STREAM ANALYTICS
CSL7970

POST GRADUATE DIPLOMA IN DATA ENGINEERING

ASSIGNMENT 1

SUBMITTED BY:

NIRAJ BHAGCHANDANI [G23AI2087]
BHAVESH ARORA [G23AI2126]

COURSE INSTRUCTOR

NITIN AWATHARE

SUBMISSION DATE: 23rd DECEMBER, 2024

DEPARTMENT OF AIDE
INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR

1. Retrieve the data stream that represents only the TCP traffic flowing through the network. To do this, you should become familiar with the SiLK suite, a tool for analyzing the network and its components.

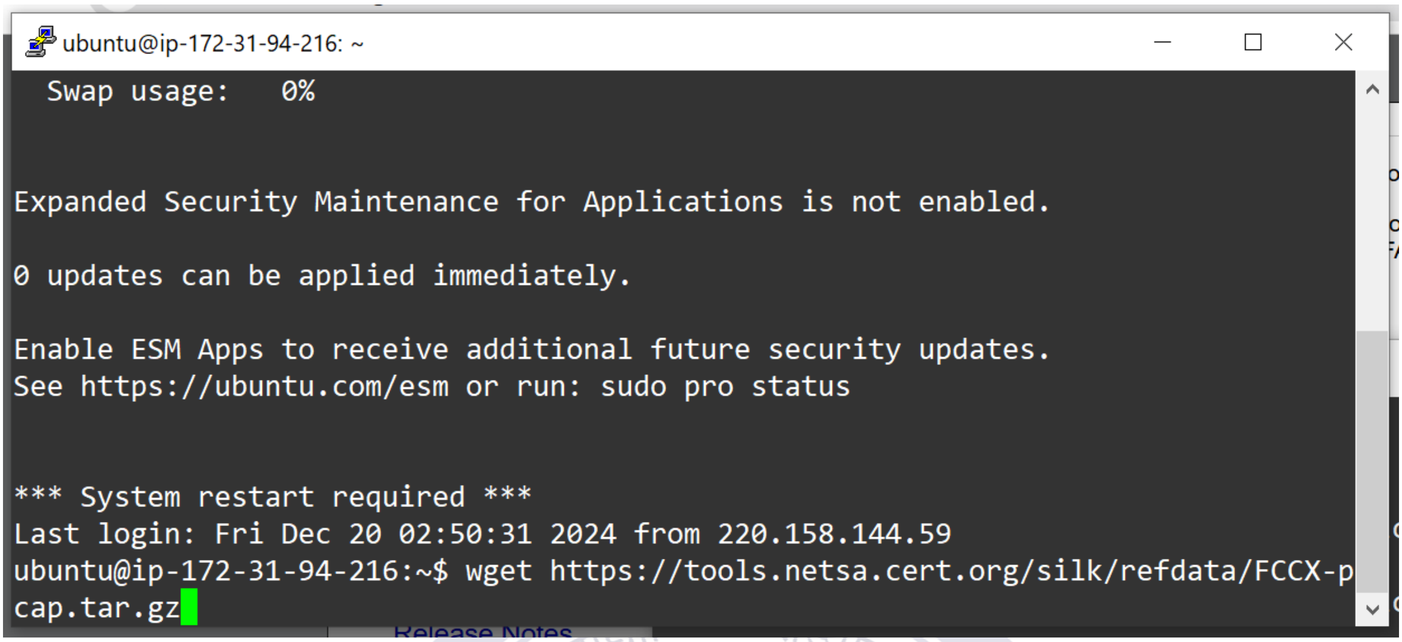
Step 1: Installing various packages as well as downloading certain files from NETSA website, so here are the commands for the same.

```
$ sudo apt update
$ sudo apt upgrade -y
$ sudo apt install dos2unix -y
$ sudo apt install build-essential -y
$ sudo apt-get install s3cmd -y
$ sudo apt install libglib2.0-dev liblzo2-dev zlib1g-dev libgnutls28-dev libpcap-dev python3.8-dev
$ sudo apt install libmaxminddb-dev libssl-dev
$ sudo apt install libpcre3 libpcre3-dev
$ sudo apt install unzip
$ sudo apt install libssl-dev
$ sudo apt install build-essential autoconf automake
$ sudo apt install build-essential autoconf automake libtool libpcre3-dev libssl-dev
$ sudo apt install gcc-11 g++-11
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
$ unzip awscliv2.zip
$ sudo ./aws/install
$ aws --version

$ wget https://tools.netsa.cert.org/releases/yaf-2.16.1.tar.gz
$ wget https://tools.netsa.cert.org/releases/silk-3.23.1.tar.gz
$ wget https://tools.netsa.cert.org/releases/libfixbuf-2.5.0.tar.gz
$ wget https://tools.netsa.cert.org/releases/analysis-pipeline-5.11.4.tar.gz
$ wget https://tools.netsa.cert.org/releases/netsa-python-1.5.tar.gz
$ wget https://tools.netsa.cert.org/releases/rayon-1.4.3.tar.gz
$ wget https://tools.netsa.cert.org/releases/libschemaTools-1.4.tar.gz
```

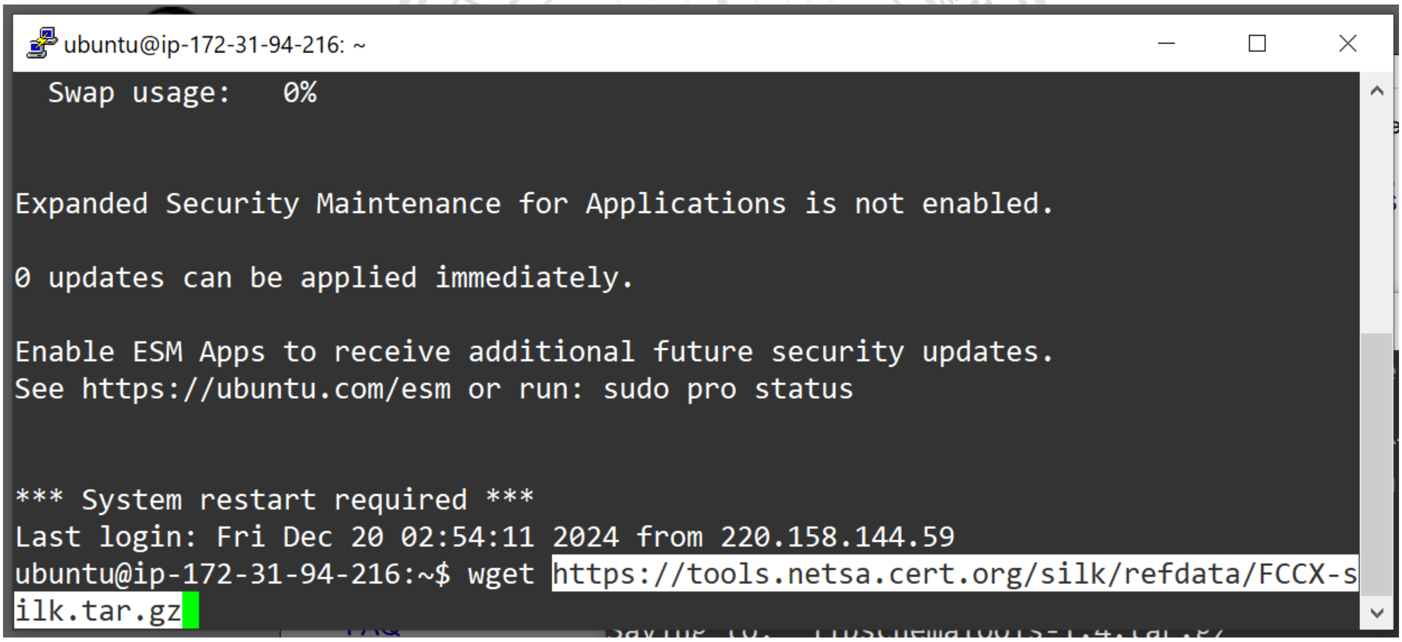
Step – 2 Installing pre-requisite softwares before installing packages.

```
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ sudo apt install -y build-essential gcc g++ make zlib1g-dev libglib2.0-dev libssl-dev libcurl4-openssl-dev libpcap-dev libtool automake autoconf
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
build-essential is already the newest version (12.10ubuntu1).
gcc is already the newest version (4:13.2.0-7ubuntu1).
g++ is already the newest version (4:13.2.0-7ubuntu1).
make is already the newest version (4.3-4.1build2).
zlib1g-dev is already the newest version (1:1.3.dfsg-3.1ubuntu2.1).
libglib2.0-dev is already the newest version (2.80.0-6ubuntu3.2).
libssl-dev is already the newest version (3.0.13-0ubuntu3.4).
libcurl4-openssl-dev is already the newest version (8.5.0-2ubuntu10.6).
libpcap-dev is already the newest version (1.10.4-4.1ubuntu3).
libtool is already the newest version (2.4.7-7build1).
automake is already the newest version (1:1.16.5-1.3ubuntu1).
autoconf is already the newest version (2.71-3).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
ubuntu@ip-172-31-94-216:~/silk-3.23.1$
```

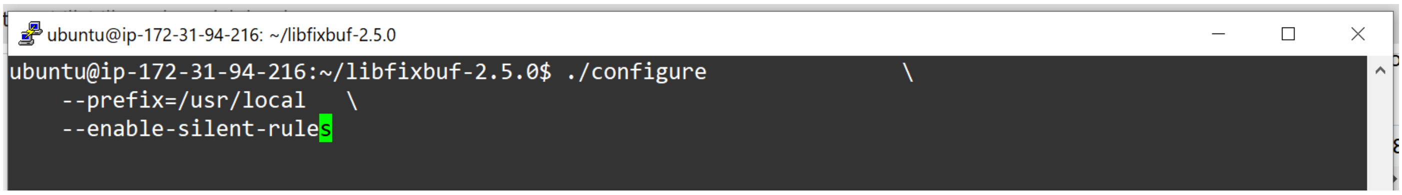
Step – 3 Downloading FCCX-pcap.tar.gz file from NETSA



Step – 4 Downloading FCCX-silk.tar.gz file from NETSA



Step-5 Configuring libfixbuf package



Step – 6 Compiling libfixbuf package

ubuntu@ip-172-31-94-216: ~/libfixbuf-2.5.0

```
    * Install directory:            /usr/local
    * pkg-config path:
    * GLIB:                         -lgthread-2.0 -pthread -lglib-2.0
    * OpenSSL Support:              NO
    * DTLS Support:                 NO
    * SCTP Support:                 NO
    * Spread Toolkit Support:       NO
    * Compiler (CC):                gcc
    * Compiler flags (CFLAGS):      -I. -I$(top_srcdir)/include -Wall -Wextra -Wshadow -Wpointer-arith -W
format=2 -Wunused -Wduplicated-cond -Wwrite-strings -Wmissing-prototypes -Wstrict-prototypes -DNDEBUG -DG
_DISABLE_ASSERT -g -O2
    * Linker flags (LDFLAGS):
    * Libraries (LIBS):             -lpthread

ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$ make
```

Step – 7 Installing libfixbuf using make install command.

ubuntu@ip-172-31-94-216: ~/libfixbuf-2.5.0

```
make[1]: Leaving directory '/home/ubuntu/libfixbuf-2.5.0'
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$ sudo make install
Making install in src
make[1]: Entering directory '/home/ubuntu/libfixbuf-2.5.0/src'
make  install-recursive
make[2]: Entering directory '/home/ubuntu/libfixbuf-2.5.0/src'
Making install in infomodel
make[3]: Entering directory '/home/ubuntu/libfixbuf-2.5.0/src/infomodel'
make  install-am
make[4]: Entering directory '/home/ubuntu/libfixbuf-2.5.0/src/infomodel'
make[5]: Entering directory '/home/ubuntu/libfixbuf-2.5.0/src/infomodel'
make[5]: Nothing to be done for 'install-exec-am'.
```

Step 8 Installing relevant packages for Silk Suite Support

ubuntu@ip-172-31-94-216: ~/libfixbuf-2.5.0

```
python3-pip is already the newest version (24.0+dfsg-1ubuntu1.1).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$ sudo apt install -y \
    build-essential \
    gcc \
    g++ \
    make \
    zlib1g-dev \
    libglib2.0-dev \
    libpcap-dev \
    libtool \
    automake \
    autoconf \
    libfixbuf-dev \
    libssl-dev \
    libcurl4-openssl-dev \
    python3 \
    python3-pip
```

Step – 8 Checking / Verifying the libfixbuf package installed correctly or not.

```
ubuntu@ip-172-31-94-216: ~/libfixbuf-2.5.0
zlib1g-dev is already the newest version (1:1.3.dfsg-3.1ubuntu2.1).
libglib2.0-dev is already the newest version (2.80.0-6ubuntu3.2).
libpcap-dev is already the newest version (1.10.4-4.1ubuntu3).
libtool is already the newest version (2.4.7-7build1).
automake is already the newest version (1:1.16.5-1.3ubuntu1).
autoconf is already the newest version (2.71-3).
libfixbuf-dev is already the newest version (2.4.1+ds-2.1build2).
libssl-dev is already the newest version (3.0.13-0ubuntu3.4).
libcurl4-openssl-dev is already the newest version (8.5.0-2ubuntu10.6).
python3 is already the newest version (3.12.3-0ubuntu2).
python3-pip is already the newest version (24.0+dfsg-1ubuntu1.1).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$ ls /usr/local/lib | grep fixbuf
libfixbuf.a
libfixbuf.la
libfixbuf.so
libfixbuf.so.9
libfixbuf.so.9.1.0
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$
```

Step 9 Coping certain files to ldconfig and reloading it.

```
ubuntu@ip-172-31-94-216: ~/libfixbuf-2.5.0
automake is already the newest version (1:1.16.5-1.3ubuntu1).
autoconf is already the newest version (2.71-3).
libfixbuf-dev is already the newest version (2.4.1+ds-2.1build2).
libssl-dev is already the newest version (3.0.13-0ubuntu3.4).
libcurl4-openssl-dev is already the newest version (8.5.0-2ubuntu10.6).
python3 is already the newest version (3.12.3-0ubuntu2).
python3-pip is already the newest version (24.0+dfsg-1ubuntu1.1).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$ ls /usr/local/lib | grep fixbuf
libfixbuf.a
libfixbuf.la
libfixbuf.so
libfixbuf.so.9
libfixbuf.so.9.1.0
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$ echo "/usr/local/lib" | sudo tee -a /etc/ld.so.conf.d/libfixbu
f.conf
/usr/local/lib
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$ sudo ldconfig
ubuntu@ip-172-31-94-216:~/libfixbuf-2.5.0$
```

Step – 10 Extracting Silk-Suite using tar command.

```
ubuntu@ip-172-31-94-216: ~
ubuntu@ip-172-31-94-216:~$ tar -xvf silk-3.23.1.tar.gz
```

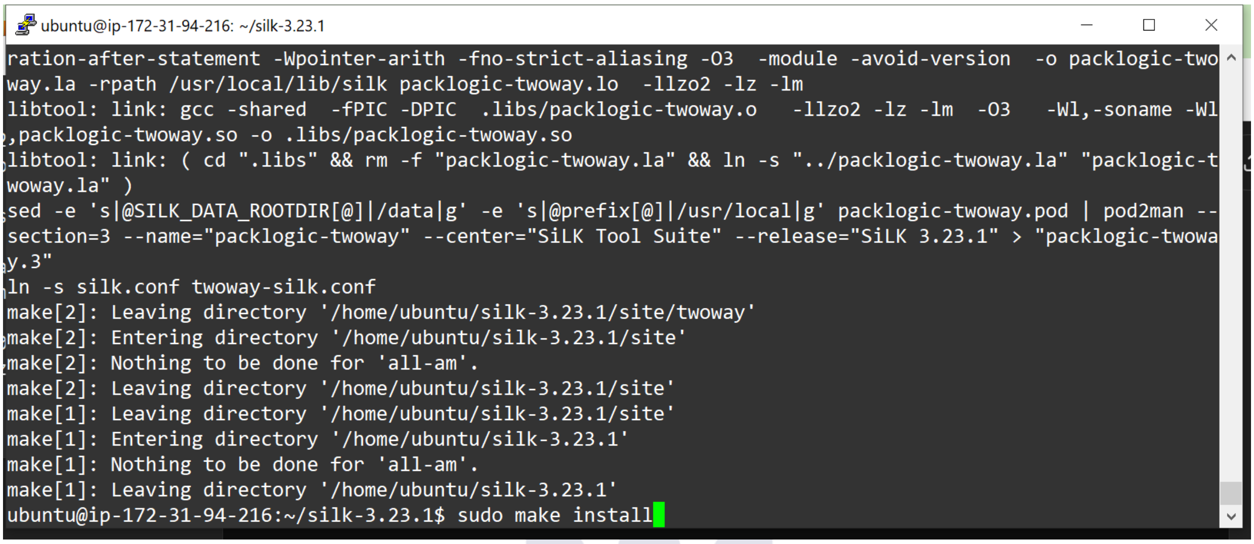
Step – 11 Compiling Silk Suite using configure command.

```
ubuntu@ip-172-31-94-216: ~/silk-3.23.1
silk-3.23.1/src/rwipa/tests/rwipaimport-version.pl
silk-3.23.1/src/rwipa/tests/rwipaexport-no-catalog.pl
silk-3.23.1/src/rwipa/tests/rwipaimport-no-config.pl
silk-3.23.1/src/rwipa/tests/rwipaimport-help.pl
silk-3.23.1/src/rwipa/tests/rwipaexport-lone-command.pl
silk-3.23.1/src/rwipa/tests/rwipaexport-help.pl
silk-3.23.1/src/rwipa/Makefile.am
silk-3.23.1/src/rwipa/rwipaimport.pod
silk-3.23.1/src/rwipa/rwipa.h
silk-3.23.1/src/rwipa/Makefile.in
silk-3.23.1/src/rwipa/ipafilter.c
silk-3.23.1/src/rwipa/rwipaimport.c
silk-3.23.1/src/rwipa/rwipaexport.pod
silk-3.23.1/src/rwipa/ipafilter.pod
ubuntu@ip-172-31-94-216:~$ ./configure --prefix=/usr/local --enable-ipv6 --with-libfixbuf=/usr/local
-bash: ./configure: No such file or directory
ubuntu@ip-172-31-94-216:~$ cd silk-3.23.1/
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ ./configure --prefix=/usr/local --enable-ipv6 --with-libfixbuf=/us
r/local
```

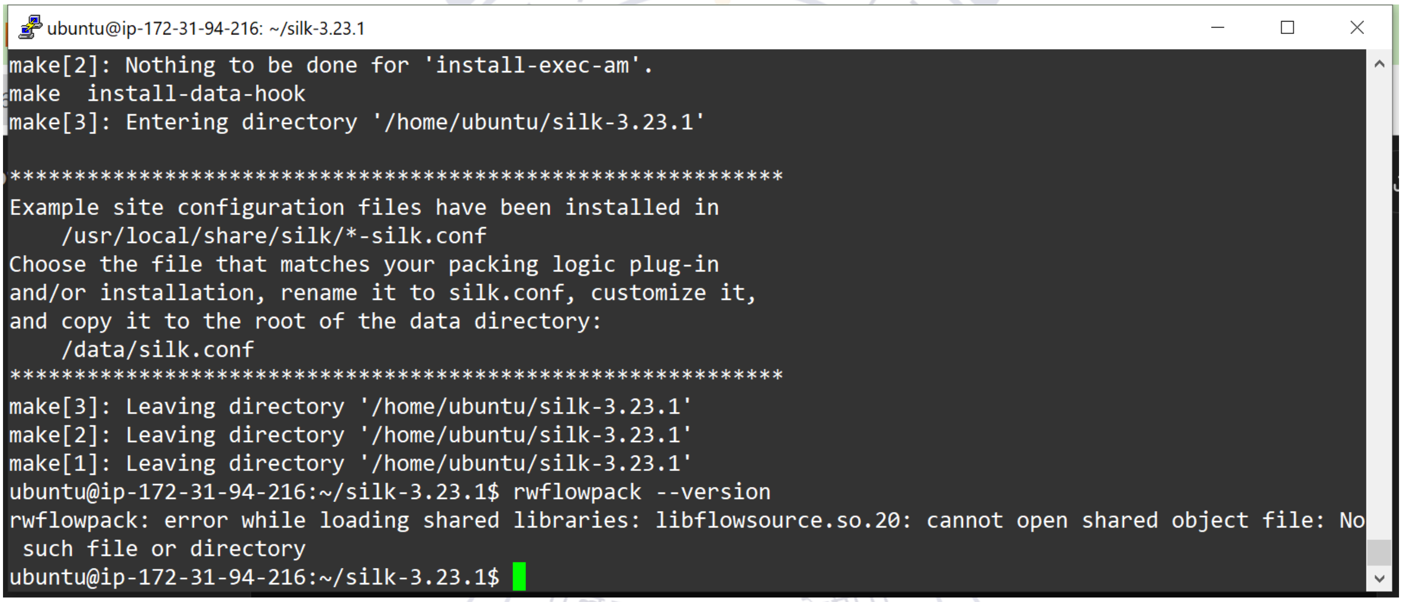
Step – 12 Builing Suilk Suite using make command.

```
ubuntu@ip-172-31-94-216: ~/silk-3.23.1
    * MaxMindDB support:            YES (-lmaxminddb)
    * ZLIB support:                 YES (-lz)
    * LZO support:                  YES (-llzo2)
    * SNAPPY support:               NO
    * LIBPCAP support:              YES (-lpcap)
    * C-ARES support:               NO
    * ADNS support:                 NO
    * Python interpreter:           /usr/bin/python3
    * Python support:               NO
    * Build analysis tools:         YES
    * Build packing tools:          YES
    * Compiler (CC):                gcc
    * Compiler flags (CFLAGS):      -I$(srcdir) -I$(top_builddir)/src/include -I$(top_srcdir)/src/include
 -DNDEBUG -D_ALL_SOURCE=1 -D_GNU_SOURCE=1 -Wall -Wextra -Wmissing-prototypes -Wformat=2 -Wdeclaration-aft
er-statement -Wpointer-arith -fno-strict-aliasing -O3
    * Linker flags (LDFLAGS):
    * Libraries (LIBS):             -llzo2 -lz -lm

ubuntu@ip-172-31-94-216:~/silk-3.23.1$ make
```

Step – 13 Installing Silk Suite using make install command



Step – 14 Checking if rwflowpack is correctly installed or not.



Step – 15 Exporting binary files to PATH environment variable for running the commands anywhere from the terminal.

```
ubuntu@ip-172-31-94-216: ~/silk-3.23.1
rwsort
rwsplit
rwstats
rwswapbytes
rwtotal
rwtuc
rwuniq
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ rwflowpack --version
rwflowpack: error while loading shared libraries: libflowsource.so.20: cannot open shared object file: No
 such file or directory
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ export PATH=$PATH:/usr/local/bin
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ source ~/.bashrc
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ export PATH=$PATH:/usr/local/bin >> ~/.bashrc
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ ls /usr/local/lib | grep libflowsource
libflowsource.la
libflowsource.so
libflowsource.so.20
libflowsource.so.20.0.0
ubuntu@ip-172-31-94-216:~/silk-3.23.1$
```

Step – 16 Checking where the file libflowsource files are located and thus, setting environment variable as such.

```
ubuntu@ip-172-31-94-216: ~/silk-3.23.1
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ ls /usr/local/lib | grep libflowsource
libflowsource.la
libflowsource.so
libflowsource.so.20
libflowsource.so.20.0.0
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ sudo nano /etc/ld.so.conf.d/silk.conf
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ sudo ldconfig
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ ldconfig -p | grep libflowsource
        libflowsource.so.20 (libc6,x86-64) => /usr/local/lib/libflowsource.so.20
        libflowsource.so (libc6,x86-64) => /usr/local/lib/libflowsource.so
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ rwflowpack --version
rwflowpack: part of SiLK 3.23.1; configuration settings:
    * Root of packed data tree:         /data
    * Packing logic:                    Run-time plug-in
    * Timezone support:                 UTC
    * Available compression methods:    none [default], zlib, lzo1x
    * IPv6 network connections:         yes
    * IPv6 flow record support:         yes
    * IPset record compatibility:       1.0.0
    * IPFIX/NetFlow9/sFlow collection:  ipfix,netflow9,sflow
    * Transport encryption:             GnuTLS
    * PySiLK support:                   no
    * Enable assert():                  no
Copyright (C) 2001-2024 by Carnegie Mellon University
GNU General Public License (GPL) Rights pursuant to Version 2, June 1991.
```

Step – 17 Checking the ouput of rwflowpack as well as setting up the environment variable also setting up the SILK_CONFIG_FILE for silk.conf file.

```
ubuntu@ip-172-31-94-216: ~/silk-3.23.1
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ rwflowpack --version
rwflowpack: part of SiLK 3.23.1; configuration settings:
    * Root of packed data tree:         /data
    * Packing logic:                    Run-time plug-in
    * Timezone support:                 UTC
    * Available compression methods:    none [default], zlib, lzo1x
    * IPv6 network connections:         yes
    * IPv6 flow record support:         yes
    * IPset record compatibility:       1.0.0
    * IPFIX/NetFlow9/sFlow collection:  ipfix,netflow9,sflow
    * Transport encryption:             GnuTLS
    * PySiLK support:                   no
    * Enable assert():                  no
Copyright (C) 2001-2024 by Carnegie Mellon University
GNU General Public License (GPL) Rights pursuant to Version 2, June 1991.
Some included library code covered by LGPL 2.1; see source for details.
Government Purpose License Rights (GPLR) pursuant to DFARS 252.227-7013.
Send bug reports, feature requests, and comments to netsa-help@cert.org.
ubuntu@ip-172-31-94-216:~/silk-3.23.1$ export SILK_CONFIG_FILE=/usr/local/etc/silk.conf >> ~/.bashrc
ubuntu@ip-172-31-94-216:~/silk-3.23.1$
```

Step – 18 Untar yaf file to home location of Ubuntu.

```
ubuntu@ip-172-31-94-216: ~
ubuntu@ip-172-31-94-216:~$ tar -xvf yaf-2.16.1.tar.gz
```

Step – 19 Compiling the yaf package using configure command.

```
ubuntu@ip-172-31-94-216: ~/yaf-2.16.1
ubuntu@ip-172-31-94-216:~$ cd yaf-2.16.1/
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ ./configure                 \
    --prefix=/usr/local      \
    --enable-silent-rules    \
    --enable-applabel        \
    --enable-metadata        \
    --enable-plugins
```

Step – 20 Building up the yaf package using make command.

```
ubuntu@ip-172-31-94-216: ~/yaf-2.16.1
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ make
```

Step – 21 Instaling yaf package using make install command.

```
ubuntu@ip-172-31-94-216: ~/yaf-2.16.1
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ sudo make install
```

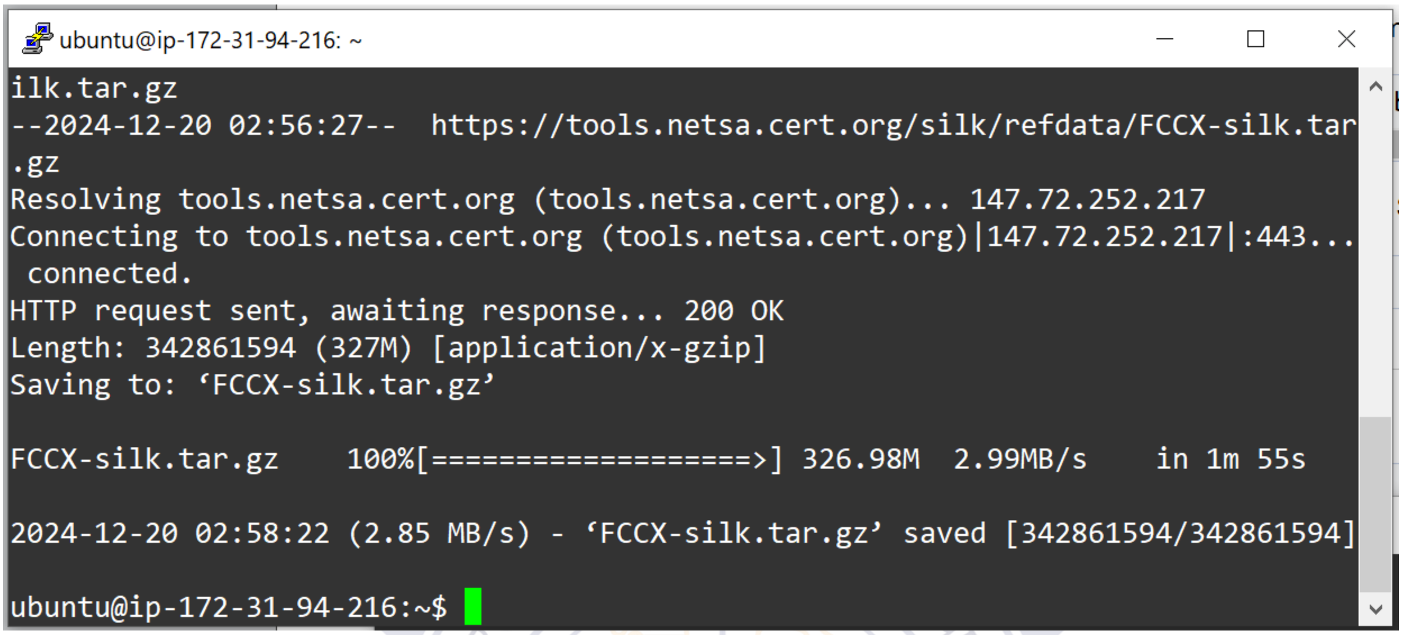
Step – 22 Setting up yaf package to autostart init.d /etc.init.d/ folder.

```
ubuntu@ip-172-31-94-216: ~/yaf-2.16.1
make[2]: Leaving directory '/home/ubuntu/yaf-2.16.1/scripts'
make[1]: Leaving directory '/home/ubuntu/yaf-2.16.1/scripts'
Making install in doc
make[1]: Entering directory '/home/ubuntu/yaf-2.16.1/doc'
make[2]: Entering directory '/home/ubuntu/yaf-2.16.1/doc'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/ubuntu/yaf-2.16.1/doc'
make[1]: Leaving directory '/home/ubuntu/yaf-2.16.1/doc'
make[1]: Entering directory '/home/ubuntu/yaf-2.16.1'
make[2]: Entering directory '/home/ubuntu/yaf-2.16.1'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/lib/pkgconfig'
 /usr/bin/install -c -m 644 libyaf.pc '/usr/local/lib/pkgconfig'
make[2]: Leaving directory '/home/ubuntu/yaf-2.16.1'
make[1]: Leaving directory '/home/ubuntu/yaf-2.16.1'
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ cp -v /tmp/yaf-2.12.2/etc/init.d/yaf /etc/init.d/yaf
cp: cannot stat '/tmp/yaf-2.12.2/etc/init.d/yaf': No such file or directory
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ cp -v etc/init.d/yaf /etc/init.d/yaf
'etc/init.d/yaf' -> '/etc/init.d/yaf'
cp: cannot create regular file '/etc/init.d/yaf': Permission denied
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ sudo cp -v etc/init.d/yaf /etc/init.d/yaf
'etc/init.d/yaf' -> '/etc/init.d/yaf'
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ sudo chmod a+x /etc/init.d/yaf
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$
```

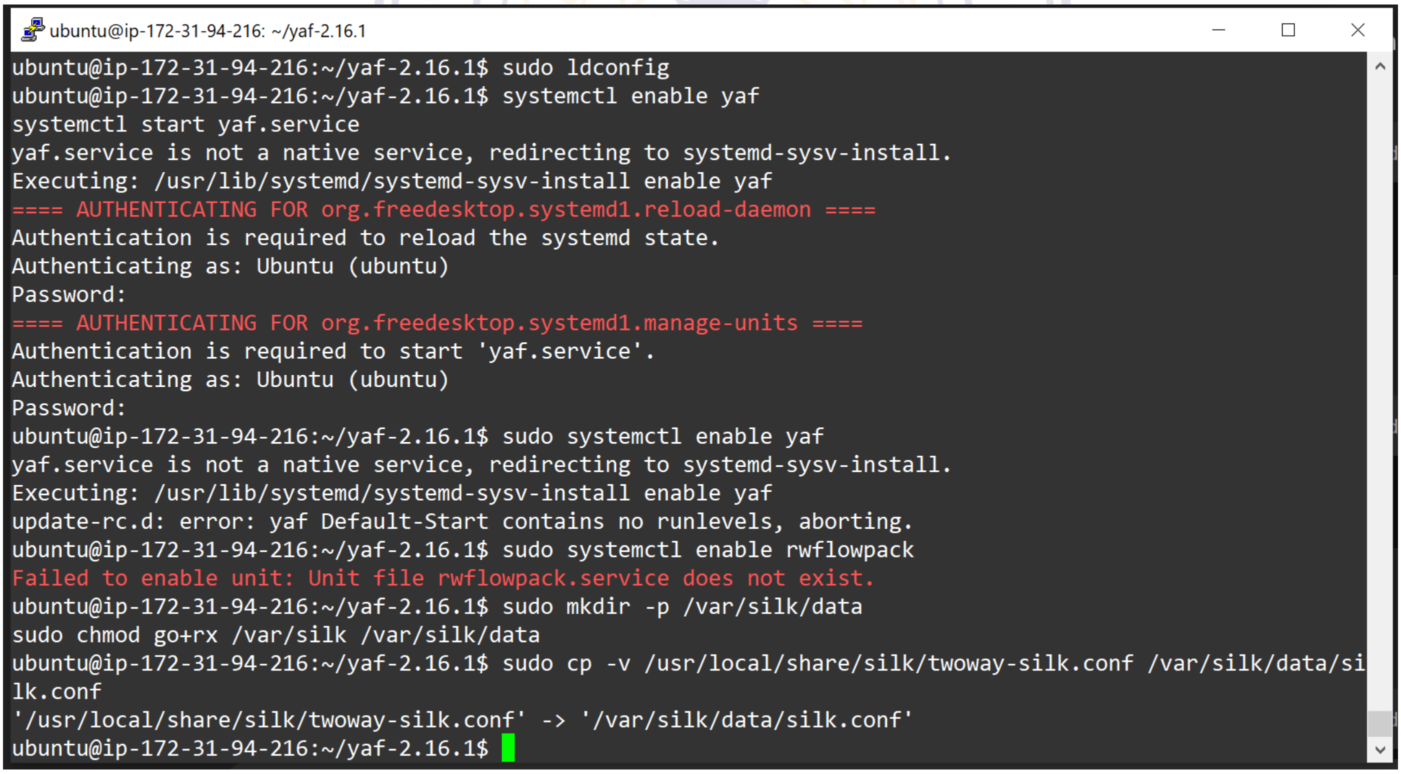
Step – 23 Adding environment Variable such as LD_LIBRARY_PATH also reconfiguring ldconfig command using ldconfig command.

```
ubuntu@ip-172-31-94-216: ~/yaf-2.16.1
/etc/ld.so.conf.d/silk.conf:/usr/local/lib
/etc/ld.so.conf.d/x86_64-linux-gnu.conf:/usr/local/lib/x86_64-linux-gnu
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ cat /usr/local/lib
cat: /usr/local/lib: Is a directory
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ ls /usr/local/lib
libairframe-2.16.1.so.4      libfixbuf.so.9.1.0       libsilk-thrd.so.5.1.7   libyaf.so
libairframe-2.16.1.so.4.0.0  libflowsource.la         libsilk.la              pkgconfig
libairframe.la               libflowsource.so         libsilk.so              python3.12
libairframe.so               libflowsource.so.20      libsilk.so.30           silk
libfixbuf.a                  libflowsource.so.20.0.0  libsilk.so.30.0.1       yaf
libfixbuf.la                 libsilk-thrd.la          libyaf-2.16.1.so.4
libfixbuf.so                 libsilk-thrd.so          libyaf-2.16.1.so.4.0.0
libfixbuf.so.9               libsilk-thrd.so.5        libyaf.la
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ ls /usr/local/lib/silk
app-mismatch.la  cutmatch.la  flowrate.la        packlogic-generic.la
app-mismatch.so  cutmatch.so  flowrate.so        packlogic-generic.so
conficker-c.la   flowkey.la   int-ext-fields.la  packlogic-twoway.la
conficker-c.so   flowkey.so   int-ext-fields.so  packlogic-twoway.so
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH >> ~/.bashrc
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ source ~/.bashrc
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ ldconfig
/sbin/ldconfig.real: Can't create temporary cache file /etc/ld.so.cache~: Permission denied
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$ sudo ldconfig
ubuntu@ip-172-31-94-216:~/yaf-2.16.1$
```

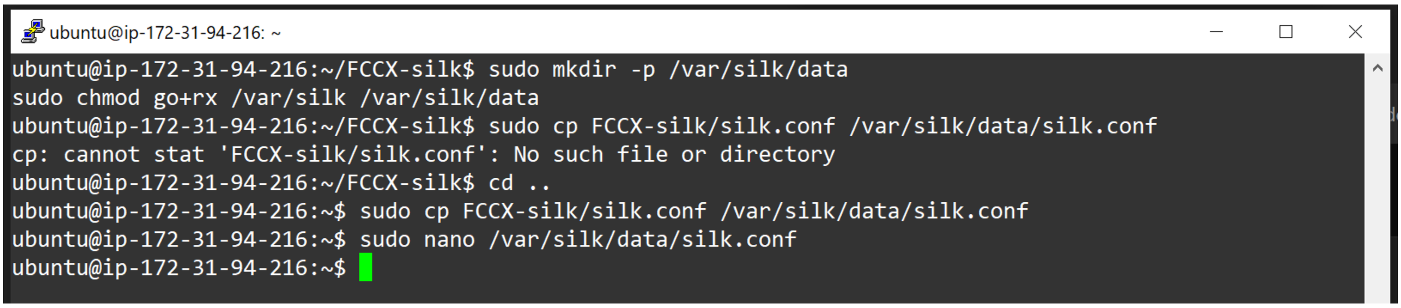
Step-24 Unizipping FCCX-silk.tar.gz using unzip command



Step – 25 Enabling yaf to autostart also, copying twoway-silk.conf to its desired location.

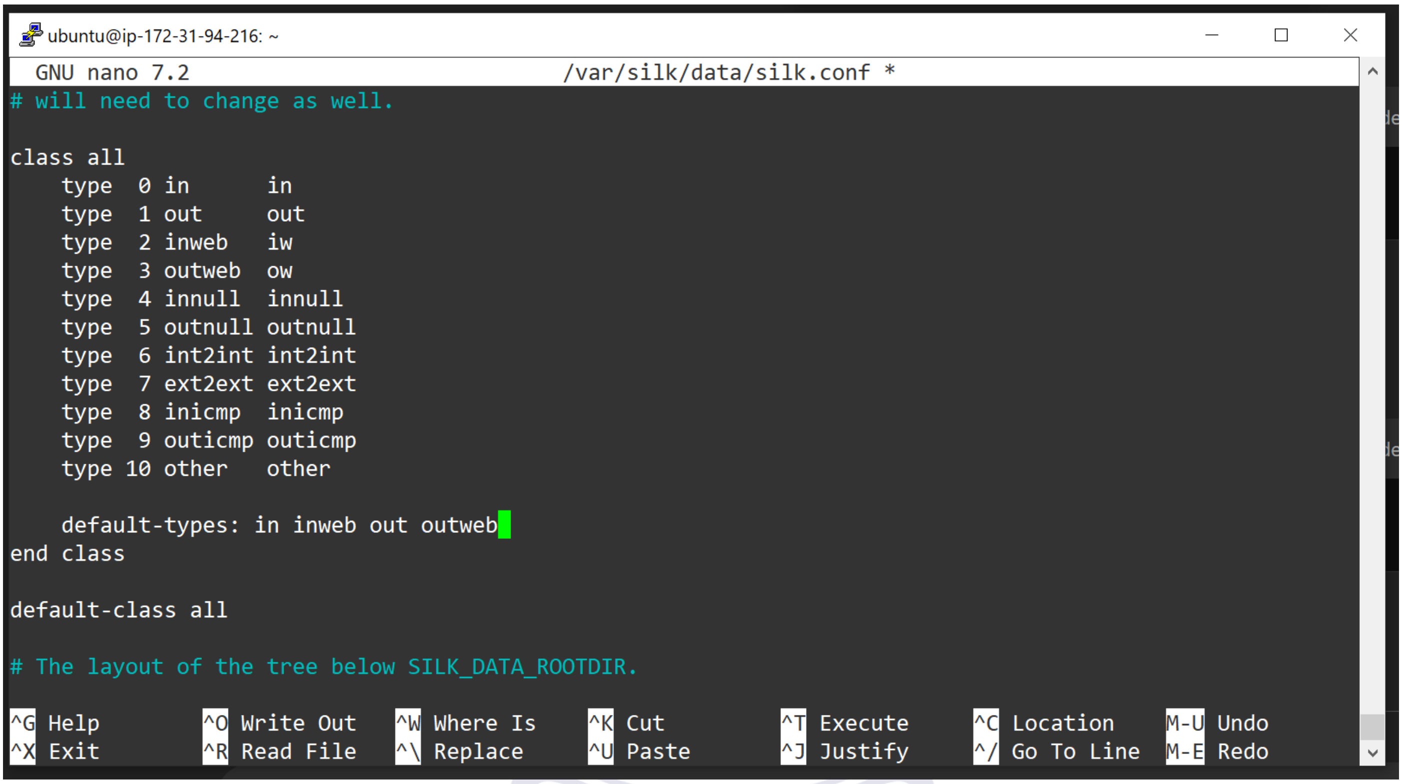


Step – 26 Editing certain parts of silk.conf file.



```
ubuntu@ip-172-31-94-216: ~
ubuntu@ip-172-31-94-216:~/FCCX-silk$ sudo mkdir -p /var/silk/data
sudo chmod go+rx /var/silk /var/silk/data
ubuntu@ip-172-31-94-216:~/FCCX-silk$ sudo cp FCCX-silk/silk.conf /var/silk/data/silk.conf
cp: cannot stat 'FCCX-silk/silk.conf': No such file or directory
ubuntu@ip-172-31-94-216:~/FCCX-silk$ cd ..
ubuntu@ip-172-31-94-216:~$ sudo cp FCCX-silk/silk.conf /var/silk/data/silk.conf
ubuntu@ip-172-31-94-216:~$ sudo nano /var/silk/data/silk.conf
ubuntu@ip-172-31-94-216:~$
```

Step – 27 After Changing silk.conf file it should look something like this.



```
ubuntu@ip-172-31-94-216: ~
  GNU nano 7.2                    /var/silk/data/silk.conf *
# will need to change as well.

class all
    type  0 in      in
    type  1 out     out
    type  2 inweb   iw
    type  3 outweb  ow
    type  4 innull  innull
    type  5 outnull outnull
    type  6 int2int int2int
    type  7 ext2ext ext2ext
    type  8 inicmp  inicmp
    type  9 outicmp outicmp
    type 10 other   other

    default-types: in inweb out outweb
end class

default-class all

# The layout of the tree below SILK_DATA_ROOTDIR.

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo
```

Step – 28 Editing sensors.conf file and after editing it it was looking like this, as ipblocks IP Address was given from my AWS EC2 instance.

```
ubuntu@ip-172-31-94-216: ~
ubuntu@ip-172-31-94-216:~/FCCX-silk$ sudo mkdir -p /var/silk/data
sudo chmod go+rx /var/silk /var/silk/data
ubuntu@ip-172-31-94-216:~/FCCX-silk$ sudo cp FCCX-silk/silk.conf /var/silk/data/silk.conf
cp: cannot stat 'FCCX-silk/silk.conf': No such file or directory
ubuntu@ip-172-31-94-216:~/FCCX-silk$ cd ..
ubuntu@ip-172-31-94-216:~$ sudo cp FCCX-silk/silk.conf /var/silk/data/silk.conf
ubuntu@ip-172-31-94-216:~$ sudo nano /var/silk/data/silk.conf
ubuntu@ip-172-31-94-216:~$ sudo nano /var/silk/sensors.conf
ubuntu@ip-172-31-94-216:~$ cat /var/silk/sensors.conf
probe S0 ipfix
  listen-on-port 18001
  protocol tcp
  listen-as-host 127.0.0.1
end probe

group my-network
  ipblocks 172.31.94.216/32  # Internal network IP block
end group

sensor S0
  ipfix-probes S0
  internal-ipblocks @my-network
  external-ipblocks remainder
end sensor
ubuntu@ip-172-31-94-216:~$
```

Step – 29 Adding rwflowpack.service in such a manner that it should not fail the service after starting so re-configuring it.

```
ubuntu@ip-172-31-94-216: /usr/local
    LD_LIBRARY_PATH="/home/ubuntu/silk-3.23.1/src/libflowsource/.libs:/home/ubuntu/silk-3.23.1/src/libsilk/.libs:/usr/local/lib:$LD_LIBRARY_PATH"

    # Some systems cannot cope with colon-terminated LD_LIBRARY_PATH
    # The second colon is a workaround for a bug in BeOS R4 sed
    LD_LIBRARY_PATH=`$ECHO "$LD_LIBRARY_PATH" | /usr/bin/sed 's/::*$//'`

    export LD_LIBRARY_PATH

    if test "$libtool_execute_magic" != "%%%MAGIC variable%%%"; then
      # Run the actual program with our arguments.
      func_exec_program ${1+"$@"}
    fi
  else
    # The program doesn't exist.
    $ECHO "$0: error: '$progdir/$program' does not exist" 1>&2
    $ECHO "This script is just a wrapper for $program." 1>&2
    $ECHO "See the libtool documentation for more information." 1>&2
    exit 1
  fi
fi
ubuntu@ip-172-31-94-216:/usr/local$ which rwflowpack
/usr/local/sbin/rwflowpack
ubuntu@ip-172-31-94-216:/usr/local$ sudo nano /etc/systemd/system/rwflowpack.service
ubuntu@ip-172-31-94-216:/usr/local$
```

Step – 30 Here is the code for the same.

```
ubuntu@ip-172-31-94-216: /usr/local
  GNU nano 7.2                    /etc/systemd/system/rwflowpack.service *
[Unit]
Description=SiLK rwflowpack Service
After=network.target

[Service]
ExecStart=/usr/local/sbin/rwflowpack \
    --sensor-configuration=/var/silk/sensors.conf \
    --site-configuration=/var/silk/data/silk.conf \
    --output-mode=local-storage
Restart=on-failure
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=rwflowpack
User=root
Group=root

[Install]
WantedBy=multi-user.target


^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo
```

Step – 31 Here is the code for rwflowpack to start and check its status to ideal or inactive/active.

```
ubuntu@ip-172-31-94-216: /usr/local

    export LD_LIBRARY_PATH

    if test "$libtool_execute_magic" != "%%%MAGIC variable%%%"; then
      # Run the actual program with our arguments.
      func_exec_program ${1+"$@"}
    fi
  else
    # The program doesn't exist.
    $ECHO "$0: error: '$progdir/$program' does not exist" 1>&2
    $ECHO "This script is just a wrapper for $program." 1>&2
    $ECHO "See the libtool documentation for more information." 1>&2
    exit 1
  fi
fi
ubuntu@ip-172-31-94-216:/usr/local$ which rwflowpack
/usr/local/sbin/rwflowpack
ubuntu@ip-172-31-94-216:/usr/local$ sudo nano /etc/systemd/system/rwflowpack.service
ubuntu@ip-172-31-94-216:/usr/local$ sudo systemctl daemon-reload
ubuntu@ip-172-31-94-216:/usr/local$ sudo systemctl enable rwflowpack
sudo systemctl start rwflowpack
Created symlink /etc/systemd/system/multi-user.target.wants/rwflowpack.service → /etc/systemd/system/rwfl
owpack.service.
ubuntu@ip-172-31-94-216:/usr/local$ sudo systemctl start rwflowpack
ubuntu@ip-172-31-94-216:/usr/local$
```

Step –32 Setting up the certain variables such as SILK_CONFIG_FILE and SILK_DATA_ROOTDIR and do the needful.

```
ubuntu@ip-172-31-94-216: ~/FCCX-silk
./S9/outweb/2015/06:
02  17  18

./S9/outweb/2015/06/02:
ow-S9_20150602.13

./S9/outweb/2015/06/17:
ow-S9_20150617.14  ow-S9_20150617.16  ow-S9_20150617.18  ow-S9_20150617.20  ow-S9_20150617.22
ow-S9_20150617.15  ow-S9_20150617.17  ow-S9_20150617.19  ow-S9_20150617.21  ow-S9_20150617.23

./S9/outweb/2015/06/18:
ow-S9_20150618.00  ow-S9_20150618.04  ow-S9_20150618.08  ow-S9_20150618.12  ow-S9_20150618.16
ow-S9_20150618.01  ow-S9_20150618.05  ow-S9_20150618.09  ow-S9_20150618.13  ow-S9_20150618.17
ow-S9_20150618.02  ow-S9_20150618.06  ow-S9_20150618.10  ow-S9_20150618.14
ow-S9_20150618.03  ow-S9_20150618.07  ow-S9_20150618.11  ow-S9_20150618.15
ubuntu@ip-172-31-94-216:~/FCCX-silk$ ls -R . >abc.txt
ubuntu@ip-172-31-94-216:~/FCCX-silk$ pwd
/home/ubuntu/FCCX-silk
ubuntu@ip-172-31-94-216:~/FCCX-silk$ export SILK_CONFIG_FILE=/home/ubuntu/FCCX-silk/silk.conf
export SILK_DATA_ROOTDIR=/home/ubuntu/FCCX-silk
ubuntu@ip-172-31-94-216:~/FCCX-silk$ echo $SILK_CONFIG_FILE
echo $SILK_DATA_ROOTDIR
/home/ubuntu/FCCX-silk/silk.conf
/home/ubuntu/FCCX-silk
ubuntu@ip-172-31-94-216:~/FCCX-silk$
```

Step - 33 Fetching all the records using rwfilter and then store it in filtered.rw. After which, running rwstats command to look at first 5 data from the dataset of FCCX-silk

(Note: Here I have taken the date from 2000 to 2039 for experimental purpose but actual date lies for 2015 dataset only, but I forgot to take that screenshot, so pasted this screenshot. Also there is no change in the output as said)

```
ubuntu@ip-172-31-94-216: /data
ubuntu@ip-172-31-94-216:/data$ rwfilter \
--start-date=2000/01/01 \
--end-date=2039/01/01 \
--type=all \
--site-config-file=/data/silk.conf \
--all-destination=filtered.rw
ubuntu@ip-172-31-94-216:/data$ rwstats filtered.rw --fields=protocol --count=5

INPUT: 39210497 Records for 7 Bins and 39210497 Total Records
OUTPUT: Top 5 Bins by Records
pro|    Records|   %Records|    cumul_%|
 17|   24630960|  62.817260|  62.817260|
  6|   14137746|  36.056023|  98.873284|
  1|     436648|   1.113600|  99.986884|
 89|       2756|   0.007029|  99.993912|
 47|       2332|   0.005947|  99.999860|
ubuntu@ip-172-31-94-216:/data$
ubuntu@ip-172-31-94-216:/data$
```

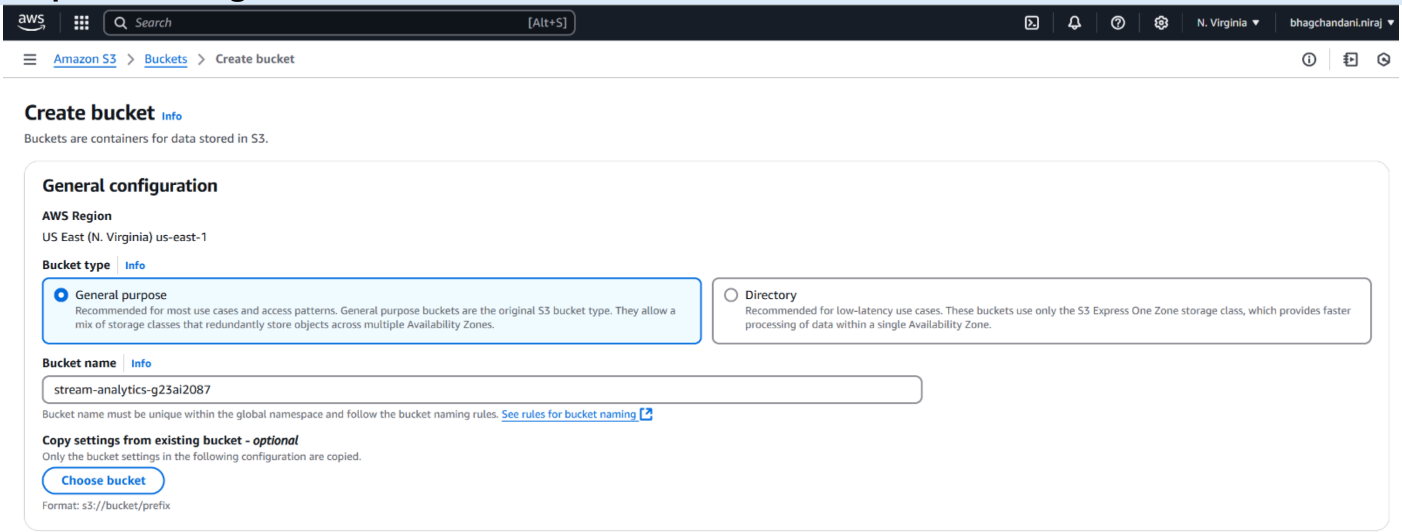
Step - 34 Converting all the data fetched to csv file and then printing the first 5 records using cat command.

```
ubuntu@ip-172-31-94-216:/data$ rwcut sample.rw --output=all_d.csv
ubuntu@ip-172-31-94-216:/data$ cat all_d.csv | head -5
                                    sIP|                                    dIP|sPort|dPort|pro|     packets|
  bytes|   flags|                    sTime|    duration|                    eTime|sen|
                        192.168.121.145|                             10.0.40.53|59624| 5723|  6|           7|
   2200|FS PA   |2015/06/02T13:39:31.779000|    0.012000|2015/06/02T13:39:31.791000| S5|
                         192.168.121.77|                             10.0.40.53|49206| 5723|  6|           7|
   2200|FS PA   |2015/06/02T13:39:44.244000|    0.014000|2015/06/02T13:39:44.258000| S5|
                         192.168.121.57|                             10.0.40.53|60222| 5723|  6|           8|
   2240|FS PA   |2015/06/02T13:39:44.369000|    0.011000|2015/06/02T13:39:44.380000| S5|
                        192.168.121.244|                             10.0.40.53|56312| 5723|  6|           6|
   2160|FS PA   |2015/06/02T13:39:51.966000|    0.015000|2015/06/02T13:39:51.981000| S5|
ubuntu@ip-172-31-94-216:/data$
```

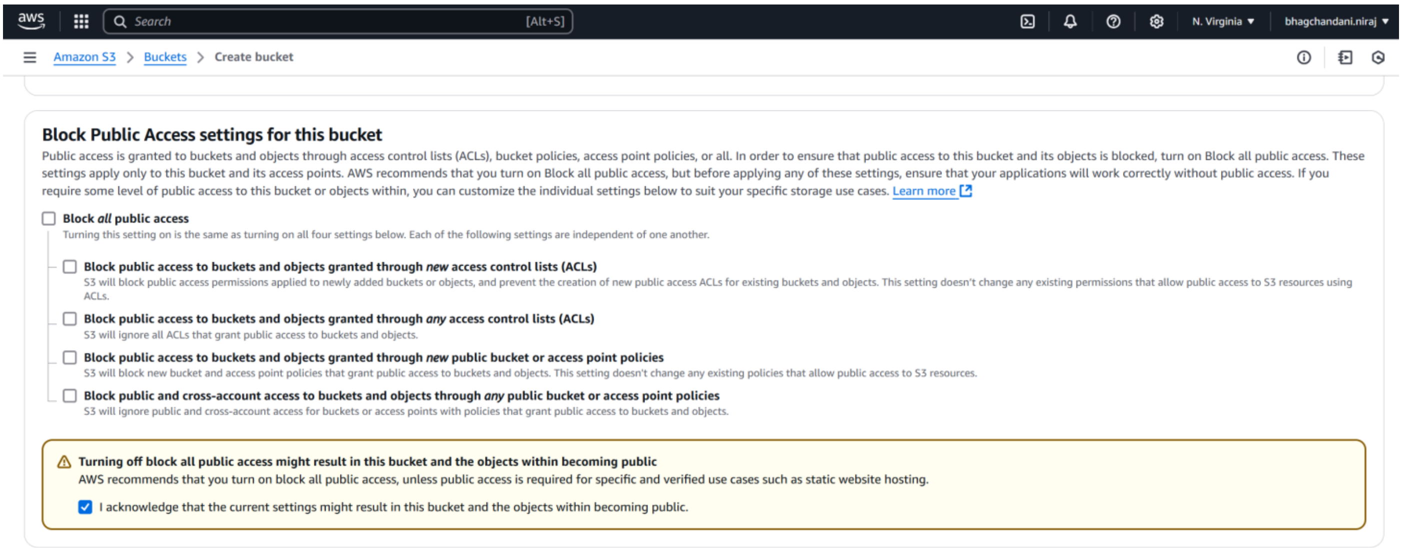
2. Classify nodes based on the amount of TCP traffic they process. For instance, you can group nodes that handle TCP traffic within a specific range, such as 10-100 packets per second, into one class. Use any two algorithms, say Very Fast Decision Trees (VFDT) and On Demand Classification, discussed in the lectures to perform this classification. During the demonstration, you should demonstrate the accuracy of your classification by sending a query that retrieves data from at least two different ranges.

Configuring Amazon AWS – S3 for Storing CSV File

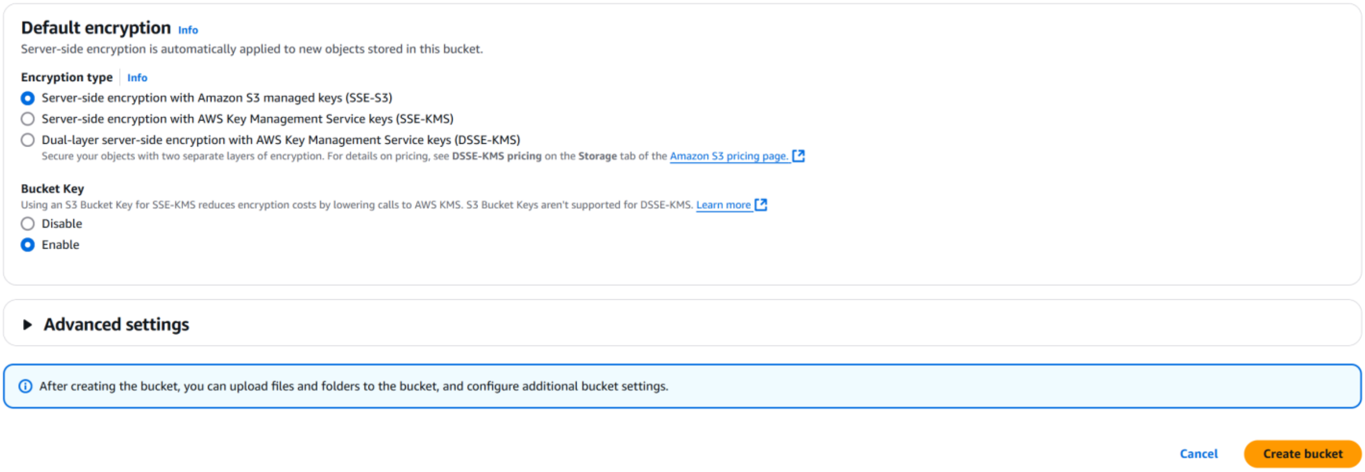
Step – 1 Creating Bucket in Amazon S3



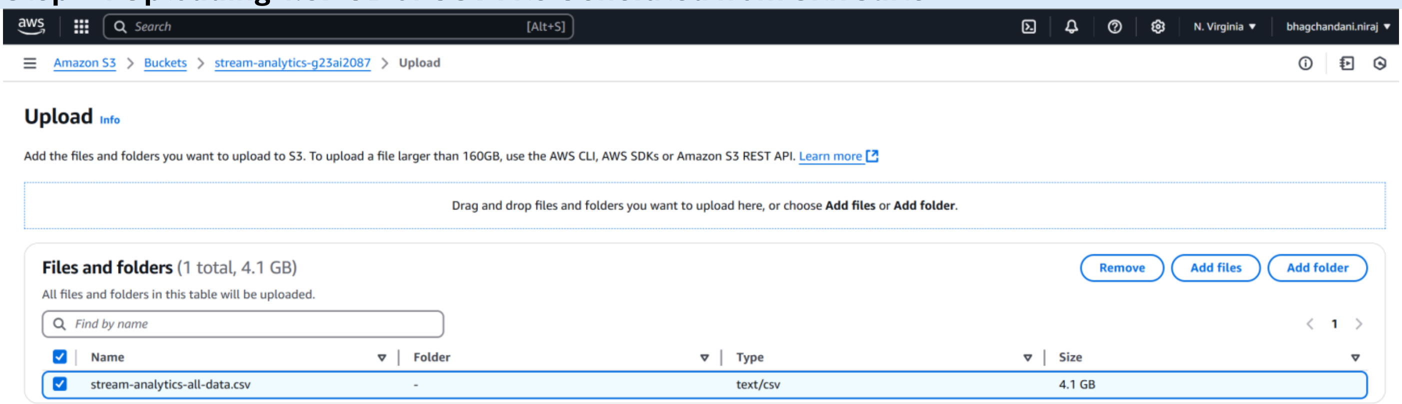
Step – 2 Allowing Public Access for this bucket



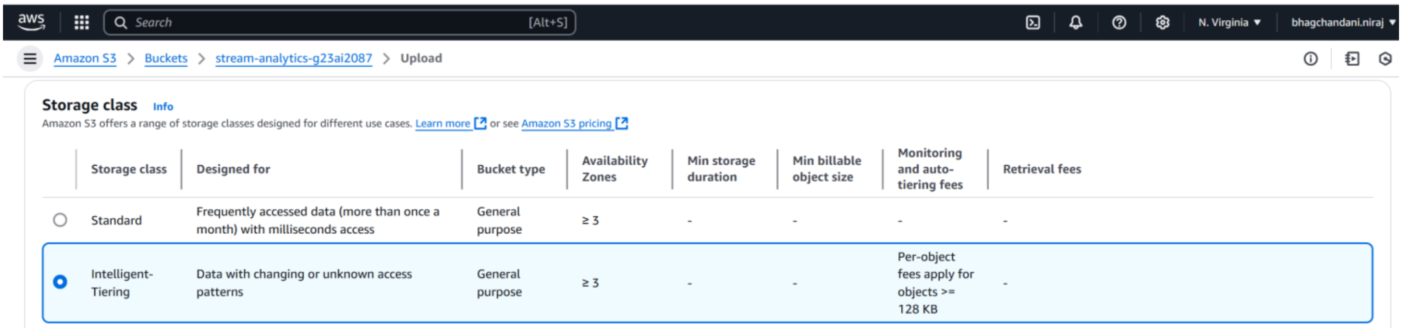
Step – 3 Creating Bucket with Following Encryption Settings.



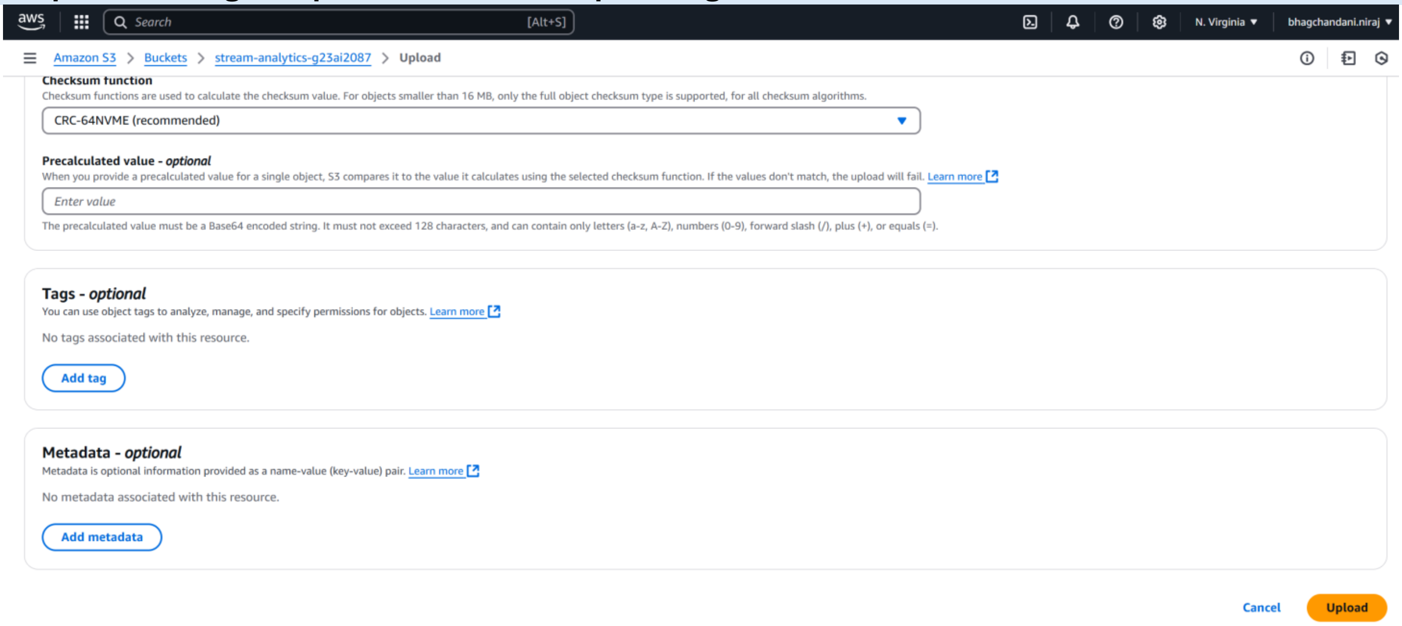
Step – 4 Uploading 4.37 GB of CSV File Generated from Silk Suite



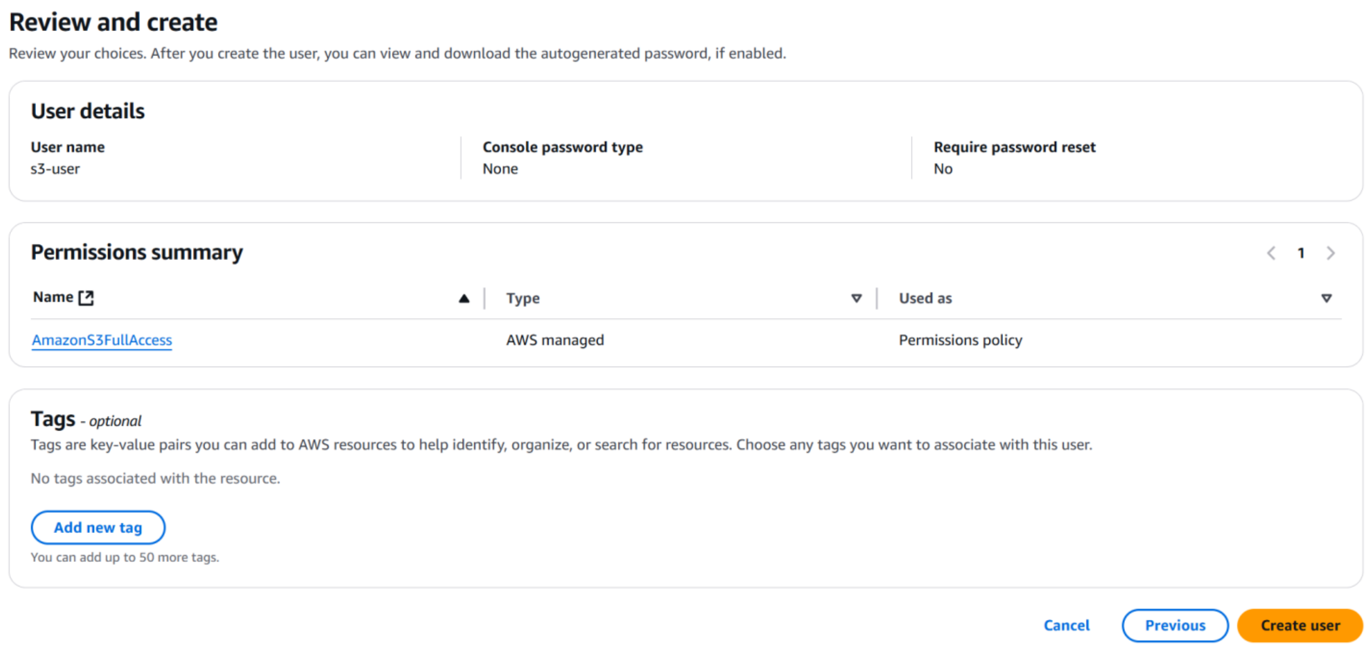
Step – 5 Selecting Storage Class as Intelligent Tiering



Step – 6 Clicking on Updload Button for Uploading the CSV File from Local to S3 Bucket.



Step – 7 Creating Amazon S3 Full Access Account for getting AWS Key and , AWS Secret ID and other details.



Step 1: Install Required Libraries

In this step, we install the necessary libraries for data processing, model training, and visualization. These include:

- numpy: A library for numerical computations.
- river: For implementing Very Fast Decision Trees (VFDT) and online machine learning.
- boto3: For accessing AWS S3 to load data.
- tqdm: For displaying progress bars during iterative processes.

```
!pip install numpy
!pip install river
!pip install boto3
!pip install tqdm
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: boto3 in c:\users\dell\appdata\roaming\python\python312\site-packages (1.35.86)
Requirement already satisfied: botocore<1.36.0,>=1.35.86 in c:\users\dell\appdata\roaming\python\python312\site-packages (from boto3) (1.35.86)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in c:\users\dell\appdata\roaming\python\python312\site-packages (from boto3) (1.0.1)
Requirement already satisfied: s3transfer<0.11.0,>=0.10.0 in c:\users\dell\appdata\roaming\python\python312\site-packages (from boto3) (0.10.4)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in c:\program files\python312\lib\site-packages (from botocore<1.36.0,>=1.35.86->boto3) (2.
9.0.post0)
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in c:\program files\python312\lib\site-packages (from botocore<1.36.0,>=1.35.86->boto3) (2.
2.2)
Requirement already satisfied: six>=1.5 in c:\program files\python312\lib\site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.36.0,>=1.35.86->
boto3) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Collecting tqdm
  Downloading tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Requirement already satisfied: colorama in c:\program files\python312\lib\site-packages (from tqdm) (0.4.6)
Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
Installing collected packages: tqdm
Successfully installed tqdm-4.67.1
```

Step 2: Import Libraries and Initialize Global Variables

In this step, we import the necessary libraries for handling data, accessing AWS S3, and managing progress bars during file downloads.

- os: For handling file paths.
- pandas: For loading and processing CSV files into dataframes.
- boto3: For accessing and downloading files from AWS S3.
- StringIO: For handling CSV content in-memory.
- tqdm: For displaying progress bars during file downloads.

We also declare data as a global variable to store the loaded dataset for further processing.

```python
import os
import pandas as pd
import boto3
from io import StringIO
from tqdm import tqdm  # For progress bar

# Declare `data` as a global variable
data = None
```

Step 3: Define Functions to Load Data

We define two functions to load the dataset:

1. **load_local_file()**:
    - Loads a CSV file from the local system using pandas.read_csv.
    - Displays a preview of the data.

2. **load_s3_file_direct()**:
    - Downloads a CSV file from AWS S3 using the boto3 library.
    - Displays a progress bar while downloading the file.
    - Loads the downloaded CSV into a Pandas dataframe.

Key Points:

- The AWS S3 credentials (aws_access_key_id and aws_secret_access_key) and bucket details are hardcoded in this example.
- The function uses a tqdm progress bar to provide feedback during the download.

```
def load_local_file():
    global data  # Declare as global to make it accessible outside the function
    csv_file_path = r"C:\Users\dell\Downloads\stream-analytics-all-data.csv"
    if os.path.exists(csv_file_path):
        try:
            data = pd.read_csv(csv_file_path, delimiter='|')
            print("Data loaded successfully from local file!")
            print(data.head())
        except Exception as e:
            print(f"Error reading the local file: {e}")
    else:
        print("File does not exist. Please check the path and try again.")

def load_s3_file_direct():
    global data  # Declare as global to make it accessible outside the function
    aws_access_key_id = r"<aws-key-id>"
    aws_secret_access_key = r"<aws-secret-access-key>"
    bucket_name = "stream-analytics-g23ai2087"
    file_key = "stream-analytics-all-data.csv"

    try:
        # Create S3 client
        s3 = boto3.client(
            's3',
            aws_access_key_id=aws_access_key_id,
            aws_secret_access_key=aws_secret_access_key,
        )

        # Get the object metadata to determine the file size
        obj_metadata = s3.head_object(Bucket=bucket_name, Key=file_key)
        file_size = obj_metadata['ContentLength']

        # Download the object with a progress bar
        response = s3.get_object(Bucket=bucket_name, Key=file_key)

        # Use a progress bar to monitor the download
        chunk_size = 10 * 1024 * 1024  # 10 MB chunks
        with tqdm(total=file_size, unit='B', unit_scale=True, desc="Downloading") as pbar:
            csv_content = ""
```

```
            for chunk in response['Body'].iter_chunks(chunk_size):
                csv_content += chunk.decode('utf-8')
                pbar.update(len(chunk))

        # Load the CSV content directly into pandas DataFrame
        data = pd.read_csv(StringIO(csv_content), delimiter='|')
        print("\nData loaded successfully from S3!")
        print(data.head())
    except Exception as e:
        print(f"Error loading file from S3: {e}")
```

Step 4: Create a Menu for Data Loading

We define a menu() function to allow the user to select how they want to load the dataset:

1. **Load from Local File**: Select option 1 to load a local CSV file.
2. **Load from S3**: Select option 2 to load the CSV file directly from AWS S3.

Key Points:

- The menu() function interacts with the user via the command line to get the choice.
- The load_local_file() or load_s3_file_direct() function is called based on the user's input.

```
def menu():
    print("\nMenu:")
    print("1. Load from local CSV file")
    print("2. Load directly from AWS S3")

    choice = input("Enter your choice (1/2): ").strip()
    if choice == '1':
        load_local_file()
    elif choice == '2':
        load_s3_file_direct()
    else:
        print("Invalid choice. Please enter 1, 2, or 3.")
```

Step 5: Run the Menu

Finally, we run the menu() function to allow the user to choose the data loading method. After loading the dataset, the global data variable will contain the loaded DataFrame, which can be used for further analysis and processing.

```
# Run the menu
menu()
```

Output:

```
Menu:
1. Load from local CSV file
2. Load directly from AWS S3
Enter your choice (1/2):  1
Data loaded successfully from local file!
                                     sIP  \
0                          192.168.121.145
1                           192.168.121.77
```

```
2                           192.168.121.57
3                          192.168.121.244
4                           192.168.121.57

                                      dIP  sPort  dPort  pro       packets  \
0                             10.0.40.53  59624   5723    6             7
1                             10.0.40.53  49206   5723    6             7
2                             10.0.40.53  60222   5723    6             8
3                             10.0.40.53  56312   5723    6             6
4                             10.0.40.20  60223    389    6            11

             bytes     flags                      sTime      duration  \
0             2200   FS PA     2015/06/02T13:39:31.779000         0.012
1             2200   FS PA     2015/06/02T13:39:44.244000         0.014
2             2240   FS PA     2015/06/02T13:39:44.369000         0.011
3             2160   FS PA     2015/06/02T13:39:51.966000         0.015
4             2877   FS PA     2015/06/02T13:39:54.712000         0.047

                        eTime  sen  Unnamed: 12
0  2015/06/02T13:39:31.791000   S5          NaN
1  2015/06/02T13:39:44.258000   S5          NaN
2  2015/06/02T13:39:44.380000   S5          NaN
3  2015/06/02T13:39:51.981000   S5          NaN
4  2015/06/02T13:39:54.759000   S5          NaN
```

Step 6: Preview the Loaded Data

In this step, we display the first few rows of the loaded dataset to ensure it has been successfully loaded and is in the correct format.

Key Points:

- The data variable contains the dataset loaded using the menu options (either from a local file or AWS S3).
- The head() function from Pandas is used to display the first few rows of the DataFrame.

```
# Display the first few rows to ensure it's loaded correctly
data.head()
```

Output:

[23]:

	sIP	dIP	sPort	dPort	pro	packets	bytes	flags	sTime	duration	eTime	sen	Unnamed: 12
0	192.168.121.145	10.0.40.53	59624	5723	6	7	2200	FS PA	2015/06/02T13:39:31.779000	0.012	2015/06/02T13:39:31.791000	S5	NaN
1	192.168.121.77	10.0.40.53	49206	5723	6	7	2200	FS PA	2015/06/02T13:39:44.244000	0.014	2015/06/02T13:39:44.258000	S5	NaN
2	192.168.121.57	10.0.40.53	60222	5723	6	8	2240	FS PA	2015/06/02T13:39:44.369000	0.011	2015/06/02T13:39:44.380000	S5	NaN
3	192.168.121.244	10.0.40.53	56312	5723	6	6	2160	FS PA	2015/06/02T13:39:51.966000	0.015	2015/06/02T13:39:51.981000	S5	NaN
4	192.168.121.57	10.0.40.20	60223	389	6	11	2877	FS PA	2015/06/02T13:39:54.712000	0.047	2015/06/02T13:39:54.759000	S5	NaN

Step 7: Explore the Dataset Headers

In this step, we count and list all the headers (columns) in the loaded dataset to understand its structure and ensure the expected fields are present.

Key Points:

- **Count Headers**: The len() function is used to count the total number of columns in the dataset.
- **List Headers**: The columns attribute of the Pandas DataFrame provides the list of all column names.

```
# Count the number of headers
num_headers = len(data.columns)
print(f"Number of headers (columns): {num_headers}")

# List all headers
print("Headers in the CSV:")
print(data.columns.tolist())
```

Output:

```
Number of headers (columns): 13
Headers in the CSV:
['                                    sIP', '                                        dIP', 'sPort',
'dPort', 'pro', '     packets', '          bytes', '   flags', '                                 sTime', '
duration', '                       eTime', 'sen', 'Unnamed: 12']
```

Step 8: Dataset Overview and Summary Statistics

In this step, we examine the overall structure of the dataset and compute summary statistics for the numeric columns.

Key Points:

- **data.info()**:
  - Displays a concise summary of the dataset, including:
    - Total number of rows and columns.
    - Data types of each column.
    - Number of non-null values for each column.
- **data.describe()**:
  - Provides summary statistics (e.g., count, mean, standard deviation, min, max) for all numeric columns in the dataset.
  - Helps in understanding the distribution and range of numeric variables.

```
# Display data information
data.info()

# Get a summary of numeric columns
data.describe()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21216409 entries, 0 to 21216408
Data columns (total 13 columns):
 #   Column                                   Dtype
---  ------                                   -----
 0                                       sIP  object
 1                                       dIP  object
 2   sPort                                    int64
 3   dPort                                    int64
 4   pro                                      int64
 5        packets                             int64
 6             bytes                          int64
 7      flags                                 object
 8                        sTime               object
 9       duration                             float64
 10                       eTime               object
```

```
 11  sen                                         object
 12  Unnamed: 12                                 float64
dtypes: float64(2), int64(5), object(6)
memory usage: 2.1+ GB
```

[25]:

	sPort	dPort	pro	packets	bytes	duration	Unnamed: 12
count	2.121641e+07	2.121641e+07	2.121641e+07	2.121641e+07	2.121641e+07	2.121641e+07	0.0
mean	3.418730e+04	1.197113e+04	1.328963e+01	4.829432e+00	7.823922e+02	2.350012e+00	NaN
std	2.457024e+04	2.240215e+04	5.302419e+00	2.322078e+02	4.679487e+04	2.718544e+01	NaN
min	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	2.300000e+01	0.000000e+00	NaN
25%	4.217000e+03	5.300000e+01	6.000000e+00	1.000000e+00	7.400000e+01	0.000000e+00	NaN
50%	4.707500e+04	5.300000e+01	1.700000e+01	2.000000e+00	1.470000e+02	0.000000e+00	NaN
75%	5.490900e+04	5.723000e+03	1.700000e+01	4.000000e+00	2.120000e+02	1.000000e-02	NaN
max	6.553500e+04	6.553500e+04	8.900000e+01	4.057340e+05	6.828201e+07	1.800922e+03	NaN

Step 9: Filter Data for TCP Protocol

In this step, we filter the dataset to include only rows where the protocol is TCP. This is done by checking if the pro column has a value of 6, which represents the TCP protocol.

Key Points:

- **Filtering**:
  - The data DataFrame is filtered using a condition data['pro'] == 6.
  - The filtered dataset is stored in a new DataFrame named tcp_data.
- **Copy of Filtered Data**:
  - We use the .copy() method to avoid potential warnings about modifying the original DataFrame.
- **Preview**:
  - The first few rows of the filtered dataset are displayed using the head() function.

```
# Filter rows where protocol is TCP (pro == 6)
tcp_data = data[data['pro'] == 6].copy()

# Display the filtered data
print(tcp_data.head())
```

Output:

```
                                  sIP  \
0                     192.168.121.145
1                      192.168.121.77
2                      192.168.121.57
3                     192.168.121.244
4                      192.168.121.57

                                  dIP  sPort  dPort  pro      packets  \
0                          10.0.40.53  59624   5723    6            7
1                          10.0.40.53  49206   5723    6            7
2                          10.0.40.53  60222   5723    6            8
```

```
3                              10.0.40.53  56312    5723    6             6
4                              10.0.40.20  60223     389    6            11

             bytes     flags                      sTime      duration  \
0             2200  FS PA     2015/06/02T13:39:31.779000        0.012
1             2200  FS PA     2015/06/02T13:39:44.244000        0.014
2             2240  FS PA     2015/06/02T13:39:44.369000        0.011
3             2160  FS PA     2015/06/02T13:39:51.966000        0.015
4             2877  FS PA     2015/06/02T13:39:54.712000        0.047

                        eTime  sen  Unnamed: 12
0  2015/06/02T13:39:31.791000   S5          NaN
1  2015/06/02T13:39:44.258000   S5          NaN
2  2015/06/02T13:39:44.380000   S5          NaN
3  2015/06/02T13:39:51.981000   S5          NaN
4  2015/06/02T13:39:54.759000   S5          NaN
```

Step 10: Clean Column Names in Filtered Data

In this step, we clean the column names of the filtered TCP dataset (tcp_data) to remove any leading or trailing spaces. Additionally, we rename specific columns for consistent naming.

Key Points:

- **Strip Leading/Trailing Spaces**:
  - The str.strip() function is applied to tcp_data.columns to remove unnecessary spaces in column names.
- **Rename Columns**:
  - Specific columns are renamed for better readability and consistency using the rename() function with inplace=True.
- **Verification**:
  - The column names are displayed before and after cleaning to confirm the changes.

```
print(tcp_data.columns.tolist())
# Strip leading/trailing spaces from column names
tcp_data.columns = tcp_data.columns.str.strip()

# Verify again
print(tcp_data.columns.tolist())
tcp_data.rename(columns={' duration ': 'duration'}, inplace=True)
```

Output:

```
['                                    sIP', '                                    dIP', 'sPort', 'dPort', 'pro', '     packets', '          bytes', '   flags', '                     sTime', ' duration', '                        eTime', 'sen', 'Unnamed: 12']
['sIP', 'dIP', 'sPort', 'dPort', 'pro', 'packets', 'bytes', 'flags', 'sTime', 'duration', 'eTime', 'sen', 'Unnamed: 12']
```

Step 11: Calculate Packets Per Second

In this step, we calculate a new column, packets_per_second, for the TCP dataset. This column represents the rate of packets handled per second for each row.

Key Points:

- **Calculation**:
  - The formula for packets_per_second is: packets_per_second = packets/duration
  - If duration is 0, the value is set to 0 to avoid division by zero.

- **Lambda Function**:
  - A lambda function is applied to each row of the DataFrame using the apply() method with axis=1.
- **Preview**:
  - The first few rows of the dataset, including the new column, are displayed to verify the calculation.

```
tcp_data['packets_per_second'] = tcp_data.apply(
    lambda row: row['packets'] / row['duration'] if row['duration'] > 0 else 0,
    axis=1
)

# Preview the results
print(tcp_data[['sIP', 'packets', 'duration', 'packets_per_second']].head())
```

Output:

```
sIP  packets  duration  \
0                          192.168.121.145         7      0.012
1                           192.168.121.77         7      0.014
2                           192.168.121.57         8      0.011
3                          192.168.121.244         6      0.015
4                           192.168.121.57        11      0.047

   packets_per_second
0          583.333333
1          500.000000
2          727.272727
3          400.000000
4          234.042553
```

Step 12: Train and Test Models for TCP Traffic Classification

In this step, we perform the following tasks to classify TCP traffic rates based on the calculated packets_per_second:

1. **Ensure Valid Duration Values**:
   - Replace 0 values in the duration column with 1 to avoid division by zero.
2. **Calculate packets_per_second**:
   - Compute the traffic rate as packets divided by duration for each row in the dataset.
3. **Define Traffic Classes**:
   - Create a new column traffic_class to group traffic rates into the following categories:
     - Very Low: 0-10 packets/sec
     - Low: 10-100 packets/sec
     - Medium: 100-1000 packets/sec
     - High: Above 1000 packets/sec
4. **Define Features and Target**:
   - Define X (features) as the packets_per_second column and y (target) as the traffic_class column.
5. **Split the Dataset**:
   - Split the dataset into training and testing sets using an 80-20 split, stratifying by traffic_class.
6. **Handle Missing Values**:

- Use a SimpleImputer to replace missing values with the mean of the respective columns.

7. **Train Very Fast Decision Tree (VFDT)**:
   - Incrementally train the VFDT model using the river library on the training data.
8. **Predict with VFDT**:
   - Use the trained VFDT model to predict traffic classes on the test set and display a classification report.
9. **Train K-Nearest Neighbors (KNN)**:
   - Train the KNN classifier with 3 neighbors using the training data.
10. **Predict with KNN**:
    - Use the trained KNN model to predict traffic classes on the test set and display a classification report.
11. **Query Results for Specific Ranges**:
    - Query both VFDT and KNN models for traffic rates of 50 packets/sec and 500 packets/sec to demonstrate their predictions.

Key Points:

- **VFDT**:
  - Handles incremental learning, suitable for online data streams.
- **KNN**:
  - Performs lazy evaluation, classifying data based on nearest neighbors.
- **Evaluation**:
  - Classification reports provide precision, recall, F1-score, and support for each class.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from river.tree import HoeffdingTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

# Step 1: Load the dataset (assuming tcp_data is already loaded)
# Ensure duration values are valid to avoid division by zero
tcp_data['duration'] = tcp_data['duration'].replace(0, 1)

# Step 2: Calculate packets_per_second if not already present
tcp_data['packets_per_second'] = tcp_data['packets'] / tcp_data['duration']

# Step 3: Define the traffic class based on packets_per_second
tcp_data['traffic_class'] = pd.cut(
    tcp_data['packets_per_second'],
    bins=[0, 10, 100, 1000, float('inf')],
    labels=['Very Low', 'Low', 'Medium', 'High']
)

# Step 4: Define features (X) and target (y)
X = tcp_data[['packets_per_second']].copy()
y = tcp_data['traffic_class']

# Step 5: Split the dataset into training and testing sets
```

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

# Handle missing values (if any)
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Step 6: Train the VFDT (HoeffdingTreeClassifier)
vfdt = HoeffdingTreeClassifier()
for i in range(len(X_train)):
    x = {'packets_per_second': X_train[i][0]}
    vfdt.learn_one(x, str(y_train.iloc[i]))

# Predict using VFDT
vfdt_predictions = [vfdt.predict_one({'packets_per_second': X_test[i][0]}) for i in
range(len(X_test))]
y_test_str = y_test.astype(str)  # Convert y_test to string for comparison
print("VFDT Classification Report:")
print(classification_report(y_test_str, vfdt_predictions))

# Step 7: Train KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predict using KNN
knn_predictions = knn.predict(X_test)
print("KNN Classification Report:")
print(classification_report(y_test, knn_predictions))

# Step 8: Query results for specific ranges
query1 = pd.DataFrame([[50]], columns=['packets_per_second'])
query2 = pd.DataFrame([[500]], columns=['packets_per_second'])

print(f"Query 1 (50 packets/sec) - VFDT: {vfdt.predict_one({'packets_per_second':
50})}")
print(f"Query 2 (500 packets/sec) - VFDT: {vfdt.predict_one({'packets_per_second':
500})}")
print(f"Query 1 (50 packets/sec) - KNN: {knn.predict(query1)}")
print(f"Query 2 (500 packets/sec) - KNN: {knn.predict(query2)}")
```

Output:

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

        High       1.00      1.00      1.00     95743
         Low       1.00      1.00      1.00    152509
      Medium       1.00      1.00      1.00    691047
    Very Low       1.00      1.00      1.00    439027
```

```
    accuracy                           1.00   1378326
   macro avg       1.00      1.00      1.00   1378326
weighted avg       1.00      1.00      1.00   1378326

KNN Classification Report:
              precision    recall  f1-score   support

        High       1.00      1.00      1.00     95743
         Low       1.00      1.00      1.00    152509
      Medium       1.00      1.00      1.00    691047
    Very Low       1.00      1.00      1.00    439027

    accuracy                           1.00   1378326
   macro avg       1.00      1.00      1.00   1378326
weighted avg       1.00      1.00      1.00   1378326

Query 1 (50 packets/sec) - VFDT: Low
Query 2 (500 packets/sec) - VFDT: Medium
Query 1 (50 packets/sec) - KNN: ['Low']
Query 2 (500 packets/sec) - KNN: ['Medium']
```

Step 13: Visualize Traffic Class Distribution

In this step, we create a bar plot to visualize the distribution of the traffic classes (traffic_class) in the TCP dataset.

Key Points:

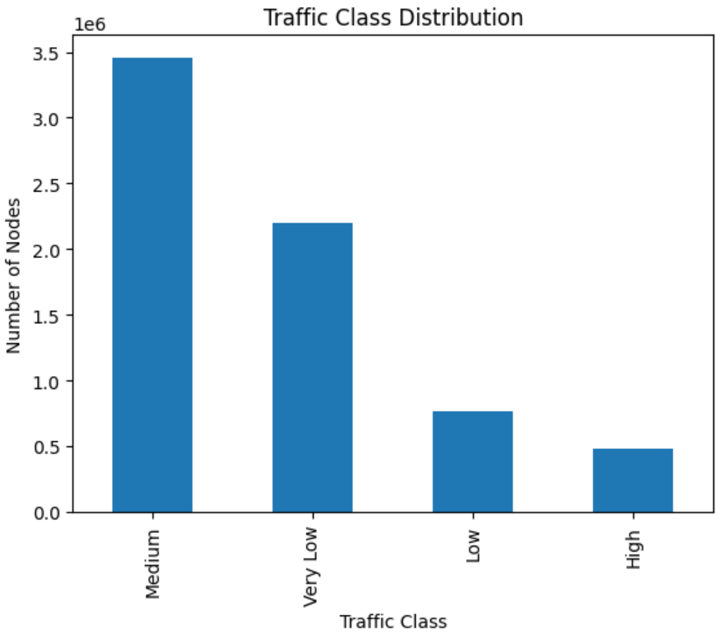
- **Value Counts**:
  - The value_counts() function is used to count the occurrences of each traffic class.
- **Bar Plot**:
  - A bar plot is created using the plot(kind='bar') function in Pandas.
- **Visualization**:
  - The plot includes a title, labeled axes, and displays the number of nodes in each traffic class.

Purpose:

- This visualization helps understand the distribution of traffic classes, enabling us to identify any imbalances in the dataset.

```python
import matplotlib.pyplot as plt

# Plot traffic class distribution
tcp_data['traffic_class'].value_counts().plot(kind='bar')
plt.title("Traffic Class Distribution")
plt.xlabel("Traffic Class")
plt.ylabel("Number of Nodes")
plt.show()
```

Output:



Step 14: Display Training and Testing Dataset Sizes

In this step, we display the number of samples in the training and testing datasets after splitting the original dataset.

Key Points:

- **Training Samples**:
  - The size of the training dataset is determined using the len() function on X_train.
- **Testing Samples**:
  - The size of the testing dataset is determined using the len() function on X_test.
- **Purpose**:
  - This step ensures that the dataset is split correctly, and both training and testing datasets have the expected number of samples.

Output:

- The number of training and testing samples is printed for verification.

```
print(f"Number of training samples: {len(X_train)}")
print(f"Number of test samples: {len(X_test)}")
```

Output:

```
Number of training samples: 5513304
Number of test samples: 1378326
```

Step 15: Train and Test Optimized KNN Classifier

In this step, we train and test an optimized K-Nearest Neighbors (KNN) classifier on the TCP dataset to predict traffic classes.

Key Points:

1. **Initialization**:
   - The KNN model is initialized with the following parameters:
     - n_neighbors=3: Considers the 3 nearest neighbors for classification.
     - algorithm='ball_tree': Uses a Ball Tree data structure for efficient neighbor searches.
     - n_jobs=-1: Utilizes all available CPU cores for parallel computation.

2. **Training**:
    - The KNN model is trained using the fit() method on the training dataset (X_train and y_train).
3. **Testing on a Subset**:
    - A subset of the test dataset (first 100 samples) is used to evaluate the model for faster computation.
4. **Predictions**:
    - Predictions are made using the predict() method on the subset of test data (subset_X_test).
5. **Accuracy Score**:
    - The accuracy_score() function computes the classification accuracy, comparing the predicted values with the true labels (subset_y_test).

Purpose:
- To evaluate the performance of the optimized KNN classifier in predicting traffic classes.

Output:
- The accuracy of the KNN model on the subset of test data is printed.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Initialize KNN with optimizations
knn = KNeighborsClassifier(n_neighbors=3, algorithm='ball_tree', n_jobs=-1)
knn.fit(X_train, y_train)

# Use a subset of X_test
subset_X_test = X_test[:100]
subset_y_test = y_test[:100]

# Predict on the subset
knn_predictions = knn.predict(subset_X_test)
knn_accuracy = accuracy_score(subset_y_test, knn_predictions)

print(f"KNN Accuracy (optimized): {knn_accuracy}")
```

Output:
```
KNN Accuracy (optimized): 1.0
```

3. Detect the anomaly in the fetched data. Anomalies are defined as the nodes which send large number of TCP packets per sec, say 1K per sec. You are free to define your own threshold based on your observation in the given data stream.

Step 1: Identify and Display Anomalous Nodes

In this step, we identify nodes with anomalous TCP traffic based on a predefined threshold for packets_per_second. Anomalous nodes are then displayed for further analysis.

Key Points:

1. **Anomaly Threshold**:
    - Define a threshold value (anomaly_threshold = 1000) for identifying anomalies in traffic rates.

2. **Filter Anomalous Nodes**:
   - Filter rows in the dataset where the packets_per_second value exceeds the defined threshold.
3. **Display Anomalous Nodes**:
   - Display the source IP (sIP), destination IP (dIP), and packets_per_second values for the detected anomalies.

Purpose:

- To detect and analyze nodes with unusually high TCP traffic rates, which may indicate abnormal behavior or events.

Output:

- A list of anomalous nodes exceeding the threshold, along with their key attributes.

```
# Define the threshold for anomalies
anomaly_threshold = 1000

# Identify anomalous nodes
anomalies = tcp_data[tcp_data['packets_per_second'] > anomaly_threshold]

# Display anomalous nodes
print(f"Anomalies detected (packets_per_second > {anomaly_threshold}):")
print(anomalies[['sIP', 'dIP', 'packets_per_second']])
```

Output:

```
Anomalies detected (packets_per_second > 1000):
                                      sIP  \
1004                      192.168.141.189
1451                        192.168.40.92
1501                        192.168.40.25
1773                        192.168.40.20
3650                        192.168.40.51
...                                   ...
21216399                      10.0.40.21
21216400                      10.0.40.21
21216401                      10.0.40.21
21216403                      10.0.40.21
21216406                      10.0.40.21

                                      dIP  packets_per_second
1004                          67.215.0.8         2000.000000
1451                          10.0.40.23         1500.000000
1501                          10.0.40.20         2000.000000
1773                     192.168.143.239         1666.666667
3650                          10.0.40.20         2000.000000
...                                   ...                 ...
21216399                      10.0.20.59         2500.000000
21216400                      10.0.20.59         2500.000000
21216401                      10.0.20.59         2500.000000
21216403                      10.0.20.59         2500.000000
21216406                      10.0.20.58         3000.000000

[478716 rows x 3 columns]
```

Step 2: Count and Analyze Anomalies

In this step, we count the total number of anomalous nodes and analyze the distribution of their packets_per_second values.

Key Points:

1. **Count Anomalies**:
   - Use the shape[0] attribute to get the total number of rows (anomalies) in the filtered dataset (anomalies).
2. **Analyze Distribution**:
   - Use the describe() method to generate summary statistics for the packets_per_second column of the anomalous nodes.
   - Key statistics include:
     - Count: Total number of anomalies.
     - Mean: Average packets_per_second value for anomalies.
     - Standard Deviation (std): Spread of the values.
     - Min/Max: Minimum and maximum traffic rates for anomalies.
     - Percentiles (25%, 50%, 75%): Quartiles of the data distribution.

Purpose:

- To quantify and analyze the traffic characteristics of the identified anomalies.

Output:

- Total number of anomalies detected.
- Summary statistics for the packets_per_second column in the anomalous data.

```
# Count the number of anomalies
num_anomalies = anomalies.shape[0]
print(f"Number of anomalies detected: {num_anomalies}")

# Analyze distribution of packets_per_second
print("Summary statistics for anomalies:")
print(anomalies['packets_per_second'].describe())
```

Output:

```
Number of anomalies detected: 478716
Summary statistics for anomalies:
count    478716.000000
mean       1512.644088
std         508.879721
min        1000.000000
25%        1250.000000
50%        1333.333333
75%        1500.000000
max       18899.478293
Name: packets_per_second, dtype: float64
```

Step 3: Visualize Packets Per Second (Normal vs Anomalous Nodes)

In this step, we create a scatter plot to visualize the packets_per_second values for all nodes, distinguishing between normal and anomalous nodes.

Key Points:

1. **Scatter Plot**:
   - Each point represents a node with its source IP (sIP) on the x-axis and its packets_per_second value on the y-axis.
2. **Normal Nodes**:

- Plotted in the default color (blue) to represent nodes with traffic below the anomaly threshold.

3. **Anomalous Nodes**:
   - Highlighted in red to indicate nodes with traffic exceeding the anomaly threshold.
4. **Threshold Line**:
   - A horizontal line (axhline) is added to represent the anomaly threshold (anomaly_threshold).
5. **Customization**:
   - Labels, title, and legend are added for clarity.
   - Source IPs are rotated for better readability on the x-axis.

Purpose:

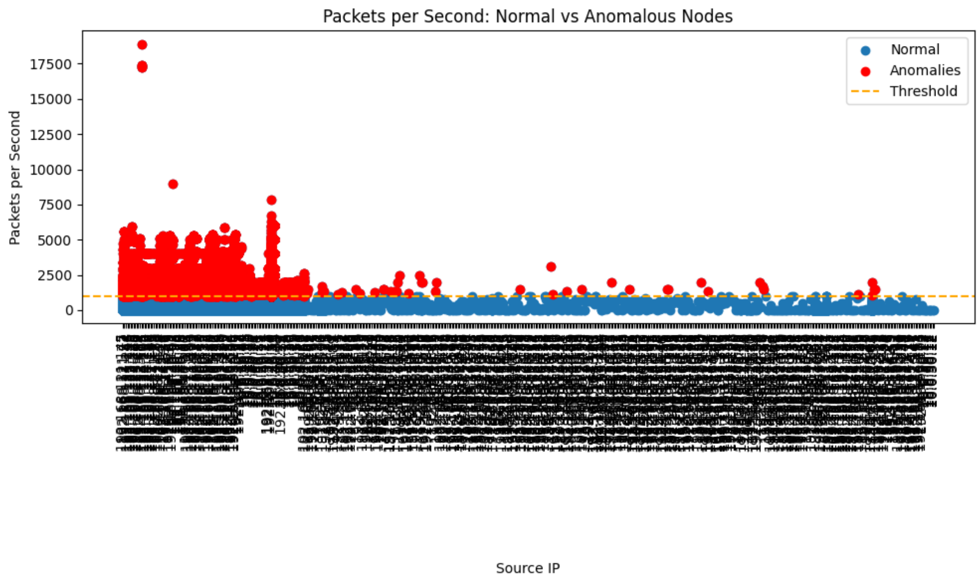
- To visually distinguish between normal and anomalous nodes based on their packets_per_second values.

Output:

- A scatter plot showing the distribution of normal and anomalous nodes.

```
import matplotlib.pyplot as plt

# Scatter plot of packets_per_second
plt.figure(figsize=(10, 6))
plt.scatter(tcp_data['sIP'], tcp_data['packets_per_second'], label="Normal")
plt.scatter(anomalies['sIP'], anomalies['packets_per_second'], color='red',
label="Anomalies")
plt.axhline(y=anomaly_threshold, color='orange', linestyle='--', label="Threshold")
plt.title("Packets per Second: Normal vs Anomalous Nodes")
plt.xlabel("Source IP")
plt.ylabel("Packets per Second")
plt.legend()
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Output:



Step 4 : Visualize Packets Per Second by IP Group (Normal vs Anomalous)

In this step, we group the TCP data by the first two segments of the IP address (e.g., 192.168) and analyze the average packets_per_second within each group. The results are visualized in a bar chart, distinguishing between normal and anomalous groups.

Key Points:

1. **Group by IP**:
   - The ip_group column is created by extracting the first two segments of the source IP (sIP).
   - Data is grouped by ip_group, and the average packets_per_second is calculated for each group.
2. **Identify Anomalous Groups**:
   - Groups with an average packets_per_second exceeding the anomaly threshold (anomaly_threshold = 1000) are marked as anomalous using a new column is_anomaly.
3. **Bar Plot**:
   - Normal groups are plotted in blue.
   - Anomalous groups are highlighted in red.
   - A horizontal orange line represents the anomaly threshold.
4. **Customization**:
   - The bar chart includes labels, a title, and a legend for clear interpretation.
   - The x-axis labels (IP groups) are rotated for readability.

Purpose:

- To identify and visualize patterns in traffic behavior across different IP groups, highlighting groups with anomalous traffic rates.

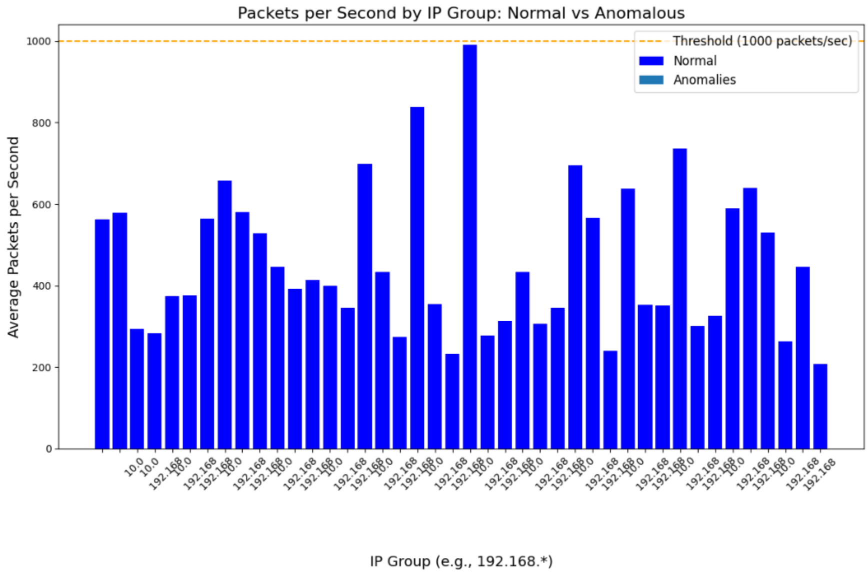
Output:

- A bar chart showing average packets_per_second for each IP group, distinguishing between normal and anomalous groups.

```
import pandas as pd
import matplotlib.pyplot as plt

# Group the data by the first two segments of the IP address (e.g., '192.168')
tcp_data['ip_group'] = tcp_data['sIP'].apply(lambda x: '.'.join(x.split('.')[:2]))

# Aggregate packets per second by group
grouped_data = tcp_data.groupby('ip_group')['packets_per_second'].mean().reset_index()

# Identify anomalous groups
anomaly_threshold = 1000  # Example threshold for anomalies
grouped_data['is_anomaly'] = grouped_data['packets_per_second'] > anomaly_threshold

# Plot the grouped data
plt.figure(figsize=(12, 8))

# Normal groups
normal_groups = grouped_data[~grouped_data['is_anomaly']]
plt.bar(
    normal_groups['ip_group'],
    normal_groups['packets_per_second'],
    color='blue',  # Normal group color
    label="Normal",
```

```
)

# Anomalous groups
anomalous_groups = grouped_data[grouped_data['is_anomaly']]
plt.bar(
    anomalous_groups['ip_group'],
    anomalous_groups['packets_per_second'],
    color='red',  # Anomalous group color
    label="Anomalies",
)

# Add a threshold line
plt.axhline(
    y=anomaly_threshold,
    color='orange',
    linestyle='--',
    label=f"Threshold ({anomaly_threshold} packets/sec)",
)

# Title and labels
plt.title("Packets per Second by IP Group: Normal vs Anomalous", fontsize=16)
plt.xlabel("IP Group (e.g., 192.168.*)", fontsize=14)
plt.ylabel("Average Packets per Second", fontsize=14)

# Rotate x-axis labels for readability
plt.xticks(rotation=45, fontsize=10)

# Add legend
plt.legend(loc="upper right", fontsize=12)

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

Output:



Step 5: Group and Count Anomalies by Source IP

In this step, we group the anomalous nodes by their source IP (sIP) and count the number of anomalies for each source IP.

Key Points:

1. **Group by Source IP**:
    - Use the groupby() function to group the anomalies by the sIP column.
2. **Count Anomalies**:
    - Use the size() function to count the number of anomalies for each source IP.
    - Reset the index of the resulting DataFrame to create a new column anomaly_count that stores the counts.
3. **Sort Results**:
    - The results are sorted in descending order of the anomaly_count column to identify the most frequently anomalous source IPs.
4. **Output**:
    - A list of source IPs with their corresponding anomaly counts is displayed.

Purpose:

- To identify and quantify the nodes (source IPs) contributing to anomalous traffic patterns.

Output:

- A sorted list of source IPs with their anomaly counts, starting from the most anomalous.

```
# Group by source IP and count anomalies
anomalies_by_sIP = anomalies.groupby('sIP').size().reset_index(name='anomaly_count')
print("Anomalies by Source IP:")
print(anomalies_by_sIP.sort_values(by='anomaly_count', ascending=False))
```

Output:

```
Anomalies by Source IP:
                              sIP  anomaly_count
3                      10.0.40.21          52182
10                     10.0.40.53          50821
44                   192.168.70.10         26882
1                      10.0.20.59          25355
34                   192.168.40.20         17370
..                            ...            ...
```

```
143                          10.0.20.59             1
144                        192.168.121.77           1
145                       192.168.163.154           1
146                         192.168.50.12           1
147                         192.168.70.10           1

[148 rows x 2 columns]
```

Step 6: Calculate Total and Percentage of Anomalous Packets

In this step, we calculate the total number of packets handled by anomalous nodes and determine their percentage contribution to the overall packets in the dataset.

Key Points:

1. **Total Anomalous Packets**:
   - Use the sum() function on the packets column of the anomalies DataFrame to calculate the total number of packets handled by anomalous nodes.
2. **Percentage of Anomalous Packets**:
   - Calculate the percentage of packets handled by anomalous nodes relative to the total packets in the dataset: percentage = (total anomalous packets) / total packets in dataset) * 100
3. **Output**:
   - The percentage of anomalous packets is displayed, rounded to two decimal places.

Purpose:

- To quantify the impact of anomalous nodes on the overall traffic in the dataset.

Output:

- The total number of packets handled by anomalous nodes.
- The percentage of anomalous packets relative to the total traffic.

```
# Calculate total packets for anomalies
total_anomalous_packets = anomalies['packets'].sum()

# Calculate percentage of anomalous packets
percentage_anomalous_packets = (total_anomalous_packets / tcp_data['packets'].sum()) *
100
print(f"Percentage of anomalous packets: {percentage_anomalous_packets:.2f}%")
```

Output:

```
Percentage of anomalous packets: 9.67%
```