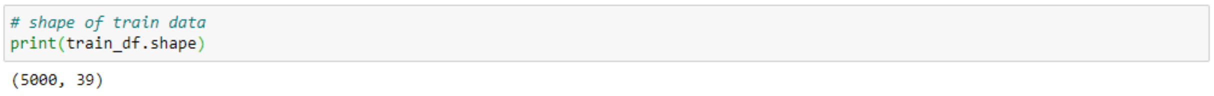
Model Building and Prediction

The objective of this work is to build the best machine learning models based on the training set in order to predict the target values using only 7 features.

Data Exploration/Analysis

This step is crucial. The goal is to get a better understanding of the data. That intuition will be helpful when modeling, imputing missing data.

One easy way to start is to count the number of rows and columns in our dataset:



```
# shape of train data
print(train_df.shape)
```

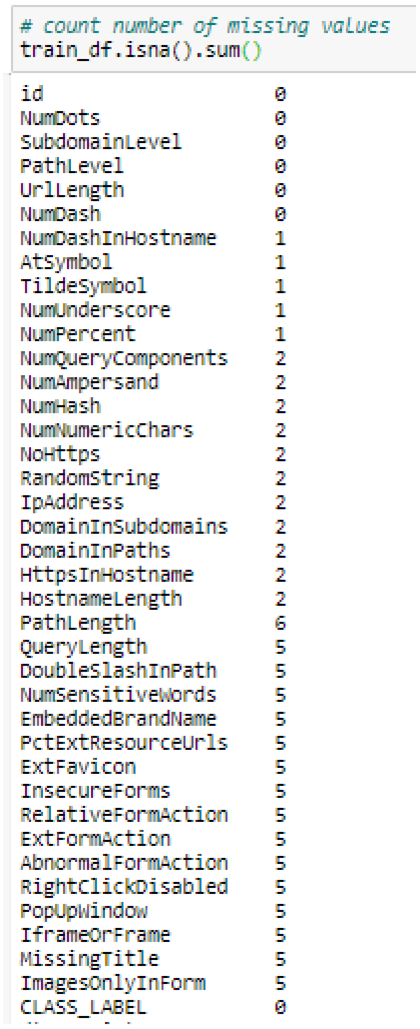
```
(5000, 39)
```

(5000, 39) as output which means our train dataset has 5000 rows and 39 columns.

- ➢ Our target variable is CLASS_LABEL: a binary variable which takes two values 0 or 1
- ➢ 38 features: some are floats, and others are objects.

- **<u>Missing value:</u>**

We count the number of missing values in each column.



```
# count number of missing values
train_df.isna().sum()
```

```
id                      0
NumDots                 0
SubdomainLevel         0
PathLevel               0
UrlLength               0
NumDash                 0
NumDashInHostname       1
AtSymbol                1
TildeSymbol             1
NumUnderscore           1
NumPercent              1
NumQueryComponents      2
NumAmpersand            2
NumHash                 2
NumNumericChars         2
NoHttps                 2
RandomString            2
IpAddress               2
DomainInSubdomains      2
DomainInPaths           2
HttpsInHostname         2
HostnameLength          2
PathLength              6
QueryLength             5
DoubleSlashInPath       5
NumSensitiveWords       5
EmbeddedBrandName       5
PctExtResourceUrls      5
ExtFavicon              5
InsecureForms           5
RelativeFormAction      5
ExtFormAction           5
AbnormalFormAction      5
RightClickDisabled      5
PopUpWindow             5
IframeOrFrame           5
MissingTitle            5
ImagesOnlyInForm        5
CLASS_LABEL             0
```

several features have missing values, but it's not much, so we can delete them.

- **<u>Converting Features:</u>** We have converted all string features to numerical.
- **<u>Standardization:</u>**

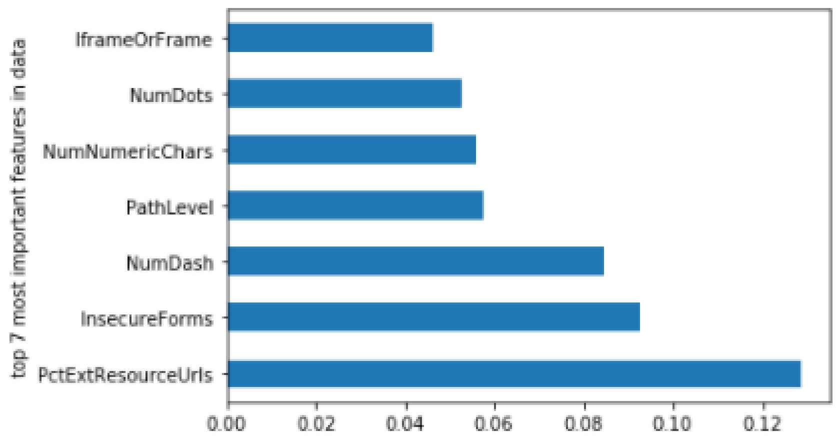
One of the most important transformations you need to apply to your data is feature scaling. With few exceptions, Machine Learning algorithms don't perform well when the input numerical attributes have very different scales.
We are going to use Standardization where the values are centered around the mean with a unit standard deviation.

Building Machine Learning Models

it is clearly a typical supervised learning task since you are given labelled training set examples.
Moreover, it is also a typical classification task, since we are asked to predict a class of our target variable.

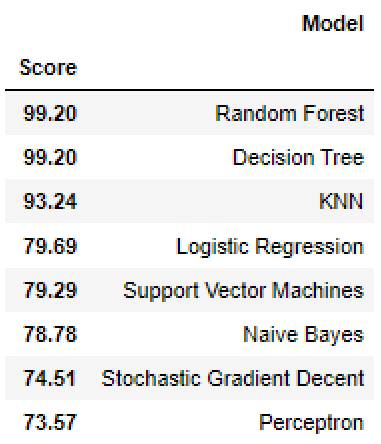
- **Feature Selection:**

since we allowed to use 7 features only, we are going to use the feature importance technique. Feature importance gives us a score for each feature of our data, the higher the score more important or relevant is the feature towards our output variable.
Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 7 features for the dataset.



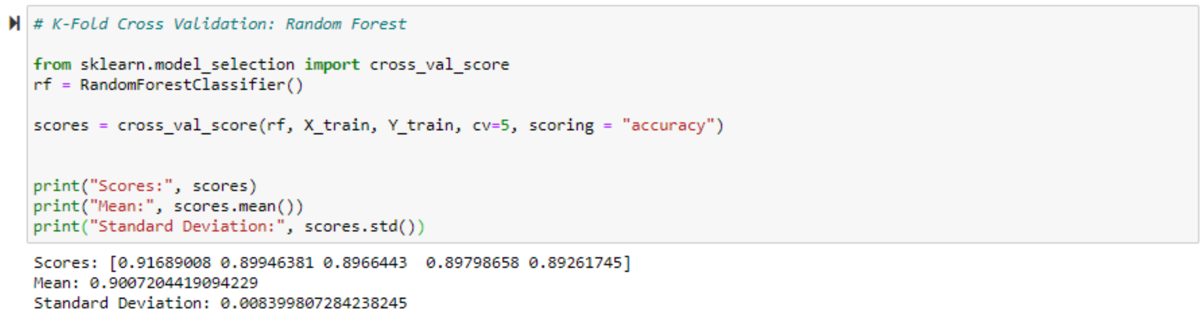
the graph below shows the 7 most features to include in our building models.

Afterwards we started training 8 different machine learning models:



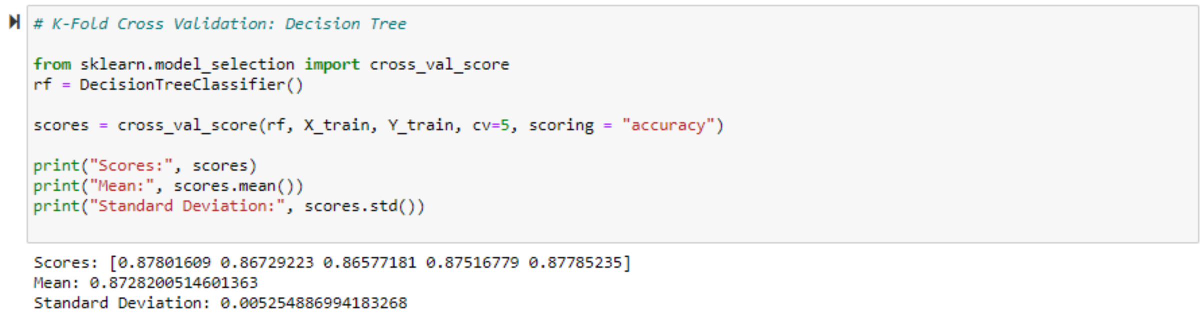
	Model
Score	
99.20	Random Forest
99.20	Decision Tree
93.24	KNN
79.69	Logistic Regression
79.29	Support Vector Machines
78.78	Naive Bayes
74.51	Stochastic Gradient Decent
73.57	Perceptron

We picked two of them (random forest and Decision Tree) and applied cross validation on it. K-Fold Cross Validation randomly splits the training data into **K subsets called folds**.



```
# K-Fold Cross Validation: Random Forest

from sklearn.model_selection import cross_val_score
rf = RandomForestClassifier()

scores = cross_val_score(rf, X_train, Y_train, cv=5, scoring = "accuracy")


print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())
```

```
Scores: [0.91689008 0.89946381 0.8966443  0.89798658 0.89261745]
Mean: 0.9007204419094229
Standard Deviation: 0.008399807284238245
```



```
# K-Fold Cross Validation: Decision Tree

from sklearn.model_selection import cross_val_score
rf = DecisionTreeClassifier()

scores = cross_val_score(rf, X_train, Y_train, cv=5, scoring = "accuracy")

print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())
```

```
Scores: [0.87801609 0.86729223 0.86577181 0.87516779 0.87785235]
Mean: 0.8728200514601363
Standard Deviation: 0.005254886994183268
```

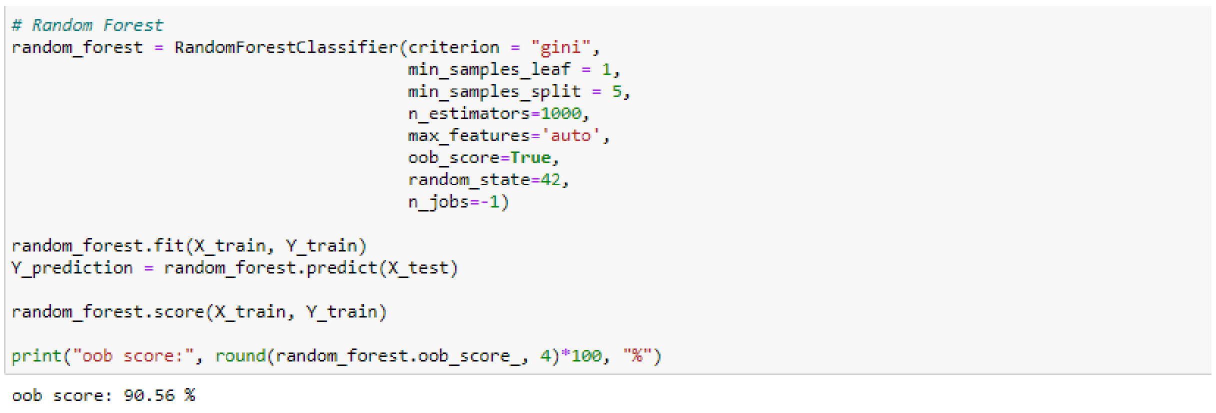
The Random Forest model has an average accuracy of 99% with a standard deviation of 0.8%. The standard deviation shows us; how precise the estimates are.

This means in our case that the accuracy of our Random Forest can differ + -0.8%.

The accuracy is still really good and since random forest is an easy to use model, we will try to increase its performance.

- **Hyperparameter Tuning**

We have tried different values of the parameters, and we have chosen those which maximize the precision.

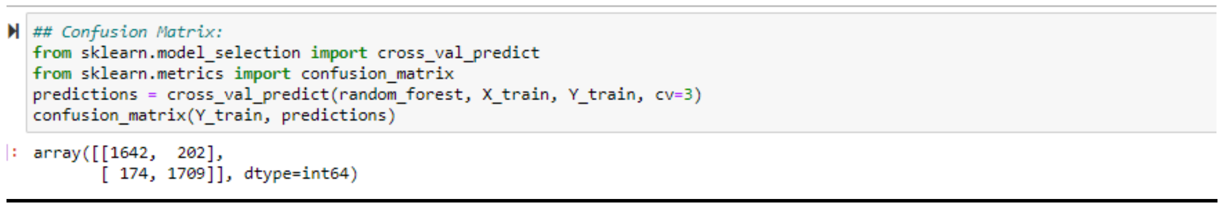


```
# Random Forest
random_forest = RandomForestClassifier(criterion = "gini",
                                       min_samples_leaf = 1,
                                       min_samples_split = 5,
                                       n_estimators=1000,
                                       max_features='auto',
                                       oob_score=True,
                                       random_state=42,
                                       n_jobs=-1)

random_forest.fit(X_train, Y_train)
Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)

print("oob score:", round(random_forest.oob_score_, 4)*100, "%")
```

```
oob score: 90.56 %
```

Further Evaluation

- **Confusion Matrix:**



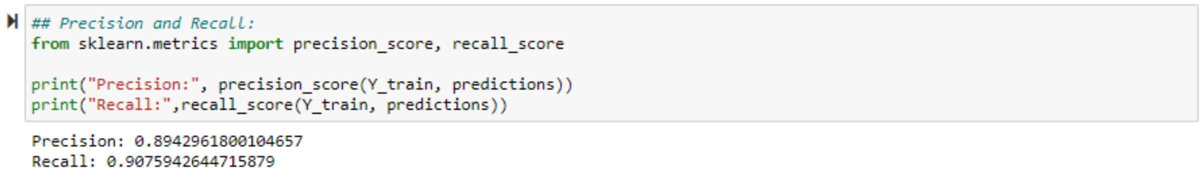
```
## Confusion Matrix:
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
predictions = cross_val_predict(random_forest, X_train, Y_train, cv=3)
confusion_matrix(Y_train, predictions)
```

```
array([[1642,  202],
       [ 174, 1709]], dtype=int64)
```

1642 true negatives and 202 false negatives

174 false positives and 1709 true positives

- **Precision and Recall:**

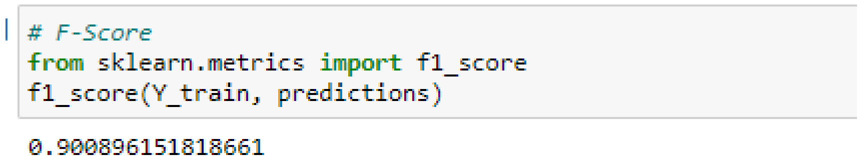


```
## Precision and Recall:
from sklearn.metrics import precision_score, recall_score

print("Precision:", precision_score(Y_train, predictions))
print("Recall:",recall_score(Y_train, predictions))
```

```
Precision: 0.8942961800104657
Recall: 0.9075942644715879
```

Our model predicts 89% of the time, correctly (precision).

- **F-Score**



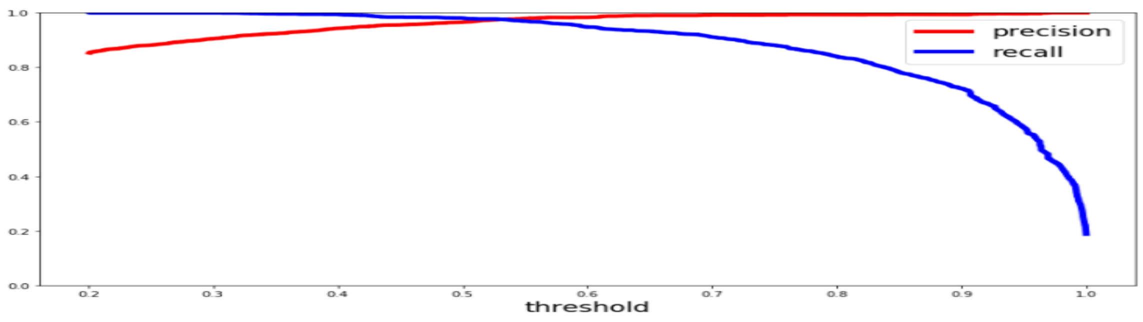
```
# F-Score
from sklearn.metrics import f1_score
f1_score(Y_train, predictions)
```

```
0.900896151818661
```

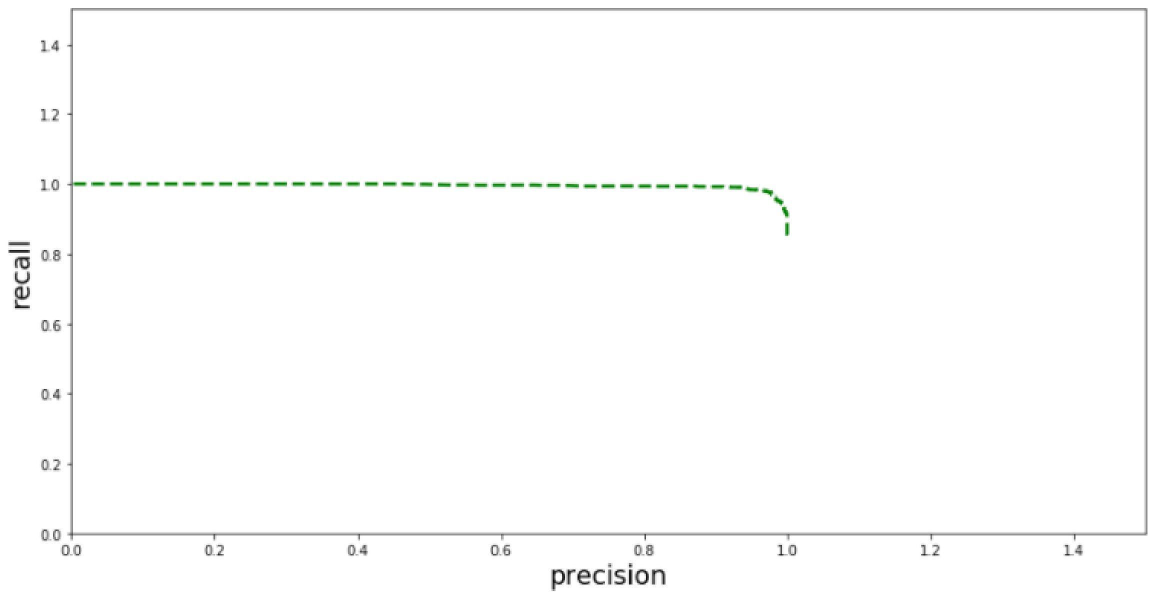
The F-score is a combination of precision and recall

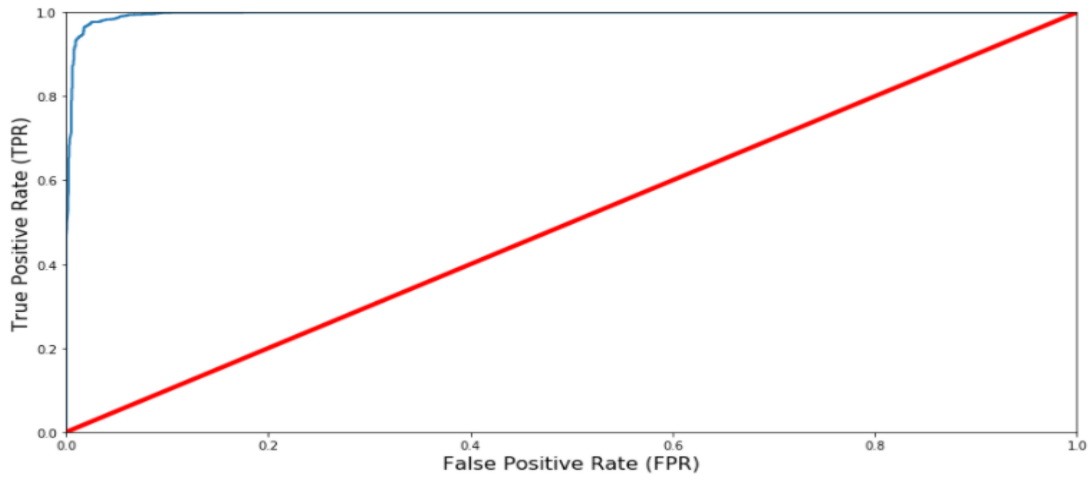
There we have it, a 90 % F-score. The score is high.

- **Precision Recall Curve**



- **<u>ROC AUC Curve</u>**



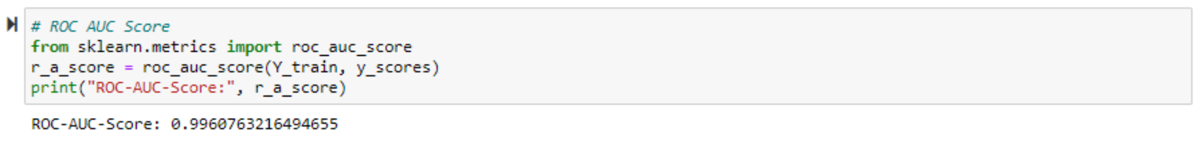


Our ***Random Forest model*** seems to do a good job.

- **<u>ROC AUC Score</u>**

The ROC AUC Score is the corresponding score to the ROC AUC Curve. It is simply computed by measuring the area under the curve, which is called AUC.

A classifier that is 100% correct, would have a ROC AUC Score of 1 and a completely random classifier would have a score of 0.5.



```
# ROC AUC Score
from sklearn.metrics import roc_auc_score
r_a_score = roc_auc_score(Y_train, y_scores)
print("ROC-AUC-Score:", r_a_score)
```

```
ROC-AUC-Score: 0.9960763216494655
```

That score is good enough to submit the predictions for the test-set.