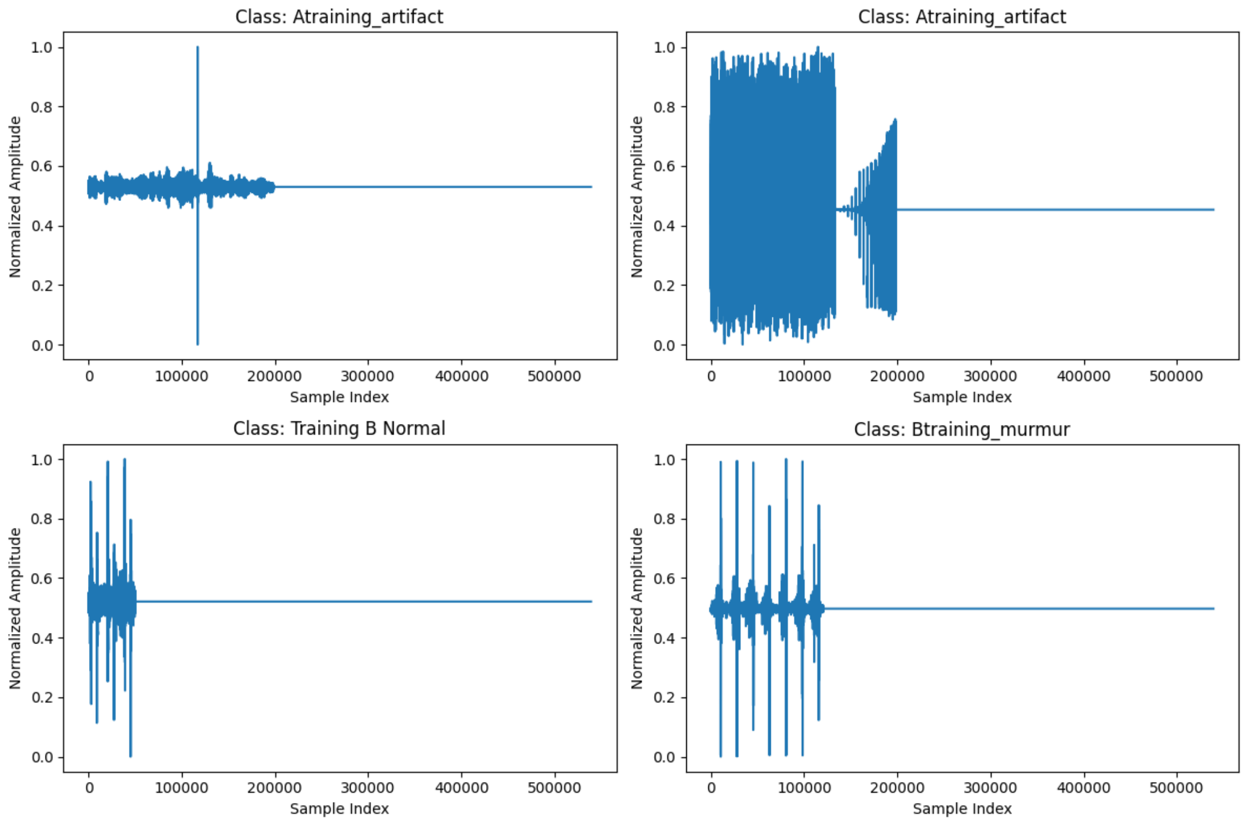
Biomedicial Callsifier

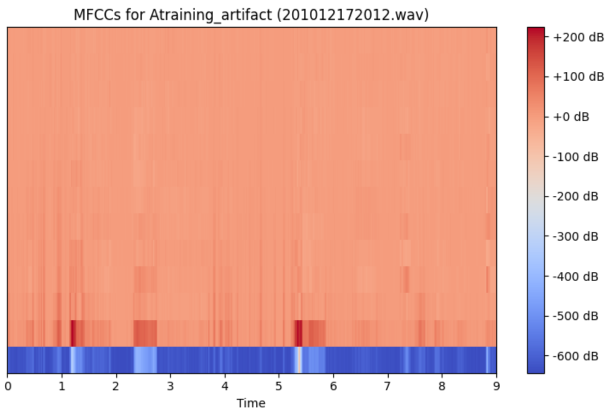
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import librosa
import os

DATASET_PATH = r"C:\Users\suman\Downloads\ai suman\Dataset Part 1\
Heart_sount_dataset"

data = []
labels = []
sr = 22050

for dataset_name in os.listdir(DATASET_PATH):
    dataset_dir = os.path.join(DATASET_PATH, dataset_name)
    if os.path.isdir(dataset_dir):
        for class_name in os.listdir(dataset_dir):
            class_dir = os.path.join(dataset_dir, class_name)
            if os.path.isdir(class_dir):
                for fname in os.listdir(class_dir):
                    if fname.lower().endswith(".wav"):
                        fpath = os.path.join(class_dir, fname)
                        try:
                            signal, file_sr = librosa.load(fpath,
sr=sr)
                            data.append(signal)
                            labels.append(class_name)
                        except Exception as e:
                            print(f"Error loading {fpath}: {e}")

print(f"Total audio samples loaded: {len(data)}")

if len(data) == 0:
    print("No audio data found. Please check dataset path and
structure.")
else:
    max_len = max(len(s) for s in data)
    padded_data = np.array([np.pad(s, (0, max_len - len(s)),
'constant') for s in data])

    df = pd.DataFrame(padded_data)
    df['label'] = labels
    scaler = MinMaxScaler()
    signal_data = df.drop('label', axis=1).values
    normalized_signal_data = scaler.fit_transform(signal_data.T).T
    df_normalized = pd.DataFrame(normalized_signal_data,
columns=df.columns[:-1])
    df_normalized['label'] = labels

    fig, axes = plt.subplots(2, 2, figsize=(12, 8))
    axes = axes.flatten()

    for i in range(4):
        row_index = np.random.randint(len(df_normalized))
        signal = df_normalized.iloc[row_index, :-1].values
        label = df_normalized.iloc[row_index, -1]
        axes[i].plot(signal)
        axes[i].set_title(f'Class: {label}')
        axes[i].set_xlabel('Sample Index')
        axes[i].set_ylabel('Normalized Amplitude')

    plt.tight_layout()
    plt.show()
    print(df_normalized.head())

Total audio samples loaded: 683
```

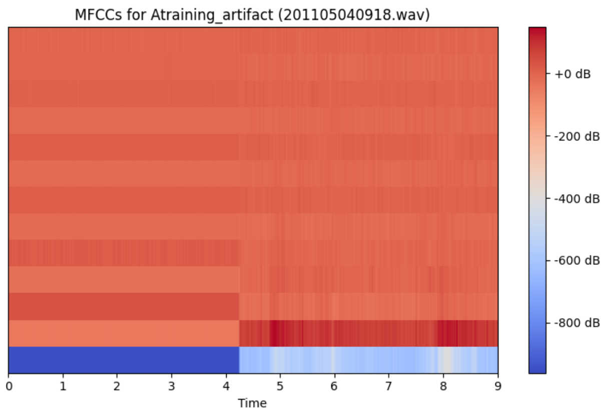


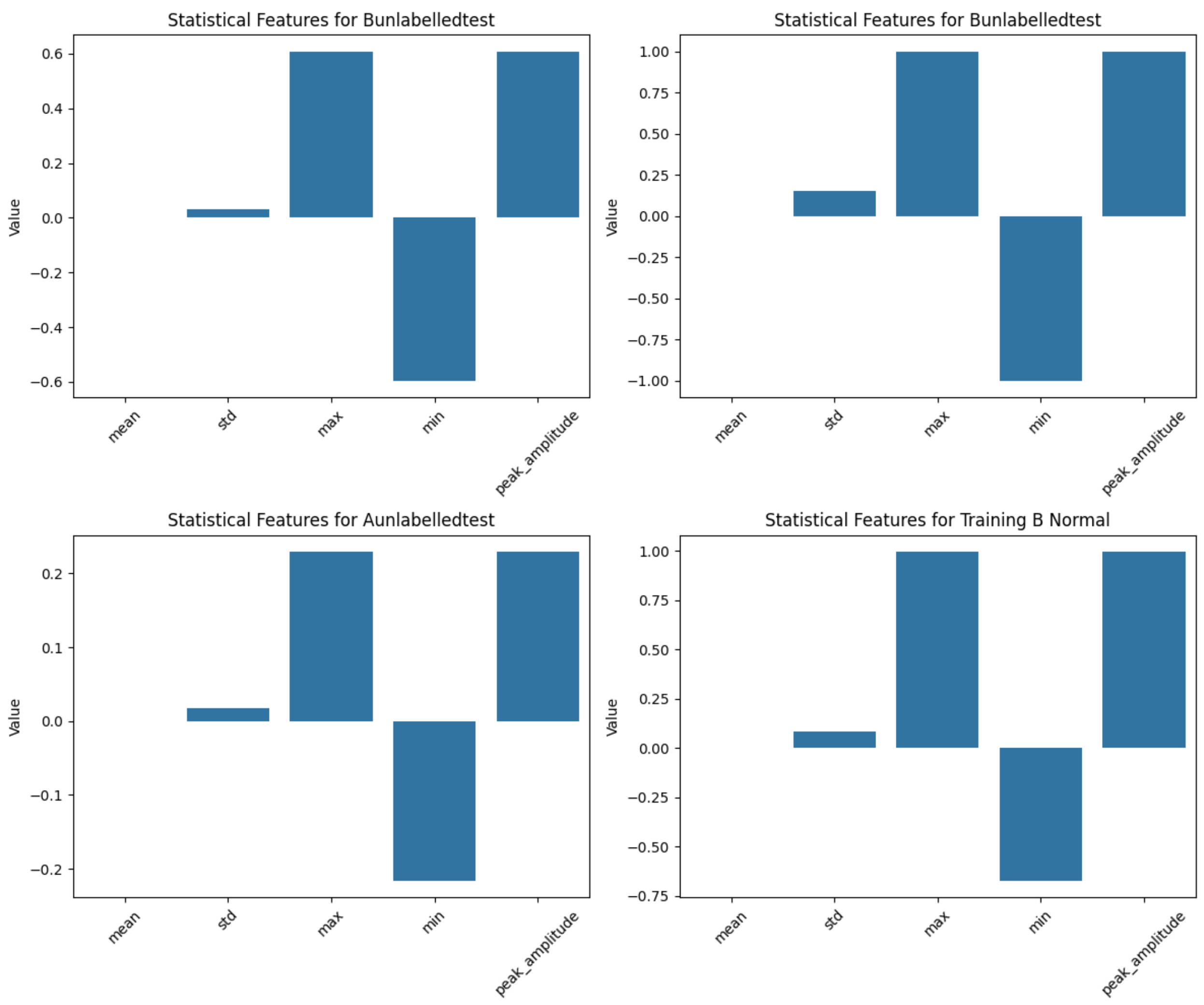
```
          0          1          2          3          4          5
6  \
0  0.506175  0.506066  0.506034  0.506041  0.506125  0.506224
0.506226
1  0.521508  0.521879  0.521900  0.521785  0.521716  0.521950
0.521606
2  0.513764  0.508600  0.510990  0.511940  0.514677  0.516584
0.515283
3  0.547782  0.545484  0.546443  0.544580  0.546316  0.550238
0.550870
4  0.494319  0.495040  0.499570  0.503376  0.504970  0.501841
0.500621

          7         8         9  ...    539086    539087    539088
539089  \
0  0.506146  0.506036  0.506028  ...  0.506158  0.506158  0.506158
0.506158
1  0.522069  0.521860  0.521535  ...  0.521842  0.521842  0.521842
0.521842
2  0.512572  0.513687  0.513675  ...  0.512795  0.512795  0.512795
0.512795
3  0.552719  0.553827  0.554481  ...  0.549421  0.549421  0.549421
0.549421
4  0.500215  0.500507  0.500411  ...  0.494241  0.494241  0.494241
0.494241

     539090    539091    539092    539093    539094
label
0  0.506158  0.506158  0.506158  0.506158  0.506158
Atraining_artifact
1  0.521842  0.521842  0.521842  0.521842  0.521842
Atraining_artifact
2  0.512795  0.512795  0.512795  0.512795  0.512795
Atraining_artifact
3  0.549421  0.549421  0.549421  0.549421  0.549421
Atraining_artifact
4  0.494241  0.494241  0.494241  0.494241  0.494241
Atraining_artifact

[5 rows x 539096 columns]
```

Feature extraction

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import librosa
import librosa.display
import soundfile as sf
import os
import warnings
from IPython.display import display

warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)

dataset_path = r"C:\Users\suman\Downloads\ai suman\Dataset Part 1\
Heart_sount_dataset"

def load_audio(file_path):
    try:
        if not os.path.exists(file_path):
            return None, None
        signal, sr = sf.read(file_path, dtype="float32")
        if len(signal.shape) > 1:
            signal = np.mean(signal, axis=1)
    except:
        try:
            signal, sr = librosa.load(file_path, sr=None, mono=True)
        except:
            return None, None
    return signal, sr

def extract_features(signal, sr=22050):
    mfccs = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13,
hop_length=512)
    features = {
        'mfccs_mean': np.mean(mfccs, axis=1),
        'mfccs_std': np.std(mfccs, axis=1),
        'mean': np.mean(signal),
        'std': np.std(signal),
        'max': np.max(signal),
        'min': np.min(signal),
        'peak_amplitude': np.max(np.abs(signal))
    }
    return features

extracted_features_list = []
valid_files = []

for root, dirs, files in os.walk(dataset_path):
    for file in files:
        if file.endswith(".wav"):
            file_path = os.path.join(root, file)
            signal, sr = load_audio(file_path)
            if signal is None or len(signal) == 0:
                continue
            label = os.path.basename(root)
            features = extract_features(signal, sr=sr)
            mfccs_mean_flat = features['mfccs_mean'].flatten()
            mfccs_std_flat = features['mfccs_std'].flatten()
            feature_row = {f'mfccs_mean_{i}': mfccs_mean_flat[i] for i
in range(len(mfccs_mean_flat))}
            feature_row.update({f'mfccs_std_{i}': mfccs_std_flat[i]
for i in range(len(mfccs_std_flat))})
            feature_row.update({
                'mean': features['mean'],
                'std': features['std'],
                'max': features['max'],
                'min': features['min'],
                'peak_amplitude': features['peak_amplitude'],
                'label': label,
                'file': file
            })
            extracted_features_list.append(feature_row)
            valid_files.append((file_path, label, file))

df_features = pd.DataFrame(extracted_features_list)

if len(valid_files) >= 2:
    fig, axes = plt.subplots(1, 2, figsize=(15, 5))
    for i in range(2):
        file_path, label, file_name = valid_files[i]
        signal, sr = load_audio(file_path)
        if signal is None or len(signal) == 0:
            continue
        mfccs = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13,
hop_length=512)
        if mfccs.shape[1] == 0:
            continue
        img = librosa.display.specshow(mfccs, sr=sr, x_axis='time',
ax=axes[i])
        axes[i].set_title(f'MFCCs for {label} ({file_name})')
        fig.colorbar(img, ax=axes[i], format='%+2.0f dB')
    plt.tight_layout()
    plt.show()

if len(df_features) > 0:
    fig, axes = plt.subplots(2, 2, figsize=(12, 10))
    axes = axes.flatten()
    statistical_features = ['mean', 'std', 'max', 'min',
'peak_amplitude']
    for i in range(4):
        row_index = np.random.randint(len(df_features))
        feature_values = df_features.iloc[row_index]
[statistical_features].values
        label = df_features.iloc[row_index]['label']
        sns.barplot(x=statistical_features, y=feature_values,
ax=axes[i])
        axes[i].set_title(f'Statistical Features for {label}')
        axes[i].set_ylabel('Value')
        axes[i].tick_params(axis='x', rotation=45)
    plt.tight_layout()
    plt.show()

display(df_features.head())
```







```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"mfccs_mean_0","rawType":"float32","type":"float"},
{"name":"mfccs_mean_1","rawType":"float32","type":"float"},
{"name":"mfccs_mean_2","rawType":"float32","type":"float"},
{"name":"mfccs_mean_3","rawType":"float32","type":"float"},
{"name":"mfccs_mean_4","rawType":"float32","type":"float"},
{"name":"mfccs_mean_5","rawType":"float32","type":"float"},
{"name":"mfccs_mean_6","rawType":"float32","type":"float"},
{"name":"mfccs_mean_7","rawType":"float32","type":"float"},
{"name":"mfccs_mean_8","rawType":"float32","type":"float"},
{"name":"mfccs_mean_9","rawType":"float32","type":"float"},
{"name":"mfccs_mean_10","rawType":"float32","type":"float"},
{"name":"mfccs_mean_11","rawType":"float32","type":"float"},
{"name":"mfccs_mean_12","rawType":"float32","type":"float"},
{"name":"mfccs_std_0","rawType":"float32","type":"float"},
{"name":"mfccs_std_1","rawType":"float32","type":"float"},
{"name":"mfccs_std_2","rawType":"float32","type":"float"},
{"name":"mfccs_std_3","rawType":"float32","type":"float"},
{"name":"mfccs_std_4","rawType":"float32","type":"float"},
{"name":"mfccs_std_5","rawType":"float32","type":"float"},
{"name":"mfccs_std_6","rawType":"float32","type":"float"},
{"name":"mfccs_std_7","rawType":"float32","type":"float"},
{"name":"mfccs_std_8","rawType":"float32","type":"float"},
{"name":"mfccs_std_9","rawType":"float32","type":"float"},
{"name":"mfccs_std_10","rawType":"float32","type":"float"},
{"name":"mfccs_std_11","rawType":"float32","type":"float"},
{"name":"mfccs_std_12","rawType":"float32","type":"float"},
{"name":"mean","rawType":"float32","type":"float"},
{"name":"std","rawType":"float32","type":"float"},
{"name":"max","rawType":"float32","type":"float"},
{"name":"min","rawType":"float32","type":"float"},
{"name":"peak_amplitude","rawType":"float32","type":"float"},
{"name":"label","rawType":"object","type":"string"},
{"name":"file","rawType":"object","type":"string"}],"ref":"a4601573-
6251-48dc-b8b2-e28e2b8238db","rows":[["0","-
599.93896","33.781063","7.35434","4.5494447","2.182476","2.7137213","0
.70974946","-0.5813915","-1.2105308","-2.7702992","-1.950883","-
1.373695","-
1.6228255","63.559986","40.340088","13.842911","12.959423","9.30077","
7.30098","5.678077","5.4349117","4.6704197","4.63749","3.8292885","3.3
45399","3.240959","-1.0466322e-05","0.014274816","0.70410156","-
0.7289734","0.7289734","Atraining_artifact","201012172012.wav"],
["1","-752.226","23.831118","8.511935","-12.289284","6.590361","-
13.384438","8.085058","-10.728707","8.7330475","-
9.553118","3.778762","-4.386584","-
2.2409499","193.43013","67.308914","27.99302","15.005083","11.636027",
"4.58469","5.4975467","4.2518296","5.7577662","2.9655578","3.7120292",
"4.586085","3.1971843","-1.1055222e-
06","0.00084005133","0.016021729","-
0.018188477","0.018188477","Atraining_artifact","201105040918.wav"],
["2","-587.1053","56.313488","-21.25168","5.092139","-13.107715","-
3.1940339","-3.2685952","-9.355012","3.2398045","-
8.664276","1.9791583","-5.9559555","-
1.5866339","78.01337","21.039032","9.313816","12.808322","11.165461","
10.16053","6.3035564","4.477588","5.2280607","4.9290957","4.7142653","
4.2985764","4.010035","4.744103e-07","0.0010501887","0.020721436","-
0.027069092","0.027069092","Atraining_artifact","201105041959.wav"],
["3","-349.67032","141.82634","-21.67626","23.579344","-2.894292","-
3.9263988","-8.095814","-8.885119","-7.6928396","-5.4134545","-
5.656429","-9.26078","-
6.8140674","65.990234","31.441687","24.155802","12.032792","11.656161"
,"9.121105","7.678339","7.4909124","6.634921","6.0082426","6.1362395",
"6.3897085","5.5943704","-1.3695388e-05","0.017936688","0.88171387","-
0.90515137","0.90515137","Atraining_artifact","201105051017.wav"],
["4","-333.93628","115.90495","-7.5759363","5.6850233","1.2802495","-
1.1377603","2.2690852","-7.8378334","-2.7766035","-10.6930485","-
0.80935055","-7.148301","-
```

5.4792757","85.54612","22.55842","16.148848","8.672056","9.865658","7.1070833","6.7316422","7.4879208","6.4449434","6.3067145","6.4818907","6.2358828","7.083377","-2.8633624e-05","0.038326383","0.886261","-0.94033813","0.94033813","Atraining_artifact","201105060108.wav"]],"shape":{"columns":33,"rows":5}}

Data splitting

```
from sklearn.model_selection import train_test_split

X = df_features.drop('label', axis=1)
y = df_features['label']
X_train, X_temp, y_train, y_temp = train_test_split(X, y, 
test_size=0.3, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, 
test_size=0.5, random_state=42, stratify=y_temp)

print("Training set shape (X_train, y_train):", X_train.shape, 
y_train.shape)
print("Validation set shape (X_val, y_val):", X_val.shape, 
y_val.shape)
print("Test set shape (X_test, y_test):", X_test.shape, y_test.shape)
```

```
Training set shape (X_train, y_train): (582, 32) (582,)
Validation set shape (X_val, y_val): (125, 32) (125,)
Test set shape (X_test, y_test): (125, 32) (125,)
```

Model building

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

logistic_regression_model = LogisticRegression(random_state=42)
random_forest_model = RandomForestClassifier(random_state=42)
svm_model = SVC(random_state=42)

dnn_model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

dnn_model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

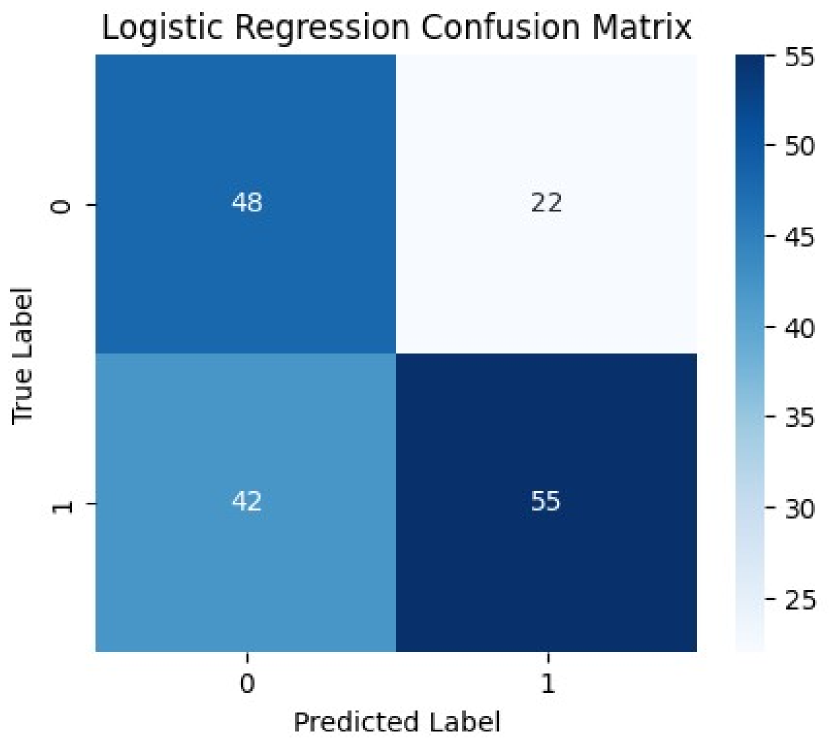
```python
print("Logistic Regression Model:")
print(logistic_regression_model)
print("\nRandom Forest Model:")
print(random_forest_model)
print("\nSVM Model:")
print(svm_model)
print("\nDeep Neural Network Model Summary:")
dnn_model.summary()
```

```
Logistic Regression Model:
LogisticRegression(random_state=42)

Random Forest Model:
RandomForestClassifier(random_state=42)

SVM Model:
SVC(random_state=42)

Deep Neural Network Model Summary:

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ dense (Dense)                   │ (None, 64)             │
2,112 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dense_1 (Dense)                 │ (None, 32)             │
2,080 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dense_2 (Dense)                 │ (None, 1)              │
33 │
└─────────────────────────────────┴────────────────────────┴──────────
─────┘

 Total params: 4,225 (16.50 KB)

 Trainable params: 4,225 (16.50 KB)

 Non-trainable params: 0 (0.00 B)
```

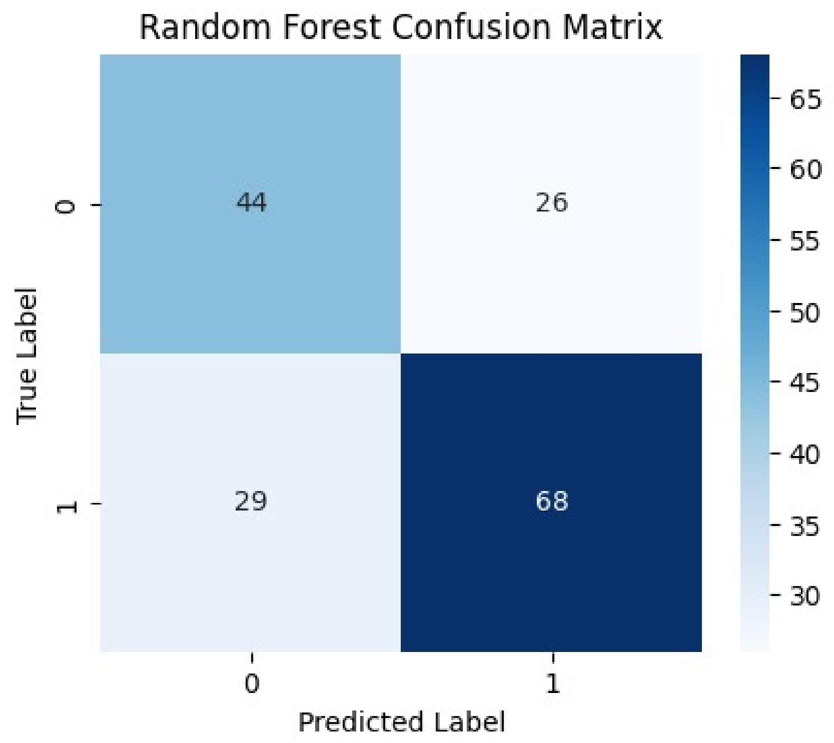
Model training and evaluation

```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

models = {
    "Logistic Regression": LogisticRegression(max_iter=2000,
class_weight="balanced"),
    "Random Forest": RandomForestClassifier(n_estimators=300,
max_depth=20, class_weight="balanced"),
    "SVM (RBF)": SVC(kernel="rbf", C=10, gamma='scale',
probability=True, class_weight="balanced")
}

for name, model in models.items():
    model.fit(X_train_res, y_train_res)
    y_pred = model.predict(X_test)

    print(f"\n{name} Results")
    print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
    print("\nClassification Report:\n", classification_report(y_test,
y_pred, zero_division=0))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} Confusion Matrix")
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```

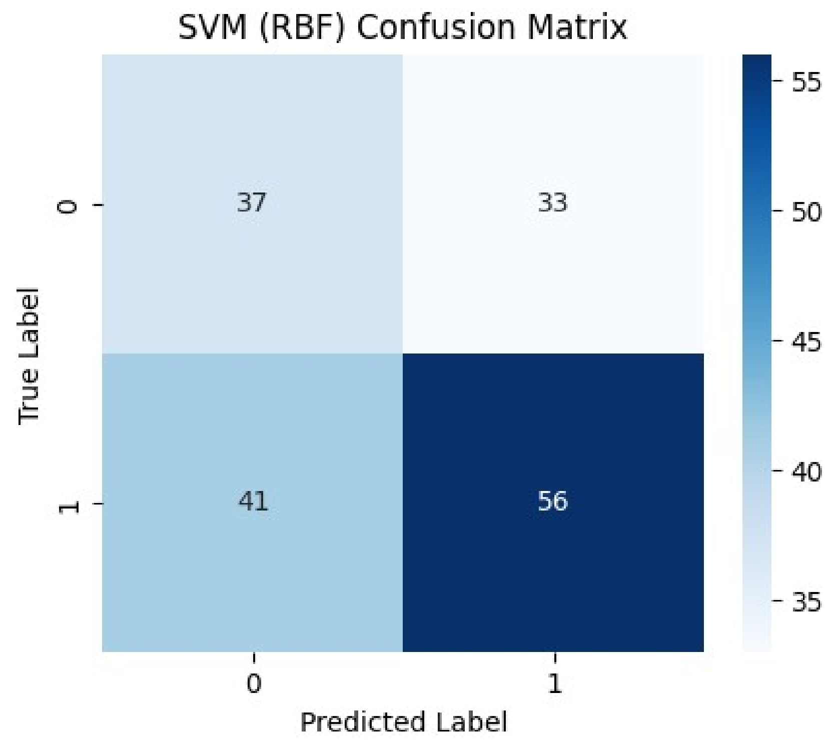
```
Logistic Regression Results
Accuracy: 0.6168

Classification Report:
               precision    recall  f1-score   support

           0       0.53      0.69      0.60        70
           1       0.71      0.57      0.63        97

    accuracy                           0.62       167
   macro avg       0.62      0.63      0.62       167
weighted avg       0.64      0.62      0.62       167
```



```
Random Forest Results
Accuracy: 0.6707

Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.63      0.62        70
           1       0.72      0.70      0.71        97

    accuracy                           0.67       167
   macro avg       0.66      0.66      0.66       167
weighted avg       0.67      0.67      0.67       167
```



```
SVM (RBF) Results
Accuracy: 0.5569

Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.53      0.50        70
           1       0.63      0.58      0.60        97

    accuracy                           0.56       167
   macro avg       0.55      0.55      0.55       167
weighted avg       0.56      0.56      0.56       167
```

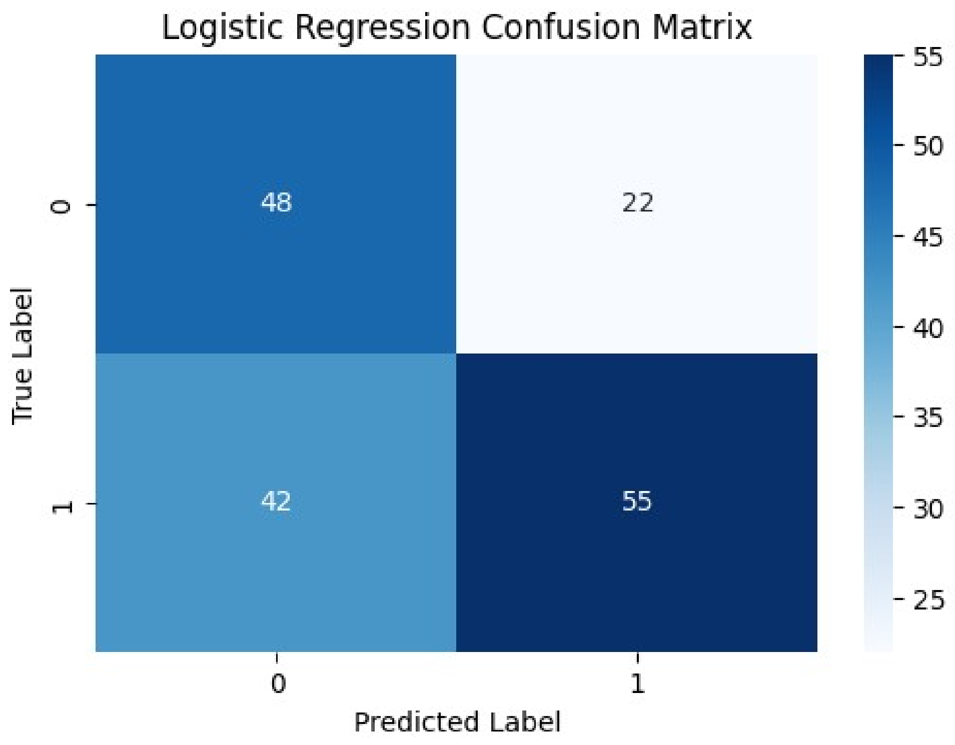


Model Validation

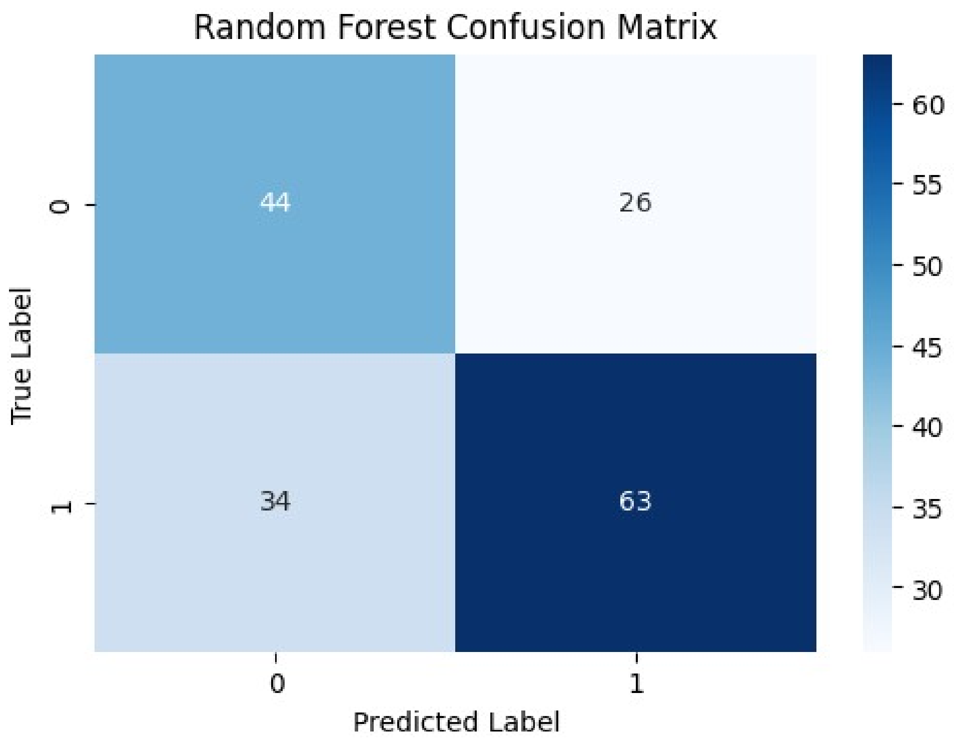
```python
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, classification_report, confusion_matrix,
roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
logistic_regression_model = LogisticRegression(max_iter=2000,
class_weight="balanced")
logistic_regression_model.fit(X_train, y_train)

random_forest_model = RandomForestClassifier(n_estimators=300,
max_depth=20, class_weight="balanced")
random_forest_model.fit(X_train, y_train)

svm_model = SVC(kernel="rbf", C=10, gamma='scale', probability=True,
class_weight="balanced")
svm_model.fit(X_train, y_train)

models = {
    "Logistic Regression": logistic_regression_model,
    "Random Forest": random_forest_model,
    "SVM": svm_model
}
```

```
for name, model in models.items():
    print(f"\n {name} Evaluation")

    y_pred = model.predict(X_test)
    print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
    print("Precision:", round(precision_score(y_test, y_pred,
average='weighted', zero_division=0), 4))
    print("Recall:", round(recall_score(y_test, y_pred,
average='weighted', zero_division=0), 4))
    print("F1-score:", round(f1_score(y_test, y_pred,
average='weighted', zero_division=0), 4))
    print("\nClassification Report:\n", classification_report(y_test,
y_pred, zero_division=0))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} Confusion Matrix")
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)
        if len(np.unique(y_test)) == 2:
            auc = roc_auc_score(y_test, y_score[:,1])
        else:
            auc = roc_auc_score(pd.get_dummies(y_test), y_score,
multi_class='ovr', average='weighted')
        print("ROC-AUC:", round(auc, 4))
```

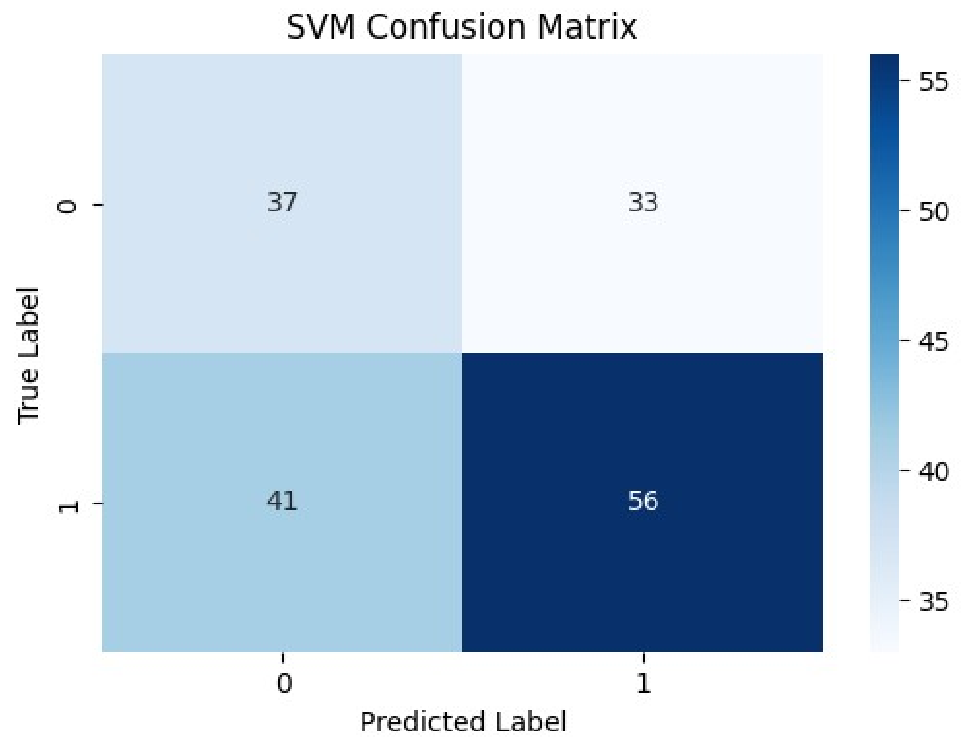
```
 Logistic Regression Evaluation
Accuracy: 0.6168
Precision: 0.6384
Recall: 0.6168
F1-score: 0.6187

Classification Report:
               precision    recall  f1-score   support

           0       0.53      0.69      0.60        70
           1       0.71      0.57      0.63        97

    accuracy                           0.62       167
   macro avg       0.62      0.63      0.62       167
weighted avg       0.64      0.62      0.62       167
```



```
ROC-AUC: 0.6736

 Random Forest Evaluation
Accuracy: 0.6407
Precision: 0.6476
Recall: 0.6407
F1-score: 0.6427

Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.63      0.59        70
           1       0.71      0.65      0.68        97

    accuracy                           0.64       167
   macro avg       0.64      0.64      0.64       167
weighted avg       0.65      0.64      0.64       167
```



```
ROC-AUC: 0.7041

 SVM Evaluation
Accuracy: 0.5569
Precision: 0.5643
Recall: 0.5569
F1-score: 0.5593

Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.53      0.50        70
           1       0.63      0.58      0.60        97

    accuracy                           0.56       167
   macro avg       0.55      0.55      0.55       167
weighted avg       0.56      0.56      0.56       167
```

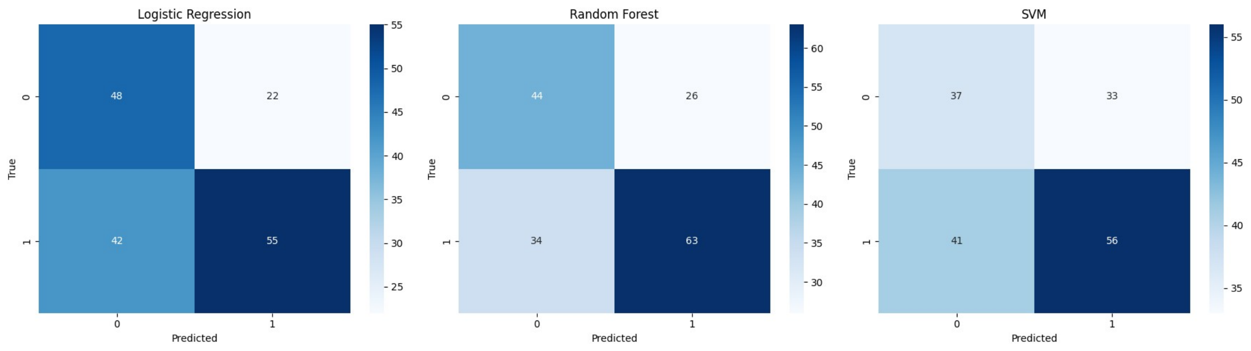


```
ROC-AUC: 0.6358
```

Confusion matrix and visualizations

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

all_models = {
    'Logistic Regression': logistic_regression_model,
    'Random Forest': random_forest_model,
    'SVM': svm_model
}

fig, axes = plt.subplots(1, len(all_models), figsize=(18, 5))

for ax, (name, model) in zip(axes, all_models.items()):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
    ax.set_title(name)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('True')
```

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


Logistic Regression
Random Forest
SVM
True
Predicted