МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский технологический университет «МИСИС»

Новотроицкий филиал

ФАКУЛЬТЕТ МЕТАЛЛУРГИЧЕСКИХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИКИ И ЕСТЕСТВОЗНАНИЯ

НАПРАВЛЕНИЕ 09.03.03 ПРИКЛАДНАЯ ИНФОРМАТИКА

ДИСЦИПЛИНА ПРОГРАММНЫЕ СИСТЕМЫ ИНЖЕНЕРНОГО АНАЛИЗА

КУРСОВОЙ ПРОЕКТ

на тему: Методы визуализации данных с помощью библиотек языка программирования Python

Студент группы БПИ-21 И. А. Слинько

Руководитель _____ Е. Г. Филиппов

Оценка работы _____ «____» _____ 2024 г.

Новотроицк, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ
ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский технологический университет «МИСИС»
Новотроицкий филиал

Факультет Металлургических технологий

Кафедра Математики и естествознания

УТВЕРЖДАЮ

Зав. кафедрой Швалева А.В.

«____»_____ 2024 г.

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ

курсового проекта

студенту группы БПИ-21 Слинько Илье Андреевичу
(*Ф.И.О. полностью*)

1. Тема КП Методы визуализации данных с помощью библиотек языка программирования Python

2. Исходные данные ГОСТ Р ИСО/МЭК 12207 Процессы жизненного цикла программных средств; ГОСТ Р ИСО/МЭК 14764 Сопровождение программных средств; ГОСТ 34.602-2020 Информационные технологии (ИТ). Комплекс стандартов на автоматизированные системы; ГОСТ 19.504-79. ЕСПД. Руководство программиста

3. Основная литература Учебник «Изучаем Python» Марк Лутц; Учебник «Программирование на языке высокого уровня Python» Д. Ю. Федоров; Учебник «Автоматизация рутиных задач с помощью Python: практическое руководство для начинающих» Эл Свейгарт; Учебник «Говори на языке диаграмм» Джин Желязны;

Учебник «Information Dashboard Design: The Effective Visual Communication of Data» Стивен Фью

4. Лабораторное оборудование и методики, которые должны быть использованы

5. Использование ЭВМ Ноутбук Lenovo IdeaPad 3 Gaming 15ARH05

6. План выполнения КП

Название разделов работы	Сроки	Форма промежуточной отчетности	Примечания
Введение	06.02.2024–01.03.2024		
Теоретическая часть	02.03.2024–16.03.2024		
Практическая часть	17.03.2024–20.04.2024		
Заключение	21.04.2024–30.04.2024		
Защита курсового проекта	06.05.2024		

7. Руководитель работы _____ доцент к.ф.-м.н Филиппов Е.Г.
(подпись) (должность, звание, Ф.И.О.)

Дата выдачи задания « _____» _____ 2024 г.

Задание принял
к исполнению студент _____ Слинько И. А.
(подпись) (Ф.И.О)

Содержание

Введение

"A picture is worth a thousand words", – гласит английская поговорка. В русском языке есть аналог: *"Лучше один раз увидеть, чем сто раз услышать"*.

Эти пословицы отлично отражают идею о том, что изображения могут передавать информацию гораздо эффективнее, чем текст или числа. То же самое можно сказать и про данные; ведь подобно тому, как одна картинка может заменить тысячу слов, визуализация данных может помочь нам лучше понять сложные структуры и закономерности, скрытые в больших объёмах информации.

Визуализация данных – это процесс представления информации и аналитики в графическом формате. Этот подход широко используется в различных областях, таких как наука, бизнес, медицина, и др. Посредством визуализации мы можем выявлять тенденции, обнаруживать аномалии, делать выводы и принимать обоснованные решения на основе данных.

Искусство визуализации данных уходит корнями в далёкое прошлое, когда древние цивилизации использовали рисунки и карты для передачи информации. С появлением компьютеров в 20 веке визуализация данных стала более динамичной и мощной, предоставляя новые инструменты для анализа информации.

На сегодняшний день существует множество программных средств для создания качественных графиков и диаграмм, например:

- язык программирования и пакет прикладных программ *Matlab*;
- программа для работы с электронными таблицами, а также функциональный инструмент визуализации и анализа данных *MS Excel*;
- язык программирования для статистических вычислений и визуализации данных *R*;
- платформа для создания визуального контента *Visme* (*powered by AI*);
- программное обеспечение для интерактивной бизнес-аналитики и визуализации данных *Tableau*;
- высокоуровневый язык программирования общего назначения *Python*.

В данной курсовой работе было принято решение использовать язык программирования Python в качестве основного инструмента анализа данных. Этот выбор обусловлен его простотой и универсальностью, а также наличием библиотек, включая те, что предназначены для визуализации данных. Кроме того, я уже изучаю Python и планирую продолжить это в будущем, поскольку он обладает широкой популярностью в области *Data Science*.

1 Теоретическая часть

1.1 Введение в язык программирования Python

1.1.1 История Python

Гвидо ван Россум (*Guido van Rossum*) задумал Python в 1980-х годах, а приступил к его созданию в декабре 1989 года в центре математики и информатики в Нидерландах (*Centrum Wiskunde & Informatica – CWI*).

Изначально язык был полностью любительским проектом: ван Россум просто хотел чем-то занять себя на рождественских каникулах. Название языка было взято из комедийного телешоу *BBC "Летающий цирк Монти Пайтона*", большим поклонником которого являлся программист.

Язык Python был задуман как потомок языка программирования *ABC*, способный к обработке исключений и взаимодействию с операционной системой *Amoeba*. Python свободно распространялся через интернет и со временем у него появились последователи, заинтересованные в развитии этого языка программирования.

Гвидо опубликовал первую версию кода Python (версия 0.9.0) в 1991 году. Он уже включал в себя ряд полезных возможностей. Например, различные типы данных и функции для обработки ошибок.

В версии Python 1.0, выпущенной в 1994 году, были реализованы новые функции для простой обработки списка данных: сопоставление, фильтрация и сокращение.

Python 2.0 был выпущен 16 октября 2000 года с новыми полезными функциями для программистов, такими как поддержка символов *Unicode* и упрощённый способ циклического просмотра списка.

Python 3.0 вышел 3 декабря 2008 года. Эта версия включала функцию печати и дополнительную поддержку деления чисел и обработки ошибок.

Самая свежая версия вышла 2 октября 2023 года – Python 3.12.0

По данным *GitHub* Python занимает второе по популярности место, уступая только *JavaScript*. Он также пятикратно брал звание "*Programming language of the year*" (по версии *TIOBE Index*): в 2007, 2010, 2018, 2020 и 2021 годах.

1.1.2 Характеристики Python

Основными характеристиками языка программирования Python являются:

1) Объектная ориентированность

Поддержка ООП позволяет разрабатывать код в виде объектов, которые взаимодействуют друг с другом. Это делает код более модульным и повторно используемым.

2) Мультипарадигменность

Программист всё ещё может выбирать между аспектно-ориентированным, структурным, императивным, процедурным, функциональным программированием и другими парадигмами.

3) Интерпретируемость

Код на нём выполняется построчно, в режиме реального времени. Это свойство позволяет быстро исправлять и проверять код без необходимости компиляции.

4) Динамическая типизация

Тип переменной определяется автоматически, во время выполнения кода. Это упрощает процесс программирования и делает его гибким при работе с данными различного типа.

5) Кроссплатформенность

Программы на Python могут запускаться на различных операционных системах без изменений, что обеспечивает их переносимость и универсальность.

1.1.3 Плюсы и минусы Python

К **преимуществам** языка Python можно отнести:

1) Читабельность кода

Визуально простой синтаксис Python позволяет легко читать и понимать код на этом языке, что ускоряет разработку и поддержку проектов.

2) Простоту освоения

Python обладает простым и логичным синтаксисом, что делает его доступным даже для начинающих программистов.

3) Широкую поддержку и активное сообщество

Существует огромное количество библиотек и фреймворков для Python, а также активное сообщество разработчиков, готовых помочь и поделиться опытом

4) Сочетание с другими языками программирования.

Python легко способен интегрироваться с другими языками программирования, такими как *Java*, *C* и *C++*.

5) Масштабируемость

Возможность адаптации высокоуровневой логики позволяет проектам, разработанным на Python, масштабироваться и расширяться.

К **недостаткам** языка Python можно отнести:

1) Низкую производительность и ресурсоёмкость

Программирование на Python требует высоких вычислительных мощностей серверов и компьютеров, что делает его не таким быстрым, как хотелось бы. И поскольку Python является интерпретируемым языком, это тоже делает его медленнее, по сравнению с компилируемыми языками, такими как C++ или Java.

2) Глобальную блокировку интерпретатора (GIL)

Это способ синхронизации потоков, который используется в некоторых интерпретируемых языках программирования, например, в Python и Ruby. Хотя GIL является самым простым способом избежать конфликтов при одновременном обращении разных потоков к одним и тем же участкам памяти, у такого подхода есть недостаток – ограничение параллельности вычислений. Также он не позволяет достигать высокой эффективности вычислений при работе на многоядерных и мультипроцессорных системах.

3) Трудность переноса проектов на другие системы

Проблема возникает из-за зависимости языка программирования от сторонних библиотек и модулей. Эти библиотеки могут иметь зависимости от определенных версий Python, операционных систем, других библиотек и внешних программ.

1.1.4 Использование Python

Рассмотрим сферы, где применяется Python:

1) Веб-разработка (на стороне сервера)

Веб-разработка на стороне сервера включает в себя сложные серверные функции, с помощью которых веб-сайты отображают информацию для пользователя. Например, веб-сайты должны взаимодействовать с базами данных и другими веб-сайтами, а также защищать данные при их отправке по сети. Для веб-разработки на Python понадобится знание фреймворков. Наиболее популярные – *Django* и *Flask*.

2) Автоматизация с помощью скриптов

Язык скриптов – это язык программирования, который автоматизирует задачи, обычно выполняемые людьми. Программисты широко используют скрипты Python для автоматизации многих повседневных задач.

3) Разработка программного обеспечения

Разработчики ПО часто используют Python для различных задач разработки и программных приложений, среди которых: разработка прототипов ПО, разработка настольных приложений с использованием библиотек графического пользовательского интерфейса (ГПИ), разработка игр, управление программными проектами.

4) Тестирование программного обеспечения

Разработчики используют среды модульного тестирования Python (*Unittest, Robot* и *PyUnit*) для тестирования написанных функций. Тестировщики программного обеспечения используют Python для написания тестовых примеров для различных сценариев.

5) Data Science и Machine Learning

Python предоставляет множество инструментов для анализа и визуализации данных, таких как библиотеки *SciPy, pandas* и *Matplotlib*. Эти инструменты позволяют проводить научные вычисления, анализировать данные и строить графики для визуализации результатов.

Далее будут рассмотрены наиболее популярные библиотеки для визуализации данных. Но перед этим ознакомимся с видами диаграмм, которые будем строить.

1.2 Виды диаграмм

В данной курсовой работе будут представлены следующие виды диаграмм:

1) Гистограмма

Гистограмма – это визуальное представление распределения частоты или плотности значений в наборе данных. Она показывает, сколько раз каждое значение встречается в данных.

Гистограмма состоит из столбцов, где каждый столбец представляет диапазон значений, а высота столбца показывает частоту или плотность значений в этом диапазоне.

Гистограммы используются для оценки формы и характера распределения данных, выявления выбросов или аномалий, а также сравнения распределений между различными группами данных.

2) Столбчатая диаграмма

Столбчатая диаграмма – это график, который показывает относительные значения категорий, сравнивая их между собой.

Диаграмма состоит из вертикальных столбцов, где длина каждого столбца пропорциональна значению, которое он представляет.

Столбчатые диаграммы используются для наглядного сравнения нескольких категорий данных или изменений во времени.

3) Круговая диаграмма

Круговая диаграмма – это круговой график, который показывает долю каждой категории в общем объеме или сумме.

Диаграмма состоит из секторов круга, пропорциональных долям каждой категории. Угол каждого сектора соответствует доле от общей суммы.

Круговые диаграммы используются для иллюстрации структуры данных и быстрой оценки доли каждого элемента в целом. Они удобны для визуализации относительных величин.

4) Линейный график

Линейный график – это график, который показывает изменение значения переменной во времени или по другому параметру.

График состоит из точек, соединенных линиями. Каждая точка представляет значение переменной в определенный момент времени или при определенном условии.

Линейные графики используются для визуализации тенденций, трендов и изменений во времени. Они помогают анализировать и прогнозировать поведение данных.

5) График функции

График функции – это графическое представление зависимости одной переменной от другой в виде кривой линии.

График представляет собой набор точек, полученных путем подстановки значений переменной в функцию и построения соответствующих значений функции.

Графики функций используются для исследования и визуализации математических функций, а также для анализа их свойств и поведения.

6) Точечная диаграмма или диаграмма рассеяния

Точечная диаграмма – это график, который показывает отношение между двумя переменными, представленное в виде точек на плоскости.

График состоит из точек, каждая из которых представляет одно значение одной переменной по отношению к другой переменной.

Точечные диаграммы используются для определения наличия корреляции или взаимосвязи между двумя переменными и для выявления выбросов или аномалий в данных.

7) Диаграмма с областями

Диаграмма с областями – это график, который используется для визуализации изменения величины или структуры набора данных с течением времени или по другому параметру. Он представляет собой линию, которая окружена областью, заполненной цветом.

График состоит из линии, представляющей значение переменной, окруженной заливаемой областью, которая обычно представляет диапазон изменения этой переменной.

Диаграммы с областями используются для отображения изменений величины данных и иллюстрации диапазона значений переменной.

8) Радиальная (лепестковая) диаграмма

Радиальная диаграмма – это графическое представление данных в виде круговой диаграммы, в которой значения представлены секторами, выходящими из центра.

Диаграмма состоит из секторов, расположенных вокруг центральной точки. Углы секторов пропорциональны значениям, которые они представляют.

Радиальные диаграммы используются для отображения относительных долей категорий в общем объеме или сумме данных. Они обеспечивают наглядное представление структуры данных и помогают выявлять доминирующие или наиболее значимые категории.

2 Практическая часть

2.1 Matplotlib

2.1.1 Сведения о библиотеке

Matplotlib – это библиотека на языке программирования Python для визуализации данных двумерной и трехмерной графикой. Она позволяет создавать изображения, которые могут быть использованы для иллюстраций в публикациях, интерактивной графики, научных публикаций, пользовательских интерфейсов, веб-приложений и других сферах. Matplotlib начался как проект, подражающий графическим командам MATLAB, но в настоящее время является независимым от него проектом.

Matplotlib предоставляет гибкий и легко конфигурируемый пакет для визуализации данных. Он обладает процедурным интерфейсом pylab, предоставляющим аналоги команд MATLAB. Библиотека работает с несколькими графическими библиотеками, включая wxWindows и PyGTK. Matplotlib поддерживает множество видов графиков и диаграмм, включая графики, диаграммы рассеяния, столбчатые диаграммы, круговые диаграммы, контурные графики, спектральные диаграммы и многое другое.

Matplotlib позволяет указывать оси координат, решетку, добавлять надписи и пояснения, использовать логарифмическую шкалу или полярные координаты. Для трехмерных графиков используется набор инструментов mplot3d, а также существуют другие наборы инструментов для различных целей, такие как картография, работа с Excel и другие утилиты. Matplotlib также позволяет создавать анимированные изображения и поддерживает множество форматов файлов для сохранения изображений.

Matplotlib широко используется в научных исследованиях, инженерии, финансах, медицине, анализе данных и других областях, где требуется визуализация данных.

2.1.2 Гистограмма в Matplotlib

Рассмотрим построение гистограммы с помощью библиотеки Matplotlib.

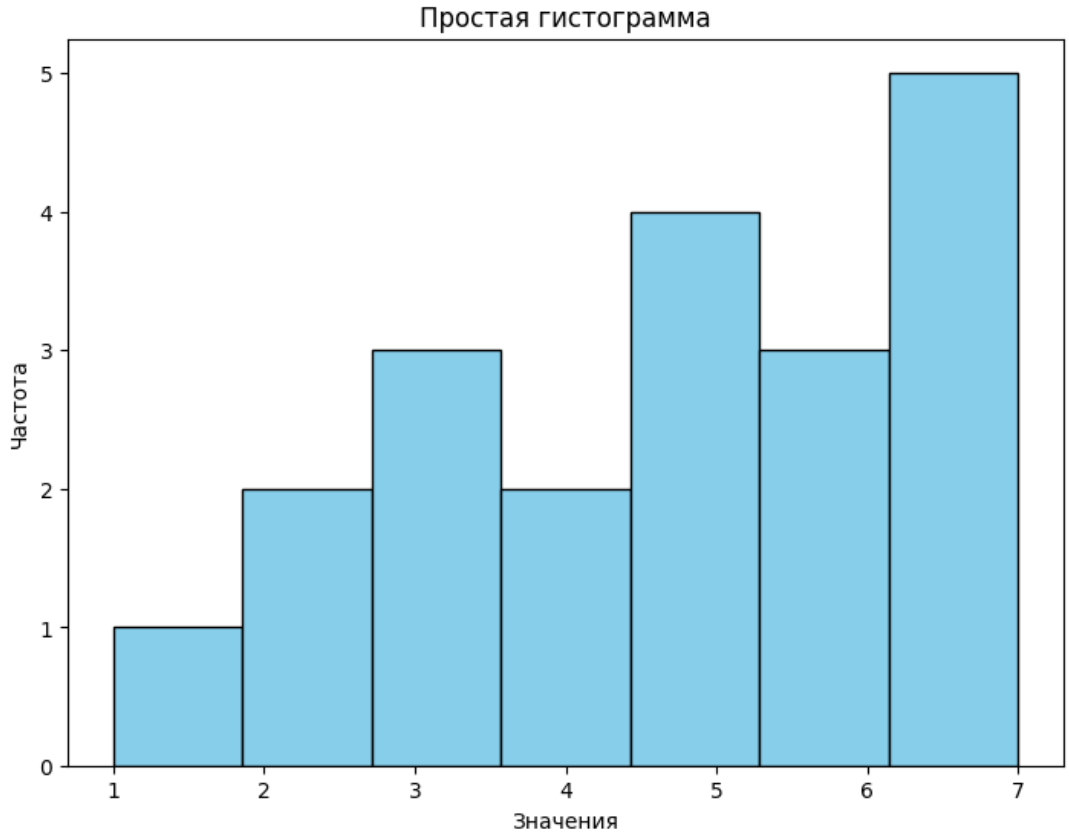

Рисунок 1.2.1 – Гистограмма в Matplotlib

Этот код создаёт гистограмму для набора данных *data*. Функция *hist()* используется для построения гистограммы, аргумент *bins* указывает количество столбцов в гистограмме. Также добавляем заголовок и подписи осей с помощью функций *title()* и *xlabel(), ylabel()*. И, наконец, функция *show()* используется для отображения графика.

```python
import matplotlib.pyplot as plt

# Данные для построения гистограммы
data = [1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7, 7, 7, 7]

# Построение гистограммы
plt.hist(data, bins=7, color='skyblue', edgecolor='black')

# Добавление заголовка и подписей осей
plt.title('Простая гистограмма')
plt.xlabel('Значения')
plt.ylabel('Частота')

# Отображение гистограммы
plt.show()
```

Рисунок 1.2.2 – Построение гистограммы в Matplotlib

2.1.3 Столбчатая диаграмма в Matplotlib

Рассмотрим построение столбчатой диаграммы с помощью библиотеки Matplotlib.

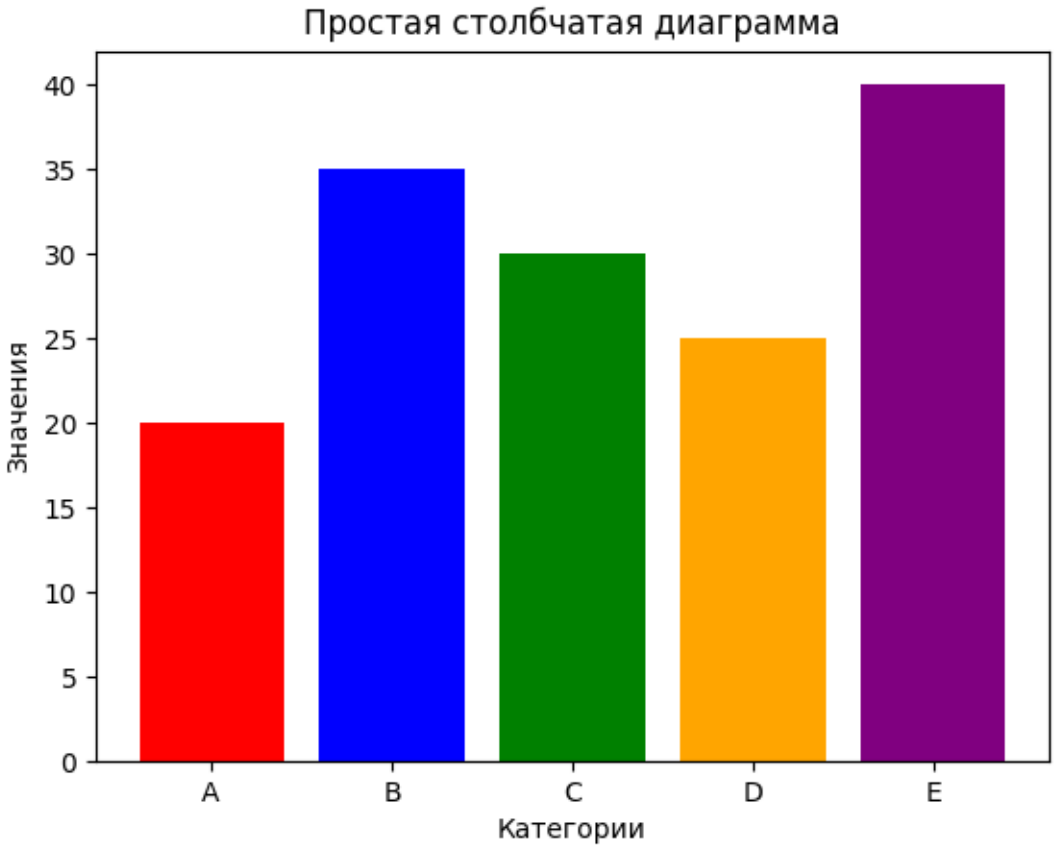

Рисунок 1.3.1 – Столбчатая диаграмма в Matplotlib

Этот код создает столбчатую диаграмму с категориями, представленными в виде списка *categories*, и соответствующими значениями в списке *values*. Функция *bar()* используется для построения столбцов диаграммы. Также добавляем заголовок и подписи осей с помощью функций *title()*, *xlabel()*, *ylabel()*. И, наконец, функция *show()* используется для отображения графика.

```python
import matplotlib.pyplot as plt

# Данные для столбчатой диаграммы
categories = ['A', 'B', 'C', 'D', 'E']
values = [20, 35, 30, 25, 40]

# Цвета для каждого прямоугольника
colors = ['red', 'blue', 'green', 'orange', 'purple']

# Построение столбчатой диаграммы с разноцветными прямоугольниками
plt.bar(categories, values, color=colors)

# Добавление заголовка и подписей осей
plt.title('Простая столбчатая диаграмма')
plt.xlabel('Категории')
plt.ylabel('Значения')

# Отображение столбчатой диаграммы
plt.show()
```

Рисунок 1.3.2 – Построение столбчатой диаграммы в Matplotlib

2.1.4 Круговая диаграмма в Matplotlib

Рассмотрим построение круговой диаграммы с помощью библиотеки Matplotlib.

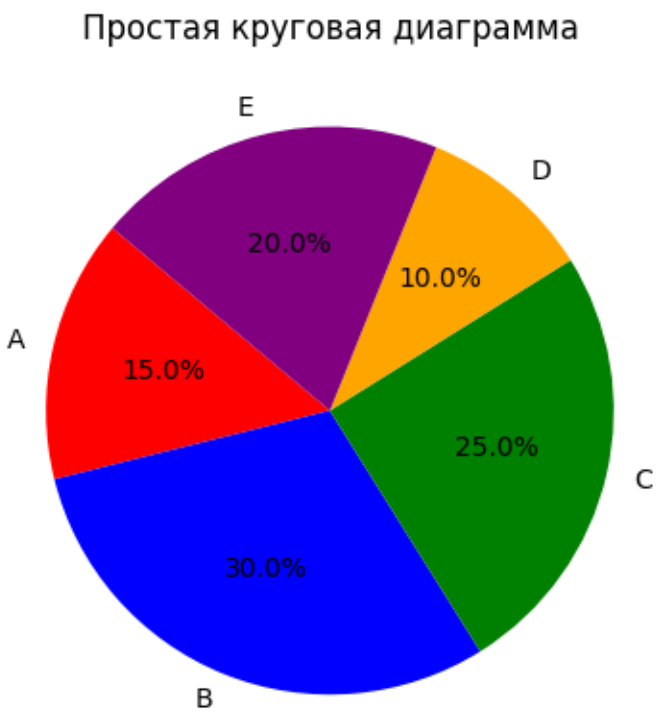

Рисунок 1.4.1 – Круговая диаграмма в Matplotlib

Этот код создает круговую диаграмму, где каждый сегмент представляет собой категорию из списка *labels*, а размер каждого сегмента определяется значениями из списка *sizes*. Цвета для каждого сегмента задаются в списке *colors*. Функция *autopct* добавляет процентное значение каждого сегмента, а *startangle* устанавливает начальный угол, с которого начинается построение круговой диаграммы.

```python
import matplotlib.pyplot as plt

# Данные для круговой диаграммы
labels = ['A', 'B', 'C', 'D', 'E']
sizes = [15, 30, 25, 10, 20]

# Цвета для каждого сегмента круговой диаграммы
colors = ['red', 'blue', 'green', 'orange', 'purple']

# Построение круговой диаграммы
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)

# Добавление заголовка
plt.title('Простая круговая диаграмма')

# Отображение круговой диаграммы
plt.show()
```

Рисунок 1.4.2 – Построение круговой диаграммы в Matplotlib

2.1.5 Линейный график в Matplotlib

Рассмотрим построение линейного графика с помощью библиотеки Matplotlib.

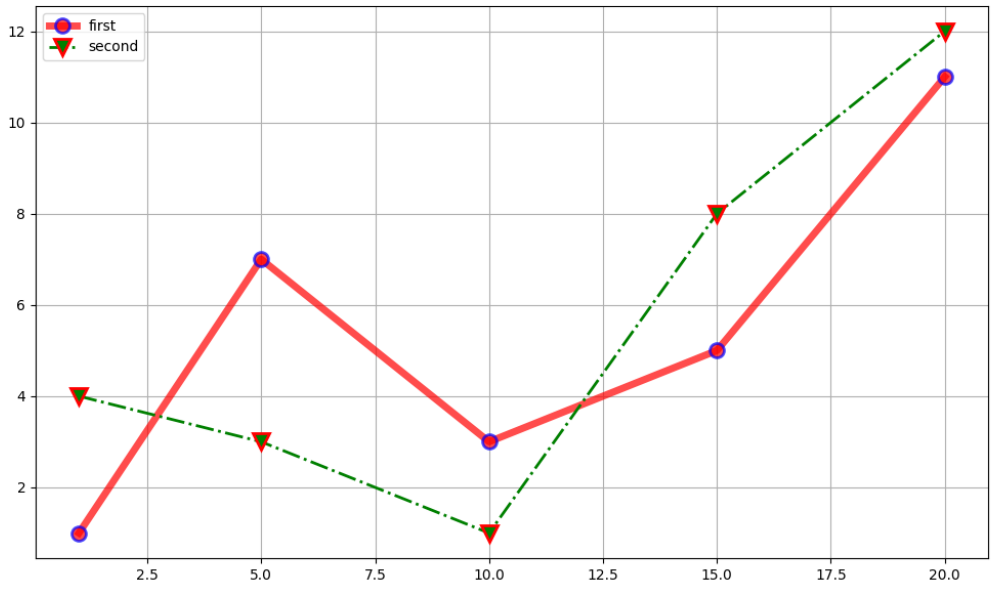

Рисунок 1.5.1 – Линейный график в Matplotlib

Этот код создает линейный график с использованием библиотеки Matplotlib на языке программирования Python.

Вначале определяются данные для оси X и два набора данных для оси Y. Затем создается новая фигура для графика с заданным размером. Далее строятся две линии на графике с использованием функции plt.plot(), каждая из которых представляет один из наборов данных.

Первая линия обозначена красными круглыми маркерами и имеет толщину линии 5 пикселей. Вторая линия обозначена зелеными треугольными маркерами и имеет штрих-пунктирную линию. Обе линии также имеют метки для легенды.

После построения линий добавляется легенда, чтобы обозначить, какие данные соответствуют каким линиям. Затем включается сетка на графике с помощью plt.grid(True).

Наконец, график отображается с помощью plt.show(), позволяя пользователю увидеть итоговую визуализацию данных..

```python
import matplotlib.pyplot as plt  # Импортируем библиотеку для визуализации дан

# Задаем данные для оси X и два набора данных для оси Y
x = [1, 5, 10, 15, 20]
y1 = [1, 7, 3, 5, 11]
y2 = [4, 3, 1, 8, 12]

# Создаем новую фигуру для графика и задаем ее размер
plt.figure(figsize=(12, 7))

# Строим график для первого набора данных (y1)
plt.plot(x, y1, 'o-r', alpha=0.7, label="first", lw=5, mec='b', mew=2, ms=10)

# Строим график для второго набора данных (y2)
plt.plot(x, y2, 'v-.g', label="second", mec='r', lw=2, mew=2, ms=12)

# Добавляем легенду
plt.legend()

# Включаем сетку на графике
plt.grid(True)

# Отображаем график
plt.show()
```

Рисунок 1.5.2 – Построение линейного графика в Matplotlib

2.1.6 График функции в Matplotlib

Рассмотрим построение графика функции с помощью библиотеки Matplotlib.

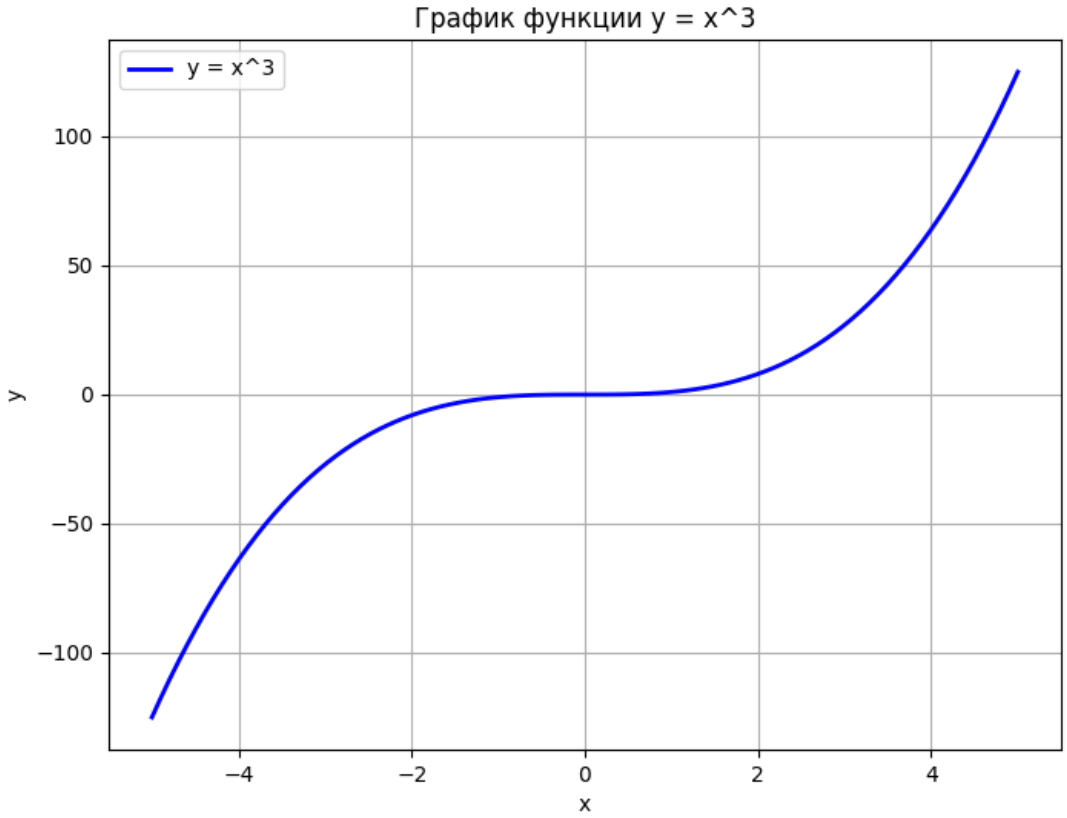

Рисунок 1.6.1 – График функции $y = x^3$ в Matplotlib

Этот код использует библиотеки Matplotlib и *NumPy* для создания графика функции $y = x^3$.

Сначала он импортирует эти библиотеки. NumPy используется для создания массива значений x с помощью *np.linspace()*. Далее задаются функциональные зависимости x и y в соответствии с уравнением $y = x^3$.

Затем функция *plt.plot()* используется для построения линейного графика функции $y = x^3$ с использованием массивов x и y.

График имеет определенный цвет (синий по умолчанию), толщину линии (по умолчанию), а также метку "$y = x^3$" для включения в легенду. После этого добавляются метки осей и заголовок графика с помощью функций *plt.xlabel()*, *plt.ylabel()* и *plt.title()*. Включается сетка для лучшей визуализации с помощью *plt.grid(True)*.

Наконец, вызывается *plt.show()*, чтобы отобразить построенный график.

```python
import numpy as np
import matplotlib.pyplot as plt

# Задаем диапазон значений по оси X
x = np.linspace(-5, 5, 100)

# Вычисляем значения функции y = x^3
y = x ** 3

# Создаем новую фигуру
plt.figure(figsize=(8, 6))

# Строим график функции
plt.plot(x, y, label='y = x^3', color='b', linestyle='-', linewidth=2)

# Добавляем заголовок и подписи к осям
plt.title('График функции y = x^3')
plt.xlabel('x')
plt.ylabel('y')

# Включаем сетку
plt.grid(True)

# Добавляем легенду
plt.legend()

# Отображаем график
plt.show()
```

Рисунок 1.6.2 – Построение графика функции $y = x^3$ в Matplotlib

2.1.7 Диаграмма рассеяния в Matplotlib

Рассмотрим построение диаграммы рассеяния с линией регрессии с помощью библиотеки Matplotlib.

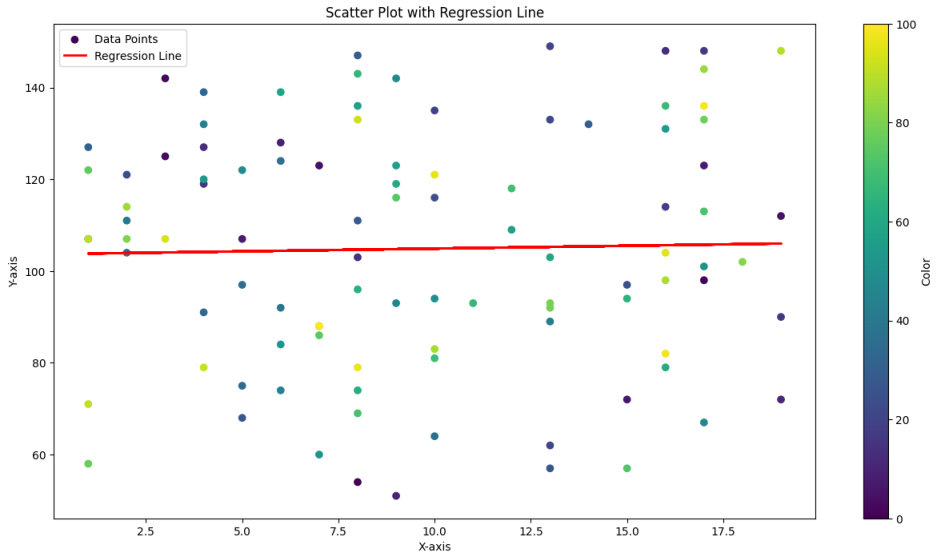

Рисунок 1.7.1 – Диаграмма рассеяния с линией регрессии в Matplotlib

Этот код строит точечную диаграмму с линией регрессии и цветовой шкалой.

В начале кода импортируются необходимые библиотеки: *matplotlib.pyplot* для визуализации данных и *numpy* для работы с массивами чисел.

Затем создаются случайные данные для оси X и оси Y, а также массив цветов для каждой точки на графике. Функция *plt.scatter()* используется для построения точечной диаграммы. Аргумент c указывает массив цветов, а *cmap* задает цветовую карту для интерпретации значений цветов.

Создается модель линейной регрессии с помощью *LinearRegression()* из библиотеки *sklearn*. Модель обучается на данных точечной диаграммы, и предсказанные значения *y_pred* вычисляются для построения линии регрессии.

Функция *plt.plot()* используется для построения линии регрессии на графике. Цвет линии задается как красный, толщина линии - 2 пикселя.

Цветовая шкала добавляется с помощью *plt.colorbar()*. Заголовок шкалы указывается с помощью аргумента *label*.

Заголовок и метки осей устанавливаются с помощью функций *plt.title()*, *plt.xlabel()* и *plt.ylabel()*. Легенда включается с помощью *plt.legend()*.

Наконец, вызывается *plt.show()* для отображения графика.

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

# Создание данных
x = np.random.randint(1, 20, size=100)  # Генерация 100 случайных чисел для оси X
y = np.random.randint(50, 150, size=100)  # Генерация 100 случайных чисел для оси Y
colors = np.linspace(0, 100, 100)  # Генерация 100 цветов от 0 до 100

# Построение точечной диаграммы
plt.scatter(x, y, c=colors, cmap='viridis', label='Data Points')

# Создание модели линейной регрессии
model = LinearRegression()
model.fit(x.reshape(-1, 1), y)
y_pred = model.predict(x.reshape(-1, 1))

# Построение линии регрессии
plt.plot(x, y_pred, color='red', linewidth=2, label='Regression Line')

# Добавление цветовой шкалы
plt.colorbar(label='Color')

# Добавление заголовка и меток осей
plt.title('Scatter Plot with Regression Line')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
```

Рисунок 1.7.2 – Построение диаграммы рассеяния с линией регрессии в Matplotlib

2.1.8 Диаграмма с областями в Matplotlib

Рассмотрим построение диаграммы с областями с помощью библиотеки Matplotlib.

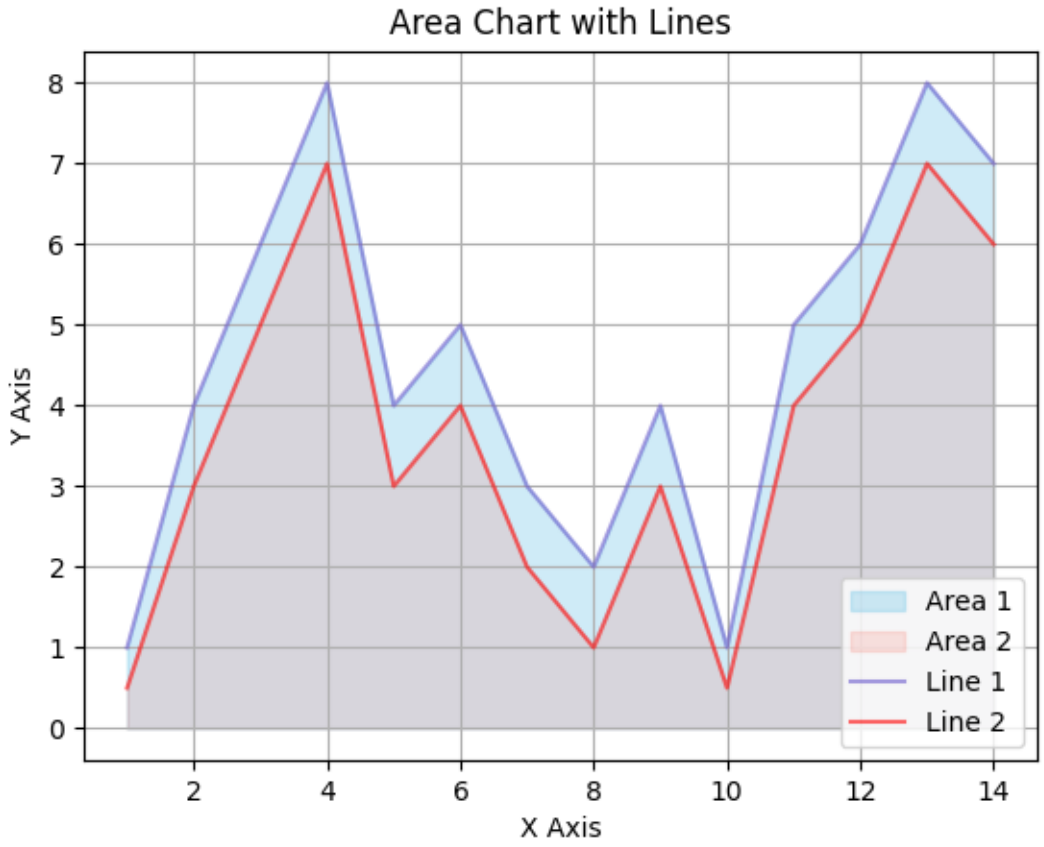

Рисунок 1.8.1 – Диаграмма с областями в Matplotlib

Этот код используется для создания графика с двумя областями и линиями на одном поле.

Переменные x, y_1 и y_2 представляют данные для осей X и Y для двух различных графиков. Функция fill_between() используется для заполнения областей под графиками с разной прозрачностью. Первый вызов заполняет область под графиком y1 цветом "skyblue" с прозрачностью 0.4, а второй вызов заполняет область под графиком y2 цветом "salmon" с прозрачностью 0.2.

Функция plot() используется для построения линий графиков y_1 и y_2.

Линии графиков окрашены в цвета "Slateblue" и "red" соответственно с альфа-каналом 0.6. Функция grid(True) включает сетку на графике.

Наконец, вызывается show() для отображения графика.

Для добавления легенды к графику и названий осей X и Y можно использовать функции xlabel(), ylabel() и legend().

```python
import numpy as np
import matplotlib.pyplot as plt

# Создание данных
x = range(1, 15)
y1 = [1, 4, 6, 8, 4, 5, 3, 2, 4, 1, 5, 6, 8, 7]
y2 = [0.5, 3, 5, 7, 3, 4, 2, 1, 3, 0.5, 4, 5, 7, 6]

# Заполнение областей под графиками с разной прозрачностью
plt.fill_between(x, y1, color="skyblue", alpha=0.4)
plt.fill_between(x, y2, color="salmon", alpha=0.2)

# Построение линий для обозначения данных
plt.plot(x, y1, color="Slateblue", alpha=0.6)
plt.plot(x, y2, color="red", alpha=0.6)

# Настройка сетки
plt.grid(True)

# Отображение графика
plt.show()

```

Рисунок 1.8.2 – Построение диаграммы с областями в Matplotlib

2.1.9 Радиальная диаграмма в Matplotlib

Рассмотрим построение радиальной диаграммы с помощью библиотеки Matplotlib.

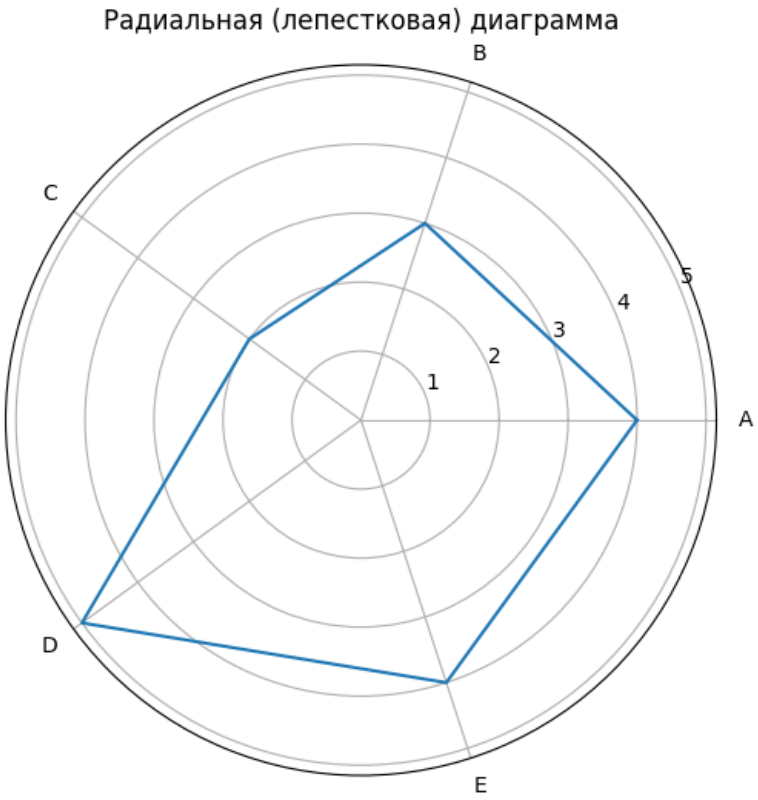

Рисунок 1.9.1 – Радиальная (лепестковая) диаграмма в Matplotlib

Этот код использует библиотеки NumPy и Matplotlib для создания радиальной (лепестковой) диаграммы.

В начале мы определяем категории и их значения. Затем мы генерируем углы для каждой категории с помощью *np.linspace()*, чтобы равномерно распределить их по кругу.

Мы затем "закрываем" круг, добавляя первое значение к концу списка значений и первый угол к концу списка углов. Это делается для того, чтобы круг был полностью закрыт. Далее мы используем функцию *plt.polar()*, чтобы построить радиальную диаграмму, передавая углы и значения как аргументы.

Чтобы добавить подписи категорий к осям, мы используем *plt.xticks()*, передавая углы и категории как аргументы.

Наконец, мы добавляем заголовок к графику с помощью *plt.title()*, указывая на то, что это радиальная (лепестковая) диаграмма.

Затем мы отображаем график с помощью *plt.show()*.

```python
import numpy as np
import matplotlib.pyplot as plt

# Создание данных
categories = ['A', 'B', 'C', 'D', 'E']
values = [4, 3, 2, 5, 4]

# Подготовка углов для лепестков
angles = np.linspace(0, 2 * np.pi, len(categories), endpoint=False).tolist()

# Закрываем круг
values += values[:1]
angles += angles[:1]

# Построение радиальной диаграммы
plt.figure(figsize=(6, 6))
plt.polar(angles, values)

# Добавление названий категорий
plt.xticks(angles[:-1], categories)

# Настройка заголовка
plt.title('Радиальная (лепестковая) диаграмма')

# Отображение графика
plt.show()
```

Рисунок 1.9.2 – Построение радиальной (лепестковой) диаграммы в Matplotlib

2.2 Plotly

2.2.1 Сведения о библиотеке

Plotly – это библиотека на языке программирования Python для визуализации данных, предоставляющая широкий спектр возможностей для создания интерактивных графиков и диаграмм. С ее помощью можно создавать как двумерные, так и трехмерные графики, а также добавлять анимацию и элементы управления пользовательского интерфейса.

Plotly легко интегрируется с другими библиотеками Python, такими как Pandas и NumPy, что делает ее удобным инструментом для анализа и визуализации данных.

Она также поддерживает множество форматов данных, включая CSV, Excel, JSON и SQL, что обеспечивает гибкость в работе с различными источниками данных.

Plotly широко используется в научных исследованиях, финансовом анализе, бизнес-аналитике, медицине, образовании и других областях, где требуется визуализация данных с возможностью интерактивного взаимодействия. Благодаря своей гибкости и функциональности, Plotly является популярным выбором среди аналитиков данных и разработчиков по всему миру.

2.2.2 Гистограмма в Plotly

Рассмотрим построение гистограммы с помощью библиотеки Plotly.

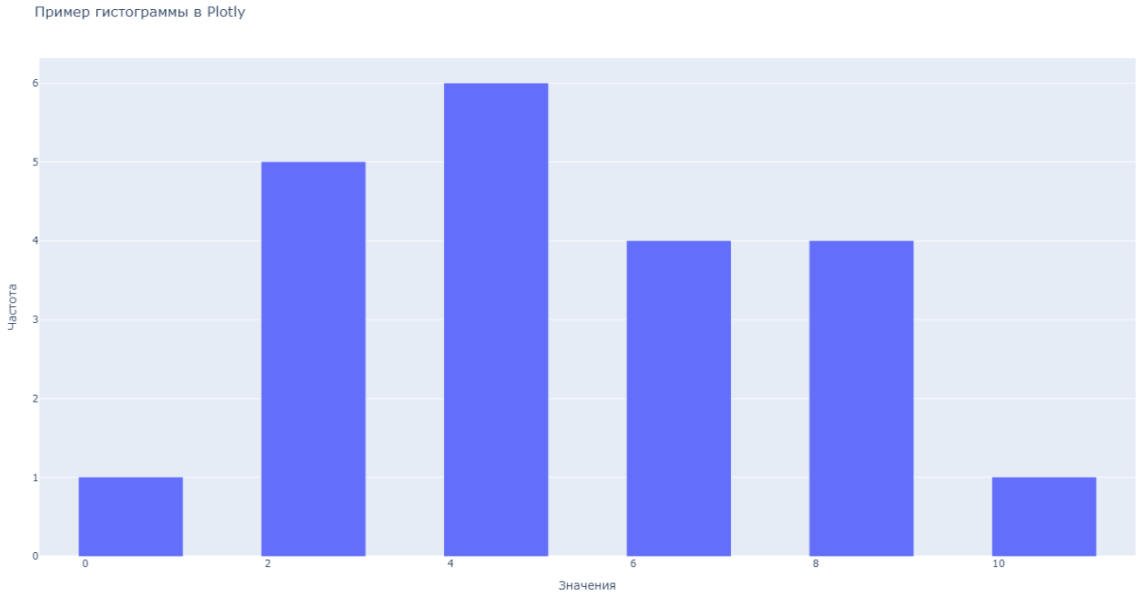

Рисунок 2.2.1 – Гистограмма в Plotly

Этот код создает гистограмму в библиотеке Plotly с использованием предоставленных данных.

import plotly.graph_objects as go: Эта строка импортирует необходимый класс *Figure* из библиотеки Plotly.

x = [1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 6, 7, 7, 7, 8, 8, 8, 9, 10]: Этот список x содержит значения, которые будут использоваться для построения гистограммы.

fig = go.Figure(data=[go.Histogram(x=x)]): Здесь создается объект Figure с данными в виде гистограммы. Объект *Histogram* принимает аргумент x, который указывает на данные для построения гистограммы.

fig.update_layout(...): Этот метод обновляет макет графика, включая заголовок (*title*) и подписи осей (*xaxis* и *yaxis*). В данном случае добавляется заголовок графика, а также названия осей X и Y.

fig.show(): Здесь вызывается метод *show()*, чтобы отобразить график в интерактивном режиме.

```python
import plotly.graph_objects as go

# Создание данных для гистограммы
x = [1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 6, 7, 7, 7, 8, 8, 8, 9, 10]
# Построение гистограммы
fig = go.Figure(data=[go.Histogram(x=x)])

# Настройка осей и макета
fig.update_layout(
    title='Пример гистограммы в Plotly',
    xaxis=dict(title='Значения'),
    yaxis=dict(title='Частота'),
    bargap=0.01,  # Промежуток между столбцами
    bargroupgap=0.05  # Промежуток между группами столбцов
)

# Отображение графика
fig.show()
```

Рисунок 2.2.2 – Построение гистограммы в Plotly

2.2.3 Столбчатая диаграмма в Plotly

Рассмотрим построение столбчатой диаграммы с помощью библиотеки Plotly.

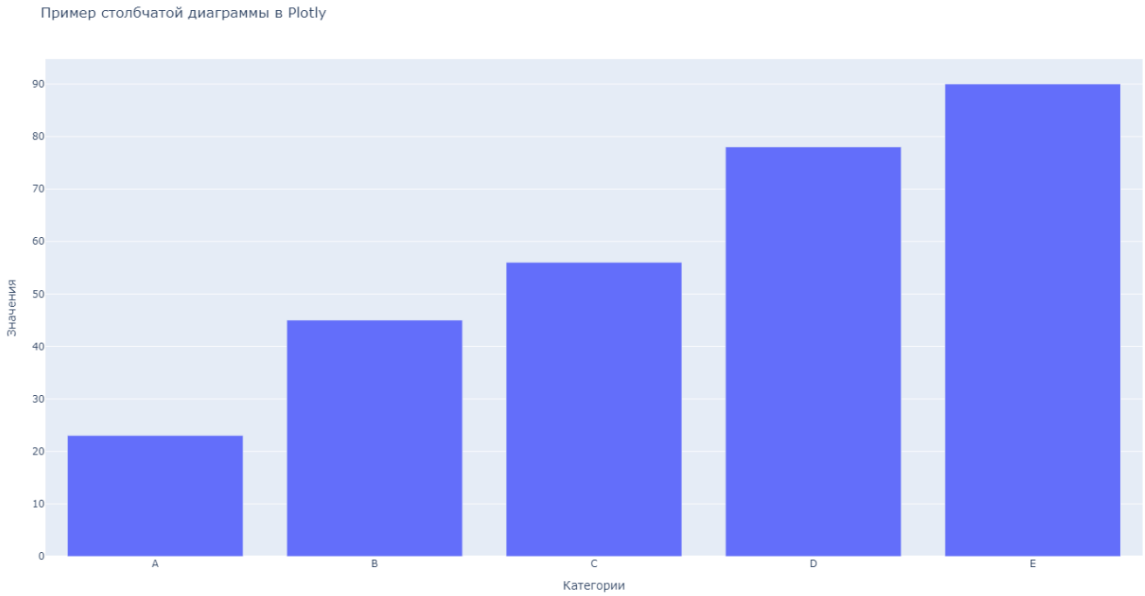

Рисунок 2.3.1 – Столбчатая диаграмма в Plotly

Этот скрипт использует библиотеку Plotly для создания столбчатой диаграммы.

На первом шаге мы создаем данные для диаграммы, представленные в двух списках: *categories*, содержащем категории, и *values*, содержащем значения для каждой категории.

Затем мы используем эти данные для построения столбчатой диаграммы, где ось X представляет собой категории, а ось Y - их соответствующие значения.

Для настройки внешнего вида диаграммы мы добавляем заголовок и подписи к осям X и Y.

Наконец, мы отображаем график, используя метод *show()*.

```python
import plotly.graph_objects as go

# Создание данных для столбчатой диаграммы
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 90]

# Построение столбчатой диаграммы
fig = go.Figure(data=[go.Bar(x=categories, y=values)])

# Настройка осей и макета
fig.update_layout(
    title='Пример столбчатой диаграммы в Plotly',
    xaxis=dict(title='Категории'),
    yaxis=dict(title='Значения'),
)

# Отображение графика
fig.show()
```

Рисунок 2.3.2 – Построение столбчатой диаграммы в Plotly

2.2.4 Круговая диаграмма в Plotly

Рассмотрим построение круговой диаграммы с помощью библиотеки Plotly.

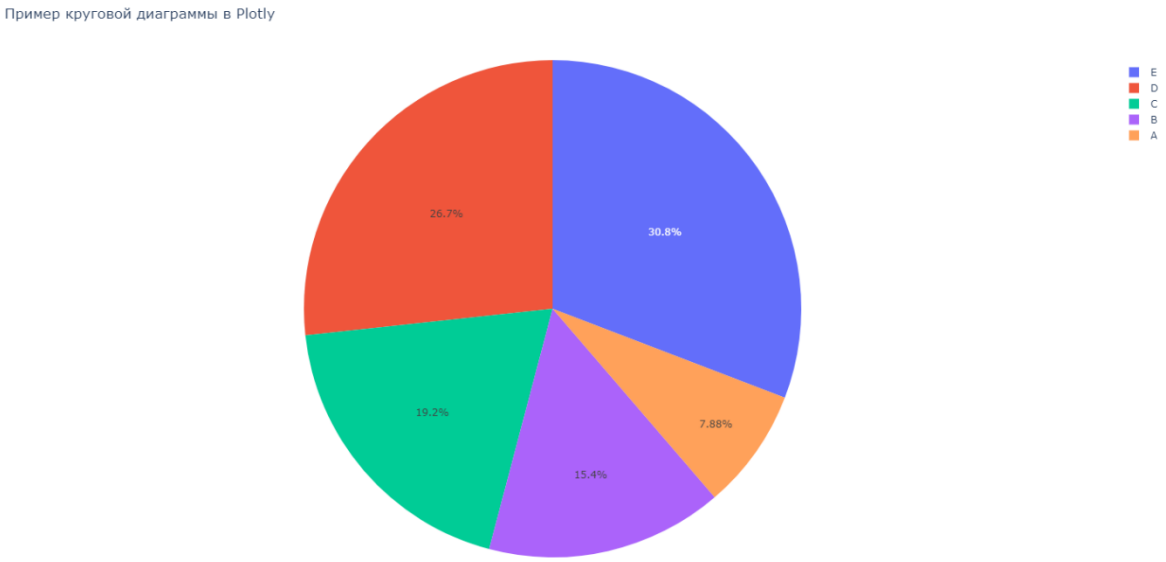

Рисунок 2.4.1 – Круговая диаграмма в Plotly

Этот код создает круговую диаграмму, используя данные *labels* и *values*, где *labels* содержит подписи секторов, а *values* - их соответствующие значения.

Затем мы строим круговую диаграмму с помощью *go.Pie()*, передавая ей эти данные.

Наконец, мы добавляем заголовок к диаграмме и отображаем ее с помощью метода *show()*.

```python
import plotly.graph_objects as go

# Создание данных для круговой диаграммы
labels = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 90]

# Построение круговой диаграммы
fig = go.Figure(data=[go.Pie(labels=labels, values=values)])

# Настройка макета
fig.update_layout(
    title='Пример круговой диаграммы в Plotly'
)

# Отображение графика
fig.show()
```

Рисунок 2.4.2 – Построение круговой диаграммы в Plotly

2.2.5 Линейный график в Plotly

Рассмотрим построение линейного графика с помощью библиотеки Plotly.

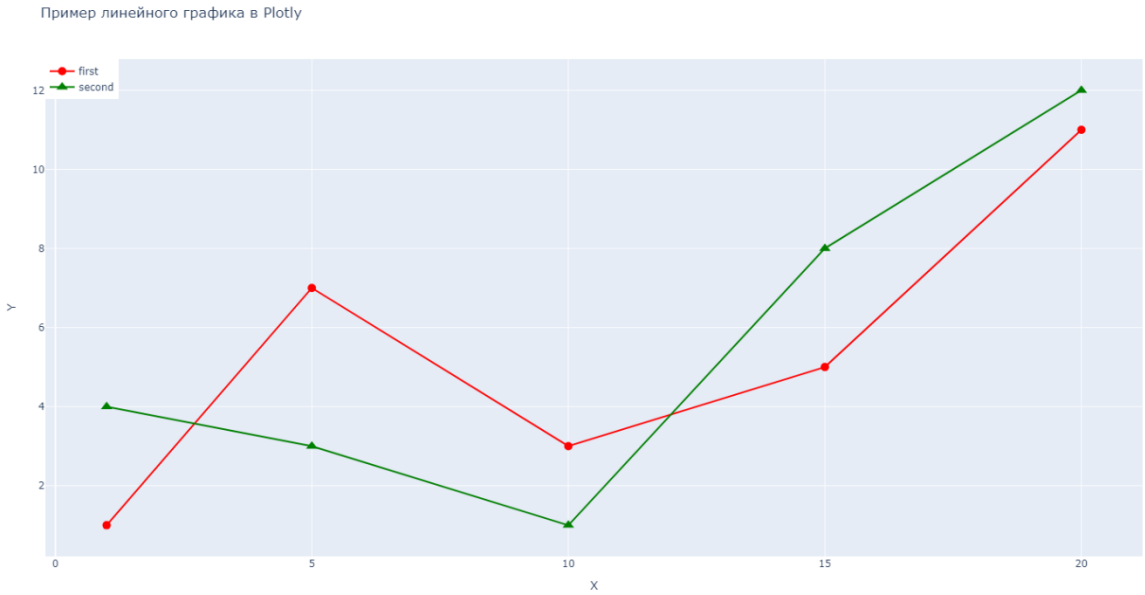

Рисунок 2.5.1 – Линейный график в Plotly

Этот код создает линейный график в Plotly, используя данные x, y_1 и y_2.

Мы добавляем два следа (*go.Scatter*), один для каждого набора данных, и настраиваем их внешний вид (цвет, размер, символ).

Затем мы добавляем заголовок к графику и отображаем его с помощью метода *show()*.

```python
import plotly.graph_objects as go

x = [1, 5, 10, 15, 20]
y1 = [1, 7, 3, 5, 11]
y2 = [4, 3, 1, 8, 12]

# Построение линейного графика
fig = go.Figure()

fig.add_trace(go.Scatter(x=x, y=y1, mode='markers+lines', name='first', marker=dict(color='red', size=10)))
fig.add_trace(go.Scatter(x=x, y=y2, mode='markers+lines', name='second', marker=dict(color='green', size=12, symbol='triangle-up')))

# Настройка макета
fig.update_layout(
    title='Пример линейного графика в Plotly',
    xaxis=dict(title='X'),
    yaxis=dict(title='Y'),
    legend=dict(x=0, y=1, traceorder='normal')
)

# Отображение графика
fig.show()
```

Рисунок 2.5.2 – Построение линейного графика в Plotly

2.2.6 График функции в Plotly

Рассмотрим построение графика функции с помощью библиотеки Plotly.

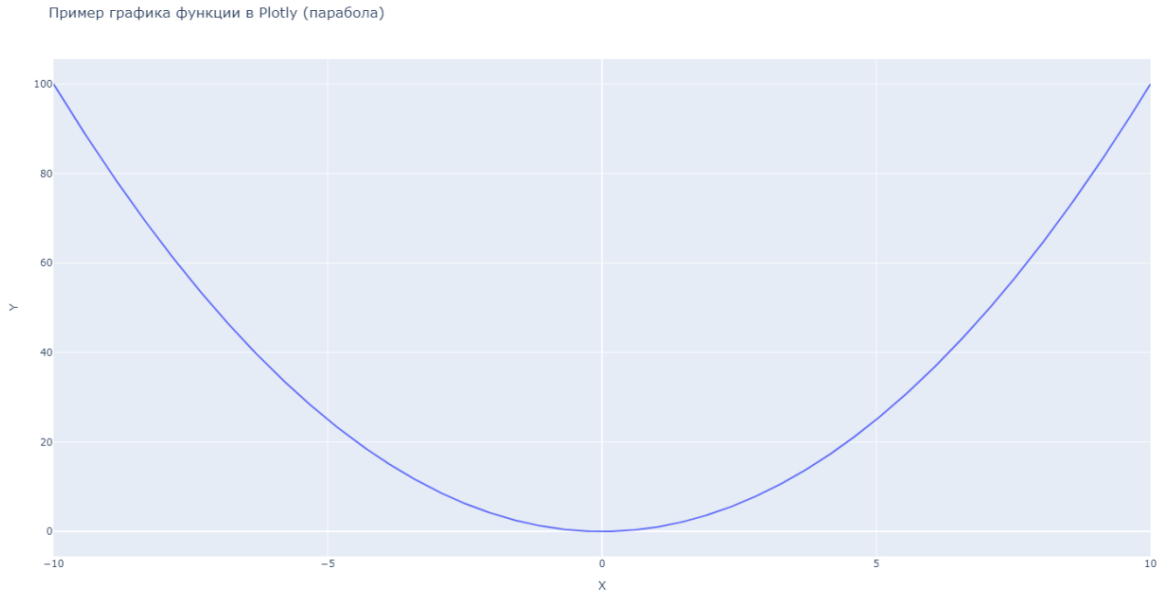

Рисунок 2.6.1 – График функции $y = x^2$ в Plotly

Этот код создает график функции $y = x^2$ (парабола) в Plotly.

Мы используем библиотеку NumPy для создания массива значений x в диапазоне от -10 до 10 с 400 точками.

Затем мы вычисляем значения y как квадрат от x.

Далее мы добавляем линию графика с помощью *go.Scatter* и настраиваем макет графика.

Наконец, мы отображаем график с помощью метода *show()*.

```python
import numpy as np
import plotly.graph_objects as go

# Создание данных для графика
x = np.linspace(-10, 10, 400)
y = x ** 2

# Построение графика функции
fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=y, mode='lines', name='y = x^2'))

# Настройка макета
fig.update_layout(
    title='Пример графика функции в Plotly (парабола)',
    xaxis=dict(title='X'),
    yaxis=dict(title='Y'),
)

# Отображение графика
fig.show()
```

Рисунок 2.6.2 – Построение графика функции $y = x^2$ в Plotly

2.2.7 Точечная диаграмма в Plotly

Рассмотрим построение точечной диаграммы с помощью библиотеки Plotly.

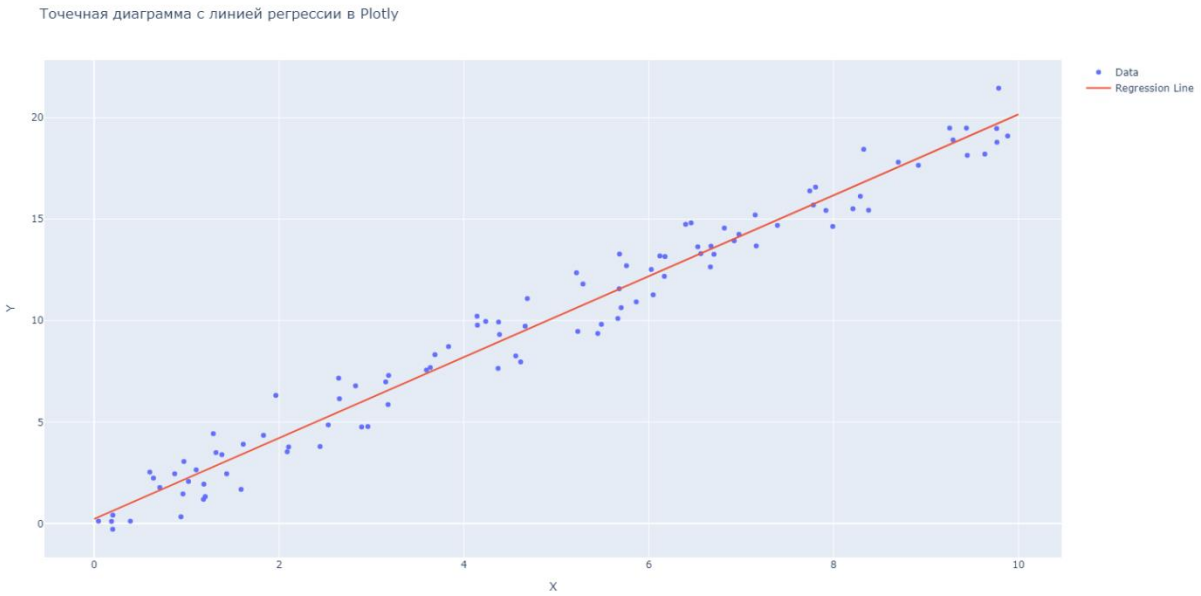

Рисунок 2.7.1 – Диаграмма рассеяния с линией регрессии в Plotly

В этом примере мы создаем случайные данные x и y для точечной диаграммы, где y зависит от x с добавлением нормального шума.

Затем мы вычисляем коэффициенты линейной регрессии с помощью функции *linregress* из библиотеки *SciPy*.

Мы строим точечную диаграмму и линию регрессии с помощью *go.Scatter*, а затем настраиваем макет графика и отображаем его с помощью метода *show()*.

```python
import numpy as np
import plotly.graph_objects as go
from scipy import stats

# Создание данных для точечной диаграммы
np.random.seed(0)
x = np.random.rand(100) * 10
y = 2 * x + np.random.randn(100)

# Вычисление коэффициентов линейной регрессии
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

# Построение точечной диаграммы
fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=y, mode='markers', name='Data'))

# Построение линии регрессии
x_range = np.linspace(0, 10, 100)
y_regression = slope * x_range + intercept
fig.add_trace(go.Scatter(x=x_range, y=y_regression, mode='lines', name='Regression Line'))

# Настройка макета
fig.update_layout(
    title='Точечная диаграмма с линией регрессии в Plotly',
    xaxis=dict(title='X'),
    yaxis=dict(title='Y'),
)

# Отображение графика
fig.show()
```

Рисунок 2.7.2 – Построение диаграммы рассеяния с линией регрессии в Plotly

2.2.8 Диаграмма с областями в Plotly

Рассмотрим построение диаграммы с областями с помощью библиотеки Plotly.

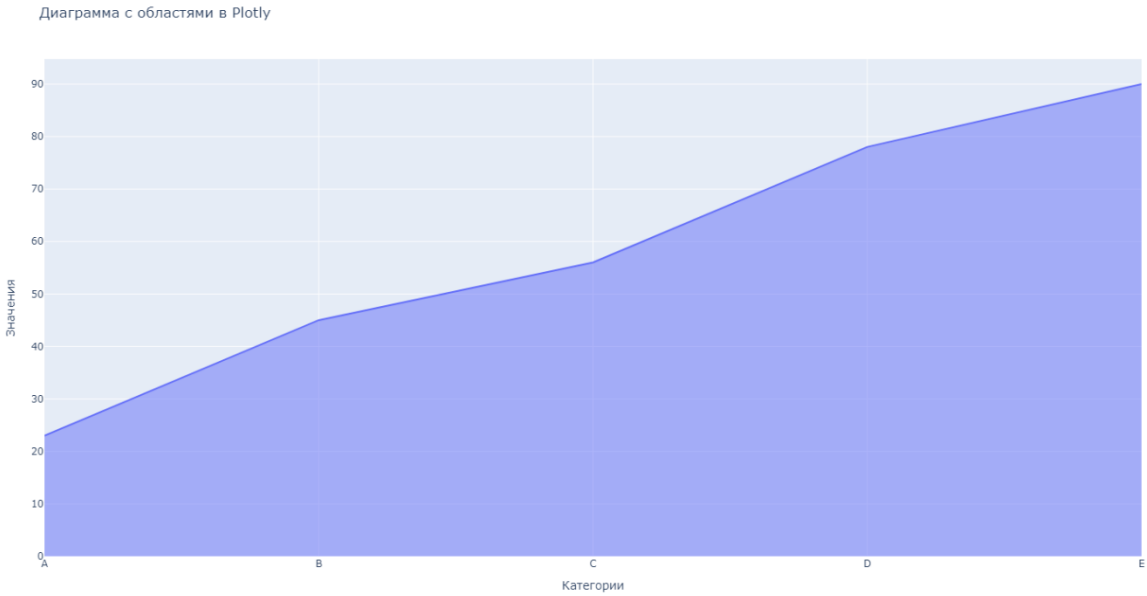

Рисунок 2.8.1 – Диаграмма с областями в Plotly

В этом примере мы создаем диаграмму с областями, где каждая область под кривой линии заполняется цветом.

Мы используем функцию *go.Scatter* для создания линейного графика с параметром *fill='tozeroy'*, чтобы заполнить область под кривой.

Затем мы настраиваем макет графика с помощью метода *update_layout()* и отображаем график с помощью метода *show()*.

```
import plotly.graph_objects as go

# Создание данных
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 90]

# Построение диаграммы с областями
fig = go.Figure(data=[go.Scatter(x=categories, y=values, mode='lines', fill='tozeroy')])

# Настройка макета
fig.update_layout(
    title='Диаграмма с областями в Plotly',
    xaxis=dict(title='Категории'),
    yaxis=dict(title='Значения'),
)

# Отображение графика
fig.show()
```

Рисунок 2.8.2 – Построение диаграммы с областями в Plotly

2.2.9 Радиальная диаграмма в Plotly

Рассмотрим построение радиальной диаграммы с помощью библиотеки Plotly.

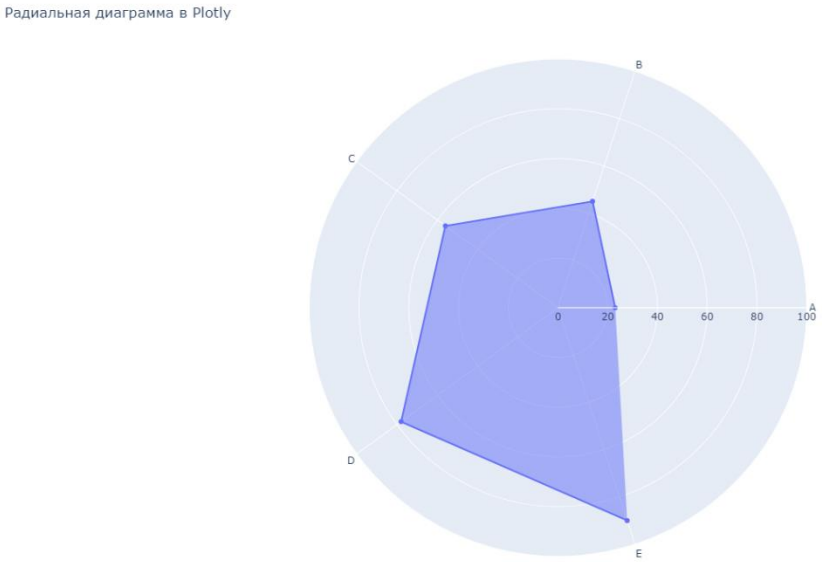

Рисунок 2.9.1 – Радиальная (лепестковая) диаграмма в Plotly

В этом примере мы используем функцию *go.Scatterpolar*, чтобы построить радиальную (лепестковую) диаграмму. Мы передаем значения r (радиус) и theta (углы) для каждой категории. Параметр fill='toself' используется для заполнения области под кривой. Затем мы настраиваем макет графика с помощью метода *update_layout()* и отображаем график с помощью метода *show()*.

```
import plotly.graph_objects as go

# Создание данных
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 90]

# Построение радиальной (лепестковой) диаграммы
fig = go.Figure(data=go.Scatterpolar(r=values, theta=categories, fill='toself'))

# Настройка макета
fig.update_layout(
    polar=dict(radialaxis=dict(visible=True, range=[0, 100])),
    showlegend=False,
    title='Радиальная диаграмма в Plotly'
)

# Отображение графика
fig.show()
```

Рисунок 2.9.2 – Построение радиальной (лепестковой) диаграммы в Plotly

Заключение

Визуализация данных играет ключевую роль в современном анализе данных. С помощью неё мы можем превратить большие объемы информации в понятные и наглядные графики, что позволяет обнаружить закономерности, тренды и взаимосвязи, а также делать информированные решения на основе данных. С развитием технологий появляется всё больше инструментов, предоставляющих разнообразные функциональные возможности и графические приемы.

В нашей работе мы подробно рассмотрели две ведущие библиотеки для визуализации данных на языке программирования Python: Matplotlib и Plotly. Обе библиотеки предлагают широкий спектр инструментов для создания качественных графиков и диаграмм, но имеют свои недостатки и преимущества.

Matplotlib, как старейшая и широко используемая библиотека, выделяется отличной поддержкой и обширным функционалом. Она предоставляет обширные возможности для настройки графиков и диаграмм.
Но при этом Matplotlib может быть сложным в использовании из-за своей гибкости и разнообразия функций. Хоть с помощью него и можно реализовать практически любой тип графика или диаграммы, иногда может потребоваться значительное количество работы, чтобы получить качественный и эстетичный результат. Также одним из основных недостатков Matplotlib является его ограничение в интерактивности. Поскольку Matplotlib создает статичные изображения, они не предоставляют такого уровня взаимодействия, какие могут обеспечить интерактивные веб-графики.

Plotly, с другой стороны, предлагает более современный и интерактивный подход к визуализации данных. Его возможности по созданию веб-графиков с высокой степенью интерактивности делают его отличным выбором для создания веб-приложений, дашбордов и интерактивных отчетов. Кроме того, Plotly обладает простым и понятным синтаксисом, что делает его удобным для быстрой визуализации данных.

В зависимости от конкретных потребностей проекта, типа данных и требуемого уровня интерактивности, можно выбрать подходящую библиотеку для визуализации данных. Matplotlib подходит для создания статических изображений и автоматизации создания графиков в скриптах, в то время как Plotly отлично подходит для создания интерактивных веб-графиков и дашбордов. Обе библиотеки являются мощными инструментами для визуализации данных и могут быть эффективно использованы в различных областях анализа данных и визуализации.

Список использованных источников

1. PB Python. Overview of Python Visualization Tools [Электронный ресурс]. - URL: https://pbpython.com/visualization-tools-1.html (дата обращения: 15.03.2024)
2. PB Python. Introduction to Data Visualization with Altair [Электронный ресурс]. - URL: https://pbpython.com/altair-intro.html (дата обращения: 20.03.2024)
3. Ritza, Jacob. Matplotlib vs. seaborn vs. Plotly vs. MATLAB vs. ggplot2 vs. pandas [Электронный ресурс]. - URL: https://ritza.co/articles/matplotlib-vs-seaborn-vs-plotly-vs-MATLAB-vs-ggplot2-vs-pandas/ (дата обращения: 25.03.2024)
4. Pandas Documentation [Электронный ресурс]. - URL: https://pandas.pydata.org/docs/ (дата обращения: 30.03.2024)
5. Seaborn Documentation [Электронный ресурс]. - URL: https://seaborn.pydata.org/tutorial.html (дата обращения: 05.04.2024)
6. Bokeh Documentation [Электронный ресурс]. - URL: https://docs.bokeh.org/en/latest/docs/user_guide.html (дата обращения: 10.04.2024)
7. Pygal Documentation [Электронный ресурс]. - URL: https://www.pygal.org/en/stable/documentation/index.html (дата обращения: 15.04.2024)
8. Matplotlib Documentation [Электронный ресурс]. - URL: https://matplotlib.org/stable/users/index (дата обращения: 20.04.2024)
9. Altair Documentation [Электронный ресурс]. - URL: https://altair-viz.github.io/user_guide/data.html (дата обращения: 21.04.2024)
10. Plotly Documentation [Электронный ресурс]. - URL: https://plotly.com/python/ (дата обращения: 22.04.2024)

11. Manim Documentation [Электронный ресурс]. - URL: https://docs.manim.community/en/stable/ (дата обращения: 23.04.2024)

12. Matplotlib - Википедия [Электронный ресурс]. - URL: https://ru.wikipedia.org/wiki/Matplotlib (дата обращения: 24.04.2024)

13. Top 5 Python Libraries for Data Visualization [Электронный ресурс]. - URL: https://www.youtube.com/watch?v=jNiQaErXg8s (дата обращения: 25.04.2024)

14. 7 Python Data Visualization Libraries in 15 minutes [Электронный ресурс]. - URL: https://www.youtube.com/watch?v=4O_o53ag3ag (дата обращения: 26.04.2024)

15. Data Visualization Libraries For Python [Электронный ресурс]. - URL: https://www.youtube.com/watch?v=bUSXh45g4Zw (дата обращения: 27.04.2024)

16. How I make science animations [Электронный ресурс]. - URL: https://www.youtube.com/watch?v=yaa13eehgzo (дата обращения: 28.04.2024)

17. Python: Построение графиков по данным из файла [Электронный ресурс]. - URL: https://habr.com/ru/articles/748282/ (дата обращения: 29.04.2024)

18. ТОП-5 диаграмм для визуализации данных бизнес-аналитики [Электронный ресурс]. - URL: https://vc.ru/services/1048220-top-5-diagramm-dlya-vizualizacii-dannyh-biznes-analitiki (дата обращения: 30.04.2024)

19. 10 основных диаграмм для анализа данных [Электронный ресурс]. - URL: https://vc.ru/u/1389654-machine-learning/745030-10-osnovnyh-diagramm-dlya-analiza-dannyh (дата обращения: 30.04.2024)

20. 44 типа графиков, идеально подходящих для любой ведущей отрасли [Электронный ресурс]. - URL: https://visme.co/blog/ru/типы-графиков/ (дата обращения: 30.04.2024)

21. Виды диаграмм: 5 популярных способов визуализации данных [Электронный ресурс]. - URL: https://www.unisender.com/ru/blog/vidy-diagramm-sposoby-vizualizacii-dannyh/ (дата обращения: 30.04.2024)

22. Диаграммы — что это такое [Электронный ресурс]. - URL: https://alexkolokolov.com/ru/blog/diagrammy-chto-eto-takoe (дата обращения: 30.04.2024)

23. 10 лучших инструментов визуализации данных на 2022 год [Электронный ресурс]. - URL: https://visme.co/blog/ru/vizualizaciya-dannyh/ (дата обращения: 30.04.2024)

24. История языка программирования Python [Электронный ресурс]. - URL: https://ru.wikipedia.org/wiki/История_языка_программирования_Python (дата обращения: 30.04.2024)

25. Преимущества языка Python [Электронный ресурс]. - URL: https://www.hocktraining.com/blog/preimuschestva-yazyka-python (дата обращения: 30.04.2024)

26. Всё о языке программирования Python: растущая популярность, плюсы и минусы, сферы применения [Электронный ресурс]. - URL: https://practicum.yandex.ru/blog/vsyo-o-yazyke-programmirovaniya-python/#istoriya-sozdaniya-yazyka (дата обращения: 30.04.2024)

27. Основы языка программирования Python [Электронный ресурс]. - URL: https://www.nic.ru/help/osnovy-yazyka-programmirovaniya-python_11662.html (дата обращения: 30.04.2024)

28. Что такое Python? [Электронный ресурс]. - URL: https://aws.amazon.com/ru/what-is/python/ (дата обращения: 30.04.2024)

29. Язык Python, преимущества и недостатки [Электронный ресурс]. - URL: https://www.sostav.ru/blogs/269037/39709 (дата обращения: 30.04.2024)