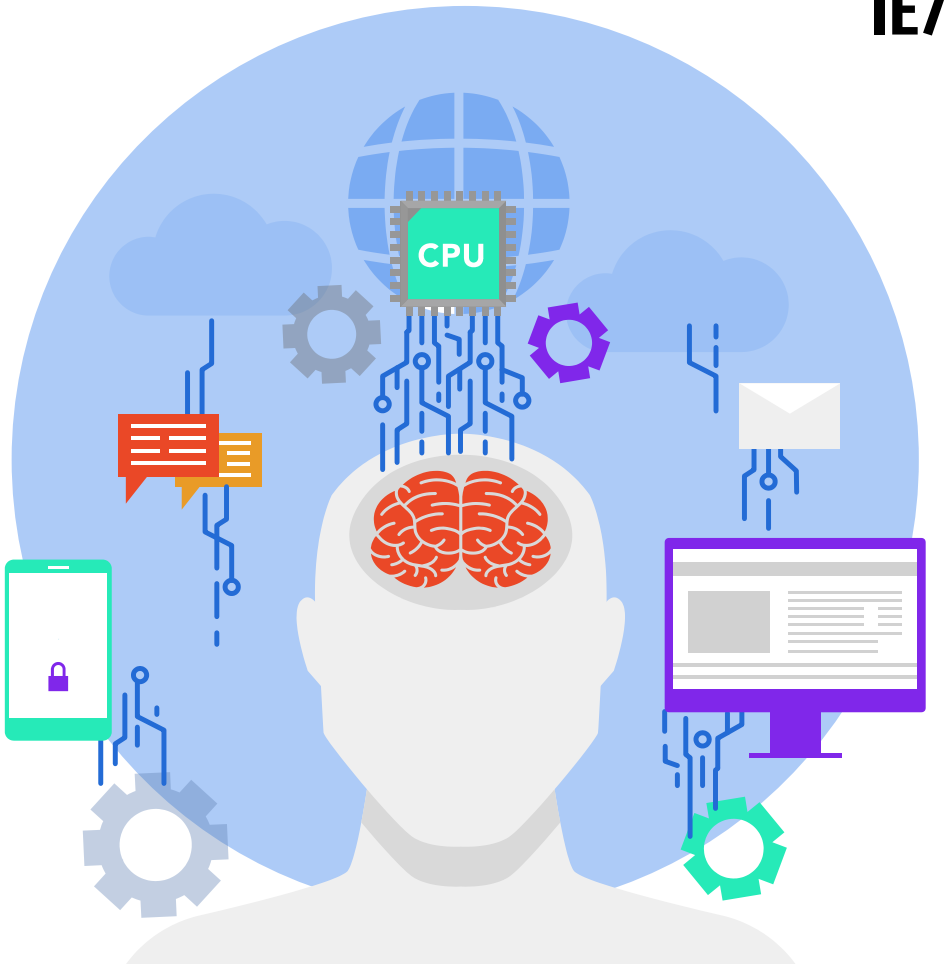


IE7275: Data Mining in Engineering

UNSUPERVISED MACHINE LEARNING Project

Topic:

PDF MALWARE DETECTION

Team Members: **Archit Raj | Raghav Kandpal**

Presentation Date: **04/28/2022**

Table of Content

01 Project Overview

02 Problem Setting

03 Problem Definition

04 Data Sources

05 Data Description

06 Data Exploration

07 Data Mining tasks

08 Data Mining Models/ Methods

09 Performance Evaluation

10 Insights of Decision Making

11 Conclusion, Impact, and Future Extensions

1. Project Overview

History:

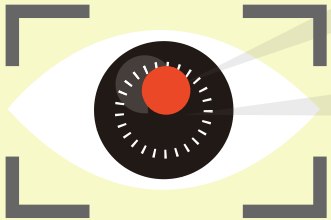
- PDf, Portable Document Format, was created in 1990s by Adobe Systems.
- Used to share documents, including text formatting and inline images
- Mutually compatible and readable file type
- Because of its inclusion of multimedia content embedded, there's a high possibility of containment of malicious entity

Goal of the Project and Application of Unsupervised Machine Learning Model:

- Analyzing the collected pdf dataset
- Applying the Data Mining techniques to understand the data trends between a malicious or benign pdf document.
- Engineering the Supervised ML Model to quantify the results via previous patterns
- Evaluation the performance of the ML model.

2. Problem Setting

What Exactly Was Wrong?



Popularity — One format can hold it all

Feasibility — Most reliable and preserves the document formatting

Target for Hackers: Hackers tend to utilize the advance features by embedding malicious threads to the PDf Documents

3. Problem Definition

OBJECTIVE

OUTCOME

Identify and Analyze

Use concepts of Data Mining and Machine Learning to identify the best model classification from the dataset

Predictions

To create predictors that will correctly detect whether a document is benign or contains malware

4. Data Sources

About the Data Source

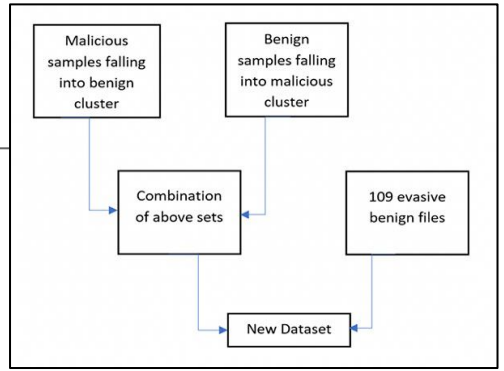
The dataset has been taken from the Evasive-PDFMal2022 created by The Canadian Institute of Cybersecurity at the University of New Brunswick in Fredericton

Data Source

[Link to the Data Source]

UNB
EST. 1785
UNIVERSITY OF NEW BRUNSWICK

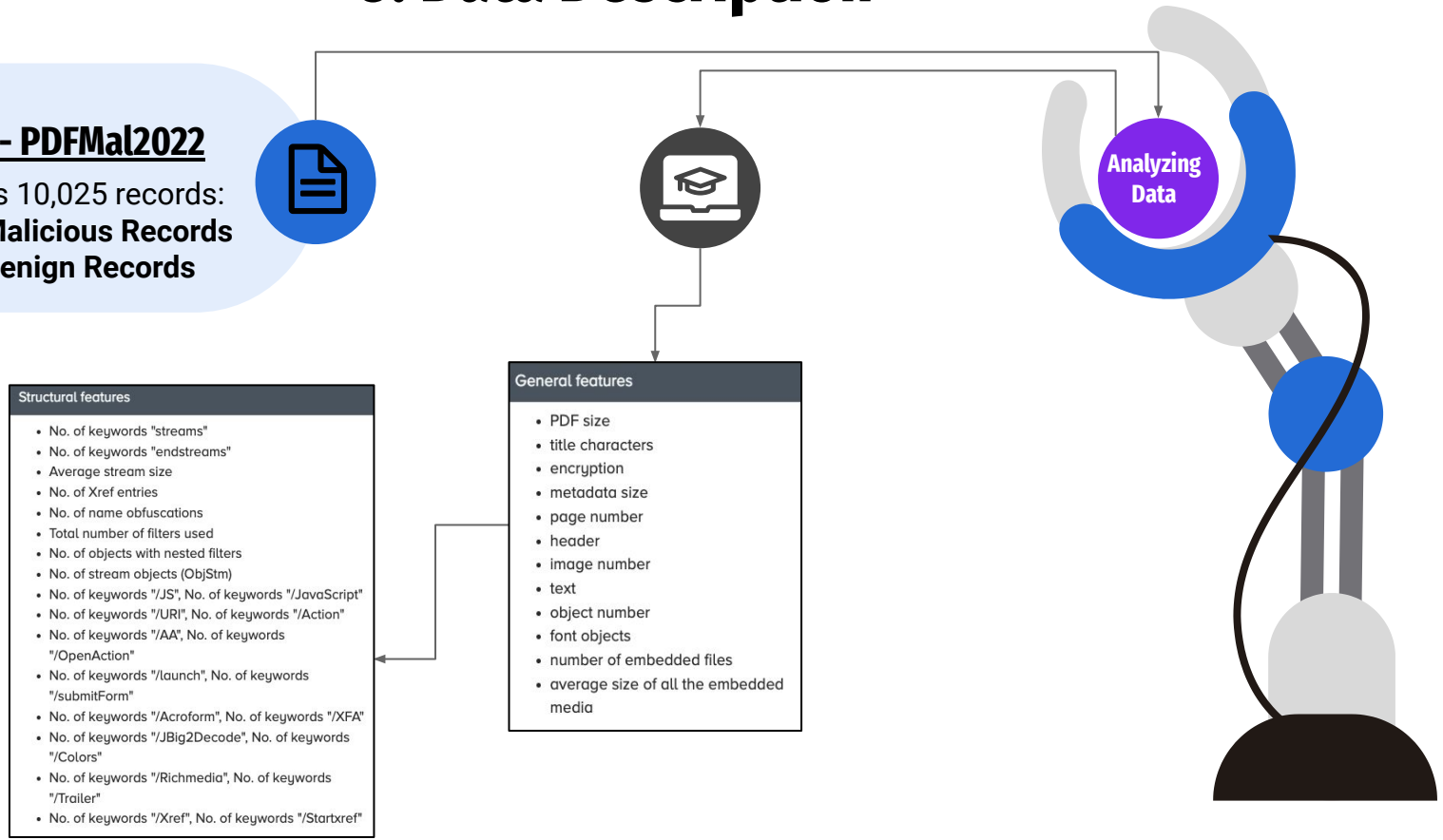
Process of Data Collection:



5. Data Description

Evasive - PDFMal2022

Contains 10,025 records:
- **5,557 Malicious Records**
- **4,468 Benign Records**



General features

- PDF size
- title characters
- encryption
- metadata size
- page number
- header
- image number
- text
- object number
- font objects
- number of embedded files
- average size of all the embedded media

Structural features

- No. of keywords "streams"
- No. of keywords "endstreams"
- Average stream size
- No. of Xref entries
- No. of name obfuscations
- Total number of filters used
- No. of objects with nested filters
- No. of stream objects (ObjStm)
- No. of keywords "/JS", No. of keywords "/JavaScript"
- No. of keywords "/URI", No. of keywords "/Action"
- No. of keywords "/AA", No. of keywords "/OpenAction"
- No. of keywords "/launch", No. of keywords "/submitForm"
- No. of keywords "/Acroform", No. of keywords "/XFA"
- No. of keywords "/JBig2Decode", No. of keywords "/Colors"
- No. of keywords "/Richmedia", No. of keywords "/Trailer"
- No. of keywords "/Xref", No. of keywords "/Startxref"

6.1. Data Exploration

Process Involved

A. Data Collection

01 Importing the libraries and the PDFMalware.csv file on the data frame

```
#Standard Library Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

```
#Load the dataset
from google.colab import files
file = files.upload()  #upload file into google colab session
df_pdf = pd.read_csv("PDFMalware2022.csv")
df_pdf.head()
```

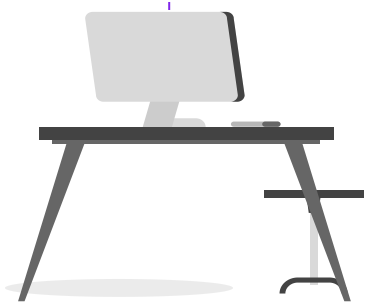
02 Checked the dimension of variables

```
#checking the shape
df_pdf.shape
```

```
(10026, 33)
```

03 Checked the data types using .info()

```
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Fine name           10026 non-null  object
 1   pdfsize             10026 non-null  float64
 2   metadata size       10026 non-null  float64
 3   pages               10026 non-null  float64
 4   xref Length         10026 non-null  float64
 5   title characters    10026 non-null  float64
 6   isEncrypted         10026 non-null  float64
 7   embedded files      10026 non-null  float64
 8   images              10026 non-null  object
 9   text                10026 non-null  object
 10  header              10026 non-null  object
 11  obj                 10026 non-null  object
 12  endobj              10026 non-null  object
 13  stream              10026 non-null  float64
 14  endstream           10026 non-null  object
 15  xref                10026 non-null  object
 16  trailer             10026 non-null  float64
 17  startxref           10026 non-null  object
 18  pageno              10026 non-null  object
 19  encrypt             10026 non-null  float64
 20  ObjStm              10026 non-null  float64
 21  JS                  10026 non-null  object
 22  Javascript          10026 non-null  object
 23  AA                  10026 non-null  object
 24  OpenAction          10026 non-null  object
 25  Acroform            10026 non-null  object
 26  JBIG2Decode         10026 non-null  object
 27  RichMedia           10026 non-null  object
 28  launch              10026 non-null  object
 29  EmbeddedFile        10026 non-null  object
 30  XFA                 10026 non-null  object
 31  Colors              10026 non-null  float64
 32  Class               10026 non-null  object
dtypes: float64(12), object(21)
```



B. Data Processing

01 Checking the Null Values using .isna().sum()

```
Fine name           0
pdfsize             1
metadata size       1
pages               1
xref Length         1
title characters    1
isEncrypted         1
embedded files      1
images              1
text                1
header              1
obj                 3
endobj              3
stream              3
endstream           3
xref                3
trailer             3
startxref           3
pageno              3
encrypt             3
ObjStm              3
JS                  3
Javascript          3
AA                  3
OpenAction          3
Acroform            3
JBIG2Decode         3
RichMedia           3
launch              3
EmbeddedFile        3
XFA                 3
Colors              3
Class               1
dtype: int64
```

02 Filling the missing values with 0s and removing duplicate values

```
#Filling the missing values
df_pdf.fillna(0, inplace=True)
```

```
#Checking Duplicate Rows
duplicate_rows_df = df_pdf[df_pdf.duplicated()]
duplicate_rows_df.shape
```

6.2. Data Exploration

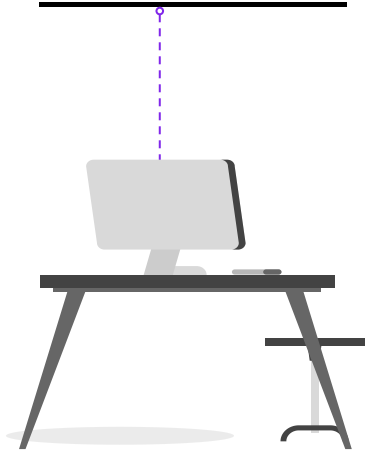
Process Involved

Data Exploration

01 **Target Variable:** Malicious 1s and Benign 0s

```
df_pdf["Class"].value_counts()

1    5558
0    4468
Name: Class, dtype: int64
```



02 **Predictor Variables:**

PdfsizeMetadata size, pages, xref Length, title characters, isEncrypted, Embedded files, images, text, header, obj, endobj, stream, endstream, xref, trailer, startxref, pageno, encrypt, ObjStm, JS, JavaScript, AA, OpenAction, Acroforrm, JBIG2Decode, RichMedia, launch, XFA, Colors.

03 **Facts to be Considered:**

- **PDF size:** The malicious PDF size usually tends to vary from the benign due to its variation in page size and content.
- **Metadata size**: It has a direct relation with the type of Class(Malicious or Benign). Metadata is the section where information about the PDF file is provided, which can be exploited for embedding hidden contents.
- **Document Encryption**: This feature shows whether the PDF document is password protected or not.
- **Pages**: Malicious PDF files tend to have fewer pages (most of them have one blank page) as they are not concerned about content presentation.
- **Text**: Content presentation is not the objective of malware PDF files, they may include less text in their files.
- **Size:** The malicious PDF size usually tends to vary from benign due to its variation in page size and content.
- **Embedded Files:** PDFs are capable of attaching or embedding different types of files within themselves that might be used for exploitation, including other PDF files, doc files, images, etc.
- **Embedded media:** The count of Embedded files in the PDF might lead to an insight into the content of the embedded files.
- **Objects Count:** As PDFs are made of objects, the number of objects combined with the rest of the features can represent the PDF in general.
- **Count of font objects:** Font objects indicate the types of fonts used for the PDF text.
- **PDF Header:** As PDF header obscure is common for evading anti-virus scans, malicious PDF files tend to modify the header format.
- **Image**: PDF files may contain one or any number of images.

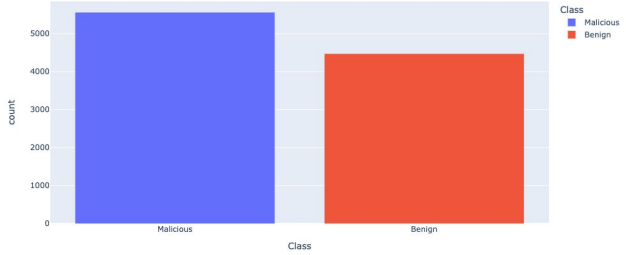
7.1. Data Mining Tasks

Data Visualization:

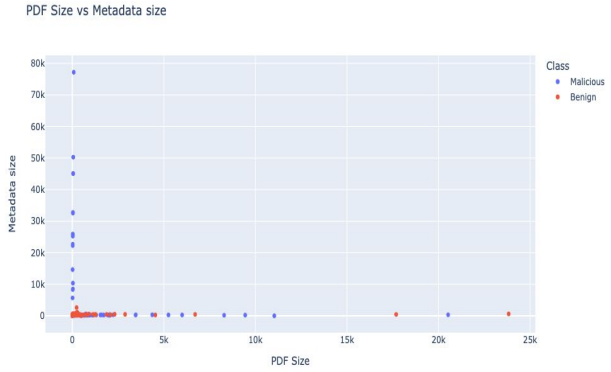
(With the help of Python's Seaborn and Plotly libraries)

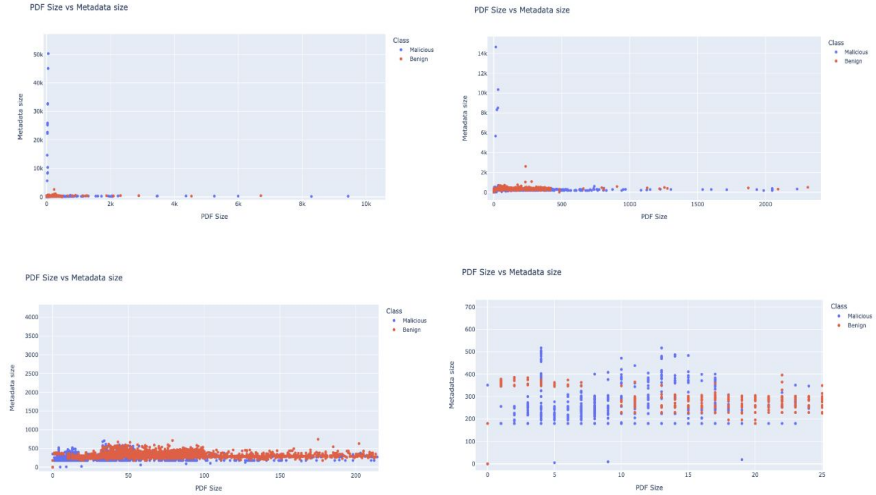
1. Class Distribution:

The distribution of class is checked to ensure the set to be somewhat balanced.



2. Scatter Plot:





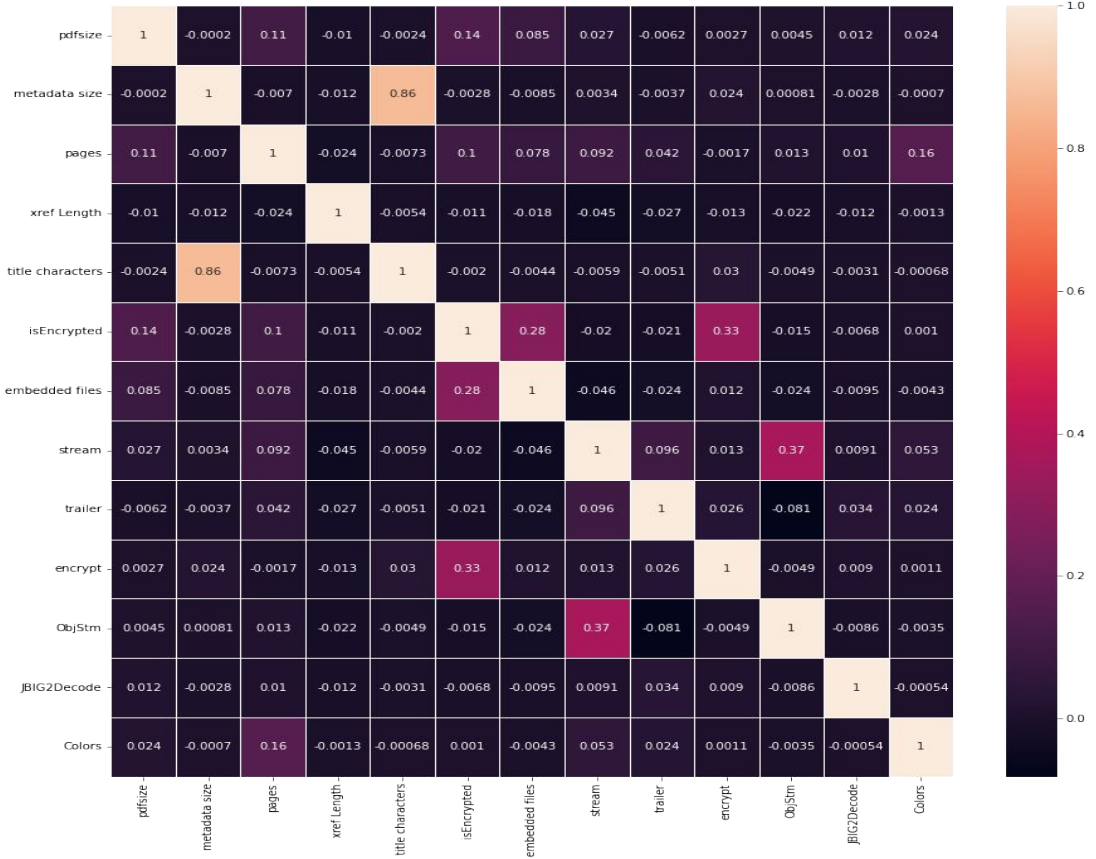
7.2. Data Mining Tasks

Data Visualization:

(With the help of Python's Seaborn and Plotly libraries)

3. Correlation Plot

- Correlation Plot between all the numerical variables.
- Metadata Size and Number of Title Characters are highly positively correlated (0.86).



8. Data Mining Models/ Methods

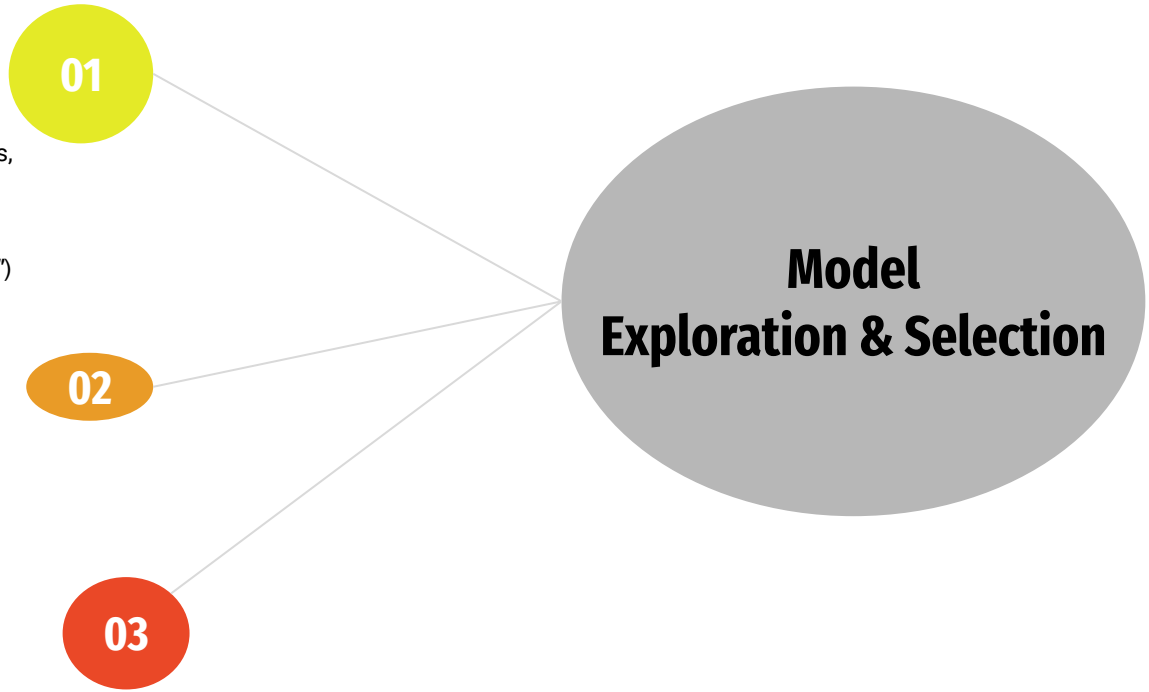
Selecting the Model for Prediction

Response Variable:

- 1 as Malicious and 2 as Benign

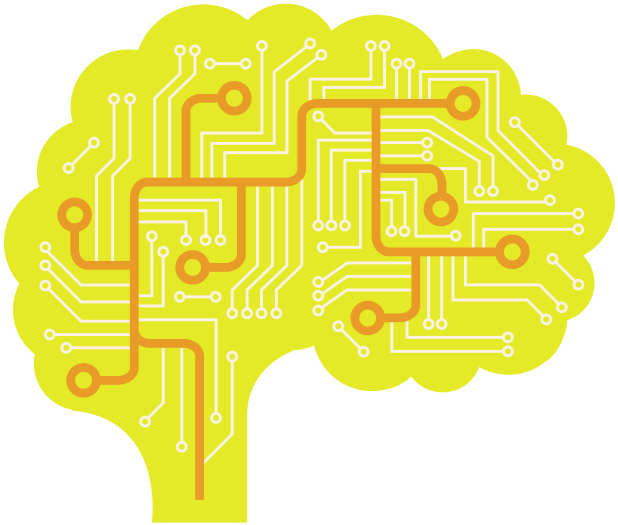
Predictor Variables:

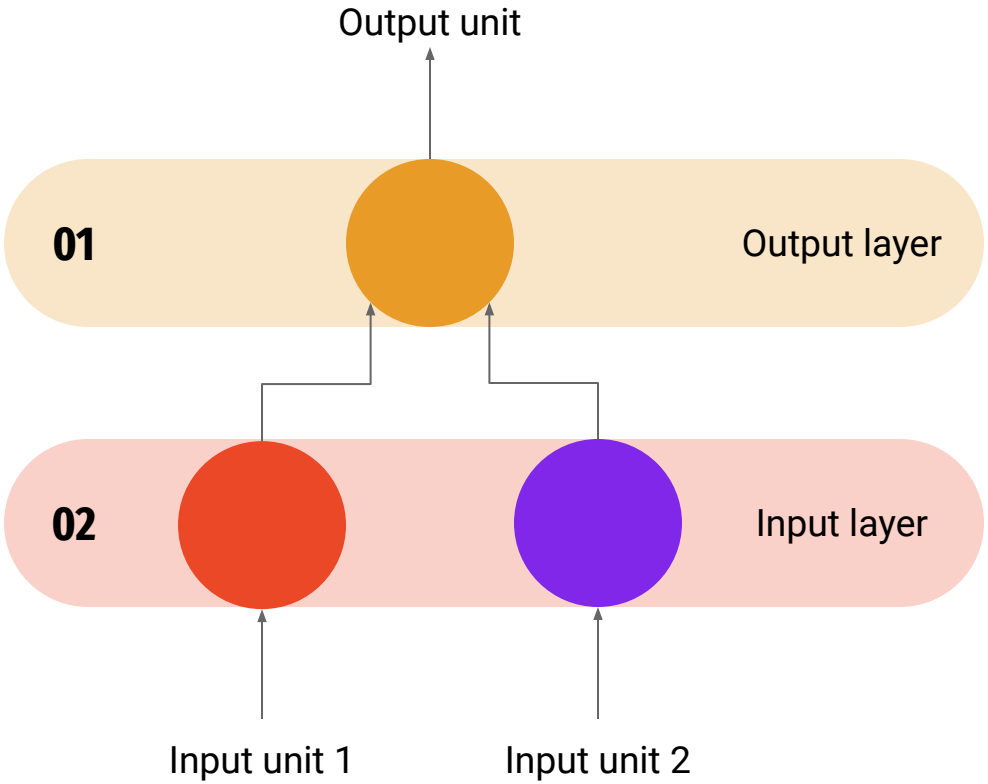
PDF Size, Metadata Size, Number of Pages, XREF Length, Title Characters, Trailer, Stream, Embedded Characters, No. of stream objects (ObjStm), No. of keywords "/JBig2Decode", No. of keywords *"/Colors"*)



8. Data Mining Models/ Methods

Neural network
Yes, Saturn is a gas giant that has rings





9. Performance Evaluation

9.1. Implementation of Selected Model: SVM

Importing the libraries & dataset 01

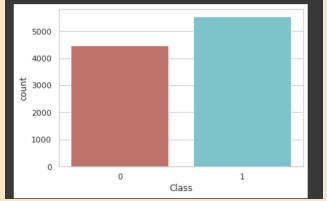
```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

```
from google.colab import files
file = files.upload()
```

```
Choose Files  PDFMalware2022 (1).csv
• PDFMalware2022 (1).csv(text/csv) - 2351532 bytes, last modified: 4/10/2022 - 100% done
Saving PDFMalware2022 (1).csv to PDFMalware2022 (1).csv
```

Creating the histogram to check the proportion of Malicious(1) and Benign file type using Seaborn library(0) 02

```
sns.countplot(x='Class', data = df, palette='hls')
plt.show()
plt.savefig('count_plot')
```



Splitting the data into train and test using SMOTE technique 04

```
[13] from sklearn.model_selection import train_test_split
     from imblearn.over_sampling import SMOTE

[14] y = df['Class']
     X = df.drop(['File name', 'Class'], axis=1)

[15] os = SMOTE(random_state=0)
     columns = X.columns

     os_X,os_y = os.fit_resample(X, y)
     os_X = pd.DataFrame(data=os_X,columns=columns )
     os_y= pd.DataFrame(data=os_y,columns=['Class'])
```

Checked the size of Oversampled data, Num of No Subscription in Oversampled Data, Proportion of Subscription and No Subscription Data 05

```
print("length of oversampled data is ",len(os_X))
print("Number of no subscription in oversampled data",len(os_y[os_y['Class']==0]))
print("Number of subscription",len(os_y[os_y['Class']==1]))
print("Proportion of no subscription data in oversampled data is ",len(os_y[os_y['Class']==0])/len(os_X))
print("Proportion of subscription data in oversampled data is ",len(os_y[os_y['Class']==1])/len(os_X))
```

```
length of oversampled data is  11070
Number of no subscription in oversampled data 5535
Number of subscription 5535
Proportion of no subscription data in oversampled data is  0.5
Proportion of subscription data in oversampled data is  0.5
```

Used Standard Scalar in order to fit the Oversampled data and utilized Recursive Feature Elimination algorithm to test the accuracy 06

```
[17] from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler().fit(os_X)
     os_X_scaled = pd.DataFrame(scaler.transform(os_X), columns=os_X.columns)

[18] from sklearn.feature_selection import RFE
     from sklearn.svm import SVC
     estimator = SVC(kernel="linear")
     rfe = RFE(estimator, n_features_to_select=20)
     rfe = rfe.fit(os_X_scaled, os_y.values.ravel())
     cols_to_keep = rfe.support_
     print(cols_to_keep)
```

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

Continued

9. Performance Evaluation

9.1. Implementation of Selected Model: SVM

Applying GridSearch Cross Validation to better tune the hyperparameters of the Support Vector Classifier

07

```
[19] cols = np.array(os_X.columns)[np.array(cols_to_keep)]

[20] X = os_X_scaled[cols]
     y = os_y

[21] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0, shuffle = True)

[22] svm = SVC(kernel="linear")
     svm.fit(X_train, y_train)
     predicted = svm.predict(X_test)

     /usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vec
       y = column_or_1d(y, warn=True)

[24] from sklearn import metrics

[25] metrics.accuracy_score(y_test, predicted)

     0.9494219653179191
```

Lastly applying SVM(Supply Vector Classifier) to fit our Split data to return the best fit hyperplane dividing it into an accuracy score of **0.9494219653179191.**

Continued

9. Performance Evaluation

9.1. Implementation of Selected Model: SVM

Continued

Converting the Columns in arrays (with NumPy) and then Splitting our data into 25% test size

07

```
[19] cols = np.array(os_X.columns)[np.array(cols_to_keep)]

[20] X = os_X_scaled[cols]
     y = os_y

[21] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0, shuffle = True)

[22] svm = SVC(kernel="linear")
     svm.fit(X_train, y_train)
     predicted = svm.predict(X_test)

     /usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vec
       y = column_or_1d(y, warn=True)

[24] from sklearn import metrics

[25] metrics.accuracy_score(y_test, predicted)

     0.9494219653179191
```

Lastly applying SVM(Supply Vector Classifier) to fit our Split data to return the best fit hyperplane dividing it into an accuracy score of **0.9494219653179191**.

9. Performance Evaluation

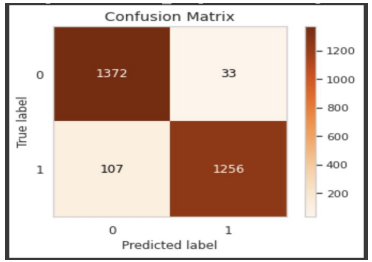
9.2. Performance Evaluation

Classification Report with y_test and Predicted Values

	precision	recall	f1-score	support
0	0.93	0.98	0.95	1405
1	0.97	0.92	0.95	1363
accuracy			0.95	2768
macro avg	0.95	0.95	0.95	2768
weighted avg	0.95	0.95	0.95	2768

10.1. Insights of Decision Making

Confusion Matrix



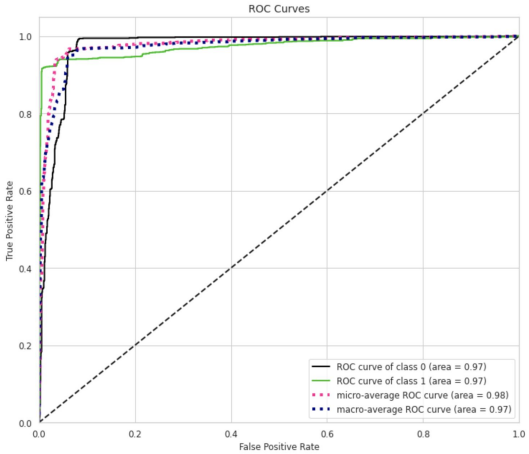
01

Interpretation Based on the Predicted Model

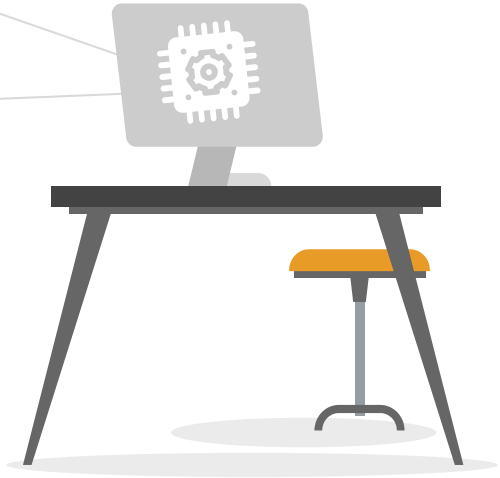
The confusion matrix let us analyze how our classification algorithm is doing from our various other classes of data.

02

ROC Curve

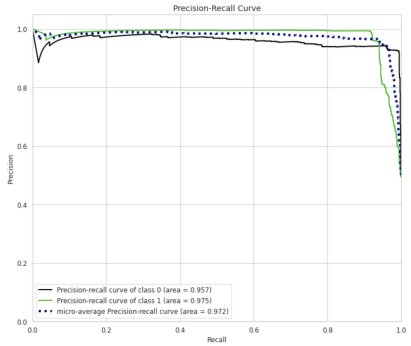


ROC Curve defines the random guess model covering 50% area of ROC curve showing the area of class 0 ROC curve as 0.97, area of class 1 ROC Curve as 0.97, area of Micro-average ROC curve as 0.98 and area of macro-average ROC Curve as 0.97



10.2. Insights of Decision Making

Precision-Recall Curve

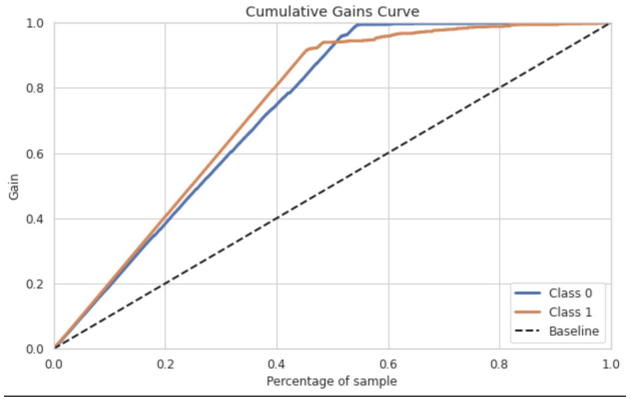


03

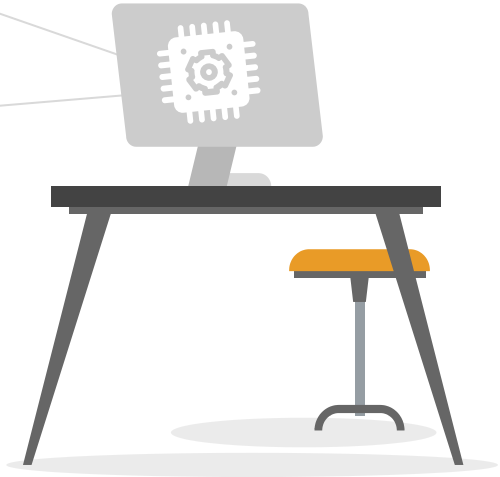
Interpretation Based on the Predicted Model

Target variable 'Class' is more than 97% which is good. Therefore, we can assume each class covers more than 95% area so that we can be sure that our model is doing well predicting each class even in an imbalanced dataset situation.

Culmination Gains Curve



04



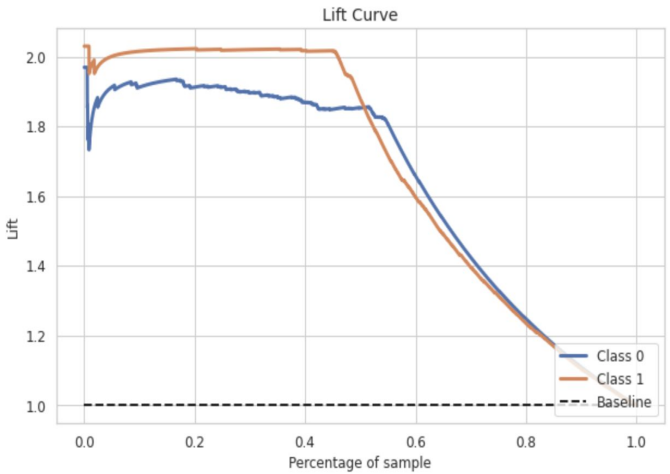
Percentage of samples in a given category, Class 0 and 1, which were truly predicted by targeting a percentage of the total number of samples, which also means that we took that many percentages of samples from the total percentage that we get from the curve for y-axis are labels which were truly guessed by model from a total number of samples of that class in that many samples. The dashed line in the chart is the baseline curve (random guess model) and our model should perform better than it and both class curves should be above it ideally.

10.3. Insights of Decision Making

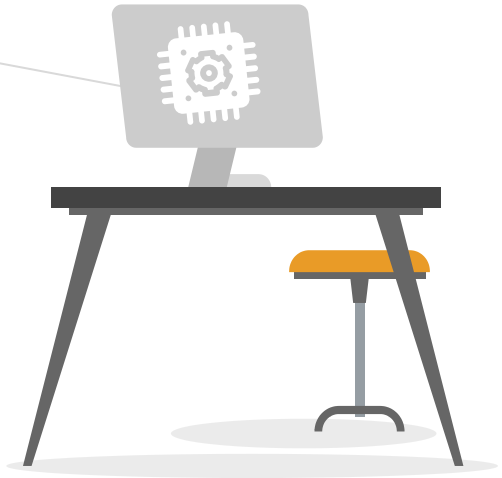
Interpretation Based on the Predicted Model

05

Lift Curve



Cumulating the chart by taking a ratio of cumulative gains for each curve to the baseline and showing this ratio on its y-axis.



11. Conclusion, Impact & Future Extensions

- Based on the prediction analysis, this project can be used by many multinational companies to check and make it easier for the users to understand different techniques to breakdown malicious vs benign file type.

- According to the higher accuracy score, we could analyze how SVM Machine learning model classify different outcomes from target variable Class: Malicious vs Benign with respect to Predictive variables.