IMPORTING LIBRARIES

In [1]:
```
#Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
#Data Loading
Data = pd.read_csv('Data.csv')
Data.head()
```

Out[2]:

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

DATA ANALYSIS

In [3]:
```
Data.shape
```

Out[3]:
```
(10000, 19)
```

In [4]:
```
Data.columns
```

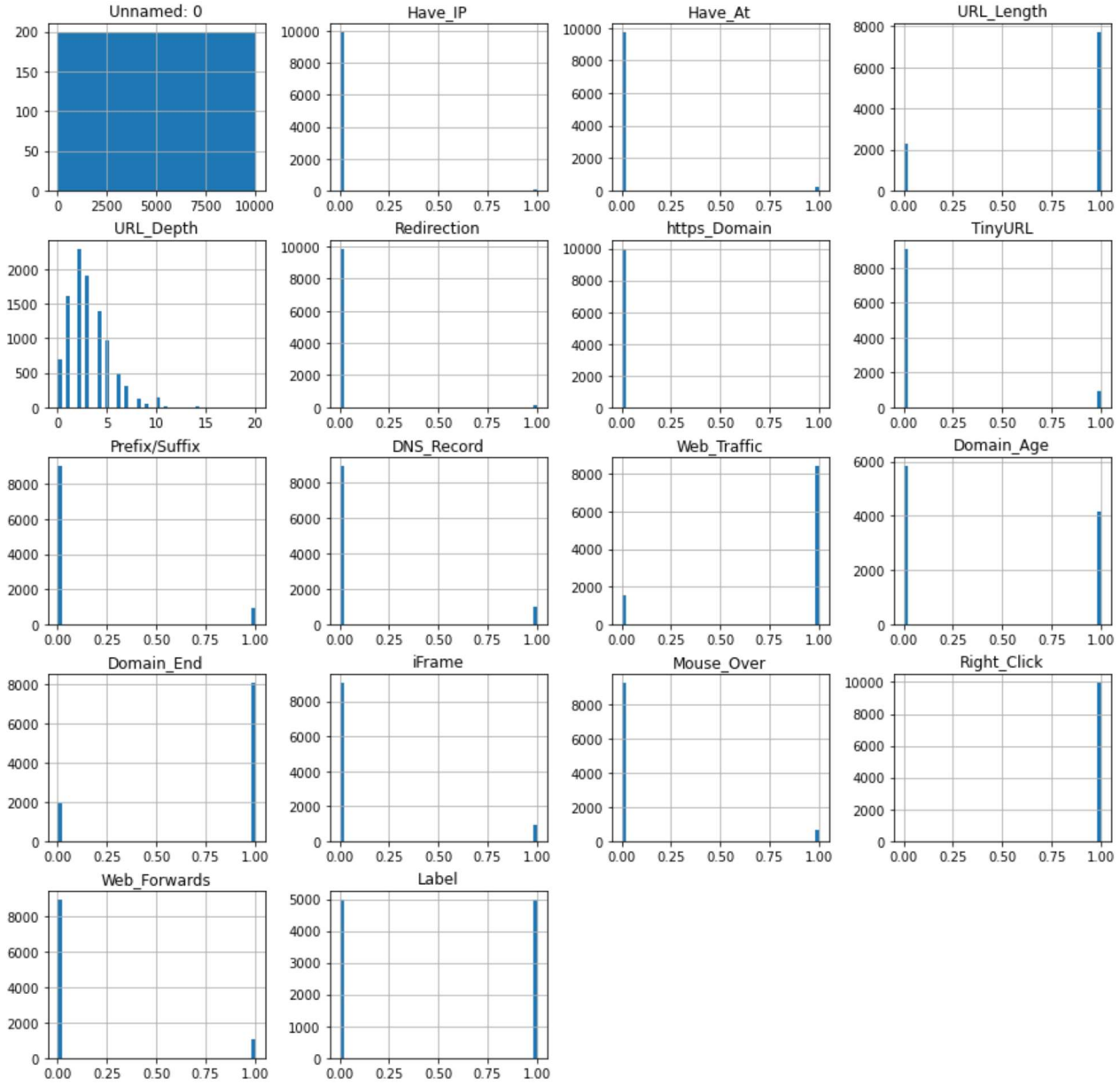
Out[4]:
```
Index(['Unnamed: 0', 'Domain', 'Have_IP', 'Have_At', 'URL_Length', 'URL_Depth',
       'Redirection', 'https_Domain', 'TinyURL', 'Prefix/Suffix', 'DNS_Record',
       'Web_Traffic', 'Domain_Age', 'Domain_End', 'iFrame', 'Mouse_Over',
       'Right_Click', 'Web_Forwards', 'Label'],
      dtype='object')
```

In [5]:
```
Data.info()
```

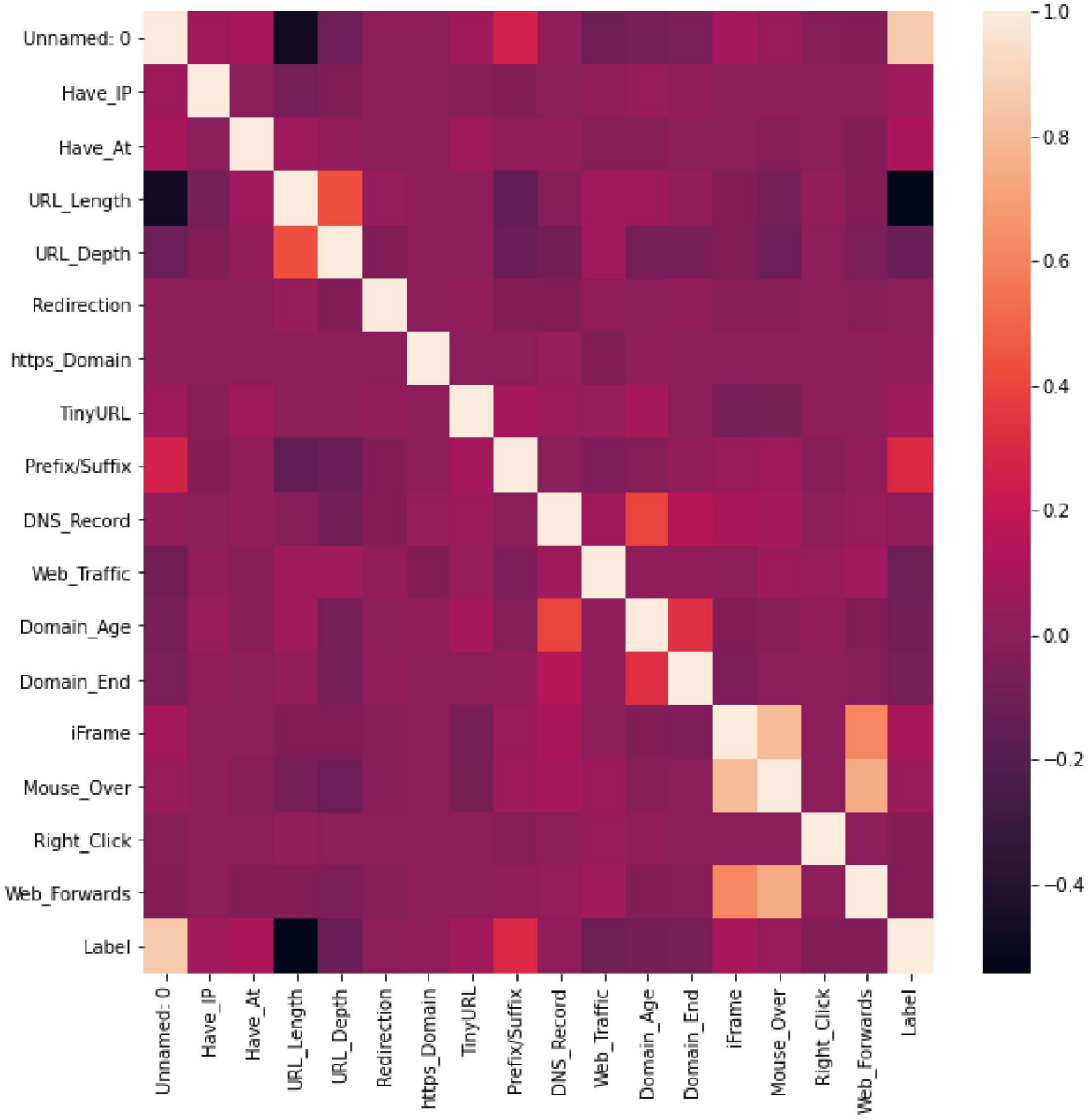
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Unnamed: 0     10000 non-null  int64 
 1   Domain         10000 non-null  object
 2   Have_IP        10000 non-null  int64 
 3   Have_At        10000 non-null  int64 
 4   URL_Length     10000 non-null  int64 
 5   URL_Depth      10000 non-null  int64 
 6   Redirection    10000 non-null  int64 
 7   https_Domain   10000 non-null  int64 
 8   TinyURL        10000 non-null  int64 
```

```
 9   Prefix/Suffix  10000 non-null  int64
 10  DNS_Record     10000 non-null  int64
 11  Web_Traffic    10000 non-null  int64
 12  Domain_Age     10000 non-null  int64
 13  Domain_End     10000 non-null  int64
 14  iFrame         10000 non-null  int64
 15  Mouse_Over     10000 non-null  int64
 16  Right_Click    10000 non-null  int64
 17  Web_Forwards   10000 non-null  int64
 18  Label          10000 non-null  int64
dtypes: int64(18), object(1)
memory usage: 1.4+ MB
```

In [6]:
```
Data.hist(bins = 50,figsize = (15,15))
plt.show()
```



In [7]:
```
#Heatmap Plotting
plt.figure(figsize=(10,10))
sns.heatmap(Data.corr())
plt.show()
```



In [8]:
```
Data.describe()
```

Out[8]:

	Unnamed: 0	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domai
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000
mean	4999.50000	0.005500	0.022600	0.773400	3.072000	0.013500	0.00020
std	2886.89568	0.073961	0.148632	0.418653	2.128631	0.115408	0.01414
min	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	2499.75000	0.000000	0.000000	1.000000	2.000000	0.000000	0.00000
50%	4999.50000	0.000000	0.000000	1.000000	3.000000	0.000000	0.00000
75%	7499.25000	0.000000	0.000000	1.000000	4.000000	0.000000	0.00000
max	9999.00000	1.000000	1.000000	1.000000	20.000000	1.000000	1.00000

In [9]:
```
Data_1 = Data.drop(['Domain'], axis = 1).copy()
```

In [10]:
```
Data.isnull().sum()
```

Out[10]:
```
Unnamed: 0       0
Domain           0
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
```

In [11]:
```
Data_1 = Data_1.sample(frac=1).reset_index(drop=True)
Data_1.head()
```

Out[11]:

	Unnamed: 0	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/S
0	8574	0	0	1	3	0	0	0	
1	8324	0	0	1	4	0	0	0	
2	468	0	0	1	2	0	0	0	
3	8294	0	0	0	0	0	0	0	
4	7551	0	0	1	3	0	0	0	

In [12]:
```
y = Data_1['Label']
X = Data_1.drop('Label',axis=1)
X.shape, y.shape
```

Out[12]:
```
((10000, 17), (10000,))
```

In [13]:
```
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[13]:
```
((8000, 17), (2000, 17))
```

In [14]:
```
from sklearn.metrics import accuracy_score
```

In [15]:
```
MLModel = []
Train = []
```

```
Test = []
def storeResults(model,a,b):
  MLModel.append(model)
  Train.append(round(a, 3))
  Test.append(round(b, 3))
```

DECISION TREE

In [16]:
```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
Tree = DecisionTreeClassifier(max_depth = 5)
Tree.fit(X_train, y_train)
```

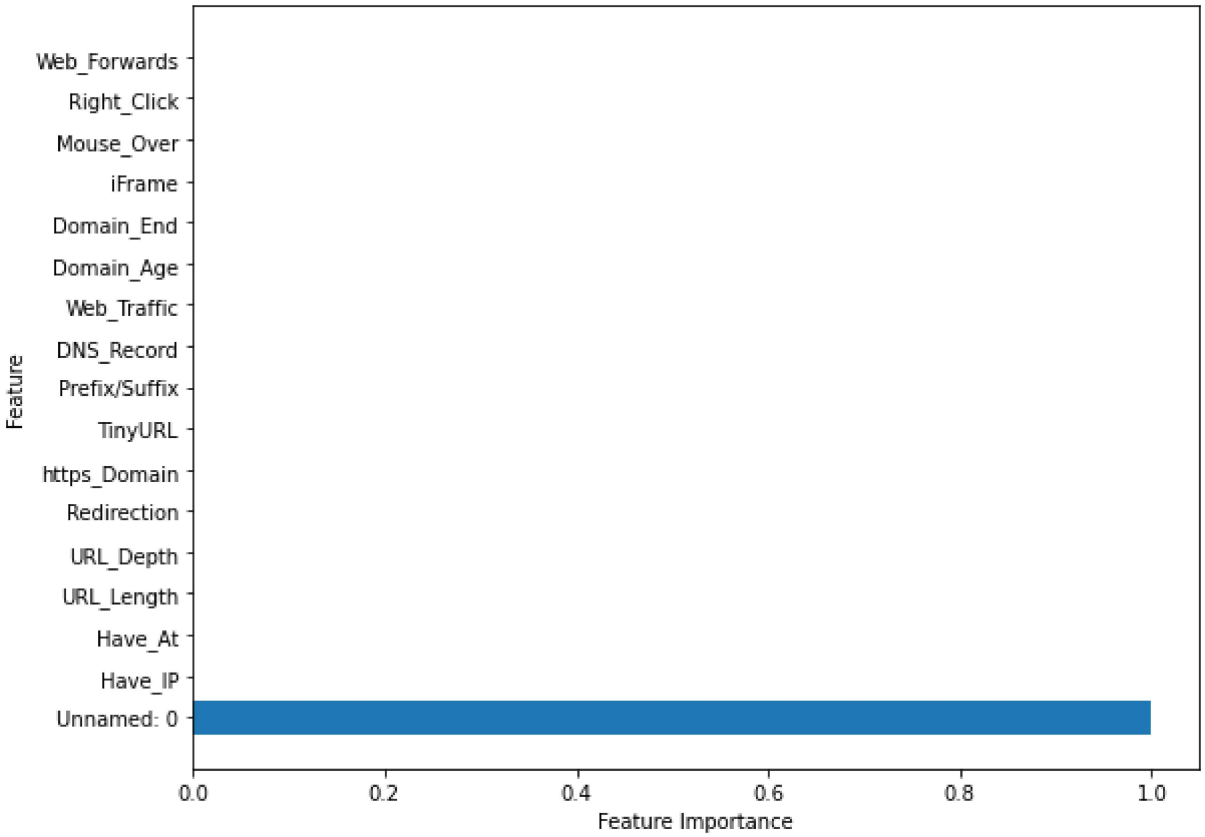
Out[16]:
```
DecisionTreeClassifier(max_depth=5)
```

In [17]:
```
y_test_tree = Tree.predict(X_test)
y_train_tree = Tree.predict(X_train)
```

In [18]:
```
#Accuracy Score
Train_tree = accuracy_score(y_train,y_train_tree)
Test_tree = accuracy_score(y_test,y_test_tree)
print("Decision Tree: Accuracy on training Data: {:.3f}".format(Train_tree))
print("Decision Tree: Accuracy on test Data: {:.3f}".format(Test_tree))
```

```
Decision Tree: Accuracy on training Data: 1.000
Decision Tree: Accuracy on test Data: 1.000
```

In [19]:
```
plt.figure(figsize=(9,7))
num_features = X_train.shape[1]
plt.barh(range(num_features), Tree.feature_importances_, align='center')
plt.yticks(np.arange(num_features), X_train.columns)
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.show()
```



In [20]:
```
storeResults('Decision Tree', Train_tree, Test_tree)
```

RANDOM FOREST CLASSIFIER

In [21]:
```
#Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
Forest = RandomForestClassifier(max_depth=5)
Forest.fit(X_train, y_train)
```

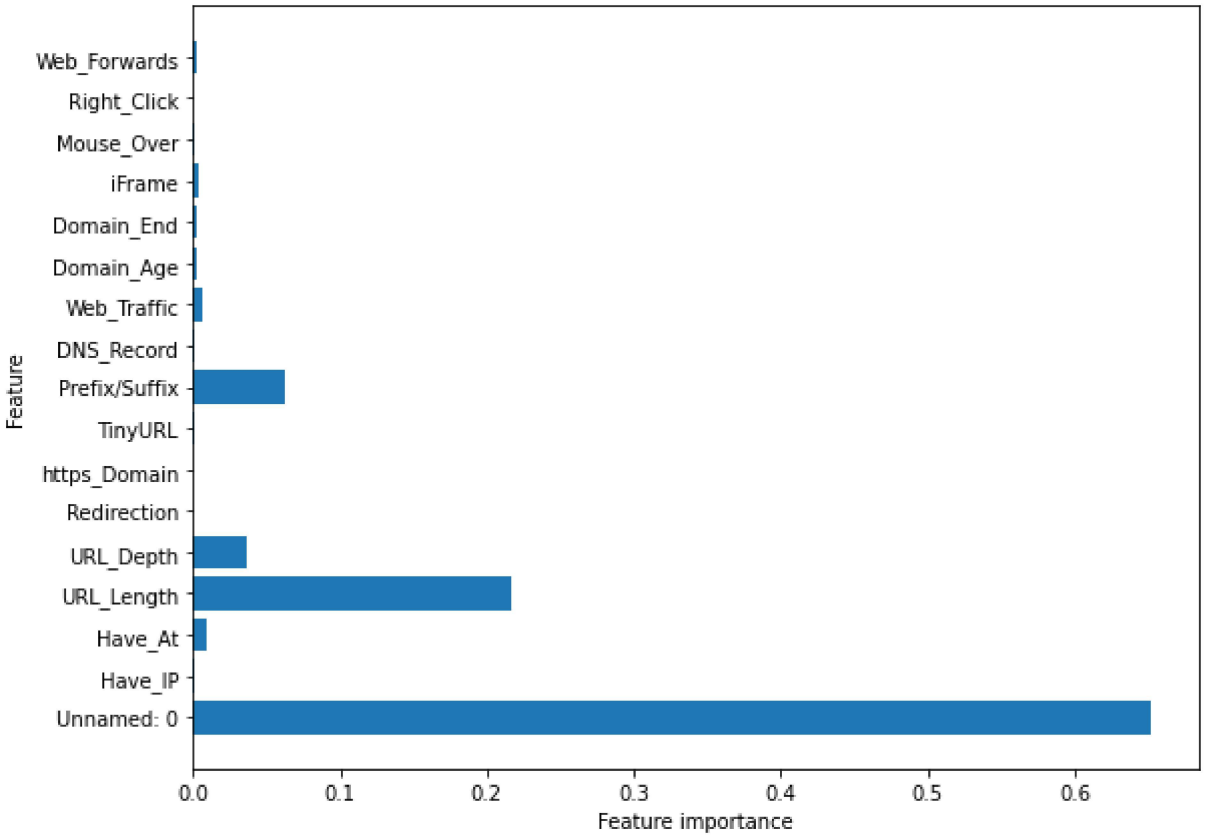
Out[21]:
```
RandomForestClassifier(max_depth=5)
```

In [22]:
```
y_test_forest = Forest.predict(X_test)
y_train_forest = Forest.predict(X_train)
```

In [23]:
```
#Random Forest Accuracy
Train_forest = accuracy_score(y_train,y_train_forest)
Test_forest = accuracy_score(y_test,y_test_forest)
print("Random forest: Accuracy on training Data: {:.3f}".format(Train_forest))
print("Random forest: Accuracy on test Data: {:.3f}".format(Test_forest))
```

```
Random forest: Accuracy on training Data: 1.000
Random forest: Accuracy on test Data: 1.000
```

In [24]:
```
plt.figure(figsize=(9,7))
num_features = X_train.shape[1]
plt.barh(range(num_features), Forest.feature_importances_, align='center')
plt.yticks(np.arange(num_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



In [25]:
```
storeResults('Random Forest', Train_forest, Test_forest)
```

MULTILAYER PERCEPTRON

In [26]:
```
#MLP Classifier
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(alpha=0.001, hidden_layer_sizes=([100,100,100]))
mlp.fit(X_train, y_train)
```

Out[26]:
```
MLPClassifier(alpha=0.001, hidden_layer_sizes=[100, 100, 100])
```

In [27]:
```
y_test_mlp = mlp.predict(X_test)
y_train_mlp = mlp.predict(X_train)
```

In [28]:
```
#Multilayer Perceptrons
Train_mlp = accuracy_score(y_train,y_train_mlp)
Test_mlp = accuracy_score(y_test,y_test_mlp)
print("Multilayer Perceptrons: Accuracy on training Data: {:.3f}".format(Train_mlp))
print("Multilayer Perceptrons: Accuracy on test Data: {:.3f}".format(Test_mlp))
```

```
Multilayer Perceptrons: Accuracy on training Data: 0.968
Multilayer Perceptrons: Accuracy on test Data: 0.970
```

In [29]:
```
storeResults('Multilayer Perceptrons', Train_mlp, Test_mlp)
```

In [30]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\91787\anaconda3\lib\site-packages (1.
6.1)
Requirement already satisfied: numpy in c:\users\91787\anaconda3\lib\site-packages (from
```

```
xgboost) (1.19.5)
Requirement already satisfied: scipy in c:\users\91787\anaconda3\lib\site-packages (from
xgboost) (1.5.2)
```

XG BOOST CLASSIFIER

In [31]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(learning_rate=0.25,max_depth=7)
xgb.fit(X_train, y_train)
```

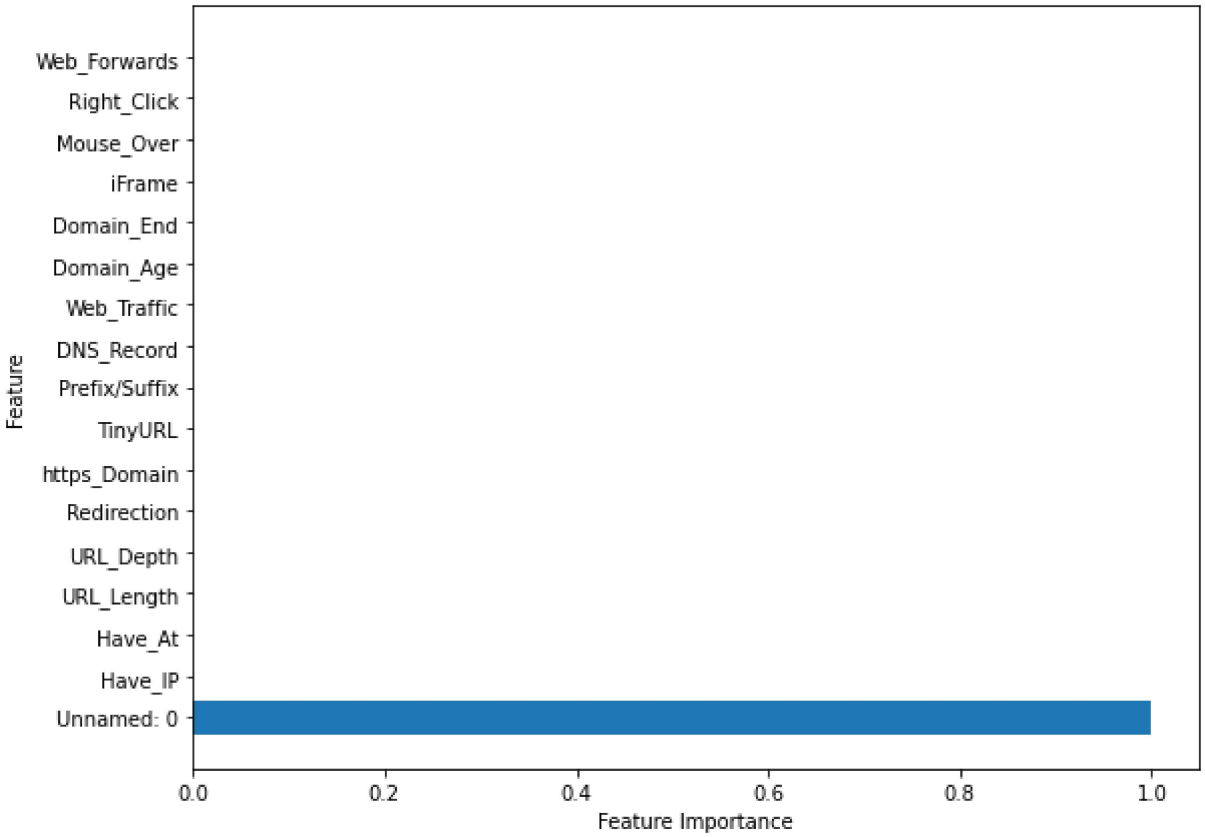
Out[31]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.25, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=7, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [32]:
```
y_test_xgb = xgb.predict(X_test)
y_train_xgb = xgb.predict(X_train)
```

In [33]:
```
#XGBoost
Train_xgb = accuracy_score(y_train,y_train_xgb)
Test_xgb = accuracy_score(y_test,y_test_xgb)
print("XGBoost: Accuracy on training Data: {:.3f}".format(Train_xgb))
print("XGBoost : Accuracy on test Data: {:.3f}".format(Test_xgb))
```

```
XGBoost: Accuracy on training Data: 1.000
XGBoost : Accuracy on test Data: 1.000
```

In [34]:
```
plt.figure(figsize=(9,7))
num_features = X_train.shape[1]
plt.barh(range(num_features), xgb.feature_importances_, align='center')
plt.yticks(np.arange(num_features), X_train.columns)
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.show()
```



In [35]:
```
storeResults('XGBoost', Train_xgb, Test_xgb)
```

AUTO ENCODER

In [36]:
```
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
```

In [37]:
```
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(10
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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 17)]              0
_________________________________________________________________
dense (Dense)                (None, 17)                306
```

```
dense_1 (Dense)              (None, 17)                306
_________________________________________________________________
dense_2 (Dense)              (None, 15)                270
_________________________________________________________________
dense_5 (Dense)              (None, 17)                272
_________________________________________________________________
dense_6 (Dense)              (None, 17)                306
=================================================================
Total params: 1,460
Trainable params: 1,460
Non-trainable params: 0
_________________________________________________________________
```

In [38]:
```
autoencoder.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history = autoencoder.fit(X_train, X_train, epochs=10, batch_size=64, shuffle=True, val
```

```
Epoch 1/10
100/100 [==============================] - 3s 11ms/step - loss: -4448.4448 - accuracy:
0.0000e+00 - val_loss: -4618.3364 - val_accuracy: 0.0000e+00
Epoch 2/10
100/100 [==============================] - 0s 4ms/step - loss: -4452.1377 - accuracy: 0.
0106 - val_loss: -4621.2021 - val_accuracy: 0.9975
Epoch 3/10
100/100 [==============================] - 0s 4ms/step - loss: -4454.1860 - accuracy: 0.
9998 - val_loss: -4622.5771 - val_accuracy: 0.9994
Epoch 4/10
100/100 [==============================] - 0s 4ms/step - loss: -4455.0972 - accuracy: 0.
9997 - val_loss: -4623.0371 - val_accuracy: 1.0000
Epoch 5/10
100/100 [==============================] - 0s 4ms/step - loss: -4454.5459 - accuracy: 0.
9998 - val_loss: -4623.0859 - val_accuracy: 1.0000
Epoch 6/10
100/100 [==============================] - 0s 4ms/step - loss: -4455.2446 - accuracy: 0.
9998 - val_loss: -4623.0728 - val_accuracy: 1.0000
Epoch 7/10
100/100 [==============================] - 0s 4ms/step - loss: -4455.2441 - accuracy: 0.
9998 - val_loss: -4623.0737 - val_accuracy: 1.0000
Epoch 8/10
100/100 [==============================] - 0s 4ms/step - loss: -4455.2456 - accuracy: 0.
9998 - val_loss: -4623.0737 - val_accuracy: 1.0000
Epoch 9/10
100/100 [==============================] - 0s 3ms/step - loss: -4455.2456 - accuracy: 0.
9998 - val_loss: -4623.0747 - val_accuracy: 1.0000
Epoch 10/10
100/100 [==============================] - 0s 3ms/step - loss: -4455.2461 - accuracy: 0.
9998 - val_loss: -4623.0752 - val_accuracy: 1.0000
```

In [39]:
```
Train_auto = autoencoder.evaluate(X_train, X_train)[1]
Test_auto = autoencoder.evaluate(X_test, X_test)[1]
print('\nAutoencoder: Accuracy on training Data: {:.3f}' .format(Train_auto))
print('Autoencoder: Accuracy on test Data: {:.3f}' .format(Test_auto))
```

```
250/250 [==============================] - 2s 4ms/step - loss: -4488.8120 - accuracy: 0.
9999
63/63 [==============================] - 1s 3ms/step - loss: -4457.3916 - accuracy: 1.00
00

Autoencoder: Accuracy on training Data: 1.000
Autoencoder: Accuracy on test Data: 1.000
```

In [40]:
```
storeResults('AutoEncoder', Train_auto, Test_auto)
```

SUPPORT VECTOR MACHINE

In [41]:
```
#SVC
from sklearn.svm import SVC
svm = SVC(kernel='linear', C=1.0, random_state=12)
svm.fit(X_train, y_train)
```

Out[41]:
```
SVC(kernel='linear', random_state=12)
```

In [42]:
```
y_test_svm = svm.predict(X_test)
y_train_svm = svm.predict(X_train)
```

In [43]:
```
#SVM Accuracy
Train_svm = accuracy_score(y_train,y_train_svm)
Test_svm = accuracy_score(y_test,y_test_svm)
print("SVM: Accuracy on training Data: {:.3f}".format(Train_svm))
print("SVM : Accuracy on test Data: {:.3f}".format(Test_svm))
```

```
SVM: Accuracy on training Data: 1.000
SVM : Accuracy on test Data: 1.000
```

In [44]:
```
storeResults('SVM', Train_svm, Test_svm)
```

FINAL RESULT

In [45]:
```
#Final Result
results = pd.DataFrame({ 'ML Model': MLModel,'Train Accuracy': Train,'Test Accuracy': T
results
```

Out[45]:

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	1.000	1.00
1	Random Forest	1.000	1.00
2	Multilayer Perceptrons	0.968	0.97
3	XGBoost	1.000	1.00
4	AutoEncoder	1.000	1.00
5	SVM	1.000	1.00

In [46]:
```
results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=False)
```

Out[46]:

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	1.000	1.00
1	Random Forest	1.000	1.00
3	XGBoost	1.000	1.00
4	AutoEncoder	1.000	1.00
5	SVM	1.000	1.00
2	Multilayer Perceptrons	0.968	0.97

In [47]:
```
import pickle
pickle.dump(xgb, open("XGBoostClassifier.pickle.dat", "wb"))
```

In [48]:
```
loaded_model = pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))
loaded_model
```

Out[48]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.25, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=7, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```