Recon-Attack Analysis and Detection

In [2]:
```
import zipfile
import numpy as np
import os
import pandas as pd
```

In [3]:
```
# Path to the ZIP file
zip_file_path = '/home/dev/Desktop/analysis/CICIoT2023.zip'

# Directory where you want to extract the files
extract_dir = 'extracted_files'

# Create the directory if it doesn't exist
os.makedirs(extract_dir, exist_ok=True)

# Open and extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

# List the extracted files
extracted_files = os.listdir(extract_dir)
```

In [4]:
```
#Finding a good size of subset of data
Benign_Reccon_attack_ratio = {}
for file_name in extracted_files:
    file_path = os.path.join(extract_dir, file_name)
    if file_name.endswith('.csv'):
        df = pd.read_csv(file_path)
#         Benign_Reccon_attack_ratio[file_name]=(len(df[df['label'].str.cont
        Benign_Reccon_attack_ratio[file_name]=len(df[df['label'].str.contair
        del df
```

In [5]:
```
#Select the file with max in order to get good subset (as it contain more at
file=max(Benign_Reccon_attack_ratio.items(), key=lambda x: x[1])
file
```

Out[5]:
```
('part-00131-363d1ba3-8ab5-4f96-bc25-4d5862db7cb9-c000.csv', 3543)
```

In [6]:
```
file_path = os.path.join(extract_dir, file[0])
data = pd.read_csv(file_path)
```

In [7]:
```
#checking the retrieved data
data
```

Out[7]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	D
0	0.000000	0.00	1.00	64.00	1.008054	1.008054	
1	0.000000	54.00	6.00	64.00	1.499689	1.499689	
2	0.056092	15942.00	17.00	64.00	5759.748962	5759.748962	
3	0.000000	54.00	6.00	64.00	119.268178	119.268178	
4	0.000000	0.00	1.00	64.00	4.289468	4.289468	
...	...	...	...	...	...	...	
445750	0.249863	25228.50	16.83	63.36	10767.335546	10767.335546	
445751	3.901119	108.00	6.00	64.00	0.512674	0.512674	
445752	0.381013	44776.88	17.00	63.80	5971.800067	5971.800067	
445753	0.000000	54.00	6.00	64.00	125.572325	125.572325	
445754	0.884086	50906.19	16.05	70.00	164.043179	164.043179	

445755 rows × 47 columns

In [8]:
```
data.columns
```

Out[8]:
```
Index(['flow_duration', 'Header_Length', 'Protocol Type', 'Duration', 'Rat
e',
       'Srate', 'Drate', 'fin_flag_number', 'syn_flag_number',
       'rst_flag_number', 'psh_flag_number', 'ack_flag_number',
       'ece_flag_number', 'cwr_flag_number', 'ack_count', 'syn_count',
       'fin_count', 'urg_count', 'rst_count', 'HTTP', 'HTTPS', 'DNS', 'Teln
et',
       'SMTP', 'SSH', 'IRC', 'TCP', 'UDP', 'DHCP', 'ARP', 'ICMP', 'IPv', 'L
LC',
       'Tot sum', 'Min', 'Max', 'AVG', 'Std', 'Tot size', 'IAT', 'Number',
       'Magnitue', 'Radius', 'Covariance', 'Variance', 'Weight', 'label'],
      dtype='object')
```

In [9]:
```
data
```

Out[9]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	D
0	0.000000	0.00	1.00	64.00	1.008054	1.008054	
1	0.000000	54.00	6.00	64.00	1.499689	1.499689	
2	0.056092	15942.00	17.00	64.00	5759.748962	5759.748962	
3	0.000000	54.00	6.00	64.00	119.268178	119.268178	
4	0.000000	0.00	1.00	64.00	4.289468	4.289468	
...	...	...	...	...	...	...	
445750	0.249863	25228.50	16.83	63.36	10767.335546	10767.335546	
445751	3.901119	108.00	6.00	64.00	0.512674	0.512674	
445752	0.381013	44776.88	17.00	63.80	5971.800067	5971.800067	
445753	0.000000	54.00	6.00	64.00	125.572325	125.572325	
445754	0.884086	50906.19	16.05	70.00	164.043179	164.043179	

445755 rows × 47 columns

In [10]:
```
#Getting information of our dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 445755 entries, 0 to 445754
Data columns (total 47 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   flow_duration    445755 non-null  float64
 1   Header_Length    445755 non-null  float64
 2   Protocol Type    445755 non-null  float64
 3   Duration         445755 non-null  float64
 4   Rate             445755 non-null  float64
 5   Srate            445755 non-null  float64
 6   Drate            445755 non-null  float64
 7   fin_flag_number  445755 non-null  float64
 8   syn_flag_number  445755 non-null  float64
 9   rst_flag_number  445755 non-null  float64
 10  psh_flag_number  445755 non-null  float64
 11  ack_flag_number  445755 non-null  float64
 12  ece_flag_number  445755 non-null  float64
 13  cwr_flag_number  445755 non-null  float64
 14  ack_count        445755 non-null  float64
 15  syn_count        445755 non-null  float64
 16  fin_count        445755 non-null  float64
 17  urg_count        445755 non-null  float64
 18  rst_count        445755 non-null  float64
 19  HTTP             445755 non-null  float64
 20  HTTPS            445755 non-null  float64
 21  DNS              445755 non-null  float64
 22  Telnet           445755 non-null  float64
 23  SMTP             445755 non-null  float64
 24  SSH              445755 non-null  float64
 25  IRC              445755 non-null  float64
 26  TCP              445755 non-null  float64
 27  UDP              445755 non-null  float64
 28  DHCP             445755 non-null  float64
 29  ARP              445755 non-null  float64
 30  ICMP             445755 non-null  float64
 31  IPv              445755 non-null  float64
 32  LLC              445755 non-null  float64
 33  Tot sum          445755 non-null  float64
 34  Min              445755 non-null  float64
 35  Max              445755 non-null  float64
 36  AVG              445755 non-null  float64
 37  Std              445755 non-null  float64
 38  Tot size         445755 non-null  float64
 39  IAT              445755 non-null  float64
 40  Number           445755 non-null  float64
 41  Magnitue         445755 non-null  float64
 42  Radius           445755 non-null  float64
 43  Covariance       445755 non-null  float64
 44  Variance         445755 non-null  float64
 45  Weight           445755 non-null  float64
 46  label            445755 non-null  object 
dtypes: float64(46), object(1)
memory usage: 159.8+ MB
```

Data Processing

Cleaning the data

In [11]:
```
# Checking for Null or missing values
data.isnull().sum()
```

Out[11]:
```
flow_duration      0
Header_Length      0
Protocol Type      0
Duration           0
Rate               0
Srate              0
Drate              0
fin_flag_number    0
syn_flag_number    0
rst_flag_number    0
psh_flag_number    0
ack_flag_number    0
ece_flag_number    0
cwr_flag_number    0
ack_count          0
syn_count          0
fin_count          0
urg_count          0
rst_count          0
HTTP               0
HTTPS              0
DNS                0
Telnet             0
SMTP               0
SSH                0
IRC                0
TCP                0
UDP                0
DHCP               0
ARP                0
ICMP               0
IPv                0
LLC                0
Tot sum            0
Min                0
Max                0
AVG                0
Std                0
Tot size           0
IAT                0
Number             0
Magnitue           0
Radius             0
Covariance         0
Variance           0
Weight             0
label              0
dtype: int64
```

No missing values are found.

Checking for duplicates Values

In [12]:
```
#Finding the redundant values
data.duplicated()
```

Out[12]:
```
0         False
1         False
2         False
3         False
4         False
          ...
445750    False
445751    False
445752    False
445753    False
445754    False
Length: 445755, dtype: bool
```

In [13]:
```
data.duplicated().sum()
```

Out[13]:
```
0
```

No duplicate Value found

Pcking Up Bengin and Recon Types of Attacks

In [14]:
```
data = data[data['label'].str.contains('Recon|VulnerabilityScan|BenignTraffi
data
```

Out[14]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	Drate
43	59.958907	4129929.4	6.0	114.9	60.987132	60.987132	0.0
66	20.643120	2113572.2	6.0	181.6	73.599931	73.599931	0.0
107	88.251740	40191.5	7.6	74.9	6.565013	6.565013	0.0
126	62.872302	404714.0	6.0	165.0	73.020604	73.020604	0.0
128	82.240797	80154.8	5.4	52.8	16.384413	16.384413	0.0
...	...	...	...	...	...	...	...
445573	11.422471	399375.0	5.4	191.2	145.342632	145.342632	0.0
445681	40.818306	43853.8	6.0	59.6	12.298885	12.298885	0.0
445692	1.364146	88276.0	6.0	234.0	59.010156	59.010156	0.0
445734	101.062645	486943.2	6.0	74.0	16.920060	16.920060	0.0
445741	141.856169	168198.5	13.7	82.7	17.694076	17.694076	0.0

14050 rows × 47 columns

In [15]:
```
data.describe()
```

Out[15]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	
count	14050.000000	1.405000e+04	14050.000000	14050.000000	14050.000000	14050
mean	103.596066	7.890157e+05	7.513639	110.420140	1857.628296	1857
std	1120.568175	1.233461e+06	2.272351	50.074671	17273.832386	17273
min	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0
25%	7.423453	2.072235e+04	6.000000	69.800000	17.634973	17
50%	26.829248	2.248935e+05	6.500000	96.150000	45.631121	45
75%	60.353202	1.064398e+06	8.200000	137.300000	75.530150	75
max	64023.939954	8.754550e+06	17.000000	255.000000	466041.906529	466041

8 rows × 46 columns

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

Out[16]:
```
(14050, 47)
```

Finding Constant Column

In [17]:
```
constant_columns = data.columns[data.nunique() <= 1]
constant_columns
```

Out[17]:
```
Index(['Drate', 'ece_flag_number', 'cwr_flag_number', 'Telnet', 'SMTP', 'SSH',
       'IRC', 'DHCP', 'ICMP'],
      dtype='object')
```

In the data preprocessing phase, nine columns containing constant values were identified and subsequently removed from the DataFrame, as they did not contribute to the variability in the dataset or influence the analysis outcomes.

In [18]:
```
data = data.drop(columns=constant_columns)
data.shape
```

Out[18]:
```
(14050, 38)
```

In [19]:
```
data.label.unique()
```

Out[19]:
```
array(['BenignTraffic', 'Recon-OSScan', 'Recon-PortScan',
       'VulnerabilityScan', 'Recon-HostDiscovery', 'Recon-PingSweep'],
      dtype=object)
```

In [20]:
```
data.label.value_counts()
```

Out[20]:
```
label
BenignTraffic          10507
Recon-HostDiscovery     1310
Recon-OSScan            1005
Recon-PortScan           813
VulnerabilityScan        387
Recon-PingSweep           28
Name: count, dtype: int64
```

Label column Encoding

In [21]:
```
#write down the mapping for label column in two classification (Benign=0 and
label_mapping = {
    'BenignTraffic': 0,
    'Recon-OSScan': 1,
    'Recon-PortScan': 1,
    'VulnerabilityScan': 1,
    'Recon-HostDiscovery': 1,
    'Recon-PingSweep': 1,
}
new = data
```

In [22]:
```
data.loc[:, 'label'] = data['label'].replace(label_mapping)
data['label']
```

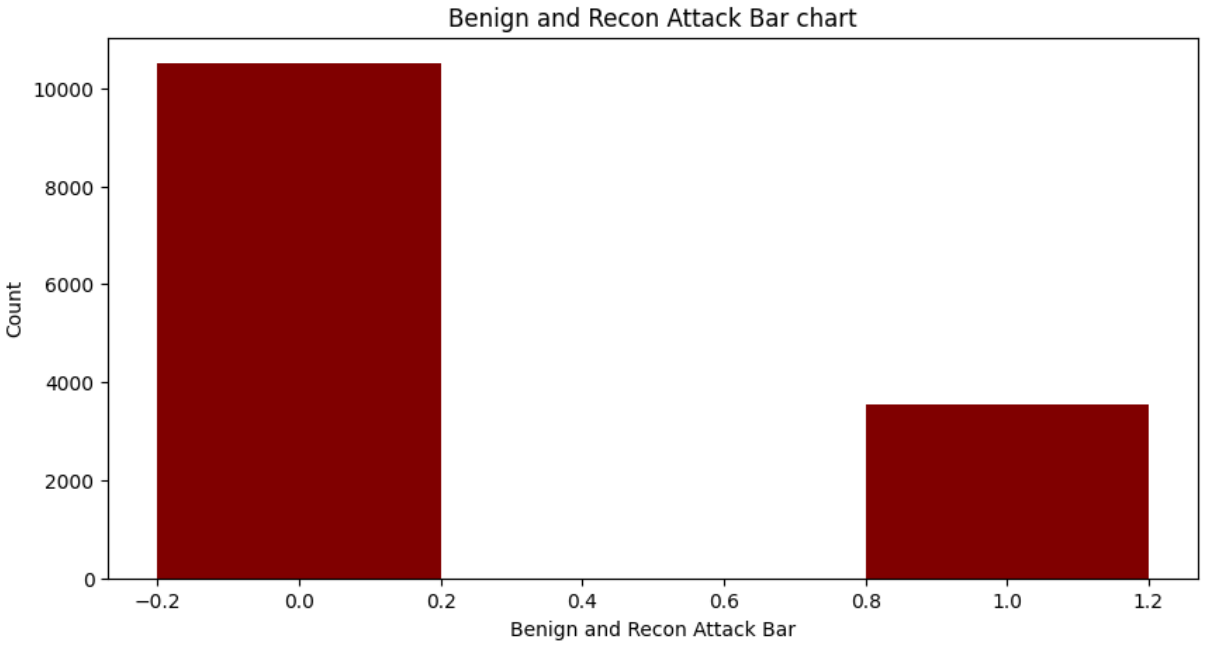
Out[22]:
```
43        0
66        0
107       1
126       0
128       1
         ..
445573    0
445681    0
445692    0
445734    0
445741    0
Name: label, Length: 14050, dtype: object
```

Performing EDA

In [23]:
```
#Importing required libraries
import seaborn as sns
import matplotlib.pyplot as plt
```

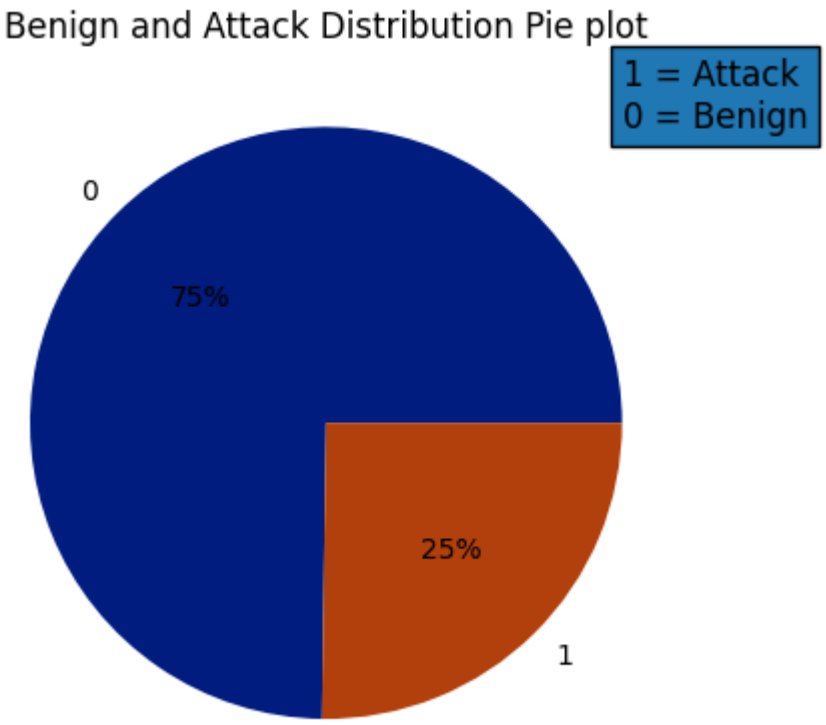
In [48]:
```
pp = data.label.value_counts()

courses = list(pp.keys())
values = list(pp.values)

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
```

```
        width = 0.4)

plt.xlabel("Benign and Recon Attack Bar")
plt.ylabel("Count")
plt.title("Benign and Recon Attack Bar chart")

plt.savefig('Benign_and_Recon_Attack_Bar_chart.png', dpi=300, bbox_inches='t
```



In [143…

```
#plotting pie graph
labelClass =dict(data['label'].sort_values().value_counts())
values = labelClass.values()
keys = labelClass.keys()

# define Seaborn color palette to use
palette_color = sns.color_palette('dark')

plt.title("Benign and Attack Distribution Pie plot")
plt.text(1,1,"1 = Attack\n0 = Benign", fontsize= 12, style='normal', bbox={}

# plotting data on chart
plt.pie(values, labels=keys, colors=palette_color, autopct='%.0f%%')

plt.savefig('label Distribution Pie plot.png', dpi=300, bbox_inches='tight')
```



Description

The pie plot or pie graph shows the distribution of the Benign and Recon-Attacks in the dataset. It reveals that the majority of data are Benign. The dataset contains 75% Benign data and 25% Attacks data.

Bivariate Analysis

Analyzing relationship of varibales with eachother

In [154…

```
#Analyzing relationship of varibales with eachother
correlation = data.corr() #returns the correlation cofficeient against each
correlation
```

Out[154]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Sra
flow_duration	1.000000	-0.029169	0.085628	-0.025707	-0.000404	-0.0004
Header_Length	-0.029169	1.000000	-0.247008	0.236107	-0.038061	-0.0380
Protocol Type	0.085628	-0.247008	1.000000	-0.146833	0.102933	0.1029
Duration	-0.025707	0.236107	-0.146833	1.000000	-0.029824	-0.0298
Rate	-0.000404	-0.038061	0.102933	-0.029824	1.000000	1.0000
Srate	-0.000404	-0.038061	0.102933	-0.029824	1.000000	1.0000
fin_flag_number	-0.000765	-0.007499	-0.005060	-0.009000	-0.001275	-0.0012
syn_flag_number	-0.011377	-0.127665	-0.124890	-0.186053	-0.019860	-0.0198
rst_flag_number	-0.012077	-0.098395	-0.099794	0.012492	-0.015456	-0.0154
psh_flag_number	0.006773	-0.055653	-0.068791	0.044940	0.000365	0.0003
ack_flag_number	-0.064189	0.295756	-0.568206	0.256141	-0.086251	-0.0862
ack_count	0.006234	-0.145861	0.019208	-0.091099	0.003867	0.0038
syn_count	-0.043780	0.065444	-0.341537	0.145234	-0.064851	-0.0648
fin_count	-0.006831	-0.046498	-0.055602	-0.021116	-0.006576	-0.0065
urg_count	0.011075	0.260501	-0.039463	-0.043472	-0.012296	-0.0122
rst_count	-0.024147	0.978102	-0.234494	0.215217	-0.038220	-0.0382
HTTP	-0.015741	0.056870	-0.108097	-0.076395	0.003395	0.0033
HTTPS	-0.060109	0.390631	-0.433829	0.334052	-0.059461	-0.0594
DNS	-0.001807	-0.026635	0.121127	-0.013144	0.042883	0.0428
TCP	-0.072250	0.221151	-0.713249	0.173747	-0.108501	-0.1085
UDP	0.058132	-0.149272	0.715892	-0.126589	0.069050	0.0690
ARP	0.000636	-0.016123	-0.067202	-0.046527	-0.002714	-0.0027
IPv	-0.000732	0.027338	0.075166	0.059096	-0.000807	-0.0008
LLC	-0.000732	0.027338	0.075166	0.059096	-0.000807	-0.0008
Tot sum	-0.033079	0.278534	-0.225754	-0.067284	-0.034622	-0.0346
Min	-0.020633	0.191459	-0.157592	-0.121863	-0.021523	-0.0215
Max	-0.035819	0.245483	-0.200799	-0.024925	-0.034048	-0.0340
AVG	-0.037584	0.308840	-0.254362	-0.116111	-0.037621	-0.0376
Std	-0.034839	0.244101	-0.203762	-0.009966	-0.033897	-0.0338
Tot size	-0.034756	0.308325	-0.260993	-0.245162	-0.040984	-0.0409
IAT	0.003387	0.002328	-0.002092	0.011736	0.004657	0.0046
Number	0.003415	0.002633	-0.001732	0.011831	0.004630	0.0046

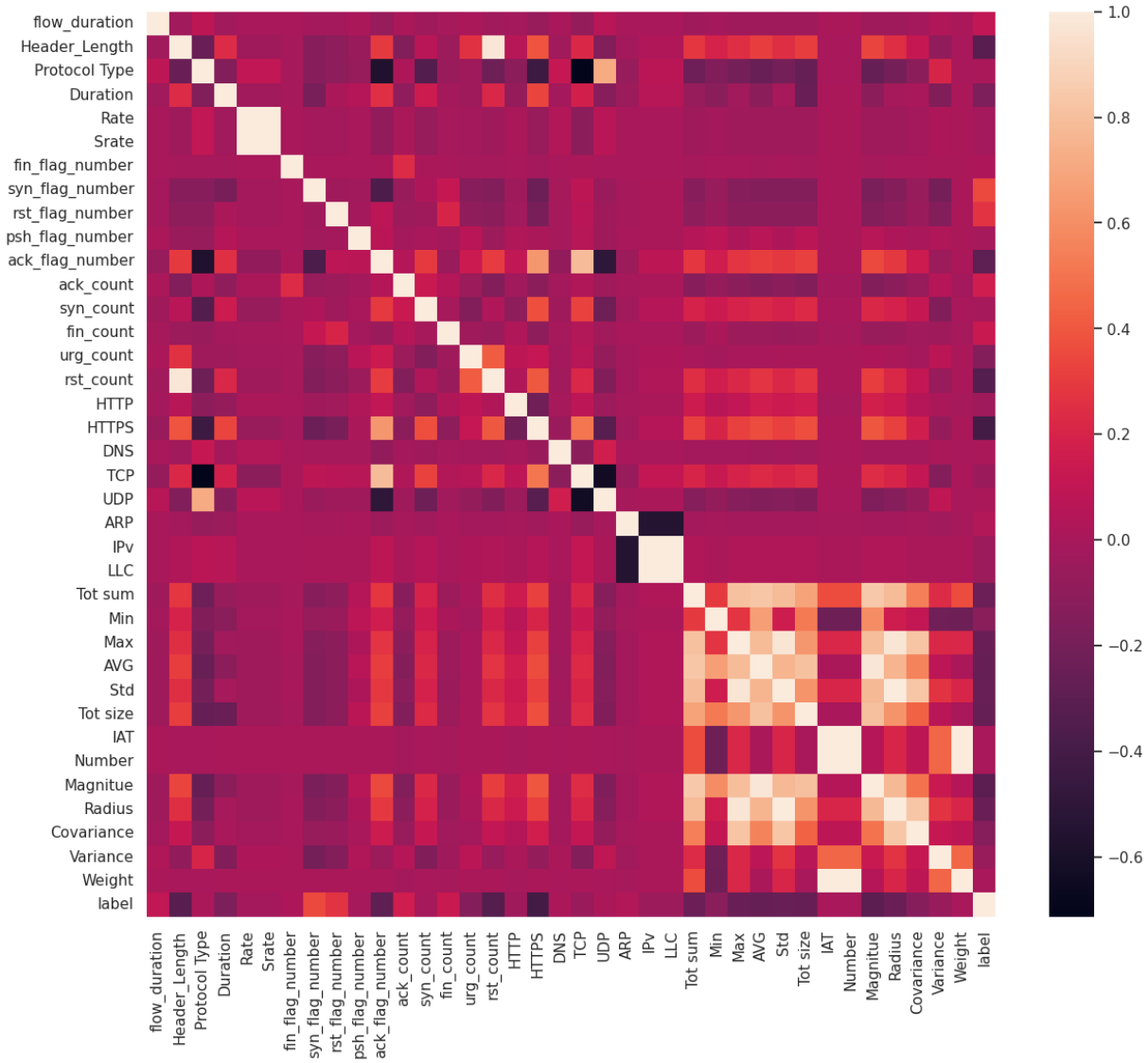
	flow_duration	Header_Length	Protocol Type	Duration	Rate	Sra
Magnitue	-0.039051	0.337073	-0.259900	-0.114449	-0.036218	-0.0362
Radius	-0.034764	0.243867	-0.203637	-0.009460	-0.033798	-0.0337
Covariance	-0.019120	0.122228	-0.109740	-0.000895	-0.020872	-0.0208
Variance	0.027696	-0.088708	0.203099	-0.156158	0.022097	0.0220
Weight	0.003419	0.002622	-0.001792	0.011880	0.004632	0.0046
label	0.098883	-0.321105	0.011164	-0.170334	-0.021240	-0.0212

38 rows × 38 columns

In [155…
```
min(list(correlation.min()))
```

Out[155]: -0.7132486919622638

In [165…
```
#HeatMap
#defining size of the picture

sns.set(rc={'figure.figsize':(14,12)})
sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correl
plt.savefig('heatmap.png', dpi=300)
```



In [166…

```
correlation.sort_values('label')
```

Out[166]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Sra
HTTPS	-0.060109	0.390631	-0.433829	0.334052	-0.059461	-0.0594
rst_count	-0.024147	0.978102	-0.234494	0.215217	-0.038220	-0.0382
Header_Length	-0.029169	1.000000	-0.247008	0.236107	-0.038061	-0.0380
Magnitue	-0.039051	0.337073	-0.259900	-0.114449	-0.036218	-0.0362
ack_flag_number	-0.064189	0.295756	-0.568206	0.256141	-0.086251	-0.0862
AVG	-0.037584	0.308840	-0.254362	-0.116111	-0.037621	-0.0376
Tot size	-0.034756	0.308325	-0.260993	-0.245162	-0.040984	-0.0409
Std	-0.034839	0.244101	-0.203762	-0.009966	-0.033897	-0.0338
Radius	-0.034764	0.243867	-0.203637	-0.009460	-0.033798	-0.0337
Max	-0.035819	0.245483	-0.200799	-0.024925	-0.034048	-0.0340
Tot sum	-0.033079	0.278534	-0.225754	-0.067284	-0.034622	-0.0346
Duration	-0.025707	0.236107	-0.146833	1.000000	-0.029824	-0.0298
urg_count	0.011075	0.260501	-0.039463	-0.043472	-0.012296	-0.0122
Covariance	-0.019120	0.122228	-0.109740	-0.000895	-0.020872	-0.0208
Min	-0.020633	0.191459	-0.157592	-0.121863	-0.021523	-0.0215
Variance	0.027696	-0.088708	0.203099	-0.156158	0.022097	0.0220
TCP	-0.072250	0.221151	-0.713249	0.173747	-0.108501	-0.1085
IPv	-0.000732	0.027338	0.075166	0.059096	-0.000807	-0.0008
LLC	-0.000732	0.027338	0.075166	0.059096	-0.000807	-0.0008
HTTP	-0.015741	0.056870	-0.108097	-0.076395	0.003395	0.0033
Srate	-0.000404	-0.038061	0.102933	-0.029824	1.000000	1.0000
Rate	-0.000404	-0.038061	0.102933	-0.029824	1.000000	1.0000
syn_count	-0.043780	0.065444	-0.341537	0.145234	-0.064851	-0.0648
psh_flag_number	0.006773	-0.055653	-0.068791	0.044940	0.000365	0.0003
Number	0.003415	0.002633	-0.001732	0.011831	0.004630	0.0046
Weight	0.003419	0.002622	-0.001792	0.011880	0.004632	0.0046
IAT	0.003387	0.002328	-0.002092	0.011736	0.004657	0.0046
UDP	0.058132	-0.149272	0.715892	-0.126589	0.069050	0.0690
DNS	-0.001807	-0.026635	0.121127	-0.013144	0.042883	0.0428
Protocol Type	0.085628	-0.247008	1.000000	-0.146833	0.102933	0.1029
fin_flag_number	-0.000765	-0.007499	-0.005060	-0.009000	-0.001275	-0.0012
ARP	0.000636	-0.016123	-0.067202	-0.046527	-0.002714	-0.0027

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Sra
flow_duration	1.000000	-0.029169	0.085628	-0.025707	-0.000404	-0.0004
fin_count	-0.006831	-0.046498	-0.055602	-0.021116	-0.006576	-0.0065
ack_count	0.006234	-0.145861	0.019208	-0.091099	0.003867	0.0038
rst_flag_number	-0.012077	-0.098395	-0.099794	0.012492	-0.015456	-0.0154
syn_flag_number	-0.011377	-0.127665	-0.124890	-0.186053	-0.019860	-0.0198
label	0.098883	-0.321105	0.011164	-0.170334	-0.021240	-0.0212

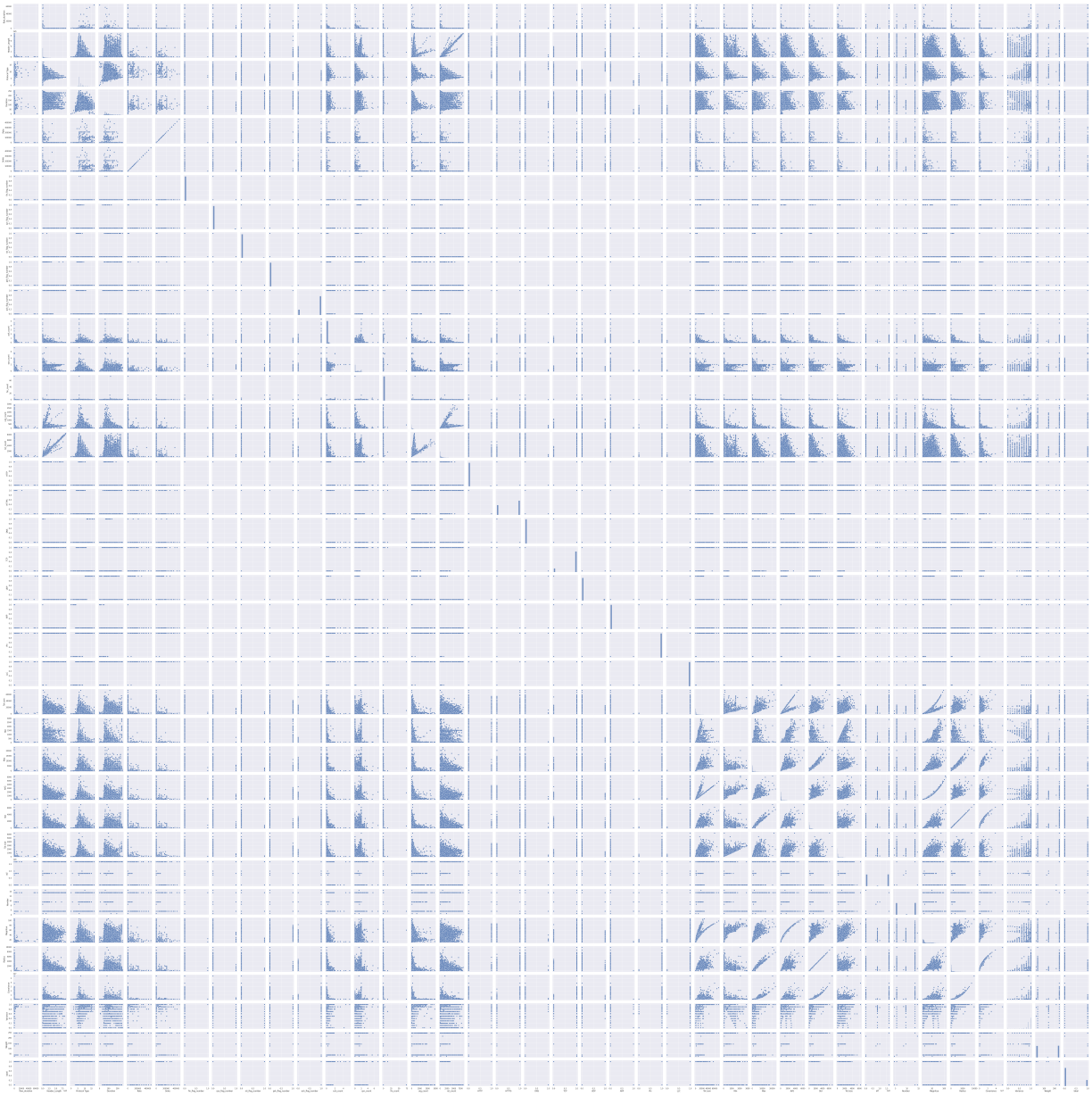
38 rows × 38 columns

Outcomes:

Max Positive correlation: Between label and syn_flag_number, Max Negative Correlation: Between label and HTTPS

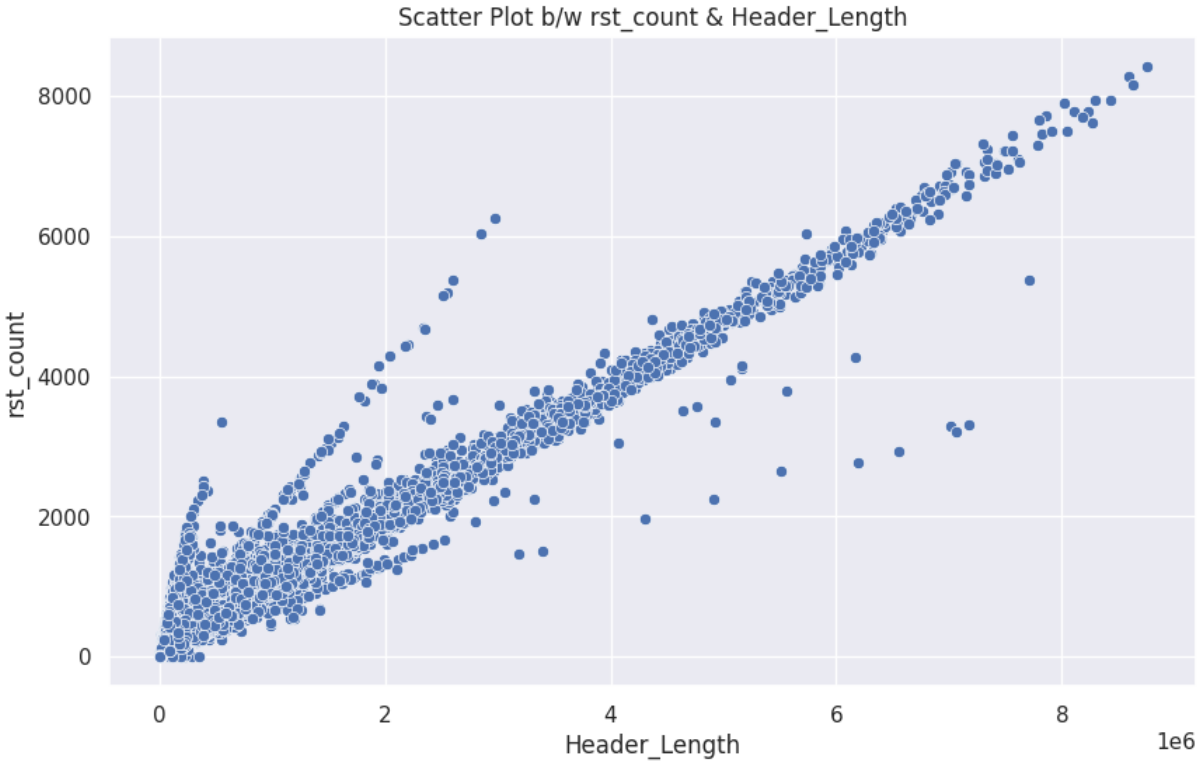
In [168…

```
sns.set(rc={'figure.figsize':(14,12)})
sns.pairplot(data)
plt.savefig('pairplot.png', dpi=300, bbox_inches='tight')
```

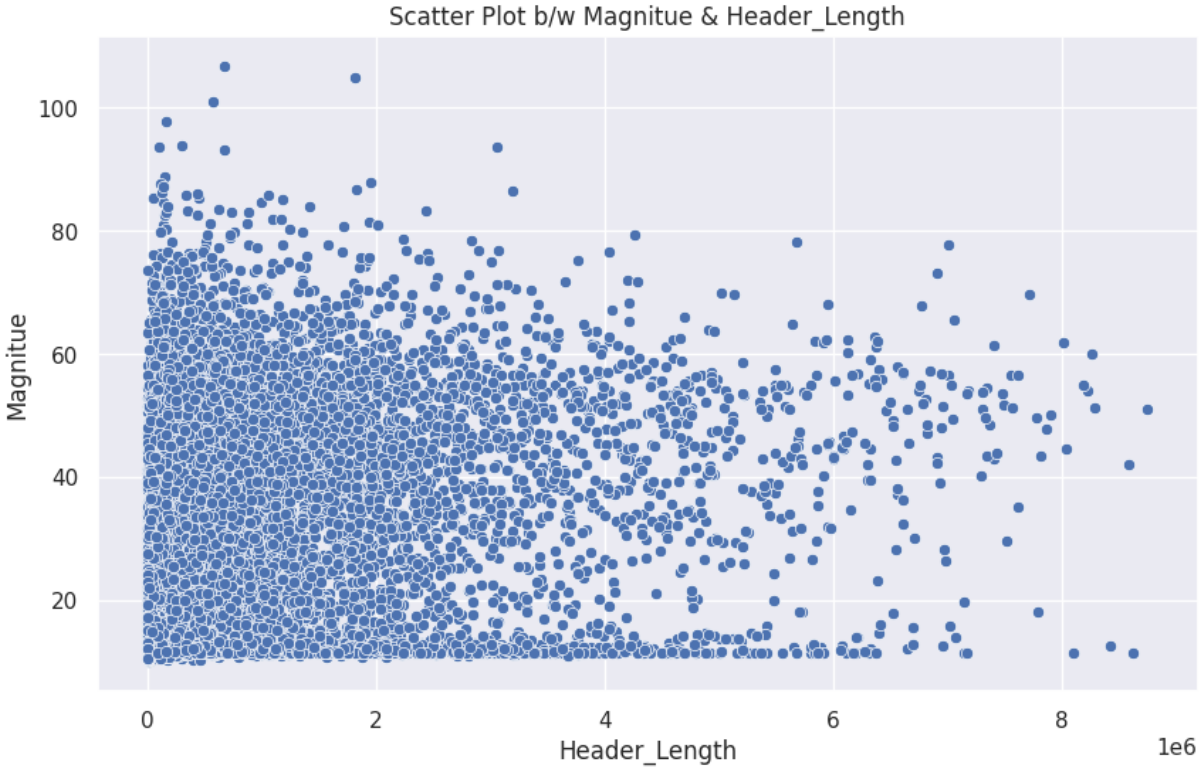


In [175…

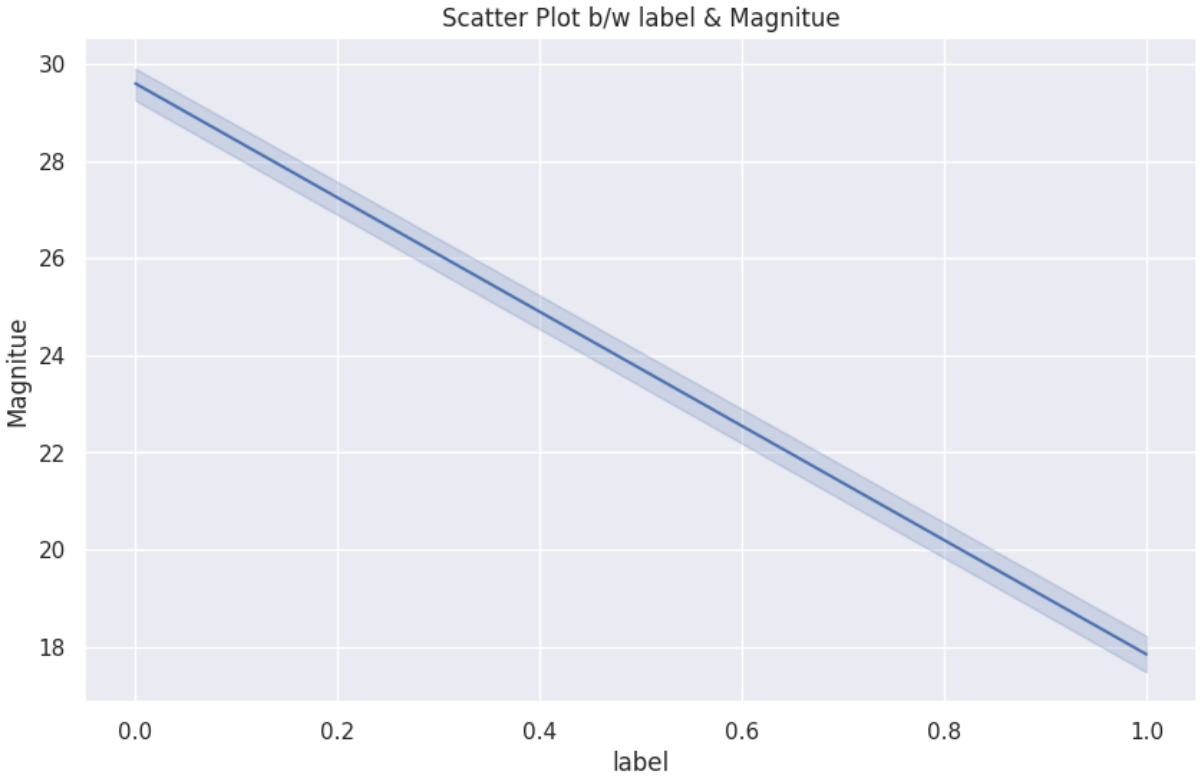
```
sns.scatterplot(x='Header_Length',y='rst_count', data= data).set(title='Scat
sns.set(rc={'figure.figsize':(10,6)})
plt.savefig('Header_Length_and_rst_count_scatter_plot.png', dpi=300)
```



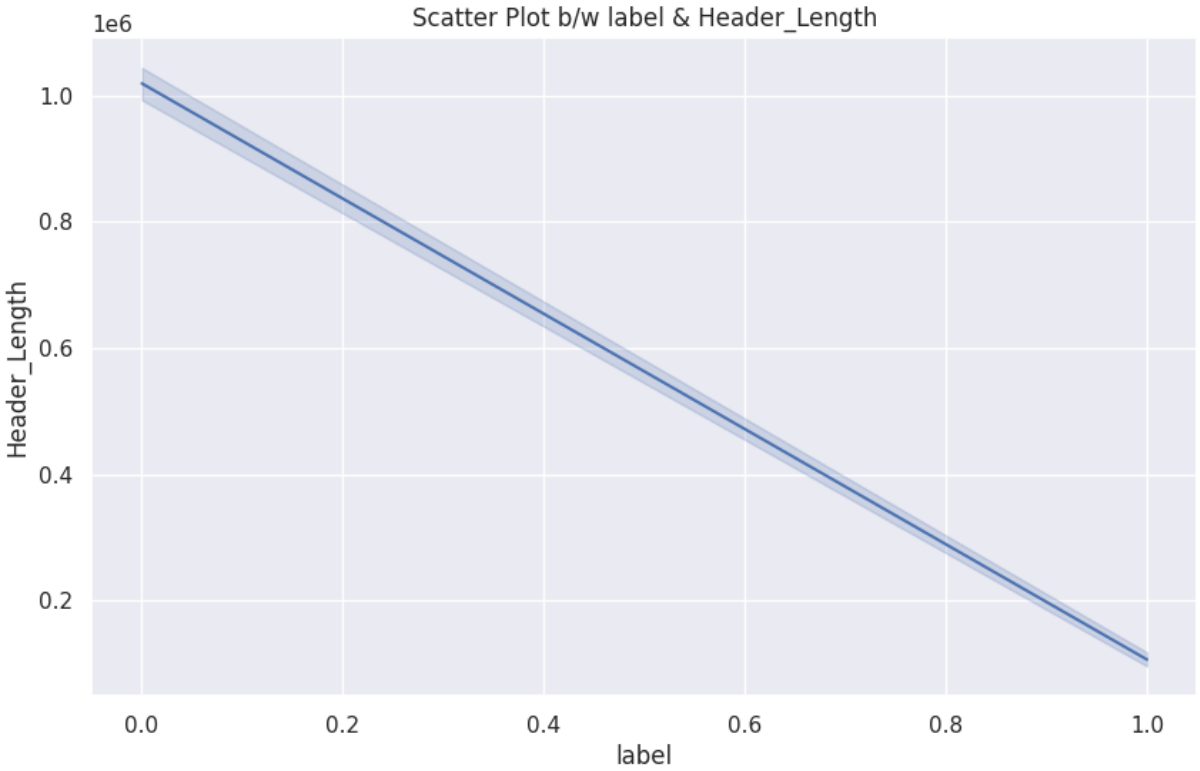
In [176…
```
sns.scatterplot(x='Header_Length',y='Magnitue', data= data).set(title='Scatt
sns.set(rc={'figure.figsize':(10,6)})
plt.savefig('Header_Length_and_Magnitue_scatter_plot.png', dpi=300)
```



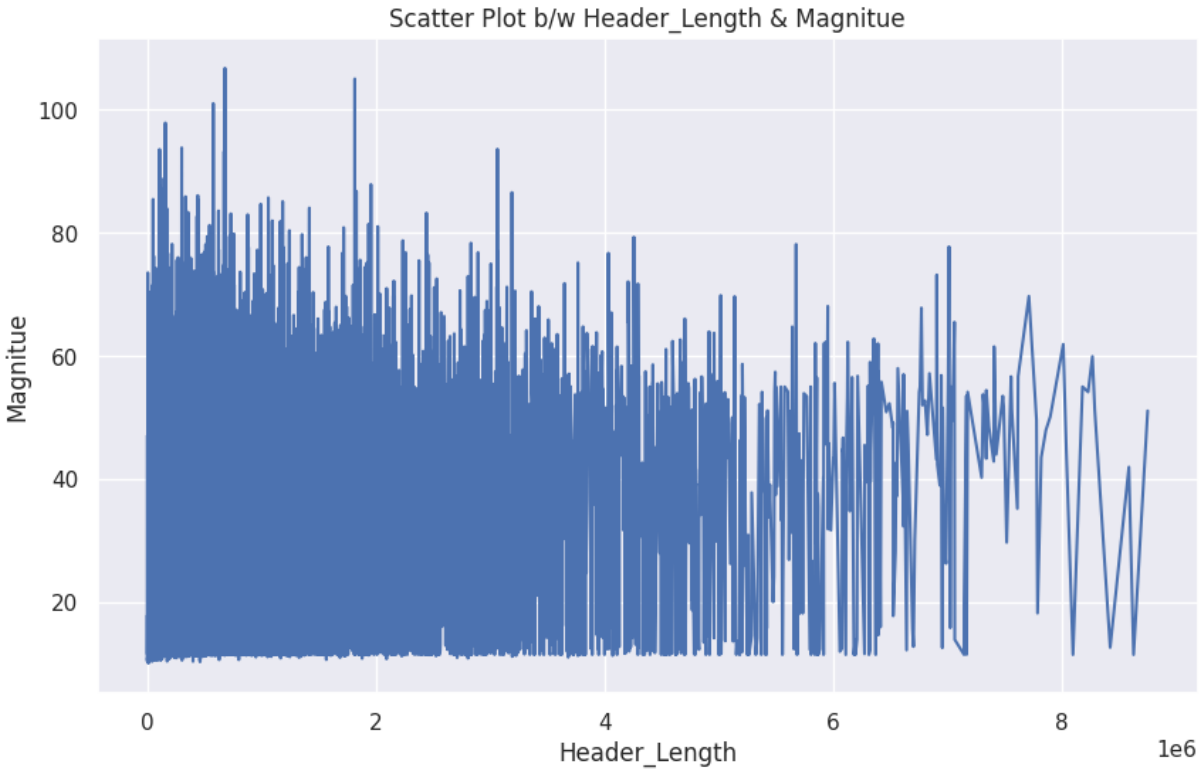
In [181…
```
# draw lineplot
sns.lineplot(x='label',y='Magnitue', data= data).set(title='Scatter Plot b/w
plt.savefig('label_and_Magnitue_Line_plot.png', dpi=300)
```



In [180…
```
# draw lineplot
sns.lineplot(x='label',y='Header_Length', data= data).set(title='Scatter Plo
plt.savefig('label_and_Header_Length_Line_plot.png', dpi=300)
```



In [182…
```
# draw lineplot
sns.lineplot(x='Header_Length',y='Magnitue', data= data).set(title='Scatter
plt.savefig('Header_Length_and_Magnitue_Line_plot.png', dpi=300)
```



Description:

This scatter plot reflects that rst_count and header_length are in a moderate positive relation. As the value of the Header_Length and Magnitude are rises there are more chance that of Attack. There outliers can be seen in the plot.

Scaling

In [49]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()

normalized_data = pd.DataFrame(scaler.fit_transform(data), columns=data.colu
normalized_data
```

Out[49]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	fin_flag_
0	-0.038943	2.708665	-0.666135	0.089467	-0.104013	-0.104013	-(
1	-0.074030	1.073892	-0.666135	1.421525	-0.103283	-0.103283	-(
2	-0.013694	-0.607114	0.038006	-0.709369	-0.107164	-0.107164	-(
3	-0.036343	-0.311575	-0.666135	1.090008	-0.103316	-0.103316	-(
4	-0.019058	-0.574713	-0.930188	-1.150725	-0.106595	-0.106595	-(
...	...	...	...	...	...	...	
14045	-0.082259	-0.315903	-0.930188	1.613245	-0.099130	-0.099130	-(
14046	-0.056025	-0.604144	-0.666135	-1.014923	-0.106832	-0.106832	-(
14047	-0.091235	-0.568129	-0.666135	2.467999	-0.104128	-0.104128	-(
14048	-0.002261	-0.244907	-0.666135	-0.727342	-0.106564	-0.106564	-(
14049	0.034145	-0.503331	2.722546	-0.553596	-0.106519	-0.106519	-(

14050 rows × 38 columns

Model Preparation

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics, svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, ConfusionMatri
```

Separating Features and labels

In [25]:
```
features = data.drop('label', axis=1)
features
```

Out[25]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	fin_fla
43	59.958907	4129929.4	6.0	114.9	60.987132	60.987132	
66	20.643120	2113572.2	6.0	181.6	73.599931	73.599931	
107	88.251740	40191.5	7.6	74.9	6.565013	6.565013	
126	62.872302	404714.0	6.0	165.0	73.020604	73.020604	
128	82.240797	80154.8	5.4	52.8	16.384413	16.384413	
...	...	...	...	...	...	...	
445573	11.422471	399375.0	5.4	191.2	145.342632	145.342632	
445681	40.818306	43853.8	6.0	59.6	12.298885	12.298885	
445692	1.364146	88276.0	6.0	234.0	59.010156	59.010156	
445734	101.062645	486943.2	6.0	74.0	16.920060	16.920060	
445741	141.856169	168198.5	13.7	82.7	17.694076	17.694076	

14050 rows × 37 columns

In [26]:
```
features = features.to_numpy()
features.shape
```

Out[26]:
```
(14050, 37)
```

In [27]:
```
label = np.array(data['label'])
```

Splitting the dataset into Train and Test

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(features, label, test_si
y_train = y_train.astype('int')
y_test = y_test.astype('int')
```

In [29]:
```
Accuracy_dict = {}
```

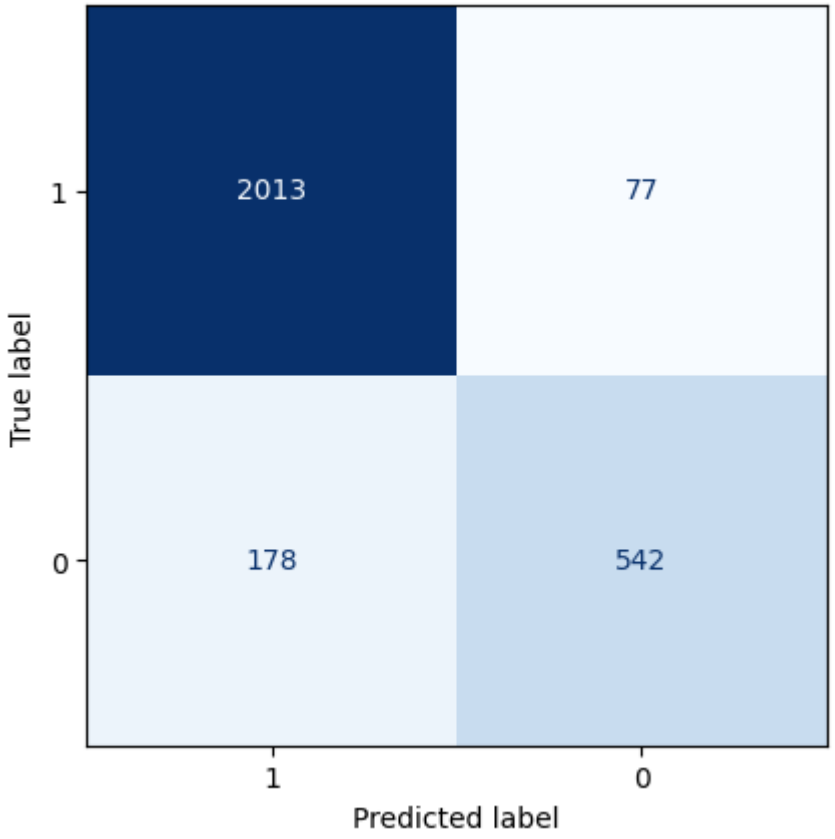
In [30]:
```
def confusionMatrix(y_test, y_pred, model_name):
    cm = confusion_matrix(y_test, y_pred)
    cm_report = metrics.classification_report(y_test,y_pred, zero_division=0
    print(cm_report)
    cmd = ConfusionMatrixDisplay(cm, display_labels=['1','0'])
    cmd.plot(colorbar=False, cmap='Blues')
    plt.savefig("confusion_matrix_with_"+model_name+"_37_variables.png")
```

KNN Model

In [31]:
```
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

```
ac  = accuracy_score(y_test, y_pred)
Accuracy_dict['KNN']=ac
confusionMatrix(y_test, y_pred, "KNN")
```

```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      2090
           1       0.88      0.75      0.81       720

    accuracy                           0.91      2810
   macro avg       0.90      0.86      0.87      2810
weighted avg       0.91      0.91      0.91      2810
```

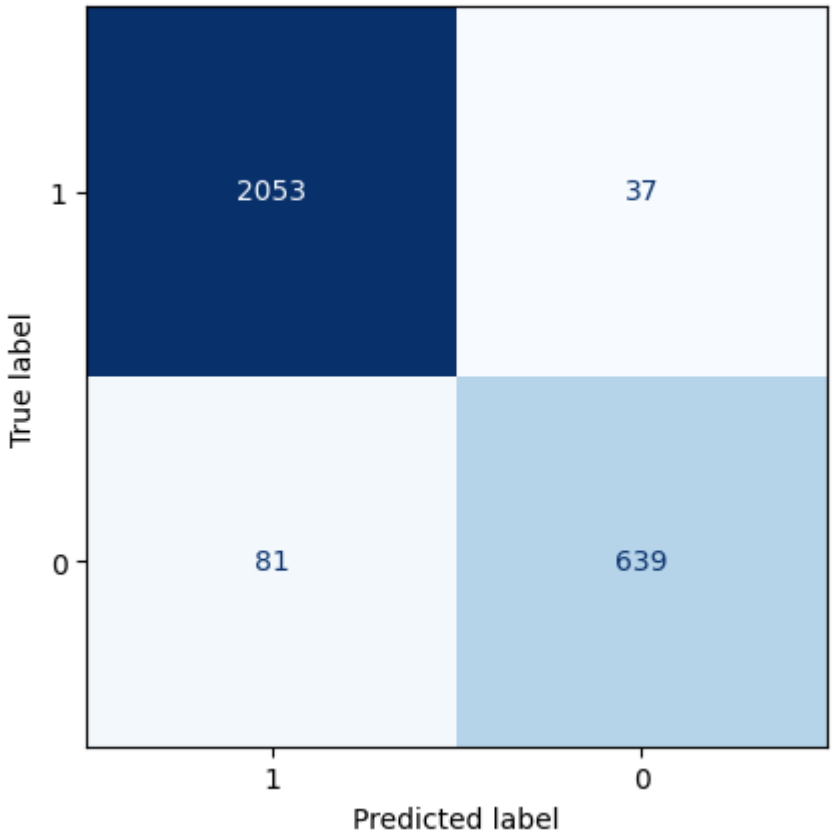


KNN Give us 91% of Accuracy on this data

Random Forest Model

In [32]:
```
model = RandomForestClassifier(n_estimators=500, random_state=1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
ac=accuracy_score(y_test, y_pred)
print("Accuracy is: ",ac*100)
Accuracy_dict['RFM'] = ac
confusionMatrix(y_test, y_pred, "RFM")
```

```
Accuracy is:  95.80071174377224
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      2090
           1       0.95      0.89      0.92       720

    accuracy                           0.96      2810
   macro avg       0.95      0.93      0.94      2810
weighted avg       0.96      0.96      0.96      2810
```

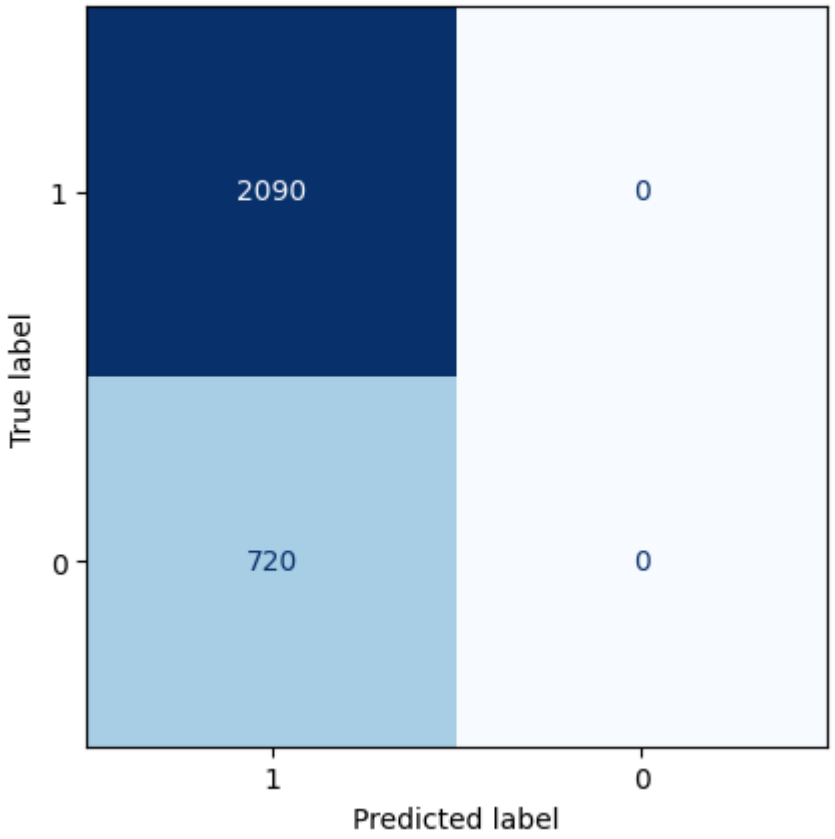


Random Forest Model Give 96% of Accuracy on this data

Logistic Regression

In [33]:
```
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
ac = accuracy_score(y_test, y_pred)
print("Accuracy is: ",round(ac*100,2))
Accuracy_dict['Logistic Regression'] = ac
confusionMatrix(y_test, y_pred, "Logistic_Regression")
```

```
Accuracy is:  74.38
              precision    recall  f1-score   support

           0       0.74      1.00      0.85      2090
           1       0.00      0.00      0.00       720

    accuracy                           0.74      2810
   macro avg       0.37      0.50      0.43      2810
weighted avg       0.55      0.74      0.63      2810
```

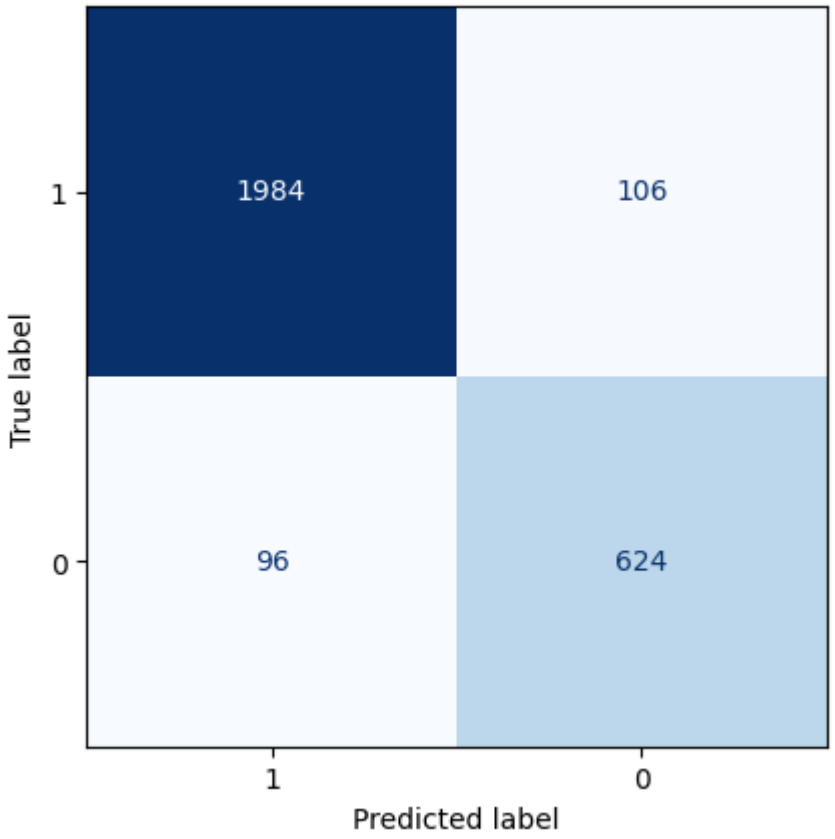


Logistic Regression Model Give 74.38% of Accuracy on this data

Decision Tree

In [34]:
```
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
ac = accuracy_score(y_test, y_pred)
print("Accuracy is: ",round(ac*100,2))
Accuracy_dict['Decision Tree'] = ac
confusionMatrix(y_test, y_pred, "Decision_Tree")
```

```
Accuracy is:  92.81
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      2090
           1       0.85      0.87      0.86       720

    accuracy                           0.93      2810
   macro avg       0.90      0.91      0.91      2810
weighted avg       0.93      0.93      0.93      2810
```

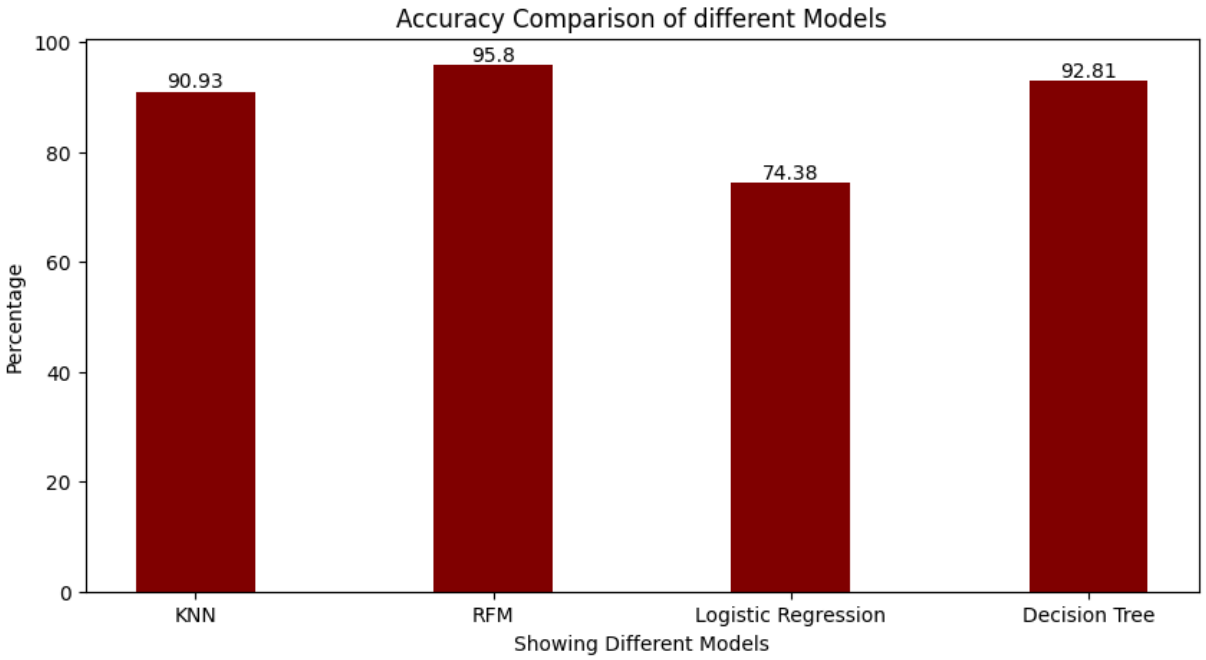


Decision Tree Model Give 93% of Accuracy on this data

Which Model Perform Good

In [42]:
```
modified_data = {key: round(value * 100, 2) for key, value in Accuracy_dict.
courses = list(modified_data.keys())
values = list(modified_data.values())

fig = plt.figure(figsize=(10, 5))

# Creating the bar plot
bars = plt.bar(courses, values, color='maroon', width=0.4)

# Adding the value labels on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='cent
```

```
plt.xlabel("Showing Different Models")
plt.ylabel("Percentage")
plt.title("Accuracy Comparison of different Models")

# Saving the figure
plt.savefig('Benign_and_Recon_Attack_Bar_chart.png', dpi=300, bbox_inches='t

plt.show()
```



Conclusion

Based on the analysis, it's evident that the Random Forest model outperforms the other models in terms of accuracy, achieving an impressive 96%. This suggests that Random Forest is highly effective in capturing the underlying patterns in the data, making it a reliable choice for your classification task.

The KNN and Decision Tree models also perform well, with accuracies of 91% and 93% respectively, indicating that they are reasonable alternatives, though slightly less robust than Random Forest. Logistic Regression, on the other hand, shows a significantly lower accuracy of 74%, suggesting that it might struggle with the complexity or non-linearity in the data.

In summary, the Random Forest model stands out as the best-performing model for this dataset, providing the most accurate predictions.

In [ ]: