Celia M11209802

For the final, I used GPT-3.5-Turbo.

```python
import time
import numpy as np
import pandas as pd
import streamlit as st
from streamlit_option_menu import option_menu
from streamlit_extras.add_vertical_space import add_vertical_space
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chat_models import ChatOpenAI
from langchain.chains.question_answering import load_qa_chain
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException
import warnings
warnings.filterwarnings('ignore')


def streamlit_config():

    # page configuration
    st.set_page_config(page_title='<Resume Analyzer AI>', layout="wide")

    # page header transparent color
    page_background_color = """
    <style>

    [data-testid="stHeader"]
    {
    background: rgba(0,0,0,0);
    }

    </style>
    """
    st.markdown(page_background_color, unsafe_allow_html=True)

    # title and position
    st.markdown(f'<h1 style="text-align: center;"Resume Analyzer',
                unsafe_allow_html=True)


class resume_analyzer:

    def pdf_to_chunks(pdf):
        # read pdf and it returns memory address
        pdf_reader = PdfReader(pdf)

        # extrat text from each page separately
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text()

        # Split the long text into small chunks.
        text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=700,
            chunk_overlap=200,
            length_function=len)

        chunks = text_splitter.split_text(text=text)
        return chunks


    def openai(openai_api_key, chunks, analyze):

        # Using OpenAI service for embedding
        embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)

        # Facebook AI Similarity Serach library help us to convert text data to numerical vector
```

```python
class resume_analyzer:
    def openai(openai_api_key, chunks, analyze):
        embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)

        # Facebook AI Similarity Serach library help us to convert text data to numerical vector
        vectorstores = FAISS.from_texts(chunks, embedding=embeddings)

        # compares the query and chunks, enabling the selection of the top 'K' most similar chunks based on their similarity scores.
        docs = vectorstores.similarity_search(query=analyze, k=3)

        # creates an OpenAI object, using the ChatGPT 3.5 Turbo model
        llm = ChatOpenAI(model='gpt-3.5-turbo', api_key=openai_api_key)

        # question-answering (QA) pipeline, making use of the load_qa_chain function
        chain = load_qa_chain(llm=llm, chain_type='stuff')

        response = chain.run(input_documents=docs, question=analyze)
        return response


    def summary_prompt(query_with_chunks):

        query = f''' need to detailed summarization of below resume and finally conclude them

                """
                {query_with_chunks}
                """
                '''
        return query


    def resume_summary():

        with st.form(key='Summary'):

            # User Upload the Resume
            add_vertical_space(1)
            pdf = st.file_uploader(label='Upload Your Resume', type='pdf')
            add_vertical_space(1)

            # Enter OpenAI API Key
            col1,col2 = st.columns([0.6,0.4])
            with col1:
                openai_api_key = st.text_input(label='Enter OpenAI API Key', type='password')
            add_vertical_space(2)

            # Click on Submit Button
            submit = st.form_submit_button(label='Submit')
            add_vertical_space(1)

        add_vertical_space(3)
        if submit:
            if pdf is not None and openai_api_key != '':
                try:
                    with st.spinner('Processing...'):

                        pdf_chunks = resume_analyzer.pdf_to_chunks(pdf)

                        summary_prompt = resume_analyzer.summary_prompt(query_with_chunks=pdf_chunks)

                        summary = resume_analyzer.openai(openai_api_key=openai_api_key, chunks=pdf_chunks, analyze=summary_prompt)

                    st.markdown(f'<h4 style="color: orange;">Summary:</h4>', unsafe_allow_html=True)
                    st.write(summary)

                except Exception as e:
                    st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)

            elif pdf is None:
```

```python
class resume_analyzer:
    def resume_summary():

                except Exception as e:
                    st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)

            elif pdf is None:
                st.markdown(f'<h5 style="text-align: center;color: orange;">Please Upload Your Resume</h5>', unsafe_allow_html=True)

            elif openai_api_key == '':
                st.markdown(f'<h5 style="text-align: center;color: orange;">Please Enter OpenAI API Key</h5>', unsafe_allow_html=True)


    def job_title_prompt(query_with_chunks):

        query = f''' what are the job roles i apply to likedin based on below?

                    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                    {query_with_chunks}
                    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                    '''
        return query


    def job_title_suggestion():

        with st.form(key='Job Titles'):

            # User Upload the Resume
            add_vertical_space(1)
            pdf = st.file_uploader(label='Upload Your Resume', type='pdf')
            add_vertical_space(1)

            # Enter OpenAI API Key
            col1,col2 = st.columns([0.6,0.4])
            with col1:
                openai_api_key = st.text_input(label='Enter OpenAI API Key', type='password')
            add_vertical_space(2)

            # Click on Submit Button
            submit = st.form_submit_button(label='Submit')
            add_vertical_space(1)

        add_vertical_space(3)
        if submit:
            if pdf is not None and openai_api_key != '':
                try:
                    with st.spinner('Processing...'):

                        pdf_chunks = resume_analyzer.pdf_to_chunks(pdf)

                        summary_prompt = resume_analyzer.summary_prompt(query_with_chunks=pdf_chunks)

                        summary = resume_analyzer.openai(openai_api_key=openai_api_key, chunks=pdf_chunks, analyze=summary_prompt)

                        job_title_prompt = resume_analyzer.job_title_prompt(query_with_chunks=summary)

                        job_title = resume_analyzer.openai(openai_api_key=openai_api_key, chunks=pdf_chunks, analyze=job_title_prompt)

                    st.markdown(f'<h4 style="color: orange;">Job Titles:</h4>', unsafe_allow_html=True)
                    st.write(job_title)

                except Exception as e:
                    st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)

            elif pdf is None:
                st.markdown(f'<h5 style="text-align: center;color: orange;">Please Upload Your Resume</h5>', unsafe_allow_html=True)

            elif openai_api_key == '':
                st.markdown(f'<h5 style="text-align: center;color: orange;">Please Enter OpenAI API Key</h5>', unsafe_allow_html=True)
```

```python
class linkedin_scraper:

    def webdriver_setup():

        options = webdriver.ChromeOptions()
        options.add_argument('--headless')
        options.add_argument('--no-sandbox')
        options.add_argument('--disable-dev-shm-usage')

        driver = webdriver.Chrome(options=options)
        driver.maximize_window()
        return driver


    def get_userinput():

        add_vertical_space(2)
        with st.form(key='linkedin_scarp'):

            add_vertical_space(1)
            col1,col2,col3 = st.columns([0.5,0.3,0.2], gap='medium')
            with col1:
                job_title_input = st.text_input(label='Job Title')
                job_title_input = job_title_input.split(',')
            with col2:
                job_location = st.text_input(label='Job Location', value='Taiwan')
            with col3:
                job_count = st.number_input(label='Job Count', min_value=1, value=1, step=1)

            # Submit Button
            add_vertical_space(1)
            submit = st.form_submit_button(label='Submit')
            add_vertical_space(1)

        return job_title_input, job_location, job_count, submit


    def build_url(job_title, job_location):

        b = []
        for i in job_title:
            x = i.split()
            y = '%20'.join(x)
            b.append(y)

        job_title = '%2C%20'.join(b)
        link = f"https://in.linkedin.com/jobs/search?keywords={job_title}&location={job_location}&location=Taiwan&position=1&pageNum=0"

        return link


    def open_link(driver, link):

        while True:
            # Break the Loop if the Element is Found, Indicating the Page Loaded Correctly
            try:
                driver.get(link)
                driver.implicitly_wait(5)
                time.sleep(3)
                driver.find_element(by=By.CSS_SELECTOR, value='span.switcher-tabs__placeholder-text.m-auto')
                return

            # Retry Loading the Page
            except NoSuchElementException:
                continue


    def link_open_scrolldown(driver, link, job_count):

        # Open the Link in LinkedIn
```

(function) gap: Any

```python
class linkedin_scraper:


    def link_open_scrolldown(driver, link, job_count):

        # Open the Link in LinkedIn
        linkedin_scraper.open_link(driver, link)

        # Scroll Down the Page
        for i in range(0,job_count):
            # Simulate clicking the Page Up button
            body = driver.find_element(by=By.TAG_NAME, value='body')
            body.send_keys(Keys.PAGE_UP)
            # Scoll down the Page to End
            driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
            driver.implicitly_wait(2)
            # Click on See More Jobs Button if Present
            try:
                x = driver.find_element(by=By.CSS_SELECTOR, value="button[aria-label='See more jobs']").click()
                driver.implicitly_wait(5)
            except:
                pass


    def job_title_filter(scrap_job_title, user_job_title_input):

        # User Job Title Convert into Lower Case
        user_input = [i.lower().strip() for i in user_job_title_input]

        # scraped Job Title Convert into Lower Case
        scrap_title = [i.lower().strip() for i in [scrap_job_title]]

        # Verify Any User Job Title in the scraped Job Title
        confirmation_count = 0
        for i in user_input:
            if all(j in scrap_title[0] for j in i.split()):
                confirmation_count += 1

        # Return Job Title if confirmation_count greater than 0 else return NaN
        if confirmation_count > 0:
            return scrap_job_title
        else:
            return np.nan


    def scrap_company_data(driver, job_title_input, job_location):

        # scraping the Company Data
        company = driver.find_elements(by=By.CSS_SELECTOR, value='h4[class="base-search-card__subtitle"]')
        company_name = [i.text for i in company]

        location = driver.find_elements(by=By.CSS_SELECTOR, value='span[class="job-search-card__location"]')
        company_location = [i.text for i in location]

        title = driver.find_elements(by=By.CSS_SELECTOR, value='h3[class="base-search-card__title"]')
        job_title = [i.text for i in title]

        url = driver.find_elements(by=By.XPATH, value='//a[contains(@href, "/jobs/")]')
        website_url = [i.get_attribute('href') for i in url]

        # combine the all data to single dataframe
        df = pd.DataFrame(company_name, columns=['Company Name'])
        df['Job Title'] = pd.DataFrame(job_title)
        df['Location'] = pd.DataFrame(company_location)
        df['Website URL'] = pd.DataFrame(website_url)

        # Return Job Title if there are more than 1 matched word else return NaN
        df['Job Title'] = df['Job Title'].apply(lambda x: linkedin_scraper.job_title_filter(x, job_title_input))

        # Return Location if User Job Location in Scraped Location else return NaN
```

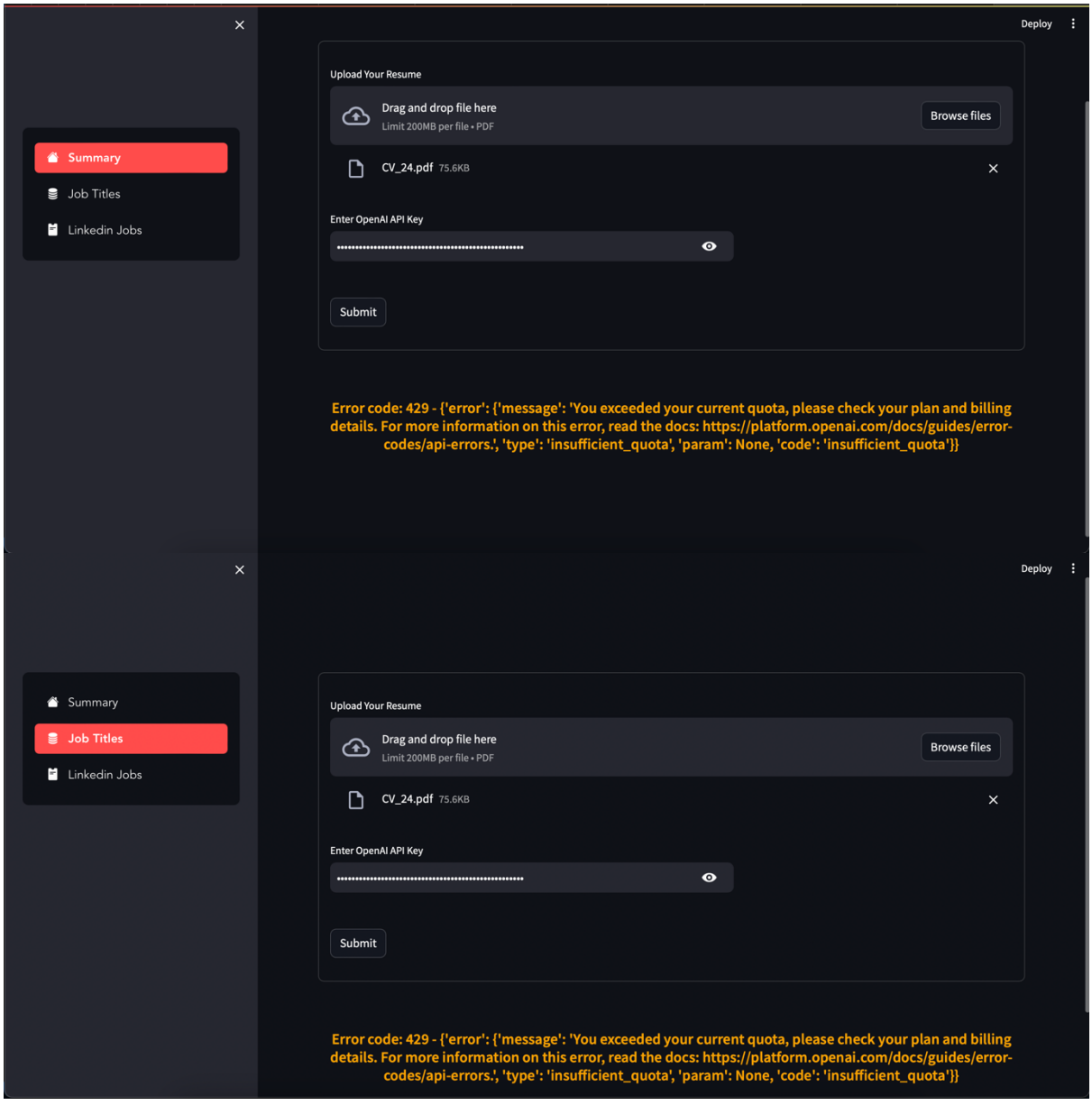
```python
class linkedin_scraper:
    def scrap_company_data(driver, job_title_input, job_location):

        # Return Location if User Job Location in Scraped Location else return NaN
        df['Location'] = df['Location'].apply(lambda x: x if job_location.lower() in x.lower() else np.nan)

        # Drop Null Values and Reset Index
        df = df.dropna()
        df.reset_index(drop=True, inplace=True)

        return df


    def scrap_job_description(driver, df, job_count):

        # Get URL into List
        website_url = df['Website URL'].tolist()

        # Scrap the Job Description
        job_description, description_count = [], 0
        for i in range(0, len(website_url)):
            try:
                # Open the Link in LinkedIn
                linkedin_scraper.open_link(driver, website_url[i])

                # Click on Show More Button
                driver.find_element(by=By.CSS_SELECTOR, value='button[data-tracking-control-name="public_jobs_show-more-html-btn"]').click()
                driver.implicitly_wait(5)
                time.sleep(1)

                # Get Job Description
                description = driver.find_elements(by=By.CSS_SELECTOR, value='div[class="show-more-less-html__markup relative overflow-hidden"]')
                data = [i.text for i in description][0]

                if len(data.strip()) > 0:
                    job_description.append(data)
                    description_count += 1
                else:
                    job_description.append('Description Not Available')

            # If URL cannot Loading Properly
            except:
                job_description.append('Description Not Available')

            # Check Description Count Meets User Job Count
            if description_count == job_count:
                break

        # Filter the Job Description
        df = df.iloc[:len(job_description), :]

        # Add Job Description in Dataframe
        df['Job Description'] = pd.DataFrame(job_description, columns=['Description'])
        df['Job Description'] = df['Job Description'].apply(lambda x: np.nan if x=='Description Not Available' else x)
        df = df.dropna()
        df.reset_index(drop=True, inplace=True)
        return df


    def display_data_userinterface(df_final):

        # Display the Data in User Interface
        add_vertical_space(1)
        if len(df_final) > 0:
            for i in range(0, len(df_final)):

                st.markdown(f'<h3 style="color: orange;">Job Posting Details : {i+1}</h3>', unsafe_allow_html=True)
                st.write(f"Company Name : {df_final.iloc[i,0]}")
                st.write(f"Job Title    : {df_final.iloc[i,1]}")
                st.write(f"Location     : {df_final.iloc[i,2]}")
```

```python
class linkedin_scraper:
    def display_data_userinterface(df_final):
            for i in range(0, len(df_final)):

                st.markdown(f'<h3 style="color: orange;">Job Posting Details : {i+1}</h3>', unsafe_allow_html=True)
                st.write(f"Company Name : {df_final.iloc[i,0]}")
                st.write(f"Job Title    : {df_final.iloc[i,1]}")
                st.write(f"Location     : {df_final.iloc[i,2]}")
                st.write(f"Website URL  : {df_final.iloc[i,3]}")

                with st.expander(label='Job Desription'):
                    st.write(df_final.iloc[i, 4])
                add_vertical_space(3)

        else:
            st.markdown(f'<h5 style="text-align: center;color: orange;">No Matching Jobs Found</h5>',
                        unsafe_allow_html=True)


    def main():

        # Initially set driver to None
        driver = None

        try:
            job_title_input, job_location, job_count, submit = linkedin_scraper.get_userinput()
            add_vertical_space(2)

            if submit:
                if job_title_input != [] and job_location != '':

                    with st.spinner('Chrome Webdriver Setup Initializing...'):
                        driver = linkedin_scraper.webdriver_setup()

                    with st.spinner('Loading More Job Listings...'):

                        # build URL based on User Job Title Input
                        link = linkedin_scraper.build_url(job_title_input, job_location)

                        # Open the Link in LinkedIn and Scroll Down the Page
                        linkedin_scraper.link_open_scrolldown(driver, link, job_count)

                    with st.spinner('scraping Job Details...'):

                        # Scraping the Company Name, Location, Job Title and URL Data
                        df = linkedin_scraper.scrap_company_data(driver, job_title_input, job_location)

                        # Scraping the Job Descriptin Data
                        df_final = linkedin_scraper. scrap_job_description(driver, df, job_count)

                    # Display the Data in User Interface
                    linkedin_scraper.display_data_userinterface(df_final)


                # If User Click Submit Button and Job Title is Empty
                elif job_title_input == []:
                    st.markdown(f'<h5 style="text-align: center;color: orange;">Job Title is Empty</h5>',
                                unsafe_allow_html=True)

                elif job_location == '':
                    st.markdown(f'<h5 style="text-align: center;color: orange;">Job Location is Empty</h5>',
                                unsafe_allow_html=True)

        except Exception as e:
            add_vertical_space(2)
            st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)

        finally:
            if driver:
                driver.quit()
```

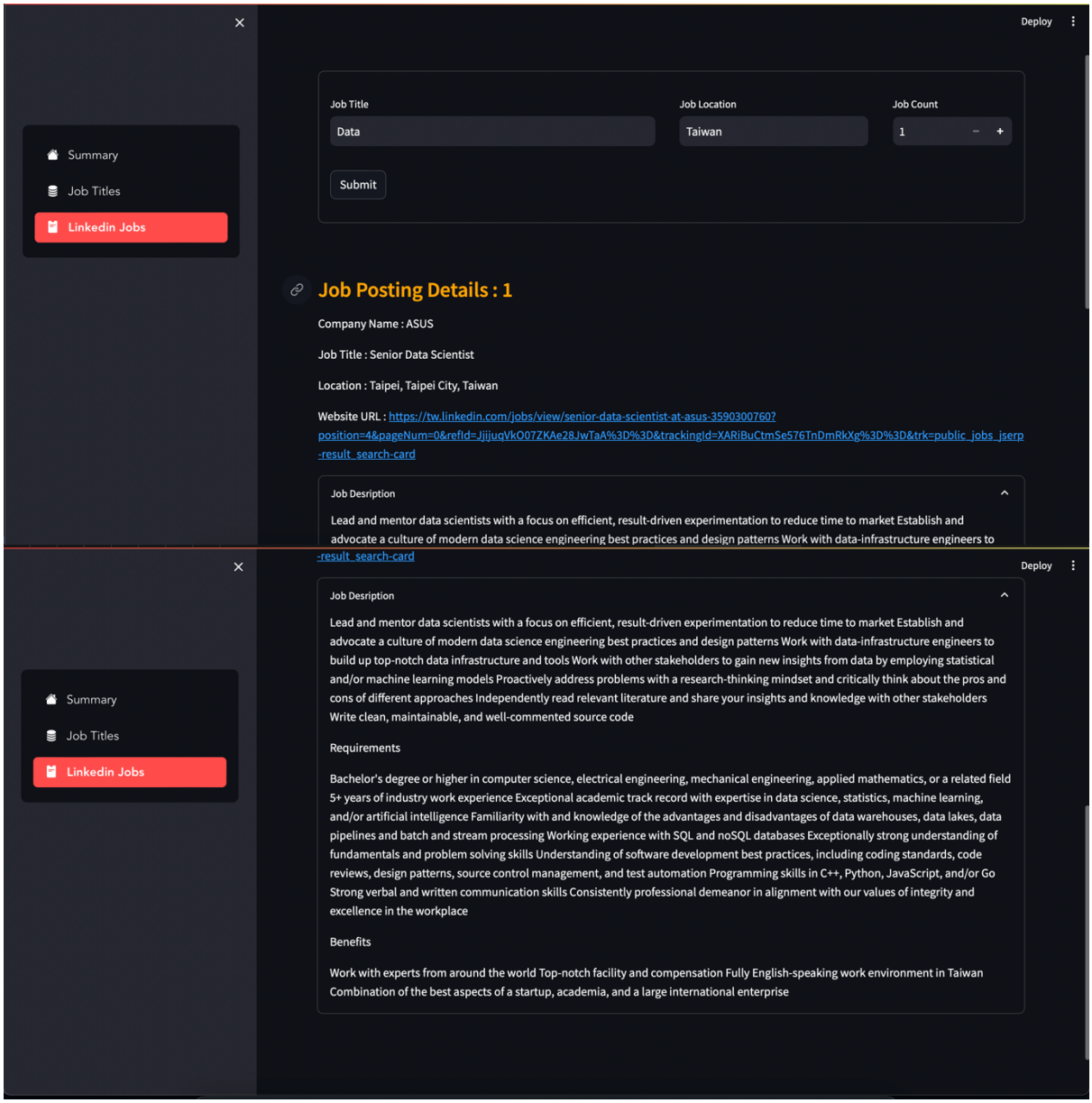
```
                    df_final = linkedin_scraper.scrap_job_description(driver, df, job_count)

                    # Display the Data in User Interface
                    linkedin_scraper.display_data_userinterface(df_final)


                # If User Click Submit Button and Job Title is Empty
                elif job_title_input == []:
                    st.markdown(f'<h5 style="text-align: center;color: orange;">Job Title is Empty</h5>',
                                unsafe_allow_html=True)

                elif job_location == '':
                    st.markdown(f'<h5 style="text-align: center;color: orange;">Job Location is Empty</h5>',
                                unsafe_allow_html=True)

        except Exception as e:
            add_vertical_space(2)
            st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)

        finally:
            if driver:
                driver.quit()



# Streamlit Configuration Setup
streamlit_config()
add_vertical_space(5)



with st.sidebar:

    add_vertical_space(4)

    option = option_menu(menu_title='', options=['Summary', 'Job Titles', 'Linkedin Jobs'],
                         icons=['house-fill', 'database-fill', 'pass-fill', 'list-ul', 'linkedin'])



if option == 'Summary':

    resume_analyzer.resume_summary()


elif option == 'Job Titles':

    resume_analyzer.job_title_suggestion()



elif option == 'Linkedin Jobs':

    linkedin_scraper.main()
```

RESULTS:

For the results, I still encounter an error while entering the OPENAI key. I tried to use my friends OpenAI key, but it led me to error 401 (lack of authentication credentials). And if I use mine, it will lead to error 429.



For the linkedin jobs, it can also help people who are currently seeking for jobs to find jobs across the world. However, for now, the job location is only set to Taiwan.



The primary purpose of this project is to help recruiters and employers sort out suitable candidates for their companies. However, I also added a feature that allows job seekers to easily find job openings through our website and use a **summary** tool to help summarize their CVs. Overall, I believe this project can be helpful not only for recruiters but also for job seekers.

For this project, I initially aimed to create a simple version, but every time I tried, I encountered errors that I couldn't figure out how to fix. I looked for solutions on websites and asked GPT, but nothing helped. So, I decided to switch to a more complex approach that might yield better results.

I started with Distil-GPT and wanted to use LLaMA 2, but unfortunately, LLaMA 2 couldn't run on my Mac. Despite that, Distil-GPT produced decent results.

Last week, I tried my best and attempted to use Pinecone, which isn’t an LLM. This week, I updated the project by integrating GPT-3.5 Turbo and using Streamlit to run it. Fortunately, I didn’t encounter any major issues, but still some API-related errors like 429 and 401 still occur.