Step 1 : Import Dataset And Preprocessing

In [1]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load the Excel file
file_path = 'Dataset.xlsx'
data = pd.read_excel(file_path)
```

In [2]:
```
# Display the first few rows
print("First 5 rows of the dataset:")
print(data.head())
```

```
First 5 rows of the dataset:
         Group   Kynurenine   Serotonin  Quinolinic acid        HIAA  \
0  Lung_cancer  2249.666940  407.370690       486.802969  135.926868
1  Lung_cancer  1812.275457   43.079546        74.608364   55.758398
2  Lung_cancer  2482.680951  340.997488        71.482022   32.371241
3  Lung_cancer  3566.573184  168.643625       222.020189   69.663812
4  Lung_cancer  2232.203826  411.608973       210.750064   31.457593

   Tryptamin  Antranillic acid  Indole-3-lactic acid  Indole-3-acetic acid  \
0   0.093102         74.904346            260.997626           1713.027711
1   0.077787          9.089924            716.617621            577.462903
2   0.126461         21.705576            709.612735           1666.923200
3   0.073761         24.789829            822.232828            767.956643
4   0.078109         15.223110            453.228236           1455.820332

   Indole-3-carboxaldehyde  ...  C16-1-OH    C16-OH     C18-2     C18-1  \
0                19.523107  ...  0.041468  0.002295  0.084394  0.132631
1                31.580789  ...  0.051896  0.001930  0.047796  0.091214
2                35.729249  ...  0.050163  0.002295  0.233086  0.425281
3                66.279242  ...  0.048238  0.002317  0.123458  0.264157
4                49.783224  ...  0.079068  0.002633  0.069019  0.141034

       C18  C18-1-OH    C18-OH   ADMA   SDMA  Choline
0  0.045411  0.003286  0.002047  0.826  1.906   33.279
1  0.027423  0.001090  0.001738  0.811  0.765   25.916
2  0.126856  0.005042  0.001993  1.652  1.760   65.039
3  0.059143  0.003052  0.001364  0.843  1.116   30.842
4  0.056948  0.002202  0.002455  0.738  0.619   26.732

[5 rows x 64 columns]
```

In [3]:
```
# Display dataset structure
print("\nDataset information:")
data.info()
```

```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 218 entries, 0 to 217
Data columns (total 64 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Group                    218 non-null    object
 1   Kynurenine               218 non-null    float64
 2   Serotonin                218 non-null    float64
 3   Quinolinic acid          218 non-null    float64
 4   HIAA                     218 non-null    float64
 5   Tryptamin                218 non-null    float64
 6   Antranillic acid         218 non-null    float64
 7   Indole-3-lactic acid     218 non-null    float64
 8   Indole-3-acetic acid     218 non-null    float64
 9   Indole-3-carboxaldehyde  218 non-null    float64
 10  Indole-3-acrylic acid    218 non-null    float64
 11  Indole-3-propionic acid  218 non-null    float64
 12  Indole-3-butyric acid    218 non-null    float64
 13  Xanturenic acid          218 non-null    float64
 14  Kynurenic acid           218 non-null    float64
 15  Gly                      218 non-null    float64
 16  Ala                      218 non-null    float64
 17  Pro                      218 non-null    float64
 18  Val                      218 non-null    float64
 19  Leu                      218 non-null    float64
 20  Ile                      218 non-null    float64
 21  Orn                      218 non-null    float64
 22  Asp                      218 non-null    float64
 23  Phe                      218 non-null    float64
 24  Arg                      218 non-null    float64
 25  Cit                      218 non-null    float64
 26  Ser                      218 non-null    float64
 27  Thr                      218 non-null    float64
 28  Lys                      218 non-null    float64
 29  Trp                      218 non-null    float64
 30  Tyr                      218 non-null    float64
 31  Meth                     218 non-null    float64
 32  C0                       218 non-null    float64
 33  C2                       218 non-null    float64
 34  C3                       218 non-null    float64
 35  C4                       218 non-null    float64
 36  C5-1                     218 non-null    float64
 37  C5                       218 non-null    float64
 38  C6                       218 non-null    float64
 39  C5-OH                    218 non-null    float64
 40  C5-DC                    218 non-null    float64
 41  C8-1                     218 non-null    float64
 42  C6-DC                    218 non-null    float64
 43  C10-2                    218 non-null    float64
 44  C10-1                    218 non-null    float64
 45  C10                      218 non-null    float64
 46  C12-1                    218 non-null    float64
 47  C12                      218 non-null    float64
 48  C14-2                    218 non-null    float64
 49  C14-1                    218 non-null    float64
 50  C14                      218 non-null    float64
 51  C14-OH                   218 non-null    float64
 52  C16-1                    218 non-null    float64
 53  C16                      218 non-null    float64
```

```
 54  C16-1-OH                218 non-null    float64
 55  C16-OH                  218 non-null    float64
 56  C18-2                   218 non-null    float64
 57  C18-1                   218 non-null    float64
 58  C18                     218 non-null    float64
 59  C18-1-OH                218 non-null    float64
 60  C18-OH                  218 non-null    float64
 61  ADMA                    218 non-null    float64
 62  SDMA                    218 non-null    float64
 63  Choline                 218 non-null    float64
dtypes: float64(63), object(1)
memory usage: 109.1+ KB
```

In [4]:
```
# Display a statistical summary of each column
print("\nStatistical Summary of dataset: ")
print(data.describe())
```

```
Statistical Summary of dataset:
       Kynurenine    Serotonin  Quinolinic acid        HIAA   Tryptamin  \
count   218.000000   218.000000       218.000000  218.000000  218.000000
mean   2733.184385   623.806072       141.427483   82.154112    0.187697
std    1082.149771   436.064109       149.447621  102.469863    0.143057
min     581.659850    30.021253        29.788981   10.264901    0.010412
25%    2102.644948   352.252271        75.486626   41.357925    0.087274
50%    2555.665161   545.530631       107.831794   59.003050    0.141356
75%    3070.083199   776.571068       161.676772   81.649731    0.249939
max    9462.247186  3505.424635      1737.169183  923.825783    0.988306

       Antranillic acid  Indole-3-lactic acid  Indole-3-acetic acid  \
count        218.000000            218.000000            218.000000
mean          20.145692            696.222509           1756.732163
std           19.824655            385.517701            859.773715
min            3.671316            124.970687            294.898945
25%           11.383823            466.455863           1148.384642
50%           15.159818            598.200048           1659.773823
75%           23.110936            836.503839           2133.596930
max          221.960533           3343.480307           4590.185133

       Indole-3-carboxaldehyde  Indole-3-acrylic acid  ...    C16-1-OH  \
count               218.000000             218.000000  ...  218.000000
mean                 54.359023              16.279492  ...    0.064649
std                  31.230552              25.181066  ...    0.032758
min                  10.085085               0.134148  ...    0.001453
25%                  34.652062               5.930434  ...    0.042326
50%                  45.839050              10.754244  ...    0.055370
75%                  65.141650              18.936207  ...    0.080470
max                 204.145566             329.266051  ...    0.209485

           C16-OH       C18-2       C18-1         C18    C18-1-OH      C18-OH  \
count  218.000000  218.000000  218.000000  218.000000  218.000000  218.000000
mean     0.001704    0.099771    0.171496    0.057323    0.002272    0.001851
std      0.000556    0.045073    0.076252    0.023063    0.001367    0.000491
min      0.000079    0.022127    0.035562    0.014325    0.000000    0.000000
25%      0.001267    0.069462    0.116157    0.041156    0.001334    0.001567
50%      0.001737    0.092103    0.159748    0.055345    0.002000    0.001844
75%      0.002099    0.119005    0.213814    0.066998    0.002792    0.002107
max      0.003142    0.326812    0.564237    0.162845    0.009357    0.003427

             ADMA        SDMA     Choline
count  218.000000  218.000000  218.000000
mean     0.682445    0.630537   34.922550
std      0.238501    0.281918   12.635667
min      0.258000    0.008000   17.910000
25%      0.523500    0.438000   27.400500
50%      0.626000    0.603500   32.668500
75%      0.829750    0.769500   37.566250
max      1.652000    1.906000  127.507000

[8 rows x 63 columns]
```

In [5]:
```
# Display data type
print("\nData types: ")
print(data.dtypes)
```

```
Data types:
Group               object
Kynurenine         float64
Serotonin          float64
Quinolinic acid    float64
HIAA               float64
                    ...
C18-1-OH           float64
C18-OH             float64
ADMA               float64
SDMA               float64
Choline            float64
Length: 64, dtype: object
```

In [6]:
```python
# Identify missing values
print("\nColumns with missing values:")

# Check for missing values in the dataset
missing_values = data.isnull().sum()

# If there are missing values, print the columns with missing values
if missing_values.any():
    print(missing_values[missing_values > 0])
else:
    # If there are no missing values, print "no missing value"
    print("No missing values")
```

```
Columns with missing values:
No missing values
```

In [7]:
```python
# Encode categorical 'Group' column using Label Encoding
if 'Group' in data.columns and data['Group'].dtype == 'object':
    encoder = LabelEncoder()
    data['Group'] = encoder.fit_transform(data['Group'])

print("\nNew dataset with 'Group' column using Label Encoding:")
print(data)
```

```
New dataset with 'Group' column using Label Encoding:
     Group   Kynurenine   Serotonin  Quinolinic acid        HIAA  Tryptamin  \
0        1  2249.666940  407.370690       486.802969  135.926868   0.093102
1        1  1812.275457   43.079546        74.608364   55.758398   0.077787
2        1  2482.680951  340.997488        71.482022   32.371241   0.126461
3        1  3566.573184  168.643625       222.020189   69.663812   0.073761
4        1  2232.203826  411.608973       210.750064   31.457593   0.078109
..     ...          ...         ...              ...         ...        ...
213      0  2552.411721  204.940093        98.650400   60.177646   0.140174
214      0  3315.467931  416.326442       131.672509   56.061266   0.146750
215      0  4583.185385  682.422121       183.196092  153.933355   0.167480
216      0  3081.832973  632.118196       145.341497   89.517709   0.122661
217      0  2295.635906  398.060312        69.455667  740.079312   0.309298

     Antranillic acid  Indole-3-lactic acid  Indole-3-acetic acid  \
0           74.904346            260.997626           1713.027711
1            9.089924            716.617621            577.462903
2           21.705576            709.612735           1666.923200
3           24.789829            822.232828            767.956643
4           15.223110            453.228236           1455.820332
..                ...                   ...                   ...
213         15.634764            720.705601           1784.116026
214         13.170071            543.435756           1124.862646
215         33.277130           1035.385660           1801.072488
216         36.142228            845.556805           2284.022165
217         18.034117            422.448492           2078.398744

     Indole-3-carboxaldehyde  ...  C16-1-OH    C16-OH     C18-2     C18-1  \
0                  19.523107  ...  0.041468  0.002295  0.084394  0.132631
1                  31.580789  ...  0.051896  0.001930  0.047796  0.091214
2                  35.729249  ...  0.050163  0.002295  0.233086  0.425281
3                  66.279242  ...  0.048238  0.002317  0.123458  0.264157
4                  49.783224  ...  0.079068  0.002633  0.069019  0.141034
..                       ...  ...       ...       ...       ...       ...
213                49.285531  ...  0.029185  0.001838  0.100887  0.234643
214                43.994419  ...  0.065723  0.002000  0.095678  0.110249
215               123.780417  ...  0.139945  0.002995  0.052837  0.075015
216               104.515905  ...  0.083160  0.002378  0.075278  0.101778
217                30.122534  ...  0.125351  0.002539  0.053806  0.078886

          C18  C18-1-OH    C18-OH   ADMA   SDMA  Choline
0    0.045411  0.003286  0.002047  0.826  1.906   33.279
1    0.027423  0.001090  0.001738  0.811  0.765   25.916
2    0.126856  0.005042  0.001993  1.652  1.760   65.039
3    0.059143  0.003052  0.001364  0.843  1.116   30.842
4    0.056948  0.002202  0.002455  0.738  0.619   26.732
..        ...       ...       ...    ...    ...      ...
213  0.056889  0.003981  0.001535  0.630  0.551   28.851
214  0.040257  0.000332  0.002127  0.684  0.390   27.371
215  0.035428  0.000645  0.002782  0.584  0.423   27.061
216  0.042429  0.001376  0.002108  0.570  0.457   32.133
217  0.039509  0.001088  0.002035  0.552  0.675   30.955

[218 rows x 64 columns]
```

In [8]:
```python
# Save the preprocessed data to an Excel file
output_file_path = 'Preprocessed_Dataset.xlsx'  # Output file path
try:
    data.to_excel(output_file_path, index=False)
    print(f"\nPreprocessed data saved to: {output_file_path}")
```

```
except PermissionError:
    print("\nError: Unable to save the file. Please check the file path or permi
```

```
Preprocessed data saved to: Preprocessed_Dataset.xlsx
```

Step 2 : Import Preprocessing Dataset

In [9]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load your dataset
file_path = 'Preprocessed_Dataset.xlsx'
data = pd.read_excel(file_path)

# Separate features (X) and target (y)
X = data.drop(columns=['Group'])  # Exclude 'Group' from the features
y = data['Group']

# Standardize the feature values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
```

In [10]:
```
#import pickle
#
## Save the scaler
#with open('scaler.pkl', 'wb') as file:
#    pickle.dump(scaler, file)
```

In [11]:
```
# Save testing dataset
# Combine the testing dataset (features and target)
X_test_df = pd.DataFrame(X_test, columns=X.columns)  # Convert X_test to DataFra
y_test_df = pd.DataFrame(y_test.reset_index(drop=True), columns=['Group'])  # Co
test_dataset = pd.concat([X_test_df, y_test_df], axis=1)  # Combine features ana

# Save the testing dataset to a file (e.g., CSV or Excel)
test_dataset_file = 'Testing_Dataset.xlsx'  # Change to .csv for a CSV file
test_dataset.to_excel(test_dataset_file, index=False)  # Save as Excel
# For CSV: test_dataset.to_csv('Testing_Dataset.csv', index=False)

print(f"Testing dataset saved as {test_dataset_file}")
```

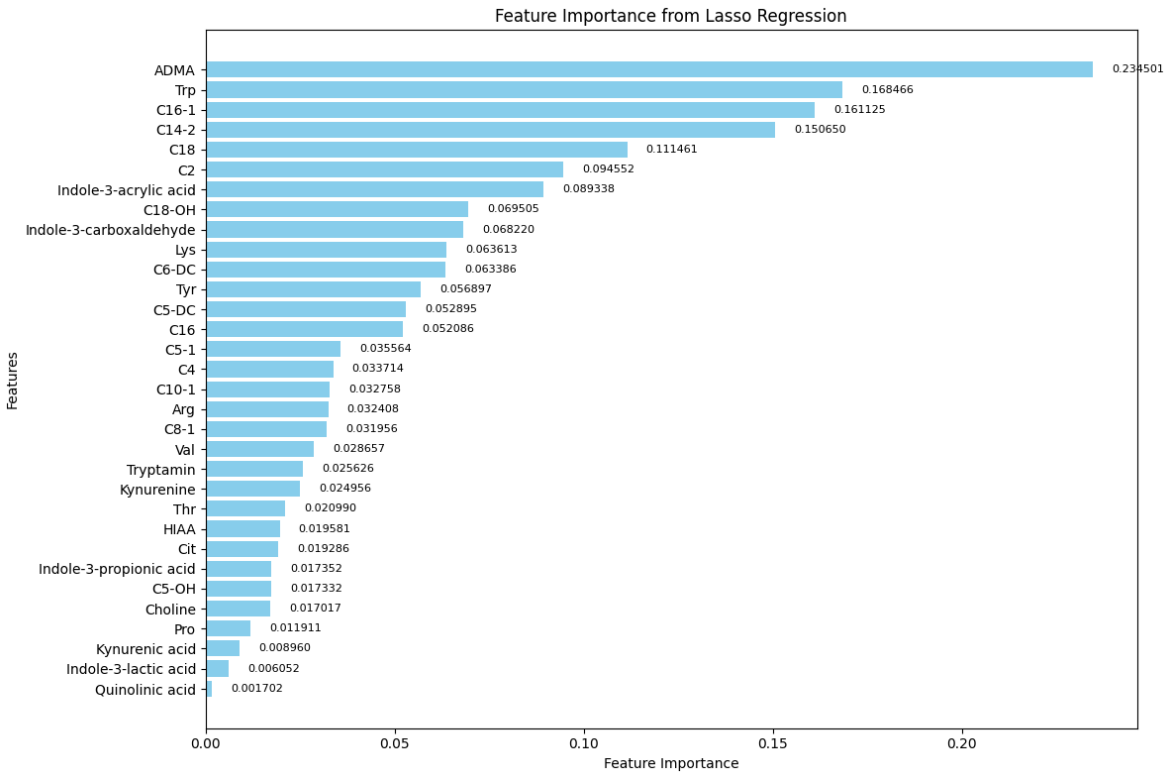
```
Testing dataset saved as Testing_Dataset.xlsx
```

Step 3 : Lasso Regression For Feature Selection (Lasso shrinks less important feature coefficients to zero)

In [12]:
```
from sklearn.linear_model import Lasso
import numpy as np
import matplotlib.pyplot as plt

# Apply Lasso Regression
lasso = Lasso(alpha=0.01)
lasso.fit(X_train, y_train)

# Get the feature importance
feature_importance = np.abs(lasso.coef_)
```

```
# Map feature importance to feature names and list out them
important_features = pd.Series(feature_importance, index=X.columns)  # Ensure co
important_features = important_features[important_features > 0].sort_values(asce

# Print the count of features with importance > 0
print(f"\nNumber of features with importance > 0: {important_features.shape[0]}\

# Print all features with importance > 0
print("All features with the importance > 0:")
print(important_features)

# Plot the bar chart
plt.figure(figsize=(12, 8))
bars = plt.barh(important_features.index, important_features.values, color='skyb

# Add annotations to the bars
for bar, value in zip(bars, important_features.values):
    plt.text(
        bar.get_width() + 0.005,  # Position text slightly beyond the bar
        bar.get_y() + bar.get_height() / 2,  # Center text vertically
        f"{value:.6f}",  # Format the value to 6 decimal places
        va='center',  # Align text vertically to the center
        fontsize=8  # Font size for annotations
    )

# Customize the chart
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance from Lasso Regression')
plt.gca().invert_yaxis()  # To display the most important features at the top
plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```

```
Number of features with importance > 0: 32

All features with the importance > 0:
ADMA                        0.234501
Trp                         0.168466
C16-1                       0.161125
C14-2                       0.150650
C18                         0.111461
C2                          0.094552
Indole-3-acrylic acid       0.089338
C18-OH                      0.069505
Indole-3-carboxaldehyde     0.068220
Lys                         0.063613
C6-DC                       0.063386
Tyr                         0.056897
C5-DC                       0.052895
C16                         0.052086
C5-1                        0.035564
C4                          0.033714
C10-1                       0.032758
Arg                         0.032408
C8-1                        0.031956
Val                         0.028657
Tryptamin                   0.025626
Kynurenine                  0.024956
Thr                         0.020990
HIAA                        0.019581
Cit                         0.019286
Indole-3-propionic acid     0.017352
C5-OH                       0.017332
Choline                     0.017017
Pro                         0.011911
Kynurenic acid              0.008960
Indole-3-lactic acid        0.006052
Quinolinic acid             0.001702
dtype: float64
```



Step 4 : Random Forest Feature Importance (Random Forest provides a robust importance score for each feature)

Method 1 : Use a threshold based on the mean and median feature importance to select features

In [13]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import seaborn as sns

# Train a Random Forest Classifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Extract feature importance
rf_feature_importance = rf.feature_importances_

# Map feature importance to feature names
rf_important_features = pd.Series(rf_feature_importance, index=X.columns)
rf_important_features = rf_important_features.sort_values(ascending=False)

# Compute mean and median importance
mean_importance = rf_important_features.mean()
median_importance = rf_important_features.median()

# Select features above the mean importance threshold
selected_features_mean = rf_important_features[rf_important_features > mean_impo
print(f"\nNumber of features above mean importance: {selected_features_mean.shap
print(f"\nFeatures above mean importance ({mean_importance:.4f}):")
print(selected_features_mean)
```

```
Number of features above mean importance: 21

Features above mean importance (0.0159):
ADMA                        0.067053
C6-DC                       0.058265
C16-1-OH                    0.054920
Indole-3-carboxaldehyde     0.044340
Asp                         0.043080
Orn                         0.032257
Serotonin                   0.027863
C18-OH                      0.026965
Indole-3-acrylic acid       0.025506
Indole-3-propionic acid     0.025419
Tryptamin                   0.025132
HIAA                        0.024365
C16-1                       0.022065
Quinolinic acid             0.021271
C10-1                       0.020166
C18                         0.018470
Indole-3-lactic acid        0.018291
Cit                         0.017776
SDMA                        0.017770
C0                          0.017605
C5-OH                       0.016827
dtype: float64
```

In [14]:
```
# Select features above the median importance threshold
selected_features_median = rf_important_features[rf_important_features > median_
```

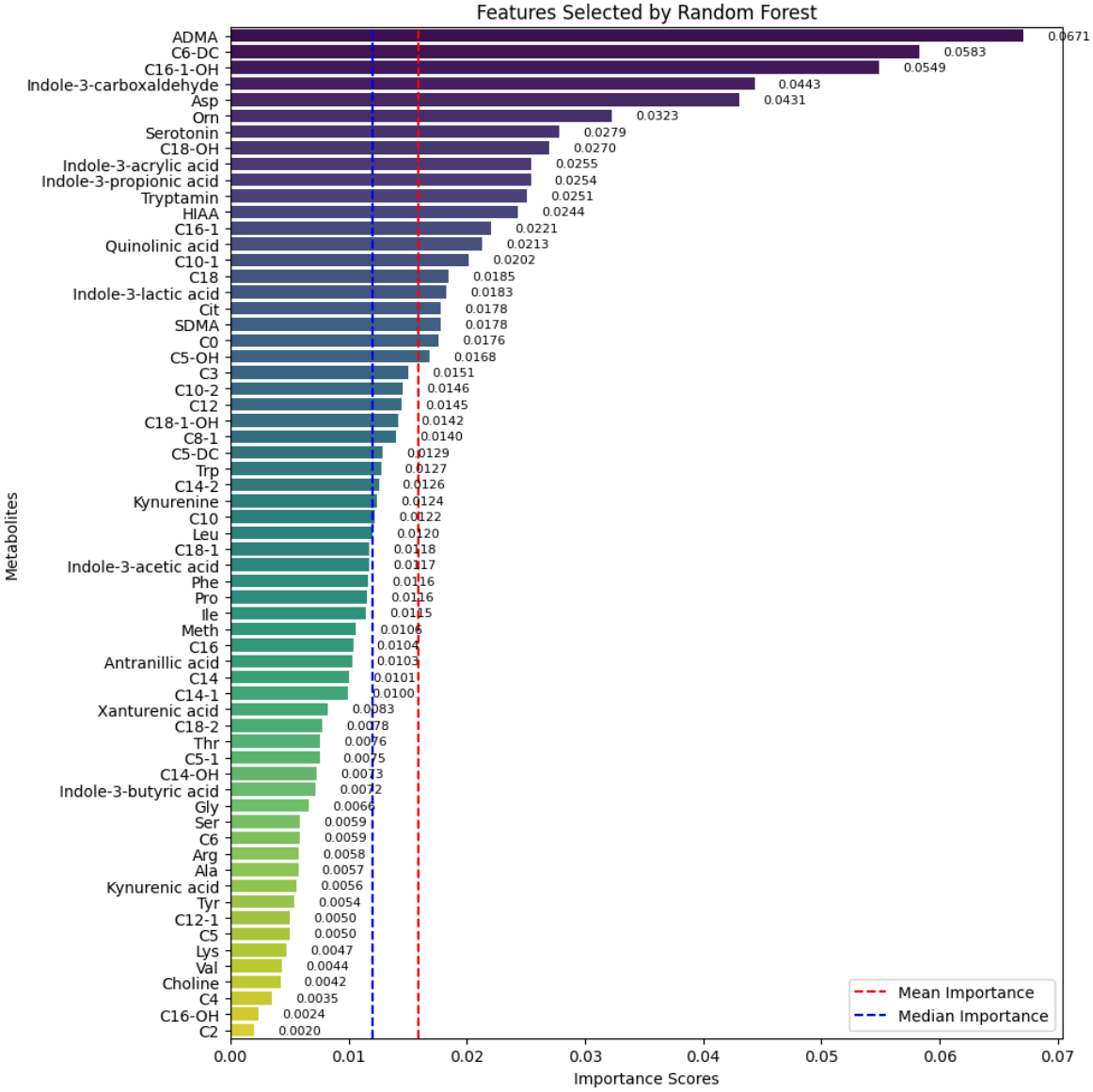
```
print(f"\nNumber of features above median importance: {selected_features_median.
print(f"\nFeatures above median importance ({median_importance:.4f}):")
print(selected_features_median)
```

```
Number of features above median importance: 31

Features above median importance (0.0120):
ADMA                        0.067053
C6-DC                       0.058265
C16-1-OH                    0.054920
Indole-3-carboxaldehyde     0.044340
Asp                         0.043080
Orn                         0.032257
Serotonin                   0.027863
C18-OH                      0.026965
Indole-3-acrylic acid       0.025506
Indole-3-propionic acid     0.025419
Tryptamin                   0.025132
HIAA                        0.024365
C16-1                       0.022065
Quinolinic acid             0.021271
C10-1                       0.020166
C18                         0.018470
Indole-3-lactic acid        0.018291
Cit                         0.017776
SDMA                        0.017770
C0                          0.017605
C5-OH                       0.016827
C3                          0.015088
C10-2                       0.014585
C12                         0.014489
C18-1-OH                    0.014165
C8-1                        0.013986
C5-DC                       0.012922
Trp                         0.012748
C14-2                       0.012556
Kynurenine                  0.012432
C10                         0.012238
dtype: float64
```

In [15]:
```
# Bar plot of feature importance
plt.figure(figsize=(10, 10))
sns.barplot(x=rf_important_features.values, y=rf_important_features.index, palet

# Annotate bars with importance values
for i, value in enumerate(rf_important_features.values):
    plt.text(value + 0.002, i, f"{value:.4f}", va='center', fontsize=8)

# Add mean and median lines
plt.axvline(mean_importance, color='r', linestyle='--', label='Mean Importance')
plt.axvline(median_importance, color='b', linestyle='--', label='Median Importan

# Add labels and title
plt.xlabel("Importance Scores")
plt.ylabel("Metabolites")
plt.title("Features Selected by Random Forest")
plt.legend()
plt.tight_layout()
```

```
# Show the plot
plt.show()
```

```
C:\Users\behsh\AppData\Local\Temp\ipykernel_20812\1111784011.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=rf_important_features.values, y=rf_important_features.index, pale
tte='viridis')
```



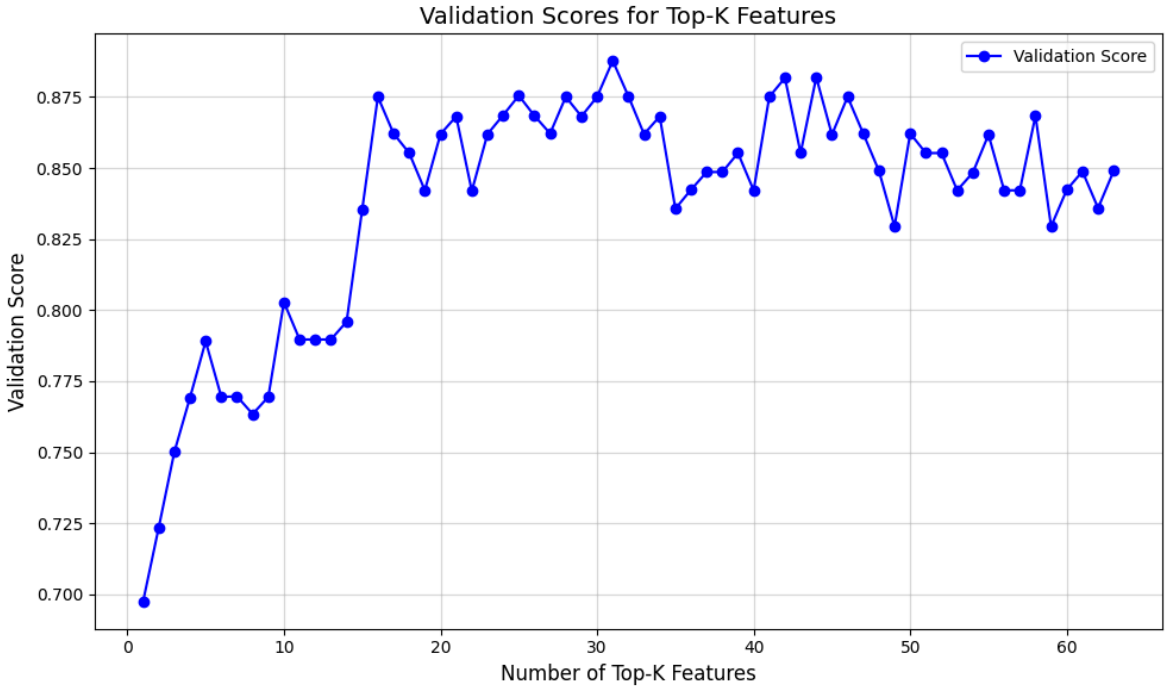
Method 2 : Experiments with top-K features (1-63) to validate model performance using cross-validation

In [16]:
```
# Convert X_train to a DataFrame (if it's a NumPy array)
X_train_df = pd.DataFrame(X_train, columns=X.columns)  # Ensure X.columns has co

# Experiment with top-K features (1-63) and validate performance
validation_scores = {}
print("\nValidation scores for top-K features:")
for k in range(1, 64):  # Test top 1 to 63 features
    top_k_features = rf_important_features.head(k).index.tolist()
    X_reduced = X_train_df[top_k_features]  # Use DataFrame indexing
    scores = cross_val_score(rf, X_reduced, y_train, cv=5, scoring='accuracy')
```

```
    mean_score = scores.mean()
    validation_scores[k] = mean_score
    print(f"Top-{k} features: Validation Score = {mean_score:.4f}")

# Plot validation scores as a graph
plt.figure(figsize=(10, 6))
plt.plot(list(validation_scores.keys()), list(validation_scores.values()), marke

# Graph labels and title
plt.xlabel("Number of Top-K Features", fontsize=12)
plt.ylabel("Validation Score", fontsize=12)
plt.title("Validation Scores for Top-K Features", fontsize=14)
plt.grid(alpha=0.5)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.legend(fontsize=10)
plt.tight_layout()

# Show the plot
plt.show()
```

```
Validation scores for top-K features:
Top-1 features: Validation Score = 0.6974
Top-2 features: Validation Score = 0.7234
Top-3 features: Validation Score = 0.7501
Top-4 features: Validation Score = 0.7692
Top-5 features: Validation Score = 0.7890
Top-6 features: Validation Score = 0.7695
Top-7 features: Validation Score = 0.7697
Top-8 features: Validation Score = 0.7634
Top-9 features: Validation Score = 0.7695
Top-10 features: Validation Score = 0.8028
Top-11 features: Validation Score = 0.7897
Top-12 features: Validation Score = 0.7897
Top-13 features: Validation Score = 0.7897
Top-14 features: Validation Score = 0.7959
Top-15 features: Validation Score = 0.8355
Top-16 features: Validation Score = 0.8751
Top-17 features: Validation Score = 0.8622
Top-18 features: Validation Score = 0.8553
Top-19 features: Validation Score = 0.8422
Top-20 features: Validation Score = 0.8619
Top-21 features: Validation Score = 0.8682
Top-22 features: Validation Score = 0.8419
Top-23 features: Validation Score = 0.8619
Top-24 features: Validation Score = 0.8686
Top-25 features: Validation Score = 0.8755
Top-26 features: Validation Score = 0.8684
Top-27 features: Validation Score = 0.8622
Top-28 features: Validation Score = 0.8751
Top-29 features: Validation Score = 0.8682
Top-30 features: Validation Score = 0.8751
Top-31 features: Validation Score = 0.8877
Top-32 features: Validation Score = 0.8753
Top-33 features: Validation Score = 0.8619
Top-34 features: Validation Score = 0.8682
Top-35 features: Validation Score = 0.8357
Top-36 features: Validation Score = 0.8424
Top-37 features: Validation Score = 0.8486
Top-38 features: Validation Score = 0.8486
Top-39 features: Validation Score = 0.8553
Top-40 features: Validation Score = 0.8422
Top-41 features: Validation Score = 0.8751
Top-42 features: Validation Score = 0.8817
Top-43 features: Validation Score = 0.8553
Top-44 features: Validation Score = 0.8819
Top-45 features: Validation Score = 0.8617
Top-46 features: Validation Score = 0.8751
Top-47 features: Validation Score = 0.8622
Top-48 features: Validation Score = 0.8490
Top-49 features: Validation Score = 0.8295
Top-50 features: Validation Score = 0.8622
Top-51 features: Validation Score = 0.8553
Top-52 features: Validation Score = 0.8553
Top-53 features: Validation Score = 0.8422
Top-54 features: Validation Score = 0.8484
Top-55 features: Validation Score = 0.8617
Top-56 features: Validation Score = 0.8422
Top-57 features: Validation Score = 0.8422
Top-58 features: Validation Score = 0.8684
Top-59 features: Validation Score = 0.8295
```

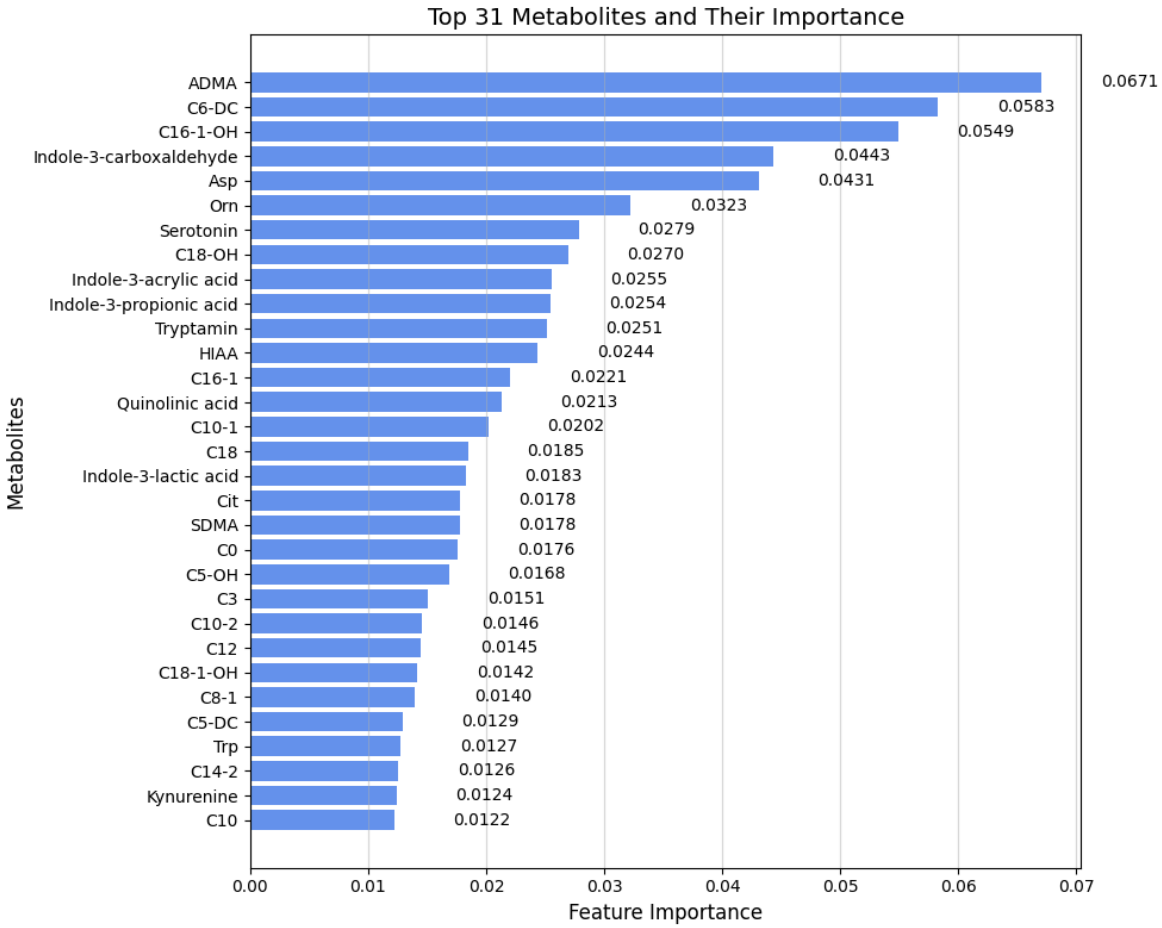
```
Top-60 features: Validation Score = 0.8424
Top-61 features: Validation Score = 0.8488
Top-62 features: Validation Score = 0.8359
Top-63 features: Validation Score = 0.8490
```



In [17]:
```python
# Find the optimal number of features
optimal_k = max(validation_scores, key=validation_scores.get)
print(f"\nOptimal number of features based on cross-validation: {optimal_k}")
print(f"Validation score with {optimal_k} features: {validation_scores[optimal_k

# List out the top metabolites for the optimal_k features
top_features_optimal_k = rf_important_features.head(optimal_k)
print("\nTop metabolites for the optimal number of features:")
print(top_features_optimal_k)

# Visualization: Horizontal bar chart for top metabolites and their importance
plt.figure(figsize=(10, 8))
bars = plt.barh(top_features_optimal_k.index, top_features_optimal_k.values, col

# Add labels to each bar
for bar in bars:
    width = bar.get_width()
    plt.text(
        width + 0.005,  # Slightly offset from the bar for better visibility
        bar.get_y() + bar.get_height() / 2,  # Center vertically in the bar
        f"{width:.4f}",  # Format the value to 4 decimal places
        va='center',  # Center align the label vertically
        fontsize=10
    )

# Labels and title
plt.xlabel("Feature Importance", fontsize=12)
plt.ylabel("Metabolites", fontsize=12)
plt.title(f"Top {optimal_k} Metabolites and Their Importance", fontsize=14)
plt.gca().invert_yaxis()  # Invert y-axis for better readability
plt.grid(axis='x', alpha=0.5)
plt.tight_layout()
```

```
# Show the plot
plt.show()
```

```
Optimal number of features based on cross-validation: 31
Validation score with 31 features: 0.8877

Top metabolites for the optimal number of features:
ADMA                        0.067053
C6-DC                       0.058265
C16-1-OH                    0.054920
Indole-3-carboxaldehyde     0.044340
Asp                         0.043080
Orn                         0.032257
Serotonin                   0.027863
C18-OH                      0.026965
Indole-3-acrylic acid       0.025506
Indole-3-propionic acid     0.025419
Tryptamin                   0.025132
HIAA                        0.024365
C16-1                       0.022065
Quinolinic acid             0.021271
C10-1                       0.020166
C18                         0.018470
Indole-3-lactic acid        0.018291
Cit                         0.017776
SDMA                        0.017770
C0                          0.017605
C5-OH                       0.016827
C3                          0.015088
C10-2                       0.014585
C12                         0.014489
C18-1-OH                    0.014165
C8-1                        0.013986
C5-DC                       0.012922
Trp                         0.012748
C14-2                       0.012556
Kynurenine                  0.012432
C10                         0.012238
dtype: float64
```



Step 5 : Recursive Feature Elimination (RFE)

In [18]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Initialize a Logistic Regression model
log_reg = LogisticRegression()
```

Loop through feature sets from 1 to 63

In [19]:
```
# Dictionary to store cross-validation scores for each feature set size
cv_scores3 = {}

# Loop through feature sets from 1 to 63
for num_features in range(1, 64):
    # Apply Recursive Feature Elimination (RFE) with num_features
    rfe = RFE(estimator=log_reg, n_features_to_select=num_features)
    rfe.fit(X_train_df, y_train)  # Use the DataFrame version of X_train

    # Get the selected features based on the RFE process
    selected_features_rfe = X_train_df.columns[rfe.support_].tolist()

    # Perform cross-validation and store the results
    X_train_reduced = X_train_df[selected_features_rfe]

    # Ensure that y_train has the same number of samples as X_train_reduced
    assert X_train_reduced.shape[0] == y_train.shape[0], f"Shape mismatch: {X_tr

    scores = cross_val_score(log_reg, X_train_reduced, y_train, cv=5, scoring='a
```

```
    cv_scores3[num_features] = scores.mean()

    # Print selected features and their CV score
    print(f"Selected features for {num_features} features: {selected_features_rf
    print(f"Cross-validation score for {num_features} features: {cv_scores3[num_

# Find the optimal number of features based on cross-validation
optimal_num_features3 = max(cv_scores3, key=cv_scores3.get)
print(f"\nOptimal number of features based on cross-validation: {optimal_num_fea
print(f"Validation score with {optimal_num_features3} features: {cv_scores3[opti

# Re-run RFE for the optimal number of features
rfe_optimal = RFE(estimator=log_reg, n_features_to_select=optimal_num_features3)
rfe_optimal.fit(X_train_df, y_train)

# Get the selected features for the optimal number of features
optimal_selected_features3 = X_train_df.columns[rfe_optimal.support_].tolist()
print(f"\nSelected features for the optimal number of features ({optimal_num_fea
print(optimal_selected_features3)

# Visualization
plt.figure(figsize=(10, 6))

# Plot the cross-validations scores
plt.plot(list(cv_scores3.keys()), list(cv_scores3.values()), marker='o', color='

# Highlight the optimal number of features
optimal_score = cv_scores3[optimal_num_features3]
plt.scatter(optimal_num_features3, optimal_score, color='darkorange', label=f'Op

# Add annotatioin for the optimal point
plt.text(optimal_num_features3, optimal_score,
         f'  {optimal_num_features3} features\n  Scores: {optimal_score:.4f}',
         fontsize=10, color='darkorange')

# Add titles and labels
plt.title("Cross-Validation Scores for RFE Selected Features", fontsize=14)
plt.xlabel("Number of Selected Features", fontsize=12)
plt.ylabel("Cross-Validation Score", fontsize=12)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

```
Selected features for 1 features: ['ADMA']
Cross-validation score for 1 features: 0.7303
Selected features for 2 features: ['Trp', 'ADMA']
Cross-validation score for 2 features: 0.8477
Selected features for 3 features: ['Trp', 'C14-2', 'ADMA']
Cross-validation score for 3 features: 0.8280
Selected features for 4 features: ['Trp', 'C14-2', 'C16-1', 'ADMA']
Cross-validation score for 4 features: 0.8609
Selected features for 5 features: ['Trp', 'C6-DC', 'C14-2', 'C16-1', 'ADMA']
Cross-validation score for 5 features: 0.8809
Selected features for 6 features: ['Trp', 'C6-DC', 'C14-2', 'C16-1', 'C18', 'ADM
A']
Cross-validation score for 6 features: 0.9142
Selected features for 7 features: ['Trp', 'C8-1', 'C6-DC', 'C14-2', 'C16-1', 'C1
8', 'ADMA']
Cross-validation score for 7 features: 0.9342
Selected features for 8 features: ['Trp', 'C5-OH', 'C8-1', 'C6-DC', 'C14-2', 'C16
-1', 'C18', 'ADMA']
Cross-validation score for 8 features: 0.9540
Selected features for 9 features: ['Trp', 'C2', 'C5-OH', 'C8-1', 'C6-DC', 'C14-
2', 'C16-1', 'C18', 'ADMA']
Cross-validation score for 9 features: 0.9540
Selected features for 10 features: ['Cit', 'Trp', 'C2', 'C5-OH', 'C8-1', 'C6-DC',
'C14-2', 'C16-1', 'C18', 'ADMA']
Cross-validation score for 10 features: 0.9475
Selected features for 11 features: ['Cit', 'Trp', 'C2', 'C5-OH', 'C8-1', 'C6-DC',
'C14-2', 'C16-1', 'C16', 'C18', 'ADMA']
Cross-validation score for 11 features: 0.9406
Selected features for 12 features: ['Indole-3-carboxaldehyde', 'Cit', 'Trp', 'C
2', 'C5-OH', 'C8-1', 'C6-DC', 'C14-2', 'C16-1', 'C16', 'C18', 'ADMA']
Cross-validation score for 12 features: 0.9538
Selected features for 13 features: ['Indole-3-carboxaldehyde', 'Cit', 'Thr', 'Tr
p', 'C2', 'C5-OH', 'C8-1', 'C6-DC', 'C14-2', 'C16-1', 'C16', 'C18', 'ADMA']
Cross-validation score for 13 features: 0.9471
Selected features for 14 features: ['Indole-3-carboxaldehyde', 'Cit', 'Thr', 'Ly
s', 'Trp', 'C2', 'C5-OH', 'C8-1', 'C6-DC', 'C14-2', 'C16-1', 'C16', 'C18', 'ADM
A']
Cross-validation score for 14 features: 0.9604
Selected features for 15 features: ['Indole-3-carboxaldehyde', 'Val', 'Cit', 'Th
r', 'Lys', 'Trp', 'C2', 'C5-OH', 'C8-1', 'C6-DC', 'C14-2', 'C16-1', 'C16', 'C18',
'ADMA']
Cross-validation score for 15 features: 0.9540
Selected features for 16 features: ['Indole-3-carboxaldehyde', 'Val', 'Cit', 'Th
r', 'Lys', 'Trp', 'C2', 'C4', 'C5-OH', 'C8-1', 'C6-DC', 'C14-2', 'C16-1', 'C16',
'C18', 'ADMA']
Cross-validation score for 16 features: 0.9604
Selected features for 17 features: ['Indole-3-carboxaldehyde', 'Val', 'Cit', 'Th
r', 'Lys', 'Trp', 'C2', 'C4', 'C5-OH', 'C8-1', 'C6-DC', 'C10-1', 'C14-2', 'C16-
1', 'C16', 'C18', 'ADMA']
Cross-validation score for 17 features: 0.9602
Selected features for 18 features: ['Indole-3-carboxaldehyde', 'Val', 'Cit', 'Th
r', 'Lys', 'Trp', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C14-
2', 'C16-1', 'C16', 'C18', 'ADMA']
Cross-validation score for 18 features: 0.9535
Selected features for 19 features: ['Indole-3-carboxaldehyde', 'Indole-3-acrylic
acid', 'Val', 'Cit', 'Thr', 'Lys', 'Trp', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1',
'C6-DC', 'C10-1', 'C14-2', 'C16-1', 'C16', 'C18', 'ADMA']
Cross-validation score for 19 features: 0.9469
Selected features for 20 features: ['Indole-3-carboxaldehyde', 'Indole-3-acrylic
acid', 'Val', 'Cit', 'Thr', 'Lys', 'Trp', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1',
```

```
'C6-DC', 'C10-1', 'C14-2', 'C16-1', 'C16', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 20 features: 0.9540
Selected features for 21 features: ['Indole-3-carboxaldehyde', 'Indole-3-acrylic
acid', 'Val', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C
8-1', 'C6-DC', 'C10-1', 'C14-2', 'C16-1', 'C16', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 21 features: 0.9538
Selected features for 22 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Val', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C
5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C14-2', 'C16-1', 'C16', 'C18', 'C18-O
H', 'ADMA']
Cross-validation score for 22 features: 0.9602
Selected features for 23 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Val', 'Cit', 'Thr', 'Lys',
'Trp', 'Tyr', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C14-2', 'C
16-1', 'C16', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 23 features: 0.9735
Selected features for 24 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Val', 'Cit', 'Thr', 'Lys',
'Trp', 'Tyr', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C14-2', 'C
16-1', 'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 24 features: 0.9669
Selected features for 25 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Pro', 'Val', 'Cit', 'Thr',
'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C14
-2', 'C16-1', 'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 25 features: 0.9669
Selected features for 26 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Pro', 'Val', 'Cit', 'Thr',
'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C1
2', 'C14-2', 'C16-1', 'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 26 features: 0.9669
Selected features for 27 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Pro', 'Val', 'Asp', 'Cit',
'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-
1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 27 features: 0.9604
Selected features for 28 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Pr
o', 'Val', 'Asp', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C5-OH', 'C5-D
C', 'C8-1', 'C6-DC', 'C10-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-1', 'C18', 'C1
8-OH', 'ADMA']
Cross-validation score for 28 features: 0.9671
Selected features for 29 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Pr
o', 'Val', 'Asp', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C6', 'C5-OH',
'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-1', 'C1
8', 'C18-OH', 'ADMA']
Cross-validation score for 29 features: 0.9604
Selected features for 30 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Pr
o', 'Val', 'Asp', 'Arg', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C6', 'C5
-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-1',
'C18', 'C18-OH', 'ADMA']
Cross-validation score for 30 features: 0.9738
Selected features for 31 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Pr
o', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C
6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12', 'C14-2', 'C16-1', 'C16',
'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 31 features: 0.9738
```

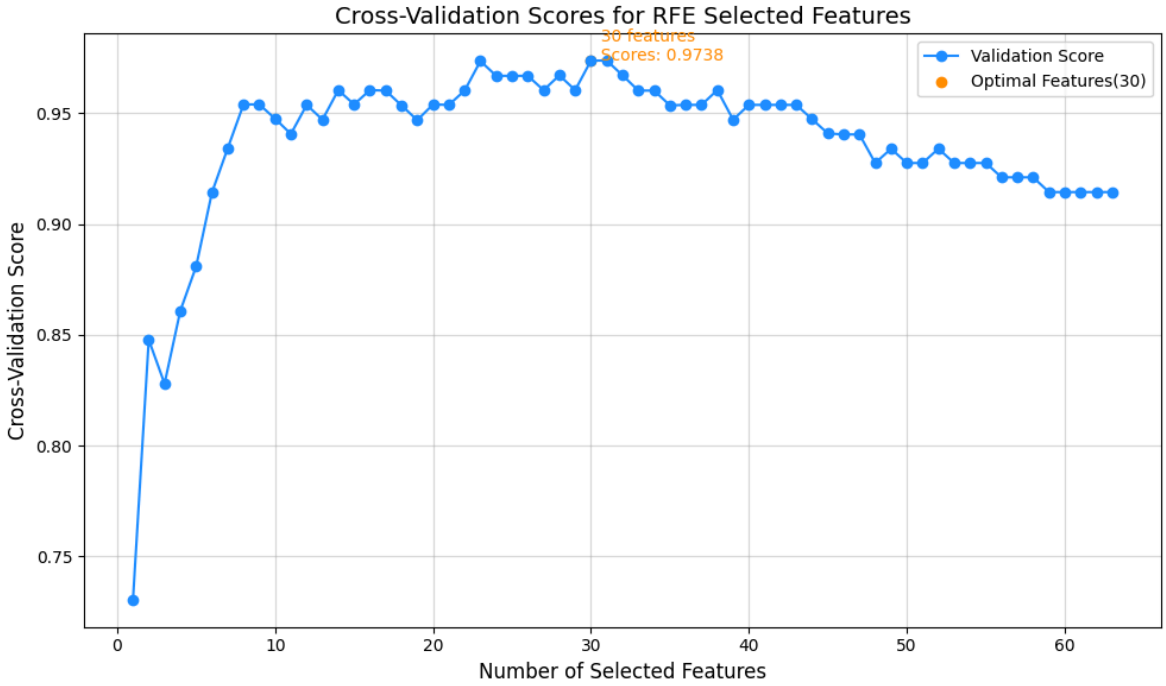
```
Selected features for 32 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Pr
o', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C
6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1',
'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 32 features: 0.9671
Selected features for 33 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Pr
o', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2',
'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12-1', 'C12', 'C14-2',
'C16-1', 'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 33 features: 0.9602
Selected features for 34 features: ['Quinolinic acid', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Pr
o', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2',
'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12-1', 'C12', 'C14-2',
'C16-1', 'C16', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 34 features: 0.9602
Selected features for 35 features: ['Quinolinic acid', 'HIAA', 'Indole-3-carboxal
dehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric ac
id', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr', 'Lys', 'Trp', 'Tyr', 'Met
h', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12-1', 'C12',
'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 35 features: 0.9535
Selected features for 36 features: ['Quinolinic acid', 'HIAA', 'Tryptamin', 'Indo
le-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole
-3-butyric acid', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr', 'Lys', 'Trp',
'Tyr', 'Meth', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-1', 'C12
-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA']
Cross-validation score for 36 features: 0.9538
Selected features for 37 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr',
'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC',
'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C18', 'C18-O
H', 'ADMA']
Cross-validation score for 37 features: 0.9538
Selected features for 38 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr',
'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC',
'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C18', 'C18-O
H', 'ADMA', 'SDMA']
Cross-validation score for 38 features: 0.9604
Selected features for 39 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Thr',
'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC',
'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C1
8', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 39 features: 0.9471
Selected features for 40 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit',
'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1',
'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-2', 'C18
-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 40 features: 0.9538
Selected features for 41 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
```

```
cid', 'Indole-3-butyric acid', 'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit',
'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C4', 'C5-1', 'C6', 'C5-OH', 'C5-DC',
'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-
2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 41 features: 0.9538
Selected features for 42 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit',
'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C4', 'C5-1', 'C6', 'C5-OH', 'C5
-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16',
'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 42 features: 0.9538
Selected features for 43 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit',
'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C6', 'C5-O
H', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1',
'C16', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 43 features: 0.9538
Selected features for 44 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Kynurenic acid', 'Ala', 'Pro', 'Val', 'Asp', 'Ph
e', 'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C
5-1', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12',
'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 44 features: 0.9473
Selected features for 45 features: ['Serotonin', 'Quinolinic acid', 'HIAA', 'Tryp
tamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic a
cid', 'Indole-3-butyric acid', 'Xanturenic acid', 'Kynurenic acid', 'Ala', 'Pro',
'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C
2', 'C3', 'C4', 'C5-1', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-
1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C18', 'C18-OH',
'ADMA', 'SDMA']
Cross-validation score for 45 features: 0.9409
Selected features for 46 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-
3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic acid', 'Kynurenic acid',
'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Ty
r', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C
10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-2', 'C18-1', 'C18',
'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 46 features: 0.9404
Selected features for 47 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser',
'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C6', 'C5-OH', 'C5-
DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16',
'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 47 features: 0.9404
Selected features for 48 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser',
'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C6', 'C5-OH', 'C5-
DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1', 'C16',
'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 48 features: 0.9275
Selected features for 49 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
```

```
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Asp', 'Phe', 'Arg', 'Cit',
'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C6', 'C5-O
H', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C16-1',
'C16', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 49 features: 0.9340
Selected features for 50 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C6',
'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C1
6-1', 'C16', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 50 features: 0.9275
Selected features for 51 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C6',
'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-2', 'C1
6-1', 'C16', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-OH', 'C18-OH', 'ADMA', 'SD
MA']
Cross-validation score for 51 features: 0.9275
Selected features for 52 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C5',
'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-
2', 'C16-1', 'C16', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-OH', 'C18-OH', 'ADM
A', 'SDMA']
Cross-validation score for 52 features: 0.9340
Selected features for 53 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C5',
'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-
2', 'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-OH', 'C
18-OH', 'ADMA', 'SDMA']
Cross-validation score for 53 features: 0.9275
Selected features for 54 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C5',
'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C12-1', 'C12', 'C14-
2', 'C14', 'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-
OH', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 54 features: 0.9275
Selected features for 55 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C2', 'C3', 'C4', 'C5-1', 'C5',
'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C10', 'C12-1', 'C12',
'C14-2', 'C14', 'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C
18-1-OH', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 55 features: 0.9275
Selected features for 56 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
```

```
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C0', 'C2', 'C3', 'C4', 'C5-1',
'C5', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C10', 'C12-1',
'C12', 'C14-2', 'C14', 'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C
18', 'C18-1-OH', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 56 features: 0.9211
Selected features for 57 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Orn', 'Asp', 'Phe', 'Arg',
'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C0', 'C2', 'C3', 'C4', 'C5-1',
'C5', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C10', 'C12-1',
'C12', 'C14-2', 'C14-1', 'C14', 'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C
18-1', 'C18', 'C18-1-OH', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 57 features: 0.9211
Selected features for 58 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Ile', 'Orn', 'Asp', 'Phe',
'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C0', 'C2', 'C3', 'C4',
'C5-1', 'C5', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C10',
'C12-1', 'C12', 'C14-2', 'C14-1', 'C14', 'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C
18-2', 'C18-1', 'C18', 'C18-1-OH', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 58 features: 0.9211
Selected features for 59 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-acetic acid', 'Indole-3-carboxaldehyde', 'Indole-3
-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-butyric acid', 'Xanturenic a
cid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Ile', 'Orn', 'Asp', 'Phe',
'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth', 'C0', 'C2', 'C3', 'C4',
'C5-1', 'C5', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C10',
'C12-1', 'C12', 'C14-2', 'C14-1', 'C14', 'C14-OH', 'C16-1', 'C16', 'C16-1-OH', 'C
16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-OH', 'C18-OH', 'ADMA', 'SDMA']
Cross-validation score for 59 features: 0.9144
Selected features for 60 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-lactic acid', 'Indole-3-acetic acid', 'Indole-3-ca
rboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-buty
ric acid', 'Xanturenic acid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Il
e', 'Orn', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth',
'C0', 'C2', 'C3', 'C4', 'C5-1', 'C5', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C
10-2', 'C10-1', 'C10', 'C12-1', 'C12', 'C14-2', 'C14-1', 'C14', 'C14-OH', 'C16-
1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-OH', 'C18-OH',
'ADMA', 'SDMA']
Cross-validation score for 60 features: 0.9144
Selected features for 61 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-lactic acid', 'Indole-3-acetic acid', 'Indole-3-ca
rboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-buty
ric acid', 'Xanturenic acid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Il
e', 'Orn', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr', 'Meth',
'C0', 'C2', 'C3', 'C4', 'C5-1', 'C5', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C
10-2', 'C10-1', 'C10', 'C12-1', 'C12', 'C14-2', 'C14-1', 'C14', 'C14-OH', 'C16-
1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-OH', 'C18-OH',
'ADMA', 'SDMA', 'Choline']
Cross-validation score for 61 features: 0.9144
Selected features for 62 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Indole-3-lactic acid', 'Indole-3-acetic acid', 'Indole-3-ca
rboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Indole-3-buty
ric acid', 'Xanturenic acid', 'Kynurenic acid', 'Gly', 'Ala', 'Pro', 'Val', 'Le
u', 'Ile', 'Orn', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser', 'Thr', 'Lys', 'Trp', 'Tyr',
```

```
'Meth', 'C0', 'C2', 'C3', 'C4', 'C5-1', 'C5', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6
-DC', 'C10-2', 'C10-1', 'C10', 'C12-1', 'C12', 'C14-2', 'C14-1', 'C14', 'C14-OH',
'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C18', 'C18-1-OH', 'C18-O
H', 'ADMA', 'SDMA', 'Choline']
Cross-validation score for 62 features: 0.9144
Selected features for 63 features: ['Kynurenine', 'Serotonin', 'Quinolinic acid',
'HIAA', 'Tryptamin', 'Antranillic acid', 'Indole-3-lactic acid', 'Indole-3-acetic
acid', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3-propionic ac
id', 'Indole-3-butyric acid', 'Xanturenic acid', 'Kynurenic acid', 'Gly', 'Ala',
'Pro', 'Val', 'Leu', 'Ile', 'Orn', 'Asp', 'Phe', 'Arg', 'Cit', 'Ser', 'Thr', 'Ly
s', 'Trp', 'Tyr', 'Meth', 'C0', 'C2', 'C3', 'C4', 'C5-1', 'C5', 'C6', 'C5-OH', 'C
5-DC', 'C8-1', 'C6-DC', 'C10-2', 'C10-1', 'C10', 'C12-1', 'C12', 'C14-2', 'C14-
1', 'C14', 'C14-OH', 'C16-1', 'C16', 'C16-1-OH', 'C16-OH', 'C18-2', 'C18-1', 'C1
8', 'C18-1-OH', 'C18-OH', 'ADMA', 'SDMA', 'Choline']
Cross-validation score for 63 features: 0.9144

Optimal number of features based on cross-validation: 30
Validation score with 30 features: 0.9738

Selected features for the optimal number of features (30):
['Quinolinic acid', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3
-propionic acid', 'Indole-3-butyric acid', 'Pro', 'Val', 'Asp', 'Arg', 'Cit', 'Th
r', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C1
0-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
```



Step 6 : Union and intersect features selected by Lasso, Random Forest, and RFE Features selected by Lasso

In [20]:
```
# Lasso: Select features with importance > 0
selected_features_lasso = important_features[important_features > 0].index.tolis

# Print the number of features and the selected features list
print(f"\nNumber of features above importance > 0: {len(selected_features_lasso)
print("selected_features_lasso:")
print(selected_features_lasso)
```

```
Number of features above importance > 0: 32
selected_features_lasso:
['ADMA', 'Trp', 'C16-1', 'C14-2', 'C18', 'C2', 'Indole-3-acrylic acid', 'C18-OH',
'Indole-3-carboxaldehyde', 'Lys', 'C6-DC', 'Tyr', 'C5-DC', 'C16', 'C5-1', 'C4',
'C10-1', 'Arg', 'C8-1', 'Val', 'Tryptamin', 'Kynurenine', 'Thr', 'HIAA', 'Cit',
'Indole-3-propionic acid', 'C5-OH', 'Choline', 'Pro', 'Kynurenic acid', 'Indole-3
-lactic acid', 'Quinolinic acid']
```

In [21]:
```
# Features selected by Random Forest
selected_features_rf = top_features_optimal_k.index.tolist()

# Print the number of features and the selected features list
print(f"\nNumber of features selected from random forest: {len(selected_features
print("selected_features_rf:")
print(selected_features_rf)
```

```
Number of features selected from random forest: 31
selected_features_rf:
['ADMA', 'C6-DC', 'C16-1-OH', 'Indole-3-carboxaldehyde', 'Asp', 'Orn', 'Serotoni
n', 'C18-OH', 'Indole-3-acrylic acid', 'Indole-3-propionic acid', 'Tryptamin', 'H
IAA', 'C16-1', 'Quinolinic acid', 'C10-1', 'C18', 'Indole-3-lactic acid', 'Cit',
'SDMA', 'C0', 'C5-OH', 'C3', 'C10-2', 'C12', 'C18-1-OH', 'C8-1', 'C5-DC', 'Trp',
'C14-2', 'Kynurenine', 'C10']
```

In [22]:
```
# Features selected by RFE
selected_features_rfe = optimal_selected_features3

# Print the number of features and the selected features list
print(f"\nNumber of features selected from RFE: {len(selected_features_rfe)}")
print("selected_features_rfe:")
print(selected_features_rfe)
```

```
Number of features selected from RFE: 30
selected_features_rfe:
['Quinolinic acid', 'Indole-3-carboxaldehyde', 'Indole-3-acrylic acid', 'Indole-3
-propionic acid', 'Indole-3-butyric acid', 'Pro', 'Val', 'Asp', 'Arg', 'Cit', 'Th
r', 'Lys', 'Trp', 'Tyr', 'C2', 'C4', 'C6', 'C5-OH', 'C5-DC', 'C8-1', 'C6-DC', 'C1
0-1', 'C12', 'C14-2', 'C16-1', 'C16', 'C18-1', 'C18', 'C18-OH', 'ADMA']
```

In [23]:
```
%pip install matplotlib-venn
```

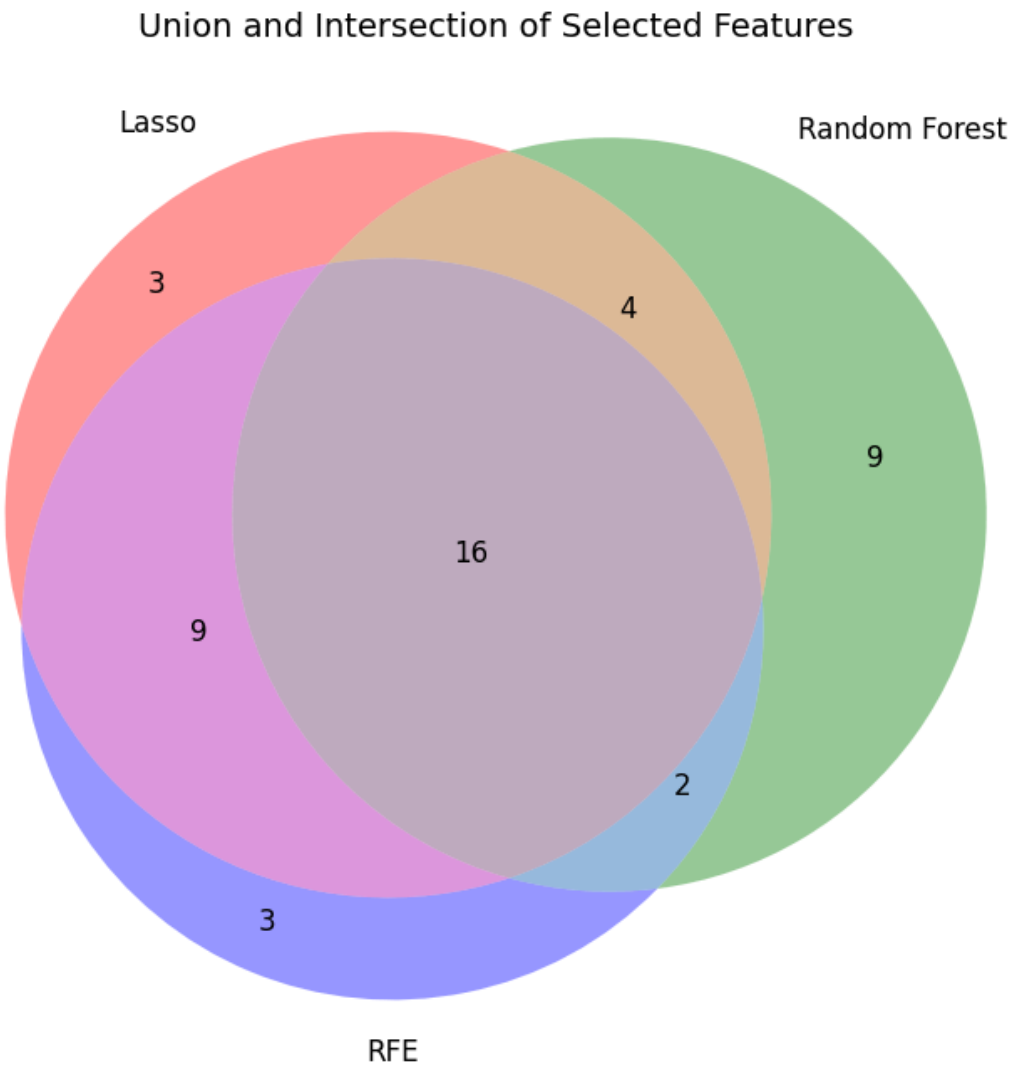
```
Requirement already satisfied: matplotlib-venn in c:\users\behsh\appdata\local\pr
ograms\python\python311\lib\site-packages (1.1.1)
Requirement already satisfied: matplotlib in c:\users\behsh\appdata\local\program
s\python\python311\lib\site-packages (from matplotlib-venn) (3.9.2)
Requirement already satisfied: numpy in c:\users\behsh\appdata\local\programs\pyt
hon\python311\lib\site-packages (from matplotlib-venn) (1.26.4)
Requirement already satisfied: scipy in c:\users\behsh\appdata\local\programs\pyt
hon\python311\lib\site-packages (from matplotlib-venn) (1.14.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\behsh\appdata\local\p
rograms\python\python311\lib\site-packages (from matplotlib->matplotlib-venn) (1.
3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\behsh\appdata\local\progr
ams\python\python311\lib\site-packages (from matplotlib->matplotlib-venn) (0.12.
1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\behsh\appdata\local
\programs\python\python311\lib\site-packages (from matplotlib->matplotlib-venn)
(4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\behsh\appdata\local
\programs\python\python311\lib\site-packages (from matplotlib->matplotlib-venn)
(1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\behsh\appdata\roaming
\python\python311\site-packages (from matplotlib->matplotlib-venn) (23.1)
Requirement already satisfied: pillow>=8 in c:\users\behsh\appdata\local\programs
\python\python311\lib\site-packages (from matplotlib->matplotlib-venn) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\behsh\appdata\local\p
rograms\python\python311\lib\site-packages (from matplotlib->matplotlib-venn) (3.
2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\behsh\appdata\roa
ming\python\python311\site-packages (from matplotlib->matplotlib-venn) (2.9.0.pos
t0)
Requirement already satisfied: six>=1.5 in c:\users\behsh\appdata\roaming\python
\python311\site-packages (from python-dateutil>=2.7->matplotlib->matplotlib-venn)
(1.16.0)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [24]:

```python
from matplotlib_venn import venn3

# Assuming selected_features_lasso, selected_features_rf, selected_features_rfe
# Union: All features selected by any method
union_features = list(set(selected_features_lasso + selected_features_rf + selec

# Intersection: Only features selected by all methods
intersect_features = list(set(selected_features_lasso).intersection(selected_fea

# Print combined and common features
print("Union features (union of all methods):")
print(union_features)
print(f"\nNumber of Union features (union of all methods): {len(union_features)}

print("\nIntersect features (intersection of all methods):")
print(intersect_features)
print(f"\nNumber of Intersect features (intersection of all methods): {len(inter

# Set representation for Venn diagram
set_lasso = set(selected_features_lasso)
set_rf = set(selected_features_rf)
set_rfe = set(selected_features_rfe)
```

```
# Create Venn diagram
plt.figure(figsize=(10, 8))
venn = venn3([set_lasso, set_rf, set_rfe], ('Lasso', 'Random Forest', 'RFE'))

# Highlight intersection and union
for subset_id in ('100', '010', '001', '110', '101', '011', '111'):
    label = venn.get_label_by_id(subset_id)
    if label:  # Check if the subset has a label
        label.set_fontsize(12)

plt.title("Union and Intersection of Selected Features", fontsize=14)
plt.show()
```

```
Union features (union of all methods):
['C16-1-OH', 'Orn', 'C5-1', 'Val', 'C18-1-OH', 'C18', 'Indole-3-carboxaldehyde',
'Indole-3-acrylic acid', 'C6', 'C16-1', 'Tyr', 'Tryptamin', 'HIAA', 'Pro', 'C14-
2', 'C12', 'Cit', 'Kynurenic acid', 'C0', 'Indole-3-propionic acid', 'C8-1', 'Ar
g', 'Serotonin', 'SDMA', 'C18-OH', 'C6-DC', 'Choline', 'Kynurenine', 'ADMA', 'As
p', 'Quinolinic acid', 'C3', 'C10-1', 'Indole-3-lactic acid', 'Thr', 'C18-1', 'C5
-DC', 'C5-OH', 'C10', 'C16', 'Indole-3-butyric acid', 'C10-2', 'Lys', 'C2', 'Tr
p', 'C4']

Number of Union features (union of all methods): 46

Intersect features (intersection of all methods):
['C5-DC', 'C8-1', 'C5-OH', 'C16-1', 'C18-OH', 'C6-DC', 'ADMA', 'C14-2', 'Indole-3
-acrylic acid', 'Cit', 'Quinolinic acid', 'C10-1', 'Indole-3-carboxaldehyde', 'C1
8', 'Trp', 'Indole-3-propionic acid']

Number of Intersect features (intersection of all methods): 16
```



Step 7 : Prepare Data for Model Training

In [25]:
```
from sklearn.model_selection import StratifiedKFold

# Ensure X_train and X_test are DataFrames
X_test_df = pd.DataFrame(X_test, columns=X.columns)

# Prepare X_train and X_test using the selected common features (intersection fe
X_train_selected = X_train_df[intersect_features]
X_test_selected = X_test_df[intersect_features]

# Combine data for K-Fold Cross Validation
X_selected = pd.concat([X_train_selected, X_test_selected])
y_combined = pd.concat([pd.Series(y_train), pd.Series(y_test)])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

Step 8: Model Training and Validation

Using Logistic Regression

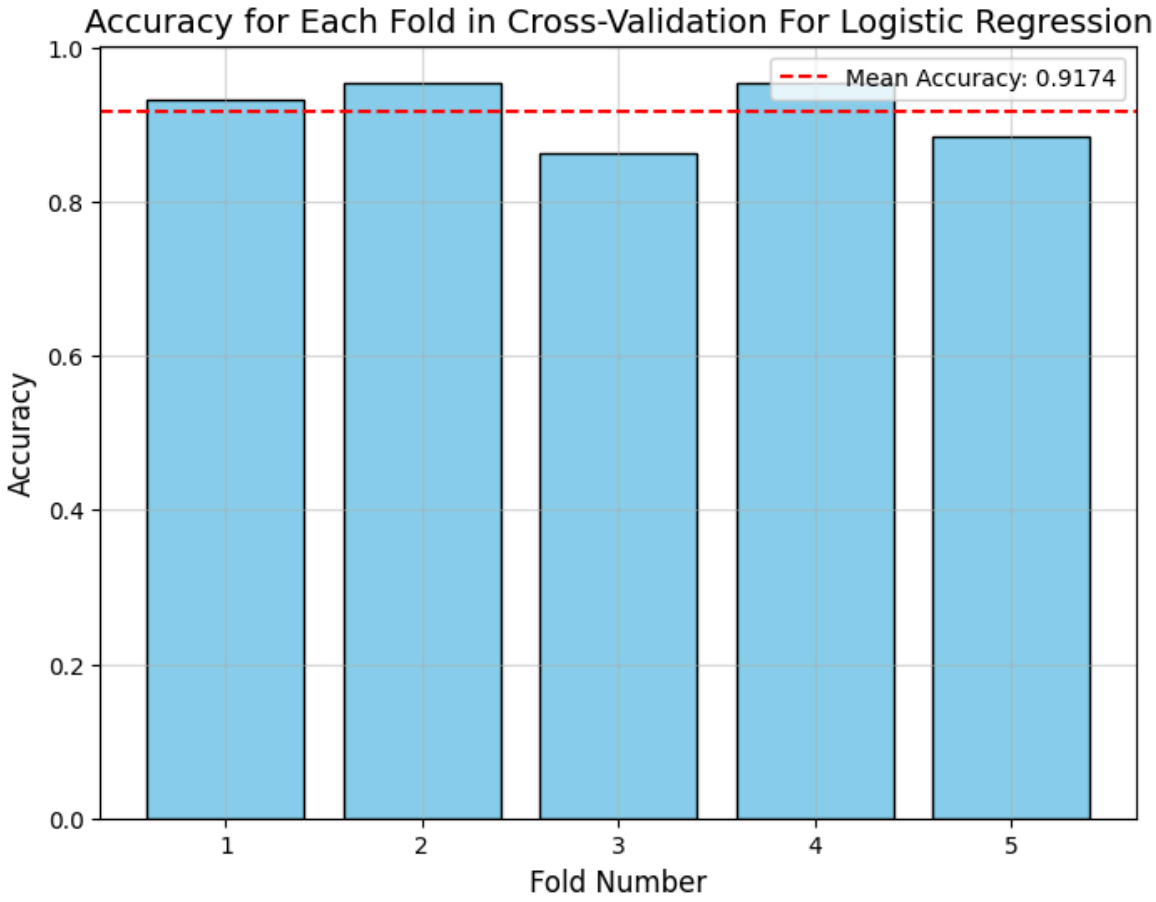
In [26]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_mat

# Initialize the logistic regression model
```

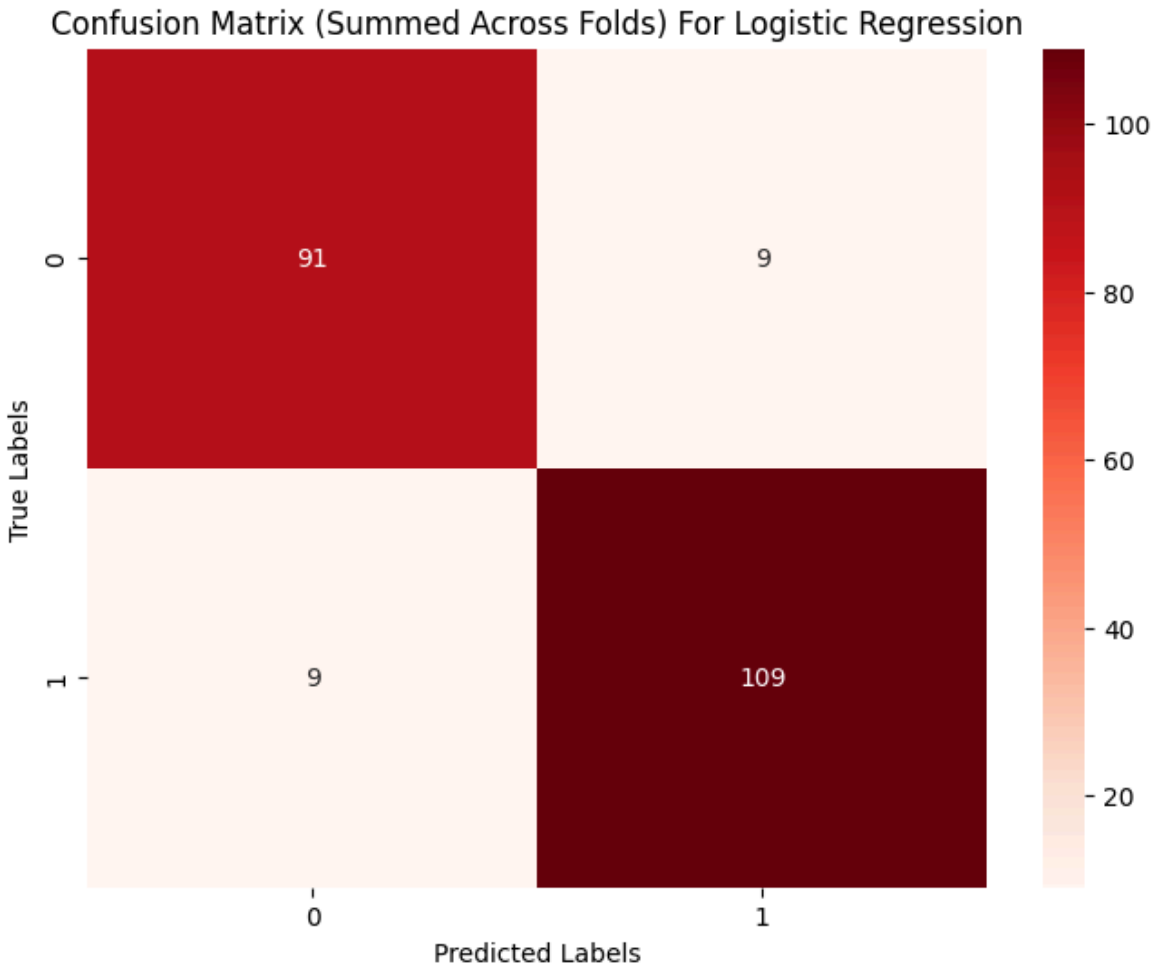
```
lr_model = LogisticRegression(max_iter=1000)

# Store metrics for each fold
fold_accuracies = []
confusion_matrices = []

for fold, (train_idx, val_idx) in enumerate(skf.split(X_selected, y_combined)):
    # Split into training and validation sets
    X_train_fold = X_selected.iloc[train_idx]
    y_train_fold = y_combined.iloc[train_idx]
    X_val_fold = X_selected.iloc[val_idx]
    y_val_fold = y_combined.iloc[val_idx]

    # Train the model
    lr_model.fit(X_train_fold, y_train_fold)

    # Predict on the validation set
    y_val_pred = lr_model.predict(X_val_fold)

    # Calculate accuracy
    accuracy = accuracy_score(y_val_fold, y_val_pred)
    fold_accuracies.append(accuracy)

    # Store confusion matrix
    conf_matrix = confusion_matrix(y_val_fold, y_val_pred)
    confusion_matrices.append(conf_matrix)

    # Print metrics for each fold
    print(f"Fold {fold + 1} Accuracy: {accuracy:.4f}")
    print(f"Classification Report for Fold {fold + 1}:\n")
    print(classification_report(y_val_fold, y_val_pred))
    print("-" * 50)

# Average accuracy across folds
mean_accuracy = sum(fold_accuracies) / len(fold_accuracies)
print(f"Mean Accuracy Across Folds: {mean_accuracy:.4f}")

# Visualize fold accuracies
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(fold_accuracies) + 1), fold_accuracies, color='skyblue', ed
plt.axhline(mean_accuracy, color='red', linestyle='--', label=f'Mean Accuracy: {
plt.xticks(range(1, len(fold_accuracies) + 1))
plt.title("Accuracy for Each Fold in Cross-Validation For Logistic Regression",
plt.xlabel("Fold Number", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.show()

# Plot combined confusion matrix
sum_conf_matrix = sum(confusion_matrices)
plt.figure(figsize=(8, 6))
sns.heatmap(
    sum_conf_matrix,
    annot=True,
    fmt="d",
    cmap="Reds",
    xticklabels=lr_model.classes_,
    yticklabels=lr_model.classes_,
)
```

```
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix (Summed Across Folds) For Logistic Regression")
plt.show()
```

```
Fold 1 Accuracy: 0.9318
Classification Report for Fold 1:

              precision    recall  f1-score   support

           0       0.95      0.90      0.92        20
           1       0.92      0.96      0.94        24

    accuracy                           0.93        44
   macro avg       0.93      0.93      0.93        44
weighted avg       0.93      0.93      0.93        44

--------------------------------------------------
Fold 2 Accuracy: 0.9545
Classification Report for Fold 2:

              precision    recall  f1-score   support

           0       1.00      0.90      0.95        20
           1       0.92      1.00      0.96        24

    accuracy                           0.95        44
   macro avg       0.96      0.95      0.95        44
weighted avg       0.96      0.95      0.95        44

--------------------------------------------------
Fold 3 Accuracy: 0.8636
Classification Report for Fold 3:

              precision    recall  f1-score   support

           0       0.79      0.95      0.86        20
           1       0.95      0.79      0.86        24

    accuracy                           0.86        44
   macro avg       0.87      0.87      0.86        44
weighted avg       0.88      0.86      0.86        44

--------------------------------------------------
Fold 4 Accuracy: 0.9535
Classification Report for Fold 4:

              precision    recall  f1-score   support

           0       0.91      1.00      0.95        20
           1       1.00      0.91      0.95        23

    accuracy                           0.95        43
   macro avg       0.95      0.96      0.95        43
weighted avg       0.96      0.95      0.95        43

--------------------------------------------------
Fold 5 Accuracy: 0.8837
Classification Report for Fold 5:

              precision    recall  f1-score   support

           0       0.94      0.80      0.86        20
           1       0.85      0.96      0.90        23
```

```
              accuracy                           0.88        43
             macro avg       0.89      0.88      0.88        43
          weighted avg       0.89      0.88      0.88        43

--------------------------------------------------
Mean Accuracy Across Folds: 0.9174
```





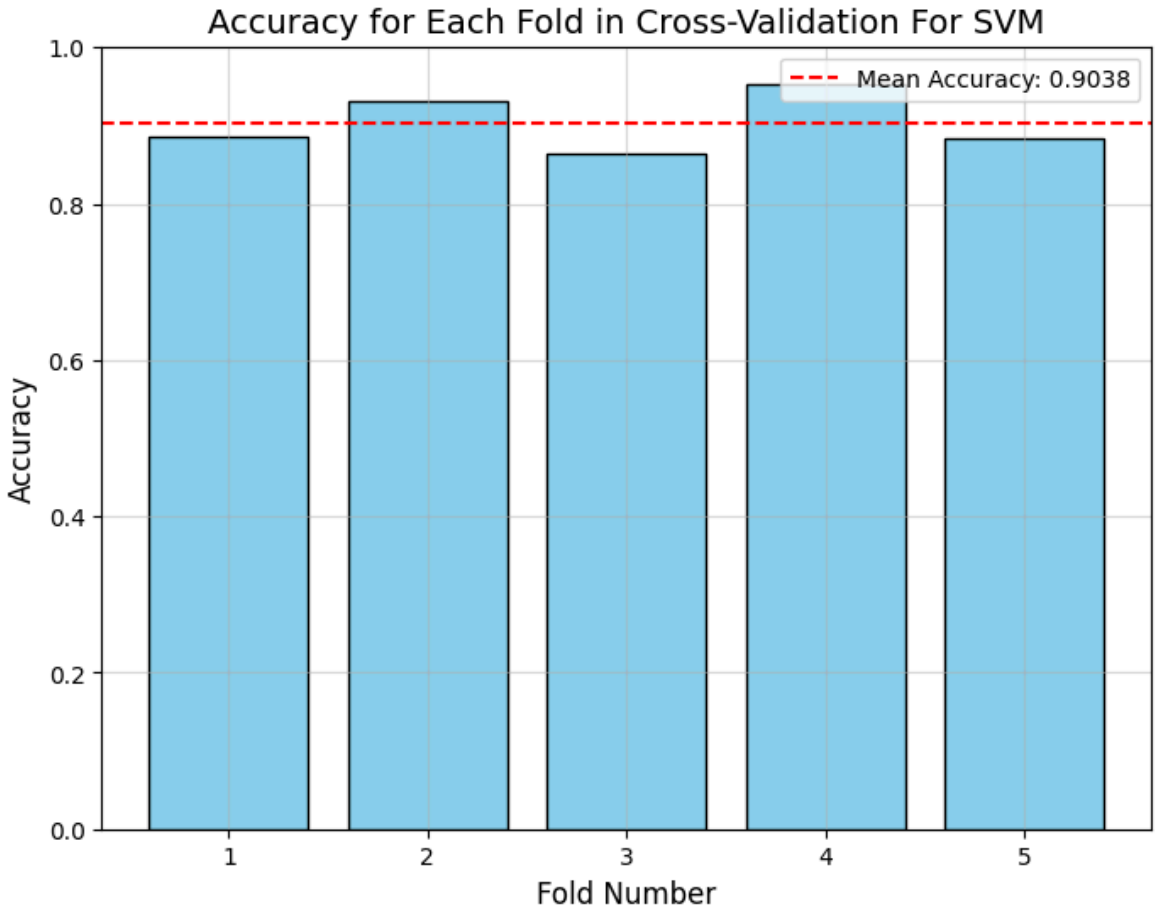
Using Support Vector Machine (SVM)

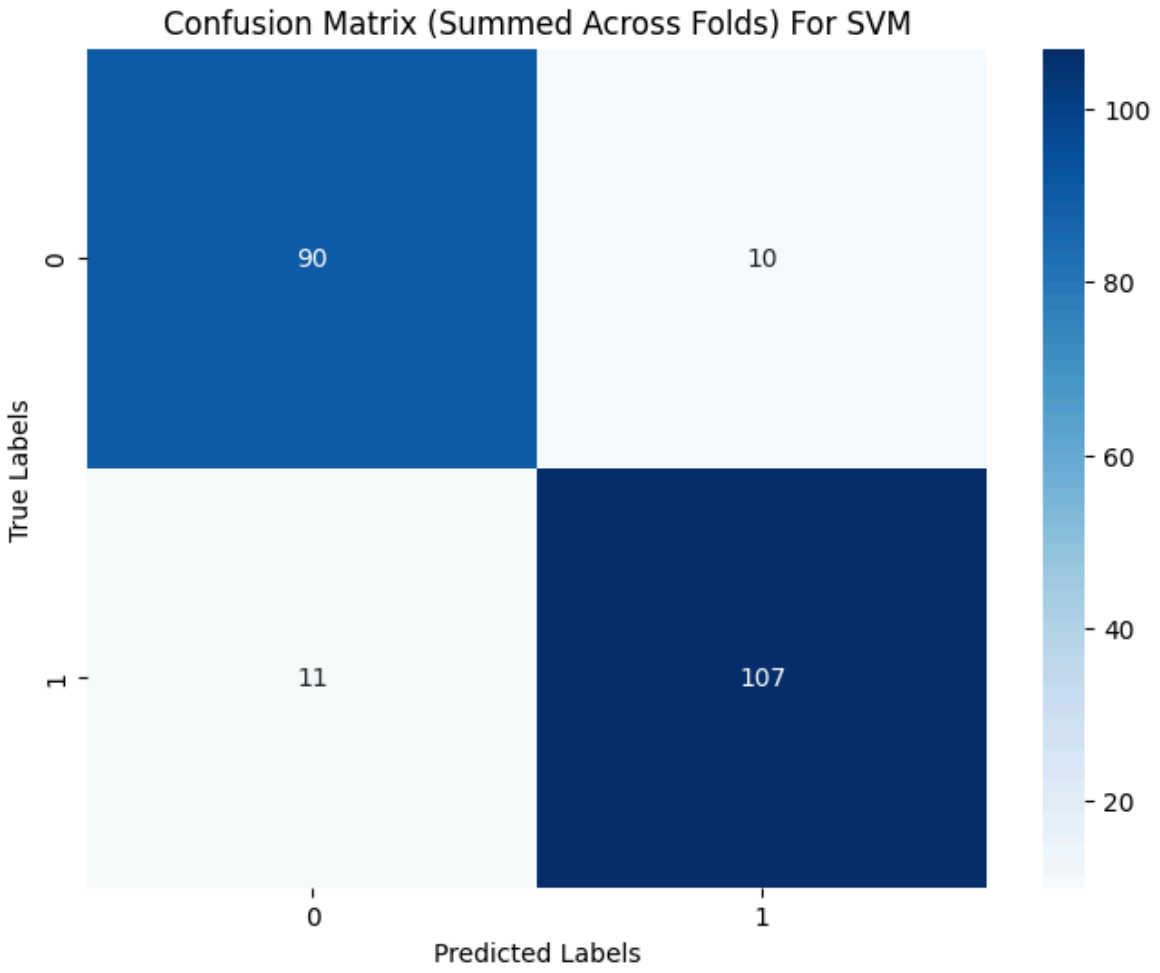
In [27]:
```
from sklearn.svm import SVC

# Initialize the SVM model
svm_model = SVC(kernel='linear', C=1.0, random_state=42)  # You can experiment w

# Store metrics for each fold
fold_accuracies = []
confusion_matrices = []

for fold, (train_idx, val_idx) in enumerate(skf.split(X_selected, y_combined)):
    # Split into training and validation sets
    X_train_fold = X_selected.iloc[train_idx]
    y_train_fold = y_combined.iloc[train_idx]
    X_val_fold = X_selected.iloc[val_idx]
    y_val_fold = y_combined.iloc[val_idx]

    # Train the model
    svm_model.fit(X_train_fold, y_train_fold)

    # Predict on the validation set
    y_val_pred = svm_model.predict(X_val_fold)

    # Calculate accuracy
    accuracy = accuracy_score(y_val_fold, y_val_pred)
    fold_accuracies.append(accuracy)

    # Store confusion matrix
```

```
    conf_matrix = confusion_matrix(y_val_fold, y_val_pred)
    confusion_matrices.append(conf_matrix)

    # Print metrics for each fold
    print(f"Fold {fold + 1} Accuracy: {accuracy:.4f}")
    print(f"Classification Report for Fold {fold + 1}:\n")
    print(classification_report(y_val_fold, y_val_pred))
    print("-" * 50)

# Average accuracy across folds
mean_accuracy = sum(fold_accuracies) / len(fold_accuracies)
print(f"Mean Accuracy Across Folds: {mean_accuracy:.4f}")

# Visualize fold accuracies
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(fold_accuracies) + 1), fold_accuracies, color='skyblue', ed
plt.axhline(mean_accuracy, color='red', linestyle='--', label=f'Mean Accuracy: {
plt.xticks(range(1, len(fold_accuracies) + 1))
plt.title("Accuracy for Each Fold in Cross-Validation For SVM", fontsize=14)
plt.xlabel("Fold Number", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.show()

# Plot combined confusion matrix
sum_conf_matrix = sum(confusion_matrices)
plt.figure(figsize=(8, 6))
sns.heatmap(
    sum_conf_matrix,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=svm_model.classes_,
    yticklabels=svm_model.classes_,
)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix (Summed Across Folds) For SVM")
plt.show()
```

```
Fold 1 Accuracy: 0.8864
Classification Report for Fold 1:

              precision    recall  f1-score   support

           0       0.89      0.85      0.87        20
           1       0.88      0.92      0.90        24

    accuracy                           0.89        44
   macro avg       0.89      0.88      0.88        44
weighted avg       0.89      0.89      0.89        44

--------------------------------------------------
Fold 2 Accuracy: 0.9318
Classification Report for Fold 2:

              precision    recall  f1-score   support

           0       0.95      0.90      0.92        20
           1       0.92      0.96      0.94        24

    accuracy                           0.93        44
   macro avg       0.93      0.93      0.93        44
weighted avg       0.93      0.93      0.93        44

--------------------------------------------------
Fold 3 Accuracy: 0.8636
Classification Report for Fold 3:

              precision    recall  f1-score   support

           0       0.79      0.95      0.86        20
           1       0.95      0.79      0.86        24

    accuracy                           0.86        44
   macro avg       0.87      0.87      0.86        44
weighted avg       0.88      0.86      0.86        44

--------------------------------------------------
Fold 4 Accuracy: 0.9535
Classification Report for Fold 4:

              precision    recall  f1-score   support

           0       0.91      1.00      0.95        20
           1       1.00      0.91      0.95        23

    accuracy                           0.95        43
   macro avg       0.95      0.96      0.95        43
weighted avg       0.96      0.95      0.95        43

--------------------------------------------------
Fold 5 Accuracy: 0.8837
Classification Report for Fold 5:

              precision    recall  f1-score   support

           0       0.94      0.80      0.86        20
           1       0.85      0.96      0.90        23
```

```
    accuracy                           0.88        43
   macro avg       0.89      0.88      0.88        43
weighted avg       0.89      0.88      0.88        43

--------------------------------------------------
Mean Accuracy Across Folds: 0.9038
```





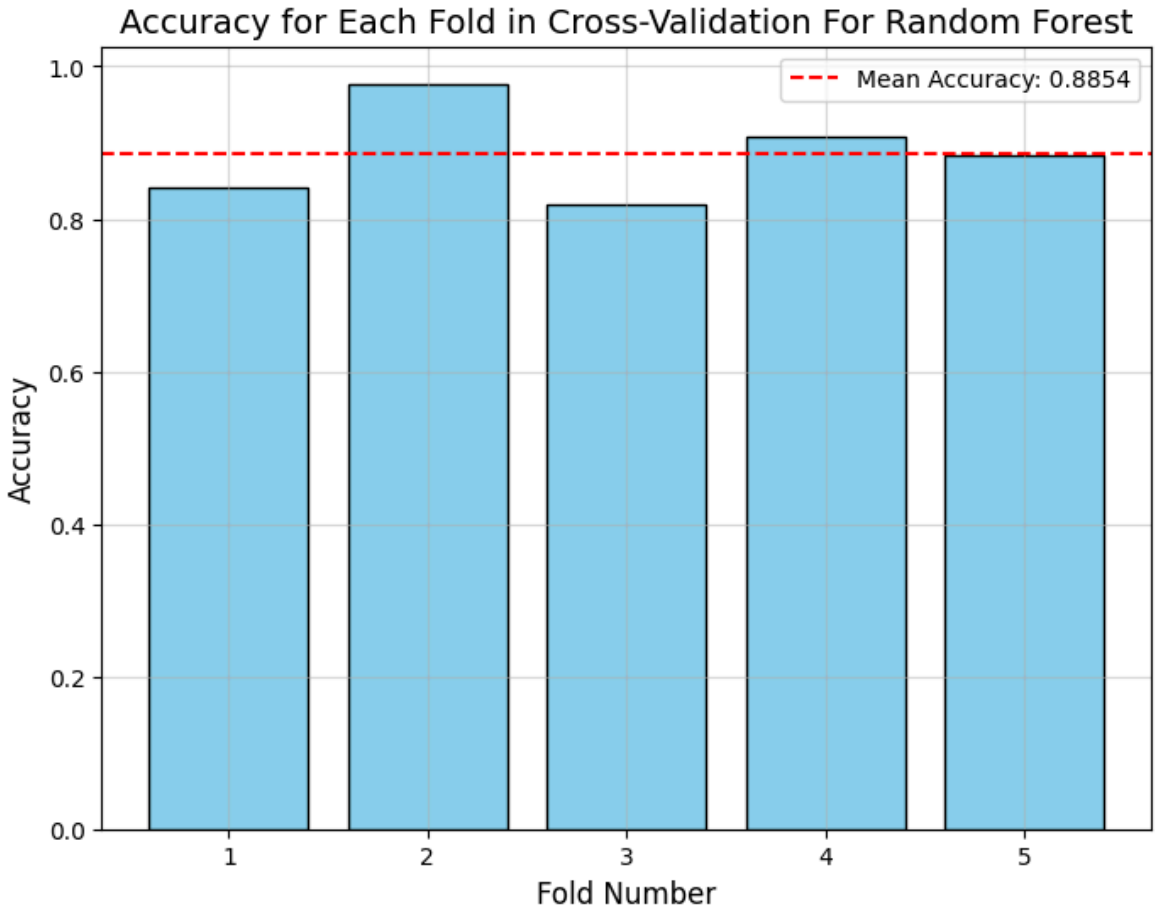
Using Random Forest Classifier

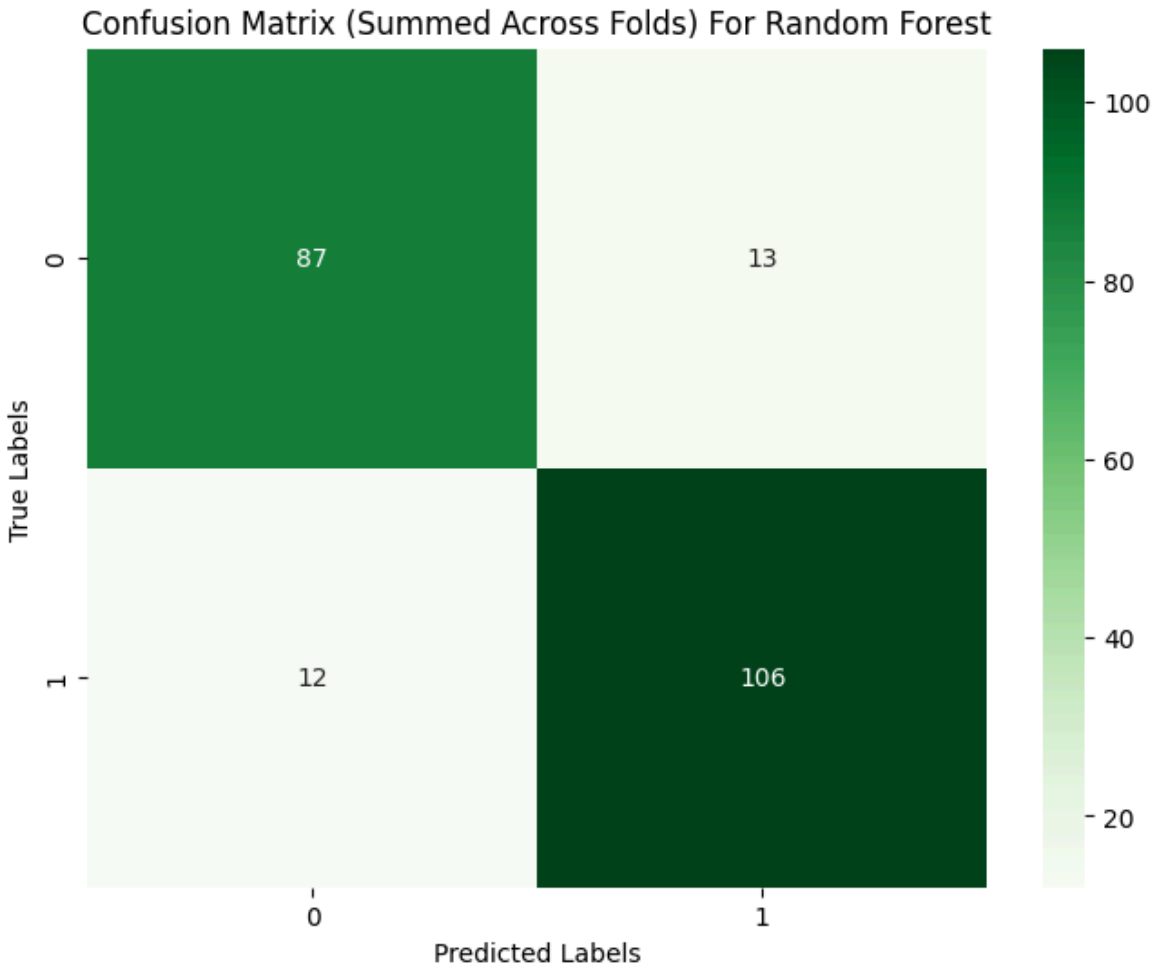
In [28]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Store metrics for each fold
fold_accuracies = []
confusion_matrices = []

for fold, (train_idx, val_idx) in enumerate(skf.split(X_selected, y_combined)):
    # Split into training and validation sets
    X_train_fold = X_selected.iloc[train_idx]
    y_train_fold = y_combined.iloc[train_idx]
    X_val_fold = X_selected.iloc[val_idx]
    y_val_fold = y_combined.iloc[val_idx]

    # Train the model
    rf_model.fit(X_train_fold, y_train_fold)

    # Predict on the validation set
    y_val_pred = rf_model.predict(X_val_fold)

    # Calculate accuracy
    accuracy = accuracy_score(y_val_fold, y_val_pred)
    fold_accuracies.append(accuracy)

    # Store confusion matrix
```

```
    conf_matrix = confusion_matrix(y_val_fold, y_val_pred)
    confusion_matrices.append(conf_matrix)

    # Print metrics for each fold
    print(f"Fold {fold + 1} Accuracy: {accuracy:.4f}")
    print(f"Classification Report for Fold {fold + 1}:\n")
    print(classification_report(y_val_fold, y_val_pred))
    print("-" * 50)

# Average accuracy across folds
mean_accuracy = sum(fold_accuracies) / len(fold_accuracies)
print(f"Mean Accuracy Across Folds: {mean_accuracy:.4f}")

# Visualize fold accuracies
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(fold_accuracies) + 1), fold_accuracies, color='skyblue', ed
plt.axhline(mean_accuracy, color='red', linestyle='--', label=f'Mean Accuracy: {
plt.xticks(range(1, len(fold_accuracies) + 1))
plt.title("Accuracy for Each Fold in Cross-Validation For Random Forest", fontsi
plt.xlabel("Fold Number", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.show()

# Plot combined confusion matrix
sum_conf_matrix = sum(confusion_matrices)
plt.figure(figsize=(8, 6))
sns.heatmap(
    sum_conf_matrix,
    annot=True,
    fmt="d",
    cmap="Greens",
    xticklabels=rf_model.classes_,
    yticklabels=rf_model.classes_,
)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix (Summed Across Folds) For Random Forest")
plt.show()
```

```
Fold 1 Accuracy: 0.8409
Classification Report for Fold 1:

              precision    recall  f1-score   support

           0       0.84      0.80      0.82        20
           1       0.84      0.88      0.86        24

    accuracy                           0.84        44
   macro avg       0.84      0.84      0.84        44
weighted avg       0.84      0.84      0.84        44

--------------------------------------------------
Fold 2 Accuracy: 0.9773
Classification Report for Fold 2:

              precision    recall  f1-score   support

           0       1.00      0.95      0.97        20
           1       0.96      1.00      0.98        24

    accuracy                           0.98        44
   macro avg       0.98      0.97      0.98        44
weighted avg       0.98      0.98      0.98        44

--------------------------------------------------
Fold 3 Accuracy: 0.8182
Classification Report for Fold 3:

              precision    recall  f1-score   support

           0       0.80      0.80      0.80        20
           1       0.83      0.83      0.83        24

    accuracy                           0.82        44
   macro avg       0.82      0.82      0.82        44
weighted avg       0.82      0.82      0.82        44

--------------------------------------------------
Fold 4 Accuracy: 0.9070
Classification Report for Fold 4:

              precision    recall  f1-score   support

           0       0.86      0.95      0.90        20
           1       0.95      0.87      0.91        23

    accuracy                           0.91        43
   macro avg       0.91      0.91      0.91        43
weighted avg       0.91      0.91      0.91        43

--------------------------------------------------
Fold 5 Accuracy: 0.8837
Classification Report for Fold 5:

              precision    recall  f1-score   support

           0       0.89      0.85      0.87        20
           1       0.88      0.91      0.89        23
```

```
              accuracy                           0.88        43
             macro avg       0.88      0.88      0.88        43
          weighted avg       0.88      0.88      0.88        43

--------------------------------------------------
Mean Accuracy Across Folds: 0.8854
```





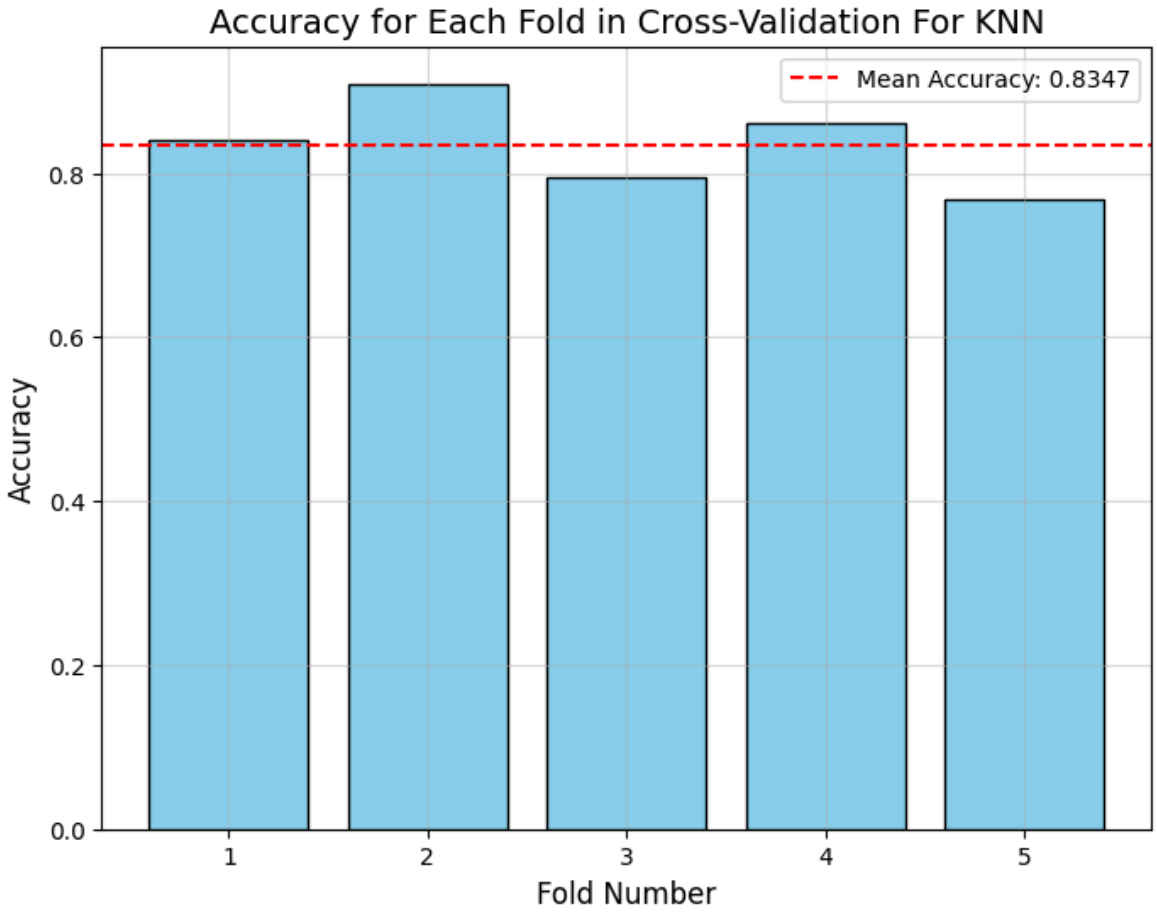
Using K-Nearest Neighbors (KNN)

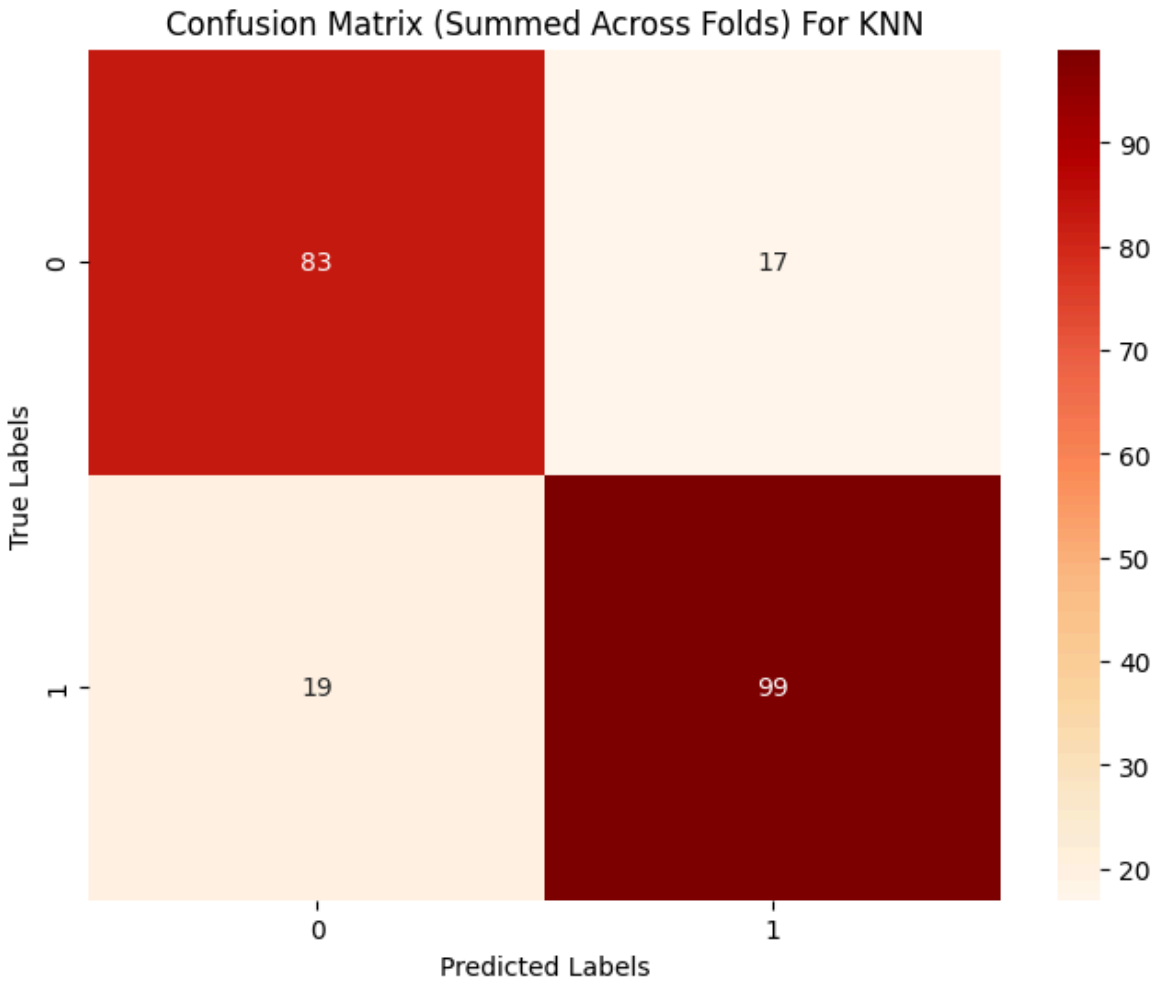
In [29]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=5)

# Store metrics for each fold
fold_accuracies = []
confusion_matrices = []

for fold, (train_idx, val_idx) in enumerate(skf.split(X_selected, y_combined)):
    # Split into training and validation sets
    X_train_fold = X_selected.iloc[train_idx]
    y_train_fold = y_combined.iloc[train_idx]
    X_val_fold = X_selected.iloc[val_idx]
    y_val_fold = y_combined.iloc[val_idx]

    # Train the model
    knn_model.fit(X_train_fold, y_train_fold)

    # Predict on the validation set
    y_val_pred = knn_model.predict(X_val_fold)

    # Calculate accuracy
    accuracy = accuracy_score(y_val_fold, y_val_pred)
    fold_accuracies.append(accuracy)

    # Store confusion matrix
    conf_matrix = confusion_matrix(y_val_fold, y_val_pred)
```

```
    confusion_matrices.append(conf_matrix)

    # Print metrics for each fold
    print(f"Fold {fold + 1} Accuracy: {accuracy:.4f}")
    print(f"Classification Report for Fold {fold + 1}:\n")
    print(classification_report(y_val_fold, y_val_pred))
    print("-" * 50)

# Average accuracy across folds
mean_accuracy = sum(fold_accuracies) / len(fold_accuracies)
print(f"Mean Accuracy Across Folds: {mean_accuracy:.4f}")

# Visualize fold accuracies
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(fold_accuracies) + 1), fold_accuracies, color='skyblue', ed
plt.axhline(mean_accuracy, color='red', linestyle='--', label=f'Mean Accuracy: {
plt.xticks(range(1, len(fold_accuracies) + 1))
plt.title("Accuracy for Each Fold in Cross-Validation For KNN", fontsize=14)
plt.xlabel("Fold Number", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.show()

# Plot combined confusion matrix
sum_conf_matrix = sum(confusion_matrices)
plt.figure(figsize=(8, 6))
sns.heatmap(
    sum_conf_matrix,
    annot=True,
    fmt="d",
    cmap="OrRd",
    xticklabels=knn_model.classes_,
    yticklabels=knn_model.classes_,
)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix (Summed Across Folds) For KNN")
plt.show()
```

```
Fold 1 Accuracy: 0.8409
Classification Report for Fold 1:

              precision    recall  f1-score   support

           0       0.81      0.85      0.83        20
           1       0.87      0.83      0.85        24

    accuracy                           0.84        44
   macro avg       0.84      0.84      0.84        44
weighted avg       0.84      0.84      0.84        44

--------------------------------------------------
Fold 2 Accuracy: 0.9091
Classification Report for Fold 2:

              precision    recall  f1-score   support

           0       0.94      0.85      0.89        20
           1       0.88      0.96      0.92        24

    accuracy                           0.91        44
   macro avg       0.91      0.90      0.91        44
weighted avg       0.91      0.91      0.91        44

--------------------------------------------------
Fold 3 Accuracy: 0.7955
Classification Report for Fold 3:

              precision    recall  f1-score   support

           0       0.74      0.85      0.79        20
           1       0.86      0.75      0.80        24

    accuracy                           0.80        44
   macro avg       0.80      0.80      0.80        44
weighted avg       0.80      0.80      0.80        44

--------------------------------------------------
Fold 4 Accuracy: 0.8605
Classification Report for Fold 4:

              precision    recall  f1-score   support

           0       0.85      0.85      0.85        20
           1       0.87      0.87      0.87        23

    accuracy                           0.86        43
   macro avg       0.86      0.86      0.86        43
weighted avg       0.86      0.86      0.86        43

--------------------------------------------------
Fold 5 Accuracy: 0.7674
Classification Report for Fold 5:

              precision    recall  f1-score   support

           0       0.75      0.75      0.75        20
           1       0.78      0.78      0.78        23
```

```
    accuracy                           0.77        43
   macro avg       0.77      0.77      0.77        43
weighted avg       0.77      0.77      0.77        43

--------------------------------------------------
Mean Accuracy Across Folds: 0.8347
```





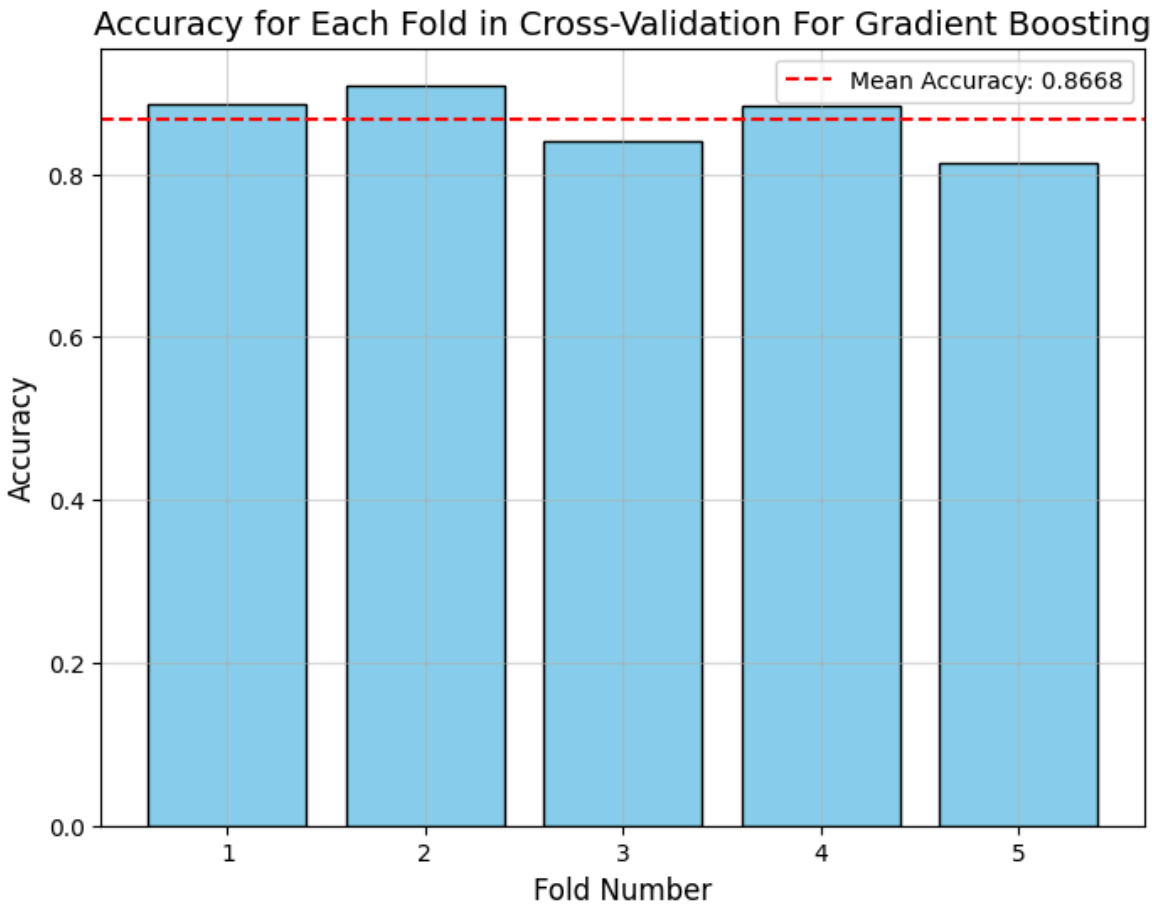
Using Gradient Boosting Classifier

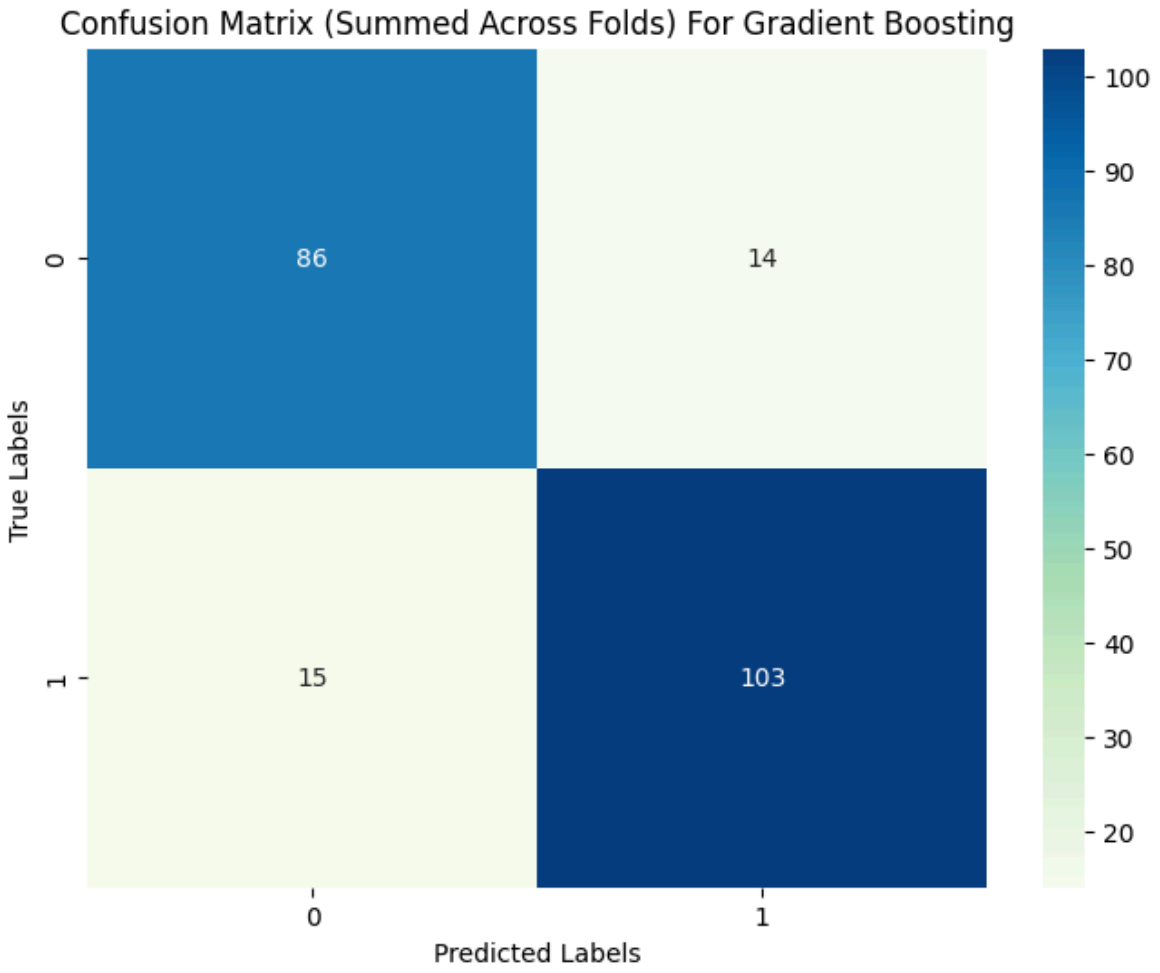
In [30]:
```
from sklearn.ensemble import GradientBoostingClassifier

gb_model = GradientBoostingClassifier(random_state=42)

# Store metrics for each fold
fold_accuracies = []
confusion_matrices = []

for fold, (train_idx, val_idx) in enumerate(skf.split(X_selected, y_combined)):
    # Split into training and validation sets
    X_train_fold = X_selected.iloc[train_idx]
    y_train_fold = y_combined.iloc[train_idx]
    X_val_fold = X_selected.iloc[val_idx]
    y_val_fold = y_combined.iloc[val_idx]

    # Train the model
    gb_model.fit(X_train_fold, y_train_fold)

    # Predict on the validation set
    y_val_pred = gb_model.predict(X_val_fold)

    # Calculate accuracy
    accuracy = accuracy_score(y_val_fold, y_val_pred)
    fold_accuracies.append(accuracy)

    # Store confusion matrix
    conf_matrix = confusion_matrix(y_val_fold, y_val_pred)
```

```
    confusion_matrices.append(conf_matrix)

    # Print metrics for each fold
    print(f"Fold {fold + 1} Accuracy: {accuracy:.4f}")
    print(f"Classification Report for Fold {fold + 1}:\n")
    print(classification_report(y_val_fold, y_val_pred))
    print("-" * 50)

# Average accuracy across folds
mean_accuracy = sum(fold_accuracies) / len(fold_accuracies)
print(f"Mean Accuracy Across Folds: {mean_accuracy:.4f}")

# Visualize fold accuracies
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(fold_accuracies) + 1), fold_accuracies, color='skyblue', ed
plt.axhline(mean_accuracy, color='red', linestyle='--', label=f'Mean Accuracy: {
plt.xticks(range(1, len(fold_accuracies) + 1))
plt.title("Accuracy for Each Fold in Cross-Validation For Gradient Boosting", fo
plt.xlabel("Fold Number", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.show()

# Plot combined confusion matrix
sum_conf_matrix = sum(confusion_matrices)
plt.figure(figsize=(8, 6))
sns.heatmap(
    sum_conf_matrix,
    annot=True,
    fmt="d",
    cmap="GnBu",
    xticklabels=gb_model.classes_,
    yticklabels=gb_model.classes_,
)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix (Summed Across Folds) For Gradient Boosting")
plt.show()
```

```
Fold 1 Accuracy: 0.8864
Classification Report for Fold 1:

              precision    recall  f1-score   support

           0       0.86      0.90      0.88        20
           1       0.91      0.88      0.89        24

    accuracy                           0.89        44
   macro avg       0.89      0.89      0.89        44
weighted avg       0.89      0.89      0.89        44

--------------------------------------------------
Fold 2 Accuracy: 0.9091
Classification Report for Fold 2:

              precision    recall  f1-score   support

           0       0.90      0.90      0.90        20
           1       0.92      0.92      0.92        24

    accuracy                           0.91        44
   macro avg       0.91      0.91      0.91        44
weighted avg       0.91      0.91      0.91        44

--------------------------------------------------
Fold 3 Accuracy: 0.8409
Classification Report for Fold 3:

              precision    recall  f1-score   support

           0       0.81      0.85      0.83        20
           1       0.87      0.83      0.85        24

    accuracy                           0.84        44
   macro avg       0.84      0.84      0.84        44
weighted avg       0.84      0.84      0.84        44

--------------------------------------------------
Fold 4 Accuracy: 0.8837
Classification Report for Fold 4:

              precision    recall  f1-score   support

           0       0.86      0.90      0.88        20
           1       0.91      0.87      0.89        23

    accuracy                           0.88        43
   macro avg       0.88      0.88      0.88        43
weighted avg       0.88      0.88      0.88        43

--------------------------------------------------
Fold 5 Accuracy: 0.8140
Classification Report for Fold 5:

              precision    recall  f1-score   support

           0       0.83      0.75      0.79        20
           1       0.80      0.87      0.83        23
```

```
              accuracy                           0.81        43
             macro avg       0.82      0.81      0.81        43
          weighted avg       0.82      0.81      0.81        43

--------------------------------------------------
Mean Accuracy Across Folds: 0.8668
```





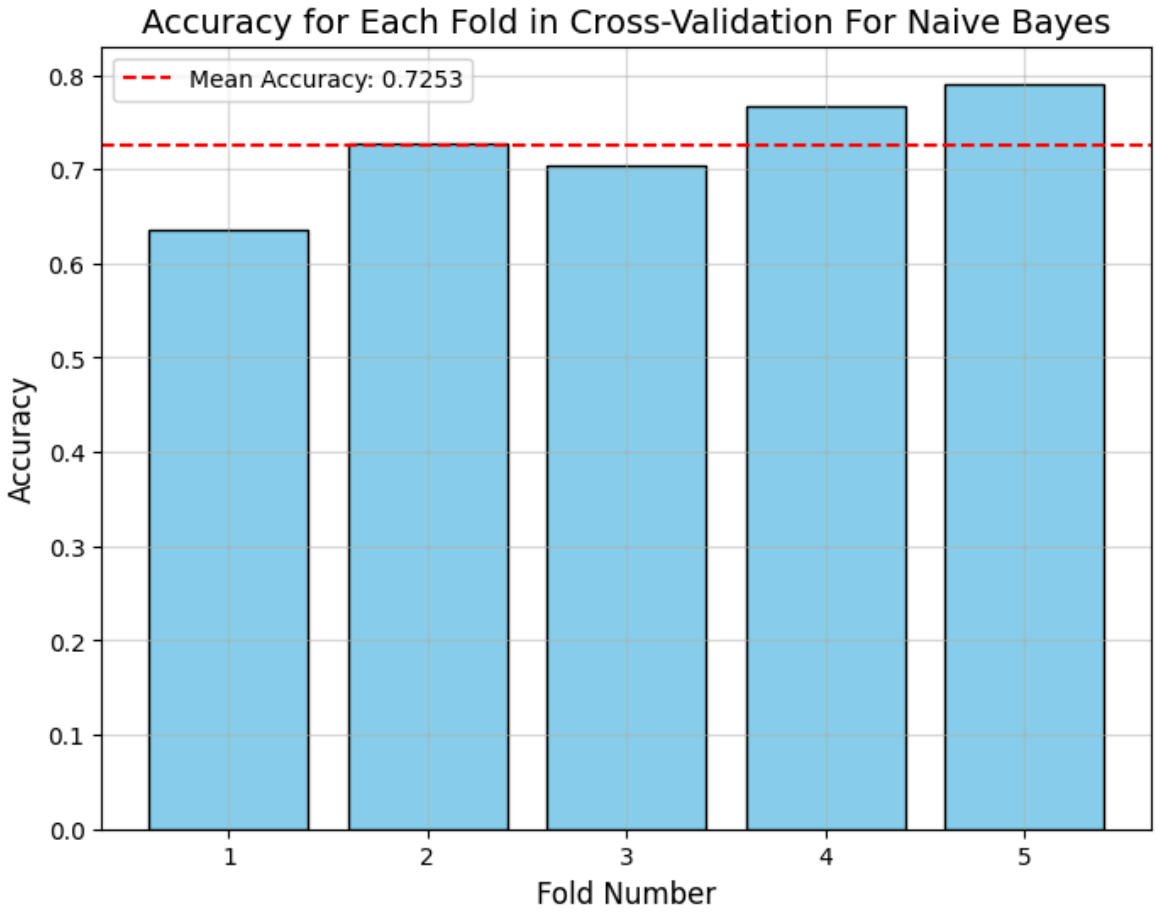
Using Naive Bayes Classifier

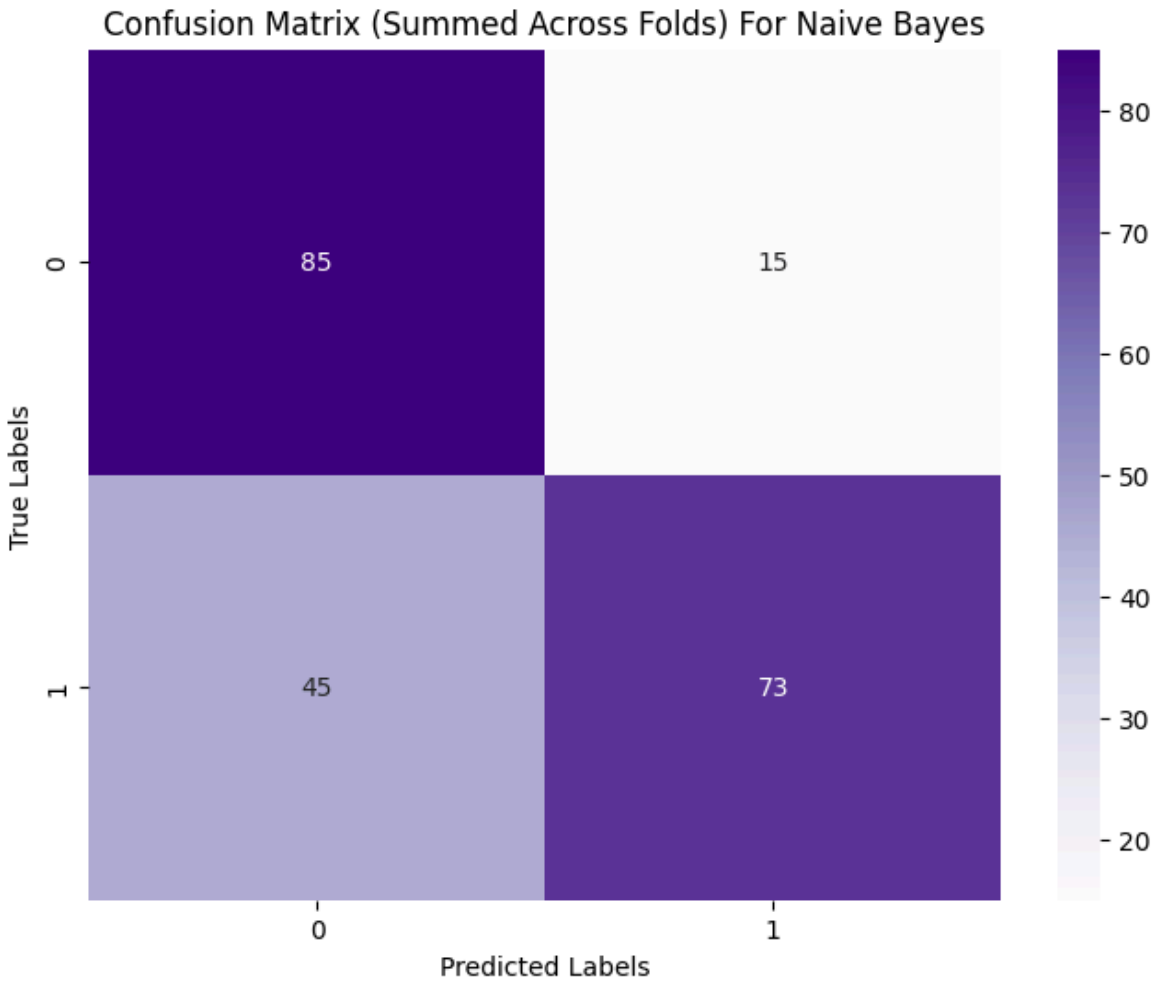
In [31]:
```
from sklearn.naive_bayes import GaussianNB

nb_model = GaussianNB()

# Store metrics for each fold
fold_accuracies = []
confusion_matrices = []

for fold, (train_idx, val_idx) in enumerate(skf.split(X_selected, y_combined)):
    # Split into training and validation sets
    X_train_fold = X_selected.iloc[train_idx]
    y_train_fold = y_combined.iloc[train_idx]
    X_val_fold = X_selected.iloc[val_idx]
    y_val_fold = y_combined.iloc[val_idx]

    # Train the model
    nb_model.fit(X_train_fold, y_train_fold)

    # Predict on the validation set
    y_val_pred = nb_model.predict(X_val_fold)

    # Calculate accuracy
    accuracy = accuracy_score(y_val_fold, y_val_pred)
    fold_accuracies.append(accuracy)

    # Store confusion matrix
    conf_matrix = confusion_matrix(y_val_fold, y_val_pred)
```

```
    confusion_matrices.append(conf_matrix)

    # Print metrics for each fold
    print(f"Fold {fold + 1} Accuracy: {accuracy:.4f}")
    print(f"Classification Report for Fold {fold + 1}:\n")
    print(classification_report(y_val_fold, y_val_pred))
    print("-" * 50)

# Average accuracy across folds
mean_accuracy = sum(fold_accuracies) / len(fold_accuracies)
print(f"Mean Accuracy Across Folds: {mean_accuracy:.4f}")

# Visualize fold accuracies
plt.figure(figsize=(8, 6))
plt.bar(range(1, len(fold_accuracies) + 1), fold_accuracies, color='skyblue', ed
plt.axhline(mean_accuracy, color='red', linestyle='--', label=f'Mean Accuracy: {
plt.xticks(range(1, len(fold_accuracies) + 1))
plt.title("Accuracy for Each Fold in Cross-Validation For Naive Bayes", fontsize
plt.xlabel("Fold Number", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.show()

# Plot combined confusion matrix
sum_conf_matrix = sum(confusion_matrices)
plt.figure(figsize=(8, 6))
sns.heatmap(
    sum_conf_matrix,
    annot=True,
    fmt="d",
    cmap="Purples",
    xticklabels=nb_model.classes_,
    yticklabels=nb_model.classes_,
)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix (Summed Across Folds) For Naive Bayes")
plt.show()
```

```
Fold 1 Accuracy: 0.6364
Classification Report for Fold 1:

              precision    recall  f1-score   support

           0       0.58      0.75      0.65        20
           1       0.72      0.54      0.62        24

    accuracy                           0.64        44
   macro avg       0.65      0.65      0.64        44
weighted avg       0.66      0.64      0.63        44

--------------------------------------------------
Fold 2 Accuracy: 0.7273
Classification Report for Fold 2:

              precision    recall  f1-score   support

           0       0.65      0.85      0.74        20
           1       0.83      0.62      0.71        24

    accuracy                           0.73        44
   macro avg       0.74      0.74      0.73        44
weighted avg       0.75      0.73      0.73        44

--------------------------------------------------
Fold 3 Accuracy: 0.7045
Classification Report for Fold 3:

              precision    recall  f1-score   support

           0       0.63      0.85      0.72        20
           1       0.82      0.58      0.68        24

    accuracy                           0.70        44
   macro avg       0.73      0.72      0.70        44
weighted avg       0.74      0.70      0.70        44

--------------------------------------------------
Fold 4 Accuracy: 0.7674
Classification Report for Fold 4:

              precision    recall  f1-score   support

           0       0.69      0.90      0.78        20
           1       0.88      0.65      0.75        23

    accuracy                           0.77        43
   macro avg       0.79      0.78      0.77        43
weighted avg       0.79      0.77      0.77        43

--------------------------------------------------
Fold 5 Accuracy: 0.7907
Classification Report for Fold 5:

              precision    recall  f1-score   support

           0       0.72      0.90      0.80        20
           1       0.89      0.70      0.78        23
```

```
    accuracy                           0.79        43
   macro avg       0.80      0.80      0.79        43
weighted avg       0.81      0.79      0.79        43

--------------------------------------------------
Mean Accuracy Across Folds: 0.7253
```





Step 9 : Save the trained model

In [32]:
```
import pickle

with open('logistic_regression_model.pkl', 'wb') as file:
    pickle.dump(lr_model, file)
```