```
import warnings
warnings.filterwarnings('ignore')


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
pd.options.display.float_format = '{:.2f}'.format
```

API + Permission

```
data = pd.read_csv('Dataset/Drebin/api+permdrebin.csv')
data.head()
```

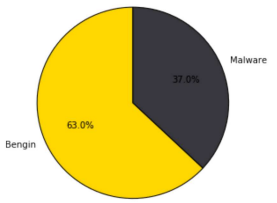
	SEND_SMS	READ_PHONE_STATE	GET_ACCOUNTS	RECEIVE_SMS	READ_SMS	USE_CREDENTIALS	MANAGE_ACCOUNTS	WRITE_SMS	READ_SYNC_SETTINGS	AUTH
0	1	1	0	0	0	0	0	0	0	
1	1	1	0	1	1	0	0	0	0	
2	1	1	0	0	0	0	0	0	0	
3	0	1	0	0	1	0	0	1	0	
4	0	1	0	0	0	0	0	0	0	

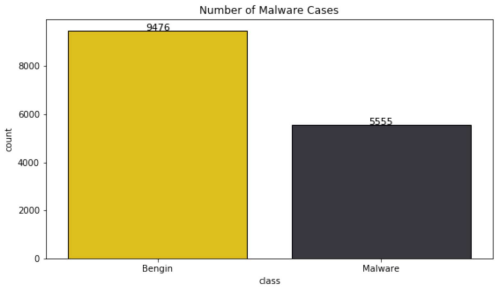
5 rows × 182 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15031 entries, 0 to 15030
Columns: 182 entries, SEND_SMS to class
dtypes: int64(182)
memory usage: 20.9 MB
```

```
data = data.dropna()


fraud = len(data[data['class'] == 1]) / len(data) * 100
nofraud = len(data[data['class'] == 0]) / len(data) * 100
fraud_percentage = [nofraud,fraud]

colors = ['#FFD700','#3B3B3C']

fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (20,5))
plt.subplot(1,2,1)
plt.pie(fraud_percentage,labels = ['Bengin','Malware'],autopct='%1.1f%%',startangle = 90,colors = colors,
       wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})

plt.subplot(1,2,2)
ax = sns.countplot(x='class',data = data,edgecolor = 'black',palette = colors)
for rect in ax.patches:
    ax.text(rect.get_x() + rect.get_width() / 2, rect.get_height() + 2, rect.get_height(), horizontalalignment='center', fontsize = 11)
ax.set_xticklabels(['Bengin','Malware'])
plt.title('Number of Malware Cases');
```






```
X = data.drop('class', axis=1)
y = data['class']


from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
model = LogisticRegression()

rfe = RFE(model, step=2)
rfe = rfe.fit(X, y)

# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))
```

```
Selected features: ['SEND_SMS', 'READ_PHONE_STATE', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'WRITE_SMS', 'WRIT
```

```
X_new = data[['SEND_SMS', 'READ_PHONE_STATE', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'WRITE_SMS', 'WRITE_HISTORY_


rfe = rfe.fit(X_new, y)

# summarize the selection of the attributes
print('Selected features: %s' % list(X_new.columns[rfe.support_]))
```

```
Selected features: ['SEND_SMS', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'READ_HISTORY_BOOKMARKS', 'INTERNET',
```

```
X1 = data[['SEND_SMS', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'READ_HISTORY_BOOKMARKS', 'INTERNET', 'NFC', 'ACCES
rfe = rfe.fit(X1, y)

# summarize the selection of the attributes
print('Selected features: %s' % list(X1.columns[rfe.support_]))
```

```
Selected features: ['SEND_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'READ_HISTORY_BOOKMARKS', 'NFC', 'ACCESS_LOCATION_EXTRA_COMMANDS',
```

```
X_final = data[['SEND_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'READ_HISTORY_BOOKMARKS', 'NFC', 'ACCESS_LOCATION_EXTRA_COMMANDS', 'BIND_R


X_final
```

	SEND_SMS	USE_CREDENTIALS	MANAGE_ACCOUNTS	READ_HISTORY_BOOKMARKS	NFC	ACCESS_L
0	1	0	0	0	0	
1	1	0	0	0	0	
2	1	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	1	0	
...	...	...	...	...	...	
15026	0	0	0	0	0	
15027	0	0	0	0	0	
15028	0	0	0	0	0	
15029	0	1	0	0	0	
15030	0	0	0	0	0	

15031 rows × 22 columns

```
df = pd.DataFrame(data, columns=['SEND_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'READ_HISTORY_BOOKMARKS', 'NFC', 'ACCESS_LOCATION_EXTRA_C
```

```
df.to_csv('debrin.csv')
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_final, y, test_size=0.2) # keeping 15% for test
```

```
ML_Model = []
accuracy = []
precision = []
recall = []
f1score = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
    ML_Model.append(model)
    accuracy.append(round(a, 3))
    precision.append(round(b, 3))
    recall.append(round(c, 3))
    f1score.append(round(d, 3))
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
log = LogisticRegression()

# fit the model
log.fit(X_train,y_train)
y_pred = log.predict(X_test)

lr_acc_a = accuracy_score(y_test,y_pred)
lr_prec_a = precision_score(y_test,y_pred)
lr_rec_a = recall_score(y_test,y_pred)
lr_f1_a = f1_score(y_test,y_pred)
```

```
storeResults('API+Permission : LR',lr_acc_a,lr_prec_a,lr_rec_a,lr_f1_a)
```

SVM

```
# Support Vector Classifier model
from sklearn.svm import SVC
svc = SVC()

# fit the model
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)

svc_acc_a = accuracy_score(y_test,y_pred)
svc_prec_a = precision_score(y_test,y_pred)
svc_rec_a = recall_score(y_test,y_pred)
svc_f1_a = f1_score(y_test,y_pred)


storeResults('API+Permission : SVC',svc_acc_a,svc_prec_a,svc_rec_a,svc_f1_a)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)

# fit the model
neigh.fit(X_train,y_train)
y_pred = neigh.predict(X_test)

knn_acc_a = accuracy_score(y_test,y_pred)
knn_prec_a = precision_score(y_test,y_pred)
knn_rec_a = recall_score(y_test,y_pred)
knn_f1_a = f1_score(y_test,y_pred)


storeResults('API+Permission : KNN',knn_acc_a,knn_prec_a,knn_rec_a,knn_f1_a)
```

Random Forest

```
# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(n_estimators=10)

# fit the model
forest.fit(X_train,y_train)
y_pred = forest.predict(X_test)

rf_acc_a = accuracy_score(y_test,y_pred)
rf_prec_a = precision_score(y_test,y_pred)
rf_rec_a = recall_score(y_test,y_pred)
rf_f1_a = f1_score(y_test,y_pred)


storeResults('API+Permission : RF',rf_acc_a,rf_prec_a,rf_rec_a,rf_f1_a)
```

Decision Tree

```
# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)

# fit the model
tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

dt_acc_a = accuracy_score(y_test,y_pred)
dt_prec_a = precision_score(y_test,y_pred)
dt_rec_a = recall_score(y_test,y_pred)
dt_f1_a = f1_score(y_test,y_pred)


storeResults('API+Permission : DT',dt_acc_a,dt_prec_a,dt_rec_a,dt_f1_a)
```

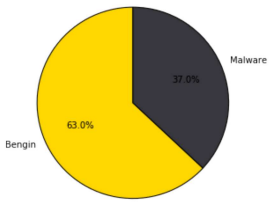
Only API

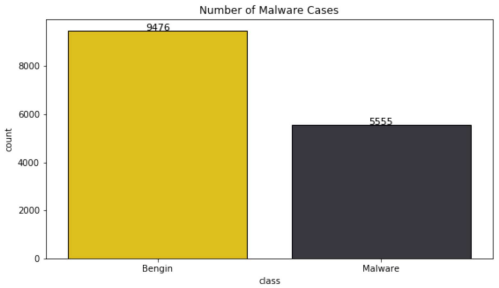
```
data = pd.read_csv('Dataset/Drebin/onlyapidrebin.csv')
data.head()
```

	transact	onServiceConnected	bindService	attachInterface	ServiceConnection	androi
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	

5 rows × 73 columns

```
data = data.dropna()


fraud = len(data[data['class'] == 1]) / len(data) * 100
nofraud = len(data[data['class'] == 0]) / len(data) * 100
fraud_percentage = [nofraud,fraud]

colors = ['#FFD700','#3B3B3C']

fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (20,5))
plt.subplot(1,2,1)
plt.pie(fraud_percentage,labels = ['Bengin','Malware'],autopct='%1.1f%%',startangle = 90,colors = colors,
       wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})

plt.subplot(1,2,2)
ax = sns.countplot(x='class',data = data,edgecolor = 'black',palette = colors)
for rect in ax.patches:
    ax.text(rect.get_x() + rect.get_width() / 2, rect.get_height() + 2, rect.get_height(), horizontalalignment='center', fontsize = 11)
ax.set_xticklabels(['Bengin','Malware'])
plt.title('Number of Malware Cases');
```






```
X = data.drop('class', axis=1)
y = data['class']


from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
model = LogisticRegression()

rfe = RFE(model, step=2)
rfe = rfe.fit(X, y)

# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))
```

```
Selected features: ['transact', 'onServiceConnected', 'Ljava.lang.Class.getCanonicalName', 'Ljava.lang.Class.getMethods', 'Ljava.net.URL
```

```
X_1 = data[['transact', 'onServiceConnected', 'Ljava.lang.Class.getCanonicalName', 'Ljava.lang.Class.getMethods', 'Ljava.net.URLDecoder', 'a


rfe = rfe.fit(X_1, y)

# summarize the selection of the attributes
print('Selected features: %s' % list(X_1.columns[rfe.support_]))
```

```
Selected features: ['transact', 'Ljava.lang.Class.getCanonicalName', 'Ljava.lang.Class.getMethods', 'android.telephony.SmsManager', 'get
```

```
X_1
```

	transact	onServiceConnected	Ljava.lang.Class.getCanonicalName	Ljava.lang.Clas
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	
...	...	...	...	
15026	1	1	1	
15027	0	0	0	
15028	0	0	0	
15029	1	1	1	
15030	1	1	1	

15031 rows × 36 columns

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_1, y, test_size=0.2) # keeping 15% for test
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
log1 = LogisticRegression()

# fit the model
log1.fit(X_train,y_train)
y_pred = log1.predict(X_test)

lr_acc_b = accuracy_score(y_test,y_pred)
lr_prec_b = precision_score(y_test,y_pred)
lr_rec_b = recall_score(y_test,y_pred)
lr_f1_b = f1_score(y_test,y_pred)


storeResults('API : LR',lr_acc_b,lr_prec_b,lr_rec_b,lr_f1_b)
```

SVC

```
# Support Vector Classifier model
from sklearn.svm import SVC
svc1 = SVC()

# fit the model
svc1.fit(X_train,y_train)
y_pred = svc1.predict(X_test)

svc_acc_b = accuracy_score(y_test,y_pred)
svc_prec_b = precision_score(y_test,y_pred)
svc_rec_b = recall_score(y_test,y_pred)
svc_f1_b = f1_score(y_test,y_pred)


storeResults('API : SVC',svc_acc_b,svc_prec_b,svc_rec_b,svc_f1_b)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
neigh1 = KNeighborsClassifier(n_neighbors=3)

# fit the model
neigh1.fit(X_train,y_train)
y_pred = neigh1.predict(X_test)

knn_acc_b = accuracy_score(y_test,y_pred)
knn_prec_b = precision_score(y_test,y_pred)
knn_rec_b = recall_score(y_test,y_pred)
knn_f1_b = f1_score(y_test,y_pred)


storeResults('API : KNN',knn_acc_b,knn_prec_b,knn_rec_b,knn_f1_b)
```

Random FOrest

```
# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest1 = RandomForestClassifier(n_estimators=10)

# fit the model
forest1.fit(X_train,y_train)
y_pred = forest1.predict(X_test)

rf_acc_b = accuracy_score(y_test,y_pred)
rf_prec_b = precision_score(y_test,y_pred)
rf_rec_b = recall_score(y_test,y_pred)
rf_f1_b = f1_score(y_test,y_pred)


storeResults('API : RF',rf_acc_b,rf_prec_b,rf_rec_b,rf_f1_b)
```

Decision Tree

```
# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree1 = DecisionTreeClassifier(max_depth=30)

# fit the model
tree1.fit(X_train, y_train)

y_pred = tree1.predict(X_test)

dt_acc_b = accuracy_score(y_test,y_pred)
dt_prec_b = precision_score(y_test,y_pred)
dt_rec_b = recall_score(y_test,y_pred)
dt_f1_b = f1_score(y_test,y_pred)


storeResults('API : DT',dt_acc_b,dt_prec_b,dt_rec_b,dt_f1_b)
```

Only Permission

```
data = pd.read_csv('Dataset/Drebin/onlypermissionsdrebin.csv')
data.head()
```

	SEND_SMS	READ_PHONE_STATE	GET_ACCOUNTS	RECEIVE_SMS	READ_SMS	USE_CREDENTIALS	MAN
0	1	1	0	0	0	0	
1	1	1	0	1	1	0	
2	1	1	0	0	0	0	
3	0	1	0	0	1	0	
4	0	1	0	0	0	0	

5 rows × 110 columns

```
data = data.dropna()


X = data.drop('classs', axis=1)
y = data['classs']
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
model = LogisticRegression()

rfe = RFE(model, step=2)
rfe = rfe.fit(X, y)

# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))
```

```
Selected features: ['SEND_SMS', 'READ_PHONE_STATE', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'AUTHENTICATE_ACCC
```

```
X_1 = data[['SEND_SMS', 'READ_PHONE_STATE', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'AUTHENTICATE_ACCOUNTS', 'WRIT


rfe = rfe.fit(X_1, y)

# summarize the selection of the attributes
print('Selected features: %s' % list(X_1.columns[rfe.support_]))
```

```
Selected features: ['SEND_SMS', 'READ_PHONE_STATE', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'WRITE_HISTORY_BOC
```

```
X_final = data[['SEND_SMS', 'READ_PHONE_STATE', 'GET_ACCOUNTS', 'READ_SMS', 'USE_CREDENTIALS', 'MANAGE_ACCOUNTS', 'WRITE_HISTORY_BOOKMARKS',


X_final
```

	SEND_SMS	READ_PHONE_STATE	GET_ACCOUNTS	READ_SMS	USE_CREDENTIALS	MANAGE_ACCOU
0	1	1	0	0	0	
1	1	1	0	1	0	
2	1	1	0	0	0	
3	0	1	0	1	0	
4	0	1	0	0	0	
...	...	...	...	...	...	
15026	0	1	0	0	0	
15027	0	0	0	0	0	
15028	0	1	0	0	0	
15029	0	0	1	0	1	
15030	0	0	0	0	0	

15031 rows × 27 columns

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_final, y, test_size=0.2) # keeping 15% for test
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
log2 = LogisticRegression()

# fit the model
log2.fit(X_train,y_train)
y_pred = log2.predict(X_test)

lr_acc_c = accuracy_score(y_test,y_pred)
lr_prec_c = precision_score(y_test,y_pred)
lr_rec_c = recall_score(y_test,y_pred)
lr_f1_c = f1_score(y_test,y_pred)
```

```
storeResults('Premission : LR',lr_acc_c,lr_prec_c,lr_rec_c,lr_f1_c)
```

SVC

```
# Support Vector Classifier model
from sklearn.svm import SVC
svc2 = SVC()

# fit the model
svc2.fit(X_train,y_train)
y_pred = svc2.predict(X_test)

svc_acc_c = accuracy_score(y_test,y_pred)
svc_prec_c = precision_score(y_test,y_pred)
svc_rec_c = recall_score(y_test,y_pred)
svc_f1_c = f1_score(y_test,y_pred)


storeResults('Premission : SVC',svc_acc_c,svc_prec_c,svc_rec_c,svc_f1_c)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
neigh2 = KNeighborsClassifier(n_neighbors=3)

# fit the model
neigh2.fit(X_train,y_train)
y_pred = neigh2.predict(X_test)

knn_acc_c = accuracy_score(y_test,y_pred)
knn_prec_c = precision_score(y_test,y_pred)
knn_rec_c = recall_score(y_test,y_pred)
knn_f1_c = f1_score(y_test,y_pred)


storeResults('Premission : KNN',knn_acc_c,knn_prec_c,knn_rec_c,knn_f1_c)
```

Random FOrest

```
# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest2 = RandomForestClassifier(n_estimators=10)

# fit the model
forest2.fit(X_train,y_train)
y_pred = forest2.predict(X_test)

rf_acc_c = accuracy_score(y_test,y_pred)
rf_prec_c = precision_score(y_test,y_pred)
rf_rec_c = recall_score(y_test,y_pred)
rf_f1_c = f1_score(y_test,y_pred)


storeResults('Premission : RF',rf_acc_c,rf_prec_c,rf_rec_c,rf_f1_c)
```

Decision Tree

```
# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree2 = DecisionTreeClassifier(max_depth=30)

# fit the model
tree2.fit(X_train, y_train)

y_pred = tree2.predict(X_test)

dt_acc_c = accuracy_score(y_test,y_pred)
dt_prec_c = precision_score(y_test,y_pred)
dt_rec_c = recall_score(y_test,y_pred)
dt_f1_c = f1_score(y_test,y_pred)


storeResults('Premission : DT',dt_acc_c,dt_prec_c,dt_rec_c,dt_f1_c)
```

Comparison

```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'f1_score' : f1score,
                        'Recall'   : recall,

                      })


result
```

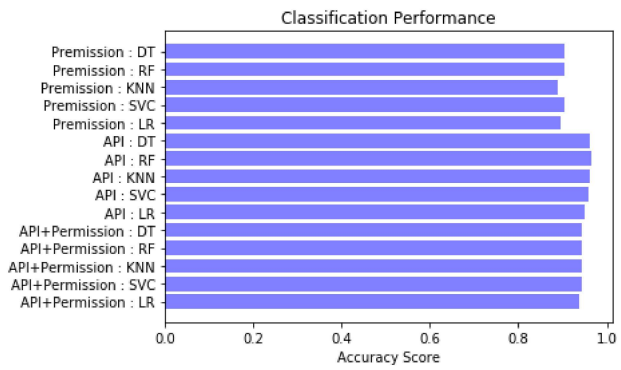
	ML Model	Accuracy	Precision	f1_score	Recall
0	API+Permission : LR	0.94	0.92	0.92	0.92
1	API+Permission : SVC	0.94	0.93	0.93	0.92
2	API+Permission : KNN	0.94	0.93	0.93	0.92
3	API+Permission : RF	0.94	0.93	0.93	0.92
4	API+Permission : DT	0.94	0.93	0.93	0.92
5	API : LR	0.95	0.94	0.93	0.93
6	API : SVC	0.96	0.96	0.95	0.94
7	API : KNN	0.96	0.96	0.95	0.94
8	API : RF	0.96	0.97	0.95	0.94
9	API : DT	0.96	0.96	0.95	0.94
10	Premission : LR	0.90	0.95	0.85	0.77
11	Premission : SVC	0.90	0.96	0.86	0.78
12	Premission : KNN	0.89	0.82	0.86	0.91
13	Premission : RF	0.91	0.96	0.86	0.79
14	Premission : DT	0.91	0.96	0.86	0.79

Graph

```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

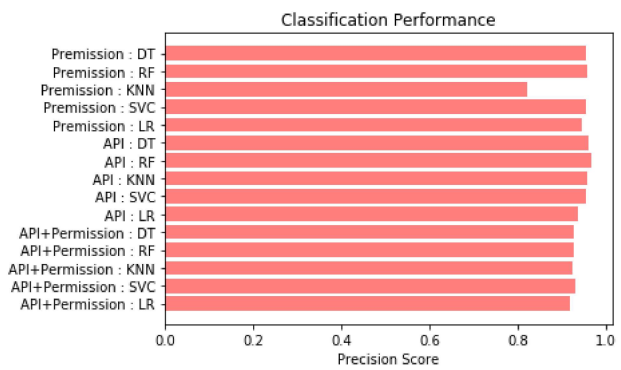
Accuracy

```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```



Precision

```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



Recall

```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
```