Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Отчет по лабораторной работе №2**

По дисциплине: «Новые технологии в РПС»

Тема: «Обработка данных»

|  |
| --- |
| Выполнил:  студент группы  М.ПИН.РИС - 23.08  Тимофеев Александр Сергеевич |
| Проверила:  старший преподаватель  кафедры ПО  Корнеева Е.И. |

Тверь 2023

**Вариант №1-2**

**Постановка задачи:**

* Рефакторинг лабораторной 1
* Работа с pandas и csv(описание датасета)

0. Ваш csv файл должен содержать в первой колонке дату в формате ISO 8601 (без времени).

1. Написать скрипт, который разобъёт исходный csv файл на файл X.csv и Y.csv, с одинаковым количеством строк. Первый будет содержать даты, второй - данные.

2. Написать скрипт, который разобъёт исходный csv файл на N файлов, где каждый отдельный файл будет соответствовать одному году. Файлы называются по первой и последней дате, которую они содержат. (если файл содержит данные с первого января 2001 по 31 декабря 2001, то файл назвать 20010101\_20011231.csv)

3. Написать скрипт, который разобъёт исходный csv файл на N файлов, где каждый отдельный файл будет соответствовать одной неделе. Файлы называются по первой и последней дате, которую они содержат.

4. Написать скрипт, содержащий функцию, принимающую на вход дату (тип datetime) и возвращающий данные для этой даты (из файла) или None если данных для этой даты нет. Функция должна быть представлена в четырёх версиях в зависимости от типа входных файлов, из которых будут прочитаны данные (пункты 0–3). Написать функцию next(), которая будет при первом вызове возвращать данные для самой ранней возможной даты (возвращается кортеж (дата, данные)), а при каждом следующем вызове данные для следующей по порядку даты. Если попадается дата, для которой данные отсутствуют, то она игнорируется и возвращаются данные для следующей валидной даты.

5. Написать на основе предыдущего пункта классы итераторы (пример ниже).

**Код программы (Python):**

Библиотеки:

import os

import requests

import re

import csv

import pandas as pd

import xml.etree.ElementTree as ET

from bs4 import BeautifulSoup

from datetime import datetime

from google.colab import drive

Привинчивание GoogleDrive к Colab:

drive.mount('/content/drive')

Константы:  
CURR\_DIR = '/content/drive/MyDrive/lab2\_ntrps'

#в каждом из датасетов присутствует поле date, остальные поля различаются

IMAGES\_FIELDS = ['date', 'file\_name', 'url']

CURRENCY\_FIELDS = ['date', 'nominal', 'value', 'vunitRate']

TEXT\_FIELDS = ['date', ]

Проверка наличия репозиториев:

def check\_repository(dir: str, name: str) -> None:

    dataset\_directory = os.path.join(dir, name)

    if not os.path.exists(dataset\_directory):

        os.makedirs(dataset\_directory)

    return dataset\_directory

Запись csv файла:

def write\_csv\_file(path: str, data: list) -> None:

    mode = 'w' if not os.path.exists(path) else 'a'

    with open(path, mode, newline='') as csv\_file:

      csv\_writer = csv.writer(csv\_file)

      csv\_writer.writerow(data)

Валюты

Получения списка всех валют:

def get\_ids\_of\_currency() -> None:

    url = 'http://www.cbr.ru/scripts/XML\_daily.asp'

    response = requests.get(url, headers={'User-Agent':'Mozilla/5.0'})

    if(response.status\_code == 200):

        xml\_page = ET.fromstring(response.content)

        data = []

        for tag in xml\_page.findall('Valute'):

            id = tag.get('ID')

            num\_code = tag.find('NumCode').text

            char\_code = tag.find('CharCode').text

            currency\_name = tag.find('Name').text

            data.append([id, num\_code, char\_code, currency\_name])

        with open(CURR\_DIR + f'/datasets/currency/ids\_currency.csv', 'w', newline='', encoding='utf-8') as csv\_file:

            csv\_writer = csv.writer(csv\_file)

            csv\_writer.writerow(['id', 'num\_code', 'char\_code', 'currency\_name'])

            for row in data:

                csv\_writer.writerow(row)

Получение ID валюты по её буквенному обозначению:

def get\_currency\_id(file\_path: str, char\_code: str) -> str:

    df = pd.read\_csv(file\_path)

    currency\_id = df[df['char\_code'] == char\_code]['id'].values[0]

    return currency\_id

Создание датасета валюты:  
def write\_currency\_dataset(name\_currency:str, start\_date: str, end\_date: str) -> None:

    id\_currency = get\_currency\_id(CURR\_DIR + '/datasets/currency/ids\_currency.csv', name\_currency)

    if id\_currency:

        url = f"https://www.cbr.ru/scripts/XML\_dynamic.asp?date\_req1={start\_date}&date\_req2={end\_date}&VAL\_NM\_RQ={id\_currency}"

        response = requests.get(url, headers={'User-Agent':'Mozilla/5.0'})

        #в отличии от json надо проверить всего лишь раз что страница существует, на ней и так будут все курсы валюты

        if(response.status\_code == 200):

            xml\_page = ET.fromstring(response.content)

            data = []

            for tag in xml\_page.findall('Record'):

                date = datetime.strptime(tag.get('Date'), '%d.%m.%Y').strftime('%Y-%m-%d')

                nominal = tag.find('Nominal').text

                value = tag.find('Value').text.replace(',', '.')

                vunit\_Rate = tag.find('VunitRate').text.replace(',', '.')

                data.append([date, nominal, value, vunit\_Rate])

            start\_date = datetime.strptime(start\_date, '%d/%m/%Y').strftime('%Y%m%d')

            end\_date= datetime.strptime(end\_date, '%d/%m/%Y').strftime('%Y%m%d')

            with open(CURR\_DIR + f'/datasets/currency/{name\_currency}\_{start\_date}\_{end\_date}.csv', 'w', newline='', encoding='utf-8') as csv\_file:

                csv\_writer = csv.writer(csv\_file)

                csv\_writer.writerow(['date', 'nominal', 'value', 'vunitRate'])

                for row in data:

                    csv\_writer.writerow(row)

        else:

            print('Ошибка: Дата указана неверно!')

    else:

        print('Ошибка: Код валюты не найден!')

Картинки

Парсер ссылки на полную картинку:

def parser\_url(url: str) -> str:

    pattern = r'img\_url=([^&]+)&text='

    match = re.search(pattern, url)

    if match:

        img\_url\_encoded = match.group(1)

        img\_url\_decoded = img\_url\_encoded.replace('%2F', '/').replace('%3A', ':')

        return img\_url\_decoded

    else:

        print('Ошибка: Ссылка после img\_url не найдена в URL')

Рассчитываем необходимое количество картинок по их количеству:

def calc\_pages(num\_images: int) -> int:

    return num\_images // 30 + (num\_images % 30 > 0) if num\_images > 30 else 1

Получаем HTML теги для картинок (полных или миниатюр):

def get\_html\_tags(mini\_images: bool) -> tuple[str, str, str]:

    if mini\_images:

        return 'img', 'serp-item\_\_thumb', 'src'

    else:

        return 'a', 'serp-item\_\_link', 'href'

Функция скачивания картинок:

def download\_image(url: str, save\_path: str) -> bool:

    try:

        response = requests.get(url, headers={'User-Agent':'Mozilla/5.0'}, stream=True)

        if(response.status\_code == 200):

            with open(save\_path, 'wb') as file:

                for chunk in response.iter\_content(1024):

                    file.write(chunk)

            return True

        else:

            print(f'Ошибка: Не удалось загрузить изображение: {url}')

            return False

    except Exception as e:

        print(f'Ошибка при загрузке изображения: {url}')

        return False

def download\_images(query: str, num\_images: int, mini\_images: bool = False) -> None:

    pages = calc\_pages(num\_images)

    class\_folder = check\_repository(CURR\_DIR, f'datasets/images/{query}')

    downloaded\_count = 0

    base\_url = 'https:'

    csv\_file\_path = CURR\_DIR + '/datasets/images/' + f'{query}\_dataset.csv'

    write\_csv\_file(csv\_file\_path, IMAGES\_FIELDS)

    tag, tag\_class, tag\_source = get\_html\_tags(mini\_images)

    #а вот это чтобы без движков было, грузим странички

    for page in range(0, pages):

        search\_url = f'https://yandex.ru/images/search?text={query}&p={page}'

        #сделал с with для автоматического закрытия соединения

        with requests.get(search\_url, headers={'User-Agent':'Mozilla/5.0'}) as response:

            soup = BeautifulSoup(response.text, 'html.parser')

            for a in soup.find\_all(tag, class\_=tag\_class):

                img\_url = a[tag\_source]

                # получаем полный URL изображения

                if mini\_images and not img\_url.startswith('http'):

                    img\_url = base\_url + img\_url

                elif img\_url.startswith('/images'):

                    img\_url = parser\_url(img\_url)

                #csv\_image\_filename = image\_filename

                image\_filename = f'{downloaded\_count:04d}.jpg'

                image\_path = os.path.join(class\_folder, image\_filename)

                if(download\_image(img\_url, image\_path)):

                    downloaded\_count += 1

                    print(f'Загружено изображений для {query}: {downloaded\_count}/{num\_images}')

                    write\_csv\_file(csv\_file\_path, [datetime.now().strftime('%Y-%m-%d'), image\_filename, img\_url])

                if(downloaded\_count >= num\_images):

                    break

Слияние двух дата сетов X.csv (дата) и Y.csv (данные):

def merge\_data\_with\_date(file\_csv\_x: str = 'X.csv', file\_csv\_y: str = 'Y.csv') -> pd.DataFrame:

    if not os.path.exists(file\_csv\_x) or not os.path.exists(file\_csv\_y):

        return 'Ошибка: файла не существуют'

    df\_x = pd.read\_csv(file\_csv\_x)

    df\_y = pd.read\_csv(file\_csv\_y)

    if 'date' not in df\_x.columns or df\_x.shape[1] != 1:

        return 'Ошибка: Файл X.csv не содержит поле date'

    merged\_df = pd.merge(df\_x, df\_y, on='date', how='inner')

    return merged\_df

Функция проверки датасета на необходимые поля:

def check\_dataset(df: pd.DataFrame, required\_fields: list) -> bool:

    for field in required\_fields:

        if field not in df.columns:

            return False

    return True

Создание датасета из файлов:

def create\_dataset\_from\_files(files: list, fields: list) -> pd.DataFrame:

    df = pd.DataFrame()

    for file in files:

        data = pd.read\_csv(file)

        if check\_dataset(data, fields):

            data['date'] = pd.to\_datetime(data['date'])

            df = df.append(data, ignore\_index=True)

        else:

            print(f'Ошибка: Файл {file} не содержит необходимых полей')

        #FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

        #df = df.append(data, ignore\_index=True) --> менять на df = df.append(data)

        #df = pd.concat(df\_list, ignore\_index=True) добавлять в данную строчку (просто раскомменитить)

    return df

Перезапись даты для картинок:

def rewrite\_dates(df: pd.DataFrame, start\_date: datetime) -> pd.DataFrame:

    df['date'] = [start\_date + pd.DateOffset(days=i) for i in range(len(df))]

    return df

Сохранения датасета в .csv:

def save\_new\_dataset(df: pd.DataFrame, file\_to\_save: str, index\_custom: bool = False) -> None:

    df.to\_csv(file\_to\_save, index=index\_custom)

Вывод всех .csv файлов в директории:

def print\_csv\_files\_in\_dir(dirs: list) -> list:

    csv\_files = []

    for directory in dirs:

        for root, dirs, files in os.walk(directory):

            for file in files:

                if file.endswith('.csv'):

                    csv\_files.append(os.path.basename(file))

    return csv\_files

Вывод всех .csv файлов в виде дерева:

def print\_csv\_dir\_tree(dir: str, file\_extension: str = '.csv', tab: str = '') -> None:

    print(tab + os.path.basename(dir) + '/')

    tab += '    '

    for path in sorted(os.listdir(dir)):

        full\_path = os.path.join(dir, path)

        if os.path.isfile(full\_path) and full\_path.endswith(file\_extension):

            print(tab + os.path.basename(full\_path))

        elif os.path.isdir(full\_path):

            print\_csv\_dir\_tree(full\_path, file\_extension, tab)

Разделение даты от данных:

def separation\_date\_by\_data(df: pd.DataFrame) -> None:

    df\_date = df['date']

    df\_data = df.drop('date', axis=1)

    df\_date.to\_csv(CURR\_DIR + '/csv/csv\_date\_by\_data/X.csv', index=False)

    df\_data.to\_csv(CURR\_DIR + '/csv/csv\_date\_by\_data/Y.csv', index=False)

Разделение данных по годам:

def separation\_by\_years(df: pd.DataFrame) -> None:

    df['date'] = pd.to\_datetime(df['date'])

    for year, group in df.groupby(df['date'].dt.year):

        start\_date = group['date'].min().strftime('%Y%m%d')

        end\_date = group['date'].max().strftime('%Y%m%d')

        filename = f'{start\_date}\_{end\_date}.csv'

        group.to\_csv(CURR\_DIR + '/csv/csv\_years/' + filename, index=False)

Разделение данных по неделям:

def separation\_by\_weeks(df: pd.DataFrame) -> None:

    df['date'] = pd.to\_datetime(df['date'])

    #вылазиет предупреждение вот как советует использовать данный цикл сам пандас (из-за устаревшей версии)

    #FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated. Please use Series.dt.isocalendar().week instead.

    #for (year, week), group in df.groupby([df['date'].dt.isocalendar().year, df['date'].dt.isocalendar().week]):

    for (year, week), group in df.groupby([df['date'].dt.year, df['date'].dt.week]):

        start\_date = group['date'].min().strftime('%Y%m%d')

        end\_date = group['date'].max().strftime('%Y%m%d')

        filename = f'{start\_date}\_{end\_date}.csv'

        group.to\_csv(CURR\_DIR + '/csv/csv\_weeks/' + filename, index=False)

Получение данных по дате:

def get\_data\_from\_date(df: pd.DataFrame, date: datetime) -> None | pd.DataFrame:

    data = df[df['date'] == date]

    if data.empty:

        return None

    else:

        return data.drop(columns=['date'])

Функция next() (получения данных из датасета):

def next(df: pd.DataFrame, index: int) -> None | tuple[str]:

    if index < len(df):

        #return tuple(df.loc[index, ['date', 'file\_name', 'url']]) #возвращает строки по меткам

        return tuple(df.iloc[index]) #возвращает строки по целочисленным значениям

    return None

Итератор класса данных:

class DataIterator:

    def \_\_init\_\_(self, df: pd.DataFrame):

        self.df = df

        self.counter = 0

    def \_\_iter\_\_(self):

        return self

    def \_\_next\_\_(self):

        if self.counter < len(self.df):

            result = tuple(self.df.iloc[self.counter])

            self.counter += 1

            return result

        else:

            raise StopIteration

Тестирование:

Картинки

def test\_images():

    download\_images('grey bear', 10, False)

    download\_images('blue bear', 1, True)

    df = create\_dataset\_from\_files([CURR\_DIR + '/datasets/images/black bear\_dataset.csv'], IMAGES\_FIELDS)

    print(df)

    rewrite\_dates(df, datetime(2023, 1, 1))

    print(df)

    df = create\_dataset\_from\_files([CURR\_DIR + '/datasets/images/test\_weeks\_dataset.csv'], IMAGES\_FIELDS) #CURR\_DIR + '/csv/brown bear\_dataset.csv'

    separation\_date\_by\_data(df)

    separation\_by\_years(df)

    separation\_by\_weeks(df)

    print('---Получение данных по определенной дате---')

    print(get\_data\_from\_date(df, datetime(2023, 1, 24)))

    print('---Работа next()---')

    for index in range(0, len(df)):

        print(next(df, index))

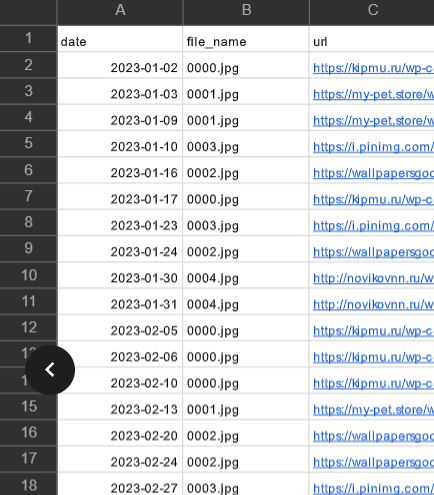
    print('---Работа итератора---')

    iterator = DataIterator(df)

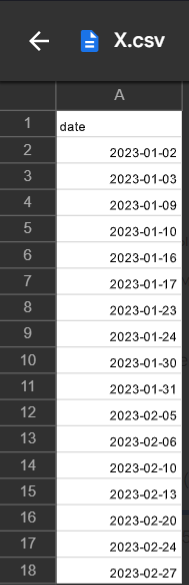
    for item in iterator:

        print(item)

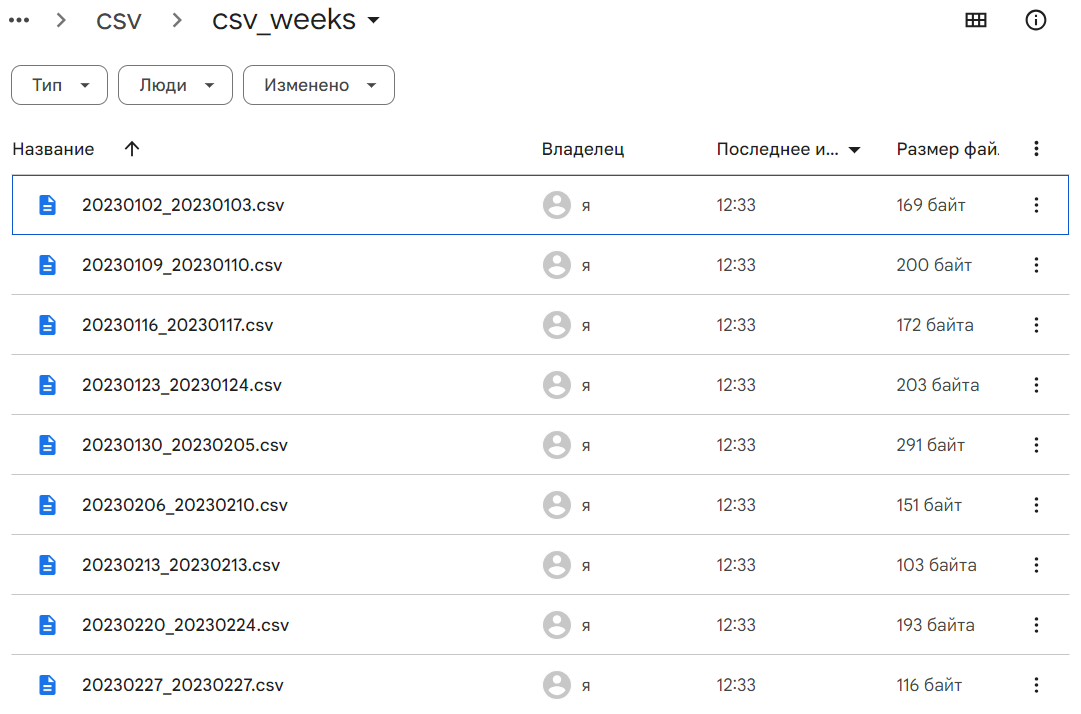
Тестируемый датасет:



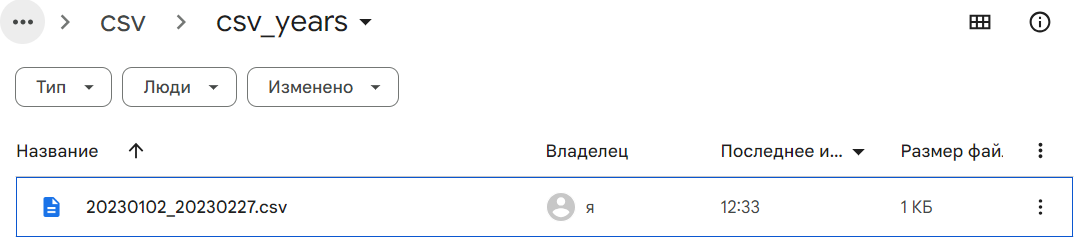
Разделение даты от данных:



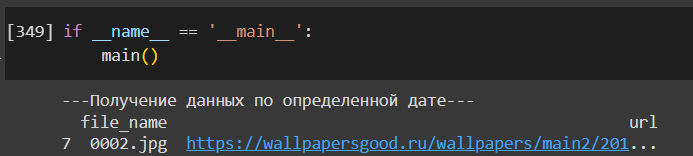


Разделение по неделям:  


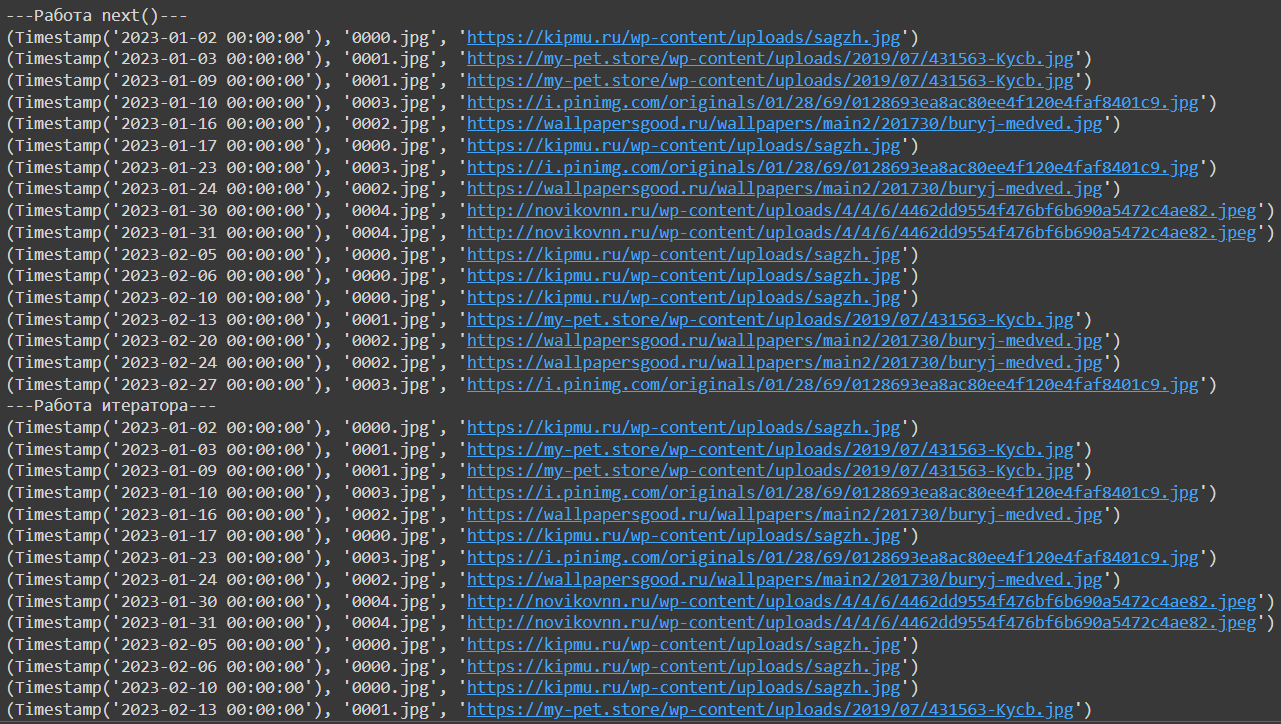
Разделение по годам:



Получение данных по дате:



Работа функции next() и итератора:



Валюты

def test\_currency():

    get\_ids\_of\_currency()

    write\_currency\_dataset('EUR', '01/01/1991', '31/12/2023') #USD

    df = create\_dataset\_from\_files([CURR\_DIR + '/datasets/currency/EUR\_19910101\_20231231.csv'], CURRENCY\_FIELDS)

    separation\_date\_by\_data(df)

    separation\_by\_years(df)

    separation\_by\_weeks(df)

    print('---Получение данных по определенной дате---')

    print(get\_data\_from\_date(df, datetime(2023, 1, 24)))

    print('---Работа next()---')

    for index in range(0, len(df)):

        print(next(df, index))

    print('---Работа итератора---')

    iterator = DataIterator(df)

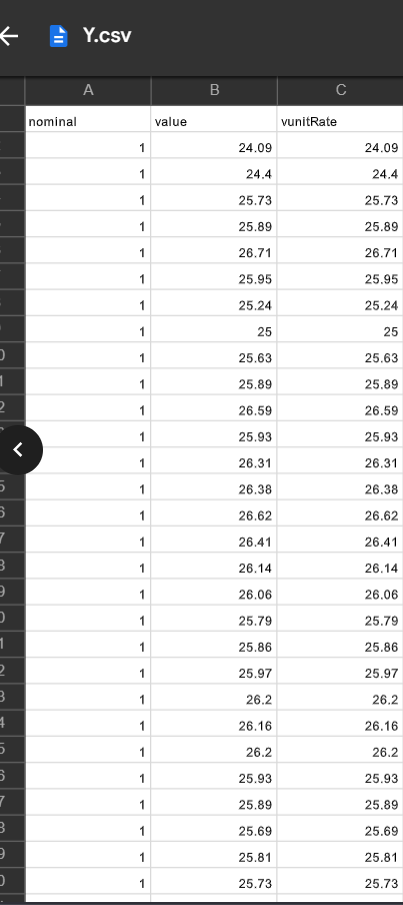
    for item in iterator:

        print(item)

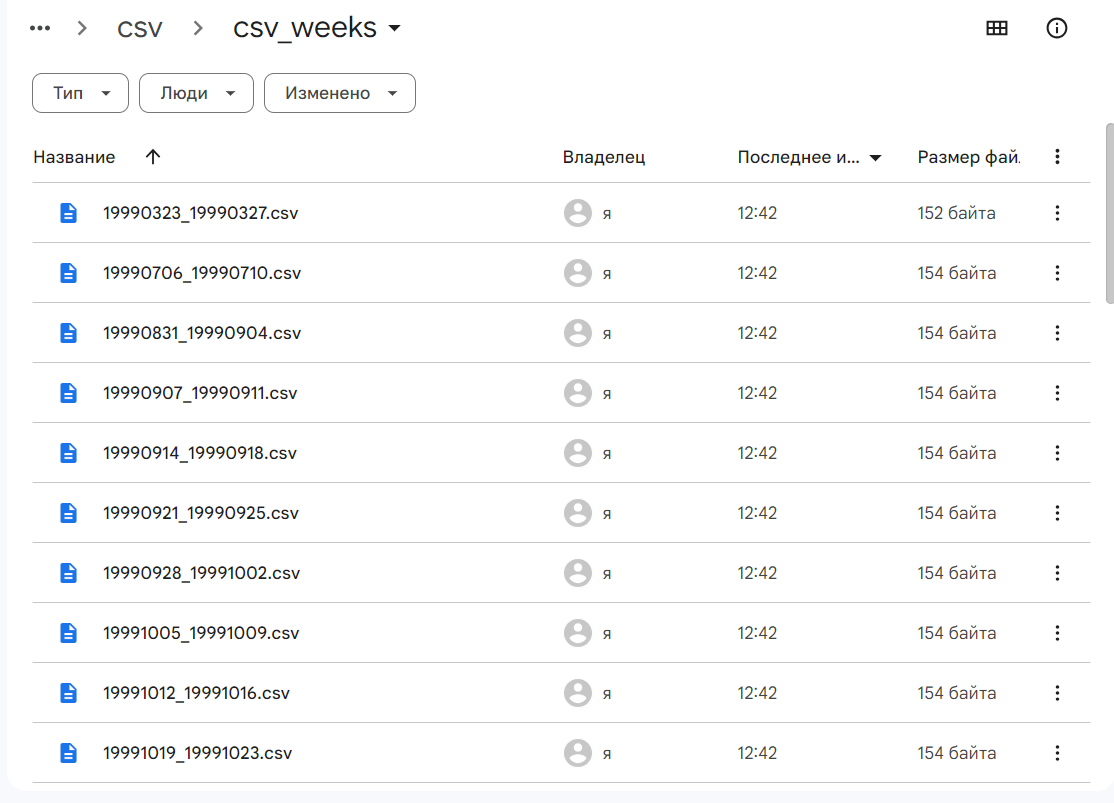
Тестируемый датасет:

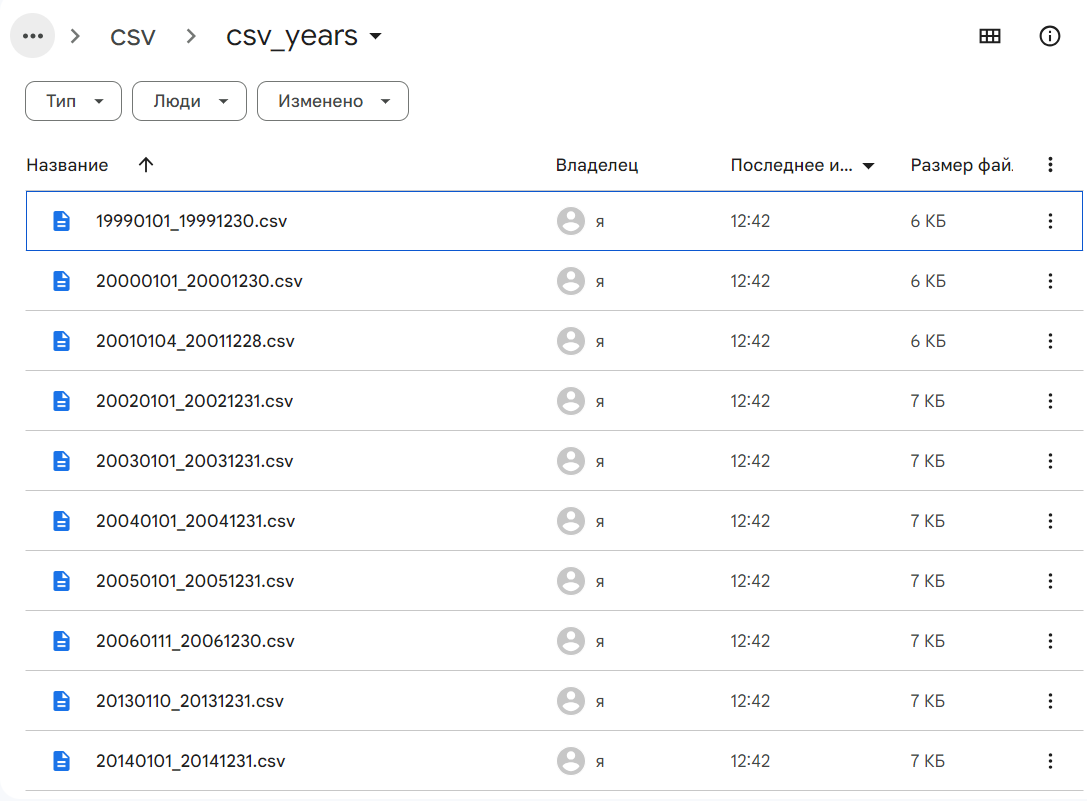


Разделение по неделям:

Разделение по неделям:



Разделение по годам:

Получение данных по дате:

