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, [7], 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[3].
- **Serviceable available market (SAM)** is the portion of TAM targeted and served by a company's products or services[3].
- **Serviceable obtainable market (SOM)**, or share of market, is the percentage of SAM which is realistically reached[3].

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 section 4 (the Python models themselves are included as appendices in section E to section G respectively). The results of these models are presented in section 5. 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 section 6. Next, the results and sensitivity of the models are discussed in section 7 and a conclusion is provided in section 8.

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, section 3.1 describes the TAM in which the TruCol consultancy will operate. To this end, section 3.1.1 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 pharmaceutics market in section 3.1.2 to get some insight in their respective market sizes. Furthermore, section 3.1.3 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 optimalisation 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 optimalisation is documented by McKinsey & Company in the "how they help their clients" segment of their website[4]. The study how reports McKinsey's team, among which McKinsey's Strategic Network Analytic Center, helped an Asian logistics company. With McKinseys 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 optimalisation across 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 across the entire logistics sector by summing the net profit of the logistics sector. [2] claims that this company [8] valued the logistics market at 8.1 trillion in 2016. Additionally, [2] claims [8] estimates the logistics market value will grow to 15.5 trillion in 2023. However, no figures on profit are found. Therefore, individual companies are explored.

For DHL one can find on pdf page 37/170 in [5] that the annual profit for DHL in 2019 was 4.1 billion.

For UPS one can find on pdf page 4/257 in [9] that the annual unadjusted operating profit for UPS in 2020 was 7.7 billion. Note, [10] 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 [6].

- **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

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

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 $74.2 \cdot 0.001 \cdot 0.01 = \$0.74 million$

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 successful 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 [1].
- **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.

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 a 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.

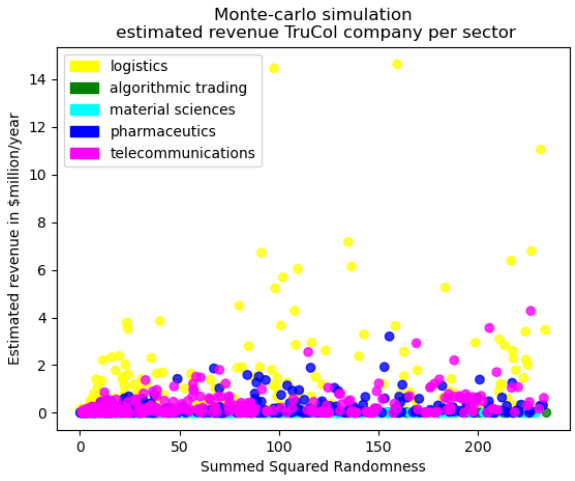

Figure 1: A scatterplot generated by a Monte-Carlo simulation to provide an impression on the estimated projected total addressable market per sector for the TruCol consultancy company.

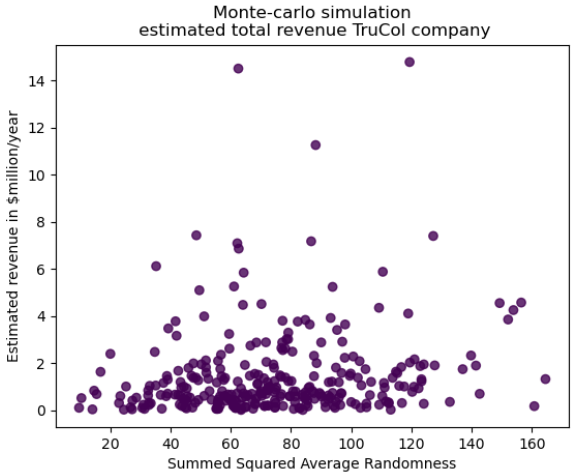

Figure 2: A scatterplot generated by a Monte-Carlo simulation to provide an estimate on the projected total addressable market for the TruCol consultancy company.

6 Sensitivity Analysis

6.1 Top Down

Omitted due to time constraints.

7 Discussion

We expect that the main points for improvement in this analysis are the datapoints that are used. In particular, the generalisation of the profit margin of the logistics sector to other sectors could be replaced by the actual data of the other sectors. Furthermore, the profit margin of the logistics sector could be searched directly instead of deriving it based on the of its larger companies.

8 Conclusion

A rough estimate based on various datasources has been generated to estimate the yearly revenue of the TruCol company. Additional iterations with improved datapoints is recommended to obtain a more accurate estimate. The market analysis does not yet include the growth that may be captured in diversification, emerging markets such as neuromorphic computing and in-house automation/AI-engines. Before taking these potentials into account, however, a more accurate assessment of the starting market is recommended.

References

[1] Zahed Allahyari and Artem R Oganov. Coevolutionary search for optimal materials in the space of all possible compounds. *npj Computational Materials*, 6(1):1–10, 2020.

[2] Andrew Allen. Logistics industry to be worth $15.5tn by 2023. `https://www.cips.org/supply-management/news/2016/november/logistics-industry-forecast-to-be-worth-155tn-by-2023/`. Accessed: 2021-07-11.

[3] The businessplan shop. Tam sam som - what it means and why it matters. `https://www.thebusinessplanshop.com/blog/en/entry/tam_sam_som`. Accessed: 2021-07-11.

[4] McKinsey & Company. Algorithmic route optimization improves revenue for a logistics company. `https://www.mckinsey.com/business-functions/mckinsey-analytics/how-we-help-clients/algorithmic-route-optimization-improves-revenue-for-a-logistics-company#`. Accessed: 2021-07-11.

[5] DHL. 2019 financial year - focussing on our profitable core. `https://www.dpdhl.com/content/dam/dpdhl/en/media-center/investors/documents/annual-reports/DPDHL-2019-Annual-Report.pdf`. Accessed: 2021-07-11.

[6] FedEx. Fedex corp. reports fourth quarter and full-year earnings. https://s21.q4cdn.com/665674268/files/doc_news/investor/2020/FedEx-Q4-FY20-Earnings-Release.pdf. Accessed: 2021-07-11.

[7] Haje Jan Kamps. *Pitch Deck Design*. Apress, Berkeley, CA, 2020.

[8] Transparency Market Research. Logistics market - snapshot. https://www.transparencymarketresearch.com/logistics-market.html. Accessed: 2021-07-11.

[9] UPS. Notice of 2021 annual meeting of shareowners and proxy statement & 2020 annual report on form 10-k. https://investors.ups.com/_assets/_67e21ed5c7d1164af5b2ef48cec32803/ups/db/1110/9465/annual_report/UPS_2021_Proxy_Statement_and_2020_Annual_Report%3B_Form_10-K.pdf. Accessed: 2021-07-11.

[10] UPS. Ups q21 earnings call. https://investors.ups.com/_assets/_67e21ed5c7d1164af5b2ef48cec32803/ups/db/1111/9835/file/UPS_1Q21_Earnings_Webcast_Deck_Final.pdf. Accessed: 2021-07-11.

A Appendix __main__.py

```
import os
from .Main import Main
from .Model_top_down import Model_top_down

print(f"Hi, I'll be running the main code, and I'll let you know when
    ↪ I'm done.")
project_nr = 1
main = Main()

# run monte-carlo for revenue estimation
model = Model_top_down(project_nr)


# export the code to latex
main.export_code_to_latex(project_nr)

# compile the latex report
main.compile_latex_report(project_nr)

print(f"Done.")
```

B Appendix Main.py

```python
# Example code that creates plots directly in report
# Code is an implementation of a genetic algorithm
import random
from matplotlib import pyplot as plt
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np

from .Compile_latex import Compile_latex
from .Plot_to_tex import Plot_to_tex as plt_tex
from .Export_code_to_latex import export_code_to_latex

# define global variables for genetic algorithm example
string_length = 100
mutation_chance = 1.0 / string_length
max_iterations = 1500


class Main:
    def __init__(self):
        pass

    def export_code_to_latex(self, project_nr):
        export_code_to_latex("main.tex", project_nr)

    def compile_latex_report(self, project_nr):
        """compiles latex code to pdf"""
        compile_latex = Compile_latex(project_nr, "main.tex")

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of
           ↪ the computation."""
        return x + 2


if __name__ == "__main__":
    # initialize main class
    main = Main()
```

C Appendix Compile_latex.py

```python
# runs a jupyter notebook and converts it to pdf

import os
import shutil
import nbformat
from nbconvert.preprocessors import ExecutePreprocessor


class Compile_latex:
    def __init__(self, project_nr, latex_filename):
        self.script_dir = self.get_script_dir()
        relative_dir = f"latex/project{project_nr}/"
        self.compile_latex(relative_dir, latex_filename)
        self.clean_up_after_compilation(latex_filename)
        self.move_pdf_into_latex_dir(relative_dir, latex_filename)

    # runs jupyter notebook
    def compile_latex(self, relative_dir, latex_filename):
        os.system(f"pdflatex {relative_dir}{latex_filename}")

    def clean_up_after_compilation(self, latex_filename):
        latex_filename_without_extention = latex_filename[:-4]
        print(f"latex_filename_without_extention={
            ↪ latex_filename_without_extention}")
        self.delete_file_if_exists(f"{
            ↪ latex_filename_without_extention}.aux")
        self.delete_file_if_exists(f"{
            ↪ latex_filename_without_extention}.log")
        self.delete_file_if_exists(f"texput.log")

    def move_pdf_into_latex_dir(self, relative_dir, latex_filename):
        pdf_filename = f"{latex_filename[:-4]}.pdf"
        destination = f"{self.get_script_dir()}/../../../{
            ↪ relative_dir}{pdf_filename}"

        try:
            shutil.move(pdf_filename, destination)
        except:
            print("Error while moving file ", pdf_filename)

    def delete_file_if_exists(self, filename):
        try:
            os.remove(filename)
        except:
            print(
                f"Error while deleting file: {filename} but that is
                    ↪ not too bad because the intention is for it to
                    ↪ not be there."
            )

    def get_script_dir(self):
        """returns the directory of this script regardles of from
            ↪ which level the code is executed"""
        return os.path.dirname(__file__)


if __name__ == "__main__":
    main = Compile_latex()
```

D Appendix Datapoints.py

```python
# The bottom up model that computes the TAM and TSM
import random
from matplotlib import pyplot as plt
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np

from .Plot_to_tex import Plot_to_tex as plt_tex


class Datapoints:
    def __init__(self):
        """ Initialise the datapoints and compute basic datapoints
        that can be derived from the datapoints and/or assumptions.
        """
        # Source: https://www.dpdhl.com/content/dam/dpdhl/en/media-center/investors/documents/annual-reports/DPDHL-2019-Annual-Report.pdf
        self.profit_dhl = 4.1 * 10 ** 9  # dollar
        # Source:
        self.profit_fedex = 1.29 * 10 ** 9  # dollar
        # Source: https://investors.ups.com/_assets/_67e21ed5c7d1164af5b2ef48cec32803/ups/db/1110/9465/annual_report/UPS_2021_Proxy_Statement_and_2020_Annual_Report%3B_Form_10-K.pdf
        self.profit_ups = 7.7 * 10 ** 9  # dollar
        # Sum the profit of these three companies
        self.profit_dhl_fedex_ups = (
            self.profit_dhl + self.profit_fedex + self.profit_ups
        )

        # sam and tam factors
        self.sam_factor=0.003
        self.tam_factor=0.004
        print(f'self.profit_dhl_fedex_ups={self.profit_dhl_fedex_ups}')

        # Source: https://www.cips.org/supply-management/news/2016/november/logistics-industry-forecast-to-be-worth-155tn-by-2023/
        # NOTE: this article seems an unreliable source and is outdated, hence the 0.15 should possibly be changed/updated.
        self.logistics_market_share_dhl_fedex_ups = 0.15

        # Compute the remaining market share.
        self.logistics_market_share_remaining = (
            1 - self.logistics_market_share_dhl_fedex_ups
        )

        # Assume avg market profit per dollar market share is uniform.
        self.logistics_market_profit = self.get_logistics_market_profit()
        #print(f'logistics_market_profit={logistics_market_profit}')


        # Conservative estimate based on 0.16 demonstrated by McKinsey & Company study.
```

```
        # Source: https://www.mckinsey.com/business-functions/mckinsey-analytics/how-we-help-clients/algorithmic-route-optimization-improves-revenue-for-a-logistics-company#
        self.profit_gain_by_trucol_protocol = 0.04

        # Estimate based on analogy where a good constraint modeller
        # can reach significant gains in algorithmic efficiency of solution.
        self.profit_gain_by_trucol_protocol_consultancy = 0.002
        # Conservative estimate based on a max of 1.00, for companies
        # that intend to improve algorithmic efficiency without increasing profit.
        self.fraction_of_profit_shared_with_trucol = 0.01

        # Source: Statistica.com or marketsandmarkets.com
        # TODO: re-find exact link
        self.logistics_market_size = 5.5 * 10 ** 12
        # Source: Statistica.com or marketsandmarkets.com
        # TODO: re-find exact link
        self.algo_trading_market_size = 11.1 * 10 ** 9
        # Source: Statistica.com or marketsandmarkets.com
        # TODO: re-find exact link
        self.material_sciences_market_size = 1 * 10 ** 9
        # Source: Statistica.com or marketsandmarkets.com
        # TODO: re-find exact link
        self.pharmaceutics_market_size = 1.27 * 10 ** 12
        # Source: Statistica.com or marketsandmarkets.com
        # TODO: re-find exact link
        self.telecommunications_market_size = 1.7 * 10 ** 12

        logistics_profit_fraction=self.logistics_market_profit/ self.logistics_market_size
        print(f'self.profit_dhl_fedex_ups={self.profit_dhl_fedex_ups} ')
        print(f'self.logistics_market_size={self.logistics_market_size}')
        print(f'logistics_profit_fraction={logistics_profit_fraction} ')

        # Compute and assume profit margins
        self.compute_profit_margins()  # for logistics sector the data is known
        self.assume_profit_margins()  # for the other sectors the data is assumed

        # Compute profit per market
        self.compute_market_profit()

    def get_logistics_market_profit(self):
        """ The basic computation that is done here is a
        cross multiplication of (see pdf):
        0.15/0.85=profit-three-companies/profit-remainder."""
        net_profit_remainder = (
            self.profit_dhl_fedex_ups
            * self.logistics_market_share_remaining
            / self.logistics_market_share_dhl_fedex_ups
        )
        net_profit_logistics_market = net_profit_remainder + self.profit_dhl_fedex_ups
        return net_profit_logistics_market

```

```
    def get_market_profit(self, market_size, profit_margin):
        return market_size * profit_margin

    def get_market_profit_margin(self, market_size, net_profit):
        return net_profit / market_size

    def compute_profit_margins(self):
        # Compute the profit margin based on market size and profit.
        self.logistics_market_profit_margin = self.
            ↪ get_market_profit_margin(
            self.logistics_market_size, self.logistics_market_profit
        )

    def assume_profit_margins(self):

        # Assume the profit margin in the algorithmic trading market
            ↪ equals that of the logistics market
        self.algo_trading_market_profit_margin = self.
            ↪ logistics_market_profit_margin

        # Assume the profit margin in the material sciences market
            ↪ equals that of the logistics market
        self.material_sciences_market_profit_margin = (
            self.logistics_market_profit_margin
        )

        # Assume the profit margin in the pharmaceutics market equals
            ↪  that of the logistics market
        self.pharmaceutics_market_profit_margin = self.
            ↪ logistics_market_profit_margin

        # Assume the profit margin in the telecommunications market
            ↪ equals that of the logistics market
        self.telecommunications_market_profit_margin = (
            self.logistics_market_profit_margin
        )

    def compute_market_profit(self):
        """ Computes the profit per market sector based on
        the assumption that the profit margin in each sector
        equals that of the logistics market."""
        self.algo_trading_market_profit = self.get_market_profit(
            self.algo_trading_market_size, self.
                ↪ algo_trading_market_profit_margin
        )
        self.material_sciences_market_profit = self.get_market_profit
            ↪ (
            self.material_sciences_market_size,
            self.material_sciences_market_profit_margin,
        )
        self.pharmaceutics_market_profit = self.get_market_profit(
            self.pharmaceutics_market_size, self.
                ↪ pharmaceutics_market_profit_margin
        )
        self.telecommunications_market_profit = self.
            ↪ get_market_profit(
            self.telecommunications_market_size,
            self.telecommunications_market_profit_margin,
        )
```

E Appendix Export_code_to_latex.py

```python
# runs a jupyter notebook and converts it to pdf
import os
import shutil
import nbformat
from nbconvert.preprocessors import ExecutePreprocessor


def export_code_to_latex(main_latex_filename, project_nr):
    """This function exports the python files and compiled pdfs of jupiter notebooks into the
    latex of the same project number. First it scans which appendices (without code, without
    notebooks) are already manually included in the main latex code. Next, all appendices
    that contain the python code are eiter found or created in the following order:
    First, the __main__.py file is included, followed by the main.py file, followed by all
    python code files in alphabetic order. After this, all the pdfs of the compiled notebooks
    are added in alphabetic order of filename. This order of appendices is overwritten in the
    main tex file.

    :param main_latex_filename: Name of the main latex document of this project number
    :param project_nr: The number  indicating which project this code pertains to.
    """
    script_dir = get_script_dir()
    relative_dir = f"latex/project{project_nr}/"
    appendix_dir = script_dir + "/../../../" + relative_dir + "Appendices/"
    path_to_main_latex_file = (
        f"{script_dir}/../../../{relative_dir}/{main_latex_filename}"
    )
    root_dir = script_dir[0 : script_dir.rfind(f"code/project{project_nr}")]

    # Get paths to files containing python code.
    python_filepaths = get_filenames_in_dir("py", script_dir, ["__init__.py"])
    compiled_notebook_pdf_filepaths = get_compiled_notebook_paths(script_dir)

    # Check which files are already included in the latex appendicess.
    python_files_already_included_in_appendices = get_code_files_already_included_in_appendices(
        python_filepaths, appendix_dir, ".py", project_nr, root_dir
    )
    notebook_pdf_files_already_included_in_appendices = get_code_files_already_included_in_appendices(
        compiled_notebook_pdf_filepaths, appendix_dir, ".ipynb", project_nr, root_dir,
    )

    # Get which appendices are still missing.
    missing_python_files_in_appendices = get_code_files_not_yet_included_in_appendices(
```

```
        python_filepaths, python_files_already_included_in_appendices
            ↪ , ".py"
    )
    missing_notebook_files_in_appendices =
        ↪ get_code_files_not_yet_included_in_appendices(
        compiled_notebook_pdf_filepaths,
        notebook_pdf_files_already_included_in_appendices,
        ".pdf",
    )

    # Create the missing appendices.
    created_python_appendix_filenames = create_appendices_with_code(
        appendix_dir, missing_python_files_in_appendices, ".py",
            ↪ project_nr, root_dir
    )

    created_notebook_appendix_filenames = create_appendices_with_code
        ↪ (
        appendix_dir,
        missing_notebook_files_in_appendices,
        ".ipynb",
        project_nr,
        root_dir,
    )

    appendices = get_list_of_appendix_files(
        appendix_dir, compiled_notebook_pdf_filepaths,
            ↪ python_filepaths
    )

    main_tex_code, start_index, end_index, appendix_tex_code =
        ↪ get_appendix_tex_code(
        path_to_main_latex_file
    )

    # assumes non-included non-code appendices should not be included
        ↪ :
    # overwrite the existing appendix lists with the current appendix
        ↪  list.
    (
        non_code_appendices,
        main_non_code_appendix_inclusion_lines,
    ) = get_order_of_non_code_appendices_in_main(appendices,
        ↪ appendix_tex_code)

    python_appendix_filenames = list(
        map(
            lambda x: x.appendix_filename,
            filter_appendices_by_type(appendices, "python"),
        )
    )
    sorted_created_python_appendices = sort_python_appendices(
        filter_appendices_by_type(appendices, "python")
    )
    sorted_python_appendix_filenames = list(
        map(lambda x: x.appendix_filename,
            ↪ sorted_created_python_appendices)
    )

    notebook_appendix_filenames = list(
        map(
            lambda x: x.appendix_filename,
```

```
            filter_appendices_by_type(appendices, "notebook"),
        )
    )
    sorted_created_notebook_appendices =
       ↪ sort_notebook_appendices_alphabetically(
        filter_appendices_by_type(appendices, "notebook")
    )
    sorted_notebook_appendix_filenames = list(
        map(lambda x: x.appendix_filename,
           ↪ sorted_created_notebook_appendices)
    )

    appendix_latex_code = create_appendices_latex_code(
        main_non_code_appendix_inclusion_lines,
        sorted_created_notebook_appendices,
        project_nr,
        sorted_created_python_appendices,
    )

    updated_main_tex_code = substitute_appendix_code(
        end_index, main_tex_code, start_index, appendix_latex_code
    )

    overwrite_content_to_file(updated_main_tex_code,
       ↪ path_to_main_latex_file)


def create_appendices_latex_code(
    main_non_code_appendix_inclusion_lines,
    notebook_appendices,
    project_nr,
    python_appendices,
):
    """Creates the latex code that includeds the appendices in the
       ↪ main latex file.

    :param main_non_code_appendix_inclusion_lines: latex code that
       ↪ includes the appendices that do not contain python code nor
       ↪  notebooks
    :param notebook_appendices: List of Appendix objects representing
       ↪  appendices that include the pdf files of compiled Jupiter
       ↪ notebooks
    :param project_nr: The number indicating which project this code
       ↪ pertains to.
    :param python_appendices: List of Appendix objects representing
       ↪ appendices that include the python code files.
    """
    main_appendix_inclusion_lines =
       ↪ main_non_code_appendix_inclusion_lines

    appendices_of_all_types = [python_appendices, notebook_appendices
       ↪ ]
    main_appendix_inclusion_lines.append(
        f"\IfFileExists{{latex/project{project_nr}/main.tex}}{{"
    )
    main_appendix_inclusion_lines = append_latex_inclusion_command(
        appendices_of_all_types, True, main_appendix_inclusion_lines,
           ↪  project_nr,
    )
    main_appendix_inclusion_lines.append(f"}}{{")
    main_appendix_inclusion_lines = append_latex_inclusion_command(
```

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

```
        i for i in appendices if i.code_filename is not None
    ]
    appendices_sorted_a_z = sort_appendices_on_code_filename(
        filtered_remaining_appendices
    )
    return return_appendices + appendices_sorted_a_z


def sort_notebook_appendices_alphabetically(appendices):
    """Sorts notebook appendix objects alphabetic order of their pdf
       ↪ filenames.

    :param appendices: List of Appendix objects
    """
    return_appendices = []
    filtered_remaining_appendices = [
        i for i in appendices if i.code_filename is not None
    ]
    appendices_sorted_a_z = sort_appendices_on_code_filename(
        filtered_remaining_appendices
    )
    return return_appendices + appendices_sorted_a_z


def sort_appendices_on_code_filename(appendices):
    """Returns a list of Appendix objects that are sorted and  based
       ↪ on the property: code_filename.
    Assumes the incoming appendices only contain python files.

    :param appendices: List of Appendix objects
    """
    attributes = list(map(lambda x: x.code_filename, appendices))
    sorted_indices = sorted(range(len(attributes)), key=lambda k:
       ↪ attributes[k])
    sorted_list = []
    for i in sorted_indices:
        sorted_list.append(appendices[i])
    return sorted_list


def get_order_of_non_code_appendices_in_main(appendices,
   ↪ appendix_tex_code):
    """Scans the lines of appendices in the main code, and returns
       ↪ the lines
    of the appendices that do not contain code, in the order in which
       ↪  they were
    included in the main latex file.

    :param appendices: List of Appendix objects
    :param appendix_tex_code: latex code from the main latex file
       ↪ that includes the appendices
    """
    non_code_appendices = []
    non_code_appendix_lines = []
    appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
    for line in appendix_tex_code:
        appendix_filename = get_filename_from_latex_appendix_line(
           ↪ appendices, line)

        # Check if line is not commented
        if not appendix_filename is None:
            if not line_is_commented(line, appendix_filename):
```

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

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

```
                Appendix(appendix_filepath, appendix_filecontent,
                    ↪ appendix_type)
            )
    return appendices


def get_filename_from_latex_inclusion_command(
    appendix_line, extension, start_substring
):
    """returns the code/notebook filename in a latex command which
        ↪ includes that code in an appendix.
    The inclusion command includes a python code or jupiter notebook
        ↪ pdf.

    :param appendix_line: :Line of latex code (in particular expected
        ↪  to be the latex code from an appendix.).
    :param extension: The file extension of the file that is sought
        ↪ in the appendix line. Either ".py" or ".pdf".
    :param start_substring: The substring that characterises the
        ↪ latex inclusion command.
    """
    start_index = appendix_line.index(start_substring)
    end_index = appendix_line.index(extension)
    return get_filename_from_dir(
        appendix_line[start_index : end_index + len(extension)]
    )


def get_filenames_in_dir(extension, path, excluded_files=None):
    """Returns a list of the relative paths to all files within the
        ↪ some path that match
    the given file extension.

    :param extension: The file extension of the file that is sought
        ↪ in the appendix line. Either ".py" or ".pdf".
    :param path: Absolute filepath in which files are being sought.
    :param excluded_files: (Default value = None) Files that will not
        ↪  be included even if they are found.
    """
    filepaths = []
    for r, d, f in os.walk(path):
        for file in f:
            if file.endswith(extension):
                if (excluded_files is None) or (
                    (not excluded_files is None) and (not file in
                        ↪ excluded_files)
                ):
                    filepaths.append(r + "/" + file)
    return filepaths


def get_code_files_already_included_in_appendices(
    absolute_code_filepaths, appendix_dir, extension, project_nr,
        ↪ root_dir
):
    """Returns a list of code filepaths that are already properly
        ↪ included the latex appendix files of this project.

    :param absolute_code_filepaths: List of absolute paths to the
        ↪ code files (either python files or compiled jupyter
        ↪ 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
431 def check_if_appendix_contains_file(
432     appendix_content, code_filepath, extension, project_nr, root_dir
433 ):
434     """Scans an appendix content to determine whether it contains a
            ↪ substring that
435     includes a code file (of either python or compiled notebook=pdf
            ↪ extension).
436
437     :param appendix_content: content in an appendix latex file.
438     :param code_filepath: Absolute path to a code file (either python
            ↪  files or compiled jupyter notebook pdfs).
439     :param extension: The file extension of the file that is sought
            ↪ in the appendix line. Either ".py" or ".pdf".
440     :param project_nr: The number  indicating which project this code
            ↪  pertains to.
441     :param root_dir: The root directory of this repository.
442     """
443     # convert code_filepath to the inclusion format in latex format
444     latex_relative_filepath = (
445         f"latex/project{project_nr}/../../{code_filepath[len(root_dir
                ↪ ):]}"
446     )
447     latex_command = get_latex_inclusion_command(extension,
            ↪ latex_relative_filepath)
448     return get_line_of_latex_command(appendix_content, latex_command)
449
450
451 def get_line_of_latex_command(appendix_content, latex_command):
452     """Returns the line number of a latex command if it is found.
            ↪ Returns -1 otherwise.
```

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

```
    """Reads content of a file and returns it as a list of strings,
       ↪ with one string per line.

    :param filepath: path towards the file that is being read.
    """
    with open(filepath) as f:
        content = f.readlines()
    return content


def get_code_files_not_yet_included_in_appendices(
    code_filepaths, contained_codes, extension
):
    """Returns a list of filepaths that are not yet properly included
       ↪  in some appendix of this project.

    :param code_filepath: Absolute path to all the code files in
       ↪ this project (source directory).
    (either python files or compiled jupyter notebook pdfs).
    :param contained_codes: list of  Appendix objects that include
       ↪ either python files or compiled jupyter notebook pdfs,
       ↪ which
    are already included in the appendix tex files. (Does not care
       ↪ whether those appendices are also actually
    included in the main or not.)
    :param extension: The file extension of the file that is sought
       ↪ in the appendix line. Either ".py" or ".pdf".
    """
    contained_filepaths = list(
        map(lambda contained_file: contained_file.code_filepath,
           ↪ contained_codes)
    )
    not_contained = []
    for filepath in code_filepaths:
        if not filepath in contained_filepaths:
            not_contained.append(filepath)
    return not_contained


def create_appendices_with_code(
    appendix_dir, code_filepaths, extension, project_nr, root_dir
):
    """Creates the latex appendix files in with relevant codes
       ↪ included.

    :param appendix_dir: Absolute path that contains the appendix .
       ↪ tex files.
    :param code_filepaths: Absolute path to code files that are not
       ↪ yet included in an appendix
    (either python files or compiled jupyter notebook pdfs).
    :param extension: The file extension of the file that is sought
       ↪ in the appendix line. Either ".py" or ".pdf".
    :param project_nr: The number  indicating which project this code
       ↪  pertains to.
    :param root_dir: The root directory of this repository.
    """
    appendix_filenames = []
    appendix_reference_index = (
        get_index_of_auto_generated_appendices(appendix_dir,
           ↪ extension) + 1
    )

```

```
    for code_filepath in code_filepaths:
        latex_relative_filepath = (
            f"latex/project{project_nr}/../../{code_filepath[len(root_dir):]}"
        )
        code_path_from_latex_main_path = f"../../{code_filepath[len(root_dir):]}"
        content = []
        filename = get_filename_from_dir(code_filepath)

        content = create_section(appendix_reference_index, filename, content)
        content = add_include_code_in_appendix(
            content,
            code_filepath,
            code_path_from_latex_main_path,
            extension,
            latex_relative_filepath,
            project_nr,
            root_dir,
        )

        overwrite_content_to_file(
            content,
            f"{appendix_dir}Auto_generated_{extension[1:]}_App{appendix_reference_index}.tex",
            False,
        )
        appendix_filenames.append(
            f"Auto_generated_{extension[1:]}_App{appendix_reference_index}.tex"
        )
        appendix_reference_index = appendix_reference_index + 1
    return appendix_filenames


def add_include_code_in_appendix(
    content,
    code_filepath,
    code_path_from_latex_main_path,
    extension,
    latex_relative_filepath,
    project_nr,
    root_dir,
):
    """Includes the latex code that includes code in the script.

    :param content: The latex content that is being written to an appendix.
    :param code_path_from_latex_main_path: the path to the code as seen from the folder that contains main.tex.
    :param extension: The file extension of the file that is sought in the appendix line. Either ".py" or ".pdf".
    :param latex_relative_filepath_to_codefile: The latex compilation requires a relative path towards code files
    that are included. Therefore, a relative path towards the code is given.
    """
    content.append(
        f"\IfFileExists{{latex/project{project_nr}/../../{code_filepath[len(root_dir):]}}}{{"
    )
```

```
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
            ↪ in the appendix line. Either ".py" or ".pdf".
616     :param appendix_dir: Absolute path that contains the appendix .
            ↪ tex files.
617     """
618     max_index = -1
619     appendices =
            ↪ get_auto_generated_appendix_filenames_of_specific_extension
            ↪ (
620         appendix_dir, extension
621     )
622     for appendix in appendices:
623         substring = f"Auto_generated_{extension[1:]}_App"
624         # remove left of index
625         remainder = appendix[appendix.rfind(substring) + len(
                ↪ substring) :]
626         # remove right of index
627         index = int(remainder[:-4])
628         if index > max_index:
629             max_index = index
630     return max_index
631
632
633 def get_auto_generated_appendix_filenames_of_specific_extension(
634     appendix_dir, extension
635 ):
636     """Returns the list of auto generated appendices of
637     a specific extension type.
638
639     :param extension: The file extension of the file that is sought
            ↪ in the appendix line. Either ".py" or ".pdf".
640     :param appendix_dir: Absolute path that contains the appendix .
            ↪ tex files.
641     """
642     appendices_of_extension_type = []
643
644     # get all appendices
645     appendix_files = get_filenames_in_dir(".tex", appendix_dir)
646
647     # get appendices of particular extention type
648     for appendix_filepath in appendix_files:
649         right_of_slash = appendix_filepath[appendix_filepath.rfind("/
                ↪ ") + 1 :]
650         if (
651             right_of_slash[: 15 + len(extension) - 1]
652             == 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
        ↪ contains a section that
660     indicates the section is an appendix, and indicates which pyhon
        ↪ or notebook file is
661     being included in that appendix.
662 
663     :param appendix_reference_index: A counter that is used in the
        ↪ label to ensure the appendix section labels are unique.
664     :param code_filename: file name of the code file that is included
665     :param content: A list of strings that make up the appendix, with
        ↪  one line per element.
666     """
667     # write section
668     left = "\section{Appendix "
669     middle = code_filename.replace("_", "\_")
670     right = "}\label{app:"
671     end = "}"  # TODO: update appendix reference index
672     content.append(f"{left}{middle}{right}{appendix_reference_index}{
        ↪ end}")
673     return content
674 
675 
676 def overwrite_content_to_file(content, filepath, content_has_newlines
    ↪ =True):
677     """Writes a list of lines of tex code from the content argument
        ↪ to a .tex file
678     using overwriting method. The content has one line per element.
679 
680     :param content: The content that is being written to file.
681     :param filepath: Path towards the file that is being read.
682     :param content_has_newlines:  (Default value = True)
683     """
684     with open(filepath, "w") as f:
685         for line in content:
686             if content_has_newlines:
687                 f.write(line)
688             else:
689                 f.write(line + "\n")
690 
691 
692 def get_appendix_tex_code(main_latex_filename):
693     """gets the latex appendix code from the main tex file.
694 
695     :param main_latex_filename: Name of the main latex document of
        ↪ this project number
696     """
697     main_tex_code = read_file(main_latex_filename)
698     start = "\\begin{appendices}"
699     end = "\end{appendices}"
700     start_index = get_index_of_substring_in_list(main_tex_code, start
        ↪ ) + 1
701     end_index = get_index_of_substring_in_list(main_tex_code, end)
702     return main_tex_code, start_index, end_index, main_tex_code[
        ↪ start_index:end_index]
703 
704 
```

```
def get_index_of_substring_in_list(lines, target_substring):
    """Returns the index of the line in which the first character of a latex substring if it is found
    uncommented in the incoming list.

    :param lines: List of lines of latex code.
    :param target_substring: Some latex command/code that is sought in the incoming text.
    """
    for i in range(0, len(lines)):
        if target_substring in lines[i]:
            if not line_is_commented(lines[i], target_substring):
                return i


def update_appendix_tex_code(appendix_filename, is_from_root_dir, project_nr):
    """Returns the latex command that includes an appendix .tex file in an appendix environment
    as can be used in the main tex file.

    :param appendix_filename: Name of the appendix that is included by the generated command.
    :param project_nr: The number indicating which project this code pertains to.
    """
    if is_from_root_dir:
        left = f"\input{{latex/project{project_nr}/"
    else:
        left = "\input{"
    middle = "Appendices/"
    right = "} \\newpage\n"
    return f"{left}{middle}{appendix_filename}{right}"


def substitute_appendix_code(
    end_index, main_tex_code, start_index, updated_appendices_tex_code
):
    """Replaces the old latex code that included the appendices in the main.tex file with the new latex
    commands that include the appendices in the latex report.

    :param end_index: Index at which the appendix section ends right before the latex \end{appendix} line,
    :param main_tex_code: The code that is saved in the main .tex file.
    :param start_index: Index at which the appendix section starts right after the latex \begin{appendix} line,
    :param updated_appendices_tex_code: The newly created code that includes all the relevant appendices.
    (relevant being (in order): manually created appendices, python codes, pdfs of compiled jupiter notebooks).
    """
    updated_main_tex_code = (
        main_tex_code[0:start_index]
        + updated_appendices_tex_code
        + main_tex_code[end_index:]
    )
    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 
762 def get_script_dir():
763     """returns the directory of this script regardles of from which
        ↪ level the code is executed"""
764     return os.path.dirname(__file__)
765 
766 
767 class Appendix_with_code:
768     """stores in which appendix file and accompanying line number in
        ↪ the appendix in which a code file is
769     already included. Does not take into account whether this
        ↪ appendix is in the main tex file or not
770     """
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 
787 class Appendix:
788     """stores in appendix files and type of appendix."""
789 
790     def __init__(
791         self,
792         appendix_filepath,
793         appendix_content,
794         appendix_type,
795         code_filename=None,
796         appendix_inclusion_line=None,
797     ):
798         self.appendix_filepath = appendix_filepath
799         self.appendix_filename = get_filename_from_dir(self.
            ↪ appendix_filepath)
800         self.appendix_content = appendix_content
801         self.appendix_type = appendix_type  # TODO: perform
            ↪ validation of input values
802         self.code_filename = code_filename
803         self.appendix_inclusion_line = appendix_inclusion_line
```

F Appendix Model_bottom_up.py

```
# The bottom up model that computes the TAM and TSM
import random
from matplotlib import pyplot as plt
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np

from .Plot_to_tex import Plot_to_tex as plt_tex


class Model_bottom_up:
    def __init__(self):
        pass

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of
           ↪  the computation."""
        return x + 2
```

G Appendix Model_top_down.py

```
# The bottom up model that computes the TAM and TSM

from matplotlib import lines
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.ticker as ticker
import numpy as np
import os
import random

from .Plot_to_tex import Plot_to_tex as plt_tex
from .Datapoints import Datapoints


class Model_top_down:
    def __init__(self, project_nr):
        self.project_nr = project_nr
        self.nr_simulations = 300
        self.dp = Datapoints()
        x_series, y_series = self.estimate_revenue()
        self.plot_data_series(x_series, y_series)
        # self. get normal dist ()
        y = self.sum_revenues(y_series)
        x = self.avg_randomness(x_series)

        self.plot_data(x, y)

    def sum_revenues(self, x_series):
        summed_series = []
        for i in range(0, len(x_series[0])):
            summed = 0
            for j in range(0, len(x_series)):
                summed = summed + x_series[j][i]

            summed_series.append(summed)
        return summed_series

    def avg_randomness(self, series):
        summed_series = []
        for i in range(0, len(series[0])):
            summed = 0
            for j in range(0, len(series)):
                summed = summed + series[j][i]
            summed_series.append(summed / len(series))
        return summed_series

    def estimate_revenue(self):
        revenue_logistics, randomness_logistics = self.
            ↪ estimate_logistics_revenue(
            self.nr_simulations,
            self.dp.profit_gain_by_trucol_protocol_consultancy,
            self.dp.logistics_market_profit,
            self.dp.fraction_of_profit_shared_with_trucol,
            self.dp.sam_factor,
            self.dp.tam_factor
        )
        # self.plot_data(randomness_logistics, revenue_logistics)

        # algo
```

```
        revenue_algo_trading, randomness_algo_trading = self.
            ↪ estimate_logistics_revenue(
            self.nr_simulations,
            self.dp.profit_gain_by_trucol_protocol_consultancy,
            self.dp.algo_trading_market_profit,
            self.dp.fraction_of_profit_shared_with_trucol,
            self.dp.sam_factor,
            self.dp.tam_factor
        )
        # self.plot_data(randomness_algo_trading,
            ↪ revenue_algo_trading)

        # material
        (
            revenue_material_sciences,
            randomness_material_sciences,
        ) = self.estimate_logistics_revenue(
            self.nr_simulations,
            self.dp.profit_gain_by_trucol_protocol_consultancy,
            self.dp.material_sciences_market_profit,
            self.dp.fraction_of_profit_shared_with_trucol,
            self.dp.sam_factor,
            self.dp.tam_factor
        )
        print(
            f"material_sciences_market_profit={self.dp.
                ↪ material_sciences_market_profit}"
        )
        # self.plot_data(randomness_material_sciences,
            ↪ material_sciences_market_profit)

        # pharma
        (
            revenue_pharmaceutics,
            randomness_pharmaceutics,
        ) = self.estimate_logistics_revenue(
            self.nr_simulations,
            self.dp.profit_gain_by_trucol_protocol_consultancy,
            self.dp.pharmaceutics_market_profit,
            self.dp.fraction_of_profit_shared_with_trucol,
            self.dp.sam_factor,
            self.dp.tam_factor
        )
        print(f"pharmaceutics_market_profit={self.dp.
            ↪ pharmaceutics_market_profit}")
        # self.plot_data(randomness_pharmaceutics,
            ↪ revenue_pharmaceutics)

        # tele
        (
            revenue_telecommunications,
            randomness_telecommunications,
        ) = self.estimate_logistics_revenue(
            self.nr_simulations,
            self.dp.profit_gain_by_trucol_protocol_consultancy,
            self.dp.telecommunications_market_profit,
            self.dp.fraction_of_profit_shared_with_trucol,
            self.dp.sam_factor,
            self.dp.tam_factor
        )
        print(
```

```
            f"telecommunications_market_profit={self.dp.telecommunications_market_profit}"
        )
        # self.plot_data(randomness_telecommunications, revenue_telecommunications)

        # Concatenate all datapoints
        x_series = [
            randomness_logistics,
            randomness_algo_trading,
            randomness_material_sciences,
            randomness_pharmaceutics,
            randomness_telecommunications,
        ]
        y_series = [
            revenue_logistics,
            revenue_algo_trading,
            revenue_material_sciences,
            revenue_pharmaceutics,
            revenue_telecommunications,
        ]
        return x_series, y_series

#     def estimate_logistics_revenue(
#         self, N, gain, market_profit, shared_profit_fraction
#     ):
#         revenue_estimates = []
#         randomness = []
#         for i in range(0, N):
#             # TODO: change to get the range as specified in datapoints per parameter
#             rand_a = float(np.random.rand(1) * 2)
#             rand_b = float(np.random.rand(1) * 2)
#             rand_c = float(np.random.rand(1) * 2)
#             randomness.append((1 - rand_a) ** 2 + (1 - rand_b) ** 2 + (1 - rand_c) ** 2)
#             revenue_estimates.append(
#                 market_profit * rand_a * gain * rand_b * shared_profit_fraction * rand_c
#             )
#
#         return revenue_estimates, randomness

    def estimate_logistics_revenue(
        self, N, gain_range, market_profit_range, shared_profit_fraction_range, sam_factor, tam_factor
    ):
        print(f'market_profit_range={market_profit_range}')
        print(f'shared_profit_fraction_range={shared_profit_fraction_range}')

        revenue_estimates = []
        randomness = []
        for i in range(0, N):
            # TODO: change to get the range as specified in datapoints per parameter
            rand_market_profit = float(np.random.rand(1) * 5) # factor 0 to 5 as the computed profit margin of 0.0158 seems slightly low
            rand_gain = float(np.random.rand(1) * 16) # map gain to range of 0.1 to 16% based on McKinsey study
```

```
164             #rand_c = float(np.random.rand(1) * 2) # map profit
                    ↪ fraction from 0.1 to 10
165             rand_tam = float(np.random.rand(1) *2) # map tam to
                    ↪ factor 2 as it is a rough estimate
166             rand_sam = float(np.random.rand(1) * 2) # map samto
                    ↪ factor 2 as it is a rough estimate
167             rand_profit_fraction = float(np.random.rand(1) * 50) #
                    ↪ map profit fraction (shared with TruCol) from 1 to
                    ↪ 50%
168             randomness.append((1 - rand_market_profit) ** 2 + (1 -
                    ↪ rand_gain) ** 2 +(1 - rand_tam) ** 2+(1 - rand_sam)
                    ↪  ** 2)
169             rand_market_profit_range=market_profit_range *
                    ↪ rand_market_profit
170             rand_gain_rainge= gain_range * rand_gain
171             rand_sam_range=sam_factor*rand_sam
172             rand_tam_range=tam_factor*rand_tam
173
174             rand_shared_profit_fraction_range=
                    ↪ shared_profit_fraction_range*rand_profit_fraction
175             #exit()
176             revenue_estimates.append(
177                  rand_market_profit_range*rand_sam_range*
                        ↪ rand_tam_range*rand_gain_rainge*
                        ↪ rand_profit_fraction
178             )
179         return revenue_estimates, randomness
180
181     def estimate_pharmaceutics_revenue(self):
182         return 0
183
184     def estimate_algo_trading_revenue(self):
185         return 0
186
187     def estimate_material_sciences_revenue(self):
188         return 0
189
190     def estimate_telecommunications_revenue(self):
191         return 0
192
193     def plot_data(self, x, y):
194         N = self.nr_simulations
195
196         # Random colour for points, vector of length N
197         colors = np.ones(N)
198
199         # Plot figure
200         fig, ax = plt.subplots()
201
202         # Set y-axis scale to millions
203         scale_y = 1e6
204         ticks_y = ticker.FuncFormatter(lambda x, pos: "{0:g}".format(
                    ↪ x / scale_y))
205         ax.yaxis.set_major_formatter(ticks_y)
206
207         # Specify units of y-axis
208         ax.set_ylabel("$ million")
209
210         plt.scatter(x, y, c=colors, alpha=0.8)
211         plt.xlabel("Summed Squared Average Randomness")
212         plt.ylabel("Estimated revenue in $million/year")
```

```
        plt.title("Monte-carlo simulation\n estimated total revenue TruCol company")

        # Export/save plot
        # plt.show()
        plt.savefig(
            os.path.dirname(__file__)
            + "/../../../latex/project"
            + str(self.project_nr)
            + "/Images/"
            + "revenue_sum"
            + ".png"
        )

    def plot_data_series(self, x_series, y_series):
        x = [item for sublist in x_series for item in sublist]
        y = [item for sublist in y_series for item in sublist]
        N = len(x)

        # random colour for points, vector of length N
        colors, legend_colors = self.get_colors(x_series, y_series)

        # Plot figure
        fig, ax = plt.subplots()

        # Set y-axis scale to millions
        scale_y = 1e6
        ticks_y = ticker.FuncFormatter(lambda x, pos: "{0:g}".format(x / scale_y))
        ax.yaxis.set_major_formatter(ticks_y)

        # Specify units of y-axis
        ax.set_ylabel("$ million")

        # Manually create the legend based on hardcoded colours
        logistics = mpatches.Patch(color="yellow", label="logistics")
        algo_trading = mpatches.Patch(color="green", label="algorithmic trading")
        material_sciences = mpatches.Patch(color="cyan", label="material sciences")
        pharmaceutics = mpatches.Patch(color="blue", label="pharmaceutics")
        telecommunications = mpatches.Patch(color="magenta", label="telecommunications")
        plt.legend(
            handles=[
                logistics,
                algo_trading,
                material_sciences,
                pharmaceutics,
                telecommunications,
            ]
        )

        # Generate the scatterplot
        plt.scatter(x, y, c=colors, alpha=0.8)
        plt.xlabel("Summed Squared Randomness")
        plt.ylabel("Estimated revenue in $million/year")
        plt.title(
            "Monte-carlo simulation\n estimated revenue TruCol company per sector"
        )
```

```
268
269         # Export/save plot
270         # plt.show()
271         plt.savefig(
272             os.path.dirname(__file__)
273             + "/../../../latex/project"
274             + str(self.project_nr)
275             + "/Images/"
276             + "revenue_per_sector"
277             + ".png"
278         )
279
280     def get_colors(self, x_series, y_series):
281
282         # Create list to store colors
283         color_arr = []
284
285         # Hardcode the colours for the dataseries
286         colors = ["yellow", "green", "cyan", "blue", "magenta"]
287
288         # Flatten the lists per sector into a single list
289         x = [item for sublist in x_series for item in sublist]
290
291         # Give each datapoint of a sector the same colour
292         for i in range(0, len(x_series)):
293             for elem in range(0, len(x_series[i])):
294                 color_arr.append(colors[i])
295         return color_arr, colors
296
297     def get_normal_dist(self):
298         # Creating a series of data of in range of 1-50.
299         x = np.linspace(1, 50, 200)
300
301         # Calculate mean and Standard deviation.
302         mean = np.mean(x)
303         sd = np.std(x)
304
305         # Apply function to the data.
306         pdf = self.normal_dist(x, mean, sd)
307
308         # Plotting the Results
309         plt.plot(x, pdf, color="red")
310         plt.xlabel("Data points")
311         plt.ylabel("Probability Density")
312
313     def normal_dist(self, x, mean, sd):
314         prob_density = (np.pi * sd) * np.exp(-0.5 * ((x - mean) / sd)
                ↪  ** 2)
315         return prob_density
316
317     def addTwo(self, x):
318         """adds two to the incoming integer and returns the result of
                ↪  the computation."""
319         return x + 2
```

H Appendix Model_value_theory.py

```
# The bottom up model that computes the TAM and TSM
import random
from matplotlib import pyplot as plt
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np

from .Plot_to_tex import Plot_to_tex as plt_tex


class Model_bottom_up:
    def __init__(self):
        pass

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of
           ↪ the computation."""
        return x + 2
```

I Appendix Plot_to_tex.py

```
### Call this from another file, for project 11, question 3b:
### from Plot_to_tex import Plot_to_tex as plt_tex
### multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints), dtype=int); # actually fill with data
### lineLabels = [] # add a label for each dataseries
### plt_tex.plotMultipleLines(plt_tex,single_x_series, multiple_y_series,"x-axis label [units]","y-axis label [units]",lineLabels,"3b",4,11)
### 4b=filename
### 4 = position of legend, e.g. top right.
###
### For a single line, use:
### plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)), dataseries,"x-axis label [units]","y-axis label [units]", lineLabel,"3b",4,11)

### You can also plot a table directly into latex, see example_create_a_table(..)
###
### Then put it in latex with for example:
###\begin{table}[H]
###     \centering
###     \caption{Results some computation.}\label{tab:some_computation}
###     \begin{tabular}{|c|c|} % remember to update this to show all columns of table
###         \hline
###         \input{latex/project3/tables/q2.txt}
###     \end{tabular}
###\end{table}
import random
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np
import os


class Plot_to_tex:
    def __init__(self):
        self.script_dir = self.get_script_dir()
        print("Created main")

    # plot graph (legendPosition = integer 1 to 4)
    def plotSingleLine(
        self,
        x_path,
        y_series,
        x_axis_label,
        y_axis_label,
        label,
        filename,
        legendPosition,
        project_nr,
    ):
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(x_path, y_series, c="b", ls="-", label=label, fillstyle="none")
        plt.legend(loc=legendPosition)
        plt.xlabel(x_axis_label)
```

```
        plt.ylabel(y_axis_label)
        plt.savefig(
            os.path.dirname(__file__)
            + "/../../../latex/project"
            + str(project_nr)
            + "/Images/"
            + filename
            + ".png"
        )

    #         plt.show();

    # plot graphs
    def plotMultipleLines(
        self, x, y_series, x_label, y_label, label, filename,
            ↪ legendPosition, project_nr
    ):
        fig = plt.figure()
        ax = fig.add_subplot(111)

        # generate colours
        cmap = self.get_cmap(len(y_series[:, 0]))

        # generate line types
        lineTypes = self.generateLineTypes(y_series)

        for i in range(0, len(y_series)):
            # overwrite linetypes to single type
            lineTypes[i] = "-"
            ax.plot(
                x,
                y_series[i, :],
                ls=lineTypes[i],
                label=label[i],
                fillstyle="none",
                c=cmap(i),
            )
            # color

        # configure plot layout
        plt.legend(loc=legendPosition)
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        plt.savefig(
            os.path.dirname(__file__)
            + "/../../../latex/project"
            + str(project_nr)
            + "/Images/"
            + filename
            + ".png"
        )

        print(f"plotted lines")

    # Generate random line colours
    # Source: https://stackoverflow.com/questions/14720331/how-to-
        ↪ generate-random-colors-in-matplotlib
    def get_cmap(n, name="hsv"):
        """Returns a function that maps each index in 0, 1, ..., n-1
            ↪ to a distinct
        RGB color; the keyword argument name must be a standard mpl
            ↪ 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
        ↪ ) and pass it to this object
130     def put_table_in_tex(self, table_matrix, filename, project_nr):
131         cols = np.shape(table_matrix)[1]
132         format = "%s"
133         for col in range(1, cols):
134             format = format + " & %s"
135         format = format + ""
136         plt.savetxt(
137             os.path.dirname(__file__)
138             + "/../../../latex/project"
139             + str(project_nr)
140             + "/tables/"
141             + filename
142             + ".txt",
143             table_matrix,
144             delimiter=" & ",
145             fmt=format,
146             newline="  \\\\ \hline \n",
147         )
148
149     # replace this with your own table creation and then pass it to
        ↪ put_table_in_tex(..)
150     def example_create_a_table(self):
151         project_nr = "1"
152         table_name = "example_table_name"
153         rows = 2
154         columns = 4
155         table_matrix = np.zeros((rows, columns), dtype=object)
156         table_matrix[:, :] = ""  # replace the standard zeros with
            ↪ emtpy cell
157         print(table_matrix)
158         for column in range(0, columns):
159             for row in range(0, rows):
160                 table_matrix[row, column] = row + column
161         table_matrix[1, 0] = "example"
162         table_matrix[0, 1] = "grid sizes"
163
164         self.put_table_in_tex(table_matrix, table_name, project_nr)
165
166     def get_script_dir(self):
167         """returns the directory of this script regardles of from
            ↪ 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()
```