

```

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_Int_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: 0.492

	tipo	generada	real	cosine_similarity
Number				
1.0	pues yo veo que le falta comunicación, verdad?...	Ay yo me di el tiempo me di el tiempo de leer ...		0.429709
2.0	Movilidad, tráfico y seguridad. Porque siempre...	Mira Bueno A mi modo de ver yo siento que la g...		0.529842
3.0	ok buenas tardes soy david tengo treinta y sei...	rimero el pueblo Pues es básicamente lo que pi...		0.442684
4.0	yo creo que es un mensaje bonito, pero dime, p...	No vivimos de apoyos. Vivimos el esfuerzo, yo ...		0.565725

```

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')

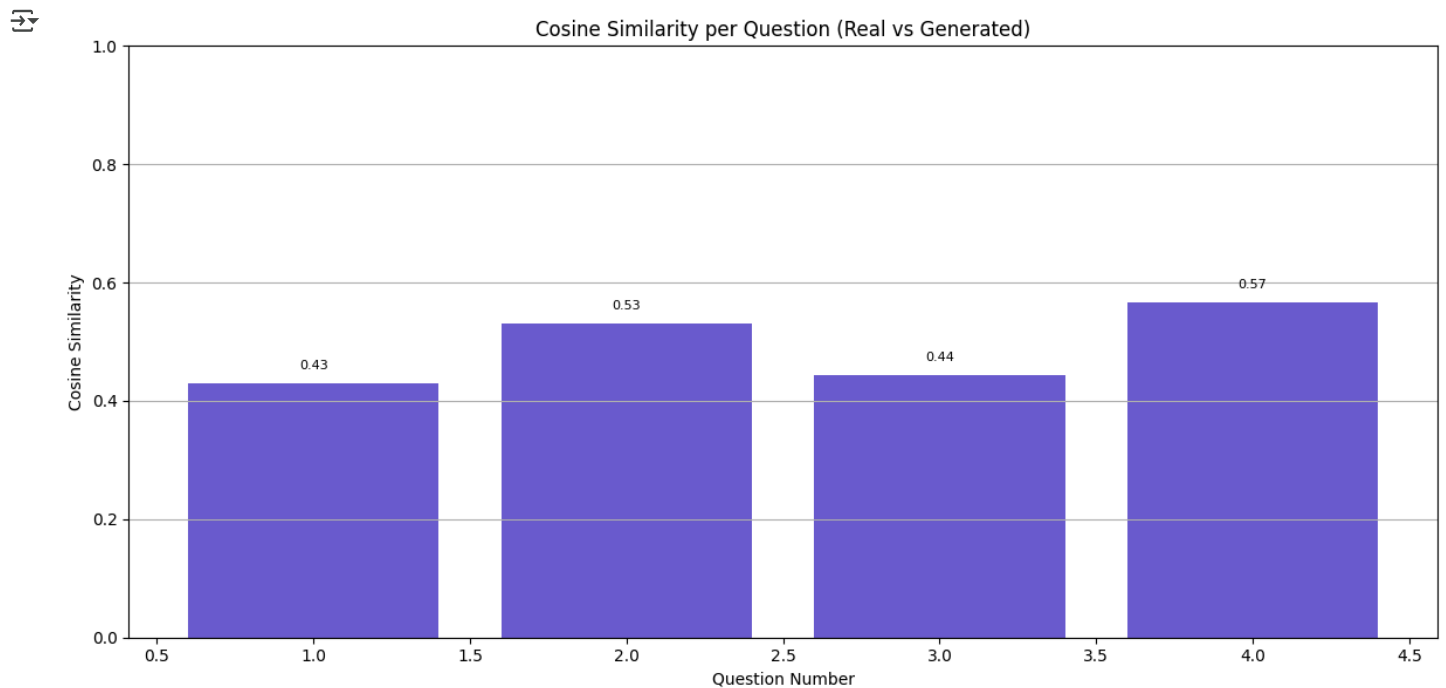
# Add labels and title
plt.title('Cosine Similarity per Question (Real vs Generated)')
plt.xlabel('Question Number')
plt.ylabel('Cosine Similarity')
plt.ylim(0, 1) # similarity scores range from 0 to 1
plt.grid(axis='y')

# 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 evaluation data
df = pd.read_csv('/content/MTY_Int_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	0.261936
1	generada	1	0.298358
2	generada	1	0.241645
3	generada	1	0.309553
4	generada	1	0.331421

```
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-10-3516458435.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-10-3516458435.py:28: FutureWarning:
```

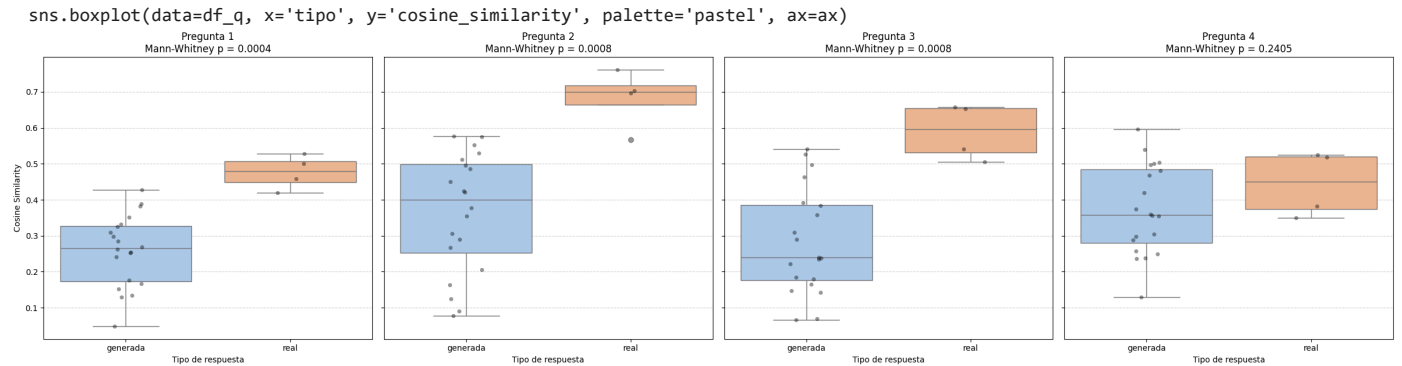
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.boxplot(data=df_q, x='tipo', y='cosine_similarity', palette='pastel', ax=ax)
/tmp/ipython-input-10-3516458435.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-10-3516458435.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`



Resultados estadísticos por pregunta:

Pregunta 1: U = 79.0, p = 0.0004 → ✔ SIGNIFICATIVA
 Pregunta 2: U = 78.0, p = 0.0008 → ✔ SIGNIFICATIVA
 Pregunta 3: U = 78.0, p = 0.0008 → ✔ SIGNIFICATIVA
 Pregunta 4: U = 56.0, p = 0.2405 → ❓ No significativa

CDMX

```
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/CDMX_Int_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: 0.567

	tipo	generada	real	cosine_similarity
Number				
1.0	yo la veo como una persona inteligente o intel...	Pues yo Bueno yo tenía una ideología casi pare...		0.664892
2.0	principal problema es la inseguridad. segundo ...	En mi lo Pues bueno, bueno, Te voy a contar un...		0.354434
3.0	es una meta que se plantea el presidente claud...	primero el pueblo Pues yo creo que es un eslog...		0.579890
4.0	Claro, el pueblo primero, no los de arriba. Es...	Sí también está bueno, porque pues realmente n...		0.669012

```
import matplotlib.pyplot as plt

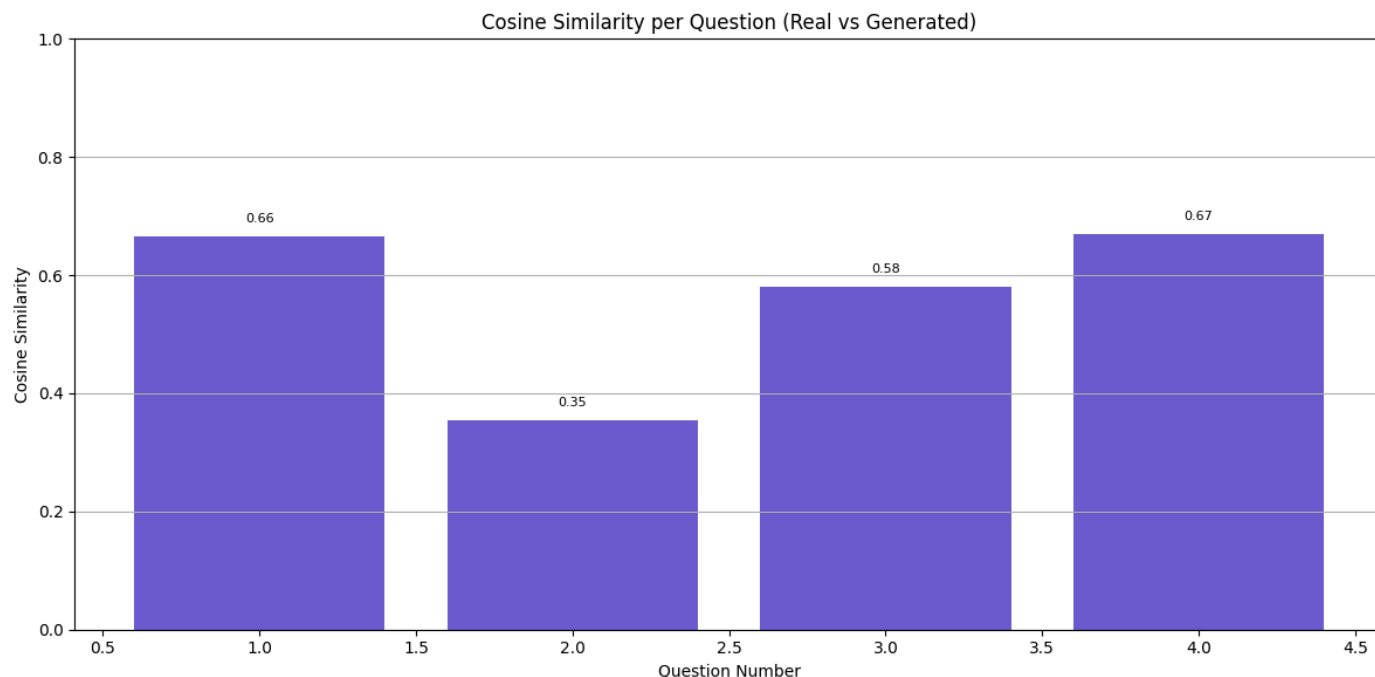
# Sort the grouped DataFrame by similarity score
grouped_sorted = grouped.sort_values(by='cosine_similarity', ascending=False)

# 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 evaluation data
df = pd.read_csv('/content/CDMX_Int_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         0.537050
1  generada         1         0.423308
2  generada         1         0.536698
3  generada         1         0.371851
4  generada         1         0.552183
```

```
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-14-3516458435.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-14-3516458435.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-14-3516458435.py:28: FutureWarning: