In [ ]:
```
That's a scam! do not fall for it
Ofcourse this is not email, but just another type of spam

- Email Phishing is the most prominent form of phishing.
- The attacker sends a deceptive email that appears to be from a legitimate sour
- The emails often demand sensitive information, such as login credentials, soci
Some Stats:
```

Aim: The aim of this project is to develop a robust email spam detection system using machine learning techniques. By analyzing the content and characteristics of emails, the system should be able to accurately classify incoming emails as either spam or legitimate (ham).

We will be exploring below models:

1. LogisticRegression
2. Super Vector Machine
3. Random Forest Classifier

The data consist of two columns

1. which is the actual email
2. Label of whether the email is Spam or Ham(not spam)

Dataset-link :- https://github.com/hardik0980/Email-spam-Detection-/blob/main/Spam%20mail.csv

Importing some important Libraries & Load the Data

In [6]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score , classification_report , recall_scor
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder

import warnings
warnings.simplefilter("ignore")
```

In [7]:
```
mail_data = pd.read_csv("mail_data - mail_data.csv")
```

Exploratory Data Analysis & Data Preprocessing!

In [9]:
```
mail_data.head(10)
```

Out[9]:

	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...
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 [11]:
```
mail_data.isna().sum()
```

Out[11]:
```
Category    0
Message     0
dtype: int64
```

In [12]:
```
 # replace the null values with a null string
```

In [13]:
```
new_mail_data = mail_data.where((pd.notnull(mail_data)),"")
```

In [14]:
```
new_mail_data.head()
```

Out[14]:

	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...

In [15]:
```
new_mail_data.describe()
```

Out[15]:

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

In [16]:
```
new_mail_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 [17]:
```
print("number of duplicate values in the Dataset:",new_mail_data.duplicated().su
```

```
number of duplicate values in the Dataset: 415
```

In [18]:
```
# cheching the number of rows and columns in the dataframe
print(" number of rows and columns in Dataset:",new_mail_data.shape)
```

```
number of rows and columns in Dataset: (5572, 2)
```

Spam mail as 1; Ham mail as 0; : Using by LabelEncoder

In [20]:
```
lb = LabelEncoder()
lb.fit(new_mail_data["Category"])
new_mail_data["Category"]=lb.transform(new_mail_data["Category"])
print(new_mail_data.Category)
```

```
0       0
1       0
2       1
3       0
4       0
       ..
5567    1
5568    0
5569    0
5570    0
5571    0
Name: Category, Length: 5572, dtype: int32
```

spam = 1 ham = 0

In [21]:
```
new_mail_data.Category.value_counts()
```

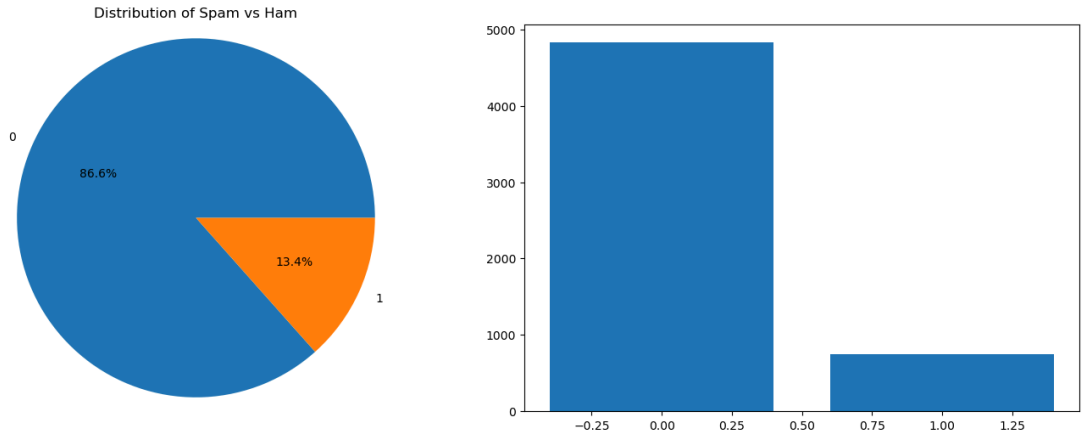
Out[21]:
```
Category
0    4825
1     747
Name: count, dtype: int64
```

In [22]:
```
print("percentage of Ham is :",100*4825/new_mail_data["Category"].count().sum())
print("percentage of Spam is :",100* 747/new_mail_data["Category"].count().sum()
```

```
percentage of Ham is : 86.59368269921033
percentage of Spam is : 13.406317300789663
```

1. The dataset has 4825 emails (86.6%) labled as Ham while 747 (13.4%) labaled as Spam. 2. "ham" is the predominant category. 3. The dataset contains 5,169 unique texts. 4. The most frequent text being "Sorry, I'll call later," occurring 30 times.

In [23]:
```
# Calculate the count of each label

category_counts = new_mail_data.Category.value_counts()

# Plotting the pie chart

plt.figure(figsize=(14,10))

plt.subplot(2,2,1)
plt.pie(category_counts, labels=category_counts.index , autopct="%1.1f%%") #auto
plt.title("Distribution of Spam vs Ham")
plt.axis("equal") ## Equal aspect ratio ensures that pie is drawn as a circle.

plt.subplot(2,2,2)
plt.bar(category_counts.index ,category_counts )
plt.tight_layout()
plt.show()
```



In [24]:
```
x = new_mail_data.Message   # independent

y = new_mail_data.Category  # dependent
```

In [25]:
```
print(x)
```

```
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                              ...
5567    This is the 2nd time we have tried 2 contact u...
5568                 Will ü b going to esplanade fr home?
5569    Pity, * was in mood for that. So...any other s...
5570    The guy did some bitching but I acted like i'd...
5571                           Rofl. Its true to its name
Name: Message, Length: 5572, dtype: object
```

In [26]:
```
print(y)
```

```
0       0
1       0
2       1
3       0
4       0
       ..
5567    1
5568    0
5569    0
5570    0
5571    0
Name: Category, Length: 5572, dtype: int32
```

In [27]:
```
# splitting the data into training data & test data

x_train,x_test,y_train,y_test =train_test_split(x,y,test_size=0.2,random_state=3
```

In [28]:
```
print("\nTotal shape of x is :",x.shape,"\nshape of X_train is :",x_train.shape,
```

```
Total shape of x is : (5572,)
shape of X_train is : (4457,)
shape of x_test is : (1115,)
```

TfidfVectorizer is a feature extraction tool in scikit-learn that converts raw text data into numerical features based on the Term Frequency-Inverse Document Frequency (TF-IDF) scoring. 1. Lower Casing¶ 2. Remove Extra White Spaces 3. Remove HTML Tags 4. Remove URLs 5. Remove Punctuations 6. Remove Special Characters 7. Remove Numeric Values 8. Remove Non-alpha Numeric 9. Handling ChatWords 10. Handling StopWords 11. Handling Emojis 12. Stemming TfidfVectorizer incorporates several of these preprocessing steps automatically, while also noting any steps that might need to be handled separately. The following preprocessing tasks are covered by TfidfVectorizer itself: 1. Lowercasing (lowercase=True). 2. Removing Stop Words (stop_words="english"). 3. Removing Punctuation (automatically handled in tokenization).

feature Extraction

In [30]:
```
# trainsfrom the text data to feature vectors that can be used as input to the L

feature_extraction = TfidfVectorizer(min_df = 1, stop_words="english" , lowercas

x_train_feature =feature_extraction.fit_transform(x_train)
x_test_feature = feature_extraction.transform(x_test)

# convert y_train and y_test values as integers

y_train= y_train.astype("int")
y_test = y_test.astype("int")
```

In [31]:
```
print(x_train_feature)
```

```
(0, 5413)     0.6198254967574347
(0, 4456)     0.4168658090846482
(0, 2224)     0.413103377943378
(0, 3811)     0.34780165336891333
(0, 2329)     0.38783870336935383
(1, 4080)     0.18880584110891163
(1, 3185)     0.29694482957694585
(1, 3325)     0.31610586766078863
(1, 2957)     0.3398297002864083
(1, 2746)     0.3398297002864083
(1, 918)      0.22871581159877646
(1, 1839)     0.2784903590561455
(1, 2758)     0.3226407885943799
(1, 2956)     0.33036995955537024
(1, 1991)     0.33036995955537024
(1, 3046)     0.2503712792613518
(1, 3811)     0.17419952275504033
(2, 407)      0.509272536051008
(2, 3156)     0.4107239318312698
(2, 2404)     0.45287711070606745
(2, 6601)     0.6056811524587518
(3, 2870)     0.5864269879324768
(3, 7414)     0.8100020912469564
(4, 50)       0.23633754072626942
(4, 5497)     0.15743785051118356
:     :
(4454, 4602)  0.2669765732445391
(4454, 3142)  0.32014451677763156
(4455, 2247)  0.37052851863170466
(4455, 2469)  0.35441545511837946
(4455, 5646)  0.33545678464631296
(4455, 6810)  0.29731757715898277
(4455, 6091)  0.23103841516927642
(4455, 7113)  0.30536590342067704
(4455, 3872)  0.3108911491788658
(4455, 4715)  0.30714144758811196
(4455, 6916)  0.19636985317119715
(4455, 3922)  0.31287563163368587
(4455, 4456)  0.24920025316220423
(4456, 141)   0.292943737785358
(4456, 647)   0.30133182431707617
(4456, 6311)  0.30133182431707617
(4456, 5569)  0.4619395404299172
(4456, 6028)  0.21034888000987115
(4456, 7154)  0.24083218452280053
(4456, 7150)  0.3677554681447669
(4456, 6249)  0.17573831794959716
(4456, 6307)  0.2752760476857975
(4456, 334)   0.2220077711654938
(4456, 5778)  0.16243064490100795
(4456, 2870)  0.31523196273113385
```

In [32]:
```
print(x_test_feature)
```

```
(0, 7271)     0.1940327008179069
(0, 6920)     0.20571591693537986
(0, 5373)     0.2365698724638063
(0, 5213)     0.1988547357502182
(0, 4386)     0.18353336340308998
(0, 1549)     0.2646498848307188
(0, 1405)     0.3176863938914351
(0, 1361)     0.25132445289897426
(0, 1082)     0.2451068436245027
(0, 1041)     0.28016206931555726
(0, 405)      0.2381316303003606
(0, 306)      0.23975986557206702
(0, 20)       0.30668032384591537
(0, 14)       0.26797874471323896
(0, 9)        0.2852706805264544
(0, 1)        0.2381316303003606
(1, 7368)     0.29957800964520975
(1, 6732)     0.42473488678029325
(1, 6588)     0.3298937975962767
(1, 6507)     0.26731535902873493
(1, 6214)     0.3621564482127515
(1, 4729)     0.22965776503163893
(1, 4418)     0.3457696891316818
(1, 3491)     0.496093956101028
(2, 7205)     0.22341717215670331
:     :
(1110, 3167)  0.5718357066163949
(1111, 7353)  0.4991205841293424
(1111, 6787)  0.40050175714278885
(1111, 6033)  0.4714849709283488
(1111, 3227)  0.44384935772735523
(1111, 2440)  0.4137350055985486
(1112, 7071)  0.33558524648843113
(1112, 6777)  0.32853717524096393
(1112, 6297)  0.3056896872268727
(1112, 5778)  0.22807428098549426
(1112, 5695)  0.3381604952481646
(1112, 5056)  0.2559183043595413
(1112, 4170)  0.3307835623173863
(1112, 2329)  0.241856898377491
(1112, 1683)  0.4017087436272034
(1112, 1109)  0.35334496762883244
(1113, 4080)  0.3045947361955407
(1113, 4038)  0.37023520529413706
(1113, 3811)  0.28103080586555096
(1113, 3281)  0.33232508601719535
(1113, 3113)  0.33840833425155675
(1113, 2852)  0.5956422931588335
(1113, 2224)  0.3337959267435311
(1114, 4557)  0.5196253874825217
(1114, 4033)  0.8543942045002639
```

In [ ]:

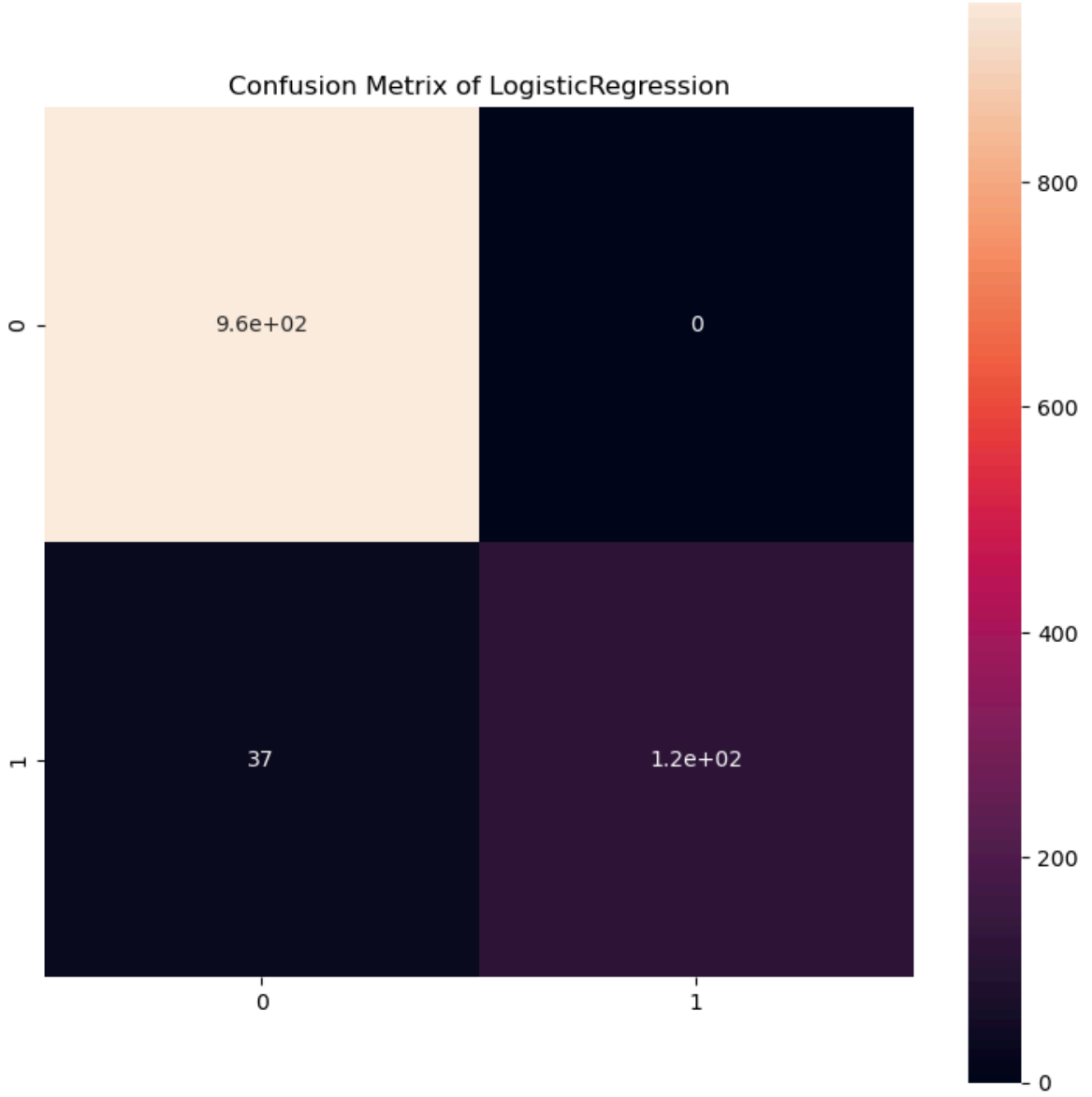
Training and testing model with LogisticRegression

In [34]:
```
Logistic_model = LogisticRegression()
Logistic_model.fit(x_train_feature,y_train)
```

Out[34]:
```
LogisticRegression()
```

In [35]:
```
# prediction on training data

prediction_on_training_data = Logistic_model.predict(x_train_feature)
accuracy_on_training_data = accuracy_score(y_train , prediction_on_training_data
print("Logistic_accuracy_on_training_data is :",accuracy_on_training_data)

# prediction on testing data

prediction_on_testing_data = Logistic_model.predict(x_test_feature)
accuracy_on_testing_data = accuracy_score(y_test , prediction_on_testing_data)
print("Logistic_accuracy_on_testing_data is :",accuracy_on_testing_data)

print("\nclassification_report of LogisticRegression\n",classification_report(y_



print("\n Confusion Metrix of LogisticRegression \n",confusion_matrix(y_test , p
conf_mat = confusion_matrix(y_test , prediction_on_testing_data)
plt.figure(figsize=(9,9))
sns.heatmap(conf_mat , annot=True , square=True)
plt.title("Confusion Metrix of LogisticRegression")
plt.show()
```

```
Logistic_accuracy_on_training_data is : 0.9676912721561588
Logistic_accuracy_on_testing_data is : 0.9668161434977578

classification_report of LogisticRegression
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       960
           1       1.00      0.76      0.86       155

    accuracy                           0.97      1115
   macro avg       0.98      0.88      0.92      1115
weighted avg       0.97      0.97      0.96      1115


 Confusion Metrix of LogisticRegression
 [[960   0]
 [ 37 118]]
```



In [ ]:

Training and testing model with Support Vector Machines

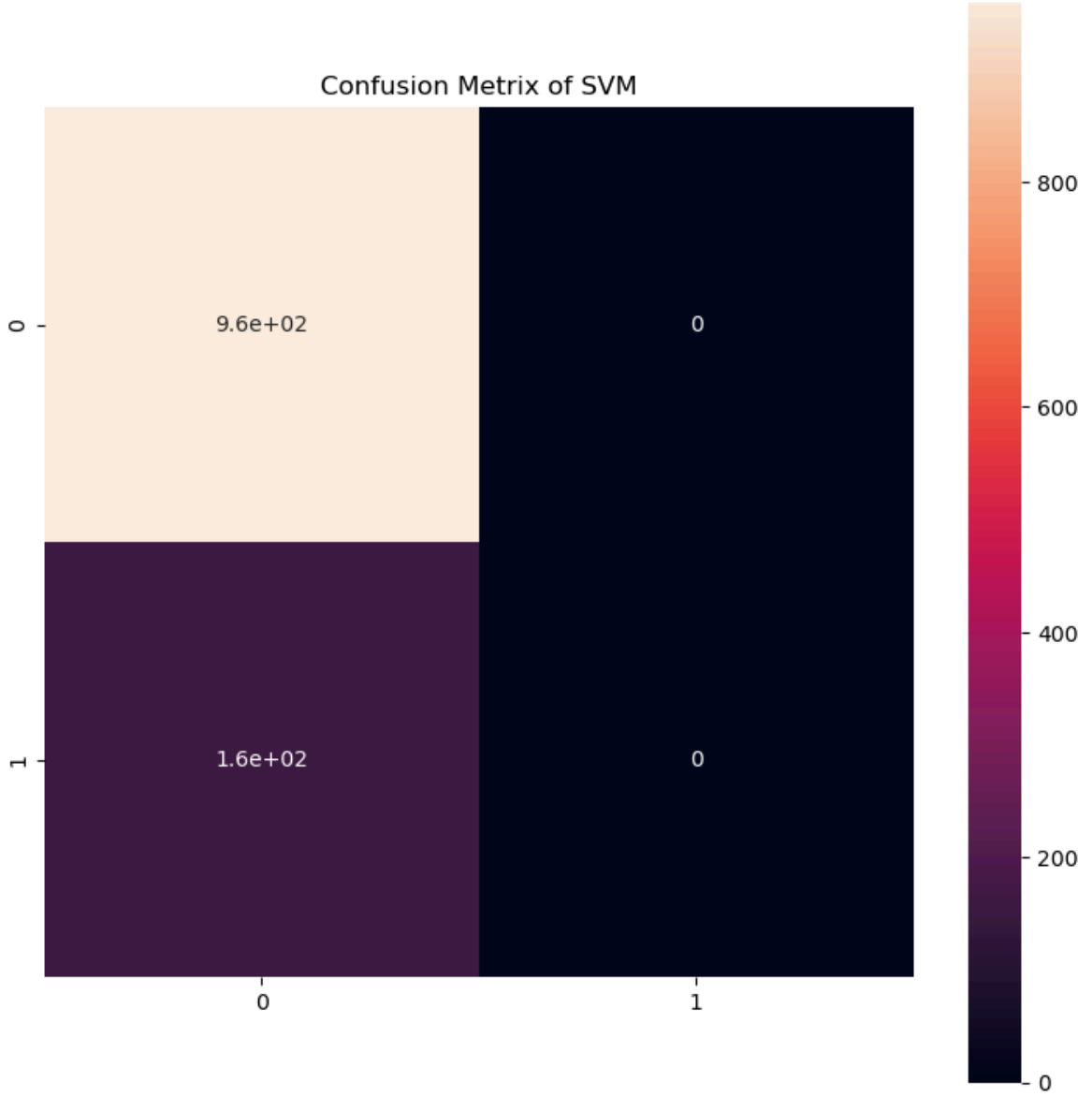
Additional for SVM parameter You May Consider :--- C: Try a range of values, like 0.1, 1, 10, 100, to see what works best for your data. Gamma: Besides 'auto', you can use 'scale', which adjusts gamma as 1 / (\text{n_features} \times \text{X.var()}), or experiment with specific values such as 0.01, 0.1, 1. Kernel: If RBF doesn't yield desired results, try the linear kernel for linear separable data or poly for polynomial boundaries.

In [37]:
```
# Support Vector Machines (SVM) with tuned parameters
from sklearn.svm import SVC

svm_model = SVC(C=10.0, kernel='rbf', gamma='auto')
svm_model.fit(x_train_feature, y_train)
```

Out[37]:
```
SVC
SVC(C=10.0, gamma='auto')
```

In [38]:
```
prediction_on_training_data = svm_model.predict(x_train_feature)
accuracy_on_training_data = accuracy_score(y_train , prediction_on_training_data
print("svm_accuracy_on_training_data is :",accuracy_on_training_data)

prediction_on_testing_data = svm_model.predict(x_test_feature)
accuracy_on_testing_data = accuracy_score(y_test , prediction_on_testing_data)
print("svm_accuracy_on_testing_data is :",accuracy_on_testing_data)

print("\nclassification_report of SVM\n",classification_report(y_test , predicti


print("\n Confusion Metrix of SVM \n",confusion_matrix(y_test , prediction_on_te
conf_mat = confusion_matrix(y_test , prediction_on_testing_data)
plt.figure(figsize=(9,9))
sns.heatmap(conf_mat , annot=True , square=True)
plt.title("Confusion Metrix of SVM")
plt.show()
```

```
svm_accuracy_on_training_data is : 0.8671752299753197
svm_accuracy_on_testing_data is : 0.8609865470852018

classification_report of SVM
              precision    recall  f1-score   support

           0       0.86      1.00      0.93       960
           1       0.00      0.00      0.00       155

    accuracy                           0.86      1115
   macro avg       0.43      0.50      0.46      1115
weighted avg       0.74      0.86      0.80      1115


 Confusion Metrix of SVM
 [[960   0]
 [155   0]]
```



In [ ]:

Training and testing model with Random Forest classifier

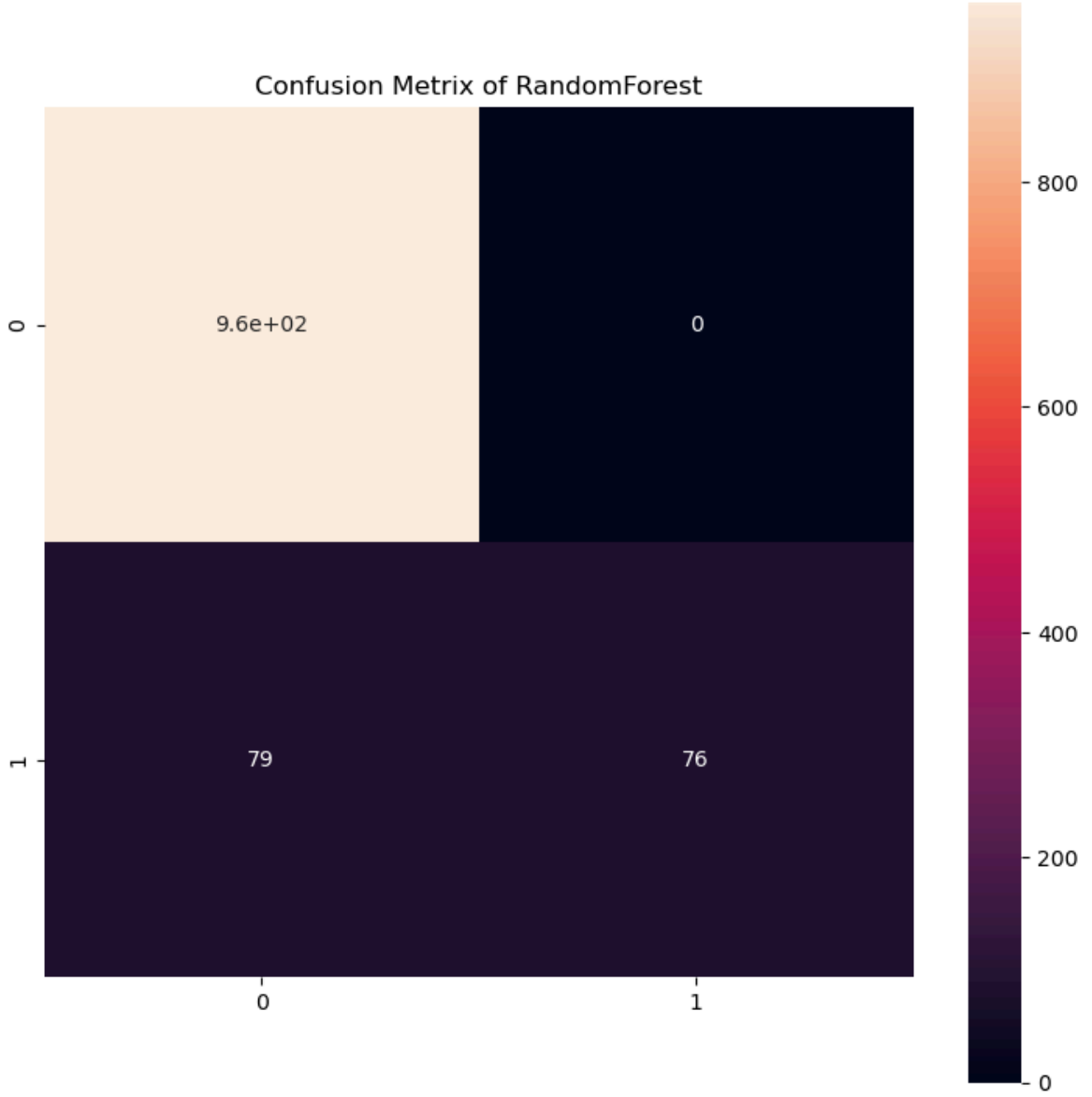
Additional Random Forest Parameters You May Consider max_features: The number of features to consider when looking for the best split. For example, using "sqrt" (square root of total features) or "log2" can make trees more diverse and reduce overfitting. min_samples_leaf: The minimum number of samples required to be at a leaf node. Setting this to a higher value can help generalize the model. bootstrap: If True, each tree is trained on a random subset of the data with replacement. Setting it to False (training each tree on the entire dataset) can sometimes improve performance.

In [40]:
```
# Random Forests with tuned parameters
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, max_depth=15, min_samples_sp
rf_model.fit(x_train_feature, y_train)
```

Out[40]:
```
RandomForestClassifier(max_depth=15, min_samples_split=10)
```

In [41]:
```
prediction_on_training_data = rf_model.predict(x_train_feature)
accuracy_on_training_data = accuracy_score(y_train , prediction_on_training_data
print("RF_accuracy_on_training_data is :",accuracy_on_training_data)

prediction_on_testing_data = rf_model.predict(x_test_feature)
accuracy_on_testing_data = accuracy_score(y_test , prediction_on_testing_data)
print("RF_accuracy_on_testing_data is :",accuracy_on_testing_data)

print("\nclassification_report RandomForest\n",classification_report(y_test , pr



print("\n Confusion Metrix of RandomForest \n",confusion_matrix(y_test , predict
conf_mat = confusion_matrix(y_test , prediction_on_testing_data)
plt.figure(figsize=(9,9))
sns.heatmap(conf_mat , annot=True , square=True)
plt.title("Confusion Metrix of RandomForest")
plt.show()
```

```
RF_accuracy_on_training_data is : 0.9382993044648867
RF_accuracy_on_testing_data is : 0.9291479820627803

classification_report RandomForest
              precision    recall  f1-score   support

           0       0.92      1.00      0.96       960
           1       1.00      0.49      0.66       155

    accuracy                           0.93      1115
   macro avg       0.96      0.75      0.81      1115
weighted avg       0.93      0.93      0.92      1115


 Confusion Metrix of RandomForest
 [[960   0]
 [ 79  76]]
```



In [ ]:

Accuracy of the Logistic Regression model on Testing data is 0.96681

Accuracy of the Super vector machine model on testing data is 0.8609

Accuracy of the random Forest Classifier model on testing data is 0.9228

In [ ]:

Demo of Email

1. XXXMobileMovieClub: To use your credit, click the WAP link in the next txt message or click here>> http://wap. xxxmobilemovieclub.com?n=QJKGIGHJJGCBL 2. I've been searching for the right words to thank you for this breather. I promise i wont take your help for granted and will fulfil my promise. You have been wonderful and a blessing at all times. 3. As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune 4.Thanks for your subscription to Ringtone UK your mobile will be charged £5/month Please confirm by replying YES or NO. If you reply NO you will not be charged

Building Prediction system

In [48]:
```
input_mail =["Thanks for your subscription to Ringtone UK your mobile will be ch

input_data_features = feature_extraction.transform(input_mail)

# making prediction

Logistic_Regression_predictions = Logistic_model.predict(input_data_features)

SVM_predictions = svm_model.predict(input_data_features)

Random_Forest_predictions = rf_model.predict(input_data_features)

# Print predictions for each model
print("Logistic Regression Prediction:", Logistic_Regression_predictions[0])
print("SVM Prediction:", SVM_predictions[0])
print("Random Forest Prediction:", Random_Forest_predictions[0])

# Decide on a final prediction (e.g., majority vote or based on one model)
# Example: Using Random Forest prediction as the final output
final_prediction = Random_Forest_predictions[0]

if final_prediction == 1:
    print("Spam mail")
else:
    print("Ham mail")
```

```
Logistic Regression Prediction: 1
SVM Prediction: 0
Random Forest Prediction: 0
Ham mail
```

In [ ]: