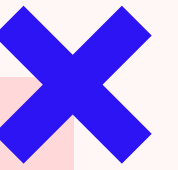
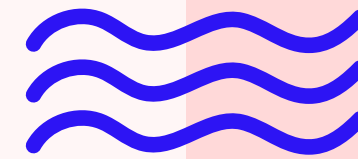


ANDROID MALWARE DETECTION USING APP PERMISSIONS



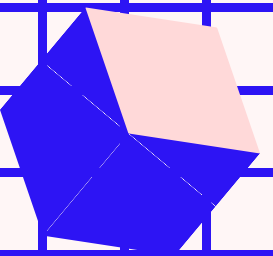
Project Report

Group - 7

Bijendar Prasad(2019238)

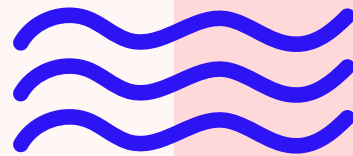
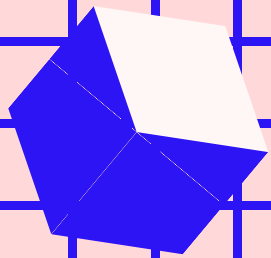
Abhishek Chaturvedi(2019401)

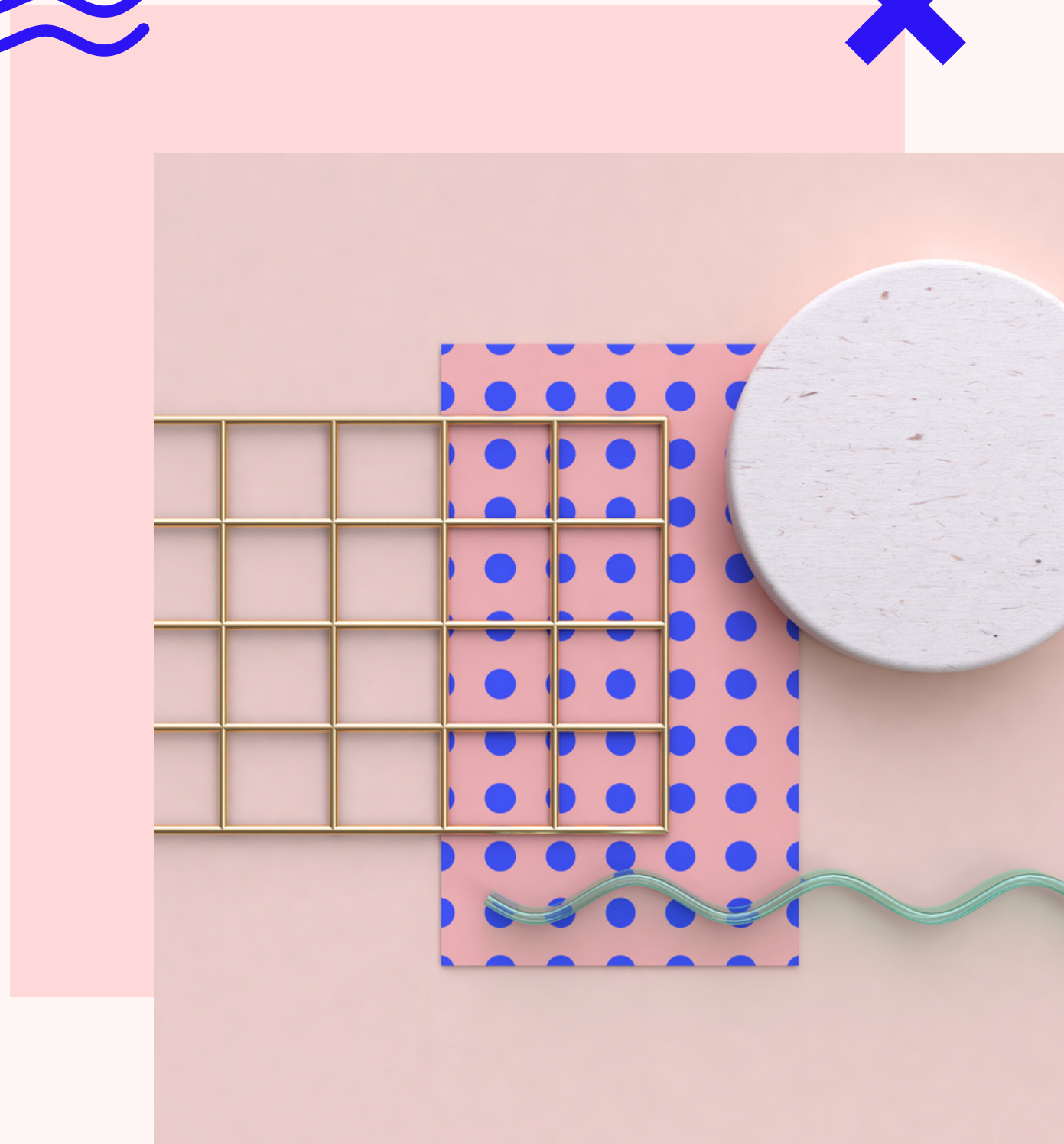
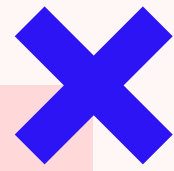
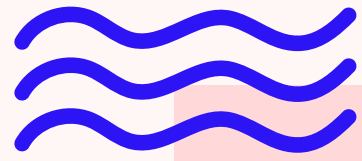
Deepesh Kumar(2019159)



Motivation

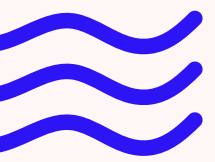
With the increasing boom in the android market, there is a constant increase of apps with malicious activities. According to ZDNet, 10%–24% of apps over the Play store could be malicious applications. On the surface, these apps look like any other standard app, but they exploit the user system in various harmful ways. The current methods to detect malwares are both resource heavy and exhaustive, yet fail to compete with the pace of new malwares.



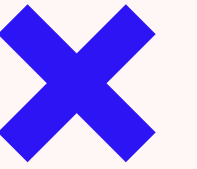


Introduction

Despite the increasing malwares, there is not yet an effective and robust method to detect malware applications. With the increasing applicativity of Machine Learning in various domains, we believe the issue of detecting Malware can be solved using Machine Learning techniques. Our project aims at a detailed and systematic study of malware detection using machine learning techniques, and further creating an efficient ML model which could classify the apps into benign(0) and malware(1) based on the requested app permissions.

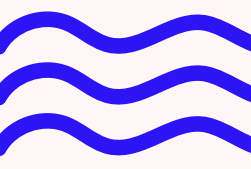


Literature Review




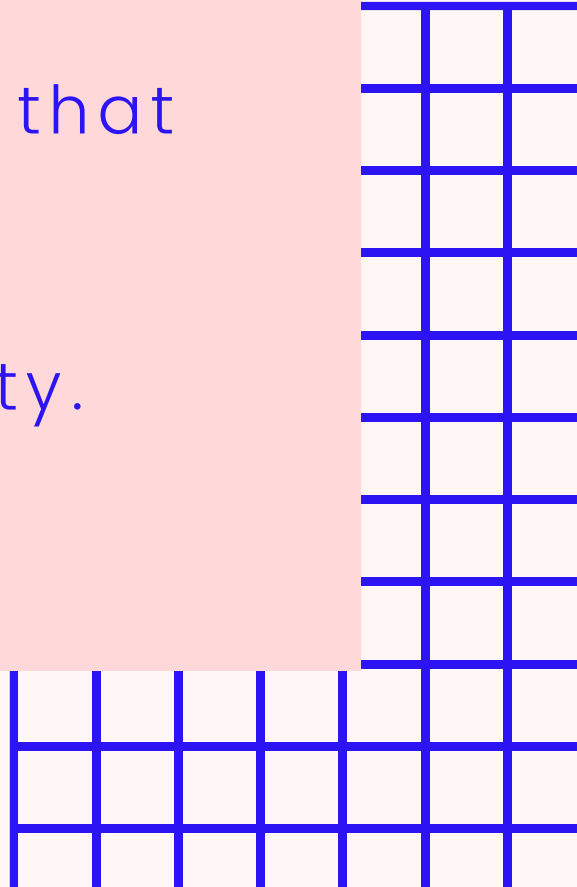
PAPER 1: ANDROID MALWARE PREDICTION USING MACHINE LEARNING TECHNIQUES

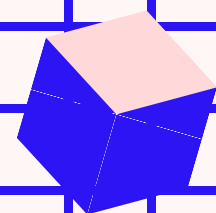
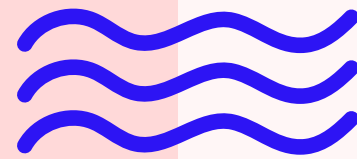
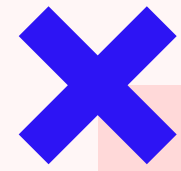
- The report analyzes machine-learning methods used to predict Android malware.
- Support vector machines, decision trees, naive Bayes, and random forests are commonly used.
- Characteristics like permissions, API requests, system calls, and network traffic are used to predict malware.
- Shortcomings of previous research include limited datasets and difficulty with obfuscated and polymorphic malware.
- Feature selection and dimensionality reduction methods can boost effectiveness.
- The author demonstrates the performance of machine learning techniques.
- However, they do not compare the feasibility, practicality, and performance of classical techniques and new ML techniques.
- Further, the study also fails to deliver why one ML technique outperforms the other.
- Another criticism against the paper is its lack of reasoning on why the app permissions stand out to be such a good measure for classifying benign and malicious applications.





PAPER 2: AN EFFICIENT ANDROID MALWARE PREDICTION USING ENSEMBLE MACHINE LEARNING ALGORITHMS

- The authors combine classification methods to categorize Android apps as malicious or benign.
 - They tested their classifier on a dataset of over 17,000 Android apps and found that their ensemble approach outperformed individual classification algorithms.
 - The ensemble approach had a high accuracy of over 99% in detecting malware.
 - The authors evaluated other machine-learning strategies and found that their approach outperformed them.
 - They conclude that their ensemble approach is highly successful for identifying malware in Android apps and can increase mobile security.
- 
- 

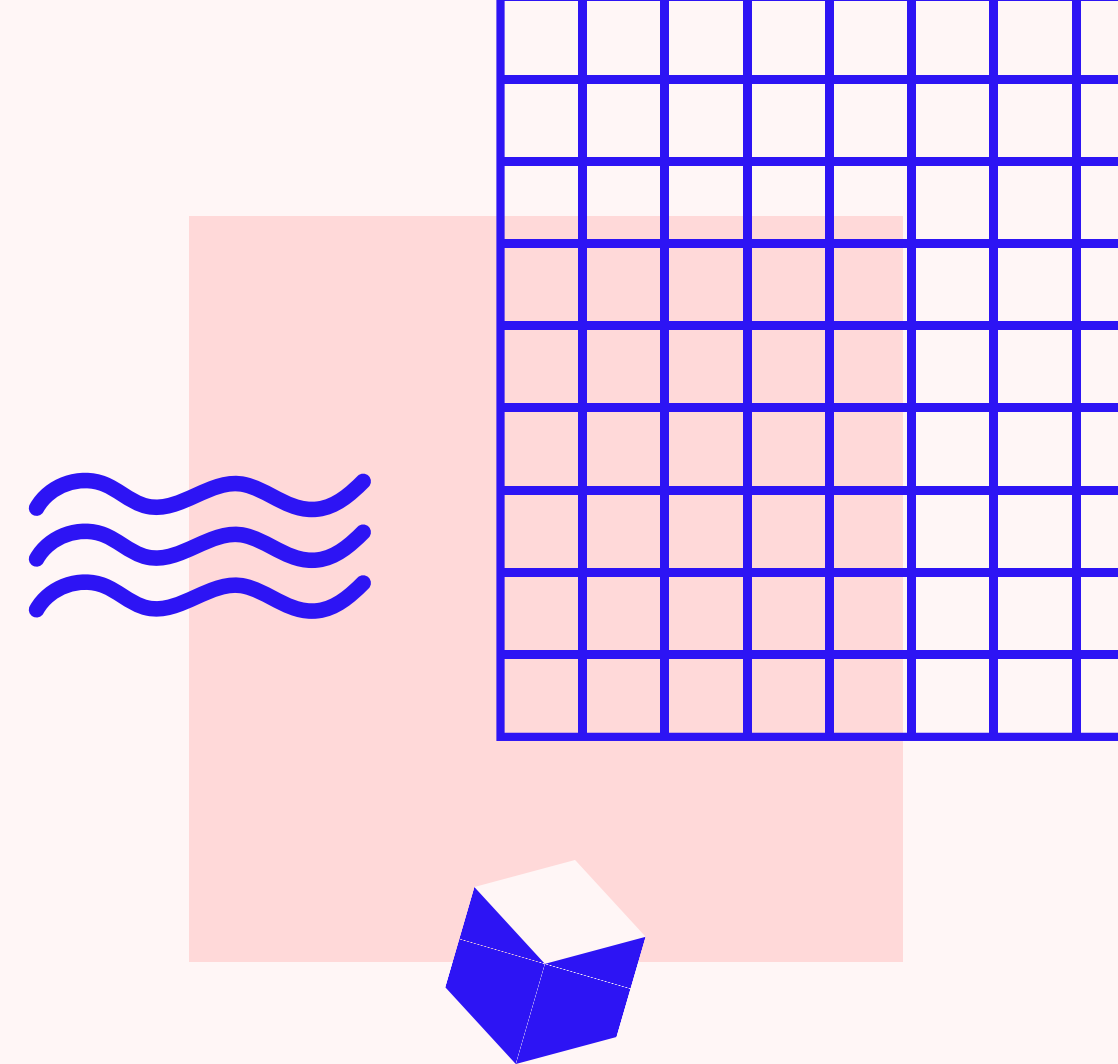


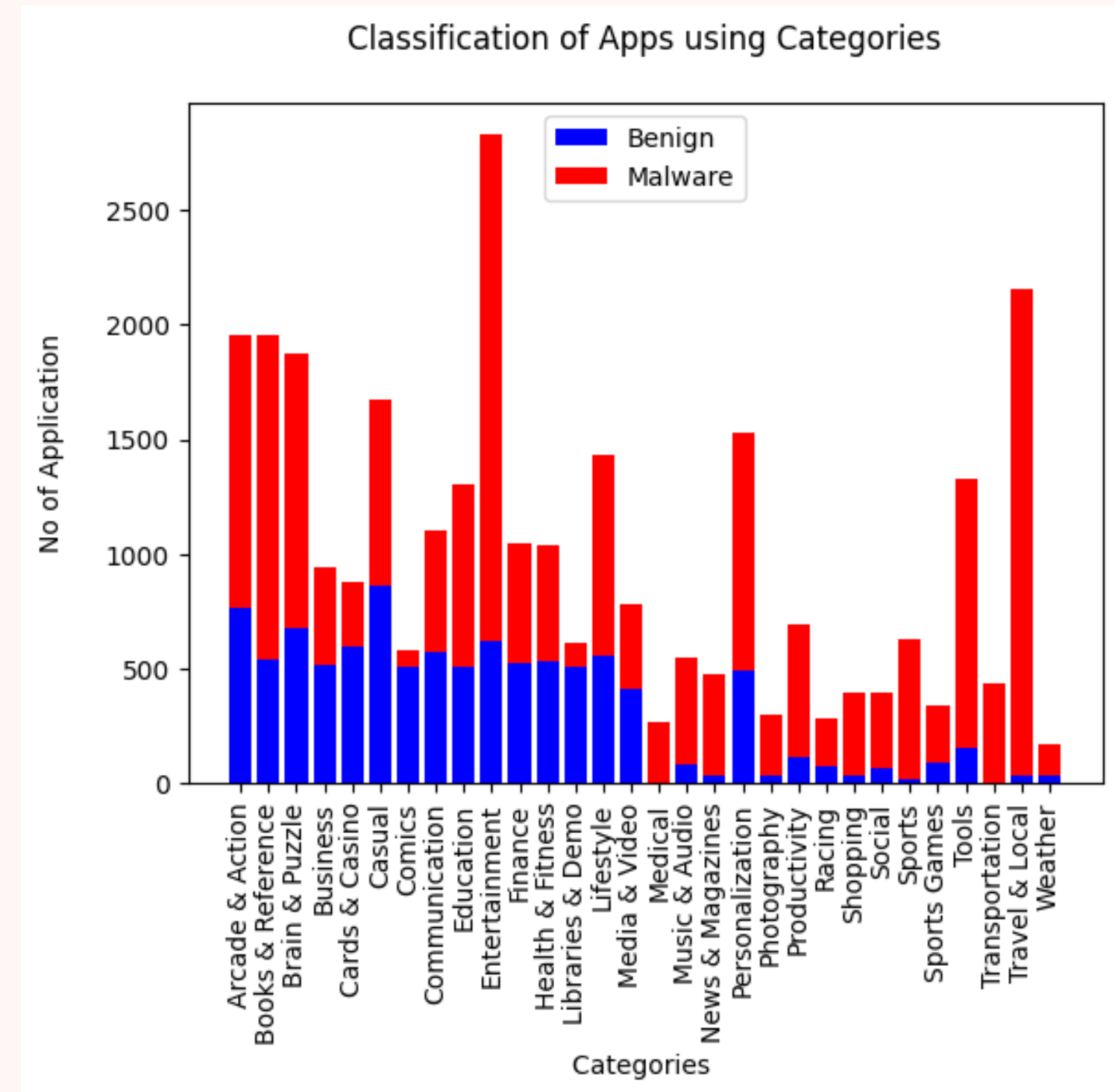
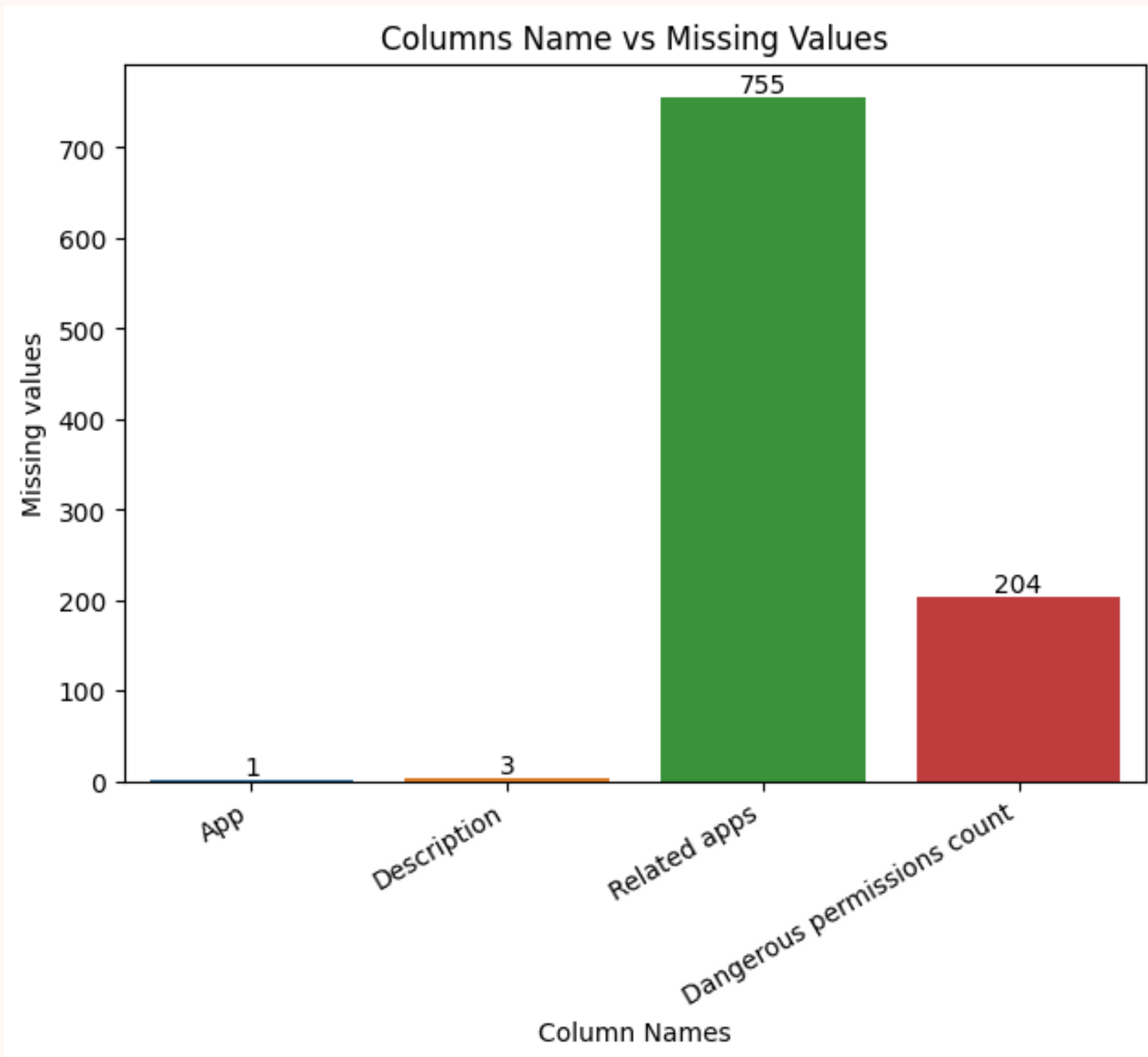
Dataset Description

- Dataset has been taken from Kaggle.
- Data contains the details of the permission of almost 30k app
- There are 183 features in the dataset like Dangerous Permissions Count, Safe Permissions Count, Rating, Category, etc.
- There is one target class (binary- 0/1) named - 'Class', indicating Benign(0) and Malware(1) applications.
- There are 29,999 records with 20,000 malwares and 9,999 benign apps.

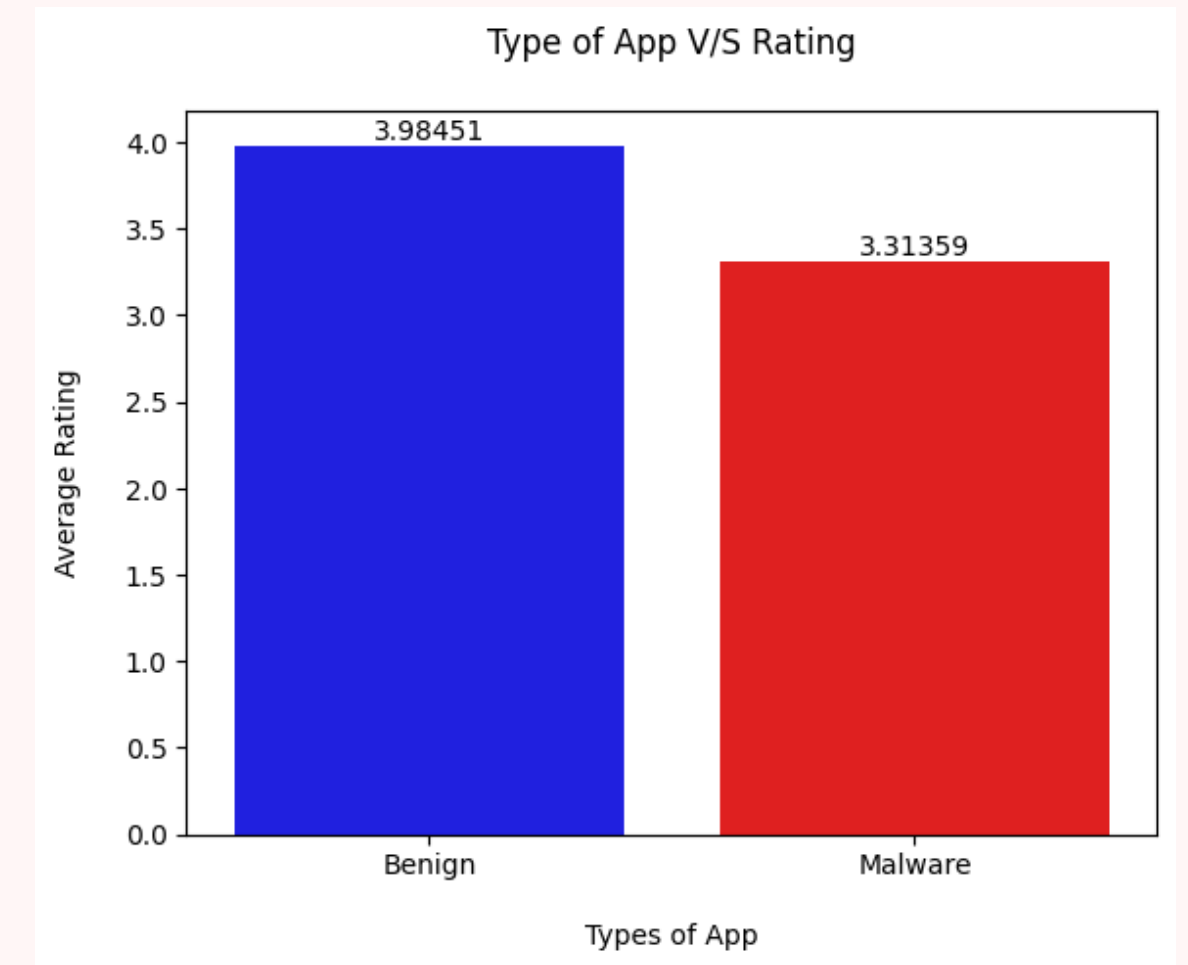
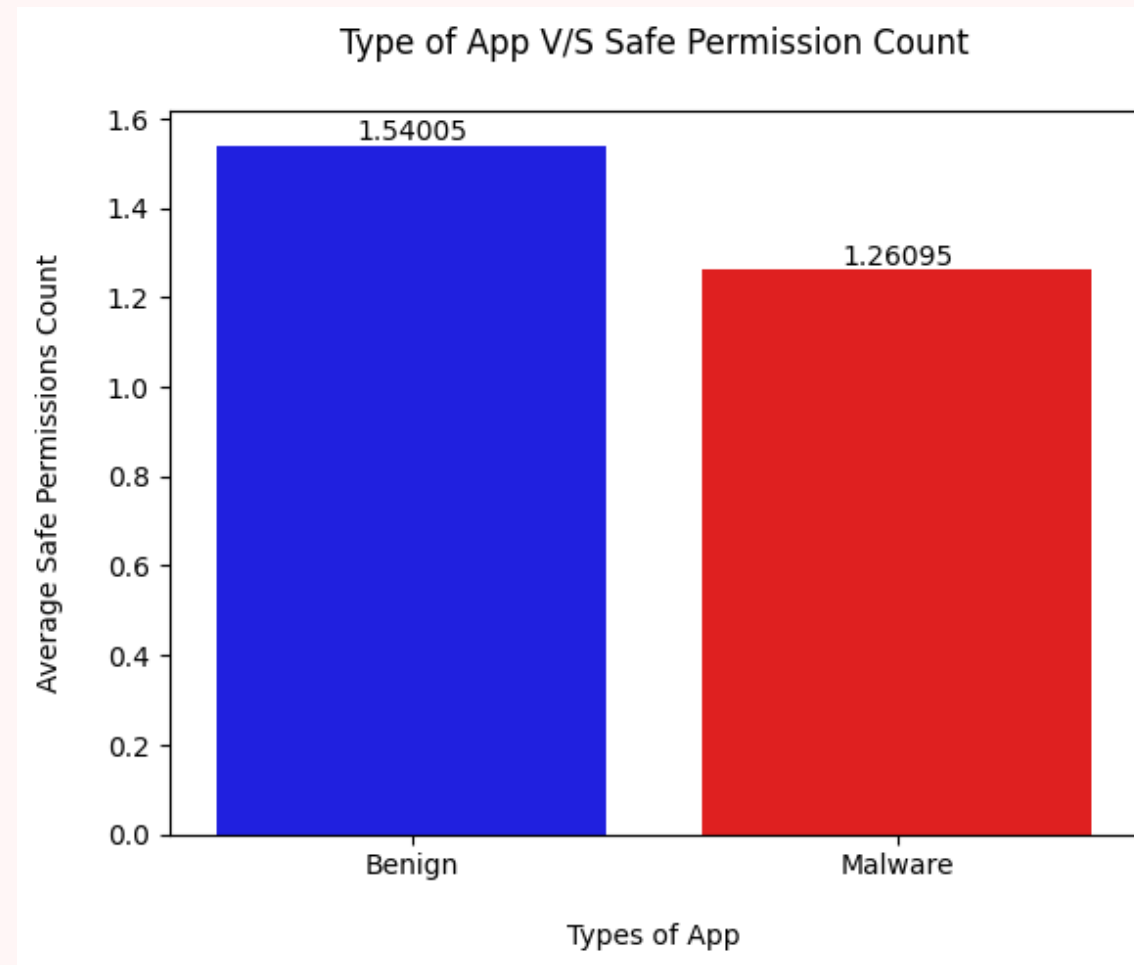
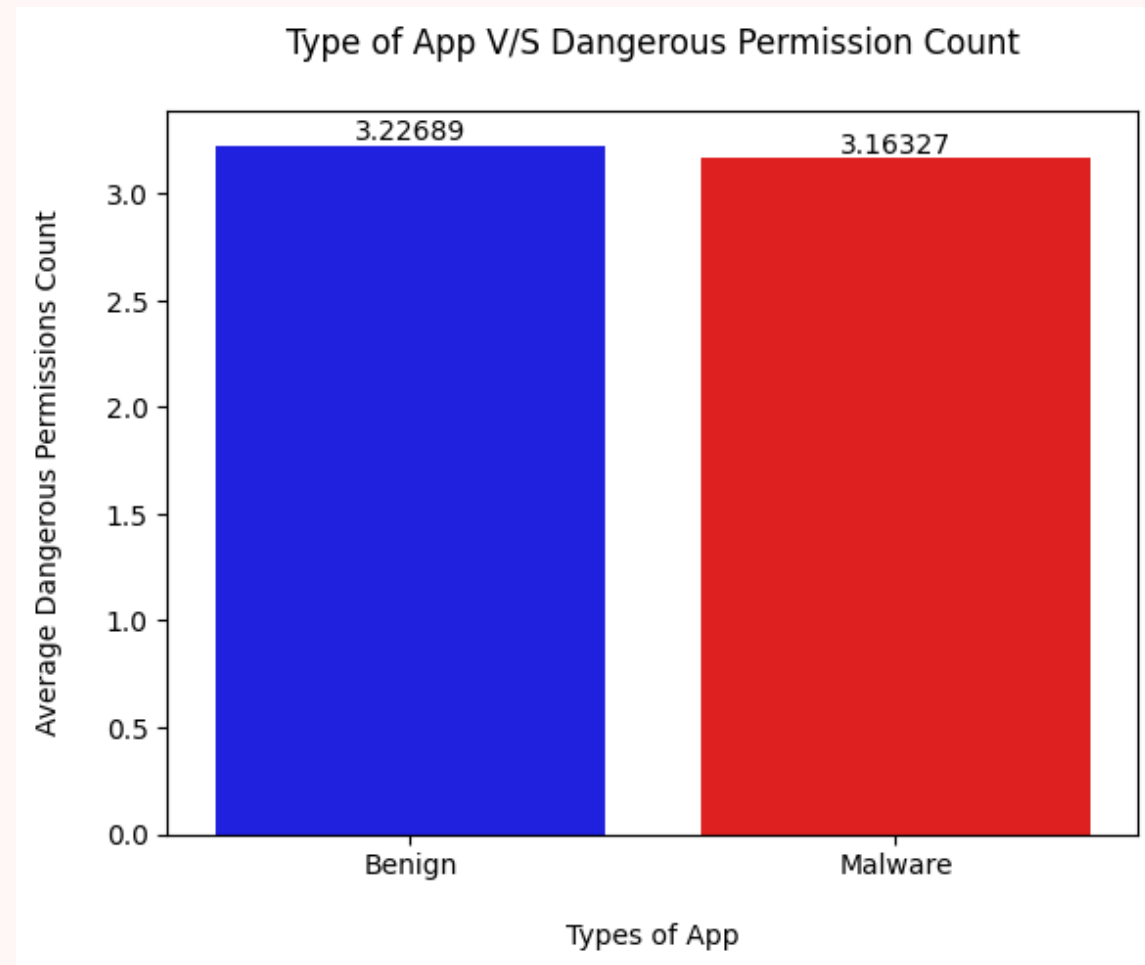
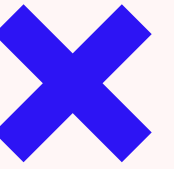
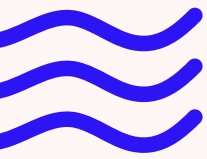
Preprocessing, Visualization, and Analysis:

- The data is first imported from a CSV file and loaded into a dataframe for ease of use.
- Some Columns like 'Dangerous Permissions Count' contains null values. We update such values by the mean of their respective columns.
- Several plots are built to better understand/analyse the data.
- Data is analysed based on the distribution of Malware and Benign applications in various settings, and several plots were made to visualise the results.
- Matplotlib and Seaborn are used for plotting and visualization.

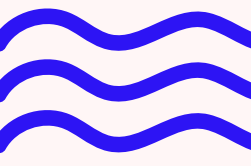


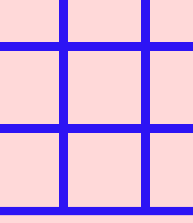
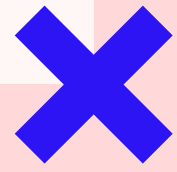


Plot for missing values and classification



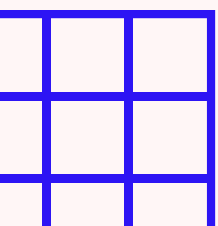
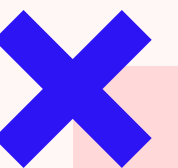
Plots for each features

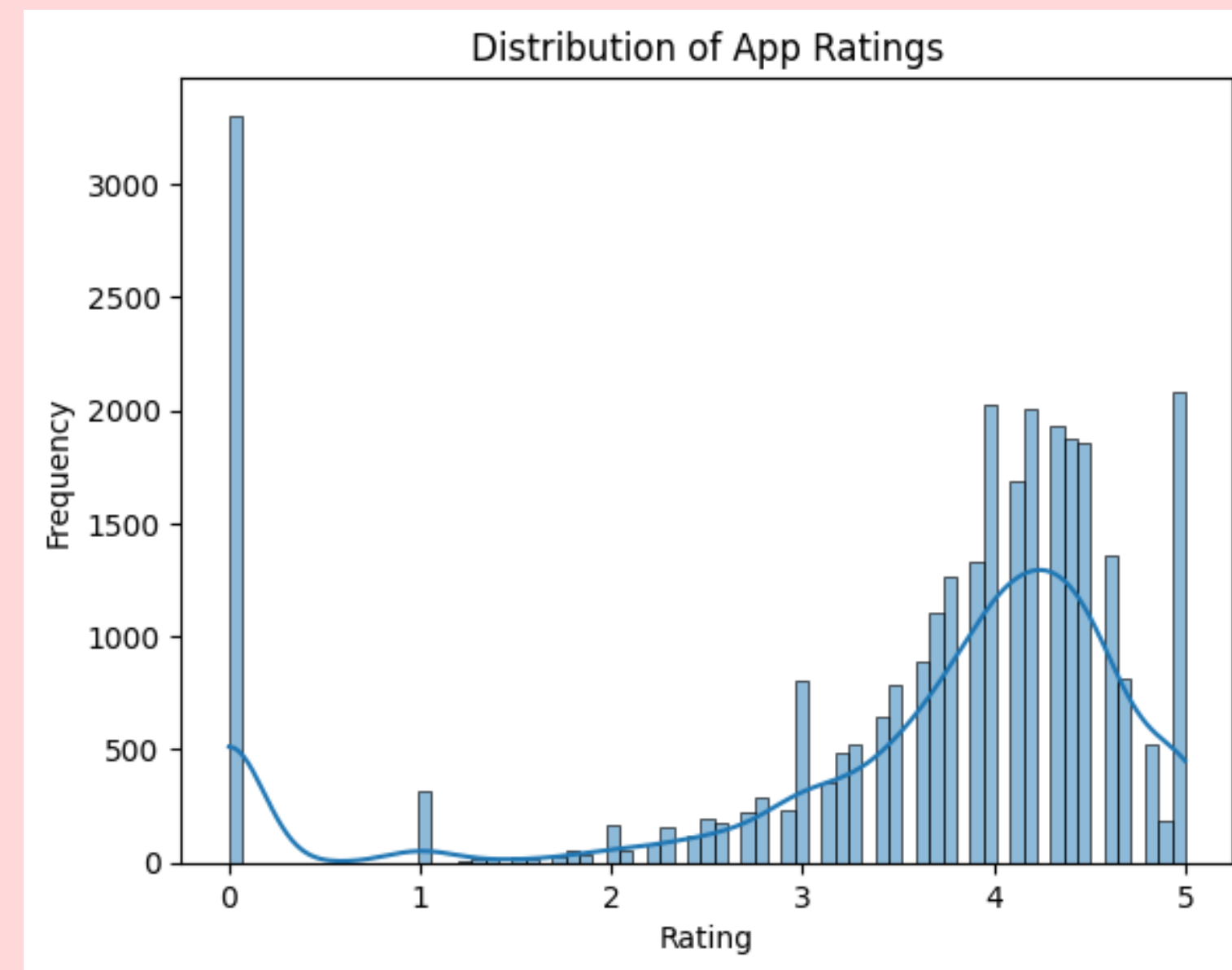
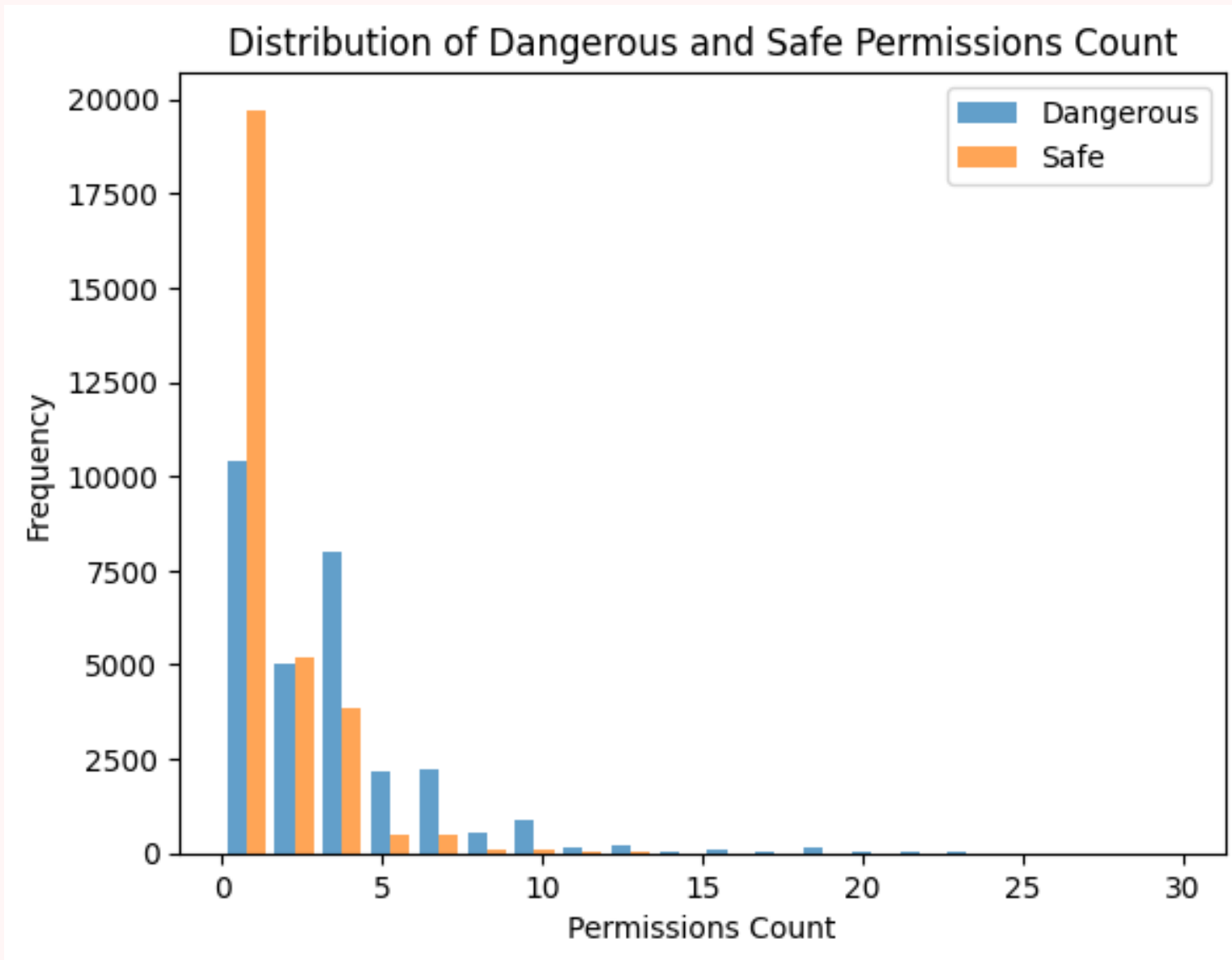




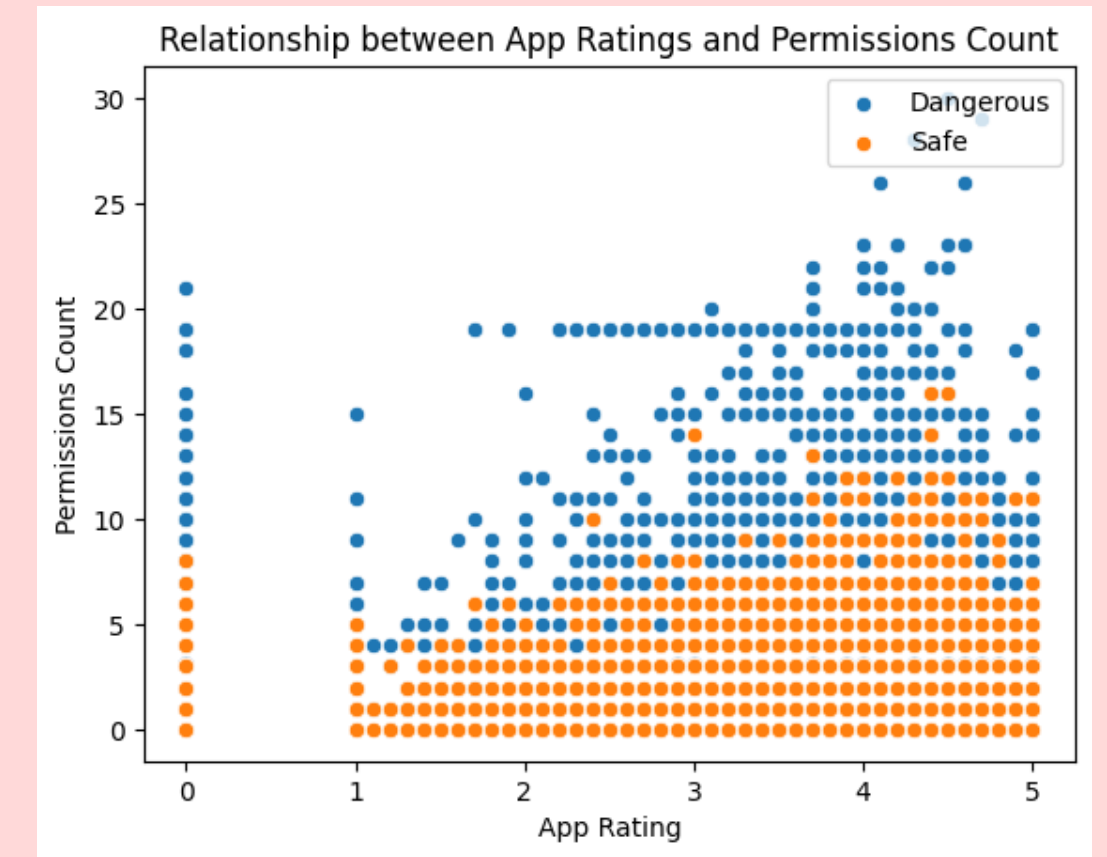
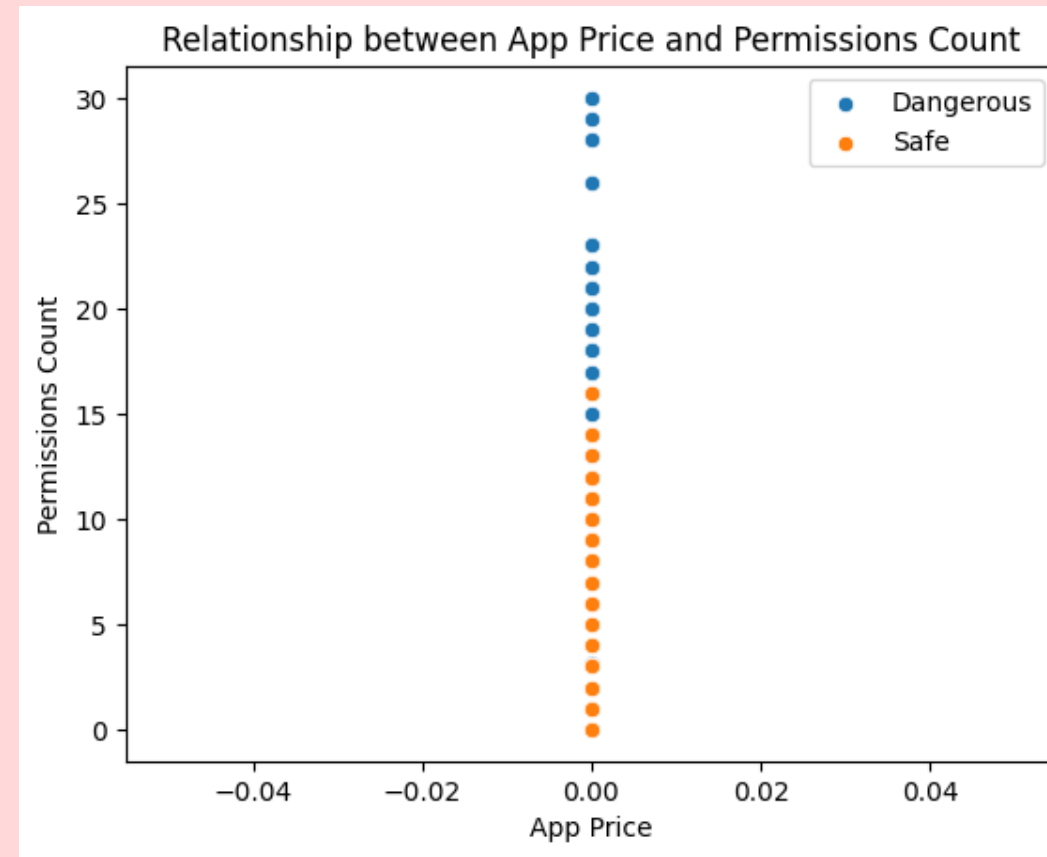
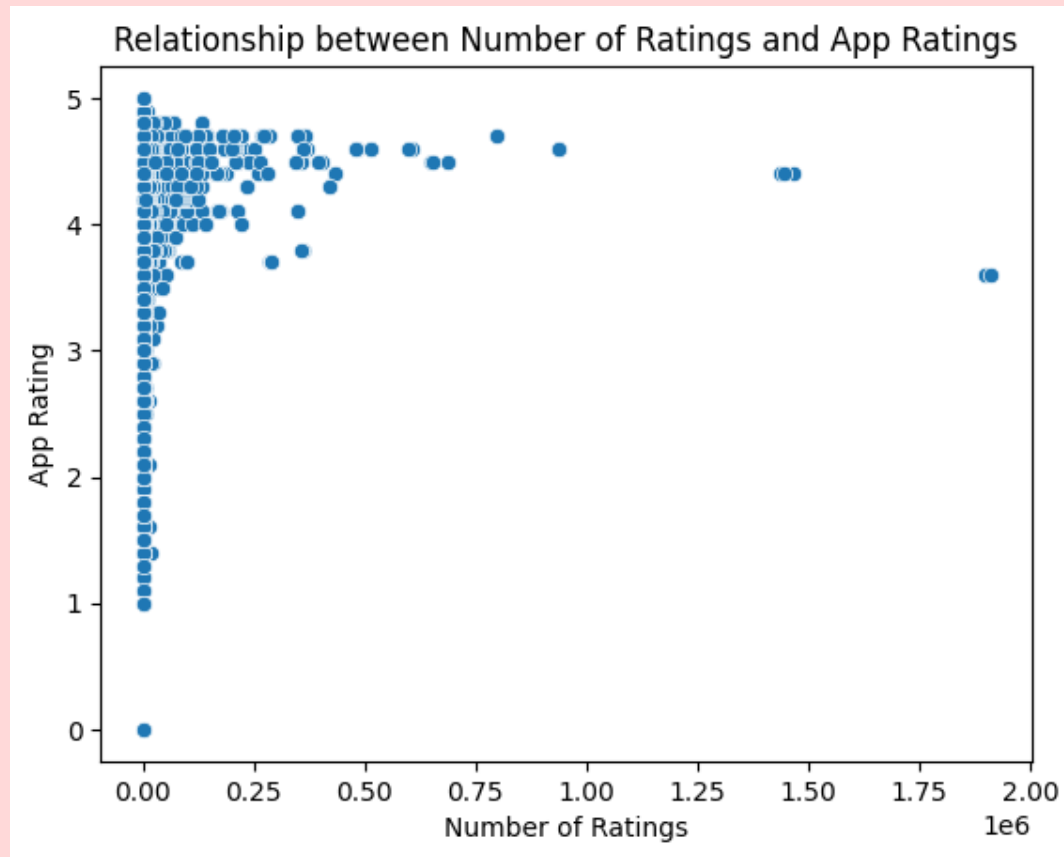
Exploratory Data Analysis(EDA)

The EDA for the Android Permission Dataset provided valuable insights into the relationships between different features in the dataset and helped us identify the most important features for predicting the app rating. It also provided a foundation for further analysis using machine learning techniques.





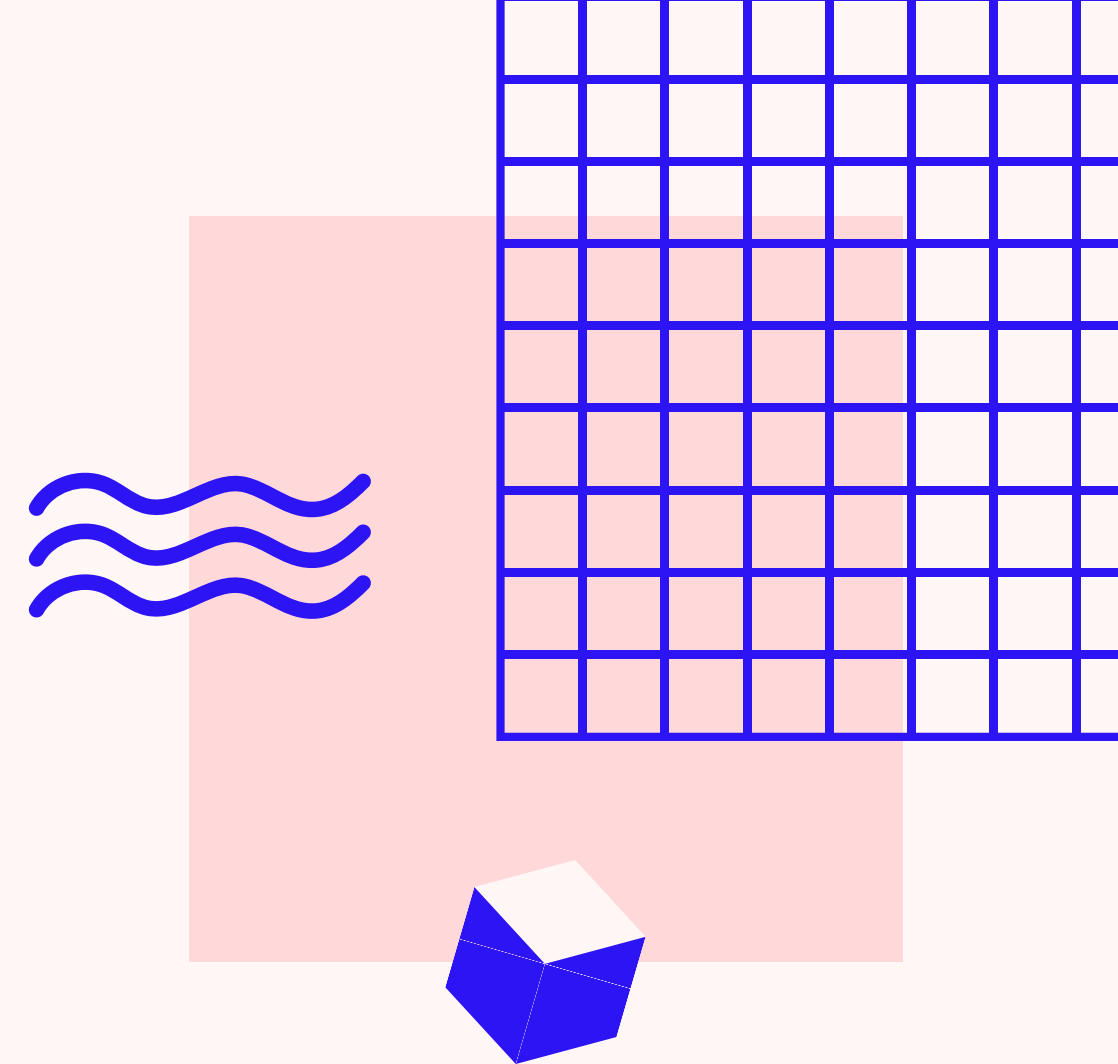
Distribution Plots

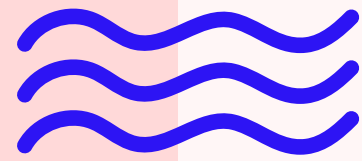
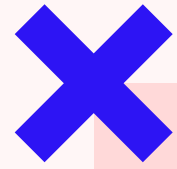


Relationship Plots

Methodology

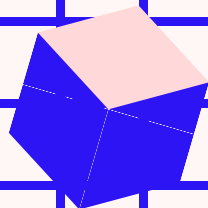
- Data is preprocessed and split into testing and training sets at an 8:2 ratio.
- Under and oversampling techniques were attempted but were not promising.
- Various classifiers were applied, but the outcomes were unsatisfactory.
- The dataset contained multivariate data tables, requiring PCA to be applied to each dataset.
- Random Forest was applied to the dataset, resulting in a significant improvement in accuracy.



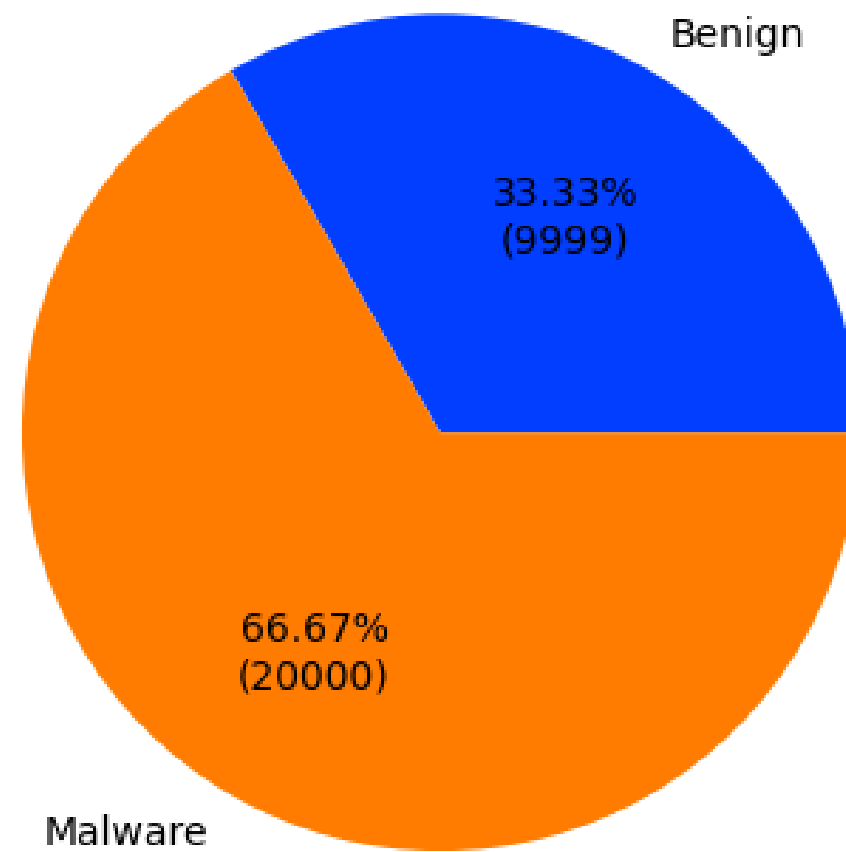


Methodology (continued)

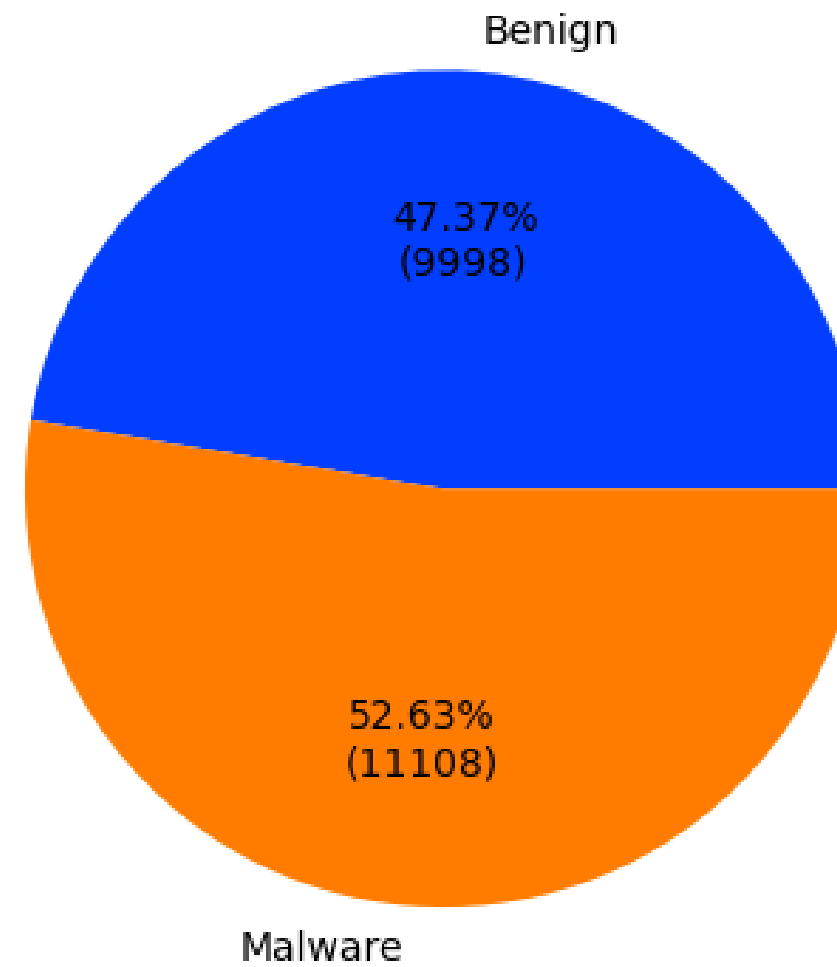
- Boosting was used to further increase prediction accuracy on unsampled and feature-selected datasets.
- SVM and MLP were applied to the final dataset, achieving the best results.
- Significant progress was made in accuracy after feature selection and boosting.



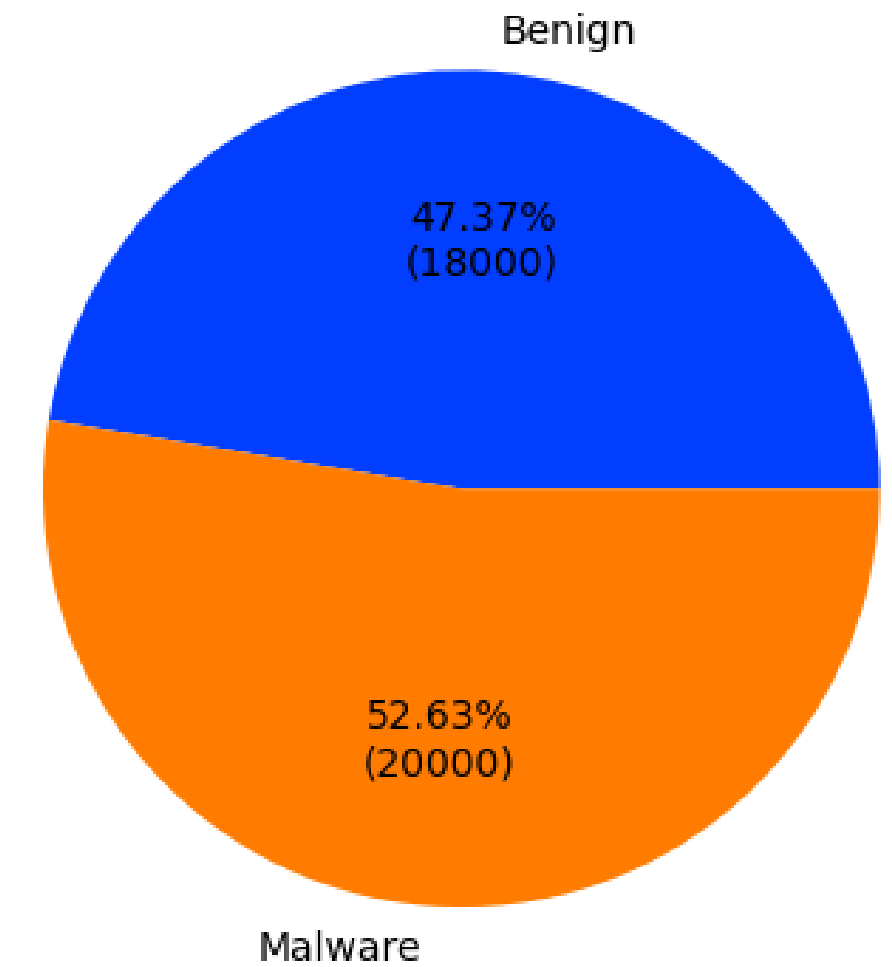
Class Distribution - Unsampled



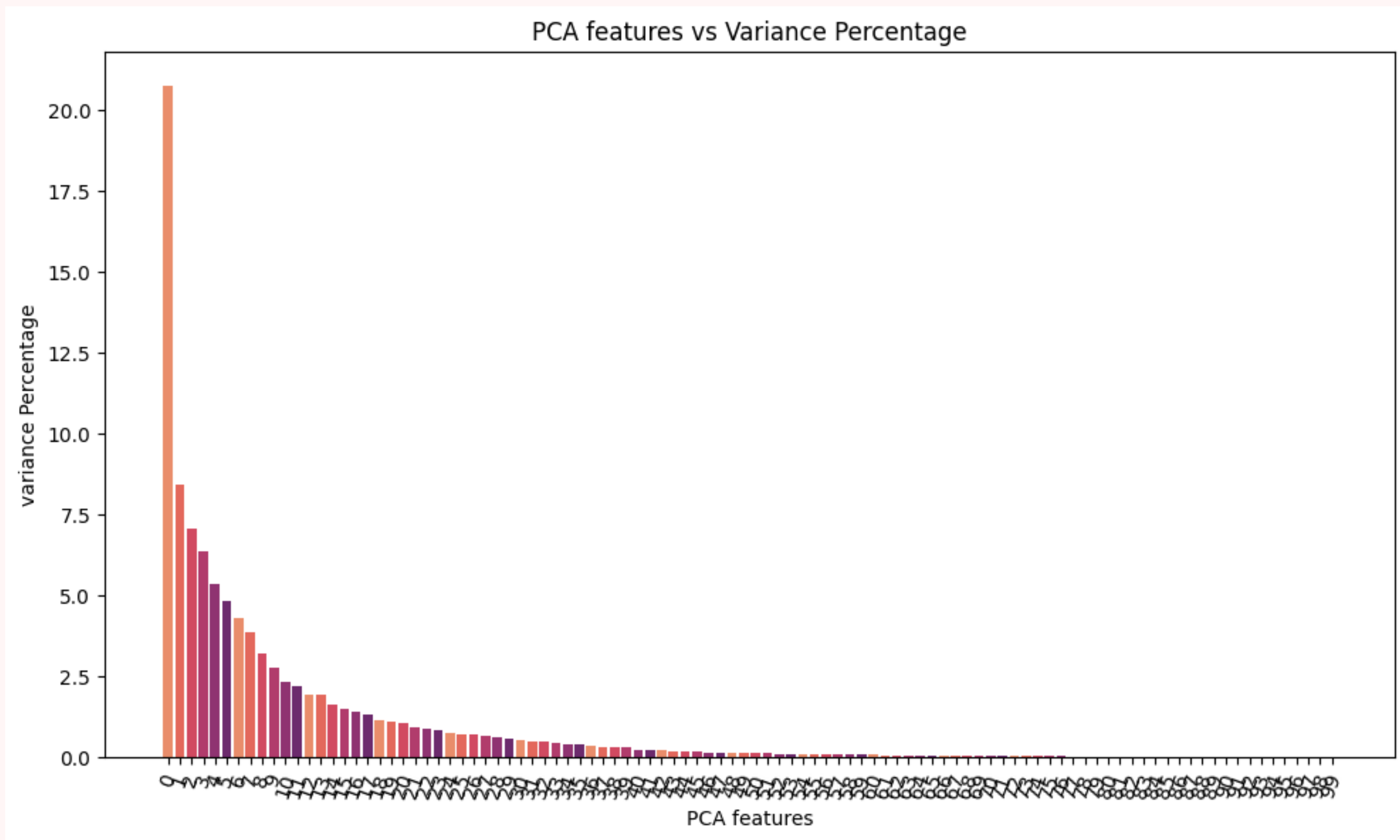
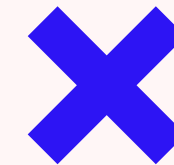
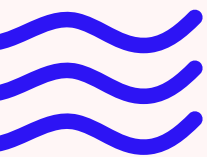
Class Distribution - Undersampled



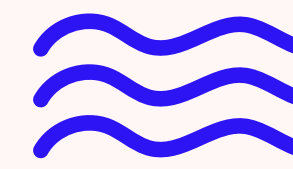
Class Distribution - Oversampled



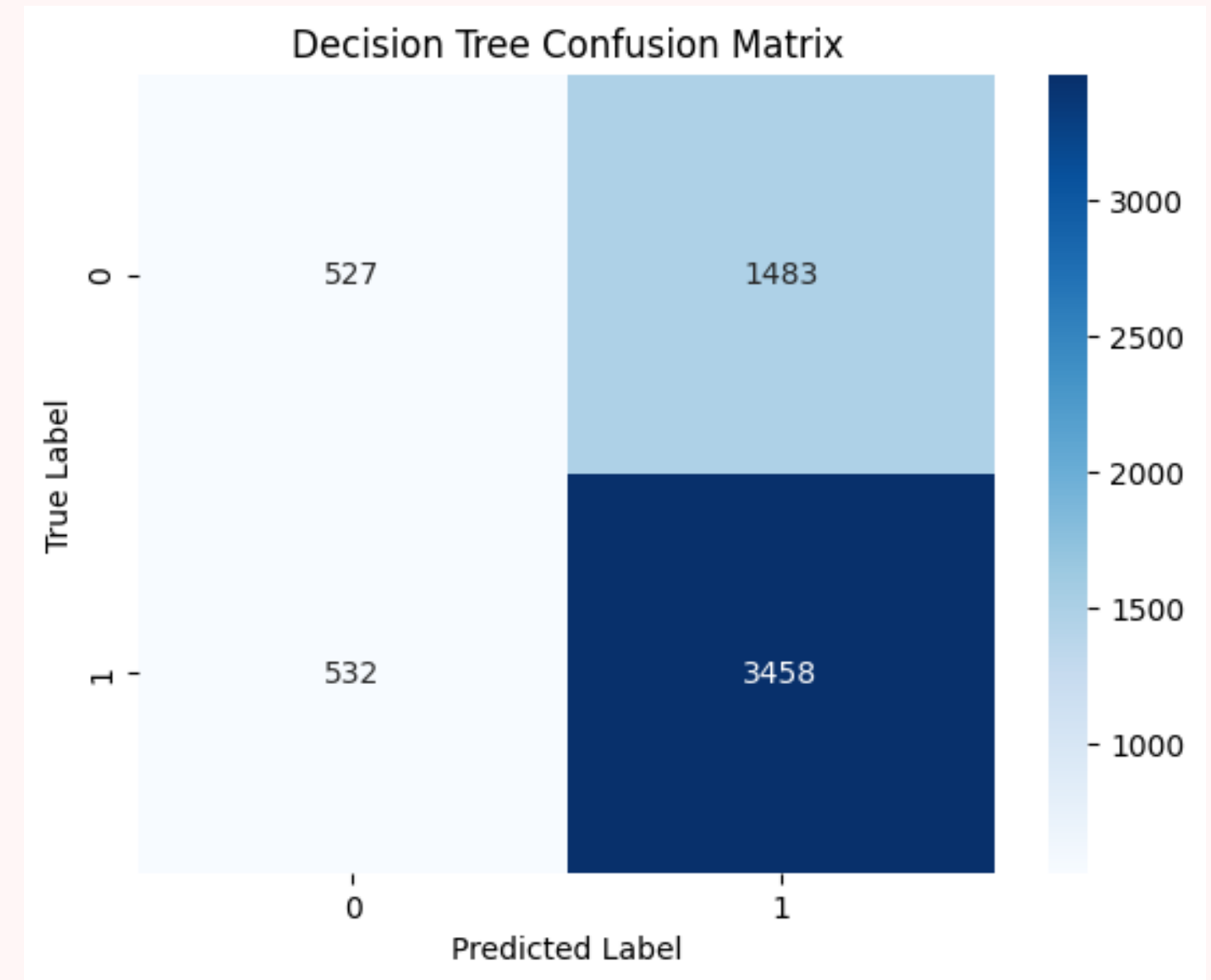
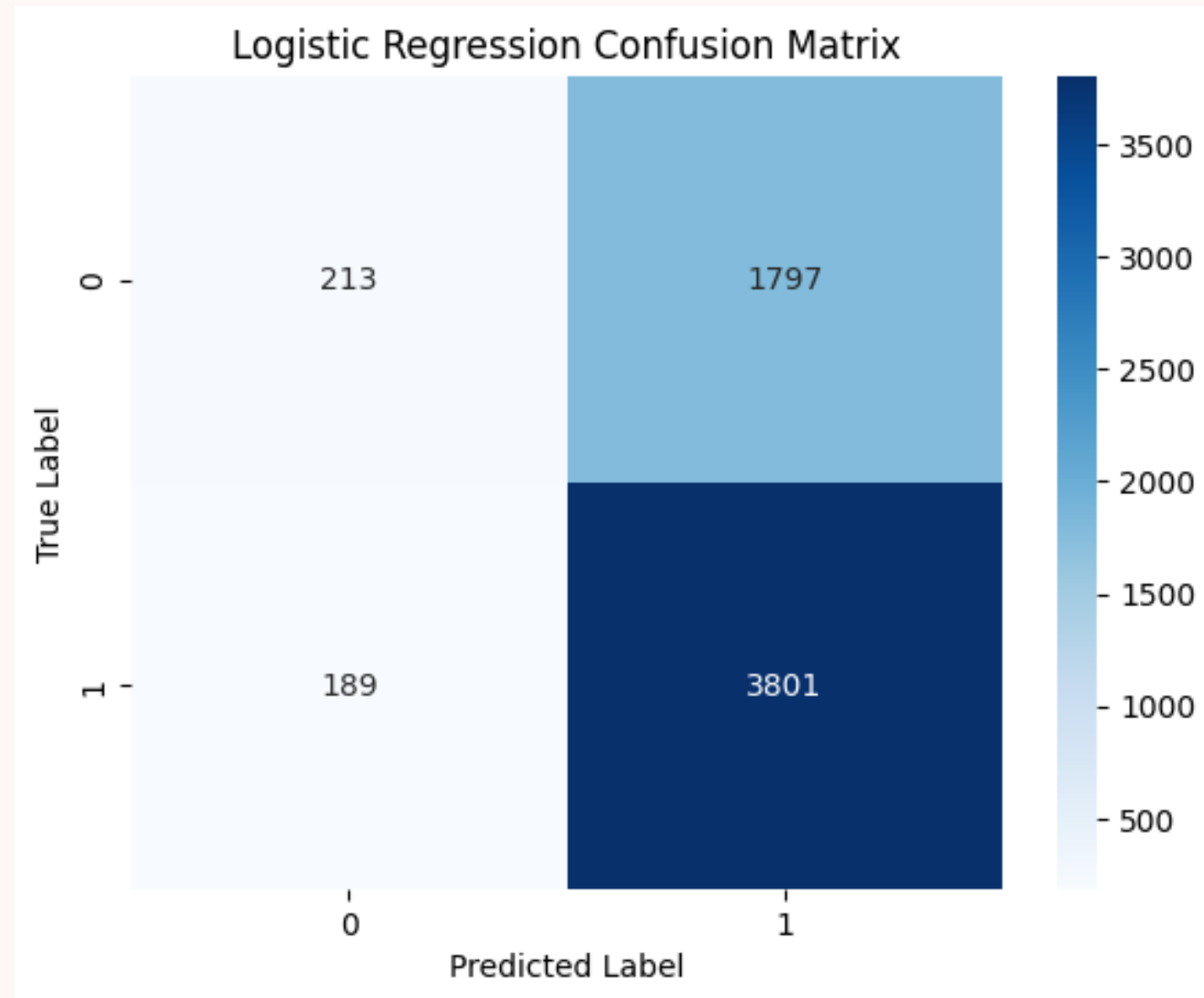
Class Distribution Plots

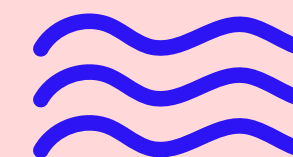
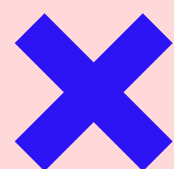
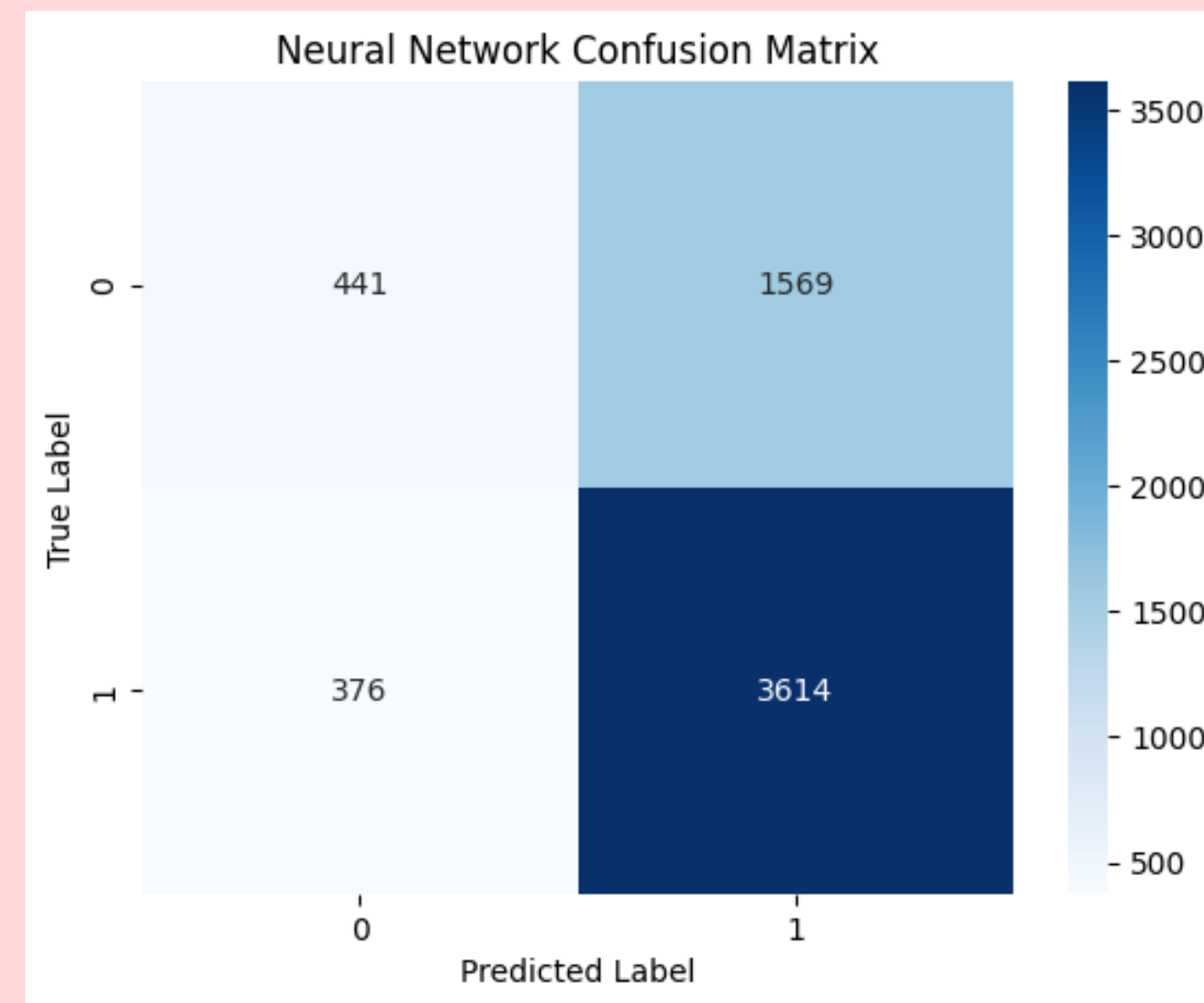
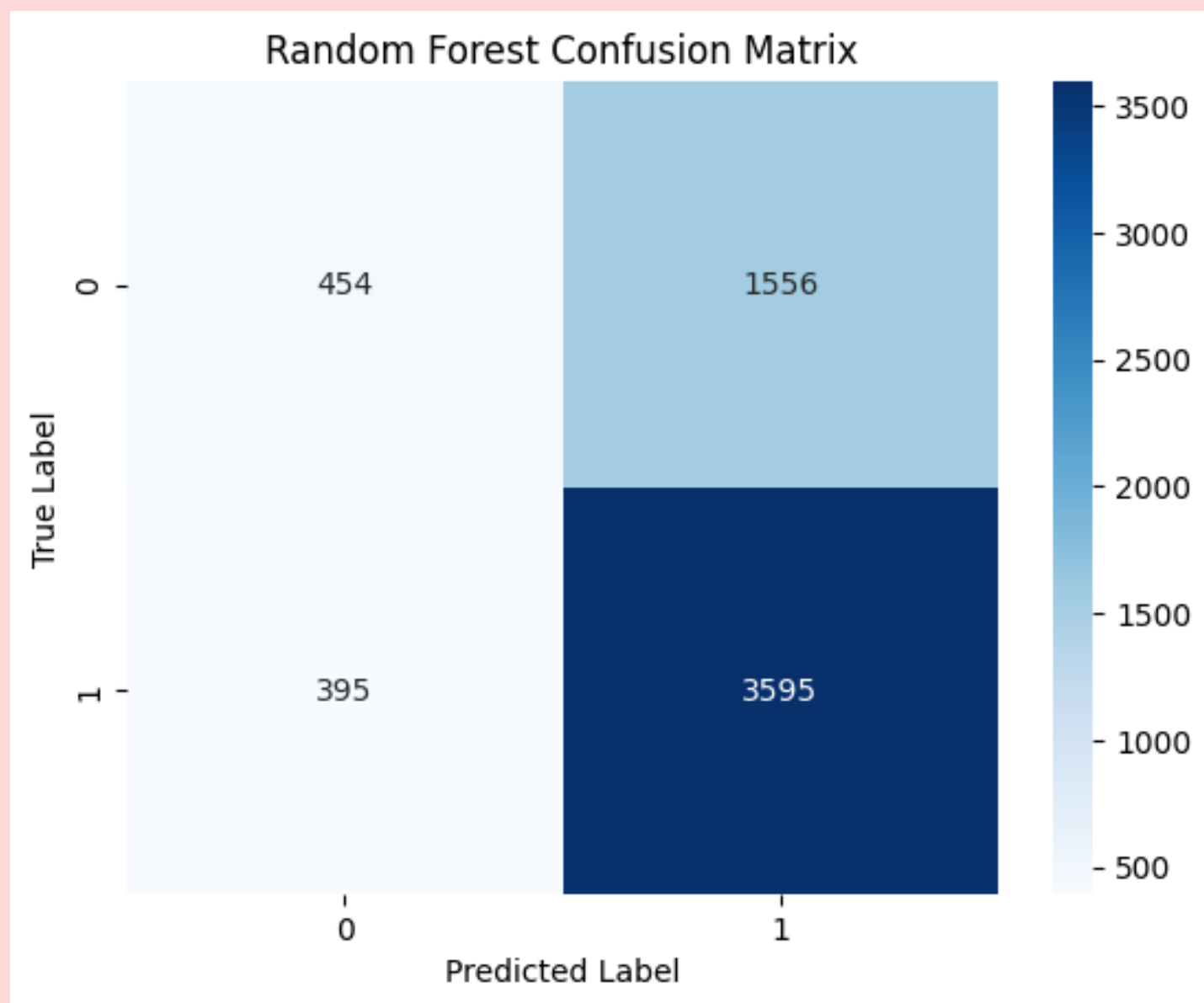
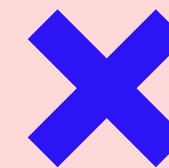
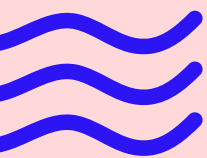


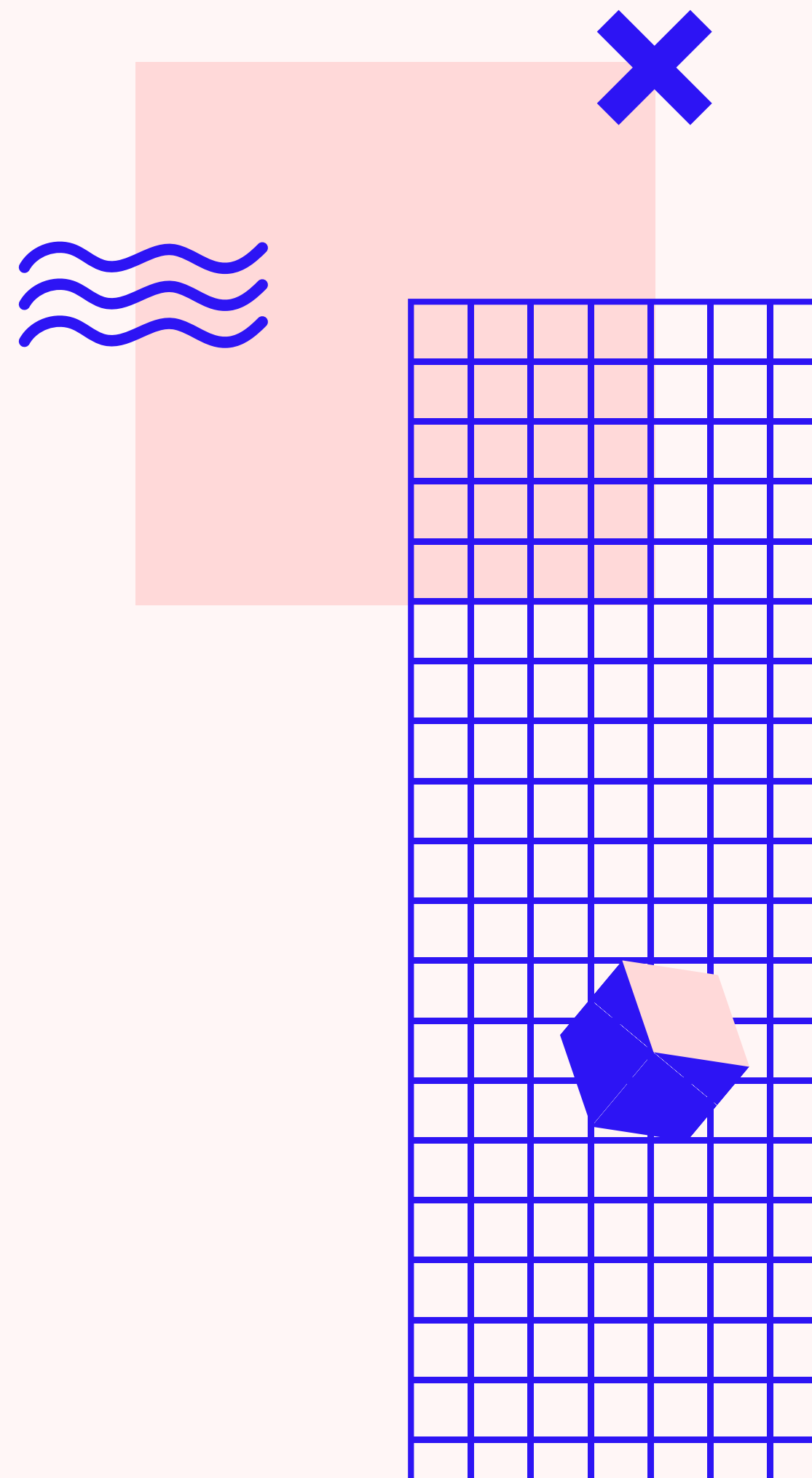
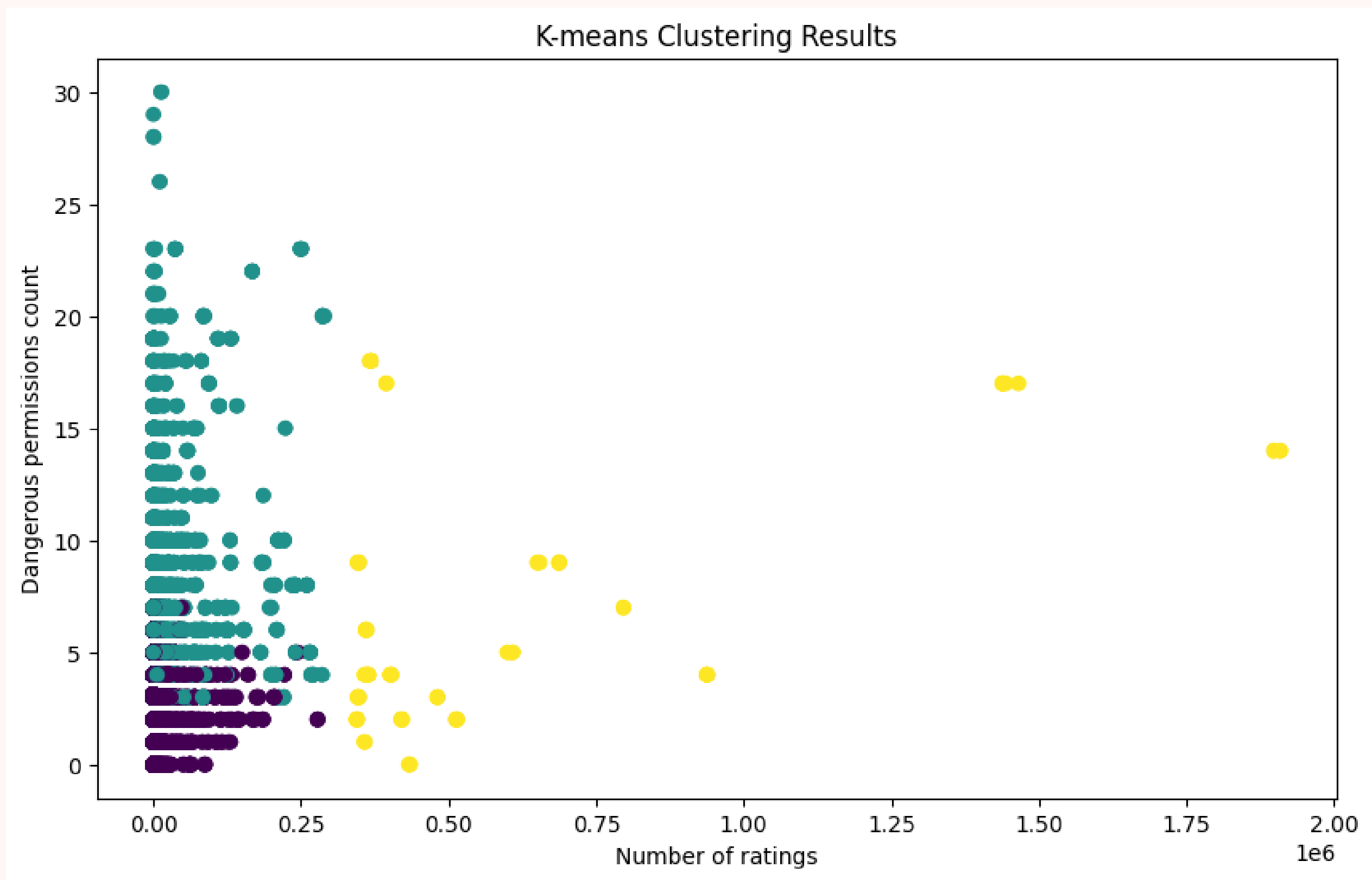
PCA Plots

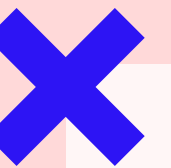
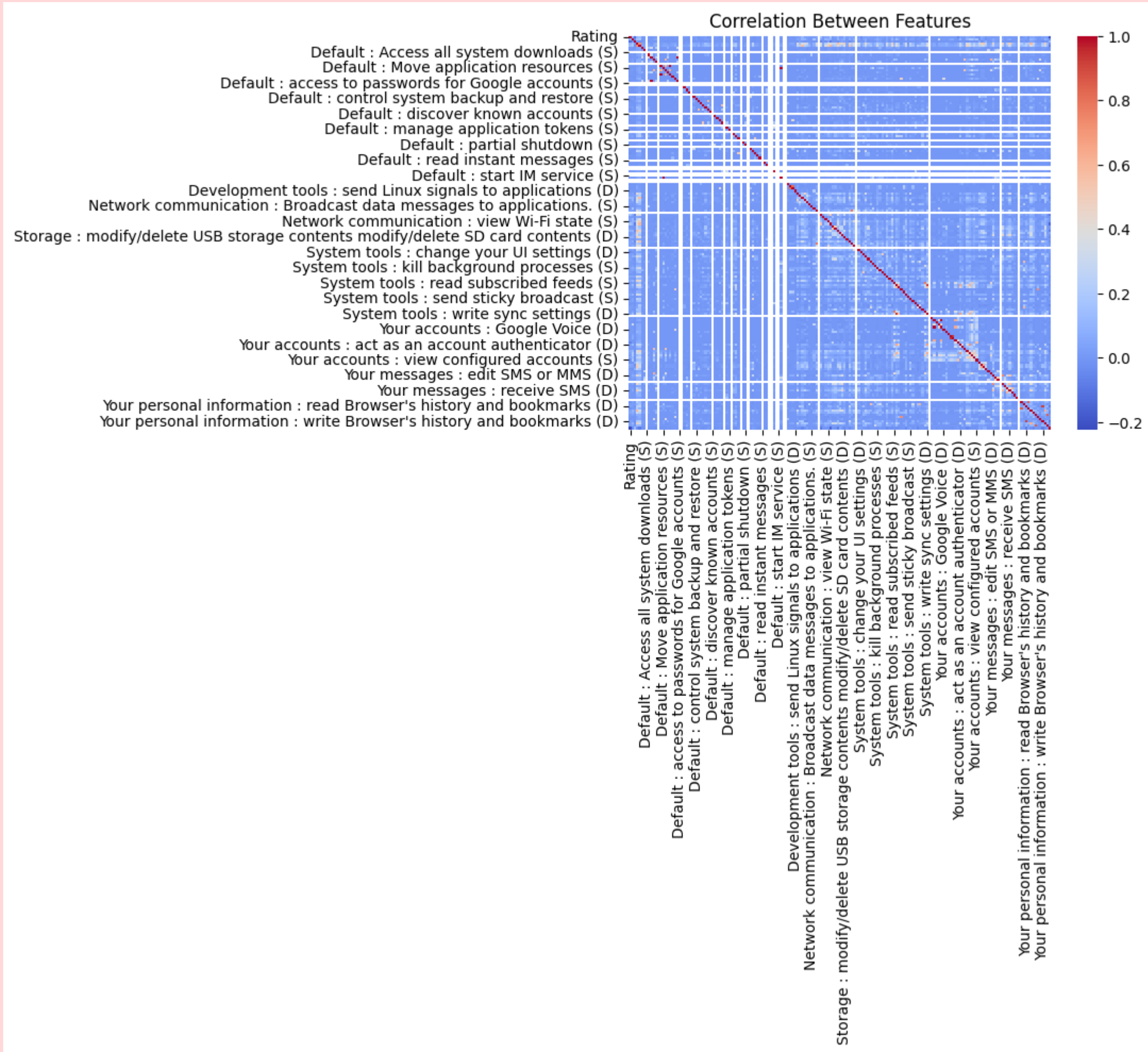
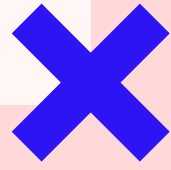


Results and Analysis









Model Trained on the UnSampled Data

Decision Tree (Criterion=Gini ,max_depth=10,max_leaf_nodes=10)	
Precision	0.7306158617634028
Accuracy	0.6791666666666667
Recall	0.8230596456201648
Roc_Auc	0.6064620857303031

Logistic Regression (Default)	
Precision	0.6682820855614974
Accuracy	0.6678333333333333
Recall	0.9980034938857
Roc_Auc	0.5010087715289011

GaussianNB (Default)	
Precision	0.9617117117117117
Accuracy	0.5371666666666667
Recall	0.3196905415522835
Roc_Auc	0.6470504890400655

Overall

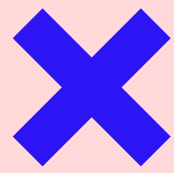
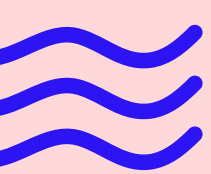
Classifier Algorithm	Optimal parameters	Precision	Accuracy	Recall	ROC_AUC
Logistic Regression	test_size=0.2, random_state=42	0.66828208 55614974	0.66783333 33333333	0.99800349 38857	0.50100877 15289011
Gaussian NB	Default	0.96171171 17117117	0.53716666 66666667	0.31969054 15522835	0.64705048 90400655
Decision Tree	Criterion=Gini, max_depth=10, max_leaf_nodes=10	0.73061586 17634028	0.67916666 66666667	0.82305964 56201648	0.60646208 57303031

	Unsampled	Oversampled	Undersampled
Logistic	Training Accuracy 0.69 Test Accuracy 0.68 Recall Score 0.95 ROC Score 0.53	Training Accuracy 0.63 Test Accuracy 0.62 Recall Score 0.66 ROC Score 0.61	Training Accuracy 0.63 Test Accuracy 0.63 Recall Score 0.67 ROC Score 0.62
Naive	Training Accuracy 0.68 Test Accuracy 0.67 Recall Score 0.97 ROC Score 0.52	Training Accuracy 0.53 Test Accuracy 0.53 Recall Score 0.98 ROC Score 0.51	Training Accuracy 0.53 Test Accuracy 0.53 Recall Score 0.99 ROC Score 0.50
Decision Tree	Training Accuracy 0.67 Test Accuracy 0.67 Recall Score 0.99 ROC Score 0.51	Training Accuracy 0.57 Test Accuracy 0.55 Recall Score 0.68 ROC Score 0.54	Training Accuracy 0.55 Test Accuracy 0.56 Recall Score 0.79 ROC Score 0.55

As we see in the Tabulation, Accuracy follows the order as follow:

Decision Tree › Logistic Regression › Gaussian Naive Bayes.

This were the results before we had done sampling and under-sampling.



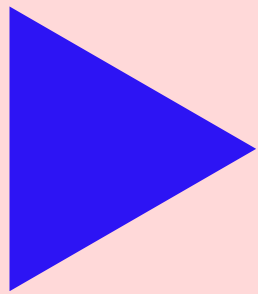
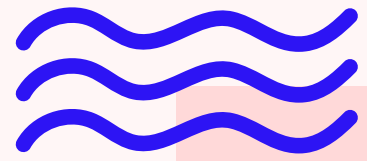
Logistic	Optimal Parameter	Accuracy	Recall	ROC
SVM	default	Training Accuracy 0.85 Test Accuracy 0.85	0.94	0.80
Random Forest	n_estimators=200, n_jobs = -1	Training Accuracy 0.87 Test Accuracy 0.86	0.93	0.81
MLP	random_state = 42, max_iter = 300	Training Accuracy 0.85 Test Accuracy 0.85	0.95	0.80

By looking at the result, we can say that Random Forest perform best among all the classifiers with Accuracy of 86%. As we see in the Tabulation that, Accuracy follows the order as follow:

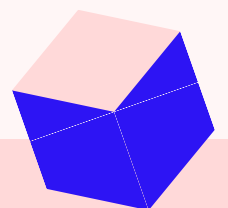
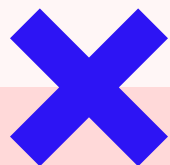
Random Forest › MLP › SVM

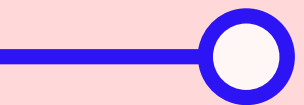


Conclusion



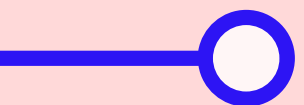
- **Learning**– Different ways to visualize the data for better understanding of features. Machine Learning models like Logistic Regression, Naive Bayes and Decision Tree to model the problem. How to use platforms like Kaggle and Google Colab. How to work and collaborate in teams.
- **Work Left**– We conducted study, but there is always opportunity for some improvement. We hope that our project will be useful in future research.





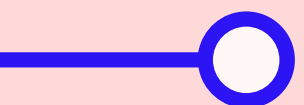
BIJENDAR PRASAD

Preprocessing, Training Model, Data Visualization,
Exploratory Data Analysis, Feature Selection, Result Analysis



ABHISHEK CHATURVEDI

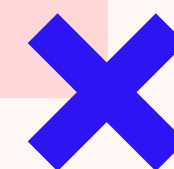
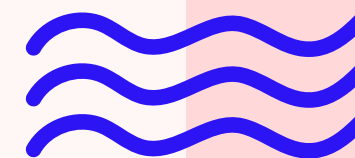
Data Preprocessing, Data Visualisation, Model Training, Model Testing, Report Writing, Model Tuning



DEEPESH KUMAR

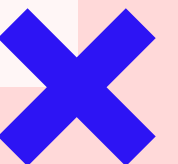
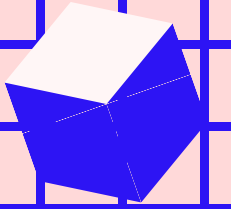
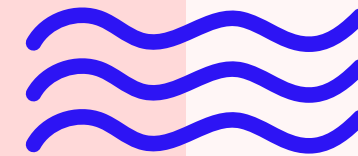
Model Testing, Data Preprocessing, Data Visualisation, Report Writing

Distribution of work among group members



References

- [1] Hareram Kumar (2022) The Research Paper, Android Malware Prediction using Machine Learning Techniques: A Review.
- [2] Neamat Al Sarah (2021) Online Paper, An Efficient Android Malware Prediction Using Ensemble machine learning algorithms
- [3] Machine Learning for Android Malware Detection Using Permission and API Calls
- [4] Dynamic Permissions based Android Malware Detection using Machine Learning Techniques
- [5] Android Permission Dataset





THANK YOU