```
!pip install -U sentence-transformers
```

```
Requirement already satisfied: sentence-transformers in /usr/local/lib/python3.11/dist-packages (4.1.0)
Collecting sentence-transformers
  Downloading sentence_transformers-5.0.0-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (4.
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (4.67.1)
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (2.6.0+cu124)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (1.6.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (1.16.0)
Requirement already satisfied: huggingface-hub>=0.20.0 in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (0.33.4
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (11.3.0)
Requirement already satisfied: typing_extensions>=4.5.0 in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (4.14.
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-transformer
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-tra
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-tran
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-transfor
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence-transformer
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (3.1.6)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-trans
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-tra
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=1.11.0->sentence-transformers)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (3
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (1
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.11.0->sente
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence-tran
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentenc
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->se
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->senten
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence-transformers) (1.
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence-transforme
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.11.0->sentence-transf
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->senten
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->
Downloading sentence_transformers-5.0.0-py3-none-any.whl (470 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 470.2/470.2 kB 31.6 MB/s eta 0:00:00
```

```
from sentence_transformers import SentenceTransformer
import pandas as pd

# Load the CSV file
df = pd.read_csv("/content/MTY_Int_Sentiment_Comparation.csv", encoding="latin1")

# Ensure consistent types
df["Number"] = df["Number"].astype(str)
df["respuesta"] = df["respuesta"].astype(str)

# Load the multilingual embedding model
model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")

# Filter real and generated responses
df_real = df[df["tipo"] == "real"]
df_gen = df[df["tipo"] == "generada"]

# Group responses by question number (concatenate all responses per question)
```

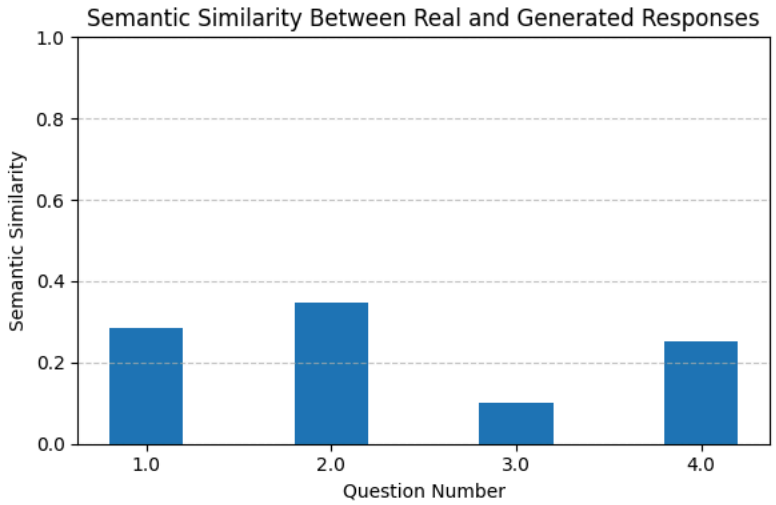
```
grouped_real = df_real.groupby("Number")["respuesta"].apply(lambda x: " ".join(x)).reset_index()
grouped_gen = df_gen.groupby("Number")["respuesta"].apply(lambda x: " ".join(x)).reset_index()

# Merge both groups on the question number
df_merged = pd.merge(grouped_real, grouped_gen, on="Number", suffixes=("_real", "_gen"))

# Generate embeddings
emb_real = model.encode(df_merged["respuesta_real"].tolist(), convert_to_tensor=True)
emb_gen = model.encode(df_merged["respuesta_gen"].tolist(), convert_to_tensor=True)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100% 229/229 [00:00<00:00, 27.2kB/s]
config_sentence_transformers.json: 100% 122/122 [00:00<00:00, 15.7kB/s]
README.md: 3.89k/? [00:00<00:00, 398kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 6.48kB/s]
config.json: 100% 645/645 [00:00<00:00, 85.9kB/s]
model.safetensors: 100% 471M/471M [00:02<00:00, 247MB/s]
tokenizer_config.json: 100% 480/480 [00:00<00:00, 58.9kB/s]
tokenizer.json: 100% 9.08M/9.08M [00:01<00:00, 6.26MB/s]
special_tokens_map.json: 100% 239/239 [00:00<00:00, 31.6kB/s]
config.json: 100% 190/190 [00:00<00:00, 24.0kB/s]
```

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Filter to keep only question numbers present in both sets
valid_numbers = set(df_real["Number"]).intersection(set(df_gen["Number"]))
df_valid = df_merged[df_merged["Number"].isin(valid_numbers)].copy()

# Initialize list for similarity scores
similarities = []

# Compute cosine similarity for each valid question
for i, row in df_valid.iterrows():
    try:
        # Generate embeddings
        emb_r = model.encode(row["respuesta_real"], convert_to_tensor=True).unsqueeze(0)
        emb_g = model.encode(row["respuesta_gen"], convert_to_tensor=True).unsqueeze(0)

        # Compute cosine similarity
        sim = cosine_similarity(emb_r.cpu().numpy(), emb_g.cpu().numpy())[0][0]
    except Exception as e:
        sim = None

    similarities.append(sim)

# Add results to the dataframe
df_valid["semantic_similarity"] = similarities

# Display final table
df_valid[["Number", "semantic_similarity"]]
```

	Number	semantic_similarity
0	1.0	0.283566
1	2.0	0.346277
2	3.0	0.101455
3	4.0	0.252968

```
import matplotlib.pyplot as plt

# Plot semantic similarity by question
plt.figure(figsize=(6, 4))
plt.bar(df_valid["Number"], df_valid["semantic_similarity"], width=0.4)
plt.xlabel("Question Number")
plt.ylabel("Semantic Similarity")
plt.title("Semantic Similarity Between Real and Generated Responses")
plt.ylim(0, 1)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()

# Convert embedding tensors to lists
df_merged["embedding_real"] = [vec.tolist() for vec in emb_real]
df_merged["embedding_gen"] = [vec.tolist() for vec in emb_gen]

# Merge similarity scores from df_valid
df_export = pd.merge(df_merged, df_valid[["Number", "semantic_similarity"]], on="Number", how="left")

# Export to CSV
df_export.to_csv("/content/MTY_Semantic_Embeddings.csv", index=False)
```



Haz doble clic (o pulsa Intro) para editar

```
from sentence_transformers import SentenceTransformer
import pandas as pd

# Load the model (si no lo has cargado antes)
model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")

# Asegúrate de tener bien cargado el CSV con los valores correctos
df = pd.read_csv("/content/MTY_Int_Sentiment_Comparation.csv", encoding="latin1")

# Clean and prepare
df["respuesta"] = df["respuesta"].astype(str)
df["Number"] = df["Number"].astype(str)

# Generate embeddings for each response individually
embeddings = model.encode(df["respuesta"].tolist(), convert_to_tensor=False)

# Add them as a new column
df["embedding"] = [e.tolist() for e in embeddings]

# Save the full DataFrame with embeddings
df.to_csv("/content/MTY_AllResponses_withEmbeddings.csv", index=False)
```

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
```

```
# Load data
df = pd.read_csv('/content/MTY_Int_Sentiment_Comparation.csv', encoding='latin1')
df = df.dropna(subset=['respuesta', 'Number', 'tipo'])
df["respuesta"] = df["respuesta"].astype(str)
df["Number"] = df["Number"].astype(str)

# Load sentence embedding model
model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")

results = []

# Loop over each question
for q in sorted(df['Number'].unique()):
    df_q = df[df['Number'] == q]

    # Get all real responses
    real_responses = df_q[df_q['tipo'] == 'real']['respuesta'].dropna().tolist()

    if len(real_responses) < 2:
        continue  # not enough data to compute reference

    # Create reference embedding for the full set of real responses
    real_reference_text = " ".join(real_responses)
    ref_embedding = model.encode(real_reference_text, convert_to_tensor=True).unsqueeze(0)

    # Compare each generated response to real reference
    for r in df_q[df_q['tipo'] == 'generada']['respuesta'].dropna().tolist():
        emb_r = model.encode(r, convert_to_tensor=True).unsqueeze(0)
        score = cosine_similarity(ref_embedding.cpu().numpy(), emb_r.cpu().numpy())[0][0]
        results.append({'tipo': 'generada', 'pregunta': int(float(q)), 'cosine_semantico': score})

    # Leave-one-out for real responses
    real_list = df_q[df_q['tipo'] == 'real']['respuesta'].dropna().tolist()
    for i, r in enumerate(real_list):
        other_real = real_list[:i] + real_list[i+1:]
        if not other_real:
            continue
        ref_text = " ".join(other_real)
        ref_emb = model.encode(ref_text, convert_to_tensor=True).unsqueeze(0)
        emb_r = model.encode(r, convert_to_tensor=True).unsqueeze(0)
        score = cosine_similarity(ref_emb.cpu().numpy(), emb_r.cpu().numpy())[0][0]
        results.append({'tipo': 'real', 'pregunta': int(float(q)), 'cosine_semantico': score})

# Save and display
df_sem_sim = pd.DataFrame(results)
df_sem_sim.to_csv('/content/semantic_similarity_vs_question_set.csv', index=False)
df_sem_sim.head()
```

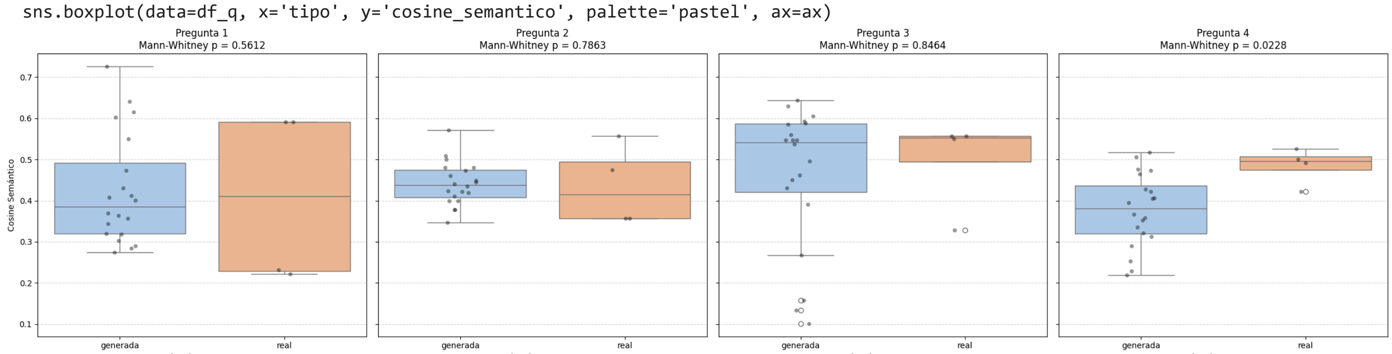
	tipo	pregunta	cosine_semantico
0	generada	1	0.283566
1	generada	1	0.408447
2	generada	1	0.602968
3	generada	1	0.549466
4	generada	1	0.318203

Pasos siguientes: Generar código con df_sem_sim | Ver gráficos recomendados | New interactive sheet

```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import mannwhitneyu

# Get unique question numbers
questions = df_sem_sim['pregunta'].unique()

# Set up figure for subplots
fig, axes = plt.subplots(1, len(questions), figsize=(6 * len(questions), 6), sharey=True)

# Store results
stat_results = []

# Loop over each question
```

```
for i, q in enumerate(sorted(questions)):
    ax = axes[i] if len(questions) > 1 else axes

    # Filter data for this question
    df_q = df_sem_sim[df_sem_sim['pregunta'] == q]
    real_scores = df_q[df_q['tipo'] == 'real']['cosine_semantico']
    gen_scores = df_q[df_q['tipo'] == 'generada']['cosine_semantico']

    # Mann-Whitney U test
    u_stat, p_value = mannwhitneyu(real_scores, gen_scores, alternative='two-sided')
    stat_results.append({'pregunta': q, 'U': u_stat, 'p_value': p_value})

    # Boxplot + stripplot
    sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
    sns.stripplot(data=df_q, x='tipo', y='cosine_semantico', color='black', alpha=0.4, ax=ax)

    ax.set_title(f'Pregunta {q}\nMann-Whitney p = {p_value:.4f}')
    ax.set_xlabel('Tipo de respuesta')
    ax.set_ylabel('Cosine Semántico' if i == 0 else '')
    ax.grid(axis='y', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()

# Print test results
print("\nResultados estadísticos por pregunta:")
for r in stat_results:
    signif = "✅ SIGNIFICATIVA" if r['p_value'] < 0.05 else "🔍 No significativa"
    print(f"Pregunta {r['pregunta']}: U = {r['U']}, p = {r['p_value']:.4f} → {signif}")
```

```
/tmp/ipython-input-8-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
/tmp/ipython-input-8-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
/tmp/ipython-input-8-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
/tmp/ipython-input-8-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
```




```
Resultados estadísticos por pregunta:
Pregunta 1: U = 32.0, p = 0.5612 → 🔍 No significativa
Pregunta 2: U = 36.0, p = 0.7863 → 🔍 No significativa
Pregunta 3: U = 43.0, p = 0.8464 → 🔍 No significativa
Pregunta 4: U = 69.0, p = 0.0228 → ✅ SIGNIFICATIVA
```

CDMX

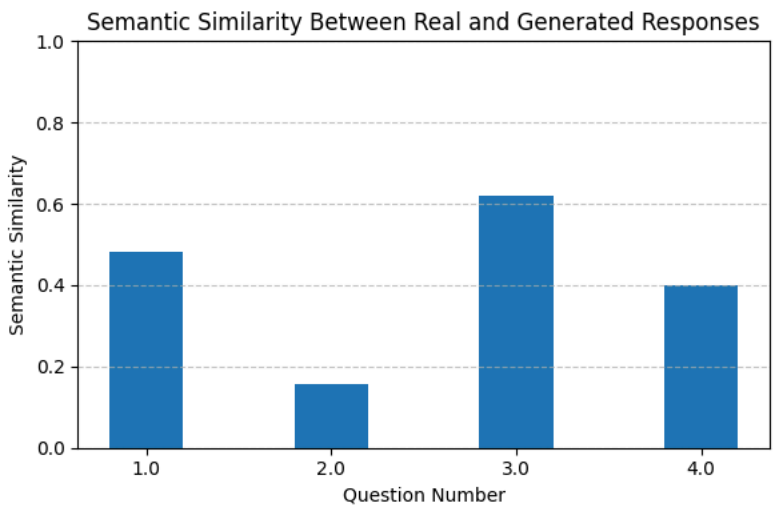
```
from sentence_transformers import SentenceTransformer
import pandas as pd

# Load the CSV file
df = pd.read_csv("/content/CDMX_Int_Sentiment_Comparation.csv", encoding="latin1")

# Ensure consistent types
df["Number"] = df["Number"].astype(str)
df["respuesta"] = df["respuesta"].astype(str)

# Load the multilingual embedding model
model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")

# Filter real and generated responses
df_real = df[df["tipo"] == "real"]
df_gen = df[df["tipo"] == "generada"]

# Group responses by question number (concatenate all responses per question)
grouped_real = df_real.groupby("Number")["respuesta"].apply(lambda x: " ".join(x)).reset_index()
grouped_gen = df_gen.groupby("Number")["respuesta"].apply(lambda x: " ".join(x)).reset_index()

# Merge both groups on the question number
df_merged = pd.merge(grouped_real, grouped_gen, on="Number", suffixes=("_real", "_gen"))

# Generate embeddings
emb_real = model.encode(df_merged["respuesta_real"].tolist(), convert_to_tensor=True)
emb_gen = model.encode(df_merged["respuesta_gen"].tolist(), convert_to_tensor=True)
```

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Filter to keep only question numbers present in both sets
valid_numbers = set(df_real["Number"]).intersection(set(df_gen["Number"]))
df_valid = df_merged[df_merged["Number"].isin(valid_numbers)].copy()

# Initialize list for similarity scores
similarities = []

# Compute cosine similarity for each valid question
for i, row in df_valid.iterrows():
    try:
        # Generate embeddings
        emb_r = model.encode(row["respuesta_real"], convert_to_tensor=True).unsqueeze(0)
        emb_g = model.encode(row["respuesta_gen"], convert_to_tensor=True).unsqueeze(0)

        # Compute cosine similarity
        sim = cosine_similarity(emb_r.cpu().numpy(), emb_g.cpu().numpy())[0][0]
    except Exception as e:
        sim = None

    similarities.append(sim)

# Add results to the dataframe
df_valid["semantic_similarity"] = similarities

# Display final table
df_valid[["Number", "semantic_similarity"]]
```

	Number	semantic_similarity
0	1.0	0.483006
1	2.0	0.155695
2	3.0	0.619091
3	4.0	0.401177

```
import matplotlib.pyplot as plt

# Plot semantic similarity by question
plt.figure(figsize=(6, 4))
plt.bar(df_valid["Number"], df_valid["semantic_similarity"], width=0.4)
plt.xlabel("Question Number")
```

```
plt.ylabel("Semantic Similarity")
plt.title("Semantic Similarity Between Real and Generated Responses")
plt.ylim(0, 1)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()

# Convert embedding tensors to lists
df_merged["embedding_real"] = [vec.tolist() for vec in emb_real]
df_merged["embedding_gen"] = [vec.tolist() for vec in emb_gen]

# Merge similarity scores from df_valid
df_export = pd.merge(df_merged, df_valid[["Number", "semantic_similarity"]], on="Number", how="left")

# Export to CSV
df_export.to_csv("/content/CDMX_Semantic_Embeddings.csv", index=False)
```




```
from sentence_transformers import SentenceTransformer
import pandas as pd

# Load the model (si no lo has cargado antes)
model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")

# Asegúrate de tener bien cargado el CSV con los valores correctos
df = pd.read_csv("/content/CDMX_Int_Sentiment_Comparation.csv", encoding="latin1")

# Clean and prepare
df["respuesta"] = df["respuesta"].astype(str)
df["Number"] = df["Number"].astype(str)

# Generate embeddings for each response individually
embeddings = model.encode(df["respuesta"].tolist(), convert_to_tensor=False)

# Add them as a new column
df["embedding"] = [e.tolist() for e in embeddings]

# Save the full DataFrame with embeddings
df.to_csv("/content/MTY_AllResponses_withEmbeddings.csv", index=False)
```

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Load data
df = pd.read_csv('/content/CDMX_Int_Sentiment_Comparation.csv', encoding='latin1')
df = df.dropna(subset=['respuesta', 'Number', 'tipo'])
df["respuesta"] = df["respuesta"].astype(str)
df["Number"] = df["Number"].astype(str)

# Load sentence embedding model
model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")
```

```
results = []

# Loop over each question
for q in sorted(df['Number'].unique()):
    df_q = df[df['Number'] == q]

    # Get all real responses
    real_responses = df_q[df_q['tipo'] == 'real']['respuesta'].dropna().tolist()

    if len(real_responses) < 2:
        continue  # not enough data to compute reference

    # Create reference embedding for the full set of real responses
    real_reference_text = " ".join(real_responses)
    ref_embedding = model.encode(real_reference_text, convert_to_tensor=True).unsqueeze(0)

    # Compare each generated response to real reference
    for r in df_q[df_q['tipo'] == 'generada']['respuesta'].dropna().tolist():
        emb_r = model.encode(r, convert_to_tensor=True).unsqueeze(0)
        score = cosine_similarity(ref_embedding.cpu().numpy(), emb_r.cpu().numpy())[0][0]
        results.append({'tipo': 'generada', 'pregunta': int(float(q)), 'cosine_semantico': score})

    # Leave-one-out for real responses
    real_list = df_q[df_q['tipo'] == 'real']['respuesta'].dropna().tolist()
    for i, r in enumerate(real_list):
        other_real = real_list[:i] + real_list[i+1:]
        if not other_real:
            continue
        ref_text = " ".join(other_real)
        ref_emb = model.encode(ref_text, convert_to_tensor=True).unsqueeze(0)
        emb_r = model.encode(r, convert_to_tensor=True).unsqueeze(0)
        score = cosine_similarity(ref_emb.cpu().numpy(), emb_r.cpu().numpy())[0][0]
        results.append({'tipo': 'real', 'pregunta': int(float(q)), 'cosine_semantico': score})

# Save and display
df_sem_sim = pd.DataFrame(results)
df_sem_sim.to_csv('/content/semantic_similarity_vs_question_set.csv', index=False)
df_sem_sim.head()
```

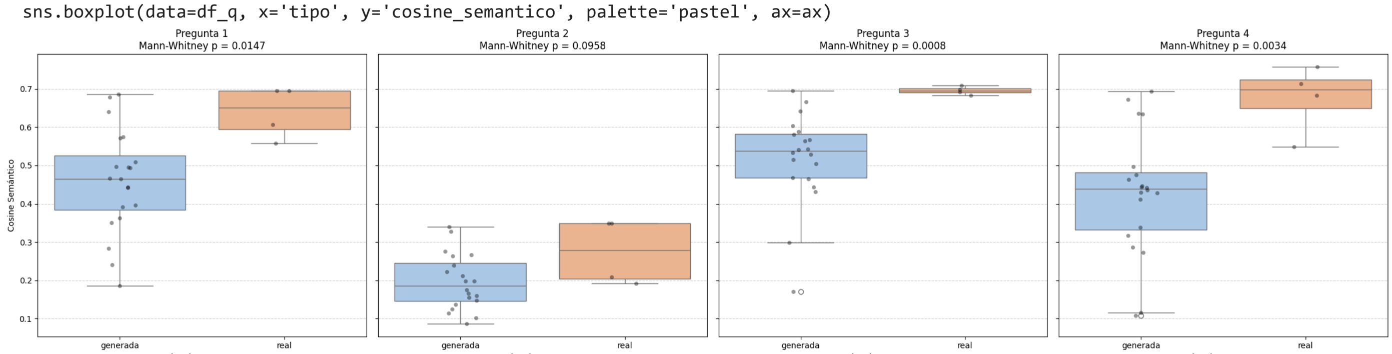
	tipo	pregunta	cosine_semantico
0	generada	1	0.496447
1	generada	1	0.464198
2	generada	1	0.240286
3	generada	1	0.509565
4	generada	1	0.186194

Pasos siguientes: Generar código con df_sem_sim | Ver gráficos recomendados | New interactive sheet

```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import mannwhitneyu

# Get unique question numbers
questions = df_sem_sim['pregunta'].unique()

# Set up figure for subplots
fig, axes = plt.subplots(1, len(questions), figsize=(6 * len(questions), 6), sharey=True)

# Store results
stat_results = []

# Loop over each question
for i, q in enumerate(sorted(questions)):
    ax = axes[i] if len(questions) > 1 else axes

    # Filter data for this question
    df_q = df_sem_sim[df_sem_sim['pregunta'] == q]
    real_scores = df_q[df_q['tipo'] == 'real']['cosine_semantico']
    gen_scores = df_q[df_q['tipo'] == 'generada']['cosine_semantico']

    # Mann-Whitney U test
```

```
    u_stat, p_value = mannwhitneyu(real_scores, gen_scores, alternative='two-sided')
    stat_results.append({'pregunta': q, 'U': u_stat, 'p_value': p_value})

    # Boxplot + stripplot
    sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
    sns.stripplot(data=df_q, x='tipo', y='cosine_semantico', color='black', alpha=0.4, ax=ax)

    ax.set_title(f'Pregunta {q}\nMann-Whitney p = {p_value:.4f}')
    ax.set_xlabel('Tipo de respuesta')
    ax.set_ylabel('Cosine Semántico' if i == 0 else '')
    ax.grid(axis='y', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()

# Print test results
print("\nResultados estadísticos por pregunta:")
for r in stat_results:
    signif = "✅ SIGNIFICATIVA" if r['p_value'] < 0.05 else "🔍 No significativa"
    print(f"Pregunta {r['pregunta']}: U = {r['U']}, p = {r['p_value']:.4f} → {signif}")
```

```
/tmp/ipython-input-14-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
/tmp/ipython-input-14-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
/tmp/ipython-input-14-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
/tmp/ipython-input-14-4270462877.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(data=df_q, x='tipo', y='cosine_semantico', palette='pastel', ax=ax)
```




```
Resultados estadísticos por pregunta:
Pregunta 1: U = 72.0, p = 0.0147 → ✅ SIGNIFICATIVA
Pregunta 2: U = 62.0, p = 0.0958 → 🔍 No significativa
Pregunta 3: U = 78.0, p = 0.0008 → ✅ SIGNIFICATIVA
Pregunta 4: U = 75.0, p = 0.0034 → ✅ SIGNIFICATIVA
```