In [143…
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import re
```

In [144…
```
per_trial = pd.read_csv('/Users/Patron/Desktop/project/data/per_trial_cleane
pre_study = pd.read_csv('/Users/Patron/Desktop/project/data/pre_study_cleane
```

In [145…
```
pre_study.shape
```

Out[145…
```
(26, 9)
```

In [146…
```
per_trial.shape
```

Out[146…
```
(26, 19)
```

In [147…
```
merged = pd.merge(per_trial, pre_study, on='participant_id', how='inner')
merged.head()
```

Out[147…

	participant_id	input_type	model_order	diagnosis_type	shown_confidence_A	sho
0	aish	text	A, B	conflicting	Yes (3)	
1	akan	text	B, A	consistent	Yes (5)	
2	akas	text	A, B	consistent	No	
3	anan	text	B, A	conflicting	Yes (2)	
4	anja	image	B, A	consistent	No	

5 rows × 27 columns

In [148…
```
# Step: Define Anchoring Behavior Metric
# If model_order == A_first and preferred_model == A --> anchored
# If model_order == B_first and preferred_model == B --> anchored
merged['anchored_behavior'] = np.where(
    ((merged['model_order_encoded'] == 0) & (merged['preferred_model_encodec
    ((merged['model_order_encoded'] == 1) & (merged['preferred_model_encodec
    1, 0
)
```

In [149…
```
# Step: Define Automation Behavior Metric
# If shown_confidence_A > shown_confidence_B and preferred_model == A --> au
# If shown_confidence_B > shown_confidence_A and preferred_model == B --> au
merged['automation_behavior'] = np.where(
    ((merged['shown_confidence_A_numeric'] > merged['shown_confidence_B_nume
    ((merged['shown_confidence_B_numeric'] > merged['shown_confidence_A_nume
```

```
    1, 0
)
```

In [150…

```
def confirmation_bias(row):
    # first shown model
    first_model = 'A' if row['model_order_encoded'] == 0 else 'B'
    preferred_model = 'A' if row['preferred_model_encoded'] == 0 else 'B'

    if (row['first_response_confidence_rating'] < row['final_confidence_rati
       (row['first_response_confidence_rating'] > row['final_confidence_rati
        return 1
    else:
        return 0

merged['confirmation_behavior'] = merged.apply(confirmation_bias, axis=1)
```

In [151…

```
# Step: Aggregate Behavior Per Participant
bias_behavior_summary = merged.groupby('participant_id').agg({
    'anchored_behavior': 'mean',
    'automation_behavior': 'mean',
    'confirmation_behavior': 'mean',
    'anchoring_bias': 'first',
    'automation_bias': 'first',
    'confirmation_bias': 'first'
}).reset_index()

bias_behavior_summary
```

Out [151…

	participant_id	anchored_behavior	automation_behavior	confirmation_behavior	a
0	aish	0.0	1.0	1.0	
1	akan	1.0	0.0	0.0	
2	akas	0.0	0.0	0.0	
3	anan	0.0	0.0	1.0	
4	anja	0.0	0.0	0.0	
5	arju	1.0	0.0	0.0	
6	ashu	1.0	1.0	0.0	
7	jyot	0.0	0.0	0.0	
8	kath	0.0	0.0	1.0	
9	kesa	1.0	0.0	0.0	
10	likh	1.0	0.0	0.0	
11	madh	0.0	0.0	1.0	
12	mano	0.0	0.0	1.0	
13	moni	1.0	0.0	1.0	
14	navi	1.0	1.0	0.0	
15	nira	0.0	1.0	0.0	
16	prak	0.0	0.0	1.0	
17	ramm	1.0	1.0	0.0	
18	sah	0.0	1.0	0.0	
19	sasi	0.0	0.0	0.0	
20	shar	1.0	0.0	1.0	
21	soni	1.0	0.0	0.0	
22	srir	1.0	1.0	0.0	
23	swet	1.0	1.0	1.0	
24	vish	0.0	0.0	1.0	
25	vive	0.0	0.0	1.0	

In [152…

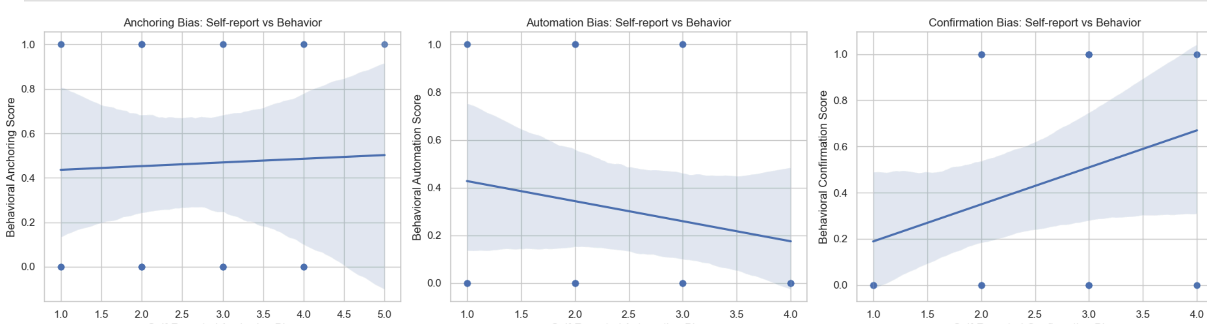
```
#Correlation analysis (Self-reported Bias vs Behavior)
correlations = {}

# Anchoring
corr_anchor, p_anchor = stats.pearsonr(bias_behavior_summary['anchoring_bias
correlations['Anchoring'] = (corr_anchor, p_anchor)

# Automation
```

```
corr_auto, p_auto = stats.pearsonr(bias_behavior_summary['automation_bias'],
correlations['Automation'] = (corr_auto, p_auto)

# Confirmation
corr_confirm, p_confirm = stats.pearsonr(bias_behavior_summary['confirmatior
correlations['Confirmation'] = (corr_confirm, p_confirm)

# Display
for bias_type, (corr, pval) in correlations.items():
    print(f"{bias_type} Correlation = {corr:.3f}, p-value = {pval:.3f}")
```

```
Anchoring Correlation = 0.037, p-value = 0.857
Automation Correlation = -0.190, p-value = 0.352
Confirmation Correlation = 0.315, p-value = 0.117
```

In [153…

```
# Scatterplots for each Bias

fig, axs = plt.subplots(1, 3, figsize=(18, 5))

# Anchoring
sns.regplot(data=bias_behavior_summary, x='anchoring_bias', y='anchored_beha
axs[0].set_title('Anchoring Bias: Self-report vs Behavior')
axs[0].set_xlabel('Self-Reported Anchoring Bias')
axs[0].set_ylabel('Behavioral Anchoring Score')

# Automation
sns.regplot(data=bias_behavior_summary, x='automation_bias', y='automation_b
axs[1].set_title('Automation Bias: Self-report vs Behavior')
axs[1].set_xlabel('Self-Reported Automation Bias')
axs[1].set_ylabel('Behavioral Automation Score')

# Confirmation
sns.regplot(data=bias_behavior_summary, x='confirmation_bias', y='confirmati
axs[2].set_title('Confirmation Bias: Self-report vs Behavior')
axs[2].set_xlabel('Self-Reported Confirmation Bias')
axs[2].set_ylabel('Behavioral Confirmation Score')

plt.tight_layout()
plt.show()
```



In [154…

```
# Clean preferred model labels
merged["preferred_model_cleaned"] = merged["preferred_model"].str.strip().st
```

In [155…

```
# Split reasoning into comment_A and comment_B using heuristic
def split_model_comments(text):
    text_lower = text.lower()
```

```
    comment_a = ""
    comment_b = ""

    # Patterns for A and B
    match_a = re.search(r"(model\s*a[^\.!?]*)", text_lower)
    match_b = re.search(r"(model\s*b[^\.!?]*)", text_lower)

    if match_a:
        comment_a = text[match_a.start():match_a.end()]
    if match_b:
        comment_b = text[match_b.start():match_b.end()]

    alt_a = re.search(r"(response\s*a[^\.!?]*)", text_lower)
    alt_b = re.search(r"(response\s*b[^\.!?]*)", text_lower)

    if not comment_a and alt_a:
        comment_a = text[alt_a.start():alt_a.end()]
    if not comment_b and alt_b:
        comment_b = text[alt_b.start():alt_b.end()]

    return pd.Series([comment_a.strip(), comment_b.strip()])

merged[["comment_A", "comment_B"]] = merged["trust_reasoning_text"].apply(sp
```

In [156…

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification,

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

# Create pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model=model, tokenizer=t

# --- Sentiment Analysis Function (RoBERTa) ---
def get_sentiment(text):
    try:
        result = sentiment_pipeline(text[:512])[0]  # RoBERTa has 512-token
        score = result["score"]

        if score > 0.55:
            label = "Positive"
        elif score < 0.47:
            label = "Negative"
        else:
            label = "Neutral"

        return score, label
    except:
        return 0.0, "Neutral"
```

```
Device set to use mps:0
```

In [157…

```
valid_themes = {
    "Confidence Signaling and Trust Calibration": "The Clinical Authority ar
    "Concise and Goal-Oriented Communication": "The Clear and Logical Explar
```

```
    "Biomedical Precision and Domain Alignment": "The Clinical Authority and
    "Excessive Reasoning or Detail": "Over-Explanation",
    "Contextual Appropriateness and Visual Grounding": "The Clear and Logica
    "Structured Reasoning and Explanation Clarity": "The Clear and Logical E
}

# Function to infer theme from trust_reasoning_text
def infer_theme(text):
    if isinstance(text, str):
        text = text.lower()
        if "medical" in text or "accurate" in text or "research" in text:
            return "Biomedical Precision and Domain Alignment"
        elif "clear" in text or "easy" in text or "step" in text or "organiz
            return "Structured Reasoning and Explanation Clarity"
        elif "short" in text or "to the point" in text or "concise" in text:
            return "Concise and Goal-Oriented Communication"
        elif "long" in text or "overwhelming" in text or "too much" in text:
            return "Excessive Reasoning or Detail"
        elif "confidence" in text or "trust" in text:
            return "Confidence Signaling and Trust Calibration"
        else:
            return "Contextual Appropriateness and Visual Grounding"
    return "Contextual Appropriateness and Visual Grounding"

# Add all theme and category columns
for col in ['themes_A', 'themes_B', 'themes_preferred', 'themes_nonpreferred
    merged[col] = merged['trust_reasoning_text'].apply(infer_theme)
    merged[f"{col}_category"] = merged[col].map(valid_themes)
```

In [158…

```
# # --- Apply sentiment + theme to A and B comments ---
merged["sentiment_score_A"], merged["sentiment_label_A"] = zip(*merged["comm
# merged["themes_A"] = merged["comment_A"].apply(extract_themes)

merged["sentiment_score_B"], merged["sentiment_label_B"] = zip(*merged["comm
# merged["themes_B"] = merged["comment_B"].apply(extract_themes)
```

In [159…

```
# --- Create preferred and non-preferred comment columns ---
merged["preferred_comment"] = merged.apply(
    lambda row: row["comment_A"] if row["preferred_model_cleaned"] == "a" el

merged["nonpreferred_comment"] = merged.apply(
    lambda row: row["comment_B"] if row["preferred_model_cleaned"] == "a" el
```

In [160…

```
# # --- Apply to preferred and non-preferred ---
merged["sentiment_score_preferred"], merged["sentiment_label_preferred"] = z
# merged["themes_preferred"] = merged["preferred_comment"].apply(extract_the

merged["sentiment_score_nonpreferred"], merged["sentiment_label_nonpreferred
# merged["themes_nonpreferred"] = merged["nonpreferred_comment"].apply(extra
```

In [161…

```
columns_to_show = [
    "preferred_model_cleaned",
    "comment_A", "sentiment_label_A", "themes_A",
    "comment_B", "sentiment_label_B", "themes_B",
    "preferred_comment", "sentiment_label_preferred", "themes_preferred",
```

```
    "nonpreferred_comment", "sentiment_label_nonpreferred", "themes_nonprefe
]

# Preview result
merged[columns_to_show].head()
```

Out[161…

	preferred_model_cleaned	comment_A	sentiment_label_A	themes_A	commen
0	b	Model A highlighted quick remedies, while Mode...	Positive	Excessive Reasoning or Detail	Moc stre long-
1	b		Negative	Confidence Signaling and Trust Calibration	mode resp impr my tru
2	b		Negative	Concise and Goal-Oriented Communication	Mode resp was s and it in
3	a		Negative	Confidence Signaling and Trust Calibration	
4	a		Negative	Contextual Appropriateness and Visual Grounding	

In [162…

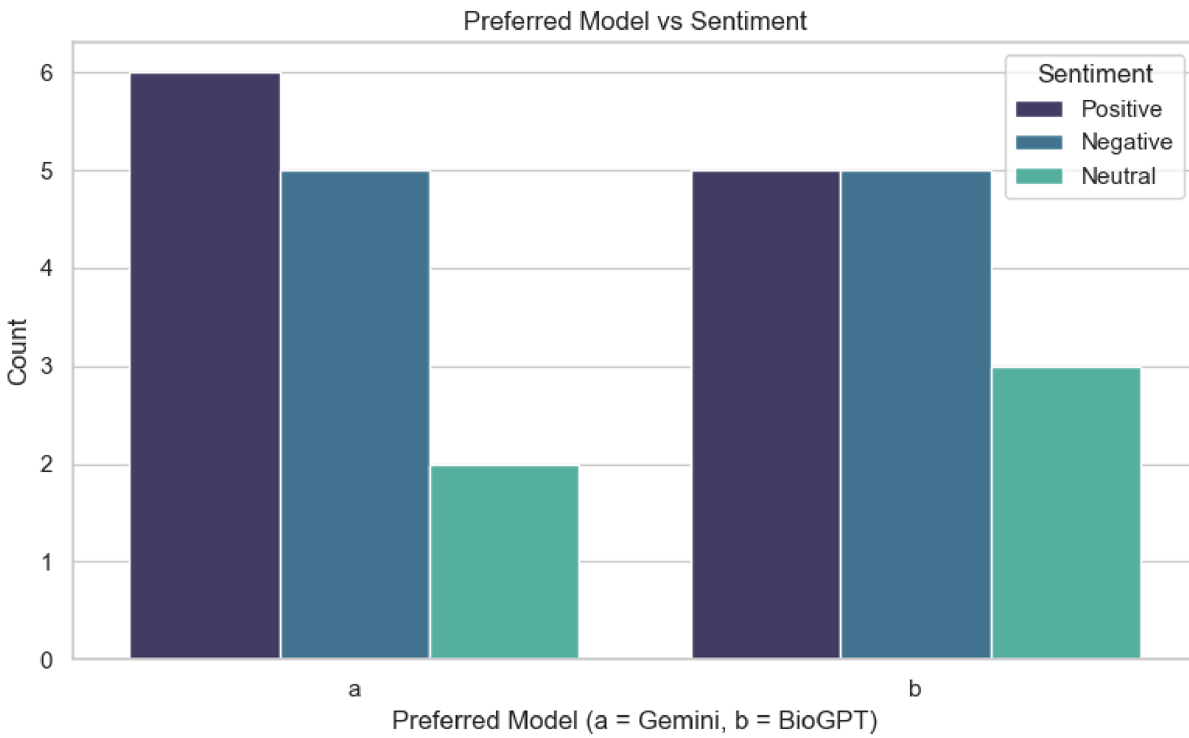
```
columns_to_show = [
    "preferred_model_cleaned",
    "comment_A", "sentiment_label_A", "themes_A",
    "comment_B", "sentiment_label_B", "themes_B",
    "preferred_comment", "sentiment_label_preferred", "themes_preferred",
    "nonpreferred_comment", "sentiment_label_nonpreferred", "themes_nonprefe
]

# Preview result
merged[columns_to_show].head()
```

Out [162…

	preferred_model_cleaned	comment_A	sentiment_label_A	themes_A	commen
0	b	Model A highlighted quick remedies, while Mode...	Positive	Excessive Reasoning or Detail	Mo stre long-
1	b		Negative	Confidence Signaling and Trust Calibration	mode resp impr my tru
2	b		Negative	Concise and Goal-Oriented Communication	Mode resp was s and it in
3	a		Negative	Confidence Signaling and Trust Calibration	
4	a		Negative	Contextual Appropriateness and Visual Grounding	

In [163…

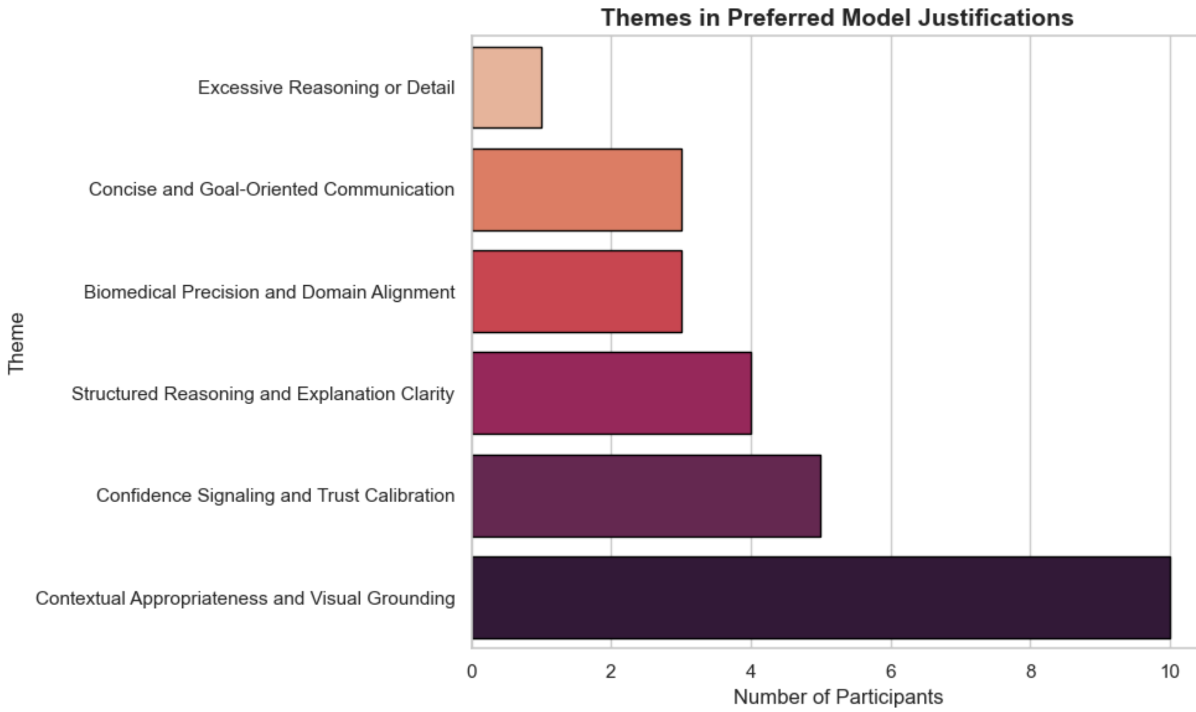
```python
# Sentiment by Preferred Model (Grouped Bar)
plt.figure(figsize=(8, 5))
sns.countplot(
    data=merged,
    x="preferred_model_cleaned",
    hue="sentiment_label_preferred",
    palette="mako",
    order=["a", "b"]  # Force 'a' (Gemini) before 'b' (BioGPT)
)
plt.title("Preferred Model vs Sentiment")
plt.xlabel("Preferred Model (a = Gemini, b = BioGPT)")
plt.ylabel("Count")
plt.legend(title="Sentiment")
plt.tight_layout()
plt.show()
```



In [164…

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

# Explode and filter out "Other"
theme_counts = (
    merged["themes_preferred"]
    .str.split(", ")
    .explode()
    .loc[lambda x: x != "Other"]
    .value_counts()
    .head(10)  # Top 10 themes
    .sort_values()
)

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

# Plot using Seaborn
plt.figure(figsize=(10, 6))
sns.barplot(
    x=theme_counts.values,
    y=theme_counts.index,
    palette="rocket_r",  # clean gradient color
    edgecolor="black"
)

plt.title("Themes in Preferred Model Justifications", fontsize=14, weight='b
plt.xlabel("Number of Participants")
plt.ylabel("Theme")
plt.tight_layout()
plt.show()
```

```
/var/folders/jn/cnxhhxhn42d2qql_g6vm7l040000gq/T/ipykernel_61439/1666803457.
py:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the
same effect.

  sns.barplot(
```



In [165…

```python
# Split reasoning into comment_A and comment_B using heuristic
def split_model_comments(text):
    text_lower = text.lower()
    comment_a = ""
    comment_b = ""

    # Patterns for A and B
    match_a = re.search(r"(model\s*a[^\.!?]*)", text_lower)
    match_b = re.search(r"(model\s*b[^\.!?]*)", text_lower)

    if match_a:
        comment_a = text[match_a.start():match_a.end()]
    if match_b:
        comment_b = text[match_b.start():match_b.end()]

    alt_a = re.search(r"(response\s*a[^\.!?]*)", text_lower)
    alt_b = re.search(r"(response\s*b[^\.!?]*)", text_lower)

    if not comment_a and alt_a:
        comment_a = text[alt_a.start():alt_a.end()]
    if not comment_b and alt_b:
        comment_b = text[alt_b.start():alt_b.end()]

    return pd.Series([comment_a.strip(), comment_b.strip()])
```

```
merged[["comment_A", "comment_B"]] = merged["trust_reasoning_text"].apply(sp
```

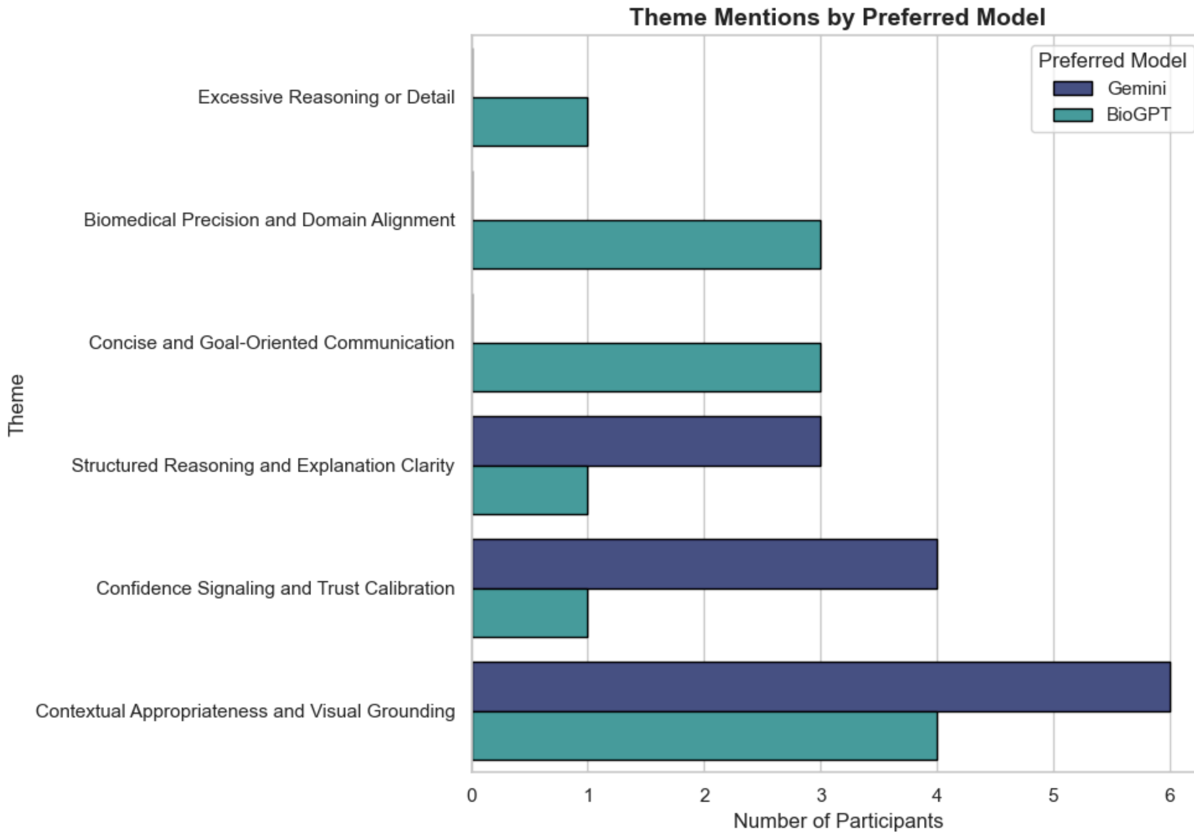
In [166…

```
theme_model = merged.copy()
theme_model["themes_preferred_split"] = theme_model["themes_preferred"].str.
exploded_theme_model = theme_model.explode("themes_preferred_split")

theme_model_ct = pd.crosstab(
    exploded_theme_model["themes_preferred_split"],
    exploded_theme_model["preferred_model_cleaned"]
).rename(columns={"a": "Gemini", "b": "BioGPT"}).sort_values(by="BioGPT", as

print(theme_model_ct)
```

```
preferred_model_cleaned                          Gemini  BioGPT
themes_preferred_split
Contextual Appropriateness and Visual Grounding       6       4
Biomedical Precision and Domain Alignment             0       3
Concise and Goal-Oriented Communication               0       3
Confidence Signaling and Trust Calibration            4       1
Excessive Reasoning or Detail                         0       1
Structured Reasoning and Explanation Clarity          3       1
```

In [167…

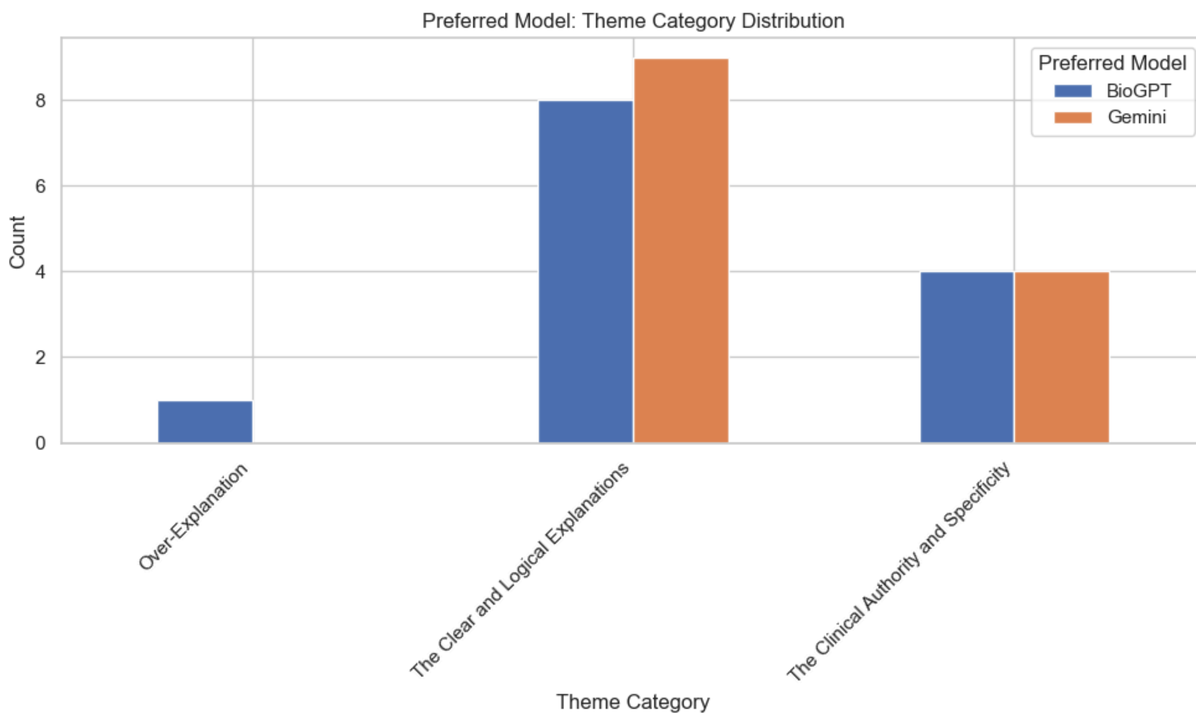
```
import seaborn as sns
import matplotlib.pyplot as plt

# Reset index to long format
theme_plot_df = theme_model_ct.reset_index().melt(
    id_vars="themes_preferred_split",
    var_name="Preferred Model",
    value_name="Count"
)

# Sort by total count (Gemini + BioGPT)
theme_plot_df["total"] = theme_plot_df.groupby("themes_preferred_split")["Co
theme_plot_df = theme_plot_df.sort_values(by="total", ascending=True)

# Plot
plt.figure(figsize=(10, 7))
sns.barplot(
    data=theme_plot_df,
    y="themes_preferred_split",
    x="Count",
    hue="Preferred Model",
    palette="mako",
    edgecolor="black"
)

plt.title("Theme Mentions by Preferred Model", fontsize=14, weight='bold')
plt.xlabel("Number of Participants")
plt.ylabel("Theme")
plt.legend(title="Preferred Model")
plt.tight_layout()
plt.show()
```



In [ ]:

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

theme_to_category = {
    "Biomedical Precision and Domain Alignment": "The Clinical Authority anc
    "Confidence Signaling and Trust Calibration": "The Clinical Authority ar
    "Structured Reasoning and Explanation Clarity": "The Clear and Logical E
    "Concise and Goal-Oriented Communication": "The Clear and Logical Explar
    "Contextual Appropriateness and Visual Grounding": "The Clear and Logica
    "Excessive Reasoning or Detail": "Over-Explanation"
}

for col in ['themes_A', 'themes_B', 'themes_preferred', 'themes_nonpreferred
    if f"{col}_category" not in merged.columns:
        merged[f"{col}_category"] = merged[col].map(theme_to_category)

# Function to find preferred model and theme category
def get_preferred_model_category(row):
    if row['model_order'] == "A, B" and row['preferred_model'] == 'A':
        return row['themes_A_category'], 'Gemini'
    elif row['model_order'] == "A, B" and row['preferred_model'] == 'B':
        return row['themes_B_category'], 'BioGPT'
    elif row['model_order'] == "B, A" and row['preferred_model'] == 'A':
        return row['themes_B_category'], 'Gemini'
    elif row['model_order'] == "B, A" and row['preferred_model'] == 'B':
        return row['themes_A_category'], 'BioGPT'
    else:
        return None, None

# Apply function and create two new columns
```

```
merged[['preferred_category', 'preferred_model_name']] = merged.apply(
    lambda row: pd.Series(get_preferred_model_category(row)), axis=1
)

# Count category frequencies for each preferred model
category_counts = merged.groupby('preferred_model_name')['preferred_category

# Plot
category_counts.T.plot(kind='bar', figsize=(10, 6))
plt.xlabel('Theme Category')
plt.ylabel('Count')
plt.title('Preferred Model: Theme Category Distribution')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.legend(title="Preferred Model")
plt.show()
```



In [169…

```
merged.to_csv("/Users/Patron/Desktop/project/data/data_merged.csv",index=Fal
```