```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import imblearn

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

#Load Data
train = pd.read_csv("/Train_data.csv")
test = pd.read_csv("/Test_data.csv")

print(train.head(4))

print("Training data has {} rows & {} columns".format(train.shape[0],train.shape[1]))

print(test.head(4))

print("Testing data has {} rows & {} columns".format(test.shape[0],test.shape[1]))
```

```
   duration protocol_type  ... dst_host_srv_rerror_rate    class
0         0           tcp  ...                     0.00   normal
1         0           udp  ...                     0.00   normal
2         0           tcp  ...                     0.00  anomaly
3         0           tcp  ...                     0.01   normal

[4 rows x 42 columns]
Training data has 25192 rows & 42 columns
   duration protocol_type  ... dst_host_rerror_rate dst_host_srv_rerror_rate
0         0           tcp  ...                  1.0                      1.0
1         0           tcp  ...                  1.0                      1.0
2         2           tcp  ...                  0.0                      0.0
3         0          icmp  ...                  0.0                      0.0

[4 rows x 41 columns]
Testing data has 22544 rows & 41 columns
```

```
#Exploratory Analysis
# Descriptive statistics
train.describe()

print(train['num_outbound_cmds'].value_counts())
print(test['num_outbound_cmds'].value_counts())

#'num_outbound_cmds' is a redundant column so remove it from both train & test datasets
train.drop(['num_outbound_cmds'], axis=1, inplace=True)
test.drop(['num_outbound_cmds'], axis=1, inplace=True)
```

```
# Attack Class Distribution
train['class'].value_counts()
```

```
0    25192
Name: num_outbound_cmds, dtype: int64
0    22544
Name: num_outbound_cmds, dtype: int64
normal     13449
anomaly    11743
Name: class, dtype: int64
```

```
#Scalling numerical attributes
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit variance
cols = train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(test.select_dtypes(include=['float64','int64']))

# turn the result back to a dataframe
sc_traindf = pd.DataFrame(sc_train, columns = cols)
sc_testdf = pd.DataFrame(sc_test, columns = cols)
```

```
#Encoding categorical attributes
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

# extract categorical attributes from both training and test sets
cattrain = train.select_dtypes(include=['object']).copy()
cattest = test.select_dtypes(include=['object']).copy()

# encode the categorical attributes
traincat = cattrain.apply(encoder.fit_transform)
testcat = cattest.apply(encoder.fit_transform)

# separate target column from encoded data
enctrain = traincat.drop(['class'], axis=1)
cat_Ytrain = traincat[['class']].copy()
```

```
#Union of processed numerical and categorical data
train_x = pd.concat([sc_traindf,enctrain],axis=1)
train_y = cat_Ytrain
train_x.shape

test_df = pd.concat([sc_testdf,testcat],axis=1)
test_df.shape
```

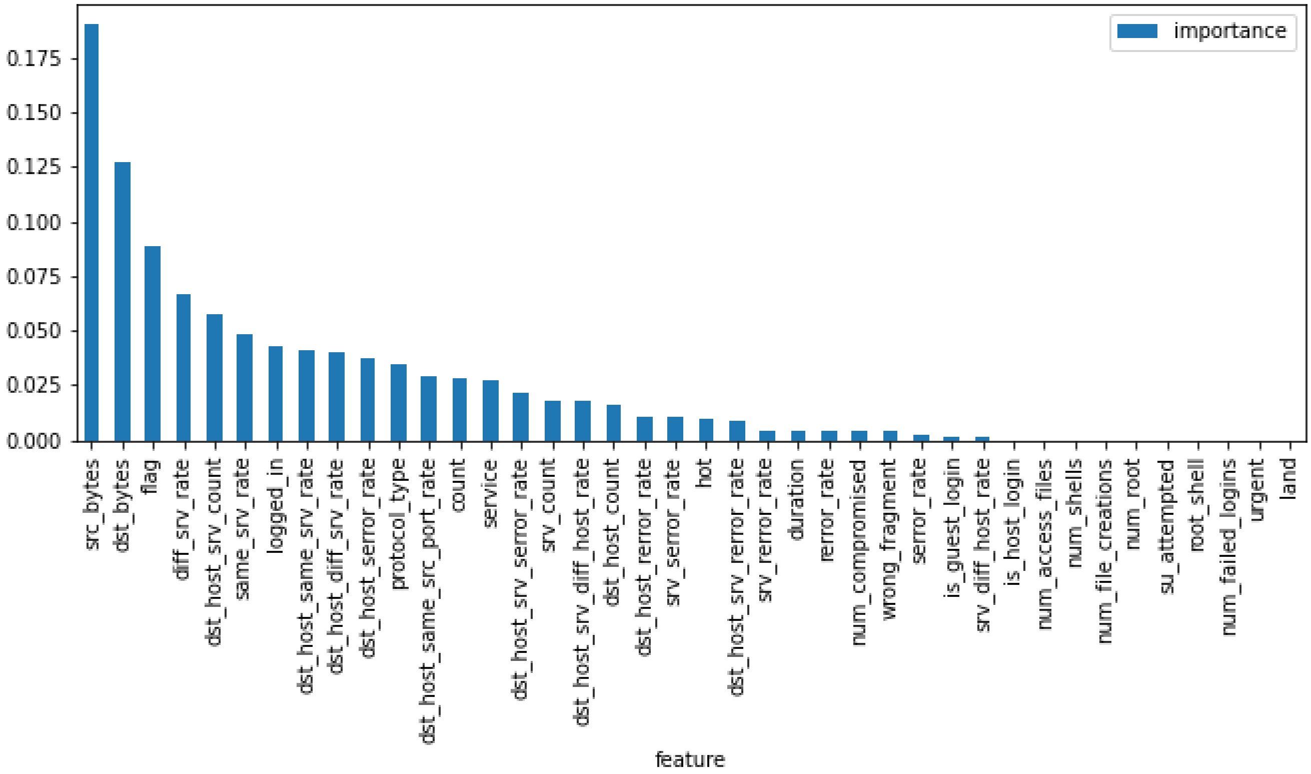
```
(22544, 40)
```

```
#Feature Selection
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier();

# fit random forest classifier on the training set
rfc.fit(train_x, train_y);

# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':train_x.columns,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index('feature')

# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();
```



```
#Recursive feature elimination
from sklearn.feature_selection import RFE
import itertools

rfc = RandomForestClassifier()

# create the RFE model and select 15 attributes
rfe = RFE(rfc, n_features_to_select=15)
rfe = rfe.fit(train_x, train_y)

# summarize the selection of the attributes
feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), train_x.columns)]
```

```
selected_features = [v for i, v in feature_map if i==True]

selected_features
```

```
['src_bytes',
 'dst_bytes',
 'logged_in',
 'count',
 'srv_count',
 'same_srv_rate',
 'diff_srv_rate',
 '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',
 'protocol_type',
 'service',
 'flag']
```

```
a = [i[0] for i in feature_map]

train_x = train_x.iloc[:,a]
test_df = test_df.iloc[:,a]
```

```
#Dataset Partition
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test = train_test_split(train_x,train_y,train_size=0.70, random_stat

#Fitting Models
# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu', input_di

# Adding the second hidden layer
classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
```

```
classifier.fit(X_train, Y_train, batch_size = 10, epochs = 100)
```

```
Epoch 1/100
1764/1764 [==============================] - 2s 946us/step - loss: 0.2384 - accuracy
Epoch 2/100
1764/1764 [==============================] - 2s 938us/step - loss: 0.1615 - accuracy
Epoch 3/100
1764/1764 [==============================] - 2s 955us/step - loss: 0.1397 - accuracy
Epoch 4/100
1764/1764 [==============================] - 2s 939us/step - loss: 0.1256 - accuracy
Epoch 5/100
1764/1764 [==============================] - 2s 953us/step - loss: 0.1159 - accuracy
Epoch 6/100
1764/1764 [==============================] - 2s 946us/step - loss: 0.1072 - accuracy
Epoch 7/100
1764/1764 [==============================] - 2s 951us/step - loss: 0.0997 - accuracy
Epoch 8/100
1764/1764 [==============================] - 2s 947us/step - loss: 0.0956 - accuracy
Epoch 9/100
1764/1764 [==============================] - 2s 959us/step - loss: 0.0912 - accuracy
Epoch 10/100
1764/1764 [==============================] - 2s 954us/step - loss: 0.0872 - accuracy
Epoch 11/100
1764/1764 [==============================] - 2s 957us/step - loss: 0.0858 - accuracy
Epoch 12/100
1764/1764 [==============================] - 2s 942us/step - loss: 0.0835 - accuracy
Epoch 13/100
1764/1764 [==============================] - 2s 949us/step - loss: 0.0826 - accuracy
Epoch 14/100
1764/1764 [==============================] - 2s 963us/step - loss: 0.0824 - accuracy
Epoch 15/100
1764/1764 [==============================] - 2s 962us/step - loss: 0.0811 - accuracy
Epoch 16/100
1764/1764 [==============================] - 2s 937us/step - loss: 0.0805 - accuracy
Epoch 17/100
1764/1764 [==============================] - 2s 955us/step - loss: 0.0786 - accuracy
Epoch 18/100
1764/1764 [==============================] - 2s 963us/step - loss: 0.0784 - accuracy
Epoch 19/100
1764/1764 [==============================] - 2s 949us/step - loss: 0.0765 - accuracy
Epoch 20/100
1764/1764 [==============================] - 2s 964us/step - loss: 0.0750 - accuracy
Epoch 21/100
1764/1764 [==============================] - 2s 958us/step - loss: 0.0757 - accuracy
Epoch 22/100
1764/1764 [==============================] - 2s 952us/step - loss: 0.0754 - accuracy
Epoch 23/100
1764/1764 [==============================] - 2s 960us/step - loss: 0.0753 - accuracy
Epoch 24/100
1764/1764 [==============================] - 2s 985us/step - loss: 0.0735 - accuracy
Epoch 25/100
1764/1764 [==============================] - 2s 981us/step - loss: 0.0726 - accuracy
Epoch 26/100
1764/1764 [==============================] - 2s 953us/step - loss: 0.0718 - accuracy
Epoch 27/100
1764/1764 [==============================] - 2s 979us/step - loss: 0.0714 - accuracy
Epoch 28/100
```

```
1764/1764 [==============================] - 2s 954us/step - loss: 0.0716 - accuracy
Epoch 29/100
1764/1764 [==============================] - 2s 950us/step - loss: 0.0698 - accuracy
```

```
yhat_train = (classifier.predict(X_train) > 0.5)
yhat_test = (classifier.predict(X_test) > 0.5)


# PREDICTING FOR TEST DATA
pred_ann = classifier.predict(test_df)


#Evaluate Models
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from keras.wrappers.scikit_learn import KerasClassifier

def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu', inpu
    classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy
    return classifier

classifier = KerasClassifier(build_fn = build_classifier, batch_size = 10, epochs = 100)

scores = cross_val_score(estimator = classifier,X = X_train, y = Y_train, cv = 10, n_jobs = 1
```

```
1588/1588 [==============================] - 2s 1ms/step - loss: 0.1084 - accuracy:
Epoch 14/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.1065 - accuracy:
Epoch 15/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.1042 - accuracy:
Epoch 16/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.1036 - accuracy:
Epoch 17/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.1012 - accuracy:
Epoch 18/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0998 - accuracy:
Epoch 19/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0989 - accuracy:
Epoch 20/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0977 - accuracy:
Epoch 21/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0965 - accuracy:
Epoch 22/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0945 - accuracy:
Epoch 23/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0940 - accuracy:
Epoch 24/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0930 - accuracy:
Epoch 25/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0923 - accuracy:
```

```
Epoch 26/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0904 - accuracy:
Epoch 27/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0893 - accuracy:
Epoch 28/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0885 - accuracy:
Epoch 29/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0877 - accuracy:
Epoch 30/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0867 - accuracy:
Epoch 31/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0848 - accuracy:
Epoch 32/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0848 - accuracy:
Epoch 33/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0840 - accuracy:
Epoch 34/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0832 - accuracy:
Epoch 35/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0835 - accuracy:
Epoch 36/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0821 - accuracy:
Epoch 37/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0819 - accuracy:
Epoch 38/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0817 - accuracy:

Epoch 39/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0808 - accuracy:
Epoch 40/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0805 - accuracy:
Epoch 41/100
1588/1588 [==============================] - 2s 1ms/step - loss: 0.0793 - accuracy:
Epoch 42/100
```

```
accuracy = metrics.accuracy_score(Y_train, yhat_train)
confusion_matrix = metrics.confusion_matrix(Y_train, yhat_train)
classification = metrics.classification_report(Y_train, yhat_train)
print()
print('============================== ANN Model Evaluation ==============================' )
print()
print ("Cross Validation Mean Score:" "\n", scores.mean())
print()
print ("Model Accuracy:" "\n", accuracy)
print()
print("Confusion matrix:" "\n", confusion_matrix)
print()
print("Classification report:" "\n", classification)
print()
```

```
============================== ANN Model Evaluation ==============================

Cross Validation Mean Score:
 0.9796981692314148
```

```
Model Accuracy:
 0.9859929681297493

Confusion matrix:
 [[8170   75]
 [ 172 9217]]

Classification report:
               precision    recall  f1-score   support

           0       0.98      0.99      0.99      8245
           1       0.99      0.98      0.99      9389

    accuracy                           0.99     17634
   macro avg       0.99      0.99      0.99     17634
weighted avg       0.99      0.99      0.99     17634
```

```
#Validate Models
accuracy = metrics.accuracy_score(Y_test, yhat_test)
confusion_matrix = metrics.confusion_matrix(Y_test, yhat_test)
classification = metrics.classification_report(Y_test, yhat_test)
print()
print('============================== ANN Model Test Results ==============================')
print()
print ("Model Accuracy:" "\n", accuracy)
print()
print("Confusion matrix:" "\n", confusion_matrix)
print()
print("Classification report:" "\n", classification)
print()
```

```
============================== ANN Model Test Results ==============================

Model Accuracy:
 0.9861074358295846

Confusion matrix:
 [[3465   33]
 [  72 3988]]

Classification report:
               precision    recall  f1-score   support

           0       0.98      0.99      0.99      3498
           1       0.99      0.98      0.99      4060

    accuracy                           0.99      7558
   macro avg       0.99      0.99      0.99      7558
weighted avg       0.99      0.99      0.99      7558
```