Detailed Project

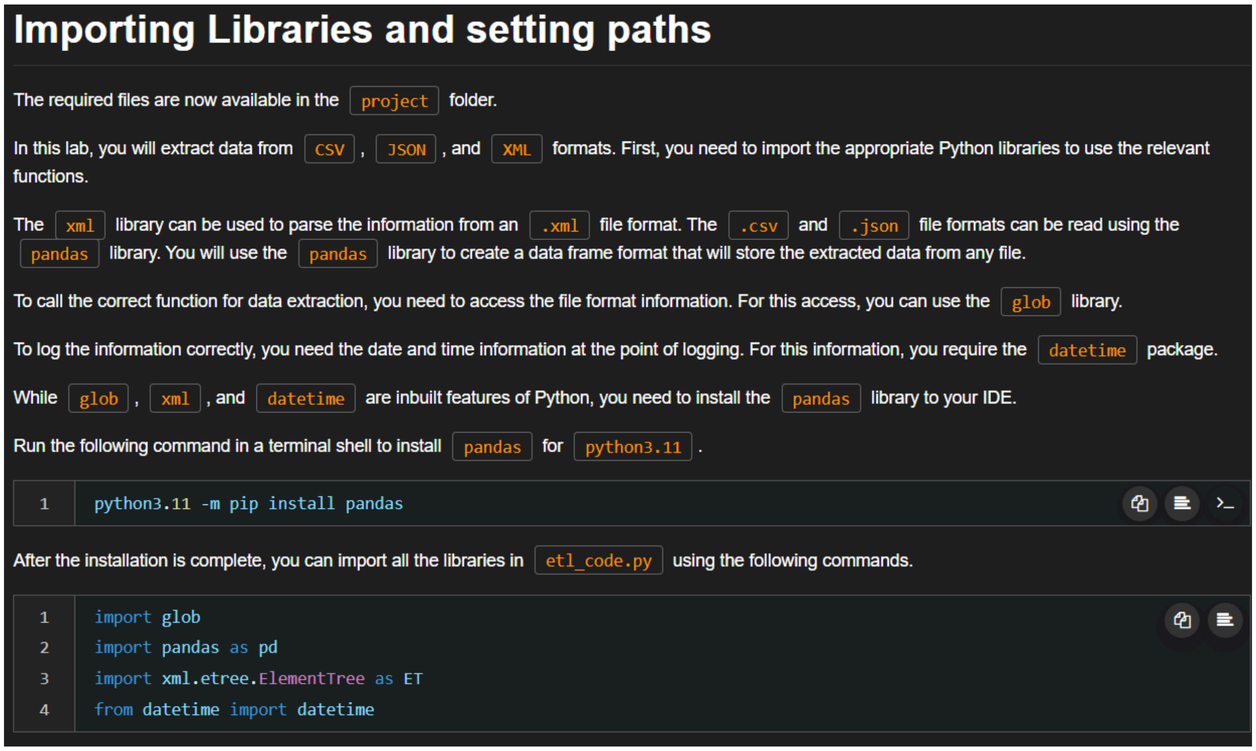
Introduction

Extract, Transform and Load (ETL) operations are of extreme importance in the role of a Data engineer. A data engineer extracts data from multiple sources and different file formats, transforms the extracted data to predefined settings and then loads the data to a database for further processing. In this lab, you will get hands-on practice of performing these operations.

Objectives

After completing this lab, you will be able to:

- Read CSV, JSON, and XML file types.
- Extract the required data from the different file types.
- Transform data to the required format.
- Save the transformed data in a ready-to-load format, which can be loaded into an RDBMS.



Importing Libraries and setting paths

The required files are now available in the `project` folder.

In this lab, you will extract data from `CSV`, `JSON`, and `XML` formats. First, you need to import the appropriate Python libraries to use the relevant functions.

The `xml` library can be used to parse the information from an `.xml` file format. The `.csv` and `.json` file formats can be read using the `pandas` library. You will use the `pandas` library to create a data frame format that will store the extracted data from any file.

To call the correct function for data extraction, you need to access the file format information. For this access, you can use the `glob` library.

To log the information correctly, you need the date and time information at the point of logging. For this information, you require the `datetime` package.

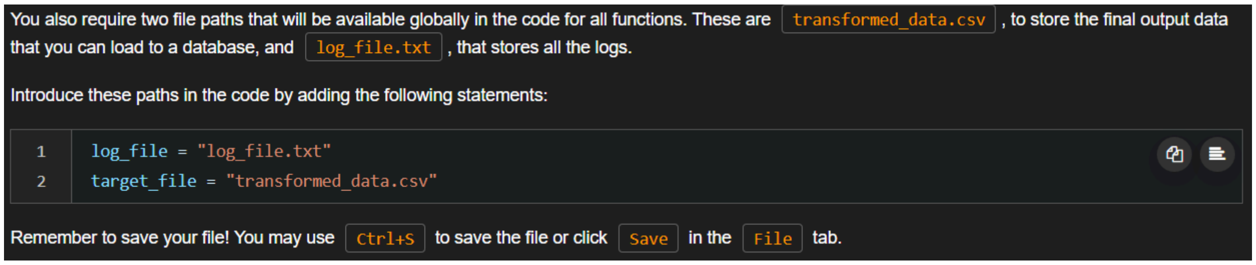
While `glob`, `xml`, and `datetime` are inbuilt features of Python, you need to install the `pandas` library to your IDE.

Run the following command in a terminal shell to install `pandas` for `python3.11`.

```
python3.11 -m pip install pandas
```

After the installation is complete, you can import all the libraries in `etl_code.py` using the following commands.

```
import glob
import pandas as pd
import xml.etree.ElementTree as ET
from datetime import datetime
```

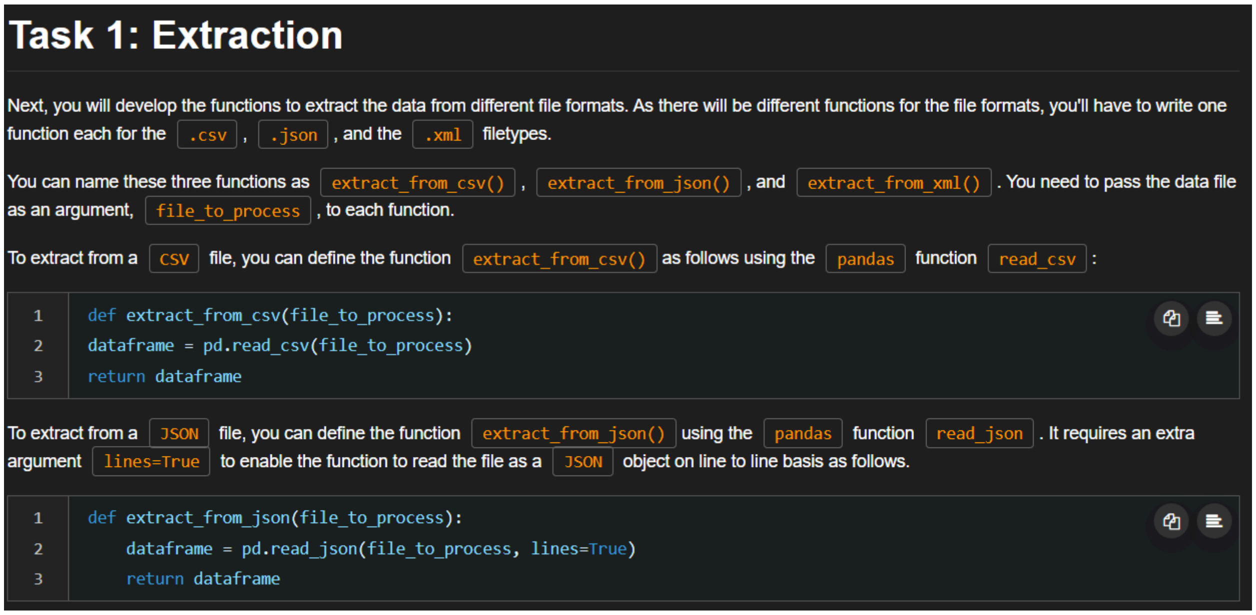


You also require two file paths that will be available globally in the code for all functions. These are `transformed_data.csv`, to store the final output data that you can load to a database, and `log_file.txt`, that stores all the logs.

Introduce these paths in the code by adding the following statements:

```
log_file = "log_file.txt"
target_file = "transformed_data.csv"
```

Remember to save your file! You may use `Ctrl+S` to save the file or click `Save` in the `File` tab.



Task 1: Extraction

Next, you will develop the functions to extract the data from different file formats. As there will be different functions for the file formats, you'll have to write one function each for the `.csv`, `.json`, and the `.xml` filetypes.

You can name these three functions as `extract_from_csv()`, `extract_from_json()`, and `extract_from_xml()`. You need to pass the data file as an argument, `file_to_process`, to each function.

To extract from a `CSV` file, you can define the function `extract_from_csv()` as follows using the `pandas` function `read_csv`:

```
def extract_from_csv(file_to_process):
dataframe = pd.read_csv(file_to_process)
return dataframe
```

To extract from a `JSON` file, you can define the function `extract_from_json()` using the `pandas` function `read_json`. It requires an extra argument `lines=True` to enable the function to read the file as a `JSON` object on line to line basis as follows.

```
def extract_from_json(file_to_process):
    dataframe = pd.read_json(file_to_process, lines=True)
    return dataframe
```

Task 1: Extraction

Next, you will develop the functions to extract the data from different file formats. As there will be different functions for the file formats, you'll have to write one function each for the `.csv`, `.json`, and the `.xml` filetypes.

You can name these three functions as `extract_from_csv()`, `extract_from_json()`, and `extract_from_xml()`. You need to pass the data file as an argument, `file_to_process`, to each function.

To extract from a `CSV` file, you can define the function `extract_from_csv()` as follows using the `pandas` function `read_csv`:

```python
def extract_from_csv(file_to_process):
dataframe = pd.read_csv(file_to_process)
return dataframe
```

To extract from a `JSON` file, you can define the function `extract_from_json()` using the `pandas` function `read_json`. It requires an extra argument `lines=True` to enable the function to read the file as a `JSON` object on line to line basis as follows.

```python
def extract_from_json(file_to_process):
    dataframe = pd.read_json(file_to_process, lines=True)
    return dataframe
```

To extract from an `XML` file, you need first to parse the data from the file using the `ElementTree` function. You can then extract relevant information from this data and append it to a pandas dataframe as follows.

Note: Adding Data to DataFrames using `pd.concat`

In this lab, we use `pd.concat` to append data to an existing DataFrame. This method is recommended because the `append` method is deprecated. `pd.concat` offers better efficiency and flexibility, especially when combining multiple DataFrames.

Why use `pd.concat`:

- *`pd.concat` is more efficient when adding rows or combining multiple DataFrames.*
- *It provides better control over the operation, allowing you to concatenate along rows or columns.*
- *It also includes the `ignore_index=True` argument, which resets the index to avoid duplication when combining DataFrames.*

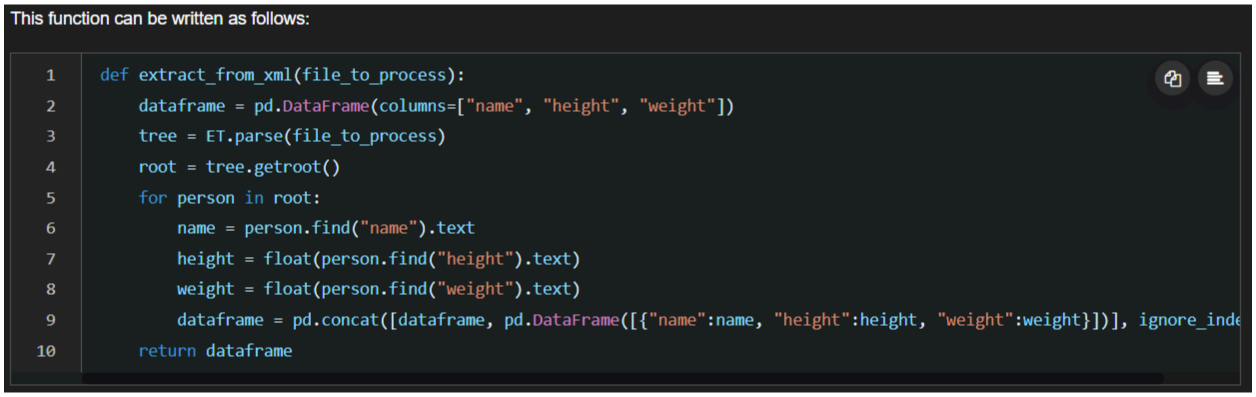
Example:

```python
import pandas as pd

# Create DataFrames
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5], 'B': [6]})

# Use concat
result = pd.concat([df1, df2], ignore_index=True)
print(result)
```

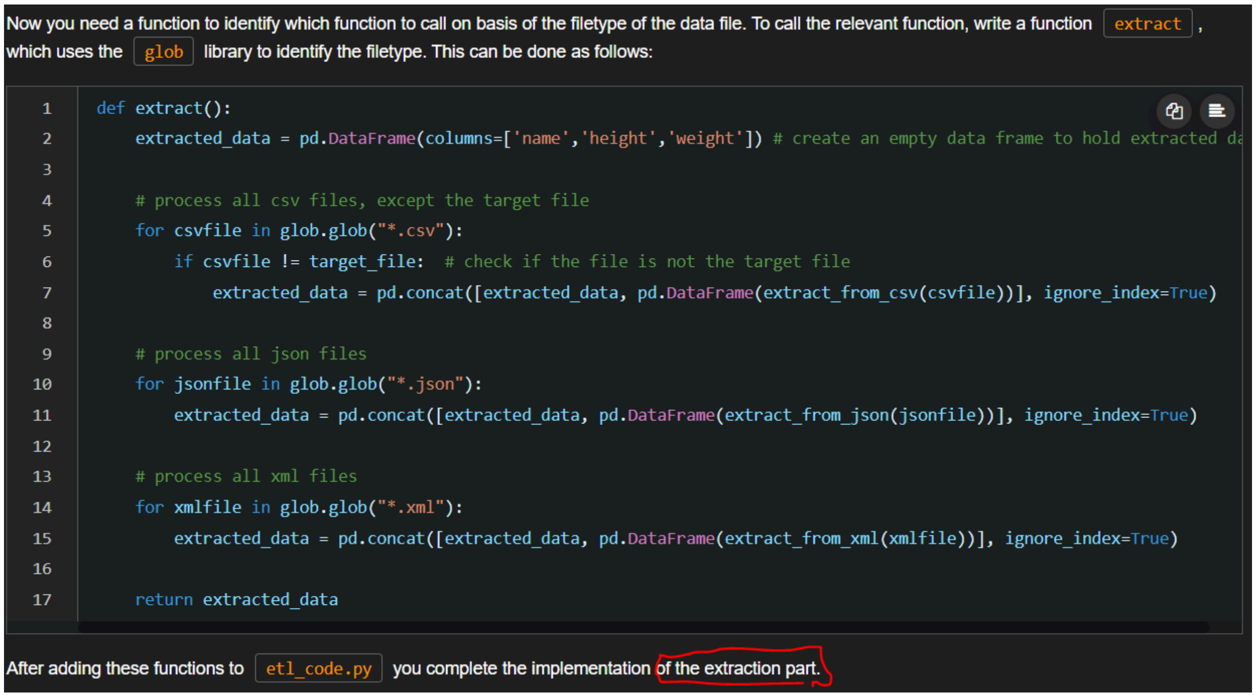
Output:

```
 A B
0 1 3
1 2 4
2 5 6
```

Note: You must know the headers of the extracted data to write this function. In this data, you extract "name", "height", and "weight" headers for different persons.

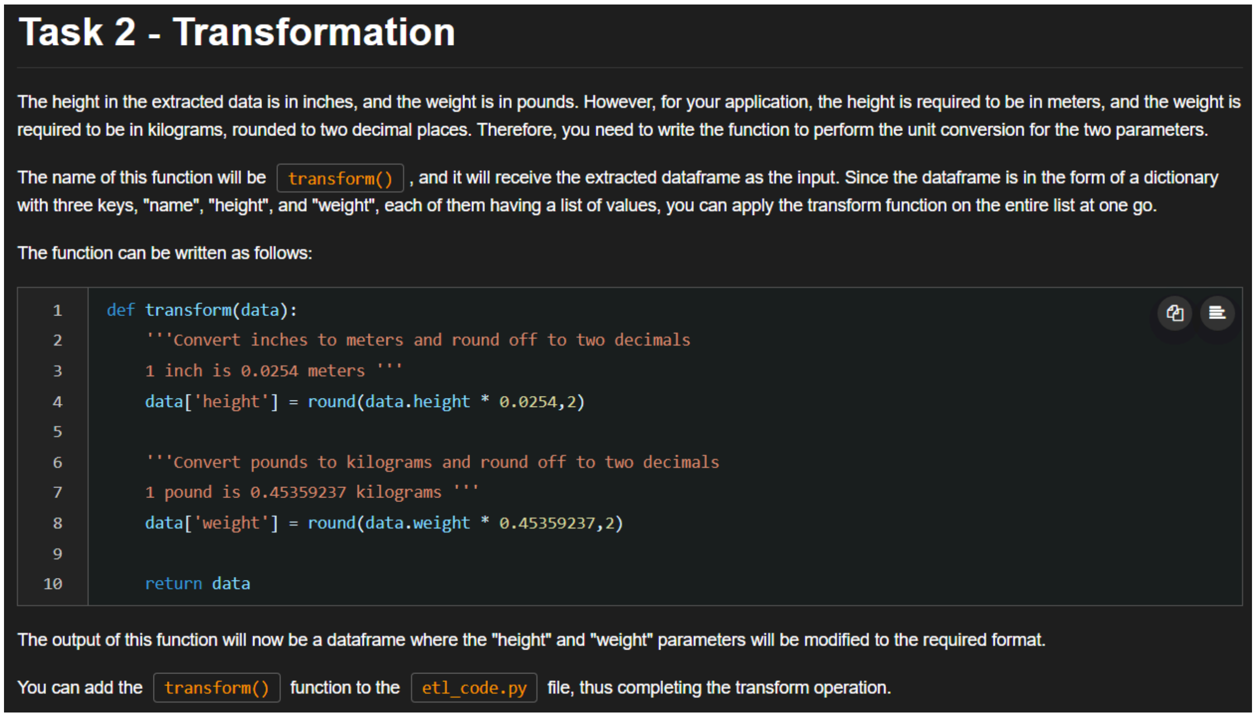

This function can be written as follows:

```python
def extract_from_xml(file_to_process):
    dataframe = pd.DataFrame(columns=["name", "height", "weight"])
    tree = ET.parse(file_to_process)
    root = tree.getroot()
    for person in root:
        name = person.find("name").text
        height = float(person.find("height").text)
        weight = float(person.find("weight").text)
        dataframe = pd.concat([dataframe, pd.DataFrame([{"name":name, "height":height, "weight":weight}])], ignore_inde
    return dataframe
```


Now you need a function to identify which function to call on basis of the filetype of the data file. To call the relevant function, write a function `extract`, which uses the `glob` library to identify the filetype. This can be done as follows:

```python
def extract():
    extracted_data = pd.DataFrame(columns=['name','height','weight']) # create an empty data frame to hold extracted da

    # process all csv files, except the target file
    for csvfile in glob.glob("*.csv"):
        if csvfile != target_file:  # check if the file is not the target file
            extracted_data = pd.concat([extracted_data, pd.DataFrame(extract_from_csv(csvfile))], ignore_index=True)

    # process all json files
    for jsonfile in glob.glob("*.json"):
        extracted_data = pd.concat([extracted_data, pd.DataFrame(extract_from_json(jsonfile))], ignore_index=True)

    # process all xml files
    for xmlfile in glob.glob("*.xml"):
        extracted_data = pd.concat([extracted_data, pd.DataFrame(extract_from_xml(xmlfile))], ignore_index=True)

    return extracted_data
```

After adding these functions to `etl_code.py` you complete the implementation of the extraction part.



Task 2 - Transformation

The height in the extracted data is in inches, and the weight is in pounds. However, for your application, the height is required to be in meters, and the weight is required to be in kilograms, rounded to two decimal places. Therefore, you need to write the function to perform the unit conversion for the two parameters.

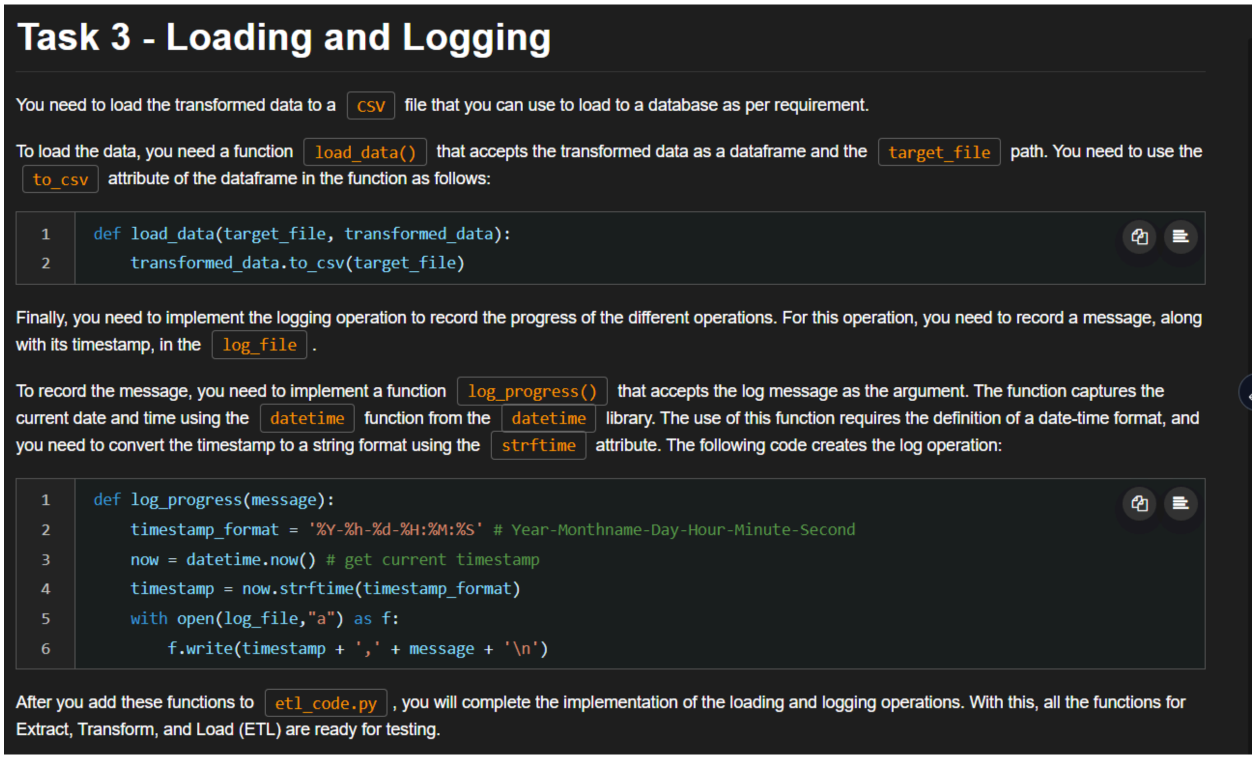
The name of this function will be `transform()`, and it will receive the extracted dataframe as the input. Since the dataframe is in the form of a dictionary with three keys, "name", "height", and "weight", each of them having a list of values, you can apply the transform function on the entire list at one go.

The function can be written as follows:

```python
def transform(data):
    '''Convert inches to meters and round off to two decimals
    1 inch is 0.0254 meters '''
    data['height'] = round(data.height * 0.0254,2)

    '''Convert pounds to kilograms and round off to two decimals
    1 pound is 0.45359237 kilograms '''
    data['weight'] = round(data.weight * 0.45359237,2)

    return data
```

The output of this function will now be a dataframe where the "height" and "weight" parameters will be modified to the required format.

You can add the `transform()` function to the `etl_code.py` file, thus completing the transform operation.



Task 3 - Loading and Logging

You need to load the transformed data to a `CSV` file that you can use to load to a database as per requirement.

To load the data, you need a function `load_data()` that accepts the transformed data as a dataframe and the `target_file` path. You need to use the `to_csv` attribute of the dataframe in the function as follows:

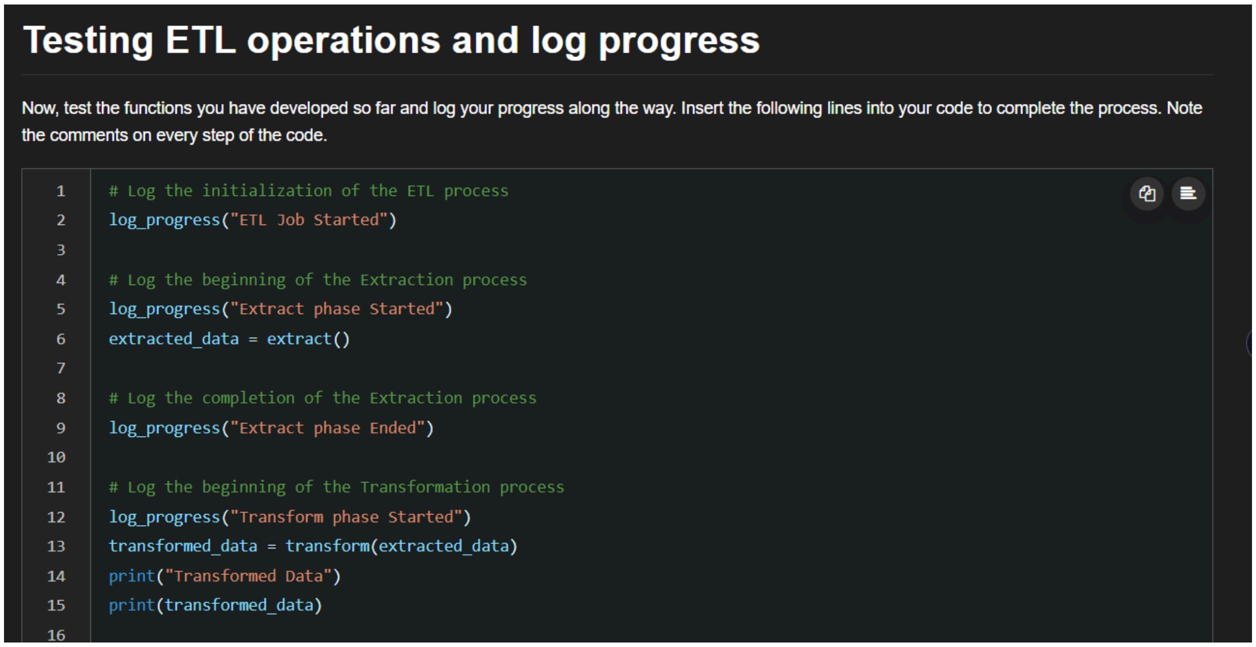
```python
def load_data(target_file, transformed_data):
    transformed_data.to_csv(target_file)
```

Finally, you need to implement the logging operation to record the progress of the different operations. For this operation, you need to record a message, along with its timestamp, in the `log_file`.

To record the message, you need to implement a function `log_progress()` that accepts the log message as the argument. The function captures the current date and time using the `datetime` function from the `datetime` library. The use of this function requires the definition of a date-time format, and you need to convert the timestamp to a string format using the `strftime` attribute. The following code creates the log operation:

```python
def log_progress(message):
    timestamp_format = '%Y-%h-%d-%H:%M:%S' # Year-Monthname-Day-Hour-Minute-Second
    now = datetime.now() # get current timestamp
    timestamp = now.strftime(timestamp_format)
    with open(log_file,"a") as f:
        f.write(timestamp + ',' + message + '\n')
```

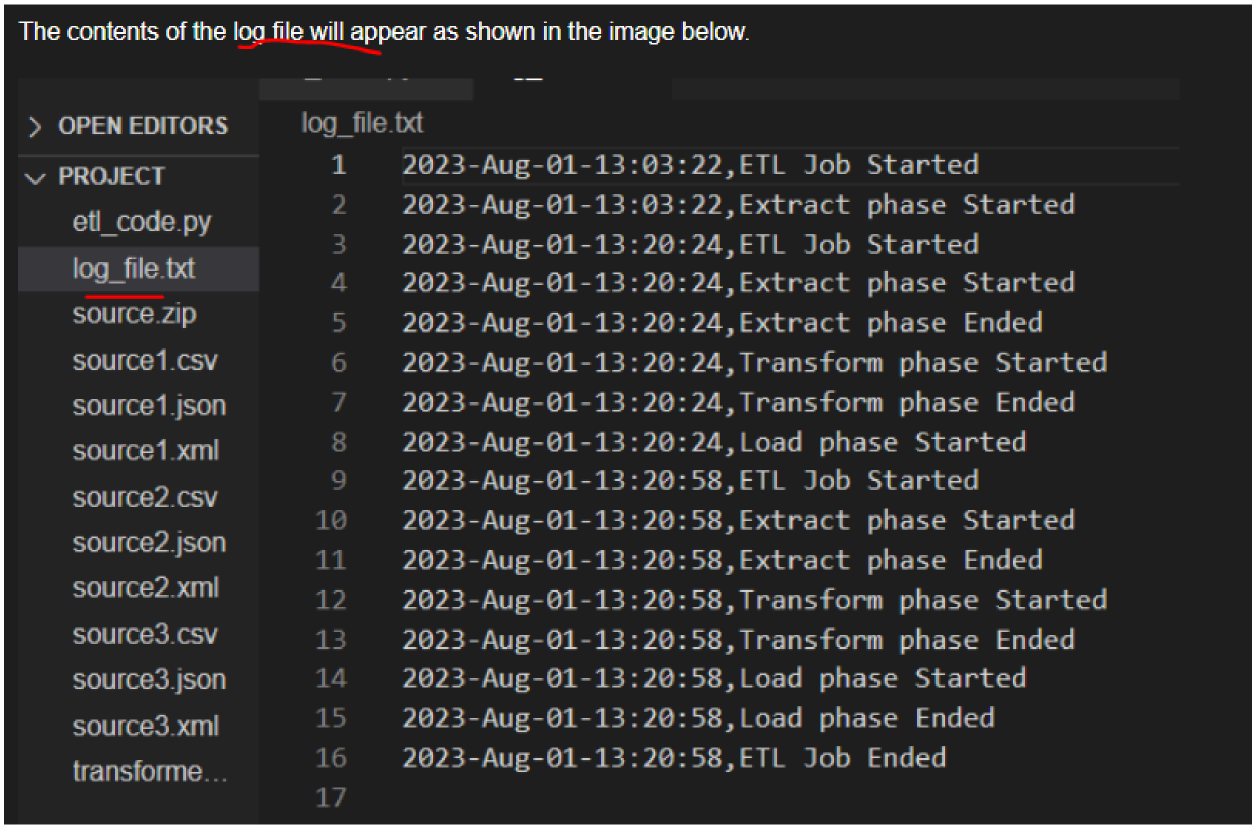
After you add these functions to `etl_code.py`, you will complete the implementation of the loading and logging operations. With this, all the functions for Extract, Transform, and Load (ETL) are ready for testing.



Testing ETL operations and log progress

Now, test the functions you have developed so far and log your progress along the way. Insert the following lines into your code to complete the process. Note the comments on every step of the code.

```python
# Log the initialization of the ETL process
log_progress("ETL Job Started")

# Log the beginning of the Extraction process
log_progress("Extract phase Started")
extracted_data = extract()

# Log the completion of the Extraction process
log_progress("Extract phase Ended")

# Log the beginning of the Transformation process
log_progress("Transform phase Started")
transformed_data = transform(extracted_data)
print("Transformed Data")
print(transformed_data)

```



The contents of the log file will appear as shown in the image below.

```
2023-Aug-01-13:03:22,ETL Job Started
2023-Aug-01-13:03:22,Extract phase Started
2023-Aug-01-13:20:24,ETL Job Started
2023-Aug-01-13:20:24,Extract phase Started
2023-Aug-01-13:20:24,Extract phase Ended
2023-Aug-01-13:20:24,Transform phase Started
2023-Aug-01-13:20:24,Transform phase Ended
2023-Aug-01-13:20:24,Load phase Started
2023-Aug-01-13:20:58,ETL Job Started
2023-Aug-01-13:20:58,Extract phase Started
2023-Aug-01-13:20:58,Extract phase Ended
2023-Aug-01-13:20:58,Transform phase Started
2023-Aug-01-13:20:58,Transform phase Ended
2023-Aug-01-13:20:58,Load phase Started
2023-Aug-01-13:20:58,Load phase Ended
2023-Aug-01-13:20:58,ETL Job Ended
```