Customer Credit Application Prediction System

In [1]:
```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report, roc_curve,
from sklearn.impute import SimpleImputer
from imblearn import over_sampling
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import mutual_info_classif
from sklearn.tree import DecisionTreeClassifier, plot_tree
import kds
from imblearn.pipeline import Pipeline
```

Data Preparation

In [2]:
```
# Load in data
data = pd.read_csv('datafile_full.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var
0	125172.0	0	10.6216	0.1667	15.5182	10.1634	9.8419	-0.1156	7.4348	14.69
1	93976.0	0	13.1492	2.9210	6.0275	5.7623	12.5731	7.9090	5.3149	17.84
2	128558.0	0	7.9077	4.3397	8.5575	4.7743	10.3623	-6.6784	5.9665	17.23
3	138582.0	0	6.1757	-1.5021	16.5598	8.9091	8.0741	5.4070	5.3698	10.87
4	109242.0	0	10.6910	-6.5074	11.0558	4.4293	8.7645	-15.5975	5.6018	20.13

5 rows × 202 columns

In [4]:
```
# Overview of data
pd.set_option('display.max_columns', None)
print(data.info())
```

```
print(str('Number of rows: ') + str(print(data.shape[0])))
print(str('Number of rows: ') + str(print(data.shape[1])))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 202 entries, ID_code to var_199
dtypes: float64(201), int64(1)
memory usage: 154.1 MB
None
100000
Number of rows: None
202
Number of rows: None
```

In [5]:
```
# NA values
data.isna().any(axis=1).sum()
```

Out[5]:
```
np.int64(40072)
```

In [6]:
```
# check the variables for NA values
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
data.isna().sum().sort_values(ascending = False)
```

Out[6]:

var_45	40003
var_96	5004
var_76	5003
var_18	5
var_12	3
var_33	3
var_16	3
var_134	3
var_50	3
var_41	3
var_47	2
var_9	2
var_39	2
var_8	2
var_95	2
var_118	2
var_125	2
var_108	2
var_77	2
var_107	2
var_117	2
var_93	2
var_7	2
var_10	2
var_1	2
var_68	1
var_46	1
var_48	1
var_49	1
var_51	1
var_25	1
var_61	1
var_62	1
var_53	1
var_27	1
var_37	1
var_59	1
var_70	1
var_22	1
var_15	1
var_17	1
var_19	1
var_136	1
var_138	1
var_141	1
var_140	1
var_150	1
var_102	1
var_103	1
var_90	1
var_120	1
var_123	1
var_121	1
var_128	1
var_100	1
var_99	1
var_105	1
var_104	1
var_87	1
var_79	1

```
var_85         1
var_94         1
var_97         1
var_24         1
var_11         1
var_13         1
var_184        1
var_166        1
var_180        1
var_185        1
var_173        1
var_6          0
var_4          0
var_5          0
var_71         0
var_69         0
var_66         0
var_67         0
var_65         0
var_64         0
var_60         0
var_63         0
var_56         0
var_58         0
var_57         0
var_43         0
var_44         0
var_52         0
var_54         0
var_55         0
var_40         0
var_42         0
var_20         0
var_14         0
var_23         0
var_21         0
var_28         0
var_26         0
var_31         0
var_30         0
var_29         0
var_32         0
var_35         0
var_34         0
var_36         0
var_38         0
target         0
ID_code        0
var_2          0
var_0          0
var_3          0
var_89         0
var_91         0
var_92         0
var_101        0
var_98         0
var_88         0
var_81         0
var_111        0
var_110        0
```

```
var_106        0
var_109        0
var_119        0
var_116        0
var_114        0
var_115        0
var_113        0
var_112        0
var_75         0
var_72         0
var_78         0
var_73         0
var_80         0
var_83         0
var_86         0
var_84         0
var_82         0
var_74         0
var_135        0
var_133        0
var_132        0
var_131        0
var_129        0
var_130        0
var_126        0
var_127        0
var_124        0
var_122        0
var_146        0
var_145        0
var_143        0
var_144        0
var_142        0
var_139        0
var_151        0
var_152        0
var_154        0
var_153        0
var_156        0
var_157        0
var_158        0
var_155        0
var_159        0
var_160        0
var_162        0
var_161        0
var_147        0
var_148        0
var_149        0
var_137        0
var_167        0
var_165        0
var_164        0
var_163        0
var_171        0
var_172        0
var_170        0
var_168        0
var_175        0
var_176        0
```

```
var_178        0
var_177        0
var_179        0
var_181        0
var_174        0
var_169        0
var_183        0
var_182        0
var_186        0
var_187        0
var_188        0
var_189        0
var_190        0
var_191        0
var_192        0
var_193        0
var_194        0
var_195        0
var_196        0
var_197        0
var_198        0
var_199        0
dtype: int64
```

In [7]:
```
# summmary statistics
data.describe()
```

Out[7]:

	ID_code	target	var_0	var_1	var_2	
count	100000.000000	100000.000000	100000.000000	99998.000000	100000.000000	1000
mean	99916.419220	0.100380	10.681089	-1.645731	10.717789	
std	57689.007176	0.300508	3.042589	4.057037	2.642314	
min	2.000000	0.000000	0.408400	-14.091000	2.117100	
25%	49950.750000	0.000000	8.461025	-4.776975	8.723400	
50%	99884.500000	0.000000	10.533900	-1.623700	10.589800	
75%	149756.500000	0.000000	12.754825	1.355375	12.516925	
max	200000.000000	1.000000	19.737900	10.376800	19.353000	

In [8]:
```
# drop var_45 (has 40.000 NAs)
data = data.drop(columns = ['var_45', 'ID_code'])
print(str('Number of Remaining NAs: ') + str(data.isna().any(axis = 1).sum()))
```

```
Number of Remaining NAs: 5101
```

In [9]:
```
# total target varibale distribution
data['target'].value_counts()
```

Out[9]:
```
target
0    89962
1    10038
Name: count, dtype: int64
```

In [10]:
```
# check the relationship of remaining missing values with the target variable
remaining_na = data[data.isna().any(axis = 1)]

# proportion of the target value for remaining na values
print(str('Number of observations for target value:') + str(remaining_na['target
print()
print(str('Proportion of observations for target value:') + str(remaining_na['ta
```

```
Number of observations for target value:target
0    4543
1     558
Name: count, dtype: int64

Proportion of observations for target value:target
0    0.89061
1    0.10939
Name: proportion, dtype: float64
```

Removing the remaining rows with NA values would only result in removing roughly 5% of observations that belong to the target variable category of 1 (the minority class). For this reason, it is acceptable to remove all remaining rows with NA values.

In [11]:
```
# drop remaining NAs
data = data.dropna()
print(str('Number of Remaining NAs: ') + str(data.isna().any(axis = 1).sum()))
```

```
Number of Remaining NAs: 0
```

In [12]:
```
# number of columns and rows after removing NAs
print(str('Number of rows: ') + str(data.shape[0]))
print(str('Number of columns: ') + str(data.shape[1]))
```

```
Number of rows: 94899
Number of columns: 200
```

In [13]:
```
print('Proportion of classes of target variable')
data['target'].value_counts(normalize = True)
```

```
Proportion of classes of target variable
```

Out[13]:
```
target
0    0.900104
1    0.099896
Name: proportion, dtype: float64
```

Training and Test Sets

In [14]:
```
# partition variables
X = data.drop(columns = 'target')
y = data['target']

random_seed = 2

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, rando
```

Information Gain

In [15]:
```
# Use mutual_info_classif to compute information gain values of the attributes
IG_scores = mutual_info_classif(X_train, y_train, random_state=random_seed)

# Create a DataFrame with the results
weights = pd.DataFrame({
    'attr_importance': IG_scores
}, index=X_train.columns)

# Use the features with IG more than 0.004
weights = weights[weights['attr_importance'] > 0.004]

# Extract the names of those features
features = weights.index.tolist()

print(f"Number of features with positive information gain: {len(features)}")
print(f"\nFeatures: {features}")
print(f"\nInformation gain values:\n{weights.sort_values('attr_importance', asce
```

```
Number of features with positive information gain: 6

Features: ['var_12', 'var_22', 'var_81', 'var_110', 'var_146', 'var_164']

Information gain values:
         attr_importance
var_12          0.004983
var_81          0.004854
var_146         0.004386
var_22          0.004236
var_164         0.004189
var_110         0.004013
```

The information gain of each individual variable is extremely low. Therefore, they do not contain much information in predicting the target variable. We will train the models on all remaining variables.

Modeling

SVM

In [16]:
```
# define pipeline
pipeline = Pipeline(steps=[
    ('undersample', RandomUnderSampler(
        sampling_strategy=0.67,
        random_state=random_seed
    )),
    ('scaler', StandardScaler()),
    ('svm', SVC(
        kernel='rbf',
        random_state=random_seed,
        probability=True
    ))
])

# set parameters
param_grid = {
```

```
    'svm__C': [0.1, 1, 6]
}

# grid search
svm_grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=3,
    scoring='recall',
    n_jobs=-1
)

svm_grid_search.fit(X_train, y_train)

# results
print("Best C:", svm_grid_search.best_params_['svm__C'])
print("Best recall:", svm_grid_search.best_score_)

svm_best = svm_grid_search.best_estimator_
```

```
Best C: 6
Best recall: 0.7025316455696201
```

Decision Tree

In [17]:
```
# decision tree model
decision_tree = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=12,
    min_samples_leaf=50,
    min_samples_split=300,
    class_weight='balanced',
    random_state=random_seed
)

# train
decision_tree.fit(X_train, y_train)
```

Out[17]:

DecisionTreeClassifier

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=12, min_samples_leaf=50, min_samples_s
plit=300,
                       random_state=2)
```

In [18]:
```
# feature importance dt
feature = X_train.columns
f_importance_dt = decision_tree.feature_importances_

feature_importance_dt = pd.DataFrame({'Feature':feature, 'Importance':f_importan

feature_importance_dt
```

Out[18]:

	Feature	Importance
80	var_81	0.083294
138	var_139	0.081786
12	var_12	0.058914
109	var_110	0.046921
26	var_26	0.039333
22	var_22	0.033488
98	var_99	0.032903
173	var_174	0.030775
52	var_53	0.030365
6	var_6	0.023812
79	var_80	0.021267
108	var_109	0.020846
145	var_146	0.019426
77	var_78	0.018784
189	var_190	0.016429
0	var_0	0.015374
147	var_148	0.015296
165	var_166	0.014981
21	var_21	0.014768
1	var_1	0.014680
47	var_48	0.013757
168	var_169	0.013327
75	var_76	0.012955
153	var_154	0.012771
178	var_179	0.012749
40	var_40	0.012350
2	var_2	0.011847
197	var_198	0.011818
35	var_35	0.011523
46	var_47	0.008017
3	var_3	0.007733
31	var_31	0.007488
44	var_44	0.007318

	Feature	Importance
90	var_91	0.007313
34	var_34	0.007056
176	var_177	0.007031
164	var_165	0.006995
103	var_104	0.006714
132	var_133	0.006503
70	var_71	0.006415
121	var_122	0.006311
198	var_199	0.006143
148	var_149	0.006065
32	var_32	0.005928
163	var_164	0.005767
122	var_123	0.005551
76	var_77	0.005379
162	var_163	0.005314
166	var_167	0.005200
81	var_82	0.005068
190	var_191	0.004948
146	var_147	0.004831
169	var_170	0.004713
156	var_157	0.004652
28	var_28	0.004312
85	var_86	0.004199
71	var_72	0.004058
180	var_181	0.003993
66	var_67	0.003977
126	var_127	0.003817
191	var_192	0.003687
106	var_107	0.003644
39	var_39	0.003636
5	var_5	0.003611
133	var_134	0.003347
16	var_16	0.003163

	Feature	Importance
25	var_25	0.003101
127	var_128	0.002847
187	var_188	0.002816
8	var_8	0.002767
18	var_18	0.002753
93	var_94	0.002739
196	var_197	0.002703
124	var_125	0.002660
141	var_142	0.002637
123	var_124	0.002569
24	var_24	0.002505
42	var_42	0.002482
97	var_98	0.002427
152	var_153	0.002260
95	var_96	0.002054
36	var_36	0.002042
172	var_173	0.001993
89	var_90	0.001931
171	var_172	0.001896
158	var_159	0.001866
182	var_183	0.001829
94	var_95	0.001525
140	var_141	0.001158
69	var_70	0.000000
65	var_66	0.000000
63	var_64	0.000000
64	var_65	0.000000
61	var_62	0.000000
62	var_63	0.000000
58	var_59	0.000000
68	var_69	0.000000
67	var_68	0.000000
23	var_23	0.000000

	Feature	Importance
30	var_30	0.000000
33	var_33	0.000000
37	var_37	0.000000
38	var_38	0.000000
9	var_9	0.000000
13	var_13	0.000000
11	var_11	0.000000
10	var_10	0.000000
17	var_17	0.000000
15	var_15	0.000000
20	var_20	0.000000
19	var_19	0.000000
4	var_4	0.000000
57	var_58	0.000000
60	var_61	0.000000
59	var_60	0.000000
56	var_57	0.000000
55	var_56	0.000000
53	var_54	0.000000
54	var_55	0.000000
41	var_41	0.000000
49	var_50	0.000000
51	var_52	0.000000
50	var_51	0.000000
48	var_49	0.000000
45	var_46	0.000000
43	var_43	0.000000
14	var_14	0.000000
7	var_7	0.000000
29	var_29	0.000000
27	var_27	0.000000
82	var_83	0.000000
86	var_87	0.000000

	Feature	Importance
84	var_85	0.000000
83	var_84	0.000000
78	var_79	0.000000
134	var_135	0.000000
131	var_132	0.000000
129	var_130	0.000000
130	var_131	0.000000
128	var_129	0.000000
120	var_121	0.000000
119	var_120	0.000000
125	var_126	0.000000
111	var_112	0.000000
112	var_113	0.000000
113	var_114	0.000000
114	var_115	0.000000
115	var_116	0.000000
116	var_117	0.000000
117	var_118	0.000000
118	var_119	0.000000
110	var_111	0.000000
107	var_108	0.000000
104	var_105	0.000000
105	var_106	0.000000
74	var_75	0.000000
73	var_74	0.000000
72	var_73	0.000000
87	var_88	0.000000
91	var_92	0.000000
92	var_93	0.000000
88	var_89	0.000000
96	var_97	0.000000
99	var_100	0.000000
100	var_101	0.000000

	Feature	Importance
101	var_102	0.000000
102	var_103	0.000000
160	var_161	0.000000
159	var_160	0.000000
161	var_162	0.000000
157	var_158	0.000000
154	var_155	0.000000
151	var_152	0.000000
155	var_156	0.000000
135	var_136	0.000000
142	var_143	0.000000
139	var_140	0.000000
136	var_137	0.000000
137	var_138	0.000000
143	var_144	0.000000
144	var_145	0.000000
149	var_150	0.000000
150	var_151	0.000000
181	var_182	0.000000
179	var_180	0.000000
175	var_176	0.000000
177	var_178	0.000000
170	var_171	0.000000
167	var_168	0.000000
174	var_175	0.000000
183	var_184	0.000000
188	var_189	0.000000
184	var_185	0.000000
185	var_186	0.000000
186	var_187	0.000000
194	var_195	0.000000
193	var_194	0.000000
192	var_193	0.000000

	Feature	Importance
195	var_196	0.000000

In [19]:
```
# feature selection for decision tree
feature_selection = feature_importance_dt[feature_importance_dt['Importance'] >
X_train_selected = X_train[feature_selection]
X_test_selected = X_test[feature_selection]
```

In [20]:
```
# decision tree model with select features
decision_tree_selected = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=12,
    min_samples_leaf=50,
    min_samples_split=300,
    class_weight='balanced',
    random_state=random_seed
)

# train
decision_tree_selected.fit(X_train_selected, y_train)
```

Out[20]:
```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=12, min_samples_leaf=50, min_samples_s
plit=300,
                       random_state=2)
```

Logistic Regression

In [21]:
```
# define pipeline
pipeline = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('logreg', LogisticRegression(
        penalty="l2",
        solver="liblinear",
        class_weight="balanced",
        random_state=random_seed,
        max_iter=1000
    ))
])

# sets parameters
log_param_grid = {
    'logreg__C': [0.01, 0.1, 1, 10]
}

# grid search
log_grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=log_param_grid,
    cv=3,
    scoring="recall",
    n_jobs=-1
```

```
)

log_grid_search.fit(X_train, y_train)

# results
print("Best C:", log_grid_search.best_params_['logreg__C'])
print("Best CV Recall:", log_grid_search.best_score_)

logreg_best = log_grid_search.best_estimator_
```

```
Best C: 0.01
Best CV Recall: 0.7689873417721519
```

Random Forest

In [22]:
```
rf_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_leaf': [1, 5],
    'max_features': ['sqrt']
}

rf = RandomForestClassifier(class_weight='balanced', random_state=random_seed)

rf_grid = GridSearchCV(
    estimator=rf,
    param_grid=rf_param_grid,
    scoring='recall',
    cv=3,
    n_jobs=-1
)

rf_grid.fit(X_train, y_train)

print("Best parameters:", rf_grid.best_params_)
print("Best CV recall:", rf_grid.best_score_)

rf_best = rf_grid.best_estimator_
```

```
Best parameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 5,
'n_estimators': 100}
Best CV recall: 0.09433393610608799
```

In [23]:
```
# feature importance rf
feature = X_train.columns
f_importance_rf = rf_best.feature_importances_

feature_importance_rf = pd.DataFrame({'Feature':feature, 'Importance':f_importan

feature_importance_rf
```

Out[23]:

	Feature	Importance
80	var_81	0.049323
138	var_139	0.037102
12	var_12	0.027238
109	var_110	0.025602
26	var_26	0.022673
52	var_53	0.020696
22	var_22	0.019217
6	var_6	0.018396
145	var_146	0.016912
165	var_166	0.015644
98	var_99	0.015356
79	var_80	0.014528
75	var_76	0.014309
77	var_78	0.013063
108	var_109	0.013028
173	var_174	0.012854
21	var_21	0.012183
189	var_190	0.011837
13	var_13	0.011738
2	var_2	0.011531
147	var_148	0.011447
178	var_179	0.011357
132	var_133	0.010906
197	var_198	0.010567
40	var_40	0.009614
0	var_0	0.008865
164	var_165	0.008676
1	var_1	0.008292
163	var_164	0.007717
153	var_154	0.007701
114	var_115	0.007586
44	var_44	0.006728
191	var_192	0.006715

	Feature	Importance
120	var_121	0.006680
176	var_177	0.006633
190	var_191	0.006288
169	var_170	0.006158
90	var_91	0.006130
148	var_149	0.006097
66	var_67	0.006024
34	var_34	0.005808
91	var_92	0.005768
126	var_127	0.005528
33	var_33	0.005429
88	var_89	0.005157
168	var_169	0.005079
121	var_122	0.005025
106	var_107	0.005000
93	var_94	0.004938
9	var_9	0.004784
171	var_172	0.004748
55	var_56	0.004456
161	var_162	0.004450
18	var_18	0.004422
35	var_35	0.004372
74	var_75	0.004330
85	var_86	0.004301
183	var_184	0.004297
187	var_188	0.004206
70	var_71	0.004193
107	var_108	0.004158
122	var_123	0.004114
36	var_36	0.004061
146	var_147	0.003991
118	var_119	0.003859
89	var_90	0.003831

	Feature	Importance
154	var_155	0.003812
130	var_131	0.003650
94	var_95	0.003627
111	var_112	0.003619
196	var_197	0.003616
127	var_128	0.003542
166	var_167	0.003509
144	var_145	0.003480
86	var_87	0.003451
47	var_48	0.003442
92	var_93	0.003361
32	var_32	0.003342
117	var_118	0.003259
172	var_173	0.003159
156	var_157	0.003141
57	var_58	0.003104
129	var_130	0.003077
134	var_135	0.003042
65	var_66	0.003039
5	var_5	0.003038
179	var_180	0.003038
28	var_28	0.003037
24	var_24	0.003007
162	var_163	0.002994
51	var_52	0.002988
23	var_23	0.002953
105	var_106	0.002920
185	var_186	0.002920
110	var_111	0.002920
43	var_43	0.002849
113	var_114	0.002849
103	var_104	0.002848
69	var_70	0.002837

	Feature	Importance
81	var_82	0.002791
133	var_134	0.002787
50	var_51	0.002765
149	var_150	0.002759
31	var_31	0.002752
140	var_141	0.002741
194	var_195	0.002684
11	var_11	0.002683
124	var_125	0.002681
198	var_199	0.002669
48	var_49	0.002660
193	var_194	0.002649
137	var_138	0.002629
3	var_3	0.002595
84	var_85	0.002567
192	var_193	0.002558
76	var_77	0.002557
158	var_159	0.002554
20	var_20	0.002553
64	var_65	0.002547
135	var_136	0.002545
136	var_137	0.002544
160	var_161	0.002544
61	var_62	0.002542
186	var_187	0.002532
16	var_16	0.002524
141	var_142	0.002516
96	var_97	0.002505
59	var_60	0.002498
170	var_171	0.002474
112	var_113	0.002469
27	var_27	0.002464
131	var_132	0.002464

	Feature	Importance
56	var_57	0.002448
42	var_42	0.002438
104	var_105	0.002432
139	var_140	0.002428
8	var_8	0.002424
73	var_74	0.002421
119	var_120	0.002415
54	var_55	0.002407
25	var_25	0.002406
62	var_63	0.002369
53	var_54	0.002369
150	var_151	0.002357
87	var_88	0.002351
71	var_72	0.002351
175	var_176	0.002349
72	var_73	0.002321
67	var_68	0.002313
143	var_144	0.002307
128	var_129	0.002294
95	var_96	0.002288
155	var_156	0.002286
83	var_84	0.002279
49	var_50	0.002273
151	var_152	0.002273
182	var_183	0.002250
159	var_160	0.002239
30	var_30	0.002236
4	var_4	0.002216
78	var_79	0.002208
37	var_37	0.002180
60	var_61	0.002178
39	var_39	0.002176
17	var_17	0.002169

	Feature	Importance
97	var_98	0.002164
46	var_47	0.002164
125	var_126	0.002164
101	var_102	0.002161
188	var_189	0.002161
177	var_178	0.002160
123	var_124	0.002155
142	var_143	0.002152
152	var_153	0.002150
195	var_196	0.002145
63	var_64	0.002138
19	var_19	0.002133
29	var_29	0.002133
82	var_83	0.002127
102	var_103	0.002097
38	var_38	0.002096
115	var_116	0.002089
41	var_41	0.002055
7	var_7	0.002042
45	var_46	0.002032
167	var_168	0.002025
14	var_14	0.001994
174	var_175	0.001945
68	var_69	0.001923
15	var_15	0.001914
10	var_10	0.001890
157	var_158	0.001876
116	var_117	0.001870
181	var_182	0.001828
184	var_185	0.001825
99	var_100	0.001818
100	var_101	0.001813
58	var_59	0.001746

	Feature	Importance
180	var_181	0.001573

In [24]:
```
# feature selection for rf
features_rf = feature_importance_rf[feature_importance_rf['Importance'] > 0.005]
X_train_selected_rf = X_train[features_rf]
X_test_selected_rf = X_test[features_rf]
```

In [25]:
```
# rf with select features
rf_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_leaf': [1, 5],
    'max_features': ['sqrt']
}

rf = RandomForestClassifier(class_weight='balanced', random_state=random_seed)

rf_grid_selected = GridSearchCV(
    estimator=rf,
    param_grid=rf_param_grid,
    scoring='recall',
    cv=3,
    n_jobs=-1
)

rf_grid_selected.fit(X_train_selected, y_train)

print("Best parameters:", rf_grid_selected.best_params_)
print("Best CV recall:", rf_grid_selected.best_score_)

rf_best_selected = rf_grid_selected.best_estimator_
```

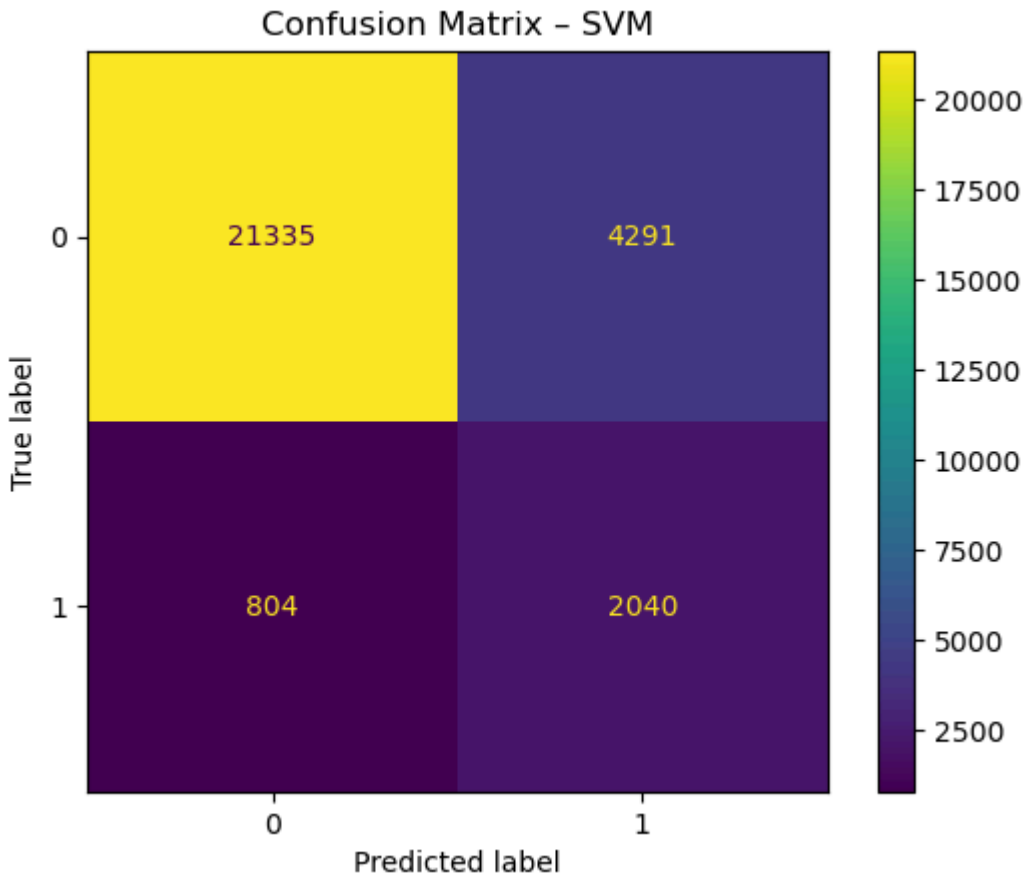
```
Best parameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 5,
'n_estimators': 100}
Best CV recall: 0.22739602169981918
```

Evaluation

SVM Evaluation

In [29]:
```
# SVM predictions
svm_pred = svm_best.predict(X_test)
svm_prob = svm_best.predict_proba(X_test)[:,1]

# confusion matrix
svm_cm = confusion_matrix(y_test, svm_pred, labels = svm_best.classes_)
svm_disp = ConfusionMatrixDisplay(confusion_matrix = svm_cm, display_labels = sv
svm_disp.plot(values_format = None)
plt.title("Confusion Matrix - SVM")
plt.show()

# prediction scores
print("\nSVM Performance:")
print('Accuracy Score: ', accuracy_score(y_test, svm_pred))
```

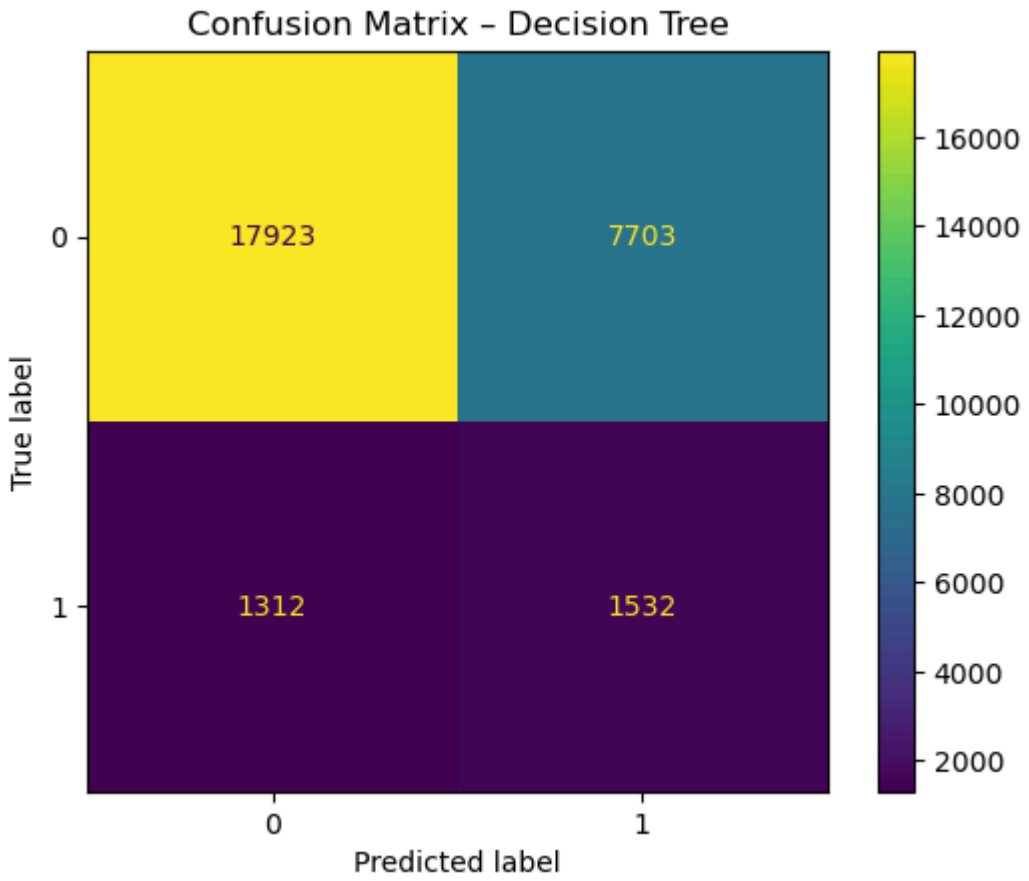
```
print('Recall Score: ', recall_score(y_test, svm_pred))
print('Precision Score: ', precision_score(y_test, svm_pred))
print('F1 Score: ', f1_score(y_test, svm_pred))
```




```
SVM Performance:
Accuracy Score:  0.8210396909027046
Recall Score:  0.7172995780590717
Precision Score:  0.3222239772547781
F1 Score:  0.4446866485013624
```

Decision Tree Evaluation

In [31]:
```
# decision tree predictions
dt_pred = decision_tree.predict(X_test)
dt_prob = decision_tree.predict_proba(X_test)[:,1]

# Confusion Matrix
dt_cm = confusion_matrix(y_test, dt_pred, labels = decision_tree.classes_)
dt_disp_best = ConfusionMatrixDisplay(confusion_matrix=dt_cm, display_labels = d
dt_disp_best.plot(values_format=None)
plt.title("Confusion Matrix – Decision Tree")
plt.show()

# Metrics
print("\nDecision Tree Model Performance:")
print("Accuracy:", accuracy_score(y_test, dt_pred))
print("Recall:", recall_score(y_test, dt_pred))
print("Precision:", precision_score(y_test, dt_pred))
print("F1 Score:", f1_score(y_test, dt_pred))
```

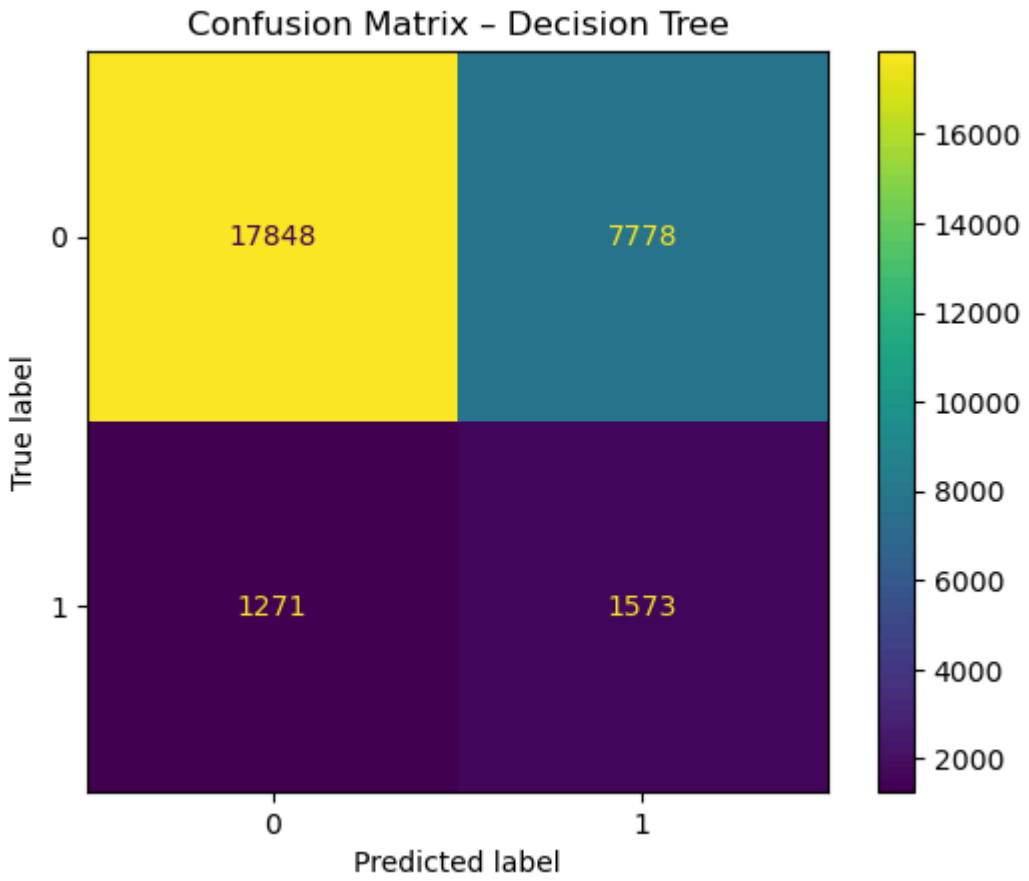



```
Decision Tree Model Performance:
Accuracy: 0.6833508956796628
Recall: 0.5386779184247539
Precision: 0.16589063345966432
F1 Score: 0.2536633827303585
```

Decision Tree with Selected Features Evaluation

In [32]:

```
# decision tree predictions
dt_selected_pred = decision_tree_selected.predict(X_test_selected)
dt_selected_prob = decision_tree_selected.predict_proba(X_test_selected)[:,1]

# Confusion Matrix
dt_cm = confusion_matrix(y_test, dt_selected_pred, labels = decision_tree_select
dt_disp_best = ConfusionMatrixDisplay(confusion_matrix=dt_cm, display_labels = d
dt_disp_best.plot(values_format=None)
plt.title("Confusion Matrix – Decision Tree")
plt.show()

# Metrics
print("\nDecision Tree Model Performance:")
print("Accuracy:", accuracy_score(y_test, dt_selected_pred))
print("Recall:", recall_score(y_test, dt_selected_pred))
print("Precision:", precision_score(y_test, dt_selected_pred))
print("F1 Score:", f1_score(y_test, dt_selected_pred))
```

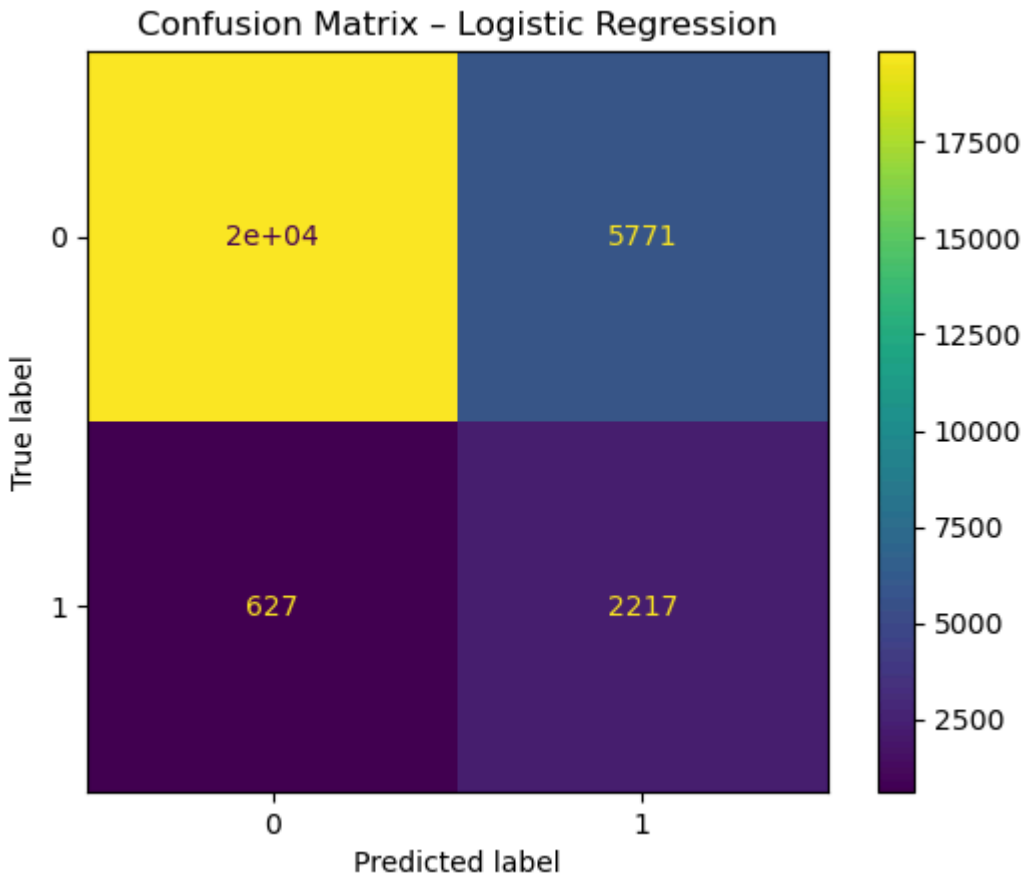



```
Decision Tree Model Performance:
Accuracy: 0.6821566561292589
Recall: 0.5530942334739803
Precision: 0.16821730296225001
F1 Score: 0.25797457974579746
```

Logistic Regression Evaluation

In [33]:

```python
# log predictions
log_pred = logreg_best.predict(X_test)
log_prob = logreg_best.predict_proba(X_test)[:,1]

# Confusion Matrix
log_cm_best = confusion_matrix(y_test, log_pred, labels = logreg_best.classes_)
log_disp_best = ConfusionMatrixDisplay(confusion_matrix=log_cm_best, display_lab
log_disp_best.plot(values_format=None)
plt.title("Confusion Matrix – Logistic Regression")
plt.show()

# Metrics
print("\nBest Logistic Regression Model Performance:")
print("Accuracy:", accuracy_score(y_test, log_pred))
print("Recall:", recall_score(y_test, log_pred))
print("Precision:", precision_score(y_test, log_pred))
print("F1 Score:", f1_score(y_test, log_pred))
```

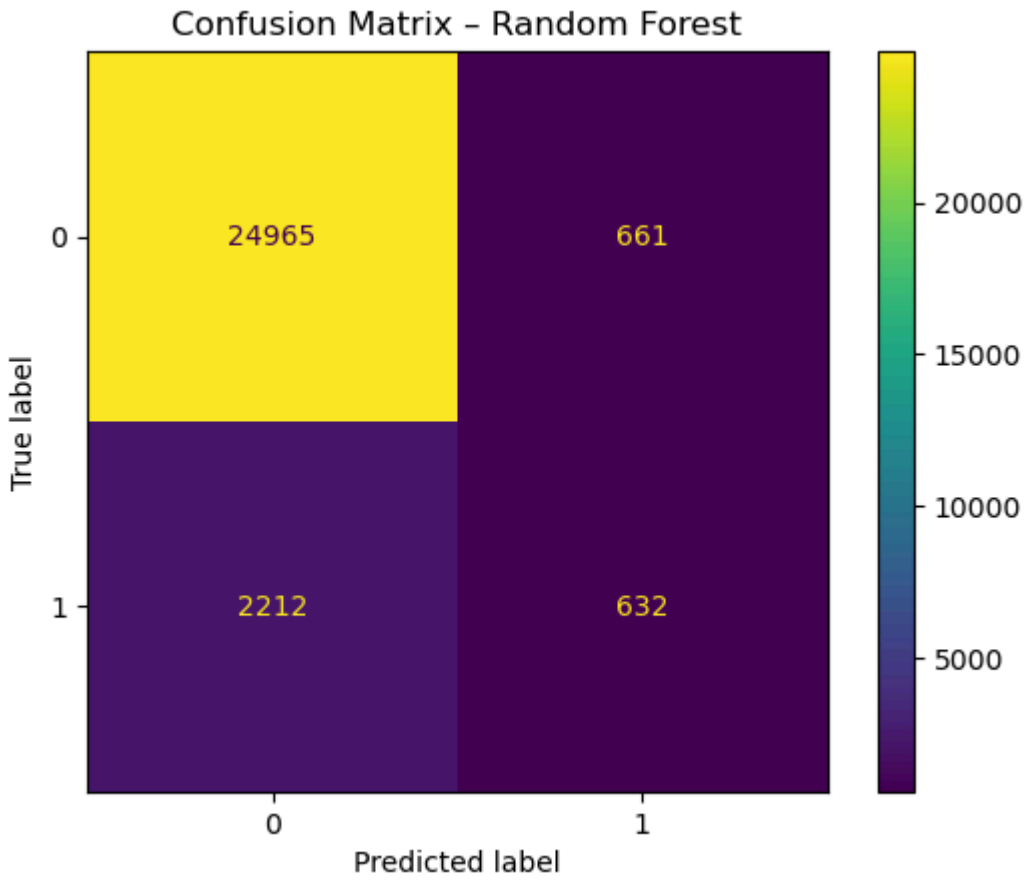



```
Best Logistic Regression Model Performance:
Accuracy: 0.7752722163681067
Recall: 0.7795358649789029
Precision: 0.27754131196795195
F1 Score: 0.4093426883308715
```

Random Forest Evaluation

In [34]:

```python
# Random Forest Predictions
rf_pred = rf_best.predict(X_test)
rf_prob = rf_best.predict_proba(X_test)[:,1]

# Confusion Matrix
rf_cm = confusion_matrix(y_test, rf_pred, labels=rf_best.classes_)
rf_disp = ConfusionMatrixDisplay(confusion_matrix=rf_cm, display_labels=rf_best.
rf_disp.plot(values_format=None)
plt.title("Confusion Matrix – Random Forest")
plt.show()

# Evaluation Metrics
print("Random Forest Performance:")
print("Accuracy:  ", accuracy_score(y_test, rf_pred))
print("Recall:    ", recall_score(y_test, rf_pred))
print("Precision: ", precision_score(y_test, rf_pred))
print("F1 Score:  ", f1_score(y_test, rf_pred))
```

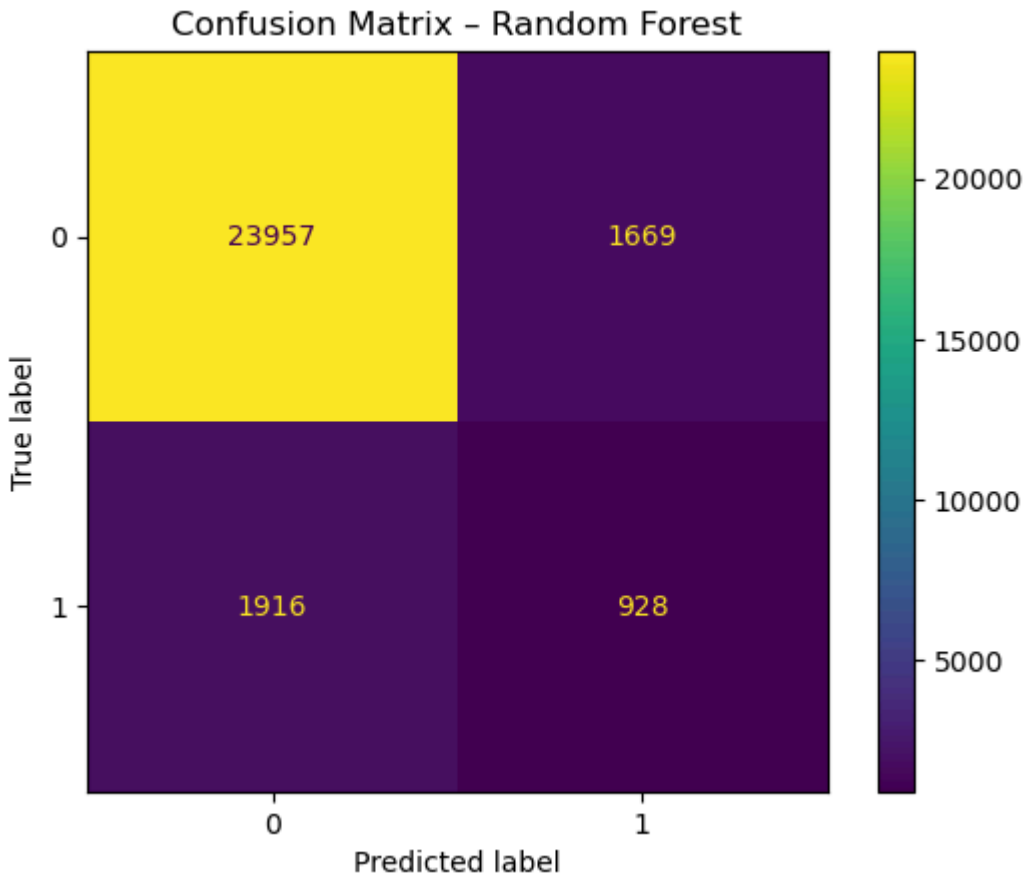



```
Random Forest Performance:
Accuracy:   0.8990867579908676
Recall:     0.2222222222222222
Precision:  0.48878576952822894
F1 Score:   0.3055354121343969
```

Random Forest with Select Features Evaluation

In [35]:

```python
# Random Forest Predictions
rf_selected_pred = rf_best_selected.predict(X_test_selected)
rf_selected_prob = rf_best_selected.predict_proba(X_test_selected)[:,1]

# Confusion Matrix
rf_cm = confusion_matrix(y_test, rf_selected_pred, labels=rf_best_selected.class
rf_disp = ConfusionMatrixDisplay(confusion_matrix=rf_cm, display_labels=rf_best_
rf_disp.plot(values_format=None)
plt.title("Confusion Matrix - Random Forest")
plt.show()

# Evaluation Metrics
print("Random Forest Performance:")
print("Accuracy:  ", accuracy_score(y_test, rf_selected_pred))
print("Recall:    ", recall_score(y_test, rf_selected_pred))
print("Precision: ", precision_score(y_test, rf_selected_pred))
print("F1 Score:  ", f1_score(y_test, rf_selected_pred))
```

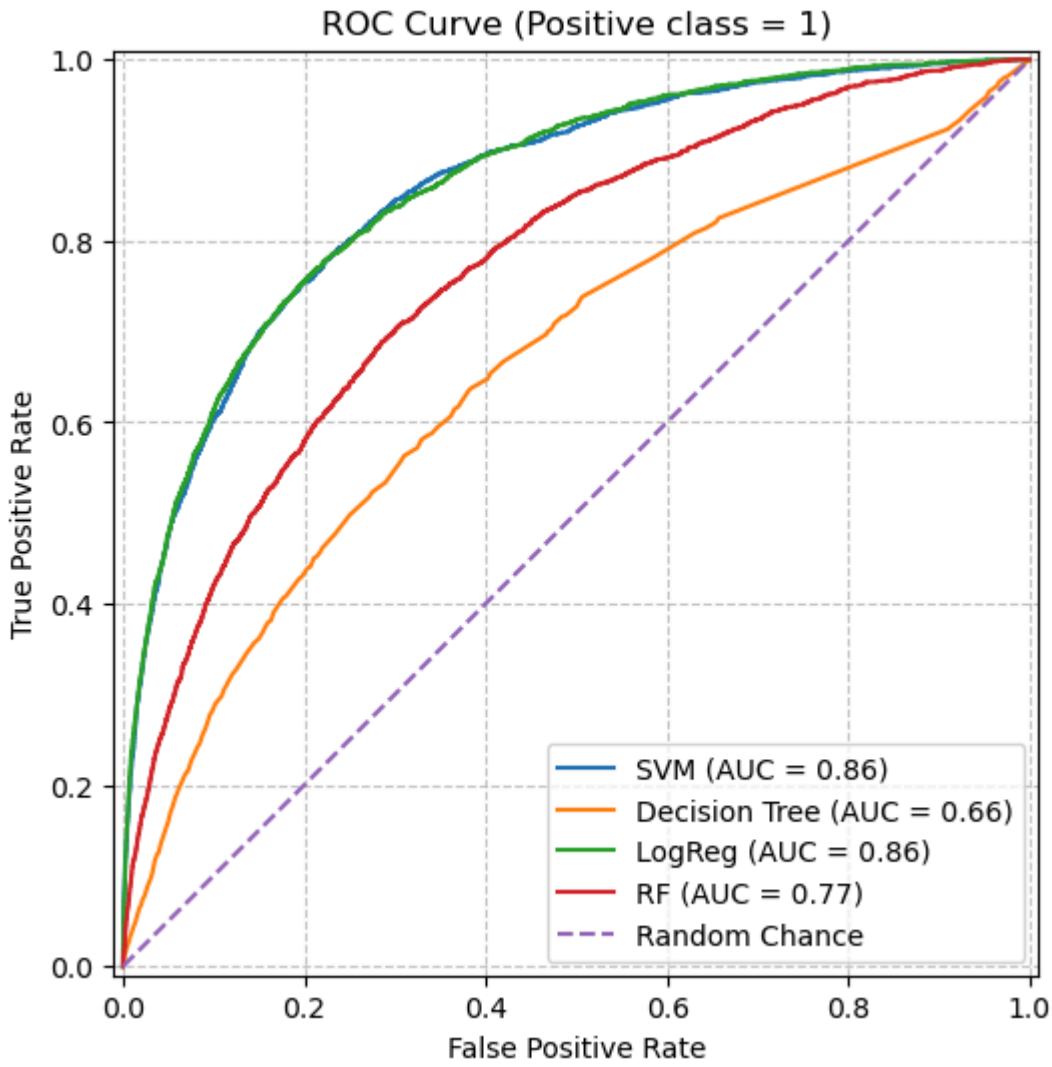



```
Random Forest Performance:
Accuracy:   0.8740779768177028
Recall:     0.3263009845288326
Precision:  0.3573353869849827
F1 Score:   0.3411137658518655
```

Comparison of all Models

ROC Curve

In [36]:
```python
# ROC Curve for all models
fig, ax = plt.subplots(figsize=(6, 6))

RocCurveDisplay.from_predictions(y_test, svm_prob, name="SVM", ax=ax)
RocCurveDisplay.from_predictions(y_test, dt_selected_prob, name="Decision Tree",
RocCurveDisplay.from_predictions(y_test, log_prob, name="LogReg", ax=ax)
RocCurveDisplay.from_predictions(y_test, rf_selected_prob, name="RF", ax=ax)

ax.plot([0, 1], [0, 1], "--", label="Random Chance")
ax.set_xlabel("False Positive Rate")
ax.set_ylabel("True Positive Rate")
ax.set_title("ROC Curve (Positive class = 1)")
ax.legend()
ax.grid(True, linestyle="--", alpha=0.7)

plt.show()
```

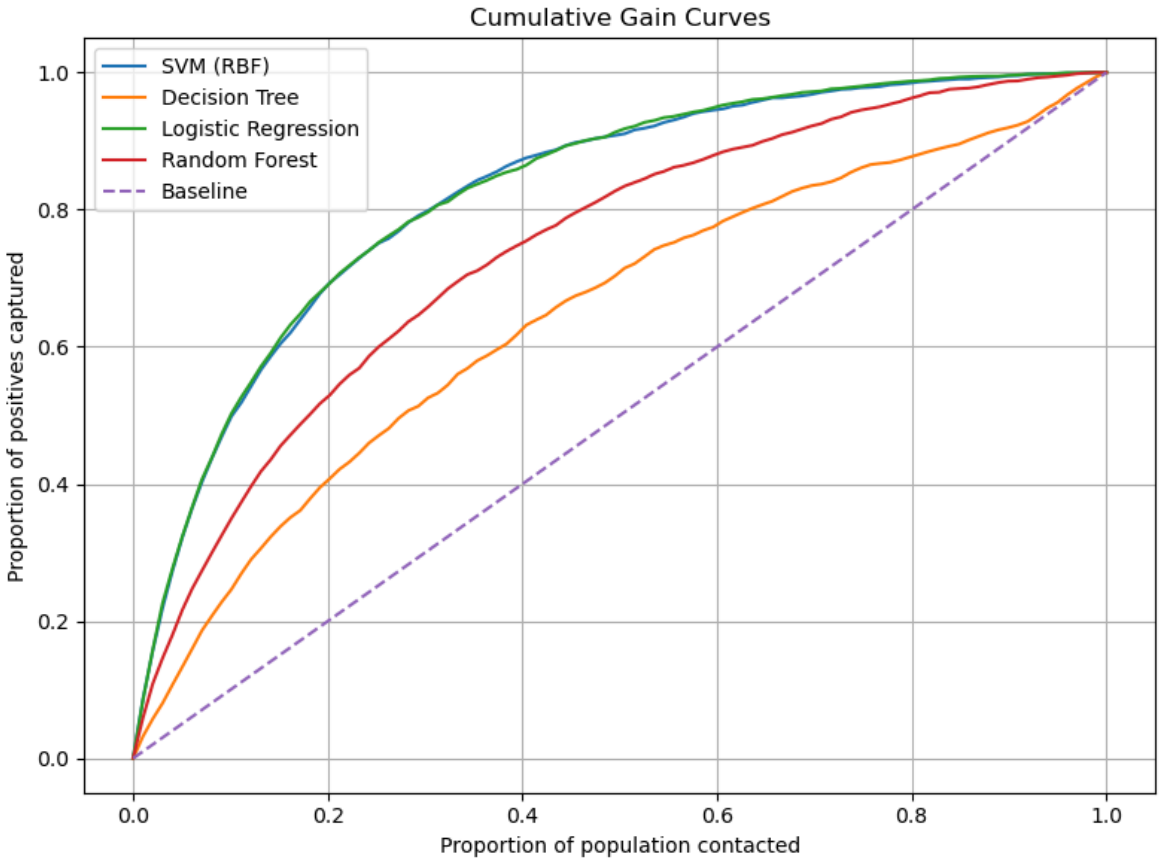


Cumulative Gain Chart

In [37]:

```python
# Define a function to compute the cumulative gain curve
def cumulative_gains(y_true, y_prob, resolution=100):
    # Sort predicted probabilities descending
    order = np.argsort(y_prob)[::-1]

    # True labels sorted according to predicted rank
    y_sorted = np.array(y_true)[order]

    # Cumulative true positives
    cum_positives = np.cumsum(y_sorted)

    # Total positives
    total_positives = y_sorted.sum()

    # Evenly spaced proportions of population contacted
    perc_population = np.linspace(0, 1, resolution)

    # Convert % population into cutoffs
    cutoff = (perc_population * len(y_true)).astype(int)

    # Compute gain at each cutoff
    gains = cum_positives[np.clip(cutoff - 1, 0, None)] / total_positives

    return perc_population, gains
```

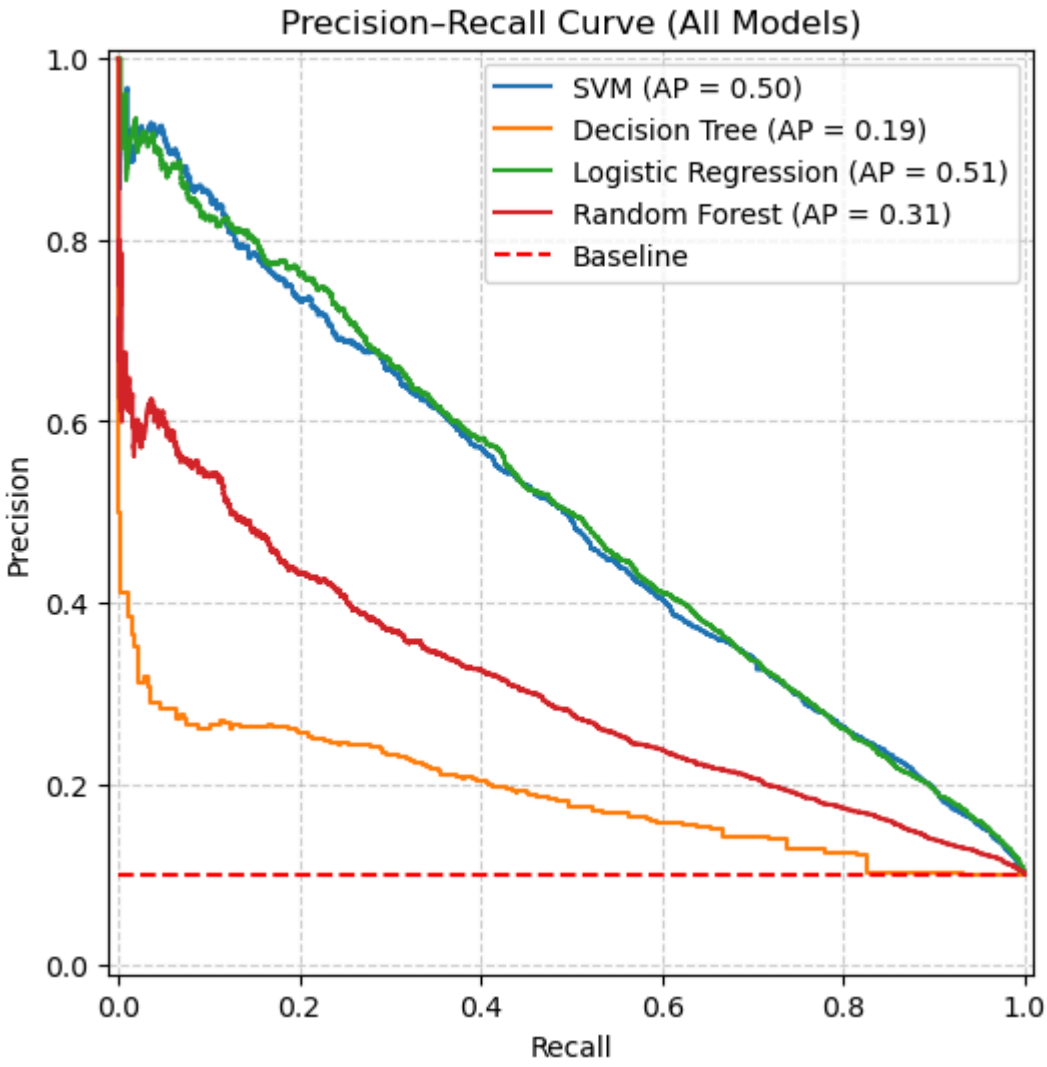
```
# Compute curves
x_svm, g_svm = cumulative_gains(y_test, svm_prob)
x_dt,  g_dt  = cumulative_gains(y_test, dt_selected_prob)
x_log, g_log = cumulative_gains(y_test, log_prob)
x_rf,  g_rf  = cumulative_gains(y_test, rf_selected_prob)


# Plot
plt.figure(figsize=(8, 6))
plt.plot(x_svm, g_svm, label="SVM (RBF)")
plt.plot(x_dt,  g_dt,  label="Decision Tree")
plt.plot(x_log, g_log, label="Logistic Regression")
plt.plot(x_rf,  g_rf,  label="Random Forest")
plt.plot([0, 1], [0, 1], '--', label="Baseline")

plt.xlabel("Proportion of population contacted")
plt.ylabel("Proportion of positives captured")
plt.title("Cumulative Gain Curves")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Precision-Recall Curve

In [38]:
```
fig, ax = plt.subplots(figsize=(7, 6))

# SVM
PrecisionRecallDisplay.from_predictions(
    y_test, svm_prob, name="SVM", ax=ax
)
```

```python
# Decision Tree
PrecisionRecallDisplay.from_predictions(
    y_test, dt_selected_prob, name="Decision Tree", ax=ax
)

# Logistic Regression
PrecisionRecallDisplay.from_predictions(
    y_test, log_prob, name="Logistic Regression", ax=ax
)

# Random Forest
PrecisionRecallDisplay.from_predictions(
    y_test, rf_selected_prob, name="Random Forest", ax=ax
)

# Baseline = positive class prevalence
baseline = y_test.mean()
ax.hlines(baseline, xmin=0, xmax=1, colors='red', linestyles='--', label='Baseli

ax.set_title("Precision-Recall Curve (All Models)")
ax.set_xlabel("Recall")
ax.set_ylabel("Precision")
ax.legend()
ax.grid(True, linestyle="--", alpha=0.6)

plt.show()
```



Model Comparison Summary

In [39]:
```
# Accuracy
accuracy_svm = accuracy_score(y_test, svm_pred)
accuracy_log = accuracy_score(y_test, log_pred)
accuracy_dt = accuracy_score(y_test, dt_selected_pred)
accuracy_rf = accuracy_score(y_test, rf_selected_pred)

# Precision
precision_svm = precision_score(y_test, svm_pred)
precision_log = precision_score(y_test, log_pred)
precision_dt = precision_score(y_test, dt_selected_pred)
precision_rf = precision_score(y_test, rf_selected_pred)

# Recall
recall_svm = recall_score(y_test, svm_pred)
recall_log = recall_score(y_test, log_pred)
recall_dt = recall_score(y_test, dt_selected_pred)
recall_rf = recall_score(y_test, rf_selected_pred)

# F1-Score
f1_svm = f1_score(y_test, svm_pred)
f1_log = f1_score(y_test, log_pred)
f1_dt = f1_score(y_test, dt_selected_pred)
f1_rf = f1_score(y_test, rf_selected_pred)

# AUC Score
auc_svm = roc_auc_score(y_test, svm_prob)
auc_log = roc_auc_score(y_test, log_prob)
auc_dt = roc_auc_score(y_test, dt_selected_prob)
auc_rf = roc_auc_score(y_test, rf_selected_prob)

comparison_df = pd.DataFrame({
    'Model': ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest'],

    'Accuracy':  [accuracy_log, accuracy_svm, accuracy_dt, accuracy_rf],
    'Precision': [precision_log, precision_svm, precision_dt, precision_rf],
    'Recall':    [recall_log, recall_svm, recall_dt, recall_rf],
    'F1-Score':  [f1_log, f1_svm, f1_dt, f1_rf],
    'AUC':       [auc_log, auc_svm, auc_dt, auc_rf]
})

print("="*80)
print("MODEL PERFORMANCE COMPARISON")
print("="*80)
print(comparison_df.to_string(index=False))
print("="*80)

# Identify the best model per metric
print("\nBest Models by Metric:")
print(f"Highest Accuracy:  {comparison_df.loc[comparison_df['Accuracy'].idxmax()
      f"({comparison_df['Accuracy'].max():.4f})")

print(f"Highest Precision: {comparison_df.loc[comparison_df['Precision'].idxmax(
      f"({comparison_df['Precision'].max():.4f})")

print(f"Highest Recall:    {comparison_df.loc[comparison_df['Recall'].idxmax(),
      f"({comparison_df['Recall'].max():.4f})")
```

```
print(f"Highest F1-Score:  {comparison_df.loc[comparison_df['F1-Score'].idxmax()
      f"({comparison_df['F1-Score'].max():.4f})")

print(f"Highest AUC:       {comparison_df.loc[comparison_df['AUC'].idxmax(), 'Mo
      f"({comparison_df['AUC'].max():.4f})")

print("="*80)
```

```
================================================================================
MODEL PERFORMANCE COMPARISON
================================================================================
              Model  Accuracy  Precision   Recall  F1-Score      AUC
Logistic Regression  0.775272   0.277541 0.779536  0.409343 0.860398
                SVM  0.821040   0.322224 0.717300  0.444687 0.858406
      Decision Tree  0.682157   0.168217 0.553094  0.257975 0.659945
      Random Forest  0.874078   0.357335 0.326301  0.341114 0.768369
================================================================================

Best Models by Metric:
Highest Accuracy:  Random Forest (0.8741)
Highest Precision: Random Forest (0.3573)
Highest Recall:    Logistic Regression (0.7795)
Highest F1-Score:  SVM (0.4447)
Highest AUC:       Logistic Regression (0.8604)
================================================================================
```

The Logistic Regression model achieved the highest recall score (~78%), which is the most important metric for our scenario because accurately identifying customers likely to apply for a business credit is the primary goal. To balance capturing these customers while avoiding unnecessary allocation of resources to those who will not apply, we will tune the model's probability threshold to maximize the F1-score. Using the F1-score ensures that we account for both recall (correctly identifying potential applicants) and precision (avoiding false positives), aligning the model's performance with the business objectives.

The consistently low precision across all models indicates that a large proportion of positive predictions are false positives.

Threshold Tuning for the Best Model

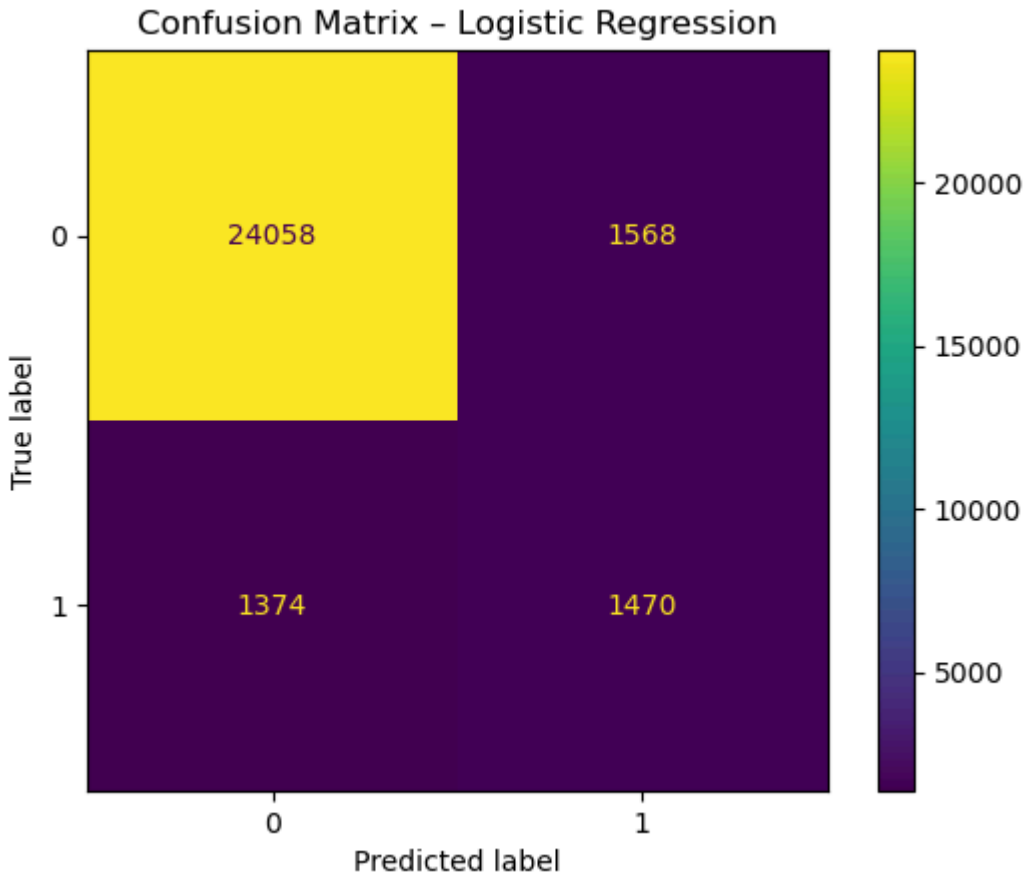
In [40]:
```
log_prob = logreg_best.predict_proba(X_test)[:,1]
y_true = y_test     # must match X_test_scaled

thresholds = []
recalls = []
precisions = []
f1_scores = []

for t in [i/100 for i in range(1, 100)]:  # thresholds 0.01 → 0.99
    y_pred = (log_prob >= t).astype(int)

    thresholds.append(t)
    recalls.append(recall_score(y_true, y_pred))
    precisions.append(precision_score(y_true, y_pred))
    f1_scores.append(f1_score(y_true, y_pred))
```

In [41]:
```
# get best f1-score
best_threshold = thresholds[f1_scores.index(max(f1_scores))]
print("Best threshold for F1:", best_threshold)
```

```
Best threshold for F1: 0.76
```

In [42]:
```
y_pred_test = (log_prob >= best_threshold).astype(int)
```

In [43]:
```
# confusion matrix
final_logreg_cm = confusion_matrix(y_test, y_pred_test, labels = logreg_best.cla
final_logreg_disp = ConfusionMatrixDisplay(confusion_matrix = final_logreg_cm, d
final_logreg_disp.plot(values_format = None)
plt.title("Confusion Matrix – Logistic Regression")
plt.show()

# prediction scores
print("\nLogistic Regression Performance:")
print('Accuracy Score: ', accuracy_score(y_test, y_pred_test))
print('Recall Score: ', recall_score(y_test, y_pred_test))
print('Precision Score: ', precision_score(y_test, y_pred_test))
print('F1 Score: ', f1_score(y_test, y_pred_test))
```




```
Logistic Regression Performance:
Accuracy Score:  0.8966631541974007
Recall Score:  0.5168776371308017
Precision Score:  0.4838709677419355
F1 Score:  0.499829989799388
```

Tuning the probability threshold to maximize the F1-score of the Logistic Regression model will reduce the number of true positives the model is able to identify, but greatly decrease the number of false positive predictions made.

In [44]:
```
# store the final model
final_model = logreg_best
final_threshold = best_threshold
```

It is important to note that due to computational limitations, an undersampled version of the training data was used to train the SVM model. The SVM may potentially perform better if trained on the entire training data (or a balanced class data), such as the other models, but this was not possible due to the time it would take to train the model on the available computers.