Email Classification and Prediction using Machine Learning

Submitted By: -

Steps to be followed:

1. Data loading and understanding
2. Data Cleaning
3. Data preprocessing
4. Data Visualization (EDA)
5. Feature extraction
6. Topic modelling
7. Model building
8. Model Evaluation
9. Prediction of Emails

In [1]:

```
# Importing necessary Libraries
import pandas as pd
import numpy as np

# Importing NLTK Library for Text processing
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
from collections import Counter
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

# Libraries for Visualisation
import plotly.graph_objects as go
from sklearn.model_selection import train_test_split

# For conversion of text
from sklearn.feature_extraction.text import TfidfVectorizer

# Machine Learning Models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [2]:

```
# Importing the data
data = pd.read_csv(r"C:\Users\Tanmayee\OneDrive\Documents\Personal\Other\Coventry\Bhanu\mail_data.csv
data
```

Out[2]:

	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

5572 rows × 2 columns

In [3]:

```
# Finding out number of rows and columns in dataset
data.shape
```

Out[3]:

```
(5572, 2)
```

In [4]:

```
# Type of data
data.info()
```

```
<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
```

In [5]:

```
data.describe()
```

Out[5]:

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

In [6]:

```
# Check for null values in the DataFrame
null_values = data.isnull().sum()

# Display the columns with null values
print("Columns with null values:")
print(null_values[null_values > 0])
```

```
Columns with null values:
Series([], dtype: int64)
```

In [7]:

```
# Replace null values with an empty string
df = data.fillna('')
```

In [8]:

```
# Create a dictionary to map values
mapping = {'spam': 0, 'ham': 1}

# Update 'Category' values using the mapping dictionary
df['Category'] = df['Category'].map(mapping)

# Display the updated DataFrame
df
```

Out[8]:

	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 × 2 columns

In [9]:

```
import plotly.express as px

# Assuming 'data' is your DataFrame containing the 'Category' column
fig = px.histogram(data, x='Category')

# Display the plot
fig.show()
```

In [10]:

```
# Function to clean the text
def clean_text(text):
    # Remove special characters and numbers
    text = re.sub('[^a-zA-Z]', ' ', text)

    # Convert text to lowercase
    text = text.lower()

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [token for token in tokens if token not in stop_words]

    # Join the tokens back into a single string
    cleaned_text = ' '.join(filtered_tokens)

    return cleaned_text
```

In [11]:

```
# Apply the cleaning function to the "Message" column
df['Message'] = df['Message'].apply(clean_text)
```

In [12]:

```
df['Message']
```

Out[12]:

```
0       go jurong point crazy available bugis n great ...
1                                   ok lar joking wif u oni
2       free entry wkly comp win fa cup final tkts st ...
3                     u dun say early hor u c already say
4                  nah think goes usf lives around though
                              ...
5567    nd time tried contact u u pound prize claim ea...
5568                              b going esplanade fr home
5569                                  pity mood suggestions
5570    guy bitching acted like interested buying some...
5571                                         rofl true name
Name: Message, Length: 5572, dtype: object
```

In [13]:

```
import vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
ana = SentimentIntensityAnalyzer()
```

In [14]:

```
def sa_score(sentence):
    marks = ana.polarity_scores(sentence)
    print(f"{sentence:-<40} {str(marks)}")
```

In [15]:

```
#testing the function
m1  = "b going esplanade fr home"
m2 = "pity mood suggestions"
m3 = "I am not able to classify the emails in respective folders"
print (sa_score(m1))
print (sa_score(m2))
print (sa_score(m3))
```

```
b going esplanade fr home--------------- {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compoun
d': 0.0}
None
pity mood suggestions------------------- {'neg': 0.524, 'neu': 0.476, 'pos': 0.0, 'comp
ound': -0.296}
None
I am not able to classify the emails in respective folders {'neg': 0.0, 'neu': 0.781,
'pos': 0.219, 'compound': 0.4215}
None
```

In [16]:

```
tokenizer = RegexpTokenizer(r'\w+')
```

In [17]:

```
wd = df['Message'].apply(tokenizer.tokenize)
wd.head()
```

Out[17]:

```
0    [go, jurong, point, crazy, available, bugis, n...
1                        [ok, lar, joking, wif, u, oni]
2    [free, entry, wkly, comp, win, fa, cup, final,...
3        [u, dun, say, early, hor, u, c, already, say]
4       [nah, think, goes, usf, lives, around, though]
Name: Message, dtype: object
```

In [18]:

```
# Flatten the list of word tokens
complete = [word for tokens in wd for word in tokens]

# Calculate the lengths of each word token list
dl = [len(tokens) for tokens in wd]

# Create a vocabulary set from all the word tokens
VOCAB = sorted(list(set(word for tokens in wd for word in tokens)))

# Print the results
print(f"{len(complete)} words total, with a vocabulary size of {len(VOCAB)}")
```

```
50385 words total, with a vocabulary size of 7637
```

In [19]:

```
# Checking most common words
count_complete = Counter(complete)
count_complete.most_common(100)
```

Out[19]:

```
[('u', 1224),
 ('call', 605),
 ('get', 397),
 ('ur', 391),
 ('gt', 318),
 ('lt', 316),
 ('ok', 293),
 ('go', 289),
 ('free', 288),
 ('know', 262),
 ('got', 253),
 ('like', 247),
 ('good', 247),
 ('day', 244),
 ('come', 233),
 ('time', 221),
 ('love', 215),
 ('send', 200),
 ('want', 196),
 ('text', 195),
 ('p', 188),
 ('txt', 184),
 ('n', 176),
 ('one', 176),
 ('going', 173),
 ('r', 171),
 ('need', 169),
 ('home', 167),
 ('stop', 163),
 ('lor', 162),
 ('k', 160),
 ('today', 160),
 ('sorry', 160),
 ('see', 159),
 ('still', 158),
 ('back', 153),
 ('da', 151),
 ('dont', 149),
 ('reply', 148),
 ('mobile', 144),
 ('take', 140),
 ('hi', 140),
 ('tell', 139),
 ('new', 136),
 ('please', 135),
 ('later', 135),
 ('pls', 134),
 ('think', 132),
 ('c', 127),
 ('phone', 127),
 ('dear', 125),
 ('week', 124),
 ('well', 120),
 ('night', 118),
 ('much', 116),
 ('great', 115),
 ('oh', 115),
 ('hope', 114),
 ('msg', 113),
 ('hey', 112),
 ('claim', 111),
 ('na', 109),
 ('happy', 108),
 ('wat', 107),
 ('b', 107),
 ('give', 104),
 ('yes', 103),
```

```
 ('way', 103),
 ('make', 102),
 ('work', 101),
 ('www', 100),
 ('e', 98),
 ('number', 97),
 ('message', 96),
 ('wan', 95),
 ('say', 93),
 ('tomorrow', 93),
 ('right', 92),
 ('already', 91),
 ('prize', 91),
 ('ask', 89),
 ('said', 89),
 ('cash', 88),
 ('amp', 88),
 ('yeah', 87),
 ('im', 87),
 ('really', 86),
 ('win', 84),
 ('life', 84),
 ('meet', 83),
 ('find', 83),
 ('miss', 80),
 ('let', 79),
 ('morning', 79),
 ('babe', 79),
 ('thanks', 78),
 ('last', 78),
 ('com', 76),
```

In [20]:

```
# Assuming you have a DataFrame called 'gf' with a column named 'body_new'
df = pd.DataFrame({'Message': ['email body text 1', 'email body text 2', 'email body text 3']})

# Define additional stopwords to add
to_add = ['FW', 'ga', 'httpitcappscorpenroncomsrrsauthemaillinkaspidpage', 'cc', 'aa', 'aaa', 'aaaa',
          'hou', 'cc', 'etc', 'subject', 'pm']

# Create a set of stopwords
stopwords = set(STOPWORDS)

# Add additional stopwords to the set
stopwords.update(to_add)

# Generate the word cloud
wordcloud = WordCloud(
    collocations=False,
    width=1600, height=800,
    background_color='white',
    stopwords=stopwords,
    max_words=150,
    random_state=42
).generate(' '.join(df['Message']))

# Display the word cloud
plt.figure(figsize=(9, 8))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

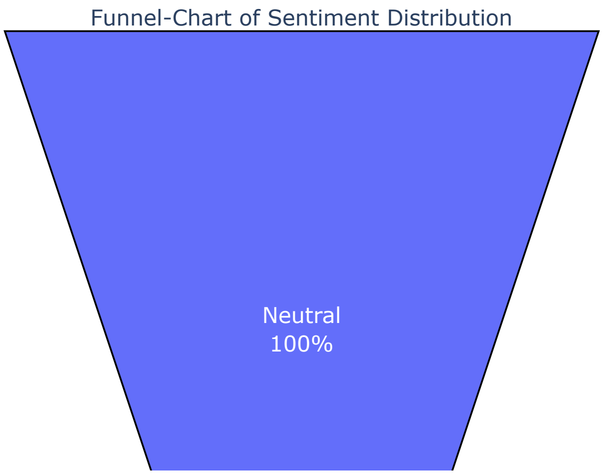
In [21]:

```
# Apply sentiment analysis to each review in 'body_new' column
df['scores'] = df['Message'].map(lambda review: ana.polarity_scores(review))

# Extract the compound score from the 'scores' dictionary
df['compound'] = df['scores'].map(lambda score_dict: score_dict['compound'])

# Assign sentiment labels based on the compound score
df['Sentiment'] = df['compound'].map(lambda x: 'Positive' if x >= 0.05 else 'Negative' if x <= -0.05
```

The error is solved using below code.

In [22]:

```
# Grouping and aggregating the data
var1 = df['Message'].groupby(df['Sentiment']).count().reset_index().sort_values(by='Message', ascendi

# Creating the figure object
fig = go.Figure()

# Adding the Funnel area trace
fig.add_trace(
    go.Funnelarea(
        text=var1['Sentiment'],
        values=var1['Message'],
        title="Funnel-Chart of Sentiment Distribution",
        textposition="inside",
        marker=dict(colors=["#636EFA", "#EF553B", "#00CC96"], line=dict(color='#000000', width=1))
    )
)

# Displaying the figure
fig.show()
```



In [23]:

```
stopwords = set(STOPWORDS)

df_positive = df[df["Sentiment"] == "Positive"]
comment_words = ' '.join(df_positive.Message.astype(str).str.lower())

# Concatenate all positive messages into a single string
comment_words = ' '.join(df_positive.Message.astype(str).str.lower())

# Check if there are words to generate the word cloud
if len(comment_words) > 0:
    # Create the WordCloud object
    wordcloud = WordCloud(width=800, height=800, background_color='white', stopwords=stopwords)

    # Generate the word cloud
    wordcloud.generate(comment_words)

    # Plot the word cloud
    plt.figure(figsize=(8, 8))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()
else:
    print("No words found to generate the word cloud.")
```

```
No words found to generate the word cloud.
```

In [24]:

```
# Create a dictionary to map values
mapping = {'spam': 0, 'ham': 1}

# Update 'Category' values using the mapping dictionary
data['Category'] = data['Category'].map(mapping)
```

In [25]:

```
Texts = data['Message']
Label = data['Category']
```

In [26]:

```
# Split the data into a training set (70%) and a test set (30%)
X_train, X_test, y_train, y_test = train_test_split(Texts, Label, test_size=0.3, random_state=42)

# Print the shape of the training and test sets
print("Train Data:", X_train.shape, y_train.shape)
print("Test Data:", X_test.shape, y_test.shape)
```

```
Train Data: (3900,) (3900,)
Test Data: (1672,) (1672,)
```

In [27]:

```
# Feature Extraction
# Transforming Text Data to feature vector that is used as input to Machine Learning Models.
# Create an instance of TfidfVectorizer
feature_extraction = TfidfVectorizer(min_df=1, stop_words='english', lowercase=True)

# Fit and transform the training set
X_train_features = feature_extraction.fit_transform(X_train)

# Transform the test set
X_test_features = feature_extraction.transform(X_test)
```

In [28]:

```
# Convert Y_train and Y_test as integer
y_train = y_train.astype('int')
y_test = y_test.astype('int')
```

Logistic Regression

In [29]:

```
model = LogisticRegression()
```

In [30]:

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

Out[30]:

```
LogisticRegression()
```

In [31]:

```
pred_LR_train = model.predict(X_train_features)
accuracy_LR_train = accuracy_score(y_train,pred_LR_train)
accuracy_LR_train
```

Out[31]:

```
0.963076923076923
```

In [32]:

```
pred_LR_test = model.predict(X_test_features)
accuracy_LR_test = accuracy_score(y_test,pred_LR_test)
accuracy_LR_test
```

Out[32]:

```
0.9659090909090909
```

In [33]:

```
# Generate the classification report
classification_report_LR = classification_report(y_test, pred_LR_test)

# Print the classification report
print(classification_report_LR)
```

```
              precision    recall  f1-score   support

           0       0.99      0.75      0.86       224
           1       0.96      1.00      0.98      1448

    accuracy                           0.97      1672
   macro avg       0.98      0.88      0.92      1672
weighted avg       0.97      0.97      0.96      1672
```

Hyper parameter Tunning

In [34]:

```
from sklearn.preprocessing import MaxAbsScaler

# Scale the features using MaxAbsScaler
scaler = MaxAbsScaler()
X_train_features_scaled = scaler.fit_transform(X_train_features)
X_test_features_scaled = scaler.transform(X_test_features)

# Create the Logistic Regression model and fit with scaled features
model = LogisticRegression(max_iter=1000)
model.fit(X_train_features_scaled, y_train)
```

Out[34]:

```
LogisticRegression(max_iter=1000)
```

In [35]:

```
# Create the Logistic Regression model with the 'saga' solver
model = LogisticRegression(max_iter=1000, solver='saga')
model.fit(X_train_features, y_train)
```

Out[35]:

```
LogisticRegression(max_iter=1000, solver='saga')
```

In [36]:

```
# Create the Logistic Regression model with L2 regularization (Ridge)
model = LogisticRegression(max_iter=1000, penalty='l2')
model.fit(X_train_features, y_train)
```

Out[36]:

```
LogisticRegression(max_iter=1000)
```

In [37]:

```
from sklearn.metrics import accuracy_score

# Train accuracy for the model with max_iter=1000
pred_train = model.predict(X_train_features_scaled)
accuracy_train = accuracy_score(y_train, pred_train)
print("Train Accuracy (max_iter=1000):", accuracy_train)

# Test accuracy for the model with max_iter=1000
pred_test = model.predict(X_test_features_scaled)
accuracy_test = accuracy_score(y_test, pred_test)
print("Test Accuracy (max_iter=1000):", accuracy_test)

# Train accuracy for the model with 'saga' solver
pred_train_saga = model.predict(X_train_features)
accuracy_train_saga = accuracy_score(y_train, pred_train_saga)
print("Train Accuracy (solver='saga'):", accuracy_train_saga)

# Test accuracy for the model with 'saga' solver
pred_test_saga = model.predict(X_test_features)
accuracy_test_saga = accuracy_score(y_test, pred_test_saga)
print("Test Accuracy (solver='saga'):", accuracy_test_saga)

# Train accuracy for the model with L2 regularization (Ridge)
pred_train_ridge = model.predict(X_train_features)
accuracy_train_ridge = accuracy_score(y_train, pred_train_ridge)
print("Train Accuracy (L2 regularization - Ridge):", accuracy_train_ridge)

# Test accuracy for the model with L2 regularization (Ridge)
pred_test_ridge = model.predict(X_test_features)
accuracy_test_ridge = accuracy_score(y_test, pred_test_ridge)
print("Test Accuracy (L2 regularization - Ridge):", accuracy_test_ridge)
```

```
Train Accuracy (max_iter=1000): 0.99
Test Accuracy (max_iter=1000): 0.9850478468899522
Train Accuracy (solver='saga'): 0.963076923076923
Test Accuracy (solver='saga'): 0.9659090909090909
Train Accuracy (L2 regularization - Ridge): 0.963076923076923
Test Accuracy (L2 regularization - Ridge): 0.9659090909090909
```

Random Forest

In [38]:

```
# Create a Random Forest classifier object
rf_classifier = RandomForestClassifier()

# Fit the classifier on the training data
rf_classifier.fit(X_train_features, y_train)
```

Out[38]:

```
RandomForestClassifier()
```

In [39]:

```
pred_RF_train =rf_classifier.predict(X_train_features)
accuracy_RF_train = accuracy_score(y_train,pred_RF_train)
accuracy_RF_train
```

Out[39]:

```
1.0
```

In [40]:

```
pred_RF_test = rf_classifier.predict(X_test_features)
accuracy_RF_test = accuracy_score(y_test,pred_RF_test)
accuracy_RF_test
```

Out[40]:

```
0.9796650717703349
```

In [41]:

```
# Generate the classification report
classification_report_RF = classification_report(y_test, pred_RF_test)

# Print the classification report
print(classification_report_RF)
```

```
              precision    recall  f1-score   support

           0       1.00      0.85      0.92       224
           1       0.98      1.00      0.99      1448

    accuracy                           0.98      1672
   macro avg       0.99      0.92      0.95      1672
weighted avg       0.98      0.98      0.98      1672
```

Hyper parameter Tunning Random Forest

In [42]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Create the Random Forest classifier
rf_classifier = RandomForestClassifier()

# Define the hyperparameters to search
param_grid = {
    'n_estimators': [50, 100, 150],   # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4]     # Minimum number of samples required to be at a leaf node
}

# Create the Grid Search object
grid_search = GridSearchCV(rf_classifier, param_grid, cv=5, scoring='accuracy')

# Perform Grid Search with training data
grid_search.fit(X_train_features, y_train)

# Get the best hyperparameters and corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy:", best_accuracy)
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 1
0, 'n_estimators': 100}
Best Accuracy: 0.9753846153846153
```

Decision Tree

In [43]:

```
# Create a Decision Tree classifier object
dt_classifier = DecisionTreeClassifier()

# Fit the classifier on the training data
dt_classifier.fit(X_train_features, y_train)
```

Out[43]:

```
DecisionTreeClassifier()
```

In [44]:

```
pred_DT_train =dt_classifier.predict(X_train_features)
accuracy_DT_train = accuracy_score(y_train,pred_DT_train)
accuracy_DT_train
```

Out[44]:

```
1.0
```

In [45]:

```
pred_DT_test = dt_classifier.predict(X_test_features)
accuracy_DT_test = accuracy_score(y_test,pred_DT_test)
accuracy_DT_test
```

Out[45]:

```
0.9700956937799043
```

In [46]:

```
# Generate the classification report
classification_report_DT = classification_report(y_test, pred_DT_test)

# Print the classification report
print(classification_report_DT)
```

```
              precision    recall  f1-score   support

           0       0.92      0.85      0.88       224
           1       0.98      0.99      0.98      1448

    accuracy                           0.97      1672
   macro avg       0.95      0.92      0.93      1672
weighted avg       0.97      0.97      0.97      1672
```

Hyper parameter Tunning

In [47]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Create the Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Define the hyperparameters to search
param_grid = {
    'criterion': ['gini', 'entropy'],  # Criteria for choosing attributes
    'max_depth': [None, 10, 20, 30],   # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],   # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4]      # Minimum number of samples required to be at a leaf node
}

# Create the Grid Search object
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring='accuracy')

# Perform Grid Search with training data
grid_search.fit(X_train_features, y_train)

# Get the best hyperparameters and corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy:", best_accuracy)
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 5}
Best Accuracy: 0.9592307692307692
```

SVM

In [48]:

```
# Create an SVM classifier object
svm_classifier = SVC()

# Fit the classifier on the training data
svm_classifier.fit(X_train_features, y_train)
```

Out[48]:

```
SVC()
```

In [49]:

```
pred_SVM_train =svm_classifier.predict(X_train_features)
accuracy_SVM_train = accuracy_score(y_train,pred_SVM_train)
accuracy_SVM_train
```

Out[49]:

```
0.9987179487179487
```

In [50]:

```
pred_SVM_test = svm_classifier.predict(X_test_features)
accuracy_SVM_test = accuracy_score(y_test,pred_SVM_test)
accuracy_SVM_test
```

Out[50]:

```
0.9832535885167464
```

In [51]:

```
# Generate the classification report
classification_report_SVM = classification_report(y_test, pred_SVM_test)

# Print the classification report
print(classification_report_SVM)
```

```
              precision    recall  f1-score   support

           0       1.00      0.88      0.93       224
           1       0.98      1.00      0.99      1448

    accuracy                           0.98      1672
   macro avg       0.99      0.94      0.96      1672
weighted avg       0.98      0.98      0.98      1672
```

Comparison of Classification Report

In [52]:

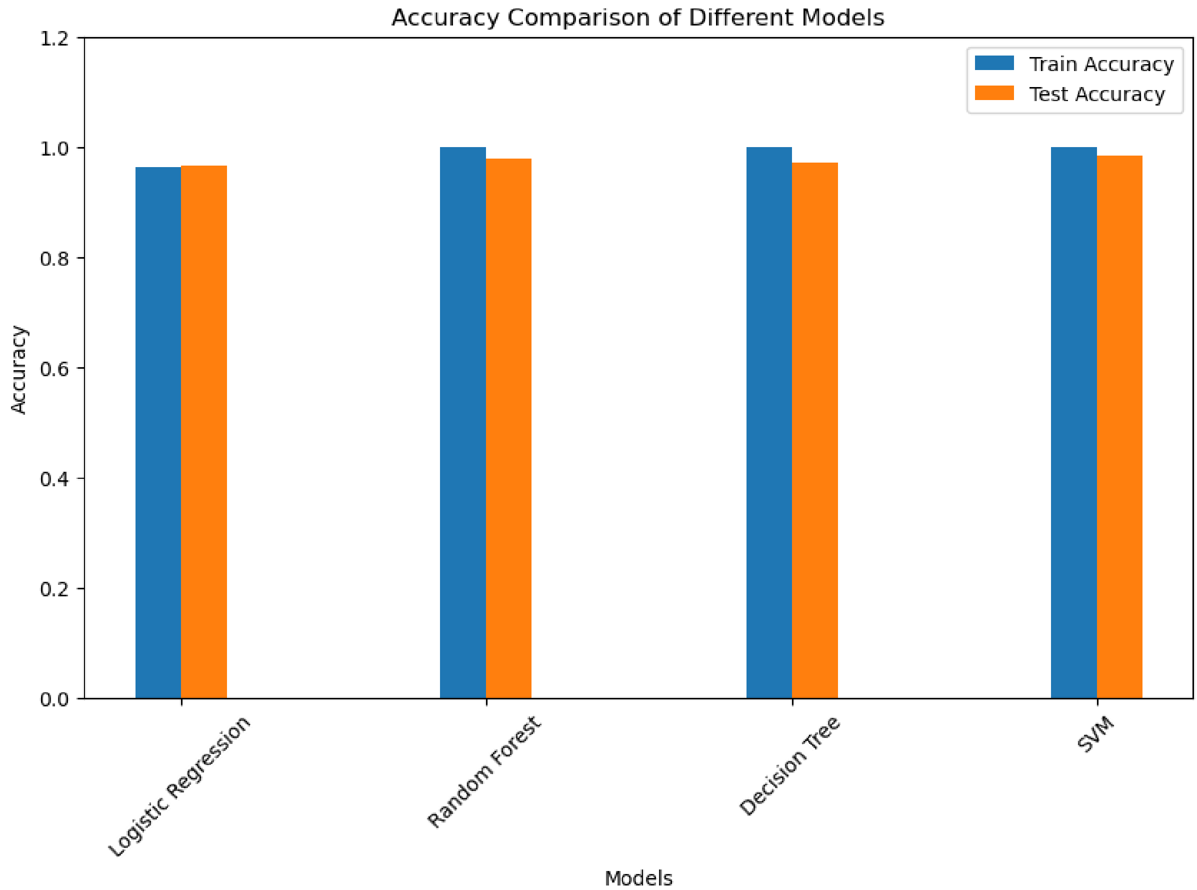
```
# Create a list of model names
models = ['Logistic Regression', 'Random Forest', 'Decision Tree', 'SVM']

# Create a list of predicted labels
preds = [pred_LR_test, pred_RF_test, pred_DT_test, pred_SVM_test]

# Create an empty list to store the classification reports
classification_reports = []

# Generate the classification reports for each model and store them
for model_name, pred in zip(models, preds):
    report = classification_report(y_test, pred)
    classification_reports.append((model_name, report))

# Print the classification reports side by side
for model_name, report in classification_reports:
    print(f"Classification Report for {model_name}:")
    print(report)
    print("\n" + "=" * 80 + "\n")
```

```
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.99      0.75      0.86       224
           1       0.96      1.00      0.98      1448

    accuracy                           0.97      1672
   macro avg       0.98      0.88      0.92      1672
weighted avg       0.97      0.97      0.96      1672


================================================================================

Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       1.00      0.85      0.92       224
           1       0.98      1.00      0.99      1448

    accuracy                           0.98      1672
   macro avg       0.99      0.92      0.95      1672
weighted avg       0.98      0.98      0.98      1672


================================================================================

Classification Report for Decision Tree:
              precision    recall  f1-score   support

           0       0.92      0.85      0.88       224
           1       0.98      0.99      0.98      1448

    accuracy                           0.97      1672
   macro avg       0.95      0.92      0.93      1672
weighted avg       0.97      0.97      0.97      1672


================================================================================

Classification Report for SVM:
              precision    recall  f1-score   support

           0       1.00      0.88      0.93       224
           1       0.98      1.00      0.99      1448

    accuracy                           0.98      1672
   macro avg       0.99      0.94      0.96      1672
weighted avg       0.98      0.98      0.98      1672


================================================================================
```

Comparison of Accuracy

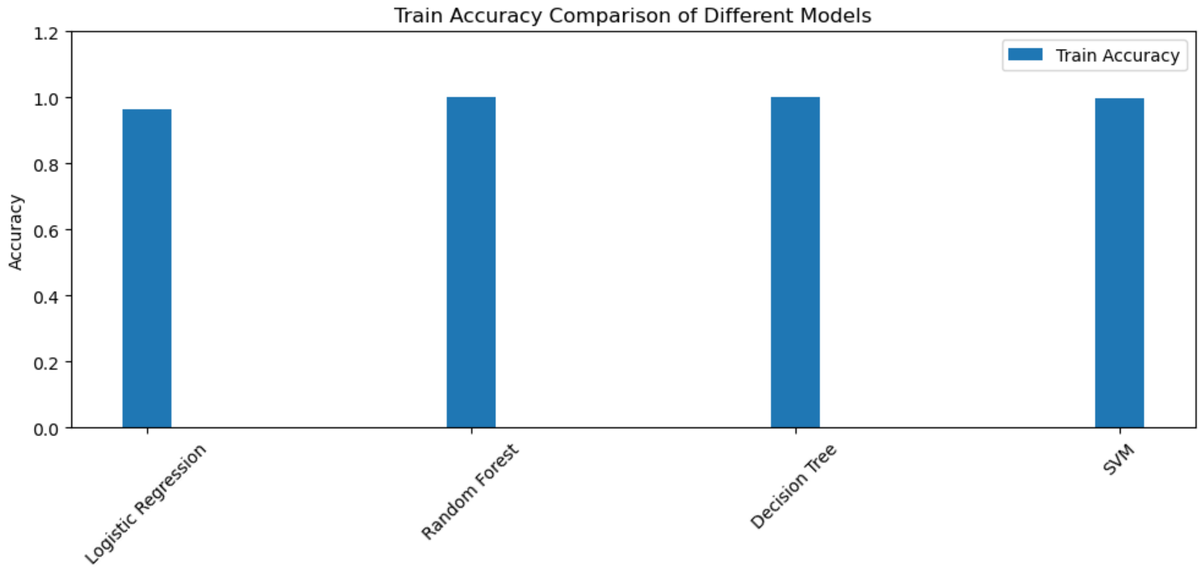
In [53]:

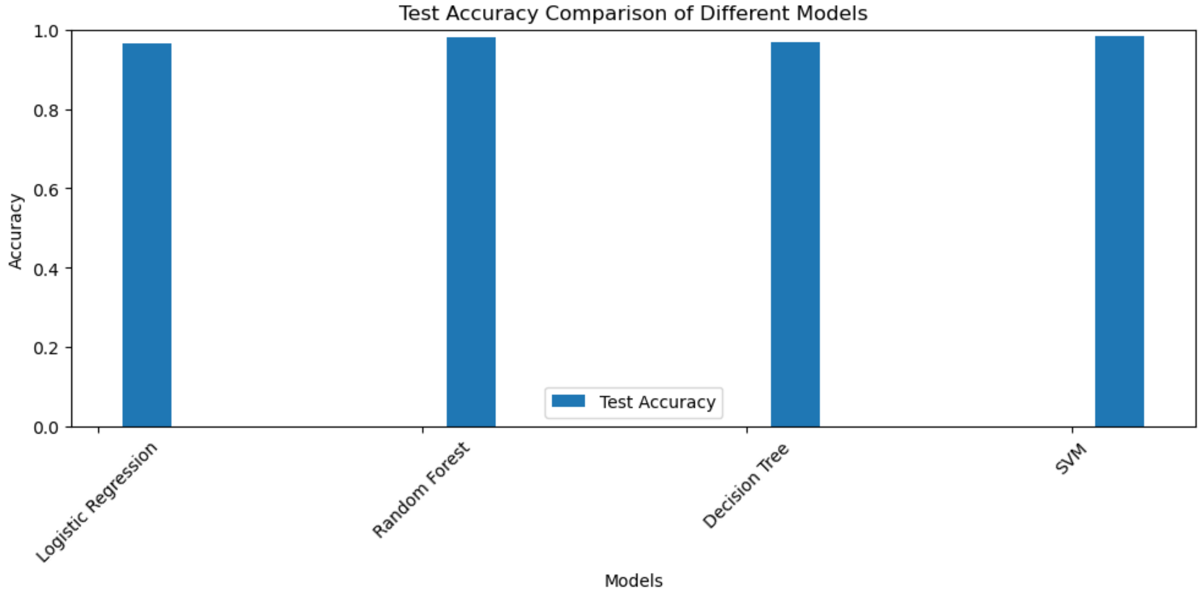
```
# Accuracy values
accuracies_train = [accuracy_LR_train, accuracy_RF_train, accuracy_DT_train, accuracy_SVM_train]
accuracies_test = [accuracy_LR_test, accuracy_RF_test, accuracy_DT_test, accuracy_SVM_test]

# Model names
models = ['Logistic Regression', 'Random Forest', 'Decision Tree', 'SVM']

# Set the width of the bars
bar_width = 0.15

# Define the positions of the bars
train_positions = np.arange(len(models))
test_positions = train_positions + bar_width

# Plotting the accuracy graphs
plt.figure(figsize=(10, 6))
plt.bar(train_positions, accuracies_train, label='Train Accuracy', width=bar_width)
plt.bar(test_positions, accuracies_test, label='Test Accuracy', width=bar_width)
plt.ylim(0, 1.2)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Different Models')
plt.legend()
plt.xticks(train_positions + bar_width / 2, models, rotation=45)
plt.show()
```



In [54]:

```
# Create subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

# Plotting the train accuracy graph
ax1.bar(train_positions, accuracies_train, label='Train Accuracy', width=bar_width)
ax1.set_ylim(0, 1.2)
ax1.set_xlabel('Models')
ax1.set_ylabel('Accuracy')
ax1.set_title('Train Accuracy Comparison of Different Models')
ax1.set_xticks(train_positions)
ax1.set_xticklabels(models, rotation=45)
ax1.legend()

# Plotting the test accuracy graph
ax2.bar(test_positions, accuracies_test, label='Test Accuracy', width=bar_width)
ax2.set_ylim(0, 1.)
ax2.set_xlabel('Models')
ax2.set_ylabel('Accuracy')
ax2.set_title('Test Accuracy Comparison of Different Models')
ax2.set_xticks(train_positions)
ax2.set_xticklabels(models, rotation=45)
ax2.legend()

# Adjust spacing between subplots
plt.tight_layout()

# Show the plots
plt.show()
```





As seen from the above results both Random Forest and SVM has performed well in terms of Accuracy. So we will select the SVM model to classify the emails

Building Prediction Model

In [55]:

```
data_input = ["SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 1

# Convert text to numbers
ndata = feature_extraction.transform(data_input)
```

In [56]:

```
# Making Predictions for the new email 'ndata'
prediction = svm_classifier.predict(ndata)

if prediction[0] == 1:
    print('It is a ham mail')
else:
    print('It is a spam mail')
```

```
It is a spam mail
```