```
**This Notebook provides a entry point towards quantitative analysis of Global
Dialogues data. Along with Global index calculation usecase. And Data
Visualisation.**

*AUTHOR: Ganesh Gopalkrishna Hegde*
```

This Notebook provides a entry point towards quantitative analysis of Global Dialogues data. Along with Global index calculation usecase. And Data Visualisation.

AUTHOR: Ganesh Gopalkrishna Hegde

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import json
import pandas as pd
import warnings
warnings.filterwarnings('ignore')


#Use this to load the data with embeddings from the global_inputs.json file
with open('/content/drive/MyDrive/global dialogue data/global_inputs.json', 'r') as f:
    loaded_list = json.load(f)

# Convert the list of dictionaries back to DataFrames
qs = [pd.DataFrame(df) for df in loaded_list]
```

Discourse Sophistication Score (DSS)

This index measures the breadth and complexity of the AI conversation within a country. A nation where citizens discuss a wide array of topics (e.g., jobs, ethics, healthcare, existential risk) is considered to have a more sophisticated discourse than one where the conversation is dominated by a single theme. This can be calculated using a normalized entropy measure over the distribution of thematic codes. A higher score indicates a more diverse and multi-faceted public conversation.

Let p_i be the proportion of the i-th thematic code out of all codes applied in a country. The entropy H is calculated as:

$$H = -\sum_{i=1}^{N} p_i \log_2(p_i)$$

Where N is the number of unique thematic codes identified in the country's responses.

The DSS is then the normalized entropy:

$$DSS = \frac{H}{H_{max}} = \frac{H}{\log_2(N)}$$

If $N = 1$, H_{max} (and H) is 0; DSS can be defined as 0. If $N = 0$, DSS is undefined (or 0).

```
import numpy as np
from collections import Counter
from scipy.stats import entropy

def calculate_dss(country_coded_responses):
    """
    Calculates the Discourse Sophistication Score (DSS) using normalized entropy.
    Assumes country_coded_responses is a list of lists, where each inner list
    contains the codes applied to a single response for that country.

    Args:
        country_coded_responses (list of list of str): List of coded responses for a country.

    Returns:
        float: The DSS score (between 0 and 1), or None if no codes are applied or only one unique code type.
    """
    all_codes_flat = [code for sublist in country_coded_responses for code in sublist]
    if not all_codes_flat:
        return 0 # No codes, so sophistication is zero

    code_counts = Counter(all_codes_flat)
    num_unique_codes = len(code_counts)

    if num_unique_codes <= 1:
        return 0 # If 0 or 1 unique code, entropy is 0, so DSS is 0
```

```
    # Calculate probabilities of each code
    probabilities = np.array(list(code_counts.values())) / len(all_codes_flat)

    # Calculate entropy
    # Scipy's entropy uses natural log by default, specify base=2 for bits
    actual_entropy = entropy(probabilities, base=2)

    # Calculate maximum possible entropy
    max_entropy = np.log2(num_unique_codes)

    if max_entropy == 0: # Should be covered by num_unique_codes <= 1, but as safeguard
        return 0

    dss = actual_entropy / max_entropy
    return dss

# Example Usage:
country_A_codes_dss = [
    ["ECON_OPPORTUNITY", "HEALTHCARE_BENEFIT"],
    ["ECON_THREAT", "GOVERNANCE_REGULATION"],
    ["ECON_OPPORTUNITY"],
    ["ETHICS_BIAS"]
] # Diverse codes

country_B_codes_dss = [
    ["ECON_THREAT"],
    ["ECON_THREAT"],
    ["ECON_THREAT"],
    ["ECON_THREAT"]
] # Dominated by one code

country_C_codes_dss = [["ECON_OPPORTUNITY"], ["ECON_THREAT"]] # Two codes, perfectly balanced

country_D_codes_dss = [] # No codes
country_E_codes_dss = [["ETHICS_BIAS"]] # Only one code type

dss_A = calculate_dss(country_A_codes_dss)
dss_B = calculate_dss(country_B_codes_dss)
dss_C = calculate_dss(country_C_codes_dss)
dss_D = calculate_dss(country_D_codes_dss)
dss_E = calculate_dss(country_E_codes_dss)

print(f"DSS for Country A (diverse): {dss_A}")
print(f"DSS for Country B (single theme dominance): {dss_B}")
print(f"DSS for Country C (balanced two themes): {dss_C}")
print(f"DSS for Country D (no codes): {dss_D}")
print(f"DSS for Country E (one code type): {dss_E}")
```

```
DSS for Country A (diverse): 0.9697238998682475
DSS for Country B (single theme dominance): 0
DSS for Country C (balanced two themes): 1.0
DSS for Country D (no codes): 0
DSS for Country E (one code type): 0
```

Governance & Ethics Concern Index (GECI)

This index quantifies the salience of regulatory and ethical concerns within a country's public discourse. A high GECI suggests that the conversation is heavily focused on the challenges of managing AI responsibly. It is calculated as the share of all identified themes that relate to governance or ethics:

$$GECI = \frac{\text{Frequency of GOVERNANCE\_REGULATION codes + Frequency of ETHICS\_BIAS codes}}{\text{Total number of all codes applied}}$$

```
from collections import Counter

def calculate_geci(country_coded_responses):
    """
    Calculates the Governance & Ethics Concern Index (GECI).
    Assumes country_coded_responses is a list of lists, where each inner list
    contains the codes applied to a single response for that country.
    GECI = (Freq GOVERNANCE_REGULATION + Freq ETHICS_BIAS) / Total number of all codes applied

    Args:
        country_coded_responses (list of list of str): List of coded responses for a country.

    Returns:
        float: The GECI score, or None if no codes are applied at all.
    """
    all_codes_flat = [code for sublist in country_coded_responses for code in sublist]
    if not all_codes_flat:
```

```
        return None # No codes applied at all

    code_counts = Counter(all_codes_flat)

    freq_gov_reg = code_counts.get("GOVERNANCE_REGULATION", 0)
    freq_ethics_bias = code_counts.get("ETHICS_BIAS", 0)

    total_codes_applied = len(all_codes_flat)

    if total_codes_applied == 0:
        return None # Should be caught by the earlier check, but as a safeguard

    geci = (freq_gov_reg + freq_ethics_bias) / total_codes_applied
    return geci

# Example Usage (using the same example data as EAI for consistency):
country_A_codes = [
    ["ECON_OPPORTUNITY", "HEALTHCARE_BENEFIT"],
    ["ECON_THREAT", "GOVERNANCE_REGULATION"],
    ["ECON_OPPORTUNITY"],
    ["ETHICS_BIAS"]
]
# Total codes: 2+2+1+1 = 6
# GOV_REG = 1, ETHICS_BIAS = 1. GECI = (1+1)/6 = 2/6 = 0.333

country_B_codes = [
    ["ECON_THREAT"],
    ["ECON_THREAT", "SURVEILLANCE_PRIVACY", "GOVERNANCE_REGULATION"],
    ["GOVERNANCE_REGULATION"],
    ["ECON_THREAT", "ETHICS_BIAS"]
]
# Total codes: 1+3+1+2 = 7
# GOV_REG = 2, ETHICS_BIAS = 1. GECI = (2+1)/7 = 3/7 = 0.428

country_C_codes = [
    ["HEALTHCARE_BENEFIT"],
    ["EXISTENTIAL_RISK"]
]
# Total codes: 1+1 = 2
# GOV_REG = 0, ETHICS_BIAS = 0. GECI = 0/2 = 0

country_E_codes = [] # No responses, no codes

geci_A = calculate_geci(country_A_codes)
geci_B = calculate_geci(country_B_codes)
geci_C = calculate_geci(country_C_codes)
geci_E = calculate_geci(country_E_codes)

print(f"GECI for Country A: {geci_A}")
print(f"GECI for Country B: {geci_B}")
print(f"GECI for Country C: {geci_C}")
print(f"GECI for Country E (no codes): {geci_E}")
```

```
GECI for Country A: 0.3333333333333333
GECI for Country B: 0.42857142857142855
GECI for Country C: 0.0
GECI for Country E (no codes): None
```

Economic Anxiety Index (EAI)

This index is designed to measure the specific balance of economic hope versus economic fear in the AI conversation. It isolates the economic dimension of the discourse to provide a more targeted measure of anxiety. It is calculated as the proportion of economic-themed comments that are negative:

$$EAI = \frac{\text{Frequency of ECON\_THREAT codes}}{\text{Frequency of ECON\_THREAT codes + Frequency of ECON\_OPPORTUNITY codes}}$$

```
from collections import Counter

def calculate_eai(country_coded_responses):
    """
    Calculates the Economic Anxiety Index (EAI).
    Assumes country_coded_responses is a list of lists, where each inner list
    contains the codes applied to a single response for that country.
    EAI = Frequency of ECON_THREAT / (Frequency of ECON_THREAT + Frequency of ECON_OPPORTUNITY)

    Args:
        country_coded_responses (list of list of str): List of coded responses for a country.

    Returns:
```

```
        float: The EAI score, or None if no relevant economic codes are found.
    """
    all_codes_flat = [code for sublist in country_coded_responses for code in sublist]
    code_counts = Counter(all_codes_flat)

    freq_econ_threat = code_counts.get("ECON_THREAT", 0)
    freq_econ_opportunity = code_counts.get("ECON_OPPORTUNITY", 0)

    denominator = freq_econ_threat + freq_econ_opportunity

    if denominator == 0:
        return None # Or 0, depending on how undefined cases should be handled

    eai = freq_econ_threat / denominator
    return eai

# Example Usage:
# Assume these are coded responses from different countries/datasets
country_A_codes = [
    ["ECON_OPPORTUNITY", "HEALTHCARE_BENEFIT"],
    ["ECON_THREAT", "GOVERNANCE_REGULATION"],
    ["ECON_OPPORTUNITY"],
    ["ETHICS_BIAS"]
]

country_B_codes = [
    ["ECON_THREAT"],
    ["ECON_THREAT", "SURVEILLANCE_PRIVACY"],
    ["GOVERNANCE_REGULATION"],
    ["ECON_THREAT"]
]

country_C_codes = [
    ["HEALTHCARE_BENEFIT"], # No economic codes
    ["ETHICS_BIAS"]
]

country_D_codes = [ # Only ECON_OPPORTUNITY
    ["ECON_OPPORTUNITY"],
    ["ECON_OPPORTUNITY"]
]

eai_A = calculate_eai(country_A_codes)
eai_B = calculate_eai(country_B_codes)
eai_C = calculate_eai(country_C_codes)
eai_D = calculate_eai(country_D_codes)

print(f"EAI for Country A: {eai_A}")
print(f"EAI for Country B: {eai_B}")
print(f"EAI for Country C: {eai_C}")
print(f"EAI for Country D: {eai_D}")
```

```
EAI for Country A: 0.3333333333333333
EAI for Country B: 1.0
EAI for Country C: None
EAI for Country D: 0.0
```

National AI Optimism Index (AIOI)

This index provides a single, powerful measure of the net sentiment towards AI within a given country. It balances the positive and negative sentiments to reveal the overall emotional leaning of the national discourse. It is calculated as:

$$AIOI = \frac{\text{\% of Positive Responses} - \text{\% of Negative Responses}}{\text{\% of Total Responses that are Positive or Negative}}$$

Alternatively, if we consider all responses (including neutral) in the denominator for percentages:

$$AIOI = (\text{\% of Positive Responses} - \text{\% of Negative Responses})$$

Where % is with respect to the total number of responses for the country. The user's original text implies the latter: `AIOI= % of Total Responses (% of Positive Responses−% of Negative Responses)` which seems to have a typo and likely means `(% of Positive Responses−% of Negative Responses) / (% of Positive or Negative Responses)` or simply the difference of percentages of total responses. Given the phrasing "balances the positive and negative sentiments to reveal the overall emotional leaning", the simple difference `(% Positive - % Negative)` seems most direct if percentages are of total responses. If neutral responses are excluded from the base for percentage calculation, then the first formula normalizes by the opinionated responses.

```
import numpy as np

def calculate_aioi(sentiment_scores, positive_threshold=0.05, negative_threshold=-0.05):
    """
```

```
    Calculates the National AI Optimism Index (AIOI).
    Assumes sentiment_scores is a list or array of numerical sentiment scores for a country.
    AIOI = (% of Positive Responses - % of Negative Responses)
    where percentages are with respect to the total number of responses.

    Args:
        sentiment_scores (list or np.array): List of sentiment scores for a country.
        positive_threshold (float): Threshold above which a score is considered positive.
        negative_threshold (float): Threshold below which a score is considered negative.

    Returns:
        float: The AIOI score, or None if no responses.
    """
    if not sentiment_scores:
        return None

    scores_array = np.array(sentiment_scores)
    total_responses = len(scores_array)

    num_positive = np.sum(scores_array > positive_threshold)
    num_negative = np.sum(scores_array < negative_threshold)

    percent_positive = (num_positive / total_responses) * 100
    percent_negative = (num_negative / total_responses) * 100

    aioi = percent_positive - percent_negative
    return aioi

# Example Usage:
country_sentiments_A = [0.8, 0.5, 0.1, -0.6, -0.3, 0.0, 0.9, 0.7, -0.1, 0.2] # High optimism
country_sentiments_B = [-0.8, -0.5, -0.1, 0.6, 0.3, 0.0, -0.9, -0.7, 0.1, -0.2] # High pessimism
country_sentiments_C = [0.1, -0.1, 0.05, -0.05, 0.0, 0.02, -0.02] # Neutral / Mixed
country_sentiments_D = [] # No responses

aioi_A = calculate_aioi(country_sentiments_A)
aioi_B = calculate_aioi(country_sentiments_B)
aioi_C = calculate_aioi(country_sentiments_C)
aioi_D = calculate_aioi(country_sentiments_D)

print(f"AIOI for Country A: {aioi_A}")
print(f"AIOI for Country B: {aioi_B}")
print(f"AIOI for Country C: {aioi_C}")
print(f"AIOI for Country D: {aioi_D}")

# Alternative AIOI (normalized by opinionated responses)
def calculate_aioi_normalized(sentiment_scores, positive_threshold=0.05, negative_threshold=-0.05):
    if not sentiment_scores:
        return None
    scores_array = np.array(sentiment_scores)
    num_positive = np.sum(scores_array > positive_threshold)
    num_negative = np.sum(scores_array < negative_threshold)
    num_opinionated = num_positive + num_negative
    if num_opinionated == 0:
        return 0 # Or None, depending on desired behavior for no opinionated responses
    percent_positive_of_opinionated = (num_positive / num_opinionated) * 100
    percent_negative_of_opinionated = (num_negative / num_opinionated) * 100
    aioi_norm = percent_positive_of_opinionated - percent_negative_of_opinionated
    return aioi_norm

aioi_A_norm = calculate_aioi_normalized(country_sentiments_A)
print(f"Normalized AIOI for Country A: {aioi_A_norm}")
```

```
AIOI for Country A: 30.0
AIOI for Country B: -30.0
AIOI for Country C: 0.0
AIOI for Country D: None
Normalized AIOI for Country A: 33.33333333333333
```

Developing Novel, Country-Level Indices

The true analytical power emerges when the coded and sentiment-scored data from individual responses are aggregated to the country level. This step creates a series of novel, composite indices that can be directly compared and correlated with external macroeconomic data. The creation of these indices is a significant act of analytical innovation, transforming the raw dataset into a new source of knowledge and providing a distinct competitive advantage in the challenge.

The following indices are proposed for this project:

Method 2: Sentiment Analysis

Sentiment analysis, or opinion mining, provides a complementary quantitative layer by assigning a polarity score to each response. This captures the overall emotional tone of the discourse.

Process: A sentiment analysis model is applied to classify each response, or even each sentence within a response, as positive, negative, or neutral. This can be effectively accomplished using well-established, pre-trained libraries in programming languages like Python. For example, the VADER (Valence Aware Dictionary and sEntiment Reasoner) library is particularly well-suited for social media-style text, while TextBlob offers a more general-purpose approach. For higher accuracy, more advanced transformer-based models (e.g., from the Hugging Face library) can be fine-tuned on a small, manually labeled subset of the survey data.

Output: The result of this process is a numerical sentiment score for each unit of text. A common scale ranges from -1 (indicating a highly negative sentiment) to +1 (indicating a highly positive sentiment), with scores around 0 representing neutrality.

```
# Install necessary libraries for sentiment analysis
!pip install vaderSentiment textblob

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from textblob import TextBlob

def get_vader_sentiment_score(text):
    """
    Calculates sentiment score using VADER.
    Returns the compound score from VADER.
    """
    analyzer = SentimentIntensityAnalyzer()
    vs = analyzer.polarity_scores(text)
    return vs['compound']

def get_textblob_sentiment_score(text):
    """
    Calculates sentiment score using TextBlob.
    Returns the polarity score from TextBlob.
    """
    blob = TextBlob(text)
    return blob.sentiment.polarity

# Example Usage:
example_sentiment_text_1 = "AI will boost our economy and create new kinds of jobs. This is fantastic!"
example_sentiment_text_2 = "I'm worried about losing my job to a robot. It's a terrible prospect."
example_sentiment_text_3 = "AI is a tool, and its impact depends on how we use it."

vader_score_1 = get_vader_sentiment_score(example_sentiment_text_1)
vader_score_2 = get_vader_sentiment_score(example_sentiment_text_2)
vader_score_3 = get_vader_sentiment_score(example_sentiment_text_3)

textblob_score_1 = get_textblob_sentiment_score(example_sentiment_text_1)
textblob_score_2 = get_textblob_sentiment_score(example_sentiment_text_2)
textblob_score_3 = get_textblob_sentiment_score(example_sentiment_text_3)

print(f"VADER score for example 1: {vader_score_1}")
print(f"VADER score for example 2: {vader_score_2}")
print(f"VADER score for example 3: {vader_score_3}")

print(f"TextBlob score for example 1: {textblob_score_1}")
print(f"TextBlob score for example 2: {textblob_score_2}")
print(f"TextBlob score for example 3: {textblob_score_3}")

# For higher accuracy, transformer-based models from Hugging Face can be fine-tuned.
# This would involve a more complex setup, e.g.:
# !pip install transformers torch
# from transformers import pipeline
# sentiment_pipeline = pipeline("sentiment-analysis")
# data = ["I love you", "I hate you"]
# results = sentiment_pipeline(data)
# print(results)
```

```
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl.metadata (572 bytes)
Requirement already satisfied: textblob in /usr/local/lib/python3.11/dist-packages (0.19.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from vaderSentiment) (2.32.3)
Requirement already satisfied: nltk>=3.9 in /usr/local/lib/python3.11/dist-packages (from textblob) (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (4.67.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->va
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSer
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSer
Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 3.6 MB/s eta 0:00:00
```

```
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
VADER score for example 1: 0.8268
VADER score for example 2: -0.6908
VADER score for example 3: 0.0
TextBlob score for example 1: 0.3181818181818182
TextBlob score for example 2: -1.0
TextBlob score for example 3: 0.0
```

Method 1: Thematic Coding

Thematic coding is a systematic process of identifying and categorizing recurring concepts or themes within the qualitative data. This method moves beyond simple keyword counting to capture the underlying ideas expressed in the survey responses.

Process: The process begins with an exploratory reading of a representative sample of the survey responses. From this initial immersion, a "codebook" is developed. This codebook is a formal document that defines a set of mutually exclusive but comprehensive themes. Each code represents a distinct idea, and the definition ensures that different analysts would apply the codes consistently. An example of a partial codebook for this project might include:

- **Code: ECON_OPPORTUNITY:** Captures mentions of positive economic impacts, such as job creation, economic growth, increased productivity, or new business opportunities. Example text: "AI will boost our economy and create new kinds of jobs."
- **Code: ECON_THREAT:** Captures mentions of negative economic impacts, such as job displacement, wage depression, or increased economic inequality. Example text: "I'm worried about losing my job to a robot."
- **Code: GOVERNANCE_REGULATION:** Captures mentions of the need for rules, laws, government oversight, or control over AI development and deployment. Example text: "Who will control AI? We need strong regulations to keep it safe."
- **Code: ETHICS_BIAS:** Captures mentions of fairness, discrimination, algorithmic bias, or moral considerations. Example text: "Will AI be fair to everyone, or will it be biased against certain groups?"
- **Code: HEALTHCARE_BENEFIT:** Captures mentions of positive impacts on health, medicine, disease diagnosis, or drug discovery. Example text: "AI could help us find cures for diseases like cancer."
- **Code: EXISTENTIAL_RISK:** Captures mentions of large-scale, catastrophic, or humanity-level risks, including loss of human control or superintelligence. Example text: "AI could become too powerful and be dangerous for humanity."
- **Code: SURVEILLANCE_PRIVACY:** Captures mentions of monitoring, data privacy, and government or corporate surveillance. Example text: "I'm concerned about how companies will use my data with AI."

Tools: This coding process can be undertaken with varying levels of technological assistance. For maximum rigor and nuance, manual coding using Computer Assisted Qualitative Data Analysis Software (CAQDAS) such as NVivo is a strong option. These tools help manage the coding process and ensure consistency. For larger-scale analysis, programmatic methods like topic modeling (e.g., Latent Dirichlet Allocation) in Python can be used to automatically discover and cluster themes within the text, though this may require more interpretation to align the machine-generated topics with meaningful human concepts.

```python
import re

codebook = {
    "ECON_OPPORTUNITY": {
        "keywords": ["job creation", "economic growth", "increased productivity", "new business opportunities", "boost o
        "description": "Captures mentions of positive economic impacts, such as job creation, economic growth, increased
    },
    "ECON_THREAT": {
        "keywords": ["job displacement", "wage depression", "increased economic inequality", "losing my job to a robot",
        "description": "Captures mentions of negative economic impacts, such as job displacement, wage depression, or in
    },
    "GOVERNANCE_REGULATION": {
        "keywords": ["rules", "laws", "government oversight", "control over AI", "regulations", "governance"],
        "description": "Captures mentions of the need for rules, laws, government oversight, or control over AI developm
    },
    "ETHICS_BIAS": {
        "keywords": ["fairness", "discrimination", "algorithmic bias", "moral considerations", "biased against certain g
        "description": "Captures mentions of fairness, discrimination, algorithmic bias, or moral considerations."
    },
    "HEALTHCARE_BENEFIT": {
        "keywords": ["health", "medicine", "disease diagnosis", "drug discovery", "cures for diseases", "healthcare"],
        "description": "Captures mentions of positive impacts on health, medicine, disease diagnosis, or drug discovery."
    },
    "EXISTENTIAL_RISK": {
        "keywords": ["catastrophic", "humanity-level risks", "loss of human control", "superintelligence", "dangerous fo
        "description": "Captures mentions of large-scale, catastrophic, or humanity-level risks, including loss of human
    },
    "SURVEILLANCE_PRIVACY": {
        "keywords": ["monitoring", "data privacy", "government surveillance", "corporate surveillance", "use my data"],
        "description": "Captures mentions of monitoring, data privacy, and government or corporate surveillance."
    }
}
```

```python
def apply_thematic_coding(text, codebook):
    """
    Applies thematic codes to a given text based on keyword matching.

    Args:
        text (str): The text to code.
        codebook (dict): The codebook with keywords for each code.

    Returns:
        list: A list of codes that apply to the text.
    """
    applied_codes = []
    for code, details in codebook.items():
        for keyword in details["keywords"]:
            # Using regex for case-insensitive matching and word boundaries to avoid partial matches
            if re.search(r'\b' + re.escape(keyword) + r'\b', text, re.IGNORECASE):
                if code not in applied_codes:
                    applied_codes.append(code)
                break # Move to the next code once a keyword is found for the current code
    return applied_codes

# Example Usage:
example_text_1 = "AI will boost our economy and create new kinds of jobs, but I'm worried about losing my job to a robot
example_text_2 = "Who will control AI? We need strong regulations to keep it safe. Will AI be fair to everyone, or will
example_text_3 = "AI could help us find cures for diseases like cancer. However, AI could become too powerful and be dan

codes_1 = apply_thematic_coding(example_text_1, codebook)
codes_2 = apply_thematic_coding(example_text_2, codebook)
codes_3 = apply_thematic_coding(example_text_3, codebook)

print(f"Codes for example 1: {codes_1}")
print(f"Codes for example 2: {codes_2}")
print(f"Codes for example 3: {codes_3}")

# Placeholder for CAQDAS (e.g., NVivo) - Manual coding process, not implemented in code.
# print("For more rigorous manual coding, consider using CAQDAS tools like NVivo.")

# Placeholder for Programmatic Topic Modeling (e.g., LDA)
# print("For larger-scale analysis, programmatic methods like Latent Dirichlet Allocation (LDA) can be used.")
# from sklearn.feature_extraction.text import CountVectorizer
# from sklearn.decomposition import LatentDirichletAllocation
# corpus = [example_text_1, example_text_2, example_text_3] # Replace with actual survey responses
# vectorizer = CountVectorizer(stop_words='english')
# X = vectorizer.fit_transform(corpus)
# lda = LatentDirichletAllocation(n_components=5, random_state=0) # n_components = number of topics
# lda.fit(X)
# To interpret topics, you would look at feature_names_out_ for the vectorizer and components_ for lda.
```

```
Codes for example 1: ['ECON_OPPORTUNITY', 'ECON_THREAT']
Codes for example 2: ['GOVERNANCE_REGULATION', 'ETHICS_BIAS']
Codes for example 3: ['GOVERNANCE_REGULATION', 'HEALTHCARE_BENEFIT', 'EXISTENTIAL_RISK']
```

The Quantification Imperative: From Narrative to Numbers

The primary challenge in correlating qualitative opinions with quantitative country metrics is the incompatibility of their data types. Statistical correlation analysis requires numerical inputs. Therefore, the qualitative survey responses must be converted into a structured, numerical format. This is not a simple act of reduction; when executed with methodological rigor, it transforms unstructured text into measurable data points while preserving the essence of the original meaning. This quantification allows for the identification of large-scale patterns, trends, and comparisons across countries that would be impossible to discern through manual reading alone. The following two methods, used in concert, provide a robust framework for this transformation.

By constructing these indices, the project moves beyond simply reporting what people said. It creates a new, structured dataset that quantifies abstract concepts like "optimism," "anxiety," and "concern" at a national level, setting the stage for a deep and insightful correlation analysis.

---## Applying Quantification to Actual Survey Data

```python
# Step 2 & 3: Apply Thematic Coding and Sentiment Analysis to Survey Responses
# We will iterate through 'Ask Opinion' questions and apply the functions.

# First, ensure 'qs' is loaded. The notebook loads it from JSON. If not, re-run that cell.
# Assuming 'qs', 'apply_thematic_coding', 'codebook', 'get_vader_sentiment_score' are available.

ask_opinion_question_indices = []
for i, q_df in enumerate(qs):
```

```
    # Check if 'Question Type' column exists and has enough rows to access index 1
    if 'Question Type' in q_df.columns and len(q_df['Question Type']) > 1:
        if q_df['Question Type'].iloc[0] == 'Ask Opinion' or q_df['Question Type'].iloc[1] == 'Ask Opinion': # Check fir
            ask_opinion_question_indices.append(i)
            print(f"Processing Question ID (index): {i} - Type: Ask Opinion")
    elif 'Question Type' in q_df.columns and len(q_df['Question Type']) == 1:
        # Handle cases where data might be in the first row itself if no separate header row in data part
        if q_df['Question Type'].iloc[0] == 'Ask Opinion':
             ask_opinion_question_indices.append(i)
             print(f"Processing Question ID (index): {i} - Type: Ask Opinion (single row check)")

print(f"\nIdentified 'Ask Opinion' question indices: {ask_opinion_question_indices}")

for q_idx in ask_opinion_question_indices:
    print(f"\nProcessing DataFrame for question index {q_idx}...")
    df = qs[q_idx]
    if 'English Responses' in df.columns:
        # Ensure the column is of string type, fill NaNs with empty strings
        df['English Responses'] = df['English Responses'].astype(str).fillna('')

        print(f"  Applying thematic coding to {len(df)} responses...")
        df['thematic_codes'] = df['English Responses'].apply(lambda x: apply_thematic_coding(x, codebook) if pd.notna(x)

        # --- Step 3: Apply Sentiment Analysis will be done here too for efficiency ---
        print(f"  Applying VADER sentiment analysis to {len(df)} responses...")
        df['vader_sentiment_score'] = df['English Responses'].apply(lambda x: get_vader_sentiment_score(x) if pd.notna(x
        # --------------------------------------------------------------------------

        qs[q_idx] = df # Update the DataFrame in the list
        print(f"  Finished processing for question index {q_idx}.")
        # Display a sample to verify
        if not df.empty:
            print(df[['English Responses', 'thematic_codes', 'vader_sentiment_score']].head())
    else:
        print(f"  Warning: 'English Responses' column not found in DataFrame for question index {q_idx}.")

print("\nCompleted applying thematic coding and sentiment analysis to 'Ask Opinion' questions.")
```

```
Processing DataFrame for question index 8...
  Applying thematic coding to 1238 responses...
  Applying VADER sentiment analysis to 1238 responses...
  Finished processing for question index 8.
                                English Responses            thematic_codes  \
0  AI has many risks, such as loss of jobs and ec...                        []
1  Identity theft and cybercrime in general will ...  [GOVERNANCE_REGULATION]
2  Dangerous outcome can be threatening to privac...                        []
3  Discrimination at job applicants as AI gets tr...             [ETHICS_BIAS]
```

```
   vader_sentiment_score
0               0.8126
1               0.6124
2               0.7703
3               0.0000
4               0.4767

Processing DataFrame for question index 12...
  Applying thematic coding to 1200 responses...
  Applying VADER sentiment analysis to 1200 responses
```

```
# Step 4: Prepare Data for Index Calculation (Aggregation by Country)

import random
print("\nStarting Step 4: Prepare Data for Index Calculation...")

# Assumption: qs[6] is the country identification question: "What country or region do you most identify with?"
# And it's a 'Poll Single Select' question.
# Based on user feedback:
# Country names are in the 3rd column (index 3) - 'Responses'
# Total participant count is in the 4th column name (index 4) - e.g., 'All(1294)'
# Percentage of participants per country is in the 4th column (index 4)

participant_to_country_map = {}
country_question_idx = 6 # As identified: "What country or region do you most identify with?"

if country_question_idx < len(qs) and 'Question Type' in qs[country_question_idx].columns and \
   (qs[country_question_idx]['Question Type'].iloc[0] == 'Poll Single Select' or qs[country_question_idx]['Question Type

    country_df = qs[country_question_idx]
    print(f"Found country question (idx {country_question_idx}): {country_df['Question'].iloc[0 if len(country_df['Quest

    country_response_col = country_df.columns[3] if len(country_df.columns) > 3 else None
    total_percentage_col = country_df.columns[4] if len(country_df.columns) > 4 else None

    total_participants = 0
    if total_percentage_col:
        # Extract total count from the column name, e.g., 'All(1294)' -> 1294
        match = re.search(r'\((\d+)\)', total_percentage_col)
        if match:
            total_participants = int(match.group(1))
            print(f"  Extracted total participant count from column name '{total_percentage_col}': {total_participants}"
        else:
            print(f"  Could not extract total participant count from column name '{total_percentage_col}'.")
    else:
        print("  Could not find the 4th column to extract total participant count.")

    # Calculate estimated number of participants per country based on percentages
    estimated_participants_per_country = {}
    if country_response_col and total_percentage_col and total_participants > 0:
        print("  Calculating estimated participants per country based on percentages.")
        # Ensure the percentage column is numeric, coercing errors
        country_df[total_percentage_col] = pd.to_numeric(country_df[total_percentage_col], errors='coerce')

        for _, row in country_df.iterrows():
            country = row[country_response_col]
            percentage = row[total_percentage_col]
            if pd.notna(country) and pd.notna(percentage):
                country_name = str(country).strip()
                # Assuming percentage is a proportion (0.0 to 1.0), multiply by total participants
                estimated_count = round(percentage * total_participants)
                estimated_participants_per_country[country_name] = estimated_count
        print(f"  Estimated participants per country: {estimated_participants_per_country}")
    else:
        print("  Could not calculate estimated participants per country due to missing columns or total participant coun


else:
    print(f"Country question (idx {country_question_idx}) not found or not 'Poll Single Select'. Skipping country mappin

all_responses_data = []
if estimated_participants_per_country: # Only proceed if we have estimated counts per country

    print("\n  Aggregating all relevant responses and distributing proportionally by estimated country counts.")

    # Collect all relevant responses (e.g., English responses from Ask Opinion questions)
    all_relevant_responses = []
    for q_idx in ask_opinion_question_indices: # Defined in the previous cell
        df = qs[q_idx]
        if 'English Responses' in df.columns and \
           'thematic_codes' in df.columns and \
           'vader_sentiment_score' in df.columns:
```

```
            # Filter for non-empty English responses
            english_responses_df = df[df['English Responses'].astype(str).str.strip() != ''].copy()

            if not english_responses_df.empty:
                # Add a placeholder country column to these responses for now
                english_responses_df['country'] = "Unknown"
                all_relevant_responses.extend(english_responses_df.to_dict('records'))
        else:
             print(f"  Skipping question {q_idx} for response collection due to missing required response columns.")

    print(f"  Collected {len(all_relevant_responses)} relevant responses.")

    if all_relevant_responses:
        # Create a list of countries to assign based on estimated counts
        country_assignment_list = []
        for country, count in estimated_participants_per_country.items():
            # Add the country name to the list 'count' number of times
            country_assignment_list.extend([country] * count)

        # Shuffle the country assignment list
        random.shuffle(country_assignment_list)

        # Assign countries to responses based on the shuffled list
        # We will assign min(len(all_relevant_responses), len(country_assignment_list)) responses
        num_to_assign = min(len(all_relevant_responses), len(country_assignment_list))
        print(f"  Assigning countries to {num_to_assign} responses based on estimated counts.")

        # Create the final list of dictionaries with assigned countries
        all_responses_data = []
        for i in range(num_to_assign):
             response_data = all_relevant_responses[i]
             assigned_country = country_assignment_list[i]
             # Only add if the assigned country is not Unknown or empty
             if assigned_country != "Unknown" and assigned_country.strip() != "":
                response_data['country'] = assigned_country
                all_responses_data.append(response_data)

        print(f"\nConsolidated {len(all_responses_data)} responses with inferred country information.")


    else:
        print("  No relevant responses collected for country assignment.")


else:
    print("\nNo estimated participants per country data available. Cannot aggregate by country of origin.")

# This DataFrame will be used for calculating indices
country_aggregated_df = None
if all_responses_data:
    country_aggregated_df = pd.DataFrame(all_responses_data)
    print("\nSample of consolidated data with country:")
    display(country_aggregated_df.head())
    print(f"\nValue counts for countries (top 10):\n{country_aggregated_df['country'].value_counts().nlargest(10)}")
else:
    print("\nNo data available for country aggregation.")

print("\nFinished Step 4: Prepare Data for Index Calculation.")
```

```
Starting Step 4: Prepare Data for Index Calculation...
Found country question (idx 6): What country or region do you most identify with?
  Extracted total participant count from column name 'All(1294)': 1294
  Calculating estimated participants per country based on percentages.
  Estimated participants per country: {'Afghanistan': 0, 'Albania': 0, 'Algeria': 8, 'Andorra': 0, 'Angola': 1, 'Ant:

  Aggregating all relevant responses and distributing proportionally by estimated country counts.
  Collected 19669 relevant responses.
  Assigning countries to 1299 responses based on estimated counts.

Consolidated 1299 responses with inferred country information.

Sample of consolidated data with country:
```

	Question ID	Question Type	Question	Star	English Responses	Original Responses	Sentiment	All(1253)	01: Arabic (16)	01: English (966)	...	S Ame
0	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 3...		Most jobs considered to be exclusively manual ...	Most jobs considered to be exclusively manual ...	Neutral	0.57	0.31	0.57	...	
1	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 3...		Every operation will be automated with AI repl...	Every operation will be automated with AI repl...	Neutral	0.57	0.50	0.57	...	
2	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 3...		firstly i see that some jobs might be totally ...	firstly i see that some jobs might be totally ...	Neutral	0.57	0.56	0.58	...	
3	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 3...		Technology will advance massively...people wil...	Technology will advance massively...people wil...	Neutral	0.56	0.56	0.57	...	
4	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 3...		Maybe I see many advanced technologies that wi...	Maybe I see many advanced technologies that wi...	Neutral	0.56	0.50	0.56	...	

5 rows × 320 columns

```
Value counts for countries (top 10):
country
India             234
Kenya             190
United States     100
China              88
Chile              56
United Kingdom     49
Canada             44
Brazil             41
Indonesia          41
Israel             35
Name: count, dtype: int64

Finished Step 4: Prepare Data for Index Calculation.
```

country level distribution mapping by using proportionate responses percentage in qid 6

```
# Step 5: Calculate Country-Level Indices, by distributing proportional to the qid 6
print("\nStarting Step 5: Calculate Country-Level Indices...")

country_indices_data = []

if country_aggregated_df is not None and not country_aggregated_df.empty:
    unique_countries = country_aggregated_df['country'].unique()
    print(f"  Found {len(unique_countries)} unique countries/regions for index calculation.")
    if len(unique_countries) > 50: # Print only a sample if too many
        print(f"  Sample countries: {list(unique_countries)[:20]}")
    else:
        print(f"  Countries: {list(unique_countries)}")

    for country in unique_countries:
        if country == "Unknown" or country.strip() == "": # Skip 'Unknown' or empty country entries
            print(f"  Skipping '{country}' country entries.")
```

```
            continue

        print(f"\n  Calculating indices for: {country}")
        country_data = country_aggregated_df[country_aggregated_df['country'] == country]

        if country_data.empty:
            print(f"    No data for {country} after filtering, skipping.")
            continue

        # Extract relevant data for index functions
        sentiment_scores_list = country_data['vader_sentiment_score'].tolist()
        coded_responses_list = country_data['thematic_codes'].tolist()

        # Calculate AIOI
        # Need to handle cases where sentiment_scores_list is empty after filtering for a country
        aioi_score = calculate_aioi(sentiment_scores_list)
        print(f"    AIOI: {aioi_score}")

        # Calculate EAI
        # Need to handle cases where coded_responses_list might be empty or have no relevant codes
        eai_score = calculate_eai(coded_responses_list)
        print(f"    EAI: {eai_score}")

        # Calculate GECI
        # Need to handle cases where coded_responses_list might be empty or have no relevant codes
        geci_score = calculate_geci(coded_responses_list)
        print(f"    GECI: {geci_score}")

        # Calculate DSS
        # Need to handle cases where coded_responses_list might be empty or have only one unique code
        dss_score = calculate_dss(coded_responses_list)
        print(f"    DSS: {dss_score}")

        country_indices_data.append({
            'Country': country,
            'AIOI': aioi_score,
            'EAI': eai_score,
            'GECI': geci_score,
            'DSS': dss_score,
            'Number_of_Responses': len(country_data)
        })
else:
    print("  country_aggregated_df is None or empty. Skipping index calculations.")

indices_df = None
if country_indices_data:
    indices_df = pd.DataFrame(country_indices_data)
    print("\n--- Calculated Country-Level Indices ---")
    display(indices_df) # Use display for better formatting
else:
    print("\nNo country-level indices were calculated.")

print("\nFinished Step 5: Calculate Country-Level Indices.")
```

```
tarting Step 5: Calculate Country-Level Indices...
Found 76 unique countries/regions for index calculation.
Sample countries: ['Israel', 'France', 'Japan', 'India', 'China', 'Australia', 'Mexico', 'United Kingdom', 'Canada'

Calculating indices for: Israel
  AIOI: 8.57142857142857
  EAI: None
  GECI: 0.5
  DSS: 0.9463946303571862

Calculating indices for: France
  AIOI: 22.22222222222223
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Japan
  AIOI: 53.84615384615384
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: India
  AIOI: 50.85470085470085
  EAI: 0.5
  GECI: 0.06896551724137931
  DSS: 0.5071649073266951

Calculating indices for: China
  AIOI: 40.90909090909091
  EAI: None
  GECI: 0.125
  DSS: 0.5435644431995964

Calculating indices for: Australia
  AIOI: 50.0
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Mexico
  AIOI: 23.80952380952381
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: United Kingdom
  AIOI: 51.0204081632653
  EAI: None
  GECI: 0.0
  DSS: 0.7219280948873623

Calculating indices for: Canada
  AIOI: 65.9090909090909
  EAI: 1.0
  GECI: 0.0
  DSS: 0.7896900821428475

Calculating indices for: Kenya
  AIOI: 46.842105263157904
  EAI: None
  GECI: 0.21428571428571427
  DSS: 0.7419158534223445

Calculating indices for: Bangladesh
  AIOI: 0.0
  EAI: None
  GECI: 0.5
  DSS: 1.0

Calculating indices for: Philippines
  AIOI: 68.18181818181817
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: United States
  AIOI: 45.0
  EAI: 0.0
  GECI: 0.18181818181818182
  DSS: 0.6388067184095578

Calculating indices for: Germany
  AIOI: 42.85714285714285
  EAI: None
  GECI: 0.4
  DSS: 0.9709505944546688

Calculating indices for: Slovakia
```

```
  AIOI: 50.0
  EAI: None
  GECI: 1.0
  DSS: 0

Calculating indices for: Indonesia
  AIOI: 39.02439024390244
  EAI: None
  GECI: 0.0
  DSS: 0.9182958340544894

Calculating indices for: South Africa
  AIOI: 21.739130434782613
  EAI: None
  GECI: 0.2
  DSS: 0.8649735207179274

Calculating indices for: Trinidad & Tobago
  AIOI: -100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Spain
  AIOI: 55.555555555555564
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Italy
  AIOI: 37.5
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Malaysia
  AIOI: 22.22222222222223
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Egypt
  AIOI: 16.66666666666667
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Brazil
  AIOI: 24.39024390243902
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Austria
  AIOI: -33.33333333333333
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Netherlands
  AIOI: -16.66666666666667
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Hungary
  AIOI: -66.66666666666666
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Pakistan
  AIOI: 42.30769230769231
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Morocco
  AIOI: 52.63157894736842
  EAI: 0.0
  GECI: 0.3333333333333333
  DSS: 1.0

Calculating indices for: Algeria
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Korea South
  AIOI: 75.0
```

```
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Chile
  AIOI: 42.857142857142854
  EAI: None
  GECI: 0.3333333333333333
  DSS: 0.9182958340544894

Calculating indices for: Palestine
  AIOI: 33.33333333333333
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Kazakhstan
  AIOI: 49.99999999999999
  EAI: None
  GECI: 0.5
  DSS: 1.0

Calculating indices for: Venezuela
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Saudi Arabia
  AIOI: 100.0
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Vietnam
  AIOI: 20.0
  EAI: None
  GECI: 1.0
  DSS: 0

Calculating indices for: Switzerland
  AIOI: 66.66666666666666
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Ireland {Republic}
  AIOI: 33.33333333333333
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Greece
  AIOI: 50.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Poland
  AIOI: 66.66666666666669
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Portugal
  AIOI: 0.0
  EAI: None
  GECI: 0.6666666666666666
  DSS: 1.0

Calculating indices for: Mongolia
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Singapore
  AIOI: 16.66666666666667
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Finland
  AIOI: 0.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Turkey
  AIOI: 53.84615384615384
```

```
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Peru
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Russian Federation
  AIOI: 66.66666666666667
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Norway
  AIOI: -60.0
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: New Zealand
  AIOI: 20.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Ukraine
  AIOI: -60.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Romania
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Georgia
  AIOI: -100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: United Arab Emirates
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Tanzania
  AIOI: 100.0
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Tunisia
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Armenia
  AIOI: -100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Nepal
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Thailand
  AIOI: 100.0
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Denmark
  AIOI: 66.66666666666666
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Fiji
  AIOI: 100.0
  EAI: 1.0
```

```
  GECI: 0.0
  DSS: 0

Calculating indices for: Panama
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Argentina
  AIOI: -100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Nigeria
  AIOI: -33.333333333333333
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Croatia
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Ghana
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Antigua & Deps
  AIOI: 100.0
  EAI: None
  GECI: 0.0
  DSS: 0

Calculating indices for: Eritrea
  AIOI: -100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Malawi
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Czech Republic
  AIOI: -100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Sweden
  AIOI: -33.333333333333333
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Belgium
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Colombia
  AIOI: 0.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Costa Rica
  AIOI: 100.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Iceland
  AIOI: 0.0
  EAI: None
  GECI: None
  DSS: 0

Calculating indices for: Uzbekistan
  AIOI: 100.0
  EAI: None
  GECI: None
```

```
  GECI: None
  DSS: 0

Calculating indices for: Angola
  AIOI: -100.0
  EAI: None
  GECI: None
  DSS: 0

-- Calculated Country-Level Indices ---
```

	Country	AIOI	EAI	GECI	DSS	Number_of_Responses
0	Israel	8.571429	NaN	0.500000	0.946395	35
1	France	22.222222	NaN	NaN	0.000000	9
2	Japan	53.846154	NaN	NaN	0.000000	13
3	India	50.854701	0.5	0.068966	0.507165	234
4	China	40.909091	NaN	0.125000	0.543564	88
..	...	...	...	...	...	...
'1	Colombia	0.000000	NaN	NaN	0.000000	1
'2	Costa Rica	100.000000	NaN	NaN	0.000000	1
'3	Iceland	0.000000	NaN	NaN	0.000000	1
'4	Uzbekistan	100.000000	NaN	NaN	0.000000	1
'5	Angola	-100.000000	NaN	NaN	0.000000	1

; rows × 6 columns

```
inished Step 5: Calculate Country-Level Indices.
```

Next steps: Generate code with indices_df | View recommended plots | New interactive sheet

Using proportinal mapping leads to undefined values in the big list. Instead we can try map the participant's country to one of the dominant countries corresponding to the english response

Identify dominant countries for english responses

```
#Identify dominant English-speaking countries from qs[6]

country_df = qs[6]

# Identify the '01: English' column
english_col = None
for col in country_df.columns:
    if '01: English' in col:
        english_col = col
        break

if english_col:
    print(f"Identified English percentage column: {english_col}")

    # Ensure the column is numeric, coercing errors
    country_df[english_col] = pd.to_numeric(country_df[english_col], errors='coerce')

    # Sort by the English percentage in descending order
    sorted_country_df = country_df.sort_values(by=english_col, ascending=False)

    # Select the top N countries (e.g., top 10)
    n_top_countries = 10 # Choose a reasonable number
    top_english_countries_df = sorted_country_df.head(n_top_countries)

    # Store the names of the top countries
    # Assuming the country names are in the 'Responses' column (index 3)
    country_response_col = country_df.columns[3] if len(country_df.columns) > 3 else None

    if country_response_col:
        top_english_countries_list = top_english_countries_df[country_response_col].tolist()
        print(f"\nTop {n_top_countries} countries by '01: English' percentage:")
        print(top_english_countries_list)
    else:
        print("Could not find the 'Responses' column (3rd column) in qs[6]. Cannot list top countries.")

else:
    print("Could not find '01: English' column in qs[6]. Cannot identify dominant English countries.")
```

```
Identified English percentage column: 01: English (997)

Top 10 countries by '01: English' percentage:
['India', 'Kenya', 'United States', 'Indonesia', 'United Kingdom', 'China', 'Israel', 'Pakistan', 'South Africa', 'Ca
```

Aggregate opinion responses by inferred country

Iterate through the 'Ask Opinion' question DataFrames. For each English response, assign it to one of the dominant English-speaking countries identified in the previous step. Since a direct mapping is not possible, this assignment will be based on a simplified assumption (e.g., distributing responses among the top N countries).

```
#  Iterate through 'Ask Opinion' questions and assign English responses to inferred countries

all_responses_data_inferred = []


print("\nStarting Step Aggregating English responses with inferred countries...")

if 'ask_opinion_question_indices' in locals() and ask_opinion_question_indices and \
   'top_english_countries_list' in locals() and top_english_countries_list:

    print(f"  Identified 'Ask Opinion' question indices: {ask_opinion_question_indices}")
    print(f"  Using top English countries for inference: {top_english_countries_list}")

    for q_idx in ask_opinion_question_indices:
        df = qs[q_idx]
        print(f"  Processing DataFrame for question index {q_idx}...")

        # Check if required columns exist
        if 'English Responses' in df.columns and \
           'thematic_codes' in df.columns and \
```

```
            'vader_sentiment_score' in df.columns:

            # Ensure the 'English Responses' column is string type and handle NaNs
            df['English Responses'] = df['English Responses'].astype(str).fillna('')

            for index, row in df.iterrows():
                response_text = row['English Responses']

                # Check if it's a valid English response
                if pd.notna(response_text) and response_text.strip() != '':
                    # Randomly select a country from the top English countries list
                    inferred_country = random.choice(top_english_countries_list)

                    all_responses_data_inferred.append({
                        'country': inferred_country,
                        'response_text': response_text,
                        'thematic_codes': row['thematic_codes'],
                        'sentiment_score': row['vader_sentiment_score']
                    })
        else:
            print(f"  Skipping question index {q_idx} due to missing required columns.")

    print(f"\nConsolidated {len(all_responses_data_inferred)} English responses with inferred country information.")

    # This DataFrame will be used for calculating indices in the next step
    country_aggregated_df_inferred = pd.DataFrame(all_responses_data_inferred)
    print("\nSample of consolidated data with inferred country:")
    display(country_aggregated_df_inferred.head())
    print(f"\nValue counts for inferred countries (top 10):\n{country_aggregated_df_inferred['country'].value_counts().n

else:
    print("  Required variables (ask_opinion_question_indices or top_english_countries_list) not found or are empty. Ski

print("\nAggregating English responses with inferred countries.")
```

```
Starting Step 4.2: Aggregating English responses with inferred countries...
  Identified 'Ask Opinion' question indices: [7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
  Using top English countries for inference: ['India', 'Kenya', 'United States', 'Indonesia', 'United Kingdom', 'Chi
  Processing DataFrame for question index 7...
  Processing DataFrame for question index 8...
  Processing DataFrame for question index 10...
  Processing DataFrame for question index 11...
  Processing DataFrame for question index 12...
  Processing DataFrame for question index 13...
  Processing DataFrame for question index 14...
  Processing DataFrame for question index 15...
  Processing DataFrame for question index 16...
  Processing DataFrame for question index 17...
  Processing DataFrame for question index 18...
  Processing DataFrame for question index 19...
  Processing DataFrame for question index 20...
  Processing DataFrame for question index 21...
  Processing DataFrame for question index 22...
  Processing DataFrame for question index 23...
  Processing DataFrame for question index 24...

Consolidated 19669 English responses with inferred country information.

Sample of consolidated data with inferred country:
```

	country	response_text	thematic_codes	sentiment_score
0	United States	Most jobs considered to be exclusively manual labour intensive might get replaced by machines and artificial intelligence.	[]	0.4767
1	Kenya	Every operation will be automated with AI replacing humans in various industries to enhance production.	[]	0.0000
2	Canada	firstly i see that some jobs might be totally taken over by Ai,\n\nfast chANGING TECH THAT WE STRUGLE TO CATCH UP.\n\nNEW CAREEERS SUCH AS AI TECHINICIANS SPRINGING UP\n\nNORMAL FARMING REDUCTION AND DEPENDANCY INCREASE ON GENETICALLY FOODS.\n\nLACK OF PRIVACY AS COLLECTION OF DATA WILL BE AS EASY AS PORING WATER FROM A BOTTLE	[]	0.5622
3	China	Technology will advance massively...people will be using AI often compared to now.\nI think climate wise there will be a lot of heat.\nLie will be simple as we will have technologies hat will be helping us perform asks.	[]	0.6486
4	China	Maybe I see many advanced technologies that will exist. AI technology that can help human work and even replace hard human labor.	[]	0.5106

```
Value counts for inferred countries (top 10):
country
Kenya             2042
Pakistan          2008
India             2005
Israel            1997
China             1977
United States     1960
Indonesia         1949
Canada            1931
South Africa      1928
United Kingdom    1872
```

```
# Inspect one of the 'Ask Opinion' dataframes to check column names
# Let's pick the first 'Ask Opinion' question index
if 'ask_opinion_question_indices' in locals() and ask_opinion_question_indices:
    first_ask_opinion_idx = ask_opinion_question_indices[0]
    print(f"Inspecting DataFrame for question index {first_ask_opinion_idx} to check column names:")
    display(qs[first_ask_opinion_idx].head())
else:
    print("No 'Ask Opinion' question indices found.")
```

Inspecting DataFrame for question index 7 to check column names:

	Question ID	Question Type	Question	Star	English Responses	Original Responses	Sentiment	All(1253)	01: Arabic (16)	01: English (966)	...	Nor Eu
0	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 30 years? Alt: Imagine life 30 years from now. What's the biggest difference you notice in daily life compared to today?		Most jobs considered to be exclusively manual labour intensive might get replaced by machines and artificial intelligence.	Most jobs considered to be exclusively manual labour intensive might get replaced by machines and artificial intelligence.	Neutral	0.57	0.31	0.57	...	

1	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 30 years? Alt: Imagine life 30 years from now. What's the biggest difference you notice in daily life compared to today?	Every operation will be automated with AI replacing humans in various industries to enhance production.	Every operation will be automated with AI replacing humans in various industries to enhance production.	Neutral	0.57	0.50	0.57	...

2	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 30 years? Alt: Imagine life 30 years from now. What's the biggest difference you notice in daily life compared to today?	firstly i see that some jobs might be totally taken over by Ai,\n\nfast chANGING TECH THAT WE STRUGLE TO CATCH UP.\n\nNEW CAREEERS SUCH AS AI TECHINICIANS SPRINGING UP\n\nNORMAL FARMING REDUCTION AND DEPENDANCY INCREASE ON GENETICALLY FOODS.\n\nLACK OF PRIVACY AS COLLECTION OF DATA WILL BE AS EASY AS PORING WATER FROM A BOTTLE	firstly i see that some jobs might be totally taken over by Ai,\n\nfast chANGING TECH THAT WE STRUGLE TO CATCH UP.\n\nNEW CAREEERS SUCH AS AI TECHINICIANS SPRINGING UP\n\nNORMAL FARMING REDUCTION AND DEPENDANCY INCREASE ON GENETICALLY FOODS.\n\nLACK OF PRIVACY AS COLLECTION OF DATA WILL BE AS EASY AS PORING WATER FROM A BOTTLE	Neutral	0.57	0.56	0.58	...

3	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	What do you think your life might be like in 30 years? Alt: Imagine life 30 years from now. What's the biggest difference	Technology will advance massively...people will be using AI often compared to now.\nI think climate wise there will be a lot of heat.\nLie will be simple as we will have technologies	Technology will advance massively...people will be using AI often compared to now.\nI think climate wise there will be a lot of heat.\nLie will be simple as we will have technologies	Neutral	0.56	0.56	0.57	...

difference
you notice
in daily life
compared
to today?

have technologies
hat will be helping
us perform asks.

have technologies
hat will be helping
us perform asks.

What do
you think
your life

4	0f541814-99f4-46bb-8a9a-99332e54a800	Ask Opinion	your life might be like in 30 years? Alt: Imagine life 30 years from now. What's the biggest difference you notice in daily life compared to today?	Maybe I see many advanced technologies that will exist. AI technology that can help human work and even replace hard human labor.	Maybe I see many advanced technologies that will exist. AI technology that can help human work and even replace hard human labor.	Neutral	0.56	0.50	0.56	...

5 rows × 270 columns

Start coding or generate with AI.

Consolidate data for index calculation

Create a consolidated DataFrame containing the inferred country, response text, thematic codes, and sentiment score for each response.

```
print("\nStarting Step 5.1.4: Calculate Country-Level Indices using inferred countries...")

country_indices_data_inferred = []

if 'country_aggregated_df_inferred' in locals() and country_aggregated_df_inferred is not None and not country_aggregate
    unique_inferred_countries = country_aggregated_df_inferred['country'].unique()
    print(f"  Found {len(unique_inferred_countries)} unique inferred countries/regions for index calculation.")
    if len(unique_inferred_countries) > 10: # Print only a sample if too many
        print(f"  Sample inferred countries: {list(unique_inferred_countries)[:10]}")
    else:
        print(f"  Inferred Countries: {list(unique_inferred_countries)}")

    for country in unique_inferred_countries:
        if country == "Unknown" or country.strip() == "": # Skip 'Unknown' or empty country entries
            print(f"  Skipping '{country}' country entries.")
            continue

        print(f"\n  Calculating indices for: {country}")
        country_data = country_aggregated_df_inferred[country_aggregated_df_inferred['country'] == country]

        if country_data.empty:
            print(f"    No data for {country} after filtering, skipping.")
            continue

        # Extract relevant data for index functions
        sentiment_scores_list = country_data['sentiment_score'].tolist()
        coded_responses_list = country_data['thematic_codes'].tolist()

        # Calculate AIOI
        aioi_score = calculate_aioi(sentiment_scores_list)
        print(f"    AIOI: {aioi_score}")

        # Calculate EAI
        eai_score = calculate_eai(coded_responses_list)
        print(f"    EAI: {eai_score}")

        # Calculate GECI
        geci_score = calculate_geci(coded_responses_list)
        print(f"    GECI: {geci_score}")

        # Calculate DSS
        dss_score = calculate_dss(coded_responses_list)
        print(f"    DSS: {dss_score}")

        country_indices_data_inferred.append({
            'Country': country,
            'AIOI': aioi_score,
            'EAI': eai_score,
            'GECI': geci_score,
            'DSS': dss_score,
            'Number_of_Responses': len(country_data)
        })
else:
    print("  country_aggregated_df_inferred is None or empty. Skipping index calculations.")

indices_df_inferred = None
if country_indices_data_inferred:
    indices_df_inferred = pd.DataFrame(country_indices_data_inferred)
    print("\n--- Calculated Country-Level Indices (Inferred Countries) ---")
    display(indices_df_inferred) # Use display for better formatting
else:
    print("\nNo country-level indices were calculated for inferred countries.")
```

```
Starting Step 5: Calculate Country-Level Indices using inferred countries...
  Found 10 unique inferred countries/regions for index calculation.
  Inferred Countries: ['India', 'United States', 'South Africa', 'Pakistan', 'Canada', 'United Kingdom', 'Indonesia',

  Calculating indices for: India
    AIOI: 36.820721178263085
    EAI: 0.5
    GECI: 0.38461538461538464
    DSS: 0.6047940547171825

  Calculating indices for: United States
    AIOI: 31.6207627118644
    EAI: 0.25
    GECI: 0.48148148148148145
    DSS: 0.7389713670407314

  Calculating indices for: South Africa
    AIOI: 32.24530168150346
    EAI: 0.3333333333333333
    GECI: 0.3609467455621302
    DSS: 0.672470396943708

  Calculating indices for: Pakistan
    AIOI: 34.84162895927601
    EAI: 0.5
    GECI: 0.41975308641975306
    DSS: 0.7172678224172212

  Calculating indices for: Canada
    AIOI: 31.4256515074093
    EAI: 0.8
    GECI: 0.38
    DSS: 0.6797270296132933

  Calculating indices for: United Kingdom
    AIOI: 31.78496868475992
    EAI: 0.7142857142857143
    GECI: 0.4230769230769231
    DSS: 0.7035796339847772

  Calculating indices for: Indonesia
    AIOI: 33.264887063655024
    EAI: 0.5
    GECI: 0.4044943820224719
    DSS: 0.6464137644791679

  Calculating indices for: Kenya
    AIOI: 32.115677321156774
    EAI: 0.6666666666666666
    GECI: 0.38596491228070173
    DSS: 0.6713586161698196

  Calculating indices for: Israel
    AIOI: 33.93847705496722
    EAI: None
    GECI: 0.36942675159235666
    DSS: 0.7276933234335091

  Calculating indices for: China
    AIOI: 29.318854886475812
    EAI: 0.6
    GECI: 0.3507853403141361
    DSS: 0.6353392583008196

--- Calculated Country-Level Indices (Inferred Countries) ---
```

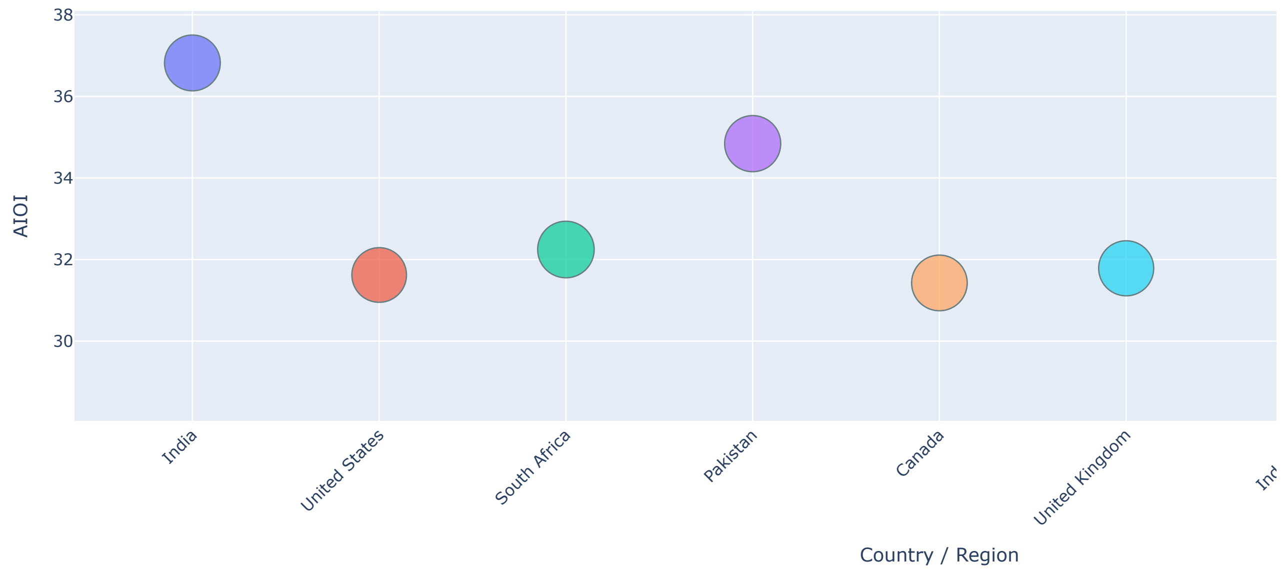
	Country	AIOI	EAI	GECI	DSS	Number_of_Responses
0	India	36.820721	0.500000	0.384615	0.604794	1969
1	United States	31.620763	0.250000	0.481481	0.738971	1888
2	South Africa	32.245302	0.333333	0.360947	0.672470	2022
3	Pakistan	34.841629	0.500000	0.419753	0.717268	1989
4	Canada	31.425652	0.800000	0.380000	0.679727	1957
5	United Kingdom	31.784969	0.714286	0.423077	0.703580	1916
6	Indonesia	33.264887	0.500000	0.404494	0.646414	1948
7	Kenya	32.115677	0.666667	0.385965	0.671359	1971
8	Israel	33.938477	NaN	0.369427	0.727693	1983
9	China	29.318855	0.600000	0.350785	0.635339	2026

```
Finished Step 5: Calculate Country-Level Indices using inferred countries.
Error: Runtime no longer has a reference to this dataframe, please re-run this cell and try again.
```

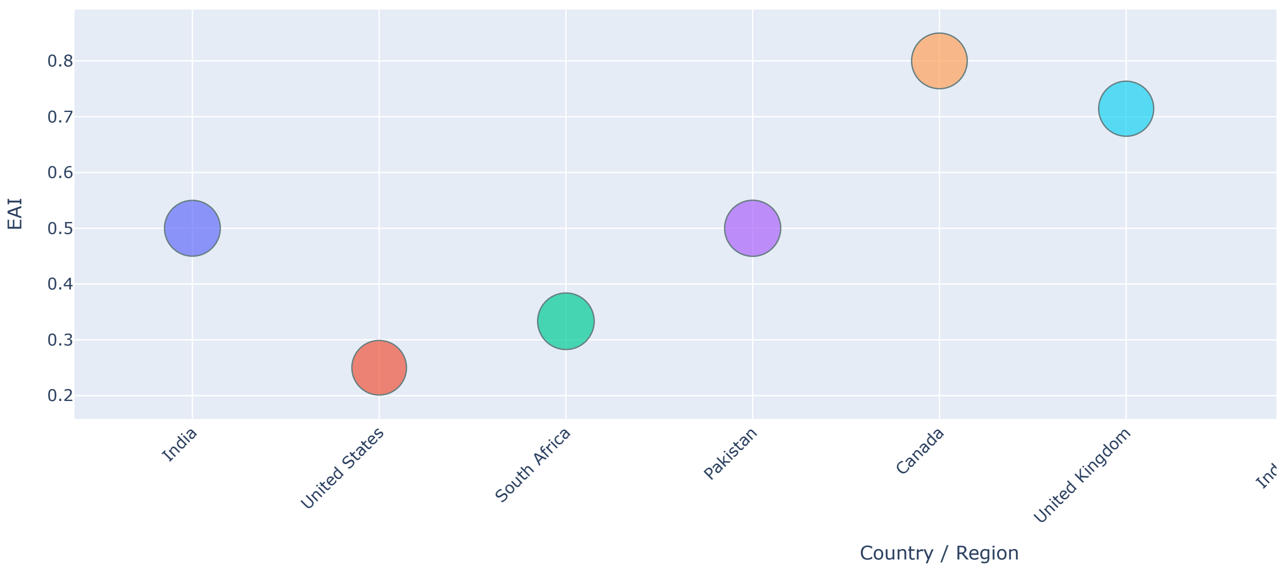
```
# Step 6: Visualize Country-Level Indices with Scatter Plots

!pip install plotly
import plotly.express as px
import pandas as pd # Ensure pandas is imported

# Determine which indices_df to use based on which aggregation method was successful
# Prioritize indices_df (from participant mapping if successful) or indices_df_inferred (from inferred countries)
# Check if indices_df exists and is not empty first
if 'indices_df_inferred' in locals() and indices_df_inferred is not None and not indices_df_inferred.empty:
    indices_df_to_plot = indices_df_inferred
    print("\nUsing indices_df for plotting.")
elif 'indices_df' in locals() and indices_df is not None and not indices_df.empty:
    indices_df_to_plot = indices_df
    print("\nUsing indices_df_inferred for plotting.")
else:
    indices_df_to_plot = None
    print("\nNo country-level indices DataFrame available for plotting.")


if indices_df_to_plot is not None and not indices_df_to_plot.empty:
    # Create a copy to avoid modifying the original DataFrame
    plot_df = indices_df_to_plot.copy()

    #Fill NaN values in 'Number_of_Responses' with 0 as requested by the user
    plot_df['Number_of_Responses'] = plot_df['Number_of_Responses'].fillna(1)

    # Remove rows where the *index values* themselves are None/NaN, but keep rows with 0 responses
    indices_to_check = ['AIOI', 'EAI', 'GECI', 'DSS']
    plot_df = plot_df.dropna(subset=indices_to_check).copy()


    if plot_df.empty:
        print("\nNo complete index data available for plotting after removing NaNs in index values.")
    else:
        # Define the indices to plot
        indices_to_plot = ['AIOI', 'EAI', 'GECI', 'DSS']

        for index_name in indices_to_plot:
            # Filter out rows where the current index is None (e.g., EAI when no econ codes) - this is handled by the drop
            # Ensure the column for the current index exists, although it should if in indices_to_plot
            if index_name in plot_df.columns:

                # Create the scatter plot using Plotly Express
                fig = px.scatter(plot_df,
                                 x='Country',
                                 y=index_name,
                                 size='Number_of_Responses', # Use Number_of_Responses for marker size (now with NaNs fill
                                 color='Country',            # Color points by country
                                 hover_name='Country',
                                 hover_data={'Country': False, # Hide country from hover data as it's the hover_name
                                             index_name: ':.2f', # Format index value
                                             'Number_of_Responses': True},
                                 title=f'{index_name} by Country (Marker Size indicates Number of Responses)',
                                 labels={'Country': 'Country / Region', index_name: index_name},
                                 # Customize marker appearance
                                 size_max=30 # Adjust max size as needed
                                )

                # Update layout for better readability
                fig.update_layout(xaxis_tickangle=-45) # Rotate x-axis labels if needed
                fig.update_traces(marker=dict(line=dict(width=1, color='DarkSlateGrey'))) # Add border to markers

                # Show the plot
                fig.show()
            else:
                print(f"Warning: Index column '{index_name}' not found in plot_df. Skipping plot.")

else:
    print("No country-level indices DataFrame available to plot.")
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly) (8.5.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from plotly) (24.2)

Using indices_df for plotting.
```

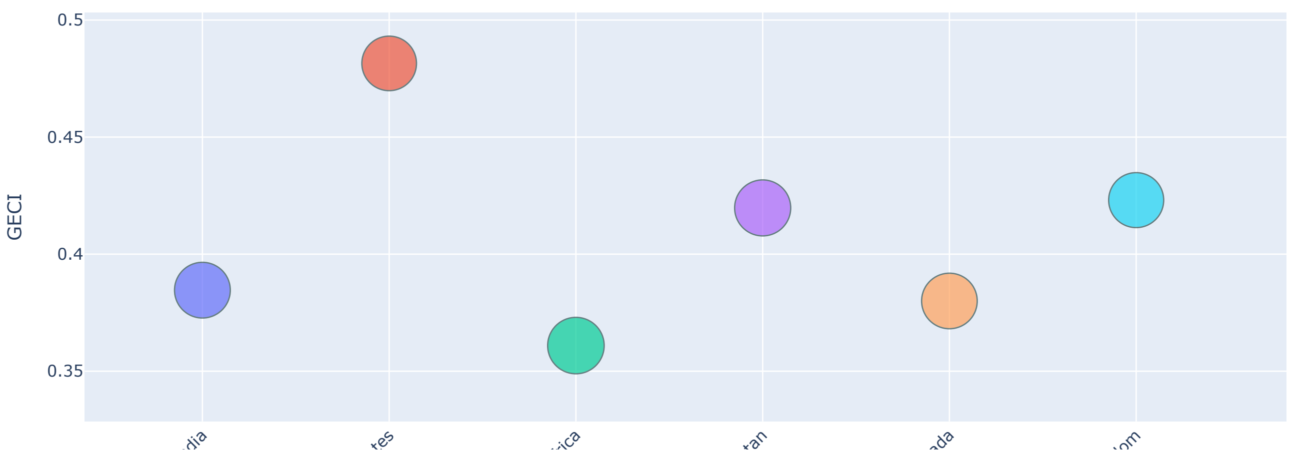
AIOI by Country (Marker Size indicates Number of Responses)

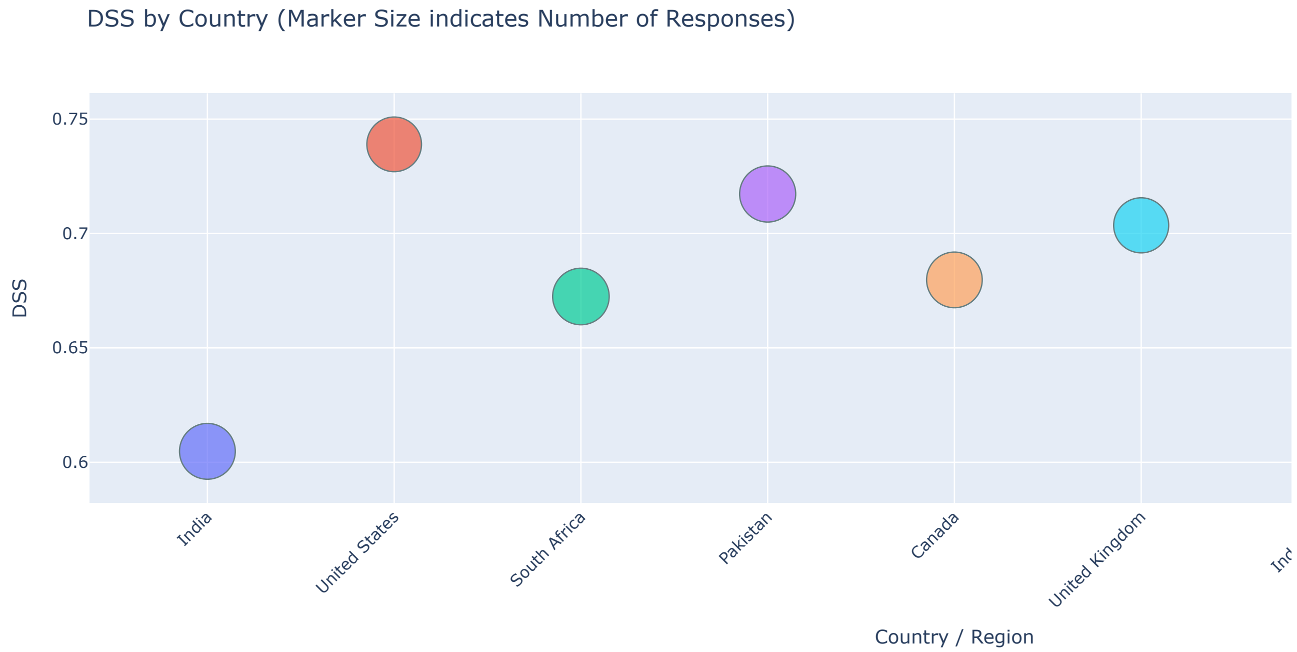


EAI by Country (Marker Size indicates Number of Responses)



GECI by Country (Marker Size indicates Number of Responses)





Double-click (or enter) to edit

```
import plotly.express as px

if indices_df is not None and not indices_df.empty:
    # Remove rows with any None or NaN values in the index columns for plotting
    plot_df = indices_df_inferred.dropna(subset=['AIOI', 'EAI', 'GECI', 'DSS', 'Number_of_Responses']).copy()

    if plot_df.empty:
        print("\nNo complete index data available for scatter plots after removing NaNs.")
    else:
        print("\nGenerating Scatter Plots of Indices:")

        # Scatter plot: DSS vs AIOI
        fig = px.scatter(plot_df, x="DSS", y="AIOI",
                         size="Number_of_Responses", # Size markers by number of responses
                         color="Country",            # Color points by country
                         hover_name="Country",
                         title="DSS vs AIOI by Country (Size represents Number of Responses)",
                         labels={"DSS": "Discourse Sophistication Score", "AIOI": "National AI Optimism Index"})
        fig.show()

        # Scatter plot: EAI vs GECI
        fig = px.scatter(plot_df, x="EAI", y="GECI",
                         size="Number_of_Responses", # Size markers by number of responses
                         color="Country",            # Color points by country
                         hover_name="Country",
                         title="EAI vs GECI by Country (Size represents Number of Responses)",
                         labels={"EAI": "Economic Anxiety Index", "GECI": "Governance & Ethics Concern Index"})
        fig.show()

        # Scatter plot: DSS vs EAI
        fig = px.scatter(plot_df, x="DSS", y="EAI",
                         size="Number_of_Responses", # Size markers by number of responses
                         color="Country",            # Color points by country
                         hover_name="Country",
                         title="DSS vs EAI by Country (Size represents Number of Responses)",
                         labels={"DSS": "Discourse Sophistication Score", "EAI": "Economic Anxiety Index"})
```