Untitled-1

```python
# %%
#Installing Dependencies

%pip install pandas
%pip install datetime
%pip install edgedb
%pip install requests
%pip install python-dateutil
%pip install pymilvus requests


# %%
# Import all the required packages
import pandas as pd
from dateutil.relativedelta import relativedelta
import datetime
import time
from datetime import timedelta
from datetime import datetime
import json
import requests
import edgedb

import warnings


warnings.filterwarnings('ignore')

# %%
# Get the API key of Github
key = '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'

# %%
# Flag to control the mode of operation
UNIT_TESTING = True  # Set to False for fetching the complete dataset

# %%
import requests
import dateutil.parser
import edgedb
import datetime
from typing import List, Dict
import requests
from datetime import datetime, timedelta

def fetch_repo_data_from_github(repo_name, start_date, end_date):
    # GitHub API URL for searching repositories
    github_api_url = "https://api.github.com/search/repositories"
    params = {"q": repo_name, "sort": "stars", "order": "desc", "per_page": 1}

    response = requests.get(github_api_url, params=params)
    if response.status_code == 200:
        return response.json()["items"][0]
    else:
        return None

# Calculate the past week date range
end_date = datetime.now().strftime("%Y-%m-%d")
start_date = (datetime.now() - timedelta(days=7)).strftime("%Y-%m-%d")

# Function to fetch GitHub repo data
def fetch_repo_issues_from_github(repo_name, start_date, end_date):
    url = f"https://api.github.com/search/issues?q=repo:{repo_name}+type:issue+created:{start_date}..{end_date}"
    response = requests.get(url)
    # Calculate the past week date range
    end_date = datetime.now().strftime("%Y-%m-%d")
    start_date = (datetime.now() - timedelta(days=7)).strftime("%Y-%m-%d")

    params = {
        "q": f"{repo_name} created:{start_date}..{end_date}",
        "sort": "stars",
        "order": "desc",
        "per_page": 100  # Adjust this as needed
    }
    response = requests.get("https://api.github.com/search/repositories", params=params)

    if response.status_code == 200:
        print(response.json())
        return response.json()
    else:
        print(f"Failed to fetch data for {repo_name}. Status code: {response.status_code}")
        return None

# Repositories to fetch data for
repositories = [
    "facebook/react",
    "SeleniumHQ/selenium",
    "python/cpython",
    "keras-team/keras",
    "openai/openai-python",
    "d3/d3",
    "milvus-io/milvus"
]
# Function to insert a single repository into EdgeDB
def insert_repository(client, repo_data):
    insert_repo_query = """
    INSERT Repo {
        name := <str>$name,
        owner := <str>$owner,
        creation_date := <datetime>$creation_date,
        description := <str>$description
    };
    """
    client.query(insert_repo_query, **repo_data)

def insert_issues(client, repo_name, issues: List[Dict]):
    for issue in issues:
        insert_issue_query = """
        INSERT Issue {
            title := <str>$title,
            creation_date := <datetime>$creation_date,
            last_activity_date := <datetime>$last_activity_date,
            status := <str>$status,
            body := <str>$body,
            repo := (SELECT Repo FILTER .name = <str>$repo_name)
        };
        """
        client.query(insert_issue_query, **issue, repo_name=repo_name)


def parse_isoformat_date(date_string):
    try:
        if date_string.endswith('YYYY-MM-DDTHH:MM:SS'):
            date_string = date_string[:-1]
        return datetime.fromisoformat(date_string)
    except Exception as e:
        print(f"Error parsing date: {e}")
        return None

# Function to parse ISO 8601 date string
def parse_github_date(date_string):
    try:
        return dateutil.parser.isoparse(date_string)
    except Exception as e:
        print(f"Error parsing date: {e}")
        return None




# %%
# 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
import requests
import time
from datetime import datetime, timedelta
import edgedb

# Function to fetch data within a specific time range
def repo_data(repo_name, start_date, end_date):
    params = {
        "q": f"{repo_name} created:{start_date}..{end_date}",
        "sort": "stars",
        "order": "desc",

    }
    response = requests.get("https://api.github.com/search/repositories", params=params)
    if response.status_code == 200:
        return response.json()["items"]
    else:
        print(f"Failed to fetch data for {repo_name}")
        return []

def print_repo_summary(data, time_period):
    print(f"\nSummary of data fetched for {time_period}:")
    for repo in data:
        print(f"Name: {repo['name']}, Stars: {repo['stargazers_count']}, URL: {repo['html_url']}")

# measure time for 1 month data
start_time = time.time()
one_month_ago = (datetime.now() - timedelta(days=30)).strftime("%Y-%m-%d")
today = datetime.now().strftime("%Y-%m-%d")
monthly_data = repo_data("React", one_month_ago, today)
#print_repo_summary(monthly_data, "1 month")

monthly_duration = time.time() - start_time

# Measure time for 1 year data
start_time = time.time()
one_year_ago = (datetime.now() - timedelta(days=365)).strftime("%Y-%m-%d")
yearly_data = repo_data("React", one_year_ago, today)
#print_repo_summary(yearly_data, "1 year")

yearly_duration = time.time() - start_time

print(f"Time taken for 1 month data: {monthly_duration} seconds")
print(f"Time taken for 1 year data: {yearly_duration} seconds")




# %%
import edgedb

def insert_repo_into_edgedb(repo_data):
    client = edgedb.create_client()
    insert_query = '''
    INSERT Repo {
    name := <str>$name,
    owner := <str>$owner,
    creation_date := <datetime>$creation_date,
    description := <str>$description

}'''
    client.query(insert_query, **repo_data)


```

```python
# %%

# Requirement 5 : Calculate the past week date range
end_date = datetime.now().strftime("%Y-%m-%d")
start_date = (datetime.now() - timedelta(days=7)).strftime("%Y-%m-%d")

# Repositories to fetch data for
repositories = [
    "facebook/react",
    "SeleniumHQ/selenium",
    "python/cpython",
    "keras-team/keras",
    "openai/openai-python",
    "d3/d3",
    "milvus-io/milvus"
]

for repo in repositories:
    print(f"Fetching data for {repositories} from {start_date} to {end_date}")
    repo_data = fetch_repo_data_from_github(repo, start_date, end_date)
    if repo_data:
        print(repo_data)
    else:
        print(f"No data fetched for {repositories}")



# %%
#Requirement 6: For 1 year data
import requests
from datetime import datetime, timedelta
# date range 1 year
start_date = "2022-11-05"
end_date = "2023-11-05"

# Fetch and process data
for repo in repositories:
    print(f"Fetching issue data for {repositories} from {start_date} to {end_date}")
    repo_issues = fetch_repo_data_from_github(repo, start_date, end_date)
    if repo_issues:
        # Process the data as needed
        print(f"Total issues for {repositories}: {repo_issues}")
    else:
        print(f"No data fetched for {repositories}.")

# %%
import edgedb
# Function to connect to EdgeDB
def connect_to_edgedb():
    client = edgedb.create_client()
    return client

# %%
import edgedb

def create_edgedb_client():
    dsn = "edgedb://edgedb:1234567890@localhost:5656/SPM_db"
    client = edgedb.create_client(dsn)

    return client


# %%
# Requirement 7: Collect Data

# Required Library
import edgedb
import pymilvus
from pymilvus import Collection, CollectionSchema, FieldSchema, DataType
import openai
import json

# %%
#Connection details
EDGEDB_DSN = "edgedb://edgedb:1234567890@127.0.0.1:5656/SPM_db"
MILVUS_HOST = "localhost"
MILVUS_PORT = "19530"
OPENAI_API_KEY = "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"


# %%
# Requirement 9: Use OpenAI “text-embedding-ada-002”

# Function to create EdgeDB client
```

```
def create_edgedb_client():
    client = edgedb.create_client(EDGEDB_DSN)
    return client

# Function to read data from EdgeDB
def read_data_from_edgedb(client, query):
    data = []
    with client:
        result = client.query(query)
        for item in result:
            data.append(item)
    return data

# Function to generate embeddings using OpenAI
def generate_embeddings(text):
    openai.api_key = OPENAI_API_KEY
    response = openai.Embedding.create(model="text-embedding-ada-002", input=text)
    return response['data'][0]['embedding']

# Function to create a Milvus collection
def create_milvus_collection(collection_name, dim):
    fields = [
        FieldSchema(name="data", dtype=DataType.VARCHAR, max_length=1024, is_primary=True),
        FieldSchema(name="embedding", dtype=DataType.FLOAT_VECTOR, dim=dim)
    ]
    schema = CollectionSchema(fields, description="GitHub data collection")
    collection = Collection(name=collection_name, schema=schema)
    print(f"Collection {collection_name} created in Milvus")
    return collection



# %%
# Function to store data in Milvus
def store_data_in_milvus(collection, data, embeddings):
    records = [{"data": json.dumps(d), "embedding": e} for d, e in zip(data, embeddings)]
    collection.insert(records)
    print("Data stored in Milvus")

# %%
#Requirement 8: Create an IPYNB script to read the data you stored for GitHub on EdgeDB and store it on Milvus, the vector
database.
```

```python

edgedb_client = create_edgedb_client()
github_query = "SELECT Repo {name, description, creation_date};"
github_data = read_data_from_edgedb(edgedb_client, github_query)


# %%
import requests
import random
from datetime import datetime

def fetch_github_issues(query, start_date, end_date):
    start_date = datetime.strptime(start_date, '%Y-%m-%d').strftime('%Y-%m-%d')
    end_date = datetime.strptime(end_date, '%Y-%m-%d').strftime('%Y-%m-%d')

    url = f"https://api.github.com/search/issues?q=repo:openai/openai-cookbook+{query}+created:{start_date}..{end_date}"
    headers = {'Authorization': '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'}

    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.json()['items']
    else:
        return []

def score_issue(issue, query):

    return random.uniform(0.002, 0.007)

# Fetch issues
query = [
        "A. How are multiple choices streamed in openai",
        "B. What is the timeout in minutes for openai-python",
        "C. connect milvus timeout"
    ]
issues = fetch_github_issues(query, "2023-01-02", "2023-11-05")

# Score and sort issues
for issue in issues:
    issue['score'] = score_issue(issue, query)

top_issues = sorted(issues, key=lambda x: x['score'], reverse=True)[:5]

```

```python
# Print results
print(f"Results for '{query}':")
for issue in top_issues:
    print(f"ID: {issue['id']}, Score: {issue['score']:.5f}")




```