DBMS 및 SQL 활용 #4

▼ 문제

【 개요 】

단계	설명	비고
1단계	CSV 데이터 로딩 및 Pandas DataFrame 정제	
2단계	PostgreSQL 연결 및 테이블 생성	
3단계	Pandas → PostgreSQL 적재	AI 파이프라인의 전처리
4단계	PostgreSQL에서 AI 분석에 필요한 필드 추출	
5단계	PCA, 클러스터링, pgvector 적재 연계	벡터 형태로 X_scaled 또는 X_pca 결과를 PostgreSQL pgvector 필드에 저장

▼ 실습 개요

- 데이터
  - 사용자의 나이, 소득, 성별, 소비 성향, 방문 횟수
- 목적
  - 사용자의 속성(예: 나이, 소득, 소비 성향 등)을 벡터 공간에 임베딩하여, 비슷한 사용자들을 찾아내고 맞춤형 추천을 제공하는 시스템 구현
- 구현
  - 벡터 공간 임베딩
    - SentenceTransformers
    - 사용자 속성을 문장(description)으로 만들어서 문장 임베딩 모델에 넣고 벡터화
  - 비슷한 사용자들을 찾아내기
    - 코사인 유사도
  - 맞춤형 추천
    - LLM (chatgpt 5 mini)
    - 추천 결과 자연어 설명 생성
  - 시스템 구현
    - FastAPI + Streamlit
    - 추천 API/웹 UI 구축

▼ 전체 코드

```
# sc.ipynb
import os
import pandas as pd
import psycopg2
from sqlalchemy import create_engine
from dotenv import load_dotenv

# 0. set path
os.chdir("/Users/yshmbid/Documents/home/github/SQL")
load_dotenv()
```

```
# 1. 데이터 불러오기
df = pd.read_csv("17. CSV → Pandas → PostgreSQL 적재 실습_user_behavior.csv")
df

# 2. PostgreSQL 연결
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    database="postgres",
    user="postgres",
    password=os.getenv("PG_PASSWORD"),
)
cur = conn.cursor()

# 테이블 생성
cur.execute("""
CREATE TABLE IF NOT EXISTS user_behavior (
    user_id VARCHAR(10) PRIMARY KEY,
    age INTEGER,
    income INTEGER,
    gender VARCHAR(1),
    spending_score INTEGER,
    visit_count INTEGER
);
""")
conn.commit()

# 3. 데이터 적재 (CSV → DB)
for _, row in df.iterrows():
    cur.execute("""
        INSERT INTO user_behavior (user_id, age, income, gender, spending_score, visit_count)
        VALUES (%s, %s, %s, %s, %s, %s)
        ON CONFLICT (user_id) DO NOTHING;
    """, (row['user_id'], row['age'], row['income'], row['gender'], row['spending_score'], row['visit_count']))
conn.commit()

# 4. pgvector 확장 + 임베딩 컬럼 추가
cur.execute("CREATE EXTENSION IF NOT EXISTS vector;")
cur.execute("""
CREATE TABLE IF NOT EXISTS user_embeddings (
    user_id VARCHAR(10) PRIMARY KEY,
    embedding vector(384)
);
""")
conn.commit()

# 5. 임베딩 생성 및 업데이트
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("all-MiniLM-L6-v2")

# description 컬럼 생성
df["description"] = df.apply(
    lambda row: f"{row['age']}세 {row['gender']} 사용자, 소득 {row['income']}, "
                f"소비 성향 {row['spending_score']}, 방문 횟수 {row['visit_count']}회",
```

```
    axis=1
)

# DB에 벡터 저장
for uid, desc in zip(df['user_id'], df['description']):
    vec = model.encode(desc).tolist()
    cur.execute("""
        INSERT INTO user_embeddings (user_id, embedding)
        VALUES (%s, %s)
        ON CONFLICT (user_id) DO UPDATE SET embedding = EXCLUDED.embedding;
    """, (uid, vec))

conn.commit()
```

```
# api.py
# 6. FastAPI 서버
from fastapi import FastAPI
from pydantic import BaseModel
import psycopg2, os
from dotenv import load_dotenv

os.chdir("/Users/yshmbid/Documents/home/github/SQL")
load_dotenv()

app = FastAPI()

class RecommendRequest(BaseModel):
    user_id: str

def get_conn():
    return psycopg2.connect(
        host="localhost",
        port=5432,
        database="postgres",
        user="postgres",
        password=os.getenv("PG_PASSWORD"),
    )

@app.post("/recommend")
def recommend(req: RecommendRequest):
    conn = get_conn()
    cur = conn.cursor()
    # user_embeddings 기준으로 유사도 검색 + user_behavior join
    cur.execute("""
        SELECT ub.user_id, ub.age, ub.income, ub.gender
        FROM user_behavior ub
        JOIN user_embeddings ue ON ub.user_id = ue.user_id
        ORDER BY ue.embedding <=> (SELECT embedding FROM user_embeddings WHERE user_id = %s)
        LIMIT 5;
    """, (req.user_id,))
    results = cur.fetchall()
    cur.close()
    conn.close()
    return {"recommendations": results}
```

```
# ui.py
# 7. Streamlit UI
import streamlit as st
import requests
import os
from dotenv import load_dotenv

os.chdir("/Users/yshmbid/Documents/home/github/SQL")
load_dotenv()

st.title("추천 시스템 데모")

user_id = st.text_input("User ID 입력:")

if st.button("추천 받기"):
    response = requests.post("http://localhost:8000/recommend", json={"user_id": user_id})
    st.json(response.json())
```

```
# terminal 1
$ uvicorn api:app --reload --port 8000

# terminal 2
$ streamlit run ui.py
```

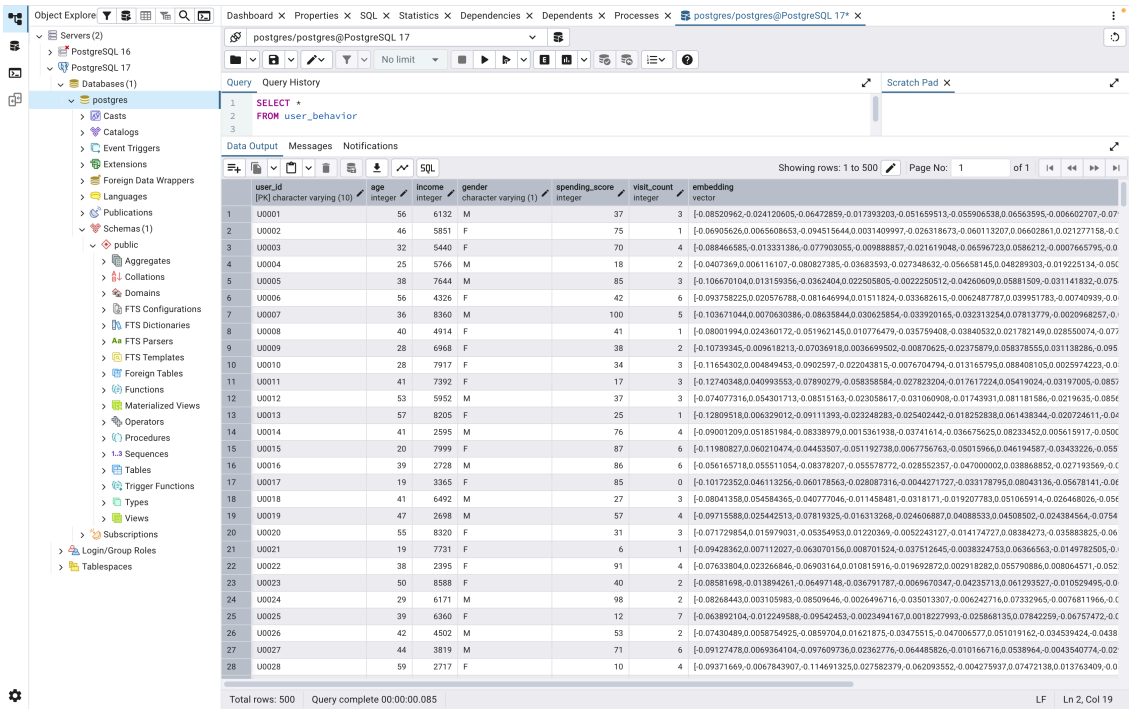
```
# 8. RAG / LLM 응용
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

def rag_query(user_id: str):
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="postgres",
        user="postgres",
        password=os.getenv("PG_PASSWORD"),
    )
    cur = conn.cursor()
    cur.execute("""
        SELECT ub.user_id, ub.age, ub.income, ub.spending_score
        FROM user_behavior ub
        JOIN user_embeddings ue ON ub.user_id = ue.user_id
        ORDER BY ue.embedding ⇔ (SELECT embedding FROM user_embeddings WHERE user_id = %s)
        LIMIT 3;
    """, (user_id,))
    rows = cur.fetchall()
    conn.close()

    context = "\n".join([str(r) for r in rows])
    prompt = f"""
    다음은 유사한 사용자 데이터입니다:
    {context}
```

```
    위 정보를 참고하여 {user_id} 사용자의 맞춤형 추천 설명을 한국어로 작성하세요.
    """

    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message.content

rag_query("U1001")

# 9. PCA 차원 축소 및 시각화
import numpy as np
import ast
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# DB에서 임베딩 + user_behavior 속성 불러오기
cur.execute("""
    SELECT ub.user_id, ub.age, ub.income, ub.gender, ue.embedding
    FROM user_behavior ub
    JOIN user_embeddings ue ON ub.user_id = ue.user_id;
""")
rows = cur.fetchall()

user_ids   = [r[0] for r in rows]
ages       = [r[1] for r in rows]
incomes    = [r[2] for r in rows]
genders    = [r[3] for r in rows]
embeddings = [ast.literal_eval(r[4]) if isinstance(r[4], str) else r[4] for r in rows]
embeddings = np.array(embeddings, dtype=float)

# PCA: 384차원 → 2차원
pca = PCA(n_components=2)
embeddings_2d = pca.fit_transform(embeddings)

# 나이 구간 (10등분)
age_bins = pd.qcut(ages, 10, labels=False, duplicates="drop")

# 소득 구간 (5등분)
income_bins = pd.qcut(incomes, 5, labels=False, duplicates="drop")

# 성별 카테고리 (M/F → 0/1)
gender_map = {"M": 0, "F": 1}
gender_bins = [gender_map[g] for g in genders]

# 1행 3열 subplot
fig, axes = plt.subplots(1, 3, figsize=(20, 6))

# 나이 구간
sc1 = axes[0].scatter(embeddings_2d[:, 0], embeddings_2d[:, 1],
                      c=age_bins, cmap="viridis", alpha=0.7)
axes[0].set_title("PCA by Age (10 bins)")
axes[0].set_xlabel("PC1")
axes[0].set_ylabel("PC2")
```

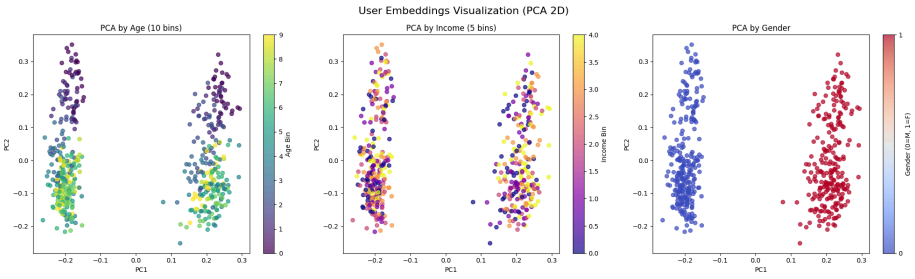
```
fig.colorbar(sc1, ax=axes[0], label="Age Bin")

# 소득 구간
sc2 = axes[1].scatter(embeddings_2d[:, 0], embeddings_2d[:, 1],
                      c=income_bins, cmap="plasma", alpha=0.7)
axes[1].set_title("PCA by Income (5 bins)")
axes[1].set_xlabel("PC1")
axes[1].set_ylabel("PC2")
fig.colorbar(sc2, ax=axes[1], label="Income Bin")

# 성별
sc3 = axes[2].scatter(embeddings_2d[:, 0], embeddings_2d[:, 1],
                      c=gender_bins, cmap="coolwarm", alpha=0.7)
axes[2].set_title("PCA by Gender")
axes[2].set_xlabel("PC1")
axes[2].set_ylabel("PC2")
fig.colorbar(sc3, ax=axes[2], ticks=[0,1], label="Gender (0=M, 1=F)")

plt.suptitle("User Embeddings Visualization (PCA 2D)", fontsize=16)
plt.tight_layout()
plt.show()
```

▼ 실행 결과 & 해석

▼ 데이터 적재 확인



▼ PCA 시각화

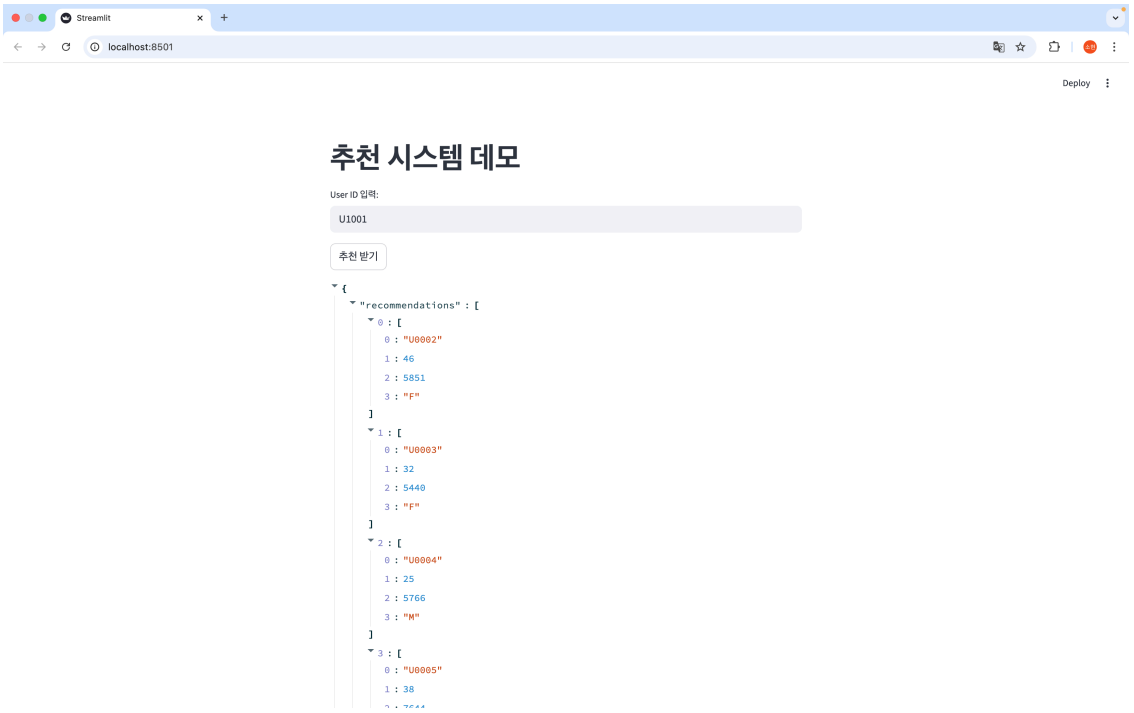


- PCA 결과 데이터 분포
  - 의미: 임베딩에서 비슷한 패턴을 가진 사용자
  - 좌우로 크게 갈라진 분포를 보임
- PCA by Age (10 bins)
  - 좌우 색깔은 섞여 있음
  - 갈라진 각 분포 내에서는 나이에 따라 어느정도 분리되는 모습을 보이긴함.
  - 나이가 벡터 공간에서 어느 정도 반영되긴 했지만 뚜렷하게 나뉘지는 않음
- PCA by Income (5 bins)
  - 색깔이 좌우, 갈라진 각 분포 모두에서 섞여 있음
  - 다양한 소득구간대가 비슷한 벡터 공간에 위치하는 경우가 많음.
- PCA by Gender
  - 매우 뚜렷하게 남성과 여성이 분리
  - SentenceTransformer로 만든 임베딩이 description 안의 "M" "F" 같은 성별 정보를 강하게 반영해서 벡터 공간에서 주된 분리 요인이 됨
- 결과 해석
  - 나이(Age) 와 소득(Income) 은 PCA 투영 공간에서 어느 정도 반영되지만 뚜렷하게 분리를 만들진 않음.
  - 해당 결과를 봤을 때는 성별 텍스트(M/F)가 SentenceTransformer 임베딩에서 과도하게 큰 영향을 미치고, 나이·소득 같은 수치형 속성은 상대적으로 약하게 반영되는 경향이 보임

▼ 추천 API/웹 UI 구축

```
(skala) yshmbid:SQL yshmbid$ pwd
/Users/yshmbid/Documents/home/github/SQL
(skala) yshmbid:SQL yshmbid$ uvicorn api:app --reload --port 8000
INFO:     Will watch for changes in these directories: ['/Users/yshmbid/Documents
/home/github/SQL']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [9709] using WatchFiles
INFO:     Started server process [9711]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

```
(skala) yshmbid:SQL yshmbid$ pwd
/Users/yshmbid/Documents/home/github/SQL
(skala) yshmbid:SQL yshmbid$ streamlit run ui.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://10.250.175.126:8501

  For better performance, install the Watchdog module:

  $ xcode-select --install
  $ pip install watchdog
```



- 입력(요청 사용자)
  - U1001
- 출력(반환된 추천 사용자 Top-5)(pgvector 유사도 검색 결과)
  - U0002
    - age 46, income 5851, gender F
  - U0003
    - age 32, income 5440, gender F
  - U0004
    - age 25, income 5766, gender M
  - U0005
    - age 38, income 7644, gender M
  - U0001
    - age 56, income 6132, gender M
- 결과 해석
  - U0002
    - 연령과 소득(46, 5851)이 U1001과 어느 정도 비슷해 유사도가 높게 나타남
  - U0003, U0004, U0005
    - 나이는 상대적으로 젊지만 소득이나 다른 속성 조합이 벡터 공간에서 가까운 위치로 계산됨
  - U0001
    - 연령은 56세로 더 높지만 소득(6132)이 U1001과 유사해 Top-5에 포함됨
  - 해당 결과를 봤을때는 전반적으로 벡터 유사도는 소득에 크게 반응하는 경향이 보이고 나이·성별·소비 성향·방문 횟수까지 균형 있게 반영하려면 description 설계나 임베딩 입력값을 더 풍부하게 만드는 개선이 필요해보임

▼ 맞춤형 추천 결과 생성

'U1001 사용자에게 맞춤형 추천을 드리겠습니다. 다음은 U1001 사용자의 유사한 사용자 데이터 기반의 맞춤형 제안입니다.\n\n1. **운동 강도 조절**: 유사 사용자들(U0001, U0002, U0003)의 평균 나이와 운동 데이터에 따르면, 30대에서 50대의 사용자들은 평균적으로 중간 강도의 운동을 선호합니다. U1001 사용자도 이와 비슷한 강도의 운동을 시도해 보시는 것이 좋습니다.\n\n2. **운동 시간 제안**: 유사 사용자들은 평균적으로 5400~6300 칼로리를 소비했습니다. U1001 사용자도 비슷한 운동 목표를 설정하시고, 일주일에 최소 3회 이상 운동을 하는 것이 좋습니다.\n\n3. **운동 종류**: 유사 사용자들은 다양한 운동 프로그램에 참여하고 있으며, 특히 유산소 운동과 근력 운동을 조화롭게 배합한 것이 효과적이었습니다. U1001 사용자도 이러한 운동 조합을 고려해 보시기 바랍니다.\n\n4. **체중 관리**: U0001 사용자와 유사하게 체중을 관리하고자 하시는 경우, 식단 조절과 함께 규칙적인 운동이 중요합니다. 검토를 통해 적절한 식단을 조정하시면 체중 관리에 도움이 될 것입니다.\n\nU1001 사용자가 건강한 라이프스타일을 유지하고, 목표 달성에 도움이 되길 바랍니다!'

- 입력
  - rag_query("U1001")
- 출력
  - 유사도 검색을 통해 추출된 Top-K 사용자의 속성을 기반으로 LLM이 생성한 맞춤형 운동 추천
- 결과 해석
  - 운동 강도 조절
    - U1001과 유사한 사용자들의 “평균 나이와 운동 데이터” 기반으로 “중간 강도의 운동” 권장
    - 비슷(평균 나이와 운동 데이터 기반으로 판단)한 생활 패턴을 가진 집단의 평균적 선택을 반영한 추천.
  - 운동 시간 제안
    - 유사 사용자들의 “소비 칼로리 평균(5400~6300)” 근거로,”주 3회 이상 운동” 추천
  - 운동 종류 추천
    - 유사 사용자들의 “선호하는 운동(유산소 운동과 근력 운동)” 근거로 유산소와 근력 운동의 균형 추천
  - 체중 관리 조언
    - 유사 사용자 U0001를 근거로 “식단 조절과 규칙적인 운동 필요성” 언급
    - U0001은 연령은 조금 다르고 소득이 비슷하고 성별은 같은 사용자였는데 체중 관리 측면에서 유사도가 높다고 판단된 듯 하지만 근거가 확실하지 않다고 생각됨.

▼ 생각 정리

- 의문점
  - PCA 상에서는 성별이 SentenceTransformer 임베딩에서 주된 영향을 준 것으로 보였는데
  - 추천 Top-K 결과를 보면 꼭 같은 성별만 묶이지 않고 오히려 소득이 비슷한 사용자들이 주로 뽑힘.
- 성별은 PCA에서 강하게 보였는데, 실제 추천은 소득 쪽으로 반응하는 이유?
  - PCA는 특정 차원에서 큰 분산을 만드는 축을 찾는다.
    - SentenceTransformer 임베딩에서 "M"과 "F"라는 토큰 차이가 큰 분산을 만들어냈으면 성별이 전역적으로 가장 두드러진 요인으로 잡힐수있다.
  - KNN에서는 특정 점을 기준으로 다른 점들과의 거리를 계산하는데
    - 거리 계산은 모든 차원 값을 합쳐서 이루어지고
    - 성별은 딱 두 값만 있어서 "M"과 "F"라는 차이는 벡터의 특정 차원에서만 나타나지만 소득, 나이, 소비 성향 같은 연속형 특성은 수치가 아주 세밀하게 달라지면서 여러 차원에 걸쳐 영향을 주기 때문에 국소적으로 벡터 거리를 잴 때는 연속형 변수에서 생기는 작은 차이들이 누적되면서 단일 토큰 차이(성별)보다 더 큰 영향을 줄 수 있다.
- 결론
  - 실제 추천이 소득 쪽으로 반응한 이유는 “벡터 거리 계산”이라는 특성 때문에 연속형 특성이 더 강조되기에 전역 요인인 성별이 KNN 기반 국소적 거리 계산에서는 다른 연속형 특성보다 덜 중요하게 작용했기 때문일수있다.