```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

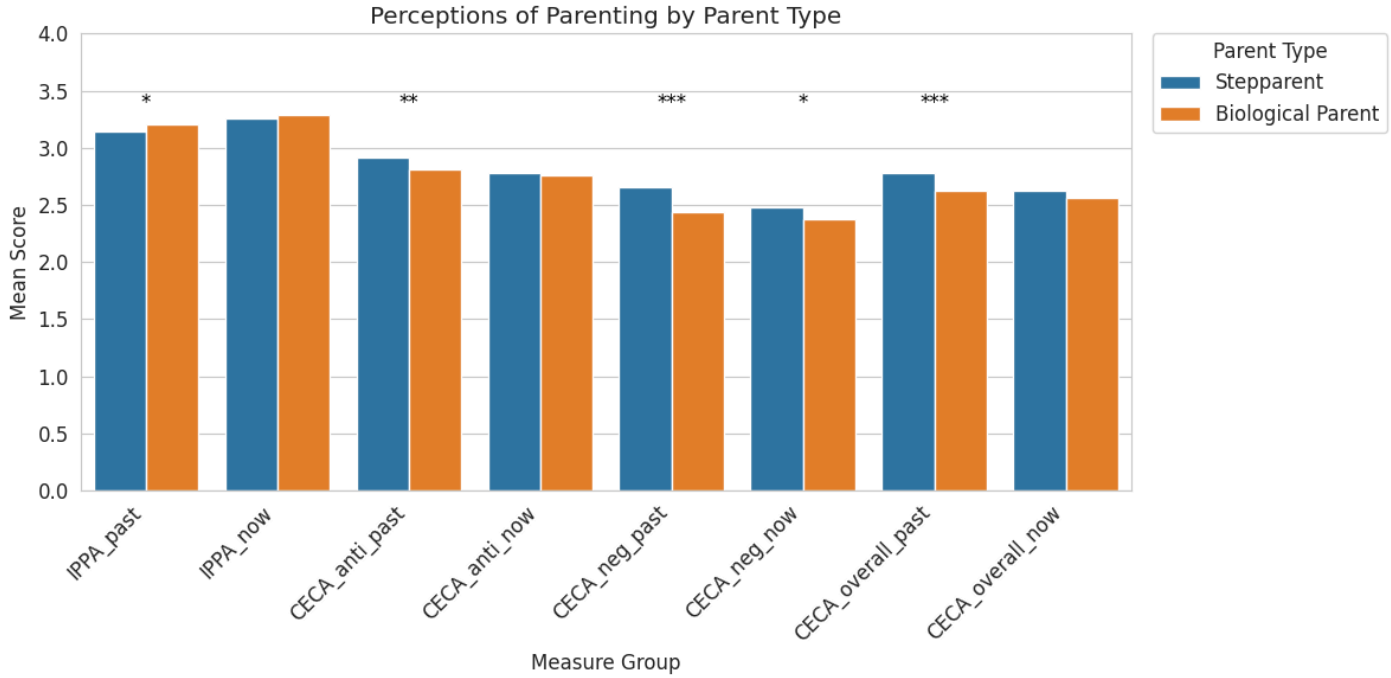
Choose Files No file chosen

```
!pip install pingouin
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
import pingouin as pg
import matplotlib as mpl
from statsmodels.stats.anova import AnovaRM
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from statsmodels.formula.api import ols

df = pd.read_csv('stepchild_0609.csv')

df.head()



print(df.head())

print(df.info())

print(df.isnull().sum())


# Define the dependent variables
dependent_vars = [
    'IPPA_step_past_M', 'IPPA_step_now_M', 'IPPA_bio_past_M', 'IPPA_bio_now_M',
    'CECA_step_past_anti_M', 'CECA_step_now_anti_M', 'CECA_bio_past_anti_M', 'CECA_bio_now_anti_M',
    'CECA_step_past_neg_M', 'CECA_step_now_neg_M', 'CECA_bio_past_neg_M', 'CECA_bio_now_neg_M',
    'CECA_step_past_over_M', 'CECA_step_now_over_M', 'CECA_bio_past_over_M', 'CECA_bio_now_over_M'
]

# Reshape the data to long format
df_long = pd.melt(df, id_vars=['phone'], value_vars=dependent_vars,
                  var_name='Condition', value_name='Score')

# Extract Time and ParentType from the Condition column
df_long['Time'] = df_long['Condition'].apply(lambda x: 'past' if 'past' in x else 'now')
df_long['ParentType'] = df_long['Condition'].apply(lambda x: 'bio' if 'bio' in x else 'step')

# Check for unique values in each group
unique_counts = df_long.groupby(['Condition', 'ParentType'])['Score'].nunique()
print("Unique value counts per group:\n", unique_counts)

# Filter out groups with less than two unique values
valid_conditions = unique_counts[unique_counts >= 2].index
filtered_df_long = df_long[df_long.set_index(['Condition', 'ParentType']).index.isin(valid_conditions)].copy()

# Verify the filtered data
print("Filtered data:\n", filtered_df_long)

# Ensure that the filtered data has at least two unique values for each group
filtered_unique_counts = filtered_df_long.groupby(['Condition', 'ParentType'])['Score'].nunique()
print("Filtered unique value counts per group:\n", filtered_unique_counts)

# Define pairs for comparison
comparison_pairs = [
    ('IPPA_step_past_M', 'IPPA_bio_past_M'),
    ('IPPA_step_now_M', 'IPPA_bio_now_M'),
    ('CECA_step_past_anti_M', 'CECA_bio_past_anti_M'),
    ('CECA_step_now_anti_M', 'CECA_bio_now_anti_M'),
    ('CECA_step_past_neg_M', 'CECA_bio_past_neg_M'),
    ('CECA_step_now_neg_M', 'CECA_bio_now_neg_M'),
    ('CECA_step_past_over_M', 'CECA_bio_past_over_M'),
    ('CECA_step_now_over_M', 'CECA_bio_now_over_M')
]

# Perform pairwise t-tests for each pair
pairwise_results = []
for step_var, bio_var in comparison_pairs:
    result = pg.ttest(df[step_var], df[bio_var], paired=True)
    result['Variable'] = f'{step_var} vs {bio_var}'
    pairwise_results.append(result)
```

```
# Combine results into a DataFrame
pairwise_results_df = pd.concat(pairwise_results, ignore_index=True)

# Output the p-values for each comparison
print(pairwise_results_df[['Variable', 'p-val']])

# Set the font to Arial
mpl.rcParams['font.sans-serif'] = ['Arial']
mpl.rcParams['font.size'] = 12

# Define a specific colorblind-friendly palette with custom colors
custom_palette = sns.color_palette([(0.121, 0.466, 0.705), (1.0, 0.498, 0.054)])  # Blue and Orange

# Set the plot style to white background
sns.set_style("whitegrid")

# Combine all bar plots into a single figure
plt.figure(figsize=(14, 10))

# Define the order of conditions for plotting
order = [
    'IPPA_step_past_M', 'IPPA_bio_past_M',
    'IPPA_step_now_M', 'IPPA_bio_now_M',
    'CECA_step_past_anti_M', 'CECA_bio_past_anti_M',
    'CECA_step_now_anti_M', 'CECA_bio_now_anti_M',
    'CECA_step_past_neg_M', 'CECA_bio_past_neg_M',
    'CECA_step_now_neg_M', 'CECA_bio_now_neg_M',
    'CECA_step_past_over_M', 'CECA_bio_past_over_M',
    'CECA_step_now_over_M', 'CECA_bio_now_over_M'
]



# 添加 MeasureGroup 列
def assign_measure_group(condition):
    if 'IPPA' in condition and 'past' in condition:
        return 'IPPA_past'
    elif 'IPPA' in condition and 'now' in condition:
        return 'IPPA_now'
    elif 'CECA' in condition and 'anti' in condition and 'past' in condition:
        return 'CECA_anti_past'
    elif 'CECA' in condition and 'anti' in condition and 'now' in condition:
        return 'CECA_anti_now'
    elif 'CECA' in condition and 'neg' in condition and 'past' in condition:
        return 'CECA_neg_past'
    elif 'CECA' in condition and 'neg' in condition and 'now' in condition:
        return 'CECA_neg_now'
    elif 'CECA' in condition and 'over' in condition and 'past' in condition:
        return 'CECA_overall_past'
    elif 'CECA' in condition and 'over' in condition and 'now' in condition:
        return 'CECA_overall_now'
    else:
        return 'Other'

filtered_df_long['MeasureGroup'] = filtered_df_long['Condition'].apply(assign_measure_group)
plot_df = filtered_df_long[filtered_df_long['MeasureGroup'] != 'Other'].copy()

# 配对 t 检验结果：step vs bio for each MeasureGroup
pairwise_results = []
for group in plot_df['MeasureGroup'].unique():
    df_group = plot_df[plot_df['MeasureGroup'] == group]
    step = df_group[df_group['ParentType'] == 'step']['Score'].reset_index(drop=True)
    bio = df_group[df_group['ParentType'] == 'bio']['Score'].reset_index(drop=True)
    if len(step) == len(bio):  # paired
        result = pg.ttest(step, bio, paired=True)
        p = result['p-val'].values[0]
        pairwise_results.append((group, p))

# 转换为字典
pval_dict = dict(pairwise_results)

# 开始绘图
plt.figure(figsize=(12, 6))
sns.barplot(data=plot_df, x='MeasureGroup', y='Score', hue='ParentType',
            ci=None, palette=custom_palette)

plt.title('Perceptions of Parenting by Parent Type')
plt.ylabel('Mean Score')
plt.xlabel('Measure Group')
plt.xticks(rotation=45, ha='right')
plt.ylim(0, 4)

# Adding significance markers
measure_groups = plot_df['MeasureGroup'].unique()
for i, group in enumerate(measure_groups):
```

```
        p = pval_dict.get(group, None)
        if p is not None:
            if p < 0.001:
                marker = '***'
            elif p < 0.01:
                marker = '**'
            elif p < 0.05:
                marker = '*'
            else:
                marker = None
            if marker:
                # Set the position for the marker
                max_height = plot_df[plot_df['MeasureGroup'] == group]['Score'].max()
                plt.text(i, 3.3, marker, ha='center', va='bottom', color='black', fontsize=12)

# 设置图例在右上角，图外
plt.legend(title='Parent Type', labels=['Stepparent', 'Biological Parent'],
           loc='upper left', bbox_to_anchor=(1.02, 1), borderaxespad=0)

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

```
<ipython-input-9-60d78ead9169>:41: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data=plot_df, x='MeasureGroup', y='Score', hue='ParentType',
```




```
# Define a function to perform repeated measures ANOVA

def reshape_data_for_anova(df, step_var, bio_var):
    df_long = pd.melt(df, id_vars=['phone'], value_vars=[step_var, bio_var],
                      var_name='ParentType', value_name='Score')
    df_long['ParentType'] = df_long['ParentType'].apply(lambda x: 'step' if 'step' in x else 'bio')
    # Ensure the 'Score' column is numeric
    df_long['Score'] = pd.to_numeric(df_long['Score'], errors='coerce')
    return df_long

def perform_repeated_measures_anova(data, step_var, bio_var):
    reshaped_data = reshape_data_for_anova(data, step_var, bio_var)
    aovrm = AnovaRM(reshaped_data, depvar='Score', subject='phone', within=['ParentType'], aggregate_func='mean')
    res = aovrm.fit()
    print(res.summary())

# IPPA Past Comparison
perform_repeated_measures_anova(df, 'IPPA_step_past_M', 'IPPA_bio_past_M')

# IPPA Now Comparison
perform_repeated_measures_anova(df, 'IPPA_step_now_M', 'IPPA_bio_now_M')

# CECA Antipathy - Past Comparison
perform_repeated_measures_anova(df, 'CECA_step_past_anti_M', 'CECA_bio_past_anti_M')

# CECA Antipathy - Now Comparison
```

```
perform_repeated_measures_anova(df, 'CECA_step_now_anti_M', 'CECA_bio_now_anti_M')

# CECA Neglect - Past Comparison
perform_repeated_measures_anova(df, 'CECA_step_past_neg_M', 'CECA_bio_past_neg_M')

# CECA Neglect - Now Comparison
perform_repeated_measures_anova(df, 'CECA_step_now_neg_M', 'CECA_bio_now_neg_M')

# CECA Overprotection - Past Comparison
perform_repeated_measures_anova(df, 'CECA_step_past_over_M', 'CECA_bio_past_over_M')

# CECA Overprotection - Now Comparison
perform_repeated_measures_anova(df, 'CECA_step_now_over_M', 'CECA_bio_now_over_M')

# Sexual Abuse Comparison
perform_repeated_measures_anova(df, 'sexualAbuse_step', 'sexualAbuse_bio')

# Physical Abuse Comparison
perform_repeated_measures_anova(df, 'physicalAbuse_step', 'physicalAbuse_bio')

# Prepare data for H7a, H7b, and H7c

# H7a: Stepparents vs. Biological Parents
# Compare sexualAbuse and abuse between stepparents and biological parents
sexual_abuse_comparison = df[['sexualAbuse_step', 'sexualAbuse_bio']].copy()
physical_abuse_comparison = df[['physicalAbuse_step', 'physicalAbuse_bio']].copy()

# H7b: Stepfathers vs. Stepmothers
# Filter data for stepparents and compare based on male_gender
stepfather_data = df[df['male_gender'] == 1][['sexualAbuse_step', 'physicalAbuse_step']].copy()
stepmother_data = df[df['male_gender'] == 0][['sexualAbuse_step', 'physicalAbuse_step']].copy()

# H7c: Biofathers vs. Biomothers
# Filter data for biological parents and compare based on biologicalparent_male
biofather_data = df[df['biologicalparent_male'] == 1][['sexualAbuse_bio', 'physicalAbuse_bio']].copy()
biomother_data = df[df['biologicalparent_male'] == 0][['sexualAbuse_bio', 'physicalAbuse_bio']].copy()

# Output the prepared data for verification
print("Sexual Abuse Comparison (Stepparents vs Biological Parents):\n", sexual_abuse_comparison.head())
print("Physical Abuse Comparison (Stepparents vs Biological Parents):\n", physical_abuse_comparison.head())
print("Stepfather Data:\n", stepfather_data.head())
print("Stepmother Data:\n", stepmother_data.head())
print("Biofather Data:\n", biofather_data.head())
print("Biomother Data:\n", biomother_data.head())
```

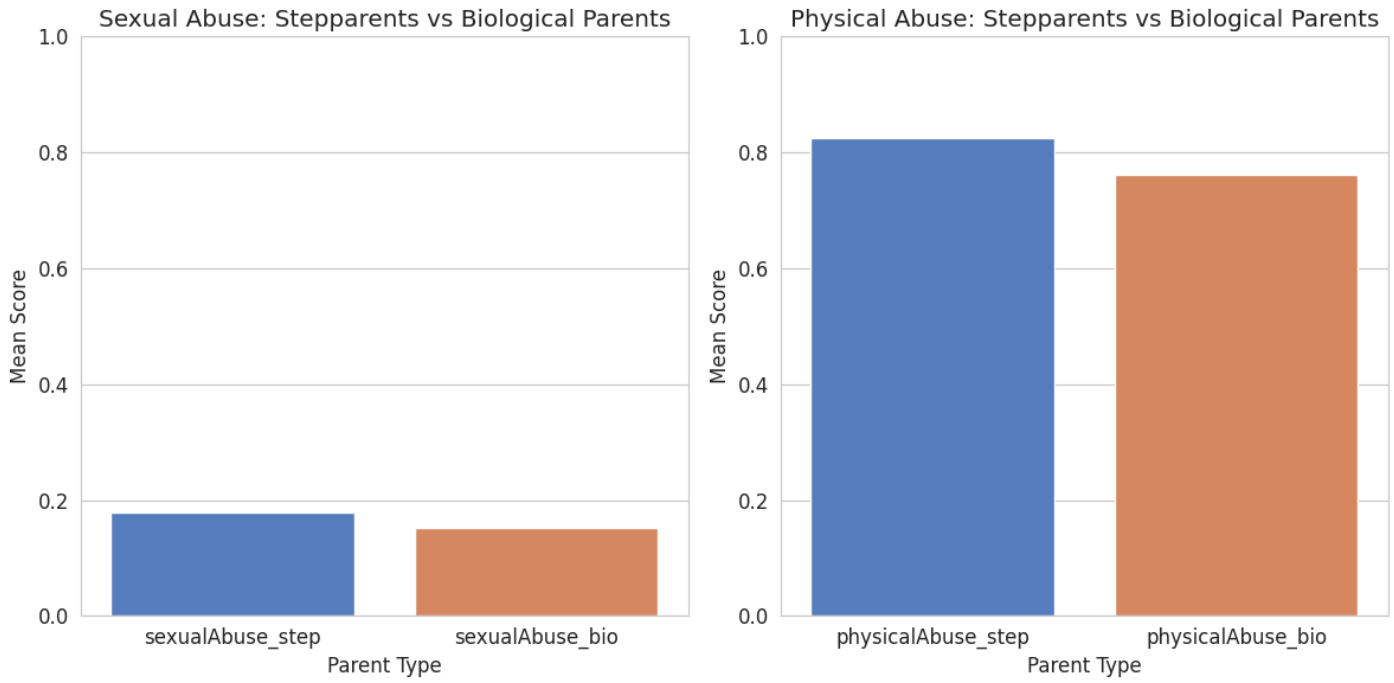
```
                  Anova
========================================
           F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType  4.2206 1.0000 222.0000 0.0411
========================================

                  Anova
========================================
           F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType  0.8700 1.0000 222.0000 0.3520
========================================

                  Anova
========================================
           F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType  8.1890 1.0000 222.0000 0.0046
========================================

                  Anova
========================================
           F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType  0.5970 1.0000 222.0000 0.4406
========================================

                  Anova
========================================
           F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType 13.6405 1.0000 222.0000 0.0003
========================================

                  Anova
========================================
           F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType  4.2479 1.0000 222.0000 0.0405
========================================

                  Anova
========================================
```

```
            F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType 13.6106 1.0000 222.0000 0.0003
========================================

                  Anova
========================================
            F Value Num DF  Den DF  Pr > F
----------------------------------------
ParentType  3.0218 1.0000 222.0000 0.0835
========================================

                  Anova
```

```
# H7a Analysis

# Convert abuse_step and abuse_bio to numeric, coercing errors to NaN
physical_abuse_comparison['physicalAbuse_step'] = pd.to_numeric(physical_abuse_comparison['physicalAbuse_step'], errors='coerce')
physical_abuse_comparison['physicalAbuse_bio'] = pd.to_numeric(physical_abuse_comparison['physicalAbuse_bio'], errors='coerce')

# Paired t-test for physical abuse
paired_ttest_result = pg.ttest(physical_abuse_comparison['physicalAbuse_step'], physical_abuse_comparison['physicalAbuse_bio'], paired=True)
print("Paired t-test for Physical Abuse (Stepparents vs Biological Parents):")
print(paired_ttest_result[['T', 'p-val']])

# H7b Analysis: Stepfathers vs. Stepmothers
# Convert abuse_step to numeric in stepfather_data and stepmother_data, coercing errors to NaN
stepfather_data['physicalAbuse_step'] = pd.to_numeric(stepfather_data['physicalAbuse_step'], errors='coerce')
stepmother_data['physicalAbuse_step'] = pd.to_numeric(stepmother_data['physicalAbuse_step'], errors='coerce')

# Convert abuse_bio to numeric in biofather_data and biomother_data, coercing errors to NaN
biofather_data['physicalAbuse_bio'] = pd.to_numeric(biofather_data['physicalAbuse_bio'], errors='coerce')
biomother_data['physicalAbuse_bio'] = pd.to_numeric(biomother_data['physicalAbuse_bio'], errors='coerce')

# Independent samples t-test for sexual abuse
step_sexual_ttest_result = pg.ttest(stepfather_data['sexualAbuse_step'], stepmother_data['sexualAbuse_step'], paired=False)
print("Independent Samples t-test for Sexual Abuse (Stepfathers vs Stepmothers):")
print(step_sexual_ttest_result[['T', 'p-val']])

# Independent samples t-test for physical abuse
step_physical_ttest_result = pg.ttest(stepfather_data['physicalAbuse_step'], stepmother_data['physicalAbuse_step'], paired=False)
print("Independent Samples t-test for Physical Abuse (Stepfathers vs Stepmothers):")
print(step_physical_ttest_result[['T', 'p-val']])

# H7c Analysis: Biofathers vs. Biomothers
# Independent samples t-test for sexual abuse
bio_sexual_ttest_result = pg.ttest(biofather_data['sexualAbuse_bio'], biomother_data['sexualAbuse_bio'], paired=False)
print("Independent Samples t-test for Sexual Abuse (Biofathers vs Biomothers):")
print(bio_sexual_ttest_result[['T', 'p-val']])

# Independent samples t-test for physical abuse
bio_physical_ttest_result = pg.ttest(biofather_data['physicalAbuse_bio'], biomother_data['physicalAbuse_bio'], paired=False)
print("Independent Samples t-test for Physical Abuse (Biofathers vs Biomothers):")
print(bio_physical_ttest_result[['T', 'p-val']])

# H7a Visualization: Stepparents vs. Biological Parents
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.barplot(data=sexual_abuse_comparison.melt(), x='variable', y='value', ci=None, palette='muted')
plt.title('Sexual Abuse: Stepparents vs Biological Parents')
plt.ylabel('Mean Score')
plt.xlabel('Parent Type')
plt.ylim(0, 1)  # Set y-axis limit
# Add significance marker using paired t-test for physical abuse
if paired_ttest_result['p-val'].values[0] < 0.001:
    plt.text(0.5, 0.9, '***', ha='center', va='bottom', color='black')
elif paired_ttest_result['p-val'].values[0] < 0.01:
    plt.text(0.5, 0.9, '**', ha='center', va='bottom', color='black')
elif paired_ttest_result['p-val'].values[0] < 0.05:
    plt.text(0.5, 0.9, '*', ha='center', va='bottom', color='black')

plt.subplot(1, 2, 2)
sns.barplot(data=physical_abuse_comparison.melt(), x='variable', y='value', ci=None, palette='muted')
plt.title('Physical Abuse: Stepparents vs Biological Parents')
plt.ylabel('Mean Score')
plt.xlabel('Parent Type')
plt.ylim(0, 1)  # Set y-axis limit
# Add significance marker using paired t-test for physical abuse
if paired_ttest_result['p-val'].values[0] < 0.001:
    plt.text(0.5, 0.9, '***', ha='center', va='bottom', color='black')
elif paired_ttest_result['p-val'].values[0] < 0.01:
    plt.text(0.5, 0.9, '**', ha='center', va='bottom', color='black')
elif paired_ttest_result['p-val'].values[0] < 0.05:
    plt.text(0.5, 0.9, '*', ha='center', va='bottom', color='black')
```

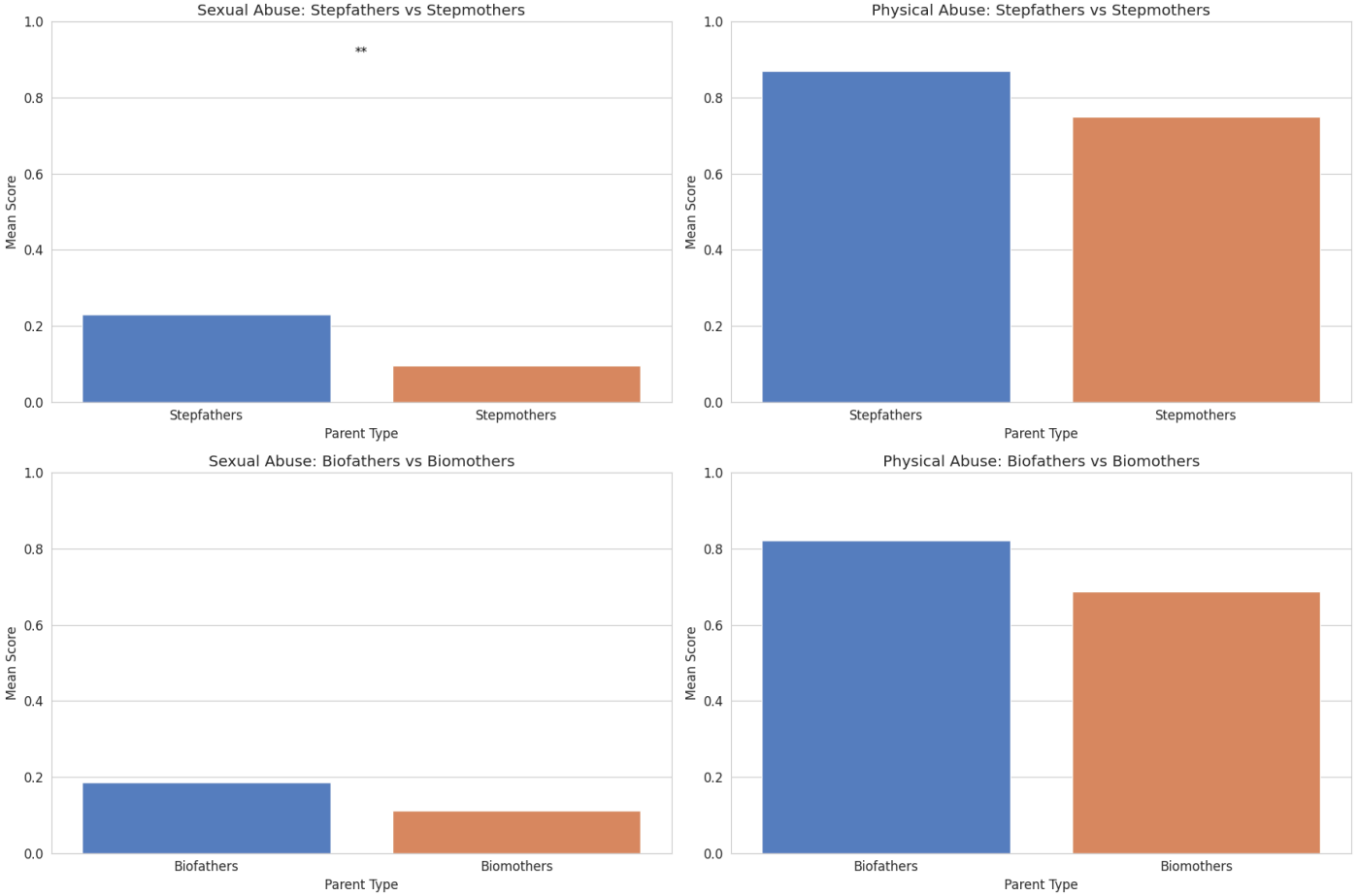
```
plt.tight_layout()
plt.show()

# H7b and H7c Visualization: Stepfathers vs. Stepmothers and Biofathers vs. Biomothers
plt.figure(figsize=(18, 12))

# H7b: Stepfathers vs. Stepmothers
plt.subplot(2, 2, 1)
sns.barplot(x=['Stepfathers', 'Stepmothers'], y=[stepfather_data['sexualAbuse_step'].mean(), stepmother_data['sexualAbuse_step'].mean()], ci=None, palette='
plt.title('Sexual Abuse: Stepfathers vs Stepmothers')
plt.ylabel('Mean Score')
plt.xlabel('Parent Type')
plt.ylim(0, 1)  # Set y-axis limit
# Add significance marker
if step_sexual_ttest_result['p-val'].values[0] < 0.001:
    plt.text(0.5, 0.9, '***', ha='center', va='bottom', color='black')
elif step_sexual_ttest_result['p-val'].values[0] < 0.01:
    plt.text(0.5, 0.9, '**', ha='center', va='bottom', color='black')
elif step_sexual_ttest_result['p-val'].values[0] < 0.05:
    plt.text(0.5, 0.9, '*', ha='center', va='bottom', color='black')

plt.subplot(2, 2, 2)
sns.barplot(x=['Stepfathers', 'Stepmothers'], y=[stepfather_data['physicalAbuse_step'].mean(), stepmother_data['physicalAbuse_step'].mean()], ci=None, palet
plt.title('Physical Abuse: Stepfathers vs Stepmothers')
plt.ylabel('Mean Score')
plt.xlabel('Parent Type')
plt.ylim(0, 1)  # Set y-axis limit
# Add significance marker
if step_physical_ttest_result['p-val'].values[0] < 0.001:
    plt.text(0.5, 0.9, '***', ha='center', va='bottom', color='black')
elif step_physical_ttest_result['p-val'].values[0] < 0.01:
    plt.text(0.5, 0.9, '**', ha='center', va='bottom', color='black')
elif step_physical_ttest_result['p-val'].values[0] < 0.05:
    plt.text(0.5, 0.9, '*', ha='center', va='bottom', color='black')

# H7c: Biofathers vs. Biomothers
plt.subplot(2, 2, 3)
sns.barplot(x=['Biofathers', 'Biomothers'], y=[biofather_data['sexualAbuse_bio'].mean(), biomother_data['sexualAbuse_bio'].mean()], ci=None, palette='muted'
plt.title('Sexual Abuse: Biofathers vs Biomothers')
plt.ylabel('Mean Score')
plt.xlabel('Parent Type')
plt.ylim(0, 1)  # Set y-axis limit
# Add significance marker
if bio_sexual_ttest_result['p-val'].values[0] < 0.001:
    plt.text(0.5, 0.9, '***', ha='center', va='bottom', color='black')
elif bio_sexual_ttest_result['p-val'].values[0] < 0.01:
    plt.text(0.5, 0.9, '**', ha='center', va='bottom', color='black')
elif bio_sexual_ttest_result['p-val'].values[0] < 0.05:
    plt.text(0.5, 0.9, '*', ha='center', va='bottom', color='black')

plt.subplot(2, 2, 4)
sns.barplot(x=['Biofathers', 'Biomothers'], y=[biofather_data['physicalAbuse_bio'].mean(), biomother_data['physicalAbuse_bio'].mean()], ci=None, palette='mu
plt.title('Physical Abuse: Biofathers vs Biomothers')
plt.ylabel('Mean Score')
plt.xlabel('Parent Type')
plt.ylim(0, 1)  # Set y-axis limit
# Add significance marker
if bio_physical_ttest_result['p-val'].values[0] < 0.001:
    plt.text(0.5, 0.9, '***', ha='center', va='bottom', color='black')
elif bio_physical_ttest_result['p-val'].values[0] < 0.01:
    plt.text(0.5, 0.9, '**', ha='center', va='bottom', color='black')
elif bio_physical_ttest_result['p-val'].values[0] < 0.05:
    plt.text(0.5, 0.9, '*', ha='center', va='bottom', color='black')

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

```
Paired t-test for Physical Abuse (Stepparents vs Biological Parents):
                T     p-val
T-test  0.961361  0.337417
Independent Samples t-test for Sexual Abuse (Stepfathers vs Stepmothers):
                T     p-val
T-test  2.800908  0.005558
Independent Samples t-test for Physical Abuse (Stepfathers vs Stepmothers):
                T     p-val
T-test  0.597317  0.551037
Independent Samples t-test for Sexual Abuse (Biofathers vs Biomothers):
                T   p-val
T-test  1.572781  0.1172
Independent Samples t-test for Physical Abuse (Biofathers vs Biomothers):
                T     p-val
T-test  0.730222  0.466051
<ipython-input-14-b458d99fd071>:46: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data=sexual_abuse_comparison.melt(), x='variable', y='value', ci=None, palette='muted')
<ipython-input-14-b458d99fd071>:46: FutureWarning:

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

  sns.barplot(data=sexual_abuse_comparison.melt(), x='variable', y='value', ci=None, palette='muted')
<ipython-input-14-b458d99fd071>:60: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data=physical_abuse_comparison.melt(), x='variable', y='value', ci=None, palette='muted')
<ipython-input-14-b458d99fd071>:60: FutureWarning:

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

  sns.barplot(data=physical_abuse_comparison.melt(), x='variable', y='value', ci=None, palette='muted')
```




```
<ipython-input-14-b458d99fd071>:81: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x=['Stepfathers', 'Stepmothers'], y=[stepfather_data['sexualAbuse_step'].mean(), stepmother_data['sexualAbuse_step'].mean()], ci=None, pal
<ipython-input-14-b458d99fd071>:81: FutureWarning:

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

  sns.barplot(x=['Stepfathers', 'Stepmothers'], y=[stepfather_data['sexualAbuse_step'].mean(), stepmother_data['sexualAbuse_step'].mean()], ci=None, pal
<ipython-input-14-b458d99fd071>:95: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x=['Stepfathers', 'Stepmothers'], y=[stepfather_data['physicalAbuse_step'].mean(), stepmother_data['physicalAbuse_step'].mean()], ci=None,
<ipython-input-14-b458d99fd071>:95: FutureWarning:

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

  sns.barplot(x=['Stepfathers', 'Stepmothers'], y=[stepfather_data['physicalAbuse_step'].mean(), stepmother_data['physicalAbuse_step'].mean()], ci=None,
<ipython-input-14-b458d99fd071>:110: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x=['Biofathers', 'Biomothers'], y=[biofather_data['sexualAbuse_bio'].mean(), biomother_data['sexualAbuse_bio'].mean()], ci=None, palette='
<ipython-input-14-b458d99fd071>:110: FutureWarning:
```

```
<ipython-input-14-b458d99fd071>:118: FutureWarning:

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

  sns.barplot(x=['Biofathers', 'Biomothers'], y=[biofather_data['sexualAbuse_bio'].mean(), biomother_data['sexualAbuse_bio'].mean()], ci=None, palette='
<ipython-input-14-b458d99fd071>:124: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x=['Biofathers', 'Biomothers'], y=[biofather_data['physicalAbuse_bio'].mean(), biomother_data['physicalAbuse_bio'].mean()], ci=None, palet
<ipython-input-14-b458d99fd071>:124: FutureWarning:

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

  sns.barplot(x=['Biofathers', 'Biomothers'], y=[biofather_data['physicalAbuse_bio'].mean(), biomother_data['physicalAbuse_bio'].mean()], ci=None, palet
```



Start coding or generate with AI.