#Data Preprocessing and Exploration

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import arff
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, roc_curve, confusion_matrix, classification_report
)
from keras.models import Model
from keras.layers import Input, Dense, Dropout
from keras.regularizers import l2
from keras.optimizers import Adam

# Load the .arff data
def load_arff_data(filename):
    data, meta = arff.loadarff(filename)
    df = pd.DataFrame(data)
    return df

# Load your training and test datasets
train_df = load_arff_data('ECG5000_TRAIN.arff')
test_df = load_arff_data('ECG5000_TEST.arff')

print(train_df.columns)
print('\n\n',train_df.describe)
print('\n\n',train_df.isna().sum())
print('\n\n',test_df.columns)
print('\n\n',test_df.describe)
print('\n\n',test_df.isna().sum())
```

```
Index(['att1', 'att2', 'att3', 'att4', 'att5', 'att6', 'att7', 'att8',
'att9',
       'att10',
       ...
       'att132', 'att133', 'att134', 'att135', 'att136', 'att137',
'att138',
       'att139', 'att140', 'target'],
      dtype='object', length=141)


 <bound method NDFrame.describe of           att1      att2      att3
att4      att5      att6      att7  \
0   -0.112522 -2.827204 -3.773897 -4.349751 -4.376041 -3.474986 -
2.181408
1   -1.100878 -3.996840 -4.285843 -4.506579 -4.022377 -3.234368 -
```

```
1.566126
2   -0.567088 -2.593450 -3.874230 -4.584095 -4.187449 -3.151462 -
1.742940
3    0.490473 -1.914407 -3.616364 -4.318823 -4.268016 -3.881110 -
2.993280
4    0.800232 -0.874252 -2.384761 -3.973292 -4.338224 -3.802422 -
2.534510
..        ...       ...       ...       ...       ...       ...
...
495 -0.478577 -1.779959 -2.398159 -3.170112 -3.559732 -3.573956 -
2.989770
496 -1.325210 -2.480992 -2.965356 -3.342392 -3.176351 -2.891528 -
2.369679
497 -0.021964 -0.912434 -1.903353 -2.662829 -3.122156 -3.451490 -
3.392982
498  0.288011 -1.098020 -2.500250 -3.598599 -3.650608 -3.281587 -
2.231601
499 -1.133674 -2.702941 -3.120979 -3.558669 -3.312442 -2.607641 -
1.354939

         att8      att9     att10  ...    att132    att133    
att134  \
0   -1.818286 -1.250522 -0.477492  ...  0.792168  0.933541  0.796958   

1   -0.992258 -0.754680  0.042321  ...  0.538356  0.656881  0.787490   

2   -1.490659 -1.183580 -0.394229  ...  0.886073  0.531452  0.311377   

3   -1.671131 -1.333884 -0.965629  ...  0.350816  0.499111  0.600345   

4   -1.783423 -1.594450 -0.753199  ...  1.148884  0.958434  1.059025   

..        ...       ...       ...  ...       ...       ...       ...   

495 -2.270605 -1.688277 -1.359872  ...  1.160885  1.456331  2.209421   

496 -1.598750 -1.071751 -0.891843  ... -0.172154 -0.864803 -1.549854   

497 -2.929937 -2.256294 -1.690706  ...  1.339479  1.457995  2.128078   

498 -1.250656 -1.072574 -0.434310  ... -0.029242  0.071414  0.118161   

499 -1.014740 -0.796023 -0.259599  ... -3.206942 -2.941677 -2.557140   


       att135    att136    att137    att138    att139    att140 
target  
0    0.578621  0.257740  0.228077  0.123431  0.925286  0.193137 
b'1'  
1    0.724046  0.555784  0.476333  0.773820  1.119621 -1.436250 
```

```
b'1'
2   -0.021919 -0.713683 -0.532197  0.321097  0.904227 -0.421797
b'1'
3    0.842069  0.952074  0.990133  1.086798  1.403011 -0.383564
b'1'
4    1.371682  1.277392  0.960304  0.971020  1.614392  1.421456
b'1'
..        ...       ...       ...       ...       ...       ...    ..
.
495  2.507175  2.198534  1.705849  1.492642  1.561890  1.520161
b'4'
496 -2.460243 -3.366562 -3.466546 -2.718380 -1.855209 -1.539958
b'4'
497  2.630759  2.295748  1.764967  1.444280  1.432347  1.457028
b'4'
498 -0.071967 -0.171214  0.131211  0.049872  0.010915 -0.081534
b'5'
499 -1.487946 -1.118880 -0.737113 -0.110840  0.001858 -0.122639
b'5'

[500 rows x 141 columns]>


 att1      0
att2      0
att3      0
att4      0
att5      0
         ..
att137    0
att138    0
att139    0
att140    0
target    0
Length: 141, dtype: int64


 Index(['att1', 'att2', 'att3', 'att4', 'att5', 'att6', 'att7',
'att8', 'att9',
       'att10',
       ...
       'att132', 'att133', 'att134', 'att135', 'att136', 'att137',
'att138',
       'att139', 'att140', 'target'],
      dtype='object', length=141)


 <bound method NDFrame.describe of            att1      att2      att3
att4      att5      att6      att7  \
0     3.690844  0.711414 -2.114091 -4.141007 -4.574472 -3.431909 -
```

```
1.950791
1    -1.348132 -3.996038 -4.226750 -4.251187 -3.477953 -2.228422 -
1.808488
2     1.024295 -0.590314 -1.916949 -2.806989 -3.527905 -3.638675 -
2.779767
3     0.545657 -1.014383 -2.316698 -3.634040 -4.196857 -3.758093 -
3.194444
4     0.661133 -1.552471 -3.124641 -4.313351 -4.017042 -3.005993 -
1.832411
...        ...       ...       ...       ...       ...       ...
...
4495 -1.122969 -2.252925 -2.867628 -3.358605 -3.167849 -2.638360 -
1.664162
4496 -0.547705 -1.889545 -2.839779 -3.457912 -3.929149 -3.966026 -
3.492560
4497 -1.351779 -2.209006 -2.520225 -3.061475 -3.065141 -3.030739 -
2.622720
4498 -1.124432 -1.905039 -2.192707 -2.904320 -2.900722 -2.761252 -
2.569705
4499  0.728813  0.192597 -0.733884 -1.779456 -2.345908 -2.977565 -
3.380053

          att8      att9     att10  ...    att132    att133    att134
\
0    -1.107067 -0.632322  0.334577  ...  0.022847  0.188937  0.480932

1    -1.534242 -0.779861 -0.397999  ...  1.570938  1.591394  1.549193

2    -2.019031 -1.980754 -1.440680  ...  0.443502  0.827582  1.237007

3    -2.221764 -1.588554 -1.202146  ...  0.777530  1.119240  0.902984

4    -1.503886 -1.071705 -0.521316  ...  1.280823  1.494315  1.618764

...        ...       ...       ...  ...       ...       ...       ...

4495 -0.935655 -0.866953 -0.645363  ... -0.472419 -1.310147 -2.029521

4496 -2.695270 -1.849691 -1.374321  ...  1.258419  1.907530  2.280888

4497 -2.044092 -1.295874 -0.733839  ... -1.512234 -2.076075 -2.586042

4498 -2.043893 -1.490538 -0.938473  ... -2.821782 -3.268355 -3.634981

4499 -3.417164 -3.030925 -2.313867  ...  1.267275  1.678989  2.483389


        att135    att136    att137    att138    att139    att140
target
0     0.629250  0.577291  0.665527  1.035997  1.492287 -1.905073
```

```
b'1'
1     1.193077  0.515134  0.126274  0.267532  1.071148 -1.164009
b'1'
2     1.235121  1.738103  1.800767  1.816301  1.473963  1.389767
b'1'
3     0.554098  0.497053  0.418116  0.703108  1.064602 -0.044853
b'1'
4     1.447449  1.238577  1.749692  1.986803  1.422756 -0.357784
b'1'
...        ...       ...       ...       ...       ...       ...      .
..
4495 -3.221294 -4.176790 -4.009720 -2.874136 -2.008369 -1.808334
b'4'
4496  1.895242  1.437702  1.193433  1.261335  1.150449  0.804932
b'2'
4497 -3.322799 -3.627311 -3.437038 -2.260023 -1.577823 -0.684531
b'2'
4498 -3.168765 -2.245878 -1.262260 -0.443307 -0.559769  0.108568
b'2'
4499  2.569073  2.122891  1.753963  1.538975  1.713781  1.309382
b'2'

[4500 rows x 141 columns]>


 att1      0
att2      0
att3      0
att4      0
att5      0
         ..
att137    0
att138    0
att139    0
att140    0
target    0
Length: 141, dtype: int64
```

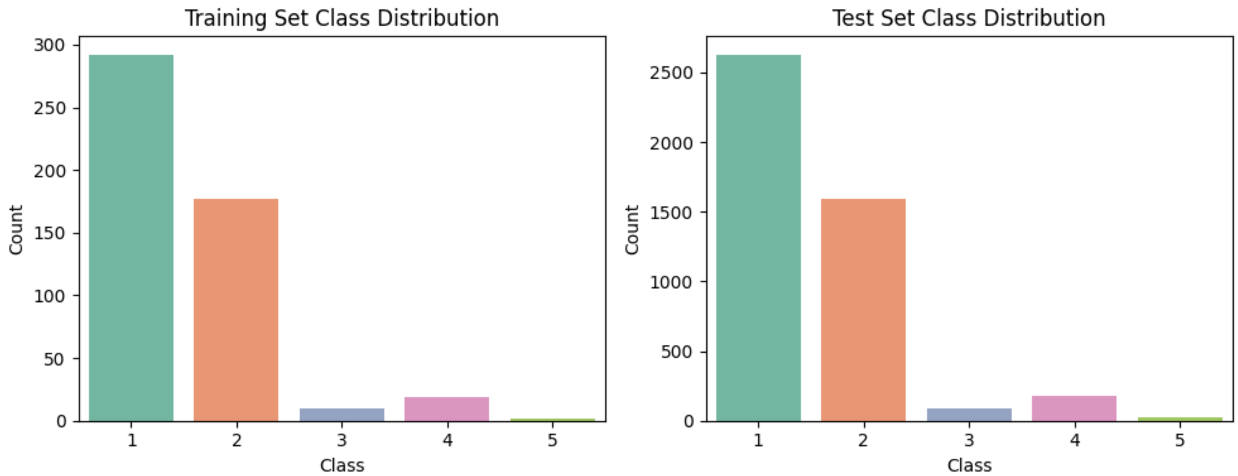
Class Distribution

```python
# Check unique classes and their counts in train and test
print("Class distribution in training set:")
print(train_df['target'].value_counts())

print("\nClass distribution in test set:")
print(test_df['target'].value_counts())

# Plot class distribution
```

```python
plt.figure(figsize=(10, 4))

# Training set
plt.subplot(1, 2, 1)
sns.countplot(x='target', data=train_df, palette='Set2')
plt.title('Training Set Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')

# Test set
plt.subplot(1, 2, 2)
sns.countplot(x='target', data=test_df, palette='Set2')
plt.title('Test Set Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')

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

```
Class distribution in training set:
target
b'1'    292
b'2'    177
b'4'     19
b'3'     10
b'5'      2
Name: count, dtype: int64

Class distribution in test set:
target
b'1'    2627
b'2'    1590
b'4'     175
b'3'      86
b'5'      22
Name: count, dtype: int64

<ipython-input-3-52419079b109>:13: 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.countplot(x='target', data=train_df, palette='Set2')
<ipython-input-3-52419079b109>:20: 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.countplot(x='target', data=test_df, palette='Set2')
```



Test 1

```
# Decode the target labels from bytes to regular strings (e.g., b'1' →
'1')
test_df['target'] = test_df['target'].apply(lambda x: x.decode('utf-
8'))

# Extract the test features by dropping the target column
X_test_raw = test_df.drop(columns=['target']).values

# Create binary labels: 0 for normal (class '1'), 1 for all other
classes (anomalies)
y_test = test_df['target'].apply(lambda x: 0 if x == '1' else
1).values

# Print the distribution of normal vs. anomalous samples in the test
set
print("Final y_test distribution:", np.unique(y_test,
return_counts=True))

Final y_test distribution: (array([0, 1]), array([2627, 1873]))
```

Autoencoder V1 (Base)

```
# Extract training features by removing the target column
X_train_raw = train_df.drop(columns=['target']).values

# Standardize the features: zero mean and unit variance
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train_raw)
X_test = scaler.transform(X_test_raw)

# Split a small portion of the training set for validation (10%)
X_train, X_val = train_test_split(X_train, test_size=0.1,
random_state=42)

def build_simple_autoencoder(input_dim):
    input_layer = Input(shape=(input_dim,))
    x = Dense(128, activation='relu')(input_layer)
    x = Dense(64, activation='relu')(x)
    x = Dense(32, activation='relu')(x)
    bottleneck = Dense(16, activation='relu')(x)
    x = Dense(32, activation='relu')(bottleneck)
    x = Dense(64, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    output_layer = Dense(input_dim, activation='linear')(x)
    autoencoder = Model(inputs=input_layer, outputs=output_layer)
    autoencoder.compile(optimizer=Adam(learning_rate=0.001),
loss='mse')
    return autoencoder

autoencoder = build_simple_autoencoder(X_train.shape[1])

history = autoencoder.fit(X_train, X_train, epochs=100, batch_size=32,
validation_data=(X_val, X_val), verbose=1)

Epoch 1/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 62ms/step - loss: 0.9940 - val_loss:
1.0284
Epoch 2/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 0.7516 - val_loss:
0.7580
Epoch 3/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.6087 - val_loss:
0.5939
Epoch 4/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.4574 - val_loss:
0.5169
Epoch 5/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.3445 - val_loss:
0.4823
Epoch 6/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 0.3702 - val_loss:
0.4607
Epoch 7/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.3191 - val_loss:
0.4467
Epoch 8/100
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.2963 - val_loss:
0.4233
Epoch 9/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.2715 - val_loss:
0.4006
Epoch 10/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 0.2482 - val_loss:
0.3751
Epoch 11/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 0.2287 - val_loss:
0.3600
Epoch 12/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.2191 - val_loss:
0.3480
Epoch 13/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.1886 - val_loss:
0.3466
Epoch 14/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.2088 - val_loss:
0.3429
Epoch 15/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1870 - val_loss:
0.3296
Epoch 16/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1789 - val_loss:
0.3232
Epoch 17/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1675 - val_loss:
0.3196
Epoch 18/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1796 - val_loss:
0.3238
Epoch 19/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1627 - val_loss:
0.3005
Epoch 20/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1491 - val_loss:
0.3084
Epoch 21/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1508 - val_loss:
0.3025
Epoch 22/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1562 - val_loss:
0.2984
Epoch 23/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1453 - val_loss:
0.3012
Epoch 24/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1444 - val_loss:
```

```
0.2883
Epoch 25/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.1321 - val_loss:
0.2954
Epoch 26/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1332 - val_loss:
0.2911
Epoch 27/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.1293 - val_loss:
0.2938
Epoch 28/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1362 - val_loss:
0.2813
Epoch 29/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1295 - val_loss:
0.2810
Epoch 30/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1271 - val_loss:
0.2777
Epoch 31/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1102 - val_loss:
0.2778
Epoch 32/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1274 - val_loss:
0.2692
Epoch 33/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1113 - val_loss:
0.2666
Epoch 34/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1087 - val_loss:
0.2871
Epoch 35/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1347 - val_loss:
0.2969
Epoch 36/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1276 - val_loss:
0.2707
Epoch 37/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1025 - val_loss:
0.2678
Epoch 38/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1051 - val_loss:
0.2598
Epoch 39/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1036 - val_loss:
0.2606
Epoch 40/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0993 - val_loss:
0.2656
```

```
Epoch 41/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1022 - val_loss:
0.2535
Epoch 42/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0946 - val_loss:
0.2568
Epoch 43/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1030 - val_loss:
0.2466
Epoch 44/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0928 - val_loss:
0.2530
Epoch 45/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0902 - val_loss:
0.2473
Epoch 46/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0957 - val_loss:
0.2470
Epoch 47/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0985 - val_loss:
0.2590
Epoch 48/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1104 - val_loss:
0.2735
Epoch 49/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1053 - val_loss:
0.2478
Epoch 50/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0921 - val_loss:
0.2397
Epoch 51/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0859 - val_loss:
0.2365
Epoch 52/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0921 - val_loss:
0.2408
Epoch 53/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0840 - val_loss:
0.2340
Epoch 54/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0797 - val_loss:
0.2446
Epoch 55/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0808 - val_loss:
0.2372
Epoch 56/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0773 - val_loss:
0.2395
Epoch 57/100
```

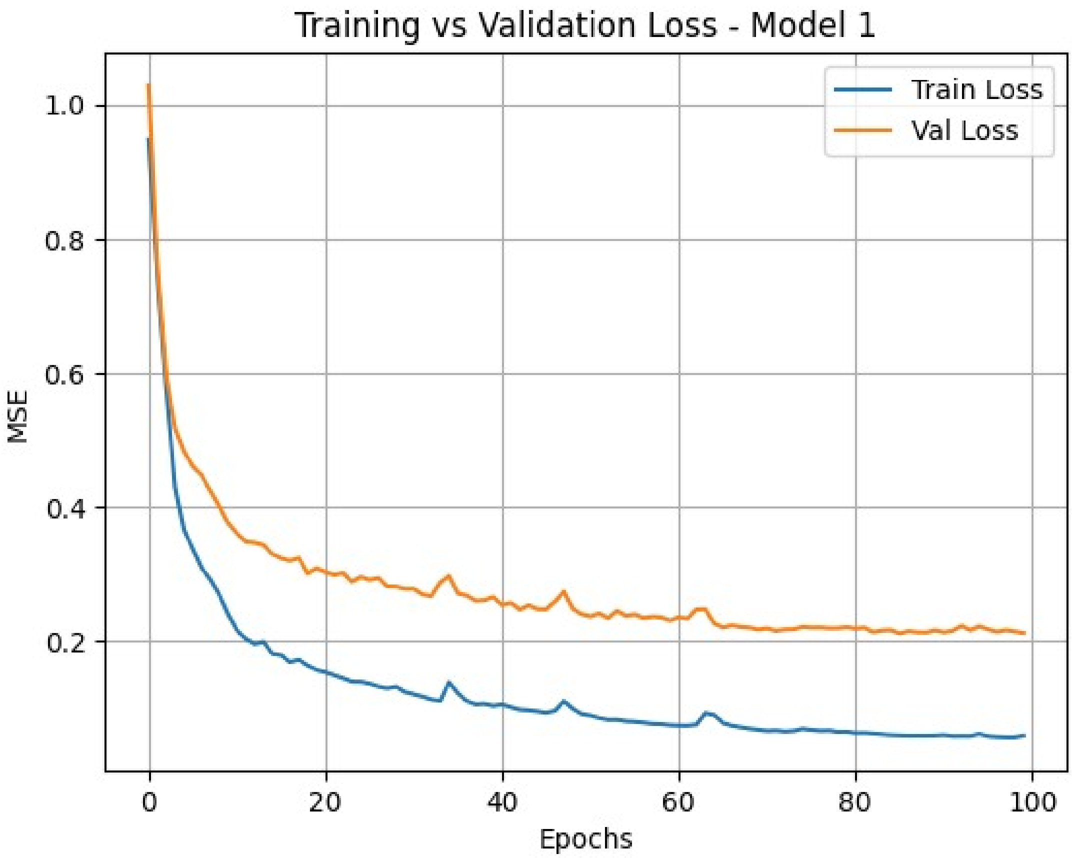
```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0778 - val_loss:
0.2341
Epoch 58/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0775 - val_loss:
0.2361
Epoch 59/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0757 - val_loss:
0.2352
Epoch 60/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0725 - val_loss:
0.2303
Epoch 61/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0753 - val_loss:
0.2352
Epoch 62/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0730 - val_loss:
0.2334
Epoch 63/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0729 - val_loss:
0.2467
Epoch 64/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0937 - val_loss:
0.2472
Epoch 65/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1023 - val_loss:
0.2259
Epoch 66/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0732 - val_loss:
0.2199
Epoch 67/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0735 - val_loss:
0.2232
Epoch 68/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0719 - val_loss:
0.2208
Epoch 69/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0693 - val_loss:
0.2197
Epoch 70/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0681 - val_loss:
0.2168
Epoch 71/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0654 - val_loss:
0.2187
Epoch 72/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0746 - val_loss:
0.2150
Epoch 73/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0636 - val_loss:
```

```
0.2170
Epoch 74/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0635 - val_loss:
0.2172
Epoch 75/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0713 - val_loss:
0.2209
Epoch 76/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0692 - val_loss:
0.2198
Epoch 77/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0683 - val_loss:
0.2201
Epoch 78/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0651 - val_loss:
0.2188
Epoch 79/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0631 - val_loss:
0.2189
Epoch 80/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0647 - val_loss:
0.2205
Epoch 81/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0646 - val_loss:
0.2183
Epoch 82/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0661 - val_loss:
0.2197
Epoch 83/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0640 - val_loss:
0.2131
Epoch 84/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0600 - val_loss:
0.2155
Epoch 85/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0612 - val_loss:
0.2162
Epoch 86/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0589 - val_loss:
0.2110
Epoch 87/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0605 - val_loss:
0.2145
Epoch 88/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0561 - val_loss:
0.2129
Epoch 89/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0568 - val_loss:
0.2125
```

```
Epoch 90/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0573 - val_loss:
0.2158
Epoch 91/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0591 - val_loss:
0.2129
Epoch 92/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0571 - val_loss:
0.2151
Epoch 93/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0574 - val_loss:
0.2224
Epoch 94/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0572 - val_loss:
0.2159
Epoch 95/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0588 - val_loss:
0.2218
Epoch 96/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0593 - val_loss:
0.2172
Epoch 97/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0556 - val_loss:
0.2136
Epoch 98/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0558 - val_loss:
0.2159
Epoch 99/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0561 - val_loss:
0.2141
Epoch 100/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0585 - val_loss:
0.2117
```

Loss Plot

```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Training vs Validation Loss - Model 1')
plt.xlabel('Epochs'); plt.ylabel('MSE'); plt.legend(); plt.grid(True)
plt.show()
```



Testing Evaluation

```
X_test_reconstructed = autoencoder.predict(X_test)
reconstruction_errors = np.mean(np.square(X_test -
X_test_reconstructed), axis=1)
X_train_reconstructed = autoencoder.predict(X_train)
train_errors = np.mean(np.square(X_train - X_train_reconstructed),
axis=1)
threshold = np.percentile(train_errors, 90)
print(f"Anomaly Threshold (90th percentile): {threshold:.4f}")
```

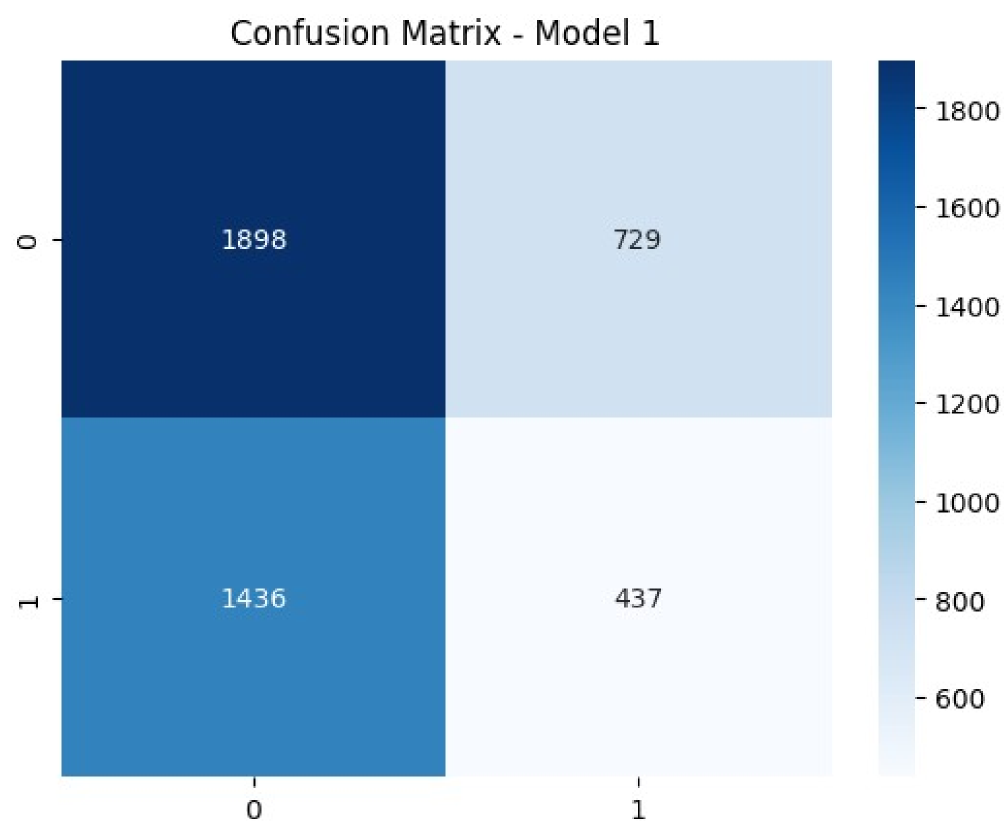
```
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Anomaly Threshold (90th percentile): 0.0898
```

```
y_pred = (reconstruction_errors > threshold).astype(int)
print("\n--- Evaluation Report (Model 1) ---")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred):.4f}")
print(f"Recall: {recall_score(y_test, y_pred):.4f}")
print(f"F1: {f1_score(y_test, y_pred):.4f}")
```

```python
print(f"ROC-AUC: {roc_auc_score(y_test, reconstruction_errors):.4f}")
print(classification_report(y_test, y_pred))
```

```
--- Evaluation Report (Model 1) ---
Accuracy: 0.5189
Precision: 0.3748
Recall: 0.2333
F1: 0.2876
ROC-AUC: 0.4230
              precision    recall  f1-score   support

           0       0.57      0.72      0.64      2627
           1       0.37      0.23      0.29      1873

    accuracy                           0.52      4500
   macro avg       0.47      0.48      0.46      4500
weighted avg       0.49      0.52      0.49      4500
```

```python
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True,
cmap='Blues', fmt='d')
plt.title("Confusion Matrix - Model 1")
plt.show()
```



Autoencoder V2 (Normal Only)

```
train_df['target'] = train_df['target'].apply(lambda x: x.decode('utf-
8'))
train_normal_df = train_df[train_df['target'] == '1']
X_train_raw = train_normal_df.drop(columns=['target']).values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train_raw)
X_test = scaler.transform(X_test_raw)
X_train, X_val = train_test_split(X_train, test_size=0.1,
random_state=42)

autoencoder = build_simple_autoencoder(X_train.shape[1])

history = autoencoder.fit(X_train, X_train, epochs=100, batch_size=32,
validation_data=(X_val, X_val), verbose=1)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - loss: 0.9442 - val_loss:
0.8206
Epoch 2/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.9301 - val_loss:
0.7340
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.8918 - val_loss:
0.6326
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.7002 - val_loss:
0.5255
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.5663 - val_loss:
0.4454
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.4245 - val_loss:
0.3768
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.3717 - val_loss:
0.3225
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3344 - val_loss:
0.2999
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.3136 - val_loss:
0.2960
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3160 - val_loss:
0.2706
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2695 - val_loss:
0.2650
```

```
Epoch 12/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2687 - val_loss:
0.2535
Epoch 13/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2658 - val_loss:
0.2562
Epoch 14/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2631 - val_loss:
0.2465
Epoch 15/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2634 - val_loss:
0.2483
Epoch 16/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2427 - val_loss:
0.2390
Epoch 17/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2380 - val_loss:
0.2345
Epoch 18/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2291 - val_loss:
0.2282
Epoch 19/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.2298 - val_loss:
0.2228
Epoch 20/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.2208 - val_loss:
0.2215
Epoch 21/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2081 - val_loss:
0.2148
Epoch 22/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.2090 - val_loss:
0.2114
Epoch 23/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.1976 - val_loss:
0.2061
Epoch 24/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.1964 - val_loss:
0.2008
Epoch 25/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.1868 - val_loss:
0.1957
Epoch 26/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.1766 - val_loss:
0.1956
Epoch 27/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.1784 - val_loss:
0.1892
Epoch 28/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1753 - val_loss:
0.1861
Epoch 29/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1847 - val_loss:
0.1900
Epoch 30/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1710 - val_loss:
0.1852
Epoch 31/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1591 - val_loss:
0.1820
Epoch 32/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1478 - val_loss:
0.1794
Epoch 33/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1660 - val_loss:
0.1796
Epoch 34/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1483 - val_loss:
0.1762
Epoch 35/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1539 - val_loss:
0.1755
Epoch 36/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1550 - val_loss:
0.1749
Epoch 37/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1439 - val_loss:
0.1741
Epoch 38/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1453 - val_loss:
0.1741
Epoch 39/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1521 - val_loss:
0.1744
Epoch 40/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1508 - val_loss:
0.1705
Epoch 41/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1435 - val_loss:
0.1731
Epoch 42/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1384 - val_loss:
0.1702
Epoch 43/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1321 - val_loss:
0.1690
Epoch 44/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1377 - val_loss:
```

```
0.1679
Epoch 45/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1416 - val_loss:
0.1662
Epoch 46/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1284 - val_loss:
0.1671
Epoch 47/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1286 - val_loss:
0.1673
Epoch 48/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1320 - val_loss:
0.1627
Epoch 49/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1313 - val_loss:
0.1690
Epoch 50/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1275 - val_loss:
0.1641
Epoch 51/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.1196 - val_loss:
0.1630
Epoch 52/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1333 - val_loss:
0.1661
Epoch 53/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1209 - val_loss:
0.1642
Epoch 54/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1195 - val_loss:
0.1677
Epoch 55/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1114 - val_loss:
0.1611
Epoch 56/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1206 - val_loss:
0.1623
Epoch 57/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1171 - val_loss:
0.1613
Epoch 58/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1180 - val_loss:
0.1648
Epoch 59/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1155 - val_loss:
0.1609
Epoch 60/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1246 - val_loss:
0.1625
```

```
Epoch 61/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1145 - val_loss:
0.1623
Epoch 62/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1311 - val_loss:
0.1657
Epoch 63/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1195 - val_loss:
0.1630
Epoch 64/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1125 - val_loss:
0.1538
Epoch 65/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.1125 - val_loss:
0.1589
Epoch 66/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1024 - val_loss:
0.1542
Epoch 67/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1085 - val_loss:
0.1530
Epoch 68/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0995 - val_loss:
0.1493
Epoch 69/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0986 - val_loss:
0.1465
Epoch 70/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0928 - val_loss:
0.1514
Epoch 71/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.1031 - val_loss:
0.1494
Epoch 72/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0992 - val_loss:
0.1499
Epoch 73/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0934 - val_loss:
0.1451
Epoch 74/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0968 - val_loss:
0.1490
Epoch 75/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0891 - val_loss:
0.1448
Epoch 76/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0902 - val_loss:
0.1453
Epoch 77/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0883 - val_loss:
0.1431
Epoch 78/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0943 - val_loss:
0.1453
Epoch 79/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0858 - val_loss:
0.1407
Epoch 80/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0872 - val_loss:
0.1416
Epoch 81/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0854 - val_loss:
0.1432
Epoch 82/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0873 - val_loss:
0.1417
Epoch 83/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0881 - val_loss:
0.1421
Epoch 84/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0844 - val_loss:
0.1436
Epoch 85/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0860 - val_loss:
0.1434
Epoch 86/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0824 - val_loss:
0.1405
Epoch 87/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0833 - val_loss:
0.1400
Epoch 88/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0858 - val_loss:
0.1403
Epoch 89/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0840 - val_loss:
0.1381
Epoch 90/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0853 - val_loss:
0.1402
Epoch 91/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0827 - val_loss:
0.1352
Epoch 92/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0845 - val_loss:
0.1365
Epoch 93/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0788 - val_loss:
```

```
0.1374
Epoch 94/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0760 - val_loss:
0.1365
Epoch 95/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0758 - val_loss:
0.1351
Epoch 96/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0842 - val_loss:
0.1357
Epoch 97/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0775 - val_loss:
0.1390
Epoch 98/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0790 - val_loss:
0.1354
Epoch 99/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0747 - val_loss:
0.1357
Epoch 100/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0770 - val_loss:
0.1367
```

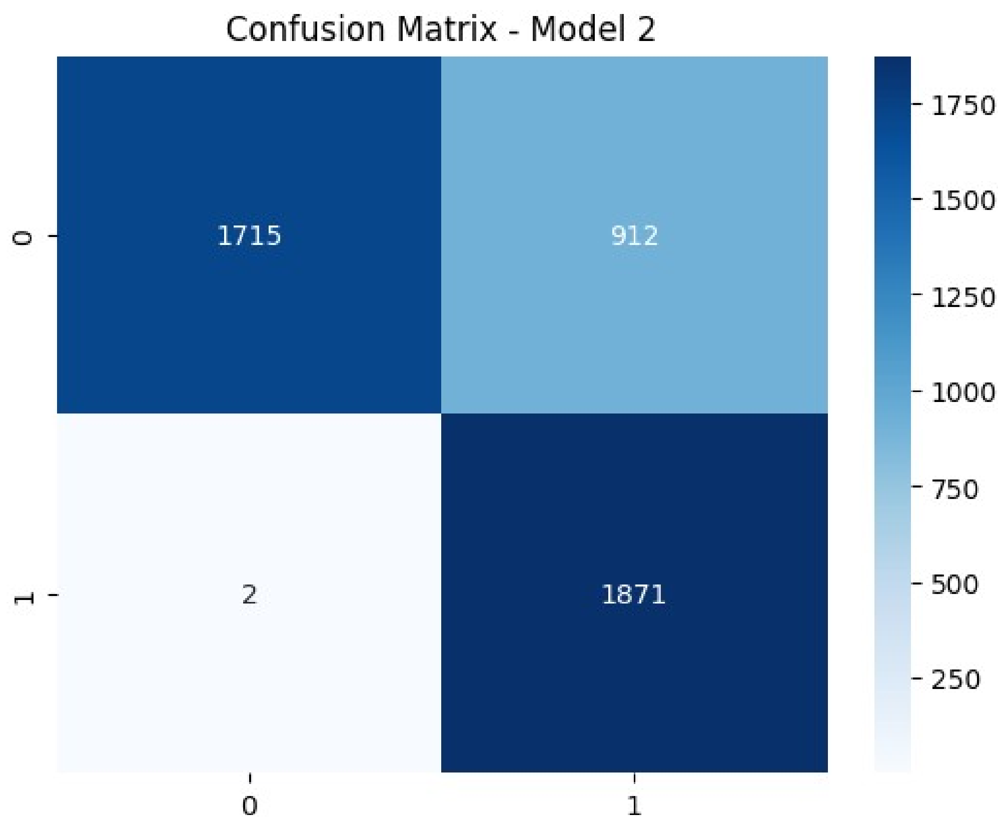
Test Evaluation (V2)

```
X_test_reconstructed = autoencoder.predict(X_test)
reconstruction_errors = np.mean(np.square(X_test -
X_test_reconstructed), axis=1)
X_train_reconstructed = autoencoder.predict(X_train)
train_errors = np.mean(np.square(X_train - X_train_reconstructed),
axis=1)
threshold = np.percentile(train_errors, 90)
print(f"Anomaly Threshold (90th percentile): {threshold:.4f}")
```

```
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Anomaly Threshold (90th percentile): 0.1178
```

```
y_pred = (reconstruction_errors > threshold).astype(int)
print("\n--- Evaluation Report (Model 2 - Normal Only) ---")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred):.4f}")
print(f"Recall: {recall_score(y_test, y_pred):.4f}")
print(f"F1: {f1_score(y_test, y_pred):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, reconstruction_errors):.4f}")
print(classification_report(y_test, y_pred))
```

```
--- Evaluation Report (Model 2 - Normal Only) ---
Accuracy: 0.7969
Precision: 0.6723
Recall: 0.9989
F1: 0.8037
ROC-AUC: 0.9857
              precision    recall  f1-score   support

           0       1.00      0.65      0.79      2627
           1       0.67      1.00      0.80      1873

    accuracy                           0.80      4500
   macro avg       0.84      0.83      0.80      4500
weighted avg       0.86      0.80      0.80      4500


sns.heatmap(confusion_matrix(y_test, y_pred), annot=True,
cmap='Blues', fmt='d')
plt.title("Confusion Matrix - Model 2")
plt.show()
```



Autoencoder V3 (Normal + L2 + Dropout)

```
def build_regularized_autoencoder(input_dim):
    input_layer = Input(shape=(input_dim,))
    x = Dense(128, activation='relu', kernel_regularizer=l2(0.001))
(input_layer)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu', kernel_regularizer=l2(0.001))(x)
    x = Dropout(0.2)(x)
    x = Dense(32, activation='relu')(x)
    bottleneck = Dense(16, activation='relu')(x)
    x = Dense(32, activation='relu')(bottleneck)
    x = Dense(64, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    output_layer = Dense(input_dim, activation='linear')(x)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

autoencoder = build_regularized_autoencoder(X_train.shape[1])

history = autoencoder.fit(X_train, X_train, epochs=100, batch_size=32,
validation_data=(X_val, X_val), verbose=1)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - loss: 1.1496 - val_loss:
1.0417
Epoch 2/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 1.1396 - val_loss:
1.0055
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.9782 - val_loss:
0.9322
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.9161 - val_loss:
0.8641
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.8680 - val_loss:
0.8090
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.7799 - val_loss:
0.7012
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6783 - val_loss:
0.6052
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.5900 - val_loss:
0.5086
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6086 - val_loss:
```

```
0.4670
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.4884 - val_loss:
0.4369
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.5079 - val_loss:
0.4186
Epoch 12/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4714 - val_loss:
0.4043
Epoch 13/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.4544 - val_loss:
0.3931
Epoch 14/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.4646 - val_loss:
0.3887
Epoch 15/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.4314 - val_loss:
0.3775
Epoch 16/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4887 - val_loss:
0.3723
Epoch 17/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.4711 - val_loss:
0.3674
Epoch 18/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4199 - val_loss:
0.3569
Epoch 19/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4114 - val_loss:
0.3458
Epoch 20/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4094 - val_loss:
0.3428
Epoch 21/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3951 - val_loss:
0.3491
Epoch 22/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4256 - val_loss:
0.3436
Epoch 23/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4129 - val_loss:
0.3395
Epoch 24/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.4272 - val_loss:
0.3287
Epoch 25/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3899 - val_loss:
0.3313
```

```
Epoch 26/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3822 - val_loss:
0.3344
Epoch 27/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3928 - val_loss:
0.3229
Epoch 28/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3635 - val_loss:
0.3191
Epoch 29/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3819 - val_loss:
0.3181
Epoch 30/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3607 - val_loss:
0.3202
Epoch 31/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3585 - val_loss:
0.3237
Epoch 32/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3686 - val_loss:
0.3187
Epoch 33/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.3411 - val_loss:
0.3093
Epoch 34/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3510 - val_loss:
0.3019
Epoch 35/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3390 - val_loss:
0.3131
Epoch 36/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3424 - val_loss:
0.3026
Epoch 37/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3629 - val_loss:
0.2983
Epoch 38/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.3157 - val_loss:
0.3049
Epoch 39/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.3185 - val_loss:
0.3087
Epoch 40/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.3144 - val_loss:
0.3084
Epoch 41/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.3467 - val_loss:
0.3049
Epoch 42/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.3108 - val_loss:
0.3000
Epoch 43/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.3148 - val_loss:
0.3024
Epoch 44/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.3204 - val_loss:
0.2968
Epoch 45/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.3370 - val_loss:
0.2969
Epoch 46/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.3054 - val_loss:
0.2940
Epoch 47/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3216 - val_loss:
0.2942
Epoch 48/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3219 - val_loss:
0.2839
Epoch 49/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3109 - val_loss:
0.2913
Epoch 50/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3080 - val_loss:
0.2828
Epoch 51/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.3147 - val_loss:
0.2939
Epoch 52/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3006 - val_loss:
0.2861
Epoch 53/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2814 - val_loss:
0.2843
Epoch 54/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3034 - val_loss:
0.2883
Epoch 55/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2929 - val_loss:
0.2779
Epoch 56/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.2833 - val_loss:
0.2857
Epoch 57/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.3014 - val_loss:
0.2786
Epoch 58/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.2822 - val_loss:
```

```
0.2767
Epoch 59/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.2773 - val_loss:
0.2863
Epoch 60/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2924 - val_loss:
0.2787
Epoch 61/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.2852 - val_loss:
0.2752
Epoch 62/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2702 - val_loss:
0.2692
Epoch 63/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2592 - val_loss:
0.2711
Epoch 64/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2670 - val_loss:
0.2664
Epoch 65/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2743 - val_loss:
0.2671
Epoch 66/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2702 - val_loss:
0.2661
Epoch 67/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2716 - val_loss:
0.2674
Epoch 68/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.2737 - val_loss:
0.2745
Epoch 69/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2967 - val_loss:
0.2793
Epoch 70/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2749 - val_loss:
0.2639
Epoch 71/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2552 - val_loss:
0.2638
Epoch 72/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 0.2577 - val_loss:
0.2657
Epoch 73/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.2658 - val_loss:
0.2709
Epoch 74/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.2678 - val_loss:
0.2618
```

```
Epoch 75/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - loss: 0.2816 - val_loss:
0.2689
Epoch 76/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - loss: 0.2666 - val_loss:
0.2618
Epoch 77/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.2580 - val_loss:
0.2627
Epoch 78/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.2622 - val_loss:
0.2600
Epoch 79/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 0.2639 - val_loss:
0.2531
Epoch 80/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - loss: 0.2613 - val_loss:
0.2548
Epoch 81/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.2704 - val_loss:
0.2652
Epoch 82/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.2516 - val_loss:
0.2486
Epoch 83/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.2519 - val_loss:
0.2556
Epoch 84/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.2428 - val_loss:
0.2513
Epoch 85/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.2462 - val_loss:
0.2573
Epoch 86/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.2390 - val_loss:
0.2446
Epoch 87/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.2650 - val_loss:
0.2569
Epoch 88/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.2331 - val_loss:
0.2543
Epoch 89/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - loss: 0.2603 - val_loss:
0.2603
Epoch 90/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.2608 - val_loss:
0.2493
Epoch 91/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.2414 - val_loss:
0.2531
Epoch 92/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.2463 - val_loss:
0.2587
Epoch 93/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.2702 - val_loss:
0.2448
Epoch 94/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2745 - val_loss:
0.2446
Epoch 95/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2586 - val_loss:
0.2476
Epoch 96/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2462 - val_loss:
0.2345
Epoch 97/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.2543 - val_loss:
0.2422
Epoch 98/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2648 - val_loss:
0.2376
Epoch 99/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.2475 - val_loss:
0.2461
Epoch 100/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.2538 - val_loss:
0.2970
```

```
X_test_reconstructed = autoencoder.predict(X_test)
reconstruction_errors = np.mean(np.square(X_test -
X_test_reconstructed), axis=1)
X_train_reconstructed = autoencoder.predict(X_train)
train_errors = np.mean(np.square(X_train - X_train_reconstructed),
axis=1)
```

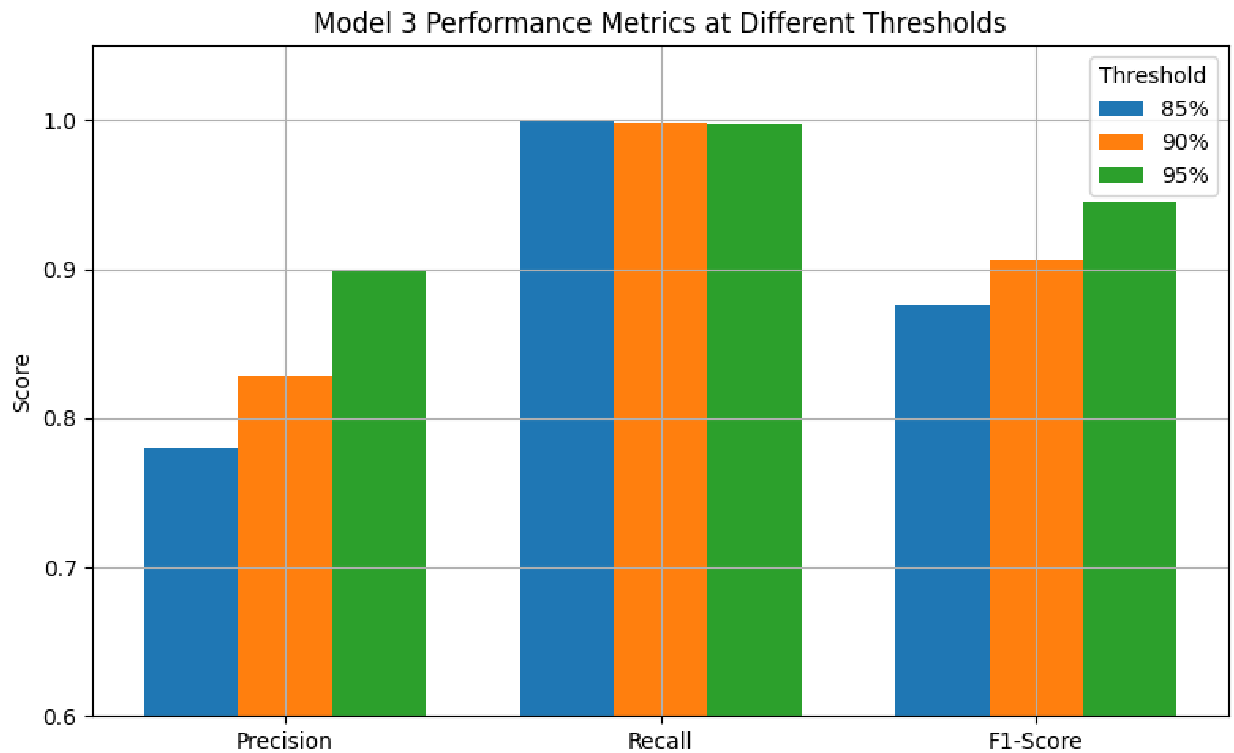
```
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

```
thresholds = [85, 90, 95]
for t in thresholds:
    threshold = np.percentile(train_errors, t)
    y_pred = (reconstruction_errors > threshold).astype(int)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

```python
    print(f"\n□ Threshold: {t}th percentile")
    print(f"Threshold value: {threshold:.4f}")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f} | Recall: {rec:.4f} | F1: {f1:.4f}")
```

```
□ Threshold: 85th percentile
Threshold value: 0.3146
Accuracy : 0.8822
Precision: 0.7799 | Recall: 0.9989 | F1: 0.8759

□ Threshold: 90th percentile
Threshold value: 0.3730
Accuracy : 0.9136
Precision: 0.8289 | Recall: 0.9984 | F1: 0.9058

□ Threshold: 95th percentile
Threshold value: 0.5298
Accuracy : 0.9522
Precision: 0.8993 | Recall: 0.9968 | F1: 0.9456
```

```python
# Scores per threshold
thresholds = ['85%', '90%', '95%']
precisions = [0.7799, 0.8289, 0.8993]
recalls = [0.9989, 0.9984, 0.9968]
f1_scores = [0.8759, 0.9058, 0.9456]

# Reorganize for metric-based grouping
metrics = ['Precision', 'Recall', 'F1-Score']
scores = [precisions, recalls, f1_scores]

x = np.arange(len(metrics))
width = 0.25

plt.figure(figsize=(8, 5))
plt.bar(x - width, [s[0] for s in scores], width, label='85%')
plt.bar(x,         [s[1] for s in scores], width, label='90%')
plt.bar(x + width, [s[2] for s in scores], width, label='95%')

plt.xticks(x, metrics)
plt.ylim(0.6, 1.05)
plt.ylabel('Score')
plt.title('Model 3 Performance Metrics at Different Thresholds')
plt.legend(title='Threshold')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Summary:

Project Summary:

This project applies unsupervised deep learning techniques to detect anomalies in ECG signals from the ECG5000 dataset. The approach is based on autoencoders, which are trained to learn and reconstruct patterns of normal heartbeats. The reconstruction error is used to identify anomalous signals that deviate from this learned normal pattern.

Models Developed:

1. Model 1 (Baseline): Trained on both normal and abnormal data.
   - Weak anomaly separation due to learning to reconstruct everything.
2. Model 2 (Clean): Trained only on class '1' (normal data).
   - Significantly improved performance by focusing on normal patterns only.
3. Model 3 (Optimized): Trained on normal data with Dropout and L2 regularization.
   - Achieved the best results, with high precision, recall, and F1-score.

Evaluation Metrics:

- Reconstruction error used to identify anomalies.
- Thresholds (85%, 90%, 95%) tested to control sensitivity.

- Metrics: Precision, Recall, F1-Score, Accuracy, ROC-AUC.

Why Normal-Only Training Works:

By training exclusively on normal data, the model becomes highly sensitive to anything unfamiliar (anomalies), resulting in higher reconstruction error. This improves anomaly detection accuracy and ensures better separation between normal and abnormal signals.

Conclusion:

Autoencoders are highly effective for anomaly detection when trained only on clean data. Proper threshold tuning and regularization significantly improve detection performance.