Isaiah Bagwell
CYSE 420
Professor Khallouli
10/6/2025

Code

```
import pandas as pd
from sklearn.feature_selection import VarianceThreshold
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

Python

```
df = pd.read_csv('PhiUSIIL_Phishing_URL_Dataset.csv')
df
```

Python

	FILENAME	URL	URLLength	Domain	DomainLength	IsDomainIP	TLD	URLSimilarityIndex	CharContinuationRate	TLDLegitimateProb	...	Pay	Crypto
0	521848.txt	https://www.southbankmosaics.com	31	www.southbankmosaics.com	24	0	com	100.000000	1.000000	0.522907	...	0	0
1	31372.txt	https://www.uni-mainz.de	23	www.uni-mainz.de	16	0	de	100.000000	0.666667	0.032650	...	0	0
2	597387.txt	https://www.voicefmradio.co.uk	29	www.voicefmradio.co.uk	22	0	uk	100.000000	0.866667	0.028555	...	0	0
3	554095.txt	https://www.sfnmjournal.com	26	www.sfnmjournal.com	19	0	com	100.000000	1.000000	0.522907	...	1	1
4	151578.txt	https://www.rewildingargentina.org	33	www.rewildingargentina.org	26	0	org	100.000000	1.000000	0.079963	...	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
235790	660997.txt	https://www.skincareliving.com	29	www.skincareliving.com	22	0	com	100.000000	1.000000	0.522907	...	1	0
235791	77185.txt	https://www.winchester.gov.uk	28	www.winchester.gov.uk	21	0	uk	100.000000	0.785714	0.028555	...	1	0
235792	622132.txt	https://www.nononsensedesign.be	30	www.nononsensedesign.be	23	0	be	100.000000	1.000000	0.003319	...	0	0
235793	7503962.txt	https://patient-cell-40f5.updatedlogmylogin.wo...	55	patient-cell-40f5.updatedlogmylogin.workers.dev	47	0	dev	28.157537	0.465116	0.000961	...	0	0
235794	384822.txt	https://www.alternativefinland.com	33	www.alternativefinland.com	26	0	com	100.000000	1.000000	0.522907	...	0	0

235795 rows × 56 columns

The first part of the code imports all the required modules for feature selection, model, and scoring. Next, I read the data from the 'PhiUSIIL_Phishing_URL_Dataset.csv' file using pandas as 'df' and output the dataframe 'df'.

```
numerical_features = df.select_dtypes(include=['int64', 'float64'])
numerical_features
```

Python

	URLLength	DomainLength	IsDomainIP	URLSimilarityIndex	CharContinuationRate	TLDLegitimateProb	URLCharProb	TLDLength	NoOfSubDomain	HasObfuscation	...	Pay	Crypto	HasCopyrightInfo	NoOf
0	31	24	0	100.000000	1.000000	0.522907	0.061933	3	1	0	...	0	0	1	
1	23	16	0	100.000000	0.666667	0.032650	0.050207	2	1	0	...	0	0	1	
2	29	22	0	100.000000	0.866667	0.028555	0.064129	2	2	0	...	0	0	1	
3	26	19	0	100.000000	1.000000	0.522907	0.057606	3	1	0	...	1	1	1	
4	33	26	0	100.000000	1.000000	0.079963	0.059441	3	1	0	...	1	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
235790	29	22	0	100.000000	1.000000	0.522907	0.058739	3	1	0	...	1	0	1	
235791	28	21	0	100.000000	0.785714	0.028555	0.053834	2	2	0	...	1	0	0	
235792	30	23	0	100.000000	1.000000	0.003319	0.063093	2	1	0	...	0	0	1	
235793	55	47	0	28.157537	0.465116	0.000961	0.050211	3	2	0	...	0	0	0	
235794	33	26	0	100.000000	1.000000	0.522907	0.060596	3	1	0	...	0	0	1	

235795 rows × 51 columns

```
input_features = numerical_features.drop(columns=['label'])
target = numerical_features['label']
input_features
```

Python

	URLLength	DomainLength	IsDomainIP	URLSimilarityIndex	CharContinuationRate	TLDLegitimateProb	URLCharProb	TLDLength	NoOfSubDomain	HasObfuscation	...	Bank	Pay	Crypto	HasCopyrightInfo
0	31	24	0	100.000000	1.000000	0.522907	0.061933	3	1	0	...	1	0	0	1
1	23	16	0	100.000000	0.666667	0.032650	0.050207	2	1	0	...	0	0	0	1
2	29	22	0	100.000000	0.866667	0.028555	0.064129	2	2	0	...	0	0	0	1
3	26	19	0	100.000000	1.000000	0.522907	0.057606	3	1	0	...	0	1	1	1
4	33	26	0	100.000000	1.000000	0.079963	0.059441	3	1	0	...	1	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
235790	29	22	0	100.000000	1.000000	0.522907	0.058739	3	1	0	...	0	1	0	1
235791	28	21	0	100.000000	0.785714	0.028555	0.053834	2	2	0	...	0	1	0	0

Above is where I drop all of the categorical data types (ones that are not numerical) from “df” and place them in a new dataframe “numerical_features”. Following this, I drop the target column “label” and place it into the variable “input_features”. To make sure the target column can be re-added later, I place that column and its data into the variable “target”. Finally, I output the “input_features” dataframe.

```
selector = VarianceThreshold(threshold=0.5)

print(selector.fit_transform(input_features))
```

Python

```
[[ 31.         24.        100.        ... 119.          0.
  124.       ]
 [ 23.         16.        100.        ...  39.          0.
  217.       ]
 [ 29.         22.        100.        ...  42.          2.
    5.       ]
 ...
 [ 30.         23.        100.        ...  58.          2.
   67.       ]
 [ 55.         47.         28.15753735 ...   0.          0.
    0.       ]
 [ 33.         26.        100.        ... 256.          0.
  261.       ]]
```

```
mask = selector.get_support()
mask
```

Python

```
array([ True,  True, False,  True, False, False, False, False, False,
       False,  True, False,  True, False,  True, False,  True, False,
        True,  True, False, False,  True,  True, False,  True,  True,
       False, False, False, False, False, False,  True,  True, False,
       False, False, False, False, False, False, False, False,  True,
        True,  True,  True,  True,  True])
```

```
selected_columns = input_features.columns[mask]
print(selected_columns)
```

Python

```
Index(['URLLength', 'DomainLength', 'URLSimilarityIndex', 'NoOfObfuscatedChar',
       'NoOfLettersInURL', 'NoOfDegitsInURL', 'NoOfEqualsInURL',
       'NoOfAmpersandInURL', 'NoOfOtherSpecialCharsInURL', 'LineOfCode',
       'LargestLineLength', 'DomainTitleMatchScore', 'URLTitleMatchScore',
       'NoOfPopup', 'NoOfiFrame', 'NoOfImage', 'NoOfCSS', 'NoOfJS',
       'NoOfSelfRef', 'NoOfEmptyRef', 'NoOfExternalRef'],
      dtype='object')
```

In the above photo, I began to use the VarianceThreshold at 0.5 as a “selector” to filter out the input_features dataframe, which was printed out to the console. Next, I used the .getsupport() function to display the feature columns that passed the threshold or not (denoted by true or false) and placed it into “mask”. Finally, I applied the true/values to the columns in input_features to output the column names that passed the variance threshold filter selection. The new dataframe was “selected_columns,” which was printed to the console.

```
updated_input_features = input_features[selected_columns]
updated_input_features
```

Python

	URLLength	DomainLength	URLSimilarityIndex	NoOfObfuscatedChar	NoOfLettersInURL	NoOfDegitsInURL	NoOfEqualsInURL	NoOfAmpersandInURL	NoOfOtherSpecialCharsInURL	LineOfCode	...	DomainTitl
0	31	24	100.000000	0	18	0	0	0	1	558	...	
1	23	16	100.000000	0	9	0	0	0	2	618	...	
2	29	22	100.000000	0	15	0	0	0	2	467	...	
3	26	19	100.000000	0	13	0	0	0	1	6356	...	
4	33	26	100.000000	0	20	0	0	0	1	6089	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
235790	29	22	100.000000	0	16	0	0	0	1	2007	...	
235791	28	21	100.000000	0	14	0	0	0	2	1081	...	
235792	30	23	100.000000	0	17	0	0	0	1	709	...	
235793	55	47	28.157537	0	39	3	0	0	5	125	...	
235794	33	26	100.000000	0	20	0	0	0	1	1038	...	

235795 rows × 21 columns

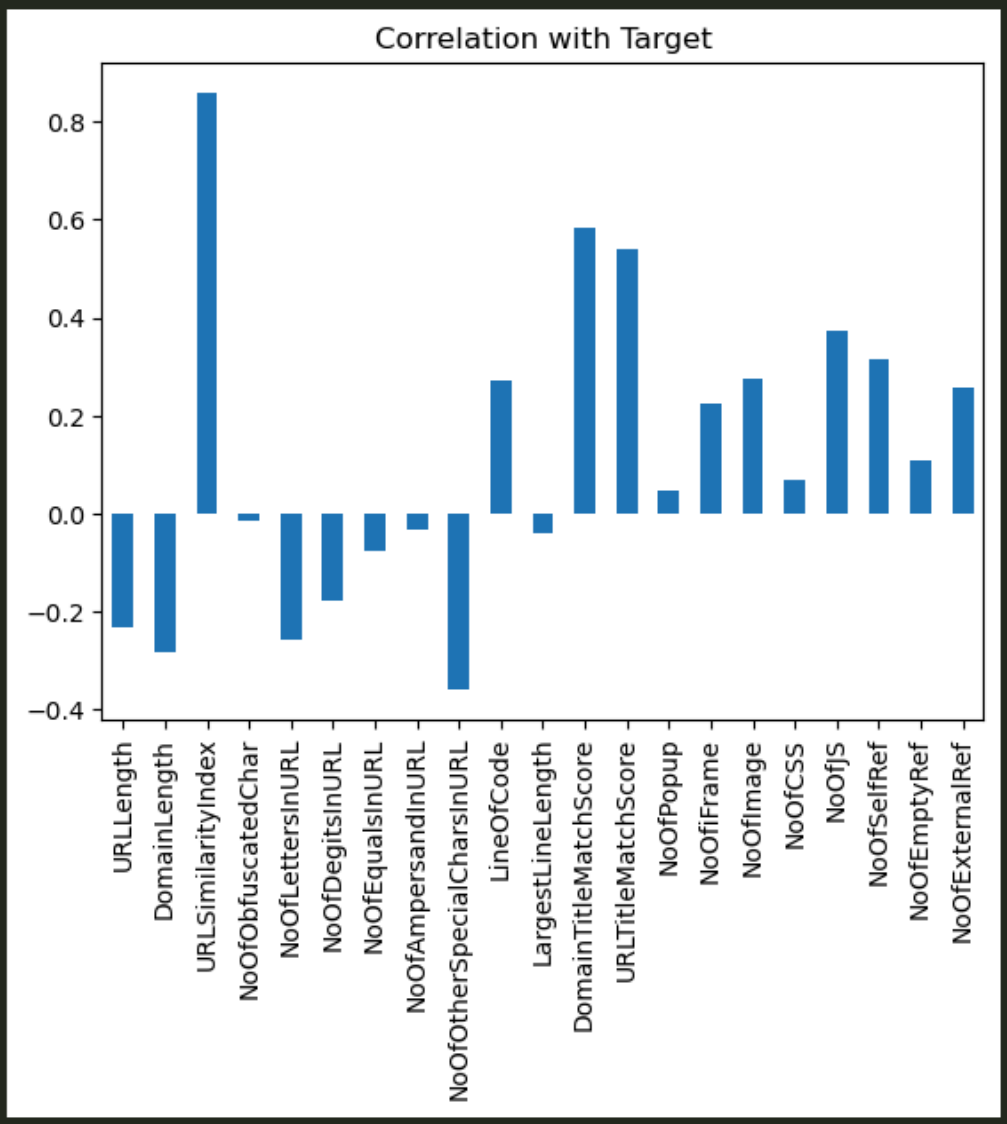
Here, I created a new dataframe “updated_input_features” using the selected columns from the input_features dataframe

```
correlations = updated_input_features.corrwith(target)
print(f'Correlations with target:\n{correlations}')
```

```
Correlations with target:
URLLength                    -0.233445
DomainLength                 -0.283152
URLSimilarityIndex            0.860358
NoOfObfuscatedChar           -0.015315
NoOfLettersInURL             -0.258090
NoOfDegitsInURL              -0.177980
NoOfEqualsInURL              -0.076963
NoOfAmpersandInURL           -0.034622
NoOfOtherSpecialCharsInURL   -0.358891
LineOfCode                    0.272257
LargestLineLength            -0.041111
DomainTitleMatchScore         0.584905
URLTitleMatchScore            0.539419
NoOfPopup                     0.047391
NoOfiFrame                    0.225822
NoOfImage                     0.274658
NoOfCSS                       0.068109
NoOfJS                        0.373500
NoOfSelfRef                   0.316211
NoOfEmptyRef                  0.109235
NoOfExternalRef               0.258627
dtype: float64
```

The above photo displays the ‘updated_input_features’ dataframe’s correlation with the target variable (which holds the label column as the target) that was placed into ‘correlations’ and printed to the console.

```
correlations.plot(kind="bar", title="Correlation with Target")
plt.show()
```



Here, I plotted the results of the updated/selected input feature's correlation with the target using plt.show().

```
model_features = pd.concat([updated_input_features, target], axis=1)
model_features
```

	URLLength	DomainLength	URLSimilarityIndex	NoOfObfuscatedChar	NoOfLettersInURL	NoOfDegitsInURL	NoOfEqualsInURL	NoOfA
0	31	24	100.000000	0	18	0	0	
1	23	16	100.000000	0	9	0	0	
2	29	22	100.000000	0	15	0	0	
3	26	19	100.000000	0	13	0	0	
4	33	26	100.000000	0	20	0	0	
...	...	...	...	...	...	...	...	
235790	29	22	100.000000	0	16	0	0	
235791	28	21	100.000000	0	14	0	0	
235792	30	23	100.000000	0	17	0	0	
235793	55	47	28.157537	0	39	3	0	
235794	33	26	100.000000	0	20	0	0	

235795 rows × 22 columns

```
X = updated_input_features
y = target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
```

```
dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

The first code block concatenates the target variable (label column) with the updated input features to which were placed into the new dataframe variable ‘model_features’. The model features were output to the console. The second code block sets up the train/test split by placing the updated input features into X and the target variable into y, then the parameters of shuffle and 80% train/20% test were specified. Finally, I placed the DecisionTreeClassifier into the variable ‘dtc’ and input the training test sets into the model with the default parameters (random_state = 42).

```
    y_pred = dtc.predict(X_test)
    print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
    print(f'Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}')
]
```

```
Accuracy: 0.9997879513984606
Confusion Matrix:
[[20264     5]
 [    5 26885]]
```

Finally, I got the model to predict the results of the X_test set and placed the results into y_pred. Following this, I printed the accuracy and confusion matrix of the model using the test answers and results (y_test and y_pred, respectively).