Библиотеки для анализа данных

Гейне М.А.

23.10.2024

Одни из сильных сторон Python - большое сообщество и разнообразие библиотек. И именно эти особенности делают Python столь применимым для анализа данных.

Стек библиотек для анализа данных:

- NumPy;
- Pandas;
- Matplotlib;
- SciPy;
- Sklearn.

NumPy

- NumPy - это фундаментальная библиотека Python для численных вычислений.
- Она обеспечивает поддержку больших многомерных массивов и матриц, а также набор математических функций для работы с ними.
- Она является основой для многих других библиотек для работы с данными, таких как Pandas и SciPy.

Основные особенности

- **ndarray (N-dimensional array):** Эффективно обрабатывает большие массивы данных, используя однородные массивы. По сравнению со списками Python ndarray быстрее и занимает меньше памяти.
- **Бродкастинг:** Поддерживает поэлементные операции между массивами разной формы, позволяя эффективно вычислять без циклов.
- **Векторизация:** Устраняет необходимость в явных циклах, ускоряя математические операции над массивами.
- **Математические функции:** Предлагает широкий спектр математических операций (таких как тригонометрические, статистические и алгебраические функции) над массивами.

Почему NumPy?

- Массивы NumPy гораздо эффективнее списков Python как по скорости, так и по потреблению памяти.
- Обеспечивает основу для более продвинутых научных вычислений (например, анализ данных, машинное обучение).
- Крайне важен для матричных операций и работы с многомерными данными в машинном обучении и научном моделировании.

Причины производительности

1. Однородность типов данных и массивы C

- Массивы NumPy хранятся в смежных блоках памяти и требуют, чтобы все элементы были одного типа данных (например, все float или все int). Списки Python могут содержать объекты разных типов, что приводит к большим затратам памяти и проверке типов.
- Такая однородная структура позволяет NumPy напрямую взаимодействовать с низкоуровневым кодом на C и Fortran, который очень оптимизирован для численных вычислений.

2. Размещение в памяти и эффективный доступ

- **Смежная память:** Массивы NumPy размещаются в памяти смежно. Это гарантирует, что при обращении к элементам массива процессор сможет эффективнее кэшировать данные и быстро их получать.
- **Отказ от расходов на указатели:** Списки Python представляют собой коллекции указателей на объекты, и доступ к элементам предполагает разыменование этих указателей. NumPy, с другой стороны, хранит элементы плотно упакованными, избегая расходов на указатели.

3. Векторизация

- NumPy выполняет операции над целыми массивами сразу, без необходимости перебирать отдельные элементы в коде Python. Это называется векторизацией, когда операции применяются непосредственно к массивам, в том числе с помощью скомпилированного кода на языке C.
- Например, сложение двух массивов по элементам выполняется одной инструкцией на уровне C, а не итерацией по каждому элементу в цикле Python.

4. Оптимизированный бэкэнд на языках C и Fortran:

- NumPy - это обертка вокруг таких высокооптимизированных библиотек, как BLAS (Basic Linear Algebra Subprograms) и LAPACK (Linear Algebra Package). Эти библиотеки реализованы на низкоуровневых языках, таких как C и Fortran, которые намного быстрее интерпретируемого кода Python.
- Многие тяжелые вычисления (например, умножение матриц, линейная алгебра) перекладываются на эти оптимизированные библиотеки.

5. Эффективное использование кэша процессора:

- Поскольку массивы NumPy являются смежными в памяти, при обращении процессора к одному элементу, скорее всего, соседние элементы также будут загружены в кэш. Это увеличивает скорость доступа к данным при последующих операциях за счёт **локальности ссылок**.

6. Низкоуровневые SIMD-инструкции:

- NumPy может использовать преимущества **SIMD (Single Instruction, Multiple Data)** инструкций CPU, когда одна операция может быть применена к нескольким блокам данных одновременно. Это может значительно ускорить векторные операции над массивами.

```python
import numpy as np

# 1. Create a NumPy array (1D and 2D)
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# 2. Vectorized operations
result_add = array1 + 10
result_mul = array1 * 2

# 3. Broadcasting
broadcast_result = array2 + array1[:3]

# 4. Indexing and slicing
element = array2[1, 2]
row_slice = array2[1, :]

# 5. Statistical operations (mean, sum, etc.)
mean_val = np.mean(array2)
sum_along_axis = np.sum(array2, axis=0)

# 6. Aggregating conditions (boolean indexing)
greater_than_five = array2[array2 > 5]
```

```
Original 1D array: [1 2 3 4 5]
Original 2D array:
 [[1 2 3]
  [4 5 6]
  [7 8 9]]
Result of adding 10: [11 12 13 14 15]
Result of multiplying by 2: [ 2  4  6  8 10]
Broadcasting result:
 [[ 2  4  6]
  [ 8 10 12]
  [12 14 16]]
Accessed element: 6
Second row slice: [4 5 6]
Mean of array2: 5.0
Sum along columns: [12 15 18]
Elements greater than 5: [6 7 8 9]
```

Pandas

Pandas - это универсальная и мощная библиотека анализа данных на языке Python, широко используемая для работы со структурированными данными. Pandas построена поверх NumPy и в значительной степени опирается на ее базовые структуры данных, что делает эти две библиотеки тесно взаимосвязанными.

Структуры данных

- **Series**: Одномерный маркированный массив, способный хранить данные любого типа (целые числа, с плавающей запятой, строки и т. д.). По сути, это столбец в таблице.
- **DataFrame**: Двумерная структура данных с метками, похожая на электронную таблицу или таблицу SQL. Она состоит из нескольких серий, выровненных в формате таблицы.
- Pandas использует **массивы NumPy** в качестве основной структуры данных для своих Series и DataFrames. Каждый столбец в Pandas DataFrame - это, по сути, **маркированный массив NumPy** (Series), что позволяет эффективно хранить данные и манипулировать ими.
- **Однородные типы данных**: Как и массивы NumPy, каждый столбец Pandas (серия) содержит один тип данных, но в отличие от NumPy, фреймы Pandas DataFrame могут содержать несколько типов данных (например, целое число, плавающая запятая, строка) в разных столбцах.

Методы обработки данных

- **Индексация и интервалы**: Pandas предоставляет такие мощные методы, как `.loc[]`, `.iloc[]` для выбора строк и столбцов по меткам или позициям.
- **Обработка отсутствующих данных**: Pandas имеет встроенную функциональность для обработки значений NaN (Not a Number) с помощью таких методов, как `.dropna()` и `.fillna()`.

- Pandas имеет широкий набор встроенных методов для статистики (`mean()` , `sum()` , `std()` , `min()` , `max()` и т. д.), а также для агрегирования, фильтрации, сортировки и изменения формы данных.
- Pandas может работать с широким спектром форматов файлов, таких как CSV, Excel, базы данных SQL, JSON и другие. Его функции, такие как `read_csv()` , `to_csv()` , `read_excel()` и т.д., упрощают загрузку и сохранение данных.
- Pandas хорошо сочетается с другими библиотеками для работы с данными, такими как NumPy, Matplotlib и Scikit-learn. Например, он легко конвертирует массивы NumPy в Pandas DataFrames, а его структуры данных часто используются в качестве входных данных для моделей машинного обучения.

Почему Pandas?

- **Удобство**: Абстрагируется от большей части сложностей, связанных с анализом данных, что упрощает выполнение таких распространенных задач, как очистка, преобразование и визуализация данных.
- **Производительность**: Хотя Pandas не так быстр, как NumPy, для численных вычислений, он оптимизирован для работы с данными и эффективно работает с большими наборами данных.
- **Выразительный синтаксис**: Pandas предлагает лаконичный и понятный синтаксис для сложных операций, что делает его удобным выбором как для аналитиков, так и для разработчиков.

```python
import pandas as pd

# Create a DataFrame with labeled index and columns
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 32, 23, 45, 29],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix'],
    'Salary': [70000, 80000, 50000, 120000, 95000]
}

df = pd.DataFrame(data)
df.index = ['A001', 'A002', 'A003', 'A004', 'A005']  # Setting custom labeled index

# Display the DataFrame
print("DataFrame with labeled index:\n", df)
```

```
DataFrame with labeled index:
        Name  Age         City  Salary
A001   Alice   25     New York   70000
A002     Bob   32  Los Angeles   80000
A003 Charlie   23      Chicago   50000
A004   David   45      Houston  120000
A005     Eva   29      Phoenix   95000
```

```python
# 1. Querying data with labeled indexing
# Select a row using the label (e.g., 'A002')
row = df.loc['A002']  # Get row for Bob
print("\nData for 'A002' (Bob):\n", row)

# 2. Selecting specific columns for a row
name_and_salary = df.loc['A004', ['Name', 'Salary']]  # Get 'Name' and 'Salary' of 'A004' (David)
print("\nName and Salary for 'A004' (David):\n", name_and_salary)
```

```
Data for 'A002' (Bob):
Name              Bob
Age                32
City      Los Angeles
Salary          80000
Name: A002, dtype: object

Name and Salary for 'A004' (David):
Name         David
Salary      120000
Name: A004, dtype: object
```

```python
# 3. Filtering data using condition-based queries
# Query people with Salary greater than 80,000
high_earners = df[df['Salary'] > 80000]
print("\nPeople with Salary greater than 80,000:\n", high_earners)

# 4. Filtering data based on multiple conditions
# Query people who live in 'New York' or have a Salary greater than 90,000
filtered_data = df[(df['City'] == 'New York') | (df['Salary'] > 90000)]
print("\nPeople living in 'New York' or with Salary greater than 90,000:\n", filtered_data)
```

```
People with Salary greater than 80,000:
        Name  Age     City  Salary
A004   David   45  Houston  120000
A005     Eva   29  Phoenix   95000

People living in 'New York' or with Salary greater than 90,000:
       Name  Age      City  Salary
A001  Alice   25  New York   70000
A004  David   45   Houston  120000
A005    Eva   29   Phoenix   95000
```

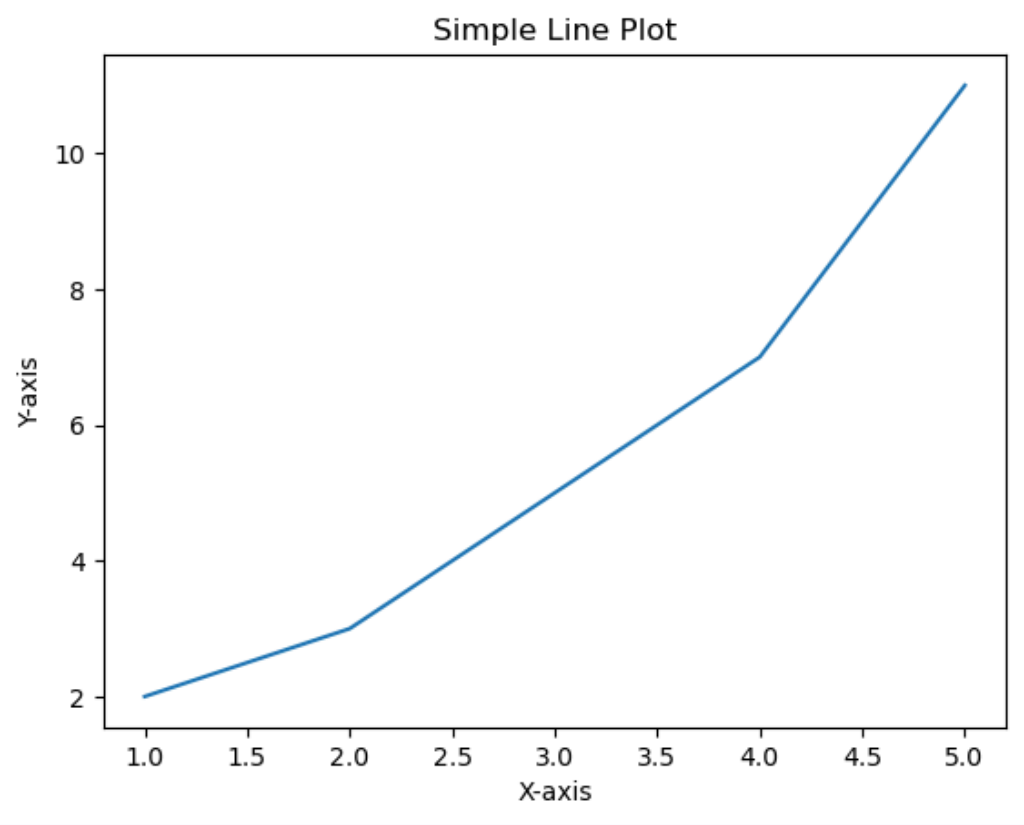
Matplotlib

Matplotlib - это широко используемая библиотека визуализации данных на Python, которая служит основой для многих других библиотек визуализации. Matplotlib предназначен для создания статических, анимированных и интерактивных визуализаций на Python. Он широко используется для построения графиков, диаграмм и рисунков, что делает его бесценным для анализа и представления данных.

Базовые компоненты

- **Рисунок (Figure)**: Общее окно или страница, содержащая один или несколько графиков. Можно считать, что это холст.
- **Оси (Axes)**: Отдельный график в пределах фигуры. Каждая ось имеет свои оси x и y, метки и может содержать несколько линий или серий данных.
- **Художник (Artist)**: Любой объект, являющийся частью фигуры, например линии, текст, маркеры и т. д.

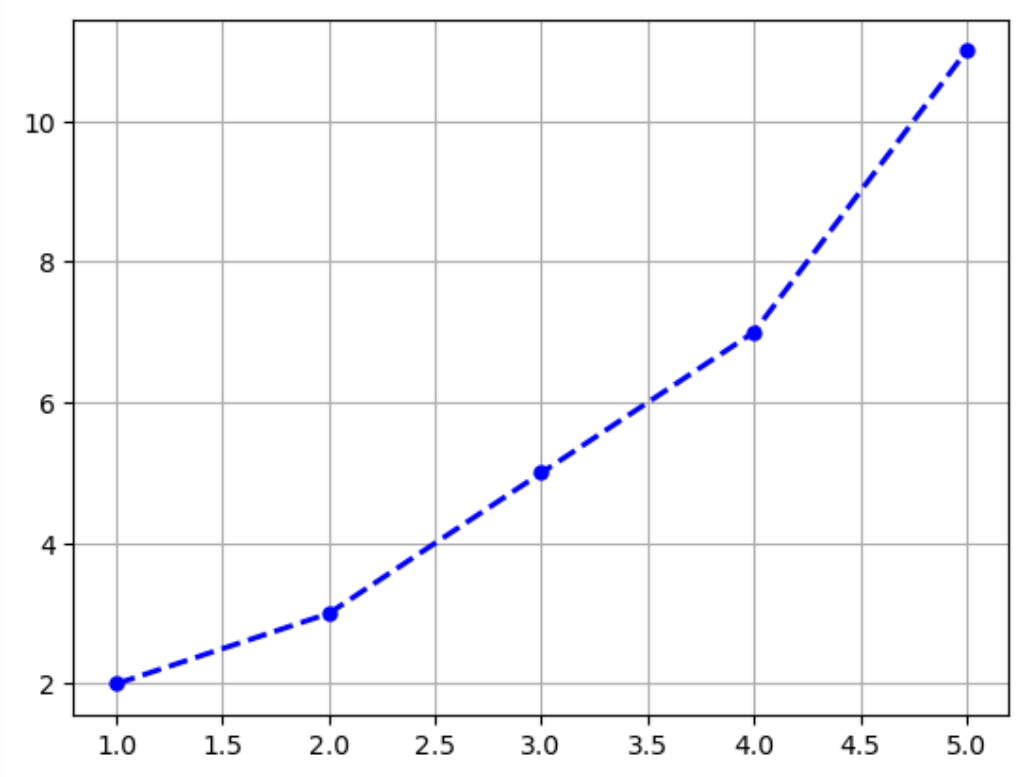
```python
import matplotlib.pyplot as plt

# Example: Simple Line Plot
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.plot(x, y)
plt.title('Simple Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



Кастомизация

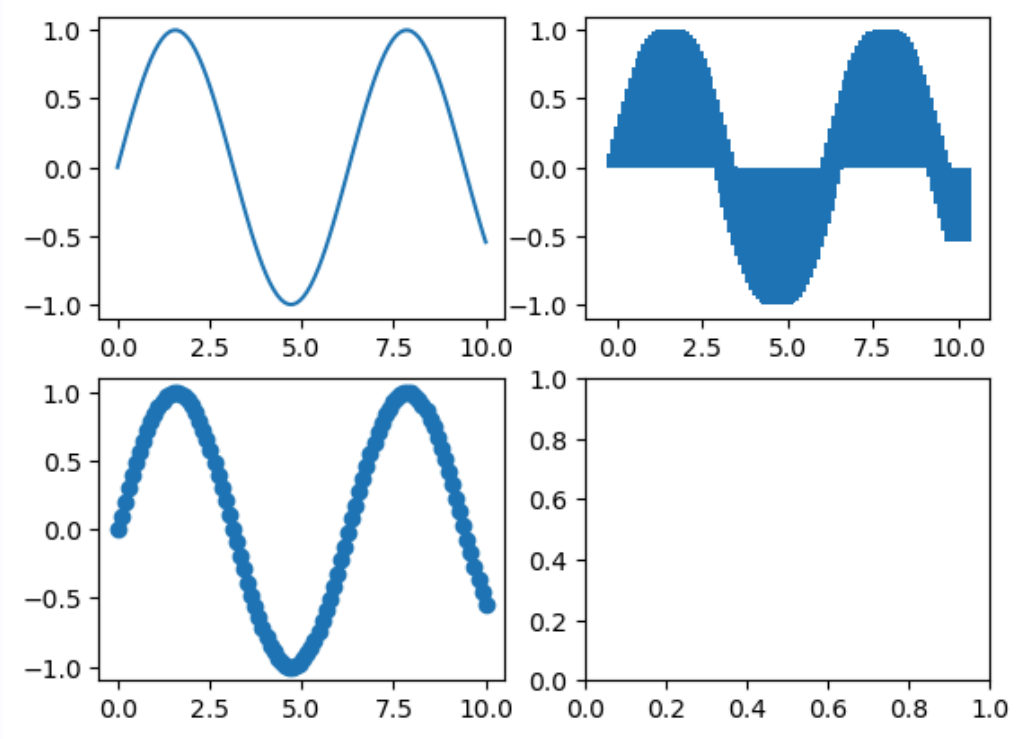
Matplotlib позволяет широко настраивать графики. Вы можете настраивать цвета, маркеры, стили линий, добавлять аннотации, легенды и линии сетки, чтобы улучшить наглядность и представление визуализации.

```
plt.plot(x, y, color='blue', marker='o',
         linestyle='--', linewidth=2, markersize=5)
plt.grid()
```



Subplots

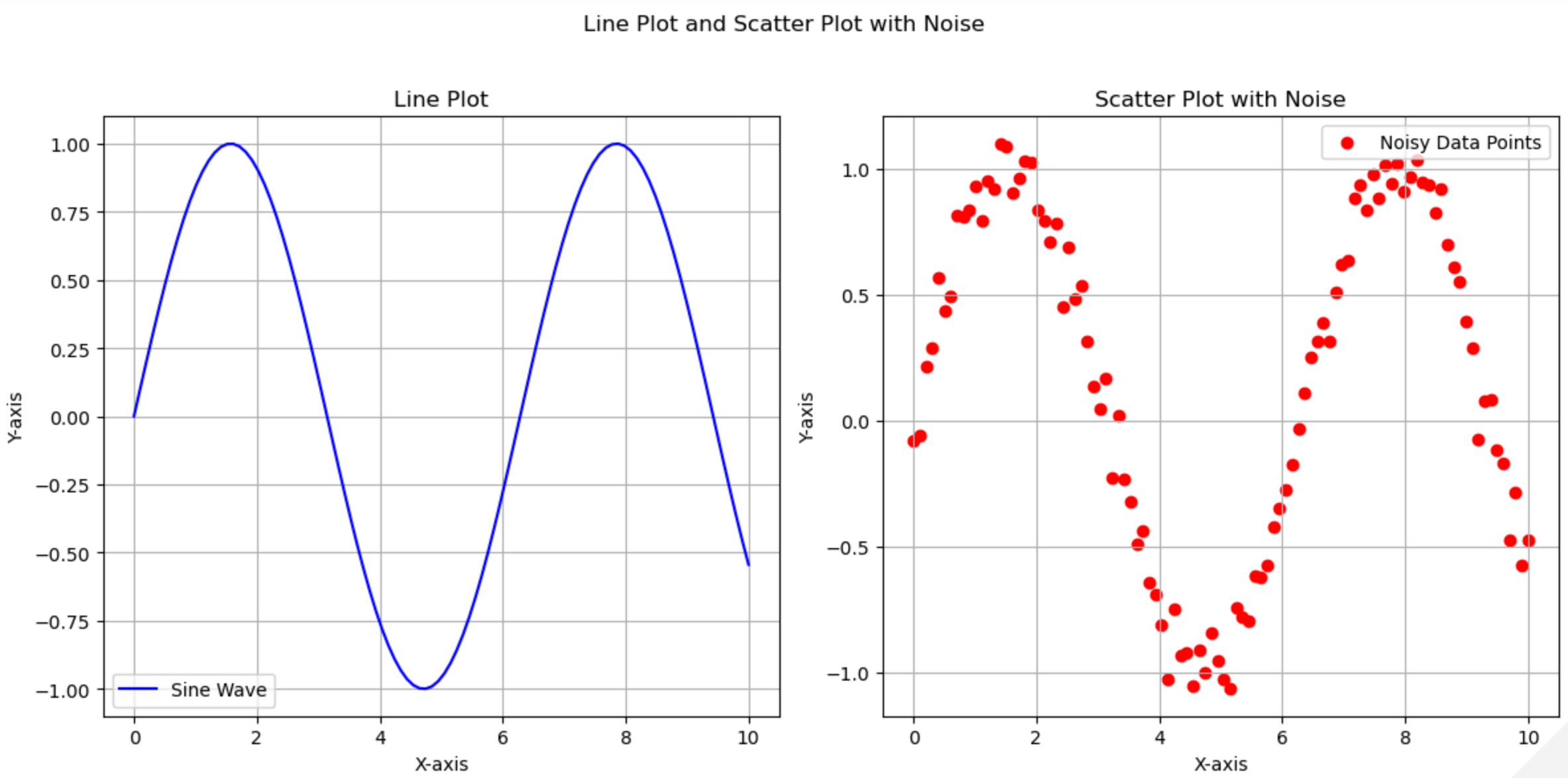
С помощью функции `subplots()` можно создать несколько графиков в рамках одного рисунка, что удобно для сравнения различных наборов данных или демонстрации различий между несколькими графиками.

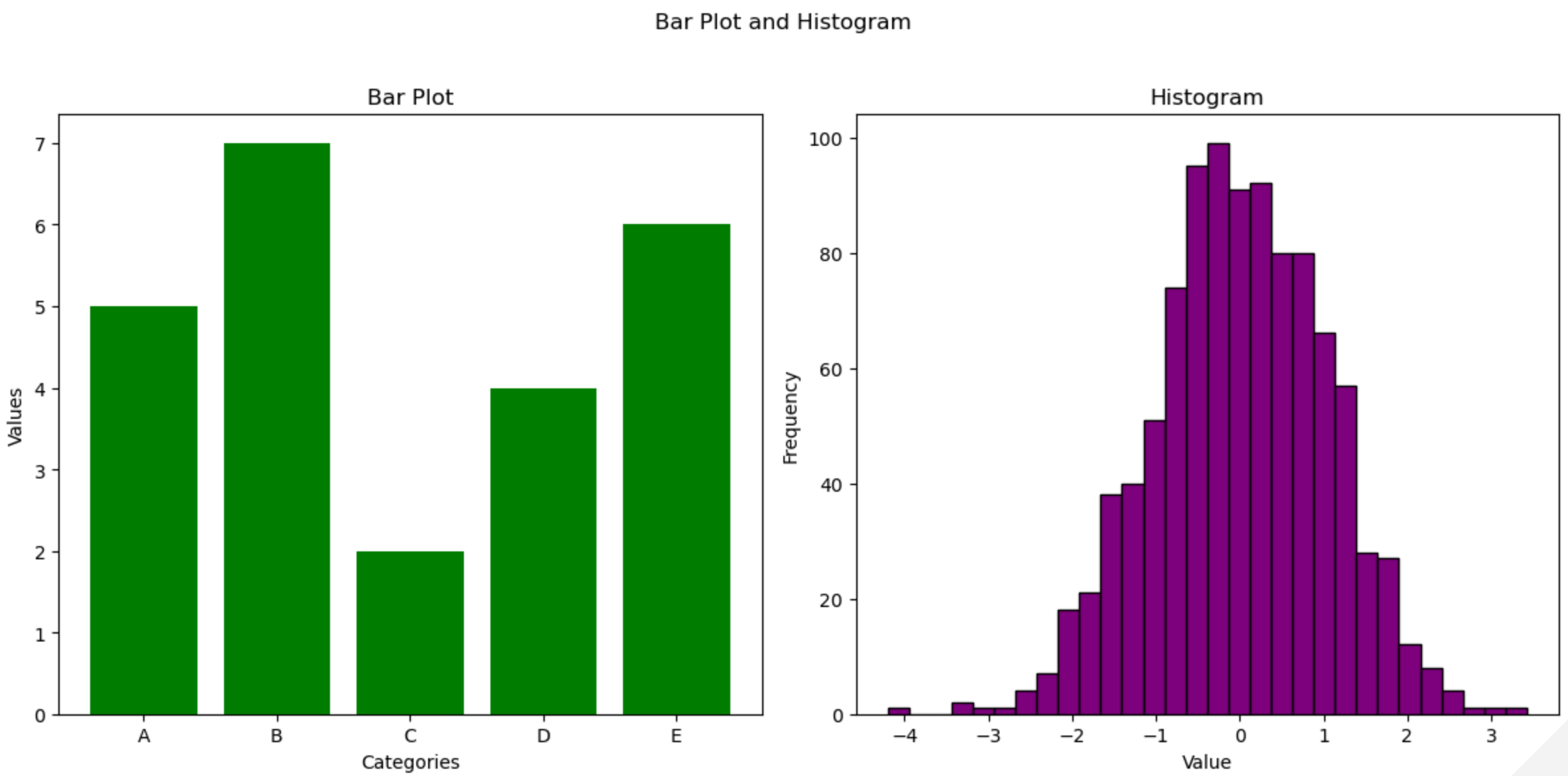


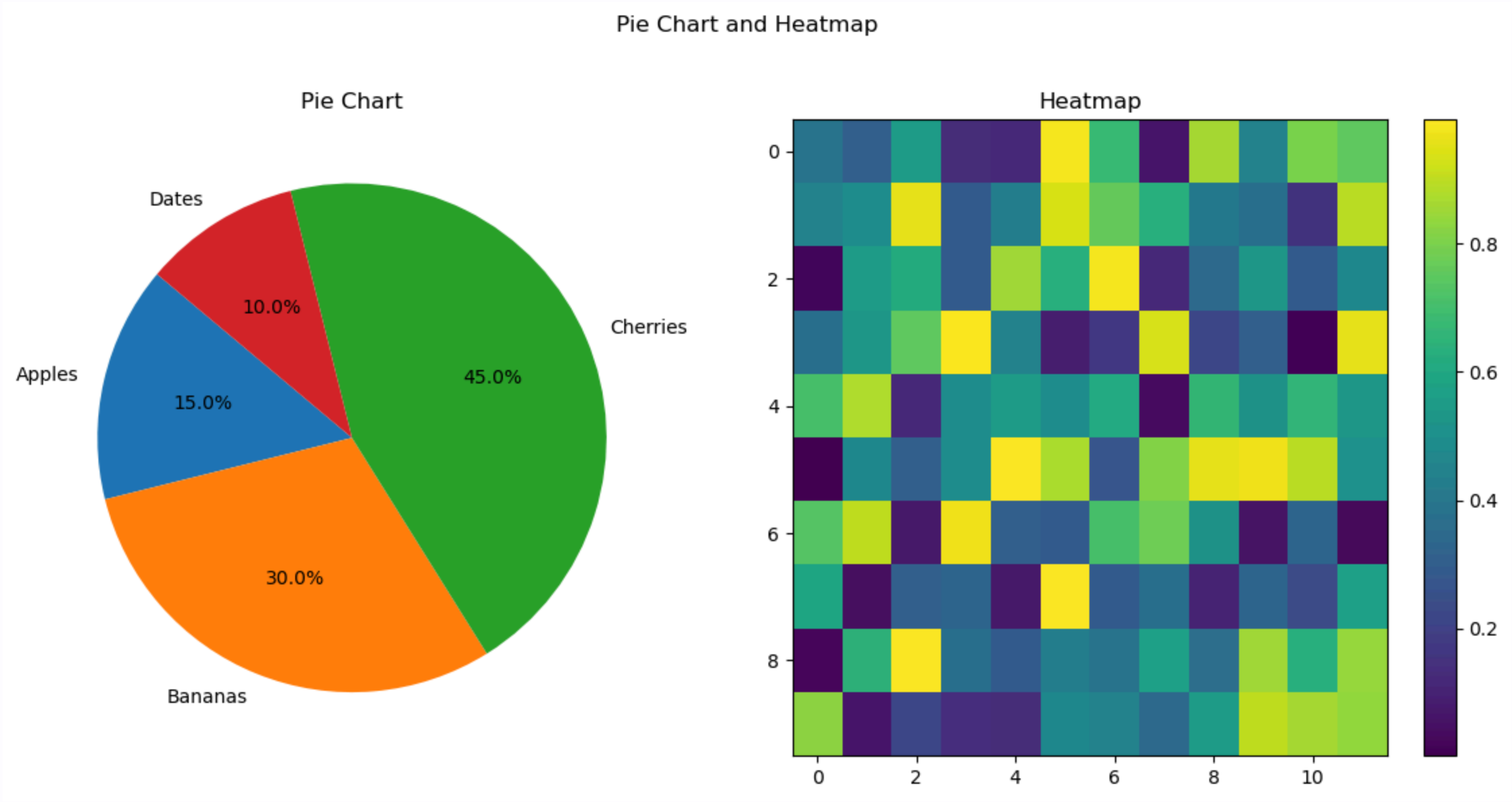

```
fig, axs = plt.subplots(2, 2)  # 2 rows, 2 columns
axs[0, 0].plot(x, y)  # Top-left plot
axs[0, 1].bar(x, y)   # Top-right plot
axs[1, 0].scatter(x, y)  # Bottom-left plot
plt.show()
```

Типы графиков

- Matplotlib поддерживает широкий спектр графиков, включая:
  - Линейные графики (line plots)
  - Графики рассеяния (scatter plots)
  - Столбчатые диаграммы (bar plots)
  - Гистограммы (histograms)
  - Круговые диаграммы (pie charts)
  - Коробчатые диаграммы (box plots)
  - Тепловые карты (heatmaps)
- Каждый тип графика можно настроить с помощью заголовков, меток, легенд и различных стилей.

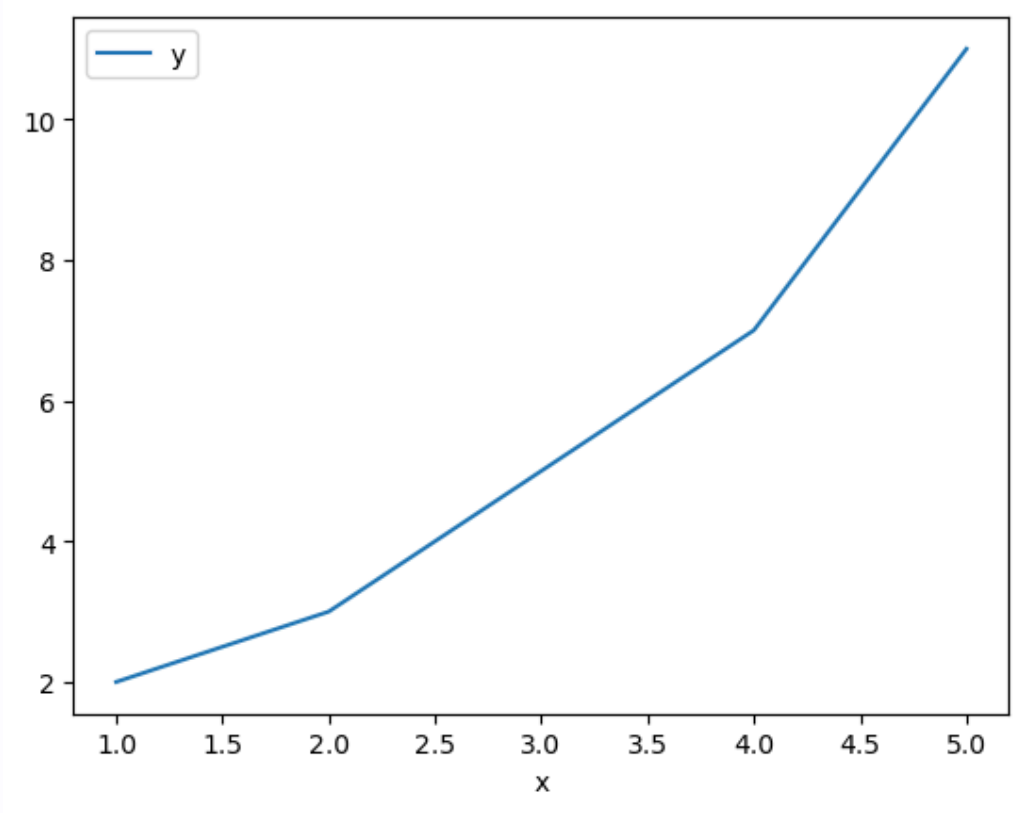

Line Plot and Scatter Plot with Noise
Line Plot
Y-axis
X-axis
Sine Wave
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
Scatter Plot with Noise
Noisy Data Points
1.0
0.5
0.0
−0.5
−1.0


Bar Plot and Histogram
Bar Plot
Values
Categories
A
B
C
D
E
0
1
2
3
4
5
6
7
Histogram
Frequency
Value
0
20
40
60
80
100
−4
−3
−2
−1
0
1
2
3


Pie Chart and Heatmap
Pie Chart
Dates
10.0%
Cherries
Apples
45.0%
15.0%
30.0%
Bananas
Heatmap
0
2
4
6
8
0
2
4
6
8
10
0.8
0.6
0.4
0.2

Интеграция с другими библиотеками

```
import pandas as pd

df = pd.DataFrame({'x': [1, 2, 3, 4, 5],
        'y': [2, 3, 5, 7, 11]})
df.plot(x='x', y='y', kind='line')
plt.show()
```



Seaborn

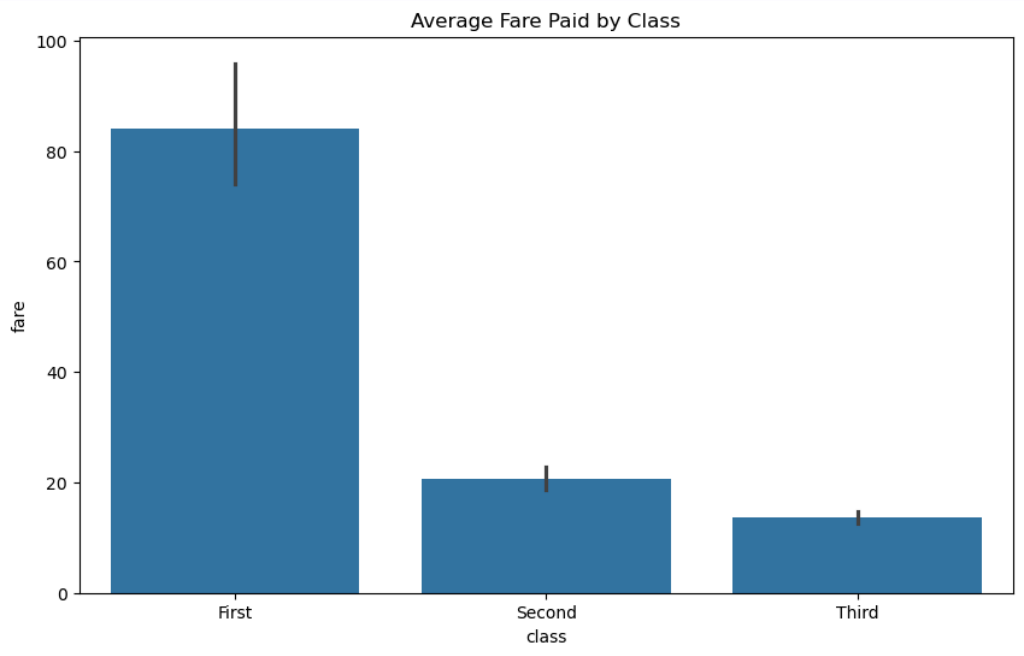
Seaborn - это мощная библиотека визуализации данных, построенная на базе Matplotlib, которая упрощает процесс создания информативных и привлекательных статистических графиков. Она предоставляет высокоуровневый интерфейс для построения привлекательных и информативных статистических графиков и особенно хорошо подходит для визуализации сложных наборов данных. Seaborn поставляется с несколькими встроенными темами и цветовыми палитрами для повышения эстетической привлекательности графиков и предлагает функции для визуализации одномерных и двумерных распределений, категориальных данных и моделей линейной регрессии.

- **Статистические графики**: Легко создавайте такие графики, как коробчатые диаграммы, скрипочные диаграммы, тепловые карты и парные графики, которые помогают понять распределение и взаимосвязи в данных.
- **Встроенные темы**: Seaborn предоставляет множество цветовых палитр и стилей, которые можно легко применить к вашим графикам.
- **Интеграция с Pandas**: Seaborn легко работает с Pandas DataFrames, позволяя легко строить графики для сложных наборов данных.

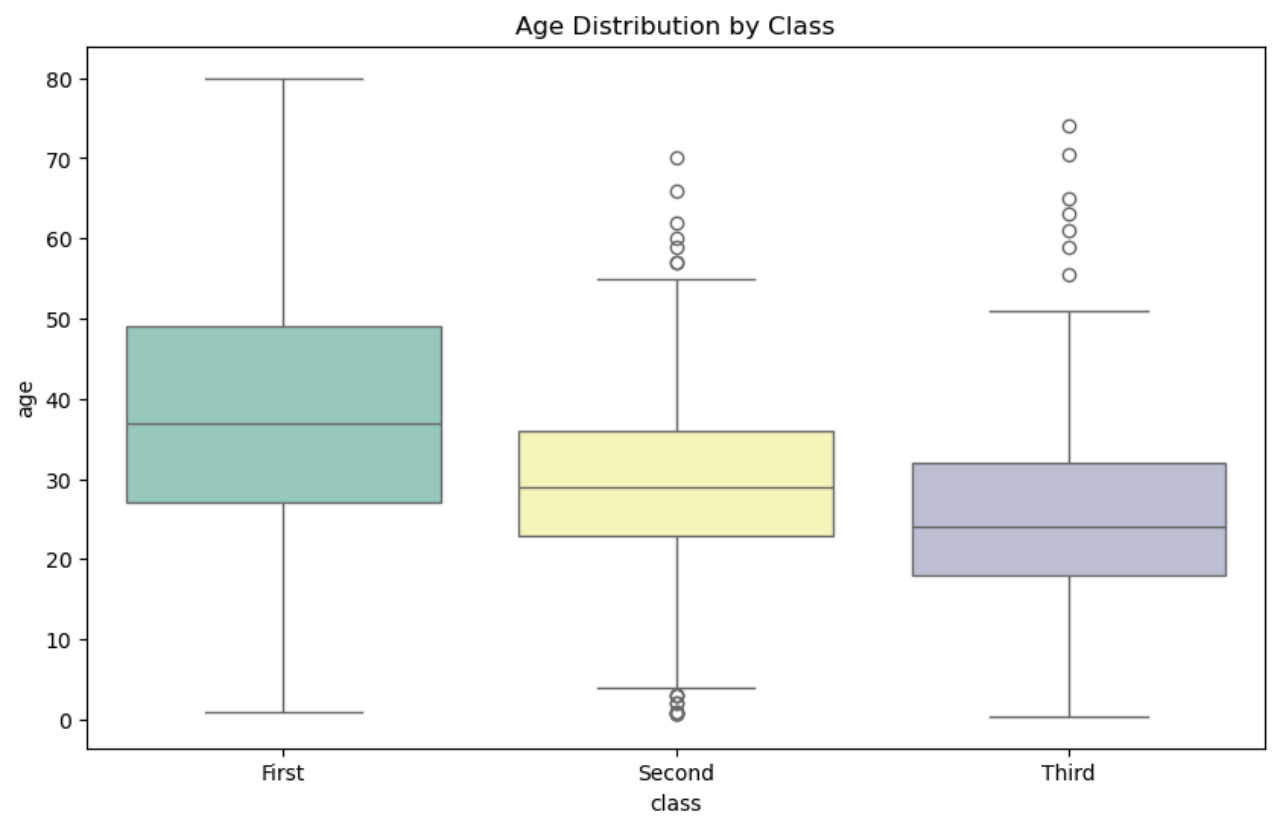
```python
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Sample dataset: Titanic dataset available in Seaborn
titanic = sns.load_dataset('titanic')

# Display the first few rows of the DataFrame to understand its structure
print(titanic.head())
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

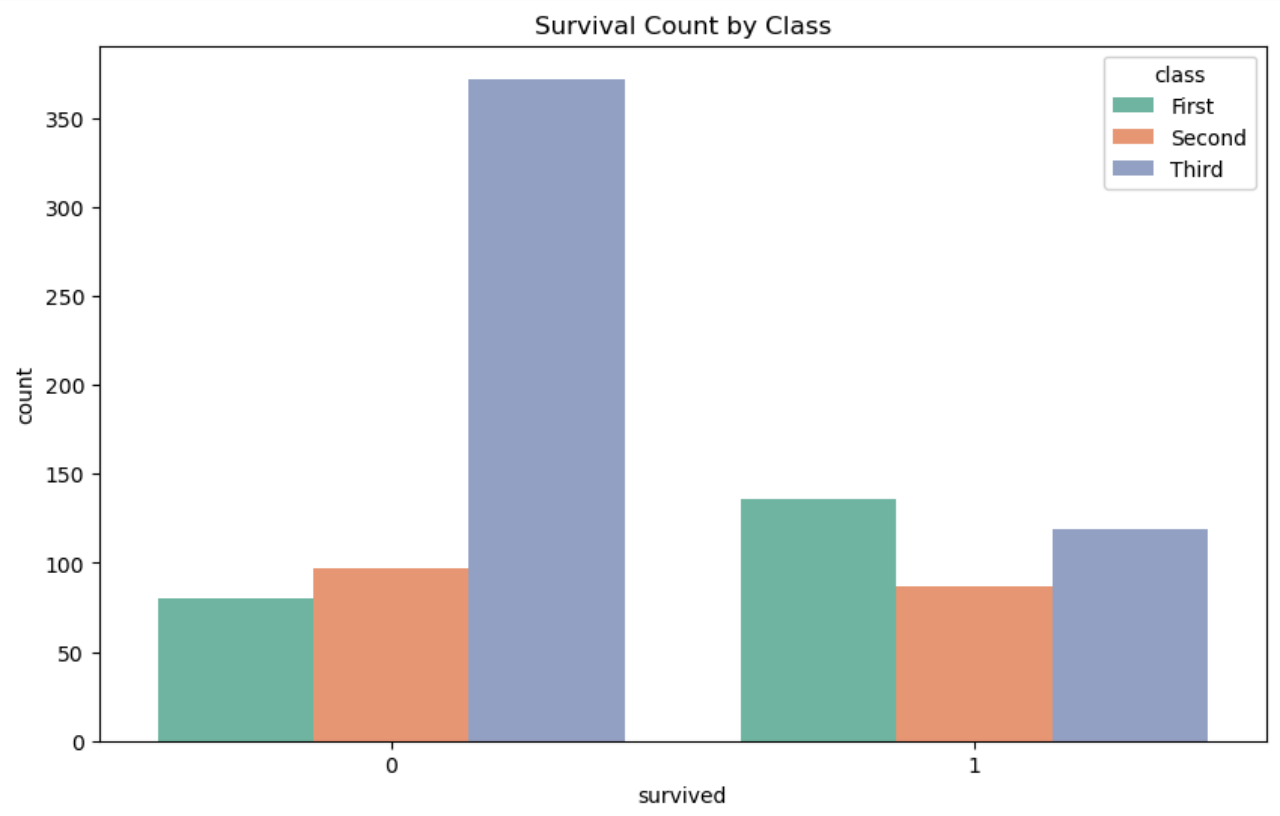
```
plt.figure(figsize=(10, 6))
sns.barplot(x='class', y='fare',
        data=titanic, estimator=lambda x: x.mean())
plt.title('Average Fare Paid by Class')
plt.show()
```



```
plt.figure(figsize=(10, 6))
sns.boxplot(x='class', y='age',
        data=titanic, palette='Set3')
plt.title('Age Distribution by Class')
plt.show()
```



```
plt.figure(figsize=(10, 6))
sns.countplot(x='survived', hue='class',
        data=titanic, palette='Set2')
plt.title('Survival Count by Class')
plt.show()
```



SciPy

SciPy - это библиотека Python, используемая для научных и технических вычислений. Она построена на базе NumPy и расширяет ее возможности, предоставляя дополнительные функции для оптимизации, интегрирования, интерполяции, решения задач с собственными значениями и других задач, связанных с научными вычислениями. SciPy широко используется в таких областях, как наука о данных, инженерия и математика, благодаря богатому набору численных методов.

Основные функции

- **Оптимизация**: Функции для минимизации или максимизации функций (например, `scipy.optimize`).
- **Интеграция**: Инструменты численного интегрирования (например, `scipy.integrate`).
- **Линейная алгебра**: Расширенные операции линейной алгебры (например, `scipy.linalg`).
- **Обработка сигналов**: Инструменты для фильтрации и обработки сигналов (например, `scipy.signal`).
- **Статистические функции**: Распределения вероятностей и статистические тесты (например, `scipy.stats`).

Пример

$$f(x) = x^2 + 5sin(x)$$

```python
import numpy as np
from scipy.optimize import minimize

# Define the function to minimize
def objective_function(x):
    return x**2 + 5*np.sin(x)

# Initial guess for the value of x
initial_guess = 2.0

# Use the minimize function to find the minimum of the objective function
result = minimize(objective_function, initial_guess)

# Display the result
print(f"Optimal value of x: {result.x[0]:.4f}")
print(f"Minimum value of the function: {result.fun:.4f}")
```

```
Optimal value of x: -1.1105
Minimum value of the function: -3.2464
```

Sklearn

Scikit-learn (более известная как Sklearn) - это мощная и широко используемая библиотека Python для машинного обучения. Она предоставляет простые и эффективные инструменты для добычи данных, анализа данных и задач машинного обучения. Sklearn построен на базе NumPy, SciPy и Matplotlib, что делает его ключевой частью экосистемы Python для науки о данных.

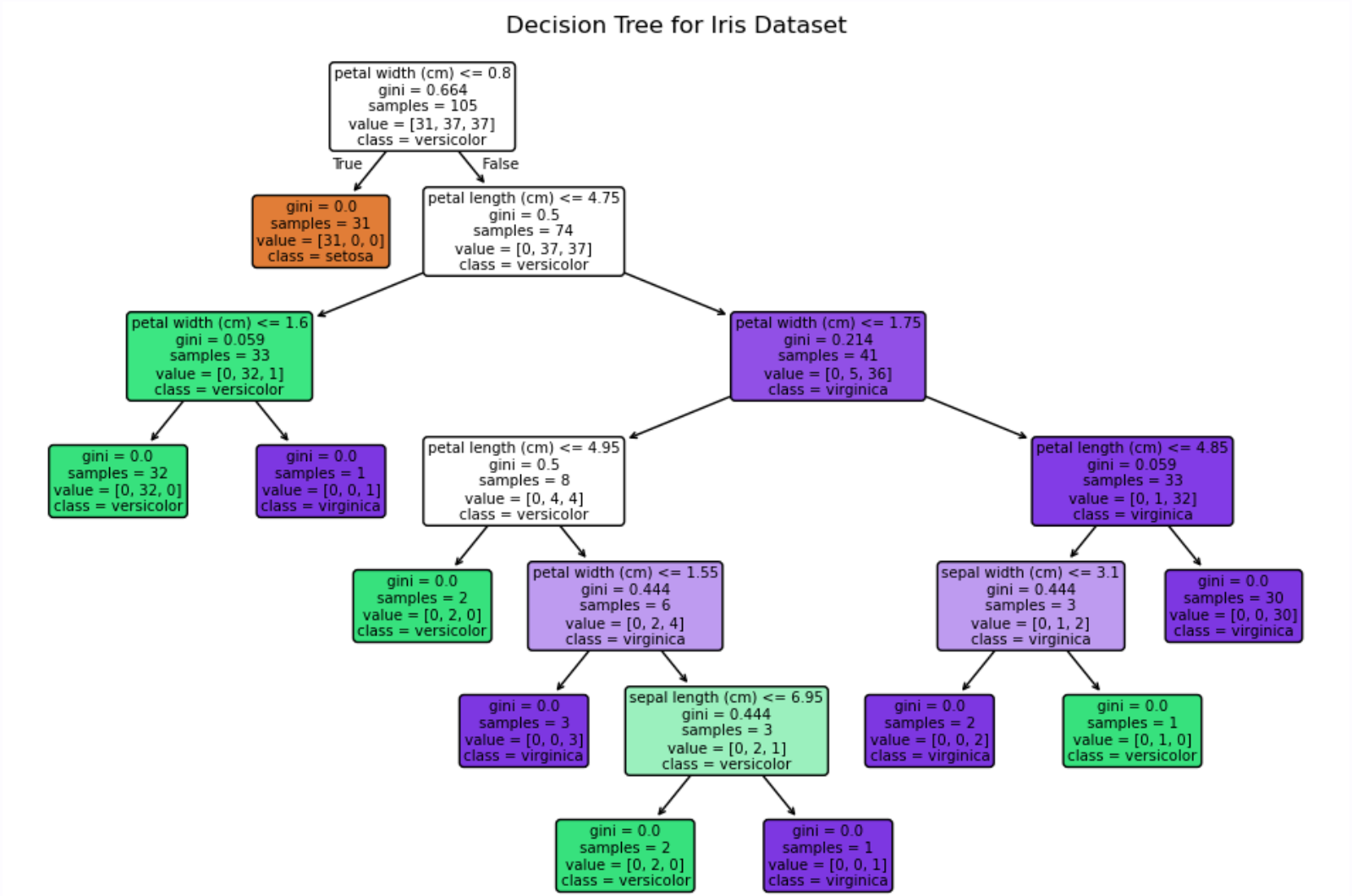
Основные возможности

- **Обучение с учителем**: Реализует многие популярные алгоритмы машинного обучения для классификации (например, Decision Trees, SVM) и регрессии (например, Linear Regression).
- **Обучение без учителя**: Предлагает алгоритмы кластеризации, такие как K-means, иерархическая кластеризация и PCA (анализ главных компонент) для уменьшения размерности.
- **Предобработка данных**: Инструменты для масштабирования, нормализации данных и кодирования категориальных переменных.
- **Выбор модели**: Функции, такие как кросс-валидация и grid search, для оценки и точной настройки моделей.
- **Конвейеры**: Упрощение рабочих процессов за счет объединения в цепочку этапов предварительной обработки и моделей.

```python
import sklearn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Decision Tree classifier
clf = DecisionTreeClassifier()

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model accuracy: {accuracy:.2f}")
```

```
Model accuracy: 1.00
```

```
from sklearn.tree import plot_tree

plt.figure(figsize=(12, 8))  # Set the figure size
plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.target_names, rounded=True)
plt.title("Decision Tree for Iris Dataset")
plt.show()
```



Спасибо за внимание!