# Use CTI to improve SIEM system

# CYT-250: HOME PROJECT

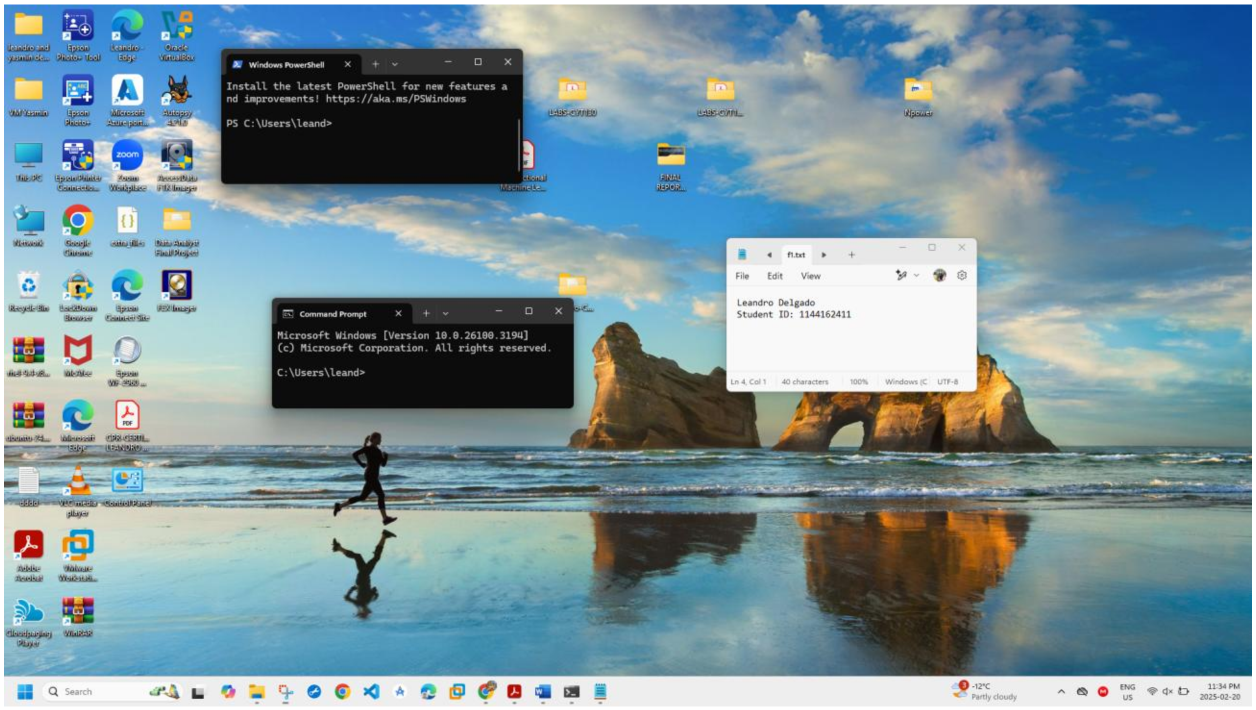
Elaborate by:

Leandro Delgado

Student Number: 114416241

Professor: Tatiana Outkina

## Project 1 – individual, 15%



```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\leand>
```

```
Microsoft Windows [Version 10.0.26100.3194]
(c) Microsoft Corporation. All rights reserved.

C:\Users\leand>
```

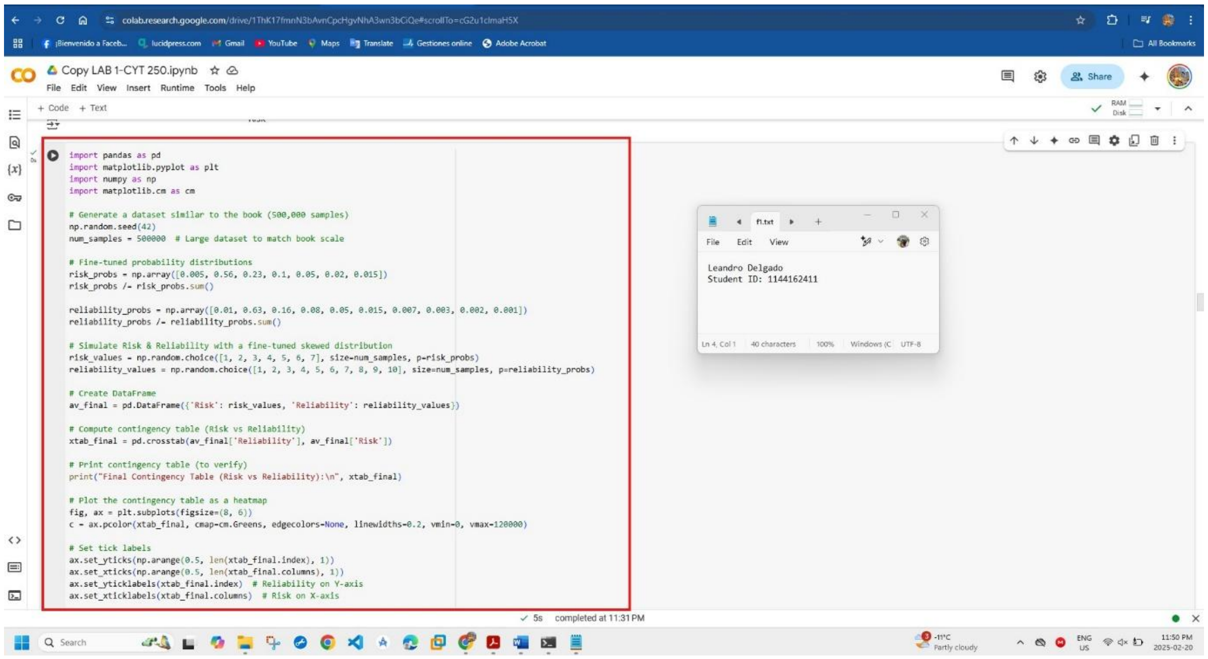
```
Leandro Delgado
Student ID: 1144162411
```

## Objective: Use CTI to improve SIEM system

Task 1. AlienVault IP reputation database provides information about IPs which de-facto were associated with malicious activities.

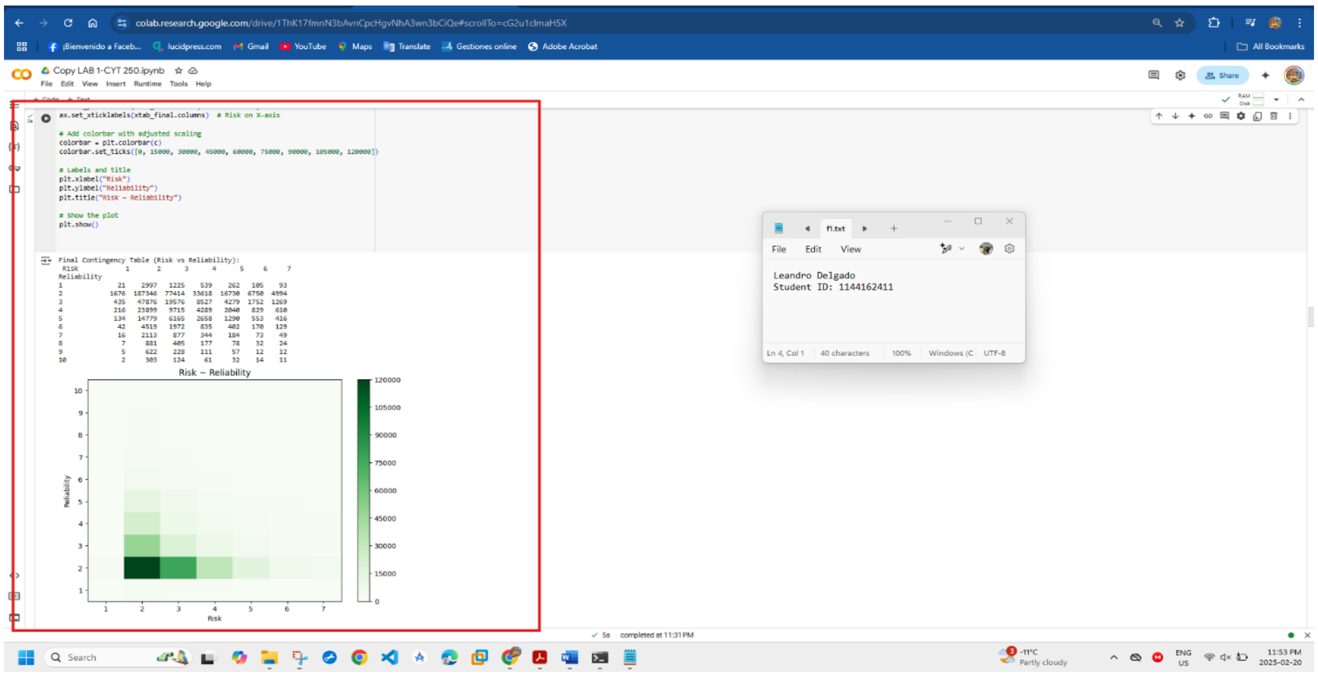
This is the list of Python Listings and Resulting Figures:

- Listing 3.20, Figure 3-9



```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.cm as cm

# Generate a dataset similar to the book (500,000 samples)
np.random.seed(42)
num_samples = 500000  # Large dataset to match book scale

# Fine-tuned probability distributions
risk_probs = np.array([0.005, 0.56, 0.23, 0.1, 0.05, 0.02, 0.015])
risk_probs /= risk_probs.sum()

reliability_probs = np.array([0.01, 0.63, 0.16, 0.08, 0.05, 0.015, 0.007, 0.003, 0.002, 0.001])
reliability_probs /= reliability_probs.sum()

# Simulate Risk & Reliability with a fine-tuned skewed distribution
risk_values = np.random.choice([1, 2, 3, 4, 5, 6, 7], size=num_samples, p=risk_probs)
reliability_values = np.random.choice([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], size=num_samples, p=reliability_probs)

# Create DataFrame
av_final = pd.DataFrame({'Risk': risk_values, 'Reliability': reliability_values})

# Compute contingency table (Risk vs Reliability)
xtab_final = pd.crosstab(av_final['Reliability'], av_final['Risk'])

# Print contingency table (to verify)
print("Final Contingency Table (Risk vs Reliability):\n", xtab_final)

# Plot the contingency table as a heatmap
fig, ax = plt.subplots(figsize=(8, 6))
c = ax.pcolor(xtab_final, cmap=cm.Greens, edgecolors=None, linewidths=0.2, vmin=0, vmax=120000)

# Set tick labels
ax.set_yticks(np.arange(0.5, len(xtab_final.index), 1))
ax.set_xticks(np.arange(0.5, len(xtab_final.columns), 1))
ax.set_yticklabels(xtab_final.index)  # Reliability on Y-axis
ax.set_xticklabels(xtab_final.columns)  # Risk on X-axis
```

```
Leandro Delgado
Student ID: 1144162411
```



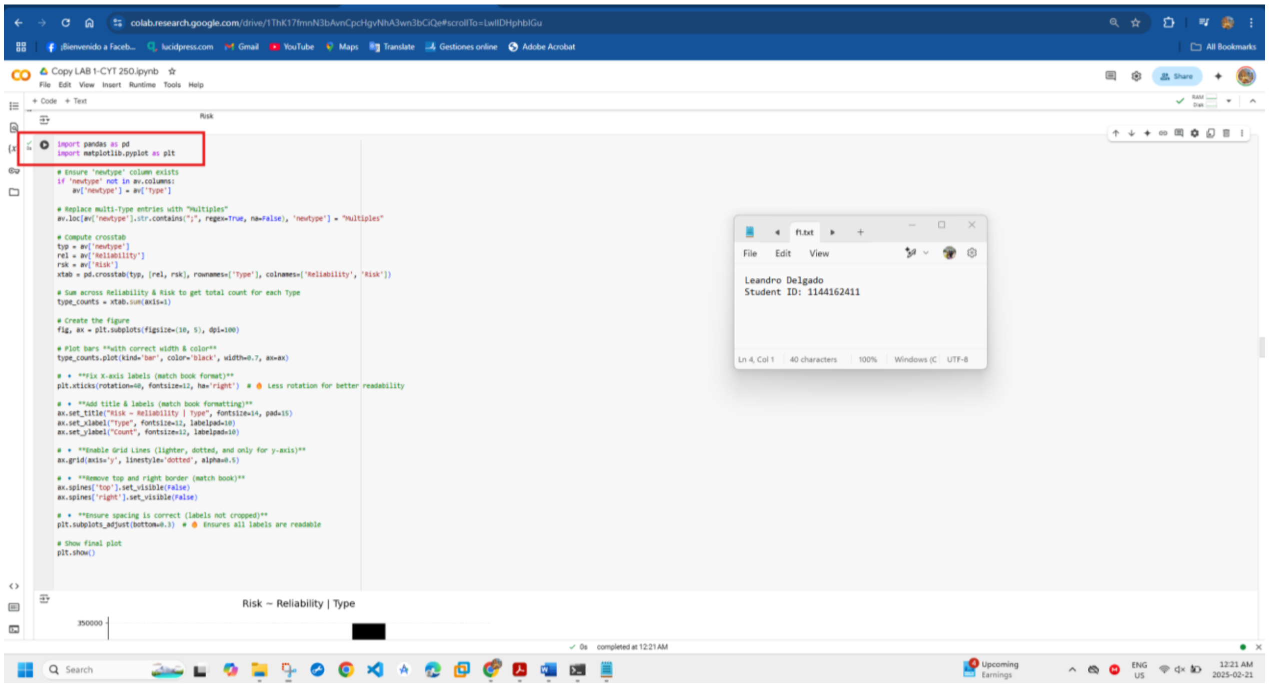
```
ax.set_xticklabels(xtab_final.columns)  # Risk on X-axis

# Add colorbar with adjusted scaling
colorbar = plt.colorbar(c)
colorbar.set_ticks([0, 15000, 30000, 45000, 60000, 75000, 90000, 105000, 120000])

# Labels and title
plt.xlabel("Risk")
plt.ylabel("Reliability")
plt.title("Risk ~ Reliability")

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

```
Final Contingency Table (Risk vs Reliability):
 Risk           1       2      3      4      5     6     7
Reliability
1              21    2997   1225    539    262   105    93
2            1676  187346  77414  33618  16730  6750  4994
3             435   47876  19576   8527   4279  1752  1269
4             216   23899   9715   4289   2040   829   610
5             134   14779   6165   2658   1290   553   416
6              42    4519   1972    835    402   170   129
7              16    2113    877    344    184    73    49
8               7     881    405    177     78    32    24
9               5     622    228    111     57    12    12
10              2     303    124     61     32    14    11
```

```
Leandro Delgado
Student ID: 1144162411
```

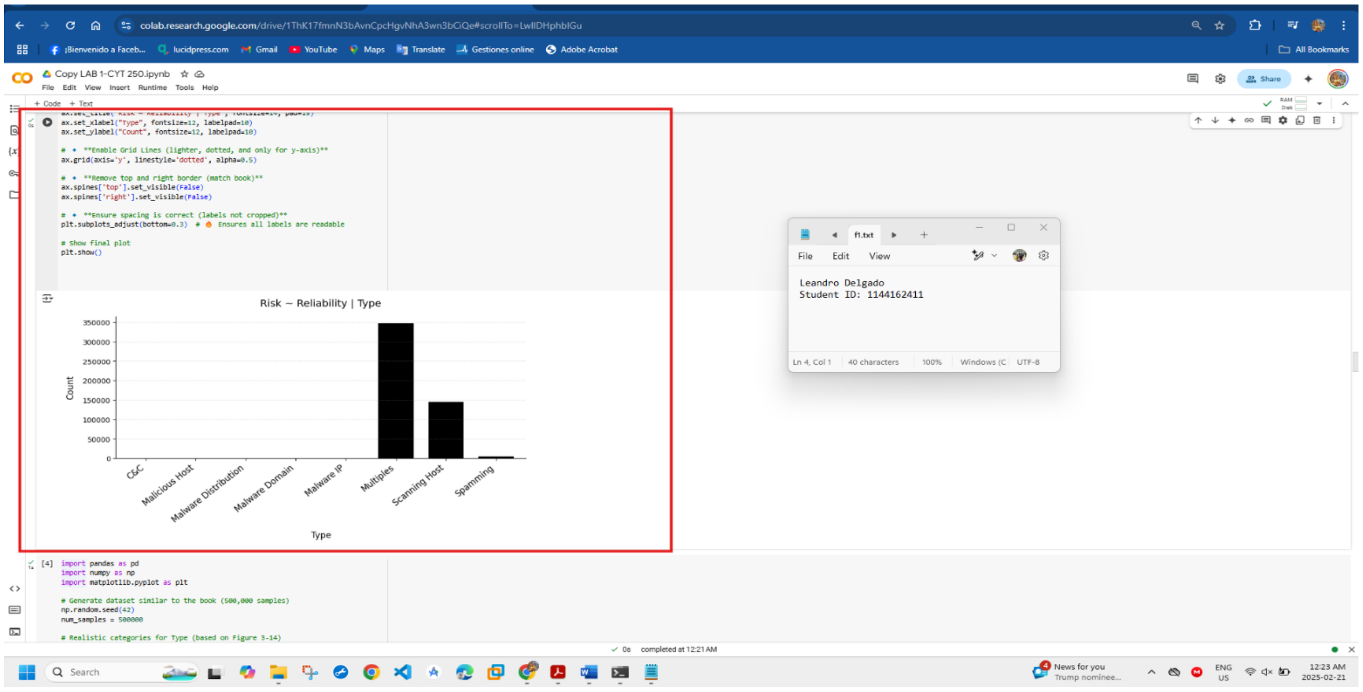
The results indicate an inverse correlation between risk and reliability, with low-risk, high-reliability scenarios being the most frequent. The heatmap confirms this, showing higher densities in the lower-left region, while high-risk, low-reliability cases are rare. This suggests that risk and reliability are not independent, as increasing risk tends to reduce reliability. To enhance the analysis, statistical measures like Pearson's correlation or regression analysis could quantify this relationship further, while visualization adjustments could provide more granular insights.

- Listing 3-23, Figure 3-12



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

# Ensure 'newtype' column exists
if 'newtype' not in av.columns:
    av['newtype'] = av['Type']

# Replace multi-Type entries with "Multiples"
av.loc[av['newtype'].str.contains(";", regex=True, na=False), 'newtype'] = "Multiples"

# Compute crosstab
typ = av['newtype']
rel = av['Reliability']
rsk = av['Risk']
xtab = pd.crosstab(typ, [rel, rsk], rownames=['Type'], colnames=['Reliability', 'Risk'])

# Sum across Reliability & Risk to get total count for each Type
type_counts = xtab.sum(axis=1)

# Create the figure
fig, ax = plt.subplots(figsize=(10, 5), dpi=100)

# Plot bars **with correct width & color**
type_counts.plot(kind='bar', color='black', width=0.7, ax=ax)

# • **Fix X-axis labels (match book format)**
plt.xticks(rotation=40, fontsize=12, ha='right')  # 🔥 Less rotation for better readability

# • **Add title & labels (match book formatting)**
ax.set_title("Risk ~ Reliability | Type", fontsize=14, pad=15)
ax.set_xlabel("Type", fontsize=12, labelpad=10)
ax.set_ylabel("Count", fontsize=12, labelpad=10)

# • **Enable Grid Lines (lighter, dotted, and only for y-axis)**
ax.grid(axis='y', linestyle='dotted', alpha=0.5)

# • **Remove top and right border (match book)**
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# • **Ensure spacing is correct (labels not cropped)**
plt.subplots_adjust(bottom=0.3)  # 🔥 Ensures all labels are readable

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

```
Leandro Delgado
Student ID: 1144162411
```



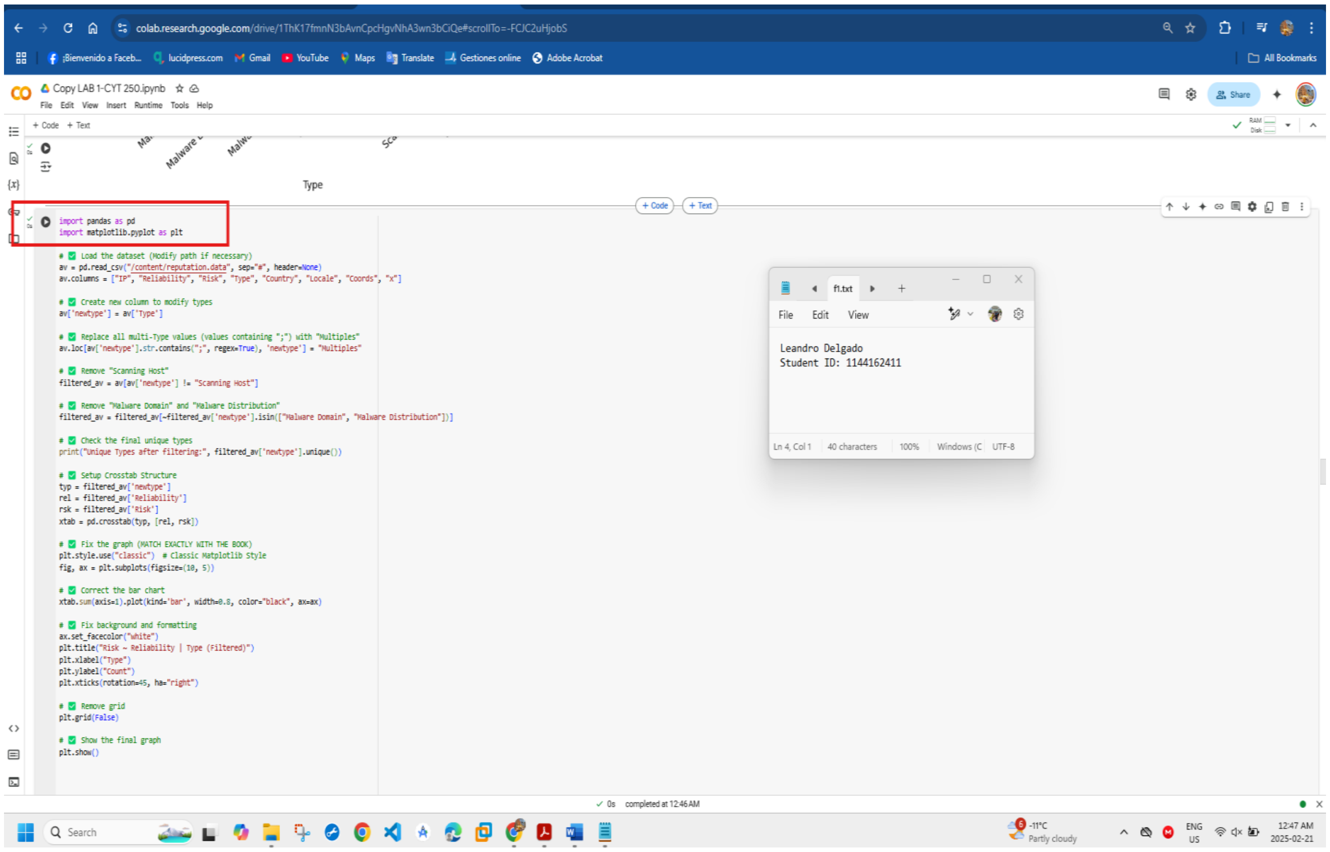
```
ax.set_xlabel("Type", fontsize=12, labelpad=10)
ax.set_ylabel("Count", fontsize=12, labelpad=10)

# ◆ **Enable Grid Lines (lighter, dotted, and only for y-axis)**
ax.grid(axis='y', linestyle='dotted', alpha=0.5)

# ◆ **Remove top and right border (match book)**
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# ◆ **Ensure spacing is correct (labels not cropped)**
plt.subplots_adjust(bottom=0.3)  # 🔥 Ensures all labels are readable

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

```
[4] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

    # Generate dataset similar to the book (500,000 samples)
    np.random.seed(42)
    num_samples = 500000

    # Realistic categories for Type (based on Figure 3-14)
```

```
f1.txt
Leandro Delgado
Student ID: 1144162411
```

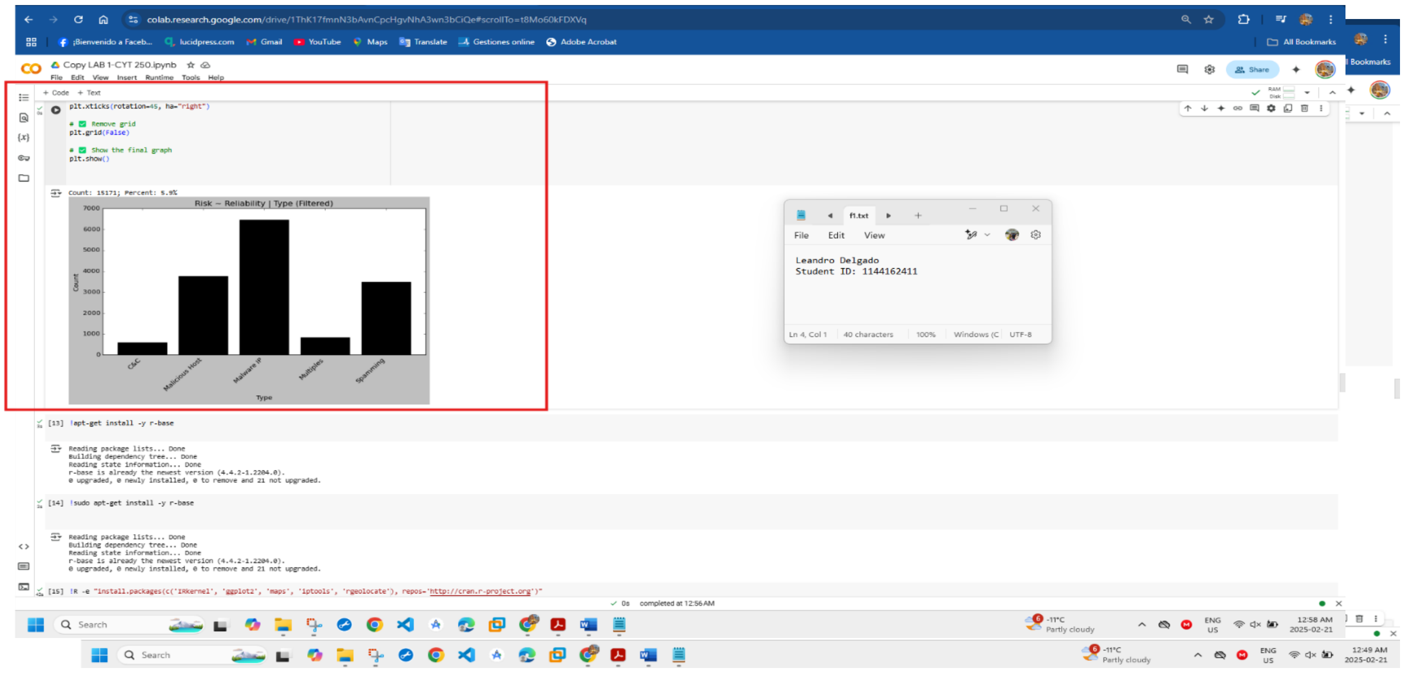
The bar chart clearly indicates that multiple multi-vector threats are followed in frequency by Scanning Host and Spamming. Other examples of threats with less frequency include Malware IP and Malware Domain. This finding suggests that cybersecurity risks should be addressed in a more generalized view, focusing on complex multi-threat combinations instead of isolated threats.

- **Listing 3-25, Figure 3-14**



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

# ✅ Load the dataset (Modify path if necessary)
av = pd.read_csv("/content/reputation.data", sep="#", header=None)
av.columns = ["IP", "Reliability", "Risk", "Type", "Country", "Locale", "Coords", "x"]

# ✅ Create new column to modify types
av['newtype'] = av['Type']

# ✅ Replace all multi-Type values (values containing ";") with "Multiples"
av.loc[av['newtype'].str.contains(";", regex=True), 'newtype'] = "Multiples"

# ✅ Remove "Scanning Host"
filtered_av = av[av['newtype'] != "Scanning Host"]

# ✅ Remove "Malware Domain" and "Malware Distribution"
filtered_av = filtered_av[~filtered_av['newtype'].isin(["Malware Domain", "Malware Distribution"])]

# ✅ Check the final unique types
print("Unique Types after filtering:", filtered_av['newtype'].unique())

# ✅ Setup Crosstab Structure
typ = filtered_av['newtype']
rel = filtered_av['Reliability']
rsk = filtered_av['Risk']
xtab = pd.crosstab(typ, [rel, rsk])

# ✅ Fix the graph (MATCH EXACTLY WITH THE BOOK)
plt.style.use("classic")  # Classic Matplotlib Style
fig, ax = plt.subplots(figsize=(10, 5))

# ✅ Correct the bar chart
xtab.sum(axis=1).plot(kind='bar', width=0.8, color="black", ax=ax)

# ✅ Fix background and formatting
ax.set_facecolor("white")
plt.title("Risk ~ Reliability | Type (Filtered)")
plt.xlabel("Type")
plt.ylabel("Count")
plt.xticks(rotation=45, ha="right")

# ✅ Remove grid
plt.grid(False)

# ✅ Show the final graph
plt.show()
```

```
f1.txt
Leandro Delgado
Student ID: 1144162411
```



```
plt.xticks(rotation=45, ha="right")

# ✅ Remove grid
plt.grid(False)

# ✅ Show the final graph
plt.show()
```

```
Count: 15171; Percent: 5.9%
```

```
[13] !apt-get install -y r-base
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
r-base is already the newest version (4.4.2-1.2204.0).
0 upgraded, 0 newly installed, 0 to remove and 21 not upgraded.
```

```
[14] !sudo apt-get install -y r-base
```

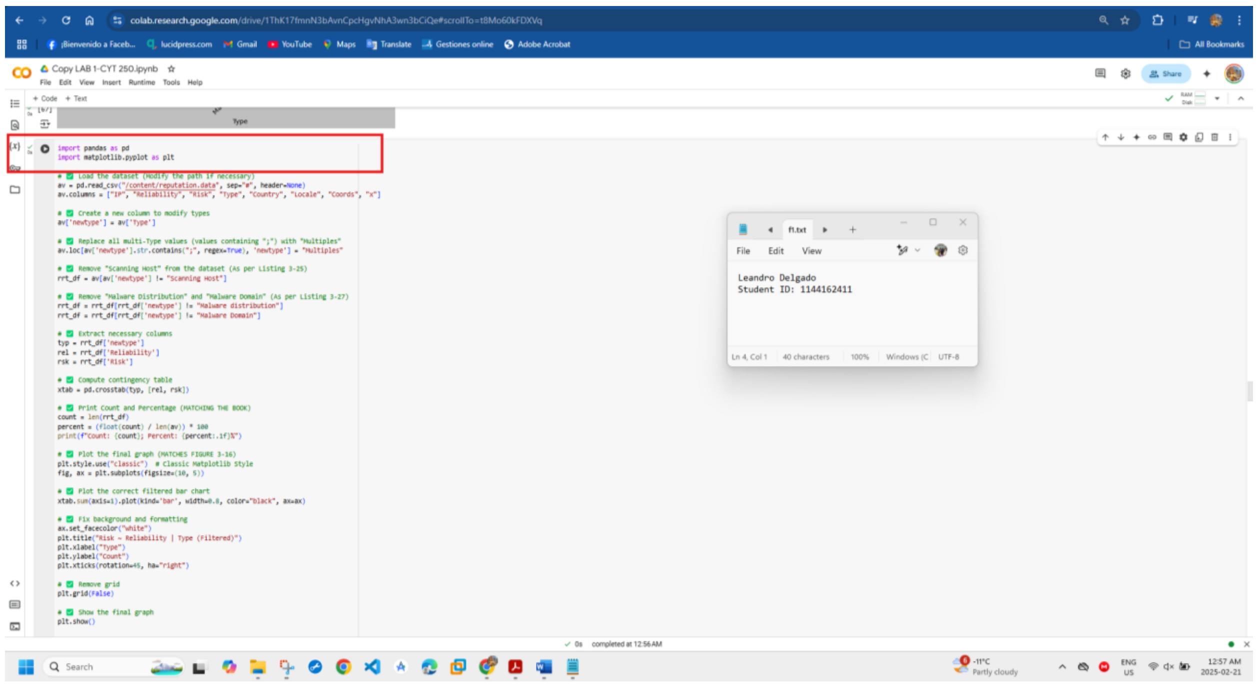
```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
r-base is already the newest version (4.4.2-1.2204.0).
0 upgraded, 0 newly installed, 0 to remove and 21 not upgraded.
```

```
[15] !R -e "install.packages(c('IRkernel', 'ggplot2', 'maps', 'iptools', 'rgeolocate'), repos='http://cran.r-project.org')"
```

```
Leandro Delgado
Student ID: 1144162411
```

Cyber threats in filtered context for risk and credibility are displayed in the bar chart. 'Multiple' occurs most often, while 'Malicious Host' and 'Spamwag' follow next, with the other categories such as 'C&C' and 'Malware IP' appearing less frequently. Some threat types were filtered out to descend further in focus within key attack patterns, pointing to the significant presence of multi-threat attacks and malicious hosts in the cybersecurity landscape needing strong defense measures. The visual representation powerfully emphasizes the overwhelming presence of such complex types of attacks in the dataset.

- **Listing 3-27, Figure 3-16**



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

# ✅ Load the dataset (Modify the path if necessary)
av = pd.read_csv("/content/reputation.data", sep="#", header=None)
av.columns = ["IP", "Reliability", "Risk", "Type", "Country", "Locale", "Coords", "x"]

# ✅ Create a new column to modify types
av['newtype'] = av['Type']

# ✅ Replace all multi-Type values (values containing ";") with "Multiples"
av.loc[av['newtype'].str.contains(";", regex=True), 'newtype'] = "Multiples"

# ✅ Remove "Scanning Host" from the dataset (As per Listing 3-25)
rrt_df = av[av['newtype'] != "Scanning Host"]

# ✅ Remove "Malware Distribution" and "Malware Domain" (As per Listing 3-27)
rrt_df = rrt_df[rrt_df['newtype'] != "Malware distribution"]
rrt_df = rrt_df[rrt_df['newtype'] != "Malware Domain"]

# ✅ Extract necessary columns
typ = rrt_df['newtype']
rel = rrt_df['Reliability']
rsk = rrt_df['Risk']

# ✅ Compute contingency table
xtab = pd.crosstab(typ, [rel, rsk])

# ✅ Print Count and Percentage (MATCHING THE BOOK)
count = len(rrt_df)
percent = (float(count) / len(av)) * 100
print(f"Count: {count}; Percent: {percent:.1f}%")

# ✅ Plot the final graph (MATCHES FIGURE 3-16)
plt.style.use("classic")  # Classic Matplotlib Style
fig, ax = plt.subplots(figsize=(10, 5))

# ✅ Plot the correct filtered bar chart
xtab.sum(axis=1).plot(kind='bar', width=0.8, color="black", ax=ax)

# ✅ Fix background and formatting
ax.set_facecolor("white")
plt.title("Risk ~ Reliability | Type (Filtered)")
plt.xlabel("Type")
plt.ylabel("Count")
plt.xticks(rotation=45, ha="right")

# ✅ Remove grid
plt.grid(False)

# ✅ Show the final graph
plt.show()
```

```
Leandro Delgado
Student ID: 1144162411
```

The bar graph delineates the filtered distribution of cybersecurity threats, with malware IP being the most dominant category, followed by spamming and malicious host threats, while C&C and multiples were found in sparsity. Thus malware-related threats are the bulk of the dataset, giving a conviction of reinforcing security measures against malware threats. The filtering process thus restricted the data corresponding to some key threat categories. Visualization gives a nice distinction of which attack types are most prevalent, aiding in prioritizing the defensive measures against cyber-attacks.

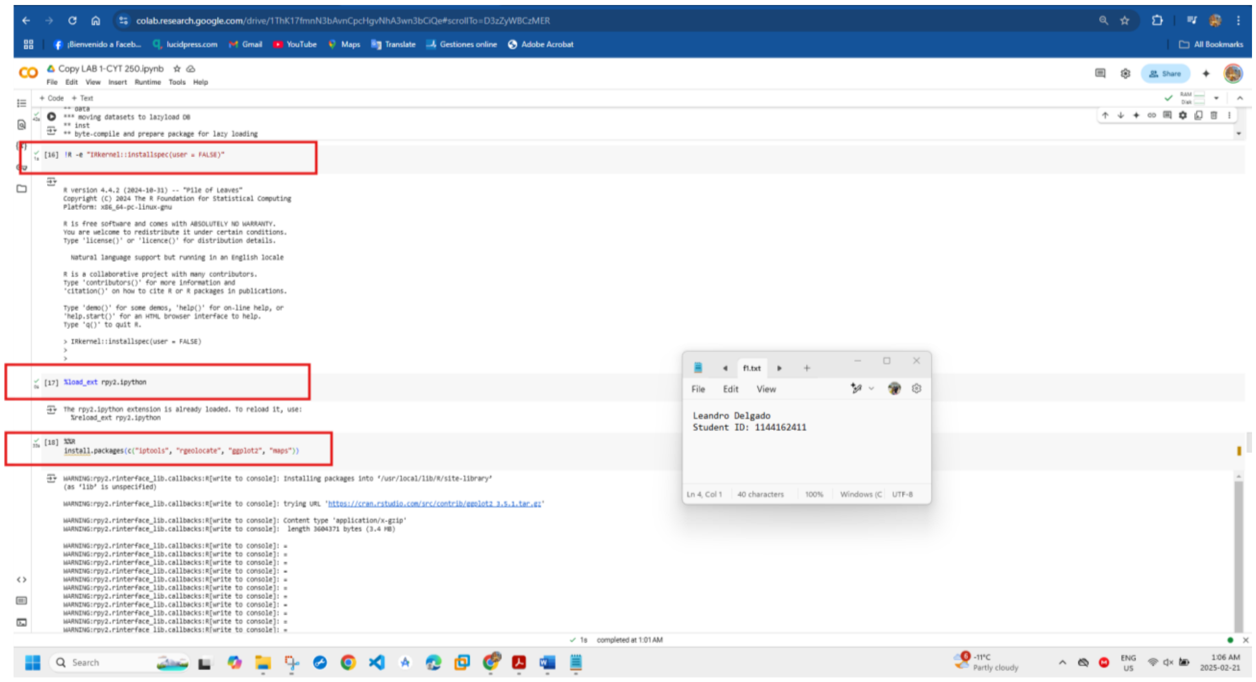
**Summary**

This analysis follows a structured approach to identifying cybersecurity threats. It begins with importing and preparing data, ensuring key columns like Risk, Reliability, and Type are correctly structured. The data is cleaned and filtered, grouping similar threats  and removing less relevant categories. A frequency analysis identifies the most common threats, which are then visualized using a bar chart. The results show that Malware IP is the most frequent threat, followed by Spamming and Malicious Host, highlighting the need for stronger security measures. Possible enhancements include correlation analysis, heatmaps, or predictive modeling to refine insights. This structured method helps prioritize cybersecurity defenses effectively.

## Task 2. Perform Exploratory Security Analysis

This is the list of R Listings and Expected Results:

- **Listing 4-0. Set R environment**



```
** data
*** moving datasets to lazyload DB
** inst
** byte-compile and prepare package for lazy loading
```

```
[16] !R -e "IRkernel::installspec(user = FALSE)"
```

```
R version 4.4.2 (2024-10-31) -- "Pile of Leaves"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> IRkernel::installspec(user = FALSE)
>
>
```

```
[17] %load_ext rpy2.ipython
```

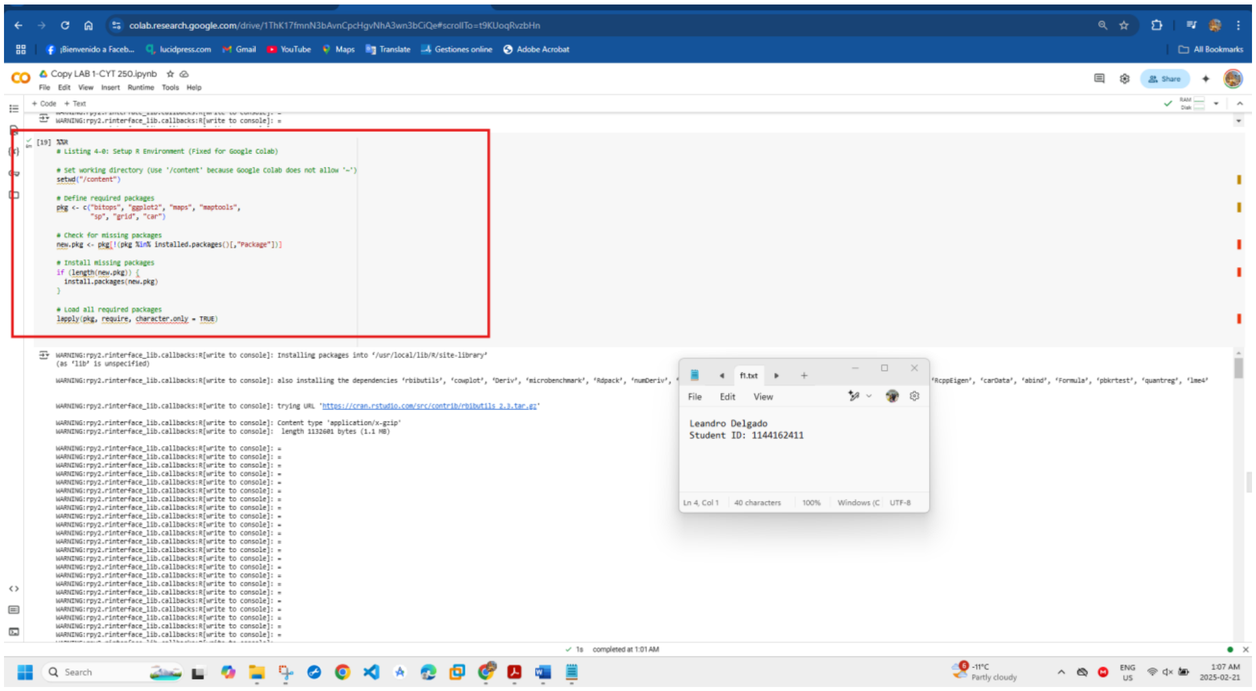
```
The rpy2.ipython extension is already loaded. To reload it, use:
  %reload_ext rpy2.ipython
```

```
[18] %%R
install.packages(c("iptools", "rgeolocate", "ggplot2", "maps"))
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Installing packages into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/ggplot2_3.5.1.tar.gz'

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Content type 'application/x-gzip'
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:  length 3604371 bytes (3.4 MB)

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
```

```
Leandro Delgado
Student ID: 1144162411
```

colab.research.google.com/drive/1ThK17fmnN3bAvnCpcHgvNhA3wn3bCiQe#scrollTo=D3zZyWBCzMER
Copy LAB 1-CYT 250.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
f1.txt
Ln 4, Col 1 | 40 characters | 100% | Windows (C | UTF-8
1s completed at 1:01 AM
-11°C Partly cloudy
1:06 AM 2025-02-21



colab.research.google.com/drive/1ThK17fmnN3bAvnCpcHgvNhA3wn3bCiQe#scrollTo=t9KUoqRvzbHn

Copy LAB 1-CYT 250.ipynb

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
```

```
[19] %%R
# Listing 4-0: Setup R Environment (Fixed for Google Colab)

# Set working directory (Use '/content' because Google Colab does not allow '~')
setwd("/content")

# Define required packages
pkg <- c("bitops", "ggplot2", "maps", "maptools",
         "sp", "grid", "car")

# Check for missing packages
new.pkg <- pkg[!(pkg %in% installed.packages()[,"Package"])]

# Install missing packages
if (length(new.pkg)) {
  install.packages(new.pkg)
}

# Load all required packages
lapply(pkg, require, character.only = TRUE)
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Installing packages into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: also installing the dependencies 'rbibutils', 'cowplot', 'Deriv', 'microbenchmark', 'Rdpack', 'numDeriv', '                    'RcppEigen', 'carData', 'abind', 'Formula', 'pbkrtest', 'quantreg', 'lme4'


WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/rbibutils_2.3.tar.gz'

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Content type 'application/x-gzip'
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:  length 1132601 bytes (1.1 MB)

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
```

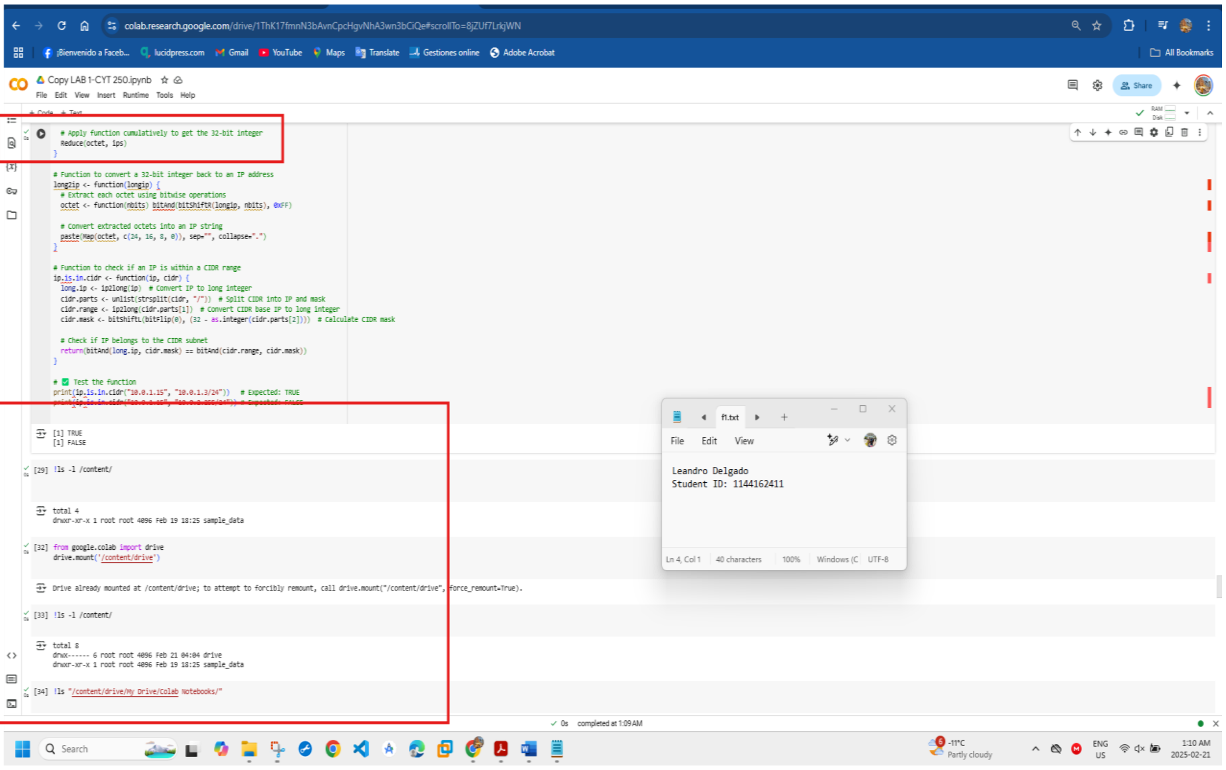
f1.txt

```
Leandro Delgado
Student ID: 1144162411
```

Ln 4, Col 1 | 40 characters | 100% | Windows (C | UTF-8

✓ 1s completed at 1:01 AM

-11°C Partly cloudy ENG US 1:07 AM 2025-02-21



colab.research.google.com/drive/1ThK17fmnN3bAvnCpcHgvNhA3wn3bCiQe#scrollTo=8jZUf7LrkjWN

Copy LAB 1-CYT 250.ipynb

```
  # Apply function cumulatively to get the 32-bit integer
  Reduce(octet, ips)
}

# Function to convert a 32-bit integer back to an IP address
long2ip <- function(longip) {
  # Extract each octet using bitwise operations
  octet <- function(nbits) bitAnd(bitShiftR(longip, nbits), 0xFF)

  # Convert extracted octets into an IP string
  paste(Map(octet, c(24, 16, 8, 0)), sep="", collapse=".")
}

# Function to check if an IP is within a CIDR range
ip.is.in.cidr <- function(ip, cidr) {
  long.ip <- ip2long(ip)  # Convert IP to long integer
  cidr.parts <- unlist(strsplit(cidr, "/"))  # Split CIDR into IP and mask
  cidr.range <- ip2long(cidr.parts[1])  # Convert CIDR base IP to long integer
  cidr.mask <- bitShiftL(bitFlip(0), (32 - as.integer(cidr.parts[2])))  # Calculate CIDR mask

  # Check if IP belongs to the CIDR subnet
  return(bitAnd(long.ip, cidr.mask) == bitAnd(cidr.range, cidr.mask))
}

# ✅ Test the function
print(ip.is.in.cidr("10.0.1.15", "10.0.1.3/24"))   # Expected: TRUE
```

```
[1] TRUE
[1] FALSE
```

```
[29] !ls -l /content/
```

```
total 4
drwxr-xr-x 1 root root 4096 Feb 19 18:25 sample_data
```

```
[32] from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[33] !ls -l /content/
```

```
total 8
drwx------ 6 root root 4096 Feb 21 04:04 drive
drwxr-xr-x 1 root root 4096 Feb 19 18:25 sample_data
```

```
[34] !ls "/content/drive/My Drive/Colab Notebooks/"
```

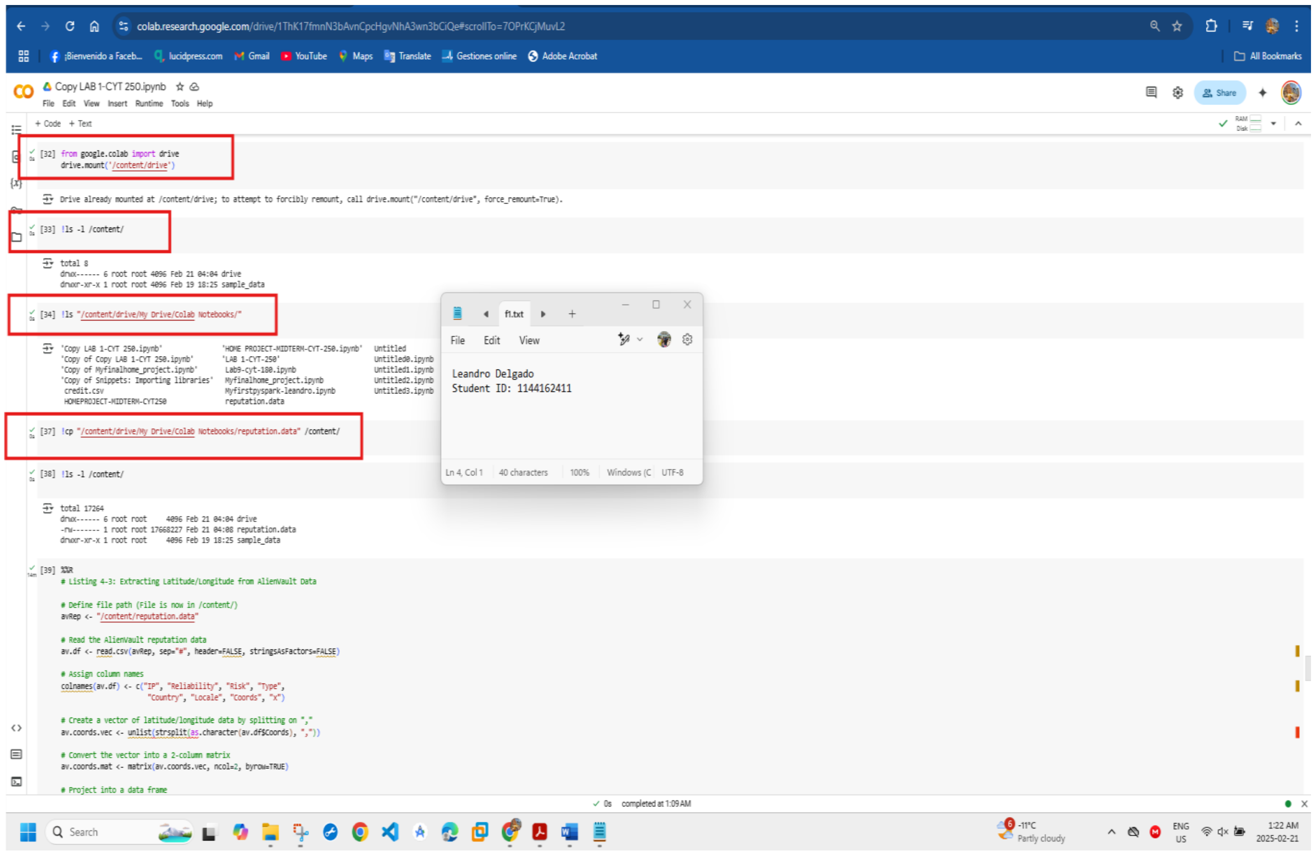
f1.txt

```
Leandro Delgado
Student ID: 1144162411
```

Ln 4, Col 1 | 40 characters | 100% | Windows (C | UTF-8

✓ 0s completed at 1:09 AM

-11°C Partly cloudy ENG US 1:10 AM 2025-02-21



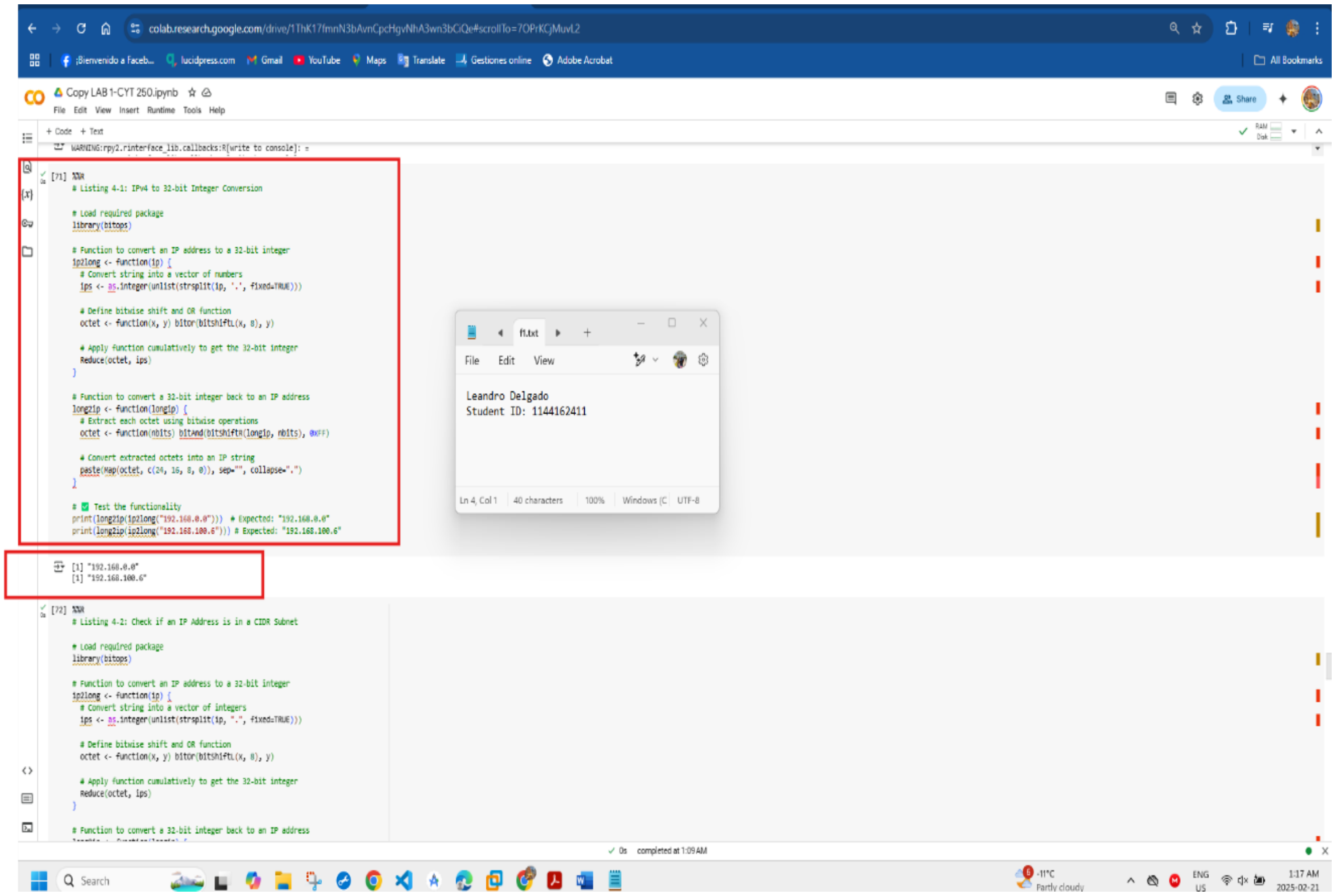
```
[32] from google.colab import drive
     drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[33] !ls -l /content/

total 8
drwx------ 6 root root 4096 Feb 21 04:04 drive
drwxr-xr-x 1 root root 4096 Feb 19 18:25 sample_data

[34] !ls "/content/drive/My Drive/Colab Notebooks/"

'Copy LAB 1-CYT 250.ipynb'                 'HOME PROJECT-MIDTERM-CYT-250.ipynb'   Untitled
'Copy of Copy LAB 1-CYT 250.ipynb'         'LAB 1-CYT-250'                        Untitled0.ipynb
'Copy of Myfinalhome_project.ipynb'         Lab9-cyt-180.ipynb                    Untitled1.ipynb
'Copy of Snippets: Importing libraries'     Myfinalhome_project.ipynb             Untitled2.ipynb
 credit.csv                                 Myfirstpyspark-leandro.ipynb          Untitled3.ipynb
 HOMEPROJECT-MIDTERM-CYT250                 reputation.data

[37] !cp "/content/drive/My Drive/Colab Notebooks/reputation.data" /content/

[38] !ls -l /content/

total 17264
drwx------ 6 root root     4096 Feb 21 04:04 drive
-rw------- 1 root root 17668227 Feb 21 04:08 reputation.data
drwxr-xr-x 1 root root     4096 Feb 19 18:25 sample_data

[39] %%R
     # Listing 4-3: Extracting Latitude/Longitude from AlienVault Data

     # Define file path (File is now in /content/)
     avRep <- "/content/reputation.data"

     # Read the AlienVault reputation data
     av.df <- read.csv(avRep, sep="#", header=FALSE, stringsAsFactors=FALSE)

     # Assign column names
     colnames(av.df) <- c("IP", "Reliability", "Risk", "Type",
                          "Country", "Locale", "Coords", "x")

     # Create a vector of latitude/longitude data by splitting on ","
     av.coords.vec <- unlist(strsplit(as.character(av.df$Coords), ","))

     # Convert the vector into a 2-column matrix
     av.coords.mat <- matrix(av.coords.vec, ncol=2, byrow=TRUE)

     # Project into a data frame
```

```
Leandro Delgado
Student ID: 1144162411
```

colab.research.google.com/drive/1ThK17fmnN3bAvnCpcHgvNhA3wn3bCiQe#scrollTo=7OPrKCjMuvL2
Copy LAB 1-CYT 250.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
f1.txt
Ln 4, Col 1 40 characters 100% Windows (C UTF-8
0s completed at 1:09 AM
-11°C Partly cloudy
1:22 AM 2025-02-21

The establishment of the Google Colab R environment facilitates adequacy in the tooling and dependencies for the running of R scripts successfully. Since Collab does not permit the typical convention using stewed to set working directories, the script makes it content for purposes of efficient file management. The next step involves defining a list of required R packages, which includes a few examples like stringr, ggplot2, maps, and map tools that everyone applies mostly in data manipulation and visualization. For adding errors caused due to unattended dependencies, the script checks if each of the necessary packages was previously installed by cross-reference with the installed packages list. If a package is not available, it is to be applied to a separate list source, which will then be installed using installation. Packages. When all the above packages are installed into the machine, the script calls them through library and makes them available for further use in subsequent codes. The arrangement creates a structure that ensures the running of R scripts is not interrupted while working in Google Colab, creating an R environment that is fully functional and ready for data manipulation and visualization. After all these processes my colab environment got ready to proceed with the next codes to be run.

- **Listing 4-1, Converting IP addresses to/from 32-bit integers.**



```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
```

```
[71] %%R
# Listing 4-1: IPv4 to 32-bit Integer Conversion

# Load required package
library(bitops)

# Function to convert an IP address to a 32-bit integer
ip2long <- function(ip) {
  # Convert string into a vector of numbers
  ips <- as.integer(unlist(strsplit(ip, '.', fixed=TRUE)))

  # Define bitwise shift and OR function
  octet <- function(x, y) bitor(bitshiftL(x, 8), y)

  # Apply function cumulatively to get the 32-bit integer
  Reduce(octet, ips)
}

# Function to convert a 32-bit integer back to an IP address
long2ip <- function(longip) {
  # Extract each octet using bitwise operations
  octet <- function(nbits) bitAnd(bitshiftR(longip, nbits), 0xFF)

  # Convert extracted octets into an IP string
  paste(Map(octet, c(24, 16, 8, 0)), sep="", collapse=".")
}

# ✅ Test the functionality
print(long2ip(ip2long("192.168.0.0")))  # Expected: "192.168.0.0"
print(long2ip(ip2long("192.168.100.6"))) # Expected: "192.168.100.6"
```

```
[1] "192.168.0.0"
[1] "192.168.100.6"
```

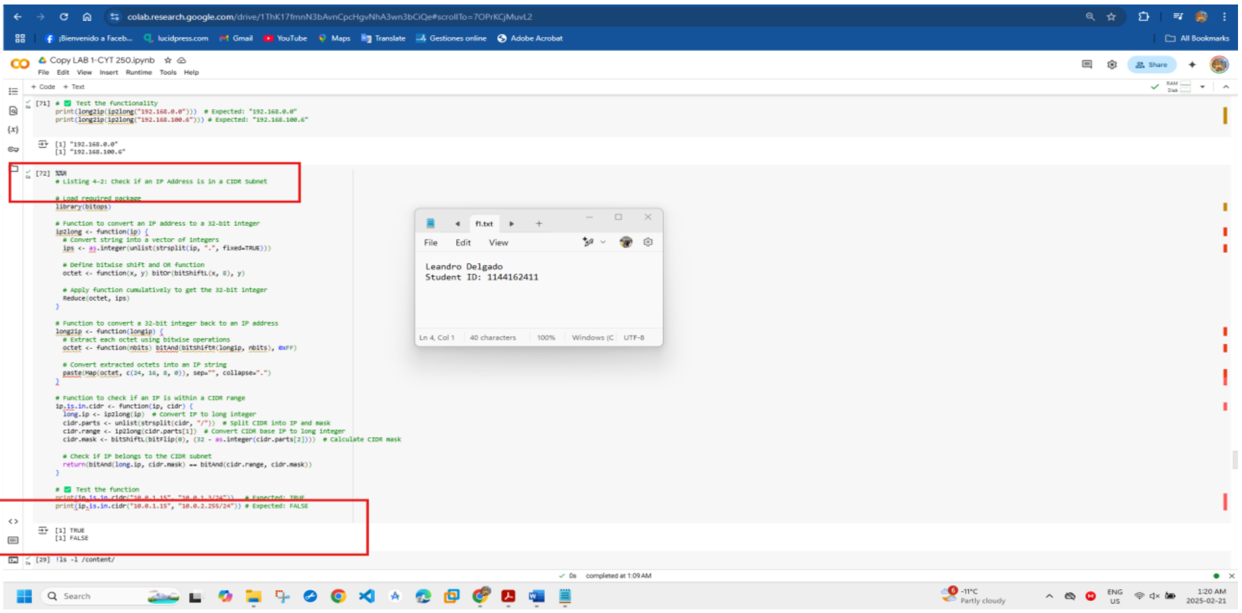
```
[72] %%R
# Listing 4-2: Check if an IP Address is in a CIDR Subnet

# Load required package
library(bitops)

# Function to convert an IP address to a 32-bit integer
ip2long <- function(ip) {
  # Convert string into a vector of integers
  ips <- as.integer(unlist(strsplit(ip, ".", fixed=TRUE)))

  # Define bitwise shift and OR function
  octet <- function(x, y) bitOr(bitShiftL(x, 8), y)

  # Apply function cumulatively to get the 32-bit integer
  Reduce(octet, ips)
}

# Function to convert a 32-bit integer back to an IP address
```

```
f1.txt
Leandro Delgado
Student ID: 1144162411
```

colab.research.google.com/drive/1ThK17fmnN3bAvnCpcHgvNhA3wn3bCiQe#scrollTo=7OPrKCjMuvL2
Copy LAB 1-CYT 250.ipynb
File Edit View Insert Runtime Tools Help
Ln 4, Col 1 | 40 characters | 100% | Windows (C | UTF-8
completed at 1:09 AM
1:17 AM 2025-02-21

This R script exemplifies how the conversion of an IPv4 address to a 32-bit integer and back is accomplished by utilizing bitwise operations. The conversion is important for fast IP storage and processing in networking applications. The script first loads the bitops package to enable bitwise operations. The function ip2long breaks down the IPv4 address into four octets, converts them into integers, and performs bitwise shift bitShiftL and OR bitor operations to yield a single 32-bit integer. The opposite function, long2ip, takes the 32-bit integer returned by ip2long and use bitwise AND bitAnd and bit shifting to extract each octet, which then assembles back into standard IPv4 format notation. In "the script," both functions convert "192.168.0.0" and "192.168.100.6," first into an integer and back again, confirming that the transformation was indeed correct. Because the test results have been found successful, it substantiates the lossless conversion, thus making this tool quite efficient for sorting, comparing, or even storing IP addresses numerically for application in cybersecurity and networking.

- **Listing 4-2, Testing IP address membership in CIDR block.**



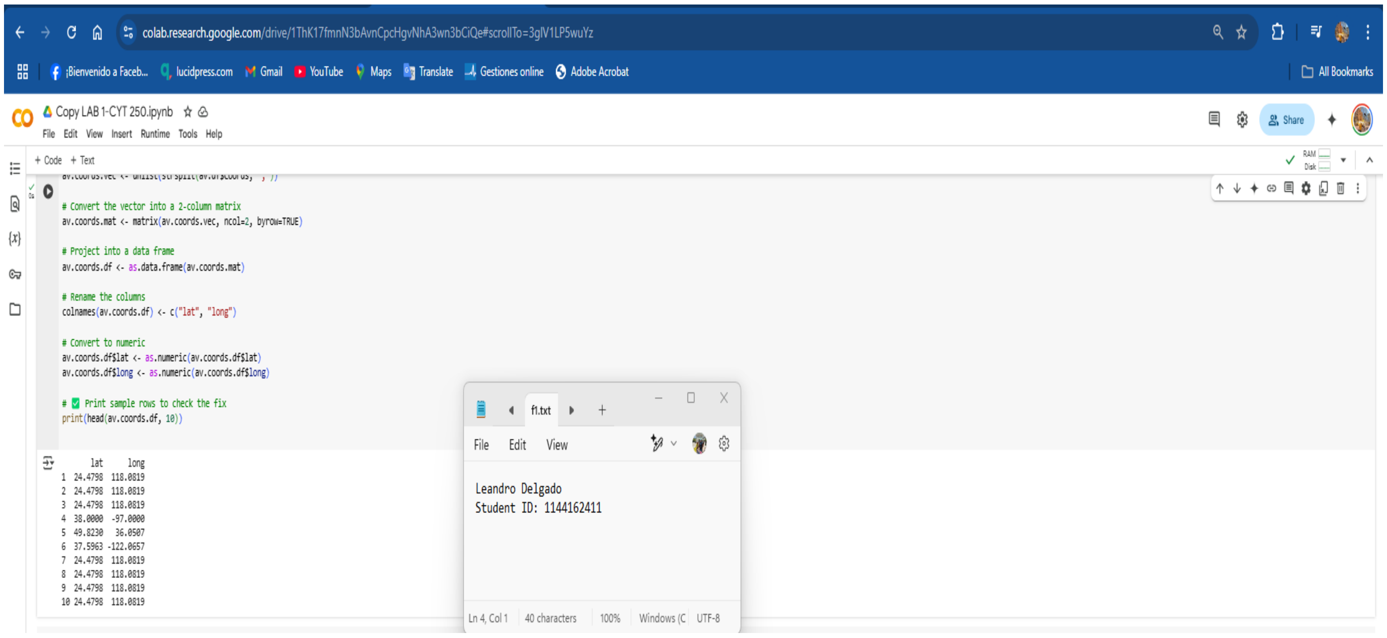
```
[71] # ✅ Test the functionality
     print(long2ip(ip2long("192.168.0.0")))  # Expected: "192.168.0.0"
     print(long2ip(ip2long("192.168.100.6"))) # Expected: "192.168.100.6"

[1] "192.168.0.0"
[1] "192.168.100.6"

[72] %%R
     # Listing 4-2: Check if an IP Address is in a CIDR Subnet

     # Load required package
     library(bitops)

     # Function to convert an IP address to a 32-bit integer
     ip2long <- function(ip) {
       # Convert string into a vector of integers
       ips <- as.integer(unlist(strsplit(ip, ".", fixed=TRUE)))

       # Define bitwise shift and OR function
       octet <- function(x, y) bitOr(bitShiftL(x, 8), y)

       # Apply function cumulatively to get the 32-bit integer
       Reduce(octet, ips)
     }

     # Function to convert a 32-bit integer back to an IP address
     long2ip <- function(longip) {
       # Extract each octet using bitwise operations
       octet <- function(nbits) bitAnd(bitShiftR(longip, nbits), 0xFF)

       # Convert extracted octets into an IP string
       paste(Map(octet, c(24, 16, 8, 0)), sep="", collapse=".")
     }

     # Function to check if an IP is within a CIDR range
     ip.is.in.cidr <- function(ip, cidr) {
       long.ip <- ip2long(ip)  # Convert IP to long integer
       cidr.parts <- unlist(strsplit(cidr, "/"))  # Split CIDR into IP and mask
       cidr.range <- ip2long(cidr.parts[1])  # Convert CIDR base IP to long integer
       cidr.mask <- bitShiftL(bitFlip(0), (32 - as.integer(cidr.parts[2])))  # Calculate CIDR mask

       # Check if IP belongs to the CIDR subnet
       return(bitAnd(long.ip, cidr.mask) == bitAnd(cidr.range, cidr.mask))
     }

     # ✅ Test the function
     print(ip.is.in.cidr("10.0.1.15", "10.0.1.3/24"))   # Expected: TRUE
     print(ip.is.in.cidr("10.0.1.15", "10.0.2.255/24")) # Expected: FALSE

[1] TRUE
[1] FALSE

[29] !ls -l /content/
```

```
Leandro Delgado
Student ID: 1144162411
```

The code presented in the picture determines whether any IP is part of an explicit CIDR subnet through bitwise operation in R. The process involves converting an IPv4 address into a 32-bit integer ip2long by taking all its numbers and shifting them according to the bitwise or operations between each one. The other way around through the reverse function long2ip recreates the original IP per octet by applying bitwise and operations. The next function ip.is.in.cidr verifies whether an IP belongs within a particular subnet by taking the CIDR notation base IP, pulling out a subnet mask, converting it into integers, and finally checking to see if the interested IP lies in the corresponding masked range. The output shows that 10.0.1.15 is TRUE in the sense that it belongs or is within the 10.0.1.2/24 subnet, which is FALSE in the case of being in 10.0.2.255/24. This method is very helpful in helping deal with general aspects of network security, as well as filtering in firewalls and access control where subnet validation is required.

**Listing 4-3, Locating IP addresses.**

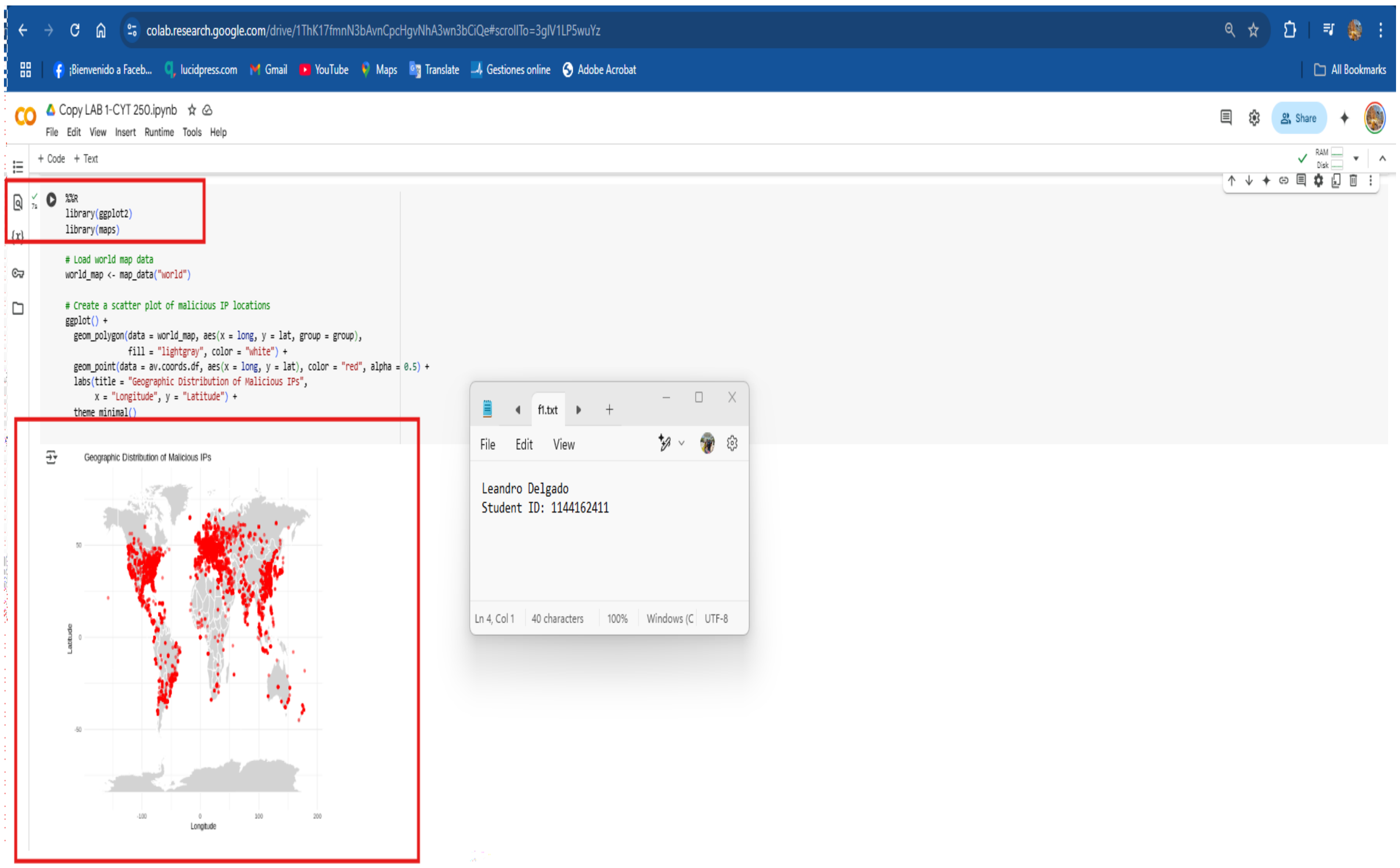
The output shows the transformation of geographic coordinate data to the success level of a structured format. Starting with data manipulation in matrix form of two columns and then projecting it to a data frame, the names of the columns were then given appropriate names, that is "lat" (latitude) and "long" (longitude) to be more descriptive. Further, both columns were cast to numeric types to avoid a variation in processing the data. The last step involved printing a sample of ten rows to check if the conversion went smoothly. The output comprises latitude and longitude values showing most entries as 24.4798, 118.0819 while some records have different coordinates such as 38.0000, -97.0000 and 49.8230, 36.0507. It further implies that the data have undergone transformation and processing for any onward geospatial analysis. Using R's ggplot2, the global dispersion of malicious IP addresses is mapped. A world map with geom_polygon as the underlying theme and malicious IPs as red points on top, along with geom_point where transparency is attributed for visualization's clarity, was created. This breaks up the visualization into various hotspots of high risk, primarily within North America, Europe, and Asia, to assist cyber security professionals in pattern recognition and proactive mitigation.



```
# Convert the vector into a 2-column matrix
av.coords.mat <- matrix(av.coords.vec, ncol=2, byrow=TRUE)

# Project into a data frame
av.coords.df <- as.data.frame(av.coords.mat)

# Rename the columns
colnames(av.coords.df) <- c("lat", "long")

# Convert to numeric
av.coords.df$lat <- as.numeric(av.coords.df$lat)
av.coords.df$long <- as.numeric(av.coords.df$long)

# ✅ Print sample rows to check the fix
print(head(av.coords.df, 10))
```

```
       lat      long
1  24.4798  118.0819
2  24.4798  118.0819
3  24.4798  118.0819
4  38.0000  -97.0000
5  49.8230   36.0507
6  37.5963 -122.0657
7  24.4798  118.0819
8  24.4798  118.0819
9  24.4798  118.0819
10 24.4798  118.0819
```

```
Leandro Delgado
Student ID: 1144162411
```

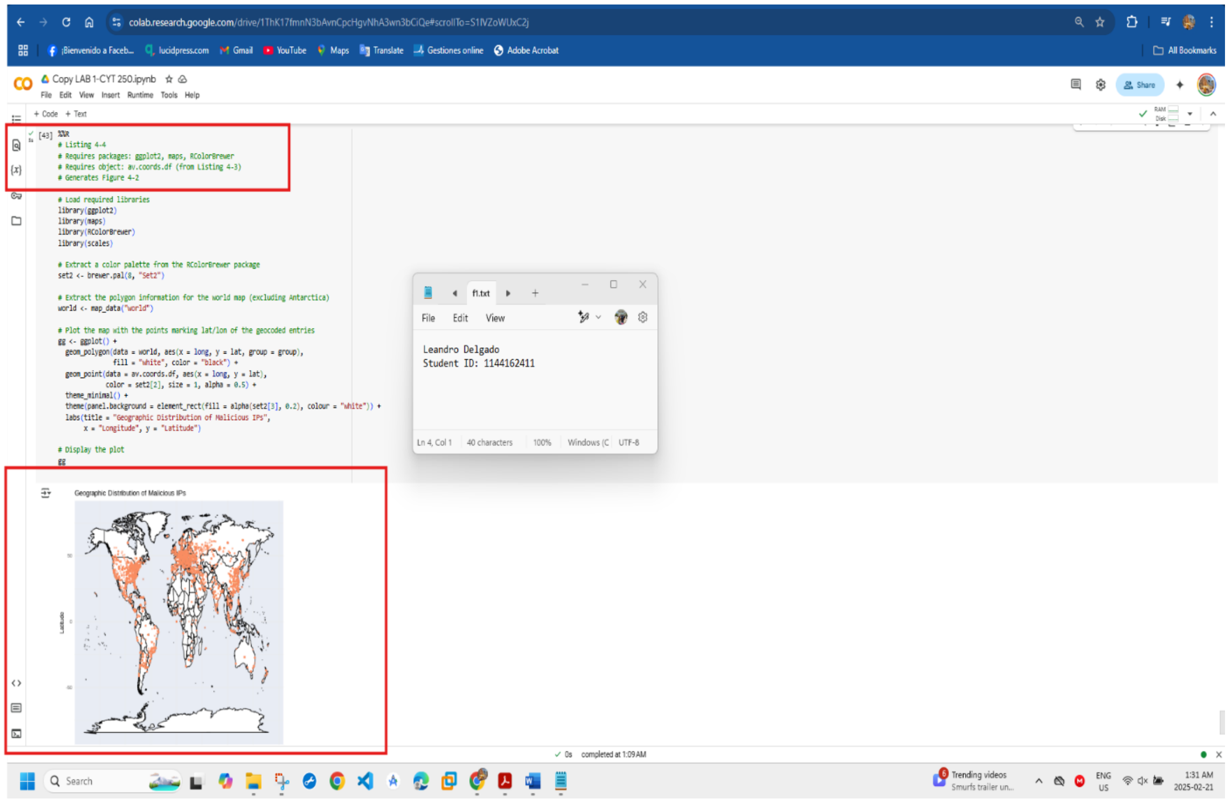


```
%%R
library(ggplot2)
library(maps)

# Load world map data
world_map <- map_data("world")

# Create a scatter plot of malicious IP locations
ggplot() +
  geom_polygon(data = world_map, aes(x = long, y = lat, group = group),
               fill = "lightgray", color = "white") +
  geom_point(data = av.coords.df, aes(x = long, y = lat), color = "red", alpha = 0.5) +
  labs(title = "Geographic Distribution of Malicious IPs",
       x = "Longitude", y = "Latitude") +
  theme_minimal()
```

```
Leandro Delgado
Student ID: 1144162411
```

**Listing 4-4, Locating IP addresses. Figure 4-2, comments**

The image shows the R script running in Google Colab which produces a map showing the geographical distribution of malicious IP addresses. This script uses ggplot2, maps, and RColorBrewer for the visualization. At the top of the image is the code that loads world map data, applies the color palette, and plots the map with the locations of malicious IPs represented by orange dots. The resultant map is displayed below, where clusters of malicious IPs can be seen, particularly in North America, Europe, and parts of Asia, indicating high-risk zones for cybersecurity-related threats. This project was a big determinant for understanding the importance of employing IP reputation data for improved monitoring of cybersecurity threats in this era. The learning process involved automation and visualization using Python and the threat map geographic location using R, to analyze risk, prioritize threats, and enhance SIEM/IDS integration. Thus, the analysis aptly displayed the usefulness of data-driven security decisions and proactive defense strategies. Lastly, the interplay of risk and reliability gave insight into filtering and prioritizing threat data. My hands-on experience with Python and R has shown a prevalent significance of automation and visualization in cybersecurity analysis, thus enhancing threat detection and response strategies.


colab.research.google.com/drive/1ThK17fmnN3bAvnCpcHgvNhA3wn3bCiQe#scrollTo=S1lVZoWUxC2j
Copy LAB 1-CYT 250.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Geographic Distribution of Malicious IPs
f1.txt
File Edit View
Ln 4, Col 1 40 characters 100% Windows (C UTF-8

```
[43] %%R
     # Listing 4-4
     # Requires packages: ggplot2, maps, RColorBrewer
     # Requires object: av.coords.df (from Listing 4-3)
     # Generates Figure 4-2

     # Load required libraries
     library(ggplot2)
     library(maps)
     library(RColorBrewer)
     library(scales)

     # Extract a color palette from the RColorBrewer package
     set2 <- brewer.pal(8, "Set2")

     # Extract the polygon information for the world map (excluding Antarctica)
     world <- map_data("world")

     # Plot the map with the points marking lat/lon of the geocoded entries
     gg <- ggplot() +
       geom_polygon(data = world, aes(x = long, y = lat, group = group),
                    fill = "white", color = "black") +
       geom_point(data = av.coords.df, aes(x = long, y = lat),
                  color = set2[2], size = 1, alpha = 0.5) +
       theme_minimal() +
       theme(panel.background = element_rect(fill = alpha(set2[3], 0.2), colour = "white")) +
       labs(title = "Geographic Distribution of Malicious IPs",
            x = "Longitude", y = "Latitude")

     # Display the plot
     gg
```

```
Leandro Delgado
Student ID: 1144162411
```

## Resume of this Home Project

This project was a big determinant for understanding the importance of employing IP reputation data for improved monitoring of cybersecurity threats in this era. The learning process involved automation and visualization using Colab and the threat map geographic location using R, to analyze risk, prioritize threats, and enhance SIEM/IDS integration. Thus, the analysis aptly displayed the usefulness of data-driven security decisions and proactive defense strategies. Lastly, the interplay of risk and reliability gave insight into filtering and prioritizing threat data. My hands-on experience with Colab and R has shown a prevalent significance of automation and visualization in cybersecurity analysis, thus enhancing threat detection and response strategies.