НАУЧНЫЙ ЖУРНАЛ
НАУКА И МИРОВОЗЗРЕНИЕ

УДК-

ВВЕДЕНИЕ В МОДУЛЬ NUMPY. ОСНОВЫ РАБОТЫ С PANDAS. РАЗВЕДЫВАТЕЛЬНЫЙ АНАЛИЗ ДАННЫХ

Гараджаева Сульгун Атаевна
Старший преподаватель, Туркменский государственный университет имени Махтумкули
г. Ашхабад Туркменистан

Овездурдыева Ирина Курбангельдиевна
Старший преподаватель, Туркменский государственный университет имени Махтумкули
г. Ашхабад Туркменистан

Аннотация

Статья посвящена основам работы с библиотеками NumPy и Pandas в языке программирования Python. Эти инструменты являются основополагающими в анализе данных и широко применяются в сфере науки, бизнеса и машинного обучения. Также рассматриваются базовые приёмы разведывательного анализа данных (Exploratory Data Analysis, EDA), включая визуализацию и первичную обработку информации.

Ключевые слова: Python, NumPy, Pandas, анализ данных, массивы, таблицы, EDA, визуализация, библиотека.

1. Введение

Современная работа с данными невозможна без использования специализированных инструментов и библиотек, облегчающих обработку, анализ и визуализацию больших объёмов информации. Python как один из самых популярных языков программирования в науке о данных предлагает обширный набор средств, среди которых особенно важны библиотеки NumPy и Pandas.

NumPy предоставляет функционал для работы с многомерными массивами и выполнения высокопроизводительных числовых операций. Pandas, в свою очередь, специализируется на удобной работе с табличными структурами данных и их анализе. Совместное использование этих библиотек позволяет эффективно проводить разведывательный анализ данных, выявлять зависимости, аномалии и закономерности.

2. Основы библиотеки NumPy

NumPy (Numerical Python) — это фундаментальная библиотека для научных вычислений в Python, предоставляющая широкий набор возможностей для работы с многомерными массивами и матричными операциями. Она используется для выполнения вычислений с массивами, преобразования данных, а также для статистических операций, оптимизации алгоритмов и векторизации операций.

Основные возможности NumPy включают:

- **Эффективную работу с многомерными массивами** (объект ndarray):

  NumPy предоставляет объект ndarray, который является контейнером для однородных данных. Это означает, что массив может содержать элементы одного типа, например, целые числа или числа с плавающей запятой. В отличие от стандартных списков Python, массивы NumPy позволяют быстро и эффективно выполнять операции над большими объёмами данных.

- **Математические функции**:

  NumPy включает в себя множество встроенных математических функций для работы с массивами. Это функции для выполнения арифметических операций, статистических вычислений, линейной алгебры (умножение матриц, определители, собственные значения), а также для работы с тригонометрическими, логарифмическими и экспоненциальными функциями.

- **Инструменты генерации случайных чисел**:

  NumPy имеет развитую систему для работы с случайными величинами, включая генерацию случайных чисел, выборки и статистические распределения, такие как нормальное распределение, распределение Пуассона, равномерное распределение и другие.

- **Векторизация операций**:

  Векторизация позволяет ускорить выполнение операций над массивами без использования циклов. Это достигается благодаря использованию низкоуровневых оптимизированных операций, реализованных в библиотеке, что значительно повышает производительность. Векторизация используется для применения операций ко всем элементам массива одновременно, что в значительной степени ускоряет вычисления по сравнению с циклическим применением операций в обычных списках Python.

Пример:

```
import numpy as np

# Создание одномерных массивов
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Операции с массивами
print(a + b)  # [5 7 9]
print(a * b)  # [ 4 10 18]

# Математическая функция (синус)
print(np.sin(a))  # [0.84147098 0.90929743 0.14112001]

# Генерация случайных чисел
random_numbers = np.random.rand(3)  # Массив из 3 случайных чисел от 0 до 1
print(random_numbers)

# Операции с многомерными массивами
matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])

print(np.dot(matrix_a, matrix_b))  # Умножение матриц
```

Важные аспекты работы с ndarray:

1. **Размерность и форма массивов**:
   Массивы NumPy могут быть одномерными (вектор), двумерными (матрица) и многомерными. Размерность массива можно получить через атрибут .ndim, а его форму через атрибут .shape.
2. arr = np.array([[1, 2, 3], [4, 5, 6]])
3. print(arr.ndim) # 2 (двумерный массив)
4. print(arr.shape) # (2, 3) (2 строки, 3 столбца)
5. **Типы данных**:
   NumPy позволяет задавать типы данных для элементов массива с помощью параметра dtype. Это даёт возможность контролировать точность данных и оптимизировать использование памяти.
6. int_array = np.array([1, 2, 3], dtype=np.int32)
7. float_array = np.array([1.5, 2.5, 3.5], dtype=np.float64)
8. **Изменение формы массива**:
   С помощью метода .reshape() можно изменять форму массива, не изменяя самих данных. Это полезно при работе с многомерными массивами и для подготовки данных к анализу.
9. arr = np.array([1, 2, 3, 4, 5, 6])
10. reshaped_arr = arr.reshape(2, 3) # Преобразует в массив 2x3

11. print(reshaped_arr)
12. **Математические операции**:
    Операции, такие как умножение, деление, возведение в степень, выполняются поэлементно, что позволяет легко манипулировать данными.
13. arr = np.array([1, 2, 3, 4])
14. print(arr * 2) # [2, 4, 6, 8]
15. print(arr ** 2) # [1, 4, 9, 16]
16. **Использование логических операций**:
    NumPy поддерживает выполнение логических операций, таких как сравнение элементов массива, что полезно для фильтрации данных.
17. arr = np.array([1, 2, 3, 4])
18. print(arr > 2) # [False False True True]

3. Основы библиотеки Pandas

Pandas — это мощная библиотека Python, предназначенная для удобного хранения и анализа структурированных данных, таких как таблицы и временные ряды. Она предоставляет два основных типа данных:

- **Series** — одномерный массив, подобный списку Python, но с дополнительным индексом. Он может быть использован для представления одной переменной или одного столбца данных.
- **DataFrame** — двумерная таблица данных, состоящая из строк и столбцов, где каждый столбец может быть разного типа данных (например, числовой, строковый и т. д.). DataFrame аналогичен таблице в базе данных или электронных таблицах.

Pandas предоставляет множество удобных инструментов для работы с данными:

- **Загрузка данных** из различных форматов, таких как CSV, Excel, SQL, и других.
- **Фильтрация и группировка** данных с использованием условий и агрегатных функций.
- **Изменение структуры данных**, включая сортировку, переименование, объединение таблиц и их разбиение.
- **Обработка пропущенных значений** с помощью удобных методов для заполнения или удаления данных.

Пример загрузки данных:

```
import pandas as pd

# Загрузка данных из CSV-файла
df = pd.read_csv('data.csv')

# Просмотр первых строк таблицы
print(df.head())  # Покажет первые 5 строк DataFrame
```

Основные возможности Pandas

1. **Загрузка данных** Pandas поддерживает загрузку данных из различных источников. Для чтения данных используется метод read_, который позволяет загружать данные в DataFrame или Series. Примеры загрузки:
2. df_csv = pd.read_csv('data.csv') # Загрузка из CSV
3. df_excel = pd.read_excel('data.xlsx') # Загрузка из Excel
4. df_sql = pd.read_sql('SELECT * FROM table', connection) # Загрузка из SQL
5. **Основные операции с DataFrame** После загрузки данных в DataFrame можно выполнять различные операции, такие как фильтрация строк, выбор столбцов и другие манипуляции:
    - **Просмотр первых и последних строк**:
    - print(df.head()) # Первая пятёрка строк
    - print(df.tail()) # Последняя пятёрка строк
    - **Фильтрация данных**:
    - filtered_df = df[df['column_name'] > 10] # Выбор строк, где значения в столбце больше 10
    - **Выбор определённых столбцов**:
    - selected_columns = df[['column1', 'column2']] # Выбор нескольких столбцов
6. **Манипуляции с данными** Pandas позволяет изменять структуру данных через различные методы:
    - **Сортировка данных**:
    - sorted_df = df.sort_values(by='column_name', ascending=False) # Сортировка по убыванию
    - **Переименование столбцов**:
    - df.rename(columns={'old_name': 'new_name'}, inplace=True) # Переименование столбца
    - **Объединение и соединение таблиц**:
    - merged_df = pd.merge(df1, df2, on='common_column') # Объединение по общему столбцу
7. **Группировка и агрегация данных** Pandas поддерживает мощные инструменты для группировки данных и применения агрегатных функций. Это удобно для статистического анализа и создания отчетов:
8. grouped_df = df.groupby('category_column').agg({'numeric_column': 'sum'}) # Суммирование по категориям
9. **Обработка пропущенных значений** Работа с пропущенными данными — важный аспект анализа данных. Pandas предоставляет функции для заполнения пропусков или их удаления:
    - **Удаление пропущенных значений**:
    - df.dropna(inplace=True) # Удаляет строки с пропущенными значениями
    - **Заполнение пропусков**:
    - df.fillna(value=0, inplace=True) # Заполнение пропусков нулями

10. **Применение функций к данным** В Pandas можно легко применять функции к данным в таблице, используя метод apply для обработки каждого элемента, строки или столбца:
11. df['new_column'] = df['column'].apply(lambda x: x * 2) # Применение функции ко всем элементам столбца
12. **Временные ряды** Pandas предоставляет удобные инструменты для работы с временными рядами, включая методы для преобразования дат и выполнения временных операций:
13. df['date_column'] = pd.to_datetime(df['date_column']) # Преобразование столбца в тип datetime
14. df.set_index('date_column', inplace=True) # Установка временного индекса

Пример: анализ данных с использованием Pandas

Предположим, у нас есть набор данных о продажах товаров в различных категориях. Мы можем выполнить базовый анализ:

```
import pandas as pd

# Загрузка данных
df = pd.read_csv('sales_data.csv')

# Просмотр первых строк данных
print(df.head())

# Группировка данных по категориям и подсчёт суммы продаж
category_sales = df.groupby('category')['sales'].sum()

# Вывод результатов
print(category_sales)
```

4. Разведывательный анализ данных (EDA)

Exploratory Data Analysis (EDA) — это процесс предварительного исследования данных с целью получения представления о структуре, закономерностях и потенциальных проблемах в наборе данных.

Основные этапы EDA включают:

- **Обзор структуры данных**: размерность, типы переменных, наличие пропусков;
- **Описательная статистика**: средние, медианы, стандартное отклонение;
- **Визуализация**: гистограммы, диаграммы рассеяния, ящики с усами (boxplot);
- **Поиск выбросов и аномалий**.

Пример визуализации:

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.histplot(df['age'], bins=20)
plt.title('Распределение возраста')
plt.show()
```

EDA помогает выбрать подходящие методы очистки данных, масштабирования, а также строить гипотезы для последующего анализа.

5. Практическое значение и перспективы

Использование NumPy и Pandas в сочетании с методами EDA позволяет аналитикам, научным сотрудникам и инженерам принимать обоснованные решения на основе данных. Эти инструменты широко применяются в:

- маркетинговой аналитике;
- экономике и финансах;
- биоинформатике;
- машинном обучении;
- цифровых гуманитарных науках.

Рост объёмов данных и развитие технологий анализа делают знания NumPy и Pandas неотъемлемой частью компетенций современного специалиста.

Заключение

В статье рассмотрены основные принципы работы с библиотеками NumPy и Pandas, а также важнейшие этапы разведывательного анализа данных. Эти инструменты позволяют упростить и ускорить процессы обработки информации, делая анализ более доступным и продуктивным. В дальнейшем они служат фундаментом для построения моделей машинного обучения и глубокого анализа.

Литература

1. VanderPlas J. *Python Data Science Handbook*. O’Reilly Media, 2016.
2. McKinney W. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O’Reilly Media, 2018.
3. Raschka S., Mirjalili V. *Python Machine Learning*. Packt Publishing, 2020.
4. Официальная документация NumPy — https://numpy.org/
5. Официальная документация Pandas — https://pandas.pydata.org/