run_csi

December 4, 2023

1 Part 1

[ ]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

# Load the dataset
file_path = 'csv/laughter-corpus.csv'  # Replace with your file path
data = pd.read_csv(file_path)

# Display the first few rows for a quick overview
data.head()
```

[ ]:

	Gender	Role	Duration
0	Female	Caller	0.961
1	Male	Receiver	0.630
2	Female	Caller	1.268
3	Male	Receiver	0.146
4	Female	Caller	0.276

[ ]:
```python
data.head(20)
```

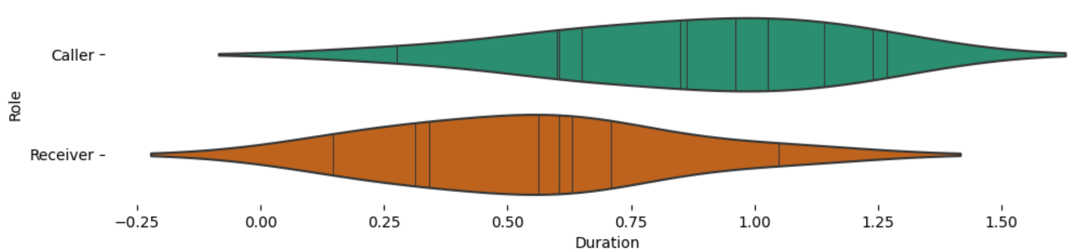
[ ]:

	Gender	Role	Duration
0	Female	Caller	0.961
1	Male	Receiver	0.630
2	Female	Caller	1.268
3	Male	Receiver	0.146
4	Female	Caller	0.276
5	Male	Receiver	0.562
6	Female	Caller	1.141
7	Female	Caller	0.600
8	Female	Caller	1.239
9	Female	Caller	0.850
10	Male	Receiver	0.605
11	Female	Caller	1.026

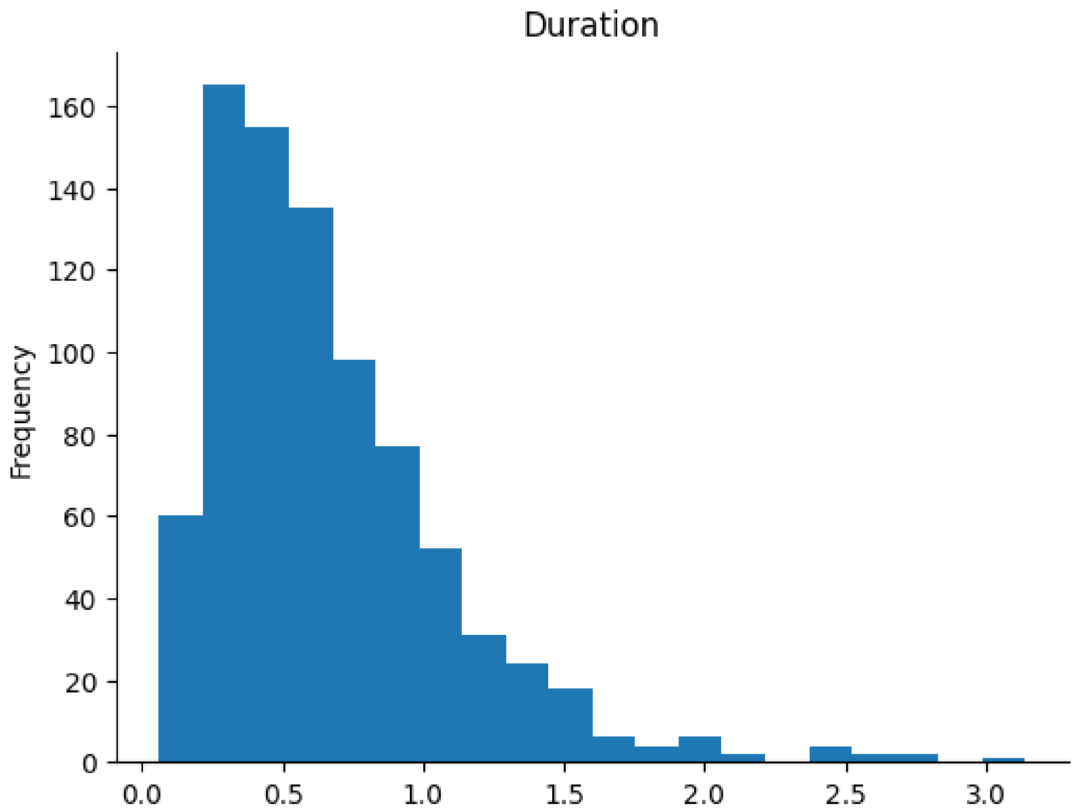
```
12    Male  Receiver     0.314
13  Female    Caller     1.026
14  Female    Caller     0.605
15    Male  Receiver     0.710
16  Female    Caller     0.862
17    Male  Receiver     0.341
18  Female    Caller     0.651
19    Male  Receiver     1.048
```

[ ]:
```python
figsize = (12, 1.2 * len(data['Role'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(data.head(20), x='Duration', y='Role', inner='stick',␣
 ↪palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

```
/home/alex/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/home/alex/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/home/alex/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



[ ]:
```python
data['Duration'].plot(kind='hist', bins=20, title='Duration')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



1.1 Chi-Square Test for Count Data (Questions 1 and 2)

1. Question 1: Number of Laughter Events - Women vs. Men Is the number of laughter events higher for women than for men? H1: Women have a higher number of laughter events than men. H0: There is no significant difference in the number of laughter events between women and men.

- Observed Frequencies: Counts of laughter events for each gender.
- Expected Frequencies: Assuming no gender difference, we expect the counts to be proportional to the number of speakers of each gender.

2. Question 2: Number of Laughter Events - Callers vs. Receivers Is the number of laughter events higher for callers than for receivers? H1: Callers have a higher number of laughter events than receivers. H0: There is no significant difference in the number of laughter events between callers and receivers.

- Observed Frequencies: Counts of laughter events for each role.
- Expected Frequencies: Assuming no role difference, we expect the counts to be equal for callers and receivers.

[ ]:
```
# Count the number of laughter events for each gender and role
gender_counts = data['Gender'].value_counts()
role_counts = data['Role'].value_counts()
```

[ ]:
```python
# Occurance of gender
pd.DataFrame(gender_counts)
```

[ ]:
```
        count
Gender
Female    496
Male      346
```

[ ]:
```python
# Occurance of role
pd.DataFrame(role_counts)
```

[ ]:
```
          count
Role
Caller      505
Receiver    337
```

[ ]:
```python
# Creating contingency tables for Chi-Square tests
gender_cross_tab = pd.DataFrame({
    'Count': gender_counts, # Count Occurances
    'Total': [63, 57] # Total number of female and male speakers (in this order)
})
role_cross_tab = pd.DataFrame({
    'Count': role_counts, # Count Occurances
    'Total': [60, 60] # Total number of callers and receivers (in this order)
})
total = gender_cross_tab['Count'].values.sum()
total_pop = gender_cross_tab['Total'].values.sum()
```

[ ]:
```python
gender_cross_tab
```

[ ]:
```
        Count  Total
Gender
Female    496     63
Male      346     57
```

[ ]:
```python
role_cross_tab
```

[ ]:
```
          Count  Total
Role
Caller      505     60
Receiver    337     60
```

[ ]:
```python
# Function to calculate Chi-Square statistic
def chi_square_calc(observed, expected):
    return np.sum((observed - expected) ** 2 / expected)

# Function to calculate degrees of freedom for Chi-Square test
```

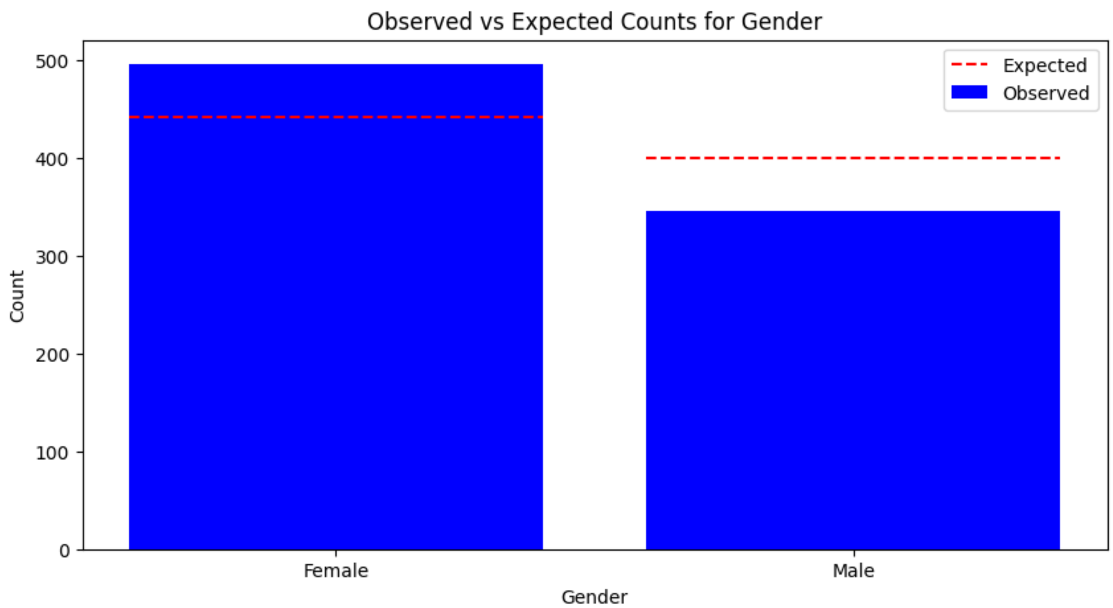
```
def chi_square_dof(observed):
    return (observed.shape[0] - 1) * (observed.shape[1] - 1)
```

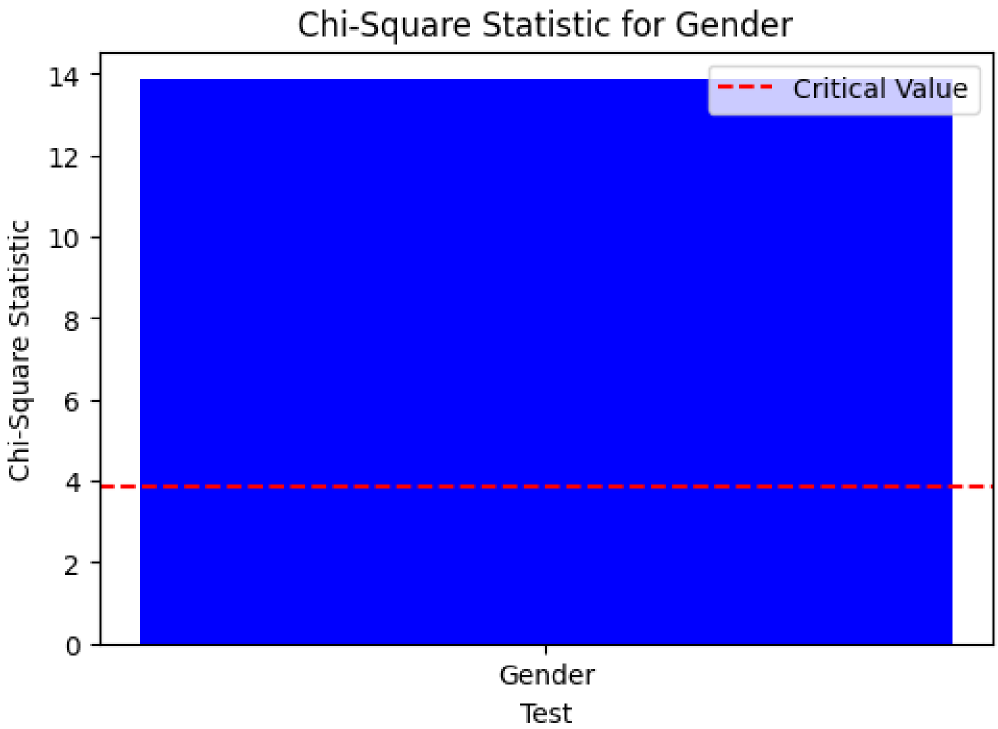
[ ]:
```
# Manual calculation of the Chi-Square statistic for gender

# Expected counts assuming no gender difference
expected_counts_gender = [gender_cross_tab['Total'].iloc[0] / total_pop * total,
                          gender_cross_tab['Total'].iloc[1] / total_pop * total,
                          ]

# Create a dataframe with gender counts and expected counts gender
gender_counts_df = pd.DataFrame({
    'Occurrences': gender_counts,
    'Expected': expected_counts_gender
})
print(gender_counts_df)

# Chi-Square statistic calculation
chi2_gender = chi_square_calc(gender_counts, expected_counts_gender)
chi2_dof_gender = chi_square_dof(gender_cross_tab)
print(f"\nQ1:Chi-Square Statistic = {chi2_gender}\nQ1:Degrees of Freedom =␣
↪{chi2_dof_gender}")
```

```
        Occurrences  Expected
Gender
Female          496    442.05
Male            346    399.95

Q1:Chi-Square Statistic = 13.861744622839753
Q1:Degrees of Freedom = 1
```

[ ]:
```
# Plotting the observed and expected counts for gender
plt.figure(figsize=(10, 5))
observed_bars = plt.bar(gender_counts_df.index,␣
↪gender_counts_df['Occurrences'], label='Observed', color='blue')
bar_width = observed_bars[0].get_width()
x_positions = [bar.get_x() + bar_width/2 for bar in observed_bars]
plt.hlines(y=gender_counts_df['Expected'].iloc[0], xmin=x_positions[0] -␣
↪bar_width/2, xmax=x_positions[0] + bar_width/2, color='red',␣
↪linestyles='--', label='Expected')
plt.hlines(y=gender_counts_df['Expected'].iloc[1], xmin=x_positions[1] -␣
↪bar_width/2, xmax=x_positions[1] + bar_width/2, color='red', linestyles='--')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Observed vs Expected Counts for Gender')
plt.legend()
plt.show()
```

```python
# Plotting the Chi-Square statistic for gender
plt.figure(figsize=(6, 4))
plt.bar(['Gender'], [chi2_gender], color='blue')
plt.axhline(y=3.84, color='red', linestyle='--', label='Critical Value')
plt.xlabel('Test')
plt.ylabel('Chi-Square Statistic')
plt.title('Chi-Square Statistic for Gender')
plt.legend()
plt.show()
```





Since the Chi-Square Statistic is 13.861744622839753 at 1 degree of freedom is less than the common alpha level of 0.05 at 3.84, we can reject the null hypothesis (H0). This suggests that there is a statistically significant difference in the number of laughter events between women and men, with women having more laughter events.

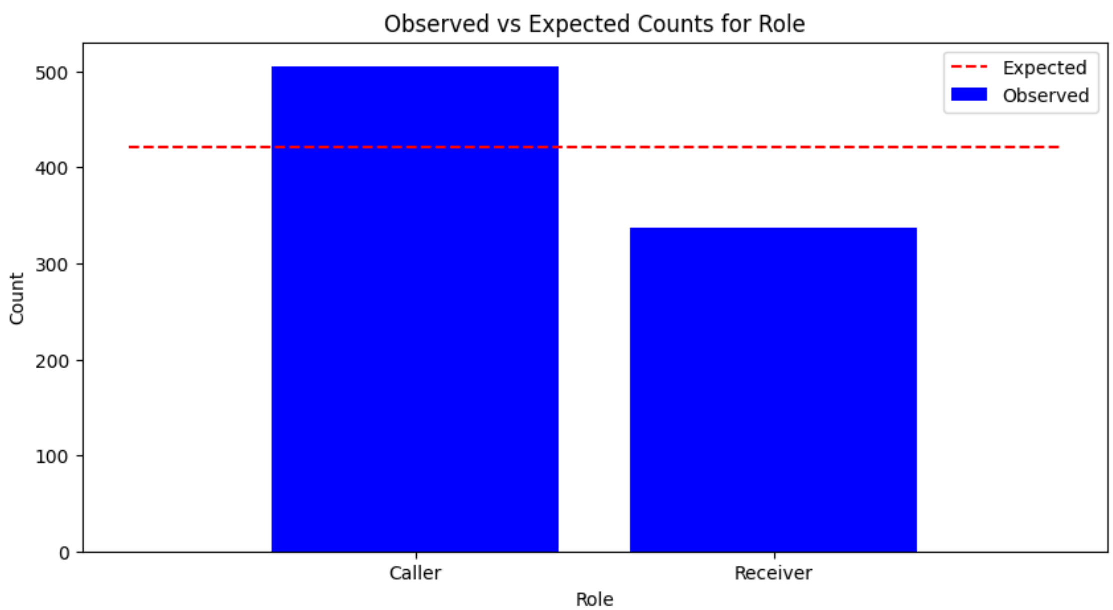
[ ]:
```
# Manual calculation of the Chi-Square statistic for role

# Expected counts assuming no role difference
expected_counts_role = [role_cross_tab['Total'].iloc[0] / total_pop * total,
                        role_cross_tab['Total'].iloc[1] / total_pop * total,
                        ]

# Create a dataframe with role counts and expected counts role
role_counts_df = pd.DataFrame({
    'Occurrences': role_counts,
    'Expected': expected_counts_role
})
print(role_counts_df)

# Chi-Square statistic calculation
chi2_role = chi_square_calc(role_counts, expected_counts_role)
chi2_dof_role = chi_square_dof(role_cross_tab)
```

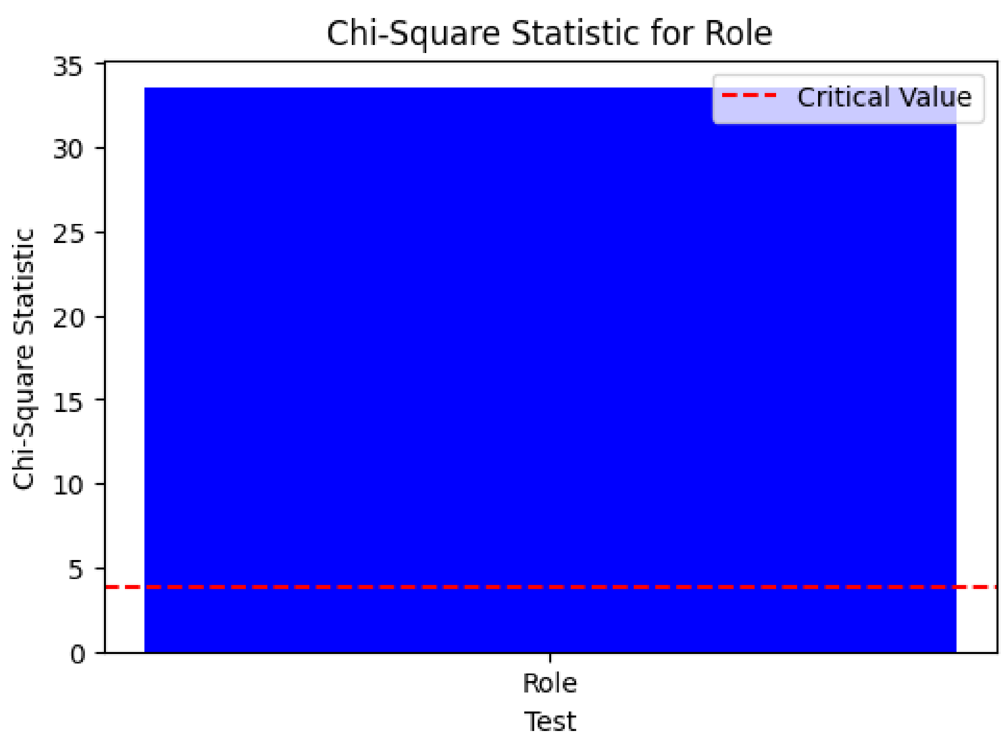
```
# Display the Chi-Square statistic for gender
print(f"\nQ1:Chi-Square Statistic = {chi2_role}\nQ1:Degrees of Freedom =
 ↪{chi2_dof_role}")
```

```
          Occurrences  Expected
Role
Caller            505     421.0
Receiver          337     421.0

Q1:Chi-Square Statistic = 33.52019002375297
Q1:Degrees of Freedom = 1
```

[ ]:

```
# Plotting the observed and expected counts for role
plt.figure(figsize=(10, 5))
observed_bars = plt.bar(role_counts_df.index, role_counts_df['Occurrences'],
 ↪label='Observed', color='blue')
bar_width = observed_bars[0].get_width()
x_positions = [bar.get_x() + bar_width/2 for bar in observed_bars]
plt.hlines(y=role_counts_df['Expected'].iloc[0], xmin=x_positions[0] -
 ↪bar_width, xmax=x_positions[-1] + bar_width, color='red', label='Expected',
 ↪linestyles='--')
plt.xlabel('Role')
plt.ylabel('Count')
plt.title('Observed vs Expected Counts for Role')
plt.legend()
plt.show()

# Plotting the Chi-Square statistic for role
plt.figure(figsize=(6, 4))
plt.bar(['Role'], [chi2_role], color='blue')
plt.axhline(y=3.84, color='red', linestyle='--', label='Critical Value')
plt.xlabel('Test')
plt.ylabel('Chi-Square Statistic')
plt.title('Chi-Square Statistic for Role')
plt.legend()
plt.show()
```





The Chi-Square Statistic is 33.52019002375297 at 1 degree of freedom, which it is slightly less than 0.05 at 3.84, indicates that the difference in the number of laughter events between callers and receivers is statistically significant. We can reject the null hypothesis and conclude that callers tend to have more laughter events than receivers.

1.2 Student's t-Test for Continuous Data (Questions 3 and 4)

3. Question 3: Duration of Laughter Events - Women vs. Men Are laughter events longer for women? H1: Laughter events are longer for women than for men. H0: There is no significant difference in the duration of laughter events between women and men.

- Sample Means and Variances: Calculated from the duration data for each gender.

4. Question 4: Duration of Laughter Events - Callers vs. Receivers Are laughter events longer for callers? H1: Laughter events are longer for callers than for receivers. H0: There is no significant difference in the duration of laughter events between callers and receivers.

- Sample Means and Variances: Calculated from the duration data for each role.

[ ]:
```
# Separate data by gender and role for duration analysis
female_duration = data[data['Gender'] == 'Female']['Duration']
male_duration = data[data['Gender'] == 'Male']['Duration']
caller_duration = data[data['Role'] == 'Caller']['Duration']
receiver_duration = data[data['Role'] == 'Receiver']['Duration']
```

[ ]:
```
# Function to calculate t-statistic for independent samples
def t_test_independent(sample1, sample2):
    mean1, mean2 = np.mean(sample1), np.mean(sample2)
    std1, std2 = np.std(sample1, ddof=1), np.std(sample2, ddof=1)
    n1, n2 = len(sample1), len(sample2)

    pooled_se = np.sqrt(std1**2/n1 + std2**2/n2)
    t_statistic = (mean1 - mean2) / pooled_se
    df = (std1**2/n1 + std2**2/n2)**2 / ((std1**2/n1)**2/(n1-1) + (std2**2/
 ↪n2)**2/(n2-1))
    return t_statistic, df
```

[ ]:
```
# Calculate mean and standard deviation for each group and store them in␣
 ↪variables
female_mean, female_std = np.mean(female_duration), np.std(female_duration,␣
 ↪ddof=1)
male_mean, male_std = np.mean(male_duration), np.std(male_duration, ddof=1)
caller_mean, caller_std = np.mean(caller_duration), np.std(caller_duration,␣
 ↪ddof=1)
receiver_mean, receiver_std = np.mean(receiver_duration), np.
 ↪std(receiver_duration, ddof=1)

# Create dataframes for male and female
gender_stats = pd.DataFrame({
    'Gender': ['Male', 'Female'],
    'Mean Duration': [male_mean, female_mean],
    'Standard Deviation': [male_std, female_std]
```

```
})

# Create dataframes for caller and receiver
role_stats = pd.DataFrame({
    'Role': ['Caller', 'Receiver'],
    'Mean Duration': [caller_mean, receiver_mean],
    'Standard Deviation': [caller_std, receiver_std]
})

# Display the dataframes
print(gender_stats.to_string(index=False))
print()
print(role_stats.to_string(index=False))
```

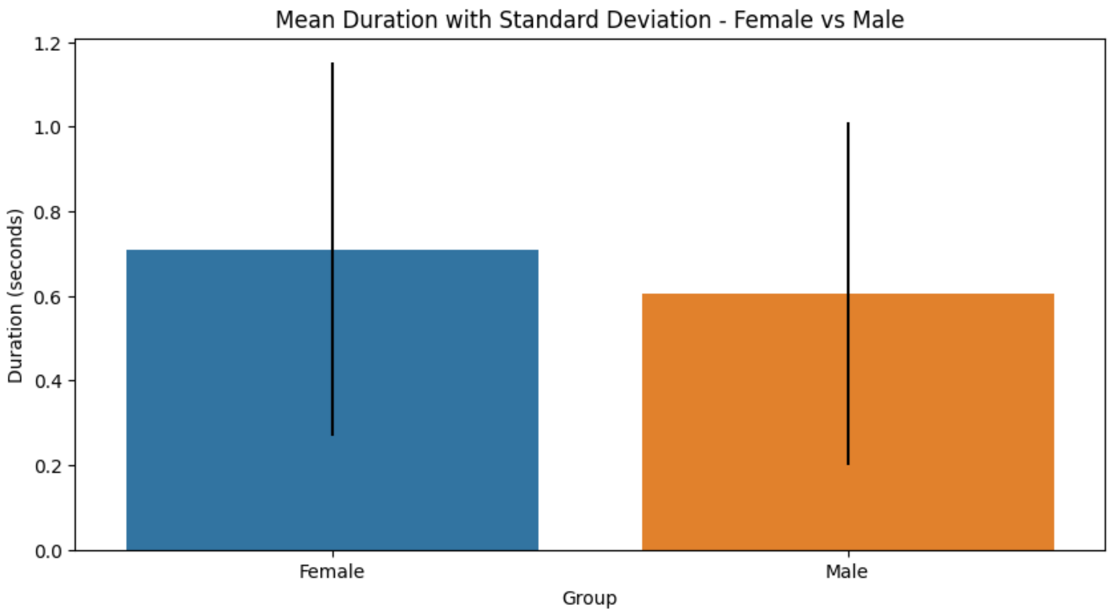
```
Gender  Mean Duration  Standard Deviation
  Male       0.606231            0.404662
Female       0.709685            0.441615

    Role  Mean Duration  Standard Deviation
  Caller       0.745766            0.461145
Receiver       0.549401            0.346070
```

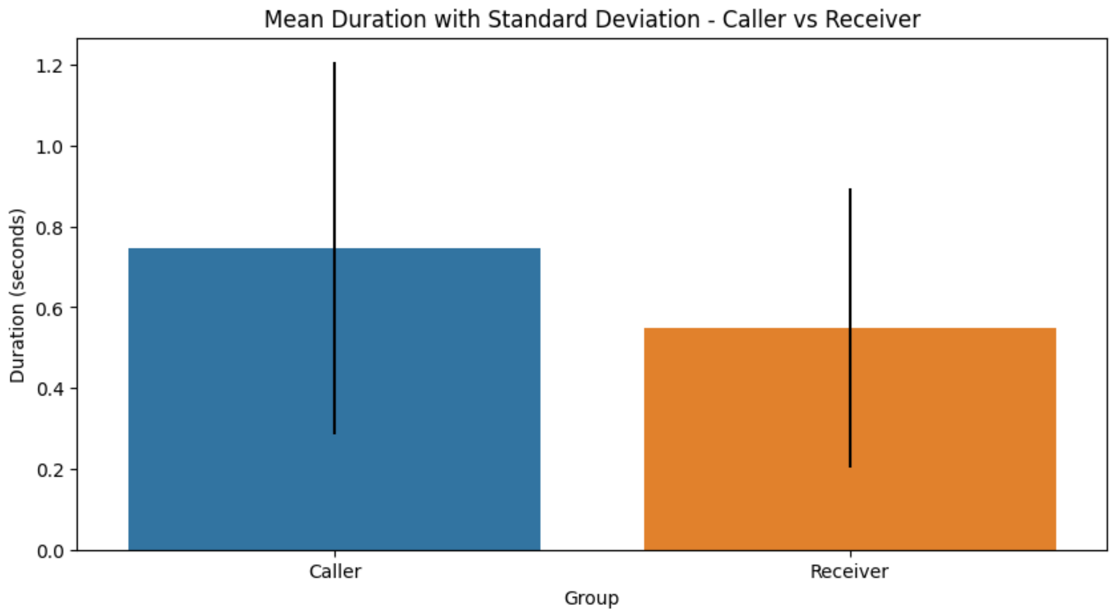
[ ]:
```
# Calculating t-statistic and degrees of freedom for gender and role
t_statistic_gender, df_gender = t_test_independent(female_duration,␣
 ↪male_duration)
t_statistic_role, df_role = t_test_independent(caller_duration,␣
 ↪receiver_duration)

# Display results
print(f"T-Test for Gender Duration (t-statistic): {t_statistic_gender}, Degrees␣
 ↪of Freedom: {df_gender}")
print(f"T-Test for Role Duration (t-statistic): {t_statistic_role}, Degrees of␣
 ↪Freedom: {df_role}")
```

```
T-Test for Gender Duration (t-statistic): 3.5145791170880627, Degrees of
Freedom: 780.7755366076725
T-Test for Role Duration (t-statistic): 7.046925950205163, Degrees of Freedom:
828.5128834922407
```

[ ]:
```
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)

# Plotting bar graphs for mean and standard deviation - Female vs Male
plt.figure(figsize=(10, 5))
sns.barplot(x=['Female', 'Male'], y=[female_mean, male_mean], yerr=[female_std,␣
 ↪male_std])
plt.title('Mean Duration with Standard Deviation - Female vs Male')
```

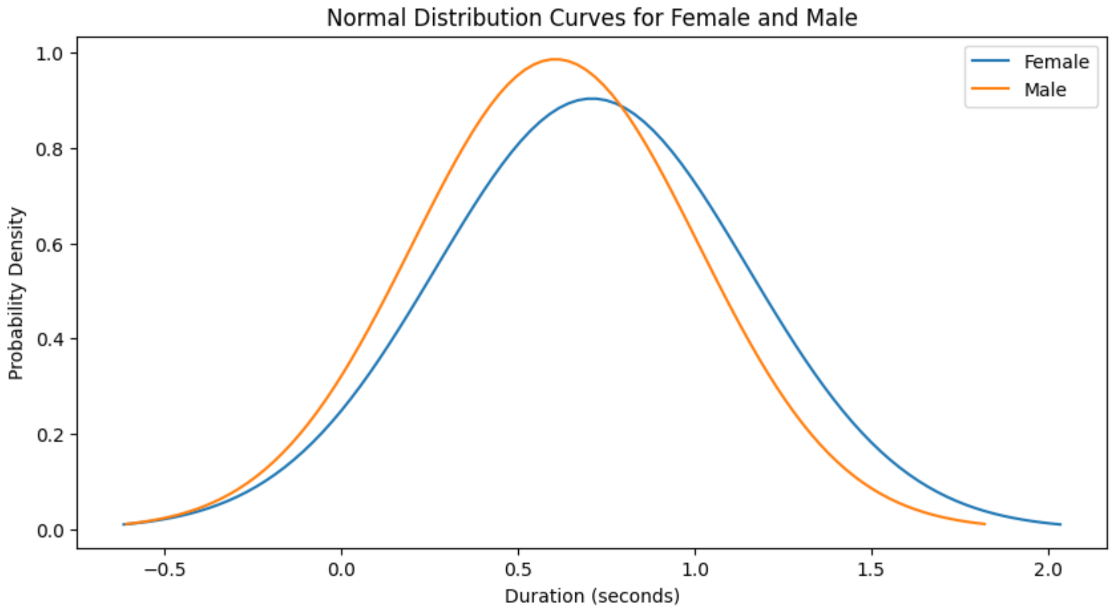
```python
plt.xlabel('Group')
plt.ylabel('Duration (seconds)')
plt.show()

# Plotting bar graphs for mean and standard deviation - Caller vs Receiver
plt.figure(figsize=(10, 5))
sns.barplot(x=['Caller', 'Receiver'], y=[caller_mean, receiver_mean],␣
 ↪yerr=[caller_std, receiver_std])
plt.title('Mean Duration with Standard Deviation - Caller vs Receiver')
plt.xlabel('Group')
plt.ylabel('Duration (seconds)')
plt.show()
```

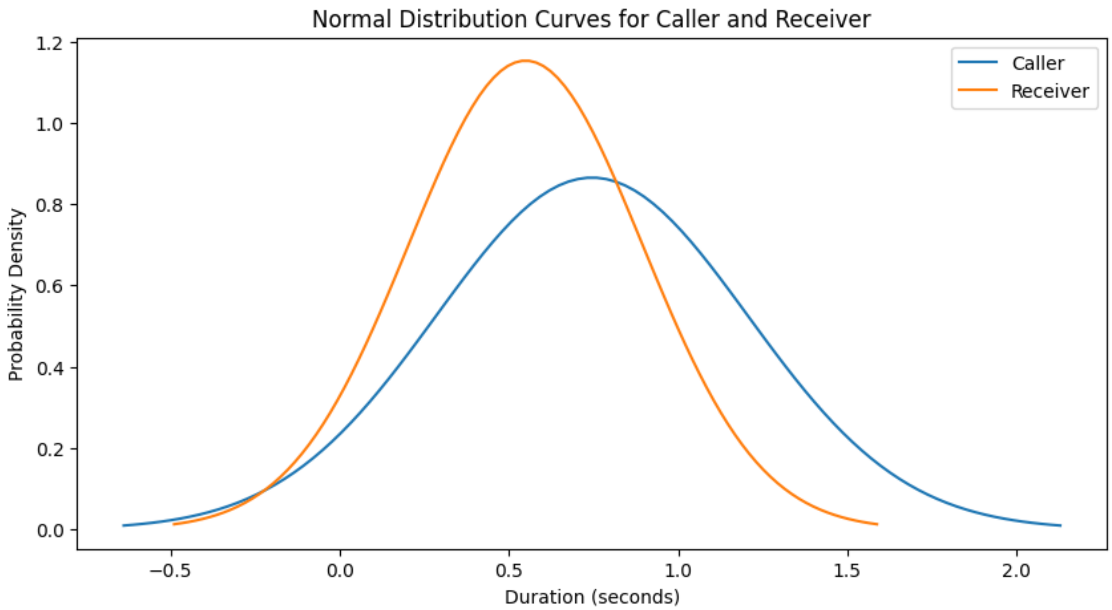




[ ]:
```
from scipy.stats import norm # Import normal distribution function only to
 ↪graph my results

# Function to plot normal distribution
def plot_normal_distribution(mean, std, label):
    # Generate a range of values
    x = np.linspace(mean - 3*std, mean + 3*std, 100)
    # Calculate the normal distribution values
    y = norm.pdf(x, mean, std)
    plt.plot(x, y, label=label)

# Plotting normal distribution curves for female and male
plt.figure(figsize=(10, 5))
plot_normal_distribution(female_mean, female_std, 'Female')
plot_normal_distribution(male_mean, male_std, 'Male')
plt.title('Normal Distribution Curves for Female and Male')
plt.xlabel('Duration (seconds)')
plt.ylabel('Probability Density')
plt.legend()
plt.show()

# Plotting normal distribution curves for caller and receiver
plt.figure(figsize=(10, 5))
plot_normal_distribution(caller_mean, caller_std, 'Caller')
plot_normal_distribution(receiver_mean, receiver_std, 'Receiver')
plt.title('Normal Distribution Curves for Caller and Receiver')
plt.xlabel('Duration (seconds)')
```

```
plt.ylabel('Probability Density')
plt.legend()
plt.show()
```





2 Part 2

[ ]:
```
# Load data
train_data = pd.read_csv("csv/training-part-2.csv")
test_data = pd.read_csv("csv/test-part-2.csv")
```

[ ]:
```
# Separating features and labels in both training and test datasets
X_train = train_data.drop('Class', axis=1)      # Features for training
y_train = train_data['Class']                   # Labels for training
X_test = test_data.drop('Class', axis=1)        # Features for testing
y_test = test_data['Class']                     # Labels for testing

def calculate_parameters(features, labels):
    ''' Calculates mean and variance for each class in the dataset'''
    classes = labels.unique()
    parameters = {}
    for cls in classes:
        class_features = features[labels == cls]
        parameters[cls] = {
            'mean': class_features.mean(),
            'variance': class_features.var()
        }
    return parameters
```

[ ]:
```
def gaussian_pdf(x, mean, variance):
    """ Gaussian Probability Density Function """
    exponent = np.exp(-(x - mean) ** 2 / (2 * variance))
    return (1 / np.sqrt(2 * np.pi * variance)) * exponent

def classify(features, parameters):
    """ Classify each instance in features """
    classes = list(parameters.keys())
    likelihoods = {cls: 0 for cls in classes}
    for cls in classes:
        likelihoods[cls] = np.prod(gaussian_pdf(features,
                                               parameters[cls]['mean'],
                                               parameters[cls]['variance']))
    return max(likelihoods, key=likelihoods.get)
```

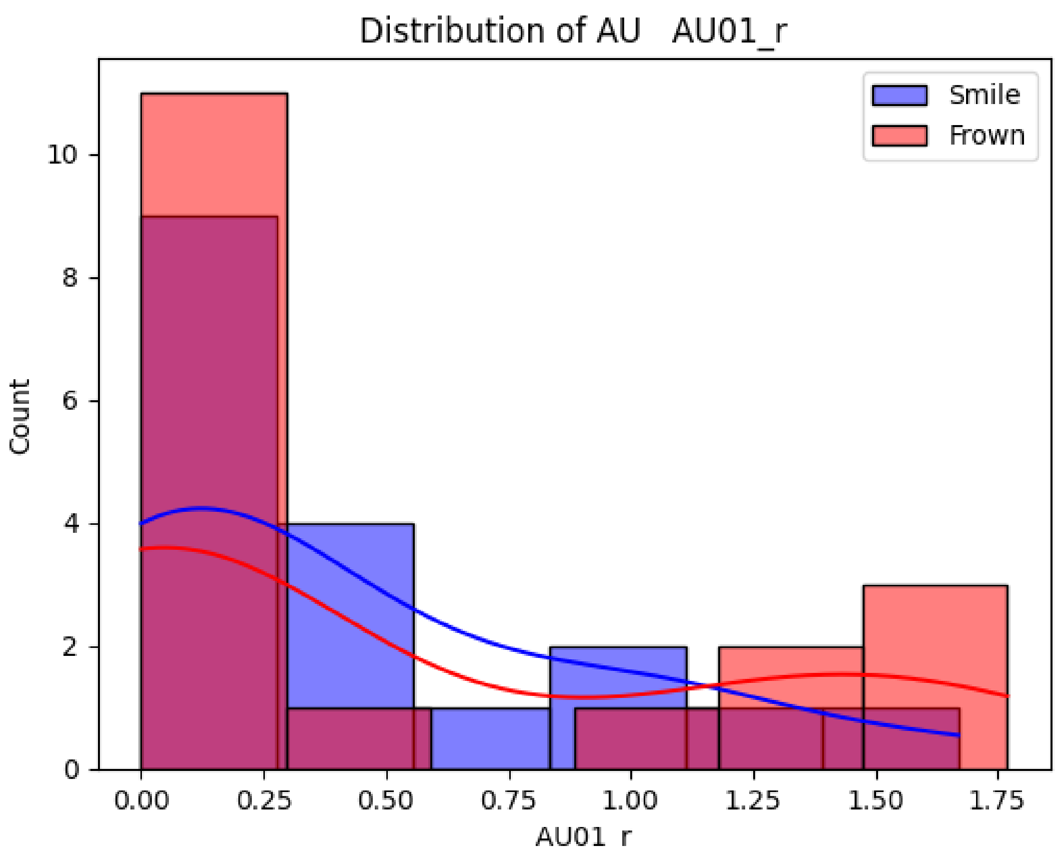
[ ]:
```
def calculate_error_rate(predictions, actual_labels):
    """ Calculate error rate for predictions """
    incorrect_predictions = (predictions != actual_labels).sum()
    total_predictions = len(actual_labels)
    error_rate = incorrect_predictions / total_predictions
    return error_rate * 100  # To get the percentage

def predict_row(row, parameters):
```

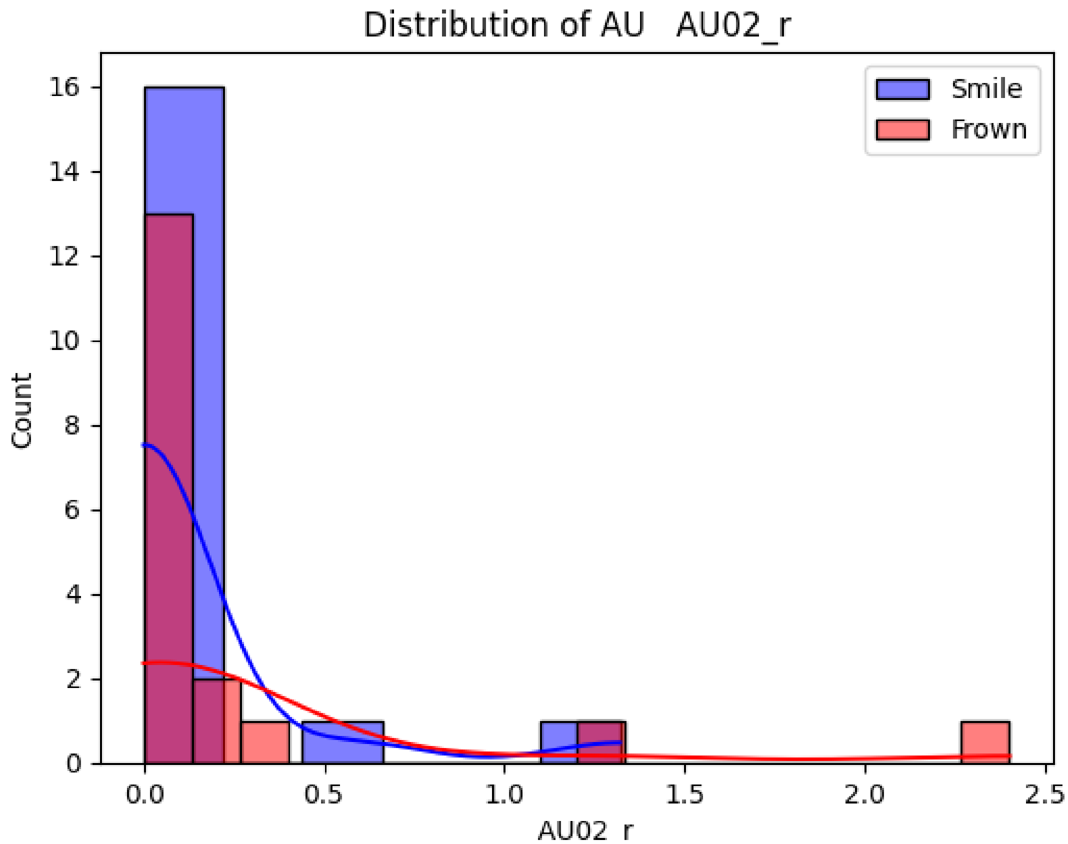
```
    """ Classify a single row of features """
    return classify(row, parameters)
```

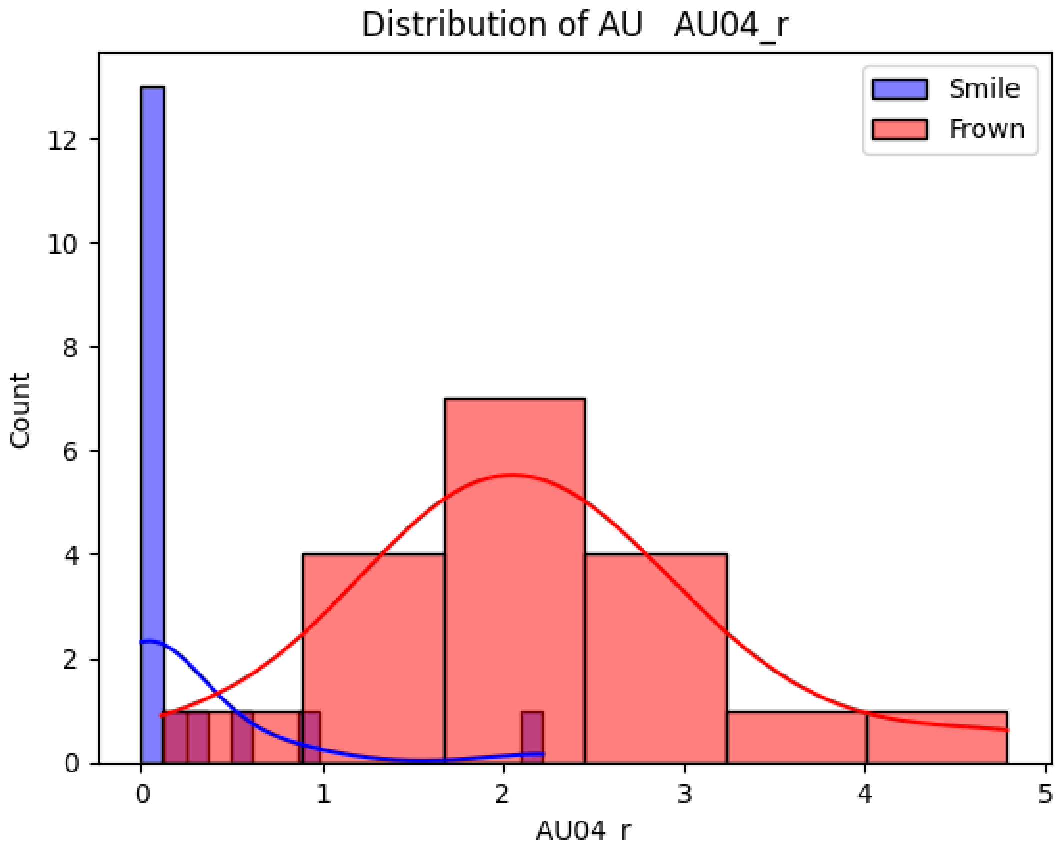
[ ]:
```
# Calculating mean and variance for each class in the training set
parameters = calculate_parameters(X_train, y_train)

# Predicting class for each instance in the test set
predictions = []
for index, row in X_test.iterrows():
    predicted_class = predict_row(row, parameters)
    predictions.append(predicted_class)

# Convert predictions to a Pandas Series for easy comparison
predictions = pd.Series(predictions, index=X_test.index)

# Calculate error rate and display predictions vs actual labels
error_rate = calculate_error_rate(predictions, y_test)
comparison_df = pd.DataFrame({'Predictions': predictions, 'Actual': y_test})

# Highlight when a prediction doesn't match an actual
comparison_df['Match'] = comparison_df['Predictions'] == comparison_df['Actual']
comparison_df['Match'] = comparison_df['Match'].apply(lambda x: 'Yes' if x else␣
 ↪'No')

print(f"Error Rate: {error_rate}%")
print(comparison_df.to_string(index=False))
```

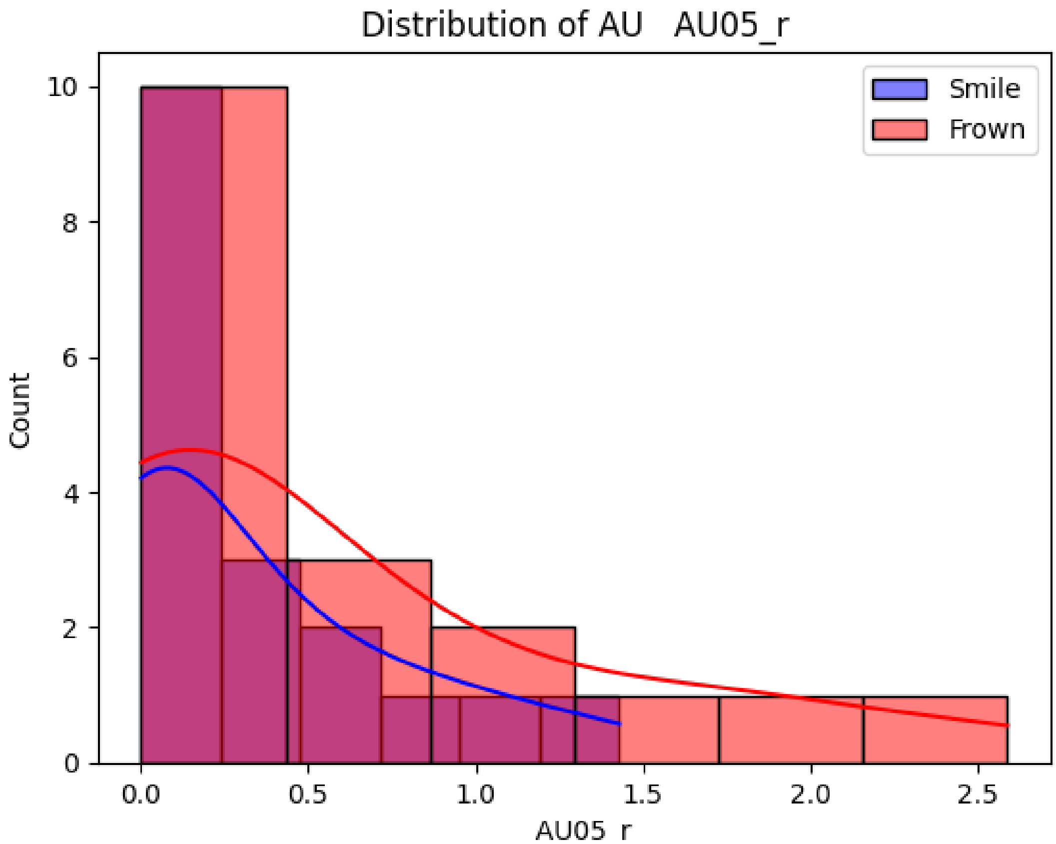
```
Error Rate: 12.5%
Predictions Actual Match
      frown  frown   Yes
      frown  frown   Yes
      frown  frown   Yes
      frown  frown   Yes
      smile  frown    No
      frown  frown   Yes
      frown  frown   Yes
      frown  frown   Yes
      smile  smile   Yes
      smile  smile   Yes
      smile  smile   Yes
      smile  smile   Yes
      smile  smile   Yes
      smile  smile   Yes
      frown  smile    No
      smile  smile   Yes
```

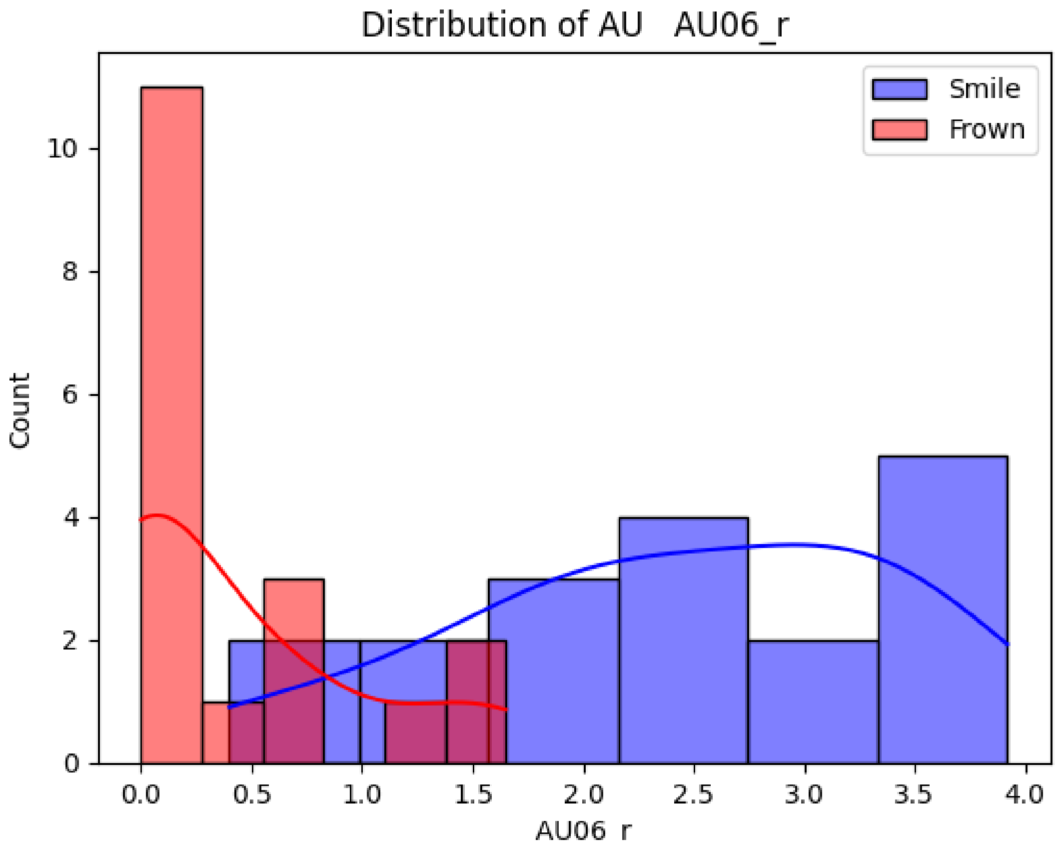
[ ]:
```
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)

# Histograms/Density Plots
for au in X_train.columns:
    sns.histplot(X_train[y_train == 'smile'][au], color='blue', label='Smile',␣
 ↪kde=True)
    sns.histplot(X_train[y_train == 'frown'][au], color='red', label='Frown',␣
 ↪kde=True)
    plt.title(f"Distribution of AU {au}")
    plt.legend()
    plt.show()
```

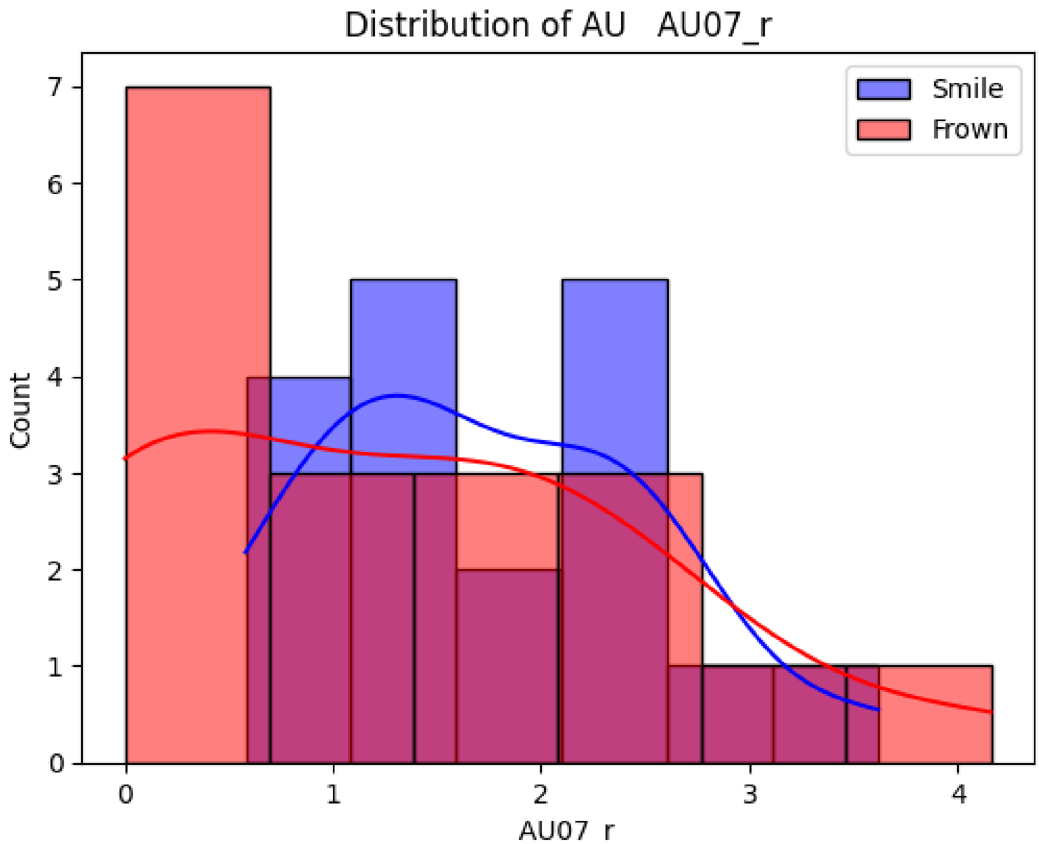


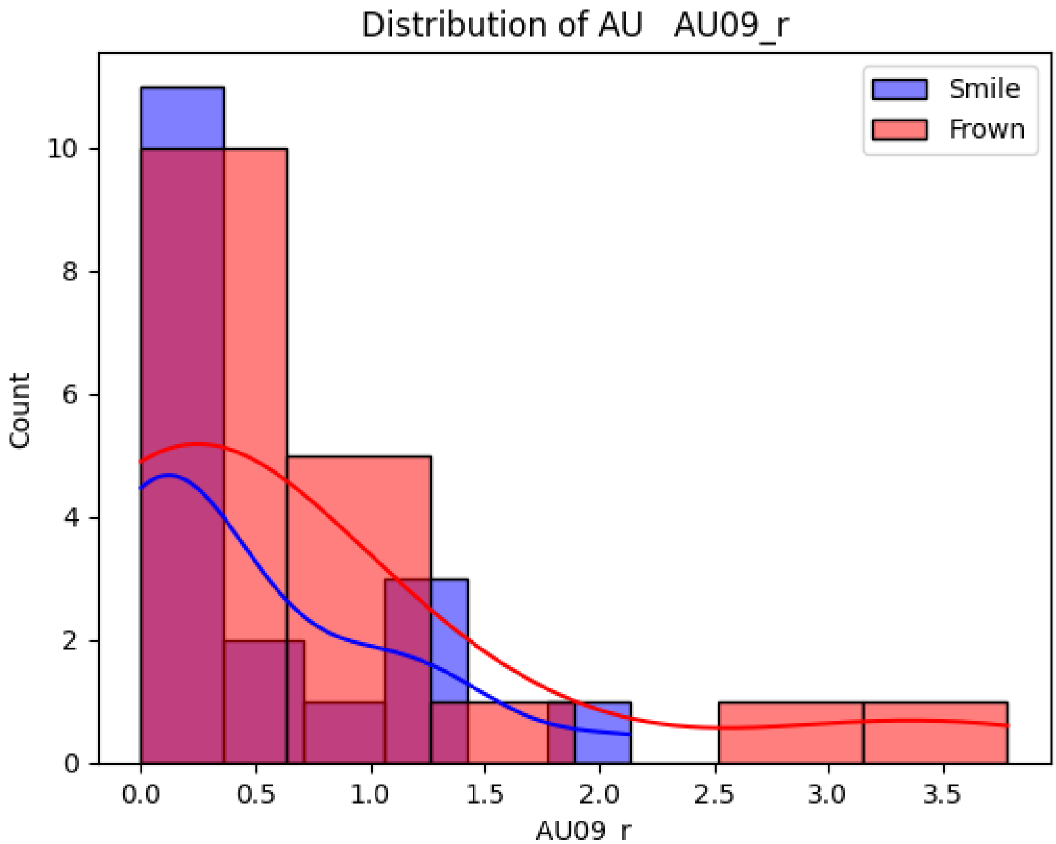

Distribution of AU   AU02_r
Smile
Frown
Count
AU02_r
0
2
4
6
8
10
12
14
16
0.0
0.5
1.0
1.5
2.0
2.5


Distribution of AU AU04_r
Smile
Frown
Count
AU04_r
0
1
2
3
4
5
0
2
4
6
8
10
12

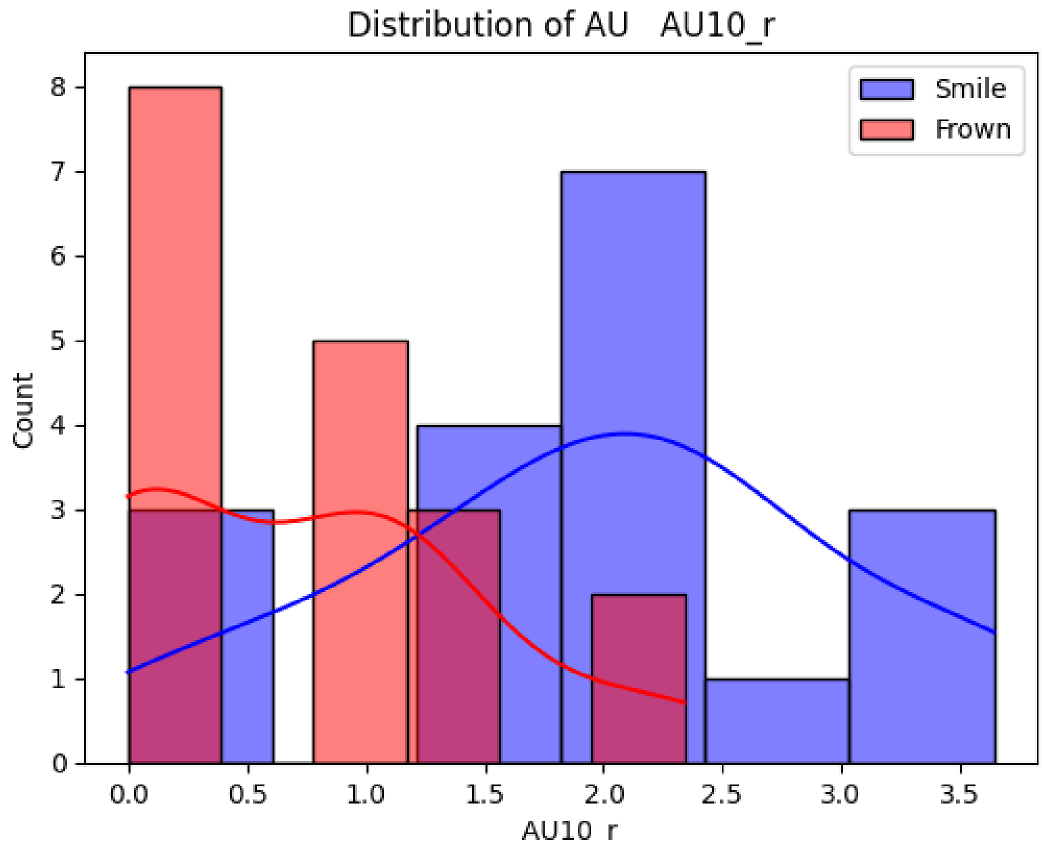

Distribution of AU AU05_r
Smile
Frown
Count
AU05_r
0
2
4
6
8
10
0.0
0.5
1.0
1.5
2.0
2.5

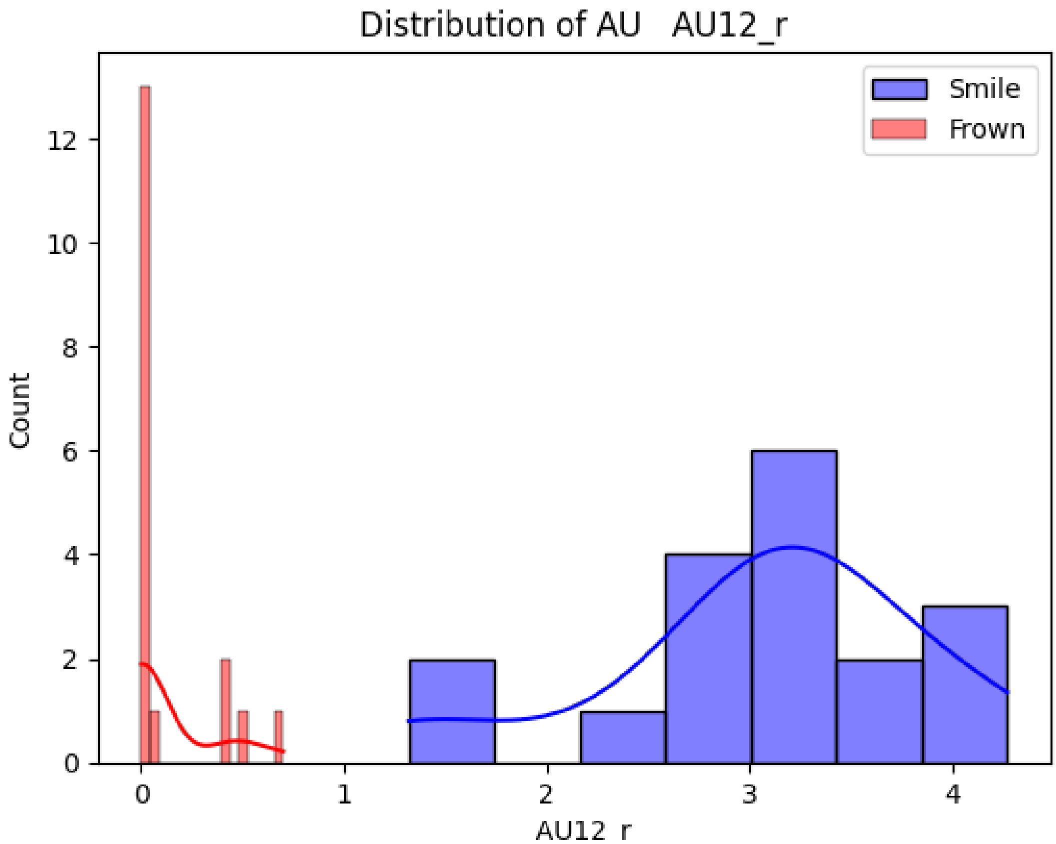

Distribution of AU AU06_r
Smile
Frown
Count
AU06_r
0
2
4
6
8
10
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

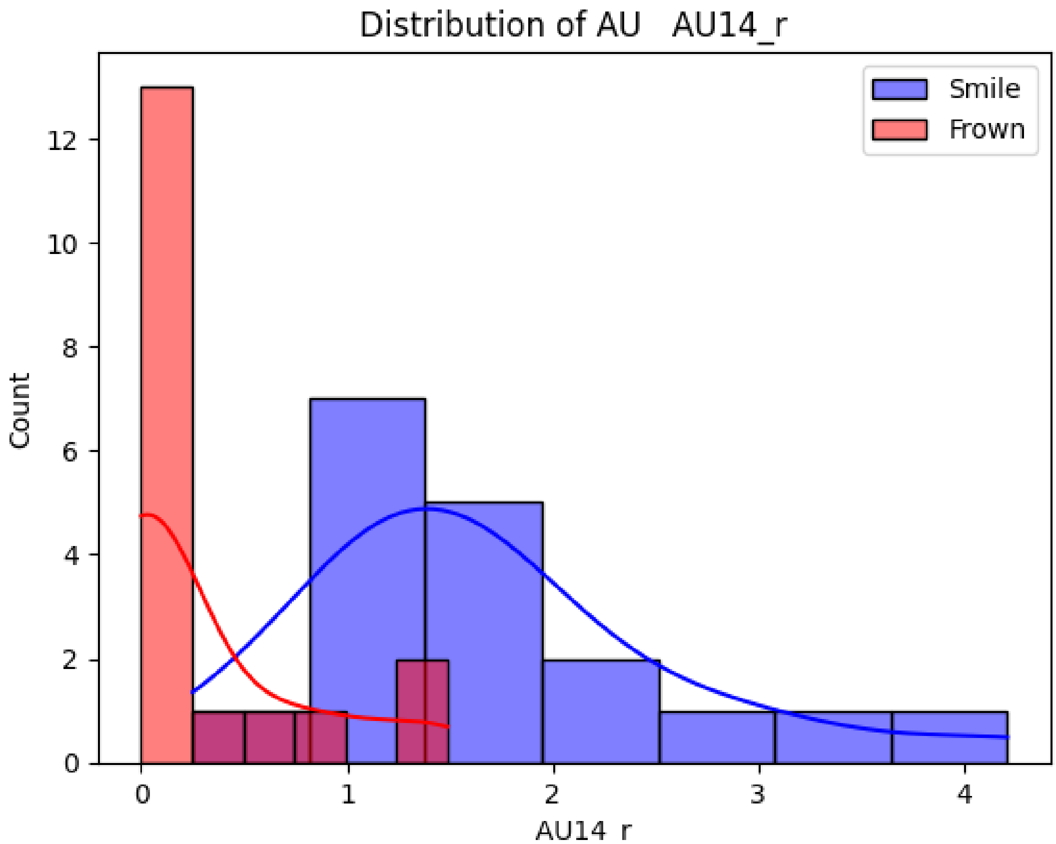

Distribution of AU AU07_r
Smile
Frown
Count
AU07_r
0
1
2
3
4
5
6
7
0
1
2
3
4

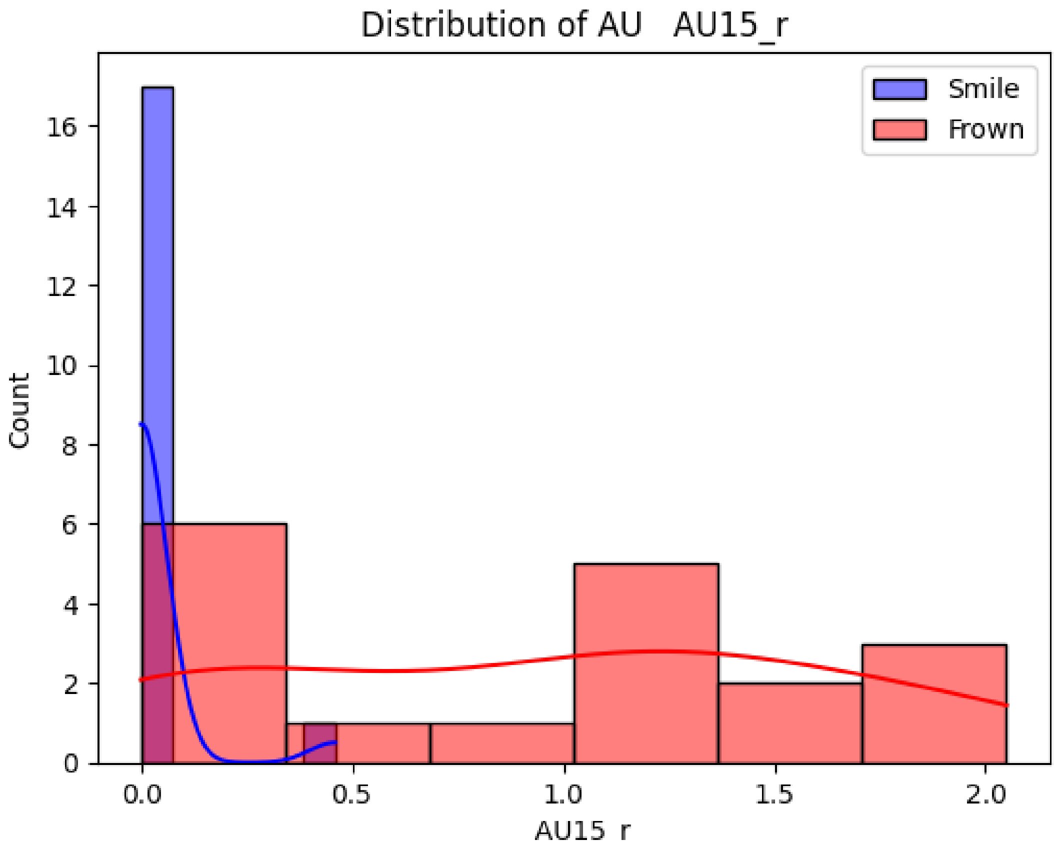


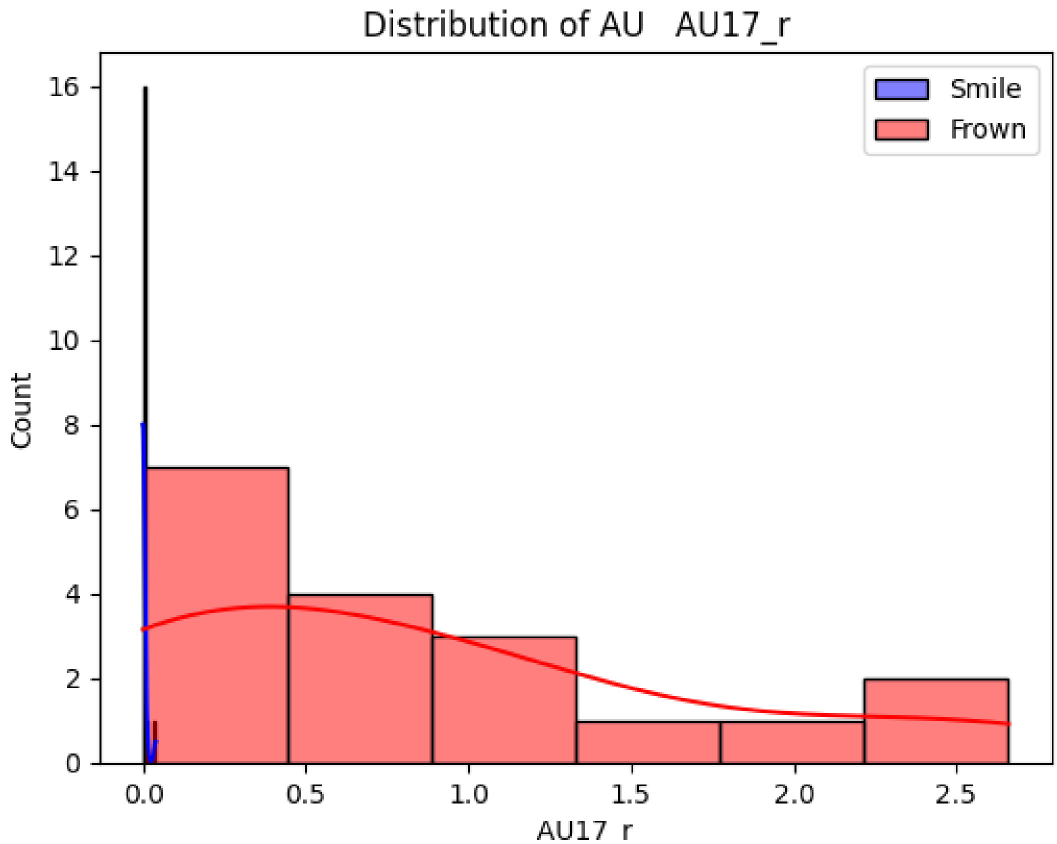
Distribution of AU AU09_r
Smile
Frown
Count
AU09_r
0
2
4
6
8
10
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

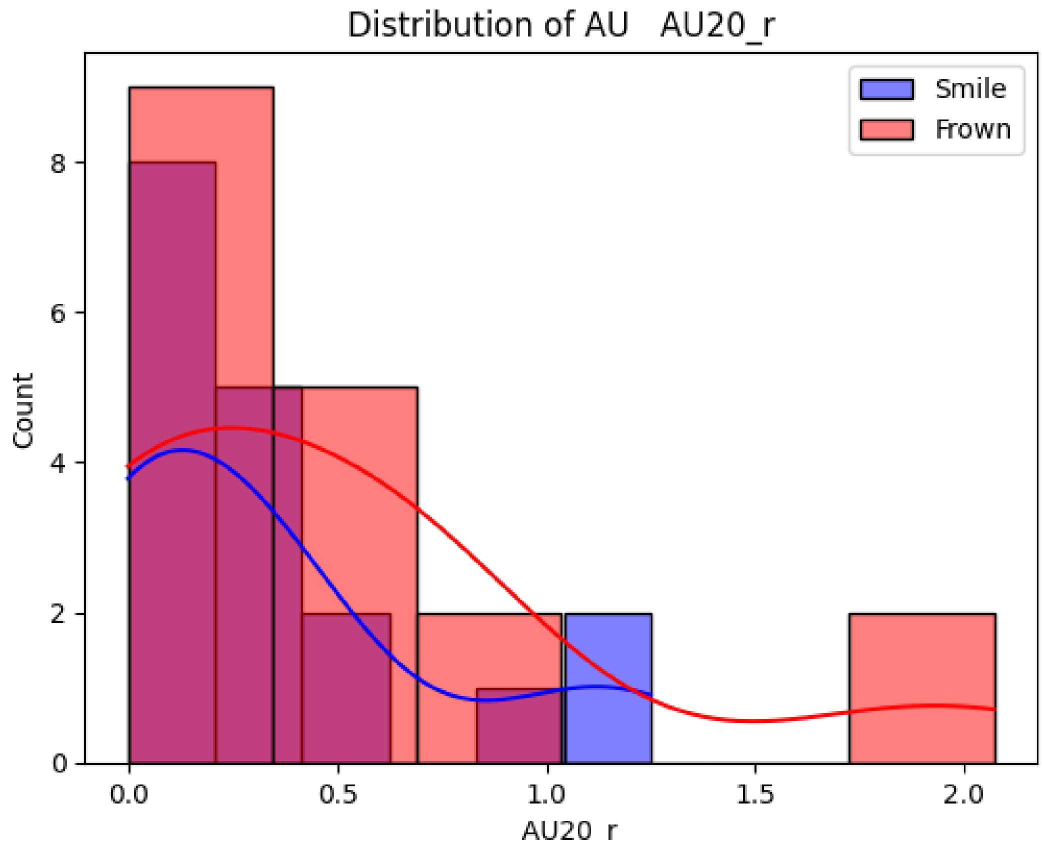

Distribution of AU AU10_r
Smile
Frown
Count
AU10_r
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

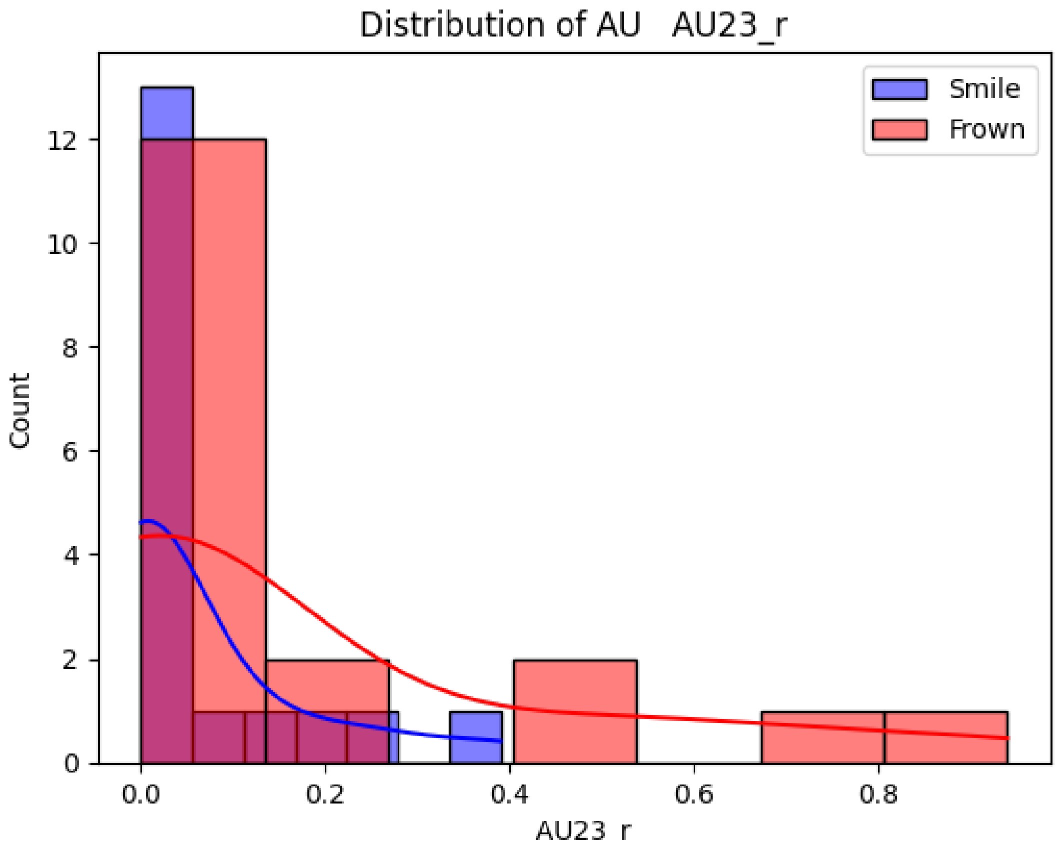

Distribution of AU AU12_r
Smile
Frown
Count
0
2
4
6
8
10
12
0
1
2
3
4
AU12_r

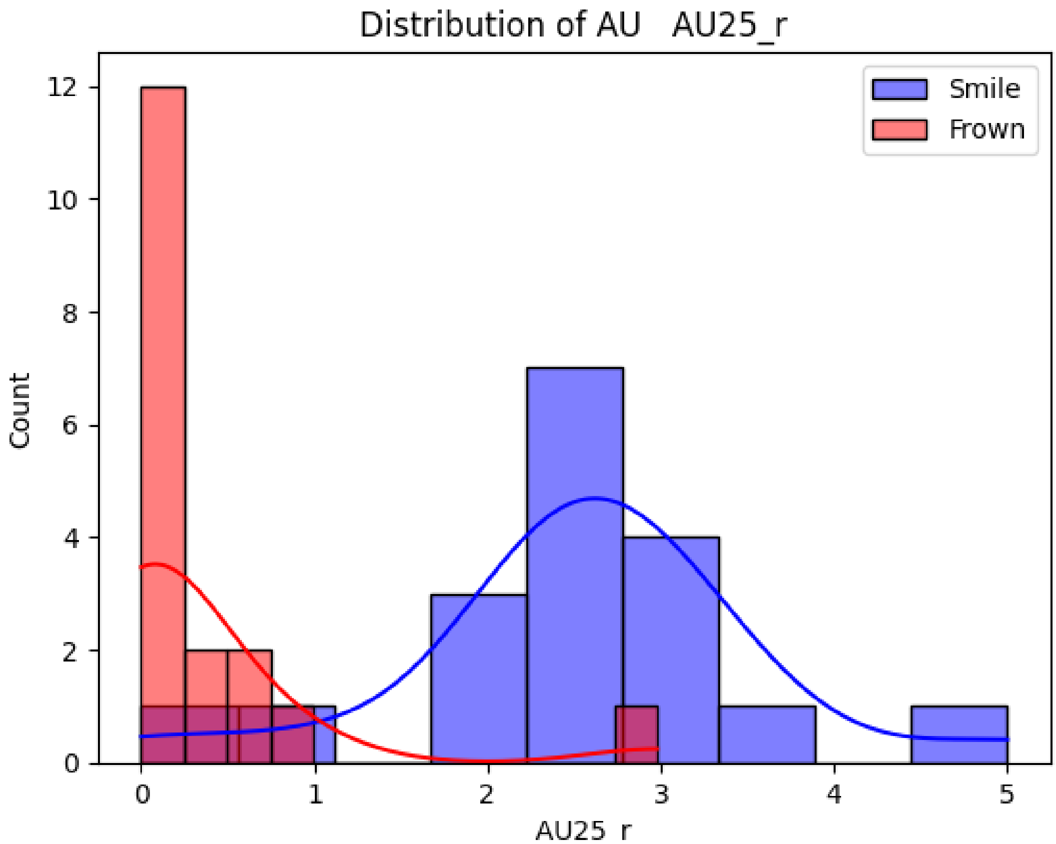

Distribution of AU AU14_r
Smile
Frown
Count
AU14_r
0
2
4
6
8
10
12
0
1
2
3
4


Distribution of AU AU15_r
Smile
Frown
Count
AU15_r
0
2
4
6
8
10
12
14
16
0.0
0.5
1.0
1.5
2.0

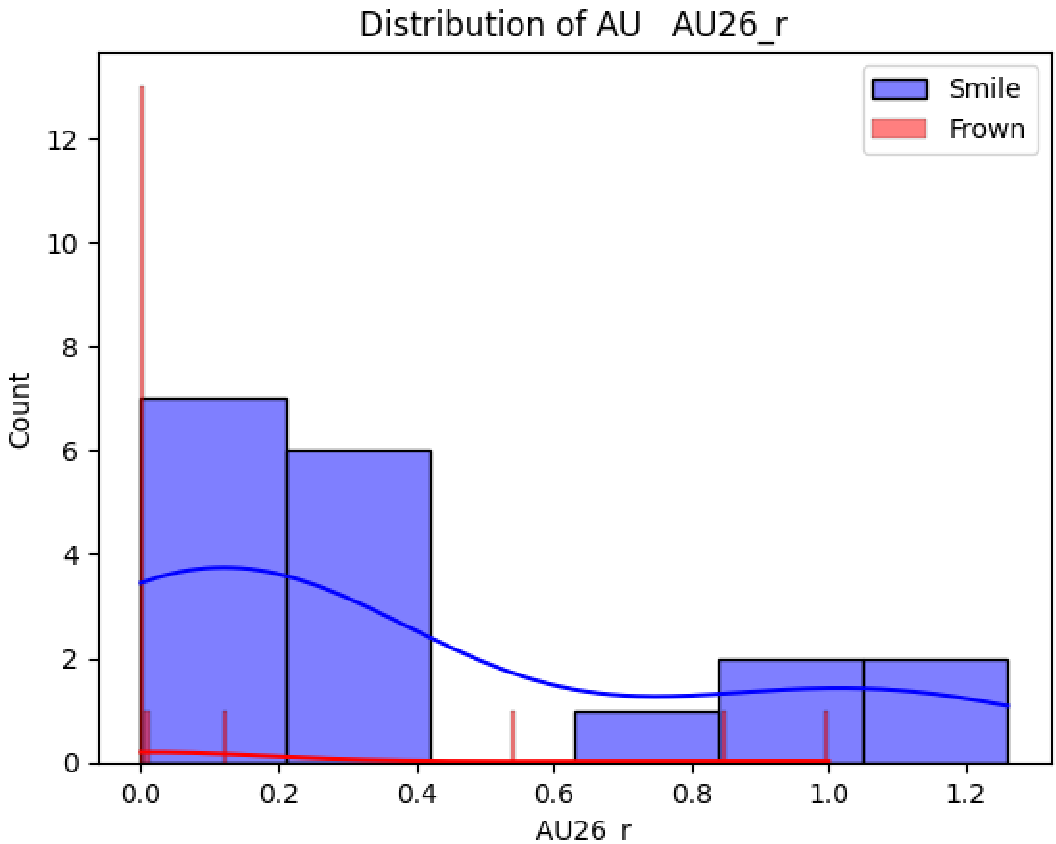

Distribution of AU   AU17_r
Smile
Frown
Count
AU17_r
0
2
4
6
8
10
12
14
16
0.0
0.5
1.0
1.5
2.0
2.5

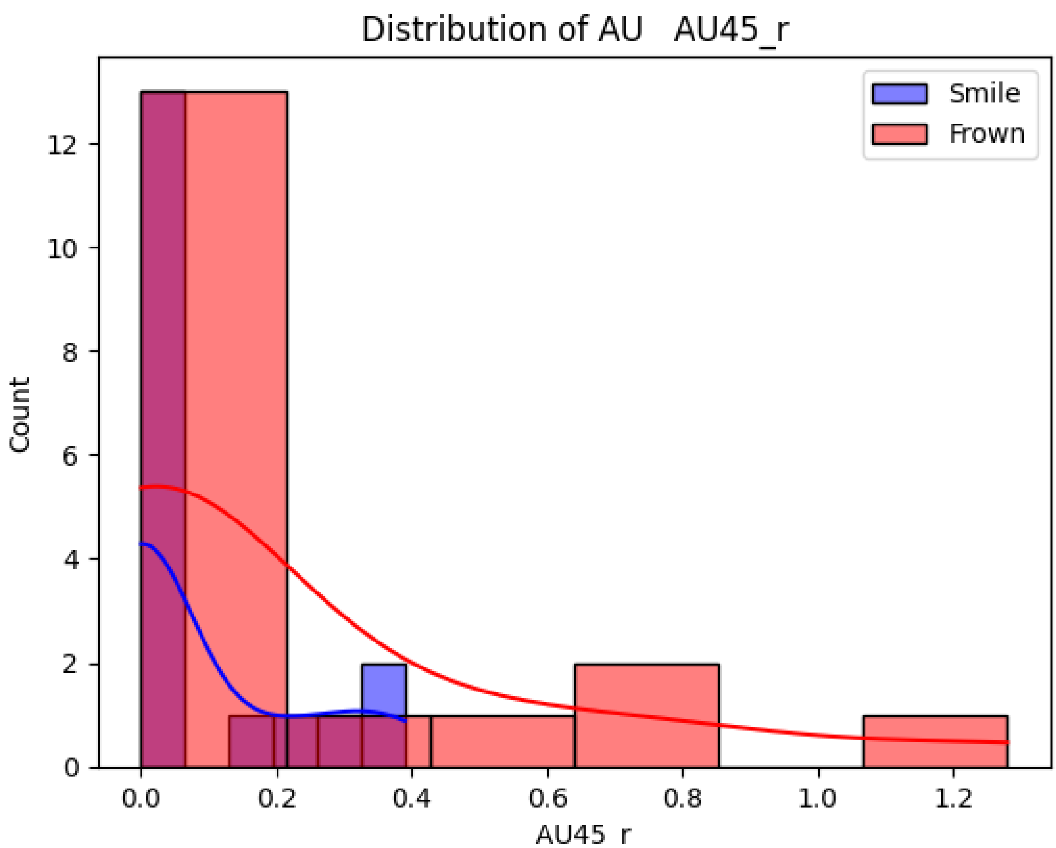

Distribution of AU AU20_r
Smile
Frown
Count
AU20_r
0
2
4
6
8
0.0
0.5
1.0
1.5
2.0


Distribution of AU AU23_r
Smile
Frown
Count
AU23_r
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8

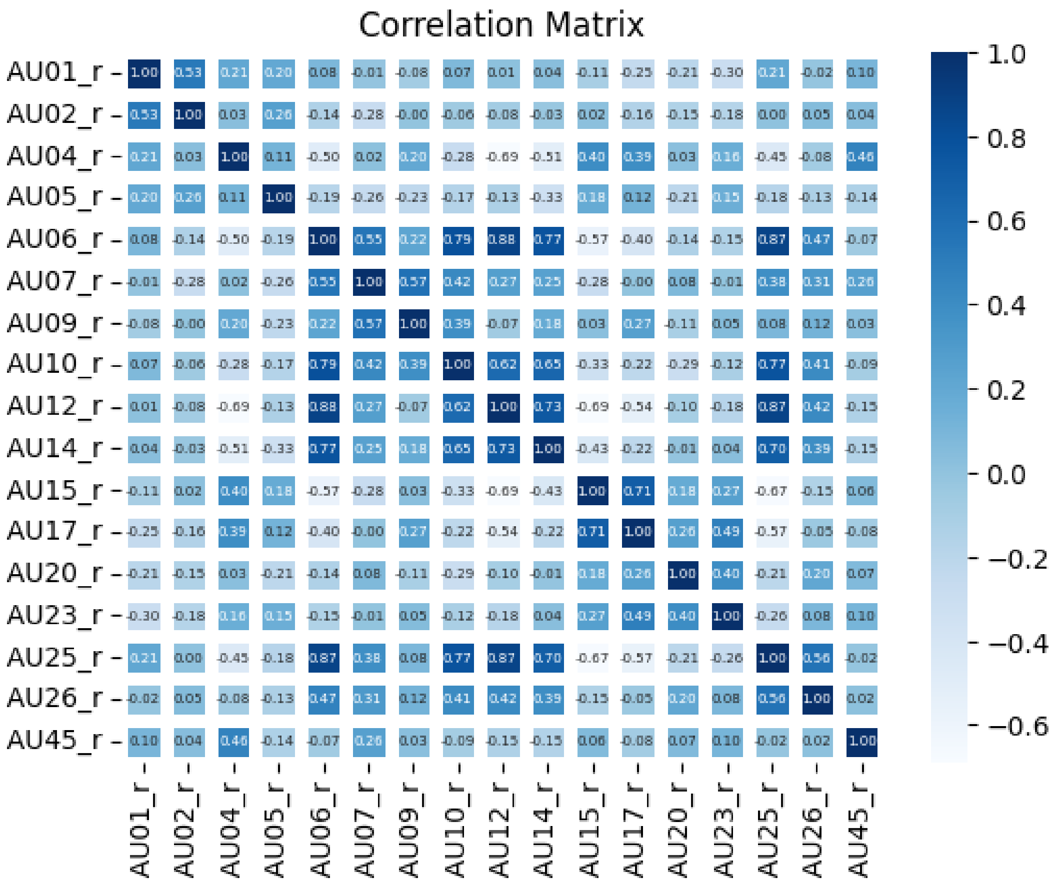


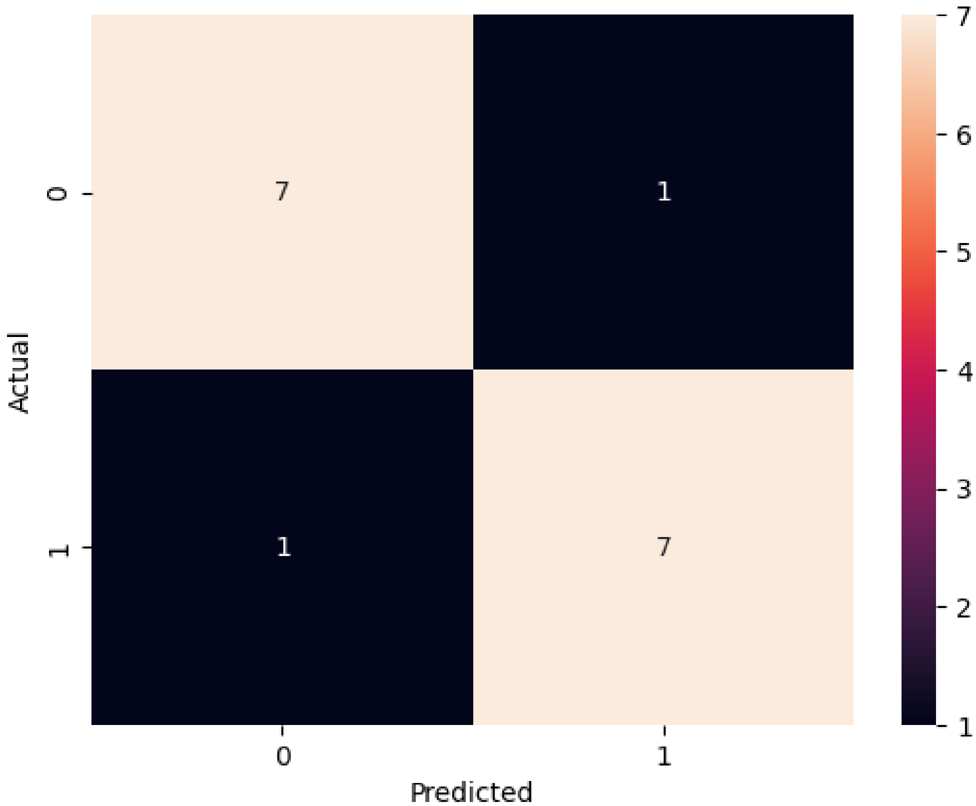
Distribution of AU   AU25_r
Smile
Frown
Count
AU25_r
0
2
4
6
8
10
12
0
1
2
3
4
5


Distribution of AU AU26_r
Smile
Frown
Count
AU26_r
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0
1.2




```
[ ]: from sklearn.metrics import confusion_matrix

# Correlation Matrix
corr_matrix = X_train.corr()
sns.heatmap(corr_matrix, annot=True, cmap='Blues', linewidths=5.0,␣
 ↪annot_kws={"size": 5}, fmt=".2f")
plt.title("Correlation Matrix")
plt.show()

# Confusion Matrix
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot=True, fmt='d')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```


Correlation Matrix
AU01_r
AU02_r
AU04_r
AU05_r
AU06_r
AU07_r
AU09_r
AU10_r
AU12_r
AU14_r
AU15_r
AU17_r
AU20_r
AU23_r
AU25_r
AU26_r
AU45_r
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6




```
[ ]: from sklearn.metrics import classification_report
     print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

       frown       0.88      0.88      0.88         8
       smile       0.88      0.88      0.88         8

    accuracy                           0.88        16
   macro avg       0.88      0.88      0.88        16
weighted avg       0.88      0.88      0.88        16
```