatter plot

sns.regplot(mass, longevity, data=data)

# Show plot

plt.show()
```

Можно сделать вывод, что существует линейная зависимость между массой тела и максимальной продолжительностью жизни для класса Маммалия.

Squarify

На данном этапе мы кратко поговорим о картах деревьев. Карты деревьев отображают иерархические данные в виде набора вложенных прямоугольников. Каждая группа представлена прямоугольником, площадь которого пропорциональна его значению. Используя цветовые схемы, можно представлять иерархии: группы, подгруппы и так далее. По сравнению с круговыми диаграммами, древовидные карты эффективно используют пространство. Matplotlib и Seaborn не предлагают древовидных карт, поэтому используется библиотека Squarify, построенная поверх Matplotlib. Seaborn является отличным дополнением для создания цветовых палитр.

Следующий фрагмент кода является основным примером карты дерева. Для него требуется библиотека Squarify:

Практическая часть

```
%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

import squarify

colors = sns.light_palette(“brown”, 4)

squarify.plot(sizes=[50, 25, 10, 15], label=[“Group A”, “Group B”, “Group C”, “Group D”], color=colors)

plt.axis(“off”)

plt.show()
```

Результат показан на следующей диаграмме:

Рисунок 23 Карта деревьев

Задание для самостоятельной работы №3

Просмотр водопользования

В этом упражнении мы будем использовать карту-дерево для визуализации процентного соотношения воды, используемой для различных целей:

1. С помощью pandas считывать данные, расположенные в подпапке.
2. 2. Использовать карту дерева для визуализации использования воды.
3. Показывать проценты для каждой плитки и добавлять заголовок.
4. После выполнения предыдущих шагов, ожидаемые значения должны быть следующими:

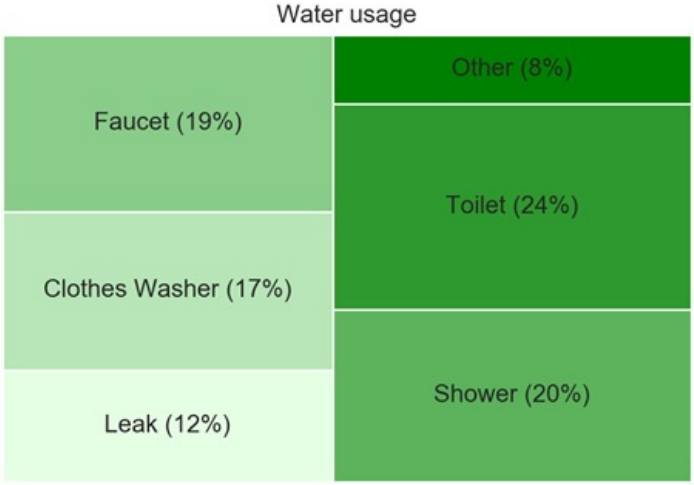

Рисунок 24 Карта деревьев использования воды

Заключение

В этой главе мы продемонстрировали, как Seaborn помогает создавать визуально привлекательные фигуры. Мы обсудили различные варианты управления эстетикой фигуры, такие как стиль фигуры, управление позвоночниками и настройка контекста визуализации. Мы подробно обсудили цветовые палитры. Далее были представлены визуализации для визуализации одномерных и двумерных распределений. Кроме того, мы обсудили FacetGrids, которые могут быть использованы для создания нескольких графиков, и регрессионные графики как способ анализа связей между двумя переменными. Наконец, мы обсудили библиотеку Squarify, которая используется для создания карт деревьев. В следующей главе мы покажем, как визуализировать геопространственные данные различными способами с помощью библиотеки Geoplotlib.