|  |
| --- |
| 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 № 1**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 1 3](#_Toc209531312)

[Введение 3](#_Toc209531313)

[Шаги выполнения 3](#_Toc209531314)

[Результат работы: 19](#_Toc209531315)

[Вывод: 19](#_Toc209531316)

[Список использованных источников и литературы: 19](#_Toc209531317)

ПРАКТИЧЕСКАЯ РАБОТА 1

Введение

**Цель**

Сформировать три отдельных набора данных, представляющих собой временные ряды, многомерные данные и наборы текста из открытых источников. Приоритетным инструментом для сбора данных является использование API.

Шаги выполнения

1. Найти подходящий API для каждого типа данных. Зарегистрироваться и получить ключи доступа к API.

Временной ряд:

<https://open-meteo.com/en/docs>

Многомерные данные:

<https://pokeapi.co/>

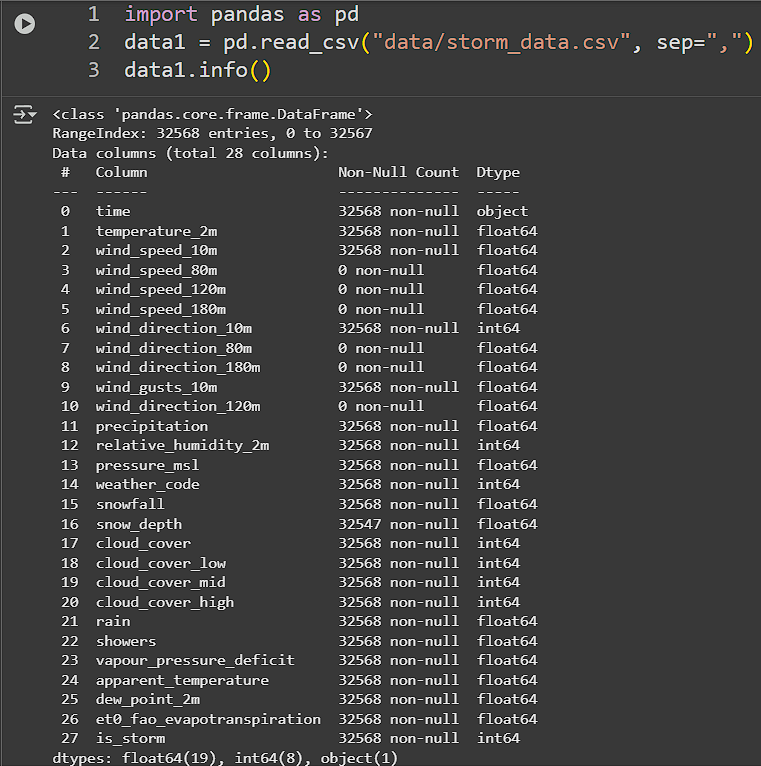
Наборы тексты:

<https://newsapi.org/>

1. Разработка скриптов для парсинга.

Листинг 1 – для временного ряда

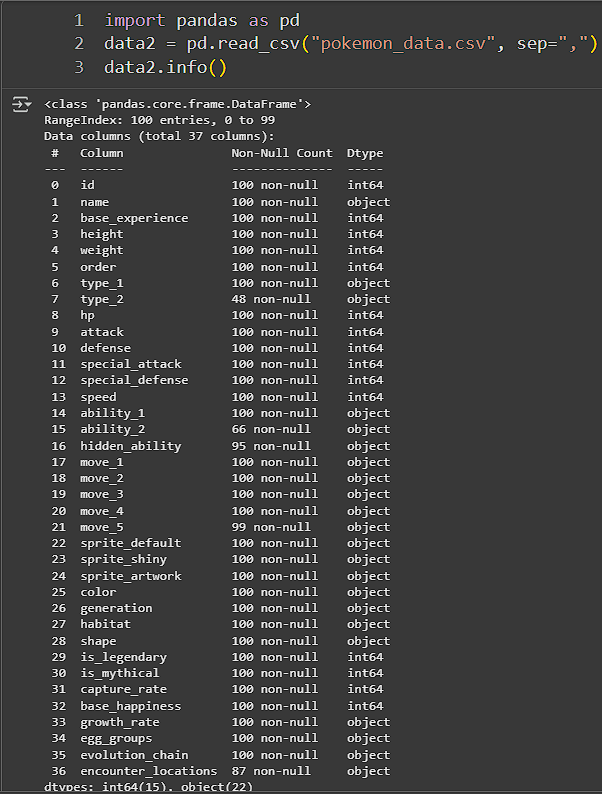
|  |
| --- |
| import requests  import pandas as pd  import numpy as np  import requests\_cache  from retry\_requests import retry  import logging  from typing import Optional, List, Dict, Tuple  from datetime import datetime, timedelta  import json  from urllib.parse import urlparse, parse\_qs  import re  import os  logging.basicConfig(  level=logging.INFO,  format='%(asctime)s - %(levelname)s - %(message)s',  handlers=[  logging.FileHandler('storm\_warning\_system.log', encoding='utf-8'),  logging.StreamHandler()  ]  )  logger = logging.getLogger(\_\_name\_\_)  cache\_session = requests\_cache.CachedSession('.cache', expire\_after=86400)  retry\_session = retry(cache\_session, retries=5, backoff\_factor=0.3)  def parse\_parameters\_from\_url(url: str) -> Tuple[Optional[float], Optional[float], List[str], Optional[str]]:  try:  parsed = urlparse(url)  query = parse\_qs(parsed.query)  latitude = float(query.get('latitude', [None])[0]) if query.get('latitude') else None  longitude = float(query.get('longitude', [None])[0]) if query.get('longitude') else None  hourly\_params = []  if 'hourly' in query:  hourly\_value = query['hourly'][0]  hourly\_params = hourly\_value.split(',') if hourly\_value else []  timezone = query.get('timezone', [None])[0]  if timezone:  timezone = requests.utils.unquote(timezone)  logger.info(f"Извлечено из URL: lat={latitude}, lon={longitude}, tz={timezone}")  logger.info(f"Параметры hourly: {hourly\_params}")  return latitude, longitude, hourly\_params, timezone  except Exception as e:  logger.error(f"Ошибка при парсинге URL: {e}")  return None, None, [], None  def get\_historical\_weather(  latitude: float,  longitude: float,  start\_date: str,  end\_date: str,  hourly\_params: Optional[List[str]] = None,  timezone: str = "Europe/Moscow"  ) -> Optional[pd.DataFrame]:  url = "https://archive-api.open-meteo.com/v1/archive"  params\_from\_url = [  "temperature\_2m", "wind\_speed\_10m", "wind\_speed\_80m", "wind\_speed\_120m",  "wind\_speed\_180m", "wind\_direction\_10m", "wind\_direction\_80m",  "wind\_direction\_180m", "wind\_gusts\_10m", "wind\_direction\_120m",  "precipitation", "relative\_humidity\_2m", "pressure\_msl", "weather\_code", "snowfall",  "snow\_depth", "cloud\_cover", "cloud\_cover\_low", "cloud\_cover\_mid",  "cloud\_cover\_high", "rain", "showers", "vapour\_pressure\_deficit",  "apparent\_temperature", "dew\_point\_2m", "et0\_fao\_evapotranspiration"  ]  final\_params = hourly\_params if hourly\_params else params\_from\_url  params = {  "latitude": latitude,  "longitude": longitude,  "start\_date": start\_date,  "end\_date": end\_date,  "hourly": ",".join(final\_params),  "timezone": timezone  }  try:  logger.info(f"Запрос к API: {url}")  logger.info(f"Параметры: {params}")  response = retry\_session.get(url, params=params, timeout=60)  response.raise\_for\_status()  data = response.json()  if data.get("error"):  logger.error(f"API вернул ошибку: {data['reason']}")  return None  if "hourly" not in data or not data["hourly"]:  logger.warning("Нет данных в ключе 'hourly'")  return None  df = pd.DataFrame(data["hourly"])  df["time"] = pd.to\_datetime(df["time"])  df.set\_index("time", inplace=True)  for col in df.columns:  df[col] = pd.to\_numeric(df[col], errors='coerce')  logger.info(f"Успешно загружено: {len(df)} записей, {len(df.columns)} переменных")  logger.info(f"Столбцы: {list(df.columns)}")  return df  except requests.exceptions.HTTPError as e:  logger.error(f"HTTP ошибка: {e}")  if hasattr(e, 'response') and e.response is not None:  logger.error(f"Response content: {e.response.text}")  return None  except requests.exceptions.Timeout:  logger.error("Таймаут запроса к API")  return None  except requests.exceptions.RequestException as e:  logger.error(f"Ошибка соединения: {e}")  return None  except Exception as e:  logger.error(f"Неизвестная ошибка: {e}")  return None  def create\_storm\_labels(  df: pd.DataFrame,  wind\_threshold: float = 15.0,  gust\_threshold: float = 25.0,  precip\_threshold: float = 7.0,  pressure\_drop\_window: int = 3,  pressure\_drop\_threshold: float = 4.0,  include\_weather\_code: bool = True  ) -> pd.DataFrame:  if df is None or df.empty:  return df  df = df.copy()  storm\_conditions = pd.Series(False, index=df.index)  # 1. Сильный ветер на разных высотах  wind\_columns = [col for col in df.columns if col.startswith('wind\_speed\_')]  for wind\_col in wind\_columns:  storm\_conditions |= (df[wind\_col] >= wind\_threshold)  # 2. Порывы ветра  if "wind\_gusts\_10m" in df.columns:  storm\_conditions |= (df["wind\_gusts\_10m"] >= gust\_threshold)  # 3. Сильные осадки  precip\_columns = ["precipitation", "rain", "showers"]  for precip\_col in precip\_columns:  if precip\_col in df.columns:  storm\_conditions |= (df[precip\_col] >= precip\_threshold)  # 4. Быстрое падение давления (штормовой признак)  pressure\_col = None  for col in ["pressure\_msl", "surface\_pressure"]:  if col in df.columns:  pressure\_col = col  break  if pressure\_col:  delta\_p = df[pressure\_col].diff(periods=pressure\_drop\_window)  rapid\_drop = delta\_p < -pressure\_drop\_threshold  storm\_conditions |= rapid\_drop.fillna(False)  # 5. Опасные погодные коды  if include\_weather\_code and "weather\_code" in df.columns:  storm\_codes = [65, 75, 82, 85, 86, 95, 96, 99]  storm\_conditions |= df["weather\_code"].isin(storm\_codes)  # 6. Высокая облачность (признак неустойчивости)  cloud\_columns = [col for col in df.columns if col.startswith('cloud\_cover')]  for cloud\_col in cloud\_columns:  if cloud\_col in df.columns:  storm\_conditions |= (df[cloud\_col] >= 80) # Облачность > 80%  df["is\_storm"] = storm\_conditions.astype(int)  storm\_count = df["is\_storm"].sum()  logger.info(f"Создано меток шторма: {storm\_count} из {len(df)} ({storm\_count / len(df) \* 100:.2f}%)")  return df  def save\_results(df: pd.DataFrame, filename: str, metadata: Optional[Dict] = None):  try:  os.makedirs('data', exist\_ok=True)  filepath = f"data/{filename}"  df.to\_csv(filepath, encoding='utf-8')  logger.info(f"Данные сохранены: {filepath}")  if metadata:  metapath = filepath.replace('.csv', '\_metadata.json')  with open(metapath, 'w', encoding='utf-8') as f:  json.dump(metadata, f, indent=2, ensure\_ascii=False)  logger.info(f"Метаданные сохранены: {metapath}")  except Exception as e:  logger.error(f"Ошибка при сохранении: {e}")  def main():  source\_url = (  "https://open-meteo.com/en/docs?hourly=temperature\_2m,wind\_speed\_10m,wind\_speed\_80m,"  "wind\_speed\_120m,wind\_speed\_180m,wind\_direction\_10m,wind\_direction\_80m,wind\_direction\_180m,"  "wind\_gusts\_10m,wind\_direction\_120m,precipitation,relative\_humidity\_2m,pressure\_msl,weather\_code,"  "snowfall,snow\_depth,cloud\_cover,cloud\_cover\_low,cloud\_cover\_mid,cloud\_cover\_high,rain,"  "showers,vapour\_pressure\_deficit,apparent\_temperature,dew\_point\_2m,et0\_fao\_evapotranspiration&timezone=Europe%2FMoscow&"  "latitude=55.7522&longitude=37.6156"  )  latitude, longitude, hourly\_params, timezone = parse\_parameters\_from\_url(source\_url)  if latitude is None or longitude is None:  latitude, longitude = 55.7522, 37.6156  logger.warning("Используются координаты по умолчанию: Москва")  if not timezone:  timezone = "Europe/Moscow"  logger.warning("Используется временная зона по умолчанию: Europe/Moscow")  start\_date = '2022-01-01'  end\_date = '2025-09-18'  logger.info("Запуск парсера погодных данных Open-Meteo")  logger.info(f"Координаты: {latitude}, {longitude} | Таймзона: {timezone}")  logger.info(f"Период: {start\_date} — {end\_date}")  logger.info(f"Запрашиваемые параметры: {hourly\_params}")  df = get\_historical\_weather(  latitude=latitude,  longitude=longitude,  start\_date=start\_date,  end\_date=end\_date,  hourly\_params=hourly\_params,  timezone=timezone  )  if df is None or df.empty:  logger.error("Не удалось получить данные или данные пустые.")  return  labeled\_df = create\_storm\_labels(  df,  wind\_threshold=15,  gust\_threshold=25,  precip\_threshold=7,  pressure\_drop\_window=3,  pressure\_drop\_threshold=4.0,  include\_weather\_code=True  )  save\_results(  labeled\_df,  "storm\_data.csv",  metadata={  "source\_url": source\_url,  "location": {"lat": latitude, "lon": longitude},  "timezone": timezone,  "date\_range": {"start": start\_date, "end": end\_date},  "hourly\_params": hourly\_params,  "storm\_criteria": {  "wind\_speed\_\* >= km/h": 15,  "wind\_gusts\_10m >= km/h": 25,  "precipitation/rain/showers >= mm/h": 7,  "pressure\_drop >= hPa/3h": 4.0,  "storm\_weather\_codes": [65, 75, 82, 85, 86, 95, 96, 99],  "cloud\_cover >= %": 80  },  "generated\_at": datetime.now().isoformat(),  "total\_rows": len(labeled\_df),  "storm\_events\_count": int(labeled\_df["is\_storm"].sum()),  "columns\_list": list(labeled\_df.columns)  }  )  logger.info(f"Общее количество записей: {len(labeled\_df)}")  logger.info(f"Количество штормовых событий: {labeled\_df['is\_storm'].sum()}")  logger.info(f"Доля штормовых событий: {labeled\_df['is\_storm'].mean() \* 100:.2f}%")  logger.info(f"Период данных: от {labeled\_df.index.min()} до {labeled\_df.index.max()}")  logger.info("Готово: данные успешно получены и сохранены")  if \_\_name\_\_ == "\_\_main\_\_":  main() |



**Рисунок 1 – Временной ряд**

Листинг 2 – для многомерных данных

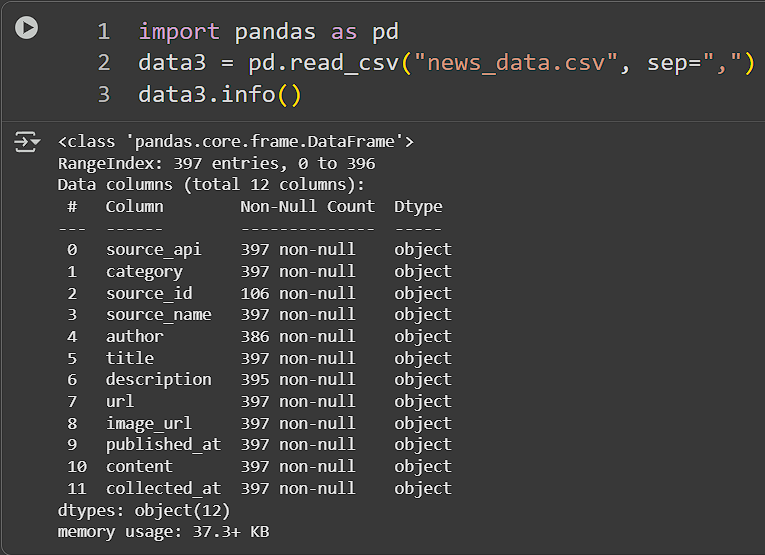
|  |
| --- |
| import requests  import time  import csv  from typing import List, Dict, Any  BASE\_URL = "https://pokeapi.co/api/v2"  def get\_json(url: str) -> Dict[str, Any]:  try:  resp = requests.get(url, timeout=10)  resp.raise\_for\_status()  return resp.json()  except requests.exceptions.RequestException as e:  print(f"Ошибка при запросе {url}: {e}")  return {}  def get\_pokemon\_list(limit: int = 100, offset: int = 0) -> List[Dict[str, Any]]:  url = f"{BASE\_URL}/pokemon?limit={limit}&offset={offset}"  data = get\_json(url)  return data.get("results", [])  def get\_encounter\_locations(pokemon\_url: str) -> str:  location\_url = pokemon\_url + "/encounters"  try:  locations = get\_json(location\_url)  if locations:  names = {  loc.get("location\_area", {}).get("name", "unknown")  for loc in locations if loc.get("location\_area")  }  return ", ".join(sorted(names)[:10])  return ""  except:  return ""  def get\_evolution\_chain(evolution\_chain\_url: str) -> str:  if not evolution\_chain\_url:  return ""  data = get\_json(evolution\_chain\_url)  if not data:  return ""  def parse\_chain(chain):  name = chain.get("species", {}).get("name", "")  evolves\_to = chain.get("evolves\_to", [])  if not evolves\_to:  return name  next\_names = [parse\_chain(evo) for evo in evolves\_to]  return " -> ".join([name] + next\_names)  return parse\_chain(data.get("chain", {}))  def get\_pokemon\_details(pokemon\_url: str) -> Dict[str, Any]:  data = get\_json(pokemon\_url)  if not data:  return {}  species\_url = data.get("species", {}).get("url")  species\_data = get\_json(species\_url) if species\_url else {}  stats = {}  for s in data.get("stats", []):  stat\_name = s["stat"]["name"]  stats[stat\_name] = s["base\_stat"]  types = [t["type"]["name"] for t in data.get("types", [])]  type\_1 = types[0] if len(types) > 0 else ""  type\_2 = types[1] if len(types) > 1 else ""  # Способности  abilities = []  hidden\_ability = ""  for ab in data.get("abilities", []):  if ab.get("is\_hidden"):  hidden\_ability = ab["ability"]["name"]  else:  abilities.append(ab["ability"]["name"])  ability\_1 = abilities[0] if len(abilities) > 0 else ""  ability\_2 = abilities[1] if len(abilities) > 1 else ""  moves = [m["move"]["name"] for m in data.get("moves", [])[:5]]  while len(moves) < 5:  moves.append("")  return {  "id": data.get("id"),  "name": data.get("name"),  "base\_experience": data.get("base\_experience"),  "height": data.get("height"),  "weight": data.get("weight"),  "order": data.get("order"),  # Типы  "type\_1": type\_1,  "type\_2": type\_2,  # Способности  "ability\_1": ability\_1,  "ability\_2": ability\_2,  "hidden\_ability": hidden\_ability,  "hp": stats.get("hp", 0),  "attack": stats.get("attack", 0),  "defense": stats.get("defense", 0),  "special\_attack": stats.get("special-attack", 0),  "special\_defense": stats.get("special-defense", 0),  "speed": stats.get("speed", 0),  # Движения (по столбцам)  "move\_1": moves[0],  "move\_2": moves[1],  "move\_3": moves[2],  "move\_4": moves[3],  "move\_5": moves[4],  # Спрайты  "sprite\_default": data.get("sprites", {}).get("front\_default"),  "sprite\_shiny": data.get("sprites", {}).get("front\_shiny"),  "sprite\_artwork": data.get("sprites", {}).get("other", {}).get("official-artwork", {}).get("front\_default"),  # Species  "color": species\_data.get("color", {}).get("name"),  "generation": species\_data.get("generation", {}).get("name"),  "habitat": species\_data.get("habitat", {}).get("name") if species\_data.get("habitat") else "",  "shape": species\_data.get("shape", {}).get("name"),  "is\_legendary": 1 if species\_data.get("is\_legendary", False) else 0,  "is\_mythical": 1 if species\_data.get("is\_mythical", False) else 0,  "capture\_rate": species\_data.get("capture\_rate"),  "base\_happiness": species\_data.get("base\_happiness"),  "growth\_rate": species\_data.get("growth\_rate", {}).get("name"),  "egg\_groups": ", ".join([eg["name"] for eg in species\_data.get("egg\_groups", [])]),  # Эволюция  "evolution\_chain": get\_evolution\_chain(species\_data.get("evolution\_chain", {}).get("url")),  # Локации  "encounter\_locations": get\_encounter\_locations(pokemon\_url),  }  def parse\_all\_pokemon(total\_limit: int = 100, pause\_sec: float = 0.1) -> List[Dict[str, Any]]:  all\_pokemon = []  offset = 0  batch\_size = 100  print(f"Начинаем сбор данных о {total\_limit} покемонах...")  while len(all\_pokemon) < total\_limit:  remaining = total\_limit - len(all\_pokemon)  current\_batch = min(batch\_size, remaining)  batch = get\_pokemon\_list(limit=current\_batch, offset=offset)  if not batch:  break  for pokemon in batch:  details = get\_pokemon\_details(pokemon["url"])  if details:  all\_pokemon.append(details)  print(f"{details['id']:>3} | {details['name']:<12} | {details['type\_1']}/{details['type\_2']}")  time.sleep(pause\_sec)  offset += len(batch)  print(f"Готово: {len(all\_pokemon)} покемонов обработано.")  return all\_pokemon  def save\_to\_csv(data: List[Dict[str, Any]], filename: str = "pokemon\_data.csv"):  if not data:  print("Нет данных для сохранения.")  return  fieldnames = [  "id", "name", "base\_experience", "height", "weight", "order",  "type\_1", "type\_2",  "hp", "attack", "defense", "special\_attack", "special\_defense", "speed",  "ability\_1", "ability\_2", "hidden\_ability",  "move\_1", "move\_2", "move\_3", "move\_4", "move\_5",  "sprite\_default", "sprite\_shiny", "sprite\_artwork",  "color", "generation", "habitat", "shape",  "is\_legendary", "is\_mythical", "capture\_rate", "base\_happiness",  "growth\_rate", "egg\_groups", "evolution\_chain", "encounter\_locations"  ]  with open(filename, "w", newline="", encoding="utf-8") as f:  writer = csv.DictWriter(f, fieldnames=fieldnames)  writer.writeheader()  writer.writerows(data)  print(f"Данные сохранены в '{filename}' ({len(data)} записей, {len(fieldnames)} столбцов)")  def main():  pokemon\_data = parse\_all\_pokemon(total\_limit=100, pause\_sec=0.1)  save\_to\_csv(pokemon\_data, "pokemon\_data.csv")  if \_\_name\_\_ == "\_\_main\_\_":  main() |



**Рисунок 2 – Многомерные данные**

Листинг 3 – для наборов текста

|  |
| --- |
| import requests  import pandas as pd  import time  from datetime import datetime, timedelta  NEWS\_API\_KEY = "98a99dffe46c409e81bc6398aff29096"  NEWS\_URL = "https://newsapi.org/v2/everything"  CATEGORIES = [  'sports', 'technology', 'health',  'business', 'science', 'politics',  'music', 'environment', 'entertainment',  'ai', 'cybersecurity',  'crypto', 'gaming', 'space', 'fashion',  'travel', 'food', 'books', 'wellness',  'renewables', 'edtech', 'robotics', 'philanthropy'  ]  DOMAINS = {  'sports': 'espn.com,bbc.com/sport',  'technology': 'techcrunch.com,engadget.com',  'health': 'who.int,webmd.com',  'business': 'reuters.com,bloomberg.com',  'science': 'sciencemag.org,nature.com',  'music': 'rollingstone.com, billboard.com, pitchfork.com, nme.com, spin.com',  'politics': 'reuters.com/politics, politico.com, theguardian.com/world',  'environment': 'ipcc.ch, grist.org, carbonbrief.org',  'entertainment': 'variety.com, hollywoodreporter.com',  'ai': 'syncedreview.com, arxiv.org, towardsdatascience.com',  'cybersecurity': 'krebsonsecurity.com, therecord.media, darkreading.com',  'crypto': 'coindesk.com, theblock.co, cointelegraph.com',  'gaming': 'ign.com, polygon.com, eurogamer.net',  'space': 'nasa.gov, spacex.com, skyandtelescope.org',  'fashion': 'vogue.com, wwd.com, businessoffashion.com',  'travel': 'cntraveler.com, lonelyplanet.com, skyradar.com, travelandleisure.com',  'food': 'eater.com, bonappetit.com, foodandwine.com, theinfatuation.com',  'books': 'nytimes.com/books, theguardian.com/books, lrb.co.uk, bookforum.com',  'wellness': 'goop.com, mindbodygreen.com, well.blogs.nytimes.com, tinyhearts.com',  'renewables': 'renewableenergyworld.com, greentechmedia.com, insideclimatenews.org',  'edtech': 'edutopia.org, edsurge.com, timeshighereducation.com/edtech',  'robotics': 'therobotreport.com, ieee.org/spectrum, robohub.org',  'philanthropy': 'ssir.org, philanthropy.com, globalgiving.org'  }  def fetch\_news\_by\_category(category, days=30, limit=50):  from\_date = (datetime.now() - timedelta(days=days)).strftime('%Y-%m-%d')  to\_date = datetime.now().strftime('%Y-%m-%d')  domain\_filter = DOMAINS.get(category, "")  params = {  'q': category,  'from': from\_date,  'to': to\_date,  'sortBy': 'publishedAt',  'language': 'en',  'pageSize': 100,  'page': 1,  'domains': domain\_filter,  'apiKey': NEWS\_API\_KEY  }  try:  response = requests.get(NEWS\_URL, params=params, timeout=15)  response.raise\_for\_status()  data = response.json()  articles = []  for item in data.get('articles', []):  if len(articles) >= limit:  break  articles.append({  'source\_api': 'newsapi.org',  'category': category,  'source\_id': item['source']['id'],  'source\_name': item['source']['name'],  'author': item.get('author'),  'title': item['title'],  'description': item.get('description'),  'url': item['url'],  'image\_url': item.get('urlToImage'),  'published\_at': item['publishedAt'],  'content': item.get('content'),  'collected\_at': datetime.now().isoformat()  })  print(f"Получено {len(articles)} новостей по теме '{category}'")  return articles  except Exception as e:  print(f"Ошибка при загрузке новостей ({category}): {e}")  return []  print("Сбор текстовых данных (новости)...")  all\_articles = []  for category in CATEGORIES:  articles = fetch\_news\_by\_category(category, days=30, limit=30)  all\_articles.extend(articles)  time.sleep(1.5)  if all\_articles:  df\_news = pd.DataFrame(all\_articles)  df\_news['published\_at'] = pd.to\_datetime(df\_news['published\_at'])  df\_news.sort\_values(by='published\_at', ascending=False, inplace=True)  df\_news.to\_csv('news\_data.csv', index=False)  print(f"Сохранено {len(df\_news)} новостных записей в textual\_news\_data.csv") |



**Рисунок 3 – Набор текста**

1. Сохранение данных в csv.

Результат работы:

Данную работу можете увидеть в блокноте Jupyter Notebook.

<https://drive.google.com/file/d/1o0rhZqLS5oI57PGN8sMn_clyT9e35E8Q/view?usp=sharing>

Вывод:

Научился парсить через API.

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