In [1]:

```
import pandas as pd
```

In [2]:

```
df = pd.read_csv("/Users/bruce/Documents/Study/phishing ML/Supportive_Docs/my_final_
```

In [3]:

```
df.isnull().sum()
```

Out[3]:

```
url       0
label     0
result    0
dtype: int64
```

In [4]:

```
print(df['label'].value_counts())
print(df['result'].value_counts())
```

```
malicious    108523
benign        35378
Name: label, dtype: int64
1    108523
0     35378
Name: result, dtype: int64
```

In [5]:

```
from urllib.parse import urlparse
```

In [6]:

```
df_process = df
```

In [7]:

```
of URL
_length'] = df_process['url'].apply(lambda x: len(x))



of Hostname
tname_length'] = df_process['url'].apply(lambda x: len(str(urlparse(x).hostname)))



ngth of the path from URL
h_length'] = df_process['url'].apply(lambda x: len(urlparse(x).path))



ngth of first directory

URL).path
2]) == '//':
len(a.split('/')[2])


urn len(a.split('/')[1])

urn 0

stDir_length'] = df_process['url'].apply(lambda v: fd(v))



l()


f XYZ



al-'] = df_process['url'].apply(lambda i: i.count('-'))



al@'] = df_process['url'].apply(lambda i: i.count('@'))
```

```
al?'] = df_process['url'].apply(lambda i: i.count('?'))
```

```
al%'] = df_process['url'].apply(lambda i: i.count('%'))
```

```
al.'] = df_process['url'].apply(lambda i: i.count('.'))
```

```
al='] = df_process['url'].apply(lambda i: i.count('='))
```

```
alhttp'] = df_process['url'].apply(lambda i: i.count('http'))
```

```
alhttps'] = df_process['url'].apply(lambda i: i.count('https'))
```

```
alwww'] = df_process['url'].apply(lambda i: i.count('www'))
```

```
 digits in the URL
(url):

l:
numeric():
its = digits + 1
ts
aldigits']= df_process['url'].apply(lambda i: totaldigits(i))
```

```
 letters in the URL
s(url):
l:
alpha():
ters = letters + 1
ers
alletters']= df_process['url'].apply(lambda i: totalletters(i))



 directories in the URL
rl):
lparse(url).path
lir.count('/')
aldirs'] = df_process['url'].apply(lambda i: totaldirs(i))



ts in netloc
alnetlocdots'] = df_process['url'].apply(lambda i: urlparse(i).netloc.count('.'))



gned Numbers Authority (IANA) approved list of TLD's

","r")

pend(i.removesuffix('\n').lower())



D's in the netloc


):


 TLD_List:
nt+=1
t

alTLD'] = df_process['url'].apply(lambda i: count_TLD(urlparse(i).netloc.split('.')))


ation Features
```

```
 not in domain


ddress(url):
search(
?\\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\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
-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})
fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6

 match.group()
-1

 'No matching pattern found'
1
_of_ip'] = df_process['url'].apply(lambda i: having_ip_address(i))




hortening service
service(url):
search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.g
       'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snip
       'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.
       'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|
       'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly
       'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\
       'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.
       'tr\.im|link\.zip\.net',
       url)

-1

1
rt_url'] = df_process['url'].apply(lambda i: shortening_service(i))




l()




e




()
```

```
el'].value_counts()
```

```
sv("URL_test.csv")
```

```
['aaa', 'aarp', 'abarth', 'abb', 'abbott', 'abbvie', 'abc', 'able', 'a
bogado', 'abudhabi', 'ac', 'academy', 'accenture', 'accountant', 'acco
untants', 'aco', 'actor', 'ad', 'adac', 'ads', 'adult', 'ae', 'aeg',
'aero', 'aetna', 'af', 'afl', 'africa', 'ag', 'agakhan', 'agency', 'a
i', 'aig', 'airbus', 'airforce', 'airtel', 'akdn', 'al', 'alfaromeo',
'alibaba', 'alipay', 'allfinanz', 'allstate', 'ally', 'alsace', 'alsto
m', 'am', 'amazon', 'americanexpress', 'americanfamily', 'amex', 'amfa
m', 'amica', 'amsterdam', 'analytics', 'android', 'anquan', 'anz', 'a
o', 'aol', 'apartments', 'app', 'apple', 'aq', 'aquarelle', 'ar', 'ara
b', 'aramco', 'archi', 'army', 'arpa', 'art', 'arte', 'as', 'asda', 'a
sia', 'associates', 'at', 'athleta', 'attorney', 'au', 'auction', 'aud
i', 'audible', 'audio', 'auspost', 'author', 'auto', 'autos', 'avianc
a', 'aw', 'aws', 'ax', 'axa', 'az', 'azure', 'ba', 'baby', 'baidu', 'b
anamex', 'bananarepublic', 'band', 'bank', 'bar', 'barcelona', 'barcla
ycard', 'barclays', 'barefoot', 'bargains', 'baseball', 'basketball',
'bauhaus', 'bayern', 'bb', 'bbc', 'bbt', 'bbva', 'bcg', 'bcn', 'bd',
'be', 'beats', 'beauty', 'beer', 'bentley', 'berlin', 'best', 'bestbu
y', 'bet', 'bf', 'bg', 'bh', 'bharti', 'bi', 'bible', 'bid', 'bike',
'bing', 'bingo', 'bio', 'biz', 'bj', 'black', 'blackfriday', 'blockbus
```

In [8]:

```
df_process.head()
```

Out[8]:

	url	label	result	url_length	hostname_length	path_lengt
0	http://1337x.to/torrent/1048648/American-Snipe...	benign	0	83	8	6
1	http://1337x.to/torrent/1110018/Blackhat-2015-...	benign	0	83	8	6
2	http://1337x.to/torrent/1122940/Blackhat-2015-...	benign	0	83	8	6
3	http://1337x.to/torrent/1124395/Fast-and-Furio...	benign	0	83	8	6
4	http://1337x.to/torrent/1145504/Avengers-Age-o...	benign	0	83	8	6

5 rows × 23 columns

In [9]:

```
df_train= pd.read_csv("URL_phase1.csv")
```

In [10]:

```
df_train.head()
```

Out[10]:

	Unnamed: 0	url	label	result	url_length	hostname_length	path_length
0	0	https://www.google.com	benign	0	22	14	0
1	1	https://www.youtube.com	benign	0	23	15	0
2	2	https://www.facebook.com	benign	0	24	16	0
3	3	https://www.baidu.com	benign	0	21	13	0
4	4	https://www.wikipedia.org	benign	0	25	17	0

5 rows × 24 columns

In [11]:

```
# model training
```

In [12]:

```
# finalize train dataset
```

In [13]:

```
df_train.head()
```

Out[13]:

	Unnamed: 0	url	label	result	url_length	hostname_length	path_length
0	0	https://www.google.com	benign	0	22	14	0
1	1	https://www.youtube.com	benign	0	23	15	0
2	2	https://www.facebook.com	benign	0	24	16	0
3	3	https://www.baidu.com	benign	0	21	13	0
4	4	https://www.wikipedia.org	benign	0	25	17	0

5 rows × 24 columns

In [14]:

```
df_ml_train = df_train
```

In [15]:

```
df_ml_train.drop(['Unnamed: 0','url','label'], axis=1, inplace =True)
```

In [16]:

```
df_ml_train.head()
```

Out[16]:

	result	url_length	hostname_length	path_length	FirstDir_length	total-	total@	total?	total%
0	0	22	14	0	0	0	0	0	0
1	0	23	15	0	0	0	0	0	0
2	0	24	16	0	0	0	0	0	0
3	0	21	13	0	0	0	0	0	0
4	0	25	17	0	0	0	0	0	0

5 rows × 21 columns

In [17]:

```
df_ml_train_X = df_ml_train.loc[:, df_ml_train.columns != 'result']
```

In [18]:

```
df_ml_train_X.head()
```

Out[18]:

	url_length	hostname_length	path_length	FirstDir_length	total-	total@	total?	total%	total.
0	22	14	0	0	0	0	0	0	2
1	23	15	0	0	0	0	0	0	2
2	24	16	0	0	0	0	0	0	2
3	21	13	0	0	0	0	0	0	2
4	25	17	0	0	0	0	0	0	2

In [19]:

```
df_ml_train_Y = df_ml_train['result']
```

In [20]:

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
```

In [21]:

```
x_sample_train, y_sample_train = SMOTE().fit_resample(df_ml_train_X, df_ml_train_Y.v
```

In [22]:

```
print(df_ml_train_X.shape)
print(df_ml_train_X.shape)
print(x_sample_train.shape)
print(y_sample_train.shape)
```

```
(450176, 20)
(450176, 20)
(691476, 20)
(691476,)
```

In [23]:

```
#finalize test dataset
```

In [24]:

```
df_test= pd.read_csv("URL_test.csv")
```

In [25]:

```
df_test.head()
```

Out[25]:

alhttps	totalwww	totaldigits	totalletters	totaldirs	totalnetlocdots	totalTLD	use_of_ip	short_url
0	0	18	49	4	1	1	1	1
0	0	23	43	4	1	1	1	1
0	0	22	44	4	1	1	1	1
0	0	18	46	4	1	1	1	1
0	0	18	48	4	1	1	1	1

In [26]:

```
df_test.drop(['Unnamed: 0','url','label'], axis=1, inplace =True)
```

In [27]:

```
df_test_X = df_test.loc[:, df_test.columns != 'result']
```

In [28]:

```
df_test_Y = df_test['result']
```

In [29]:

```
print(df_test_X.shape)
print(df_test_Y.shape)
```

```
(143901, 20)
(143901,)
```

KNN with Hyper-parameter tuning

In [30]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [31]:

```
knn_model_hyper = KNeighborsClassifier(n_neighbors=10,weights='distance',algorithm='
                                       leaf_size=30, p=1, metric='manhattan')
```

In [32]:

```
knn_model_hyper.fit(x_sample_train,y_sample_train)
```

Out[32]:

```
KNeighborsClassifier(algorithm='ball_tree', metric='manhattan', n_nei
ghbors=10,
                     p=1, weights='distance')
```

In [33]:

```
knn_pred = knn_model_hyper.predict(df_test_X)
```

In [34]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [36]:

```
knn_accuracy = accuracy_score(df_test_Y, knn_pred)
```

In [37]:

```
print("Accuracy of KNN (Hyper-parameter) on Testing dataset : ",round(knn_accuracy,3
```

```
Accuracy of KNN (Hyper-parameter) on Testing dataset :  0.437
```

KNN : without hyper-parameter tuning

In [38]:

```
knn_model_2 = KNeighborsClassifier()
```

In [39]:

```
knn_model_2.fit(x_sample_train,y_sample_train)
```

Out[39]:

```
KNeighborsClassifier
KNeighborsClassifier()
```

In [40]:

```
knn_pred_2 = knn_model_2.predict(df_test_X)
```

In [41]:

```
knn_accuracy_2 = accuracy_score(df_test_Y, knn_pred_2)
```

In [42]:

```
print("Accuracy of KNN (without Hyper-parameters) on Testing dataset : ",round(knn_a
```

```
Accuracy of KNN (without Hyper-parameters) on Testing dataset :  0.445
```

Decision Tree without Hyper-parameter tuning

In [43]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [44]:

```
dt_model = DecisionTreeClassifier()
```

In [45]:

```
dt_model.fit(x_sample_train, y_sample_train)
```

Out[45]:

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [46]:

```
dt_model_pred = dt_model.predict(df_test_X)
```

In [47]:

```
dt_accuracy = accuracy_score(df_test_Y, dt_model_pred)
```

In [48]:

```
print('The accuracy of Decision Tree Classifier is : ', round(dt_accuracy,3))
```

```
The accuracy of Decision Tree Classifier is :  0.753
```

Decision Tree with Hyper-parameter tuning

In [49]:

```
dt_model_hyper = DecisionTreeClassifier(criterion='entropy')
```

In [50]:

```
dt_model_hyper.fit(x_sample_train, y_sample_train)
```

Out[50]:

```
DecisionTreeClassifier(criterion='entropy')
```

In [51]:

```
dt_model_pred_hyper = dt_model_hyper.predict(df_test_X)
```

In [52]:

```
dt_accuracy_hyper = accuracy_score(df_test_Y, dt_model_pred_hyper)
```

In [53]:

```
print('The accuracy of Decision Tree Classifier (with hyper-parameter tuning) is : '
```

```
The accuracy of Decision Tree Classifier (with hyper-parameter tuning)
is :  0.753
```

Random Forest without Hyper-parameters

In [54]:

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier()

rf_model.fit(x_sample_train, y_sample_train)

rf_model_pred = rf_model.predict(df_test_X)

rf_accuracy = accuracy_score(df_test_Y, rf_model_pred)

print('The accuracy of Random Forest(without hyper-parameer) is : ' , round(rf_accur
```

```
The accuracy of Random Forest(without hyper-parameer) is :  0.754
```

Random Forest with Hyper-parameters

In [55]:

```
from sklearn.ensemble import RandomForestClassifier

rf_model_hyper = RandomForestClassifier(n_estimators=150,max_features=None, criteric

rf_model_hyper.fit(x_sample_train, y_sample_train)

rf_model_hyper_pred = rf_model_hyper.predict(df_test_X)

rf_hyper_accuracy = accuracy_score(df_test_Y, rf_model_hyper_pred)

print('The accuracy of Random Forest(with hyper-parameer tuning) is : ' , round(rf_h
```

```
The accuracy of Random Forest(with hyper-parameer tuning) is :  0.754
```

In [56]:

```
print(rf_hyper_accuracy)
```

```
0.7543172041889911
```

Extremely Randomized Tree Classifier

In [58]:

```
from sklearn.ensemble import ExtraTreesClassifier

xrf_model = ExtraTreesClassifier()
xrf_model.fit(x_sample_train, y_sample_train)
xrf_pred = xrf_model.predict(df_test_X)
xrf_accuracy = accuracy_score(df_test_Y, xrf_pred)
print("Accuracy of XRF is : ",round(xrf_accuracy,3))
```

```
Accuracy of XRF is :  0.754
```

SAVING the model

In [60]:

```
import pickle
```

In [61]:

```
with open('saved_xrf','wb') as f:
    pickle.dump(xrf_model,f)
```

Trying models after scaling

In [63]:

```
from sklearn import preprocessing
```

In [64]:

```
scaling = preprocessing.StandardScaler()
```

In [65]:

```
x_sample_train_scaled = scaling.fit_transform(x_sample_train)
```

In [66]:

```
df_test_X_scaled = scaling.fit_transform(df_test_X)
```

In [68]:

```
x_sample_train_scaled
```

Out[68]:

```
array([[-0.85279654, -0.64418721, -1.05736092, ...,  2.14813952,
         0.11976771,  0.26375863],
       [-0.83153122, -0.51045365, -1.05736092, ...,  2.14813952,
         0.11976771,  0.26375863],
       [-0.8102659 , -0.3767201 , -1.05736092, ..., -0.26136056,
         0.11976771,  0.26375863],
       ...,
       [-0.78900058, -1.31285499, -0.74631482, ..., -0.26136056,
         0.11976771, -4.00865115],
       [-0.2998982 , -0.91165432,  0.07018117, ..., -0.26136056,
         0.11976771,  0.26375863],
       [ 0.82716379,  0.15821412,  1.70317316, ..., -0.26136056,
         0.11976771,  0.26375863]])
```

Extra Tree after scaling

In [70]:

```
xrf_model_scaled = ExtraTreesClassifier(criterion='entropy')
xrf_model_scaled.fit(x_sample_train_scaled, y_sample_train)
xrf_pred_scaled = xrf_model_scaled.predict(df_test_X_scaled)
xrf_scaled_accuracy = accuracy_score(df_test_Y, xrf_pred_scaled)
print("Accuracy of XRF is : ",round(xrf_scaled_accuracy,3))
```

```
Accuracy of XRF is :  0.747
```

XGBoost on scaled data

In [71]:

```
from xgboost import XGBClassifier
```

In [92]:

```
boost = XGBClassifier(n_estimators=250, max_depth=5, learning_rate=1, objective='bir
```

In [93]:

```
boost.fit(x_sample_train_scaled, y_sample_train)
```

Out[93]:

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytr
ee=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-
1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=1, max_bin=25
6,
              max_cat_threshold=64, max_cat_to_onehot=4, max_delta_st
```

In [94]:

```
boost_pred_scaled = boost.predict(df_test_X_scaled)
```

In [95]:

```
boost_pred_scaled_accuracy = accuracy_score(df_test_Y, boost_pred_scaled)
```

In [97]:

```
print("Accuracy of XGBoost is : ",round(boost_pred_scaled_accuracy,3))
```

```
Accuracy of XGBoost is :  0.765
```

DEEP LEARNING

In [99]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, B
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
2022-11-29 21:06:51.449844: I tensorflow/core/platform/cpu_feature_gua
rd.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
```

In [101]:

```
x_sample_train_scaled.shape
```

Out[101]:

```
(691476, 20)
```

In [102]:

```
# model building
model = Sequential()
model.add(Dense(32, activation = 'relu', input_shape = (20, )))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_4 (Dense)              (None, 32)                672

 dense_5 (Dense)              (None, 16)                528

 dense_6 (Dense)              (None, 8)                 136

 dense_7 (Dense)              (None, 1)                 9

=================================================================
Total params: 1,345
Trainable params: 1,345
Non-trainable params: 0
_________________________________________________________________
```

In [103]:

```
# compiling model
opt = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer= opt ,loss='binary_crossentropy',metrics=['acc'])
```

In [105]:

```
# custom callback to stop the training when certain metric value is reached

# stop training when validation loss reach 0.1
class myCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('val_loss')>0.1):
            print("\nReached 0.1 val_loss so cancelling training!")
            self.model.stop_training = True

callback = myCallback()
```

In [106]:

```
ze=260, callbacks=[callback],validation_data=(df_test_X_scaled,df_test_Y),verbose=1)
```

```
Epoch 1/3
2657/2660 [============================>.] - ETA: 0s - loss: 0.1141 -
acc: 0.9745
Reached 0.1 val_loss so cancelling training!
2660/2660 [==============================] - 7s 2ms/step - loss: 0.114
0 - acc: 0.9745 - val_loss: 1.8937 - val_acc: 0.4041
```

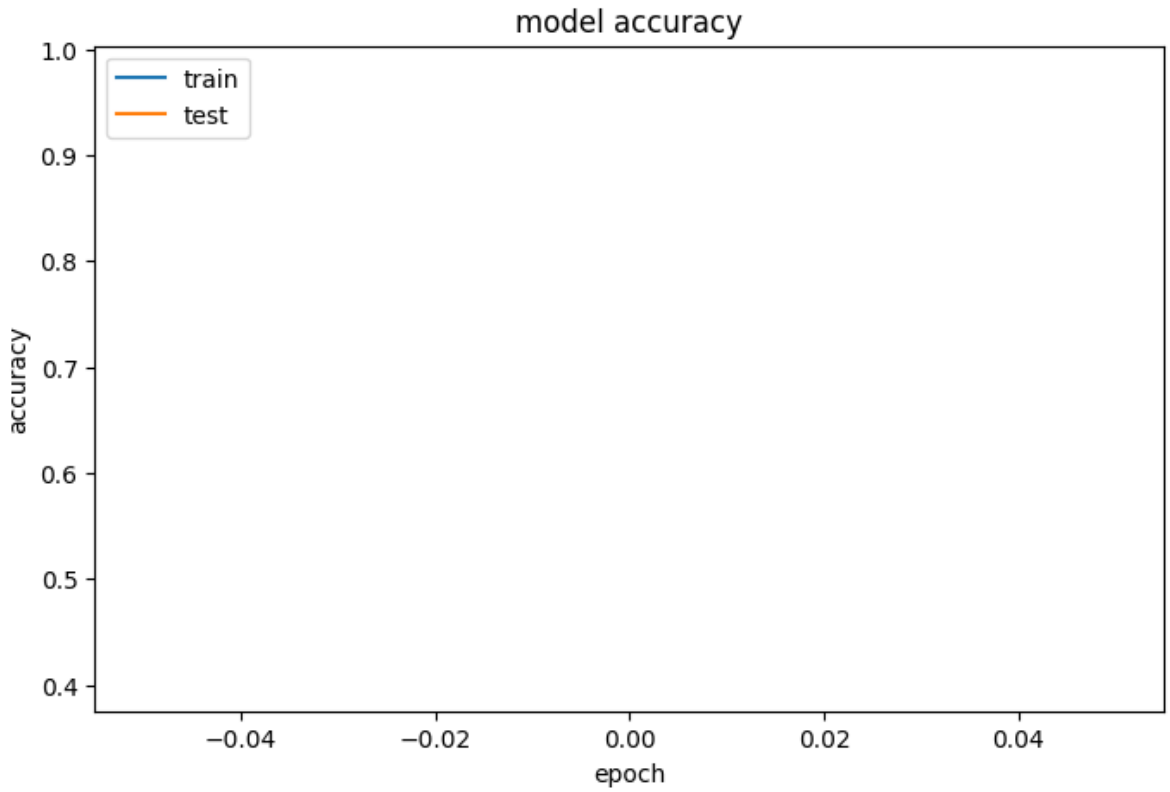
In [108]:

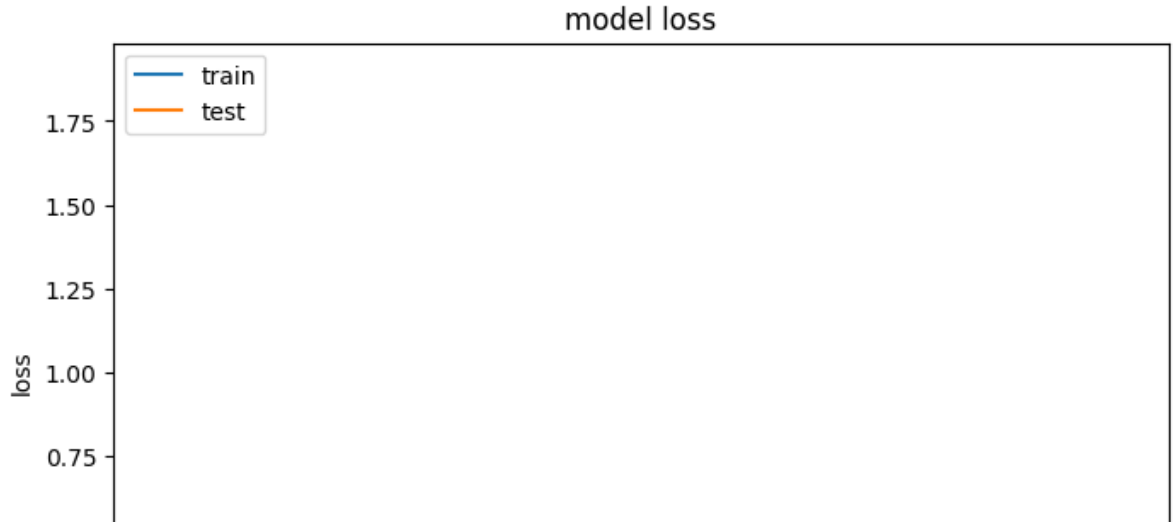
```
import matplotlib.pyplot as plt
```

In [109]:

```
# DISPLAYING MODEL TRAINING HISTORY

# list all data in history
print(history.history.keys())

# summarize history for accuracy
plt.figure(figsize=(8,5))
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.figure(figsize=(8,5))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```





In [110]:

```
# Evaluate the model on test dataset
loss, acc = model.evaluate(df_test_X_scaled,df_test_Y,verbose=1)
print('Test loss: {}'.format(loss))
print('Test Accuracy: {}'.format(acc))
```

```
4497/4497 [==============================] - 3s 708us/step - loss: 1.8
937 - acc: 0.4041
Test loss: 1.8936975002288818
Test Accuracy: 0.40408337116241455
```

In [112]:

```
import numpy as np
```

In [113]:

```
# predicting on test data.
pred_test = model.predict(df_test_X_scaled)
for i in range (len(pred_test)):
    if (pred_test[i] < 0.5):
        pred_test[i] = 0
    else:
        pred_test[i] = 1
pred_test = pred_test.astype(int)
```

```
4497/4497 [==============================] - 3s 616us/step
PREDICTED :
Mallicious
Mallicious
Mallicious
Mallicious
Mallicious
Mallicious
Mallicious
Mallicious
Mallicious
Mallicious
Mallicious


ACTUAL :
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
Non Mallicious
```

In [ ]:

```
def view_result(array):
    array = np.array(array)
    for i in range(len(array)):
        if array[i] == 0:
            print("Non Mallicious")
        else:
            print("Mallicious")

print("PREDICTED : ")
view_result(pred_test[:11])
print("\n")
print("ACTUAL : ")
view_result(df_test_Y[:11])
```

NEW TENSOR FLOW

In [114]:

```
import tensorflow as ft
```

In [115]:

```
ft.random.set_seed(42)

model = ft.keras.Sequential([
    ft.keras.layers.Dense(128, activation='relu'),
    ft.keras.layers.Dense(256, activation='relu'),
    ft.keras.layers.Dense(256, activation='relu'),
    ft.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(
    loss=ft.keras.losses.binary_crossentropy,
    optimizer=ft.keras.optimizers.Adam(lr=0.03),
    metrics=[
        ft.keras.metrics.BinaryAccuracy(name='accuracy'),
        ft.keras.metrics.Precision(name='precision'),
        ft.keras.metrics.Recall(name='recall')
    ]
)

history = model.fit(x_sample_train_scaled, y_sample_train, epochs=100)
```

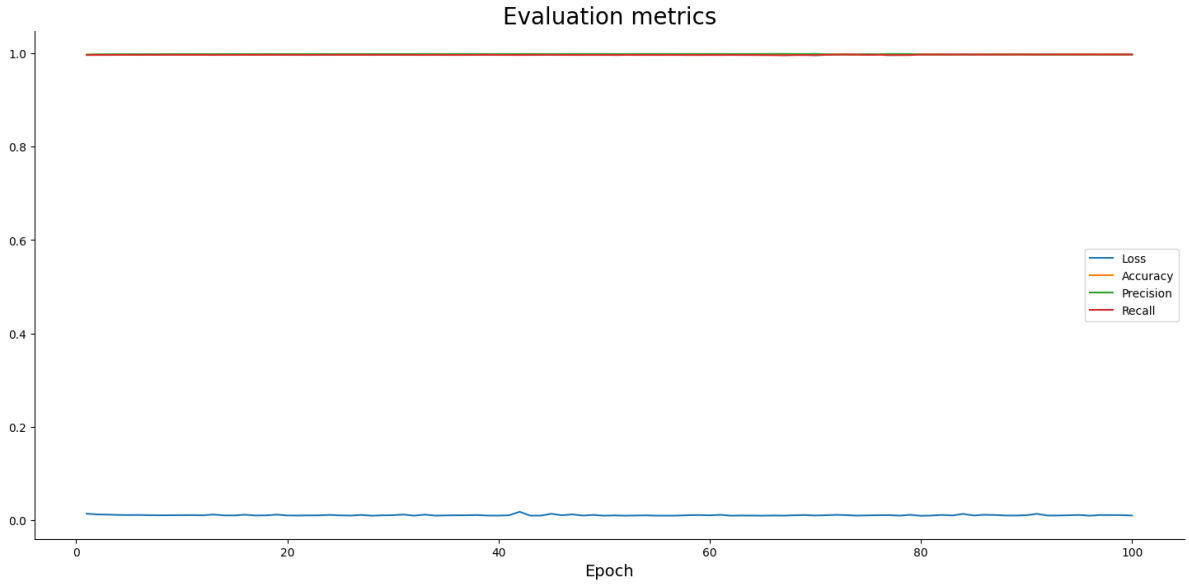
```
Epoch 95/100
21609/21609 [==============================] - 37s 2ms/step - loss: 0.
0109 - accuracy: 0.9977 - precision: 0.9976 - recall: 0.9978
Epoch 96/100
21609/21609 [==============================] - 39s 2ms/step - loss: 0.
0093 - accuracy: 0.9977 - precision: 0.9976 - recall: 0.9978
Epoch 97/100
21609/21609 [==============================] - 38s 2ms/step - loss: 0.
0107 - accuracy: 0.9977 - precision: 0.9976 - recall: 0.9978

Epoch 98/100
21609/21609 [==============================] - 38s 2ms/step - loss: 0.
0107 - accuracy: 0.9977 - precision: 0.9976 - recall: 0.9978
Epoch 99/100
21609/21609 [==============================] - 39s 2ms/step - loss: 0.
0106 - accuracy: 0.9977 - precision: 0.9977 - recall: 0.9978
Epoch 100/100
21609/21609 [==============================] - 38s 2ms/step - loss: 0.
0097 - accuracy: 0.9977 - precision: 0.9977 - recall: 0.9978
```

In [116]:

```
import matplotlib.pyplot as plt
from matplotlib import rcParams

rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

In [117]:

```
plt.plot(np.arange(1, 101), history.history['loss'], label='Loss')
plt.plot(np.arange(1, 101), history.history['accuracy'], label='Accuracy')
plt.plot(np.arange(1, 101), history.history['precision'], label='Precision')
plt.plot(np.arange(1, 101), history.history['recall'], label='Recall')
plt.title('Evaluation metrics', size=20)
plt.xlabel('Epoch', size=14)
plt.legend();
```



In [118]:

```
predictions = model.predict(df_test_X_scaled)
predictions
```

```
4497/4497 [==============================] - 4s 828us/step
```

Out[118]:

```
array([[1.0000000e+00],
       [1.0000000e+00],
       [1.0000000e+00],
       ...,
       [7.4329654e-10],
       [7.5284329e-10],
       [1.0000000e+00]], dtype=float32)
```

In [119]:

```
prediction_classes = [1 if prob > 0.5 else 0 for prob in np.ravel(predictions)]
```

In [127]:

```
df_test_X_scaled
```

Out[127]:

```
array([[-0.22924993, -1.41252652,  0.83056877, ..., -0.39404494,
         0.01419747,  0.20019433],
       [-0.22924993, -1.41252652,  0.83056877, ..., -0.39404494,
         0.01419747,  0.20019433],
       [-0.22924993, -1.41252652,  0.83056877, ..., -0.39404494,
         0.01419747,  0.20019433],
       ...,
       [-0.31900902, -0.12666924, -0.64084925, ..., -0.39404494,
         0.01419747,  0.20019433],
       [-0.65560562, -0.2874014 , -0.64084925, ..., -0.39404494,
         0.01419747,  0.20019433],
       [-0.88000335,  3.7309026 , -0.86295009, ..., -0.39404494,
         0.01419747, -4.99514643]])
```

In [120]:

```
loss, accuracy, precision, recall = model.evaluate(df_test_X_scaled, df_test_Y)
loss, accuracy, precision, recall
```

```
4497/4497 [==============================] - 5s 995us/step - loss: 74
8.7009 - accuracy: 0.6412 - precision: 0.7276 - recall: 0.8380
```

Out[120]:

```
(748.7008666992188, 0.6411769390106201, 0.7275629043579102, 0.83798825
74081421)
```

In [122]:

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(df_test_Y, prediction_classes))
```

```
[[ 1325 34053]
 [17582 90941]]
```

In [123]:

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

print(f'Accuracy:  {accuracy_score(df_test_Y, prediction_classes):.2f}')
print(f'Precision: {precision_score(df_test_Y, prediction_classes):.2f}')
print(f'Recall:    {recall_score(df_test_Y, prediction_classes):.2f}')
```

```
Accuracy:  0.64
Precision: 0.73
Recall:    0.84
```

In [ ]:

```

```