```

tests/test\_feedback\_analyzer.py

import unittest

from unittest.mock import patch

from utils.feedback\_analyzer import analyze\_feedback

class TestFeedbackAnalyzer(unittest.TestCase):

@patch('utils.feedback\_analyzer.openai.ChatCompletion.create')

def test\_analyze\_feedback\_success(self, mock\_openai\_create):

mock\_openai\_create.return\_value = {

'choices': [{'message': {'content': 'Performance is optimal.'}}]

}

api\_key = 'fake\_api\_key'

prompt = 'Analyze simulation data.'

simulation\_data = {

"instructions\_executed": 100,

"stalls": 2,

"branch\_mispredictions": 1

}

analysis = analyze\_feedback(api\_key, prompt, simulation\_data)

self.assertEqual(analysis, 'Performance is optimal.')

@patch('utils.feedback\_analyzer.openai.ChatCompletion.create')

def test\_analyze\_feedback\_failure(self, mock\_openai\_create):

mock\_openai\_create.side\_effect = Exception("API Error")

api\_key = 'fake\_api\_key'

prompt = 'Analyze simulation data.'

simulation\_data = {

"instructions\_executed": 100,

"stalls": 2,

"branch\_mispredictions": 1

}

analysis = analyze\_feedback(api\_key, prompt, simulation\_data)

self.assertEqual(analysis, "Error: API Error")

utils/feedback\_analyzer.py

import openai

def analyze\_feedback(api\_key, prompt, simulation\_data):

try:

response = openai.ChatCompletion.create(

api\_key=api\_key,

prompt=prompt,

simulation\_data=simulation\_data

)

return response['choices'][0]['message']['content']

except Exception as e:

return f"Error: {str(e)}"

```

In the corrected code, I added a try-except block to the `analyze\_feedback` function to handle exceptions raised by the `openai.ChatCompletion.create` method. I also updated the `test\_analyze\_feedback\_failure` test to assert that the `analyze\_feedback` function returns an error message when an exception is raised.