```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,VotingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,recall_score,precision_score,confusion_matrix
```

```
df = pd.read_csv("/content/Training _Data.csv")
df
```

Show hidden output

```
df.drop(['source_ip', 'destination_ip', 'Index'], axis=1, inplace=True)
df
```

Show hidden output

```
le = LabelEncoder()
df['protocol'] = le.fit_transform(df['protocol'])
```

```
df
```

Show hidden output

```
df['Traffic_Label'] = df['Traffic_Label'].map({'Normal Traffic': 1, 'DDoS Traffic': 2})
```

```
scaler = MinMaxScaler()
```

```
scaler = MinMaxScaler()
features_to_scale = ['flow_duration', 'mean_forward_iat',
                     'min_forward_iat','max_forward_iat',
                     'std_forward_iat', 'mean_backward_iat',
                         'min_backward_iat', 'max_backward_iat',
                     'std_backward_iat',
                         'mean_flow_iat', 'min_flow_iat',
                     'max_flow_iat', 'std_flow_iat',
                         'mean_active_time', 'min_active_time',
                     'max_active_time',
                         'std_active_time', 'mean_idle_time', 'min_idle_time',
                         'max_idle_time', 'std_idle_time']
```

```
df[features_to_scale] = scaler.fit_transform(df[features_to_scale])
```

```
X = df.drop('Traffic_Label', axis=1)
y = df['Traffic_Label']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
svc=SVC(kernel='linear')
```

```
svc.fit(X_train,y_train)
```

▸ SVC

```
svc_pred=svc.predict(X_test)
```

```
svc_accuracy=accuracy_score(y_test,svc_pred)
svc_recall=recall_score(y_test,svc_pred)
svc_precision=precision_score(y_test,svc_pred)
print("SVC Accuracy Score:", svc_accuracy)
print("SVC Recall Score:", svc_recall)
print("SVC Precision Score:", svc_precision)
```

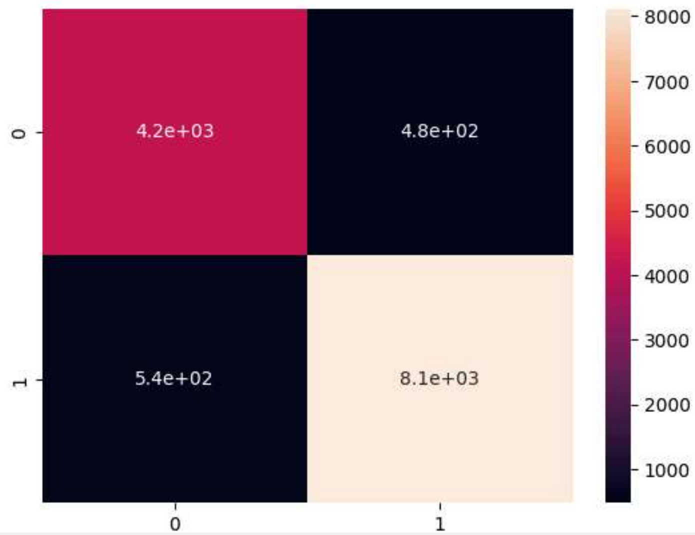
```
SVC Accuracy Score: 0.923464022833108
SVC Recall Score: 0.8976784178847808
SVC Precision Score: 0.8849332485696122
```

```
svc_conf=confusion_matrix(y_test,svc_pred)
svc_conf
```

```
array([[4176,  476],
       [ 543, 8119]])
```

```
import seaborn as sns
sns.heatmap(svc_conf, annot=True)
```

```
<Axes: >
```




```
lr=LogisticRegression()
lr.fit(X_train,y_train)
```

```
▸ LogisticRegression
```

```
lr_pred=lr.predict(X_test)
```

```
lr_accuracy=accuracy_score(y_test,lr_pred)
lr_recall=recall_score(y_test,lr_pred)
lr_precision=precision_score(y_test,lr_pred)
print("Logistic Regression Accuracy Score:", lr_accuracy)
print("Logistic Regression Recall Score:", lr_recall)
print("Logistic Regression Precision Score:", lr_precision)
```

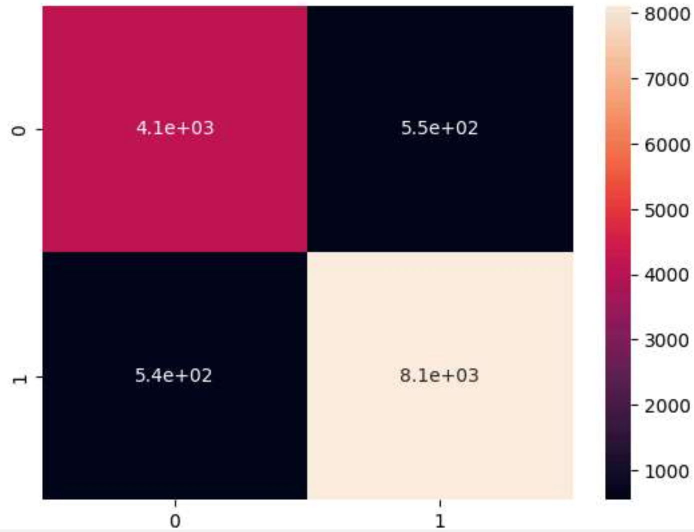
```
Logistic Regression Accuracy Score: 0.9177557458314556
Logistic Regression Recall Score: 0.88134135855546
Logistic Regression Precision Score: 0.8830497523153134
```

```
lr_conf=confusion_matrix(y_test,lr_pred)
lr_conf
```

```
array([[4100,  552],
       [ 543, 8119]])
```

```
sns.heatmap(lr_conf, annot=True)
```

```
<Axes: >
```




```
dtc=DecisionTreeClassifier(criterion='entropy')
dtc.fit(X_train,y_train)
```

```
▸ DecisionTreeClassifier
```

```
dtc_pred=dtc.predict(X_test)
```

```
dtc_accuracy=accuracy_score(y_test,dtc_pred)
dtc_recall=recall_score(y_test,dtc_pred)
dtc_precision=precision_score(y_test,dtc_pred)
print("Decision Tree Accuracy Score:", dtc_accuracy)
print("Decision Tree Recall Score:", dtc_recall)
print("Decision Tree Precision Score:", dtc_precision)
```

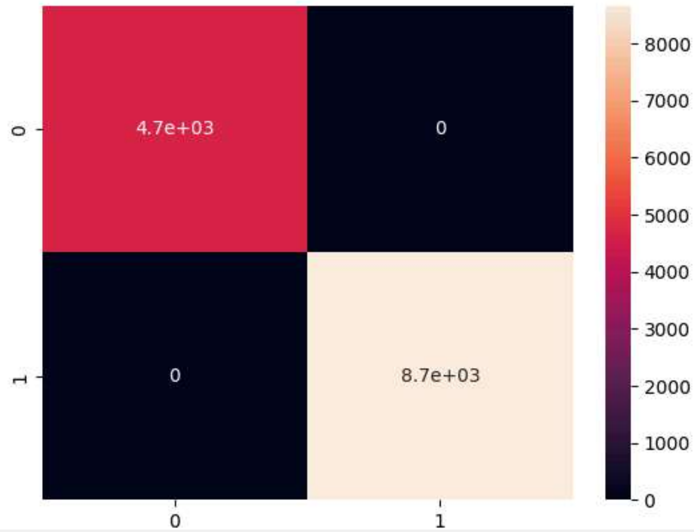
```
Decision Tree Accuracy Score: 1.0
Decision Tree Recall Score: 1.0
Decision Tree Precision Score: 1.0
```

```
dtc_conf=confusion_matrix(y_test,dtc_pred)
dtc_conf
```

```
array([[4652,    0],
       [   0, 8662]])
```

```
sns.heatmap(dtc_conf, annot=True)
```

```
<Axes: >
```




```
rfc=RandomForestClassifier(n_estimators=20)
rfc.fit(X_train,y_train)
```

```
▸ RandomForestClassifier
```

```
rfc_pred=rfc.predict(X_test)
```

```
rfc_accuracy=accuracy_score(y_test,rfc_pred)
rfc_recall=recall_score(y_test,rfc_pred)
rfc_precision=precision_score(y_test,rfc_pred)
print("Random Forest Accuracy Score:", rfc_accuracy)
print("Random Forest Recall Score:", rfc_recall)
print("Random Forest Precision Score:", rfc_precision)
```

```
Random Forest Accuracy Score: 0.9998497821841671
Random Forest Recall Score: 1.0
Random Forest Precision Score: 0.9995702621400946
```

```
rfc_conf=confusion_matrix(y_test,rfc_pred)
rfc_conf
```

```
array([[4652,    0],
       [   2, 8660]])
```

```
sns.heatmap(rfc_conf, annot=True)
```

```
<Axes: >
```

```
model_1 = XGBClassifier()
model_2 = RandomForestClassifier(n_estimators=20)
model_3 = DecisionTreeClassifier()
model_4=SVC()
```

```
ensemble_model = VotingClassifier(
    estimators=[ ('xgb', model_1), ('rf', model_2),('dtc',model_3),('svc',model_4)], voting='hard')
```

```
ensemble_model.fit(X_train, y_train)
```

```
VotingClassifier
xgb                                                   rf
XGBClassifier                                         RandomForestClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,        RandomForestClassifier(n_estimators=20)    Decisi
              colsample_bylevel=None, colsample_bynode=None,                                                   Decisio
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
ensemble_pred = ensemble_model.predict(X_test)
```

```
ensemble_accuracy=accuracy_score(y_test,ensemble_pred)
ensemble_recall=recall_score(y_test,ensemble_pred)
ensemble_precision=precision_score(y_test,ensemble_pred)
print("Ensemble Learning Accuracy Score:", ensemble_accuracy)
print("Ensemble Learning Recall Score:", ensemble_recall)
print("Ensemble Learning Precision Score:", ensemble_precision)
```

```
"""Cons. of using Ensemble Learning in this problem
Ensemble model is  more complex and harder to interpret than single models.
So as we can see Decision Tree Classifer is working well in this problem.
So we can use Decision Tree Classifier for this problem.
Decision Tree Accuracy Score: 1.0
Decision Tree Recall Score: 1.0
Decision Tree Precision Score: 1.0

Managing and deploying multiple models will be tough in production environment.
"""
```