|  |
| --- |
| ч  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](#_Toc208863383)

[Введение 3](#_Toc208863384)

[Шаги выполнения 3](#_Toc208863385)

[Результат работы: 16](#_Toc208863386)

[Вывод: 17](#_Toc208863387)

[Список использованных источников и литературы: 17](#_Toc208863388)

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://airportgap.com/>

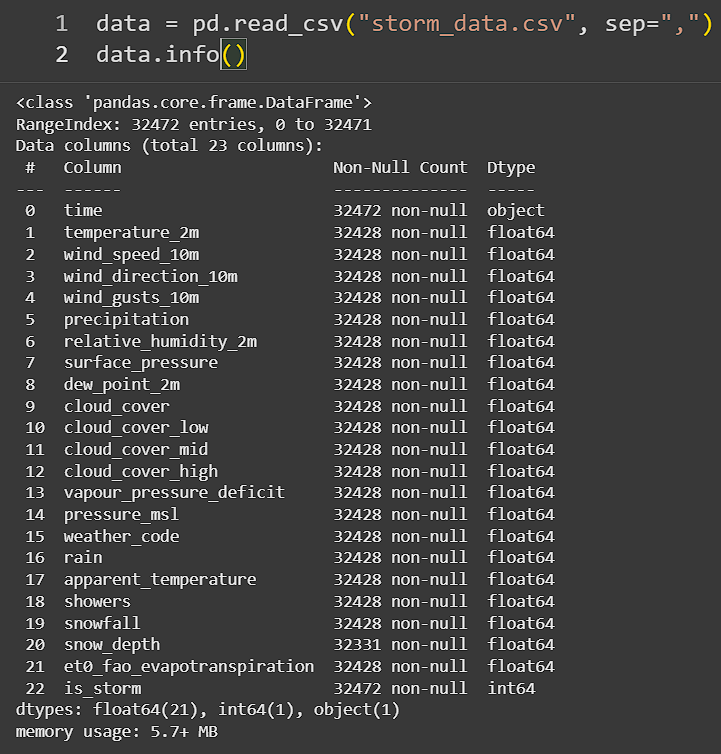
Наборы тексты:

<https://newsapi.org/>

1. Разработка скриптов для парсинга.

Листинг 1 – для временного ряда

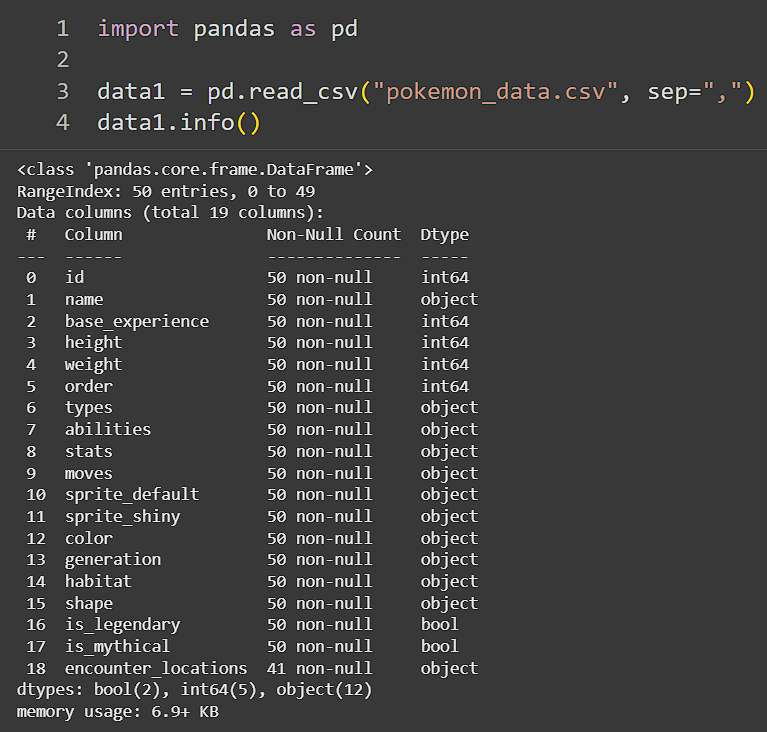
|  |
| --- |
| import requests  import pandas as pd  import numpy as np  import requests\_cache  from retry\_requests import retry  import logging  from typing import Optional, List, Dict  from datetime import datetime, timedelta  import json  import os  from urllib.parse import urlparse, parse\_qs  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\_hourly\_params\_from\_url(url: str) -> List[str]:  try:  parsed = urlparse(url)  query = parse\_qs(parsed.query)  hourly\_params = query.get('hourly', [])  if hourly\_params and isinstance(hourly\_params, list):  return hourly\_params[0].split(',')  return []  except Exception as e:  logger.error(f"Ошибка при парсинге URL: {e}")  return []  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"  default\_hourly = [  "temperature\_2m", "wind\_speed\_10m", "wind\_direction\_10m", "wind\_gusts\_10m",  "precipitation", "relative\_humidity\_2m", "surface\_pressure", "dew\_point\_2m",  "cloud\_cover", "cloud\_cover\_low", "cloud\_cover\_mid", "cloud\_cover\_high",  "vapour\_pressure\_deficit", "pressure\_msl", "weather\_code", "rain",  "apparent\_temperature", "showers", "snowfall", "snow\_depth",  "et0\_fao\_evapotranspiration"  ]  params = {  "latitude": latitude,  "longitude": longitude,  "start\_date": start\_date,  "end\_date": end\_date,  "hourly": ",".join(hourly\_params or default\_hourly),  "timezone": timezone  }  try:  logger.info(f"Запрос к API: {url} | {start\_date} – {end\_date}")  response = retry\_session.get(url, params=params, timeout=45)  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")  return df  except requests.exceptions.HTTPError as e:  if response.status\_code == 400:  logger.error(f"Ошибка 400 (неверные параметры): {response.text}")  else:  logger.error(f"HTTP ошибка: {e}")  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 (если есть wind\_speed\_10m)  if "wind\_speed\_10m" in df.columns:  storm\_conditions |= (df["wind\_speed\_10m"] >= wind\_threshold)  # 2. Порывы ветра  if "wind\_gusts\_10m" in df.columns:  storm\_conditions |= (df["wind\_gusts\_10m"] >= gust\_threshold)  # 3. Сильные осадки  if "precipitation" in df.columns:  storm\_conditions |= (df["precipitation"] >= precip\_threshold)  # 4. Быстрое падение давления (штормовой признак)  if "pressure\_msl" in df.columns:  delta\_p = df["pressure\_msl"].diff(periods=pressure\_drop\_window)  rapid\_drop = delta\_p < -pressure\_drop\_threshold  storm\_conditions |= rapid\_drop.fillna(False)  # 5. Опасные погодные коды (грозы, ливни, град, метель)  if include\_weather\_code and "weather\_code" in df.columns:  storm\_codes = [65, 75, 82, 85, 86, 95, 96, 99]  # 65: сильный дождь, 75: сильный снег, 82: сильный ливень, 95-99: грозы с градом  storm\_conditions |= df["weather\_code"].isin(storm\_codes)  df["is\_storm"] = storm\_conditions.astype(int)  logger.info(f"Создано меток шторма: {df['is\_storm'].sum()} из {len(df)}")  return df  def save\_results(df: pd.DataFrame, filename: str, metadata: Optional[Dict] = None):  try:  filepath = f"{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\_direction\_10m,wind\_gusts\_10m,"  "precipitation,relative\_humidity\_2m,surface\_pressure,dew\_point\_2m,"  "cloud\_cover,cloud\_cover\_low,cloud\_cover\_mid,cloud\_cover\_high,"  "vapour\_pressure\_deficit,pressure\_msl,weather\_code,rain,apparent\_temperature,"  "showers,snowfall,snow\_depth,et0\_fao\_evapotranspiration"  "&timezone=Europe%2FMoscow&latitude=55.7522&longitude=37.6156"  )  hourly\_params = parse\_hourly\_params\_from\_url(source\_url)  if not hourly\_params:  logger.warning("Не удалось распарсить параметры из URL, используем стандартные")  parsed = parse\_qs(urlparse(source\_url).query)  try:  latitude = float(parsed['latitude'][0])  longitude = float(parsed['longitude'][0])  timezone = parsed.get('timezone', ['Europe/Moscow'])[0]  except KeyError as e:  logger.error(f"Отсутствует обязательный параметр: {e}")  return  except Exception as e:  logger.error(f"Ошибка при чтении параметров: {e}")  return  # end\_date = datetime.now().strftime('%Y-%m-%d')  # start\_date = (datetime.now() - timedelta(days=90)).strftime('%Y-%m-%d')  start\_date = '2022-01-01'  end\_date = '2025-09-14'  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:  logger.error("Не удалось получить данные.")  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\_10m >= km/h": 15,  "wind\_gusts\_10m >= km/h": 25,  "precipitation >= mm/h": 7,  "pressure\_drop >= hPa/3h": 4.0,  "storm\_weather\_codes": [65, 75, 82, 85, 86, 95, 96, 99]  },  "generated\_at": datetime.now().isoformat(),  "total\_rows": len(labeled\_df),  "storm\_events\_count": int(labeled\_df["is\_storm"].sum())  }  )  logger.info("Готово: данные успешно получены и сохранены")  if \_\_name\_\_ == "\_\_main\_\_":  main() |



**Рисунок 1 – Временной ряд**

Листинг 2 – для многомерных данных

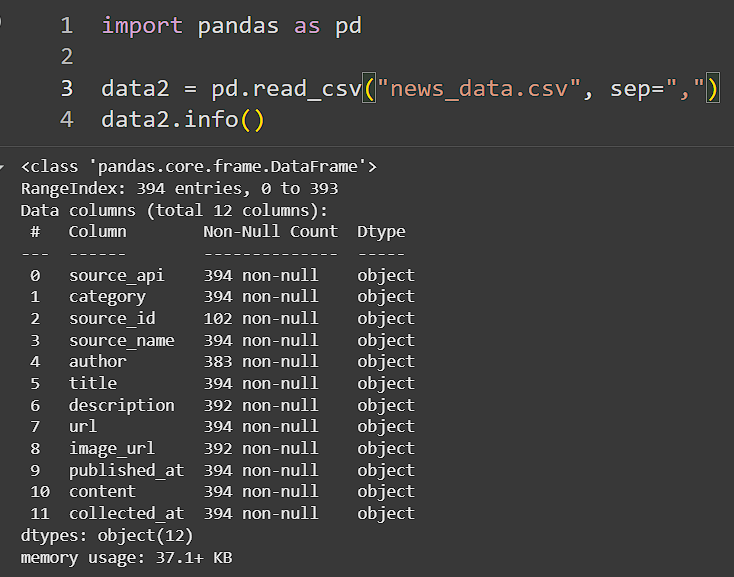
|  |
| --- |
| import requests  import time  import csv  from typing import List, Dict, Any  BASE\_URL = "https://pokeapi.co/api/v2"  def get\_json(url: str) -> Dict[str, Any]:  resp = requests.get(url)  resp.raise\_for\_status()  return resp.json()  def get\_pokemon\_list(limit: int = 50, offset: int = 0) -> List[Dict[str, Any]]:  url = f"{BASE\_URL}/pokemon?limit={limit}&offset={offset}"  data = get\_json(url)  return data.get("results", [])  def get\_pokemon\_details(pokemon\_url: str) -> Dict[str, Any]:  data = get\_json(pokemon\_url)  species\_url = data.get("species", {}).get("url")  species\_data = get\_json(species\_url) if species\_url else {}  location\_url = pokemon\_url + "encounters"  try:  locations = get\_json(location\_url)  encounter\_locations = ", ".join(set(  loc["location\_area"]["name"] for loc in locations  ))  except:  encounter\_locations = ""  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"),  "types": ", ".join([t["type"]["name"] for t in data.get("types", [])]),  "abilities": ", ".join([ab["ability"]["name"] for ab in data.get("abilities", [])]),  "stats": ", ".join([f"{st['stat']['name']}:{st['base\_stat']}" for st in data.get("stats", [])]),  "moves": ", ".join([m["move"]["name"] for m in data.get("moves", [])[:10]]),  "sprite\_default": data.get("sprites", {}).get("front\_default"),  "sprite\_shiny": data.get("sprites", {}).get("front\_shiny"),  "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 None,  "shape": species\_data.get("shape", {}).get("name") if species\_data.get("shape") else None,  "is\_legendary": species\_data.get("is\_legendary"),  "is\_mythical": species\_data.get("is\_mythical"),  "encounter\_locations": encounter\_locations  }  def parse\_all\_pokemon(limit: int = 50, pause\_sec: float = 0.3) -> List[Dict[str, Any]]:  all\_pokemon = []  offset = 0  while len(all\_pokemon) < limit:  batch = get\_pokemon\_list(limit=min(20, limit - len(all\_pokemon)), offset=offset)  if not batch:  break  for p in batch:  details = get\_pokemon\_details(p["url"])  all\_pokemon.append(details)  print(f"Fetched {details['name']}")  time.sleep(pause\_sec)  offset += len(batch)  return all\_pokemon  def save\_to\_csv(data: List[Dict[str, Any]], filename: str = "pokemon\_data.csv"):  if not data:  return  with open(filename, "w", newline="", encoding="utf-8") as f:  writer = csv.DictWriter(f, fieldnames=data[0].keys())  writer.writeheader()  writer.writerows(data)  print(f"Данные сохранены в {filename}")  def main():  pokemon\_data = parse\_all\_pokemon(limit=2000, pause\_sec=0.3)  save\_to\_csv(pokemon\_data)  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