Ответы на экзаменационные вопросы по технологиям обработки данных

1. Процедура построения "one-hot encoding" для одномерного массива numpy из целых неотрицательных чисел

One-hot encoding — это метод преобразования категориальных данных в числовой формат, где каждая категория представляется бинарным вектором.

Алгоритм построения:

1. Определяем количество уникальных значений в массиве
2. Создаем матрицу размером (n_samples, n_classes)
3. Для каждого элемента исходного массива устанавливаем 1 в соответствующую позицию

Пример реализации:

```python
import numpy as np

def one_hot_encode(arr):
    n_classes = np.max(arr) + 1
    n_samples = len(arr)

    # Создаем матрицу нулей
    one_hot = np.zeros((n_samples, n_classes))

    # Устанавливаем 1 в нужных позициях
    one_hot[np.arange(n_samples), arr] = 1

    return one_hot

# Пример использования
arr = np.array([0, 1, 2, 1, 0])
result = one_hot_encode(arr)
# Результат: [[1, 0, 0], [0, 1, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0]]
```

Готовые решения в библиотеках:

- `sklearn.preprocessing.OneHotEncoder`
- `pd.get_dummies()` в pandas

---

2. Технологический стек Python для обработки и анализа данных

Python как glue language:

Python называют "клеящим языком" благодаря способности легко интегрировать компоненты, написанные на разных языках программирования. Это достигается через:

- C-расширения (CPython API)
- Биндинги к библиотекам на C/C++/Fortran
- Простой синтаксис для вызова внешних программ

Основные компоненты стека:

1. **NumPy** - базовая библиотека для работы с многомерными массивами
2. **Pandas** - анализ и манипуляция структурированными данными
3. **Matplotlib/Seaborn** - визуализация данных
4. **SciPy** - научные вычисления
5. **Scikit-learn** - машинное обучение
6. **Jupyter** - интерактивная разработка

Специфика NumPy:

- **Производительность**: операции выполняются в скомпилированном коде C
- **Векторизация**: операции применяются ко всему массиву сразу
- **Эффективная память**: непрерывное размещение данных в памяти
- **Основа экосистемы**: большинство библиотек использует массивы NumPy

Роль в экосистеме:

NumPy предоставляет единый интерфейс для работы с числовыми данными, что позволяет различным библиотекам эффективно взаимодействовать друг с другом.

---

3. Организация массивов в NumPy

Хранение данных:

NumPy использует **ndarray** - N-мерный массив с элементами одного типа:

- Данные хранятся в непрерывном блоке памяти
- Метаданные содержат информацию о форме, типе данных и стратегии доступа
- Поддерживается row-major (C-style) и column-major (Fortran-style) порядок

Создание массивов:

```python
import numpy as np

# Из списка
arr1 = np.array([1, 2, 3, 4])

# Заполненные значениями
arr2 = np.zeros((3, 4))         # нули
arr3 = np.ones((2, 3))          # единицы
arr4 = np.full((2, 2), 7)       # заданное значение

# Диапазоны
arr5 = np.arange(0, 10, 2)      # [0, 2, 4, 6, 8]
arr6 = np.linspace(0, 1, 5)     # равномерно распределенные

# Случайные числа
arr7 = np.random.random((3, 3))
```

Принципы реализации операций:

1. **Векторизация**: операции применяются поэлементно без явных циклов
2. **Broadcasting**: автоматическое приведение массивов к совместимым размерам
3. **Ufuncs**: универсальные функции, оптимизированные для массивов
4. **View vs Copy**: операции могут возвращать представление или копию данных

---

4. Универсальные функции и применение функций по осям в NumPy

Универсальные функции (ufuncs):

Это функции, которые выполняют поэлементные операции над массивами NumPy с автоматической векторизацией.

Типы ufuncs:

1. **Унарные**: `np.sin`, `np.cos`, `np.exp`, `np.log`, `np.sqrt`
2. **Бинарные**: `np.add`, `np.multiply`, `np.power`, `np.maximum`

Методы ufuncs:

```python
arr = np.array([1, 2, 3, 4, 5])

# reduce - применяет операцию последовательно
np.add.reduce(arr)        # сумма всех элементов
np.multiply.reduce(arr)   # произведение всех элементов

# accumulate - накопительная операция
np.add.accumulate(arr)    # [1, 3, 6, 10, 15]

# outer - внешнее произведение
np.multiply.outer([1, 2], [3, 4])  # [[3, 4], [6, 8]]
```

Применение функций по осям:

```python
arr = np.array([[1, 2, 3],
                [4, 5, 6]])

# По столбцам (axis=0)
np.sum(arr, axis=0)      # [5, 7, 9]

# По строкам (axis=1)
np.sum(arr, axis=1)      # [6, 15]

# По всем осям
np.sum(arr)              # 21
```

---

5. Принцип распространения значений (Broadcasting) в NumPy

Общий алгоритм Broadcasting:

1. Выравнивание размерностей справа
2. Проверка совместимости размеров (размер должен быть равен 1 или совпадать)
3. Растягивание массивов до общего размера

Правила Broadcasting:

- Размеры сравниваются справа налево
- Размеры совместимы, если они равны или один из них равен 1
- Отсутствующие размеры считаются равными 1

Примеры:

```python
# Пример 1: скаляр и массив
a = np.array([1, 2, 3])      # shape: (3,)
b = 10                       # shape: ()
result = a * b               # [10, 20, 30]

# Пример 2: массивы разных размеров
a = np.array([[1, 2, 3],     # shape: (2, 3)
              [4, 5, 6]])
b = np.array([10, 20, 30])   # shape: (3,)
result = a + b               # [[11, 22, 33], [14, 25, 36]]

# Пример 3: добавление размерности
a = np.array([[1], [2]])     # shape: (2, 1)
b = np.array([10, 20, 30])   # shape: (3,)
result = a + b               # shape: (2, 3)
```

Маскирование и индексирование:

Булево маскирование:

```python
arr = np.array([1, 2, 3, 4, 5])
mask = arr > 3
filtered = arr[mask]         # [4, 5]
```

Fancy indexing:

```python
arr = np.array([10, 20, 30, 40, 50])
indices = [1, 3, 4]
selected = arr[indices]      # [20, 40, 50]

# Многомерное индексирование
arr2d = np.array([[1, 2], [3, 4], [5, 6]])
rows = [0, 2]
cols = [1, 0]
result = arr2d[rows, cols]  # [2, 5]
```

---

6. Организация Pandas DataFrame и индексация

Структура DataFrame:

DataFrame - это двумерная структура данных с:

- **Индексом строк** (row index)
- **Индексом столбцов** (column index)
- **Данными** различных типов в столбцах

Создание DataFrame:

```python
import pandas as pd

# Из словаря
df = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
})

# С пользовательским индексом
df = pd.DataFrame(data, index=['row1', 'row2', 'row3'])
```

Типы индексации:

1. Label-based (loc):

```python
# Выбор по меткам
df.loc['row1', 'A']          # конкретная ячейка
df.loc['row1':'row2', 'A':'B'] # диапазон
df.loc[df['A'] > 1, 'B']     # условная индексация
```

2. Position-based (iloc):

```python
# Выбор по позициям
df.iloc[0, 1]                # первая строка, второй столбец
df.iloc[0:2, 1:3]           # диапазон по позициям
df.iloc[[0, 2], [1, 2]]     # конкретные позиции
```

3. Series индексация:

python

```python
series = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
series['a']                  # по метке
series[0]                    # по позиции
series['a':'b']              # срез по меткам
```

Универсальные функции в Pandas:

python

```python
# Применение функций
df.apply(np.sqrt)            # к каждому элементу
df.apply(lambda x: x.max() - x.min())  # к каждому столбцу
df.apply(lambda x: x.max() - x.min(), axis=1)  # к каждой строке
```

Работа с пустыми значениями:

python

```python
# Обнаружение NaN
df.isna()                    # булев DataFrame
df.notna()                   # обратная операция

# Удаление NaN
df.dropna()                  # удалить строки с NaN
df.dropna(axis=1)            # удалить столбцы с NaN
df.dropna(how='all')         # только если все NaN

# Заполнение NaN
df.fillna(0)                 # заполнить нулями
df.fillna(method='forward') # прямое заполнение
df.fillna(df.mean())         # средними значениями
```

---

7. Объединение данных из нескольких Pandas DataFrame

Основные методы объединения:

1. Concatenation (pd.concat):

```python
# Вертикальное объединение
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
result = pd.concat([df1, df2])

# Горизонтальное объединение
result = pd.concat([df1, df2], axis=1)

# С контролем индексов
result = pd.concat([df1, df2], ignore_index=True)
```

2. Merge (SQL-подобные операции):

```python
df1 = pd.DataFrame({'key': ['A', 'B', 'C'], 'value1': [1, 2, 3]})
df2 = pd.DataFrame({'key': ['A', 'B', 'D'], 'value2': [4, 5, 6]})

# Inner join (пересечение)
result = pd.merge(df1, df2, on='key', how='inner')

# Left join
result = pd.merge(df1, df2, on='key', how='left')

# Outer join (объединение)
result = pd.merge(df1, df2, on='key', how='outer')

# По нескольким ключам
result = pd.merge(df1, df2, on=['key1', 'key2'])
```

3. Join (объединение по индексу):

```python
df1 = pd.DataFrame({'A': [1, 2]}, index=['x', 'y'])
df2 = pd.DataFrame({'B': [3, 4]}, index=['x', 'y'])
result = df1.join(df2)
```

Общая логика:

- **Concat**: простое склеивание по осям
- **Merge**: реляционное объединение по ключам
- **Join**: объединение по индексам

- Важно учитывать совпадающие имена столбцов и типы данных

---

8. Принципы работы с файлами и операционные системы

Файлы в операционных системах:

Файл - это именованная последовательность байтов, хранящаяся на устройстве хранения данных.

Основные понятия:

- **Путь к файлу**: абсолютный и относительный
- **Права доступа**: чтение, запись, выполнение
- **Файловая система**: организация хранения файлов
- **Дескриптор файла**: идентификатор открытого файла

Текстовые vs Бинарные файлы:

Текстовые файлы:

- Содержат читаемый текст
- Используют кодировку (UTF-8, ASCII, etc.)
- Имеют концепцию строк (разделители `\n`, `\r\n`)
- Могут обрабатываться текстовыми редакторами

Бинарные файлы:

- Содержат данные в двоичном формате
- Не имеют концепции строк
- Более компактные
- Требуют специальных программ для чтения

Работа с файлами в Python:

```python
# Текстовый режим
with open('file.txt', 'r', encoding='utf-8') as f:
    content = f.read()

# Бинарный режим
with open('file.bin', 'rb') as f:
    data = f.read()

# Запись
with open('output.txt', 'w', encoding='utf-8') as f:
    f.write('Hello, World!')
```

Режимы открытия файлов:

- `'r'` - чтение (текст)
- `'w'` - запись (текст)
- `'a'` - добавление (текст)
- `'rb'`, `'wb'`, `'ab'` - бинарные режимы
- `'r+'`, `'w+'` - чтение и запись

---

9. Сериализация и десериализация данных. Формат JSON

Сериализация/Десериализация:

- **Сериализация**: преобразование объектов в формат для хранения/передачи
- **Десериализация**: восстановление объектов из сериализованного формата

Цели сериализации:

- Сохранение состояния объектов
- Передача данных по сети
- Обмен данными между приложениями
- Кэширование

Формат JSON (JavaScript Object Notation):

Характеристики:

- Текстовый формат
- Легко читается человеком
- Поддерживается большинством языков программирования

- Основан на подмножестве JavaScript

Типы данных JSON:

- **Строки**: `"hello"`
- **Числа**: `42`, `3.14`
- **Булевы**: `true`, `false`
- **null**: `null`
- **Массивы**: `[1, 2, 3]`
- **Объекты**: `{"key": "value"}`

Пример JSON:

```json
{
    "name": "John Doe",
    "age": 30,
    "is_student": false,
    "courses": ["Math", "Physics"],
    "address": {
        "street": "123 Main St",
        "city": "New York"
    },
    "phone": null
}
```

Работа с JSON в Python:

```python
import json

# Данные Python
data = {
    'name': 'Alice',
    'age': 25,
    'skills': ['Python', 'SQL']
}

# Сериализация (Python -> JSON)
json_string = json.dumps(data, indent=2)
print(json_string)

# Сохранение в файл
with open('data.json', 'w') as f:
    json.dump(data, f, indent=2)

# Десериализация (JSON -> Python)
parsed_data = json.loads(json_string)

# Загрузка из файла
with open('data.json', 'r') as f:
    loaded_data = json.load(f)

# Работа с русским текстом
json.dumps(data, ensure_ascii=False, indent=2)
```

---

10. Формат XML и модель DOM. Работа с BeautifulSoup

XML (eXtensible Markup Language):

Расширяемый язык разметки для структурированного представления данных.

Характеристики XML:

- Иерархическая структура
- Самоописывающийся формат
- Поддержка атрибутов и пространств имен
- Строгий синтаксис

Пример XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book id="1" language="en">
        <title>Python Programming</title>
        <author>John Smith</author>
        <year>2023</year>
        <genres>
            <genre>Programming</genre>
            <genre>Education</genre>
        </genres>
    </book>
    <book id="2" language="ru">
        <title>Анализ данных</title>
        <author>Иван Петров</author>
        <year>2022</year>
    </book>
</library>
```

Модель DOM (Document Object Model):

DOM представляет XML-документ как древовидную структуру объектов в памяти.

Компоненты DOM:

- **Узлы** (nodes): элементы, атрибуты, текст
- **Элементы** (elements): теги XML
- **Атрибуты** (attributes): свойства элементов
- **Дерево**: иерархическая структура

Работа с BeautifulSoup:

Установка и импорт:

```python
# pip install beautifulsoup4 lxml
from bs4 import BeautifulSoup
```

Парсинг XML:

```python
xml_content = """
<library>
    <book id="1">
        <title>Python Programming</title>
        <author>John Smith</author>
    </book>
</library>
"""

# Создание объекта BeautifulSoup
soup = BeautifulSoup(xml_content, 'xml')

# Поиск элементов
title = soup.find('title')
print(title.text)  # "Python Programming"

# Поиск всех элементов
books = soup.find_all('book')
for book in books:
    print(book.get('id'))

# Навигация по дереву
library = soup.library
first_book = library.book
author = first_book.author.text

# CSS-селекторы
titles = soup.select('book title')
book_with_id = soup.select('book[id="1"]')
```

Изменение XML:

```python
# Изменение текста
soup.find('title').string = 'New Title'

# Добавление элементов
new_book = soup.new_tag('book', id='3')
new_title = soup.new_tag('title')
new_title.string = 'Data Science'
new_book.append(new_title)
soup.library.append(new_book)

# Удаление элементов
soup.find('author').decompose()
```

---

11. Формат CSV и работа с ним в Python

Формат CSV (Comma-Separated Values):

Простой текстовый формат для представления табличных данных.

Характеристики CSV:

- Строки разделены переносами строк
- Столбцы разделены запятыми (или другими разделителями)
- Первая строка часто содержит заголовки
- Простота и широкая поддержка

Пример CSV:

```csv
name,age,city,salary
John,25,New York,50000
Alice,30,London,60000
Bob,35,Paris,55000
```

Особенности формата:

- **Кавычки**: для строк с разделителями или переносами
- **Экранирование**: удвоение кавычек внутри строк
- **Кодировка**: обычно UTF-8 или ASCII
- **Разделители**: запятая, точка с запятой, табуляция

Работа с CSV в Python:

Модуль csv:

```python
import csv

# Чтение CSV
with open('data.csv', 'r', encoding='utf-8') as file:
    reader = csv.reader(file)
    headers = next(reader)  # первая строка
    for row in reader:
        print(row)

# Чтение с DictReader
with open('data.csv', 'r', encoding='utf-8') as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row['name'], row['age'])

# Запись CSV
data = [
    ['name', 'age', 'city'],
    ['John', 25, 'NYC'],
    ['Alice', 30, 'LA']
]

with open('output.csv', 'w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerows(data)

# Запись с DictWriter
fieldnames = ['name', 'age', 'city']
with open('output.csv', 'w', newline='', encoding='utf-8') as file:
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerow({'name': 'John', 'age': 25, 'city': 'NYC'})
```

Pandas для CSV:

```python
import pandas as pd

# Чтение CSV
df = pd.read_csv('data.csv',
                 encoding='utf-8',
                 sep=',',
                 header=0)

# Продвинутые параметры
df = pd.read_csv('data.csv',
                 dtype={'age': int},
                 parse_dates=['date'],
                 na_values=['', 'NULL', 'N/A'])

# Запись CSV
df.to_csv('output.csv',
          index=False,
          encoding='utf-8',
          sep=',')
```

---

12. Использование Excel и библиотека XLWings

Excel для обработки данных:

Excel остается популярным инструментом для анализа данных благодаря:

- Интуитивному интерфейсу
- Мощным функциям и формулам
- Возможностям визуализации
- Широкому распространению в бизнесе

Библиотека XLWings:

Возможности XLWings:

- Чтение и запись данных Excel
- Выполнение макросов VBA
- Интеграция Python с Excel
- Автоматизация задач Excel

Принципы работы:

XLWings использует COM-интерфейс Windows или AppleScript на macOS для взаимодействия с Excel.

Установка:

```bash
pip install xlwings
```

Основные примеры:

python

```python
import xlwings as xw
import pandas as pd

# Открытие Excel приложения
app = xw.App(visible=True)

# Создание новой книги
wb = app.books.add()
ws = wb.sheets['Sheet1']

# Запись данных
ws.range('A1').value = 'Hello Excel!'
ws.range('A2:C4').value = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Чтение данных
value = ws.range('A1').value
table = ws.range('A2:C4').value

# Работа с DataFrame
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['NY', 'LA', 'Chicago']
})

# Запись DataFrame в Excel
ws.range('E1').value = df

# Чтение в DataFrame
df_from_excel = ws.range('E1').options(pd.DataFrame, header=1, index=False).value

# Форматирование
ws.range('A1').color = (255, 255, 0)  # желтый фон
ws.range('A1').font.bold = True

# Формулы
ws.range('D1').formula = '=SUM(A2:A4)'

# Диаграммы
chart = ws.charts.add()
chart.set_source_data(ws.range('A1:B4'))
chart.chart_type = 'line'

# Сохранение и закрытие
wb.save('example.xlsx')
```

```
wb.close()
app.quit()
```

Полезные функции:

```python
# Работа с существующим файлом
wb = xw.Book('existing_file.xlsx')

# Автоматическое определение диапазона
ws.range('A1').expand().value

# Макросы VBA
wb.macro('MacroName')()

# Обновление связей
wb.api.UpdateLinks = 1
```

---

13. Основы работы с Matplotlib

Организация системы координат:

Figure и Axes:

- **Figure**: холст для рисования (окно или файл)
- **Axes**: система координат внутри Figure
- **Axis**: отдельная ось (X или Y)

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

# Создание Figure и Axes
fig, ax = plt.subplots(figsize=(8, 6))

# Простой график
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
ax.plot(x, y)
```

Оформление осей:

```python
# Подписи осей
ax.set_xlabel('X axis label')
ax.set_ylabel('Y axis label')
ax.set_title('Plot Title')

# Пределы осей
ax.set_xlim(0, 10)
ax.set_ylim(-1, 1)

# Деления на осях (ticks)
ax.set_xticks([0, np.pi, 2*np.pi])
ax.set_xticklabels(['0', 'π', '2π'])

# Сетка
ax.grid(True, alpha=0.3)

# Легенда
ax.legend(['sin(x)'])
```

Цвета и цветовые карты:

Способы задания цветов:

```python
# Именованные цвета
ax.plot(x, y, color='red')
ax.plot(x, y, color='blue')

# Hex-коды
ax.plot(x, y, color='#FF5733')

# RGB кортежи
ax.plot(x, y, color=(0.2, 0.6, 0.8))

# Короткие обозначения
ax.plot(x, y, 'r-')  # красная линия
ax.plot(x, y, 'b--') # синяя пунктирная
```

Цветовые карты (colormaps):

python

```python
# Для scatter plot
colors = np.random.rand(50)
ax.scatter(x[:50], y[:50], c=colors, cmap='viridis')

# Популярные colormaps
# viridis, plasma, inferno, magma
# hot, cool, spring, winter
# RdYlBu, RdBu, coolwarm
```

Стили линий и маркеры:

Стили линий:

python

```python
ax.plot(x, y, linestyle='-')     # сплошная
ax.plot(x, y, linestyle='--')    # пунктирная
ax.plot(x, y, linestyle='-.')    # штрих-пунктир
ax.plot(x, y, linestyle=':')     # точечная

# Толщина линии
ax.plot(x, y, linewidth=2)
```

Маркеры:

python

```python
ax.plot(x, y, marker='o')        # круги
ax.plot(x, y, marker='s')        # квадраты
ax.plot(x, y, marker='^')        # треугольники
ax.plot(x, y, marker='*')        # звездочки
ax.plot(x, y, marker='x')        # крестики

# Размер маркеров
ax.plot(x, y, marker='o', markersize=8)

# Комбинированный стиль
ax.plot(x, y, 'ro-', linewidth=2, markersize=6)
```

---

14. Pyplot и объектно-ориентированный интерфейс matplotlib

Два интерфейса Matplotlib:

1. Pyplot (процедурный стиль):

python

```python
import matplotlib.pyplot as plt

# Простой способ
plt.figure(figsize=(10, 6))
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('Simple Plot')
plt.show()
```

2. Объектно-ориентированный интерфейс:

python

```python
# Более гибкий подход
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_title('OO Plot')
plt.show()
```

Управление фигурами:

Создание множественных фигур:

python

```python
# Несколько отдельных фигур
fig1 = plt.figure(1)
plt.plot([1, 2, 3], [1, 4, 9])

fig2 = plt.figure(2)
plt.plot([1, 2, 3], [1, 2, 3])

# Переключение между фигурами
plt.figure(1)
plt.title('Figure 1')
```

Создание множественных графиков:

Subplots:

python

```python
# Сетка 2x2
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Доступ к конкретному subplot
axes[0, 0].plot([1, 2, 3], [1, 4, 9])
axes[0, 1].plot([1, 2, 3], [1, 2, 3])
axes[1, 0].scatter([1, 2, 3], [3, 2, 1])
axes[1, 1].bar([1, 2, 3], [3, 7, 5])

# Общий заголовок
fig.suptitle('Multiple Subplots')

# Автоматическая компоновка
plt.tight_layout()
```

Продвинутые layouts:

python

```python
# Неравномерная сетка
fig = plt.figure(figsize=(12, 8))

# GridSpec для сложных layouts
from matplotlib.gridspec import GridSpec
gs = GridSpec(3, 3, figure=fig)

ax1 = fig.add_subplot(gs[0, :])    # верхний ряд
ax2 = fig.add_subplot(gs[1, :-1])  # средний левый
ax3 = fig.add_subplot(gs[1:, -1])  # правый столбец
ax4 = fig.add_subplot(gs[-1, 0])   # нижний левый
ax5 = fig.add_subplot(gs[-1, -2])  # нижний средний
```

Различные типы графиков:

```python
# Линейный график
ax.plot(x, y, label='Line plot')

# Точечный график
ax.scatter(x, y, label='Scatter plot')

# Столбчатая диаграмма
ax.bar(categories, values, label='Bar plot')

# Гистограмма
ax.hist(data, bins=20, alpha=0.7, label='Histogram')

# Boxplot
ax.boxplot(data, labels=['Group 1', 'Group 2'])

# Круговая диаграмма
ax.pie(sizes, labels=labels, autopct='%1.1f%%')

# Контурный график
ax.contour(X, Y, Z, levels=10)
ax.contourf(X, Y, Z, levels=10, cmap='viridis')

# Тепловая карта
im = ax.imshow(data, cmap='hot', interpolation='nearest')
plt.colorbar(im)

# 3D график (требует from mpl_toolkits.mplot3d import Axes3D)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')
```

---

15. Визуализация данных с помощью Pandas

Встроенные методы визуализации:

Pandas предоставляет удобные методы для быстрой визуализации данных, основанные на matplotlib.

Для Series:

```python
import pandas as pd
import numpy as np

# Создание Series
s = pd.Series(np.random.randn(100).cumsum(),
              index=pd.date_range('2023-01-01', periods=100))

# Различные типы графиков
s.plot()                    # линейный график
s.plot(kind='hist')         # гистограмма
s.plot(kind='box')          # box plot
s.plot(kind='density')      # плотность распределения
```

Для DataFrame:

```python
# Создание DataFrame
df = pd.DataFrame({
    'A': np.random.randn(100).cumsum(),
    'B': np.random.randn(100).cumsum(),
    'C': np.random.randn(100).cumsum()
}, index=pd.date_range('2023-01-01', periods=100))

# Линейные графики
df.plot()                   # все столбцы
df.plot(y='A')              # один столбец
df.plot(x='A', y='B', kind='scatter')  # scatter plot

# Столбчатые диаграммы
df.sum().plot(kind='bar')   # вертикальные столбцы
df.sum().plot(kind='barh')  # горизонтальные столбцы

# Гистограммы
df.plot(kind='hist', alpha=0.7, bins=20)
df.hist(figsize=(12, 8))    # множественные гистограммы

# Box plots
df.plot(kind='box')
df.boxplot(column=['A', 'B', 'C'])

# Круговые диаграммы
df.iloc[0].plot(kind='pie', autopct='%1.1f%%')
```

Продвинутые методы визуализации:

Группировка и агрегация:

```python
# Группировка с визуализацией
grouped_data = df.groupby('category').sum()
grouped_data.plot(kind='bar', stacked=True)

# Pivot tables с визуализацией
pivot_df = df.pivot_table(values='value',
                          index='date',
                          columns='category')
pivot_df.plot(kind='area', stacked=True)
```

Параметры настройки:

```python
# Настройка размера фигуры
df.plot(figsize=(12, 6))

# Настройка цветов
df.plot(color=['red', 'blue', 'green'])

# Настройка стилей
df.plot(style=['--', '-.', ':'])

# Подписи и заголовки
df.plot(title='My Plot',
        xlabel='X Label',
        ylabel='Y Label')

# Легенда
df.plot(legend=True, loc='upper right')

# Сетка
df.plot(grid=True, alpha=0.3)

# Логарифмическая шкала
df.plot(logy=True)
```

Специальные типы графиков:

```python
# Scatter matrix
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize=(12, 12), alpha=0.2)

# Параллельные координаты
from pandas.plotting import parallel_coordinates
parallel_coordinates(df, 'class_column')

# Rad viz
from pandas.plotting import radviz
radviz(df, 'class_column')

# Andrews curves
from pandas.plotting import andrews_curves
andrews_curves(df, 'class_column')
```

---

16. Анализ данных: типы признаков и статистический анализ

Типы признаков:

1. Количественные (Numerical):

- **Дискретные**: счетные значения (количество детей, число покупок)
- **Непрерывные**: измеримые значения (рост, вес, доход)

2. Качественные (Categorical):

- **Номинальные**: без порядка (цвет, пол, марка автомобиля)
- **Порядковые**: с естественным порядком (образование, размер одежды)

3. Специальные типы:

- **Бинарные**: два возможных значения (да/нет, 0/1)
- **Временные**: даты и времена
- **Текстовые**: свободный текст

Анализ распределений:

Для количественных признаков:

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

# Основные статистики
df['column'].describe()

# Гистограмма
df['column'].hist(bins=30)
plt.title('Distribution of Column')

# Плотность распределения
df['column'].plot(kind='density')

# Q-Q plot для проверки нормальности
from scipy import stats
stats.probplot(df['column'], dist="norm", plot=plt)

# Тесты на нормальность
stats.shapiro(df['column'])      # Shapiro-Wilk test
stats.kstest(df['column'], 'norm')  # Kolmogorov-Smirnov test
```

Для категориальных признаков:

```python
# Частотная таблица
df['category'].value_counts()

# Доли
df['category'].value_counts(normalize=True)

# Столбчатая диаграмма
df['category'].value_counts().plot(kind='bar')

# Круговая диаграмма
df['category'].value_counts().plot(kind='pie')
```

Меры центральной тенденции:

```python
# Среднее арифметическое
mean_val = df['column'].mean()

# Медиана (50-й перцентиль)
median_val = df['column'].median()

# Мода (наиболее частое значение)
mode_val = df['column'].mode()[0]

# Среднее геометрическое
from scipy.stats import gmean
geom_mean = gmean(df['column'])

# Среднее гармоническое
from scipy.stats import hmean
harm_mean = hmean(df['column'])

# Усеченное среднее
from scipy.stats import trim_mean
trimmed_mean = trim_mean(df['column'], 0.1)  # убираем 10% крайних значений
```

Меры разброса:

```python
# Дисперсия
variance = df['column'].var()

# Стандартное отклонение
std_dev = df['column'].std()

# Размах
range_val = df['column'].max() - df['column'].min()

# Межквартильный размах
q75, q25 = df['column'].quantile([0.75, 0.25])
iqr = q75 - q25

# Коэффициент вариации
cv = std_dev / mean_val
```

Поиск выбросов:

Метод межквартильного размаха (IQR):

python

```python
def find_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers

outliers = find_outliers_iqr(df['column'])
```

Z-score метод:

python

```python
from scipy import stats

def find_outliers_zscore(data, threshold=3):
    z_scores = np.abs(stats.zscore(data))
    outliers = data[z_scores > threshold]
    return outliers

outliers = find_outliers_zscore(df['column'])
```

Модифицированный Z-score:

python

```python
def find_outliers_modified_zscore(data, threshold=3.5):
    median = np.median(data)
    mad = np.median(np.abs(data - median))
    modified_z_scores = 0.6745 * (data - median) / mad
    outliers = data[np.abs(modified_z_scores) > threshold]
    return outliers
```

Анализ взаимного распределения:

Двумерный анализ:

python

```python
# Scatter plot
plt.scatter(df['x'], df['y'])
plt.xlabel('X variable')
plt.ylabel('Y variable')

# Двумерная гистограмма
plt.hist2d(df['x'], df['y'], bins=20)
plt.colorbar()

# Hexbin plot
plt.hexbin(df['x'], df['y'], gridsize=20)
plt.colorbar()

# Совместное распределение
from scipy.stats import gaussian_kde
xy = np.vstack([df['x'], df['y']])
density = gaussian_kde(xy)
```

Корреляционный анализ:

python

```python
# Корреляционная матрица
corr_matrix = df.corr()

# Различные методы корреляции
pearson_corr = df['x'].corr(df['y'], method='pearson')    # линейная
spearman_corr = df['x'].corr(df['y'], method='spearman')  # ранговая
kendall_corr = df['x'].corr(df['y'], method='kendall')    # тау Кендалла

# Визуализация корреляционной матрицы
import seaborn as sns
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)

# Статистическая значимость корреляции
from scipy.stats import pearsonr
correlation, p_value = pearsonr(df['x'], df['y'])
```

---

17. Разведочный анализ данных с помощью Seaborn

Возможности Seaborn:

Seaborn - это библиотека статистической визуализации, построенная на основе matplotlib, которая предоставляет высокоуровневый интерфейс для создания красивых графиков.

Основные преимущества:

- Встроенные статистические функции
- Красивые стили по умолчанию
- Простота работы с pandas DataFrame
- Специализированные графики для анализа данных

Настройка стилей:

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

# Установка стиля
sns.set_style("whitegrid")      # whitegrid, darkgrid, white, dark, ticks
sns.set_palette("husl")         # цветовая палитра
sns.set_context("notebook")     # notebook, paper, talk, poster
```

Основные типы графиков:

1. Распределения:

```python
# Гистограмма с кривой плотности
sns.histplot(data=df, x='column', kde=True)

# Плотность распределения
sns.kdeplot(data=df, x='column')

# Распределение по категориям
sns.histplot(data=df, x='value', hue='category', multiple='stack')

# Rug plot (отметки на оси)
sns.rugplot(data=df, x='column')

# Распределение для нескольких переменных
sns.displot(data=df, x='column', col='category', kind='hist')
```

2. Категориальные данные:

python

```python
# Box plot
sns.boxplot(data=df, x='category', y='value')

# Violin plot
sns.violinplot(data=df, x='category', y='value')

# Strip plot
sns.stripplot(data=df, x='category', y='value', jitter=True)

# Swarm plot
sns.swarmplot(data=df, x='category', y='value')

# Bar plot с доверительными интервалами
sns.barplot(data=df, x='category', y='value', estimator=np.mean)

# Count plot
sns.countplot(data=df, x='category')

# Point plot
sns.pointplot(data=df, x='category', y='value')
```

3. Взаимосвязи между переменными:

python

```python
# Scatter plot с регрессией
sns.scatterplot(data=df, x='x', y='y', hue='category')
sns.regplot(data=df, x='x', y='y')

# Линия регрессии для каждой категории
sns.lmplot(data=df, x='x', y='y', hue='category')

# Residual plot
sns.residplot(data=df, x='x', y='y')

# Joint plot (scatter + распределения)
sns.jointplot(data=df, x='x', y='y', kind='scatter')
sns.jointplot(data=df, x='x', y='y', kind='reg')
sns.jointplot(data=df, x='x', y='y', kind='hex')

# Pair plot (все против всех)
sns.pairplot(data=df, hue='category')
```

4. Матричные графики:

```python
# Корреляционная матрица
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)

# Кластерная карта
sns.clustermap(corr_matrix, annot=True, cmap='coolwarm')

# Heatmap для pivot table
pivot_table = df.pivot_table(values='value', index='row', columns='col')
sns.heatmap(pivot_table, annot=True)
```

Комплексные графики:

FacetGrid для множественных графиков:

```python
# Создание сетки графиков
g = sns.FacetGrid(df, col='category', row='subcategory', height=4)
g.map(plt.hist, 'value', bins=15)
g.add_legend()

# Применение различных функций
g = sns.FacetGrid(df, col='category', hue='type')
g.map(sns.scatterplot, 'x', 'y', alpha=0.7)
g.add_legend()
```

PairGrid для попарных сравнений:

```python
g = sns.PairGrid(df, hue='category')
g.map_diag(sns.histplot)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.add_legend()
```

Статистические графики:

python

```python
# Регрессионный анализ
sns.regplot(data=df, x='x', y='y', order=2)  # полиномиальная регрессия

# Логистическая регрессия
sns.regplot(data=df, x='x', y='binary_y', logistic=True)

# Доверительные интервалы
sns.lineplot(data=df, x='time', y='value', ci=95)

# Временные ряды
sns.lineplot(data=df, x='date', y='value', hue='category')
```

Продвинутые возможности:

python

```python
# Кастомизация палитр
custom_palette = sns.color_palette("husl", 8)
sns.set_palette(custom_palette)

# Аннотации на графиках
ax = sns.scatterplot(data=df, x='x', y='y')
for i, txt in enumerate(df['labels']):
    ax.annotate(txt, (df['x'][i], df['y'][i]))

# Комбинирование с matplotlib
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
sns.boxplot(data=df, x='cat', y='val', ax=axes[0,0])
sns.violinplot(data=df, x='cat', y='val', ax=axes[0,1])
sns.stripplot(data=df, x='cat', y='val', ax=axes[1,0])
sns.barplot(data=df, x='cat', y='val', ax=axes[1,1])
```

---

18. Форматирование строк в Python

Три основных способа форматирования:

1. %-форматирование (старый стиль):

```python
# Основной синтаксис
name = "Alice"
age = 25
message = "Hello, %s! You are %d years old." % (name, age)

# Форматирование чисел
price = 19.99
formatted = "Price: $%.2f" % price  # "Price: $19.99"

# Различные спецификаторы
%s    # строка
%d    # целое число
%f    # число с плавающей точкой
%x    # шестнадцатеричное число
%o    # восьмеричное число
%e    # экспоненциальная запись
%g    # общий формат числа

# Примеры
"Integer: %d" % 42
"Float: %.3f" % 3.14159
"String: %s" % "hello"
"Hex: %x" % 255          # "Hex: ff"
"Padded: %05d" % 42      # "Padded: 00042"
"Left aligned: %-10s" % "text"  # "Left aligned: text      "
```

2. Метод format():

```python
# Позиционные аргументы
message = "Hello, {}! You are {} years old.".format("Alice", 25)

# Именованные аргументы
message = "Hello, {name}! You are {age} years old.".format(name="Alice", age=25)

# Индексы
message = "Hello, {0}! You are {1} years old. {0} is a nice name.".format("Alice", 25)

# Форматирование чисел
"Price: ${:.2f}".format(19.99)
"Percentage: {:.1%}".format(0.75)      # "Percentage: 75.0%"
"Scientific: {:.2e}".format(1234.56)   # "Scientific: 1.23e+03"

# Выравнивание и заполнение
"{:>10}".format("text")       # "      text" (правое выравнивание)
"{:<10}".format("text")       # "text      " (левое выравнивание)
"{:^10}".format("text")       # "   text   " (центрирование)
"{:*^10}".format("text")      # "***text***" (заполнение символом)

# Работа с атрибутами и элементами
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person = Person("Alice", 25)
"Name: {p.name}, Age: {p.age}".format(p=person)

data = {"name": "Alice", "age": 25}
"Name: {d[name]}, Age: {d[age]}".format(d=data)
```

3. f-строки (f-strings) - Python 3.6+:

python

```python
# Основной синтаксис
name = "Alice"
age = 25
message = f"Hello, {name}! You are {age} years old."

# Выражения внутри f-strings
x = 10
y = 20
result = f"The sum of {x} and {y} is {x + y}"

# Вызов функций
import math
number = 16
result = f"Square root of {number} is {math.sqrt(number):.2f}"

# Форматирование
price = 19.99
f"Price: ${price:.2f}"
f"Percentage: {0.75:.1%}"
f"Binary: {42:b}"          # "Binary: 101010"
f"Hex: {255:x}"            # "Hex: ff"

# Выравнивание
text = "hello"
f"{text:>10}"              # "     hello"
f"{text:<10}"              # "hello     "
f"{text:^10}"              # "  hello   "
f"{text:*^10}"             # "**hello***"

# Работа с датами
from datetime import datetime
now = datetime.now()
f"Current time: {now:%Y-%m-%d %H:%M:%S}"

# Отладочная информация (Python 3.8+)
x = 42
f"{x=}"                    # "x=42"
f"{x=:.2f}"                # "x=42.00"

# Многострочные f-strings
name = "Alice"
age = 25
message = f"""
Hello, {name}!
You are {age} years old.
```

```
Next year you will be {age + 1}.
"""
```

Специальные случаи и продвинутые техники:

```
python
```

```python
# Условное форматирование
status = "active"
f"Status: {'🟢' if status == 'active' else '🔴'}"

# Форматирование списков
numbers = [1, 2, 3, 4, 5]
f"Numbers: {', '.join(map(str, numbers))}"

# Динамическое форматирование
width = 10
precision = 2
value = 3.14159
f"{value:{width}.{precision}f}"  # "      3.14"

# Экранирование фигурных скобок
f"{{This is literal braces}} but {42} is a variable"
# "{This is literal braces} but 42 is a variable"

# Вложенные f-strings (Python 3.12+)
name = "Alice"
f"Hello, {f'{name.upper()}'}"  # "Hello, ALICE"
```

---

19. Основы работы с регулярными выражениями

Что такое регулярные выражения:

Регулярные выражения (regex) - это мощный инструмент для поиска, сопоставления и манипуляции текстом на основе шаблонов.

Базовый синтаксис:

Литералы и метасимволы:

```python
# Литералы - обычные символы
"hello" соответствует строке "hello"

# Метасимволы - специальные символы
. ^ $ * + ? { } [ ] \ | ( )
```

Основные метасимволы:

```python
.       # любой символ (кроме новой строки)
^       # начало строки
$       # конец строки
*       # 0 или более повторений предыдущего символа
+       # 1 или более повторений
?       # 0 или 1 повторение (опциональный символ)
\       # экранирование метасимволов
|       # логическое ИЛИ
```

Символьные классы:

```python
[abc]      # любой из символов a, b, c
[a-z]      # любая строчная буква
[A-Z]      # любая заглавная буква
[0-9]      # любая цифра
[a-zA-Z]   # любая буква
[^abc]     # любой символ КРОМЕ a, b, c

# Предопределенные классы
\d         # цифра [0-9]
\D         # не цифра [^0-9]
\w         # буква, цифра или подчеркивание [a-zA-Z0-9_]
\W         # не буква, не цифра, не подчеркивание
\s         # пробельный символ (пробел, табуляция, новая строка)
\S         # не пробельный символ
```

Квантификаторы:

```python
{n}        # точно n повторений
{n,}       # n или более повторений
{n,m}      # от n до m повторений
{,m}       # до m повторений

# Примеры
\d{3}      # ровно 3 цифры
\d{2,4}    # от 2 до 4 цифр
\d{3,}     # 3 или более цифр
[a-z]{1,5} # от 1 до 5 строчных букв
```

Группы и захват:

```python
()         # группа захвата
(?:...)    # группа без захвата
(?P<name>...) # именованная группа

# Примеры
(\d{2})-(\d{2})-(\d{4})  # захват дня, месяца, года
(?P<day>\d{2})-(?P<month>\d{2})-(?P<year>\d{4})  # именованные группы
```

Примеры распространенных шаблонов:

Email:

```python
email_pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}
```

Телефон:

```python
phone_pattern = r'^\+?1?[-.\s]?\(?([0-9]{3})\)?[-.\s]?([0-9]{3})[-.\s]?([0-9]{4})$
```

Вопрос 20

Модуль re в Python. Примеры использования регулярных выражений

Технологии обработки данных

---

1. Назначение модуля re

Модуль **re** предоставляет поддержку регулярных выражений в Python - инструмента для поиска, сопоставления и обработки текстовых данных по шаблонам.

2. Основные функции

re.search(pattern, string) - поиск первого вхождения

re.match(pattern, string) - проверка с начала строки

re.findall(pattern, string) - все совпадения в виде списка

re.sub(pattern, repl, string) - замена совпадений

re.split(pattern, string) - разделение строки

3. Специальные символы

Символ	Описание	Пример
\d	Любая цифра (0-9)	\d{3} → 123
\w	Буквенно-цифровой символ	\w+ → word
\s	Пробельный символ	\s+ → пробелы
.	Любой символ	a.b → axb
^	Начало строки	^abc
$	Конец строки	abc$
*	0 или более повторений	ab*
+	1 или более повторений	ab+
{n,m}	От n до m повторений	\d{2,4}

4. Практические примеры

Поиск телефонных номеров

```
import re text = "Телефон: +7(495)123-45-67" pattern = r"\+7\(\d{3}\)\d{3}-\d{2}-\d{2}" result = re.search(pattern, text) print(result.group()) # +7(495)123-45-67
```

Валидация email

```
email = "user@example.com" pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$" if re.match(pattern, email): print("Корректный email")
```

Извлечение всех чисел

```
text = "Цены: 100р, 250р, 75р" prices = re.findall(r"\d+", text) print(prices) # ['100', '250', '75']
```

Замена даты (формат ДД.ММ.ГГГГ → ГГГГ-ММ-ДД)

```
text = "дата: 15.03.2024" new_text = re.sub(r"(\d{2})\.(\d{2})\.(\d{4})", r"\3-\2-\1", text)
print(new_text) # дата: 2024-03-15
```

5. Группы захвата

```
# Именованные группы pattern = r"(?P<day>\d{2})\.(?P<month>\d{2})\.(?P<year>\d{4})" match =
re.search(pattern, "15.03.2024") if match: print(match.group('year')) # 2024
print(match.groupdict()) # {'day': '15', 'month': '03', 'year': '2024'}
```

6. Компиляция и флаги

```
# Компиляция для повторного использования pattern = re.compile(r"\d+", re.IGNORECASE) numbers =
pattern.findall("123 ABC 456") # Основные флаги: # re.IGNORECASE - игнорирование регистра #
re.MULTILINE - многострочный режим # re.DOTALL - точка включает \n
```

Важно: Регулярные выражения - мощный инструмент для валидации данных, извлечения информации и обработки текста. Используются для парсинга логов, валидации форм, обработки больших текстовых массивов.