Департамент образования города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»

Институт цифрового образования
Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

«Проектный практикум по разработке ETL-решений »

Лабораторная работа № 4.1

«Работа с XML и JSON файлами»

Выполнила:

Студентка группы АДЭУ-211

Кравцова Алёна Евгеньевна

Руководитель:

Босенко Т.М

Москва

2025

Задание:

4.1.1. Настроить среду и рабочий каталог;

4.1.2. Загрузить данные;

4.1.3. Проверить качество данных (например, отсутствующие значения и выбросы);

4.1.4. Удалить столбцы (множество пропусков в значениях, бесполезные столбцы для анализа);

Работа будет выполняться в Google Colab. Сперва необходимо настроить рабочую среду и установить dask (Рис. 1). Dask — это библиотека для параллельных и распределённых вычислений в Python. Она позволяет обрабатывать очень большие объёмы данных, которые не помещаются в оперативную память, с помощью привычного синтаксиса pandas, NumPy и scikit-learn.

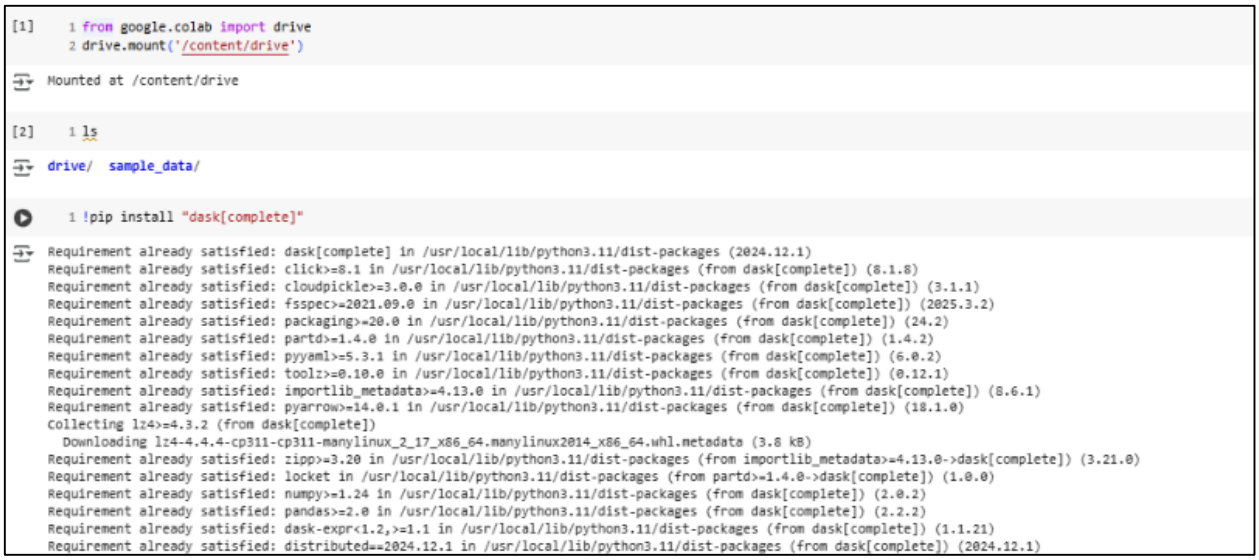



Рис. 1 – Установка dask

Также загрузим необходимые библиотеки для работы (Рис. 2).

```
[4]
# import libraries
import sys
import os

## import dask libraries
import dask.dataframe as dd
from dask.diagnostics import ProgressBar

# import libraries
import pandas as pd
```

```
[5]
cwd = os.getcwd()

# print
print('', sys.executable)
print('', cwd)
```

```
/usr/bin/python3
/content
```

Рис. 2 – Установка библиотек

Далее загрузим датасет (Рис. 3).


Рис. 3 – Загрузка данных

Далее проведём обработку данных и посмотрим отсутствующие значения (Рис. 4). Полученный вывод содержит описание задачи, которую будет делать Dask.

```
[8]
# count missing values
missing_values = df.isnull().sum()
missing_values
```

```
Dask Series Structure:
npartitions=1
MedianStudentsPerTeacher    int64
zpid                          ...
Dask Name: sum, 5 expressions
Expr=(~ NotNull(frame=ArrowStringConversion(frame=FromMapProjectable(3102fa8)))).sum()
```

Рис. 4 – Получение пустых значений

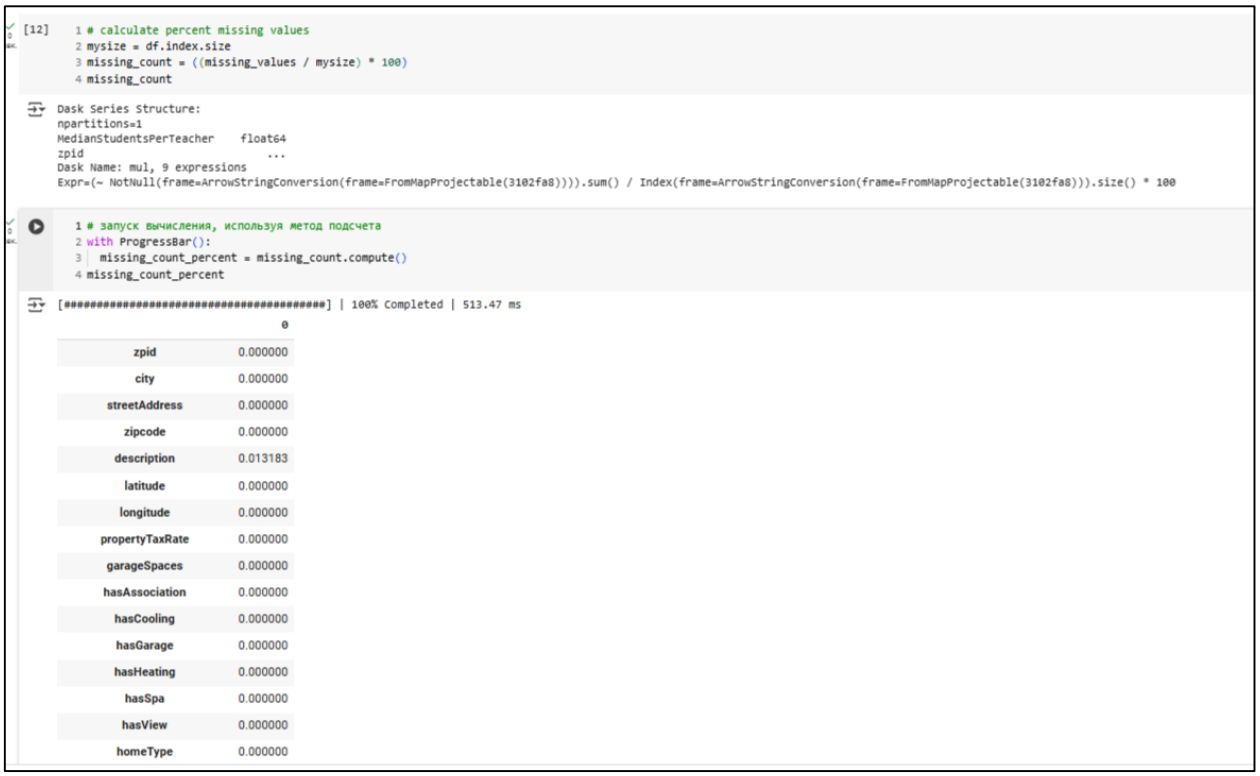
С помощью команды .compute выводим количество пустых строк по каждому столбцу (Рис. 5).

```
1 df.isnull().sum().compute()
```

| | 0 |
|---|---|
| zpid | 0 |
| city | 0 |
| streetAddress | 0 |
| zipcode | 0 |
| description | 2 |
| latitude | 0 |
| longitude | 0 |
| propertyTaxRate | 0 |
| garageSpaces | 0 |
| hasAssociation | 0 |
| hasCooling | 0 |
| hasGarage | 0 |
| hasHeating | 0 |
| hasSpa | 0 |
| hasView | 0 |
| homeType | 0 |
| parkingSpaces | 0 |
| yearBuilt | 0 |
| latestPrice | 0 |
| numPriceChanges | 0 |

Рис. 5 - .compute()

Можно посчитать процентное содержание null’ов  и также с помощью *.compute()* вывести сведения по каждому столбцу (Рис. 6).



```
[12]  1 # calculate percent missing values
      2 mysize = df.index.size
      3 missing_count = ((missing_values / mysize) * 100)
      4 missing_count
```

```
Dask Series Structure:
npartitions=1
MedianStudentsPerTeacher    float64
zpid                            ...
Dask Name: mul, 9 expressions
Expr=(~ NotNull(frame=ArrowStringConversion(frame=FromMapProjectable(3102fa8)))).sum() / Index(frame=ArrowStringConversion(frame=FromMapProjectable(3102fa8))).size() * 100
```

```
1 # запуск вычисления, используя метод подсчета
2 with ProgressBar():
3   missing_count_percent = missing_count.compute()
4 missing_count_percent
```

```
[########################################] | 100% Completed | 513.47 ms
```

| | 0 |
|---|---|
| zpid | 0.000000 |
| city | 0.000000 |
| streetAddress | 0.000000 |
| zipcode | 0.000000 |
| description | 0.013183 |
| latitude | 0.000000 |
| longitude | 0.000000 |
| propertyTaxRate | 0.000000 |
| garageSpaces | 0.000000 |
| hasAssociation | 0.000000 |
| hasCooling | 0.000000 |
| hasGarage | 0.000000 |
| hasHeating | 0.000000 |
| hasSpa | 0.000000 |
| hasView | 0.000000 |
| homeType | 0.000000 |

Рис. 6 – Процентное содержание пустых строк

Итак, данные не содержат пустых строк, соответственно можно не очищать их от пустых значений.

Столбец «homeImage» содержит ссылку на изображение, что не понесет никакой смысловой нагрузки для анализа, поэтому удалим этот столбец (Рис. 7).

2.1.4. **Удалить столбцы (пропуски в значениях, бесполезные столбцы для анализа).**

```
# операция фильтрации разреженных столбцов (более 60% пропущенных значений) и сохраняем оставшиеся
columns_to_drop = missing_count_percent[missing_count_percent > 60].index
print(columns_to_drop)

# удаление ненужных столбцов
with ProgressBar():
    #df_dropped = df.drop(columns_to_drop, axis=1).persist()
    df_dropped = df.drop('homeImage', axis=1).compute()
```

```
Index([], dtype='object')
[########################################] | 100% Completed | 443.10 ms
```

```
[17]  df_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15171 entries, 0 to 15170
Data columns (total 46 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   zpid                        15171 non-null  int64
 1   city                        15171 non-null  string
 2   streetAddress               15171 non-null  string
 3   zipcode                     15171 non-null  int64
 4   description                 15169 non-null  string
 5   latitude                    15171 non-null  float64
 6   longitude                   15171 non-null  float64
 7   propertyTaxRate             15171 non-null  float64
 8   garageSpaces                15171 non-null  int64
 9   hasAssociation              15171 non-null  bool
 10  hasCooling                  15171 non-null  bool
 11  hasGarage                   15171 non-null  bool
 12  hasHeating                  15171 non-null  bool
 13  hasSpa                      15171 non-null  bool
 14  hasView                     15171 non-null  bool
 15  homeType                    15171 non-null  string
 16  parkingSpaces               15171 non-null  int64
```

Рис. 7 – Удаление «ненужного» столбца

Практическая работа 4.2. Визуализация ориентированных ациклических графов (DAG)

Задание:

4.2.1. Визуализировать DAG с одним узлом и зависимостями;

4.2.1. Визуализировать DAG с более чем одним узлом и зависимостями.

DASK использует библиотеку Graviz для создания визуального представления групп DAG, созданных планировщиком. Для упрощения будем использовать объект Dask Delayed вместо DataFrames, поскольку они становятся довольно большими и их трудно визуализировать. Поэтому необходимо его установить (Рис. 8).

```
[18]  # import library
      import dask.delayed as delayed
```

Рис. 8 – Установка Dask Delayed

Визуализируем DAG с одним узлом и зависимостями (). Круги указывают на функцию и вычисления, а квадраты — промежуточный или конечный результат. Входящие стрелки представляют зависимости. Функция increment не имеет никаких зависимостей, а функция add — две. Таким образом, функция add должна ждать, пока не будут вычислены объекты x и y.

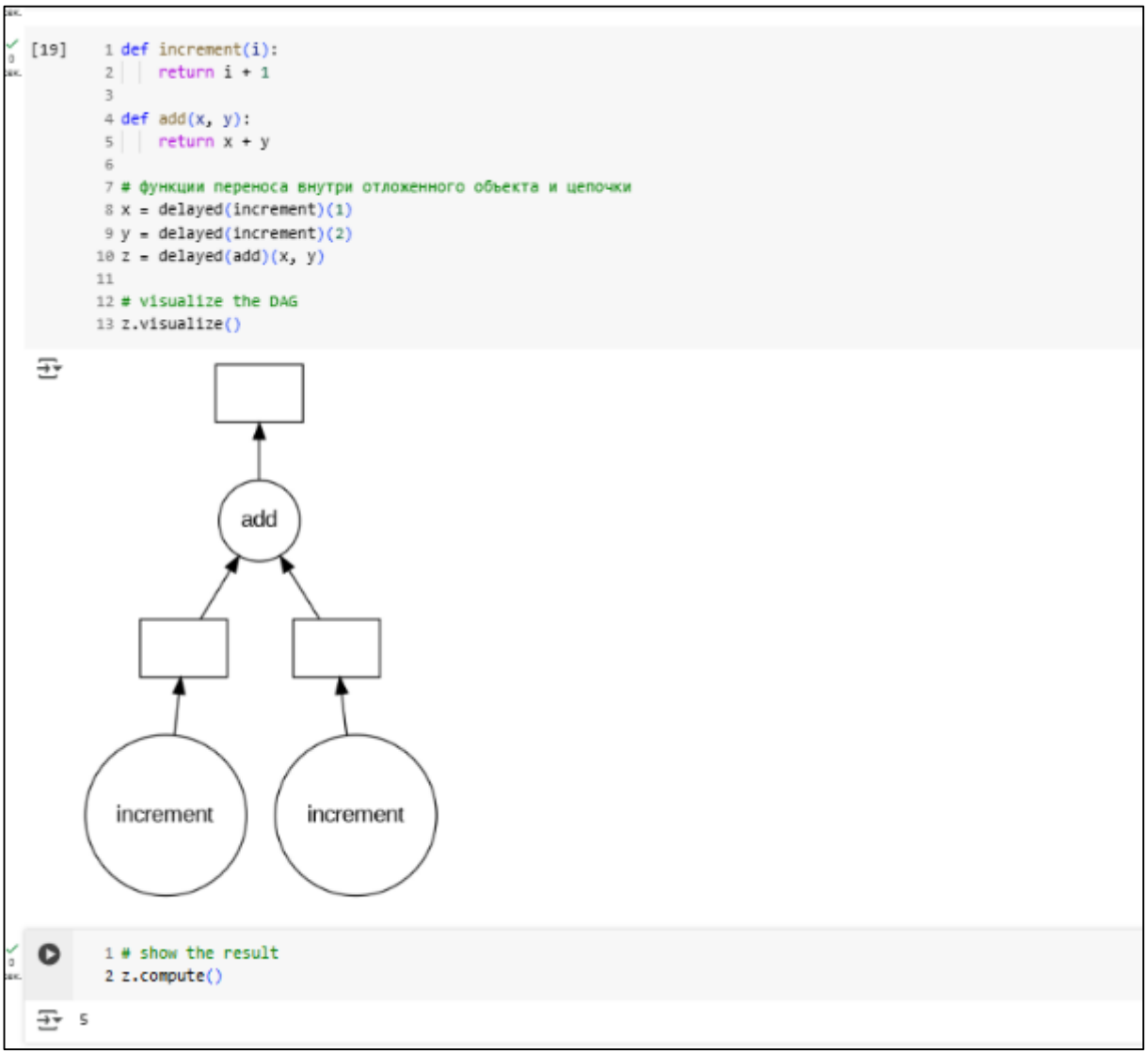



Рис. 9 – Визуализация DAG

Далее визуализируем DAG с несколькими узлами (Рис. 9).

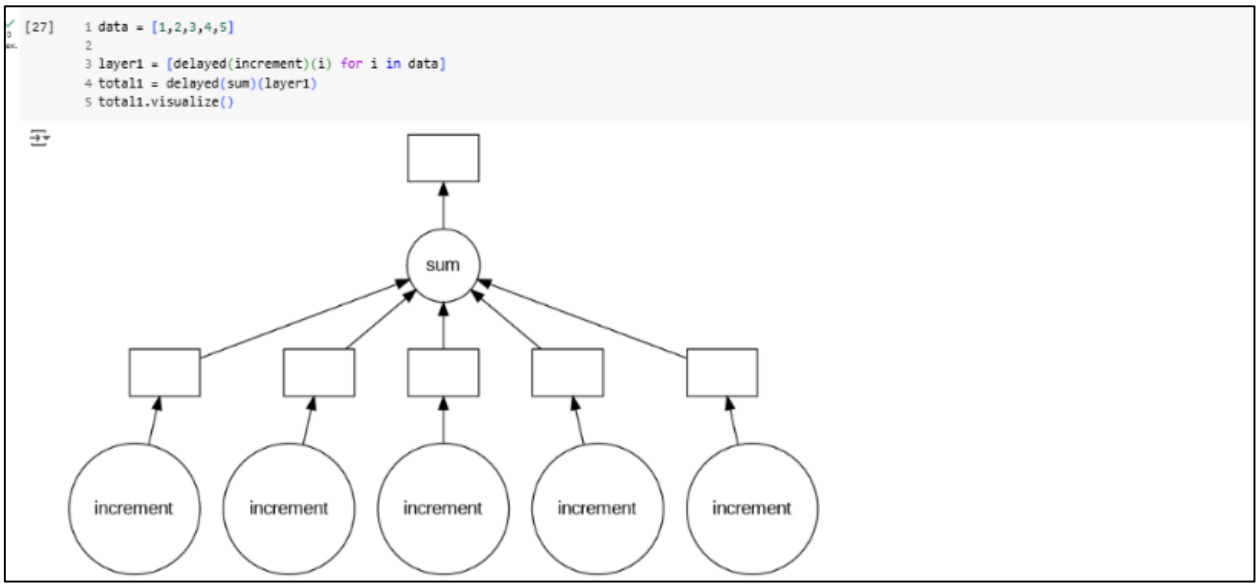
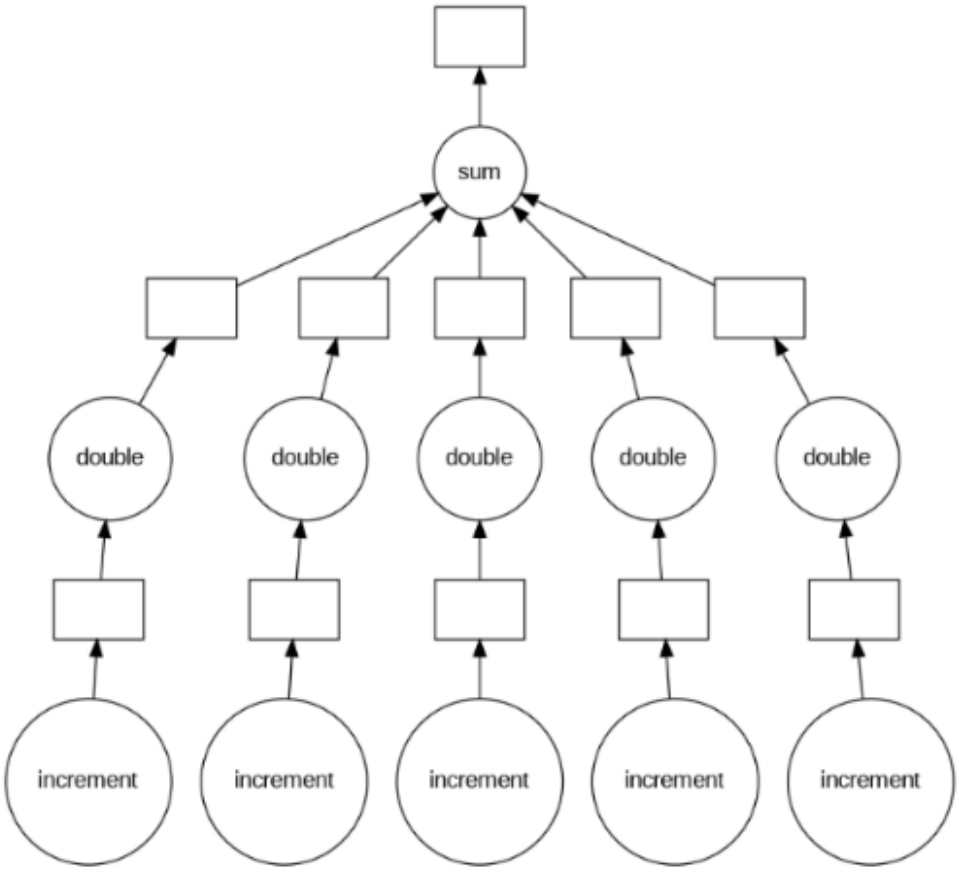



Рис. 10 – Визуализация layer1

Layer1 создается путем циклического перебора списка данных в dask. Этот слой объединяет ранее созданное приращение функции со значениями в списке, а затем использует встроенное суммирование функции для объединения результатов. Layer2 построен циклически для каждого объекта, созданного в Layer1 (Рис. 11).

```
def double(x):
    return x * 2

layer2 = [delayed(double)(j) for j in layer1]
total2 = delayed(sum)(layer2)#.persist()
total2.visualize()
```




```
[29]  z = total2.compute()
      z
```

```
40
```

Рис. 11 – layer2

Вывод: был получен опыт работы с базовыми командами Dask, а также объектом Dask Delayed.