In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, ac
from imblearn.over_sampling import SMOTE
from sklearn.pipeline import Pipeline
```

In [2]:
```
column_names = ['duration',                ' protocol_type',
                           ' service',                            ' flag',
                         ' src_bytes',                       ' dst_bytes',
                              ' land',                  ' wrong_fragment',
                            ' urgent',                             ' hot',
                 ' num_failed_logins',                       ' logged_in',
                   ' num_compromised',                      ' root_shell',
                      ' su_attempted',                        ' num_root',
                ' num_file_creations',                      ' num_shells',
                  ' num_access_files',                ' num_outbound_cmds',
                     ' is_host_login',                   ' is_guest_login',
                             ' count',                       ' srv_count',
                       ' serror_rate',                   ' srv_error_rate',
                       ' rerror_rate',                  ' srv_rerror_rate',
                     ' same_srv_rate',                   ' diff_srv_rate',
                ' srv_diff_host_rate',                  ' dst_host_count',
                ' dst_host_srv_count',          ' dst_host_same_srv_rate',
            ' dst_host_diff_srv_rate', ' dst_host_same_src_port_rate',
       ' dst_host_srv_diff_host_rate',             ' dst_host_serror_rate',
          ' dst_host_srv_serror_rate',             ' dst_host_rerror_rate',
                ' dst_host_srv_rerror_rate']
```

In [3]:
```
file_paths = [
    'Data_of_Attack_Back.csv',
    'Data_of_Attack_Back_BufferOverflow.csv',
    'Data_of_Attack_Back_FTPWrite.csv',
    'Data_of_Attack_Back_GuessPassword.csv',
    'Data_of_Attack_Back_Neptune.csv',
    'Data_of_Attack_Back_NMap.csv',
    'Data_of_Attack_Back_Normal.csv',
    'Data_of_Attack_Back_PortSweep.csv',
    'Data_of_Attack_Back_RootKit.csv',
    'Data_of_Attack_Back_Satan.csv',
    'Data_of_Attack_Back_Smurf.csv',
]

```

In [4]:
```

labels = ['Back', 'BufferOverflow', 'FTPWrite', 'GuessPassword', 'Neptu


```

In [5]:
```
dataframes = []
for file_paths, label in zip(file_paths, labels):
    if label == 'FTPWrite':
        df = pd.read_csv(file_paths, header=None, names=column_names)
    else:
        df = pd.read_csv(file_paths)

    df['Label'] = label
    dataframes.append(df)


```

In [6]:
```
# Combine into a single DataFrame
df = pd.concat(dataframes, ignore_index=True)
```

In [7]:
```
df
```

Out[7]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment
0	0.0	0.00	0.00	0.0	0.54540	0.08314	0	0.0
1	0.0	0.00	0.00	0.0	0.54540	0.08314	0	0.0
2	0.0	0.00	0.00	0.0	0.54540	0.08314	0	0.0
3	0.0	0.00	0.00	0.0	0.54540	0.08314	0	0.0
4	0.0	0.00	0.00	0.0	0.54540	0.08314	0	0.0
...	...	...	...	...	...	...	...	...
817546	0.0	0.02	0.09	0.0	0.01032	0.00000	0	0.0
817547	0.0	0.02	0.09	0.0	0.01032	0.00000	0	0.0
817548	0.0	0.02	0.09	0.0	0.01032	0.00000	0	0.0
817549	0.0	0.02	0.09	0.0	0.01032	0.00000	0	0.0
817550	0.0	0.01	0.12	0.0	0.00028	0.00000	0	0.3

817551 rows × 42 columns

Data Preprocessing

In [8]:
```
# Drop rows with missing values or fill them
df.dropna(inplace=True)  # or data.fillna(0, inplace=True)

# Split data into features and target
X = df.drop('Label', axis=1)
y = df['Label']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

```

In [9]:
```
#Handling Imbalanced Data
```

In [10]:
```
smote = SMOTE()
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)

```

In [11]:
```
#Feature Scaling
scaler = StandardScaler()
X_train_sm = scaler.fit_transform(X_train_sm)
X_test = scaler.transform(X_test)

```

In [12]:
```
#Model Training
```

In [13]:
```
clf = RandomForestClassifier()
```

In [14]:
```
clf.fit(X_train_sm, y_train_sm)
```

Out[14]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

Model Evaluation

In [15]:
```
y_pred = clf.predict(X_test)
```

In [16]:
```
# Print classification report and accuracy
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
C:\Users\Asus\anaconda3\Lib\site-packages\sklearn\metrics\_classification.
py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Asus\anaconda3\Lib\site-packages\sklearn\metrics\_classification.
py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Asus\anaconda3\Lib\site-packages\sklearn\metrics\_classification.
py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

	precision	recall	f1-score	support
Back	0.99	1.00	1.00	192
BufferOverflow	1.00	0.62	0.77	8
FTPWrite	1.00	1.00	1.00	1
GuessPassWord	1.00	1.00	1.00	9
NMap	0.99	1.00	1.00	301
Neptune	1.00	1.00	1.00	45575
Normal	1.00	1.00	1.00	115143
PortSweep	1.00	1.00	1.00	582
RootKit	0.00	0.00	0.00	3
Satan	1.00	1.00	1.00	1098
Smurf	1.00	1.00	1.00	599
accuracy			1.00	163511
macro avg	0.91	0.87	0.89	163511
weighted avg	1.00	1.00	1.00	163511

```
Accuracy: 0.9998899156631664
```

In [17]:
```
# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[   192      0      0      0      0      0      0      0      0      0
      0]
 [     0      5      0      0      0      0      3      0      0      0
      0]
 [     0      0      1      0      0      0      0      0      0      0
      0]
 [     0      0      0      9      0      0      0      0      0      0
      0]
 [     0      0      0      0    301      0      0      0      0      0
      0]
 [     0      0      0      0      0  45575      0      0      0      0
      0]
 [     1      0      0      0      2      0 115137      1      0      2
      0]
 [     0      0      0      0      0      0      1    581      0      0
      0]
 [     0      0      0      0      0      0      2      0      0      1
      0]
 [     0      0      0      0      0      0      4      1      0   1093
      0]
 [     0      0      0      0      0      0      0      0      0      0
    599]]
```

Q1.Binomial classification: Detect anomalies by predicting Activity is normal or attack

In [18]:
```
#Create a Binary Target Variable
df['Binary_Label'] = df['Label'].apply(lambda x: 0 if x == 'Normal' els
```

In [19]:
```
# Features (drop the 'Label' and 'Binary_Label' columns)
X = df.drop(['Label', 'Binary_Label'], axis=1)
```

In [20]:
```
# Binary target variable
y_bin = df['Binary_Label']
```

In [21]:
```
# Split data into training and testing sets
X_train_bin, X_test_bin, y_train_bin, y_test_bin = train_test_split(X,
```

In [22]:
```
#Train a Model
from sklearn.linear_model import LogisticRegression
```

In [23]:
```
# Initialize the Logistic Regression model
log_reg_bin = LogisticRegression(max_iter=1000)  # Increase max_iter i
```

In [24]:

```python
# Fit the model to the training data
log_reg_bin.fit(X_train_bin, y_train_bin)
```

Out[24]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [25]:

```python
# Predict on the testing set
y_pred_bin = log_reg_bin.predict(X_test_bin)
```

In [26]:

```python
#Evaluate the Model

# Print the classification report to see precision, recall, and F1-scor
print(classification_report(y_test_bin, y_pred_bin))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    115143
           1       1.00      0.99      0.99     48368

    accuracy                           0.99    163511
   macro avg       1.00      0.99      0.99    163511
weighted avg       0.99      0.99      0.99    163511
```

In [27]:

```python
# Print the confusion matrix
print(confusion_matrix(y_test_bin, y_pred_bin))
```

```
[[114954    189]
 [   649  47719]]
```

Precision for both classes (normal activities and attacks) is very high, near or at 1.00, indicating that the model has a very high accuracy in predicting positive samples.

Recall is also impressive, especially for the normal activities (1.00), indicating that the model is almost perfect in identifying all the actual normal activities. For attacks, the recall is slightly lower (0.99), but still very high, indicating that the model identifies most of the actual attacks correctly.

F1-score, which is the harmonic mean of precision and recall, is near perfect for both classes, reinforcing the model's balanced performance in terms of precision and recall.

The confusion matrix further clarifies the results:

Out of 115,143 true normal activities, 114,954 were correctly classified as normal, with only 189 misclassified as attacks. Out of 48,368 true attacks, 47,719 were correctly identified, with 649 misclassified as normal activities. The accuracy of 0.99 suggests that the model correctly predicts the class for 99% of the cases in your test set.

In [ ]:

```python

```

Q2 . Multinomial Classification: Detecting type of activity by predicting Activity is Normal or Back or Buffer Over flow or FTP Write or Guess Password or Neptune or N-Map or Port Sweep or Root Kit or Satan or Smurf

In [28]:
```
# Separate features and target variable
X = df.drop('Label', axis=1)  # Drop the 'Label' column to get the feat
y = df['Label']  # Target variable is the activity type
```

In [29]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [30]:
```
# Create a pipeline that first standardizes the data then applies the c
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier())
])
```

In [31]:
```
# Train the model
pipeline.fit(X_train, y_train)
```

Out[31]:
```
▸ Pipeline
  ▸ StandardScaler
▸ RandomForestClassifier
```

In [32]:
```
# Predict on the test set
y_pred = pipeline.predict(X_test)
```

In [33]:

```
# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred, labels=['Normal', 'Back', 'Buffe

```

```
Classification Report:
                precision    recall  f1-score   support

           Back       1.00      1.00      1.00       192
 BufferOverflow       1.00      0.88      0.93         8
       FTPWrite       0.50      1.00      0.67         1
  GuessPassWord       1.00      1.00      1.00         9
           NMap       0.99      1.00      1.00       301
        Neptune       1.00      1.00      1.00     45575
         Normal       1.00      1.00      1.00    115143
      PortSweep       1.00      1.00      1.00       582
        RootKit       1.00      0.33      0.50         3
          Satan       1.00      1.00      1.00      1098
          Smurf       1.00      1.00      1.00       599

       accuracy                           1.00    163511
      macro avg       0.95      0.93      0.92    163511
   weighted avg       1.00      1.00      1.00    163511

Confusion Matrix:
[[115143      0      0      0      0      0      0      0      0      0
       0]
 [     0    192      0      0      0      0      0      0      0      0
       0]
 [     0      0      0      0      0      0      0      0      0      0
       0]
 [     0      0      0      0      0      0      0      0      0      0
       0]
 [     0      0      0      0      0      0      0      0      0      0
       0]
 [     0      0      0      0      0  45575      0      0      0      0
       0]
 [     0      0      0      0      0      0      0      0      0      0
       0]
 [     0      0      0      0      0      0      0      0      0      0
       0]
 [     0      0      0      0      0      0      0      0      0      0
       0]
 [     0      0      0      0      0      0      0      0      0   1096
       0]
 [     0      0      0      0      0      0      0      0      0      0
     599]]
```

In [34]:
```
# Print confusion matrix
print("Confusion Matrix for Binary Classification:")
print(confusion_matrix(y_test_bin, y_pred_bin))

```

```
Confusion Matrix for Binary Classification:
[[114954    189]
 [   649  47719]]
```

In [35]:

```
# Print classification report
print("Classification Report for Multinomial Classification:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("Confusion Matrix for Multinomial Classification:")
print(confusion_matrix(y_test, y_pred, labels=labels))

```

```
Classification Report for Multinomial Classification:
                precision    recall  f1-score   support

          Back       1.00      1.00      1.00       192
BufferOverflow       1.00      0.88      0.93         8
      FTPWrite       0.50      1.00      0.67         1
 GuessPassWord       1.00      1.00      1.00         9
          NMap       0.99      1.00      1.00       301
       Neptune       1.00      1.00      1.00     45575
        Normal       1.00      1.00      1.00    115143
     PortSweep       1.00      1.00      1.00       582
       RootKit       1.00      0.33      0.50         3
         Satan       1.00      1.00      1.00      1098
         Smurf       1.00      1.00      1.00       599

      accuracy                           1.00    163511
     macro avg       0.95      0.93      0.92    163511
  weighted avg       1.00      1.00      1.00    163511

Confusion Matrix for Multinomial Classification:
[[   192      0      0      0      0      0      0      0      0      0
       0]
 [     0      7      1      0      0      0      0      0      0      0
       0]
 [     0      0      1      0      0      0      0      0      0      0
       0]
 [     0      0      0      9      0      0      0      0      0      0
       0]
 [     0      0      0      0  45575      0      0      0      0      0
       0]
 [     0      0      0      0      0    301      0      0      0      0
       0]
 [     0      0      0      0      0      0 115143      0      0      0
       0]
 [     0      0      0      0      0      0      0    582      0      0
       0]
 [     0      0      0      0      0      1      0      0      1      1
       0]
 [     0      0      0      0      0      1      0      1      0   1096
       0]
 [     0      0      0      0      0      0      0      0      0      0
     599]]
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:
```
1
```

Overall Performance:

High Accuracy: The model achieves near-perfect accuracy (100%) across the board, which is excellent for a multiclass classification problem. Precision and Recall: For most activity types, both precision and recall are very high, often reaching 1.00, indicating the model's strong capability to correctly identify and classify different types of network activities.

Observations:

1.Neptune and Normal Activities: The model performs exceptionally well in identifying 'Neptune' and 'Normal' activities, which have the highest number of instances, with perfect precision and recall scores.

2.Buffer Overflow and RootKit: These categories have lower sample sizes and show some variation in recall scores ('BufferOverflow' at 0.88 and 'RootKit' at 0.33), suggesting the model may struggle slightly more with these less-represented classes.

3.FTPWrite: Despite having only one instance in the test set, the model identified it correctly, though the precision is lower (0.50) due to the model's overprediction in this category.

Areas for Improvement:

1.Handling Rare Classes: The variance in performance for 'BufferOverflow' and 'RootKit' points to potential challenges in handling rare classes. Techniques like oversampling, synthetic data generation (SMOTE), or cost-sensitive learning might improve performance in these categories.

2.FTPWrite Misclassification: The model's overprediction for 'FTPWrite' suggests a need for further investigation. It might be beneficial to explore feature relevance for this category or adjust class weighting to mitigate this bias.