Инструменты обработки и визуализации данных

Тема 8. Визуализация с Bokeh и Plotly

Визуализация с библиотекой Bokeh

Bokeh — это библиотека для визуализации данных, предоставляющая интерактивные диаграммы и графики. Bokeh отображает свои графики с помощью HTML и JavaScript, используя современные веб-браузеры для элегантного и лаконичного построения графики с высоким уровнем интерактивности.

Особенности Bokeh:

- *Гибкость*: Bokeh может использоваться как для стандартных задач построения графиков, так и для сложных и нестандартных сценариев.
- *Производительность*: Его взаимодействие с другими популярными инструментами, такими как Pandas и Jupyter notebook, очень простое.
- *Интерактивность*: Он создает интерактивные графики, которые изменяются в зависимости от действий пользователя.
- *Мощность*: Создание визуализаций для специализированных сценариев использования может осуществляться путем добавления JavaScript.
- *Совместное использование*: Визуальные данные можно совместно использовать. Их также можно отображать в блокнотах Jupyter.
- *Открытый исходный код*: Bokeh — это проект с открытым исходным кодом.

Установка Bokeh

Bokeh можно установить как с помощью менеджера пакетов `conda`, так и с помощью `pip`.

```
conda install bokeh
```

```
pip install bokeh
```

Интерфейсы Bokeh

Bokeh прост в использовании, поскольку предоставляет простой интерфейс

для специалистов по обработке данных, которые не хотят отвлекаться на его реализацию, а также предоставляет подробный интерфейс для разработчиков и инженеров-программистов, которым может потребоваться больший контроль над Bokeh для создания более сложных функций. Для этого Bokeh использует многоуровневый подход.

Bokeh.models

Этот класс представляет собой библиотеку Python для Bokeh, содержащую классы моделей, которые обрабатывают данные JSON, созданные библиотекой JavaScript Bokeh (BokehJS). Большинство моделей очень простые, состоящие из очень небольшого количества атрибутов или не содержащие методов.

bokeh.plotting

Это интерфейс среднего уровня, предоставляющий возможности построения графиков, аналогичные Matplotlib или MATLAB. Он работает с данными, которые необходимо отобразить на графике, и создает допустимые оси, сетки и инструменты. Основной класс этого интерфейса — класс `Figure`.

Создадим простейший график:

In [1]:
```
from bokeh.io import output_notebook

output_notebook() # Activate notebook output
```

Loading BokehJS ...

In [2]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Линейный график", width=600, height=400)

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

# plotting the line graph
graph.line(x, y)

# displaying the model
show(graph)
```

В этом примере мы создали простой график с заголовком «Линейный график». При использовании Jupyter результат будет создан в новой вкладке браузера.

Аннотации и легенды

Аннотации — это дополнительная информация, такая как заголовки, легенды, стрелки и т. д., которую можно добавить к графикам. В приведенном выше примере мы уже видели, как добавить заголовки к графику. В этом разделе мы рассмотрим легенды.

Добавление легенд к графикам может помочь правильно описать и определить их, тем самым обеспечивая большую ясность. Легенды в Bokeh просты в реализации. Они могут быть простыми, автоматически сгруппированными, созданными вручную, явно индексированными, а также интерактивными.

In [3]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title="Bokeh Line Graph", width=600, height=400)

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

# plotting the 1st line graph
graph.line(x, x, legend_label="Line 1")

# plotting the 2nd line graph with a
# different color
graph.line(y, x, legend_label="Line 2",
           line_color="green")

# displaying the model
show(graph)
```

В приведенном выше примере мы построили две разные линии с легендой, которая просто указывает, какая из них линия 1, а какая — линия 2. Цвет в легенде также различается.

Настройка легенды

Легенду в Bokeh можно настроить с помощью следующих свойств.

Свойство	Описание
legend.label_text_font	изменить шрифт метки по умолчанию на указанное имя шрифта
legend.label_text_font_size	размер шрифта в пунктах
legend.location	установить метку в указанном месте.
legend.title	установить заголовок для метки легенды

legend.orientation	установить горизонтальное (по умолчанию) или вертикальное положение
legend.clicking_policy	указать, что должно происходить при щелчке по легенде

In [4]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title="Bokeh Line Graph", width=600, height=400)

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

# plotting the 1st line graph
graph.line(x, x, legend_label="Line 1")

# plotting the 2nd line graph with a
# different color
graph.line(y, x, legend_label="Line 2",
           line_color="green")

graph.legend.title = "Title of the legend"
graph.legend.location ="top_left"
graph.legend.label_text_font_size = "17pt"

# displaying the model
show(graph)
```

Построение различных типов графиков

В терминологии Bokeh глифы (glyphs) означают основные строительные блоки графиков Bokeh, такие как линии, прямоугольники, квадраты и т. д. Графики Bokeh создаются с помощью интерфейса `bokeh.plotting`, который использует стандартный набор инструментов и стилей.

Линейный график

Линейные графики используются для представления зависимости между двумя данными X и Y на разных осях. Линейный график можно создать с помощью метода `line()` модуля `plotting`.

Синтаксис:

`line(параметры)`

In [5]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
```

```
graph = figure(title = "Bokeh Line Graph", width=600, height=400)

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

# plotting the line graph
graph.line(x, y)

# displaying the model
show(graph)
```

Столбчатая диаграмма

Столбчатая диаграмма — это график, представляющий категорию данных в виде прямоугольных столбцов, длина и высота которых пропорциональны значениям, которые они представляют. Она может быть двух типов: горизонтальные и вертикальные столбцы. Каждый из них можно создать с помощью функций `hbar()` и `vbar()` интерфейса построения графиков соответственно.

Синтаксис:

```
hbar(параметры)
```

```
vbar(параметры)
```

Пример 1: Создание горизонтальной столбчатой диаграммы

In [6]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Bokeh Bar Graph", width=600, height=400)

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]

# height / thickness of the plot
height = 0.5

# plotting the bar graph
graph.hbar(x, right = y, height = height)

# displaying the model
show(graph)
```

Пример 2: Создание вертикальной столбчатой диаграммы

In [7]:
```
# importing the modules
from bokeh.plotting import figure, show
```

```
# instantiating the figure object
graph = figure(title = "Bokeh Bar Graph", width=600, height=400)

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]

# height / thickness of the plot
width = 0.5

# plotting the bar graph
graph.vbar(x, top = y, width = width)

# displaying the model
show(graph)
```

Диаграмма рассеяния

Диаграмма рассеяния — это набор точек, представляющих отдельные данные по горизонтальной и вертикальной осям. На графике значения двух переменных отображаются по осям X и Y, а расположение точек показывает корреляцию между ними. Её можно построить с помощью метода `scatter()` модуля `plotting`.

Синтаксис:

```
scatter(parameters)
```

Пример:

In [8]:
```
# importing the modules
from bokeh.plotting import figure, show
import random

# instantiating the figure object
graph = figure(title = "Bokeh Scatter Graph", width=600, height=400)

# points to be plotted
x = [n for n in range(256)]
y = [random.random() + 1 for n in range(256)]

# plotting the graph
graph.scatter(x, y)

# displaying the model
show(graph)
```

График с затенением (Patch Plot)

График с затенением затеняет область, отображающую группу объектов с одинаковыми свойствами. Его можно создать с помощью метода `patch()` модуля `plotting`.

Синтаксис:

```
patch(parameters)
```

Пример:

In [9]:
```
# importing the modules
from bokeh.plotting import figure, show
import random

# instantiating the figure object
graph = figure(title = "Bokeh Patch Plot", width=600, height=400)

# points to be plotted
x = [n for n in range(256)]
y = [random.random() + 1 for n in range(256)]

# plotting the graph
graph.patch(x, y)

# displaying the model
show(graph)
```

Диаграмма областей

Диаграмма областей (Area Plot) определяется как закрашенные области между двумя рядами данных, имеющими общую площадь. Класс Bokeh `Figure` имеет два метода: `varea()` и `harea()`.

Синтаксис:

```
varea(x, y1, y2, **kwargs)

harea(x1, x2, y, **kwargs)
```

Пример 1: Создание вертикальной диаграммы областей

In [10]:
```
# Implementation of bokeh function
from bokeh.plotting import figure, show

x = [1, 2, 3, 4, 5]
y1 = [2, 4, 5, 2, 4]
y2 = [1, 2, 2, 3, 6]

p = figure(width=600, height=400)

# area plot
p.varea(x=x, y1=y1, y2=y2, fill_color="green")

show(p)
```

Пример 2: Создание горизонтальной диаграммы областей

In [11]:
```
# Implementation of bokeh function
```

```
from bokeh.plotting import figure, show

y = [1, 2, 3, 4, 5]
x1 = [2, 4, 5, 2, 4]
x2 = [1, 2, 2, 3, 6]

p = figure(width=600, height=400)

# area plot
p.harea(x1=x1, x2=x2, y=y, fill_color="green")

show(p)
```

Круговая диаграмма

Bokeh не предоставляет прямого метода для построения круговой диаграммы. Её можно создать с помощью метода `wedge()`. В методе `wedge()` основными параметрами являются координаты `x` и `y` клина, радиус, начальный угол и конечный угол клина. Чтобы клинья выглядели как круговая диаграмма, параметры `x`, `y` и радиус всех клиньев будут одинаковыми. Мы будем корректировать только начальный и конечный углы.

Синтаксис:

```
wedge(parameters)
```

Пример:

In [12]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Bokeh Wedge Graph", width=400, height=400)

# the points to be plotted
x = 0
y = 0

# radius of the wedge
radius = 1.

# start angle of the wedge
start_angle = 1

# end angle of the wedge
end_angle = 2

# plotting the graph
graph.wedge(x, y, radius = radius,
            start_angle = start_angle,
            end_angle = end_angle)
```

```
# displaying the model
show(graph)
```

Создание различных фигур

Класс `Figure` в Bokeh позволяет создавать векторизованные глифы различных форм, таких как круг, прямоугольник, овал, многоугольник и т. д. Давайте рассмотрим их подробнее.

Круг

Класс `figure` в Bokeh содержит следующие методы для рисования глифов круга:

- Метод `scatter()` с маркером по умолчанию используется для добавления глифа круга к фигуре и требует указания координат `x` и `y` его центра.
- Метод `scatter()` с маркером `'circle_cross'` используется для добавления глифа круга с крестом «+» в центре к фигуре.
- Метод `scatter()` с маркером `'circle_x'` используется для добавления глифа круга с крестом «X» в центре к фигуре.

Пример:

In [13]:
```
from bokeh.plotting import figure, show

# creating the figure object
plot = figure(width = 400, height = 400)

plot.scatter(x = [1, 2, 3], y = [3, 7, 5], size = 30)

show(plot)
```

In [14]:
```
#import numpy as np
from bokeh.plotting import figure, show

# creating the figure object
plot = figure(width = 400, height = 400)

plot.scatter(
    marker='circle_cross',
    x=[1, 2, 3],
    y=[3, 7, 5],
    size=30,
    line_color="firebrick",    # Цвет контура
    fill_color="lightgreen",   # Цвет заливки
    line_width=2,              # Толщина линии
    alpha=0.8                  # Прозрачность
)
```

```
show(plot)
```

Эллипс

Метод `ellipse()` можно использовать для построения эллипсов (овалов) на графике.

Синтаксис:

```
ellipse(параметры)
```
Пример:

In [15]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title="Bokeh Ellipse Graph", width=400, height=400)

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [i * 2 for i in x]

# plotting the graph
graph.ellipse(x, y,
        height = 0.5,
        width = 1)

# displaying the model
show(graph)
```

Треугольник

Треугольник можно создать с помощью метода `scatter()` с маркером `'triangle'`.

Синтаксис:

```
scatter(marker='triangle', ...)
```
Пример:

In [16]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title="Bokeh Triangle Graph", width=400, height=400)

# the points to be plotted
x = 1
y = 1

# plotting the graph
```

```
graph.scatter(marker='triangle', x=x, y=y, size = 150)

# displaying the model
show(graph)
```

Прямоугольник

Как и круги и овалы, прямоугольники можно построить в Bokeh. Для этого используется метод `rect()`.

Синтаксис:

`rect(параметры)`
Пример:

In [17]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Bokeh Rectangle Graph", match_aspect = True)

# the points to be plotted
x = 0
y = 0
width = 10
height = 5

# plotting the graph
graph.rect(x, y, width, height)

# displaying the model
show(graph)
```

Многоугольник

Функция Bokeh также может использоваться для построения нескольких многоугольников на графике. Построение нескольких многоугольников на графике можно выполнить с помощью метода `multi_polygons()` модуля `plotting`.

Синтаксис:

`multi_polygons(parameters)`
Пример:

In [18]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Bokeh Multiple Polygons Graph",
               width=400, height=400)
```

```
# the points to be plotted
xs = [[[[1, 1, 3, 4]]]]
ys = [[[[1, 3, 2 ,1]]]]

# plotting the graph
graph.multi_polygons(xs, ys)

# displaying the model
show(graph)
```

Построение нескольких графиков

Bokeh предоставляет несколько вариантов компоновки для создания нескольких графиков. К таким вариантам относятся:

- Вертикальная компоновка
- Горизонтальная компоновка
- Сеточная компоновка

Вертикальные компоновки

Вертикальная компоновка устанавливает все графики в вертикальном положении и может быть создана с помощью метода `column()`.

In [19]:
```
from bokeh.io import show
from bokeh.layouts import column
from bokeh.plotting import figure

x = [1, 2, 3, 4, 5, 6]
y0 = x
y1 = [i * 2 for i in x]
y2 = [i ** 2 for i in x]

# create a new plot
s1 = figure(width=500, height=200)
s1.scatter(x, y0, size=10, alpha=0.5)

# create another one
s2 = figure(width=500, height=200)
s2.scatter(marker='triangle', x=x, y=y1, size=10, alpha=0.5)

# create and another
s3 = figure(width=500, height=200)
s3.scatter(marker='square', x=x, y=y2, size=10, alpha=0.5)

# put all the plots in a VBox
p = column(s1, s2, s3)

# show the results
show(p)
```

Горизонтальная компоновка

Горизонтальная компоновка размещает все графики горизонтально. Ее можно создать с помощью метода `row()`.

Пример:

In [20]:
```
from bokeh.io import show
from bokeh.layouts import row
from bokeh.plotting import figure

x = [1, 2, 3, 4, 5, 6]
y0 = x
y1 = [i * 2 for i in x]
y2 = [i ** 2 for i in x]

# create a new plot
s1 = figure(width=200, height=400)
s1.scatter(x, y0, size=10, alpha=0.5)

# create another one
s2 = figure(width=200, height=400)
s2.scatter(marker='triangle', x=x, y=y1, size=10, alpha=0.5)

# create and another
s3 = figure(width=200, height=400)
s3.scatter(marker='square', x=x, y=y2, size=10, alpha=0.5)

# put all the plots in a VBox
p = row(s1, s2, s3)

# show the results
show(p)
```

Сетка

Метод `gridplot()` можно использовать для размещения всех графиков в виде сетки. Также можно передать значение `None`, чтобы оставить пустое место для графика.

Пример:

In [21]:
```
from bokeh.io import show
from bokeh.layouts import gridplot
from bokeh.plotting import figure

x = [1, 2, 3, 4, 5, 6]
y0 = x
y1 = [i * 2 for i in x]
y2 = [i ** 2 for i in x]

# create a new plot
s1 = figure()
s1.scatter(x, y0, size=10, alpha=0.5)
```

```
# create another one
s2 = figure()
s2.scatter(marker='triangle', x=x, y=y1, size=10, alpha=0.5)

# create and another
s3 = figure()
s3.scatter(marker='square', x=x, y=y2, size=10, alpha=0.5)

# put all the plots in a grid
p = gridplot([[s1, None], [s2, s3]], width=200, height=200)

# show the results
show(p)
```

Интерактивная визуализация данных

Одной из ключевых особенностей Bokeh, отличающей его от других библиотек визуализации, является добавление интерактивности к графику. Давайте рассмотрим различные интерактивные элементы, которые можно добавить к графику.

Настройка инструментов графика

На всех приведенных выше графиках вы, должно быть, заметили панель инструментов, которая чаще всего появляется справа от графика. Bokeh предоставляет нам методы для работы с этими инструментами. Инструменты можно разделить на четыре категории.

- *Жесты*: Эти инструменты обрабатывают жесты, такие как перемещение. Существует три типа жестов:
  - Инструменты перемещения/перетаскивания
  - Инструменты щелчка/касания
  - Инструменты прокрутки/щипка
- *Действия*: Эти инструменты обрабатывают нажатия кнопок.
- *Инспекторы*: Эти инструменты отображают информацию или аннотируют график, например, инструмент наведения курсора.
- *Инструменты редактирования*: Это инструменты с несколькими жестами, которые позволяют добавлять и удалять глифы с графика.

Настройка положения панели инструментов

Мы можем задать положение панели инструментов в соответствии со своими потребностями. Это можно сделать, передав параметр `toolbar_location` методу `figure()`. Возможные значения этого параметра:

- "above"
- "below"

- "left"
- "right"

Пример:

In [22]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Bokeh ToolBar",
               width=400, height=400,
               toolbar_location="below")

# the points to be plotted
x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]

# height / thickness of the plot
width = 0.5

# plotting the scatter graph
graph.scatter(x, y)

# displaying the model
show(graph)
```

Интерактивные легенды

В разделе «Аннотации и легенды» мы рассмотрели список всех параметров легенд, однако параметр `click_policy` мы еще не обсуждали. Это свойство делает легенду интерактивной. Существует два типа интерактивности:

- *Скрытие* (Hiding): скрывает глифы.
- *Отключение* (Muting): скрытие глифа приводит к его полному исчезновению, с другой стороны, отключение глифа просто уменьшает его значимость в зависимости от параметров.

Пример 1: Скрытие легенды

In [23]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Bokeh Hiding Glyphs", width=400, height=400)

# plotting the graph
graph.vbar(x = 1, top = 5,
        width = 1, color = "violet",
        legend_label = "Violet Bar")
graph.vbar(x = 2, top = 5,
        width = 1, color = "green",
```

```
        legend_label = "Green Bar")
graph.vbar(x = 3, top = 5,
        width = 1, color = "yellow",
        legend_label = "Yellow Bar")
graph.vbar(x = 4, top = 5,
        width = 1, color = "red",
        legend_label = "Red Bar")

# enable hiding of the glyphs
graph.legend.click_policy = "hide"

# displaying the model
show(graph)
```

Пример 2: Отключение легенды

In [24]:
```
# importing the modules
from bokeh.plotting import figure, show

# instantiating the figure object
graph = figure(title = "Bokeh Hiding Glyphs", width=400, height=400)

# plotting the graph
graph.vbar(x = 1, top = 5,
        width = 1, color = "violet",
        legend_label = "Violet Bar",
        muted_alpha=0.2)
graph.vbar(x = 2, top = 5,
        width = 1, color = "green",
        legend_label = "Green Bar",
        muted_alpha=0.2)
graph.vbar(x = 3, top = 5,
        width = 1, color = "yellow",
        legend_label = "Yellow Bar",
        muted_alpha=0.2)
graph.vbar(x = 4, top = 5,
        width = 1, color = "red",
        legend_label = "Red Bar",
        muted_alpha=0.2)

# enable hiding of the glyphs
graph.legend.click_policy = "mute"

# displaying the model
show(graph)
```

Добавление виджетов на график

Bokeh предоставляет функции графического интерфейса пользователя, аналогичные HTML-формам, такие как кнопки, ползунки, флажки и т. д. Они обеспечивают интерактивный интерфейс для графика, позволяющий изменять параметры графика, модифицировать данные графика и т. д. Давайте посмотрим, как использовать и добавлять некоторые часто

используемые виджеты.

- *Кнопки*: Этот виджет добавляет на график простую кнопку. Нам нужно передать пользовательскую функцию JavaScript в метод `CustomJS()` класса модели.

Синтаксис:

```
Button(label, icon, callback)
```

Пример:

In [25]:
```
from bokeh.io import show
from bokeh.models import Button, CustomJS

button = Button(label="Кнопка")
button.js_on_click(CustomJS(
  code="console.log('button: click!', this.toString())"
))

show(button)
```

- *CheckboxGroup*: Добавляет стандартный флажок на график. Аналогично кнопкам, нам необходимо передать пользовательскую функцию JavaScript в метод `CustomJS()` класса модели.

Пример:

In [26]:
```
from bokeh.io import show
from bokeh.models import CheckboxGroup, CustomJS

L = ["First", "Second", "Third"]

# the active parameter sets checks the selected value
# by default
checkbox_group = CheckboxGroup(
    labels=L,
    active=[0, 2],
)

checkbox_group.js_on_change('active', CustomJS(code="""
    console.log('checkbox_group: active=' + this.active, this.toStrin
"""))

show(checkbox_group)
```

- *RadioGroup*: Добавляет простую радиокнопку и принимает пользовательскую функцию JavaScript.

Синтаксис:

`RadioGroup(labels, active)`
Пример:

In [27]:
```
from bokeh.io import show
from bokeh.models import RadioGroup, CustomJS

L = ["First", "Second", "Third"]

# the active parameter sets checks the selected value
# by default
radio_group = RadioGroup(labels=L, active=1)

radio_group.js_on_change('active', CustomJS(code="""
    console.log('radio_group: active=' + this.active, this.toString()
"""))

show(radio_group)
```

- *Sliders* (Ползунки): Добавляет ползунок на график. Для этого также требуется пользовательская функция JavaScript.

Синтаксис:

`Slider(start, end, step, value)`
Пример:

In [28]:
```
from bokeh.io import show
from bokeh.models import CustomJS, Slider

slider = Slider(start=1, end=20, value=1, step=2, title="Slider")

slider.js_on_change("value", CustomJS(code="""
    console.log('slider: value=' + this.value, this.toString())
"""))

show(slider)
```

- *DropDown*: Добавляет выпадающий список на график, и, как и любой другой виджет, требует наличия пользовательской функции JavaScript в качестве функции обратного вызова.

Пример:

In [29]:
```
from bokeh.io import show
from bokeh.models import CustomJS, Dropdown

menu = [("First", "First"), ("Second", "Second"), ("Third", "Third")]

dropdown = Dropdown(
    label="Dropdown Menu", button_type="success", menu=menu
)
```

```
dropdown.js_on_event("menu_item_click", CustomJS(
    code="console.log('dropdown: ' + this.item, this.toString())"))

show(dropdown)
```

- *Tab Widget* (Виджет вкладок): Виджет вкладок добавляет вкладки, и каждая вкладка отображает отдельный график.

Пример:

In [30]:
```
from bokeh.plotting import figure, show
from bokeh.models import TabPanel, Tabs

fig1 = figure(width=300, height=300)

x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

fig1.line(x, y, line_color='green')
tab1 = TabPanel(child=fig1, title="Tab 1")

fig2 = figure(width=300, height=300)

fig2.line(y, x, line_color='red')
tab2 = TabPanel(child=fig2, title="Tab 2")

all_tabs = Tabs(tabs=[tab1, tab2])

show(all_tabs)
```

Варианты визуализации диаграмм

- в ноутбуке Jupyter
- в отдельной вкладке браузера

In [31]:
```
from bokeh.io import reset_output
reset_output()

from bokeh.plotting import figure, show
from bokeh.models import TabPanel, Tabs

fig1 = figure(width=300, height=300)

x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

fig1.line(x, y, line_color='green')
tab1 = TabPanel(child=fig1, title="Tab 1")

fig2 = figure(width=300, height=300)

fig2.line(y, x, line_color='red')
```

```
tab2 = TabPanel(child=fig2, title="Tab 2")

all_tabs = Tabs(tabs=[tab1, tab2])

show(all_tabs)
```

In [32]:
```
from bokeh.io import reset_output
reset_output()

from bokeh.plotting import figure, output_file, show
from bokeh.models import TabPanel, Tabs

fig1 = figure(width=300, height=300)

x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

fig1.line(x, y, line_color='green')
tab1 = TabPanel(child=fig1, title="Tab 1")

fig2 = figure(width=300, height=300)

fig2.line(y, x, line_color='red')
tab2 = TabPanel(child=fig2, title="Tab 2")

all_tabs = Tabs(tabs=[tab1, tab2])

output_file("tabs.html")
```

In [33]:
```
from IPython.display import HTML

HTML(filename="tabs.html")
```

Out[33]:

Визуализация с библиотекой Plotly

In [34]:
```
import plotly
print(plotly.__version__)
import plotly.express as px
```

```
6.5.0
```

In [35]:
```
import pandas as pd
import numpy as np
```

Структура Plotly

В Plotly есть два основных модуля, используемых для создания визуализаций:

- `plotly.graph_objects`

Этот модуль предоставляет классы Python для построения графиков с использованием объектов, таких как `Figure`, `Layout`, и типов графиков, таких как `Scatter`, `Bar` и `Box`. Графики имеют следующую структуру:

- `data`: Список графиков (например, scatter, bar)
- `layout`: Конфигурация графика (например, оси, заголовок, легенда)
- `frames`: Используется для анимации

Каждый рисунок (figure) сериализуется в JSON и отображается Plotly.js. Вложенные элементы, такие как `layout.legend`, представляют собой словари настраиваемых свойств.

- `plotly.express`

Это высокоуровневый модуль, используемый для быстрого создания полных графиков. Внутри он использует `graph_objects` и возвращает экземпляр `graph_objects.Figure`.

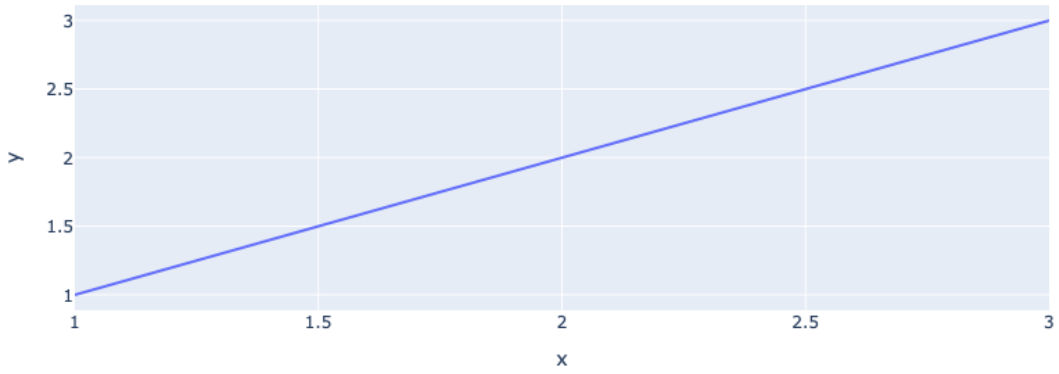
Пример:

In [36]:
```
import plotly.express as px

# Creating the Figure instance
fig = px.line(x=[1,2, 3], y=[1, 2, 3])

# printing the figure instance
print(fig)
```

```
Figure({
    'data': [{'hovertemplate': 'x=%{x}<br>y=%{y}<extra></extra>',
              'legendgroup': '',
              'line': {'color': '#636efa', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '',
              'orientation': 'v',
              'showlegend': False,
              'type': 'scatter',
              'x': {'bdata': 'AQID', 'dtype': 'i1'},
              'xaxis': 'x',
              'y': {'bdata': 'AQID', 'dtype': 'i1'},
              'yaxis': 'y'}],
    'layout': {'legend': {'tracegroupgap': 0},
               'margin': {'t': 60},
               'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title':
{'text': 'x'}},
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title':
{'text': 'y'}}}
})
```

In [37]:
```
# install nbformat
import plotly.express as px
```

```
fig = px.line(x=[1, 2, 3], y=[1, 2, 3])

fig.show()
```



В приведенном выше примере импортируется модуль plotly.express, который возвращает экземпляр Figure. Мы создали простой линейный график, передав координаты x и y точек, которые нужно отобразить.

Создание различных типов графиков

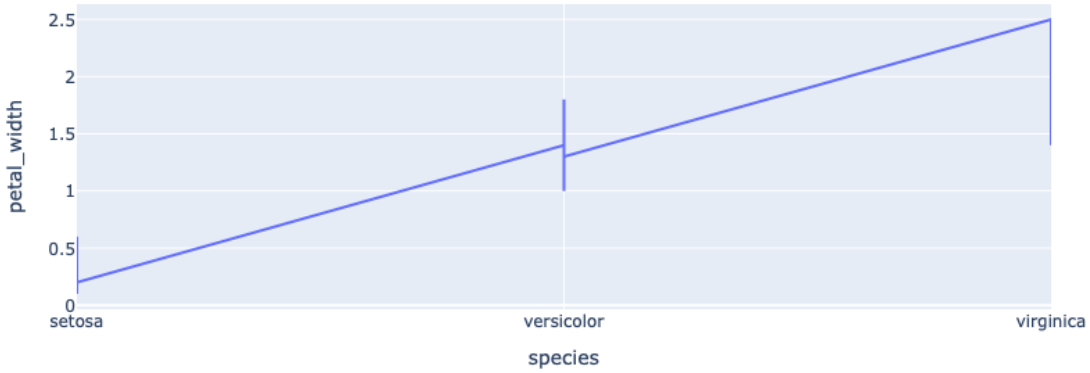
С помощью Plotly можно создать более 40 типов графиков, и каждый график можно создать, используя классы `plotly.express` и `plotly.graph_objects`. Давайте рассмотрим некоторые часто используемые графики с помощью Plotly.

Линейный график

Линейный график в Plotly — это гораздо более доступное и замечательное дополнение к Plotly, которое позволяет обрабатывать различные типы данных и создавать легко настраиваемую статистику. С помощью `px.line` каждая позиция данных представляется как вершина (местоположение которой задается столбцами `x` и `y`) полилинии в двумерном пространстве.

Пример:

In [5]:
```
import plotly.express as px

df = px.data.iris()

# plotting the line chart
fig = px.line(df, x="species", y="petal_width")

fig.show()
```

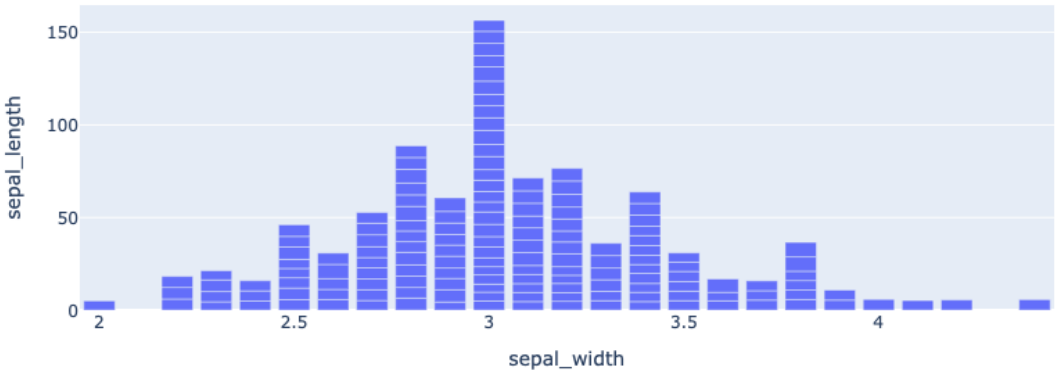


Столбчатая диаграмма

Столбчатая диаграмма — это графическое представление данных, в котором категориальные данные представлены в виде прямоугольных столбцов, высота или длина которых пропорциональны представляемым ими значениям. Другими словами, это графическое представление набора данных. Эти наборы данных содержат числовые значения переменных, которые представляют собой длину или высоту.

Пример:

In [6]:
```
import plotly.express as px

df = px.data.iris()

# plotting the bar chart
fig = px.bar(df, x="sepal_width", y="sepal_length")

fig.show()
```



Гистограммы

Гистограмма представляет собой прямоугольную область для отображения статистической информации, пропорциональную частоте переменной и ее

ширине в последовательных числовых интервалах. Это графическое представление, которое распределяет группу точек данных по различным заданным диапазонам. Особенностью гистограммы является отсутствие промежутков между столбцами, что делает ее похожей на вертикальную столбчатую диаграмму.

Пример:

In [7]:
```
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the histogram
fig = px.histogram(df, x="sepal_length", y="petal_width")

# showing the plot
fig.show()
```

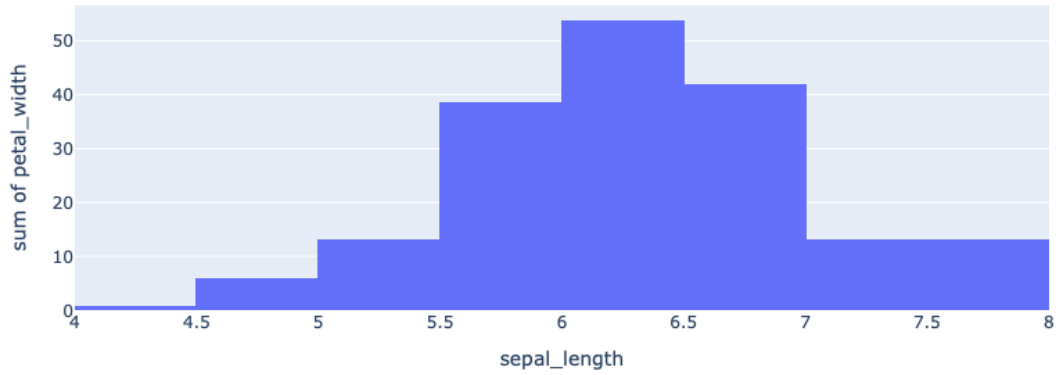


Диаграмма рассеяния и пузырьковая диаграмма

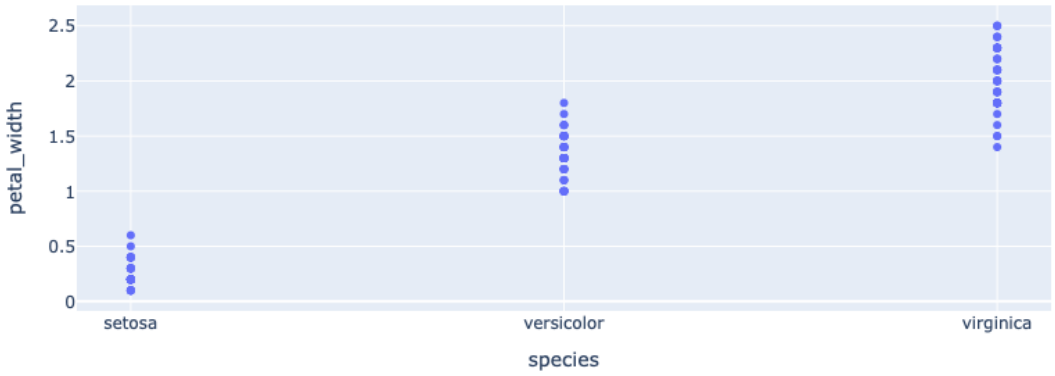
Диаграмма рассеяния — это набор пунктирных точек, представляющих отдельные данные по горизонтальной и вертикальной осям. На графике значения двух переменных отображаются по осям X и Y, а расположение точек показывает корреляцию между ними.

Пузырьковая диаграмма — это диаграмма рассеяния с пузырьками (кругами, закрашенными разными цветами). Размеры пузырьков зависят от другой переменной в данных. Ее можно создать с помощью метода `scatter()` библиотеки `plotly.express`.

Пример 1: Диаграмма рассеяния

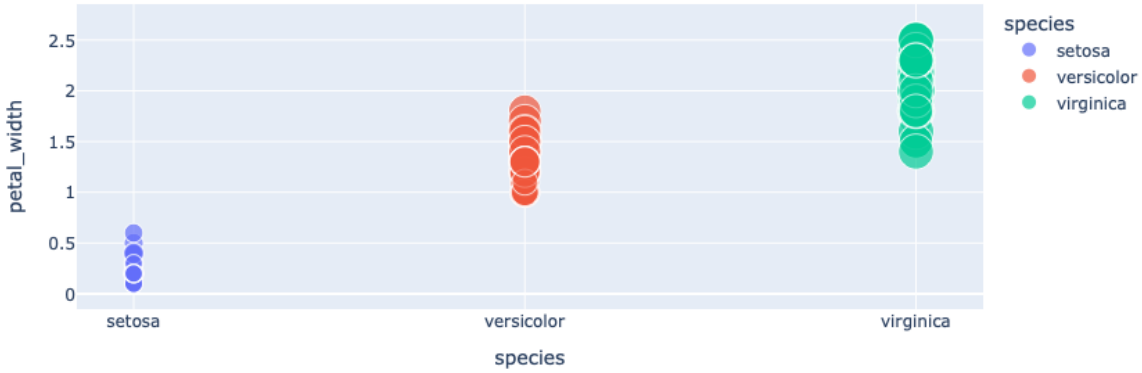
In [8]:
```
import plotly.express as px

# using the iris dataset
df = px.data.iris()
```

```
# plotting the scatter chart
fig = px.scatter(df, x="species", y="petal_width")

# showing the plot
fig.show()
```



Пример 2: Пузырьковая диаграмма

In [9]:
```
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the bubble chart
fig = px.scatter(df, x="species", y="petal_width",
                 size="petal_length", color="species")

# showing the plot
fig.show()
```



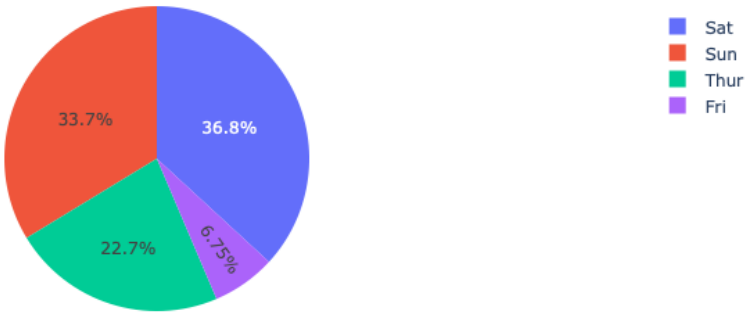
Круговые диаграммы

Круговая диаграмма — это круговая статистическая диаграмма, разделенная на сектора для иллюстрации числовых пропорций. Она представляет собой особую диаграмму, использующую «сектора», где

каждый сектор показывает относительные размеры данных. Круговая диаграмма разрезается в виде радиусов на сегменты, описывающие относительные частоты или величины, также известная как круговая диаграмма.

Пример:

In [10]:
```
import plotly.express as px

# using the tips dataset
df = px.data.tips()

# plotting the pie chart
fig = px.pie(df, values="total_bill", names="day")

# showing the plot
fig.show()
```



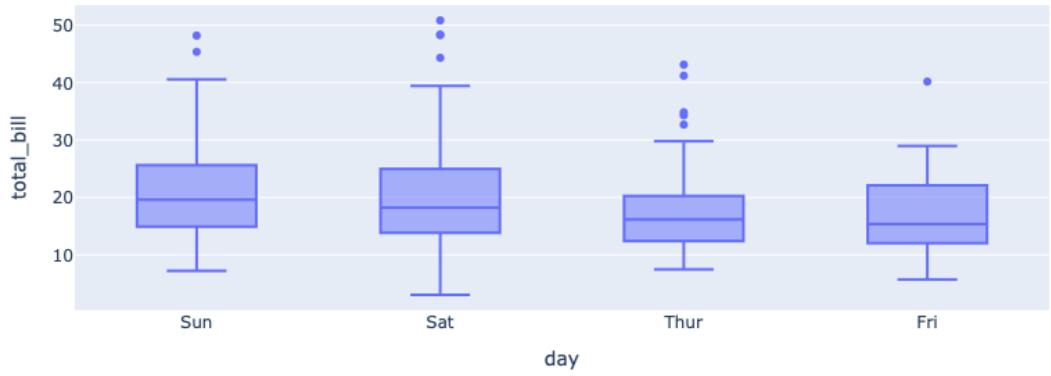
Диаграммы размаха

Диаграмма размаха, также известная как диаграмма «усы» (Whisker plot), создается для отображения сводной информации о наборе значений данных, имеющих такие свойства, как минимум, первый квартиль, медиана, третий квартиль и максимум. На диаграмме размаха создается прямоугольник от первого квартиля до третьего, также имеется вертикальная линия, проходящая через этот прямоугольник в точке медианы. Здесь по оси X откладывается отображаемая информация, а по оси Y — частотное распределение.

Пример:

In [11]:
```
import plotly.express as px

# using the tips dataset
df = px.data.tips()

# plotting the box chart
```

```
fig = px.box(df, x="day", y="total_bill")

# showing the plot
fig.show()
```

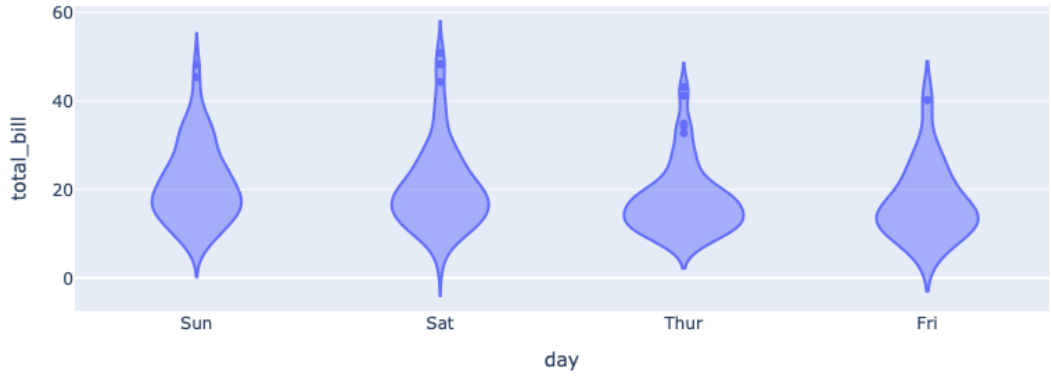


Скрипичные диаграммы

Скрипичная диаграмма — это метод визуализации распределения числовых данных различных переменных. Она похожа на диаграмму размаха, но с повернутыми графиками с каждой стороны, что дает больше информации об оценке плотности по оси Y. Плотность зеркально отражается и переворачивается, а полученная фигура заполняется, создавая изображение, напоминающее скрипку. Преимущество скрипичной диаграммы заключается в том, что она может показать нюансы в распределении, которые не заметны на диаграмме размаха. С другой стороны, диаграмма размаха более четко показывает выбросы в данных.

Пример:

In [12]:
```
import plotly.express as px

# using the tips dataset
df = px.data.tips()

# plotting the violin chart
fig = px.violin(df, x="day", y="total_bill")

# showing the plot
fig.show()
```

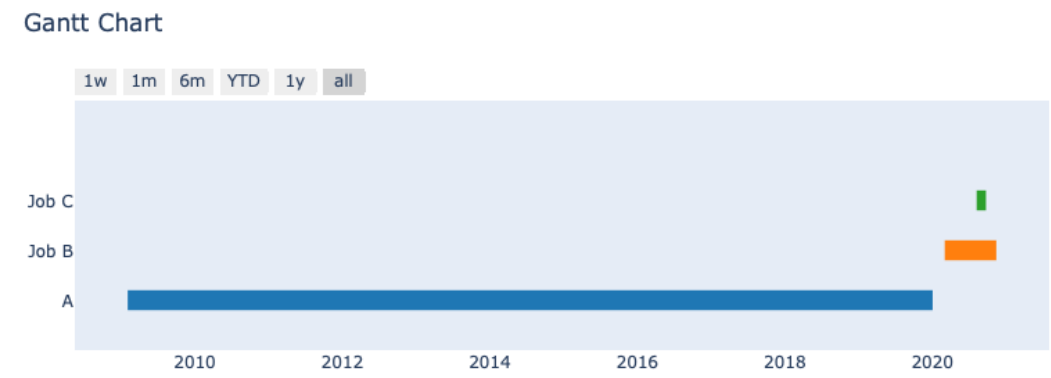


Диаграммы Ганта

Диаграмма Ганта (Generalized Activity Normalization Time Table) — это тип диаграммы, в которой горизонтальные линии показывают объем выполненной работы или завершенного производства за определенный период времени по отношению к запланированному объему для этих проектов.

Пример:

In [13]:
```
import plotly.figure_factory as ff

# Data to be plotted
df = [dict(Task="A", Start='2020-01-01', Finish='2009-02-02'),
    dict(Task="Job B", Start='2020-03-01', Finish='2020-11-11'),
    dict(Task="Job C", Start='2020-08-06', Finish='2020-09-21')]

# Creating the plot
fig = ff.create_gantt(df)
fig.show()
```



Контурные графики

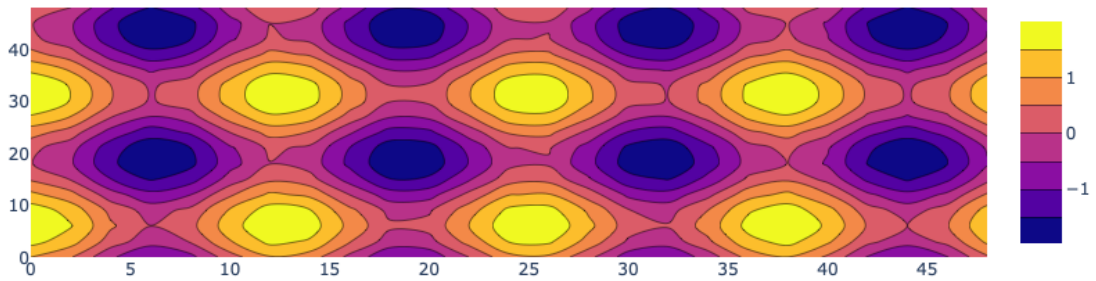
Контурные графики, также называемые графиками уровней, — это

инструмент для проведения многомерного анализа и визуализации трехмерных графиков в двумерном пространстве. Если мы рассматриваем X и Y как переменные, которые хотим отобразить на графике, то отклик Z будет отображен в виде срезов на плоскости X-Y, поэтому контуры иногда называют Z-срезами или изооткликами.

Контурный график используется в случае, когда необходимо увидеть изменения некоторого значения (Z) как функцию относительно двух значений (X, Y). Рассмотрим приведенный ниже пример.

Пример:

In [14]:
```
import plotly.graph_objects as go


# Creating the X, Y value that will
feature_x = np.arange(0, 50, 2)
feature_y = np.arange(0, 50, 3)

# Creating 2-D grid of features
[X, Y] = np.meshgrid(feature_x, feature_y)

Z = np.cos(X / 2) + np.sin(Y / 4)

# plotting the figure
fig = go.Figure(data =
    go.Contour(x = feature_x, y = feature_y, z = Z))

fig.show()
```



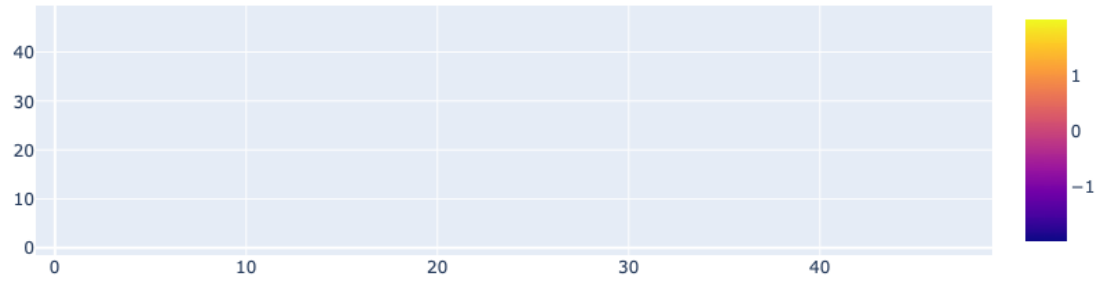
Тепловые карты

Тепловая карта определяется как графическое представление данных с использованием цветов для визуализации значений матрицы. В ней для представления более распространенных значений или высокой активности используются более яркие цвета, в основном красноватые, а для представления менее распространенных значений или активности

предпочтительны более темные цвета. Название тепловой карты также обозначает матрицу затенения.

Пример:

In [15]:
```
import plotly.graph_objects as go


feature_x = np.arange(0, 50, 2)
feature_y = np.arange(0, 50, 3)

# Creating 2-D grid of features
[X, Y] = np.meshgrid(feature_x, feature_y)

Z = np.cos(X / 2) + np.sin(Y / 4)

# plotting the figure
fig = go.Figure(data =
     go.Heatmap(x = feature_x, y = feature_y, z = Z,))

fig.show()
```



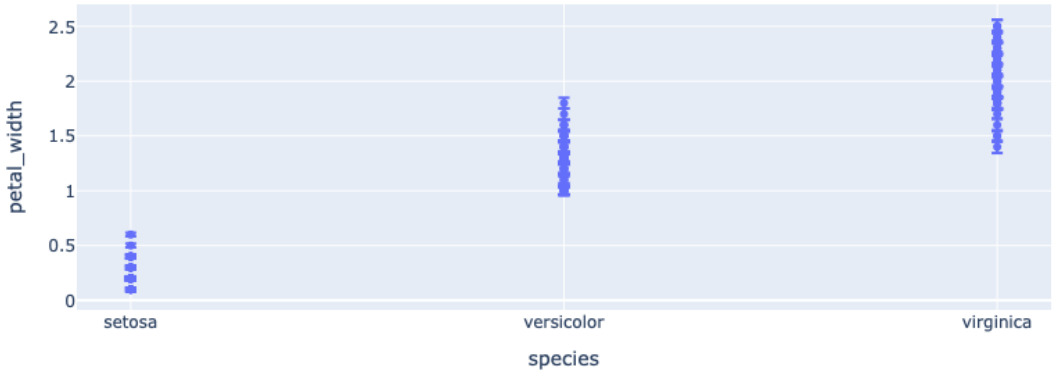
Диаграммы ошибки

Для функций, представляющих двумерные точки данных, таких как `px.scatter`, `px.line`, `px.bar` и т. д., диаграммы ошибки (error bars) указываются в виде имени столбца, содержащего значения `error_x` (погрешность по оси `x`) и `error_y` (погрешность по оси `y`). Погрешности представляют собой графическое отображение изменений данных и используются на графиках для обозначения погрешности или неопределенности в сообщаемых данных.

Пример:

In [16]:
```
import plotly.express as px

# using the iris dataset
df = px.data.iris()
```

```
# Calculating the error field
df["error"] = df["petal_length"]/100

# plotting the scatter chart
fig = px.scatter(df, x="species", y="petal_width",
                 error_x="error", error_y="error")

# showing the plot
fig.show()
```



3D-диаграммы

3D-линейные графики

Линейный график в Plotly — это гораздо более доступное и удобное дополнение к Plotly, позволяющее обрабатывать различные типы данных и создавать удобные для оформления статистические данные. С помощью `px.line_3d` каждая позиция данных представляется как вершина (местоположение которой задается столбцами `x`, `y` и `z`) полилинии в 3D-пространстве.

Пример:

In [17]:
```
import plotly.express as px

# data to be plotted
df = px.data.tips()

# plotting the figure
fig = px.line_3d(df, x="sex", y="day",
                 z="time", color="sex")

fig.show()
```

Трехмерная диаграмма рассеяния

Трехмерная диаграмма рассеяния позволяет строить двухмерные графики, которые можно улучшить, добавив до трех дополнительных переменных, используя семантику параметров оттенка, размера и стиля. Все параметры управляют визуальной семантикой, которая используется для идентификации различных подмножеств. Использование избыточной семантики может помочь сделать графики более доступными. Ее можно создать с помощью функции `scatter_3d` класса `plotly.express`.

Пример:

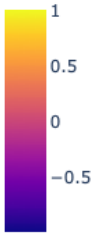
In [18]:
```
import plotly.express as px

# Data to be plotted
df = px.data.iris()

# Plotting the figure
fig = px.scatter_3d(df, x = 'sepal_width',
                    y = 'sepal_length',
                    z = 'petal_width',
                    color = 'species')

fig.show()
```

Трехмерные поверхностные графики

Поверхностный график — это график, отображающий трехмерные данные: X, Y и Z. Вместо отображения отдельных точек данных, поверхностный график показывает функциональную зависимость между зависимой переменной Y и двумя независимыми переменными X и Z. Этот график используется для различения зависимых и независимых переменных.

Пример:

In [19]:
```
import plotly.graph_objects as go
import numpy as np

# Data to be plotted
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T
z = np.cos(x ** 2 + y ** 2)

# plotting the figure
fig = go.Figure(data=[go.Surface(x=x, y=y, z=z)])

fig.show()
```



Интерактивные возможности построения графиков

Plotly предоставляет различные инструменты для взаимодействия с графиками, такие как добавление выпадающих списков, кнопок, ползунков и т. д. Их можно создавать с помощью атрибута update menu макета графика. Давайте рассмотрим, как это сделать подробно.

Создание выпадающего меню

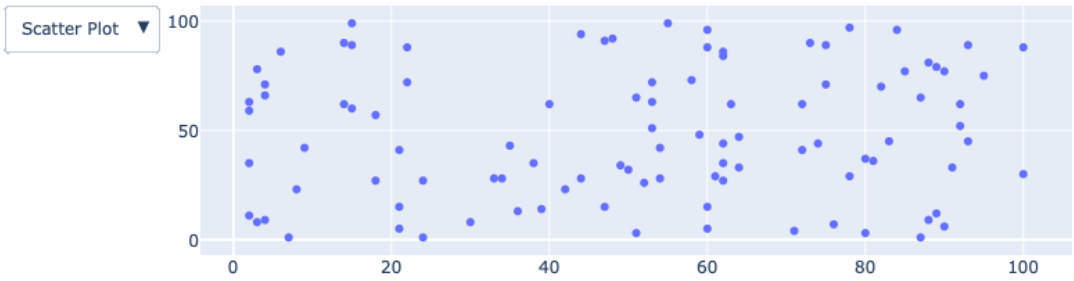
Выпадающее меню — это часть кнопки меню, которая постоянно отображается на экране. Каждая кнопка меню связана с виджетом Menu,

который может отображать варианты выбора для этой кнопки меню при нажатии на нее. В Plotly существует 4 возможных метода изменения графиков с помощью метода update menu.

- `restyle` : изменение данных или атрибутов данных
- `relayout` : изменение атрибутов макета
- `update` : изменение данных и атрибутов макета
- `animate` : запуск или приостановка анимации

Пример:

In [20]:
```
import plotly.graph_objects as px
import numpy as np

# creating random data through randomint
np.random.seed(42)

# Data to be Plotted
random_x = np.random.randint(1, 101, 100)
random_y = np.random.randint(1, 101, 100)

plot = px.Figure(data=[px.Scatter(
    x=random_x,
    y=random_y,
    mode='markers',)
])

# Add dropdown
plot.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=["type", "scatter"],
                    label="Scatter Plot",
                    method="restyle"
                ),
                dict(
                    args=["type", "bar"],
                    label="Bar Chart",
                    method="restyle"
                )
            ]),
            direction="down",
        ),
    ]
)

plot.show()
```



В приведенном выше примере мы создали два графика для одних и тех же данных. Доступ к этим графикам осуществляется через выпадающее меню.

Добавление кнопок на график

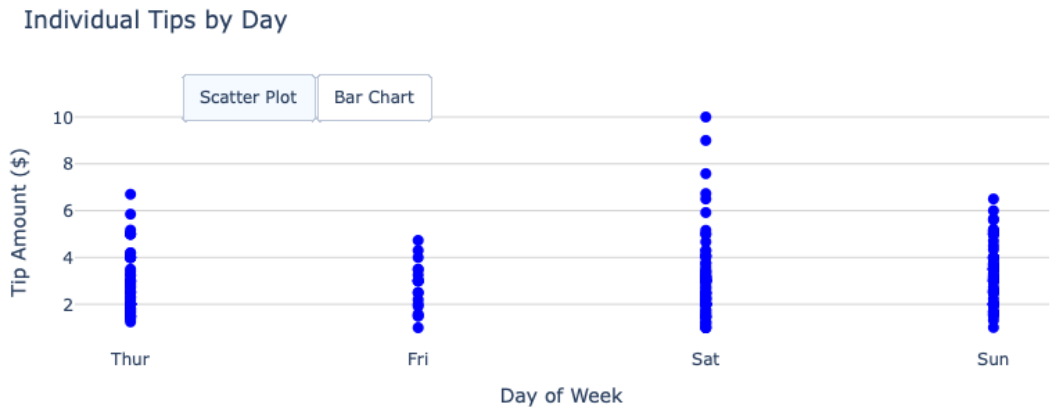
В Plotly пользовательские кнопки действий используются для быстрого выполнения действий непосредственно из записи. Пользовательские кнопки можно добавлять в макеты страниц в CRM, маркетинге и пользовательских приложениях. Также существует 4 возможных метода, которые можно применять к пользовательским кнопкам:

- `restyle` : изменение данных или атрибутов данных
- `relayout` : изменение атрибутов макета
- `update` : изменение данных и атрибутов макета
- `animate` : запуск или приостановка анимации

Пример:

In [22]:
```
import plotly.graph_objects as go
import plotly.express as px

# Загружаем встроенные данные (корректный способ)
df = px.data.tips()

# Агрегируем данные для bar chart (усредненные чаевые по дням)
bar_data = df.groupby('day', as_index=False)['tip'].mean()

# Создаем фигуру с двумя трейсами
fig = go.Figure()

# Scatter plot (видим по умолчанию)
fig.add_trace(go.Scatter(
    x=df['day'],
    y=df['tip'],
    mode='markers',
    name='Individual Tips',
    marker=dict(color='blue', size=8),
```

```
    visible=True
))

# Bar chart (скрыт по умолчанию)
fig.add_trace(go.Bar(
    x=bar_data['day'],
    y=bar_data['tip'],
    name='Avg Tips',
    marker_color='firebrick',
    visible=False
))

# Настраиваем кнопки переключения
fig.update_layout(
    updatemenus=[
        dict(
            type="buttons",
            direction="left",
            buttons=[
                dict(
                    label="Scatter Plot",
                    method="update",
                    args=[{"visible": [True, False]},
                          {"title": "Individual Tips by Day"}]
                ),
                dict(
                    label="Bar Chart",
                    method="update",
                    args=[{"visible": [False, True]},
                          {"title": "Average Tips by Day"}]
                )
            ],
            pad={"r": 10, "t": 10},
            showactive=True,
            x=0.11,
            xanchor="left",
            y=1.15,
            yanchor="top"
        ),
    ],
    title="Restaurant Tips Visualization",
    xaxis_title="Day of Week",
    yaxis_title="Tip Amount ($)",
    template="plotly_white",
    height=500
)

# Добавляем легенду и сетку
fig.update_xaxes(categoryorder='array', categoryarray=['Thur', 'Fri',
fig.update_yaxes(gridcolor='lightgray')

# Отображаем график
fig.show()
```



В этом примере мы также создаём два разных графика на одних и тех же данных, и оба графика доступны с помощью кнопок.

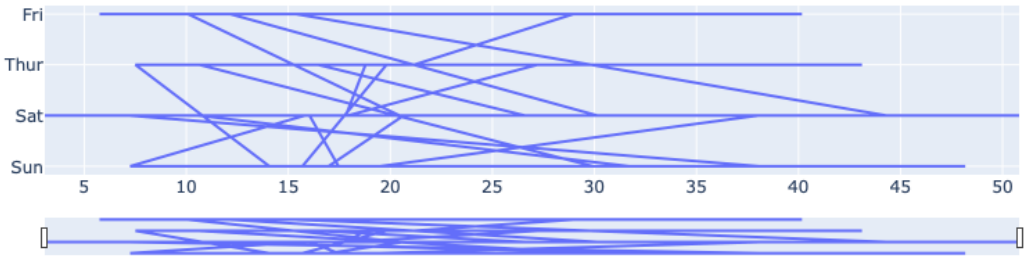
Создание ползунков и селекторов для графика

В Plotly ползунок диапазона — это настраиваемый элемент управления ввода типа диапазона. Он позволяет выбрать значение или диапазон значений между заданным минимальным и максимальным диапазоном. А селектор диапазона — это инструмент для выбора диапазонов для отображения на диаграмме. Он предоставляет кнопки для выбора предварительно настроенных диапазонов на диаграмме. Он также предоставляет поля ввода, где можно вручную ввести минимальные и максимальные даты.

Пример:

In [23]:
```
import plotly.graph_objects as px
import plotly.express as go
import numpy as np

df = go.data.tips()

x = df['total_bill']
y = df['day']

plot = px.Figure(data=[px.Scatter(
    x=x,
    y=y,
    mode='lines',)
])

plot.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                    step="day",
                    stepmode="backward"),
```

```
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
    )
)

plot.show()
```



Задание на ЛР №7

В соответствии с индивидуальным заданием (вариантом), переданным через программу «Мессенджер Яндекс», выполните следующие работы, используя для построения рисунков функции библиотеки plotly:

1. Считайте из заданного набора данных репозитария UCI значения трех признаков и метки класса.
2. Если среди меток класса имеются пропущенные значения, то удалите записи с пропущенными метками класса. Если в признаках имеются пропущенные значения, то выведите процент записей набора данных с пропущенными значениями и замените пропущенные значения на значения, указанные в индивидуальном задании. При отсутствии пропущенных значений удалите не менее 5% точек набор данных как выбросы при помощи стандартизованной оценки и выведите процент удаленных точек.
3. Постройте диаграмму рассеяния (px.scatter) на плоскости с координатами, соответствующими двум признакам с наибольшей корреляцией, отображая точки различных классов разными цветами и маркерами переменных размеров в зависимости значений третьего признака.
4. Постройте линейные графики (px.line) зависимости признака с наибольшей дисперсией от признака с наименьшей дисперсией для разных классов.

5. Постройте скрипичную диаграмму (px.violin) зависимости признака с наименьшей дисперсией от классов.
6. Постройте диаграмму размаха (px.box) зависимости признака с наибольшей дисперсией от классов. Перечислите признаки, в которых выявлены выбросы.
7. Постройте столбчатую диаграмму (px.bar), показывающую зависимость признака с наибольшей разностью между третьим и первым квартилями от класса.
8. Постройте гистограмму (px.histogram), показывающую зависимость между первым и третьим признаком.
9. Постройте трехмерный линейный график (px.line_3d) для трех признаков, отображая линии для различных классов разными цветами.
10. Постройте трехмерную диаграмму рассеяния (px.scatter_3d) для трех признаков, отображая точки различных классов разными цветами.