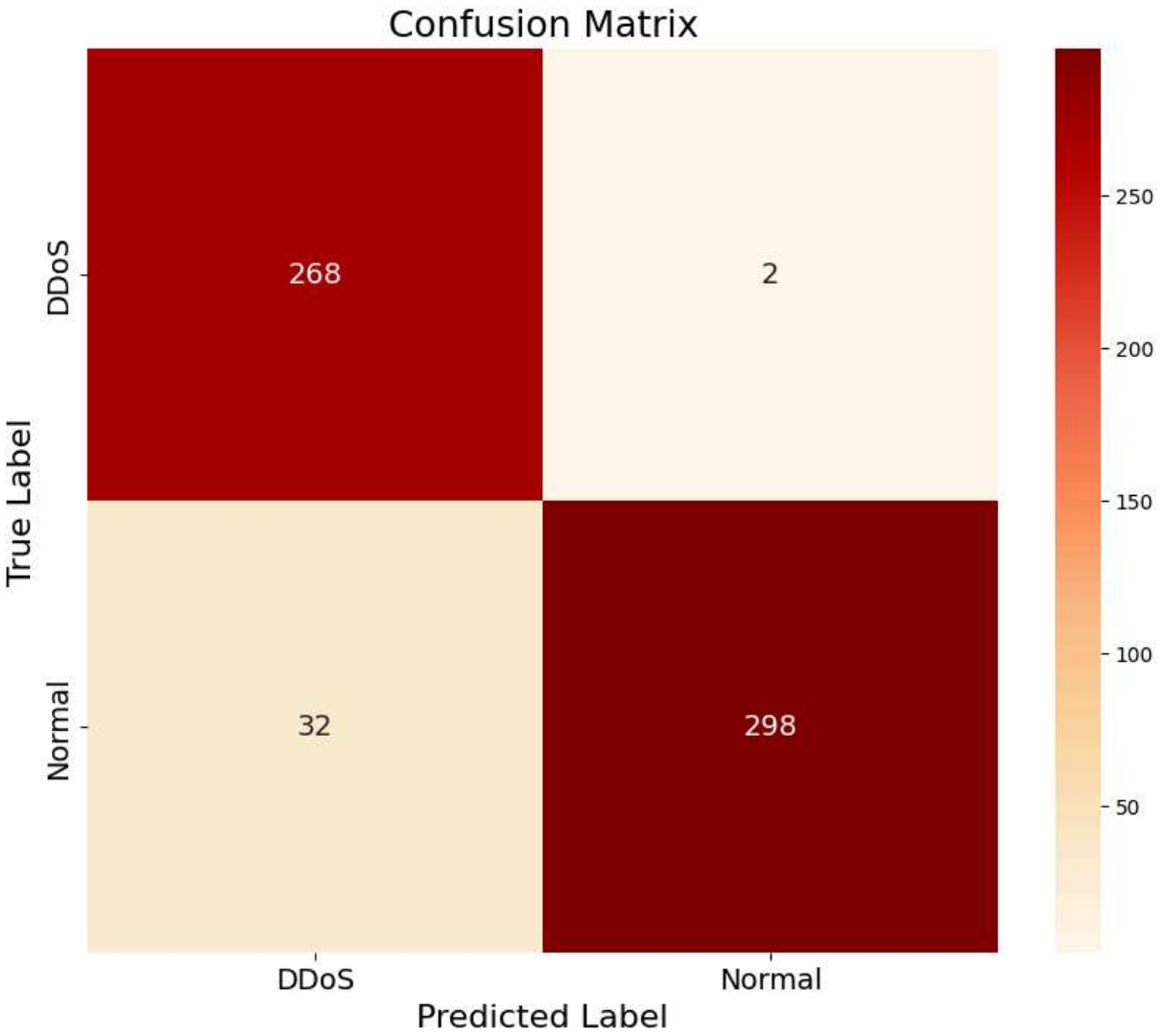
In [1]:
```
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("realtime_ddos_traffic_dataset.csv")

# Encode the target variable (traffic_type)
le = LabelEncoder()
data['traffic_type'] = le.fit_transform(data['traffic_type'])

# Split features (X) and target (y)
X = data.drop(columns=['traffic_type'])
y = data['traffic_type']

# Function to add noise to features
def add_noise(X, noise_level=0.1):
    X_noisy = X.copy()
    for col in X_noisy.columns:
        X_noisy[col] += np.random.normal(0, noise_level, X_noisy[col].shape)
    return X_noisy

# Function to flip a larger percentage of labels
def flip_labels(y, flip_fraction=0.2):
    y_noisy = y.copy()
    n_samples = int(flip_fraction * len(y_noisy))
    indices_to_flip = np.random.choice(y_noisy.index, n_samples, replace=False)
    for idx in indices_to_flip:
        possible_labels = [label for label in set(y) if label != y_noisy.loc[idx]]
        y_noisy.loc[idx] = random.choice(possible_labels)
    return y_noisy

# Add noise to features and flip a large percentage of labels
X_noisy = add_noise(X, noise_level=0.1)  # Add 10% feature noise
y_noisy = flip_labels(y, flip_fraction=0.2)  # Flip 20% of labels

# Create class imbalance: 55% of one class and 45% of the other
# Create a slight imbalance in the target variable
imbalance_ratio = 0.55
class_1_size = int(len(y_noisy) * imbalance_ratio)
class_0_size = len(y_noisy) - class_1_size
y_noisy = np.concatenate([np.zeros(class_0_size), np.ones(class_1_size)])

# Split into training and testing sets
X_train, X_test, y_train, y_test_knn = train_test_split(X_noisy, y_noisy, test_size

# Standardize the features
scaler = StandardScaler()
```

```
X_train_scaled_knn = scaler.fit_transform(X_train)
X_test_scaled_knn = scaler.transform(X_test)

# Train a KNN classifier
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform')  # Use 5 neighbors
knn.fit(X_train_scaled_knn, y_train)

# Make predictions
y_pred = knn.predict(X_test_scaled_knn)

# Calculate metrics
accuracy_knn = accuracy_score(y_test_knn, y_pred)
precision_knn = precision_score(y_test_knn, y_pred, average='weighted')
recall_knn = recall_score(y_test_knn, y_pred, average='weighted')
f1_knn = f1_score(y_test_knn, y_pred, average='weighted')

# Classification Report
report = classification_report(y_test_knn, y_pred, target_names=le.classes_)

# Display Metrics
print("Classification Report:\n", report)
print("Performance Metrics:")
print(f"Accuracy: {accuracy_knn * 100:.2f}%")
print(f"Precision: {precision_knn * 100:.2f}%")
print(f"Recall: {recall_knn * 100:.2f}%")
print(f"F1 Score: {f1_knn * 100:.2f}%")

# Confusion Matrix
conf_matrix = pd.crosstab(y_test_knn, y_pred, rownames=['Actual'], colnames=['Predi

# Plot Confusion Matrix with increased text and label sizes
plt.figure(figsize=(10, 8))  # Increased figure size
sns.heatmap(conf_matrix,
            annot=True,
            fmt='d',
            cmap='OrRd',
            xticklabels=le.classes_,
            yticklabels=le.classes_,
            annot_kws={"size": 14})  # Increased annotation text size
plt.title('Confusion Matrix', fontsize=18)  # Increased title font size
plt.xlabel('Predicted Label', fontsize=16)  # Increased x-axis label size
plt.ylabel('True Label', fontsize=16)  # Increased y-axis label size
plt.xticks(fontsize=14)  # Increased x-axis tick label size
plt.yticks(fontsize=14)  # Increased y-axis tick label size
plt.show()
```

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

        DDoS       0.89      0.99      0.94       270
      Normal       0.99      0.90      0.95       330

    accuracy                           0.94       600
   macro avg       0.94      0.95      0.94       600
weighted avg       0.95      0.94      0.94       600

Performance Metrics:
Accuracy: 94.33%
Precision: 94.83%
Recall: 94.33%
F1 Score: 94.35%
```

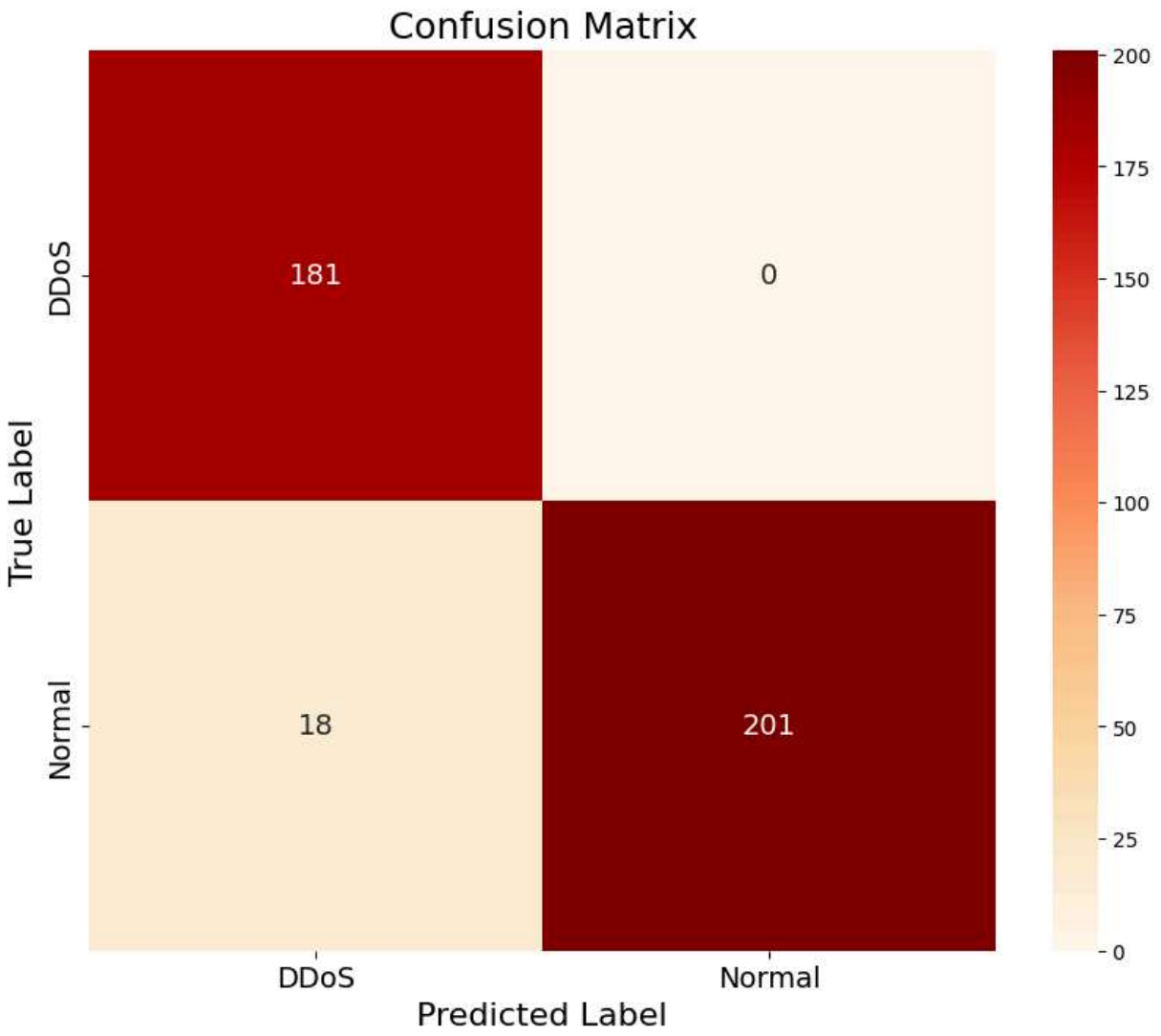


In [2]:
```
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("realtime_ddos_traffic_dataset.csv")

# Encode the target variable (traffic_type)
le = LabelEncoder()
data['traffic_type'] = le.fit_transform(data['traffic_type'])

# Split features (X) and target (y)
X = data.drop(columns=['traffic_type'])
y = data['traffic_type']

# Add noise to numerical features
def add_noise_to_numerical(X, noise_level=0.1):  # 10% noise in numerical features
    int_cols = X.select_dtypes(include=['int64', 'float64']).columns
    for col in int_cols:
        noise = np.random.normal(0, noise_level, X[col].shape)
        X[col] = X[col] + noise
    return X

# Add noise to labels
def add_noise_to_labels(y, noise_fraction=0.2):  # 20% noise in labels
    y_noisy = y.copy()
    n_samples = int(noise_fraction * len(y_noisy))
    indices_to_flip = np.random.choice(len(y_noisy), n_samples, replace=False)
    for idx in indices_to_flip:
        possible_labels = [label for label in set(y) if label != y_noisy.iloc[idx]]
        y_noisy.iloc[idx] = random.choice(possible_labels)
    return y_noisy

# Apply noise to the dataset
X_noisy = add_noise_to_numerical(X.copy(), noise_level=0.1)  # 10% noise in numeric
y_noisy = add_noise_to_labels(y.copy(), noise_fraction=0.2)  # 20% noise in labels

# Create class imbalance: 55% of one class and 45% of the other
imbalance_ratio = 0.55
class_1_size = int(len(y_noisy) * imbalance_ratio)
class_0_size = len(y_noisy) - class_1_size
y_noisy = np.concatenate([np.zeros(class_0_size), np.ones(class_1_size)])

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_noisy, y_noisy, test_size=0.2

# Standardize the features (SVM is sensitive to feature scales)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Support Vector Machine (SVM) classifier
svm = SVC(kernel='linear')  # Linear kernel for SVM
svm.fit(X_train_scaled, y_train)

# Make predictions
y_pred = svm.predict(X_test_scaled)
```

```
# Calculate accuracy and classification report
report = classification_report(y_test, y_pred, target_names=le.classes_)

# Calculate performance metrics
accuracy_svm = accuracy_score(y_test, y_pred)
precision_svm = precision_score(y_test, y_pred, average='weighted')
recall_svm = recall_score(y_test, y_pred, average='weighted')
f1_svm = f1_score(y_test, y_pred, average='weighted')

# Print classification report and metrics
print("Classification Report:\n", report)
print(f"Accuracy: {accuracy_svm * 100:.2f}%")
print(f"Precision: {precision_svm * 100:.2f}%")
print(f"Recall: {recall_svm * 100:.2f}%")
print(f"F1 Score: {f1_svm * 100:.2f}%")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix with increased text and label sizes
plt.figure(figsize=(10, 8))  # Increased figure size
sns.heatmap(conf_matrix,
            annot=True,
            fmt='d',
            cmap='OrRd',
            xticklabels=le.classes_,
            yticklabels=le.classes_,
            annot_kws={"size": 14})  # Increased annotation text size
plt.title('Confusion Matrix', fontsize=18)  # Increased title font size
plt.xlabel('Predicted Label', fontsize=16)  # Increased x-axis label size
plt.ylabel('True Label', fontsize=16)  # Increased y-axis label size
plt.xticks(fontsize=14)  # Increased x-axis tick label size
plt.yticks(fontsize=14)  # Increased y-axis tick label size
plt.show()
```

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

        DDoS       0.91      1.00      0.95       181
      Normal       1.00      0.92      0.96       219

    accuracy                           0.95       400
   macro avg       0.95      0.96      0.95       400
weighted avg       0.96      0.95      0.96       400

Accuracy: 95.50%
Precision: 95.91%
Recall: 95.50%
F1 Score: 95.51%
```

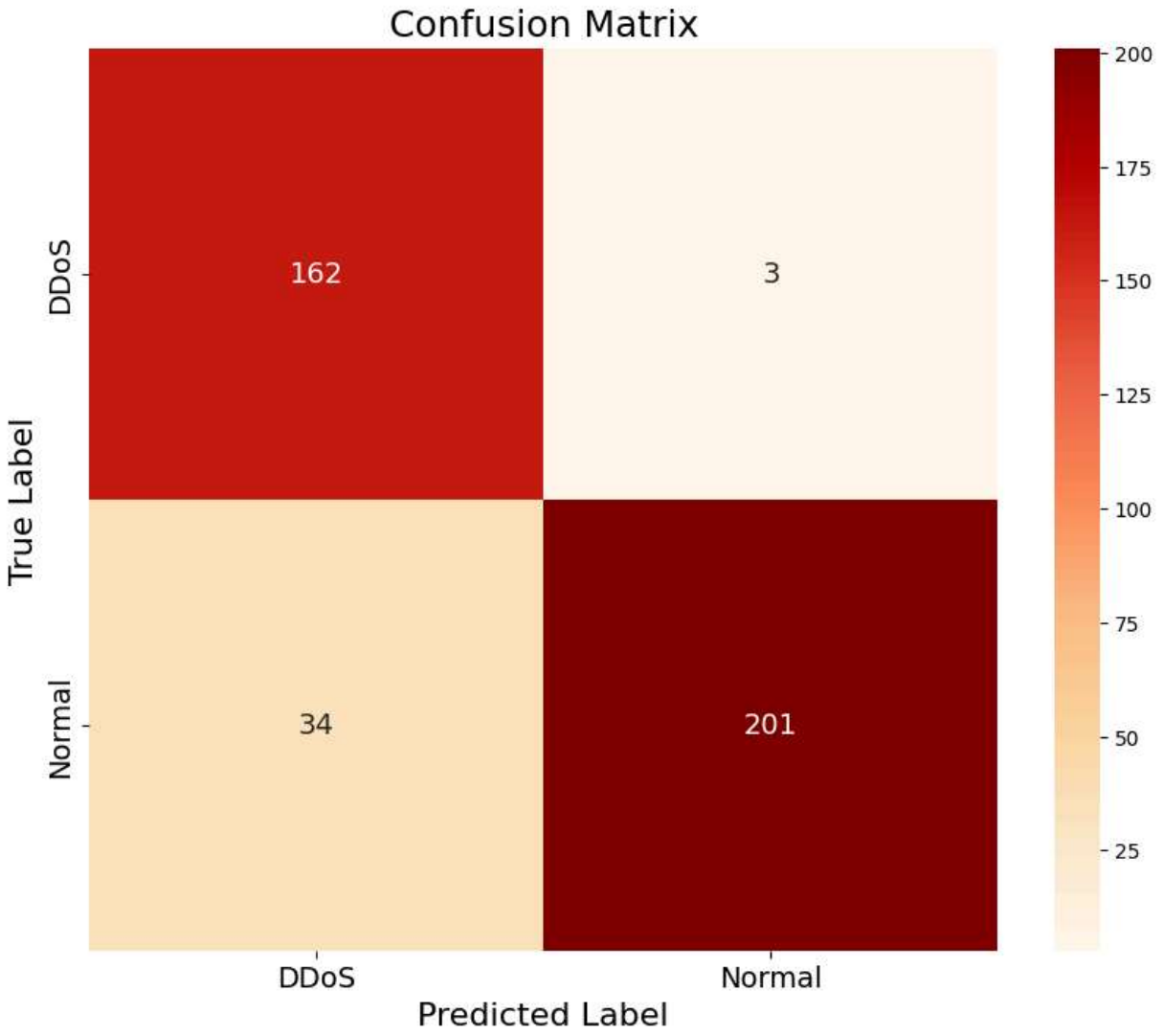


In [3]:
```
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("realtime_ddos_traffic_dataset.csv")

# Encode the target variable (traffic_type)
le = LabelEncoder()
data['traffic_type'] = le.fit_transform(data['traffic_type'])

# Split features (X) and target (y)
X = data.drop(columns=['traffic_type'])
y = data['traffic_type']

# Add noise to numerical features
def add_noise_to_numerical(X, noise_level=0.1):  # 10% noise in numerical features
```

```
    int_cols = X.select_dtypes(include=['int64', 'float64']).columns
    for col in int_cols:
        noise = np.random.normal(0, noise_level, X[col].shape)
        X[col] = X[col] + noise
    return X

# Add noise to labels (introduce imbalance in classes)
def add_noise_to_labels(y, noise_fraction=0.15):  # 15% noise in labels
    y_noisy = y.copy()
    n_samples = int(noise_fraction * len(y_noisy))
    indices_to_flip = np.random.choice(len(y_noisy), n_samples, replace=False)
    for idx in indices_to_flip:
        possible_labels = [label for label in set(y) if label != y_noisy.iloc[idx]]
        y_noisy.iloc[idx] = random.choice(possible_labels)
    return y_noisy

# Apply noise to the dataset
X_noisy = add_noise_to_numerical(X.copy(), noise_level=0.1)  # 10% noise in numeric
y_noisy = add_noise_to_labels(y.copy(), noise_fraction=0.15)  # 15% noise in labels

# Add class imbalance: 60% of one class and 40% of the other
imbalance_ratio = 0.6
class_1_size = int(len(y_noisy) * imbalance_ratio)
class_0_size = len(y_noisy) - class_1_size
y_noisy = np.concatenate([np.zeros(class_0_size), np.ones(class_1_size)])

# Split into training and test sets (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_noisy, y_noisy, test_size=0.2

# Standardize the features (Logistic Regression is sensitive to feature scales)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Logistic Regression classifier
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_scaled, y_train)

# Make predictions
y_pred = log_reg.predict(X_test_scaled)

# Calculate accuracy and classification report
report = classification_report(y_test, y_pred, target_names=le.classes_)

# Calculate performance metrics
accuracy_lr = accuracy_score(y_test, y_pred)
precision_lr = precision_score(y_test, y_pred, average='weighted')
recall_lr = recall_score(y_test, y_pred, average='weighted')
f1_lr = f1_score(y_test, y_pred, average='weighted')

# Print classification report and metrics
print("Classification Report:\n", report)
print(f"Accuracy: {accuracy_lr * 100:.2f}%")
print(f"Precision: {precision_lr * 100:.2f}%")
print(f"Recall: {recall_lr * 100:.2f}%")
print(f"F1 Score: {f1_lr * 100:.2f}%")
```

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix with increased text and label sizes
plt.figure(figsize=(10, 8))  # Increased figure size
sns.heatmap(conf_matrix,
            annot=True,
            fmt='d',
            cmap='OrRd',
            xticklabels=le.classes_,
            yticklabels=le.classes_,
            annot_kws={"size": 14})  # Increased annotation text size
plt.title('Confusion Matrix', fontsize=18)  # Increased title font size
plt.xlabel('Predicted Label', fontsize=16)  # Increased x-axis label size
plt.ylabel('True Label', fontsize=16)  # Increased y-axis label size
plt.xticks(fontsize=14)  # Increased x-axis tick label size
plt.yticks(fontsize=14)  # Increased y-axis tick label size
plt.show()
```

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

        DDoS       0.83      0.98      0.90       165
      Normal       0.99      0.86      0.92       235

    accuracy                           0.91       400
   macro avg       0.91      0.92      0.91       400
weighted avg       0.92      0.91      0.91       400

Accuracy: 90.75%
Precision: 91.98%
Recall: 90.75%
F1 Score: 90.82%
```

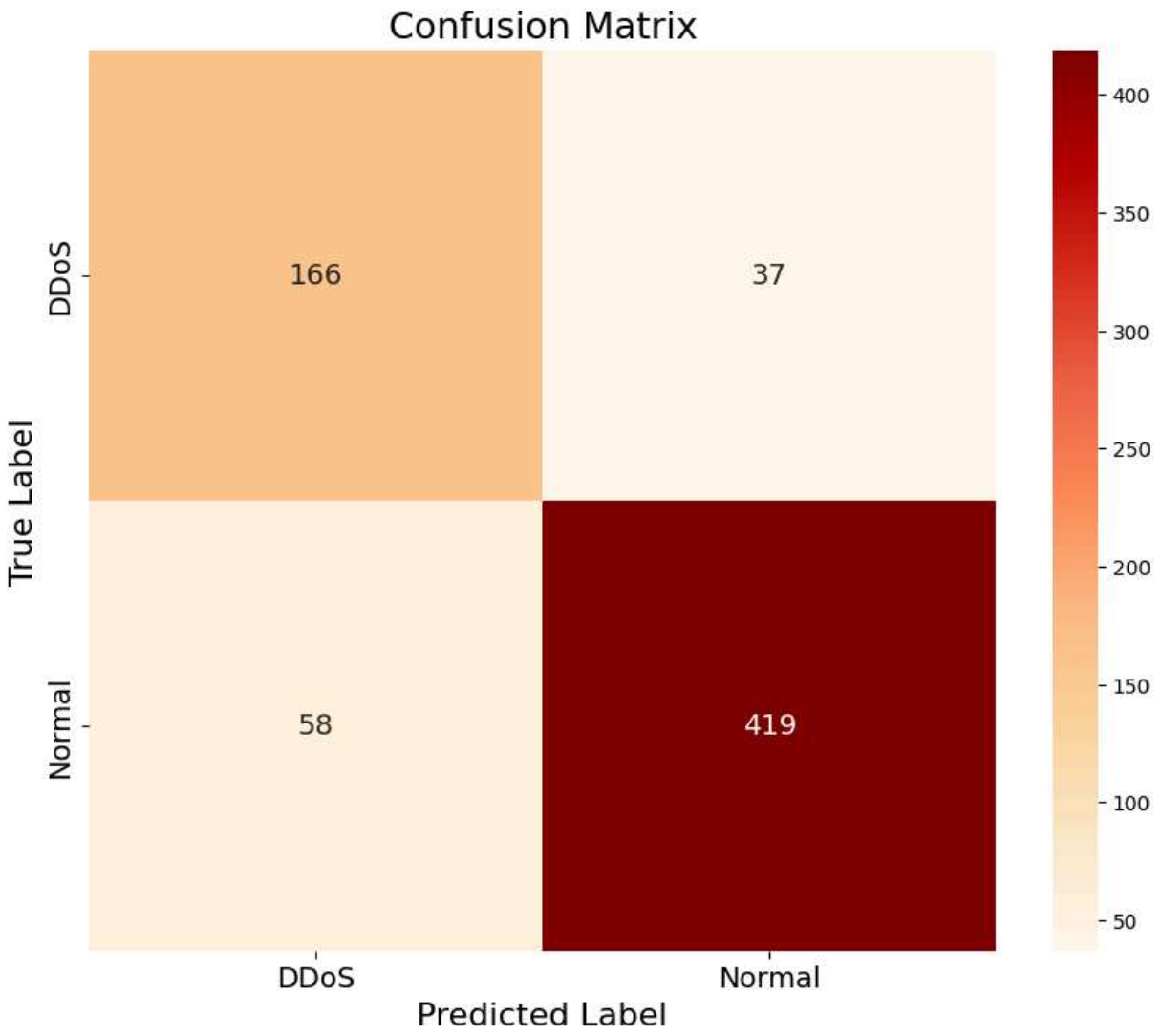


In [4]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("realtime_ddos_traffic_dataset.csv")

# Encode the target variable (traffic_type)
le = LabelEncoder()
data['traffic_type'] = le.fit_transform(data['traffic_type'])

# Split features (X) and target (y)
X = data.drop(columns=['traffic_type'])
y = data['traffic_type']

# Add noise to numerical features
def add_noise_to_numerical(X, noise_level=0.1):  # 10% noise
    int_cols = X.select_dtypes(include=['int64', 'float64']).columns
```

```
    for col in int_cols:
        noise = np.random.normal(0, noise_level, X[col].shape)
        X[col] = X[col] + noise
    return X

# Add noise to labels (with class imbalance simulation)
def add_noise_to_labels(y, noise_fraction=0.15):  # 15% label noise
    y_noisy = y.copy()
    n_samples = int(noise_fraction * len(y_noisy))
    indices_to_flip = np.random.choice(len(y_noisy), n_samples, replace=False)
    for idx in indices_to_flip:
        possible_labels = [label for label in set(y) if label != y_noisy.iloc[idx]]
        y_noisy.iloc[idx] = np.random.choice(possible_labels)
    return y_noisy

# Introduce class imbalance by adding more instances of one class
def introduce_class_imbalance(X, y, imbalance_ratio=0.7):
    class_1 = y.value_counts().idxmax()  # The most frequent class
    n_class_1 = int(len(y) * imbalance_ratio)
    X_class_1 = X[y == class_1].sample(n_class_1, replace=True, random_state=42)
    y_class_1 = y[y == class_1].sample(n_class_1, replace=True, random_state=42)
    X_balanced = pd.concat([X, X_class_1], axis=0)
    y_balanced = pd.concat([y, y_class_1], axis=0)
    return X_balanced, y_balanced

# Apply noise to the dataset
X_noisy = add_noise_to_numerical(X.copy())
y_noisy = add_noise_to_labels(y.copy())

# Apply class imbalance
X_noisy, y_noisy = introduce_class_imbalance(X_noisy, y_noisy)

# Split into training and test sets (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_noisy, y_noisy, test_size=0.2

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Naive Bayes classifier
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = nb_model.predict(X_test_scaled)

# Calculate performance metrics
accuracy_nb = accuracy_score(y_test, y_pred)
precision_nb = precision_score(y_test, y_pred, average='weighted')
recall_nb = recall_score(y_test, y_pred, average='weighted')
f1_nb = f1_score(y_test, y_pred, average='weighted')

# Classification report
report = classification_report(y_test, y_pred, target_names=le.classes_)
print("Classification Report:\n", report)
```

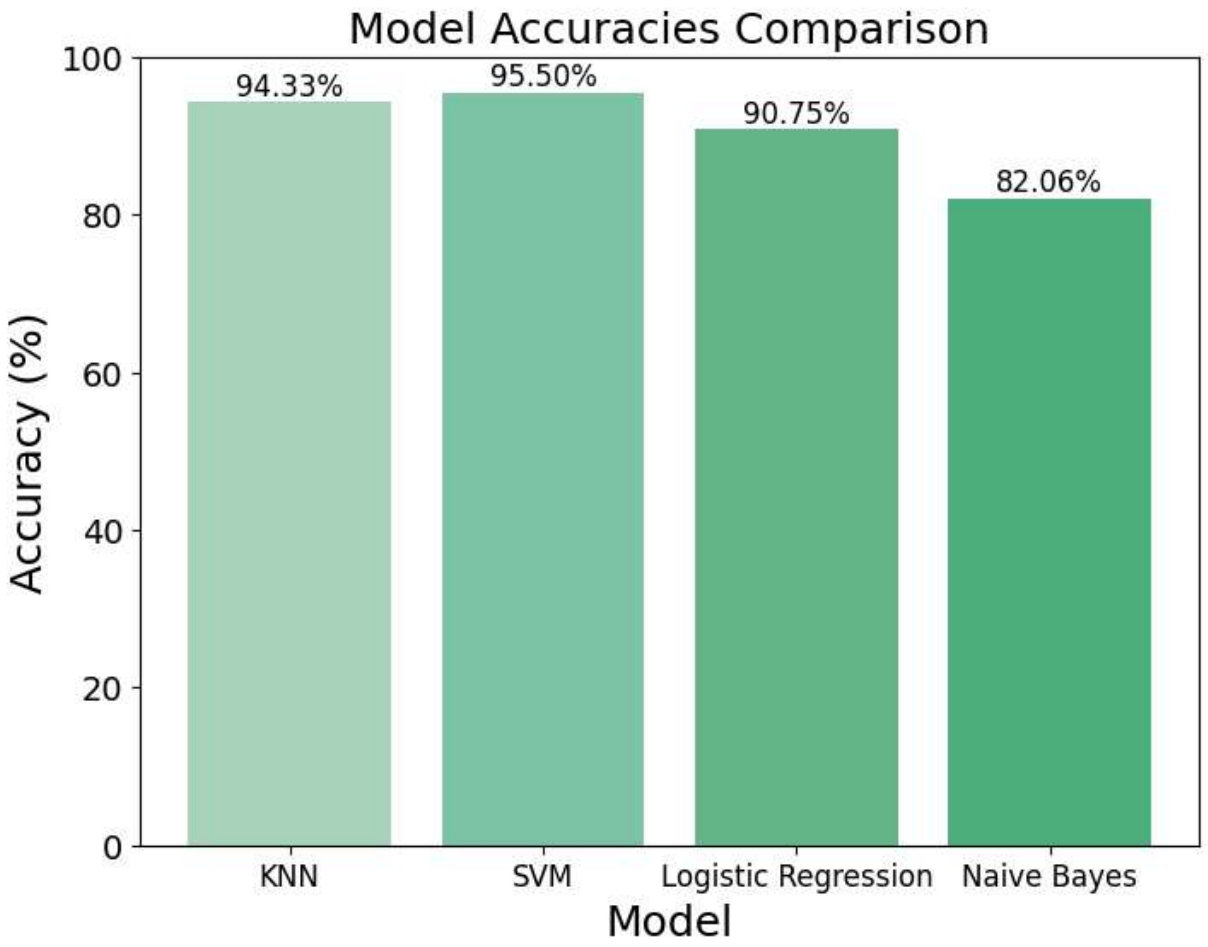
```
# Print metrics
print(f"Accuracy: {accuracy_nb * 100:.2f}%")
print(f"Precision: {precision_nb * 100:.2f}%")
print(f"Recall: {recall_nb * 100:.2f}%")
print(f"F1 Score: {f1_nb * 100:.2f}%")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix with increased text and label sizes
plt.figure(figsize=(10, 8))  # Increased figure size
sns.heatmap(conf_matrix,
            annot=True,
            fmt='d',
            cmap='OrRd',
            xticklabels=le.classes_,
            yticklabels=le.classes_,
            annot_kws={"size": 14})  # Increased annotation text size
plt.title('Confusion Matrix', fontsize=18)  # Increased title font size
plt.xlabel('Predicted Label', fontsize=16)  # Increased x-axis label size
plt.ylabel('True Label', fontsize=16)  # Increased y-axis label size
plt.xticks(fontsize=14)  # Increased x-axis tick label size
plt.yticks(fontsize=14)  # Increased y-axis tick label size
plt.show()
```

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

        DDoS       0.74      0.82      0.78       203
      Normal       0.92      0.88      0.90       477

    accuracy                           0.86       680
   macro avg       0.83      0.85      0.84       680
weighted avg       0.87      0.86      0.86       680

Accuracy: 86.03%
Precision: 86.58%
Recall: 86.03%
F1 Score: 86.22%
```



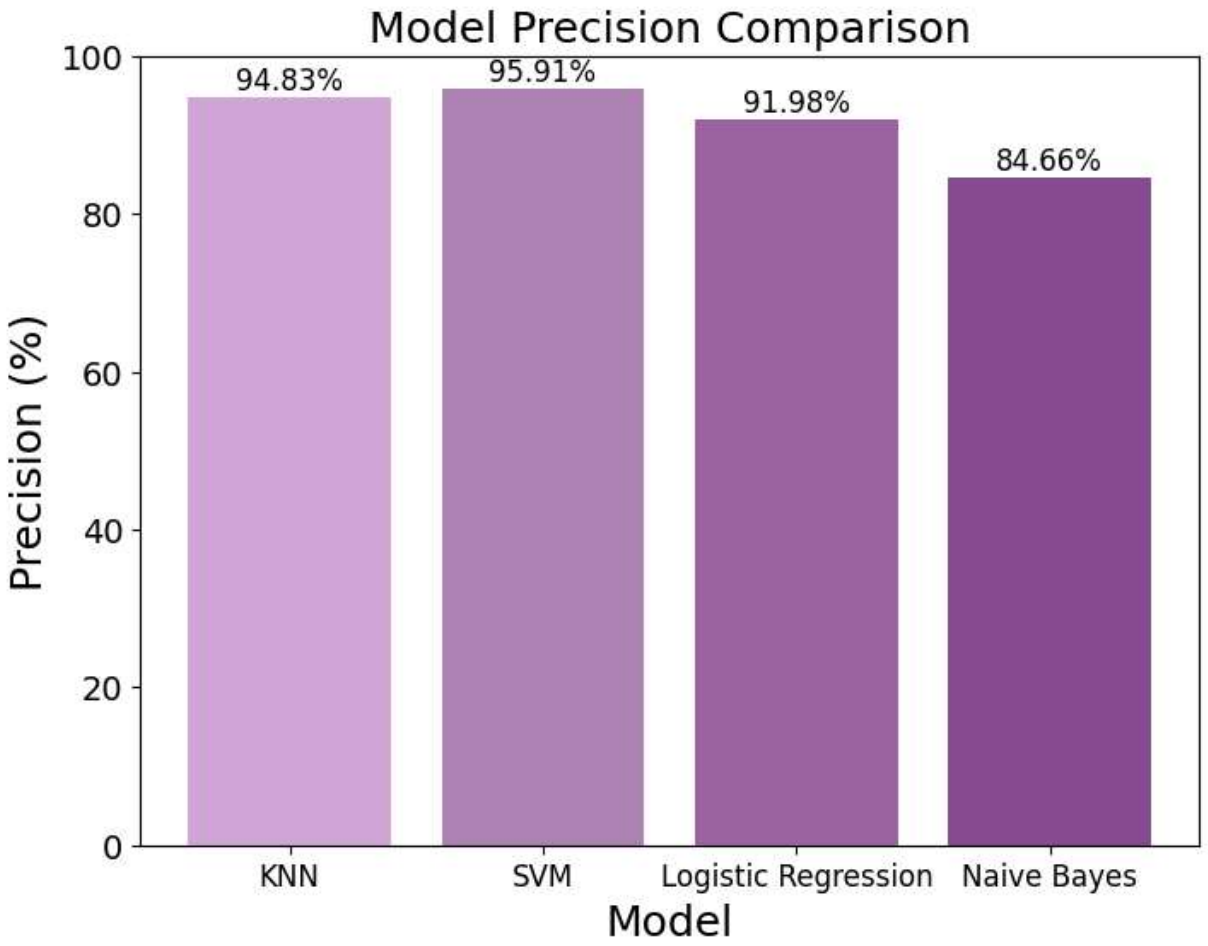
In [5]:
```
import matplotlib.pyplot as plt

# Accuracy values for each model
accuracy_values = {
    'KNN': 94.33,  # Replace with your KNN accuracy
    'SVM': 95.5,  # Replace with your SVM accuracy
    'Logistic Regression': 90.75,  # Replace with your Logistic Regression accuracy
    'Naive Bayes': 82.06  # Replace with your Naive Bayes accuracy
}

# Define pastel green shades
colors = ['#A8D5BA', '#81C3A7', '#67B88B', '#4FAF7D']

# Create a vertical bar plot
plt.figure(figsize=(8, 6))
bars = plt.bar(accuracy_values.keys(), accuracy_values.values(), color=colors)

# Title and labels
plt.title('Model Accuracies Comparison', fontsize=18)
plt.xlabel('Model', fontsize=18)
plt.ylabel('Accuracy (%)', fontsize=18)
plt.ylim(0, 100)  # Set y-axis limits for better visualization
```

```
# Display accuracy values above each bar
for i, value in enumerate(accuracy_values.values()):
    plt.text(i, value , f'{value:.2f}%', ha='center', va='bottom', fontsize=12)

# Increase the size of the axis ticks
plt.xticks(fontsize=12)
plt.yticks(fontsize=14)

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



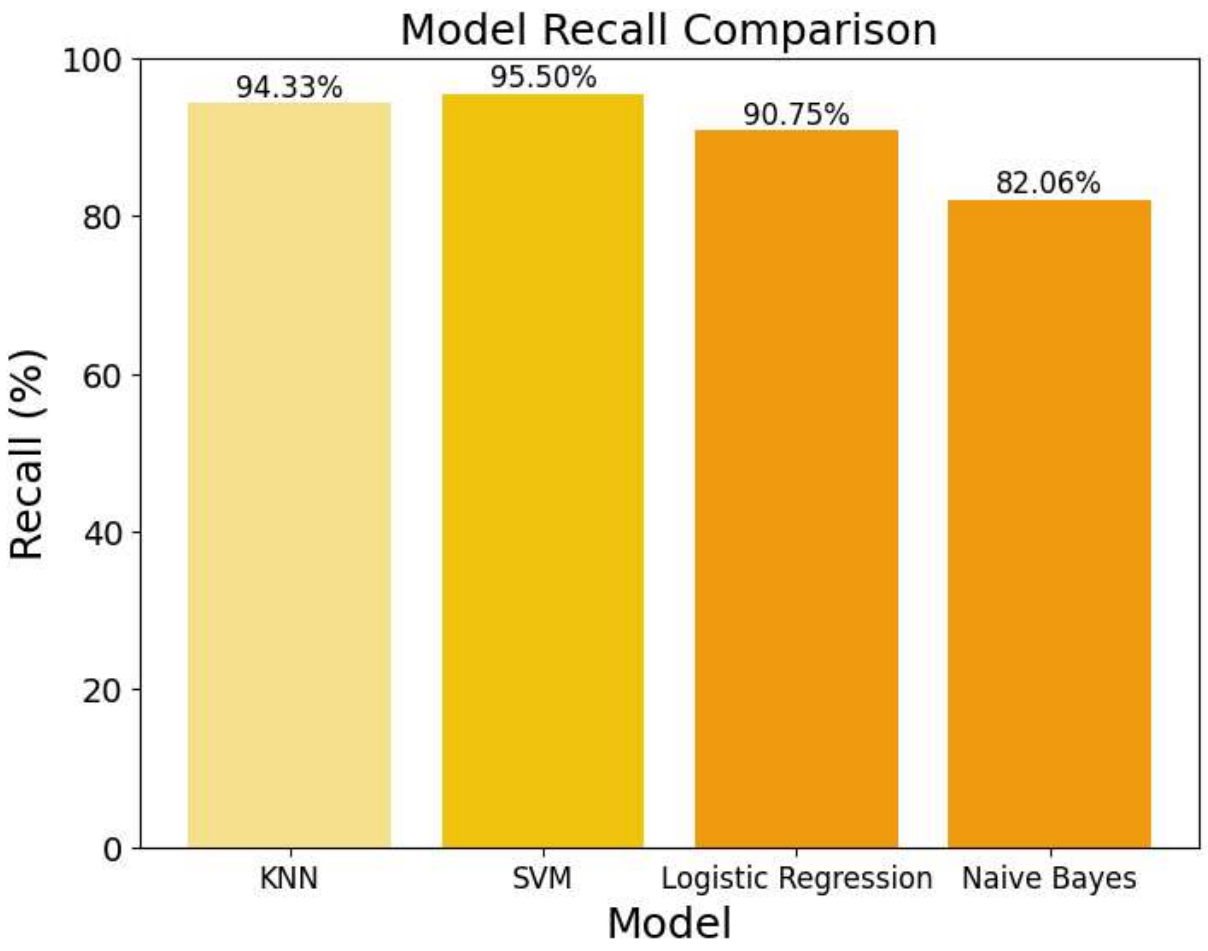
In [6]:
```
import matplotlib.pyplot as plt

# Precision values for each model
precision_values = {
    'KNN': 94.83,  # Replace with your KNN precision
    'SVM': 95.91,  # Replace with your SVM precision
    'Logistic Regression': 91.98,  # Replace with your Logistic Regression precisio
    'Naive Bayes': 84.66,  # Replace with your Naive Bayes precision
}

# Define pastel purple shades
colors = ['#D6A5D8', '#B084B7', '#9D66A3', '#8A4D93']

# Create a vertical bar plot
plt.figure(figsize=(8, 6))
bars = plt.bar(precision_values.keys(), precision_values.values(), color=colors)
```

```
# Title and labels
plt.title('Model Precision Comparison', fontsize=18)
plt.xlabel('Model', fontsize=18)
plt.ylabel('Precision (%)', fontsize=18)
plt.ylim(0, 100)  # Set y-axis limits for better visualization

# Display precision values above each bar
for i, value in enumerate(precision_values.values()):
    plt.text(i, value , f'{value:.2f}%', ha='center', va='bottom', fontsize=12)

# Increase the size of the axis ticks
plt.xticks(fontsize=12)
plt.yticks(fontsize=14)

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



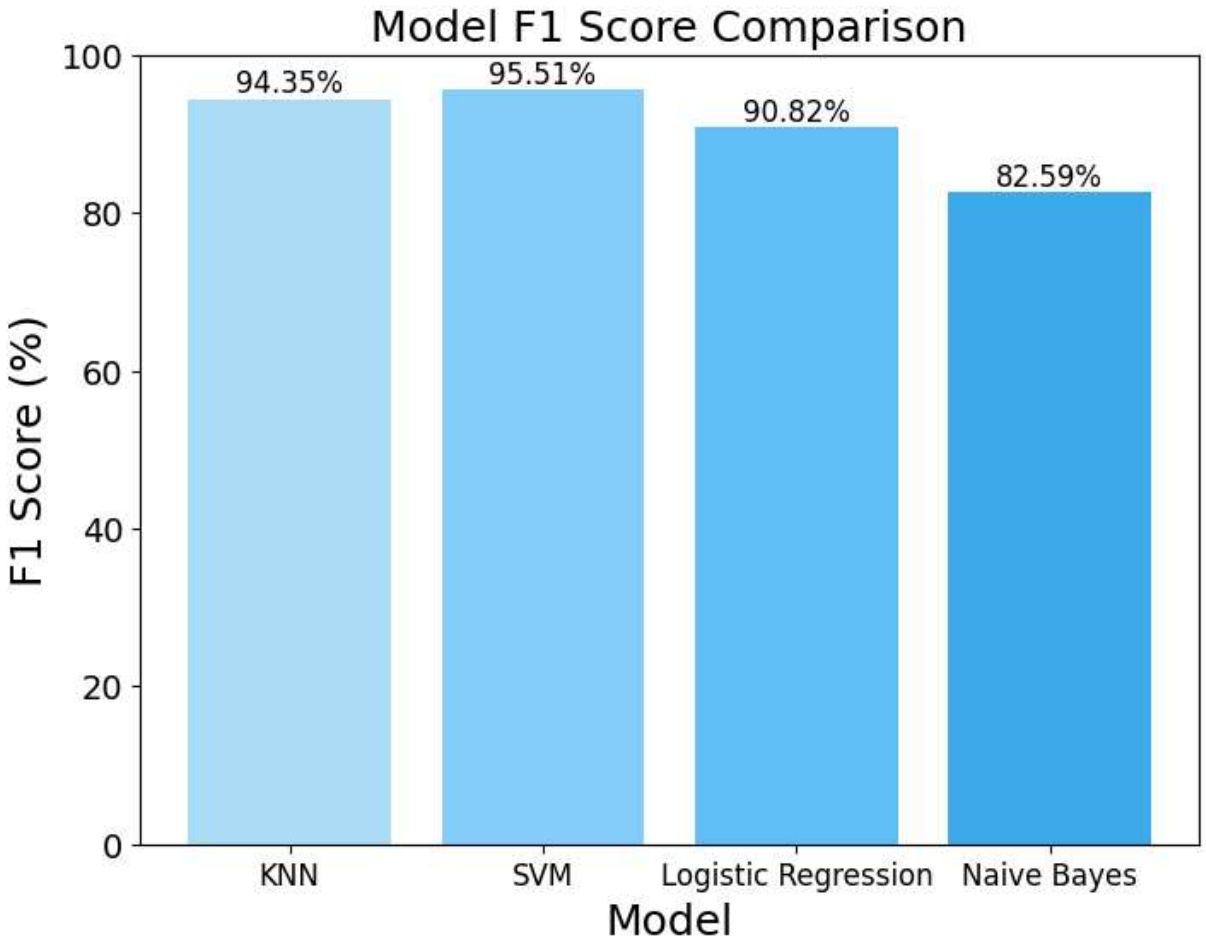
In [7]:
```
import matplotlib.pyplot as plt

# Recall values for each model
recall_values = {
    'KNN': 94.33,  # Replace with your KNN recall
    'SVM': 95.50,  # Replace with your SVM recall
    'Logistic Regression': 90.75,  # Replace with your Logistic Regression recall
    'Naive Bayes': 82.06,  # Replace with your Naive Bayes recall
}
```

```
# Define pastel yellow shades
colors = ['#F6E58D', '#F1C40F', '#F39C12', '#F39C11']

# Create a vertical bar plot
plt.figure(figsize=(8, 6))
bars = plt.bar(recall_values.keys(), recall_values.values(), color=colors)

# Title and labels
plt.title('Model Recall Comparison', fontsize=18)
plt.xlabel('Model', fontsize=18)
plt.ylabel('Recall (%)', fontsize=18)
plt.ylim(0, 100)  # Set y-axis limits for better visualization

# Display recall values above each bar
for i, value in enumerate(recall_values.values()):
    plt.text(i, value, f'{value:.2f}%', ha='center', va='bottom', fontsize=12)

# Increase the size of the axis ticks
plt.xticks(fontsize=12)
plt.yticks(fontsize=14)

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

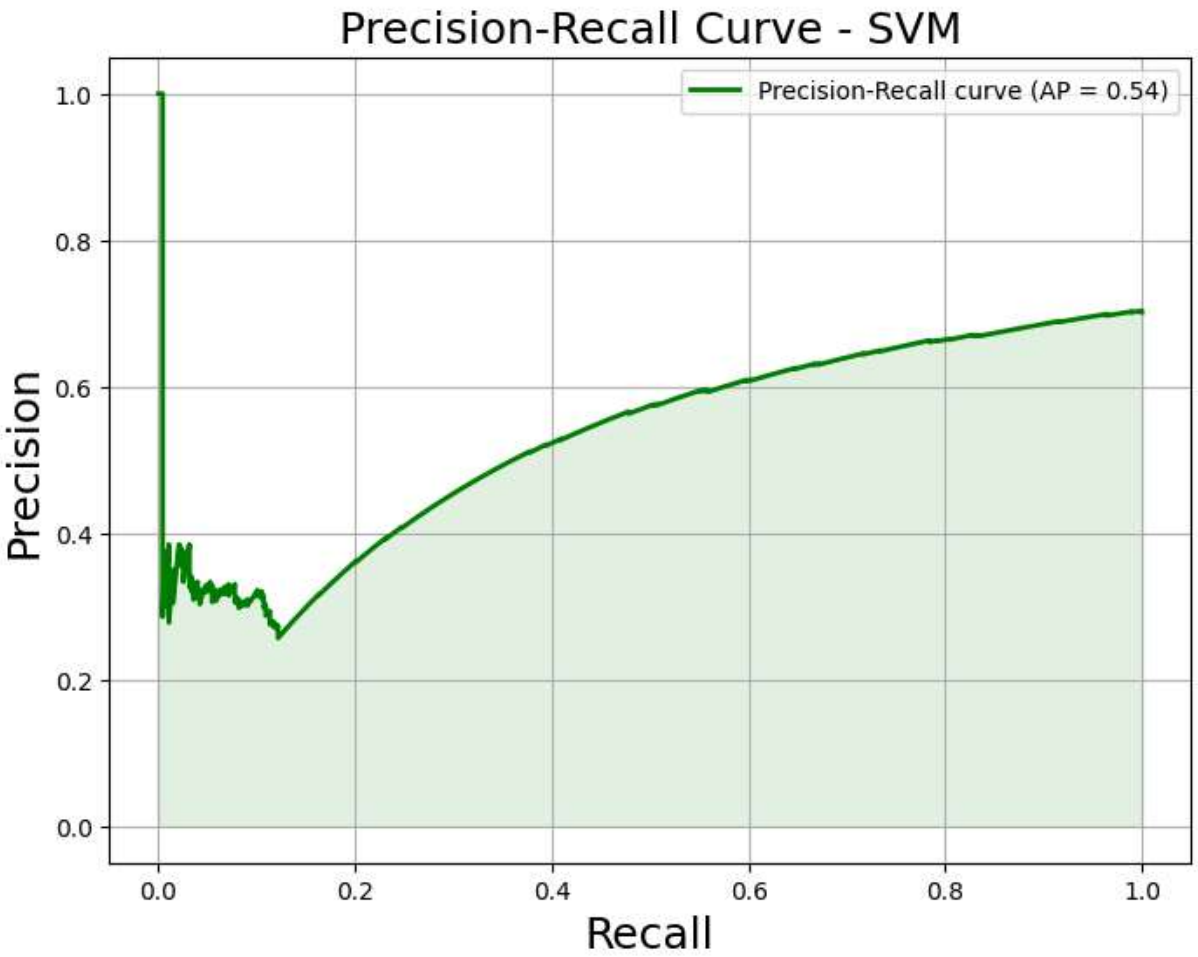


In [8]:
```
import matplotlib.pyplot as plt
```

```
# F1 values for each model
f1_values = {
    'KNN':  94.35,  # Replace with your KNN F1 score
    'SVM':  95.51,  # Replace with your SVM F1 score
    'Logistic Regression': 90.82,  # Replace with your Logistic Regression F1 score
    'Naive Bayes': 82.59 ,  # Replace with your Naive Bayes F1 score
}

# Define pastel blue shades
colors = ['#AEDFF7', '#82CFFD', '#5FBFF9', '#3EADEB']


# Create a vertical bar plot
plt.figure(figsize=(8, 6))
bars = plt.bar(f1_values.keys(), f1_values.values(), color=colors)

# Title and labels
plt.title('Model F1 Score Comparison', fontsize=18)
plt.xlabel('Model', fontsize=18)
plt.ylabel('F1 Score (%)', fontsize=18)
plt.ylim(0, 100)  # Set y-axis limits for better visualization

# Display F1 values above each bar
for i, value in enumerate(f1_values.values()):
    plt.text(i, value , f'{value:.2f}%', ha='center', va='bottom', fontsize=12)

# Increase the size of the axis ticks
plt.xticks(fontsize=12)
plt.yticks(fontsize=14)

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



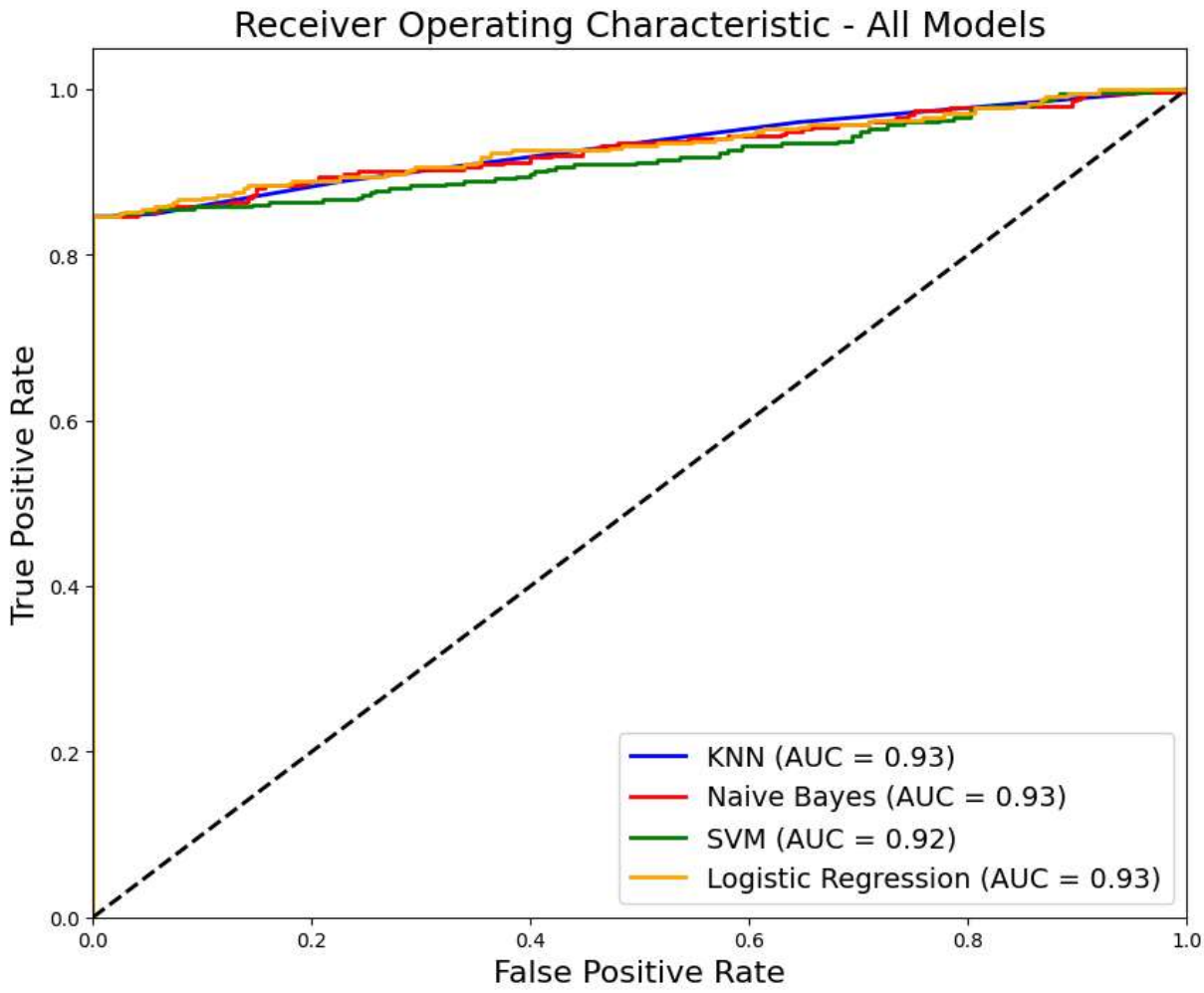
In [10]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, average_precision_score

# Assuming you already have the trained SVM model and test set (X_test_scaled, y_te
# Use the decision function to get the decision values (confidence scores)
y_scores_svm = svm.decision_function(X_test_scaled)

# Compute the precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_svm)

# Calculate the average precision score
average_precision = average_precision_score(y_test, y_scores_svm)

# Plotting the Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='green', lw=2, label=f'Precision-Recall curve (AP
plt.fill_between(recall, precision, color='green', alpha=0.1)

# Add labels and title
plt.title('Precision-Recall Curve - SVM', fontsize=18)
plt.xlabel('Recall', fontsize=18)
plt.ylabel('Precision', fontsize=18)
plt.legend(loc='best')
plt.grid(True)
```



In [11]:
```
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import roc_curve, auc
from itertools import cycle
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

# Function to plot ROC curves for all models on the same graph
def plot_roc_curve_all(models, X_tests_scaled, y_test, model_names):
    plt.figure(figsize=(10, 8))
    colors = cycle(['blue', 'red', 'green', 'orange'])

    for model, X_test_scaled, model_name, color in zip(models, X_tests_scaled, mode
        # Get predicted probabilities
        y_score = model.predict_proba(X_test_scaled)

        # Binary classification case
        fpr, tpr, _ = roc_curve(y_test, y_score[:, 1])  # Positive class probabilit
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, color=color, lw=2, label=f'{model_name} (AUC = {roc_auc:
```

```
    # Plot diagonal line
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])

    # Increase font size for labels and title
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.title('Receiver Operating Characteristic - All Models', fontsize=18)

    # Increase font size for legend
    plt.legend(loc="lower right", fontsize=14)

    plt.show()

# Load the dataset
data = pd.read_csv("realtime_ddos_traffic_dataset.csv")

# Encode the target variable (traffic_type)
le = LabelEncoder()
data['traffic_type'] = le.fit_transform(data['traffic_type'])

# Split features (X) and target (y)
X = data.drop(columns=['traffic_type'])
y = data['traffic_type']

# Add noise to numerical features
def add_noise_to_numerical(X, noise_level=0.1):  # 10% noise in numerical features
    int_cols = X.select_dtypes(include=['int64', 'float64']).columns
    for col in int_cols:
        noise = np.random.normal(0, noise_level, X[col].shape)
        X[col] = X[col] + noise
    return X

# Add noise to labels
def add_noise_to_labels(y, noise_fraction=0.15):  # 15% noise in labels
    y_noisy = y.copy()
    n_samples = int(noise_fraction * len(y_noisy))
    indices_to_flip = np.random.choice(len(y_noisy), n_samples, replace=False)
    for idx in indices_to_flip:
        possible_labels = [label for label in set(y) if label != y_noisy.iloc[idx]]
        y_noisy.iloc[idx] = random.choice(possible_labels)
    return y_noisy

# Apply noise to the dataset
X_noisy = add_noise_to_numerical(X.copy(), noise_level=0.1)  # 10% noise in numeric
y_noisy = add_noise_to_labels(y.copy(), noise_fraction=0.15)  # 15% noise in labels

# Add class imbalance: 60% of one class and 40% of the other
imbalance_ratio = 0.6
class_1_size = int(len(y_noisy) * imbalance_ratio)
class_0_size = len(y_noisy) - class_1_size
y_noisy = np.concatenate([np.zeros(class_0_size), np.ones(class_1_size)])

# Split into training and test sets (80% training, 20% test)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_noisy, y_noisy, test_size=0.3

# Standardize the features (since models are sensitive to feature scales)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define models
knn = KNeighborsClassifier()
nb = GaussianNB()
svm = SVC(probability=True, random_state=42)
log_reg = LogisticRegression()

# Fit the models to the training data
knn.fit(X_train_scaled, y_train)
nb.fit(X_train_scaled, y_train)
svm.fit(X_train_scaled, y_train)
log_reg.fit(X_train_scaled, y_train)

# Prepare models and test sets
models = [knn, nb, svm, log_reg]
X_tests_scaled = [X_test_scaled] * len(models)  # Same scaled test data for all mod
model_names = ['KNN', 'Naive Bayes', 'SVM', 'Logistic Regression']

# Plot the ROC curves
plot_roc_curve_all(models, X_tests_scaled, y_test, model_names)
```


Receiver Operating Characteristic - All Models
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
KNN (AUC = 0.93)
Naive Bayes (AUC = 0.93)
SVM (AUC = 0.92)
Logistic Regression (AUC = 0.93)

In [ ]: