# Task1: ETL Pipeline Using Open-Source Tools

Objective: Build a simple ETL (Extract, Transform, Load) pipeline using open-source tools.

Use available open sources like jupyter notebook or google collab for execution

**Getting Started with Google Colab**

1. **Access Colab:**
   * Go to Google Colab.
   * You may need to sign in with your Google account.
2. **Creating a New Notebook:**
   * Click on "File" > "New notebook" to create a new notebook.
3. **Writing Python Code:**
   * In the new notebook, you can write Python code in the cells. Each cell can be run independently by clicking the play button on the left or by pressing Shift + Enter.

According to the requirement take any csv file from Kaggle (eg: titanic.csv) https://www.kaggle.com/datasets/yasserh/titanic-dataset?resource=download



Pandas: Is a Python library used for working with data sets.

Sqlite3:Used to integrate the SQLite database with Python

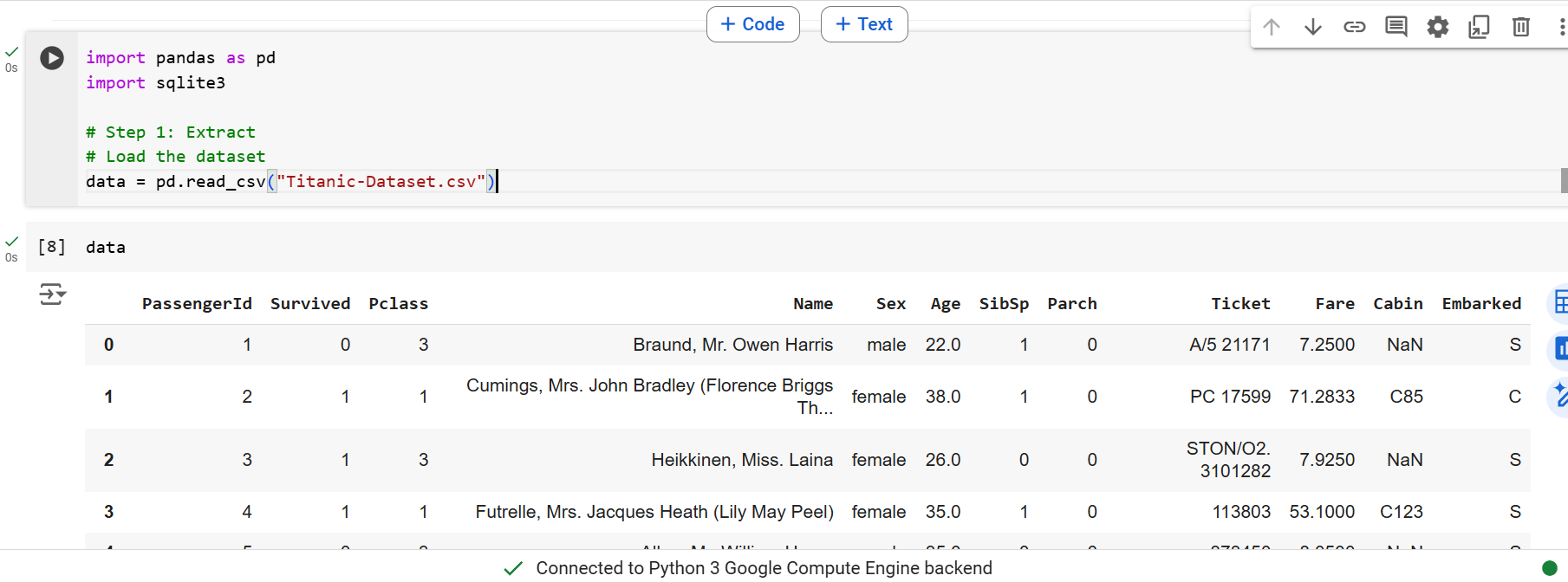
import pandas as pd

import sqlite3

# Step 1: Extract

# Load the dataset

data = pd.read\_csv("Titanic-Dataset.csv")



# Step 2: Transform

# 1. Filter rows to include only passengers who survived

survivors = data[data['Survived'] == 1]

# 2. Calculate the average age of survivors

average\_age = survivors['Age'].mean()

# 3. Remove rows with missing values in dataset

transformed\_data = data.dropna()

# Print average age

print(f"Average age of survivors: {average\_age}")

output: Average age of survivors: 28.343689655172415



# Step 3: Load

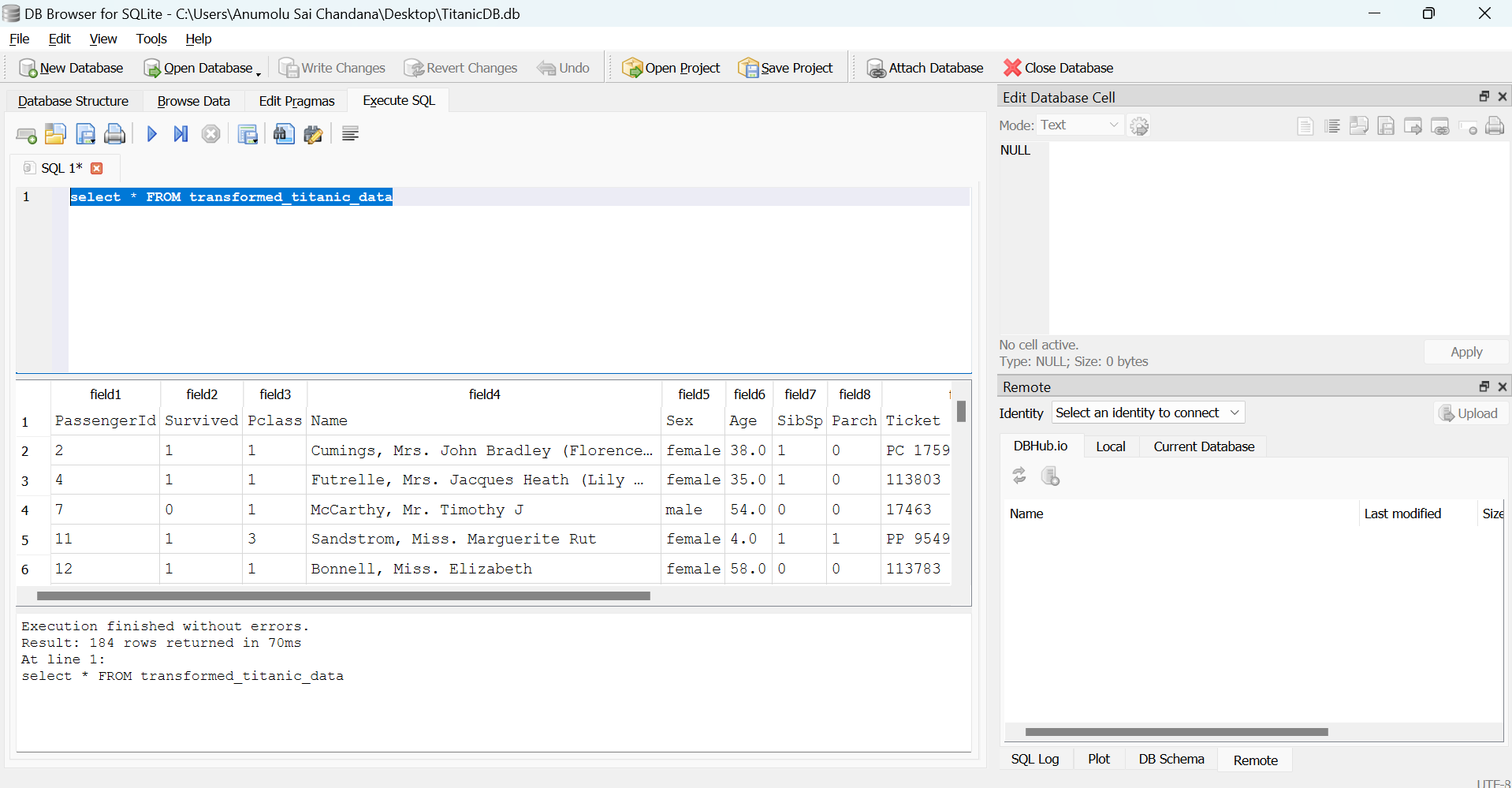
# Load transformed data into a new CSV file

transformed\_data.to\_csv('transformed\_titanic\_data.csv', index=False)

Open sqlite and import transformed\_titanic\_data.csv file and create table name as transformed\_titanic\_data and DB connection as titanic.db

Run this Query

select \* FROM transformed\_titanic\_data



GITHUB: Go to the [GitHub Desktop website](https://desktop.github.com/) and click on the "Download for Windows" button.

Once the download is complete, open the installer file (usually in your Downloads folder) and follow the prompts to install.

After installation, launch GitHub Desktop and sign in with your GitHub account.

[Titanic-Dataset.csv](https://github.com/Anumolu999/ETLPipeline/blob/main/Titanic-Dataset.csv) (random datasource from Kaggle)

[ETLPipeline.ipynb](https://github.com/Anumolu999/ETLPipeline/blob/main/ETLPipeline.ipynb) (python code for extract,transform,load)

[transformed\_titanic\_data.csv](https://github.com/Anumolu999/ETLPipeline/blob/main/transformed_titanic_data.csv) (obtained transformed data file)

[transformed\_titanic\_data.sql](https://github.com/Anumolu999/ETLPipeline/blob/main/transformed_titanic_data.sql) (loaded transformed data in S qlite)

# TASK2: SQL Data Analysis on a Local Database

Perform basic data analysis using SQL on any local or cloud-based relational database.

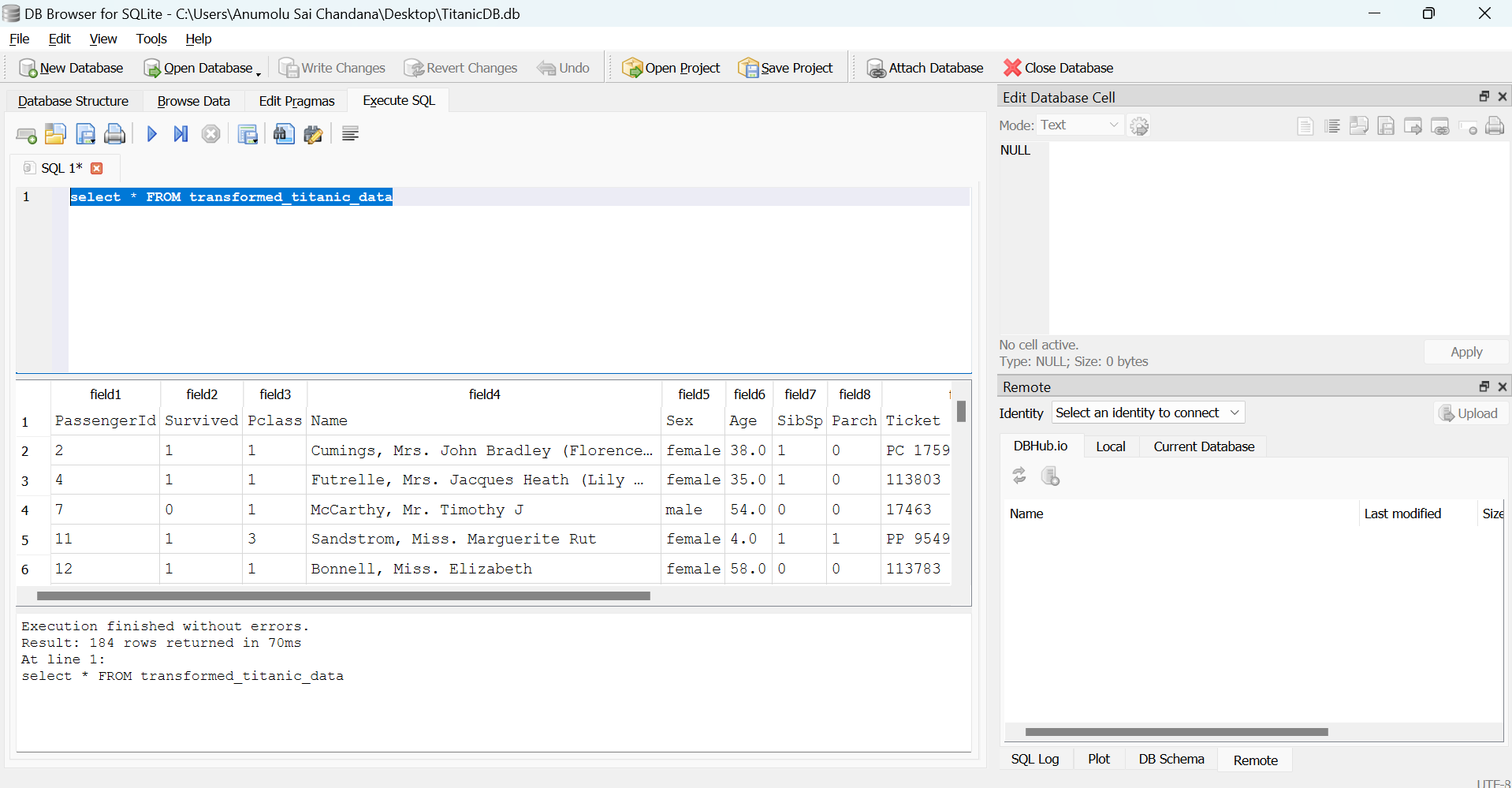
**For Windows**

1. **Download SQLite:**
   * Go to the SQLite download page.
   * Under "Precompiled Binaries for Windows," download the sqlite-tools-standardwin64-x86.zip.
2. **Extract the Files:**
   * Extract the contents of the zip file to a desired location (e.g., C:\sqlite).
3. **Add SQLite to PATH:**
   * Right-click on "This PC" or "My Computer" and select "Properties."
   * Click on "Advanced system settings."
   * Click on the "Environment Variables" button.
   * In the "System variables" section, find the "Path" variable and click "Edit."
   * Click "New" and add the path to the folder where you extracted SQLite (e.g., C:\sqlite).
   * Click "OK" to close all dialog boxes.
4. **Verify Installation:**
   * Open Command Prompt and type sqlite3.
   * If installed correctly, you should see the SQLite command line prompt.

Choose Transformed titanic csv file

select \* from transformed\_titanic\_data;

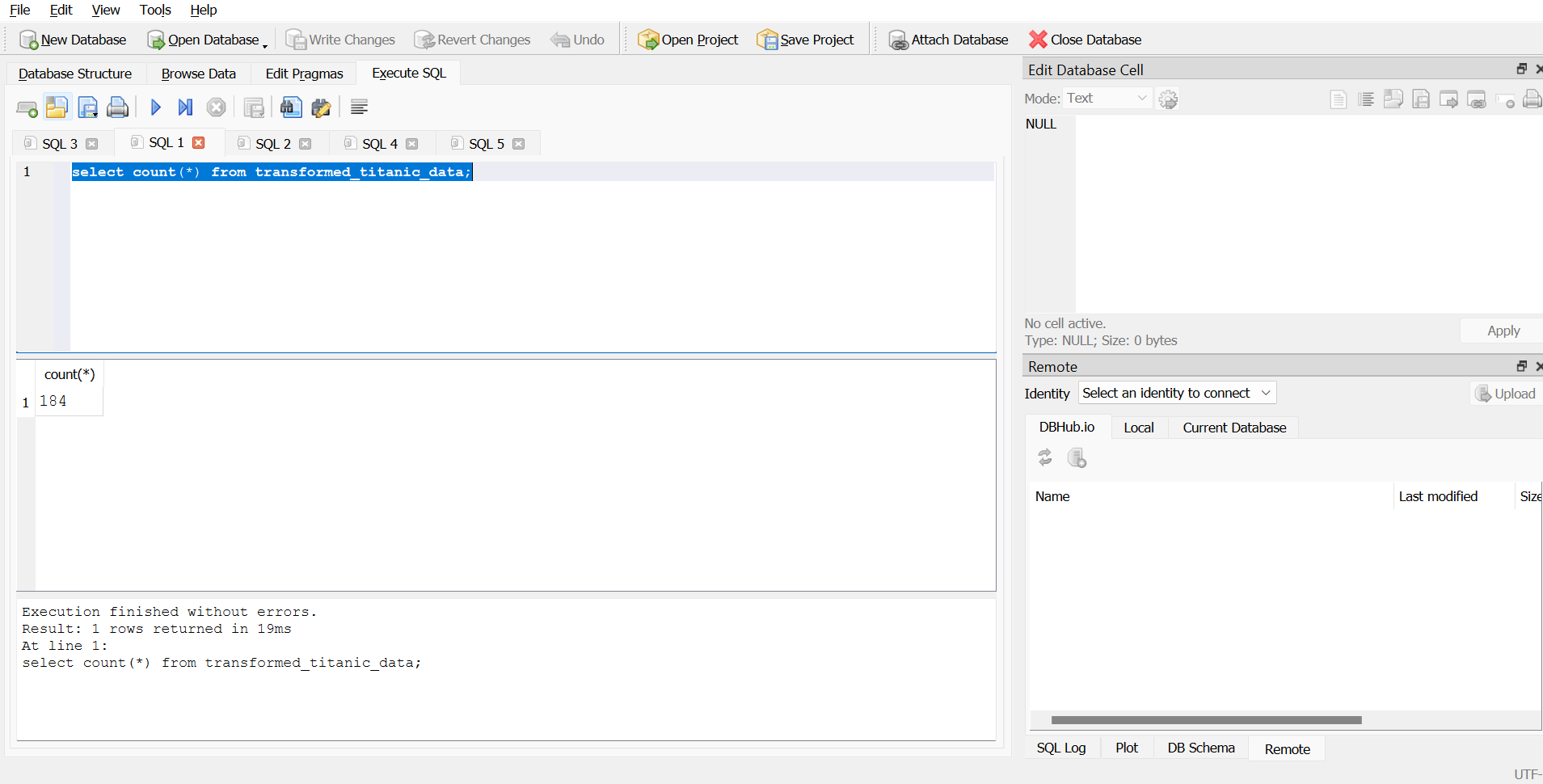
This query will return the total number of rows in the table.



To count the total number of records

select count(\*) from transformed\_titanic\_data;

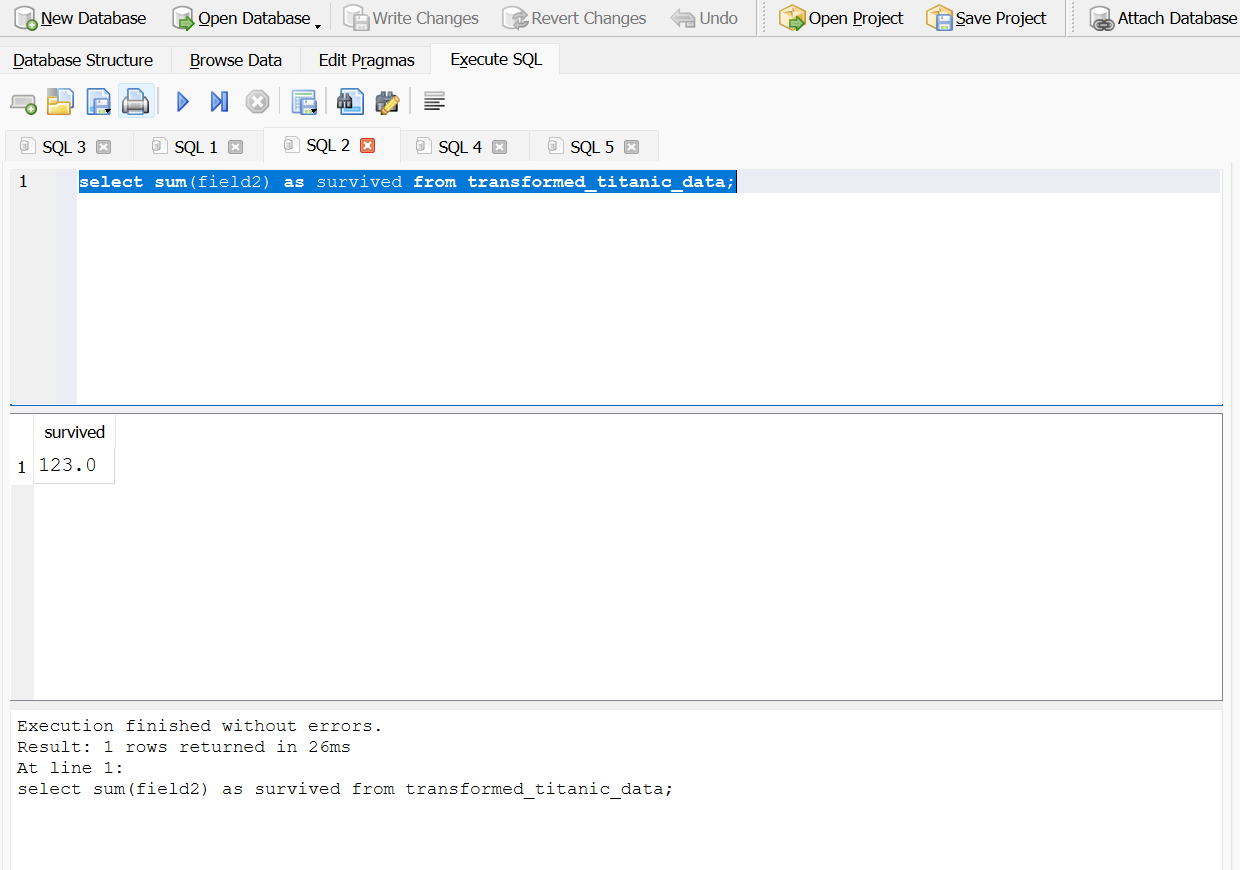
This query will return the total number of rows in the table.



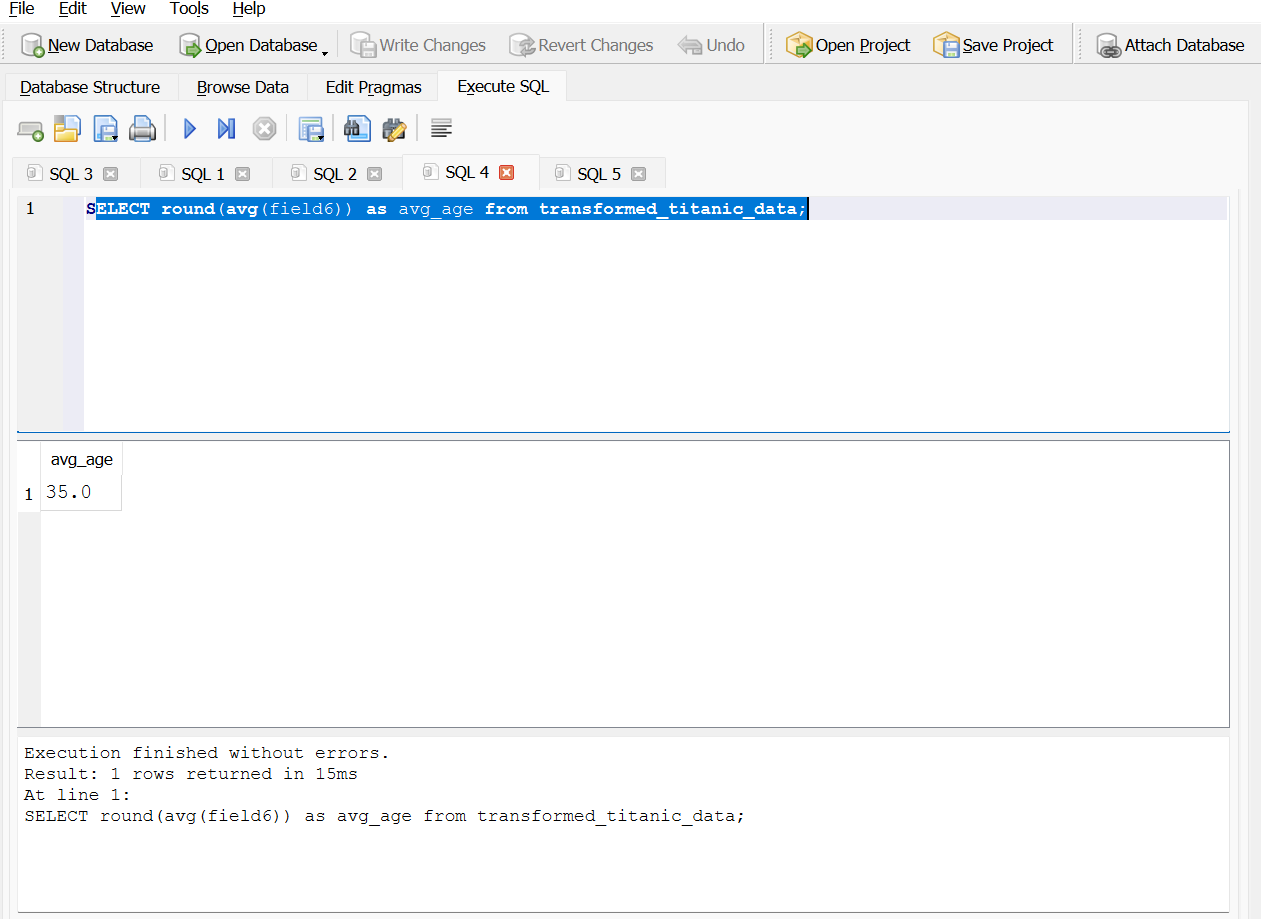
sum or average of a numerical column

select sum(field2) as survived from transformed\_titanic\_data;

This query will return the sum of the values in field2, which presumably indicates the number of survivors.

SELECT round(avg(field6)) as avg\_age from transformed\_titanic\_data;

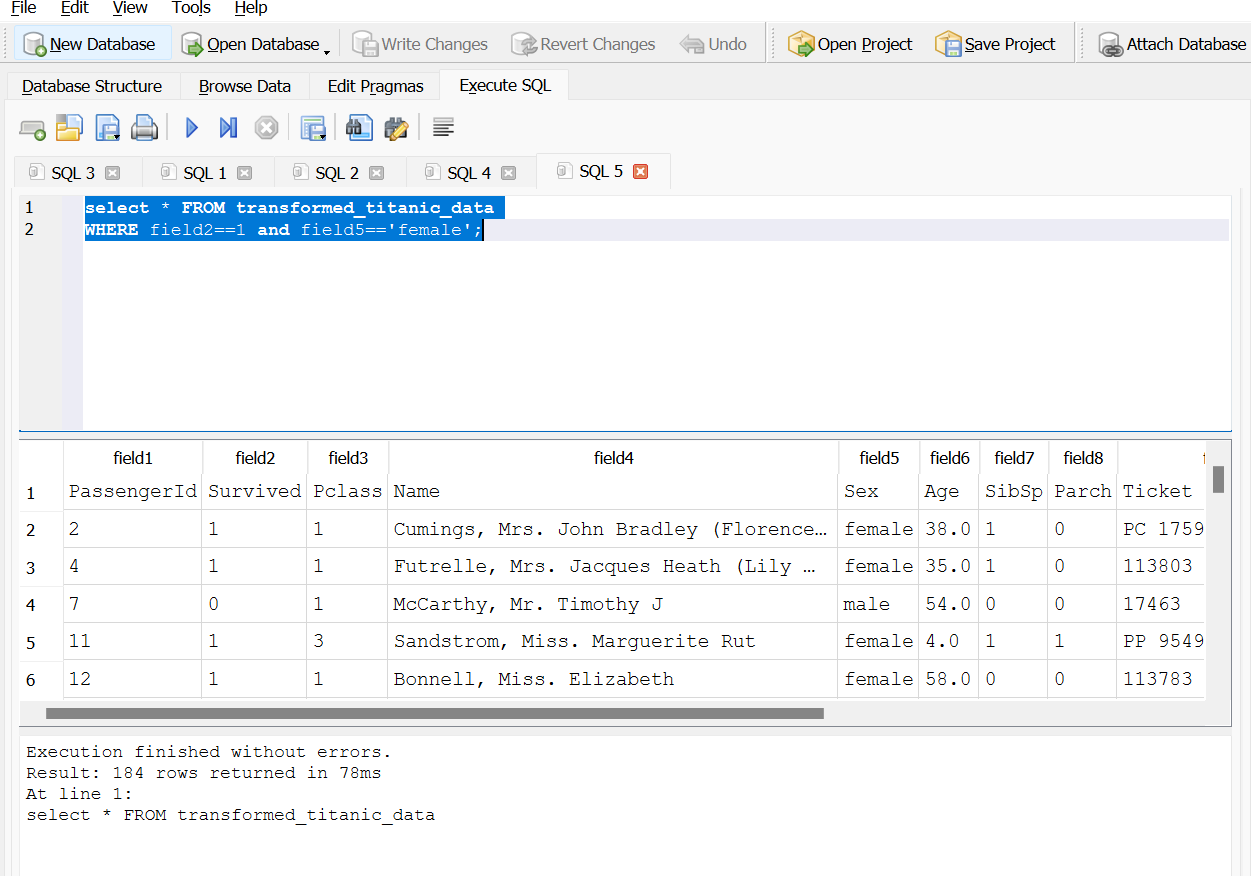
This will return the average age rounded to the nearest whole number

Filter the dataset based on a condition:

select \* FROM transformed\_titanic\_data

WHERE field2==1 and field5=='female';

This will retrieve all records from the transformed\_titanic\_data table where field2 is equal to 1 and field5 is 'female'.



In Github :

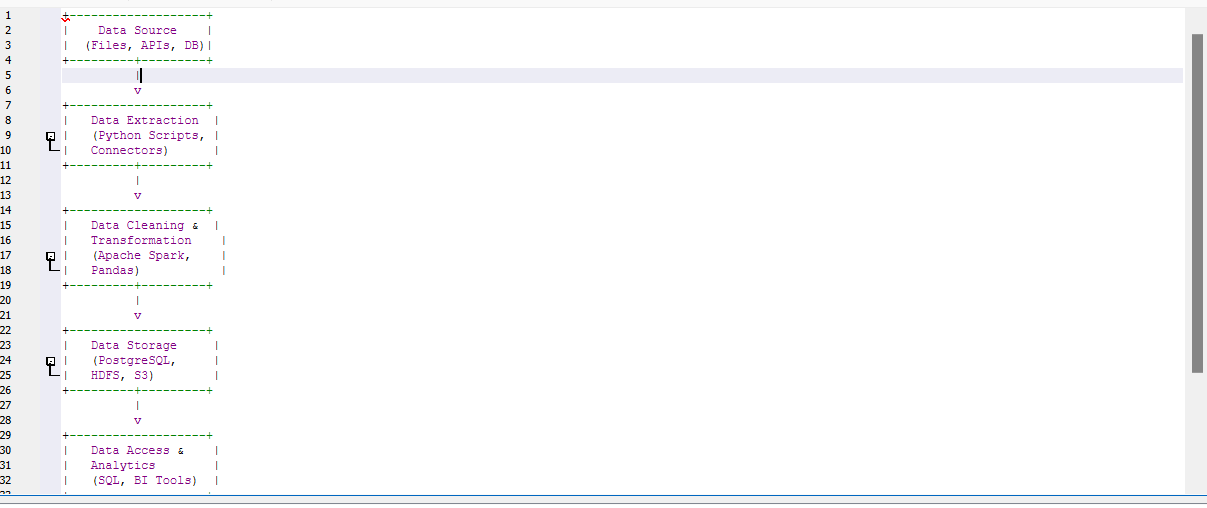
Click on Add file , select upload files

Choose the saved file by selecting from particular folder and upload

Click on commit changes

[titanic queries.sqbpro](https://github.com/Anumolu999/ETLPipeline/blob/main/titanic%20queries.sqbpro) contains SQL queries

# 3.Design a Data Pipeline for Batch Processing



**Explanation**

**Data Source**:

The data can come from various sources such as flat files (CSV, JSON), APIs (RESTful services), or databases (e.g., MySQL, MongoDB). Each source requires a tailored extraction approach.

**Data Extraction**:

Data extraction can be achieved using custom Python scripts or open-source connectors that pull data from APIs or databases. For file sources, simple file reading operations (e.g., using pandas.read\_csv) can be employed.

**Data Cleaning & Transformation**:

Once extracted, the data goes through cleaning and transformation. This step involves:

* + - Removing duplicates
    - Handling missing values
    - Normalizing data formats
    - Aggregating or filtering data as needed
  + Tools like **Apache Spark** (for distributed processing) or **Pandas** (for smaller datasets) can be utilized to perform these operations efficiently.

**Data Storage**:

After transformation, the processed data is stored in a data storage system such as **PostgreSQL** (for relational data) or **HDFS/S3** (for larger unstructured datasets). This allows for scalable storage and efficient querying.

**Data Access & Analytics**:

The final data can be accessed for reporting and analysis using SQL queries, or by employing Business Intelligence tools (e.g., Tableau, Looker) that connect to the storage system.

**Job Scheduling**:

To ensure that the pipeline runs at regular intervals (e.g., daily), a job scheduling tool like **Apache Airflow** can be used. It allows you to define workflows, set dependencies, and manage the execution of various tasks. Alternatively, simpler setups can utilize **cron jobs** for scheduling script execution.

**Data Flow**

The pipeline begins with extracting data from the designated source. Once data is pulled, it flows to the transformation step, where it is cleaned and structured as needed. The transformed data is then loaded into the chosen storage system for future access. Finally, the job scheduling component ensures that the entire process runs automatically at specified intervals, maintaining up-to-date datasets for analysis.

This architecture provides a robust framework for batch data processing, leveraging open-source tools to ensure scalability, flexibility, and efficiency in handling large datasets.

GITHUB: [3.architecture.sql](https://github.com/Anumolu999/ETLPipeline/blob/main/3.architecture.sql)

# 4.Data Quality Check Script

Write a script to perform basic data quality checks on a dataset

Take Random data source from kaggle

<https://www.kaggle.com/datasets/arathipraj/house-data>

Open JUPYTER Notebook import house\_data.csv file

import pandas as pd

import matplotlib.pyplot as plt

#import seaborn as sns

#import numpy as np

#from scipy.stats import norm

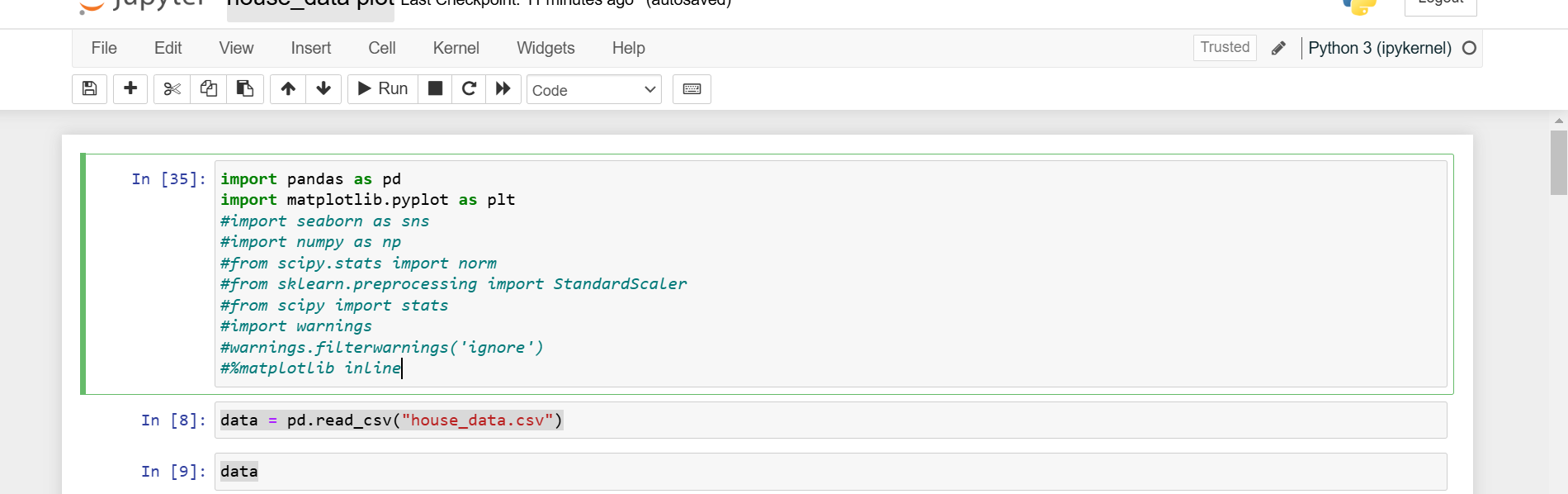
#from sklearn.preprocessing import StandardScaler

#from scipy import stats

#import warnings

#warnings.filterwarnings('ignore')

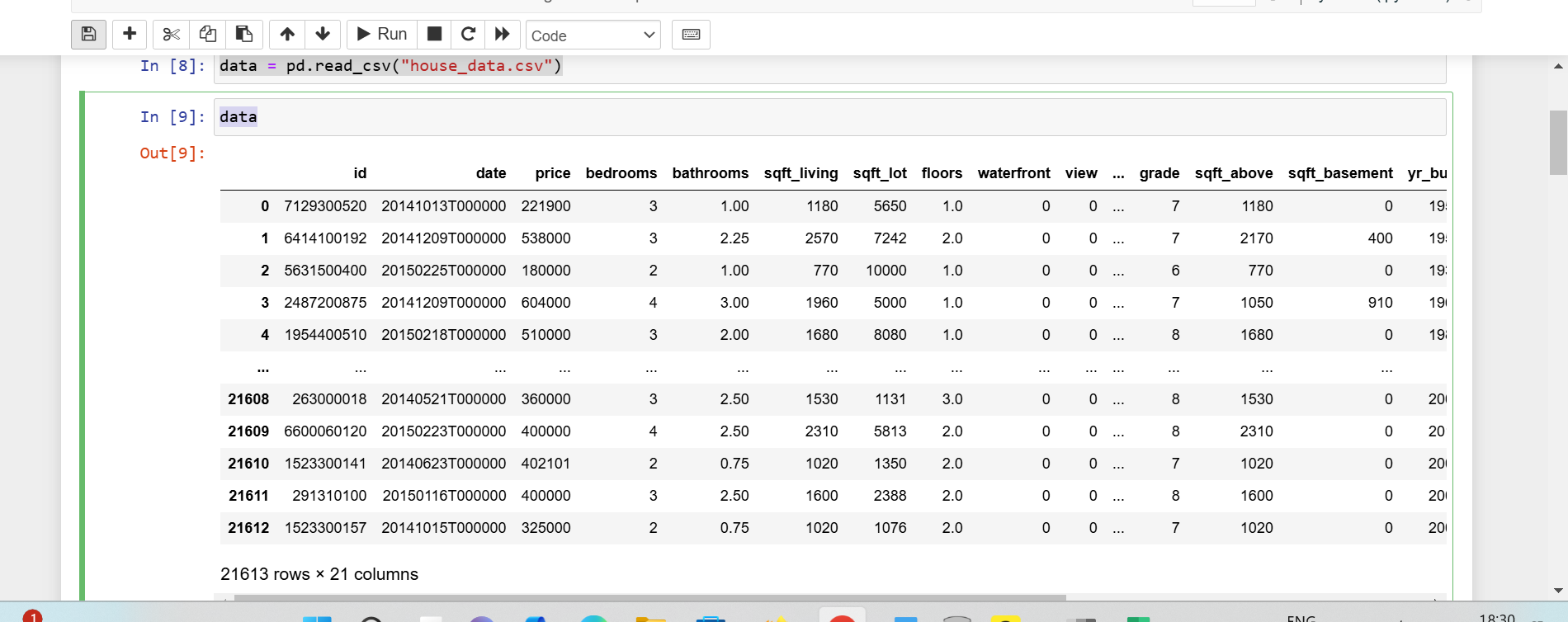
#%matplotlib inline



Read a comma-separated values (csv) file into DataFrame

data = pd.read\_csv("house\_data.csv")

data



df\_train = pd.read\_csv('house\_data.csv')

df\_train.

The df. shape method provides information about the number of rows and columns in a DataFrame quickly and easily.

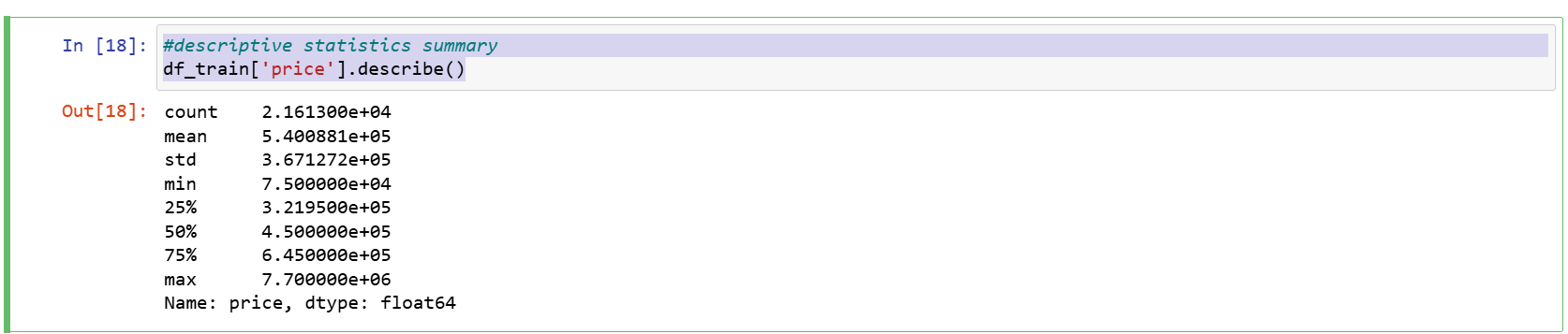
df\_train.shape

(21613, 21)

#descriptive statistics summary

df\_train['price'].describe()

Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.



#missing data

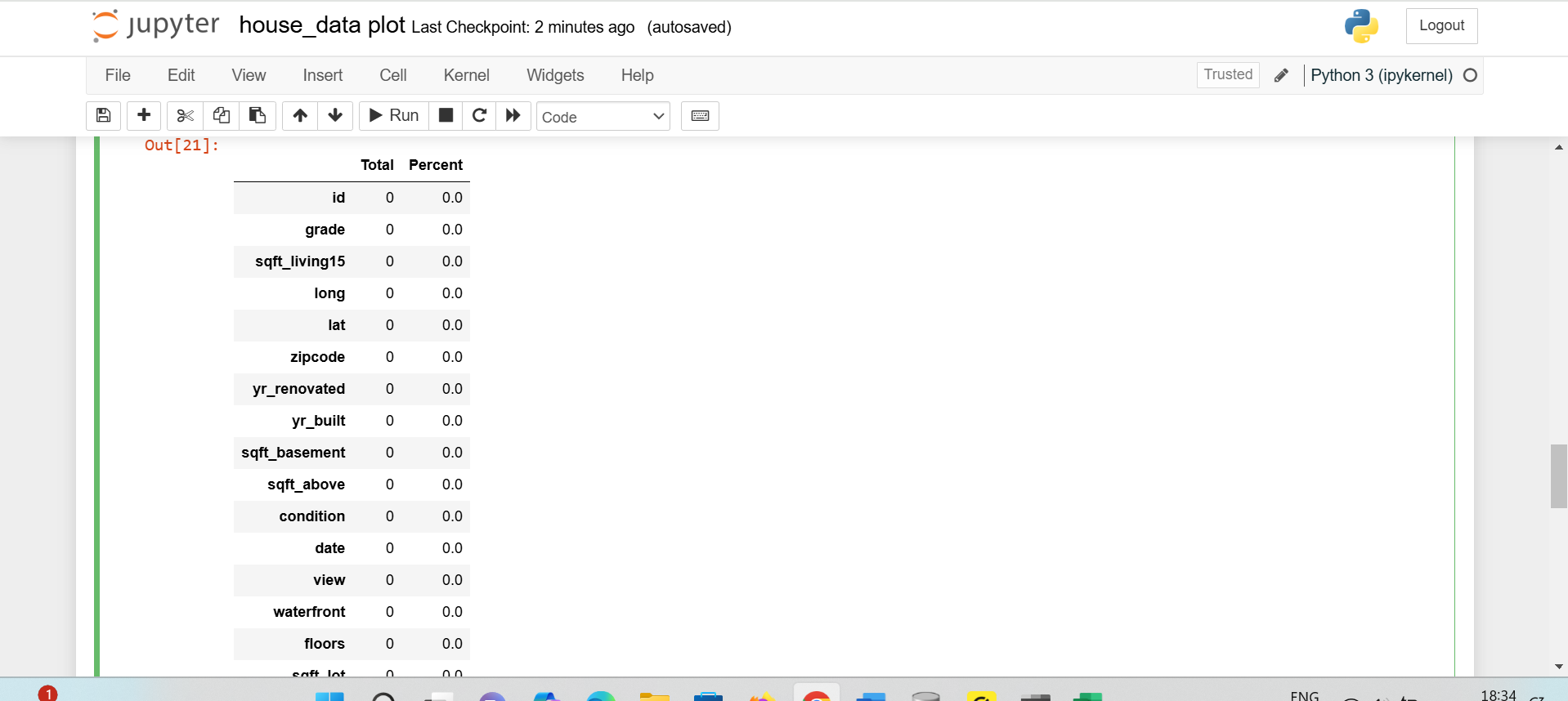
Detect missing values for an array-like object

total = df\_train.isnull().sum().sort\_values(ascending=False)

percent =(df\_train.isnull().sum()/df\_train.isnull().count()).sort\_values(ascending=False)

missing\_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

missing\_data.head(20)



sum(df\_train.isnull().sum())

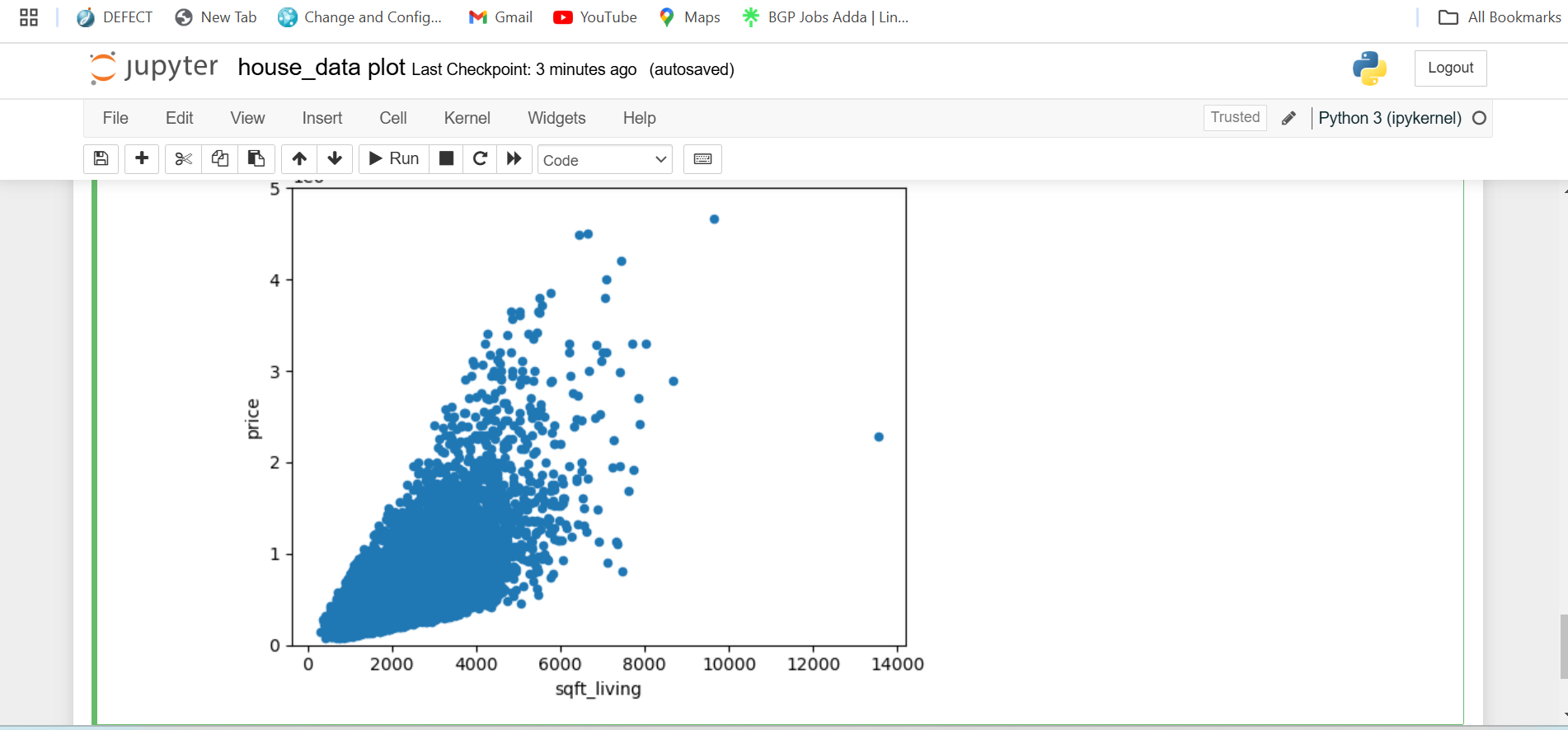
result:-0

#scatter plot area/price

var = 'sqft\_living'

data = pd.concat([df\_train['price'], df\_train[var]], axis=1)

data.plot.scatter(x=var, y='price', ylim=(0,5000000));



GITHUB: [house\_data.csv](https://github.com/Anumolu999/ETLPipeline/blob/main/house_data.csv)

[house\_data plot.py](https://github.com/Anumolu999/ETLPipeline/blob/main/house_data%20plot.py)