Enhancing Credit Scoring Models Through Time Series Clustering

Note:
1. Some Code in the Notebook Has Been Modified to Comply with Experian's Data and Privacy Policies.
2. Visualizations Are Attached Separately for Added Clarity.

Delphi Model

1. Import Libraries

In [55]:
```
# Importing easy_peas3 for data acquisition and loading
import easy_peas3
from easy_peas3 import S3

# Importing necessary libraries for data manipulation, visualization, and analysis
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Importing garbage collector to manage memory during runtime
import gc

# Importing tqdm for progress bars in Jupyter Notebooks
from tqdm.autonotebook import tqdm

# Importing preprocessing tools from scikit-learn
from sklearn.preprocessing import StandardScaler

# Importing tools for model selection and evaluation
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from tabulate import tabulate

# Suppressing warnings to ignore any unnecessary warnings
import warnings
warnings.filterwarnings('ignore')

# Configuring pandas display options for better dataframe visualization
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)
```

2. Data Loading

In [56]:
```
# Creating an S3 object and connecting to the project bucket
s3 = S3(project_bucket='**')
bucket = s3.project_bucket()
```

In [57]:

```python
# Reading Flag Data from the specified path
# 'security_classification' indicates the sensitivity level of the data
# 'subpath' specifies the exact location of the CSV file within the S3 bucket
# 'low_memory' is set to False to ensure the file is processed without memory optimization concerns

data_flag = bucket.read_data_assets_csv(
    security_classification="**",
    subpath='**',
    low_memory=False
)
```

In [58]:

```python
# List of Delphi summary attributes provided by Experian
delphi_features = [
    'E1_A_03', 'E1_A_04', 'E1_A_05', 'E1_A_06', 'E1_A_07', 'E1_A_08', 'E1_A_09', 'E1_A_10', 'E1_A_11',
    'E1_B_01', 'E1_B_07', 'E1_B_08', 'E1_B_09', 'E1_B_13',
    'E1_D_01',
    'E1_E_01', 'E1_E_02',
    'ND_ECC_03', 'ND_ECC_07',
    'ND_HAC_09', 'ND_INC_03',
    'ND_PSD_01', 'ND_PSD_04', 'ND_PSD_11',
    'EA1_D_01', 'EA1_D_02',
    'EA1_E_01', 'EA1_E_04',
    'SP_A_04', 'SP_A_10',
    'SP_B1_11', 'SP_B1_16',
    'SP_B2_19',
    'SP_B3_22', 'SP_B3_23',
    'SP_E1_28', 'SP_F1_30', 'SP_F2_33',
    'SP_F3_34', 'SP_F3_35', 'SP_F3_36',
    'SP_G_37', 'SP_G_38',
    'VM01_SP_VM2_15', 'VM01_SP_VM2_18', 'VM01_SP_VM2_20', 'VM01_SP_VM2_25', 'VM01_SP_VM2_29', 'VM01_SP_VM2_32',
    'VM01_SP_VM2_34',
    'VM02_SP_VM1_18',
    'VM04_SP_VM1_07', 'VM04_SP_VM1_08', 'VM04_SP_VM1_15', 'VM04_SP_VM1_18', 'VM04_SP_VM1_23', 'VM04_SP_VM1_24',
    'VM05_SP_VM1_07',
    'VM07_SP_VM1_09', 'VM07_SP_VM1_14', 'VM07_SP_VM1_25', 'VM07_SP_VM1_26', 'VM07_SP_VM1_04',
    'VM08_SP_VM2_07', 'VM08_SP_VM2_15', 'VM08_SP_VM2_16', 'VM08_SP_VM2_18', 'VM08_SP_VM2_20', 'VM08_SP_VM2_22', 'VM08_SP_VM2_23',
    'VM10_SP_VM2_08', 'VM10_SP_VM2_09', 'VM10_SP_VM2_14', 'VM10_SP_VM2_22', 'VM10_SP_VM2_23', 'VM10_SP_VM2_32',
    'TRD_STL_14', 'TRD_STL_22', 'TRD_STL_23',
    'HC_A_03', 'HC_A_14', 'HC_B_03',
    'HC_C_01', 'HC_C_05',
    'HC_D_01', 'HC_D_02',
    'HC_P_07',
    'PD_A_01', 'PD_A_02', 'PD_A_03',
    'PD_B_05',
    'PD_D_15',
    'TRD_A_04', 'TRD_A_06', 'TRD_A_08', 'TRD_A_11', 'TRD_A_12', 'TRD_A_13',
    'TRD_B_01', 'TRD_B_03', 'TRD_B_04', 'TRD_B_06', 'TRD_B_07', 'TRD_B_08', 'TRD_B_15', 'TRD_B_19', 'TRD_B_20', 'TRD_B_21',
    'TRD_B_24', 'TRD_B_25', 'TRD_B_33', 'TRD_B_34', 'TRD_B_36', 'TRD_B_53',
    'TRD_C_02', 'TRD_C_03', 'TRD_C_05', 'TRD_C_10',
    'TRD_P_01', 'TRD_P_02', 'TRD_P_08', 'TRD_P_09', 'TRD_P_10', 'TRD_P_11', 'TRD_P_38',
    'TRD_O_01', 'TRD_O_05', 'TRD_O_07',
    'E2_G_01', 'E2_G_02', 'E2_G_05', 'E2_G_06', 'E2_G_08', 'E2_G_09', 'E2_G_10',
    'E2_H_01', 'E2_H_07', 'E2_H_08', 'E2_H_09',
    'SPA_A_02',
    'SPA_B2_19',
    'SPA_F1_30', 'SPA_F2_32', 'SPA_F3_34',
    'SPA_G_37',
    'E4_Q_03', 'E4_Q_04', 'E4_Q_17',
    'ND_ERL_01',
    'AGE_MOST_RECENT'
]
```

In [ ]:

```python
# Experian provided attribute segmentation

CAIS_Status = ['E1_B_07', 'E2_H_08', 'E2_H_07', 'E1_B_08', 'ND_HAC_09']  # U, T, N (-1) 0..8 - OHE

Age = ['SP_G_37', 'VM04_SP_VM1_15', 'VM08_SP_VM2_15', 'SPA_G_37', 'E2_G_06', 'E1_A_11', 'ND_ECC_07', 'E1_A_06', 'E1_A_03', 'SP_G_38']

Arrears = ['HC_A_03', 'HC_P_07', 'VM07_SP_VM1_04']

Value100 = ['E2_G_10', 'E1_A_10', 'E2_G_05', 'EA1_D_02']

Count = ['VM04_SP_VM1_24', 'E1_E_02', 'E2_G_01', 'ND_PSD_04', 'VM01_SP_VM2_32', 'SP_B1_11', 'TRD_STL_23', 'VM08_SP_VM2_07', 'SP_F1_30',
         'VM01_SP_VM2_18', 'VM08_SP_VM2_18', 'VM10_SP_VM2_32', 'SP_B1_16', 'EA1_E_04', 'ND_INC_03', 'VM07_SP_VM1_25', 'VM04_SP_VM1_23',
         'E1_B_01', 'VM07_SP_VM1_26', 'E1_D_01', 'E1_B_09', 'E1_A_08', 'VM04_SP_VM1_08', 'E1_A_04', 'E2_H_09', 'VM01_SP_VM2_25', 'E1_A_07',
         'VM05_SP_VM1_07', 'E1_E_01', 'ND_ECC_03', 'VM04_SP_VM1_07', 'E1_A_09', 'PD_B_05', 'VM08_SP_VM2_16', 'ND_PSD_01', 'EA1_E_01',
         'PD_A_01', 'E2_G_08', 'VM10_SP_VM2_08', 'PD_D_15', 'SPA_F1_30', 'HC_C_01', 'HC_D_02', 'ND_PSD_11'] + ['TRD_A_11', 'TRD_O_05', 'TRD_A_06', 'TRD_A_08']

Year_ER = ['E4_Q_04', 'E4_Q_03']

Payment = ['TRD_P_02', 'TRD_P_08', 'SP_B3_22', 'SP_B3_23']

Actual_Limits = ['VM08_SP_VM2_20', 'VM07_SP_VM1_09', 'SP_F2_33', 'SP_E1_28', 'SP_A_10', 'SPA_F2_32', 'PD_A_03', 'VM10_SP_VM2_09', 'VM01_SP_VM2_20']

CLU2 = ['VM10_SP_VM2_22', 'VM10_SP_VM2_23', 'VM08_SP_VM2_23']

PTSBR = ['TRD_P_11', 'TRD_P_10']

CLU3 = ['TRD_C_02', 'TRD_C_05']

Time_Since = ['TRD_A_13', 'VM02_SP_VM1_18', 'HC_D_01', 'VM04_SP_VM1_18', 'VM10_SP_VM2_14', 'VM07_SP_VM1_14', 'AGE_MOST_RECENT',
              'PD_A_02', 'HC_C_05', 'HC_B_03', 'TRD_STL_22', 'TRD_STL_14'] + ['TRD_B_53', 'TRD_O_07']

Arrears_Balance_Traj = ['TRD_B_33', 'TRD_B_34', 'TRD_B_36', 'TRD_A_04', 'TRD_B_24'] + ['TRD_C_10']

Precentage_Change = ['TRD_B_04', 'TRD_B_01', 'TRD_B_03', 'TRD_B_20', 'TRD_B_19']

Balance_Trend = ['SPA_B2_19', 'SP_B2_19']

CLU1 = ['SP_F3_35', 'SP_F3_34', 'SP_F3_36', 'SPA_F3_34']

Numerical_with_neg = Age + Arrears + Value100 + Count + Payment + Actual_Limits + PTSBR + CLU2 + CLU3 + Time_Since + Year_ER
neg_lookup = {-1: 'No_Data', -2: 'No_CAIS', -3: 'No_Account', -4: 'Zero_Limit', 'T' : 'No_Data', 'N' : 'No_CAIS', 'D' : 'Dormant', 'U' : 'Unknown'}
CAIS_lookup = {'T' : 'No_Data', 'N' : 'No_CAIS', 'D' : 'Dormant', 'U' : 'Unknown'}

Numerical_with_large_neg = Arrears_Balance_Traj + Precentage_Change
large_neg_lookup = {-999998: 'No_Enough_Data', -999997: 'No_Account'}

balance_trd_lookup = {9997: 'Unknown_Avg', 9998: 'Avg_Below500', 9999: 'No_CAIS'}

OHE_features = ['TRD_O_01', 'ND_ERL_01', 'E4_Q_17']
```

In [5]:
```python
# Read the Delphi Data from the specified path

# Reading the data in chunks due to its large size
df_iter = bucket.read_data_assets_csv(
    security_classification="**",
    subpath='**',
    usecols=delphi_features + ['UNIQUEID', 'RETRO_DATE'],  # Specify columns to be read
    chunksize=300000,  # Number of rows per chunk
    iterator=True,  # Return an iterator for reading the data in chunks
    low_memory=False
)

# Print a success message
print("Data successfully read from S3 bucket.")

# Initialize an empty DataFrame to store the concatenated data chunks
data_asset_delphi = pd.DataFrame()

# Iterate over each data chunk
for i, chunk in tqdm(enumerate(df_iter)):

    # Concatenate the chunk into the main DataFrame
    data_asset_delphi = pd.concat([data_asset_delphi, chunk])
    del chunk
    gc.collect()  # Clear memory

    # Update total number of unique IDs
    N = data_asset_delphi.UNIQUEID.unique().shape[0]

# Print the total count of unique IDs and the total number of CAIS accounts
print(['\nTotal UNIQUEIDs:', N, 'Total CAIS Accounts:', len(data_asset_delphi)])
```

```
Data successfully read from S3 bucket.
['\nTotal UNIQUEIDs:', 320733, 'Total CAIS Accounts:', 320733]
```

In [ ]:
```python
# Merge the Delphi Data with the Flag Data on 'UNIQUEID'
data_asset_delphi = pd.merge(data_asset_delphi, data_flag, on='UNIQUEID')
```

3. Data Pre-Processing

Note:
Data Pre-Processing Guidelines for Handling the Delphi Data Were Provided by Experian.

In [62]:

```python
def preprocess_non_numeric(df_in: pd.DataFrame):
    """
    Preprocess non-numeric columns in the input DataFrame by converting specified categorical values to numeric.

    Parameters:
    df_in (pd.DataFrame): The input DataFrame containing the columns to preprocess.

    Returns:
    pd.DataFrame: The DataFrame with the specified non-numeric columns transformed.
    """
    for f in ['E1_B_07', 'E1_B_08', 'VM07_SP_VM1_04', 'HC_A_03', 'HC_P_07', 'E2_H_07', 'E2_H_08']:
        # Replace specified categorical values with -1
        df_in[f] = df_in[f].apply(lambda x: -1 if x in ['U', 'T', 'D', 'N'] else x)
        # Convert the column to np.float16 to save memory
        df_in[f] = df_in[f].astype(np.float16)
```

In [63]:

```python
# List to store binary feature names
Binary_Features = []

# Features that are scaled
Scale_Features = CAIS_Status + Numerical_with_neg + Numerical_with_large_neg + Balance_Trend + CLU1

def encode_features(feature_list, lookup_dict, title=None):
    """
    Encode features in the DataFrame by creating binary columns for each category defined in the lookup dictionary.

    Parameters:
    feature_list (list): List of feature names to encode.
    lookup_dict (dict): Dictionary mapping category values to new column names.
    title (str, optional): Title for the progress bar description.
    """
    for f in tqdm(feature_list, desc=title, ncols=1000, position=0, leave=True, unit=' feature'):
        # Iterate over each value in the lookup dictionary
        for v in lookup_dict.keys():
            x = data_asset_delphi[f] == v
            if np.any(x):
                # Create a binary column for each category and update Binary_Features list
                data_asset_delphi[f+'__'+lookup_dict[v]] = np.int16(x)
                data_asset_delphi.loc[x, f] = 0
                Binary_Features.append(f+'__'+lookup_dict[v])
        # Convert the original feature column to np.float32
        data_asset_delphi[f] = np.float32(data_asset_delphi[f])

# Encode features with specific lookup dictionaries
encode_features(CAIS_Status, CAIS_lookup, 'Encode CAIS')
encode_features(Numerical_with_neg, neg_lookup, 'Numeric features with negative codes')
encode_features(Numerical_with_large_neg, large_neg_lookup, 'Numeric features with LARGE negative codes')
encode_features(Balance_Trend, balance_trd_lookup, 'Balance Trend features')
```

```
Encode CAIS: 100%|██████████████████████████████| 5/5 [00:00<00:00, 258.73 feature/s]
Numeric features with negative codes: 100%|██████████████████████████████| 101/101 [00:00<00:00, 291.95 feature/s]
Numeric features with LARGE negative codes: 100%|██████████████████████████████| 11/11 [00:00<00:00, 91.56 feature/s]
Balance Trend features: 100%|██████████████████████████████| 2/2 [00:00<00:00, 60.31 feature/s]
```

In [64]:

```python
# Encode Credit Limit Utilisation (CLU) features by creating a binary column for specific values

for f in tqdm(CLU1, desc='Credit Limit Utilisation Features', ncols=1000, unit=' feature'):
    # Check if the feature column contains the values 9997 or 9999
    x = data_asset_delphi[f].isin([9997, 9999])
    if np.any(x):
        # Create a binary column indicating the presence of these values
        data_asset_delphi[f+'__No_CAIS'] = np.int16(x)
        # Set the original feature column values to 0 where the condition is met
        data_asset_delphi.loc[x, f] = 0
        # Append the new binary feature name to the Binary_Features list
        Binary_Features.append(f+'__No_CAIS')
```

```
Credit Limit Utilisation features: 100%|██████████████████████████████| 4/4 [00:00<00:00, 189.79 feature/s]
```

In [65]:

```python
# Create a binary target column where 'B' in GB_FLAG is encoded as 1 and all other values as 0
data_asset_delphi['target'] = np.int16(data_asset_delphi.GB_FLAG == 'B')
```

In [66]:
```python
# Apply One-Hot Encoding to specified features
for f in OHE_features:
    # Initialize OneHotEncoder with specified parameters
    ohe = OneHotEncoder(sparse_output=False, dtype=np.uint8)

    # Fit the encoder on the feature column and transform the data
    ohe.fit(data_asset_delphi[f].values.reshape(-1, 1))
    data = ohe.transform(data_asset_delphi[f].values.reshape(-1, 1))

    # Get the names of the new one-hot encoded features
    _ohe_features = list(ohe.get_feature_names_out([f]))

    # Add the one-hot encoded features to the DataFrame
    data_asset_delphi.loc[:, _ohe_features] = data

    # Clean up temporary variables
    del data, ohe

    # Add the names of the new one-hot encoded features to the Binary_Features list
    Binary_Features.extend(_ohe_features)
```

4. Data Modelling - Logistic Regression

In [75]:

```python
%%time

# Define the features to be used for model training (excluding the target column)
features = data_asset_delphi.columns.drop('target')

# Initialize lists to store ROC AUC test scores
score_test = []

# Set parameters for Logistic Regression
param = {'C': 0.1,
         'class_weight': {1: 0.8, 0: 0.2},
         'max_iter': 10000,
         'n_jobs': -1}

# Create an empty DataFrame to store model results
results = pd.DataFrame()

# Set up Stratified K-Fold cross-validation with 10 folds
kfold = StratifiedKFold(n_splits=10, random_state=217, shuffle=True)

# Iterate through each fold in the cross-validation
for i, (itrain, itest) in enumerate(kfold.split(X=np.zeros(len(data_asset_delphi)), y=data_asset_delphi.target)):

    # Split data into training and test sets based on current fold indices
    train = data_asset_delphi.iloc[itrain]
    test = data_asset_delphi.iloc[itest]

    # Standardize features using StandardScaler
    scaler = StandardScaler()
    train[features] = scaler.fit_transform(train[features])
    test[features] = scaler.transform(test[features])

    # Initialize and train the Logistic Regression model
    model = LogisticRegression(**param)
    model.fit(train[features], train.target)

    # Predict probabilities for the test set
    t = model.predict_proba(test[features])[:, 1]

    # Append predictions, true labels, and fold index to the results DataFrame
    results = pd.concat([results, pd.DataFrame({'pred_target': t, 'target': test.target.values, 'fold': i})])

    # Compute and store the ROC AUC score for the current fold
    score_test.append(roc_auc_score(y_true=test.target, y_score=t))

# Print the results DataFrame containing predictions, true labels, and fold information
print(results)

# Clean up memory by deleting temporary variables and invoking garbage collection
del test, train
gc.collect()
```

```
     pred_target  target  fold
0   7.288690e-07       0     0
1   7.058109e-11       0     0
2   1.224628e-06       0     0
3   8.772239e-01       0     0
4   1.076075e-08       0     0
..           ...     ...   ...
86  1.202777e-12       0     9
87  2.441750e-10       0     9
88  2.332908e-06       0     9
89  8.071920e-05       0     9
90  7.455574e-06       0     9

[919 rows x 3 columns]
CPU times: user 2.43 s, sys: 502 ms, total: 2.93 s
Wall time: 13.1 s
```

Out[75]: 8501

In [76]:

```python
def results_report(y_true, y_pred_prob, plab='Pos', nlab='Neg', report_title='Model Performance Report',
                   w=None, model_name=None, prob_threshold=0.5):
    """
    Generates a performance report for a binary classification model.

    Parameters:
    y_true : True binary labels.
    y_pred_prob : Predicted probabilities for the positive class.
    plab : str, optional (default='Pos')
        Label for the positive class in plots.
    nlab : str, optional (default='Neg')
        Label for the negative class in plots.
    report_title : str, optional (default='Model Performance Report')
        Title of the performance report.
    w : array-like, optional (default=None)
        Sample weights.
    model_name : str, optional (default=None)
        Name of the model to log metrics.
    prob_threshold : float, optional (default=0.5)
        Threshold to convert probabilities to binary predictions.
    """
    print(f'----------------------------------------- {report_title} -----------------------------------------')

    # Convert predicted probabilities to binary predictions based on the threshold
    y_pred = np.int8(y_pred_prob >= prob_threshold)

    fig = plt.figure(figsize=(10, 3.5))

    # Plot Confusion Matrix
    plt.subplot(1, 2, 1)
    mc = confusion_matrix(y_true, y_pred)
    mc = mc.astype(float)
    mc[0, :] = mc[0, :] / (len(y_true) - sum(y_true))  # Normalize confusion matrix for class 0
    mc[1, :] = mc[1, :] / sum(y_true)  # Normalize confusion matrix for class 1

    # Create a heatmap for the confusion matrix
    heatmap = sns.heatmap(mc, annot=True, annot_kws={'size':12}, fmt='0.4f')
    heatmap.yaxis.set_ticklabels([nlab, plab], rotation=90, ha='right', fontsize=12)
```

```python
    heatmap.xaxis.set_ticklabels([nlab, plab], rotation=0, ha='right', fontsize=12)
    plt.ylabel('True Label', fontsize=12)
    plt.xlabel('Predicted Label', fontsize=12)

    # Plot ROC Curve
    plt.subplot(1, 2, 2)
    fpr, tpr, threshold = roc_curve(y_true, y_pred_prob, sample_weight=w)
    roc_auc = auc(fpr, tpr)

    # Calculate and print performance metrics
    accuracy = accuracy_score(y_true, y_pred, sample_weight=w)
    precision = precision_score(y_true, y_pred, sample_weight=w)
    recall = recall_score(y_true, y_pred, sample_weight=w)
    f1 = f1_score(y_true, y_pred, sample_weight=w)
    gini_coefficient = 2 * roc_auc - 1

    # Prepare and display the metrics table
    table_data = [
        ['Accuracy', accuracy],
        ['Precision', precision],
        ['Recall', recall],
        ['F1 Score', f1],
        ['AUC', roc_auc],
        ['Gini Coefficient', gini_coefficient]
    ]
    print(tabulate(table_data, headers=['Metric', 'Score'], tablefmt='fancy_grid'))

    # Log the AUC and Gini Coefficient if model_name is provided
    if model_name is not None:
        logvalue(model_name + '_auc', roc_auc)
        logvalue(model_name + '_gini', gini_coefficient)

    # Plot ROC Curve
    plt.title('ROC Curve', fontsize=12)
    plt.plot(fpr, tpr, 'b', label=f'AUC = {roc_auc:.4f}')
    plt.legend(loc='lower right', fontsize=12)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate (Sensitivity)', fontsize=12)
    plt.xlabel('False Positive Rate (1-Specificity)', fontsize=12)

    plt.show()

# Generate and display the model performance report
results_report(results.target, results.pred_target, plab='Bad', nlab='Good', report_title='Model Performance Report', prob_threshold=0.5)
```

-- Model Performance Report --

Metric	Score
Accuracy	0.89445
Precision	0.660256
Recall	0.70068
F1 Score	0.679868
AUC	0.904683
Gini Coefficient	0.809365