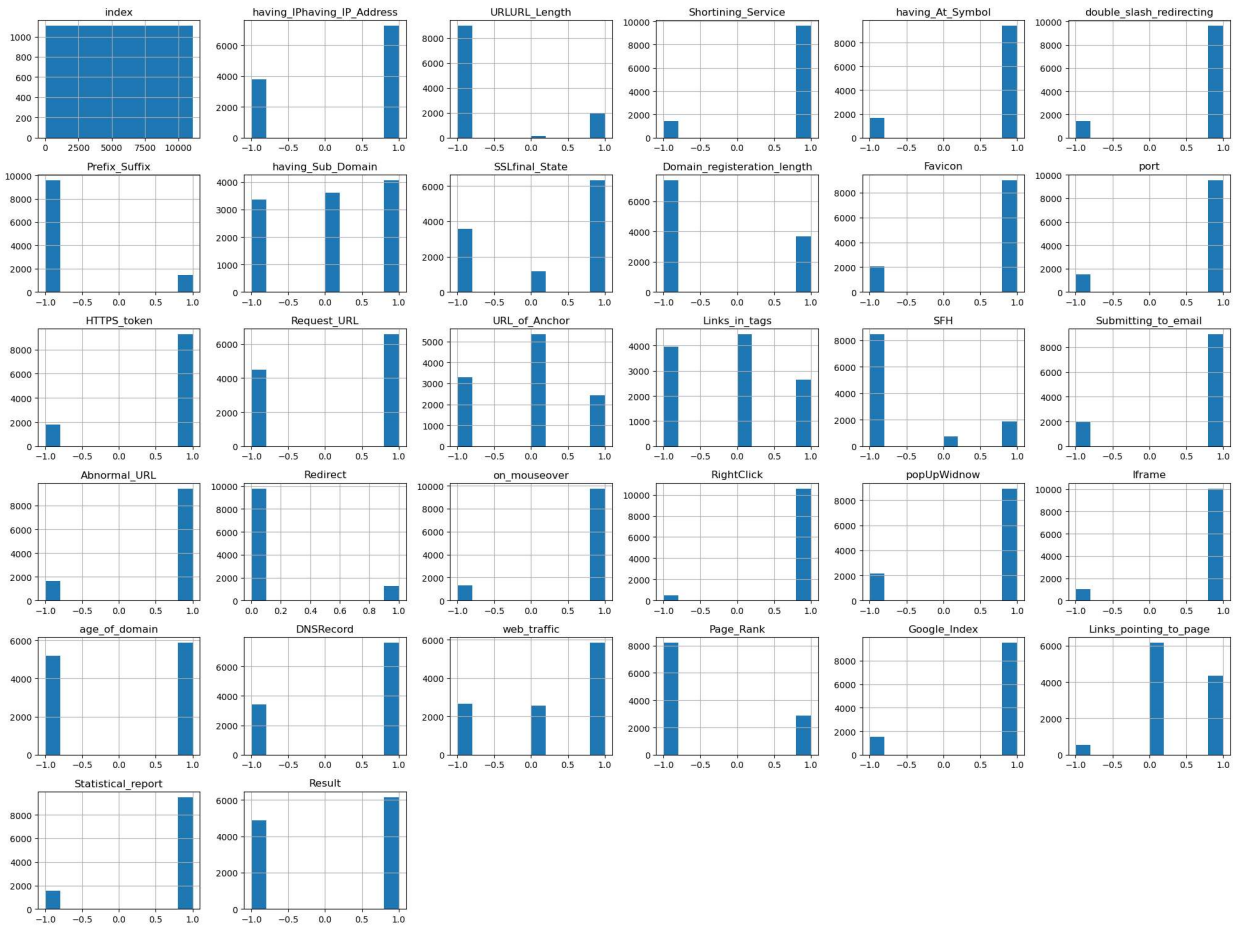
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import torch.nn as nn
import seaborn as sns
import torch
```

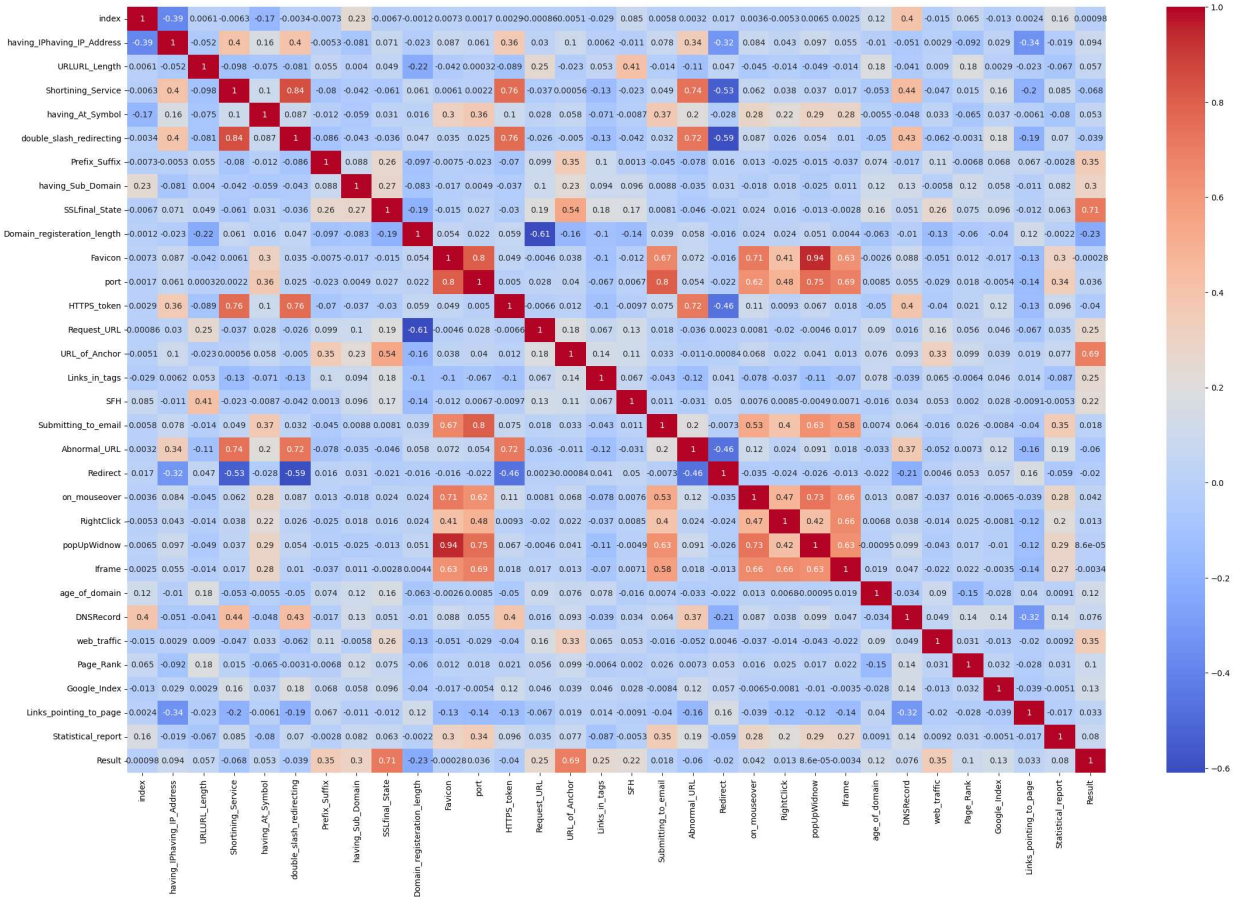
In [2]:
```
#Import the dataset
data = pd.read_csv('capstone dataset.csv')
```

In [3]:
```
#Explore the data using histograma
data.hist(figsize=(20, 15))
plt.tight_layout()
plt.show()
```



Features most closely affiliated with phishing URLs are 'URLURL_Length', 'Prefix_Suffix', 'SFH', 'Redirect', and 'Page_Rank'.

In [4]:
```
#Explore the data using heatmaps
plt.figure(figsize=(27, 17))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.show()
```



Features most closely correlated with result are 'SSLfinal_State' and 'URL_of_Anchor', which are also closely correlated with each other.

In [5]:
```
#View first five rows of the dataset
print(data.head())
```

```
   index  having_IPhaving_IP_Address  URLURL_Length  Shortining_Service  \
0      1                          -1              1                   1
1      2                           1              1                   1
2      3                           1              0                   1
3      4                           1              0                   1
4      5                           1              0                  -1

   having_At_Symbol  double_slash_redirecting  Prefix_Suffix  \
0                 1                        -1             -1
1                 1                         1             -1
2                 1                         1             -1
3                 1                         1             -1
4                 1                         1             -1

   having_Sub_Domain  SSLfinal_State  Domain_registeration_length  ...  \
0                 -1              -1                           -1  ...
1                  0               1                           -1  ...
2                 -1              -1                           -1  ...
3                 -1              -1                            1  ...
4                  1               1                           -1  ...

   popUpWidnow  Iframe  age_of_domain  DNSRecord  web_traffic  Page_Rank  \
0            1       1             -1         -1           -1         -1
1            1       1             -1         -1            0         -1
2            1       1              1         -1            1         -1
3            1       1             -1         -1            1         -1
4           -1       1             -1         -1            0         -1

   Google_Index  Links_pointing_to_page  Statistical_report  Result
0             1                       1                  -1      -1
1             1                       1                   1      -1
2             1                       0                  -1      -1
3             1                      -1                   1      -1
4             1                       1                   1       1

[5 rows x 32 columns]
```

In [6]:
```
#Determine the number of samples present in the data
print("Number of samples in dataset:", len(data))
```

```
Number of samples in dataset: 11055
```

In [7]:
```
#Determine the number of unique elements in all features
for column in data.columns:
    print(f"Unique elements in {column}: {data[column].unique()}")
```

```
Unique elements in index: [    1     2     3 ... 11053 11054 11055]
Unique elements in having_IPhaving_IP_Address: [-1  1]
Unique elements in URLURL_Length: [ 1  0 -1]
Unique elements in Shortining_Service: [ 1 -1]
Unique elements in having_At_Symbol: [ 1 -1]
Unique elements in double_slash_redirecting: [-1  1]
Unique elements in Prefix_Suffix: [-1  1]
Unique elements in having_Sub_Domain: [-1  0  1]
Unique elements in SSLfinal_State: [-1  1  0]
Unique elements in Domain_registeration_length: [-1  1]
Unique elements in Favicon: [ 1 -1]
Unique elements in port: [ 1 -1]
Unique elements in HTTPS_token: [-1  1]
Unique elements in Request_URL: [ 1 -1]
Unique elements in URL_of_Anchor: [-1  0  1]
Unique elements in Links_in_tags: [ 1 -1  0]
Unique elements in SFH: [-1  1  0]
Unique elements in Submitting_to_email: [-1  1]
Unique elements in Abnormal_URL: [-1  1]
Unique elements in Redirect: [0 1]
Unique elements in on_mouseover: [ 1 -1]
Unique elements in RightClick: [ 1 -1]
Unique elements in popUpWidnow: [ 1 -1]
Unique elements in Iframe: [ 1 -1]
Unique elements in age_of_domain: [-1  1]
Unique elements in DNSRecord: [-1  1]
Unique elements in web_traffic: [-1  0  1]
Unique elements in Page_Rank: [-1  1]
Unique elements in Google_Index: [ 1 -1]
Unique elements in Links_pointing_to_page: [ 1  0 -1]
Unique elements in Statistical_report: [-1  1]
Unique elements in Result: [-1  1]
```

In [8]:
```
#Check for null values in any features
print(data.isnull().sum())
```

```
index                          0
having_IPhaving_IP_Address     0
URLURL_Length                  0
Shortining_Service             0
having_At_Symbol               0
double_slash_redirecting       0
Prefix_Suffix                  0
having_Sub_Domain              0
SSLfinal_State                 0
Domain_registeration_length    0
Favicon                        0
port                           0
HTTPS_token                    0
Request_URL                    0
URL_of_Anchor                  0
Links_in_tags                  0
SFH                            0
Submitting_to_email            0
Abnormal_URL                   0
Redirect                       0
on_mouseover                   0
RightClick                     0
popUpWidnow                    0
Iframe                         0
age_of_domain                  0
DNSRecord                      0
web_traffic                    0
Page_Rank                      0
Google_Index                   0
Links_pointing_to_page         0
Statistical_report             0
Result                         0
dtype: int64
```

In [9]:
```
#Change 'Result' column to have labels of 1 and 0 instead of 1 and -1 to make binary c
data['Result'] = data['Result'].replace(-1, 0)
```

In [10]:
```
#Remove features that might be correlated with some threshold
#Correlation threshold is 0.9
#'PopUpWidnow' [sic] and 'Favicon' are highly correlated with each other and weakly co
#Drop just 'PopUpWidnow' for now, but if model performance is poor, consider dropping
def remove_correlated_features(data, threshold):
    correlated_columns = set()
    corr_matrix = data.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:
                colname = corr_matrix.columns[i]
                correlated_columns.add(colname)
    data_mod = data.copy()
    for col in correlated_columns:
        if col in data_mod.columns:
            del data_mod[col]
    return data_mod, correlated_columns

threshold_value = 0.9
mod_data, removed_columns = remove_correlated_features(data, threshold_value)
print("Removed columns:", removed_columns)
```

```
Removed columns: {'popUpWidnow'}
```

In [76]:
```
#Split data into training and testing sets
from sklearn.model_selection import train_test_split

X = mod_data.drop('Result', axis=1).values
y = data['Result'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [77]:
```
#Build classification model using a binary classifier to detect phishing URLs
class URLClassModel(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.linear1 = nn.Linear(input_size, 128)
        self.linear2 = nn.Linear(128, 64)
        self.linear3 = nn.Linear(64, 1)

    def forward(self, x):
        x = torch.relu(self.linear1(x))
        x = torch.relu(self.linear2(x))
        return torch.sigmoid(self.linear3(x))

    def predict(self, x):
        with torch.no_grad():
            pred = self.forward(x)
            return (pred >= 0.5).float()
```

In [78]:
```
#Convert datasets to tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)
```

In [79]:
```
#Create an instance of the model
input_size = X_train.shape[1]
model = URLClassModel(input_size)
```

In [80]:
```
#Establish hyperparameters
lr = 0.0001
epochs = 500
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

In [81]:
```
#Train the URL classification model
losses = []
for i in range(epochs):
    y_pred = model.forward(X_train_tensor)
    loss = criterion(y_pred, y_train_tensor)
    print("epoch:", i, "loss:", loss.item())
    losses.append(loss.item())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
epoch: 0 loss: 40.994163513183594
epoch: 1 loss: 40.48670196533203
epoch: 2 loss: 39.98867416381836
epoch: 3 loss: 39.28947448730469
epoch: 4 loss: 38.64054870605469
epoch: 5 loss: 37.65824890136719
epoch: 6 loss: 35.92600631713867
epoch: 7 loss: 33.64764404296875
epoch: 8 loss: 29.62323760986328
epoch: 9 loss: 22.15572738647461
epoch: 10 loss: 3.6679294109344482
epoch: 11 loss: 1.1071182489395142
epoch: 12 loss: 2.809910297393799
epoch: 13 loss: 3.730644464492798
epoch: 14 loss: 3.0865325927734375
epoch: 15 loss: 1.4828635454177856
epoch: 16 loss: 1.054864525794983
epoch: 17 loss: 2.0602195262908936
epoch: 18 loss: 2.3916850090026855
epoch: 19 loss: 2.0965046882629395
epoch: 20 loss: 1.3160040378570557
epoch: 21 loss: 0.6874426007270813
epoch: 22 loss: 1.4006483554840088
epoch: 23 loss: 1.5961506366729736
epoch: 24 loss: 1.024658203125
epoch: 25 loss: 0.755527138710022
epoch: 26 loss: 1.1880277395248413
epoch: 27 loss: 1.23599112033844
epoch: 28 loss: 0.889746367931366
epoch: 29 loss: 0.719582200050354
epoch: 30 loss: 1.0621819496154785
epoch: 31 loss: 0.9726356863975525
epoch: 32 loss: 0.676957368850708
epoch: 33 loss: 0.8652349710464478
epoch: 34 loss: 0.9628180861473083
epoch: 35 loss: 0.7800068259239197
epoch: 36 loss: 0.7088032960891724
epoch: 37 loss: 0.8609588742256165
epoch: 38 loss: 0.7656683325767517
epoch: 39 loss: 0.685696005821228
epoch: 40 loss: 0.7961599230766296
epoch: 41 loss: 0.767187774181366
epoch: 42 loss: 0.6737966537475586
epoch: 43 loss: 0.7533531188964844
epoch: 44 loss: 0.747346818447113
epoch: 45 loss: 0.6731841564178467
epoch: 46 loss: 0.7321374416351318
epoch: 47 loss: 0.7304474115371704
epoch: 48 loss: 0.6730685234069824
epoch: 49 loss: 0.7162692546844482
epoch: 50 loss: 0.7118736505508423
epoch: 51 loss: 0.6725254654884338
epoch: 52 loss: 0.7087305188179016
epoch: 53 loss: 0.6979984045028687
epoch: 54 loss: 0.6724028587341309
epoch: 55 loss: 0.7019550204277039
epoch: 56 loss: 0.6843198537826538
epoch: 57 loss: 0.67557692527771
epoch: 58 loss: 0.6951003670692444
epoch: 59 loss: 0.6763960123062134
```

```
epoch: 60 loss: 0.6774216294288635
epoch: 61 loss: 0.6878458261489868
epoch: 62 loss: 0.6707430481910706
epoch: 63 loss: 0.6801028847694397
epoch: 64 loss: 0.6799128651618958
epoch: 65 loss: 0.6698258519172668
epoch: 66 loss: 0.6798743605613708
epoch: 67 loss: 0.6729152202606201
epoch: 68 loss: 0.6716175079345703
epoch: 69 loss: 0.6769644021987915
epoch: 70 loss: 0.6692183017730713
epoch: 71 loss: 0.6731657385826111
epoch: 72 loss: 0.6724945306777954
epoch: 73 loss: 0.668572723865509
epoch: 74 loss: 0.6729088425636292
epoch: 75 loss: 0.668716549873352
epoch: 76 loss: 0.6694474220275879
epoch: 77 loss: 0.6702276468276978
epoch: 78 loss: 0.6665209531784058
epoch: 79 loss: 0.6682778000831604
epoch: 80 loss: 0.6660217046737671
epoch: 81 loss: 0.6665626168251038
epoch: 82 loss: 0.6663954257965088
epoch: 83 loss: 0.6651903390884399
epoch: 84 loss: 0.6662573218345642
epoch: 85 loss: 0.664637565612793
epoch: 86 loss: 0.6653842926025391
epoch: 87 loss: 0.664601743221283
epoch: 88 loss: 0.662880003452301
epoch: 89 loss: 0.6729545593261719
epoch: 90 loss: 0.664680540561676
epoch: 91 loss: 0.6685184240341187
epoch: 92 loss: 0.6669313311576843
epoch: 93 loss: 0.6642126441001892
epoch: 94 loss: 0.6679233908653259
epoch: 95 loss: 0.662444531917572
epoch: 96 loss: 0.6666568517684937
epoch: 97 loss: 0.6628766655921936
epoch: 98 loss: 0.6641493439674377
epoch: 99 loss: 0.663650393486023
epoch: 100 loss: 0.6617317199707031
epoch: 101 loss: 0.6634476184844971
epoch: 102 loss: 0.6609870791435242
epoch: 103 loss: 0.6618334054946899
epoch: 104 loss: 0.6596449613571167
epoch: 105 loss: 0.6603994965553284
epoch: 106 loss: 0.659303605556488
epoch: 107 loss: 0.6597132682800293
epoch: 108 loss: 0.6590138077735901
epoch: 109 loss: 0.6589828729629517
epoch: 110 loss: 0.6586461067199707
epoch: 111 loss: 0.6583083868026733
epoch: 112 loss: 0.6582183241844177
epoch: 113 loss: 0.6576958894729614
epoch: 114 loss: 0.6577273011207581
epoch: 115 loss: 0.6571407914161682
epoch: 116 loss: 0.6571860909461975
epoch: 117 loss: 0.6566076278686523
epoch: 118 loss: 0.6566194891929626
epoch: 119 loss: 0.6560739874839783
```

```
epoch: 480 loss: 0.4863181412220001
epoch: 481 loss: 0.48600032925605774
epoch: 482 loss: 0.48568326234817505
epoch: 483 loss: 0.4853637218475342
epoch: 484 loss: 0.4850350320339203
epoch: 485 loss: 0.4830268621444702
epoch: 486 loss: 0.4882514476776123
epoch: 487 loss: 0.49658817052841187
epoch: 488 loss: 0.505852997303009
epoch: 489 loss: 0.5040703415870667
epoch: 490 loss: 0.4987840950489044
epoch: 491 loss: 0.48896414041519165
epoch: 492 loss: 0.48332610726356506
epoch: 493 loss: 0.48265862464904785
epoch: 494 loss: 0.48579561710357666
epoch: 495 loss: 0.49043938517570496
epoch: 496 loss: 0.4921858012676239
epoch: 497 loss: 0.4918529689311981
epoch: 498 loss: 0.48741716146469116
epoch: 499 loss: 0.483333945274353
```

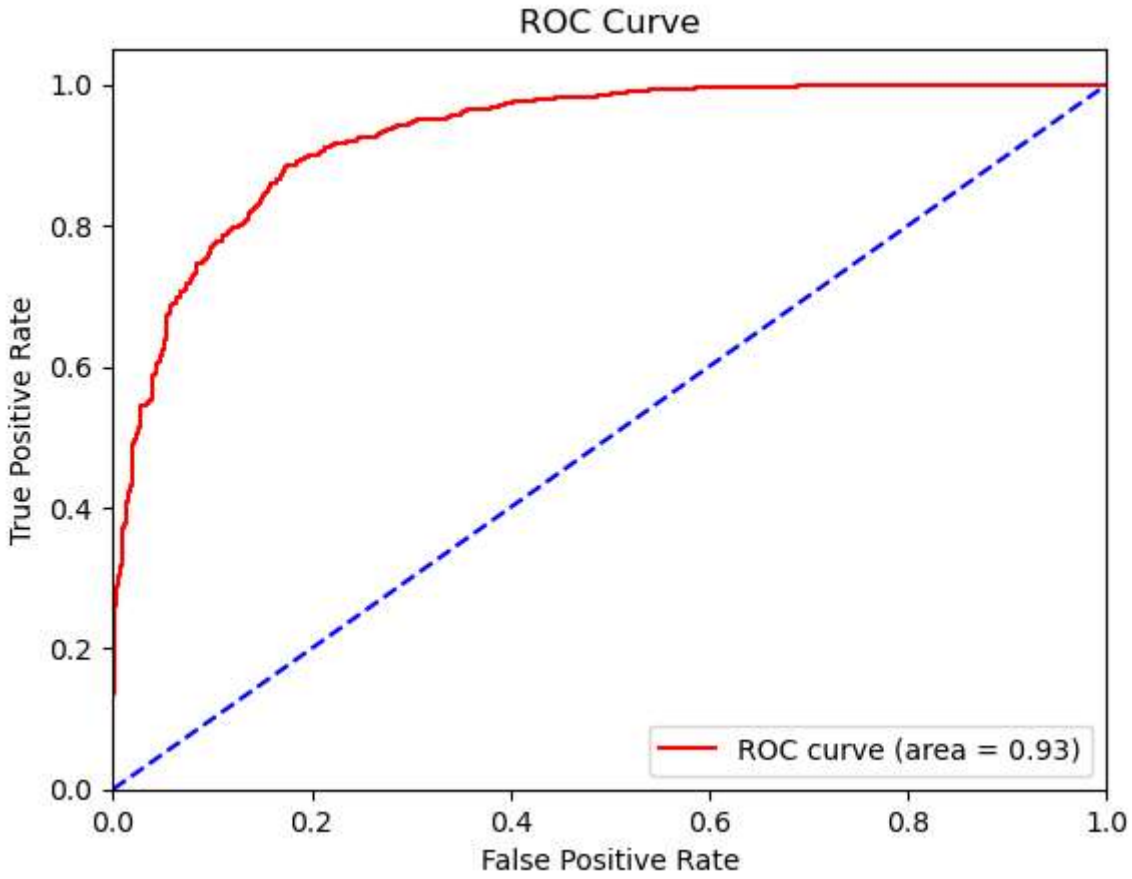
In [82]:

```
#Illustrate diagnostic ability of this binary classifier by plotting the ROC curve
from sklearn.metrics import roc_curve, auc

model.eval()
with torch.no_grad():
    y_prob = model(X_test_tensor)
y_prob = y_prob.numpy()
y_test_numpy = y_test_tensor.numpy()

fpr, tpr, thresholds = roc_curve(y_test_numpy, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='blue', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



In [83]:
```
#Validate the accuracy of data by the K-Fold cross-validation technique
from sklearn.model_selection import KFold

X_train_numpy = X_train_tensor.numpy()
y_train_numpy = y_train_tensor.numpy()
accuracies = []

kf = KFold(n_splits=5, shuffle=True, random_state=42)
for train, test in kf.split(X_train_numpy):
    X_val_fold = torch.tensor(X_train_numpy[test], dtype=torch.float32)
    y_val_fold = torch.tensor(y_train_numpy[test], dtype=torch.float32).view(-1, 1)

    model.eval()
    with torch.no_grad():
        y_pred = model.predict(X_val_fold)
        accuracy = (y_pred == y_val_fold).float().mean()
        accuracies.append(accuracy.item())

avg_accuracy = np.mean(accuracies)
print(f"Average accuracy after 5-fold cross-validation: {avg_accuracy:.4f}")
```

```
Average accuracy after 5-fold cross-validation: 0.8535
```

In [84]:
```
from sklearn.metrics import classification_report

model.eval()
with torch.no_grad():
    train_preds = model.predict(X_train_tensor)
train_preds_numpy = train_preds.numpy()

print("Training data classification report:")
```

```
print(classification_report(y_train_numpy, train_preds_numpy))

with torch.no_grad():
    test_preds = model.predict(X_test_tensor)
test_preds_numpy = test_preds.numpy()

print("\nTest data classification report:")
print(classification_report(y_test_numpy, test_preds_numpy))
```

```
Training data classification report:
              precision    recall  f1-score   support

         0.0       0.85      0.81      0.83      3942
         1.0       0.85      0.89      0.87      4902

    accuracy                           0.85      8844
   macro avg       0.85      0.85      0.85      8844
weighted avg       0.85      0.85      0.85      8844


Test data classification report:
              precision    recall  f1-score   support

         0.0       0.86      0.80      0.83       956
         1.0       0.86      0.90      0.88      1255

    accuracy                           0.86      2211
   macro avg       0.86      0.85      0.85      2211
weighted avg       0.86      0.86      0.86      2211
```

The model ended up having a training accuracy of 0.85 and a validation accuracy of 0.86. The initial accuracies were very poor, but increasing the number of perceptrons per convolutional layer helped. Increasing the number of epochs during which the model underwent training helped far more.

In [ ]: