Universiteit Leiden
Opleiding Informatica

ETA: A modular approach to analyzing time series data

Name:	Joost Martens
Studentnr:	s1314106
Date:	27/01/2017
1st supervisor:	Drs. Bas van Stein
2nd supervisor:	Dr. Wojtek Kowalczyk

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

ETA: A modular approach to analyzing time series data

Joost Martens

Abstract

Time series data can be defined as data taken over time. A large amount of time series data is being gathered every day. Analyzing time series data requires multiple techniques to understand, prepare, visualize and gain knowledge from the data.

Although several free tools for time series data analysis exist, none provide a platform that feature a range of techniques while providing users the ability to implement their own algorithms. *Environment for Time series Analysis* (ETA) is the platform we propose to fill this gap. The goal of our platform is to provide an easy way to perform all data analysis related to time series and enable users to expand on it by easy creation of their own modules. ETA is open source and written in *Python*, which has a lot of packages that can be utilized. Some modules for profiling, plotting, preprocessing and machine learning are already provided in ETA.

Acknowledgements

I would like to express my deepest appreciation to my academic supervisor Bas van Stein for his feedback, time, effort and constant guidance throughout the work. I also want to thank Shengfa Miao who helped earlier in the process and who was just as helpful in that stage of the work.

I would also like to express my gratitude towards my father, mother, brother, both of my sisters and my grandparents whom I can always count on and who have always been there for me. Without them I would not have been the person I am today. Finally, I would like to thank my girlfriend for her support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The amount of data collected is getting larger every day. Corporations, organizations, scientists and institutions are trying to gather as much data as possible. Even data that is not useful yet, but which might become useful in the future, is already being gathered. At the same time people are creating more and more data themselves. A lot of their equipment like smartphones, smartwatches and home automation systems are generating and gathering data.

Data comes in all kinds of different types. Our primary focus is time series data. *"A time series is a sequence of observations taken sequentially in time" [2].* A large amount of time series data comes from sensors that continously collect data. An example is a weather station that measures the temperature and wind speed every second. These large amounts of data make data analysis and knowledge discovery possible, which is difficult by default. Other challenges come from data being missing, incorrect or inconsistent. This leads to misinterpretation of the data and can lead to undesired results. To correct this data one has to understand it and its structure. The data can then be preprocessed to give a clean data set.

There are already several free time series data analysis software tools available. However, to our knowledge we are missing a platform that provides a range of time series analysis tasks and is easy to work with while giving the user the option to implement their own techniques. That is why we propose ETA, *Environment for Time series Analysis*. Our open source[1] platform offers a framework for different functionalities to work with time series data that can be expanded with modules.

ETA is an evironment where one can analyze, process, visualize and gain insight into data. This means tackling the challenges mentioned above, which requires multiple data analysis techniques. It is an all–in–one platform where one can easily use data analysis techniques. Because numerous different techniques exist and the fact that one might want to design its own algorithm or functionality the platform has to

[1] Source–code can be found at `https://bitbucket.org/eta_tkinter/eta/`.

be modular. That is why we created ETA and each of its core functionalities to be modular. Users can also implement their own modules on top of the existing framework. Because of this framework the user does not need to focus on functionalities like data loading, data handling, parameter handling, or any other functionality already implemented. One of these core functionalities is data profiling. Data profiling can be used to gain information and statistics about the data. The data can be cleaned and prepared for analysis using preprocessing techniques. Machine learning algorithms can be applied for classification, clustering and regression. Finally, the data can be visualized using various plotting techniques. All of these functionalities have some modules already implemented. Using them is done by simply selecting the module from ETA's menu. The modules can also be improved or the platform can be expanded with new modules. The choice for Python as the platforms programming language makes the implementation of modules very easy. It is one of the most popular programming languages in scientific computing. Python's (scientific) libraries allow easy implementation of new functionality as well.

The next chapter provides some background on data analysis. It shows some data analysis challenges and solutions using an example data set. A comparison of different time series analysis software will then be shown in related work. Chapter 4 will discuss ETA as a platform and the design choices of its architecture and its interface. After showing the framework, ETA's functionalities and modules will be discussed. In chapter 5 we show how easy it is to use ETA by giving a step–by–step guide on how to use ETA, without the need of any programming. Finally, we will present our conclusions and suggest future work to be done to further improve the platform in chapter 6.

Chapter 2

Background

Data can be classified into two different types: categorical and numerical. Categorical data is data that can be sorted into groups or categories. Examples are sex, blood group or nationality. Numerical data is data that can be measured. They can be placed in ascending or descending order. Examples are height, income or age. Time series data is a form of numerical data.

A data set is most commonly represented by a data matrix. Every column of the table represents a particular variable (feature). Each row corresponds to a given member of the data set (a sample). An example of a data set is given in table 2.1.

	Columns				
	index	date	temperature	condition	ski–rentals
Rows	1	2017-01-01	1.2	cloudy	2
	2	2017-01-02	0.6	rain	17
	3	2017-01-03	-3.0	snow	22.5
	4	2017-01-04	2.3	cloudy	24
	5	2017-01-05	1.3	rain	19
	6	2017-01-06	-2.3	snow	25
	7	2017-01-07	-10.0	snow	27
	8	2017-01-08	-3.7	?	33
	9	2017-01-09	0.3	cloudy	35
	10	2017-01-11	1.1	cloudy	28

Table 2.1: A self–created time series data set.

This data set has some problems. For example, it contains the value 22.5 in ski–rentals. This should not be possible. The best solution would be to change the value to 22 or 23. The second problem is that the data has a ? as a condition. The temperature of that sample is negative and the amount of ski–rentals is rising. As one can see from the other data rows, this probably means it was snowing. The next problem is that the data set misses a day between index 9 and 10; 2017-01-10 is missing. To fill this, one could make its temperature the average of the previous and next day. Since both days are cloudy one can also select cloudy as its condition.

For its ski–rental value one might not want to overestimate and take the lowest value of the two. A not so obvious fourth problem is the value -10.0. It might be possible that this was the actual temperature; but since the change in temperature is so significant it is also possible the sensor was malfunctioning. This is called an outlier.

The problems above are an example of the importance of data analysis. As you can see, with data analysis one should be able to detect errors and know how to solve them. Getting rid of these problems is what the data analysis technique called preprocessing does. The second problem could be fixed by applying a different technique called machine learning; by applying a machine learning algorithm on the rest of the data one can predict the condition. Visualizing the temperature over time would show the outlier of the fourth problem. And although this data set is small enough for a user to understand, the data set could contain millions of rows. This is why one might want to get more insight into their data to detect possible problems, which can be done using data profiling. These data analysis techniques are all available in ETA and will be discussed in further detail later in section 4.4, core.

Time Series: Seasonality, Trends & Forecasting

If the time series data set shown above had information spanning multiple years, one would see an example of *seasonality*. The temperature will be low in the winter and high in the summer. This is called seasonality, the cyclic fluctuation over a period of time. The temperature during the day also displays seasonality.

Trend is how the value behaves over time without the effects of the calendar and other irregularities. An example is that the temperature is rising due to global warming. This is called an upward trend. With time series data it is important to take seasonality into account. This is especially important with *forecasting* [3]. Forecasting is the process of using past data to predict future data.

Exploratory Data Analysis

Exploratory Data Analsysis (EDA) is an approach to data analysis. Usually one has some assumptions or hypotheses that they want to test on the data. EDA postpones this and first lets the user explore the underlying structure and model of the data. Based on the exploration, one can create new hypotheses or alter current ones.

Exploratory Data Analysis is one of the reasons we created ETA. Although people might want to test a technique or model of their own, provided techniques allow users to gain insight into their data and how to approach or improve their data and techniques. This maximizes the insight into a data set which is the primary objective of EDA. Other reasons [4] someone might want to use EDA is to extract important variables, detect outliers and anomalies, test underlying assumptions and determine optimal settings for future processes.

Most of the techniques used in EDA are graphical. Graphics are the best way to show the structure of the data and give the user the resources to analyze it. The importance of graphical representations of data will be discussed in more detail later in section 4.4.1, plotting. The other way to explore the data is the use of quantitative techniques. These yield numerical or tabular output. A simple example is calculating the average value of a data column. Data profiling, which will be discussed in section 4.4.2, allows exploratory data analysis by making use of both graphical and quantitative techniques.

Chapter 3

Related Work

Weka *Waikato Environment for Knowledge Analysis* (Weka) [5] is a popular machine learning software tool. It is developed using the Java programming language. Weka also allows the implementation of new algorithms. Since version 3.7.3, a package can be installed to allow forecasting for time series. The weka interface can be seen in figure 3.1. Weka has a machine learning approach where it aims at creating models. However, it does not provide the flexibility for all the tasks we deem useful for data analysis. It is also limited in handling large data files.

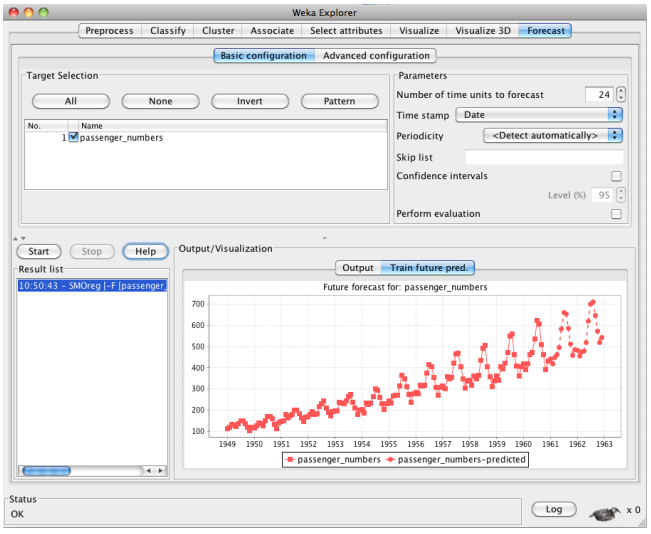


Figure 3.1: Weka time series forecast example.

Orange Orange [6] is a machine learning and data mining tool. It is built using Python and C++ and allows the user to write Python scripts. It also has an interface that allows visual programming where users can drag components in an Orange workflow. This interface can be seen in figure 3.2. One of its biggest advantages is that Orange is very easy to learn. It also features a package manager that contains a time series package. However, the visual capabilities for that package are limited. For example, the line

plot for time series did not allow us to save the image. Orange was also slow with importing the 3*MB* Infrawatch [7] data file and was not able to import the 300*MB* file at all. For comparison ETA imports the smaller file instantly and the bigger file within a minute.

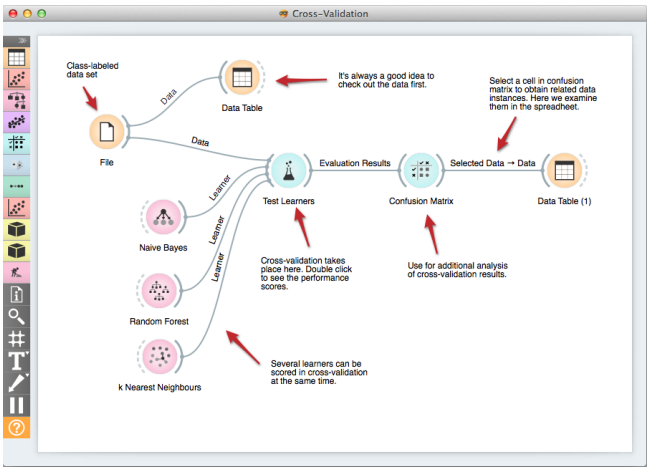


Figure 3.2: An example of a Orange workflow.

KEEL *Knowledge Extraction based on Evolutionary Learning* (KEEL) [8] is a software tool to assess evolutionary algorithms for data mining and provide solutions to data mining. It has a collection of libraries for preprocessing and machine learning techniques. However, it lacks in visualization techniques. It also has problems handling the import of large files. When testing it with the Infrawatch file it took several minutes to load the smaller file and crashed on loading the bigger file. Trying to perform operations on the smaller file made the application crash as well.

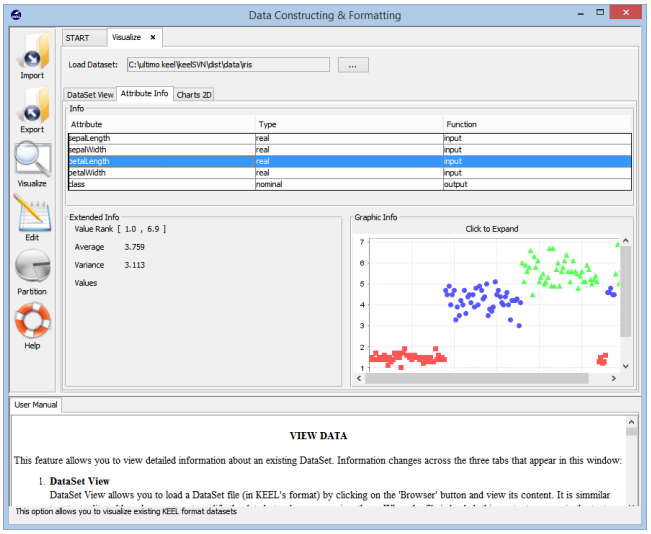


Figure 3.3: An example of the user interface of KEEL.

A comparison to other time series analysis software can be found in the bachelor thesis of Lars Hopman [9].

Chapter 4

ETA: Environment for Time series Analysis

Environment for Time series Analysis (ETA) is an open source platform designed for time series data analysis. Although its focus area is time series data, it can handle different data as well. Therefor it can be expanded and used for different areas.

ETA's power lies in its modularity; the user can implement their own algorithms by creating a module and utilize built–in functionalities such as data loading and parameter handling. However, ETA is built to be easy to use. No programming knowledge is required to make use of ETA's implemented modules. Performing data analysis is as simple as a few simple clicks in ETA. All functionality is available from the menu and parameters are inserted from a pop–up window.

ETA's supported data analyzing techniques and modules will be explained in section 4.4, core. The second section will explain the architecture of ETA and the third section shows the graphical user interface. But first the choice for the use of *Python* and its packages will be explained.

Lars Hopman also contributed to the development of ETA. For his contributions see his bachelor thesis [9].

4.1 Python

Since an important part of ETA is the ability to expand on the functionality and create modules, modularity is key. This also means that ETA should be available on as many operating systems as possible.

Using Python as the underlying language is the perfect solution for numerous reasons. Firstly, it is supported

by all major operating systems. Secondly, it is already a widely known and used programming language in statistics and data mining. This is also because it is known for its readability which in turn makes it easy to learn.

Finally one of Python's greatest strengths is that it has a large standard library. On top of that the Python Package Index or *PyPI*, which is the official third-party software repository, currently contains over 95.000 packages and is growing continously. These packages allow us to easily create the graphical user interface, handle data frames, create plots, implement machine learning algorithms, and much more. Below some of the packages used in ETA will be explained. These packages can all easily be installed or updated by running a simple shell script, which will be shown in chapter 5. New packages can be added by simply adding its name to the requirements text file and running the shell script provided.

Tktinter The *tkinter* [10] — Tk interface — package is the standard Python interface to the *Tk* GUI toolkit. It is available on Windows and Macintosh systems and most Unix platforms. Tkinter comes included with the Python standard distribution. It offers a native look and feel on all platforms.

Pandas *Pandas* [11] provides high–performance, easy–to–use data structures and data analysis tools for the Python programming language. Pandas is used to import the data and store it in a pandas data frame. A pandas data frame is a 2-dimensional data structure with labeled columns of potentially different data types. Some of pandas built–in data frame manipulation functions are also used.

Numpy *NumPy* [12] is the fundamental package for scientific computing with Python. It adds support for multi–dimensional arrays. It comes with a large library of mathematical functions to operate on these arrays. In ETA, it is mostly used for array manipulations on data from a pandas data frame.

Scikit–learn *Scikit–learn* [13] (or *sklearn*) is a machine learning library for Python. It has simple and efficient tools for data mining and data analysis and features various machine learning algorithms. It is designed to interoperate with the previous mentioned library numpy.

Matplotlib *Matploblib* [14] is a Python 2D plotting library which produces various figures in interactive environments across platforms. It can generate various plots, charts, etc. using just a few lines of codes. Its style or properties can be controlled via an object oriented interface or via a set of functions. Matplotlib is used to provide the plots in ETA.

Seaborn *Seaborn* [15] is a Python visualization based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics. This module is used to make the matplotlib plots more aestatically pleasing.

As you can see by the Python modules described above a lot of these modules have functionality that can be used inside the platform. These modules make it easier to create and implement certain functionalities and

modules to expand the platform.

4.2 Architecture

The architecture will be introduced using the workflow in figure 4.1. Generally it shows the steps the program goes through while the user makes use of it. The first step, from now on referred to using the notation (1), shows the initialization of the handlers and the creation of the Graphical User Interface (*GUI*). The second step (2) shows what happens when a file is loaded and the third step (3) shows the use of one of the functionality handlers. All of this is discussed more thoroughly below.

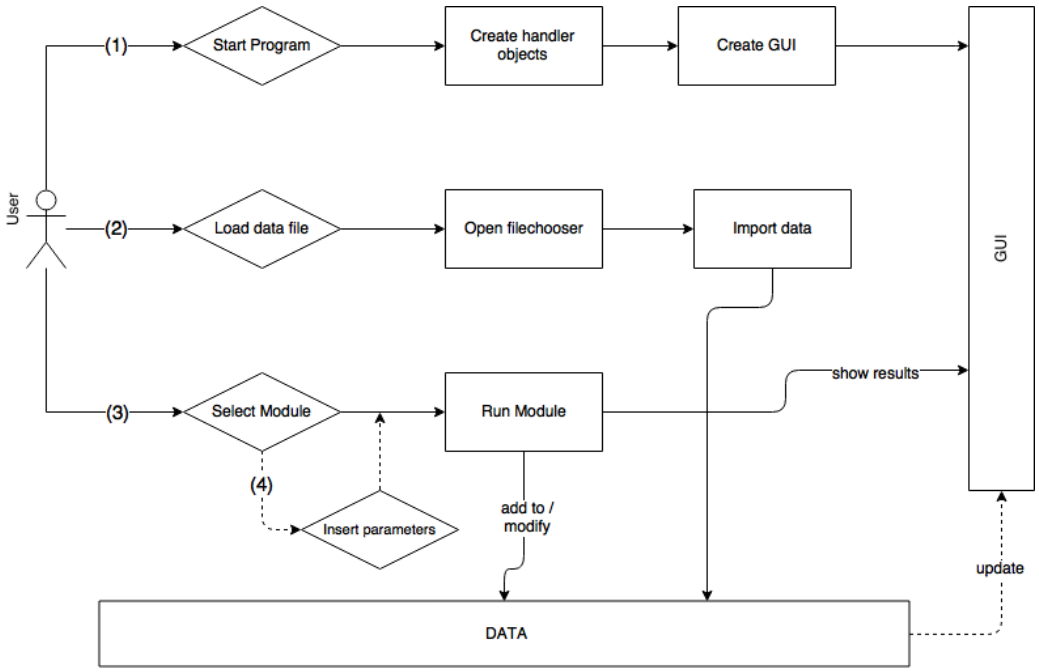


Figure 4.1: The worflow of the platform.

When a user starts the application (1), it calls the *coreHandler*. It first creates all the handler objects: *moduleHandler*, *parameterHandler*, *filesHandler*, *dataframesHandler*, *guiHandler* and what from now on will be referred to as functionality handlers: *plotHandler*, *profilingHandler*, *preProcessingHandler*, *machineLearningHandler*. The coreHandler keeps track of these handlers. When a handler want to make a call to another handler it does this through the corehandler. This implementation preserves the modularity of the framework and allows easy creation of new handlers or modification of current handlers without breaking the software. After the handlers are created, the coreHandler will make a call to the guiHandler to initialize the interface which completes starting up the application.

The guiHandler creates the root window containing nested frames for each different GUI element. In the class diagram of figure 4.2 you can see how the GUI is build. Each of those elements is handled by a different

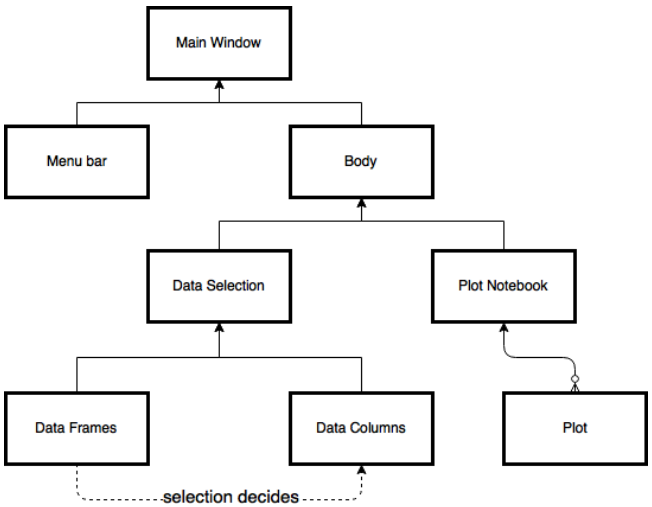


Figure 4.2: The class diagram of the GUI.

class which makes changing or replacing a GUI element very easy. Each element will be explained in the next section of this chapter.

When creating the menu bar of the GUI the guiHandler works together with the moduleHandler. For each of the functionality handlers, the moduleHandler provides the modules from the correct module folder. Modules nested inside a folder in the module folder will be displayed in a submenu. The moduleHandler also returns the module object when the user chooses one from the menu.

To use the platform, the user must first load a data file. When the user selects the menu item to load a file (2), a window to choose a file will pop up. This window is native to the operating system as can be seen in figure 4.3. When a file is chosen the guiHandler passes the file name to the filesHandler. The filesHandler recognizes the file extension and loads the data accordingly. The data will then be returned as a pandas data frame and passed on to the dataFramesHandler. The dataFramesHandler adds the data frame to the list and handles further data calls to the data frame. Examples are getting the data, adding or removing columns from a data frame or creating a new empty data frame.

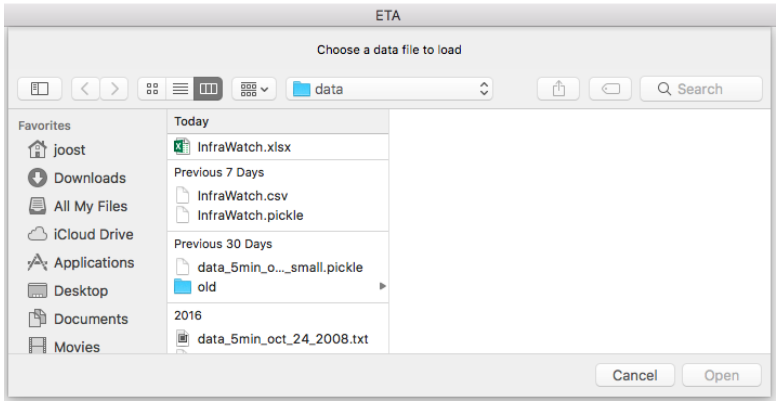


Figure 4.3: macOS Sierra native file chooser for ETA.

With the data set loaded into the platform, one can now make use of the functionality handlers. Some modules and other functionalities inside the platform require parameters. The parameterHandler takes care of that. The process of handling parameters (4) for modules goes as follows; first it gets the parameters from the module. It then creates a window with the help of the guiHandler containing the label and an entry option according to the parameter type. An example of a window and a list of parameter types can be found below in the section 4.3.4, parameters. If there are previousy used parameters saved, it will get those and insert them as the selected value. If those are not available, default values from the module itself will be used if available. The user can now select the parameters. When done, the selected parameters will be saved and passed along to the functionality call.

These functionality calls are generally done in the following way. First the user selects a module to be used. A call to the moduleHandler is made to get this module. A call to the dataFramesHandler will then be made to get the currently selected data. After that, a call will be made to the parameterHandler to get the parameters for the module. Finally the corresponding functionality handler from the module will be called using the module, data and parameters. This will then perform the modules actions. The profilingHandler will generally output a profiling report. The plotHandler will create a plot. The preProcessingHandler will alter the current data frame or create a new one. And the machineLearningHandler will create a model, alter the current data frame or create a new one or output a score when respectively fit, predict or score are chosen from its module options.

The modularity of the platform does not only come from its modules, it also comes from the different handlers. For example, it would be possible for the guiHandler to create the user interface in a webpage instead of the tkinter window. The remaining handlers and calls to it can be kept the same. If you would like to add support for a new file extension you could easily alter the filesHandler to support it. Although four functionality handlers have been implemented, it is possible to implement new data analysis techniques by creating a new category and a handler to support it.

4.3 GUI

ETA's interface is built using tkinter. It uses different classes for each element as shown in the class diagram of figure 4.2. It can be divided into four major parts: the menu bar, the data selection, the plotting notebook and finally parameters. Each part will be discussed below. An overview of the full interface is shown in figure 4.4.

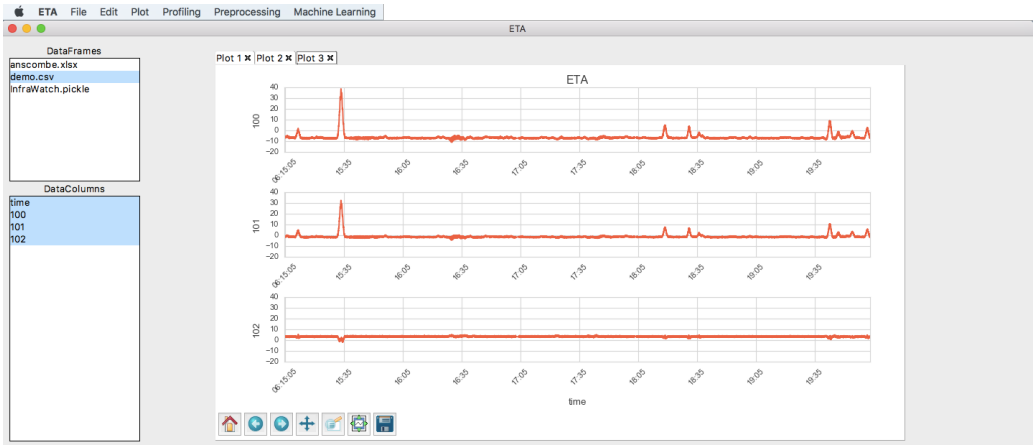


Figure 4.4: An overview of the interace on macOS Sierra.

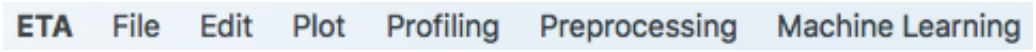


Figure 4.5: Menu Bar

4.3.1 Menu bar

The menu bar is where all the functionality can be selected. This avoids clutter in the GUI. Each menu contains a dropdown with more options. The different menu items can be seen above and its options will be discussed below.

File This menu contains items to open or save a data file. Allowed file extensions currently are:

.csv - comma-seperated value

.pickle / .pkl - pickle, Pythons own file extension.

***NOTE**: pickle files can be hacked, don't trust pickle files sent over the internet as they can run malicious Python code.*

.xlsx - Microsoft excel spreadsheet file

.txt - plain text file

It is also possible to reload the last succesfully loaded file. When the program has been closed and opened again this remains possible.

Lastly the user can save the current data frame to a file. ETA currently has support for **.pickle**, **.csv** and **.xlsx** (Excel).

Edit This menu contains functionalities to change the selection of the data columns, remove columns from the data frame or add or remove a data frame. The current selection can also be printed to the terminal. The items and their functionality are listed below:

Select all	Select all of the data columns.
Select none	Select none of the data columns.
Print selection	Output selected data to the terminal.
Add new data frame	Add a user generated (empty) data frame.
Rename selected data frame	Rename selected data frame.
Rename selected data columns	Rename selected data columns.
Remove selected data frame	Remove the current source data frame.
Remove selected data columns	Remove the selected columns from the dataset.
Remove unselected data columns	Remove the columns that are not selected from the dataset.

Other The last 4 menu bar items contain the modules of the handler corresponding to that item. They are **Profiling**, **Plotting**, **Preprocessing** and **Machine Learning**. These modules will be listed in the subsection of each item in section 4.4 of this chapter.

4.3.2 Data selection

To select which data you want to use you first have to select the data frame you want to use from the data frames listbox. In the data frames listbox you find the datasets the user loaded in, as well as any data frames added from the Edit menu. It is also possible a data frame has been generated when using one of the functionalities.

When a data frame is selected the columns it contains will be shown below it in the data columns listbox. Each column will be below one another. Selecting or deselecting a column is done by simply clicking on it. Selected columns will have a blue background while unselected columns have a white background. Most of the time actions will only use the selected data columns as data unless specified otherwise.

These listboxes allow for easy switching between data frames and data columns and provide an overview of all the data loaded. For instance, a dropdown menu does not give this overview unless clicked upon.

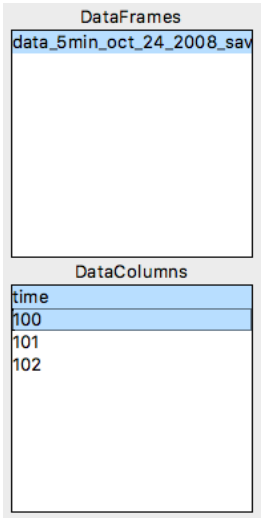


Figure 4.6: Data selection.

4.3.3 Plotting Notebook

Figure 4.7: Notebook tabs.

Figure 4.8: Plot Toolbar

When a plot is created a tab is added to the notebook. The tab itself contains the name of the tab and a button to close the tab. The content of this tab contains two elements of the plot. The first element is a matplotlib toolbar. This toolbar contains different built–in functionalities. In figure 4.8 you can see how the toolbar looks. Its functionalities are (from left to right):

reset original view	this resets the plot to how it originally looked
back to previous view	this undoes the last action performed on the view
forward to next view	this redoes the last action performed on the view
pan axes with left mouse	this gives the ability to pan inside the plot
zoom to rectangle	this gives the ability to zoom in on the plot by selecting an area
configure subplots	a pop-up window appears with sliders to change some parameters of the plot, these parameters are 'left,' 'bottom', 'right', 'top', 'wspace', and 'hspace'
save the figure	opens a filechooser window to choose where to save the plot, you can give insert a filename and choose a file extensions (options are: *.eps, *.pdf, *.pgf, *.png, *.ps, *.raw, *.rgba, *.svg, *.svgz)

The second element of the tab is the plot itself. How the plot looks fully depends on the chosen plotting option, its parameters and the data used. The toolbar functionalities can be used to modify the plot.

In figure 4.9 you can see an example of how it all looks like together.

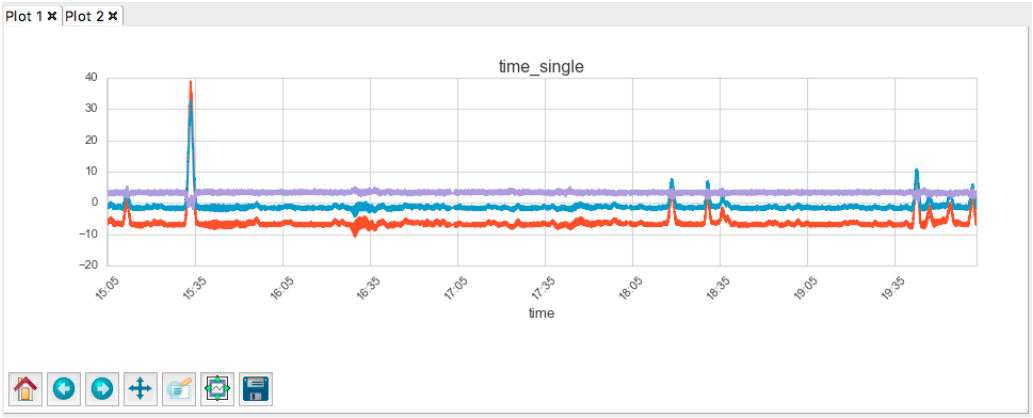


Figure 4.9: An example of the plotting notebook.

4.3.4 Parameters

Lastly, some of the functionalities require the user to select or insert some parameters. A window containing these parameters is created by first requesting the parameters of the module or function. If available it also request the last used parameters or if those do not exist the default parameters. Each parameter will be shown below eachother containing a label and the input option. ETA currently has the following input options:

- **string** – a sequence of characters, e.g. “foobar”
- **integer** – a number without decimal points, e.g. “12”
- **float** – a number with decimal points, e.g. “24.89”

  Above options use a textbox to write the input.

- **columns** - the selected data columns of the current data frame
- **list** - the items specified in the parameters list options

  Columns and list use a dropdown menu to select an option.

- **boolean** - a checkbox to indicate whether to do the indicated label action or not

In figure 4.10 an example of a parameter window is shown.

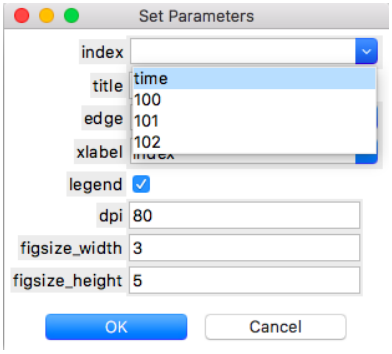


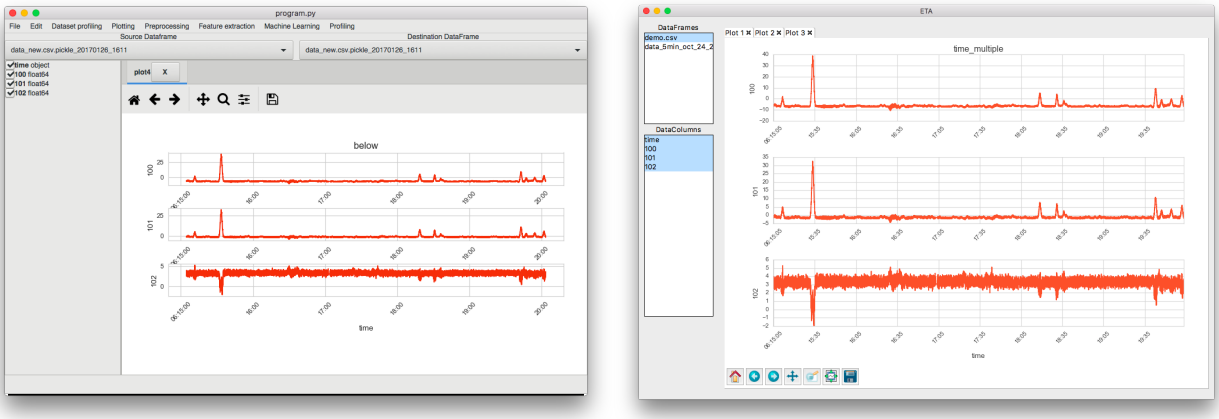
Figure 4.10: An example of a parameters pop-up window. Currently selected is ‘index’ showing its dropdown menu containing the data columns.

4.3.5 Evolution of the GUI

ETA aims for an easy–to–use design where inexperienced users are able to operate the program without much of a hassle. This makes the GUI a very important part of the system. The GUI should therefor look well and mainly be user–friendly. The design of the GUI was an iterative process and several GUI libraries were considered.

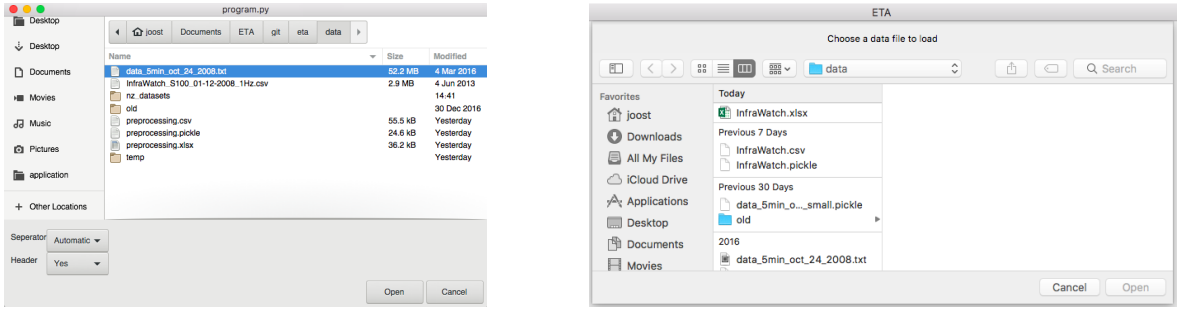
Previously *PyGObject* [16] (aka *PyGI*), instead of tkinter, was used to create the GUI. PyGObject is a Python package based on *GTK+* [17]. A side–by–side comparison between the previous PyGObject interface and the current tkinter interface is shown in figures 4.11 and 4.12.

As you can see in the comparison the tkinter interface looks less cluttered. This is because the data frame selection is moved to a listbox above the data columns listbox instead of having a dropdown menu. Previously, there were also two data frame selection dropdowns; one for the data frame and one for the target data frame. The target data frame was not a neccessary option since the target data frame would almost always



(a) PyGObject (b) Tkinter

Figure 4.11: A comparison between the user interfaces.



(a) PyGObject (b) Tkinter

Figure 4.12: A comparison between the filechooser windows.

be the same as the source data frame. This look also gives the user a better overview of the data frames and makes the selection of a data frame easier and faster.

The tkinter interface also looks more like the native desktop environment[1]. You can see this especially with the filechoosers in figure 4.12; PyGObject uses an own custom filechooser window while tkinter uses the native filechooser window. The native look that tkinter offers gives more functionality and is, in our oppinion, more appealing. PyGObject also has a lot of cross–platform inconsistency while tkinter is more stable and reliable. PyGObject requires difficult installation steps when not using a Linux distribution while tkinter is included in the Python standard distribution. This makes it more difficult to contribute to ETA which counteract its design goals. These reasons made us use tkinter for ETA's interface, which we think is currently the best option.

[1]Desktop environment used: *macOS Sierra*

4.4 Core

ETA differentiates between four different functionalities in ETA. These are *plotting*, *profiling*, *preprocessing* and *machine learning*. Each functionality will be discussed and its implementation in ETA will be shown.

4.4.1 Plotting

Broadly speaking data analysis can be split into two parts: *quantitative* and *graphical* [4]. Quantitative techniques have numeric or tabular output while graphical techniques have graphs or plots as output. Below the importance of these graphical techniques will be shown.

Anscombe's quartet [18] is a famous example of why graphical representation of data is important and shows the dangers of only using quantitative techniques. Anscombe created the four datasets shown in table 4.1. Each of these datasets have the nearly identical descriptive statistics as seen in table 4.2. The x value of the first three data sets are also the same.

1		2		3		4	
x1	y1	x2	y2	x3	y3	x4	y4
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

Table 4.1: The four datasets of Anscombe's quartet

Number of observations	11
Mean of the x	9
Mean of the y	7.5
Regression cofficient	0.5
Equation of regression line	3 + 0.5x
Sample variance of y	4.125
Residual standard deviation	1.236
Correlation between x and y	0.816

Table 4.2: Descriptive statistics of Anscombe's quartet

When one only views the quantitative statistics of these datasets, one might think they are equivalent. However, when the datasets are plotted and one fits a regression line, the result is the graphical output shown in figure 4.13.

The plots show that the data sets are very different. The first data set appears to be linear with some variance. The second data set does not follow a linear relationship, it might be quadratic. The third dataset is linear with a single outlier. And the fourth dataset shows a constant x value with a single outlier as well. It also shows that a single outlier is enough to produce a diagonal linear regression line even though the relationship is not linear.

These examples show the importance of visualizing data using a graphical representation. This does not mean that quantitative statistics are useless; they can be misleading on their own. Because of the visualization one

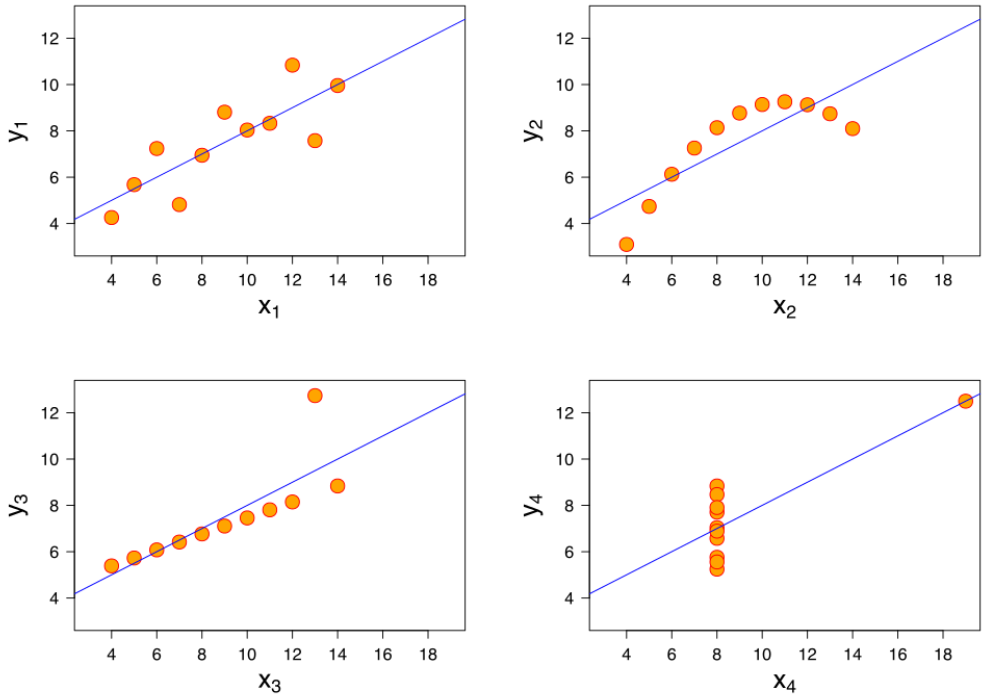


Figure 4.13: Scatter plots with regression lines of Anscombe's quartet — freely used from Wikipedia

can avoid forming false conclusions. Patterns, relationships and characteristics are more easily discovered from a visualization of the data as well. Visually representing information allows us to reduce the complex cognitive work for viewers and let them focus on the analytical process. Another advantage of a graphical representation is that computers lack the capability a human posesses. Humans have knowledge and intuition and can make decisions that cannot be automated by a computer. Data visualization allows the user to better understand the data and improve further processes and results using human resources a computer does not have.

This means that data visualization is aimed at users. Therefor the presentation of the data is very important. Presentation is one of three goals of data visualization. The aim of a presentation is to efficiently and effectively communicate the results of an analysis to a user. The ability to recognize and understand the data is key to a good visual representation. The second goal is exploratory data analysis, which is previously explained in chapter 2. The last goal is confirmatory analysis where one has one or more hypotheses which the visualization confirms or rejects.

Because data sets and goals can differ, the design of each data visualization should therefor differ as well. The modules shown below consist of various plotting techniques. They can be used for each of aformentioned goals. However a user might want to create their own module or build upon an existing one to reach their goal.

time_single For this plot the user has to select the time series value to use as the index. This value is shown

on the x-axis. The values of other selected columns are shown on the y–axis in a single plot as shown in figure 4.14.

time_multiple This plot is almost the same as time_single. Only instead of the columns being plotted in a single plot, they are shown in subplots below eachother as shown in figure 4.15.

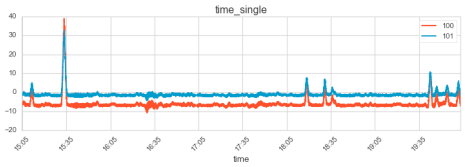


Figure 4.14: Time series plotted using a single plot.

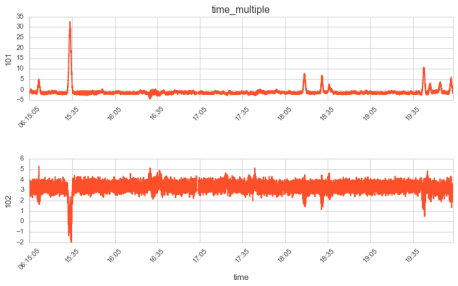


Figure 4.15: Time series plotted using multiple subplots.

scatter In a scatter plot a value on the x–axis is plotted against a value on the y–axis. The scatter plot is useful to view the relationship between two values. An example can be seen in figure 4.16.

box A box plot shows the variation of a value. It shows the maximum and minimum value as the top and bottom of the line, the third and first quartile as the beginning and end of the "box" and the median represented by the line inside the box. The range between the first and third quartile is called the *inter quartile range* or IQR. An example can be seen in figure 4.17.

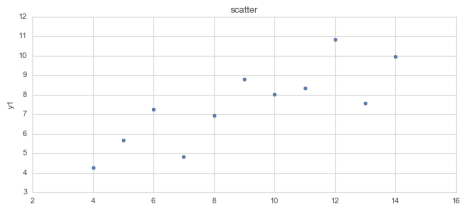


Figure 4.16: A scatter plot for the first dataset of Anscombe's quartet.

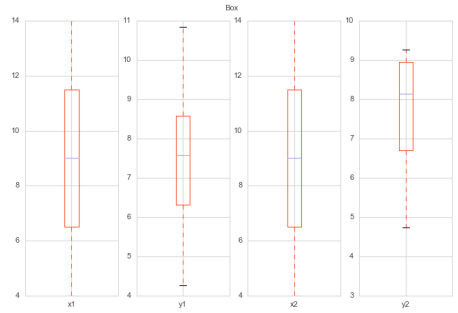


Figure 4.17: A box plot for the first two datasets of Anscombe's quartet.

histogram A histogram splits the values in equally sized bins. It then counts the number of points that fall into each bin. With a histogram one can show the center, spread and skewness of the data. It also shows the presence of outliers. The vertical axis contains the frequency (or counts for each bin) and the horizontal axis shows the bin values. It is also possible to normalize the frequency where it shows a value between 0.0 and 1.0. Another option is to show a cumulitive histogram: the frequency of the bins are its own value plus all the values of smaller bins. Two examples for histograms can be seen in figure 4.18 and figure 4.19.

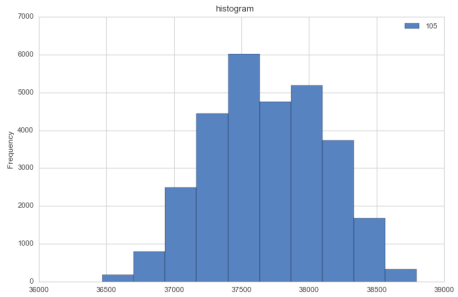


Figure 4.18: A histogram with a clear center.

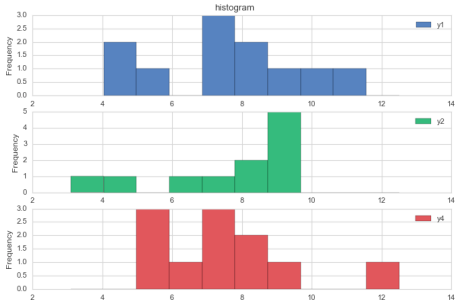


Figure 4.19: Histograms of y1, y2 and y4 from Anscombe's quartet shown below eachother.

4.4.2 Profiling

Data profiling [19] is not a well established or defined research field yet. The following definition can be given: *"Data profiling is the process of examining data available from an existing information source [...] and collecting statistics or informative summaries about that data."*[2] We believe this is an important and useful process while researching data.

Data profiling can be classified into two tasks. The first one is from a single source of data. This can be further broken down into profiling tasks using a single column or multiple columns. The second one is from multiple data sources and how they overlap. ETA only focuses on the first and uses only data from a single data set to profile. The goal is to examine the data set and collect descriptive statistics, such as mean, minimum, maximum, percentile, frequency and provide information about the data such as data types, data length and value patterns, completeness and uniqueness of columns, keys and foreign keys. These results can then be displayed in tables, charts or other visualizations. Since data profiling targets users the visualization of the results is very important. The displayed results give insight into the data set and tells the user about the quality of the data. Using this insight a user can then apply constraints or rules to further processes such as data cleansing or optimization. For instance, inconsistent formatting, missing values or outliers can be detected and transformed or discarded by preprocessing methods, optimized by parameters given by the user using the data profiling insight.
However data profiling comes with its own challenges. Depending on the algorithm, the volume of the data and desired output it could take a large amount of time to analyze the full data set. It is also possible the data set does not fit into memory. A solution to these problems is taking a sample of the data set, however this will not give a complete picture of the data. There are also tools that do not automatically detect dependencies between columns, only user–suggested pairs, making it computationally less complex.

ETA has implemented the following functions for data profiling:

[2] Wikipedia on "Data Profiling", 01/2017

Folder: print This folder contains modules that print their output to the terminal. Many use pandas integrated functions to generate the output.

print A very basic feature which prints the selected data to the terminal. It can be effective for small data sets.

describe Generate various summary statistics. Output depends on the data type of the column.

correlation Compute pairwise correlation of columns. Available methods are pearson (standard correlation coefficient), kendall (Kendall Tau correlation coefficient) and spearman (Spearman rank correlation).

Pandas–profiling Pandas–profiling [20] generates profile reports from a pandas data frame. For each column the following statistics — if relevant for the column type — are presented in an interactive HTML report:

- Essentials: type, unique values, missing values
- Quantile statistics like minimum value, Q1, median, Q3, maximum, range, interquartile range
- Descriptive statistics like mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness
- Most frequent values
- Histogram

Note: Pandas–profiling requires an internet connection to download the Bootstrap and JQuery libraries in order to generate the HTML report.

An example of a pandas–profiling report can be seen in figure 4.20, figure 4.21 and figure 4.28.

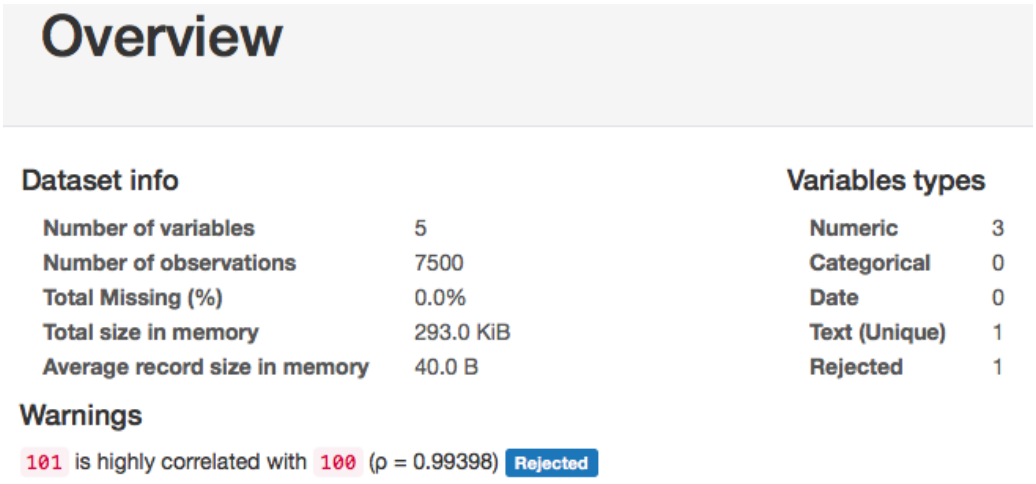


Figure 4.20: Pandas–profiling overview.

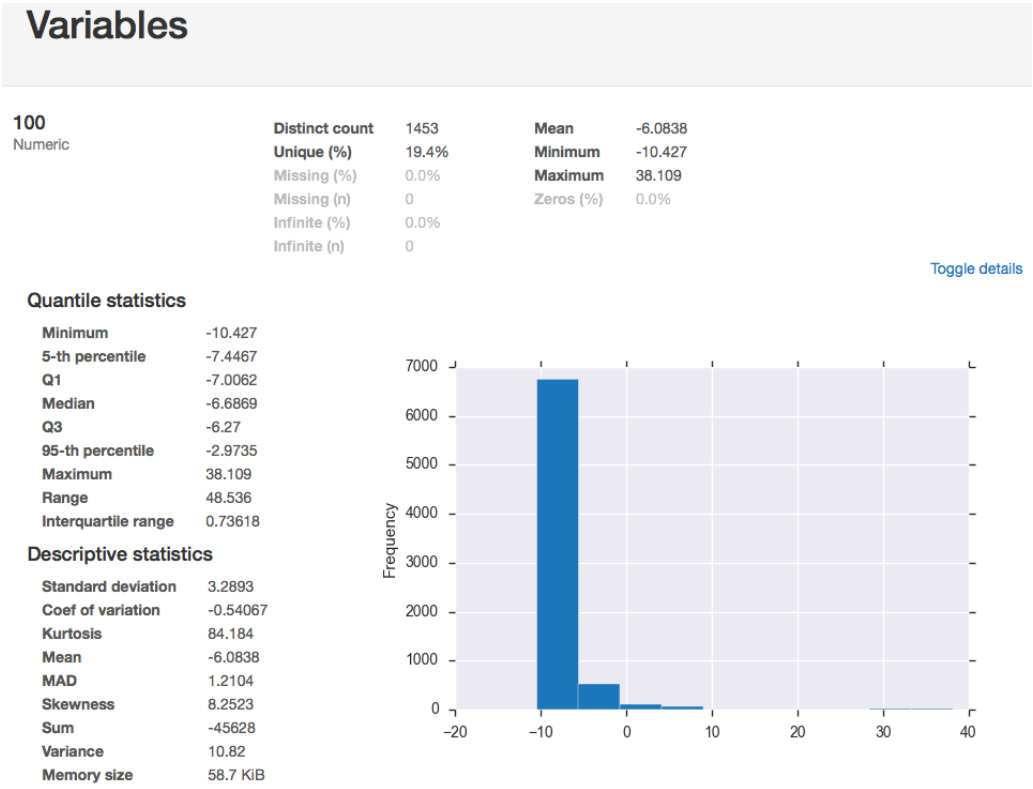


Figure 4.21: Pandas–profiling statistics of a single column.

Plots Using ETA's plotting methods you can visualize data. Although this option is not found under the profiling menu, users can use the plotting methods to profile the data. An example would be the histogram plot which can also be found in the pandas–profiling report.

4.4.3 Preprocessing

Low–quality data may lead to low–quality results. This is why data preprocessing [21] is an important step in knowledge discovery. When collecting data from the real–world, it is very possible inaccurate data is produced. This can come from faulty data collection instruments, human or computer errors at data entry, purposely submitted incorrect data values (e.g. "January 1" for birthday to avoid submitting personal data), errors in data transmission (e.g. age 113 becomes age 13 because the data structure only handles length 0 to 99) or one of many other reasons. It can also be incomplete because values are not inserted, are not available or are deleted. Another problem is that the data might be inconsistent; an example with time series is that dates can come in different formats, e.g. 31-12-2016 or 12-31-16. Inconsistencies can also come from different naming conventions. Say a person is named 'Roberto' but is normally called 'Rob'. When both names are used in the data environment it might not be clear they are from the same person and should be aggregated as such. It is also possible the same record exists twice in the database both using a different name which leads to duplicates. Improving the data quality by dealing with these challenges consequently improves the

results from analyzing the data. It can also improve the efficiency and ease of further steps. To improve data quality a differentiation between four processes is made: *data cleaning*, *data integration*, *data reduction* and *data transformation*. Note that there is much overlap between these processes.

Data Cleaning Data cleaning corrects data problems from the challenges described above. Missing data can be dealt with by ignoring or deleting the records containing the missing data or by filling the missing values. Filling the missing values can be done, for example, with a global constant, the mean value or some other algoritm to find the most probable value. This does bias the data since the value may not be correct. Noisy data (data containing a random error or variance in a measured variable) should be smoothed. Another example is binning, which is used in a histogram. The histogram is discussed under section 4.4.1, plotting. Outliers are also a form of noisy data. They are data objects that differentiate significantly from the rest. Often they are the cause of a measurement error. And finally duplicate data should be removed. Some problems may occur if key values contain duplicates. Say the profit of a day is entered twice, once with the value 5000 and once with 3000. Here the actual value can be one of both, the mean value or even the sum.

Data Integration Data integration [22] involves combining data from different data sources. An example is when two companies want to merge their databases. With the increased amount of data and the need to share the importance of proper data integration has increased. No data integration tools have been implemented in ETA.

Data Reduction Data reduction reduces the size of the data to improve efficiency and reduce complexity while producing (almost) the same analytical results. Dimensionality reduction and numerosity reduction are data reduction strategies. Dimensionality reduction applies data encoding schemes to obtain a reduced or compressed representation of the original data. Numerosity reduction replaces data by alternative, smaller representations.

Data Transformation Data transformations are procedures that try to contribute toward the succes of further processes. Some methods provide better results if the data has been transformed. Normalization scales numerical data to a smaller range, such as values between 0.0 and 1.0. Discretization and concept hierarchy generation replaces raw values of attributes. Discretization replaces the values of a numeric attribute by interval labels or conceptual labels. An example is where age values are replaced by 18–24, 25–30, etc. or *youth*, *adult*, *senior*. Concept hierarchy generation generalizes attributes like *city* to a higher-level concept like *country*.

Many preprocessing techniques have already been developed. However due to the amount of data and complexity of the problems, preprocessing remains an important and active area of research. ETA has some of those techniques implemented:

baseline Baseline creates a constant value based on the mean of the data that is within one sigma from the mean (inliers). The formula used is as follows:

```
mean, std = data.mean(), data.std()
inliers = (mean0 - std0 < data) & (mean0 + std0 > data)
baseline = data[inliers].mean()
```

An example of why this can be useful is that one can calculate the area of a peak by calculating the area between the baseline and the peak as seen in figure 4.22.

euclidean Euclidean calculates the euclidean distance, the distance between two points, between two columns. The formula used is as follows, where *c1* and *c2* are the two columns specified[3]:

```
eu_dist = (data[col1] - data[col2]).abs()
```

standardization Standardization or Z–score normalization transforms the data so the mean value equals 0 and the standard deviation equals 1. Values above the mean value will get a positive value and values below get a negative value. The formula used is as follows:

```
z_score = (data[col] - data[col].mean()) / data[col].std()
```

minmaxscalar MinMaxScalar scales the values of each column to be between the given minimum and maximum value. The formula used is as follows, where maximum and minimum are two values specified:

```
data_std = (data - data.min()) / (data.max() - data.min())
data_scaled = data_std * (maximum - minimum) + minimum
```

An example of this can be seen in figure 4.23.

normalization Normalization scales the values of each column to a value between 0.0 and 1.0. This is the same as using minmaxscalar with minimum 0 and maximum 1.

Some machine learning algorithms require the data to be between 0 and 1 or work better if it is.

clip Clip requires a lower bound and an upper bound from the user. Values lower than the lower bound will get the value of the lower bound and values greater than the upper bound will get the value of the upper bound. Thus all new values are between the lower and upper bound.

rolling Rolling transforms the data using a specified method over a specified window. The window is the number of observations used. For example, it is possible to smooth the data by using the mean value over a certain window. This is a way to reduce noise. The formula used for window size 25 and method mean is as follows:

```
roll_mean = data.rolling(window=25).mean()
```

Other methods are sum, median, min, max and std[4]. An example of this can be seen in figure 4.23.

[3]Specified means that the user gave this input through the use of the parameters.
[4]standard deviation

rolling_correlation Rolling_correlation calculates the correlation between two columns over a specified window. The formula used for window size 10 is as follows:

```
roll_corr = data[col1].rolling(window=10).corr(other=data[col2])
```

resample Resample converts the offset of the datetime index to the specified value. This can range from years to nanoseconds. An aggregation method can also be chosen. The methods are sum, mean, std, sem[5], max, min, median, first and last.

replace_missing Replace_missing replaces missing values. The mean, median or most frequent value can be used to replace missing values.

PAA *Piecewise Aggregate Approximation* (PAA) [23] calculates the mean value of a window and uses that value to describe the whole window. PAA is an example of dimensionality reduction. An example of PAA is shown in figure 4.24.

SAX *Symbolic Aggregate approXimation* (SAX) [24] is a method for representing time series as a symbol. It first uses PAA to devide the time series data into sections and then assigns a symbol to those sections. An example of this can be seen if figure 4.26.

PCA *Principal Component Analysis* (PCA) [25] is used to reduce the dimensionality of a set of data. It is a way to show the structure of the data as completely as possible by using as few variables as possible. An example of PCA is shown in figure 4.25; PCA tries to preserve the interesting peeks of both values (the blue peek on the left and two red peak are somewhat preserved) while retaining the general structure.

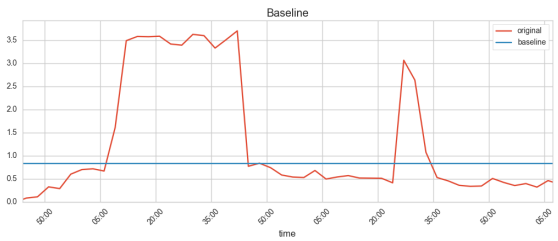


Figure 4.22: An example of peak values with a baseline.

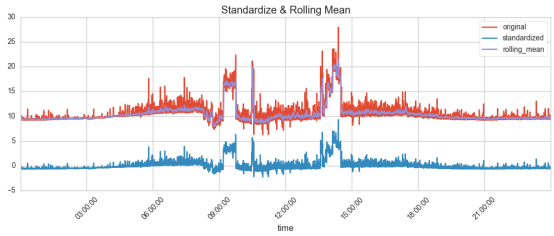


Figure 4.23: An example of a standardization and rolling mean.

In figure 4.27 you can see how various of the preprocessing techniques are applied to a data set. The original data set is the Infrawatch[6] [7] 24 hour data set resampled using the mean value over 3 minutes and was then scaled between the values 0 and 5.

[5] standard error of mean
[6] `http://infrawatch.liacs.nl`

Figure 4.24: An example of PAA.

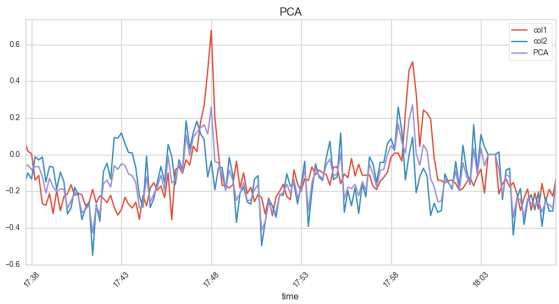


Figure 4.25: An example of a PCA.

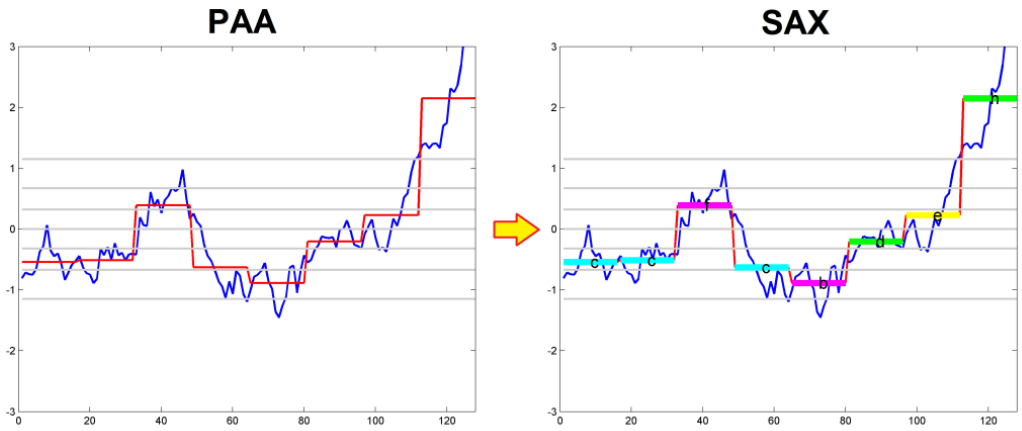


Figure 4.26: An example of SAX [1].

4.4.4 Machine Learning

There is no single definition for machine learning. It can be defined as a branch of artificial intelligence, how to predict the future based on the past, develop methods that can automatically detect patterns in data, or how to construct computer programs that automatically improve with experience. All definitions have in common that the machine "learns" from data, hence the term machine learning. Below the machine learning principles implemented in ETA will be explained.

First, a differentiation between two categories machine learning categories is made: *supervised learning* and *unsupervised learning*.

Supervised Learning Supervised learning learns by example. With supervised learning the data contains a label or attribute that one wants to predict. For supervised learning the algorithm fits a model and then uses that to predict target values.

Normally the data set is split into a training set and a testing set. The user then uses the training set to fit the model and the testing set to test the model. ETA contains the ability to split data into a training and test set. A test size can be specified and the user can randomize the data which does not split the data in order but shuffles it.

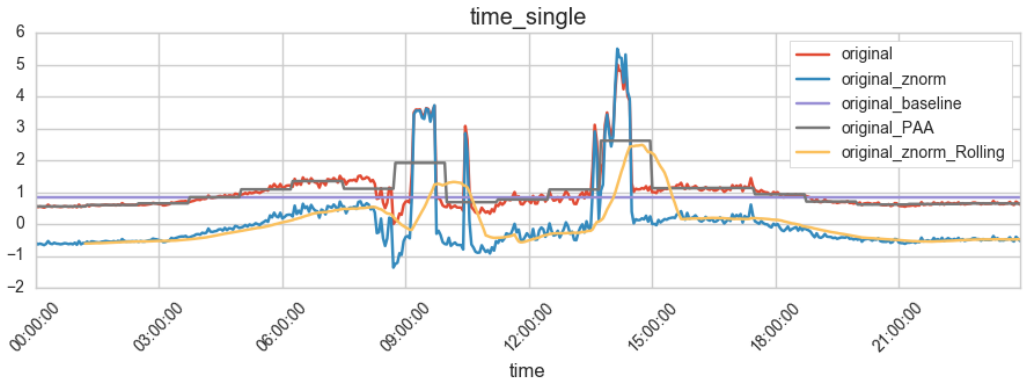


Figure 4.27: An example of various preprocessing modules used on a data set.

A scoring algorithm can then evaluate the model by comparing the predicted values and actual values. This structure for supervised learning is implemented in ETA. Each supervised machine learning module has to contain the following three functions. Each function can have its own parameters which have to be specified in the module.

Fit This function fits the model to the selected data. At least one parameter should be given to specify the target value. The data will then be split into the target data and data without the target. The ETA modules currently implemented have scikit–learn algorithms which use a $fit(X, y)$ method where y is the target value and X the data without the target. This will train a model that will be stored in a pickle file in the */models* folder.

Predict Predict uses the stored model to predict the target values. The user can apply the model on the testing data without the target column selected. The scikit–learn algorithms use a $predict(X)$ method that, given unlabeled observations X, returns predicted values y. The predicted values will be added to the data frame.

Score Score evaluates the model by comparing the predicted values with the actual values. Thus, at least two parameters are needed to specify these values. Depending on the module, scoring methods generate a score based on these values. These scores will then be returned and shown in the terminal.

ETA currently contains two machine learning techniques for supervised machine learning.

Classification The goal of classification is to build a model that can seperate data into distinct classes. A model is built by inputting labelled data. The algorithms learns from this data and creates a model. In this case one can call the model a classifier. The classifier can then be used to predict the label on a different data set. An example is when one has pictures of a dog and pictures of a cat.

A classifier is created by training the model on this data. When this classifier is used on pictures of either a cat or a dog, it will try to correctly identify the data as either being an image of a dog or a cat.

Regression Regression is closely related to classification. Instead of the target being a label it consists of numerical values. An example would be to predict the values of a stock.

Unsupervised Learning In unsupervised learning the data does not contain the target values. It analyzes data which does not include a pre–labeled class. ETA currently supports clustering for unsupervised learning.

Clustering Clustering tries to group data instances together. This is done by trying to maximize the similarity for data instances within a group and minimize the similarity between groups.

Some scikit–learn machine learning methods have been implemented in ETA. The goal of these methods was to show that these machine learning functionalities were possible. Currently these machine learning algorithms are implemented: LinearSVC *(Classification)*, CART *(Classification)*, LassoLarsCV *(Regression)* and K–Means *(Clustering)*. Documentation about these algorithms can be found on the scikit–learn website[7].

Additional machine learning algorithms can be implemented easily using ETA's modular approach. Not only scikit-learn algorithms are supported, one can also implement their own Python algorithms into the ETA system.

[7] `http://scikit-learn.org/`

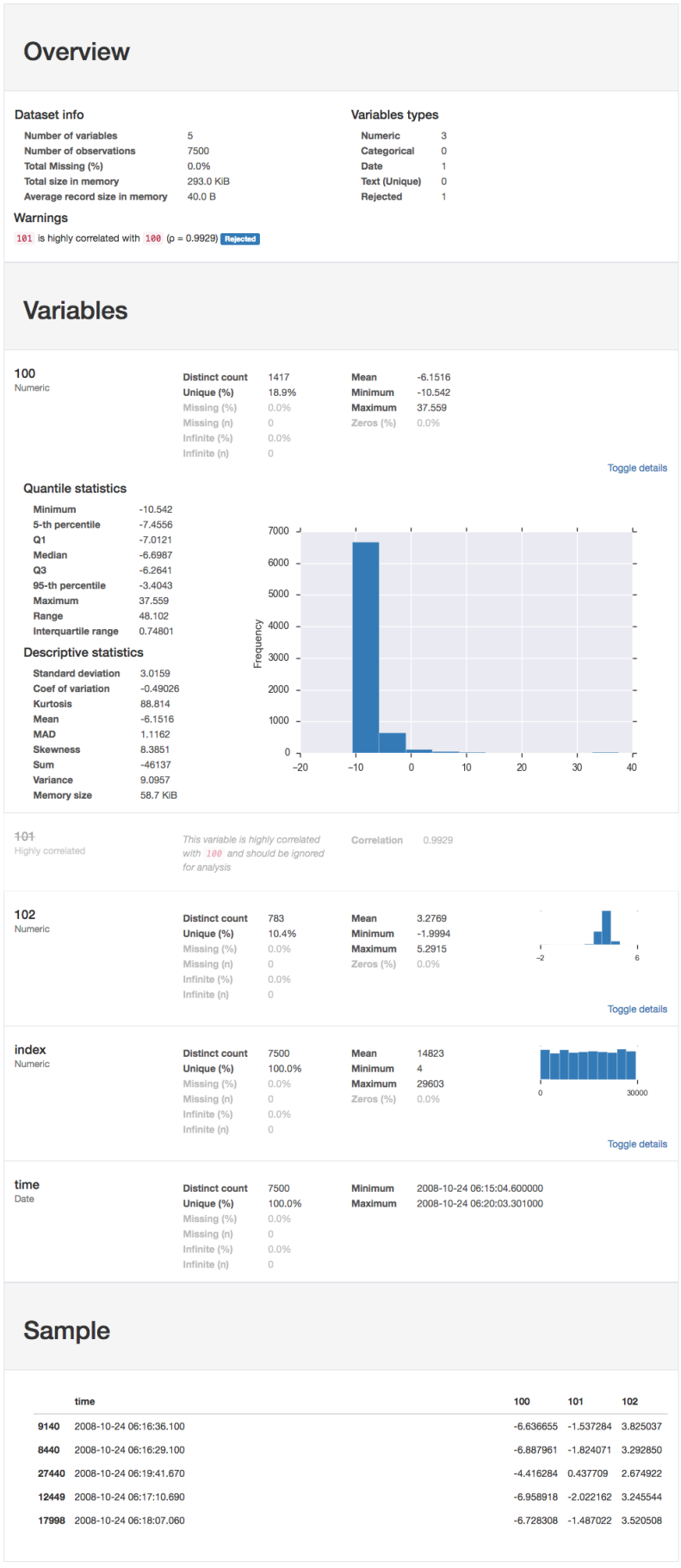


Figure 4.28: An example of a pandas–profiling generated report.

Chapter 5

A guide to ETA

In this chapter we will show how easy it is to use ETA for time series data analysis[1]. Only built–in functionality is used; no coding is required. We will use the *Infrawatch* [26] dataset[2], containing data from sensors installed on the Hollandse Brug in the Netherlands.

First the Python packages need to be installed and updated using *pip*, a package management system. The following lines of code are used:

```
pip install --upgrade pip
pip install -r requirements.txt
pip install --upgrade -r requirements.txt
```

This can also be done with a single command using our shell script:

```
sh run_pip.sh
```

After the required Python packages have been installed and updated. One can now run the program:

```
python program.py
```

We are now greeted with an empty GUI (5.1). The next step is to import a data file. We go to *File* in the menu bar and click on *Open*. This will prompt a file chooser window to appear where we will select our data file (5.2). After the dataset is selected we will see our data as shown in figure 5.3. After renaming (figure 5.4) our data frame and data columns we will end up with the updated data names as shown figure 5.5.

Now we would like to see how this data looks. Select our *time_single* module under the *Plotting* menu which

[1] For this guide macOS Sierra is used as operating system.
[2] From: `http://infrawatch.liacs.nl`

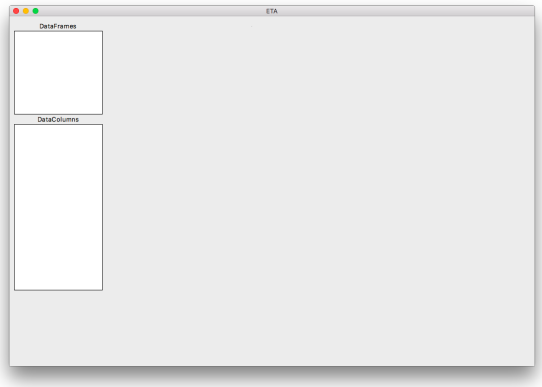


Figure 5.1: ETA's empty GUI.

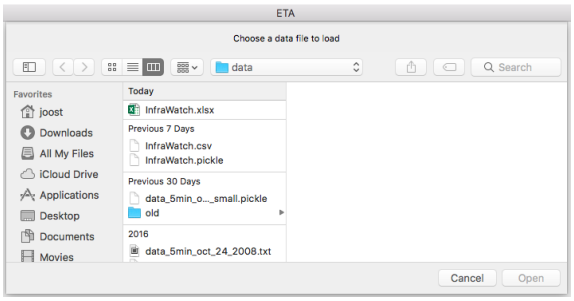


Figure 5.2: ETA's file chooser.

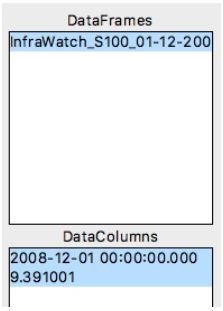
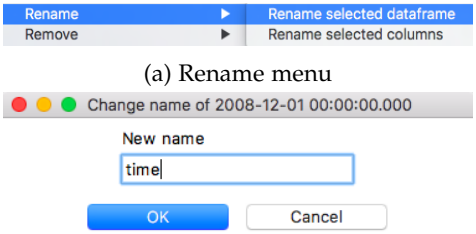


Figure 5.3: Original data set names.



(a) Rename menu

(b) Rename pop–up.

Figure 5.4: Rename process.

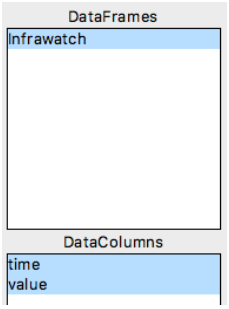


Figure 5.5: Renamed data set.

first prompts us with a parameter selection window (figure 5.6). We select the values as seen in the figure and click on *Ok*. We now have our plot shown inside of the GUI (figure 5.7).

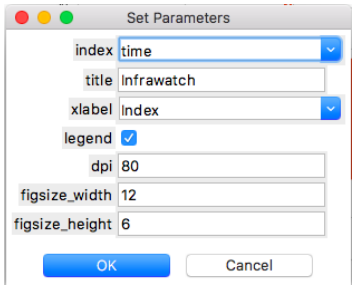


Figure 5.6: The parameter selection window for module *time_single*.

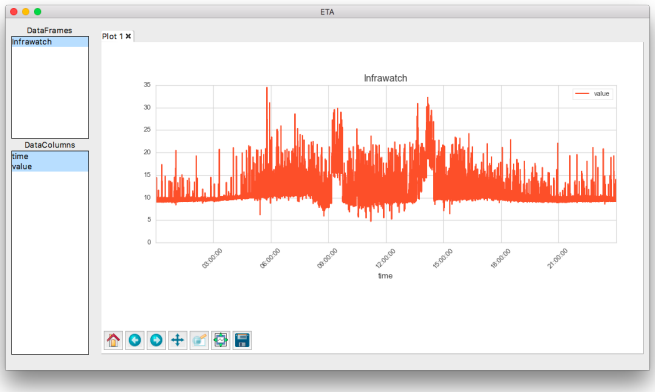


Figure 5.7: A plot of the Infrawatch data using the *time_single* module.

This data looks very noisy. Lets apply a rolling mean algorithm to smooth the data. We select "**method**: *mean*" and "**window_size**: *20*" as our parameters. If we now plot using the *time_multiple* plot, configure the subplots to fit the plot more nicely (figure 5.9) and save the plot using the plot toolbar we get the plot as shown in figure 5.8.

Finally, when we use the *PAA* module with the parameters "**frame_size**: *500*" and **same_length_as_source_df**

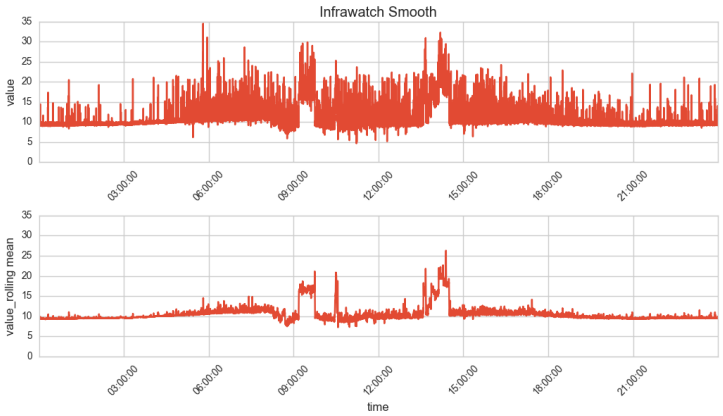


Figure 5.8: A plot using the *time_multiple* module saved with the plot toolbar.

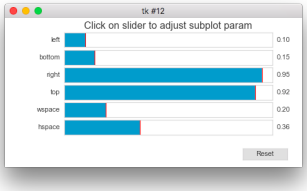


Figure 5.9: The plot toolbar configure menu.

selected and plot the resulting column we get the result in our GUI shown in figure 5.10.

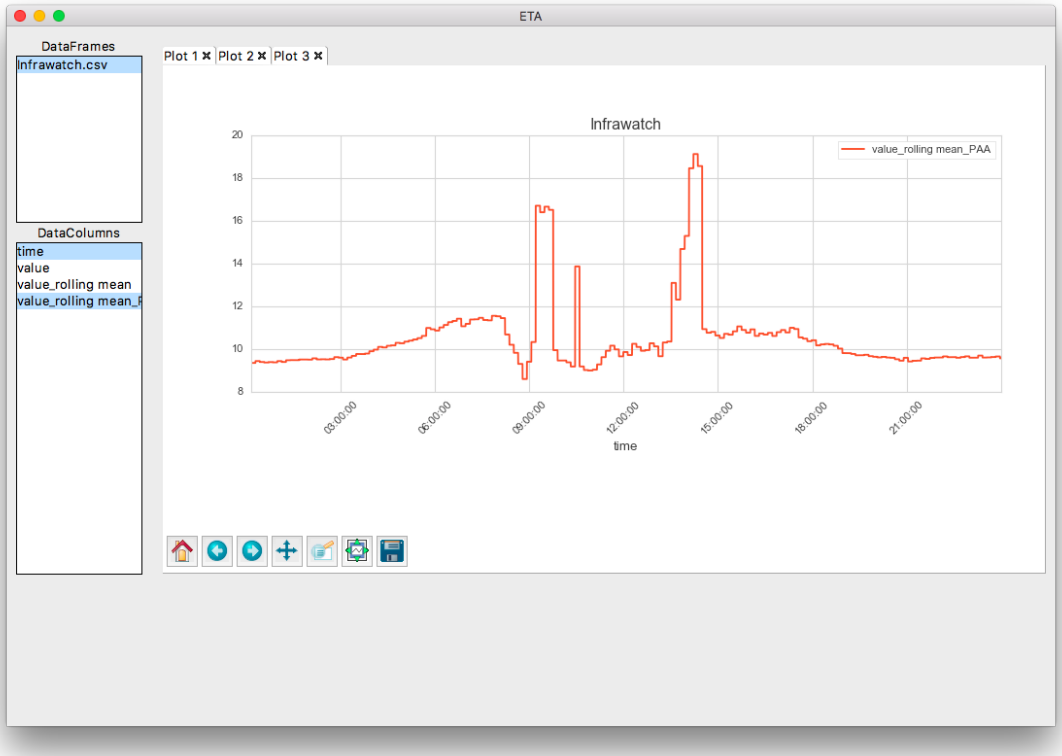


Figure 5.10: The ETA interface after using the steps mentioned in this chapter.

From the resulting plot we can see clearly the parts where it is rush hour. A conclusion one could make is that the morning rush hour starts a bit after 9 and ends a bit before 10, with a small spike after 10, and the afternoon it starts more gradually at around half past 2 reaching its peak hight at around quarter past 2. If one used the plot we started with, it would have been much harder to reach these conclusions.

Chapter 6

Conclusions and future work

Environment for Time series Analysis (ETA) was made to fill a gap in time series software. ETA is an open source[1] time series data analysis platform written in Python. ETA contains a range of functionalities needed to analyze time series data. Data profiling, plotting, preprocessing and machine learning are all included inside the platform. All these functionalities can easily be used by selecting the modules from the menu. No coding is required to use our platform. The user interface uses a nice and clean overview to show the data and results. This makes working with data easier and more comfortable.

ETA also focuses on modularity and expandability. It allows the users to easily create and modify modules. Users can implement their own algorithms and functionality and expand the platform. The user can focus on the implementation of their own algorithm since ETA handles the data, parameters, output and much more. Because Python is widely used in statistics and data mining and contains a lot of useful packages the platform can be further expanded with either implementing more functionality in the source code or creating more modules. An example of interesting machine learning implementations would be Google's *Tensorflow* [27] or the recently released deep learning framework *PyTorch*[2].

ETA provides some modules and therefor functionality already within the platform. We have built a solid foundation for a modular platform that can handle data analysis tasks. However, other functionality is proposed inside the thesis to further improve the platform. New modules with new functionality will provide useful additions to the platform. Modification or creation of modules inside ETA will make this easier. And because the platform should be community driven to take advantage of the expandability of our platform, a way to share and import modules should also be implemented. Speed improvement for handling large amounts of data should be made to make the platform more scalable. For example, the HDF5 file format [28] would allow for better handling of large amounts of data in memory than csv does. Lastly, a lot of time series

[1] Source–code can be found at `https://bitbucket.org/eta_tkinter/eta/`.
[2] See `http://pytorch.org` for more information.

data is generated real time. Implementing data streaming and functionality for it would be an interesting field to explore.

Bibliography

[1] C. Cassisi, P. Montalto, M. Aliotta, A. Cannata, and A. Pulvirenti, "Similarity measures and dimensionality reduction techniques for time series data mining," in *Advances in Data Mining Knowledge Discovery and Applications* (A. InTech, ed.), Available from: InTech, 2012.

[2] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[3] Z. Wang, P. Li, L. Li, C. Huang, and M. Liu, "Modeling and forecasting average temperature for weather derivative pricing," *Advances in Meteorology*, vol. 2015, 2015.

[4] "Nist/sematech e-handbook of statistical methods," Jan 2017. `http://www.itl.nist.gov/div898/handbook/`.

[5] E. Frank, M. A. Hall, and I. H. Witten, "The weka workbench," *Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition*, vol. 2016, 2016.

[6] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, *et al.*, "Orange: data mining toolbox in python," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 2349–2353, 2013.

[7] A. Knobbe, H. Blockeel, A. Koopman, T. Calders, B. Obladen, C. Bosma, H. Galenkamp, E. Koenders, and J. Kok, "Infrawatch: Data management of large systems for monitoring infrastructural performance," in *Proceedings of the 9th International Conference on Advances in Intelligent Data Analysis*, IDA'10, (Berlin, Heidelberg), pp. 91–102, Springer-Verlag, 2010.

[8] J. Alcalá-Fdez, L. Sanchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, *et al.*, "Keel: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, no. 3, pp. 307–318, 2009.

[9] L. Hopman, "Eta: A machine learning oriented platform for high-dimensional time series analysis," 2016. Available at `http://liacs.leidenuniv.nl/edu/bachelorie/bachelorscripties/`.

[10] P. Hughes, "Python and tkinter programming," *Linux J.*, vol. 2000, Sept. 2000.

[11] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[12] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science Engineering*, vol. 13, pp. 22–30, March 2011.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, . Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science Engineering*, vol. 9, pp. 90–95, May 2007.

[15] M. Waskom, O. Botvinnik, drewokane, P. Hobson, David, Y. Halchenko, S. Lukauskas, J. B. Cole, J. Warmenhoven, J. de Ruiter, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, M. Martin, A. Miles, K. Meyer, T. Augspurger, T. Yarkoni, P. Bachant, M. Williams, C. Evans, C. Fitzgerald, Brian, D. Wehner, G. Hitz, E. Ziegler, A. Qalieh, and A. Lee, "seaborn: v0.7.1," Jun 2016.

[16] J. Henstridge and J. Dahlin, "Pygobject," `https://wiki.gnome.org/Projects/PyGObject`.

[17] A. Krause, *Foundations of GTK+ development*. Apress, 2007.

[18] F. J. Anscombe, "Graphs in statistical analysis," *The American Statistician*, vol. 27, no. 1, pp. 17–21, 1973.

[19] F. Naumann, "Data profiling revisited," *SIGMOD Rec.*, vol. 42, pp. 40–49, Feb. 2014.

[20] J. Polfliet, "pandas–profiling: v1.2.0," Aug 2016. `http://github.com/jospolfliet/pandas-profiling`.

[21] J. Han and M. Kamber, *Data Mining: Concepts and Techniques.*, vol. 3rd ed of *The Morgan Kaufmann Series in Data Management Systems*. Morgan Kaufmann, 2011.

[22] A. Doan, Z. G. Ives, and A. Halevy, *Principles of Data Integration.* Morgan Kaufmann, 2012.

[23] C. Guo, H. Li, and D. Pan, *An Improved Piecewise Aggregate Approximation Based on Statistical Features for Time Series Mining*, pp. 234–244. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[24] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, (New York, NY, USA), pp. 2–11, ACM, 2003.

[25] L. I. Smith, "A tutorial on principal components analysis," tech. rep., Cornell University, USA, February

26 2002.

[26] U. Vespier, A. Knobbe, S. Nijssen, and J. Vanschoren, "Mdl-based analysis of time series at multiple time-scales," in *Proceedings ECML-PKDD*, 2012.

[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[28] The HDF Group, "Hierarchical Data Format, version 5," 1997-2017. `http://www.hdfgroup.org/HDF5/`.

Appendix A

Implementing a module

To implement a module you have to add a python file (a file with file-extension .py) in the appropiate modules folder. Upon running ETA it will automatically be added to the menu of the corresponding handler. While ETA is running the user can alter the code and run the module without having to quit the application. Each module must contain the structure of the template; a file is provided for each functionality inside the `\modules\templates` folder. A generalized example of the templates is provided below in listing 1. A user can then modify that structure by putting the parameters required inside of the ordered dictionary in $getParameters()$. It can also put default values of these parameters in the dictionary in $getDefaults()$. Both of these functions are optional. If no parameters are required, no function has to be present. The $run()$ function of the module will be called by the handler, with some parameters provided by the handler. These parameters differ for each handler. The code inside of $run()$ will be executed. This is where the user inserts his own code. Finally $run()$ will return a dictionary called 'returned' to the handler. This dictionary has to have the structure of the dictionary in the template of that functionality.

```
1  import pandas
2  # import your own packages here

3  class Module():
4      def __init__(self):
5          self.moduleName = 'name'

6      def run(self,data,parameters):
7          column = parameters['column1'] # Get your parameters

8          # put your code here

9          returned = {
10             'key1': value1,
11             'key2': True
```

```
        } # the keys and values in this differ for each functionality
        return returned # return above dict

    # Optional, if no parameters are required this can be left out
    def getParameters(self):
        from collections import OrderedDict # for keeping options in order
        settings = OrderedDict([
            # ('KEY', 'OPTION'), THE DICT STRUCTURE
            ('column1', 'columns'), # a column selection from selected columns
            ('str', 'string'), # a string textbox
            ('list', ['Option1','Option2','Option3']), # a dropdown menu of list items
            ('int', 'integer'), # a textbox that gets converted to an integer
            ('flt', 'float'), # a textbox that gets converted to a float
            ('bool', 'boolean')
        ])
        return settings

    # Optional if no default values are desired
    def getDefaults(self):
        defaults = {
            # Put default values of the settings here
            'column1': None,
            'str': 'String',
            'list': 'Option1',
            'int': '100',
            'flt': '200.20',
            'bool': True
        }
        return defaults
```

Listing 1: A generalized version of a module template.