CODE 1

```
# Handle missing values if any
df = df.dropna()

# Assuming 'label' is the target column and 'email' is the text column
X = df['email']  # or other features, depending on the dataset
y = df['label']  # Spam or Not Spam

# Text vectorization (if needed, using CountVectorizer or TfidfVectorizer)
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(X)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

# Initialize the KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can tune the number of neighbors

# Fit the model
knn.fit(X_train, y_train)

# Make predictions
y_pred_knn = knn.predict(X_test)

# Evaluate the model
print("KNN Model Performance:")
print("Accuracy: ", accuracy_score(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))

from sklearn.svm import SVC

# Initialize the SVM model
svm = SVC(kernel='linear')  # You can try other kernels as well

# Fit the model
svm.fit(X_train, y_train)

# Make predictions
```

```
y_pred_svm = svm.predict(X_test)

# Evaluate the model
print("SVM Model Performance:")
print("Accuracy: ", accuracy_score(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion Matrix for KNN
cm_knn = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm_knn, annot=True, fmt="d", cmap="Blues", xticklabels=["Not Spam", "Spam"],
yticklabels=["Not Spam", "Spam"])
plt.title("KNN Confusion Matrix")
plt.show()

# Confusion Matrix for SVM
cm_svm = confusion_matrix(y_test, y_pred_svm)
sns.heatmap(cm_svm, annot=True, fmt="d", cmap="Blues", xticklabels=["Not Spam", "Spam"],
yticklabels=["Not Spam", "Spam"])
plt.title("SVM Confusion Matrix")
plt.show()
```

CODE 2

```
#!/usr/bin/env python
# coding: utf-8
import heapq
# Node structure for Huffman Tree
class HuffmanNode:
def __init__(self, char, freq):
self.char = char
self.freq = freq
self.left = None
self.right = None
# Defining less than operator for priority queue comparison
def __lt__(self, other):
return self.freq < other.freq
# Function to generate Huffman codes
def generate_codes(root, current_code, codes):
if root is None:
return
if root.char is not None:
```

```
codes[root.char] = current_code
generate_codes(root.left, current_code + "0", codes)
generate_codes(root.right, current_code + "1", codes)
# Function to build Huffman Tree
def build_huffman_tree(frequency):
heap = []
# Insert all characters with their frequencies into the heap
for char, freq in frequency.items():
heapq.heappush(heap, HuffmanNode(char, freq))
# Merge nodes until we have one tree
while len(heap) > 1:
node1 = heapq.heappop(heap)
node2 = heapq.heappop(heap)
# Create a new internal node with the combined frequency
merged = HuffmanNode(None, node1.freq + node2.freq)
merged.left = node1
merged.right = node2
heapq.heappush(heap, merged)
# The root of the Huffman Tree
return heapq.heappop(heap)
# Function to calculate frequency of characters
def calculate_frequency(data):
frequency = {}
for char in data:
if char not in frequency:
frequency[char] = 0
frequency[char] += 1
return frequency
# Huffman Encoding process
def huffman_encoding(data):
frequency = calculate_frequency(data)
huffman_tree_root = build_huffman_tree(frequency)
codes = {}
generate_codes(huffman_tree_root, "", codes)
# Encode the input data
encoded_data = "".join([codes[char] for char in data])
return encoded_data, huffman_tree_root
# Huffman Decoding process
def huffman_decoding(encoded_data, huffman_tree_root):
decoded_data = ""
current_node = huffman_tree_root
for bit in encoded_data:
if bit == '0':
current_node = current_node.left
```

```
else:
current_node = current_node.right
if current_node.left is None and current_node.right is None:
decoded_data += current_node.char
current_node = huffman_tree_root
return decoded_data
# Driver code
if __name__ == "__main__":
data = input("Enter the string to be encoded using Huffman Encoding: ")
encoded_data, huffman_tree_root = huffman_encoding(data)
print(f"\nEncoded Data: {encoded_data}")
decoded_data = huffman_decoding(encoded_data, huffman_tree_root)
print(f"Decoded Data: {decoded_data}")
```