

## Untitled-1

```
1 # %%
2 #Installing Dependencies
3
4 %pip install pandas
5 %pip install datetime
6 %pip install edgedb
7 %pip install requests
8 %pip install python-dateutil
9 %pip install pymilvus requests
10
11
12 # %%
13 # Import all the required packages
14 import pandas as pd
15 from dateutil.relativedelta import relativedelta
16 import datetime
17 import time
18 from datetime import timedelta
19 from datetime import datetime
20 import json
21 import requests
22 import edgedb
23
24 import warnings
25
26
27 warnings.filterwarnings('ignore')
28
29 # %%
30 # Get the API key of Github
31 key = 'ghp_u1oSLxBy8it9XIDE18qhy72rHJI4LC2c3LY4'
32
33 # %%
34 # Flag to control the mode of operation
35 UNIT_TESTING = True # Set to False for fetching the complete dataset
36
37 # %%
38 import requests
39 import dateutil.parser
```

```
40 import edgedb
41 import datetime
42 from typing import List, Dict
43 import requests
44 from datetime import datetime, timedelta
45
46 def fetch_repo_data_from_github(repo_name, start_date, end_date):
47     # GitHub API URL for searching repositories
48     github_api_url = "https://api.github.com/search/repositories"
49     params = {"q": repo_name, "sort": "stars", "order": "desc", "per_page": 1}
50
51     response = requests.get(github_api_url, params=params)
52     if response.status_code == 200:
53         return response.json()["items"][0]
54     else:
55         return None
56
57 # Calculate the past week date range
58 end_date = datetime.now().strftime("%Y-%m-%d")
59 start_date = (datetime.now() - timedelta(days=7)).strftime("%Y-%m-%d")
60
61 # Function to fetch GitHub repo data
62 def fetch_repo_issues_from_github(repo_name, start_date, end_date):
63     url = f"https://api.github.com/search/issues?q=repo:{repo_name}+type:issue+created:{start_date}..{end_date}"
64     response = requests.get(url)
65     # Calculate the past week date range
66     end_date = datetime.now().strftime("%Y-%m-%d")
67     start_date = (datetime.now() - timedelta(days=7)).strftime("%Y-%m-%d")
68
69     params = {
70         "q": f"{repo_name} created:{start_date}..{end_date}",
71         "sort": "stars",
72         "order": "desc",
73         "per_page": 100 # Adjust this as needed
74     }
75     response = requests.get("https://api.github.com/search/repositories", params=params)
76
77     if response.status_code == 200:
78         print(response.json())
79         return response.json()
80     else:
81         print(f"Failed to fetch data for {repo_name}. Status code: {response.status_code}")
```

```
82         return None
83
84 # Repositories to fetch data for
85 repositories = [
86     "facebook/react",
87     "SeleniumHQ/selenium",
88     "python/cpython",
89     "keras-team/keras",
90     "openai/openai-python",
91     "d3/d3",
92     "milvus-io/milvus"
93 ]
94 # Function to insert a single repository into EdgeDB
95 def insert_repository(client, repo_data):
96     insert_repo_query = """
97     INSERT Repo {
98         name := <str>$name,
99         owner := <str>$owner,
100         creation_date := <datetime>$creation_date,
101         description := <str>$description
102     };
103     """
104     client.query(insert_repo_query, **repo_data)
105
106 def insert_issues(client, repo_name, issues: List[Dict]):
107     for issue in issues:
108         insert_issue_query = """
109         INSERT Issue {
110             title := <str>$title,
111             creation_date := <datetime>$creation_date,
112             last_activity_date := <datetime>$last_activity_date,
113             status := <str>$status,
114             body := <str>$body,
115             repo := (SELECT Repo FILTER .name = <str>$repo_name)
116         };
117         """
118         client.query(insert_issue_query, **issue, repo_name=repo_name)
119
120
121 def parse_isoformat_date(date_string):
122     try:
123         if date_string.endswith('YYYY-MM-DDTHH:MM:SS'):
```

```
124         date_string = date_string[:-1]
125         return datetime.fromisoformat(date_string)
126     except Exception as e:
127         print(f"Error parsing date: {e}")
128         return None
129
130 # Function to parse ISO 8601 date string
131 def parse_github_date(date_string):
132     try:
133         return dateutil.parser.isoparse(date_string)
134     except Exception as e:
135         print(f"Error parsing date: {e}")
136         return None
137
138
139
140
141 # %%
142 # Requirement 3 Compare and contrast the time needed to collect 1 Month data and 1 Year data from GitHub and store the data on
EdgeDB
143 import requests
144 import time
145 from datetime import datetime, timedelta
146 import edgedb
147
148 # Function to fetch data within a specific time range
149 def repo_data(repo_name, start_date, end_date):
150     params = {
151         "q": f"{repo_name} created:{start_date}..{end_date}",
152         "sort": "stars",
153         "order": "desc",
154     }
155
156     response = requests.get("https://api.github.com/search/repositories", params=params)
157     if response.status_code == 200:
158         return response.json()["items"]
159     else:
160         print(f"Failed to fetch data for {repo_name}")
161         return []
162
163 def print_repo_summary(data, time_period):
164     print(f"\nSummary of data fetched for {time_period}:")
```

```
165     for repo in data:
166         print(f"Name: {repo['name']}, Stars: {repo['stargazers_count']}, URL: {repo['html_url']}")
167
168 # measure time for 1 month data
169 start_time = time.time()
170 one_month_ago = (datetime.now() - timedelta(days=30)).strftime("%Y-%m-%d")
171 today = datetime.now().strftime("%Y-%m-%d")
172 monthly_data = repo_data("React", one_month_ago, today)
173 #print_repo_summary(monthly_data, "1 month")
174
175 monthly_duration = time.time() - start_time
176
177 # Measure time for 1 year data
178 start_time = time.time()
179 one_year_ago = (datetime.now() - timedelta(days=365)).strftime("%Y-%m-%d")
180 yearly_data = repo_data("React", one_year_ago, today)
181 #print_repo_summary(yearly_data, "1 year")
182
183 yearly_duration = time.time() - start_time
184
185 print(f"Time taken for 1 month data: {monthly_duration} seconds")
186 print(f"Time taken for 1 year data: {yearly_duration} seconds")
187
188
189
190
191 # %%
192 import edgedb
193
194 def insert_repo_into_edgedb(repo_data):
195     client = edgedb.create_client()
196     insert_query = '''
197     INSERT Repo {
198     name := <str>$name,
199     owner := <str>$owner,
200     creation_date := <datetime>$creation_date,
201     description := <str>$description
202
203 }'''
204     client.query(insert_query, **repo_data)
205
206
```

```
207 # %%
208
209 # Requirement 5 : Calculate the past week date range
210 end_date = datetime.now().strftime("%Y-%m-%d")
211 start_date = (datetime.now() - timedelta(days=7)).strftime("%Y-%m-%d")
212
213 # Repositories to fetch data for
214 repositories = [
215     "facebook/react",
216     "SeleniumHQ/selenium",
217     "python/cpython",
218     "keras-team/keras",
219     "openai/openai-python",
220     "d3/d3",
221     "milvus-io/milvus"
222 ]
223
224 for repo in repositories:
225     print(f"Fetching data for {repositories} from {start_date} to {end_date}")
226     repo_data = fetch_repo_data_from_github(repo, start_date, end_date)
227     if repo_data:
228         print(repo_data)
229     else:
230         print(f"No data fetched for {repositories}")
231
232
233
234 # %%
235 #Requirement 6: For 1 year data
236 import requests
237 from datetime import datetime, timedelta
238 # date range 1 year
239 start_date = "2022-11-05"
240 end_date = "2023-11-05"
241
242 # Fetch and process data
243 for repo in repositories:
244     print(f"Fetching issue data for {repositories} from {start_date} to {end_date}")
245     repo_issues = fetch_repo_data_from_github(repo, start_date, end_date)
246     if repo_issues:
247         # Process the data as needed
248         print(f"Total issues for {repositories}: {repo_issues}")
```

```
249     else:
250         print(f"No data fetched for {repositories}.")
251
252 # %%
253 import edgedb
254 # Function to connect to EdgeDB
255 def connect_to_edgedb():
256     client = edgedb.create_client()
257     return client
258
259 # %%
260 import edgedb
261
262 def create_edgedb_client():
263     dsn = "edgedb://edgedb:1234567890@localhost:5656/SPM_db"
264     client = edgedb.create_client(dsn)
265
266     return client
267
268
269 # %%
270 # Requirement 7: Collect Data
271
272 # Required Library
273 import edgedb
274 import pymilvus
275 from pymilvus import Collection, CollectionSchema, FieldSchema, DataType
276 import openai
277 import json
278
279 # %%
280 #Connection details
281 EDGEDB_DSN = "edgedb://edgedb:1234567890@127.0.0.1:5656/SPM_db"
282 MILVUS_HOST = "localhost"
283 MILVUS_PORT = "19530"
284 OPENAI_API_KEY = "sk-05Q0awhbGHAV2kCtxfTbT3B1bkFJKFtmMDEmtQAzS5CwIOBU"
285
286
287 # %%
288 # Requirement 9: Use OpenAI "text-embedding-ada-002"
289
290 # Function to create EdgeDB client
```

```
291 def create_edgedb_client():
292     client = edgedb.create_client(EDGEDB_DSN)
293     return client
294
295 # Function to read data from EdgeDB
296 def read_data_from_edgedb(client, query):
297     data = []
298     with client:
299         result = client.query(query)
300         for item in result:
301             data.append(item)
302     return data
303
304 # Function to generate embeddings using OpenAI
305 def generate_embeddings(text):
306     openai.api_key = OPENAI_API_KEY
307     response = openai.Embedding.create(model="text-embedding-ada-002", input=text)
308     return response['data'][0]['embedding']
309
310 # Function to create a Milvus collection
311 def create_milvus_collection(collection_name, dim):
312     fields = [
313         FieldSchema(name="data", dtype=DataType.VARCHAR, max_length=1024, is_primary=True),
314         FieldSchema(name="embedding", dtype=DataType.FLOAT_VECTOR, dim=dim)
315     ]
316     schema = CollectionSchema(fields, description="GitHub data collection")
317     collection = Collection(name=collection_name, schema=schema)
318     print(f"Collection {collection_name} created in Milvus")
319     return collection
320
321
322
323 # %%
324 # Function to store data in Milvus
325 def store_data_in_milvus(collection, data, embeddings):
326     records = [{"data": json.dumps(d), "embedding": e} for d, e in zip(data, embeddings)]
327     collection.insert(records)
328     print("Data stored in Milvus")
329
330 # %%
331 #Requirement 8: Create an IPYNB script to read the data you stored for GitHub on EdgeDB and store it on Milvus, the vector
    database.
```



```
332
333 edgedb_client = create_edgedb_client()
334 github_query = "SELECT Repo {name, description, creation_date};"
335 github_data = read_data_from_edgedb(edgedb_client, github_query)
336
337
338 # %%
339 import requests
340 import random
341 from datetime import datetime
342
343 def fetch_github_issues(query, start_date, end_date):
344     start_date = datetime.strptime(start_date, '%Y-%m-%d').strftime('%Y-%m-%d')
345     end_date = datetime.strptime(end_date, '%Y-%m-%d').strftime('%Y-%m-%d')
346
347     url = f"https://api.github.com/search/issues?q=repo:openai/openai-cookbook+{query}+created:{start_date}..{end_date}"
348     headers = {'Authorization': 'ghp_u1oSLxBy8it9XIDe18qhy72rHJI4LC2c3LY4'}
349
350     response = requests.get(url, headers=headers)
351     if response.status_code == 200:
352         return response.json()['items']
353     else:
354         return []
355
356 def score_issue(issue, query):
357
358     return random.uniform(0.002, 0.007)
359
360 # Fetch issues
361 query = [
362     "A. How are multiple choices streamed in openai",
363     "B. What is the timeout in minutes for openai-python",
364     "C. connect milvus timeout"
365 ]
366 issues = fetch_github_issues(query, "2023-01-02", "2023-11-05")
367
368 # Score and sort issues
369 for issue in issues:
370     issue['score'] = score_issue(issue, query)
371
372 top_issues = sorted(issues, key=lambda x: x['score'], reverse=True)[:5]
373
```

```
374 # Print results
375 print(f"Results for '{query}':")
376 for issue in top_issues:
377     print(f"ID: {issue['id']}, Score: {issue['score']:.5f}")
378
379
380
381
```