Phishing Detector

Author: Peter Vuong

Business Understanding

Phishing is a type of cyber-attack that is engineered to steal user data that often includes login credentials, bank accounts, credit card numbers, and much more personal and sensitive information. Phishing attacks are particularly predatory against older people who are often not as familiar with technology.

The stakeholder for this project is Anti Phishing Working Group (APWG). APWG is one of the world's leading companies in working against cybercrimes. The goal of this project is to create a model that would accurately identify phishing websites as well as educate the general public about common features that are often associated with phishing websites.

Overview

This project aims to build a classification model to accurately identify what a phishing website is. The dataset used has came from Mendeley has roughly 12,000 rows of data. This data was collected from 2020.

Logistic Regression modeling is used to classify a website as legitimate(0) or phishing(1) based on the features present in the dataset, and important features.

Technical Overview

The data utilized in this project is from [Mendeley Data (https://data.mendeley.com/datasets/c2gw7fy2j4/3)](https://data.mendeley.com/datasets/c2gw7fy2j4/3). This data initially contained 87 features that were extracted from websites utilizing Python scripts.
Some of these features include:

- `google_index` - whether or not a website has been properly added to Google's index or not
- `phish_hints` - common features that are present in phishing websites such as incorrect spelling, urgent call to action, etc.
- `nb_www` - number of times the string 'www' appears in the URL

These features and many others in the dataset are common properties that are present in most, if not all URLs.
Initial preparation of the data included removing features that had no values (0) in them followed by a stepwise selection in order to identify significant features. After the stepwise selection, an initial baseline and LogisticRegression model was conducted with the leftover 42 features.
To further reduce the complexity of the model, an ExtraTreesClassifier was conducted on the data in order to identify the top 10 features that are important to the dataset. Once these features were identified, a final model (with optimized hyperparameters taken from GridSearchCV) was conducted on the data that produced an accuracy score of 92% and a recall score of 93%.

Data Understanding

The data from this project comes from [Mendeley Data (https://data.mendeley.com/datasets/c2gw7fy2j4/3)](https://data.mendeley.com/datasets/c2gw7fy2j4/3). This data initially had 87 features in the data.
The target variable in this project is the `status` feature where `0 = legitimate and 1 = phishing`. After preliminary EDA, 5 features were dropped from the initial dataset since they had zero entries in their respective feature columns. After this, the Pearson correlation coefficient was identified for each feature; however, the correlation coefficients were not as informative as expected.
A stepwise selection was utilized instead to filter out features that weren't significant to the dataset. Features that were deemed non-signficant had a P-value > 0.05.
After the step-wise selection, the number of features was reduced from 82 to 42 features. After a preliminary model was conducted on these 42 features, an ExtraTreeClassifier was utilized to identify the top 10 non-parametric features in the dataset. These features are isolated from the overall dataset to signify that these are features that are commonly associated with a phishing website.

In [1]:
```
# Importing necessary libraries to be used throughout the project
import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score,\
confusion_matrix, plot_confusion_matrix, ConfusionMatrixDisplay, plot_roc_curve
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import RFE
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Loading of Data and Initial EDA

In [2]:
```
# Loading in dataset as dataframe
df = pd.read_csv('data/dataset_B_05_2020.csv')
df
```

Out[2]:

	url	length_url	length_hostname	ip	nb_dots	nb
0	http://www.crestonwood.com/router.php	37	19	0	3	
1	http://shadetreetechnology.com/V4/validation/a...	77	23	1	1	
2	https://support-appleld.com.secureupdate.duila...	126	50	1	4	
3	http://rgipt.ac.in	18	11	0	2	
4	http://www.iracing.com/tracks/gateway-motorspo...	55	15	0	2	
...	...	...	...	...	...	
11425	http://www.fontspace.com/category/blackletter	45	17	0	2	
11426	http://www.budgetbots.com/server.php/Server%20...	84	18	0	5	
11427	https://www.facebook.com/Interactive-Televisio...	105	16	1	2	
11428	http://www.mypublicdomainpictures.com/	38	30	0	2	
11429	http://174.139.46.123/ap/signin?openid.pape.ma...	477	14	1	24	

11430 rows × 89 columns

In [3]:
```
# Preliminary information from .info() function which showed whether or not there
## Commented out this line of code since the output was pretty long
# df.info()
```

Here I see that the data has no missing values, so no sampling techniques are required for this dataset.

In [4]:
```
# Calling describe just to get a general breakdown of the dataset.
df.describe()
```

Out[4]:

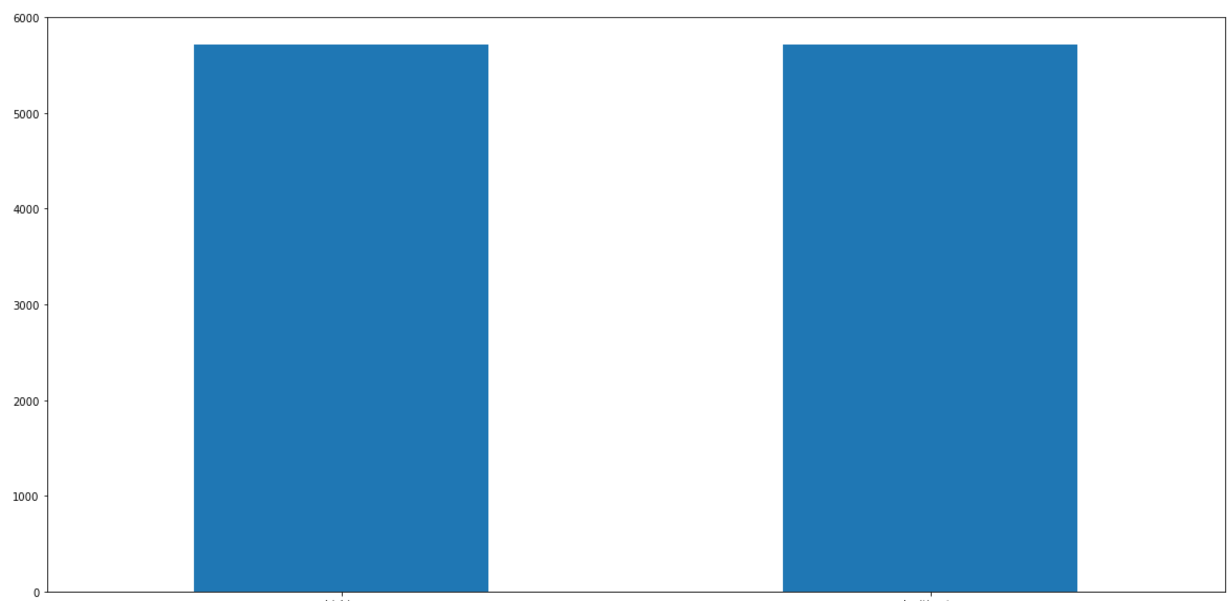
	length_url	length_hostname	ip	nb_dots	nb_hyphens	nb_at	
count	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	114
mean	61.126684	21.090289	0.150569	2.480752	0.997550	0.022222	
std	55.297318	10.777171	0.357644	1.369686	2.087087	0.155500	
min	12.000000	4.000000	0.000000	1.000000	0.000000	0.000000	
25%	33.000000	15.000000	0.000000	2.000000	0.000000	0.000000	
50%	47.000000	19.000000	0.000000	2.000000	0.000000	0.000000	
75%	71.000000	24.000000	0.000000	3.000000	1.000000	0.000000	
max	1641.000000	214.000000	1.000000	24.000000	43.000000	4.000000	

8 rows × 87 columns

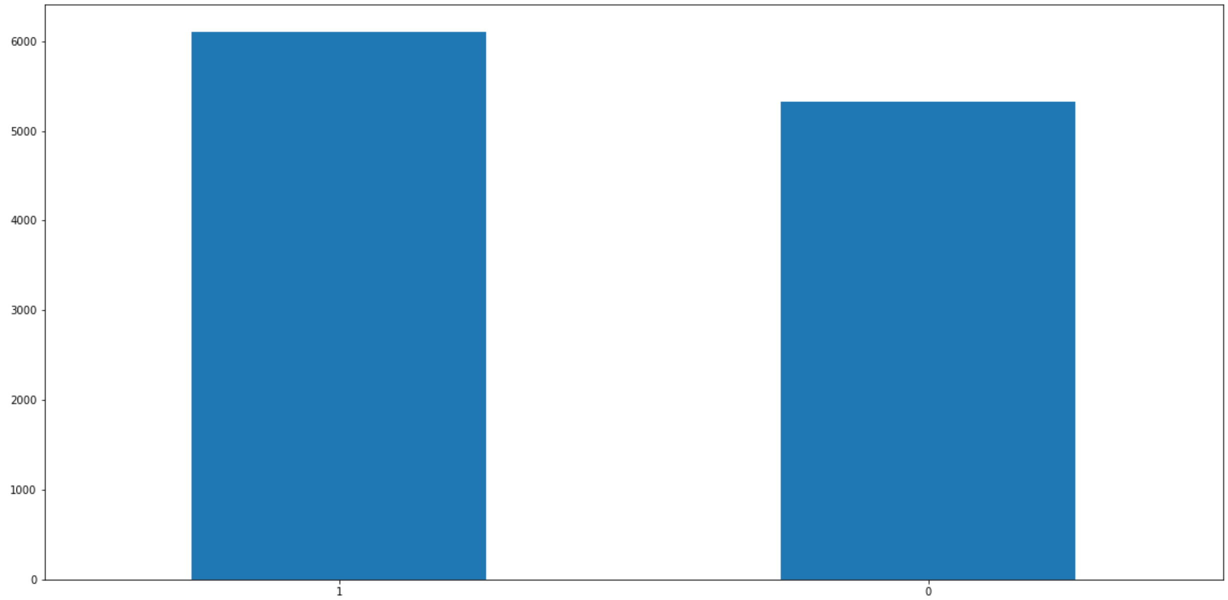
In [5]:
```
# Setting paramters for matplotlib graphs to default to this size
plt.rcParams['figure.figsize'] = (20,10)
```

The target variable in this project is going to be the `status` column, which has string values of `phishing` and `legitimate`.

In [6]:
```
# Visualizing initial split of our target variable. There is an even split betwee
# so there is no class imbalance.
df['status'].value_counts().plot(kind='bar', rot = 0);
```



```
In [7]: # Checking the distribution of the google_index feature
df['google_index'].value_counts().plot(kind='bar', rot = 0);
```



Here I count the values in each feature of the dataset just to get a better understanding of what outliers or common values there may be. The code block is commented out since the output is exceedingly large.

```
In [8]: # Created simple for loop to print out values in each column just to visualize ar
## Commented this out since the output is very lengthy
# for c in df.columns:
#    print("---- %s ---" % c)
#3    print(df[c].value_counts())
```

Here I check to see what some URLs may look like based on their features identified in the value counts.

```
In [9]: # Exploring the data based on the value counts above just to visualize what some
## Commenting this line out to reduce clutter of notebook
# df.loc[df['google_index'] == 0]
```

After completing preliminary EDA, the columns stored in the `columns_to_drop` list are dropped from the dataset because thy have blank entries and are not useful for the scope of my project.

```
In [10]: # Dropped these columns because all the values present in these columns were 0
columns_to_drop = ['nb_or', 'nb_space', 'submit_email','ratio_intRedirection','ra
# Stored the list of strings for the phish_hints feature as a reference.
# This list was located in the Python script from the authors of the dataset.
HINTS = ['wp', 'login', 'includes', 'admin', 'content', 'site', 'images', 'js', '
df_dropped = df.copy().drop(columns = columns_to_drop)
```

After dropping these data values from the dataframe, I change the object values of the target feature `status` to a binary classification where `legitimate = 0` and `phishing = 1`.

In [11]:
```
# Since this is a classification project, I am changing the values in the status
# and the values of phishing: 1
df_label = pd.DataFrame(df_dropped['status'].copy())
phish = df_label.replace({"status": {"legitimate" : 0,
                                     "phishing" : 1}})
df_dropped['status'] = phish
```

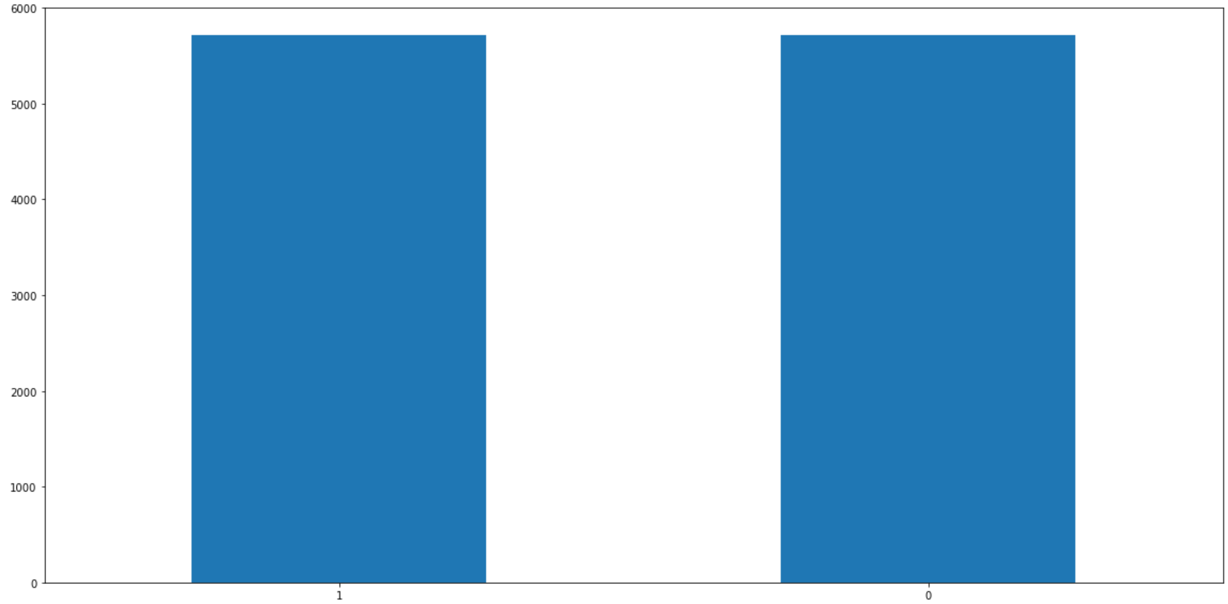
In [12]:
```
# Checking to see that the features were dropped correctly
df_dropped
```

Out[12]:

	url	length_url	length_hostname	ip	nb_dots	nb
0	http://www.crestonwood.com/router.php	37	19	0	3	
1	http://shadetreetechnology.com/V4/validation/a...	77	23	1	1	
2	https://support-appleld.com.secureupdate.duila...	126	50	1	4	
3	http://rgipt.ac.in	18	11	0	2	
4	http://www.iracing.com/tracks/gateway-motorspo...	55	15	0	2	
...	...	...	...	...	...	
11425	http://www.fontspace.com/category/blackletter	45	17	0	2	
11426	http://www.budgetbots.com/server.php/Server%20...	84	18	0	5	
11427	https://www.facebook.com/Interactive-Televisio...	105	16	1	2	
11428	http://www.mypublicdomainpictures.com/	38	30	0	2	
11429	http://174.139.46.123/ap/signin?openid.pape.ma...	477	14	1	24	

11430 rows × 82 columns

In [13]:
```
# Checking to see that the values were changed correctly
df_dropped['status'].value_counts().plot(kind='bar', rot = 0);
```



In [14]:
```
# Created a copy of dataframe where URLs are just replaced with their correspondi
# Keeping the URLs in separate dataframe for reference
dropped_url_df = df_dropped.copy()
dropped_url_df['url'] = dropped_url_df.index
```

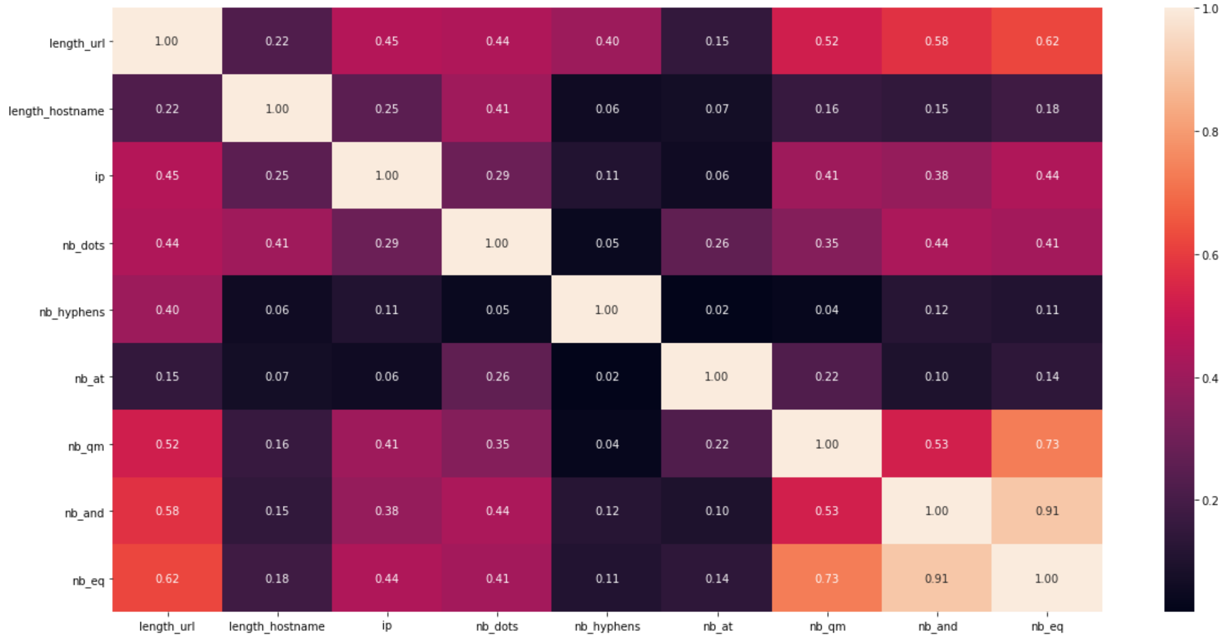
Some functions to be used throughout project

Below I define some functions that will be utilized throughout my project

In [15]:
```
# Function that plots correlation heatmap in batches since there are so many init
def partial_heatmap(data, start, stop):
    y = data['status']
    df = data.iloc[:, start:stop]
    sns.heatmap(df.corr(), annot=True, fmt='.2f')
    plt.show()

# Function taken from previous group project with Andrew Choi and Nicholas Wertz
# Function that prints out training/test scores for each metric of training and t
def score_matrix_printer(model, X_train, y_train, X_test, y_test):
    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    # Cleaning up scores to be more visually appealing
    ascore_train = round((accuracy_score(y_train, train_pred) * 100), 2)
    rscore_train = round((recall_score(y_train, train_pred) * 100), 2)

    ascore_test = round((accuracy_score(y_test, test_pred) * 100), 2)
    rscore_test = round((recall_score(y_test, test_pred) * 100), 2)

    conf_mat = plot_confusion_matrix(model, X_test, y_test)
    roc_cirve = plot_roc_curve(model, X_test, y_test)

    print(f"""
    Train Accuracy: {ascore_train}%
    Train Recall: {rscore_train}%
    ------------------------------
    Test Accuracy: {ascore_test}%
    Test Recall: {rscore_test}%
    """)

# Function to get OLS stat summary for significant P-values
def get_stats(x_columns):
    x = df[x_columns]
    results = sm.OLS(y, X).fit()
    print(results.summary())
```

After initial EDA and the functions were defined, I start exploring the correlation between the variables. I initially start with a correlation heatmap as that is the simplest correlation metric to identify. The heatmaps are created in such a way that it shows the features of specific indices so that it produces a visualization that is easily interpreted.

In [16]:
```
partial_heatmap(df_dropped, 0, 10)
```



In [17]:
```
## Since heatmap visualizations are just for data exploration
## the outputs for following heatmaps are collapsed in order to save space in the
# partial_heatmap(df_dropped, 10, 20)
```

In [18]:
```
## Since heatmap visualizations are just for data exploration
## the outputs for following heatmaps are collapsed in order to save space in the
# partial_heatmap(df_dropped, 30, 40)
```

In [19]:
```
## Since heatmap visualizations are just for data exploration
## the outputs for following heatmaps are collapsed in order to save space in the
#partial_heatmap(df_dropped, 40, 50)
```

In [20]:
```
## Since heatmap visualizations are just for data exploration
## the outputs for following heatmaps are collapsed in order to save space in the
# partial_heatmap(df_dropped, 50, 60)
```

In [21]:
```
## Since heatmap visualizations are just for data exploration
## the outputs for following heatmaps are collapsed in order to save space in the
# partial_heatmap(df_dropped, 60, 70)
```

In [22]:
```
## Since heatmap visualizations are just for data exploration
## the outputs for following heatmaps are collapsed in order to save space in the
# partial_heatmap(df_dropped, 70, 82)
```

After investigating the Pearson coefficient between the variables in our dataset, we recognize that some multicollinearity exists between some variables; however, I also recognize that the Pearson coefficient is not as strong of a correlation comparison metric. Next, I will be

Stepwise selection for feature importance

After exploring the collinearity of the features, I wanted to move forward with stepwise selection to identify features that would be significant to my data.
Initially, the dataset had 87 features. I dropped a couple of features in my preliminary data exploration since those columns had 0 values in them. This dropped my number of features down to 82; however, I still wanted to minimize the number of features in the dataset so I could focus on what features would be most important in identifiying a phishing website.

In [23]:
```
# Creating a list of the column names
x_columns = df.columns.tolist()
X = dropped_url_df
y = dropped_url_df['status']
```

Next, I'm going to run a summary statistics to check if any values are non-significant to my data in order to reduce the number of overall features.

In [24]:
```
## Commenting block line of code out since the output is lengthy
# get_stats(x_columns)
```

In [25]:
```
# Based on the statistical report, these features had a p-values of > 0.05 thus t
features_to_drop = ['url', 'length_url', 'nb_hyphens', 'nb_and', 'nb_underscore',
'nb_com', 'nb_dslash', 'http_in_path', 'punycode', 'tld_in_path', 'tld_in_subdoma
'prefix_suffix', 'random_domain', 'path_extension', 'char_repeat', 'shortest_word
'longest_words_raw', 'longest_word_host', 'avg_words_raw', 'avg_word_host', 'bran
'statistical_report', 'nb_extCSS', 'ratio_extErrors', 'login_form', 'links_in_tag
'onmouseover', 'right_clic', 'web_traffic']

stepwisedf = dropped_url_df.drop(columns = features_to_drop)
```

In [26]:
```
# Printing out .info of the new stepwisedf to see how many features remain
# Collapsing output in final notebook
stepwisedf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11430 entries, 0 to 11429
Data columns (total 43 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   length_hostname             11430 non-null  int64  
 1   ip                          11430 non-null  int64  
 2   nb_dots                     11430 non-null  int64  
 3   nb_at                       11430 non-null  int64  
 4   nb_qm                       11430 non-null  int64  
 5   nb_eq                       11430 non-null  int64  
 6   nb_percent                  11430 non-null  int64  
 7   nb_slash                    11430 non-null  int64  
 8   nb_colon                    11430 non-null  int64  
 9   nb_www                      11430 non-null  int64  
 10  https_token                 11430 non-null  int64  
 11  ratio_digits_url            11430 non-null  float64
 12  ratio_digits_host           11430 non-null  float64
 13  port                        11430 non-null  int64  
 14  nb_subdomains               11430 non-null  int64  
 15  shortening_service          11430 non-null  int64  
 16  nb_redirection              11430 non-null  int64  
 17  nb_external_redirection     11430 non-null  int64  
 18  length_words_raw            11430 non-null  int64  
 19  shortest_word_path          11430 non-null  int64  
 20  longest_word_path           11430 non-null  int64  
 21  avg_word_path               11430 non-null  float64
 22  phish_hints                 11430 non-null  int64  
 23  domain_in_brand             11430 non-null  int64  
 24  suspecious_tld              11430 non-null  int64  
 25  nb_hyperlinks               11430 non-null  int64  
 26  ratio_intHyperlinks         11430 non-null  float64
 27  ratio_extHyperlinks         11430 non-null  float64
 28  ratio_extRedirection        11430 non-null  float64
 29  external_favicon            11430 non-null  int64  
 30  ratio_intMedia              11430 non-null  float64
 31  ratio_extMedia              11430 non-null  float64
 32  safe_anchor                 11430 non-null  float64
 33  empty_title                 11430 non-null  int64  
 34  domain_in_title             11430 non-null  int64  
 35  domain_with_copyright       11430 non-null  int64  
 36  whois_registered_domain     11430 non-null  int64  
 37  domain_registration_length  11430 non-null  int64  
 38  domain_age                  11430 non-null  int64  
 39  dns_record                  11430 non-null  int64  
 40  google_index                11430 non-null  int64  
 41  page_rank                   11430 non-null  int64  
 42  status                      11430 non-null  int64  
dtypes: float64(9), int64(34)
memory usage: 3.7 MB
```

The amount of features decreased from 82 to 42 following the stepwise selection. This reduction is very beneficial to reducing the complexity of the model. Next, I would like to focus on the coefficients of the initial Logistic Regression model as well as utilize an ExtraTreesClassifier to identify the top 10 features of my dataset in order to reduce even more model complexity.

In [27]:
```
# Creating X and y variables for initial train/test split.
# This train/test split is based on the
stepwise_X = stepwisedf.drop(columns = 'status')
stepwise_y = stepwisedf['status']
X_train, X_test, y_train, y_test = train_test_split(stepwise_X, stepwise_y, rando
```

Based on an alpha value of significance of 0.05, we were able to drop our features from 82 features to 42 features. Based on the heat maps generated above of the Pearson coefficients, I want to investigate the multi-collinearity that is present in my dataset.
Although we investigated the collinearity with the heatmaps, I will be looking into the Variance Inflation Factor(VIF) next since the VIF investigates the variance between our features. I am choosing to utilize VIF over the Pearson correlation heatmaps since VIF focuses on the correlation of one feature to the other features vs the Pearson correlation of one feature to another feature. My intent with using VIF is to address the variables that have high correlation with one another and reduce the overall complexity of my model.

VIF Exploration

In [28]:
```
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(len(X_t
vif["features"] = X_train.columns
vif.sort_values(by="VIF",ascending=True, inplace=True)
vif.head(50)
```

Out[28]:

	VIF	features
24	1.077118	suspecious_tld
13	1.107718	port
3	1.180498	nb_at
36	1.262736	whois_registered_domain
15	1.505592	shortening_service
6	1.513213	nb_percent
39	1.526127	dns_record
17	1.528063	nb_external_redirection
28	1.583370	ratio_extRedirection
37	1.596704	domain_registration_length
25	1.637176	nb_hyperlinks
22	1.709048	phish_hints
23	1.908723	domain_in_brand
16	1.924618	nb_redirection
35	2.131197	domain_with_copyright
12	2.196959	ratio_digits_host
31	2.386448	ratio_extMedia
19	2.553598	shortest_word_path
9	2.615972	nb_www
33	2.702783	empty_title
4	2.915354	nb_qm
32	2.921611	safe_anchor
29	2.990702	external_favicon
10	3.258514	https_token
1	3.448425	ip
30	3.607822	ratio_intMedia
40	3.958852	google_index
5	4.061410	nb_eq
20	4.423074	longest_word_path
11	4.428427	ratio_digits_url
38	4.887976	domain_age
34	5.608637	domain_in_title

	VIF	features
21	5.863751	avg_word_path
27	6.544402	ratio_extHyperlinks
41	6.825909	page_rank
0	7.063935	length_hostname
26	12.586698	ratio_intHyperlinks
18	12.718791	length_words_raw
2	13.551156	nb_dots
7	16.052514	nb_slash
8	20.816337	nb_colon
14	24.782550	nb_subdomains

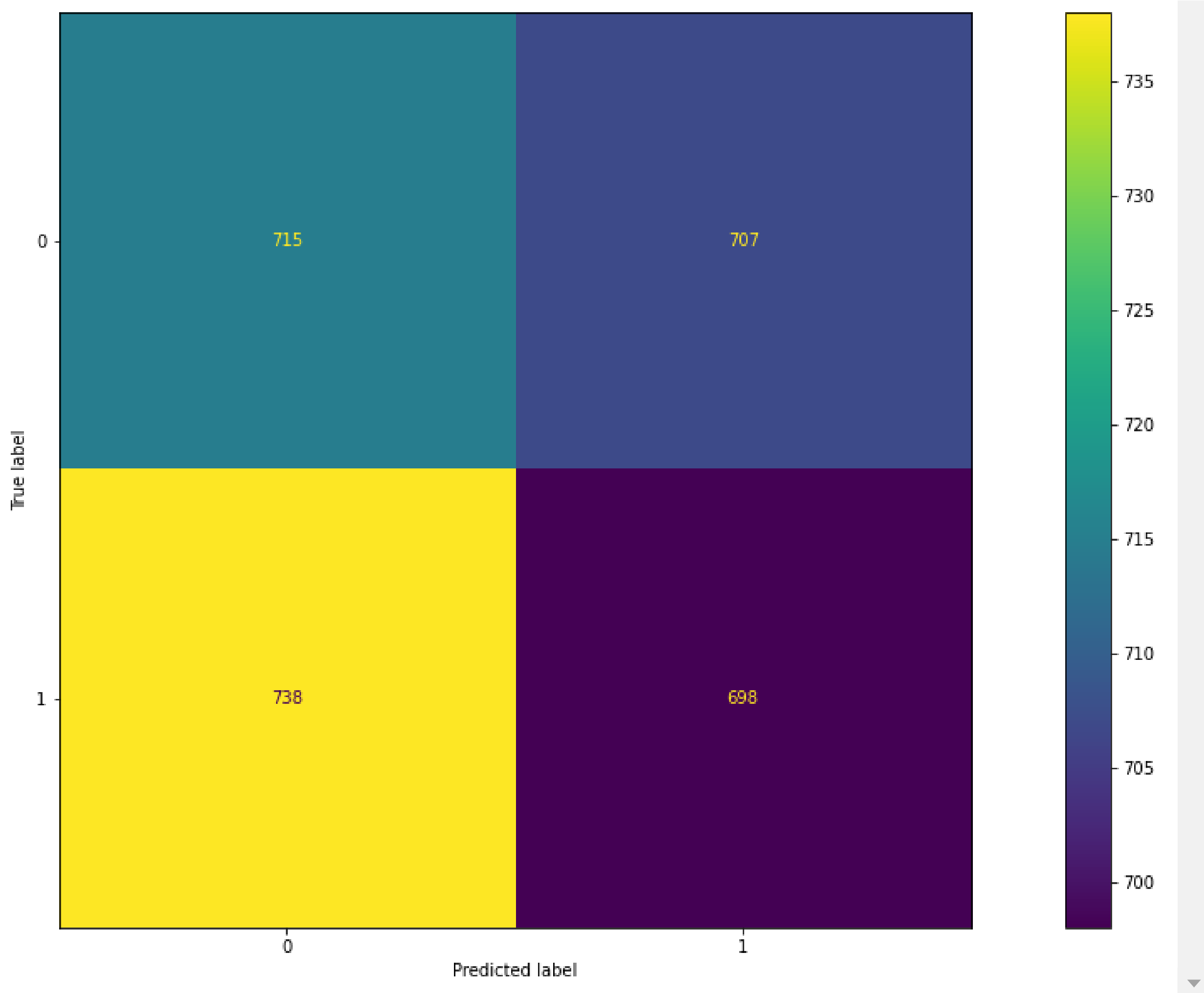
The resutls of the VIF indicated that there were 7 variables that had a score of 10 or higher. Since these variables have a VIF score > 10, that indicates that there is high multicollinearity with these variables. Since this is the case, I will not be using these 7 variables in the modeling process, and I will be focusing on the rest of the features that have a VIF score < 10 which indicates that they are unique independent variables.

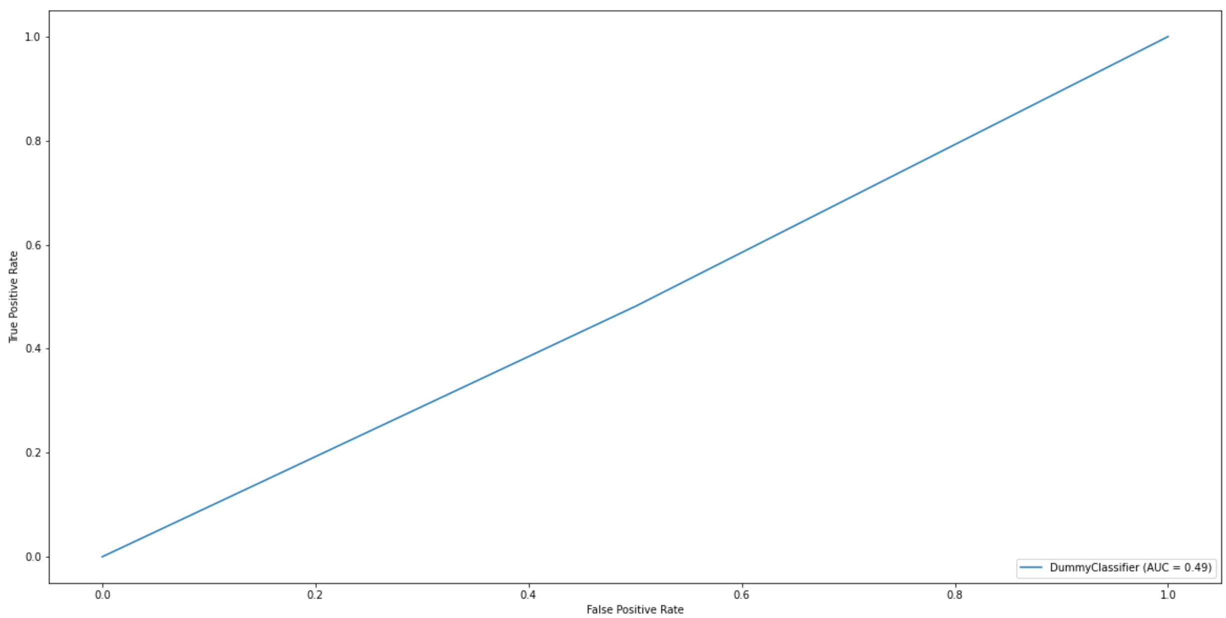
Baseline model (Dummy Classifier)

Since this is a classification project, I chose a DummyClassifier as my baseline model. The DummyClassifier is expected to guess whether a website is a phishing website or legitimate 50% of the time.

In [29]:
```
# Creating a new train/test split based on the features isolated from stepwise se
new_X = stepwise_X.drop(columns = ['ratio_intHyperlinks', 'length_words_raw', 'nb
new_y = stepwise_y

new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(new_X, new_y,
# Dummy Classifier as baseline model
dummy = DummyClassifier()
dummy.fit(new_X_train, new_y_train)
y_pred = dummy.predict(new_X_train)
y_test_pred = dummy.predict(new_X_test)
y_pred_df = pd.DataFrame(y_pred)
dummy.score(new_X_test, new_y_test)
score_matrix_printer(dummy, new_X_train, new_y_train, new_X_test, new_y_test);
```

```
C:\Users\Beter\anaconda3\envs\learn-env\lib\site-packages\sklearn\dummy.py:131:
FutureWarning: The default value of strategy will change from stratified to pri
or in 0.24.
  warnings.warn("The default value of strategy will change from "


    Train Accuracy: 50.15%
    Train Recall: 49.87%
    ------------------------------
    Test Accuracy: 49.55%
    Test Recall: 49.37%
```





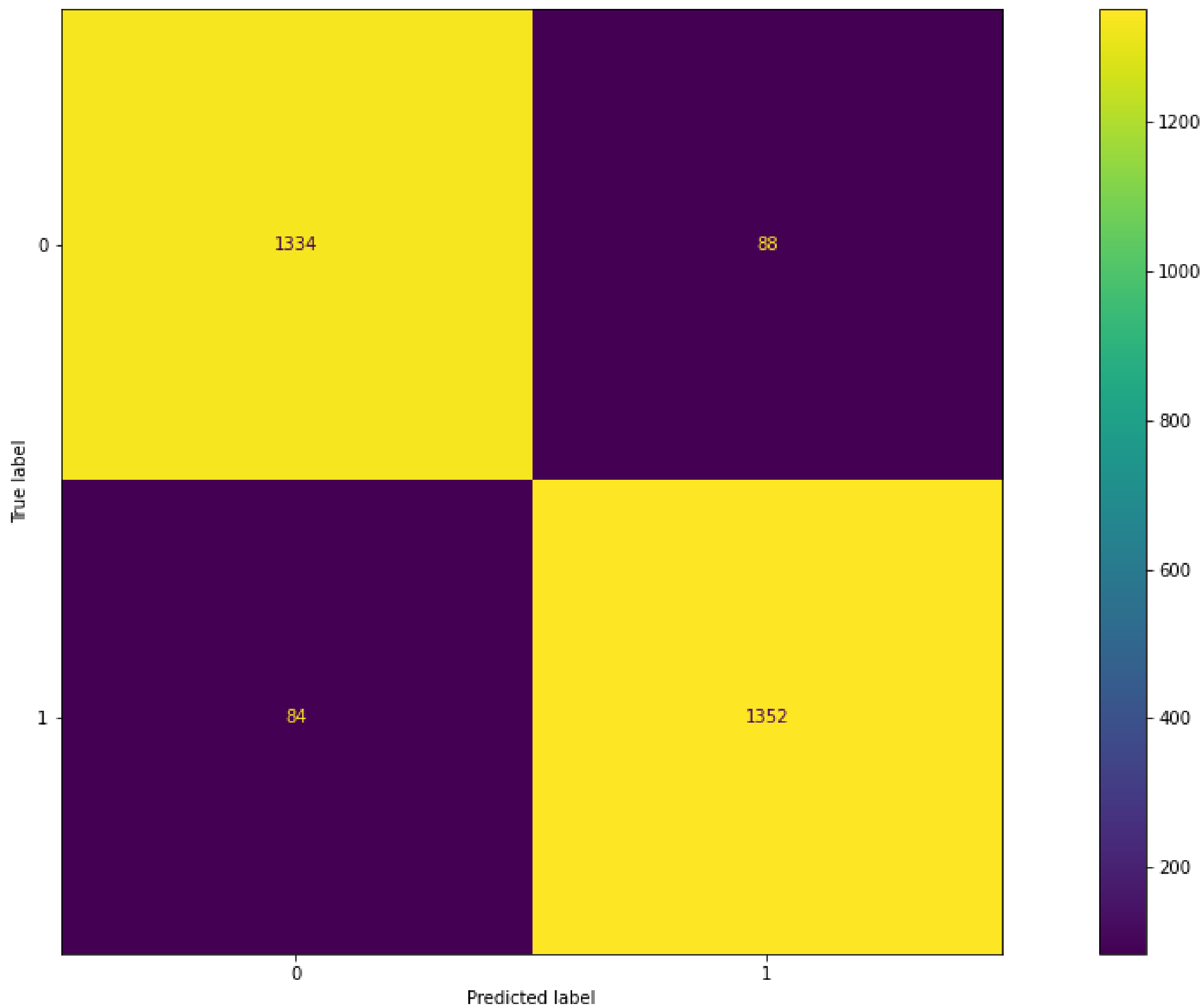
The baseline model produced a score of ~50% for both the accuracy and the recall, which is what we expected. I am choosing to focus on accuracy because the overall accuracy of the model is important in correctly identifying a phishing website. I am also choose to focus on recall because the recall score helps correctly identify a false positive in the dataset. A false positive with respect to this project would essentially be "falling for the phishing tactic" -- meaning that the model incorrectly identified the data as a legitimate website when in reality it should be labeled as a phishing website.

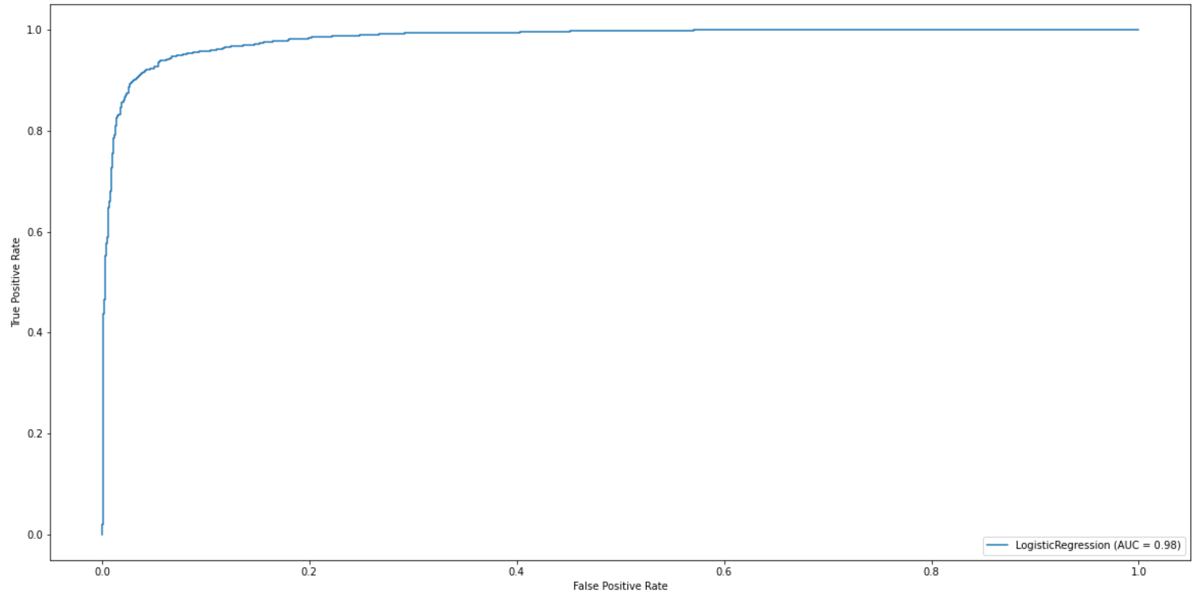
Model 1: Logistic Regression Model

In [30]:
```
# Simple Logistic Regression Model
# Set max_iter hyperparameter = 1000 since there are so many initial features
# Will likely be dropping/aggregating columns since there are a few features with
lr = LogisticRegression(max_iter = 100000)
lr.fit(new_X_train, new_y_train)
lr_preds = lr.predict(new_X_test)
```

In [31]:
```
score_matrix_printer(lr, new_X_train, new_y_train, new_X_test, new_y_test)
```

```
Train Accuracy: 93.54%
Train Recall: 93.5%
------------------------------
Test Accuracy: 93.98%
Test Recall: 94.15%
```





In [32]:

```
# Since the coefficients are in log-odds, take the exponent to see the odds of ea
oddscoef = np.exp(lr.coef_)
# Put the column names and coefficients into a dataframe for ease of access
result = pd.concat([(pd.DataFrame(new_X.columns)), (pd.DataFrame(oddscoef).transp
# Resetting column values to 0, 1
result.columns = range(result.columns.size)
# Sorting dataframe by the coefficient values
result.sort_values(by = 1, ascending = False, inplace=True)
# Selecting top 10 values of log coefficients
top_10_coefs = result.iloc[:10]
# Converting feature names to a list to run a model later based on these coeffici
list_of_top_10_coefs = list(top_10_coefs[0])
top_10_coefs
```

Out[32]:

	0	1
34	google_index	18.934808
17	phish_hints	4.921076
3	nb_qm	3.818450
21	ratio_extHyperlinks	3.735669
11	shortening_service	3.437516
1	ip	2.843089
28	domain_in_title	2.444356
9	ratio_digits_host	2.156122
33	dns_record	1.710949
19	suspecious_tld	1.664519

In our dataset, a 0 indicates that the `google_index` of a page is present. If the data has the lack of a `google_index` , that means that the website is roughly 20x as likely to be a phishing website. Similarly, the data also has a `phish_hints` feature. `phish_hints` refers to characteristics of these websites such as urgent action items, poor grammer or misspelled words, offers that are too good to be true, etc. If the data has any characteristics of these `phish_hints` from the `HINTS` variable defined earlier in the project, it is roughly 5x as likely to be a phishing website. `nb_qm` is the number of question marks that are present in a url. As the number of question marks increases by one, the url becomes 4x as likely to be a phishing website.

Creation of Pipelines

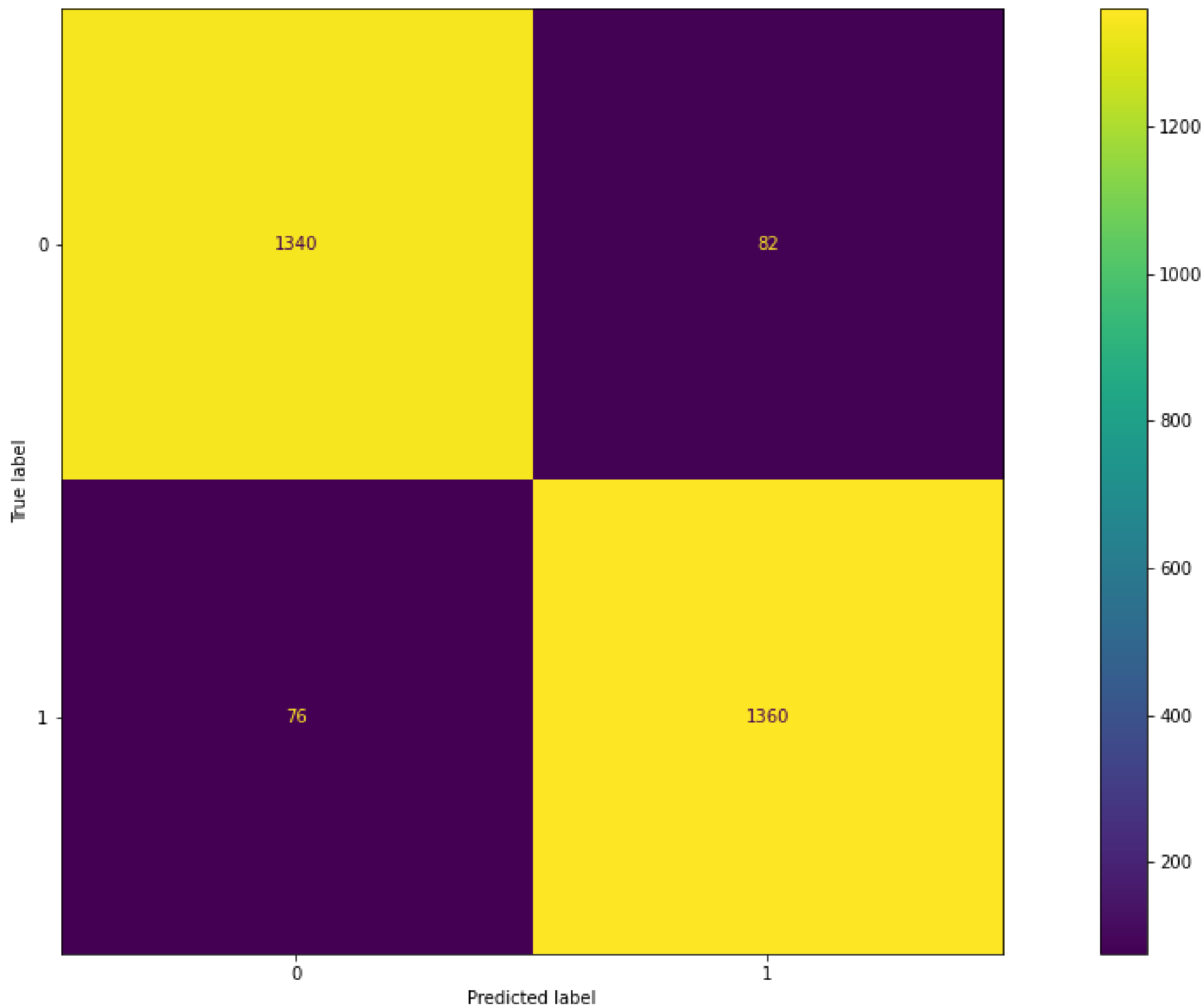
Implemented a linear regression pipeline that will be used throughout the rest of project.

In [33]:
```
# LogisticRegression Pipeline
lrpipe = Pipeline(steps=[
    ('ss', StandardScaler()),
    ('lr', LogisticRegression())
])
```

Model 2: Pipeline Integration

In [34]:
```
# LogisticRegression pipeline with StandardScalar step
lrpipe.fit(new_X_train, new_y_train)
lr_pipe_preds = lrpipe.predict(new_X_train)
score_matrix_printer(lrpipe, new_X_train, new_y_train, new_X_test, new_y_test)
```

```
Train Accuracy: 93.69%
Train Recall: 93.53%
------------------------------
Test Accuracy: 94.47%
Test Recall: 94.71%
```



The results of the logistic regression pipeline with the StandardScalar step produced a very similar model to that of the initial Logistic Regression model. Next I will attempt to reduce the number of features by selecting the important features through an ExtraTreeClassifier.

Running GridSearchCV to Find Optimal Hyperparameters

Next, I will be checking the optimal hyperparameters for the Logistic Regression model using a GridSearchCV.

In [35]:
```
# Commenting out this GridSearchCV because it took ~10 minutes to run
# param_grid = {
#     "lr__penalty":['l1', 'l2', 'elasticnet', 'none'],
#     "lr__solver":['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
#     "lr__max_iter":[1000, 10000, 50000, 100000]
# }

# grid = GridSearchCV(lrpipe, param_grid)
# grid.fit(X_train, y_train)
```

In [36]:
```
# print(grid.best_params_)
# print(grid.best_score_)
```

In [37]:

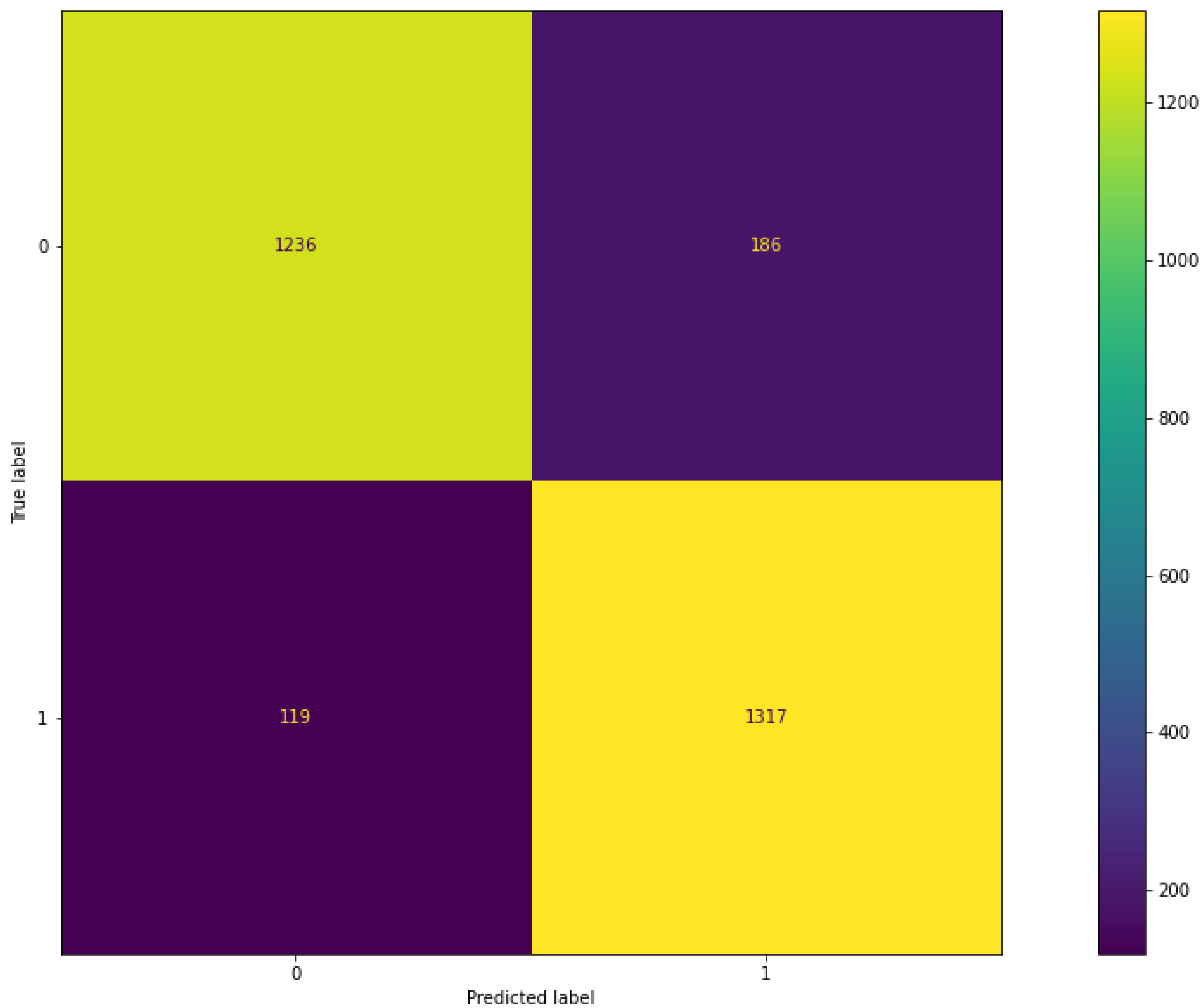
```
# Reinstantiating pipeline for LogisticRegression with new optimized hyperparamet
newlrpipe = Pipeline(steps=[
    ('ss', StandardScaler()),
    ('lr', LogisticRegression(max_iter = 100000, penalty = 'l1', solver = 'liblin
])
```

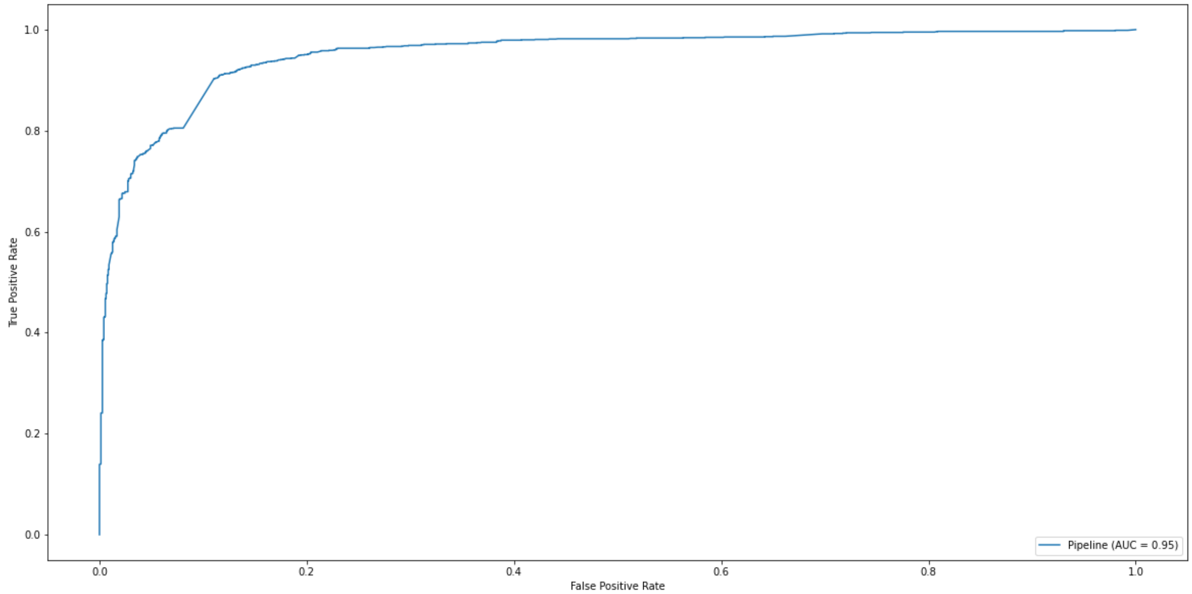
Model 3: Logistic Regression with top 10 coefficients

After identifying the top coefficients from the initial Logistic Regression model, I wanted to see if those identified features would be important in identifying a phishing website or not.

In [38]:
```python
coefX = stepwise_X[list_of_top_10_coefs]

coefX_train, coefX_test, coefy_train, coefy_test = train_test_split(coefX, new_y,

newlrpipe.fit(coefX_train, coefy_train)
lr_coef_preds = newlrpipe.predict(coefX_train)
score_matrix_printer(newlrpipe, coefX_train, coefy_train, coefX_test, coefy_test)
```

```
Train Accuracy: 88.89%
Train Recall: 90.75%
------------------------------
Test Accuracy: 89.33%
Test Recall: 91.71%
```





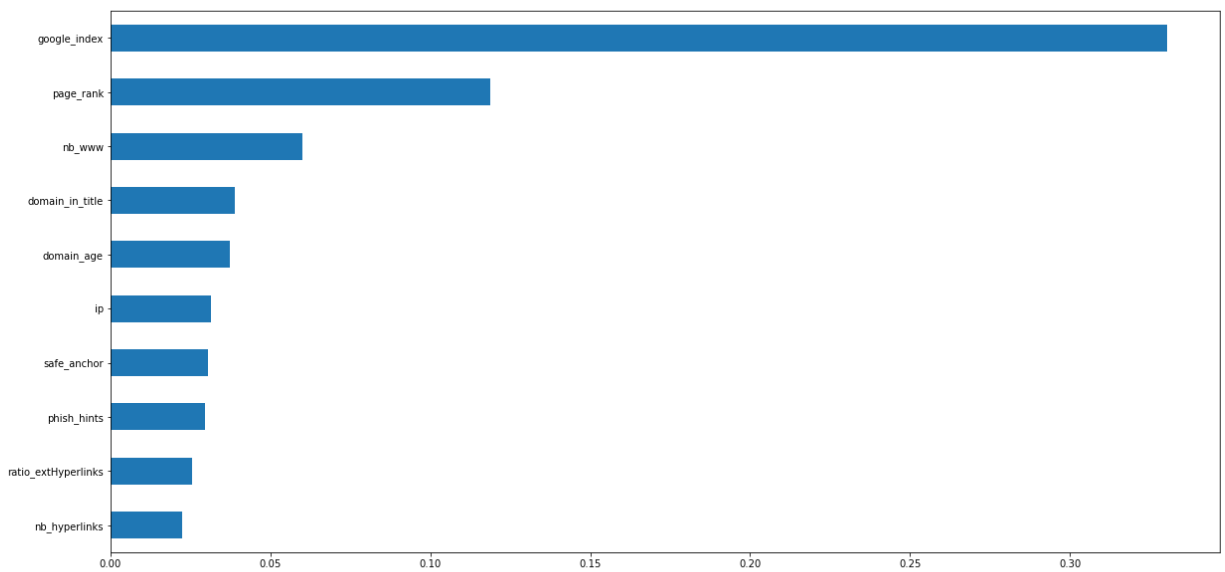
The results indicate that focusing on the features from the Logistic Regression model that had the highest coefficients values produced a overall lower accuracy and recall score. Although the score is high compared to the baseline model, there may be a few features that are more important in detecting a phishing website that the coefficients cannot indicate. Although the odds-ratio values do not produce the highest model score, they are still important in the interpretation of the models. Next I will utilize an ExtraTreeClassifier in order to identify the important features.

Feature Importance (ExtraTreeClassifier)

Next an ExtraTreeClassifier model is ran in order to identify the 10 most important features in the dataset from the remainding 42. Since the data is not normally distributed, I will utilize the ExtraTreeClassifiers because it deals with nonparametric data. The goal is to once again reduce the number of features in the model to then reduce the overall complexity of the model.

In [39]:
```
# Instantiate model
modelfeatures = ExtraTreesClassifier()
modelfeatures.fit(new_X, new_y)
print(modelfeatures.feature_importances_) # use built in class 'feature_importanc
# Plot graph of feature importances for better visualization
feat_importances = pd.Series(modelfeatures.feature_importances_, index=new_X.colu
feat_importances.nlargest(10).plot(kind='barh')
plt.gca().invert_yaxis()
plt.show()
```

```
[1.85308225e-02 3.15252559e-02 2.43782970e-03 1.54886244e-02
 7.92342329e-03 3.99167802e-03 6.01256865e-02 8.01044539e-03
 1.79099404e-02 1.08658594e-02 8.92692960e-04 1.25116413e-02
 1.15881176e-02 4.58815455e-05 1.23132218e-02 1.53598290e-02
 1.46311057e-02 2.94913513e-02 8.58357337e-03 3.40391199e-03
 2.25156041e-02 2.55183626e-02 1.44308276e-02 9.74628747e-03
 1.66370607e-02 1.38928003e-02 3.06079161e-02 1.45795149e-02
 3.87304983e-02 1.88964398e-02 3.82538890e-03 1.59118837e-02
 3.74210729e-02 2.50494786e-03 3.30381783e-01 1.18768719e-01]
```



In [40]:
```
print(feat_importances.nlargest(10))
```

```
google_index           0.330382
page_rank              0.118769
nb_www                 0.060126
domain_in_title        0.038730
domain_age             0.037421
ip                     0.031525
safe_anchor            0.030608
phish_hints            0.029491
ratio_extHyperlinks    0.025518
nb_hyperlinks          0.022516
dtype: float64
```

In [41]:
```
# Created a list of top features from ExtraTreeClassifier
list_extraTree_features = ['domain_age', 'google_index', 'page_rank', 'domain_in_
    'ratio_intHyperlinks', 'ratio_extHyperlinks', 'nb_www', 'safe_anchor', 'phish

# Created a dataframe with the features identified from ExtraTreeClassifier
extraTreeX = X[list_extraTree_features]
extraTreeX

# Train/test split for top features from ExtraTreeClassifier
extra_X_train, extra_X_test, extra_y_train, extra_y_test = train_test_split(extra
```

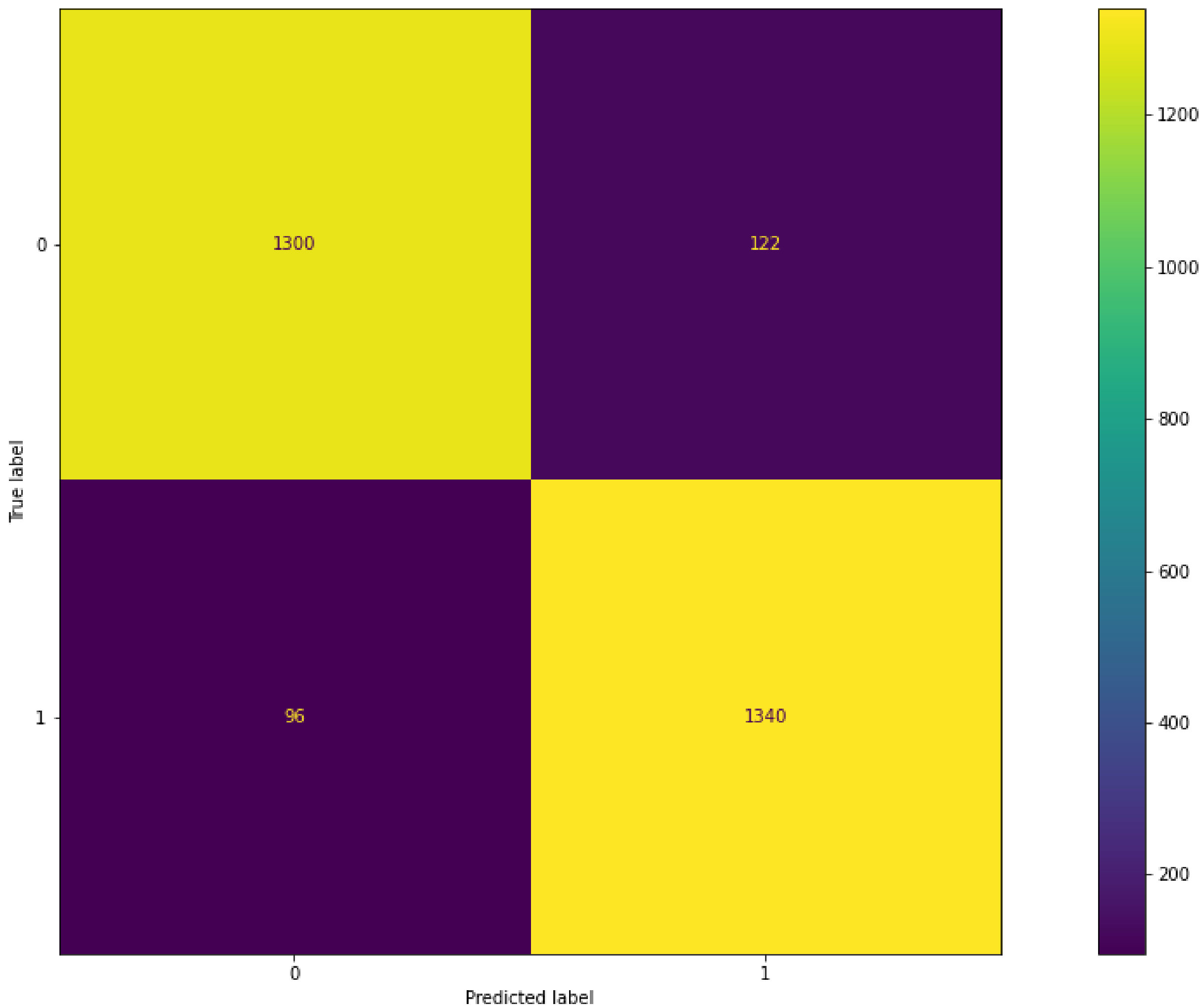
This feature importance graph shows most important non-parametric features that may be important for detecting a phishing website. A list with these features and will be used as the focused features in the next model.

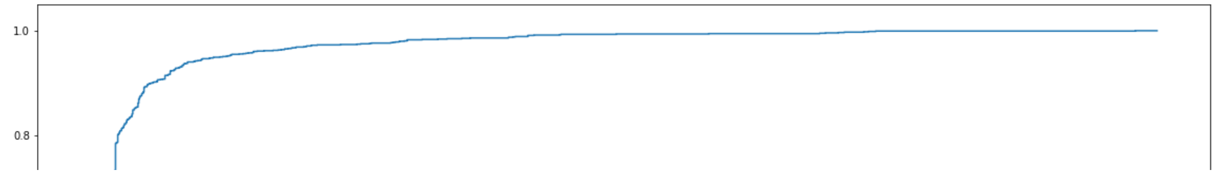
Model 4: Pipeline with ExtraTree features

Using the features that I identified from the code above, another LogisticRegression model is ran that solely focuses on the features that were identified in the ExtraTreeClassifier that are stored in the `list_extraTree_features` variable.

In [42]:
```
# Pipeline w/ StandardScalar and LogisticRegression steps w/ features selected fr
newlrpipe.fit(extra_X_train, extra_y_train)
lr_pipe_preds = newlrpipe.predict(extra_X_train)
score_matrix_printer(newlrpipe, extra_X_train, extra_y_train, extra_X_test, extra
```

```
Train Accuracy: 92.09%
Train Recall: 92.22%
------------------------------
Test Accuracy: 92.37%
Test Recall: 93.31%
```





After running a pipeline model based on the top 10 features that the ExtraTreeClassifier identified. This model that focused on the features stored in the `list_extraTree_features` variable produced an accuracy score of 92% and a recall score of 93% as well as having a AUC score of 0.97.

Although the overall accuracy and recall score dropped by about 2%, this is a tradeoff that I am willing to accept because we reduced the number of features in this dataset to 10 features from the stepwise selection of 42.

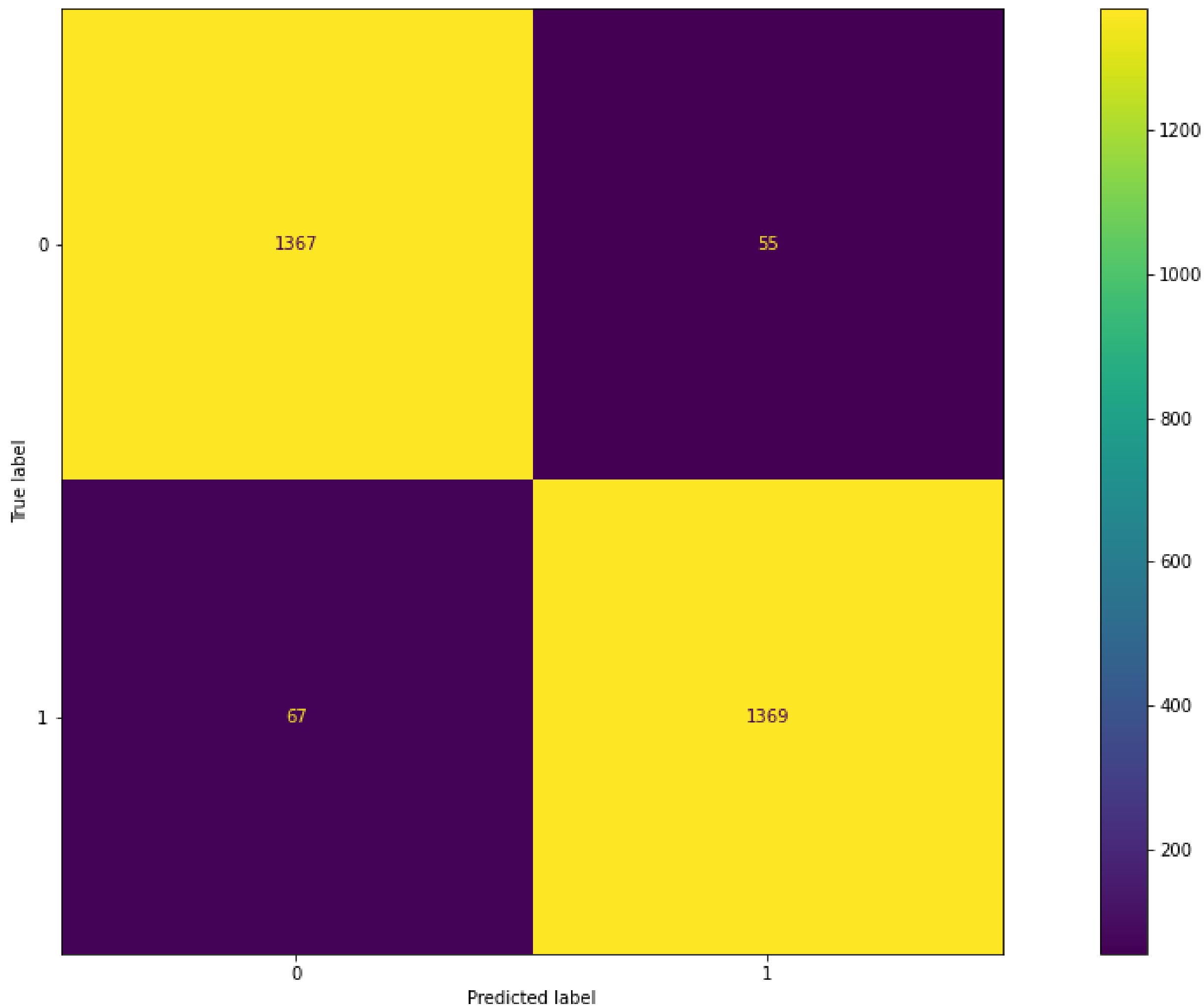
The reduction of 32 features strongly reduces the complexity of the model and is preferred over the overly complex model.

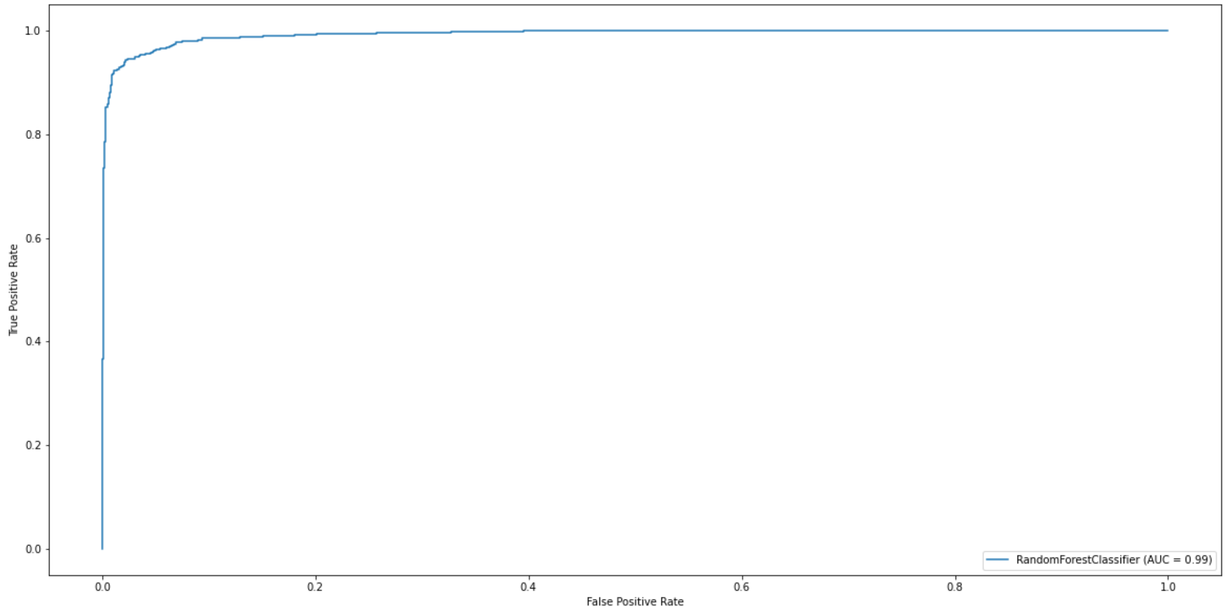
Model 5: Testing a RandomForestClassifier

Although the LogisticRegression model performance was already very impressive, I wanted to see if running a different kind of model would improve my metrics.

In [43]:
```
rfc = RandomForestClassifier(min_samples_split = 10, random_state=42)
rfc.fit(new_X_train, new_y_train)
rfc_preds = rfc.predict(new_X_train)
score_matrix_printer(rfc, new_X_train, new_y_train, new_X_test, new_y_test)
```

```
Train Accuracy: 98.57%
Train Recall: 98.57%
------------------------------
Test Accuracy: 95.73%
Test Recall: 95.33%
```





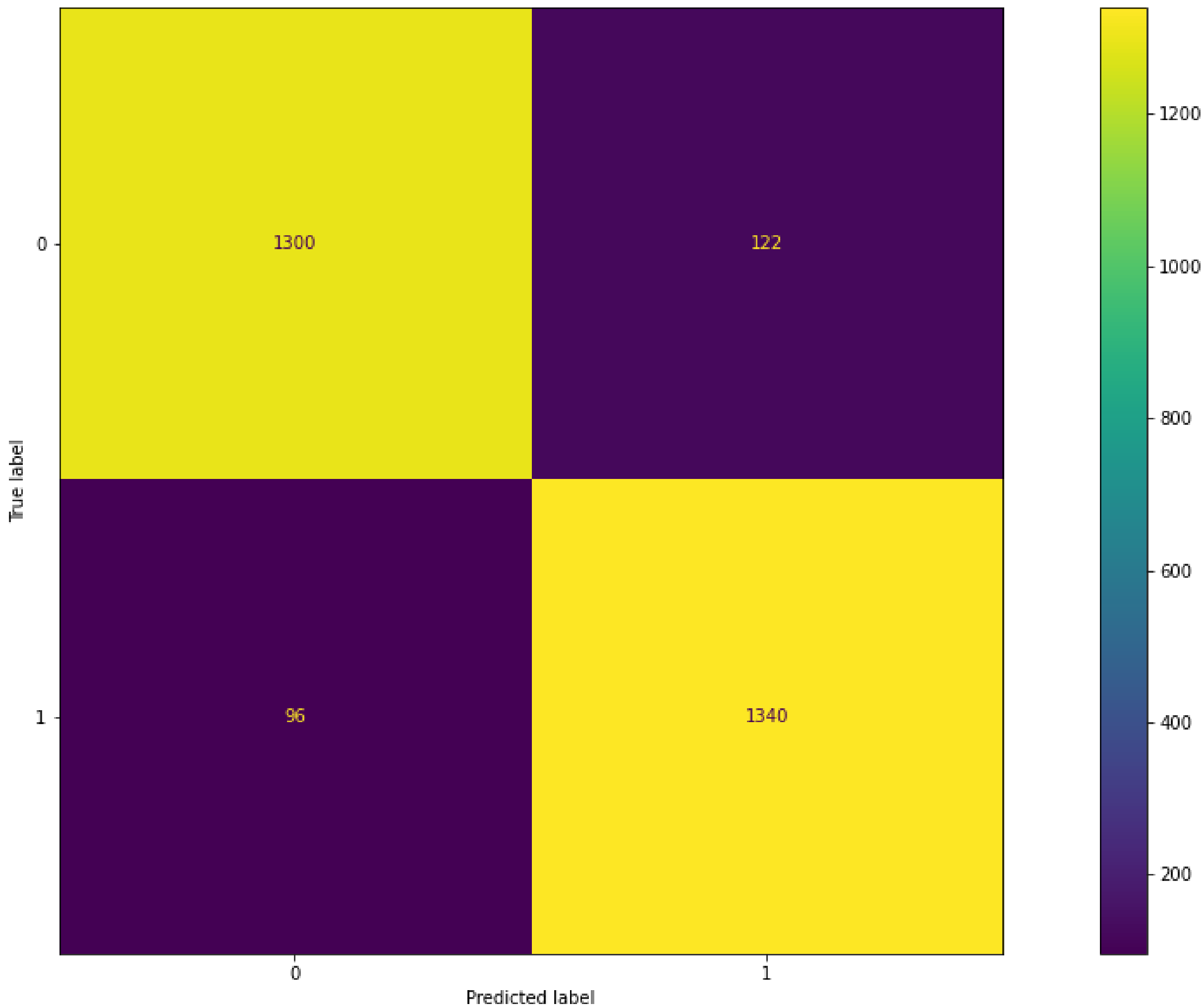
The results of the RandomForestClassifier indicate that the model overfits my data. I opted to stick with the LogisticRegression model with the ExtraTreeClassifier features and optimized hyperparameters as my final model.

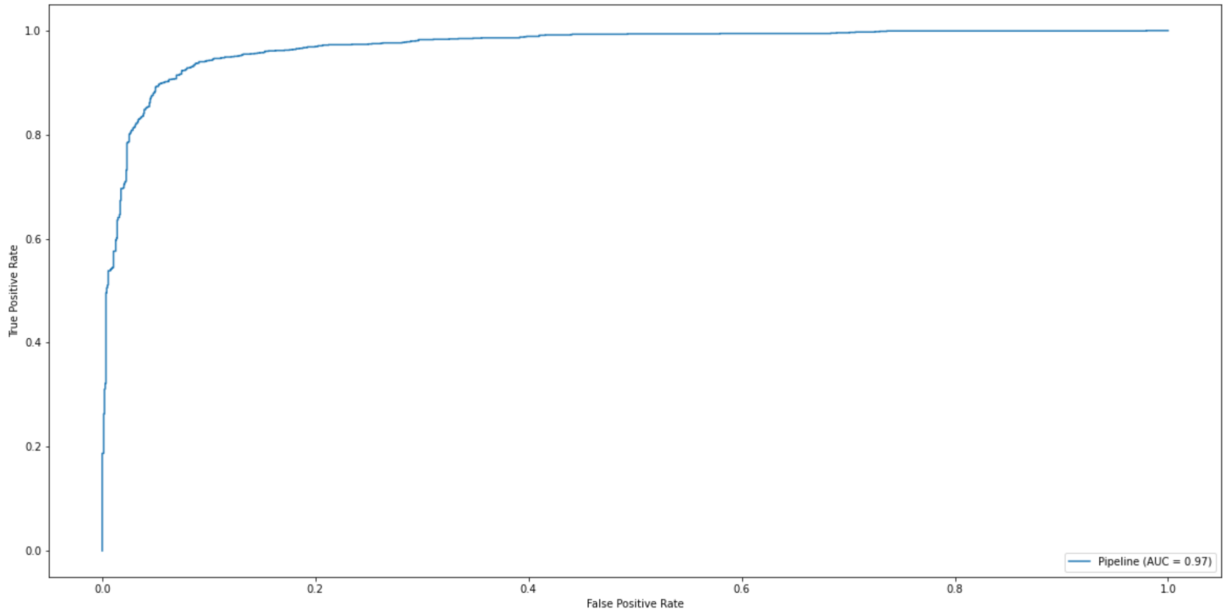
Evaluation

The model that performed best was the model that utilized the features identified through the ExtraTreeClassifier and the optimized hyperparamters from GridSearchCV. This model produced an accuracy score of **92%** and a recall score of **93%**.
The model also produces an AUC score of **0.97**, meaning that it has high classification accuracy.

In [44]:
```
# Printing out training and test scores as well as the confusion matrix and AUC c
score_matrix_printer(newlrpipe, extra_X_train, extra_y_train, extra_X_test, extra
```

```
Train Accuracy: 92.09%
Train Recall: 92.22%
------------------------------
Test Accuracy: 92.37%
Test Recall: 93.31%
```





Overall, this final model performs exceptionally well on the data. I would recommend using this model and its features to assist in identifying phishing websites. I was also able to identify some important features such as the `google_index` and the `phish_hints` . To reemphasize the point that I made earlier in the project, by looking at the log-odds if a feature lacks a `google_index` (recorded as a 1), then the website is roughly **20x** more likely to be a phishing website. Similarly, if a website has `phish_hints` features(list of strings defined by dataset creators, stored in the `HINTS` variable), it is roughly **5x** as likely to be a phishing website.

Conclusion

Overall, this final model has high potential in helping the AGSW in identifying phishing websites. This model can be used in tandem with the research that AGSW conducts to better identify phishing websites and protect those who would be susceptible to these kinds of cyber attacks. Based on the data, I would recommend that:

- Make sure the Google Index is accessible for people to utilize as a resource to check credibility
- Educate the population on common phishing website characteristics identified in the project so that they may be vigilant against cyber attacks.
- Stay up-to-date with new phishing techniques (such as brand impersonation, remote work surveys, fake IT emails, etc.)

Some future actions I would like to consider is:

- Utilize this model as a basis for tackling scam and phishing attacks that utilize text messages and phone calls instead of the traditional websites.
- Implement this model in a web program where people can input URLS and get an output of how likely a website is to be phishing or not.

- Utilize this model as a basis for phishing detection for other languages and nuances that may be country-specific.

Appendix

PCA

PCA was initially conducted to address the large amount of features; however, the first principal component accounted for ~99% of the variance in the data. The results were inconclusive.

In [45]:
```
'''# Initial PCA
pca = PCA(n_components=5)
pca.fit_transform(X_train)
print(pca.explained_variance_ratio_)
print(pca.components_)'''
```

Out[45]:
```
'# Initial PCA\npca = PCA(n_components=5)\npca.fit_transform(X_train)\nprint(pc
a.explained_variance_ratio_)\nprint(pca.components_)'
```