Final Project - Cyber Security for Book Website

Danny Mathieson

Contents

Exploratory Data Analysis

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

# set styling for plots
sns.set_style('whitegrid')

# Turn off interactive plotting for pyplot so we can choose to plot inline or not for
plt.ioff()
```

Out[1]:
```
<contextlib.ExitStack at 0x10aff5590>
```

In [2]:
```
# Create folders for plots & model objects
dirs = ['images',
    'images/feature_histograms',
    'images/Logistic_Regression',
    'images/Random_Forest',
    'images/KNN',
    'images/SVC',
    'model_objects'
]

for dir in dirs:
    if not os.path.exists(dir):
        os.makedirs(dir)
```

In [3]:
```
# Import data into pandas dataframe
# Assumes that you've stored the dataset in the same directory as this notebook
df = pd.read_csv('./datasets/dataset.csv', index_col='index')
```

```
print(df.shape)
```

```
(11055, 31)
```

In [4]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11055 entries, 1 to 11055
Data columns (total 31 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   having_IPhaving_IP_Address   11055 non-null  int64
 1   URLURL_Length                11055 non-null  int64
 2   Shortining_Service           11055 non-null  int64
 3   having_At_Symbol             11055 non-null  int64
 4   double_slash_redirecting     11055 non-null  int64
 5   Prefix_Suffix                11055 non-null  int64
 6   having_Sub_Domain            11055 non-null  int64
 7   SSLfinal_State               11055 non-null  int64
 8   Domain_registeration_length  11055 non-null  int64
 9   Favicon                      11055 non-null  int64
 10  port                         11055 non-null  int64
 11  HTTPS_token                  11055 non-null  int64
 12  Request_URL                  11055 non-null  int64
 13  URL_of_Anchor                11055 non-null  int64
 14  Links_in_tags                11055 non-null  int64
 15  SFH                          11055 non-null  int64
 16  Submitting_to_email          11055 non-null  int64
 17  Abnormal_URL                 11055 non-null  int64
 18  Redirect                     11055 non-null  int64
 19  on_mouseover                 11055 non-null  int64
 20  RightClick                   11055 non-null  int64
 21  popUpWidnow                  11055 non-null  int64
 22  Iframe                       11055 non-null  int64
 23  age_of_domain                11055 non-null  int64
 24  DNSRecord                    11055 non-null  int64
 25  web_traffic                  11055 non-null  int64
 26  Page_Rank                    11055 non-null  int64
 27  Google_Index                 11055 non-null  int64
 28  Links_pointing_to_page       11055 non-null  int64
 29  Statistical_report           11055 non-null  int64
 30  Result                       11055 non-null  int64
dtypes: int64(31)
memory usage: 2.7 MB
None
```

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

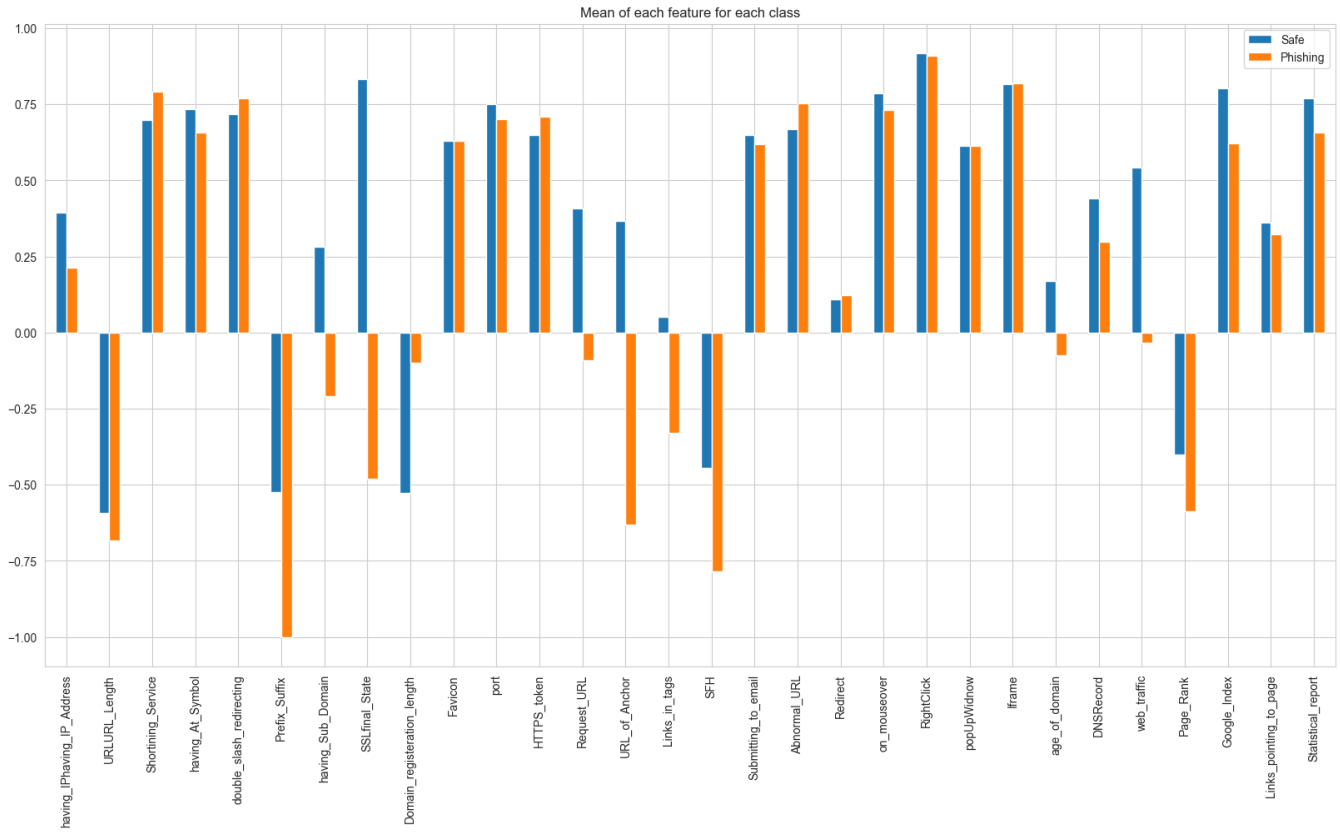
	count	mean	std	min	25%	50%	75%	max
having_IPhaving_IP_Address	11055.0	0.313795	0.949534	-1.0	-1.0	1.0	1.0	1.0
URLURL_Length	11055.0	-0.633198	0.766095	-1.0	-1.0	-1.0	-1.0	1.0
Shortining_Service	11055.0	0.738761	0.673998	-1.0	1.0	1.0	1.0	1.0
having_At_Symbol	11055.0	0.700588	0.713598	-1.0	1.0	1.0	1.0	1.0
double_slash_redirecting	11055.0	0.741474	0.671011	-1.0	1.0	1.0	1.0	1.0
Prefix_Suffix	11055.0	-0.734962	0.678139	-1.0	-1.0	-1.0	-1.0	1.0
having_Sub_Domain	11055.0	0.063953	0.817518	-1.0	-1.0	0.0	1.0	1.0
SSLfinal_State	11055.0	0.250927	0.911892	-1.0	-1.0	1.0	1.0	1.0
Domain_registeration_length	11055.0	-0.336771	0.941629	-1.0	-1.0	-1.0	1.0	1.0
Favicon	11055.0	0.628584	0.777777	-1.0	1.0	1.0	1.0	1.0
port	11055.0	0.728268	0.685324	-1.0	1.0	1.0	1.0	1.0
HTTPS_token	11055.0	0.675079	0.737779	-1.0	1.0	1.0	1.0	1.0
Request_URL	11055.0	0.186793	0.982444	-1.0	-1.0	1.0	1.0	1.0
URL_of_Anchor	11055.0	-0.076526	0.715138	-1.0	-1.0	0.0	0.0	1.0
Links_in_tags	11055.0	-0.118137	0.763973	-1.0	-1.0	0.0	0.0	1.0
SFH	11055.0	-0.595749	0.759143	-1.0	-1.0	-1.0	-1.0	1.0
Submitting_to_email	11055.0	0.635640	0.772021	-1.0	1.0	1.0	1.0	1.0
Abnormal_URL	11055.0	0.705292	0.708949	-1.0	1.0	1.0	1.0	1.0
Redirect	11055.0	0.115694	0.319872	0.0	0.0	0.0	0.0	1.0
on_mouseover	11055.0	0.762099	0.647490	-1.0	1.0	1.0	1.0	1.0
RightClick	11055.0	0.913885	0.405991	-1.0	1.0	1.0	1.0	1.0
popUpWidnow	11055.0	0.613388	0.789818	-1.0	1.0	1.0	1.0	1.0
Iframe	11055.0	0.816915	0.576784	-1.0	1.0	1.0	1.0	1.0
age_of_domain	11055.0	0.061239	0.998168	-1.0	-1.0	1.0	1.0	1.0
DNSRecord	11055.0	0.377114	0.926209	-1.0	-1.0	1.0	1.0	1.0
web_traffic	11055.0	0.287291	0.827733	-1.0	0.0	1.0	1.0	1.0
Page_Rank	11055.0	-0.483673	0.875289	-1.0	-1.0	-1.0	1.0	1.0
Google_Index	11055.0	0.721574	0.692369	-1.0	1.0	1.0	1.0	1.0
Links_pointing_to_page	11055.0	0.344007	0.569944	-1.0	0.0	0.0	1.0	1.0
Statistical_report	11055.0	0.719584	0.694437	-1.0	1.0	1.0	1.0	1.0
Result	11055.0	0.113885	0.993539	-1.0	-1.0	1.0	1.0	1.0

1. Each sample has 32 features ranging from -1,0,1. Explore the data using histogram, heatmaps.

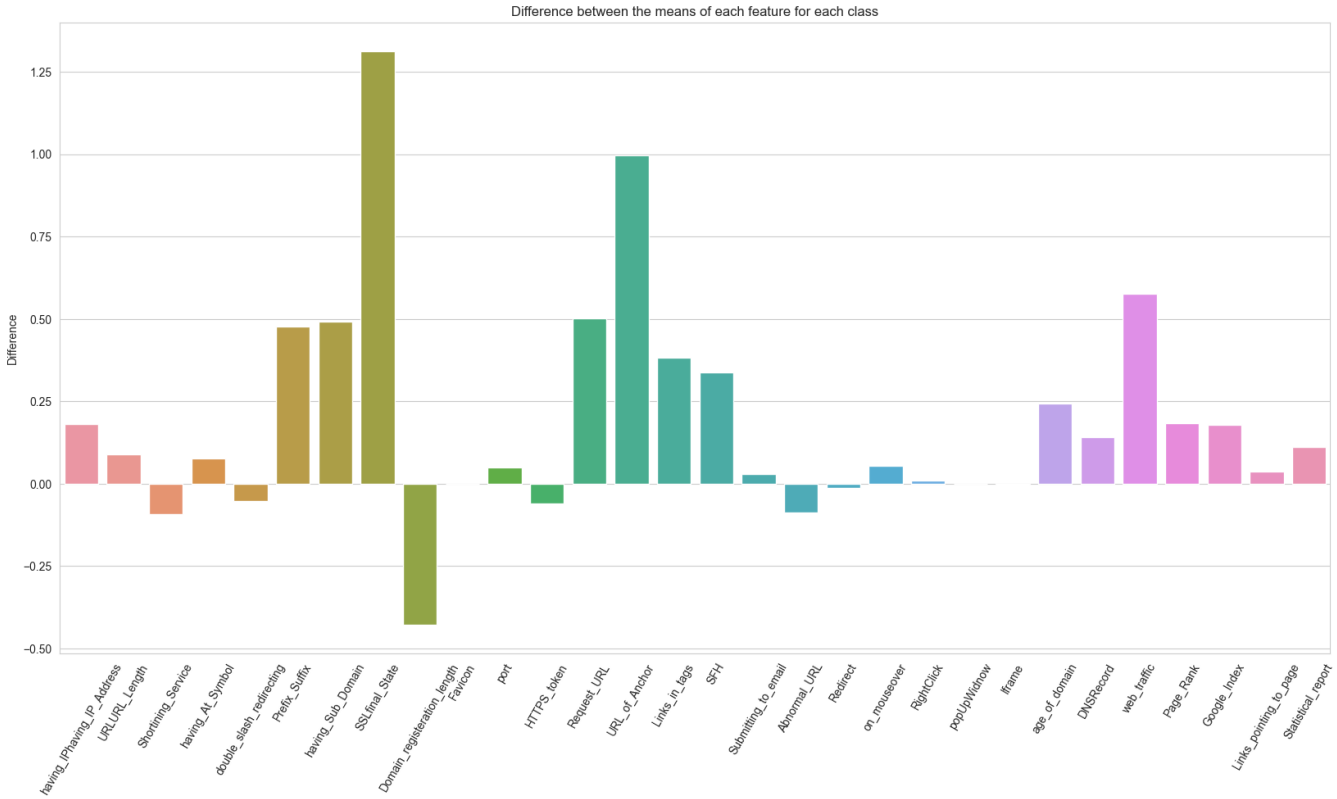
In [6]:
```
# Build a countplot for each feature with respect to the Result column
for col in df.columns:
    if col != 'Result':
        plt.figure()
        sns.countplot(x=col, hue='Result', data=df)
        plt.title('Distribution of {} by Result'.format(col))
        plt.savefig(f'./images/feature_histograms/{col}.png')
        # Don't show the plot, just save it
        plt.close()
    else:
        plt.figure()
        sns.countplot(x='Result', data=df)
        plt.title('Distribution of Result')
        plt.savefig('./images/feature_histograms/Result.png')
        # Don't show the plot, just save it
        plt.close()

# These plots are not the most useful to the outcome of the project and take up a lot
# I'll add the PNG files to the project submission for reference
```

In [7]:
```
# Build a barplot for the mean difference between the two classes for each column on
# create a new dataframe with the mean of each column for each class
df_means = pd.DataFrame(columns=['Safe', 'Phishing'])
for col in df.columns:
    if col != 'Result':
        df_means.loc[col] = [df[df['Result'] == 1][col].mean(), df[df['Result'] == -1

# Add a column for the difference between the two classes
df_means['Difference'] = df_means['Safe'] - df_means['Phishing']
```

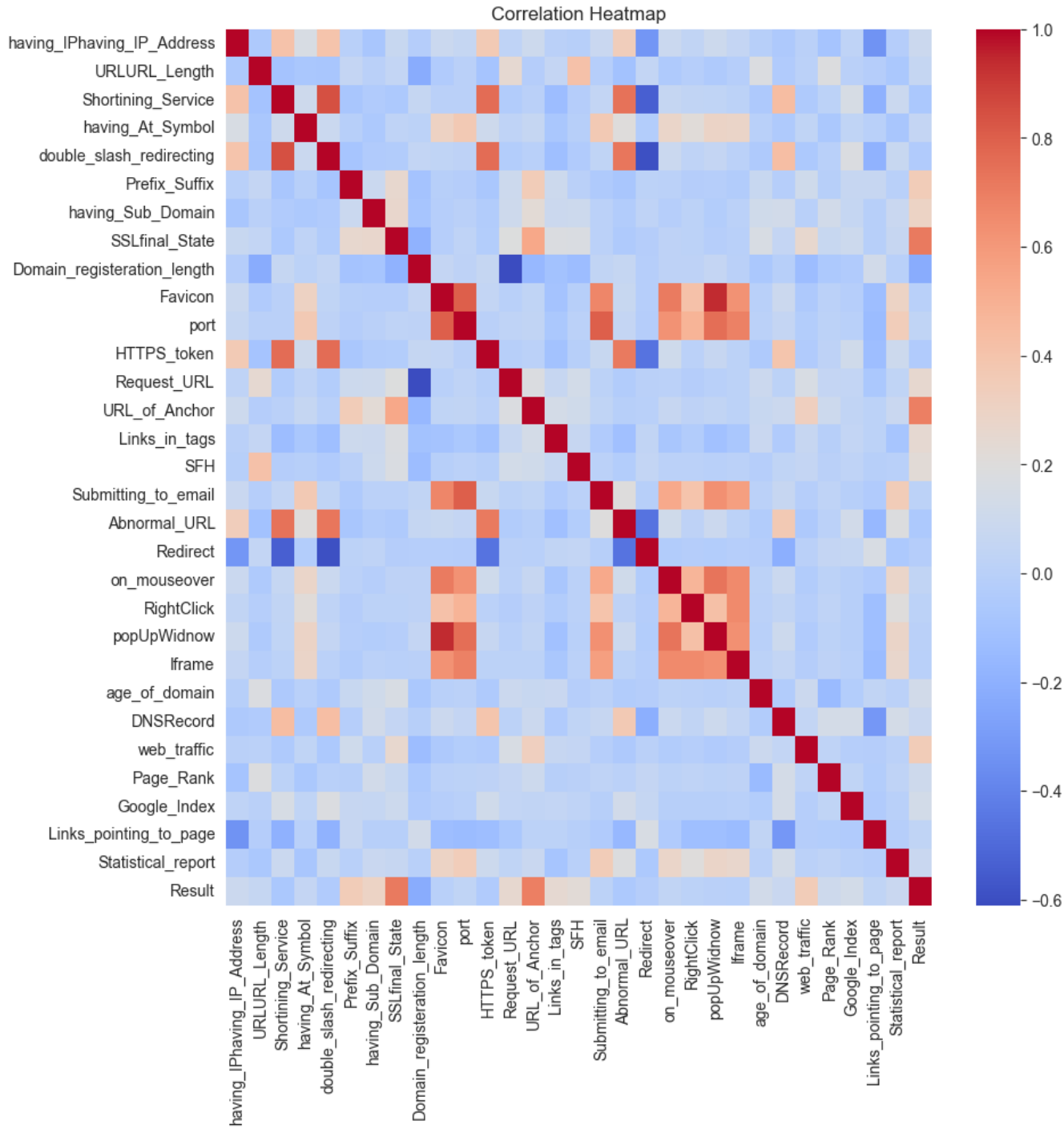
In [8]:
```
# plot the means (excluding the difference column)
df_means.drop('Difference', axis=1).plot(kind='bar', figsize=(20, 10))
plt.title('Mean of each feature for each class')
plt.savefig('./images/feature_mean.png')
plt.show()
```



In [9]:
```
# plot the difference between the two classes with seaborn
plt.figure(figsize=(20, 10))
sns.barplot(x=df_means.index, y=df_means['Difference'])
plt.title('Difference between the means of each feature for each class')
plt.xticks(rotation=60)
plt.savefig('./images/feature_mean_difference.png')
plt.show()
```



In [10]:
```
# Create a Heatmap of the correlation between all columns
# size the figure
plt.figure(figsize=(10, 10))
sns.heatmap(df.corr(), cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.savefig('./images/correlation_heatmap.png')
plt.show()
```



In [11]:
```
# Display the pairs of columns with higher than 0.5 correlation and lower than -0.5 c
print('Higher than 0.5 correlation:\n')
print(df.corr()[df.corr() > 0.5].unstack().drop_duplicates().sort_values(ascending=Fa
print('\n\nLower than -0.5 correlation:\n')
print(df.corr()[df.corr() < -0.5].unstack().drop_duplicates().sort_values(ascending=F
```

```
Higher than 0.5 correlation:

having_IPhaving_IP_Address  having_IPhaving_IP_Address    1.000000
Favicon                     popUpWidnow                   0.939633
Shortining_Service          double_slash_redirecting      0.842796
Favicon                     port                          0.803834
port                        Submitting_to_email           0.799088
double_slash_redirecting    HTTPS_token                   0.760799
Shortining_Service          HTTPS_token                   0.757838
port                        popUpWidnow                   0.748517
Shortining_Service          Abnormal_URL                  0.739290
on_mouseover                popUpWidnow                   0.733629
double_slash_redirecting    Abnormal_URL                  0.723724
HTTPS_token                 Abnormal_URL                  0.716287
SSLfinal_State              Result                        0.714741
Favicon                     on_mouseover                  0.706179
URL_of_Anchor               Result                        0.692935
port                        Iframe                        0.687044
Favicon                     Submitting_to_email           0.668317
on_mouseover                Iframe                        0.659478
RightClick                  Iframe                        0.655863
Submitting_to_email         popUpWidnow                   0.629462
popUpWidnow                 Iframe                        0.629406
Favicon                     Iframe                        0.627607
port                        on_mouseover                  0.623298
Submitting_to_email         Iframe                        0.577490
SSLfinal_State              URL_of_Anchor                 0.535786
Submitting_to_email         on_mouseover                  0.531656
having_IPhaving_IP_Address  URLURL_Length                       NaN
dtype: float64


Lower than -0.5 correlation:

Shortining_Service           Redirect                    -0.534530
double_slash_redirecting     Redirect                    -0.591478
Domain_registeration_length  Request_URL                 -0.609970
having_IPhaving_IP_Address   having_IPhaving_IP_Address        NaN
dtype: float64
```

In [12]:
```
# Repeat correlation exercise with just Safe and Phising classes
safe_df = df[df['Result'] == 1]
phishing_df = df[df['Result'] == -1]
```

In [13]:
```
# plot heatmap for safe df
plt.figure(figsize=(10, 10))
sns.heatmap(safe_df.corr(), cmap='coolwarm')
plt.title('Correlation Heatmap for Safe Sites')
plt.savefig('./images/correlation_heatmap_safe.png')
# Don't show the plot, just save it for space considerations
plt.close()

# plot heatmap for phishing df
plt.figure(figsize=(10, 10))
sns.heatmap(phishing_df.corr(), cmap='coolwarm')
plt.title('Correlation Heatmap for Phishing Sites')
plt.savefig('./images/correlation_heatmap_phishing.png')
# Don't show the plot, just save it for space considerations
plt.close()
```

In [14]:
```
# Display the pairs of columns with higher than 0.5 correlation and lower than -0.5 c
print('Higher than 0.5 correlation for Safe Sites:\n')
print(safe_df.corr()[safe_df.corr() > 0.5].unstack().drop_duplicates().sort_values(as

print('\n\nLower than -0.5 correlation for Safe Sites:\n')
```

```
print(safe_df.corr()[safe_df.corr() < -0.5].unstack().drop_duplicates().sort_values(a

print('\n\n\n\nHigher than 0.5 correlation for Phishing Sites:\n')
print(phishing_df.corr()[phishing_df.corr() > 0.5].unstack().drop_duplicates().sort_v

print('\n\nLower than -0.5 correlation for Phishing Sites:\n')
print(phishing_df.corr()[phishing_df.corr() < -0.5].unstack().drop_duplicates().sort_
```

```
Higher than 0.5 correlation for Safe Sites:

having_IPhaving_IP_Address  having_IPhaving_IP_Address    1.000000
Favicon                     popUpWidnow                   0.950593
Shortining_Service          double_slash_redirecting      0.872403
double_slash_redirecting    HTTPS_token                   0.803623
Shortining_Service          HTTPS_token                   0.797206
Favicon                     port                          0.790246
port                        Submitting_to_email           0.770645
double_slash_redirecting    Abnormal_URL                  0.770486
Shortining_Service          Abnormal_URL                  0.746589
port                        popUpWidnow                   0.741383
HTTPS_token                 Abnormal_URL                  0.730055
port                        Iframe                        0.702711
on_mouseover                popUpWidnow                   0.694733
Favicon                     on_mouseover                  0.679511
popUpWidnow                 Iframe                        0.651931
RightClick                  Iframe                        0.646044
Favicon                     Iframe                        0.644911
                            Submitting_to_email           0.636978
on_mouseover                Iframe                        0.626598
Submitting_to_email         popUpWidnow                   0.618690
port                        on_mouseover                  0.615609
Submitting_to_email         Iframe                        0.596429
                            on_mouseover                  0.525055
double_slash_redirecting    DNSRecord                     0.516863
Shortining_Service          DNSRecord                     0.508461
having_IPhaving_IP_Address  URLURL_Length                      NaN
dtype: float64


Lower than -0.5 correlation for Safe Sites:

HTTPS_token                  Redirect                     -0.540576
Abnormal_URL                 Redirect                     -0.541179
Shortining_Service           Redirect                     -0.602753
Domain_registeration_length  Request_URL                  -0.646375
double_slash_redirecting     Redirect                     -0.663717
having_IPhaving_IP_Address   having_IPhaving_IP_Address         NaN
dtype: float64



Higher than 0.5 correlation for Phishing Sites:

having_IPhaving_IP_Address  having_IPhaving_IP_Address    1.000000
Favicon                     popUpWidnow                   0.925852
port                        Submitting_to_email           0.831672
Favicon                     port                          0.822133
Shortining_Service          double_slash_redirecting      0.796417
on_mouseover                popUpWidnow                   0.781192
port                        popUpWidnow                   0.759082
Favicon                     on_mouseover                  0.739763
Shortining_Service          Abnormal_URL                  0.724551
Favicon                     Submitting_to_email           0.706990
on_mouseover                Iframe                        0.700921
double_slash_redirecting    HTTPS_token                   0.696619
Shortining_Service          HTTPS_token                   0.696134
HTTPS_token                 Abnormal_URL                  0.694355
port                        Iframe                        0.671307
RightClick                  Iframe                        0.668648
double_slash_redirecting    Abnormal_URL                  0.650142
Submitting_to_email         popUpWidnow                   0.642985
port                        on_mouseover                  0.630317
```

```
Favicon                       Iframe                        0.605655
popUpWidnow                   Iframe                        0.600846
Submitting_to_email           Iframe                        0.554777
                              on_mouseover                  0.539159
on_mouseover                  RightClick                    0.511714
having_IPhaving_IP_Address    URLURL_Length                      NaN
dtype: float64


Lower than -0.5 correlation for Phishing Sites:

double_slash_redirecting      Redirect                     -0.503235
Domain_registeration_length   Request_URL                  -0.528385
having_IPhaving_IP_Address    having_IPhaving_IP_Address         NaN
dtype: float64
```

2. Determine the number of samples present in the data, unique elements in all the features.

3. Check if there is any null value in any features.

In [15]:
```python
# Determine the number os samples in the dataset
print(f'Samples:\t{df.shape[0]}\n')

# Determine number of unique samples in the dataset
print(f'Unique Samples:\t{df.drop_duplicates().shape[0]}\n')

# Determine number of unique columns in the dataset
print(f'Unique Columns:\t{df.drop_duplicates().shape[1]}\n')

# Check for null values in any features
print(df.isnull().sum())
```

```
Samples:        11055

Unique Samples: 5849

Unique Columns: 31

having_IPhaving_IP_Address     0
URLURL_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
```

Correlation of Features and Feature Selection

Back to Top

In [16]:
```python
# Remove features with greater than 0.7 correlation
correlation_df = df.drop(['Result'], axis=1).corr()

# identify columns with >= 0.7 correlation
correlated_columns = set()
for i in range(len(correlation_df.columns)):
    for j in range(i):
        if abs(correlation_df.iloc[i, j]) >= 0.7:
            colname = correlation_df.columns[i]
            correlated_columns.add(colname)

print(f'Columns with >= 0.7 correlation:\n\n{correlated_columns}\n')
```

```
Columns with >= 0.7 correlation:

{'double_slash_redirecting', 'Abnormal_URL', 'on_mouseover', 'popUpWidnow', 'HTTPS_to
ken', 'port', 'Submitting_to_email'}
```

In [17]:
```python
# Drop columns with >= 0.7 correlation
df.drop(correlated_columns, axis=1, inplace=True)
```

Building Classification Model

Back to Top

1. Build classification models using a binary classifier to detect malicious or phishing URLs.

In [18]:
```
# Build a binary classification model to predict whether a website is safe or phishin
# import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
import pickle
```

In [19]:
```
# split data into features and target
X = df.drop('Result', axis=1)
y = df['Result']

# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
```

In [20]:
```
# print train and test shapes
print(f'X Train Shape:\t{X_train.shape}')
print(f'X Test Shape:\t{X_test.shape}')
print(f'y Train Shape:\t{y_train.shape}')
print(f'y Test Shape:\t{y_test.shape}')
```

```
X Train Shape:  (8844, 23)
X Test Shape:   (2211, 23)
y Train Shape:  (8844,)
y Test Shape:   (2211,)
```

In [21]:
```
# scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# save the scaler to disk for future use
pickle.dump(scaler, open('./model_objects/scaler.pkl', 'wb'))
```

In [22]:
```
# Fitting the Logistic Regression Model
log_reg_model = LogisticRegression()
log_reg_model.fit(X_train, y_train)

# save the model to disk for future use
pickle.dump(log_reg_model, open('./model_objects/log_reg_model.pkl', 'wb'))
```

In [23]:
```
# Fitting the SVC Model
svc_model = SVC()
svc_model.fit(X_train, y_train)

# save the model to disk for future use
pickle.dump(svc_model, open('./model_objects/svc_model.pkl', 'wb'))
```

In [24]:
```
# Fitting the KNN Model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
```

```
# save the model to disk for future use
pickle.dump(knn_model, open('./model_objects/knn_model.pkl', 'wb'))
```

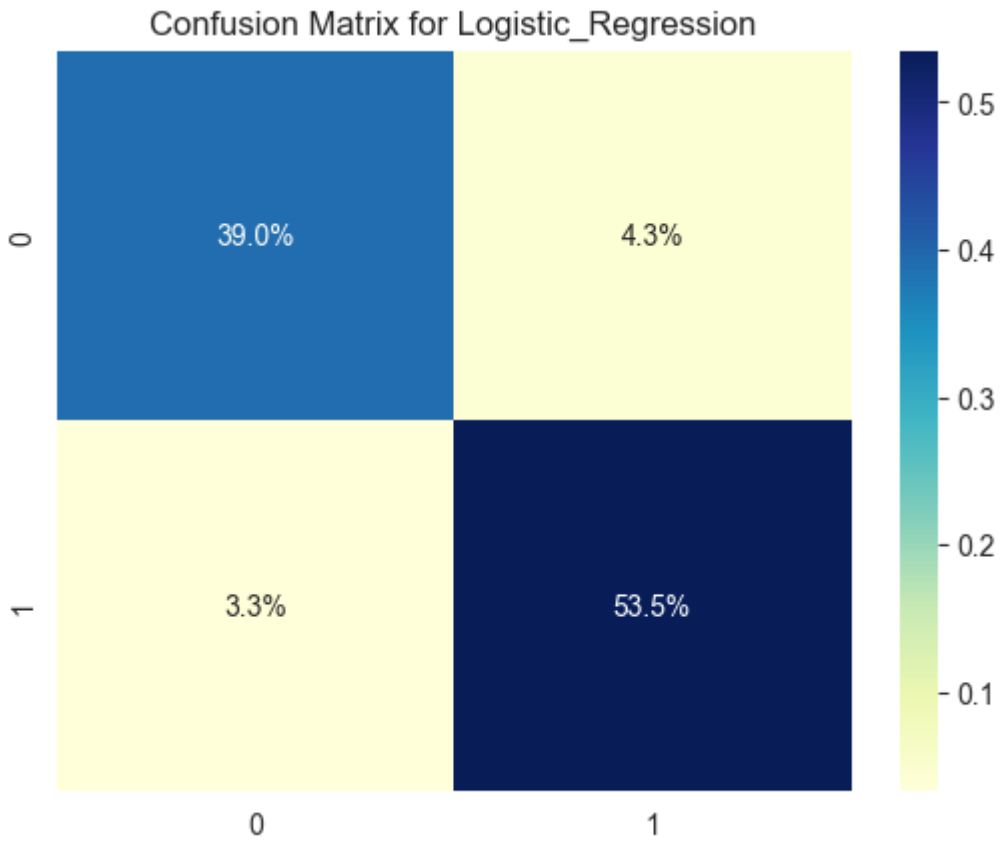
In [25]:
```
# Fitting the Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)

# save the model to disk for future use
pickle.dump(rf_model, open('./model_objects/rf_model.pkl', 'wb'))
```

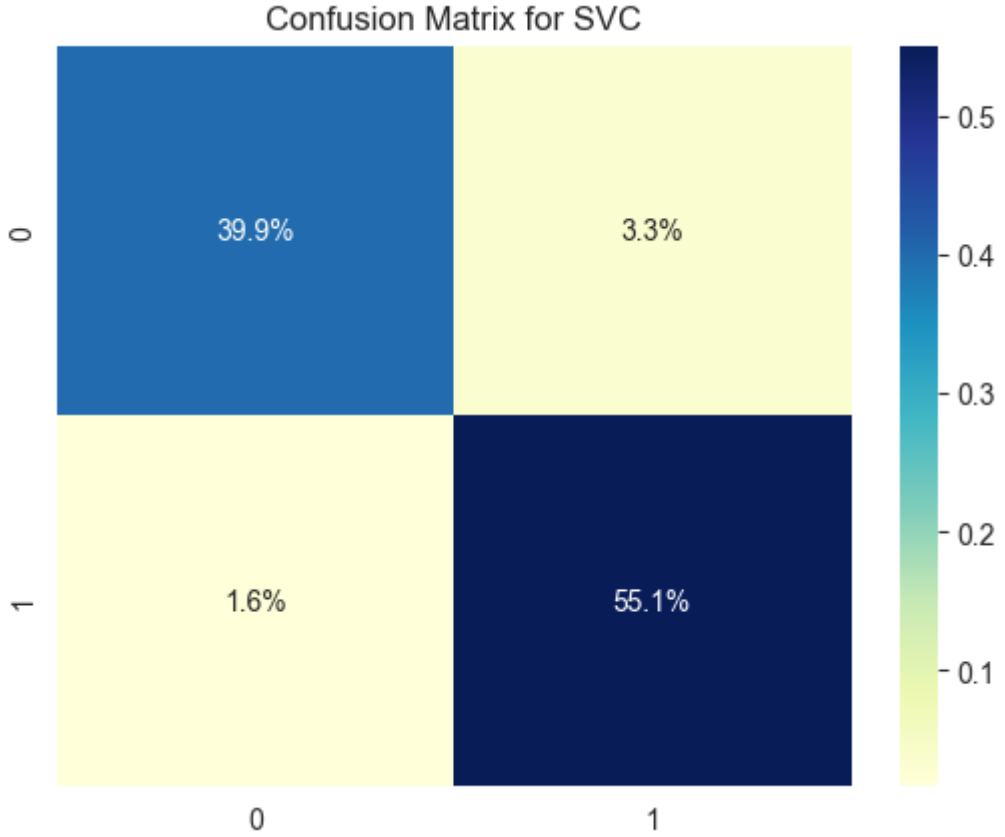
In [26]:
```
# make predictions on the test set
log_reg_y_pred = log_reg_model.predict(X_test)
svc_y_pred = svc_model.predict(X_test)
knn_y_pred = knn_model.predict(X_test)
rf_y_pred = rf_model.predict(X_test)
```

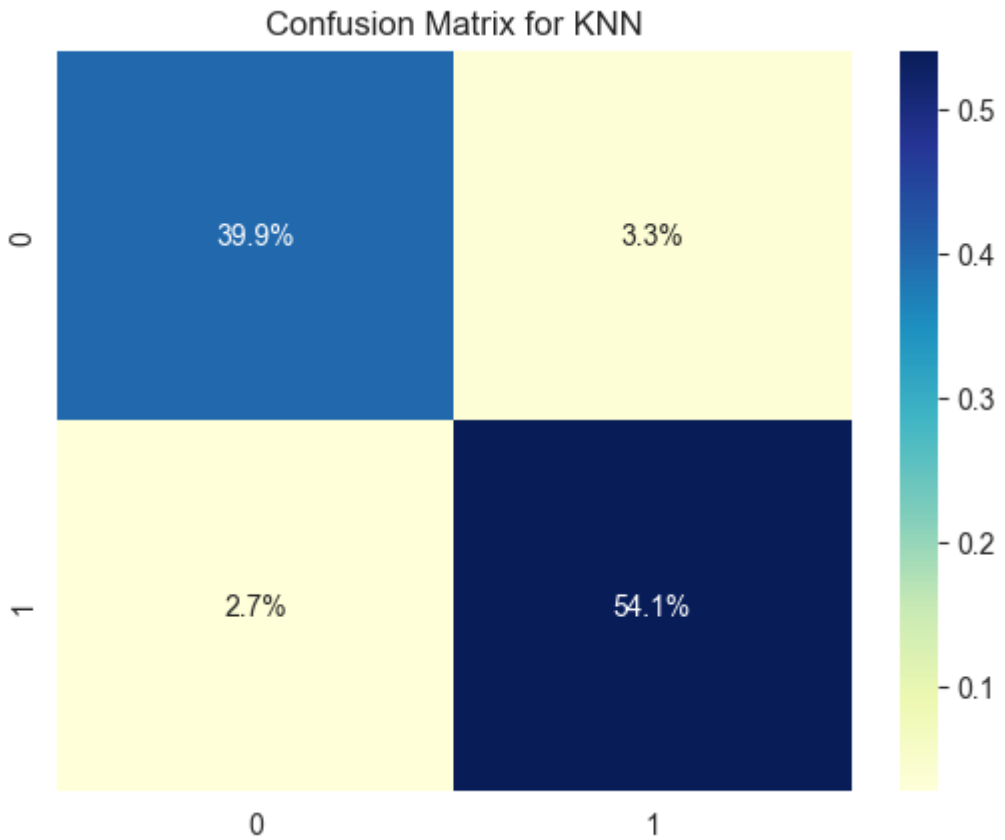
In [27]:
```
# Create arrays for the models and their predictions
models = [log_reg_model, svc_model, knn_model, rf_model]
predictions = [log_reg_y_pred, svc_y_pred, knn_y_pred, rf_y_pred]
names = ['Logistic_Regression', 'SVC', 'KNN', 'Random_Forest']
```

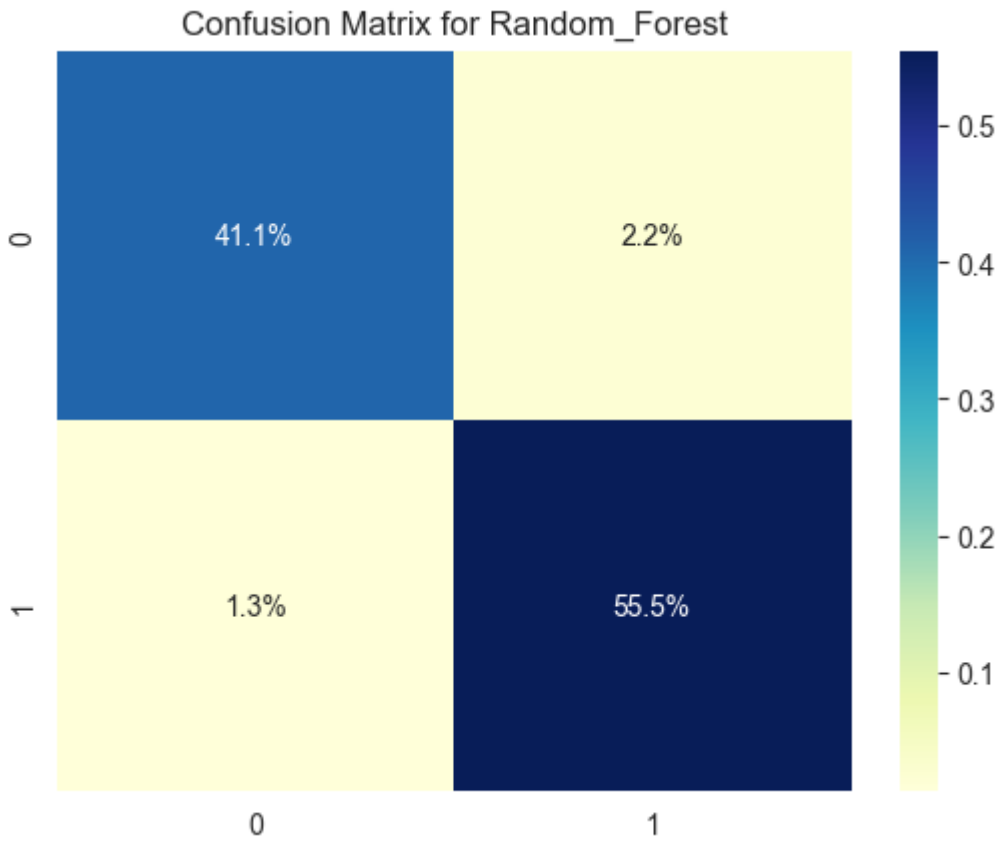
2. Illustrate the diagnostic ability of this binary classifier by plotting the ROC curve.

In [28]:
```
# plot the confusion matrix for each model
for prediction, name in zip(predictions, names):
    plt.figure()
    cnf_mat = confusion_matrix(y_test, prediction)
    sns.heatmap(cnf_mat/cnf_mat.sum(), annot=True, fmt='.1%', cmap='YlGnBu')
    plt.title(f'Confusion Matrix for {name}')
    plt.savefig(f'./images/{name}/confusion_matrix.png')
    plt.show()
```




Confusion Matrix for SVC
39.9%
3.3%
1.6%
55.1%
0
1
0
1
0.5
0.4
0.3
0.2
0.1


Confusion Matrix for KNN
39.9%
3.3%
2.7%
54.1%
0
1
0
1
0.5
0.4
0.3
0.2
0.1



In [29]:

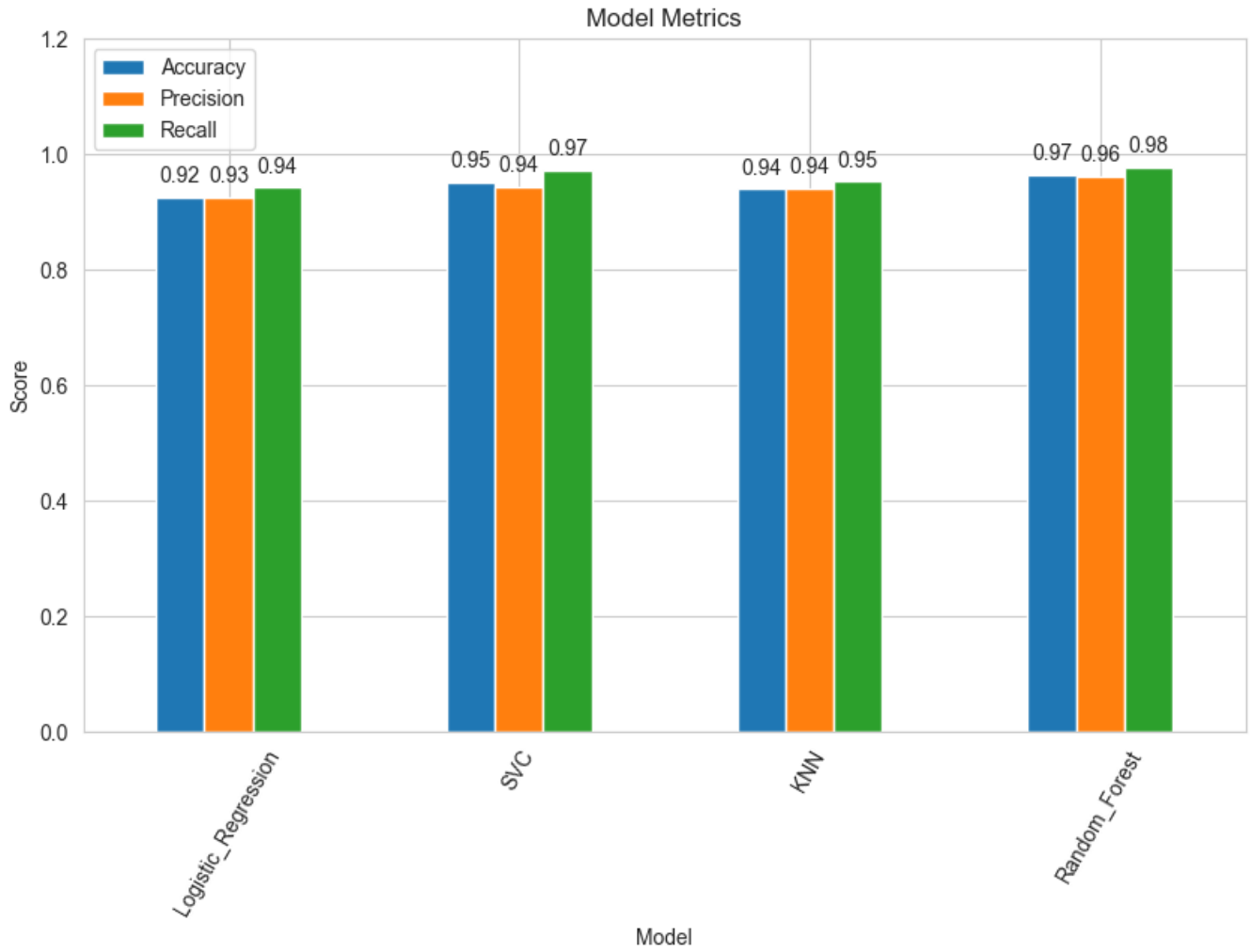
```python
# print the classification report for each model
for model, prediction, name in zip(models, predictions, names):
    print(f'Classification Report for {name}:\n')
    # print the classification report
    print(classification_report(y_test, prediction))
    # write the classification report to a text file
    with open(f'./images/{name}/classification_report.txt', 'w') as f:
        f.write(classification_report(y_test, prediction))
    print('\n\n')
```

```
Classification Report for Logistic_Regression:

              precision    recall  f1-score   support

          -1       0.92      0.90      0.91       956
           1       0.93      0.94      0.93      1255

    accuracy                           0.92      2211
   macro avg       0.92      0.92      0.92      2211
weighted avg       0.92      0.92      0.92      2211



Classification Report for SVC:

              precision    recall  f1-score   support

          -1       0.96      0.92      0.94       956
           1       0.94      0.97      0.96      1255

    accuracy                           0.95      2211
   macro avg       0.95      0.95      0.95      2211
weighted avg       0.95      0.95      0.95      2211



Classification Report for KNN:

              precision    recall  f1-score   support

          -1       0.94      0.92      0.93       956
           1       0.94      0.95      0.95      1255

    accuracy                           0.94      2211
   macro avg       0.94      0.94      0.94      2211
weighted avg       0.94      0.94      0.94      2211



Classification Report for Random_Forest:

              precision    recall  f1-score   support

          -1       0.97      0.95      0.96       956
           1       0.96      0.98      0.97      1255

    accuracy                           0.97      2211
   macro avg       0.97      0.96      0.96      2211
weighted avg       0.97      0.97      0.97      2211
```

In [30]:

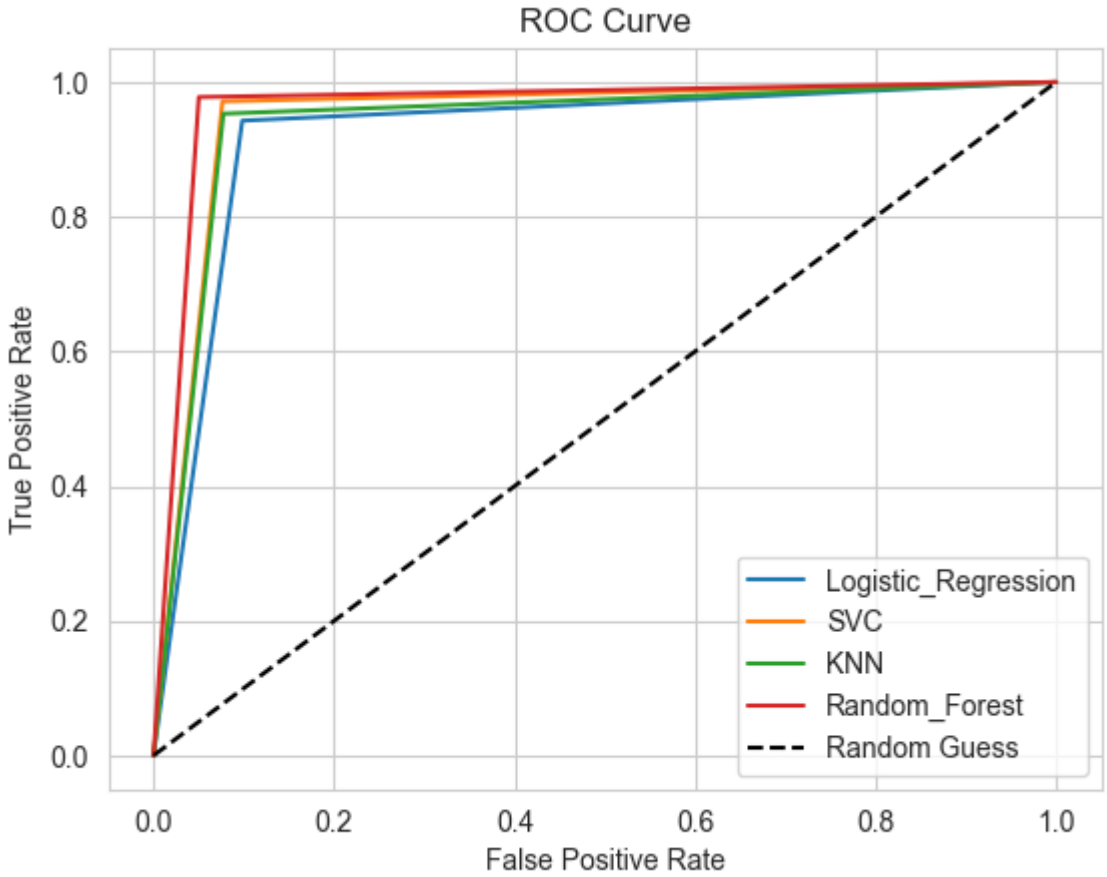
```python
# Make a dataframe of the accuracy, precision, and recall for each model
model_metrics = pd.DataFrame(columns=['Accuracy', 'Precision', 'Recall'], index=names
model_metrics.index.name = 'Model'
for model, prediction, name in zip(models, predictions, names):
    model_metrics.loc[name] = [accuracy_score(y_test, prediction),
                               precision_score(y_test, prediction),
                               recall_score(y_test, prediction)
                               ]
```

```
# plot the model metrics
model_metrics.plot(kind='bar', figsize=(10, 6))

# set y max to 1.2 to make the plot look nicer
plt.ylim(0, 1.2)

# add values to the bars
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.ge
                       ha='center', va='center', xytext=(0, 10), textcoords='offset p

plt.title('Model Metrics')
plt.xlabel('Model')
plt.ylabel('Score')
plt.xticks(rotation=60)
plt.legend(loc='upper left')
plt.savefig('./images/model_metrics.png')
plt.show()
```



In [31]:

```
# plot the ROC curve for each model
for model, prediction, name in zip(models, predictions, names):
    fpr, tpr, thresholds = roc_curve(y_test, prediction)
    plt.plot(fpr, tpr, label=f'{name}')

# plot the diagonal line for random guess
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.savefig('./images/roc_curve.png')
plt.show()
```



3. Validate the accuracy of data by the K-Fold cross-validation technique.

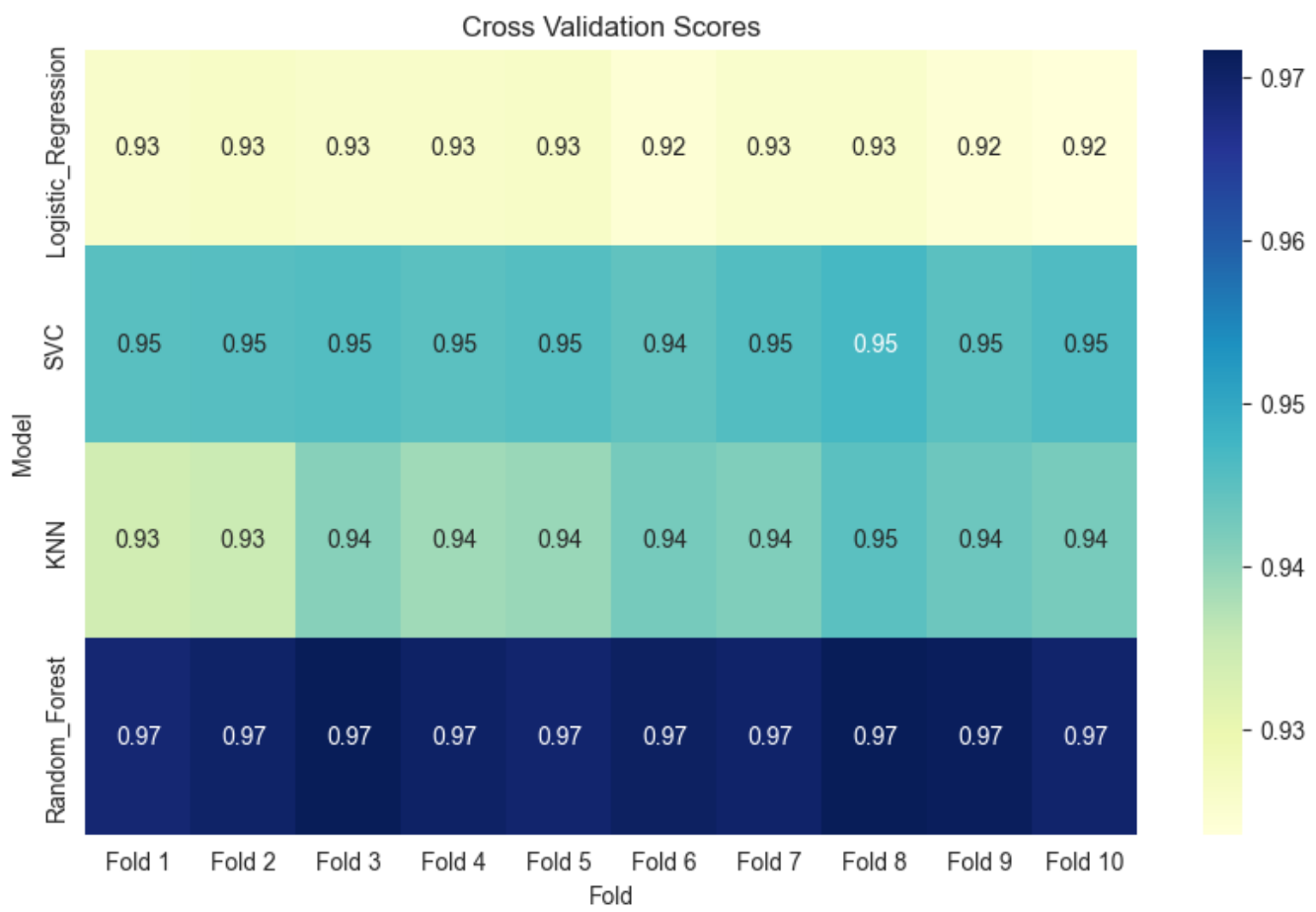
In [32]:
```
# Initialize K-Fold cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits=10)
```

In [33]:
```
# Build a dataframe to store the cross validation scores for each model across 10 fol
cv_scores = pd.DataFrame(columns=[f'Fold {i+1}' for i in range(10)], index=names)

# Remember we have the non-split data in X and y
for i, (train_index, test_index) in enumerate(kf.split(X)):
    # Loop over each model to get the cross validation score for each model
    for model, name in zip(models, names):
        # Get the cross validation score
        score = cross_val_score(model, X.iloc[train_index], y.iloc[train_index], cv=1
        # Add the score to the dataframe
        cv_scores.loc[name, f'Fold {i+1}'] = score
        # Print the score
        print(f'Fold {i+1} {name} Cross Validation Score: {score:.4f}')

print(cv_scores)
```

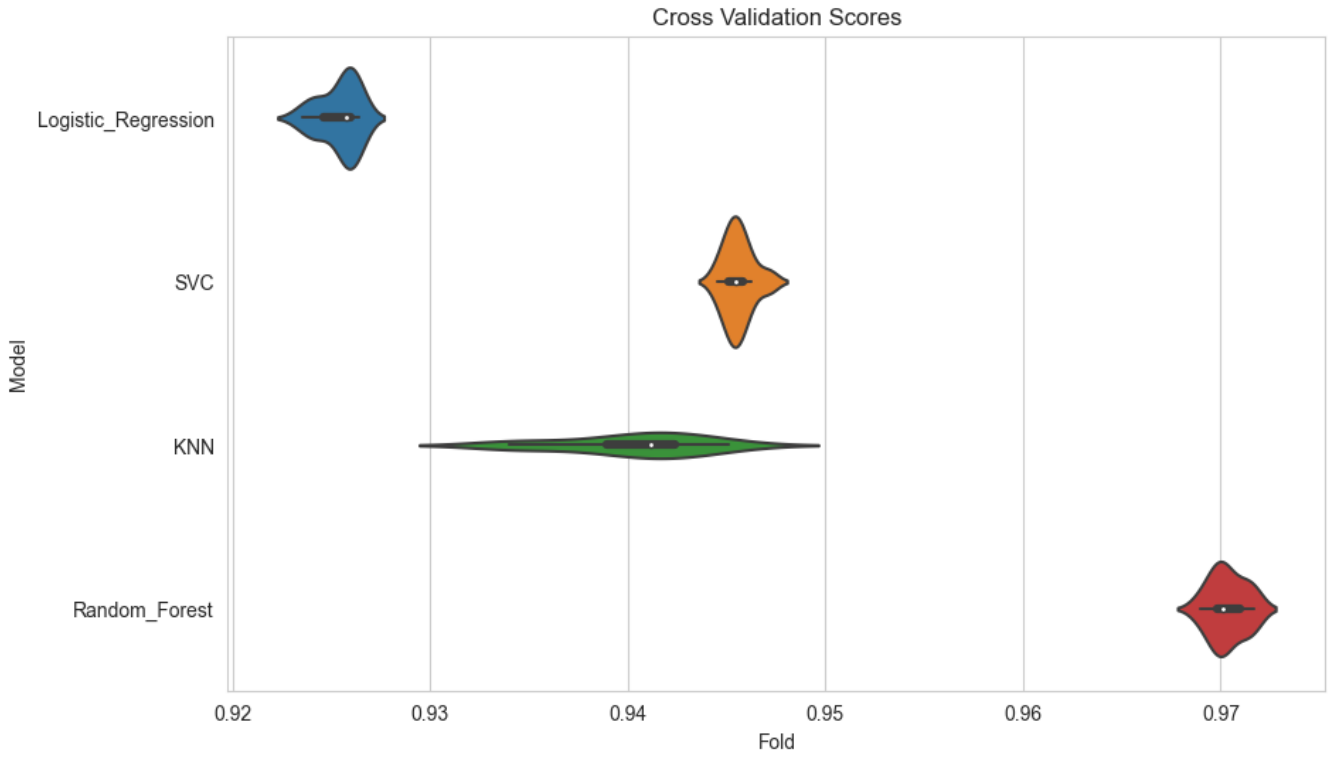
```
Fold 1 Logistic_Regression Cross Validation Score: 0.9259
Fold 1 SVC Cross Validation Score: 0.9453
Fold 1 KNN Cross Validation Score: 0.9340
Fold 1 Random_Forest Cross Validation Score: 0.9689
Fold 2 Logistic_Regression Cross Validation Score: 0.9264
Fold 2 SVC Cross Validation Score: 0.9454
Fold 2 KNN Cross Validation Score: 0.9349
Fold 2 Random_Forest Cross Validation Score: 0.9700
Fold 3 Logistic_Regression Cross Validation Score: 0.9258
Fold 3 SVC Cross Validation Score: 0.9458
Fold 3 KNN Cross Validation Score: 0.9409
Fold 3 Random_Forest Cross Validation Score: 0.9717
Fold 4 Logistic_Regression Cross Validation Score: 0.9260
Fold 4 SVC Cross Validation Score: 0.9450
Fold 4 KNN Cross Validation Score: 0.9388
Fold 4 Random_Forest Cross Validation Score: 0.9702
Fold 5 Logistic_Regression Cross Validation Score: 0.9262
Fold 5 SVC Cross Validation Score: 0.9455
Fold 5 KNN Cross Validation Score: 0.9395
Fold 5 Random_Forest Cross Validation Score: 0.9695
Fold 6 Logistic_Regression Cross Validation Score: 0.9243
Fold 6 SVC Cross Validation Score: 0.9445
Fold 6 KNN Cross Validation Score: 0.9424
Fold 6 Random_Forest Cross Validation Score: 0.9706
Fold 7 Logistic_Regression Cross Validation Score: 0.9255
Fold 7 SVC Cross Validation Score: 0.9457
Fold 7 KNN Cross Validation Score: 0.9414
Fold 7 Random_Forest Cross Validation Score: 0.9699
Fold 8 Logistic_Regression Cross Validation Score: 0.9257
Fold 8 SVC Cross Validation Score: 0.9471
Fold 8 KNN Cross Validation Score: 0.9451
Fold 8 Random_Forest Cross Validation Score: 0.9716
Fold 9 Logistic_Regression Cross Validation Score: 0.9243
Fold 9 SVC Cross Validation Score: 0.9450
Fold 9 KNN Cross Validation Score: 0.9433
Fold 9 Random_Forest Cross Validation Score: 0.9712
Fold 10 Logistic_Regression Cross Validation Score: 0.9235
Fold 10 SVC Cross Validation Score: 0.9462
Fold 10 KNN Cross Validation Score: 0.9423
Fold 10 Random_Forest Cross Validation Score: 0.9697
                        Fold 1    Fold 2    Fold 3    Fold 4    Fold 5  \
Logistic_Regression  0.925923  0.926425  0.925822  0.926024  0.926224
SVC                  0.945321  0.945422  0.945824   0.94502  0.945522
KNN                  0.933964  0.934869  0.940899  0.938788  0.939492
Random_Forest        0.968941  0.970047  0.971655  0.970248  0.969544

                        Fold 6    Fold 7    Fold 8    Fold 9   Fold 10
Logistic_Regression  0.924322  0.925528  0.925729  0.924322  0.923518
SVC                  0.944523  0.945729  0.947136  0.945025  0.946231
KNN                  0.942412  0.941407  0.945126  0.943317  0.942312
Random_Forest        0.970553   0.96995  0.971558  0.971156  0.969749
```

In [34]:

```python
# Plot the cross validation scores for each model with a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(cv_scores.astype(float), annot=True, cmap='YlGnBu')
plt.title('Cross Validation Scores')
plt.xlabel('Fold')
plt.ylabel('Model')
plt.savefig('./images/cross_validation_scores_heatmap.png')
plt.show()
```



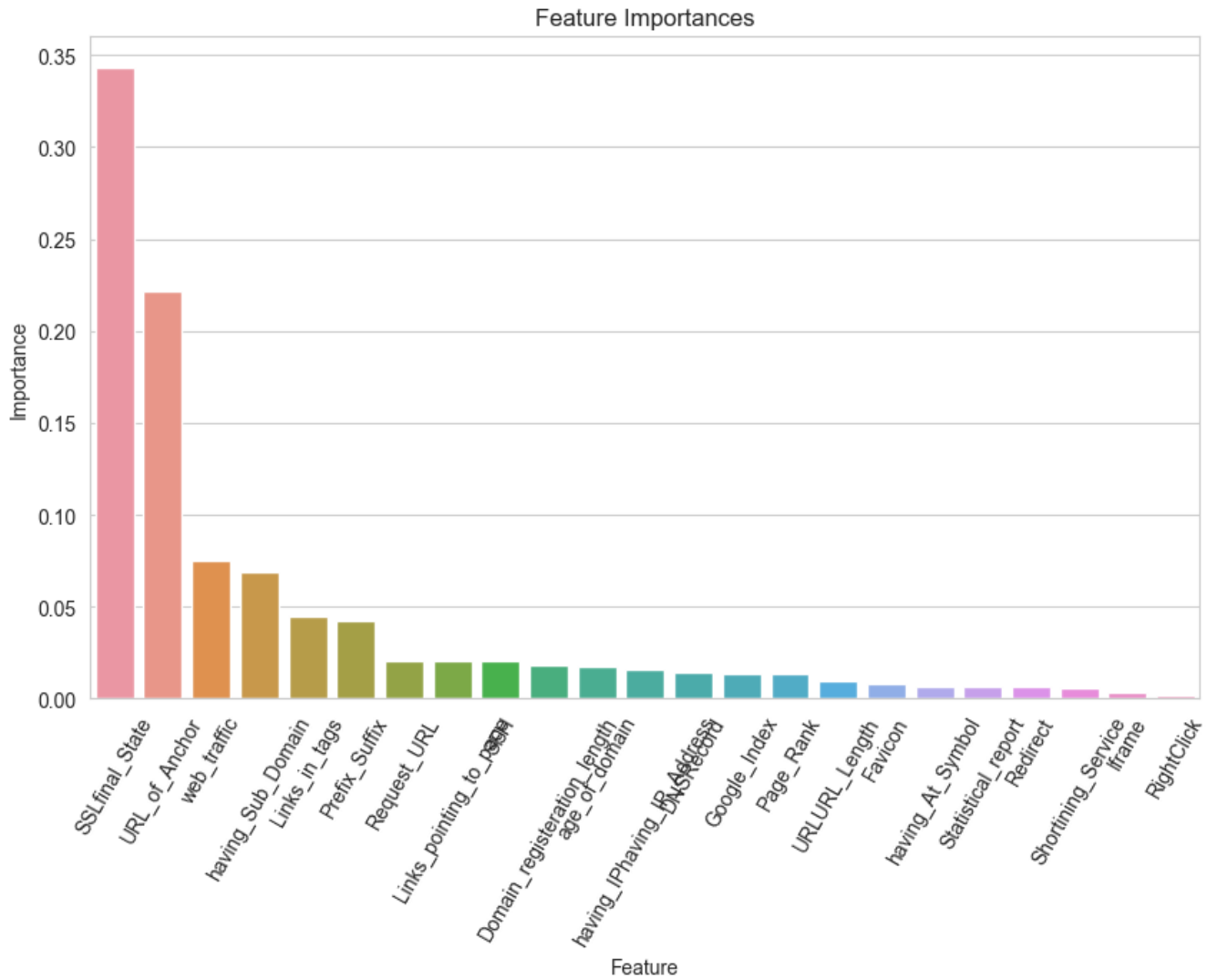
In [35]:
```
# Plot the cross validation scores for each model with a violin plot
plt.figure(figsize=(10, 6))
sns.violinplot(data=cv_scores.transpose().astype(float), orient='h')
plt.title('Cross Validation Scores')
plt.xlabel('Fold')
plt.ylabel('Model')
plt.savefig('./images/cross_validation_scores_violin.png')
plt.show()
```



4. The final output consists of the model, which will give maximum accuracy on the validation dataset with selected attributes.

In [36]:
```
# Random Forest is the best model
best_model = rf_model
```

In [37]:
```
# Let's visualize the feature importances
# Get the feature importances
feature_importances = pd.DataFrame(best_model.feature_importances_, index=X.columns,
# Sort the feature importances
feature_importances = feature_importances.sort_values('Importance', ascending=False)
# Plot the feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances.index, y=feature_importances['Importance'])
plt.xticks(rotation=60)
plt.title('Feature Importances')
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.savefig('./images/feature_importances.png')
plt.show()
```



In [38]:
```
# Visualize the decision tree
from sklearn.tree import export_graphviz
import pydot
from IPython.display import Image
```

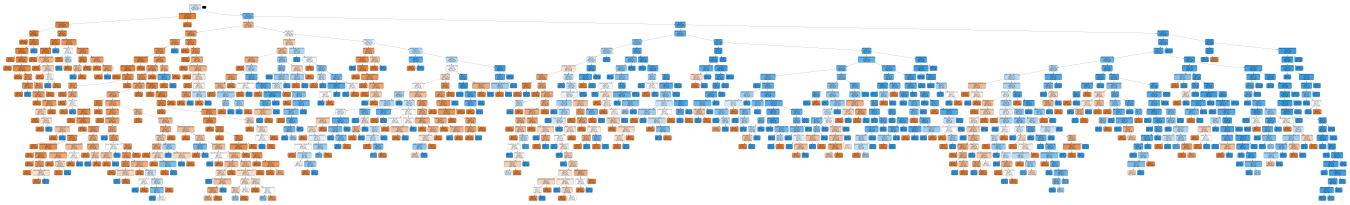
In [39]:
```
# Get the first tree from the random forest
tree = best_model.estimators_[0]

# Export the image to a dot file
export_graphviz(tree,
                out_file='./images/Random_Forest/decision_tree.dot',
                feature_names=X.columns,
                class_names=['Phishing', 'Safe'],
                rounded=True,
```

```
                precision=1,
                filled=True
            )
```

The following cell will not work unless you've installed GraphViz on your machine and added it to your path

In [40]:
```
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('./images/Random_Forest/decision_tree.dot')

# Write graph to a png file (vertical orientation)
graph.write_png('./images/Random_Forest/decision_tree.png')

# Visualize the decision tree
Image(filename='./images/Random_Forest/decision_tree.png')
```

Out[40]:



Back to Top