Phishing URL Detection

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,recall_score,precision_score,confusion_matrix,classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import pickle
pd.set_option('display.max_columns', None)
```

Import Dataset

```
# Import Dataset
df=pd.read_csv('dataset_phishing.csv')
df.head()
```

	url	length_url	length_hostname	ip	nb_dots	nb_hyphens	nb_at
0	http://www.crestonwood.com/router.php	37	19	0	3	0	0
1	http://shadetreetechnology.com/V4/validation/a...	77	23	1	1	0	0
2	https://support-appleId.com.secureupdate.duila...	126	50	1	4	1	0
3	http://rgipt.ac.in	18	11	0	2	0	0
4	http://www.iracing.com/tracks/gateway-motorspo...	55	15	0	2	2	0

```
df.isna().sum()
```

	0
url	0
length_url	0
length_hostname	0
ip	0
nb_dots	0
...	...
web_traffic	0
dns_record	0
google_index	0
page_rank	0
status	0

89 rows × 1 columns

dtype: int64

```
# removing missing values

df.dropna(inplace=True)


features = [
    'length_url', 'length_hostname', 'ip', 'nb_dots', 'nb_hyphens', 'nb_at', 'nb_qm', 'nb_and', 'nb_or', 'nb_
    'nb_underscore', 'nb_tilde', 'nb_percent', 'nb_slash', 'nb_star', 'nb_colon', 'nb_comma', 'nb_semicolumn'
    'nb_dollar', 'nb_space', 'nb_www', 'nb_com', 'nb_dslash', 'http_in_path', 'https_token', 'ratio_digits_ur
    'ratio_digits_host', 'punycode', 'shortening_service', 'path_extension', 'phish_hints', 'domain_in_brand'
    'brand_in_subdomain', 'brand_in_path', 'suspecious_tld'
]


# target feature mapping

df['status'] = df['status'].map({'phishing': 1, 'legitimate': 0})


df['status'].value_counts()
```

status	count
0	5715
1	5715

dtype: int64

```
df.describe()
```

	length_url	length_hostname	ip	nb_dots	nb_hyphens	nb_at	nb_qm
count	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000
mean	61.126684	21.090289	0.150569	2.480752	0.997550	0.022222	0.141207
std	55.297318	10.777171	0.357644	1.369686	2.087087	0.155500	0.364456
min	12.000000	4.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	33.000000	15.000000	0.000000	2.000000	0.000000	0.000000	0.000000
50%	47.000000	19.000000	0.000000	2.000000	0.000000	0.000000	0.000000
75%	71.000000	24.000000	0.000000	3.000000	1.000000	0.000000	0.000000
max	1641.000000	214.000000	1.000000	24.000000	43.000000	4.000000	3.000000

```
df.shape
```

```
(11430, 89)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11430 entries, 0 to 11429
Data columns (total 89 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   url                         11430 non-null  object
 1   length_url                  11430 non-null  int64
 2   length_hostname             11430 non-null  int64
 3   ip                          11430 non-null  int64
 4   nb_dots                     11430 non-null  int64
 5   nb_hyphens                  11430 non-null  int64
```

```
 6   nb_at                       11430 non-null  int64
 7   nb_qm                       11430 non-null  int64
 8   nb_and                      11430 non-null  int64
 9   nb_or                       11430 non-null  int64
 10  nb_eq                       11430 non-null  int64
 11  nb_underscore               11430 non-null  int64
 12  nb_tilde                    11430 non-null  int64
 13  nb_percent                  11430 non-null  int64
 14  nb_slash                    11430 non-null  int64
 15  nb_star                     11430 non-null  int64
 16  nb_colon                    11430 non-null  int64
 17  nb_comma                    11430 non-null  int64
 18  nb_semicolumn               11430 non-null  int64
 19  nb_dollar                   11430 non-null  int64
 20  nb_space                    11430 non-null  int64
 21  nb_www                      11430 non-null  int64
 22  nb_com                      11430 non-null  int64
 23  nb_dslash                   11430 non-null  int64
 24  http_in_path                11430 non-null  int64
 25  https_token                 11430 non-null  int64
 26  ratio_digits_url            11430 non-null  float64
 27  ratio_digits_host           11430 non-null  float64
 28  punycode                    11430 non-null  int64
 29  port                        11430 non-null  int64
 30  tld_in_path                 11430 non-null  int64
 31  tld_in_subdomain            11430 non-null  int64
 32  abnormal_subdomain          11430 non-null  int64
 33  nb_subdomains               11430 non-null  int64
 34  prefix_suffix               11430 non-null  int64
 35  random_domain               11430 non-null  int64
 36  shortening_service          11430 non-null  int64
 37  path_extension              11430 non-null  int64
 38  nb_redirection              11430 non-null  int64
 39  nb_external_redirection     11430 non-null  int64
 40  length_words_raw            11430 non-null  int64
 41  char_repeat                 11430 non-null  int64
 42  shortest_words_raw          11430 non-null  int64
 43  shortest_word_host          11430 non-null  int64
 44  shortest_word_path          11430 non-null  int64
 45  longest_words_raw           11430 non-null  int64
 46  longest_word_host           11430 non-null  int64
 47  longest_word_path           11430 non-null  int64
 48  avg_words_raw               11430 non-null  float64
 49  avg_word_host               11430 non-null  float64
 50  avg_word_path               11430 non-null  float64
 51  phish_hints                 11430 non-null  int64
 52  domain_in_brand             11430 non-null  int64
```

```
# Select only the numerical columns from the dataframe
numerical_df = df.select_dtypes(include=['float64', 'int64'])

# Compute the correlation matrix on the numerical columns
corr_matrix = numerical_df.corr()


status_corr = corr_matrix['status']
status_corr.shape
```

```
(88,)
```

```
# Function for selecting features that are above than threshold value

def feature_selector_correlation(cmatrix, threshold):

    selected_features = []
    feature_score = []
    i=0
    for score in cmatrix:
        if abs(score)>threshold:
            selected_features.append(cmatrix.index[i])
            feature_score.append( ['{:3f}'.format(score)])
```

```
        i+=1
    result = list(zip(selected_features,feature_score))
    return result


features_selected = feature_selector_correlation(status_corr, 0.2)
features_selected
```

```
[('length_url', ['0.248580']),
 ('length_hostname', ['0.238322']),
 ('ip', ['0.321698']),
 ('nb_dots', ['0.207029']),
 ('nb_qm', ['0.294319']),
 ('nb_eq', ['0.233386']),
 ('nb_slash', ['0.242270']),
 ('nb_www', ['-0.443468']),
 ('ratio_digits_url', ['0.356395']),
 ('ratio_digits_host', ['0.224335']),
 ('tld_in_subdomain', ['0.208884']),
 ('prefix_suffix', ['0.214681']),
 ('shortest_word_host', ['0.223084']),
 ('longest_words_raw', ['0.200147']),
 ('longest_word_path', ['0.212709']),
 ('phish_hints', ['0.335393']),
 ('nb_hyperlinks', ['-0.342628']),
 ('ratio_intHyperlinks', ['-0.243982']),
 ('empty_title', ['0.207043']),
 ('domain_in_title', ['0.342807']),
 ('domain_age', ['-0.331889']),
 ('google_index', ['0.731171']),
 ('page_rank', ['-0.511137']),
 ('status', ['1.000000'])]
```

```
selected_features = []
for feature, score in features_selected:
    if feature != 'status':
        selected_features.append(feature)


selected_features
```

```
['length_url',
 'length_hostname',
 'ip',
 'nb_dots',
 'nb_qm',
 'nb_eq',
 'nb_slash',
 'nb_www',
 'ratio_digits_url',
 'ratio_digits_host',
 'tld_in_subdomain',
 'prefix_suffix',
 'shortest_word_host',
 'longest_words_raw',
 'longest_word_path',
 'phish_hints',
 'nb_hyperlinks',
 'ratio_intHyperlinks',
 'empty_title',
 'domain_in_title',
 'domain_age',
 'google_index',
 'page_rank']
```

Train Test Split and Feature Scaling

```
X = df[selected_features]
y = df['status']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Training

```
classifiers = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'SVM': SVC(),
    'KNN': KNeighborsClassifier()
}


param_grids = {
    'Logistic Regression': {
        'C': [0.1, 1, 10]
    },
    'Random Forest': {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20]
    },
    'Gradient Boosting': {
        'n_estimators': [100, 200],
        'learning_rate': [0.01, 0.1, 1]
    },
    'SVM': {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf']
    },
    'KNN': {
        'n_neighbors': [3, 5, 7,9],
        'p': [1, 2]
    }
}


results = {}
for name, clf in classifiers.items():
    grid_search = GridSearchCV(estimator=clf, param_grid=param_grids[name], cv=5, n_jobs=-1, scoring='accurac
    grid_search.fit(X_train_scaled, y_train)
    results[name] = grid_search


for name, grid_search in results.items():
    print(f"{name}:")
    print("Best Parameters:", grid_search.best_params_)
    print("Best Score:", grid_search.best_score_)
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test_scaled)
    test_accuracy = accuracy_score(y_test, y_pred)
    print("Test Accuracy:", test_accuracy)
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print()
```

```
Logistic Regression:
Best Parameters: {'C': 10}
Best Score: 0.9321055550074672
Test Accuracy: 0.9384184744576627
Confusion Matrix:
 [[1332   90]
```

```
 [  86 1350]]
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      1422
           1       0.94      0.94      0.94      1436

    accuracy                           0.94      2858
   macro avg       0.94      0.94      0.94      2858
weighted avg       0.94      0.94      0.94      2858


Random Forest:
Best Parameters: {'max_depth': 20, 'n_estimators': 200}
Best Score: 0.9610363632033909
Test Accuracy: 0.9611616515045487
Confusion Matrix:
 [[1366   56]
 [  55 1381]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      1422
           1       0.96      0.96      0.96      1436

    accuracy                           0.96      2858
   macro avg       0.96      0.96      0.96      2858
weighted avg       0.96      0.96      0.96      2858


Gradient Boosting:
Best Parameters: {'learning_rate': 0.1, 'n_estimators': 200}
Best Score: 0.9589367615691051
Test Accuracy: 0.9587123862841148
Confusion Matrix:
 [[1361   61]
 [  57 1379]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      1422
           1       0.96      0.96      0.96      1436

    accuracy                           0.96      2858
   macro avg       0.96      0.96      0.96      2858
weighted avg       0.96      0.96      0.96      2858


SVM:
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Best Score: 0.9564866253219074
Test Accuracy: 0.9622113365990203
```

```
print("Summary of Best Models:")
for name, grid_search in results.items():
    print(f"{name}:")
    print("Best Parameters:", grid_search.best_params_)
    print("Best Score (CV):", grid_search.best_score_)
    print()
```

```
Summary of Best Models:
Logistic Regression:
Best Parameters: {'C': 10}
Best Score (CV): 0.9321055550074672

Random Forest:
Best Parameters: {'max_depth': 20, 'n_estimators': 200}
Best Score (CV): 0.9610363632033909

Gradient Boosting:
Best Parameters: {'learning_rate': 0.1, 'n_estimators': 200}
Best Score (CV): 0.9589367615691051
```

```
SVM:
Best Parameters: {'C': 10, 'kernel': 'rbf'}
Best Score (CV): 0.9564866253219074

KNN:
Best Parameters: {'n_neighbors': 3, 'p': 1}
Best Score (CV): 0.9510034665641551
```

```
model=RandomForestClassifier(max_depth=20,n_estimators=100)
model.fit(X_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=20)
```

```
# saving model for deployment

with open('phishing_model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)

with open('scaler.pkl', 'wb') as scaler_file:
    pickle.dump(scaler, scaler_file)
```