Name: Yutong Du

Date: 2023-11-25

Email: Yutong.du@aalto.fi

Student ID: 101481573

Task 1: Capturing data

* **Acquiring packet capture data**

1. **What kind of trace file and tool/s you are using to perform the packet capture?**

**Wireshark**

1. **Date, time, duration, measurement setting (in terms of profile if you are using the Wireshark) or file name if you are using some public traces.**

**25/11/2023: From 17:31 to 19:31. Measurement setting: Default measurement setting of Wireshark. File name: final\_a.pcap.**

* **Provide a short sample (10 lines or so) of the data taken from your capture file.**

**Wireshark) or file name if you are using the some public traces.**

|  |
| --- |
|  |

* **Packet data PS1**

**1.1: Visualise packet distribution by port numbers.**

**Code:**

|  |
| --- |
| import pyshark import matplotlib.pyplot as plt from collections import Counter from operator import itemgetter  tshark\_path = 'D:\\0x00\_Softwares\\Wireshark\\tshark.exe' file\_path = 'files/final\_a.pcap'  cap = pyshark.FileCapture(file\_path, tshark\_path=tshark\_path, keep\_packets=True)  port\_counts = Counter() for packet in cap:  if 'TCP' in packet or 'UDP' in packet:  layer = packet.tcp if 'TCP' in packet else packet.udp  port\_counts[layer.dstport] += 1  cap.close()  sorted\_port\_counts = dict(sorted(port\_counts.items(), key=itemgetter(1), reverse=True)[:20])  plt.figure(figsize=(10, 10)) plt.pie(sorted\_port\_counts.values(), labels=sorted\_port\_counts.keys(), autopct='%1.1f%%', startangle=140) plt.title('Top 20 Packet Distribution by Port Numbers') plt.show() |

**Packet distribution:**

|  |
| --- |
|  |

**1.2: Plot traffic volume as a function of time with at least two sufficiently different time scales.**

**Exported the part1.pcap as part1.csv.**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt  csv\_file\_path = 'files/final\_a.csv'  df = pd.read\_csv(csv\_file\_path)  df['Time'] = pd.to\_datetime(df['Time'], unit='s')  traffic\_volume = df.groupby('Time')['Length'].sum().reset\_index()  traffic\_per\_second = traffic\_volume.set\_index('Time').resample('S').sum().fillna(0) traffic\_per\_minute = traffic\_volume.set\_index('Time').resample('T').sum().fillna(0)  plt.figure(figsize=(14, 7))  plt.subplot(1, 2, 1) plt.plot(traffic\_per\_second.index, traffic\_per\_second['Length'], linestyle='-') plt.title('Traffic Volume per Second') plt.xlabel('Time (second)') plt.ylabel('Traffic Volume (bytes)') plt.xticks(rotation=45)  plt.subplot(1, 2, 2) plt.plot(traffic\_per\_minute.index, traffic\_per\_minute['Length'], marker='o', linestyle='-', color='orange') plt.title('Traffic Volume per Minute') plt.xlabel('Time (minute)') plt.ylabel('Traffic Volume (bytes)') plt.xticks(rotation=45)  plt.tight\_layout() plt.show() |

**Result:**

|  |
| --- |
|  |

**1.3: Plot packet length distribution (use bins of width 1 byte), its empirical cumulative distribution function and key summary statistics.**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import numpy as np  csv\_file\_path = 'files/final\_a.csv'  df = pd.read\_csv(csv\_file\_path)  plt.figure(figsize=(14, 7))  plt.subplot(1, 2, 1) bin\_width = 100 bins = range(min(df['Length']), max(df['Length']) + bin\_width, bin\_width) plt.hist(df['Length'], bins=bins, color='blue', alpha=0.7, log=True) plt.title('Packet Length Distribution (Log Scale)') plt.xlabel('Packet Length (bytes)') plt.ylabel('Frequency (Log Scale)')  plt.subplot(1, 2, 2) sorted\_length = np.sort(df['Length']) yvals = np.arange(1, len(sorted\_length) + 1) / len(sorted\_length) plt.plot(sorted\_length, yvals, marker='.', linestyle='none') plt.title('Empirical Cumulative Distribution Function (ECDF)') plt.xlabel('Packet Length (bytes)') plt.ylabel('ECDF')  plt.tight\_layout() plt.show()  print("Summary statistics for packet lengths:") print(df['Length'].describe()) |

**Result:**

|  |
| --- |
|  |

**Key summary statistics:**

|  |
| --- |
| **Summary statistics for packet lengths:**  count 606080.000000  mean 1021.099314  std 1893.895590  min 42.000000  25% 93.000000  50% 1292.000000  75% 1292.000000  max 65226.000000 |

* **Flow data PS2**

**1.4: Visualise flow distribution by port numbers.**

**Use the command below to convert part2.cap to flow data:**

|  |
| --- |
| **tshark -r final\_a.pcap -q -z conv,tcp > final\_a.txt** |

**Code:**

|  |
| --- |
| import pandas as pd import numpy as np import matplotlib.pyplot as plt from matplotlib.lines import Line2D  def convert\_to\_byte(row):  units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}   try:  ld\_bytes\_unit = str(row['ld\_bytes\_unit']).lower()  factor = units[ld\_bytes\_unit]  ld\_kb = row['ld\_bytes'] \* factor   rd\_bytes\_unit = str(row['rd\_bytes\_unit']).lower()  factor = units[rd\_bytes\_unit]  rd\_kb = row['rd\_bytes'] \* factor   total\_bytes\_unit = str(row['total\_bytes\_unit']).lower()  factor = units[total\_bytes\_unit]  total\_kb = row['total\_bytes'] \* factor   return pd.Series({'ld\_bytes': ld\_kb, 'rd\_bytes': rd\_kb, 'total\_bytes': total\_kb, 'server\_ip': row['second\_ip\_interface']})  except KeyError as e:  print(f"Error processing row {row}: {e}")  raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  df = pd.read\_csv('files/final\_a.txt', sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  new\_column\_names = ["first\_ip\_interface", "arrow", "second\_ip\_interface", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  "start", "duration"]  df.columns = new\_column\_names  pd.set\_option('display.max\_columns', None)  df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  df['port'] = df['second\_ip\_interface'].str.split(':').str[1].astype(str)  port\_flow\_count = df.groupby('port').size().reset\_index(name='count') port\_flow\_count = port\_flow\_count.sort\_values(by='count', ascending=False)  plt.figure(figsize=(10, 6)) bar\_plot = plt.bar(port\_flow\_count['port'], port\_flow\_count['count'])  for bar in bar\_plot:  yval = bar.get\_height()  plt.text(bar.get\_x() + bar.get\_width() / 2, yval, int(yval), va='bottom', ha='center')  plt.xlabel('Port Number') plt.ylabel('Number of Flows') plt.title('Flow Count Distribution by Port Numbers (Descending)') plt.xticks(rotation=45) plt.show() |

**Flow distribution:**

|  |
| --- |
|  |

**1.5: Plot traffic volume as a function of time with at least two sufficiently different time scales.**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import re import datetime  # Function to parse each line of the data def parse\_line(line):  match = re.search(r'(\d+\.\d+\.\d+\.\d+:\d+)\s+<->\s+(\d+\.\d+\.\d+\.\d+:\d+)\s+(\d+)\s+(\d+\s+\w+)\s+(\d+)\s+(\d+\s+\w+)\s+(\d+)\s+(\d+\s+\w+)\s+(\d+\.\d+)', line)  if match:  return {  "Source\_IP": match.group(1),  "Destination\_IP": match.group(2),  "Upload\_Frames": int(match.group(3)),  "Upload\_Bytes": match.group(4),  "Download\_Frames": int(match.group(5)),  "Download\_Bytes": match.group(6),  "Total\_Frames": int(match.group(7)),  "Total\_Bytes": match.group(8),  "Start": float(match.group(9))  }  else:  return None  # Function to convert the total bytes to a uniform unit (bytes) def convert\_bytes(byte\_str):  number, unit = byte\_str.split()  number = float(number)  unit = unit.lower()  if unit == 'kb':  return number \* 1024  elif unit == 'mb':  return number \* 1024 \* 1024  elif unit == 'gb':  return number \* 1024 \* 1024 \* 1024  else:  return number  # Parsing the entire file file\_path = 'files/final\_a.txt' # Replace with your file path parsed\_full\_data = [] with open(file\_path, 'r') as file:  for \_ in range(5): # Skipping the first five lines  next(file)  for line in file: # Parsing each line in the file  parsed\_line = parse\_line(line)  if parsed\_line:  parsed\_line["Total\_Bytes"] = convert\_bytes(parsed\_line["Total\_Bytes"])  parsed\_full\_data.append(parsed\_line)  # Convert the parsed data to a DataFrame full\_df = pd.DataFrame(parsed\_full\_data)  # Convert the 'Start' column to a datetime format using a reference date reference\_date = datetime.datetime(1970, 1, 1) full\_df['Start'] = pd.to\_datetime(full\_df['Start'], unit='s', origin=reference\_date)  # Resampling data to different time scales # 1. Resampling to seconds df\_seconds = full\_df.resample('1S', on='Start').sum()  # 2. Resampling to minutes df\_minutes = full\_df.resample('1T', on='Start').sum()  # Plotting the data fig, axs = plt.subplots(2, 1, figsize=(12, 10))  # Plot for second-wise data axs[0].plot(df\_seconds.index, df\_seconds['Total\_Bytes']) axs[0].set\_title('Traffic Volume per Second') axs[0].set\_xlabel('Time') axs[0].set\_ylabel('Total Bytes') axs[0].grid(True)  # Plot for minute-wise data axs[1].plot(df\_minutes.index, df\_minutes['Total\_Bytes'], marker='x',color='orange') axs[1].set\_title('Traffic Volume per Minute') axs[1].set\_xlabel('Time') axs[1].set\_ylabel('Total Bytes') axs[1].grid(True)  # Adjust layout and display the plot plt.tight\_layout() plt.show() |

**Result:**

|  |
| --- |
|  |

**1.6: Visualise flow distribution by country.**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt from geoip2.database import Reader  def convert\_to\_byte(row):  units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}   try:  ld\_bytes\_unit = str(row['ld\_bytes\_unit']).lower()  factor = units[ld\_bytes\_unit]  ld\_kb = row['ld\_bytes'] \* factor   rd\_bytes\_unit = str(row['rd\_bytes\_unit']).lower()  factor = units[rd\_bytes\_unit]  rd\_kb = row['rd\_bytes'] \* factor   total\_bytes\_unit = str(row['total\_bytes\_unit']).lower()  factor = units[total\_bytes\_unit]  total\_kb = row['total\_bytes'] \* factor   return pd.Series({'ld\_bytes': ld\_kb, 'rd\_bytes': rd\_kb, 'total\_bytes': total\_kb, 'server\_ip': row['second\_ip\_interface']})  except KeyError as e:  print(f"Error processing row {row}: {e}")  raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  df = pd.read\_csv('files/final\_a.txt', sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  new\_column\_names = ["first\_ip\_interface", "arrow", "second\_ip\_interface", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  "start", "duration"]  df.columns = new\_column\_names  pd.set\_option('display.max\_columns', None)  df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  geoip\_reader = Reader('others/GeoLite2-Country.mmdb')  def get\_country(ip):  try:  response = geoip\_reader.country(ip)  return response.country.name  except:  return "Unknown"  df['country'] = df['second\_ip\_interface'].str.split(':').str[0].apply(get\_country)  country\_traffic = df.groupby('country').size()  plt.figure(figsize=(12, 8)) country\_traffic.plot(kind='bar') plt.xlabel('Country') plt.ylabel('Total Traffic (Bytes)') plt.title('Flow Distribution by Country') plt.xticks(rotation=45)  # 添加数字标签 for i, v in enumerate(country\_traffic):  plt.text(i, v, str(v), ha='center', va='bottom', fontsize=8)  plt.show() |

**Result:**

|  |
| --- |
|  |

**1.7: Plot origin-destination pairs both by data volume and by flows (Zipf type plot).**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt from geoip2.database import Reader  def convert\_to\_byte(row):  units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}   try:  ld\_bytes\_unit = str(row['ld\_bytes\_unit']).lower()  factor = units[ld\_bytes\_unit]  ld\_kb = row['ld\_bytes'] \* factor   rd\_bytes\_unit = str(row['rd\_bytes\_unit']).lower()  factor = units[rd\_bytes\_unit]  rd\_kb = row['rd\_bytes'] \* factor   total\_bytes\_unit = str(row['total\_bytes\_unit']).lower()  factor = units[total\_bytes\_unit]  total\_kb = row['total\_bytes'] \* factor   return pd.Series({'ld\_bytes': ld\_kb, 'rd\_bytes': rd\_kb, 'total\_bytes': total\_kb, 'server\_ip': row['second\_ip\_interface']})  except KeyError as e:  print(f"Error processing row {row}: {e}")  raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  df = pd.read\_csv('files/final\_a.txt', sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  new\_column\_names = ["first\_ip\_interface", "arrow", "second\_ip\_interface", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  "start", "duration"]  df.columns = new\_column\_names  pd.set\_option('display.max\_columns', None)  df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  df['src\_dst\_pair'] = df['first\_ip\_interface'].str.split(':').str[0] + ' - ' + df['second\_ip\_interface'].str.split(':').str[0]  traffic\_data = df.groupby('src\_dst\_pair')['total\_bytes'].sum() flow\_counts = df.groupby('src\_dst\_pair').size()  sorted\_traffic\_data = traffic\_data.sort\_values(ascending=False) sorted\_flow\_counts = flow\_counts.sort\_values(ascending=False)  plt.figure(figsize=(10, 6)) plt.plot(sorted\_traffic\_data.values) plt.xlabel('Source-Destination Pairs') plt.ylabel('Total Data Volume (Bytes)') plt.title('Zipf Plot of Data Volume by Source-Destination Pairs') plt.yscale('log') plt.xscale('log') plt.show()  plt.figure(figsize=(10, 6)) plt.plot(sorted\_flow\_counts.values) plt.xlabel('Source-Destination Pairs') plt.ylabel('Number of Flows') plt.title('Zipf Plot of Flows by Source-Destination Pairs') plt.yscale('log') plt.xscale('log') plt.show() |

**Result:**

|  |
| --- |
|  |
|  |

**1.8: Plot flow length distribution, its empirical cumulative distribution function and key summary statistics.**

**Code:**

|  |
| --- |
| df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  # Plot flow length distribution (Histogram) plt.figure(figsize=(10, 6)) sns.histplot(df['total\_frames'], kde=False) plt.title('Flow Length Distribution (Total Frames)') plt.xlabel('Flow Length (Frames)') plt.ylabel('Frequency') plt.yscale('log') plt.show()  # Plot the ECDF plt.figure(figsize=(10, 6)) sns.ecdfplot(df['total\_frames']) plt.title('Empirical Cumulative Distribution Function (ECDF) of Flow Length') plt.xlabel('Flow Length (Frames)') plt.ylabel('ECDF') plt.grid(True) # Adding a grid for better readability plt.show()  # Display key summary statistics print("Key Summary Statistics (Total Frames):") print(df['total\_frames'].describe()) |

**Result:**

|  |
| --- |
|  |
|  |

**Key Summary Statistics (Total Frames):**

|  |
| --- |
| **count 995.000000**  **mean 40.890452**  **std 375.466692**  **min 1.000000**  **25% 11.000000**  **50% 12.000000**  **75% 24.000000**  **max 10616.000000** |

**1.9: Fit a distribution for the flow lengths and validate the model.**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt from distfit import distfit  def convert\_to\_byte(row):  units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}   try:  ld\_bytes\_unit = str(row['ld\_bytes\_unit']).lower()  factor = units[ld\_bytes\_unit]  ld\_kb = row['ld\_bytes'] \* factor   rd\_bytes\_unit = str(row['rd\_bytes\_unit']).lower()  factor = units[rd\_bytes\_unit]  rd\_kb = row['rd\_bytes'] \* factor   total\_bytes\_unit = str(row['total\_bytes\_unit']).lower()  factor = units[total\_bytes\_unit]  total\_kb = row['total\_bytes'] \* factor   return pd.Series({'ld\_bytes': ld\_kb, 'rd\_bytes': rd\_kb, 'total\_bytes': total\_kb, 'server\_ip': row['second\_ip\_interface']})  except KeyError as e:  print(f"Error processing row {row}: {e}")  raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  df = pd.read\_csv('files/part2.txt', sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  new\_column\_names = ["first\_ip\_interface", "arrow", "second\_ip\_interface", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  "start", "duration"]  df.columns = new\_column\_names  pd.set\_option('display.max\_columns', None)  df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  # Assuming df is your DataFrame and 'total\_frames' is the column with flow lengths data = df['total\_frames']   # Create a distfit object and fit it dist = distfit() dist.fit\_transform(data)  # Print the summary to see the results print(dist.summary) # Plot the fitted distribution dist.plot() plt.show() |

**Output:**

|  |
| --- |
| **name score loc scale \**  **0 genextreme 0.015843 12.103706 4.945311**  **1 dweibull 0.016562 11.0 13.534621**  **2 lognorm 0.020574 0.528073 16.274495**  **3 beta 0.026604 1.0 15453.774142**  **4 expon 0.0267 1.0 39.890452**  **5 pareto 0.026941 -68.606324 69.606324**  **6 t 0.03152 11.0 0.0**  **7 norm 0.033972 40.890452 375.277967**  **8 loggamma 0.034104 -159286.069997 21061.325873**  **9 uniform 0.034583 1.0 10615.0**  **10 gamma 0.034612 1.0 3.873971**  **arg \**  **0 (-0.24076588695122442,)**  **1 (0.6259545216499753,)**  **2 (0.7483495099312536,)**  **3 (0.8766237096548108, 465.67368151541666)**  **4 ()**  **5 (3.9559426249446146,)**  **6 (0.07154809459743079,)**  **7 ()**  **8 (1928.0662400216588,)**  **9 ()**  **10 (0.0011427786749111637,)**  **params \**  **0 (-0.24076588695122442, 12.103705824070051, 4.9...**  **1 (0.6259545216499753, 10.999999999999996, 13.53...**  **2 (0.7483495099312536, 0.5280725812790955, 16.27...**  **3 (0.8766237096548108, 465.67368151541666, 0.999...**  **4 (1.0, 39.890452261306535)**  **5 (3.9559426249446146, -68.60632418696868, 69.60...**  **6 (0.07154809459743079, 10.999999999643801, 6.47...**  **7 (40.890452261306535, 375.2779674100434)**  **8 (1928.0662400216588, -159286.06999677245, 2106...**  **9 (1.0, 10615.0)**  **10 (0.0011427786749111637, 0.9999999999999999, 3....**  **model bootstrap\_score \**  **0 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **1 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **2 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **3 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **4 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **5 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **6 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **7 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **8 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **9 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **10 <scipy.stats.\_distn\_infrastructure.rv\_continuo... 0**  **bootstrap\_pass color**  **0 None #e41a1c**  **1 None #e41a1c**  **2 None #377eb8**  **3 None #4daf4a**  **4 None #984ea3**  **5 None #ff7f00**  **6 None #ffff33**  **7 None #a65628**  **8 None #f781bf**  **9 None #999999**  **10 None #999999** |

**Validation with plots:**

|  |
| --- |
|  |

**1.10: Compare the number of flows with 1, 10, 60, 120 and 1800 second timeouts. In this, you need to generate flow data multiple times.**

**Code:**

|  |
| --- |
| from scapy.all import rdpcap, PacketList from datetime import datetime, timedelta  # 读取 PCAP 文件 packets = rdpcap('files/part2.pcap')  # 定义超时设置 timeouts = [1, 10, 60, 120, 1800] # 单位为秒  # 创建一个字典来存储不同超时设置下的流数量 flows\_for\_timeouts = {}  for timeout in timeouts:  # 初始化流列表和当前流的数据包列表  flows = []  current\_flow\_packets = []  last\_packet\_time = None   for packet in packets:  if 'IP' in packet and 'TCP' in packet:  # 获取当前数据包的时间戳  current\_packet\_time = datetime.fromtimestamp(float(packet.time))   # 检查是否是流的第一个数据包或者当前数据包与上一个数据包的时间差是否超过了超时设置  if last\_packet\_time is None or (current\_packet\_time - last\_packet\_time).total\_seconds() > timeout:  # 如果是新的流，先保存当前流，然后开始一个新的流  if current\_flow\_packets:  flows.append(PacketList(current\_flow\_packets))  current\_flow\_packets = [packet]  else:  # 否则，将数据包添加到当前流中  current\_flow\_packets.append(packet)   # 更新最后一个数据包的时间  last\_packet\_time = current\_packet\_time   # 保存最后一个流  if current\_flow\_packets:  flows.append(PacketList(current\_flow\_packets))   # 记录当前超时设置下的流数量  flows\_for\_timeouts[timeout] = len(flows)  for timeout, count in flows\_for\_timeouts.items():  print(f"Timeout: {timeout} seconds, Flow Count: {count}") |

**Result:**

|  |
| --- |
| **Timeout: 1 seconds, Flow Count: 575**  **Timeout: 10 seconds, Flow Count: 11**  **Timeout: 60 seconds, Flow Count: 1**  **Timeout: 120 seconds, Flow Count: 1**  **Timeout: 1800 seconds, Flow Count: 1** |

* **TCP connection data PS3**

**1.11: Round-trip times and their variance.**

**Use the command for converting .pcap to .csv and delete rows that are not relative:**

|  |
| --- |
| **tcptrace -l -r -n --csv part3.pcap > part3.csv** |

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import seaborn as sns  # 加载数据 file\_path = 'files/part3.csv' df = pd.read\_csv(file\_path)  # 查找相关的列 rtt\_avg\_columns = ['RTT\_avg\_a2b', 'RTT\_avg\_b2a'] retrans\_max\_columns = ['max\_#\_retrans\_a2b', 'max\_#\_retrans\_b2a']  # 分析 RTT 平均值与最大重传次数的关系 for rtt\_col, retrans\_col in zip(rtt\_avg\_columns, retrans\_max\_columns):  # 绘制散点图来查看这两个变量之间的关系  plt.figure(figsize=(10, 6))  sns.scatterplot(x=df[rtt\_col], y=df[retrans\_col])  plt.title(f'Relationship between {rtt\_col} and {retrans\_col}')  plt.xlabel('Average RTT')  plt.ylabel('Max Number of Retransmissions')  plt.show()   # 计算相关系数  correlation = df[rtt\_col].corr(df[retrans\_col])  print(f"Correlation between {rtt\_col} and {retrans\_col}: {correlation}") |

**Result:**

|  |
| --- |
|  |
|  |

**Output:**

|  |
| --- |
| **Correlation between RTT\_avg\_a2b and max\_#\_retrans\_a2b: 0.19473284922777326**  **Correlation between RTT\_avg\_b2a and max\_#\_retrans\_b2a: -0.01957916419893941** |

I have analyzed the relationship between the average Round-Trip Time (RTT) and the maximum number of retransmissions in both directions (a2b and b2a). Here are the results of the analysis:

For RTT\_avg\_a2b and max\_#\_retrans\_a2b:

The scatter plot displays the relationship between these two variables.

The correlation coefficient is 0.195, indicating a slight positive correlation.

For RTT\_avg\_b2a and max\_#\_retrans\_b2a:

The scatter plot shows the relationship between these two variables.

The correlation coefficient is -0.020, indicating almost no correlation.

These results suggest that the relationship between the average RTT and the number of retransmissions may not be very strong, especially in the b2a direction.

**1.12: Total traffic volume during the connection (you get the volume from PS2).**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt from geoip2.database import Reader  def convert\_to\_byte(row):  units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}   try:  ld\_bytes\_unit = str(row['ld\_bytes\_unit']).lower()  factor = units[ld\_bytes\_unit]  ld\_kb = row['ld\_bytes'] \* factor   rd\_bytes\_unit = str(row['rd\_bytes\_unit']).lower()  factor = units[rd\_bytes\_unit]  rd\_kb = row['rd\_bytes'] \* factor   total\_bytes\_unit = str(row['total\_bytes\_unit']).lower()  factor = units[total\_bytes\_unit]  total\_kb = row['total\_bytes'] \* factor   return pd.Series({'ld\_bytes': ld\_kb, 'rd\_bytes': rd\_kb, 'total\_bytes': total\_kb, 'server\_ip': row['second\_ip\_interface']})  except KeyError as e:  print(f"Error processing row {row}: {e}")  raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  df = pd.read\_csv('files/part2.txt', sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  new\_column\_names = ["first\_ip\_interface", "arrow", "second\_ip\_interface", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  "start", "duration"]  df.columns = new\_column\_names  pd.set\_option('display.max\_columns', None)  df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  print(df["total\_bytes"].sum()) |

**Result total bytes:**

|  |
| --- |
| **54698449** |