Logging Module Integration

Logging in Python is a built-in module that provides a way to log events and messages in a running program. It allows developers to record and store information about a program's behavior during runtime, which can be used for debugging, performance monitoring, and auditing purposes. The logging module provides functions and classes that enable developers to create loggers, handlers, filters, and formatters to customize the logging output based on their specific needs. By using the logging module, developers can capture valuable information about their programs' activities, such as error messages, warning messages, debug messages, and other types of events, and save them to a file, display them on the console, or send them to a remote server for analysis.

Logging levels in Python define the severity of an event or message being logged. The logging module defines six standard logging levels and each level is having a standard numeric value:

Level	Numeric Value	Usage
CRITICAL	50	A very serious error, indicating that the program itself may be unable to continue running.
ERROR	40	Due to a more serious problem, the software has not been able to perform some function.
WARNING	30	An indication that something unexpected or undesirable happened, or indicative of some problem in the near future (e.g., ‘disk space low’). The software is still working as expected.
INFO	20	Confirmation that things are working as expected.
DEBUG	10	Detailed information, typically of interest only when diagnosing problems.
NOSET	0	The lowest possible level, used when the logger is not configured to perform any level checking.

When a log message is generated, it is assigned a level based on the severity of the event being logged. The logging module allows developers to set the minimum logging level that should be displayed or stored, which can be used to filter out less important messages and focus on the critical ones. For example, if a developer sets the logging level to WARNING, only messages with a level of WARNING, ERROR, and CRITICAL will be displayed or stored, while messages with a lower level of severity, such as DEBUG and INFO, will be ignored.

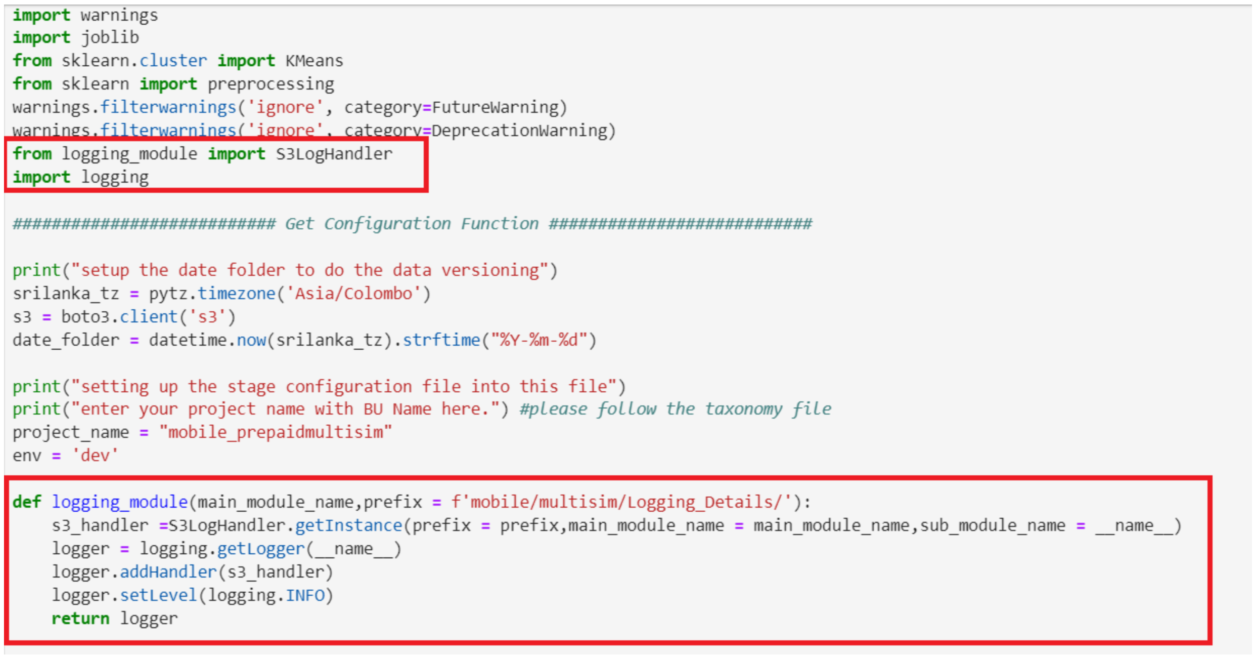
logging_module.py- This module is customized to our Sagemaker and S3 environment. It facilitates logging and a way to store log records in a bucket on Amazon S3(dlk-cloud-tier-10-preprocessed-ml-dev), which will be useful for archiving, monitoring, or further analysis of log .Also the logging records above the WARNING level are automatically printed in the cloud watch.

Once the necessary logs are created, it will be sent to the **dlk-cloud-tier-10-preprocessed-ml-dev** S3 bucket as a new file with a key prefix that includes the current date and time. The module is implemented using the singleton design pattern, which ensures that all subsequent calls to the class return the same instance. So that, it restricts creating a new log file in the S3 bucket in every time the code runs, and the logs will be updated in the same log file.

Now let's see how the logging function can be integrated to our source code by using the developed module.

Place the **logging_module.py** inside the same directory which your code file exists. You can find the module codes from here. Module Library

Then insert below highlighted codes to your code file.



```python
import warnings
import joblib
from sklearn.cluster import KMeans
from sklearn import preprocessing
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)
from logging_module import S3LogHandler
import logging

########################### Get Configuration Function ###########################

print("setup the date folder to do the data versioning")
srilanka_tz = pytz.timezone('Asia/Colombo')
s3 = boto3.client('s3')
date_folder = datetime.now(srilanka_tz).strftime("%Y-%m-%d")

print("setting up the stage configuration file into this file")
print("enter your project name with BU Name here.") #please follow the taxonomy file
project_name = "mobile_prepaidmultisim"
env = 'dev'

def logging_module(main_module_name,prefix = f'mobile/multisim/Logging_Details/'):
    s3_handler =S3LogHandler.getInstance(prefix = prefix,main_module_name = main_module_name,sub_module_name = __name__)
    logger = logging.getLogger(__name__)
    logger.addHandler(s3_handler)
    logger.setLevel(logging.INFO)
    return logger
```

You can give the prefix as in the folder path you want to save the log file.Since the prefix is given as **'mobile/multisim/Logging_Details/',** log file will be created inside that particular path inside the **dlk-cloud-tier-10-preprocessed-ml-dev.**

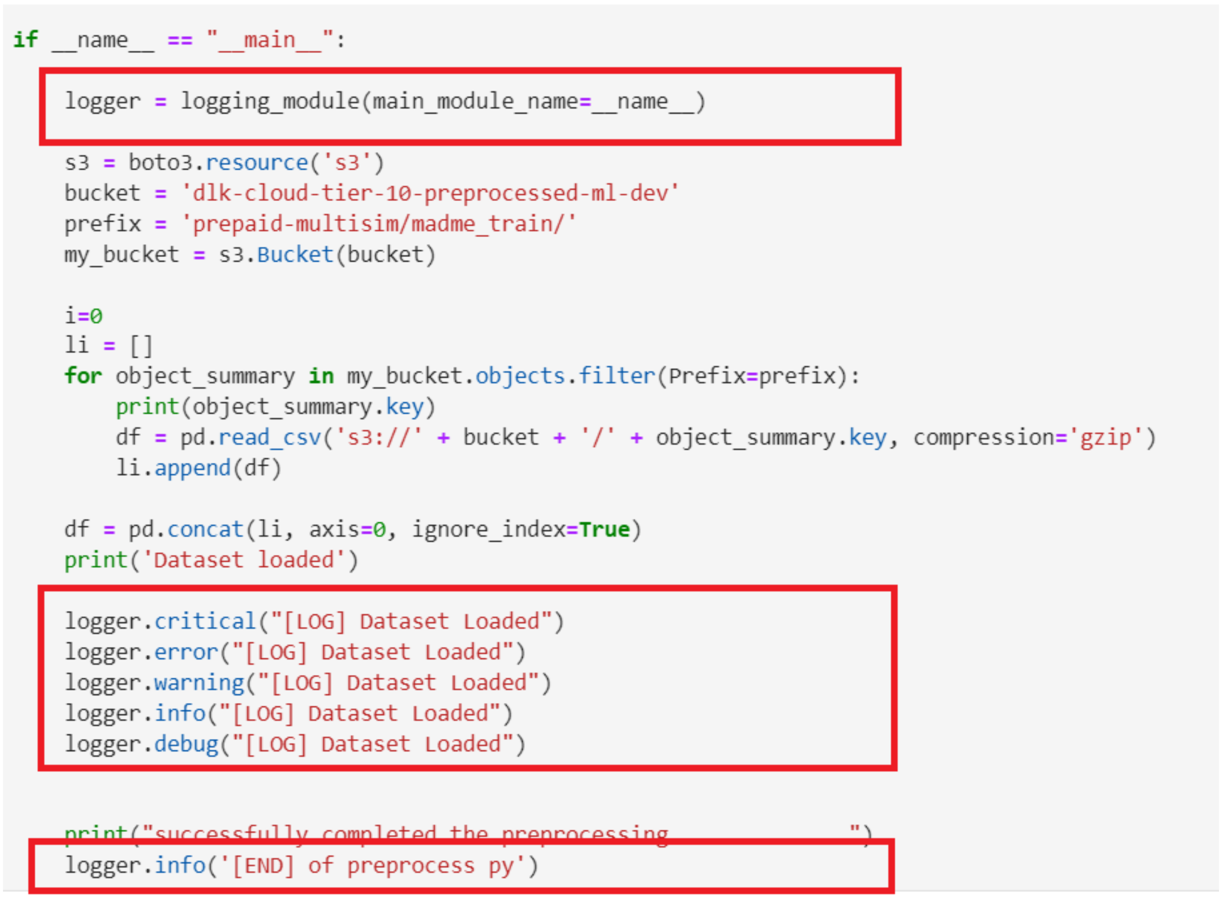
Here according to **logger.setLevel(logging.INFO)** logging method is set to logging level of a logger object to INFO. As mentioned earlier, logging levels range from DEBUG (lowest level) to CRITICAL (highest level). By setting the logging level to INFO, the logger object will handle log messages with a severity of INFO and higher (WARNING, ERROR, and CRITICAL). Log messages with a lower severity (DEBUG) will be ignored.

Now you can add below higlighted codes inside the main function.

Here you can specify the main_module_name or it will automatically pick the code file name as the main_module_name.

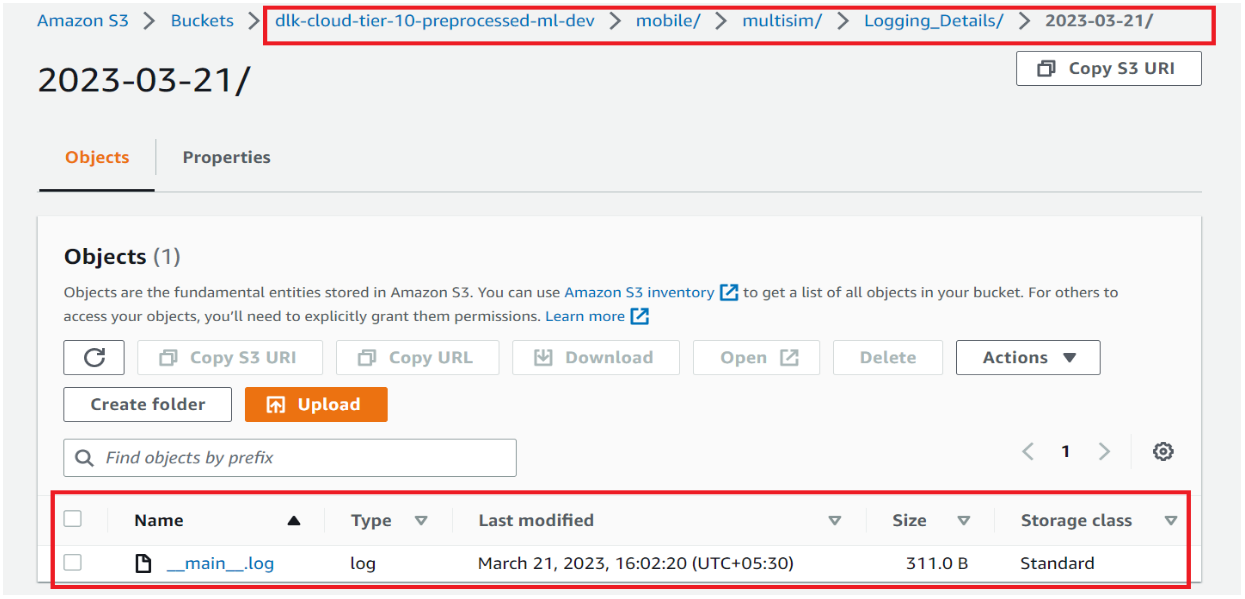
And logger function is called five times (critical,error,warning,info,debug) , just to demonstrate how the each logging level will work.

logger.info('[END] of preprocess py'), message being logged indicates the end of the "preprocess.py" script.



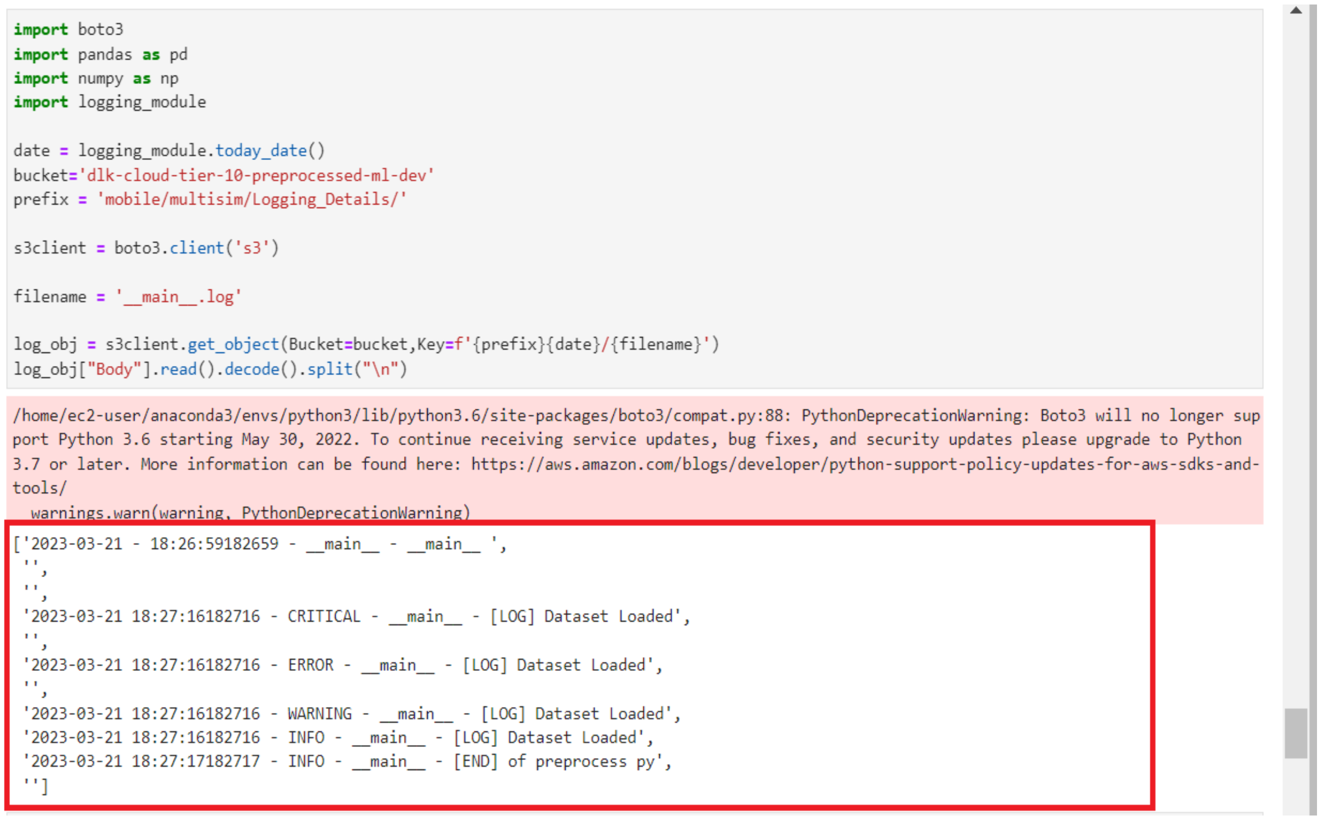
```python
if __name__ == "__main__":
    logger = logging_module(main_module_name=__name__)

    s3 = boto3.resource('s3')
    bucket = 'dlk-cloud-tier-10-preprocessed-ml-dev'
    prefix = 'prepaid-multisim/madme_train/'
    my_bucket = s3.Bucket(bucket)

    i=0
    li = []
    for object_summary in my_bucket.objects.filter(Prefix=prefix):
        print(object_summary.key)
        df = pd.read_csv('s3://' + bucket + '/' + object_summary.key, compression='gzip')
        li.append(df)

    df = pd.concat(li, axis=0, ignore_index=True)
    print('Dataset loaded')

    logger.critical("[LOG] Dataset Loaded")
    logger.error("[LOG] Dataset Loaded")
    logger.warning("[LOG] Dataset Loaded")
    logger.info("[LOG] Dataset Loaded")
    logger.debug("[LOG] Dataset Loaded")


    print("successfully completed the preprocessing                ")
    logger.info('[END] of preprocess py')
```

Once you execute the code, you can find the log file inside the folder path specified as the prefix earlier, inside the dlk-cloud-tier-10-preprocessed-ml-dev S3 bucket.


Amazon S3 > Buckets > dlk-cloud-tier-10-preprocessed-ml-dev > mobile/ > multisim/ > Logging_Details/ > 2023-03-21/
2023-03-21/
Copy S3 URI
Objects
Properties
Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
1

	Name	Type	Last modified	Size	Storage class
	__main__.log	log	March 21, 2023, 16:02:20 (UTC+05:30)	311.0 B	Standard

You can view the log file as below.

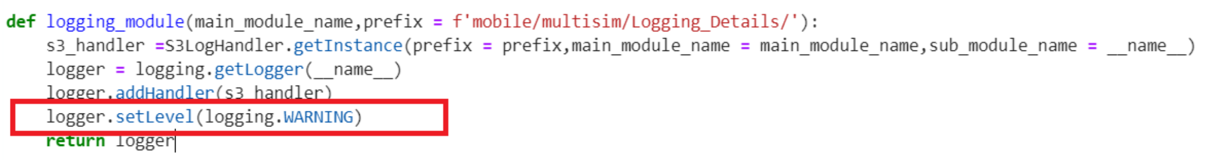


```
import boto3
import pandas as pd
import numpy as np
import logging_module

date = logging_module.today_date()
bucket='dlk-cloud-tier-10-preprocessed-ml-dev'
prefix = 'mobile/multisim/Logging_Details/'

s3client = boto3.client('s3')

filename = '__main__.log'

log_obj = s3client.get_object(Bucket=bucket,Key=f'{prefix}{date}/{filename}')
log_obj["Body"].read().decode().split("\n")
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/boto3/compat.py:88: PythonDeprecationWarning: Boto3 will no longer sup
port Python 3.6 starting May 30, 2022. To continue receiving service updates, bug fixes, and security updates please upgrade to Python
3.7 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-
tools/
  warnings.warn(warning, PythonDeprecationWarning)
['2023-03-21 - 18:26:59182659 - __main__ - __main__ ',
 '',
 '',
 '2023-03-21 18:27:16182716 - CRITICAL - __main__ - [LOG] Dataset Loaded',
 '',
 '2023-03-21 18:27:16182716 - ERROR - __main__ - [LOG] Dataset Loaded',
 '',
 '2023-03-21 18:27:16182716 - WARNING - __main__ - [LOG] Dataset Loaded',
 '2023-03-21 18:27:16182716 - INFO - __main__ - [LOG] Dataset Loaded',
 '2023-03-21 18:27:17182717 - INFO - __main__ - [END] of preprocess py',
 '']
```

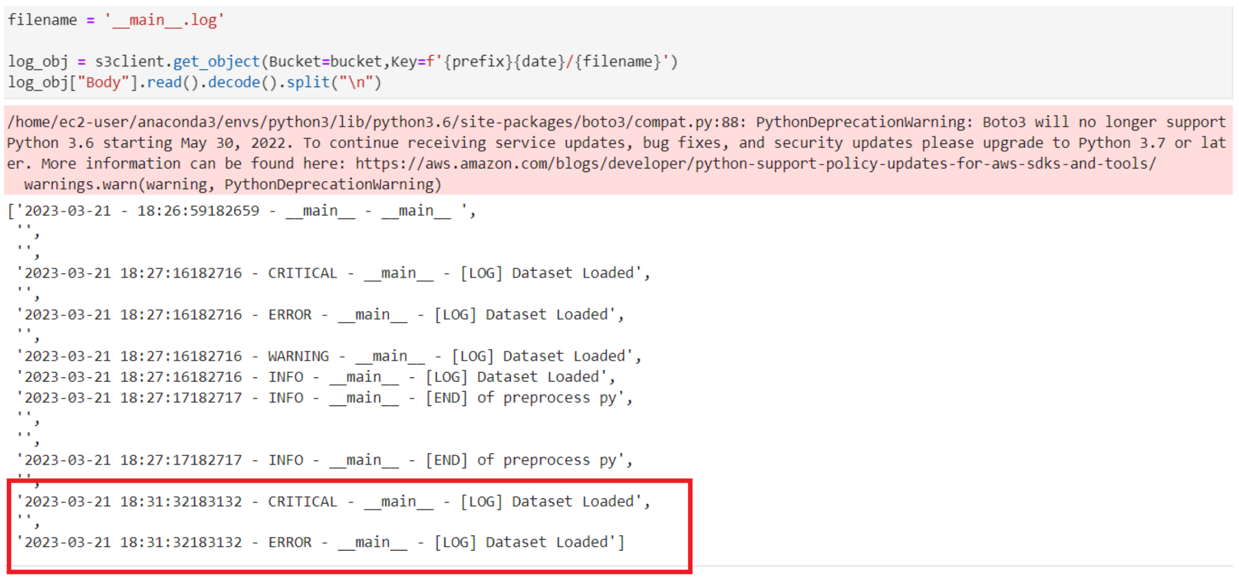
As you can see, since the logger is set to info level, this contains only the log messages with a severity of **INFO and higher (WARNING, ERROR, and CRITICAL)**.

Now lets set the logger level to warning as below and check for the logs.



```
def logging_module(main_module_name,prefix = f'mobile/multisim/Logging_Details/'):
    s3_handler =S3LogHandler.getInstance(prefix = prefix,main_module_name = main_module_name,sub_module_name = __name__)
    logger = logging.getLogger(__name__)
    logger.addHandler(s3_handler)
    logger.setLevel(logging.WARNING)
    return logger
```

As we can see the below, when the logger is set to **WARNING** level it does not show up the **INFO** Level.



```
filename = '__main__.log'

log_obj = s3client.get_object(Bucket=bucket,Key=f'{prefix}{date}/{filename}')
log_obj["Body"].read().decode().split("\n")
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/boto3/compat.py:88: PythonDeprecationWarning: Boto3 will no longer support
Python 3.6 starting May 30, 2022. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.7 or lat
er. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
['2023-03-21 - 18:26:59182659 - __main__ - __main__ ',
 '',
 '',
 '2023-03-21 18:27:16182716 - CRITICAL - __main__ - [LOG] Dataset Loaded',
 '',
 '2023-03-21 18:27:16182716 - ERROR - __main__ - [LOG] Dataset Loaded',
 '',
 '2023-03-21 18:27:16182716 - WARNING - __main__ - [LOG] Dataset Loaded',
 '2023-03-21 18:27:16182716 - INFO - __main__ - [LOG] Dataset Loaded',
 '2023-03-21 18:27:17182717 - INFO - __main__ - [END] of preprocess py',
 '',
 '',
 '2023-03-21 18:27:17182717 - INFO - __main__ - [END] of preprocess py',
 '',
 '2023-03-21 18:31:32183132 - CRITICAL - __main__ - [LOG] Dataset Loaded',
 '',
 '2023-03-21 18:31:32183132 - ERROR - __main__ - [LOG] Dataset Loaded']
```

In order to print the logs in the cloud watch , logging level should be set to WARNING or high level.

Module Development Dependencies

general_module is imported inside the logging_module.py, which consists of several functions that has been developed for various objectives as below.

Function	Input	Output	Objective
today_date()	No input	Current date	Get current date
exe_id()	No input	Current timestamp	Returns current timestamp as execution id. (Ex :`20230220T085700`)
now_time()	No input	Current time	Returns current time
convert_to_heap (offer_list_dict)	offer_list_dict : list of offers	No output	Converts offer list into a priority queue(heap)

week_of_month()	No input	current week	Returns the week corresponding to the current date
day_of_week()	No input	current day of week	Returns day of week corresponding to the current date
create_snowflake_connection (secret_name,region)	secret_name : Name of the secret key region_name : Name of region	Creates the snowflake connection and returns the object	Creates the snowflake connection and returns the object
get_config_file()	No input	Configurations	Gets all the configurations in the config file

Snowflake_module is used inside the general_module which facilitates for below functions.

Function	**Input**	**Output**	**Objective**
get_secret (secret_name,region_name)	secret_name : Name of the secret key region_name : Name of region	Return secret key	Returns the snowflake secret key values.
createSnowflakeConnection (snowflake_secret)	snowflake_secret : dict object consisting of secret_name and region to create the connection.	snowflake connector (Snowflake object which is used to connect to snowflake warehouses and databases.)	Creates the connection to snowflake using the secret name and region.