```
# Download datasets from kaggle

import json
import os

if not os.path.exists("lex-fridman-text-embedding-3-large-128.zip"):
  kaggle_json = {"username": "michaeltreynolds","key": "149701be742f30a8a0526762c61beea0"}
  kaggle_dir = os.path.expanduser(".kaggle")
  os.makedirs(kaggle_dir, exist_ok=True)
  kaggle_config_path = os.path.join(kaggle_dir, "kaggle.json")
  with open(kaggle_config_path, 'w') as f:
    json.dump(kaggle_json, f)

  !kaggle datasets download -d michaeltreynolds/lex-fridman-text-embedding-3-large-128
```

```
Dataset URL: https://www.kaggle.com/datasets/michaeltreynolds/lex-fridman-text-embedding-3-large-128
License(s): MIT
Downloading lex-fridman-text-embedding-3-large-128.zip to /content
100% 591M/594M [00:25<00:00, 28.8MB/s]
100% 594M/594M [00:25<00:00, 24.4MB/s]
```

```
# Unzip kaggle data

!unzip lex-fridman-text-embedding-3-large-128.zip
!unzip lex-fridman-text-embedding-3-large-128/*.zip
```

```
Archive:  lex-fridman-text-embedding-3-large-128.zip
  inflating: documents/documents/batch_request_0lw3vrQqdWbdBRurTGNMHU76.jsonl
  inflating: documents/documents/batch_request_3GozevpriRRzieX4za9xfNmY.jsonl
  inflating: documents/documents/batch_request_3maYxwEF1svWRpbYr10br6Wv.jsonl
  inflating: documents/documents/batch_request_7hQA9m3pUXx22JXInjYZNAu2.jsonl
  inflating: documents/documents/batch_request_7oR3UbWsHgESFLr5eqL3jkMD.jsonl
  inflating: documents/documents/batch_request_CXU6VbN4SDinplgj1MpILc1u.jsonl
  inflating: documents/documents/batch_request_Fve0NjohSf5Qe0bZtAnp8D5A.jsonl
  inflating: documents/documents/batch_request_If8QibqlgU0XRU5FcD5zpiuL.jsonl
  inflating: documents/documents/batch_request_KQAdZBdQHYMI2MfEMXf3ytLM.jsonl
  inflating: documents/documents/batch_request_MFMHpnaCSkNrbgAgOxCzaLOl.jsonl
  inflating: documents/documents/batch_request_NDF2wpJfxtprLcfkIUYE2iWQ.jsonl
  inflating: documents/documents/batch_request_S6oh8W0n2tNadcBvYxOzzYNx.jsonl
  inflating: documents/documents/batch_request_WsWMgjQhn3wXtMZcXuKc1ZE7.jsonl
  inflating: documents/documents/batch_request_Z6MhxEvYsKphmnJTno6L6QkW.jsonl
  inflating: documents/documents/batch_request_gWb7SDYIzTMTN4plMW2auahA.jsonl
  inflating: documents/documents/batch_request_hE3eSd3c5AQWcMWpaV0dBJPh.jsonl
  inflating: documents/documents/batch_request_n6Q1PS1f6wiNnWJd6qzIcbKf.jsonl
  inflating: embedding/embedding/0lw3vrQqdWbdBRurTGNMHU76.jsonl
  inflating: embedding/embedding/3GozevpriRRzieX4za9xfNmY.jsonl
  inflating: embedding/embedding/3maYxwEF1svWRpbYr10br6Wv.jsonl
  inflating: embedding/embedding/7hQA9m3pUXx22JXInjYZNAu2.jsonl
  inflating: embedding/embedding/7oR3UbWsHgESFLr5eqL3jkMD.jsonl
  inflating: embedding/embedding/CXU6VbN4SDinplgj1MpILc1u.jsonl
  inflating: embedding/embedding/Fve0NjohSf5Qe0bZtAnp8D5A.jsonl
  inflating: embedding/embedding/If8QibqlgU0XRU5FcD5zpiuL.jsonl
  inflating: embedding/embedding/KQAdZBdQHYMI2MfEMXf3ytLM.jsonl
  inflating: embedding/embedding/MFMHpnaCSkNrbgAgOxCzaLOl.jsonl
  inflating: embedding/embedding/NDF2wpJfxtprLcfkIUYE2iWQ.jsonl
  inflating: embedding/embedding/S6oh8W0n2tNadcBvYxOzzYNx.jsonl
  inflating: embedding/embedding/WsWMgjQhn3wXtMZcXuKc1ZE7.jsonl
  inflating: embedding/embedding/Z6MhxEvYsKphmnJTno6L6QkW.jsonl
  inflating: embedding/embedding/gWb7SDYIzTMTN4plMW2auahA.jsonl
  inflating: embedding/embedding/hE3eSd3c5AQWcMWpaV0dBJPh.jsonl
  inflating: embedding/embedding/n6Q1PS1f6wiNnWJd6qzIcbKf.jsonl
unzip:  cannot find or open lex-fridman-text-embedding-3-large-128/*.zip, lex-fridman-text-embedding-3-large-128/*.zip.zip o

No zipfiles found.
```

```
# Use specific libraries
!pip install datasets==2.20.0 psycopg2==2.9.9 pgcopy==1.6.0
import psycopg2
```

```
Collecting datasets==2.20.0
  Downloading datasets-2.20.0-py3-none-any.whl.metadata (19 kB)
Collecting psycopg2==2.9.9
  Downloading psycopg2-2.9.9.tar.gz (384 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 384.9/384.9 kB 6.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting pgcopy==1.6.0
  Downloading pgcopy-1.6.0-py2.py3-none-any.whl.metadata (3.4 kB)
```

```
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (3.17.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (17.0.0)
Collecting pyarrow-hotfix (from datasets==2.20.0)
  Downloading pyarrow_hotfix-0.6-py3-none-any.whl.metadata (3.6 kB)
Collecting dill<0.3.9,>=0.3.0 (from datasets==2.20.0)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (4.67.1)
Collecting xxhash (from datasets==2.20.0)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess (from datasets==2.20.0)
  Downloading multiprocess-0.70.17-py311-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.5.0,>=2023.1.0 (from fsspec[http]<=2024.5.0,>=2023.1.0->datasets==2.20.0)
  Downloading fsspec-2024.5.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (3.11.11)
Requirement already satisfied: huggingface-hub>=0.21.2 in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (0
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets==2.20.0) (6.0.2)
Requirement already satisfied: pytz in /usr/local/lib/python3.11/dist-packages (from pgcopy==1.6.0) (2024.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets==2
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets==2.20.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets==2.20.0) (25
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets==2.20.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets==2.20.
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets==2.20.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets==2.20.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets==2.2
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->dataset
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->dataset
INFO: pip is looking at multiple versions of multiprocess to determine which version is compatible with other requirements.
Collecting multiprocess (from datasets==2.20.0)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets==2.2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets==2.20.0) (20
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->dat
Downloading datasets-2.20.0-py3-none-any.whl (547 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 547.8/547.8 kB 25.0 MB/s eta 0:00:00
Downloading pgcopy-1.6.0-py2.py3-none-any.whl (13 kB)
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 7.1 MB/s eta 0:00:00
Downloading fsspec-2024.5.0-py3-none-any.whl (316 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.1/316.1 kB 21.2 MB/s eta 0:00:00
Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 11.4 MB/s eta 0:00:00
Downloading pyarrow_hotfix-0.6-py3-none-any.whl (7.9 kB)
Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
```

```
# Get your own trial account at timescaledb and paste your own connection string

#TODO
CONNECTION = "postgres://tsdbadmin:ce5wdkfs60x8mbik@fg68fwkc0s.kamfryw8sk.tsdb.cloud.timescale.com:33215/tsdb?sslmode=require"


# Use this if you want to start over on your postgres table!

DROP_TABLE = "DROP TABLE IF EXISTS podcast, segment"
with psycopg2.connect(CONNECTION) as conn:
    cursor = conn.cursor()
    cursor.execute(DROP_TABLE)
    conn.commit() # Commit the changes



# Useful function that takes a pd.DataFrame and copies it directly into a table.

import pandas as pd
import io
import psycopg2

from typing import List

def fast_pg_insert(df: pd.DataFrame, connection: str, table_name: str, columns: List[str]) -> None:
    """
        Inserts data from a pandas DataFrame into a PostgreSQL table using the COPY command for fast insertion.

        Parameters:
        df (pd.DataFrame): The DataFrame containing the data to be inserted.
        connection (str): The connection string to the PostgreSQL database.
```

```
            table_name (str): The name of the target table in the PostgreSQL database.
            columns (List[str]): A list of column names in the target table that correspond to the DataFrame columns.

            Returns:
            None
        """
        conn = psycopg2.connect(connection)
        _buffer = io.StringIO()
        df.to_csv(_buffer, sep=";", index=False, header=False)
        _buffer.seek(0)
        with conn.cursor() as c:
            c.copy_from(
                file=_buffer,
                table=table_name,
                sep=";",
                columns=columns,
                null=''
            )
        conn.commit()
        conn.close()


    # Create table statements that you'll write
    #TODO

    import os

    # may need to run this to enable vector data type if you didn't select AI in service
    # CREATE_EXTENSION = "CREATE EXTENSION vector"

    # TODO: Add create table statement
    CREATE_PODCAST_TABLE = """
    CREATE TABLE podcast (
      id text PRIMARY KEY,
      title text
    )
    """

    # TODO: Add create table statement
    CREATE_SEGMENT_TABLE = """
    CREATE TABLE segment (
      id text PRIMARY KEY,
      start_time text,
      end_time text,
      content text,
      embedding vector(128),
      podcast_id text REFERENCES podcast(id)
    )

    """

    conn = psycopg2.connect(CONNECTION)
    # TODO: Create tables with psycopg2 (example: https://www.geeksforgeeks.org/executing-sql-query-with-psycopg2-in-python/)
    #conn.cursor().execute(CREATE_EXTENSION)
    conn.cursor().execute(CREATE_PODCAST_TABLE)
    conn.cursor().execute(CREATE_SEGMENT_TABLE)
    conn.commit()
    conn.close()



    ## Extract needed data out of JSONL files. This may be the hard part!

    # TODO: What data do we need?
    # TODO: What data is in the documents jsonl files?
    # TODO: What data is in the embedding jsonl files?
    # TODO: Get some pandas data frames for our two tables so we can copy the data in!
    import os
    import json
    import pandas as pd
    import glob
    from collections import defaultdict

    def getJsonlFileList(path):
      return glob.glob(os.path.join(path, "*.jsonl"))

    # TODO: Read documents files (documents/documents)
    podcast = {}
```

```
segment = {}
for jsonl in getJsonlFileList("documents/documents"):
  for line in open(jsonl, 'r'):
    data = json.loads(line)
    podcast_id = data["body"]["metadata"]["podcast_id"]
    if podcast_id not in podcast:
      podcast[podcast_id] = {
        "id": podcast_id,
        "title": data["body"]["metadata"]["title"]
      }

    segment_id = data["custom_id"]
    segment[segment_id] = {
      "id": segment_id,
      "content": data["body"]["input"],
      "podcast_id": podcast_id,
      "start_time": data["body"]["metadata"]["start_time"],
      "stop_time": data["body"]["metadata"]["stop_time"]
    }

for jsonl in getJsonlFileList("embedding/embedding"):
  for line in open(jsonl, 'r'):
    data = json.loads(line)
    segment_id = data["custom_id"]
    segment[segment_id]["embedding"] = data["response"]["body"]["data"][0]["embedding"]

import gc
df_podcast = pd.DataFrame(podcast.values())
del podcast
gc.collect()

df_segments = pd.DataFrame(segment.values())
del segment
gc.collect()
```

```
0
```

```
#### Optional #####
# In addition to the embedding and document files you might like to load
# the full podcast raw data via the hugging face datasets library

# from datasets import load_dataset
# ds = load_dataset("Whispering-GPT/lex-fridman-podcast")



# TODO Copy all the "podcast" data into the podcast postgres table!
fast_pg_insert(df_podcast, CONNECTION, "podcast", ["id", "title"])


# TODO Copy all the "segment" data into the segment postgres table!
# HINT 1: use the recommender.utils.fast_pg_insert function to insert data into the database
# otherwise inserting the 800k documents will take a very, very long time
# HINT 2: if you don't want to use all your memory and crash
# colab, you'll need to either send the data up in chunks
# or write your own function for copying it up. Alternative to chunking maybe start
# with writing it to a CSV and then copy it up?

batch_size = 50_000

for i in range(0, len(df_segments), batch_size):
    chunk = df_segments.iloc[i : i + batch_size]
    fast_pg_insert(chunk, CONNECTION, "segment", ["id", "content", "podcast_id", "start_time", "end_time", "embedding"])

del df_segments
gc.collect()
```

```
8
```

```
## This script is used to query the database
import os
import psycopg2


# Write your queries
# Q1) What are the five most similar segments to segment "267:476"
# Input: "that if we were to meet alien life at some point"
```

```
# For each result return the podcast name, the segment id, segment raw text,  the start time, stop time, and embedding distance

conn = psycopg2.connect(CONNECTION)
cur = conn.cursor()
cur.execute("""
SELECT title, segment.id, content, start_time, end_time, embedding <-> (SELECT embedding FROM segment WHERE id = '267:476') as d
FROM segment
  JOIN podcast ON podcast.id = segment.podcast_id
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

conn.commit()
conn.close()
```

```
('Podcast: David Silver: AlphaGo, AlphaZero, and Deep Reinforcement Learning | Lex Fridman Podcast #86', '267:476', ' that i
('Podcast: Ryan Graves: UFOs, Fighter Jets, and Aliens | Lex Fridman Podcast #308', '113:2792', ' encounters, human beings,
('Podcast: Richard Dawkins: Evolution, Intelligence, Simulation, and Memes | Lex Fridman Podcast #87', '268:1019', ' Suppose
('Podcast: Jeffrey Shainline: Neuromorphic Computing and Optoelectronic Intelligence | Lex Fridman Podcast #225', '305:3600'
('Podcast: Michio Kaku: Future of Humans, Aliens, Space Travel & Physics | Lex Fridman Podcast #45', '18:464', ' So I think
```

```
# Q2) What are the five most dissimilar segments to segment "267:476"
# Input: "that if we were to meet alien life at some point"
# For each result return the podcast name, the segment id, segment raw text, the start time, stop time, and embedding distance

conn = psycopg2.connect(CONNECTION)
cur = conn.cursor()
cur.execute("""
SELECT title, segment.id, content, start_time, end_time, embedding <-> (SELECT embedding FROM segment WHERE id != '267:476' LIMI
FROM segment
  JOIN podcast ON podcast.id = segment.podcast_id
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

conn.commit()
conn.close()
```

```
('Podcast: Tyler Cowen: Economic Growth & the Fight Against Conformity & Mediocrity | Lex Fridman Podcast #174', '287:693',
('Podcast: Josh Barnett: Philosophy of Violence, Power, and the Martial Arts | Lex Fridman #165', '126:1101', " I'd be curio
('Podcast: John Vervaeke: Meaning Crisis, Atheism, Religion & the Search for Wisdom | Lex Fridman Podcast #317', '282:674',
('Podcast: Sean Kelly: Existentialism, Nihilism, and the Search for Meaning | Lex Fridman Podcast #227', '255:1649', ' Let m
('Podcast: Robert Breedlove: Philosophy of Bitcoin from First Principles | Lex Fridman Podcast #176', '77:1252', " I don't k
```

```
# Q3) What are the five most similar segments to segment '48:511'

# Input: "Is it is there something especially interesting and profound to you in terms of our current deep learning neural netwo
# For each result return the podcast name, the segment id, segment raw text,  the start time, stop time, and embedding distance
conn = psycopg2.connect(CONNECTION)
cur = conn.cursor()
cur.execute("""
SELECT title, segment.id, content, start_time, end_time, embedding <-> (SELECT embedding FROM segment WHERE id = '48:511') as di
FROM segment
  JOIN podcast ON podcast.id = segment.podcast_id
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

conn.commit()
conn.close()
```

```
('Podcast: Matt Botvinick: Neuroscience, Psychology, and AI at DeepMind | Lex Fridman Podcast #106', '48:511', ' Is it is th
('Podcast: Andrew Huberman: Neuroscience of Optimal Performance | Lex Fridman Podcast #139', '155:648', ' Is there something
('Podcast: Cal Newport: Deep Work, Focus, Productivity, Email, and Social Media | Lex Fridman Podcast #166', '61:3707', ' of
('Podcast: Matt Botvinick: Neuroscience, Psychology, and AI at DeepMind | Lex Fridman Podcast #106', '48:512', " And our bra
('Podcast: Yann LeCun: Dark Matter of Intelligence and Self-Supervised Learning | Lex Fridman Podcast #258', '276:2642', ' H
```

```
# Q4) What are the five most similar segments to segment '51:56'

# Input: "But what about like the fundamental physics of dark energy? Is there any understanding of what the heck it is?"
```

```
# For each result return the podcast name, the segment id, segment raw text,  the start time, stop time, and embedding distance
conn = psycopg2.connect(CONNECTION)
cur = conn.cursor()
cur.execute("""
SELECT title, segment.id, content, start_time, end_time, embedding <-> (SELECT embedding FROM segment WHERE id = '51:56') as dis
FROM segment
  JOIN podcast ON podcast.id = segment.podcast_id
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

conn.commit()
conn.close()
```

```
('Podcast: Alex Filippenko: Supernovae, Dark Energy, Aliens & the Expanding Universe | Lex Fridman Podcast #137', '51:56', '
('Podcast: George Hotz: Hacking the Simulation & Learning to Drive with Neural Nets | Lex Fridman Podcast #132', '308:144',
('Podcast: Lex Fridman: Ask Me Anything - AMA January 2021 | Lex Fridman Podcast', '243:273', " Like, what's up with this da
('Podcast: Katherine de Kleer: Planets, Moons, Asteroids & Life in Our Solar System | Lex Fridman Podcast #184', '196:685',
('Podcast: Alex Filippenko: Supernovae, Dark Energy, Aliens & the Expanding Universe | Lex Fridman Podcast #137', '51:36', '
```

```
# Q5) For each of the following podcast segments, find the five most similar podcast episodes. Hint: You can do this by averaging

#     a) Segment "267:476"

#     b) Segment '48:511'

#     c) Segment '51:56'

# For each result return the Podcast title and the embedding distance
conn = psycopg2.connect(CONNECTION)
cur = conn.cursor()
cur.execute("""
SELECT podcast.title, AVG(embedding <-> (SELECT embedding FROM segment WHERE id = '267:476')) as distance
FROM podcast
  JOIN segment ON podcast.id = segment.podcast_id
GROUP BY podcast.title, segment.embedding
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

cur.execute("""
SELECT podcast.title, AVG(embedding <-> (SELECT embedding FROM segment WHERE id = '48:511')) as distance
FROM podcast
  JOIN segment ON podcast.id = segment.podcast_id
GROUP BY podcast.title, segment.embedding
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

cur.execute("""
SELECT podcast.title, AVG(embedding <-> (SELECT embedding FROM segment WHERE id = '51:56')) as distance
FROM podcast
  JOIN segment ON podcast.id = segment.podcast_id
GROUP BY podcast.title, segment.embedding
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

conn.commit()
conn.close()
```

```
('Podcast: David Silver: AlphaGo, AlphaZero, and Deep Reinforcement Learning | Lex Fridman Podcast #86', 0.0)
('Podcast: Ryan Graves: UFOs, Fighter Jets, and Aliens | Lex Fridman Podcast #308', 0.6483450674336982)
('Podcast: Richard Dawkins: Evolution, Intelligence, Simulation, and Memes | Lex Fridman Podcast #87', 0.6558106859320757)
('Podcast: Jeffrey Shainline: Neuromorphic Computing and Optoelectronic Intelligence | Lex Fridman Podcast #225', 0.65954331
('Podcast: Michio Kaku: Future of Humans, Aliens, Space Travel & Physics | Lex Fridman Podcast #45', 0.6662026419636159)
('Podcast: Matt Botvinick: Neuroscience, Psychology, and AI at DeepMind | Lex Fridman Podcast #106', 0.0)
('Podcast: Andrew Huberman: Neuroscience of Optimal Performance | Lex Fridman Podcast #139', 0.652299685331962)
('Podcast: Cal Newport: Deep Work, Focus, Productivity, Email, and Social Media | Lex Fridman Podcast #166', 0.7121050124628
```

```
('Podcast: Matt Botvinick: Neuroscience, Psychology, and AI at DeepMind | Lex Fridman Podcast #106', 0.7195603322334674)
('Podcast: Yann LeCun: Dark Matter of Intelligence and Self-Supervised Learning | Lex Fridman Podcast #258', 0.7357217735737
('Podcast: Alex Filippenko: Supernovae, Dark Energy, Aliens & the Expanding Universe | Lex Fridman Podcast #137', 0.0)
('Podcast: George Hotz: Hacking the Simulation & Learning to Drive with Neural Nets | Lex Fridman Podcast #132', 0.668196522
('Podcast: Lex Fridman: Ask Me Anything - AMA January 2021 | Lex Fridman Podcast', 0.7355511762966292)
('Podcast: Katherine de Kleer: Planets, Moons, Asteroids & Life in Our Solar System | Lex Fridman Podcast #184', 0.763114159
('Podcast: Alex Filippenko: Supernovae, Dark Energy, Aliens & the Expanding Universe | Lex Fridman Podcast #137', 0.79220194
```

```
# Q6) For podcast episode id = VeH7qKZr0WI, find the five most similar podcast episodes. Hint: you can do a similar averaging pro

# Input Episode: "Balaji Srinivasan: How to Fix Government, Twitter, Science, and the FDA | Lex Fridman Podcast #331"
# For each result return the Podcast title and the embedding distance
conn = psycopg2.connect(CONNECTION)
cur = conn.cursor()
cur.execute("""
SELECT podcast.title, AVG(embedding <-> (SELECT embedding FROM segment WHERE podcast_id = 'VeH7qKZr0WI' LIMIT 1)) as distance
FROM podcast
  JOIN segment ON podcast.id = segment.podcast_id
WHERE podcast.id != 'VeH7qKZr0WI'
GROUP BY podcast.title, segment.embedding
ORDER BY distance
LIMIT 5;
""")
for row in cur.fetchall():
  print(row)

conn.commit()
conn.close()
```

```
('Podcast: Alien Debate: Sara Walker and Lee Cronin | Lex Fridman Podcast #279', 0.0)
('Podcast: Albert Bourla: Pfizer CEO | Lex Fridman Podcast #249', 0.0)
('Podcast: Alex Garland: Ex Machina, Devs, Annihilation, and the Poetry of Science | Lex Fridman Podcast #77', 0.0)
('Podcast: Alex Gladstein: Bitcoin, Authoritarianism, and Human Rights | Lex Fridman Podcast #231', 0.0)
('Podcast: Anca Dragan: Human-Robot Interaction and Reward Engineering | Lex Fridman Podcast #81', 0.0)
```

Deliverables

You will turn in a ZIP or PDF file containing all your code and a PDF file with the queries and results for questions 1-7.