```
Empieza a programar o a crear código con IA.
```

```
!pip install nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
import nltk
import pandas as pd
import re
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
```

```
def process_transcription(file_name):
    """
    Process a transcription file to extract structured data.
    """
    try:
        with open(file_name, 'r', encoding='utf-8') as f:
            lines = f.readlines()
    except FileNotFoundError:
        raise FileNotFoundError("The specified file was not found.")

    # Extract metadata from the file name
    match = re.match(r"Grupo (.+) Plaza (.+) Edades (.+) NSE (.+)\.txt", file_name.split('/')[-1])
    if not match:
        raise ValueError("File name format is incorrect. Expected format: 'Grupo [Name] Plaza [Name] Edades [Range] NSE [Value].txt'")

    Grupo, Plaza, Edades, NSE = match.groups()

    data = []
    # Adjusted pattern to handle both mm:ss and hh:mm:ss
    pattern = r"^(\d{1,2}:\d{2}(?::\d{2})?)\s+([^:]+):\s+(.+)"

    for line in lines:
        match = re.match(pattern, line)
        if match:
            Tiempo, Nombre, participation = match.groups()

            # Normalize mm:ss to hh:mm:ss
            if len(Tiempo.split(':')) == 2:
                Tiempo = f"00:{Tiempo}"

            data.append({
                "Grupo": Grupo,
                'Tiempo': Tiempo,
                'Nombre': Nombre,
                'Participación': participation.strip(),
                'Plaza': Plaza,
                'NSE': NSE,
                'Edades': Edades
            })

    if not data:
        raise ValueError("No valid data extracted from the transcription file.")

    df = pd.DataFrame(data)
    return df

# List of file names
file_names = [
    "/content/Grupo SG01 Plaza XAL Edades 20 - 30 años NSE C-D+.txt",
    "/content/Grupo SG022 Plaza XAL Edades 35 - 55 años NSE C-D+.txt",
    "/content/Grupo SG03 Plaza XAL Edades 30 - 50 años NSE C+B.txt"
]

all_dataframes = []

try:
    for file_name in file_names:
        # Process each transcription file
        df = process_transcription(file_name)
        all_dataframes.append(df)

    # Combine all dataframes into one
    combined_df = pd.concat(all_dataframes, ignore_index=True)

    # Display the first rows of the combined DataFrame
    print(combined_df.head())

    # Save the combined DataFrame to a CSV file
    combined_df.to_csv("combined_transcriptions.csv", index=False, encoding='utf-8')
    print("All files processed and saved successfully.")

except Exception as e:
    print(f"An error occurred: {e}")

# List of known moderators
moderators = ["Yvone Carrillo",  "Yvon Carrillo", "Dan Cortés", "Carlos Villanueva Avilez", "Karime Galicia", "Mario Juárez", "Natalia Rodríg
```

```
  Grupo    Tiempo                           Nombre  \
0  SG01  00:00:15                   Karime Galicia
1  SG01  00:00:35  Atala Paulina Alvarez Fernandez
2  SG01  00:00:36                   Karime Galicia
3  SG01  00:00:40                    Alery Aguilar
4  SG01  00:00:41                   Karime Galicia

                                     Participación Plaza   NSE        Edades
0  Hola chicos, Buenas tardes, María Diana activa...   XAL  C-D+  20 - 30 años
1                                    Buenas tardes.   XAL  C-D+  20 - 30 años
2                      Buenas tardes Hola Alessandro   XAL  C-D+  20 - 30 años
3                                    Buenas tardes.   XAL  C-D+  20 - 30 años
4  Buenas tardes, chicos, tengo un dispositivo a ...   XAL  C-D+  20 - 30 años
All files processed and saved successfully.
```

```
def anonymize_and_flag(df):
    """
    Add a column to flag if the person is a Moderador or Participante,
    """
    # Create a column to flag if the row corresponds to a Moderator
    df['Rol'] = df['Nombre'].apply(lambda x: 'Moderador' if x in moderators else 'Participante')

    return df

# Load the previously saved DataFrame
combined_df = pd.read_csv("/content/combined_transcriptions.csv")

# Process the DataFrame to flag roles and anonymize participant names
processed_df = anonymize_and_flag(combined_df)

# Save the processed DataFrame to a new CSV file
processed_df.to_csv("Base de datos.csv", index=False, encoding='utf-8')

print("Roles de Moderadores y participantes agregados")
```

```
Roles de Moderadores y participantes agregados
```

```
# Download required NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab') # This line was added

def clean_text(participación):
    """
    Clean and preprocess the participation text.
    """
    # Remove non-alphanumeric characters, except spaces
    text = re.sub(r'[^\w\s]', '', participación)
    # Convert to lowercase
    text = text.lower()
    # Tokenize text
    tokens = word_tokenize(text, language='spanish')
    # Join tokens back into a string without removing stopwords
    return ' '.join(tokens)


def preprocess_data(df):
    """
    Apply text cleaning to the participation column without removing stopwords.
    """
    df['Participación limpia'] = df['Participación'].apply(clean_text)
    return df

# Load the previously processed DataFrame
processed_df = pd.read_csv("/content/Base de datos.csv")

# Clean the participation column
cleaned_df = preprocess_data(processed_df)

# Save the cleaned DataFrame to a new CSV file
cleaned_df.to_csv("Base de datos 2.csv", index=False, encoding='utf-8')

print("Participation text cleaned and saved successfully.")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
Participation text cleaned and saved successfully.
```

Coatza

```
# Final adjusted thematic categories in English (excluding proper nouns)
thematic_categories = {
    "Orgullo": ["orgullo", "orgulloso", "identidad"],
    "Problematicas": ["corrupción", "impunidad", "transa", "mordida", "problema", "violencia", "inseguridad", "narco", "robo", "asalto", "tr
    "Elecciones": ["elección", "candidato", "candidata", "fraude", "optimismo", "desilusión", "votación", "legitimidad", "sentimiento", "emc
    "Morena": ["obrador", "amlo", "morena", "moreno", "guinda", "andres manuel lopez"],
    "Movimiento Ciudadano": ["movimiento ciudadano", "naranja", "canción"],
    "PRIAN": ["pri", "anaya", "pan", "prian", "xochitl"],
}

# Propagation categories (Only for Moderators, but propagates to Participants and Moderators)
propagation_categories = {
    "contexto social": ["seguimos con el tema del contexto social", "seguimos con contexto social", "cómo ven el estado de veracruz"],
    "contexto local": ["seguimos con el contexto local", "sigamos con la sección del contexto local"],
    "pemex": ["qué significa pemex", "cómo funciona pemex", "cómo ves a pemex"],
    "panorama electoral": ["seguimos con el tema del panorama electoral", "seguimos con el panorama electoral", "seguimos con panorama elect
    "percepción de partidos": ["seguimos con percepción de partidos", "seguimos con la percepción de partidos", "sigamos con la parte de la
    "Maynez": ["maynes", "maynez", "maines", "mainez", "por ahí el nombre de movimiento"],
    "yunes": ["seguimos con la sección del bloque yunes", "seguimos con el bloque yunes"],
    "evaluacion de candidatos": ["seguimos con la evaluación de candidatos", "seguimos con candidatos", "que no hay que juzgar a la gente"],
    "evaluacion de boleta": ["seguimos con la evaluación de boleta"],
}

# Function to classify participation into multiple themes
def classify_multiple_thematic_categories(text, categories):
    assigned_themes = set()
    if isinstance(text, str):
        text = text.lower()  # Normalize text to lowercase
        for theme, keywords in categories.items():
            for keyword in keywords:
                if re.search(rf'\b{keyword}\b', text):
                    assigned_themes.add(theme)
    return ", ".join(assigned_themes) if assigned_themes else "Sin Clasificar"

# Function to classify propagation categories (Only for Moderators)
def classify_propagation_categories(text, categories):
    assigned_themes = set()
    if isinstance(text, str):
        text = text.lower()  # Normalize text to lowercase
        for theme, keywords in categories.items():
            for keyword in keywords:
                if re.search(rf'\b{keyword}\b', text):
                    assigned_themes.add(theme)
    return ", ".join(assigned_themes) if assigned_themes else "Sin Clasificar"

# Apply thematic classification to both participants and moderators (No propagation)
def classify_thematic_categories(group):
    group['Categorias extra'] = group['Participación limpia'].apply(
        lambda text: classify_multiple_thematic_categories(text, thematic_categories)
    )
    return group

# Apply propagation classification only to moderators
def classify_propagation(group):
    group['Categorias bloque'] = group.apply(
        lambda row: classify_propagation_categories(row['Participación limpia'], propagation_categories)
        if row['Rol'] == 'Moderador' else "Sin Clasificar", axis=1
    )
    return group

# Propagate moderator themes to all responses within the same group and plaza
def propagate_moderator_themes(group):
    current_theme = None
    for index, row in group.iterrows():
        if row['Categorias bloque'] != "Sin Clasificar":  # Moderator introduces a new theme
            current_theme = row['Categorias bloque']
        elif current_theme:  # Propagate to both Moderators and Participants
            if row['Categorias bloque'] == "Sin Clasificar":
                group.at[index, 'Categorias bloque'] = current_theme
            else:
                existing_themes = set(row['Categorias bloque'].split(", "))
                new_themes = existing_themes.union(set(current_theme.split(", ")))
                group.at[index, 'Categorias bloque'] = ", ".join(new_themes)
    return group

# Load the cleaned transcriptions file
data_path = "/content/Base de datos 2.csv"
cleaned_df = pd.read_csv(data_path, encoding='utf-8')

# Sort data chronologically within groups
cleaned_df = cleaned_df.sort_values(by=['Grupo', 'Plaza', 'Tiempo'])

# Apply thematic classification (for both Moderators and Participants)
cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_thematic_categories)

# Apply propagation classification (only for Moderators)
cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_propagation)

# Propagate propagation themes from Moderators to both Moderators and Participants
cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(propagate_moderator_themes)

# Save the results to a new CSV file
output_path = "Base de datos final.csv"
cleaned_df.to_csv(output_path, index=False, encoding='utf-8-sig')

print(f"Multi-thematic classification with updated rules completed, saved to '{output_path}'.")
```

```
Multi-thematic classification with updated rules completed, saved to 'Base de datos final.csv'.
<ipython-input-17-b7435c29f7ba>:84: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprec
  cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_thematic_categories)
<ipython-input-17-b7435c29f7ba>:87: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprec
  cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_propagation)
<ipython-input-17-b7435c29f7ba>:90: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprec
  cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(propagate_moderator_themes)
```

Veracruz

```
# Final adjusted thematic categories in English (excluding proper nouns)
thematic_categories = {
    "Orgullo": ["orgullo", "orgulloso", "identidad"],
    "Problematicas": ["corrupción", "impunidad", "transa", "mordida", "problema", "violencia", "inseguridad", "narco", "robo", "asalto", "tr
    "Elecciones": ["elección", "candidato", "candidata", "fraude", "optimismo", "desilusión", "votación", "legitimidad", "sentimiento", "emo
    "Morena": ["obrador", "amlo", "morena", "moreno", "guinda", "andres manuel lopez"],
    "Movimiento Ciudadano": ["movimiento ciudadano", "naranja", "canción"],
    "PRIAN": ["pri", "anaya", "pan", "prian", "xochitl"],
    "Maynez": ["maynes", "maynez", "maines", "mainez"],
    "Belem Palmeros": ["belem", "belén", "palmeros"],
    "Raul Zarrabal": ["raul", "raúl", "sarrabal", "zarrabal"],
}

# Propagation categories (Only for Moderators, but propagates to Participants and Moderators)
```

```
propagation_categories = {
    "Contexto estatal": ["contarme acerca de veracruz", "habláramos del estado de veracruz","hablemos de su estado", "cómo ve al estado en g
    "Contexto local": ["los tres principales problemas de xalapa", "hablemos de la gente de xalapa", "hablemos de su municipio", "en cuanto
    "Panorama electoral": ["hablemos de el panorama electoral", "hablemos del panorama electoral", "hablemos de panorama electoral","seguimo
    "PRI": ["tres palabras para describir al pri ", "hablemos del pri", "empezando con el pri", "hablando súper rápido del pri"],
    "PAN": ["del pan qué me dicen", "hablemos del pan",  "tres palabras para describir al pan", "me cuenta el pan"],
    "Morena": [" la cuestión de xalapa talina", "hablemos de morena", "tres palabras para describir a morena", "hablamos de morena"],
    "Movimiento Ciudadano": ["tres palabras movimiento ciudadano", "hablar de movimiento ciudadano", "hablemos de movimiento ciudadano","mov
    "Maynez": ["okay okay alguien sabe quién es él", "fue candidato presidencial", "jorge álvarez máynez", "alvarez","desde diciembre es", "
    "yunes": ["hablemos de los yunes", "respecto los yunes", "acerca de los yunes"],
    "Roman Moreno": ["joven empresario dueño", "precandidato para movimiento ciudadano", "ha apoyado campañas y gobiernos"],
    "Maribel Ramírez": ["diputada local del pan", "comisión para la igualdad de género", "candidata por el pan para la alcaldía"],
    "Silvio Lagos": ["silvio lagos fue secretario general", "director estatal de asuntos", "candidato por parte del pri"],
    "Daniela Griego": ["ex directora general del instituto", "promotora de la Cuarta ", "daniela griego"],
    "Evaluacion de boleta": ["va a ganar las elecciones municipales", "quién creen que va a ganar"],
}

# Function to classify participation into multiple themes
def classify_multiple_thematic_categories(text, categories):
    assigned_themes = set()
    if isinstance(text, str):
        text = text.lower()  # Normalize text to lowercase
        for theme, keywords in categories.items():
            for keyword in keywords:
                if re.search(rf'\b{keyword}\b', text):
                    assigned_themes.add(theme)
    return ", ".join(assigned_themes) if assigned_themes else "Sin Clasificar"

# Function to classify propagation categories (Only for Moderators)
def classify_propagation_categories(text, categories):
    assigned_themes = set()
    if isinstance(text, str):
        text = text.lower()  # Normalize text to lowercase
        for theme, keywords in categories.items():
            for keyword in keywords:
                if re.search(rf'\b{keyword}\b', text):
                    assigned_themes.add(theme)
    return ", ".join(assigned_themes) if assigned_themes else "Sin Clasificar"

# Apply thematic classification to both participants and moderators (No propagation)
def classify_thematic_categories(group):
    group['Categorias extra'] = group['Participación limpia'].apply(
        lambda text: classify_multiple_thematic_categories(text, thematic_categories)
    )
    return group

# Apply propagation classification only to moderators
def classify_propagation(group):
    group['Categorias bloque'] = group.apply(
        lambda row: classify_propagation_categories(row['Participación limpia'], propagation_categories)
        if row['Rol'] == 'Moderador' else "Sin Clasificar", axis=1
    )
    return group

# Propagate moderator themes to all responses within the same group and plaza
def propagate_moderator_themes(group):
    current_theme = None
    for index, row in group.iterrows():
        if row['Categorias bloque'] != "Sin Clasificar":  # Moderator introduces a new theme
            current_theme = row['Categorias bloque']
        elif current_theme:  # Propagate to both Moderators and Participants
            if row['Categorias bloque'] == "Sin Clasificar":
                group.at[index, 'Categorias bloque'] = current_theme
            else:
                existing_themes = set(row['Categorias bloque'].split(", "))
                new_themes = existing_themes.union(set(current_theme.split(", ")))
                group.at[index, 'Categorias bloque'] = ", ".join(new_themes)
    return group

# Load the cleaned transcriptions file
data_path = "/content/Base de datos 2.csv"
cleaned_df = pd.read_csv(data_path, encoding='utf-8')

# Sort data chronologically within groups
cleaned_df = cleaned_df.sort_values(by=['Grupo', 'Plaza', 'Tiempo'])

# Apply thematic classification (for both Moderators and Participants)
```

```
cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_thematic_categories)

# Apply propagation classification (only for Moderators)
cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_propagation)

# Propagate propagation themes from Moderators to both Moderators and Participants
cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(propagate_moderator_themes)

# Save the results to a new CSV file
output_path = "Base de datos final.csv"
cleaned_df.to_csv(output_path, index=False, encoding='utf-8-sig')

print(f"Multi-thematic classification with updated rules completed, saved to '{output_path}'.")
```

```
<ipython-input-19-61cd10b1e46a>:92: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprec
  cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_thematic_categories)
<ipython-input-19-61cd10b1e46a>:95: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprec
  cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(classify_propagation)
Multi-thematic classification with updated rules completed, saved to 'Base de datos final.csv'.
<ipython-input-19-61cd10b1e46a>:98: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprec
  cleaned_df = cleaned_df.groupby(['Grupo', 'Plaza'], group_keys=False).apply(propagate_moderator_themes)
```

```
Veracruz
```