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

In [2]:
```
#Loading the data
df = pd.read_csv('Phising_Training_Dataset.csv')
df.head()
```

Out[2]:

ɔn_length	...	popUpWidnow	Iframe	age_of_domain	DNSRecord	web_traffic	Page_Rank	Google_Index	Links_pointing_to_page	Statistical_report	Result
-1	...	1	1	-1	-1	-1	-1	1	1	-1	-1
-1	...	1	1	-1	-1	0	-1	1	1	1	-1
-1	...	1	1	1	-1	1	-1	1	0	-1	-1
1	...	1	1	-1	-1	1	-1	1	-1	1	-1
-1	...	-1	1	-1	-1	0	-1	1	1	1	1

Data information

In [3]:
```
print("Shape of data: ",df.shape)
print("")
print("Columns present in the data: \n",df.columns)
print("")
print("Data informainformation: ")
df.info()
```

```
Shape of data:  (8955, 32)

Columns present in the data:
 Index(['key', 'having_IP', 'URL_Length', 'Shortining_Service',
       'having_At_Symbol', 'double_slash_redirecting', 'Prefix_Suffix',
```

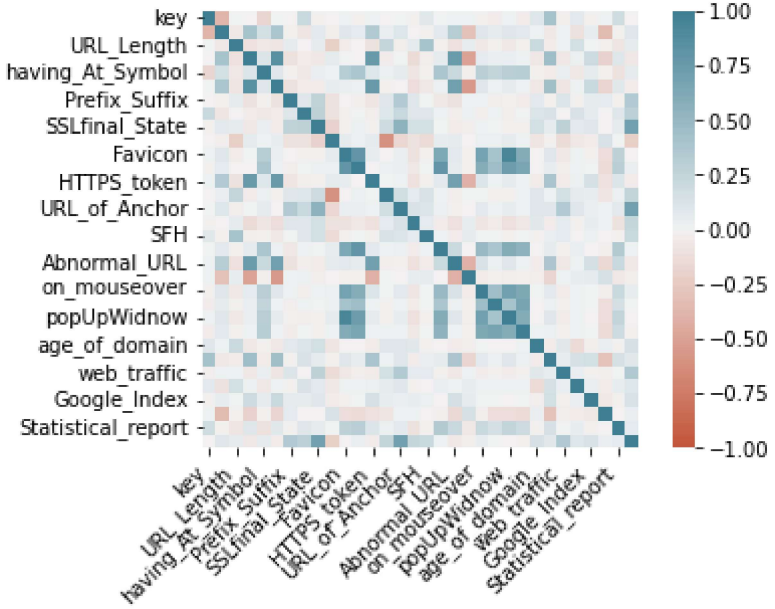
```
       'having_Sub_Domain', 'SSLfinal_State', 'Domain_registeration_length',
       'Favicon', 'port', 'HTTPS_token', 'Request_URL', 'URL_of_Anchor',
       'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL',
       'Redirect', 'on_mouseover', 'RightClick', 'popUpWidnow', 'Iframe',
       'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank',
       'Google_Index', 'Links_pointing_to_page', 'Statistical_report',
       'Result'],
      dtype='object')

Data informainformation:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8955 entries, 0 to 8954
Data columns (total 32 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   key                          8955 non-null   int64
 1   having_IP                    8955 non-null   int64
 2   URL_Length                   8955 non-null   int64
 3   Shortining_Service           8955 non-null   int64
 4   having_At_Symbol             8955 non-null   int64
 5   double_slash_redirecting     8955 non-null   int64
 6   Prefix_Suffix                8955 non-null   int64
 7   having_Sub_Domain            8955 non-null   int64
 8   SSLfinal_State               8955 non-null   int64
 9   Domain_registeration_length  8955 non-null   int64
 10  Favicon                      8955 non-null   int64
 11  port                         8955 non-null   int64
 12  HTTPS_token                  8955 non-null   int64
 13  Request_URL                  8955 non-null   int64
 14  URL_of_Anchor                8955 non-null   int64
 15  Links_in_tags                8955 non-null   int64
 16  SFH                          8955 non-null   int64
 17  Submitting_to_email          8955 non-null   int64
 18  Abnormal_URL                 8955 non-null   int64
 19  Redirect                     8955 non-null   int64
 20  on_mouseover                 8955 non-null   int64
 21  RightClick                   8955 non-null   int64
 22  popUpWidnow                  8955 non-null   int64
 23  Iframe                       8955 non-null   int64
 24  age_of_domain                8955 non-null   int64
 25  DNSRecord                    8955 non-null   int64
 26  web_traffic                  8955 non-null   int64
 27  Page_Rank                    8955 non-null   int64
 28  Google_Index                 8955 non-null   int64
```

```
 29  Links_pointing_to_page       8955 non-null   int64
 30  Statistical_report           8955 non-null   int64
 31  Result                       8955 non-null   int64
dtypes: int64(32)
memory usage: 2.2 MB
```

Visualizing the data

In [4]:

```
import seaborn as sns
corr = df.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



Preprocessing

In [5]:
```
df.describe ()
```

Out[5]:

	key	having_IP	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_
count	8955.000000	8955.000000	8955.000000	8955.000000	8955.000000	8955.000000	8955.000000	8955.000000	8955.0
mean	16821.000000	0.307203	-0.635734	0.740480	0.709436	0.740704	-0.735343	0.071803	0.2
std	2585.230164	0.951697	0.763660	0.672116	0.704809	0.671870	0.677733	0.817419	0.9
min	12344.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.0
25%	14582.500000	-1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000	-1.000000	-1.0
50%	16821.000000	1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000	0.000000	1.0
75%	19059.500000	1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000	1.000000	1.0
max	21298.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

8 rows × 32 columns

In [6]:
```
#We dont need the key column for modelling
df_train = df.drop("key", axis = 1)
df_train.head()
```

Out[6]:

	having_IP	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_State	Domain_regist
0	-1	1	1	1	-1	-1	-1	-1	
1	1	1	1	1	1	-1	0	1	
2	1	0	1	1	1	-1	-1	-1	
3	1	0	1	1	1	-1	-1	-1	
4	1	0	-1	1	1	-1	1	1	

5 rows × 31 columns

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
key                          0
having_IP                    0
URL_Length                   0
Shortining_Service           0
having_At_Symbol             0
double_slash_redirecting     0
Prefix_Suffix                0
having_Sub_Domain            0
SSLfinal_State               0
Domain_registeration_length  0
Favicon                      0
port                         0
HTTPS_token                  0
Request_URL                  0
URL_of_Anchor                0
Links_in_tags                0
SFH                          0
Submitting_to_email          0
Abnormal_URL                 0
Redirect                     0
on_mouseover                 0
RightClick                   0
popUpWidnow                  0
Iframe                       0
age_of_domain                0
DNSRecord                    0
web_traffic                  0
Page_Rank                    0
Google_Index                 0
Links_pointing_to_page       0
Statistical_report           0
Result                       0
dtype: int64
```

In [8]:
```
y = df['Result']
X = df.drop('Result',axis=1)
X.shape, y.shape
```

Out[8]: ((8955, 31), (8955,))

In [9]:
```
# 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 = 0.4, random_state = 12)
X_train.shape, X_test.shape
```

Out[9]: ((5373, 31), (3582, 31))

In [10]:
```
#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 Classifier

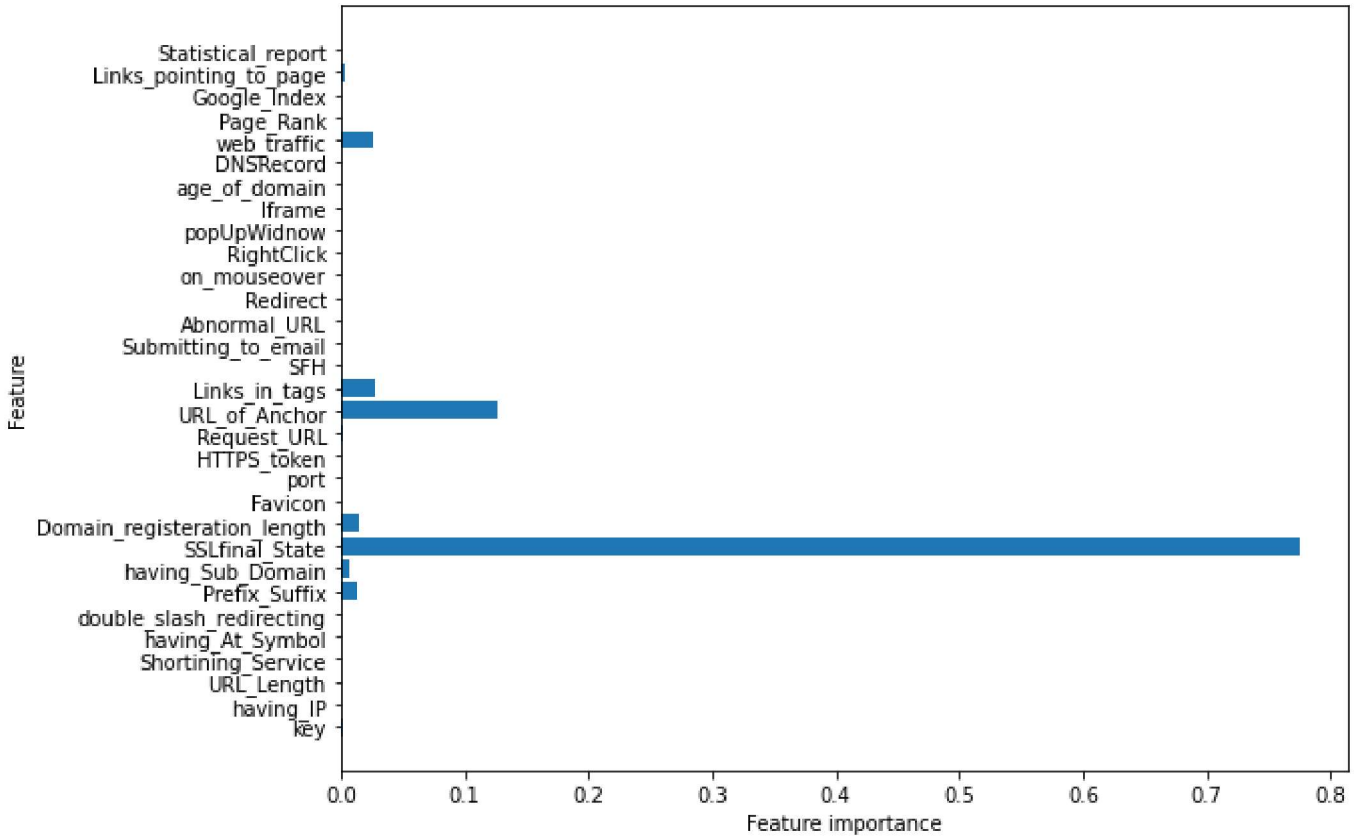
In [11]:
```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)
```

Evaluation

In [12]:
```
#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: 0.933
Decision Tree: Accuracy on test Data: 0.916
```

In [13]:
```
#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 [14]:
```
#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 Classifier

In [15]:
```
# 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)
```

Out[15]:
```
RandomForestClassifier(max_depth=5)
```

In [16]:
```
#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 [17]:
```
#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: 0.939
Random forest: Accuracy on test Data: 0.923
```

In [18]:
```
storeResults('Random Forest', acc_train_forest, acc_test_forest)
```

Multilayer Perceptrons (MLPs): Deep Learning

In [19]:
```
# 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[19]:
```
MLPClassifier(alpha=0.001, hidden_layer_sizes=[100, 100, 100])
```

In [20]:
```
#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: 0.435
Multilayer Perceptrons: Accuracy on test Data: 0.442
```

In [21]:

```
storeResults('Multilayer Perceptrons', acc_train_mlp, acc_test_mlp)
```

XGBoost Classifier

In [22]:

```
#Loading the data
df1 = pd.read_csv('Phising_Training_Dataset.csv')
df1["Result"].replace({-1: 0, 1: 1}, inplace=True)
df1.head()
```

Out[22]:

	key	having_IP	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_State	Domai
0	12344	-1	1	1	1	-1	-1	-1	-1	
1	12345	1	1	1	1	1	-1	0	1	
2	12346	1	0	1	1	1	-1	-1	-1	
3	12347	1	0	1	1	1	-1	-1	-1	
4	12348	1	0	-1	1	1	-1	1	1	

5 rows × 32 columns

In [23]:

```
y = df1['Result']
X = df1.drop('Result',axis=1)
X.shape, y.shape
```

Out[23]: ((8955, 31), (8955,))

In [24]:
```
# 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 = 0.4, random_state = 12)
X_train.shape, X_test.shape
```

Out[24]: ((5373, 31), (3582, 31))

In [25]:
```
#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)
#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)
storeResults('XGBoost', acc_train_xgb, acc_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: 1.000
XGBoost : Accuracy on test Data: 0.954
```

In [26]:
```
#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[26]: SVC(kernel='linear', random_state=12)

In [27]:
```
#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 [28]:
```
#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)
storeResults('SVM', acc_train_svm, acc_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: 0.924
SVM : Accuracy on test Data: 0.908
```

Results

In [29]:
```
#creating dataframe
results = pd.DataFrame({ 'ML Model': ML_Model,
    'Train Accuracy': acc_train,
    'Test Accuracy': acc_test})
#Sorting the datafram on accuracy
results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=False)
```

Out[29]:

	ML Model	Train Accuracy	Test Accuracy
3	XGBoost	1.000	0.954
1	Random Forest	0.939	0.923
0	Decision Tree	0.933	0.916
4	SVM	0.924	0.908
2	Multilayer Perceptrons	0.435	0.442

In [30]:
```
# save XGBoost model to file
import pickle
pickle.dump(xgb, open("XGBoostClassifier.pickle.dat", "wb"))
```

In [31]:
```
X_test = pd.read_csv("Phising_Testing_Dataset.csv")
```

In [32]:
```
# load model from file
model =  pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))
predicted_val = model.predict(X_test)
```

In [33]:
```
X_test["Results"] = predicted_val
X_test["Results"].replace({0: -1, 1: 1}, inplace=True)
X_test.head()
```

Out[33]:

	key	having_IP	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_State	Domai
0	21338	1	1	1	1	1	1	-1	1	
1	21339	1	-1	1	1	1	-1	0	-1	
2	21340	1	-1	1	1	1	-1	0	0	
3	21341	-1	-1	-1	1	-1	-1	-1	-1	
4	21342	1	-1	1	1	1	-1	1	1	

5 rows × 32 columns

In [ ]:
```
```