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 transparant 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 tingle with it yourself. Additionally, because the market analysis consists of a rough estimate, three different estimation methods are used for generating the TAM, SAM and SOM estimates. The redundancy is introduced to establish some referenceframe within the results.

The assumptions and datapoints for the respective models are specified in ??. Next, 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 Assumptions

2.1 Top Down

2.2 Bottom Up

2.3 Value Theory

3 Model Description

This

3.1 Market

To compute the TAM, SAM and SOM, some form of market definition can be used. To this end it can be valuable to specify exactly what the TruCol consultancy does, where it adds value and how it does that. Furthermore, since these three afore mentioned 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 companies can get the most out of the TruCol protocol. To understand this the following assumptions are shared:

- **asu-0:** Task completions of tasks that are completed using the TruCol protocol are deterministically verifyable.
- **asu-1:** Solutions of 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 costprice that is currently available in this world.
- **asu-3:** No personel 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.

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.

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 subsegment 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 succesfully hiring a consultancy for algorithmic optimalisation 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 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 overal 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 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 telecomunications 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.

3.2 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 based on requirement specification. We expect that users of the TruCol protocol will develop a tactical advantage on requirement specifications for AI engines.

3.3 Market Size

3.4 Market Trajectory

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. So 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.

3.4.1 Top Down

The Top-Down approach

3.4.2 Bottom Up

3.4.3 Value Theory

4 Results

4.1 Top Down

4.2 Top Down

4.3 Top Down

5 Sensitivity Analysis

5.1 Top Down

5.2 Bottom Up

5.3 Value Theory

6 Discussion

6.1 Top Down

6.2 Bottom Up

6.3 Value Theory

7 Conclusion

A Appendix __main__.py

```python
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
        )

        # 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()

        # 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

        # 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):

```

```python
        # 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

```
# 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
        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(
        appendices_of_all_types, False, main_appendix_inclusion_lines
          ↪ , project_nr,
    )
    return main_appendix_inclusion_lines


def append_latex_inclusion_command(
    appendices_of_all_types, is_from_root_dir,
      ↪ main_appendix_inclusion_lines, project_nr
):
    for appendix_type in appendices_of_all_types:
        for appendix in appendix_type:
            line = update_appendix_tex_code(
                appendix.appendix_filename, is_from_root_dir,
                  ↪ project_nr
            )
            print(f"appendix.appendix_filename={appendix.
              ↪ appendix_filename}")
            main_appendix_inclusion_lines.append(line)
    return main_appendix_inclusion_lines


def filter_appendices_by_type(appendices, appendix_type):
    """Returns the list of all appendices of a certain appendix type,
      ↪  from the incoming list of Appendix objects.

    :param appendices: List of Appendix objects
    :param appendix_type: Can consist of "no_code", "python", or "
      ↪ notebook" and indicates different appendix types
    """
    return_appendices = []
    for appendix in appendices:
        if appendix.appendix_type == appendix_type:
            return_appendices.append(appendix)
    return return_appendices


def sort_python_appendices(appendices):
    """First puts __main__.py, followed by main.py followed by a-z
      ↪ code files.

    :param appendices: List of Appendix objects
    """
    return_appendices = []
    for appendix in appendices:  # first get appendix containing
      ↪ __main__.py
        if (appendix.code_filename == "__main__.py") or (
            appendix.code_filename == "__Main__.py"
        ):
            return_appendices.append(appendix)
            appendices.remove(appendix)
    for appendix in appendices:  # second get appendix containing
      ↪ main.py
        if (appendix.code_filename == "main.py") or (
            appendix.code_filename == "Main.py"
        ):
            return_appendices.append(appendix)
            appendices.remove(appendix)
    return_appendices

    # Filter remaining appendices in order of a-z
    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):
```

```python
                appendix = get_appendix_from_filename(appendices, appendix_filename)
                if appendix.appendix_type == "no_code":
                    non_code_appendices.append(appendix)
                    non_code_appendix_lines.append(line)
    return non_code_appendices, non_code_appendix_lines


def get_filename_from_latex_appendix_line(appendices, appendix_line):
    """Returns the first filename from a list of incoming filenames that
    occurs in a latex code line.

    :param appendices: List of Appendix objects
    :param appendix_line: latex code (in particular expected to be the code from main that is used to include appendix latex files.)
    """
    for filename in list(map(lambda appendix: appendix.appendix_filename, appendices)):
        if filename in appendix_line:
            if not line_is_commented(appendix_line, filename):
                return filename


def get_appendix_from_filename(appendices, appendix_filename):
    """Returns the first Appendix object with an appendix filename that matches the incoming appendix_filename.
    The Appendix objects are selected from an incoming list of Appendix objects.

    :param appendices: List of Appendix objects
    :param appendix_filename: name of a latex appendix file, ends in .tex,
    """
    for appendix in appendices:
        if appendix_filename == appendix.appendix_filename:
            return appendix


def get_compiled_notebook_paths(script_dir):
    """Returns the list of jupiter notebook filepaths that were compiled successfully and that are
    included in the same dias this script (the src directory).

    :param script_dir: absolute path of this file.
    """
    notebook_filepaths = get_filenames_in_dir(".ipynb", script_dir)
    compiled_notebook_filepaths = []

    # check if the jupyter notebooks were compiled
    for notebook_filepath in notebook_filepaths:

        # swap file extension
        notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf")

        # check if file exists
        if os.path.isfile(notebook_filepath):
            compiled_notebook_filepaths.append(notebook_filepath)
    return compiled_notebook_filepaths

```

```python

def get_list_of_appendix_files(
    appendix_dir, absolute_notebook_filepaths, absolute_python_filepaths
):
    """Returns a list of Appendix objects that contain all the appendix files with .tex extension.

    :param appendix_dir: Absolute path that contains the appendix .tex files.
    :param absolute_notebook_filepaths: List of absolute paths to the  compiled notebook pdf files.
    :param absolute_python_filepaths: List of absolute paths to the python files.
    """
    appendices = []
    appendices_paths = get_filenames_in_dir(".tex", appendix_dir)

    for appendix_filepath in appendices_paths:
        appendix_type = "no_code"
        appendix_filecontent = read_file(appendix_filepath)
        line_nr_python_file_inclusion = get_line_of_latex_command(
            appendix_filecontent, "\pythonexternal{"
        )
        line_nr_notebook_file_inclusion = get_line_of_latex_command(
            appendix_filecontent, "\includepdf[pages="
        )
        if line_nr_python_file_inclusion > -1:
            appendix_type = "python"
            # get python filename
            line = appendix_filecontent[line_nr_python_file_inclusion]
            filename = get_filename_from_latex_inclusion_command(
                line, ".py", "\pythonexternal{"
            )
            appendices.append(
                Appendix(
                    appendix_filepath,
                    appendix_filecontent,
                    appendix_type,
                    filename,
                    line,
                )
            )
        if line_nr_notebook_file_inclusion > -1:
            appendix_type = "notebook"
            line = appendix_filecontent[line_nr_notebook_file_inclusion]
            filename = get_filename_from_latex_inclusion_command(
                line, ".pdf", "\includepdf[pages="
            )
            appendices.append(
                Appendix(
                    appendix_filepath,
                    appendix_filecontent,
                    appendix_type,
                    filename,
                    line,
                )
            )
        else:
            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).
```

```python
    :param appendix_dir: Absolute path that contains the appendix .tex files.
    :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_files = get_filenames_in_dir(".tex", appendix_dir)
    contained_codes = []
    for code_filepath in absolute_code_filepaths:
        for appendix_filepath in appendix_files:
            appendix_filecontent = read_file(appendix_filepath)
            line_nr = check_if_appendix_contains_file(
                appendix_filecontent, code_filepath, extension, project_nr, root_dir
            )
            if line_nr > -1:
                # add filepath to list of files that are already in the appendices
                contained_codes.append(
                    Appendix_with_code(
                        code_filepath,
                        appendix_filepath,
                        appendix_filecontent,
                        line_nr,
                        ".py",
                    )
                )
    return contained_codes


def check_if_appendix_contains_file(
    appendix_content, code_filepath, extension, project_nr, root_dir
):
    """Scans an appendix content to determine whether it contains a substring that
    includes a code file (of either python or compiled notebook=pdf extension).

    :param appendix_content: content in an appendix latex file.
    :param code_filepath: Absolute path to a code file (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.
    """
    # convert code_filepath to the inclusion format in latex format
    latex_relative_filepath = (
        f"latex/project{project_nr}/../../{code_filepath[len(root_dir):]}"
    )
    latex_command = get_latex_inclusion_command(extension, latex_relative_filepath)
    return get_line_of_latex_command(appendix_content, latex_command)


def get_line_of_latex_command(appendix_content, latex_command):
    """Returns the line number of a latex command if it is found. Returns -1 otherwise.

    :param appendix_content: content in an appendix latex file.
    :param latex_command: A line of latex code. (Expected to come from some appendix)
    """
    # check if the file is in the latex code
    line_nr = 0
    for line in appendix_content:
        if latex_command in line:
            if line_is_commented(line, latex_command):
                commented = True
            else:
                return line_nr
        line_nr = line_nr + 1
    return -1


def line_is_commented(line, target_substring):
    """Returns True if a latex code line is commented, returns False otherwise

    :param line: A line of latex code that contains a relevant command (target substring).
    :param target_substring: Used to determine whether the command that is found is commented or not.
    """
    left_of_command = line[: line.rfind(target_substring)]
    if "%" in left_of_command:
        return True
    return False


def get_latex_inclusion_command(extension, latex_relative_filepath_to_codefile):
    """Creates and returns a latex command that includes either a python file or a compiled jupiter
    notebook pdf (whereeever the command is placed). The command is intended to be placed in the appendix.

    :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.
    """
    if extension == ".py":
        left = "\pythonexternal{"
        right = "}"
        latex_command = f"{left}{latex_relative_filepath_to_codefile}{right}"
    elif extension == ".ipynb":

        left = "\includepdf[pages=-]{"
        right = "}"
        latex_command = f"{left}{latex_relative_filepath_to_codefile}{right}"
    return latex_command


def read_file(filepath):
    """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
    )

```

```python
    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):]}}}{{"
    )
    # append current line
    content.append(get_latex_inclusion_command(extension,
       ↪ latex_relative_filepath))
    content.append(f"}}{{")
    content.append(
        get_latex_inclusion_command(extension,
           ↪ code_path_from_latex_main_path)
    )
    content.append(f"}}")
    return content


def get_index_of_auto_generated_appendices(appendix_dir, extension):
    """Returns the maximum index of auto generated appendices of
    a specific extension type.

    :param extension: The file extension of the file that is sought
       ↪ in the appendix line. Either ".py" or ".pdf".
    :param appendix_dir: Absolute path that contains the appendix .
       ↪ tex files.
    """
    max_index = -1
    appendices =
       ↪ get_auto_generated_appendix_filenames_of_specific_extension
       ↪ (
        appendix_dir, extension
    )
    for appendix in appendices:
        substring = f"Auto_generated_{extension[1:]}_App"
        # remove left of index
        remainder = appendix[appendix.rfind(substring) + len(
           ↪ substring) :]
        # remove right of index
        index = int(remainder[:-4])
        if index > max_index:
            max_index = index
    return max_index


def get_auto_generated_appendix_filenames_of_specific_extension(
    appendix_dir, extension
):
    """Returns the list of auto generated appendices of
    a specific extension type.

    :param extension: The file extension of the file that is sought
       ↪ in the appendix line. Either ".py" or ".pdf".
    :param appendix_dir: Absolute path that contains the appendix .
       ↪ tex files.
    """
    appendices_of_extension_type = []

    # get all appendices
    appendix_files = get_filenames_in_dir(".tex", appendix_dir)

    # get appendices of particular extention type
    for appendix_filepath in appendix_files:
        right_of_slash = appendix_filepath[appendix_filepath.rfind("/
           ↪ ") + 1 :]
        if (
            right_of_slash[: 15 + len(extension) - 1]
            == f"Auto_generated_{extension[1:]}"
```

```python
        ):
            appendices_of_extension_type.append(appendix_filepath)
    return appendices_of_extension_type


def create_section(appendix_reference_index, code_filename, content):
    """Creates the header of a latex appendix file, such that it contains a section that
    indicates the section is an appendix, and indicates which pyhon or notebook file is
    being included in that appendix.

    :param appendix_reference_index: A counter that is used in the label to ensure the appendix section labels are unique.
    :param code_filename: file name of the code file that is included
    :param content: A list of strings that make up the appendix, with one line per element.
    """
    # write section
    left = "\section{Appendix "
    middle = code_filename.replace("_", "\_")
    right = "}\label{app:"
    end = "}"  # TODO: update appendix reference index
    content.append(f"{left}{middle}{right}{appendix_reference_index}{end}")
    return content


def overwrite_content_to_file(content, filepath, content_has_newlines=True):
    """Writes a list of lines of tex code from the content argument to a .tex file
    using overwriting method. The content has one line per element.

    :param content: The content that is being written to file.
    :param filepath: Path towards the file that is being read.
    :param content_has_newlines:  (Default value = True)
    """
    with open(filepath, "w") as f:
        for line in content:
            if content_has_newlines:
                f.write(line)
            else:
                f.write(line + "\n")


def get_appendix_tex_code(main_latex_filename):
    """gets the latex appendix code from the main tex file.

    :param main_latex_filename: Name of the main latex document of this project number
    """
    main_tex_code = read_file(main_latex_filename)
    start = "\\begin{appendices}"
    end = "\end{appendices}"
    start_index = get_index_of_substring_in_list(main_tex_code, start) + 1
    end_index = get_index_of_substring_in_list(main_tex_code, end)
    return main_tex_code, start_index, end_index, main_tex_code[start_index:end_index]


```

```python
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


```

```python
def get_filename_from_dir(path):
    """Returns a filename from an absolute path to a file.

    :param path: path to a file of which the name is queried.
    """
    return path[path.rfind("/") + 1 :]


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


class Appendix_with_code:
    """stores in which appendix file and accompanying line number in the appendix in which a code file is
    already included. Does not take into account whether this appendix is in the main tex file or not
    """

    def __init__(
        self,
        code_filepath,
        appendix_filepath,
        appendix_content,
        file_line_nr,
        extension,
    ):
        self.code_filepath = code_filepath
        self.appendix_filepath = appendix_filepath
        self.appendix_content = appendix_content
        self.file_line_nr = file_line_nr
        self.extension = extension


class Appendix:
    """stores in appendix files and type of appendix."""

    def __init__(
        self,
        appendix_filepath,
        appendix_content,
        appendix_type,
        code_filename=None,
        appendix_inclusion_line=None,
    ):
        self.appendix_filepath = appendix_filepath
        self.appendix_filename = get_filename_from_dir(self.appendix_filepath)
        self.appendix_content = appendix_content
        self.appendix_type = appendix_type  # TODO: perform validation of input values
        self.code_filename = code_filename
        self.appendix_inclusion_line = appendix_inclusion_line
```

F Appendix Model_bottom_up.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 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.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.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,
        )
        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,
        )
        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,
        )
        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_pharmaceutics_revenue(self):
        return 0

    def estimate_algo_trading_revenue(self):
        return 0

    def estimate_material_sciences_revenue(self):
        return 0

    def estimate_telecommunications_revenue(self):
        return 0

    def plot_data(self, x, y):
        N = self.nr_simulations

        # Random colour for points, vector of length N
        colors = np.ones(N)

        # 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")

        plt.scatter(x, y, c=colors, alpha=0.8)
        plt.xlabel("Summed Squared Average Randomness")
        plt.ylabel("Estimated revenue in $million/year")
        plt.title("Monte-carlo simulation\n estimated total revenue
           ↪ TruCol consultancy")

        # 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 consultancy per sector"
        )

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

    def get_colors(self, x_series, y_series):

        # Create list to store colors
        color_arr = []

        # Hardcode the colours for the dataseries
        colors = ["yellow", "green", "cyan", "blue", "magenta"]

        # Flatten the lists per sector into a single list
        x = [item for sublist in x_series for item in sublist]

        # Give each datapoint of a sector the same colour
        for i in range(0, len(x_series)):
            for elem in range(0, len(x_series[i])):
                color_arr.append(colors[i])
        return color_arr, colors

    def get_normal_dist(self):
        # Creating a series of data of in range of 1-50.
        x = np.linspace(1, 50, 200)

        # Calculate mean and Standard deviation.
        mean = np.mean(x)
        sd = np.std(x)

        # Apply function to the data.
        pdf = self.normal_dist(x, mean, sd)

        # Plotting the Results
        plt.plot(x, pdf, color="red")
        plt.xlabel("Data points")
        plt.ylabel("Probability Density")

    def normal_dist(self, x, mean, sd):
        prob_density = (np.pi * sd) * np.exp(-0.5 * ((x - mean) / sd) ** 2)
        return prob_density

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

H Appendix Model_value_theory.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 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)
```

```python
        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."""
        return plt.cm.get_cmap(name, n)

    def generateLineTypes(y_series):
        # generate varying linetypes
        typeOfLines = list(lines.lineStyles.keys())

        while len(y_series) > len(typeOfLines):
            typeOfLines.append("-.")

        # remove void lines
        for i in range(0, len(y_series)):
            if typeOfLines[i] == "None":
                typeOfLines[i] = "-"
            if typeOfLines[i] == "":
                typeOfLines[i] = ":"
            if typeOfLines[i] == " ":
                typeOfLines[i] = "--"
        return typeOfLines

    # Create a table with: table_matrix = np.zeros((4,4),dtype=object) and pass it to this object
    def put_table_in_tex(self, table_matrix, filename, project_nr):
        cols = np.shape(table_matrix)[1]
        format = "%s"
        for col in range(1, cols):
            format = format + " & %s"
        format = format + ""
        plt.savetxt(
            os.path.dirname(__file__)
            + "/../../../latex/project"
            + str(project_nr)
            + "/tables/"
            + filename
            + ".txt",
            table_matrix,
            delimiter=" & ",
            fmt=format,
            newline="  \\\\ \hline \n",
        )

    # replace this with your own table creation and then pass it to put_table_in_tex(..)
    def example_create_a_table(self):
        project_nr = "1"
        table_name = "example_table_name"
        rows = 2
        columns = 4
        table_matrix = np.zeros((rows, columns), dtype=object)
        table_matrix[:, :] = ""  # replace the standard zeros with emtpy cell
        print(table_matrix)
        for column in range(0, columns):
            for row in range(0, rows):
                table_matrix[row, column] = row + column
        table_matrix[1, 0] = "example"
        table_matrix[0, 1] = "grid sizes"

        self.put_table_in_tex(table_matrix, table_name, project_nr)

    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 = Plot_to_tex()
    main.example_create_a_table()
```