```
!pip install -q transformers
!pip install -q torch
!pip install -q sentencepiece
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 117.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 101.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 53.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 44.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 20.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 94.3 MB/s eta 0:00:00
```

```
import pandas as pd
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from scipy.special import softmax
import torch

df_mty = pd.read_csv("/content/MTY_Answers_Entrevistas.csv")
df_cdmx = pd.read_csv("/content/CDMX_Answers_interview.csv")

# Load Model and tokenizer
MODEL = "cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```

```
/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(
config.json: 100%  747/747 [00:00<00:00, 95.8kB/s]
vocab.json:  899k/? [00:00<00:00, 14.1MB/s]
merges.txt:  456k/? [00:00<00:00, 37.2MB/s]
special_tokens_map.json: 100%  150/150 [00:00<00:00, 19.3kB/s]
pytorch_model.bin: 100%  499M/499M [00:02<00:00, 395MB/s]
```

```
# Function to clasify sentiment
def get_sentiment_score(text):
    try:
        if pd.isna(text) or text.strip() == "":
            return pd.Series([0.0, 1.0, 0.0, 0.0], index=['score_neg', 'score_neu', 'score_pos', 'sentimiento_real'])

        encoded_input = tokenizer(text, return_tensors='pt', truncation=True, max_length=512)
        with torch.no_grad():
            output = model(**encoded_input)
        scores = softmax(output.logits[0].numpy())
        sentiment_score = scores[2] - scores[0]  # positive - negative
        return pd.Series([scores[0], scores[1], scores[2], sentiment_score], index=['score_neg', 'score_neu', 'score_pos', 'sentimiento_real

    except Exception as e:
        print(f"Error procesando texto: {text[:30]}... — {str(e)}")
        return pd.Series([0.0, 1.0, 0.0, 0.0], index=['score_neg', 'score_neu', 'score_pos', 'sentimiento_real'])
```

```
df_mty[['score_neg', 'score_neu', 'score_pos', 'sentimiento_real']] = df_mty['Answer'].apply(get_sentiment_score)
df_mty.to_csv("/content/MTY_Answers_entrevistas_sentimient.csv", index=False)

df_cdmx[['score_neg', 'score_neu', 'score_pos', 'sentimiento_real']] = df_cdmx['Answer'].apply(get_sentiment_score)
df_cdmx.to_csv("/content/CDMX_Answers_entrevistas_sentimient.csv", index=False)
```

```
model.safetensors: 100%  499M/499M [00:03<00:00, 181MB/s]
```

```
# Load the file with generated answers
df_mty_gen = pd.read_csv("/content/MTY_Answers_generated.csv", encoding='ISO-8859-1')
df_cdmx_gen = pd.read_csv("/content/CDMX_Answers_generated.csv", encoding='ISO-8859-1')
```

```
# Apply sentiment analysis to the 'Answer_generated' column
df_mty_gen[['score_neg', 'score_neu', 'score_pos', 'sentimiento_generado']] = df_mty_gen['Answer_generated'].apply(get_sentiment_score)
df_cdmx_gen[['score_neg', 'score_neu', 'score_pos', 'sentimiento_generado']] = df_cdmx_gen['Answer_generated'].apply(get_sentiment_score)

# Save the updated file with sentiment scores
df_mty_gen.to_csv("/content/MTY_Answers_generated_sentimient.csv", index=False)
df_cdmx_gen.to_csv("/content/CDMX_Answers_generated_sentimient.csv", index=False)
```

```
import pandas as pd

# Load real and generated responses (with sentiment scores)
df_mty_int = pd.read_csv("/content/MTY_Answers_entrevistas_sentimient.csv")
df_mty_gen = pd.read_csv("/content/MTY_Answers_generated_sentimient.csv")

df_cdmx_int = pd.read_csv("/content/CDMX_Answers_entrevistas_sentimient.csv")
df_cdmx_gen = pd.read_csv("/content/CDMX_Answers_generated_sentimient.csv")

# Group by 'pregunta' and calculate average sentiment
mty_avg_sent_real = df_mty_int.groupby("Number")["sentimiento_real"].mean().reset_index()
mty_avg_sent_real.rename(columns={"sentimiento_real": "avg_sentimiento_real"}, inplace=True)

mty_avg_sent_gen = df_mty_gen.groupby("Number")["sentimiento_generado"].mean().reset_index()
mty_avg_sent_gen.rename(columns={"sentimiento_generado": "avg_sentimiento_generado"}, inplace=True)

cdmx_avg_sent_real = df_cdmx_int.groupby("Number")["sentimiento_real"].mean().reset_index()
cdmx_avg_sent_real.rename(columns={"sentimiento_real": "avg_sentimiento_real"}, inplace=True)

cdmx_avg_sent_gen = df_cdmx_gen.groupby("Number")["sentimiento_generado"].mean().reset_index()
cdmx_avg_sent_gen.rename(columns={"sentimiento_generado": "avg_sentimiento_generado"}, inplace=True)


# Merge both results on the question
mty_merged_avg = pd.merge(mty_avg_sent_real, mty_avg_sent_gen, on="Number")
cdmx_merged_avg = pd.merge(cdmx_avg_sent_real, cdmx_avg_sent_gen, on="Number")

# Display or export results
print(mty_merged_avg)
print(cdmx_merged_avg)
mty_merged_avg.to_csv("/content/mty_avg_sentimiento_comparado.csv", index=False)
cdmx_merged_avg.to_csv("/content/cdmx_avg_sentimiento_comparado.csv", index=False)
```

```
   Number  avg_sentimiento_real  avg_sentimiento_generado
0     1.0             -0.164309                 -0.112690
1     2.0             -0.075175                 -0.158285
2     3.0             -0.130239                 -0.072299
3     4.0             -0.058371                 -0.135207
   Number  avg_sentimiento_real  avg_sentimiento_generado
0     1.0             -0.070763                 -0.118984
1     2.0             -0.099033                 -0.230552
2     3.0             -0.057069                 -0.081162
3     4.0             -0.089282                 -0.127978
```

```
import pandas as pd

# Load both real and generated datasets
df_real = pd.read_csv("/content/CDMX_Answers_entrevistas_sentimient.csv")
df_gen = pd.read_csv("/content/CDMX_Answers_generated_sentimient.csv")

# Add a column to indicate source
df_real["tipo"] = "real"
df_gen["tipo"] = "generada"

# Standardize sentiment column names
df_real.rename(columns={"sentimiento_real": "sentimiento"}, inplace=True)
df_gen.rename(columns={"sentimiento_generado": "sentimiento"}, inplace=True)

# Standardize answer column names (optional)
df_real.rename(columns={"Answer": "respuesta"}, inplace=True)
df_gen.rename(columns={"Answer_generated": "respuesta"}, inplace=True)

# Select only relevant columns
cols = ["Number", "respuesta", "sentimiento", "tipo"]
df_combined = pd.concat([df_real[cols], df_gen[cols]], ignore_index=True)
```

```
# Compute average sentiment per question and type
avg_by_group = df_combined.groupby(["Number", "tipo"])["sentimiento"].mean().reset_index()
avg_by_group.rename(columns={"sentimiento": "avg_sentimiento"}, inplace=True)

# Merge the average back to each row
df_final = pd.merge(df_combined, avg_by_group, on=["Number", "tipo"], how="left")

# Save the final combined file
df_final.to_csv("/content/CDMX_Int_Sentiment_Comparation.csv", index=False)
```

```
import pandas as pd

# Load both real and generated datasets
df_real = pd.read_csv("/content/MTY_Answers_entrevistas_sentimient.csv")
df_gen = pd.read_csv("/content/MTY_Answers_generated_sentimient.csv")

# Add a column to indicate source
df_real["tipo"] = "real"
df_gen["tipo"] = "generada"

# Standardize sentiment column names
df_real.rename(columns={"sentimiento_real": "sentimiento"}, inplace=True)
df_gen.rename(columns={"sentimiento_generado": "sentimiento"}, inplace=True)

# Standardize answer column names (optional)
df_real.rename(columns={"Answer": "respuesta"}, inplace=True)
df_gen.rename(columns={"Answer_generated": "respuesta"}, inplace=True)

# Select only relevant columns
cols = ["Number", "respuesta", "sentimiento", "tipo"]
df_combined = pd.concat([df_real[cols], df_gen[cols]], ignore_index=True)

# Compute average sentiment per question and type
avg_by_group = df_combined.groupby(["Number", "tipo"])["sentimiento"].mean().reset_index()
avg_by_group.rename(columns={"sentimiento": "avg_sentimiento"}, inplace=True)

# Merge the average back to each row
df_final = pd.merge(df_combined, avg_by_group, on=["Number", "tipo"], how="left")

# Save the final combined file
df_final.to_csv("/content/MTY_Sentiment_Comparation.csv", index=False)
```

STADISTICAL TEST

```
import pandas as pd
from scipy.stats import ttest_rel

# Load the CSV file
df = pd.read_csv('/content/CDMX_Int_Sentiment_Comparation.csv')

# Initialize a list to collect results for each question
results = []

# Loop over each unique question number
for q_num in sorted(df['Number'].dropna().unique()):
    # Filter data for the current question
    q_data = df[df['Number'] == q_num]

    # Separate real and generated sentiment values
    real_q = q_data[q_data['tipo'] == 'real']['sentimiento'].reset_index(drop=True)
    gen_q = q_data[q_data['tipo'] == 'generada']['sentimiento'].reset_index(drop=True)

    # Make sure we compare only pairs of responses
    min_len = min(len(real_q), len(gen_q))
    if min_len == 0:
        continue  # Skip questions that don't have both real and generated responses

    real_q = real_q[:min_len]
    gen_q = gen_q[:min_len]

    # Perform paired t-test
    t_stat, p_val = ttest_rel(real_q, gen_q)
```

```
    # Save the results
    results.append({
        "Question (Number)": int(q_num),
        "N pairs": min_len,
        "Mean Real": real_q.mean(),
        "Mean Generated": gen_q.mean(),
        "Mean Difference": gen_q.mean() - real_q.mean(),
        "t-statistic": t_stat,
        "p-value": p_val,
        "Significant (p < 0.05)": p_val < 0.05
    })

# Convert the list of results to a DataFrame
results_df = pd.DataFrame(results)

# Display the result
results_df
```

	Question (Number)	N pairs	Mean Real	Mean Generated	Mean Difference	t-statistic	p-value	Significant (p < 0.05)
0	1	4	-0.070763	-0.115397	-0.044633	2.820731	0.066698	False
1	2	4	-0.099033	-0.346343	-0.247310	4.169098	0.025118	True
2	3	4	-0.057069	-0.086258	-0.029189	0.540252	0.626552	False
3	4	4	-0.089282	-0.110439	-0.021157	0.731543	0.517411	False

```
from scipy.stats import wilcoxon

# Initialize a list to collect Wilcoxon results
wilcoxon_results = []

# Loop over each unique question number
for q_num in sorted(df['Number'].dropna().unique()):
    # Filter data for the current question
    q_data = df[df['Number'] == q_num]

    # Separate real and generated sentiment values
    real_q = q_data[q_data['tipo'] == 'real']['sentimiento'].reset_index(drop=True)
    gen_q = q_data[q_data['tipo'] == 'generada']['sentimiento'].reset_index(drop=True)

    # Match the length for paired comparison
    min_len = min(len(real_q), len(gen_q))
    if min_len == 0:
        continue  # Skip questions with missing pairs

    real_q = real_q[:min_len]
    gen_q = gen_q[:min_len]

    # Perform Wilcoxon signed-rank test
    try:
        stat, p_val = wilcoxon(real_q, gen_q)
    except ValueError:
        # If all differences are zero or the sample is too small, skip
        continue

    # Store the result
    wilcoxon_results.append({
        "Question (Number)": int(q_num),
        "N pairs": min_len,
        "Mean Real": real_q.mean(),
        "Mean Generated": gen_q.mean(),
        "Mean Difference": gen_q.mean() - real_q.mean(),
        "Wilcoxon statistic": stat,
        "p-value": p_val,
        "Significant (p < 0.05)": p_val < 0.05
    })

# Convert to DataFrame
wilcoxon_df = pd.DataFrame(wilcoxon_results)

# Display the result
wilcoxon_df
```

	Question (Number)	N pairs	Mean Real	Mean Generated	Mean Difference	Wilcoxon statistic	p-value	Significant (p < 0.05)
0	1	4	-0.070763	-0.115397	-0.044633	1.0	0.250	False
1	2	4	-0.099033	-0.346343	-0.247310	0.0	0.125	False
2	3	4	-0.057069	-0.086258	-0.029189	3.0	0.625	False
3	4	4	-0.089282	-0.110439	-0.021157	3.0	0.625	False

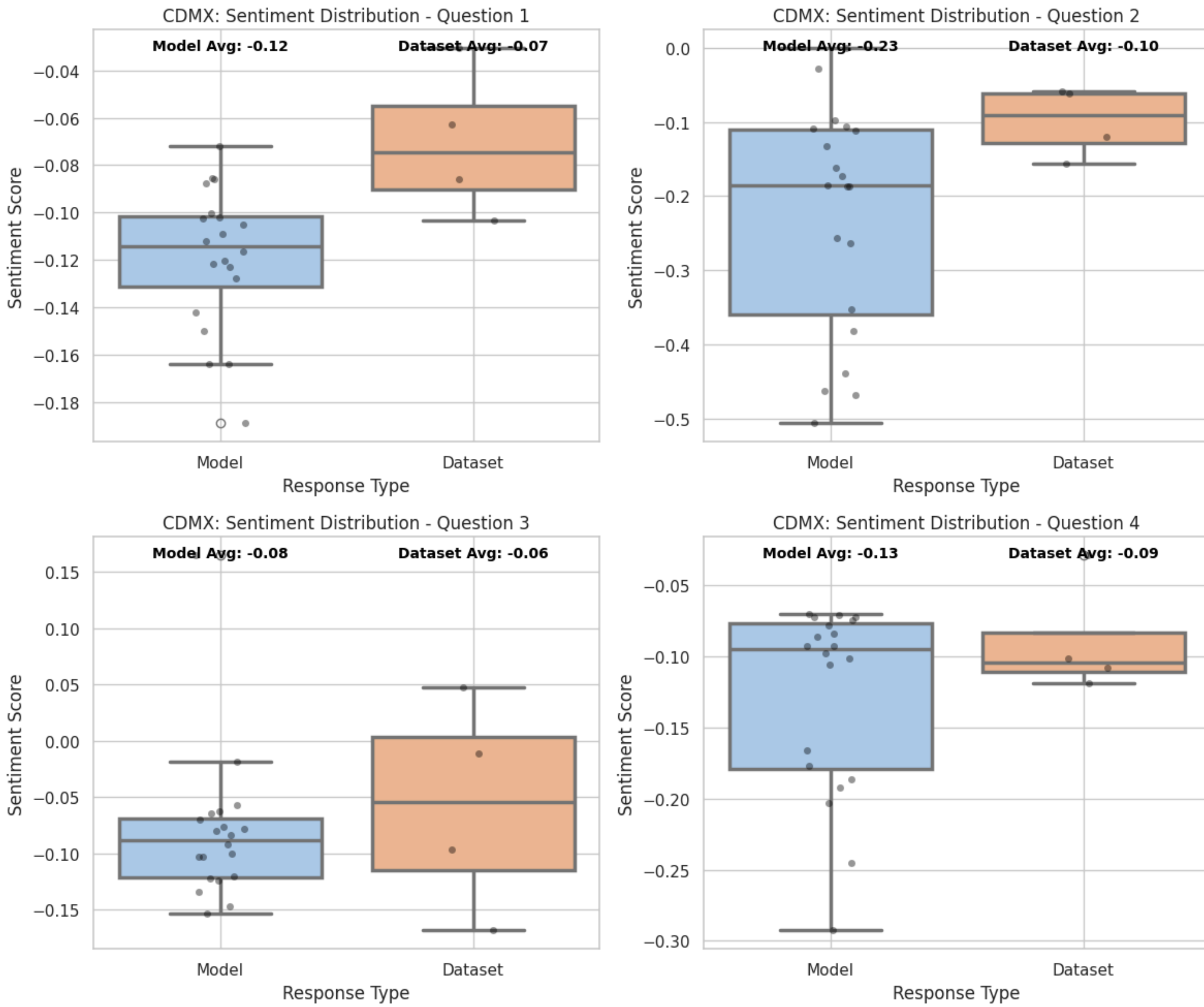
```
from scipy.stats import ks_2samp

# Initialize a list to collect KS test results
ks_results = []

# Loop through each unique question
for q_num in sorted(df['Number'].dropna().unique()):
    # Filter data for the current question
    q_data = df[df['Number'] == q_num]

    # Separate sentiment values
    real_q = q_data[q_data['tipo'] == 'real']['sentimiento'].reset_index(drop=True)
    gen_q = q_data[q_data['tipo'] == 'generada']['sentimiento'].reset_index(drop=True)

    # Perform K-S test only if both have values
    if len(real_q) > 0 and len(gen_q) > 0:
        ks_stat, p_val = ks_2samp(real_q, gen_q)
        ks_results.append({
            "Question (Number)": int(q_num),
            "N Real": len(real_q),
            "N Generated": len(gen_q),
            "Mean Real": real_q.mean(),
            "Mean Generated": gen_q.mean(),
            "K-S statistic": ks_stat,
            "p-value": p_val,
            "Significant (p < 0.05)": p_val < 0.05
        })

# Convert results to DataFrame
ks_df = pd.DataFrame(ks_results)

# Display the table
ks_df
```

	Question (Number)	N Real	N Generated	Mean Real	Mean Generated	K-S statistic	p-value	Significant (p < 0.05)
0	1	4	20	-0.070763	-0.118984	0.65	0.086580	False
1	2	4	20	-0.099033	-0.230552	0.65	0.086580	False
2	3	4	20	-0.057069	-0.081162	0.45	0.447770	False
3	4	4	20	-0.089282	-0.127978	0.35	0.749482	False

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set style for consistency
sns.set(style="whitegrid")

# Load the data (assuming df is already loaded and processed)
# df = pd.read_csv('/content/CDMX_Int_Sentiment_Comparation.csv') # Assuming this was done earlier

# Get unique question numbers
question_numbers = sorted(df['Number'].dropna().unique())

# Create a figure and a set of subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten() # Flatten the 2x2 array of axes for easy iteration

# Loop through each question and plot on a different subplot
for i, q_num in enumerate(question_numbers):
    q_data = df[df['Number'] == q_num].copy()

    if q_data['tipo'].nunique() < 2:
        continue
```

```
    # Rename labels for clarity
    q_data['tipo'] = q_data['tipo'].replace({'real': 'Dataset', 'generada': 'Model'})

    # Boxplot
    ax = sns.boxplot(
        data=q_data,
        x='tipo',
        y='sentimiento',
        order=['Model', 'Dataset'],
        palette='pastel',
        linewidth=2.5,
        ax=axes[i] # Specify the subplot to draw on
    )

    # Stripplot
    sns.stripplot(
        data=q_data,
        x='tipo',
        y='sentimiento',
        order=['Model', 'Dataset'],
        color='black',
        alpha=0.4,
        ax=axes[i] # Specify the subplot to draw on
    )

    # Calculate and add average sentiment text to the side
    avg_sentiment_model = q_data[q_data['tipo'] == 'Model']['sentimiento'].mean()
    avg_sentiment_dataset = q_data[q_data['tipo'] == 'Dataset']['sentimiento'].mean()

    # Adjust y-coordinate based on the y-axis limits
    y_lim = ax.get_ylim()
    y_offset = (y_lim[1] - y_lim[0]) * 0.05 # 5% of the y-axis range as offset

    ax.text(0, y_lim[1] - y_offset, f'Model Avg: {avg_sentiment_model:.2f}',
            horizontalalignment='center', size='small', color='black', weight='semibold')
    ax.text(1, y_lim[1] - y_offset, f'Dataset Avg: {avg_sentiment_dataset:.2f}',
            horizontalalignment='center', size='small', color='black', weight='semibold')

    axes[i].set_title(f'CDMX: Sentiment Distribution - Question {int(q_num)}')
    axes[i].set_xlabel('Response Type')
    axes[i].set_ylabel('Sentiment Score')
    axes[i].grid(True)

# Adjust layout to prevent overlapping titles and labels
plt.tight_layout()

# Show the combined plot
plt.show()
```

```
/tmp/ipython-input-12-1570855158.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

  ax = sns.boxplot(
/tmp/ipython-input-12-1570855158.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

  ax = sns.boxplot(
/tmp/ipython-input-12-1570855158.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

  ax = sns.boxplot(
/tmp/ipython-input-12-1570855158.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

  ax = sns.boxplot(
```



MTY:

```
import pandas as pd
from scipy.stats import ttest_rel
```

```
# Load the CSV file
df = pd.read_csv('/content/MTY_Sentiment_Comparation.csv')

# Initialize a list to collect results for each question
results = []

# Loop over each unique question number
for q_num in sorted(df['Number'].dropna().unique()):
    # Filter data for the current question
    q_data = df[df['Number'] == q_num]

    # Separate real and generated sentiment values
    real_q = q_data[q_data['tipo'] == 'real']['sentimiento'].reset_index(drop=True)
    gen_q = q_data[q_data['tipo'] == 'generada']['sentimiento'].reset_index(drop=True)

    # Make sure we compare only pairs of responses
    min_len = min(len(real_q), len(gen_q))
    if min_len == 0:
        continue  # Skip questions that don't have both real and generated responses

    real_q = real_q[:min_len]
    gen_q = gen_q[:min_len]

    # Perform paired t-test
    t_stat, p_val = ttest_rel(real_q, gen_q)

    # Save the results
    results.append({
        "Question (Number)": int(q_num),
        "N pairs": min_len,
        "Mean Real": real_q.mean(),
        "Mean Generated": gen_q.mean(),
        "Mean Difference": gen_q.mean() - real_q.mean(),
        "t-statistic": t_stat,
        "p-value": p_val,
        "Significant (p < 0.05)": p_val < 0.05
    })

# Convert the list of results to a DataFrame
results_df = pd.DataFrame(results)

# Display the result
results_df
```

	Question (Number)	N pairs	Mean Real	Mean Generated	Mean Difference	t-statistic	p-value	Significant (p < 0.05)
0	1	4	-0.164309	-0.091830	0.072479	-0.600297	0.590627	False
1	2	4	-0.075175	-0.152666	-0.077491	5.980657	0.009357	True
2	3	4	-0.130239	-0.105476	0.024763	-0.117748	0.913708	False
3	4	4	-0.058371	-0.129777	-0.071406	1.739600	0.180309	False

```
from scipy.stats import wilcoxon

# Initialize a list to collect Wilcoxon results
wilcoxon_results = []

# Loop over each unique question number
for q_num in sorted(df['Number'].dropna().unique()):
    # Filter data for the current question
    q_data = df[df['Number'] == q_num]

    # Separate real and generated sentiment values
    real_q = q_data[q_data['tipo'] == 'real']['sentimiento'].reset_index(drop=True)
    gen_q = q_data[q_data['tipo'] == 'generada']['sentimiento'].reset_index(drop=True)

    # Match the length for paired comparison
    min_len = min(len(real_q), len(gen_q))
    if min_len == 0:
        continue  # Skip questions with missing pairs

    real_q = real_q[:min_len]
```

```
    gen_q = gen_q[:min_len]

    # Perform Wilcoxon signed-rank test
    try:
        stat, p_val = wilcoxon(real_q, gen_q)
    except ValueError:
        # If all differences are zero or the sample is too small, skip
        continue

    # Store the result
    wilcoxon_results.append({
        "Question (Number)": int(q_num),
        "N pairs": min_len,
        "Mean Real": real_q.mean(),
        "Mean Generated": gen_q.mean(),
        "Mean Difference": gen_q.mean() - real_q.mean(),
        "Wilcoxon statistic": stat,
        "p-value": p_val,
        "Significant (p < 0.05)": p_val < 0.05
    })

# Convert to DataFrame
wilcoxon_df = pd.DataFrame(wilcoxon_results)

# Display the result
wilcoxon_df
```

	Question (Number)	N pairs	Mean Real	Mean Generated	Mean Difference	Wilcoxon statistic	p-value	Significant (p < 0.05)
0	1	4	-0.164309	-0.091830	0.072479	5.0	1.000	False
1	2	4	-0.075175	-0.152666	-0.077491	0.0	0.125	False
2	3	4	-0.130239	-0.105476	0.024763	4.0	0.875	False
3	4	4	-0.058371	-0.129777	-0.071406	1.0	0.250	False

```
from scipy.stats import ks_2samp

# Initialize a list to collect KS test results
ks_results = []

# Loop through each unique question
for q_num in sorted(df['Number'].dropna().unique()):
    # Filter data for the current question
    q_data = df[df['Number'] == q_num]

    # Separate sentiment values
    real_q = q_data[q_data['tipo'] == 'real']['sentimiento'].reset_index(drop=True)
    gen_q = q_data[q_data['tipo'] == 'generada']['sentimiento'].reset_index(drop=True)

    # Perform K-S test only if both have values
    if len(real_q) > 0 and len(gen_q) > 0:
        ks_stat, p_val = ks_2samp(real_q, gen_q)
        ks_results.append({
            "Question (Number)": int(q_num),
            "N Real": len(real_q),
            "N Generated": len(gen_q),
            "Mean Real": real_q.mean(),
            "Mean Generated": gen_q.mean(),
            "K-S statistic": ks_stat,
            "p-value": p_val,
            "Significant (p < 0.05)": p_val < 0.05
        })

# Convert results to DataFrame
ks_df = pd.DataFrame(ks_results)

# Display the table
ks_df
```

	Question (Number)	N Real	N Generated	Mean Real	Mean Generated	K-S statistic	p-value	Significant (p < 0.05)
0	1	4	20	-0.164309	-0.112690	0.55	0.207039	False
1	2	4	20	-0.075175	-0.158285	0.80	0.013175	True
2	3	4	20	-0.130239	-0.072299	0.35	0.749482	False
3	4	4	20	-0.058371	-0.135207	0.65	0.086580	False

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set style for consistency
sns.set(style="whitegrid")

# Assuming df for MTY data is already loaded and processed (from cell Zi6tZd6d1eol)

# Get unique question numbers
question_numbers = sorted(df['Number'].dropna().unique())

# Create a figure and a set of subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten() # Flatten the 2x2 array of axes for easy iteration

# Loop through each question and plot on a different subplot
for i, q_num in enumerate(question_numbers):
    q_data = df[df['Number'] == q_num].copy()

    if q_data['tipo'].nunique() < 2:
        continue

    # Rename labels for clarity
    q_data['tipo'] = q_data['tipo'].replace({'real': 'Dataset', 'generada': 'Model'})

    # Boxplot
    ax = sns.boxplot(
        data=q_data,
        x='tipo',
        y='sentimiento',
        order=['Model', 'Dataset'],
        palette='pastel',
        linewidth=2.5,
        ax=axes[i] # Specify the subplot to draw on
    )

    # Stripplot
    sns.stripplot(
        data=q_data,
        x='tipo',
        y='sentimiento',
        order=['Model', 'Dataset'],
        color='black',
        alpha=0.4,
        ax=axes[i] # Specify the subplot to draw on
    )

    # Calculate and add average sentiment text to the side
    avg_sentiment_model = q_data[q_data['tipo'] == 'Model']['sentimiento'].mean()
    avg_sentiment_dataset = q_data[q_data['tipo'] == 'Dataset']['sentimiento'].mean()

    # Adjust y-coordinate based on the y-axis limits
    y_lim = ax.get_ylim()
    y_offset = (y_lim[1] - y_lim[0]) * 0.05 # 5% of the y-axis range as offset

    ax.text(0, y_lim[1] - y_offset, f'Model Avg: {avg_sentiment_model:.2f}',
            horizontalalignment='center', size='small', color='black', weight='semibold'
    ax.text(1, y_lim[1] - y_offset, f'Dataset Avg: {avg_sentiment_dataset:.2f}',
            horizontalalignment='center', size='small', color='black', weight='semibold'

    axes[i].set_title(f'MTY: Sentiment Distribution - Question {int(q_num)}')
    axes[i].set_xlabel('Response Type')
    axes[i].set_ylabel('Sentiment Score')
    axes[i].grid(True)

# Adjust layout to prevent overlapping titles and labels
```

```
plt.tight_layout()

# Show the combined plot
plt.show()
```

```
/tmp/ipython-input-16-2465138464.py:27: FutureWarning:

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

  ax = sns.boxplot(
/tmp/ipython-input-16-2465138464.py:27: FutureWarning:

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

  ax = sns.boxplot(
/tmp/ipython-input-16-2465138464.py:27: FutureWarning:

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

  ax = sns.boxplot(
/tmp/ipython-input-16-2465138464.py:27: FutureWarning:

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

  ax = sns.boxplot(
```