Importing dependencies for data preprocessing

In [1]:
```
import pandas as pd
import numpy as np
```

Reading dataset

In [2]:
```
df = pd.read_csv('data.csv', sep='|')
```

In [3]:
```
df
```

Out[3]:

	Name	md5	Machine
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332
...	...	...	...
138042	VirusShare_8e292b418568d6e7b87f2a32aee7074b	8e292b418568d6e7b87f2a32aee7074b	332
138043	VirusShare_260d9e2258aed4c8a3bbd703ec895822	260d9e2258aed4c8a3bbd703ec895822	332
138044	VirusShare_8d088a51b7d225c9f5d11d239791ec3f	8d088a51b7d225c9f5d11d239791ec3f	332
138045	VirusShare_4286dccf67ca220fe67635388229a9f3	4286dccf67ca220fe67635388229a9f3	332
138046	VirusShare_d7648eae45f09b3adb75127f43be6d11	d7648eae45f09b3adb75127f43be6d11	332

138047 rows × 57 columns

Separating attributes and target

In [4]:
```
x = df.drop(['legitimate','Name','md5'],axis='columns')
x
```

Out[4]:

	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	Size
0	332	224	258	9	0	
1	332	224	3330	9	0	
2	332	224	3330	9	0	
3	332	224	258	9	0	
4	332	224	258	9	0	
...	...	...	...	...	...	
138042	332	224	258	11	0	
138043	332	224	33167	2	25	
138044	332	224	258	10	0	
138045	332	224	33166	2	25	
138046	332	224	258	11	0	

138047 rows × 54 columns

Shape of x

In [5]:
```
x.shape
```

Out[5]:
```
(138047, 54)
```

In [6]:
```
y = df['legitimate'].values
y
```

Out[6]:
```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

Shape of y

In [7]:
```
y.shape
```

Out[7]:
```
(138047,)
```

Analysing features

In [8]:
```
i = 0
for col in x.columns:
    i += 1
    print(f"{i}) {col}")
```

```
1) Machine
2) SizeOfOptionalHeader
3) Characteristics
4) MajorLinkerVersion
5) MinorLinkerVersion
6) SizeOfCode
7) SizeOfInitializedData
8) SizeOfUninitializedData
9) AddressOfEntryPoint
10) BaseOfCode
11) BaseOfData
12) ImageBase
13) SectionAlignment
14) FileAlignment
15) MajorOperatingSystemVersion
16) MinorOperatingSystemVersion
17) MajorImageVersion
18) MinorImageVersion
19) MajorSubsystemVersion
20) MinorSubsystemVersion
21) SizeOfImage
22) SizeOfHeaders
23) CheckSum
24) Subsystem
25) DllCharacteristics
26) SizeOfStackReserve
27) SizeOfStackCommit
28) SizeOfHeapReserve
29) SizeOfHeapCommit
30) LoaderFlags
31) NumberOfRvaAndSizes
32) SectionsNb
33) SectionsMeanEntropy
34) SectionsMinEntropy
35) SectionsMaxEntropy
36) SectionsMeanRawsize
37) SectionsMinRawsize
38) SectionMaxRawsize
39) SectionsMeanVirtualsize
40) SectionsMinVirtualsize
41) SectionMaxVirtualsize
42) ImportsNbDLL
43) ImportsNb
44) ImportsNbOrdinal
45) ExportNb
46) ResourcesNb
47) ResourcesMeanEntropy
48) ResourcesMinEntropy
49) ResourcesMaxEntropy
50) ResourcesMeanSize
51) ResourcesMinSize
52) ResourcesMaxSize
53) LoadConfigurationSize
54) VersionInformationSize
```

Researching important feature from 54 total features

In [9]:
```
# importing dependencies for feature selection
```

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
```

In [10]:
```
sel = SelectFromModel(ExtraTreesClassifier())
sel.fit(x,y)
```

Out[10]:

SelectFromModel

SelectFromModel(estimator=ExtraTreesClassifier())

estimator: ExtraTreesClassifier

ExtraTreesClassifier()

ExtraTreesClassifier

ExtraTreesClassifier()

In [11]:
```
sel.get_support()

# It will return an array of boolean values.
# True for the features whose importance is greater than the mean importance
# and False for the rest
```

Out[11]:
```
array([ True,  True,  True, False, False, False, False, False, False,
       False, False,  True, False, False,  True, False, False, False,
        True, False, False, False, False,  True,  True, False, False,
       False, False, False, False, False, False,  True,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False,  True,  True, False, False, False, False,  True])
```

In [12]:
```
selected_features = x.columns[sel.get_support()]
```

In [13]:
```
selected_features
```

Out[13]:
```
Index(['Machine', 'SizeOfOptionalHeader', 'Characteristics', 'ImageBase',
       'MajorOperatingSystemVersion', 'MajorSubsystemVersion', 'Subsystem',
       'DllCharacteristics', 'SectionsMinEntropy', 'SectionsMaxEntropy',
       'ResourcesMinEntropy', 'ResourcesMaxEntropy', 'VersionInformationSize'],
      dtype='object')
```

In [14]:
```
len(selected_features)
```

Out[14]:
```
13
```

In [15]:
```
x_selft = sel.transform(x)
x_selft
```

Out[15]:
```
array([[3.32000000e+02, 2.24000000e+02, 2.58000000e+02, ...,
        2.56884382e+00, 3.53793936e+00, 1.60000000e+01],
       [3.32000000e+02, 2.24000000e+02, 3.33000000e+03, ...,
        3.42074425e+00, 5.08017686e+00, 1.80000000e+01],
       [3.32000000e+02, 2.24000000e+02, 3.33000000e+03, ...,
        2.84644859e+00, 5.27181276e+00, 1.80000000e+01],
       ...,
       [3.32000000e+02, 2.24000000e+02, 2.58000000e+02, ...,
        2.61702640e+00, 7.99048737e+00, 1.40000000e+01],
       [3.32000000e+02, 2.24000000e+02, 3.31660000e+04, ...,
        2.06096405e+00, 4.73974433e+00, 0.00000000e+00],
       [3.32000000e+02, 2.24000000e+02, 2.58000000e+02, ...,
        1.98048202e+00, 6.11537436e+00, 0.00000000e+00]])
```

In [16]:
```
x_selft.shape
```

Out[16]:
```
(138047, 13)
```

The 13 most important features identified

In [17]:
```
i = 0
for feat in selected_features:
    i += 1
    print(f"{i}) {feat}")
```

```
1) Machine
2) SizeOfOptionalHeader
3) Characteristics
4) ImageBase
5) MajorOperatingSystemVersion
6) MajorSubsystemVersion
7) Subsystem
8) DllCharacteristics
9) SectionsMinEntropy
10) SectionsMaxEntropy
11) ResourcesMinEntropy
12) ResourcesMaxEntropy
13) VersionInformationSize
```

Splitting the data into train and test splits

In [18]:
```
# importing dependency for train and test split

from sklearn.model_selection import train_test_split
```

In [19]:
```
x_train, x_test, y_train, y_test = train_test_split(x_selft, y , test_size = 0.2)
```

Shape of training and testing data

In [20]:
```
x_train.shape, x_test.shape
```

Out[20]:
```
((110437, 13), (27610, 13))
```

In [21]:
```
y_train.shape, y_test.shape
```

Out[21]:
```
((110437,), (27610,))
```

In [22]:
```
y_train = y_train.reshape(110437,)
y_test = y_test.reshape(27610,)
```

Comparing classification algorithms on the dataset

In [23]:
```
# importing models from sklearn

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBo
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

In [24]:
```
algos = {
        "DecisionTree": DecisionTreeClassifier(max_depth=10),
        "RandomForest": RandomForestClassifier(n_estimators=50),
        "GradientBoosting": GradientBoostingClassifier(n_estimators=50),
        "AdaBoost": AdaBoostClassifier(n_estimators=100),
        "GNB": GaussianNB()
    }
```

Training and validating models

In [25]:
```
# importing dependencies for validating models

from sklearn.metrics import classification_report, confusion_matrix
```

In [26]:
```
results = {}
cm = {}
for algo in algos:

    clf = algos[algo]
    clf.fit(x_train, y_train)

    y_pred = clf.predict(x_test)
    print(f"Classification Report for {algo} model :\n")
    print(classification_report(y_test,y_pred,zero_division=0),"\n")

    score = clf.score(x_test, y_test)
    print("%s : %f %%" % (algo, score*100))
    print("\n","-"*100,"\n")

    results[algo] = score
    cm[algo] = confusion_matrix(y_test,y_pred)

winner = max(results, key=results.get)
print('\nWinner algorithm is %s with a %f %% accuracy' % (winner, results[winner]*100)
```

```
Classification Report for DecisionTree model :

              precision    recall  f1-score   support

           0       0.99      0.99      0.99     19320
           1       0.98      0.98      0.98      8290

    accuracy                           0.99     27610
   macro avg       0.99      0.99      0.99     27610
weighted avg       0.99      0.99      0.99     27610


DecisionTree : 99.054690 %

 ------------------------------------------------------------------------------
---------------

Classification Report for RandomForest model :

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     19320
           1       0.99      0.99      0.99      8290

    accuracy                           0.99     27610
   macro avg       0.99      0.99      0.99     27610
weighted avg       0.99      0.99      0.99     27610


RandomForest : 99.427744 %

 ------------------------------------------------------------------------------
---------------

Classification Report for GradientBoosting model :

              precision    recall  f1-score   support

           0       0.99      0.99      0.99     19320
           1       0.98      0.98      0.98      8290

    accuracy                           0.99     27610
   macro avg       0.99      0.99      0.99     27610
weighted avg       0.99      0.99      0.99     27610


GradientBoosting : 98.783050 %

 ------------------------------------------------------------------------------
---------------

Classification Report for AdaBoost model :

              precision    recall  f1-score   support

           0       0.99      0.99      0.99     19320
           1       0.98      0.98      0.98      8290

    accuracy                           0.99     27610
   macro avg       0.98      0.98      0.98     27610
```

```
weighted avg       0.99      0.99      0.99     27610

```

```
AdaBoost : 98.649040 %

 ------------------------------------------------------------------------------
---------------
```

```
Classification Report for GNB model :

              precision    recall  f1-score   support

           0       0.70      1.00      0.82     19320
           1       1.00      0.00      0.00      8290

    accuracy                           0.70     27610
   macro avg       0.85      0.50      0.41     27610
weighted avg       0.79      0.70      0.58     27610

```

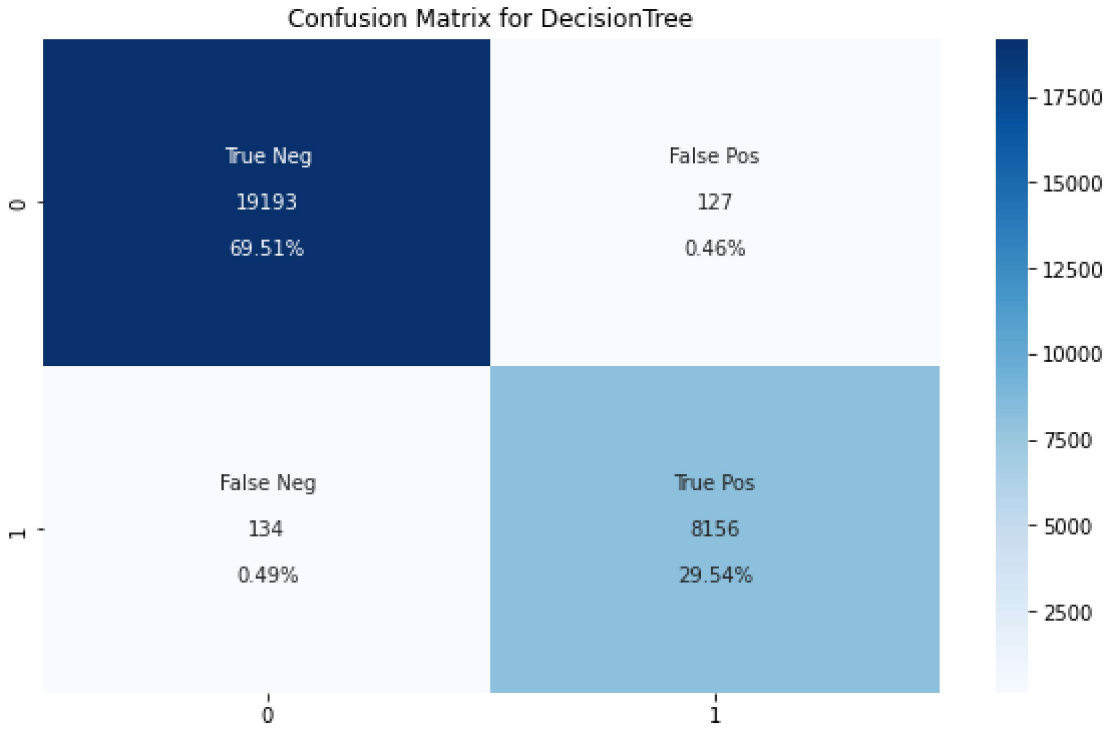
```
GNB : 69.981891 %

 ------------------------------------------------------------------------------
---------------
```

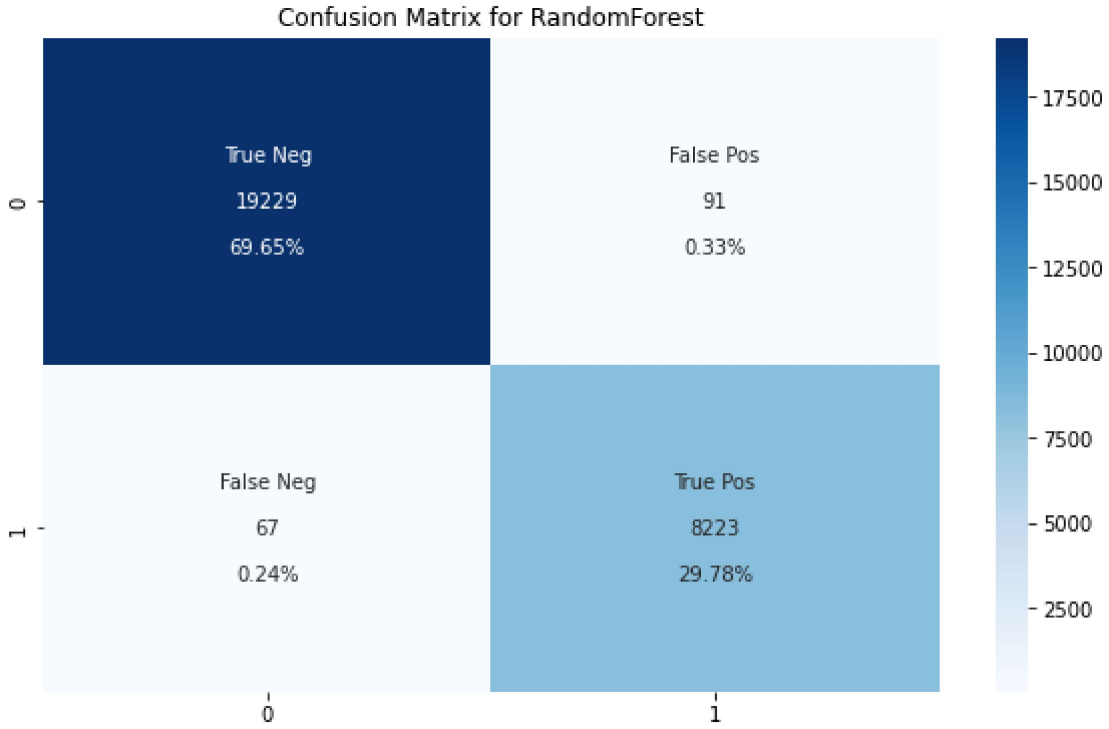
```
Winner algorithm is RandomForest with a 99.427744 % accuracy
```

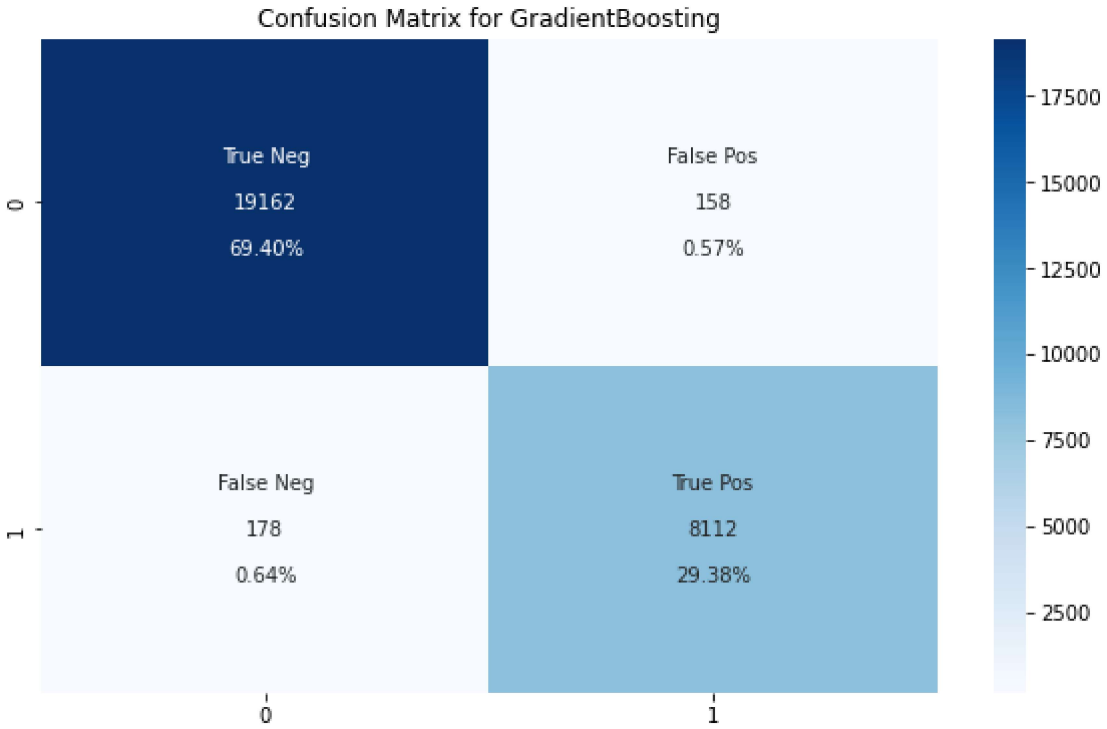
Visualizing Results

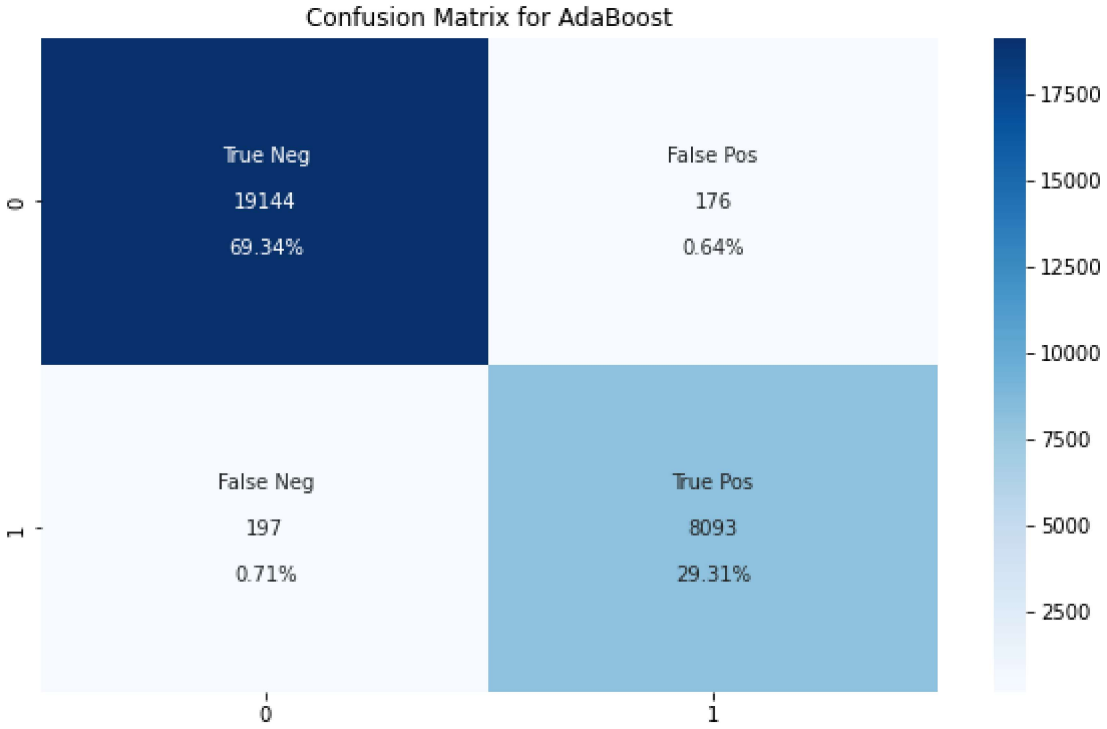
In [27]:
```
# importing dependency for visualizing

import seaborn as sns
import matplotlib.pyplot as plt
```

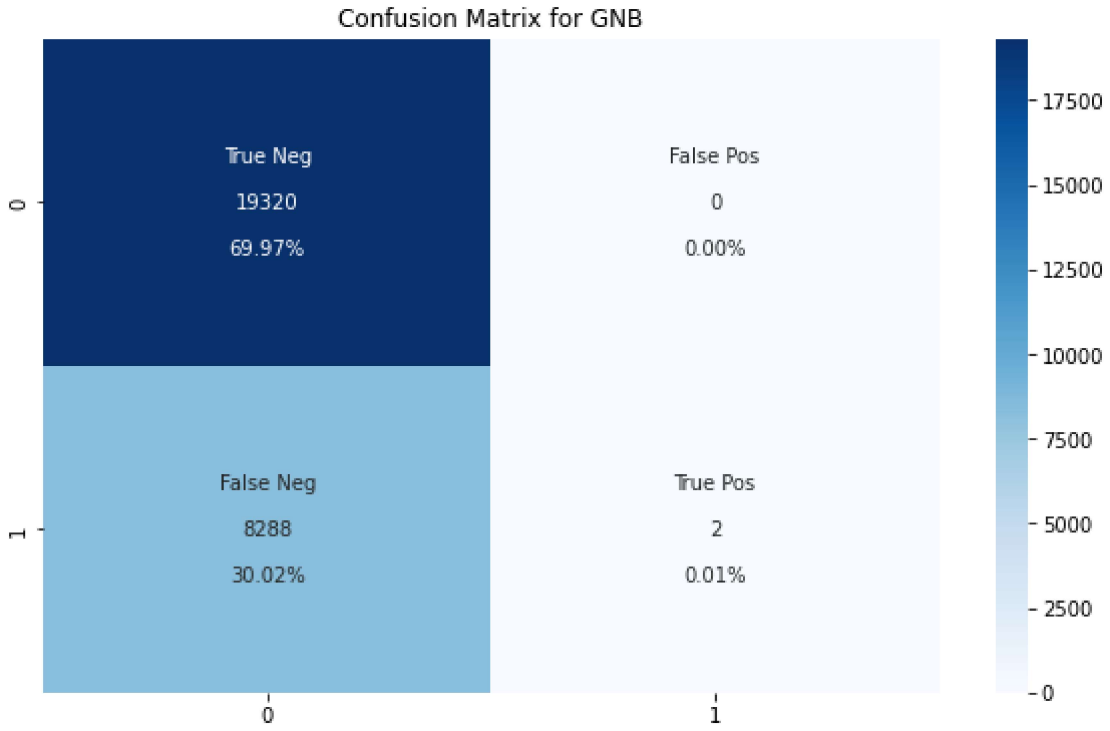
In [28]:
```
algo_names = [
        "DecisionTree",
        "RandomForest",
        "GradientBoosting",
        "AdaBoost",
        "GNB"
]
i=0
for cfm in cm:
    plt.figure(figsize=(10,6))
    plt.title(f"Confusion Matrix for {algo_names[i]}")
    i += 1
    group_names = ['True Neg','False Pos','False Neg','True Pos']
    group_counts = ["{0:0.0f}".format(value) for value in
                    cm[cfm].flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm[cfm].flatten()/np.sum(cm[cfm])]
    labels = [f"{v1}\n\n{v2}\n\n{v3}" for v1, v2, v3 in
              zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cm[cfm], annot=labels, fmt='', cmap='Blues')
```


Confusion Matrix for DecisionTree
True Neg
19193
69.51%
False Pos
127
0.46%
False Neg
134
0.49%
True Pos
8156
29.54%
0
1
0
1
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix for RandomForest
True Neg
19229
69.65%
False Pos
91
0.33%
False Neg
67
0.24%
True Pos
8223
29.78%
0
1
0
1
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix for GradientBoosting
True Neg
19162
69.40%
False Pos
158
0.57%
False Neg
178
0.64%
True Pos
8112
29.38%
0
1
0
1
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix for AdaBoost
True Neg
19144
69.34%
False Pos
176
0.64%
False Neg
197
0.71%
True Pos
8093
29.31%
0
1
0
1
17500
15000
12500
10000
7500
5000
2500



In [29]:
```
df[selected_features]
```

Out[29]:

	Machine	SizeOfOptionalHeader	Characteristics	ImageBase	MajorOperatingSystemVersion
0	332	224	258	4194304.0	0
1	332	224	3330	771751936.0	5
2	332	224	3330	771751936.0	5
3	332	224	258	771751936.0	5
4	332	224	258	771751936.0	5
...	...	...	...	...	...
138042	332	224	258	4194304.0	5
138043	332	224	33167	4194304.0	1
138044	332	224	258	4194304.0	5
138045	332	224	33166	4194304.0	1
138046	332	224	258	4194304.0	5

138047 rows × 13 columns

In [30]:
```
features = [x for x in df[selected_features].columns]
features
```

Out[30]:
```
['Machine',
 'SizeOfOptionalHeader',
 'Characteristics',
 'ImageBase',
 'MajorOperatingSystemVersion',
 'MajorSubsystemVersion',
 'Subsystem',
 'DllCharacteristics',
 'SectionsMinEntropy',
 'SectionsMaxEntropy',
 'ResourcesMinEntropy',
 'ResourcesMaxEntropy',
 'VersionInformationSize']
```

In [31]:
```
algos[winner]
```

Out[31]:
```
RandomForestClassifier(n_estimators=50)
```

Saving model and important features

In [32]:
```
import pickle
with open('model_feats','wb') as f:
    pickle.dump([algos[winner],features],f)
```