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

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

df = pd.read_csv("/content/MTY_Answers_real.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, 78.7kB/s]

vocab.json:     899k/? [00:00<00:00, 25.0MB/s]

merges.txt:     456k/? [00:00<00:00, 26.0MB/s]

special_tokens_map.json: 100%                                150/150 [00:00<00:00, 16.2kB/s]

pytorch_model.bin: 100%                                499M/499M [00:01<00:00, 373MB/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[['score_neg', 'score_neu', 'score_pos', 'sentimiento_real']] = df['Answer'].apply(get_sentiment_score)
df.to_csv("/content/MTY_Answers_real_sentimient.csv", index=False)
```

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

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

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

```
import pandas as pd

# Load real and generated responses (with sentiment scores)
df_real = pd.read_csv("/content/MTY_Answers_real_sentimient.csv")
df_gen = pd.read_csv("/content/MTY_Answers_generated_sentimient.csv")

# Group by 'pregunta' and calculate average sentiment
avg_sent_real = df_real.groupby("Number")["sentimiento_real"].mean().reset_index()
```

```
avg_sent_real.rename(columns={"sentimiento_real": "avg_sentimiento_real"}, inplace=True)

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

# Merge both results on the question
merged_avg = pd.merge(avg_sent_real, avg_sent_gen, on="Number")

# Display or export results
print(merged_avg)
merged_avg.to_csv("/content/avg_sentimiento_comparado.csv", index=False)
```

```
   Number  avg_sentimiento_real  avg_sentimiento_generado
0     1.0             -0.047824                 -0.112690
1     2.0             -0.109670                 -0.158286
```

```
import pandas as pd

# Load both real and generated datasets
df_real = pd.read_csv("/content/MTY_Answers_real_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/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       20  -0.047824       -0.112690        -0.064866     2.905217  0.009074                    True
1                  2       20  -0.109670       -0.158286        -0.048615     2.606231  0.017353                    True
```

```
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
```

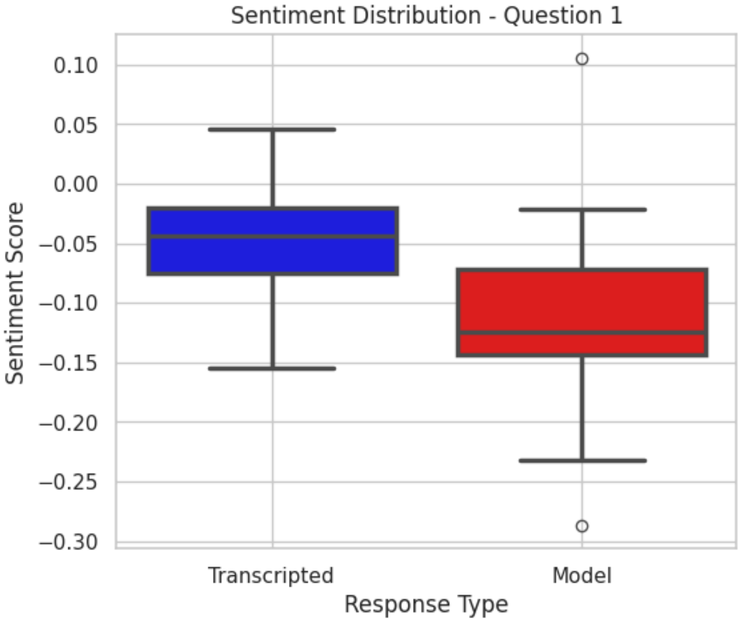
```
   Question (Number)  N pairs  Mean Real  Mean Generated  Mean Difference  Wilcoxon statistic   p-value  Significant (p < 0.05)
0                  1       20  -0.047824       -0.112690        -0.064866                28.0  0.002712                    True
1                  2       20  -0.109670       -0.158286        -0.048615                42.0  0.017181                    True
```

```
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
```

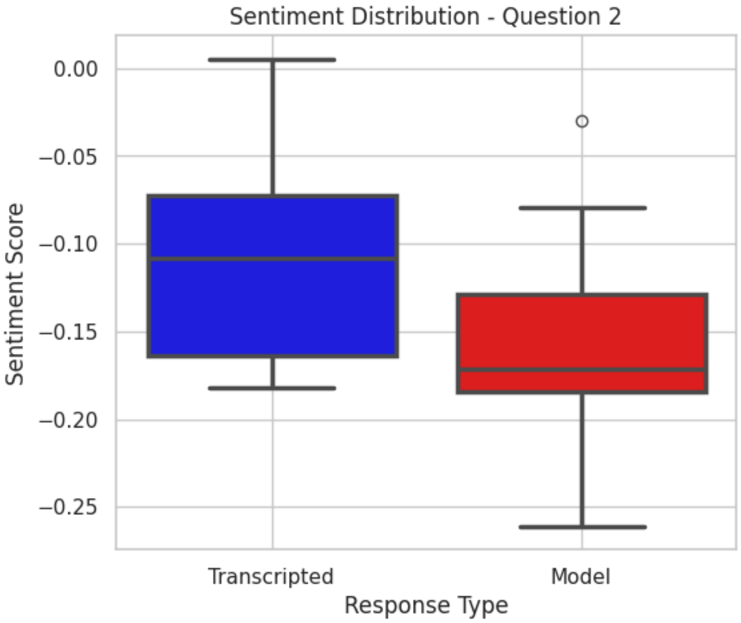
```
   Question (Number)  N Real  N Generated  Mean Real  Mean Generated  K-S statistic   p-value  Significant (p < 0.05)
0                  1      20           20  -0.047824       -0.112690            0.5  0.012299                    True
1                  2      20           20  -0.109670       -0.158286            0.4  0.081058                   False
```

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

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

# Loop through each question
question_numbers = sorted(df['Number'].dropna().unique())

for q_num in 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': 'Transcripted', 'generada': 'Model'})

    plt.figure(figsize=(6, 5))
    sns.boxplot(
        data=q_data,
        x='tipo',
        y='sentimiento',
        palette={'Transcripted': 'blue', 'Model': 'red'},
        linewidth=2.5
    )
    plt.title(f'Sentiment Distribution - Question {int(q_num)}')
    plt.xlabel('Response Type')
    plt.ylabel('Sentiment Score')
    plt.grid(True)
    plt.show()
```

```
/tmp/ipython-input-21-3022093883.py:20: 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

  sns.boxplot(
```



```
/tmp/ipython-input-21-3022093883.py:20: 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

  sns.boxplot(
```



COMPARATION VS FULL CORPUS

```
import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from torch.nn.functional import softmax
from tqdm import tqdm
```

```
# Load corpus file
df_corpus = pd.read_csv("/content/base_MTY_C+B_35-55.csv", encoding='ISO-8859-1')

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

# Function to get sentiment score from text using same logic as your image
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', 'sentiment_score'])

        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], dim=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', 'sentiment_score'
    except:
        return pd.Series([None, None, None, None], index=['score_neg', 'score_neu', 'score_pos', 'sentiment_score'])
```

```
# Apply to the "Participación" column with progress bar
tqdm.pandas()
scores_df = df_corpus['Participación'].astype(str).progress_apply(get_sentiment_score)

# Combine with original DataFrame
df_corpus_sentiment = pd.concat([df_corpus.reset_index(drop=True), scores_df], axis=1)

# Drop rows with errors
df_corpus_sentiment = df_corpus_sentiment.dropna(subset=['sentiment_score'])


# Quick check
df_corpus_sentiment[['Participación', 'sentiment_score']].head()
```

```
100%|██████████| 3321/3321 [05:10<00:00, 10.70it/s]
```

	Participación	sentiment_score
0	Ahí ya.	-0.037532
1	Esta es la primera vez que uso esta.	-0.020686
2	así	-0.066450
3	aparece mi regreso para mí.	-0.001293
4	Se minimizar mi video y ver tu video pantalla ...	-0.024201

```
# Calculate and print the average sentiment score of the corpus
avg_sentiment = df_corpus_sentiment["sentiment_score"].mean()
print(f"Average Sentiment Score (Corpus - Recalculated): {avg_sentiment:.3f}")
```

```
Average Sentiment Score (Corpus - Recalculated): -0.037
```

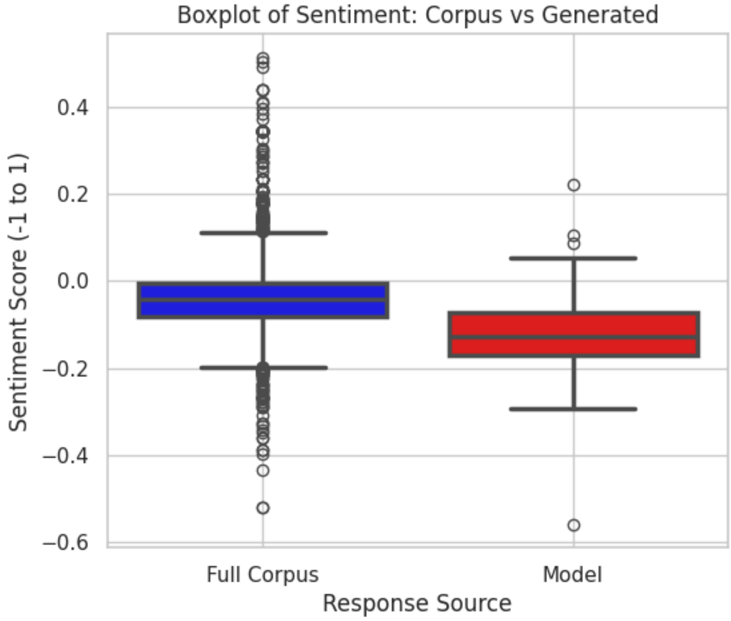
```
from scipy.stats import ttest_ind, ks_2samp

# Load generated sentiment scores
df_generated = pd.read_csv("/content/MTY_Sentiment_Comparation.csv", encoding='ISO-8859-1')
sent_gen = df_generated[df_generated["tipo"] == "generada"]["sentimiento"].dropna()

# Load recalculated corpus sentiment scores
# (assuming you already have df_corpus_sentiment from the previous step)
sent_corpus = df_corpus_sentiment["sentiment_score"].dropna()

# Perform t-test
t_stat, p_val_t = ttest_ind(sent_gen, sent_corpus)

# Perform K-S test
ks_stat, p_val_ks = ks_2samp(sent_gen, sent_corpus)

# Print results
```

```
print(f"Average Sentiment - Corpus (Recalculated): {sent_corpus.mean():.3f}")
print(f"Average Sentiment - Generated: {sent_gen.mean():.3f}")
print(f"\nT-test p-value: {p_val_t:.5f}")
print(f"K-S test p-value: {p_val_ks:.5f}")
```

```
Average Sentiment - Corpus (Recalculated): -0.037
Average Sentiment - Generated: -0.120

T-test p-value: 0.00000
K-S test p-value: 0.00000
```

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

# Combine the two series into a single DataFrame for plotting
df_plot = pd.DataFrame({
    "Sentiment": pd.concat([sent_corpus, sent_gen], ignore_index=True),
    "Source": ["Full Corpus"] * len(sent_corpus) + ["Model"] * len(sent_gen)
})

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

# Plot boxplot
plt.figure(figsize=(6, 5))
sns.boxplot(data=df_plot, x="Source", y="Sentiment", palette={"Full Corpus": "blue", "Model": "red"}, linewidth=2.5)

plt.title("Boxplot of Sentiment: Corpus vs Generated")
plt.xlabel("Response Source")
plt.ylabel("Sentiment Score (-1 to 1)")
plt.grid(True)
plt.show()
```

```
/tmp/ipython-input-27-2181211753.py:16: 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_plot, x="Source", y="Sentiment", palette={"Full Corpus": "blue", "Model": "red"}, linewidth=2.5)
```


Boxplot of Sentiment: Corpus vs Generated
Sentiment Score (-1 to 1)
Full Corpus
Model
Response Source