Project Report: Email Spam Detection

Introduction

This project aims to build a predictive system for email spam detection using a dataset obtained from Kaggle. The dataset contains various emails labeled as spam or not spam. The following steps outline the process taken to achieve an accurate and reliable spam detection model.

Dataset

- **Source**: [Kaggle Email Spam Dataset]
- **Content**: Emails labeled as spam or not spam.

Data Collection and Pre-processing:

Importing the Dataset:

```
[2]: full_data = pd.read_csv("mail_data.csv")
```

```
[3]: full_data.head(5)
```

Data Cleaning:

- **Label Encoding**:
  - Converted categorical labels (spam/not spam) into numerical format.

```
# label spam mail as 0 ; and ham as 1;
mail_data['Category'] = mail_data['Category'].map({'spam': 0, 'ham': 1})
```

```
spam - 0
ham - 1
```

Handling Missing Values:

- Checked for and handled any missing values in the dataset.

```
[10]: mail_data.isnull().sum()
```

```
[10]: Category    0
      Message     0
      dtype: int64
```

Removing Duplicates:

- Identified and removed any duplicate records.

```
mail_data.drop_duplicates()
```

	Category	Message
0	1	Go until jurong point, crazy.. Available only ...
1	1	Ok lar... Joking wif u oni...
2	0	Free entry in 2 a wkly comp to win FA Cup fina...
3	1	U dun say so early hor... U c already then say...

Exploratory Data Analysis (EDA)

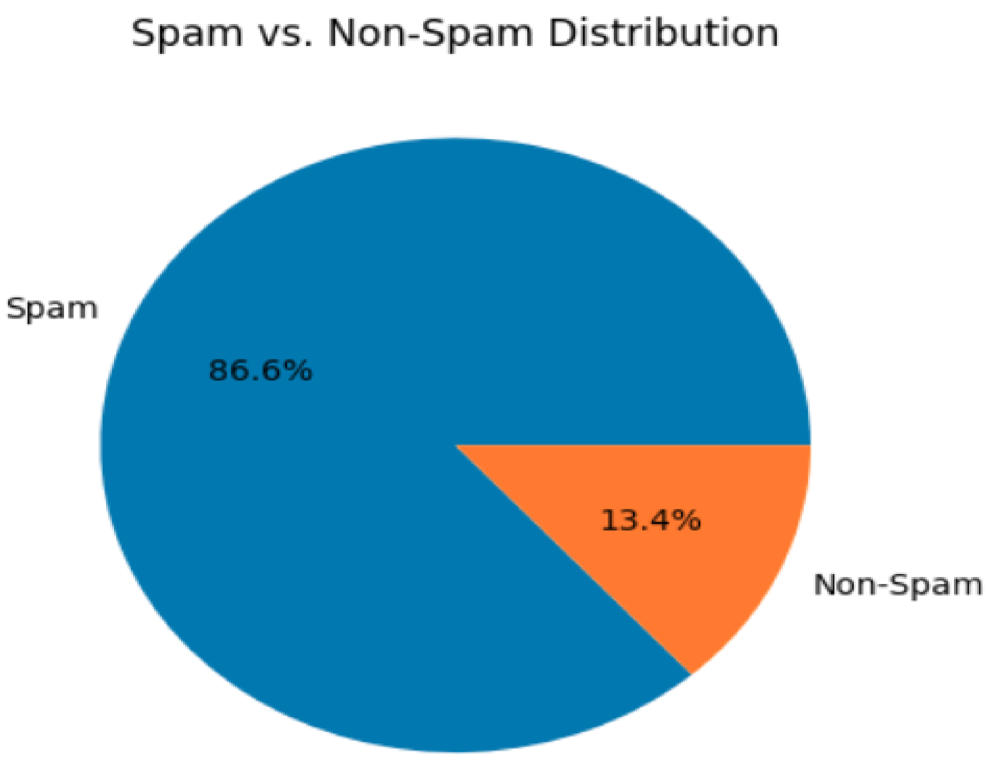
Pie Chart of Spam vs. Not Spam:

- Visualized the distribution of spam and not spam emails.

```
# Spam or non-spam pie chart
import matplotlib.pyplot as plt

# a DataFrame 'mail_data' with a 'Category' column (0 for spam, 1 for non-spam)
spam_counts = mail_data['Category'].value_counts()
labels = ['Spam', 'Non-Spam']

plt.pie(spam_counts, labels=labels, autopct='%1.1f%%')
plt.title('Spam vs. Non-Spam Distribution')
plt.show()
```



Character, Word, and Sentence Count:

- Analysed the text data to understand its structure.

```
def get_character_count(text):
    return len(text)

def get_word_count(text):
    return len(text.split())

def get_sentence_count(text):
    sentences = text.split('.')
    return len(sentences)

user_value = int(input("Enter value to see "))
email_text = mail_data['Message'].iloc[user_value]
char_count = get_character_count(email_text)
word_count = get_word_count(email_text)
sentence_count = get_sentence_count(email_text)

print(f"Character count: {char_count}")
print(f"Word count: {word_count}")
print(f"Sentence count: {sentence_count}")
```

```
Enter value to see  20
Character count: 41
Word count: 8
Sentence count: 1
```

Data Splitting

- Split the dataset into training and testing sets.

```
# seprating data and label
X = mail_data['Message']
Y = mail_data['Category']
```

```
print(X)
print("---"*20)
print(Y)
```

```
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                              ...
5567    This is the 2nd time we have tried 2 contact u...
5568                 Will ü b going to esplanade fr home?
5569    Pity, * was in mood for that. So...any other s...
5570    The guy did some bitching but I acted like i'd...
5571                           Rofl. Its true to its name
Name: Message, Length: 5572, dtype: object
------------------------------------------------------------
0       1
1       1
2       0
3       1
```

Feature Extraction

- Used TF-IDF (Term Frequency-Inverse Document Frequency) for feature extraction.

```
# transfer the text data into feature vector so that it can be used in logistic regression
feature_extraction = TfidfVectorizer(min_df = 1 , stop_words = 'english' , lowercase = True)
```

```
feature_extraction
```

```
TfidfVectorizer
TfidfVectorizer(stop_words='english')
```

```
X_train_features = feature_extraction.fit_transform(X_train)
```

```
X_test_features = feature_extraction.transform(X_test)
```

```
# Convert Y_train and Y_test to integers
Y_train  =  Y_train.astype('int')
Y_test = Y_test.astype('int')
```

```
print("X_train feature data is " , X_train_features)
```

Model Training and Evaluation

Logistic Regression:

- Trained a logistic regression model.

```
model = LogisticRegression()
```

```
model.fit(X_train_features , Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation ¶

```
# prediciton on training data
prediction_on_training_data = model.predict(X_train_features)
```

```
Accuracy = accuracy_score(prediction_on_training_data , Y_train )
```

```
print("Accuracy of the model is", Accuracy)
```

```
Accuracy of the model is 0.9676912721561588
```

prediciton on testing data

```
# prediciton on testing data
prediction_on_testing_data = model.predict(X_test_features)
```

```
Accuracy = accuracy_score(prediction_on_testing_data , Y_test)
print("Accuracy of the model is", Accuracy)
```

```
Accuracy of the model is 0.9605381165919282
```

Decision Tree Classifier:

- Trained a decision tree model.

```
clf = DecisionTreeClassifier(criterion='gini', min_samples_leaf=30, random_state=0)
clf.fit(X_train_features, Y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(min_samples_leaf=30, random_state=0)
```

```
# using training data
y_pred = clf.predict(X_train_features)
```

```
Accuracy = accuracy_score(y_pred , Y_train )
```

```
print("Accuracy of the model is", Accuracy)
```

```
Accuracy of the model is 0.941440430783038
```

```
# using testing data
y_pred = clf.predict(X_test_features)
```

```
Accuracy = accuracy_score(y_pred , Y_test )
```

```
print("Accuracy of the model is", Accuracy)
```

```
Accuracy of the model is 0.9363228699551569
```

Predictive System:

- Built a simple predictive system using the logistic regression model.

```
input_mail = ["""Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to
receive entry question(std txt rate)T&C's apply 08452810075over18's"""]

input_data_features  = feature_extraction.transform(input_mail)

prediction = model.predict(input_data_features)
# print(prediction)

if prediction[0] == 1:
    print('Ham Mail')
else:
    print('Spam Mail')
```

Conclusion:

The logistic regression model achieved a high accuracy of 96%, making it a reliable choice for the spam detection system.

The project successfully implemented data cleaning, feature extraction, model training, and evaluation to build an effective email spam detection system. Further improvements can be explored by testing other algorithms and fine-tuning the existing models.