Step-2 Details Instructions:

1. List of variables selected for model built.
    - Unemployment rate
    - 10-year Treasury yield
    - Prime rate',
    - 'House Price Index (Level)'
    - Unemployment rate_lag_1',
    - 'Unemployment rate_lag_2
    - Log of (Dow Jones Total Stock Market Index (Level)),
    - House Price Index (Level)_YOY
    - Mortgage rate.
2. Correlation and Autocorrelation Requirements:
    - For these nine variables create a correlation matrix and heatmap (table and graph). Multicollinearity for each variable, based on VIF values.
    - Create a summary statistic, with number of observations, Mean, Std. Dev, Sum, Minimum and Maximum.
    - Autocorrelation and White Noise Test: Perform Augmented Dickey-Fuller Unit Root Test (Rho, Tau, F values) to highlight autocorrelation. You can also visually inspect the autocorrelation function plot or perform statistical tests such as the Ljung-Box test or the Durbin-Watson test to check for significant autocorrelation.
    - Exclude variables with negative results (you can send the results to me at this time if required to further narrow the list down).
3. For the remaining variables built models,
    - I would recommend using the following (which I follow in SAS): GLIMMIX procedure in SAS, random effects (random residuals), binomial distribution, link equals logit.
    - You can try other alternatives (as part of step C), but please add comments on relevant code so that I can understand the applied methods.
    - model fit statistics should include AIC, BIC statistics and others (I am assuming there is a command in Python which will generate all relevant one as there is in SAS). In-Sample Actual Default rate vs. Predicted default rate (curves).
    - Perform seven to eight iterations using different combinations, such as Unemployment rate, 10-year Treasury yield.

- The results of each of these eight iterations, including the above-mentioned statistics and in-sample plots, should be added in the deliverable report.

4. Final variables (3 variables tops) should be based on variable signs, statistical significance, In-Sample RMSE, and other model fit statistics (BIC and AIC) of the eight iterations performed in step-2 above.

5. `Once the model is finalized we will perform out-of-sample testing in the next steps.`

Note: Deliver a separate HTML, and a .py and an IPYNB files for this step. Please don't create this as an add-on to Step-1, instead a separate deliverable and files.

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df_training  = pd.read_excel('Datasets/Modeling Data-V03.xlsx',sheet_name
df_training['DRS-Target Variable'] = df_training['DRS-Target Variable']/1
df_testing  = pd.read_excel('Datasets/Modeling Data-V03.xlsx',sheet_name=
df_testing['DRS-Target Variable'] = df_testing['DRS-Target Variable']/100
```

In [3]:
```
df_training
```

Out[3]:

	Scenario Name	Date	DRS-Target Variable	Real GDP growth	Nominal GDP growth	Real disposable income growth	Nominal disposable income growth	Unemploymer rat
0	Actual	2003 Q3	NaN	6.8	9.3	7.2	10.0	6.
1	Actual	2003 Q4	NaN	4.7	7.3	1.1	3.1	5.
2	Actual	2004 Q1	NaN	2.3	5.2	1.8	5.0	5.
3	Actual	2004 Q2	NaN	3.2	6.5	4.2	7.1	5.
4	Actual	2004 Q3	NaN	3.8	6.5	2.9	4.9	5.
...	...	...	...	...	...	...	...	.
61	Actual	2018 Q4	0.0283	0.9	3.0	3.0	4.7	3.
62	Actual	2019 Q1	0.0269	2.4	3.7	3.6	4.1	3.
63	Actual	2019 Q2	0.0260	3.2	5.6	-1.4	1.3	3.
64	Actual	2019 Q3	0.0244	2.8	4.1	2.3	3.4	3.
65	Actual	2019 Q4	0.0234	1.9	3.6	2.4	4.1	3.

66 rows × 19 columns

Proprocessing

In [4]:

```
def transformation(df,training=True,testing=True):
    # Step 2: Perform data transformation - Log Transformation
    #  Defines a list called Defines a list called log_transform_variable
    # that contains the names of variables to be log-transformed.

    log_transform_variables = ['Dow Jones Total Stock Market Index (Level
                                'House Price Index (Level)',
                                'Commercial Real Estate Price Index (Leve

    # Loop through the variables to be log-transformed
    # Applies the natural logarithm (np.log()) to the selected variable.
    # Creates a new column with the log-transformed values using
    for var in log_transform_variables:
        # Apply the natural logarithm to the selected variable and create
        df[f'log_{var}'] = np.log(df[var])

    # Step 3: Perform data transformation - Year-over-Year Change
```

```
    # Defines a list called yoy_change_variables that contains the names
    # of variables for which year-over-year changes will be calculated
    yoy_change_variables = ['Dow Jones Total Stock Market Index (Level)',
                            'House Price Index (Level)',
                            'Commercial Real Estate Price Index (Level)']

    # Loop through the variables for year-over-year change calculation
    for var in yoy_change_variables:
        # Calculate the percentage change over a four-quarter period (ass
        # Creates a new column with the year-over-year change values usir
        df[f'{var}_YOY'] = df[var].pct_change(3) * 100

    # Step 4: Perform data transformation - Lags/Leads
    # Defines the range of lags to be considered for lag/lead transformat
    lags = range(1, 7)  # Lags of up to six quarters
    # Defines a list called lag_lead_variables that contains the names of
    lag_lead_variables = ['Unemployment rate', '10-year Treasury yield',

    # Loop through the variables for lag/lead transformation
    # Loop through the variables for lag/lead transformation
    for var in lag_lead_variables:
        # Loop through the specified lags
        for lag in lags:
            # Shift the variable values by the specified lag and create r
            # Shifts the variable values by the specified lag using
            df[f'{var}_lag_{lag}'] = df[var].shift(lag)


    # this code will save the transformed data into csv file in the curre
    if training is True:
        df.to_csv('training-transformed_dataset.csv',index=False)
    if testing is True:
        df = df.iloc[6:]
        df.to_csv('testing-transformed_dataset.csv',index=False)

    return df

input_data = transformation(df_training,training=True)
testing_data = transformation(df_testing,testing=True)

print('shape of input Data :{}'.format(input_data.shape))
print('shape of test Data :{}'.format(testing_data.shape))
```

```
shape of input Data :(60, 43)
shape of test Data :(8, 43)
```

In [5]:

```
input_data.head()
```

Out[5]:

	Scenario Name	Date	DRS-Target Variable	Real GDP growth	Nominal GDP growth	Real disposable income growth	Nominal disposable income growth	Unemploymer rat
6	Actual	2005 Q1	0.0142	4.5	7.9	-4.8	-2.5	5.
7	Actual	2005 Q2	0.0155	2.0	5.0	3.9	6.6	5.
8	Actual	2005 Q3	0.0159	3.2	7.0	1.7	6.1	5.
9	Actual	2005 Q4	0.0164	2.3	5.6	3.4	6.7	5.
10	Actual	2006 Q1	0.0160	5.5	8.5	8.3	10.6	4.

5 rows × 43 columns

In [6]:
```
testing_data.head()
```

Out[6]:

	Scenario Name	Date	DRS-Target Variable	Real GDP growth	Nominal GDP growth	Real disposable income growth	Nominal disposable income growth	Unemploymer rat
6	Actual	2020 Q1	0.0235	-5.1	-3.9	3.0	4.3	3.
7	Actual	2020 Q2	0.0254	-31.2	-32.4	48.5	46.1	13.
8	Actual	2020 Q3	0.0284	33.8	38.7	-16.6	-13.6	8.
9	Actual	2020 Q4	0.0274	4.5	6.6	-8.3	-6.9	6.
10	Actual	2021 Q1	0.0267	6.3	10.9	54.7	60.6	6.

5 rows × 43 columns

Transformation

In [7]:
```
def select_features(df):
    interested_features = [
                         'DRS-Target Variable',
                         '10-year Treasury yield',
                         'Prime rate','House Price Index (Level)',
                         'Unemployment rate_lag_1',
                         'Unemployment rate_lag_2',
                         'log_Dow Jones Total Stock Market Index (Level)',
                         'House Price Index (Level)_YOY',
                         'Mortgage rate',
                         ]
```

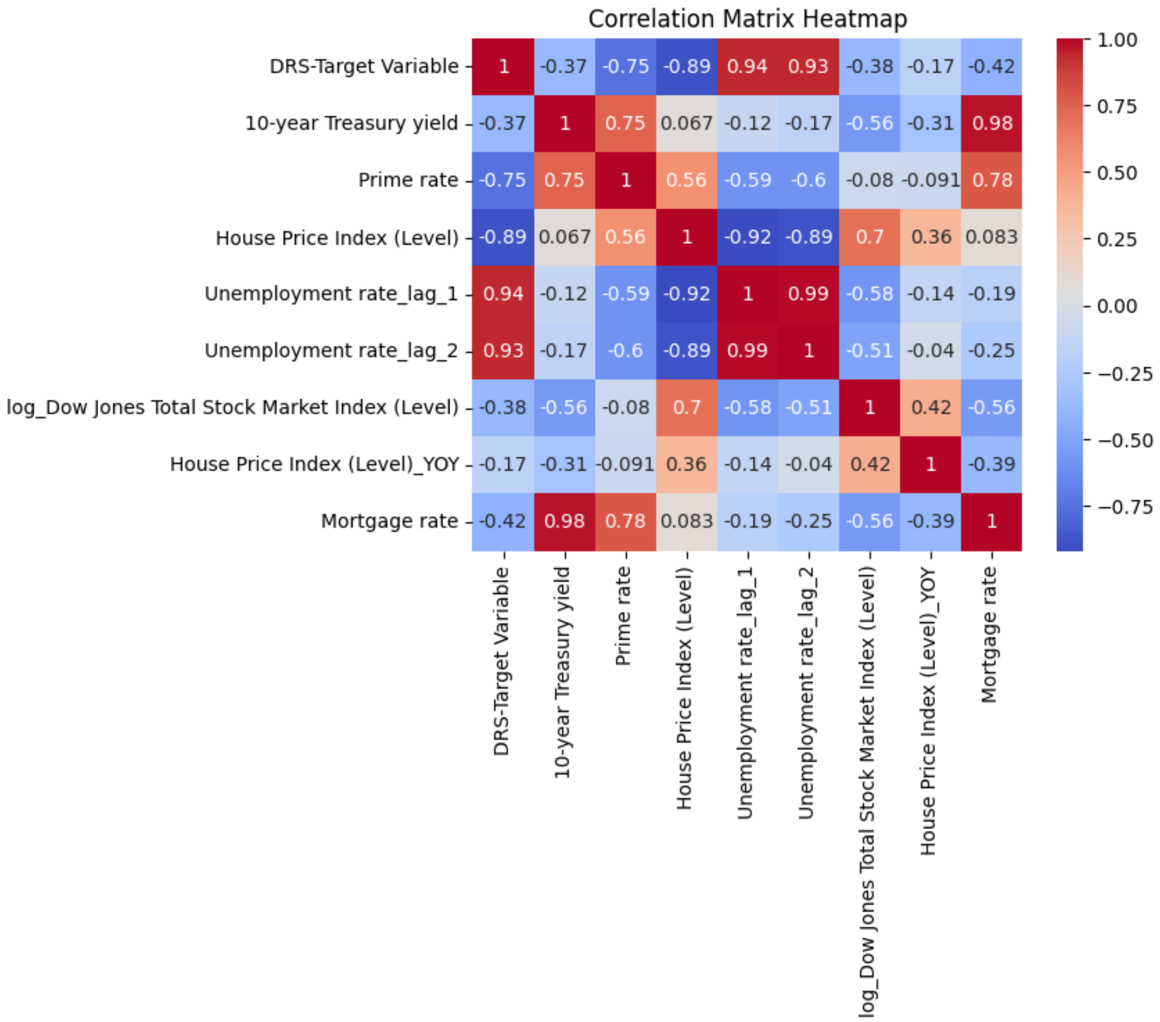
```
    df = df[interested_features]
    return df
```

In [8]:
```
interested_features = [
                        'DRS-Target Variable',
                        '10-year Treasury yield',
                        'Prime rate','House Price Index (Level)',
                        'Unemployment rate_lag_1',
                        'Unemployment rate_lag_2',
                        'log_Dow Jones Total Stock Market Index (Level)',
                        'House Price Index (Level)_YOY',
                        'Mortgage rate',
                        ]

training_data = select_features(input_data)
testing_data = select_features(testing_data)
```

Step-2

1. For these nine variables create a correlation matrix and heatmap (table and graph). Multicollinearity for each variable, based on VIF values.

In [9]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# This line calculates the correlation matrix of the df_transformed_trair
# The correlation matrix is a square matrix that shows the correlation cc
# It provides a measure of the linear relationship between variables
correlation_matrix = training_data.corr()

# This line creates a heatmap using the Seaborn library.
# The heatmap() function is used to plot the correlation matrix as a colc
# The correlation_matrix is passed as the input data. The annot=True parε
# representing the correlation coefficients. The cmap='coolwarm' paramete
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')

# This line sets the title of the plot to 'Correlation Matrix Heatmap'
# using the title() function from the Matplotlib library
plt.title('Correlation Matrix Heatmap')
plt.show()

# Summary
# Given Code : calculates the correlation matrix for a given DataFrame ar
# The heatmap provides a visual representation of the correlation betweer
# with higher correlation values shown in warmer colors and lower correla
```



In [10]:

```
# This line imports the variance_inflation_factor function
# from the statsmodels.stats.outliers_influence module.
# This function is used to calculate the variance inflation factor,
# which is a measure of multicollinearity between variables in a regressi
from statsmodels.stats.outliers_influence import variance_inflation_facto

# Create a DataFrame to store the VIF values
# this line creates an empty DataFrame called vif_data to store
# the variable names and their corresponding VIF values
vif_data = pd.DataFrame()

# This line assigns the column names of the df_transformed_
# training DataFrame to the 'Variable' column in the vif_data
# DataFrame. This will store the names of the variables for
# which VIF values are calculated.
predictors_df = training_data.iloc[:,1:]
vif_data['Variable'] = predictors_df.columns


# This line calculates the VIF values for each variable
# in the df_transformed_training DataFrame and
# assigns them to the 'VIF' column in the vif_data DataFrame.
# The VIF values are computed using a list comprehension,
# where variance_inflation_factor is applied to each column of the
# df_transformed_training DataFrame using the range() function
vif_data['VIF'] = [variance_inflation_factor(predictors_df.values, i) for
```

```
vif_data

# summary,
# Given code calculates the VIF values for each variable in the df_transf
# The VIF values indicate the degree of multicollinearity between variabl
# with higher values indicating stronger multicollinearity.
# The results are stored in a DataFrame called vif_data and
# then printed to the console.
```

Out[10]:

	Variable	VIF
0	10-year Treasury yield	399.922084
1	Prime rate	84.061378
2	House Price Index (Level)	1584.041360
3	Unemployment rate_lag_1	721.293720
4	Unemployment rate_lag_2	846.707158
5	log_Dow Jones Total Stock Market Index (Level)	2915.691100
6	House Price Index (Level)_YOY	3.165315
7	Mortgage rate	1066.915484

In [11]:

```
updated_predictor = predictors_df.drop('House Price Index (Level)',axis=1
updated_predictor
vif_data = pd.DataFrame()
vif_data['Variable'] = updated_predictor.columns
vif_data['VIF'] = [variance_inflation_factor(updated_predictor.values, i)
vif_data
```

Out[11]:

	Variable	VIF
0	10-year Treasury yield	283.130443
1	Prime rate	56.900745
2	Unemployment rate_lag_1	715.184761
3	Unemployment rate_lag_2	660.174255
4	log_Dow Jones Total Stock Market Index (Level)	141.840967
5	House Price Index (Level)_YOY	2.193561
6	Mortgage rate	730.413548

In [12]:

```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_facto

# Drop the specified columns from the predictors_df
columns_to_drop = ['House Price Index (Level)', 'log_Dow Jones Total Stoc
updated_predictor = predictors_df.drop(columns_to_drop, axis=1)

# Calculate VIF for each variable in updated_predictor
vif_data = pd.DataFrame()
vif_data['Variable'] = updated_predictor.columns
vif_data['VIF'] = [variance_inflation_factor(updated_predictor.values, i)
```

```
# Print the VIF data
print(vif_data)
```

```
                    Variable         VIF
0        10-year Treasury yield   76.471166
1                    Prime rate   55.752033
2       Unemployment rate_lag_1  707.548535
3       Unemployment rate_lag_2  659.654433
4  House Price Index (Level)_YOY    1.621184
5                 Mortgage rate  210.897864
```

In [13]:

```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_facto

# Drop the specified columns from the predictors_df
columns_to_drop = ['Unemployment rate_lag_2', 'House Price Index (Level)'
updated_predictor = predictors_df.drop(columns_to_drop, axis=1)

# Calculate VIF for each variable in updated_predictor
vif_data = pd.DataFrame()
vif_data['Variable'] = updated_predictor.columns
vif_data['VIF'] = [variance_inflation_factor(updated_predictor.values, i)

# Print the VIF data
print(vif_data)
```

```
                    Variable         VIF
0        10-year Treasury yield   73.295917
1                    Prime rate   50.581024
2       Unemployment rate_lag_1   18.334385
3  House Price Index (Level)_YOY    1.252499
4                 Mortgage rate  209.631747
```

In [14]:

```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_facto

# Drop the specified columns from the predictors_df
columns_to_drop = ['10-year Treasury yield','Unemployment rate_lag_2', 'H
updated_predictor = predictors_df.drop(columns_to_drop, axis=1)

# Calculate VIF for each variable in updated_predictor
vif_data = pd.DataFrame()
vif_data['Variable'] = updated_predictor.columns
vif_data['VIF'] = [variance_inflation_factor(updated_predictor.values, i)

# Print the VIF data
print(vif_data)
```

```
                    Variable         VIF
0                    Prime rate   50.164525
1       Unemployment rate_lag_1   16.291920
2  House Price Index (Level)_YOY    1.236072
3                 Mortgage rate  100.307276
```

In [15]:

```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_facto

# Drop the specified columns from the predictors_df
columns_to_drop = ['Mortgage rate','10-year Treasury yield','Unemployment
updated_predictor = predictors_df.drop(columns_to_drop, axis=1)
```

```
# Calculate VIF for each variable in updated_predictor
vif_data = pd.DataFrame()
vif_data['Variable'] = updated_predictor.columns
vif_data['VIF'] = [variance_inflation_factor(updated_predictor.values, i)

# Print the VIF data
print(vif_data)
```

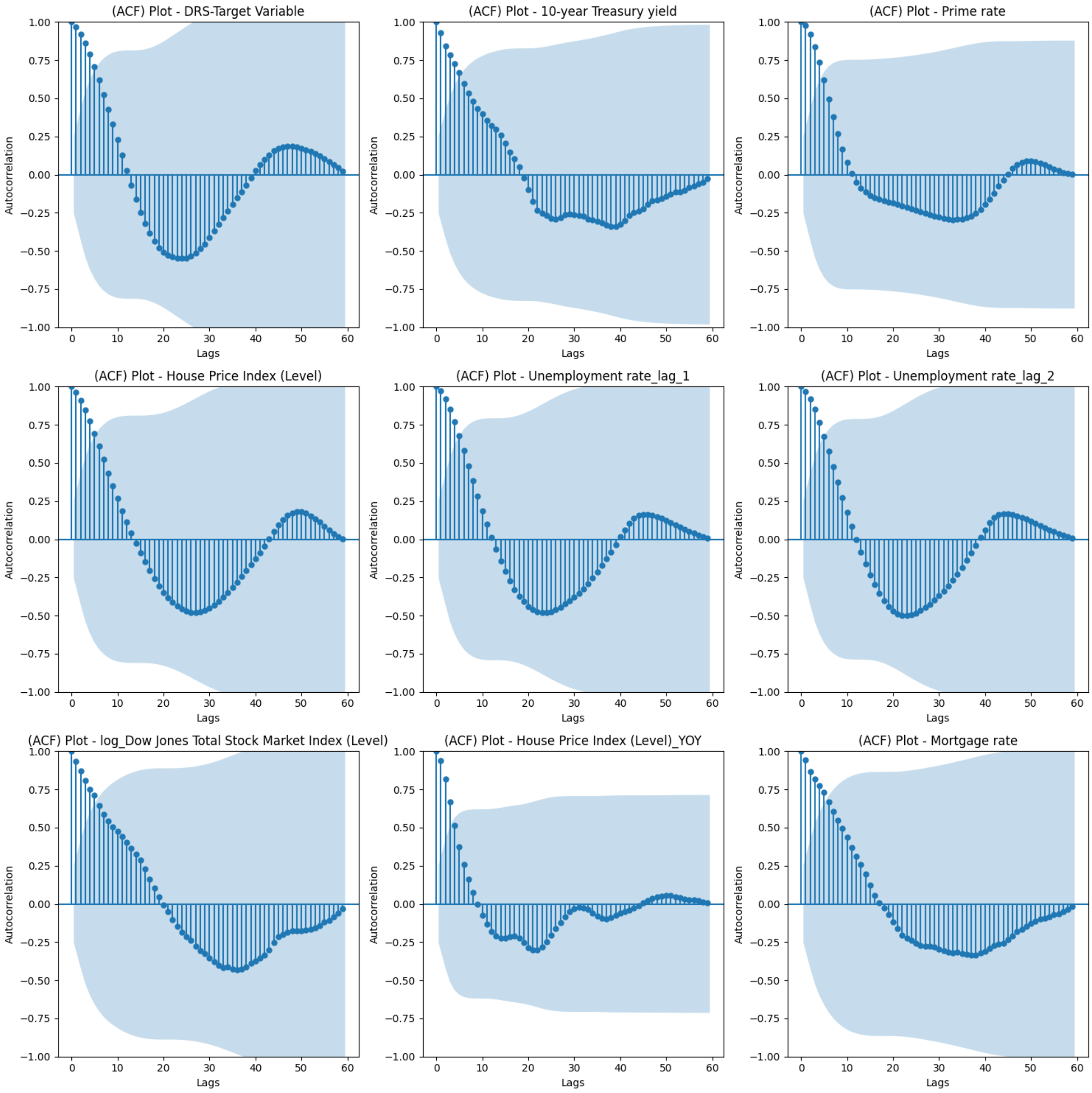
```
                        Variable       VIF
0                     Prime rate  3.294323
1        Unemployment rate_lag_1  3.285024
2  House Price Index (Level)_YOY  1.051726
```

In [16]:
```
import statsmodels.api as sm

# Perform Augmented Dickey-Fuller test for autocorrelation on each variab
# This line initializes an empty dictionary adf_results that will store t
# results of the Augmented Dickey-Fuller (ADF) test for each variable.
adf_results = {}

#  This loop iterates over each column in the df DataFrame.
for column in training_data.columns:
    # his line performs the ADF test using the sm.tsa.stattools.adfuller(
    adf_test = sm.tsa.stattools.adfuller(training_data[column])
    #This line creates a pandas Series with the ADF test results and assi
    adf_results[column] = pd.Series(adf_test[:4], index=['ADF Statistic',

# Display the ADF test results for each variable
print("\nAutocorrelation - Augmented Dickey-Fuller Test Results:")

# This loop iterates over each key-value pair in the adf_results dictiona
for column, results in adf_results.items():
    # This line prints the name of the variable being analyzed
    print("\nVariable:", column)
    # This line prints the ADF test results for the variable. The results
    # p-value, number of lags used, and number of observations used
    print(results)

# Visualize autocorrelation function (ACF) plot for each variable

# This line calculates the number of variables in the DataFrame.
num_variables = len(training_data.columns)
# This line sets the number of columns for the subplots to be displayed
num_cols = 3  # Set the number of columns for subplots

# This line calculates the number of rows needed for the subplots based o
num_rows = (num_variables + num_cols - 1) // num_cols
#  This line creates a figure and axes for the subplots, with the specifi
# The figsize parameter determines the size of the figure.
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows*5))


# The following loop iterates over each variable in the DataFrame and plo
for i, column in enumerate(training_data.columns):
    # This line calculates the row index for the subplot based on the cur
    row = i // num_cols
    # : This line calculates the column index for the subplot based on th
    col = i % num_cols
    # This line selects the current subplot for plotting.
    ax = axes[row, col]
```

```
    #  This line plots the autocorrelation function (ACF) for the variabl
    sm.graphics.tsa.plot_acf(training_data[column], lags=len(training_dat
    # This line sets the title of the subplo
    ax.set_title("(ACF) Plot - " + column)
    # This line sets the label
    ax.set_xlabel("Lags")
    ax.set_ylabel("Autocorrelation")

# This line adjusts the layout of the subplots to prevent overlapping
plt.tight_layout()
# This line displays the subplots.
plt.show()
# In summary, this code performs the Augmented Dickey-Fuller (ADF) test f
```

```
Autocorrelation - Augmented Dickey-Fuller Test Results:

Variable: DRS-Target Variable
ADF Statistic                  -2.692612
p-value                         0.075325
# Lags Used                     4.000000
Number of Observations Used    55.000000
dtype: float64

Variable: 10-year Treasury yield
ADF Statistic                  -1.184460
p-value                         0.680245
# Lags Used                     2.000000
Number of Observations Used    57.000000
dtype: float64

Variable: Prime rate
ADF Statistic                  -3.469074
p-value                         0.008818
# Lags Used                     3.000000
Number of Observations Used    56.000000
dtype: float64

Variable: House Price Index (Level)
ADF Statistic                  -0.884130
p-value                         0.793154
# Lags Used                     5.000000
Number of Observations Used    54.000000
dtype: float64

Variable: Unemployment rate_lag_1
ADF Statistic                  -1.546206
p-value                         0.510543
# Lags Used                     1.000000
Number of Observations Used    58.000000
dtype: float64

Variable: Unemployment rate_lag_2
ADF Statistic                  -1.376505
p-value                         0.593487
# Lags Used                     1.000000
Number of Observations Used    58.000000
dtype: float64

Variable: log_Dow Jones Total Stock Market Index (Level)
ADF Statistic                  -0.048824
p-value                         0.954303
# Lags Used                     0.000000
Number of Observations Used    59.000000
dtype: float64

Variable: House Price Index (Level)_YOY
ADF Statistic                  -2.108353
p-value                         0.241169
# Lags Used                     2.000000
Number of Observations Used    57.000000
dtype: float64

Variable: Mortgage rate
ADF Statistic                  -1.092718
```

```
p-value                         0.717967
# Lags Used                     2.000000
Number of Observations Used    57.000000
dtype: float64
```



In [17]:
```python
# Calculate summary statistics
summary_stats = training_data.describe().transpose()

# Display the summary statistics
print("\nSummary Statistics:")
summary_stats
```

```
Summary Statistics:
```

Out[17]:

	count	mean	std	min	25%	50%
DRS-Target Variable	60.0	0.058122	0.033367	0.014200	0.028000	0.049450
10-year Treasury yield	60.0	3.066667	1.040806	1.600000	2.200000	2.800000
Prime rate	60.0	4.546667	1.674481	3.300000	3.300000	3.500000
House Price Index (Level)	60.0	168.996667	24.676764	133.400000	142.775000	171.350000
Unemployment rate_lag_1	60.0	6.131667	1.969642	3.600000	4.600000	5.300000
Unemployment rate_lag_2	60.0	6.161667	1.943969	3.600000	4.675000	5.350000
log_Dow Jones Total Stock Market Index (Level)	60.0	9.726728	0.353813	8.992707	9.469948	9.617690
House Price Index (Level)_YOY	60.0	1.605091	5.982292	-13.422007	-1.425234	3.588308
Mortgage rate	60.0	4.708333	0.989486	3.400000	3.900000	4.400000

Modeling the Data

In [18]:

```
columns_to_drop = ['Unemployment rate_lag_2', 'House Price Index (Level)'
training_data.drop(columns_to_drop,axis=1,inplace=True)
training_data.head(5)
```

```
/tmp/ipykernel_26544/1020528254.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  training_data.drop(columns_to_drop,axis=1,inplace=True)
```

Out[18]:

	DRS-Target Variable	10-year Treasury yield	Prime rate	Unemployment rate_lag_1	House Price Index (Level)_YOY	Mortgage rate
6	0.0142	4.4	5.4	5.4	11.695906	5.8
7	0.0155	4.2	5.9	5.3	12.241055	5.7
8	0.0159	4.3	6.4	5.1	12.053301	5.8
9	0.0164	4.6	7.0	5.0	10.645724	6.2
10	0.0160	4.7	7.4	5.0	7.997763	6.2

In [19]:

```
columns_to_drop = ['Unemployment rate_lag_2', 'House Price Index (Level)'
testing_data.drop(columns_to_drop,axis=1,inplace=True)
testing_data.head(5)
```

Out[19]:

	DRS-Target Variable	10-year Treasury yield	Prime rate	Unemployment rate_lag_1	House Price Index (Level)_YOY	Mortgage rate
6	0.0235	1.4	4.4	3.6	3.776291	3.5
7	0.0254	0.7	3.3	3.8	3.928064	3.2
8	0.0284	0.6	3.3	13.0	5.184493	3.0
9	0.0274	0.9	3.3	8.8	7.185629	2.8
10	0.0267	1.4	3.3	6.8	10.291439	2.9

Family Functions:

- sm.families.Binomial(): Binomial family for binary response data.
- sm.families.Gaussian(): Gaussian family for continuous response data.
- sm.families.Poisson(): Poisson family for count data.
- sm.families.NegativeBinomial(): Negative binomial family for count data.
- sm.families.Gamma(): Gamma family for continuous positive response data.
- sm.families.InverseGaussian(): Inverse Gaussian family for continuous positive response data.

Link Functions:

- sm.families.links.logit(): Logit link function for binary response data.
- sm.families.links.identity(): Identity link function for continuous response data.
- sm.families.links.log() or sm.families.links.log1p(): Log link function for count data (Poisson, Negative Binomial).
- sm.families.links.inverse_power(): Inverse power link function for Gamma and Inverse Gaussian families.

In addition to these options, statsmodels also provides support for custom family and link functions.]

Summary interpretations

To interpret the results of the GLM model, you can analyze various aspects of the model summary. Here are some key points to consider:

1. Coefficients: The coefficients indicate the estimated effect of each predictor on the response variable. They represent the average change in the response for a one-unit increase in the predictor, assuming all other predictors are held constant. For example, if the coefficient for "Prime rate" is 0.2, it suggests that, on average, a one-unit increase in the "Prime rate" is associated with a 0.2 increase in the "DRS-Target Variable."

2. Standard Errors: The standard errors provide an estimate of the variability or uncertainty associated with the coefficient estimates. Smaller standard errors

indicate more precise estimates. In hypothesis testing, the standard errors are used to calculate t-statistics and p-values.

3. P-values: The p-values associated with the coefficients indicate the statistical significance of each predictor. They indicate the probability of observing a coefficient as extreme as the estimated coefficient if the null hypothesis (no effect) were true. Typically, a p-value below a certain threshold (e.g., 0.05) is considered statistically significant, suggesting a significant relationship between the predictor and the response.

4. AIC and BIC: The Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) are measures of model fit that balance goodness of fit and model complexity. Lower AIC and BIC values indicate better-fitting models. You can compare the AIC and BIC values of different models to assess their relative quality.

Based on the provided code, you can examine the model summary output, including the coefficients, standard errors, p-values, AIC, and BIC. Interpretation of the coefficients depends on the specific dataset and context of your analysis. Remember to consider the scale and context of your variables when interpreting the coefficient values.

In [20]:

```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Load the dataset
# data = pd.DataFrame({
#     'DRS-Target Variable': [1.42, 1.55, 1.59, 1.64, 1.60],
#     'Prime rate': [5.4, 5.9, 6.4, 7.0, 7.4],
#     'Unemployment rate_lag_1': [5.4, 5.3, 5.1, 5.0, 5.0],
#     'House Price Index (Level)_YOY': [11.695906, 12.241055, 12.053301,
# })
# df_training = data
# Define the predictors and response variable
predictors = ['Prime rate','Unemployment rate_lag_1', 'House Price Index
response = 'DRS-Target Variable'


# Fit the GLM model
# 1 -sm.families.Gamma(sm.families.links.log())
# 2- sm.families.Gaussian(sm.families.links.identity())

model_1 = sm.GLM(training_data[response], sm.add_constant(training_data[p
result_1 = model_1.fit()

# Fit the GLM with Gaussian family and identity link
model_2 = sm.GLM(training_data[response], sm.add_constant(training_data[p
result_2 = model_2.fit()

# Fit the GLM with Tweedie family and appropriate link
model_3 = sm.GLM(training_data[response], sm.add_constant(training_data[p
result_3 = model_3.fit()

# Fit the GLM with Tweedie family and appropriate link
model_4 = sm.GLM(training_data[response], sm.add_constant(training_data[p
result_4 = model_4.fit()
```

```
print('Summary of Model-1 : {}'.format(result_1.summary()))
print('Summary of Model-2 : {}'.format(result_2.summary()))
print('Summary of Model-3 : {}'.format(result_3.summary()))
print('Summary of Model-4 : {}'.format(result_4.summary()))


print('AIC : {} and BIC : {} of Model-{}'.format(result_1.aic ,result_1.b
print('AIC : {} and BIC : {} of Model-{}'.format(result_2.aic ,result_2.b
print('AIC : {} and BIC : {} of Model-{}'.format(result_3.aic ,result_3.b
print('AIC : {} and BIC : {} of Model-{}'.format(result_4.aic ,result_4.b
```

```
Summary of Model-1 :                 Generalized Linear Model Regression
Results
==============================================================================
=====
Dep. Variable:     DRS-Target Variable   No. Observations:
60
Model:                             GLM   Df Residuals:
55
Model Family:                    Gamma   Df Model:
4
Link Function:                     log   Scale:                         0.0
25471
Method:                           IRLS   Log-Likelihood:                  2
08.89
Date:                 Sun, 11 Jun 2023   Deviance:
1.4389
Time:                         13:00:34   Pearson chi2:
1.40
No. Iterations:                     13   Pseudo R-squ. (CS):
1.000
Covariance Type:             nonrobust
==============================================================================
=======================
                                    coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
const                            -2.7618      0.151    -18.342      0.000
-3.057      -2.467
Prime rate                       -0.1134      0.030     -3.730      0.000
-0.173      -0.054
Unemployment rate_lag_1           0.1941      0.016     12.450      0.000
0.164       0.225
House Price Index (Level)_YOY    -0.0317      0.004     -7.797      0.000
-0.040      -0.024
Mortgage rate                    -0.1886      0.046     -4.132      0.000
-0.278      -0.099
==============================================================================
=======================
Summary of Model-2 :                 Generalized Linear Model Regression
Results
==============================================================================
=====
Dep. Variable:     DRS-Target Variable   No. Observations:
60
Model:                             GLM   Df Residuals:
55
Model Family:                 Gaussian   Df Model:
4
Link Function:                identity   Scale:                       3.675
1e-05
Method:                           IRLS   Log-Likelihood:                  2
23.81
Date:                 Sun, 11 Jun 2023   Deviance:                     0.00
20213
Time:                         13:00:34   Pearson chi2:                   0.
00202
No. Iterations:                      3   Pseudo R-squ. (CS):
1.000
Covariance Type:             nonrobust
```

```
==============================================================================
=======================
                                  coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
const                           0.0241      0.006      4.207      0.000
0.013       0.035
Prime rate                     -0.0013      0.001     -1.141      0.254
-0.004       0.001
Unemployment rate_lag_1         0.0139      0.001     23.543      0.000
0.013       0.015
House Price Index (Level)_YOY   -0.0010      0.000     -6.316      0.000
-0.001      -0.001
Mortgage rate                  -0.0093      0.002     -5.375      0.000
-0.013      -0.006
==============================================================================
=======================
Summary of Model-3 :                 Generalized Linear Model Regression
Results
==============================================================================
=====
Dep. Variable:     DRS-Target Variable   No. Observations:
60
Model:                             GLM   Df Residuals:
55
Model Family:                  Tweedie   Df Model:
4
Link Function:                     Log   Scale:                         0.00
55154
Method:                           IRLS   Log-Likelihood:                   2
09.32
Date:                 Sun, 11 Jun 2023   Deviance:                        0.
31016
Time:                         13:00:34   Pearson chi2:
0.303
No. Iterations:                     11   Pseudo R-squ. (CS):
1.000
Covariance Type:             nonrobust
==============================================================================
=======================
                                  coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
const                          -2.7637      0.159    -17.428      0.000
-3.075      -2.453
Prime rate                     -0.1306      0.032     -4.143      0.000
-0.192      -0.069
Unemployment rate_lag_1         0.1843      0.015     12.711      0.000
0.156       0.213
House Price Index (Level)_YOY   -0.0252      0.004     -5.759      0.000
-0.034      -0.017
Mortgage rate                  -0.1606      0.046     -3.455      0.001
-0.252      -0.069
==============================================================================
=======================
Summary of Model-4 :                 Generalized Linear Model Regression
Results
==============================================================================
```

```
=====
Dep. Variable:     DRS-Target Variable   No. Observations:
60
Model:                             GLM   Df Residuals:
55
Model Family:                 Binomial   Df Model:
4
Link Function:                   logit   Scale:
1.0000
Method:                           IRLS   Log-Likelihood:             -
9.5334
Date:                 Sun, 11 Jun 2023   Deviance:                    0.0
68724
Time:                         13:00:34   Pearson chi2:
0.0676
No. Iterations:                      6   Pseudo R-squ. (CS):           0.
01932
Covariance Type:             nonrobust
==========================================================================
=======================
                                   coef    std err          z      P>|z|
[0.025      0.975]
--------------------------------------------------------------------------
-----------------------
const                           -2.7471      4.867     -0.564      0.572
-12.287       6.793
Prime rate                      -0.1489      0.966     -0.154      0.877
-2.042       1.744
Unemployment rate_lag_1          0.1919      0.406      0.473      0.636
-0.604       0.988
House Price Index (Level)_YOY   -0.0201      0.134     -0.150      0.881
-0.282       0.242
Mortgage rate                   -0.1444      1.364     -0.106      0.916
-2.817       2.528
==========================================================================
=======================
AIC : -407.77173539029945 and BIC : -223.75009403463423 of Model-1
AIC : -437.62803189159246 and BIC : -225.18692959248636 of Model-2
AIC : -408.63617736556364 and BIC : -224.87878872509634 of Model-3
AIC : 29.066764048447062 and BIC : -225.12022649373614 of Model-4
```

```
/home/iffi/anaconda3/envs/sep_darts_2/lib/python3.11/site-packages/statsmo
dels/genmod/families/links.py:13: FutureWarning: The log link alias is dep
recated. Use Log instead. The log link alias will be removed after the 0.1
5.0 release.
  warnings.warn(
/home/iffi/anaconda3/envs/sep_darts_2/lib/python3.11/site-packages/statsmo
dels/genmod/families/links.py:13: FutureWarning: The identity link alias i
s deprecated. Use Identity instead. The identity link alias will be remove
d after the 0.15.0 release.
  warnings.warn(
/home/iffi/anaconda3/envs/sep_darts_2/lib/python3.11/site-packages/statsmo
dels/genmod/families/links.py:13: FutureWarning: The logit link alias is d
eprecated. Use Logit instead. The logit link alias will be removed after t
he 0.15.0 release.
  warnings.warn(
/home/iffi/anaconda3/envs/sep_darts_2/lib/python3.11/site-packages/statsmo
dels/genmod/generalized_linear_model.py:1838: FutureWarning: The bic value
is computed using the deviance formula. After 0.13 this will change to the
log-likelihood based formula. This change has no impact on the relative ra
nk of models compared using BIC. You can directly access the log-likelihoo
d version using the `bic_llf` attribute. You can suppress this message by
calling statsmodels.genmod.generalized_linear_model.SET_USE_BIC_LLF with T
rue to get the LLF-based version now or False to retainthe deviance versio
n.
  warnings.warn(
```

In-Sampling Evaluations

In [21]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

import matplotlib.pyplot as plt

def Evaluation(df, predictors, response, fitted_model_list):
    """
    It will plot the actual and predict curves and calculates the given e
    Args:
        df (_type_): it will take training dataframe or testing dataframe
        predictors (_type_): it will take the list of perdictors variable
        response (_type_): _if will take the response variable name
        fitted_model_list (_type_): it will take list of fitted model
    """

    for model_fitted_instance in fitted_model_list:
        model = model_fitted_instance[0]
        model_name = model_fitted_instance[1]
        print(model_name)

        results_df = pd.DataFrame()
        results_df['Predicted'] = model.predict(sm.add_constant(df[predic
        results_df['Actual'] = df[response].tolist()

        # Calculate MSE
        mse = mean_squared_error(results_df['Predicted'], results_df['Act
        mae = mean_absolute_error(results_df['Predicted'], results_df['Ac

        # Calculate RMSE
        rmse = np.sqrt(mse)
```

```
        print("MSE:", mse)
        print("RMSE:", rmse)
        print('MAE', mae)


        # Plotting
        plt.figure(figsize=(10, 5))
        plt.plot(df['Date'], results_df['Actual'], label='Actual', marker
        plt.plot(df['Date'], results_df['Predicted'], label='Predicted',
        plt.xlabel('Observation')
        plt.ylabel('DRS Target Variable')
        plt.yticks(np.arange(0, 1, 0.09))
        plt.title(f'InSampling Results Curves | Model-{model_name}')
        plt.legend()

        # Add error metrics in the legend
        legend_text = f"MSE: {mse:.4f}, RMSE: {rmse:.4f}, MAE: {mae:.4f}"
        plt.legend(title=legend_text)
        plt.xticks(rotation=90)

        plt.tight_layout()
        plt.show()

import datetime

def generate_quarterly_dates(start_year, end_year):
    """
    Generate a list of quarterly dates in the format 'Q<quarter>-<year>'.

    Args:
        start_year (int): The starting year.
        end_year (int): The ending year.

    Returns:
        list: A list of quarterly dates.

    Example:
        >>> generate_quarterly_dates(2020, 2021)
        ['Q1-2020', 'Q2-2020', 'Q3-2020', 'Q4-2020', 'Q1-2021', 'Q2-2021'
    """
    start_date = datetime.date(start_year, 1, 1)
    end_date = datetime.date(end_year, 12, 31)
    quarter_dates = []

    current_date = start_date
    while current_date <= end_date:
        quarter = (current_date.month - 1) // 3 + 1
        quarter_year = "Q{}-{}".format(quarter, current_date.strftime("%Y
        quarter_dates.append(quarter_year)
        current_date += datetime.timedelta(days=92)

    return quarter_dates
```

Evalution of In-Sampling(Training Data)

In [22]:
```
testing_data
```

Out[22]:

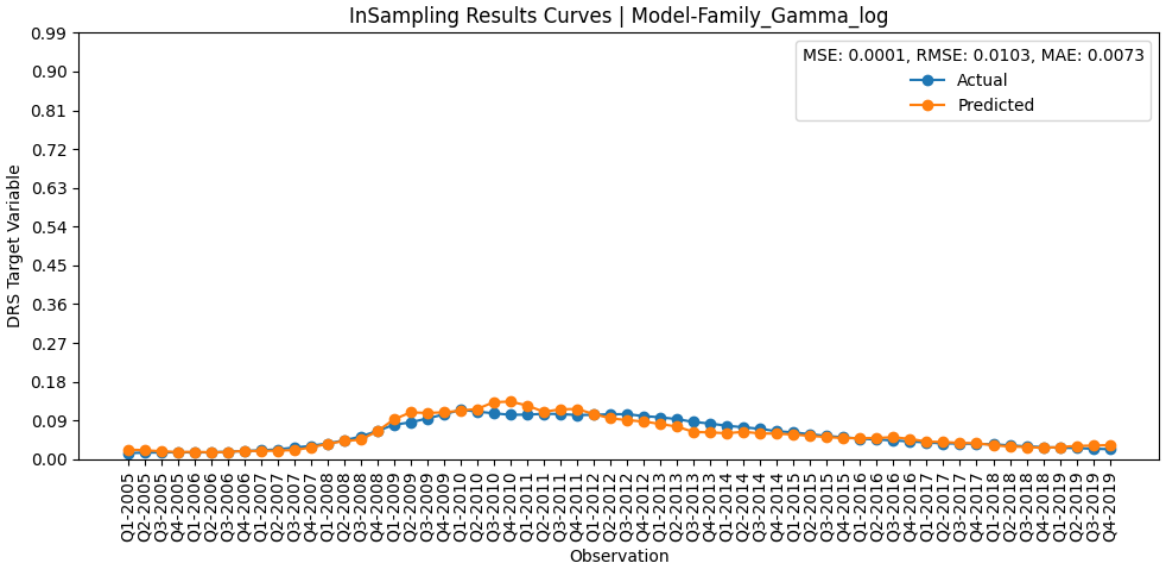
	DRS-Target Variable	10-year Treasury yield	Prime rate	Unemployment rate_lag_1	House Price Index (Level)_YOY	Mortgage rate
6	0.0235	1.4	4.4	3.6	3.776291	3.5
7	0.0254	0.7	3.3	3.8	3.928064	3.2
8	0.0284	0.6	3.3	13.0	5.184493	3.0
9	0.0274	0.9	3.3	8.8	7.185629	2.8
10	0.0267	1.4	3.3	6.8	10.291439	2.9
11	0.0248	1.6	3.3	6.2	13.188277	3.0
12	0.0231	1.4	3.3	5.9	14.224323	2.9
13	0.0228	1.6	3.3	5.1	10.528489	3.1

In [23]:

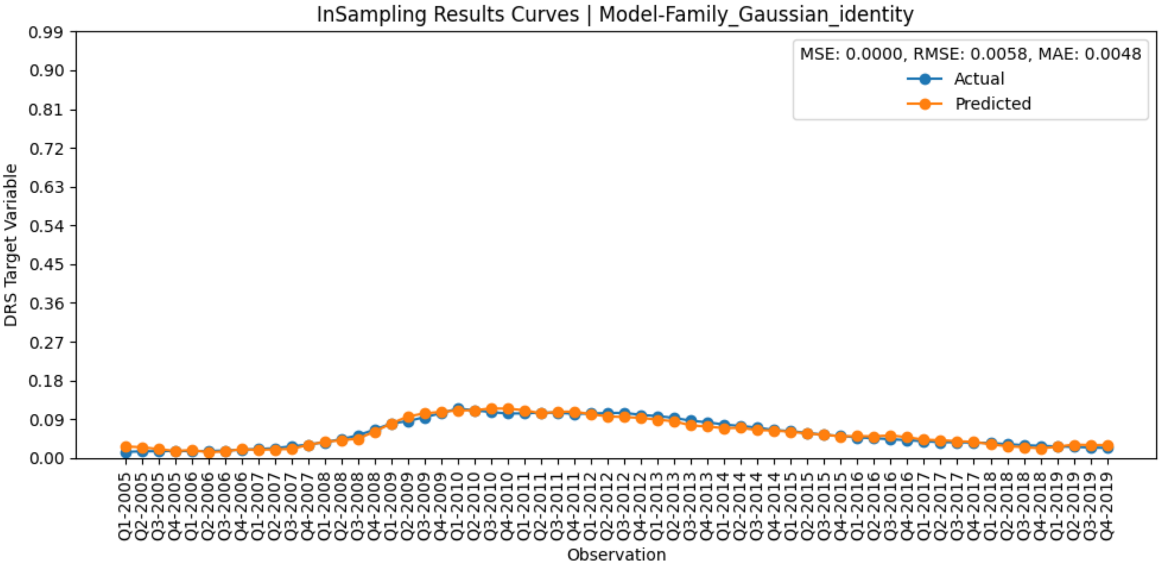
```
predictors = ['Prime rate', 'Unemployment rate_lag_1', 'House Price Index
response = 'DRS-Target Variable'
training_data['Date'] = generate_quarterly_dates(2005,2019)
df = training_data
fitted_model_list = [(result_1 ,'Family_Gamma_log' ) ,
                (result_2 ,'Family_Gaussian_identity') ,
                (result_3 ,'Family_Tweedie_log') ,
                (result_4 ,'Family_Bionomail_logit')
                ]
Evaluation(df,predictors,response,fitted_model_list)
```

```
Family_Gamma_log
MSE: 0.00010658980347289853
RMSE: 0.010324233795923964
MAE 0.007341419149024233
```

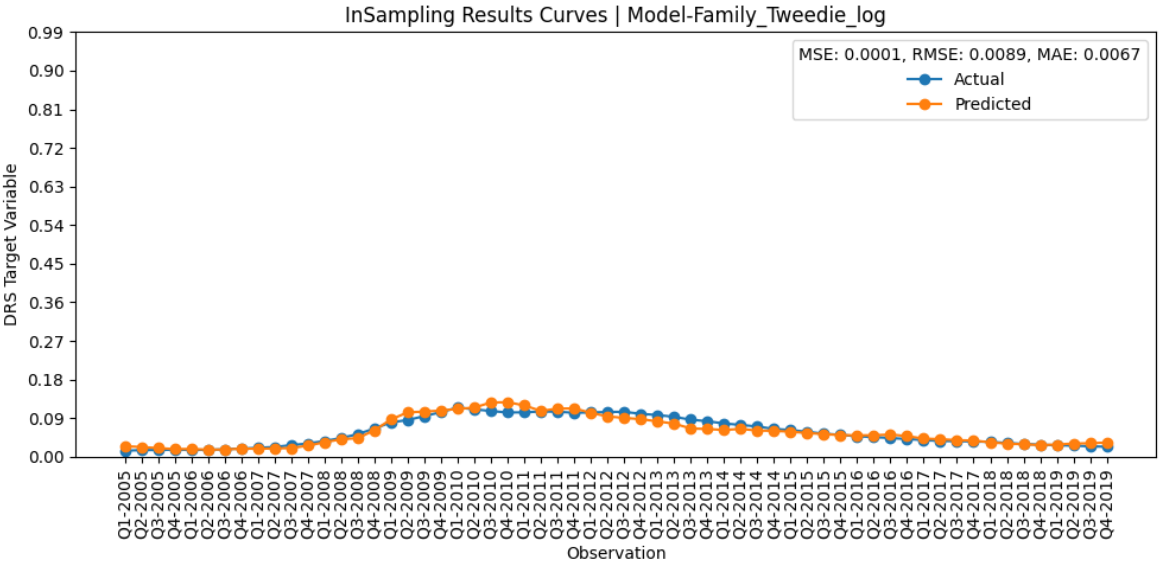
```
/tmp/ipykernel_26544/4127303491.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  training_data['Date'] = generate_quarterly_dates(2005,2019)
```



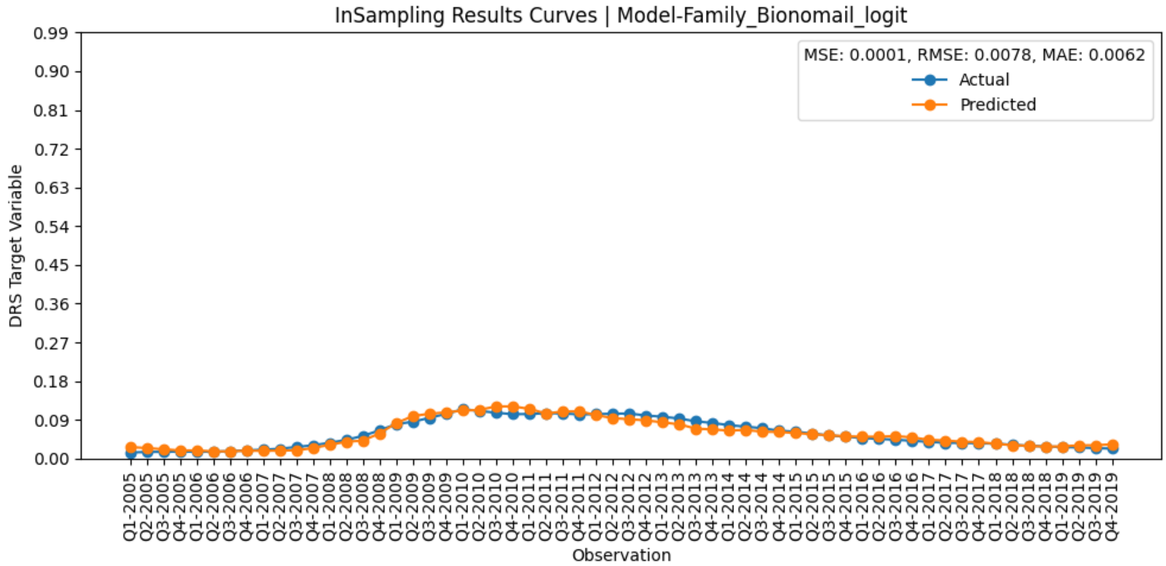
```
Family_Gaussian_identity
MSE: 3.368882881928603e-05
RMSE: 0.0058042078545901534
MAE 0.004759219320465291
```




```
Family_Tweedie_log
MSE: 8.003413189558557e-05
RMSE: 0.008946179737496087
MAE 0.006740172109786416
```



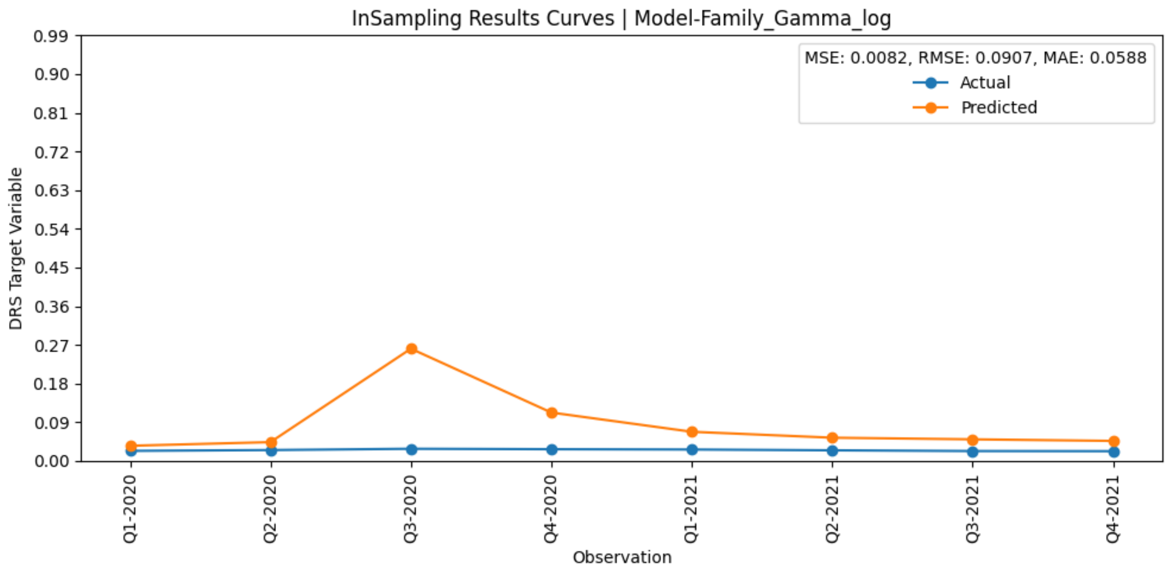

```
Family_Bionomail_logit
MSE: 6.148077868889466e-05
RMSE: 0.007840967968873146
MAE 0.006219299311077302
```



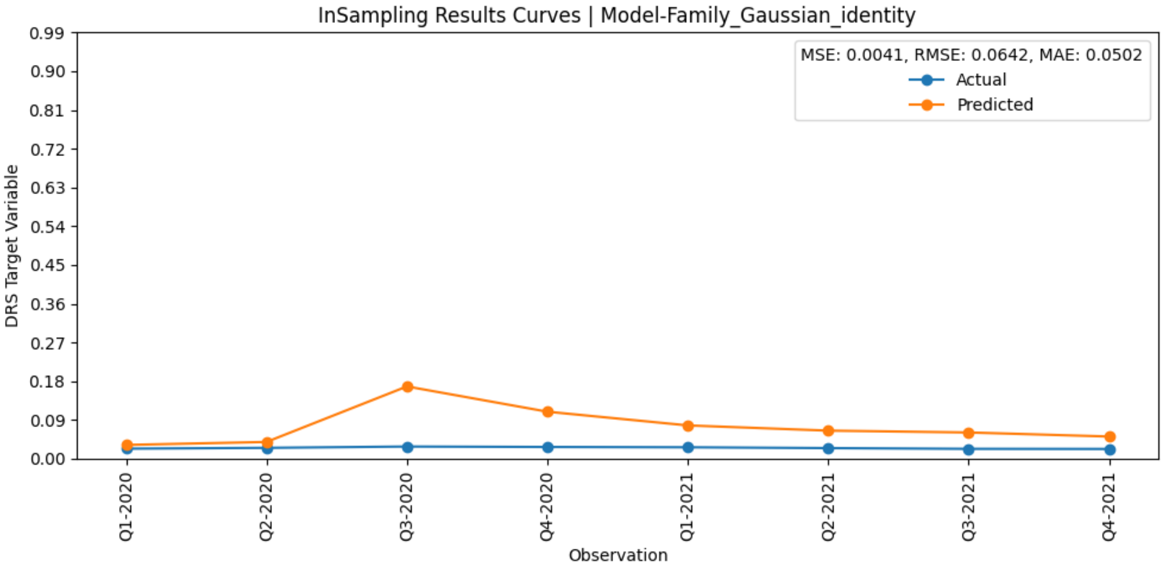
Evaluation of Out-Sampling(Testing Data~Unseen Data)

In [24]:
```
predictors = ['Prime rate', 'Unemployment rate_lag_1', 'House Price Index
response = 'DRS-Target Variable'
testing_data['Date'] = generate_quarterly_dates(2020,2021)
df = testing_data
fitted_model_list = [(result_1 ,'Family_Gamma_log' ) ,
                (result_2 ,'Family_Gaussian_identity') ,
                (result_3 ,'Family_Tweedie_log') ,
                (result_4 ,'Family_Bionomail_logit')
                ]
Evaluation(df,predictors,response,fitted_model_list)
```

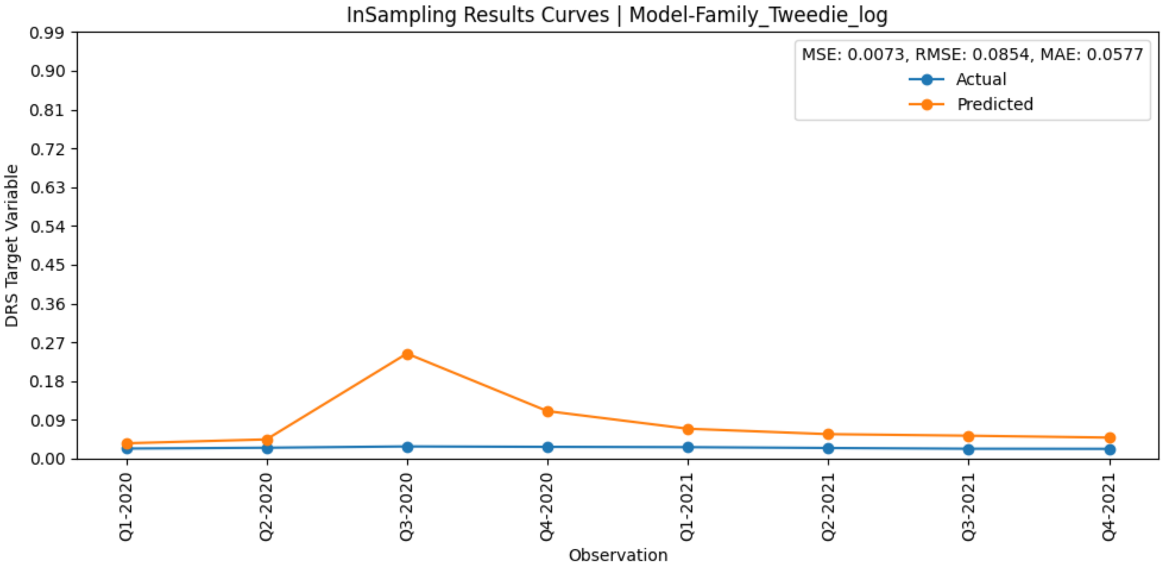
```
Family_Gamma_log
MSE: 0.008227803496060657
RMSE: 0.09070724059335426
MAE 0.058764502500823976
```



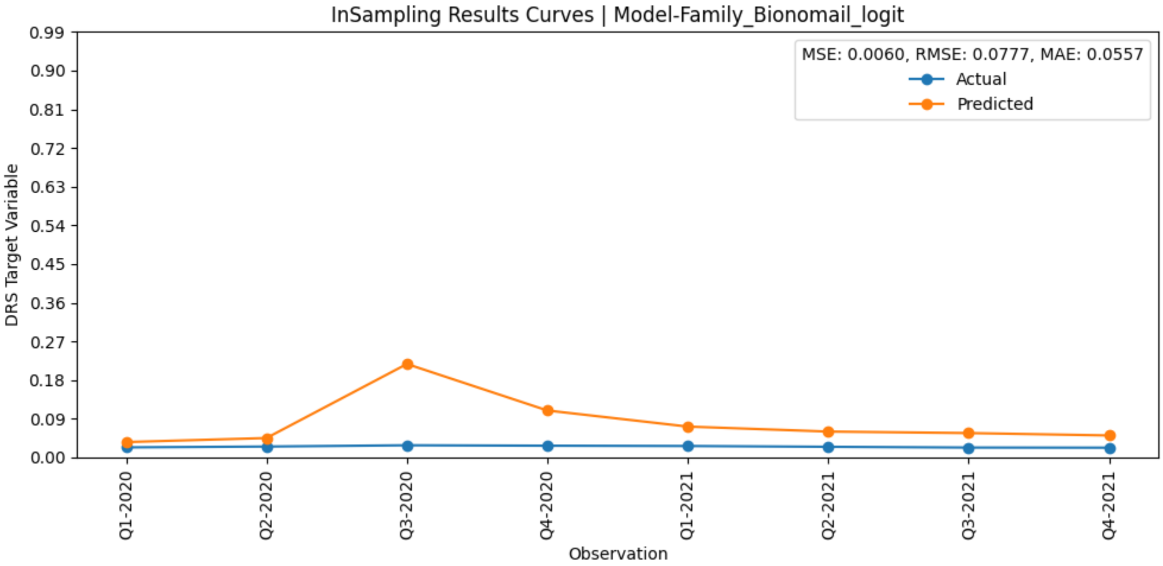

```
Family_Gaussian_identity
MSE: 0.0041187689622827865
RMSE: 0.06417763599792989
MAE 0.05024491535002664
```




```
Family_Tweedie_log
MSE: 0.007288576198042416
RMSE: 0.08537315853382968
MAE 0.057657337733400274
```




```
Family_Bionomail_logit
MSE: 0.006031002866617243
RMSE: 0.07765953171773085
MAE 0.055738194772627314
```



Instructions

In [25]:
```
# step 1 : jupyter nbconvert --to html --execute filename.ipynb # it will
# step 2 : Then you convert it to PDF by ctrl+P
```

In [ ]:
```
```