**Отчет по выполнению тестового задания**

Выполнил: Витомсков Алексей

***Задание.***

Получить данные с сайта https://novosibirsk.n1.ru

- Нужно выбрать 10 многоэтажных домов и выгрузить из них данные о продажах квартир. А именно:

• Адрес объявления

• Общая площадь квартиры

• Этаж

• Год сдачи дома

• Цена

• Материал дома

• Широта и долгота здания в котором размещено объявление

- Автоматизировать выгрузку (например, раз в день)

Получим 10 домов и все объявления о продажах в них. Получив данные за два дня по одним и тем же домам нужно посчитать изменение средней цены кв.м. в доме.

Поднять БД на PostgreSQL(или любой другой) и загрузить туда выгруженные данные за два дня.

Итогом выполнения ТЗ будет отправленная нам структура БД и примеры любых выполненных запросов на Ваше усмотрение.

***Решение.***

Задача реализована на языке Python с использование технологий flask, bs4, sqlite.

Демон apscheduler запускает функцию парсера каждые 30 минут. Парсеру передается список из 10 ссылок на страницы с домами по конкретному адресу, содержащие не более 100 объявлений. Парсе достает ссылку на каждое объявление, проходит по ней и получает данные из соответствующих тегов. Для получения широты и долготы здания ищется соответствующий скрипт и парсится в Json.

Парсер передает список списков со всеми собранными данными по конкретному дому в функцию store, которая заносит все поля в одну таблицу БД sqlite.

Пример запроса:

|  |
| --- |
| query = '''insert into mytable (area, year, floor, \  material, address, price, latitude, longtitude) values (?, ?, ?, ?, ?, ?, ?, ?)''' |

Структура таблицы

id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

area INTEGER NOT NULL,

year VARCHAR(10),

floor VARCHAR(10),

material VARCHAR(10),

address VARCHAR(100) NOT NULL,

price INTEGER NOT NULL,

latitude VARCHAR(10),

longtitude VARCHAR(10),

datetime DATETIME DEFAULT CURRENT\_TIMESTAMP NOT NULL

Клиент может в любой момент отправить Post запрос на сервер для вывода информации. Получение данных из БД на сервере проходит в 2 этапа. Получаем из БД список адресов.

Пример запроса:

|  |
| --- |
| query = 'SELECT DISTINCT address FROM mytable ORDER BY datetime LIMIT ?' |

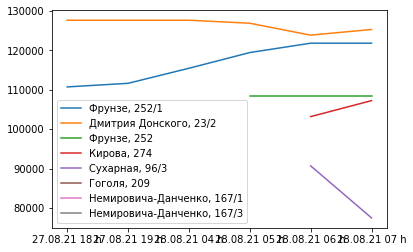
Затем для каждого адреса получаем все данные по площади и стоимости жилья.

Пример запроса:

|  |
| --- |
| query = 'SELECT area, price, datetime FROM mytable WHERE address = ? LIMIT ?' |

Создается датафрейм для данного адреса. Даты приводятся к нужному виду (в данном случае с точность до часа). Датафрейм нарезается по дате. Для каждой даты рассчитывается средняя цена за метр.

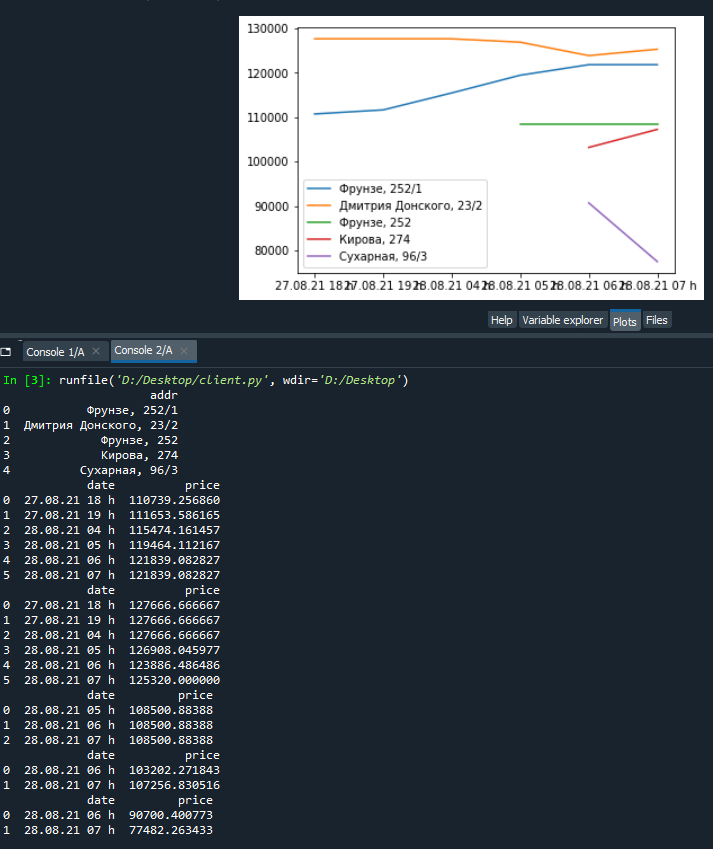
Клиент получает json в виде списка адресов и словаря дата:средняя цена. Клиент строит график в matplotlib с динамикой изменения цены по датам.



Пример данных заносящихся в БД

|  |
| --- |
| ['45.8', '2021 г.', '6 из 18', 'кирпич', 'Фрунзе, 252/1', '5400000', 55.04206906910559, 82.9663423809721]  ['62.85', '2021 г.', '12 из 18', 'кирпич - монолит', 'Фрунзе, 252/1', '6430000', 55.04206906910559, 82.9663423809721]  ['55', '2021 г.', '15 из 18', 'кирпич - монолит', 'Фрунзе, 252/1', '5990000', 55.04206906910559, 82.9663423809721]  ['60.8', '2021 г.', '2 из 10', 'кирпич - монолит', 'Фрунзе, 252/1', '6499999', 55.04206906910559, 82.9663423809721]  ['33.5', '2021 г.', '4 из 14', 'кирпич - монолит', 'Фрунзе, 252/1', '4600000', 55.04206906910559, 82.9663423809721]  ['28.57', '2021 г.', '1 из 10', 'кирпич - монолит', 'Фрунзе, 252/1', '3900000', 55.04206906910559, 82.9663423809721]  ['58', '2021 г.', '7 из 10', 'кирпич - монолит', 'Фрунзе, 252/1', '6350000', 55.04206906910559, 82.9663423809721]  ['42', '2021 г.', '10 из 10', 'кирпич - монолит', 'Фрунзе, 252/1', '5200000', 55.04206906910559, 82.9663423809721]  ['94.73', '2021 г.', '4 из 10', 'кирпич', 'Фрунзе, 252', '9600000', 55.04206906910559, 82.9663423809721]  ['95', '2021 г.', '5 из 10', 'кирпич', 'Фрунзе, 252', '9790000', 55.04206906910559, 82.9663423809721]  ['69.95', '2021 г.', '2 из 18', 'кирпич - монолит', 'Фрунзе, 252/1', '8150000', 55.04206906910559, 82.9663423809721]  ['42.9', '2021 г.', '6 из 14', 'кирпич', 'Фрунзе, 252/1', '6500000', 55.04206906910559, 82.9663423809721]  ['70.94', '2021 г.', '1 из 10', 'кирпич - монолит', 'Фрунзе, 252', '7850000', 55.04206906910559, 82.9663423809721]  ['62.85', '2021 г.', '2 из 18', 'кирпич - монолит', 'Фрунзе, 252', '7350000', 55.04206906910559, 82.9663423809721]  ['42', '2021 г.', '12 из 16', 'кирпич - монолит', 'Фрунзе, 252/1', '4950000', 55.04206906910559, 82.9663423809721]  ['34', '2021 г.', '1 из 14', 'кирпич', 'Фрунзе, 252/1', '4450000', 55.04206906910559, 82.9663423809721]  ['41.2', '2021 г.', '1 из 10', 'кирпич', 'Фрунзе, 252/1', '9500000', 55.04206906910559, 82.9663423809721]  ['94.73', '2021 г.', '10 из 10', 'кирпич - монолит', 'Фрунзе, 252/1', '9950000', 55.04206906910559, 82.9663423809721]  ['42.63', '2021 г.', '2 из 19', 'кирпич - монолит', 'Фрунзе, 252/1', '5200000', 55.04206906910559, 82.9663423809721]  ['65', '2021 г.', '6 из 18', 'кирпич', 'Фрунзе, 252/1', '6800000', 55.04206906910559, 82.9663423809721]  ['43', '2021 г.', '10 из 14', 'кирпич - монолит', 'Фрунзе, 252/1', '5299000', 55.04206906910559, 82.9663423809721]  ['63', '2021 г.', '2 из 10', 'кирпич - монолит', 'Фрунзе, 252/1', '6299000', 55.04206906910559, 82.9663423809721]  ['94.73', '2021 г.', '5 из 10', 'кирпич - монолит', 'Фрунзе, 252', '9790000', 55.04206906910559, 82.9663423809721]  ['69.95', '2021 г.', '5 из 18', 'кирпич - монолит', 'Фрунзе, 252', '7500000', 55.04206906910559, 82.9663423809721]  ['62.85', '2021 г.', '3 из 16', 'кирпич - монолит', 'Фрунзе, 252', '7350000', 55.04206906910559, 82.9663423809721]  True  100  ['70.91', None, '10 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '8863750', 55.06461450470407, 82.90435274603257]  ['52.49', None, '21 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '6718720', 55.06461450470407, 82.90435274603257]  ['48.95', None, '13 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '6363500', 55.06461450470407, 82.90435274603257]  ['52.49', None, '18 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '6718720', 55.06461450470407, 82.90435274603257]  ['95.38', None, '15 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '10682560', 55.06461450470407, 82.90435274603257]  ['96.07', None, '4 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '10567700', 55.06461450470407, 82.90435274603257]  ['65.75', None, '1 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '8547500', 55.06461450470407, 82.90435274603257]  ['96.07', None, '13 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '10759840', 55.06461450470407, 82.90435274603257]  ['48.47', None, '18 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '6301100', 55.06461450470407, 82.90435274603257]  ['95.38', None, '15 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '10682560', 55.06461450470407, 82.90435274603257]  ['65.56', None, '2 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '8522800', 55.06461450470407, 82.90435274603257]  ['95.67', None, '3 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '10523700', 55.06461450470407, 82.90435274603257]  ['52.49', None, '16 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '6718720', 55.06461450470407, 82.90435274603257]  ['47.21', None, '22 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '6137300', 55.06461450470407, 82.90435274603257]  ['65.23', None, '5 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '8479900', 55.06461450470407, 82.90435274603257]  ['70.46', None, '21 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '9018880', 55.06461450470407, 82.90435274603257]  ['95.79', None, '22 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '10728480', 55.06461450470407, 82.90435274603257]  ['52.49', None, '22 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '6718720', 55.06461450470407, 82.90435274603257]  ['95.79', None, '20 из 25', 'кирпич - монолит', 'Дмитрия Донского, 23/2', '10728480', 55.06461450470407, 82.90435274603257] |

Вывод на стороне клиента:



Код сервера

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """  Created on Thu Aug 26 19:11:11 2021  @author: aleks1212  """  from flask import Flask, request  from apscheduler.schedulers.background import BackgroundScheduler  import re  import pandas as pd  import os  from bs4 import BeautifulSoup  import requests  import json  import sqlite3  from datetime import datetime  currdir = os.getcwd()  dbid = os.path.join(currdir, r"log.db")  def parser(house):  main\_href = 'https://novosibirsk.n1.ru/'  param\_list = []  req = requests.get(house)  print(req.ok)    soup = BeautifulSoup(req.text, "html.parser")  data = soup.html.body  #print(data)  try:  links = []  main\_link = data.findAll("div", {"class": "card-title living-list-card\_\_inner-block"})  for link in main\_link:  href = re.search(r'href="\S+"', str(link))  href = href.group()[6:-1]  links.append(main\_href + href)  print(len(links))    for link in links:  req = requests.get(link)  soup = BeautifulSoup(req.text, "html.parser")  soup\_ = soup.html.body  data = soup\_.find("div", {"class": "card-living-content\_\_params"})    key\_values = data.findAll("li", {"class":"card-living-content-params-list\_\_item"})  keys = []  values = []  for key\_val in key\_values:  key = key\_val.find("span", {"class":"card-living-content-params-list\_\_name"}).text  value = key\_val.find("span", {"class":"card-living-content-params-list\_\_value"}).text  value = value.replace('\xa0м2', '')  value = value.replace(',', '.')  keys.append(key)  values.append(value)    param = ['Общая площадь', 'Год постройки', 'Этаж', 'Материал дома']  param\_val = [] #значения    for p in param:  if p in keys:  param\_val.append(values[keys.index(p)])  else:  param\_val.append(None)    address = data.find("span", {"class":"ui-kit-link\_\_inner"}).text  address = address.replace(' стр.', '')  param\_val.append(address)    data\_price = soup\_.find("span", {"class": "price"}).text  data\_price = re.sub('[^\d\.]', '', data\_price)  param\_val.append(data\_price)    scripts = soup\_.findAll('script')  script = None  for s in scripts:  if len(re.findall('\_\_INITIAL\_STATE\_\_', s.get\_text())) > 0:  script = s  break  if script != None:  script = script.get\_text()  script = re.sub(r';var pageMeta =.+','', script)  num = script.find(r'{"contexts"')  script = script[num:]  script = script.replace('undefined','false')  script = re.sub(r'new Date\(".+"\)', 'false', script)  dataform = script #str(script).strip("'<>() ").replace('\'', '\"')  json\_s = json.loads(dataform)  latitude = json\_s['\_\_INITIAL\_STATE\_\_']['OfferCard']['Location']['offerLocation']['latitude']  longtitude = json\_s['\_\_INITIAL\_STATE\_\_']['OfferCard']['Location']['offerLocation']['longtitude']  param\_val.append(latitude)  param\_val.append(longtitude)  print(param\_val)  param\_list.append(param\_val)  except Exception as e:  print('Error occured: ', e)  return(param\_list)  #каждый день запуск  def run\_parser():    houses = ['https://novosibirsk.n1.ru/search/?rubric=flats&deal\_type=sell&limit=100&addresses%5B0%5D%5Bstreet\_id%5D=865516&addresses%5B0%5D%5Bhouse\_number%5D=252%2F1',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=864462&addresses%5B0%5D%5Bhouse\_number%5D=23%2F2%20%D1%81%D1%82%D1%80.&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=864657&addresses%5B0%5D%5Bhouse\_number%5D=274%20%D1%81%D1%82%D1%80.&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=865373&addresses%5B0%5D%5Bhouse\_number%5D=96%2F3&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=864333&addresses%5B0%5D%5Bhouse\_number%5D=209&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=864940&addresses%5B0%5D%5Bhouse\_number%5D=167%2F1&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=864974&addresses%5B0%5D%5Bhouse\_number%5D=82&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=864137&addresses%5B0%5D%5Bhouse\_number%5D=32%2F1%20%D1%81%D1%82%D1%80.&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=864061&addresses%5B0%5D%5Bhouse\_number%5D=60%20%D1%81%D1%82%D1%80.&deal\_type=sell&rubric=flats&limit=100',  'https://novosibirsk.n1.ru/search/?addresses%5B0%5D%5Bstreet\_id%5D=865274&addresses%5B0%5D%5Bhouse\_number%5D=77&deal\_type=sell&rubric=flats&limit=100',  ]  n\_houses = len(houses)  for i in range(n\_houses):  param\_list = parser(houses[i])  store\_message(param\_list, dbid)  return  sched = BackgroundScheduler(daemon=True)  sched.add\_job(run\_parser,'interval',minutes=10)  sched.start()  app = Flask(\_\_name\_\_)  def create\_table\_if\_not\_exists(conn, cursor):  #cursor.execute("""DROP TABLE IF EXISTS mytable""")  #conn.commit()  cursor.execute("""CREATE TABLE IF NOT EXISTS mytable(  id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  area INTEGER NOT NULL,  year VARCHAR(10),  floor VARCHAR(10),  material VARCHAR(10),  address VARCHAR(100) NOT NULL,  price INTEGER NOT NULL,  latitude VARCHAR(10),  longtitude VARCHAR(10),  datetime DATETIME DEFAULT CURRENT\_TIMESTAMP NOT NULL)""")  conn.commit()  return  def store\_message(param\_list, dbid):  try:  #подключение базы данных  conn = sqlite3.connect(dbid) # или :memory: чтобы сохранить в RAM  with conn:  cursor = conn.cursor()  create\_table\_if\_not\_exists(conn, cursor)  query = '''insert into mytable (area, year, floor, \  material, address, price, latitude, longtitude) values (?, ?, ?, ?, ?, ?, ?, ?)'''  for param in param\_list:  conn.execute(query, param)  conn.commit()  except sqlite3.IntegrityError as e:  print('Error occured: ', e)  finally:  if conn:  conn.close()  return True  def get\_message(limit, dbid, columns = None):  #подключение базы данных  try:  conn = sqlite3.connect(dbid) # или :memory: чтобы сохранить в RAM  with conn:  cursor = conn.cursor()  create\_table\_if\_not\_exists(conn, cursor)  if columns == None:  query = 'SELECT DISTINCT address FROM mytable ORDER BY datetime LIMIT ?'  param\_list = [limit]  else:  query = 'SELECT area, price, datetime FROM mytable WHERE address = ? LIMIT ?'  param\_list = [columns, limit]  cursor.execute(query, param\_list)  conn.commit()  rows = cursor.fetchall()  except Exception:  print('Ошибка обработки')  finally:  if conn:  conn.close()  return rows  @app.route('/', methods=['POST'])  def mean\_plot():  addr\_ = get\_message(-1, dbid)  addr = []  for a in addr\_:  temp = a[0].replace(' стр.', '')  if temp not in addr:  addr.append(temp)    addr\_list = []  price\_list = []    for a in addr:  print(a)  mean\_price\_list = get\_message(-1, dbid, a)  df = pd.DataFrame(mean\_price\_list, columns = ['area', 'price', 'datetime'])  df['price'] /= df['area']  df['datetime'] = pd.to\_datetime(df['datetime'])  day = []  for i in range(df.shape[0]):  day.append(datetime.strftime(df.datetime[i], '%d.%m.%y %H h'))  df.datetime = day  df = df.sort\_values("datetime", ascending=False)  df = df.groupby(by = ['datetime']).mean()  addr\_list.append(a)  price\_list.append(df['price'].to\_dict())  return json.dumps({'addr':addr\_list, 'price':price\_list}).encode('utf-8')  if \_\_name\_\_ == "\_\_main\_\_":  app.run(host='0.0.0.0', port=5000) |

Код клиента

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """  Created on Fri Aug 27 21:36:03 2021  @author: aleks1212  """  import requests  import json  import pandas as pd  import matplotlib.pyplot as plt  body = None  keys= None  r = requests.post("http://localhost:5000/", params = keys, data = body)  response = json.loads(r.content.decode('utf-8'))  addrframe = pd.DataFrame(response['addr'], columns = ['addr'])  print(addrframe)  for i in range( addrframe.shape[0]):  priceframe = pd.DataFrame(response['price'][i].items(), columns = ['date', 'price'])  print(priceframe)  plt.plot(priceframe['date'], priceframe['price'], label = addrframe.loc[i][0])  plt.legend()  plt.show() |