Spam Detector

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Retrieve the Data

The data is located at https://static.bc-edx.com/ai/ail-v-1-0/m13/challenge/spam-data.csv

Dataset Source: [UCI Machine Learning Library]()

Import the data using Pandas. Display the resulting DataFrame to confirm the import was successful.

In [2]:

```
# Import the data
data = pd.read_csv("https://static.bc-edx.com/ai/ail-v-1-0/m13/challenge/spam-data.csv")
data.head()
```

Out[2]:

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	

5 rows × 58 columns

Predict Model Performance

You will be creating and comparing two models on this data: a Logistic Regression, and a Random Forests Classifier. Before you create, fit, and score the models, make a prediction as to which model you think will perform better. You do not need to be correct!

Write down your prediction in the designated cells in your Jupyter Notebook, and provide justification for your educated guess.

In my point of view, the Random Forest Classifier is expected to produce a better accuracy rate compared to Logistic Regression for spam detection because of the following: Another advantage of Random Forest when dealing with data is that it does not overfit, and it is also more capable of handling nonlinear data compared to Logistic Regression. Another prediction that can be made is based on several explanatory factors: Random Forest handling of high-dimensional and unbalanced data sets, typical of text-based spam classification.

Split the Data into Training and Testing Sets

In [10]:

```
# Create the labels set `y` and features DataFrame `X`
X = data.drop('spam', axis=1)
y = data['spam']
```

In [11]:

```
# Check the balance of the labels variable (`y`) by using the `value_counts` function.
y.value_counts()
```

Out[11]:

```
spam
0    2788
1    1813
Name: count, dtype: int64
```

In [13]:

```
# Split the data into X_train, X_test, y_train, y_test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=4
2)
X_train.head()
```

Out[13]:

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	w
1370	0.09	0.0	0.09	0.0	0.39	0.09	0.09	
3038	0.00	0.0	0.00	0.0	0.00	0.00	0.00	
2361	0.00	0.0	2.43	0.0	0.00	0.00	0.00	
156	0.00	0.0	0.00	0.0	1.31	0.00	1.31	
2526	0.00	0.0	0.00	0.0	0.00	0.00	0.00	

5 rows × 57 columns

Scale the Features

Use the `StandardScaler` to scale the features data. Remember that only `X_train` and `X_test` DataFrames should be scaled.

In [14]:

```
from sklearn.preprocessing import StandardScaler

# Create the StandardScaler instance
scaler = StandardScaler()
```

In [21]:

```
# Fit the Standard Scaler with the training data
scaler.fit(X_train)
scaler.fit(X_test)
```

Out[21]:

```
▼  StandardScaler i ?

StandardScaler()
```

In [22]:

```
# Scale the training data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled=scaler.fit_transform(X_test)
```

Create and Fit a Logistic Regression Model

Create a Logistic Regression model, fit it to the training data, make predictions with the testing data, and print the model's accuracy score. You may choose any starting settings you like.

In [23]:

```
# Train a Logistic Regression model and print the model score
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
```

Out[23]:

```
▼  LogisticRegression i ?

LogisticRegression()
```

In [31]:

```
# Make and save testing predictions with the saved logistic regression model using the test data
testing_predictions = logistic_model.predict(X_test_scaled)
# Review the predictions
testing_predictions[-5:]
```

Out[31]:

```
array([1, 0, 0, 0, 0])
```

In [25]:

```
# Calculate the accuracy score by evaluating `y_test` vs. `testing_predictions`.
accuracy_score(y_test, testing_predictions)
```

Out[25]:

```
0.9131378935939196
```

Create and Fit a Random Forest Classifier Model

Create a Random Forest Classifier model, fit it to the training data, make predictions with the testing data, and print the model's accuracy score. You may choose any starting settings you like.

In [27]:

```
# Train a Random Forest Classifier model and print the model score
from sklearn.ensemble import RandomForestClassifier
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=1)
random_forest_model.fit(X_train_scaled, y_train)
```

Out[27]:

```
▼      RandomForestClassifier     i ?

RandomForestClassifier(random_state=1)
```

In [33]:

```
# Make and save testing predictions with the saved logistic regression model using the test data

testing_predictions1 = random_forest_model.predict(X_test_scaled)
# Review the predictions
testing_predictions1[-5:]
```

Out[33]:

```
array([1, 0, 0, 0, 0])
```

In [36]:

```
# Calculate the accuracy score by evaluating `y_test` vs. `testing_predictions`.
accuracy_score(y_test, testing_predictions1)
```

Out[36]:

```
0.9207383279044516
```

Evaluate the Models

Which model performed better? How does that compare to your prediction? Write down your results and thoughts in the following markdown cell.

The Random Forest Classifier was slightly more accurate than the Logistic Regression with an accuracy of 92.31%, which also supports my assumption about its ability to process spam detection data. This argument means that although Random Forest outperforms other models in terms of robustness, Logistic Regression is also efficient and can be improved with additional adjustments like hyperparameter tuning and feature extraction.