Python Project for Data Engineering

A Simple ETL Project:

This notebook have the detail implementation of the project that is compulosry for the **IBM Data Engineering Professional Certification**.

Objectives

After completing this lab you will be able to:

- Read CSV and JSON file types.
- Extract data from the above file types.
- Transform data.
- Save the **transformed data in a ready-to-load format** which data engineers can use to load into an RDBMS.

Import the Important Libraries

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

EXAMPLE: 1

Downloading the Source Files which are stored in S3 bucket.

- We will use **Wget** which is a **networking command-line tool** that lets you download files and interact with REST APIs.

- It supports the HTTP , HTTPS , FTP , and FTPS internet protocols. Wget can deal with unstable and slow network connections. In the event of a download failure, Wget keeps trying until the entire file has been retrieved.

```
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
```

```
--2022-09-07 14:50:21--  https://cf-courses-data.s3.us.cloud-object-storage.appdoma
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-da
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-course
HTTP request sent, awaiting response... 200 OK
Length: 2707 (2.6K) [application/zip]
Saving to: ‘source.zip’

source.zip            0%[                    ]       0  --.-KB/s               sour

2022-09-07 14:50:22 (822 MB/s) - ‘source.zip’ saved [2707/2707]
```

In DataLore Menu bar, you will see Attach Data Option. Click on it and you will see the downloaded Zip file.

Unzip Files

```
# Now unzip the source.zip file
!unzip source.zip
```

```
Archive:  source.zip
  inflating: source3.json
  inflating: source1.csv
  inflating: source2.csv
  inflating: source3.csv
  inflating: source1.json
  inflating: source2.json
  inflating: source1.xml
  inflating: source2.xml
  inflating: source3.xml
```

You can see different files with different formats.

Set Paths:

```
tmpfile = "temp.tmp"
logfile = "logfile.txt"
targetfile = "transformed_data.csv"
```

Extract:

Lets move to our first step which is Extract. In this Step, we will extract data from each file into one file.

```
# CSV Extract Function
def extracting_from_csv(file):
    dataframe = pd.read_csv(file)
    return dataframe

# JSON Extract Function
def extracting_from_json(file):
    dataframe = pd.read_json(file, lines=True) # Read the file as a json object per l
    return dataframe

# XML Extract Function
def extracting_from_xml(file):
    dataframe = pd.DataFrame(columns=["name", "height", "weight"])
    tree = ET.parse(file)
    root = tree.getroot()
    for i in root:
        name = i.find("name").text
        height = i.find("height").text
        weight = i.find("weight").text
        dataframe = dataframe.append({"name": name, "height": height, "weight": weigh
        return dataframe
```

Extract Function:

```
def extract():
    # First Create an Empty Df
    extracted_data = pd.DataFrame(columns=['name','height','weight'])

    # Process all csv files: Use Glob
    for csvfile in glob.glob("*.csv"):
        extracted_data = extracted_data.append(extracting_from_csv(csvfile), ignore_i
        # When ignore_index=True, then the order of each row would be the same as the
        # the row was appended to the data frame.

    # Process all json files
    for jsonfile in glob.glob("*.json"):
        extracted_data = extracted_data.append(extracting_from_json(jsonfile), ignore

    # Process all xml files
    for xmlfile in glob.glob("*.xml"):
        extracted_data = extracted_data.append(extracting_from_xml(xmlfile), ignore_i

    return extracted_data
```

Transform:

In this case, th transform function does the following tasks.

1. Convert height which is in inches to millimeter
2. Convert weight which is in pounds to kilograms

```
def transformation(data):
    # Convert height from inches to millimeter
    data.height = data.height.astype(float)
    data['height'] = round(data.height * 0.0254, 2)

    # Converting weight from pounds to kilogram
    data.weight = data.weight.astype(float)
    data['weight'] = round(data.weight * 0.45359237, 2)

    return data
```

Loading Data:

As Extraction and Transformation are done successfully, now we have to load the data into csv file.

```
def load(targetfile, data_to_load):
    data_to_load.to_csv(targetfile)
```

Logging:

- I have used logging by importing the logging package in python.
- We can access logging package functionalities by using a logger.
- Logger allows us to set the format in which the logs will generate.

```
def log(message):
    timestamp_format = '%Y-%h-%d-%H:%M:%S' # Year-Monthname-Day-Hour-Minute-Second
    now = datetime.now() # Get the current time
    timestamp  = now.strftime(timestamp_format)
    with open("logfile.txt", 'a') as file:
        file.write(timestamp + ',' + message + '\n')
```

Running ETL Process:

```
log("ETL Job Started")
```

```
# Extracting Data

log("Extract Phase Started:")
extracted_data = extract()
log("Extract Phase Ended:")
extracted_data
```

	name	height	weight
0	alex	65.78	112.99
1	ajay	71.52	136.49
2	alice	69.4	153.03
3	ravi	68.22	142.34
4	joe	67.79	144.3
5	alex	65.78	112.99
6	ajay	71.52	136.49
7	alice	69.4	153.03
8	ravi	68.22	142.34
9	joe	67.79	144.3
10	alex	65.78	112.99
11	ajay	71.52	136.49
12	alice	69.4	153.03
13	ravi	68.22	142.34
14	joe	67.79	144.3
15	jack	68.7	123.3
16	tom	69.8	141.49
17	tracy	70.01	136.46
18	john	67.9	112.37
19	jack	68.7	123.3
20	tom	69.8	141.49
21	tracy	70.01	136.46
22	john	67.9	112.37
23	jack	68.7	123.3
24	tom	69.8	141.49
25	tracy	70.01	136.46
26	john	67.9	112.37
27	simon	67.90	112.37
28	simon	67.90	112.37
29	simon	67.90	112.37

```
# Transformation Data

log("Transformation Phase Started:")
transformed_data = transformation(extracted_data)
log("Transformation Phase Ended:")
transformed_data
```

	name	height	weight
0	alex	1.67	51.25
1	ajay	1.82	61.91
2	alice	1.76	69.41
3	ravi	1.73	64.56
4	joe	1.72	65.45
5	alex	1.67	51.25
6	ajay	1.82	61.91
7	alice	1.76	69.41
8	ravi	1.73	64.56
9	joe	1.72	65.45
10	alex	1.67	51.25
11	ajay	1.82	61.91
12	alice	1.76	69.41
13	ravi	1.73	64.56
14	joe	1.72	65.45
15	jack	1.74	55.93
16	tom	1.77	64.18
17	tracy	1.78	61.90
18	john	1.72	50.97
19	jack	1.74	55.93
20	tom	1.77	64.18
21	tracy	1.78	61.90
22	john	1.72	50.97
23	jack	1.74	55.93
24	tom	1.77	64.18
25	tracy	1.78	61.90
26	john	1.72	50.97
27	simon	1.72	50.97
28	simon	1.72	50.97
29	simon	1.72	50.97

```
# Loading Data

log("Loading Phase Started:")
```

```
load(targetfile, transformed_data)
log("Loading Phase Ended:")
```

```
log("ETL Job Ended")
```

So this was the simple ETL implementation thats shows how data is extracted from a web source, transform into the usable format, and then loading the data (in this case, a csv file).

EXAMPLE: 2

LET'S PERFORM ETL ON CAR DEALERSHIP DATA:

ABOUT THE DATA:

The file `dealership_data` contains CSV, JSON, and XML files for used car data which contain features named `car_model`, `year_of_manufacture`, `price`, and `fuel`.

Downloading the File:

```
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
```

```
--2022-09-07 14:50:27--  https://cf-courses-data.s3.us.cloud-object-storage.appdoma
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-da
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-course
HTTP request sent, awaiting response... 200 OK
Length: 4249 (4.1K) [application/zip]
Saving to: ‘datasource.zip’

datasource.zip        0%[                    ]       0  --.-KB/s               data

2022-09-07 14:50:27 (1.05 GB/s) - ‘datasource.zip’ saved [4249/4249]
```

Unzip the Files:

```
!unzip datasource.zip -d dealership_data
```

```
Archive:  datasource.zip
  inflating: dealership_data/used_car_prices1.csv
  inflating: dealership_data/used_car_prices2.csv
  inflating: dealership_data/used_car_prices3.csv
  inflating: dealership_data/used_car_prices1.json
  inflating: dealership_data/used_car_prices2.json
  inflating: dealership_data/used_car_prices3.json
  inflating: dealership_data/used_car_prices1.xml
  inflating: dealership_data/used_car_prices2.xml
  inflating: dealership_data/used_car_prices3.xml
```

Set Paths (Optional):

```
tmpfile    = "dealership_temp.tmp"               # file used to store all extracted d
logfile    = "dealership_logfile.txt"            # all event logs will be stored in t
targetfile = "dealership_transformed_data.csv"   # file where transformed data is sto
```

Extract

```
# CSV Extract Function
def extract_csv_data(file):
    """
    Extracts the data from a CSV file and returns a dataframe.
    """
    df = pd.read_csv(file)
    return df
```

```
# JSON extract function
def extract_json_data(file):
    """
    Extracts the data from a JSON file and returns a dataframe.
    """
    df = pd.read_json(file, lines=True)
    return df
```

```
# Add the XML extract function below, it is the same as the xml extract function abov
def extract_xml_data(file):
    """Extract data from XML file"""

    df = pd.DataFrame(columns=["car_model", "year_of_manufacture", "price", "fuel"])
    tree = ET.parse(file)
    root = tree.getroot()
    for i in root:
        car_model = i.find("car_model").text
        year_of_manufacture = int(i.find("year_of_manufacture").text)
        price = i.find("price").text
        fuel = i.find("fuel").text
        # Now Append the extracted data to df
        df = df.append({"car_model": car_model, "year_of_manufacture": year_of_manufa
        return df
```

Extract Function:

It will extract data from each file and append it to Pandas Dataframe:

```
def extract():
    extracted_data = pd.DataFrame(columns=['car_model','year_of_manufacture','price',

    #process all csv files
    for csvfile in glob.glob("dealership_data/*.csv"):
        extracted_data = extracted_data.append(extract_csv_data(csvfile), ignore_inde

    #process all json files
    for jsonfile in glob.glob("dealership_data/*.json"):
        extracted_data = extracted_data.append(extract_json_data(jsonfile), ignore_in

    #process all xml files
    for xmlfile in glob.glob("dealership_data/*.xml"):
        extracted_data = extracted_data.append(extract_xml_data(xmlfile), ignore_inde

    return extracted_data
```

Transform:

```
def transform(data):
    """ This function return data after applying transformation rules that need to be
    data.price = data.price.astype(float)
    data['price'] = round(data.price, 2)
    return data
```

Loading:

It will load the data in the required format:

```
# Load Function
def load(targetfile, data_to_load):
    data_to_load.to_csv(targetfile)
```

Logging:

```
# Log function
def log(message):
    timestamp_format = '%H:%M:%S-%h-%d-%Y'
    now = datetime.now()
    timestamp = now.strftime(timestamp_format)
    with open("dealership_logfile.txt", 'a') as f:
        f.write(timestamp+ ', '+ message+ '\n')
```

Running ETL Process:

```
# Log that you have started the ETL process
log("ETL JOB STARTED:")

# Log that you have started the Extract step
log("Extract Phase Started")

# Call the Extract function
extracted_data = extract()
```

```
# Log that you have completed the Extract step
log("Extract Phase is Ended")

# Log that you have started the Transform step
log("Transformation Phase is Started")

# Call the Transform function
transformed_data = transform(extracted_data)
# Log that you have completed the Transform step
log("Transformation Phase is Ended")

# Log that you have started the Load step
log("Loading Phase is Started")
# Call the Load function
load(targetfile, transformed_data)
# Log that you have completed the Load step
log("Loading Phase is Ended")

# Log that you have completed the ETL process
log("ETL JOB IS ENDED")
```

Author:

Umer Farooq