

Celia M11209802

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

```
1 import time
2 import numpy as np
3 import pandas as pd
4 import streamlit as st
5 from streamlit_option_menu import option_menu
6 from streamlit_extras.add_vertical_space import add_vertical_space
7 from PyPDF2 import PdfReader
8 from langchain.text_splitter import RecursiveCharacterTextSplitter
9 from langchain.embeddings.openai import OpenAIEmbeddings
10 from langchain.vectorstores import FAISS
11 from langchain.chat_models import ChatOpenAI
12 from langchain.chains.question_answering import load_qa_chain
13 from selenium import webdriver
14 from selenium.webdriver.common.by import By
15 from selenium.webdriver.common.keys import Keys
16 from selenium.common.exceptions import NoSuchElementException
17 import warnings
18 warnings.filterwarnings('ignore')
19
20
21 def streamlit_config():
22
23     # page configuration
24     st.set_page_config(page_title='<Resume Analyzer AI>', layout="wide")
25
26     # page header transparent color
27     page_background_color = ""
28     <style>
29
30     [data-testid="stHeader"]
31     {
32     background: rgba(0,0,0,0);
33     }
34
35     </style>
36     ""
37     st.markdown(page_background_color, unsafe_allow_html=True)
38
39     # title and position
40     st.markdown(f'<h1 style="text-align: center;"Resume Analyzer',
41                unsafe_allow_html=True)
42
43
44 class resume_analyzer:
45
46     def pdf_to_chunks(pdf):
47         # read pdf and it returns memory address
48         pdf_reader = PdfReader(pdf)
49
50         # extrat text from each page separately
51         text = ""
52         for page in pdf_reader.pages:
53             text += page.extract_text()
54
55         # Split the long text into small chunks.
56         text_splitter = RecursiveCharacterTextSplitter(
57             chunk_size=700,
58             chunk_overlap=200,
59             length_function=len)
60
61         chunks = text_splitter.split_text(text=text)
62         return chunks
63
64
65     def openai(openai_api_key, chunks, analyze):
66
67         # Using OpenAI service for embedding
68         embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)
69
70         # Facebook AT Similarity Search library help us to convert text data to numerical vector
```

```

44 class resume_analyzer:
45     def openai(openai_api_key, chunks, analyze):
46
47         embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)
48
49         # Facebook AI Similarity Serach library help us to convert text data to numerical vector
50         vectorstores = FAISS.from_texts(chunks, embedding=embeddings)
51
52         # compares the query and chunks, enabling the selection of the top 'K' most similar chunks based on their similarity scores.
53         docs = vectorstores.similarity_search(query=analyze, k=3)
54
55         # creates an OpenAI object, using the ChatGPT 3.5 Turbo model
56         llm = ChatOpenAI(model='gpt-3.5-turbo', api_key=openai_api_key)
57
58         # question-answering (QA) pipeline, making use of the load_qa_chain function
59         chain = load_qa_chain(llm=llm, chain_type='stuff')
60
61         response = chain.run(input_documents=docs, question=analyze)
62         return response
63
64     def summary_prompt(query_with_chunks):
65
66         query = f''' need to detailed summarization of below resume and finally conclude them
67
68         {query_with_chunks}
69
70         '''
71         return query
72
73     def resume_summary():
74
75         with st.form(key='Summary'):
76
77             # User Upload the Resume
78             add_vertical_space(1)
79             pdf = st.file_uploader(label='Upload Your Resume', type='pdf')
80             add_vertical_space(1)
81
82             # Enter OpenAI API Key
83             col1,col2 = st.columns([0.6,0.4])
84             with col1:
85                 openai_api_key = st.text_input(label='Enter OpenAI API Key', type='password')
86             add_vertical_space(2)
87
88             # Click on Submit Button
89             submit = st.form_submit_button(label='Submit')
90             add_vertical_space(1)
91
92         add_vertical_space(3)
93         if submit:
94             if pdf is not None and openai_api_key != '':
95                 try:
96                     with st.spinner('Processing...'):
97
98                         pdf_chunks = resume_analyzer.pdf_to_chunks(pdf)
99
100                         summary_prompt = resume_analyzer.summary_prompt(query_with_chunks=pdf_chunks)
101
102                         summary = resume_analyzer.openai(openai_api_key=openai_api_key, chunks=pdf_chunks, analyze=summary_prompt)
103
104                         st.markdown(f'<h4 style="color: orange;">Summary:</h4>', unsafe_allow_html=True)
105                         st.write(summary)
106
107                 except Exception as e:
108                     st.markdown(f'<h5 style="text-align: center;color: orange;"><{e}</h5>', unsafe_allow_html=True)
109
110             elif pdf is None:

```

```

44 class resume_analyzer:
97     def resume_summary():
130
131         except Exception as e:
132             st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)
133
134         elif pdf is None:
135             st.markdown(f'<h5 style="text-align: center;color: orange;">Please Upload Your Resume</h5>', unsafe_allow_html=True)
136
137         elif openai_api_key == '':
138             st.markdown(f'<h5 style="text-align: center;color: orange;">Please Enter OpenAI API Key</h5>', unsafe_allow_html=True)
139
140
141     def job_title_prompt(query_with_chunks):
142
143         query = f''' what are the job roles i apply to linkedin based on below?
144
145         {query_with_chunks}
146
147         '''
148
149         return query
150
151
152     def job_title_suggestion():
153
154         with st.form(key='Job Titles'):
155
156             # User Upload the Resume
157             add_vertical_space(1)
158             pdf = st.file_uploader(label='Upload Your Resume', type='pdf')
159             add_vertical_space(1)
160
161             # Enter OpenAI API Key
162             col1,col2 = st.columns([0.6,0.4])
163             with col1:
164                 openai_api_key = st.text_input(label='Enter OpenAI API Key', type='password')
165                 add_vertical_space(2)
166
167             # Click on Submit Button
168             submit = st.form_submit_button(label='Submit')
169             add_vertical_space(1)
170
171         add_vertical_space(3)
172         if submit:
173             if pdf is not None and openai_api_key != '':
174                 try:
175                     with st.spinner('Processing...'):
176
177                         pdf_chunks = resume_analyzer.pdf_to_chunks(pdf)
178
179                         summary_prompt = resume_analyzer.summary_prompt(query_with_chunks=pdf_chunks)
180
181                         summary = resume_analyzer.openai(openai_api_key=openai_api_key, chunks=pdf_chunks, analyze=summary_prompt)
182
183                         job_title_prompt = resume_analyzer.job_title_prompt(query_with_chunks=summary)
184
185                         job_title = resume_analyzer.openai(openai_api_key=openai_api_key, chunks=pdf_chunks, analyze=job_title_prompt)
186
187                         st.markdown(f'<h4 style="color: orange;">Job Titles:</h4>', unsafe_allow_html=True)
188                         st.write(job_title)
189
190                 except Exception as e:
191                     st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)
192
193             elif pdf is None:
194                 st.markdown(f'<h5 style="text-align: center;color: orange;">Please Upload Your Resume</h5>', unsafe_allow_html=True)
195
196             elif openai_api_key == '':
197                 st.markdown(f'<h5 style="text-align: center;color: orange;">Please Enter OpenAI API Key</h5>', unsafe_allow_html=True)

```

```

201 class linkedin_scraper:
202
203     def webdriver_setup():
204
205         options = webdriver.ChromeOptions()
206         options.add_argument('--headless')
207         options.add_argument('--no-sandbox')
208         options.add_argument('--disable-dev-shm-usage')
209
210         driver = webdriver.Chrome(options=options)
211         driver.maximize_window()
212         return driver
213
214
215     def get_userinput():
216
217         add_vertical_space(2)
218         with st.form(key='linkedin_scarp'):
219
220             add_vertical_space(1)
221             col1,col2,col3 = st.columns([0.5,0.3,0.2], gap='medium')
222             with col1:
223                 job_title_input = st.text_input(label='Job Title')
224                 job_title_input = job_title_input.split(',')
225             with col2:
226                 job_location = st.text_input(label='Job Location', value='Taiwan')
227             with col3:
228                 job_count = st.number_input(label='Job Count', min_value=1, value=1, step=1)
229
230             # Submit Button
231             add_vertical_space(1)
232             submit = st.form_submit_button(label='Submit')
233             add_vertical_space(1)
234
235         return job_title_input, job_location, job_count, submit
236
237
238     def build_url(job_title, job_location):
239
240         b = []
241         for i in job_title:
242             x = i.split()
243             y = '%20'.join(x)
244             b.append(y)
245
246         job_title = '%20%20'.join(b)
247         link = f"https://in.linkedin.com/jobs/search?keywords={job_title}&location={job_location}&location=Taiwan&position=1&pageNum=0"
248
249         return link
250
251
252     def open_link(driver, link):
253
254         while True:
255             # Break the Loop if the Element is Found, Indicating the Page Loaded Correctly
256             try:
257                 driver.get(link)
258                 driver.implicitly_wait(5)
259                 time.sleep(3)
260                 driver.find_element(by=By.CSS_SELECTOR, value='span.switcher-tabs__placeholder-text.m-auto')
261                 return
262
263             # Retry Loading the Page
264             except NoSuchElementException:
265                 continue
266
267
268     def link_open_scrollown(driver, link, job_count):
269
270         # Open the Link in LinkedIn

```

```

201 class linkedin_scraper:
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
268 def link_open_scrollDown(driver, link, job_count):
269
270     # Open the Link in LinkedIn
271     linkedin_scraper.open_link(driver, link)
272
273     # Scroll Down the Page
274     for i in range(0, job_count):
275         # Simulate clicking the Page Up button
276         body = driver.find_element(by=By.TAG_NAME, value='body')
277         body.send_keys(Keys.PAGE_UP)
278         # Scroll down the Page to End
279         driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
280         driver.implicitly_wait(2)
281         # Click on See More Jobs Button if Present
282         try:
283             x = driver.find_element(by=By.CSS_SELECTOR, value="button[aria-label='See more jobs']").click()
284             driver.implicitly_wait(5)
285         except:
286             pass
287
288
289 def job_title_filter(scrap_job_title, user_job_title_input):
290
291     # User Job Title Convert into Lower Case
292     user_input = [i.lower().strip() for i in user_job_title_input]
293
294     # scraped Job Title Convert into Lower Case
295     scrap_title = [i.lower().strip() for i in scrap_job_title]
296
297     # Verify Any User Job Title in the scraped Job Title
298     confirmation_count = 0
299     for i in user_input:
300         if all(j in scrap_title[0] for j in i.split()):
301             confirmation_count += 1
302
303     # Return Job Title if confirmation_count greater than 0 else return NaN
304     if confirmation_count > 0:
305         return scrap_job_title
306     else:
307         return np.nan
308
309
310 def scrap_company_data(driver, job_title_input, job_location):
311
312     # scraping the Company Data
313     company = driver.find_elements(by=By.CSS_SELECTOR, value='h4[class="base-search-card__subtitle"]')
314     company_name = [i.text for i in company]
315
316     location = driver.find_elements(by=By.CSS_SELECTOR, value='span[class="job-search-card__location"]')
317     company_location = [i.text for i in location]
318
319     title = driver.find_elements(by=By.CSS_SELECTOR, value='h3[class="base-search-card__title"]')
320     job_title = [i.text for i in title]
321
322     url = driver.find_elements(by=By.XPATH, value='//a[contains(@href, "/jobs/")]')
323     website_url = [i.get_attribute('href') for i in url]
324
325     # combine the all data to single dataframe
326     df = pd.DataFrame(company_name, columns=['Company Name'])
327     df['Job Title'] = pd.DataFrame(job_title)
328     df['Location'] = pd.DataFrame(company_location)
329     df['Website URL'] = pd.DataFrame(website_url)
330
331     # Return Job Title if there are more than 1 matched word else return NaN
332     df['Job Title'] = df['Job Title'].apply(lambda x: linkedin_scraper.job_title_filter(x, job_title_input))
333
334     # Return Location if User Job Location in Scraped Location else return NaN

```

```

201 class linkedin_scraper:
210     def scrap_company_data(driver, job_title_input, job_location):
233
234         # Return Location if User Job Location in Scraped Location else return NaN
235         df['Location'] = df['Location'].apply(lambda x: x if job_location.lower() in x.lower() else np.nan)
236
237         # Drop Null Values and Reset Index
238         df = df.dropna()
239         df.reset_index(drop=True, inplace=True)
240
241         return df
242
243     def scrap_job_description(driver, df, job_count):
244
245         # Get URL into List
246         website_url = df['Website URL'].tolist()
247
248         # Scrap the Job Description
249         job_description, description_count = [], 0
250         for i in range(0, len(website_url)):
251             try:
252                 # Open the Link in LinkedIn
253                 linkedin_scraper.open_link(driver, website_url[i])
254
255                 # Click on Show More Button
256                 driver.find_element(by=By.CSS_SELECTOR, value='button[data-tracking-control-name="public_jobs_show-more-html-btn"]').click()
257                 driver.implicitly_wait(5)
258                 time.sleep(1)
259
260                 # Get Job Description
261                 description = driver.find_elements(by=By.CSS_SELECTOR, value='div[class="show-more-less-html__markup relative overflow-hidden"]')
262                 data = [i.text for i in description][0]
263
264                 if len(data.strip()) > 0:
265                     job_description.append(data)
266                     description_count += 1
267                 else:
268                     job_description.append('Description Not Available')
269
270             # If URL cannot Loading Properly
271             except:
272                 job_description.append('Description Not Available')
273
274             # Check Description Count Meets User Job Count
275             if description_count == job_count:
276                 break
277
278         # Filter the Job Description
279         df = df.iloc[:len(job_description), :]
280
281         # Add Job Description in Dataframe
282         df['Job Description'] = pd.DataFrame(job_description, columns=['Description'])
283         df['Job Description'] = df['Job Description'].apply(lambda x: np.nan if x=='Description Not Available' else x)
284         df = df.dropna()
285         df.reset_index(drop=True, inplace=True)
286         return df
287
288     def display_data_userinterface(df_final):
289
290         # Display the Data in User Interface
291         add_vertical_space(1)
292         if len(df_final) > 0:
293             for i in range(0, len(df_final)):
294
295                 st.markdown(f'<h3 style="color: orange;">Job Posting Details : {i+1}</h3>', unsafe_allow_html=True)
296                 st.write(f"Company Name : {df_final.iloc[i,0]}")
297                 st.write(f"Job Title : {df_final.iloc[i,1]}")
298                 st.write(f"Location : {df_final.iloc[i,2]}")
299
400

```

```

201 class linkedin_scraper:
390     def display_data_userinterface(df_final):
391         for i in range(0, len(df_final)):
392
393             st.markdown(f'<h3 style="color: orange;">Job Posting Details : {i+1}</h3>', unsafe_allow_html=True)
394             st.write(f"Company Name : {df_final.iloc[i,0]}")
395             st.write(f"Job Title : {df_final.iloc[i,1]}")
396             st.write(f"Location : {df_final.iloc[i,2]}")
397             st.write(f"Website URL : {df_final.iloc[i,3]}")
398
399             with st.expander(label='Job Description'):
400                 st.write(df_final.iloc[i, 4])
401                 add_vertical_space(3)
402
403         else:
404             st.markdown(f'<h5 style="text-align: center;color: orange;">No Matching Jobs Found</h5>',
405                         unsafe_allow_html=True)
406
407     def main():
408
409         # Initially set driver to None
410         driver = None
411
412         try:
413             job_title_input, job_location, job_count, submit = linkedin_scraper.get_userinput()
414             add_vertical_space(2)
415
416             if submit:
417                 if job_title_input != [] and job_location != '':
418                     with st.spinner('Chrome Webdriver Setup Initializing...'):
419                         driver = linkedin_scraper.webdriver_setup()
420
421                     with st.spinner('Loading More Job Listings...'):
422                         # build URL based on User Job Title Input
423                         link = linkedin_scraper.build_url(job_title_input, job_location)
424
425                         # Open the Link in LinkedIn and Scroll Down the Page
426                         linkedin_scraper.link_open_scroll_down(driver, link, job_count)
427
428                     with st.spinner('scraping Job Details...'):
429                         # Scraping the Company Name, Location, Job Title and URL Data
430                         df = linkedin_scraper.scrap_company_data(driver, job_title_input, job_location)
431
432                         # Scraping the Job Description Data
433                         df_final = linkedin_scraper.scrap_job_description(driver, df, job_count)
434
435                         # Display the Data in User Interface
436                         linkedin_scraper.display_data_userinterface(df_final)
437
438                     # If User Click Submit Button and Job Title is Empty
439                     elif job_title_input == []:
440                         st.markdown(f'<h5 style="text-align: center;color: orange;">Job Title is Empty</h5>',
441                                     unsafe_allow_html=True)
442
443                     elif job_location == '':
444                         st.markdown(f'<h5 style="text-align: center;color: orange;">Job Location is Empty</h5>',
445                                     unsafe_allow_html=True)
446
447             except Exception as e:
448                 add_vertical_space(2)
449                 st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)
450
451             finally:
452                 if driver:
453                     driver.quit()

```

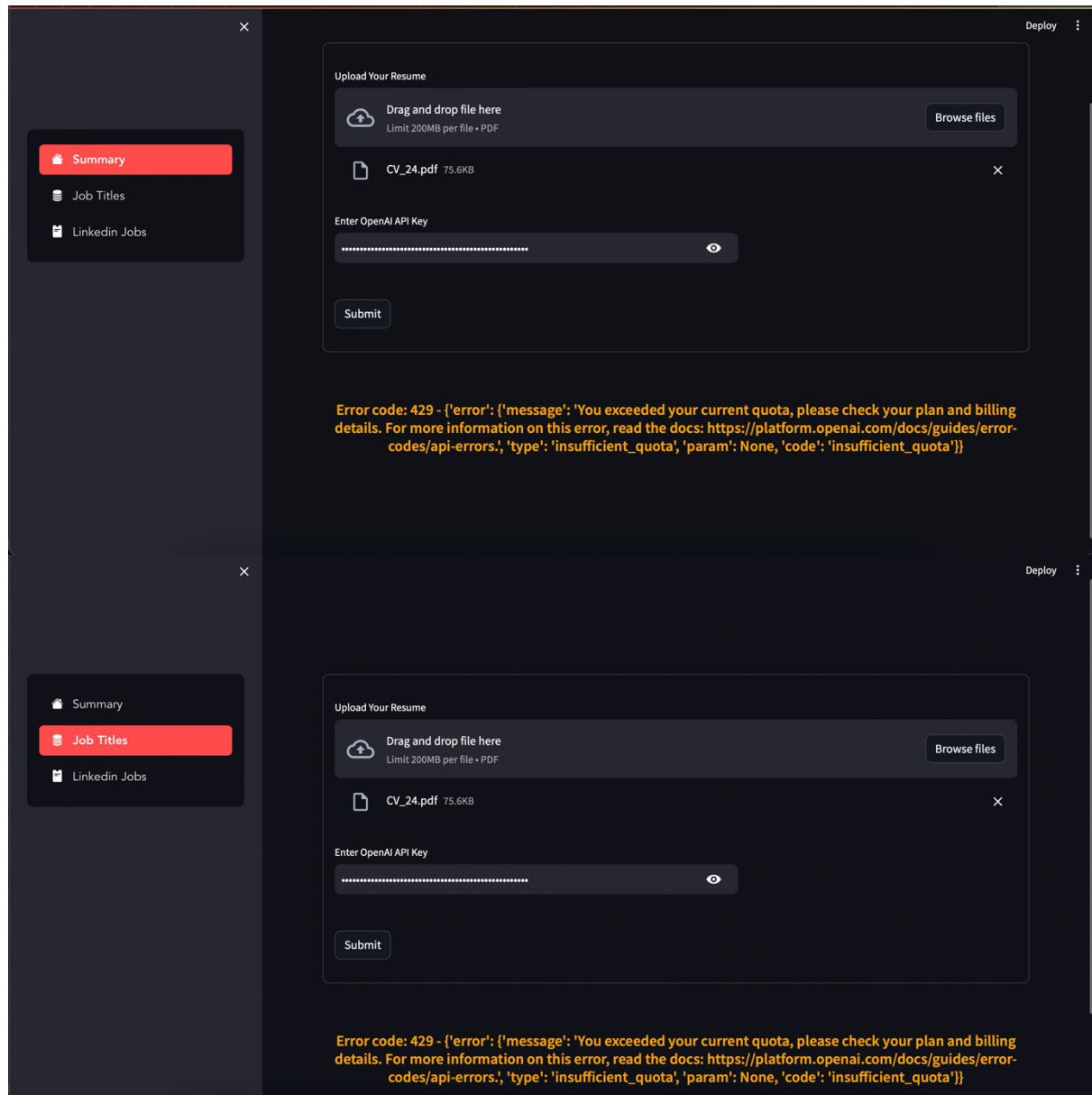
```

441         df_final = linkedin_scraper.scrap_job_description(driver, df, job_count)
442
443         # Display the Data in User Interface
444         linkedin_scraper.display_data_userinterface(df_final)
445
446         # If User Click Submit Button and Job Title is Empty
447         elif job_title_input == []:
448             st.markdown(f'<h5 style="text-align: center;color: orange;">Job Title is Empty</h5>',
449                         unsafe_allow_html=True)
450
451         elif job_location == '':
452             st.markdown(f'<h5 style="text-align: center;color: orange;">Job Location is Empty</h5>',
453                         unsafe_allow_html=True)
454
455     except Exception as e:
456         add_vertical_space(2)
457         st.markdown(f'<h5 style="text-align: center;color: orange;">{e}</h5>', unsafe_allow_html=True)
458
459     finally:
460         if driver:
461             driver.quit()
462
463
464
465
466 # Streamlit Configuration Setup
467 streamlit_config()
468 add_vertical_space(5)
469
470
471
472 with st.sidebar:
473
474     add_vertical_space(4)
475
476     option = option_menu(menu_title='', options=['Summary', 'Job Titles', 'Linkedin Jobs'],
477                          icons=['house-fill', 'database-fill', 'pass-fill', 'list-ul', 'linkedin'])
478
479
480
481 if option == 'Summary':
482
483     resume_analyzer.resume_summary()
484
485
486 elif option == 'Job Titles':
487
488     resume_analyzer.job_title_suggestion()
489
490
491
492 elif option == 'Linkedin Jobs':
493
494     linkedin_scraper.main()
495
496
497

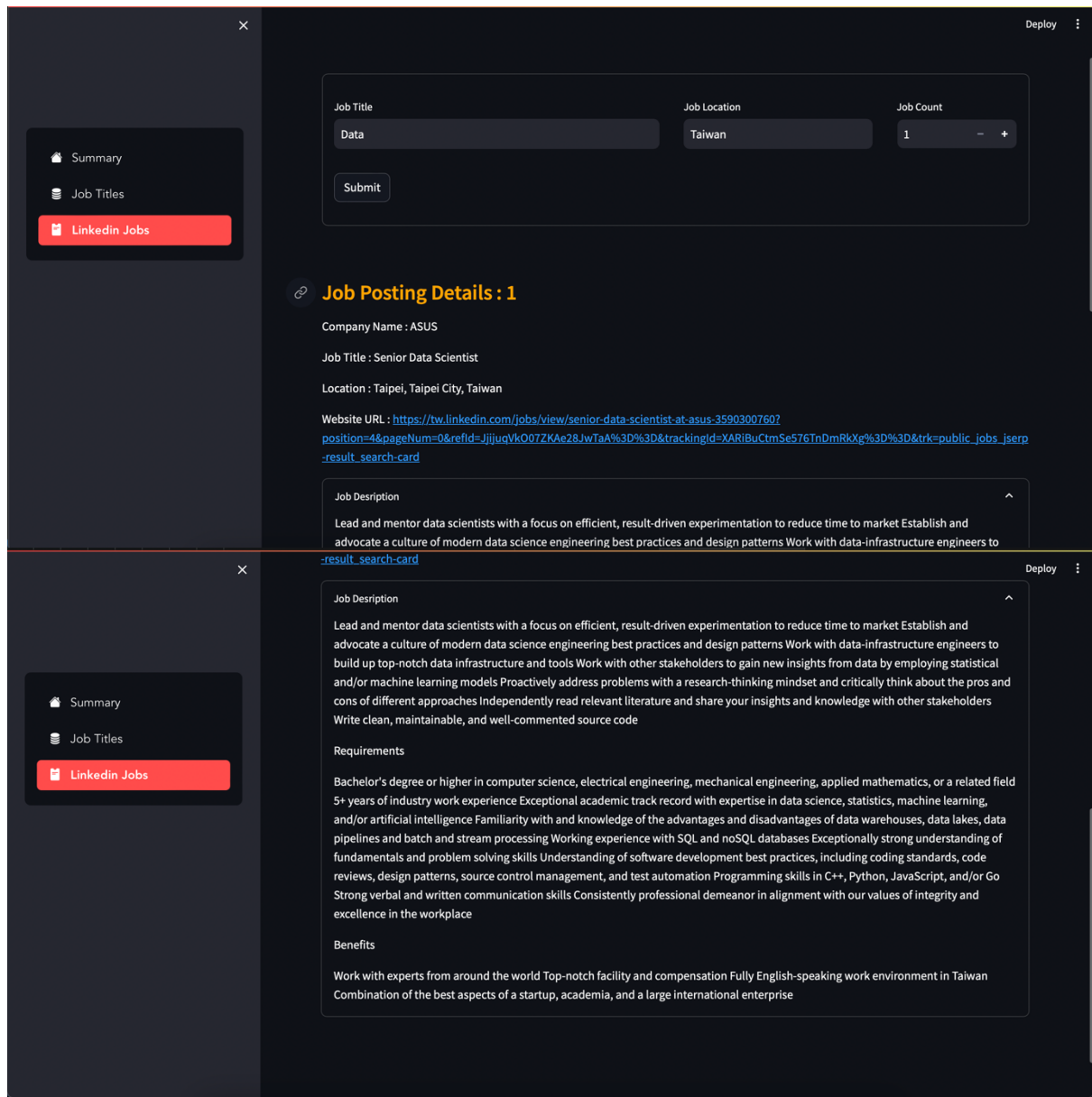
```


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.