Task: Data Simulation and DataFrame Creation

1. **Set Seed for Reproducibility**:
   - Use `np.random.seed()` to ensure that the random number generation is consistent every time the code is run. This will make the results reproducible.

2. **Specify Simulation Parameters**:
   - Set the sample size (n) to 750.
   - Set the standard deviation (σ_x) for the variables X_1 and X_2 to 1.
   - Define the correlation (ρ) between X_1 and X_2 as 0.7.

3. **Generate Normally Distributed Variables (X_1 and X_2)**:
   - Create two variables, X_1 and X_2, that follow a **joint normal distribution** with:
     - Mean of 0 for both variables.
     - Standard deviation of (σ_x) for both variables.
     - Correlation of (ρ) between them.
   - Use a covariance matrix to define the relationship between X_1 and X_2, and generate the values using `np.random.multivariate_normal()`.

4. **Generate Categorical Variable (X_3)**:
   - Create a categorical variable X_3 that can take the values `'A'`, `'B'`, and `'C'`.
   - Assign probabilities of 0.4, 0.3, and 0.3 to `'A'`, `'B'`, and `'C'`, respectively.
   - Use `np.random.choice()` to generate 750 random samples of X_3 according to the specified probabilities.

5. **Combine into a DataFrame**:
   - Create a pandas DataFrame called `df_simulation` that contains the generated variables X_1, X_2, and X_3 as columns.

Task: Simulating the Response Variable Y

1. **Set Parameters for the Simulation of Y**:
   - Define the following parameters:
     - Intercept ($\beta_0 = 1.0$)
     - Coefficient for X_1 ($\beta_1 = 0.4$)
     - Coefficient for X_2 ($\beta_2 = 0.4$)

- Coefficient for X_1^2 ($\beta_3 = 0.4$)
- Coefficient for $I(X_3 = \text{'B'})$ ($\beta_4 = 0.4$)
- Coefficient for $I(X_3 = \text{'C'})$ ($\beta_5 = 0.6$)
- Coefficient for interaction term $X_2 \times I(X_3 = \text{'B'})$ ($\beta_6 = 0.5$)
- Coefficient for interaction term $X_2 \times I(X_3 = \text{'C'})$ ($\beta_7 = 0.7$)
- Standard deviation for the error term ϵ ($\sigma_e = 1$)

2. **Generate Binary Indicators for X_3**:
   - Generate binary indicator $I(X_3 = \text{'B'})$ for when $X_3 = \text{'B'}$.
   - Generate binary indicator $I(X_3 = \text{'C'})$ for when $X_3 = \text{'C'}$.
3. **Simulate the Error Term ϵ**:
   - Generate $\epsilon \sim N(0, \sigma_e)$, where the standard deviation of the error term is $\sigma_e = 1$, and the size of ϵ matches the sample size n.
4. **Compute the Response Variable Y**:
   - Compute Y based on the following formula:

$$
\begin{aligned}
Y = {} & \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 I(X_3 = \text{'B'}) + \beta_5 I(X_3 = \text{'C'}) \\
& + \beta_6 X_2 \cdot I(X_3 = \text{'B'}) + \beta_7 X_2 \cdot I(X_3 = \text{'C'}) + \epsilon
\end{aligned}
$$

5. **Add the Response Variable Y to the DataFrame**:
   - Add the simulated Y as a new column to the DataFrame `df_simulation`.

Task: Detect Nonlinearity and Interaction Using Plots

1. **Use plots to detect the presence of nonlinearity and interaction**:
   - Generate relevant plots to visually assess the relationship between the predictors and the response variable Y.
   - Specifically, look for:
     - Nonlinearity: Are there patterns in the plots of Y against predictors that suggest a nonlinear relationship?
     - Interaction: Are there indications of interaction effects between X_1, X_2, and X_3?
2. **Comment on the findings**:
   - Based on the plots, provide a brief comment on whether the presence of nonlinearity or interaction effects is obvious.
   - Explain any visual cues that lead you to your conclusion.

Task: Model Comparison and Evaluation

1. **Consider three models**:
   - **Model 1**: The model that uses the correct set of predictors.
   - **Model 2 (Reduced)**: This model uses the following set of predictors:

     $X =$ `df_simulation[['X1', 'X2', 'I_X3_B', 'I_X3_C']]`

   - **Model 3 (Large)**: This model includes more predictors:

     $X =$ `df_simulation[['X1', 'X2', 'X1_squared', 'X2_squared', 'I_X3_B', 'I_X3_C',`

     `'X2_I_X3_B', 'X2_I_X3_C', 'X1_I_X3_B', 'X1_I_X3_C']]`

2. **Data Split**:
   - Before fitting the models, split the data into training and testing sets, using $\frac{2}{3}$ for training and $\frac{1}{3}$ for testing.

3. **Fit All Three Models**:
   - Run all three models using linear regression on the training data.
   - For each model, generate diagnostic plots, including residual plots, to assess model fit.

4. **Model Diagnostics and Assumptions**:
   - Examine the residuals of each model.
   - Comment on whether the assumptions of linear regression (linearity, homoscedasticity, normality of residuals, etc.) are met for each model.

5. **Model Comparison**:
   - Compare the models based on:
     - Akaike Information Criterion (AIC)
     - Bayesian Information Criterion (BIC)
     - Mean Squared Error (MSE) from the training data
     - Mean Squared Error (MSE) from the testing data
   - Based on these metrics, determine which model performs the best.

6. **Conclude**:
   - Choose the best model based on AIC, BIC, training MSE, and testing MSE.

Note: Explanation of Terms

- **Term:** `I_X3_B`:
  - This term represents a **binary indicator variable** based on the value of the categorical variable X_3.
  - Specifically, `I_X3_B` takes the value:
    * 1 if $X_3 =$ 'B', indicating that X_3 is equal to the category `'B'`.
    * 0 otherwise (if $X_3 \neq$ 'B').
  - Indicator variables like this are used in regression models to include categorical variables (e.g., X_3 which can take values `'A'`, `'B'`, or `'C'`).
- **Term:** `X2_I_X3_B`:
  - This is an **interaction term** between the continuous variable X_2 and the binary indicator variable `I_X3_B`.
  - The interaction is calculated as:

$$\texttt{X2\_I\_X3\_B} = X_2 \times I(X_3 = \text{'B'})$$

  - This term captures the effect of X_2 on the outcome variable, but **only when** $X_3 =$ **'B'**. When $X_3 \neq$ 'B', the interaction term is 0.
  - Interaction terms are used to model how the relationship between X_2 and the outcome changes depending on the value of X_3.
- **Purpose**:
  - `I_X3_B`: Allows us to model the specific effect of the category `'B'` in X_3 on the outcome.
  - `X2_I_X3_B`: Allows us to capture how the relationship between X_2 and the outcome is different when $X_3 =$ 'B'.

Task: Sequential Floating Forward Selection (SFFS)

1. **Fit a Linear Regression Model**:
   - Initialize a linear regression model large as defined above using the `LinearRegression()` function.
2. **Perform Sequential Floating Forward Selection (SFFS)**:
   - Use Sequential Floating Forward Selection (SFFS) to select the best set of features for predicting Y.
   - Set up SFFS with the following specifications:
     - **Model**: Use the initialized linear regression model.

  - **Selection Method**: Forward selection, with floating enabled to allow backward elimination.
  - **Scoring Metric**: Use the R^2 score as the evaluation metric for feature selection.
  - **Cross-Validation**: Perform 10-fold cross-validation (`cv=10`) to evaluate the performance of the selected features.
- Fit the SFFS model on the dataset X and response Y.

3. **Extract and Display the Selected Features**:
   - Extract the names of the selected features after the Sequential Floating Forward Selection process.
   - Print the selected features to confirm which predictors were chosen for the final model.

4. **Use a Different Scoring Method**:
   - Modify the SFFS process to use a different scoring metric (e.g., `'neg_mean_squared_error'`, `'accuracy'`, etc.).
   - Fit the model using this alternative scoring metric.
   - Compare the set of features selected by this alternative scoring method to the features selected when using the R^2 metric.

5. **Discuss the Selected Models**:
   - Compare the results of using different scoring metrics. Did the choice of scoring method result in a different set of selected features?
   - Discuss whether the alternative scoring method produced a better or worse model compared to using R^2 as the evaluation metric.
   - Analyze the set of selected predictors for each model and compare it to the known correct set of predictors used in the simulation.
   - Discuss whether any important variables were omitted or if unnecessary variables were included, based on each scoring method.

Hint: You can use the following Python code to perform Sequential Floating Forward Selection (SFFS):

```
# Linear regression model
lr = LinearRegression()

# Sequential Floating Forward Selection
sfs = SFS(lr,
          k_features='best',
          forward=True,
          floating=True,  # Enable floating
          scoring='r2',
```

```
          cv=10)

sfs = sfs.fit(X, Y)

# Selected features
selected_features = sfs.k_feature_names_
print(f"Selected features: {selected_features}")
```

Task: Stepwise Selection Using AIC

1. **Perform Stepwise Selection Based on AIC**:
   - Implement stepwise selection using the Akaike Information Criterion (AIC) as the selection criterion.
   - The stepwise process should include:
     - **Forward Step**: Iteratively add predictors that minimize the AIC.
     - **Backward Step**: Remove predictors that increase the AIC, if applicable.
   - Ensure that both forward and backward steps are applied until no further improvements in AIC can be made.
2. **Fit the Final Model with Selected Features**:
   - After selecting the best set of features based on AIC, fit a linear regression model using these selected features on the training data.
   - Display the summary of the final model.
3. **Evaluate the Model on the Testing Set**:
   - Use the selected features to predict the response variable Y on the testing set.
   - Calculate and report the Mean Squared Error (MSE) on the testing set.
4. **Discuss the Results**:
   - Reflect on the features selected by the AIC-based stepwise selection method.

Hint: You can use the following Python code as a reference to implement the task:

```
def stepwise_selection_AIC(X, Y, initial_list=[], threshold_in=0.01, threshold_out=0.01, ver
    included = list(initial_list)
    best_aic = np.inf
```

```
    while True:
        changed = False
        excluded = list(set(X.columns) - set(included))
        aic_with_candidates = pd.Series(index=excluded, dtype=float)

        for new_column in excluded:
            model = sm.OLS(Y, sm.add_constant(X[included + [new_column]])).fit()
            aic_with_candidates[new_column] = model.aic

        best_candidate_aic = aic_with_candidates.min()
        if best_candidate_aic < best_aic:
            best_aic = best_candidate_aic
            best_feature = aic_with_candidates.idxmin()
            included.append(best_feature)
            changed = True
            if verbose:
                print(f'Add {best_feature} with AIC {best_candidate_aic}')

        model = sm.OLS(Y, sm.add_constant(X[included])).fit()
        current_aic = model.aic
        aic_without_candidates = pd.Series(index=included, dtype=float)

        for candidate in included:
            model = sm.OLS(Y, sm.add_constant(X[included].drop(columns=[candidate]))).fit()
            aic_without_candidates[candidate] = model.aic

        worst_candidate_aic = aic_without_candidates.min()
        if worst_candidate_aic < best_aic:
            best_aic = worst_candidate_aic
            worst_feature = aic_without_candidates.idxmin()
            included.remove(worst_feature)
            changed = True
            if verbose:
                print(f'Remove {worst_feature} with AIC {worst_candidate_aic}')

        if not changed:
            break

    return included

# Perform stepwise selection based on AIC
selected_features = stepwise_selection_AIC(X_train, Y_train)

# Fit the final model with the selected features
X_train_selected = sm.add_constant(X_train[selected_features])
model_selected = sm.OLS(Y_train, X_train_selected).fit()
```

```
# Display the summary of the selected model
print("Selected features:", selected_features)
print(model_selected.summary())

# Use the selected features for the test set
X_test_selected = sm.add_constant(X_test[selected_features])

# Calculate the predicted MSE on the testing set
Y_pred_test = model_selected.predict(X_test_selected)
mse_test = mean_squared_error(Y_test, Y_pred_test)

print(f"Mean Squared Error (MSE) on the testing set: {mse_test}")
```

Task: Simulate New Realizations and Apply Sequential Floating Forward Selection (SFFS)

1. **Simulate 10 New Realizations of Y**:
   - Generate 10 new sets of response variable Y based on the same model and predictor variables used in the previous tasks.
   - Each realization of Y should include the same structure but with new random errors.
2. **For Each Realization, Perform Sequential Floating Forward Selection (SFFS)**:
   - For each of the 10 realizations of Y and the same X, run the Sequential Floating Forward Selection (SFFS) procedure.
   - Choose a scoring method of your preference (e.g., R^2, mean squared error) when performing the feature selection.
3. **Comment on the Selected Models**:
   - For each realization, analyze whether the SFFS procedure selects the correct set of predictors that match the true model used to simulate Y.
   - Discuss whether the correct model is selected every time or if the procedure sometimes selects incorrect features.
   - If the correct model is not always selected, reflect on possible reasons