Model Details:

SBERT = type of transformer model trained specifically to create sentence embeddings

Model name: all-MiniLM-L6-v2
This is one specific SBERT model in the family.

Details of the name:
all → trained on a variety of data sources, so it works for general-purpose text.
MiniLM → a smaller, faster transformer architecture.
L6 → 6 transformer layers (compact, good speed).
v2 → version 2 of this training setup.

Trade-off: fast (~384-dim vectors) but still accurate for semantic similarity.

With SBERT, each bio becomes a **384-dimensional vector of floats** learned by the model.

1. **Tokenize the text** → subwords (WordPiece/BPE).
2. **Transformer layers** encode context for each token.
3. **Pooling** turns all token vectors into one sentence vector.
4. **Normalize** (we set normalize_embeddings=True) → scale the vector to unit length so cosine similarity is just a dot product.

How it works:

Sample Data:

row index (i)	user_id	age	sex	orientation	location	bio_text (short)
0	100	26	f	straight	NYC	“i like **hiking** and **music**”
1	101	27	m	straight	NYC	“i love **trails** and **concerts**”
2	102	24	m	gay	SF	“i like **hiking** and **coding**”
3	103	29	f	bi	LA	“i enjoy **swimming** and **coding**”

After SBERT encoding (and normalization), suppose the model gives these **3-D embeddings**

- E[0] (user 100): **A** = [0.7071, 0.7071, 0.0000]
- E[1] (user 101): **B** = [0.6000, 0.8000, 0.0000]
- E[2] (user 102): **C** = [0.7071, 0.0000, 0.7071]
- E[3] (user 103): **D** = [0.0000, 0.0000, 1.0000]

(All four are length 1, i.e., already normalized.)

When the top_k_semantic_neighbors(user_id=100, k=2) is run:
We’re asking: *“Who are the top-2 semantic neighbors of user 100?”*

1) Find row index i

We look up where user_id == 100:
So, i = 0 (first row)

2) Compute similarities sims = E @ E[i]

Because embeddings are normalized, dot product = cosine similarity.
Compute dot products with A:

A·A = 0.7071*0.7071 + 0.7071*0.7071 + 0*0
= 0.5000 + 0.5000 + 0 = 1.0000

B·A = 0.6000*0.7071 + 0.8000*0.7071 + 0*0
= 0.4243 + 0.5657 + 0 = 0.9900 (≈ 0.9899)

C·A = 0.7071*0.7071 + 0*0.7071 + 0.7071*0
= 0.5000 + 0 + 0 = 0.5000

D·A = 0*0.7071 + 0*0.7071 + 1*0
= 0 + 0 + 0 = 0.0000

So sims = [1.0000, 0.9900, 0.5000, 0.0000]

3) Exclude self

sims[i] = -1
Now sims = [-1.0000, 0.9900, 0.5000, 0.0000].

4) Grab the top-k indices (fast)

idx = np.argpartition(-sims, k)[:k]

We negate sims so “largest similarity” becomes “smallest negative number.” For k=2, this returns the indices of the two highest sims. Top two indices are 1 (0.99) and 2 (0.50).

5) Sort those k by true similarity

idx = idx[np.argsort(-sims[idx])]

This sorts the selected indices in descending similarity. idx = [1, 2] (since 0.99 > 0.50).

6) Build the output table

user_id	age	sex	orientation	location	score (cosine)
101	27	m	straight	NYC	0.9900
102	24	m	gay	SF	0.5000

Top-2 semantic neighbors for user 100 based on bio_text meaning alone.