# **Microsoft : Classifying Cybersecurity Incidents with Machine Learning**

Note ; Please Open this document in a google document for better visibility

## Understanding about the Dataset:

Since I was new to this domain it was challenging to understand the data at a first glance so what I decided is to understand each column one by one so I researched about the columns in the context of cybersecurity and after getting the detailed understanding about the columns and relationship between them I have started the data cleaning.

I have given an explanation for each column you can refer to in my Data cleaning jupiter notebook.

## Data Cleaning and Feature Extraction:

### Finding the null values:

In here there are few columns which have a null value around 90% filling those null values with the mode imputation won’t be a good option so I decided to drop those Null values.

Train\_dataset.drop(['Id','ActionGrouped', 'SuspicionLevel','AntispamDirection', 'LastVerdict', 'Roles', 'ResourceType', 'ThreatFamily', 'EmailClusterId', 'ActionGranular','ResourceIdName'],axis=1,inplace=True)

Train\_dataset.head()

### Filling the Null values:

In the column there are a few missing values so I have gone with the mode imputation method that too after grouped the column with the category so based on the column it will fill the missing values so there won’t be an issue based on which category it belongs to it will fill the missing value.

Train\_dataset['MitreTechniques'] = Train\_dataset.groupby('Category')['MitreTechniques'].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else "Unknown"))

### Grouping the Similar Content Columns:

In here I have found that there are some columns that have a similar kind of content so I thought I could combine all those columns together and make it as a single column.

So there are around 7 groups which combine 15 columns together.

**\*\*Combining the Columns having similar meaning and contining the grouped information\*\***

**\*\****\*Group 1\****\*\***

- AccountSid , AccountUpn , Account Object Id, Account Name ---> Account\_Info ====> Used for identifying specific users or accounts involved in incidents.

**\*\****\*Group 2\****\*\***

- DeviceId,Device Name,Network Message Id ----> Device\_Info =====> Used for identifying specific Device involved in incidents.

**\*\****\*Group 3\****\*\***

- RegistryKey,RegistoryValueData,RegistryValue Name -------> Registory\_Info ======> Indicates that the data involves registry keys and values.

**\*\****\*Group 4\****\*\***

- ApplicationId,ApplicationName ------> Application\_Info ======> Refers to the applications identified by their unique IDs and names.

**\*\****\*Group 5\****\*\***

- OsFamily,OsVersion -------> OS\_Info =======> Refers to the OS configuration of the system.

**\*\****\*Group 6\****\*\***

- State,City,CountryCode -------> Location\_info =========> Refers to the Location where the incident happened.

**\*\****\*Group 7\****\*\***

- Ipaddress,Url --------> Device\_Ip ========> Refers to the Device Locations with Ip and Url

Group 1

Train\_dataset['Account\_Info'] = (Train\_dataset['AccountSid'].astype(str) + ' ' +Train\_dataset['AccountUpn'].astype(str) + ' ' +Train\_dataset['AccountObjectId'].astype(str) + ' ' +Train\_dataset['AccountName'].astype(str))

Similarly I have done the same for all other columns.

### Dropping the Old Columns:

After combining all the required columns togethers I have dropped the old columns.

Train\_dataset.drop(['Url','IpAddress','CountryCode','City','FileName','FolderPath','State','OSVersion','OSFamily','ApplicationName','ApplicationId','RegistryValueData','RegistryValueName','RegistryKey','NetworkMessageId','DeviceName','DeviceId'],axis=1,inplace=True)

### Dropping the rows in Target Columns:

I have found there are 0.5% missing values in the target column. Instead of filling those missing values I have dropped the rows since it was in target for this situation. Dropping the columns won't impact the result much but filling can create some issue though I can drop the rows.

Train\_dataset = Train\_dataset.dropna(subset=['IncidentGrade'])

### Extracting the Time and Date from the Timestamp column:

Extracted the time and date so it will be useful for the model prediction.since there is no missing or any uncleaned rows.

# Convert the 'Timestamp' column to datetime

Train\_dataset['Timestamp'] = pd.to\_datetime(Train\_dataset['Timestamp'])

# Extract time, day, month, and year into new columns

Train\_dataset['Time'] = Train\_dataset['Timestamp'].dt.strftime('%H:%M:%S') # Extract time in HH:MM:SS format

Train\_dataset['Day'] = Train\_dataset['Timestamp'].dt.day

Train\_dataset['Month'] = Train\_dataset['Timestamp'].dt.month

Train\_dataset['Year'] = Train\_dataset['Timestamp'].dt.year

### Created a Part of day with the Time for analysis and model Training purpose:

From time i have splitted the day into the 5 different groups so that it will be easier for analysis and also for the model training purposes

Train\_dataset['Time'] = pd.to\_datetime(Train\_dataset['Time'], format='%H:%M:%S').dt.time

# Extract the hour from the time column

Train\_dataset['Hour'] = Train\_dataset['Time'].apply(lambda x: x.hour)

# Define conditions for categorization

conditions = [

(Train\_dataset['Hour'] < 6),

(Train\_dataset['Hour'] >= 6) & (Train\_dataset['Hour'] < 10),

(Train\_dataset['Hour'] >= 10) & (Train\_dataset['Hour'] < 16),

(Train\_dataset['Hour'] >= 16) & (Train\_dataset['Hour'] < 19),

(Train\_dataset['Hour'] >= 19)

]

# Define corresponding labels

labels = ['Early Morning', 'Morning', 'Afternoon', 'Evening', 'Night']

# Use np.select for vectorized categorization

Train\_dataset['PartOfDay'] = pd.Series(pd.cut(Train\_dataset['Hour'],

bins=[-1, 5, 9, 15, 18, 23],

labels=labels))

# Display the updated DataFrame

print(Train\_dataset[['Time', 'PartOfDay']])

And I have dropped the old columns.

Finally I have exported the cleaned File.

**Similar process was done for cleaning the test file. In addition there is a Usage column presented so I have dropped that.**

## EDA of Cleaned Train data to understand more about the data.

Since I am going to use incident grade as a Target I have analyzed all the columns with the incident grade to understand the relation with it.

### Which organization have a most number of incidents:

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Grouping the data as you did before

a = Cleaned.groupby(['OrgId', 'IncidentGrade']).count()

# Get the top 15 OrgId by frequency

top\_15\_orgs = Cleaned['OrgId'].value\_counts().nlargest(15).index

# Filter the DataFrame to include only rows with the top 15 OrgId

Cleaned\_top\_15 = Cleaned[Cleaned['OrgId'].isin(top\_15\_orgs)]

# Creating the count plot using Seaborn for the top 15 OrgId

plt.figure(figsize=(10, 6)) # Adjust the figure size if needed

sns.countplot(x='OrgId', hue='IncidentGrade', data=Cleaned\_top\_15,

order=top\_15\_orgs)

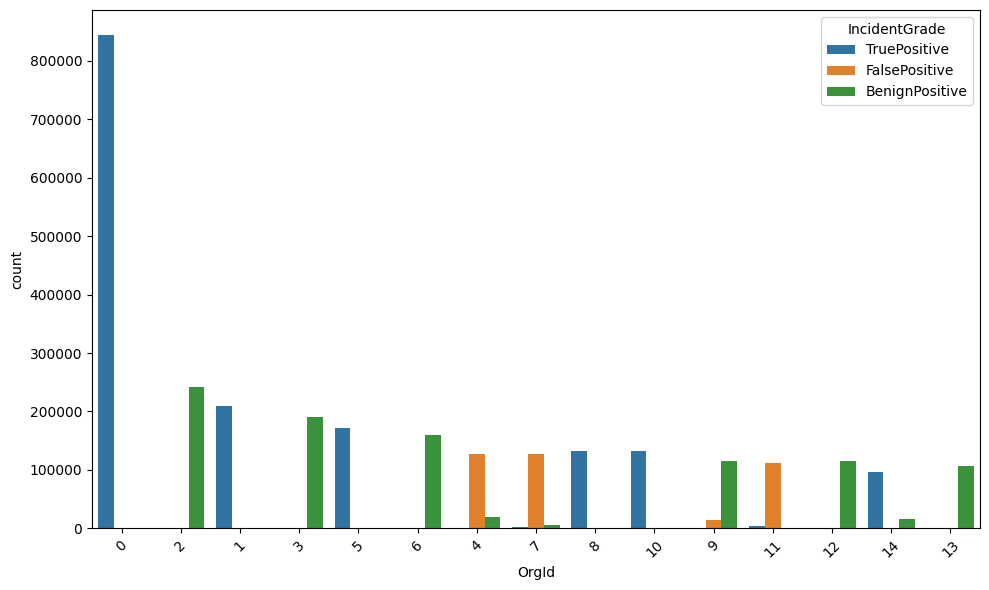
# Rotate the x-axis labels

plt.xticks(rotation=45)

# Display the plot

plt.tight\_layout()

plt.show()



### lets see which type of alert occurs most

import seaborn as sns

import matplotlib.pyplot as plt

# Get the top 15 AlertTitle by frequency

top\_15\_alerts = Cleaned['AlertTitle'].value\_counts().nlargest(15).index.tolist()

print(top\_15\_alerts)

# Filter the DataFrame to include only rows with the top 15 AlertTitle

Cleaned\_top\_15 = Cleaned[Cleaned['AlertTitle'].isin(top\_15\_alerts)]

# Create the count plot using Seaborn for the top 15 AlertTitle

plt.figure(figsize=(10, 6)) # Adjust the figure size if needed

sns.countplot(x='AlertTitle', data=Cleaned\_top\_15, order=top\_15\_alerts)

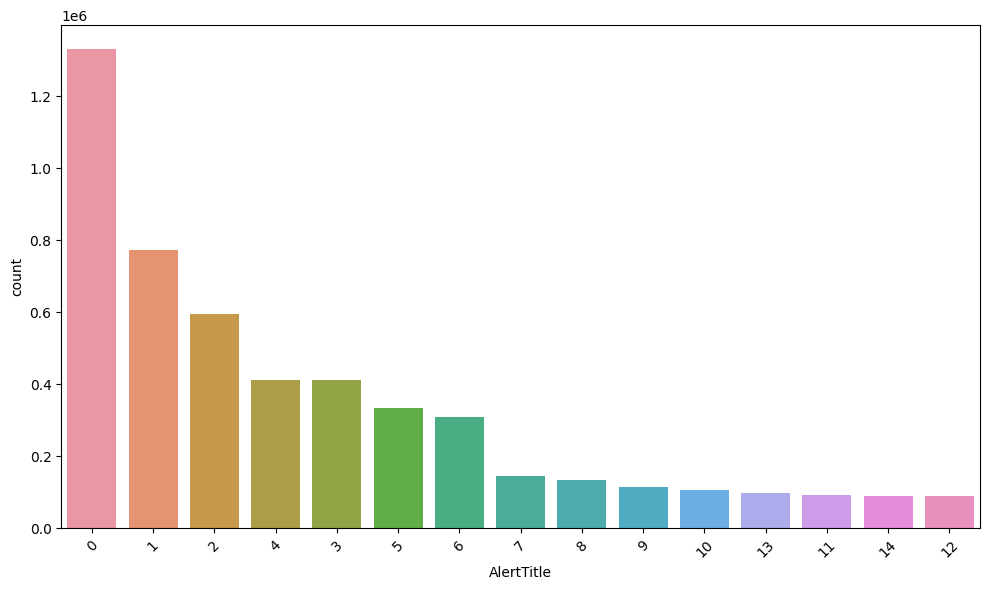
# Rotate the x-axis labels if needed

plt.xticks(rotation=45)

# Display the plot

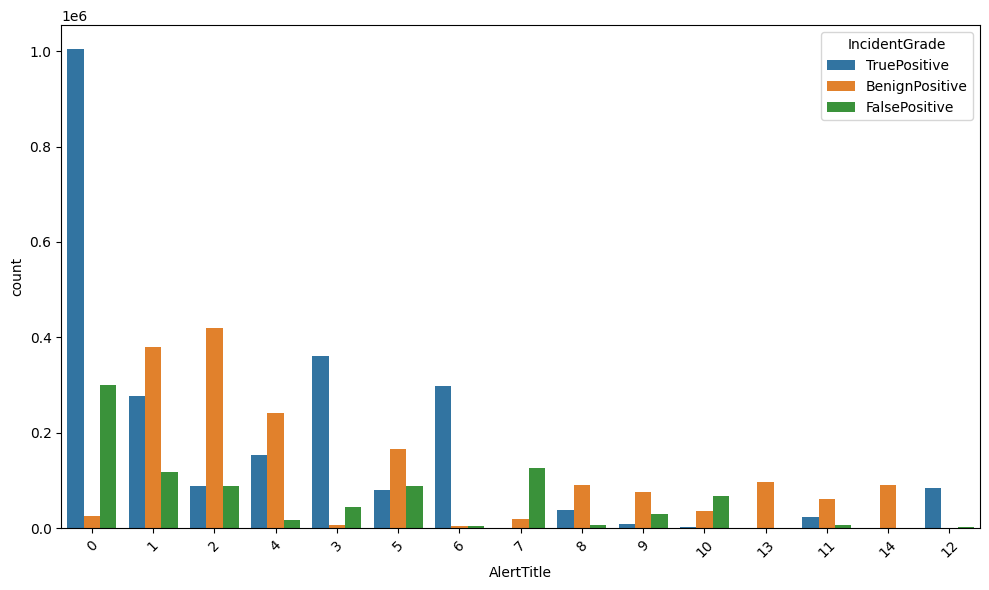
plt.tight\_layout()

plt.show()



### Let's check which type of alert Title has the most number of incident grades.

### The coding part is similar. I have changed the variable alone for getting the right result so I will Display only the Chart alone.



The Alerttitle of 0,1,2,4,3,6 are encountering the most number of the Threats so need more focus for this kind of Alert Title.Seems like this column will be useful in classification Task as well.

### Let Find which part of the day occur the most number of Incidents:

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Group by PartOfDay and IncidentGrade, and count occurrences

grouped\_data = Cleaned.groupby(['PartOfDay', 'IncidentGrade']).size().reset\_index(name='Count')

# Create a figure

plt.figure(figsize=(12, 6))

# Create a bar plot for counts of each IncidentGrade

bar\_plot = sns.barplot(x='PartOfDay', y='Count', hue='IncidentGrade', data=grouped\_data, alpha=0.6)

# Overlay a line plot for each IncidentGrade

for incident\_grade in grouped\_data['IncidentGrade'].unique():

subset = grouped\_data[grouped\_data['IncidentGrade'] == incident\_grade]

sns.lineplot(x='PartOfDay', y='Count', data=subset, marker='o', label=incident\_grade)

# Set labels and title

plt.xlabel('Part of Day')

plt.ylabel('Count')

plt.title('Count of Incident Grades by Part of Day')

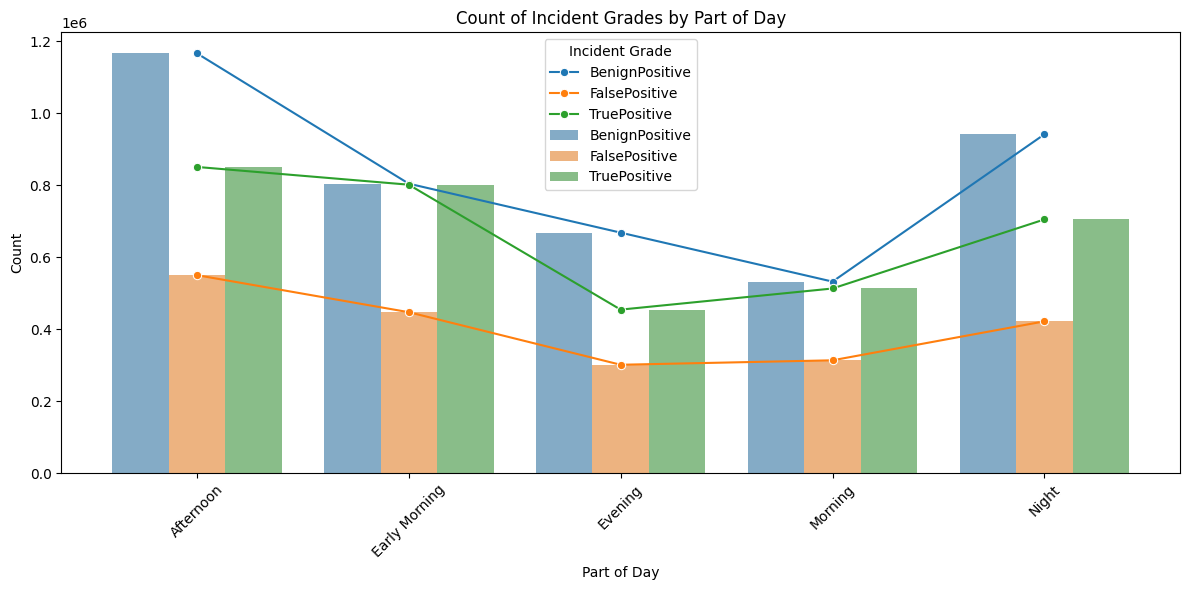
plt.legend(title='Incident Grade')

plt.xticks(rotation=45)

# Adjust the layout

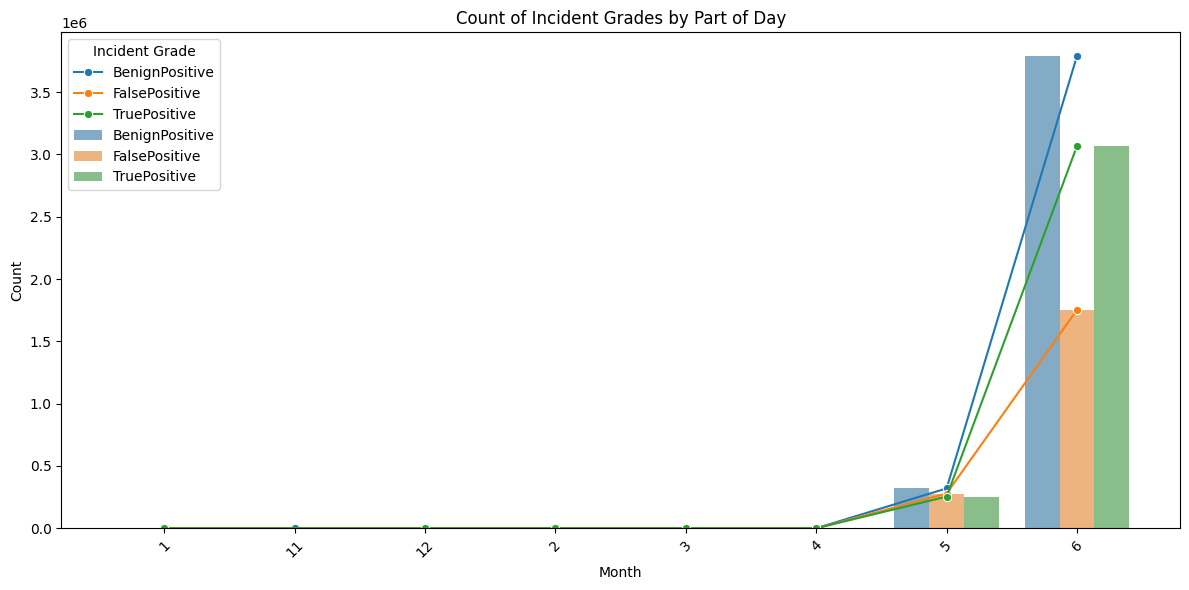
plt.tight\_layout()

plt.show()



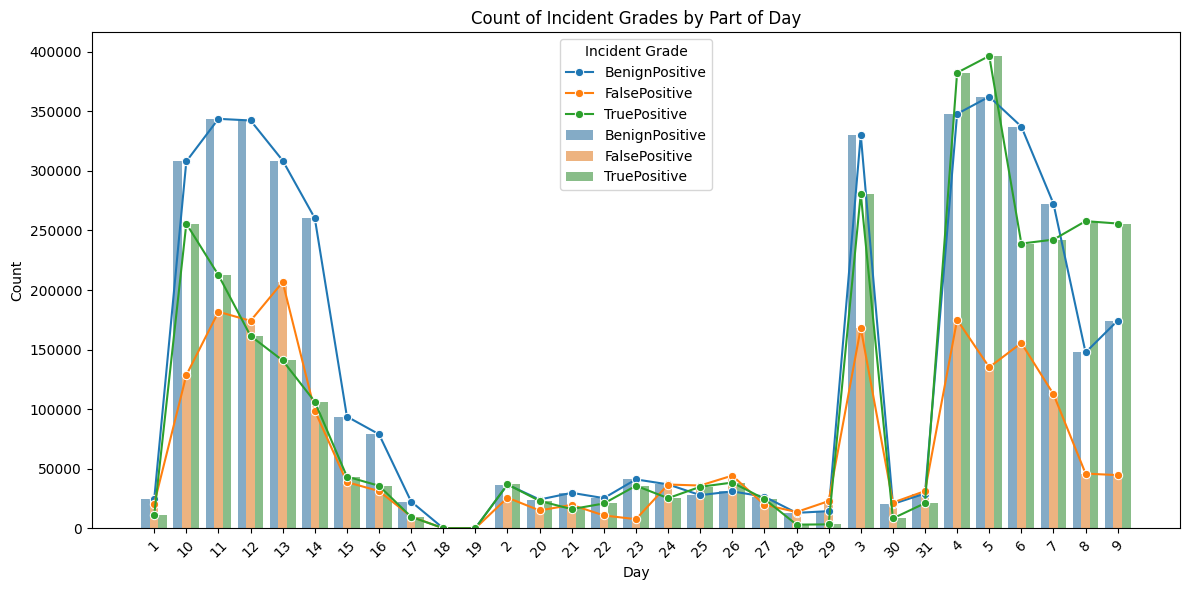
Seems Like in Afternoon,Early Morning and in Night the most number of the Threats are happening

### Lets see which month the most number of threats are happening:

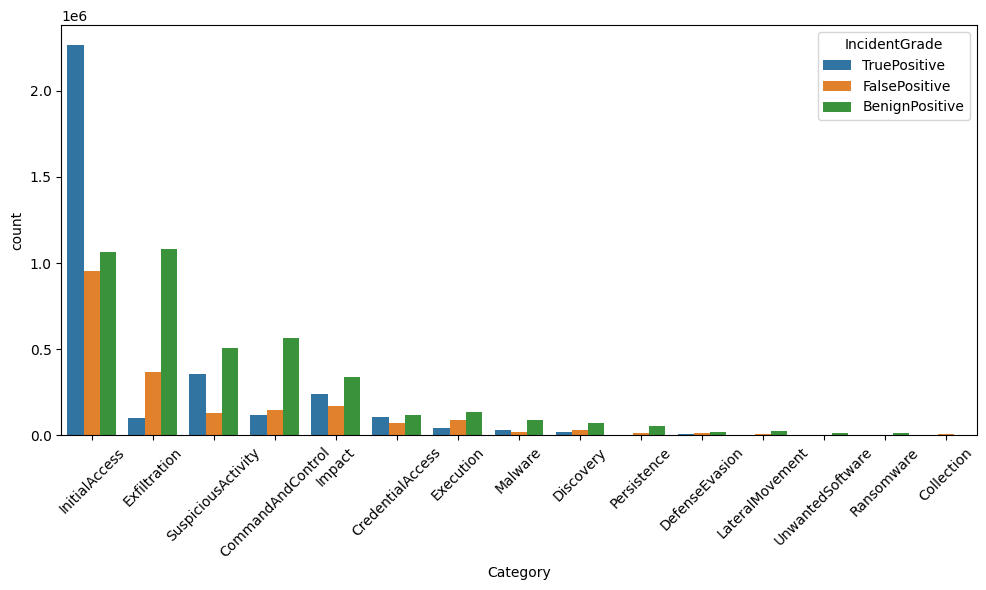


It shows in Months 6th and 5th only the most number of the incident is happening when compare to the other Months so need more attention to this thing

### Let Find Which day Likely there are more number of threats are happening:



### Which category is have a most number of threats



In categories like IntialAccess,Exfiltration,SuspiciousActivity,CommandAndControlImpact are encountering the most number of threads and especially in InitialAccess have the more True Positive when compared to other Categories.

### Lets see Which EntityType Have more number of threats

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Grouping the data as you did before

a = Cleaned.groupby(['EntityType', 'IncidentGrade']).count()

# Get the top 15 Category by frequency

top\_15\_orgs = Cleaned['EntityType'].value\_counts().nlargest(15).index

# Filter the DataFrame to include only rows with the top 15 Category

Cleaned\_top\_15 = Cleaned[Cleaned['EntityType'].isin(top\_15\_orgs)]

# Creating the count plot using Seaborn for the top 15 Category

plt.figure(figsize=(10, 6)) # Adjust the figure size if needed

sns.countplot(x='EntityType', hue='IncidentGrade', data=Cleaned\_top\_15,

order=top\_15\_orgs)

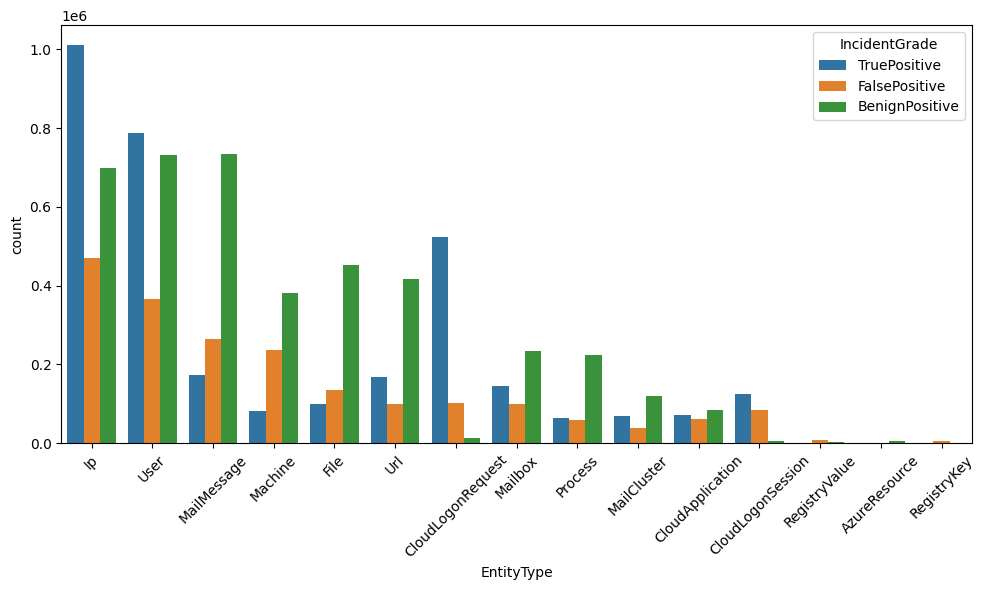
# Rotate the x-axis labels

plt.xticks(rotation=45)

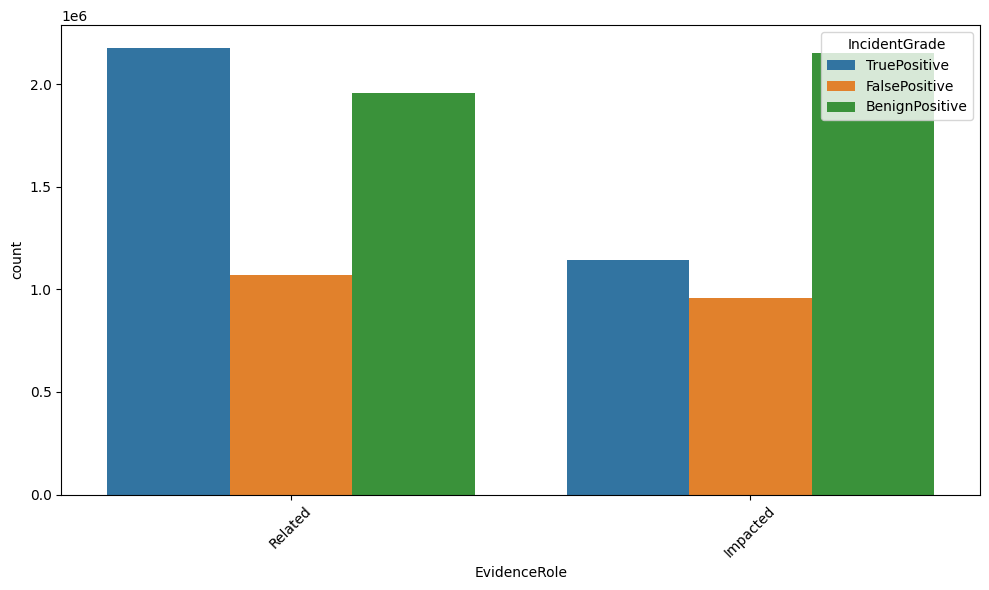
# Display the plot

plt.tight\_layout()

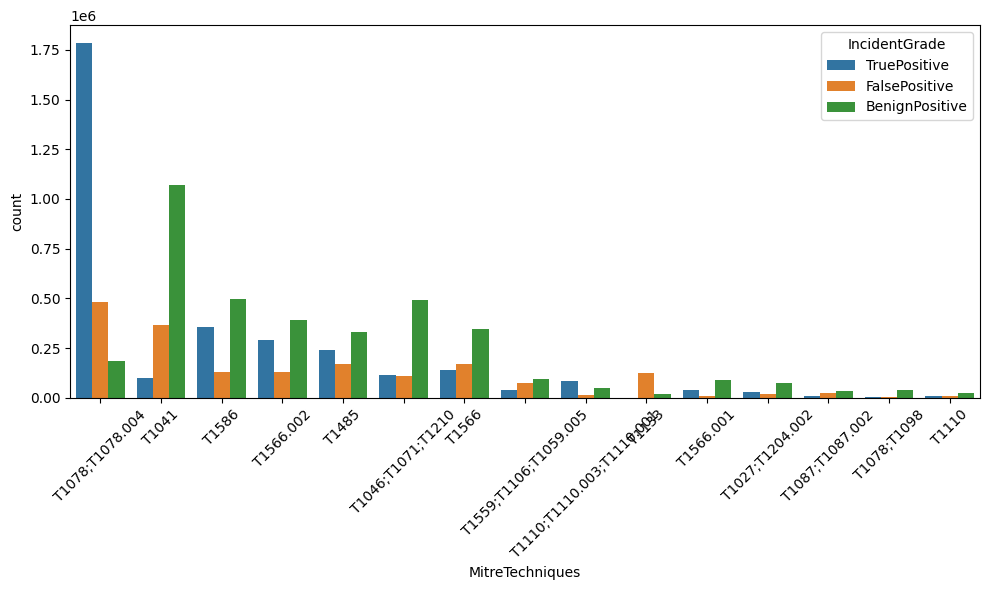
plt.show()



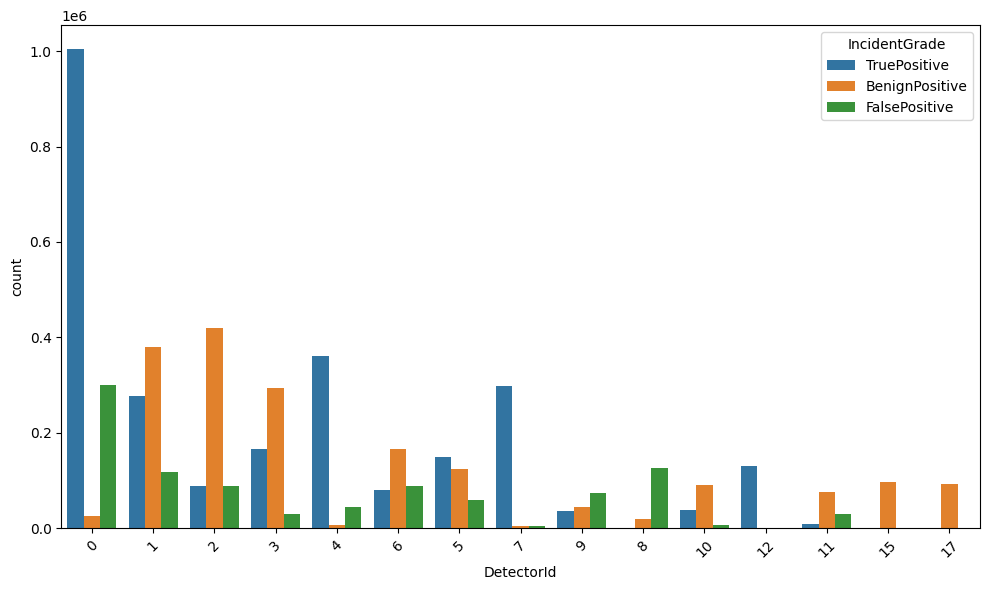
### Similarly do the same for the EvidenceRole:



### Let see which technique is mostly used for the threat creation:

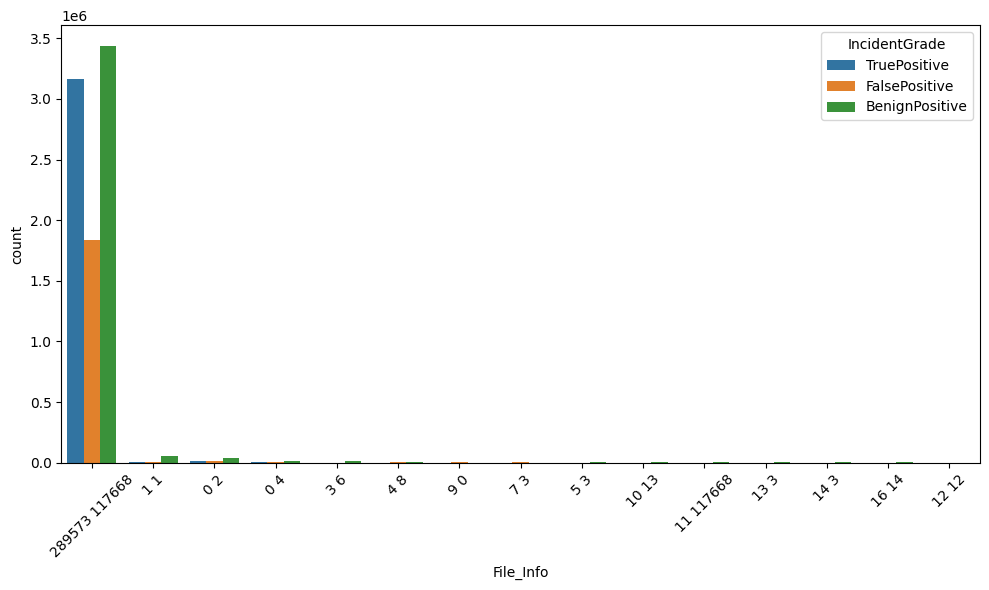


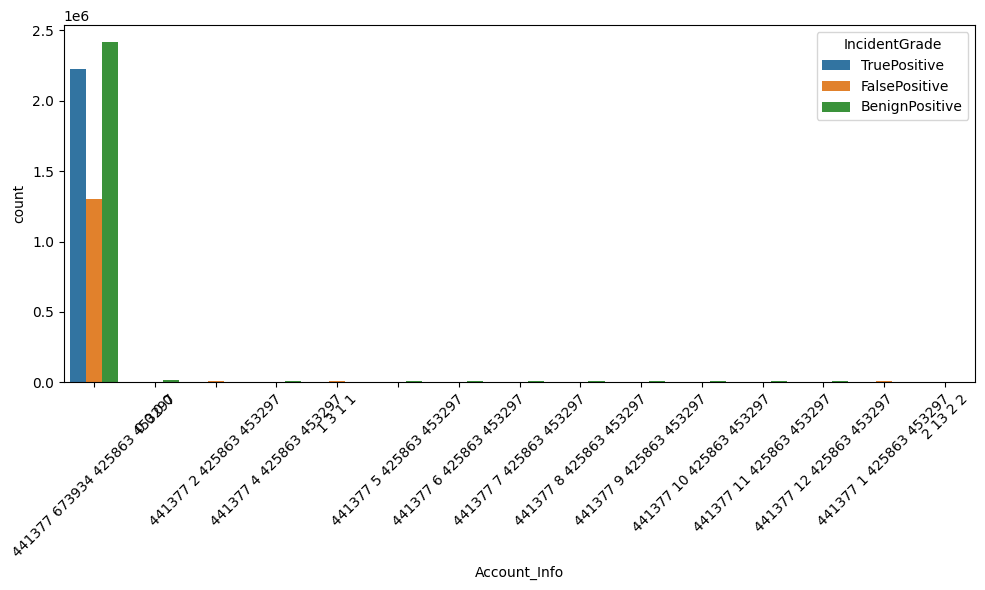
### Let's check the same for DetectorId:

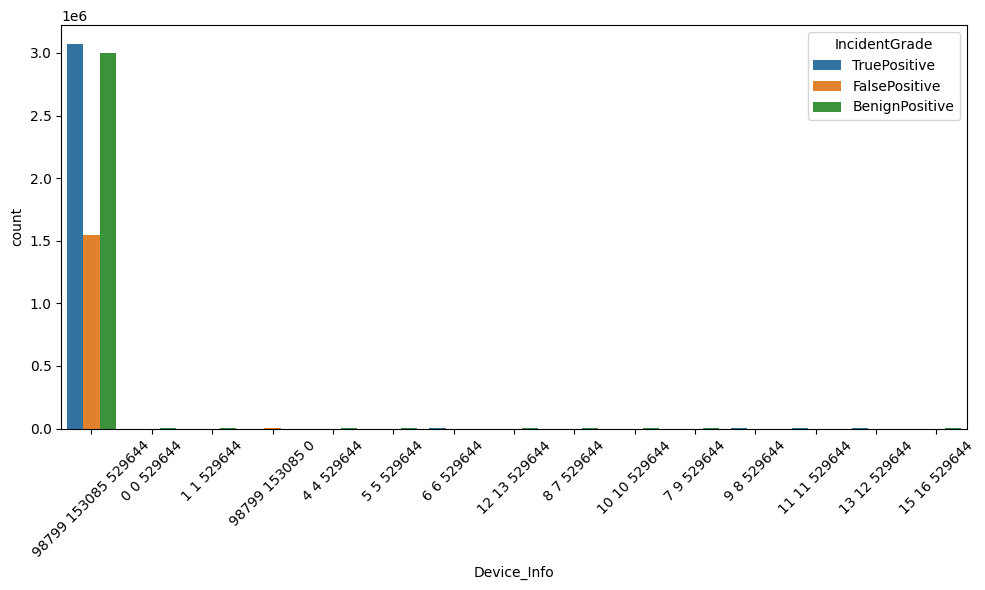


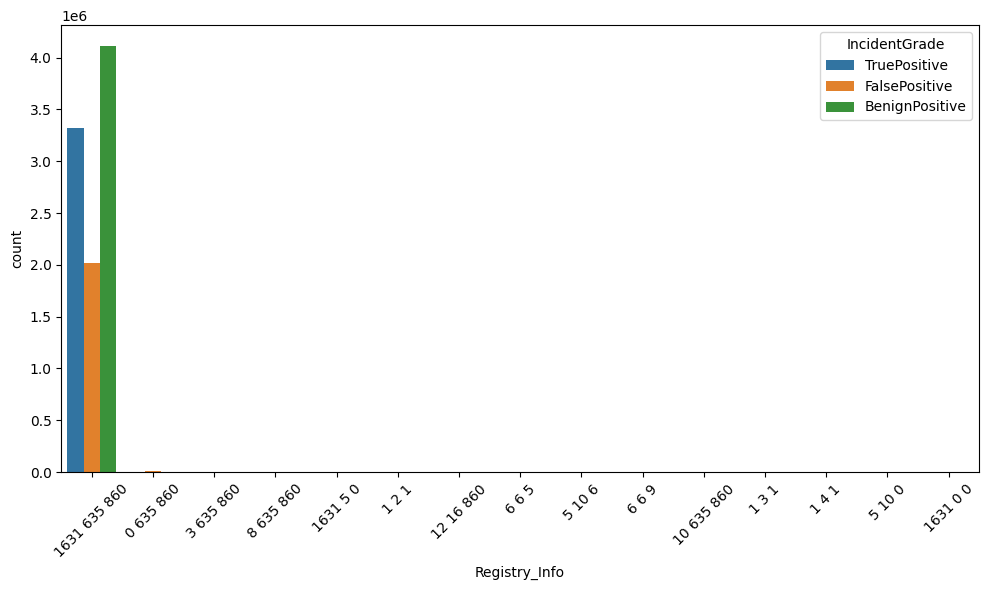
Seems like 0,4,1,7 are more use full for detecting the alert and this are getting more Threats

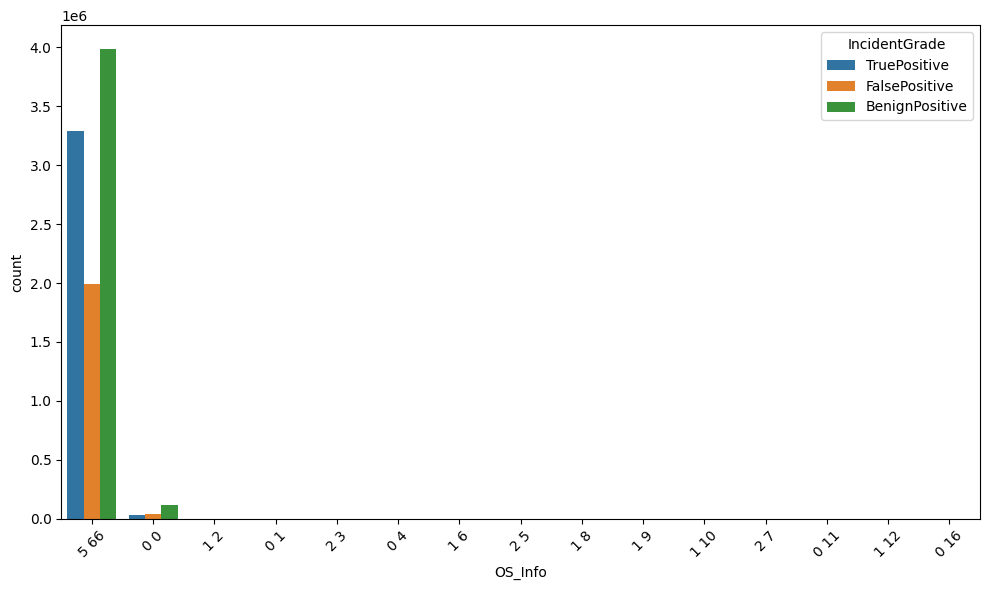
### Columns that completely rely on a single value in a row that was strongly related to the Incident Grade - It won’t be useful in classification since all are dependent on one single value.

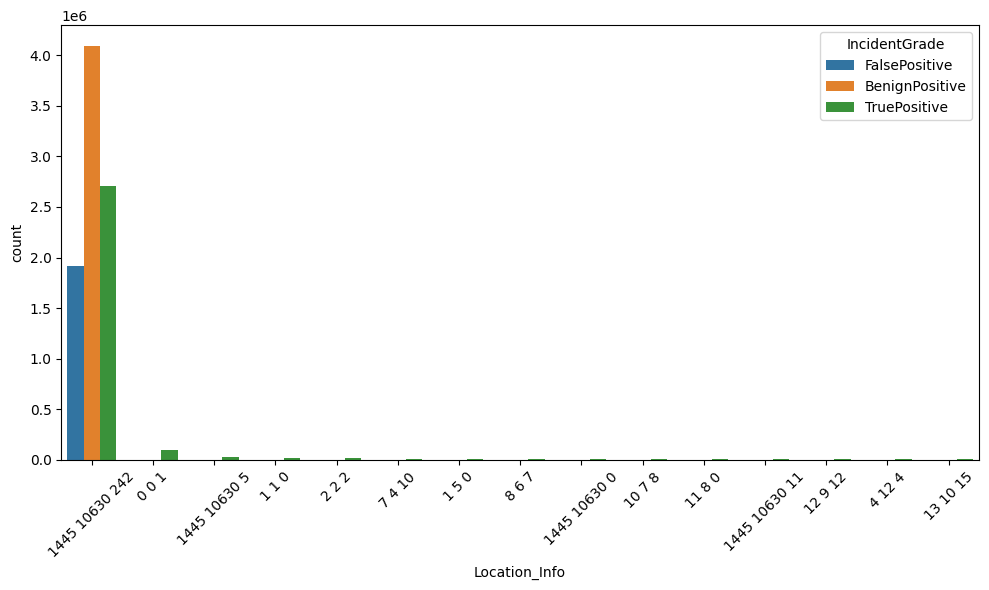


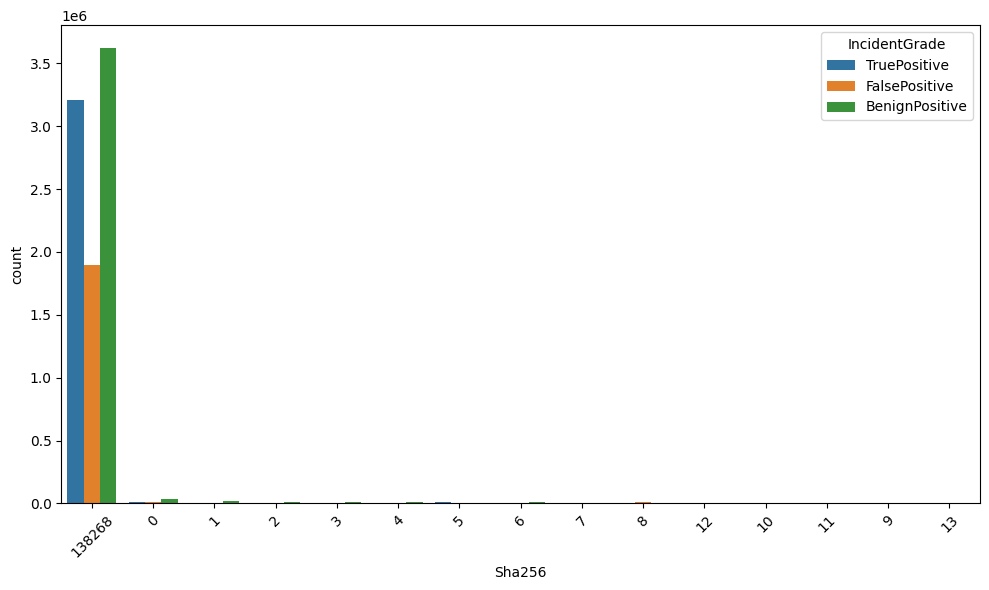


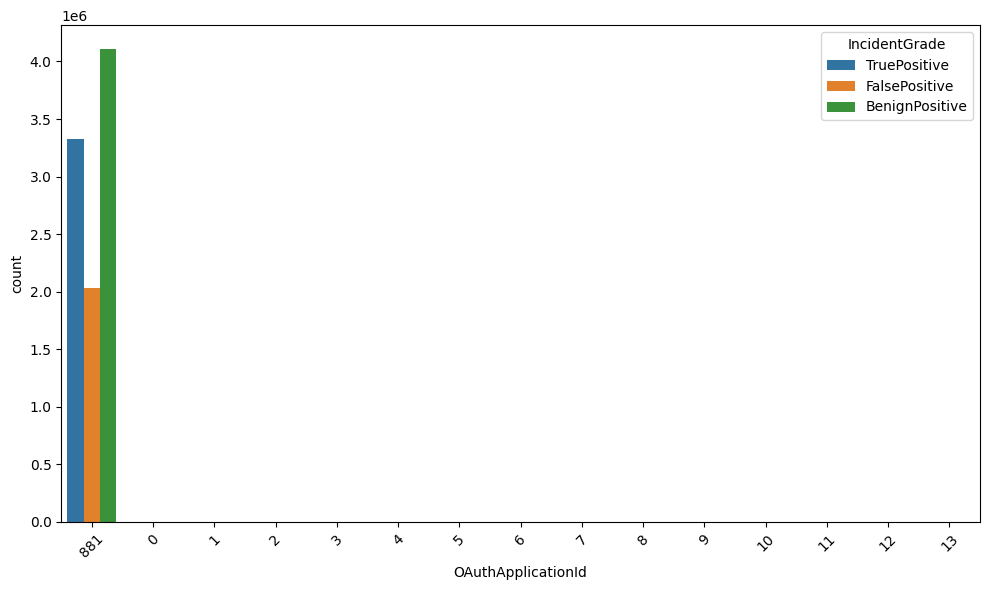












These columns are not useful for the classification task since they completely rely on one single group in a row.

Based on this I have decided to go with three different dataset for training my model and pick one best performing model.

## Model Development:

In Model development I have 3 different versions to train based on which model performs better in both train and test set. I will use that as a final Model.

## Version 1:

In this I have taken 9 columns for training the model for classifing the the Incident Grade.

Version1=Cleaned[['DetectorId','AlertTitle','Category','MitreTechniques','EntityType','EvidenceRole','Day','PartOfDay','IncidentGrade']]

Version1.head()

### Loading and Encoding the test and training data:

Ml\_Train=pd.read\_csv('/kaggle/input/ml-traindataset/ML\_Train\_Data.csv')

Ml\_Train.head()

Ml\_Test=pd.read\_csv('/kaggle/input/updated-test/Ml\_Test\_updated.csv')

Ml\_Test.head()

Encoding the train data and use that knowledge to encode the test data:

import pandas as pd

import pickle

# Categorical columns to be frequency encoded

frequency\_encoding\_columns = ['Category', 'MitreTechniques', 'EntityType', 'EvidenceRole']

# Frequency Encoding for ML\_Train

category\_mappings = {}

for column in frequency\_encoding\_columns:

# Get frequency distribution based on ML\_Train data

frequency\_map = Ml\_Train[column].value\_counts(normalize=True)

# Map frequencies to a new column in ML\_Train

Ml\_Train[column + '\_freq'] = Ml\_Train[column].map(frequency\_map)

# Store the frequency mappings to apply on ML\_Test later

category\_mappings[column] = frequency\_map.to\_dict()

# Drop the original categorical columns if needed from ML\_Train

# Ml\_Train.drop(columns=frequency\_encoding\_columns, inplace=True)

# Now, apply the same mappings to ML\_Test

for column in frequency\_encoding\_columns:

# Use the frequency mappings calculated from ML\_Train

freq\_col\_name = column + '\_freq'

Ml\_Test[freq\_col\_name] = Ml\_Test[column].map(category\_mappings[column])

# Fill any NaN values in case a category in ML\_Test is not found in ML\_Train

Ml\_Test[freq\_col\_name] = Ml\_Test[freq\_col\_name].fillna(0)

# Drop the original categorical columns if needed from ML\_Test

# Ml\_Test.drop(columns=frequency\_encoding\_columns, inplace=True)

# save the mappings to a pickle file for future use

with open(r'Freq\_enco\_mapping.pkl', 'wb') as f:

pickle.dump(category\_mappings, f)

print("Frequency encoding applied to ML\_Train and ML\_Test, and mappings saved to pickle file!")

# Import label encoder

from sklearn import preprocessing

a=['PartOfDay','IncidentGrade']

label\_encoder = preprocessing.LabelEncoder()

for i in a:

Ml\_Train[i]= label\_encoder.fit\_transform(Ml\_Train[i])

print(Ml\_Train[i].unique())

Ml\_Test[i]= label\_encoder.transform(Ml\_Test[i])

print(Ml\_Test[i].unique())

Since in 4 Columns there are more unique Categorical values and that are not ordinal as well so doing the Label encoding won't be a suitable and model can able to miscalculate that there is a rank based relation between the category present in the Columns And doing the One hot encoding also a effective or the reliable solution it will Increase the Dimension of the Data so I will Go with the Frequency Encoding

There some Ordinal Relationship between the category present in the PartofDay And the IncidentGrade So I will do a Label Encoding for those Columns

### Exporting the encoded test csv and Encoded Pkl file:

Ml\_Test.to\_csv('Ml\_Test\_Encoded.csv', index=False)

import pickle

# Save the model to a file using Pickle

with open('Encode.pkl', 'wb') as file:

pickle.dump(label\_encoder, file)

So the encoded file will be used for the future encoding if needed and the csv file will be useful for evaluating the model performance.

### Segregating the features and targets:

Features=Ml\_Train.drop('IncidentGrade',axis=1)

Features.head()

Target=Ml\_Train['IncidentGrade']

Target

### Splitting the dataset for train and test:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(Features,Target, test\_size=0.3,random\_state=42,stratify=Target)

In this I was gone with the standard splitting ratio 70 for training and 30 for testing and I have also used stratify for maintaining the equal ratio across all the classes in the target in train and test dataset.

### Scaling the features for giving the equal opportunity for all the features:

from sklearn.preprocessing import StandardScaler

# Standardize the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

### Scaled the features for both train and test dataset.

And pickle the file for future use

import pickle

# Save the model to a file using Pickle

with open('Scale1.pkl', 'wb') as file:

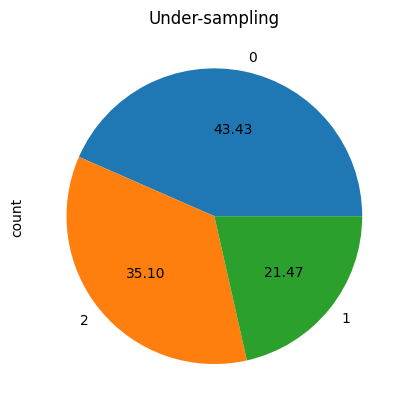
pickle.dump(scaler, file)

### Imbalance in the Target and I have gone with Undersampling.

y\_train.value\_counts()

ax = y\_train.value\_counts().plot.pie(autopct='%.2f')

\_ = ax.set\_title("Under-sampling")



Class 1 is comparatively less why I have gone with the undersampling means even though if i drop the records in the class 0 and class 2 there are still enough records for training my model and is this case it was suitable instead of creating the record that was not present dropping the existing records and make use of the remaining record won’t negatively impact the model.

from imblearn.under\_sampling import RandomUnderSampler

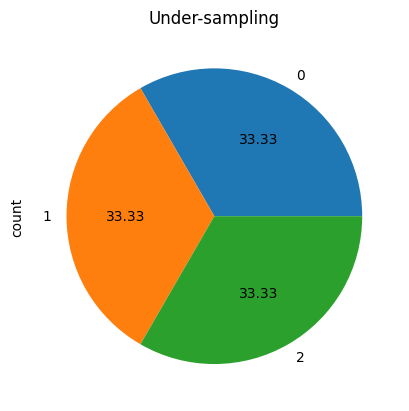
rus = RandomUnderSampler() # Numerical value

# rus = RandomUnderSampler(sampling\_strategy="not minority") # String

X\_train\_rus, y\_train\_rus = rus.fit\_resample(X\_train\_scaled, y\_train)

ax = y\_train\_rus.value\_counts().plot.pie(autopct='%.2f')

\_ = ax.set\_title("Under-sampling")



### Train the Model :

# Import required libraries

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import SGDClassifier

#from xgboost import XGBClassifier

# Initialize the models

model\_lr = LogisticRegression(random\_state=42)

model\_dt = DecisionTreeClassifier(random\_state=42)

model\_rf = RandomForestClassifier(random\_state=42, n\_estimators=100)

# Fit each model

model\_lr.fit(X\_train\_rus, y\_train\_rus)

model\_dt.fit(X\_train\_rus, y\_train\_rus)

model\_rf.fit(X\_train\_rus, y\_train\_rus)

I have used 7 models to train and picked the best 3 out of it.since the volume of the data is huge instead of using all the models at once i have used 3 models first and then used another 4 models.

model\_gb = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

model\_knn = KNeighborsClassifier(n\_neighbors=8)

model\_sgd = SGDClassifier(random\_state=42)

model\_xgb = XGBClassifier(random\_state=42)

model\_gb.fit(X\_train\_rus, y\_train\_rus)

model\_knn.fit(X\_train\_rus, y\_train\_rus)

model\_sgd.fit(X\_train\_rus, y\_train\_rus)

model\_xgb.fit(X\_train\_rus, y\_train\_rus)

### Evaluating and Validating the model:

# Import required metrics

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# List of models and their names

models = {

"Logistic Regression": model\_lr,

"Decision Tree": model\_dt,

"Random Forest": model\_rf,

"Gradient Boosting": model\_gb,

"K-Nearest Neighbors": model\_knn,

"SGD Classifier": model\_sgd,

"XGBoost": model\_xgb

}

# Loop through each model and evaluate

for name, model in models.items():

print(f"Evaluating {name}:")

# Predict on the test set

y\_pred = model.predict(X\_test\_scaled)

# Accuracy Score

acc = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {acc:.4f}")

# Classification Report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion Matrix

print("Confusion Matrix:")

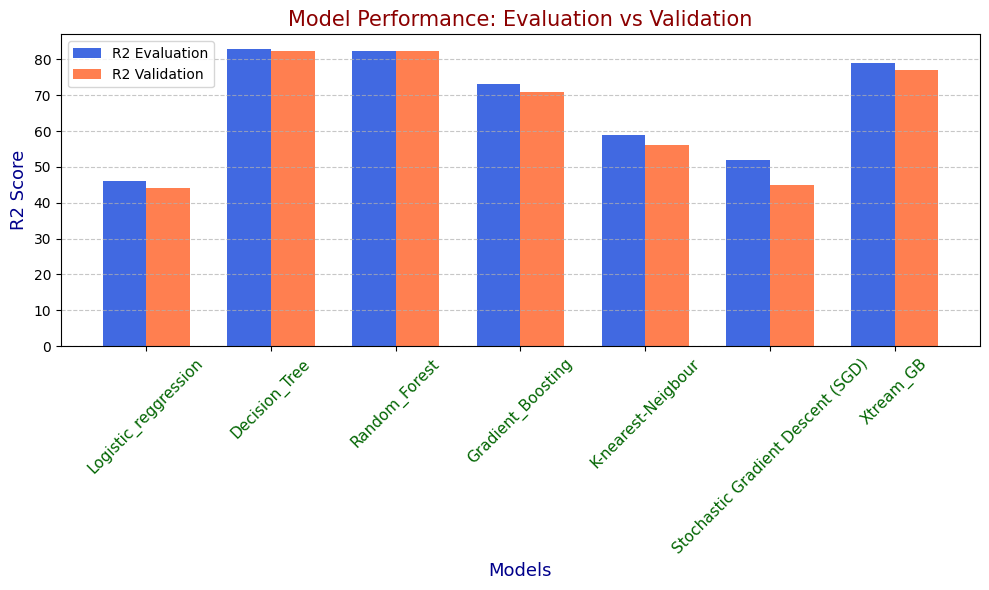
print(confusion\_matrix(y\_test, y\_pred))

print("\n" + "="\*60 + "\n")

Out of 7 models I got 3 models that perform well.

The best performing models are Decision Tree , Random Forest and Xtream GB are the well performing models for Decision tree and Random Forest I have achieved the accuracy of around 82% and for Xtream GB I have achieved 78%.

Even though it was a good accuracy but for classifying the threats accurately this is not enough so I have decide to do a feature importance and based on that I try to drop the less important feature and train the model again.

Let’s check the accuracy In Actual Test Dataset.

#### Encoded File:

Ml\_Test=pd.read\_csv('/kaggle/input/encoded-test/Ml\_Test\_Encoded.csv')

Ml\_Test.head()

Loading the encoded csv file for testing the trained model.

#### Scaling the Test File:

import pickle

import pandas as pd

# Load the scaler (assuming it is a scaler like StandardScaler or MinMaxScaler)

with open('/kaggle/input/new-test/Scale1.pkl', 'rb') as f:

scaler = pickle.load(f)

features =Ml\_Test[['DetectorId','AlertTitle','Category','MitreTechniques','EntityType','EvidenceRole','Day','PartOfDay']]

X\_test\_scaled = scaler.transform(features)

print(X\_test\_scaled)

#### Loading the Pretrained model:

import numpy as np

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Make predictions with the trained model

y\_pred = Model.predict(X\_test\_scaled)

# Calculate accuracy

accuracy = accuracy\_score(Target, y\_pred)

print(f"Accuracy Score: {accuracy:.2f}")

# Print classification report

report = classification\_report(Target, y\_pred)

print("Classification Report:\n", report)

# Print confusion matrix

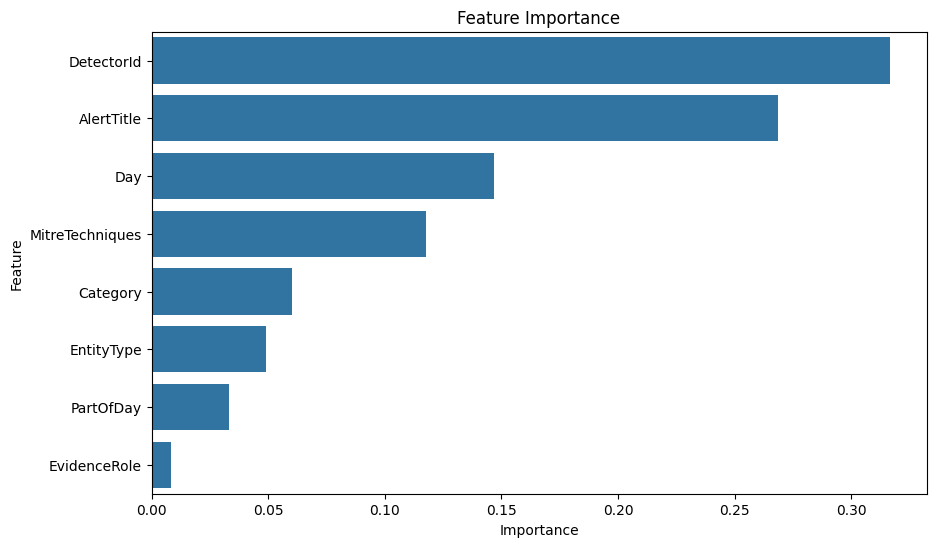
conf\_matrix = confusion\_matrix(Target, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

The accuracy of the model is 79 percent which is good but not for classification.

### Feature Importance:

By seeing this the EvidenceRole is not very important for the classification so I have decided to drop the Evidence Role and train the model again and check the models accuracy.



## 

## Version 2:

Process are same so I skip the explaining the individual steps and show the accuracy

### Loading and Encoding the test and training data:

Ml\_Train=pd.read\_csv('/kaggle/input/upgraded-version/ML\_Train\_Data.csv')

Ml\_Train.head()

import pandas as pd

import pickle

# Categorical columns to be frequency encoded

frequency\_encoding\_columns = ['Category', 'MitreTechniques', 'EntityType', 'EvidenceRole']

# Frequency Encoding for ML\_Train

category\_mappings = {}

for column in frequency\_encoding\_columns:

# Get frequency distribution based on ML\_Train data

frequency\_map = Ml\_Train[column].value\_counts(normalize=True)

# Map frequencies to a new column in ML\_Train

Ml\_Train[column + '\_freq'] = Ml\_Train[column].map(frequency\_map)

# Store the frequency mappings to apply on ML\_Test later

category\_mappings[column] = frequency\_map.to\_dict()

# Drop the original categorical columns if needed from ML\_Train

# Ml\_Train.drop(columns=frequency\_encoding\_columns, inplace=True)

# Now, apply the same mappings to ML\_Test

for column in frequency\_encoding\_columns:

# Use the frequency mappings calculated from ML\_Train

freq\_col\_name = column + '\_freq'

Ml\_Test[freq\_col\_name] = Ml\_Test[column].map(category\_mappings[column])

# Fill any NaN values in case a category in ML\_Test is not found in ML\_Train

Ml\_Test[freq\_col\_name] = Ml\_Test[freq\_col\_name].fillna(0)

# Drop the original categorical columns if needed from ML\_Test

# Ml\_Test.drop(columns=frequency\_encoding\_columns, inplace=True)

# save the mappings to a pickle file for future use

with open(r'Freq\_enco\_mapping.pkl', 'wb') as f:

pickle.dump(category\_mappings, f)

print("Frequency encoding applied to ML\_Train and ML\_Test, and mappings saved to pickle file!")

### Dropping the EvidenceRole

Features=Ml\_Train.drop(['IncidentGrade','EvidenceRole'],axis=1)

Features.head()

Target=Ml\_Train['IncidentGrade']

Target

### Splitting and standardizing the data

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(Features,Target, test\_size=0.3,random\_state=42,stratify=Target)

from sklearn.preprocessing import StandardScaler

# Standardize the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

### Doing the undersampling for the target make sure all class are equally contributing

from imblearn.under\_sampling import RandomUnderSampler

rus = RandomUnderSampler() # Numerical value

# rus = RandomUnderSampler(sampling\_strategy="not minority") # String

X\_train\_rus, y\_train\_rus = rus.fit\_resample(X\_train\_scaled, y\_train)

ax = y\_train\_rus.value\_counts().plot.pie(autopct='%.2f')

\_ = ax.set\_title("Under-sampling")

### Training the Model

# Import required libraries

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import SGDClassifier

#from xgboost import XGBClassifier

# Initialize the models

#model\_lr = LogisticRegression(random\_state=42)

model\_dt = DecisionTreeClassifier(random\_state=42)

model\_rf = RandomForestClassifier(random\_state=42, n\_estimators=100)

# Fit each model

#model\_lr.fit(X\_train\_rus, y\_train\_rus)

model\_dt.fit(X\_train\_rus, y\_train\_rus)

model\_rf.fit(X\_train\_rus, y\_train\_rus)

### Evaluating and validating

# Import required metrics

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# List of models and their names

models = {

#"Logistic Regression": model\_lr,

"Decision Tree": model\_dt,

"Random Forest": model\_rf,

#"Gradient Boosting": model\_gb,

#"K-Nearest Neighbors": model\_knn,

#"SGD Classifier": model\_sgd,

#"XGBoost": model\_xgb

}

# Loop through each model and evaluate

for name, model in models.items():

print(f"Evaluating {name}:")

# Predict on the test set

y\_pred = model.predict(X\_test\_scaled)

# Accuracy Score

acc = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {acc:.4f}")

# Classification Report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion Matrix

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\n" + "="\*60 + "\n")

### Accuracy of the Version 2

Even dropping the least important feature the model’s accuracy(78%) was not improved much so I have decided to go with the next version where I added the orgID and dropped the EvidenceRole.

## Version 3

I have followed the same process that I have done for the version 1 only difference is that I have taken out the evidence role and added the OrgId for improving the accuracy of the model.

So I am skipped the initial steps I directly explaining the accuracy of my model

## Train data

Evaluation scores - Random Forest —--> 96.63

Validation Scores - Random Forest —--> 98.65

### Decision Tree - Evaluation:

#### Accuracy Score

Decision Tree —> 98.09

#### Classification report

Classification Report:

precision recall f1-score support

0 0.99 0.98 0.99 1233246

1 0.96 0.98 0.97 609590

2 0.98 0.98 0.98 996814

accuracy 0.98 2839650

macro avg 0.98 0.98 0.98 2839650

weighted avg 0.98 0.98 0.98 2839650

All the classes are contributed equally to the accuracy and macro avg, f1-score,precision.

#### Confusion matrix

[[1207640 13975 11631]

[ 5044 599601 4945]

[ 5814 12859 978141]]

The False Positive and the False Negative are very less and the true positive and True Negative are also predicting perfect

Class 2 —--> I considered as a True Positive

Class 0 and 1 —--> considered as a False Negative in confusion matrix

### Decision Tree - Validation.

#### Accuracy Score

Decision Tree —> 98.65

#### Classification report

Classification Report:

precision recall f1-score support

0 0.99 0.98 0.99 1422377

1 0.98 0.99 0.99 1422377

2 0.99 0.99 0.99 1422377

accuracy 0.99 4267131

macro avg 0.99 0.99 0.99 4267131

weighted avg 0.99 0.99 0.99 4267131

All the classes are contributed equally to the accuracy and macro avg, f1-score,precision.

#### Confusion matrix

[[1399984 11705 10688]

[ 7961 1406757 7659]

[ 5527 13991 1402859]]

The False Positive and the False Negative are very less and the true positive and True Negative are also predicting perfect

Class 2 —--> I considered as a True Positive

Class 0 and 1 —--> considered as a False Negative in confusion matrix

### Random Forest - Evaluation:

#### Accuracy Score

Accuracy 96.63

#### Classification Report

Classification Report:

precision recall f1-score support

0 0.98 0.96 0.97 1233246

1 0.93 0.97 0.95 609590

2 0.97 0.97 0.97 996814

accuracy 0.97 2839650

macro avg 0.96 0.97 0.96 2839650

weighted avg 0.97 0.97 0.97 2839650

All the classes are contributed equally to the accuracy and macro avg, f1-score,precision.but it was not accurate as the decision tree model

#### Confusion matrix

[[1188339 24740 20167]

[ 8674 591372 9544]

[ 12827 19672 964315]]

The False Positive and the False Negative are very less but not good as decision tree model and the true positive and True Negative are also predicting perfect

Class 2 —--> I considered as a True Positive

Class 0 and 1 —--> considered as a False Negative in confusion matrix

### Random Forest - Validation

#### Accuracy Score

Accuracy —--> 98.65

The accuracy score was similar for both decision tree and random forest.

#### Classification Report

Classification Report:

precision recall f1-score support

0 0.99 0.98 0.99 1422377

1 0.98 0.99 0.99 1422377

2 0.99 0.99 0.99 1422377

accuracy 0.99 4267131

macro avg 0.99 0.99 0.99 4267131

weighted avg 0.99 0.99 0.99 4267131

All the classes are contributed equally to the accuracy and macro avg, f1-score,precision.

#### Confusion matrix

[[1398601 12678 11098]

[ 6970 1406599 8808]

[ 5111 12885 1404381]]

Finally the both models are performing well so I will keep the both models since random forest has several trees the model size is bigger for space efficiency we can use decision trees if there are no space constraints we can go with random forest.

## Test Data

### Random Forest Model

#### Accuracy Score

The accuracy score on unseen data is 91%

#### Classification Report

Classification Report:

precision recall f1-score support

0 0.94 0.93 0.93 1751475

1 0.84 0.89 0.86 902345

2 0.94 0.91 0.92 1492221

accuracy 0.91 4146041

macro avg 0.90 0.91 0.91 4146041

weighted avg 0.92 0.91 0.92 4146041

#### Confusion matrix

[[1631830 72511 47134]

[ 57938 799102 45305]

[ 51716 78625 1361880]]

### Decision Tree Model

#### Accuracy Score

The accuracy score on unseen data is 93%

#### Classification Report

Classification Report:

precision recall f1-score support

0 0.96 0.94 0.95 1751475

1 0.88 0.90 0.89 902345

2 0.94 0.95 0.94 1492221

accuracy 0.93 4146041

macro avg 0.93 0.93 0.93 4146041

weighted avg 0.93 0.93 0.93 4146041

The scope wise this model performs better when compared to the random forest model.It was perfectly predicting the Threats.

#### Confusion matrix

[[1646692 60337 44446]

[ 42520 815115 44710]

[ 28422 51618 1412181]]

Decision tree model performs one step well even in unseen data.

## How model can be implemented in Soc Workflow:

The final mode have 8 Features

OrgID,DectorID,AlertTitle,category,MitreTechniques,EntityType,Day,partofDay are the features used for the specifically identifies the thread is serious or not so serious so based on that we can able to classify the incident grade.

This feature contains both the organizational and customer related groups which makes the more robust model for the classification task.

Implementation is so simple the security system will have a track which features have which kind of threat category so based on the category the system is able to predict the class and the security will block those kinds of entries.So based on this the potential risk can be reduced.

## Conclusion

So all three models are Performing well Since the Random Forest and XGBoost accuracy, F1 score and the True positive score is also higher than the Decision Tree Model. We can Use This for the Future Purposes.