Configuration Manual

This document gives detailed guidance on utilizing various tools and software to successfully complete projects. It includes clear, step-by-step instructions for software installation and outlines a systematic approach to ensure project success. Additionally, it provides comprehensive hardware specifications necessary for running machine learning algorithms effectively, all thoroughly described within the study.

1. Hardware Specifications

Hardware Component	**Requirements**
Processor (CPU)	MacBook Pro
Memory (RAM)	16GB
Network Connection	Stable internet connection
Operating System	MacOS Sonoma 14.1.1
System type	512 SSD

2. Software Specifications

Software Component	**Requirements**
Python	Python 3.10.12
Seaborn	This is used for data visualization
NumPy	Numerical computing library
pandas	Data manipulation library
matplotlib.pyplot	This library is used for creating static, animated, and interactive visualizations in Python
plotly	It is a high-level interface which is used for creating interactive plots
plotly.graph_objects	It is a low-level interface used to construct Plotly plots.
plotly.figure_factory	used for creating complex Plotly figures.
sklearn	A machine learning library used for various tasks such as classification, regression, clustering etc
imblearn.over_sampling	This library is used for dealing with imbalanced datasets, particularly for oversampling techniques
tensorflow.keras.models.Sequential	This is a class to create a linear stack of layers in TensorFlow's high-level neural network API, Keras
An operating system	Compatible with the required software
tensorflow.keras.models.Model	a class that enables the construction of an independent model with predetermined inputs and results.

Software Component	Requirements
tensorflow.keras.optimizers	For the purpose of training deep learning models, Keras provides optimization techniques.
sklearn.preprocessing	used for preprocessing of data before applying machine learning algorithms.
sklearn.linear_model	uses the linear model of logistic regression for binary categorization
sklearn.tree	uses decision tree classifiers to do regression and classification
sklearn.metrics	Features for assessing how well machine learning models perform
tensorflow.keras.layers	Keras-provided layers for neural network creation

3. Python Packages and Imports

Data Manipulation and Machine Learning Packages:

- **numpy and pandas:** used to handle and manipulate data.
- **sklearn:** For making predictions using the loaded model, Scikit-learn is a machine learning library utilized.
- **Tensorflow:** Machine learning models are developed, trained, and implemented using TensorFlow in a variety of fields, including recommendation systems, computer vision, and natural language processing.

2. Custom Modules:

- **Smote(Synthetic Minority Over-sampling Technique):** In machine learning, this technique is used to rectify class imbalance in datasets. In order to balance the class distribution and enhance the performance of classification algorithms, it creates synthetic samples of the minority class. This works particularly well in scenarios where one class is noticeably underrepresented in comparison to other classes.
- **Autoencoder**

It is used for unsupervised learning problems, autoencoders are flexible neural network topologies. Their functions encompass dimensionality reduction, which involves condensing high-dimensional data into a lower-dimensional representation, feature

learning, which involves identifying the key aspects of the input data, and data denoising, which entails training on noisy data and reconstructing clean versions. Since autoencoders can precisely reconstruct normal occurrences and identify departures from predefined patterns, they can also be used for anomaly identification. Variations such as Variational Autoencoders (VAEs) further facilitate generative modeling by producing new data samples that closely resemble the distribution

3. Other General Imports:

A variety of Python modules and functions for general code functionality, including data structure handling and string manipulation.

4. Dataset Overview and Data Loading

The UNSW-NB15 dataset, comprising real and synthetic network traffic, contains nine attack types captured using Tcpdump, totaling 2,540,044 records. Features are extracted using Argus and Bro-IDS, generating 49 features described in UNSW-NB15_features.csv. Training and testing sets consist of 175,341 and 82,332 records, respectively.

dataset link:- https://www.kaggle.com/datasets/mrwellsdavid/unsw-nb15

Data Loading

The provided code snippet demonstrates data loading using the Pandas library. It displays the first 5 rows of the loaded data using the **head** method. This operation allows for quick exploration and understanding of the dataset's structure and content.

```
#data loading
dataframe = pd.read_csv('/content/drive/MyDrive/network_intrusion_unsw/Data/UNSW_NB15_training-set.csv')
dataframe.head()
```

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd
0	1	0.000011	udp	-	INT	2	0	496	0	90909.0902	...	1	2	0	0	0
1	2	0.000008	udp	-	INT	2	0	1762	0	125000.0003	...	1	2	0	0	0
2	3	0.000005	udp	-	INT	2	0	1068	0	200000.0051	...	1	3	0	0	0
3	4	0.000006	udp	-	INT	2	0	900	0	166666.6608	...	1	3	0	0	0
4	5	0.000010	udp	-	INT	2	0	2126	0	100000.0025	...	1	3	0	0	0

Figure 1 Output of the dataset loading

ct_src_ltm	ct_srv_dst	is_sm_ips_ports	attack_cat	label
1	2	0	Normal	0
1	2	0	Normal	0
1	3	0	Normal	0
2	3	0	Normal	0
2	3	0	Normal	0

Figure 2 output of dataset loading

5. Data Cleaning and Preprocessing

To prepare a dataset for analysis and modeling, data cleaning and preprocessing are necessary tasks. The task involves handling missing values, eliminating duplicates, transforming and encoding data, employing label encoding to convert categorical variables into numerical format, normalising numerical features using Minmax Scaler, binarizing labels, and separating the dataset for evaluation. Data quality and compatibility with machine learning and deep learning algorithms are ensured by the specific tasks that depend on the dataset's nature and objectives. This entails,.

```
dataframe = dataframe.drop(['service','id'], axis='columns')  #
#removing 'analysis' class from target column
dataframe = dataframe[dataframe['attack_cat'] != 'Analysis']
dataframe.shape
dataframe.describe()
```

The provided code snippet removes the columns 'service' and 'id' from the DataFrame. Pandas drop () function is used to delete certain rows or columns from a DataFrame. In this case, axis='columns' indicates that we want to drop columns, whereas ['service', 'id'] gives the column names to be discarded. The DataFrame is then filtered using the values in the 'attack_cat' column. It specifically removes records where the value in the 'attack_cat' column is 'Analysis'. As a result, records containing 'Analysis' in the 'attack_cat' column are excluded from the DataFrame. The **dataframe.shape** is frequently used to rapidly determine the size of a DataFrame following actions such as filtering or eliminating columns. The **dataframe.describe()** computes summary statistics for the DataFrame. The describe() function computes and delivers several statistics for the DataFrame's numeric columns, including count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum. These statistics provide a rapid summary of the DataFrame's numeric data distribution and primary patterns.

	dur	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	...	ct_src_dport_ltm
count	81655.000000	81655.000000	81655.000000	8.165500e+04	8.165500e+04	8.165500e+04	81655.000000	81655.000000	8.165500e+04	8.165500e+04	...	81655.000000
mean	1.012780	18.795395	17.685335	8.057878e+03	1.334264e+04	8.167819e+04	180.498537	96.327561	6.395813e+07	6.357653e+05	...	4.939906
std	4.725343	134.461959	116.041821	1.723509e+05	1.520934e+05	1.483181e+05	101.684286	116.777718	1.801212e+08	2.402211e+06	...	8.420424
min	0.000000	1.000000	0.000000	2.400000e+01	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	...	1.000000
25%	0.000008	2.000000	0.000000	1.140000e+02	0.000000e+00	2.849614e+01	62.000000	0.000000	1.109169e+04	0.000000e+00	...	1.000000
50%	0.016014	6.000000	2.000000	5.360000e+02	2.160000e+02	2.538071e+03	254.000000	29.000000	5.692008e+05	2.184389e+03	...	1.000000
75%	0.724006	12.000000	10.000000	1.294000e+03	9.940000e+02	1.111111e+05	254.000000	252.000000	6.514286e+07	1.641769e+04	...	4.000000
max	59.999989	10646.000000	11018.000000	1.435577e+07	1.465753e+07	1.000000e+06	255.000000	253.000000	5.268000e+09	2.082111e+07	...	59.000000

Figure 4 Output of the statistical data

ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	label
81655.000000	81655.000000	81655.000000	81655.000000	81655.000000	81655.000000	81655.000000	81655.000000	81655.000000
3.663879	7.481599	0.008352	0.008450	0.130819	6.472292	9.159500	0.011218	0.546874
5.935586	11.455213	0.091545	0.092865	0.641215	8.572818	11.159696	0.105320	0.497801
1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000
1.000000	3.000000	0.000000	0.000000	0.000000	3.000000	5.000000	0.000000	1.000000
3.000000	6.000000	0.000000	0.000000	0.000000	7.000000	11.000000	0.000000	1.000000
38.000000	63.000000	2.000000	2.000000	16.000000	60.000000	62.000000	1.000000	1.000000

Figure 4 Output of the statistical data

```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 81655 entries, 0 to 82331
Data columns (total 43 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   dur                 81655 non-null  float64
 1   proto               81655 non-null  object
 2   state               81655 non-null  object
 3   spkts               81655 non-null  int64
 4   dpkts               81655 non-null  int64
 5   sbytes              81655 non-null  int64
 6   dbytes              81655 non-null  int64
 7   rate                81655 non-null  float64
 8   sttl                81655 non-null  int64
 9   dttl                81655 non-null  int64
 10  sload               81655 non-null  float64
 11  dload               81655 non-null  float64
 12  sloss               81655 non-null  int64
 13  dloss               81655 non-null  int64
 14  sinpkt              81655 non-null  float64
 15  dinpkt              81655 non-null  float64
 16  sjit                81655 non-null  float64
 17  djit                81655 non-null  float64
 18  swin                81655 non-null  int64
 19  stcpb               81655 non-null  int64
 20  dtcpb               81655 non-null  int64
 21  dwin                81655 non-null  int64
 22  tcprtt              81655 non-null  float64
 23  synack              81655 non-null  float64
 24  ackdat              81655 non-null  float64
 25  smean               81655 non-null  int64
 26  dmean               81655 non-null  int64
 27  trans_depth         81655 non-null  int64
 28  response_body_len   81655 non-null  int64
 29  ct_srv_src          81655 non-null  int64
 30  ct_state_ttl        81655 non-null  int64
 31  ct_dst_ltm          81655 non-null  int64
 32  ct_src_dport_ltm    81655 non-null  int64
 33  ct_dst_sport_ltm    81655 non-null  int64
 34  ct_dst_src_ltm      81655 non-null  int64
 35  is_ftp_login        81655 non-null  int64
 36  ct_ftp_cmd          81655 non-null  int64
 37  ct_flw_http_mthd    81655 non-null  int64
 38  ct_src_ltm          81655 non-null  int64
 39  ct_srv_dst          81655 non-null  int64
 40  is_sm_ips_ports     81655 non-null  int64
 41  attack_cat          81655 non-null  object
 42  label               81655 non-null  int64
```

Figure 5 Data Summary

This line outputs a brief summary of the DataFrame, containing information about the index and column dtypes, non-null values, and memory utilisation. The **info()** function in

Pandas is useful for quickly understanding the structure and data types of a DataFrame, as well as checking for missing data.

```
dataframe.isnull().sum()
```

```
dur                  0
proto                0
state                0
spkts                0
dpkts                0
sbytes               0
dbytes               0
rate                 0
sttl                 0
dttl                 0
sload                0
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
djit                 0
swin                 0
stcpb                0
dtcpb                0
dwin                 0
tcprtt               0
synack               0
ackdat               0
smean                0
dmean                0
trans_depth          0
response_body_len    0
ct_srv_src           0
ct_state_ttl         0
ct_dst_ltm           0
ct_src_dport_ltm     0
ct_dst_sport_ltm     0
ct_dst_src_ltm       0
is_ftp_login         0
ct_ftp_cmd           0
ct_flw_http_mthd     0
ct_src_ltm           0
ct_srv_dst           0
is_sm_ips_ports      0
attack_cat           0
label                0
dtype: int64
```

Figure 6 Output showing the amount of null values in each column after data cleaning

This line computes the number of missing values in each column of the DataFrame. The function isnull() returns a DataFrame with the same shape as the original DataFrame. The

sum() function is then applied to this DataFrame, which calculates the sum of True values in each column, essentially calculating the amount of missing entries.

5. Dataset Visualization

```
dataframe.columns
#number of unique value in each columns
for i in dataframe.columns:
print(i,dataframe[i].unique().size)
```

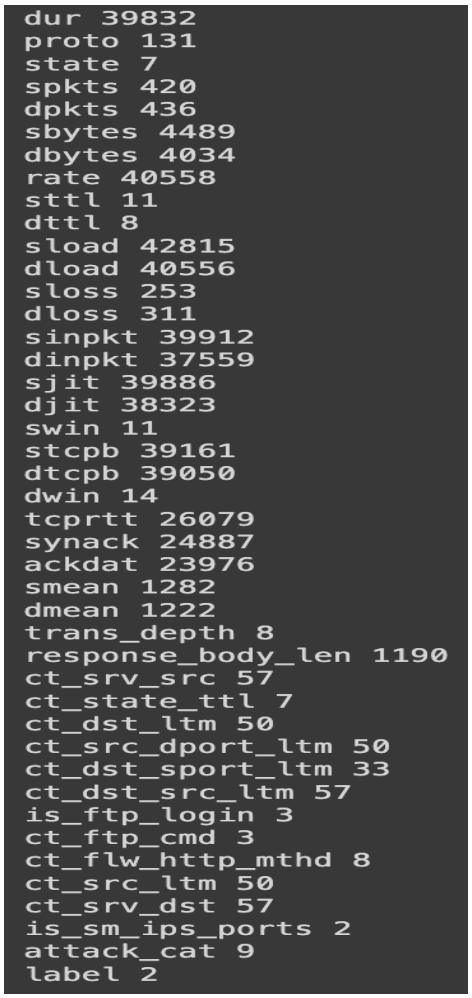

Figure 7 Output showing the unique values with the corresponding columns.

This section of the code snippet retrieves the column names from the DataFrame dataframe. The columns attribute of a Pandas DataFrame returns an Index object containing the column labels. It then, iterates through each column in the DataFrame dataset. It uses the unique() method to calculate the number of unique values in column ‘i’. The unique() method returns an array of unique values for the specified column. The array's size attribute

is then utilised to calculate the number of unique values. Finally, it prints the column name (i) and the number of unique values in that column.

```
for c in integer_names:
pd.to_numeric(dataframe[c])
for c in binary  names:
pd.to_numeric(dataframe[c])
for c in float_names:
pd.to  numeric(dataframe[c])
```

This code snippet creates loops that cycle over the integer, binary, and float feature names, attempting to convert the relevant columns in the dataset DataFrame (dataframe) to numeric data types via pd.to_numeric(). This ensures that numerical operations can be applied to these features if necessary.

```
tempdf = dataframe['attack_cat'].value_counts().reset_index()
fig = px.bar(tempdf, y='count', x='attack cat', color='attack cat',title="Value Count of Attack Category")
fig.show()
```

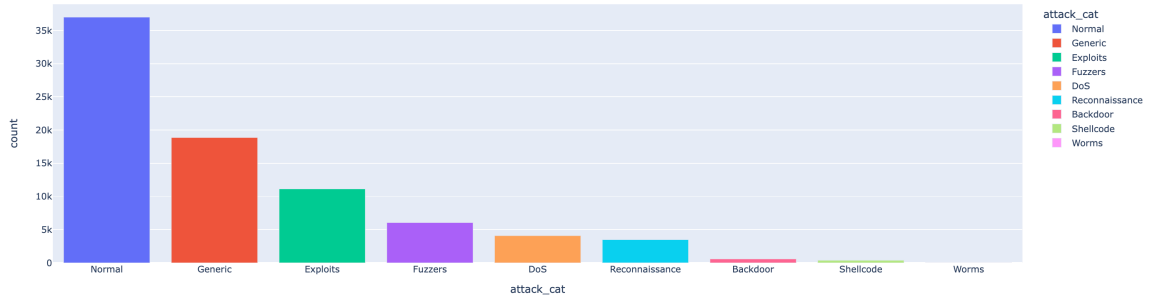

Figure 8 Value Count of Attack Category

To visualise the distribution of attack categories in the dataset, the above provided code generates a bar plot using Plotly Express (px.bar()). The value_counts() method computes the frequency of each attack category, and the DataFrame (tempdf) is used to generate the bar plot.

```
tempdf = dataframe[['state','attack_cat']].groupby(['state','attack_cat']).value_counts()
tempdf = tempdf.reset_index()
tempdf.columns = ['state','attack_cat','count']
fig = px.histogram(tempdf, x="state", y="count",
color='attack  cat', barmode='group', title="Value Count of State by Attack Category")
fig.show()
```

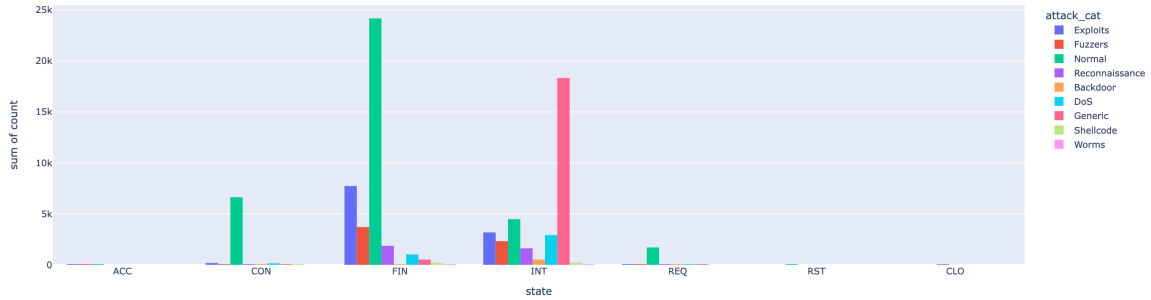

Figure 9 Value Count of State by Attack Category

The above provided code snippet groups the dataframe by the 'state' and 'attack_cat' columns and then counts the number of occurrences of each unique 'state' and 'attack_cat' value pair. The resultant Series contains the counts for each combination of 'state' and 'attack_cat'. It then resets the index of the DataFrame ‘tempdf’, turning the hierarchical index (which includes 'state', 'attack_cat', and the count of occurrences) to regular integer index columns. The code then generates a histogram using Plotly Express (px.histogram()) with the DataFrame ‘tempdf’ as an input. The 'state' column serves as the x-axis, the 'count' column as the y-axis, and the 'attack_cat' column as the colour encoding for grouping. The barmode='group' argument specifies that bars representing various attack categories should be grouped together.

6. Data Pre-processing

```
#splitting data into X and y
X = dataframe.drop(['attack  cat','label'], axis='columns')
y = dataframe['attack_cat']
col = X.select_dtypes(exclude=['float64','int64']).columns.tolist()
col
#label encoding(converting categorical value to numeric)
le = LabelEncoder()
X[col] = X[col].apply(le.fit  transform)
X.head()
```

	dur	proto	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	...	ct_dst_ltm	ct_src_dport_ltm	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd
0	0.000011	117	4	2	0	496	0	90909.0902	254	0	...	1	1	1	2	0	0
1	0.000008	117	4	2	0	1762	0	125000.0003	254	0	...	1	1	1	2	0	0
2	0.000005	117	4	2	0	1068	0	200000.0051	254	0	...	1	1	1	3	0	0
3	0.000006	117	4	2	0	900	0	166666.6608	254	0	...	2	2	1	3	0	0
4	0.000010	117	4	2	0	2126	0	100000.0025	254	0	...	2	2	1	3	0	0

Figure 10 Output of Data Pre-processing

ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_ips_ports
0	1	2	0
0	1	2	0
0	1	3	0
0	2	3	0
0	2	3	0

Figure 11 Output of Data Pre-processing

This section of the code divides the original DataFrame dataframe into two sections. Section X contains the features (independent variables) for model training, which were created by dropping the columns 'attack_cat' and 'label' along the columns axis with drop(). Section Y contains the target variable (dependent variable), which is the 'attack_cat' column. The code encodes labels for the previously identified classified columns. Label encoding translates category variables to numerical representations. It creates a LabelEncoder object ‘le’ and applies it to the category columns of X using the apply() function. The fit_transform() method of LabelEncoder applies the encoder to the data and converts categorical values into numeric labels.

```
#correlation matrix
correlation = X.corr().round(2)
fig = px.imshow(correlation, text_auto=True, aspect="auto")
fig.show()
```

This code snippet generates the correlation matrix for the features in DataFrame X. The corr() method calculates the pairwise correlation of all columns in the DataFrame. The generated correlation matrix is associated with the variable correlation. The round(2) function is used to round correlation data to two decimal places for easier reading. The code then plots the correlation matrix using Plotly Express (px.imshow()). The correlation DataFrame is supplied into the px.imshow() function. The code then creates the correlation matrix plot by using Plotly Express.

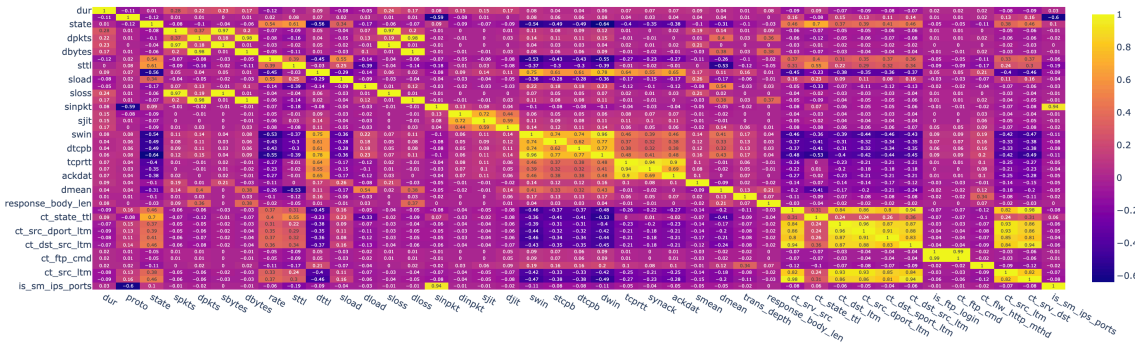

Figure 12 Correlation Matrix

```
#data balancing
orsamp = SMOTE()
X, y = orsamp.fit_resample(X, y)
tempdf = y.value  counts().reset  index()
fig = px.bar(tempdf, y='count', x='attack_cat', color='attack_cat',title="Value Count of Attack Category After Apply Smote Over Sampling")
fig.show()
```

This section of the code loads the Synthetic Minority Oversampling Technique (SMOTE) from the specified library. SMOTE is a prominent strategy for addressing class imbalance by creating synthetic samples for the minority class in order to balance the distribution of classes. The code then performs SMOTE oversampling on the feature matrix X and target vector Y. The fit_resample() method of the SMOTE object ‘orsamp’ applies the SMOTE model to the data before generating synthetic samples to balance the classes. After SMOTE, the balanced feature matrix and target vector are returned to X and Y, respectively. The code computes the count of each class in the target vector Y after SMOTE. The value_counts() method counts the occurrences of each unique value in Y, whereas the reset_index() method resets the index of the resulting series and converts it to a DataFrame. The graph is then generated using Plotly Express (px.bar()). The DataFrame tempdf is utilised as the input data. The 'count' column is designated as the y-axis, the 'attack_cat' column as the x-axis, and the 'attack_cat' column is also used for colour coding to differentiate between different attack categories.

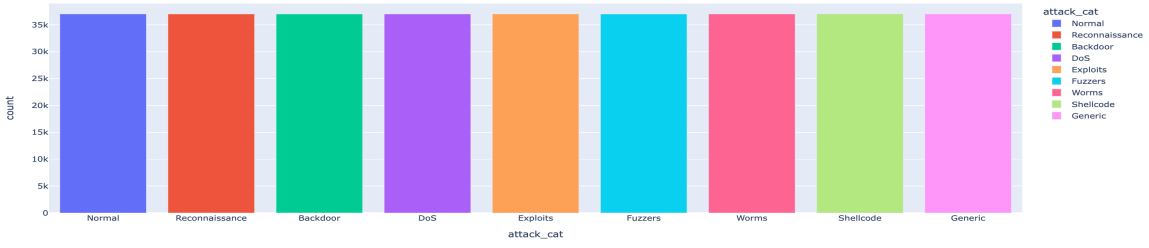

Figure 13 Value Count of Attack Category after applying SMOTE

```
X.shape
#split data in train and test with ratio of 90/10
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1)
X_train.shape
```

This section of the code first outputs the form of the feature matrix X. The code then divides the feature matrix X and target vector Y into training and testing sets. Scikit-learn's train_test_split() function is utilised for this purpose. The test_size=0.1 parameter specifies that 10% of the data will be utilised for testing and the other 90% for training. Output of the form of the training feature matrix X_train is obtained at the end.

7. Machine Learning Algorithms

Logistic regression

```
lrmodel = LogisticRegression()
lrmodel.fit(X_train,y_train)
y_pred = lrmodel.predict(X_test)
```

This section of code generates an instance of scikit-learn's Logistic Regression model. The LogisticRegression() function sets up the logistic regression model with default hyperparameters. The code then uses training data to train the logistic regression model.
The fit() method applies the model to the training data, with X_train representing the feature matrix holding the training samples and Y_train representing the corresponding target vector containing the labels. The learned logistic regression model is then applied to make predictions about the test data. The predict() method predicts the labels for the testing data, X_test being the feature matrix containing the testing samples.
The anticipated labels are stored in the variable Y_pred.

Output Accuracy Score: 0.62

1. **Confusion matrix**

```
#confusion Matrix
matrix=confusion_matrix(y_test, y_pred)
fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
fig.show()
```

The code creates a confusion matrix using the actual labels and predicted labels. The confusion matrix is a table used to assess the performance of a classification model, with each row representing the true class and each column representing the predicted class. The code then generates a plot of the confusion matrix with Plotly Express (px.imshow()). The matrix variable containing the confusion matrix data is provided as input. The text_auto = True argument automatically annotates the plot with values from the confusion matrix. The aspect="auto" argument automatically changes the plot's aspect ratio according to the size of the confusion matrix.

The labels option defines the labels for the plot's x and y axes, which are "Predicted Label" and "Actual Label" accordingly. The confusion matrix plot is then created by using Plotly Express.

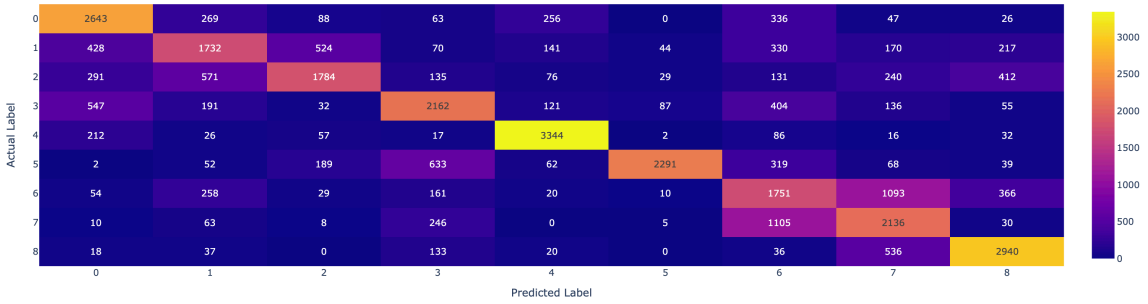

Figure 14 Confusion Matrix for Logistic Regression

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

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

      Backdoor       0.63      0.71      0.67      3728
           DoS       0.54      0.47      0.51      3656
      Exploits       0.66      0.49      0.56      3669
       Fuzzers       0.60      0.58      0.59      3735
       Generic       0.83      0.88      0.85      3792
        Normal       0.93      0.63      0.75      3655
Reconnaissance       0.39      0.47      0.42      3742
     Shellcode       0.48      0.59      0.53      3603
         Worms       0.71      0.79      0.75      3720

      accuracy                           0.62     33300
     macro avg       0.64      0.62      0.63     33300
  weighted avg       0.64      0.62      0.63     33300
```

Figure 15 Classification Report of Logistic Regression

A classification report for the performance of a logistic regression model is generated by the code snippet. The model uses 'sklearn.metrics' to calculate precision, recall, and F1-score for phishing and legitimate URLs, and it also uses 'yellowbrick.classifier' to produce a visual report that helps evaluate the model's ability to distinguish between the two classes in test data.

Similarly the rest of the algorithms were trained and tested and the results were obtained. These results are discussed in the report.

7. Visualization Techniques

This model assessment concentrates on classification of reports and confusion matrices.

uses Seaborn and Matplotlib to generate heatmaps.

True positives, negatives, and error distribution are all thoroughly examined.