

# Example to plot directly into latex

19-10-2019

## 1 Introduction

Welcome, this document presents our market analysis for the TruCol consultancy. The objective of this document is to provide some basic insight into the order of magnitude of the potential of the TruCol consultancy to generate returns for its potential investors. Based on various pitch templates, [?], and private communications, we intend to convey this information through sharing our model and estimate of the following market parameters for the TruCol consultancy:

- **Total addressable market (TAM)**, or total available market, is the total market demand for a product or service, calculated in annual revenue or unit sales if 100% of the available market is achieved[?].
- **Serviceable available market (SAM)** is the portion of TAM targeted and served by a company's products or services[?].
- **Serviceable obtainable market (SOM)**, or share of market, is the percentage of SAM which is realistically reached[?].

Since we currently have little experience on this topic within our team, we are making our data and assumptions as transparent as possible, both in this document as in our code. This way we hope to improve our model based on your feedback by enabling you to experiment with it yourself. Additionally, because the market analysis consists of a rough estimate, three different estimation methods are considered for generating the TAM, SAM and SOM estimates. The redundancy is introduced in an attempt to establish some frame of reference within the results. The assumptions and data points for the respective models are specified explicitly in this document with an identifier, and where relevant, this identifier is used in the comment of the code to link the document and code.

The models are described in ?? (the Python models themselves are included as appendices in ?? to ?? respectively). The results of these models are presented in ??. To shed some light on how sensitive the model is to for example changes in assumptions, a sensitivity analysis is presented for each model in ??. Next the results and sensitivity of the models are discussed in ?? and a conclusion is provided in ??.

We invite you to tinker with the assumptions and models yourself! The data and plots in this report are automatically updated if you run `python -m code.project1.src`. If you experience any difficulties in running the code, simply reach out to us, (click on issues on the github page) and we are happy to get you running the code.

## 2 TruCol Consultancy Business Model

Since the market size estimation models are somewhat of an abstract/subjective task, three different approaches are used in an attempt to establish some reference material with respect to accuracy.

Before the model is presented, it is important to realise that we propose a consultancy service that operates as an optimisation service. This means that if a certain activity, e.g. a logistics company has operational cost of 5 \$million/day, our consultancy service is only able to earn at most the margin of improvement we are able to bring our customer. Suppose the independent usage of the TruCol provides the customer with a 2% optimisation in their operational costs, yielding them  $5.000.000 \cdot 0.02 = 100.000/day\$$ . Suppose our expertise is able to enable them to yield a 3% optimisation by identifying the relevant development/system processes and supporting them in improved test specification. In that assumption our consultancy would bring them an additional  $3-2=1\%$  which would translate roughly to 50.000\$. That would be the value we bring to the logistics company in this hypothetical scenario.

In reality this example is oversimplified, the 2% the company could get by themselves would involve some risk pertaining to inaccurate test specification which could lead to loss of the bounty. Our company reduces this risk by providing test-specification security expertise. Furthermore, our interaction with the client may bring the client experience that can be applied in future applications of the TruCol protocol, hence the value to we bring to the client is larger than the amount they gain in terms of optimisation w.r.t. the case where they use the protocol themselves.

## 3 Markets

To use the Top Down Model, ?? describes the TAM in which the TruCol consultancy will operate. To this end, ?? discusses the market size of the logistics market, and the profit within that market. Due to time-constraints and lack of data, some datapoints and assumptions of the logistics market are applied to other sectors such as the automated trading market and pharmaceuticals market in ?? to get some insight in their respective market sizes. Furthermore, ?? provides some qualitative insight in the potential future markets that are highly suited for the TruCol protocol. These emerging markets are accordingly expected to be relevant markets to address in the near future.

### 3.1 Total Addressable Market

To compute the TAM, SAM and SOM, some form of market definition can be used. To this end, it is considered valuable to specify what the TruCol consultancy does, where it adds value and how it does that. Furthermore, since these three aforementioned estimates pertain to a potential future, the potential, yet deemed feasible, activities of the TruCol consultancy are included.

The TruCol consultancy provides advice and support to companies on how they can get the most out of the TruCol protocol. To understand this the following assumptions are shared. Under these assumptions, one can conclude that an economically rational company would try to off-load as much of their required tasks into the TruCol protocol as it would minimise their operational costs and/or improve algorithmic efficiency of their solutions.

- **asu-0:** Solutions to tasks that are completed using the TruCol protocol are deterministically verifiable.
- **asu-1:** Solutions to tasks that are completed using the TruCol protocol are of sufficient quality.
- **asu-2:** Tasks that are completed using the TruCol protocol can be solved for the lowest cost price that is currently available in this world.
- **asu-3:** No personnel needs to be attracted, screened, hired nor fired for tasks that are completed using the TruCol protocol.
- **asu-4:** Companies can benefit from public particular solutions to their task specifications.
- **asu-5:** By sampling from a bigger talent pool (this world), the average performance of the solutions will be better than what is produced by the in-house talent pool, or, for equal solution performance, a faster rate of development can be obtained on average for an equal or lower price.

We help companies identify the tasks for which they can use the TruCol protocol, and we assist them in writing safe test specifications that are not easily hackable. This implies that under the given set of assumptions, the TAM for the TruCol protocol can be defined as the total costs that the companies (and consumers) in this world are willing to pay for assistance on using the TruCol protocol.

#### 3.1.1 TruCol Total Addressable Logistics Market

This sub-sub section illustrates a rough method of estimating the logistics sub-segment of the TAM for the TruCol protocol. To do this, an example of algorithmic optimisation within the logistics market as presented by McKinsey & Company is generalised conservatively to a rough estimate of the total logistics market size.

A clear example of a logistics company successfully hiring a consultancy for algorithmic optimisation is documented by McKinsey & Company in the "how they help their clients" segment of their website[?]. The study how reports McKinsey's team, among which the McKinsey's Strategic Network Analytic Center helped an Asian logistics company. With McKinsey's team, the logistics company realised an *in line haul network cost* reduction of 3.6% while reducing their *transit time* with 0.8%, yielding an overall 16% increase in profit for the logistics company, without compromising the quality. To use this report as a valuable resource to generate some rough estimates on market size, the following assumptions are made:

- **asu-6:** The logistics company made a net profit by hiring McKinsey & Company in this particular ordeal.
- **asu-7:** The example of a 16% increase in profit is generalizable to a conservative potential 0.1% of profit increases through algorithmic optimisation accross the entire logistics industry.
- **asu-8:** Companies are willing to pay at least 1 % of their potential profit increases for the assistance the TruCol consultancy company provides in identifying opportunities for optimisation and for improving test-specification security.

Based on those assumptions, one could find a potential yearly profit increase accross the entire logistics sector by summing the net profit of the logistics sector. [?] claims that this company [?] valued the logistics market at 8.1 trillion in 2016. Additionally [?] claims [?] estimates the logistics market value will grow to 15.5 trillion in 2023. However, no figures on profit are found. Hence individual companies are explored.

For DHL one can find on pdf page 37/170 in [?] that the annual profit for DHL in 2019 was 4.1 billion.

For UPS one can find on pdf page 4/257 in [?] that the annual unadjusted operating profit for UPS in 2020 was 7.7 billion. Note, [?] says UPS had a net operating profit of 1.1 billion in Q1 of 2020, implying they had to almost double their average profit in the remaining three quarters of 2020 to be consistent with an annual 7.7 billion.

For FedEx the net income as reported for 2020 has been 1.29 \$ billion in pdf page 2/17 [?].

- **Asu-9:** The net income as reported (GAAP) by FedEx can be interpreted as the profit by FedEx.

Next, the claim that fragmentation of the global market implied in 2016 that Deutsche Post DHL, Ceva Logistics, UPS, and FedEx, control less than 15% of that global market allows estimating a limit on the net global profit made in the logistics market based on the following assumptions:

- **Asu-10:** The market segment in the global logistics market maintained by the combination of DHL, UPS and FedEx is at most 15% in 2020.
- **Asu-11:** The profit in the remaining 85% of the global logistics market has the same average yearly profitability per percent market share as the combination of DHL, UPS and FedEx.

Based on assumptions 1-11 one could estimate an upperbound of

$$\begin{aligned}
 net - profit_{DHL+UPS+FedEx} &= 4.1 + 7.7 + 1.29 = 13.09billion \\
 \frac{net - profit_{global\_logistics}}{net - profit_{DHL+UPS+FedEx}} &= \frac{0.85}{0.15} \\
 net - profit_{global\_logistics} &= net - profit_{DHL+UPS+FedEx} \frac{0.85}{0.15} \\
 net - profit_{global\_logistics} &= \frac{13.09 \cdot 0.85}{0.15} \\
 net - profit_{global\_logistics} &= 74.2billion
 \end{aligned} \tag{1}$$

Hence, if each of those companies in the logistics sector could increase their profits on average annually by .1% using algorithmic optimisation, and if they would use the TruCol protocol to do that, and if they would be willing to invest 1% of that profit in our support and assistance in getting the most out of the TruCol protocol, we would currently estimate that this would yield roughly an income of  $74.2 \cdot 0.001 \cdot 0.01 = \$0.74million$

### 3.1.2 Additional addressable markets

Since the TruCol consultancy is market agnostic, we also seek to assist in algorithmic optimisation outside the logistics market. Several markets are worth mentioning in particular as we expect them to either heavily rely on algorithmic optimisations, or because they are particularly suited for the TruCol protocol.

- **(Automated) trading** In the highly competitive market of (automated) trading, algorithmic optimisations are key to making successfull trades.
- **Space Sector** The space engineering sector already has a relatively high test driven development[?], this lowers the adoption costs of the TruCol protocol relative to most industries. Furthermore, space applications are heavily mass constrained, which generally makes them highly energy constrained as well. These energy constraints emphasise the importance of algorithmic optimisations, for example in telecommunications satellites and swarm robots.
- **Innovative Materials Research** The domain of material science has been adopting algorithmic search strategies to find new materials [?].
- **Pharmaceutical Industry** Another example of a large market that has been shifting to adopt algorithmic search strategies to find new medicines.

Each of these are multi billion dollar markets which can contribute to the TAM of the TruCol consultancy.

### 3.1.3 Emerging markets

Beyond those listed markets, the following emerging markets could be great opportunities for the TruCol consultancy to latch in and grow along in.

- **Neuromorphic Computing** This field is developing new complexity theory to adapt to the unconventional computation methods. This is an interesting opportunity to explore the versatility of the TruCol protocol.
- **Quantum Computing** This is another upcoming field with many new algorithmic implementations. The newness of the field may suggest that the amount of optimisation and exploration to be done is relatively high, possibly indicating a relatively large potential for the TruCol protocol. However, currently our team does not yet contain experience in this type of algorithmic developments.
- **Artificial Intelligence** With the introduction of GPT-3 the world has seen an example of an AI engine that is able to generate code for some basic tasks [?]. The TruCol protocol could catalyse the usage of such AI engines that are able to write code based on requirement specification. We expect that users of the TruCol protocol will develop a tactical advantage on requirement specifications for AI engines.

## 3.2 Market Trajectory

## 4 Market Analysis Model

This section describes the model that is used to perform the market analysis for the TruCol consultancy service. The model will be used to estimate the yearly revenue that is projected for this consultancy company. Typical estimation models to do this for startups are:

- *Top Down Model* - Starts with a large population with known size that make up the target market, and then narrows the market size down to the specific market segment.
- *Bottom Up Model* - Takes current pricing and/or usage of product as starting point and extrapolates up/outwards to compute the potential market size.
- *Value Theory Model* - estimates the value provided to customers and estimates how much of that value can be reflected in the product pricing.

Since the TruCol consultancy does not yet have a large body of current pricing and product usage, the Bottom Up Model is not used in this market analysis document. Similarly, the Value Theory Model is omitted in this market analysis as it is most powerful on historical data which is not yet available for the TruCol protocol. Since the market sizes of most sectors in which the TruCol consultancy intends to operate, the Top Down Model is used to derive a rough estimate of the projected yearly revenue for the TruCol consultancy.

### 4.1 Top Down Model

## 5 Results

### 5.1 Top Down Model

The code listed in the appendices generated the following estimates for the total addressable market sizes for the TruCol consultancy.

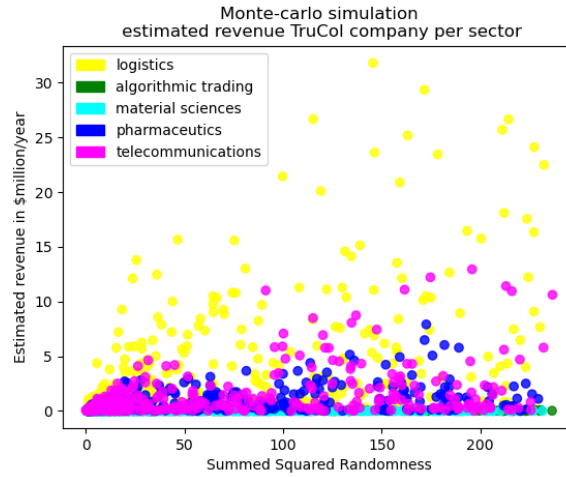


Figure 1: A scatterplot generated by a Monte-Carlo simulation to provide an impression on the estimated projected total addressable market per sector for the TruCol consultancy company.

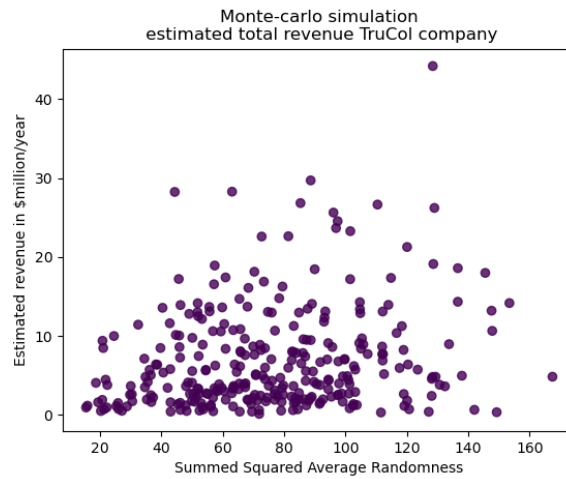


Figure 2: A scatterplot generated by a Monte-Carlo simulation to provide an estimate on the projected total addressable market for the TruCol consultancy company.

## 6 Sensitivity Analysis

### 6.1 Top Down

## 7 Discussion

## 8 Conclusion

## A Appendix \_\_main\_\_.py

---

```

1 import os
2 from .Main import Main
3 from .Model_top_down import Model_top_down
4
5 print(f"Hi, I'll be running the main code, and I'll let you know when
    ↪ I'm done.")
6 project_nr = 1
7 main = Main()
8
9 # run monte-carlo for revenue estimation
10 model = Model_top_down(project_nr)

```

```
11
12
13 # export the code to latex
14 main.export_code_to_latex(project_nr)
15
16 # compile the latex report
17 main.compile_latex_report(project_nr)
18
19 print(f"Done.")
```

---

## B Appendix Main.py

---

```
1 # Example code that creates plots directly in report
2 # Code is an implementation of a genetic algorithm
3 import random
4 from matplotlib import pyplot as plt
5 from matplotlib import lines
6 import matplotlib.pyplot as plt
7 import numpy as np
8
9 from .Compile_latex import Compile_latex
10 from .Plot_to_tex import Plot_to_tex as plt_tex
11 from .Export_code_to_latex import export_code_to_latex
12
13 # define global variables for genetic algorithm example
14 string_length = 100
15 mutation_chance = 1.0 / string_length
16 max_iterations = 1500
17
18
19 class Main:
20     def __init__(self):
21         pass
22
23     def export_code_to_latex(self, project_nr):
24         export_code_to_latex("main.tex", project_nr)
25
26     def compile_latex_report(self, project_nr):
27         """compiles latex code to pdf"""
28         compile_latex = Compile_latex(project_nr, "main.tex")
29
30     def addTwo(self, x):
31         """adds two to the incoming integer and returns the result of
32             ↪ the computation."""
33         return x + 2
34
35 if __name__ == "__main__":
36     # initialize main class
37     main = Main()
```

---

## C Appendix Compile\_latex.py

---

```
1 # runs a jupyter notebook and converts it to pdf
2
3 import os
4 import shutil
5 import nbformat
6 from nbconvert.preprocessors import ExecutePreprocessor
7
8
9 class Compile_latex:
10     def __init__(self, project_nr, latex_filename):
11         self.script_dir = self.get_script_dir()
12         relative_dir = f"latex/project{project_nr}/"
13         self.compile_latex(relative_dir, latex_filename)
14         self.clean_up_after_compilation(latex_filename)
15         self.move_pdf_into_latex_dir(relative_dir, latex_filename)
16
17     # runs jupyter notebook
18     def compile_latex(self, relative_dir, latex_filename):
19         os.system(f"pdflatex {relative_dir}{latex_filename}")
20
21     def clean_up_after_compilation(self, latex_filename):
22         latex_filename_without_extention = latex_filename[:-4]
23         print(f"latex_filename_without_extention={
24             ↪ latex_filename_without_extention}")
25         self.delete_file_if_exists(f"{
26             ↪ latex_filename_without_extention}.aux")
27         self.delete_file_if_exists(f"{
28             ↪ latex_filename_without_extention}.log")
29         self.delete_file_if_exists(f"texput.log")
30
31     def move_pdf_into_latex_dir(self, relative_dir, latex_filename):
32         pdf_filename = f"{latex_filename[:-4]}.pdf"
33         destination = f"{self.get_script_dir()}/../../../{
34             ↪ relative_dir}{pdf_filename}"
35
36         try:
37             shutil.move(pdf_filename, destination)
38         except:
39             print("Error while moving file ", pdf_filename)
40
41     def delete_file_if_exists(self, filename):
42         try:
43             os.remove(filename)
44         except:
45             print(
46                 f"Error while deleting file: {filename} but that is
47                 ↪ not too bad because the intention is for it to
48                 ↪ not be there."
49             )
50
51     def get_script_dir(self):
52         """returns the directory of this script regardless of from
53             ↪ which level the code is executed"""
54         return os.path.dirname(__file__)
55
56 if __name__ == "__main__":
57     main = Compile_latex()
```

---



## D Appendix Datapoints.py

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 from matplotlib import pyplot as plt
4 from matplotlib import lines
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 from .Plot_to_tex import Plot_to_tex as plt_tex
9
10
11 class Datapoints:
12     def __init__(self):
13         """ Initialise the datapoints and compute basic datapoints
14             that can be derived from the datapoints and/or assumptions.
15             ↪ """
16         # Source: https://www.dpdhl.com/content/dam/dpdhl/en/media-
17             ↪ center/investors/documents/annual-reports/DPDHL-2019-
18             ↪ Annual-Report.pdf
19         self.profit_dhl = 4.1 * 10 ** 9 # dollar
20         # Source:
21         self.profit_fedex = 1.29 * 10 ** 9 # dollar
22         # Source: https://investors.ups.com/_assets/
23             ↪ _67e21ed5c7d1164af5b2ef48cec32803/ups/db/1110/9465/
24             ↪ annual_report/
25             ↪ UPS_2021_Proxy_Statement_and_2020_Annual_Report%3
26             ↪ B_Form_10-K.pdf
27         self.profit_ups = 7.7 * 10 ** 9 # dollar
28         # Sum the profit of these three companies
29         self.profit_dhl_fedex_ups = (
30             self.profit_dhl + self.profit_fedex + self.profit_ups
31         )
32
33         # sam and tam factors
34         self.sam_factor=0.03
35         self.tam_factor=0.04
36         print(f'self.profit_dhl_fedex_ups={self.profit_dhl_fedex_ups}
37             ↪ ')
38
39         # Source: https://www.cips.org/supply-management/news/2016/
40             ↪ november/logistics-industry-forecast-to-be-worth-155tn-
41             ↪ by-2023/
42         # NOTE: this article seems an unreliable source and is
43             ↪ outdated, hence the 0.15 should possibly be changed/
44             ↪ updated.
45         self.logistics_market_share_dhl_fedex_ups = 0.15
46
47         # Compute the remaining market share.
48         self.logistics_market_share_remaining = (
49             1 - self.logistics_market_share_dhl_fedex_ups
50         )
51
52         # Assume avg market profit per dollar market share is uniform
53             ↪ .
54         self.logistics_market_profit = self.
55             ↪ get_logistics_market_profit()
56         #print(f'logistics_market_profit={logistics_market_profit}')
57
58         # Conservative estimate based on 0.16 demonstrated by
59             ↪ McKinsey & Company study.
```

```

46 # Source: https://www.mckinsey.com/business-functions/
    ↪ mckinsey-analytics/how-we-help-clients/algorithmic-
    ↪ route-optimization-improves-revenue-for-a-logistics-
    ↪ company#
47 self.profit_gain_by_trucol_protocol = 0.04
48
49 # Estimate based on analogy where a good constraint modeller
50 # can reach significant gains in algorithmic efficiency of
    ↪ solution.
51 self.profit_gain_by_trucol_protocol_consultancy = 0.002
52 # Conservative estimate based on a max of 1.00, for companies
53 # that intend to improve algorithmic efficiency without
    ↪ increasing profit.
54 self.fraction_of_profit_shared_with_trucol = 0.01
55
56 # Source: Statistica.com or marketsandmarkets.com
57 # TODO: re-find exact link
58 self.logistics_market_size = 5.5 * 10 ** 12
59 # Source: Statistica.com or marketsandmarkets.com
60 # TODO: re-find exact link
61 self.algo_trading_market_size = 11.1 * 10 ** 9
62 # Source: Statistica.com or marketsandmarkets.com
63 # TODO: re-find exact link
64 self.material_sciences_market_size = 1 * 10 ** 9
65 # Source: Statistica.com or marketsandmarkets.com
66 # TODO: re-find exact link
67 self.pharmaceutics_market_size = 1.27 * 10 ** 12
68 # Source: Statistica.com or marketsandmarkets.com
69 # TODO: re-find exact link
70 self.telecommunications_market_size = 1.7 * 10 ** 12
71
72 logistics_profit_fraction=self.logistics_market_profit/ self.
    ↪ logistics_market_size
73 print(f'self.profit_dhl_fedex_ups={self.profit_dhl_fedex_ups}
    ↪ ')
74 print(f'self.logistics_market_size={self.
    ↪ logistics_market_size}')
75 print(f'logistics_profit_fraction={logistics_profit_fraction}
    ↪ ')
76
77 # Compute and assume profit margins
78 self.compute_profit_margins() # for logistics sector the
    ↪ data is known
79 self.assume_profit_margins() # for the other sectors the
    ↪ data is assumed
80
81 # Compute profit per market
82 self.compute_market_profit()
83
84 def get_logistics_market_profit(self):
85     """ The basic computation that is done here is a
86     cross multiplication of (see pdf):
87     0.15/0.85=profit-three-companies/profit-remainder."""
88     net_profit_remainder = (
89         self.profit_dhl_fedex_ups
90         * self.logistics_market_share_remaining
91         / self.logistics_market_share_dhl_fedex_ups
92     )
93     net_profit_logistics_market = net_profit_remainder + self.
    ↪ profit_dhl_fedex_ups
94     return net_profit_logistics_market
95

```

```

96 def get_market_profit(self, market_size, profit_margin):
97     return market_size * profit_margin
98
99 def get_market_profit_margin(self, market_size, net_profit):
100     return net_profit / market_size
101
102 def compute_profit_margins(self):
103     # Compute the profit margin based on market size and profit.
104     self.logistics_market_profit_margin = self.
105         ↪ get_market_profit_margin(
106             self.logistics_market_size, self.logistics_market_profit
107         )
108
109 def assume_profit_margins(self):
110
111     # Assume the profit margin in the algorithmic trading market
112     ↪ equals that of the logistics market
113     self.algo_trading_market_profit_margin = self.
114     ↪ logistics_market_profit_margin
115
116     # Assume the profit margin in the material sciences market
117     ↪ equals that of the logistics market
118     self.material_sciences_market_profit_margin = (
119         self.logistics_market_profit_margin
120     )
121
122     # Assume the profit margin in the pharmaceuticals market equals
123     ↪ that of the logistics market
124     self.pharmaceuticals_market_profit_margin = self.
125     ↪ logistics_market_profit_margin
126
127     # Assume the profit margin in the telecommunications market
128     ↪ equals that of the logistics market
129     self.telecommunications_market_profit_margin = (
130         self.logistics_market_profit_margin
131     )
132
133 def compute_market_profit(self):
134     """ Computes the profit per market sector based on
135     the assumption that the profit margin in each sector
136     equals that of the logistics market. """
137     self.algo_trading_market_profit = self.get_market_profit(
138         self.algo_trading_market_size, self.
139         ↪ algo_trading_market_profit_margin
140     )
141     self.material_sciences_market_profit = self.get_market_profit
142     ↪ (
143         self.material_sciences_market_size,
144         self.material_sciences_market_profit_margin,
145     )
146     self.pharmaceuticals_market_profit = self.get_market_profit(
147         self.pharmaceuticals_market_size, self.
148         ↪ pharmaceuticals_market_profit_margin
149     )
150     self.telecommunications_market_profit = self.
151     ↪ get_market_profit(
152         self.telecommunications_market_size,
153         self.telecommunications_market_profit_margin,
154     )

```

---

## E Appendix Export\_code\_to\_latex.py

```
1 # runs a jupyter notebook and converts it to pdf
2 import os
3 import shutil
4 import nbformat
5 from nbconvert.preprocessors import ExecutePreprocessor
6
7
8 def export_code_to_latex(main_latex_filename, project_nr):
9     """This function exports the python files and compiled pdfs of
10         ↪ jupyter notebooks into the
11         latex of the same project number. First it scans which appendices
12         ↪ (without code, without
13         notebooks) are already manually included in the main latex code.
14         ↪ Next, all appendices
15         that contain the python code are either found or created in the
16         ↪ following order:
17         First, the __main__.py file is included, followed by the main.py
18         ↪ file, followed by all
19         python code files in alphabetic order. After this, all the pdfs
20         ↪ of the compiled notebooks
21         are added in alphabetic order of filename. This order of
22         ↪ appendices is overwritten in the
23         main tex file.
24
25     :param main_latex_filename: Name of the main latex document of
26         ↪ this project number
27     :param project_nr: The number indicating which project this code
28         ↪ pertains to.
29     """
30     script_dir = get_script_dir()
31     relative_dir = f"latex/project{project_nr}/"
32     appendix_dir = script_dir + "../..../" + relative_dir + "
33         ↪ Appendices/"
34     path_to_main_latex_file = (
35         f"{script_dir}/../..../{relative_dir}/{main_latex_filename}"
36     )
37     root_dir = script_dir[0 : script_dir.rfind(f"code/project{
38         ↪ project_nr}")]]
39
40     # Get paths to files containing python code.
41     python_filepaths = get_filenames_in_dir("py", script_dir, ["
42         ↪ __init__.py"])
43     compiled_notebook_pdf_filepaths = get_compiled_notebook_paths(
44         ↪ script_dir)
45
46     # Check which files are already included in the latex appendices
47     ↪ .
48     python_files_already_included_in_appendices =
49         ↪ get_code_files_already_included_in_appendices(
50             python_filepaths, appendix_dir, ".py", project_nr, root_dir
51         )
52     notebook_pdf_files_already_included_in_appendices =
53         ↪ get_code_files_already_included_in_appendices(
54             compiled_notebook_pdf_filepaths, appendix_dir, ".ipynb",
55             ↪ project_nr, root_dir,
56     )
57
58     # Get which appendices are still missing.
59     missing_python_files_in_appendices =
60         ↪ get_code_files_not_yet_included_in_appendices(
```

```

43     python_filepaths, python_files_already_included_in_appendices
44         ↪ , ".py"
45 )
46 missing_notebook_files_in_appendices =
47     ↪ get_code_files_not_yet_included_in_appendices(
48         compiled_notebook_pdf_filepaths,
49         notebook_pdf_files_already_included_in_appendices,
50         ".pdf",
51     )
52 # Create the missing appendices.
53 created_python_appendix_filenames = create_appendices_with_code(
54     ↪ appendix_dir, missing_python_files_in_appendices, ".py",
55     ↪ project_nr, root_dir
56 )
57 created_notebook_appendix_filenames = create_appendices_with_code
58     ↪ (
59     ↪ appendix_dir,
60     ↪ missing_notebook_files_in_appendices,
61     ↪ ".ipynb",
62     ↪ project_nr,
63     ↪ root_dir,
64     )
65 appendices = get_list_of_appendix_files(
66     ↪ appendix_dir, compiled_notebook_pdf_filepaths,
67     ↪ python_filepaths
68 )
69 main_tex_code, start_index, end_index, appendix_tex_code =
70     ↪ get_appendix_tex_code(
71     ↪ path_to_main_latex_file
72 )
73 # assumes non-included non-code appendices should not be included
74     ↪ :
75 # overwrite the existing appendix lists with the current appendix
76     ↪ list.
77 (
78     ↪ non_code_appendices,
79     ↪ main_non_code_appendix_inclusion_lines,
80 ) = get_order_of_non_code_appendices_in_main(appendices,
81     ↪ appendix_tex_code)
82 python_appendix_filenames = list(
83     ↪ map(
84     ↪ ↪ lambda x: x.appendix_filename,
85     ↪ ↪ filter_appendices_by_type(appendices, "python"),
86     )
87 )
88 sorted_created_python_appendices = sort_python_appendices(
89     ↪ filter_appendices_by_type(appendices, "python")
90 )
91 sorted_python_appendix_filenames = list(
92     ↪ map(lambda x: x.appendix_filename,
93     ↪ ↪ sorted_created_python_appendices)
94 )
95 notebook_appendix_filenames = list(
96     ↪ map(
97     ↪ ↪ lambda x: x.appendix_filename,

```

```

95         filter_appendices_by_type(appendices, "notebook"),
96     )
97 )
98 sorted_created_notebook_appendices =
99     ↪ sort_notebook_appendices_alphabetically(
100         filter_appendices_by_type(appendices, "notebook")
101     )
102 sorted_notebook_appendix_filenames = list(
103     ↪ map(lambda x: x.appendix_filename,
104         sorted_created_notebook_appendices)
105 )
106
107 appendix_latex_code = create_appendices_latex_code(
108     main_non_code_appendix_inclusion_lines,
109     sorted_created_notebook_appendices,
110     project_nr,
111     sorted_created_python_appendices,
112 )
113
114 updated_main_tex_code = substitute_appendix_code(
115     end_index, main_tex_code, start_index, appendix_latex_code
116 )
117
118 overwrite_content_to_file(updated_main_tex_code,
119     ↪ path_to_main_latex_file)
120
121 def create_appendices_latex_code(
122     main_non_code_appendix_inclusion_lines,
123     notebook_appendices,
124     project_nr,
125     python_appendices,
126 ):
127     """Creates the latex code that includeds the appendices in the
128     ↪ main latex file.
129
130     :param main_non_code_appendix_inclusion_lines: latex code that
131     ↪ includes the appendices that do not contain python code nor
132     ↪ notebooks
133     :param notebook_appendices: List of Appendix objects representing
134     ↪ appendices that include the pdf files of compiled Jupiter
135     ↪ notebooks
136     :param project_nr: The number indicating which project this code
137     ↪ pertains to.
138     :param python_appendices: List of Appendix objects representing
139     ↪ appendices that include the python code files.
140     """
141     main_appendix_inclusion_lines =
142     ↪ main_non_code_appendix_inclusion_lines
143
144     appendices_of_all_types = [python_appendices, notebook_appendices
145     ↪ ]
146     main_appendix_inclusion_lines.append(
147         f"\IfFileExists{{latex/project{project_nr}/main.tex}}{{{"
148     )
149     main_appendix_inclusion_lines = append_latex_inclusion_command(
150         appendices_of_all_types, True, main_appendix_inclusion_lines,
151         ↪ project_nr,
152     )
153     main_appendix_inclusion_lines.append(f"}}{{{"")
154     main_appendix_inclusion_lines = append_latex_inclusion_command(

```

```

143         appendices_of_all_types, False, main_appendix_inclusion_lines
144         ↪ , project_nr,
145     )
146     return main_appendix_inclusion_lines
147
148 def append_latex_inclusion_command(
149     appendices_of_all_types, is_from_root_dir,
150     ↪ main_appendix_inclusion_lines, project_nr
151 ):
152     for appendix_type in appendices_of_all_types:
153         for appendix in appendix_type:
154             line = update_appendix_tex_code(
155                 appendix.appendix_filename, is_from_root_dir,
156                 ↪ project_nr
157             )
158             print(f"appendix.appendix_filename={appendix.
159                 ↪ appendix_filename}")
160             main_appendix_inclusion_lines.append(line)
161     return main_appendix_inclusion_lines
162
163 def filter_appendices_by_type(appendices, appendix_type):
164     """Returns the list of all appendices of a certain appendix type,
165     ↪ from the incoming list of Appendix objects.
166
167     :param appendices: List of Appendix objects
168     :param appendix_type: Can consist of "no_code", "python", or "
169     ↪ notebook" and indicates different appendix types
170     """
171     return_appendices = []
172     for appendix in appendices:
173         if appendix.appendix_type == appendix_type:
174             return_appendices.append(appendix)
175     return return_appendices
176
177 def sort_python_appendices(appendices):
178     """First puts __main__.py, followed by main.py followed by a-z
179     ↪ code files.
180
181     :param appendices: List of Appendix objects
182     """
183     return_appendices = []
184     for appendix in appendices: # first get appendix containing
185         ↪ __main__.py
186         if (appendix.code_filename == "__main__.py") or (
187             appendix.code_filename == "__Main__.py"
188         ):
189             return_appendices.append(appendix)
190             appendices.remove(appendix)
191     for appendix in appendices: # second get appendix containing
192         ↪ main.py
193         if (appendix.code_filename == "main.py") or (
194             appendix.code_filename == "Main.py"
195         ):
196             return_appendices.append(appendix)
197             appendices.remove(appendix)
198     return return_appendices
199
200 # Filter remaining appendices in order of a-z
201 filtered_remaining_appendices = [

```



```

196         i for i in appendices if i.code_filename is not None
197     ]
198     appendices_sorted_a_z = sort_appendices_on_code_filename(
199         filtered_remaining_appendices
200     )
201     return return_appendices + appendices_sorted_a_z
202
203
204 def sort_notebook_appendices_alphabetically(appendices):
205     """Sorts notebook appendix objects alphabetic order of their pdf
206         ↪ filenames.
207
208     :param appendices: List of Appendix objects
209     """
210     return_appendices = []
211     filtered_remaining_appendices = [
212         i for i in appendices if i.code_filename is not None
213     ]
214     appendices_sorted_a_z = sort_appendices_on_code_filename(
215         filtered_remaining_appendices
216     )
217     return return_appendices + appendices_sorted_a_z
218
219 def sort_appendices_on_code_filename(appendices):
220     """Returns a list of Appendix objects that are sorted and based
221         ↪ on the property: code_filename.
222     Assumes the incoming appendices only contain python files.
223
224     :param appendices: List of Appendix objects
225     """
226     attributes = list(map(lambda x: x.code_filename, appendices))
227     sorted_indices = sorted(range(len(attributes)), key=lambda k:
228         ↪ attributes[k])
229     sorted_list = []
230     for i in sorted_indices:
231         sorted_list.append(appendices[i])
232     return sorted_list
233
234 def get_order_of_non_code_appendices_in_main(appendices,
235     ↪ appendix_tex_code):
236     """Scans the lines of appendices in the main code, and returns
237         ↪ the lines
238     of the appendices that do not contain code, in the order in which
239         ↪ they were
240     included in the main latex file.
241
242     :param appendices: List of Appendix objects
243     :param appendix_tex_code: latex code from the main latex file
244         ↪ that includes the appendices
245     """
246     non_code_appendices = []
247     non_code_appendix_lines = []
248     appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
249     for line in appendix_tex_code:
250         appendix_filename = get_filename_from_latex_appendix_line(
251             ↪ appendices, line)
252
253         # Check if line is not commented
254         if not appendix_filename is None:
255             if not line_is_commented(line, appendix_filename):

```



```

250         appendix = get_appendix_from_filename(appendices,
251         ↪ appendix_filename)
252         if appendix.appendix_type == "no_code":
253             non_code_appendices.append(appendix)
254             non_code_appendix_lines.append(line)
255     return non_code_appendices, non_code_appendix_lines
256
257 def get_filename_from_latex_appendix_line(appendices, appendix_line):
258     """Returns the first filename from a list of incoming filenames
259     ↪ that
260     occurs in a latex code line.
261
262     :param appendices: List of Appendix objects
263     :param appendix_line: latex code (in particular expected to be
264     ↪ the code from main that is used to include appendix latex
265     ↪ files.)
266     """
267     for filename in list(map(lambda appendix: appendix.
268     ↪ appendix_filename, appendices)):
269         if filename in appendix_line:
270             if not line_is_commented(appendix_line, filename):
271                 return filename
272
273 def get_appendix_from_filename(appendices, appendix_filename):
274     """Returns the first Appendix object with an appendix filename
275     ↪ that matches the incoming appendix_filename.
276     The Appendix objects are selected from an incoming list of
277     ↪ Appendix objects.
278
279     :param appendices: List of Appendix objects
280     :param appendix_filename: name of a latex appendix file, ends in
281     ↪ .tex,
282     """
283     for appendix in appendices:
284         if appendix_filename == appendix.appendix_filename:
285             return appendix
286
287 def get_compiled_notebook_paths(script_dir):
288     """Returns the list of jupyter notebook filepaths that were
289     ↪ compiled successfully and that are
290     included in the same dias this script (the src directory).
291
292     :param script_dir: absolute path of this file.
293     """
294     notebook_filepaths = get_filenames_in_dir(".ipynb", script_dir)
295     compiled_notebook_filepaths = []
296
297     # check if the jupyter notebooks were compiled
298     for notebook_filepath in notebook_filepaths:
299         # swap file extension
300         notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf"
301         ↪ ")
302
303         # check if file exists
304         if os.path.isfile(notebook_filepath):
305             compiled_notebook_filepaths.append(notebook_filepath)
306     return compiled_notebook_filepaths

```

```

302
303 def get_list_of_appendix_files(
304     appendix_dir, absolute_notebook_filepaths,
305     ↪ absolute_python_filepaths
306 ):
307     """Returns a list of Appendix objects that contain all the
308     ↪ appendix files with .tex extension.
309
310     :param appendix_dir: Absolute path that contains the appendix .
311     ↪ tex files.
312     :param absolute_notebook_filepaths: List of absolute paths to the
313     ↪ compiled notebook pdf files.
314     :param absolute_python_filepaths: List of absolute paths to the
315     ↪ python files.
316     """
317     appendices = []
318     appendices_paths = get_filenames_in_dir(".tex", appendix_dir)
319
320     for appendix_filepath in appendices_paths:
321         appendix_type = "no_code"
322         appendix_filecontent = read_file(appending_filepath)
323         line_nr_python_file_inclusion = get_line_of_latex_command(
324             appendix_filecontent, "\pythonexternal{"
325         )
326         line_nr_notebook_file_inclusion = get_line_of_latex_command(
327             appendix_filecontent, "\includepdf[pages="
328         )
329         if line_nr_python_file_inclusion > -1:
330             appendix_type = "python"
331             # get python filename
332             line = appendix_filecontent[line_nr_python_file_inclusion
333             ↪ ]
334             filename = get_filename_from_latex_inclusion_command(
335                 line, ".py", "\pythonexternal{"
336             )
337             appendices.append(
338                 Appendix(
339                     appendix_filepath,
340                     appendix_filecontent,
341                     appendix_type,
342                     filename,
343                     line,
344                 )
345             )
346         if line_nr_notebook_file_inclusion > -1:
347             appendix_type = "notebook"
348             line = appendix_filecontent[
349                 ↪ line_nr_notebook_file_inclusion]
350             filename = get_filename_from_latex_inclusion_command(
351                 line, ".pdf", "\includepdf[pages="
352             )
353             appendices.append(
354                 Appendix(
355                     appendix_filepath,
356                     appendix_filecontent,
357                     appendix_type,
358                     filename,
359                     line,
360                 )
361             )
362         else:
363             appendices.append(

```

```

357         Appendix(appendix_filepath, appendix_filecontent,
358                   ↪ appendix_type)
359     return appendices
360
361 def get_filename_from_latex_inclusion_command(
362     appendix_line, extension, start_substring
363 ):
364     """returns the code/notebook filename in a latex command which
365     ↪ includes that code in an appendix.
366     The inclusion command includes a python code or jupyter notebook
367     ↪ pdf.
368
369     :param appendix_line: :Line of latex code (in particular expected
370     ↪ to be the latex code from an appendix.).
371     :param extension: The file extension of the file that is sought
372     ↪ in the appendix line. Either ".py" or ".pdf".
373     :param start_substring: The substring that characterises the
374     ↪ latex inclusion command.
375     """
376     start_index = appendix_line.index(start_substring)
377     end_index = appendix_line.index(extension)
378     return get_filename_from_dir(
379         appendix_line[start_index : end_index + len(extension)]
380     )
381
382 def get_filenames_in_dir(extension, path, excluded_files=None):
383     """Returns a list of the relative paths to all files within the
384     ↪ some path that match
385     the given file extension.
386
387     :param extension: The file extension of the file that is sought
388     ↪ in the appendix line. Either ".py" or ".pdf".
389     :param path: Absolute filepath in which files are being sought.
390     :param excluded_files: (Default value = None) Files that will not
391     ↪ be included even if they are found.
392     """
393     filepaths = []
394     for r, d, f in os.walk(path):
395         for file in f:
396             if file.endswith(extension):
397                 if (excluded_files is None) or (
398                     (not excluded_files is None) and (not file in
399                     ↪ excluded_files)
400                 ):
401                     filepaths.append(r + "/" + file)
402     return filepaths
403
404 def get_code_files_already_included_in_appendices(
405     absolute_code_filepaths, appendix_dir, extension, project_nr,
406     ↪ root_dir
407 ):
408     """Returns a list of code filepaths that are already properly
409     ↪ included the latex appendix files of this project.
410
411     :param absolute_code_filepaths: List of absolute paths to the
412     ↪ code files (either python files or compiled jupyter
413     ↪ notebook pdfs).

```

```

404 :param appendix_dir: Absolute path that contains the appendix .
    ↪ tex files.
405 :param extension: The file extension of the file that is sought
    ↪ in the appendix line. Either ".py" or ".pdf".
406 :param project_nr: The number indicating which project this code
    ↪ pertains to.
407 :param root_dir: The root directory of this repository.
408 """
409 appendix_files = get_filenames_in_dir(".tex", appendix_dir)
410 contained_codes = []
411 for code_filepath in absolute_code_filepaths:
412     for appendix_filepath in appendix_files:
413         appendix_filecontent = read_file(appendix_filepath)
414         line_nr = check_if_appendix_contains_file(
415             appendix_filecontent, code_filepath, extension,
    ↪ project_nr, root_dir
416         )
417         if line_nr > -1:
418             # add filepath to list of files that are already in
    ↪ the appendices
419             contained_codes.append(
420                 Appendix_with_code(
421                     code_filepath,
422                     appendix_filepath,
423                     appendix_filecontent,
424                     line_nr,
425                     ".py",
426                 )
427             )
428 return contained_codes
429
430 def check_if_appendix_contains_file(
431     appendix_content, code_filepath, extension, project_nr, root_dir
432 ):
433     """Scans an appendix content to determine whether it contains a
    ↪ substring that
434     includes a code file (of either python or compiled notebook=pdf
    ↪ extension).
435
436     :param appendix_content: content in an appendix latex file.
437     :param code_filepath: Absolute path to a code file (either python
    ↪ files or compiled jupyter notebook pdfs).
438     :param extension: The file extension of the file that is sought
    ↪ in the appendix line. Either ".py" or ".pdf".
439     :param project_nr: The number indicating which project this code
    ↪ pertains to.
440     :param root_dir: The root directory of this repository.
441     """
442     # convert code_filepath to the inclusion format in latex format
443     latex_relative_filepath = (
444         f"latex/project{project_nr}/{../../{code_filepath[len(root_dir
    ↪ ):]}"
445     )
446     latex_command = get_latex_inclusion_command(extension,
    ↪ latex_relative_filepath)
447     return get_line_of_latex_command(appendix_content, latex_command)
448
449 def get_line_of_latex_command(appendix_content, latex_command):
450     """Returns the line number of a latex command if it is found.
    ↪ Returns -1 otherwise.

```

```

453 :param appendix_content: content in an appendix latex file.
454 :param latex_command: A line of latex code. (Expected to come
455 ↪ from some appendix)
456 """
457 # check if the file is in the latex code
458 line_nr = 0
459 for line in appendix_content:
460     if latex_command in line:
461         if line_is_commented(line, latex_command):
462             commented = True
463         else:
464             return line_nr
465     line_nr = line_nr + 1
466 return -1
467
468 def line_is_commented(line, target_substring):
469     """Returns True if a latex code line is commented, returns False
470     ↪ otherwise
471
472     :param line: A line of latex code that contains a relevant
473     ↪ command (target substring).
474     :param target_substring: Used to determine whether the command
475     ↪ that is found is commented or not.
476     """
477     left_of_command = line[: line.rfind(target_substring)]
478     if "%" in left_of_command:
479         return True
480     return False
481
482 def get_latex_inclusion_command(extension,
483 ↪ latex_relative_filepath_to_codefile):
484     """Creates and returns a latex command that includes either a
485     ↪ python file or a compiled jupyter
486     notebook pdf (wherever the command is placed). The command is
487     ↪ intended to be placed in the appendix.
488
489     :param extension: The file extension of the file that is sought
490     ↪ in the appendix line. Either ".py" or ".pdf".
491     :param latex_relative_filepath_to_codefile: The latex compilation
492     ↪ requires a relative path towards code files
493     that are included. Therefore, a relative path towards the code is
494     ↪ given.
495     """
496     if extension == ".py":
497         left = "\pythonexternal{"
498         right = "}"
499         latex_command = f"{left}{latex_relative_filepath_to_codefile
500     ↪ }{right}"
501     elif extension == ".ipynb":
502         left = "\includepdf[pages=-]{ "
503         right = "}"
504         latex_command = f"{left}{latex_relative_filepath_to_codefile
505     ↪ }{right}"
506     return latex_command
507
508 def read_file(filepath):

```

```

502 """Reads content of a file and returns it as a list of strings,
    ↪ with one string per line.
503
504 :param filepath: path towards the file that is being read.
505 """
506 with open(filepath) as f:
507     content = f.readlines()
508 return content
509
510 def get_code_files_not_yet_included_in_appendices(
511     code_filepaths, contained_codes, extension
512 ):
513     """Returns a list of filepaths that are not yet properly included
    ↪ in some appendix of this project.
514
515     :param code_filepath: Absolute path to all the code files in
    ↪ this project (source directory).
    (either python files or compiled jupyter notebook pdfs).
516     :param contained_codes: list of Appendix objects that include
    ↪ either python files or compiled jupyter notebook pdfs,
    ↪ which
517     are already included in the appendix tex files. (Does not care
    ↪ whether those appendices are also actually
518     included in the main or not.)
519     :param extension: The file extension of the file that is sought
    ↪ in the appendix line. Either ".py" or ".pdf".
520 """
521 contained_filepaths = list(
522     map(lambda contained_file: contained_file.code_filepath,
523         ↪ contained_codes)
524 )
525 not_contained = []
526 for filepath in code_filepaths:
527     if not filepath in contained_filepaths:
528         not_contained.append(filepath)
529 return not_contained
530
531
532 def create_appendices_with_code(
533     appendix_dir, code_filepaths, extension, project_nr, root_dir
534 ):
535     """Creates the latex appendix files in with relevant codes
    ↪ included.
536
537     :param appendix_dir: Absolute path that contains the appendix .
    ↪ tex files.
538     :param code_filepaths: Absolute path to code files that are not
    ↪ yet included in an appendix
    (either python files or compiled jupyter notebook pdfs).
539     :param extension: The file extension of the file that is sought
    ↪ in the appendix line. Either ".py" or ".pdf".
540     :param project_nr: The number indicating which project this code
    ↪ pertains to.
541     :param root_dir: The root directory of this repository.
542 """
543 appendix_filenames = []
544 appendix_reference_index = (
545     get_index_of_auto_generated_appendices(appendix_dir,
546         ↪ extension) + 1
547 )
548
549

```

```

550 for code_filepath in code_filepaths:
551     latex_relative_filepath = (
552         f"latex/project{project_nr}/../../{code_filepath[len(
553             ↪ root_dir):]}"
554     )
555     code_path_from_latex_main_path = f"../../{code_filepath[len(
556         ↪ root_dir):]}"
557     content = []
558     filename = get_filename_from_dir(code_filepath)
559
560     content = create_section(appendix_reference_index, filename,
561         ↪ content)
562     content = add_include_code_in_appendix(
563         content,
564         code_filepath,
565         code_path_from_latex_main_path,
566         extension,
567         latex_relative_filepath,
568         project_nr,
569         root_dir,
570     )
571
572     overwrite_content_to_file(
573         content,
574         f"{appendix_dir}Auto_generated_{extension[1:]}_App{
575             ↪ appendix_reference_index}.tex",
576         False,
577     )
578     appendix_filenames.append(
579         f"Auto_generated_{extension[1:]}_App{
580             ↪ appendix_reference_index}.tex"
581     )
582     appendix_reference_index = appendix_reference_index + 1
583 return appendix_filenames
584
585 def add_include_code_in_appendix(
586     content,
587     code_filepath,
588     code_path_from_latex_main_path,
589     extension,
590     latex_relative_filepath,
591     project_nr,
592     root_dir,
593 ):
594     """Includes the latex code that includes code in the script.
595
596     :param content: The latex content that is being written to an
597         ↪ appendix.
598     :param code_path_from_latex_main_path: the path to the code as
599         ↪ seen from the folder that contains main.tex.
600     :param extension: The file extension of the file that is sought
601         ↪ in the appendix line. Either ".py" or ".pdf".
602     :param latex_relative_filepath_to_codefile: The latex compilation
603         ↪ requires a relative path towards code files
604     that are included. Therefore, a relative path towards the code is
605         ↪ given.
606     """
607     content.append(
608         f"\IfFileExists{{latex/project{project_nr}/../../{
609             ↪ code_filepath[len(root_dir):]}}}{{"
610     )

```

```

601 # append current line
602 content.append(get_latex_inclusion_command(extension,
        ↳ latex_relative_filepath))
603 content.append(f"{}{}")
604 content.append(
605     get_latex_inclusion_command(extension,
        ↳ code_path_from_latex_main_path)
606 )
607 content.append(f"{}")
608 return content
609
610
611 def get_index_of_auto_generated_appendices(appendix_dir, extension):
612     """Returns the maximum index of auto generated appendices of
613     a specific extension type.
614
615     :param extension: The file extension of the file that is sought
616         ↳ in the appendix line. Either ".py" or ".pdf".
617     :param appendix_dir: Absolute path that contains the appendix .
618         ↳ tex files.
619     """
620     max_index = -1
621     appendices =
622         ↳ get_auto_generated_appendix_filenames_of_specific_extension
623         ↳ (
624             appendix_dir, extension
625         )
626     for appendix in appendices:
627         substring = f"Auto_generated_{extension[1:]}_App"
628         # remove left of index
629         remainder = appendix.rfind(substring) + len(
630             ↳ substring) :]
631         # remove right of index
632         index = int(remainder[:-4])
633         if index > max_index:
634             max_index = index
635     return max_index
636
637
638 def get_auto_generated_appendix_filenames_of_specific_extension(
639     appendix_dir, extension
640 ):
641     """Returns the list of auto generated appendices of
642     a specific extension type.
643
644     :param extension: The file extension of the file that is sought
645         ↳ in the appendix line. Either ".py" or ".pdf".
646     :param appendix_dir: Absolute path that contains the appendix .
647         ↳ tex files.
648     """
649     appendices_of_extension_type = []
650
651     # get all appendices
652     appendix_files = get_filenames_in_dir(".tex", appendix_dir)
653
654     # get appendices of particular extension type
655     for appendix_filepath in appendix_files:
656         right_of_slash = appendix_filepath[appendix_filepath.rfind("/")
657             ↳ + 1 :]
658         if (
659             right_of_slash[: 15 + len(extension) - 1]
660             == f"Auto_generated_{extension[1:]}")

```



```

653         ):
654             appendices_of_extension_type.append(appendix_filepath)
655     return appendices_of_extension_type
656
657
658 def create_section(appendix_reference_index, code_filename, content):
659     """Creates the header of a latex appendix file, such that it
660         ↳ contains a section that
661         indicates the section is an appendix, and indicates which python
662         ↳ or notebook file is
663         being included in that appendix.
664
665     :param appendix_reference_index: A counter that is used in the
666         ↳ label to ensure the appendix section labels are unique.
667     :param code_filename: file name of the code file that is included
668     :param content: A list of strings that make up the appendix, with
669         ↳ one line per element.
670     """
671     # write section
672     left = "\section{Appendix "
673     middle = code_filename.replace("-", "\-")
674     right = "}\label{app:"
675     end = "}" # TODO: update appendix reference index
676     content.append(f"{left}{middle}{right}{appendix_reference_index}{{
677         ↳ end}}")
678     return content
679
680
681 def overwrite_content_to_file(content, filepath, content_has_newlines
682     ↳ =True):
683     """Writes a list of lines of tex code from the content argument
684         ↳ to a .tex file
685     using overwriting method. The content has one line per element.
686
687     :param content: The content that is being written to file.
688     :param filepath: Path towards the file that is being read.
689     :param content_has_newlines: (Default value = True)
690     """
691     with open(filepath, "w") as f:
692         for line in content:
693             if content_has_newlines:
694                 f.write(line)
695             else:
696                 f.write(line + "\n")
697
698
699 def get_appendix_tex_code(main_latex_filename):
700     """gets the latex appendix code from the main tex file.
701
702     :param main_latex_filename: Name of the main latex document of
703         ↳ this project number
704     """
705     main_tex_code = read_file(main_latex_filename)
706     start = "\\begin{appendices}"
707     end = "\\end{appendices}"
708     start_index = get_index_of_substring_in_list(main_tex_code, start
709         ↳ ) + 1
710     end_index = get_index_of_substring_in_list(main_tex_code, end)
711     return main_tex_code, start_index, end_index, main_tex_code[
712         ↳ start_index:end_index]

```

```

705 def get_index_of_substring_in_list(lines, target_substring):
706     """Returns the index of the line in which the first character of
707         ↳ a latex substring if it is found
708         ↳ uncommented in the incoming list.
709
710     :param lines: List of lines of latex code.
711     :param target_substring: Some latex command/code that is sought
712         ↳ in the incoming text.
713     """
714     for i in range(0, len(lines)):
715         if target_substring in lines[i]:
716             if not line_is_commented(lines[i], target_substring):
717                 return i
718
719 def update_appendix_tex_code(appendix_filename, is_from_root_dir,
720     ↳ project_nr):
721     """Returns the latex command that includes an appendix .tex file
722         ↳ in an appendix environment
723         ↳ as can be used in the main tex file.
724
725     :param appendix_filename: Name of the appendix that is included
726         ↳ by the generated command.
727     :param project_nr: The number indicating which project this code
728         ↳ pertains to.
729     """
730     if is_from_root_dir:
731         left = f"\input{{latex/project{project_nr}}/"
732     else:
733         left = "\input{"
734         middle = "Appendices/"
735         right = "} \\newpage\n"
736     return f"{left}{middle}{appendix_filename}{right}"
737
738 def substitute_appendix_code(
739     end_index, main_tex_code, start_index,
740     ↳ updated_appendices_tex_code
741 ):
742     """Replaces the old latex code that included the appendices in
743         ↳ the main.tex file with the new latex
744         ↳ commands that include the appendices in the latex report.
745
746     :param end_index: Index at which the appendix section ends right
747         ↳ before the latex \end{appendix} line,
748     :param main_tex_code: The code that is saved in the main .tex
749         ↳ file.
750     :param start_index: Index at which the appendix section starts
751         ↳ right after the latex \begin{appendix} line,
752     :param updated_appendices_tex_code: The newly created code that
753         ↳ includes all the relevant appendices.
754     (relevant being (in order): manually created appendices, python
755         ↳ codes, pdfs of compiled jupyter notebooks).
756     """
757     updated_main_tex_code = (
758         main_tex_code[0:start_index]
759         + updated_appendices_tex_code
760         + main_tex_code[end_index:]
761     )
762     return updated_main_tex_code

```

```

754 def get_filename_from_dir(path):
755     """Returns a filename from an absolute path to a file.
756
757     :param path: path to a file of which the name is queried.
758     """
759     return path[path.rfind("/") + 1 :]
760
761 def get_script_dir():
762     """returns the directory of this script regardless of from which
763     ↪ level the code is executed"""
764     return os.path.dirname(__file__)
765
766 class Appendix_with_code:
767     """stores in which appendix file and accompanying line number in
768     ↪ the appendix in which a code file is
769     already included. Does not take into account whether this
770     ↪ appendix is in the main tex file or not
771     """
772     def __init__(
773         self,
774         code_filepath,
775         appendix_filepath,
776         appendix_content,
777         file_line_nr,
778         extension,
779     ):
780         self.code_filepath = code_filepath
781         self.appendix_filepath = appendix_filepath
782         self.appendix_content = appendix_content
783         self.file_line_nr = file_line_nr
784         self.extension = extension
785
786 class Appendix:
787     """stores in appendix files and type of appendix."""
788
789     def __init__(
790         self,
791         appendix_filepath,
792         appendix_content,
793         appendix_type,
794         code_filename=None,
795         appendix_inclusion_line=None,
796     ):
797
798         self.appendix_filepath = appendix_filepath
799         self.appendix_filename = get_filename_from_dir(self.
800             ↪ appendix_filepath)
801         self.appendix_content = appendix_content
802         self.appendix_type = appendix_type # TODO: perform
803             ↪ validation of input values
804         self.code_filename = code_filename
805         self.appendix_inclusion_line = appendix_inclusion_line

```

---

## F Appendix Model\_bottom\_up.py

---

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 from matplotlib import pyplot as plt
4 from matplotlib import lines
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 from .Plot_to_tex import Plot_to_tex as plt_tex
9
10
11 class Model_bottom_up:
12     def __init__(self):
13         pass
14
15     def addTwo(self, x):
16         """adds two to the incoming integer and returns the result of
17             ↪ the computation."""
18         return x + 2
```

---

## G Appendix Model\_top\_down.py

```
1 # The bottom up model that computes the TAM and TSM
2
3 from matplotlib import lines
4 import matplotlib.pyplot as plt
5 import matplotlib.patches as mpatches
6 import matplotlib.ticker as ticker
7 import numpy as np
8 import os
9 import random
10
11 from .Plot_to_tex import Plot_to_tex as plt_tex
12 from .Datapoints import Datapoints
13
14
15 class Model_top_down:
16     def __init__(self, project_nr):
17         self.project_nr = project_nr
18         self.nr_simulations = 300
19         self.dp = Datapoints()
20         x_series, y_series = self.estimate_revenue()
21         self.plot_data_series(x_series, y_series)
22         # self.get_normal_dist()
23         y = self.sum_revenues(y_series)
24         x = self.avg_randomness(x_series)
25
26         self.plot_data(x, y)
27
28     def sum_revenues(self, x_series):
29         summed_series = []
30         for i in range(0, len(x_series[0])):
31             summed = 0
32             for j in range(0, len(x_series)):
33                 summed = summed + x_series[j][i]
34
35             summed_series.append(summed)
36         return summed_series
37
38     def avg_randomness(self, series):
39         summed_series = []
40         for i in range(0, len(series[0])):
41             summed = 0
42             for j in range(0, len(series)):
43                 summed = summed + series[j][i]
44             summed_series.append(summed / len(series))
45         return summed_series
46
47     def estimate_revenue(self):
48         revenue_logistics, randomness_logistics = self.
49             ↪ estimate_logistics_revenue(
50                 self.nr_simulations,
51                 self.dp.profit_gain_by_trucol_protocol_consultancy,
52                 self.dp.logistics_market_profit,
53                 self.dp.fraction_of_profit_shared_with_trucol,
54                 self.dp.sam_factor,
55                 self.dp.tam_factor
56             )
57         # self.plot_data(randomness_logistics, revenue_logistics)
58
59         # algo
```

```

59 revenue_algo_trading, randomness_algo_trading = self.
    ↪ estimate_logistics_revenue(
60     self.nr_simulations,
61     self.dp.profit_gain_by_trucol_protocol_consultancy,
62     self.dp.algo_trading_market_profit,
63     self.dp.fraction_of_profit_shared_with_trucol,
64     self.dp.sam_factor,
65     self.dp.tam_factor
66 )
67 # self.plot_data(randomness_algo_trading,
    ↪ revenue_algo_trading)
68
69 # material
70 (
71     revenue_material_sciences,
72     randomness_material_sciences,
73 ) = self.estimate_logistics_revenue(
74     self.nr_simulations,
75     self.dp.profit_gain_by_trucol_protocol_consultancy,
76     self.dp.material_sciences_market_profit,
77     self.dp.fraction_of_profit_shared_with_trucol,
78     self.dp.sam_factor,
79     self.dp.tam_factor
80 )
81 print(
82     f"material_sciences_market_profit={self.dp.
    ↪ material_sciences_market_profit}"
83 )
84 # self.plot_data(randomness_material_sciences,
    ↪ material_sciences_market_profit)
85
86 # pharma
87 (
88     revenue_pharmaceutics,
89     randomness_pharmaceutics,
90 ) = self.estimate_logistics_revenue(
91     self.nr_simulations,
92     self.dp.profit_gain_by_trucol_protocol_consultancy,
93     self.dp.pharmaceutics_market_profit,
94     self.dp.fraction_of_profit_shared_with_trucol,
95     self.dp.sam_factor,
96     self.dp.tam_factor
97 )
98 print(f"pharmaceutics_market_profit={self.dp.
    ↪ pharmaceutics_market_profit}")
99 # self.plot_data(randomness_pharmaceutics,
    ↪ revenue_pharmaceutics)
100
101 # tele
102 (
103     revenue_telecommunications,
104     randomness_telecommunications,
105 ) = self.estimate_logistics_revenue(
106     self.nr_simulations,
107     self.dp.profit_gain_by_trucol_protocol_consultancy,
108     self.dp.telecommunications_market_profit,
109     self.dp.fraction_of_profit_shared_with_trucol,
110     self.dp.sam_factor,
111     self.dp.tam_factor
112 )
113 print(

```

```

114         f"telecommunications_market_profit={self.dp.
115             ↪ telecommunications_market_profit}"
116     )
117     # self.plot_data(randomness_telecommunications,
118         ↪ revenue_telecommunications)
119
120     # Concatenate all datapoints
121     x_series = [
122         randomness_logistics,
123         randomness_algo_trading,
124         randomness_material_sciences,
125         randomness_pharmaceutics,
126         randomness_telecommunications,
127     ]
128     y_series = [
129         revenue_logistics,
130         revenue_algo_trading,
131         revenue_material_sciences,
132         revenue_pharmaceutics,
133         revenue_telecommunications,
134     ]
135     return x_series, y_series
136
137     def estimate_logistics_revenue(
138         self, N, gain, market_profit, shared_profit_fraction
139     ):
140         revenue_estimates = []
141         randomness = []
142         for i in range(0, N):
143             # TODO: change to get the range as specified in
144             ↪ datapoints per parameter
145             rand_a = float(np.random.rand(1) * 2)
146             rand_b = float(np.random.rand(1) * 2)
147             rand_c = float(np.random.rand(1) * 2)
148             randomness.append((1 - rand_a) ** 2 + (1 - rand_b) ** 2
149                 ↪ + (1 - rand_c) ** 2)
150             revenue_estimates.append(
151                 market_profit * rand_a * gain * rand_b *
152                 ↪ shared_profit_fraction * rand_c
153             )
154         return revenue_estimates, randomness
155
156     def estimate_logistics_revenue(
157         self, N, gain_range, market_profit_range,
158         ↪ shared_profit_fraction_range, sam_factor, tam_factor
159     ):
160         print(f'market_profit_range={market_profit_range}')
161         print(f'shared_profit_fraction_range={
162             ↪ shared_profit_fraction_range}')
163         revenue_estimates = []
164         randomness = []
165         for i in range(0, N):
166             # TODO: change to get the range as specified in
167             ↪ datapoints per parameter
168             rand_market_profit = float(np.random.rand(1) * 5) #
169             ↪ factor 0 to 5 as the computed profit margin of
170             ↪ 0.0158 seems slightly low
171             rand_gain = float(np.random.rand(1) * 16) # map gain to
172             ↪ range of 0.1 to 16% based on McKinsey study
173             #rand_c = float(np.random.rand(1) * 2) # map profit
174             ↪ fraction from 0.1 to 10

```

```

164         rand_tam = float(np.random.rand(1) * 2) # map tam to
165             ↪ factor 2 as it is a rough estimate
166         rand_sam = float(np.random.rand(1) * 2) # map sam to
167             ↪ factor 2 as it is a rough estimate
168         randomness.append((1 - rand_market_profit) ** 2 + (1 -
169             ↪ rand_gain) ** 2 + (1 - rand_tam) ** 2 + (1 - rand_sam)
170             ↪ ** 2)
171         rand_market_profit_range = market_profit_range *
172             ↪ rand_market_profit
173         rand_gain_range = gain_range * rand_gain
174         rand_sam_range = sam_factor * rand_sam
175         rand_tam_range = tam_factor * rand_tam
176         #exit()
177         revenue_estimates.append(
178             rand_market_profit_range * rand_sam_range *
179             ↪ rand_tam_range * rand_gain_range
180         )
181     return revenue_estimates, randomness
182
183 def estimate_pharmaceuticals_revenue(self):
184     return 0
185
186 def estimate_algo_trading_revenue(self):
187     return 0
188
189 def estimate_material_sciences_revenue(self):
190     return 0
191
192 def estimate_telecommunications_revenue(self):
193     return 0
194
195 def plot_data(self, x, y):
196     N = self.nr_simulations
197
198     # Random colour for points, vector of length N
199     colors = np.ones(N)
200
201     # Plot figure
202     fig, ax = plt.subplots()
203
204     # Set y-axis scale to millions
205     scale_y = 1e6
206     ticks_y = ticker.FuncFormatter(lambda x, pos: "{0:g}".format(
207         ↪ x / scale_y))
208     ax.yaxis.set_major_formatter(ticks_y)
209
210     # Specify units of y-axis
211     ax.set_ylabel("$ million")
212
213     plt.scatter(x, y, c=colors, alpha=0.8)
214     plt.xlabel("Summed Squared Average Randomness")
215     plt.ylabel("Estimated revenue in $million/year")
216     plt.title("Monte-carlo simulation\n estimated total revenue
217         ↪ TruCol company")
218
219     # Export/save plot
220     # plt.show()
221     plt.savefig(
222         os.path.dirname(__file__)
223         + "/../..../latex/project"
224         + str(self.project_nr)
225         + "/Images/"

```



```

218         + "revenue_sum"
219         + ".png"
220     )
221
222     def plot_data_series(self, x_series, y_series):
223         x = [item for sublist in x_series for item in sublist]
224         y = [item for sublist in y_series for item in sublist]
225         N = len(x)
226
227         # random colour for points, vector of length N
228         colors, legend_colors = self.get_colors(x_series, y_series)
229
230         # Plot figure
231         fig, ax = plt.subplots()
232
233         # Set y-axis scale to millions
234         scale_y = 1e6
235         ticks_y = ticker.FuncFormatter(lambda x, pos: "{0:g}".format(
236             ↪ x / scale_y))
237         ax.yaxis.set_major_formatter(ticks_y)
238
239         # Specify units of y-axis
240         ax.set_ylabel("$ million")
241
242         # Manually create the legend based on hardcoded colours
243         logistics = mpatches.Patch(color="yellow", label="logistics")
244         algo_trading = mpatches.Patch(color="green", label="
245             ↪ algorithmic trading")
246         material_sciences = mpatches.Patch(color="cyan", label="
247             ↪ material sciences")
248         pharmaceuticals = mpatches.Patch(color="blue", label="
249             ↪ pharmaceuticals")
250         telecommunications = mpatches.Patch(color="magenta", label="
251             ↪ telecommunications")
252         plt.legend(
253             handles=[
254                 logistics,
255                 algo_trading,
256                 material_sciences,
257                 pharmaceuticals,
258                 telecommunications,
259             ]
260         )
261
262         # Generate the scatterplot
263         plt.scatter(x, y, c=colors, alpha=0.8)
264         plt.xlabel("Summed Squared Randomness")
265         plt.ylabel("Estimated revenue in $million/year")
266         plt.title(
267             "Monte-carlo simulation\n estimated revenue TruCol
268             ↪ company per sector"
269         )
270
271         # Export/save plot
272         # plt.show()
273         plt.savefig(
274             os.path.dirname(__file__)
275             + "/../../../../latex/project"
276             + str(self.project_nr)
277             + "/Images/"
278             + "revenue_per_sector"
279             + ".png"

```

```

274         )
275
276     def get_colors(self, x_series, y_series):
277
278         # Create list to store colors
279         color_arr = []
280
281         # Hardcode the colours for the dataserries
282         colors = ["yellow", "green", "cyan", "blue", "magenta"]
283
284         # Flatten the lists per sector into a single list
285         x = [item for sublist in x_series for item in sublist]
286
287         # Give each datapoint of a sector the same colour
288         for i in range(0, len(x_series)):
289             for elem in range(0, len(x_series[i])):
290                 color_arr.append(colors[i])
291         return color_arr, colors
292
293     def get_normal_dist(self):
294         # Creating a series of data of in range of 1-50.
295         x = np.linspace(1, 50, 200)
296
297         # Calculate mean and Standard deviation.
298         mean = np.mean(x)
299         sd = np.std(x)
300
301         # Apply function to the data.
302         pdf = self.normal_dist(x, mean, sd)
303
304         # Plotting the Results
305         plt.plot(x, pdf, color="red")
306         plt.xlabel("Data points")
307         plt.ylabel("Probability Density")
308
309     def normal_dist(self, x, mean, sd):
310         prob_density = (np.pi * sd) * np.exp(-0.5 * ((x - mean) / sd)
311             ↪ ** 2)
312         return prob_density
313
314     def addTwo(self, x):
315         """adds two to the incoming integer and returns the result of
316             ↪ the computation."""
317         return x + 2

```

---

## H Appendix Model\_value\_theory.py

---

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 from matplotlib import pyplot as plt
4 from matplotlib import lines
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 from .Plot_to_tex import Plot_to_tex as plt_tex
9
10
11 class Model_bottom_up:
12     def __init__(self):
13         pass
14
15     def addTwo(self, x):
16         """adds two to the incoming integer and returns the result of
17             ↪ the computation."""
18         return x + 2
```

---

## I Appendix Plot\_to\_tex.py

```
1  ### Call this from another file, for project 11, question 3b:
2  ### from Plot_to_tex import Plot_to_tex as plt_tex
3  ### multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints),
   ↪ dtype=int); # actually fill with data
4  ### lineLabels = [] # add a label for each dataseries
5  ### plt_tex.plotMultipleLines(plt_tex,single_x_series,
   ↪ multiple_y_series,"x-axis label [units]","y-axis label [units
   ↪ ]",lineLabels,"3b",4,11)
6  ### 4b=filename
7  ### 4 = position of legend, e.g. top right.
8  ###
9  ### For a single line, use:
10 ### plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)),
   ↪ dataseries,"x-axis label [units]","y-axis label [units]",
   ↪ lineLabel,"3b",4,11)
11
12 ### You can also plot a table directly into latex, see
   ↪ example_create_a_table(..)
13 ###
14 ### Then put it in latex with for example:
15 ### \begin{table}[H]
16 ###     \centering
17 ###     \caption{Results some computation.}\label{tab:some_computation
   ↪ }
18 ###     \begin{tabular}{|c|c|} % remember to update this to show all
   ↪ columns of table
19 ###         \hline
20 ###         \input{latex/project3/tables/q2.txt}
21 ###     \end{tabular}
22 ### \end{table}
23 import random
24 from matplotlib import lines
25 import matplotlib.pyplot as plt
26 import numpy as np
27 import os
28
29
30 class Plot_to_tex:
31     def __init__(self):
32         self.script_dir = self.get_script_dir()
33         print("Created main")
34
35     # plot graph (legendPosition = integer 1 to 4)
36     def plotSingleLine(
37         self,
38         x_path,
39         y_series,
40         x_axis_label,
41         y_axis_label,
42         label,
43         filename,
44         legendPosition,
45         project_nr,
46     ):
47         fig = plt.figure()
48         ax = fig.add_subplot(111)
49         ax.plot(x_path, y_series, c="b", ls="-", label=label,
   ↪ fillstyle="none")
50         plt.legend(loc=legendPosition)
51         plt.xlabel(x_axis_label)
```

```

52     plt.ylabel(y_axis_label)
53     plt.savefig(
54         os.path.dirname(__file__)
55         + "/../../../latex/project"
56         + str(project_nr)
57         + "/Images/"
58         + filename
59         + ".png"
60     )
61
62     #         plt.show();
63
64     # plot graphs
65     def plotMultipleLines(
66         self, x, y_series, x_label, y_label, label, filename,
67         ↪ legendPosition, project_nr
68     ):
69         fig = plt.figure()
70         ax = fig.add_subplot(111)
71
72         # generate colours
73         cmap = self.get_cmap(len(y_series[:, 0]))
74
75         # generate line types
76         lineTypes = self.generateLineTypes(y_series)
77
78         for i in range(0, len(y_series)):
79             # overwrite linetypes to single type
80             lineTypes[i] = "_"
81             ax.plot(
82                 x,
83                 y_series[i, :],
84                 ls=lineTypes[i],
85                 label=label[i],
86                 fillstyle="none",
87                 c=cmap(i),
88             )
89             # color
90
91         # configure plot layout
92         plt.legend(loc=legendPosition)
93         plt.xlabel(x_label)
94         plt.ylabel(y_label)
95         plt.savefig(
96             os.path.dirname(__file__)
97             + "/../../../latex/project"
98             + str(project_nr)
99             + "/Images/"
100             + filename
101             + ".png"
102         )
103
104         print(f"plotted lines")
105
106     # Generate random line colours
107     # Source: https://stackoverflow.com/questions/14720331/how-to-
108     ↪ generate-random-colors-in-matplotlib
109     def get_cmap(n, name="hsv"):
110         """Returns a function that maps each index in 0, 1, ..., n-1
111             ↪ to a distinct
112             RGB color; the keyword argument name must be a standard mpl
113             ↪ colormap name."""

```

```

110         return plt.cm.get_cmap(name, n)
111
112     def generateLineTypes(y_series):
113         # generate varying linetypes
114         typeOfLines = list(lines.lineStyles.keys())
115
116         while len(y_series) > len(typeOfLines):
117             typeOfLines.append("-.")
118
119         # remove void lines
120         for i in range(0, len(y_series)):
121             if typeOfLines[i] == "None":
122                 typeOfLines[i] = "-"
123             if typeOfLines[i] == ":":
124                 typeOfLines[i] = ":"
125             if typeOfLines[i] == " ":
126                 typeOfLines[i] = "--"
127         return typeOfLines
128
129     # Create a table with: table_matrix = np.zeros((4,4),dtype=object
130     ↪ ) and pass it to this object
131     def put_table_in_tex(self, table_matrix, filename, project_nr):
132         cols = np.shape(table_matrix)[1]
133         format = "%s"
134         for col in range(1, cols):
135             format = format + " & %s"
136         format = format + ""
137         plt.savetxt(
138             os.path.dirname(__file__)
139             + "/../../../latex/project"
140             + str(project_nr)
141             + "/tables/"
142             + filename
143             + ".txt",
144             table_matrix,
145             delimiter=" & ",
146             fmt=format,
147             newline=" \\\ \ \ \ \hline \n",
148         )
149
150     # replace this with your own table creation and then pass it to
151     ↪ put_table_in_tex(..)
152     def example_create_a_table(self):
153         project_nr = "1"
154         table_name = "example_table_name"
155         rows = 2
156         columns = 4
157         table_matrix = np.zeros((rows, columns), dtype=object)
158         table_matrix[:, :] = "" # replace the standard zeros with
159         ↪ empty cell
160         print(table_matrix)
161         for column in range(0, columns):
162             for row in range(0, rows):
163                 table_matrix[row, column] = row + column
164         table_matrix[1, 0] = "example"
165         table_matrix[0, 1] = "grid sizes"
166
167         self.put_table_in_tex(table_matrix, table_name, project_nr)
168
169     def get_script_dir(self):
170         """returns the directory of this script regardless of from
171         ↪ which level the code is executed"""

```

```
168         return os.path.dirname(__file__)
169
170
171 if __name__ == "__main__":
172     main = Plot_to_tex()
173     main.example_create_a_table()
```

---