Exploratory Data Analysis, Pt. 3

§

Over the last two parts of this 3-segment Guided Project, we have introduced some interesting concepts around EDA. We've performed statistical analyses on our client's data set and evaluated multiple classification algorithms. We started with a hypothesis, put it to test, refined our beliefs, came up with some conclusions, and now, it's time to face our client.

In this section, we will learn how to build a business client deliverable using LaTeX and Texmaker. We will translate all our results into an elegant report which will help with decision execution. We will close this Guided Project with some recommendations on discussing technical results with a non-technical audience.

The generated plots and test results from the last two segments can be found in the plots and outputs folder respectively.

§

Table of Contents

§

Recalling our results
Generating a technical deliverable

1. Managing expectations

Before jumping right in we need to have something clear: We might have a high-performing model presenting extremely high accuracy results. Still, if we don't communicate our results properly and translate them into actionable insights, there's a chance that the client won't understand the value behind our hard work.

Although things have changed drastically during the last years, it's common for companies to still not have an established team of data scientists and data analysts, specially when talking about small emerging businesses; managing an internal IT department is already expensive, so there is no assurance that our client will have a dedicated research team behind the curtains. And even if this is the case, there is no guarantee that they will be available for our project.

This is why we need to ensure that we're translating our results to a business-oriented language and not a research-oriented one as a complement to our technical deliverable, specially if we are presenting our findings to a commercial team, which is often the case.

Thankfully, there are some really nice tools out there that will help us achieve our goal.

2. An introduction to LaTeX & Texmaker

LaTeX is a software system for document preparation based on TeX, which can render complex mathematical formulae and multiple objects such as tables, headers, images, plots and other figures. It's the go-to language for scientific and technical journaling because of its flexibility, active community and beautiful document generating capabilities.

There are multiple TeX distributions for different applications, the most common ones being TeX Live, MiKTeX, and MacTeX. They differ mainly in their package content and platform support.

Texmaker is a widely adopted, cross-platform, open-source LaTeX editor with an integrated build tool and PDF viewer.

LaTeX syntax is not easy; it presents a steep learning curve specially if we are to explore all of its functionalities and packages. The good news is there is a vast collection of templates for multiple purposes already available. Such templates usually include a LaTeX source file `.tex`, a LaTeX class definition file `.cls`, a compiled `.pdf` example, and required digital assets such as `.png`, `.jpg` or `.pdf` files.

The nice thing about templates, is that they're fully customizable; we can download a `.cls` class file and tailor it to our needs until we have exactly what we're looking for. At first it takes some time, but in the end it's really worth it.

3. Why LaTeX?

There are tons of word processors out there that would presumably make it easier for us to generate a client deliverable. In contrast, LaTeX requires us to install a TeX distribution and a dedicated text editor. Also, LaTeX syntax has a learning curve associated. This seems too much trouble just to generate a simple text document, so why bother using LaTeX?

Well, let's enumerate some of the things we will need to include on this report, and see if LaTeX offers any advantage over other methods such as Microsoft Word or Markdown:

3.1 Basic header and paragraph formatting

On **Microsoft Word**, we need to type a header and then format it as header using a menu. This could be convenient but we can sometimes lose track of which header tag we used for each case, since there's no explicit code showing us this. Also, if we copy our headers, there's no guarantee that the formatting will translate when pasting on other word processors. For paragraph formatting, we need to make sure we're setting the properties such as spacing correctly, again, by using a menu. Also, when copying and pasting, formatting could get lost.

On **Markdown** it's much easier; all we have to do is define headers by using a hash `#` symbol. H1 will have a single hash prepended, H2 will have two hashes prepended, and so on. We can easily visualize this syntax using any text editor, and when copying and pasting content, the formatting will stay the exact same (*provided the target supports Markdown*). The problem with Markdown, is that it does not support formatting inside the actual document; we would have to create separate `.html` & `.css` files in order to customize our header styles.

On **LaTeX** it gets even better; we can define our own custom section formatting rules, and use them by simply making the calls when writing our document:

We can define a new `section` style on our style sheet:

Code

```
{\section*{\contentsname}}
```

We can style the title format:

Code

```
\titleformat{\section}
    {\relax}{\textsc{\MakeTextLowercase{\thesection}}}{1em}{\spacedlowsmallcaps}
```

Then, we can simply define a new section, and input the title:

Code

```
\section*{Abstract}
```

It will get formatted with the parameters we defined on our style sheet.

3.2 Tables

Microsoft Word makes it very straightforward to include tables in a document; we just have to specify its dimensions, insert the figure, input the values, and we're done. While it's a very straightforward process, the formatting options are limited, and everything is done via our beloved menu. This is suboptimal and can take more time if we're looking to extensively customize out figure.

Markdown also offers table support. We can use the following syntax:

Code

```
| Header 1 | Header 2 | Header 3 |
| -------- | -------- | -------- |
| Entry 1  | Entry 2  | Entry 3  |
| Entry 4  | Entry 5  | Entry 6  |
```

Output

Header 1	Header 2	Header 3
Entry 1	Entry 2	Entry 3
Entry 4	Entry 5	Entry 6

This process is far from optimal, extremely limited, and often times makes it impossible to follow where we're writing since the pipes move whenever we're inputting values in a cell. There are some plugins for certain editors that will help, such as Advanced Tables for Obsidian, but still, what we can do is very limited. Also, we cannot natively import a `.csv` table into a Markdown table; there are some web-based applications that can do this for us, *e.g.* *CSV to Markdown Converter*, but it's simply not functional and scalable having to use an external program to convert every table we generate.

LaTeX offers multiple ways to generate tables. We can generate them from scratch:

Code

```
\begin{table}[hbt]
\caption{Table of Grades}
\centering
\begin{tabular}{llr}
\toprule
\multicolumn{2}{c}{Name} \\
\cmidrule(r){1-2}
First name & Last Name & Grade \\
\midrule
John & Doe & $7.5$ \\
Richard & Miles & $2$ \\
\bottomrule
\end{tabular}
\label{tab:label}
\end{table}

Reference to Table~\vref{tab:label}
```

3.3 Images

Microsoft Word lets us include images by importing them. This is undoubtedly one of the most painstaking processes in the whole Word ecosystem; we cannot include an image by just specifying its path, we need to import the actual image. Also, if we need to include nice-looking captions, we need to create a table, insert our image on the top cell, and our caption on the bottom cell. To make things worst, formatting the picture often times leads to inconsistent outputs if we're not careful.

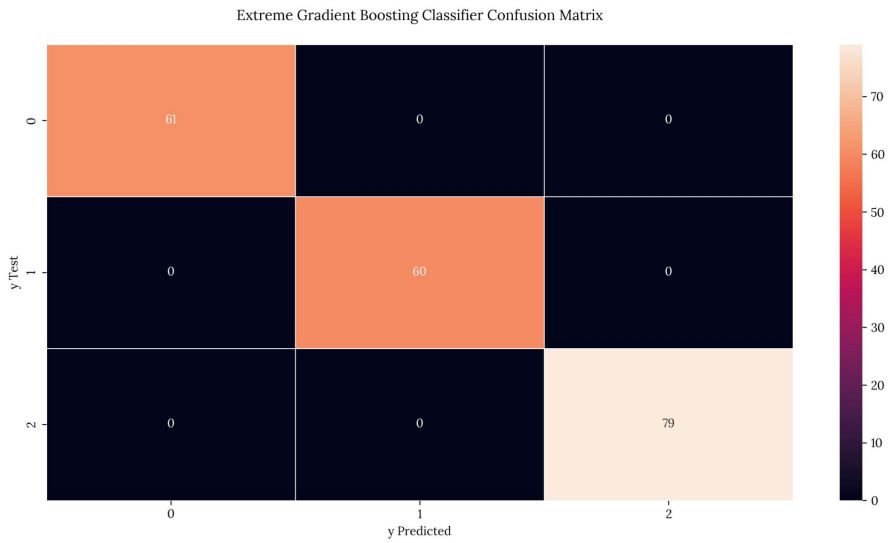
Markdown offers two ways to include images in our document.

We can include them inline:

Code

```
![Image](path_to_image)
```

Output

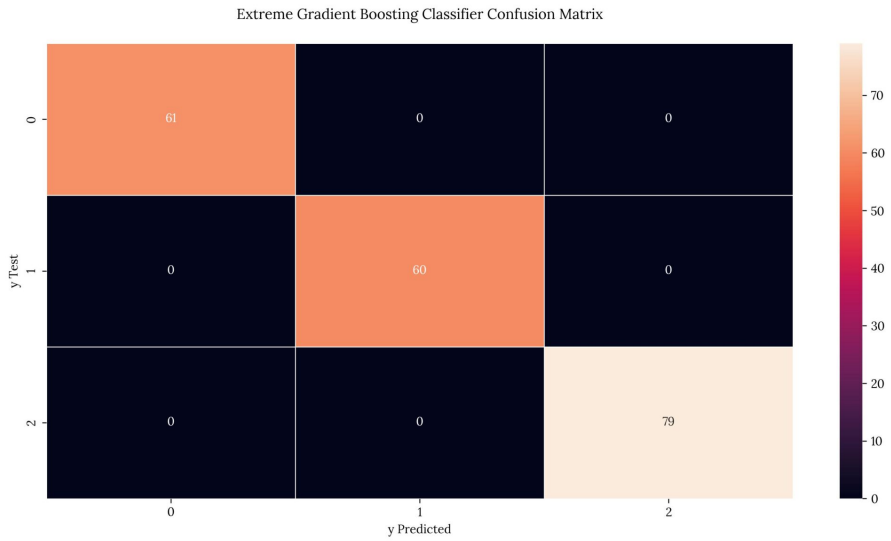


We can also use HTML:

Code

```
<img src="path_to_image" alt="Image" />
```

Output



This is convenient if we want to specify custom dimensions, but again, customizability is limited, *e.g. we cannot create an image gallery using simple Markdown*.

LaTeX offers multiple ways to include images in our document. We can include a single image:

Code

```
\begin{figure}[tb]
\centering
\includegraphics[width=0.5\columnwidth]{GalleriaStampe}
\caption[An example of a floating figure]{An example of a floating figure (a reproduction from
the \emph{Gallery of prints}, M.~Escher,\index{Escher, M.~C.} from
\url{http://www.mcescher.com/}).}
\label{fig:gallery}
\end{figure}
```

We can even create an image gallery with custom layout, and dimensions:

Code

```
\begin{figure}[tb]
\centering
\subfloat[A city market.]{\includegraphics[width=.45\columnwidth]{Lorem}} \quad
\subfloat[Forest landscape.]{\includegraphics[width=.45\columnwidth]{Ipsum}\label{fig:ipsum}} \\
\subfloat[Mountain landscape.]{\includegraphics[width=.45\columnwidth]{Dolor}} \quad
\subfloat[A tile decoration.]{\includegraphics[width=.45\columnwidth]{Sit}}
\caption[A number of pictures.]{A number of pictures with no common theme.}
\label{fig:esempio}
\end{figure}
```

Here, we previously defined a `Figures` folder which contains all the figures for this document. We simply reference them by their name, without any need for path and extension specification. Clean, simple and straightforward.

3.3 Mathematical expressions

Microsoft Word offers an equation handler where we can write mathematical formulae by selecting each mathematical symbol from a menu. This process can become tiresome after our 6th equation. Fortunately, Word also offers LaTeX integration; we can insert a new equation and then select LaTeX mode, where we'll be able to write using LaTeX syntax. The problem is, we have to do this every time we insert a new expression. Also, if we write an equation using the default Word method, translating that into LaTeX code frequently results in syntax errors. In short, Microsoft Word was not created for mathematical writing purposes.

Markdown offers LaTeX compatibility, but it's not consistent between all editors; some natively support it, some require additional plugins, some offer inconsistent previewing, and others even require different expression enclosures. Also, we cannot install LaTeX packages to extend its functionality on Markdown. Obsidian is a great Markdown editor option since it supports LaTeX writing and previewing out of the box, but is limited in what it can perform.

This field is where **LaTeX** excels. We might have noticed that the two previous options support LaTeX as their mathematical expression input language, and that's for a reason; LaTeX is the gold standard for mathematical writing. It supports every symbol, operand and Greek letter available on the mathematical language. If we're missing some exotic symbol with the default distribution packages, we can always install new ones thus extending the functionality. We can even use FontAwesome custom icons out of the box by simply calling them from inside our document.

3.4 Everything else

It is true that LaTeX syntax presents a considerable learning curve, but once we master it, a beautiful document generation heaven awaits (*in the most literal sense, since LaTeX documents are can be very aesthetical if done right*). It provides us with considerably more customizability than other word processors, there's no arguing about that; the Comprehensive TeX Archive Network (*CTAN*) currently hosts about 4,000 packages, meaning endless possibilities. From diagramming to drawing little Marmots in TikZ, LaTeX can tackle everything we throw at it.

Now that we're convinced (*hopefully*) that LaTeX is our best friend, we can continue downloading everything we will need in order to make our client shriek with delight.

4. Preparing our environment and downloading a template

For this segment we will use the TeX Live distribution along with the Texmaker editor. We will also download a template from LaTeX Templates, a phenomenal website providing free, fully-fledged material.

This tutorial will be specifically oriented towards Windows, but can easily be tailored for macOS or Linux operating systems.

We will start by installing our **Tex Live** distribution:

- Head to the Tex Live official website.
- Select *install on Windows* from the *Concise instructions, per platform* section.
- Head to the Easy install section.
- Download the `install-tl-windows.exe` executable and run it on your machine. (*Please make sure to read the official documentation carefully before installing*).
- Wait for the installation process to complete (*depending on the mirror selected, it could take several minutes, so make sure a mirror close to your current location is selected*).

We will then install **Texmaker** for Windows:

- Head to the download page.
- Select the *Desktop msi installer for windows 7/8/10/11 64 bits* package.
- Run the executable on your machine.

Upon conclusion, we should end up with the Tex Live distribution and the Texmaker application installed.

We will then download our template:

- Head to the Arsclassica Article template page.
- We can view the PDF preview by heading to the *Preview Template PDF* section. This is what our actual document will look like out of the box, but not to worry, we will fully customize it.
- Click the *Download Template Code* link.
- A `.zip` file will download. We will then need to extract its contents.
- Once we're done, we can create a new directory named `client_report` and paste all of our template's contents.
- We should have the following:
  - `Figures`
  - `article_4.tex`
  - `sample.bib`
  - `structure.tex`

We will open our template in Texmaker to make sure everything's working fine:

- Open the Texmaker application.
- Head to *File*, *Open* and select the `article_4.tex` file.
- We should end up with the `.tex` file displaying on the left panel, and a blank canvas on the right panel. This canvas is where the `.pdf` file will be displayed once we compile our document.
- To compile our document, head to *Quick Build* option on the top panel and select *run*.
- Our document will be compiled for the first time, and if everything went fine, two things will happen:
  - A `.pdf` file along with several other files will be generated in our working directory.
  - The compiled file will appear on the right panel.
- LaTeX documents have to be re-compiled each time we want to visualize our final `.pdf` document. The good thing is that this is going to take less time on next iterations.

We could use this template as-is by simply editing our `article_4.tex` file. This would not be a bad idea since our template has almost everything we require out-of-the-box. Still, we want to modify certain aspects of our document to make it truly our own. For that, we will need to locate our `structure.tex` file and open it in any text editor. VS Code offers a nice extension we can download here for LaTeX language support. This will enable syntax highlighting, formatting and other useful features that will make it easier for us to manipulate `.tex` files.

We are now ready to start customizing our template.

3. Designing our layout

If we take a close look at our `structure.tex` file, we can see that it's already nicely divided by sections.

The first section imports the required packages:

Code

```
\usepackage[
nochapters, % Turn off chapters since this is an article
beramono, % Use the Bera Mono font for monospaced text (\texttt)
eulermath,% Use the Euler font for mathematics
pdfspacing, % Makes use of pdftex' letter spacing capabilities via the microtype package
dottedtoc % Dotted lines leading to the page numbers in the table of contents
]{classicthesis} % The layout is based on the Classic Thesis style

\usepackage{arsclassica} % Modifies the Classic Thesis package

\usepackage[T1]{fontenc} % Use 8-bit encoding that has 256 glyphs

\usepackage[utf8]{inputenc} % Required for including letters with accents

\usepackage{graphicx} % Required for including images
\graphicspath{{Figures/}} % Set the default folder for images

\usepackage{enumitem} % Required for manipulating the whitespace between and within lists

\usepackage{lipsum} % Used for inserting dummy 'Lorem ipsum' text into the template

\usepackage{subfig} % Required for creating figures with multiple parts (subfigures)

\usepackage{amsmath,amssymb,amsthm} % For including math equations, theorems, symbols, etc

\usepackage{varioref} % More descriptive referencing
```

The next section defines theorem styles:

Code

```
\theoremstyle{definition} % Define theorem styles here based on the definition style (used for
definitions and examples)
\newtheorem{definition}{Definition}

\theoremstyle{plain} % Define theorem styles here based on the plain style (used for theorems,
lemmas, propositions)
\newtheorem{theorem}{Theorem}

\theoremstyle{remark} % Define theorem styles here based on the remark style (used for remarks
and notes)
```

The last section defines styling properties for hyperlinks:

Code

```
\hypersetup{
%draft, % Uncomment to remove all links (useful for printing in black and white)
colorlinks=true, breaklinks=true, bookmarks=true,bookmarksnumbered,
urlcolor=webbrown, linkcolor=RoyalBlue, citecolor=webgreen, % Link colors
pdftitle={}, % PDF title
pdfauthor={\textcopyright}, % PDF Author
pdfsubject={}, % PDF Subject
pdfkeywords={}, % PDF Keywords
pdfcreator={pdfLaTeX}, % PDF Creator
pdfproducer={LaTeX with hyperref and ClassicThesis} % PDF producer
}
```

This is actually a really simple & straightforward template, mostly due to the fact that it's using the `classicthesis` as underlying layout.

We will start by making some modifications to our structure:

Code

```
Aa
```

3.1 Executive summary

3.2 Business guide

3.3 Plot results

3.4 Tabular results

3.5 Method considerations & limitations

3.6 Conclusions & recommendations

3.7 Appendix

§

Conclusions

§

References

- A
- A

§

Copyright