# Phishing Website Detection using Machine learning

CPS 844 Project

Nidhi Biswas (501118460)

Section 2

For this project I  have chosen the Website Phishing dataset from the UCI Machine Learning Repository [1].

Website Phishing attack  has become a very common way to obtain sensitive informations like bank details, passwords from users nowadays. The purpose of my project was to use different machine learning algorithms to classify which websites are legitimate or phishing in the dataset.

I have chosen this Website Phishing dataset which contains 1353 different websites from different sources [1] , thus my dataset contains 1353 instances and 9 attributes. This dataset contains 'Result' as the target attribute which indicates whether a website is phishing (1) or legitimate (0).

In this project I have applied and  implemented 5 different classification algorithms in this dataset which are :

1. Decision tree Classifier
2. Logistic Regression
3. K-Nearest Neighbors (KNN)
4. Support Vector Machine (SVM)
5. Gaussian Naive Bayes

Furthermore, i have added the performance report of each model which assesses each model based on performance metrics including accuracy, precision, recall and F1 score.

Later on, I used a feature selection algorithm called SelectKBest method  to determine the top 3 best features, then train the models again with the new set of selected features and reimplemented the same 5 classification algorithms with the same performance metrics  and compared  the metrics of all the data mining algorithms with and without the feature selection.

The following sections will demonstrate with code and explanation on all the steps I have taken in this project :

1. explorative analysis and data preprocessing
2. algorithm implementations and performance
3. Feature selection
4. Algorithm implementations with feature selection
5. Comparison of all the 5 models with feature selection

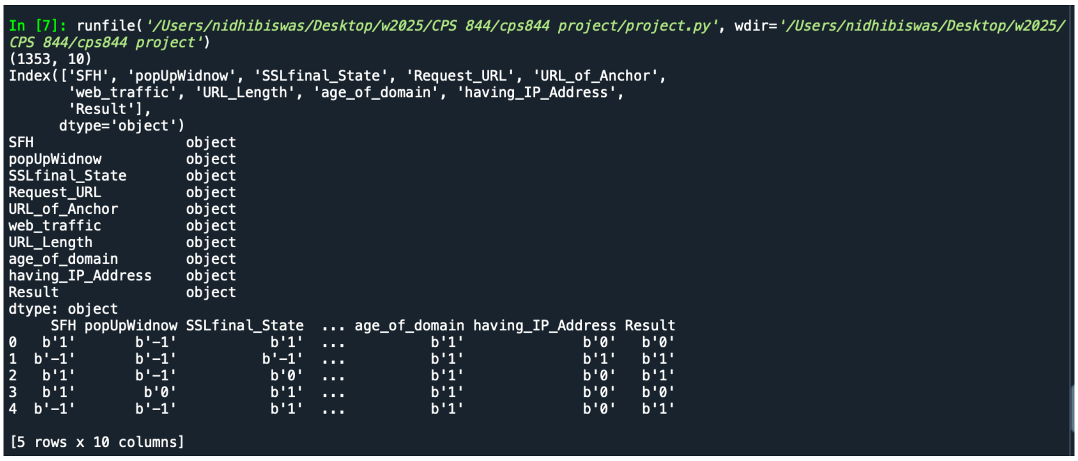
## 1. explorative analysis and data preprocessing

Data inspection:

After loading the dataset, I did some inspection of the data by printing the shape, columns, data types and head.

```
print(df.shape)
print(df.columns)
print(df.dtypes)
print(df.head())
```

I have observed that my dataset contains (1353,10) which indicates there are 1353 instances and 10 variables. I have also checked the name and data type of each variables and printed the first 5 rows

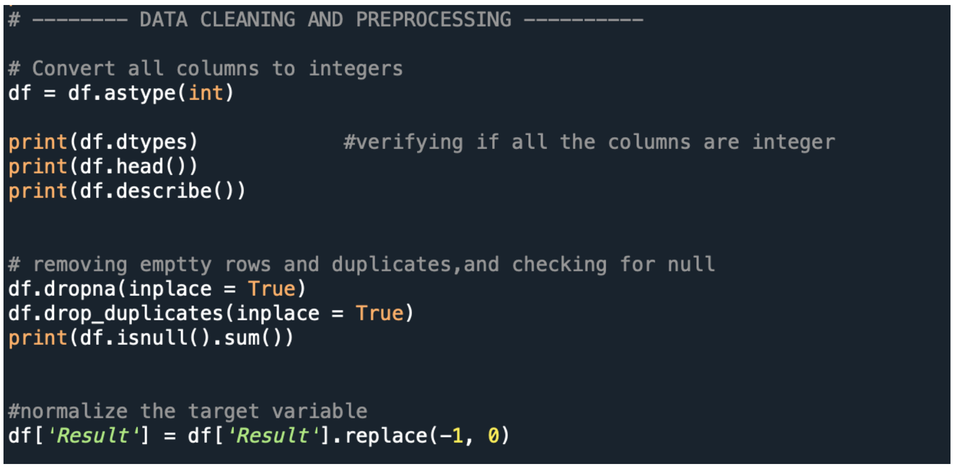


```
In [7]: runfile('/Users/nidhibiswas/Desktop/w2025/CPS 844/cps844 project/project.py', wdir='/Users/nidhibiswas/Desktop/w2025/
CPS 844/cps844 project')
(1353, 10)
Index(['SFH', 'popUpWidnow', 'SSLfinal_State', 'Request_URL', 'URL_of_Anchor',
       'web_traffic', 'URL_Length', 'age_of_domain', 'having_IP_Address',
       'Result'],
      dtype='object')
SFH                  object
popUpWidnow          object
SSLfinal_State       object
Request_URL          object
URL_of_Anchor        object
web_traffic          object
URL_Length           object
age_of_domain        object
having_IP_Address    object
Result               object
dtype: object
     SFH popUpWidnow SSLfinal_State  ... age_of_domain having_IP_Address Result
0   b'1'       b'-1'          b'1'  ...          b'1'              b'0'   b'0'
1  b'-1'       b'-1'         b'-1'  ...          b'1'              b'1'   b'1'
2   b'1'       b'-1'          b'0'  ...          b'1'              b'0'   b'1'
3   b'1'        b'0'          b'1'  ...          b'1'              b'0'   b'0'
4  b'-1'       b'-1'          b'1'  ...          b'1'              b'0'   b'1'

[5 rows x 10 columns]
```

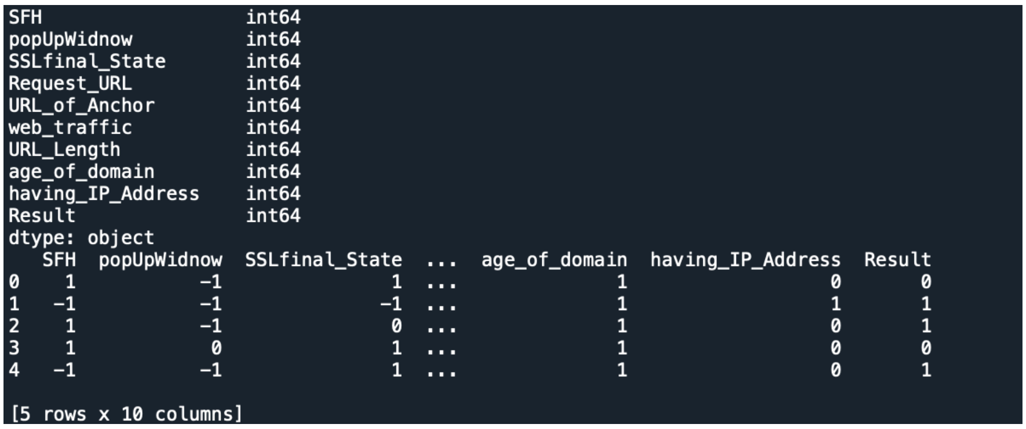
<u>Data cleaning :</u>

All the columns were originally stored as object types so I converted them to integer for future use in my ML models, and  then handled all the empty rows, duplicates and checked for null values.

Then i normalized the target variable ‘Result’ by replacing -1 with 0, so that it can be easily classified between phishing and legitimate websites using 0 and 1.



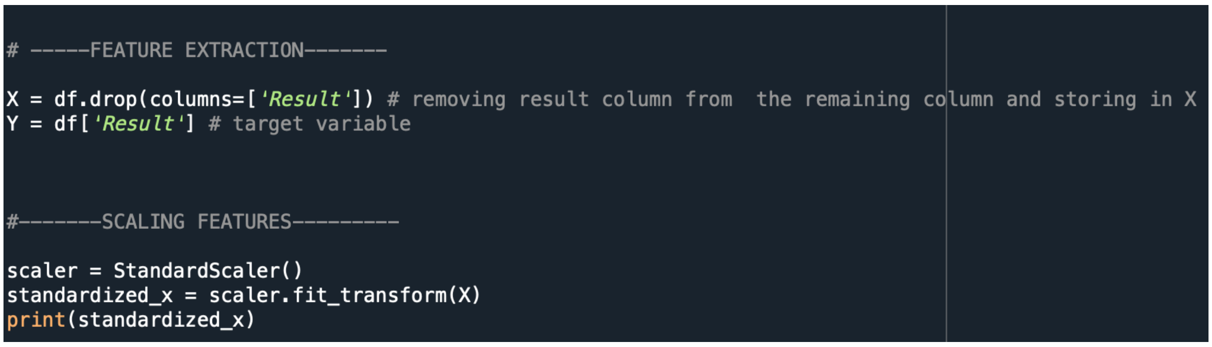
```python
# --------- DATA CLEANING AND PREPROCESSING -----------

# Convert all columns to integers
df = df.astype(int)

print(df.dtypes)            #verifying if all the columns are integer
print(df.head())
print(df.describe())


# removing emptty rows and duplicates,and checking for null
df.dropna(inplace = True)
df.drop_duplicates(inplace = True)
print(df.isnull().sum())


#normalize the target variable
df['Result'] = df['Result'].replace(-1, 0)
```



```
SFH                  int64
popUpWidnow          int64
SSLfinal_State       int64
Request_URL          int64
URL_of_Anchor        int64
web_traffic          int64
URL_Length           int64
age_of_domain        int64
having_IP_Address    int64
Result               int64
dtype: object
   SFH  popUpWidnow  SSLfinal_State  ...  age_of_domain  having_IP_Address  Result
0    1           -1               1  ...              1                  0       0
1   -1           -1              -1  ...              1                  1       1
2    1           -1               0  ...              1                  0       1
3    1            0               1  ...              1                  0       0
4   -1           -1               1  ...              1                  0       1

[5 rows x 10 columns]
```

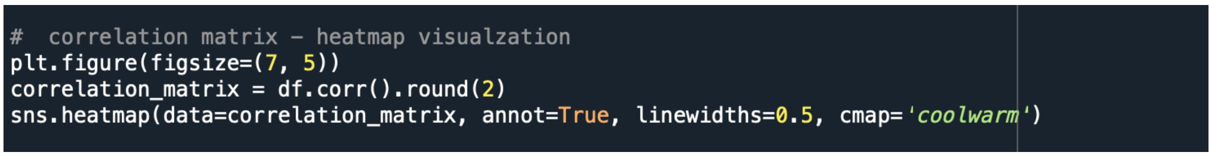
Feature extraction and scaling :

Then I separated the target variable "Result' from the dataset and stored in Y, with the remaining columns/attributes stored in X, which was standardized later.



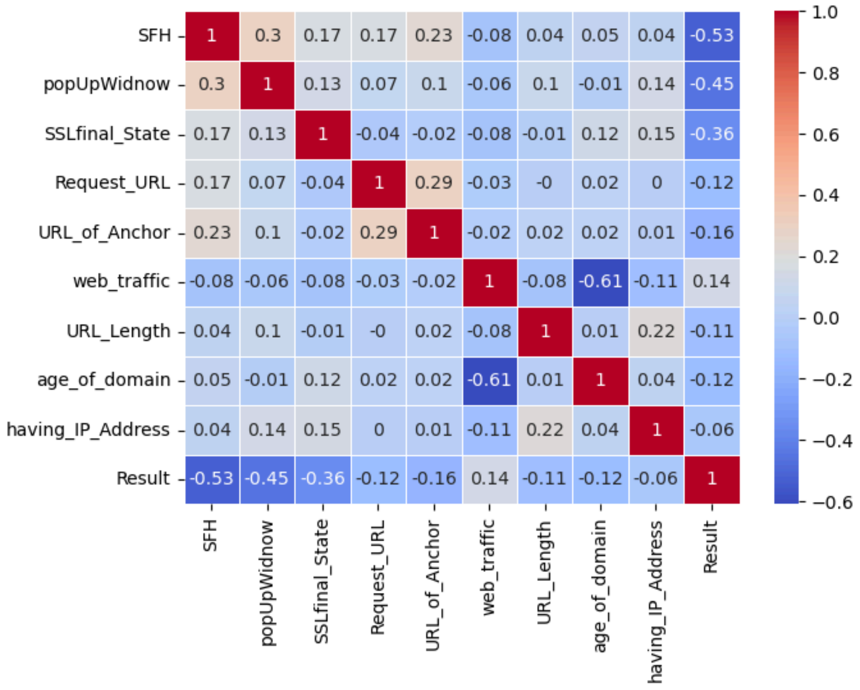
```
# -----FEATURE EXTRACTION-------

X = df.drop(columns=['Result']) # removing result column from  the remaining column and storing in X
Y = df['Result'] # target variable


#-------SCALING FEATURES---------

scaler = StandardScaler()
standardized_x = scaler.fit_transform(X)
print(standardized_x)
```

Correlation matrix - heatmap visualization :



```
#  correlation matrix - heatmap visualzation
plt.figure(figsize=(7, 5))
correlation_matrix = df.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True, linewidths=0.5, cmap='coolwarm')
```

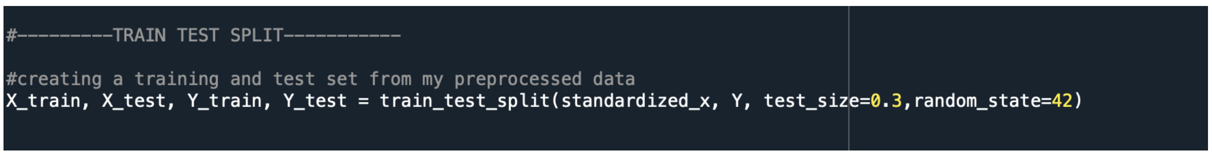
To further understand the relationship between the features and the target variable “Result”, I generated a correlation heatmap which helped identify which features are more strong (later to be proven with Feature Selection algorithm). In this heatmap it is seen SFH, popUpWidow and SSLfinal_State has strong correlation with “Result’ (in the last column)



| | SFH | popUpWidnow | SSLfinal_State | Request_URL | URL_of_Anchor | web_traffic | URL_Length | age_of_domain | having_IP_Address | Result |
|---|---|---|---|---|---|---|---|---|---|---|
| SFH | 1 | 0.3 | 0.17 | 0.17 | 0.23 | -0.08 | 0.04 | 0.05 | 0.04 | -0.53 |
| popUpWidnow | 0.3 | 1 | 0.13 | 0.07 | 0.1 | -0.06 | 0.1 | -0.01 | 0.14 | -0.45 |
| SSLfinal_State | 0.17 | 0.13 | 1 | -0.04 | -0.02 | -0.08 | -0.01 | 0.12 | 0.15 | -0.36 |
| Request_URL | 0.17 | 0.07 | -0.04 | 1 | 0.29 | -0.03 | -0 | 0.02 | 0 | -0.12 |
| URL_of_Anchor | 0.23 | 0.1 | -0.02 | 0.29 | 1 | -0.02 | 0.02 | 0.02 | 0.01 | -0.16 |
| web_traffic | -0.08 | -0.06 | -0.08 | -0.03 | -0.02 | 1 | -0.08 | -0.61 | -0.11 | 0.14 |
| URL_Length | 0.04 | 0.1 | -0.01 | -0 | 0.02 | -0.08 | 1 | 0.01 | 0.22 | -0.11 |
| age_of_domain | 0.05 | -0.01 | 0.12 | 0.02 | 0.02 | -0.61 | 0.01 | 1 | 0.04 | -0.12 |
| having_IP_Address | 0.04 | 0.14 | 0.15 | 0 | 0.01 | -0.11 | 0.22 | 0.04 | 1 | -0.06 |
| Result | -0.53 | -0.45 | -0.36 | -0.12 | -0.16 | 0.14 | -0.11 | -0.12 | -0.06 | 1 |

<u>Train-test split :</u>

Then I’ve split the dataset into training and test sets, with 70% of the data to be used for training and 30% for testing.



```
#---------TRAIN TEST SPLIT-----------

#creating a training and test set from my preprocessed data
X_train, X_test, Y_train, Y_test = train_test_split(standardized_x, Y, test_size=0.3,random_state=42)
```

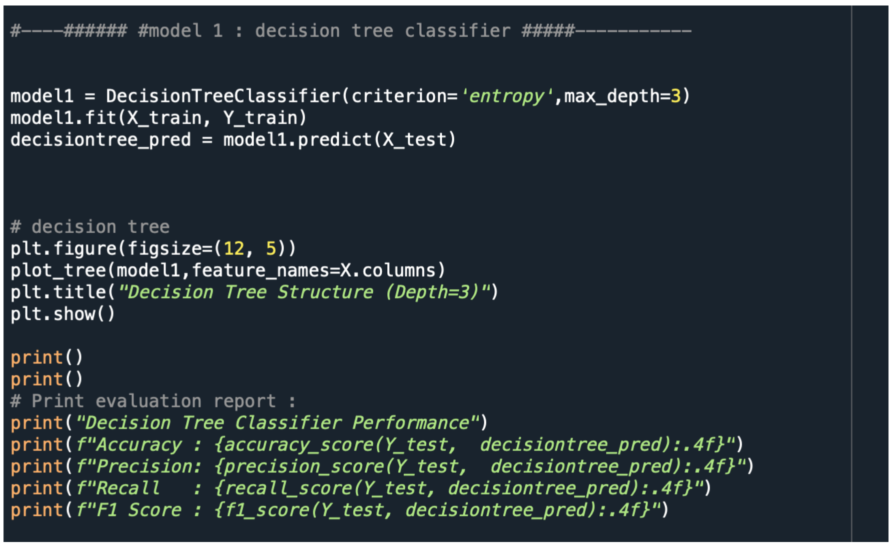
**2. algorithm implementations**

## Model 1: Decision Tree Classifier

My first model named model1 was a decision tree classifier using the criterion ‘entropy’ with a maximum depth of 3.

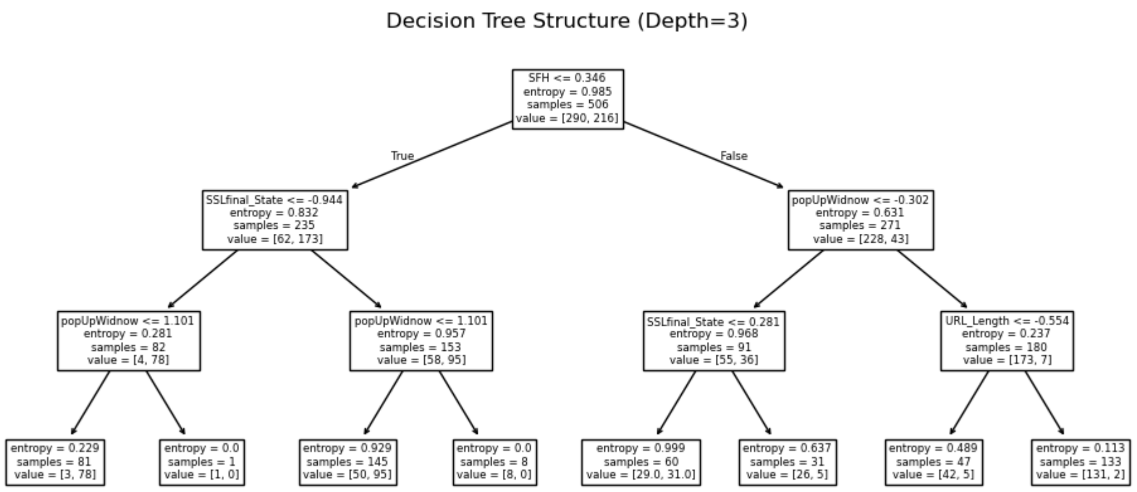
predictions were made on the test set using model1.predict(X_test), then these predictions were stored in decisiontree_pred. These predicted values were used to compare with Y_test to evaluate performance metrics.

Below is the code for my model:

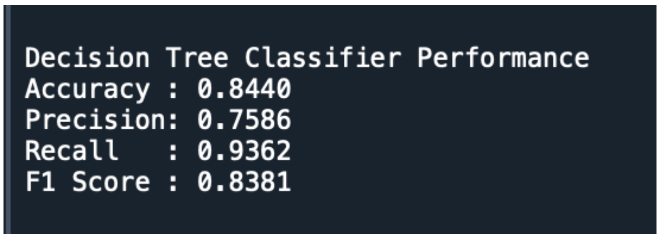


```
#----###### #model 1 : decision tree classifier #####----------


model1 = DecisionTreeClassifier(criterion='entropy',max_depth=3)
model1.fit(X_train, Y_train)
decisiontree_pred = model1.predict(X_test)



# decision tree
plt.figure(figsize=(12, 5))
plot_tree(model1,feature_names=X.columns)
plt.title("Decision Tree Structure (Depth=3)")
plt.show()

print()
print()
# Print evaluation report :
print("Decision Tree Classifier Performance")
print(f"Accuracy : {accuracy_score(Y_test,  decisiontree_pred):.4f}")
print(f"Precision: {precision_score(Y_test,  decisiontree_pred):.4f}")
print(f"Recall   : {recall_score(Y_test, decisiontree_pred):.4f}")
print(f"F1 Score : {f1_score(Y_test, decisiontree_pred):.4f}")
```

Furthermore, I have also plotted the decision tree as the image shown below:



The results of the performances were:

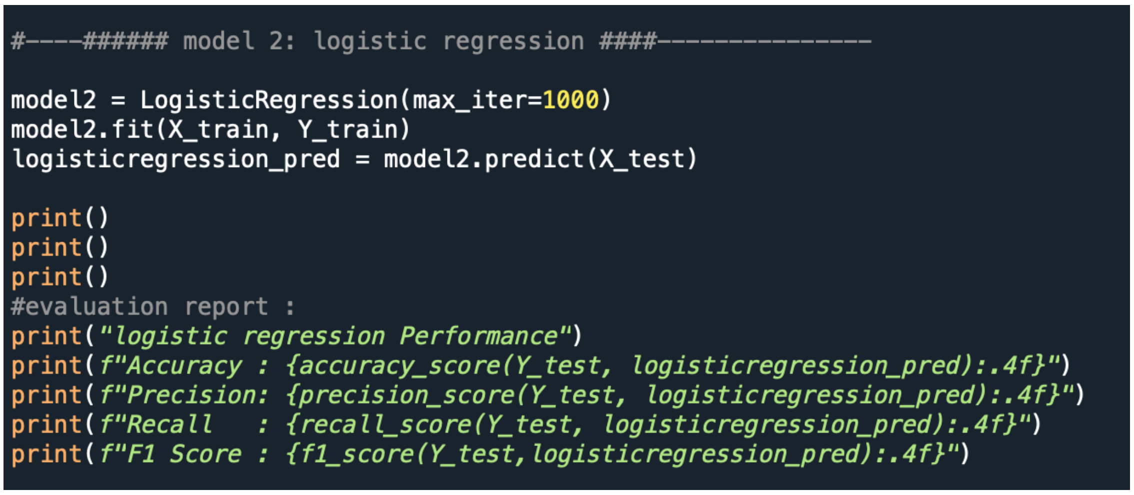


```
Decision Tree Classifier Performance
Accuracy : 0.8440
Precision: 0.7586
Recall   : 0.9362
F1 Score : 0.8381
```

## Model 2: logistic regression model

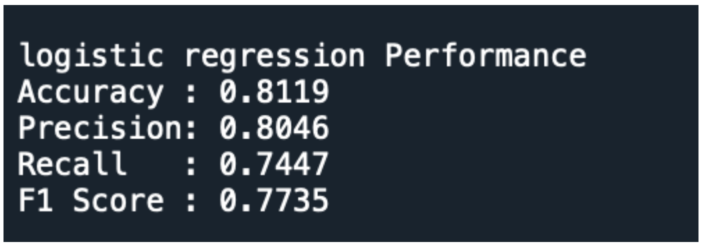
My second model named model2 was a logistic regression based model with a maximum of 1000 iterations.

predictions were made on the test set using model2.predict(X_test), then these predictions were stored in logisticregression_pred. These predicted values were used to compare with Y_test to evaluate performance metrics.



```python
#----###### model 2: logistic regression ####---------------

model2 = LogisticRegression(max_iter=1000)
model2.fit(X_train, Y_train)
logisticregression_pred = model2.predict(X_test)

print()
print()
print()
#evaluation report :
print("logistic regression Performance")
print(f"Accuracy : {accuracy_score(Y_test, logisticregression_pred):.4f}")
print(f"Precision: {precision_score(Y_test, logisticregression_pred):.4f}")
print(f"Recall    : {recall_score(Y_test, logisticregression_pred):.4f}")
print(f"F1 Score : {f1_score(Y_test,logisticregression_pred):.4f}")
```

The results of the performances were:

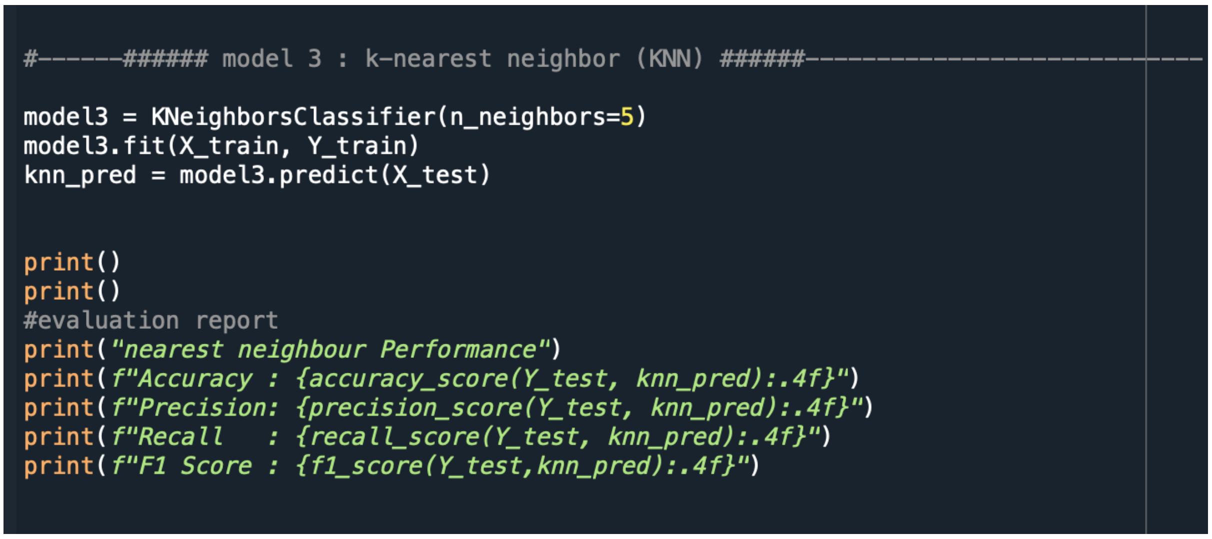


```
logistic regression Performance
Accuracy : 0.8119
Precision: 0.8046
Recall    : 0.7447
F1 Score : 0.7735
```

# Model 3 : K-Nearest Neighbors (KNN)

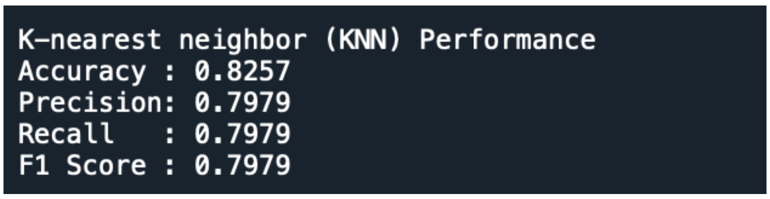
My third model named model3 was based on K-Nearest Neighbors (KNN) algorithm with k=5 neighbors.

predictions were made on the test set using model3.predict(X_test), then these predictions were stored in knn_pred. These predicted values were used to compare with Y_test to evaluate performance metrics.



```
#------###### model 3 : k-nearest neighbor (KNN) ######----------------------------

model3 = KNeighborsClassifier(n_neighbors=5)
model3.fit(X_train, Y_train)
knn_pred = model3.predict(X_test)


print()
print()
#evaluation report
print("nearest neighbour Performance")
print(f"Accuracy : {accuracy_score(Y_test, knn_pred):.4f}")
print(f"Precision: {precision_score(Y_test, knn_pred):.4f}")
print(f"Recall   : {recall_score(Y_test, knn_pred):.4f}")
print(f"F1 Score : {f1_score(Y_test,knn_pred):.4f}")
```

The results of the performances were:

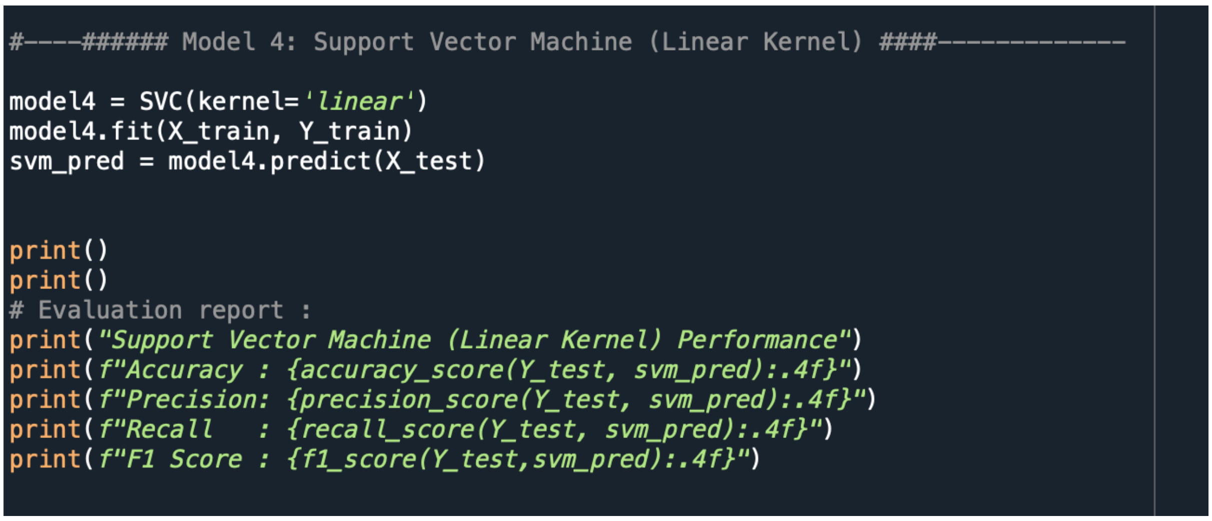


```
K-nearest neighbor (KNN) Performance
Accuracy : 0.8257
Precision: 0.7979
Recall   : 0.7979
F1 Score : 0.7979
```

## Model 4 : Support Vector Machine (SVM)

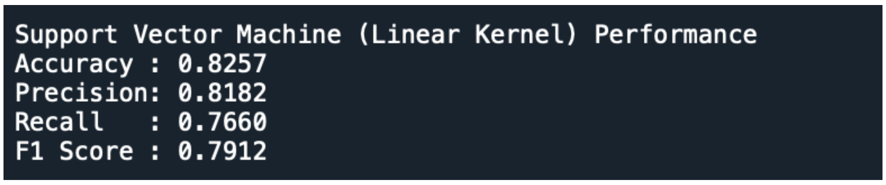
My fourth model named model4 used Support Vector Machine(SVM) algorithm with a linear kernel.

predictions were made on the test set using model4.predict(X_test), then these predictions were stored in svm_pred. These predicted values were used to compare with Y_test to evaluate performance metrics.



```python
#----###### Model 4: Support Vector Machine (Linear Kernel) ####-------------

model4 = SVC(kernel='linear')
model4.fit(X_train, Y_train)
svm_pred = model4.predict(X_test)


print()
print()
# Evaluation report :
print("Support Vector Machine (Linear Kernel) Performance")
print(f"Accuracy : {accuracy_score(Y_test, svm_pred):.4f}")
print(f"Precision: {precision_score(Y_test, svm_pred):.4f}")
print(f"Recall    : {recall_score(Y_test, svm_pred):.4f}")
print(f"F1 Score : {f1_score(Y_test,svm_pred):.4f}")
```

The results of the performances were:

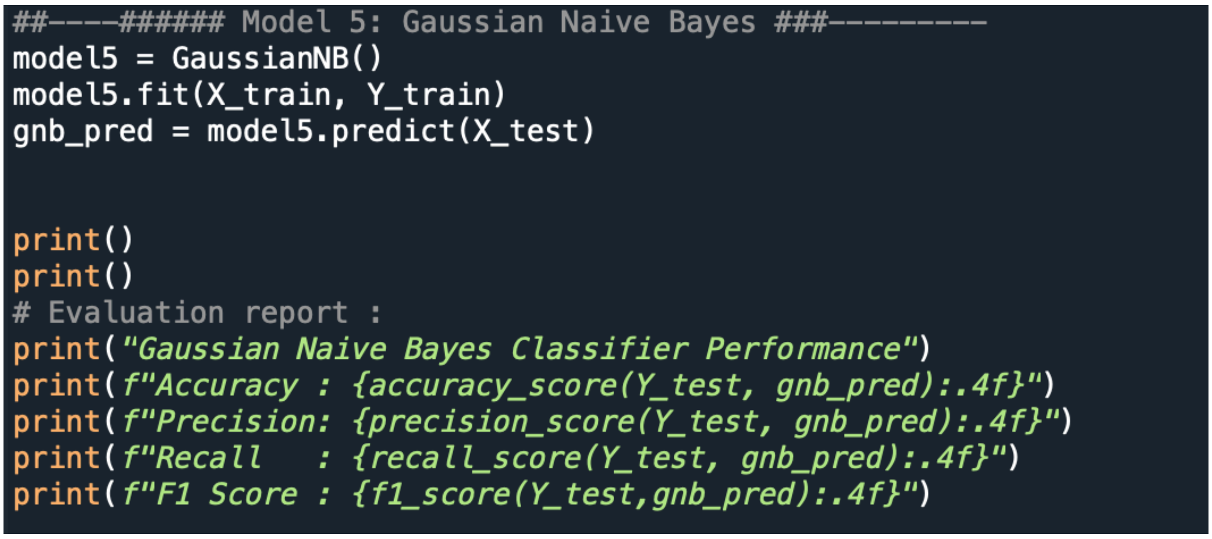


```
Support Vector Machine (Linear Kernel) Performance
Accuracy : 0.8257
Precision: 0.8182
Recall    : 0.7660
F1 Score : 0.7912
```

## Model 5 : Gaussian Naive Bayes

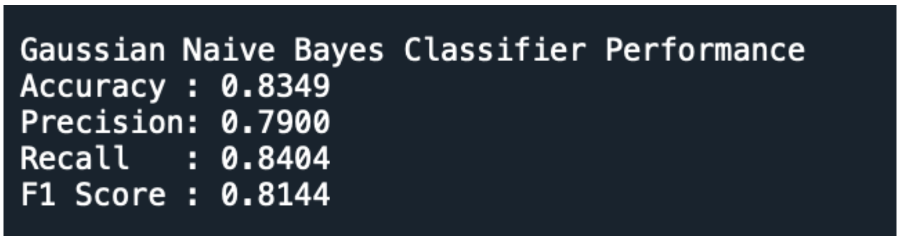
My fifth model named model5 was implemented using Gaussian Naive Bayes algorithm based on Bayes Theorem.

predictions were made on the test set using model5.predict(X_test), then these predictions were stored in gnb_pred. These predicted values were used to compare with Y_test to evaluate performance metrics.



```python
##----###### Model 5: Gaussian Naive Bayes ###---------
model5 = GaussianNB()
model5.fit(X_train, Y_train)
gnb_pred = model5.predict(X_test)


print()
print()
# Evaluation report :
print("Gaussian Naive Bayes Classifier Performance")
print(f"Accuracy : {accuracy_score(Y_test, gnb_pred):.4f}")
print(f"Precision: {precision_score(Y_test, gnb_pred):.4f}")
print(f"Recall    : {recall_score(Y_test, gnb_pred):.4f}")
print(f"F1 Score : {f1_score(Y_test,gnb_pred):.4f}")
```

The results of the performances were:

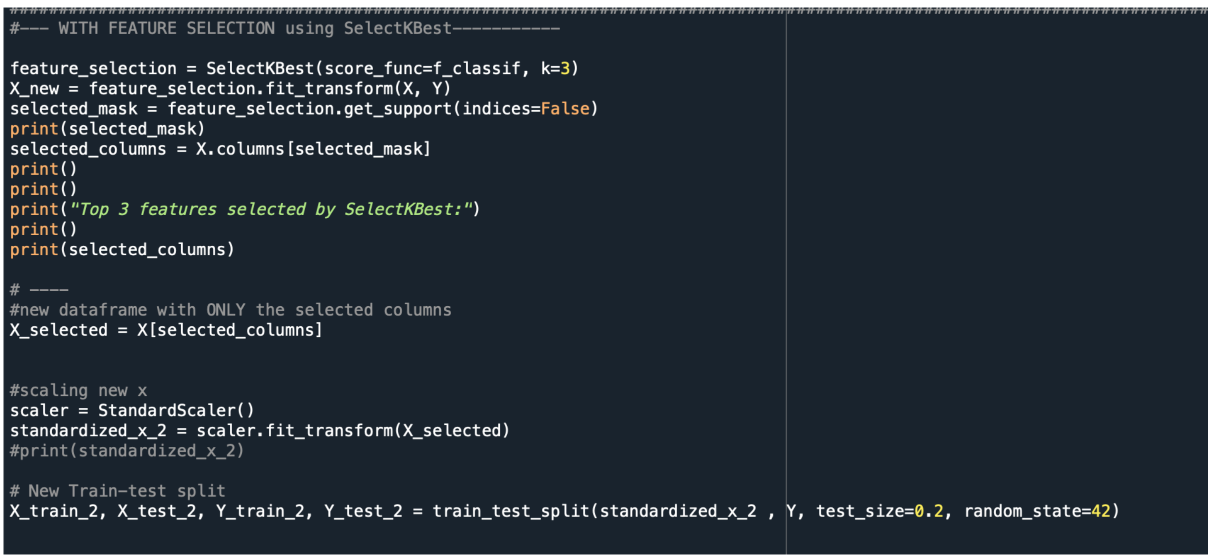


```
Gaussian Naive Bayes Classifier Performance
Accuracy : 0.8349
Precision: 0.7900
Recall    : 0.8404
F1 Score : 0.8144
```

## 3. Feature selection

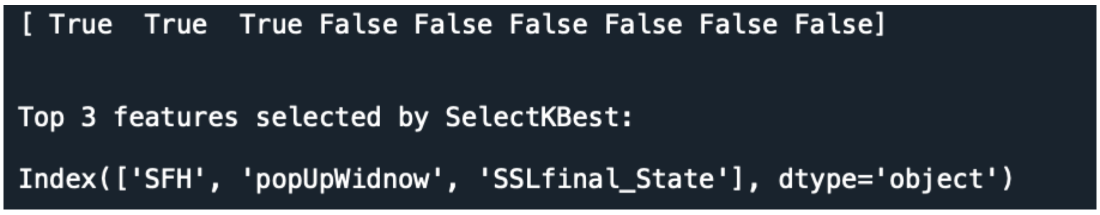
I imported SelectKBest from sklearn library and used score_func called f_classif which finds the ANOVA F-value between features. SelectKBest  gave the top 3 most important features from the dataset which was SFH, popUpWidow and SSLfinal_State.

Furthermore, I created a new dataframe containing new columns of X (with the 3 features) and performed a new train/test split with X_selected.



```
#--- WITH FEATURE SELECTION using SelectKBest-----------

feature_selection = SelectKBest(score_func=f_classif, k=3)
X_new = feature_selection.fit_transform(X, Y)
selected_mask = feature_selection.get_support(indices=False)
print(selected_mask)
selected_columns = X.columns[selected_mask]
print()
print()
print("Top 3 features selected by SelectKBest:")
print()
print(selected_columns)

# ----
#new dataframe with ONLY the selected columns
X_selected = X[selected_columns]


#scaling new x
scaler = StandardScaler()
standardized_x_2 = scaler.fit_transform(X_selected)
#print(standardized_x_2)

# New Train-test split
X_train_2, X_test_2, Y_train_2, Y_test_2 = train_test_split(standardized_x_2 , Y, test_size=0.2, random_state=42)
```

I have used get_support() which returns an array of boolean values containing true if features selected and false if not. This is then applied to x.columns to collect the top features selected by the feature selection algorithm SelectKBest. Below is the screenshot of the result:



```
[ True  True  True False False False False False False]

Top 3 features selected by SelectKBest:

Index(['SFH', 'popUpWidnow', 'SSLfinal_State'], dtype='object')
```

Above result shows True for the first 3 features which indicates the 3 selected features,

And the final print statement of the top 3 selected features by SelectKBest :

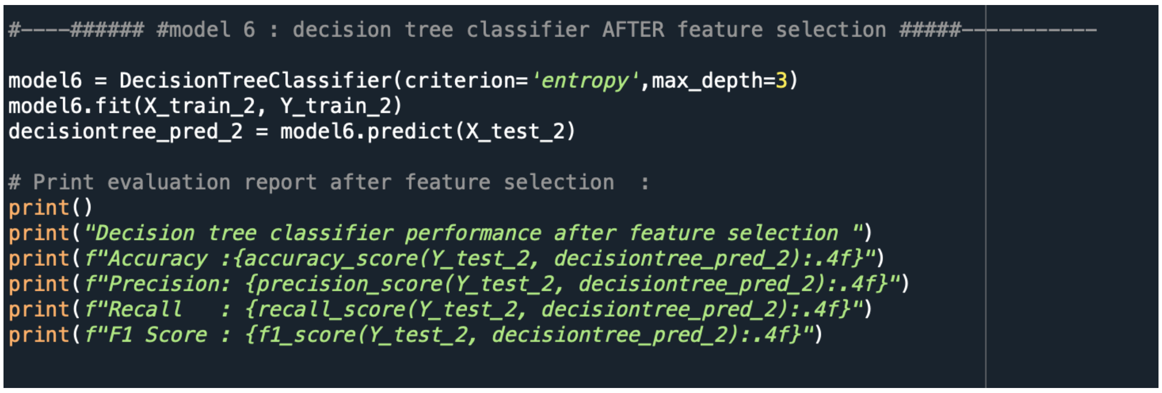
1. SFH
2. popUpWidow
3. SSLfinal_State

I have previously used a correlation matrix at the beginning of my code (after preprocessing the dataset) to check correlations between features with target variable “Result” which showed the same 3 features as above  having strong correlation with “Result’.

## 4. Algorithm implementations with feature selection

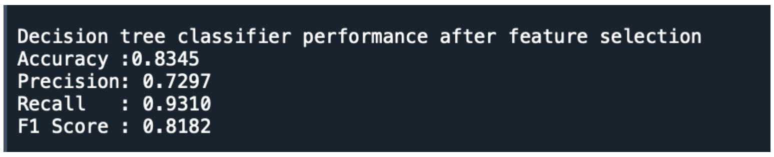
I repeated and trained the following models on the newly trained features with the top 3 selected features, made predictions on X_test_2 and compared the predicted values with Y_test_2 to evaluate performance metrics for all the following models below:

### Model 6 : Decision Tree Classifier - with feature selection



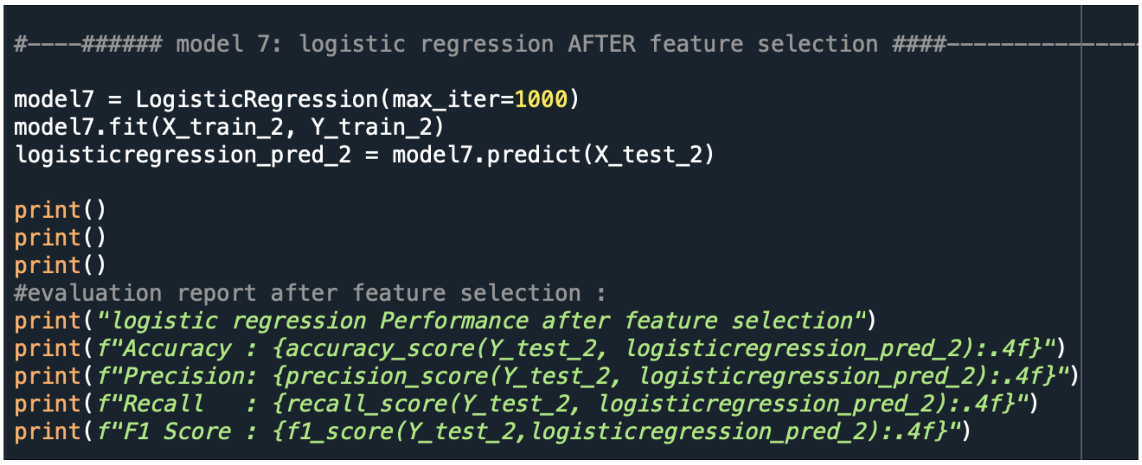
```
#----###### #model 6 : decision tree classifier AFTER feature selection #####-----------

model6 = DecisionTreeClassifier(criterion='entropy',max_depth=3)
model6.fit(X_train_2, Y_train_2)
decisiontree_pred_2 = model6.predict(X_test_2)

# Print evaluation report after feature selection  :
print()
print("Decision tree classifier performance after feature selection ")
print(f"Accuracy :{accuracy_score(Y_test_2, decisiontree_pred_2):.4f}")
print(f"Precision: {precision_score(Y_test_2, decisiontree_pred_2):.4f}")
print(f"Recall    : {recall_score(Y_test_2, decisiontree_pred_2):.4f}")
print(f"F1 Score : {f1_score(Y_test_2, decisiontree_pred_2):.4f}")
```

The results of the performances were:



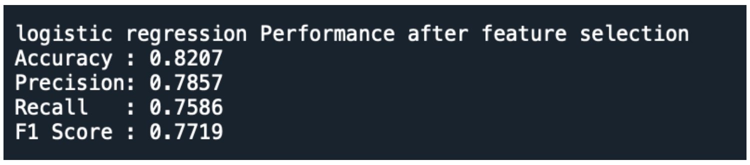
```
Decision tree classifier performance after feature selection
Accuracy :0.8345
Precision: 0.7297
Recall   : 0.9310
F1 Score : 0.8182
```

## Model 7 : Logistic Regression Model with feature selection



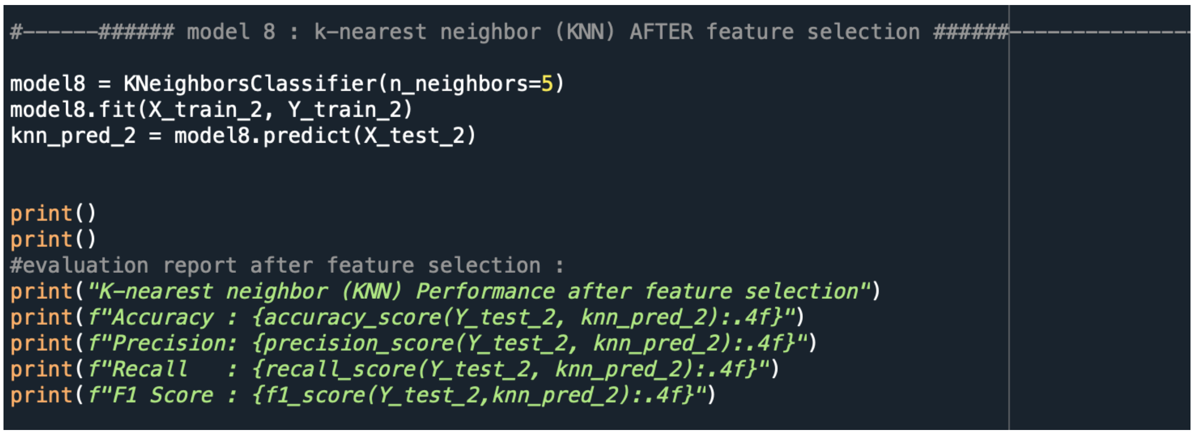
```python
#----###### model 7: logistic regression AFTER feature selection ####--------------

model7 = LogisticRegression(max_iter=1000)
model7.fit(X_train_2, Y_train_2)
logisticregression_pred_2 = model7.predict(X_test_2)

print()
print()
print()
#evaluation report after feature selection :
print("logistic regression Performance after feature selection")
print(f"Accuracy : {accuracy_score(Y_test_2, logisticregression_pred_2):.4f}")
print(f"Precision: {precision_score(Y_test_2, logisticregression_pred_2):.4f}")
print(f"Recall   : {recall_score(Y_test_2, logisticregression_pred_2):.4f}")
print(f"F1 Score : {f1_score(Y_test_2,logisticregression_pred_2):.4f}")
```

The results of the performances were:



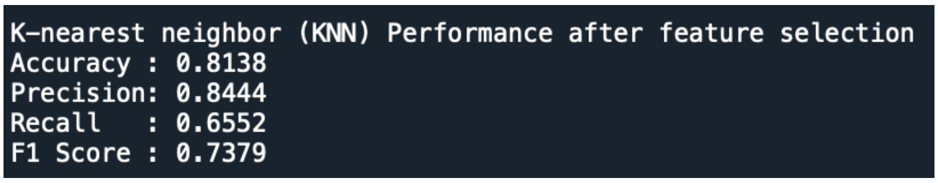
```
logistic regression Performance after feature selection
Accuracy : 0.8207
Precision: 0.7857
Recall   : 0.7586
F1 Score : 0.7719
```

## Model 8 : K-Nearest Neighbors (KNN) with feature selection



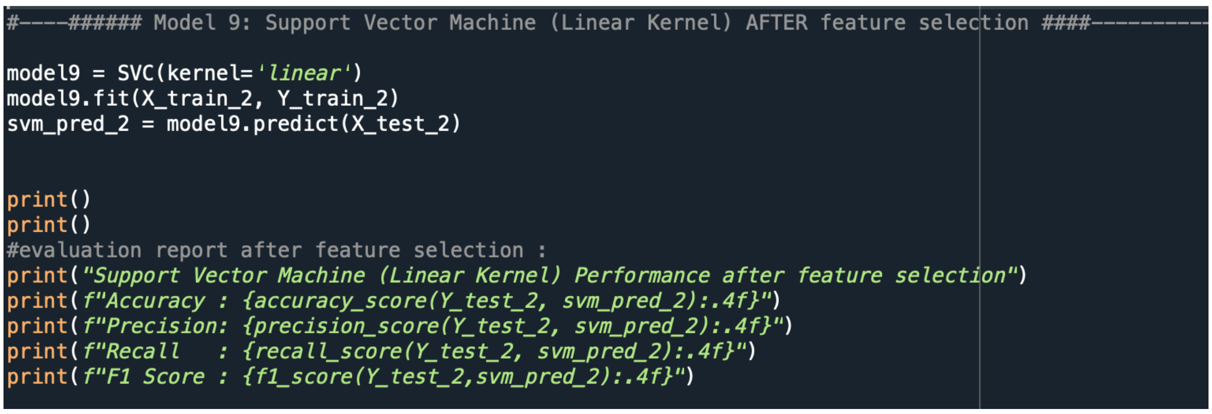
```python
#------###### model 8 : k-nearest neighbor (KNN) AFTER feature selection ######--------------

model8 = KNeighborsClassifier(n_neighbors=5)
model8.fit(X_train_2, Y_train_2)
knn_pred_2 = model8.predict(X_test_2)


print()
print()
#evaluation report after feature selection :
print("K-nearest neighbor (KNN) Performance after feature selection")
print(f"Accuracy : {accuracy_score(Y_test_2, knn_pred_2):.4f}")
print(f"Precision: {precision_score(Y_test_2, knn_pred_2):.4f}")
print(f"Recall   : {recall_score(Y_test_2, knn_pred_2):.4f}")
print(f"F1 Score : {f1_score(Y_test_2,knn_pred_2):.4f}")
```

The results of the performances were:



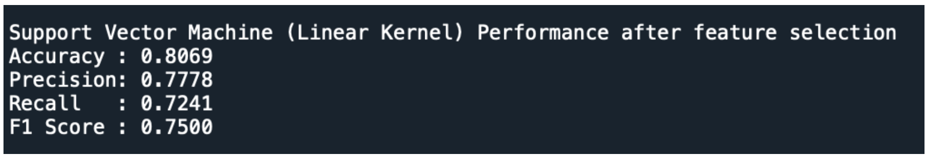
```
K-nearest neighbor (KNN) Performance after feature selection
Accuracy : 0.8138
Precision: 0.8444
Recall   : 0.6552
F1 Score : 0.7379
```

## Model 9 : Support Vector Machine (Linear Kernel) with feature selection



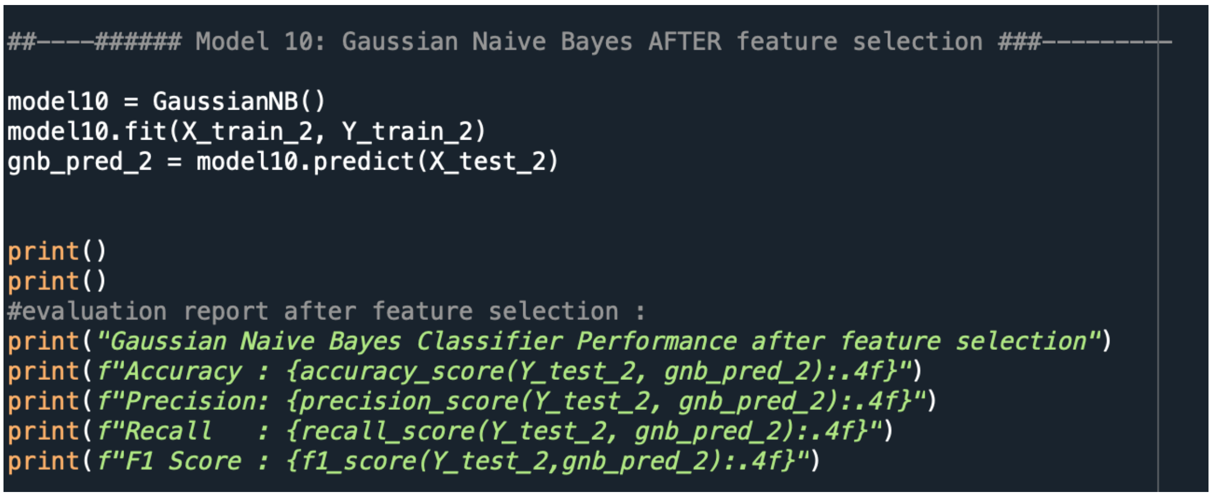
```
#----###### Model 9: Support Vector Machine (Linear Kernel) AFTER feature selection ####----------

model9 = SVC(kernel='linear')
model9.fit(X_train_2, Y_train_2)
svm_pred_2 = model9.predict(X_test_2)


print()
print()
#evaluation report after feature selection :
print("Support Vector Machine (Linear Kernel) Performance after feature selection")
print(f"Accuracy : {accuracy_score(Y_test_2, svm_pred_2):.4f}")
print(f"Precision: {precision_score(Y_test_2, svm_pred_2):.4f}")
print(f"Recall   : {recall_score(Y_test_2, svm_pred_2):.4f}")
print(f"F1 Score : {f1_score(Y_test_2,svm_pred_2):.4f}")
```

The results of the performances were:



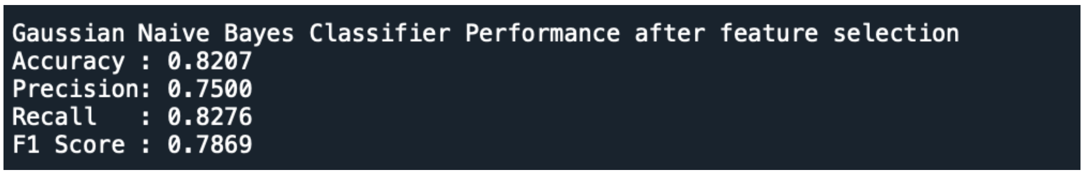
```
Support Vector Machine (Linear Kernel) Performance after feature selection
Accuracy : 0.8069
Precision: 0.7778
Recall   : 0.7241
F1 Score : 0.7500
```

## Model 10: Gaussian Naive Bayes with feature selection



```
##----###### Model 10: Gaussian Naive Bayes AFTER feature selection ###---------

model10 = GaussianNB()
model10.fit(X_train_2, Y_train_2)
gnb_pred_2 = model10.predict(X_test_2)


print()
print()
#evaluation report after feature selection :
print("Gaussian Naive Bayes Classifier Performance after feature selection")
print(f"Accuracy : {accuracy_score(Y_test_2, gnb_pred_2):.4f}")
print(f"Precision: {precision_score(Y_test_2, gnb_pred_2):.4f}")
print(f"Recall    : {recall_score(Y_test_2, gnb_pred_2):.4f}")
print(f"F1 Score : {f1_score(Y_test_2,gnb_pred_2):.4f}")
```

The results of the performances were:



```
Gaussian Naive Bayes Classifier Performance after feature selection
Accuracy : 0.8207
Precision: 0.7500
Recall    : 0.8276
F1 Score : 0.7869
```

## 5. Algorithm comparisons with/without feature selection

**Before feature selection:**

| Performance metrics: | Decision Tree Classifier | Logistic Regression | K-nearest Neighbor(KNN) | Support Vector machine(SVM) | Gaussian Naive Bayes |
|---|---|---|---|---|---|
| Accuracy | 0.8440 | 0.8119 | 0.8257 | 0.8257 | 0.8349 |
| Precision | 0.7586 | 0.8046 | 0.7979 | 0.8182 | 0.7900 |
| Recall | 0.9362 | 0.7447 | 0.7979 | 0.7660 | 0.8404 |
| F1 score | 0.8381 | 0.7735 | 0.7979 | 0.7912 | 0.8144 |

**After feature selection:**

| Performance metrics: | Decision Tree Classifier | Logistic Regression | K-nearest Neighbor(KNN) | Support Vector machine(SVM) | Gaussian Naive Bayes |
|---|---|---|---|---|---|
| Accuracy | 0.8345 | 0.8207 | 0.8138 | 0.8069 | 0.8207 |
| Precision | 0.7297 | 0.7857 | 0.8444 | 0.7778 | 0.7500 |
| Recall | 0.9310 | 0.7586 | 0.6552 | 0.7241 | 0.8276 |
| F1 score | 0.8182 | 0.7719 | 0.7379 | 0.7500 | 0.7869 |

The above table shows the summary of the performance evaluation metrics - Accuracy, Precision,Recall and F1 score of the 5 classification algorithms I have chosen to perform on my dataset. The first table contain the first 5 models without feature selection and the second table contain the next 5 models with feature selection.

Observations :

1. Decision Tree Classifier - this algorithm showed decrease in accuracy,precision ,recall,f1 score after feature selection. There was no sign of improvement in these metrics after feature selection. Thus,feature selection wasn't beneficial for this model.

2. Logistic Regression - this algorithm showed increase in Accuracy and recall after feature selection but precision was higher before feature selection so was F1 score, but slightly.

3. K-nearest neighbor(KNN) - this algorithm showed precision improved after feature selection, however it was observed that the accuracy, recall and F1 score decreased after feature selection. It was also observed that before feature selection, precision, recall and f1 score has the same values.

4. Support vector Machine(SVM) - this algorithm showed decrease in accuracy,precision ,recall,f1 score after feature selection.There was no sign of improvement in these metrics after feature selection. Thus,feature selection wasn't beneficial for this model.

5. Gaussian Naive Bayes - this algorithm showed decrease in accuracy,precision ,recall,f1 score after feature selection.There was no sign of improvement in these metrics after feature selection.   Thus,feature selection wasn't beneficial for this model.

Thus it can be observed that with feature selection it wasn’t much helpful in performance improvement of the 5 algorithms. From above comparisonm tt is observed specially in Decision tree, Support Vector Machine and Gaussiian Naive Bayes model that there was no improvement in any of the four metrics after feature selection. In fact, the values were better before feature selection was done with the full set of features.

On the other hand, Logistic regression and K-nearest neighbour had some mixed results but in majority model feature selection wasn’t beneficial.

Although feature selection using SelectKBest helped simplify the feature set by choosing top features, it did not contribute much to the performance metrics of this website phishing dataset that I have chosen for this project.

Citations :

[1] Abdelhamid, Neda. "Website Phishing." UCI Machine Learning Repository, 2014, https://doi.org/10.24432/C5B301.