Task4 : Spam SMS Detection

Background:

The rise of mobile communication has led to an increase in unwanted messages, commonly known as spam SMS. These messages are not only annoying but can also be malicious, attempting to defraud or deceive recipients. Detecting spam SMS accurately is crucial to ensure a seamless messaging experience for users and to protect them from potential scams.

Problem Statement:

The challenge is to develop a machine learning model capable of identifying spam SMS messages accurately. This problem falls under the category of text classification in natural language processing. Unlike regular SMS, spam messages often contain unusual patterns, keywords, or phrases that differentiate them from legitimate messages. The goal is to leverage these patterns and train a model that can effectively distinguish between spam and non-spam (ham) SMS.

Approach:

To address this problem, we will employ various text processing techniques to convert the textual data into numerical features suitable for machine learning models. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) and word embeddings like Word2Vec or GloVe. These methods allow us to represent words in a way that machine learning algorithms can understand.

We will explore different classification algorithms such as Naive Bayes, Logistic Regression, Random Forest, or even more advanced methods like Recurrent Neural Networks (RNNs) or Long Short-Term Memory networks (LSTMs) if dealing with sequential data. Naive Bayes is a probabilistic algorithm, while Logistic Regression and Random Forest are versatile linear and ensemble classifiers. RNNs and LSTMs are deep learning models capable of capturing sequential patterns in text.

The challenge here is to preprocess the text data, extract relevant features, and select the most suitable machine learning algorithm to build a robust and accurate spam SMS detection system.

Approach & Process:

1. Data Gathering

2. Data Cleaning and Preprocessing

3. Data Visualization and Exploration

4. Feature Enhancement and Refinement

5. Model Selection and Fine-Tuning

Import necessary libraries

In [18]:
```
## Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from wordcloud import WordCloud
```

In [19]:
```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[19]: True

In [20]:
```
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")
```

1. Data Gathering:

Our dataset, assembled from Kaggle and various public sources, provides a robust foundation for our Spam SMS Detection system. These messages, categorized as spam or ham, offer diverse examples for training. By processing this dataset, our model learns to identify spam messages, harnessing the unique language nuances found in the data acquired from Kaggle and other reliable sources.

In [21]:
```
data = pd.read_csv("./SPAM SMS DETECTION Dataset/spam.csv" ,encoding = "latin-1")
data.head()
```

Out[21]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

In [22]:
```
## Check the size
data.shape
```

Out[22]:
```
(5572, 5)
```

In [23]:
```
## Basic info on data set
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

The dataset contains 5572 entries with two significant columns: 'v1' indicating spam or ham labels, and 'v2' containing SMS text. The dataset also has three unnamed columns with numerous missing values, potentially irrelevant for analysis. Focus will be on 'v1' and 'v2' for spam SMS detection.

2. Data Cleaning and Preprocessing

In the Data Cleaning and Preprocessing step, the dataset undergoes essential operations:

1. **Removing Duplicates:** Identical entries are eliminated, ensuring each data point is unique.
2. **Removing NaN Rows and Columns:** Rows and columns with missing values are deleted, ensuring data integrity.
3. **Preprocessing the Data:** Textual data is standardized (lowercasing, removing special characters) for consistency, aiding machine learning analysis.

In [24]:
```
## Check for Duplicates and Remove them
data.duplicated().sum() ## Will give us a number of duplicates
```

Out[24]:
```
403
```

In [25]:
```
## Check for nan values

data.isna().sum()  # Will check for any duplicates
```

Out[25]:
```
v1              0
v2              0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
dtype: int64
```

In [26]:
```
## Remove the Null Column
data.drop(["Unnamed: 2","Unnamed: 3","Unnamed: 4"] ,axis = 1 ,inplace = True)
data.head(5)
```

Out[26]:

	v1	v2
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...

In [27]:
```
## Rename the column to appropriate name
data.rename(columns = {"v1":"Target","v2":"SMS"} ,inplace = True)
data.head(10)
```

Out[27]:

	Target	SMS
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...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...

In [28]:
```
## Check for Duplicates and Remove them
data.duplicated().sum() ## Will give us a number of duplicates
```

Out[28]:
```
403
```

In [29]:
```
data.drop_duplicates(inplace = True)  ## Will drops any duplicates
```

In [30]:
```
## Check for Duplicates and Remove them
data.duplicated().sum() ## Will give us a number of duplicates
```

Out[30]:
```
0
```

In [31]:
```
## Check for nan values

data.isna().sum()  # Will check for any duplicates
```

Out[31]:
```
Target    0
SMS       0
dtype: int64
```

In [32]:
```
data.dropna( inplace = True ) ## Will drop any nan containing row if exists
```

In [33]:
```
data.shape
```

Out[33]:
```
(5169, 2)
```

In [35]:

```
## function to preprocess the data
stopword = set(stopwords.words('english'))

def preprocessing(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation using regular expressions
    text = re.sub(r'[^\w\s]', '', text)

    # Remove specific characters #, @, and $
    text = re.sub(r'[#@\$]', '', text)

    # tokenize and convert to list
    tokens = word_tokenize(text)

    ## Lemmatize it
    lemmatizer  = WordNetLemmatizer()

    ## lemmatize each token
    text = [lemmatizer.lemmatize(token) for token in tokens]

    text = [word for word in text if word not in stopword]


    return " ".join(text)
```

The `preprocessing` function first converts the input text to lowercase for consistency. It then removes all punctuation marks and specific characters (#, @, $) using regular expressions, ensuring only words and spaces remain. Next, it tokenizes the text into words and lemmatizes each word, reducing them to their base forms. Finally, it removes common English stopwords (like "and" or "the") to focus on meaningful words. The processed words are joined back into a string, which is returned for further analysis, providing a clean and standardized text ready for natural language processing tasks.

In [36]:

```
## Create list of words in discription column
data["SMS_cleaned"] =  data["SMS"].apply(preprocessing)
```

In [38]:
```
data.head(10)
```

Out[38]:

	Target	SMS	SMS_cleaned
0	ham	Go until jurong point, crazy.. Available only ...	go jurong point crazy available bugis n great ...
1	ham	Ok lar... Joking wif u oni...	ok lar joking wif u oni
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	free entry 2 wkly comp win fa cup final tkts 2...
3	ham	U dun say so early hor... U c already then say...	u dun say early hor u c already say
4	ham	Nah I don't think he goes to usf, he lives aro...	nah dont think go usf life around though
5	spam	FreeMsg Hey there darling it's been 3 week's n...	freemsg hey darling 3 week word back id like f...
6	ham	Even my brother is not like to speak with me. ...	even brother like speak treat like aid patent
7	ham	As per your request 'Melle Melle (Oru Minnamin...	per request melle melle oru minnaminunginte nu...
8	spam	WINNER!! As a valued network customer you have...	winner valued network customer selected receiv...
9	spam	Had your mobile 11 months or more? U R entitle...	mobile 11 month u r entitled update latest col...

3. Data Visualization and Exploration

In this step, we'll analyze the dataset using histograms and bar charts to understand numerical and c
ategorical distributions. Additionally, we'll create word clouds for each genre, visually representin
g the most common words in their descriptions. These visualizations provide both quantitative and qua
litative insights, aiding in a comprehensive exploration of the dataset.

In [40]:
```
## Shows us the label counts
data["Target"].value_counts()
```

Out[40]:
```
Target
ham     4516
spam     653
Name: count, dtype: int64
```

In [42]:

```
import plotly.express as px
import plotly.io as pio
from IPython.display import Image

# Count the frequency of each genre
genre_counts = data["Target"].value_counts()

# Create a histogram using Plotly Express
fig = px.bar(genre_counts, x=genre_counts.index, y=genre_counts.values,
             labels={'x': 'Target', 'y': 'Frequency'},
             title='Ham or spam Distribution',
             color=genre_counts.index,
             color_discrete_sequence=px.colors.qualitative.Set1)

fig.update_layout(xaxis_tickangle=-45)

# Save the plot as a static image (PNG format)
pio.write_image(fig, 'Target_distribution.png', format='png')

# Show the static image in Jupyter Notebook

Image('Target_distribution.png')
```

Out[42]:

In [43]:
```
# pip install -U kaleido
```

In [44]:

```
# Create a violin plot using Plotly Express
fig = px.violin(data, y="Target", box=True, points="all",
                color="Target",
                title='Ham or Spam Distribution - Violin Plot',
                labels={'Genre': 'Genres'})

fig.update_layout(yaxis_title='Target')

# Save the violin plot as a static image (PNG format)
pio.write_image(fig, 'violin_plot.png', format='png')

# Display the static image in Jupyter Notebook
from IPython.display import Image
Image('violin_plot.png')
```

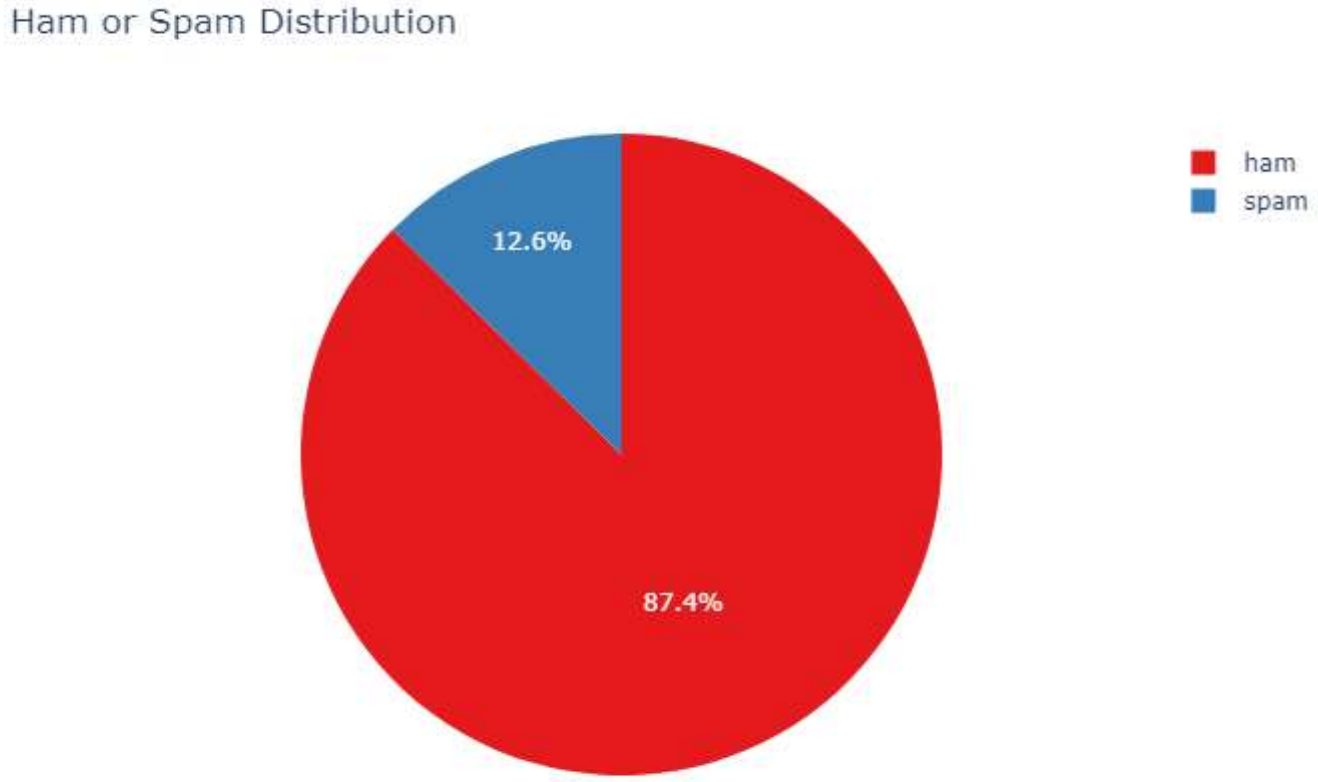
Out[44]:


Ham or Spam Distribution - Violin Plot
ham
spam
Target
Target
ham
spam

In [47]:

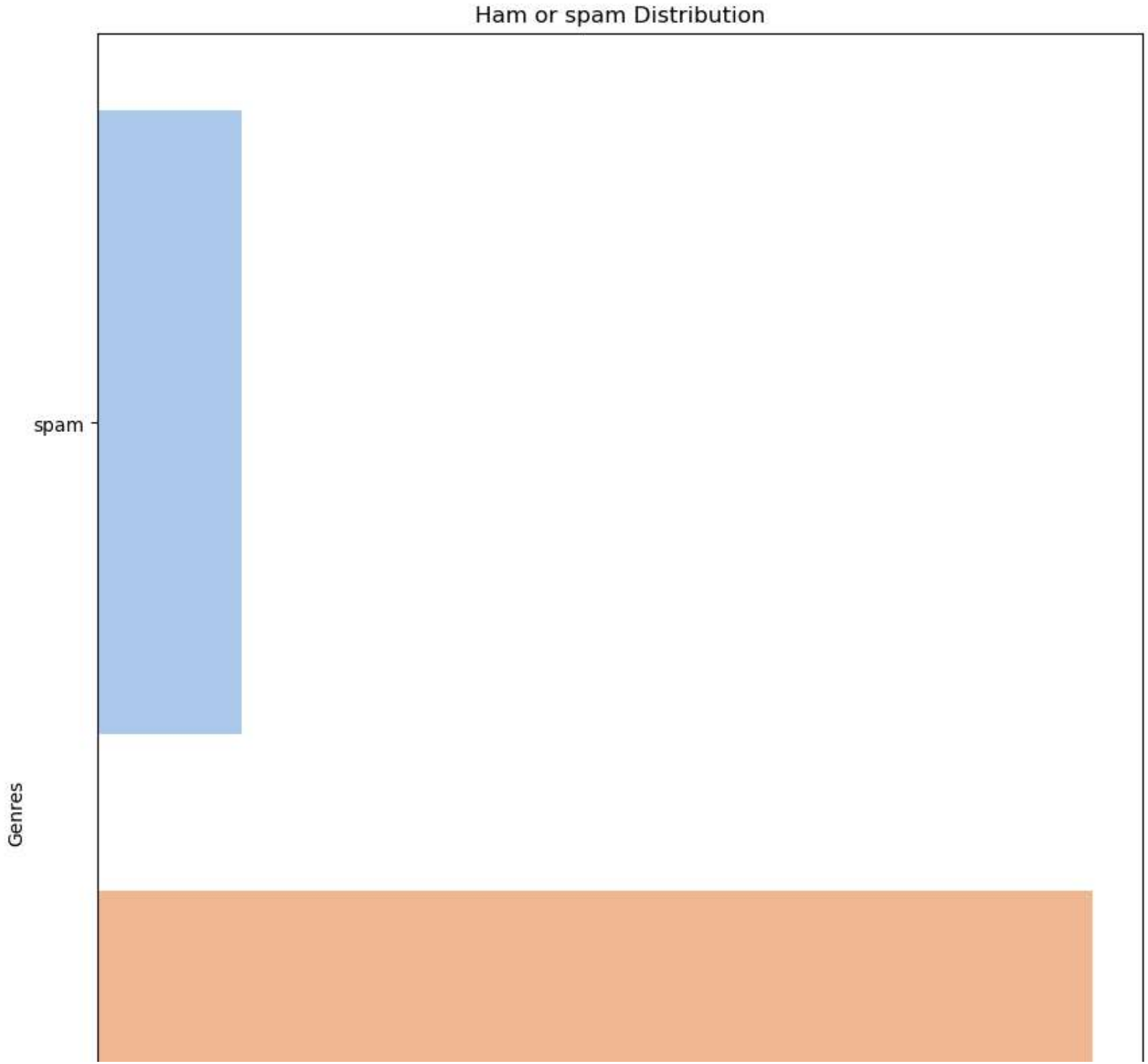
```
# Count the frequency of each genre
Target_counts = data["Target"].value_counts()

# Create an interactive pie chart using Plotly Express
fig = px.pie(Target_counts,
             names=Target_counts.index,
             values=Target_counts.values,
             title='Ham or Spam Distribution',
             color_discrete_sequence=px.colors.qualitative.Set1)

# Show the interactive pie chart
#fig.show()

# Save the pie chart as a static image (PNG format)
pio.write_image(fig, 'Target_distribution_pie_chart.png', format='png')

# Display the static image in Jupyter Notebook
Image('Target_distribution_pie_chart.png')
```

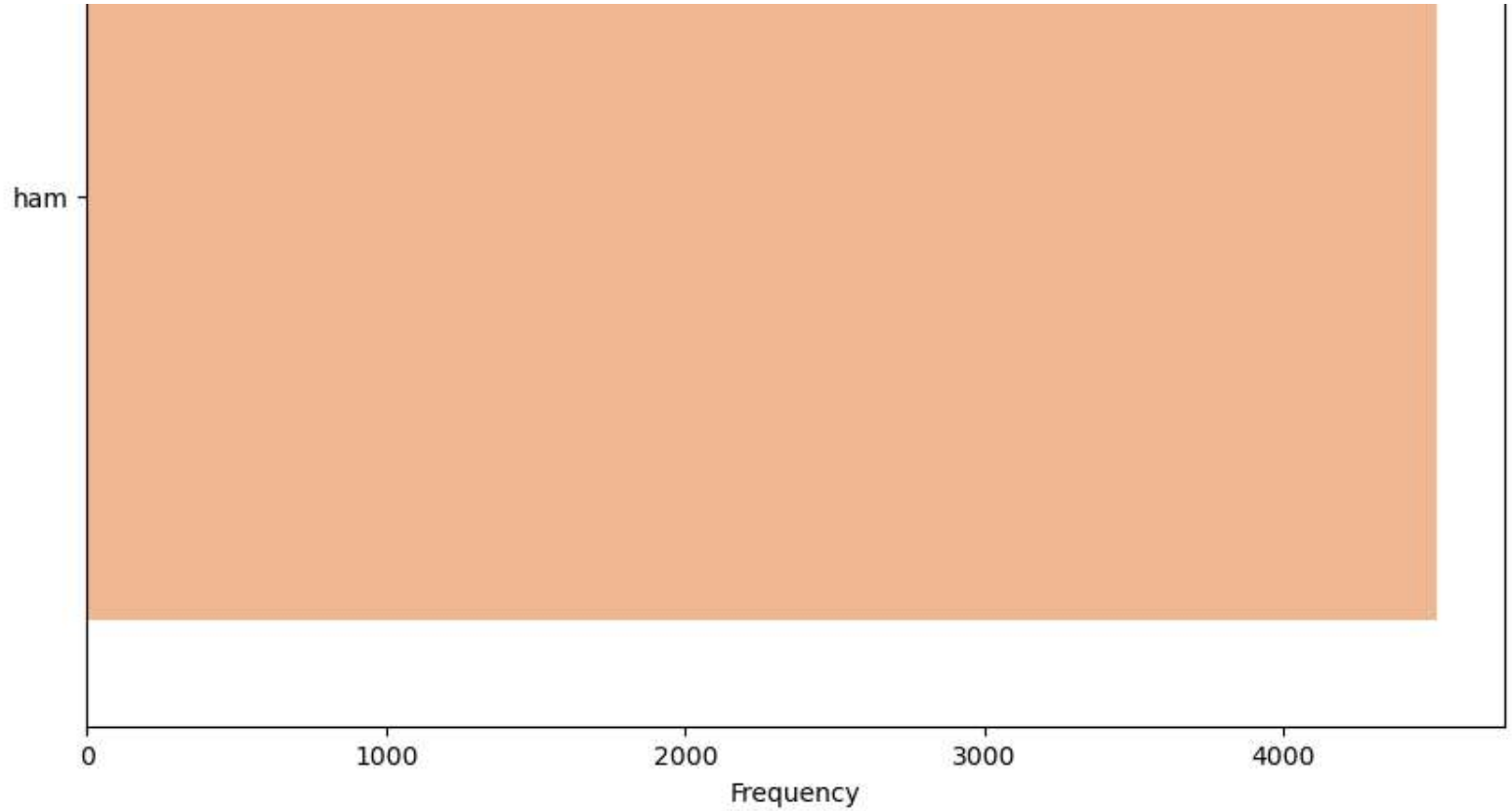
Out[47]:



In [46]:

```
# Count the frequency of each genre
Target_counts = data["Target"].value_counts()

# Sort genres based on frequency
sorted_Target = Target_counts.sort_values(ascending=True)

# Set the color palette
colors = sns.color_palette("pastel")

# Create a horizontal bar chart with Seaborn for a stylish visualization
plt.figure(figsize=(10, 15))
sns.barplot(x=sorted_Target.values, y=sorted_Target.index, palette=colors)

plt.title("Ham or spam Distribution")
plt.xlabel("Frequency")
plt.ylabel("Genres")
plt.show()
```


Ham or spam Distribution
spam
Genres


ham
0
1000
2000
3000
4000
Frequency

In [53]:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Get unique genres from the dataset
unique_Target = data['Target'].unique()

# Set up the plot grid
num_rows = len(unique_Target) // 2 + len(unique_Target) % 2
num_cols = 2
plt.figure(figsize=(15, 5 * num_rows))

# Generate word clouds for each genre
for i, Target in enumerate(unique_Target, 1):
    plt.subplot(num_rows, num_cols, i)
    Target_description = " ".join(list(data[data["Target"] == Target]['SMS']))
    wordcloud = WordCloud(max_words=400, width=900, height=400, background_color='white').generate(Target_
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for {Target} Detection', fontsize=18)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

4. Feature Enhancement and Refinement or Model Selection and Fine-Tuning

- In this section we will try:

- **Model Selection:** Choose appropriate text vectorization techniques like CountVectorizer and TF-IDF.
- **Text Vectorization:** Convert textual data into numerical vectors using CountVectorizer (token counts) and TF-IDF (word importance weights).
- **Fine-Tuning:** Experiment with different parameters and configurations to optimize the vectorization process.
- **Accuracy Evaluation:** Assess the accuracy of models generated with different vectorization techniques to determine the most effective method for the specific dataset.
- **Dimensionality Reduction:** Removing irrelevant columns reduces the dataset's dimensionality, preventing the curse of dimensionality and sparsity issues in high-dimensional spaces.
- **Computational Efficiency:** Smaller datasets with fewer features require less memory and computational resources, speeding up training and prediction processes.
- **Model Interpretability:** Models with fewer features are easier to interpret, making it simpler to understand the impact of essential variables on predictions.
- **Avoiding Overfitting:** Irrelevant features can introduce noise, leading to overfitting. Removing them reduces the risk of the model learning noise patterns.
- **Enhanced Model Performance:** Focusing on relevant features ensures the model is trained on meaningful information, potentially improving predictive accuracy.

In [70]:
```
## import necessary library for
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report,confusion_matrix,ConfusionMatrixDisplay
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

In [54]:
```
#Convert sentiment labels to numerical values for modeling
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['Target_encoded'] = label_encoder.fit_transform(data['Target'])


class_names= list(label_encoder.classes_)
class_names
```

Out[54]:
```
['ham', 'spam']
```

In [55]:
```
data.head(10)
```

Out[55]:

	Target	SMS	SMS_cleaned	Target_encoded
0	ham	Go until jurong point, crazy.. Available only ...	go jurong point crazy available bugis n great ...	0
1	ham	Ok lar... Joking wif u oni...	ok lar joking wif u oni	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	free entry 2 wkly comp win fa cup final tkts 2...	1
3	ham	U dun say so early hor... U c already then say...	u dun say early hor u c already say	0
4	ham	Nah I don't think he goes to usf, he lives aro...	nah dont think go usf life around though	0
5	spam	FreeMsg Hey there darling it's been 3 week's n...	freemsg hey darling 3 week word back id like f...	1
6	ham	Even my brother is not like to speak with me. ...	even brother like speak treat like aid patent	0
7	ham	As per your request 'Melle Melle (Oru Minnamin...	per request melle melle oru minnaminunginte nu...	0
8	spam	WINNER!! As a valued network customer you have...	winner valued network customer selected receiv...	1
9	spam	Had your mobile 11 months or more? U R entitle...	mobile 11 month u r entitled update latest col...	1

In [57]:
```
## Split the data
x = data["SMS_cleaned"]
y = data["Target_encoded"]
```

Training the Model Using TF-IDF Vectorization Technique

In [59]:
```
## Using TfidfVectorizer technique
vectorizer = TfidfVectorizer()
x_trans= vectorizer.fit_transform(x)


x_train ,x_test ,y_train ,y_test = train_test_split(x_trans ,y ,test_size = 0.3 ,random_state = 42)
```

In [60]:

```
print("x train ",x_train.shape )
print("x_test" ,x_test.shape)
print("y_train" ,y_train.shape)
print("y test" ,y_test.shape)
```

```
x train  (3618, 8856)
x_test (1551, 8856)
y_train (3618,)
y test (1551,)
```

The code uses TF-IDF Vectorizer to convert text data into numerical format. After splitting into training (70%) and testing (30%) sets, `x_train` has 3618 samples, each with 8856 features, while `x_test` has 1551 samples. Corresponding labels are stored in `y_train` (3618 entries) and `y_test` (1551 entries). This processed data is suitable for machine learning models.

Multinomial Naive Bayes classifier

In [63]:

```
mnb = MultinomialNB()
mnb.fit(x_train, y_train)

# Evaluate the model
train_accuracy = mnb.score(x_train, y_train)
test_accuracy = mnb.score(x_test, y_test)

# Print model accuracy
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")

# Make predictions and display classification report
y_pred = mnb.predict(x_test)
class_report = classification_report(y_test, y_pred)

print("Classification Report:\n", class_report)
```

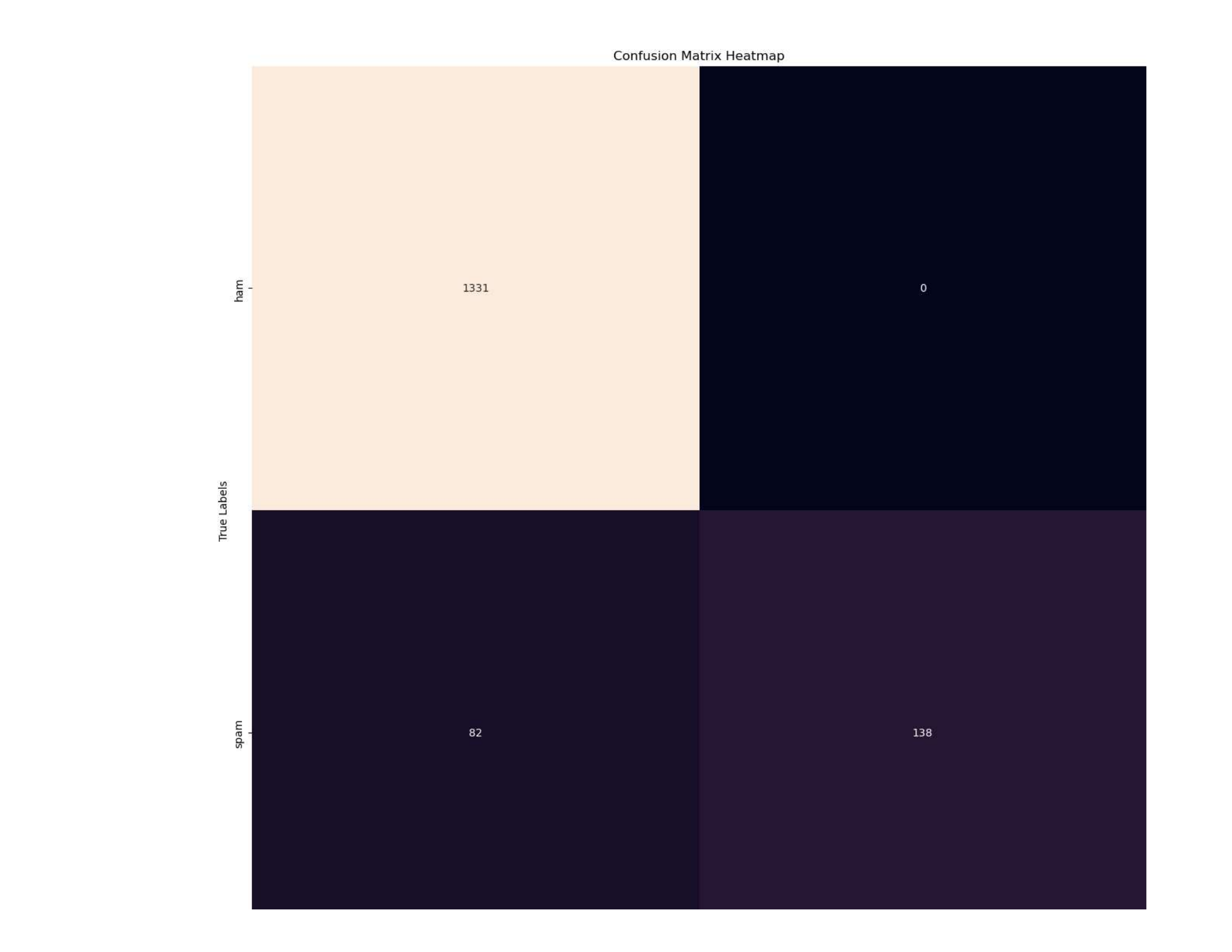
```
Training Accuracy: 0.96
Test Accuracy: 0.95
Classification Report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97      1331
           1       1.00      0.63      0.77       220

    accuracy                           0.95      1551
   macro avg       0.97      0.81      0.87      1551
weighted avg       0.95      0.95      0.94      1551
```

- **Model Type:** Multinomial Naive Bayes Classifier
- **Training Accuracy:** 96%, showcasing the model's accuracy on the training data.
- **Test Accuracy:** 95%, indicating the model's performance on unseen test data.
- **Non-Spam Messages (Class 0):**

    - **Precision:** 94%, signifying 94% of predicted non-spam messages were correct.
    - **Recall:** 100%, meaning the model identified all actual non-spam messages.
    - **F1-Score:** 97%, a balanced metric considering both precision and recall.
- **Spam Messages (Class 1):**
    - **Precision:** 100%, indicating all predicted spam messages were correct.
    - **Recall:** 63%, suggesting the model missed 37% of actual spam messages.
    - **F1-Score:** 77%, reflecting the balance between precision and recall for spam class.
- **Overall:** The model exhibits high accuracy, especially in identifying non-spam messages. However, there's room for improvement in recall for spam messages, indicating potential enhancements in correctly identifying them.

In [64]:

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(conf_matrix, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
ham
spam
1331
0
82
138

LogisticRegression

In [65]:

```
model = LogisticRegression()
model.fit(x_train ,y_train)
print("Model Score on Training data",model.score(x_train ,y_train))
print("Model Score on Testing data",model.score(x_test ,y_test))
y_pred = model.predict(x_test)
print(classification_report(y_pred ,y_test))
```

```
Model Score on Training data 0.9546710889994472
Model Score on Testing data 0.9329464861379755
              precision    recall  f1-score   support

           0       0.99      0.93      0.96      1421
           1       0.56      0.95      0.70       130

    accuracy                           0.93      1551
   macro avg       0.78      0.94      0.83      1551
weighted avg       0.96      0.93      0.94      1551
```

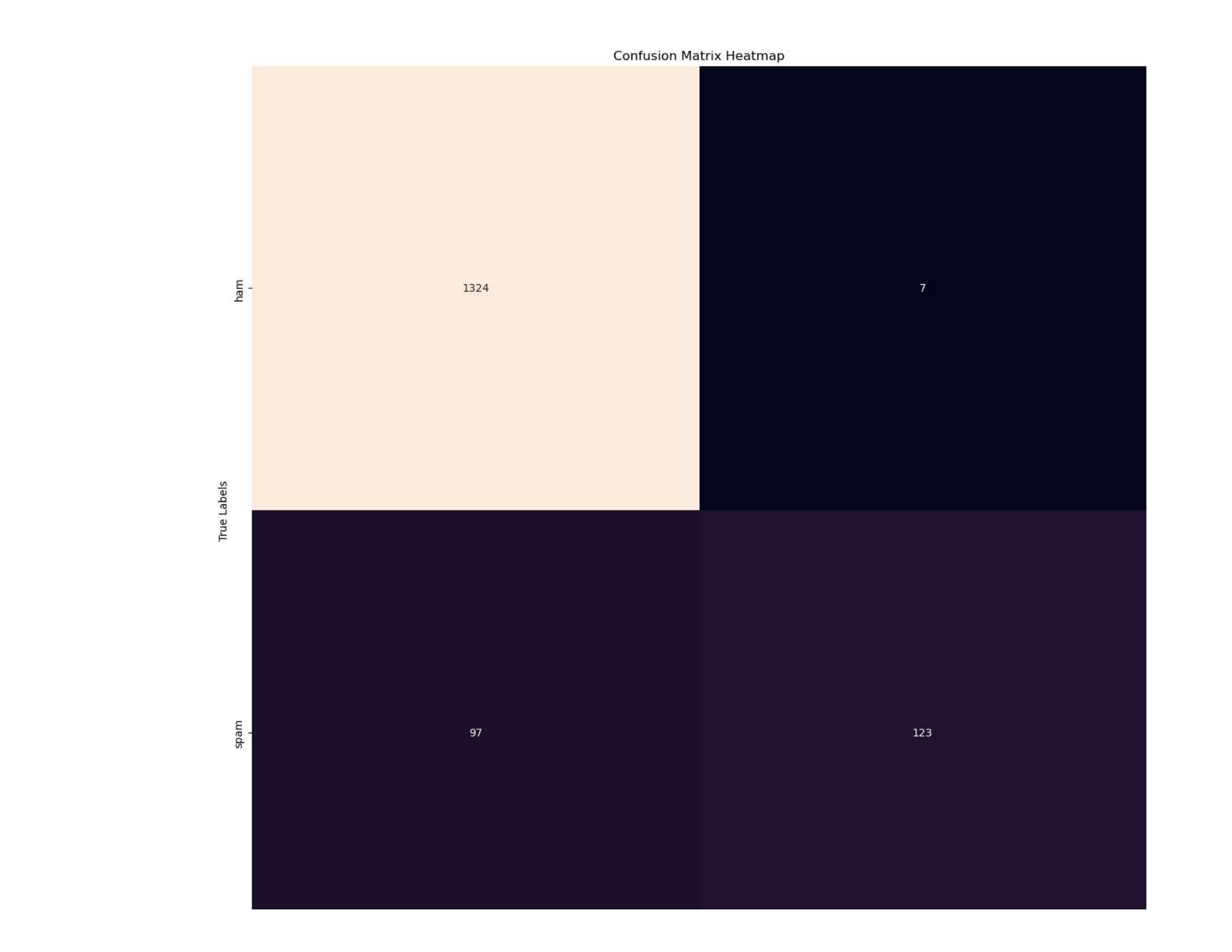
Model Type: Logistic Regression

- **Training Accuracy:** 95.47%, indicating the model's accuracy on the training data.
- **Test Accuracy:** 93.29%, showing the model's performance on unseen test data.
- **Non-Spam Messages (Class 0):**
  - **Precision:** 99%, signifying 99% of predicted non-spam messages were correct.
  - **Recall:** 93%, indicating the model identified 93% of actual non-spam messages.
  - **F1-Score:** 96%, a balanced metric considering both precision and recall.
- **Spam Messages (Class 1):**
  - **Precision:** 56%, indicating 56% of predicted spam messages were correct.

    - **Recall:** 95%, suggesting the model captured 95% of actual spam messages.
    - **F1-Score:** 70%, reflecting the balance between precision and recall for spam class.
- **Overall:** The model demonstrates a strong performance with high accuracy, especially in identifying non-spam messages. However, there's room for improvement in the precision of spam predictions, indicating potential enhancements in correctly identifying them.

In [66]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
ham
spam
1324
7
97
123

Support Vector Machine (SVC)

In [67]:

```
svm = LinearSVC()
svm.fit(x_train ,y_train)
print("Model Score on Training data",svm.score(x_train ,y_train))
print("Model Score on Testing data",svm.score(x_test ,y_test))
y_pred = svm.predict(x_test)
print(classification_report(y_pred ,y_test))
```

```
Model Score on Training data 0.9997236042012161
Model Score on Testing data 0.9742101869761445
              precision    recall  f1-score   support

           0       1.00      0.97      0.99      1363
           1       0.84      0.98      0.90       188

    accuracy                           0.97      1551
   macro avg       0.92      0.98      0.94      1551
weighted avg       0.98      0.97      0.98      1551
```

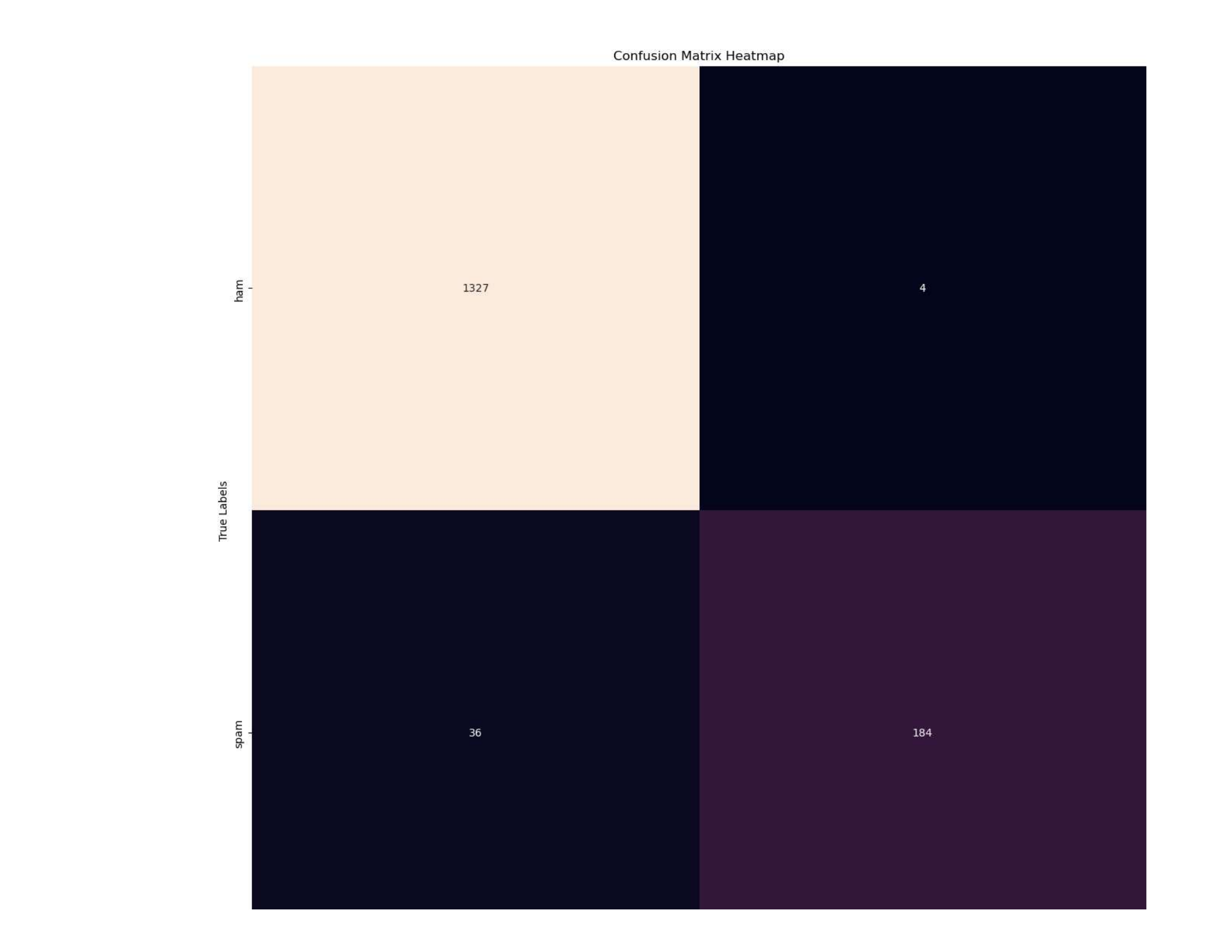
Model Type: Support Vector Machine (SVC)

- **Training Accuracy:** 99.97%, indicating the model's high accuracy on the training data.
- **Test Accuracy:** 97.42%, showcasing the model's strong performance on unseen test data.
- **Non-Spam Messages (Class 0):**
  - **Precision:** 100%, indicating all predicted non-spam messages were correct.
  - **Recall:** 97%, meaning the model captured 97% of actual non-spam messages.
  - **F1-Score:** 99%, a balanced metric considering both precision and recall.

- **Spam Messages (Class 1):**
  - **Precision:** 84%, signifying 84% of predicted spam messages were correct.
  - **Recall:** 98%, suggesting the model identified 98% of actual spam messages.
  - **F1-Score:** 90%, reflecting the balance between precision and recall for spam class.
- **Overall:** The model demonstrates exceptional accuracy and balance between precision and recall for both non-spam and spam classes, making it highly effective in spam SMS detection.

In [68]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
ham
spam
1327
4
36
184

RandomForestClassifier

In [71]:
```
# Create a Random Forest model
random_forest = RandomForestClassifier()

random_forest.fit(x_train, y_train)
print("Random Forest - Train Score:",random_forest.score(x_train, y_train))
print("Random Forest - Test Score:", random_forest.score(x_test, y_test))

y_pred = random_forest.predict(x_test)
print(classification_report(y_pred ,y_test))
```

```
Random Forest - Train Score: 1.0
Random Forest - Test Score: 0.9626047711154094
              precision    recall  f1-score   support

           0       1.00      0.96      0.98      1387
           1       0.74      0.99      0.85       164

    accuracy                           0.96      1551
   macro avg       0.87      0.98      0.91      1551
weighted avg       0.97      0.96      0.96      1551
```

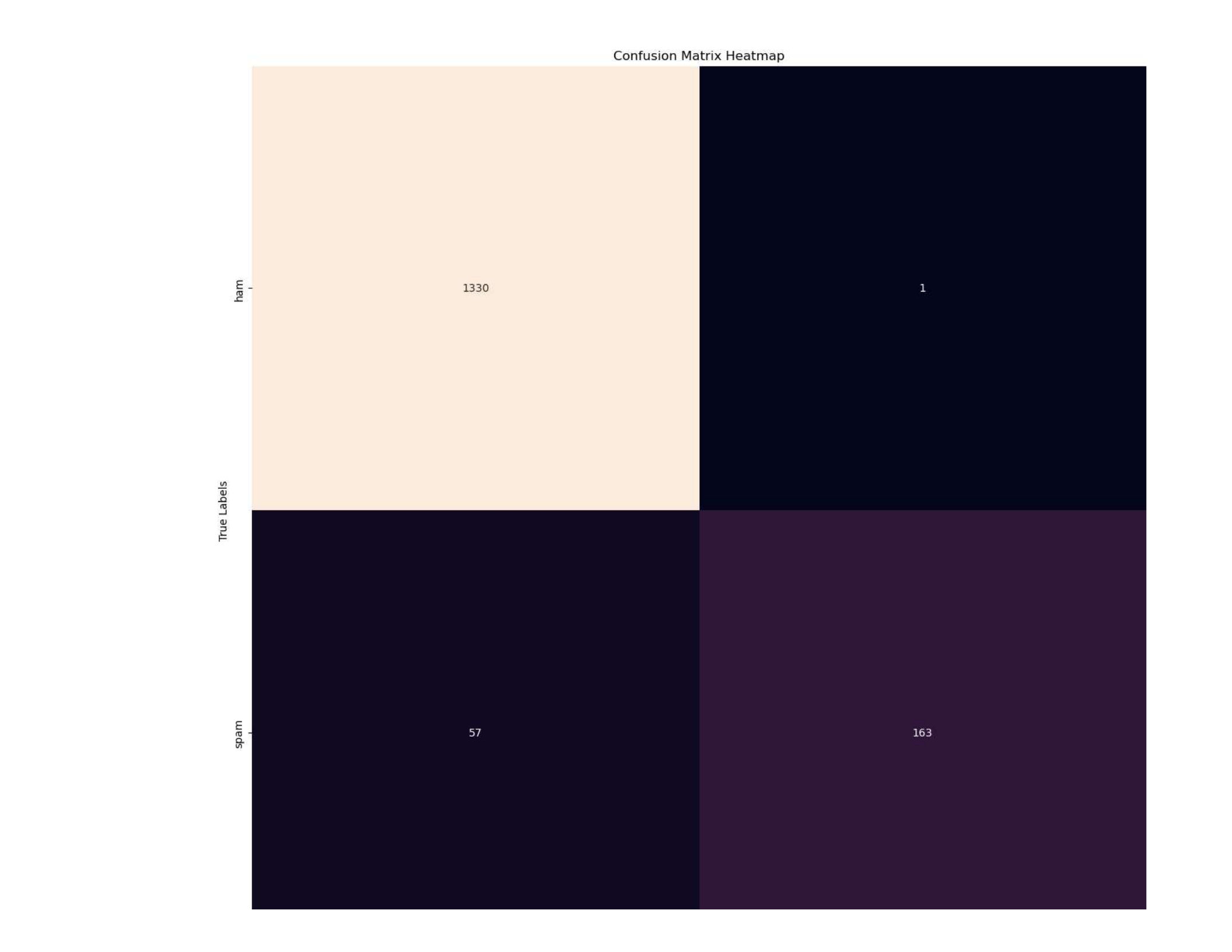
Model Type: Random Forest Classifier

- **Training Accuracy:** 100%, indicating the model's perfect accuracy on the training data.
- **Test Accuracy:** 96.26%, showing the model's strong performance on unseen test data.
- **Non-Spam Messages (Class 0):**
  - **Precision:** 100%, indicating all predicted non-spam messages were correct.
  - **Recall:** 96%, meaning the model captured 96% of actual non-spam messages.
  - **F1-Score:** 98%, a balanced metric considering both precision and recall.

- **Spam Messages (Class 1):**
  - **Precision:** 74%, signifying 74% of predicted spam messages were correct.
  - **Recall:** 99%, suggesting the model identified 99% of actual spam messages.
  - **F1-Score:** 85%, reflecting the balance between precision and recall for spam class.
- **Overall:** The model demonstrates exceptional accuracy and effectiveness in distinguishing both non-spam and spam messages, making it a robust choice for spam SMS detection.

In [72]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
ham
spam
1330
1
57
163

Conclusion:

The Support Vector Machine (SVC) model stands out as the best choice for spam SMS detection. Here's why:

- **SVC Model:**
  - **Training Accuracy:** 99.97%
  - **Test Accuracy:** 97.42%
  - **Precision (Spam Class):** 84%
  - **Recall (Spam Class):** 98%
  - **F1-Score (Spam Class):** 90%

  Advantages:
  - High accuracy in both training and test data.
  - Balanced precision and recall for spam messages, indicating strong performance in identifying both spam and non-spam messages.
  - Overall, the model provides a good balance between accuracy, precision, and recall, making it a reliable choice for practical applications.

While the Random Forest Classifier also performs well, the SVC model achieves a slightly higher recall for spam messages, which is crucial for identifying potential threats effectively. Therefore, the SVC model is recommended as the best choice for spam SMS detection in this context.

Deployment

In [78]:

```
new_mail=[  "Exclusive offer! Get 50% off on our latest collection. Limited time only."]
new_data_features=vectorizer.transform(new_mail)
prediction=svm.predict(new_data_features)
print(prediction)

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

```
[0]
Ham Mail
```

In [79]:

```
new_mails = [
    "Congratulations on winning a cash prize! Claim your reward now.",
    "Exclusive offer! Get 50% off on our latest collection. Limited time only.",
    "Your package is out for delivery. Track your order using the link provided.",
    "Reminder: Your appointment is scheduled for tomorrow at 2 PM.",
    "URGENT: Your account has been compromised. Click here to secure it.",
    "You've been selected for a job interview. Please reply to confirm your availability.",
    "Meeting at 3 PM today. Don't forget to bring the report.",
    "Click this link to receive a special discount on your next purchase.",
    "Your subscription has expired. Renew now to continue enjoying our services.",
    "Hello! How are you doing today? Let's catch up soon."
]

# Predictions for each new mail using the trained models
for i, new_mail in enumerate(new_mails):
    new_data_features = vectorizer.transform([new_mail])
    prediction_mnb = mnb.predict(new_data_features)
    prediction_lr = model.predict(new_data_features)
    prediction_svm = svm.predict(new_data_features)
    prediction_rf = random_forest.predict(new_data_features)

    # Output predictions for each model
    print(f"Mail {i + 1} Predictions:")
    print(f"Naive Bayes Prediction: {'Spam' if prediction_mnb[0] == 1 else 'Ham'}")
    print(f"Logistic Regression Prediction: {'Spam' if prediction_lr[0] == 1 else 'Ham'}")
    print(f"SVM Prediction: {'Spam' if prediction_svm[0] == 1 else 'Ham'}")
    print(f"Random Forest Prediction: {'Spam' if prediction_rf[0] == 1 else 'Ham'}")
    print("-" * 30)
```

```
Mail 1 Predictions:
Naive Bayes Prediction: Spam
Logistic Regression Prediction: Spam
SVM Prediction: Spam
Random Forest Prediction: Spam
------------------------------
Mail 2 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
Mail 3 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Spam
Random Forest Prediction: Ham
------------------------------
Mail 4 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
Mail 5 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
Mail 6 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
Mail 7 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
Mail 8 Predictions:
```

```
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
Mail 9 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
Mail 10 Predictions:
Naive Bayes Prediction: Ham
Logistic Regression Prediction: Ham
SVM Prediction: Ham
Random Forest Prediction: Ham
------------------------------
```

Thank You! 😊🙌

We appreciate your time and attention!

In [ ]: