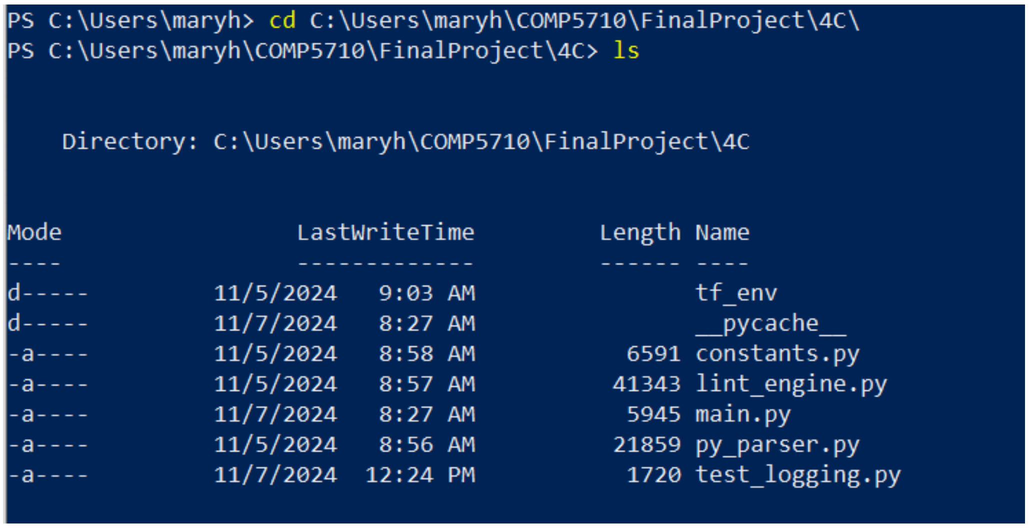
Group Project
4C

Report: 4C

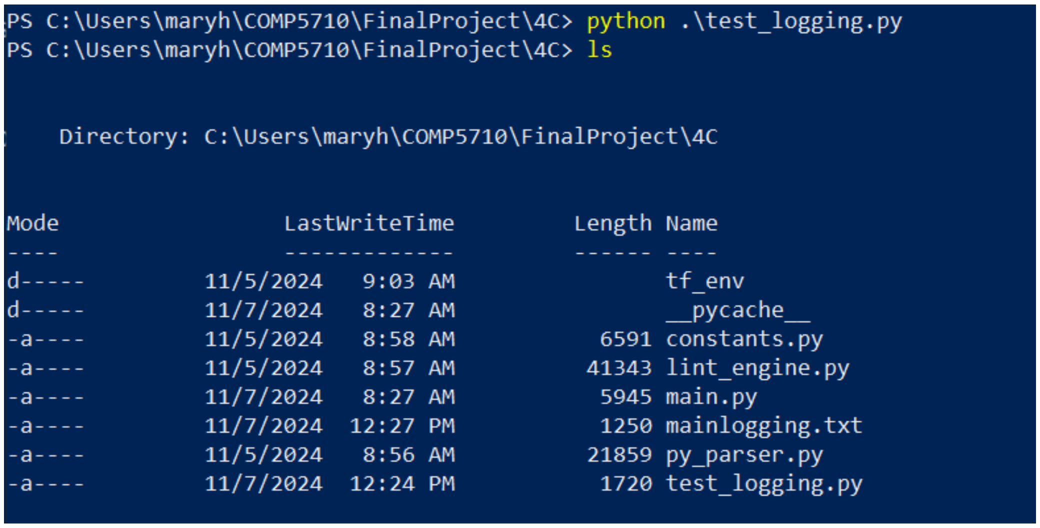
In this part of the group project we took the FAME-ML files and integrated forensics and modified 5 Python methods. We added logging to the main.py file that was given and then we created a Python test file, named test_logging.py, to test the logging. Both of these files will be given at the end of this report.

The first thing we did was edit the main.py for implementing logging, and created a test file to test the logs. Below is a screenshot that shows that we have all the files saved to one folder.



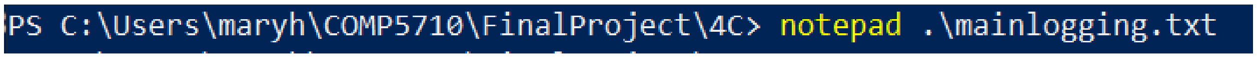
```
PS C:\Users\maryh> cd C:\Users\maryh\COMP5710\FinalProject\4C\
PS C:\Users\maryh\COMP5710\FinalProject\4C> ls


    Directory: C:\Users\maryh\COMP5710\FinalProject\4C


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          11/5/2024   9:03 AM                tf_env
d-----          11/7/2024   8:27 AM                __pycache__
-a----          11/5/2024   8:58 AM           6591 constants.py
-a----          11/5/2024   8:57 AM          41343 lint_engine.py
-a----          11/7/2024   8:27 AM           5945 main.py
-a----          11/5/2024   8:56 AM          21859 py_parser.py
-a----          11/7/2024  12:24 PM           1720 test_logging.py
```

Next we took the test_logging.py code and ran it. Once the python ran it created a mainlogging.txt file for the logging. Below is a screenshot of running the python script, and seeing that it created a text file within the folder.



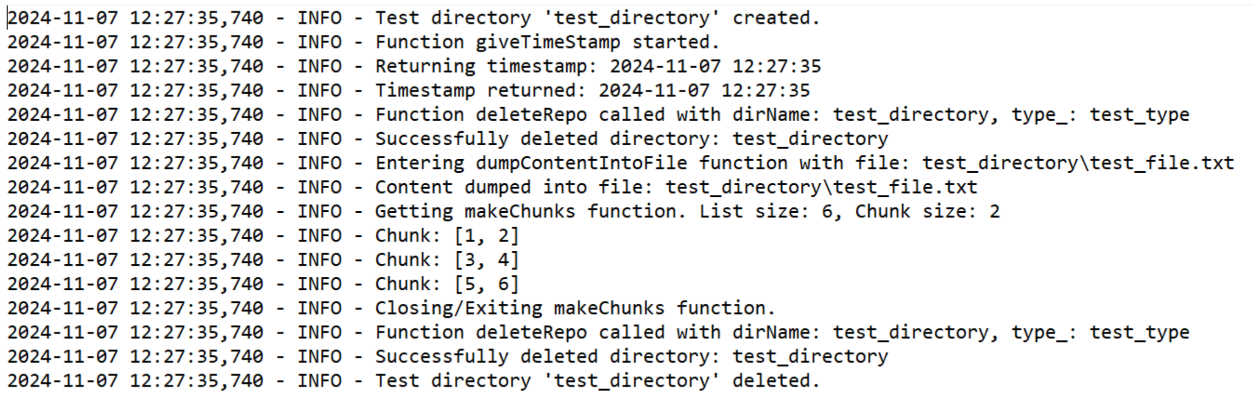
```
PS C:\Users\maryh\COMP5710\FinalProject\4C> python .\test_logging.py
PS C:\Users\maryh\COMP5710\FinalProject\4C> ls


    Directory: C:\Users\maryh\COMP5710\FinalProject\4C


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          11/5/2024   9:03 AM                tf_env
d-----          11/7/2024   8:27 AM                __pycache__
-a----          11/5/2024   8:58 AM           6591 constants.py
-a----          11/5/2024   8:57 AM          41343 lint_engine.py
-a----          11/7/2024   8:27 AM           5945 main.py
-a----          11/7/2024  12:27 PM           1250 mainlogging.txt
-a----          11/5/2024   8:56 AM          21859 py_parser.py
-a----          11/7/2024  12:24 PM           1720 test_logging.py
```

Now we opened the text file that was created in the notepad. The command used was screenshotted and placed below.



```
PS C:\Users\maryh\COMP5710\FinalProject\4C> notepad .\mainlogging.txt
```

After using the command the mainlogging.txt opened up in the notepad. Below is a screenshot of what was given when we opened the logging text file. This shows that the logs that were implemented were implemented correctly and executed successfully.



```
2024-11-07 12:27:35,740 - INFO - Test directory 'test_directory' created.
2024-11-07 12:27:35,740 - INFO - Function giveTimeStamp started.
2024-11-07 12:27:35,740 - INFO - Returning timestamp: 2024-11-07 12:27:35
2024-11-07 12:27:35,740 - INFO - Timestamp returned: 2024-11-07 12:27:35
2024-11-07 12:27:35,740 - INFO - Function deleteRepo called with dirName: test_directory, type_: test_type
2024-11-07 12:27:35,740 - INFO - Successfully deleted directory: test_directory
2024-11-07 12:27:35,740 - INFO - Entering dumpContentIntoFile function with file: test_directory\test_file.txt
2024-11-07 12:27:35,740 - INFO - Content dumped into file: test_directory\test_file.txt
2024-11-07 12:27:35,740 - INFO - Getting makeChunks function. List size: 6, Chunk size: 2
2024-11-07 12:27:35,740 - INFO - Chunk: [1, 2]
2024-11-07 12:27:35,740 - INFO - Chunk: [3, 4]
2024-11-07 12:27:35,740 - INFO - Chunk: [5, 6]
2024-11-07 12:27:35,740 - INFO - Closing/Exiting makeChunks function.
2024-11-07 12:27:35,740 - INFO - Function deleteRepo called with dirName: test_directory, type_: test_type
2024-11-07 12:27:35,740 - INFO - Successfully deleted directory: test_directory
2024-11-07 12:27:35,740 - INFO - Test directory 'test_directory' deleted.
```

This part of the group project we successfully executed and gave screenshots to support our work that we did for section 4C of the group project. Below are the main.py code used and the test_logging.py code that was used for this part of the group project.

main.py(updated for the project)

```
import os
import pandas as pd
import numpy as np
import csv
import time
from datetime import datetime
import subprocess
import shutil
from git import Repo
from git import exc
import logging

#logging for main.py
logging.basicConfig(filename='mainlogging.txt', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

def giveTimeStamp():
  logging.info("Function giveTimeStamp started.")
  tsObj = time.time()
  strToret = datetime.fromtimestamp(tsObj).strftime('%Y-%m-%d %H:%M:%S')
  #logging to return the given timestamp
  logging.info("Returning timestamp: {}".format(strToret))
  return strToret


def deleteRepo(dirName, type_):
  logging.info(f"Function deleteRepo called with dirName: {dirName}, type_: {type_}")
```

```
    try:
        if os.path.exists(dirName):
            shutil.rmtree(dirName)
            logging.info(f"Successfully deleted directory: {dirName}")
        else:
            logging.warning(f"Directory not found: {dirName}")

    except OSError as e:
        logging.error(f"Error deleting directory: {dirName}. Error: {e}")
        print("Failed to delete directory, will try manually.")


def dumpContentIntoFile(strP, fileP):
    #log information about dump
    logging.info("Entering dumpContentIntoFile function with file: {}".format(fileP))
    with open(fileP, 'w') as fileToWrite:
        fileToWrite.write(strP)
                #getting all content
        logging.info("Content dumped into file: {}".format(fileP))
        return str(os.stat(fileP).st_size)


def makeChunks(the_list, size_):
    #get info about the make chunks
    logging.info("Getting makeChunks function. List size: {}, Chunk size: {}".format(len(the_list), size_))
    for i in range(0, len(the_list), size_):
        yield the_list[i:i + size_]
    #clsoing makeChunks function
    logging.info("Closing/Exiting makeChunks function.")


def cloneRepo(repo_name, target_dir):
    #getting info about clonerepo
    logging.info("Getting cloneRepo function. Cloning repo: {}, to directory: {}".format(repo_name, target_dir))
    cmd_ = "git clone " + repo_name + " " + target_dir
    try:
        subprocess.check_output(['bash','-c', cmd_])
                #success logging
        logging.info("Successfully cloned repo: {}".format(repo_name))
    #set error to e
    except subprocess.CalledProcessError as e:
                #get errors
        logging.error("Error cloning repo: {}. Error: {}".format(repo_name, e))
        print('Skipping this repo ... trouble cloning repo:', repo_name)


def getDevEmailForCommit(repo_path_param, hash_):
    #getting the info about email for commit
    logging.info("Entering getDevEmailForCommit function with repo_path: {}, commit hash: {}".format(repo_path_param, hash_))
    author_emails = []
    cdCommand = "cd " + repo_path_param + " ; "
    commitCountCmd = " git log --format='%ae'" + hash_ + "^!"
    command2Run = cdCommand + commitCountCmd

    try:
        author_emails = str(subprocess.check_output(['bash','-c', command2Run]))
        author_emails = author_emails.split('\n')
        author_emails = [x_.replace(hash_, '') for x_ in author_emails if x_ != '\n' and '@' in x_]
        author_emails = [x_.replace('^', '') for x_ in author_emails if x_ != '\n' and '@' in x_]
        author_emails = [x_.replace('!', '') for x_ in author_emails if x_ != '\n' and '@' in x_]
        author_emails = [x_.replace('\\n', ',') for x_ in author_emails if x_ != '\n' and '@' in x_]
        author_emails = author_emails[0].split(',')
        author_emails = [x_ for x_ in author_emails if len(x_) > 3]
        author_emails = list(np.unique(author_emails))
                #found some emails
        logging.info("Found emails: {}".format(author_emails))
    #set error to e
    except subprocess.CalledProcessError as e:
                #get errors from function
        logging.error("Get error commit details for hash: {}. Error: {}".format(hash_, e))
    return author_emails

def getDevDayCount(full_path_to_repo, branchName='master', explore=1000):
    #getting info
    logging.info("Getting getDevDayCount function for repo: {}".format(full_path_to_repo))
    repo_emails = []
    all_commits = []
    all_time_list = []

    if os.path.exists(full_path_to_repo):
        repo_ = Repo(full_path_to_repo)
        try:
            all_commits = list(repo_.iter_commits(branchName))
                    #success
            logging.info("Total commits found: {}".format(len(all_commits)))
                #setting error as e
        except exc.GitCommandError as e:
                    #get the errors
            logging.error("Error accessing repository: {}. Error: {}".format(full_path_to_repo, e))

        for commit_ in all_commits:
            commit_hash = commit_.hexsha
            emails = getDevEmailForCommit(full_path_to_repo, commit_hash)
            repo_emails = repo_emails + emails

            timestamp_commit = commit_.committed_datetime
            str_time_commit = timestamp_commit.strftime('%Y-%m-%d')
            all_time_list.append(str_time_commit)

    all_day_list = [datetime(int(x_.split('-')[0]), int(x_.split('-')[1]), int(x_.split('-')[2]), 12, 30) for x_ in all_time_list]
    try:
        min_day = min(all_day_list)
        max_day = max(all_day_list)
        ds_life_days = days_between(min_day, max_day)
                #Repo life span
        logging.info("Repo life span: {} days.".format(ds_life_days))
    #set error as e
    except (ValueError, TypeError) as e:
        ds_life_days = 0
                #get the errors
        logging.error("Error calculating repo life span. Error: {}".format(e))

    ds_life_months = round(float(ds_life_days)/float(30), 5)
    #format life span
    logging.info("Repo life span in months: {}".format(ds_life_months))

    return len(repo_emails), len(all_commits), ds_life_days, ds_life_months
```

test_logging.py(made for the project)

```
import os  # Ensure os is imported
import logging
from main import giveTimeStamp, deleteRepo, dumpContentIntoFile, makeChunks, cloneRepo  # Assuming your code is in main.py

def test_logging():
    test_dir = "test_directory"

    # Ensure the directory exists before creating files inside it
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)
        logging.info(f"Test directory '{test_dir}' created.")

    # Call some of the functions from main.py to see if logging works
    timestamp = giveTimeStamp()
    logging.info(f"Timestamp returned: {timestamp}")

    # Test deleteRepo function
    deleteRepo(test_dir, 'test_type')

    # Test dumpContentIntoFile function
    content = "This is a test."
    file_path = os.path.join(test_dir, 'test_file.txt')
    # Check if the directory exists before trying to write the file
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)  # Create the directory if it doesn't exist

    dumpContentIntoFile(content, file_path)

    # Test makeChunks function
    sample_list = [1, 2, 3, 4, 5, 6]
    chunk_size = 2
    for chunk in makeChunks(sample_list, chunk_size):
        logging.info(f"Chunk: {chunk}")

    # Test cloneRepo function (provide a valid repo URL if needed)
    # Example: cloneRepo("https://github.com/yourrepo.git", "clone_test_dir")

    # Optionally, delete the directory after the test
    deleteRepo(test_dir, 'test_type')
    logging.info(f"Test directory '{test_dir}' deleted.")

if __name__ == "__main__":
    logging.basicConfig(filename='test_log.txt', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
    test_logging()
```