# Artificial Intelligence

**Project Synopsis On :**

## Malacious PDF Detector


**SUBMITTED TO: Ashima Khosla**

**SUBMITTED BY:**

Kritika (102303415)

Arsh Garg (102303414)

# PDF Malware Detection


## Abstract-

This report includes an extensive investigation of malicious PDF file detection using state-of-the-art machine learning methods. Malicious PDFs are an emerging cybersecurity threat that tends to leverage software vulnerabilities to inject malicious payloads. Based on the PDFMalware2022 dataset [1], this project explores strong data preprocessing methods, feature engineering techniques, and the deployment of classification models, namely Random Forest. The models are critically tested with accuracy measures and visual diagnostics like ROC curve and Confusion matrix. Experiments show that strategic feature selection and dimensionality reduction methods like LDA not only improve model performance but also cut down computational overhead, pointing towards the potential of machine learning for proactive malware detection.


### Introduction

PDF (Portable Document Format) documents are an essential part of contemporary digital communication. Their predictable layout, platform independence, and multimedia content support make them the document of choice for distributing official documents, reports, academic papers, and business records. In government, education, and corporate settings alike, PDFs have become a benchmark for secure, tamper-proof document transfer.

Yet, the very characteristics that contribute to the popularity of PDFs also create enormous security threats. PDFs can include scripts, interactive forms, and multimedia content—capabilities that cybercriminals increasingly use to deploy malware and carry out malicious operations on victim machines. PDFs offer an intricate/complex architecture that can be manipulated by attackers to hide malicious payloads like JavaScript-based exploits or references to harmful resources.

In the past few years, PDF-based malware has proven to be a silent and powerful attack channel, frequently evading traditional antivirus software that relies on signature-based detection. Such traditional approaches find it difficult to detect zero-day attacks and obfuscated threats, exposing systems to new and concealed malware.

This increasing threat environment emphasizes the requirement for smart, behavior-based detection technology able to actively flag suspicious files—inclusive of those never before encountered. This project seeks to create a machine learning-based classification

model with a high degree of accuracy in separating malicious and benign PDF files based on structural and behavioral features learned from the PDFMalware2022 dataset [1].

## Problem Statement

Conventional antivirus software is mostly based on signature-based detection techniques, which are not effective against zero-day attacks and obfuscated threats. There is an urgent need for intelligent systems that can identify unknown threats based on behavioral and structural features of PDF files. This project tackles the problem of developing an accurate and efficient classification model to separate benign and malicious PDF files using the PDFMalware2022 dataset [1].

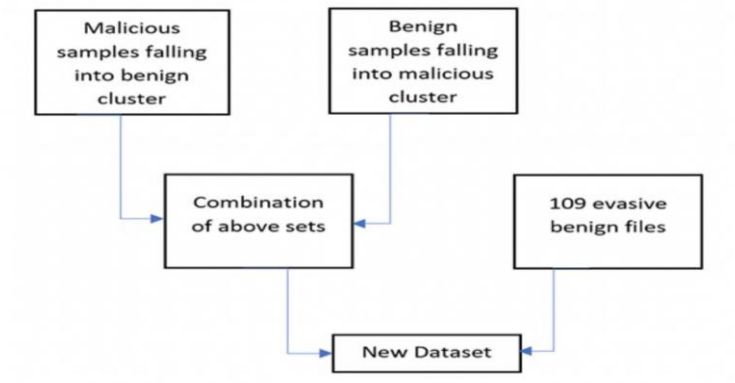
## Objectives

The primary objectives of this study are:

1. To preprocess and clean the PDFMalware2022 dataset [1].
2. To perform feature selection and dimensionality reduction.
3. To apply machine learning models for malware detection (**Random Forest**).
4. To evaluate model performance using various metrics.

## Dataset-

The CIC-Evasive-PDFMal2022 dataset [1], created by the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick, is an extensive dataset created to support the detection of evasive PDF malware. In response to the common occurrence of PDF files and their abuse by attackers, this dataset was created with the intention of being a challenging test case for assessing malware detection methods.



**Key Features-** 10,025 PDF files

- **Malicious Samples:** 5,557
- **Benign Samples:** 4,468

| General features | Structural features |
|---|---|
| • PDF size<br>• title characters<br>• encryption<br>• metadata size<br>• page number<br>• header<br>• image number<br>• text<br>• object number<br>• font objects<br>• number of embedded files<br>• average size of all the embedded media | • No. of keywords "streams"<br>• No. of keywords "endstreams"<br>• Average stream size<br>• No. of Xref entries<br>• No. of name obfuscations<br>• Total number of filters used<br>• No. of objects with nested filters<br>• No. of stream objects (ObjStm)<br>• No. of keywords "/JS", No. of keywords "/JavaScript"<br>• No. of keywords "/URI", No. of keywords "/Action"<br>• No. of keywords "/AA", No. of keywords "/OpenAction"<br>• No. of keywords "/launch", No. of keywords "/submitForm"<br>• No. of keywords "/Acroform", No. of keywords "/XFA"<br>• No. of keywords "/JBig2Decode", No. of keywords "/Colors"<br>• No. of keywords "/Richmedia", No. of keywords "/Trailer"<br>• No. of keywords "/Xref", No. of keywords "/Startxref" |

## Methodology

**1.Data Preprocessing:** Before training a machine learning model, it is important to preprocess the dataset to make it clean, consistent, and analyzable. Preprocessing of data is foundational in making the final model clean and accurate. The preprocessing steps followed are listed below:

- **<u>Dropping Rows with Missing Target Labels</u>-** In supervised learning, the model is trained on labeled data wherein every input is accompanied by a known output (Benign or Malicious). If the target label is not present, there is no basis for the model to identify what the file is, thus rendering that data useless for training. Such rows were deleted to preserve data integrity and prevent misleading the model.

Drop rows with missing target and copy

```
[45]: df = df.dropna(subset=["Class"]).copy()
      df["Class"] = df["Class"].map({"Malicious": 1, "Benign": 0})
```

- **Converting Class Labels to Binary Values**- The Class column initially had string values namely “Malicious" and "Benign". Machine Learning algorithms take only numeric input. Thus, the labels were made into binary values 1 for Malicious and 0 for Benign. This also makes performance metrics calculation easier, like accuracy, precision, and recall.

- **Removing the Identifier Column-** The File name column is merely a sample-specific unique identifier and contains no meaningful information regarding the structure or content of the PDF file itself. Including it in the training set would introduce noise, potentially causing the model to learn unhelpful patterns or overfit on particular file names. Therefore, it was eliminated.

Drop identifier column

```
[46]: if "File name" in df.columns:
          df = df.drop(columns=["File name"])
```

- **Replacing Values of -1 and -1.00 with 0**- Some of the feature values in the data were found as -1 or -1.00, which indicate missing, null, or non-applicable data. These placeholder values would affect model learning, particularly for algorithms that operate under the expectation that numerical magnitude is meaningful. These were changed to 0 to normalize the data and not mislead the model into seeing -1 as a valid input.

Replacing values of -1 with 0

```
[47]: df = df.replace([-1, -1.00],0)
```

- **Imputing missing values using Simple Imputer**- Certain features held missing or null values. Rather than dropping such rows or columns (which result in loss of useful data), missing values were replaced by the Simple Imputer method:
    - For numeric features, missing values were filled with the column's mean or median.
    - This helps maintain the general distribution of the data and enables the model to learn on an intact dataset without bias.

Impute missing numeric values

```
[48]: num_cols = df.select_dtypes(include=[np.number]).columns
if df.shape[0] > 0 and len(num_cols) > 0:
    imputer = SimpleImputer(strategy="mean")
    df[num_cols] = pd.DataFrame(imputer.fit_transform(df[num_cols]), columns=num_cols, index=df.index)
```

- **Encoding Categorical columns**- In order to prepare the dataset for machine learning algorithms, categorical variables need to be converted into a numerical format. In this project, data type object was used to identify columns that were categorical and labeled using the LabelEncoder from the sklearn.preprocessing module. This enables all the non-numeric attributes to be converted to numerical values, making it possible for machine learning models to interpret and process the data more effectively. Encoding serves to preserve the data's structural integrity without introducing bais, which is particularly significant in the case of metadata or text fields within PDFs.

Encode categorical columns

```
[49]: cat_cols = df.select_dtypes(include=["object"]).columns
for col in cat_cols:
    df[col] = LabelEncoder().fit_transform(df[col].astype(str))
```

**2.Data Visualization**-Data visualization is a starting point for any machine learning or data science project since it serves to naturally explore features of the dataset[2]. Through converting raw data into graphical representations like bar plots or histograms, we are in a position to pick up patterns, outliers, or class distributions that impact the way a model needs to be built or tested. In classification problems, particularly, visualization aids in the detection of class imbalance—a problem that can cause biased models unless it is fixed. Visualization also informs the selection of evaluation metrics.

- **Data Imbalance**-During the model evaluation stage, it is crucial to determine if the dataset is imbalanced by class [3]. Within my project, A graphical check was done for class distribution with a countplot of the Class feature, which is displayed in the figure below-

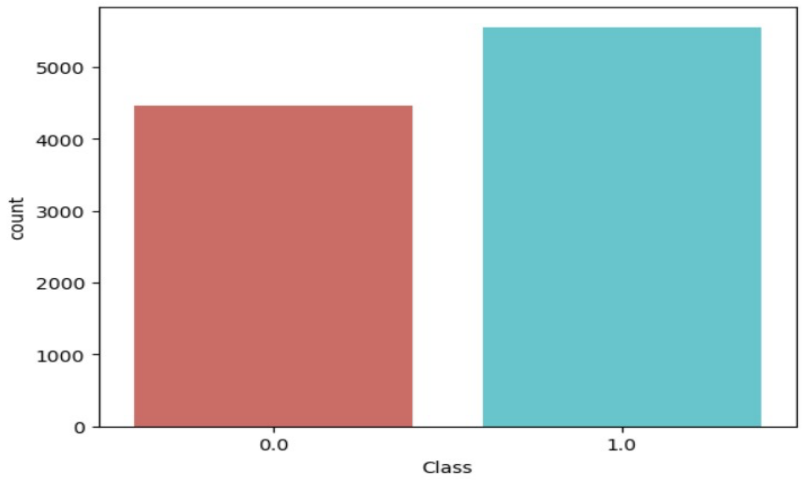



**Fig.Count of Malicious VS Benign Class**

The graph clearly shows that there is a moderate imbalance between the two classes. Class 0 (benign) contains fewer samples than Class 1 (malicious), but the disparity is not excessive. It is important to understand class imbalance since the majority of machine learning algorithms rely on the equal distribution of classes. When this does not hold, models may attain high accuracy by predicting the majority class all the time, neglecting the minority class, which is usually the class of more interest (malicious).

With class imbalance, employing accuracy as a measurement would be unfair. The model may simply output the majority class and still do very well based on accuracy but entirely miss handling the minority class. To counter this, I utilize the **F1-score**[3], which is the harmonic mean of precision and recall. It is a balanced measure allows a more accurate representation of the model's performance with imbalanced datasets.

- Precision ensures that the class predicted by the model (e.g., malicious) is indeed correct.
- Recall ensures that the model has correctly identified all true instances of the class
- **F1-score** combines both, which is necessary in situations such as threat detection, where both false negatives and false positives are expensive [3].

Checking Class Imbalance

[50]:
```
sns.countplot(x='Class', data=df,hue='Class', palette='hls',legend=False)
plt.show()
print('Percentage of Malicious vs Benign Classes')
df['Class'].value_counts()/len(df) * 100
```

- **PDF Size VS Metadata Size-** This plot was generated to investigate possible patterns in the structure of PDF files, specifically between the size of the file and the size of its metadata [4]. It assists in the comprehension of how malicious (Class 1) and benign (Class 0) PDFs can structurally differ. The graph demonstrates that although most PDFs group around smaller sizes, a significant number of malware files have disproportionately huge metadata sizes even when their overall file size is moderate. This may indicate methods like metadata bloating, which in some cases are used in malwares. Such findings justify further feature selection and more in-depth forensic analysis of PDFs.

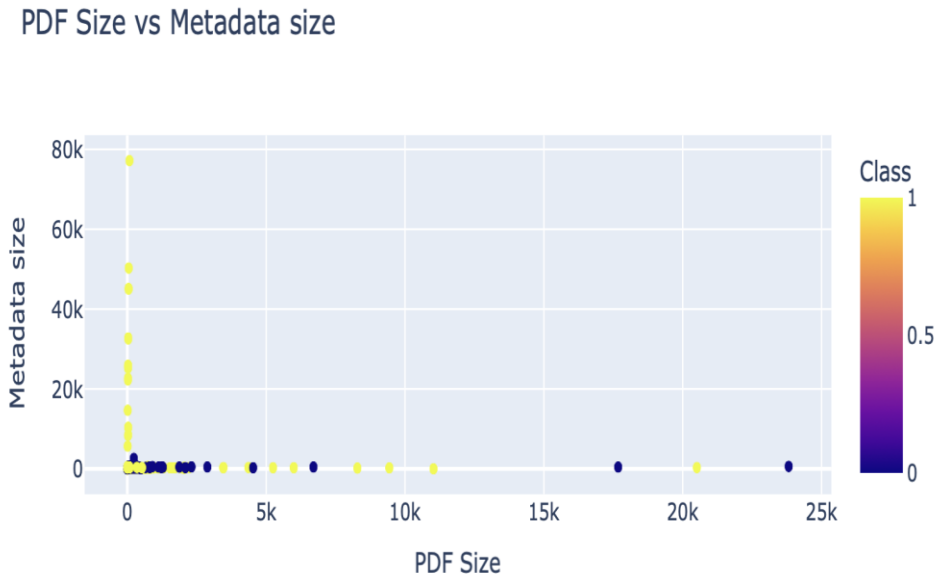



**Fig.PDF Size vs Metadata Size**

PDF Size vs Metadata size

```
[2]: import plotly.express as px
fig = px.scatter(x=df['pdfsize'], y=df['metadata size'], color=df['Class'],
                 labels={'x':'PDF Size', 'y':'Metadata size', 'color':'Class'}, title='PDF Size vs Metadata size')
fig.show()
```

- **Images in PDF-** The PDF Images analysis was added as an exploratory phase to determine if the occurrence or the number of images in a PDF has any bearing

on its labeling as malicious or benign. By grouping files by the number of embedded images (e.g., none, 1–5, 6–20, etc.), we can see trends that may indicate structural differences between the two classes. For example, malicious PDFs may not use images to save space and avoid detection, or, on the other hand, use a lot of images as a cover. These observations can inform future research in choosing structural features for malware detection in documents.

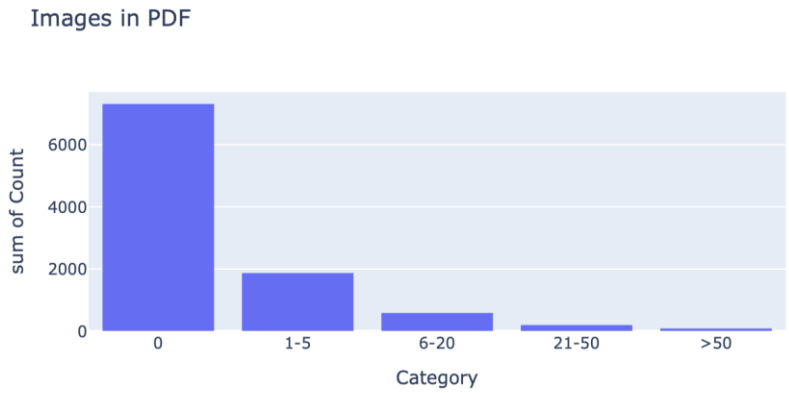



**Fig.Images in pdf**

Images in PDF

```
[52]: img = df.images
imgs={'0':0,'1-5':0,'6-20':0,'21-50':0,'>50':0}
for i in img:
    if i == '1(1)':
        next
    elif int(i) == 0:
        imgs['0']+=1
    elif int(i)>0 and int(i)<6:
        imgs['1-5']+=1
    elif int(i)>5 and int(i)<21:
        imgs['6-20']+=1
    elif int(i)>20 and int(i)<51:
        imgs['21-50']+=1
    else:
        imgs['>50']+=1

imgs_df = pd.DataFrame(imgs.items())
imgs_df.columns=['Category','Count']

px.histogram(imgs_df, x='Category', y='Count', title='Images in PDF')
```

**3. Feature Selection:** Feature engineering is a vital process for enhancing model performance by enriching the dataset with the most suitable and clean features. The following steps were used:

- **Removing highly correlated features**- Highly correlated features can cause redundancy and multicollinearity, which can deteriorate model's interpretation and performance. A correlation matrix (Heat Map) was computed to detect such

correlations. This reduces the dataset and allows models to concentrate on independent, informative features.

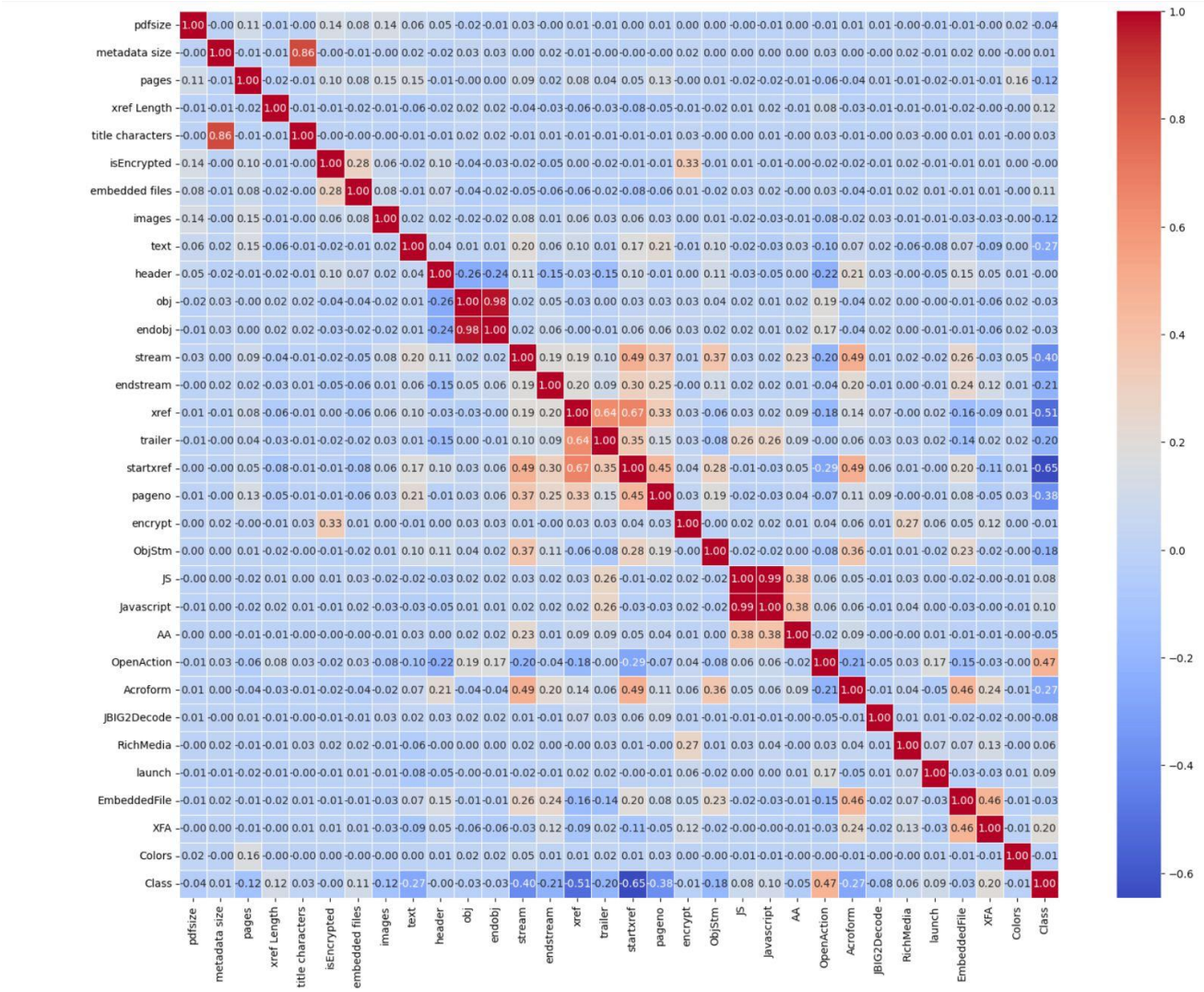



**Fig.Heat Map**

Heapmap to remove highly Correlated features

```
[54]: df_numeric = df.select_dtypes(include=[np.number])
      f, ax = plt.subplots(figsize=(20, 15))
      sns.heatmap(df_numeric.corr(), annot=True, fmt=".2f", linewidths=.5, ax=ax, cmap='coolwarm', square=True)
      plt.show()
```

As we can observe from the heatmap any one feature from endobj – obj and JS – Javascript is eliminated as these two pair of features have correlation factor > 0.90.

```
[55]: df = df.drop(['endobj','JS'], axis=1)
```

- **Dropping Low-Variance Columns-** Low-variance columns were excluded from the dataset. These columns contain little or no variability between observations and will not likely contribute valuable information to modeling.

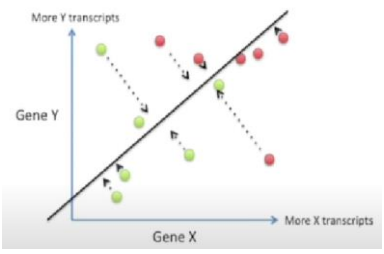
Drop low-variance columns

```
[53]: df = df.loc[:, df.std() > 0]
```

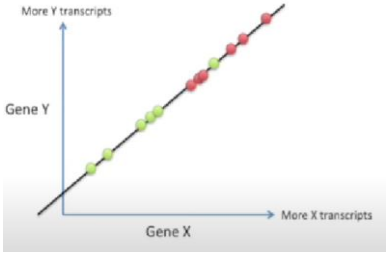
- **Applying LDA for dimensionality reduction-** In this project, Linear Discriminant Analysis (LDA) was used for dimensionality reduction. In contrast to unsupervised techniques such as PCA, LDA is a supervised learning method that considers class labels. Its main objective is to minimize the feature space while maximizing class separability.
  LDA operates by:
  - Computing the directions that are the axes which maximize the distance between several classes.
  - Mapping the data onto a lower-dimensional space that preserves as much class-discriminative information as possible.

  This not only enhances computational efficiency but also allows classification models to concentrate on the most informative features for differentiating between benign and malicious PDF files. LDA is particularly well-suited for this binary classification problem, as it efficiently reduces the information into a form that maximizes model performance.

**Before LDA**




**After LDA**




Apply LDA (1 component for binary classification)

```
[58]: lda = LDA(n_components=1)
      X_lda = lda.fit_transform(X_scaled, y)
      joblib.dump(lda, 'lda.pkl')
```

**4.Feature Scaling-** Feature scaling is an important machine learning preprocessing step that makes all numerical features equally influence model training. For this project, the StandardScaler from sklearn.preprocessing was utilized to scale the features by rescaling to unit variance. This operation assists distance-based calculation algorithms like K-Nearest Neighbors (KNN), Logistic Regression, Random Forest or SVM in functioning more efficiently. Once scaled, the fitted scaler was saved as a .pkl file (scaler.pkl) employing the joblib library to help maintain consistency in model deployment or inference by using the same transformation on new, unseen data

Feature scaling

[57]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
import joblib

joblib.dump(scaler, 'scaler.pkl')
```

**5. Model Training with Random Forest:** In this project, the Random Forest Classifier was chosen as the base machine learning model for PDF file classification as benign or malicious. Random Forest is a robust ensemble learning algorithm with decision trees as its base and is well-suited for classification problems involving complicated datasets with mixed feature types.

**Working**-Random Forest creates many decision trees at training time. Each tree is trained on a random subset of data and features (bootstrapped sampling (BS)), which adds diversity and lowers the risk of overfitting. For prediction:

- Each tree makes a "vote" for the class label.
- The final prediction is obtained by taking the majority vote over all trees.

This approach enhances accuracy and generalization over single decision trees

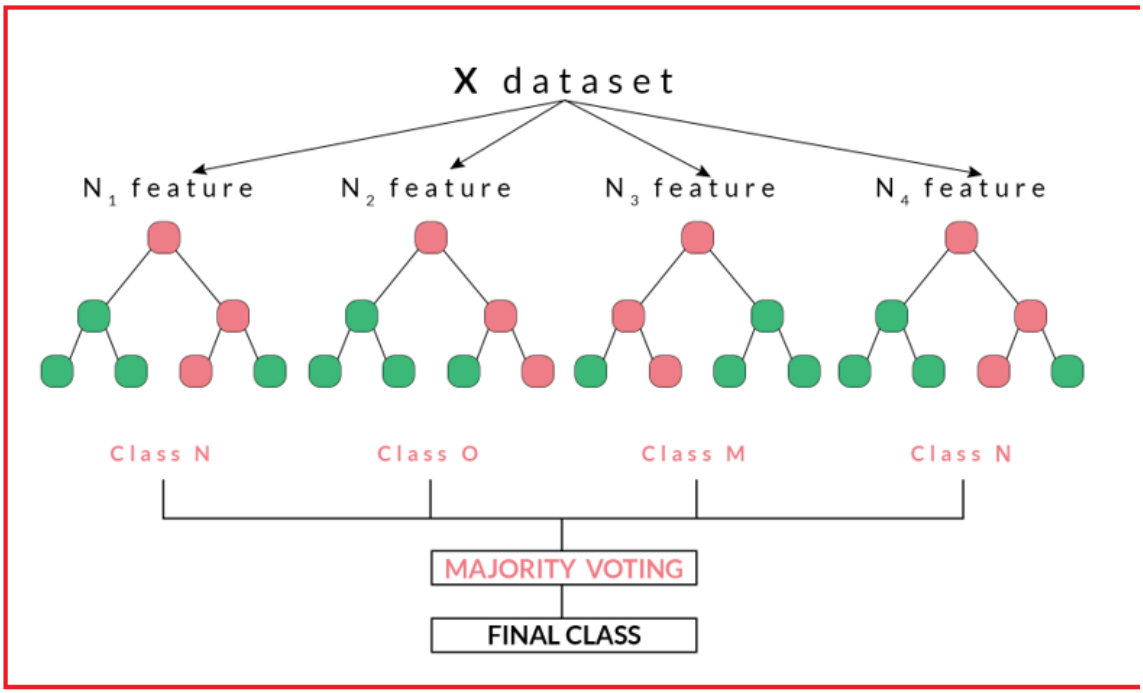



**Fig.Random Forest Classifier**

**Benefits of Random Forest-**

- **Handles high-dimensional data well**, which is helpful when there are a lot of engineered features in the dataset.
- **Robust to overfitting**, particularly when the number of trees is high enough.
- **Able to model non-linear relationships** and feature interactions without explicitly specifying.
- **Offers feature importance**, aiding to know which features are most useful for classification.

Train RandomForestClassifier on LDA data

```
[60]: clf_lda = RandomForestClassifier(n_estimators=200, max_depth=20, min_samples_split=5,
                                      min_samples_leaf=2, random_state=42, n_jobs=-1)
      clf_lda.fit(X_train_lda, y_train_lda)
      import joblib
      joblib.dump(clf_lda, "final_model.pkl")
```

**6. Evaluation:** Testing the trained model is an important step to establish its effectiveness, reliability, and fitness for real-world use. In this project, a range of evaluation metrics and visualization tools were used to evaluate the performance of the **Random Forest classifier** in identifying malicious PDF files. These comprised the

**confusion matrix**, **classification report**, **ROC curve**, and **AUC score**. The use of quantitative measures and visualizations guarantees a comprehensive analysis of the model's predictive performance.

- **Confusion Matrix:** The confusion matrix is a 2x2 table that is utilized to measure the performance of a classification model. It is used to compare the predicted class labels with the actual class labels. The confusion matrix is utilized to identify not just the overall accuracy but also the kinds of errors the model is committing. The structure of the confusion matrix is as follows:

| | **Predicted: Malicious** | **Predicted: Benign** |
|---|---|---|
| **Actual: Malicious** | True Positives (TP) | False Negatives (FN) |
| **Actual: Benign** | False Positives (FP) | True Negative (TN) |

  - **True Positives (TP):** Count of malicious PDF files accurately labeled as malicious.
  - **True Negatives (TN):** Count of benign PDF files accurately labeled as benign.
  - **False Positives (FP):** Count of benign PDF files mislabeled as malicious.
  - **False Negatives (FN):** Count of malicious PDF files mislabeled as benign.

  **Conclusions from Confusion Matrix**:

  - A high TP and TN value reveals good model precision.
  - A high FP rate would lead to tagging harmless files, which may compromise usability.
  - A high FN rate is more critical in malware detection since malicious files would not be detected.

Confusion Matrix

[62]:
```
cm_lda = confusion_matrix(y_test_lda, y_pred_lda)
sns.heatmap(cm_lda, annot=True, fmt="d", cmap="Purples", xticklabels=["Benign", "Malicious"], yticklabels=["Benign", "Malicious"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("LDA Confusion Matrix")
plt.show()
```

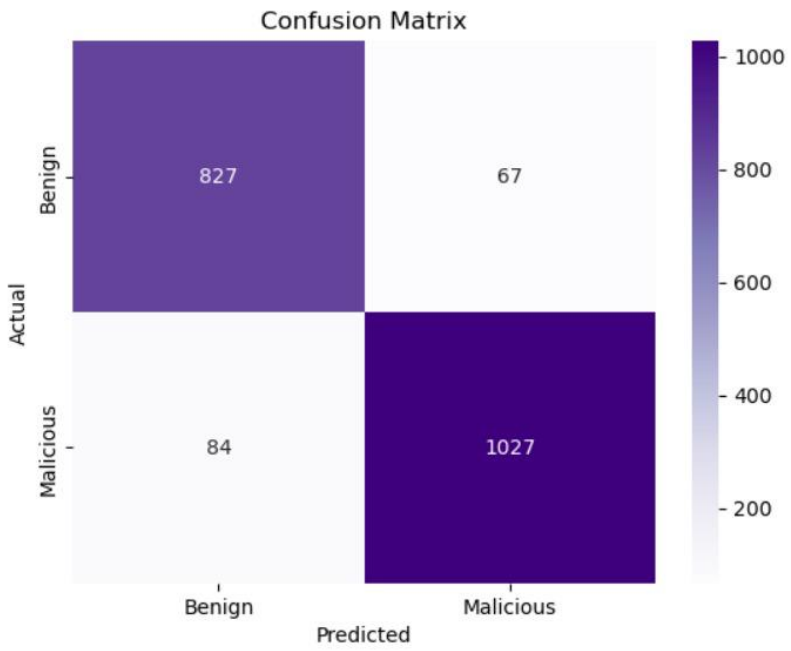



**Fig.Confusion Matrix of The Proposed Model**

- **Classification Report:** The classification report gives a breakdown of several performance measures per class (malicious and benign):
    - **Precision:** Precision shows how many of the files that the model predicted to be malicious actually are malicious. High precision implies few false positives.

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP})$$

    - **Recall:** Recall indicates how many of the actual malicious files the model correctly caught. High recall implies fewer false negatives.

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN})$$

    - **F1-Score:** F1-Score is the harmonic mean of precision and recall

$$\text{F1} - \text{Score} = 2((\text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall}))$$

    - **Accuracy:** Accuracy is the total ratio of instances that are classified correctly.

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

**Conclusion From Classification Report:**

- High recall and precision values reflect that the model is both sensitive and accurate towards detecting malicious files.
- F1-score is particularly crucial in imbalanced datasets where accuracy can be misleading alone.

Evaluating the Model

```
[61]: y_pred_lda = clf_lda.predict(X_test_lda)
y_proba_lda = clf_lda.predict_proba(X_test_lda)[:, 1]
print("\nClassification Report:\n", classification_report(y_test_lda, y_pred_lda))
print("ROC AUC Score:", roc_auc_score(y_test_lda, y_proba_lda))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.91      0.93      0.92       894
         1.0       0.94      0.92      0.93      1111

    accuracy                           0.92      2005
   macro avg       0.92      0.92      0.92      2005
weighted avg       0.92      0.92      0.92      2005

ROC AUC Score: 0.9619082713640491
```

**Fig.Classification Report of The Proposed Model**

- **ROC Curve and AUC Score:** The Receiver Operating Characteristic (ROC) curve is a plot used to represent the diagnostic capability of a binary classifier system. It graphically depicts:
  - True Positive along the Y-axis.
  - False Positive Rate along the X-axis

  The **Area Under the Curve (AUC)** measures the model's overall capability to discriminate between the two classes:

  - An AUC of 1.0 reflects perfect classification.
  - An AUC of 0.5 indicates no discriminative power (random guessing).

ROC Curve

```
[63]: fpr_lda, tpr_lda, _ = roc_curve(y_test_lda, y_proba_lda)
plt.figure(figsize=(7, 5))
plt.plot(fpr_lda, tpr_lda, label=f"LDA ROC Curve (AUC = {roc_auc_score(y_test_lda, y_proba_lda):.4f})", color='darkorange')
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("LDA - Receiver Operating Characteristic (ROC) Curve")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

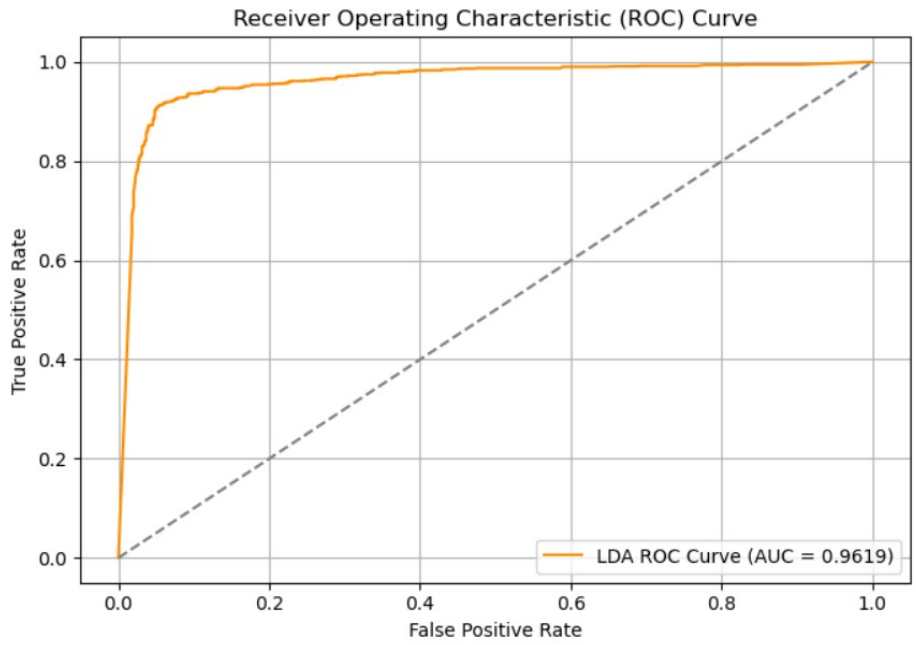


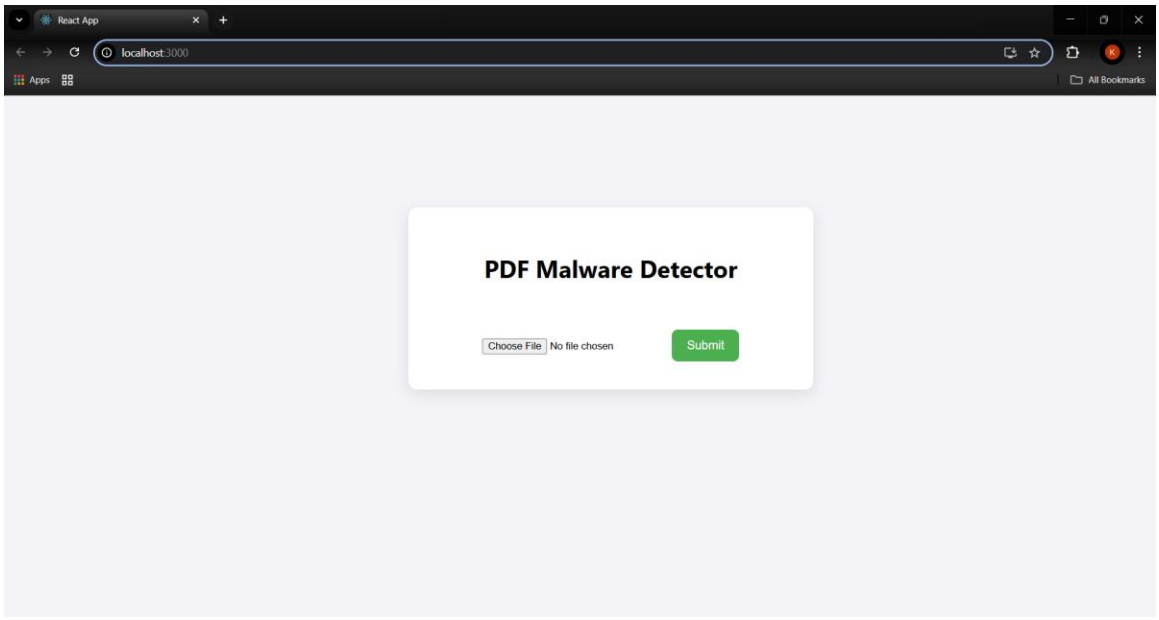

**Fig. ROC Curve of The Proposed Model**

The results of the evaluation verified that the **Random Forest Classifier** shows excellent performance in identifying malicious PDF files. With high precision, recall, accuracy, and a high AUC score, the model is highly appropriate for real-world deployment in security-critical environments.

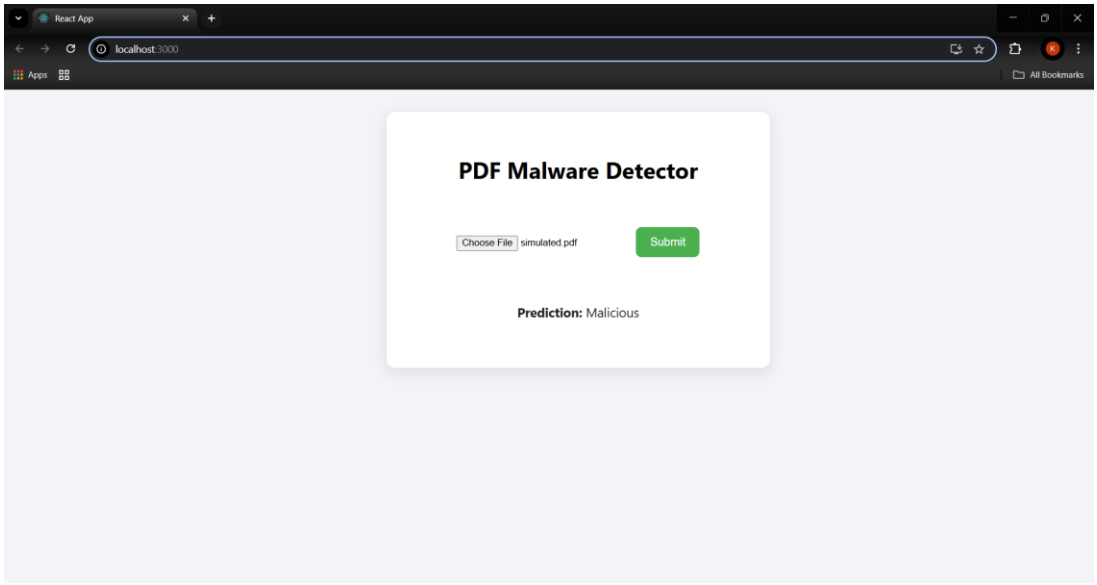
## 7. User Interface

Web-based user interface for PDF malware detection. The frontend provides a way to upload a PDF and obtain real-time classification results (malicious or not) based on the trained machine learning model**.**

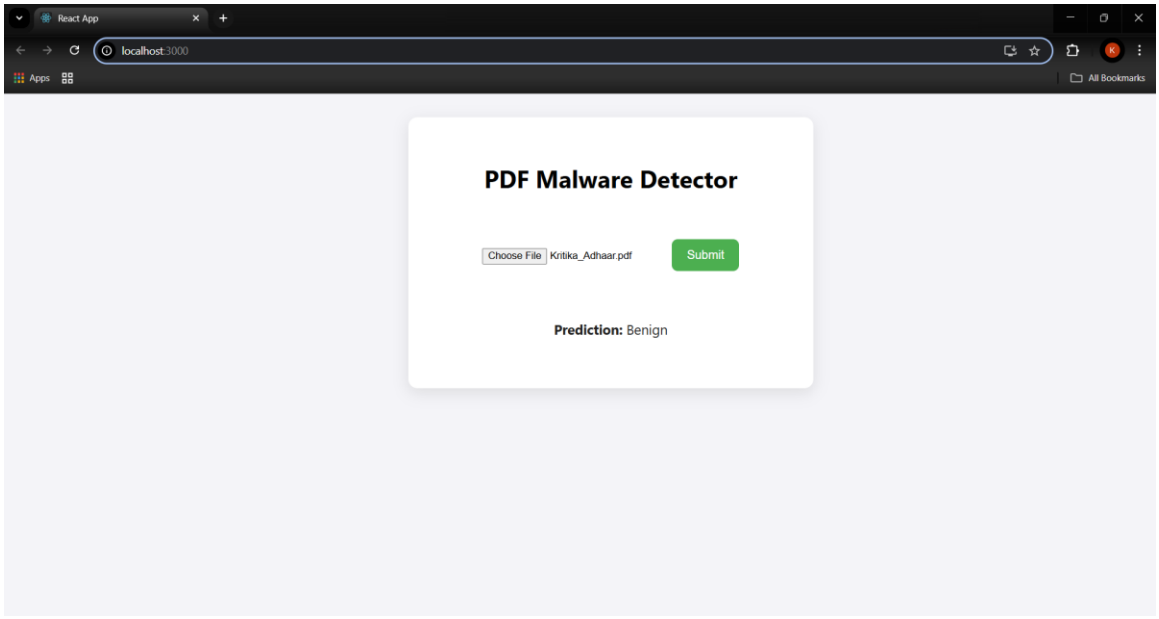
Frontend made in react and Backend integrated using Flash.




## Shows the detection of Malacious PDF :



**Shows the detection of Benign PDF :**




## 8. Conclusion:

This research is able to successfully prove the effectiveness and feasibility of machine learning methods for PDF-based malware detection, an emerging challenge in modern cybersecurity. By implementing a well-defined and well-organized machine learning pipeline that includes data preprocessing, feature engineering, dimensionality reduction, model training, and evaluation, the project was able to develop a stable classification system that can differentiate between benign and malicious PDF files.

## 9.References:

[1] https://www.unb.ca/cic/datasets/pdfmal-2022.html

[2] Tukey, J. W. (1977). Exploratory Data Analysis. Addison-Wesley.

[3] H. He and E. A. Garcia, "Learning from Imbalanced Data," in IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 9, pp. 1263-1284, Sept. 2009, doi: 10.1109/TKDE.2008.239.

[4] Chris Jarabek, David Barrera, and John Aycock. 2012. ThinAV: truly lightweight mobile cloud-based anti-malware. In Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12). Association for Computing Machinery, New York, NY, USA, 209–218. https://doi.org/10.1145/2420950.2420983