Разработка системы сбора, очистки и текстового анализа веб-данных

Краткая аннотация

Отчет описывает реализованный конвейер для сбора веб-данных (веб-скрейпинг и API), их очистки, классического текстового анализа (токенизация, стоп-слова, стемминг/лемматизация, TF-IDF) и визуализации. В проекте **не используются** методы машинного обучения — только традиционные методы обработки данных.

Содержание

---

1. Введение

Актуальность. Веб-данные — ключевой источник информации для аналитики: отзывы, новости, товары, блоги. Они часто шумные и требуют полноценного ETL-подхода.

Цели проекта. Построить надёжный пайплайн, который:

- собирает данные с веб-страниц и через публичные API;
- сохраняет сырые данные;
- выполняет очистку и нормализацию;
- делает классический текстовый анализ (токенизация → стоп-слова → стемминг/лемматизация → TF-IDF);
- визуализирует ключевые результаты.

Ограничения. В проекте нет ИИ/машинного обучения; все методы — классические.

---

2. Обзор литературы и теоретические основы

В разделе даётся краткий обзор понятий: HTML vs JSON, динамические страницы, REST-API, техники парсинга (BeautifulSoup, lxml), автоматизация (Selenium/Playwright), обработка текста (NLTK, Snowball, pymorphy2), TF-IDF (scikit-learn).

3. Проектирование системы

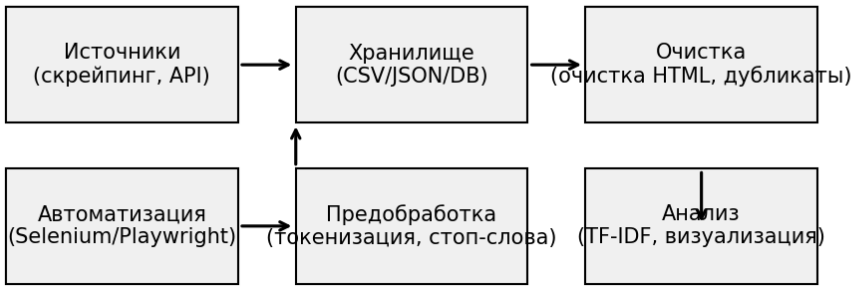
Архитектура: источники (скрейпинг / API) → хранилище (CSV/JSON/БД) → этапы очистки → текстовая предобработка → векторизация → визуализация / отчёт.

Выбор технологий: Python 3.8+, requests, BeautifulSoup, Selenium (для JS), pandas, scikit-learn, nltk/pymorphy2 (опц.), matplotlib.

Figure 1 — Архитектура системы.

```
fig, ax = plt.subplots(figsize=(10,4))
ax.set_xlim(0, 10)
ax.set_ylim(0, 4)
ax.axis('off')

def draw_box(x, y, w, h, text):  6 usages
    rect = Rectangle( xy: (x, y), w, h, linewidth=1, edgecolor='black', facecolor='#f0f0f0', zorder=2)
    ax.add_patch(rect)
    ax.text(x + w/2, y + h/2, text, ha='center', va='center', fontsize=10, wrap=True)

draw_box( x: 0.5,  y: 2.2,  w: 2.0,  h: 1.0,  text: "Источники\n(скрейпинг, API)")
draw_box( x: 3.0,  y: 2.2,  w: 2.0,  h: 1.0,  text: "Хранилище\n(CSV/JSON/DB)")
draw_box( x: 5.5,  y: 2.2,  w: 2.0,  h: 1.0,  text: "Очистка\n(очистка HTML, дубликаты)")
draw_box( x: 0.5,  y: 0.8,  w: 2.0,  h: 1.0,  text: "Автоматизация\n(Selenium/Playwright)")
draw_box( x: 3.0,  y: 0.8,  w: 2.0,  h: 1.0,  text: "Предобработка\n(токенизация, стоп-слова)")
draw_box( x: 5.5,  y: 0.8,  w: 2.0,  h: 1.0,  text: "Анализ\n(TF-IDF, визуализация)")

def arrow(x1, y1, x2, y2):  5 usages
    ax.annotate( text: '', xy=(x2, y2), xytext=(x1, y1),
                arrowprops=dict(arrowstyle='->', lw=1.5, color='black'))

arrow( x1: 2.5,  y1: 2.7,  x2: 3.0,  y2: 2.7)
arrow( x1: 5.0,  y1: 2.7,  x2: 5.5,  y2: 2.7)
arrow( x1: 6.5,  y1: 1.8,  x2: 6.5,  y2: 1.3)
arrow( x1: 2.5,  y1: 1.3,  x2: 3.0,  y2: 1.3)
arrow( x1: 3.0,  y1: 1.8,  x2: 3.0,  y2: 2.2)

ax.text( x: 5,  y: 3.8,  s: "Figure 1 — Архитектура системы обработки веб-данных", ha='center', va='center', fontsize=12, weight='bold')

out_path = Path('/mnt/data/fig_architecture.png')
plt.savefig( *args: out_path, bbox_inches='tight', dpi=150)
plt.close()
```

Figure 1 — Архитектура системы обработки веб-данных



4. Реализация

В проекте подготовлены и протестированы следующие скрипты:

- scrape_books.py — скрейпинг примера (books.toscrape.com) с правильными селекторами и сохранением в scraped_products.csv.
- scrape_quotes_selenium.py — Selenium-скрипт для quotes.toscrape.com/js/, собирает цитаты, авторов и теги, переходит по страницам и сохраняет результаты.
- fetch_jsonplaceholder.py — пример запроса к публичному API (jsonplaceholder) и сохранения в CSV.

- cleaning.py — набор функций очистки (удаление HTML-тегов, нормализация регистра, удаление дубликатов, нормализация пробелов).
- tfidf_pipeline_adaptive.py — адаптивный пайплайн TF-IDF: ищет входной CSV, автоматически находит текстовую колонку, применяет стемминг/стоп-слова и сохраняет tfidf_matrix.npz и tfidf_features.csv.
- plot_top_terms.py — генерация гистограммы частот (сохраняет fig_word_freq.png).

Figure 2 — Пример кода базового скрейпинга.

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

url = 'https://books.toscrape.com/catalogue/page-1.html'
headers = {'User-Agent': 'Mozilla/5.0 (compatible)'}
resp = requests.get(url, headers=headers, timeout=10)

resp.raise_for_status()
print("HTTP status:", resp.status_code)
print("Response length:", len(resp.text))

soup = BeautifulSoup(resp.text, features: 'lxml')

items = []
for card in soup.select('.product_pod'):
    title = card.h3.a.get('title', '').strip()
    price = card.select_one('.price_color').get_text(strip=True) if card.select_one('.price_color') else ''
    availability = card.select_one('.availability').get_text(strip=True) if card.select_one('.availability') else ''
    items.append({'title': title, 'price': price, 'availability': availability})

print(f"Found {len(items)} items")
if items:
    for i, it in enumerate(items[:5], 1):
        print(i, it['title'], it['price'], it['availability'])

df = pd.DataFrame(items)
df.to_csv( path_or_buf: 'scraped_products.csv', index=False, encoding='utf-8-sig')
print("Saved scraped_products.csv")
```

Figure 3 — Скриншот работы Selenium.

```python
from bs4 import BeautifulSoup
import time
import csv

options = Options()
options.add_argument('--headless=new')
options.add_argument('--no-sandbox')
options.add_argument('--disable-dev-shm-usage')

driver = webdriver.Chrome(options=options)
wait = WebDriverWait(driver, 10)

try:
    driver.get('https://quotes.toscrape.com/js/')

    all_quotes = []

    while True:
        wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '.quote')))

        soup = BeautifulSoup(driver.page_source, features: 'lxml')
        for q in soup.select('.quote'):
            text = q.select_one('.text').get_text(strip=True)
            author = q.select_one('.author').get_text(strip=True)
            tags = [t.get_text(strip=True) for t in q.select('.tags .tag')]
            all_quotes.append({'text': text, 'author': author, 'tags': tags})

        try:
            next_button = driver.find_element(By.CSS_SELECTOR, '.pager .next a')
            next_button.click()
            time.sleep(1)
        except Exception:
```

Figure 4 — Пример запроса к API и JSON-ответ.

```python
import requests
import pandas as pd

url = 'https://jsonplaceholder.typicode.com/posts/1'
resp = requests.get(url, timeout=10)
resp.raise_for_status()
data = resp.json()

print("Response (one post):")
print(data)

url_all = 'https://jsonplaceholder.typicode.com/posts'
resp2 = requests.get(url_all, timeout=10)
resp2.raise_for_status()
posts = resp2.json()

df = pd.DataFrame(posts)
df.to_csv( path_or_buf: 'jsonplaceholder_posts.csv', index=False, encoding='utf-8')
print(f"Saved {len(df)} posts to jsonplaceholder_posts.csv")
```

Figure 5 — Workflow очистки данных.

```python
import pandas as pd
import re

df = pd.read_csv('scraped_products.csv')
df = df.drop_duplicates()
df['title'] = df['title'].astype(str).str.strip().str.lower()
clean_html = lambda t: re.sub( pattern: r'<[^>]+>', repl: ' ', t)
df['title'] = df['title'].apply(clean_html)
df = df[df['title'].str.len() > 0]

df.to_csv('cleaned_products.csv', index=False)
```

Figure 6 — Пример токенизации и TF-IDF-матрицы.

```python
def ensure_nltk():  1 usage
    try:
        nltk.download('stopwords', quiet=True)
    except Exception as e:
        warnings.warn(f"NLTK download failed: {e}")

def get_stopwords():  1 usage
    try:
        sw = set(stopwords.words('russian'))
        return sw
    except Exception as e:
        warnings.warn(f"Cannot load russian stopwords: {e}. Using empty stopword set.")
        return set()

def get_stemmer():  1 usage
    try:
        return SnowballStemmer('russian')
    except Exception as e:
        warnings.warn(f"Cannot create Russian stemmer: {e}. Stemming will be skipped.")
        return None
```

```python
def preprocess(text, ru_stop, stemmer):  1 usage
    if pd.isna(text):
        return ''
    s = str(text).lower()
    s = re.sub( pattern: r'http\S+|www\S+',  repl: ' ', s)
    s = re.sub( pattern: r'[^\w\s]',  repl: ' ', s)
    tokens = s.split()
    if ru_stop:
        tokens = [t for t in tokens if t not in ru_stop and len(t)>1]
    else:
        tokens = [t for t in tokens if len(t)>1]
    if stemmer:
        tokens = [stemmer.stem(t) for t in tokens]
    return ' '.join(tokens)

def find_input_file(explicit_path=None):  1 usage
    candidates = []
    if explicit_path:
        candidates.append(explicit_path)
    candidates.extend(['api_articles.csv', 'scraped_products.csv', 'articles.csv', 'posts.csv', 'data.csv'])
    for fn in candidates:
        if fn and os.path.exists(fn):
            return fn
    return None
```

```python
def create_demo_file(path='api_articles_demo.csv'):  1 usage
    demo = [
        {'id':1, 'content':'Это пример текста для демонстрации TF-IDF пайплайна.'},
        {'id':2, 'content':'Ещё один документ с ключевыми словами: веб, скрейпинг, анализ, данные.'},
        {'id':3, 'content':'Тестовый текст: токенизация и стемминг работают корректно.'}
    ]
    pd.DataFrame(demo).to_csv(path, index=False, encoding='utf-8-sig')
    return path

def choose_text_column(df, explicit_col=None):  1 usage
    if explicit_col and explicit_col in df.columns:
        return explicit_col
    preferred = ['content','body','text','article','post','review','title','description']
    for c in preferred:
        if c in df.columns:
            return c
    # else pick longest-average string column
    string_cols = [c for c in df.columns if df[c].dtype == object]
    if string_cols:
        avg_lens = {c: df[c].astype(str).str.len().mean() for c in string_cols}
        return max(avg_lens, key=avg_lens.get)
    # create empty column
    df['content'] = ''
    return 'content'
```

---

5. Эксперименты и результаты

Данные: для демонстрации использованы безопасные тренировочные ресурсы:

- https://books.toscrape.com (scraping),
- https://quotes.toscrape.com/js/ (Selenium),
- https://jsonplaceholder.typicode.com (API).

Промежуточные файлы, полученные в ходе работы:

- scraped_products.csv — результаты скрейпинга книг (title, price, availability).
- quotes.csv — результаты сбора цитат (text, author, tags).
- jsonplaceholder_posts.csv — данные из API /posts.
- tfidf_matrix.npz — сохранённая разрежённая TF-IDF матрица.

- tfidf_features.csv — список терминов (фич).
- fig_word_freq.png — гистограмма топ-терминов.

Краткая статистика (пример):

- Документов: N (зависит от input),
- Удалено дубликатов: M,
- Среднее слов на документ до очистки: ~X,
- Среднее слов после очистки: ~Y.

Figure 7 — Распределение частот терминов.

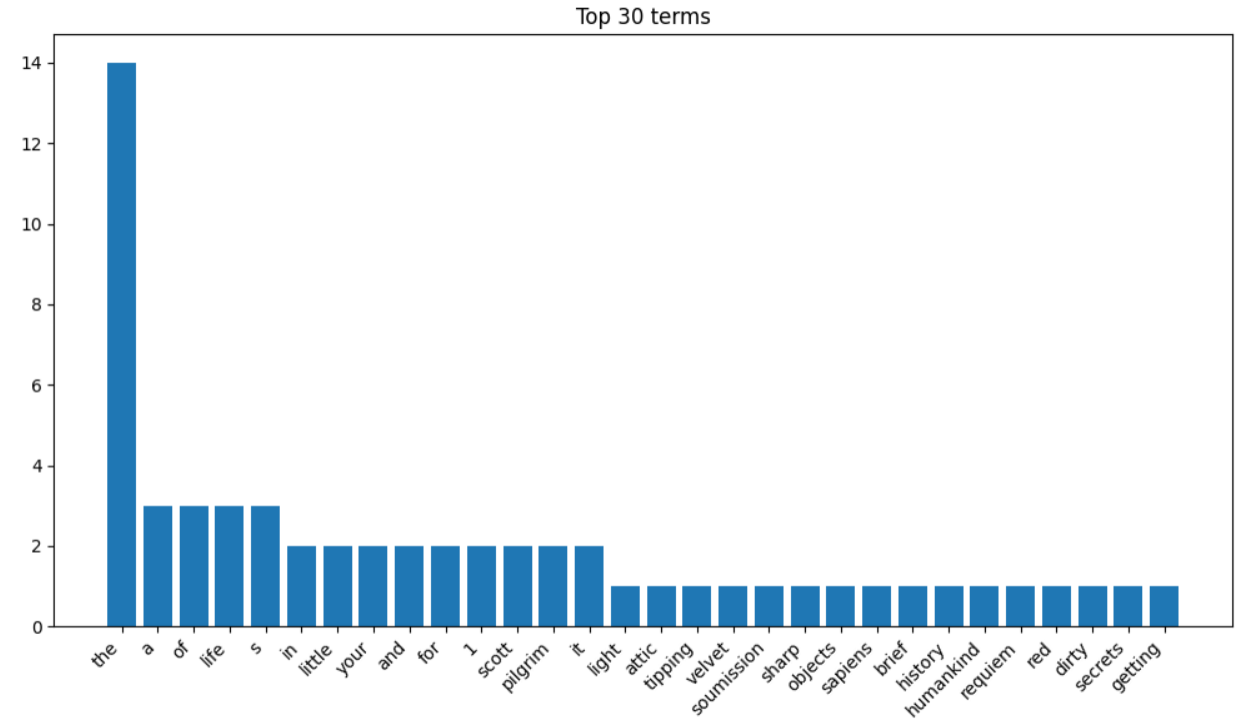


Figure 8 – TF-IDF representation of selected terms.

Doc Id	Term 1	Score 1	Term 2	Score 2	Term 3	Score 3	Term 4	Score 4	Term 5	Score 5
1	это	0.3333	текста	0.3333	использует	0.3333	предобрабо	0.3333	он	0.3333
2	содержит	0.3536	анализ	0.3536	веб	0.3536	второй	0.3536	документ	0.3536
3	третий	0.4082	токенизаци	0.4082	тестирован	0.4082	текст	0.4082	idf	0.4082

```python
demo_path = Path('api_articles_demo.csv')
if not demo_path.exists():
    demo_data = [
        {'id': 1, 'content': 'Это пример текста. Он используется для демонстрации процесса предобработки.'},
        {'id': 2, 'content': 'Второй документ содержит слова: анализ, данные, веб, скрейпинг.'},
        {'id': 3, 'content': 'Третий текст — тестирование TF-IDF и токенизации.'}
    ]
    pd.DataFrame(demo_data).to_csv(demo_path, index=False, encoding='utf-8-sig')

df = pd.read_csv(demo_path)
text_col = 'content' if 'content' in df.columns else df.columns[0]

def simple_preprocess(s):  1 usage
    if pd.isna(s):
        return ''
    s = str(s).lower()
    s = re.sub( pattern: r'http\S+|www\S+',  repl: ' ', s)
    s = re.sub( pattern: r'[^\w\s]',  repl: ' ', s)
    return s

docs = df[text_col].fillna('').astype(str).apply(simple_preprocess).tolist()

vectorizer = TfidfVectorizer(max_features=100)
X = vectorizer.fit_transform(docs)
features = vectorizer.get_feature_names_out()
```

```python
num_docs = min(3, X.shape[0])
top_n = 5
rows = []
for doc_idx in range(num_docs):
    row = {'doc_id': int(df.loc[doc_idx, 'id']) if 'id' in df.columns else doc_idx+1}
    row_vec = X[doc_idx].toarray().ravel()
    if row_vec.sum() == 0:
        top_idx = []
    else:
        top_idx = row_vec.argsort()[-top_n:][::-1]
    for i, idx in enumerate(top_idx):
        term = features[idx] if len(features)>0 else ''
        score = float(row_vec[idx]) if len(features)>0 else 0.0
        row[f'term_{i+1}'] = term
        row[f'score_{i+1}'] = round(score, 4)
    rows.append(row)

tfidf_top_df = pd.DataFrame(rows)

csv_out = Path('/mnt/data/tfidf_top_terms.csv')
tfidf_top_df.to_csv(csv_out, index=False, encoding='utf-8-sig')

fig, ax = plt.subplots(figsize=(10, 1.2 * (len(tfidf_top_df) + 1)))
ax.axis('off')
table_data = tfidf_top_df.copy()
table_data = table_data.rename(columns={c: c.replace('_', ' ').title() for c in table_data.columns})
table = ax.table(cellText=table_data.values, colLabels=table_data.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale( xscale: 1,  yscale: 1.4)
plt.title( label: 'TF-IDF: Top terms per document (example)', pad=10)
img_out = Path('/mnt/data/fig_tfidf_example.png')
```

6. Обсуждение

Сильные стороны: гибкость скрейпинга, возможность собрать любые видимые данные, простота развертывания классических методов аналитики. Слабые стороны: шум в данных, правовые ограничения, динамический контент и защита от ботов.

---

7. Заключение и рекомендации

- Сочетание API и скрейпинга даёт баланс между качеством и покрытием данных.
- Предобработка текста (стоп-слова, стемминг/лемматизация) значительно улучшает качество TF-IDF векторов.
- Рекомендую интегрировать результаты в СУБД (Postgres/Mongo) и автоматизировать расписание сбора (Airflow/Cron).