Adversarial Combat

A Deep Dive into Machine and Deep Learning Models Against Adversarial Attacks for Android Malware Detection

Fall 2023 DTSC 870
Masters Project

Team Members:
Selina Narain, Neelam Boywah, Zoya Haq

Advisor: Dr. Wenjia Li

Meet The Team

Selina Narain

- Research:
  - Random Forest, KNN Adversarial Attacks, Defense Mechanism
- Flowchart
- Models, Testing, Visualizations:
  - Random Forest Model, KNN, Adversarial Attack, Defense Mechanism

Neelam Boywah

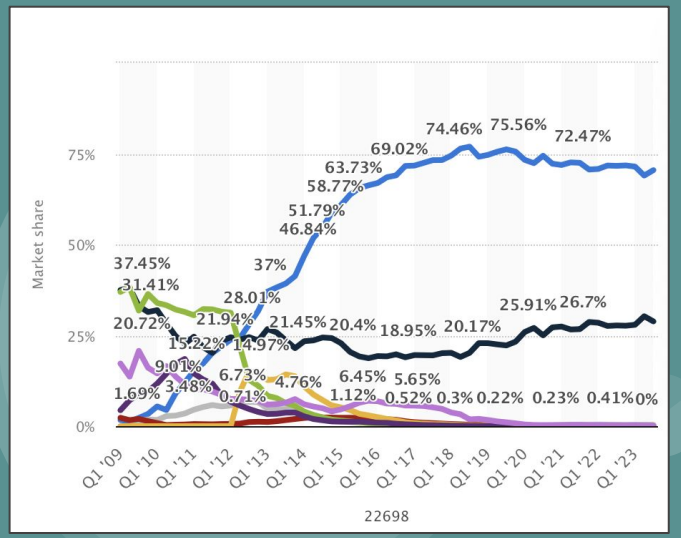
- Research:
  - Dense Neural Network, Support Vector Machine, Adversarial Attacks, Defense Mechanism
- Models, Testing, Visualizations:
  - Dense Neural Network, Support Vector Machine, Adversarial Attack, Defense Mechanism
- Power BI Dashboard

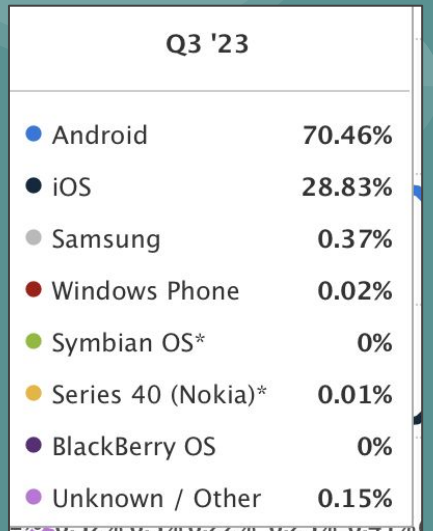
Zoya Haq

- Research:
  - Naive Bayes, Logistic Regression, Adversarial Attacks, Defense Mechanism
- Models, Testing, Visualizations:
  - Naive Bayes, Logistic Regression, Adversarial Attack, Defense Mechanism
- PowerBI Dashboard

Background

- Popularity of Android applications have increased over the years, along with this, the attempts to exploit the users of Google Play also increased.
- There are malicious apps that are concealed as actual software which has penetrated the app store.
- These signs of malware are a threat to user privacy, data security and device integrity,
- The Android OS has a dynamic and open nature which makes it more vulnerable towards malware attacks.
- Understanding this is crucial as we explore the Android malware detection world to identify areas of opportunity against the evolving digital threats.

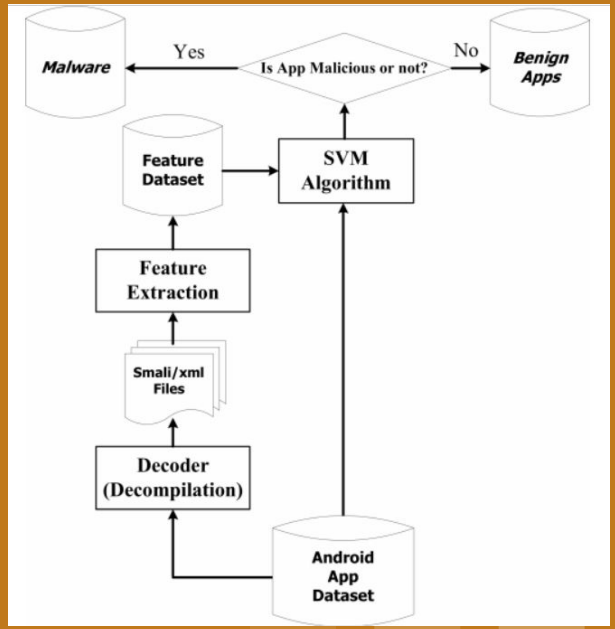




Literature Review

Detecting Malware for Android Platform: An SVM based Approach

- Support Vector Machine (SVM) algorithm incorporating risky permissions combinations and vulnerable API calls as features.
- Calculates similarity scores between the malware and benign apps in relation to suspicious API calls and using the risky permission requests as additional features when training SVM.
- Tools such as Drebin, DroidMat, and DroidAPIMiner leverage ML algorithms like SVM and KNN which display accuracy results up to 86%.
- Static program analysis has to do with decompilation, feature extraction and classification.
- Dynamic Analysis uses monitoring tools.



Literature Review

EAGLE: Evasion Attacks Guided by Local Explanations against Android Malware Classification

- EAGLE is an attack strategy
- Contains specific features to evade Android malware classifiers
- Does not require transplant gadgets unlike other aforementioned approaches
- Operates by adjusting feature values that are identified through local explanations.
- EAGLE attacks use operations to increase or decrease important feature values displaying model-agnostic behavior and achieving high misclassification rates.
- EAGLE attacks are applicable to many Android malware classifiers with arbitrary count features.
- It isn't limited to binary features and still achieves successful misclassification rates and removes the need for a large load of benign Android apps for feature extraction.

Literature Review

Adversarial Attacks and Defenses in Deep Learning

Attacks:

- Fast gradient sign method
  - Generates adversarial samples, One-step attack that updates along the direction (i.e., sign) of the gradient of the adversarial loss
- BIM and PGD
  - Iterative optimizer for multiple iterations. PGD projects the adversarial samples learned from each iteration
- Momentum iterative attack
  - Builds an ensemble of models to attack a model in the black-box/gray-box settings

Defenses

- Adversarial training
  - Intuitive defense method against adversarial samples. Attempts to improve the robustness of a model by training it with adversarial samples.
- Ensemble adversarial training
  - Incorporates adversarial samples transferred from multiple pre-trained ensemble models

Android Malware Detection Overview

Research Statement:

Our master's project in malware detection involved the application of machine learning models to analyze datasets, assisting in the development of robust detection mechanisms. In addition, we utilized an adversarial attack and corresponding defense strategy to enhance insights into cybersecurity challenges in the world of Android malware detection.

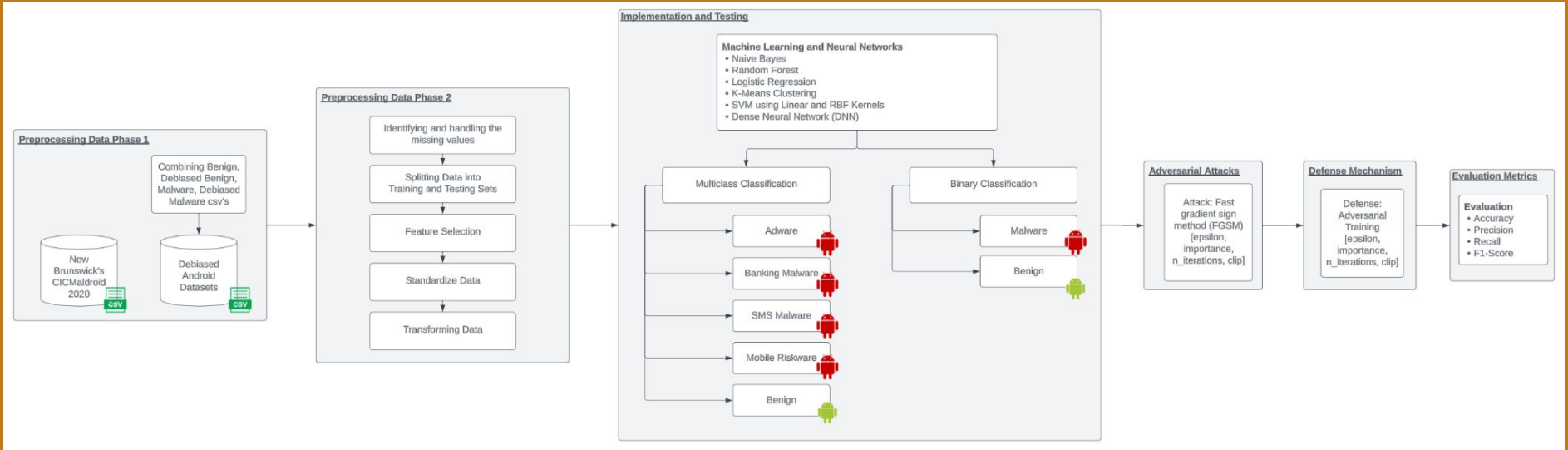
Machine Learning Models:

- Naive Bayes
- **Random Forest**
- Logistic Regression
- K-Nearest Neighbors
- SVM using Linear and RBF Kernels

Deep Learning Model

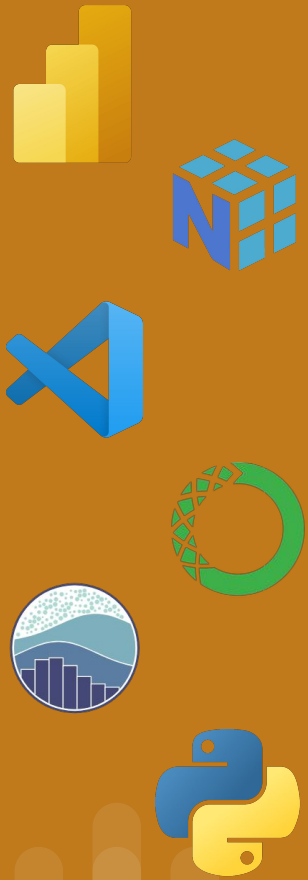
- Dense Neural Network (DNN)

Overall Workflow



Key Technologies

- Programming Language:
  - Python
- Environment Control & Package Management:
  - Anaconda, Python Virtual Environment
- IDE:
  - Visual Studio Code & Google Colaboratory
- Packages & Libraries:
  - Sci-kit Learn, NumPy, Pandas, Tensorflow, Seaborn, Matplotlib
- Datasets:
  - University of New Brunswick: CICMalDroid 2020, DADA Dataset
- Visualizations Report:
  - PowerBI Analytics Dashboard



Demo

University of New Brunswick: Canadian Institute for Cybersecurity - CICMalDroid 2020

Android Malware Dataset

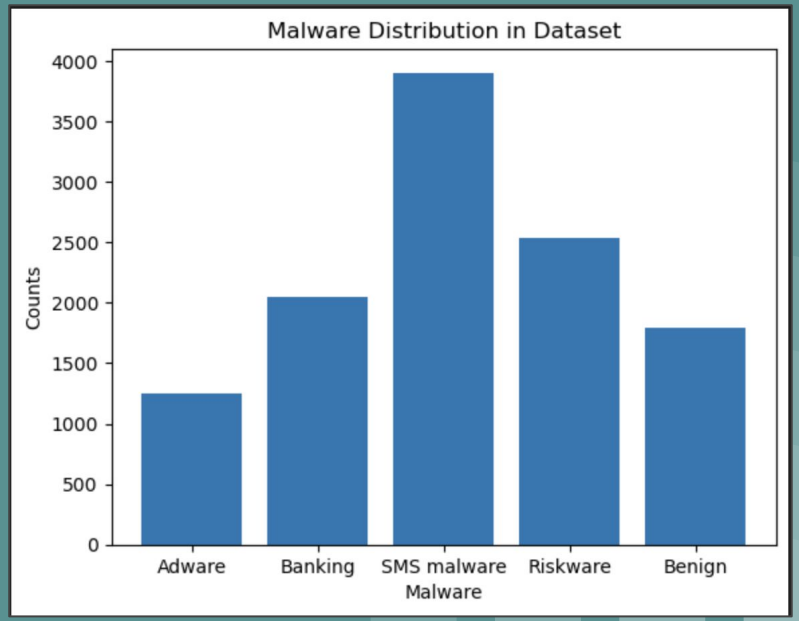
Data Sources: VirusTotal service, Contagio security blog, AMD, MalDozer

Categories: Adware: 1,253, Banking: 2,100, SMS malware: 3,904, Riskware: 2,546, Benign: 1,795, Total: 11,598

50,621 extracted features for 11,598 APK files

Static Data:

- Intent Actions
- Permissions
- Method tags
- Sensitive APIs
- Services
- Packages
- Receivers



Debiased Android Datasets: Dada Dataset

“Debiasing Android Malware Datasets: How can I trust your results if your dataset is biased?”

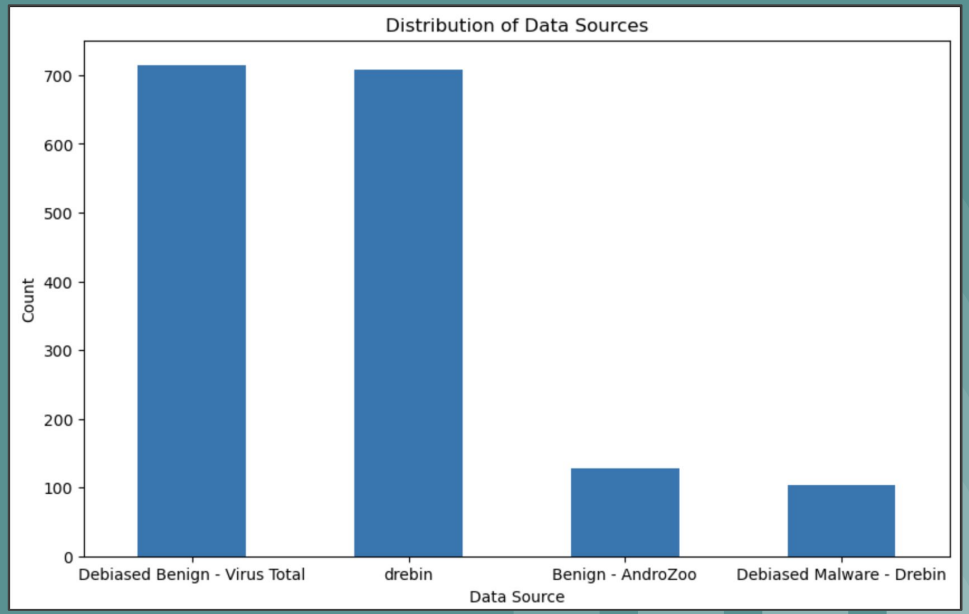
Data Sources: Drebin, AndroZoo, VirusShare

DR-AG_deb: a debiased dataset of Drebin (DR) mixed with benign samples from AndroZoo (AG) and VirusTotal

215 Features, 1655 samples

Static Data:

- SHA-256
- APK Size
- Permissions:
  - READ_PHONE_STATE
  - READ_CONTACTS
  - READ_SMS
  - ACCESS_NETWORK_STATE
  - SEND_SMS
  - Etc.
- Dataset - Data Source
- Malware



Université du Luxembourg: Preprocessing AndroZoo Data

- Collection of 23,897,277 Malicious Android Applications' APKs
- Requires AndroZoo API Access

APK Files - Compressed Files

- Identified by SHA-256 hash values
- File Contents:
  - Dalvik Executable (.dex) file
  - Cryptographic certificate(s)
  - Assets (images, audio files, libraries, etc.)
  - **Manifest file**

Process:

- Created script to obtain APK files from AndroZoo API
- Downloaded 1500 APK files
- Reverse Engineered APK tools using APKtool
  - Decompile the APK file to obtain contents
  - Issues: Corrupted APKs, Extraction of uses-permissions from Manifest.xml files, resource allocation for processing 1500 files

```python
import os
import requests
#this one
def download_androzoo_apk(api_key, sha256, output_directory):
    androzoo_url = f"https://androzoo.uni.lu/api/download?apikey={api_key}&sha256={sha256}"

    try:
        # Download the APK from AndroZoo
        response = requests.get(androzoo_url, stream=True)

        # Check if the request was successful (status code 200)
        if response.status_code == 200:
            # Save the APK file
            apk_path = os.path.join(output_directory, f"{sha256}.apk")
            with open(apk_path, 'wb') as apk_file:
                for chunk in response.iter_content(chunk_size=128):
                    apk_file.write(chunk)

            return apk_path

        else:
            print(f"Error downloading APK with SHA256: {sha256}, Status Code: {response.status_code}")

    except Exception as e:
        print(f"Error downloading APK with SHA256: {sha256}, Error: {e}")

    return None

def main():
    api_key = '43cc9fb49a950a8400cbff9a844b96576ad74c8c4f961288b80f7740375a0d70'  # AndroZoo API key
    output_directory = 'downloaded_apks'

    # Create the output directory if it doesn't exist
    os.makedirs(output_directory, exist_ok=True)

    # List of SHA256 values of APKs from AndroZoo
    apk_sha256_list = ['AAB1E99719E160242FFF3680AD5A4622728B64ABE16F8B9CBA15E6CCFFDEE770B',
                       'A89DCF9F2CEA17F4EEBA4CB9404A5C55178408B39032104DDF7EA5BEC86F1344F',
                       '89E50751C74D7313B1C89580883FDBD84EFB1C09B33777FBF680BE1152127CC8']  # SHA256 values

    for sha256 in apk_sha256_list:
        # Download APK
        apk_path = download_androzoo_apk(api_key, sha256, output_directory)

        if apk_path:
            print(f"APK with SHA256 {sha256} downloaded to {apk_path}.")

if __name__ == '__main__':
    main()
```

```
APK with SHA256 0004091FFB7AFD0B57B9AF50322E42ABEF5C454244ECC63BDB32760BF3B6DE31 downloaded to downloaded_apks/0004091FFB7AFD0B57B9AF50322E42ABEF5C454244ECC63BDB32760BF3B6DE31.apk.
APK with SHA256 00040A1493C4C5087E934B9CA044F90E9F4B2408BAF535 3BB4D16CA180A6A93 downloaded to downloaded_apks/00040A1493C4C5087E934B9CA044F90E9F4B2408BAF5353BB4D16CA180A6A93.apk.
APK with SHA256 00040A370D1532E800A18CCE2972FE550B028AC5053E396 4D3AABF93C4208ECB downloaded to downloaded_apks/00040A370D1532E800A18CCE2972FE550B028AC5053E3964D3AABF93C4208ECB.apk.
APK with SHA256 00040E532A7805BA83E27DAE884F262AC6C9D116E77DF96960 4B6DE3FB08ADD downloaded to downloaded_apks/00040E532A7805BA83E27DAE884F262AC6C9D116E77DF96960 4B6DE3FB08ADD.apk.
APK with SHA256 00040C5BD26D2B912779EF5E330B427FE5F6AC500A2F2278E7 4CE8D1824968C2 downloaded to downloaded_apks/00040C5BD26D2B912779EF5E330B427FE5F6AC500A2F2278E74CE8D1824968C2.apk.
```

```
I: Using Apktool 2.9.0 on 000039C610124B0E58FC642604A5A53972B0414D9F78CF25A01D45D3438DBC77.apk
I: Loading resource table...
I: Decoding file-resources...
I: Loading resource table from file: /Users/selinanarain/Library/apktool/framework/1.apk
I: Decoding values */* XMLs...
I: Decoding AndroidManifest.xml with resources...
I: Regular manifest package...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
I: Using Apktool 2.9.0 on 00022D057443AC60334E3DD676F30A3B4C2837DE54C09A4CB85A28D950C4E2A9.apk
I: Loading resource table...
I: Decoding file-resources...
I: Loading resource table from file: /Users/selinanarain/Library/apktool/framework/1.apk
I: Decoding values */* XMLs...
I: Decoding AndroidManifest.xml with resources...
I: Regular manifest package...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
I: Copying META-INF/services directory
```

Step 1: Preprocessing Data - CICMalDroid 2020

Feature Selection

Standardizing Data

Transforming Data

```
# Check for missing values in dataset
print(df.isnull().sum())
```

✓ 0.0s

```
ACCESS_PERSONAL_INFO___     0
ALTER_PHONE_STATE___        0
ANTI_DEBUG_____             0
CREATE_FOLDER_____          0
CREATE_PROCESS`_____        0
                           ..
watchRotation               0
windowGainedFocus           0
write                       0
writev                      0
Class                       0
Length: 471, dtype: int64
```

```
# Iterate and display a list of the df columns
for col in df.columns:
    print(col)
```

✓ 0.0s

```
ACCESS_PERSONAL_INFO___
ALTER_PHONE_STATE___
ANTI_DEBUG_____
CREATE_FOLDER_____
CREATE_PROCESS`_____
CREATE_THREAD_____
DEVICE_ACCESS_____
EXECUTE_____
FS_ACCESS____
FS_ACCESS()____
FS_ACCESS(CREATE)____
FS_ACCESS(CREATE__APPEND)__
FS_ACCESS(CREATE__READ)__
FS_ACCESS(CREATE__READ__WRITE)
FS_ACCESS(CREATE__WRITE)__
FS_ACCESS(CREATE__WRITE__APPEND)
FS_ACCESS(READ)____
FS_ACCESS(READ__WRITE)__
FS_ACCESS(WRITE)____
FS_PIPE_ACCESS___
FS_PIPE_ACCESS()___
FS_PIPE_ACCESS(READ)___
FS_PIPE_ACCESS(READ__)_
FS_PIPE_ACCESS(READ__WRITE)_
FS_PIPE_ACCESS(WRITE)___
```

```
#Features
X = df.drop(columns=['Class'])
#Target
y = df['Class']

#Splitting data into training and test
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size=0.2, random_state=42)

#ANOVA-based feature selection
selected_features = 120
k_best_features = SelectKBest(score_func = f_classif, k = selected_features)
X_train_selected_features = k_best_features.fit_transform(X_train, y_train)
X_test_selected_features = k_best_features.transform(X_test)

#Indicies of selected features
selected_features_idx = k_best_features.get_support(indices=True)

#Converting selected features to a DataFrame
X_train = X_train.iloc[:, selected_features_idx]
X_test = X_test.iloc[:, selected_features_idx]
X_train.head()
```

✓ 0.2s

	CREATE_FOLDER_____	CREATE_PROCESS`_____	CREATE_THREAD_____	EXECUTE_____	FS_ACCESS____	FS_ACCESS()____
9652	2	0	10	0	14	0
11116	12	2	26	5	44	2
8165	0	0	9	0	6	1
6913	6	0	40	0	42	6
6682	7	0	45	0	52	2

5 rows × 120 columns

```
# Scale the data using Standard Scaler function from SKLearn
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

✓ 0.0s

Step 1: Preprocessing Data - DADA Dataset

Feature Selection

Standardizing Data

Transforming Data

```
# Check for missing values in dataset
print(df.isnull().sum())
```

✓ 0.0s

```
sha256                  0
APK size                0
Year                    0
Internet Permission     0
External storage        0
                       ..
loadclass               0
url_in_exec             0
mtk_su                  0
dataset                 0
malware                 0
Length: 222, dtype: int64
```

```
# Iterate and display a list of the df columns
for col in df.columns:
    print(col)
```

✓ 0.0s

```
sha256
APK size
Year
Internet Permission
External storage
Uses Play Services
Generates UUIDs
Vibrate phone
NFC
Bluetooth
Uses HTTP
Uses JSON
Specify User-Agent
apk_size
dex_date
year
minSdkVersion
targetSdkVersion
android.permission.READ_PHONE_STATE
android.permission.READ_CONTACTS
android.permission.READ_SMS
android.permission.CAMERA
android.permission.RECORD_AUDIO
android.permission.READ_EXTERNAL_STORAGE
android.permission.READ_HISTORY_BOOKMARKS
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.GET_TASKS
```

```
#Features
X = df.drop(columns = ['APK size', 'Year', 'year', 'dex_date', 'dataset', 'sha256', 'malware'])
#Target
y = df['malware']

#Split the data into training and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

X_train.head()
```

✓ 0.0s

	Internet Permission	External storage	Uses Play Services	Generates UUIDs	Vibrate phone	NFC	Bluetooth	Uses HTTP	Uses JSON	Specify User-Agent	...
1301	1	1	0	1	0	0	0	1	0	0	...
306	1	1	1	0	0	0	0	0	0	0	...
192	1	1	1	0	0	0	0	0	0	0	...
309	1	1	1	0	0	0	0	0	0	0	...
1168	1	0	0	0	0	0	0	0	1	1	...

5 rows × 215 columns

```
# Scale the data using Standard Scaler function from SKLearn
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

✓ 0.0s

Step 2: Developing ML/DL Models - Random Forest

CICMalDroid 2020

```
#Create Random Forest Model
random_forest_model = RandomForestClassifier(n_estimators=300, random_state=42)

#Fit and train classifier on the training data
random_forest_model.fit(X_train, y_train)

#Predict the labels using X_test
y_pred_rf = random_forest_model.predict(X_test)

#Calculate the metrics for the Random Forest Classifier
random_forest_accuracy = accuracy_score(y_test, y_pred_rf)
random_forest_precision = precision_score(y_test, y_pred_rf,average='weighted')
random_forest_recall = recall_score(y_test, y_pred_rf,average='weighted')
random_forest_f1 = f1_score(y_test, y_pred_rf,average='weighted')

#Display the metrics for the Random Forest Classifier
print("Random Forest Classifier Accuracy:  %.4f" % (random_forest_accuracy))
print("Random Forest Classifier Precision: %.4f" % (random_forest_precision))
print("Random Forest Classifier Recall: %.4f" % (random_forest_recall))
print("Random Forest Classifier F1-Score: %.4f" % (random_forest_f1))
print("Classification Report: " "\n", classification_report(y_test, y_pred_rf))
print("Confusion Matrix: ", "\n", confusion_matrix(y_test, y_pred_rf))

#Create and display heatmap
conf_matrix = confusion_matrix(y_pred_rf, y_test)
sns.heatmap(conf_matrix , cmap='Blues', fmt='', xticklabels=['Adware' ,'Banking' ,'SMS malware', 'Riskware','Benign'],
            yticklabels=['Adware' ,'Banking' ,'SMS malware', 'Riskware','Benign'], annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest Classifier')
plt.show()
```

✓ 3.5s

Accuracy Score: 0.9480

DADA Dataset

```
#Create Random Forest Model
random_forest_model = RandomForestClassifier(n_estimators = 300, random_state = 42)

#Fit and train classifier on the training data
random_forest_model.fit(X_train, y_train)

#Predict the labels using X_test
y_pred_rf = random_forest_model.predict(X_test)

#Calculate the metrics for the Random Forest Classifier
randomForestAccuracy = accuracy_score(y_test, y_pred_rf)
randomForestPrecision = precision_score(y_test, y_pred_rf,average = 'weighted')
randomForestRecall = recall_score(y_test, y_pred_rf,average = 'weighted')
randomForestF1 = f1_score(y_test, y_pred_rf,average = 'weighted')

#Display the metrics for the Random Forest Classifier
print("Random Forest Classifier Accuracy: %.4f" % randomForestAccuracy)
print("Random Forest Classifier Precision: %.4f" % randomForestPrecision)
print("Random Forest Classifier Recall: %.4f" % randomForestRecall)
print("Random Forest Classifier F1-Score: %.4f" % randomForestF1)
print("Classification Report: " "\n", classification_report(y_test, y_pred_rf))
print("Confusion Matrix: ", "\n", confusion_matrix(y_test, y_pred_rf))

#Create and display heatmap
conf_matrix = confusion_matrix(y_test, y_pred_rf, labels = classes)
sns.heatmap(conf_matrix , cmap = 'Blues', fmt = 'd', annot = True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest Classifier')
plt.show()
```

✓ 0.3s

Accuracy Score: 0.9637

Step 3: Testing & Evaluation (CICMalDroid 2020 Dataset)

```
Naive Bayes Classifier Accuracy: 0.5889
Naive Bayes Classifier Precision: 0.6804
Naive Bayes Classifier Recall: 0.5889
Naive Bayes Classifier F1-Score: 0.5486
Classification Report:
              precision    recall  f1-score   support

           1       0.31      0.58      0.40       251
           2       0.68      0.13      0.22       409
           3       0.59      0.98      0.74       781
           4       0.87      0.31      0.46       507
           5       0.87      0.65      0.74       358

    accuracy                           0.59      2306
   macro avg       0.66      0.53      0.51      2306
weighted avg       0.68      0.59      0.55      2306

Confusion Matrix:
 [[145   1 100   3   2]
 [ 73  53 262  10  11]
 [  8   4 769   0   0]
 [174  18 134 159  22]
 [ 74   2  40  10 232]]
```

Naive Bayes

```
Logistic Regression Accuracy: 0.8053
Logistic Regression Precision: 0.8072
Logistic Regression Recall: 0.8053
Logistic Regression F1-Score: 0.8007
Classification Report:
              precision    recall  f1-score   support

           1       0.74      0.57      0.64       251
           2       0.72      0.75      0.74       409
           3       0.81      0.97      0.88       781
           4       0.85      0.71      0.77       507
           5       0.90      0.80      0.85       358

    accuracy                           0.81      2306
   macro avg       0.80      0.76      0.78      2306
weighted avg       0.81      0.81      0.80      2306

Confusion Matrix:
 [[143  48  33  23   4]
 [ 19 308  59  15   8]
 [  1  16 757   7   0]
 [ 19  40  66 361  21]
 [ 11  14  25  20 288]]
```

Logistic Regression

```
Best K value: 3
K-Nearest Neighbors Classifier Accuracy: 0.9055
K-Nearest Neighbors Classifier Precision: 0.9092
K-Nearest Neighbors Classifier Recall: 0.9055
K-Nearest Neighbors Classifier F1-Score: 0.9053
Classification Report:
              precision    recall  f1-score   support

           1       0.72      0.87      0.79       251
           2       0.90      0.89      0.90       409
           3       0.96      0.99      0.98       781
           4       0.93      0.92      0.92       507
           5       0.91      0.75      0.82       358

    accuracy                           0.91      2306
   macro avg       0.88      0.88      0.88      2306
weighted avg       0.91      0.91      0.91      2306

Confusion Matrix:
 [[219   7   6   8  11]
 [ 24 366   9   6   4]
 [  0   7 771   2   1]
 [ 20   7   4 465  11]
 [ 40  21  10  20 267]]
```

KNN

```
Random Forest Classifier Accuracy:  0.9480
Random Forest Classifier Precision: 0.9484
Random Forest Classifier Recall: 0.9480
Random Forest Classifier F1-Score: 0.9479
Classification Report:
              precision    recall  f1-score   support

           1       0.88      0.93      0.90       251
           2       0.97      0.92      0.94       409
           3       0.97      0.99      0.98       781
           4       0.95      0.92      0.93       507
           5       0.93      0.93      0.93       358

    accuracy                           0.95      2306
   macro avg       0.94      0.94      0.94      2306
weighted avg       0.95      0.95      0.95      2306

Confusion Matrix:
 [[233   2   4   8   4]
 [  6 377   6  10  10]
 [  0   3 775   1   2]
 [ 22   2   4 468  11]
 [  4   5   8   8 333]]
```

Random Forest

```
SVM RBF Classifier Accuracy: 0.7979
SVM RBF Classifier Precision: 0.8000
SVM RBF Classifier Recall: 0.7979
SVM RBF Classifier F1-Score: 0.7942
Classification Report:
              precision    recall  f1-score   support

           1       0.71      0.59      0.65       251
           2       0.70      0.78      0.74       409
           3       0.82      0.95      0.88       781
           4       0.85      0.68      0.76       507
           5       0.87      0.79      0.82       358

    accuracy                           0.80      2306
   macro avg       0.79      0.76      0.77      2306
weighted avg       0.80      0.80      0.79      2306

Confusion Matrix:
 [[149  28  45  20   9]
 [ 17 317  46  17  12]
 [  2  25 745   9   0]
 [ 25  65  47 347  23]
 [ 18  17  25  16 282]]
```

SVM - ‘rbf’

```
SVM Linear Classifier Accuracy: 0.8317
SVM Linear Classifier Precision: 0.8323
SVM Linear Classifier Recall: 0.8317
SVM Linear Classifier F1-Score: 0.8285
Classification Report:
              precision    recall  f1-score   support

           1       0.73      0.61      0.66       251
           2       0.73      0.83      0.78       409
           3       0.88      0.98      0.92       781
           4       0.85      0.73      0.79       507
           5       0.90      0.82      0.86       358

    accuracy                           0.83      2306
   macro avg       0.82      0.79      0.80      2306
weighted avg       0.83      0.83      0.83      2306

Confusion Matrix:
 [[152  41  23  29   6]
 [ 21 339  28  15   6]
 [  1  16 763   1   0]
 [ 17  54  43 372  21]
 [ 17  15  15  19 292]]
```

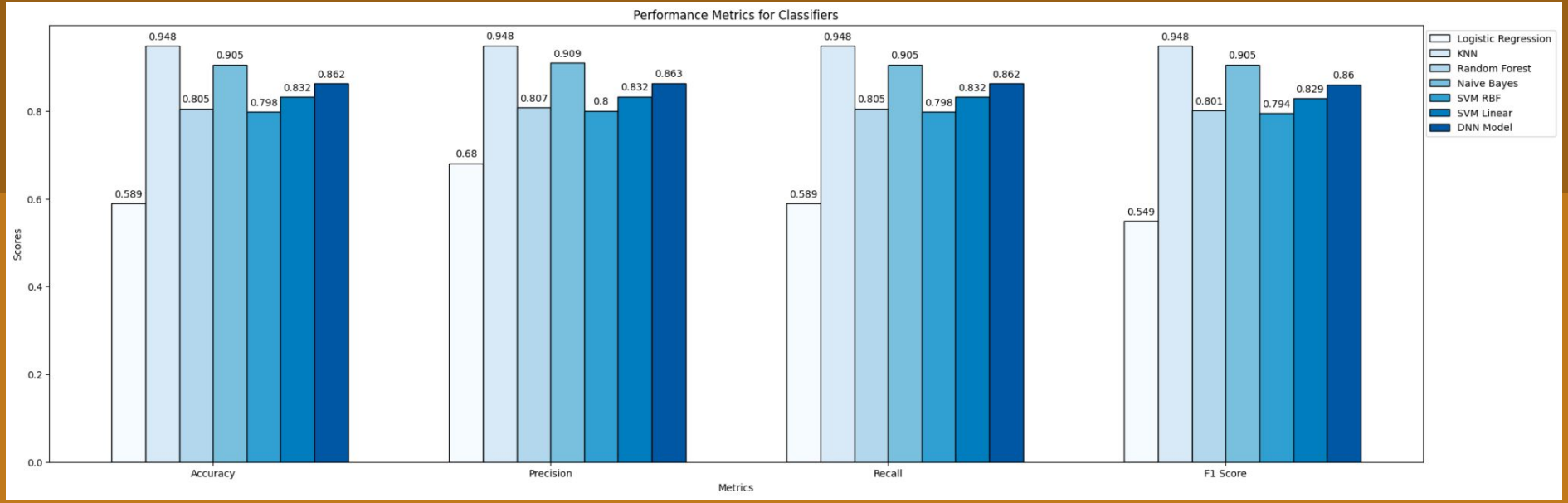
SVM - ‘linear’

```
Accuracy: 0.8625
DNN Precision: 0.8629
DNN Recall: 0.8625
DNN F1-Score: 0.8602
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.69      0.76       262
           1       0.79      0.84      0.81       424
           2       0.89      0.98      0.94       786
           3       0.90      0.80      0.84       500
           4       0.85      0.83      0.84       348

    accuracy                           0.86      2320
   macro avg       0.85      0.83      0.84      2320
weighted avg       0.86      0.86      0.86      2320

Confusion Matrix:
 [[181  26  22  12  21]
 [ 15 357  27  13  12]
 [  0   9 774   2   1]
 [ 11  49  23 399  18]
 [  6  13  20  19 290]]
```

Dense Neural Network

Visualizations of ML/DL Models

Models Performance - CICMaldroid 2020 Dataset



Step 3: Testing & Evaluation (DADA Dataset)

```
Naive Bayes Classifier Accuracy: 0.6435
Naive Bayes Classifier Precision: 0.7547
Naive Bayes Classifier Recall: 0.6435
Naive Bayes Classifier F1-Score: 0.5967
Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.98      0.74       168
           1       0.92      0.30      0.45       163

    accuracy                           0.64       331
   macro avg       0.76      0.64      0.59       331
weighted avg       0.75      0.64      0.60       331

Confusion Matrix:
 [[164   4]
 [114  49]]
```

Naive Bayes

```
Logistic Regression Accuracy: 0.9094
Logistic Regression Precision: 0.9094
Logistic Regression Recall: 0.9094
Logistic Regression F1-Score: 0.9094
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.93      0.91       168
           1       0.92      0.89      0.91       163

    accuracy                           0.91       331
   macro avg       0.91      0.91      0.91       331
weighted avg       0.91      0.91      0.91       331

Confusion Matrix:
 [[156  12]
 [ 18 145]]
```

Logistic Regression

```
Best K value: 5
K-Nearest Neighbors Classifier Accuracy: 0.9245
K-Nearest Neighbors Classifier Precision: 0.9279
K-Nearest Neighbors Classifier Recall: 0.9245
K-Nearest Neighbors Classifier F1-Score: 0.9243
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       168
           1       0.97      0.88      0.92       163

    accuracy                           0.92       331
   macro avg       0.93      0.92      0.92       331
weighted avg       0.93      0.92      0.92       331

Confusion Matrix:
 [[163   5]
 [ 20 143]]
```

KNN

```
Random Forest Classifier Accuracy: 0.9637
Random Forest Classifier Precision: 0.9640
Random Forest Classifier Recall: 0.9637
Random Forest Classifier F1-Score: 0.9637
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       168
           1       0.97      0.95      0.96       163

    accuracy                           0.96       331
   macro avg       0.96      0.96      0.96       331
weighted avg       0.96      0.96      0.96       331

Confusion Matrix:
 [[164   4]
 [  8 155]]
```

Random Forest

```
SVM RBF Classifier Accuracy: 0.9366
SVM RBF Classifier Precision: 0.9369
SVM RBF Classifier Recall: 0.9366
SVM RBF Classifier F1-Score: 0.9365
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.95      0.94       168
           1       0.95      0.92      0.93       163

    accuracy                           0.94       331
   macro avg       0.94      0.94      0.94       331
weighted avg       0.94      0.94      0.94       331

Confusion Matrix:
 [[160   8]
 [ 13 150]]
```

SVM - ‘rbf’

```
SVM Linear Classifier Accuracy: 0.9003
SVM Linear Classifier Precision: 0.9006
SVM Linear Classifier Recall: 0.9003
SVM Linear Classifier F1-Score: 0.9003
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       168
           1       0.91      0.88      0.90       163

    accuracy                           0.90       331
   macro avg       0.90      0.90      0.90       331
weighted avg       0.90      0.90      0.90       331

Confusion Matrix:
 [[154  14]
 [ 19 144]]
```

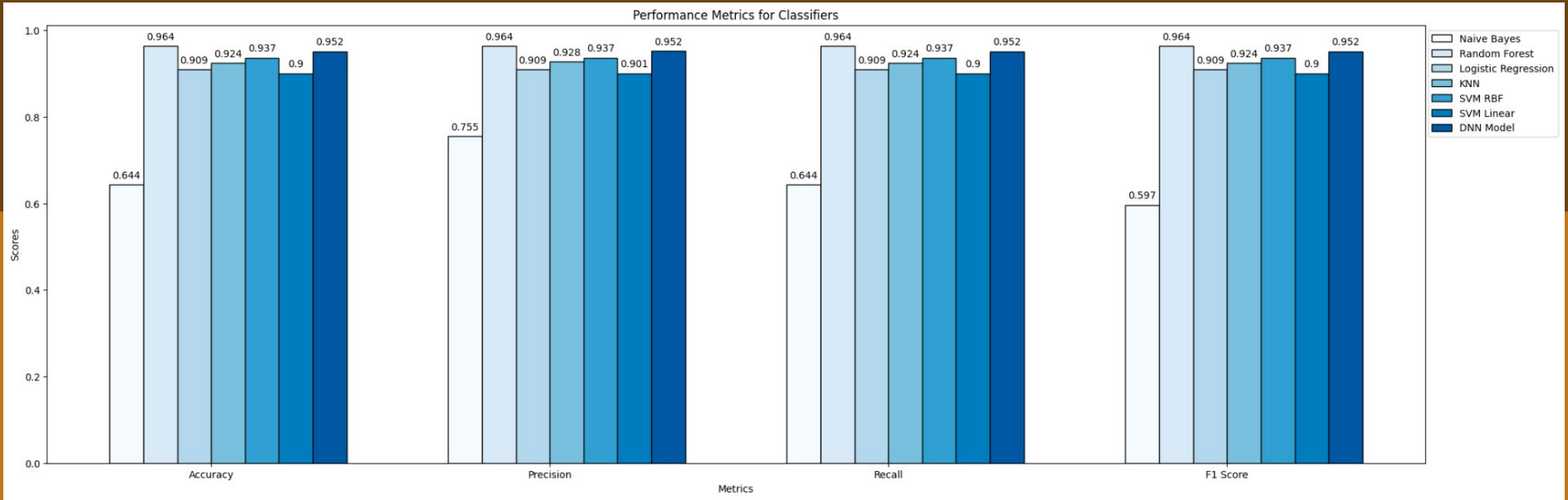
SVM - ‘linear’

```
Accuracy: 0.9517
DNN Precision: 0.9519
DNN Recall: 0.9517
DNN F1-Score: 0.9516
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       168
           1       0.96      0.94      0.95       163

    accuracy                           0.95       331
   macro avg       0.95      0.95      0.95       331
weighted avg       0.95      0.95      0.95       331

Confusion Matrix:
 [[162   6]
 [ 10 153]]
```

Dense Neural Network

Visualizations of ML/DL Models

Models Performance - DADA Dataset



Adversarial Attack & Defense Mechanism

What are Adversarial Attacks?

- Class of attacks with the goal of deteriorating the performance of classifiers.

Fast Gradient Sign Method (FGSM)

- Type of adversarial attack that uses the computed gradients of the loss of a model on the input to maximize the overall loss.
- Computation: x_{adv} is the adversarial examples, x_{benign} is benign. Cost function $J(\theta, x_{benign}, y))$, θ represents the network parameters, ϵ is used to scale the noise, sign function applied to gradients of the loss with respect to the input to compute final perturbation.

$$x_{adv} = x_{benign} + \epsilon * \text{sign}(\nabla x_{benign} J(\theta, x_{benign}, y))$$

C. Pestana, N. Akhtar, W. Liu, D. Glance and A. Mian, "Adversarial Attacks and Defense on Deep Learning Classification Models using YCbCr Color Images," 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 2021, pp. 1-9, doi: 10.1109/IJCNN52387.2021.9533495.

What are Defenses for Adversarial Attacks?

- Mechanisms used to improve the robustness of machine learning models against intentional manipulations of adversarial attacks.

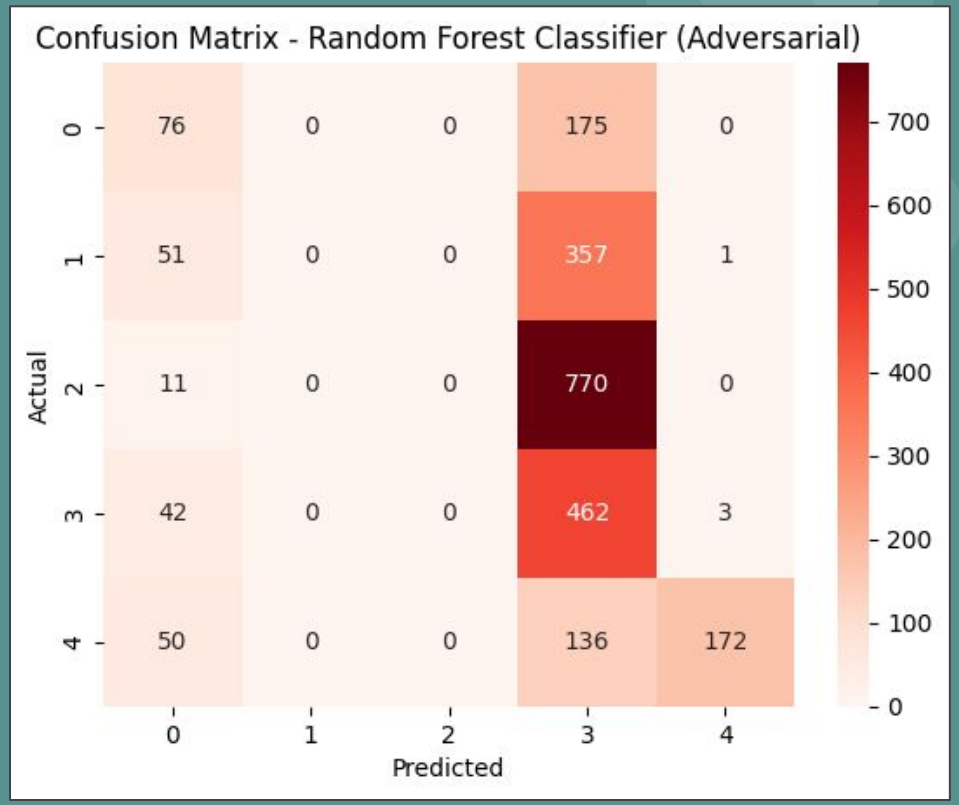
Adversarial Training

- Type of defense against adversarial attacks that trains a model based on both clean and adversarial samples.
- Helps the model to learn how to be robust against adversarial perturbations.

Step 4: CICMalDroid 2020 - FGSM Adversarial Attack Random Forest

```python
# Function to generate adversarial examples using feature importances
def fgsm_attack(X, importance, epsilon=0.1):
    perturbation = epsilon * importance
    X_adv = X + perturbation
    X_adv = np.clip(X_adv, 0, 1)
    return X_adv

# Calculate feature importances
importance = random_forest_model.feature_importances_

# Generate adversarial examples using FGSM attack
X_test_adv = fgsm_attack(X_test, importance, epsilon=0.1)

# Evaluate the model on adversarial examples
y_pred_rf_adv = random_forest_model.predict(X_test_adv)
```

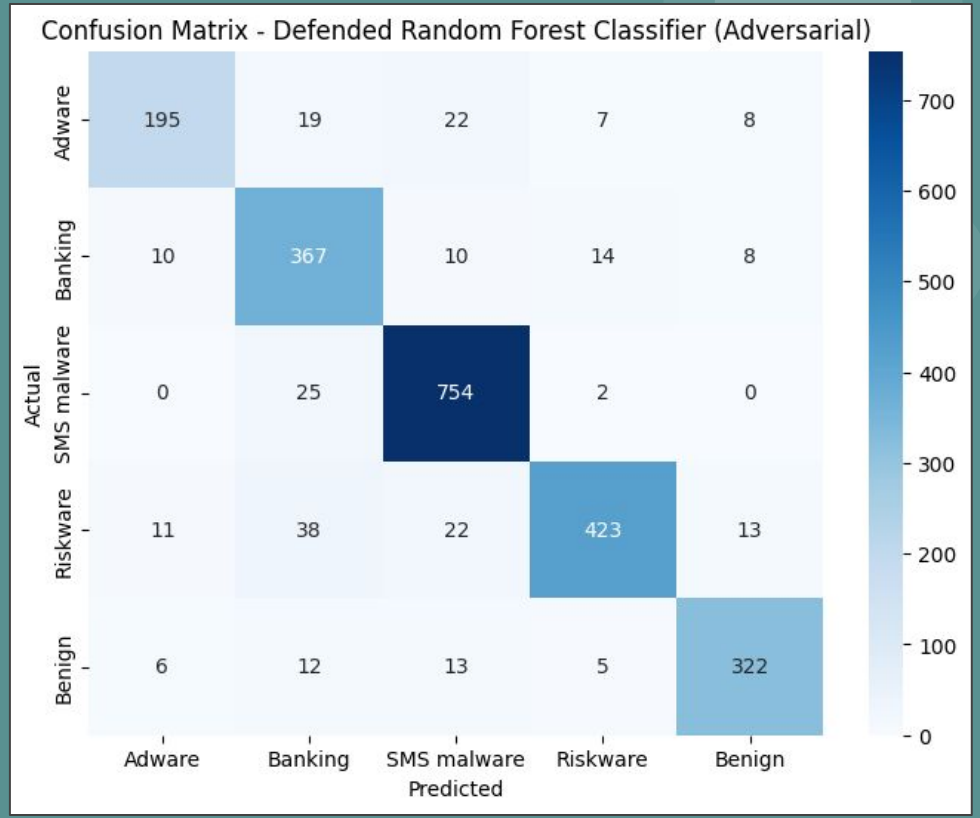
```
Metrics on Original Model:
Original Accuracy: 0.9480
Precision: 0.9484
Recall: 0.9480
F1-Score: 0.9479

Metrics on Adversarial Predictions:
Accuracy on Adversarial Examples: 0.3079
Precision: 0.2411
Recall: 0.3079
F1-Score: 0.2188
```



Step 5: CICMalDroid 2020 - FGSM Defense Mechanism Random Forest

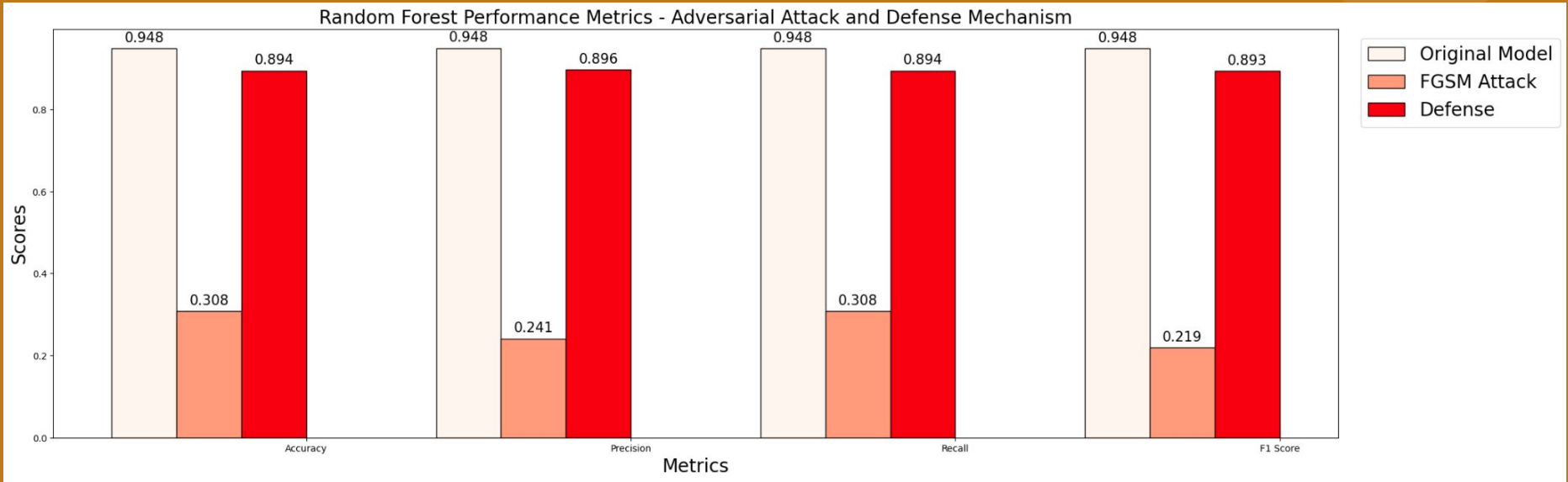
```python
# Function to perform adversarial training
def adversarial_training(model, X_train, y_train, epsilon=0.1, n_iterations=10):
    for _ in range(n_iterations):
        # Train on clean data
        model.fit(X_train, y_train)

        # Calculate feature importances
        importance = model.feature_importances_

        # Generate adversarial examples using FGSM attack
        X_train_adv = fgsm_attack(X_train, importance, epsilon)

        # Combine clean and adversarial examples
        X_train_combined = np.vstack([X_train, X_train_adv])
        y_train_combined = np.hstack([y_train, y_train])

        # Shuffle the data
        indexes = np.arange(len(X_train_combined))
        np.random.shuffle(indexes)
        X_train_combined_shuffled = X_train_combined[indexes]
        y_train_combined_shuffled = y_train_combined[indexes]

        # Retrain the model on the combined dataset
        model.fit(X_train_combined_shuffled, y_train_combined_shuffled)

    return model

# Adversarial training on the Random Forest model
adversarial_random_forest_model = adversarial_training(random_forest_model, X_train, y_train, epsilon=0.1)

# Evaluate the defended model on adversarial examples
y_pred_rf_adv = adversarial_random_forest_model.predict(X_test_adv)
```

```
Metrics on Adversarial Predictions after Adversarial Training:
Accuracy on Adversarial Examples: 0.8938
Precision: 0.8965
Recall: 0.8938
F1-Score: 0.8934
```



CICMaldroid 2020 Dataset

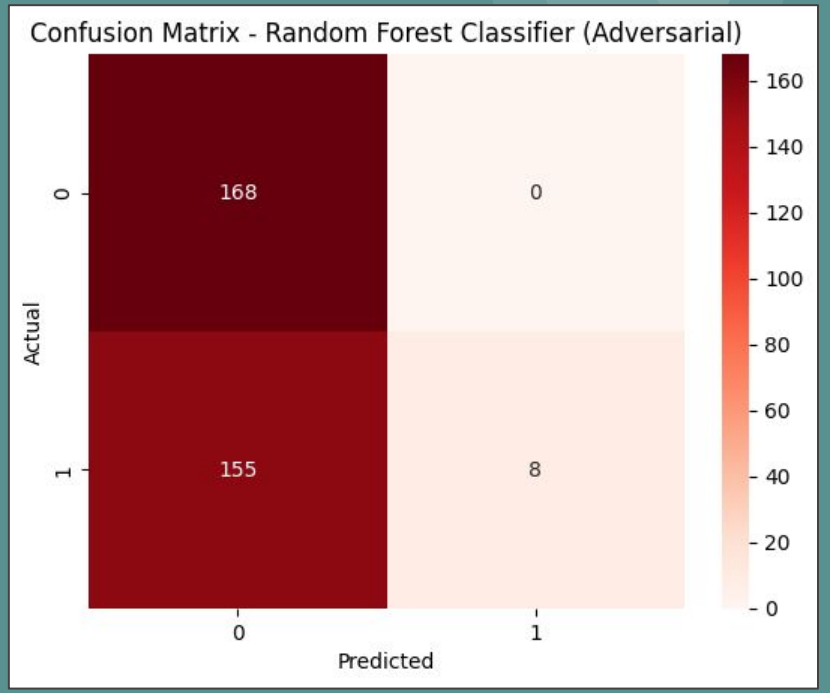
Random Forest vs. FGSM Adversarial Attack vs. Defense Mechanism



Step 4: DADA Dataset - FGSM Adversarial Attack - Random Forest

```python
# Function to generate adversarial examples using feature importances
def iterative_fgsm_attack_clip_range(X, importance, epsilon, clip_min, clip_max, num_iterations):
    perturbation = epsilon * importance
    X_adv = X.copy()

    for _ in range(num_iterations):
        X_adv += perturbation
        X_adv = np.clip(X_adv, clip_min, clip_max)

    return X_adv

# Calculate feature importances
importance = random_forest_model.feature_importances_
X_test_flt = X_test.astype('float64')

# Generate adversarial examples using iterative FGSM attack with clipping, increased epsilon, and more iterations
X_test_adv = iterative_fgsm_attack_clip_range(X_test_flt, importance, epsilon=1.0, clip_min=-1,
                                              clip_max=2, num_iterations=100)

# Evaluate the model on adversarial examples
y_pred_rf_adv = random_forest_model.predict(X_test_adv)
```

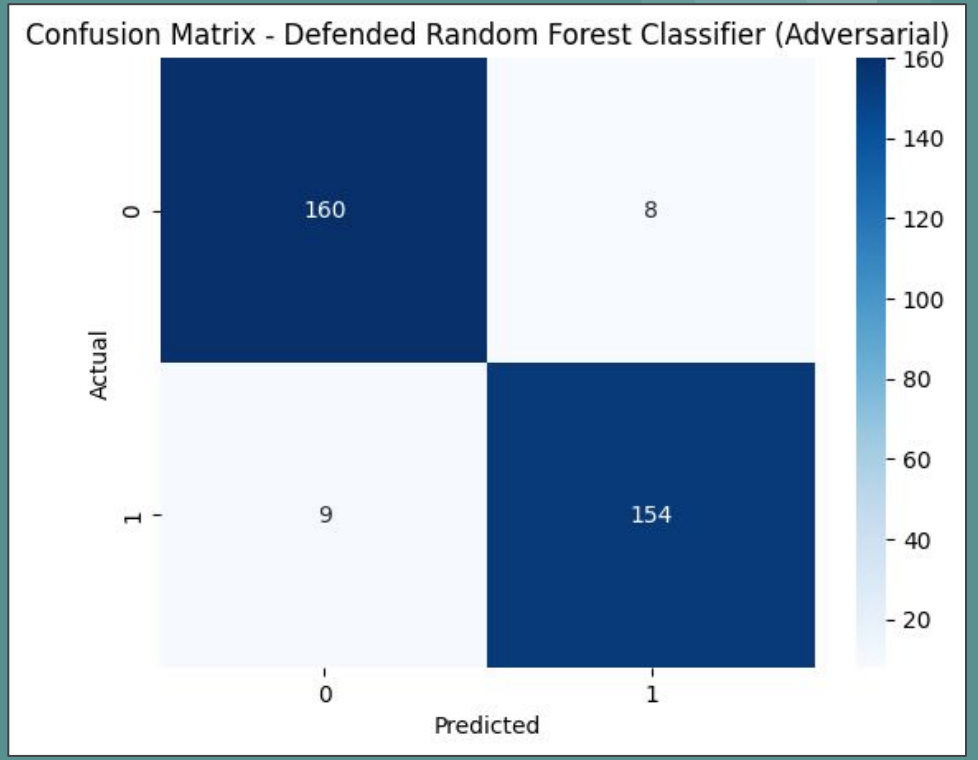
```
Metrics on Original Predictions:
Original Accuracy: 0.9637
Precision: 0.9640
Recall: 0.9637
F1-Score: 0.9637

Metrics on Adversarial Predictions:
Accuracy on Adversarial Examples: 0.5317
Precision: 0.7564
Recall: 0.5317
F1-Score: 0.3934
```



Step 5: DADA Dataset - FGSM Defense Mechanism - Random Forest

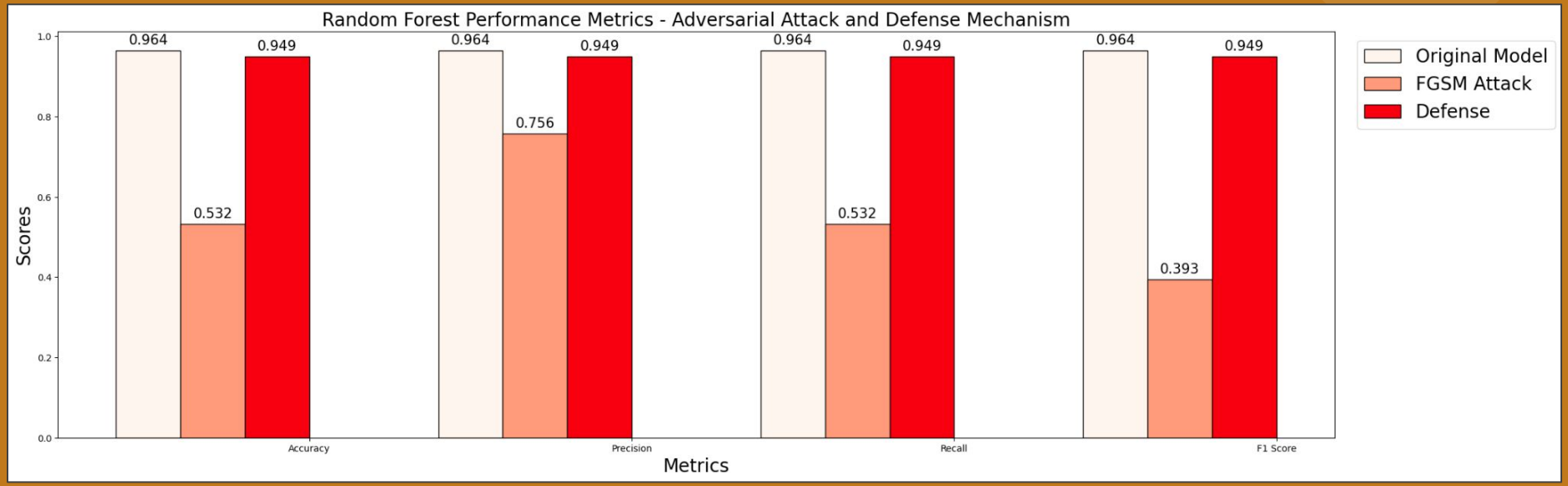
```python
# Function to perform adversarial training
def adversarial_training(model, X_train, y_train, importance, epsilon, clip_min, clip_max, num_iterations):
    for _ in range(num_iterations):
        # Generate adversarial examples
        X_train_adv = iterative_fgsm_attack_clip_range(X_train, importance, epsilon, clip_min, clip_max, num_iterations)

        # Ensure consistent data types
        X_train_combined = np.vstack([X_train.astype('float64'), X_train_adv])
        y_train_combined = np.hstack([y_train, y_train])

        # Shuffle the combined data
        idx = np.random.permutation(len(X_train_combined))
        X_train_combined_shuffled = X_train_combined[idx]
        y_train_combined_shuffled = y_train_combined[idx]

        # Retrain the model on the combined dataset
        model.fit(X_train_combined_shuffled, y_train_combined_shuffled)

    return model

# Convert data types for consistency
X_train_flt = X_train.astype('float64')
X_test_adv = X_test_adv.astype('float64')

# Adversarial training on the Random Forest model
random_forest_model_defense = adversarial_training(random_forest_model, X_train, y_train,
                                                   importance, epsilon=1.0, clip_min=-1, clip_max=2, num_iterations=100)

# Evaluate the defended model on adversarial examples
y_pred_rf_adv_defense = random_forest_model_defense.predict(X_test_adv)
```

```
Metrics on Adversarial Predictions (Defended Model):
Accuracy on Adversarial Examples: 0.9486
Precision: 0.9487
Recall: 0.9486
F1-Score: 0.9486
```



DADA Dataset

Random Forest vs. FGSM Adversarial Attack vs. Defense Mechanism



PowerBI Analytics Dashboard

Conclusions

Limitations

- With higher computational power the team would be able to extract APK files from AndroZoo and have our own developed dataset to run the developed models.
- Inability to detect corrupted APK files from AndroZoo

Final Thoughts

- Understanding the application and importance of defending machine learning models against adversarial attacks

Future Works & Improvements

- Implementing other Machine and Deep Learning Models
- Implementing different adversarial attacks on newly generated models

References

C. Pestana, N. Akhtar, W. Liu, D. Glance and A. Mian, "Adversarial Attacks and Defense on Deep Learning Classification Models using YCbCr Color Images," 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 2021, pp. 1-9, doi: 10.1109/IJCNN52387.2021.9533495.

G. D'Ambrosio and W. Li, "AdversarialDroid: A Deep Learning based Malware Detection Approach for Android System Against Adversarial Example Attacks," 2021 IEEE MIT Undergraduate Research Technology Conference (URTC), Cambridge, MA, USA, 2021, pp. 1-5, doi: 10.1109/URTC54388.2021.9701615.

H. Li, S. Zhou, W. Yuan, J. Li and H. Leung, "Adversarial-Example Attacks Toward Android Malware Detection System," in IEEE Systems Journal, vol. 14, no. 1, pp. 653-656, March 2020, doi: 10.1109/JSYST.2019.2906120.

W. Li, J. Ge and G. Dai, "Detecting Malware for Android Platform: An SVM-Based Approach," 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, New York, NY, USA, 2015, pp. 464-469, doi: 10.1109/CSCloud.2015.50.

W. Li, N. Bala, A. Ahmar, F. Tovar, A. Battu and P. Bambarkar, "A Robust Malware Detection Approach for Android System Against Adversarial Example Attacks," 2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC), Los Angeles, CA, USA, 2019, pp. 360-365, doi: 10.1109/CIC48465.2019.00050.

X. Chen et al., "Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 987-1001, 2020, doi: 10.1109/TIFS.2019.2932228.

Z. Shu and G. Yan, "EAGLE: Evasion Attacks Guided by Local Explanations against Android Malware Classification," in IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/TDSC.2023.3324265.

Z. Wang, J. Cai, S. Cheng and W. Li, "DroidDeepLearner: Identifying Android malware using deep learning," 2016 IEEE 37th Sarnoff Symposium, Newark, NJ, USA, 2016, pp. 160-165, doi: 10.1109/SARNOF.2016.7846747.

Thank you