In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import StandardScaler


# Phase 1: Data Collection (e.g., UGRansome dataset)

pd.set_option("expand_frame_repr", False)
df= pd.read_csv('/kaggle/input/ugransome-dataset/final(2).csv')
df2 = pd.DataFrame(df)
df2.columns = ['Time','Protocol','Flag','Family','Clusters','SeedAddress','E
df2
```

Out[1]:

	Time	Protocol	Flag	Family	Clusters	SeedAddress	ExpAddress
0	50	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
1	40	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
2	30	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
3	20	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
4	57	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
...	...	...	...	...	...	...	...
149038	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149039	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149040	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149041	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149042	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ

149043 rows × 14 columns

In [2]:

```
# Data cleaning
# Renaming the attack "Bonet" to "Botnet"

df2['Threats'] = df2['Threats'].str.replace('Bonet', 'Botnet')

# Print the modified DataFrame
df2
```

Out[2]:

	Time	Protocol	Flag	Family	Clusters	SeedAddress	ExpAddress
0	50	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
1	40	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
2	30	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
3	20	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
4	57	TCP	A	WannaCry	1	1DA11mPS	1BonuSr7
...	...	...	...	...	...	...	...
149038	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149039	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149040	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149041	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ
149042	33	UDP	AP	TowerWeb	3	1AEoiHYZ	1SYSTEMQ

149043 rows × 14 columns

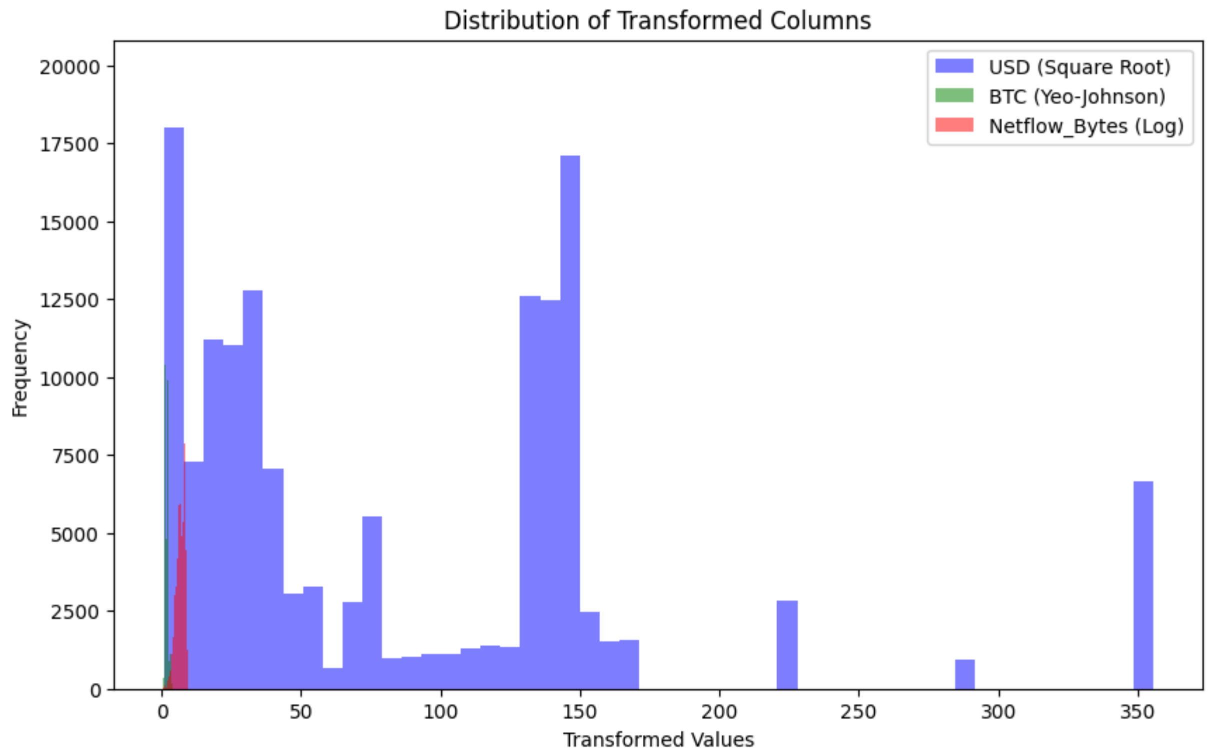
In [3]:

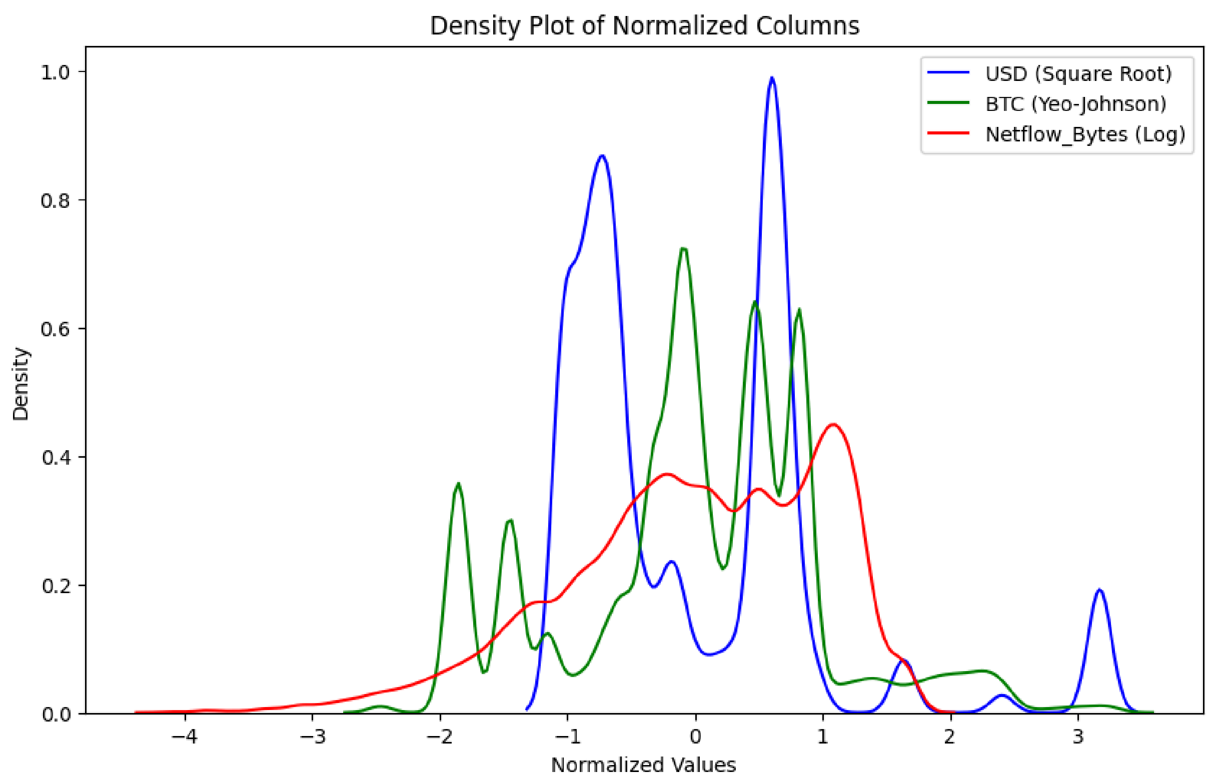
```
# Phase 2: Data Preparation (feature engineering and data transformation)

# --- Drop all duplicate rows --- #

df2 = df2.drop_duplicates()

# --- Remove negative values from time/timestamp feature --- #

df2['Time'] = df2['Time'] + 11

# adding 11 to each value in the 'Time' column of the DataFrame 'df2'.
#In other words, it's performing an element-wise addition operation on all t
#increasing each value by 11 units. This is often done in data manipulation
#by a fixed amount

# --- Math transformations to reduce skewness --- #

# --- Log transformation applied to column NETFLOW_BYTES --- #
# A log transformation involves taking the natural logarithm (base e) of eac
#Logarithmic transformations are often used to reduce the impact of extreme
#closely to a normal distribution. They are particularly useful when dealing
#where the tail of the distribution is elongated on the right side.


#The np.log() function is a common way to perform a logarithmic transformati
#The + 1 added to the data points is often used to avoid issues with taking
#It's a common practice to add a small constant like 1 to the data before ap

#By applying a log transformation to a feature, you're essentially compressi
#which can help in cases where the data exhibits a rightward skew, making it
#or modeling techniques that assume normally distributed data.
```

Loading [MathJax]/extensions/Safe.js

```
df2['Netflow_Bytes'] = np.log(df2['Netflow_Bytes']+1)



# --- Square root transformation applied to columns USD ---#


#Square Root Transformation: A square root transformation involves taking th
#specified column. In this case, it's applied to the 'USD' column.

#Square root transformations are a type of mathematical transformation used
#Just like logarithmic transformations, square root transformations can help
#a normal distribution.

#The np.sqrt() function is used to calculate the square root.
#By applying a square root transformation to the 'USD' column, the code is a
#and more suitable for certain statistical analyses or modeling techniques t
#require data to be more symmetric. It's a common technique used in data pre
#analysis or modeling

df2['USD'] = np.sqrt(df2['USD'])

# --- Yeo Johnson transformation applied to columns BTC--#


#Yeo-Johnson transformation is being applied to the 'BTC' column in the Data
#This transformation is used to modify the data in the 'BTC' column to make
#The Yeo-Johnson transformation is a mathematical transformation technique u
#It can be applied to both positive and negative values and is more versatil

#The transformation is performed using the stats.yeojohnson() function from
df2['BTC'], _ = stats.yeojohnson(df2['BTC'])


#--PLOTING TRANSFORMED DATA--#

fig, ax = plt.subplots(figsize=(10, 6))

# Plot the transformed 'USD' column
ax.hist(df2['USD'], bins=50, alpha=0.5, color='blue', label='USD (Square Roo

# Plot the transformed 'BTC' column
ax.hist(df2['BTC'], bins=50, alpha=0.5, color='green', label='BTC (Yeo-Johns

# Plot the transformed 'Netflow_Bytes' column
ax.hist(df2['Netflow_Bytes'], bins=50, alpha=0.5, color='red', label='Netflo

# Add labels and a legend
ax.set_xlabel('Transformed Values')
ax.set_ylabel('Frequency')
ax.set_title('Distribution of Transformed Columns')
ax.legend()

# Show the plot
plt.show()
```

```
# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(10, 6))

# Create a StandardScaler instance
# The StandardScaler is a common preprocessing technique used in machine lea
#It is used to standardize or normalize the features of a dataset by scaling
#deviation of 1.


#Standardizing the features is useful because it makes different features mo
#that are sensitive to the scale of the input data, such as many machine lea
#In the code provided, scaler is created as an instance of the StandardScale
#the specified columns in the df2 DataFrame using the fit_transform method,

scaler = StandardScaler()

# Normalize each column's features
df2_normalized = df2.copy()
df2_normalized[['USD', 'BTC', 'Netflow_Bytes']] = scaler.fit_transform(df2[[

# Plot the density of the normalized 'USD' column
sns.kdeplot(df2_normalized['USD'], color='blue', label='USD (Square Root)',

# Plot the density of the normalized 'BTC' column
sns.kdeplot(df2_normalized['BTC'], color='green', label='BTC (Yeo-Johnson)',

# Plot the density of the normalized 'Netflow_Bytes' column
sns.kdeplot(df2_normalized['Netflow_Bytes'], color='red', label='Netflow_Byt

# Add labels and a legend
ax.set_xlabel('Normalized Values')
ax.set_ylabel('Density')
ax.set_title('Density Plot of Normalized Columns')
ax.legend()

# Show the plot
plt.show()
```





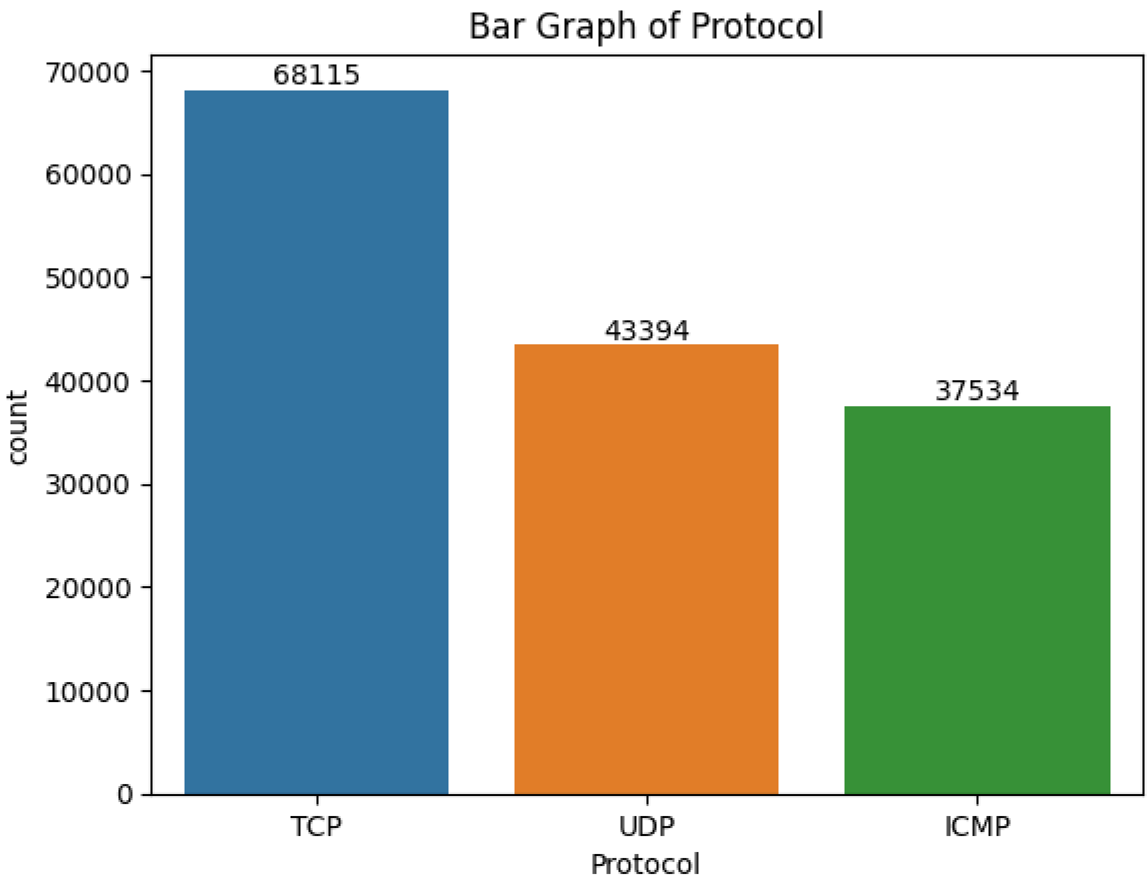
In [4]:
```
# Phase 3: Data Visualization

# --- Count visualizations --- #

# Categorical count visualizations

# Protocol count

                                countplot(x=df2['Protocol'], data=df2)
```

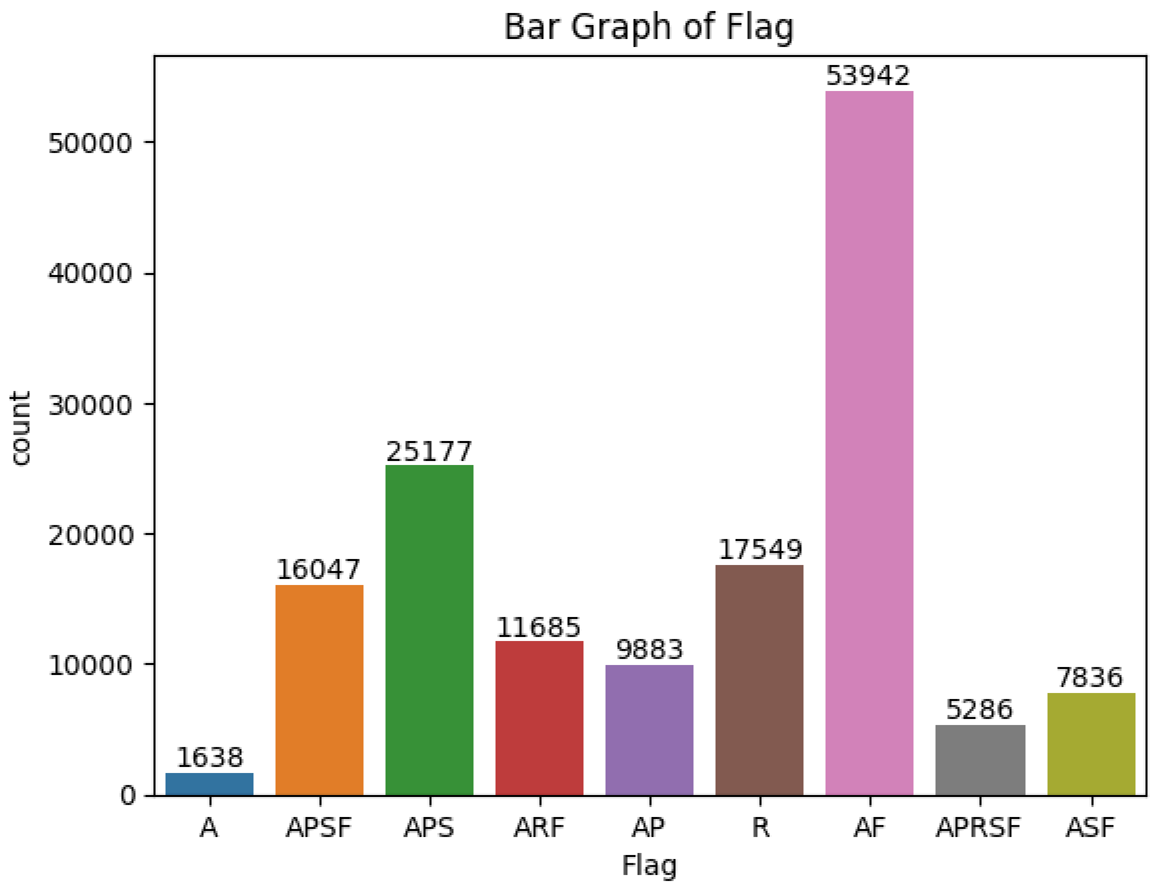
Loading [MathJax]/extensions/Safe.js

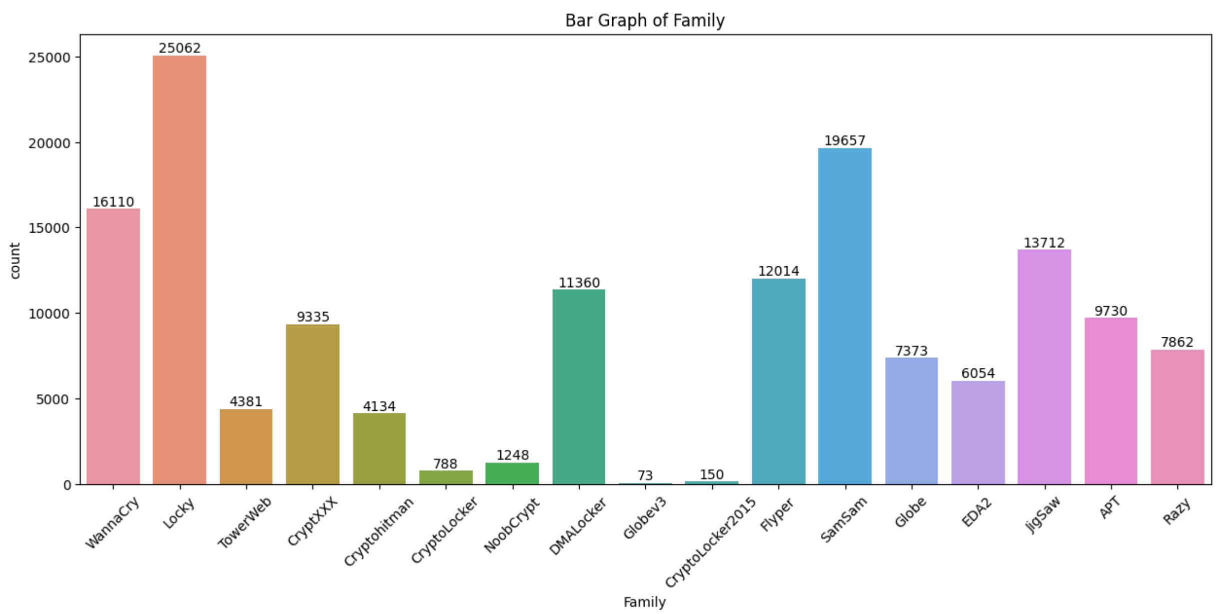
```
plt.title('Bar Graph of Protocol')

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# Flag count

ax = sns.countplot(x=df2['Flag'], data=df2)
plt.title('Bar Graph of Flag')

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# Family count

plt.figure(figsize=(15, 6))
ax = sns.countplot(x=df2['Family'], data=df2)
plt.title('Bar Graph of Family')
plt.xticks(rotation=45)
plt.xticks(fontsize=10)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# Clusters count

ax = sns.countplot(x=df2['Clusters'], data=df2)
plt.title('Bar Graph of Clusters')

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# SeedAddress count

ax = sns.countplot(x=df2['SeedAddress'], data=df2)
plt.title('Bar Graph of SeedAddress')
plt.xticks(rotation=45)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
```

```
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# ExpAddress count

ax = sns.countplot(x=df2['ExpAddress'], data=df2)
plt.title('Bar Graph of ExpAddress')
plt.xticks(rotation=45)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# IPaddress count

ax = sns.countplot(x=df2['IPaddress'], data=df2)
plt.title('Bar Graph of IPaddress')

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# Threats count

ax = sns.countplot(x=df2['Threats'], data=df2)
plt.title('Bar Graph of Threats')
plt.xticks(rotation=45)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# Port count

ax = sns.countplot(x=df2['Port'], data=df2)
plt.title('Bar Graph of Port')

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()

# Prediction count
```

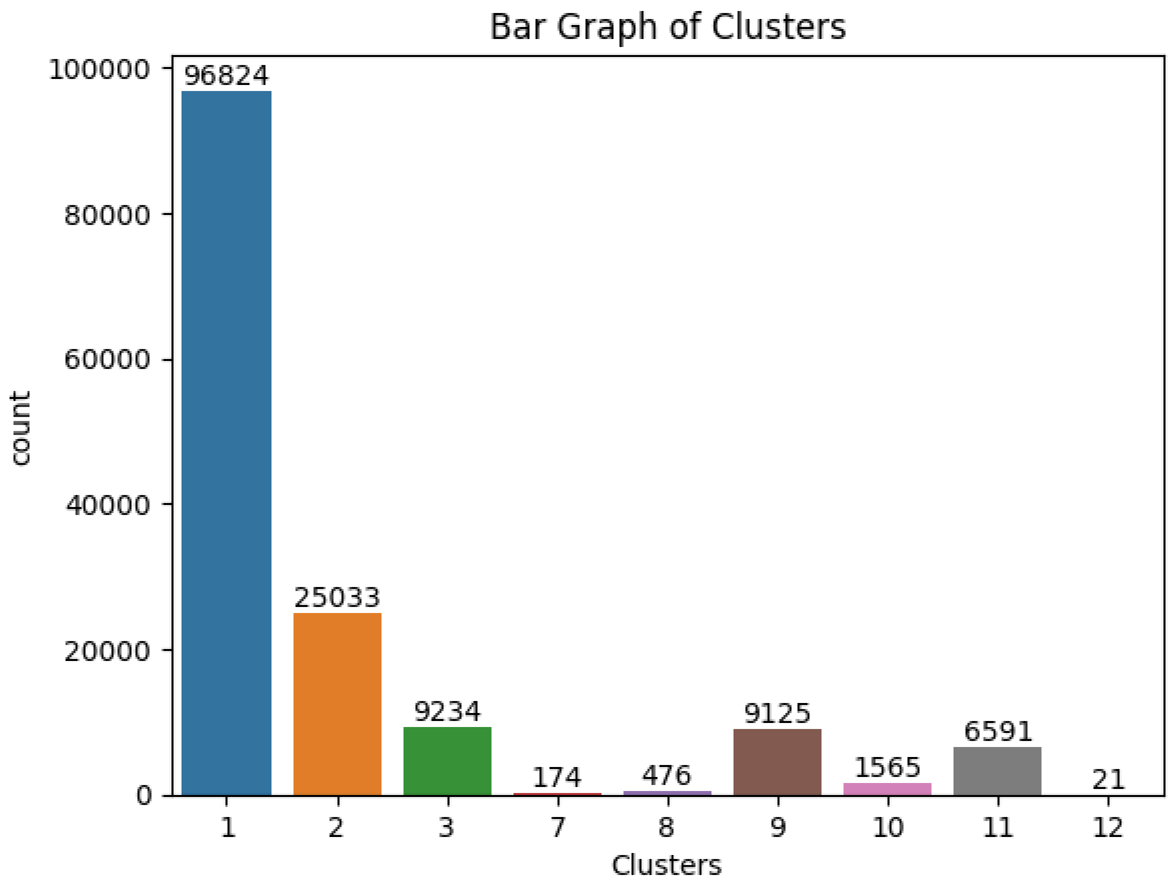
```
ax = sns.countplot(x=df2['Prediction'], data=df2)
plt.title('Bar Graph of Prediction')

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p
                ha='center', va='center', fontsize=10, color='black', xytext
                textcoords='offset points')

plt.show()
```



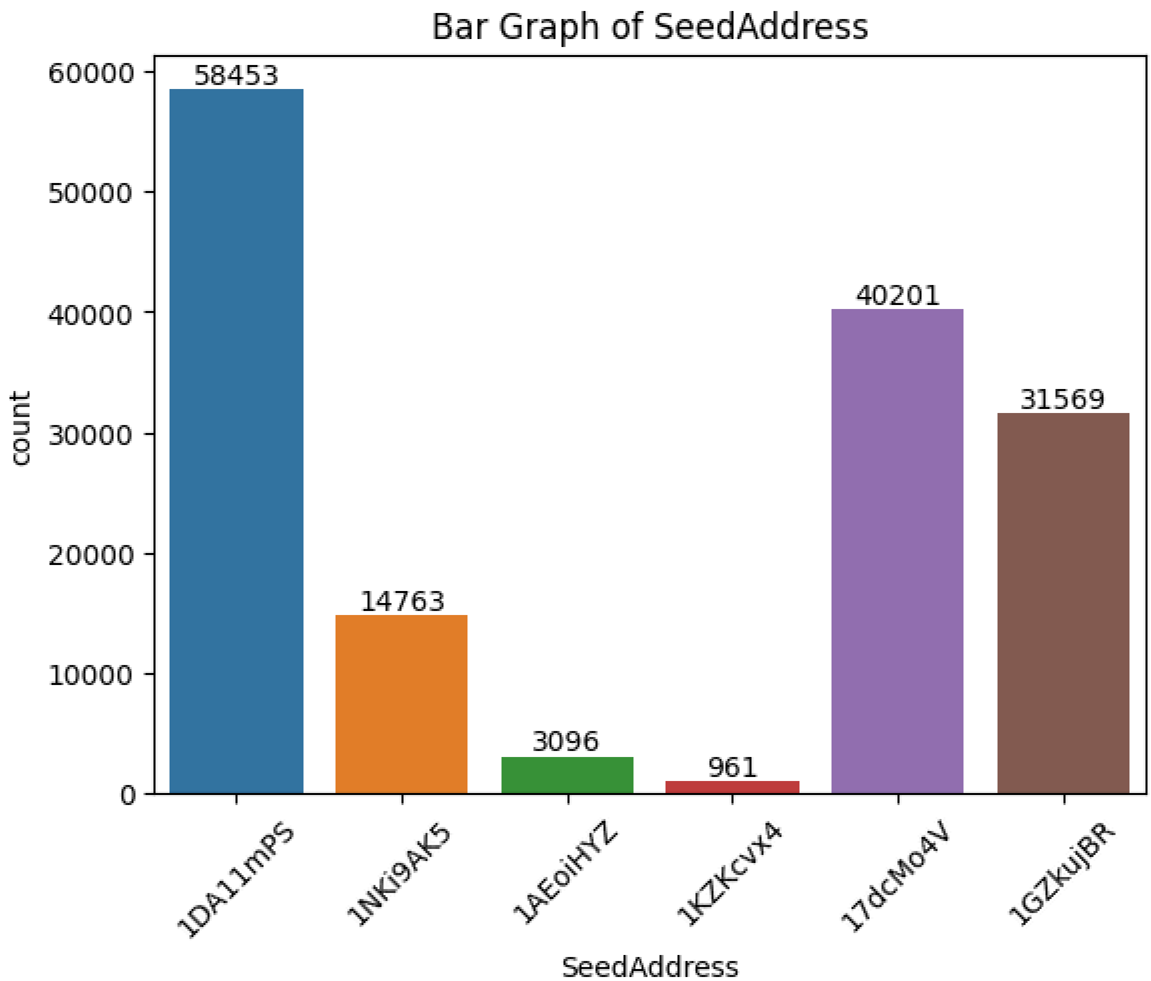




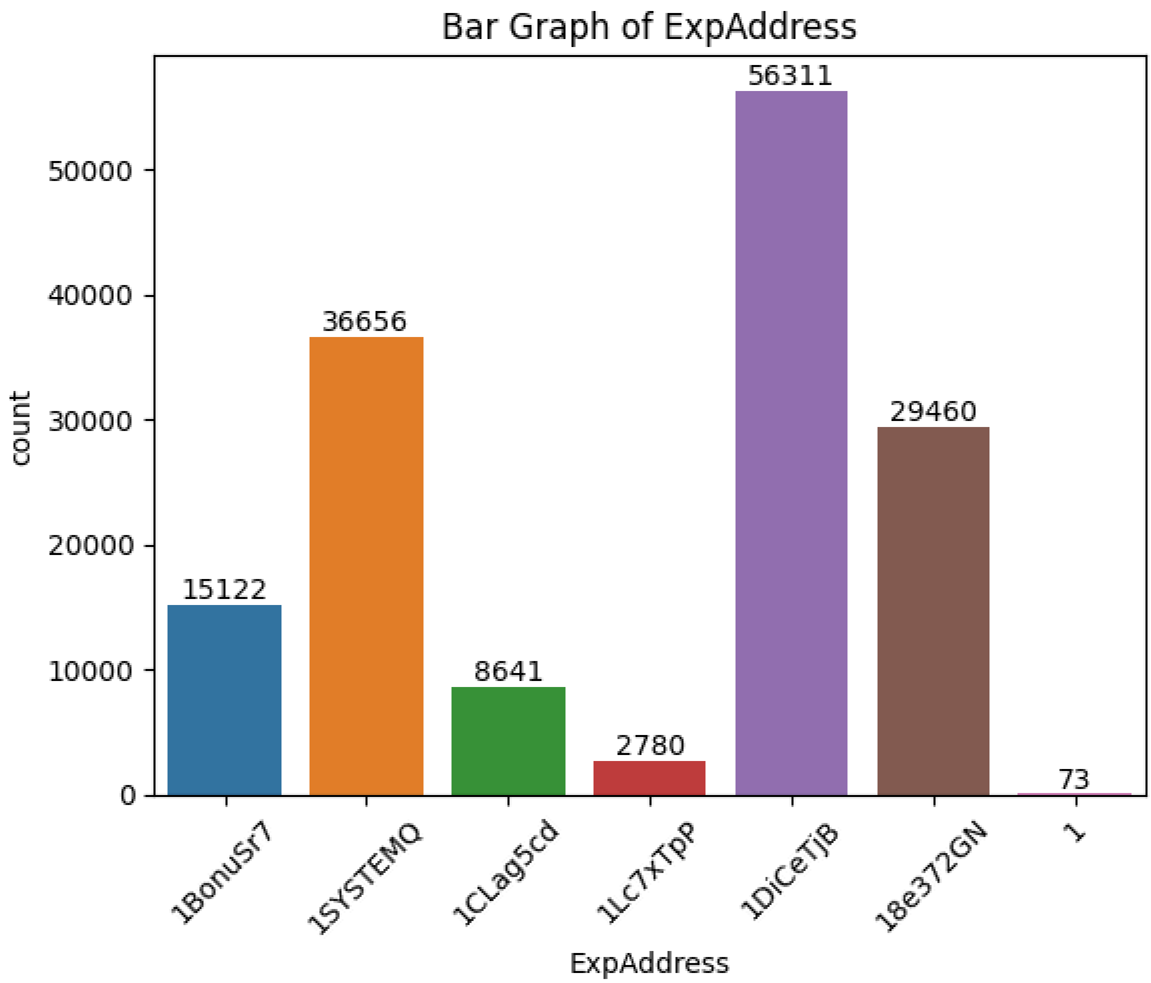
Loading [MathJax]/extensions/Safe.js

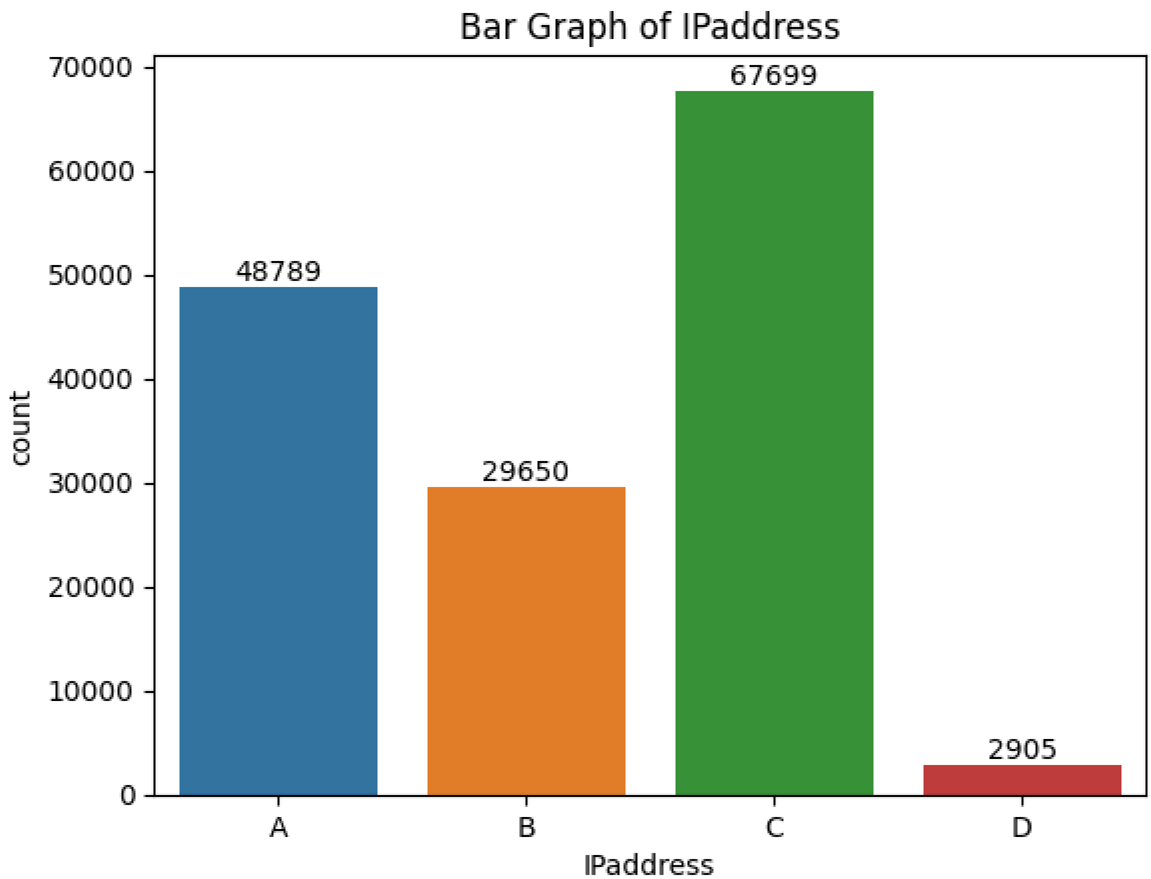


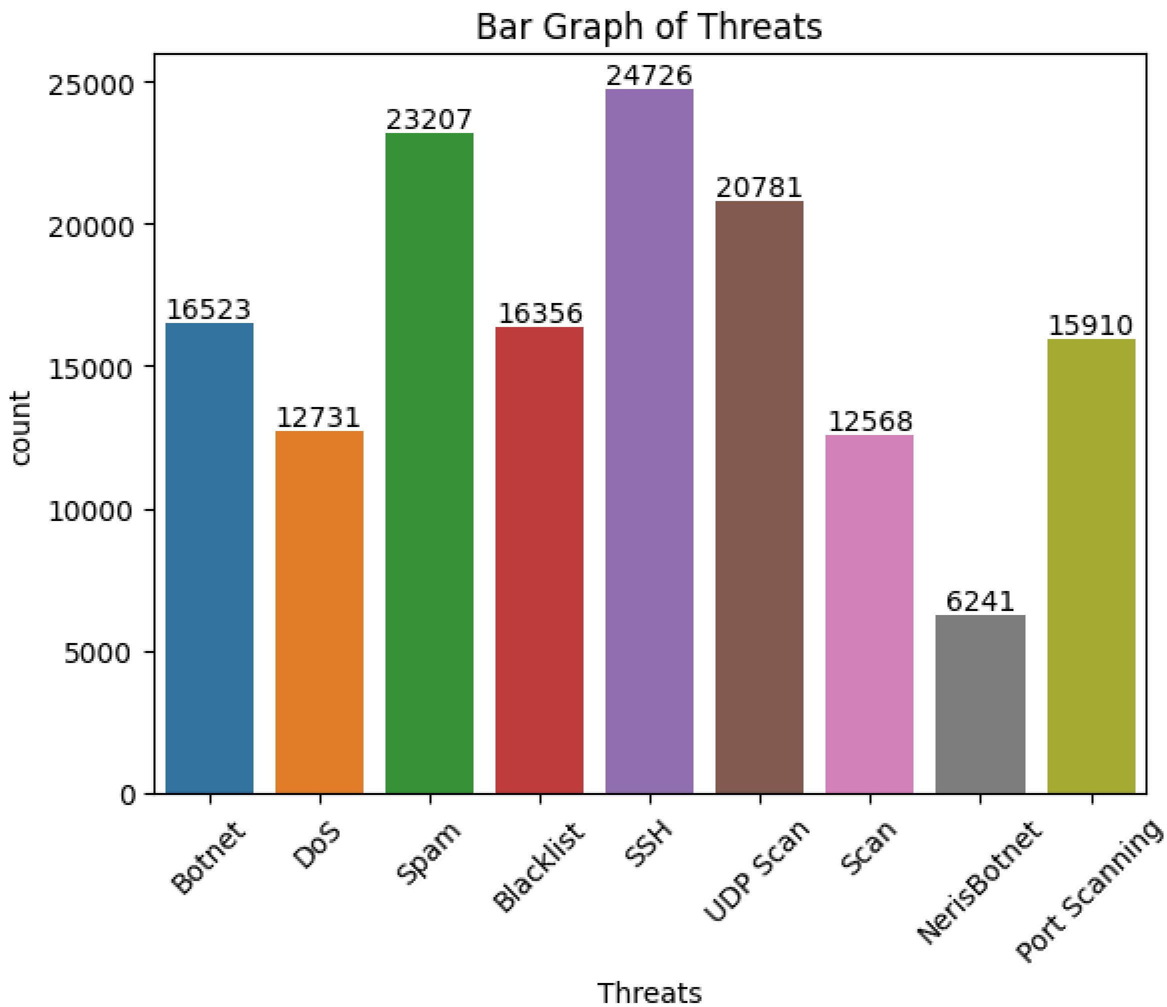
Bar Graph of Clusters
100000
80000
60000
40000
20000
0
count
96824
25033
9234
174
476
9125
1565
6591
21
1
2
3
7
8
9
10
11
12
Clusters

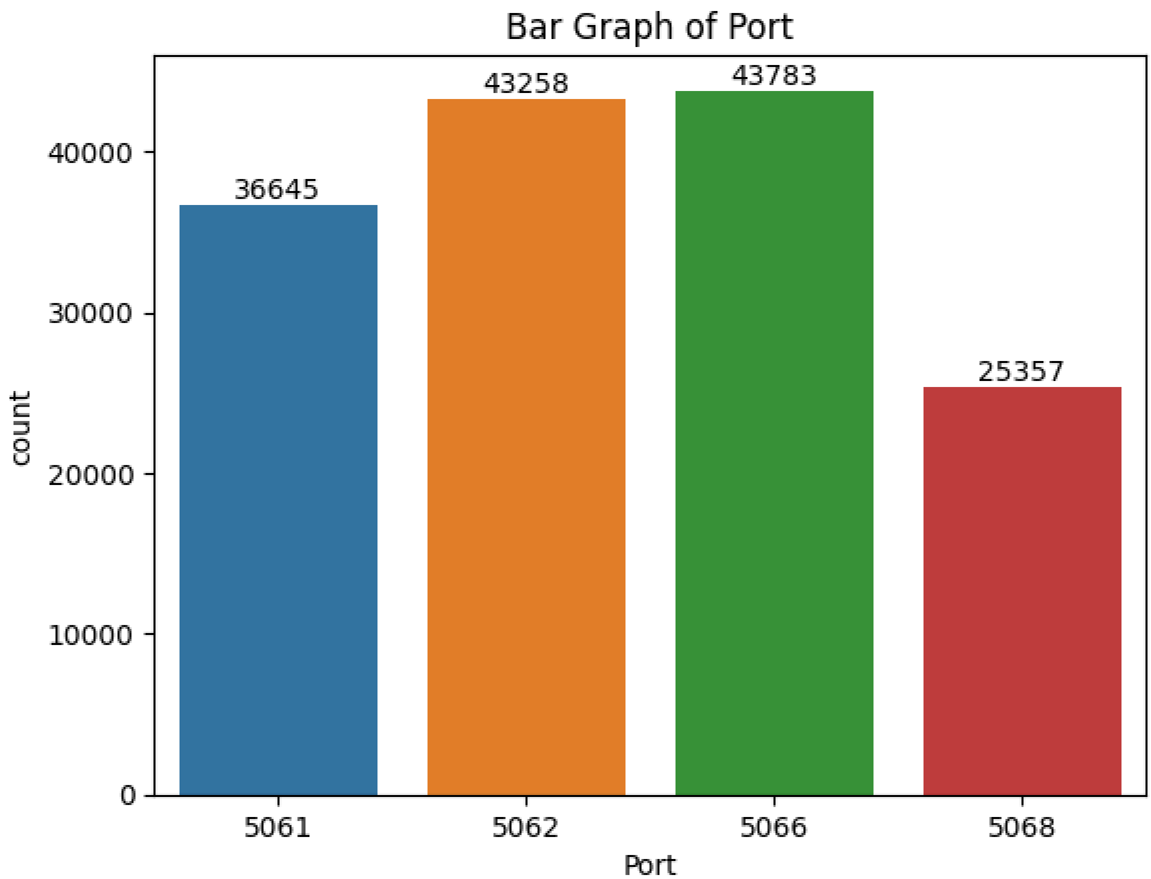


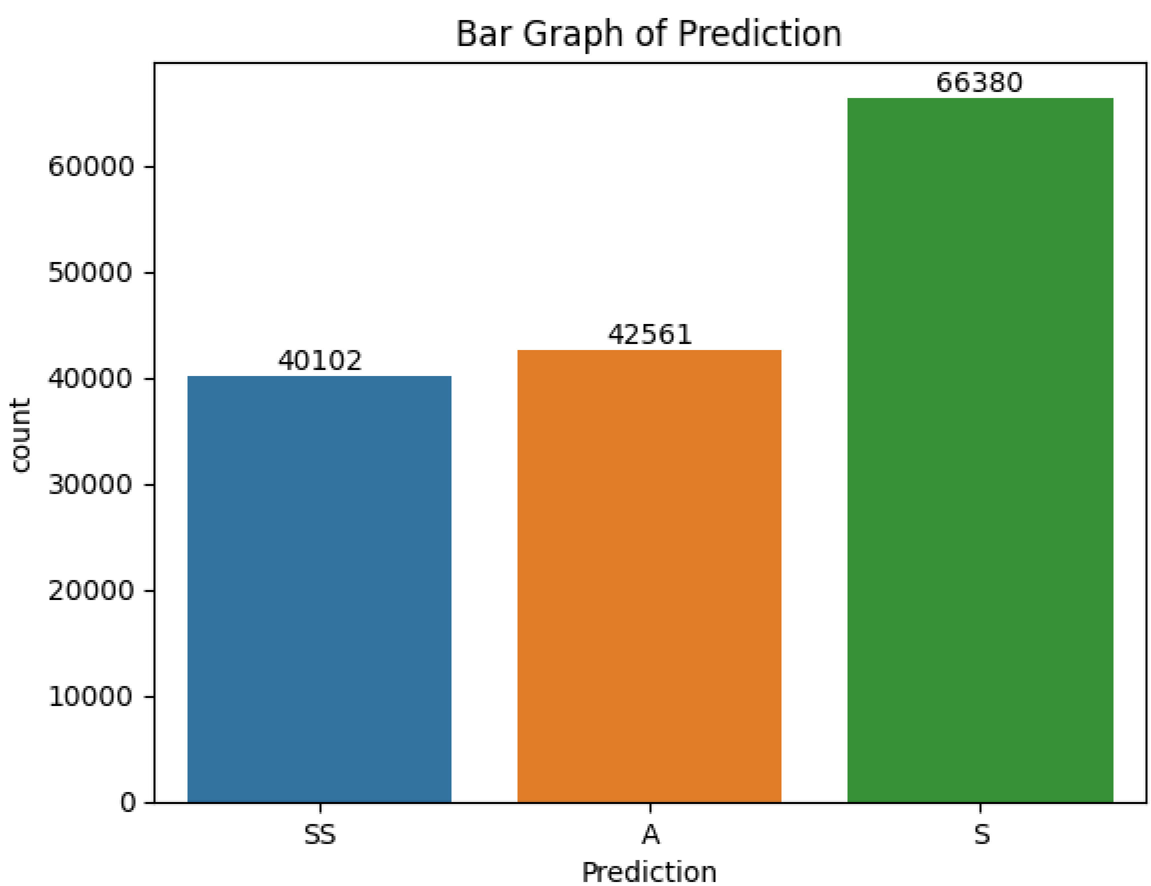
Bar Graph of SeedAddress
count
SeedAddress
60000
50000
40000
30000
20000
10000
0
58453
14763
3096
961
40201
31569
1DA11mPS
1NKi9AK5
1AEoiHYZ
1KZKcvx4
17dcMo4V
1GZkujBR


Bar Graph of ExpAddress
56311
36656
29460
15122
8641
2780
73
50000
40000
30000
20000
10000
0
count
1BonuSr7
1SYSTEMQ
1CLag5cd
1Lc7xTpP
1DiCeTjB
18e372GN
1
ExpAddress


Bar Graph of IPaddress
70000
60000
50000
40000
30000
20000
10000
0
count
48789
29650
67699
2905
A
B
C
D
IPaddress

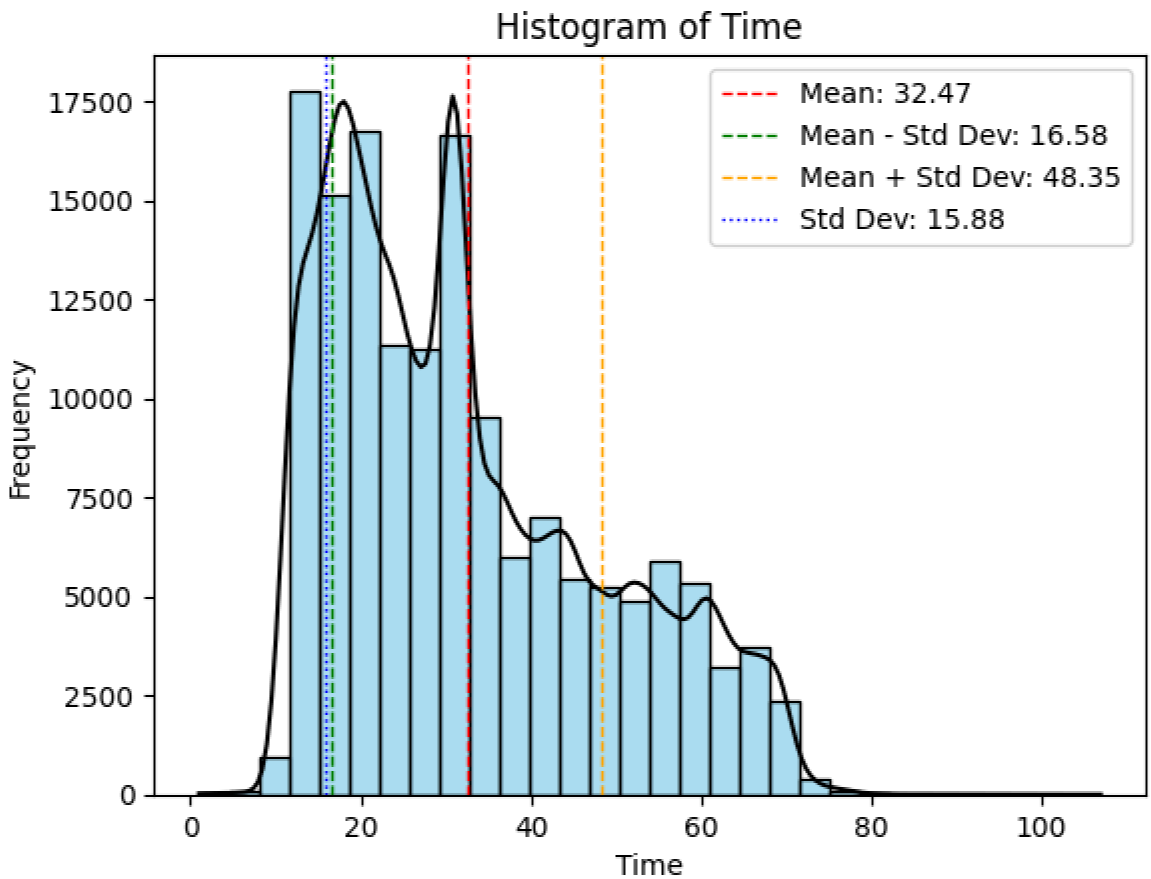

Bar Graph of Threats
count
Threats
25000
20000
15000
10000
5000
0
16523
12731
23207
16356
24726
20781
12568
6241
15910
Botnet
DoS
Spam
Blacklist
SSH
UDP Scan
Scan
NerisBotnet
Port Scanning

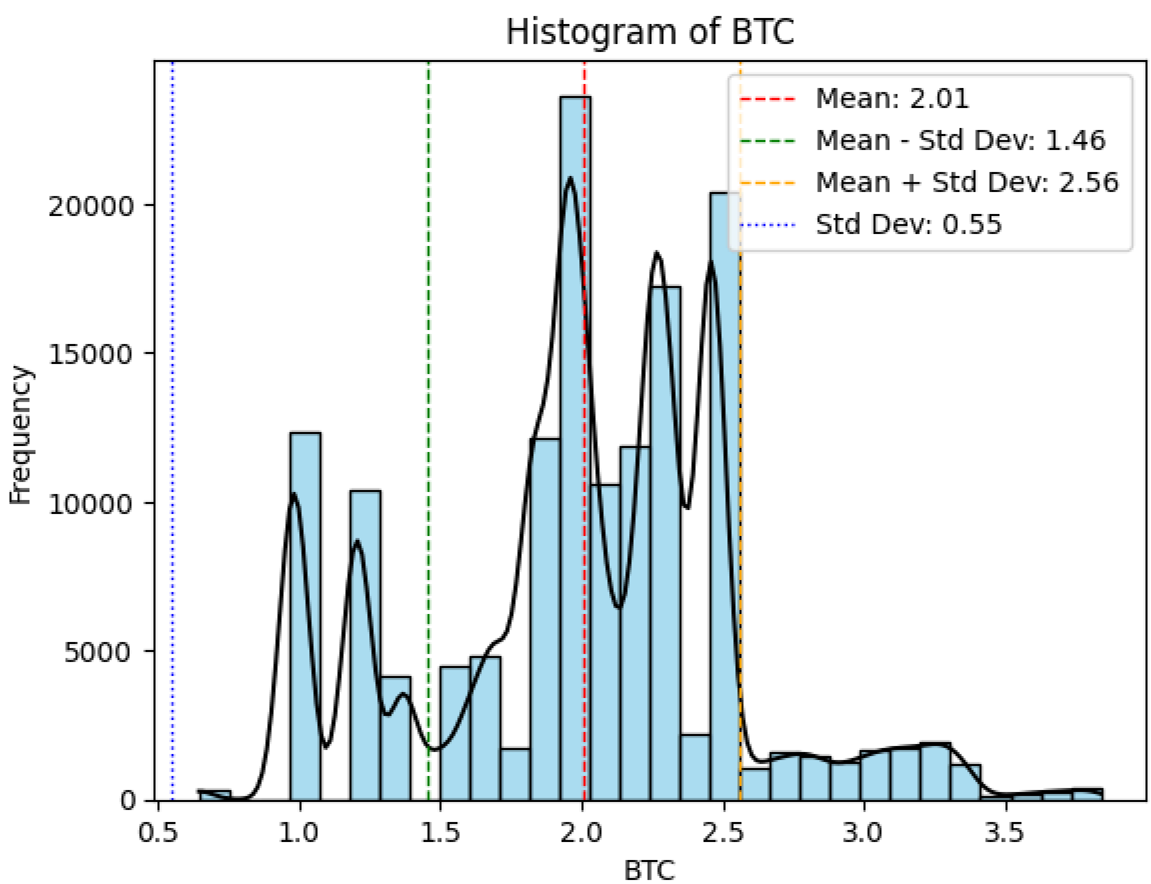

Bar Graph of Port
count
Port
36645
43258
43783
25357
5061
5062
5066
5068
0
10000
20000
30000
40000

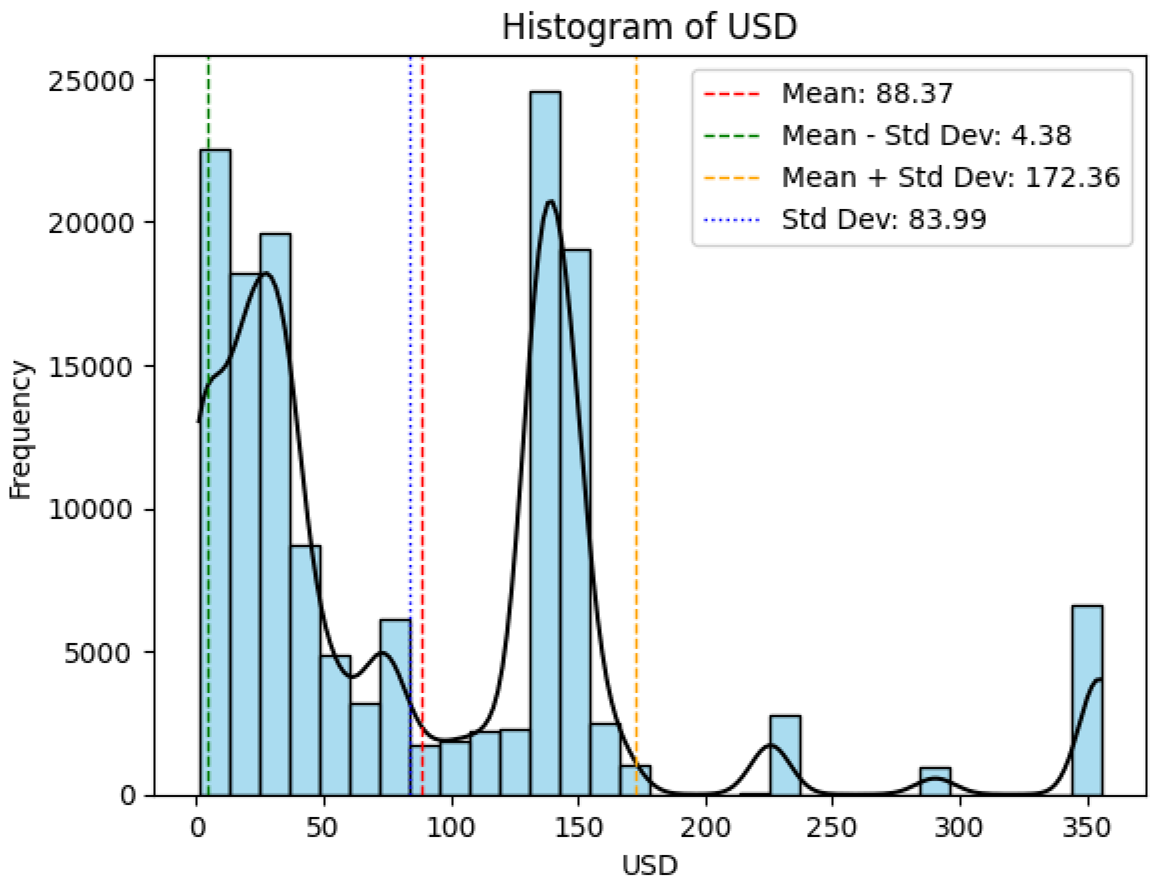

Bar Graph of Prediction
count
Prediction
40102
42561
66380
SS
A
S
0
10000
20000
30000
40000
50000
60000

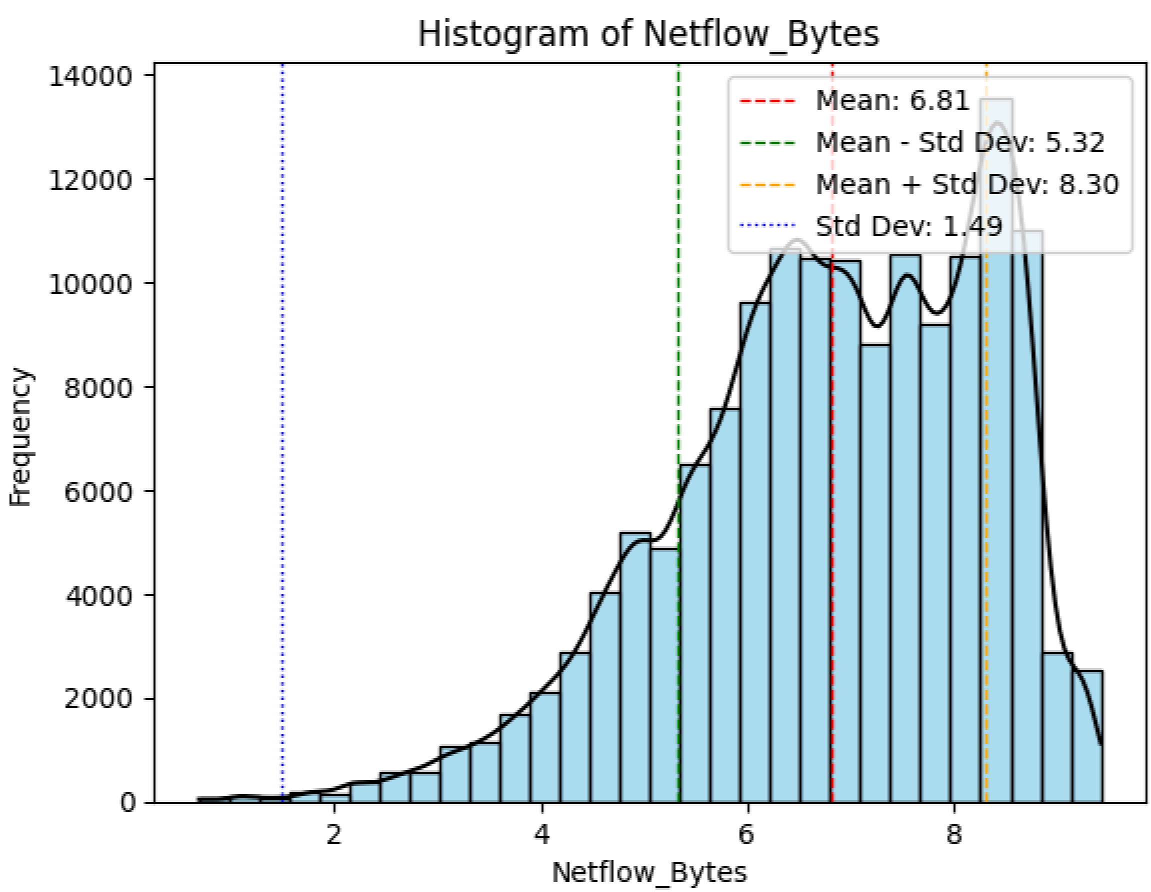
In [5]:
```
# --- Numeric visualizations (count, mean and standard deviation) --- #

# Time

feature = 'Time'
data = df2[feature]
mean = np.mean(data)
std_dev = np.std(data)

ax = sns.histplot(data, bins=30, kde=True, color='skyblue', edgecolor='black
ax.lines[0].set_color('black')

plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label=f'Mear
plt.axvline(mean - std_dev, color='green', linestyle='dashed', linewidth=1,
plt.axvline(mean + std_dev, color='orange', linestyle='dashed', linewidth=1,
plt.axvline(std_dev, color='blue', linestyle='dotted', linewidth=1, label=f'

plt.legend(loc='upper right')

plt.title(f'Histogram of {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()

# # BTC

feature = 'BTC'
data = df2[feature]
mean = np.mean(data)
std_dev = np.std(data)

ax = sns.histplot(data, bins=30, kde=True, color='skyblue', edgecolor='black
ax.lines[0].set_color('black')

plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label=f'Mear
plt.axvline(mean - std_dev, color='green', linestyle='dashed', linewidth=1,
plt.axvline(mean + std_dev, color='orange', linestyle='dashed', linewidth=1,
plt.axvline(std_dev, color='blue', linestyle='dotted', linewidth=1, label=f'

plt.legend(loc='upper right')

plt.title(f'Histogram of {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()

# # USD

feature = 'USD'
data = df2[feature]
mean = np.mean(data)
std_dev = np.std(data)

ax = sns.histplot(data, bins=30, kde=True, color='skyblue', edgecolor='black
ax.lines[0].set_color('black')
```

```
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label=f'Mear
plt.axvline(mean - std_dev, color='green', linestyle='dashed', linewidth=1,
plt.axvline(mean + std_dev, color='orange', linestyle='dashed', linewidth=1,
plt.axvline(std_dev, color='blue', linestyle='dotted', linewidth=1, label=f'

plt.legend(loc='upper right')

plt.title(f'Histogram of {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()

# Netflow_Bytes

feature = 'Netflow_Bytes'
data = df2[feature]
mean = np.mean(data)
std_dev = np.std(data)

ax = sns.histplot(data, bins=30, kde=True, color='skyblue', edgecolor='black
ax.lines[0].set_color('black')

plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label=f'Mear
plt.axvline(mean - std_dev, color='green', linestyle='dashed', linewidth=1,
plt.axvline(mean + std_dev, color='orange', linestyle='dashed', linewidth=1,
plt.axvline(std_dev, color='blue', linestyle='dotted', linewidth=1, label=f'

plt.legend(loc='upper right')

plt.title(f'Histogram of {feature}')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.show()
```


Histogram of Time
Mean: 32.47
Mean - Std Dev: 16.58
Mean + Std Dev: 48.35
Std Dev: 15.88
Frequency
Time
0
2500
5000
7500
10000
12500
15000
17500
0
20
40
60
80
100


Histogram of BTC
Mean: 2.01
Mean - Std Dev: 1.46
Mean + Std Dev: 2.56
Std Dev: 0.55
Frequency
BTC
0
5000
10000
15000
20000
0.5
1.0
1.5
2.0
2.5
3.0
3.5





Loading [MathJax]/extensions/Safe.js

In [6]:

```
#The preprocessing module in scikit-learn provides various tools and technic
#feeding it into machine learning models.
#This preprocessing is crucial to improve the quality of your data and the p

from sklearn import preprocessing
#The code segment uses scikit-learn's LabelEncoder to transform categorical
#Each categorical column, such as 'Protocol,' 'Flag,' 'Family,' 'SeedAddress
#'Prediction,' is encoded into unique numeric labels.
#This preprocessing step is essential for machine learning algorithms, as th
#instead of categorical labels.



lab_encoder = preprocessing.LabelEncoder()                    # transformat
df2['Protocol'] = lab_encoder.fit_transform(df2['Protocol'])
df2['Flag'] = lab_encoder.fit_transform(df2['Flag'])
df2['Family'] = lab_encoder.fit_transform(df2['Family'])

df2['SeedAddress'] = lab_encoder.fit_transform(df2['SeedAddress'])
df2['ExpAddress'] = lab_encoder.fit_transform(df2['ExpAddress'])
df2['IPaddress'] = lab_encoder.fit_transform(df2['IPaddress'])
df2['Threats'] = lab_encoder.fit_transform(df2['Threats'])
df2['Prediction'] = lab_encoder.fit_transform(df2['Prediction'])
df2
```

Out[6]:

	Time	Protocol	Flag	Family	Clusters	SeedAddress	ExpAddress	
0	61	1	0	16	1	2	2	0.(
1	51	1	0	16	1	2	2	0.(
2	41	1	0	16	1	2	2	0.(
3	31	1	0	16	1	2	2	0.(
4	68	1	0	16	1	2	2	0.(
...	...	...	...	...	...	...	...	
149038	44	2	2	15	3	1	6	3.(
149039	44	2	2	15	3	1	6	3.(
149040	44	2	2	15	3	1	6	3.(
149041	44	2	2	15	3	1	6	3.(
149042	44	2	2	15	3	1	6	3.(

149043 rows × 14 columns

In [7]:

```
#The train_test_split function from scikit-learn is used to split a dataset
#a training set and a testing (or validation) set. This function is commonly
#to assess the performance of a model on unseen data. It takes as input the
#and labels (y), and divides it into training data (X_train and y_train) use
#testing data (X_test and y_test) used to evaluate the model's performance.

from sklearn.model_selection import train_test_split  # library for machine
```

```
#common procedure in machine learning for splitting a dataset into training
#from scikit-learn. Here's a breakdown of what each line of code does:

X = df2.iloc[:, :-1] #This line selects all rows and all columns of the Data
#It's assuming that the last column contains the target variable or labels,

y = df2.iloc[:, -1]  # This line selects all rows but only the last column c
#This is to isolate the target variable or labels, and y will contain these

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8,
#This line uses the train_test_split function to split the data into trainir
#Here's a breakdown of the parameters:

#X and y: The feature matrix and target variable.
#train_size=0.8: This parameter specifies that 80% of the data should be use
#(you can adjust this percentage as needed).
#random_state=42: This parameter sets the random seed for reproducibility, e
#run the code.


#After running this code, you will have:

#X_train: The feature matrix for training.
#X_test: The feature matrix for testing.
#y_train: The target variable for training.
#y_test: The target variable for testing.
#These subsets can then be used for training and evaluating your machine lea
```

In [8]:
```
X_train
X_test
y_train
y_test
```

Out[8]:
```
42916     1
45544     2
137525    0
108170    1
85804     2
         ..
91256     1
132188    1
94999     2
3431      0
147946    0
Name: Prediction, Length: 29809, dtype: int64
```

In [9]:
```
#The %%time command is typically used in Jupyter Notebook environments, such
#It is called a "magic command" and is used to measure the execution time oi
#When you include %%time at the beginning of a cell, it tells Jupyter to mea
#that cell
#%%time


#          various libraries and tools for building and evaluating machine lea
```

Loading [MathJax]/extensions/Safe.js

```
# Imported models: ensemble, random forest, SVM, Naive Bayes, genetic algori
# Imported evaluation metrics: accuracy, precision, recall, f1 score

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

from sklearn.ensemble import StackingClassifier #ensmbl method of stacking c
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


from sklearn.tree import DecisionTreeClassifier   #estimator in GA
import numpy as np

import warnings
warnings.filterwarnings('ignore')
```

In [10]:

```
rf = RandomForestClassifier(n_estimators=100, random_state=42) #  It specifi
#In this case, there are 100 trees in the forest

# random_state: This parameter is used to set the random seed for reproducib
#By setting it to 42, the randomization process will be the same each time t
#ensuring consistent results for the Random Forest model.

rf.fit(X_train, y_train)

rf_pred=rf.predict(X_test)

#This code snippet uses the trained Random Forest classifier (rf) to make pr
#The predict method takes the test features in X_test as input and produces
#The predictions are stored in the rf_pred variable, which can be used for f
#the model performs on unseen data.



rf_accuracy = accuracy_score(rf_pred, y_test)
rf_report = classification_report(rf_pred, y_test)
rf_matrix = confusion_matrix(rf_pred, y_test)
print('Accuracy of Random Forest : ', round(rf_accuracy, 3))
print('Classification report of Random Forest : \n', rf_report)
print('Confusion Matrix of Random Forest : \n', rf_matrix)




#The accuracy_score function from scikit-learn is used to calculate the accu
#compared to the actual labels (y_test). This score measures the proportion
#classification_report: The classification_report function generates a compr
#F1-score, and support for each class in the classification problem. It prov
#performance for different classes.
#confusion_matrix: The confusion_matrix function computes a confusion matrix
#false positive, and false negative counts for the classification results. I
ng and where it might be making errors.
```

Loading [MathJax]/extensions/Safe.js

```
#Finally, the code prints out the accuracy, classification report, and confu
#allowing you to evaluate its performance on the test data.
```

```
Accuracy of Random Forest :  0.994
Classification report of Random Forest :
               precision    recall  f1-score   support

           0       0.99      0.99      0.99      8400
           1       0.99      0.99      0.99     13359
           2       1.00      0.99      0.99      8050

    accuracy                           0.99     29809
   macro avg       0.99      0.99      0.99     29809
weighted avg       0.99      0.99      0.99     29809

Confusion Matrix of Random Forest :
 [[ 8346    49     5]
 [   49 13280    30]
 [    8    44  7998]]
```

In [11]:

```
svr = LinearSVC()
svr.fit(X_train, y_train)
svr_pred = svr.predict(X_test)



#a Support Vector Machine (SVM) classifier with a linear kernel (LinearSVC)

#svr = LinearSVC(): An instance of the LinearSVC classifier is created.

#svr.fit(X_train, y_train): The LinearSVC classifier is trained on the trair
#This step involves finding the hyperplane that best separates the data poir
#between them.

#svr_pred = svr.predict(X_test): The trained SVM classifier is used to make
#These predictions are stored in the svr_pred variable.




svr_accuracy = accuracy_score(svr_pred, y_test)
svr_report = classification_report(svr_pred, y_test)
svr_matrix = confusion_matrix(svr_pred, y_test)
print('Accuracy of SVM : ', round(svr_accuracy, 3))
print('Classification report of SVM : \n', svr_report)
print('Confusion Matrix of SVM :\n', svr_matrix)






#svr_accuracy = accuracy_score(svr_pred, y_test): The accuracy of the SVM cl
#is calculated by comparing them to the true labels (y_test). The result is
```

```
#svr_report = classification_report(svr_pred, y_test): The classification_re
#classification report, including metrics such as precision, recall, F1-scor
#This report is stored in the svr_report variable.

#svr_matrix = confusion_matrix(svr_pred, y_test): The confusion matrix is co
#true labels (y_test). The confusion matrix provides information about the n
#false positive, and false negative predictions. It is stored in the svr_mat

#Finally, the results are printed using print statements:

#The accuracy of the SVM classifier is printed with a rounded value.
#The classification report, which includes precision, recall, F1-score, and
#The confusion matrix, which shows the distribution of true and false predic
#These metrics help evaluate the performance of the SVM classifier in terms
```

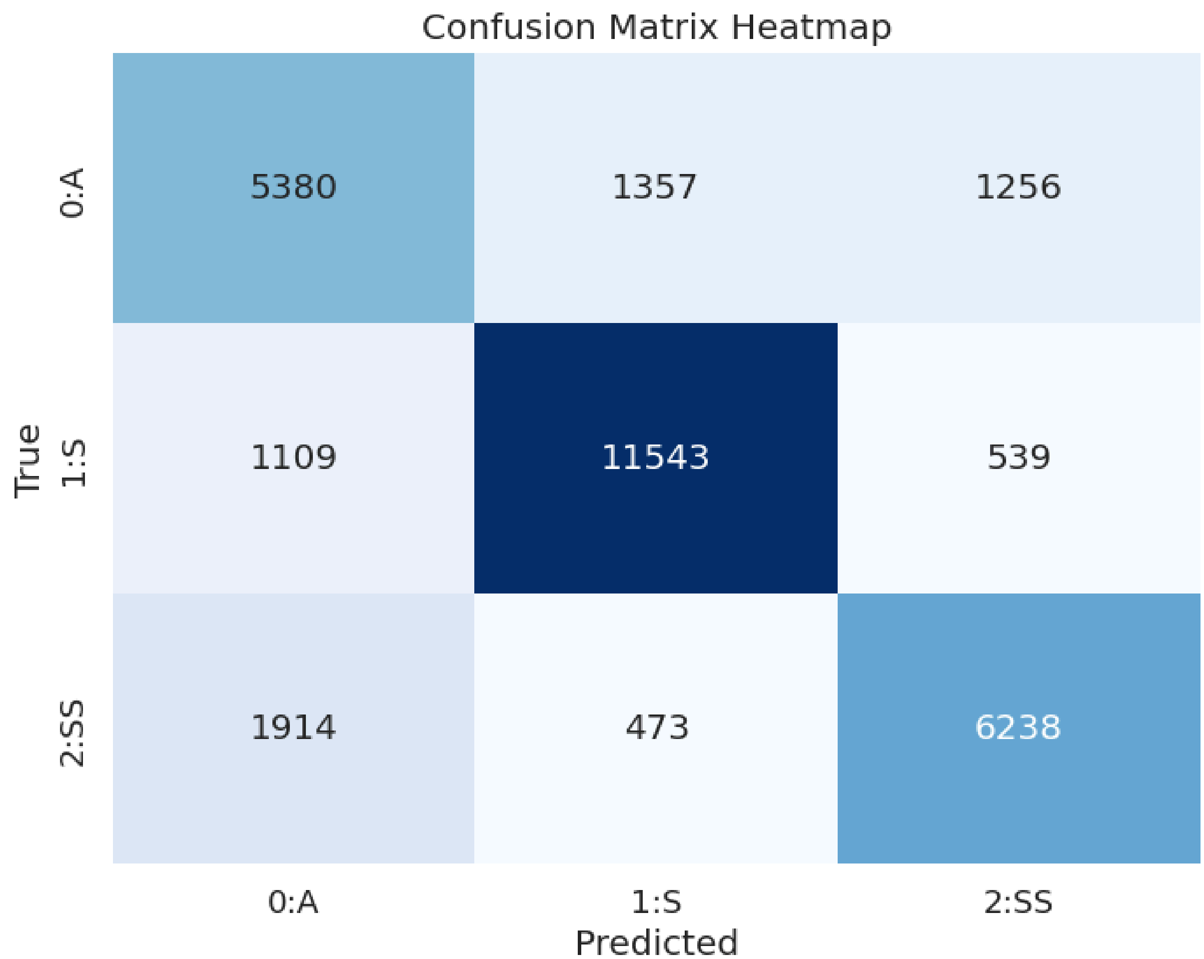
```
Accuracy of SVM :  0.574
Classification report of SVM :
               precision    recall  f1-score   support

           0       0.03      0.59      0.05       369
           1       0.68      0.93      0.79      9760
           2       0.97      0.39      0.56     19680

    accuracy                           0.57     29809
   macro avg       0.56      0.64      0.47     29809
weighted avg       0.86      0.57      0.63     29809

Confusion Matrix of SVM :
 [[ 219   73   77]
 [ 442 9122  196]
 [7742 4178 7760]]
```

In [12]:
```
#Naive Bayes Algorithm

nb = GaussianNB()
nb.fit(X_train, y_train)
nb_pred = nb.predict(X_test)


nb_accuracy = accuracy_score(nb_pred, y_test)
nb_report = classification_report(nb_pred, y_test)
nb_matrix = confusion_matrix(nb_pred, y_test)
print('Accuracy of Naive Bayes : ', round(nb_accuracy, 3))
print('Classification report of Naive Bayes : \n', nb_report)
print('Confusion Matrix of Naive Bayes :\n', nb_matrix)





# Assuming you already have nb_pred and y_test defined

nb_accuracy = accuracy_score(nb_pred, y_test)
nb_report = classification_report(nb_pred, y_test)
nb_matrix = confusion_matrix(nb_pred, y_test)
```

```python
print('Accuracy of Naive Bayes : ', round(nb_accuracy, 3))
print('Classification report of Naive Bayes : \n', nb_report)
print('Confusion Matrix of Naive Bayes :\n', nb_matrix)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)  # Adjust the font size for better readability
sns.heatmap(nb_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=["0:A", "1:S", "2:SS"], yticklabels=["0:A", "1:S", "
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix Heatmap")
plt.show()
```

```
Accuracy of Naive Bayes :  0.777
Classification report of Naive Bayes :
               precision    recall  f1-score   support

           0       0.64      0.67      0.66      7993
           1       0.86      0.88      0.87     13191
           2       0.78      0.72      0.75      8625

    accuracy                           0.78     29809
   macro avg       0.76      0.76      0.76     29809
weighted avg       0.78      0.78      0.78     29809

Confusion Matrix of Naive Bayes :
 [[ 5380  1357  1256]
 [ 1109 11543   539]
 [ 1914   473  6238]]
```


```
Accuracy of Naive Bayes :  0.777
Classification report of Naive Bayes :
               precision    recall  f1-score   support

           0       0.64      0.67      0.66      7993
           1       0.86      0.88      0.87     13191
           2       0.78      0.72      0.75      8625

    accuracy                           0.78     29809
   macro avg       0.76      0.76      0.76     29809
weighted avg       0.78      0.78      0.78     29809

Confusion Matrix of Naive Bayes :
 [[ 5380  1357  1256]
 [ 1109 11543   539]
 [ 1914   473  6238]]
```

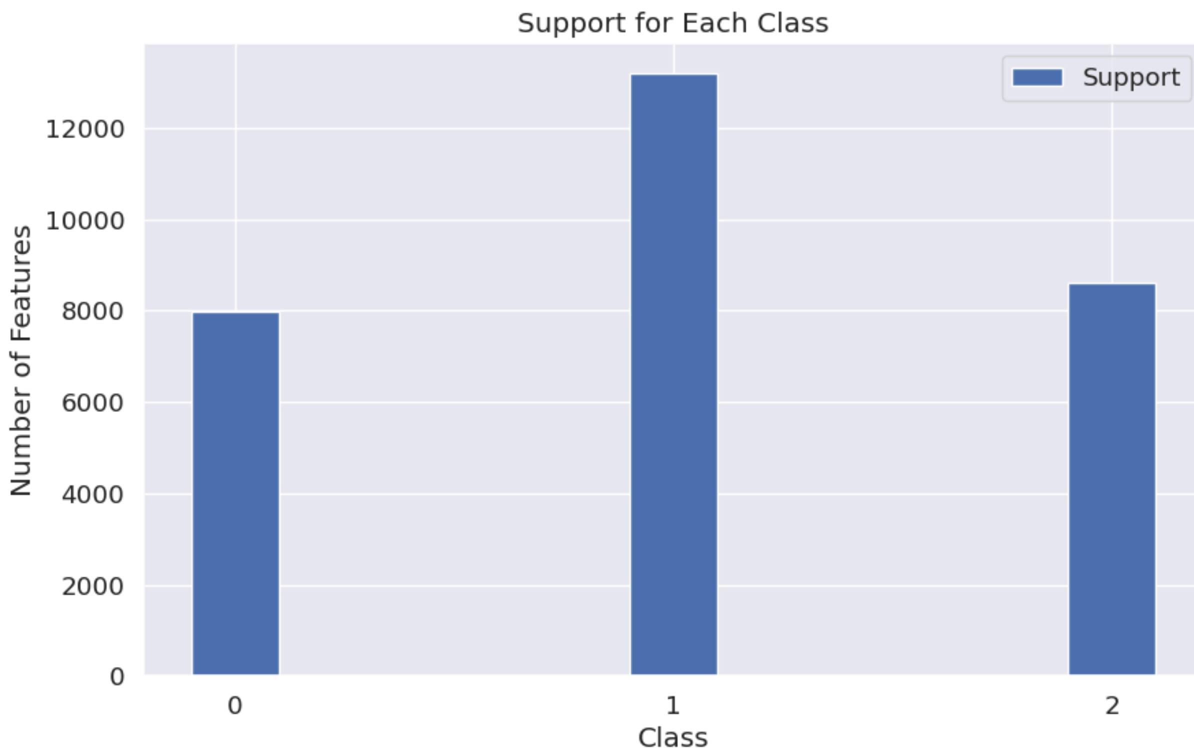


In [13]:

```
# Assuming you already have nb_pred and y_test defined

nb_accuracy = accuracy_score(nb_pred, y_test)
nb_report = classification_report(nb_pred, y_test, output_dict=True)  # Use
nb_matrix = confusion_matrix(nb_pred, y_test)

# Extract support for all classes
labels = [str(label) for label in np.unique(np.concatenate((nb_pred, y_test)
support = [nb_report[label]['support'] if label in nb_report else 0 for labe

print('Accuracy of Naive Bayes : ', round(nb_accuracy, 3))
print('Classification report of Naive Bayes : \n', classification_report(nb_
print('Confusion Matrix of Naive Bayes :\n', nb_matrix)

# Plot support
plt.figure(figsize=(10, 6))
plt.bar(labels, support, width=0.2, label='Support', align='center')

plt.xlabel('Class')
plt.ylabel('Number of Features')
plt.xticks(labels)
plt.legend()
plt.title('Support for Each Class')
plt.show()
```

Loading [MathJax]/extensions/Safe.js

```
Accuracy of Naive Bayes :  0.777
Classification report of Naive Bayes :
               precision    recall  f1-score   support

           0       0.64      0.67      0.66      7993
           1       0.86      0.88      0.87     13191
           2       0.78      0.72      0.75      8625

    accuracy                           0.78     29809
   macro avg       0.76      0.76      0.76     29809
weighted avg       0.78      0.78      0.78     29809

Confusion Matrix of Naive Bayes :
 [[ 5380  1357  1256]
 [ 1109 11543   539]
 [ 1914   473  6238]]
```

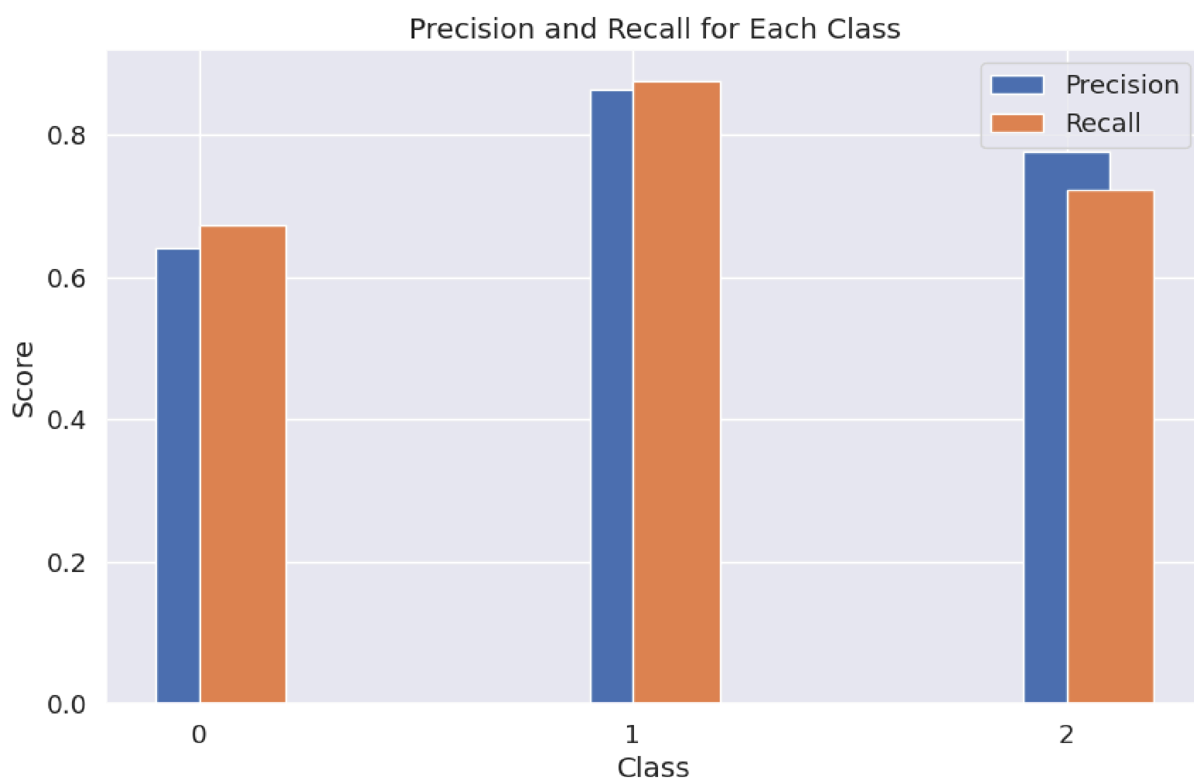


In [14]:

```
# Assuming you already have nb_pred and y_test defined

nb_accuracy = accuracy_score(nb_pred, y_test)
nb_report = classification_report(nb_pred, y_test, output_dict=True)  # Use
nb_matrix = confusion_matrix(nb_pred, y_test)

# Extract precision and recall for all classes
labels = [str(label) for label in np.unique(np.concatenate((nb_pred, y_test)
precision = [nb_report[label]['precision'] if label in nb_report else 0.0 fo
recall = [nb_report[label]['recall'] if label in nb_report else 0.0 for labe

print('Accuracy of Naive Bayes : ', round(nb_accuracy, 3))
print('Classification report of Naive Bayes : \n', classification_report(nb_

# Plot precision and recall
plt.figure(figsize=(10, 6))
```

```python
plt.bar(labels, precision, width=0.2, label='Precision', align='center')
plt.bar(labels, recall, width=0.2, label='Recall', align='edge')

plt.xlabel('Class')
plt.ylabel('Score')
plt.xticks(labels)
plt.legend()
plt.title('Precision and Recall for Each Class')
plt.show()
```

```
Accuracy of Naive Bayes :  0.777
Classification report of Naive Bayes :
               precision    recall  f1-score   support

           0       0.64      0.67      0.66      7993
           1       0.86      0.88      0.87     13191
           2       0.78      0.72      0.75      8625

    accuracy                           0.78     29809
   macro avg       0.76      0.76      0.76     29809
weighted avg       0.78      0.78      0.78     29809
```



In [15]:

```python
estimators = [
    ('rf', RandomForestClassifier(n_estimators=1000, random_state=42)),
    ('svr', LinearSVC(random_state=42))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator=GaussianNB())



code snippet, a Stacking Classifier (clf) is defined using scikit-l
```

```
#Stacking is an ensemble learning method that combines multiple base estimat

#Here's a breakdown of the code:

#estimators: This is a list of tuples, where each tuple contains the name o
#Two base estimators are defined:

#'rf': A Random Forest Classifier with 1000 estimators and a random seed of
#'svr': A Linear Support Vector Classifier (LinearSVC) with a random seed o
#clf: The Stacking Classifier is created using the StackingClassifier class.

#estimators: This parameter receives the list of base estimators defined ear
#final_estimator: This parameter specifies the meta-estimator that combines
#In this case, a Gaussian Naive Bayes (GaussianNB) classifier is used as the
#The Stacking Classifier combines the predictions of the base classifiers (
#This ensemble method can often improve classification performance by levera
```

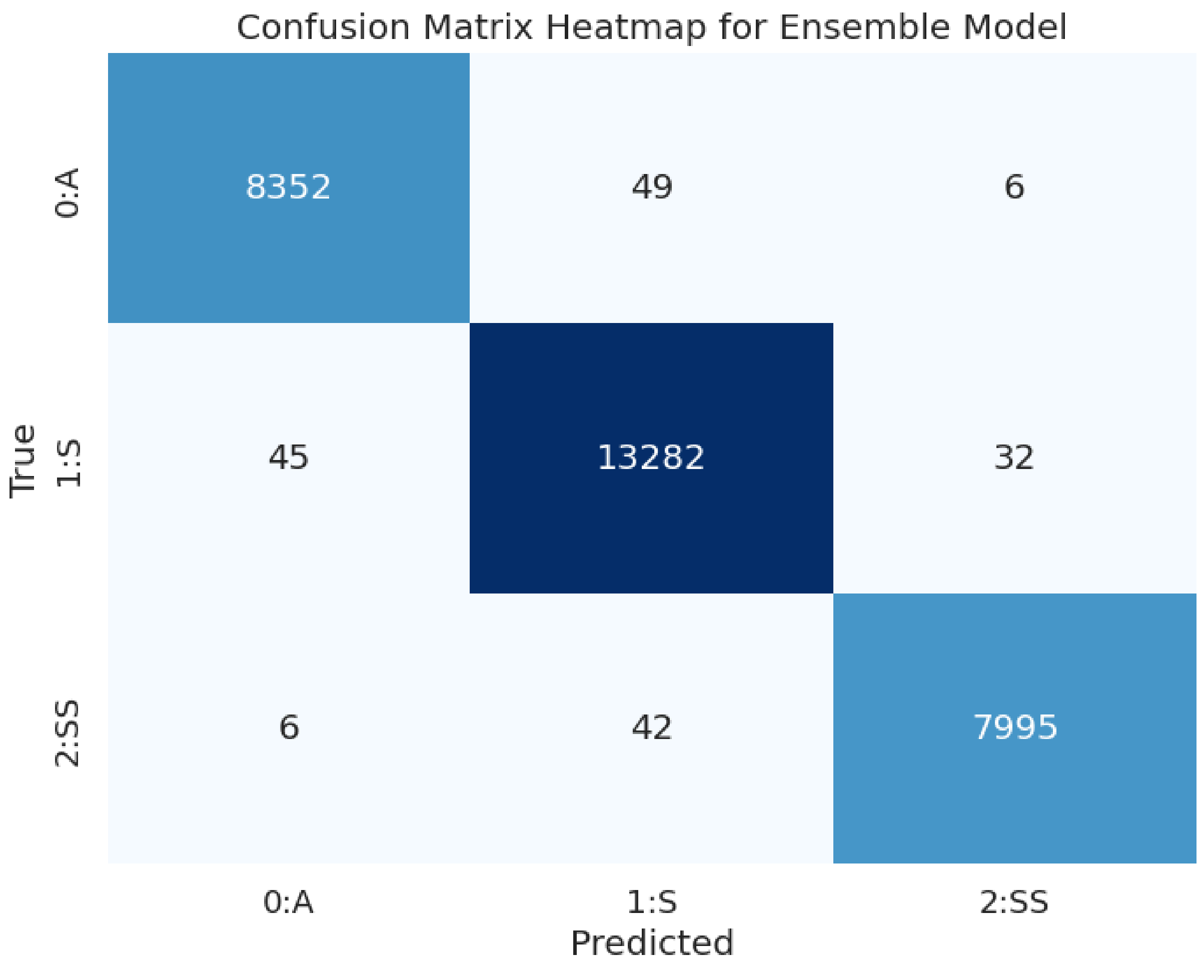
In [16]:
```
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
accuracy = accuracy_score(pred, y_test)




#In this code snippet, the Stacking Classifier (clf) is trained on the trai
#After training, the classifier is used to make predictions on the test data
#Finally, the accuracy of the predictions is calculated using scikit-learn's

#The code essentially performs the following steps:

#Trains the Stacking Classifier (clf) using the training data.
#Uses the trained classifier to predict the target labels for the test data.
#Calculates the accuracy of the predictions by comparing them to the true la
#The accuracy variable will contain the accuracy score of the Stacking Class
#This score measures how well the classifier performed in terms of correctly
```

In [17]:
```
eb_accuracy = accuracy_score(pred, y_test)
eb_matrix = confusion_matrix(pred, y_test)
eb_report = classification_report(pred, y_test)

print('Accuracy of Ensemble Model : ', round(eb_accuracy, 3))
print('Confusion Matrix of Ensemble Model : ', eb_matrix)
print('Classification Report of Ensemble Model :', eb_report)


#In this code snippet, the accuracy, confusion matrix, and classification re
#and printed.

#Here's what each part of the code does:

#eb_accuracy: Calculates the accuracy of the ensemble model's predictions by
#using the accuracy_score function.
#eb_matrix: Computes the confusion matrix for the ensemble model's predictio
#The confusion matrix provides information about the true positives, true ne
#eb_report: Generates a classification report for the ensemble model's predi
```

Loading [MathJax]/extensions/Safe.js

```
#The classification report includes metrics such as precision, recall, F1-sc
#Finally, the code prints out the accuracy, confusion matrix, and classifica
#These metrics provide insights into the model's performance in terms of cla
#classify different classes.

# Assuming you already have pred and y_test defined for your Ensemble Model

eb_accuracy = accuracy_score(pred, y_test)
eb_matrix = confusion_matrix(pred, y_test)
eb_report = classification_report(pred, y_test)

print('Accuracy of Ensemble Model : ', round(eb_accuracy, 3))
print('Confusion Matrix of Ensemble Model : \n', eb_matrix)
print('Classification Report of Ensemble Model :\n', eb_report)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)  # Adjust the font size for better readability
sns.heatmap(eb_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=["0:A", "1:S", "2:SS"], yticklabels=["0:A", "1:S", "
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix Heatmap for Ensemble Model")
plt.show()
```

```
Accuracy of Ensemble Model :  0.994
Confusion Matrix of Ensemble Model :  [[ 8352    49     6]
 [   45 13282    32]
 [    6    42  7995]]
Classification Report of Ensemble Model :               precision    recall
f1-score   support

           0       0.99      0.99      0.99      8407
           1       0.99      0.99      0.99     13359
           2       1.00      0.99      0.99      8043

    accuracy                           0.99     29809
   macro avg       0.99      0.99      0.99     29809
weighted avg       0.99      0.99      0.99     29809

Accuracy of Ensemble Model :  0.994
Confusion Matrix of Ensemble Model :
 [[ 8352    49     6]
 [   45 13282    32]
 [    6    42  7995]]
Classification Report of Ensemble Model :
               precision    recall  f1-score   support

           0       0.99      0.99      0.99      8407
           1       0.99      0.99      0.99     13359
           2       1.00      0.99      0.99      8043

    accuracy                           0.99     29809
   macro avg       0.99      0.99      0.99     29809
weighted avg       0.99      0.99      0.99     29809
```

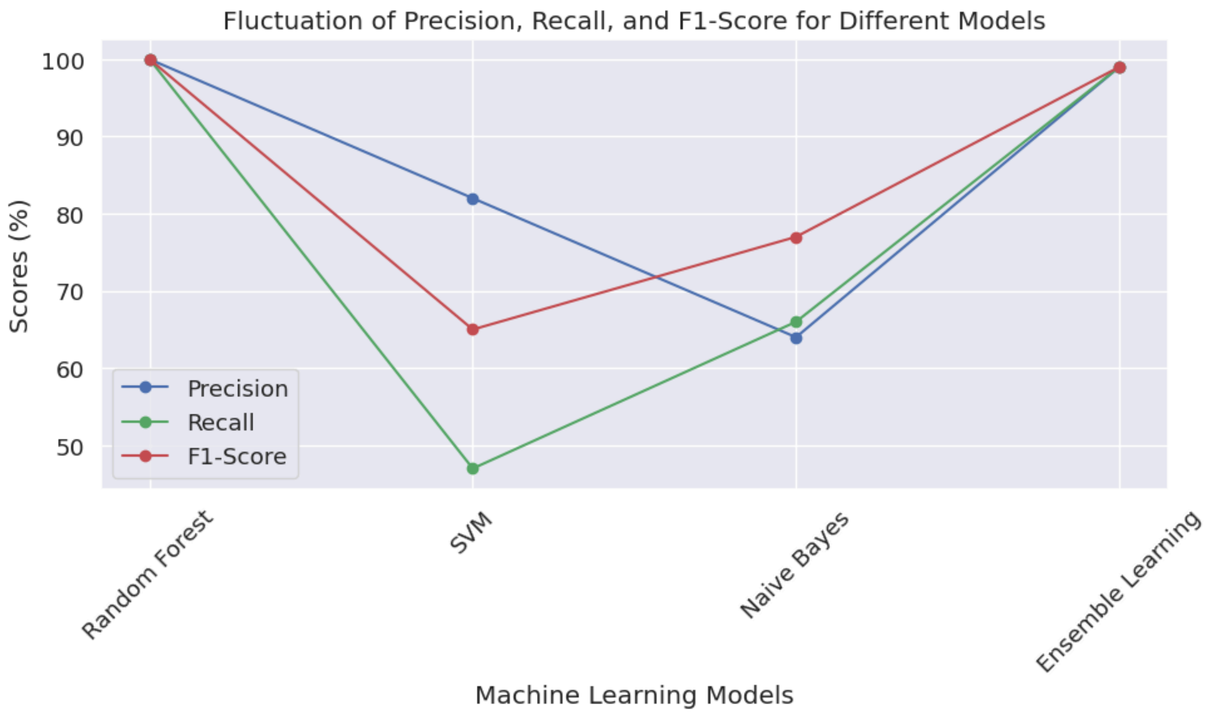


In [18]:

```
#Plot the evaluation metrics of each model in one figure

# Model names
models = ['Random Forest', 'SVM', 'Naive Bayes', 'Ensemble Learning']

# Precision scores
precision = [100, 82, 64, 99]

# Recall scores
recall = [100, 47, 66, 99]

# F1-score scores
f1_score = [100, 65, 77, 99]

# X-axis values (models)
x = range(len(models))

# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(10, 6))

# Plot precision scores
ax.plot(x, precision, marker='o', linestyle='-', color='b', label='Precision

# Plot recall scores
ax.plot(x, recall, marker='o', linestyle='-', color='g', label='Recall')
```

Loading [MathJax]/extensions/Safe.js

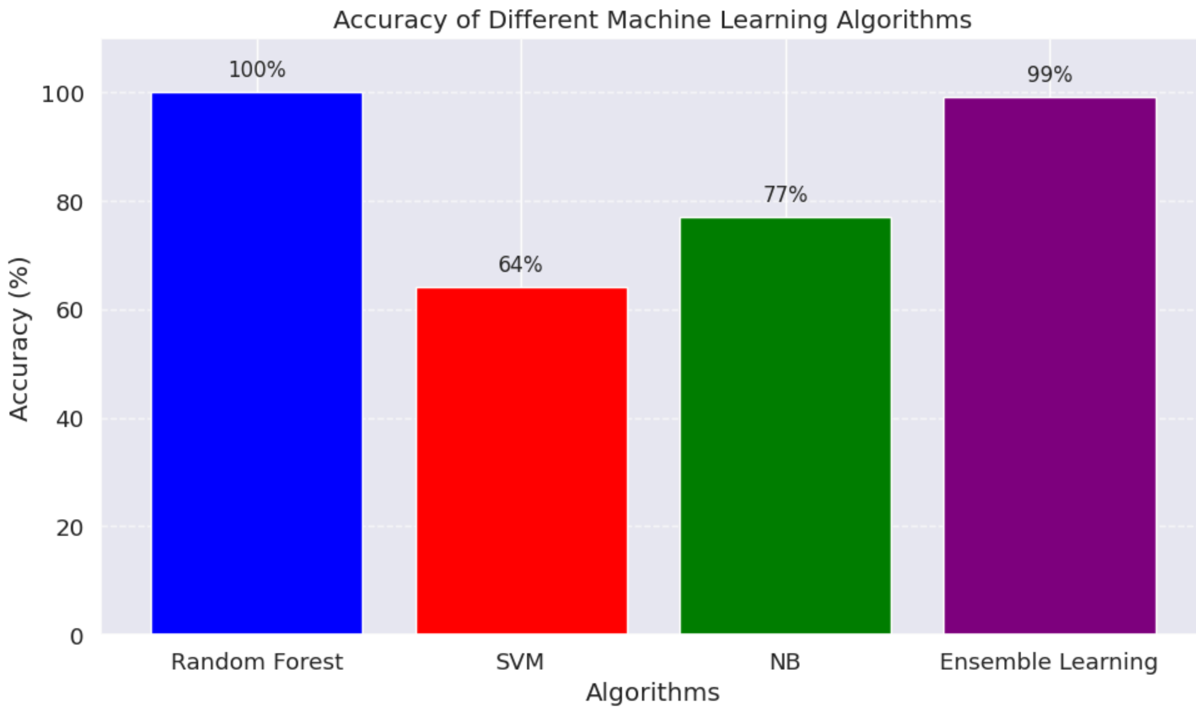
```
# Plot F1-score scores
ax.plot(x, f1_score, marker='o', linestyle='-', color='r', label='F1-Score')

# Set x-axis ticks and labels
ax.set_xticks(x)
ax.set_xticklabels(models, rotation=45)
ax.set_xlabel('Machine Learning Models')

# Set y-axis label
ax.set_ylabel('Scores (%)')

# Set plot title
ax.set_title('Fluctuation of Precision, Recall, and F1-Score for Different M

# Add a legend
ax.legend()

# Show the plot
plt.tight_layout()
plt.grid(True)
plt.show()
```



In [19]:

```
# Define the algorithms and their corresponding accuracies
algorithms = ['Random Forest', 'SVM', 'NB', 'Ensemble Learning']
accuracies = [100, 64, 77, 99]

# Create a bar graph
plt.figure(figsize=(10, 6))
plt.bar(algorithms, accuracies, color=['blue', 'red', 'green', 'purple'])
plt.ylim(0, 110)  # Set the y-axis limit for better visualization
plt.xlabel('Algorithms')
plt.ylabel('Accuracy (%)')
plt.title('Accuracy of Different Machine Learning Algorithms')
plt.grid(axis='y', linestyle='--', alpha=0.7)
```

```python
# Display the accuracy values on top of the bars
for i, v in enumerate(accuracies):
    plt.text(i, v + 2, str(v) + '%', ha='center', va='bottom', fontsize=12)

# Show the graph
plt.tight_layout()
plt.show()
```



In [ ]:

Loading [MathJax]/extensions/Safe.js