In [2]:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sc
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sc
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import random
```

In [3]:

```
# Load dataset
data = pd.read_csv("D:\MTP\Mid Term\MId Term 256 models.csv")
data
```

Out[3]:

| | Sl No | x1 | y1 | x2 | y2 | area | Model horizontal Class | Model Vertical class | Model Box Class | natural frequency of Mode 1 | natural frequency of Mode 2 | natu freque of Moc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.00 | 0.0 | 0.06 | 0.06 | 0.0036 | 1 | 1 | 1 | 6.8818 | 25.361 | 27. |
| **1** | 2 | 0.06 | 0.0 | 0.12 | 0.06 | 0.0036 | 1 | 1 | 1 | 6.8854 | 25.535 | 27. |
| **2** | 3 | 0.12 | 0.0 | 0.18 | 0.06 | 0.0036 | 1 | 1 | 1 | 6.9176 | 25.713 | 27. |
| **3** | 4 | 0.18 | 0.0 | 0.24 | 0.06 | 0.0036 | 2 | 1 | 1 | 6.9214 | 25.759 | 27. |
| **4** | 5 | 0.24 | 0.0 | 0.30 | 0.06 | 0.0036 | 2 | 1 | 1 | 6.9401 | 25.767 | 27. |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **251** | 252 | 0.66 | 0.9 | 0.72 | 0.96 | 0.0036 | 4 | 5 | 9 | 6.9393 | 25.759 | 27. |
| **252** | 253 | 0.72 | 0.9 | 0.78 | 0.96 | 0.0036 | 4 | 5 | 9 | 6.9443 | 25.757 | 27. |
| **253** | 254 | 0.78 | 0.9 | 0.84 | 0.96 | 0.0036 | 5 | 5 | 9 | 6.9254 | 25.725 | 27. |
| **254** | 255 | 0.84 | 0.9 | 0.90 | 0.96 | 0.0036 | 5 | 5 | 9 | 6.9163 | 25.691 | 27. |
| **255** | 256 | 0.90 | 0.9 | 0.96 | 0.96 | 0.0036 | 5 | 5 | 9 | 6.9305 | 25.705 | 27. |

256 rows × 15 columns

In [3]:
```python
# Remove specific columns using NumPy
columns_to_remove = ['Sl No', 'area', 'Model horizontal Class', 'Model Ver
data.drop(columns=columns_to_remove, inplace=True)
data.head()
```

Out[3]:

| | x1 | y1 | x2 | y2 | natural frequency of Mode 1 | natural frequency of Mode 2 | natural frequency of Mode 3 | natural frequency of Mode 4 | natural frequency of Mode 5 | natural frequency of Mode 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.00 | 0.0 | 0.06 | 0.06 | 6.8818 | 25.361 | 27.925 | 46.167 | 67.591 | 72.011 |
| **1** | 0.06 | 0.0 | 0.12 | 0.06 | 6.8854 | 25.535 | 27.926 | 46.424 | 68.150 | 72.073 |
| **2** | 0.12 | 0.0 | 0.18 | 0.06 | 6.9176 | 25.713 | 27.932 | 46.575 | 68.315 | 72.107 |
| **3** | 0.18 | 0.0 | 0.24 | 0.06 | 6.9214 | 25.759 | 27.926 | 46.586 | 68.098 | 72.105 |
| **4** | 0.24 | 0.0 | 0.30 | 0.06 | 6.9401 | 25.767 | 27.916 | 46.589 | 67.902 | 72.108 |

In [4]:
```python
# Select features and target variables
features = [' natural frequency of Mode 1', ' natural frequency of Mode 2'
            ' natural frequency of Mode 3', ' natural frequency of Mode 4'
            ' natural frequency of Mode 5', ' natural frequency of Mode 6'
targets = [' x1', ' y1', ' x2', ' y2']
```

In [5]:
```python
# Split the data into features (X) and targets (y)
X = data[features]
y = data[targets]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
```

In [6]:

```python
# Initialize a multi-output regression model (RandomForestRegressor as an
base_regressor = RandomForestRegressor(random_state=42)

# Create a multi-output wrapper for the regressor
multi_output_regressor = MultiOutputRegressor(base_regressor)

# Define parameter grid
param_grid = {
    'estimator__n_estimators': [100, 200, 300],
    'estimator__max_depth': [None, 10, 20, 30],
    'estimator__min_samples_split': [2, 5, 10],
    'estimator__min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV with MultiOutputRegressor
grid_search = GridSearchCV(multi_output_regressor, param_grid, cv=5)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_test =grid_search.predict(X_test)

# Once the model is trained, predict values for the training set
y_pred_train = grid_search.predict(X_train)
```

In [7]:

```python
# Evaluate the model
mse = mean_squared_error(y_test, y_pred_test)
mae = mean_absolute_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
medae = median_absolute_error(y_test, y_pred_test)
evs = explained_variance_score(y_test, y_pred_test)
msle = mean_squared_log_error(y_test, y_pred_test)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R-squared:", r2)
print("Median Absolute Error:", medae)
print("Explained Variance Score:", evs)
print("Mean Squared Logarithmic Error:", msle)
```

```
Mean Squared Error: 0.019482247101215624
Mean Absolute Error: 0.09749223107770277
R-squared: 0.7549440539660275
Median Absolute Error: 0.06641448685110714
Explained Variance Score: 0.7612940212359287
Mean Squared Logarithmic Error: 0.00999186530538691
```

In [8]:

```
#Create a DataFrame to store actual and predicted values
output_values = pd.DataFrame(columns=['Actual_x1', 'Actual_y1', 'Actual_x2
                                      'Predicted_x1', 'Predicted_y1', 'Pre

#Iterate through the training dataset and append actual and predicted valu
for i in range(len(y_train)):
    actual_x1_train = y_train.iloc[i][' x1']
    actual_y1_train = y_train.iloc[i][' y1']
    actual_x2_train = y_train.iloc[i][' x2']
    actual_y2_train = y_train.iloc[i][' y2']

    predicted_x1_train = y_pred_train[i][0]
    predicted_y1_train = y_pred_train[i][1]
    predicted_x2_train = y_pred_train[i][2]
    predicted_y2_train = y_pred_train[i][3]

    output_values = pd.concat([output_values, pd.DataFrame({'Actual_x1': [
                                                            'Actual_y1': [
                                                            'Actual_x2': [
                                                            'Actual_y2': [
                                                            'Predicted_x1'
                                                            'Predicted_y1'
                                                            'Predicted_x2'
                                                            'Predicted_y2'
                              ignore_index=True)

#Iterate through the testing dataset and append actual and predicted value
for i in range(len(y_test)):
    actual_x1_test = y_test.iloc[i][' x1']
    actual_y1_test = y_test.iloc[i][' y1']
    actual_x2_test = y_test.iloc[i][' x2']
    actual_y2_test = y_test.iloc[i][' y2']

    predicted_x1_test = y_pred_test[i][0]
    predicted_y1_test = y_pred_test[i][1]
    predicted_x2_test = y_pred_test[i][2]
    predicted_y2_test = y_pred_test[i][3]

    output_values = pd.concat([output_values, pd.DataFrame({'Actual_x1': [
                                                            'Actual_y1': [
                                                            'Actual_x2': [
                                                            'Actual_y2': [
                                                            'Predicted_x1'
                                                            'Predicted_y1'
                                                            'Predicted_x2'
                                                            'Predicted_y2'
                              ignore_index=True)

# Display the DataFrame containing actual and predicted values
print("Combined Output Values:")
output_values
```

```
Combined Output Values:
```

Out[8]:

| | Actual_x1 | Actual_y1 | Actual_x2 | Actual_y2 | Predicted_x1 | Predicted_y1 | Predicted_x2 | Pred |
|---|---|---|---|---|---|---|---|---|

| | Actual_x1 | Actual_y1 | Actual_x2 | Actual_y2 | Predicted_x1 | Predicted_y1 | Predicted_x2 | Pred |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.90 | 0.48 | 0.96 | 0.54 | 0.770767 | 0.577944 | 0.829167 | [illegible] |
| **1** | 0.24 | 0.30 | 0.30 | 0.36 | 0.259497 | 0.290839 | 0.318175 | [illegible] |
| **2** | 0.42 | 0.18 | 0.48 | 0.24 | 0.441343 | 0.185062 | 0.500666 | [illegible] |
| **3** | 0.72 | 0.78 | 0.78 | 0.84 | 0.599064 | 0.768388 | 0.657564 | [illegible] |
| **4** | 0.48 | 0.36 | 0.54 | 0.42 | 0.513619 | 0.391467 | 0.569974 | [illegible] |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **251** | 0.18 | 0.24 | 0.24 | 0.30 | 0.176288 | 0.443364 | 0.233887 | [illegible] |
| **252** | 0.42 | 0.78 | 0.48 | 0.84 | 0.239806 | 0.727956 | 0.300995 | [illegible] |
| **253** | 0.54 | 0.72 | 0.60 | 0.78 | 0.534885 | 0.731584 | 0.601093 | [illegible] |
| **254** | 0.06 | 0.66 | 0.12 | 0.72 | 0.203096 | 0.643591 | 0.264384 | [illegible] |
| **255** | 0.36 | 0.12 | 0.42 | 0.18 | 0.342342 | 0.322337 | 0.402059 | [illegible] |

256 rows × 8 columns

In [9]:

```python
# Calculate evaluation metrics for each target variable separately
metrics_dict = {}

for target in targets:
    # Select actual and predicted values for the target variable
    actual_values = output_values[f'Actual_{target.strip()}']  # Remove ex
    predicted_values = output_values[f'Predicted_{target.strip()}']  # Ren

    # Calculate evaluation metrics
    mse = mean_squared_error(actual_values, predicted_values)
    mae = mean_absolute_error(actual_values, predicted_values)
    r2 = r2_score(actual_values, predicted_values)
    medae = median_absolute_error(actual_values, predicted_values)
    evs = explained_variance_score(actual_values, predicted_values)
    msle = mean_squared_log_error(actual_values, predicted_values)

    # Store metrics in a dictionary
    metrics_dict[target] = {
        'Mean Squared Error': mse,
        'Mean Absolute Error': mae,
        'R-squared': r2,
        'Median Absolute Error': medae,
        'Explained Variance Score': evs,
        'Mean Squared Logarithmic Error': msle
    }

# Print metrics for each target variable
for target, metrics in metrics_dict.items():
    print(f"Metrics for {target}:")
    for metric, value in metrics.items():
        print(f"{metric}: {value}")
    print()

```

```
Metrics for  x1:
Mean Squared Error: 0.0070447948405507 79
Mean Absolute Error: 0.0504333633023306
R-squared: 0.9079111785548918
Median Absolute Error: 0.032262734303693
Explained Variance Score: 0.9084450856416482
Mean Squared Logarithmic Error: 0.0037588897986350193

Metrics for  y1:
Mean Squared Error: 0.004092602760262016
Mean Absolute Error: 0.040247030885385025
R-squared: 0.9465019247024573
Median Absolute Error: 0.024628170571259753
Explained Variance Score: 0.9467704617769523
Mean Squared Logarithmic Error: 0.002297926889311344

Metrics for  x2:
Mean Squared Error: 0.007089687411497405
Mean Absolute Error: 0.05071536539577299
R-squared: 0.9073243475621254
Median Absolute Error: 0.03199182793718239
Explained Variance Score: 0.9078517608219387
```

```
Mean Squared Logarithmic Error: 0.0034589469456795955

Metrics for  y2:
Mean Squared Error: 0.004128613002806427
Mean Absolute Error: 0.040239218666960286
R-squared: 0.9460312025776938
Median Absolute Error: 0.026199999999999696
Explained Variance Score: 0.9462965342809088
Mean Squared Logarithmic Error: 0.0021139367687811085
```

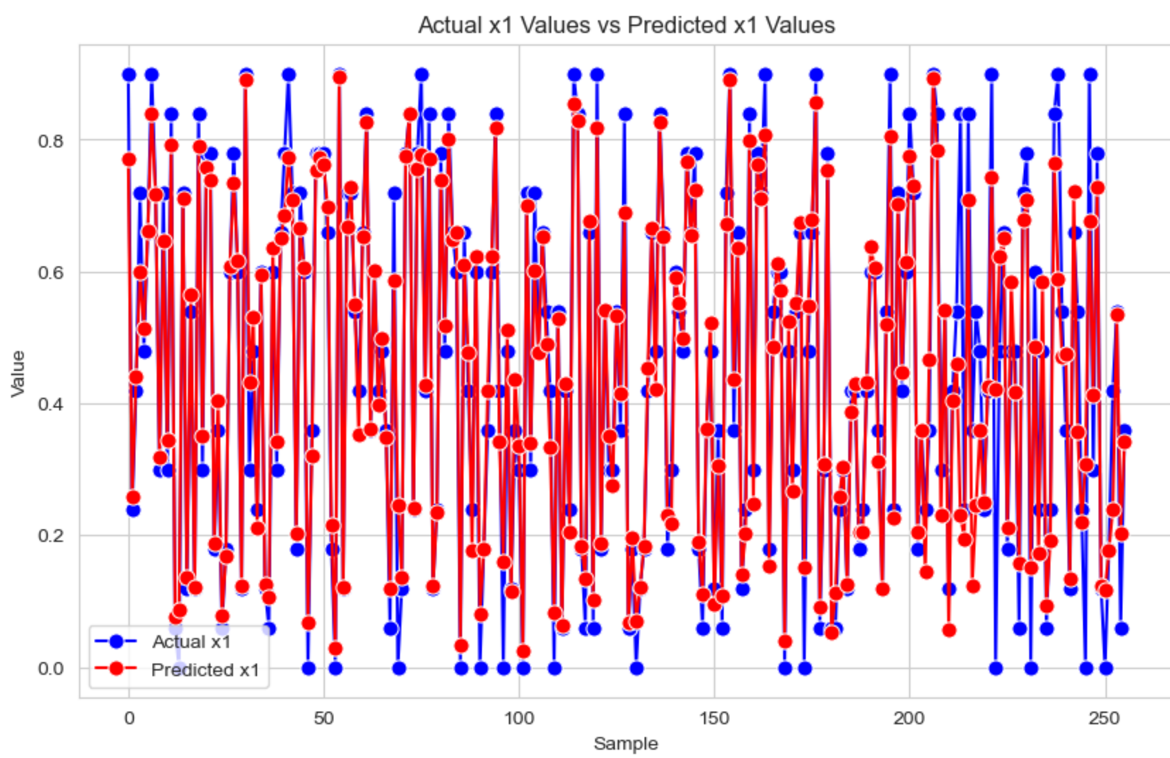
In [10]:

```python
# Specify the file path
file_path = r'D:\CV things\ML projects\MTP projects\combined_output_values

# Write the combined data to the specified file path
output_values.to_csv(file_path, index=False)

print(f"Combined data saved to '{file_path}'")
```

```
Combined data saved to 'D:\CV things\ML projects\MTP projects\combined_output
_values.csv'
```

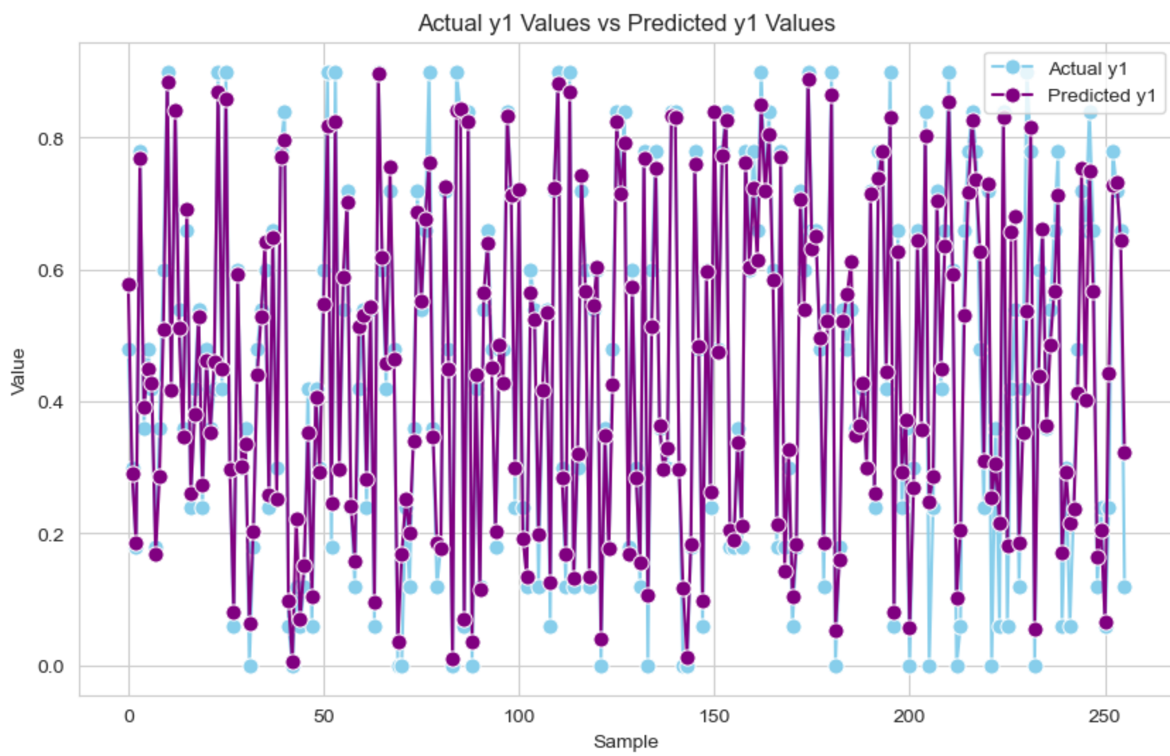
In [11]:

```
# Read the CSV file into a DataFrame
file_path = r'D:\CV things\ML projects\MTP projects\combined_output_values
output_values = pd.read_csv(file_path)

# Set the style for the plot
sns.set_style("whitegrid")

# Create a line plot for actual and predicted x1 values
plt.figure(figsize=(10, 6))

# Plot actual x1 values in blue
sns.lineplot(data=output_values['Actual_x1'], marker='o', color='blue', ma

# Plot predicted x1 values in red
sns.lineplot(data=output_values['Predicted_x1'], marker='o', color='red',

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual x1 Values vs Predicted x1 Values')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



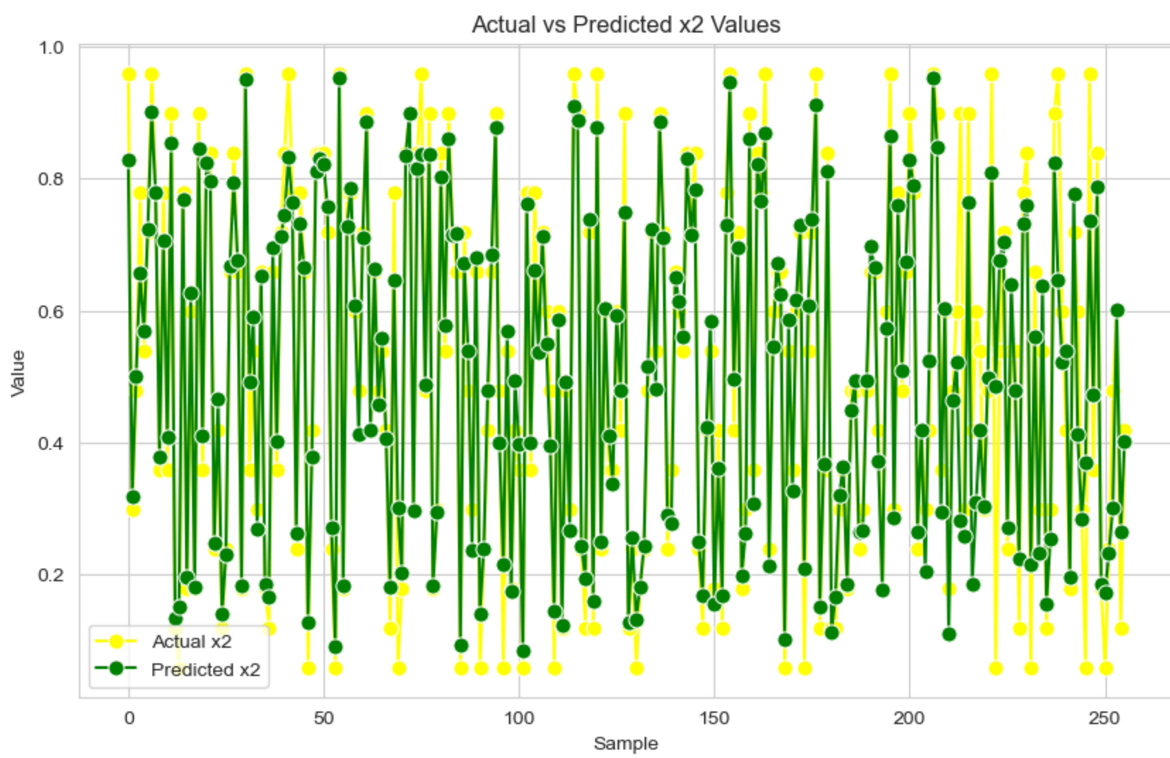
In [12]:

```
# Set the palette to 'bright'
sns.set_palette('deep')

# Create a line plot for actual and predicted y1 values
plt.figure(figsize=(10, 6))

# Plot actual y1 values in blue
sns.lineplot(data=output_values['Actual_y1'], marker='o', color='skyblue',

# Plot predicted y1 values in red
sns.lineplot(data=output_values['Predicted_y1'], marker='o', color='purple

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual y1 Values vs Predicted y1 Values')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



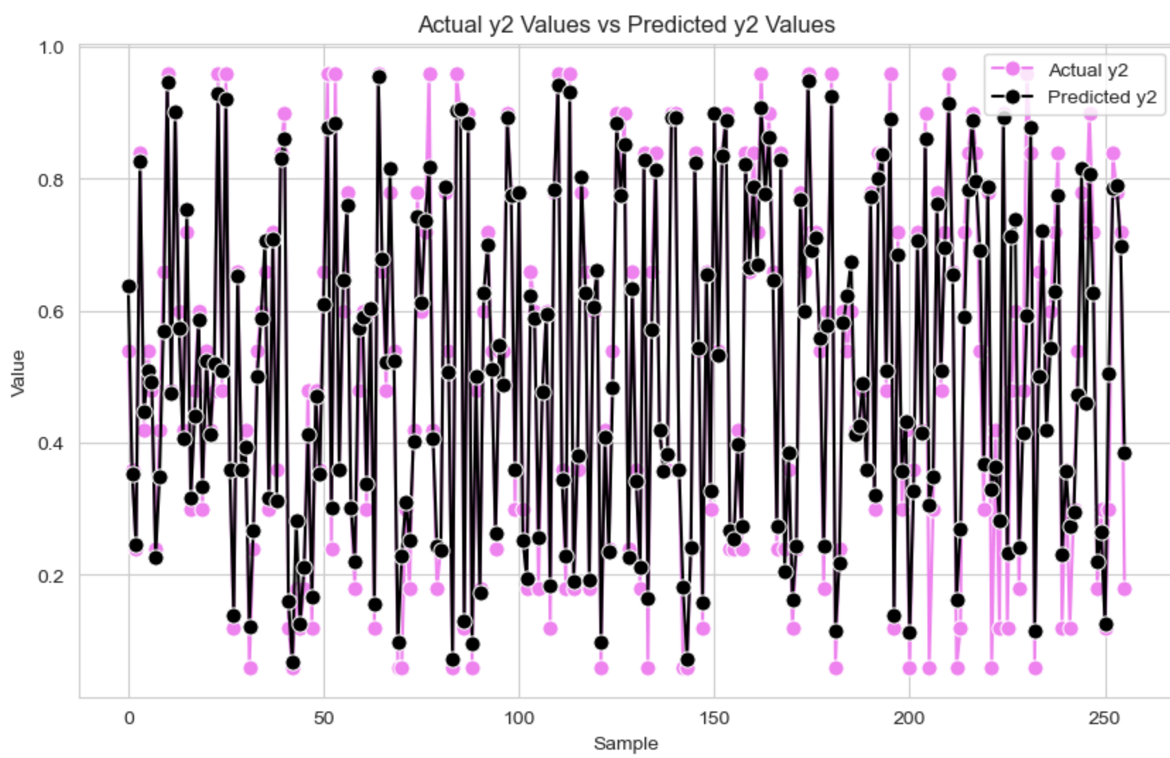
In [13]:

```python
# Create a line plot for actual and predicted x2 values
plt.figure(figsize=(10, 6))

# Plot actual x2 values in green
sns.lineplot(data=output_values['Actual_x2'], marker='o', color='yellow',

# Plot predicted x2 values in orange
sns.lineplot(data=output_values['Predicted_x2'], marker='o', color='green'

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual vs Predicted x2 Values')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



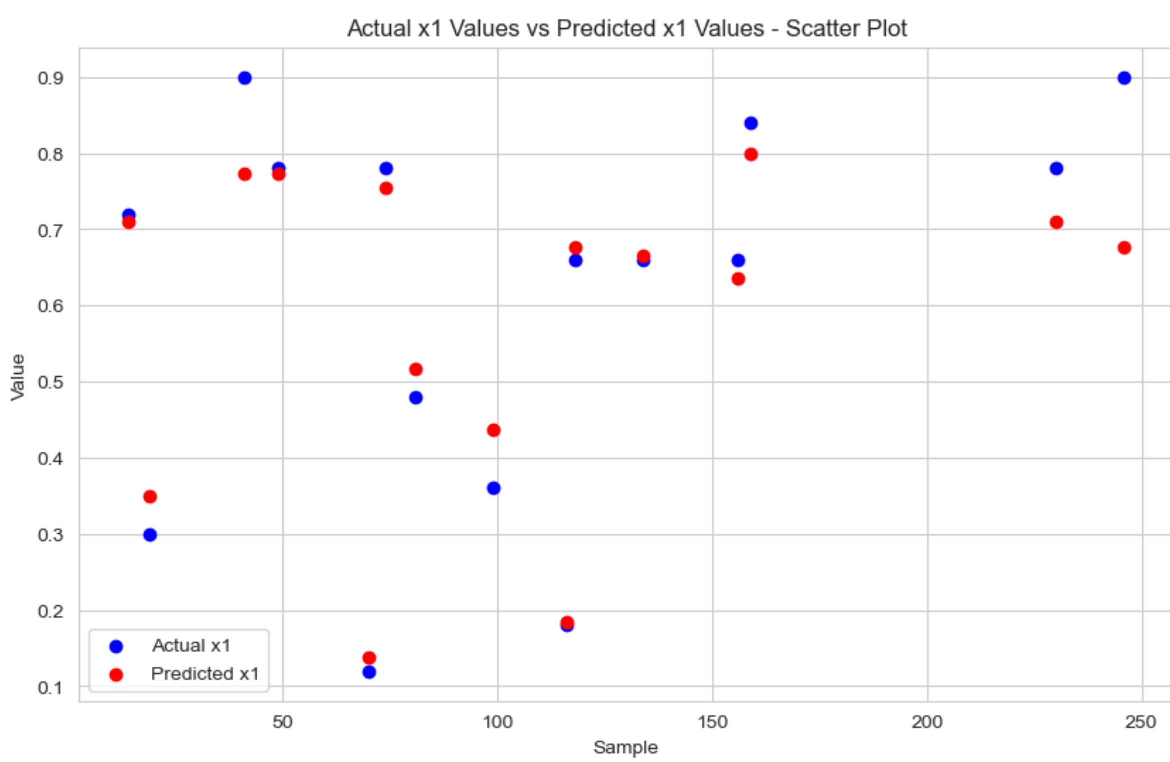
In [14]:

```python
# Create a line plot for actual and predicted y2 values
plt.figure(figsize=(10, 6))

# Plot actual y2 values in purple
sns.lineplot(data=output_values['Actual_y2'], marker='o', color='violet',

# Plot predicted y2 values in yellow
sns.lineplot(data=output_values['Predicted_y2'], marker='o', color='black'

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual y2 Values vs Predicted y2 Values')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



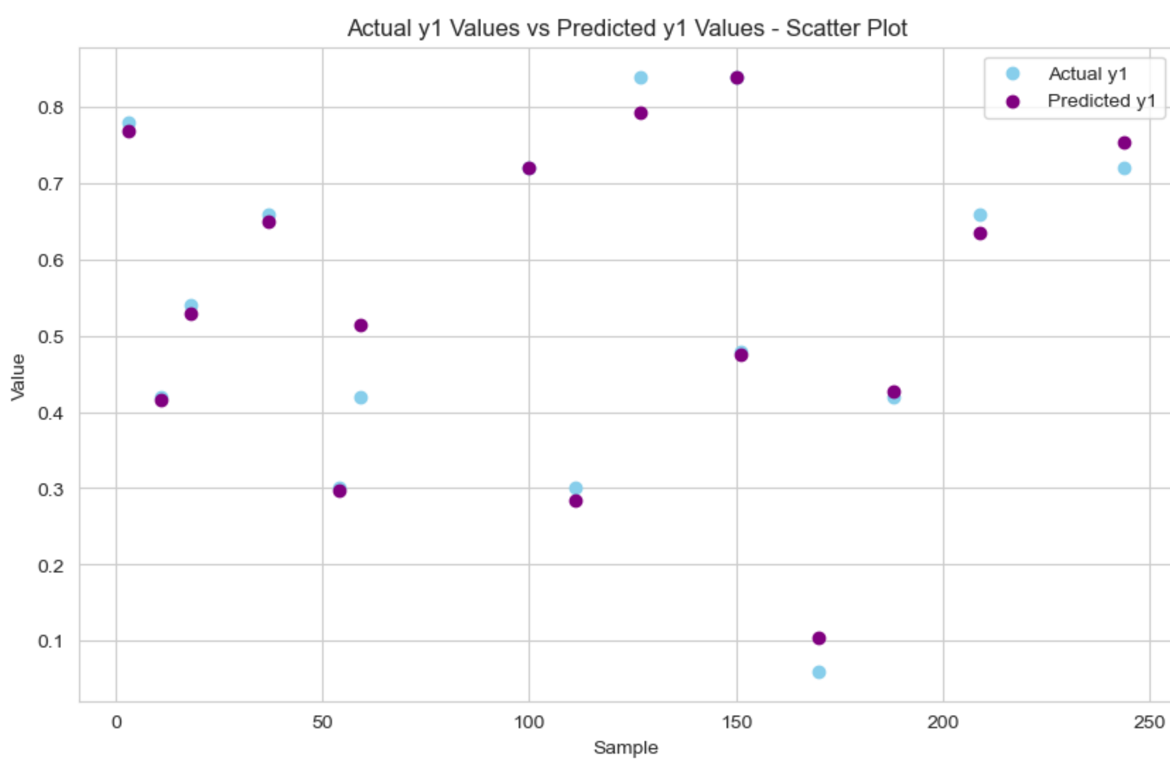
In [15]:

```
# Randomly select 100 rows from the DataFrame
sampled_data = output_values.sample(n=15)

# Create a scatter plot for actual and predicted x1 values
plt.figure(figsize=(10, 6))

# Plot actual x1 values in blue
plt.scatter(sampled_data.index, sampled_data['Actual_x1'], color='blue', l

# Plot predicted x1 values in red
plt.scatter(sampled_data.index, sampled_data['Predicted_x1'], color='red',

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual x1 Values vs Predicted x1 Values - Scatter Plot')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



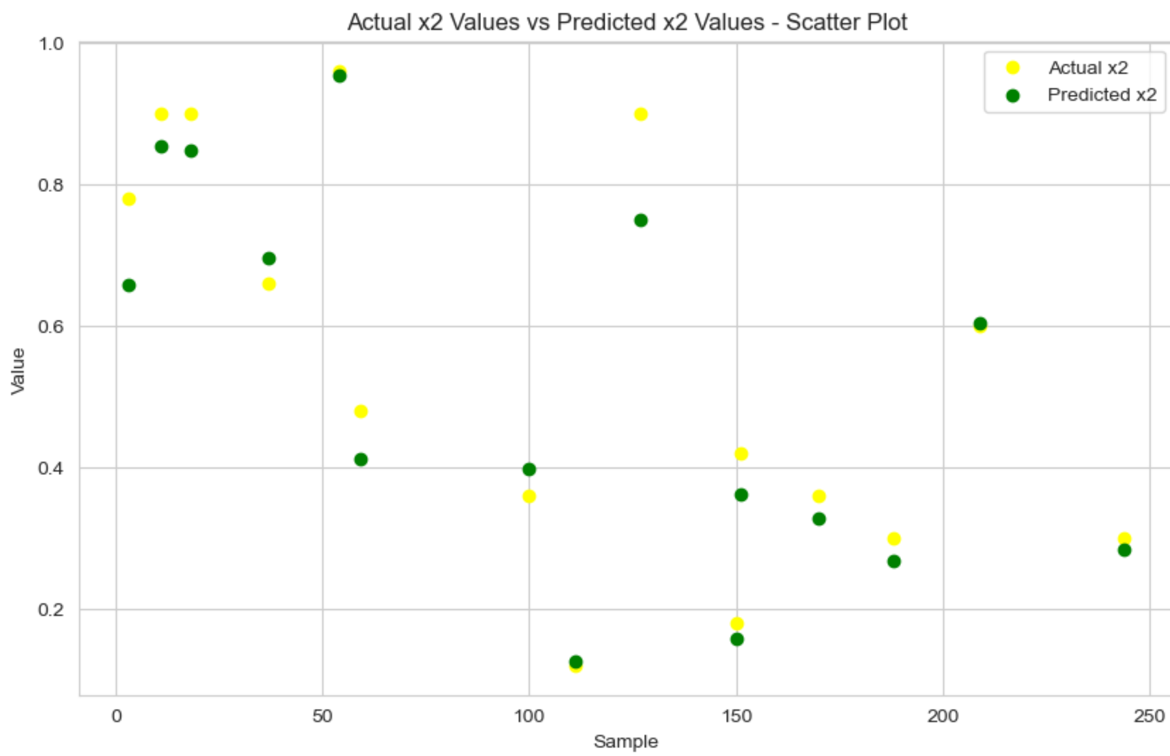
In [16]:

```
# Randomly select 15 rows from the DataFrame
sampled_data = output_values.sample(n=15, random_state=40)

# Create a scatter plot for actual and predicted y1 values
plt.figure(figsize=(10, 6))

# Plot actual y1 values in blue
plt.scatter(sampled_data.index, sampled_data['Actual_y1'], color='skyblue'

# Plot predicted y1 values in red
plt.scatter(sampled_data.index, sampled_data['Predicted_y1'], color='purpl

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual y1 Values vs Predicted y1 Values - Scatter Plot')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



In [17]:

```
# Create a scatter plot for actual and predicted x2 values
plt.figure(figsize=(10, 6))

# Plot actual x2 values in green
plt.scatter(sampled_data.index, sampled_data['Actual_x2'], color='yellow',

# Plot predicted x2 values in orange
plt.scatter(sampled_data.index, sampled_data['Predicted_x2'], color='greer

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual x2 Values vs Predicted x2 Values - Scatter Plot')

# Add legend
plt.legend()

# Show the plot
plt.show()
```

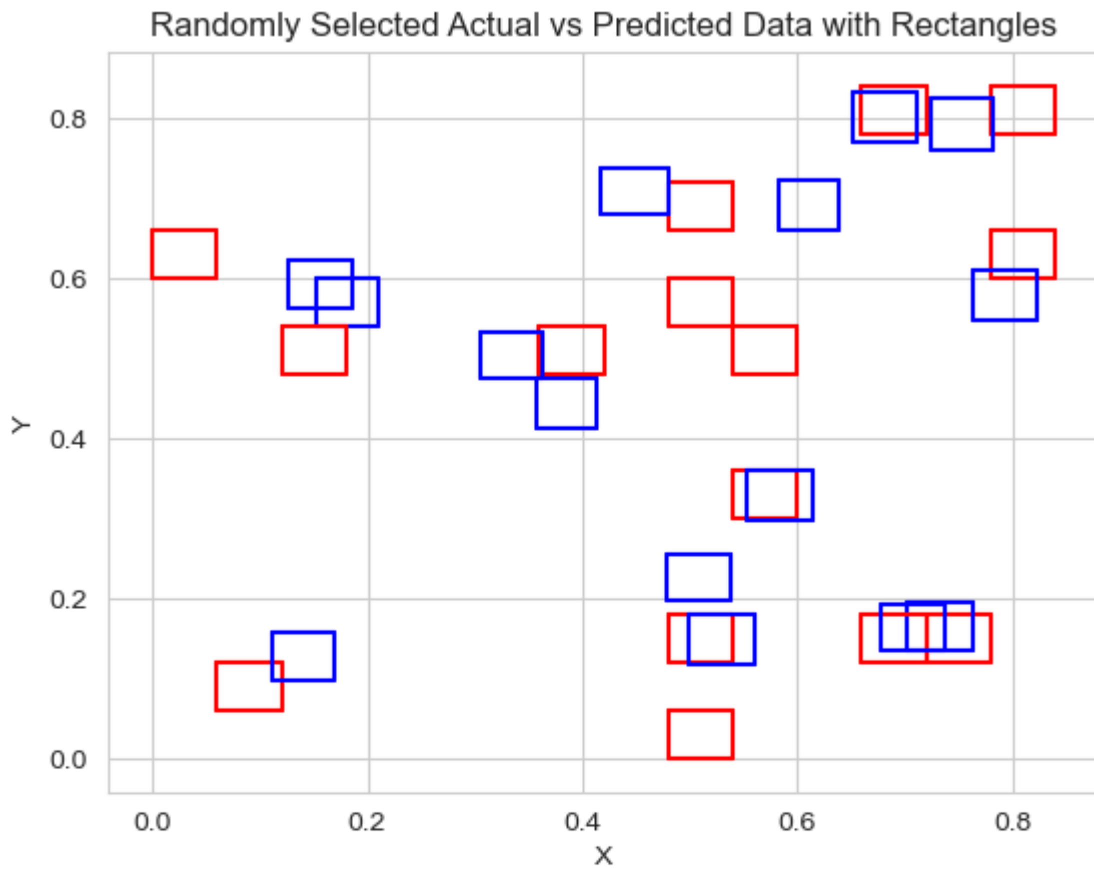


In [18]:

```
# Create a scatter plot for actual and predicted y2 values
plt.figure(figsize=(10, 6))

# Plot actual y2 values in purple
plt.scatter(sampled_data.index, sampled_data['Actual_y2'], color='violet',

# Plot predicted y2 values in yellow
plt.scatter(sampled_data.index, sampled_data['Predicted_y2'], color='black

# Add labels and title
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('Actual y2 Values vs Predicted y2 Values - Scatter Plot')

# Add legend
plt.legend()

# Show the plot
plt.show()
```

In [21]:

```
# Read the sample data from the CSV file
file_path = r'D:\CV things\ML projects\MTP projects\combined_output_values
data = pd.read_csv(file_path)

# Select 5 random rows from the data
random_indices = random.sample(range(len(data)),15)
random_data = data.iloc[random_indices]

# Extracting data from the DataFrame
for i, row in random_data.iterrows():
    actual_x1, actual_y1, actual_x2, actual_y2 = row[['Actual_x1', 'Actual
    predicted_x1, predicted_y1, predicted_x2, predicted_y2 = row[['Predict

    # Plot rectangles
    plt.plot([actual_x1, actual_x2], [actual_y1, actual_y1], color='red')
    plt.plot([actual_x1, actual_x2], [actual_y2, actual_y2], color='red')
    plt.plot([actual_x1, actual_x1], [actual_y1, actual_y2], color='red')
    plt.plot([actual_x2, actual_x2], [actual_y1, actual_y2], color='red')

    plt.plot([predicted_x1, predicted_x2], [predicted_y1, predicted_y1], c
    plt.plot([predicted_x1, predicted_x2], [predicted_y2, predicted_y2], c
    plt.plot([predicted_x1, predicted_x1], [predicted_y1, predicted_y2], c
    plt.plot([predicted_x2, predicted_x2], [predicted_y1, predicted_y2], c

plt.xlabel('X')
plt.ylabel('Y')
plt.title('Randomly Selected Actual vs Predicted Data with Rectangles')

#Show the plot
plt.show()
```


Randomly Selected Actual vs Predicted Data with Rectangles
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
X