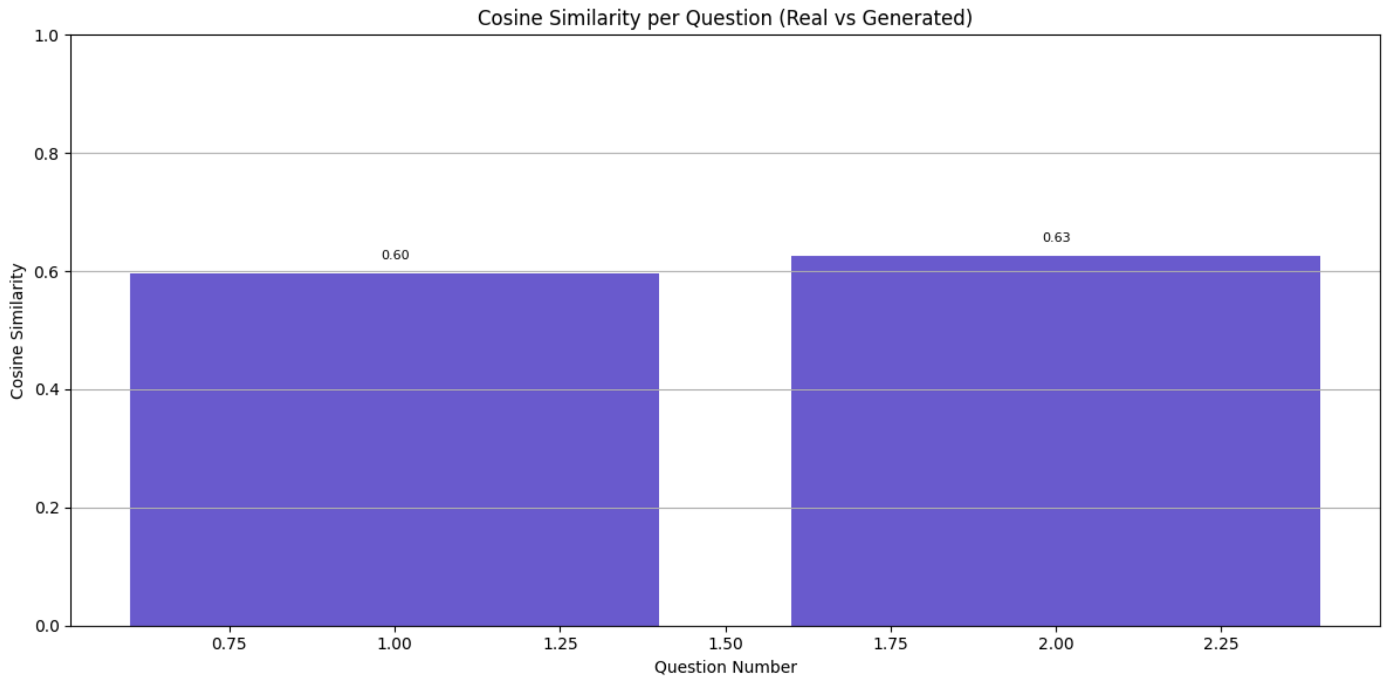
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load your DataFrame
df = pd.read_csv("/content/MTY_Sentiment_Comparation.csv", encoding='ISO-8859-1')

# Drop rows with missing question number or response
df_clean = df.dropna(subset=['Number', 'respuesta'])

# Group responses by question and type, and join all text into one string per group
grouped = df_clean.groupby(['Number', 'tipo'])['respuesta'].apply(lambda x: ' '.join(x)).unstack()

# Drop questions where either 'real' or 'generada' is missing
grouped = grouped.dropna(subset=['real', 'generada'])

# Initialize the TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Apply TF-IDF to both columns together
tfidf_matrix = vectorizer.fit_transform(grouped['real'].tolist() + grouped['generada'].tolist())

# Split into real and generated vectors
n = len(grouped)
tfidf_real = tfidf_matrix[:n]
tfidf_generated = tfidf_matrix[n:]

# Compute cosine similarity between each pair of real and generated texts
similarities = cosine_similarity(tfidf_real, tfidf_generated).diagonal()

# Add similarity scores to the grouped DataFrame
grouped['cosine_similarity'] = similarities

# Display the average similarity across all questions
average_similarity = grouped['cosine_similarity'].mean()
print(f'Average cosine similarity across all questions: {average_similarity:.3f}')

# Show the resulting DataFrame
grouped.head()
```

```
Average cosine similarity across all questions: 0.611
```

tipo	generada	real	cosine_similarity
Number			
1.0	pues yo veo que le falta comunicación, verdad?...	Lo que es Claudia Pues sí la veo muy preparada...	0.596412
2.0	Movilidad, tráfico y seguridad. Porque siempre...	Sí pues bueno, este, pues como ya lo mencionar...	0.625716

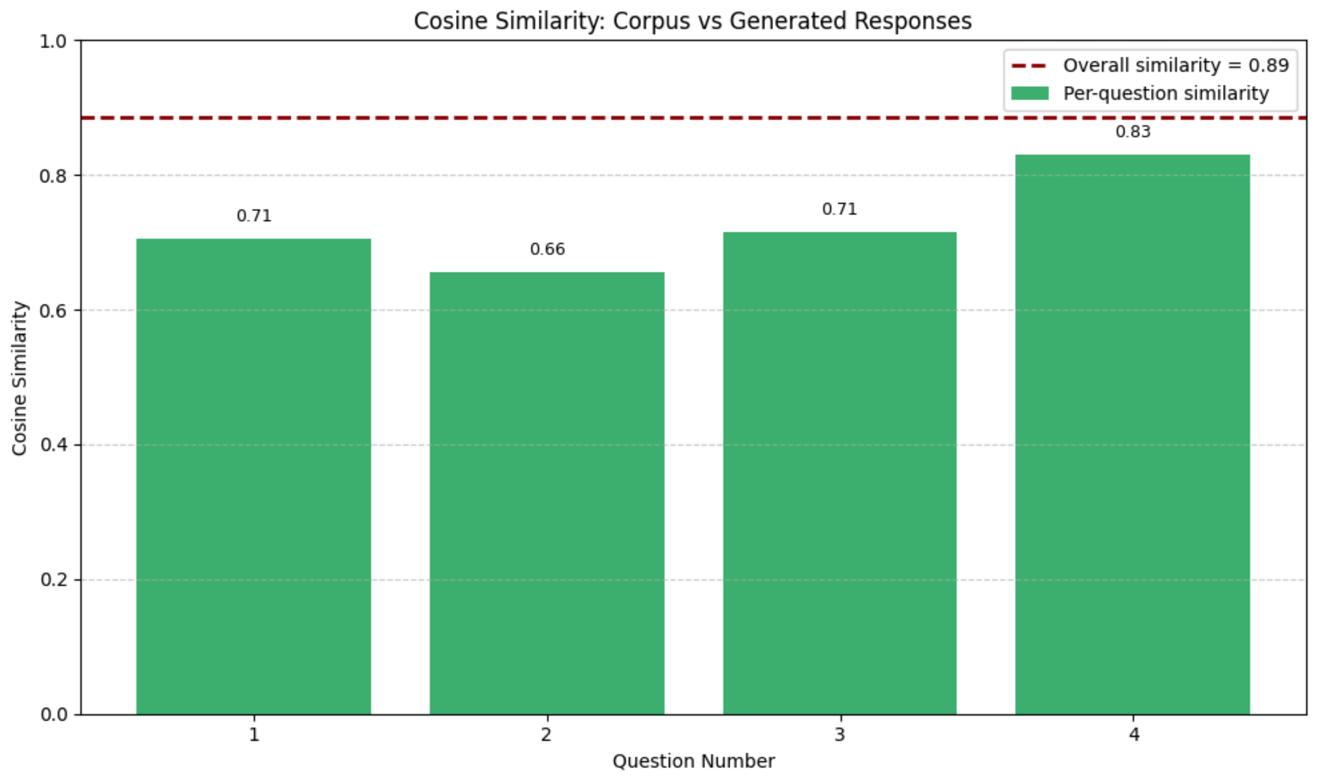
```
import matplotlib.pyplot as plt

# Sort the grouped DataFrame by similarity score
grouped_sorted = grouped.sort_values(by='cosine_similarity', ascending=False)

# Create a bar plot of cosine similarity per question
plt.figure(figsize=(12, 6))
plt.bar(grouped_sorted.index.astype(int), grouped_sorted['cosine_similarity'], color='slateblue')

# Add labels and title
plt.title('Cosine Similarity per Question (Real vs Generated)')
plt.xlabel('Question Number')
plt.ylabel('Cosine Similarity')
plt.ylim(0, 1)  # similarity scores range from 0 to 1
plt.grid(axis='y')

# Add value labels on top of bars (optional)
for i, v in enumerate(grouped_sorted['cosine_similarity']):
    plt.text(grouped_sorted.index[i], v + 0.02, f"{v:.2f}", ha='center', va='bottom', fontsize=8)

plt.tight_layout()
plt.show()
```



```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load data
df_corpus = pd.read_csv('/content/base_MTY_C+B_35-55.csv', encoding='latin1')
df_eval = pd.read_csv('/content/MTY_Sentiment_Comparation.csv', encoding='latin1')

# Get full corpus as single string
corpus_text = ' '.join(df_corpus['Participación'].dropna().astype(str).tolist())

# Get all generated responses as one string
generated_text = ' '.join(
    df_eval[df_eval['tipo'] == 'generada']['respuesta'].dropna().astype(str).tolist()
)

# Compare corpus vs generated (global)
texts_global = [corpus_text, generated_text]
vectorizer = TfidfVectorizer()
tfidf_global = vectorizer.fit_transform(texts_global)

similarity_generated = cosine_similarity(tfidf_global[0], tfidf_global[1])[0][0]
print(f"Cosine similarity: Corpus vs All Generated responses: {similarity_generated:.4f}")

# Similarity per question (generated only)
questions = df_eval['Number'].dropna().unique()
question_scores = []

for q in questions:
    generated_q = df_eval[(df_eval['Number'] == q) & (df_eval['tipo'] == 'generada')]['respuesta']
    if not generated_q.empty:
        generated_text_q = ' '.join(generated_q.dropna().astype(str).tolist())
        texts = [corpus_text, generated_text_q]
        tfidf_q = vectorizer.fit_transform(texts)
        sim = cosine_similarity(tfidf_q[0], tfidf_q[1])[0][0]
        question_scores.append({'Pregunta': int(q), 'Cosine_sim_corpus_vs_generated': sim})

# Convert to DataFrame and display
df_question_scores = pd.DataFrame(question_scores).sort_values(by='Pregunta')
df_question_scores.reset_index(drop=True, inplace=True)

print("\nCosine similarity por pregunta (corpus vs respuestas generadas):")
print(df_question_scores)
```

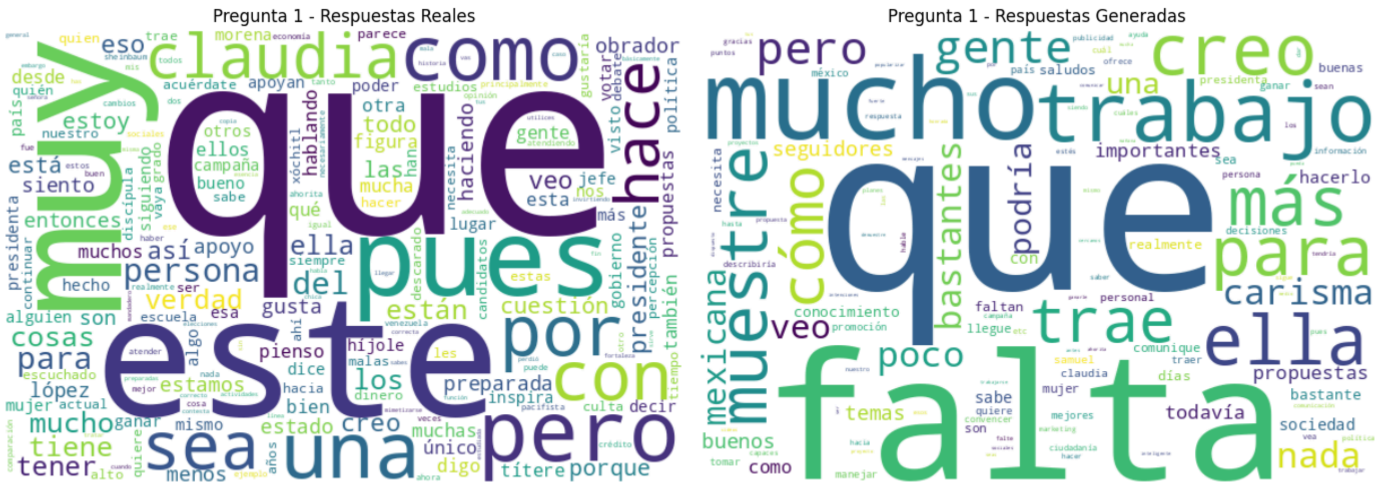
```
Cosine similarity: Corpus vs All Generated responses: 0.8860

Cosine similarity por pregunta (corpus vs respuestas generadas):
   Pregunta  Cosine_sim_corpus_vs_generated
0         1                        0.705047
1         2                        0.655358
2         3                        0.714308
3         4                        0.829190
```

```
import matplotlib.pyplot as plt

# Create a bar plot of cosine similarity per question
plt.figure(figsize=(10, 6))
plt.bar(
    df_question_scores['Pregunta'],
    df_question_scores['Cosine_sim_corpus_vs_generated'],
    color='mediumseagreen',
    label='Per-question similarity'
)

# Add horizontal line for overall similarity
plt.axhline(
    y=similarity_generated,
    color='darkred',
    linestyle='--',
    linewidth=2,
    label=f'Overall similarity = {similarity_generated:.2f}'
)

# Add labels and title
plt.title('Cosine Similarity: Corpus vs Generated Responses')
plt.xlabel('Question Number')
plt.ylabel('Cosine Similarity')
plt.ylim(0, 1)
plt.xticks(df_question_scores['Pregunta'])  # ensure integer labels on x-axis
plt.grid(axis='y', linestyle='--', alpha=0.6)

# Add text labels on top of bars
for idx, row in df_question_scores.iterrows():
    plt.text(row['Pregunta'], row['Cosine_sim_corpus_vs_generated'] + 0.02,
             f"{row['Cosine_sim_corpus_vs_generated']:.2f}",
             ha='center', va='bottom', fontsize=9)

# Show legend and plot
plt.legend()
plt.tight_layout()
plt.show()
```

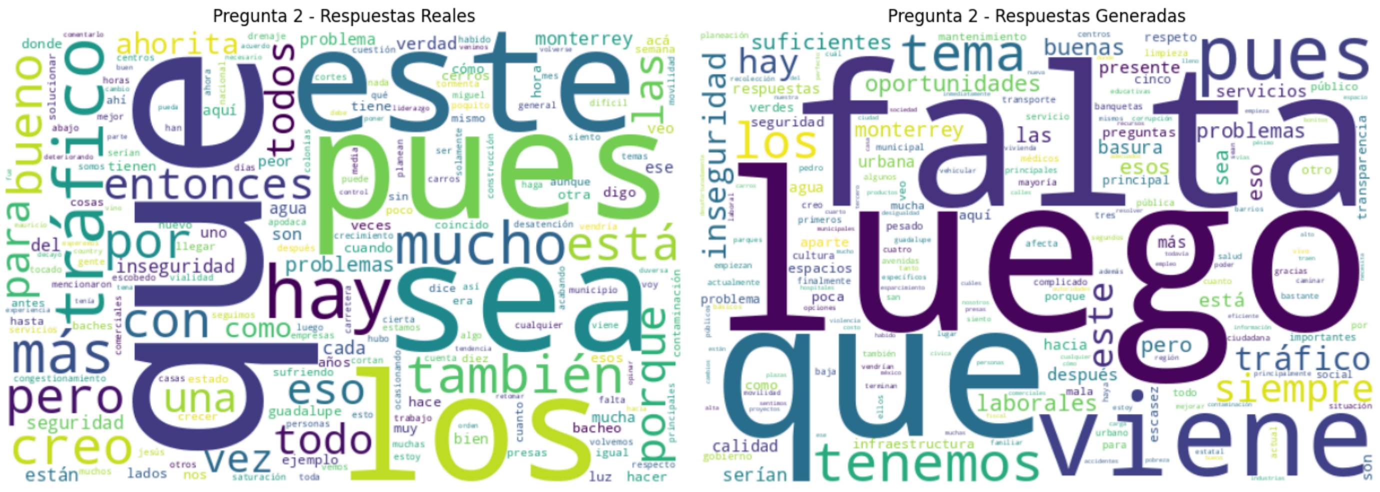


```
import pandas as pd
from collections import Counter
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import re

# Load data
df = pd.read_csv('/content/MTY_Sentiment_Comparation.csv', encoding='latin1')

# Function to clean and tokenize text
def clean_and_tokenize(text):
    text = re.sub(r'[^\w\s]', '', text.lower())  # lowercase and remove punctuation
    tokens = text.split()
    return [t for t in tokens if len(t) > 2 and not t.isnumeric()]  # filter short and numeric tokens

# Analyze question by number
for q in [1, 2]:
    print(f"\n=== Pregunta {q} ===")

    # Extract responses
    real_responses = df[(df['Number'] == q) & (df['tipo'] == 'real')]['respuesta'].dropna().astype(str)
    gen_responses = df[(df['Number'] == q) & (df['tipo'] == 'generada')]['respuesta'].dropna().astype(str)

    # Tokenize
    real_tokens = clean_and_tokenize(' '.join(real_responses))
    gen_tokens = clean_and_tokenize(' '.join(gen_responses))

    # Count frequency
    real_counter = Counter(real_tokens)
    gen_counter = Counter(gen_tokens)

    print("Top 10 palabras reales:", real_counter.most_common(10))
    print("Top 10 palabras generadas:", gen_counter.most_common(10))

    # Word clouds
    fig, axs = plt.subplots(1, 2, figsize=(14, 6))
    wc_real = WordCloud(width=600, height=400, background_color='white').generate_from_frequencies(real_counter)
    wc_gen = WordCloud(width=600, height=400, background_color='white').generate_from_frequencies(gen_counter)

    axs[0].imshow(wc_real, interpolation='bilinear')
    axs[0].axis('off')
    axs[0].set_title(f'Pregunta {q} - Respuestas Reales')

    axs[1].imshow(wc_gen, interpolation='bilinear')
    axs[1].axis('off')
    axs[1].set_title(f'Pregunta {q} - Respuestas Generadas')

    plt.tight_layout()
    plt.show()
```

```
=== Pregunta 1 ===
Top 10 palabras reales: [('que', 101), ('este', 30), ('muy', 25), ('pues', 23), ('pero', 23), ('una', 20), ('como', 19), ('por', 16), ('
Top 10 palabras generadas: [('que', 263), ('falta', 191), ('mucho', 55), ('trabajo', 48), ('creo', 35), ('ella', 34), ('para', 33), ('má
```



```
=== Pregunta 2 ===
Top 10 palabras reales: [('que', 83), ('pues', 25), ('sea', 25), ('los', 24), ('este', 23), ('hay', 19), ('tráfico', 17), ('mucho', 17),
Top 10 palabras generadas: [('luego', 75), ('falta', 70), ('que', 69), ('viene', 63), ('pues', 62), ('tenemos', 45), ('tema', 42), ('los
```



STADISTICAL TEST

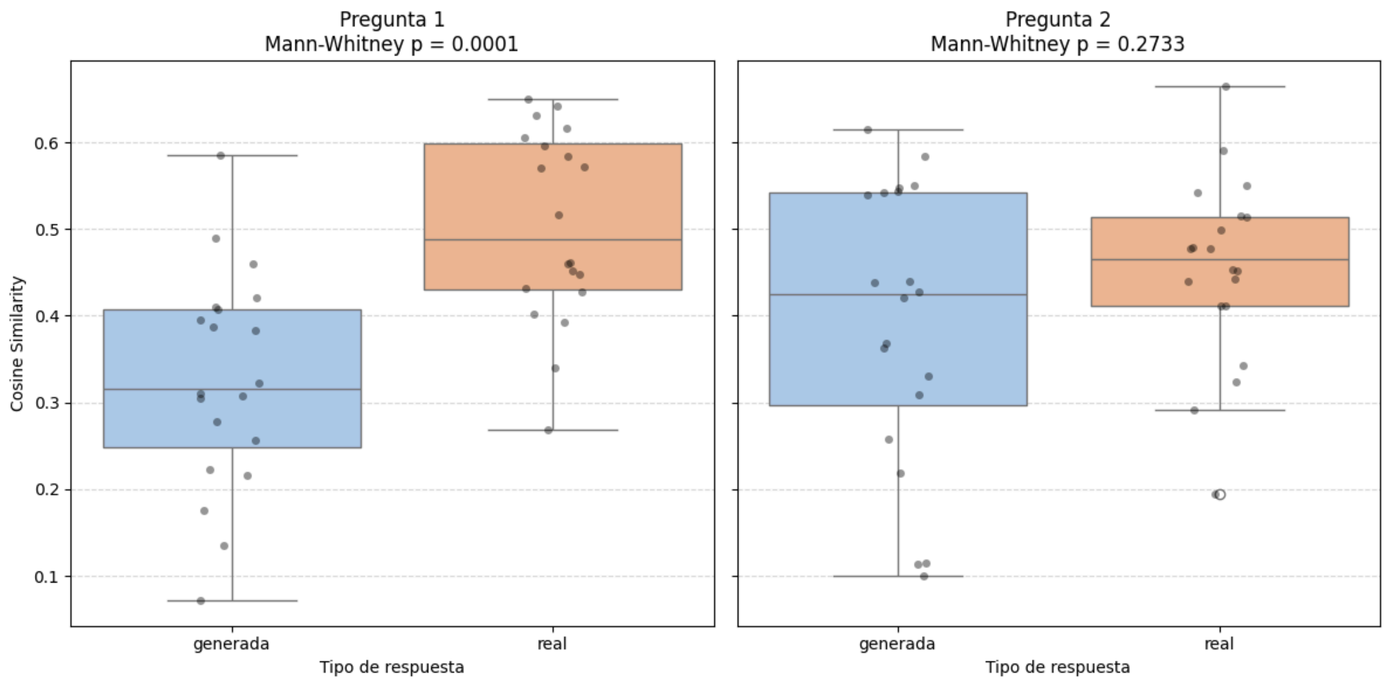
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load evaluation data
df = pd.read_csv('/content/MTY_Sentiment_Comparation.csv', encoding='latin1')

# Clean and keep only valid rows
df = df.dropna(subset=['respuesta', 'Number', 'tipo'])

# Prepare results
results = []


# Loop over each question (Number)
```

```
for q in sorted(df['Number'].unique()):
    df_q = df[df['Number'] == q]

    # Create global reference: all real responses for this question
    real_responses = df_q[df_q['tipo'] == 'real']['respuesta'].dropna().astype(str).tolist()

    if len(real_responses) < 2:
        continue  # skip if not enough data to build a reference set

    real_reference_text = ' '.join(real_responses)

    # Compare each generated response against all real responses
    for r in df_q[df_q['tipo'] == 'generada']['respuesta'].dropna().astype(str):
        texts = [real_reference_text, r]
        tfidf = TfidfVectorizer().fit_transform(texts)
        score = cosine_similarity(tfidf[0], tfidf[1])[0][0]
        results.append({'tipo': 'generada', 'pregunta': int(q), 'cosine_similarity': score})

    # Compare each real response against the rest (leave-one-out)
    real_list = df_q[df_q['tipo'] == 'real']['respuesta'].dropna().astype(str).tolist()
    for i, r in enumerate(real_list):
        # Create reference excluding the current response
        other_real = real_list[:i] + real_list[i+1:]
        if not other_real:
            continue
        ref_text = ' '.join(other_real)
        texts = [ref_text, r]
        tfidf = TfidfVectorizer().fit_transform(texts)
        score = cosine_similarity(tfidf[0], tfidf[1])[0][0]
        results.append({'tipo': 'real', 'pregunta': int(q), 'cosine_similarity': score})

# Create DataFrame with all results
df_sim_qset = pd.DataFrame(results)
print(df_sim_qset.head())

# Save as CSV
df_sim_qset.to_csv('/content/similarity_vs_question_set.csv', index=False)
```

```
       tipo  pregunta  cosine_similarity
0  generada         1           0.310035
1  generada         1           0.278295
2  generada         1           0.305559
3  generada         1           0.409566
4  generada         1           0.383323
```

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

# Get unique question numbers
questions = df_sim_qset['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_sim_qset[df_sim_qset['pregunta'] == q]
    real_scores = df_q[df_q['tipo'] == 'real']['cosine_similarity']
    gen_scores = df_q[df_q['tipo'] == 'generada']['cosine_similarity']

    # 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
    sns.boxplot(data=df_q, x='tipo', y='cosine_similarity', palette='pastel', ax=ax)
    sns.stripplot(data=df_q, x='tipo', y='cosine_similarity', 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 Similarity' 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-11-3516458435.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_similarity', palette='pastel', ax=ax)
/tmp/ipython-input-11-3516458435.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_similarity', palette='pastel', ax=ax)
```



```
Resultados estadísticos por pregunta:
Pregunta 1: U = 344.0, p = 0.0001 → ✅ SIGNIFICATIVA
Pregunta 2: U = 241.0, p = 0.2733 → 🔍 No significativa
```

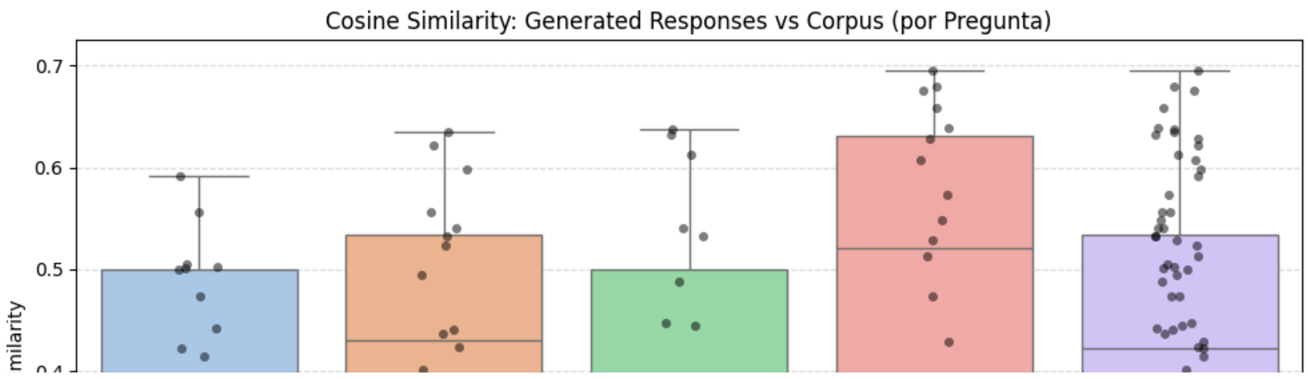
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load corpus and responses
df_corpus = pd.read_csv('/content/base_MTY_C+B_35-55.csv', encoding='latin1')
df_eval = pd.read_csv('/content/MTY_Sentiment_Comparation.csv', encoding='latin1')

# Build full corpus text
corpus_text = ' '.join(df_corpus['Participación'].dropna().astype(str).tolist())

# Filter only generated responses with valid question number
df_gen = df_eval[(df_eval['tipo'] == 'generada') & (df_eval['Number'].notna())]

# Compute cosine similarity for each generated response vs full corpus
```

```
results = []
for _, row in df_gen.iterrows():
    response_text = str(row['respuesta'])
    question_num = int(row['Number'])
    texts = [corpus_text, response_text]
    tfidf = TfidfVectorizer().fit_transform(texts)
    score = cosine_similarity(tfidf[0], tfidf[1])[0][0]
    results.append({'pregunta': question_num, 'cosine_similarity': score})

# Create DataFrame
df_sim_vs_corpus = pd.DataFrame(results)

# Add a column to group everything under "all" for global analysis
df_sim_vs_corpus_all = df_sim_vs_corpus.copy()
df_sim_vs_corpus_all['pregunta'] = 'Todas'

# Concatenate per-pregunta and all
df_sim_full = pd.concat([df_sim_vs_corpus, df_sim_vs_corpus_all], ignore_index=True)

# Display sample
df_sim_full.head()
```

	pregunta	cosine_similarity
0	1	0.414557
1	1	0.378479
2	1	0.369269
3	1	0.500607
4	1	0.499905

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

# Filtrar preguntas únicas (sin duplicar el grupo 'Todas')
preguntas = sorted(df_sim_vs_corpus['pregunta'].unique())
grupo_completo = df_sim_full[df_sim_full['pregunta'] == 'Todas']['cosine_similarity']

# Graficar boxplot por pregunta + grupo completo
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_sim_full, x='pregunta', y='cosine_similarity', palette='pastel')
sns.stripplot(data=df_sim_full, x='pregunta', y='cosine_similarity', color='black', alpha=0.5)

plt.title('Cosine Similarity: Generated Responses vs Corpus (por Pregunta)')
plt.xlabel('Pregunta')
plt.ylabel('Cosine Similarity')
plt.grid(axis='y', linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()

# Aplicar Mann-Whitney U test por cada pregunta vs grupo 'Todas'
print("Mann-Whitney U test (por pregunta vs conjunto total):")
for preg in preguntas:
    scores_preg = df_sim_full[df_sim_full['pregunta'] == preg]['cosine_similarity']
    u_stat, p_val = mannwhitneyu(scores_preg, grupo_completo, alternative='two-sided')
    signif = "✅ SIGNIFICATIVA" if p_val < 0.05 else "🔍 No significativa"
    print(f"Pregunta {preg}: U = {u_stat}, p = {p_val:.4f} → {signif}")
```

```
/tmp/ipython-input-14-1636074905.py:11: 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_sim_full, x='pregunta', y='cosine_similarity', palette='pastel')
```


Cosine Similarity: Generated Responses vs Corpus (por Pregunta)
milarity