Aalto-yliopisto
Aalto-universitetet
Aalto University

ELEC-E7130
Internet Traffic Measurement and Analysis

Data Analysis

Assignment #5

Aitor Urruticoechea Puig
aitor.urruticoecheapuig@aalto.fi
Student N°101444219
October 2023

Contents

List of Figures

Task 1: Understanding different plots

In the first task, explain the following plots briefly. For example, what the y-axis/x-axis represent, the appropriate usage scenarios, and the limitations of each plot.

1.1 Autocorrelation plot

These plots, as the name implies, are a tool that reflect the correlation points in the same data set have with each other, and is used to check the randomness in such data set. The x-axis represents the time lag between the different points, and the y-axis represents the correlation of the points between themselves. While useful for periodicity analysis and forecasting; it has limitations when in comes to analysis of non-time-series data. It can also be difficult to interpret if the data showcases complex patterns.

1.2 Boxplot

A boxplot represents graphically the distribution of the data set; including relevant statistical variables like the median, quartiles, etc. The y-axis represents the data values, while the x-axis is non-existent or can be used to include in the same boxplot different datasets, experiment numbers, etc. These can be very useful in quickly summarizing the data and giving an overall overview, or to compare datasets. Nonetheless, they do have quite the pitfalls when it comes to nuance: they provide with a simplified overview of the data; and when it comes to small datasets, outliers may be misleading.

1.3 Lag plot

A lag plot visualizes the relationship between data points and the previous version of themselves after a specified lag. The x-axis is the temporal line, while the y-axis represents the data corresponding to the time t and $t + \text{lag}$. This is interesting for pattern recognition, time dependency analysis, and even future predictions. However, this plot is very sensitive to the lag value chosen, and is not suitable for lots of types of data sets.

1.4 Parallel plot

Parallel plots are generally used to represent multidimensional datasets (more than two or three dimensions). Each data point here is represented with a line connecting it to the different axis; where each axis represents one of the relevant dimensions. It can have as many axes as needed; but it can certainly become less effective as the number of dimensions increases. Similarly, since many dimensions are normally being used, datasets that are large can be difficult to interpret with this plot. Nevertheless, they are useful for exploring such sets without necessarily neglecting dimensions, and help in identifying clusters of data, patterns, etc.

1.5 Scatter Matrix Plot

A scatter matrix plot may be used to visualize the relationships between pairs of variables in a dataset by creating scatter plots for all combinations. In this way, each pair of variables is represented by a scatter plot with two axes. These plots are useful for exploring correlations, patterns, and associations between multiple variables in a dataset. Still, it is easy for them to become cluttered with many variables; while they might not reveal higher-order relationships between variables.

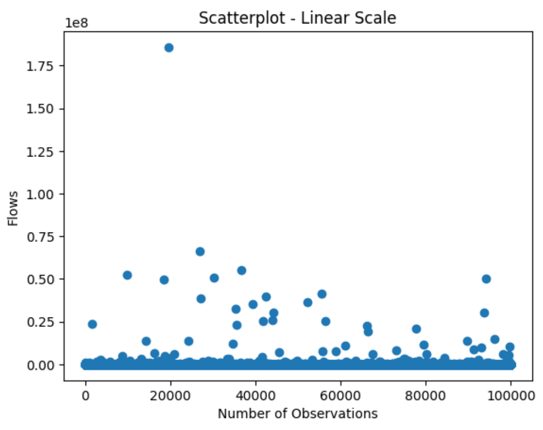
Task 2: Plot data

2.1 Plot data using different methods

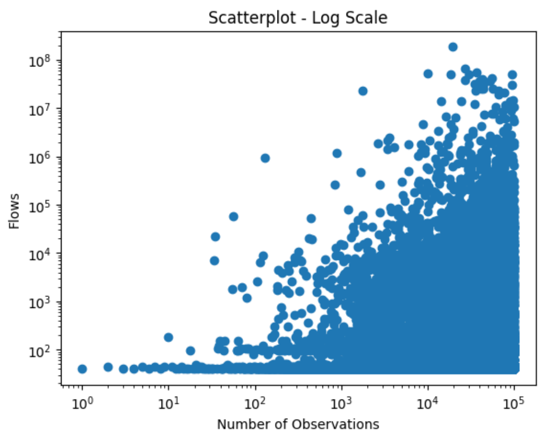
Plotting the given data in flows.txt has actually been a pretty direct job using numpy and matplotlib. The first one allows for easy lecture of the given text file and transformation into a Python object; while matplotlib provides for tools for data visualization for each one of the required plots. It also provides for tools to change the scale of the axis into the required ones. However, statistical tools provided by statsmodels have been required to generate the Empirical CDF plot; which requires a bit more of data treatment.

```python
from statsmodels.distributions.empirical_distribution import ECDF
ecdf = ECDF(data)
x = np.sort(data)
y = ecdf(x)
```

2.1.1 Scatterplot (Number of observations will reside on X-axis)



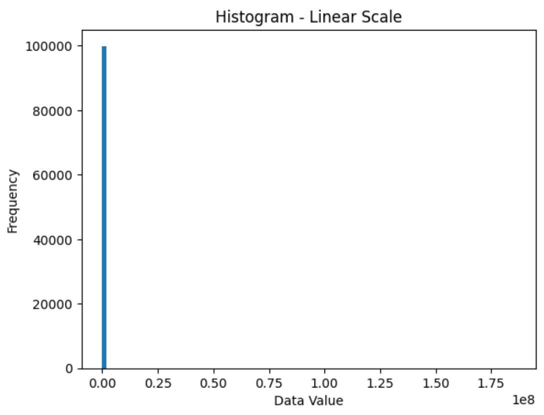
(a) Linear scale.



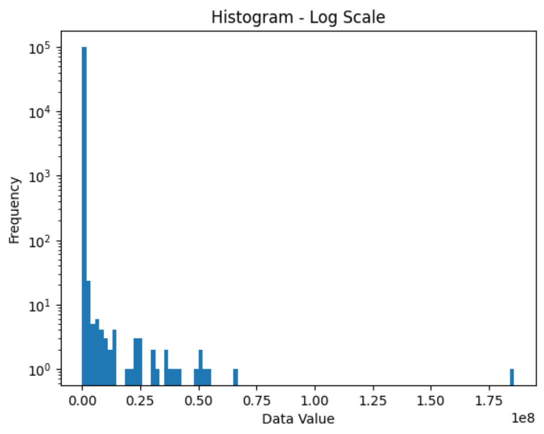
(b) Logarithmic scale.

Figure 1: Scatterplot for flows.txt.

2.1.2 Histogram (Using a suitable number of bins)



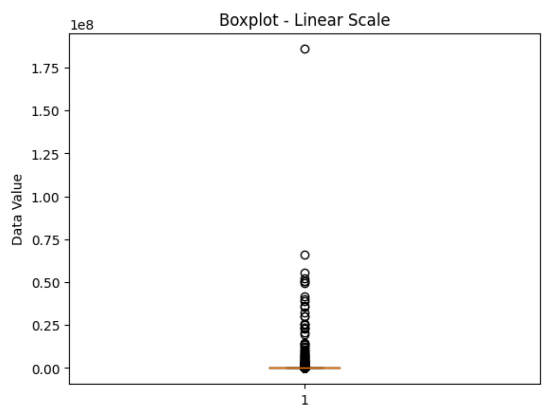
(a) Linear scale.



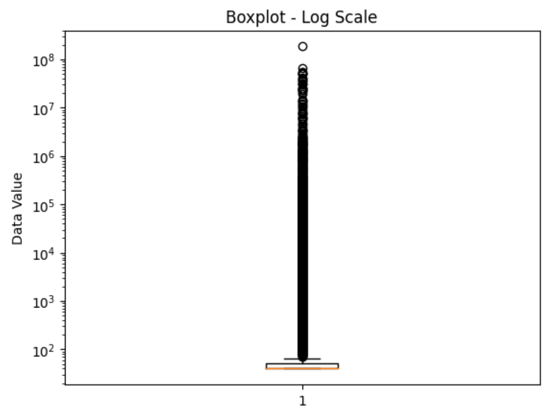
(b) Logarithmic scale.

Figure 2: Historiogram plot for flows.txt.

2.1.3 Boxplot



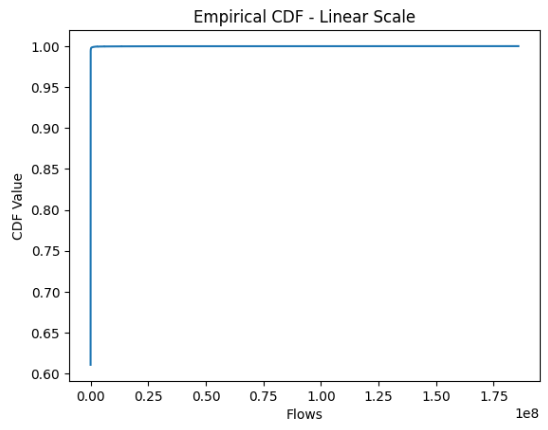
(a) Linear scale.



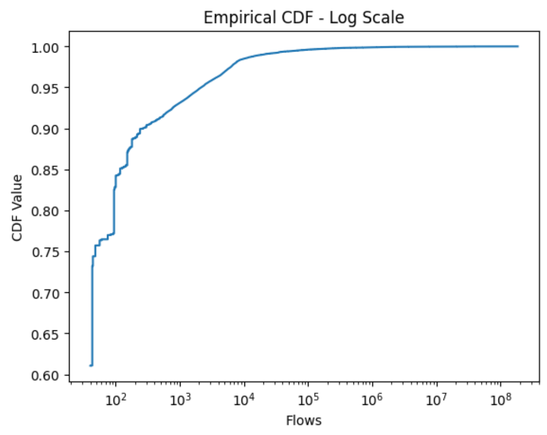
(b) Logarithmic scale.

Figure 3: Boxplot for flows.txt.

2.1.4 Empirical CDF of the variable



(a) Linear scale.



(b) Logarithmic scale.

Figure 4: Empirical CDF plot for flows.txt.

2.2 Describe the distributions choosing variables

numpy in Python has been used to obtain the desired statistical variables that best reflect the given data.

- **Minimum**: 40
- **Median**: 40.0
- **Interquartile Range**: 12.0

The choice of these three variables makes sense when looked at them together. The fact that the minimum and the median are so close reflects the heavy bias the data has towards the low end. The outliers on the higher end make the mean less relevant, for it gets very affected by them. This heavy bias on the low end is then confirmed by the interquartile range (the difference between the first and third quartiles). This is a more robust metric that the outliers on the higher end do not affect that heavily, specially when compared to the standard deviation; which for this case skyrockets to five-digit numbers.

2.3 Replot data using logarithmic values

The replots have been included in the original figures (see Figures 1, 2, 3, and 4). It is clear there that, with the heavy bias this data has on the lower end of the spectrum, the nuance offered by the higher-end outliers is lost or over-represented by linear plots. The scatter plot, for instance, is forced to concentrate the vast majority of points in just a line in the linear scale, making it look like the higher-end points are much more representative than what they actually are. Something similar happens with the box plot, where the high concentration is underrepresented by forcing it to share a tiny space. By contrast, the historiograph gets so biased by the incredibly high frequency of data on the low end that it fails to even showcase that there is data outside that range. Also extremely telling is the Empirical CDF plot, where the linear one seems to be the plot of an error; while the logarithmic one actually shows the lacking nuance of the represented data.

Overall, there are no best or worse methods to represent data; just tools for different situations. In the given case, scalar tools are certainly insufficient at telling the full picture; while logarithmic ones, specially the histogram and the empirical CDF plot, do display the distribution of the data without disregarding the enormous bias it has towards the lower end.

Task 3: Link Loads

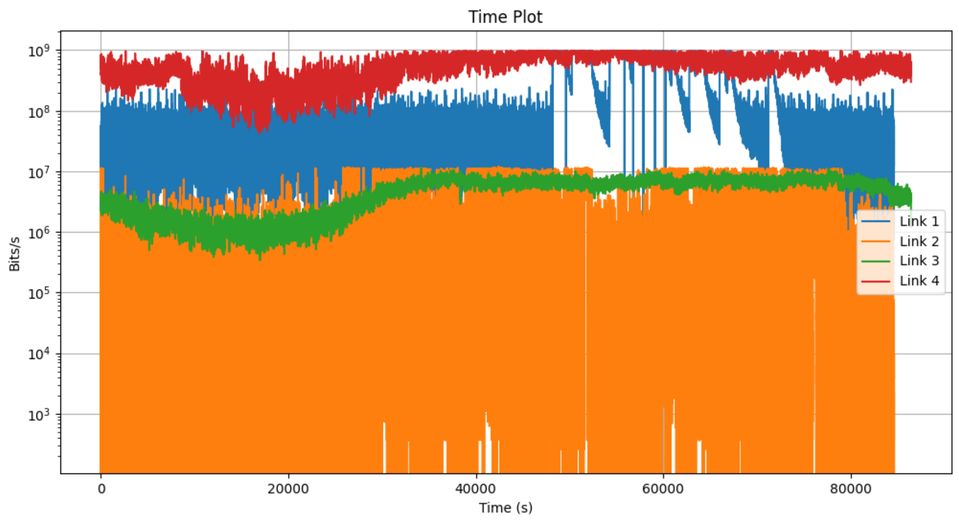
For the task 3, produce different kinds of plots that could be useful for analyzing network data such as stability and correlation.

*Download the files linkload-*X*.txt which contain link loads information (in bits per second) of different links in intervals of one second.*

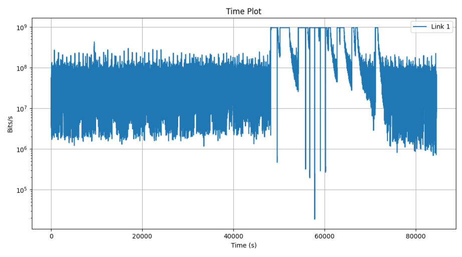
3.1 Plot the data of each link

3.1.1 Time plot

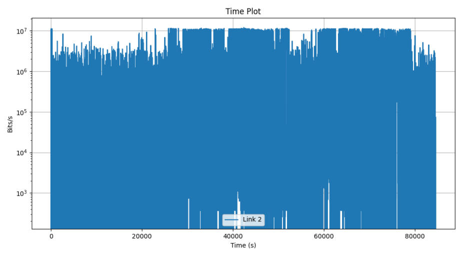
After reading the data using `pandas`, time plots are actually quite directly implemented using `matplotlib`. See Figure 5 for the plots.



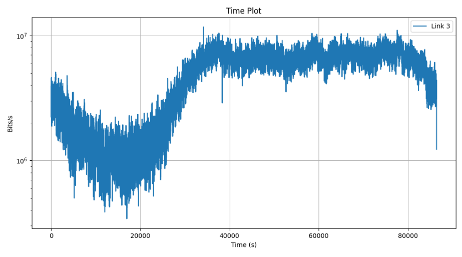
(a) Plot with all links.



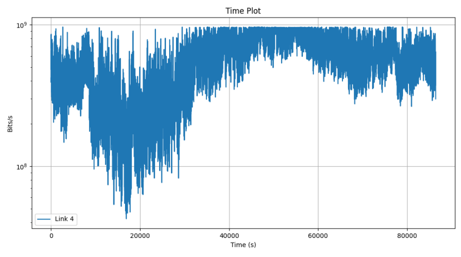
(b) Link 1.



(c) Link 2.



(d) Link 3.



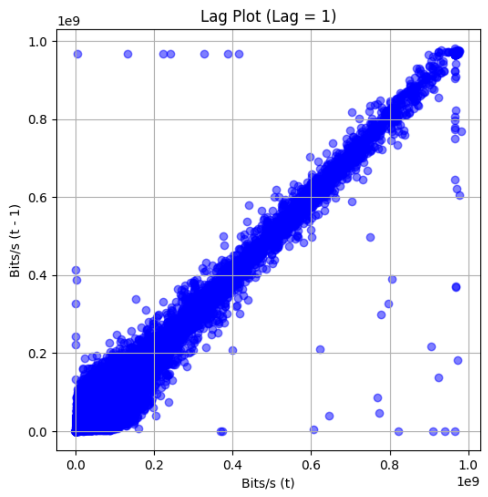
(e) Link 4.

Figure 5: Time plots for all four links.

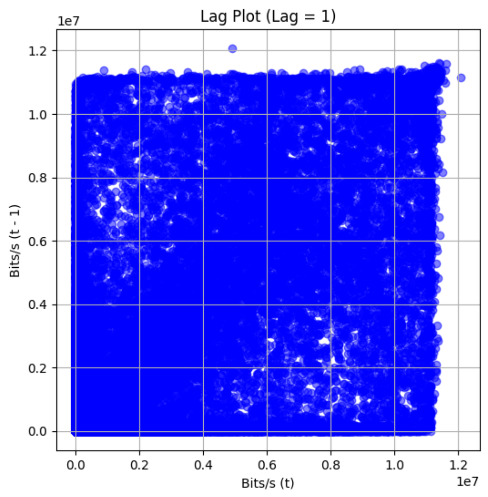
3.1.2 Lag plot (lag-1)

Lag plots need a bit more work, for the lagged data needs to also be computed. This can be done, though, quite fast using pandas' .shift() function. The obtained lag plots are available in Figure 6.

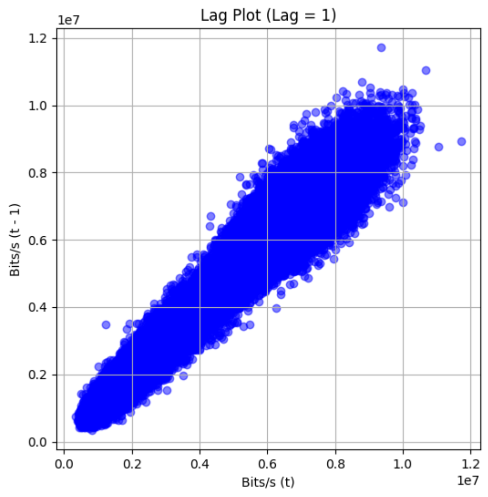
```python
lag = 1
df1['Value_Lagged'] = df1['Bits/s'].shift(lag)
```



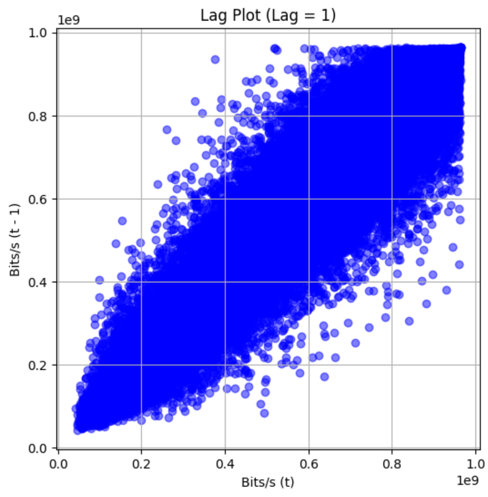
(a) Link 1.



(b) Link 2.



(c) Link 3.

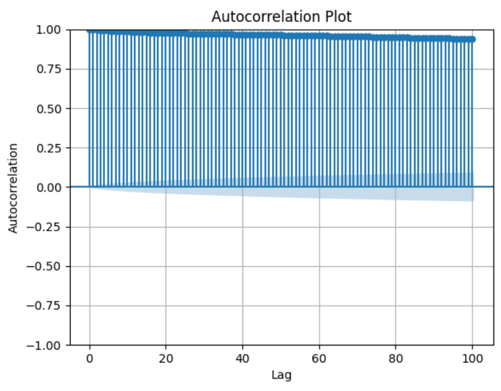


(d) Link 4.

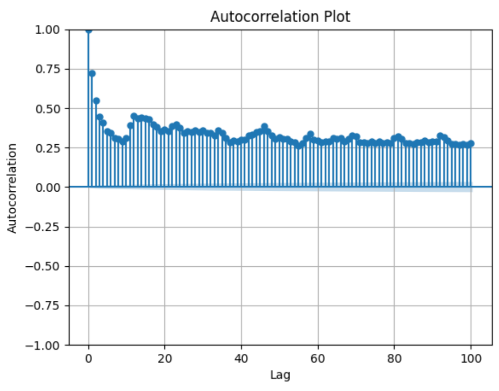
Figure 6: Lag plots for all four links.

3.1.3 Correlogram (i.e. autocorrelation plot)

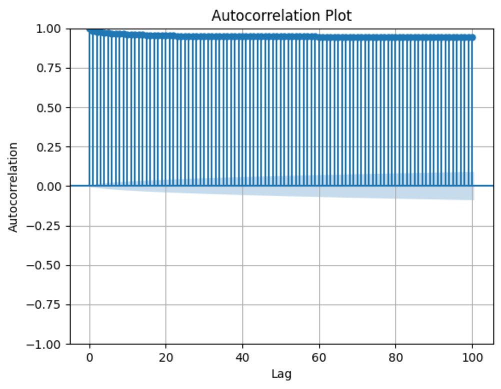
While the data for autocorrelation plots can be obtained by operating manually with the DataFrames, it is quite easier to utilise Python's statsmodels library. In it, the function plot_acf automatically creates the autocorrelation plots desired from the given data. These can be consulted in Figure 7.



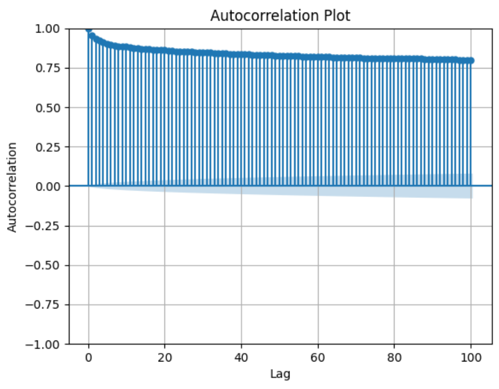
(a) Link 1.



(b) Link 2.



(c) Link 3.



(d) Link 4.

Figure 7: Autocorrelation plots for all four links.

3.2 Data inspection

While data corresponding to links 1, 3, and 4 display quite predictable data patterns, with generally gradual changes and high values for autocorrelation, Link 2 is erratic to say the least. The data rate seems to be all over the place in the time plot; which is in turn also displayed in the lag plot. Its most readable plot is the autocorrelation one, which finally displays something that one can use to draw conclusions from. It is clear that, while autocorrelation is initially high; it very quickly drops to values between 0.5 and 0.25, which indicate very low correlation.

Interestingly, Link 1 has also a period of time seen in the time series where the data seems to also deviate from the norm; but in its case the autocorrelation plot fails to showcase this. Instead, the lag plot becomes quite more useful, where the dispersed points not located in the diagonal reflect the non-normative behaviour that Link 1 does have, while not obviating the clear pattern that is observed in that link otherwise.

Task 4: Pairs plot

In the case of this task, graph a pairs plot for each one of the variables contained in the data set to verify the correlation and relation between them.

Download the bytes.csv dataset contains time series data of 4 relevant columns: transmitted bytes, received bytes, transmitted packets, and received packets.

4.1 Plot the pairs plot

The pairs plot can be done using, again, `matplotlib`. This operation requires iteration through the different variables, and special treatment to the timestamps. This is because this time the timestamps include the timezone, and `datetime` has sometimes trouble with that. In any case, once this special case is treated, the iteration through the different columns is quite direct, for the typical plots only need to be placed in the correct place, and correct skipping of the plots that would correlate a variable to itself. The final pairs plot can be consulted in Figure 8.

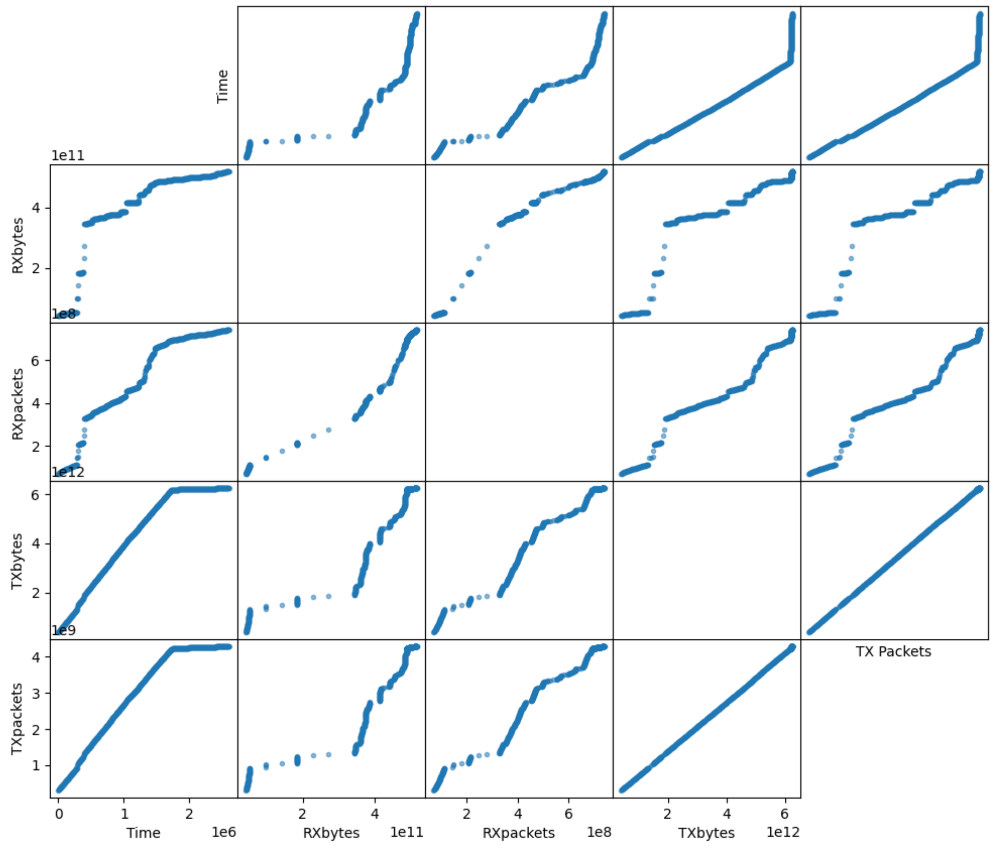


Figure 8: Pairs plot for bytes.csv.

4.2 Answer the following questions

4.2.1 Which variables correlate most to each other?

The correlation between transmitted packets and transmitted bytes is clearly the highest one. After that, it is interesting to see the transmitted packets over time, which show a clear linear trend up until $2 \cdot 10^6$ seconds, when it seems fewer packets are transmitted.

Interestingly, these correlations are not as clear for the received packets and bytes. While there is a somewhat linear correlation between these two variables, they do not follow the expected pattern of transmitted packets and bytes; and instead follow a more erratic pattern; probably displaying sone sort if packet loss along the transmission.

4.2.2 Let's assume that you decide to remove one particular column to reduce the computation load of data handling. Based on the pairs plot, what would the column be, and why?

The almost one-to-one correlation between transmitted packets and bytes makes showing both of them redundant; so any of them could be removed without losing too much information. However, since the network status and functioning is usually of interest, and here what is clearly observed is some sort of packet loss, it is most probably safer to just keep the transmitted packets and remove the transmitted bytes. This would allow for deeper analysis regarding the observed packet loss without losing too much nuance; while it would probably also account for higher reduction of load in data handling (since the bytes number is generally higher than the packets' one and thus tends to need more bits to be stored).

Task 5: Understanding time series concepts

For this task, visualize the data set by creating a time series plot, which helps in understanding the patterns and trends over time.

Download the querytime.csv dataset which depicts the query time to a distant website with a server located in Belgium.

5.1 Plot the timeseries

The time series has been created, again, using a combination of pandas and matplotlib. It can be consulted in Figure 9.

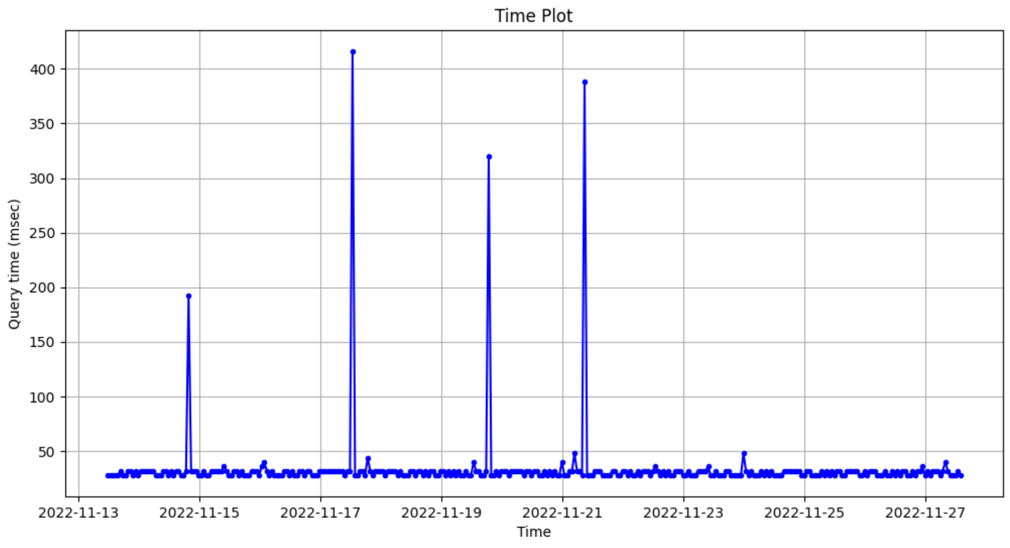


Figure 9: Time series for querytime.csv.

5.2 Answer the following questions

5.2.1 Is there any trend or seasonality?

Here query time seems to be generally quite stable, staying most of the time at around 25 miliseconds. Nonetheless, it is hard to ignore the four peaks that skyrocket the query time to over 400 miliseconds in the worst of cases, and to a remarkable 200 miliseconds in the lowest peak. These seem to occur at four afternoons-to-evenings in the early-to-mid period of the measurements. This seems to indicate that the cause for these peaks is a single or a set of connections that have either been solved or improved after the last peak happened or that have ceased to be done after the last event of high query time.

5.2.2 Is the time series stationary?

Ignoring the four aforementioned peaks, the time series is highly stationary. The data is highly time-invariant and there seems to be no noticeable trend other than remaining at around 25 miliseconds of query time.