In [117…
```
#importing basic packages
import pandas as pd
import numpy as n
import seaborn as sns
import matplotlib.pyplot as plt
```

In [118…
```
#Loading the data
data0 = pd.read_csv("C:\\Users\\NITHYASRI\\Desktop\\phishing.csv")
data0.head()
```

Out[118]:

	Index	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	PrefixSuffix-	SubDomains	HTT
0	0	1	1	1	1	1	-1	0	
1	1	1	0	1	1	1	-1	-1	
2	2	1	0	1	1	1	-1	-1	
3	3	1	0	-1	1	1	-1	1	
4	4	-1	0	-1	1	-1	-1	1	

5 rows × 32 columns

In [119…
```
#Checking the shape of the dataset
data0.shape
```

Out[119]:
```
(11054, 32)
```

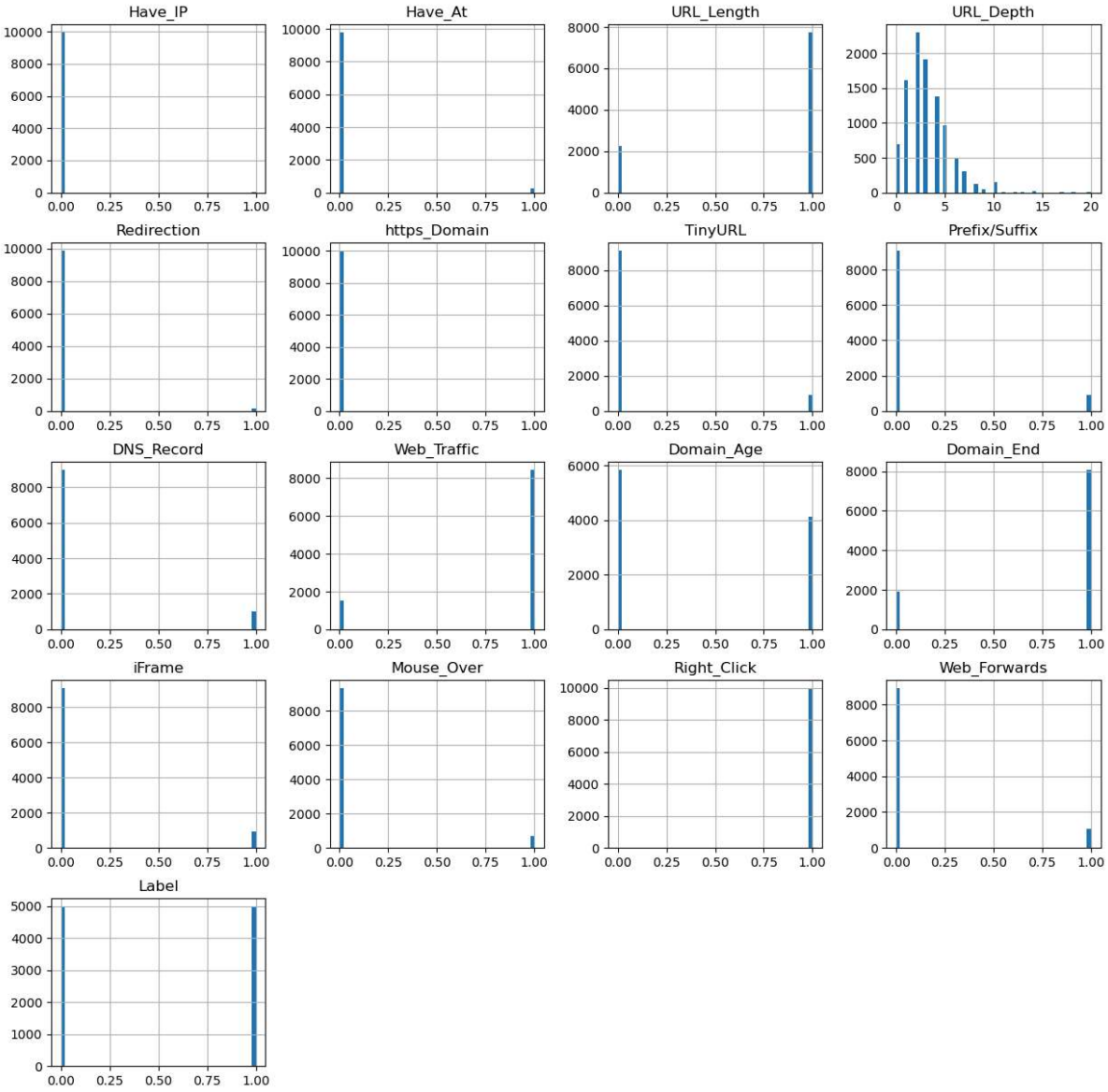
In [120…
```
#Listing the features of the dataset
data0.columns
```

Out[120]:
```
Index(['Index', 'UsingIP', 'LongURL', 'ShortURL', 'Symbol@', 'Redirecting//',
       'PrefixSuffix-', 'SubDomains', 'HTTPS', 'DomainRegLen', 'Favicon',
       'NonStdPort', 'HTTPSDomainURL', 'RequestURL', 'AnchorURL',
       'LinksInScriptTags', 'ServerFormHandler', 'InfoEmail', 'AbnormalURL',
       'WebsiteForwarding', 'StatusBarCust', 'DisableRightClick',
       'UsingPopupWindow', 'IframeRedirection', 'AgeofDomain', 'DNSRecording',
       'WebsiteTraffic', 'PageRank', 'GoogleIndex', 'LinksPointingToPage',
       'StatsReport', 'class'],
      dtype='object')
```

In [121…
```
#Information about the dataset
data0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11054 entries, 0 to 11053
Data columns (total 32 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Index                11054 non-null  int64
 1   UsingIP              11054 non-null  int64
 2   LongURL              11054 non-null  int64
 3   ShortURL             11054 non-null  int64
 4   Symbol@              11054 non-null  int64
 5   Redirecting//        11054 non-null  int64
 6   PrefixSuffix-        11054 non-null  int64
 7   SubDomains           11054 non-null  int64
 8   HTTPS                11054 non-null  int64
 9   DomainRegLen         11054 non-null  int64
 10  Favicon              11054 non-null  int64
 11  NonStdPort           11054 non-null  int64
 12  HTTPSDomainURL       11054 non-null  int64
 13  RequestURL           11054 non-null  int64
 14  AnchorURL            11054 non-null  int64
 15  LinksInScriptTags    11054 non-null  int64
 16  ServerFormHandler    11054 non-null  int64
 17  InfoEmail            11054 non-null  int64
 18  AbnormalURL          11054 non-null  int64
 19  WebsiteForwarding    11054 non-null  int64
 20  StatusBarCust        11054 non-null  int64
 21  DisableRightClick    11054 non-null  int64
 22  UsingPopupWindow     11054 non-null  int64
 23  IframeRedirection    11054 non-null  int64
 24  AgeofDomain          11054 non-null  int64
 25  DNSRecording         11054 non-null  int64
 26  WebsiteTraffic       11054 non-null  int64
 27  PageRank             11054 non-null  int64
 28  GoogleIndex          11054 non-null  int64
 29  LinksPointingToPage  11054 non-null  int64
 30  StatsReport          11054 non-null  int64
 31  class                11054 non-null  int64
dtypes: int64(32)
memory usage: 2.7 MB
```

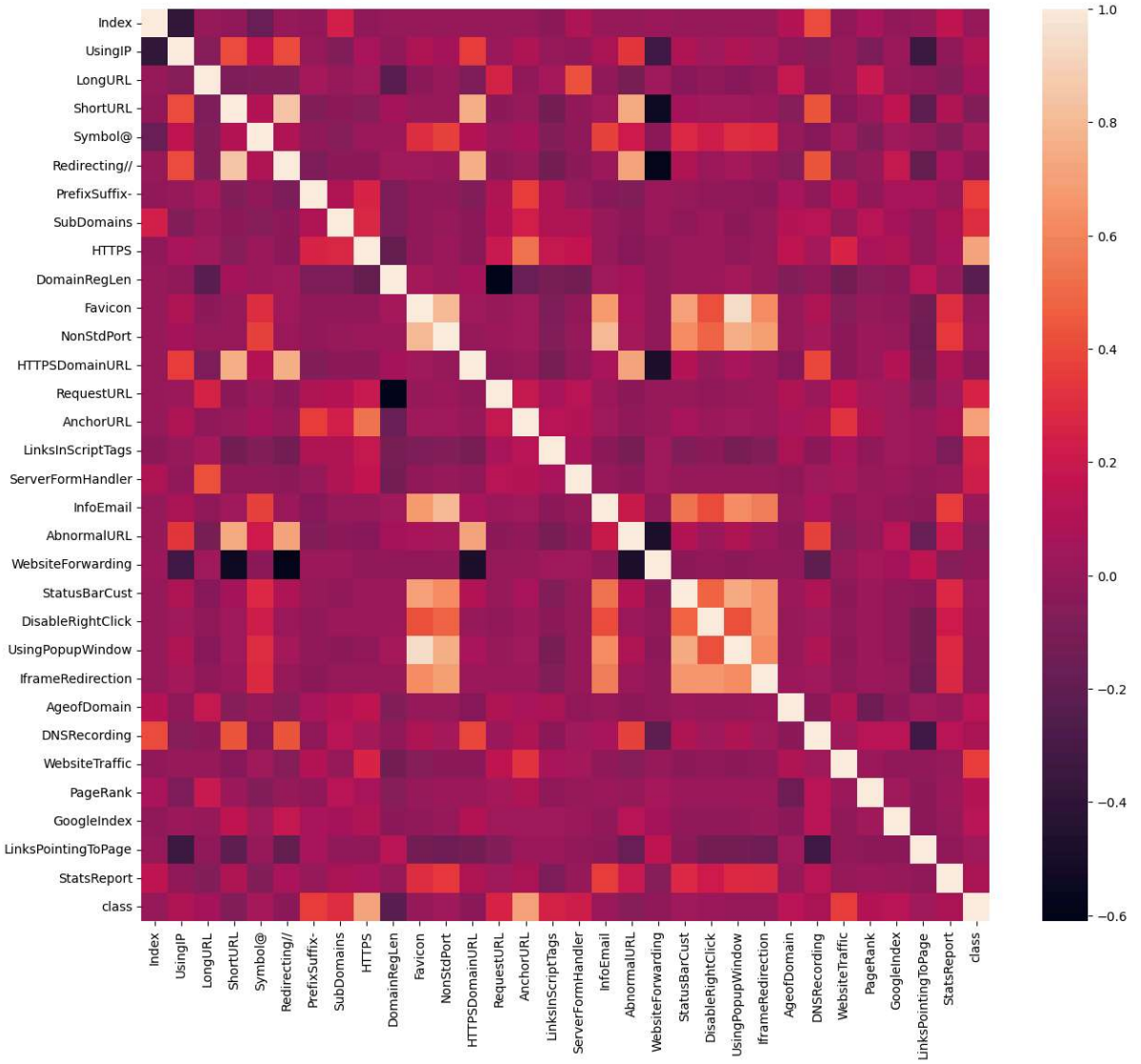
In [9]:
```
#Plotting the data distribution
data0.hist(bins = 50,figsize = (15,15))
plt.show()
```



In [122…

```
#Correlation heatmap

plt.figure(figsize=(15,13))
sns.heatmap(data0.corr())
plt.show()
```



In [123…
```
data0.describe()
```

Out[123]:

	Index	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	Prefi
count	11054.000000	11054.000000	11054.000000	11054.000000	11054.000000	11054.000000	11054
mean	5526.500000	0.313914	-0.633345	0.738737	0.700561	0.741632	-0
std	3191.159272	0.949495	0.765973	0.674024	0.713625	0.670837	0
min	0.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1
25%	2763.250000	-1.000000	-1.000000	1.000000	1.000000	1.000000	-1
50%	5526.500000	1.000000	-1.000000	1.000000	1.000000	1.000000	-1
75%	8289.750000	1.000000	-1.000000	1.000000	1.000000	1.000000	-1
max	11053.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

8 rows × 32 columns

In [125…
```
#Dropping the Domain column
data = data0.drop(['Index'], axis = 1).copy()
```

In [126…
```
#checking the data for null or missing values
data.isnull().sum()
```

Out[126]:
```
UsingIP               0
LongURL               0
ShortURL              0
Symbol@               0
Redirecting//         0
PrefixSuffix-         0
SubDomains            0
HTTPS                 0
DomainRegLen          0
Favicon               0
NonStdPort            0
HTTPSDomainURL        0
RequestURL            0
AnchorURL             0
LinksInScriptTags     0
ServerFormHandler     0
InfoEmail             0
AbnormalURL           0
WebsiteForwarding     0
StatusBarCust         0
DisableRightClick     0
UsingPopupWindow      0
IframeRedirection     0
AgeofDomain           0
DNSRecording          0
WebsiteTraffic        0
PageRank              0
GoogleIndex           0
LinksPointingToPage   0
StatsReport           0
class                 0
dtype: int64
```

In [129…
```
# shuffling the rows in the dataset so that when splitting the train and test set d
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

Out[129]:

	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	PrefixSuffix-	SubDomains	HTTPS	Dor
0	1	-1	1	1	1	1	-1	1	
1	-1	-1	1	1	1	-1	0	-1	
2	-1	1	1	1	-1	1	1	1	
3	1	-1	1	1	1	-1	-1	1	
4	1	-1	1	1	1	-1	0	1	

5 rows × 31 columns

In [131…
```
# Sepratating & assigning features and target columns to X & y
y = data['class']
X = data.drop('class',axis=1)
X.shape, y.shape
```

Out[131]:
```
((11054, 30), (11054,))
```

In [132…
```
# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
                                                    test_size = 0.2, random_state =
X_train.shape, X_test.shape
```

Out[132]:
```
((8843, 30), (2211, 30))
```

In [133…
```
#importing packages
from sklearn.metrics import accuracy_score
```

In [134…
```
# Creating holders to store the model performance results
ML_Model = []
acc_train = []
acc_test = []

#function to call for storing the results
def storeResults(model, a,b):
  ML_Model.append(model)
  acc_train.append(round(a, 3))
  acc_test.append(round(b, 3))
```

In [135…
```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth = 10)
# fit the model
tree.fit(X_train, y_train)
```

Out[135]:
```
DecisionTreeClassifier(max_depth=10)
```

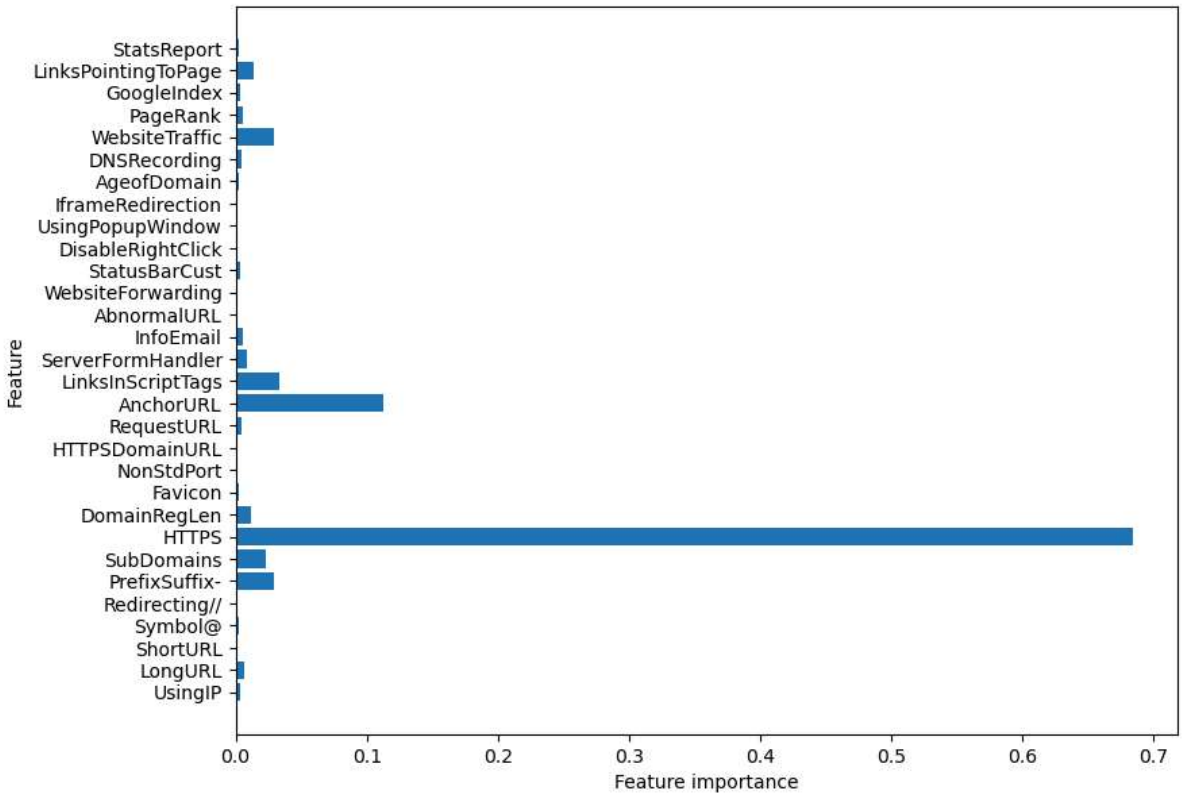
In [136…
```
#predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)
```

In [137…
```
#computing the accuracy of the model performance
acc_train_tree = accuracy_score(y_train,y_train_tree)
acc_test_tree = accuracy_score(y_test,y_test_tree)

print("Decision Tree: Accuracy on training Data: {:.3f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data: {:.3f}".format(acc_test_tree))
```

```
Decision Tree: Accuracy on training Data: 0.961
Decision Tree: Accuracy on test Data: 0.948
```

In [138…
```
#checking the feature improtance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), tree.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



In [139…
```
#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('Decision Tree', acc_train_tree, acc_test_tree)
```

In [140…
```
# Random Forest model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(max_depth=150)

# fit the model
forest.fit(X_train, y_train)
```

Out[140]:
```
RandomForestClassifier(max_depth=150)
```

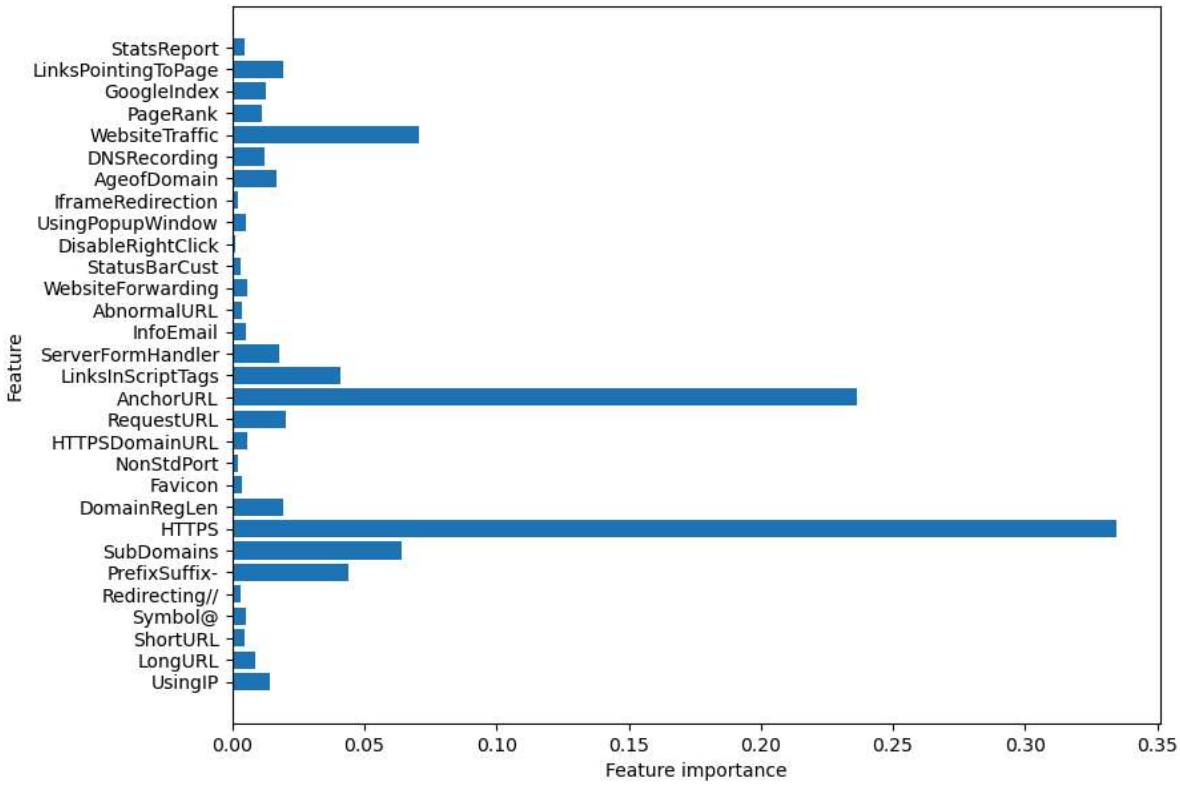
In [141…
```
#predicting the target value from the model for the samples
y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)
```

In [142…
```
#computing the accuracy of the model performance
acc_train_forest = accuracy_score(y_train,y_train_forest)
acc_test_forest = accuracy_score(y_test,y_test_forest)

print("Random forest: Accuracy on training Data: {:.3f}".format(acc_train_forest))
print("Random forest: Accuracy on test Data: {:.3f}".format(acc_test_forest))
```

```
Random forest: Accuracy on training Data: 0.991
Random forest: Accuracy on test Data: 0.971
```

In [143…
```
#checking the feature improtance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), forest.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



In [144…
```
#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('Random Forest', acc_train_forest, acc_test_forest)
```

In [145…
```
# Multilayer Perceptrons model
from sklearn.neural_network import MLPClassifier

# instantiate the model
mlp = MLPClassifier(alpha=0.001, hidden_layer_sizes=([100,100,100]))

# fit the model
mlp.fit(X_train, y_train)
```

Out[145]:
```
MLPClassifier(alpha=0.001, hidden_layer_sizes=[100, 100, 100])
```

In [146…
```
#predicting the target value from the model for the samples
y_test_mlp = mlp.predict(X_test)
y_train_mlp = mlp.predict(X_train)
```

In [147…
```
#computing the accuracy of the model performance
acc_train_mlp = accuracy_score(y_train,y_train_mlp)
acc_test_mlp = accuracy_score(y_test,y_test_mlp)

print("Multilayer Perceptrons: Accuracy on training Data: {:.3f}".format(acc_train_
print("Multilayer Perceptrons: Accuracy on test Data: {:.3f}".format(acc_test_mlp))
```

```
Multilayer Perceptrons: Accuracy on training Data: 0.989
Multilayer Perceptrons: Accuracy on test Data: 0.967
```

In [148…
```
#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('Multilayer Perceptrons', acc_train_mlp, acc_test_mlp)
```

In [161…
```
#Support vector machine model
from sklearn.svm import SVC

# instantiate the model
```

```
svm = SVC(kernel='linear', C=1.0, random_state=12)
#fit the model
svm.fit(X_train, y_train)
```

Out[161]:
```
SVC(kernel='linear', random_state=12)
```

In [162…
```
#predicting the target value from the model for the samples
y_test_svm = svm.predict(X_test)
y_train_svm = svm.predict(X_train)
```

In [163…
```
#computing the accuracy of the model performance
acc_train_svm = accuracy_score(y_train,y_train_svm)
acc_test_svm = accuracy_score(y_test,y_test_svm)

print("SVM: Accuracy on training Data: {:.3f}".format(acc_train_svm))
print("SVM : Accuracy on test Data: {:.3f}".format(acc_test_svm))
```

```
SVM: Accuracy on training Data: 0.928
SVM : Accuracy on test Data: 0.928
```

In [164…
```
#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('SVM', acc_train_svm, acc_test_svm)
```

In [165…
```
#creating dataframe
results = pd.DataFrame({ 'ML Model': ML_Model,
    'Train Accuracy': acc_train,
    'Test Accuracy': acc_test})
results
```

Out[165]:

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	0.961	0.948
1	Random Forest	0.991	0.971
2	Multilayer Perceptrons	0.989	0.967
3	SVM	0.928	0.928

In [166…
```
#Sorting the datafram on accuracy
results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=False)
```

Out[166]:

	ML Model	Train Accuracy	Test Accuracy
1	Random Forest	0.991	0.971
2	Multilayer Perceptrons	0.989	0.967
0	Decision Tree	0.961	0.948
3	SVM	0.928	0.928

In [ ]:
```
```