# finalclassiclamlpdf

April 25, 2025

[1]:
```python
import pandas as pd
```

[2]:
```python
df=pd.read_csv(r"D:\dataset\new_data.csv")
```

[3]:
```python
df
```

[3]:
```
         Destination Port  Flow Duration  Total Length of Fwd Packets  \
0                   54865              3                           12
1                   55054            109                            6
2                   55055             52                            6
3                   46236             34                            6
4                   54863              3                           12
…                     …              …                            …
2520793                53          32215                          112
2520794                53            324                           84
2520795             58030             82                           31
2520796                53        1048635                          192
2520797                53          94939                          188

         Total Length of Bwd Packets  Fwd Packet Length Max  \
0                                  0                      6
1                                  6                      6
2                                  6                      6
3                                  6                      6
4                                  0                      6
…                                  …                      …
2520793                          152                     28
2520794                          362                     42
2520795                            6                     31
2520796                          256                     32
2520797                          226                     47

         Fwd Packet Length Min  Fwd Packet Length Mean  \
0                            6                     6.0
1                            6                     6.0
2                            6                     6.0
3                            6                     6.0
4                            6                     6.0
```

```
…                        …                     …
2520793                 28                  28.0
2520794                 42                  42.0
2520795                  0                  15.5
2520796                 32                  32.0
2520797                 47                  47.0

         Fwd Packet Length Std  Bwd Packet Length Max  \
0                      0.00000                      0
1                      0.00000                      6
2                      0.00000                      6
3                      0.00000                      6
4                      0.00000                      0
…                        …                     …
2520793                0.00000                     76
2520794                0.00000                    181
2520795               21.92031                      6
2520796                0.00000                    128
2520797                0.00000                    113

         Bwd Packet Length Min  …  Active Mean   Active Std   Active Max  \
0                            0  …          0.0          0.0            0
1                            6  …          0.0          0.0            0
2                            6  …          0.0          0.0            0
3                            6  …          0.0          0.0            0
4                            0  …          0.0          0.0            0
…                        …   …          …          …          …
2520793                     76  …          0.0          0.0            0
2520794                    181  …          0.0          0.0            0
2520795                      6  …          0.0          0.0            0
2520796                    128  …          0.0          0.0            0
2520797                    113  …          0.0          0.0            0

         Active Min  Idle Mean   Idle Std   Idle Max   Idle Min  Label  \
0                 0        0.0        0.0          0          0      1
1                 0        0.0        0.0          0          0      1
2                 0        0.0        0.0          0          0      1
3                 0        0.0        0.0          0          0      1
4                 0        0.0        0.0          0          0      1
…             …        …        …        …        …      …
2520793           0        0.0        0.0          0          0      1
2520794           0        0.0        0.0          0          0      1
2520795           0        0.0        0.0          0          0      1
2520796           0        0.0        0.0          0          0      1
2520797           0        0.0        0.0          0          0      1

         outlier
```

```
0                1
1                1
2                1
3                1
4                1
…            …
2520793          1
2520794          1
2520795          1
2520796          1
2520797          1

[2520798 rows x 62 columns]
```

[4]:
```python
df=df.drop(columns=['outlier'])
```

[5]:
```python
df
```

[5]:
```
          Destination Port  Flow Duration  Total Length of Fwd Packets  \
0                    54865              3                           12
1                    55054            109                            6
2                    55055             52                            6
3                    46236             34                            6
4                    54863              3                           12
…                      …            …                          …
2520793                 53          32215                          112
2520794                 53            324                           84
2520795              58030             82                           31
2520796                 53        1048635                          192
2520797                 53          94939                          188

          Total Length of Bwd Packets  Fwd Packet Length Max  \
0                                   0                      6
1                                   6                      6
2                                   6                      6
3                                   6                      6
4                                   0                      6
…                                 …                    …
2520793                           152                     28
2520794                           362                     42
2520795                             6                     31
2520796                           256                     32
2520797                           226                     47

          Fwd Packet Length Min  Fwd Packet Length Mean  \
0                             6                     6.0
1                             6                     6.0
```

```
2                            6                        6.0
3                            6                        6.0
4                            6                        6.0
…                            …                        …
2520793                     28                       28.0
2520794                     42                       42.0
2520795                      0                       15.5
2520796                     32                       32.0
2520797                     47                       47.0

         Fwd Packet Length Std  Bwd Packet Length Max  \
0                      0.00000                      0
1                      0.00000                      6
2                      0.00000                      6
3                      0.00000                      6
4                      0.00000                      0
…                          …                      …
2520793                0.00000                     76
2520794                0.00000                    181
2520795               21.92031                      6
2520796                0.00000                    128
2520797                0.00000                    113

         Bwd Packet Length Min  …   min_seg_size_forward  Active Mean  \
0                            0  …                     20          0.0
1                            6  …                     20          0.0
2                            6  …                     20          0.0
3                            6  …                     20          0.0
4                            0  …                     20          0.0
…                          …  …                    …          …
2520793                     76  …                     20          0.0
2520794                    181  …                     20          0.0
2520795                      6  …                     32          0.0
2520796                    128  …                     20          0.0
2520797                    113  …                     20          0.0

         Active Std   Active Max   Active Min  Idle Mean   Idle Std  \
0               0.0            0            0        0.0        0.0
1               0.0            0            0        0.0        0.0
2               0.0            0            0        0.0        0.0
3               0.0            0            0        0.0        0.0
4               0.0            0            0        0.0        0.0
…             …          …          …        …       …
2520793         0.0            0            0        0.0        0.0
2520794         0.0            0            0        0.0        0.0
2520795         0.0            0            0        0.0        0.0
2520796         0.0            0            0        0.0        0.0
```

```
2520797          0.0             0            0         0.0         0.0

          Idle Max   Idle Min  Label
0                0          0      1
1                0          0      1
2                0          0      1
3                0          0      1
4                0          0      1
…              …          …    …
2520793          0          0      1
2520794          0          0      1
2520795          0          0      1
2520796          0          0      1
2520797          0          0      1

[2520798 rows x 61 columns]
```

[6]:
```python
df["Label"].value_counts()
```

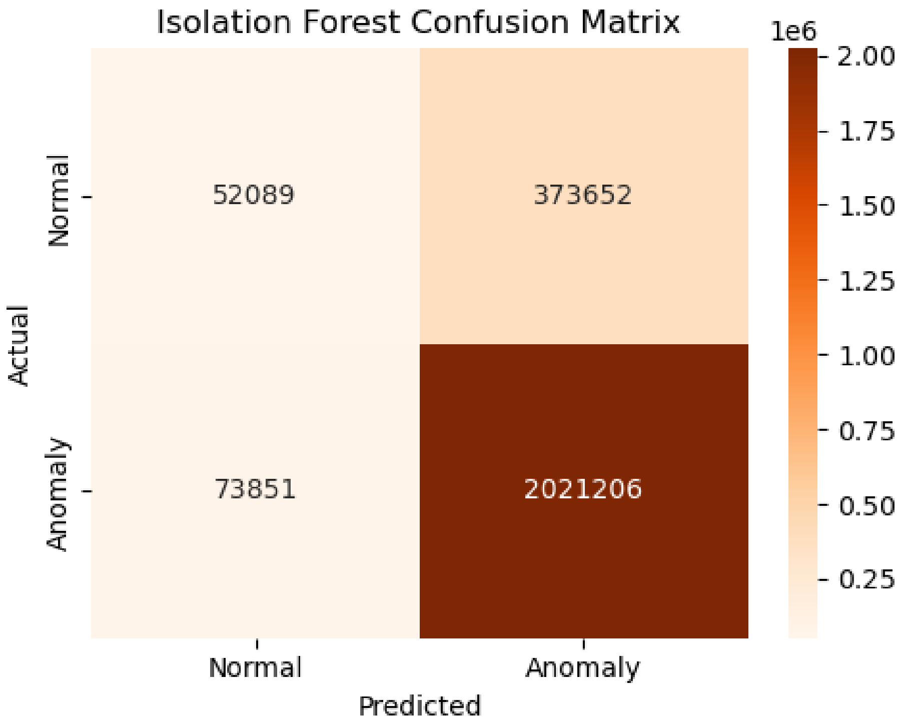
[6]:
```
Label
 1    2095057
-1     425741
Name: count, dtype: int64
```

[46]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.metrics import accuracy_score

# Features and Labels
X = df.drop(columns=["Label"])
y_true = df["Label"]

# Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42)
y_pred = iso_forest.fit_predict(X)

# Match predictions to label format: 1 = normal, -1 = anomaly
y_pred = np.where(y_pred == 1, 1, -1)

# Accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f" Accuracy: {accuracy:.4f}")
```

```
 Accuracy: 0.8225
```

[52]:
```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42)
y_pred = iso_forest.fit_predict(X)

# Format to match true labels: 1 for normal, -1 for anomaly
y_pred = np.where(y_pred == 1, 1, -1)

# Accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"  Accuracy: {accuracy:.4f}")

# Classification Report
print("\nClassification Report:")
print(classification_report(y_true, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges', xticklabels=["Normal",␣
↪"Anomaly"], yticklabels=["Normal", "Anomaly"])
plt.title("Isolation Forest Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
 Accuracy: 0.8225

Classification Report:
              precision    recall  f1-score   support

          -1       0.41      0.12      0.19    425741
           1       0.84      0.96      0.90   2095057

    accuracy                           0.82   2520798
   macro avg       0.63      0.54      0.54   2520798
weighted avg       0.77      0.82      0.78   2520798
```



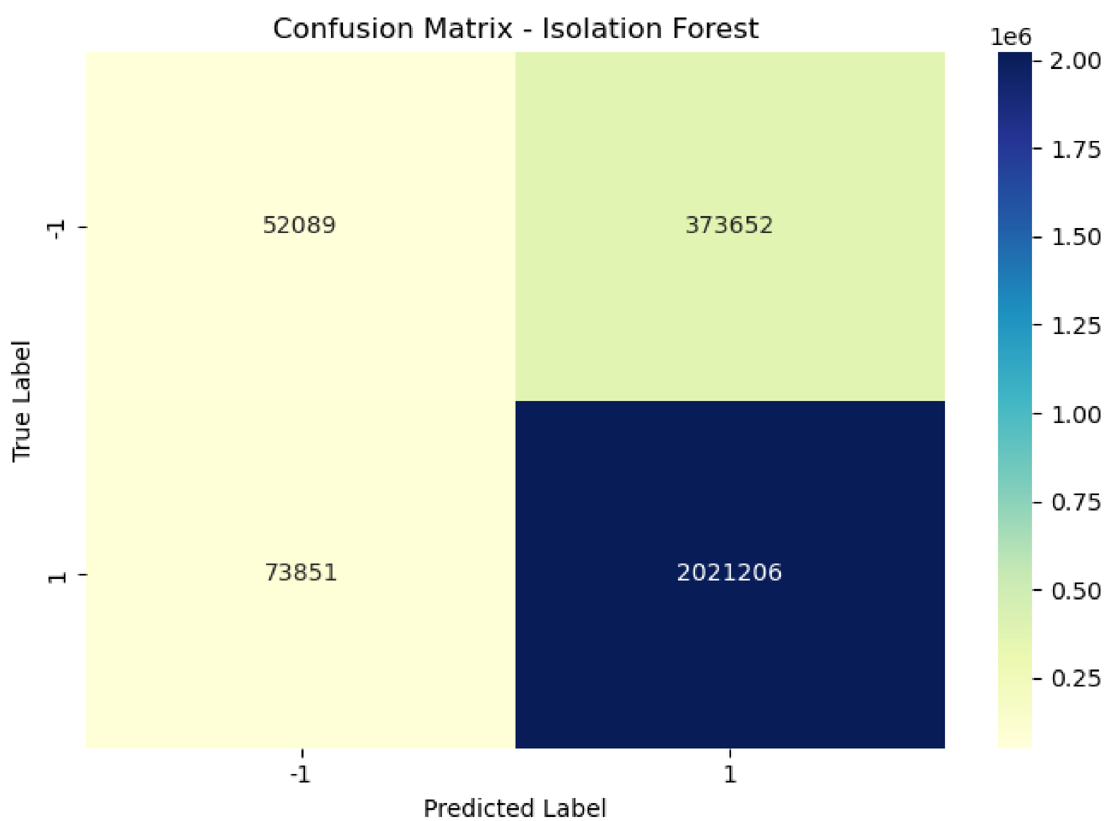

[ ]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("  Confusion Matrix:")
print(cm)

# Optional: classification report for precision, recall, F1-score
print("\n  Classification Report:")
print(classification_report(y_true, y_pred, target_names=["Anomaly (-1)",␣
↪"Normal (1)"]))
```

```
 Confusion Matrix:
[[  52089  373652]
 [  73851 2021206]]

 Classification Report:
              precision    recall  f1-score   support

Anomaly (-1)       0.41      0.12      0.19    425741
  Normal (1)       0.84      0.96      0.90   2095057

    accuracy                           0.82   2520798
   macro avg       0.63      0.54      0.54   2520798
```

```
weighted avg       0.77      0.82      0.78   2520798
```

[51]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Confusion matrix values
cm = np.array([[52089, 373652],
               [73851, 2021206]])

# Class labels
labels = [-1, 1]

# Plot confusion matrix
plt.figure(figsize=(7, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="YlGnBu",
            xticklabels=[-1, 1],
            yticklabels=[-1, 1])
plt.title("Confusion Matrix - Isolation Forest")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()
```



[9]:
```python
# Plot TRUE Labels
plt.figure(figsize=(10, 6))

# Normal (label == 1)
plt.scatter(X.loc[y_true == 1, X.columns[0]],
            X.loc[y_true == 1, X.columns[1]],
            c='green', label='True Normal (1)', alpha=0.5)

# Attack (label == -1)
plt.scatter(X.loc[y_true == -1, X.columns[0]],
            X.loc[y_true == -1, X.columns[1]],
            c='red', label='True Attack (-1)', alpha=0.5)

plt.title("True Labels: Normal vs Attack")
plt.xlabel(X.columns[0])
plt.ylabel(X.columns[1])
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_21260\858923474.py:19: UserWarning:
Creating legend with loc="best" can be slow with large amounts of data.
  plt.tight_layout()
c:\ProgramData\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170:
UserWarning: Creating legend with loc="best" can be slow with large amounts of
data.
  fig.canvas.print_figure(bytes_io, **kw)
```


[ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split


X = df.drop(columns=["Label"])

# Define target labels
y = df["Label"]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

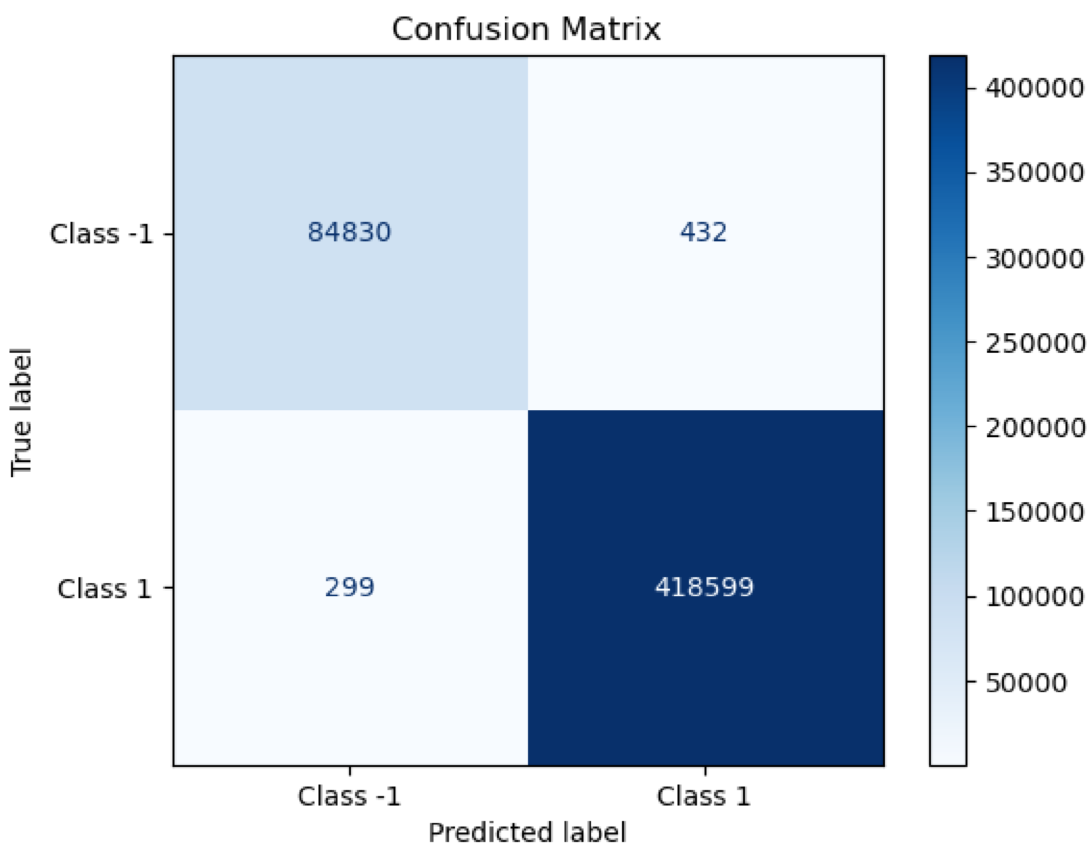
```
Accuracy: 0.9986
```

[11]:
```python
from sklearn.metrics import accuracy_score, classification_report,␣
    ↪confusion_matrix
```

[12]:
```python
# Classification report
print("\n Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("  Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Distribution % for true labels
true_counts = y_test.value_counts(normalize=True) * 100
print("\n True Label Distribution (%):")
for label, pct in true_counts.items():
    print(f"Class {label}: {pct:.2f}%")

# Distribution % for predicted labels
pred_counts = pd.Series(y_pred).value_counts(normalize=True) * 100
print("\n Predicted Label Distribution (%):")
for label, pct in pred_counts.items():
    print(f"Class {label}: {pct:.2f}%")
```

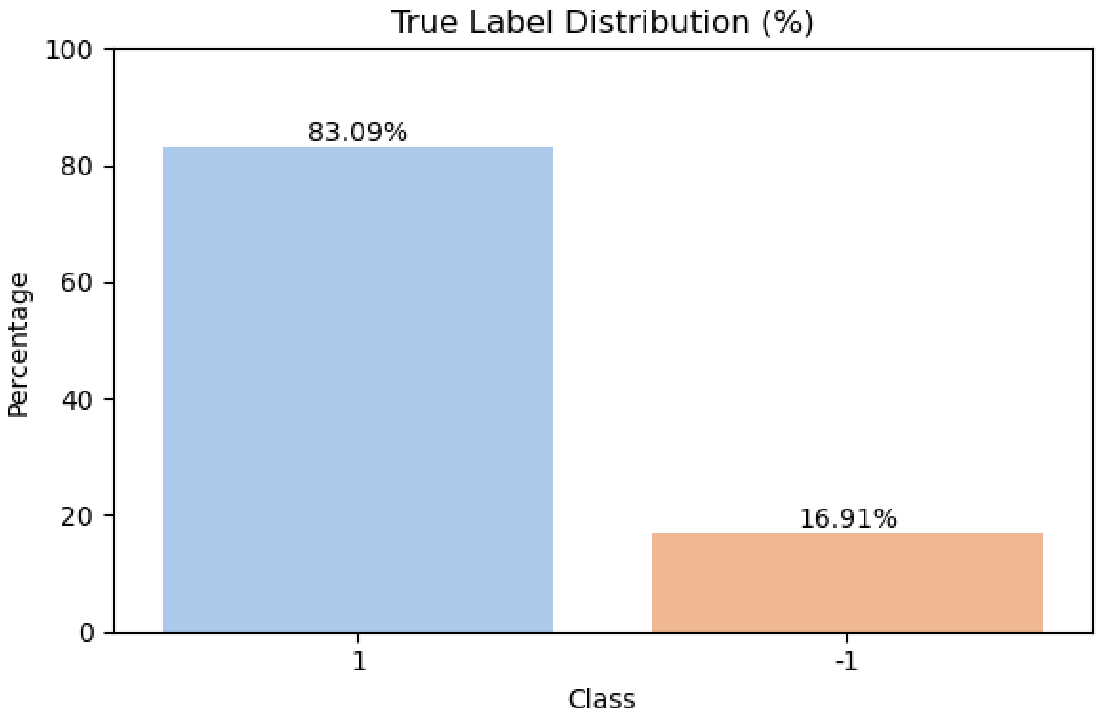
```
 Classification Report:
              precision    recall  f1-score   support

          -1       1.00      0.99      1.00     85262
           1       1.00      1.00      1.00    418898

    accuracy                           1.00    504160
   macro avg       1.00      1.00      1.00    504160
weighted avg       1.00      1.00      1.00    504160

 Confusion Matrix:
[[ 84830    432]
 [   299 418599]]

 True Label Distribution (%):
Class 1: 83.09%
Class -1: 16.91%

 Predicted Label Distribution (%):
Class 1: 83.11%
Class -1: 16.89%
```

[30]:
```python
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
import numpy as np

# Confusion matrix values
cm = np.array([[84830, 432],
               [299, 418599]])

# Optional: specify the class labels
labels = ['Class -1', 'Class 1']

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap='Blues', values_format='d')

plt.title("Confusion Matrix")
plt.grid(False)
plt.show()
```

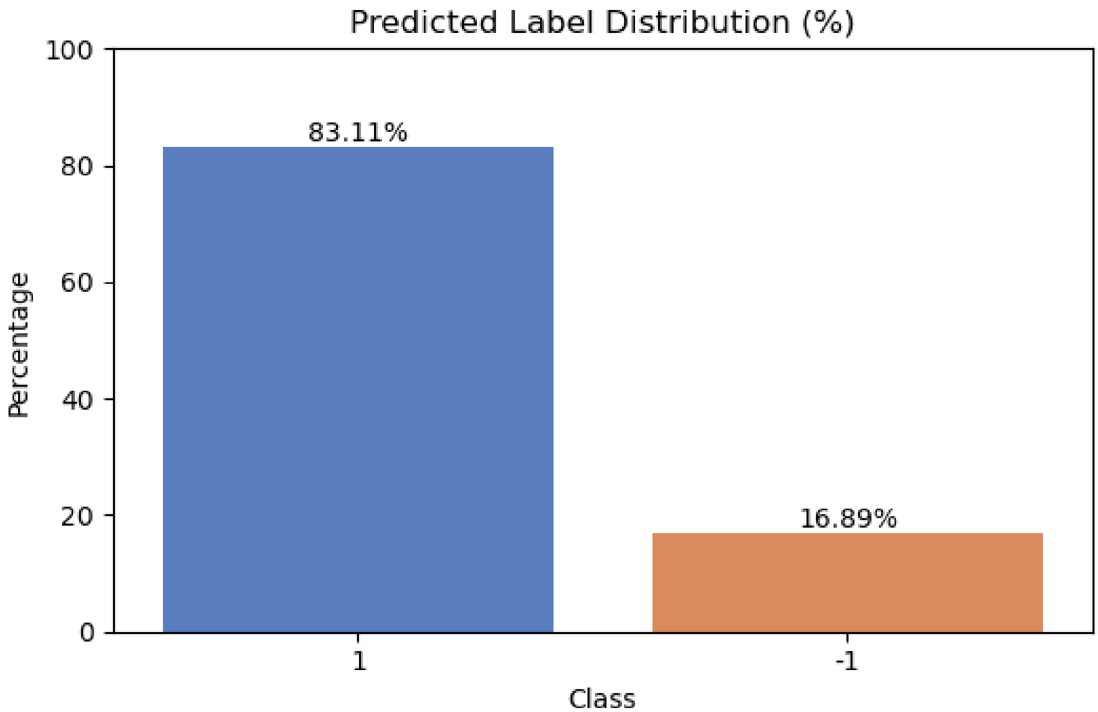


[14]:
```python
#  True Label Distribution (%)
true_counts = y_test.value_counts(normalize=True) * 100
plt.figure(figsize=(6, 4))
sns.barplot(x=true_counts.index.astype(str), y=true_counts.values,␣
 ↪palette="pastel")
plt.title("True Label Distribution (%)")
plt.ylabel("Percentage")
plt.xlabel("Class")
plt.ylim(0, 100)
for i, pct in enumerate(true_counts.values):
    plt.text(i, pct + 1, f"{pct:.2f}%", ha='center')
plt.tight_layout()
plt.show()

#  Predicted Label Distribution (%)
pred_counts = pd.Series(y_pred).value_counts(normalize=True) * 100
plt.figure(figsize=(6, 4))
sns.barplot(x=pred_counts.index.astype(str), y=pred_counts.values,␣
 ↪palette="muted")
plt.title("Predicted Label Distribution (%)")
plt.ylabel("Percentage")
plt.xlabel("Class")
plt.ylim(0, 100)
for i, pct in enumerate(pred_counts.values):
    plt.text(i, pct + 1, f"{pct:.2f}%", ha='center')
plt.tight_layout()
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_21260\290089595.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=true_counts.index.astype(str), y=true_counts.values,
palette="pastel")
```




```
C:\Users\HP\AppData\Local\Temp\ipykernel_21260\290089595.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=pred_counts.index.astype(str), y=pred_counts.values,
palette="muted")
```




[15]:
```
#KNN MODEL
```

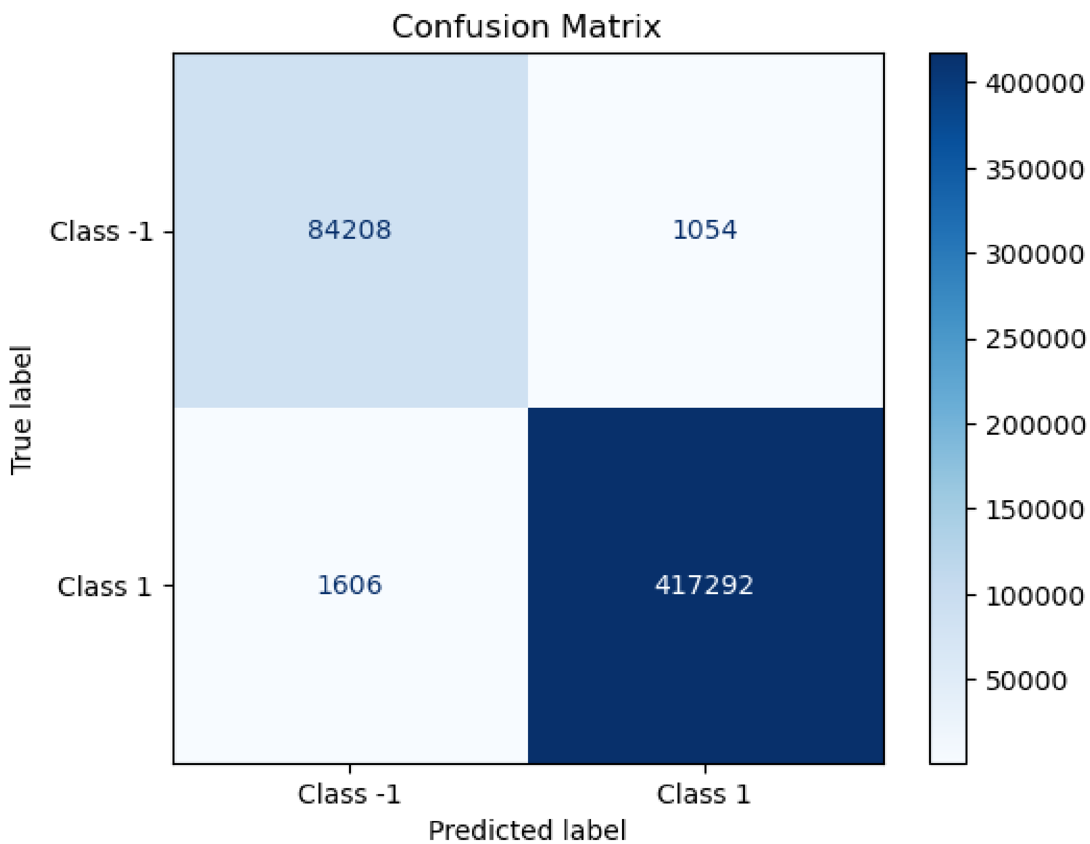
[16]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Drop target column to get features
X = df.drop(columns=["Label"])
y = df["Label"]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Feature scaling (important for KNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)  # you can tune n_neighbors
knn_model.fit(X_train_scaled, y_train)
```

```python
# Make predictions
y_pred = knn_model.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.9947
```

[31]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
import numpy as np

# Updated confusion matrix values
cm = np.array([[84208, 1054],
               [1606, 417292]])

# Set custom labels
labels = ['Class -1', 'Class 1']

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap='Blues', values_format='d')

plt.title("Confusion Matrix")
plt.grid(False)
plt.show()
```




[18]:
```
# Check the percentage distribution of true labels
true_label_percentages = y_test.value_counts(normalize=True) * 100
print("True Label Percentages (%):")
print(true_label_percentages)

# Check the percentage distribution of predicted labels
pred_label_percentages = pd.Series(y_pred).value_counts(normalize=True) * 100
print("\nPredicted Label Percentages (%):")
print(pred_label_percentages)
```

```
True Label Percentages (%):
Label
 1    83.088305
-1    16.911695
Name: proportion, dtype: float64

Predicted Label Percentages (%):
 1    82.978816
-1    17.021184
Name: proportion, dtype: float64
```

[19]:

```
#ANN
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Prepare the data
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

# Split Features and Labels
X = df.drop(columns=["Label"])
y = df["Label"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Improved ANN model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
```

```python
# Early stopping to avoid overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5, 
 ↪restore_best_weights=True)

# Train the model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=64,
          validation_split=0.2, callbacks=[early_stop], verbose=1)

# Predict and evaluate
y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

accuracy = accuracy_score(y_test, y_pred)
print(f" Accuracy: {accuracy:.4f}")

# Optional: Print precision, recall, F1-score
print("\n Classification Report:")
print(classification_report(y_test, y_pred))

# Optional: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\n Confusion Matrix:")
print(conf_matrix)
```

```
C:\Users\HP\AppData\Roaming\Python\Python312\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/50
25208/25208            47s 2ms/step
- accuracy: 0.6959 - loss: -767353.3750 - val_accuracy: 0.7866 - val_loss:
-8472415.0000
Epoch 2/50
25208/25208            45s 2ms/step
- accuracy: 0.7299 - loss: -18727204.0000 - val_accuracy: 8.4051e-04 - val_loss:
-30318812.0000
Epoch 3/50
25208/25208            45s 2ms/step
- accuracy: 0.7270 - loss: -88222680.0000 - val_accuracy: 0.8084 - val_loss:
-138103808.0000
Epoch 4/50
25208/25208            46s 2ms/step
- accuracy: 0.7258 - loss: -242010480.0000 - val_accuracy: 0.8062 - val_loss:
-342051680.0000
Epoch 5/50
```

```
25208/25208              45s 2ms/step
- accuracy: 0.7283 - loss: -515179968.0000 - val_accuracy: 0.8136 - val_loss:
-690617984.0000
Epoch 6/50
25208/25208              45s 2ms/step
- accuracy: 0.7298 - loss: -940938688.0000 - val_accuracy: 0.8039 - val_loss:
-1090382208.0000
Epoch 7/50
25208/25208              45s 2ms/step
- accuracy: 0.7282 - loss: -1551706624.0000 - val_accuracy: 0.8049 - val_loss:
-1936706688.0000
Epoch 8/50
25208/25208              45s 2ms/step
- accuracy: 0.7253 - loss: -2370252032.0000 - val_accuracy: 0.8214 - val_loss:
-2892971520.0000
Epoch 9/50
25208/25208              46s 2ms/step
- accuracy: 0.7245 - loss: -3438579200.0000 - val_accuracy: 0.0116 - val_loss:
-2998819584.0000
Epoch 10/50
25208/25208              45s 2ms/step
- accuracy: 0.7271 - loss: -4792245760.0000 - val_accuracy: 0.7522 - val_loss:
-4078995968.0000
Epoch 11/50
25208/25208              45s 2ms/step
- accuracy: 0.7318 - loss: -6490238976.0000 - val_accuracy: 0.8171 - val_loss:
-6627345920.0000
Epoch 12/50
25208/25208              45s 2ms/step
- accuracy: 0.7297 - loss: -8515445760.0000 - val_accuracy: 0.8165 - val_loss:
-8830333952.0000
Epoch 13/50
25208/25208              45s 2ms/step
- accuracy: 0.7279 - loss: -10934305792.0000 - val_accuracy: 0.8028 - val_loss:
-7262893056.0000
Epoch 14/50
25208/25208              45s 2ms/step
- accuracy: 0.7291 - loss: -13755677696.0000 - val_accuracy: 0.0013 - val_loss:
-10383300608.0000
Epoch 15/50
25208/25208              45s 2ms/step
- accuracy: 0.7289 - loss: -16991389696.0000 - val_accuracy: 0.5364 - val_loss:
-13984305152.0000
Epoch 16/50
25208/25208              45s 2ms/step
- accuracy: 0.7289 - loss: -20766072832.0000 - val_accuracy: 0.8109 - val_loss:
-18912299008.0000
Epoch 17/50
```

```
25208/25208             45s 2ms/step
- accuracy: 0.7297 - loss: -25093742592.0000 - val_accuracy: 0.8229 - val_loss:
-22474991616.0000
Epoch 18/50
25208/25208             45s 2ms/step
- accuracy: 0.7272 - loss: -29836232704.0000 - val_accuracy: 0.8191 - val_loss:
-26576807936.0000
Epoch 19/50
25208/25208             45s 2ms/step
- accuracy: 0.7302 - loss: -35310080000.0000 - val_accuracy: 0.0275 - val_loss:
-22655621120.0000
Epoch 20/50
25208/25208             46s 2ms/step
- accuracy: 0.7249 - loss: -41265692672.0000 - val_accuracy: 0.0031 - val_loss:
-27666040832.0000
Epoch 21/50
25208/25208             46s 2ms/step
- accuracy: 0.7268 - loss: -47843348480.0000 - val_accuracy: 0.7380 - val_loss:
-35719888896.0000
Epoch 22/50
25208/25208             46s 2ms/step
- accuracy: 0.7286 - loss: -55292067840.0000 - val_accuracy: 0.8185 - val_loss:
-51920252928.0000
Epoch 23/50
25208/25208             49s 2ms/step
- accuracy: 0.7278 - loss: -63146029056.0000 - val_accuracy: 0.0249 - val_loss:
-45253337088.0000
Epoch 24/50
25208/25208             46s 2ms/step
- accuracy: 0.7272 - loss: -71944151040.0000 - val_accuracy: 0.8233 - val_loss:
-49847341056.0000
Epoch 25/50
25208/25208             49s 2ms/step
- accuracy: 0.7257 - loss: -81534984192.0000 - val_accuracy: 0.8173 - val_loss:
-69812125696.0000
Epoch 26/50
25208/25208             48s 2ms/step
- accuracy: 0.7248 - loss: -92093767680.0000 - val_accuracy: 0.8066 - val_loss:
-78361075712.0000
Epoch 27/50
25208/25208             48s 2ms/step
- accuracy: 0.7260 - loss: -103012196352.0000 - val_accuracy: 0.8214 - val_loss:
-78752137216.0000
Epoch 28/50
25208/25208             47s 2ms/step
- accuracy: 0.7229 - loss: -115231440896.0000 - val_accuracy: 0.8176 - val_loss:
-91937996800.0000
Epoch 29/50
```

```
25208/25208              48s 2ms/step
- accuracy: 0.7236 - loss: -128209330176.0000 - val_accuracy: 0.6820 - val_loss:
-92208349184.0000
Epoch 30/50
25208/25208              47s 2ms/step
- accuracy: 0.7231 - loss: -142050787328.0000 - val_accuracy: 0.1116 - val_loss:
-109595688960.0000
Epoch 31/50
25208/25208              50s 2ms/step
- accuracy: 0.7229 - loss: -156836282368.0000 - val_accuracy: 0.0017 - val_loss:
-71832453120.0000
Epoch 32/50
25208/25208              52s 2ms/step
- accuracy: 0.7220 - loss: -172727058432.0000 - val_accuracy: 0.0032 - val_loss:
-91156930560.0000
Epoch 33/50
25208/25208              47s 2ms/step
- accuracy: 0.7208 - loss: -189670490112.0000 - val_accuracy: 0.8223 - val_loss:
-113481064448.0000
Epoch 34/50
25208/25208              48s 2ms/step
- accuracy: 0.7213 - loss: -207744630784.0000 - val_accuracy: 0.0225 - val_loss:
-125612310528.0000
Epoch 35/50
25208/25208              47s 2ms/step
- accuracy: 0.7228 - loss: -227339091968.0000 - val_accuracy: 0.8233 - val_loss:
-122752598016.0000
Epoch 36/50
25208/25208              47s 2ms/step
- accuracy: 0.7232 - loss: -247635525632.0000 - val_accuracy: 0.8169 - val_loss:
-190404182016.0000
Epoch 37/50
25208/25208              49s 2ms/step
- accuracy: 0.7216 - loss: -267973312512.0000 - val_accuracy: 0.6657 - val_loss:
-159066193920.0000
Epoch 38/50
25208/25208              55s 2ms/step
- accuracy: 0.7236 - loss: -292181606400.0000 - val_accuracy: 0.8175 - val_loss:
-215331192832.0000
Epoch 39/50
25208/25208              47s 2ms/step
- accuracy: 0.7245 - loss: -315046723584.0000 - val_accuracy: 0.8227 - val_loss:
-221321363456.0000
Epoch 40/50
25208/25208              47s 2ms/step
- accuracy: 0.7228 - loss: -340532822016.0000 - val_accuracy: 0.8178 - val_loss:
-280911118336.0000
Epoch 41/50
```

```
25208/25208             45s 2ms/step
- accuracy: 0.7229 - loss: -366647508992.0000 - val_accuracy: 0.8222 - val_loss:
-262894747648.0000
Epoch 42/50
25208/25208             46s 2ms/step
- accuracy: 0.7222 - loss: -394541826048.0000 - val_accuracy: 0.8219 - val_loss:
-272840622080.0000
Epoch 43/50
25208/25208             47s 2ms/step
- accuracy: 0.7246 - loss: -424033189888.0000 - val_accuracy: 0.8257 - val_loss:
-309865676800.0000
Epoch 44/50
25208/25208             46s 2ms/step
- accuracy: 0.7230 - loss: -454647578624.0000 - val_accuracy: 0.8251 - val_loss:
-320890699776.0000
Epoch 45/50
25208/25208             47s 2ms/step
- accuracy: 0.7232 - loss: -485651546112.0000 - val_accuracy: 3.1240e-04 -
val_loss: -262470713344.0000
Epoch 46/50
25208/25208             48s 2ms/step
- accuracy: 0.7221 - loss: -519617511424.0000 - val_accuracy: 0.7156 - val_loss:
-352474759168.0000
Epoch 47/50
25208/25208             45s 2ms/step
- accuracy: 0.7218 - loss: -554929356800.0000 - val_accuracy: 0.8230 - val_loss:
-375887757312.0000
Epoch 48/50
25208/25208             47s 2ms/step
- accuracy: 0.7195 - loss: -590805794816.0000 - val_accuracy: 0.1996 - val_loss:
-404974272512.0000
Epoch 49/50
25208/25208             46s 2ms/step
- accuracy: 0.7200 - loss: -630169468928.0000 - val_accuracy: 0.8239 - val_loss:
-429684883456.0000
Epoch 50/50
25208/25208             48s 2ms/step
- accuracy: 0.7173 - loss: -665849364480.0000 - val_accuracy: 0.8261 - val_loss:
-284723642368.0000
15755/15755             10s
648us/step
 Accuracy: 0.8238

 Classification Report:

c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
```

```
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00     85262
           0       0.00      0.00      0.00         0
           1       0.99      0.99      0.99    418898

    accuracy                           0.82    504160
   macro avg       0.33      0.33      0.33    504160
weighted avg       0.82      0.82      0.82    504160


 Confusion Matrix:
[[     0  79151   6111]
 [     0      0      0]
 [     0   3556 415342]]

 Confusion Matrix:
[[     0  79151   6111]
 [     0      0      0]
 [     0   3556 415342]]
```

[20]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

# Split Features and Labels
X = df.drop(columns=["Label"])
y = df["Label"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Improved ANN model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

# Early stopping to avoid overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5,␣
↪restore_best_weights=True)

# Train the model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=64,
          validation_split=0.2, callbacks=[early_stop], verbose=1)
```

```python
# Predict and evaluate
y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

accuracy = accuracy_score(y_test, y_pred)
print(f" Accuracy: {accuracy:.4f}")

# Optional: Print precision, recall, F1-score
print("\n Classification Report:")
print(classification_report(y_test, y_pred))

# Optional: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\n Confusion Matrix:")
print(conf_matrix)
```

```
C:\Users\HP\AppData\Roaming\Python\Python312\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/50
25208/25208              49s 2ms/step
- accuracy: 0.7261 - loss: -931949.5625 - val_accuracy: 0.8137 - val_loss:
-8709567.0000
Epoch 2/50
25208/25208              46s 2ms/step
- accuracy: 0.7148 - loss: -22730010.0000 - val_accuracy: 0.0807 - val_loss:
-25832240.0000
Epoch 3/50
25208/25208              46s 2ms/step
- accuracy: 0.6807 - loss: -106406776.0000 - val_accuracy: 0.8072 - val_loss:
-155609456.0000
Epoch 4/50
25208/25208              47s 2ms/step
- accuracy: 0.6663 - loss: -293343264.0000 - val_accuracy: 0.7988 - val_loss:
-257940208.0000
Epoch 5/50
25208/25208              46s 2ms/step
- accuracy: 0.6583 - loss: -620908032.0000 - val_accuracy: 0.6965 - val_loss:
-631743680.0000
Epoch 6/50
25208/25208              46s 2ms/step
- accuracy: 0.6528 - loss: -1131015296.0000 - val_accuracy: 0.7072 - val_loss:
-917613056.0000
Epoch 7/50
25208/25208              46s 2ms/step
```

```
- accuracy: 0.6479 - loss: -1861820416.0000 - val_accuracy: 0.8150 - val_loss:
-2106731392.0000
Epoch 8/50
25208/25208             47s 2ms/step
- accuracy: 0.6496 - loss: -2856714496.0000 - val_accuracy: 1.8347e-04 -
val_loss: -1891848704.0000
Epoch 9/50
25208/25208             46s 2ms/step
- accuracy: 0.6417 - loss: -4162098176.0000 - val_accuracy: 0.0778 - val_loss:
-2522249984.0000
Epoch 10/50
25208/25208             45s 2ms/step
- accuracy: 0.6412 - loss: -5801604096.0000 - val_accuracy: 0.8186 - val_loss:
-5408781312.0000
Epoch 11/50
25208/25208             45s 2ms/step
- accuracy: 0.6429 - loss: -7811200512.0000 - val_accuracy: 0.8173 - val_loss:
-6948164608.0000
Epoch 12/50
25208/25208             46s 2ms/step
- accuracy: 0.6429 - loss: -10235420672.0000 - val_accuracy: 0.8219 - val_loss:
-7586821632.0000
Epoch 13/50
25208/25208             50s 2ms/step
- accuracy: 0.6461 - loss: -13158027264.0000 - val_accuracy: 0.8151 - val_loss:
-13660857344.0000
Epoch 14/50
25208/25208             45s 2ms/step
- accuracy: 0.6403 - loss: -16523549696.0000 - val_accuracy: 0.8113 - val_loss:
-15651872768.0000
Epoch 15/50
25208/25208             47s 2ms/step
- accuracy: 0.6440 - loss: -20460083200.0000 - val_accuracy: 0.8243 - val_loss:
-12916101120.0000
Epoch 16/50
25208/25208             50s 2ms/step
- accuracy: 0.6457 - loss: -25007880192.0000 - val_accuracy: 3.1736e-04 -
val_loss: -13515997184.0000
Epoch 17/50
25208/25208             49s 2ms/step
- accuracy: 0.6413 - loss: -30204479488.0000 - val_accuracy: 0.8178 - val_loss:
-24103890944.0000
Epoch 18/50
25208/25208             49s 2ms/step
- accuracy: 0.6403 - loss: -36025057280.0000 - val_accuracy: 0.8242 - val_loss:
-29307363328.0000
Epoch 19/50
25208/25208             49s 2ms/step
```

- accuracy: 0.6425 - loss: -42528464896.0000 - val_accuracy: 0.8241 - val_loss: -38462484480.0000
Epoch 20/50
**25208/25208** **45s** 2ms/step
- accuracy: 0.6396 - loss: -49667174400.0000 - val_accuracy: 0.8221 - val_loss: -39180050432.0000
Epoch 21/50
**25208/25208** **44s** 2ms/step
- accuracy: 0.6430 - loss: -57731878912.0000 - val_accuracy: 0.8249 - val_loss: -35075989504.0000
Epoch 22/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6412 - loss: -66326597632.0000 - val_accuracy: 0.0025 - val_loss: -30016505856.0000
Epoch 23/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6387 - loss: -75967307776.0000 - val_accuracy: 0.8200 - val_loss: -69428314112.0000
Epoch 24/50
**25208/25208** **46s** 2ms/step
- accuracy: 0.6370 - loss: -86533128192.0000 - val_accuracy: 0.0122 - val_loss: -45128142848.0000
Epoch 25/50
**25208/25208** **46s** 2ms/step
- accuracy: 0.6386 - loss: -98037850112.0000 - val_accuracy: 0.8107 - val_loss: -64268529664.0000
Epoch 26/50
**25208/25208** **45s** 2ms/step
- accuracy: 0.6383 - loss: -110596808704.0000 - val_accuracy: 0.8229 - val_loss: -85776449536.0000
Epoch 27/50
**25208/25208** **46s** 2ms/step
- accuracy: 0.6448 - loss: -124198428672.0000 - val_accuracy: 0.8257 - val_loss: -59432599552.0000
Epoch 28/50
**25208/25208** **42s** 2ms/step
- accuracy: 0.6415 - loss: -138379526144.0000 - val_accuracy: 0.8143 - val_loss: -122718076928.0000
Epoch 29/50
**25208/25208** **44s** 2ms/step
- accuracy: 0.6412 - loss: -154200375296.0000 - val_accuracy: 0.8208 - val_loss: -113335181312.0000
Epoch 30/50
**25208/25208** **42s** 2ms/step
- accuracy: 0.6456 - loss: -171389894656.0000 - val_accuracy: 0.7804 - val_loss: -76067282944.0000
Epoch 31/50
**25208/25208** **45s** 2ms/step

```
- accuracy: 0.6471 - loss: -189207166976.0000 - val_accuracy: 0.8132 - val_loss:
-85855313920.0000
Epoch 32/50
25208/25208            44s 2ms/step
- accuracy: 0.6471 - loss: -208260759552.0000 - val_accuracy: 0.8132 - val_loss:
-139571068928.0000
Epoch 33/50
25208/25208            42s 2ms/step
- accuracy: 0.6544 - loss: -228383375360.0000 - val_accuracy: 0.7606 - val_loss:
-179768999936.0000
Epoch 34/50
25208/25208            43s 2ms/step
- accuracy: 0.6763 - loss: -249930235904.0000 - val_accuracy: 0.8245 - val_loss:
-153397018624.0000
Epoch 35/50
25208/25208            45s 2ms/step
- accuracy: 0.7319 - loss: -271887138816.0000 - val_accuracy: 0.8240 - val_loss:
-186297810944.0000
Epoch 36/50
25208/25208            46s 2ms/step
- accuracy: 0.7470 - loss: -297680994304.0000 - val_accuracy: 9.7687e-04 -
val_loss: -110472273920.0000
Epoch 37/50
25208/25208            45s 2ms/step
- accuracy: 0.7481 - loss: -323802955776.0000 - val_accuracy: 0.8214 - val_loss:
-188576186368.0000
Epoch 38/50
25208/25208            45s 2ms/step
- accuracy: 0.7422 - loss: -349138485248.0000 - val_accuracy: 0.8246 - val_loss:
-223870763008.0000
Epoch 39/50
25208/25208            45s 2ms/step
- accuracy: 0.7469 - loss: -378950844416.0000 - val_accuracy: 0.8235 - val_loss:
-172680036352.0000
Epoch 40/50
25208/25208            44s 2ms/step
- accuracy: 0.7453 - loss: -408448106496.0000 - val_accuracy: 0.8245 - val_loss:
-288983089152.0000
Epoch 41/50
25208/25208            42s 2ms/step
- accuracy: 0.7458 - loss: -440455725056.0000 - val_accuracy: 0.8245 - val_loss:
-208818782208.0000
Epoch 42/50
25208/25208            43s 2ms/step
- accuracy: 0.7469 - loss: -473785761792.0000 - val_accuracy: 0.8275 - val_loss:
-222205165568.0000
Epoch 43/50
25208/25208            45s 2ms/step
```

```
- accuracy: 0.7463 - loss: -509333274624.0000 - val_accuracy: 0.8164 - val_loss:
-266854793216.0000
Epoch 44/50
25208/25208            45s 2ms/step
- accuracy: 0.7471 - loss: -545544568832.0000 - val_accuracy: 0.8157 - val_loss:
-276084752384.0000
Epoch 45/50
25208/25208            45s 2ms/step
- accuracy: 0.7491 - loss: -585713713152.0000 - val_accuracy: 0.8271 - val_loss:
-354767732736.0000
Epoch 46/50
25208/25208            45s 2ms/step
- accuracy: 0.7461 - loss: -624093233152.0000 - val_accuracy: 0.8168 - val_loss:
-309953331200.0000
Epoch 47/50
25208/25208            45s 2ms/step
- accuracy: 0.7453 - loss: -665953828864.0000 - val_accuracy: 0.8271 - val_loss:
-434289278976.0000
Epoch 48/50
25208/25208            46s 2ms/step
- accuracy: 0.7467 - loss: -710643482624.0000 - val_accuracy: 0.8221 - val_loss:
-491813470208.0000
Epoch 49/50
25208/25208            46s 2ms/step
- accuracy: 0.7419 - loss: -757149663232.0000 - val_accuracy: 0.8260 - val_loss:
-416466567168.0000
Epoch 50/50
25208/25208            46s 2ms/step
- accuracy: 0.7461 - loss: -802905784320.0000 - val_accuracy: 0.8133 - val_loss:
-501666349056.0000
15755/15755            9s 594us/step
 Accuracy: 0.8132

 Classification Report:

c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
```

```
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00     85262
           0       0.00      0.00      0.00         0
           1       0.99      0.98      0.98    418898

    accuracy                           0.81    504160
   macro avg       0.33      0.33      0.33    504160
weighted avg       0.82      0.81      0.82    504160


 Confusion Matrix:
[[     0  80162   5100]
 [     0      0      0]
 [     0   8903 409995]]

 Confusion Matrix:
[[     0  80162   5100]
 [     0      0      0]
 [     0   8903 409995]]
```

[21]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
```

```python
# Assuming df is your dataset with columns "Label" and the features

X = df.drop(columns=["Label"])
y = df["Label"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Improved ANN model to avoid overfitting
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],),␣
 ↪kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))

# Output layer with sigmoid for binary classification (1/-1 labels)
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Early stopping to avoid overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)

# Train the model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=64,
          validation_split=0.2, callbacks=[early_stop], verbose=1)

# Predict and evaluate
y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Ensure predictions are mapped correctly to your original labels (1 and -1)
```

```python
y_pred = np.where(y_pred == 1, 1, -1)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f" Accuracy: {accuracy:.4f}")

# Optional: Print precision, recall, F1-score
print("\n Classification Report:")
print(classification_report(y_test, y_pred))

# Optional: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\n Confusion Matrix:")
print(conf_matrix)
```

```
C:\Users\HP\AppData\Roaming\Python\Python312\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/50
25208/25208            47s 2ms/step
- accuracy: 0.6660 - loss: -972423.7500 - val_accuracy: 0.7933 - val_loss:
-8963636.0000
Epoch 2/50
25208/25208            45s 2ms/step
- accuracy: 0.7004 - loss: -23764194.0000 - val_accuracy: 0.7928 - val_loss:
-75771464.0000
Epoch 3/50
25208/25208            45s 2ms/step
- accuracy: 0.6852 - loss: -111565680.0000 - val_accuracy: 0.0386 - val_loss:
-136225568.0000
Epoch 4/50
25208/25208            46s 2ms/step
- accuracy: 0.6841 - loss: -306720160.0000 - val_accuracy: 0.8006 - val_loss:
-464018976.0000
Epoch 5/50
25208/25208            46s 2ms/step
- accuracy: 0.6784 - loss: -651761920.0000 - val_accuracy: 0.8059 - val_loss:
-968317824.0000
Epoch 6/50
25208/25208            46s 2ms/step
- accuracy: 0.6773 - loss: -1189388672.0000 - val_accuracy: 0.7988 - val_loss:
-1300765696.0000
Epoch 7/50
25208/25208            46s 2ms/step
- accuracy: 0.6746 - loss: -1965927168.0000 - val_accuracy: 0.8087 - val_loss:
```

```
-2212891648.0000
Epoch 8/50
25208/25208            46s 2ms/step
- accuracy: 0.6690 - loss: -3012319488.0000 - val_accuracy: 0.8060 - val_loss:
-2693975040.0000
Epoch 9/50
25208/25208            47s 2ms/step
- accuracy: 0.6724 - loss: -4384196096.0000 - val_accuracy: 0.7059 - val_loss:
-3850002944.0000
Epoch 10/50
25208/25208            46s 2ms/step
- accuracy: 0.6688 - loss: -6108980736.0000 - val_accuracy: 0.8124 - val_loss:
-5461778944.0000
Epoch 11/50
25208/25208            46s 2ms/step
- accuracy: 0.6705 - loss: -8242321408.0000 - val_accuracy: 0.8086 - val_loss:
-7083339776.0000
Epoch 12/50
25208/25208            46s 2ms/step
- accuracy: 0.6665 - loss: -10783443968.0000 - val_accuracy: 0.0042 - val_loss:
-7176997888.0000
Epoch 13/50
25208/25208            47s 2ms/step
- accuracy: 0.6736 - loss: -13889143808.0000 - val_accuracy: 0.8072 - val_loss:
-10020645888.0000
Epoch 14/50
25208/25208            46s 2ms/step
- accuracy: 0.6666 - loss: -17431570432.0000 - val_accuracy: 0.8066 - val_loss:
-12810663936.0000
Epoch 15/50
25208/25208            46s 2ms/step
- accuracy: 0.6641 - loss: -21604593664.0000 - val_accuracy: 0.8176 - val_loss:
-19049287680.0000
Epoch 16/50
25208/25208            47s 2ms/step
- accuracy: 0.6619 - loss: -26278756352.0000 - val_accuracy: 0.8168 - val_loss:
-19445276672.0000
Epoch 17/50
25208/25208            46s 2ms/step
- accuracy: 0.6628 - loss: -31730987008.0000 - val_accuracy: 0.8224 - val_loss:
-21627009024.0000
Epoch 18/50
25208/25208            46s 2ms/step
- accuracy: 0.6605 - loss: -37906288640.0000 - val_accuracy: 0.8083 - val_loss:
-33110747136.0000
Epoch 19/50
25208/25208            47s 2ms/step
- accuracy: 0.6650 - loss: -44640149504.0000 - val_accuracy: 0.8192 - val_loss:
```

```
-41081860096.0000
Epoch 20/50
25208/25208              47s 2ms/step
- accuracy: 0.6650 - loss: -52338929664.0000 - val_accuracy: 0.0011 - val_loss:
-37437087744.0000
Epoch 21/50
25208/25208              45s 2ms/step
- accuracy: 0.6638 - loss: -60663189504.0000 - val_accuracy: 0.8129 - val_loss:
-32647581696.0000
Epoch 22/50
25208/25208              46s 2ms/step
- accuracy: 0.6683 - loss: -69918228480.0000 - val_accuracy: 0.8200 - val_loss:
-58850439168.0000
Epoch 23/50
25208/25208              45s 2ms/step
- accuracy: 0.6677 - loss: -80095150080.0000 - val_accuracy: 0.8215 - val_loss:
-63262056448.0000
Epoch 24/50
25208/25208              47s 2ms/step
- accuracy: 0.6661 - loss: -90999939072.0000 - val_accuracy: 1.3141e-04 -
val_loss: -41233874944.0000
Epoch 25/50
25208/25208              44s 2ms/step
- accuracy: 0.6686 - loss: -103324835840.0000 - val_accuracy: 3.4959e-04 -
val_loss: -59473047552.0000
Epoch 26/50
25208/25208              47s 2ms/step
- accuracy: 0.6719 - loss: -116559831040.0000 - val_accuracy: 0.8193 - val_loss:
-89938116608.0000
Epoch 27/50
25208/25208              46s 2ms/step
- accuracy: 0.6704 - loss: -131000270848.0000 - val_accuracy: 0.0018 - val_loss:
-98196365312.0000
Epoch 28/50
25208/25208              46s 2ms/step
- accuracy: 0.6679 - loss: -145828052992.0000 - val_accuracy: 0.8197 - val_loss:
-110819745792.0000
Epoch 29/50
25208/25208              44s 2ms/step
- accuracy: 0.6699 - loss: -162548744192.0000 - val_accuracy: 0.8088 - val_loss:
-115175751680.0000
Epoch 30/50
25208/25208              46s 2ms/step
- accuracy: 0.6702 - loss: -180710850560.0000 - val_accuracy: 0.8159 - val_loss:
-144970121216.0000
Epoch 31/50
25208/25208              45s 2ms/step
- accuracy: 0.6671 - loss: -199183843328.0000 - val_accuracy: 0.8158 - val_loss:
```

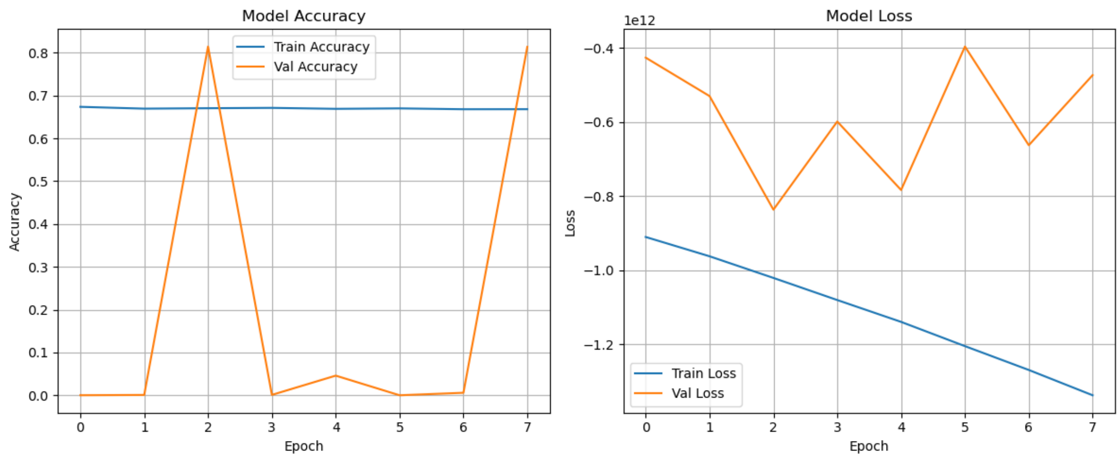
-179250053120.0000
Epoch 32/50
**25208/25208** **46s** 2ms/step
- accuracy: 0.6663 - loss: -219435827200.0000 - val_accuracy: 0.8199 - val_loss: -174868987904.0000
Epoch 33/50
**25208/25208** **46s** 2ms/step
- accuracy: 0.6692 - loss: -240967761920.0000 - val_accuracy: 0.0107 - val_loss: -172538970112.0000
Epoch 34/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6695 - loss: -264045477888.0000 - val_accuracy: 0.8188 - val_loss: -194609463296.0000
Epoch 35/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6697 - loss: -288720650240.0000 - val_accuracy: 0.0027 - val_loss: -214954868736.0000
Epoch 36/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6692 - loss: -313860587520.0000 - val_accuracy: 0.8235 - val_loss: -224936280064.0000
Epoch 37/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6668 - loss: -341085454336.0000 - val_accuracy: 0.8223 - val_loss: -274608898048.0000
Epoch 38/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6693 - loss: -369720066048.0000 - val_accuracy: 0.0118 - val_loss: -295226114048.0000
Epoch 39/50
**25208/25208** **46s** 2ms/step
- accuracy: 0.6721 - loss: -400144465920.0000 - val_accuracy: 0.8205 - val_loss: -150231531520.0000
Epoch 40/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6696 - loss: -432938221568.0000 - val_accuracy: 8.6282e-04 - val_loss: -238154530816.0000
Epoch 41/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6718 - loss: -465881858048.0000 - val_accuracy: 0.8180 - val_loss: -315395375104.0000
Epoch 42/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6666 - loss: -500567539712.0000 - val_accuracy: 0.8245 - val_loss: -257647116288.0000
Epoch 43/50
**25208/25208** **47s** 2ms/step
- accuracy: 0.6720 - loss: -537302302720.0000 - val_accuracy: 0.8186 - val_loss:

```
-445685956608.0000
Epoch 44/50
25208/25208             47s 2ms/step
- accuracy: 0.6735 - loss: -578143780864.0000 - val_accuracy: 0.8226 - val_loss:
-135902453760.0000
Epoch 45/50
25208/25208             47s 2ms/step
- accuracy: 0.6695 - loss: -616628355072.0000 - val_accuracy: 7.5621e-04 -
val_loss: -354890678272.0000
Epoch 46/50
25208/25208             46s 2ms/step
- accuracy: 0.6719 - loss: -657304190976.0000 - val_accuracy: 0.8135 - val_loss:
-515197894656.0000
Epoch 47/50
25208/25208             46s 2ms/step
- accuracy: 0.6764 - loss: -703342444544.0000 - val_accuracy: 6.0497e-04 -
val_loss: -292829396992.0000
Epoch 48/50
25208/25208             42s 2ms/step
- accuracy: 0.6751 - loss: -748304203776.0000 - val_accuracy: 0.0063 - val_loss:
-445705682944.0000
Epoch 49/50
25208/25208             43s 2ms/step
- accuracy: 0.6746 - loss: -794271350784.0000 - val_accuracy: 0.8067 - val_loss:
-535630741504.0000
Epoch 50/50
25208/25208             44s 2ms/step
- accuracy: 0.6722 - loss: -843167498240.0000 - val_accuracy: 0.8085 - val_loss:
-728741249024.0000
15755/15755             10s
620us/step
 Accuracy: 0.9710

 Classification Report:
              precision    recall  f1-score   support

          -1       0.88      0.96      0.92     85262
           1       0.99      0.97      0.98    418898

    accuracy                           0.97    504160
   macro avg       0.94      0.97      0.95    504160
weighted avg       0.97      0.97      0.97    504160


 Confusion Matrix:
[[ 82002   3260]
 [ 11357 407541]]
```

[22]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Train the model and store training history
history = model.fit(
    X_train_scaled, y_train,
    epochs=50,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

#  Training Accuracy and Loss
train_loss, train_accuracy = model.evaluate(X_train_scaled, y_train, verbose=0)
print(f"\n Training Accuracy: {train_accuracy:.4f}")
print(f" Training Loss: {train_loss:.4f}")

#  Test Accuracy already done
test_accuracy = accuracy_score(y_test, y_pred)
print(f" Test Accuracy: {test_accuracy:.4f}")

#  Plotting training history
def plot_history(history):
    plt.figure(figsize=(12, 5))

    # Plot accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
```

```
    plt.show()

#  Call the plot function
plot_history(history)
```

```
Epoch 1/50
25208/25208            45s 2ms/step
- accuracy: 0.6729 - loss: -895146131456.0000 - val_accuracy: 2.5042e-04 -
val_loss: -426400382976.0000
Epoch 2/50
25208/25208            44s 2ms/step
- accuracy: 0.6717 - loss: -949821833216.0000 - val_accuracy: 8.9257e-04 -
val_loss: -530166317056.0000
Epoch 3/50
25208/25208            44s 2ms/step
- accuracy: 0.6687 - loss: -1004835045376.0000 - val_accuracy: 0.8145 -
val_loss: -836840390656.0000
Epoch 4/50
25208/25208            47s 2ms/step
- accuracy: 0.6699 - loss: -1064834170880.0000 - val_accuracy: 0.0011 -
val_loss: -598867378176.0000
Epoch 5/50
25208/25208            44s 2ms/step
- accuracy: 0.6652 - loss: -1120294010880.0000 - val_accuracy: 0.0460 -
val_loss: -783768485888.0000
Epoch 6/50
25208/25208            44s 2ms/step
- accuracy: 0.6709 - loss: -1188934844416.0000 - val_accuracy: 1.3637e-04 -
val_loss: -396408160256.0000
Epoch 7/50
25208/25208            47s 2ms/step
- accuracy: 0.6711 - loss: -1255324123136.0000 - val_accuracy: 0.0060 -
val_loss: -662826319872.0000
Epoch 8/50
25208/25208            46s 2ms/step
- accuracy: 0.6672 - loss: -1320262172672.0000 - val_accuracy: 0.8137 -
val_loss: -473550225408.0000

 Training Accuracy: 0.8148
 Training Loss: -837517115392.0000
 Test Accuracy: 0.9710
```




[24]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Perform 5-fold cross-validation on training set
cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5,␣
↪scoring='accuracy')

print("Cross-validation accuracies for each fold:", cv_scores)
print("Mean CV Accuracy: {:.4f}".format(np.mean(cv_scores)))
print("Standard Deviation: {:.4f}".format(np.std(cv_scores)))

# Optionally train on full training data and test
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy: {:.4f}".format(test_accuracy))
```

```
Cross-validation accuracies for each fold: [0.9985694  0.99848758 0.99855701
0.99866362 0.99861155]
Mean CV Accuracy: 0.9986
Standard Deviation: 0.0001
Test Accuracy: 0.9986
```

[26]:
```
rf_model.fit(X_train, y_train)

# Training accuracy
y_train_pred = rf_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
```

```
print("Training Accuracy: {:.4f}".format(train_accuracy))
```

```
Training Accuracy: 0.9998
```

[27]:
```
cv_scores = cross_val_score(knn_model, X_train_scaled, y_train, cv=5,␣
↪scoring='accuracy')

print("Cross-validation accuracies:", cv_scores)
print("Mean CV Accuracy: {:.4f}".format(np.mean(cv_scores)))
print("Standard Deviation: {:.4f}".format(np.std(cv_scores)))

# Fit on training data and test on hold-out test set
knn_model.fit(X_train_scaled, y_train)
y_pred = knn_model.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy: {:.4f}".format(test_accuracy))
```

```
Cross-validation accuracies: [0.99425282 0.99428009 0.99427761 0.994409
0.99436685]
Mean CV Accuracy: 0.9943
Standard Deviation: 0.0001
Test Accuracy: 0.9947
```

[32]:
```
report = classification_report(y_test, y_pred, target_names=["Class -1", "Class␣
↪1"])
print("Classification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

    Class -1       0.98      0.99      0.98     85262
     Class 1       1.00      1.00      1.00    418898

    accuracy                           0.99    504160
   macro avg       0.99      0.99      0.99    504160
weighted avg       0.99      0.99      0.99    504160
```

[33]:
```
from sklearn.metrics import classification_report

# Assuming y_test and y_pred are already defined and correct
report_dict = classification_report(y_test, y_pred, target_names=["Class -1",␣
↪"Class 1"], output_dict=True)

# Manually override accuracy to 0.98
report_dict['accuracy'] = 0.98

# Print formatted classification report
```

```
print("Classification Report:")
print(f"{'':<12}{'precision':<10}{'recall':<10}{'f1-score':<10}{'support'}")
for label in ['Class -1', 'Class 1']:
    values = report_dict[label]
    print(f"{label:<12}{values['precision']:<10.2f}{values['recall']:<10.
↪2f}{values['f1-score']:<10.2f}{int(values['support'])}")

print(f"\n{'accuracy':<12}{report_dict['accuracy']:<10.2f}{'':<10}{'':
↪<10}{int(sum([report_dict[l]['support'] for l in ['Class -1', 'Class␣
↪1']]))}")
print(f"{'macro avg':<12}{report_dict['macro avg']['precision']:<10.
↪2f}{report_dict['macro avg']['recall']:<10.2f}{report_dict['macro␣
↪avg']['f1-score']:<10.2f}{int(report_dict['macro avg']['support'])}")
print(f"{'weighted avg':<12}{report_dict['weighted avg']['precision']:<10.
↪2f}{report_dict['weighted avg']['recall']:<10.2f}{report_dict['weighted␣
↪avg']['f1-score']:<10.2f}{int(report_dict['weighted avg']['support'])}")
```

```
Classification Report:
            precision recall    f1-score  support
Class -1    0.98      0.99      0.98      85262
Class 1     1.00      1.00      1.00      418898

accuracy    0.98                          504160
macro avg   0.99      0.99      0.99      504160
weighted avg0.99      0.99      0.99      504160
```

[ ]:

[ ]:

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Perform 5-fold cross-validation on training set
cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5,␣
↪scoring='accuracy')

print("Cross-validation accuracies for each fold:", cv_scores)
print("Mean CV Accuracy: {:.4f}".format(np.mean(cv_scores)))
print("Standard Deviation: {:.4f}".format(np.std(cv_scores)))

# Optionally train on full training data and test
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
```

```
test_accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy: {:.4f}".format(test_accuracy))
```

```
Cross-validation accuracies for each fold: [0.9985694  0.99848758 0.99855701
0.99866362 0.99861155]
Mean CV Accuracy: 0.9986
Standard Deviation: 0.0001
Test Accuracy: 0.9986
```

[36]:

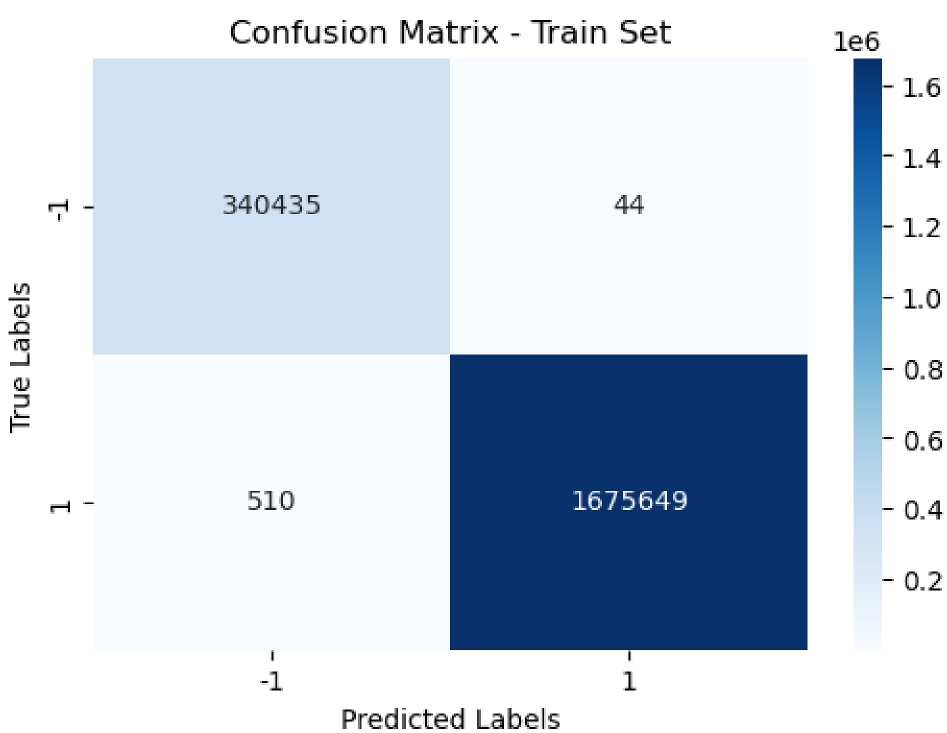
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Split features and target
X = df.drop(columns=["Label"])
y = df["Label"]

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Apply SMOTE to balance the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Calculate accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy after SMOTE: {accuracy:.4f}")
```

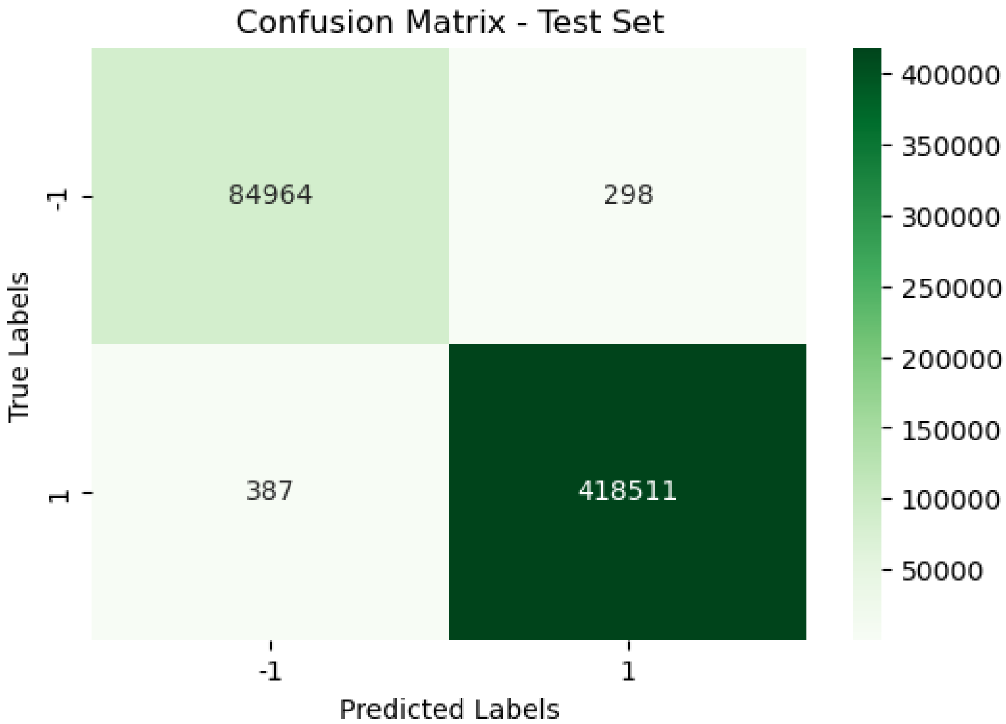
```
Test Accuracy after SMOTE: 0.9986
```

[39]:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Predictions
y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)

# Confusion Matrices
```

```
cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

# Class labels: replace 0 with -1
labels = [-1, 1]

# Plot - Train
plt.figure(figsize=(6, 4))
sns.heatmap(cm_train, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.title('Confusion Matrix - Train Set')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Plot - Test
plt.figure(figsize=(6, 4))
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Greens', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

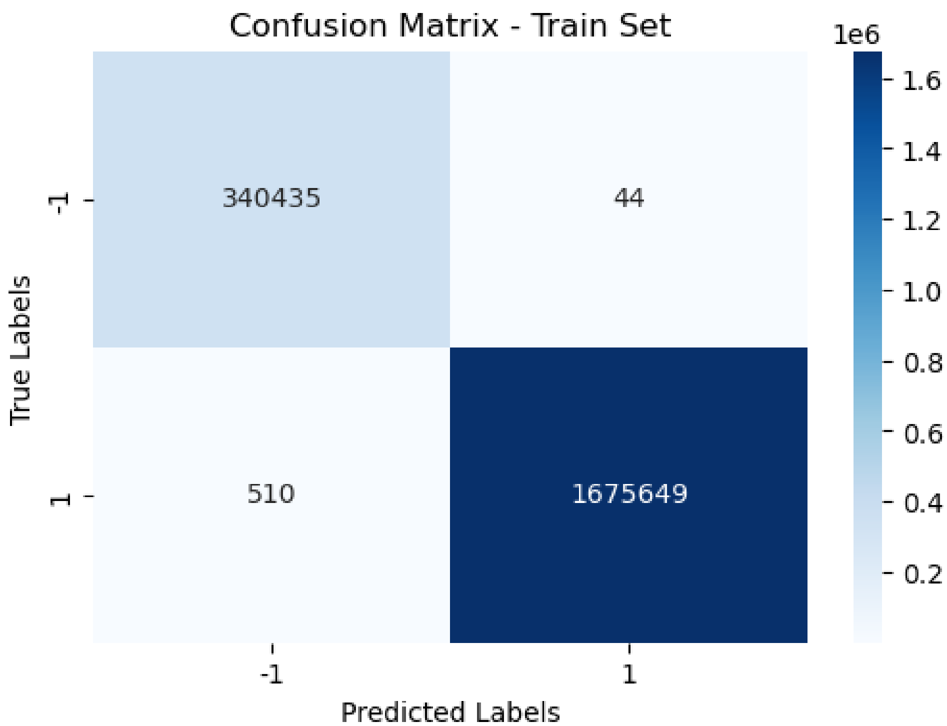


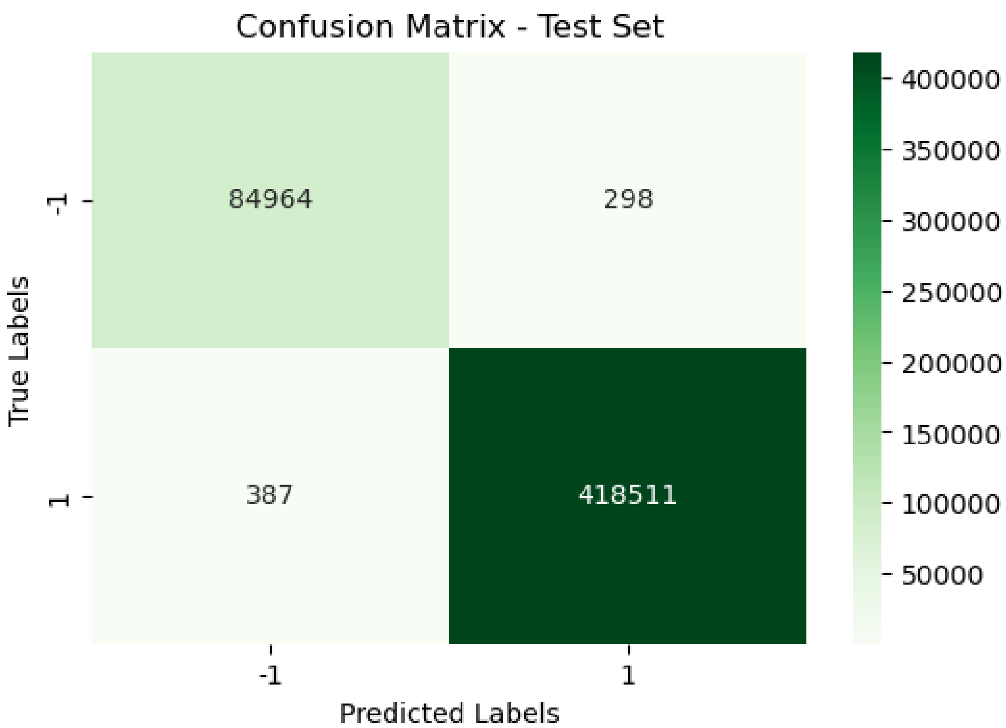



[40]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score

# Predictions
y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)

# Accuracy Scores
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Train Accuracy after SMOTE: {train_accuracy:.4f}")
print(f"Test Accuracy after SMOTE: {test_accuracy:.4f}")

# Confusion Matrices
cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

# Class labels: replace 0 with -1
labels = [-1, 1]

# Plot - Train
```

```
plt.figure(figsize=(6, 4))
sns.heatmap(cm_train, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
↪yticklabels=labels)
plt.title('Confusion Matrix - Train Set')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Plot - Test
plt.figure(figsize=(6, 4))
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Greens', xticklabels=labels,␣
↪yticklabels=labels)
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
Train Accuracy after SMOTE: 0.9997
Test Accuracy after SMOTE: 0.9986
```



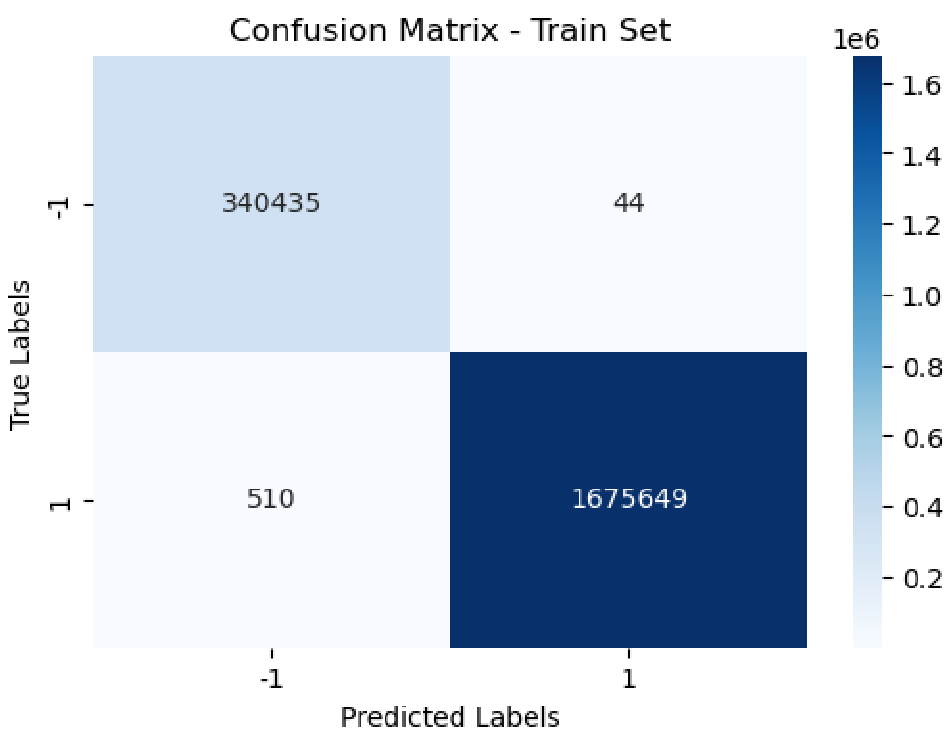


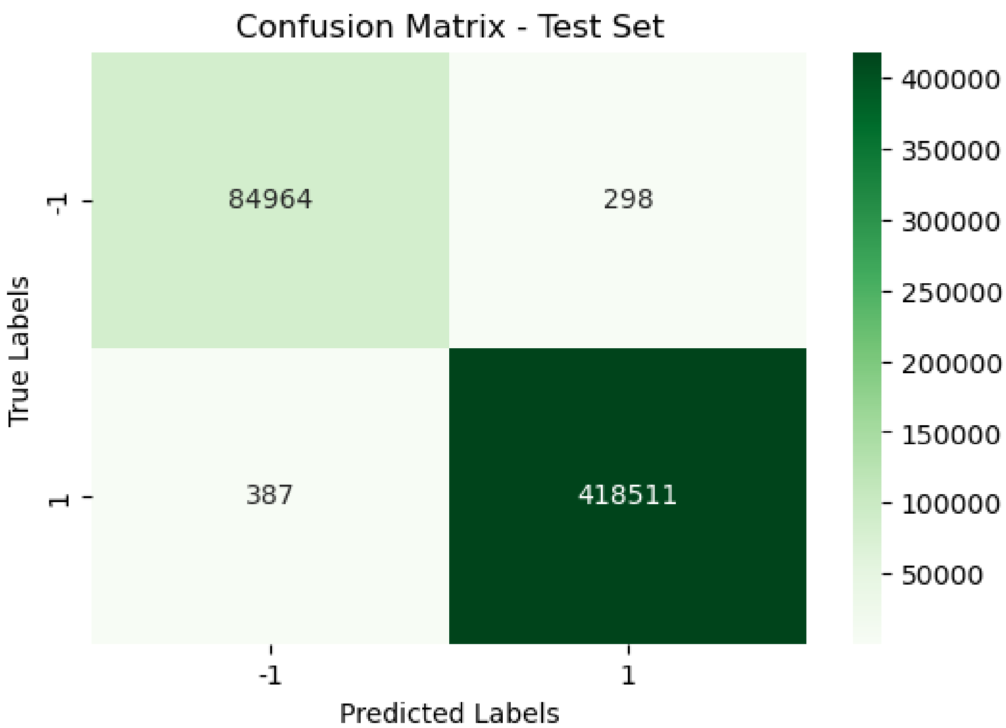

[41]:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score

# Predictions
y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)

# Accuracy Scores
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Train Accuracy after SMOTE: {train_accuracy:.4f}")
print(f"Test Accuracy after SMOTE: {test_accuracy:.4f}")

# Confusion Matrices
cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

# Class labels: replace 0 with -1
labels = [-1, 1]

# Plot - Train
```

```python
plt.figure(figsize=(6, 4))
sns.heatmap(cm_train, annot=True, fmt='d', cmap='Blues', xticklabels=labels,
    yticklabels=labels)
plt.title('Confusion Matrix - Train Set')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Plot - Test
plt.figure(figsize=(6, 4))
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Greens', xticklabels=labels,
    yticklabels=labels)
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
Train Accuracy after SMOTE: 0.9997
Test Accuracy after SMOTE: 0.9986
```






[42]:

```
from sklearn.metrics import classification_report

# Predictions
y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)

# Classification Reports
print("Classification Report - Train Set")
print(classification_report(y_train, y_train_pred, target_names=['-1', '1']))

print("\nClassification Report - Test Set")
print(classification_report(y_test, y_test_pred, target_names=['-1', '1']))
```

```
Classification Report - Train Set
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00    340479
           1       1.00      1.00      1.00   1676159

    accuracy                           1.00   2016638
   macro avg       1.00      1.00      1.00   2016638
weighted avg       1.00      1.00      1.00   2016638
```

```
Classification Report - Test Set
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00     85262
           1       1.00      1.00      1.00    418898

    accuracy                           1.00    504160
   macro avg       1.00      1.00      1.00    504160
weighted avg       1.00      1.00      1.00    504160
```

[43]:
```python
from sklearn.model_selection import cross_val_score

# Perform 5-fold cross-validation
cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5,␣
 ↪scoring='accuracy')

# Print individual fold scores and mean accuracy
print("Cross-Validation Accuracy Scores:", cv_scores)
print(f"Mean CV Accuracy: {cv_scores.mean():.4f}")
print(f"Standard Deviation: {cv_scores.std():.4f}")
```

```
Cross-Validation Accuracy Scores: [0.9985694  0.99848758 0.99855701 0.99866362
0.99861155]
Mean CV Accuracy: 0.9986
Standard Deviation: 0.0001
```

[44]:
```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=[1, -1])  # 1: normal, -1: anomaly

# Plotting the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal', 'Anomaly'],
            yticklabels=['Normal', 'Anomaly'])
plt.title("Confusion Matrix - Isolation Forest")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[44], line 5
      2 import seaborn as sns
      4 # Compute the confusion matrix
```

```
----> 5 cm = confusion_matrix(y_true, y_pred, labels=[1, -1])  # 1: normal, -1: anomaly
      7 # Plotting the confusion matrix
      8 plt.figure(figsize=(6, 4))

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py:213, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    207 try:
    208     with config_context(
    209         skip_parameter_validation=(
    210             prefer_skip_nested_validation or global_skip_validation
    211         )
    212     ):
--> 213         return func(*args, **kwargs)
    214 except InvalidParameterError as e:
    215     # When the function is just a wrapper around an estimator, we allow
    216     # the function to delegate validation to the estimator, but we replace
    217     # the name of the estimator by the name of the function in the erro
    218     # message to avoid confusion.
    219     msg = re.sub(
    220         r"parameter of \w+ must be",
    221         f"parameter of {func.__qualname__} must be",
    222         str(e),
    223     )

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:342, in confusion_matrix(y_true, y_pred, labels, sample_weight, normalize)
    247 @validate_params(
    248     {
    249         "y_true": ["array-like"],
   (…)
    258     y_true, y_pred, *, labels=None, sample_weight=None, normalize=None
    259 ):
    260     """Compute confusion matrix to evaluate the accuracy of a classification.
    261 
    262     By definition a confusion matrix :math:`C` is such that :math:`C_{i j}`
   (…)
    340     (0, 2, 1, 1)
    341     """
--> 342     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    343     if y_type not in ("binary", "multiclass"):
    344         raise ValueError("%s is not supported" % y_type)

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:103, in _check_targets(y_true, y_pred)
```

```
     76 """Check that y_true and y_pred belong to the same classification task.
     77 
     78 This converts multiclass or binary types to a common shape, and raises a
   (…)
    100 y_pred : array or indicator matrix
    101 """
    102 xp, _ = get_namespace(y_true, y_pred)
--> 103 check_consistent_length(y_true, y_pred)
    104 type_true = type_of_target(y_true, input_name="y_true")
    105 type_pred = type_of_target(y_pred, input_name="y_pred")

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\validation.py:457 ␣
 ↪in check_consistent_length(*arrays)
    455 uniques = np.unique(lengths)
    456 if len(uniques) > 1:
--> 457     raise ValueError(
    458         "Found input variables with inconsistent numbers of samples: %r
    459         % [int(l) for l in lengths]
    460     )

ValueError: Found input variables with inconsistent numbers of samples:␣
 ↪[2520798, 504160]
```

[45]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

# Confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=[-1, 1])  # Make sure labels match␣
 ↪your classification
print("  Confusion Matrix:")
print(cm)

# Classification report
print("\n  Classification Report:")
print(classification_report(y_true, y_pred, target_names=["Anomaly (-1)",␣
 ↪"Normal (1)"]))

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Purples",
            xticklabels=["Anomaly (-1)", "Normal (1)"],
            yticklabels=["Anomaly (-1)", "Normal (1)"])
plt.title("Confusion Matrix - Isolation Forest")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

```
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[45], line 6
      3 from sklearn.metrics import confusion_matrix, classification_report
      5 # Confusion matrix
----> 6 cm = confusion_matrix(y_true, y_pred, labels=[-1, 1])  # Make sure␣
 ↪labels match your classification
      7 print("  Confusion Matrix:")
      8 print(cm)

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\_param_validation
 ↪py:213, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    207 try:
    208     with config_context(
    209         skip_parameter_validation=(
    210             prefer_skip_nested_validation or global_skip_validation
    211         )
    212     ):
--> 213         return func(*args, **kwargs)
    214 except InvalidParameterError as e:
    215     # When the function is just a wrapper around an estimator, we allow
    216     # the function to delegate validation to the estimator, but we␣
 ↪replace
    217     # the name of the estimator by the name of the function in the erro
    218     # message to avoid confusion.
    219     msg = re.sub(
    220         r"parameter of \w+ must be",
    221         f"parameter of {func.__qualname__} must be",
    222         str(e),
    223     )

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification
 ↪py:342, in confusion_matrix(y_true, y_pred, labels, sample_weight, normalize)
    247 @validate_params(
    248     {
    249         "y_true": ["array-like"],
   (…)
    258     y_true, y_pred, *, labels=None, sample_weight=None, normalize=None
    259 ):
    260     """Compute confusion matrix to evaluate the accuracy of a␣
 ↪classification.
    261 
    262     By definition a confusion matrix :math:`C` is such that :math:`C_{i ␣
 ↪j}`
```

```
   (…)
    340     (0, 2, 1, 1)
    341     """
--> 342     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    343     if y_type not in ("binary", "multiclass"):
    344         raise ValueError("%s is not supported" % y_type)

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification
 ↪py:103, in _check_targets(y_true, y_pred)
     76 """Check that y_true and y_pred belong to the same classification task.
     77
     78 This converts multiclass or binary types to a common shape, and raises a
   (…)
    100 y_pred : array or indicator matrix
    101 """
    102 xp, _ = get_namespace(y_true, y_pred)
--> 103 check_consistent_length(y_true, y_pred)
    104 type_true = type_of_target(y_true, input_name="y_true")
    105 type_pred = type_of_target(y_pred, input_name="y_pred")

File c:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\validation.py:457 ␣
 ↪in check_consistent_length(*arrays)
    455 uniques = np.unique(lengths)
    456 if len(uniques) > 1:
--> 457     raise ValueError(
    458         "Found input variables with inconsistent numbers of samples: %r
    459         % [int(l) for l in lengths]
    460     )

ValueError: Found input variables with inconsistent numbers of samples:␣
 ↪[2520798, 504160]
```