In [3]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv('emails.csv')
print(df.head())
print(df.columns)
print(df.isnull().sum().sum())  # Total missing values in the dataset
print(df.describe())  # Basic statistics of numerical columns
# Preprocessing: Remove unnecessary columns and prepare the dataset for feature selection
# Assuming 'Prediction' is the target column and the rest are features

# Separate features and target variable
X = df.drop(columns=['Email No.', 'Prediction'], errors='ignore')  # Drop unnecessary columns
y = df['Prediction']

# Display the shape of the features and target
print(X.shape)
print(y.shape)
# Combine all text features into a single column (excluding Email No. and Prediction)
text_features = df.drop(['Email No.', 'Prediction'], axis=1, errors='ignore')
X = text_features.astype(str).agg(' '.join, axis=1)
y = df['Prediction']

print("Dataset shape:", df.shape)
print("\
Sample of combined text:")
print(X.head())
print("\
Target variable distribution:")
print(y.value_counts())
```

```
  Email No.  the  to  ect  and  for  of    a  you  hou  ...  connevey  jay  \
0   Email 1    0   0    1    0    0   0    2    0    0  ...         0    0
1   Email 2    8  13   24    6    6   2  102    1   27  ...         0    0
2   Email 3    0   0    1    0    0   0    8    0    0  ...         0    0
3   Email 4    0   5   22    0    5   1   51    2   10  ...         0    0
4   Email 5    7   6   17    1    5   2   57    0    9  ...         0    0

   valued  lay  infrastructure  military  allowing  ff  dry  Prediction
0       0    0               0         0         0   0    0           0
1       0    0               0         0         0   1    0           0
2       0    0               0         0         0   0    0           0
3       0    0               0         0         0   0    0           0
4       0    0               0         0         0   1    0           0

[5 rows x 3002 columns]
Index(['Email No.', 'the', 'to', 'ect', 'and', 'for', 'of', 'a', 'you', 'hou',
       ...
       'connevey', 'jay', 'valued', 'lay', 'infrastructure', 'military',
       'allowing', 'ff', 'dry', 'Prediction'],
      dtype='object', length=3002)
0
               the           to          ect          and          for  \
count  5172.000000  5172.000000  5172.000000  5172.000000  5172.000000
mean      6.640565     6.188128     5.143852     3.075599     3.124710
std      11.745009     9.534576    14.101142     6.045970     4.680522
min       0.000000     0.000000     1.000000     0.000000     0.000000
25%       0.000000     1.000000     1.000000     0.000000     1.000000
50%       3.000000     3.000000     1.000000     1.000000     2.000000
75%       8.000000     7.000000     4.000000     3.000000     4.000000
max     210.000000   132.000000   344.000000    89.000000    47.000000

                of            a          you          hou           in  ...  \
count  5172.000000  5172.000000  5172.000000  5172.000000  5172.000000  ...
mean      2.627030    55.517401     2.466551     2.024362    10.600155  ...
std       6.229845    87.574172     4.314444     6.967878    19.281892  ...
min       0.000000     0.000000     0.000000     0.000000     0.000000  ...
25%       0.000000    12.000000     0.000000     0.000000     1.000000  ...
50%       1.000000    28.000000     1.000000     0.000000     5.000000  ...
75%       2.000000    62.250000     3.000000     1.000000    12.000000  ...
max      77.000000  1898.000000    70.000000   167.000000   223.000000  ...

          connevey          jay       valued          lay  infrastructure  \
count  5172.000000  5172.000000  5172.000000  5172.000000     5172.000000
mean      0.005027     0.012568     0.010634     0.098028        0.004254
std       0.105788     0.199682     0.116693     0.569532        0.096252
min       0.000000     0.000000     0.000000     0.000000        0.000000
25%       0.000000     0.000000     0.000000     0.000000        0.000000
50%       0.000000     0.000000     0.000000     0.000000        0.000000
75%       0.000000     0.000000     0.000000     0.000000        0.000000
max       4.000000     7.000000     2.000000    12.000000        3.000000

          military     allowing           ff          dry   Prediction
count  5172.000000  5172.000000  5172.000000  5172.000000  5172.000000
mean      0.006574     0.004060     0.914733     0.006961     0.290023
std       0.138908     0.072145     2.780203     0.098086     0.453817
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     1.000000     0.000000     1.000000
max       4.000000     3.000000   114.000000     4.000000     1.000000

[8 rows x 3001 columns]
(5172, 3000)
(5172,)
Dataset shape: (5172, 3002)
Sample of combined text:
0    0 0 1 0 0 0 2 0 0 0 0 1 0 0 2 0 0 0 0 0 0 0 0 ...
1    8 13 24 6 6 2 102 1 27 18 21 13 0 1 61 4 2 0 0...
2    0 0 1 0 0 0 8 0 0 4 2 0 0 0 8 0 0 0 0 0 0 2 0 ...
3    0 5 22 0 5 1 51 2 10 1 5 9 2 0 16 2 0 0 1 1 0 ...
4    7 6 17 1 5 2 57 0 9 3 12 2 2 0 30 8 0 0 2 0 0 ...
dtype: object
Target variable distribution:
Prediction
0    3672
1    1500
Name: count, dtype: int64
```

In [50]:

```python
# Step 2: Pre-processing and TF-IDF Vectorization
tfidf = TfidfVectorizer(max_features=1000, stop_words='english')
X_tfidf = tfidf.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.3, random_state=42)

print("TF-IDF Features shape:", X_tfidf.shape)
print("\
Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

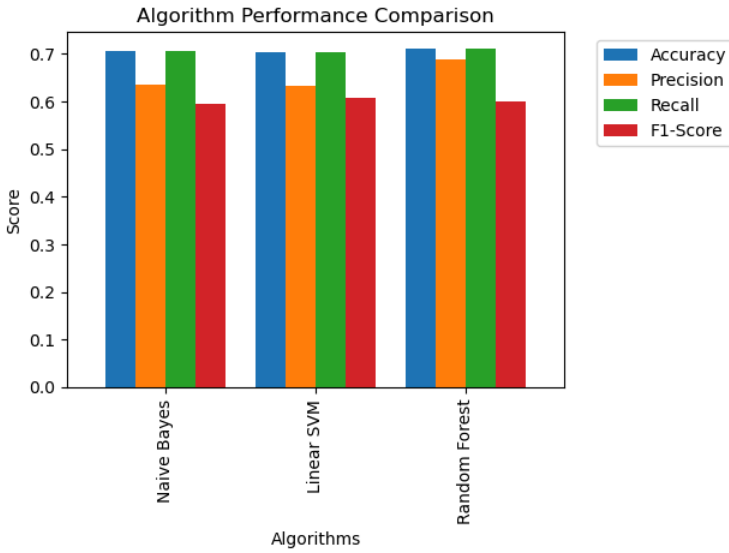
```
TF-IDF Features shape: (5172, 694)
Training set shape: (3620, 694)
Testing set shape: (1552, 694)
```

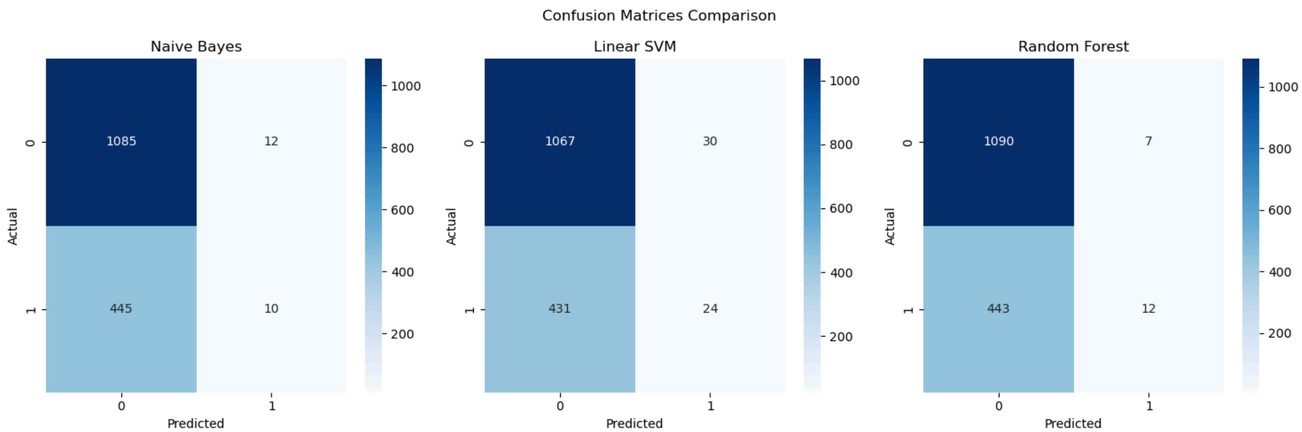
In [52]:
```
# Step 4: Apply Naive Bayes
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Make predictions
y_pred = nb_classifier.predict(X_test)

# Step 5: Generate Confusion Matrix and Classification Report
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
print("\
Classification Report:")
print(class_report)
```

```
Confusion Matrix:
[[1085   12]
 [ 445   10]]
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.99      0.83      1097
           1       0.45      0.02      0.04       455

    accuracy                           0.71      1552
   macro avg       0.58      0.51      0.43      1552
weighted avg       0.63      0.71      0.60      1552
```

In [54]:
```
# Import necessary libraries
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize classifiers
classifiers = {
    'Naive Bayes': MultinomialNB(),
    'Linear SVM': LinearSVC(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
}

# Dictionary to store results
results = {
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1-Score': []
}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    results['Accuracy'].append(accuracy_score(y_test, y_pred))
    results['Precision'].append(precision_score(y_test, y_pred, average='weighted'))
    results['Recall'].append(recall_score(y_test, y_pred, average='weighted'))
    results['F1-Score'].append(f1_score(y_test, y_pred, average='weighted'))

# Create comparison DataFrame
comparison_df = pd.DataFrame(results, index=classifiers.keys())
print("Algorithm Comparison:")
print(comparison_df)

# Create visualization
plt.figure(figsize=(12, 6))
comparison_df.plot(kind='bar', width=0.8)
plt.title('Algorithm Performance Comparison')
plt.xlabel('Algorithms')
plt.ylabel('Score')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Create confusion matrices for each classifier
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Confusion Matrices Comparison')

for i, (name, clf) in enumerate(classifiers.items()):
    y_pred = clf.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    sns.heatmap(cm, annot=True, fmt='d', ax=axes[i], cmap='Blues')
    axes[i].set_title(name)
    axes[i].set_xlabel('Predicted')
    axes[i].set_ylabel('Actual')

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

```
Algorithm Comparison:
               Accuracy  Precision    Recall  F1-Score
Naive Bayes    0.705541   0.634508  0.705541  0.596160
Linear SVM     0.702964   0.633761  0.702964  0.608909
Random Forest  0.710052   0.687733  0.710052  0.600733
<Figure size 1200x600 with 0 Axes>
```

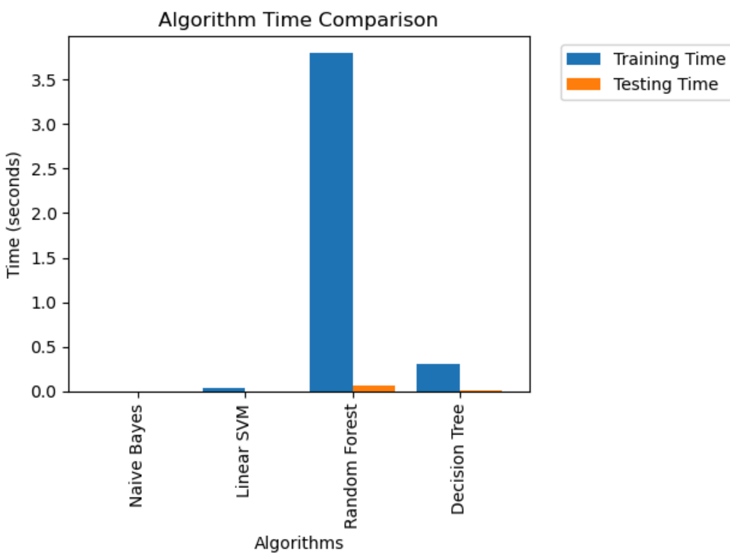




In [58]:
```
# Import additional libraries
from sklearn.tree import DecisionTreeClassifier
import time
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np

# Initialize classifiers including J48 (Decision Tree)
classifiers = {
    'Naive Bayes': MultinomialNB(),
    'Linear SVM': LinearSVC(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42)
}

# Dictionary to store results
results = {
    'Accuracy': [],
    'Training Time': [],
    'Testing Time': [],
    'Error Rate': []
}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    # Training time
    train_start = time.time()
    clf.fit(X_train, y_train)
    train_time = time.time() - train_start

    # Testing time
    test_start = time.time()
    y_pred = clf.predict(X_test)
    test_time = time.time() - test_start

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    error_rate = 1 - accuracy
```

```
    # Store results
    results['Accuracy'].append(accuracy)
    results['Training Time'].append(train_time)
    results['Testing Time'].append(test_time)
    results['Error Rate'].append(error_rate)



# Create time comparison plot
plt.figure(figsize=(12, 6))
comparison_df[['Training Time', 'Testing Time']].plot(kind='bar', width=0.8)
plt.title('Algorithm Time Comparison')
plt.xlabel('Algorithms')
plt.ylabel('Time (seconds)')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

<Figure size 1200x600 with 0 Axes>



In [ ]:
```
The Random Forest classifier achieved the best accuracy (71.01%) with the highest training time, while Naive Bayes provided a good balance between acc
```