Synopsis of The Mini Project:

Project overview: Email spam or ham detection involves building a machine learning model to classify emails as either "spam" (unwanted, unsolicited messages) or "ham" (legitimate, non-spam messages). This is an essential application of Natural Language Processing (NLP) and classification algorithms, aimed at improving email filtering and reducing the impact of spam on users.

Objectives:

- **Build a spam detection model**: Develop a machine learning model to distinguish between spam and ham emails.
- **Automate email classification**: Automate the process of sorting spam from legitimate emails.
- **Improve email security**: Reduce the risk of phishing, malware, and unwanted promotions through an accurate spam detection system.
- **Compare models**: Evaluate the performance of different classifiers (e.g., SVM, Logistic Regression, Random Forest) to identify the best-performing model.

Datasets:

1. **SpamAssassin Public Dataset**: A widely-used dataset for spam detection, containing a balanced set of ham and spam emails.

Steps to Implement the Project:

1. **Data Collection and Loading:** Importing a dataset containing email messages labeled as spam or ham (not spam).

2. **Data Preprocessing:** Prepare the dataset for machine learning by handling missing values, normalizing text data, and encoding categorical variables.

3. **Model Selection:** Choose appropriate machine learning algorithms for classifying emails as spam or ham.Implement various classifiers such as Naive Bayes, SVM, Logistic Regression, Decision Trees, and Random Forest.

4. **Model Evaluation:** Objective: Evaluate the performance of different models using relevant metrics.

5. **Visualization**: Create visual representations of model performance and spam detection results.

6. **Spam Detection**: Deploy the best-performing model to classify new emails as spam or ham.Create a function that takes a new email message as input and returns the classification.

Classification : Email ham or spam detection

Data exploration, data preprocessing, data cleaning

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Load your dataset
df = pd.read_csv('email.csv')
```

DETECTING MISSING VALUES

```
print(df.head())
df.info()
miss=df.isna().sum()
print("\nMissing Values:\n",miss)
df.dropna(inplace=True)
print("\nMissing Values:\n",df.isna().sum())


# Assuming the dataset has two columns: 'Message' and 'Category'
('spam' or 'ham')
# Map 'spam' to 1 and 'ham' to 0
df['Category'] = df['Category'].map({'spam': 1, 'ham': 0})

# Check for missing values in the target variable
missing_values = df['Category'].isnull().sum()
print(f'Missing values in Category: {missing_values}')

# Drop rows with missing values in 'Category'
df.dropna(subset=['Category'], inplace=True)

# Extract features (X) and labels (y)
X = df['Message']  # Assuming 'Message' is the column name for the
email text
y = df['Category']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Convert text data to numerical data using CountVectorizer
```

```python
vectorizer = CountVectorizer()
X_train_transformed = vectorizer.fit_transform(X_train)
X_test_transformed = vectorizer.transform(X_test)

# Train a Naive Bayes classifier
model = MultinomialNB()
model.fit(X_train_transformed, y_train)

# Predict on the test data
y_pred = model.predict(X_test_transformed)

# Calculate accuracy and print classification report
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred))

# Function to predict if a message is spam or ham
def predict_message(message):
    transformed_message = vectorizer.transform([message])
    prediction = model.predict(transformed_message)
    return 'spam' if prediction[0] == 1 else 'ham'

# Example usage
message = input("Enter a message to classify as spam or ham: ")
print(f'The message is classified as: {predict_message(message)}')
```

OUTPUT

```
  Category                                            Message
0      ham  Go until jurong point, crazy.. Available only ...
1      ham                      Ok lar... Joking wif u oni...
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham  U dun say so early hor... U c already then say...
4      ham  Nah I don't think he goes to usf, he lives aro...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5573 entries, 0 to 5572
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Category  5573 non-null   object
 1   Message   5573 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB

Missing Values:
 Category    0
Message     0
dtype: int64

Missing Values:
 Category    0
Message     0
```

```
dtype: int64
Missing values in Category: 1
Accuracy: 0.9919282511210762
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       966
         1.0       1.00      0.94      0.97       149

    accuracy                           0.99      1115
   macro avg       1.00      0.97      0.98      1115
weighted avg       0.99      0.99      0.99      1115

Enter a message to classify as spam or ham: Ok lar
The message is classified as: ham
```

ALGORITHM:

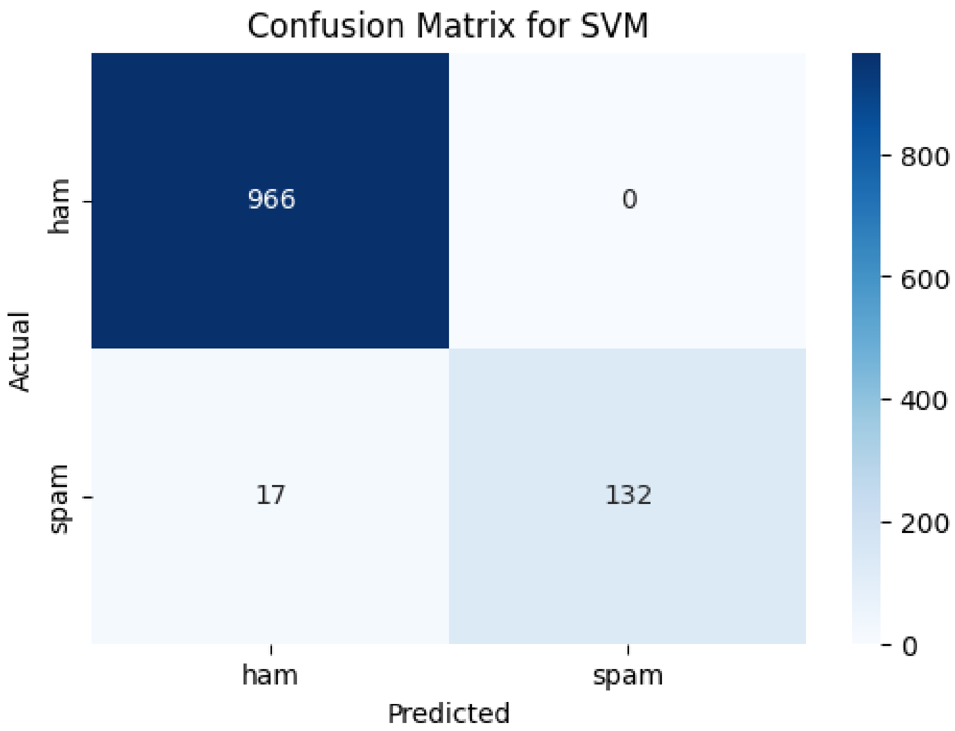
- KNN
- SVM
- NAIVE BAYES

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assume `results` is a dictionary that stores model performance from
previous steps.
# Example:
# results = {
#     'Naive Bayes': {'accuracy': 0.95, 'y_pred': y_pred_nb},
#     'SVM': {'accuracy': 0.93, 'y_pred': y_pred_svm},
#     'KNN': {'accuracy': 0.91, 'y_pred': y_pred_knn},
#     # Add other models...
# }

# Model names and accuracy values
model_names = list(results.keys())
accuracy_scores = [results[model]['accuracy'] for model in results]

# Bar Plot - Accuracy Comparison
plt.figure(figsize=(10, 6))
sns.barplot(x=model_names, y=accuracy_scores, palette='magma')
plt.title('Accuracy Comparison of Different Models', fontsize=14)
plt.ylabel('Accuracy', fontsize=12)
plt.xlabel('Models', fontsize=12)
plt.xticks(rotation=45)
plt.show()

# Scatter Plot - True vs Predicted Values for a specific model (e.g.,
Naive Bayes)
# You can change the model to visualize for another one

y_test = np.array(y_test)  # Convert y_test to numpy array if it's a
Pandas Series
y_pred_nb = results['Naive Bayes']['y_pred']  # Assuming Naive Bayes
is one of the models

plt.figure(figsize=(8, 6))
plt.scatter(range(len(y_test)), y_test, color='blue', label='True
Labels', alpha=0.6)
```

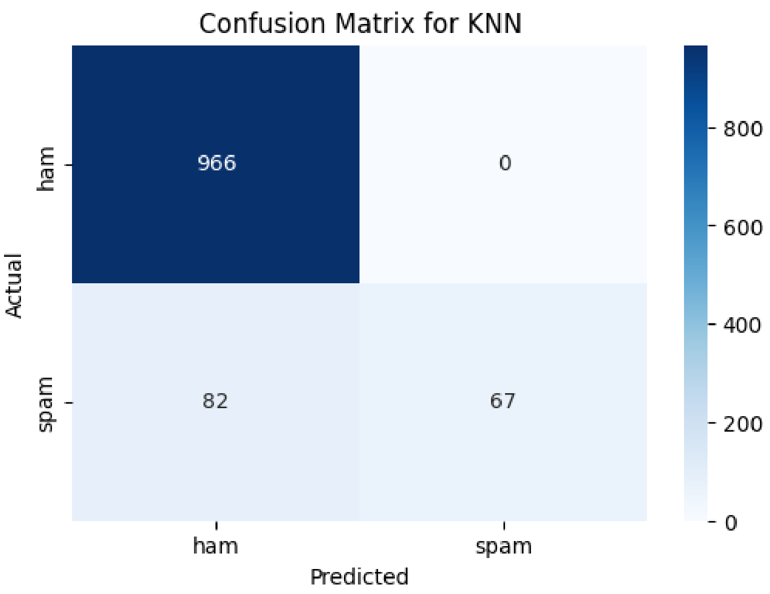
```python
plt.scatter(range(len(y_pred_nb)), y_pred_nb, color='red',


label='Predicted Labels (Naive Bayes)', alpha=0.6)
plt.title('True vs Predicted Labels (Naive Bayes)', fontsize=14)
plt.xlabel('Samples', fontsize=12)
plt.ylabel('Spam/Ham (1 = Spam, 0 = Ham)', fontsize=12)
plt.legend()
plt.show()
```

OUTPUT :

```
Category                                              Message
0      ham  Go until jurong point, crazy.. Available only ...
1      ham                      Ok lar... Joking wif u oni...
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham  U dun say so early hor... U c already then say...
4      ham  Nah I don't think he goes to usf, he lives aro...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5573 entries, 0 to 5572
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Category  5573 non-null   object
 1   Message   5573 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB

Missing Values:
 Category    0
Message     0
dtype: int64

Missing Values after dropping:
 Category    0
Message     0
dtype: int64
Missing values in y: 1

--- Naive Bayes ---
Accuracy: 0.9919282511210762
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       966
         1.0       1.00      0.94      0.97       149

    accuracy                           0.99      1115
   macro avg       1.00      0.97      0.98      1115
weighted avg       0.99      0.99      0.99      1115




--- SVM ---
Accuracy: 0.9847533632286996
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       966
         1.0       1.00      0.89      0.94       149

    accuracy                           0.98      1115
   macro avg       0.99      0.94      0.97      1115
weighted avg       0.99      0.98      0.98      1115
```




```
--- KNN ---
Accuracy: 0.9264573991031391
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96       966
         1.0       1.00      0.45      0.62       149

    accuracy                           0.93      1115
   macro avg       0.96      0.72      0.79      1115
weighted avg       0.93      0.93      0.91      1115
```

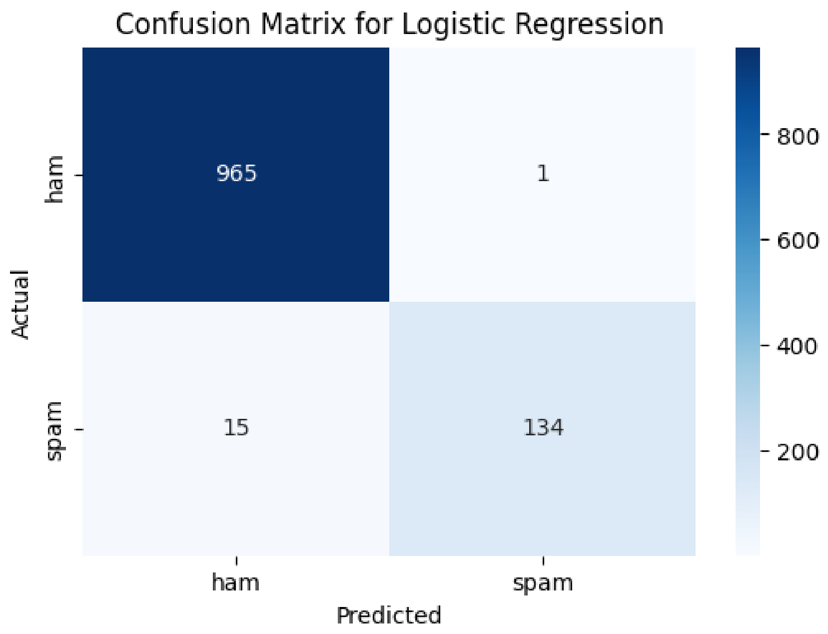


LOGISTIC REGRESSION

```
# Logistic Regression Model
log_reg = LogisticRegression(max_iter=1000)

# Train the model
log_reg.fit(X_train_transformed, y_train)

# Predict
y_pred_logreg = log_reg.predict(X_test_transformed)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print(f"\n--- Logistic Regression ---")
print(f"Accuracy: {accuracy_logreg}")
print(classification_report(y_test, y_pred_logreg))

# Plot Confusion Matrix for Logistic Regression
plot_confusion_matrix(y_test, y_pred_logreg, "Logistic Regression")
```

OUTPUT

```
--- Logistic Regression ---
Accuracy: 0.9856502242152466
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       966
         1.0       0.99      0.90      0.94       149

    accuracy                           0.99      1115
   macro avg       0.99      0.95      0.97      1115
weighted avg       0.99      0.99      0.99      1115
```

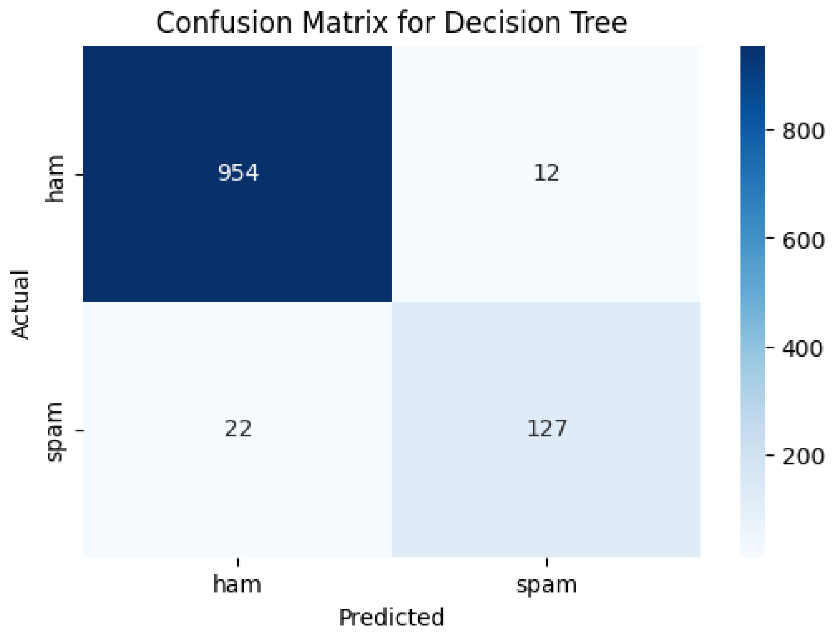


DECISION TREE CLASSIFIER

```
# Decision Tree Model
decision_tree = DecisionTreeClassifier()

# Train the model
decision_tree.fit(X_train_transformed, y_train)

# Predict
y_pred_dt = decision_tree.predict(X_test_transformed)

# Evaluate the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"\n--- Decision Tree ---")
print(f"Accuracy: {accuracy_dt}")
print(classification_report(y_test, y_pred_dt))

# Plot Confusion Matrix for Decision Tree
plot_confusion_matrix(y_test, y_pred_dt, "Decision Tree")
```

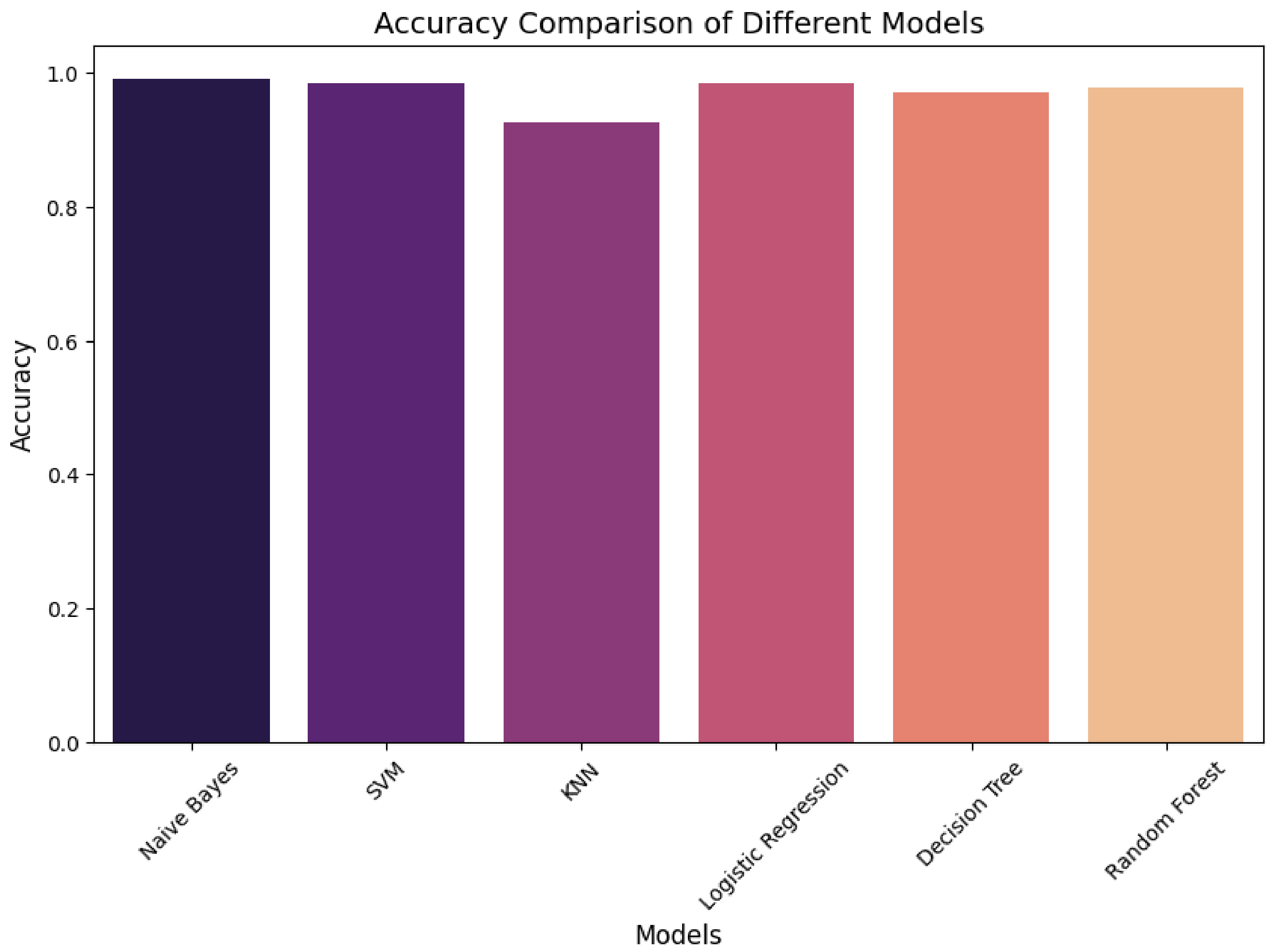
OUTPUT

```
--- Decision Tree ---
Accuracy: 0.9695067264573991
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.98       966
         1.0       0.91      0.85      0.88       149

    accuracy                           0.97      1115
   macro avg       0.95      0.92      0.93      1115
weighted avg       0.97      0.97      0.97      1115
```



```python
model_names = list(results.keys())
accuracy_scores = [results[model]['accuracy'] for model in results]

# Bar Plot - Accuracy Comparison
plt.figure(figsize=(10, 6))
sns.barplot(x=model_names, y=accuracy_scores, palette='magma')
plt.title('Accuracy Comparison of Different Models', fontsize=14)
plt.ylabel('Accuracy', fontsize=12)
plt.xlabel('Models', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```

OUTPUT:



EXAMPLE USAGE

```
def predict_message(message):
  transformed_message = vectorizer.transform([message])
  prediction = model.predict(transformed_message)
  return 'spam' if prediction[0] == 1 else 'ham'
   # Example usage
message = input("Enter a message to classify as spam or ham: ")
print(f'The message is classified as: {predict_message(message)}')
```

OUTPUT:

```
Enter a message to classify as spam or ham: OK LAR
The message is classified as: ham
```