In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_columns', None)
plt.rcParams['figure.figsize'] = (12,6)
```

In [4]:
```
data = pd.read_csv("E:\data set\Phishing_Legitimate_full.csv")
```

In [3]:
```
float_cols = data.select_dtypes('float64').columns
for c in float_cols:
    data[c] = data[c].astype('float32')

int_cols = data.select_dtypes('int64').columns
for c in int_cols:
    data[c] = data[c].astype('int32')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   id                                  10000 non-null  int32  
 1   NumDots                             10000 non-null  int32  
 2   SubdomainLevel                      10000 non-null  int32  
 3   PathLevel                           10000 non-null  int32  
 4   UrlLength                           10000 non-null  int32  
 5   NumDash                             10000 non-null  int32  
 6   NumDashInHostname                   10000 non-null  int32  
 7   AtSymbol                            10000 non-null  int32  
 8   TildeSymbol                         10000 non-null  int32  
 9   NumUnderscore                       10000 non-null  int32  
 10  NumPercent                          10000 non-null  int32  
 11  NumQueryComponents                  10000 non-null  int32  
 12  NumAmpersand                        10000 non-null  int32  
 13  NumHash                             10000 non-null  int32  
 14  NumNumericChars                     10000 non-null  int32  
 15  NoHttps                             10000 non-null  int32  
 16  RandomString                        10000 non-null  int32  
 17  IpAddress                           10000 non-null  int32  
 18  DomainInSubdomains                  10000 non-null  int32  
 19  DomainInPaths                       10000 non-null  int32  
 20  HttpsInHostname                     10000 non-null  int32  
 21  HostnameLength                      10000 non-null  int32  
 22  PathLength                          10000 non-null  int32  
 23  QueryLength                         10000 non-null  int32  
 24  DoubleSlashInPath                   10000 non-null  int32  
 25  NumSensitiveWords                   10000 non-null  int32  
 26  EmbeddedBrandName                   10000 non-null  int32  
 27  PctExtHyperlinks                    10000 non-null  float32
 28  PctExtResourceUrls                  10000 non-null  float32
 29  ExtFavicon                          10000 non-null  int32  
 30  InsecureForms                       10000 non-null  int32  
 31  RelativeFormAction                  10000 non-null  int32  
 32  ExtFormAction                       10000 non-null  int32  
 33  AbnormalFormAction                  10000 non-null  int32  
 34  PctNullSelfRedirectHyperlinks       10000 non-null  float32
 35  FrequentDomainNameMismatch          10000 non-null  int32  
 36  FakeLinkInStatusBar                 10000 non-null  int32  
 37  RightClickDisabled                  10000 non-null  int32  
 38  PopUpWindow                         10000 non-null  int32  
 39  SubmitInfoToEmail                   10000 non-null  int32  
 40  IframeOrFrame                       10000 non-null  int32  
 41  MissingTitle                        10000 non-null  int32  
 42  ImagesOnlyInForm                    10000 non-null  int32  
 43  SubdomainLevelRT                    10000 non-null  int32  
 44  UrlLengthRT                         10000 non-null  int32  
 45  PctExtResourceUrlsRT                10000 non-null  int32  
 46  AbnormalExtFormActionR              10000 non-null  int32  
 47  ExtMetaScriptLinkRT                 10000 non-null  int32  
 48  PctExtNullSelfRedirectHyperlinksRT  10000 non-null  int32  
 49  CLASS_LABEL                         10000 non-null  int32  
dtypes: float32(3), int32(47)
memory usage: 1.9 MB
```

In [7]:
```
data.rename(columns={'CLASS_LABEL': 'labels'}, inplace=True)
```

In [8]:
```
data.sample(5)
```

Out[8]:

	id	NumDots	SubdomainLevel	PathLevel	UrlLength	NumDash	NumDashInHostname	AtSymbol	TildeSymbol	NumUnderscore	Nur
6824	6825	3	1	1	103	10	3	0	0	0	
9438	9439	2	1	3	45	0	0	0	0	0	
2826	2827	3	0	5	62	1	0	0	0	0	
2580	2581	4	1	1	42	1	1	0	0	0	
5354	5355	2	1	5	134	6	0	0	0	2	

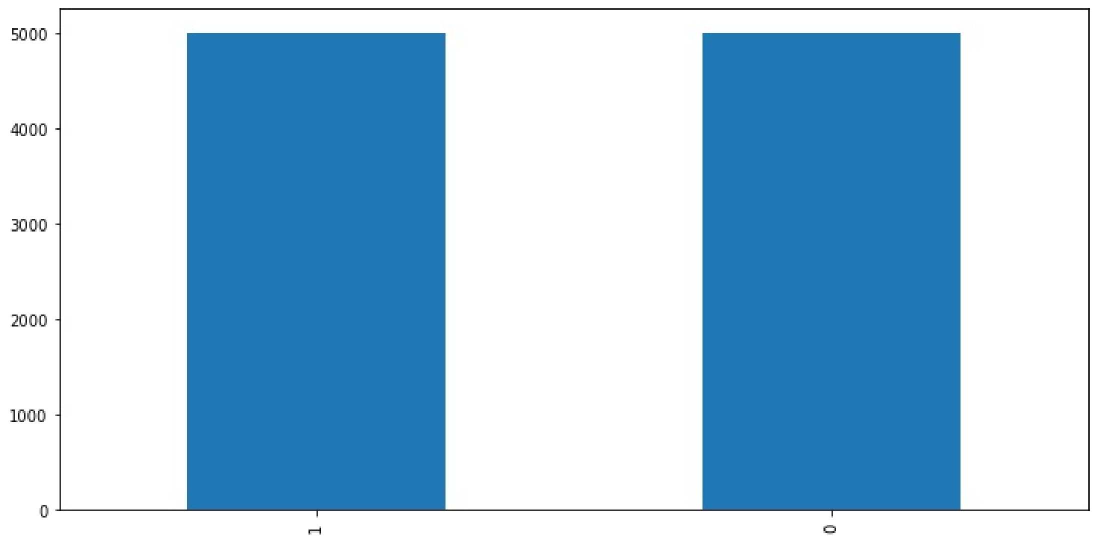
In [9]:
```python
data.describe()
```

Out[9]:

	id	NumDots	SubdomainLevel	PathLevel	UrlLength	NumDash	NumDashInHostname	AtSymbol	TildeSym
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000
mean	5000.50000	2.445100	0.586800	3.300300	70.264100	1.818000	0.138900	0.000300	0.013
std	2886.89568	1.346836	0.751214	1.863241	33.369877	3.106258	0.545744	0.017319	0.113
min	1.00000	1.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.000000	0.000
25%	2500.75000	2.000000	0.000000	2.000000	48.000000	0.000000	0.000000	0.000000	0.000
50%	5000.50000	2.000000	1.000000	3.000000	62.000000	0.000000	0.000000	0.000000	0.000
75%	7500.25000	3.000000	1.000000	4.000000	84.000000	2.000000	0.000000	0.000000	0.000
max	10000.00000	21.000000	14.000000	18.000000	253.000000	55.000000	9.000000	1.000000	1.000

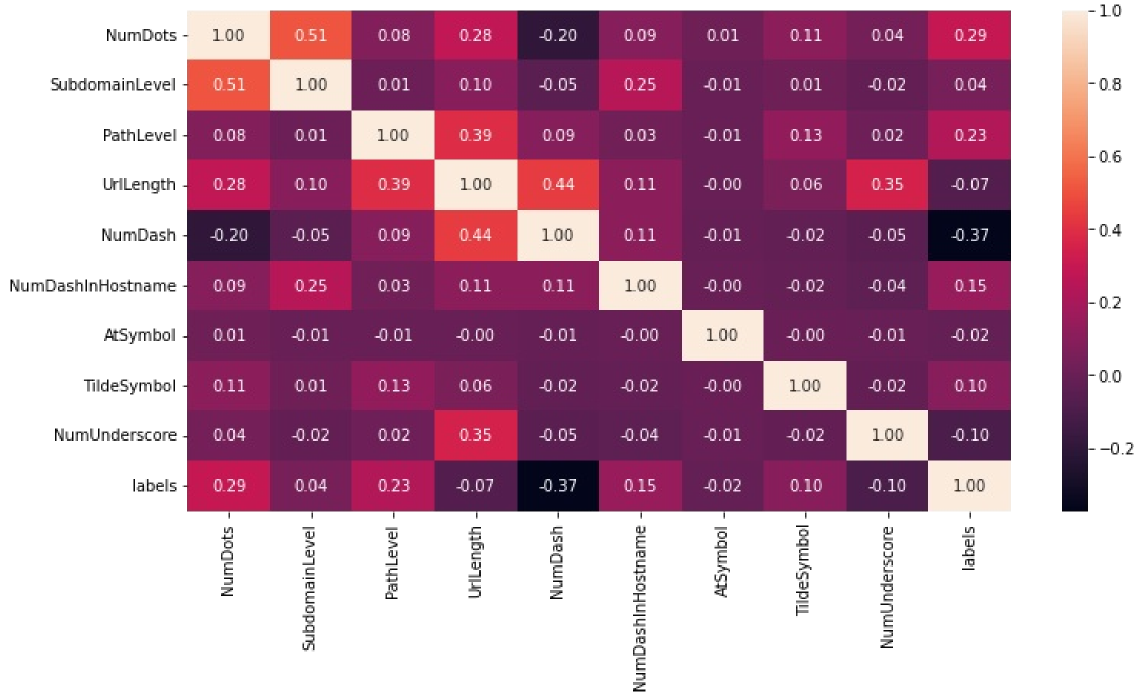
In [10]:
```python
data['labels'].value_counts().plot(kind='bar')
```

Out[10]: <AxesSubplot:>

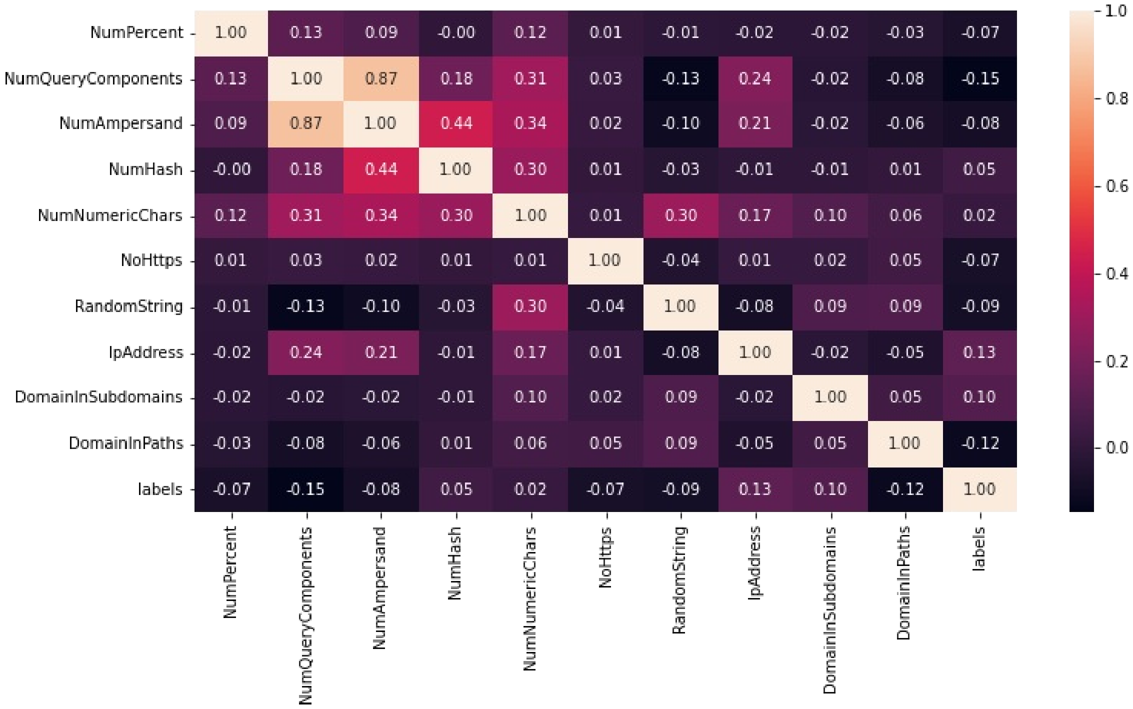


In [11]:
```python
def corr_heatmap(data, idx_s, idx_e):
    y = data['labels']
    temp = data.iloc[:, idx_s:idx_e]
    if 'id' in temp.columns:
        del temp['id']
    temp['labels'] = y
    sns.heatmap(temp.corr(), annot=True, fmt='.2f')
    plt.show()
```

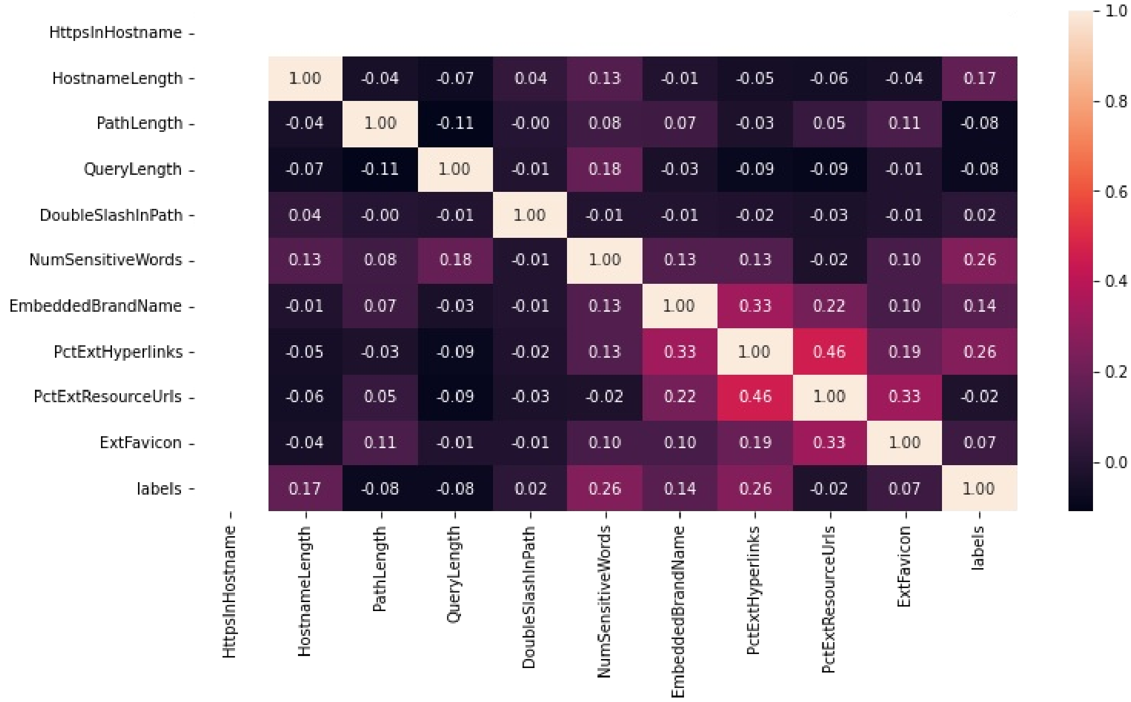
In [12]:
```python
corr_heatmap(data, 0, 10)
```



In [13]:
```
corr_heatmap(data, 10, 20)
```

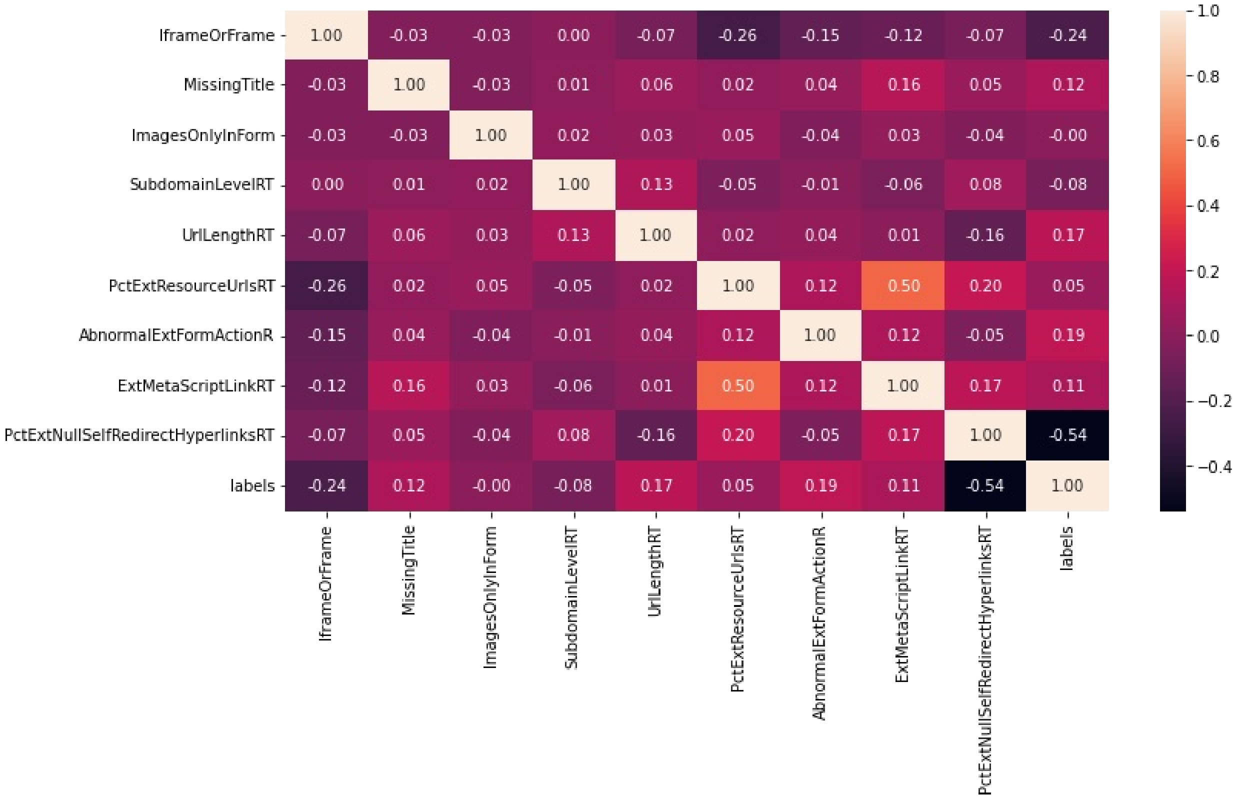


In [14]:
```
corr_heatmap(data, 20, 30)
```




```
In [15]: corr_heatmap(data, 30, 40)
```


```
In [16]: corr_heatmap(data, 40, 50)
```




```
In [17]: from sklearn.feature_selection import mutual_info_classif
```

```
In [18]: X = data.drop(['id', 'labels'], axis=1)
         y = data['labels']
```

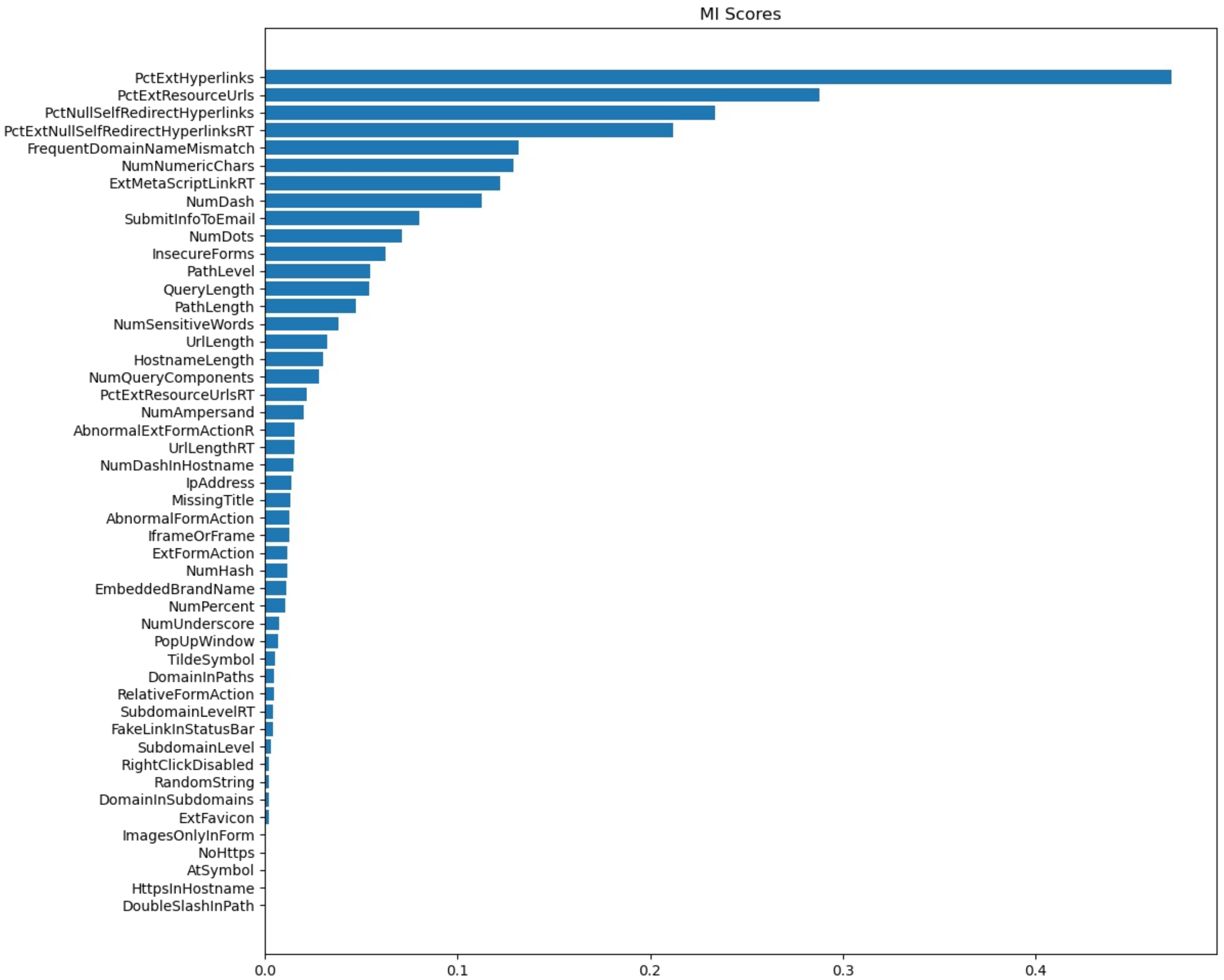
```
In [19]: discrete_features = X.dtypes == int
```

```
In [20]: mi_scores = mutual_info_classif(X, y, discrete_features=discrete_features)
         mi_scores = pd.Series(mi_scores, name='MI Scores', index=X.columns)
         mi_scores = mi_scores.sort_values(ascending=False)
         mi_scores
```

Out[20]:
```
PctExtHyperlinks                          0.470615
PctExtResourceUrls                        0.287998
PctNullSelfRedirectHyperlinks             0.233576
PctExtNullSelfRedirectHyperlinksRT        0.212144
FrequentDomainNameMismatch                0.131566
NumNumericChars                           0.129133
ExtMetaScriptLinkRT                       0.121971
NumDash                                   0.112274
SubmitInfoToEmail                         0.080130
NumDots                                   0.071204
InsecureForms                             0.062497
PathLevel                                 0.054670
QueryLength                               0.054056
PathLength                                0.047413
NumSensitiveWords                         0.038090
UrlLength                                 0.032149
HostnameLength                            0.030030
NumQueryComponents                        0.027929
PctExtResourceUrlsRT                      0.021614
NumAmpersand                              0.020247
AbnormalExtFormActionR                    0.015332
UrlLengthRT                               0.015089
NumDashInHostname                         0.014805
IpAddress                                 0.013922
MissingTitle                              0.013292
AbnormalFormAction                        0.012685
IframeOrFrame                             0.012480
ExtFormAction                             0.011569
NumHash                                   0.011511
EmbeddedBrandName                         0.011107
NumPercent                                0.010437
NumUnderscore                             0.007182
PopUpWindow                               0.006713
TildeSymbol                               0.005063
DomainInPaths                             0.004764
RelativeFormAction                        0.004580
SubdomainLevelRT                          0.004332
FakeLinkInStatusBar                       0.003984
SubdomainLevel                            0.002838
RightClickDisabled                        0.002098
RandomString                              0.002029
DomainInSubdomains                        0.001879
ExtFavicon                                0.001787
HttpsInHostname                           0.000000
ImagesOnlyInForm                          0.000000
NoHttps                                   0.000000
AtSymbol                                  0.000000
DoubleSlashInPath                         0.000000
Name: MI Scores, dtype: float64
```

In [21]:
```
def plot_mi_scores(scores):
    scores = scores.sort_values(ascending=True)
    width = np.arange(len(scores))
    ticks = list(scores.index)
    plt.barh(width, scores)
    plt.yticks(width, ticks)
    plt.title("MI Scores")

plt.figure(dpi=100, figsize=(12,12))
plot_mi_scores(mi_scores)
```



In [25]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [26]:
```
def train_logistic(data, top_n):
    top_n_features = mi_scores.sort_values(ascending=False).head(top_n).index.tolist()
    X = data[top_n_features]
    y = data['labels']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

    lr = LogisticRegression(max_iter=10000)
    lr.fit(X_train, y_train)

    y_pred = lr.predict(X_test)

    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    return precision, recall, f1, accuracy
```

In [27]:
```
arr = []
for i in range(20,51,1):
    precision, recall, f1, accuracy = train_logistic(data, i)
    print("Performance for Logistic Model with Top {} features is precision : {}, recall : {}, f1 score : {}, a
    arr.append([i, precision, recall, f1, accuracy])
```

```
Performance for Logistic Model with Top 20 features is precision : 0.8943248532289628, recall : 0.9298067141403
866, f1 score : 0.9117206982543641, accuracy : 0.9115
Performance for Logistic Model with Top 21 features is precision : 0.906496062992126, recall : 0.94364754098360
66, f1 score : 0.924698795180723, accuracy : 0.925
Performance for Logistic Model with Top 22 features is precision : 0.9306243805748265, recall : 0.939, f1 score
: 0.9347934295669488, accuracy : 0.9345
Performance for Logistic Model with Top 23 features is precision : 0.9166666666666666, recall : 0.9354508196721
312, f1 score : 0.9259634888438133, accuracy : 0.927
Performance for Logistic Model with Top 24 features is precision : 0.9279835390946503, recall : 0.9376299376299
376, f1 score : 0.9327817993795242, accuracy : 0.935
Performance for Logistic Model with Top 25 features is precision : 0.9195289499509323, recall : 0.9398194583751
254, f1 score : 0.9295634920634921, accuracy : 0.929
Performance for Logistic Model with Top 26 features is precision : 0.9164208456243854, recall : 0.9481180061037
64, f1 score : 0.932, accuracy : 0.932
Performance for Logistic Model with Top 27 features is precision : 0.9349112426035503, recall : 0.9423459244532
804, f1 score : 0.9386138613861386, accuracy : 0.938
Performance for Logistic Model with Top 28 features is precision : 0.9434914228052472, recall : 0.9285004965243
296, f1 score : 0.9359359359359359, accuracy : 0.936
Performance for Logistic Model with Top 29 features is precision : 0.9255533199195171, recall : 0.9255533199195
171, f1 score : 0.9255533199195171, accuracy : 0.926
Performance for Logistic Model with Top 30 features is precision : 0.9256360078277887, recall : 0.9422310756972
112, f1 score : 0.9338598223099704, accuracy : 0.933
Performance for Logistic Model with Top 31 features is precision : 0.9165867689357622, recall : 0.9446640316205
533, f1 score : 0.9304136253041362, accuracy : 0.9285
Performance for Logistic Model with Top 32 features is precision : 0.9058252427184466, recall : 0.9339339339339
34, f1 score : 0.9196648595367176, accuracy : 0.9185
Performance for Logistic Model with Top 33 features is precision : 0.9258188824662813, recall : 0.9533730158730
159, f1 score : 0.9393939393939394, accuracy : 0.938
Performance for Logistic Model with Top 34 features is precision : 0.9299691040164778, recall : 0.9357512953367
876, f1 score : 0.9328512396694214, accuracy : 0.935
Performance for Logistic Model with Top 35 features is precision : 0.927710843373494, recall : 0.93807106598984
77, f1 score : 0.9328621908127208, accuracy : 0.9335
Performance for Logistic Model with Top 36 features is precision : 0.9295366795366795, recall : 0.9563058589870
904, f1 score : 0.9427312775330398, accuracy : 0.9415
Performance for Logistic Model with Top 37 features is precision : 0.9315332690453231, recall : 0.9498525073746
312, f1 score : 0.9406037000973709, accuracy : 0.939
Performance for Logistic Model with Top 38 features is precision : 0.916023166023166, recall : 0.94146825396825
4, f1 score : 0.9285714285714286, accuracy : 0.927
Performance for Logistic Model with Top 39 features is precision : 0.9408866995073891, recall : 0.9408866995073
891, f1 score : 0.9408866995073891, accuracy : 0.94
Performance for Logistic Model with Top 40 features is precision : 0.9409409409409409, recall : 0.9390609390609
39, f1 score : 0.94, accuracy : 0.94
Performance for Logistic Model with Top 41 features is precision : 0.9244357212953876, recall : 0.9448345035105
316, f1 score : 0.9345238095238094, accuracy : 0.934
Performance for Logistic Model with Top 42 features is precision : 0.9257425742574258, recall : 0.9415911379657
603, f1 score : 0.9335996005991014, accuracy : 0.9335
Performance for Logistic Model with Top 43 features is precision : 0.9464469618949537, recall : 0.9484004127966
976, f1 score : 0.9474226804123711, accuracy : 0.949
Performance for Logistic Model with Top 44 features is precision : 0.9303921568627451, recall : 0.9396039603960
396, f1 score : 0.9349753694581281, accuracy : 0.934
Performance for Logistic Model with Top 45 features is precision : 0.936105476673428, recall : 0.94958847736625
52, f1 score : 0.9427987742594486, accuracy : 0.944
Performance for Logistic Model with Top 46 features is precision : 0.9460539460539461, recall : 0.9527162977867
203, f1 score : 0.94937343358396, accuracy : 0.9495
Performance for Logistic Model with Top 47 features is precision : 0.9277942631058358, recall : 0.9551934826883
91, f1 score : 0.9412945308580031, accuracy : 0.9415
Performance for Logistic Model with Top 48 features is precision : 0.9484848484848485, recall : 0.9361914257228
315, f1 score : 0.9422980431510286, accuracy : 0.9425
Performance for Logistic Model with Top 49 features is precision : 0.9375600384245918, recall : 0.9457364341085
271, f1 score : 0.9416304872165943, accuracy : 0.9395
Performance for Logistic Model with Top 50 features is precision : 0.937984496124031, recall : 0.94901960784313
72, f1 score : 0.9434697855750486, accuracy : 0.942
```

In [28]:
```
df = pd.DataFrame(arr, columns=['num_of_features', 'precision', 'recall', 'f1_score', 'accuracy'])
df
```

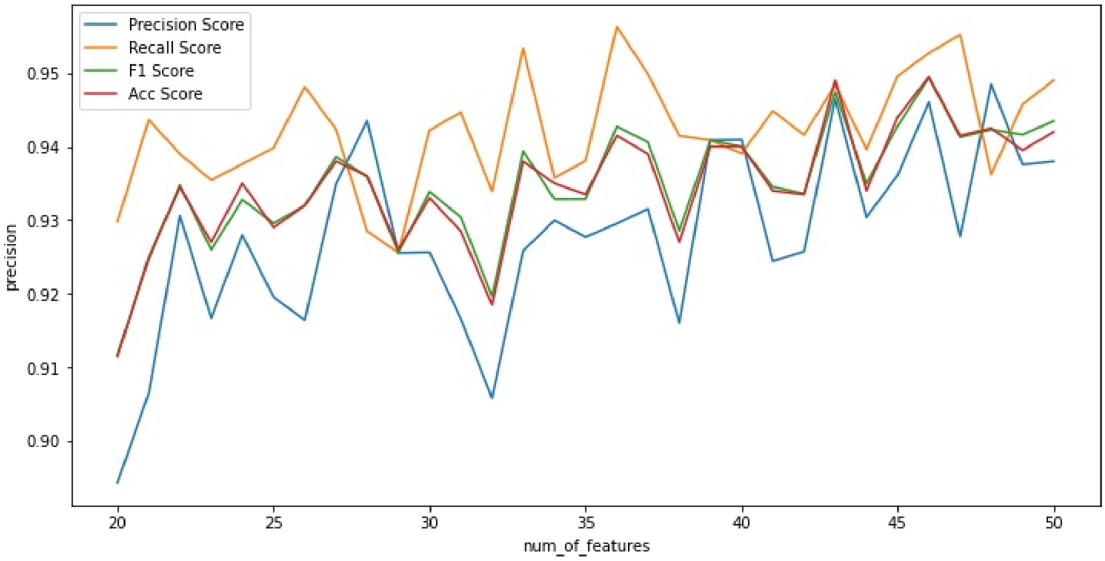
Out[28]:

	num_of_features	precision	recall	f1_score	accuracy
0	20	0.894325	0.929807	0.911721	0.9115
1	21	0.906496	0.943648	0.924699	0.9250
2	22	0.930624	0.939000	0.934793	0.9345
3	23	0.916667	0.935451	0.925963	0.9270
4	24	0.927984	0.937630	0.932782	0.9350
5	25	0.919529	0.939819	0.929563	0.9290
6	26	0.916421	0.948118	0.932000	0.9320
7	27	0.934911	0.942346	0.938614	0.9380
8	28	0.943491	0.928500	0.935936	0.9360
9	29	0.925553	0.925553	0.925553	0.9260
10	30	0.925636	0.942231	0.933860	0.9330
11	31	0.916587	0.944664	0.930414	0.9285
12	32	0.905825	0.933934	0.919665	0.9185
13	33	0.925819	0.953373	0.939394	0.9380
14	34	0.929969	0.935751	0.932851	0.9350
15	35	0.927711	0.938071	0.932862	0.9335
16	36	0.929537	0.956306	0.942731	0.9415
17	37	0.931533	0.949853	0.940604	0.9390
18	38	0.916023	0.941468	0.928571	0.9270
19	39	0.940887	0.940887	0.940887	0.9400
20	40	0.940941	0.939061	0.940000	0.9400
21	41	0.924436	0.944835	0.934524	0.9340
22	42	0.925743	0.941591	0.933600	0.9335
23	43	0.946447	0.948400	0.947423	0.9490
24	44	0.930392	0.939604	0.934975	0.9340
25	45	0.936105	0.949588	0.942799	0.9440
26	46	0.946054	0.952716	0.949373	0.9495
27	47	0.927794	0.955193	0.941295	0.9415
28	48	0.948485	0.936191	0.942298	0.9425
29	49	0.937560	0.945736	0.941630	0.9395
30	50	0.937984	0.949020	0.943470	0.9420

In [29]:

```
sns.lineplot(x='num_of_features', y='precision', data=df, label='Precision Score')
sns.lineplot(x='num_of_features', y='recall', data=df, label='Recall Score')
sns.lineplot(x='num_of_features', y='f1_score', data=df, label='F1 Score')
sns.lineplot(x='num_of_features', y='accuracy', data=df, label='Acc Score')
```

Out[29]: <AxesSubplot:xlabel='num_of_features', ylabel='precision'>



In [30]:

```
def train_rfc(data, top_n):
    top_n_features = mi_scores.sort_values(ascending=False).head(top_n).index.tolist()
    X = data[top_n_features]
    y = data['labels']
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

    rfc = cuRfc(n_estimators=500,
                split_criterion=1,
                max_depth=32,
                max_leaves=-1,
                max_features=1.0,
                n_bins=128)

    rfc.fit(X_train, y_train)

    y_pred = rfc.predict(X_test, predict_model='CPU')

    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    return precision, recall, f1, accuracy
```

In [33]:
```
df = pd.DataFrame(arr, columns=['num_of_features', 'precision', 'recall', 'f1_score', 'accuracy'])
df.head()
```

Out[33]:

	num_of_features	precision	recall	f1_score	accuracy
0	20	0.937984	0.94902	0.94347	0.942
1	21	0.937984	0.94902	0.94347	0.942
2	22	0.937984	0.94902	0.94347	0.942
3	23	0.937984	0.94902	0.94347	0.942
4	24	0.937984	0.94902	0.94347	0.942

In [43]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train=scaler.fit_transform(X_train)
x_test=scaler.transform(X_test)
```

In [45]:
```
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier(random_state = 0)
classifier.fit(x_train, y_train)
```

Out[45]:
```
RandomForestClassifier(random_state=0)
```

In [46]:
```
y_pred1=classifier.predict(x_test)
y_pred1_train=classifier.predict(x_train)
```

In [54]:
```
from sklearn.metrics import confusion_matrix,mean_absolute_error,mean_squared_error,recall_score,accuracy_score
```

In [48]:
```
confusion_matrix(y_test,y_pred1)
```

Out[48]:
```
array([[ 970,   14],
       [  14, 1002]], dtype=int64)
```

In [49]:
```
print("Recall is", recall_score(y_test,y_pred1))
```

```
Recall is 0.9862204724409449
```

In [50]:
```
print("Precision is", precision_score(y_test,y_pred1))
```

```
Precision is 0.9862204724409449
```

In [51]:
```
print("Mean Absolute Error is",mean_absolute_error(y_test,y_pred1))
```

```
Mean Absolute Error is 0.014
```

In [52]:
```
print("Mean Squared Error is",mean_squared_error(y_test,y_pred1))
```

```
Mean Squared Error is 0.014
```

In [55]:
```
print(" Root Mean Squared Error is",mean_squared_error(y_test,y_pred1,squared=False))
```

```
 Root Mean Squared Error is 0.11832159566199232
```

In [57]:
```
print("Accuracy is",accuracy_score(y_test,y_pred1))
```

```
Accuracy is 0.986
```

In [59]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       984
           1       0.99      0.99      0.99      1016

    accuracy                           0.99      2000
   macro avg       0.99      0.99      0.99      2000
weighted avg       0.99      0.99      0.99      2000
```

In [ ]:

In [ ]:

In [ ]: