SBA (Small Biz Admin.) Loan Approval Analysis & Prediction

by Jungseok Lee

In [1]:
```
from IPython.display import display, HTML
display(HTML("<style>.container { width:80% !important; }</style>"))

import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import shap
import h2o
from h2o.estimators import H2OTargetEncoderEstimator
from h2o.estimators import H2OGradientBoostingEstimator
try:
    h2o.cluster().shutdown()
except:
    pass
```

In [2]:
```
h2o.init(max_mem_size=8)
```

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
  Starting server from C:\Users\wizdo\ml-spring-2024\Lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\wizdo\AppData\Local\Temp\tmpjnaynjql
  JVM stdout: C:\Users\wizdo\AppData\Local\Temp\tmpjnaynjql\h2o_wizdo_started_from_python.out
  JVM stderr: C:\Users\wizdo\AppData\Local\Temp\tmpjnaynjql\h2o_wizdo_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
Warning: Your H2O cluster version is (4 months and 8 days) old.  There may be a newer version available.
Please download and install the latest version from: https://h2o-release.s3.amazonaws.com/h2o/latest_stable.html
```

H2O_cluster_uptime:	02 secs
H2O_cluster_timezone:	America/Chicago
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.44.0.3
H2O_cluster_version_age:	4 months and 8 days
H2O_cluster_name:	H2O_from_python_wizdo_u65zw3
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	7.984 Gb
H2O_cluster_total_cores:	0
H2O_cluster_allowed_cores:	0
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://127.0.0.1:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.10.11 final

In [3]:
```
data = h2o.import_file('./data/SBA_loans_project_2.csv')
```

```
Parse progress: |████████████████████████████████████████████████████████████████| (done) 100%
```

Display a preview of the dataset

(check missing values and types of features)

In [4]:
```
data.describe()
```

Rows:799356

Cols:20

	index	City	State	Zip	Bank	BankState	NAICS	NoEmp	
type	int	enum	enum	int	enum	enum	int	int	
mins	0.0			0.0			0.0	0.0	
mean	399677.5			53800.14742367606			398464.7250036278	11.394357457753493	1.28043
maxs	799355.0			99999.0			928120.0	9999.0	
sigma	230754.34522669343			31185.719098605336			263323.9798013513	73.98731938246111	0.45180
zeros	1			249			179717	5914	
missing	0	25	12	0	1402	1408	0	0	
0	0.0	FORT LEE	NJ	7024.0	BNB HANA BANK NATL ASSOC	NJ	425120.0	2.0	
1	1.0	WESTWEGO	LA	70094.0	JEDCO DEVELOPMENT CORPORATION	LA	812331.0	62.0	
2	2.0	DENVER	CO	80209.0	WELLS FARGO BANK NATL ASSOC	SD	541611.0	4.0	
3	3.0	WRANGELL	AK	99929.0	FIRST BANK	AK	446110.0	3.0	
4	4.0	MALVERN	AR	72104.0	CITICAPITAL SMALL BUS. FINANCE	TX	0.0	1.0	
5	5.0	HAMILTON SQUARE	NJ	8619.0	SUN NATIONAL BANK	NJ	445110.0	3.0	
6	6.0	PHOENIX	AZ	85040.0	MUTUAL OF OMAHA BANK	AZ	0.0	1.0	
7	7.0	DENVER	CO	80207.0	JPMORGAN CHASE BANK NATL ASSOC	IL	722211.0	11.0	
8	8.0	CLEVELAND	OH	44109.0	U.S. BANK NATIONAL ASSOCIATION	OH	445299.0	4.0	
9	9.0	ESCONDIDO	CA	92025.0	COMERICA BANK	TX	621210.0	4.0	

[799356 rows x 20 columns]

Clean up (Encode replace missing values)

In [5]:
```
columns = ["City","State","Bank","BankState",
            "NewExist", "RevLineCr","LowDoc"]
num_col = data.columns_by_type(coltype="numeric")
enum_col = data.columns_by_type(coltype="categorical")
all_columns = data.columns

for col in columns:
    if all_columns.index(col)*1.0 in num_col:
        print("Fillna for numerical column:...", col)
        '''Alternative way '''
        data[data[col].isna(), col] = 0
    elif all_columns.index(col)*1.0 in enum_col:
        print("Fillna for categorical column:...", col)
        data[col] = data[col].ascharacter()
        data[data[col].isna(), col] = "Missing"
        data[col] = data[col].asfactor()
```

```
Fillna for categorical column:... City
Fillna for categorical column:... State
Fillna for categorical column:... Bank
Fillna for categorical column:... BankState
Fillna for numerical column:... NewExist
Fillna for categorical column:... RevLineCr
Fillna for categorical column:... LowDoc
```

In [ ]:
```
# missing values were filled with 0 or 'Missing'
```

Split dataset to Train/Valid/Test

In [6]:
```
train,valid,test = data.split_frame(ratios=[.6, .2], seed=1234)
```

Change to categorical variables for feature engineering

In [7]:
```
cat_columns = ["City","State","Bank","BankState", "UrbanRural", "FranchiseCode",
               "NewExist", "RevLineCr","LowDoc", "Zip", "NAICS"]
```

```
encoded_columns = cat_columns
response = "MIS_Status"

train[encoded_columns+[response]] = train[encoded_columns+[response]].asfactor()
valid[encoded_columns+[response]] = valid[encoded_columns+[response]].asfactor()
test[encoded_columns+[response]] = test[encoded_columns+[response]].asfactor()
```

Add engineered features

(Excluded some features that didn't improve the model performance)

In [8]:
```
train_0 = train
```

In [9]:
```
import numpy as np

column_to_bin = col = "BalanceGross"
col_np_array = np.array(train_0[column_to_bin].as_data_frame(use_pandas=True)[column_to_bin].values, dtype=np.int64) # Convert sin

counts, breaks = np.histogram(col_np_array, bins=5)
```

```
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

In [10]:
```
min_val = min(col_np_array)-1                            # Establish min and max values
max_val = max(col_np_array)+1

new_b = [int(min_val)]                                    # Redefine breaks such that each bucket has enough support
for i in range(4):
    if counts[i] > 1000 and counts[i+1] > 1000:
        new_b.append(int(breaks[i+1]))
new_b.append(int(max_val))

names = [col + '_' + str(x) for x in range(len(new_b)-1)]  # Generate names for buckets, these will be categorical names
```

In [11]:
```
train[col+"_cut"] = train[col].cut(breaks=new_b, labels=names)
```

In [12]:
```
def cut_column(col_np_array, train, valid, test, col):

    counts, breaks = np.histogram(col_np_array, bins=100)   # Generate counts and breaks for our histogram
    min_val = min(col_np_array)-1                             # Establish min and max values
    max_val = max(col_np_array)+1

    new_b = [int(min_val)]                                     # Redefine breaks such that each bucket has enough support
    for i in range(99):
```

```python
        if counts[i] > 1000 and counts[i+1] > 1000:
            new_b.append(int(breaks[i+1]))
    new_b.append(int(max_val))

    names = [col + '_' + str(x) for x in range(len(new_b)-1)]  # Generate names for buckets, these will be categorical names

    train[col+"_cut"] = train[col].cut(breaks=new_b, labels=names)
    valid[col+"_cut"] = valid[col].cut(breaks=new_b, labels=names)
    test[col+"_cut"] = test[col].cut(breaks=new_b, labels=names)
```

In [13]:

```python
def add_features(train, valid, test):

    '''
    Helper function to add binning and interaction features to the dataset
    '''

    # Transform numerical columns to categorical via binning
    for column_to_bin in [#"NoEmp",
                          #"CreateJob",
                          "RetainedJob",
                          #"DisbursementGross",
                          "BalanceGross",
                          #"GrAppv",
                          "SBA_Appv"]:
        print("Binning Column: {}".format(column_to_bin))
        col_np_array = np.array(train[column_to_bin].as_data_frame(use_pandas=True)[column_to_bin].values, dtype=np.int64) # Conve
        cut_column(col_np_array, train, valid, test, column_to_bin)

    #Add interaction columns for a subset of columns
    interaction_cols1 = [#"NoEmp_cut",
                         #"City",
                         "State",
                         "Bank",
                         "BankState",
                         "NAICS",
                         "NewExist",
                         #"CreateJob_cut",
                         "RetainedJob_cut",
                         "UrbanRural",
                         "RevLineCr",
                         "LowDoc",
                         "FranchiseCode",
                         #"DisbursementGross_cut",
                         "BalanceGross_cut",
                         #"GrAppv_cut",
                         "SBA_Appv_cut"]
```

```python
    train_cols = train.interaction(factors=interaction_cols1,    #Generate pairwise columns
                                   pairwise=True,
                                   max_factors=1000,
                                   min_occurrence=100,
                                   destination_frame="itrain")
    valid_cols = valid.interaction(factors=interaction_cols1,
                                   pairwise=True,
                                   max_factors=1000,
                                   min_occurrence=100,
                                   destination_frame="ivalid")
    test_cols = test.interaction(factors=interaction_cols1,
                                   pairwise=True,
                                   max_factors=1000,
                                   min_occurrence=100,
                                   destination_frame="itest")

    train = train.cbind(train_cols)                              #Append pairwise columns to H2OFrames
    valid = valid.cbind(valid_cols)
    test = test.cbind(test_cols)

    print("All columns added via interaction",train_cols.columns)

    return train, valid, test
```

In [14]:

```python
train_f, valid_f, test_f = add_features(train, valid, test)
```

```
Binning Column: RetainedJob
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
Binning Column: BalanceGross
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
Binning Column: SBA_Appv
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

```
Interactions progress: |████████████████████████████████████████████████████████| (done) 100%
Interactions progress: |████████████████████████████████████████████████████████| (done) 100%
Interactions progress: |████████████████████████████████████████████████████████| (done) 100%
All columns added via interaction ['State_Bank', 'State_BankState', 'State_NAICS', 'State_NewExist', 'State_RetainedJob_cut', 'State_UrbanRural', 'State_RevLineCr', 'State_LowDoc', 'State_FranchiseCode', 'State_BalanceGross_cut', 'State_SBA_Appv_cut', 'Bank_BankState', 'Bank_NAICS', 'Bank_NewExist', 'Bank_RetainedJob_cut', 'Bank_UrbanRural', 'Bank_RevLineCr', 'Bank_LowDoc', 'Bank_FranchiseCode', 'Bank_BalanceGross_cut', 'Bank_SBA_Appv_cut', 'BankState_NAICS', 'BankState_NewExist', 'BankState_RetainedJob_cut', 'BankState_UrbanRural', 'BankState_RevLineCr', 'BankState_LowDoc', 'BankState_FranchiseCode', 'BankState_BalanceGross_cut', 'BankState_SBA_Appv_cut', 'NAICS_NewExist', 'NAICS_RetainedJob_cut', 'NAICS_UrbanRural', 'NAICS_RevLineCr', 'NAICS_LowDoc', 'NAICS_FranchiseCode', 'NAICS_BalanceGross_cut', 'NAICS_SBA_Appv_cut', 'NewExist_RetainedJob_cut', 'NewExist_UrbanRural', 'NewExist_RevLineCr', 'NewExist_LowDoc', 'NewExist_FranchiseCode', 'NewExist_BalanceGross_cut', 'NewExist_SBA_Appv_cut', 'RetainedJob_cut_UrbanRural', 'RetainedJob_cut_RevLineCr', 'RetainedJob_cut_LowDoc', 'RetainedJob_cut_FranchiseCode', 'RetainedJob_cut_BalanceGross_cut', 'RetainedJob_cut_SBA_Appv_cut', 'UrbanRural_RevLineCr', 'UrbanRural_LowDoc', 'UrbanRural_FranchiseCode', 'UrbanRural_BalanceGross_cut', 'UrbanRural_SBA_Appv_cut', 'RevLineCr_LowDoc', 'RevLineCr_FranchiseCode', 'RevLineCr_BalanceGross_cut', 'RevLineCr_SBA_Appv_cut', 'LowDoc_FranchiseCode', 'LowDoc_BalanceGross_cut', 'LowDoc_SBA_Appv_cut', 'FranchiseCode_BalanceGross_cut', 'FranchiseCode_SBA_Appv_cut', 'BalanceGross_cut_SBA_Appv_cut']
```

Re-define cat_columns for target encoding

(Exclude 4 features that are not suitable)

In [15]:
```python
cat_columns = ["City","State","Bank","BankState", "FranchiseCode",
               "RevLineCr","LowDoc"] #"Zip", "NAICS", "UrbanRural","NewExist"
```

In [16]:
```python
encoder_te = H2OTargetEncoderEstimator(#fold_column=fold_column,
                                        data_leakage_handling=None,
                                        blending=True,
                                        inflection_point=15,
                                        smoothing=10,
                                        seed=1234)

encoder_te.train(x=cat_columns,
                 y=response,
                 training_frame=train_f)
```

```
targetencoder Model Build progress: |████████████████████████████████████████████| (done) 100%
```

Out[16]: Model Details

==============

H2OTargetEncoderEstimator : TargetEncoder

Model Key: TargetEncoder_model_python_1714353241713_1

Target Encoder model summary: Summary for target encoder model

original_names	encoded_column_names
City	City_te
State	State_te
Bank	Bank_te
BankState	BankState_te
FranchiseCode	FranchiseCode_te
RevLineCr	RevLineCr_te
LowDoc	LowDoc_te

[tips]

Use `model.explain()` to inspect the model.

--

Use `h2o.display.toggle_user_tips()` to switch on/off this section.

Save target encoder path

In [17]:

```
encoder_path = h2o.save_model(model=encoder_te, path="./artifacts", force=True)
```

In [18]:

```
encoder_path
```

Out[18]:

```
'C:\\Users\\wizdo\\Downloads\\artifacts\\TargetEncoder_model_python_1714353241713_1'
```

Add the features created through target encoding.

In [19]:

```
train_te = encoder_te.transform(frame=train_f, as_training=True)
valid_te = encoder_te.transform(frame=valid_f, as_training=False)
```

```python
test_te = encoder_te.transform(frame=test_f, as_training=False)
```

Prepare for model training and tuning.

In [20]:
```python
predictors = train_te.columns
```

In [21]:
```python
for col in cat_columns+[response]:
    predictors.remove(col)
```

Remove "index" column for training

In [22]:
```python
predictors.remove("index")
```

Model traing and tuning (GBM(H2O))

In [23]:
```python
best_auc = 0
best_nfolds = 0
best_ntrees = 0
best_max_depth = 0
best_stopping_rounds = 0
best_stopping_metric = []
best_model = None

for nfolds in [3, 4, 5]:
    for ntrees in [30, 50, 80, 100, 130]:
        for stopping_rounds in [3, 5]:
            for max_depth in [3, 5]:
                for stopping_metric in ["MAE"]:
                    model = H2OGradientBoostingEstimator(nfolds = nfolds,
                                                         ntrees = ntrees,
                                                         max_depth = max_depth,
                                                         stopping_rounds = stopping_rounds,
                                                         stopping_metric = stopping_metric,
                                                         seed=1234,
                                                         keep_cross_validation_predictions = False)
                    model.train(y=response, x=predictors, training_frame=train_te, validation_frame=valid_te)

                    auc = model.model_performance(valid=True).auc()
                    if auc > best_auc:
                        best_auc = auc
```

```
                        best_nfolds = nfolds
                        best_ntrees = ntrees
                        best_max_depth = max_depth
                        best_stopping_rounds = stopping_rounds
                        best_stopping_metric = stopping_metric
                        best_model = model
                        print("Better Model found. Trained H2OGBM with (auc, nfolds, ntrees, stopping_rounds, stopping_metric):(",
                              auc, nfolds, ntrees, max_depth, stopping_rounds, stopping_metric,")")

print("Best model AUC(on valid dataset):", best_auc)
print("Best model AUC(on test dataset):", best_model.model_performance(test_data=test_te).auc())
```

```
gbm Model Build progress: |███████████████████████████| (done) 100%
Better Model found. Trained H2OGBM with (auc, nfolds, ntrees, stopping_rounds, stopping_metric):( 0.8101122794405948 3 30 3 3 MAE )
gbm Model Build progress: |███████████████████████████| (done) 100%
Better Model found. Trained H2OGBM with (auc, nfolds, ntrees, stopping_rounds, stopping_metric):( 0.817585119510624 3 30 5 3 MAE )
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
Better Model found. Trained H2OGBM with (auc, nfolds, ntrees, stopping_rounds, stopping_metric):( 0.8225207586854927 3 50 5 3 MAE )
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
Better Model found. Trained H2OGBM with (auc, nfolds, ntrees, stopping_rounds, stopping_metric):( 0.825371561631955 3 80 5 3 MAE )
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
Better Model found. Trained H2OGBM with (auc, nfolds, ntrees, stopping_rounds, stopping_metric):( 0.8260393864349381 3 100 5 3 MAE
)
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
Better Model found. Trained H2OGBM with (auc, nfolds, ntrees, stopping_rounds, stopping_metric):( 0.8260729441934694 3 130 5 3 MAE
)
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████| (done) 100%
```

```
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |███████████████████████████████████████████████████████████| (done) 100%
Best model AUC(on valid dataset): 0.8260729441934694
Best model AUC(on test dataset): 0.8231652178480686
```

As a result of 60 combinations, the auc value of the valid data was the highest(0.82607) under the condition (nfolds = 3, ntrees = 130, max_depth = 5, stopping_rounds = 3, stopping_metric = MAE), so this model was set as the best model.

Final metrics using Test dataset

(Threshold calculation and Report final AUC metric and confusion matrix on the Test dataset)

In [24]:
```
best_model.model_performance(test_data=test_te)
```

Out[24]:

ModelMetricsBinomial: gbm
** Reported on test data. **

MSE: 0.11013973593898531
RMSE: 0.33187307203053595
LogLoss: 0.355642847141753
Mean Per-Class Error: 0.27068745817623496
AUC: 0.8231652178480686
AUCPR: 0.5339127974547134
Gini: 0.6463304356961372

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.23895010495437083

	0	1	Error	Rate
0	111374.0	20312.0	0.1542	(20312.0/131686.0)
1	10780.0	17066.0	0.3871	(10780.0/27846.0)
Total	122154.0	37378.0	0.1949	(31092.0/159532.0)

Maximum Metrics: Maximum metrics at their respective thresholds

metric	threshold	value	idx
max f1	0.2389501	0.5233043	231.0
max f2	0.1091954	0.6340495	307.0
max f0point5	0.4486483	0.5410389	144.0
max accuracy	0.4890380	0.8502683	130.0
max precision	0.9741083	1.0	0.0
max recall	0.0037266	1.0	399.0
max specificity	0.9741083	1.0	0.0
max absolute_mcc	0.3038338	0.4145142	202.0
max min_per_class_accuracy	0.1543333	0.7422277	278.0
max mean_per_class_accuracy	0.1453233	0.7436559	283.0

metric	threshold	value	idx
max tns	0.9741083	131686.0	0.0
max fns	0.9741083	27829.0	0.0
max fps	0.0037266	131686.0	399.0
max tps	0.0037266	27846.0	399.0
max tnr	0.9741083	1.0	0.0
max fnr	0.9741083	0.9993895	0.0
max fpr	0.0037266	1.0	399.0
max tpr	0.0037266	1.0	399.0

Gains/Lift Table: Avg response rate: 17.45 %, avg score: 16.96 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative
1	0.0100043	0.8394239	4.8496171	4.8496171	0.8464912	0.8823035	0.8464912	0.88
2	0.0200023	0.7845444	4.3497916	4.5997827	0.7592476	0.8117718	0.8028831	0.84
3	0.0300003	0.7340418	4.0193367	4.4063411	0.7015674	0.7596531	0.7691183	0.81
4	0.0400045	0.6849076	3.7727221	4.2478867	0.6585213	0.7097847	0.7414604	0.79
5	0.0500025	0.6377627	3.6314115	4.1246226	0.6338558	0.6604728	0.7199448	0.76
6	0.1000050	0.4650787	2.9855574	3.5550900	0.5211232	0.5435609	0.6205340	0.65
7	0.1500013	0.3550272	2.2669234	3.1257370	0.3956871	0.4061945	0.5455913	0.57
8	0.2000038	0.2776231	1.8069912	2.7960402	0.3154068	0.3135642	0.4880434	0.50
9	0.3000025	0.1815220	1.3833399	2.3251500	0.2414593	0.2249186	0.4058504	0.41
10	0.4000013	0.1240623	0.9829183	1.9895973	0.1715665	0.1505517	0.3472803	0.34
11	0.5	0.0866626	0.7096261	1.7336063	0.1238639	0.1040273	0.3025976	0.29
12	0.5999987	0.0621321	0.5056445	1.5289481	0.0882593	0.0734980	0.2668749	0.26
13	0.6999975	0.0451910	0.3892888	1.3661411	0.0679496	0.0531865	0.2384573	0.23
14	0.7999962	0.0322940	0.2503084	1.2266631	0.0436908	0.0385311	0.2141117	0.20

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative
15	0.8999950	0.0203699	0.1436490	1.1063290	0.0250737	0.0264381	0.1931076	0.18
16	1.0	0.0020049	0.0430920	1.0	0.0075216	0.0111794	0.1745481	0.16

In [65]:
```
best_model.model_performance(test_data=test_te).auc()
```

Out[65]: 0.8231652178480686

In [25]:
```
best_model.model_performance(test_data=test_te).find_threshold_by_max_metric("f1")
```

Out[25]: 0.23895010495437083

In [26]:
```
best_model.model_performance(test_data=test_te).confusion_matrix()
```

Out[26]:

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.23895010495437083

	0	1	Error	Rate
0	111374.0	20312.0	0.1542	(20312.0/131686.0)
1	10780.0	17066.0	0.3871	(10780.0/27846.0)
Total	122154.0	37378.0	0.1949	(31092.0/159532.0)

The threshold 0.23895 is the boundary for predicting that the predicted value of 'MIS_status' is 0 or 1 when test_te data is entered into best_model. This threshold of 0.23895 is the value that maximizes the f1 value.

Shapley values

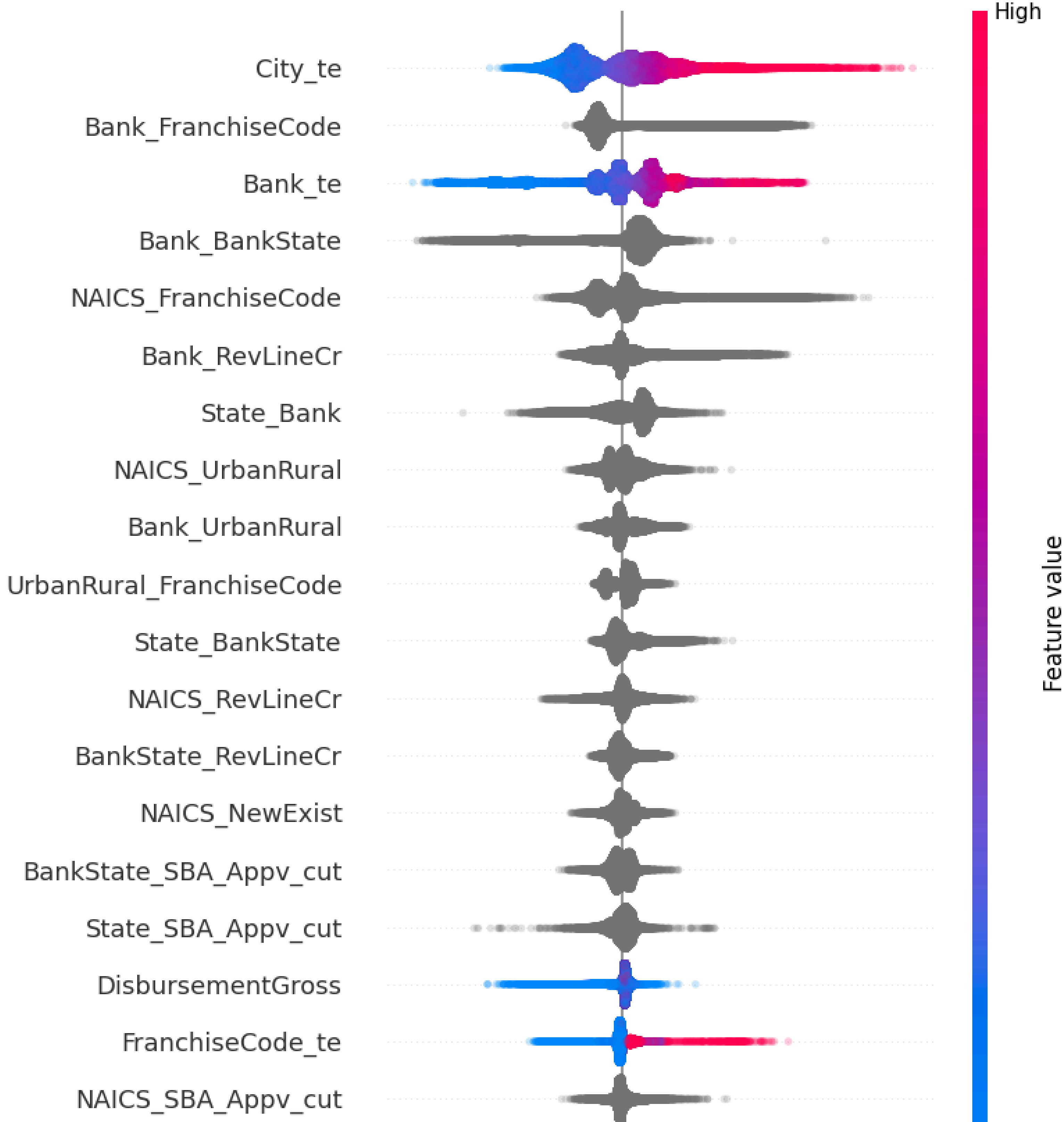
In [28]:
```
def examine_all_reason_codes(data, model):

    shap_contribs = model.predict_contributions(data)

    col_mapping = {}
    for i in data.col_names:
        related_cols = [x for x in shap_contribs.col_names if "{}.".format(i) in x]
        if len(related_cols) > 0:
            col_mapping[i] = related_cols
```

```python
    for k, v in col_mapping.items():
        if len(v) > 1:
            shap_contribs[k] = shap_contribs[v].sum(axis=1,return_frame=True)
            shap_contribs = shap_contribs.drop(v)

    shap_cols = [i for i in shap_contribs.col_names if i != "BiasTerm"]
    bias_term = shap_contribs.as_data_frame()["BiasTerm"].values

    #Convert to Pandas DF
    X = data.as_data_frame(use_pandas=True)
    shap_contribs = shap_contribs.as_data_frame(use_pandas=True)

    return shap.summary_plot(shap_contribs[shap_cols].values,
                             X[shap_cols].values,
                             feature_names=shap_cols,
                             alpha=0.2
                            )
```

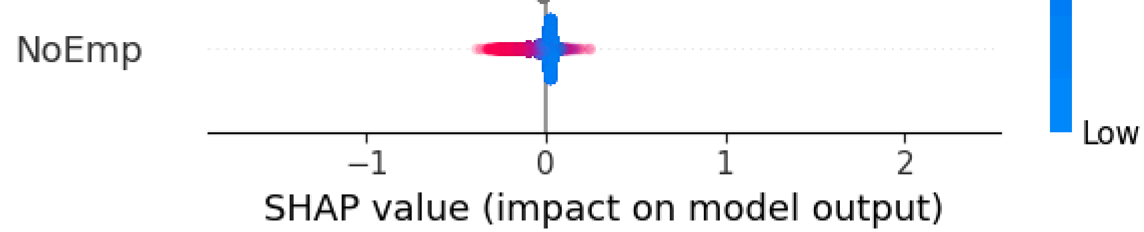
In [29]:

```python
examine_all_reason_codes(test_te, best_model)
```

```
contributions progress: |████████████████████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```



City_te
Bank_FranchiseCode
Bank_te
Bank_BankState
NAICS_FranchiseCode
Bank_RevLineCr
State_Bank
NAICS_UrbanRural
Bank_UrbanRural
UrbanRural_FranchiseCode
State_BankState
NAICS_RevLineCr
BankState_RevLineCr
NAICS_NewExist
BankState_SBA_Appv_cut
State_SBA_Appv_cut
DisbursementGross
FranchiseCode_te
NAICS_SBA_Appv_cut
High
Feature value

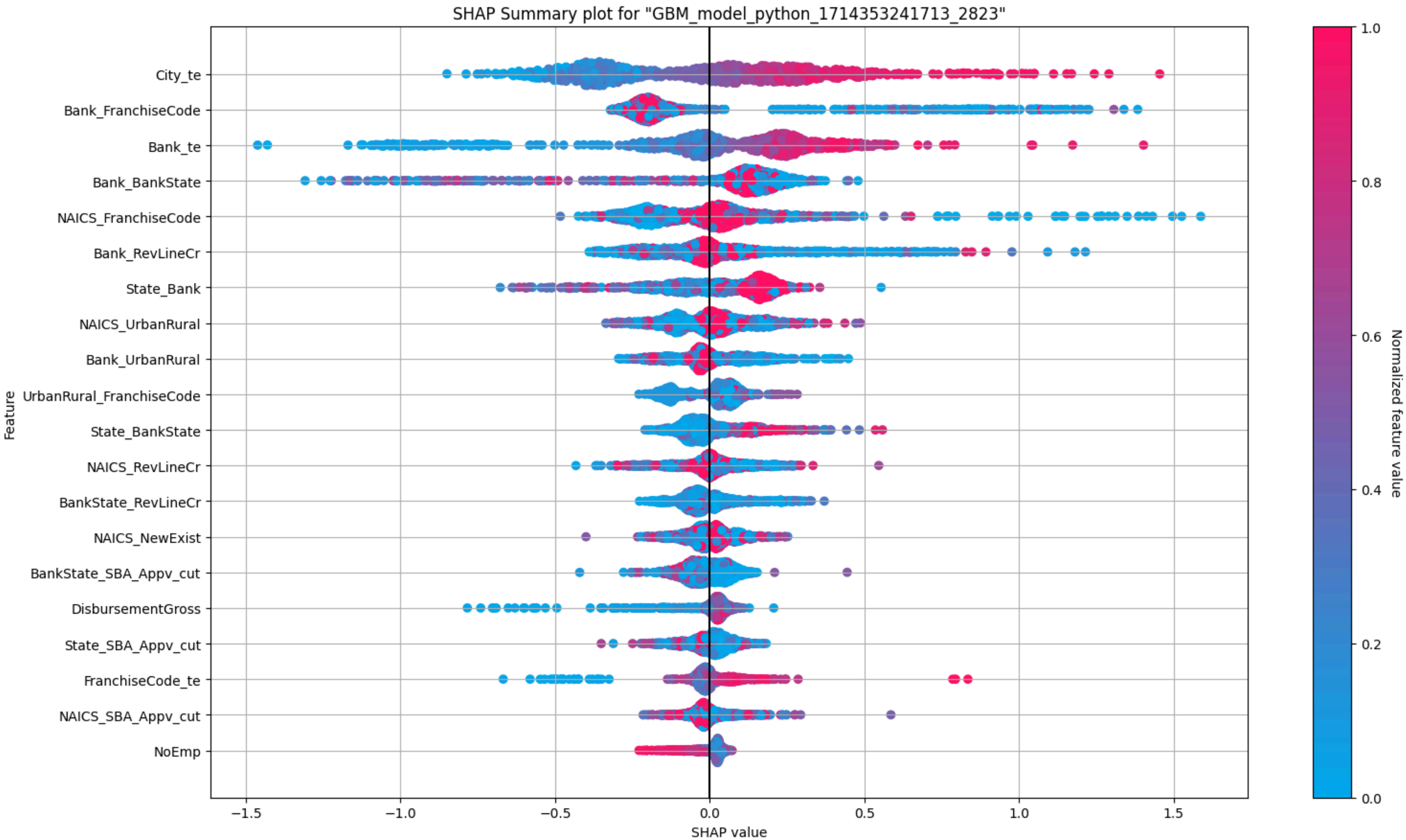


Summary plot with Shapley values

In [30]:
```
best_model.explain(test_te,include_explanations =['shap_summary']);
```

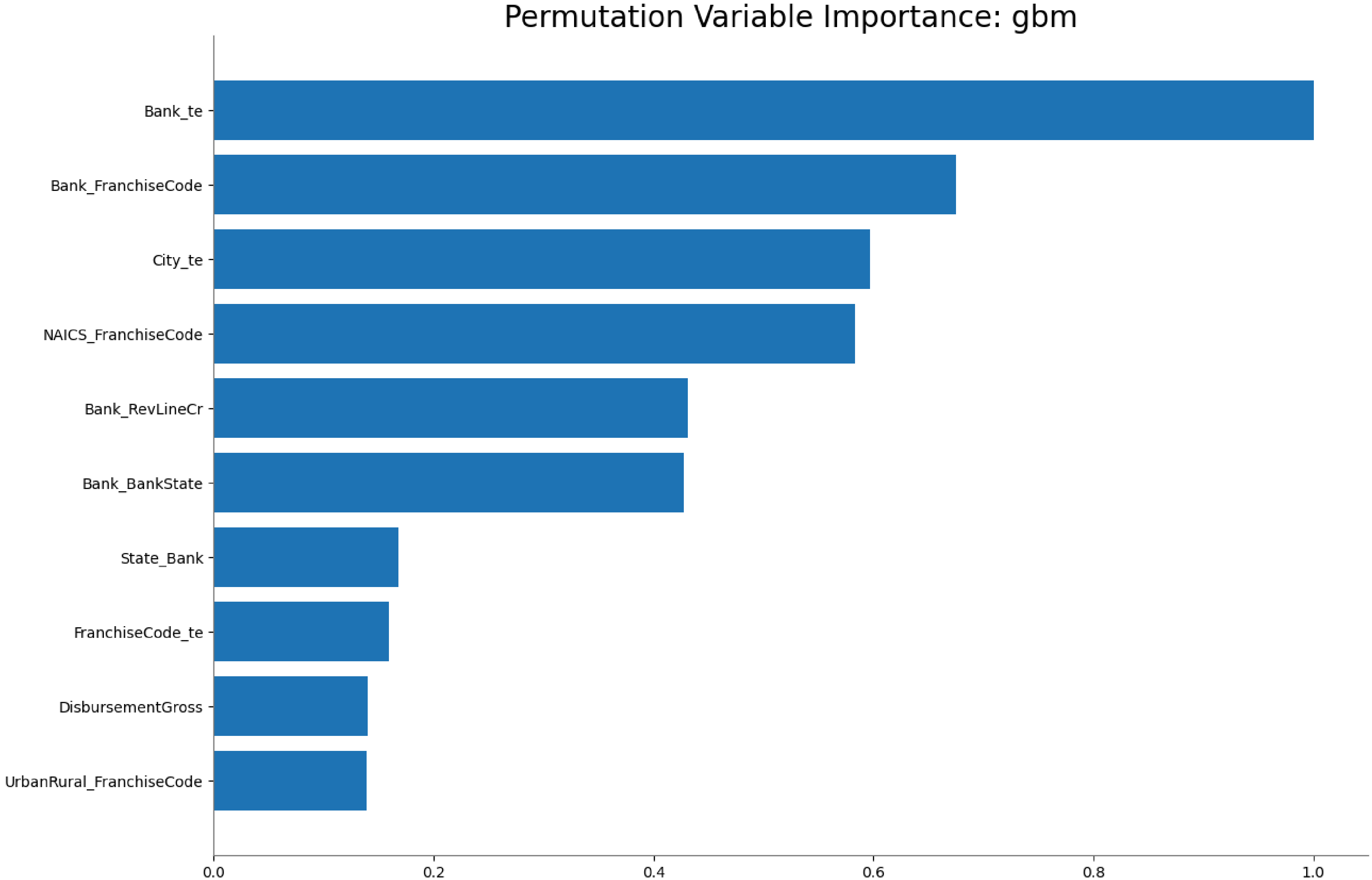
SHAP Summary

> SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function.



Permutation feature importance

In [31]:
```
best_model.permutation_importance_plot(test_te)
```


Permutation Variable Importance: gbm
Bank_te
Bank_FranchiseCode
City_te
NAICS_FranchiseCode
Bank_RevLineCr
Bank_BankState
State_Bank
FranchiseCode_te
DisbursementGross
UrbanRural_FranchiseCode
0.0
0.2
0.4
0.6
0.8
1.0

Out[31]:

Variable Importances

Variable	Relative Importance	Scaled Importance	Percentage
Bank_te	0.0284783	1.0	0.1864774
Bank_FranchiseCode	0.0192211	0.6749392	0.1258609
City_te	0.0169819	0.5963100	0.1111984
NAICS_FranchiseCode	0.0166022	0.5829771	0.1087121
Bank_RevLineCr	0.0122859	0.4314117	0.0804486
Bank_BankState	0.0121552	0.4268239	0.0795930
State_Bank	0.0047715	0.1675488	0.0312441
FranchiseCode_te	0.0045317	0.1591295	0.0296741
DisbursementGross	0.0039888	0.1400627	0.0261185
UrbanRural_FranchiseCode	0.0039447	0.1385150	0.0258299
---	---	---	---
BankState_BalanceGross_cut	0.0	0.0	0.0
BalanceGross_cut_SBA_Appv_cut	0.0	0.0	0.0
NewExist_BalanceGross_cut	0.0	0.0	0.0
BalanceGross_cut	0.0	0.0	0.0
UrbanRural_BalanceGross_cut	0.0	0.0	0.0
RevLineCr_LowDoc	0.0	0.0	0.0
BalanceGross	0.0	0.0	0.0
RetainedJob_cut_SBA_Appv_cut	0.0	0.0	0.0
RetainedJob_cut_LowDoc	0.0	0.0	0.0
NewExist_LowDoc	0.0	0.0	0.0

[87 rows x 4 columns]

What are the most important features, how they impact model predictions.

According to the Summary plot with Shapley values(or Shap summary), City_te, Bank_FranchiseCode, Bank_te are the most important features in that order. According to the Permutation feature importance graph, Bank_te, Bank_FranchiseCode, City_te are the most important features in that order.

Features affect the model prediction of 'MIS_status' by their shap values. For example, According to the Shap summary, the higher the 'City_te', the higher the shap value. In other words, Higher 'City_te' value has an influence in the direction of predicting that 'MIS_status' is 1.

Indivisual observations analysis using Shapley values.

In [39]:
```
test_te["MIS_Status"].head(20)
```

Out[39]:

MIS_Status
0
1
0
0
0
1
0
0
0
0
0
0
0
0
1
0
0
0
0
1

[20 rows x 1 column]

In [40]:
```
best_model.predict(test_te).head(20)
```

```
gbm prediction progress: |████████████████████████████████████████████████████████████████| (done) 100%
```

Out[40]:

predict	p0	p1
0	0.979175	0.0208253
0	0.952072	0.0479278
1	0.581554	0.418446
0	0.896618	0.103382
0	0.912671	0.0873286
0	0.906274	0.0937263
0	0.959727	0.0402734
1	0.66142	0.33858
0	0.96091	0.0390896
0	0.897612	0.102388
0	0.993964	0.00603563
1	0.573491	0.426509
0	0.986225	0.0137746
1	0.527061	0.472939
1	0.234944	0.765056
0	0.971343	0.0286574
0	0.973032	0.0269675
0	0.981235	0.018765
0	0.84971	0.15029
1	0.595205	0.404795

[20 rows x 3 columns]

Compare the 'MIS_Status' value of the test data and the predicted value obtained by inserting the test data into best_model, and check the correctly predicted observations(row) and incorrectly predicted observations(row) for Individual observations analysis using Shapley values.

In [41]:

```
# The function is to calculate Shapley values (contributions) and plot them for single record
def examine_indiv_reason_codes(record, model, use_matplotlib=True):
```

```
    shap_contribs = model.predict_contributions(record)

    col_mapping = {}
    for i in record.col_names:
        related_cols = [x for x in shap_contribs.col_names if "{}.".format(i) in x]
        if len(related_cols) > 0:
            col_mapping[i] = related_cols

    for k, v in col_mapping.items():
        if len(v) > 1:
            shap_contribs[k] = shap_contribs[v].sum(axis=1,return_frame=True)
            shap_contribs = shap_contribs.drop(v)

    shap_cols = [i for i in shap_contribs.col_names if i != "BiasTerm"]
    bias_term = shap_contribs.as_data_frame()["BiasTerm"].values
    X = record.as_data_frame(use_pandas=True)
    shap_contribs = shap_contribs.as_data_frame(use_pandas=True)

    return shap.force_plot(bias_term,
                           shap_contribs[shap_cols].values,
                           X[shap_cols].values,
                           shap_cols,
                           link="logit",
                           matplotlib=use_matplotlib
                          )
```

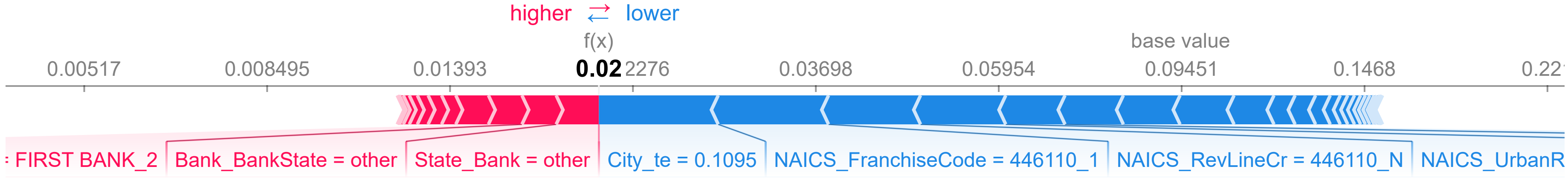
In [42]:
```
shap.initjs()
```

Label 0 is correctly identified

In [43]:
```
examine_indiv_reason_codes(test_te[0, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

Out[43]:

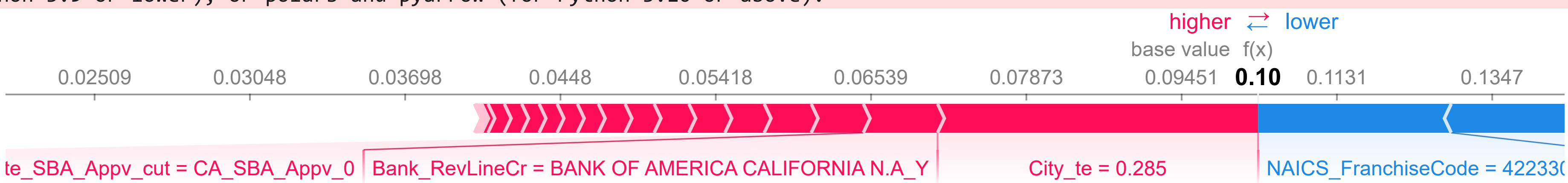


The p1 value of row0 observation is 0.02. 'State_Bank' has the most impact in the direction of predicting that 'MIS_status' is 1. 'City_te' has the most impact in the direction of predicting that 'MIS_status' is 0.

In [44]:
```
examine_indiv_reason_codes(test_te[3, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

Out[44]:



The p1 value of row3 observation is 0.10. 'City_te' has the most impact in the direction of predicting that 'MIS_status' is 1. 'NAICS_FranchiseCode' has the most impact in the direction of predicting that 'MIS_status' is 0.

Label 0 is identified as 1

In [45]:
```
examine_indiv_reason_codes(test_te[2, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

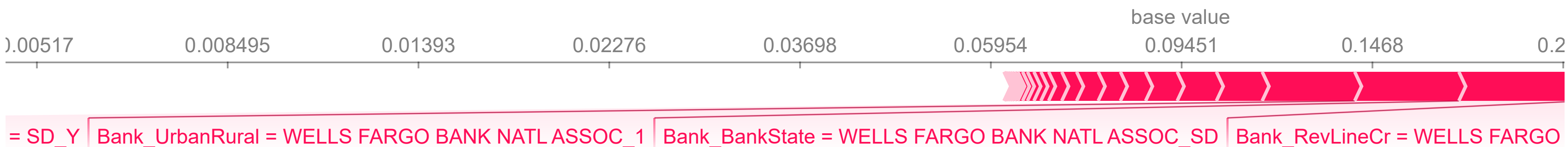
Out[45]:

The p1 value of row2 observation is 0.42. 'Bank_FranchiseCode' has the most impact in the direction of predicting that 'MIS_status' is 1. 'NAICS_FranchiseCode' has the most impact in the direction of predicting that 'MIS_status' is 0.

In [46]:
```
examine_indiv_reason_codes(test_te[7, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

Out[46]:



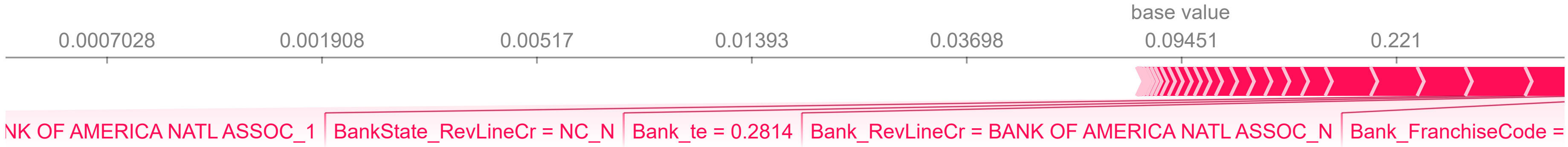
The p1 value of row7 observation is 0.34. 'Bank_RevLineCr' has the most impact in the direction of predicting that 'MIS_status' is 1. 'Bank_FranchiseCode' has the most impact in the direction of predicting that 'MIS_status' is 0.

Label 1 is correctly identified

In [48]:
```
examine_indiv_reason_codes(test_te[14, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

Out[48]:

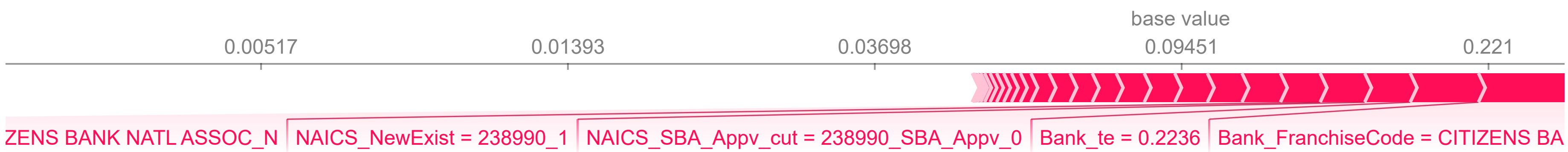


The p1 value of row14 observation is 0.77. 'Bank_FranchiseCode' has the most impact in the direction of predicting that 'MIS_status' is 1. 'BankState_NAICS' has the most impact in the direction of predicting that 'MIS_status' is 0.

In [49]:
```
examine_indiv_reason_codes(test_te[19, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

Out[49]:



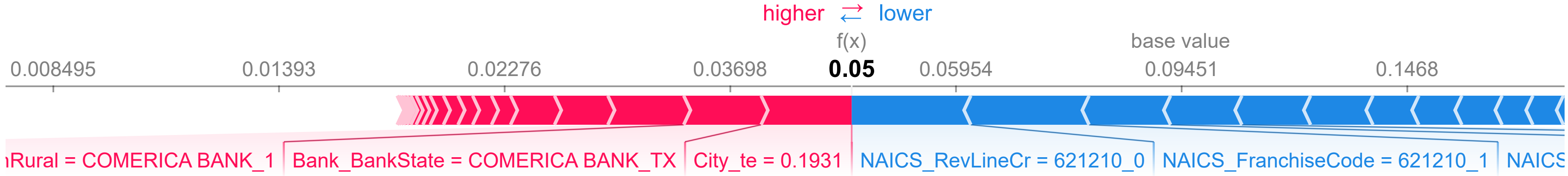
The p1 value of row19 observation is 0.40. 'Bank_FranchiseCode' has the most impact in the direction of predicting that 'MIS_status' is 1. 'City_te' has the most impact in the direction of predicting that 'MIS_status' is 0.

Label 1 is identified as 0

In [50]:
```
examine_indiv_reason_codes(test_te[1, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

Out[50]:



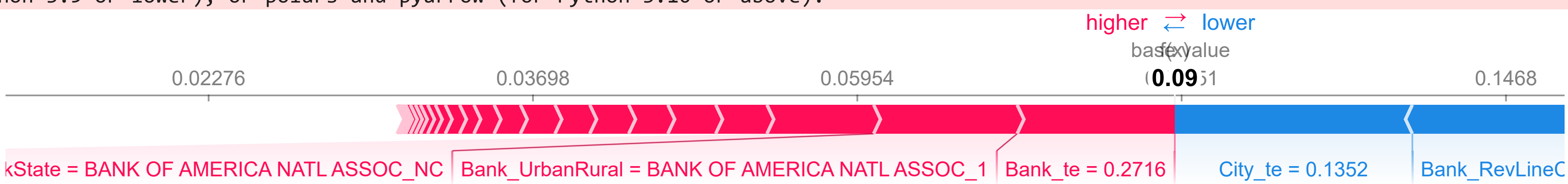
The p1 value of row1 observation is 0.05. 'City_te' has the most impact in the direction of predicting that 'MIS_status' is 1. 'NAICS_RevLineCr' has the most impact in the direction of predicting that 'MIS_status' is 0.

In [51]:

```
examine_indiv_reason_codes(test_te[5, :], best_model, use_matplotlib=False)
```

```
contributions progress: |████████████████████████████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Python 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```

Out[51]:



The p1 value of row5 observation is 0.09. 'Bank_te' has the most impact in the direction of predicting that 'MIS_status' is 1. 'City_te' has the most impact in the direction of predicting that 'MIS_status' is 0.

Residuals analysis identify and report common patterns

in the errors made by the model

In [52]:

```
predict_MIS = best_model.predict(test_te)["predict"]
predict_MIS.columns = ["predict_MIS"]
test_te_p = test_te.cbind(predict_MIS)
test_te_p[["MIS_Status", "predict_MIS"]] = test_te_p[["MIS_Status", "predict_MIS"]].asnumeric()
```

```
abs_error = abs(test_te_p["MIS_Status"] - test_te_p["predict_MIS"])

abs_error.columns = ["abs_error"]
test_te_p_abs = test_te_p.cbind(abs_error)
```

```
gbm prediction progress: |████████████████████████████████████████████████████████| (done) 100%
```

In [53]:
```
test_te_p_abs.head(1)
```

Out[53]:

City_te	State_te	Bank_te	BankState_te	FranchiseCode_te	RevLineCr_te	LowDoc_te	City	State	Bank	BankState	FranchiseCode	R
0.109508	0.111624	0.121426	0.0812697	0.127763	0.150262	0.0950888	WRANGELL	AK	FIRST BANK	AK	1	

[1 row x 98 columns]

In [54]:
```
test_te_p_abs_sort = test_te_p_abs.sort(["abs_error"])
```

In [55]:
```
response = "abs_error"
predictors_gbm = test_te_p_abs_sort.columns
predictors_gbm.remove(response)
predictors_gbm.remove("MIS_Status")
predictors_gbm.remove("predict_MIS")
predictors_gbm.remove("index")
```

In [56]:
```
res_model = H2OGradientBoostingEstimator(nfolds=best_nfolds,
                                         ntrees=best_ntrees,
                                         max_depth = max_depth,
                                         stopping_rounds=best_stopping_rounds,
                                         stopping_metric='MAE',
                                         seed=1234,
                                         keep_cross_validation_predictions = False)
res_model.train(x=predictors_gbm, y=response, training_frame=test_te_p_abs_sort)
```

```
gbm Model Build progress: |███████████████████████████████████████████████████████| (done) 100%
```

Out[56]:

Model Details
=============
H2OGradientBoostingEstimator : Gradient Boosting Machine
Model Key: GBM_model_python_1714353241713_12885

Model Summary:

number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	max_leaves	mean_leave
130.0	130.0	175900.0	5.0	5.0	5.0	22.0	32.0	30.72307

ModelMetricsRegression: gbm
** Reported on train data. **

MSE: 0.08730088487242944
RMSE: 0.2954672314698018
MAE: 0.2030319585492246
RMSLE: 0.20606227138792627
Mean Residual Deviance: 0.08730088487242944

ModelMetricsRegression: gbm
** Reported on cross-validation data. **

MSE: 0.1223206358500482
RMSE: 0.3497436716368835
MAE: 0.24192716919818685
RMSLE: 0.24607900200183777
Mean Residual Deviance: 0.1223206358500482

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid
mae	0.2419228	0.0007219	0.2425398	0.2411289	0.2420997
mean_residual_deviance	0.1223159	0.0007594	0.1230415	0.1215267	0.1223795
mse	0.1223159	0.0007594	0.1230415	0.1215267	0.1223795

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid
r2	0.1751850	0.0021526	0.1730136	0.1773184	0.1752231
residual_deviance	0.1223159	0.0007594	0.1230415	0.1215267	0.1223795
rmse	0.3497358	0.0010859	0.3507727	0.3486068	0.3498279
rmsle	0.2460746	0.0006137	0.2467145	0.2454908	0.2460186

Scoring History:

	timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance
	2024-04-28 22:19:37	27.388 sec	0.0	0.3850945	0.2965956	0.1482978
	2024-04-28 22:19:37	27.474 sec	1.0	0.3791827	0.2918395	0.1437795
	2024-04-28 22:19:37	27.536 sec	2.0	0.3739977	0.2873904	0.1398743
	2024-04-28 22:19:37	27.594 sec	3.0	0.3695775	0.2833019	0.1365876
	2024-04-28 22:19:37	27.652 sec	4.0	0.3657425	0.2795468	0.1337676
	2024-04-28 22:19:37	27.709 sec	5.0	0.3624543	0.2761083	0.1313731
	2024-04-28 22:19:37	27.767 sec	6.0	0.3595890	0.2729529	0.1293043
	2024-04-28 22:19:37	27.824 sec	7.0	0.3566522	0.2697047	0.1272008
	2024-04-28 22:19:37	27.895 sec	8.0	0.3542547	0.2668747	0.1254964
	2024-04-28 22:19:37	27.958 sec	9.0	0.3522416	0.2643809	0.1240741
---	---	---	---	---	---	---
	2024-04-28 22:19:40	30.938 sec	58.0	0.3173201	0.2213509	0.1006920
	2024-04-28 22:19:40	30.995 sec	59.0	0.3168681	0.2209577	0.1004054
	2024-04-28 22:19:40	31.051 sec	60.0	0.3163780	0.2205496	0.1000950
	2024-04-28 22:19:40	31.106 sec	61.0	0.3161196	0.2203247	0.0999316
	2024-04-28 22:19:40	31.159 sec	62.0	0.3157734	0.2200161	0.0997128
	2024-04-28 22:19:40	31.222 sec	63.0	0.3152301	0.2195304	0.0993700
	2024-04-28 22:19:40	31.276 sec	64.0	0.3149679	0.2192934	0.0992048

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance
2024-04-28 22:19:40	31.332 sec	65.0	0.3145582	0.2189551	0.0989469
2024-04-28 22:19:41	31.386 sec	66.0	0.3143027	0.2187589	0.0987862
2024-04-28 22:19:44	34.493 sec	130.0	0.2954672	0.2030320	0.0873009

[68 rows x 7 columns]

Variable Importances:

variable	relative_importance	scaled_importance	percentage
Bank_FranchiseCode	6952.7011719	1.0	0.1357536
NAICS_FranchiseCode	3892.1289062	0.5598010	0.0759950
State_Bank	3685.3530273	0.5300606	0.0719577
State_SBA_Appv_cut	3656.4809570	0.5259080	0.0713939
NAICS_UrbanRural	3463.7934570	0.4981939	0.0676316
NAICS_NewExist	2738.8505859	0.3939261	0.0534769
NAICS_RevLineCr	2468.5207520	0.3550449	0.0481986
State_BankState	2378.6149902	0.3421138	0.0464432
Bank_RevLineCr	2177.3486328	0.3131659	0.0425134
Bank_te	2018.9973145	0.2903903	0.0394215
---	---	---	---
UrbanRural_RevLineCr	0.0	0.0	0.0
UrbanRural_LowDoc	0.0	0.0	0.0
UrbanRural_BalanceGross_cut	0.0	0.0	0.0
RevLineCr_LowDoc	0.0	0.0	0.0
RevLineCr_BalanceGross_cut	0.0	0.0	0.0
LowDoc_FranchiseCode	0.0	0.0	0.0
LowDoc_BalanceGross_cut	0.0	0.0	0.0
LowDoc_SBA_Appv_cut	0.0	0.0	0.0

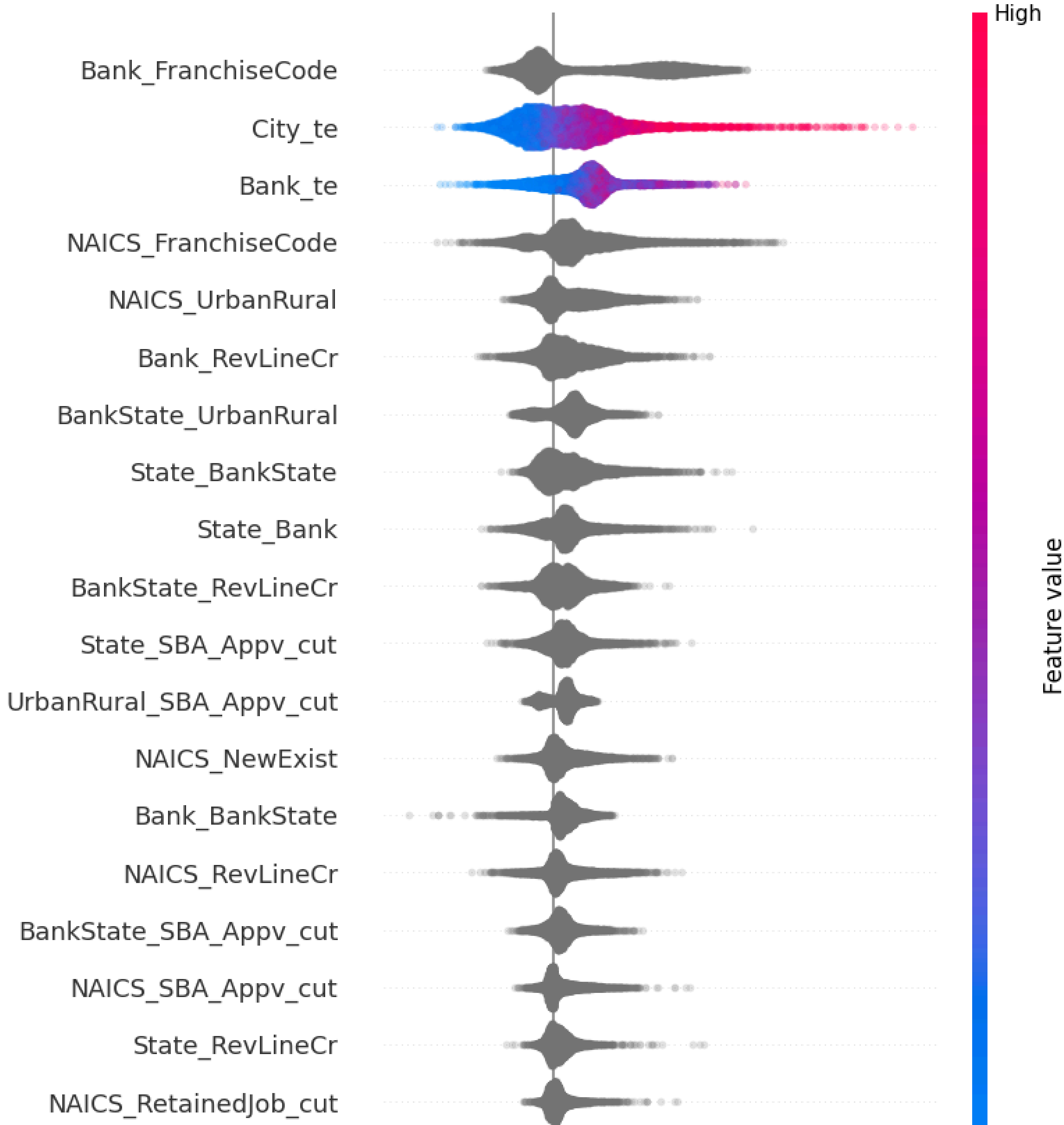
variable	relative_importance	scaled_importance	percentage
FranchiseCode_BalanceGross_cut	0.0	0.0	0.0
BalanceGross_cut_SBA_Appv_cut	0.0	0.0	0.0

[93 rows x 4 columns]

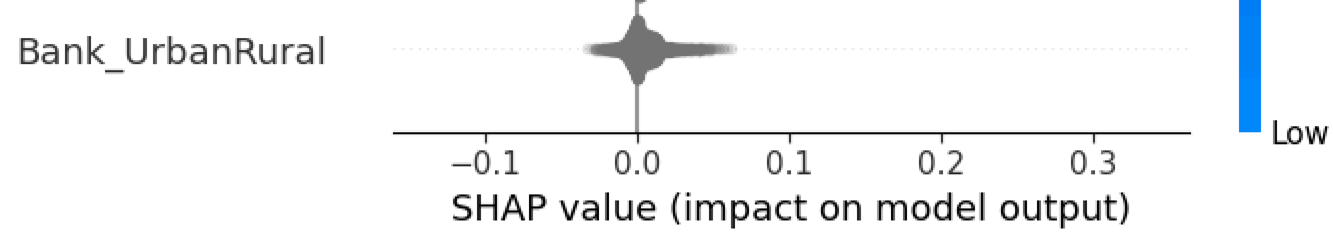
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.

In [57]:
```
examine_all_reason_codes(test_te_p_abs_sort.tail(10000),res_model)
```

```
contributions progress: |████████████████████████████| (done) 100%
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
converting H2O frame to pandas dataframe using single-thread.  For faster conversion using multi-thread, install datatable (for Pyt
hon 3.9 or lower), or polars and pyarrow (for Python 3.10 or above).
```



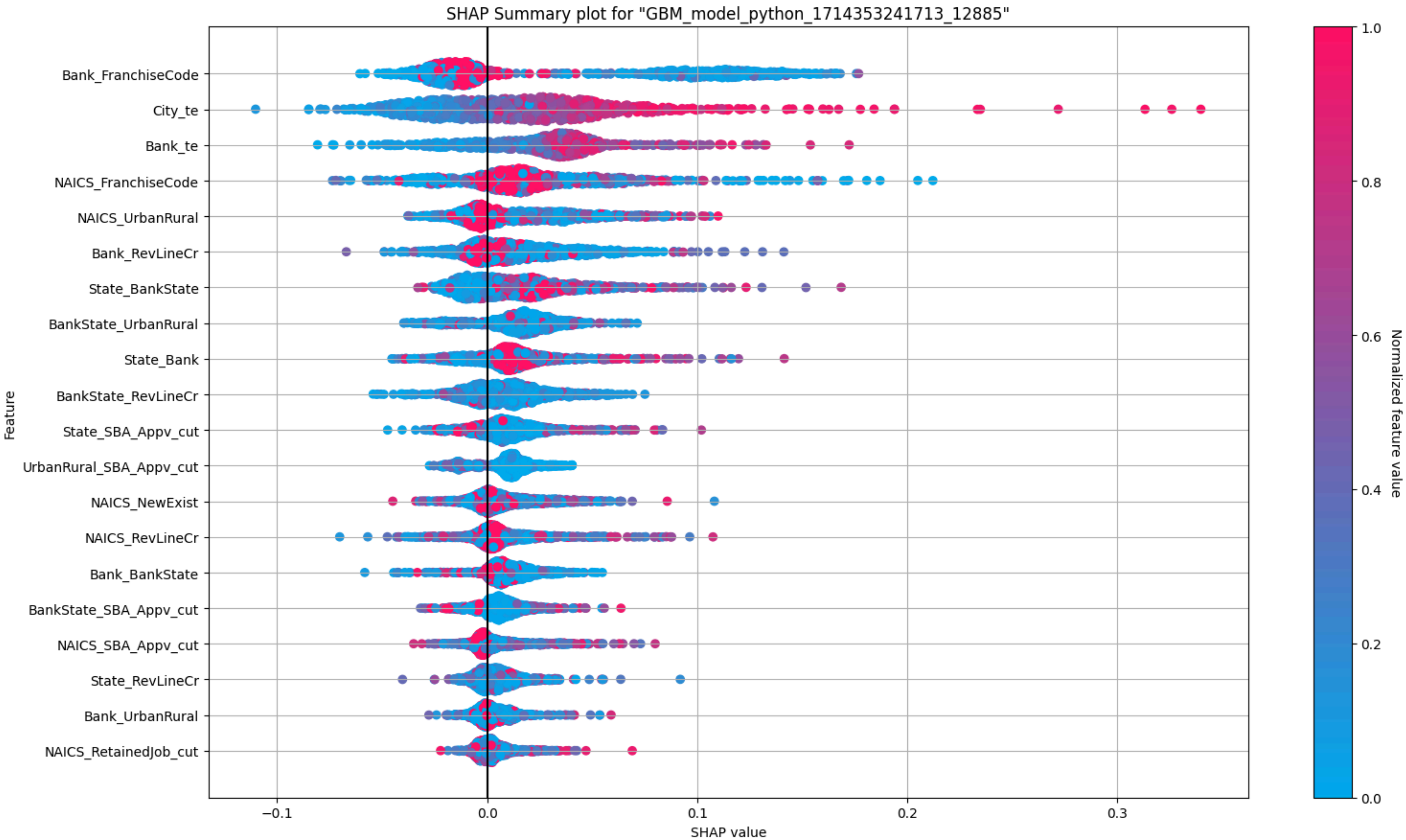
High
Feature value
Bank_FranchiseCode
City_te
Bank_te
NAICS_FranchiseCode
NAICS_UrbanRural
Bank_RevLineCr
BankState_UrbanRural
State_BankState
State_Bank
BankState_RevLineCr
State_SBA_Appv_cut
UrbanRural_SBA_Appv_cut
NAICS_NewExist
Bank_BankState
NAICS_RevLineCr
BankState_SBA_Appv_cut
NAICS_SBA_Appv_cut
State_RevLineCr
NAICS_RetainedJob_cut




```
In [58]: res_model.explain(test_te_p_abs_sort.tail(10000),include_explanations =['shap_summary']);
```

SHAP Summary

> SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function.



According to the residuals analysis, abs_error(absolute value of the difference between 'MIS_Status' value and predicted value) made by the model shows patterns like above Shap summary. In other words, Bank_FranchiseCode, City_te, Bank_te have the greatest influence on the occurrence of abs_error in that order.

Save GBM(H2O) model path

In [59]:
```
model_path = h2o.save_model(model=best_model, path="./artifacts", force=True)
```

In [60]:
```
model_path
```

Out[60]:
```
'C:\\Users\\wizdo\\Downloads\\artifacts\\GBM_model_python_1714353241713_2823'
```

Summary and Conclusion

In this project, I created and verified a model that predicts MIS_Status values using SBA_loan data. Let me explain in order. First, I loaded the data and checked the characteristics of the data, such as the type of features and missing values. And the missing values were filled with 0 or 'Missing'. Next, the dataset was divided into train, valid, and test in a 6:2:2 ratio. And as a preliminary work of feature engineering, some variables were changed to categorical variables. After that I added engineered features. Additionally, 7 variables were converted through target encoding and this target encoder was saved for the scoring funct value.

Next, I removed the index and did model training and tuning. In this process, the most appropriate parameters were found and the best model was determined. As a result of 60 combinations, the auc value of the valid data was the highest(0.82607) under the condition (nfolds = 3, ntrees = 130, max_depth = 5, stopping_rounds = 3, stopping_metric = MAE), so this model was set as the best model.

By inserting the test dataset into this best model, I obtained the auc value(auc: 0.82316), the threshold value that maximize f1(threshold = 0.23895), and the confusion matrix.

Likewise, I drew a Shap plot, Shap summary, and permutation feature importance graph using this best model and test dataset. Through this, the most important features(City_te, Bank_FranchiseCode, Bank_te) and the direction of their influence were identified. In addition, eight sample graphs were drawn to illustrate how much each feature affected the prediction when the the predicted value of each row matched MIS_Status value or not.

Next, I analyzed the residuals (the absolute value of the difference between MIS_Status and the predicted value) resulting from this model. As a result, I drew a Shap plot and Shap Summary graph and identified the features(Bank_FranchiseCode, City_te, Bank_te) that most affect these errors.

Finally, the best_model was saved for the scoring function and the storage location was checked.

To further develop the results of this project, I would like to suggest two more things. First, the auc value in the train dataset was around 0.92, but in the valid and test datasets, both came out to only around 0.82. If this result is not a characteristic of the dataset but a result of the model overfitting the train dataset, additional work is required. Second, some numerical features of this dataset were positively skewed. But I didn't utilize minmax scaler or log scaler. If this process had been added, model performance would have been better.

In [ ]: