```
#importing basic packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#Loading the data
data0 = pd.read_csv('5.urldata.csv')
data0.head()
```

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Do
0	graphicriver.net	0	0	1	1	0	
1	ecnavi.jp	0	0	1	1	1	
2	hubpages.com	0	0	1	1	0	
3	extratorrent.cc	0	0	1	3	0	
4	icicibank.com	0	0	1	3	0	

```
#Checking the shape of the dataset
data0.shape
```

```
(10000, 18)
```

```
#Listing the features of the dataset
data0.columns
```

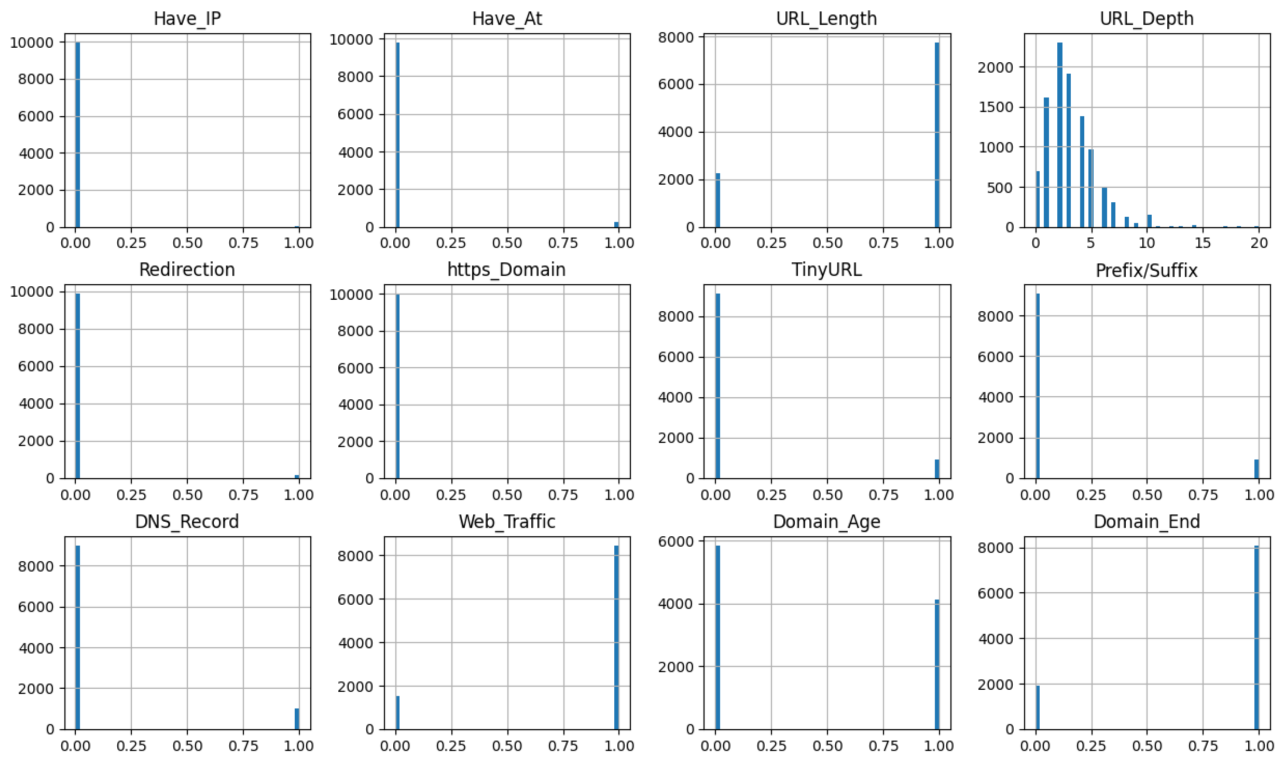
```
Index(['Domain', 'Have_IP', 'Have_At', 'URL_Length', 'URL_Depth',
       'Redirection', 'https_Domain', 'TinyURL', 'Prefix/Suffix', 'DNS_Record',
       'Web_Traffic', 'Domain_Age', 'Domain_End', 'iFrame', 'Mouse_Over',
       'Right_Click', 'Web_Forwards', 'Label'],
      dtype='object')
```

```
#Information about the dataset
data0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Domain         10000 non-null  object
 1   Have_IP        10000 non-null  int64
 2   Have_At        10000 non-null  int64
 3   URL_Length     10000 non-null  int64
 4   URL_Depth      10000 non-null  int64
 5   Redirection    10000 non-null  int64
```

```
 6   https_Domain   10000 non-null  int64
 7   TinyURL        10000 non-null  int64
 8   Prefix/Suffix  10000 non-null  int64
 9   DNS_Record     10000 non-null  int64
 10  Web_Traffic    10000 non-null  int64
 11  Domain_Age     10000 non-null  int64
 12  Domain_End     10000 non-null  int64
 13  iFrame         10000 non-null  int64
 14  Mouse_Over     10000 non-null  int64
 15  Right_Click    10000 non-null  int64
 16  Web_Forwards   10000 non-null  int64
 17  Label          10000 non-null  int64
dtypes: int64(17), object(1)
memory usage: 1.4+ MB
```

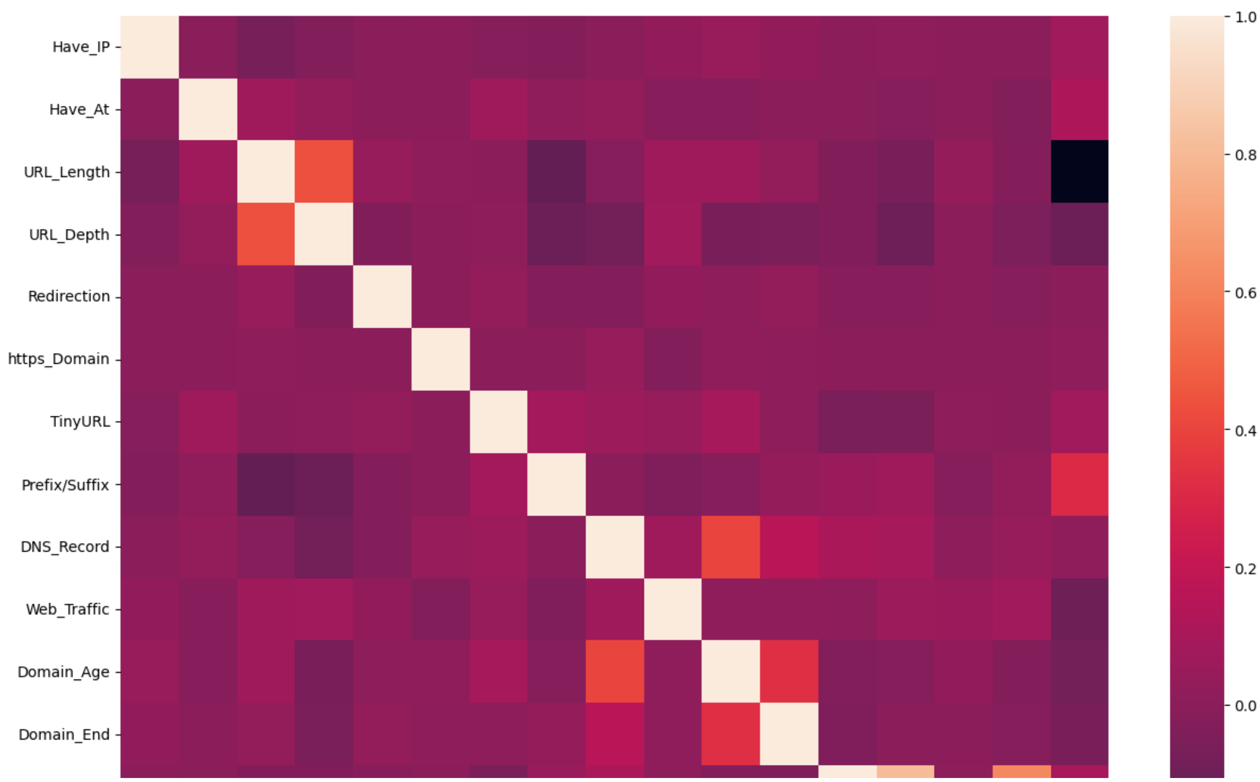
```
#Plotting the data distribution
data0.hist(bins = 50,figsize = (15,15))
plt.show()
```



```
#Correlation heatmap

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

```
<ipython-input-60-91fdee13ed62>:4: FutureWarning: The default value of numeri
  sns.heatmap(data0.corr())
```



```
data0.describe()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Do
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0
mean	0.005500	0.022600	0.773400	3.072000	0.013500	0.0
std	0.073961	0.148632	0.418653	2.128631	0.115408	0.0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.000000	1.000000	2.000000	0.000000	0.0
50%	0.000000	0.000000	1.000000	3.000000	0.000000	0.0
75%	0.000000	0.000000	1.000000	4.000000	0.000000	0.0
max	1.000000	1.000000	1.000000	20.000000	1.000000	1.0

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

```
#checking the data for null or missing values
data.isnull().sum()
```

```
Have_IP          0
Have_At          0
URL_Length       0
URL_Depth        0
Redirection      0
https_Domain     0
TinyURL          0
Prefix/Suffix    0
DNS_Record       0
Web_Traffic      0
Domain_Age       0
Domain_End       0
iFrame           0
Mouse_Over       0
Right_Click      0
Web_Forwards     0
Label            0
dtype: int64
```

```
# shuffling the rows in the dataset so that when splitting the train and test set are equa
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL
0	0	0	1	2	0	0	0
1	0	0	1	3	0	0	0
2	0	0	1	5	0	0	0
3	0	0	1	3	0	0	1
4	0	0	0	3	0	0	0

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

```
((10000, 16), (10000,))
```

```
# 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 = 12)
X_train.shape, X_test.shape
```

```
((8000, 16), (2000, 16))
```

```
#importing packages
from sklearn.metrics import accuracy_score
```

```
# 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))
```

```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)
```

```
▾        DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

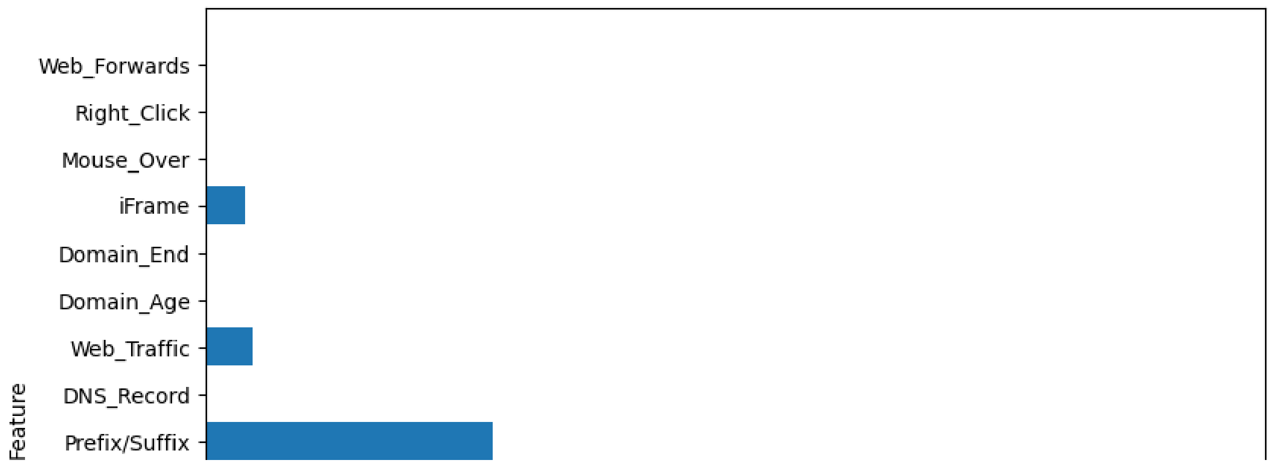
```
#predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)
```

```
#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: 0.814
Decision Tree: Accuracy on test Data: 0.810
```

```
#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()
```




```
#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)
```

```
# Random Forest model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(max_depth=5)

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

```
RandomForestClassifier
RandomForestClassifier(max_depth=5)
```

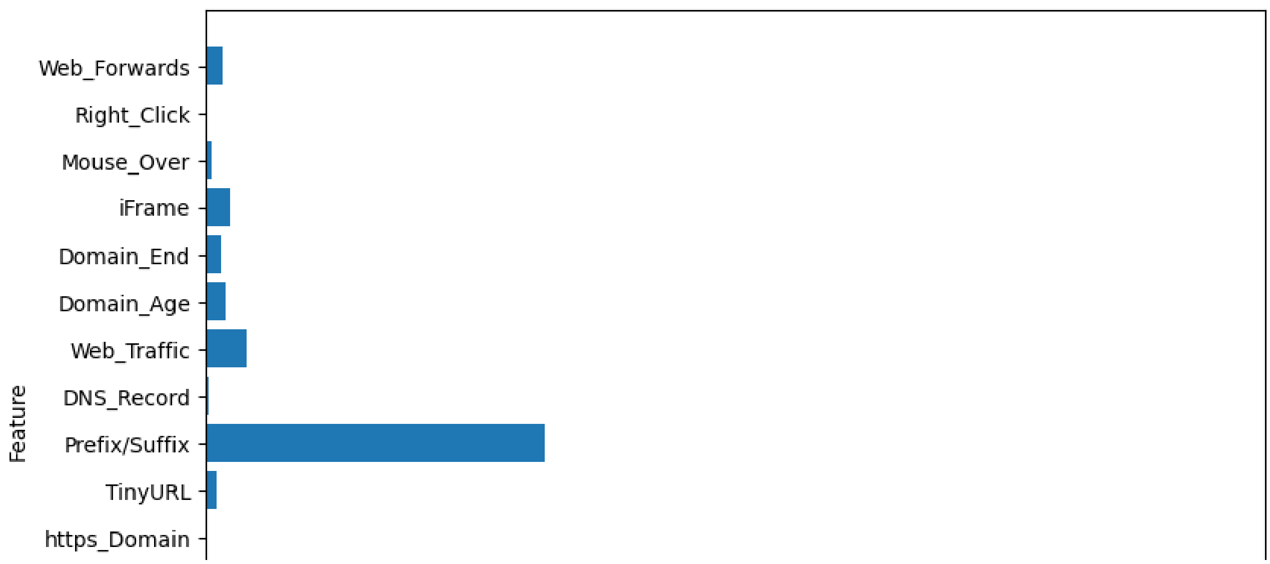
```
#predicting the target value from the model for the samples
y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)
```

```
#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: 0.818
Random forest: Accuracy on test Data: 0.807
```

```
#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()
```



```
#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)


# 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)
```

```
MLPClassifier

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

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


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

```
Multilayer Perceptrons: Accuracy on training Data: 0.864
Multilayer Perceptrons: Accuracy on test Data: 0.854
```

```
#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)


#XGBoost Classification model
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier(learning_rate=0.4,max_depth=7)
#fit the model
xgb.fit(X_train, y_train)
```

```
▾                                    XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
```

```
#predicting the target value from the model for the samples
y_test_xgb = xgb.predict(X_test)
y_train_xgb = xgb.predict(X_train)
```

```
#computing the accuracy of the model performance
acc_train_xgb = accuracy_score(y_train,y_train_xgb)
acc_test_xgb = accuracy_score(y_test,y_test_xgb)

print("XGBoost: Accuracy on training Data: {:.3f}".format(acc_train_xgb))
print("XGBoost : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
```

```
XGBoost: Accuracy on training Data: 0.870
XGBoost : Accuracy on test Data: 0.853
```

```
#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('XGBoost', acc_train_xgb, acc_test_xgb)
```

```
#importing required packages
import keras
from keras.layers import Input, Dense
from keras import regularizers
import tensorflow as tf
from keras.models import Model
from sklearn import metrics
```

```
#building autoencoder model

input_dim = X_train.shape[1]
encoding_dim = input_dim

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="relu",
                activity_regularizer=regularizers.l1(10e-4))(input_layer)
encoder = Dense(int(encoding_dim), activation="relu")(encoder)

encoder = Dense(int(encoding_dim-2), activation="relu")(encoder)
code = Dense(int(encoding_dim-4), activation='relu')(encoder)
decoder = Dense(int(encoding_dim-2), activation='relu')(code)

decoder = Dense(int(encoding_dim), activation='relu')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 input_3 (InputLayer)        [(None, 16)]              0

 dense_14 (Dense)            (None, 16)                272

 dense_15 (Dense)            (None, 16)                272

 dense_16 (Dense)            (None, 14)                238

 dense_19 (Dense)            (None, 16)                240

 dense_20 (Dense)            (None, 16)                272

=================================================================
Total params: 1294 (5.05 KB)
Trainable params: 1294 (5.05 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#compiling the model
autoencoder.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

#Training the model
history = autoencoder.fit(X_train, X_train, epochs=10, batch_size=64, shuffle=True, valida
```

```
Epoch 1/10
100/100 [==============================] - 1s 3ms/step - loss: 3.5222 - accuracy
Epoch 2/10
100/100 [==============================] - 0s 2ms/step - loss: 3.3239 - accuracy
Epoch 3/10
100/100 [==============================] - 0s 2ms/step - loss: 3.2731 - accuracy
Epoch 4/10
100/100 [==============================] - 0s 2ms/step - loss: 3.2348 - accuracy
Epoch 5/10
100/100 [==============================] - 0s 2ms/step - loss: 3.2077 - accuracy
Epoch 6/10
100/100 [==============================] - 0s 2ms/step - loss: 3.1878 - accuracy
Epoch 7/10
100/100 [==============================] - 0s 2ms/step - loss: 3.1729 - accuracy
Epoch 8/10
100/100 [==============================] - 0s 2ms/step - loss: 3.1622 - accuracy
Epoch 9/10
100/100 [==============================] - 0s 2ms/step - loss: 3.1538 - accuracy
Epoch 10/10
100/100 [==============================] - 0s 2ms/step - loss: 3.1482 - accuracy
```

```
acc_train_auto = autoencoder.evaluate(X_train, X_train)[1]
acc_test_auto = autoencoder.evaluate(X_test, X_test)[1]

print('\nAutoencoder: Accuracy on training Data: {:.3f}' .format(acc_train_auto))
print('Autoencoder: Accuracy on test Data: {:.3f}' .format(acc_test_auto))
```

```
250/250 [==============================] - 0s 1ms/step - loss: 3.1295 - accurac
63/63 [==============================] - 0s 1ms/step - loss: 3.0993 - accuracy:

Autoencoder: Accuracy on training Data: 0.001
Autoencoder: Accuracy on test Data: 0.003
```

```
#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('AutoEncoder', acc_train_auto, acc_test_auto)
```

```
#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)
```

```
SVC

SVC(kernel='linear', random_state=12)
```

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

```
#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: 0.803
SVM : Accuracy on test Data: 0.796
```

```
#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)
```

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

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	0.814	0.810
1	Random Forest	0.818	0.807
2	Multilayer Perceptrons	0.864	0.854
3	XGBoost	0.870	0.852
4	AutoEncoder	0.001	0.003

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

	ML Model	Train Accuracy	Test Accuracy
2	Multilayer Perceptrons	0.864	0.854
3	XGBoost	0.870	0.852
0	Decision Tree	0.814	0.810
1	Random Forest	0.818	0.807
5	SVM	0.804	0.796
4	AutoEncoder	0.001	0.003

```
# save XGBoost model to file
import pickle
pickle.dump(xgb, open("XGBoostClassifier.pickle.dat", "wb"))
```

```
# load model from file
loaded_model = pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))
loaded_model
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None
              enable_categorical=False, eval_metric=None, feature_types=None
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.4, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=7, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```