FinalProject Last Checkpoint: 28 minutes ago (autosaved) Logout

Menu

Python 3 (ipykernel)

Not Trusted

- File
  - New NotebookDropdown
    - Python 3 (ipykernel)
  - Open...
  -
  - Make a Copy...
  - Save as...
  - Rename...
  - Save and CheckpointCtrl-S
  -
  - Revert to CheckpointDropdown
    - Friday, June 30, 2023 4:53 AM
  -
  - Print Preview
  - Download asDropdown
    - AsciiDoc (.asciidoc)
    - HTML (.html)
    - LaTeX (.tex)
    - Markdown (.md)
    - Notebook (.ipynb)
    - PDF via LaTeX (.pdf)
    - reST (.rst)
    - Python (.py)
    - Reveal.js slides (.slides.html)
    - PDF via HTML (.html)
  - Deploy as
  -
  - Trust Notebook
  -
  - Close and Halt
- Edit
  - Cut CellsX
  - Copy CellsC
  - Paste Cells AboveShift-V
  - Paste Cells BelowV
  - Paste Cells & Replace
  - Delete CellsD,D
  - Undo Delete CellsZ
  -
  - Split CellCtrl-Shift-Minus
  - Merge Cell Above
  - Merge Cell Below
  -
  - Move Cell Up
  - Move Cell Down
  -
  - Edit Notebook Metadata
  -
  - Find and Replace
  -
  - Cut Cell Attachments
  - Copy Cell Attachments
  - Paste Cell Attachments
  -
  - Insert Image
- View
  - Toggle Header
  - Toggle Toolbar
  - Toggle Line NumbersShift-L
  - Cell Toolbar
    - None
    - Edit Metadata
    - Raw Cell Format
    - Slideshow
    - Attachments
    - Tags
- Insert
  - Insert Cell AboveA
  - Insert Cell BelowB
- Cell
  - Run CellsCtrl-Enter
  - Run Cells and Select BelowShift-Enter
  - Run Cells and Insert BelowAlt-Enter
  - Run All
  - Run All Above
  - Run All Below
  -
  - Cell Type
    - CodeY
    - MarkdownM
    - Raw NBConvertR
  -
  - Current Outputs
    - Toggle0
    - Toggle ScrollingShift-O
    - Clear
  - All Output
    - Toggle
    - Toggle Scrolling
    - Clear
- Kernel
  - InterruptI,I
  - Restart0,0
  - Restart & Clear Output
  - Restart & Run All
  - Reconnect
  - Shutdown
  -
  - Change kernel
    - Python 3 (ipykernel)
- Widgets
  - Save Notebook Widget State
  - Clear Notebook Widget State
  - Download Widget State
  - Embed Widgets
- Help
  - User Interface Tour
  - Keyboard ShortcutsH
  - Edit Keyboard Shortcuts

In [1]:

```
# Added the following lines to upload dataset to google colab, allows me to work

# on labs from multiple machines.

# from google.colab import files

# uploaded = files.upload()
```

In [35]:

```
xxxxxxxxxx


# !pip install pmdarima

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

import seaborn as sns

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.seasonal import seasonal_decompose

from pmdarima.arima import auto_arima

from math import sqrt

from sklearn.metrics import mean_squared_error

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

from sklearn.preprocessing import StandardScaler

from imblearn.over_sampling import SMOTE

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn import svm

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import roc_auc_score

from sklearn.datasets import make_classification

from sklearn.linear_model import SGDClassifier

xxxxxxxxxx


# Main Components

## Component 1:

### Dataset loading and fixing [15 points]

Load the dataset, if there is error then use the given code to bypass the warnings/error
```

Main Components¶

Component 1:¶

Dataset loading and fixing [15 points]¶

Load the dataset, if there is error then use the given code to bypass the warnings/error

In [3]:

```
df = pd.read_csv('Darknet.CSV', error_bad_lines=False)
<ipython-input-3-9aa0dffa2492>:1: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future version. Use on_bad_lines in the future.


  df = pd.read_csv('Darknet.CSV', error_bad_lines=False)
Skipping line 328: expected 85 fields, saw 125



xxxxxxxxxx


Change “Label.1” column name to “Darknet Traffic Type”
```

Change “Label.1” column name to “Darknet Traffic Type”

In [4]:

```
xxxxxxxxxx
```

```
df = df.rename(columns={'Label.1': 'Darknet Traffic Type'})
```

```
xxxxxxxxxx
```

```
Show the distribution of the “Darknet Traffic Type”. Do you see any discrepancies? Is there duplication? If there is any discrepancies fix it.
```

Show the distribution of the “Darknet Traffic Type”. Do you see any discrepancies? Is there duplication? If there is any discrepancies fix it.

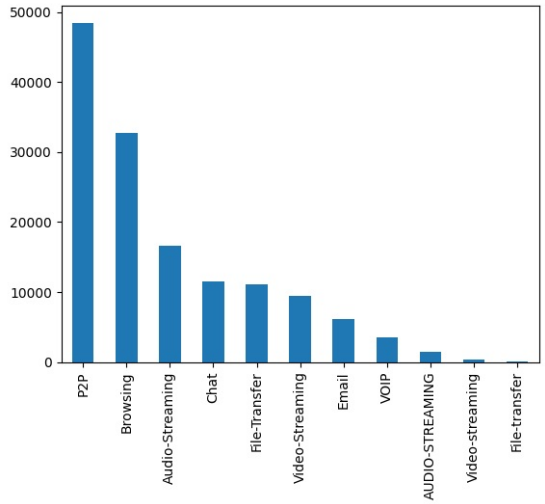
In [5]:

```
xxxxxxxxxx
```

```
df['Darknet Traffic Type'].value_counts().plot(kind='bar')
```

Out[5]:

```
<Axes: >
```




```
xxxxxxxxxx
```

```
Show the fixed distribution in a pie chart using Python, with percentages for each type.
```

Show the fixed distribution in a pie chart using Python, with percentages for each type.

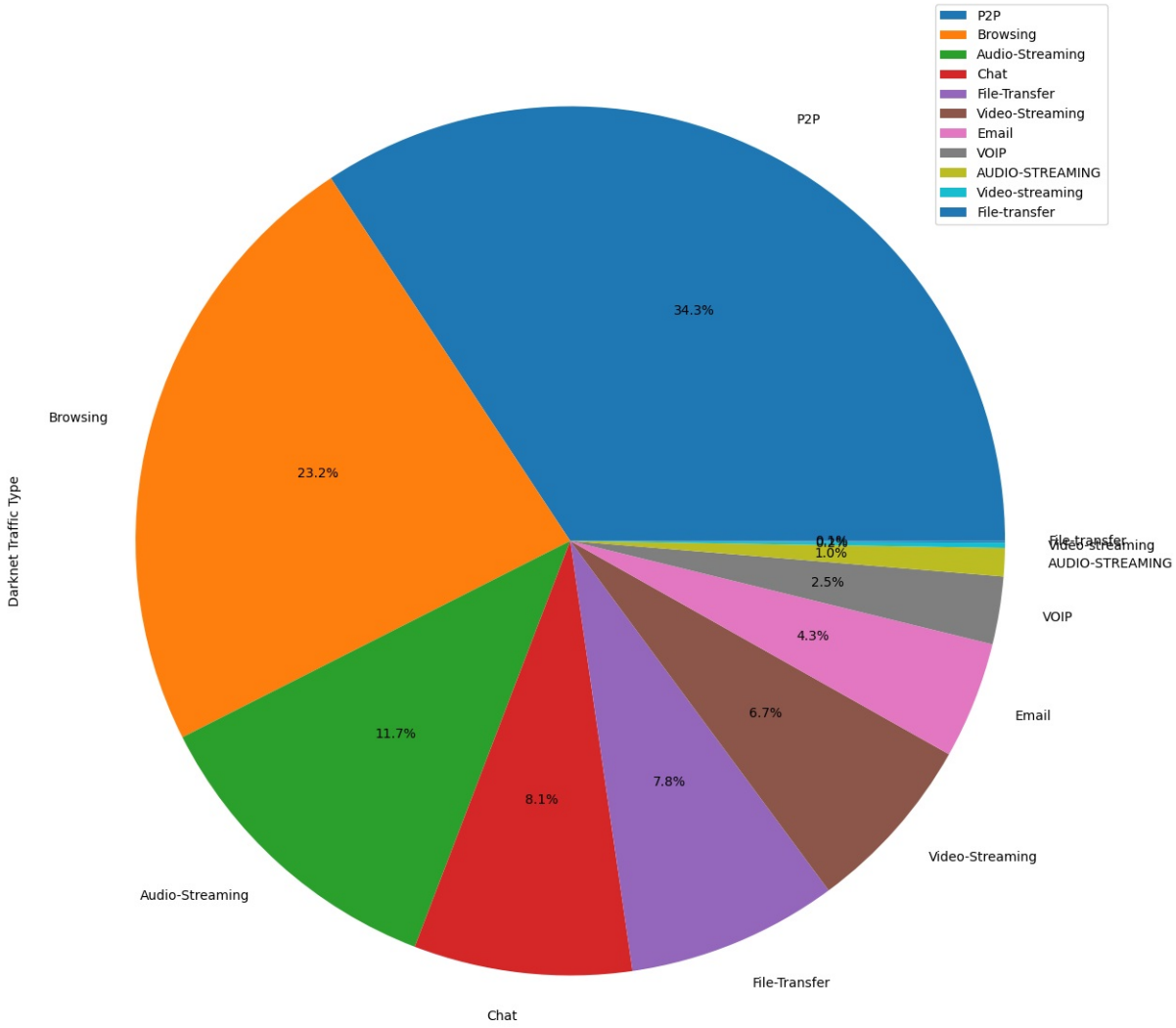
In [6]:

```
xxxxxxxxxx
```

```
df['Darknet Traffic Type'].value_counts().plot(kind='pie', legend=True, figsize=(15,15), autopct='%1.1f%%')
```

Out[6]:

```
<Axes: ylabel='Darknet Traffic Type'>
```




```
xxxxxxxxxx
```

```
Show bar graph for “Label” column distribution. How many total traffic categories?
```

Show bar graph for “Label” column distribution. How many total traffic categories?

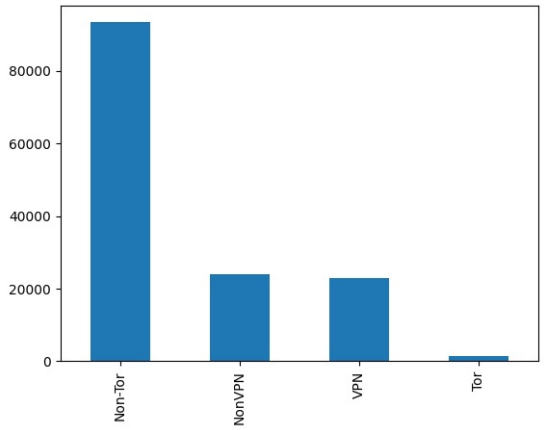
In [7]:

```
xxxxxxxxxx
```

```
df['Label'].value_counts().plot(kind='bar')
```

Out[7]:

```
<Axes: >
```




```
xxxxxxxxxx
```

```
Group “non-VPN” and “non-tor” traffic together as “Benign”. Group “VPN” and “tor” traffic together as “Darknet” for the new set of labels, let’s call is “Final_Label”. This will be later used for binary
```

Group “non-VPN” and “non-tor” traffic together as “Benign”. Group “VPN” and “tor” traffic together as “Darknet” for the new set of labels, let’s call is “Final_Label”. This will be later used for binary classification problem.

In [8]:

```
xxxxxxxxxx
```

```
def get_final_label(s):
  s = s.lower()
  if s == 'non-tor' or s == 'nonvpn':
```

```
    return 'Benign'
  elif s == 'vpn' or s == 'tor':
    return 'Darknet'
  else:
    return 'Unknown'
```

In [9]:

```
xxxxxxxxxx


def get_final_label_num(s):
  s = s.lower()
  if s == 'non-tor' or s == 'nonvpn':
    return 0
  elif s == 'vpn' or s == 'tor':
    return 1
  else:
    return 0
```

In [10]:

```
xxxxxxxxxx


labels = df['Label'].to_list()
final_labels = list(map(get_final_label, labels))
df['Final_Label'] = final_labels
num_labels = df['Label'].to_list()
final_num_labels = list(map(get_final_label_num, num_labels))
df['Final_Label_num'] = final_num_labels
xxxxxxxxxx


If there is null or nan or infinite for any rows, either remove it or use fillna, so that no null value is present in the dataframe. Show the result for your_dataframe.isnull().sum()
```

If there is null or nan or infinite for any rows, either remove it or use fillna, so that no null value is present in the dataframe. Show the result for your_dataframe.isnull().sum()

In [11]:

```
xxxxxxxxxx


df = df.fillna(0)
df.isnull().sum()
```

Out[11]:

```
Flow ID                 0
Src IP                  0
Src Port                0
Dst IP                  0
Dst Port                0
                       ..
Idle Min                0
Label                   0
Darknet Traffic Type    0
Final_Label             0
Final_Label_num         0
Length: 87, dtype: int64
```

```
xxxxxxxxxx


# 2. Pre-processing the data [10 points]
```

2. Pre-processing the data [10 points]¶

In [12]:

```
xxxxxxxxxx


# n-gram function
def create_grams(ip):
    parts = ip.split('.')
    one_gram = parts[0]
    two_gram = parts[0] + " " + parts[1]
    three_gram = parts[0] + " " + parts[1] + " " + parts[2]
    return one_gram, two_gram, three_gram
xxxxxxxxxx


Create a new “hour” column from the “timestamp” column for hour based feature extraction
```

Create a new “hour” column from the “timestamp” column for hour based feature extraction

In [13]:

```
xxxxxxxxxx


df['Timestamp'] = pd.to_datetime(df['Timestamp'])
```

In [14]:

```
xxxxxxxxxx


def get_list_of_hours(dts):
  return dts.hour
```

In [15]:

```
xxxxxxxxxx


datetimes = df['Timestamp'].to_list()
hours = list(map(get_list_of_hours, datetimes))
df['hour'] = hours
```

```
xxxxxxxxxx
```

```
Create a column for “src_ip_1gram”. For example, if “src_ip” is 10.35.192.5 then “src_ip_1gram” is 10.
```

```
Create a column for “src_ip_2gram”. For example, if “src_ip” is 10.35.192.5 then “src_ip_2gram” is 10 35
```

```
Create a column for “src_ip_3gram”. For example, if “src_ip” is 10.35.192.5 then “src_ip_3gram” is 10 35 192
```

Create a column for “src_ip_1gram”. For example, if “src_ip” is 10.35.192.5 then “src_ip_1gram” is 10.

Create a column for “src_ip_2gram”. For example, if “src_ip” is 10.35.192.5 then “src_ip_2gram” is 10 35

Create a column for “src_ip_3gram”. For example, if “src_ip” is 10.35.192.5 then “src_ip_3gram” is 10 35 192

In [16]:

```
xxxxxxxxxx
```

```
src_ip = df['Src IP'].to_list()
```

```
one_gram = []
two_gram = []
three_gram = []
```

```
for s in src_ip:
  one, two, three = create_grams(s)
  one_gram.append(one)
  two_gram.append(two)
  three_gram.append(three)
```

```
df['src_ip_1gram'] = one_gram
df['src_ip_2gram'] = two_gram
df['src_ip_2gram'] = three_gram
```

```
xxxxxxxxxx
```

```
e. How many unique src_ip in the dataset? How many unique dest_ip? How many unique IPs in total combining src_ip and dest_ip?
```

e. How many unique src_ip in the dataset? How many unique dest_ip? How many unique IPs in total combining src_ip and dest_ip?

In [17]:

```
xxxxxxxxxx
```

```
print('Number of Unique Src Ip\'s: ' + str(len(pd.unique(df['Src IP']))))
print('Number of Unique Dest Ip\'s: ' + str(len(pd.unique(df['Dst IP']))))
total_uniqu = len(pd.unique(df['Src IP'])) + len(pd.unique(df['Dst IP']))
print('Unique Ip\'s Total: ' + str(total_uniqu))
```

```
Number of Unique Src Ip's: 3914
Number of Unique Dest Ip's: 7197
Unique Ip's Total: 11111
```

```
xxxxxxxxxx
```

```
Irrelevant column removal: FlowID, TimeStamp, Src IP, Dest IP
```

Irrelevant column removal: FlowID, TimeStamp, Src IP, Dest IP

In [18]:

```
xxxxxxxxxx
```

```
df = df.drop(columns=['Flow ID', 'Src IP', 'Dst IP'])
```

```
xxxxxxxxxx
```

```
If there is categorical columns other that the labels (“Darknet traffic types”,“Label”  or “Final_Label”), then use one-hot encoding to use those columns in the model.
```

If there is categorical columns other that the labels (“Darknet traffic types”,“Label” or “Final_Label”), then use one-hot encoding to use those columns in the model.

In [19]:

```
xxxxxxxxxx
```

```
print(df.select_dtypes(include='object'))
```

```
encoder = OneHotEncoder()
onehotarray = encoder.fit_transform(df[["Darknet Traffic Type"]]).toarray()
items = [f'{"Darknet Traffic Type"}_{item}' for item in encoder.categories_[0]]
df2 = pd.DataFrame(onehotarray, columns=items)
df = pd.concat([df,df2], axis=1)
```

```
encoder = OneHotEncoder()
onehotarray = encoder.fit_transform(df[["Label"]]).toarray()
items = [f'{"Label"}_{item}' for item in encoder.categories_[0]]
df2 = pd.DataFrame(onehotarray, columns=items)
df = pd.concat([df,df2], axis=1)
```

```
encoder = OneHotEncoder()
onehotarray = encoder.fit_transform(df[["Final_Label"]]).toarray()
items = [f'{"Final_Label"}_{item}' for item in encoder.categories_[0]]
df2 = pd.DataFrame(onehotarray, columns=items)
df = pd.concat([df,df2], axis=1)
```

```
print(df.select_dtypes(exclude='object'))
```

```
         Label Darknet Traffic Type Final_Label src_ip_1gram src_ip_2gram
0       Non-Tor      AUDIO-STREAMING      Benign           10   10 152 152
1       Non-Tor      AUDIO-STREAMING      Benign           10   10 152 152
2       Non-Tor      AUDIO-STREAMING      Benign           10   10 152 152
3       Non-Tor      AUDIO-STREAMING      Benign           10   10 152 152
4       Non-Tor      AUDIO-STREAMING      Benign           10   10 152 152
...         ...                  ...         ...          ...          ...
141525      VPN                VOIP     Darknet           10       10 8 8
141526      VPN                VOIP     Darknet           10       10 8 8
141527      VPN                VOIP     Darknet           10       10 8 8
141528      VPN                VOIP     Darknet           10       10 8 8
141529      VPN                VOIP     Darknet           80   80 239 235

[141530 rows x 5 columns]
        Src Port  Dst Port  Protocol           Timestamp  Flow Duration  \
0          57158       443         6 2015-07-24 16:09:48            229
1          57159       443         6 2015-07-24 16:09:48            407
2          57160       443         6 2015-07-24 16:09:48            431
3          49134       443         6 2015-07-24 16:09:48            359
4          34697     19305         6 2015-07-24 16:09:45       10778451
...          ...       ...       ...                  ...            ...
141525     55219      5355        17 2015-05-22 13:55:03         411806
141526     64207      5355        17 2015-05-22 14:09:05         411574
141527     61115      5355        17 2015-05-22 14:19:31         422299
141528     64790      5355        17 2015-05-22 14:29:55         411855
141529     11666     60245        17 2015-05-22 14:31:23      119990044

        Total Fwd Packet  Total Bwd packets  Total Length of Fwd Packet  \
0                      1                  1                           0
1                      1                  1                           0
2                      1                  1                           0
3                      1                  1                           0
4                    591                400                       64530
...                  ...                ...                         ...
141525                 2                  0                          44
141526                 2                  0                          44
141527                 2                  0                          44
141528                 2                  0                          44
141529              5995               6000                      497585

        Total Length of Bwd Packet  Fwd Packet Length Max  ...  \
0                                0                      0  ...
1                                0                      0  ...
2                                0                      0  ...
3                                0                      0  ...
4                             6659                    131  ...
...                            ...                    ...  ...
141525                           0                     22  ...
141526                           0                     22  ...
141527                           0                     22  ...
141528                           0                     22  ...
141529                      498000                     83  ...

        Darknet Traffic Type_P2P  Darknet Traffic Type_VOIP  \
0                            0.0                        0.0
1                            0.0                        0.0
2                            0.0                        0.0
3                            0.0                        0.0
4                            0.0                        0.0
...                          ...                        ...
141525                       0.0                        1.0
141526                       0.0                        1.0
141527                       0.0                        1.0
141528                       0.0                        1.0
141529                       0.0                        1.0

        Darknet Traffic Type_Video-Streaming  \
0                                        0.0
1                                        0.0
2                                        0.0
3                                        0.0
4                                        0.0
...                                      ...
141525                                   0.0
141526                                   0.0
141527                                   0.0
141528                                   0.0
141529                                   0.0

        Darknet Traffic Type_Video-streaming  Label_Non-Tor  Label_NonVPN  \
0                                        0.0            1.0           0.0
1                                        0.0            1.0           0.0
2                                        0.0            1.0           0.0
3                                        0.0            1.0           0.0
4                                        0.0            1.0           0.0
...                                      ...            ...           ...
141525                                   0.0            0.0           0.0
141526                                   0.0            0.0           0.0
141527                                   0.0            0.0           0.0
141528                                   0.0            0.0           0.0
141529                                   0.0            0.0           0.0

        Label_Tor  Label_VPN  Final_Label_Benign  Final_Label_Darknet
0             0.0        0.0                 1.0                  0.0
1             0.0        0.0                 1.0                  0.0
2             0.0        0.0                 1.0                  0.0
3             0.0        0.0                 1.0                  0.0
4             0.0        0.0                 1.0                  0.0
...           ...        ...                 ...                  ...
141525        0.0        1.0                 0.0                  1.0
141526        0.0        1.0                 0.0                  1.0
141527        0.0        1.0                 0.0                  1.0
141528        0.0        1.0                 0.0                  1.0
141529        0.0        1.0                 0.0                  1.0

[141530 rows x 99 columns]


xxxxxxxxxx



Again make sure, nan values are dropped or fixed!
```

Again make sure, nan values are dropped or fixed!

In [20]:

```
xxxxxxxxxx



df = df.fillna(0)

df.isnull().sum()
```

Out[20]:

```
Src Port               0
Dst Port               0
Protocol               0
Timestamp              0
Flow Duration          0
                      ..
Label_NonVPN           0
Label_Tor              0
Label_VPN              0
Final_Label_Benign     0
```

```
Final_Label_Darknet    0
Length: 104, dtype: int64
```

```
xxxxxxxxxx
```

```
Do standard scaling of the data
```

Do standard scaling of the data

In [21]:

```
xxxxxxxxxx
```

```
numeric_df = df._get_numeric_data()

numeric_df = numeric_df.fillna(0)

numeric_df.replace([np.inf, -np.inf], 0, inplace=True)

sc = StandardScaler()

sc.fit(numeric_df)

scaled_df = sc.transform(numeric_df)
```

```
xxxxxxxxxx
```

```
a. Show a kernel density plot by grouping the data into “darknet” vs “benign” for the hourly distribution of traffic activities (x-axis should represent hours).
```

a. Show a kernel density plot by grouping the data into “darknet” vs “benign” for the hourly distribution of traffic activities (x-axis should represent hours).

In [22]:

```
xxxxxxxxxx
```

```
df['Final_Label']
```

Out[22]:

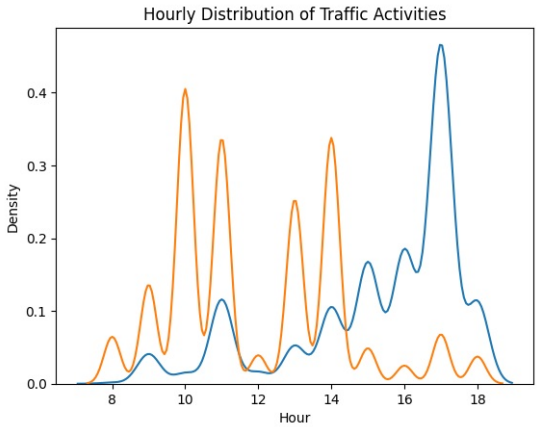
```
0          Benign
1          Benign
2          Benign
3          Benign
4          Benign
           ...
141525    Darknet
141526    Darknet
141527    Darknet
141528    Darknet
141529    Darknet
Name: Final_Label, Length: 141530, dtype: object
```

In [23]:

```
xxxxxxxxxx
```

```
darknet_data = df[df['Final_Label'] == 'Darknet']

benign_data = df[df['Final_Label'] == 'Benign']

darknet_data['hour'] = darknet_data['Timestamp'].dt.hour

benign_data['hour'] = benign_data['Timestamp'].dt.hour

sns.kdeplot(data=darknet_data, x='hour', label='Darknet')

sns.kdeplot(data=benign_data, x='hour', label='Benign')

plt.title('Hourly Distribution of Traffic Activities')

plt.xlabel('Hour')

plt.ylabel('Density')

plt.show()
```

```
<ipython-input-23-4cad6f33708a>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  darknet_data['hour'] = darknet_data['Timestamp'].dt.hour
<ipython-input-23-4cad6f33708a>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  benign_data['hour'] = benign_data['Timestamp'].dt.hour
```




```
xxxxxxxxxx
```

```
b. Show the probability distribution of certain traffic types (e.g., Audio-Streaming, P2P, email, …) given the row is “benign” (Final_Label= “benign”).
```

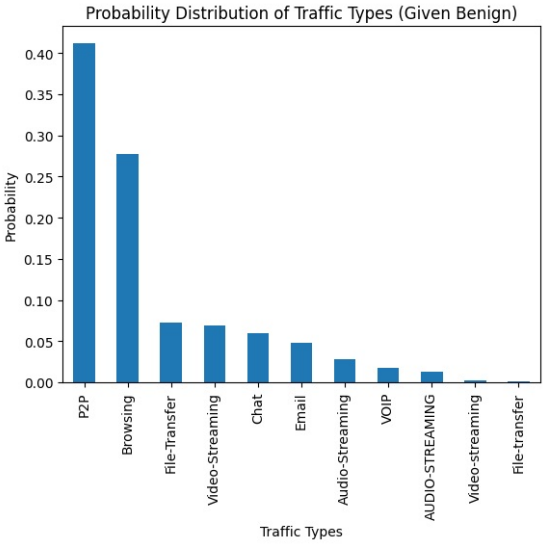
b. Show the probability distribution of certain traffic types (e.g., Audio-Streaming, P2P, email, ...) given the row is “benign” (Final_Label= “benign”).

In [24]:

```
xxxxxxxxxx
```

```
benign_traffic = df[df['Final_Label'] == 'Benign']

prob_distribution = benign_traffic['Darknet Traffic Type'].value_counts(normalize=True)

prob_distribution.plot(kind='bar')
```

```
plt.title('Probability Distribution of Traffic Types (Given Benign)')
plt.xlabel('Traffic Types')
plt.ylabel('Probability')
plt.show()
```




```
xxxxxxxxxx
```

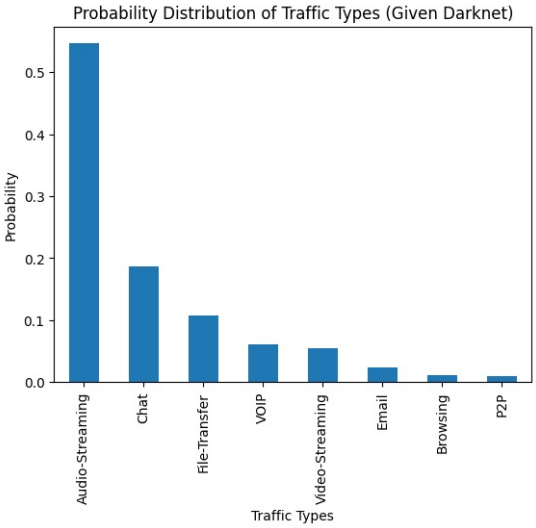
```
Show the probability distribution of certain traffic types (e.g., Audio-Streaming, P2P, email, …) given the data is “darknet” (Final_Label = “darknet”).
```

Show the probability distribution of certain traffic types (e.g., Audio-Streaming, P2P, email, …) given the data is “darknet” (Final_Label = “darknet”).

In [25]:

```
xxxxxxxxxx
```

```
darknet_traffic = df[df['Final_Label'] == 'Darknet']
prob_distribution = darknet_traffic['Darknet Traffic Type'].value_counts(normalize=True)
prob_distribution.plot(kind='bar')
plt.title('Probability Distribution of Traffic Types (Given Darknet)')
plt.xlabel('Traffic Types')
plt.ylabel('Probability')
plt.show()
```




```
xxxxxxxxxx
```

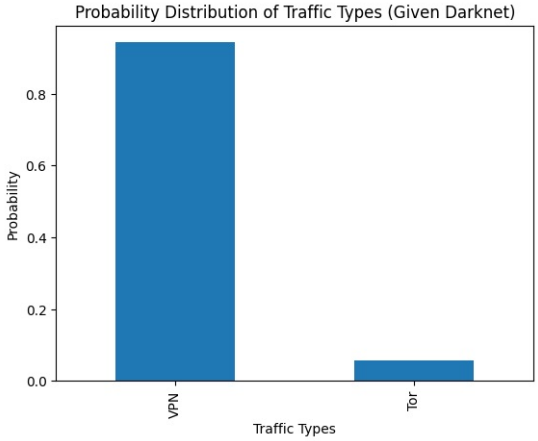
```
d. Analyze and present different columns/variables and see if there is any significant difference between the groups “darknet” vs “benign” for any specific column? Show evidence if found such relationshi
```

d. Analyze and present different columns/variables and see if there is any significant difference between the groups “darknet” vs “benign” for any specific column? Show evidence if found such relationship.

In [26]:

```
xxxxxxxxxx
```

```
prob_distribution = darknet_traffic['Label'].value_counts(normalize=True)
prob_distribution.plot(kind='bar')
plt.title('Probability Distribution of Traffic Types (Given Darknet)')
plt.xlabel('Traffic Types')
plt.ylabel('Probability')
plt.show()
```



xxxxxxxxxx

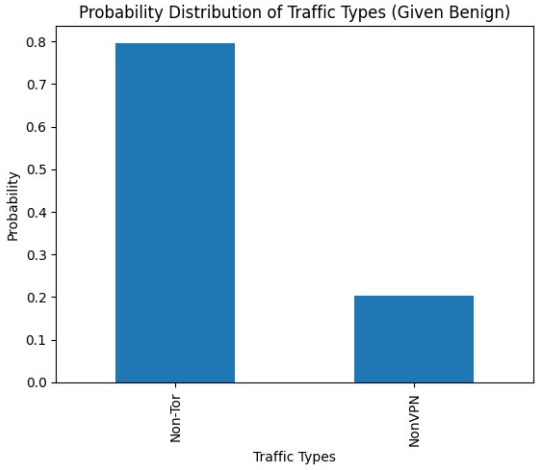
Analyze and present different columns/variables and see if there is any significant difference among the darknet Traffic Types (Column: “Darknet Traffic Type”) for any specific column? Show evidence if

Analyze and present different columns/variables and see if there is any significant difference among the darknet Traffic Types (Column: “Darknet Traffic Type”) for any specific column? Show evidence if found such relationship.

In [27]:

xxxxxxxxxx

```
prob_distribution = benign_traffic['Label'].value_counts(normalize=True)
prob_distribution.plot(kind='bar')
plt.title('Probability Distribution of Traffic Types (Given Benign)')
plt.xlabel('Traffic Types')
plt.ylabel('Probability')
plt.show()
```



xxxxxxxxxx

To the standard ML based modeling that we learned through the labs. [Make your judgements and knowledge to make sure if any intermediate step is required or optional]

To the standard ML based modeling that we learned through the labs. [Make your judgements and knowledge to make sure if any intermediate step is required or optional]

xxxxxxxxxx

Do you need to fix data imbalance issue? If needed, apply a technique to do so.

Do you need to fix data imbalance issue? If needed, apply a technique to do so.

In [300]:

xxxxxxxxxx

```
X = df.drop('Final_Label', axis=1)
Y = df['Final_Label']
original_distribution = Y.value_counts()
print("Original Training Set Distribution:")
print(original_distribution)
x1 = 117219
y1 = 24311
percent = (y1/x1) * 100
print(f'Data Balance: {percent}%')
```

```
Original Training Set Distribution:
Benign     117219
Darknet     24311
Name: Final_Label, dtype: int64
Data Balance: 20.739811805253414%
```

xxxxxxxxxx

Do apply at least 6 State-of-the -art ML models of your choice (e.g., Random Forest, Decision Tree, SVM, Logistic Regression, KNN, etc.) for the binary classification job of identifying “benign” vs “dar

Do apply at least 6 State-of-the -art ML models of your choice (e.g., Random Forest, Decision Tree, SVM, Logistic Regression, KNN, etc.) for the binary classification job of identifying “benign” vs “darknet” traffic labels (reference column “Final_label”).

In [29]:

xxxxxxxxxx

```
y = df['Final_Label_num']

x_train,x_test,y_train,y_test = train_test_split(numeric_df,y,test_size=0.2)

xxxxxxxxxx


# Random Forest Classifier:
```

Random Forest Classifier:¶

In [30]:

```
xxxxxxxxxx


classifier = RandomForestClassifier()

classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy score:", accuracy)

print(classification_report(y_test, y_pred))

tn, fp, fn, tp = confusion_matrix(y_test, y_pred, labels=[0, 1]).ravel()

fpr = fp/(fp+tn)

fnr = fn/(tp+fn)

print(f'FPR: {fpr}\nFNR: {fnr}')
```

```
Accuracy score: 0.9159663865546218
              precision    recall  f1-score   support

           0       0.92      0.97      0.95       318
           1       0.60      0.31      0.41        39

    accuracy                           0.90       357
   macro avg       0.76      0.64      0.68       357
weighted avg       0.88      0.90      0.89       357

FPR: 0.0251572327044025 17
FNR: 0.6923076923076923
```

```
xxxxxxxxxx


# Decision Tree Classifier
```

Decision Tree Classifier¶

In [31]:

```
xxxxxxxxxx


classifier = DecisionTreeClassifier()

classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy score:", accuracy)

print(classification_report(y_test, y_pred))

tn, fp, fn, tp = confusion_matrix(y_test, y_pred, labels=[0, 1]).ravel()

fpr = fp/(fp+tn)

fnr = fn/(tp+fn)

print(f'FPR: {fpr}\nFNR: {fnr}')
```

```
Accuracy score: 0.7425265188042 43
              precision    recall  f1-score   support

           0       0.92      0.52      0.66       509
           1       0.67      0.96      0.79       528

    accuracy                           0.74      1037
   macro avg       0.80      0.74      0.73      1037
weighted avg       0.80      0.74      0.73      1037

FPR: 0.481335952848723
FNR: 0.041666666666666664
```

```
xxxxxxxxxx


# Logistic Regression
```

Logistic Regression¶

In [32]:

```
classifier = LogisticRegression()

classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy score:", accuracy)

print(classification_report(y_test, y_pred))

tn, fp, fn, tp = confusion_matrix(y_test, y_pred, labels=[0, 1]).ravel()

fpr = fp/(fp+tn)

fnr = fn/(tp+fn)

print(f'FPR: {fpr}\nFNR: {fnr}')
```

```
Accuracy score: 0.8307425987423162
              precision    recall  f1-score   support

           0       0.83      1.00      0.91     23515
           1       0.00      0.00      0.00      4791

    accuracy                           0.83     28306
   macro avg       0.42      0.50      0.45     28306
weighted avg       0.69      0.83      0.75     28306

FPR: 0.0
FNR: 1.0
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
  _warn_prf(average, modifier, msg_start, len(result))
```

```
xxxxxxxxxx
```

```
# Scalable Vector Machine:
```

Scalable Vectore Machine:¶

In [33]:

```
x_train,x_test,y_train,y_test = train_test_split(scaled_df,y,test_size=0.2)

classifier = svm.SVC()

classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy score:", accuracy)

print(classification_report(y_test, y_pred))

tn, fp, fn, tp = confusion_matrix(y_test, y_pred, labels=[0, 1]).ravel()

fpr = fp/(fp+tn)

fnr = fn/(tp+fn)

print(f'FPR: {fpr}\nFNR: {fnr}')
```

```
Accuracy score: 0.9469623915139826
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       509
           1       0.94      0.96      0.95       528

    accuracy                           0.95      1037
   macro avg       0.95      0.95      0.95      1037
weighted avg       0.95      0.95      0.95      1037

FPR: 0.06483300589390963
FNR: 0.041666666666666664
```

```
xxxxxxxxxx
```

```
# K Neighbor Classifier
```

K Neighbor Classifier¶

In [34]:

```
xxxxxxxxxx
```

```
classifier = KNeighborsClassifier()

classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy score:", accuracy)

print(classification_report(y_test, y_pred))

tn, fp, fn, tp = confusion_matrix(y_test, y_pred, labels=[0, 1]).ravel()

fpr = fp/(fp+tn)

fnr = fn/(tp+fn)

print(f'FPR: {fpr}\nFNR: {fnr}')
```

```
Accuracy score: 0.7965284474445516
              precision    recall  f1-score   support

           0       0.91      0.65      0.76       509
           1       0.73      0.94      0.82       528

    accuracy                           0.80      1037
   macro avg       0.82      0.79      0.79      1037
weighted avg       0.82      0.80      0.79      1037

FPR: 0.35363457760314343
FNR: 0.058712121212121215
```

```
xxxxxxxxxx
```

```
# Stochastic Gradient Descent
```

Stochastic Gradient Descent¶

In [38]:

```
xxxxxxxxxx
```

```
classifier = SGDClassifier(loss="hinge", penalty="l2", max_iter=5)

classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy score:", accuracy)

print(classification_report(y_test, y_pred))

tn, fp, fn, tp = confusion_matrix(y_test, y_pred, labels=[0, 1]).ravel()

fpr = fp/(fp+tn)
```

```
fnr = fn/(tp+fn)

print(f'FPR: {fpr}\nFNR: {fnr}')
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:702: ConvergenceWarning: Maximum number of iteration reached before convergence. Consider increasing max_iter to impro
  warnings.warn(

              precision    recall  f1-score   support

           0       0.92      0.97      0.95       318
           1       0.60      0.31      0.41        39

    accuracy                           0.90       357
   macro avg       0.76      0.64      0.68       357
weighted avg       0.88      0.90      0.89       357

FPR: 0.025157232704402517
FNR: 0.6923076923076923



xxxxxxxxxx



Provide a tabular presentation of the standard evaluation metric to report the best performing model in your case study. Do you find one single best model that outperforms all other in every metrics? Or
```

Provide a tabular presentation of the standard evaluation metric to report the best performing model in your case study. Do you find one single best model that outperforms all other in every metrics? Or you can report individual metric based highest/top performing models.

In [303]:

```
xxxxxxxxxx



# from google.colab import files

# uploaded = files.upload()

df2 = pd.read_csv('FinalProject - Sheet1.csv')

print(df2)
            Unnamed: 0       FPR       FNR  Accuracy  F1-Score
0                  KNC  0.353635  0.058712  0.796528      0.76
1                   RF  0.025157  0.692308  0.915966      0.95
2                  SVM  0.064833  0.041667  0.946962      0.95
3        Decision Tree  0.481336  0.041667  0.742527      0.73
4  Logistic Regression  0.006452  0.021277  0.830743      0.91
5                  SGD  0.025157  0.692308  0.855634      0.95


xxxxxxxxxx



Using the timseries data for Darknet events (e.g., only Final_Label=“darknet”). Can we generate a model to forecast predictions fr number of Darknet events?



Hint: in this task you need to create a time-series DF first with days (dates) and number of darknet traffic events. Once that is generated you separate the train-test. Make sure training has 90% of the
```

Using the timseries data for Darknet events (e.g., only Final_Label=“darknet”). Can we generate a model to forecast predictions fr number of Darknet events?

Hint: in this task you need to create a time-series DF first with days (dates) and number of darknet traffic events. Once that is generated you separate the train-test. Make sure training has 90% of the days ad you will be predicting te remaining 10 of days.

In [297]:

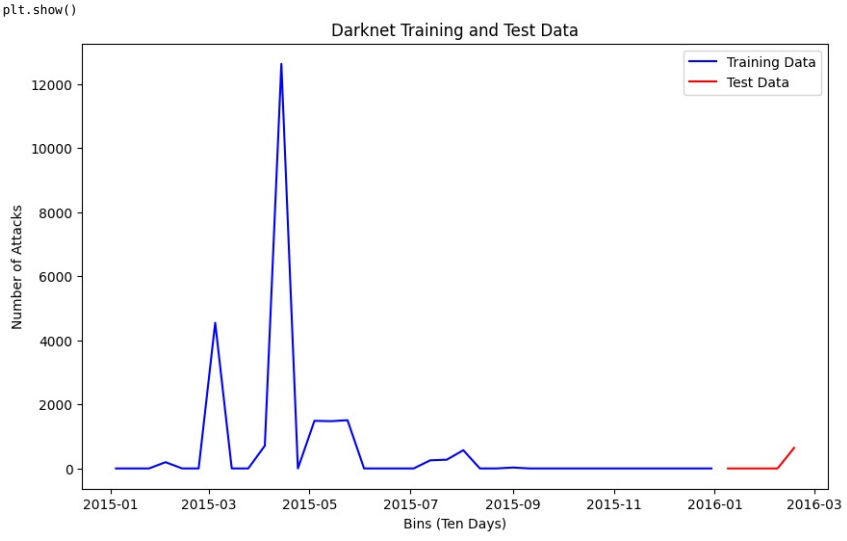
```
xxxxxxxxxx



df.sort_values(by='Timestamp')

ts = pd.DataFrame()

ts['Dates'] = df['Timestamp']

ts['#Attacks'] = df['Final_Label_num']

ts = ts.groupby(pd.Grouper(key='Dates', axis=0, freq='10D')).sum()
```

In [301]:

```
xxxxxxxxxx



train_size = int(len(ts) * 0.9)

train = ts[:train_size]

test = ts[train_size:]
```

In [295]:

```
xxxxxxxxxx



plt.figure(figsize=(10, 6))

plt.plot(train.index, train['#Attacks'], color='blue', label='Training Data')

plt.plot(test.index, test['#Attacks'], color='red', label='Test Data')

plt.xlabel('Bins (Ten Days)')

plt.ylabel('Number of Attacks')

plt.title('Darknet Training and Test Data')

plt.legend()

plt.show()
```



In [296]:

```
xxxxxxxxxx
```

```
model = auto_arima(train, trace=True, error_action='ignore', suppress_warnings=True)

model.fit(train)

forecast = model.predict(n_periods=len(test))

forecast = pd.DataFrame(forecast,index = test.index,columns=['Prediction'])

rms = sqrt(mean_squared_error(test,forecast))

print("RMSE: ", rms)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.48 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=677.173, Time=0.03 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=679.182, Time=0.05 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=679.177, Time=0.13 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=678.282, Time=0.03 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=681.356, Time=0.09 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0] intercept
Total fit time: 0.838 seconds
RMSE:  572.1452492744728
```