In [1]:

```
# !pip install python-whois
```

In [1]:

```
import pandas as pd
import itertools
from sklearn.metrics import mean_squared_error,confusion_matrix, precision_score, recall_score, auc,roc_curve
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import random
import math
from collections import Counter
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
import lightgbm as lgb
import os
import socket
import whois
from datetime import datetime
import time
from bs4 import BeautifulSoup
import urllib
import bs4
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:

```
df=pd.read_csv('urldata.csv')
df = df.iloc[: , 1:]
print(df.shape)
df.head()
```

```
(450176, 3)
```

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]:

```
# df['type'].replace("benign","0",inplace=True)
# df['type'].replace("defacement","1",inplace=True)
# df['type'].replace("phishing","1",inplace=True)
# df['type'].replace("malware","1",inplace=True)
# df['type'] = df['type'].astype(str).astype(int)
```

In [4]:

```
df.result.value_counts()
```

Out[4]:

```
0    345738
1    104438
Name: result, dtype: int64
```

Feature Engineering

In [5]:

```
import re
#Use of IP or not in domain
def having_ip_address(url):
    match = re.search(
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5]
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)' # IPv4
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
    if match:
        # print match.group()
        return 1
    else:
        # print 'No matching pattern found'
        return 0
df['use_of_ip'] = df['url'].apply(lambda i: having_ip_address(i))
```

In [6]:

```
# from urllib.parse import urlparse




# def abnormal_url(url):
#     hostname = urlparse(url).hostname
#     hostname = str(hostname)
#     match = re.search(hostname, url)
#     if match:
#         # print match.group()
#         return 1
#     else:
#         # print 'No matching pattern found'
#         return 0


# df['abnormal_url'] = df['url'].apply(lambda i: abnormal_url(i))
```

In [7]:

```
# !pip install googlesearch-python
```

In [8]:

```
df['count.'] = df['url'].apply(lambda i: i.count('.'))
df.head()
```

Out[8]:

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

In [9]:

```
df['count-www'] = df['url'].apply(lambda i: i.count('www'))
df['count@'] = df['url'].apply(lambda i: i.count('@'))
from urllib.parse import urlparse
def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')
df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i))
def no_of_embed(url):
    urldir = urlparse(url).path
    return urldir.count('//')
df['count_embed_domian'] = df['url'].apply(lambda i: no_of_embed(i))
def shortening_service(url):
    match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'
                      'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'
                      'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us
                      'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
                      'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
                      'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.
                      'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.m
                      'tr\.im|link\.zip\.net',
                      url)
    if match:
        return 1
    else:
        return 0
df['short_url'] = df['url'].apply(lambda i: shortening_service(i))
```

In [10]:

```
df['count-https'] = df['url'].apply(lambda i : i.count('https'))
df['count-http'] = df['url'].apply(lambda i : i.count('http'))
```

In [11]:

```
df['count%'] = df['url'].apply(lambda i: i.count('%'))
df['count?'] = df['url'].apply(lambda i: i.count('?'))
df['count-'] = df['url'].apply(lambda i: i.count('-'))
df['count='] = df['url'].apply(lambda i: i.count('='))
#Length of URL
df['url_length'] = df['url'].apply(lambda i: len(str(i)))
#Hostname Length
df['hostname_length'] = df['url'].apply(lambda i: len(urlparse(i).netloc))

df.head()
```

Out[11]:

	url	label	result	use_of_ip	count.	count-www	count@	count_dir	count_embed_domian	short_url	count-https	count-http	cou
0	https://www.google.com	benign	0	0	2	1	0	0	0	0	1	1	
1	https://www.youtube.com	benign	0	0	2	1	0	0	0	0	1	1	
2	https://www.facebook.com	benign	0	0	2	1	0	0	0	0	1	1	
3	https://www.baidu.com	benign	0	0	2	1	0	0	0	0	1	1	
4	https://www.wikipedia.org	benign	0	0	2	1	0	0	0	0	1	1	

In [12]:

```
def suspicious_words(url):
    match = re.search('PayPal|login|signin|bank|account|update|free|lucky|service|bonus|ebayisapi|webscr',
                      url)
    if match:
        return 1
    else:
        return 0
df['sus_url'] = df['url'].apply(lambda i: suspicious_words(i))
```

In [13]:

```
df.head()
```

Out[13]:

	url	label	result	use_of_ip	count.	count-www	count@	count_dir	count_embed_domian	short_url	count-https	count-http	cou
0	https://www.google.com	benign	0	0	2	1	0	0	0	0	1	1	
1	https://www.youtube.com	benign	0	0	2	1	0	0	0	0	1	1	
2	https://www.facebook.com	benign	0	0	2	1	0	0	0	0	1	1	
3	https://www.baidu.com	benign	0	0	2	1	0	0	0	0	1	1	
4	https://www.wikipedia.org	benign	0	0	2	1	0	0	0	0	1	1	

In [14]:

```
# !pip install tld
```

In [15]:

```
#Importing dependencies
from urllib.parse import urlparse
from tld import get_tld
import os.path

#First Directory Length
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0

df['fd_length'] = df['url'].apply(lambda i: fd_length(i))

#Length of Top Level Domain
df['tld'] = df['url'].apply(lambda i: get_tld(i,fail_silently=True))
def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1

df['tld_length'] = df['tld'].apply(lambda i: tld_length(i))
```

In [16]:

```
def digit_count(url):
    digits = 0
    for i in url:
        if i.isnumeric():
            digits = digits + 1
    return digits
df['count-digits']= df['url'].apply(lambda i: digit_count(i))
```

In [17]:

```
def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters
df['count-letters']= df['url'].apply(lambda i: letter_count(i))
```

In [18]:

```
df = df.drop("tld",1)
```

```
C:\Users\nitis\AppData\Local\Temp\ipykernel_12196\2551734815.py:1: FutureWarning: In a future version of pandas
all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  df = df.drop("tld",1)
```

In [19]:

```
# from sklearn.preprocessing import LabelEncoder

# lb_make = LabelEncoder()
# df["type_code"] = lb_make.fit_transform(df["type"])
# df["type_code"].value_counts()
```

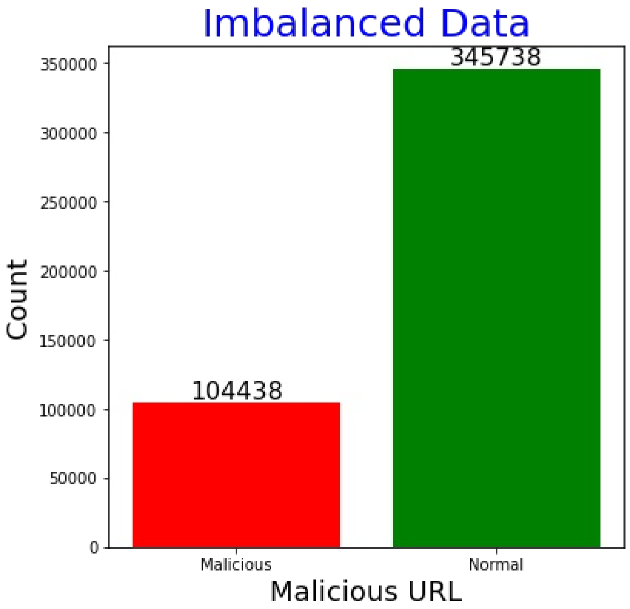
In [20]:

```
#Predictor Variables
X = df[['use_of_ip', 'count.', 'count-www', 'count@',
       'count_dir', 'count_embed_domian', 'short_url', 'count-https',
       'count-http', 'count%', 'count?', 'count-', 'count=', 'url_length',
       'hostname_length', 'sus_url', 'fd_length', 'tld_length', 'count-digits',
       'count-letters']]

#Target Variable
y = df['result']
```

In [21]:

```
plt.figure(figsize=(6,6))
names = ["Malicious", "Normal"]
```

```
count = [(df.result.values == 1).sum(), (df.result.values == 0).sum()]
plt.bar(names, count, color = ["Red", "Green"])
plt.title('Imbalanced Data', color = 'blue', fontsize= 25)
plt.xlabel('Malicious URL', fontsize= 18)
plt.ylabel('Count', fontsize= 18)
# plt.ylim(0,290)
for i in range(len(names)):
    plt.text(i, count[i], count[i], ha='center', va='bottom', fontsize=16)
plt.show()
```

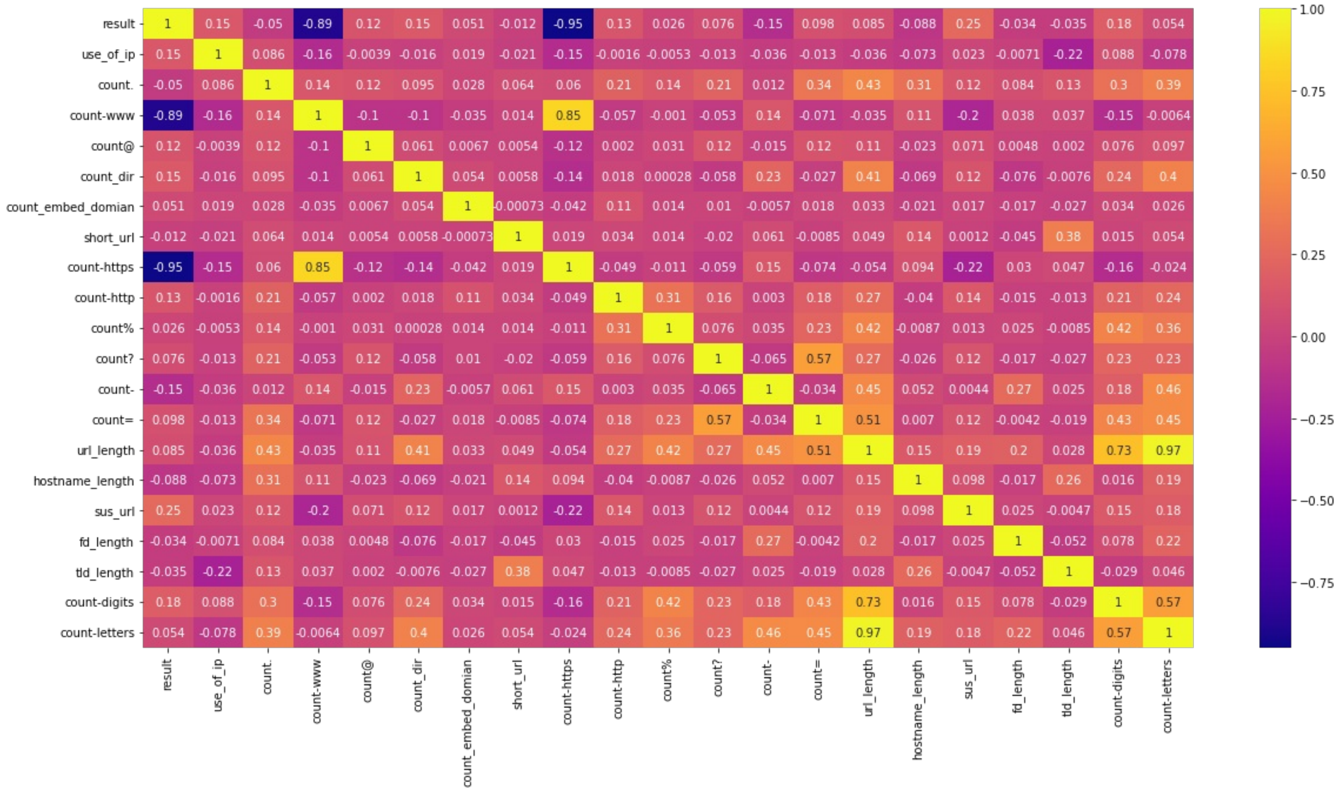


In [22]:

```
import seaborn as sns
plt.figure(figsize = (20, 10))
sns.heatmap(df.corr(), annot = True, cmap="plasma")
```

Out[22]:

```
<AxesSubplot:>
```



In [23]:

```
# from imblearn.over_sampling import RandomOverSampler
# from collections import Counter
# os = RandomOverSampler(0.9)
# x_os, y_os = os.fit_resample(X,y)
# print("Before fit {}".format(Counter(y)))
# print("After fit {}".format(Counter(y_os)))
```

In [24]:

```
# plt.figure(figsize=(6,6))
# names = ["Malicious", "Normal"]
# count = [385292, 428103]
# plt.bar(names, count, color = ["Red", "Green"])
# plt.title('Balanced Data', color = 'blue', fontsize= 25)
# plt.xlabel('Malicious URL', fontsize= 18)
# plt.ylabel('Count', fontsize= 18)
# # plt.ylim(0,290)
# for i in range(len(names)):
#     plt.text(i, count[i], count[i], ha='center', va='bottom', fontsize=16)
# plt.show()
```

ML Models Training

In [25]:

```
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.ensemble import VotingClassifier

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import mutual_info_classif
```

Evaluating Models

In [26]:

```
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.25)
```

In [27]:

```
# Support Vector Machine (SVM)
svm = SVC(C= 10, gamma= 0.01, kernel= 'rbf').fit(x_train, y_train)

# Random Forest (RF)
RF = RandomForestClassifier().fit(x_train, y_train)

# Logistic Regression (LR)
LR = LogisticRegression(max_iter=3000,C= 1.0, penalty= 'l2', solver= 'liblinear').fit(x_train, y_train)

# # AdaBoostClassifier (AB)
# AB = AdaBoostClassifier().fit(x_train, y_train)

# # Multi-layer perceptron (MLP)
# MLP = MLPClassifier().fit(x_train, y_train)

# # Voting Classifier (VC)
# est = [('svm',svm), ('lr',LR)]
# VC = VotingClassifier(estimators = est, voting ='hard').fit(x_train, y_train)
```

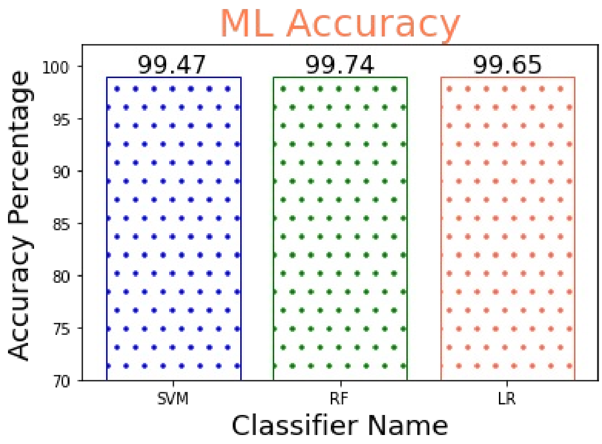
In [28]:

```
# Support Vector Machine (SVM)
svm_pre = svm.predict(x_test)
svm_sc = svm.score(x_test, y_test) * 100
svm_sc = "{:.2f}".format(svm_sc)

# Random Forest (RF)
rf_pre = RF.predict(x_test)
RF_sc = RF.score(x_test, y_test) * 100
RF_sc = "{:.2f}".format(RF_sc)

# Logistic Regression (LR)
lr_pre = LR.predict(x_test)
LR_sc = LR.score(x_test, y_test) * 100
LR_sc = "{:.2f}".format(LR_sc)

# # AdaBoostClassifier (AB)
# ab_pre = AB.predict(x_test)
# AB_sc = AB.score(x_test, y_test) * 100
# AB_sc = "{:.2f}".format(AB_sc)

# # Multi-layer perceptron (MLP)
# mlp_pre = MLP.predict(x_test)
# MLP_sc = MLP.score(x_test, y_test) * 100
# MLP_sc = "{:.2f}".format(MLP_sc)


# # Voting Classifier (VC)
# vc_pre = VC.predict(x_test)
# vc_sc = VC.score(x_test, y_test) * 100
# vc_sc = "{:.2f}".format(vc_sc)
```

Accuracy

In [44]:

```
algo = ['SVM', 'RF', 'LR']
acc = [math.floor(int(float(svm_sc))),math.floor(int(float(RF_sc))),math.floor(int(float(LR_sc)))]
pert = [svm_sc, RF_sc, LR_sc]
colors = ['blue', 'green', 'tomato']
plt.bar(algo, acc,color='white', edgecolor=colors, hatch='.')
plt.title('ML Accuracy', color = 'coral', fontsize= 25)
plt.xlabel('Classifier Name', fontsize= 18)
plt.ylabel('Accuracy Percentage', fontsize= 18)
plt.ylim(70,102)
for i in range(len(algo)):
    plt.text(i, acc[i], pert[i], ha='center', va='bottom', fontsize=16)
plt.show()
```



In [45]:

```
pd.DataFrame(classification_report(y_test, svm_pre, output_dict=True))
```

Out[45]:

	0	1	accuracy	macro avg	weighted avg
precision	0.996487	0.988887	0.994731	0.992687	0.994730
recall	0.996660	0.988317	0.994731	0.992488	0.994731
f1-score	0.996573	0.988602	0.994731	0.992588	0.994730
support	86523.000000	26021.000000	0.994731	112544.000000	112544.000000

In [46]:

```
pd.DataFrame(classification_report(y_test, rf_pre, output_dict=True))
```

Out[46]:

	0	1	accuracy	macro avg	weighted avg
precision	0.997668	0.996488	0.997397	0.997078	0.997395
recall	0.998948	0.992237	0.997397	0.995593	0.997397
f1-score	0.998308	0.994358	0.997397	0.996333	0.997395
support	86523.000000	26021.000000	0.997397	112544.000000	112544.000000

In [47]:

```
pd.DataFrame(classification_report(y_test, lr_pre, output_dict=True))
```

Out[47]:

	0	1	accuracy	macro avg	weighted avg
precision	0.997735	0.992582	0.996544	0.995158	0.996543
recall	0.997769	0.992468	0.996544	0.995119	0.996544
f1-score	0.997752	0.992525	0.996544	0.995138	0.996544
support	86523.000000	26021.000000	0.996544	112544.000000	112544.000000

In [48]:

```
# pd.DataFrame(classification_report(y_test, ab_pre, output_dict=True))
```

In [49]:

```
# pd.DataFrame(classification_report(y_test, mlp_pre, output_dict=True))
```

In [50]:

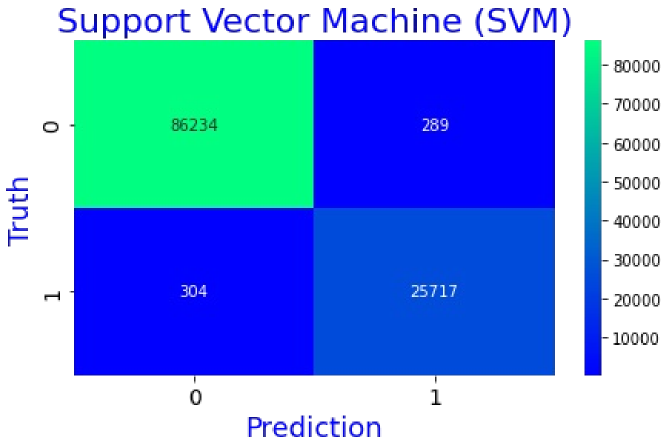
```
# pd.DataFrame(classification_report(y_test, vc_pre, output_dict=True))
```

In [51]:

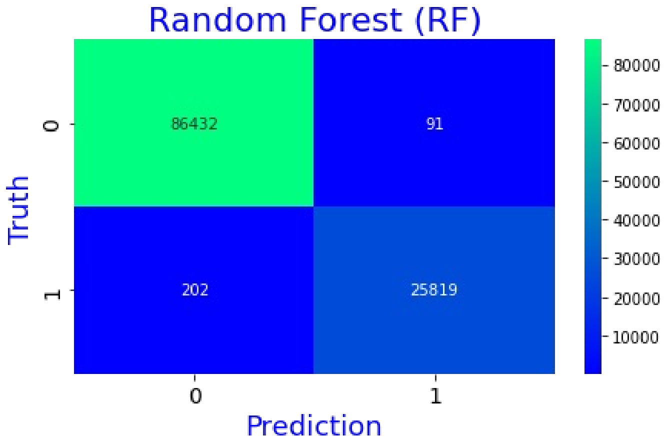
```
def print_confusion_matrix(confusion_matrix, class_names, figsize = (7,4), fontsize=14, title = "Confusion Matr
```

In [51]:
```
def print_confusion_matrix(confusion_matrix, class_names, figsize = (7,4), fontsize=14, title = "Confusion Mat
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d", cmap='winter')
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=90, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0, ha='center', fontsize=fontsize)
    plt.ylabel('Truth', fontsize=18, color='blue')
    plt.xlabel('Prediction',  fontsize=18, color='blue')
    plt.title(title, fontsize=22, color='blue')
```

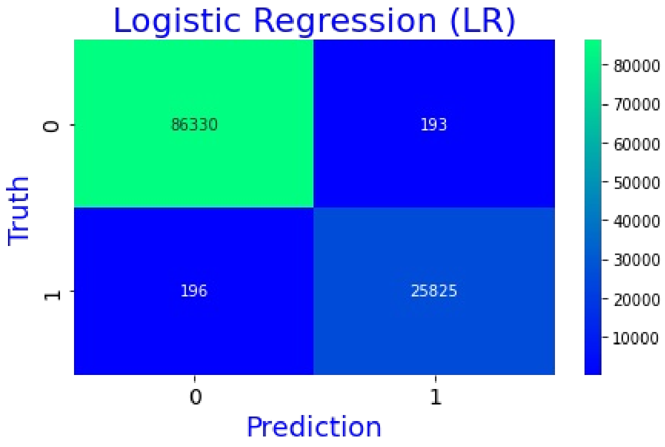
In [52]:
```
cm = confusion_matrix(y_test, svm_pre)
print_confusion_matrix(cm,["0", "1"], title="Support Vector Machine (SVM)")
```



In [53]:
```
cm = confusion_matrix(y_test, rf_pre)
print_confusion_matrix(cm,["0", "1"], title="Random Forest (RF)")
```



In [54]:
```
cm = confusion_matrix(y_test, lr_pre)
print_confusion_matrix(cm,["0", "1"], title="Logistic Regression (LR)")
```



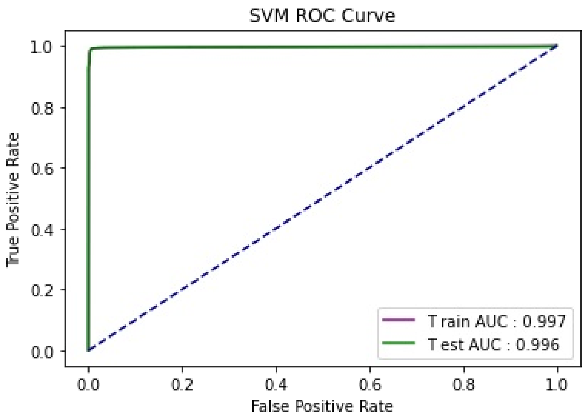
In [55]:
```
# cm = confusion_matrix(y_test, ab_pre)
# print_confusion_matrix(cm,["0", "1"], title="AdaBoost (AB)")
```

In [56]:
```
# cm = confusion_matrix(y_test, mlp_pre)
# print_confusion_matrix(cm,["0", "1"], title="Multilayer Perceptron (MLP)")
```

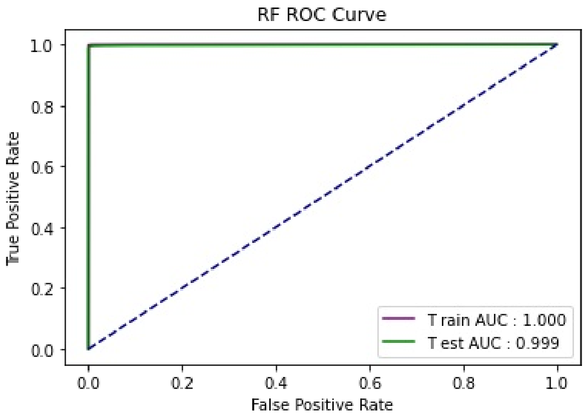
In [57]:
```
# cm = confusion_matrix(y_test, vc_pre)
# print_confusion_matrix(cm,["0", "1"], title="Voting Classifier (VC)")
```

In [27]:
```
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
```

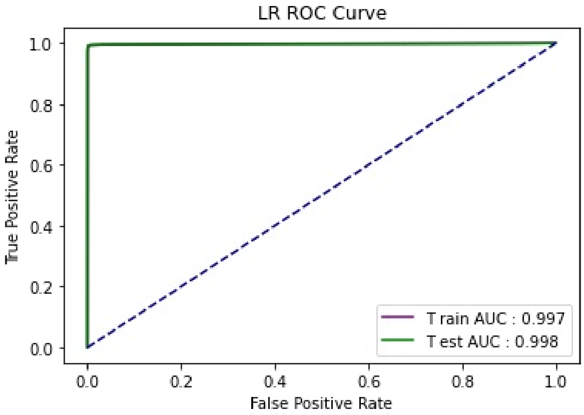
In [28]:
```
svm = SVC(probability=True).fit(x_train, y_train)
svm_probs_tr = svm.predict_proba(x_train)[:,1]
auc_tr = roc_auc_score(y_train, svm_probs_tr)
fpr_tr, tpr_tr, thresholds = roc_curve(y_train, svm_probs_tr)

svm_probs_ts = svm.predict_proba(x_test)[:,1]
auc_ts = roc_auc_score(y_test, svm_probs_ts)
fpr_ts, tpr_ts, thresholds = roc_curve(y_test, svm_probs_ts)

plt.plot(fpr_tr,tpr_tr, color='purple', label="T rain AUC : {:.3f}".format(auc_tr))
plt.plot(fpr_ts,tpr_ts, color='green', label="T est AUC : {:.3f}".format(auc_ts))
plt.plot([0,1], [0,1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('SVM ROC Curve')
plt.legend()
plt.show()
```



In [30]:
```
RF = RandomForestClassifier().fit(x_train, y_train)
rf_probs_tr = RF.predict_proba(x_train)[:,1]
auc_tr = roc_auc_score(y_train, rf_probs_tr)
fpr_tr, tpr_tr, thresholds = roc_curve(y_train, rf_probs_tr)

rf_probs_ts = RF.predict_proba(x_test)[:,1]
auc_ts = roc_auc_score(y_test, rf_probs_ts)
fpr_ts, tpr_ts, thresholds = roc_curve(y_test, rf_probs_ts)

plt.plot(fpr_tr,tpr_tr, color='purple', label="T rain AUC : {:.3f}".format(auc_tr))
plt.plot(fpr_ts,tpr_ts, color='green', label="T est AUC : {:.3f}".format(auc_ts))
plt.plot([0,1], [0,1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('RF ROC Curve')
plt.legend()
plt.show()
```



In [31]:
```
LR = LogisticRegression(max_iter=3000,C= 1.0, penalty= 'l2', solver= 'liblinear').fit(x_train, y_train)
LR_probs_tr = LR.predict_proba(x_train)[:,1]
auc_tr = roc_auc_score(y_train, LR_probs_tr)
fpr_tr, tpr_tr, thresholds = roc_curve(y_train, LR_probs_tr)

LR_probs_ts = LR.predict_proba(x_test)[:,1]
auc_ts = roc_auc_score(y_test, LR_probs_ts)
fpr_ts, tpr_ts, thresholds = roc_curve(y_test, LR_probs_ts)

plt.plot(fpr_tr,tpr_tr, color='purple', label="T rain AUC : {:.3f}".format(auc_tr))
plt.plot(fpr_ts,tpr_ts, color='green', label="T est AUC : {:.3f}".format(auc_ts))
plt.plot([0,1], [0,1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LR ROC Curve')
plt.legend()
plt.show()
```



In [32]:

```
# AB_probs_tr = AB.predict_proba(x_train)[:,1]
# auc_tr = roc_auc_score(y_train, AB_probs_tr)
# fpr_tr, tpr_tr, thresholds = roc_curve(y_train, AB_probs_tr)

# AB_probs_ts = AB.predict_proba(x_test)[:,1]
# auc_ts = roc_auc_score(y_test, AB_probs_ts)
# fpr_ts, tpr_ts, thresholds = roc_curve(y_test, AB_probs_ts)

# plt.plot(fpr_tr,tpr_tr, color='purple', label="T rain AUC : {:.3f}".format(auc_tr))
# plt.plot(fpr_ts,tpr_ts, color='green', label="T est AUC : {:.3f}".format(auc_ts))
# plt.plot([0,1], [0,1], color='darkblue', linestyle='--')
# plt.xlabel('False Positive Rate')
# plt.ylabel('True Positive Rate')
# plt.title('AB ROC Curve')
# plt.legend()
# plt.show()
```

In [33]:

```
# MLP_probs_tr = MLP.predict_proba(x_train)[:,1]
# auc_tr = roc_auc_score(y_train, MLP_probs_tr)
# fpr_tr, tpr_tr, thresholds = roc_curve(y_train, MLP_probs_tr)

# MLP_probs_ts = MLP.predict_proba(x_test)[:,1]
# auc_ts = roc_auc_score(y_test, MLP_probs_ts)
# fpr_ts, tpr_ts, thresholds = roc_curve(y_test, MLP_probs_ts)

# plt.plot(fpr_tr,tpr_tr, color='purple', label="T rain AUC : {:.3f}".format(auc_tr))
# plt.plot(fpr_ts,tpr_ts, color='green', label="T est AUC : {:.3f}".format(auc_ts))
# plt.plot([0,1], [0,1], color='darkblue', linestyle='--')
# plt.xlabel('False Positive Rate')
# plt.ylabel('True Positive Rate')
# plt.title('MLP ROC Curve')
# plt.legend()
# plt.show()
```

In [34]:

```
# est = [('svm',svm), ('lr',LR)]
# vc = VotingClassifier(estimators = est, voting='soft').fit(x_train, y_train)

# vc_probs_tr = vc.predict_proba(x_train)[:,1]
# auc_tr = roc_auc_score(y_train, vc_probs_tr)
# fpr_tr, tpr_tr, thresholds = roc_curve(y_train, vc_probs_tr)

# vc_probs_ts = vc.predict_proba(x_test)[:,1]
# auc_ts = roc_auc_score(y_test, vc_probs_ts)
# fpr_ts, tpr_ts, thresholds = roc_curve(y_test, vc_probs_ts)

# plt.plot(fpr_tr,tpr_tr, color='purple', label="T rain AUC : {:.3f}".format(auc_tr))
# plt.plot(fpr_ts,tpr_ts, color='green', label="T est AUC : {:.3f}".format(auc_ts))
# plt.plot([0,1], [0,1], color='darkblue', linestyle='--')
# plt.xlabel('False Positive Rate')
# plt.ylabel('True Positive Rate')
# plt.title('VC ROC Curve')
# plt.legend()
# plt.show()
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js