In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [2]:

```
df1 = pd.read_csv('Copy of archive(1)/urldata.csv', index_col=0)
df1.head()
```

Out[2]:

	url	label	result
0	https://www.google.com	benign	0
1	https://www.youtube.com	benign	0
2	https://www.facebook.com	benign	0
3	https://www.baidu.com	benign	0
4	https://www.wikipedia.org	benign	0

In [3]:

```
df1.result[df1['label']=='benign'].unique()
```

Out[3]:

```
array([0], dtype=int64)
```

In [4]:

```
df1.label[df1['result']==1].unique()
```

Out[4]:

```
array(['malicious'], dtype=object)
```

In [5]:

```
df2 = pd.read_csv('Copy of archive(2)/data.csv')
df2.head()
```

Out[5]:

	url	label
0	diaryofagameaddict.com	bad
1	espdesign.com.au	bad
2	iamagameaddict.com	bad
3	kalantzis.net	bad
4	slightlyoffcenter.net	bad

In [6]:

```
df2['label'].unique()
```

Out[6]:

```
array(['bad', 'good'], dtype=object)
```

In [7]:

```
df3 = pd.read_csv('Copy of archive(3)/phishing_site_urls.csv')
df3.head()
```

Out[7]:

	URL	Label
0	nobell.it/70ffb52d079109dca5664cce6f317373782/...	bad
1	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...	bad
2	serviciosbys.com/paypal.cgi.bin.get-into.herf....	bad
3	mail.printakid.com/www.online.americanexpress....	bad
4	thewhiskeydregs.com/wp-content/themes/widescre...	bad

In [8]:

```
df4 = pd.read_csv('Copy of archive(4)/malicious_phish.csv')
df4.head()
```

Out[8]:

	url	type
0	br-icloud.com.br	phishing
1	mp3raid.com/music/krizz_kaliko.html	benign
2	bopsecrets.org/rexroth/cr/1.htm	benign
3	http://www.garage-pirenne.be/index.php?option=...	defacement
4	http://adventure-nicaragua.net/index.php?optio...	defacement

In [9]:

```
df4['type'].unique()
```

Out[9]:

```
array(['phishing', 'benign', 'defacement', 'malware'], dtype=object)
```

In [10]:

```
df4.shape
```

Out[10]:

```
(651191, 2)
```

In [11]:

```
df3.shape
```

Out[11]:

```
(549346, 2)
```

In [12]:

```
df2.shape
```

Out[12]:

```
(420464, 2)
```

In [13]:

```
df1.shape
```

Out[13]:

```
(450176, 3)
```

In [14]:

```
df1.head()
```

Out[14]:

	url	label	result
0	https://www.google.com	benign	0
1	https://www.youtube.com	benign	0
2	https://www.facebook.com	benign	0
3	https://www.baidu.com	benign	0
4	https://www.wikipedia.org	benign	0

In [15]:

```
def PreProcessing(df):
    feature = {'url_length':[],
            'WWW Present':[],
            'Digit to Alphabet ratio':[],
            'Uppercase to LowercaseRatio':[],
            'Dots':[],
            'Semicolon':[],
            'Underscore':[],
            'Question Mark':[],
            'Exclamation Mark':[],
            'Hash Character':[],
            'Equals':[],
            'Percentage Character':[],
            'Ampersand':[],
            'Dash':[],
            'Double Slash':[],
            'Https in URL':[]}

    for url in df['url']:

        feature['url_length'].append(len(url))

        if 'www' in url.lower():
            feature['WWW Present'].append(1)
        else:
            feature['WWW Present'].append(0)

        if 'https' in url.lower():
            feature['Https in URL'].append(1)
        else:
            feature['Https in URL'].append(0)

        feature['Dots'].append(url.count('.'))
        feature['Semicolon'].append(url.count(';'))
        feature['Underscore'].append(url.count('_'))
        feature['Question Mark'].append(url.count('?'))
        feature['Exclamation Mark'].append(url.count('!'))
        feature['Hash Character'].append(url.count('#'))
        feature['Equals'].append(url.count('='))
        feature['Percentage Character'].append(url.count('%'))
        feature['Ampersand'].append(url.count('&'))
        feature['Dash'].append(url.count('-'))
        feature['Double Slash'].append(url.count('//'))

        alpha = 0
        num = 0
        upper = 0
        lower = 0

        for i in url:
            if i.isalpha():
                alpha += 1
            if i.isnumeric():
                num += 1
            if i.isupper():
                upper += 1
            if i.islower():
                lower += 1

        feature['Digit to Alphabet ratio'].append(num/alpha)
        feature['Uppercase to LowercaseRatio'].append(upper/lower)

    new_df = pd.DataFrame(feature)

    df = pd.concat([df, new_df], axis=1)

    return df
```

In [16]:

```
df1 = PreProcessing(df1)
df1.head()
```

Out[16]:

	url	label	result	url_length	WWW Present	Digit to Alphabet ratio	Uppercase to LowercaseRatio	Dots	Semicolon	Underscore	Question Mark	Exclamation Mark	Char
0	https://www.google.com	benign	0	22	1	0.0	0.0	2	0	0	0	0	
1	https://www.youtube.com	benign	0	23	1	0.0	0.0	2	0	0	0	0	
2	https://www.facebook.com	benign	0	24	1	0.0	0.0	2	0	0	0	0	
3	https://www.baidu.com	benign	0	21	1	0.0	0.0	2	0	0	0	0	
4	https://www.wikipedia.org	benign	0	25	1	0.0	0.0	2	0	0	0	0	

In [17]:

```
df1.tail(30)
```

Out[17]:

	url	label	result	url_length	WWW Present	Digit to Alphabet ratio	Uppercase to LowercaseRatio	Dots	Semicolon	Underscore
450146	http://leentra1.com/ST/SOS	malicious	1	26	0	0.052632	0.357143	1	0	0
450147	http://nb84.servidoraweb.net/load/dropbox2016/...	malicious	1	60	0	0.136364	0.023256	3	0	0
450148	http://www.yourbusinesswebapp.com/wp-content/t...	malicious	1	77	1	0.000000	0.000000	3	0	0
450149	http://draywalejohn.com/Google/	malicious	1	31	0	0.000000	0.041667	1	0	0
450150	http://www.sadhakayogaiyengar.com/wp-includes/...	malicious	1	69	1	0.000000	0.035714	3	0	0
450151	http://gaptrade.cl/file/bless/index.php?email=	malicious	1	46	0	0.000000	0.000000	2	0	0
450152	http://gospelchurchofchrist.org/best_update/pr...	malicious	1	63	0	0.000000	0.000000	2	0	1
450153	http://thepizzaplacesandimas.com/includes/book...	malicious	1	272	0	0.382716	0.038462	14	10	0
450154	http://ui3china.com/still/G.Docs/index.php	malicious	1	42	0	0.031250	0.066667	3	0	0
450155	http://ui3china.com/live/G.Docs/index.php	malicious	1	41	0	0.032258	0.068966	3	0	0
450156	http://orlandoresorthouses.com/wwe/script/mail...	malicious	1	70	0	0.000000	0.017544	2	0	2
450157	http://knitwear.ru/LinkedInen.html	malicious	1	34	0	0.000000	0.037037	2	0	0
450158	http://dizcorona.com/Via/Validation	malicious	1	35	0	0.000000	0.074074	1	0	0
450159	http://ayareview-document.pdf-iso.webapps-secu...	malicious	1	100	0	0.061728	0.000000	5	0	0
450160	http://www.rosespa.com.sg/ipic/Dirk/index.php	malicious	1	45	1	0.000000	0.029412	4	0	0
450161	http://facebookauthorization.whatsgratis.com/f/	malicious	1	47	0	0.000000	0.000000	2	0	0
450162	http://u.to/vYjNDw,Pattern	malicious	1	26	0	0.000000	0.250000	1	0	0
450163	https://insidethestorex.com/sd/	malicious	1	31	0	0.000000	0.000000	1	0	0
450164	http://youthsocialcircle.com/docs/Womsdhgdfhds...	malicious	1	154	0	0.000000	0.006849	1	0	0
450165	http://perrottaimmobiliare.it/img/immobiliari/...	malicious	1	78	0	0.122807	0.036364	2	0	5
450166	http://businessmobilewebapp.com/jobi/dh/dhl.htm	malicious	1	47	0	0.000000	0.000000	2	0	0
450167	http://bishopinegypt.gdn/wp-database/wp-databa...	malicious	1	56	0	0.022222	0.000000	2	0	0
450168	http://boasecg7.beget.tech/cgi-bin/index/pcg/f...	malicious	1	72	0	0.134615	0.000000	2	0	0
450169	http://gangainsulations.com/alert/GD/	malicious	1	37	0	0.000000	0.071429	1	0	0
450170	https://qiumin.xyz/qiuminxy/o95j4uW4nr/5.php	malicious	1	44	0	0.161290	0.033333	2	0	0
450171	http://ecct-it.com/docmmmnn/aptgd/index.php	malicious	1	43	0	0.000000	0.000000	2	0	0
450172	http://faboleena.com/js/infortis/jquery/plugin...	malicious	1	159	0	0.177966	0.063063	2	0	2
450173	http://faboleena.com/js/infortis/jquery/plugin...	malicious	1	147	0	0.183486	0.068627	1	0	1
450174	http://atualizapj.com/	malicious	1	22	0	0.000000	0.000000	1	0	0
450175	http://writeassociate.com/test/Portal/inicio/I...	malicious	1	143	1	0.076271	0.456790	4	0	1

In [18]:

```
df1['Dots'].unique()
```

Out[18]:

```
array([ 2,  3,  4,  5,  6,  7, 11, 10,  9, 13, 14, 16, 12, 18,  8, 15, 19,
       28, 22,  1, 17, 25, 26, 21, 27, 30, 24, 20, 23, 29,  0, 32],
      dtype=int64)
```

In [19]:

```
1 df1.describe()
```

Out[19]:

	result	url_length	WWW Present	Digit to Alphabet ratio	Uppercase to LowercaseRatio	Dots	Semicolon	Underscore	Question Mark	
count	450176.000000	450176.000000	450176.000000	450176.000000	450176.000000	450176.000000	450176.000000	450176.000000	450176.000000	4
mean	0.231994	60.237849	0.802095	0.092747	0.039481	2.620553	0.047666	0.419527	0.152854	
std	0.422105	37.571613	0.398420	0.211052	0.144652	1.144966	0.568434	1.337560	0.462658	
min	0.000000	8.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	40.000000	1.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	
50%	0.000000	52.000000	1.000000	0.018182	0.000000	2.000000	0.000000	0.000000	0.000000	
75%	0.000000	71.000000	1.000000	0.120690	0.045455	3.000000	0.000000	0.000000	0.000000	
max	1.000000	2314.000000	1.000000	5.500000	13.250000	32.000000	30.000000	200.000000	166.000000	

In [20]:

```
1 y = df1['result']
2 x = df1.drop(columns=['url', 'label', 'result'])
```

In [21]:

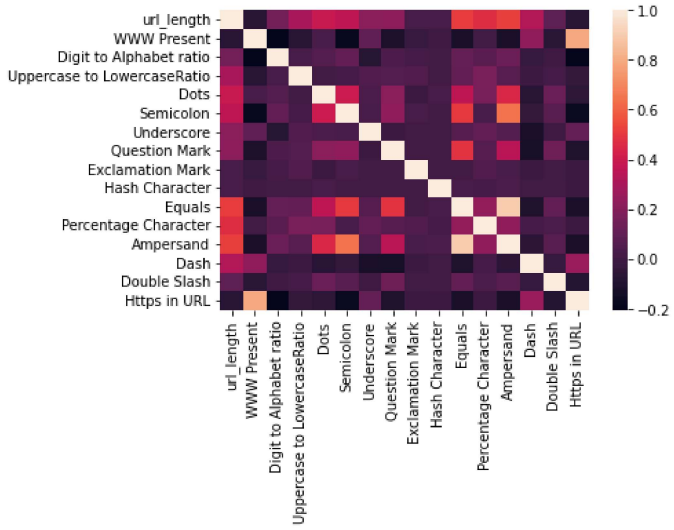
```
1 df1_x = df1.drop(index=x[x.duplicated()].index)
2 y.drop(index=x[x.duplicated()].index, inplace=True)
3 x.drop(index=x[x.duplicated()].index, inplace=True)
```

In [22]:

```
1 sns.heatmap(x.corr())
```

Out[22]:

<AxesSubplot:>



In [23]:

```
x.corr()
```

Out[23]:

	url_length	WWW Present	Digit to Alphabet ratio	Uppercase to LowercaseRatio	Dots	Semicolon	Underscore	Question Mark	Exclamation Mark	Hash Character	Equals	P
url_length	1.000000	-0.075382	0.147311	0.293656	0.390169	0.363812	0.211023	0.215891	0.022465	0.023588	0.511828	
WWW Present	-0.075382	1.000000	-0.206171	-0.077289	0.025669	-0.178890	0.093320	-0.113691	-0.026077	-0.007692	-0.122458	
Digit to Alphabet ratio	0.147311	-0.206171	1.000000	0.023532	0.053040	0.101449	-0.080851	0.042702	0.014926	0.006359	0.105507	
Uppercase to LowercaseRatio	0.293656	-0.077289	0.023532	1.000000	0.008458	0.012593	0.047721	0.051645	0.047151	0.001372	0.101979	
Dots	0.390169	0.025669	0.053040	0.008458	1.000000	0.398297	0.037363	0.208851	-0.009115	0.024302	0.359414	
Semicolon	0.363812	-0.178890	0.101449	0.012593	0.398297	1.000000	0.028710	0.231235	0.018315	0.008985	0.505223	
Underscore	0.211023	0.093320	-0.080851	0.047721	0.037363	0.028710	1.000000	-0.010083	0.005282	-0.002162	0.062523	
Question Mark	0.215891	-0.113691	0.042702	0.051645	0.208851	0.231235	-0.010083	1.000000	0.007361	0.010315	0.484411	
Exclamation Mark	0.022465	-0.026077	0.014926	0.047151	-0.009115	0.018315	0.005282	0.007361	1.000000	0.007397	0.009516	
Hash Character	0.023588	-0.007692	0.006359	0.001372	0.024302	0.008985	-0.002162	0.010315	0.007397	1.000000	0.025777	
Equals	0.511828	-0.122458	0.105507	0.101979	0.359414	0.505223	0.062523	0.484411	0.009516	0.025777	1.000000	
Percentage Character	0.470731	-0.001065	0.083422	0.166710	0.156849	0.025619	0.101988	0.067055	0.044557	0.011419	0.236830	
Ampersand	0.523710	-0.119910	0.124135	0.073258	0.453375	0.642152	0.060430	0.346849	0.029035	0.033789	0.900306	
Dash	0.325319	0.227447	-0.032886	-0.014747	-0.078185	-0.036918	-0.125059	-0.129143	-0.008297	-0.003529	-0.110103	
Double Slash	0.088822	-0.071096	-0.007523	0.014967	0.122918	0.075594	-0.004722	0.132473	-0.001325	0.002389	0.097800	
Https in URL	-0.075696	0.792487	-0.192699	-0.037557	-0.060543	-0.169472	0.104898	-0.111009	-0.010401	-0.009806	-0.117075	

In [22]:

```
from sklearn.feature_selection import chi2
```

In [25]:

```
score, p_value = chi2(x, y)
print(p_value)
```

```
[0.00000000e+000 0.00000000e+000 0.00000000e+000 2.00375117e-099
 3.24211538e-117 0.00000000e+000 0.00000000e+000 0.00000000e+000
 7.98704090e-061 2.12604316e-250 0.00000000e+000 5.00146132e-274
 0.00000000e+000 0.00000000e+000 6.50225268e-009 0.00000000e+000]
```

In [26]:

```
print(score)
```

```
[5.36950516e+04 2.82623608e+04 1.50361820e+03 4.47961077e+02
 5.29724762e+02 3.60988142e+04 8.64154522e+03 4.58422223e+03
 2.70699826e+02 1.14228990e+03 1.09177420e+04 1.25101228e+03
 1.42015757e+04 6.01914282e+04 3.36782841e+01 3.77554938e+04]
```

In [77]:

```
lst = list(map(int, df1['Https in URL'][df1['result']==1].values))
lst = np.array(lst)
len(np.where(lst==0)[0])/len(lst)
```

Out[77]:

```
0.930159520481051
```

In [79]:

```
lst = list(map(int, df1['Https in URL'][df1['result']==0].values))
lst = np.array(lst)
len(np.where(lst==1)[0])/len(lst)
```

Out[79]:

```
0.9999971076364184
```

In [84]:

```
lst = list(map(int, df1['WWW Present'][df1['result']==1].values))
lst = np.array(lst)
len(np.where(lst==0)[0])/len(lst)
```

Out[84]:

```
0.8465213811064938
```

In [82]:

```
lst = list(map(int, df1['WWW Present'][df1['result']==0].values))
lst = np.array(lst)
len(np.where(lst==1)[0])/len(lst)
```

Out[82]:

```
0.9980245156737183
```

In [130]:

```
lst = df1['url_length'][df1['result']==1].values
lst = np.array(lst)
print(len(lst[lst<150])/len(lst))
print(len(lst[lst<200])/len(lst))
print(len(lst[lst>150])/len(lst))
print(len(lst[lst>200])/len(lst))
print(len(lst[lst<100])/len(lst))
print(len(lst[lst>100])/len(lst))
print(len(lst[lst<50])/len(lst))
print(len(lst[lst>50])/len(lst))
```

```
0.9369673873494322
0.963882877879699
0.06199850629081369
0.03576284494149639
0.8692334207855378
0.12852601543499492
0.49616040138646855
0.4892376338114479
```

In [122]:

```
len(lst[lst<150])/len(lst)
```

Out[122]:

```
0.9369673873494322
```

In [129]:

```
lst = df1['url_length'][df1['result']==0].values
lst = np.array(lst)
print(len(lst[lst<150])/len(lst))
print(len(lst[lst<100])/len(lst))
print(len(lst[lst>150])/len(lst))
print(len(lst[lst>100])/len(lst))
print(len(lst[lst<50])/len(lst))
print(len(lst[lst>50])/len(lst))
```

```
0.9925058859598888
0.9336781030722686
0.007175954046127414
0.06351051952634654
0.43058905876704323
0.5491210107075299
```

In [131]:

```
lst = df1['Digit to Alphabet ratio'][df1['result']==1].values
lst = np.array(lst)
print(len(lst[lst<0.5])/len(lst))
print(len(lst[lst>0.5])/len(lst))
```

```
0.9207663877132845
0.0760259675597005
```

In [132]:

```
lst = df1['Digit to Alphabet ratio'][df1['result']==0].values
lst = np.array(lst)
print(len(lst[lst<0.5])/len(lst))
print(len(lst[lst>0.5])/len(lst))
```

```
0.9903944605452684
0.006987950413318756
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [23]:

```
1 from sklearn.preprocessing import StandardScaler
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LogisticRegression, Perceptron
```

In [24]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=47, stratify=y)
```

In [25]:

```
print(x_train.shape, x_test.shape, x.shape)
```

```
(121462, 16) (30366, 16) (151828, 16)
```

In [26]:

```
ss = StandardScaler()
ss.fit(x_test)

x_train = ss.transform(x_train)
x_test = ss.transform(x_test)
```

In [58]:

```
model = LogisticRegression()
model.fit(x_train, y_train)

pred = model.predict(x_test)
```

In [27]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, precision_score, f1_score, roc_curve, roc_auc_score
```

In [60]:

```
print(confusion_matrix(y_test, pred))
print(accuracy_score(y_test, pred), recall_score(y_test, pred), precision_score(y_test, pred),  f1_score(y_test, pred))
```

```
[[20686    64]
 [  116  9500]]
0.99407231772377 0.987936772046589 0.9933082392304475 0.9906152241918665
```

In [61]:

```
pred_train = model.predict(x_train)

print(confusion_matrix(y_train, pred_train))
print(accuracy_score(y_train, pred_train), recall_score(y_train, pred_train), precision_score(y_train, pred_train),  f1_score(y_t
```

```
[[82731   268]
 [  480 37983]]
0.9938416953450462 0.9875204742219795 0.9929936472249091 0.9902494981359334
```

In [62]:

```
percep = Perceptron(random_state=42)
percep.fit(x_train, y_train)

pred_percep = percep.predict(x_test)
print(confusion_matrix(y_test, pred_percep))
print(accuracy_score(y_test, pred_percep), recall_score(y_test, pred_percep), precision_score(y_test, pred_percep),  f1_score(y_t
```

```
[[20578   172]
 [  118  9498]]
0.9904498452216295 0.9877287853577371 0.9822130299896588 0.9849631857305818
```

In [28]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, BaggingClassifier
import xgboost as xgb
from catboost import CatBoostClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

In [134]:

```
model2 = DecisionTreeClassifier()
model2.fit(x_train, y_train)

pred2 = model2.predict(x_test)
print(confusion_matrix(y_test, pred2))
print(accuracy_score(y_test, pred2), recall_score(y_test, pred2), precision_score(y_test, pred2),  f1_score(y_test, pred2))
```

```
[[20617   133]
 [   98  9518]]
0.9923928077455049 0.9898086522462562 0.9862190446585846 0.9880105880521098
```

In [135]:

```
pred_train = model2.predict(x_train)

print(confusion_matrix(y_train, pred_train))
print(accuracy_score(y_train, pred_train), recall_score(y_train, pred_train), precision_score(y_train, pred_train),  f1_score(y_t
```

```
[[82999     0]
 [    0 38463]]
1.0 1.0 1.0 1.0
```

In [87]:

```
bag_clf = BaggingClassifier(base_estimator=model2, n_estimators=500)
bag_clf.fit(x_train, y_train)

pred_bag = bag_clf.predict(x_test)
print(confusion_matrix(y_test, pred_bag))
print(accuracy_score(y_test, pred_bag), recall_score(y_test, pred_bag), precision_score(y_test, pred_bag),  f1_score(y_test, pred
```

```
[[20701    49]
 [   92  9524]]
0.9953566488836199 0.990432612312812 0.9948814373759532 0.9926520402313825
```

In [68]:

```
import matplotlib.pyplot as plt
from sklearn import tree
plt.figure(figsize=(15,15))
tree.plot_tree(model2, filled=True, feature_names=x.columns)
```

Out[68]:

```
[Text(453.90993262965804, 803.4088235294117, 'Https in URL <= -0.471\ngini = 0.433\nsamples = 121462\nvalue = [8299
9, 38463]'),
 Text(453.043697183345, 779.4264705882352, 'gini = 0.0\nsamples = 34814\nvalue = [0, 34814]'),
 Text(454.7761680759711, 779.4264705882352, 'WWW Present <= -0.554\ngini = 0.081\nsamples = 86648\nvalue = [82999, 3
649]'),
 Text(115.9905335341203, 755.4441176470588, 'Digit to Alphabet ratio <= -0.706\ngini = 0.125\nsamples = 3196\nvalue
= [215, 2981]'),
 Text(67.36334087968952, 731.4617647058824, 'url_length <= -0.63\ngini = 0.336\nsamples = 744\nvalue = [159, 585]'),
 Text(38.121127102199225, 707.4794117647059, 'Uppercase to LowercaseRatio <= -0.226\ngini = 0.465\nsamples = 342\nva
lue = [126, 216]'),
 Text(22.58979624838292, 683.4970588235294, 'url_length <= -0.959\ngini = 0.498\nsamples = 216\nvalue = [102, 11
4]'),
 Text(6.063648124191461, 659.5147058823529, 'Dash <= -0.5\ngini = 0.412\nsamples = 69\nvalue = [49, 20]'),
 Text(4.33117723156533, 635.5323529411764, 'Hash Character <= 10.857\ngini = 0.268\nsamples = 44\nvalue = [37, 7]'),
 Text(3.4649417852522637, 611.55, 'Question Mark <= 0.504\ngini = 0.21\nsamples = 42\nvalue = [37, 5]'),
 Text(2.5987063389391976, 587.5676470588235, 'Double Slash <= 4.106\ngini = 0.139\nsamples = 40\nvalue = [37, 3]'),
 Text(1.7324708926261319, 563.5852941176471, 'Underscore <= -0.159\ngini = 0.051\nsamples = 38\nvalue = [37, 1]'),
 Text(0.8662354463130659, 539.6029411764706, 'gini = 0.0\nsamples = 37\nvalue = [37, 0]'),
```

In [31]:

```
from sklearn.model_selection import GridSearchCV
```

In [35]:

```
rf = RandomForestClassifier()
param_rf = {'n_estimators':[200, 400, 500, 600, 700, 800, 1000]}

gscv_rf = GridSearchCV(rf, param_grid=param_rf, n_jobs=-1)
gscv_rf.fit(x_train, y_train)

pred_rf = gscv_rf.predict(x_test)
print(confusion_matrix(y_test, pred_rf))
print(accuracy_score(y_test, pred_rf), recall_score(y_test, pred_rf), precision_score(y_test, pred_rf),  f1_score(y_test, pred_rf
```

```
[[20714    36]
 [   87  9529]]
0.9959494171112429 0.9909525790349417 0.9962362780972295 0.9935874042020749
```

In [36]:

```
print(gscv_rf.best_params_)
```

Out[36]:

```
{'n_estimators': 600}
```

In [37]:

```
gb = GradientBoostingClassifier()
param_gb = {'n_estimators':[700, 800, 900, 1000, 1100, 1200, 1300]}

gscv_gb = GridSearchCV(gb, param_grid=param_gb, n_jobs=-1)
gscv_gb.fit(x_train, y_train)

pred_gb = gscv_gb.predict(x_test)
print(confusion_matrix(y_test, pred_gb))
print(accuracy_score(y_test, pred_gb), recall_score(y_test, pred_gb), precision_score(y_test, pred_gb),  f1_score(y_test, pred_gb
```

```
[[20712    38]
 [   92  9524]]
0.9957188961338339 0.990432612312812 0.9960259359966535 0.9932213995202837
```

In [38]:

```
gscv_gb.best_params_
```

Out[38]:

```
{'n_estimators': 700}
```

In [39]:

```
ab = AdaBoostClassifier()
param_ab = {'n_estimators':[700, 800, 900, 1000, 1100, 1200, 1300]}

gscv_ab = GridSearchCV(ab, param_grid=param_ab, n_jobs=-1)
gscv_ab.fit(x_train, y_train)

pred_ab = gscv_ab.predict(x_test)
print(confusion_matrix(y_test, pred_ab))
print(accuracy_score(y_test, pred_ab), recall_score(y_test, pred_ab), precision_score(y_test, pred_ab),  f1_score(y_test, pred_ab
```

```
[[20683    67]
 [  108  9508]]
0.9942369755647764 0.9887687188019967 0.9930026109660575 0.9908811422020739
```

In [40]:

```
gscv_ab.best_params_
```

Out[40]:

```
{'n_estimators': 700}
```

In [41]:

```
xb = xgb.XGBClassifier()
param_xb = {'n_estimators':[700, 800, 900, 1000, 1100, 1200, 1300]}

gscv_xb = GridSearchCV(xb, param_grid=param_xb, n_jobs=-1)
gscv_xb.fit(x_train, y_train)

pred_xb = gscv_xb.predict(x_test)
print(confusion_matrix(y_test, pred_xb))
print(accuracy_score(y_test, pred_xb), recall_score(y_test, pred_xb), precision_score(y_test, pred_xb),  f1_score(y_test, pred_xb
```

```
D:\AnacondaNavigator\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_lab
el_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.
e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[13:39:26] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoo
st 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
[[20707    43]
 [   84  9532]]
0.9958176908384377 0.9912645590682196 0.995509138381201 0.9933823146266478
```

In [42]:

```
gscv_xb.best_params_
```

Out[42]:

```
{'n_estimators': 700}
```

In [32]:

```
cb = CatBoostClassifier()
param_cb = {'n_estimators':[500, 600, 700, 800, 900, 1000]}

gscv_cb = GridSearchCV(cb, param_grid=param_cb, n_jobs=-1)
gscv_cb.fit(x_train, y_train)

pred_cb = gscv_cb.predict(x_test)
print(confusion_matrix(y_test, pred_cb))
print(accuracy_score(y_test, pred_cb), recall_score(y_test, pred_cb), precision_score(y_test, pred_cb), f1_score(y_test, pred_cb)
```

```
Learning rate set to 0.079982
0:      learn: 0.4884317        total: 42.5ms   remaining: 42.5s
1:      learn: 0.3391969        total: 77.4ms   remaining: 38.6s
2:      learn: 0.2446293        total: 113ms    remaining: 37.4s
3:      learn: 0.1817542        total: 148ms    remaining: 37s
4:      learn: 0.1397579        total: 172ms    remaining: 34.3s
5:      learn: 0.1084162        total: 204ms    remaining: 33.8s
6:      learn: 0.0887626        total: 237ms    remaining: 33.6s
7:      learn: 0.0736360        total: 269ms    remaining: 33.4s
8:      learn: 0.0619247        total: 304ms    remaining: 33.5s
9:      learn: 0.0535453        total: 340ms    remaining: 33.7s
10:     learn: 0.0472770        total: 367ms    remaining: 33s
11:     learn: 0.0425446        total: 396ms    remaining: 32.6s
12:     learn: 0.0389147        total: 430ms    remaining: 32.6s
13:     learn: 0.0363875        total: 462ms    remaining: 32.5s
14:     learn: 0.0344540        total: 497ms    remaining: 32.6s
15:     learn: 0.0327808        total: 527ms    remaining: 32.4s
16:     learn: 0.0314837        total: 558ms    remaining: 32.3s
17:     learn: 0.0303401        total: 588ms    remaining: 32.1s
```

In [33]:

```
gscv_cb.best_params_
```

Out[33]:

```
{'n_estimators': 1000}
```

In [33]:

```
import math
n = (int(math.sqrt(x_train.shape[0]))//2)*2+1
```

In [34]:

```
knn = KNeighborsClassifier(n_neighbors=n, weights='distance')
knn.fit(x_train, y_train)

pred_knn = knn.predict(x_test)
print(confusion_matrix(y_test, pred_knn))
print(accuracy_score(y_test, pred_knn), recall_score(y_test, pred_knn), precision_score(y_test, pred_knn),  f1_score(y_test, pred
```

```
[[20674    76]
 [  207  9409]]
0.9906803661990384 0.9784733777038269 0.991987348444913 0.9851840217789645
```

In [45]:

```
sv = SVC()
sv.fit(x_train, y_train)

pred_sv = sv.predict(x_test)
print(confusion_matrix(y_test, pred_sv))
print(accuracy_score(y_test, pred_sv), recall_score(y_test, pred_sv), precision_score(y_test, pred_sv),  f1_score(y_test, pred_sv
```

```
[[20690    60]
 [  125  9491]]
0.9939076598827636 0.9870008319467554 0.9937179352947335 0.9903479939479313
```

In [35]:

```
for i in [53, 72, 96, 21]:
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=i, stratify=y)

    ss = StandardScaler()
    ss.fit(x_test)

    x_train = ss.transform(x_train)
    x_test = ss.transform(x_test)


    model = LogisticRegression()
    model.fit(x_train, y_train)

    pred = model.predict(x_test)
    print("LR:")
    print(confusion_matrix(y_test, pred))
    print(accuracy_score(y_test, pred), recall_score(y_test, pred), precision_score(y_test, pred),  f1_score(y_test, pred))


    percep = Perceptron(random_state=42)
    percep.fit(x_train, y_train)

    pred_percep = percep.predict(x_test)
    print("Percep:")
    print(confusion_matrix(y_test, pred_percep))
    print(accuracy_score(y_test, pred_percep), recall_score(y_test, pred_percep), precision_score(y_test, pred_percep),  f1_score


    model2 = DecisionTreeClassifier()
    model2.fit(x_train, y_train)

    pred2 = model2.predict(x_test)
    print("DTC:")
    print(confusion_matrix(y_test, pred2))
    print(accuracy_score(y_test, pred2), recall_score(y_test, pred2), precision_score(y_test, pred2),  f1_score(y_test, pred2))


    rf = RandomForestClassifier(n_estimators=600)
    rf.fit(x_train, y_train)

    pred_rf = rf.predict(x_test)
    print("RF:")
    print(confusion_matrix(y_test, pred_rf))
    print(accuracy_score(y_test, pred_rf), recall_score(y_test, pred_rf), precision_score(y_test, pred_rf),  f1_score(y_test, pre


    gb = GradientBoostingClassifier(n_estimators=700)
    gb.fit(x_train, y_train)

    pred_gb = gb.predict(x_test)
    print("GB:")
    print(confusion_matrix(y_test, pred_gb))
    print(accuracy_score(y_test, pred_gb), recall_score(y_test, pred_gb), precision_score(y_test, pred_gb),  f1_score(y_test, pre


    ab = AdaBoostClassifier(n_estimators=700)
    ab.fit(x_train, y_train)

    pred_ab = ab.predict(x_test)
    print("AB:")
    print(confusion_matrix(y_test, pred_ab))
    print(accuracy_score(y_test, pred_ab), recall_score(y_test, pred_ab), precision_score(y_test, pred_ab),  f1_score(y_test, pre


    xb = xgb.XGBClassifier(n_estimators=700)
    xb.fit(x_train, y_train)

    pred_xb = xb.predict(x_test)
    print("XGB:")
    print(confusion_matrix(y_test, pred_xb))
    print(accuracy_score(y_test, pred_xb), recall_score(y_test, pred_xb), precision_score(y_test, pred_xb),  f1_score(y_test, pre


    n = (int(math.sqrt(x_train.shape[0]))//2)*2+1

    knn = KNeighborsClassifier(n_neighbors=n, weights='distance')
    knn.fit(x_train, y_train)

    pred_knn = knn.predict(x_test)
    print("KNN:")
    print(confusion_matrix(y_test, pred_knn))
    print(accuracy_score(y_test, pred_knn), recall_score(y_test, pred_knn), precision_score(y_test, pred_knn),  f1_score(y_test,


    sv = SVC()
    sv.fit(x_train, y_train)

```

```
    pred_sv = sv.predict(x_test)
    print("SVC:")
    print(confusion_matrix(y_test, pred_sv))
    print(accuracy_score(y_test, pred_sv), recall_score(y_test, pred_sv), precision_score(y_test, pred_sv),  f1_score(y_test, pre
                                                                ...
```

```
import tensorflow as tf
from tensorflow import keras
```

```
tf.random.set_seed(17)
```

```
ann_mod = keras.models.Sequential()
ann_mod.add(keras.layers.Dense(units=128, activation='relu', input_shape=(x_train.shape[1],1)))
ann_mod.add(keras.layers.Dense(units=256, activation='relu'))
ann_mod.add(keras.layers.Dense(units=1, activation='sigmoid'))
```

```
ann_mod.summary()
```

```
ann_mod.compile(loss='binary_crossentropy', optimizer=keras.optimizers.SGD(1e-2), metrics=['accuracy'])
```

```
history = ann_mod.fit(x=x_train, y=y_train, epochs=10, validation_data=(x_test, y_test))
```

```
ann_mod.evaluate(x_test, y_test)
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(15,8))

fpr, tpr, _ = roc_curve(y_test, pred)
auc = round(roc_auc_score(y_test, pred), 4)
plt.plot(fpr,tpr,label="Logistic Regression, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_percep)
auc = round(roc_auc_score(y_test, pred_percep), 4)
plt.plot(fpr,tpr,label="Perceptron, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred2)
auc = round(roc_auc_score(y_test, pred2), 4)
plt.plot(fpr,tpr,label="Decision Tree, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_rf)
auc = round(roc_auc_score(y_test, pred_rf), 4)
plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_gb)
auc = round(roc_auc_score(y_test, pred_gb), 4)
plt.plot(fpr,tpr,label="Gradient Boosting, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_ab)
auc = round(roc_auc_score(y_test, pred_ab), 4)
plt.plot(fpr,tpr,label="Ada Boost, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_xb)
auc = round(roc_auc_score(y_test, pred_xb), 4)
plt.plot(fpr,tpr,label="XG Boost, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_knn)
auc = round(roc_auc_score(y_test, pred_knn), 4)
plt.plot(fpr,tpr,label="KNN, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_sv)
auc = round(roc_auc_score(y_test, pred_sv), 4)
plt.plot(fpr,tpr,label="SVM, AUC="+str(auc))

plt.legend()
```

In [45]:

```
lr_m = [99.41, 99.37, 99.37, 99.37, 99.37]
percep_m = [99.38, 99.28, 99.12, 99.04, 99.17]
dtc_m = [99.24, 99.32, 99.24, 99.19, 99.24]
rf_m = [99.59, 99.58, 99.55, 99.57, 99.57]
gb_m = [99.57, 99.54, 99.54, 99.53, 99.57]
ab_m = [99.42, 99.34, 99.37, 99.38, 99.39]
xgb_m = [99.58, 99.56, 99.56, 99.52, 99.53]
knn_m = [99.04, 99.15, 99.11, 99.05, 99.05]
svc_m = [99.39, 99.41, 99.45, 99.41, 99.35]
```

In [51]:

```
for i in [lr_m, percep_m, dtc_m, rf_m, gb_m, ab_m, xgb_m, knn_m, svc_m]:
    print(np.mean(i), np.std(i))
```

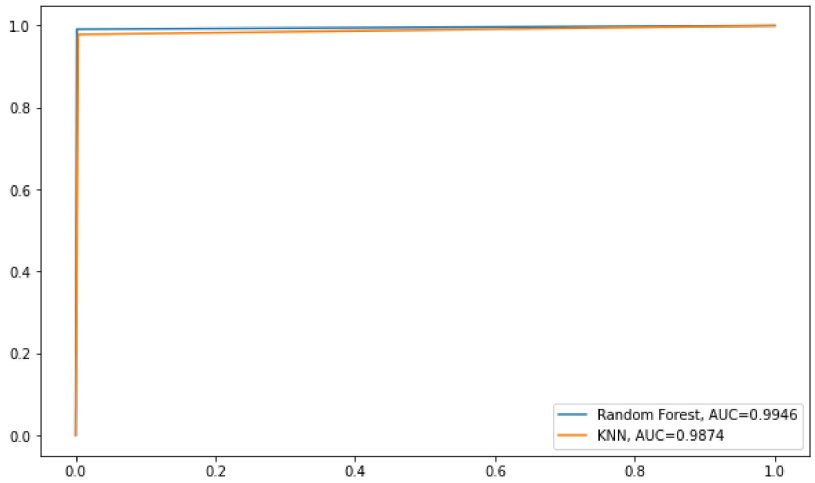
```
99.378 0.01599999999999682
99.19800000000001 0.11973303637676258
99.24600000000001 0.04176122603564083
99.572 0.013266499161423672
99.55000000000001 0.016733200530676486
99.38 0.026076809620809768
99.55 0.021908902300207474
99.08000000000001 0.04289522117905584
99.402 0.032496153618546124
```

In [67]:

```
plt.figure(figsize=(10,6))

fpr, tpr, _ = roc_curve(y_test, pred_rf)
auc = round(roc_auc_score(y_test, pred_rf), 4)
plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))

fpr, tpr, _ = roc_curve(y_test, pred_knn)
auc = round(roc_auc_score(y_test, pred_knn), 4)
plt.plot(fpr,tpr,label="KNN, AUC="+str(auc))

plt.legend()
```

Out[67]:

```
<matplotlib.legend.Legend at 0x26285bbd100>
```



In [62]:

```
rf = RandomForestClassifier(n_estimators=600)
rf.fit(x_train, y_train)

pred_rf = rf.predict(x_test)
print(confusion_matrix(y_test, pred_rf))
print(accuracy_score(y_test, pred_rf), recall_score(y_test, pred_rf), precision_score(y_test, pred_rf),  f1_score(y_test, pred_rf
```

```
[[20711    39]
 [   85  9531]]
0.9959164855430416 0.9911605657237936 0.9959247648902821 0.9935369540289796
```

In [63]:

```
n = (int(math.sqrt(x_train.shape[0]))//2)*2+1

knn = KNeighborsClassifier(n_neighbors=n, weights='distance')
knn.fit(x_train, y_train)

pred_knn = knn.predict(x_test)
print("KNN:")
print(confusion_matrix(y_test, pred_knn))
print(accuracy_score(y_test, pred_knn), recall_score(y_test, pred_knn), precision_score(y_test, pred_knn),  f1_score(y_test, pred
```

```
KNN:
[[20674    76]
 [  207  9409]]
0.9906803661990384 0.9784733777038269 0.991987348444913 0.9851840217789645
```

In [34]:

```
for i in [53, 72, 96, 21]:
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=i, stratify=y)

    ss = StandardScaler()
    ss.fit(x_test)

    x_train = ss.transform(x_train)
    x_test = ss.transform(x_test)

    cb = CatBoostClassifier(n_estimators=1000)
    cb.fit(x_train, y_train)

    pred_cb = cb.predict(x_test)
    print("CB:")
    print(confusion_matrix(y_test, pred_cb))
    print(accuracy_score(y_test, pred_cb), recall_score(y_test, pred_cb), precision_score(y_test, pred_cb),  f1_score(y_test, pre
```

```
Learning rate set to 0.079982
0:      learn: 0.4890607        total: 37.7ms   remaining: 37.7s
1:      learn: 0.3402435        total: 69.7ms   remaining: 34.8s
2:      learn: 0.2465677        total: 101ms    remaining: 33.7s
3:      learn: 0.1835613        total: 132ms    remaining: 32.8s
4:      learn: 0.1410369        total: 162ms    remaining: 32.2s
5:      learn: 0.1095808        total: 190ms    remaining: 31.5s
6:      learn: 0.0893970        total: 220ms    remaining: 31.2s
7:      learn: 0.0739401        total: 252ms    remaining: 31.3s
8:      learn: 0.0624233        total: 284ms    remaining: 31.3s
9:      learn: 0.0547426        total: 314ms    remaining: 31.1s
10:     learn: 0.0489792        total: 345ms    remaining: 31s
11:     learn: 0.0439117        total: 375ms    remaining: 30.9s
12:     learn: 0.0396619        total: 408ms    remaining: 31s
13:     learn: 0.0367419        total: 439ms    remaining: 30.9s
14:     learn: 0.0343143        total: 474ms    remaining: 31.1s
15:     learn: 0.0324322        total: 505ms    remaining: 31s
16:     learn: 0.0309395        total: 535ms    remaining: 30.9s
17:     learn: 0.0299528        total: 564ms    remaining: 30.8s
```

In [36]:

```
cb_m = [99.58, 99.53, 99.57, 99.54, 99.58]
print(np.mean(cb_m), np.std(cb_m))
```

```
99.56 0.020976176963400213
```

In [ ]:

```

```