Email Spam classifier

1.Feature Engineering :

load data into a data frame, to get a quick overview of the structure and content of the dataset

	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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

Exploratory Data Analysis :

```
df.shape
```

```
(5572, 2)
```

The number of missing values :

```
Category    0
Message     0
dtype: int64
```

⇒Based on this analysis, we can conclude that there are no missing values in either the "Category" or "Message" columns of the dataset. This means that each entry in these columns has complete data, and no additional steps are needed to handle missing values for these fields.

Display information about the DataFrame :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

⇒The DataFrame appears to be well-structured, containing complete data without any missing values.The data types are appropriate for the respective columns, and the memory usage is reasonable for the dataset size.

Display description about the DataFrame

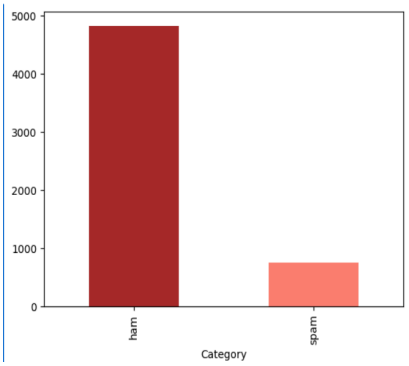
	Category	Message
count	5572	5572
unique	2	5157
top	ham	Sorry, I'll call later
freq	4825	30

⇒Category Column :

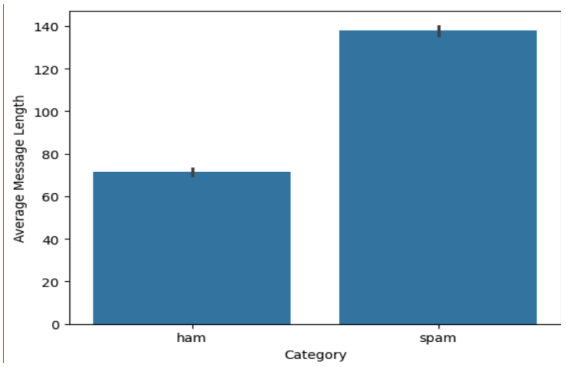
Number: The ‘Category’ column contains 5572 entries.
Unique: There are 2 unique categories in the ‘Category’ column.
Top: the most frequent category is ‘ham’, appearing 4825 times.
Frequency: ‘ham’ appears 4825 times in the ‘Category’ column.
Message’ column:

Number: There are 5572 entries in the ‘Message’ column.
Unique: There are 5157 unique messages in the ‘Message’ column.
Top: The most frequent message is ‘Sorry, I'll call later’, appearing 30 times.
Frequency: The message ‘Sorry, I'll call later’ appears 30 times in the ‘Message’ column.
⇒The ‘Category’ column contains categorical data, representing labels such as ‘ham’ (non-spam) and ‘spam’.
The ‘Message’ column contains textual data, with a wide variety of unique messages.
count the frequency of each unique value in the 'Category' column

```
Category
ham     4825
spam     747
Name: count, dtype: int64
```



Calculate message lengths:



⇒After analyzing the message lengths in the dataset, we found that the average message length for 'ham' (non-spam) messages is approximately 75 characters, while the average message length for 'spam' messages is around 140 characters. This indicates that 'spam' messages tend to be longer on average compared to 'ham' messages. Understanding these

differences in message length can provide insights into the characteristics of the messages in our dataset and may be valuable for further analysis and modeling.
Generate word cloud for 'Message' column

⇒A word cloud is a visual representation of textual data, where the size of each word indicates its frequency or importance in the data set. In a word cloud, words are generally arranged randomly or in clusters, with more frequent words appearing larger and more prominent, while less frequent words are smaller.
variable encoding:

	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...
4	1	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	0	This is the 2nd time we have tried 2 contact u...
5568	1	Will ü b going to esplanade fr home?
5569	1	Pity, * was in mood for that. So...any other s...
5570	1	The guy did some bitching but I acted like i'd...
5571	1	Rofl. Its true to its name

[5572 rows x 2 columns]

0⇒Spam
1⇒ham(Not Spam)

2.Model Engineering

Data partitioning:

To prepare our data for training, we split our dataset into training and test sets using the train_test_split function in the scikit-learn library.
Training set (X_train, Y_train):
X_train: Set of features used to train our model.
Y_train: Set of labels corresponding to the training set, used to train the model to make predictions.
Test set (X_test, Y_test) :
X_test: Set of features used to evaluate the performance of our model on unseen data.
Y_test: Set of labels corresponding to the test set, used to evaluate the model's predictions.
We used the following parameters for splitting the data: test_size: 0.2, which means that 20% of the data is reserved for the test set.
random_state: 3, used to guarantee the reproducibility of the results.

```
print(X.shape)
print(X_train.shape)
print(X_test.shape)
```

```
(5572,)
(4457,)
(1115,)
```

```
print(Y.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(5572,)
(4457,)
(1115,)
```

vectorisation of text or extraction of text features:

Most machine learning algorithms cannot process text data directly. They need numerical data as input. Text vectorisation using techniques such as TF-IDF transforms text documents into numerical vectors that can be used by machine learning algorithms.

Vectorization of text refers specifically to the process of converting text documents into digital vectors (vecteurs numériques). It is part of text feature extraction.
Extraction of text features is a more general term that includes vectorisation and other techniques for obtaining meaningful features from text data.

TF-IDF Vectorization

```
feature_extraction=TfidfVectorizer(min_df=1,stop_words='english',lowercase=True)
```

TfidfVectorizer: This is a tool from the sklearn.feature_extraction.text module. It transforms text data into numerical data that machine learning models can process. Specifically, it converts a collection of raw documents to a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features.

min_df=1: This means that terms (words) must appear in at least 1 document to be included in the vocabulary. It helps to exclude extremely rare terms that might not be informative.
stop_words='english': This tells the vectorizer to remove common English stopwords (like "the", "is", "and", etc.) that are unlikely to be useful for text analysis.
lowercase=True: This parameter ensures that all text is converted to lowercase before processing. This helps in treating "Hello" and "hello" as the same word.

Fitting and Transforming Training Data

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

fit_transform(X_train): This does two things:

Fit: It learns the vocabulary and inverse document frequency (IDF) from the X_train data. Essentially, it identifies the important words in the training data.
Transform: It then transforms the training data into a TF-IDF matrix. Each row of this matrix represents a document, and each column represents a term from the vocabulary. The values are the TF-IDF scores for each term in each document.

Transforming Testing Data:

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

transform(X_test): This transforms the X_test data into the TF-IDF matrix using the vocabulary and IDF learned from the training data. It ensures that the test data is transformed in the same way as the training data.

Converting Target Variables:

```
Y_train=Y_train.astype('int')
Y_test=Y_test.astype('int')
```

astype('int'): This converts the target variables Y_train and Y_test to integer type. This step ensures that the labels are in the correct format for training the classifier model.

Calcul de l'IDF

La formule de l'IDF est la suivante :

$$\text{IDF}(t) = \log\left(\frac{N}{1+n_t}\right)$$

où :

- t est un terme (un mot ou une phrase).
- N est le nombre total de documents dans le corpus.
- n_t est le nombre de documents contenant le terme t.
- L'addition de 1 dans le dénominateur ($n_t + 1$) est souvent utilisée pour éviter la division par zéro au cas où un terme n'apparaît dans aucun document.

Interprétation de l'IDF

- **Termes fréquents dans de nombreux documents** : Si un terme apparaît dans de nombreux documents, n_t sera grand, ce qui rendra le terme commun et son IDF sera faible. Cela signifie que le terme est moins informatif.
- **Termes rares dans le corpus** : Si un terme apparaît dans très peu de documents, n_t sera petit, ce qui rendra le terme rare et son IDF sera élevé. Cela signifie que le terme est plus informatif.

Utilisation dans TF-IDF

Le score TF-IDF pour un terme dans un document est calculé en multipliant la fréquence du terme dans le document (TF) par son IDF :

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

où :

- $\text{TF}(t, d)$ est la fréquence du terme t dans le document d.
- $\text{IDF}(t)$ est la fréquence inverse du document pour le terme t.

⇒Why the IDF is Important:

Noise reduction: The IDF helps to reduce the impact of common terms (such as ‘the’, ‘and’, ‘of’) which appear frequently in the corpus but are less informative.

Weighting of significant terms: Rare terms (with a high IDF) are often more informative and are weighted more heavily, helping machine learning models to better understand the content and context of documents.

Modelling with LogisticRegression:

Le modèle de régression logistique trouve les paramètres (coefficients et intercept) lors de l'entraînement en ajustant ces paramètres de manière itérative pour minimiser une fonction de coût spécifique. Le processus d'ajustement des paramètres est généralement effectué à l'aide d'une technique d'optimisation, telle que la descente de gradient ou la méthode du maximum de vraisemblance.

Voici une brève explication du processus :

1. **Initialisation des paramètres :**
   Au début, les paramètres du modèle sont initialisés avec des valeurs arbitraires ou nulles.

2. **Calcul des prédictions :**
   Pour chaque exemple dans l'ensemble de données, le modèle calcule les prédictions en utilisant les paramètres actuels. Ces prédictions sont généralement transformées à l'aide de la fonction logistique pour obtenir des probabilités.

3. **Calcul de la fonction de coût :**
   Ensuite, le modèle calcule une mesure de l'erreur entre les prédictions et les valeurs réelles à l'aide d'une fonction de coût spécifique, comme la perte logistique.

4. **Ajustement des paramètres :**
   Le modèle ajuste ensuite les paramètres pour minimiser la fonction de coût. Cela peut être fait en utilisant des techniques d'optimisation telles que la descente de gradient, où les paramètres sont mis à jour dans la direction opposée du gradient de la fonction de coût.

5. **Répétition :**
   Ce processus est répété pour un certain nombre d'itérations ou jusqu'à ce qu'un critère d'arrêt soit atteint, comme la convergence de la fonction de coût.

En ajustant itérativement les paramètres de cette manière, le modèle trouve les valeurs qui permettent de mieux prédire les étiquettes de classe en fonction des caractéristiques des données d'entraînement. Une fois l'entraînement termi ↓ les paramètres ajustés du modèle sont utilisés pour faire des prédictions sur de nouvelles données.

⇒

```python
# Afficher les coefficients (poids) associés à chaque caractéristique
print("Coefficients:", model.coef_)

# Afficher l'intercept (le terme de biais)
print("Intercept:", model.intercept_)
```

```
Coefficients: [[-0.69887844 -1.3540533   0.01797883 ...  0.01194058 -0.13745948
   0.01862544]]
Intercept: [2.42919151]
```

Supposons que nous ayons un ensemble de données avec une seule caractéristique (par exemple, le nombre d'heures d'étude) et une seule cible (par exemple, réussite ou échec d'un examen).

Voici un ensemble de données fictives :

Heures d'étude	Réussite (1) / Échec (0)
2	0
3	0
4	1
5	1
6	1

Nous voulons utiliser ces données pour prédire si un étudiant réussira ou échouera un examen en fonction du nombre d'heures d'étude.

Supposons que nous ayons ajusté un modèle de régression logistique à ces données et trouvé les paramètres suivants :

- Coefficient (poids) : 0.8
- Intercept : -2.5

Maintenant, pour faire une prédiction pour un nouvel étudiant qui a étudié pendant 4 heures, nous utilisons les paramètres du modèle comme suit :

1. **Calcul de la probabilité logarithmique :**
   $\text{Probabilité log} = -2.5 + 0.8 \times 4 = -2.5 + 3.2 = 0.7$
2. **Transformation en probabilité :**
   $\text{Probabilité} = \frac{1}{1+e^{-0.7}} = \frac{1}{1+0.5} = \frac{1}{1.5} \approx 0.67$
3. **Prédiction de classe :**
   Puisque la probabilité de réussite est d'environ 0.67, ce nouvel étudiant est classé comme réussissant l'examen.

En résumé, en utilisant les paramètres du modèle de régression logistique, nous avons calculé la probabilité que l'étudiant réussisse l'examen en fonction du nombre d'heures d'étude, puis nous avons utilisé cette probabilité pour faire une prédiction de classe.

Model evaluation :

```
Acc on training data: 0.9676912721561588
Precision on training data: 0.9650087478130467
Confusion Matrix on training data:
 [[ 452  140]
 [   4 3861]]
```

```
Acc on test data: 0.9668161434977578
Precision on test data: 0.9628886659979939
Confusion Matrix on test data:
 [[118  37]
 [  0 960]]
```

example of use

```
input_your_mail=["Congratulations! You have been selected as the lucky winner of our monthly prize draw. To claim your prize of $1000 cash, simply click
input_data_features=feature_extraction.transform(input_your_mail)
prediction=model.predict(input_data_features)
print(prediction)
if(prediction[0]==1):
  print('ham mail')
else:
  print('Spam mail')
```

```
[0]
Spam mail
```

Modelling with Decision tree :

```
Acc on training data: 1.0
Precision on training data: 1.0
Confusion Matrix on training data:
 [[ 592    0]
 [   0 3865]]
```

```
Acc on test data: 0.9668161434977578
Precision on test data: 0.9628886659979939
Confusion Matrix on test data:
 [[118  37]
 [  0 960]]
```

example of use

```
prediction=dt_model.predict(input_data_features)
print(prediction)
if(prediction[0]==1):
  print('ham mail')
else:
  print('Spam mail')
```

```
[0]
Spam mail
```