IT-314 Lab 5

Name - Dev Patel
ID - 202001014
Group 1 : Bhargav Sir
Date - 15/03/2023

Qs :
Select the tool of your choice. Select a git repository, use the selected tool and analyze the files from the selected repository. Submit the tool output and understanding of the errors.

Ans : Here I am using flake8 for python language for this particular assignment.

Error description table:

Code	Example Message
F401	`module` imported but unused
F402	import `module` from line `N` shadowed by loop variable
F403	'from `module` import *' used; unable to detect undefined names
F404	future import(s) `name` after other statements
F405	`name` may be undefined, or defined from star imports: `module`
F406	'from `module` import *' only allowed at module level

F407	an undefined `__future__` feature name was imported
F501	invalid `%` format literal
F502	`%` format expected mapping but got sequence
F503	`%` format expected sequence but got mapping

https://flake8.pycqa.org/en/latest/user/error-codes.html#

Error codes

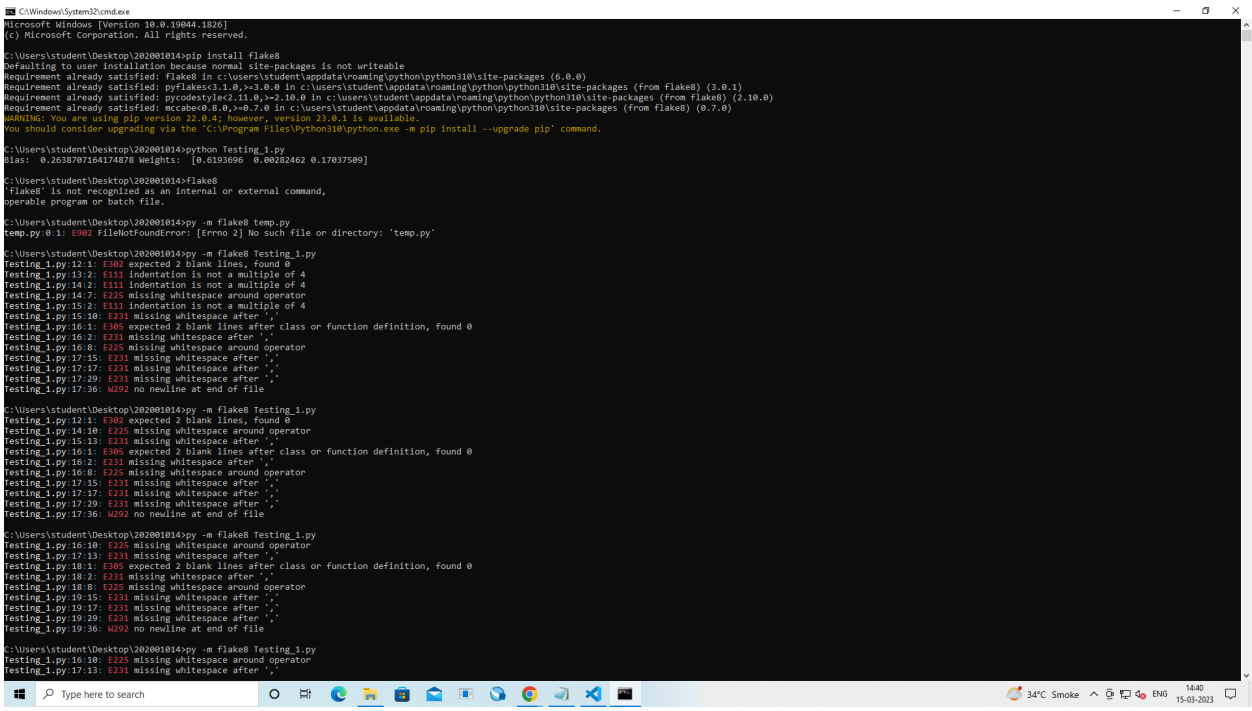
Error code	Description
ECE001	Expression is too complex (X > Y)

This error shows that the algorithm's complexity is higher than a certain point then it shows error. But it’s not happening at that type.

1.Indentation Errors: Image of Code

```
# -*- coding: utf-8 -*-
"""Testing 1.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1vNlpCJOYPQNsZhk7al8SY4FmgFf_KnGl
"""

import random
import numpy as np
def initialize(dim):
 b = random.random()
 theta=np.random.rand(dim)
 return b,theta
b,theta=initialize(3)
print("Bias: ",b,"Weights: ",theta)
```

Error images:



Here, in the above image the classified error which has been shown are type of indentation.

expected 2 blank lines, found 0
indentation is not a multiple of 4
missing whitespace after ','
no newline at end of file
Solution for above error can be achieved using tab and general code indentation methods.

After resolving all the errors we will achieve this kind of result in flake command.

```
C:\Users\student\Desktop\202001014>py -m flake8 Testing_1.py
Testing_1.py:21:39: W292 no newline at end of file

C:\Users\student\Desktop\202001014>py -m flake8 Testing_1.py

C:\Users\student\Desktop\202001014>
```

Improved Code:

```
# -*- coding: utf-8 -*-
"""Testing 1.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1vNlpCJOYPQNsZhk7al8SY4FmgFf_KnGl
"""

import random
import numpy as np


def initialize(dim):
    b = random.random()
    theta = np.random.rand(dim)
    return b, theta


b, theta = initialize(3)
print("Bias: ", b, "Weights: ", theta)
```

2.Syntax Errors: Image of Code

testing_1.py 9+ testing_2.py 9+

C: > Users > student > Desktop > 202001014 > testing_2.py

```python
myfunction(x, y):
    return x + y

else:
    print("Hello!")

if mark >= 50
    print("You passed!")

if arriving:
    print("Hi!")
esle:
    print("Bye!")

if flag:
print("Flag is set!")
```

Image of errors:

```
C:\Users\student\Desktop\202001014>py -m flake8 testing_2.py
testing_2.py:1:18: E999 SyntaxError: invalid syntax
```

3.Variable and function naming errors:

```python
Count = 0
for number in range(10):
    count = count + number
print("The count is:", count)
```

Flake analysis:

```
C:\Users\student\Desktop\202001014>py -m flake8 testing_2.py
testing_2.py:3:13: F821 undefined name 'count'
testing_2.py:4:30: W292 no newline at end of file
```

4.Unused code:

```python
# unused function
def my_func():

    # unused local variables
    a = 5

    b=2
    c=b+10
    print(b,c)


for i in range(0, 5):
    print(i)
```

Flake analysis:

```
C:\Users\student\Desktop\202001014>py -m flake8 testing_2.py
testing_2.py:4:1: W191 indentation contains tabs
testing_2.py:5:1: W191 indentation contains tabs
testing_2.py:5:2: F841 local variable 'a' is assigned to but never used
testing_2.py:6:1: W191 indentation contains tabs
testing_2.py:6:1: W293 blank line contains whitespace
testing_2.py:7:1: W191 indentation contains tabs
testing_2.py:7:3: E225 missing whitespace around operator
testing_2.py:8:1: W191 indentation contains tabs
testing_2.py:8:3: E225 missing whitespace around operator
testing_2.py:9:1: W191 indentation contains tabs
testing_2.py:9:9: E231 missing whitespace after ','
testing_2.py:13:1: W191 indentation contains tabs

C:\Users\student\Desktop\202001014>
```

5.:Line is too long

```python
import argparse


import cv2
import zmq


from camera.Camera import Camera
from constants import PORT, SERVER_ADDRESS
from utils import image_to_string


class Streamer:


    def __init__(self, server_address=SERVER_ADDRESS, port=PORT):
        """
        Tries to connect to the StreamViewer with supplied server_address
and creates a socket for future use.


        :param server_address: Address of the computer on which the
StreamViewer is running, default is `localhost`
        :param port: Port which will be used for sending the stream
        """


        print("Connecting to ", server_address, "at", port)
        context = zmq.Context()
        self.footage_socket = context.socket(zmq.PUB)
        self.footage_socket.connect('tcp://' + server_address + ':' +
port)
        self.keep_running = True


    def start(self):
        """
        Starts sending the stream to the Viewer.
        Creates a camera, takes a image frame converts the frame to string
and sends the string across the network
```

```
        :return: None
        """
        print("Streaming Started...")
        camera = Camera()
        camera.start_capture()
        self.keep_running = True



        while self.footage_socket and self.keep_running:
            try:
                frame = camera.current_frame.read()  # grab the current
frame
                image_as_string = image_to_string(frame)
                self.footage_socket.send(image_as_string)



            except KeyboardInterrupt:
                cv2.destroyAllWindows()
                break
        print("Streaming Stopped!")
        cv2.destroyAllWindows()



    def stop(self):
        """
        Sets 'keep_running' to False to stop the running loop if running.
        :return: None
        """
        self.keep_running = False





def main():
    port = PORT
    server_address = SERVER_ADDRESS



    parser = argparse.ArgumentParser()
    parser.add_argument('-s', '--server',
```

```
                        help='IP Address of the server which you want to
connect to, default'
                                ' is ' + SERVER_ADDRESS,
                        required=True)
    parser.add_argument('-p', '--port',
                        help='The port which you want the Streaming Server
to use, default'
                                ' is ' + PORT, required=False)



    args = parser.parse_args()



    if args.port:
        port = args.port
    if args.server:
        server_address = args.server



    streamer = Streamer(server_address, port)
    streamer.start()




if __name__ == '__main__':
    main()
```

Flake8 code:

```
C:\Users\student\Desktop\202001014>py -m flake8 testing_2.py
testing_2.py:15:80: E501 line too long (110 > 79 characters)
testing_2.py:17:80: E501 line too long (115 > 79 characters)
testing_2.py:30:80: E501 line too long (114 > 79 characters)
testing_2.py:64:80: E501 line too long (93 > 79 characters)
testing_2.py:68:80: E501 line too long (91 > 79 characters)
testing_2.py:83:11: W292 no newline at end of file
```

Here, the code line is too long then the error shows:

Line too long (length > 79 characters)

6) Import error:

```
import torch
import torch.nn as nn

import argparse

from transformers import GPTNeoXForCausalLM

from transformers import AutoConfig, AutoTokenizer

from transformers.modeling_utils import no_init_weights
import os


def create_empty_gptneox(config):

    import torch
    import torch.nn as nn

    _reset_parameters_linear = nn.Linear.reset_parameters
    def dummy(*args, **kargs):
        pass
    nn.Linear.reset_parameters = dummy

    # 1. disable init for faster initialization
    # 2. avoid tie token embeddings with lm_head, as we train them
separately.
    with no_init_weights(_enable=True):
        model = GPTNeoXForCausalLM(config).eval()

    nn.Linear.reset_parameters = _reset_parameters_linear

    return model
```

```
def load_decentralized_checkpoint(model, checkpoint_path, n_stages=2,
n_layer_per_stage=14):
    input_path = checkpoint_path

    assert n_stages * n_layer_per_stage >= len(model.gpt_neox.layers)
    # assert model.lm_head.weight.data is not
model.transformer.wte.weight.data

    for i in range(n_stages):

        print(f'loading stage {i}')

        checkpoint = torch.load(os.path.join(input_path,
f'prank_{i}_checkpoint.pt'), map_location=torch.device("cpu"))

        if i == 0:
            _tmp = {k[len(f"{0}."):]:v for k,v in checkpoint.items() if
k.startswith(f"0.")}
            # torch.save(_tmp, os.path.join(output_path,
f'pytorch_embs.pt'))
            model.gpt_neox.embed_in.weight.data[:] =
_tmp['embed_in.weight']

            for j in range(n_layer_per_stage):
                _tmp = {k[len(f"{j+1}."):]:v for k,v in checkpoint.items()
if k.startswith(f"{j+1}.")}
                if len(_tmp) == 0:
                    break
                # torch.save(_tmp, os.path.join(output_path,
f'pytorch_{j}.pt'))
                model.gpt_neox.layers[j].load_state_dict(_tmp)

        elif i == n_stages - 1:
            for j in range(n_layer_per_stage):
                if i*n_layer_per_stage + j == 44:
                    break
                _tmp = {k[len(f"{j}."):]:v for k,v in checkpoint.items()
if k.startswith(f"{j}.")}
                if len(_tmp) == 0:
                    break
```

```
                # torch.save(_tmp, os.path.join(output_path,
f'pytorch_{i*n_layer_per_stage + j}.pt'))
                model.gpt_neox.layers[i*n_layer_per_stage +
j].load_state_dict(_tmp)

            _tmp = {k[len(f"{j}."):]:v for k,v in checkpoint.items() if
k.startswith(f"{j}.")}
            if len(_tmp) == 0:
                break
            # torch.save(_tmp, os.path.join(output_path,
f'pytorch_lm_head.pt'))
            model.gpt_neox.final_layer_norm.weight.data[:] =
_tmp['final_layer_norm.weight']
            model.gpt_neox.final_layer_norm.bias.data[:] =
_tmp['final_layer_norm.bias']
            model.embed_out.weight.data[:] = _tmp['embed_out.weight']
            if 'embed_out.bias' in _tmp:
                model.embed_out.bias.data[:] = _tmp['embed_out.bias']

        else:
            for j in range(n_layer_per_stage):
                _tmp = {k[len(f"{j}."):]:v for k,v in checkpoint.items()
if k.startswith(f"{j}.")}
                if len(_tmp) == 0:
                    break
                # torch.save(_tmp, os.path.join(output_path,
f'pytorch_{i*n_layer_per_stage + j}.pt'))
                model.gpt_neox.layers[i*n_layer_per_stage +
j].load_state_dict(_tmp)

    return model


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Convert HF checkpoints')
    parser.add_argument('--ckpt-path', type=str, default=None,
                        help='model-name')
    parser.add_argument('--save-path', type=str, default=None,
                        help='model-name')
```

```
    parser.add_argument('--n-stages', type=int, default=8,
                        help='pipeline group size')
    parser.add_argument('--n-layer-per-stage', type=int, default=6,
                        help='n layers per GPU device')
    args = parser.parse_args()

    assert args.ckpt_path is not None
    assert args.save_path is not None

    if not os.path.exists(args.save_path):
        os.mkdir(args.save_path)

    config = AutoConfig.from_pretrained('EleutherAI/gpt-neox-20b')
    tokenizer = AutoTokenizer.from_pretrained('EleutherAI/gpt-neox-20b')
    model = create_empty_gptneox(config)
    load_decentralized_checkpoint(
        model, args.ckpt_path, n_stages=args.n_stages,
n_layer_per_stage=args.n_layer_per_stage,
    )

    model.save_pretrained(args.save_path)
    config.save_pretrained(args.save_path)
    tokenizer.save_pretrained(args.save_path)\
```

```
C:\Users\student\Desktop\202001014>py -m flake8 testing_2.py
testing_2.py:2:1: F401 'torch.nn' imported but unused
testing_2.py:16:5: F401 'torch' imported but unused
testing_2.py:17:5: F811 redefinition of unused 'nn' from line 2
testing_2.py:20:5: E306 expected 1 blank line before a nested definition, found 0
testing_2.py:33:1: E302 expected 2 blank lines, found 1
testing_2.py:33:80: E501 line too long (92 > 79 characters)
testing_2.py:43:80: E501 line too long (119 > 79 characters)
testing_2.py:46:37: E231 missing whitespace after ':'
testing_2.py:46:45: E231 missing whitespace after ','
testing_2.py:46:80: E501 line too long (92 > 79 characters)
testing_2.py:46:86: F541 f-string is missing placeholders
testing_2.py:51:43: E231 missing whitespace after ':'
testing_2.py:51:51: E231 missing whitespace after ','
testing_2.py:51:80: E501 line too long (102 > 79 characters)
testing_2.py:54:80: E501 line too long (80 > 79 characters)
testing_2.py:61:41: E231 missing whitespace after ':'
testing_2.py:61:49: E231 missing whitespace after ','
testing_2.py:61:80: E501 line too long (98 > 79 characters)
testing_2.py:64:80: E501 line too long (102 > 79 characters)
testing_2.py:65:80: E501 line too long (84 > 79 characters)
testing_2.py:67:37: E231 missing whitespace after ':'
testing_2.py:67:45: E231 missing whitespace after ','
testing_2.py:67:80: E501 line too long (94 > 79 characters)
testing_2.py:70:80: E501 line too long (80 > 79 characters)
testing_2.py:71:80: E501 line too long (92 > 79 characters)
testing_2.py:72:80: E501 line too long (88 > 79 characters)
testing_2.py:79:41: E231 missing whitespace after ':'
testing_2.py:79:49: E231 missing whitespace after ','
testing_2.py:79:80: E501 line too long (98 > 79 characters)
testing_2.py:82:80: E501 line too long (102 > 79 characters)
testing_2.py:83:80: E501 line too long (84 > 79 characters)
testing_2.py:89:1: W293 blank line contains whitespace
testing_2.py:91:63: W291 trailing whitespace
testing_2.py:93:63: W291 trailing whitespace
testing_2.py:95:59: W291 trailing whitespace
testing_2.py:97:68: W291 trailing whitespace
testing_2.py:100:1: W293 blank line contains whitespace
testing_2.py:103:1: W293 blank line contains whitespace
testing_2.py:111:80: E501 line too long (96 > 79 characters)
testing_2.py:113:1: W293 blank line contains whitespace
testing_2.py:116:46: W292 no newline at end of file
```

‘Torch.nn’ imported but not used

Here ,there is an import error and, also in pylint **“Too much branch”** is shown but in flake8 no such type of error has been shown.