In [1]:

```
# Importing Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sc
from catboost import CatBoostRegressor
from sklearn.linear_model import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

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

Out[2]:

	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]:

```
# 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 [4]:

```python
# Initialize dictionaries to store actual and predicted values for both tr
actual_predicted_train = {}
actual_predicted_test = {}
ensemble_models = {}

# Train CatBoost models for each target variable
for target in targets:
    y = data[target]
    X = data[features]  # Define features
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

    # Initialize CatBoostRegressor
    model = CatBoostRegressor(iterations=1000,
                              learning_rate=0.1,
                              depth=6,
                              loss_function='RMSE',
                              verbose=100)

    # Fit the model
    model.fit(X_train, y_train, eval_set=(X_test, y_test), use_best_model=

    # Save the actual and predicted values for the training set
    y_pred_train = model.predict(X_train)
    actual_predicted_train[target] = (y_train.values, y_pred_train)

    # Save the actual and predicted values for the test set
    y_pred_test = model.predict(X_test)
    actual_predicted_test[target] = (y_test.values, y_pred_test)

    # Evaluate the model on the training set
    mse_train, mae_train, r2_train = mean_squared_error(y_train, y_pred_tr

    print(f"Metrics for CatBoost model on {target} (Train set):")
    print("Mean Squared Error:", mse_train)
    print("Mean Absolute Error:", mae_train)
    print("R-squared:", r2_train)
    print()

    # Evaluate the model on the test set
    mse_test, mae_test, r2_test = mean_squared_error(y_test, y_pred_test),

    print(f"Metrics for CatBoost model on {target} (Test set):")
    print("Mean Squared Error:", mse_test)
    print("Mean Absolute Error:", mae_test)
    print("R-squared:", r2_test)
    print()

    # Store the model in the ensemble
    ensemble_models[target] = model
```

```
Metrics for CatBoost model on  x1 (Train set):
Mean Squared Error: 1.5197189441127308e-07
Mean Absolute Error: 0.00030298264455803057
R-squared: 0.9999980156114037

Metrics for CatBoost model on  x1 (Test set):
```

```
Mean Squared Error: 0.010245189434453796
Mean Absolute Error: 0.0787903053400416
R-squared: 0.8647388803407086

Metrics for CatBoost model on  y1 (Train set):
Mean Squared Error: 1.8121454387838993e-07
Mean Absolute Error: 0.0003236058109857457
R-squared: 0.9999975436430879

Metrics for CatBoost model on  y1 (Test set):
Mean Squared Error: 0.005991801704558089
Mean Absolute Error: 0.0617784487205046
R-squared: 0.9311511926956909

Metrics for CatBoost model on  x2 (Train set):
Mean Squared Error: 1.513778336459875e-07
Mean Absolute Error: 0.00030173945506148
R-squared: 0.999998023368413

Metrics for CatBoost model on  x2 (Test set):
Mean Squared Error: 0.010234844920400945
Mean Absolute Error: 0.07874958826521158
R-squared: 0.8648754527840071

Metrics for CatBoost model on  y2 (Train set):
Mean Squared Error: 1.5151530220910657e-07
Mean Absolute Error: 0.0002974837002828465
R-squared: 0.9999979462152876

Metrics for CatBoost model on  y2 (Test set):
Mean Squared Error: 0.006000290740568547
Mean Absolute Error: 0.061829016255648545
R-squared: 0.9310536494135028
```

In [5]:

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sc
import numpy as np

# Initialize dictionaries to store actual and predicted values for the ent
actual_predicted_all = {}

# Aggregate actual and predicted values for all target variables
for target in targets:
    # Combine actual and predicted values for the training set
    y_train_actual, y_train_predicted = actual_predicted_train[target]
    # Combine actual and predicted values for the test set
    y_test_actual, y_test_predicted = actual_predicted_test[target]

    # Combine actual and predicted values for the entire dataset
    y_actual = np.concatenate([y_train_actual, y_test_actual])
    y_predicted = np.concatenate([y_train_predicted, y_test_predicted])

    # Store the aggregated actual and predicted values
    actual_predicted_all[target] = (y_actual, y_predicted)

# Calculate evaluation metrics for the entire dataset
for target, (y_actual, y_predicted) in actual_predicted_all.items():
    mse = mean_squared_error(y_actual, y_predicted)
    mae = mean_absolute_error(y_actual, y_predicted)
    r2 = r2_score(y_actual, y_predicted)

    print(f"Metrics for entire dataset ({target}):")
    print("Mean Squared Error:", mse)
    print("Mean Absolute Error:", mae)
    print("R-squared:", r2)
    print()

```

```
Metrics for entire dataset ( x1):
Mean Squared Error: 0.0020811752064767864
Mean Absolute Error: 0.01624572006707813
R-squared: 0.9727950953401727

Metrics for entire dataset ( y1):
Mean Squared Error: 0.0012172291265780146
Mean Absolute Error: 0.012806620776981761
R-squared: 0.984088508149307

Metrics for entire dataset ( x2):
Mean Squared Error: 0.0020790735036676283
Mean Absolute Error: 0.01623645874462322
R-squared: 0.9728225685795081

Metrics for entire dataset ( y2):
Mean Squared Error: 0.0012189297954344341
Mean Absolute Error: 0.012796076250591504
R-squared: 0.9840662771838636
```

In [6]:

```
# Combine actual and predicted values for training set into a DataFrame
train_dataframes = []
for target, (y_actual, y_pred) in actual_predicted_train.items():
    train_df = pd.DataFrame({f'Actual_{target}': y_actual, f'Predicted_{ta
    train_dataframes.append(train_df)
```

In [7]:

```
# Combine actual and predicted values for test set into a DataFrame
test_dataframes = []
for target, (y_actual, y_pred) in actual_predicted_test.items():
    test_df = pd.DataFrame({f'Actual_{target}': y_actual, f'Predicted_{tar
    test_dataframes.append(test_df)
```

In [8]:

```
# Concatenate DataFrames for both training and test sets
combined_df = pd.concat([pd.concat(train_dataframes, axis=1), pd.concat(te

# Add the 'Sl No' column from the original dataset
combined_df.insert(0, 'Sl No', data['Sl No'])
```

In [9]:

```
combined_df
```

Out[9]:

	Sl No	Actual_x1	Predicted_x1	Actual_y1	Predicted_y1	Actual_x2	Predicted_x2	Actual_y2	Predicted_y2
0	1	0.90	0.900229	0.48	0.479786	0.96	0.960149	0.54	0.539756
1	2	0.24	0.240088	0.30	0.299618	0.30	0.300133	0.36	0.359747
2	3	0.42	0.420173	0.18	0.180301	0.48	0.480215	0.24	0.240258
3	4	0.72	0.719037	0.78	0.779261	0.78	0.779046	0.84	0.839042
4	5	0.48	0.480053	0.36	0.360038	0.54	0.539850	0.42	0.419905
...	...	...	...	...	...	...	...	...	...
251	252	0.18	0.204097	0.24	0.431293	0.24	0.263965	0.30	0.491159
252	253	0.42	0.282864	0.78	0.787400	0.48	0.342530	0.84	0.847805
253	254	0.54	0.494661	0.72	0.739592	0.60	0.554795	0.78	0.799166
254	255	0.06	0.125887	0.66	0.665836	0.12	0.186307	0.72	0.727134
255	256	0.36	0.378070	0.12	0.247206	0.42	0.438438	0.18	0.308482

256 rows × 9 columns

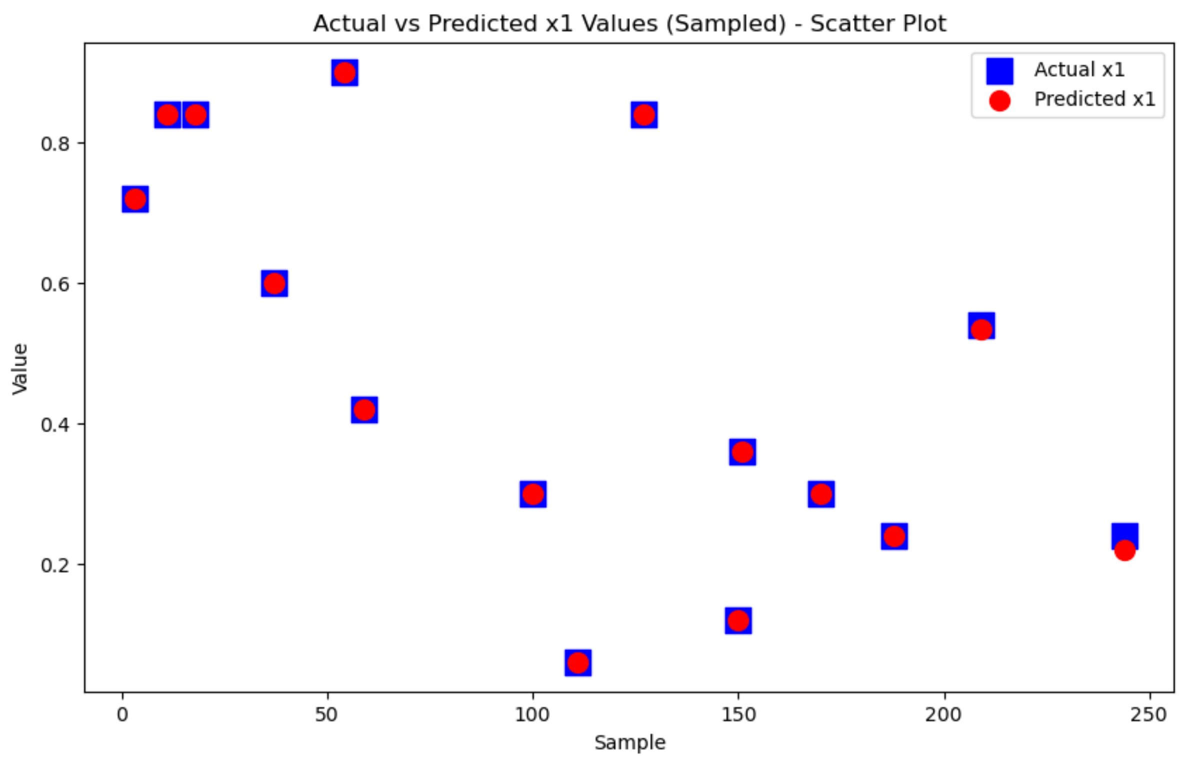
In [10]:

```
# Save the sorted combined DataFrame to an Excel file
output_file_path = r"D:\MTP\coding\last_output.xlsx"
combined_df.to_excel(output_file_path, index=False)

print(f"Last output saved to: {output_file_path}")
```

```
Last output saved to: D:\MTP\coding\last_output.xlsx
```

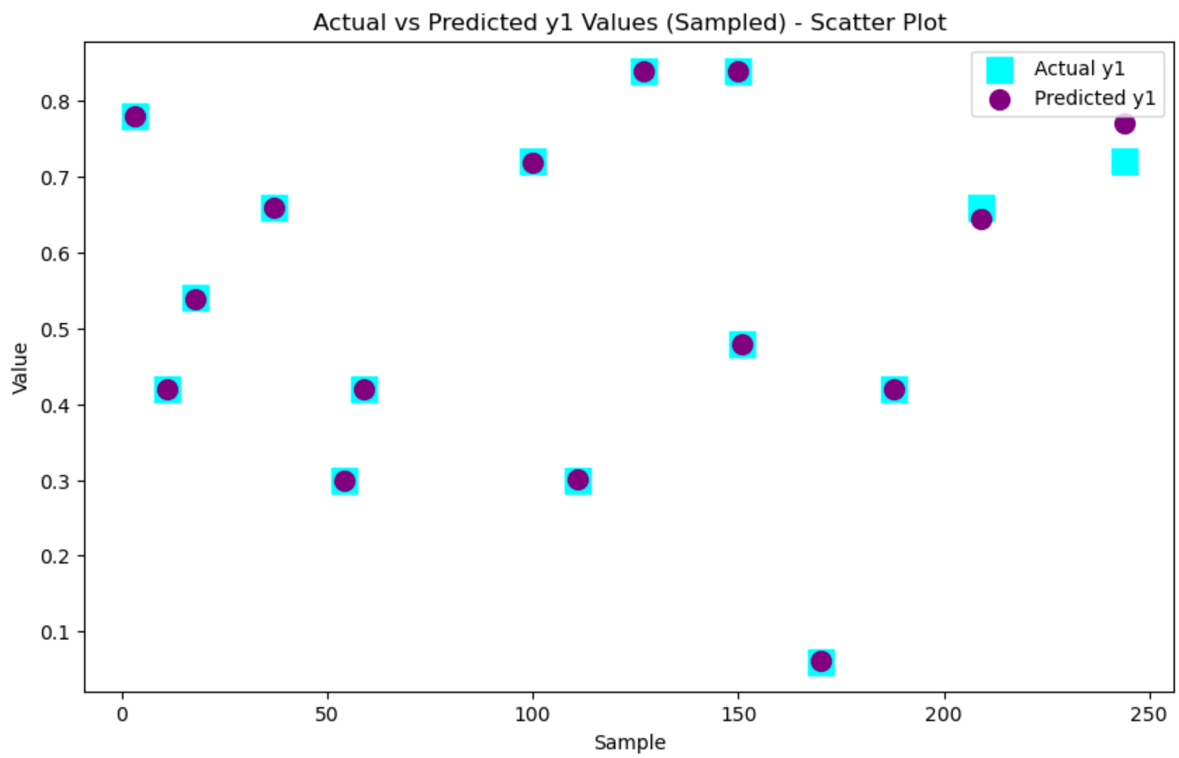
In [11]:

```
# Randomly select 100 rows from the DataFrame
sampled_data = combined_df.sample(n=15, random_state=40)

# 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',

# 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 vs Predicted x1 Values (Sampled) - Scatter Plot')

# Add legend
plt.legend()

# Show the plot
plt.show()

```

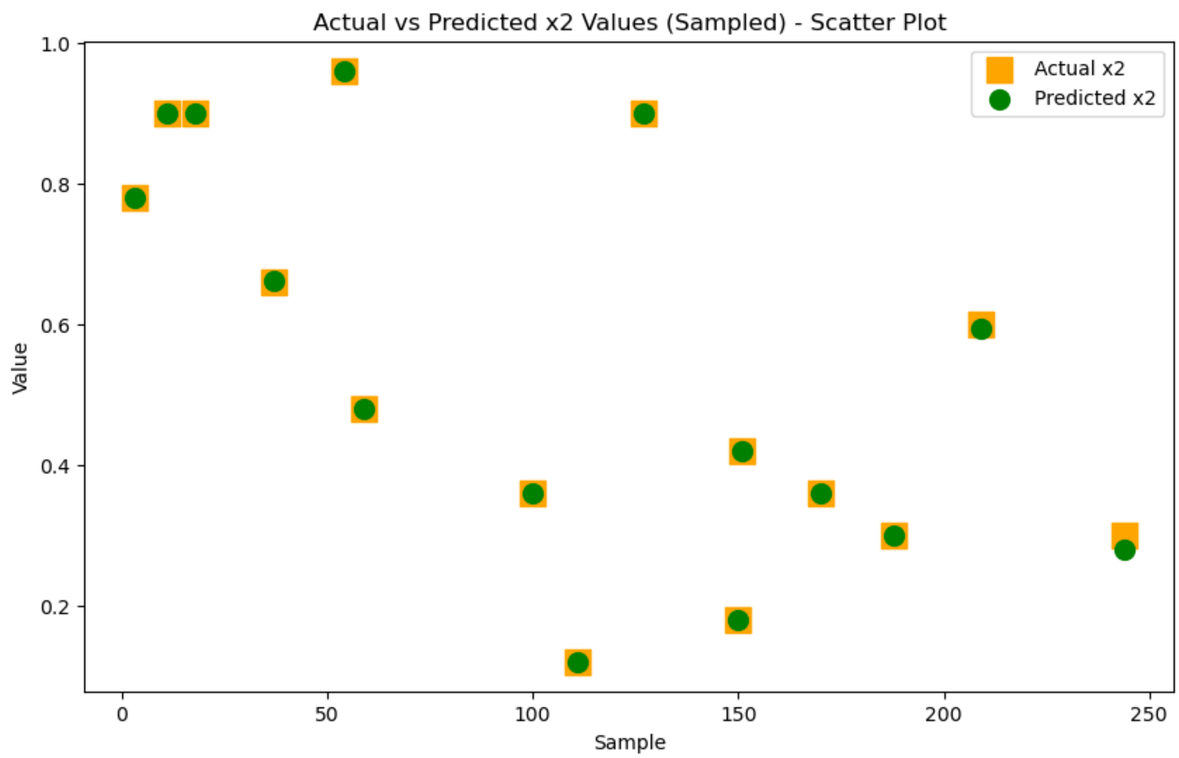


In [12]:

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

# Plot actual y1 values in green
plt.scatter(sampled_data.index, sampled_data['Actual_ y1'], color='cyan',

# Plot predicted y1 values in purple
plt.scatter(sampled_data.index, sampled_data['Predicted_ y1'], color='purp

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

# Add legend
plt.legend()

# Show the plot
plt.show()

```

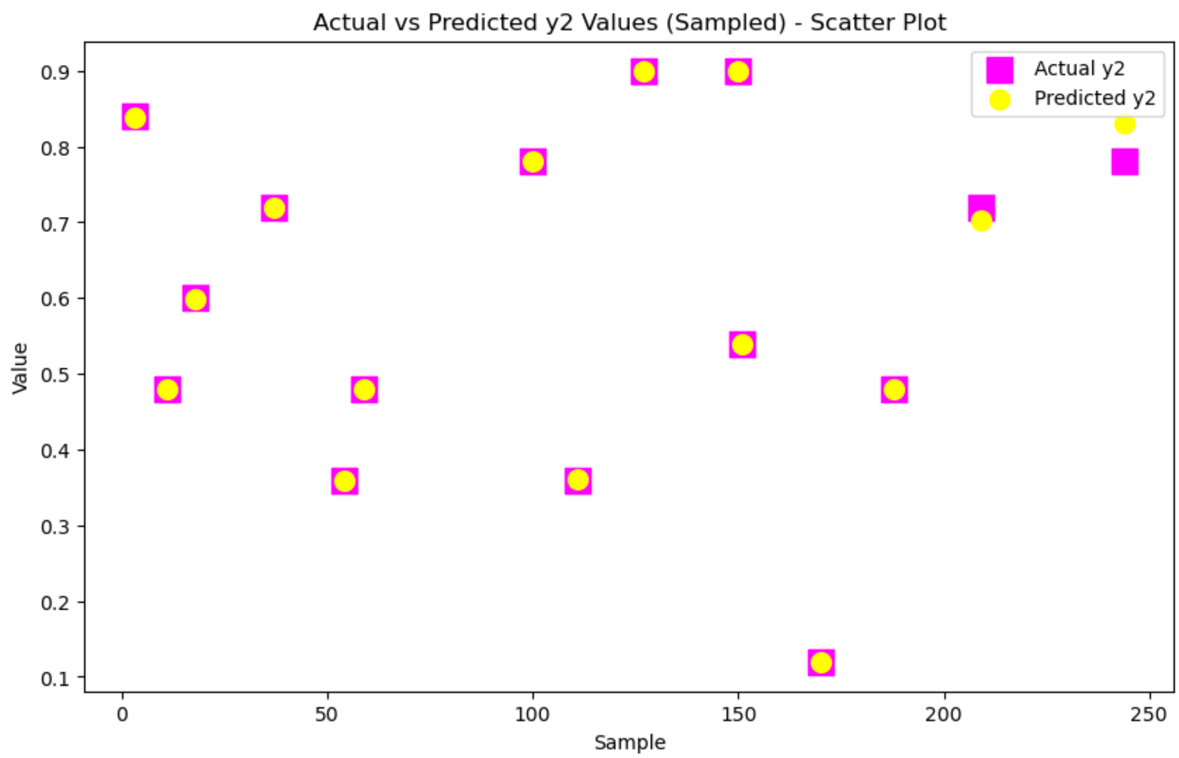


In [13]:
```
# Create a scatter plot for actual and predicted x2 values
plt.figure(figsize=(10, 6))

# Plot actual x2 values in orange
plt.scatter(sampled_data.index, sampled_data['Actual_ x2'], color='orange'

# Plot predicted x2 values in cyan
plt.scatter(sampled_data.index, sampled_data['Predicted_ x2'], color='gree

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

# Add legend
plt.legend()

# Show the plot
plt.show()

```



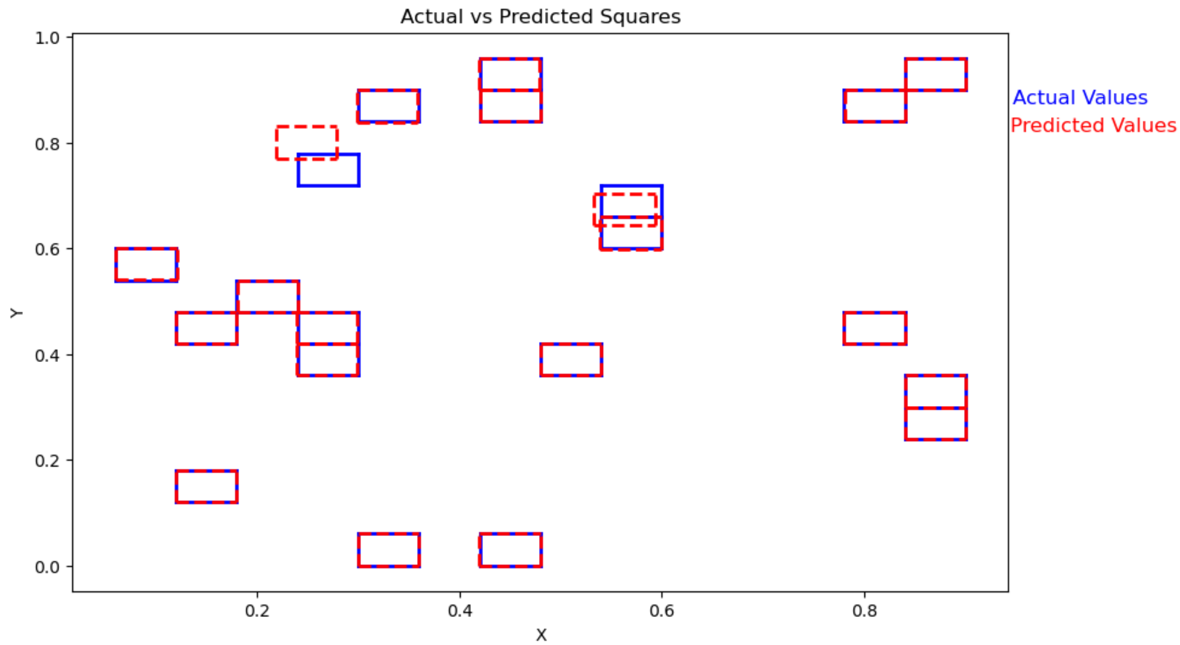
In [14]:

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

# Plot actual y2 values in magenta
plt.scatter(sampled_data.index, sampled_data['Actual_ y2'], color='magenta

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

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

# Add legend
plt.legend()

# Show the plot
plt.show()

```



In [27]:

```
import matplotlib.pyplot as plt

# Sampled data containing actual and predicted values
sampled_data = combined_df.sample(n=20)

# Create a figure and axis object
fig, ax = plt.subplots(figsize=(10, 6))

# Plot actual squares
for i, row in sampled_data.iterrows():
    actual_x1, actual_y1 = row['Actual_ x1'], row['Actual_ y1']
    actual_x2, actual_y2 = row['Actual_ x2'], row['Actual_ y2']

    # Plot actual square with a solid line
    ax.plot([actual_x1, actual_x2], [actual_y1, actual_y1], color='blue',
    ax.plot([actual_x1, actual_x2], [actual_y2, actual_y2], color='blue',
    ax.plot([actual_x1, actual_x1], [actual_y1, actual_y2], color='blue',
    ax.plot([actual_x2, actual_x2], [actual_y1, actual_y2], color='blue',

# Plot predicted squares
for i, row in sampled_data.iterrows():
    predicted_x1, predicted_y1 = row['Predicted_ x1'], row['Predicted_ y1'
    predicted_x2, predicted_y2 = row['Predicted_ x2'], row['Predicted_ y2'

    # Plot predicted square with a dashed line
    ax.plot([predicted_x1, predicted_x2], [predicted_y1, predicted_y1], co
    ax.plot([predicted_x1, predicted_x2], [predicted_y2, predicted_y2], co
    ax.plot([predicted_x1, predicted_x1], [predicted_y1, predicted_y2], co
    ax.plot([predicted_x2, predicted_x2], [predicted_y1, predicted_y2], co

# Add labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('Actual vs Predicted Squares')

# Add labels indicating actual and predicted values
ax.text(1.15, 0.9, 'Actual Values', verticalalignment='top', horizontalali
ax.text(1.18, 0.85, 'Predicted Values', verticalalignment='top', horizonta
# Show the plot
plt.show()
```


Actual vs Predicted Squares
Actual Values
Predicted Values
Y
X
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8

In [ ]:
```
1
```