|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Модели и методы предиктивной аналитики»

**Практическое занятие № 3**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Крынецкий Борис Алексеевич, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2025 г.

СОДЕРЖАНИЕ

[ПРАКТИЧЕСКАЯ РАБОТА 3 3](#_Toc212570119)

[Введение 3](#_Toc212570120)

[Шаги выполнения 3](#_Toc212570121)

[Результат работы: 8](#_Toc212570122)

[Вывод: 8](#_Toc212570123)

[Список использованных источников и литературы: 9](#_Toc212570124)

[ПРИЛОЖЕНИЯ 10](#_Toc212570125)

[Приложение А 11](#_Toc212570126)

ПРАКТИЧЕСКАЯ РАБОТА 3

Введение

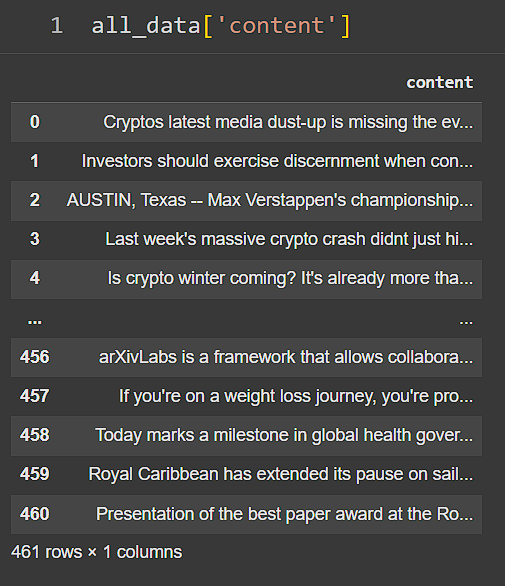
**Цель**

Обработать текстовые данные и проанализировать их.

Шаги выполнения

Для выполнения работы были использованы данные, полученные через API новостного портала <https://newsapi.org/>

В результате запроса был сформирован набор публикаций по различным категориям (технологии, бизнес, спорт и др.).



**Рисунок 1 – Данные**

1. На первом этапе данные были очищены и приведены к однородному виду. Провели предобработку данных.

Токенизация:

* Разбиение на слова
* Разбиение на предложения

Очистка текста:

* Удаление URL-адресов, HTML-тегов, эмодзи
* Удаление упоминаний (@) и хештегов (#)
* Удаление специальных символов и цифр
* Нормализация пробелов

Нормализация:

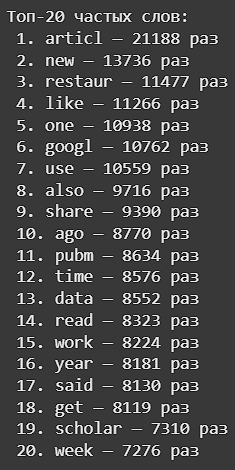
* Приведение к нижнему регистру
* Удаление стоп-слов (предлоги, союзы)
* Лемматизация (приведение к нормальной форме)
* Стемминг (обрезание до корня)



**Рисунок 2 – Предобработанные данные**

1. Частотный анализ

* Подсчет встречаемости слов с использованием Counter
* Выявление наиболее значимых терминов

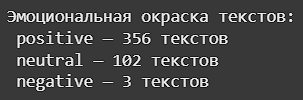


**Рисунок 3 – Топ 20 слов по встречаемости**

1. Эмоциональная окраска текстов (Анализ тональности).

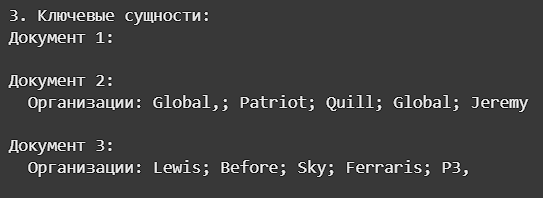
Для оценки тональности был применён простейший словарный метод, основанный на подсчёте позитивных и негативных слов.

Каждый текст классифицировался как positive, negative или neutral в зависимости от соотношения ключевых слов.



**Рисунок 4 – Анализ тональности**

1. Выявление ключевые сущности:



**Рисунок 5 – Анализ ключевых сущностей**

Наиболее часто встречались **организации и компании**, упомянутые в новостных материалах, а также **даты событий**, что подтверждает актуальность и деловой характер корпуса данных.

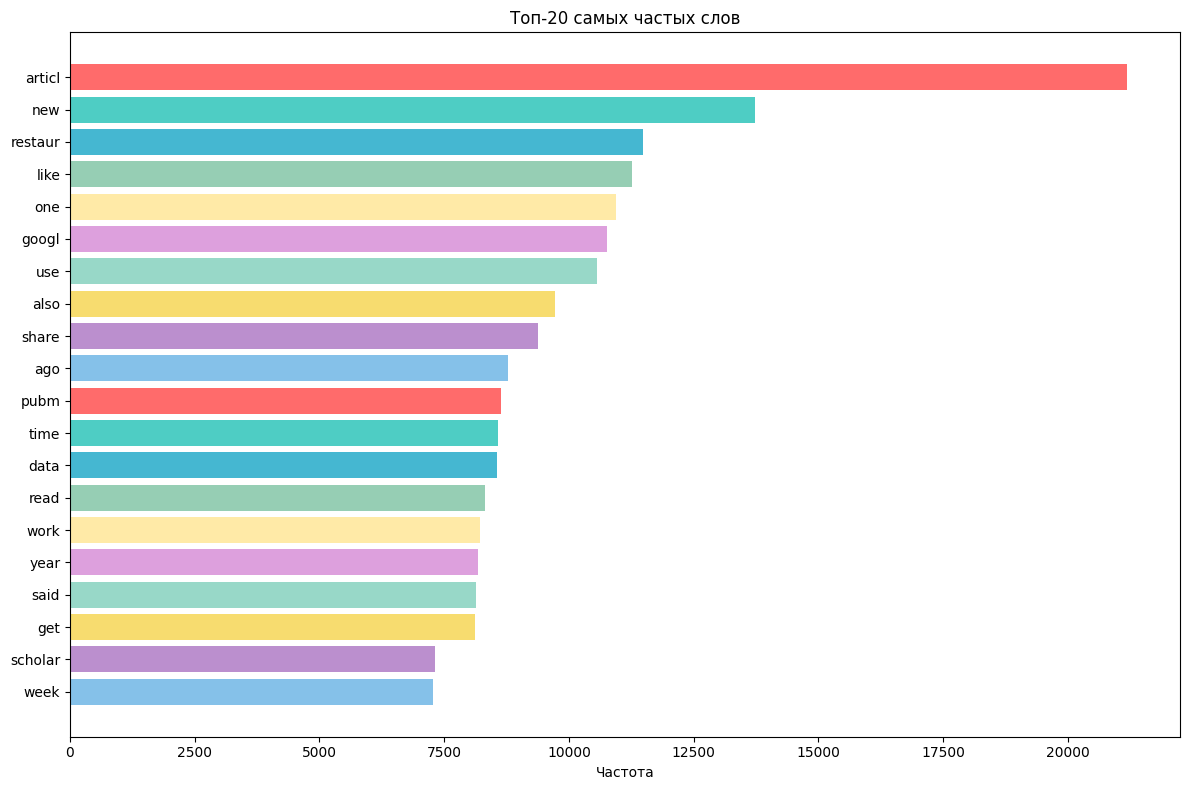
1. Классификация по категориям:

Для классификации текстов использовался словарный метод с подбором ключевых слов по тематикам: technology, sports, business, health и др.

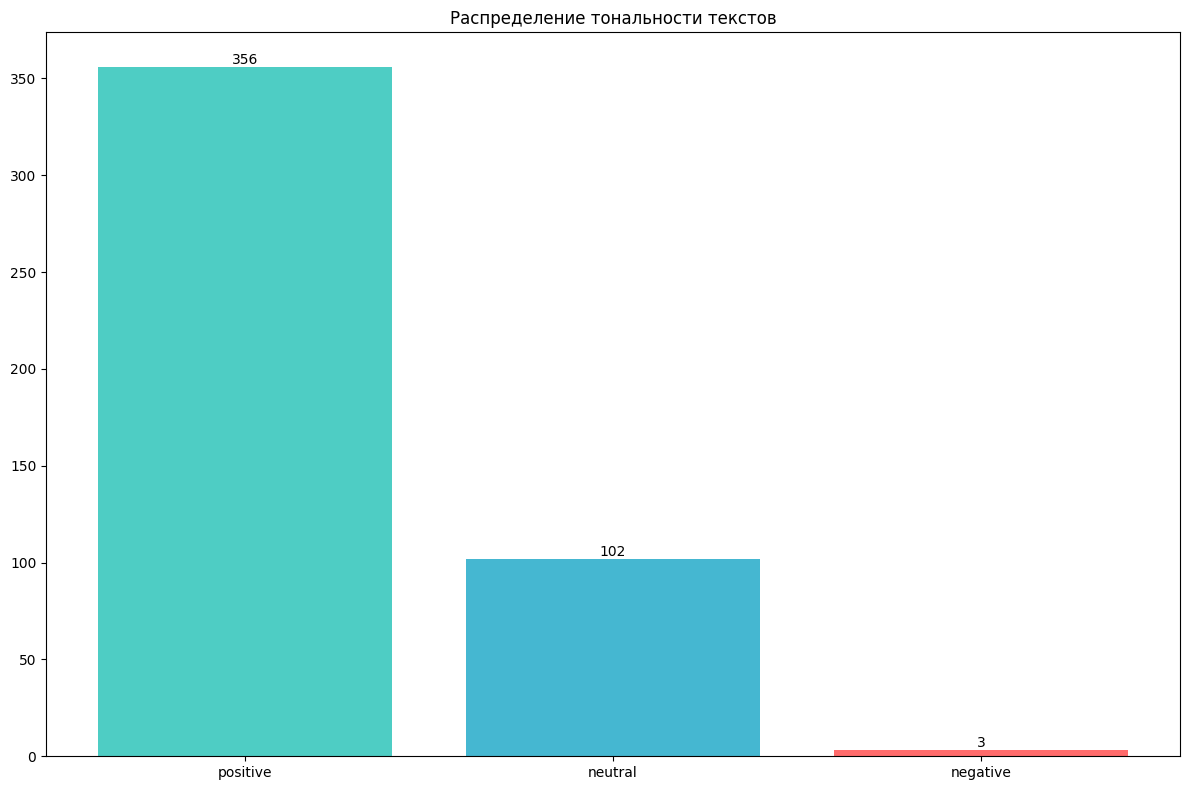


**Рисунок 6 – Классификация по категориям**

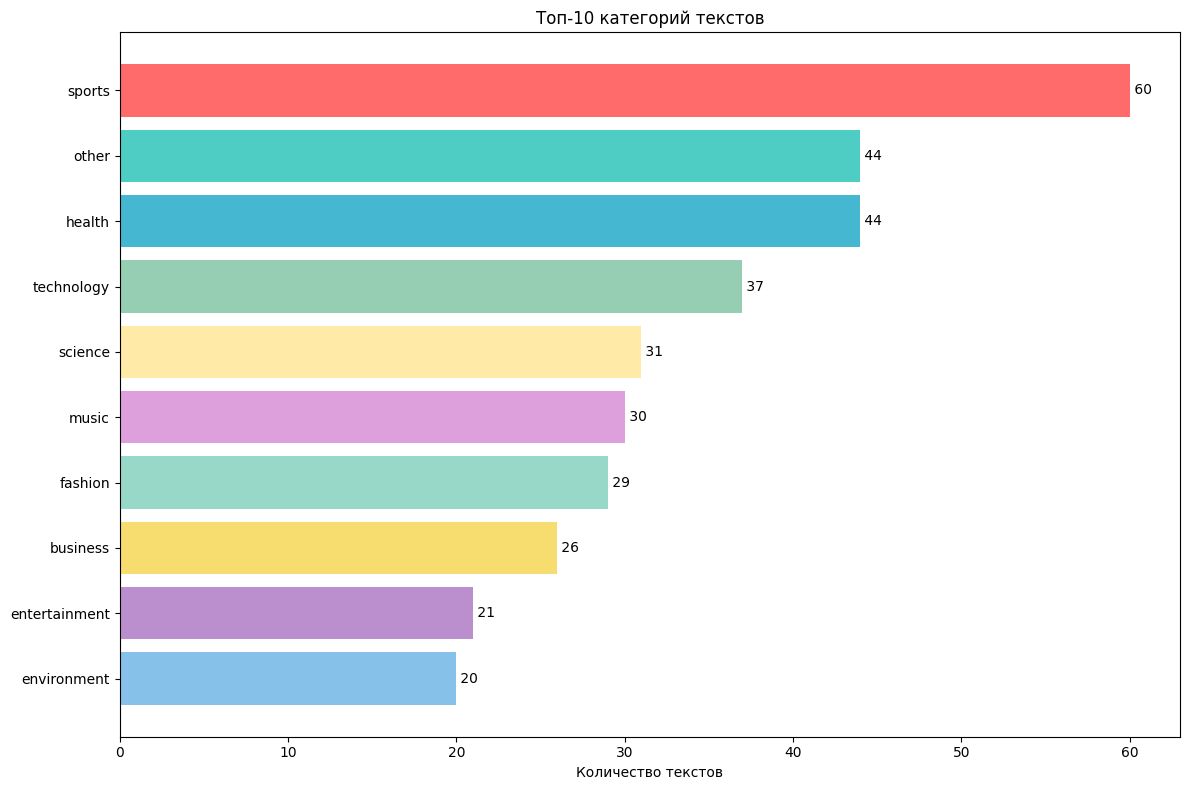
Также были построены визуализации с помощью библиотек Matplotlib и Seaborn.



**Рисунок 6 – Распределение топ-20 самых частых слов**



**Рисунок 7 – Распределение на тональности текстов**

 **Рисунок 8 – Распределение по категориям текстов**

Результат работы:

Данную работу можете увидеть в блокноте Jupyter Notebook.

<https://drive.google.com/file/d/1ajgJfs2XAOQtqXf-gfTRctmRtEFaVxJV/view?usp=sharing>

Вывод:

В ходе работы выполнена обработка и анализ текстовых данных: произведена очистка, токенизация, лемматизация, частотный анализ и оценка тональности.

Список использованных источников и литературы:

1. Ростовцев В.С. Искусственные нейронные сети,   
   Издательство "Лань", 2019. — 216 с. — URL: https://e.lanbook.com/book/122180
2. Араки М. Манга: Машинное обучение,   
   Издательство "ДМК Пресс", 2020. — 214 с. — URL: <https://e.lanbook.com/book/179473>
3. Платонов, А. В. Машинное обучение : учебное пособие для вузов / А. В. Платонов. — Москва : Издательство Юрайт, 2022. — 85 с. — (Высшее образование). — ISBN 978-5-534-15561-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/508804

ПРИЛОЖЕНИЯ

Приложение А — код на языке программирование Python

Приложение А

|  |
| --- |
| import os  import re  import warnings  from collections import Counter  import requests  from bs4 import BeautifulSoup  import time  import pandas as pd  import numpy as np  import nltk  from nltk.tokenize import word\_tokenize, sent\_tokenize  from nltk.corpus import stopwords  from nltk.stem import PorterStemmer, WordNetLemmatizer  import matplotlib.pyplot as plt  import seaborn as sns  from wordcloud import WordCloud  import plotly.express as px  import plotly.graph\_objects as go  from plotly.subplots import make\_subplots  warnings.filterwarnings("ignore")  # Настройка стилей для визуализации  plt.style.use('default')  sns.set\_palette("husl")  colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FFEAA7', '#DDA0DD', '#98D8C8', '#F7DC6F', '#BB8FCE', '#85C1E9']  # Загрузка данных NLTK  print("Загрузка ресурсов NLTK...")  resources = ['punkt', 'stopwords', 'wordnet', 'punkt\_tab', 'averaged\_perceptron\_tagger']  for resource in resources:      try:          if resource in ['punkt', 'punkt\_tab']:              nltk.data.find(*f*'tokenizers/{resource}')          else:              nltk.data.find(*f*'corpora/{resource}')      except *LookupError*:          nltk.download(resource)          print(*f*"Загружено: {resource}")  # Функция парсинга текста с веб-страницы по URL  *def* parse\_text\_from\_url(*url*):      try:          if pd.isna(url) or not isinstance(url, *str*) or not url.startswith(('http://', 'https://')):              return ''          print(*f*"Парсинг URL: {url}...")          headers = {              'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'          }          response = requests.get(url, *headers*=headers, *timeout*=10)          response.raise\_for\_status()          soup = BeautifulSoup(response.content, 'html.parser')          for script in soup(["script", "style", "nav", "footer", "header"]):              script.decompose()          text\_elements = soup.find\_all(['p', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'article', 'div'])          all\_text = []          for element in text\_elements:              text = element.get\_text().strip()              if text and len(text) > 20:                  all\_text.append(text)          full\_text = ' '.join(all\_text)          full\_text = re.sub(*r*'\s+', ' ', full\_text).strip()          time.sleep(1)          return full\_text if full\_text else ''      except requests.exceptions.RequestException as e:          print(*f*"Ошибка запроса для {url}: {e}")          return ''      except *Exception* as e:          print(*f*"Ошибка парсинга для {url}: {e}")          return ''  # Загрузка данных  all\_data = pd.read\_csv('news\_data.csv')  all\_data  text\_column = 'content' if 'content' in all\_data.columns else \               ('text' if 'text' in all\_data.columns else all\_data.columns[0])  print(*f*"Используется колонка: '{text\_column}'")  all\_data[text\_column] = all\_data[text\_column].fillna('')  all\_data['combined\_raw'] = all\_data[text\_column].astype(*str*)  all\_data['content']  all\_data['url']  # Парсинг текста из URL  parsed\_texts = []  for index, url in enumerate(all\_data['url']):      print(*f*"Парсинг {index + 1}/{len(all\_data)}: {url}...")      text = parse\_text\_from\_url(url)      parsed\_texts.append(text)      if (index + 1) % 10 == 0:          print(*f*"Прогресс: {index + 1}/{len(all\_data)} завершено")  all\_data = all\_data.head(len(parsed\_texts)).copy()  all\_data['parsed\_content'] = parsed\_texts  successful\_parses = sum(1 for text in parsed\_texts if text.strip())  print(*f*"\nУспешно распарсено: {successful\_parses}/{len(parsed\_texts)} URL")  print(*f*"Процент успеха: {successful\_parses/len(parsed\_texts)\*100*:.1f*}%")  text\_column = 'parsed\_content'  print(*f*"Используется колонка: '{text\_column}'")  all\_data[text\_column] = all\_data[text\_column].fillna('')  all\_data['combined\_raw'] = all\_data[text\_column].astype(*str*)  # Инициализация инструментов NLP  try:      stop\_words = *set*(stopwords.words('english'))  except:      stop\_words = *set*()  lemmatizer = WordNetLemmatizer()  stemmer = PorterStemmer()  # Регулярные выражения для очистки  URL = re.compile(*r*'https?://\S+|www\.\S+')  HTML = re.compile(*r*'<[^>]+>')  MENTION\_HASHTAG = re.compile(*r*'[@#]\w+')  EMOJI = re.compile(      "["*u*"\U0001F600-\U0001F64F"  *u*"\U0001F300-\U0001F5FF"  *u*"\U0001F680-\U0001F6FF"  *u*"\U00002600-\U000026FF"  *u*"\U00002700-\U000027BF"      "]+", *flags*=re.UNICODE)  ALPHANUMERIC = re.compile(*r*'[^0-9A-Za-z\s\-]')  # Предобработка данных  processed\_texts = []  sentence\_lists = []  token\_lists = []  sentiment\_labels = []  sentiment\_scores\_list = []  entity\_records = []  for doc\_index, raw\_text in enumerate(all\_data['combined\_raw']):      print(*f*"Обработка документа {doc\_index + 1}/{len(all\_data)}...")      try:          # 1. Очистка текста          if pd.isna(raw\_text):              cleaned\_text = ''          else:              text\_string = *str*(raw\_text)              text\_string = URL.sub(' ', text\_string)              text\_string = HTML.sub(' ', text\_string)              text\_string = EMOJI.sub(' ', text\_string)              text\_string = MENTION\_HASHTAG.sub(' ', text\_string)              text\_string = ALPHANUMERIC.sub(' ', text\_string)              text\_string = re.sub(*r*'\d+', ' ', text\_string)              text\_string = re.sub(*r*'\s+', ' ', text\_string).strip()              cleaned\_text = text\_string          # 2. Токенизация          if not cleaned\_text:              sentences = []              tokens = []          else:              sentences = sent\_tokenize(cleaned\_text, *language*='english')              tokens = word\_tokenize(cleaned\_text, *language*='english')          # 3. Нормализация          normalized\_tokens = []          for token in tokens:              processed\_token = token.lower().strip()              if not processed\_token or processed\_token in stop\_words or len(processed\_token) <= 2:                  continue              processed\_token = lemmatizer.lemmatize(processed\_token)              processed\_token = stemmer.stem(processed\_token)              if processed\_token and len(processed\_token) > 1:                  normalized\_tokens.append(processed\_token)          processed\_text = ' '.join(normalized\_tokens)          processed\_texts.append(processed\_text)          sentence\_lists.append(sentences)          token\_lists.append(normalized\_tokens)          # 4. Анализ тональности          if not raw\_text or *str*(raw\_text).strip() == '':              sentiment\_label = 'neutral'              sentiment\_score = 0.0          else:              text\_lower = raw\_text.lower()              positive\_words = ['good', 'great', 'excellent', 'amazing', 'wonderful', 'best', 'love', 'like', 'positive']              negative\_words = ['bad', 'terrible', 'awful', 'horrible', 'worst', 'hate', 'dislike', 'negative']              positive\_count = sum(1 for word in positive\_words if word in text\_lower)              negative\_count = sum(1 for word in negative\_words if word in text\_lower)              total = positive\_count + negative\_count              if total == 0:                  sentiment\_label = 'neutral'                  sentiment\_score = 0.0              elif positive\_count > negative\_count:                  sentiment\_score = positive\_count / total                  sentiment\_label = 'positive'              elif negative\_count > positive\_count:                  sentiment\_score = negative\_count / total                  sentiment\_label = 'negative'              else:                  sentiment\_label = 'neutral'                  sentiment\_score = 0.0          sentiment\_labels.append(sentiment\_label)          sentiment\_scores\_list.append(sentiment\_score)          # 5. Извлечение сущностей          entities = {'PERSON': [], 'ORG': [], 'DATE': [], 'GPE': []}          if raw\_text:              words = *str*(raw\_text).split()              for word in words:                  if re.match(*r*'\d{1,2}[./-]\d{1,2}[./-]\d{2,4}', word) or re.match(*r*'\d{4}-\d{2}-\d{2}', word):                      entities['DATE'].append(word)                  elif word.istitle() and len(word) > 2:                      if word.lower() not in ['the', 'and', 'for', 'with']:                          entities['ORG'].append(word)              for key in entities:                  entities[key] = *list*(*set*(entities[key]))[:5]          entity\_records.append({              'doc\_id': doc\_index,              'persons': '; '.join(entities.get('PERSON', [])),              'orgs': '; '.join(entities.get('ORG', [])),              'dates': '; '.join(entities.get('DATE', [])),              'locations': '; '.join(entities.get('GPE', []))          })      except *Exception* as error:          print(*f*"Ошибка при обработке документа {doc\_index}: {error}")          processed\_texts.append('')          sentence\_lists.append([])          token\_lists.append([])          sentiment\_labels.append('neutral')          sentiment\_scores\_list.append(0.0)          entity\_records.append({              'doc\_id': doc\_index,              'persons': '', 'orgs': '', 'dates': '', 'locations': ''          })  all\_data['processed'] = processed\_texts  all\_data['sentences'] = sentence\_lists  all\_data['tokens'] = token\_lists  all\_data['sentiment'] = sentiment\_labels  all\_data['sentiment\_score'] = sentiment\_scores\_list  all\_data  # Частотный анализ  all\_words = []  for text in processed\_texts:      if text:          all\_words.extend(text.split())  word\_frequency = Counter(all\_words)  top\_20\_words = word\_frequency.most\_common(20)  print("1. Топ-20 частых слов:")  for position, (word, frequency) in enumerate(top\_20\_words, 1):      print(*f*"    {position}. {word} — {frequency} раз")  # Анализ тональности  print("2. Эмоциональная окраска текстов:")  sentiment\_summary = all\_data['sentiment'].value\_counts().reset\_index()  sentiment\_summary.columns = ['sentiment', 'count']  for \_, row in sentiment\_summary.iterrows():      print(*f*"    {row['sentiment']} — {row['count']} текстов")  # Сущности  print("3. Ключевые сущности:")  entities\_dataframe = pd.DataFrame(entity\_records)  documents\_to\_show = min(3, len(entities\_dataframe))  for doc\_index in range(documents\_to\_show):      entity\_row = entities\_dataframe.iloc[doc\_index]      print(*f*"Документ {doc\_index + 1}:")      if entity\_row['persons']:          print(*f*"  Люди: {entity\_row['persons']}")      if entity\_row['orgs']:          print(*f*"  Организации: {entity\_row['orgs']}")      if entity\_row['locations']:          print(*f*"  Места: {entity\_row['locations']}")      if entity\_row['dates']:          print(*f*"  Даты: {entity\_row['dates']}")      print()  # Классификация по категориям  print("4. Классификация по категориям")  CATEGORY\_KEYWORDS = {      'sports': ['game', 'player', 'team', 'season', 'win', 'score', 'championship', 'match', 'league', 'tournament'],      'technology': ['tech', 'software', 'digital', 'innovation', 'startup', 'computer', 'mobile', 'app', 'device'],      'health': ['health', 'medical', 'doctor', 'hospital', 'disease', 'treatment', 'medicine', 'patient', 'care'],      'business': ['business', 'market', 'company', 'investment', 'economic', 'profit', 'finance', 'stock', 'trade'],      'science': ['science', 'research', 'study', 'scientist', 'discovery', 'experiment', 'theory', 'data'],      'politics': ['politics', 'government', 'election', 'policy', 'minister', 'president', 'law', 'vote'],      'music': ['music', 'song', 'album', 'artist', 'tour', 'concert', 'billboard', 'band', 'performance'],      'environment': ['environment', 'climate', 'nature', 'eco', 'sustainable', 'green', 'planet', 'earth'],      'entertainment': ['entertainment', 'movie', 'film', 'celebrity', 'hollywood', 'tv', 'show', 'actor'],      'ai': ['ai', 'artificial', 'intelligence', 'machine', 'learning', 'neural', 'algorithm', 'model'],      'cybersecurity': ['cybersecurity', 'security', 'hack', 'breach', 'malware', 'firewall', 'encryption'],      'crypto': ['bitcoin', 'crypto', 'blockchain', 'token', 'defi', 'nft', 'ethereum', 'digital', 'currency'],      'gaming': ['gaming', 'game', 'player', 'console', 'video', 'esports', 'stream', 'gamer'],      'space': ['space', 'nasa', 'rocket', 'planet', 'orbit', 'satellite', 'universe', 'astronomy'],      'fashion': ['fashion', 'designer', 'collection', 'runway', 'luxury', 'style', 'clothing', 'trend'],      'travel': ['travel', 'trip', 'destination', 'hotel', 'flight', 'tourist', 'vacation', 'journey'],      'food': ['food', 'restaurant', 'recipe', 'cooking', 'cuisine', 'dish', 'meal', 'culinary'],      'books': ['book', 'author', 'novel', 'literature', 'publishing', 'reading', 'story', 'writer'],      'wellness': ['wellness', 'health', 'fitness', 'yoga', 'meditation', 'mindfulness', 'selfcare'],      'renewables': ['renewable', 'energy', 'solar', 'wind', 'green', 'sustainable', 'clean', 'power'],      'edtech': ['edtech', 'education', 'learning', 'student', 'school', 'digital', 'online', 'course'],      'robotics': ['robotics', 'robot', 'automation', 'ai', 'machine', 'autonomous', 'tech'],      'philanthropy': ['philanthropy', 'charity', 'donation', 'nonprofit', 'foundation', 'give', 'support'],      'other': []  }  predicted\_categories = []  category\_confidences = []  for text in all\_data['combined\_raw']:      if not text or *str*(text).strip() == '':          predicted\_categories.append('other')          category\_confidences.append(0.0)          continue      text\_lowercase = *str*(text).lower()      category\_scores = {}      for category, keywords in CATEGORY\_KEYWORDS.items():          match\_count = sum(1 for keyword in keywords if keyword.lower() in text\_lowercase)          category\_scores[category] = match\_count      best\_category = max(category\_scores, *key*=category\_scores.get)      total\_matches = sum(category\_scores.values())      confidence = (category\_scores[best\_category] / total\_matches) if total\_matches > 0 else 0.0      if category\_scores[best\_category] == 0:          predicted\_categories.append('other')          category\_confidences.append(0.0)      else:          predicted\_categories.append(best\_category)          category\_confidences.append(round(confidence, 3))  all\_data['predicted\_category'] = predicted\_categories  all\_data['category\_confidence'] = category\_confidences  print("Категории текстов:")  category\_counts = all\_data['predicted\_category'].value\_counts()  for category, count in category\_counts.items():      print(*f*"    {category} — {count} текстов")  # 1. Визуализация топ-20 слов  words, counts = zip(\*top\_20\_words)  plt.figure(*figsize*=(12, 8))  bars = plt.barh(words, counts, *color*=colors[:len(words)])  plt.xlabel('Частота')  plt.title('Топ-20 самых частых слов')  plt.gca().invert\_yaxis()  plt.tight\_layout()  plt.show()  # 2. Визуализация тональности  sentiment\_counts = all\_data['sentiment'].value\_counts()  plt.figure(*figsize*=(12, 8))  sentiment\_colors = {'positive': '#4ECDC4', 'negative': '#FF6B6B', 'neutral': '#45B7D1'}  colors\_list = [sentiment\_colors.get(sent, '#96CEB4') for sent in sentiment\_counts.index]  bars = plt.bar(sentiment\_counts.index, sentiment\_counts.values, *color*=colors\_list)  plt.title('Распределение тональности текстов')  for bar in bars:      height = bar.get\_height()      plt.text(bar.get\_x() + bar.get\_width()/2., height,  *f*'{*int*(height)}', *ha*='center', *va*='bottom')  plt.tight\_layout()  plt.savefig('sentiment\_analysis.png', *dpi*=300, *bbox\_inches*='tight')  plt.show()  # 3. Визуализация категорий  category\_counts = all\_data['predicted\_category'].value\_counts().head(10)  plt.figure(*figsize*=(12, 8))  bars = plt.barh(category\_counts.index, category\_counts.values, *color*=colors)  plt.xlabel('Количество текстов')  plt.title('Топ-10 категорий текстов')  plt.gca().invert\_yaxis()  for bar in bars:      width = bar.get\_width()      plt.text(width, bar.get\_y() + bar.get\_height()/2.,  *f*' {*int*(width)}', *va*='center', *ha*='left')  plt.tight\_layout()  plt.show() |