Example to plot directly into latex

19-10-2019

1 Introduction

Welcome, this document presents the roadmap for the TruCol consultancy as well as the estimated costs of the accompanying human labour. The objective of this document is to provide some basic insight into the order of magnitude of the costs of developing a sustainable variant of the TruCol consultancy such that it is able to generate returns for its potential investors.

2 Assumptions

3 Model Description

4 Results

5 Sensitivity Analysis

6 Discussion

7 Conclusion

A Appendix __main__.py

```python
import os
from .Main import Main
from .Compile_gantt_locally import compile_gantt_locally

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()

# create gantt chart
main.create_gantt()

# compile the gantt chart locally
compile_gantt_locally(
    main.relative_src_filepath, main.plant_uml_java_filename, main.src_to_gantt_path
)

# 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 .Gantt import Gantt
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):
        self.project_nr = 1
        self.plant_uml_java_filename = "plantuml.jar"
        self.src_to_gantt_path = "Diagrams/created.uml"
        # self.src_to_gantt_path="Diagrams/example_gantt.uml"
        self.relative_src_filepath = f"code/project{self.project_nr}/
            ↪ src/"
        self.relative_plant_uml_java_filepath = (
            f"code/project{self.project_nr}/src/{self.
                ↪ plant_uml_java_filename}"
        )

    def create_gantt(self):
        gantt = Gantt(f"{self.relative_src_filepath}/Diagrams/created
            ↪ .uml")

    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 Activity.py

```python
# The bottom up model that computes the TAM and TSM
import random
import numpy as np
import copy

from .Plot_to_tex import Plot_to_tex as plt_tex


class Activity:
    def __init__(
        self,
        description,
        duration,
        new_tag,
        colour=None,
        parent=None,
        starts_at_child_nr_start=None,
        starts_at_child_nr_end=None,
        font_size=None,
        hourly_wage=None,
        hours_per_day=None
    ):
        self.parent = parent
        self.description = description
        self.duration = duration
        self.children = []
        self.font_size=None
        self.starts_at_child_nr_start=starts_at_child_nr_start
        self.starts_at_child_nr_end=starts_at_child_nr_end
        self.hourly_wage = hourly_wage
        if hours_per_day is None:
            self.hours_per_day=8
        else:
            self.hours_per_day=hours_per_day
        if self.parent is None:
            if colour is None:
                raise Exception("Parent activity needs a colour")
            self.colour = colour
            self.tag = []
        else:
            if (not self.parent.hourly_wage is None) and (hourly_wage
                ↪  is None):
                self.hourly_wage=self.parent.hourly_wage
            self.colour = parent.colour
            self.tag = copy.deepcopy(self.parent.tag)

        # create tag
        self.tag.append(new_tag)

    def add_children(self, children):
        self.children = children

    def get_tag(self):
        return "_".join(list(map(lambda x: str(x), self.tag)))

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

D Appendix Compile_gantt_locally.py

```
# pip install plantuml

# to compile locally:
# export  PLANTUML_LIMIT_SIZE=8192
# java -jar plantuml.jar -verbose sequenceDiagram.txt

# To compile with server:
from plantuml import PlantUML
import os
import subprocess
from os.path import abspath
from shutil import copyfile
import requests


def compile_gantt_locally(
    relative_src_filepath, plant_uml_java_filename, src_to_gantt_path
):
    relative_plant_uml_java_filepath = (
        f"{relative_src_filepath}{plant_uml_java_filename}"
    )
    print(f"relative_plant_uml_java_filepath={
       ↪ relative_plant_uml_java_filepath}")
    if got_java_file(relative_plant_uml_java_filepath):
        print("FOUND")
        os.environ["PLANTUML_LIMIT_SIZE"] = "8192"
        run_bash_command(
            f"java -jar {relative_plant_uml_java_filepath} -verbose {
               ↪ relative_src_filepath}{src_to_gantt_path}"
        )
        # export_gantt_to_latex(filename)
    pass


def check_if_java_file_exists(relative_filepath):
    if os.path.isfile(relative_filepath):
        return True
    else:
        return False

def got_java_file(relative_filepath):
    # Check if the jar file exists, curl it if not.
    if not check_if_java_file_exists(relative_filepath):
        # The java file is not found, curl it
        request_file("https://sourceforge.net/projects/plantuml/files
           ↪ /plantuml.jar/download", relative_filepath)
    # Check if the jar file exists after curling it. Raise Exception
       ↪ if it is not found after curling.
    if not check_if_java_file_exists(relative_filepath):
        raise Exception(f"File:{relative_filepath} is not accessible"
           ↪ )
    print(f'Got the Java file')
    return True

def request_file(url, output_filepath):
    import requests
    # Request the file in the url
    response = requests.get(url)
    with open(output_filepath, "wb") as f:
        f.write(response.content)
```

```
56
57 def run_bash_command(bashCommand):
58     subprocess.Popen(bashCommand, shell=True)
59
60
61 def export_gantt_to_latex(filename):
62     if filename.endswith(".png"):
63         copyfile(
64             abspath(f"./Diagrams/{filename}"),
65             abspath(f"./../../latex/researchplan/Images/{filename}"),
66         )
```

E 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()
```

F Appendix Create_python_gantt.py

```python
import numpy as np

from .Activity import Activity


def create_python_gantt():
    parents = []
    # parent one
    parent_one = Activity("description", duration=5, new_tag=0, colour="Green")
    # children
    child_one = Activity(
        "child one description", duration=2, new_tag=0, parent=parent_one
    )
    child_two = Activity(
        "child two description", duration=2, new_tag=1, parent=parent_one
    )
    # merge family
    parent_one.add_children([child_one, child_two])
    #parents.append(parent_one)

    parent_two = Activity("description two", duration=5, new_tag=1, colour="DarkOrchid")
    # children
    child_one = Activity(
        "child one description two", duration=2, new_tag=0, parent=parent_two
    )
    child_two = Activity(
        "child two description two", duration=2, new_tag=1, parent=parent_two
    )
    # merge family
    parent_two.add_children([child_one, child_two])
    #parents.append(parent_two)

    ## parent
    # parent one
    protocol = Activity(description="Develop protocol", duration=120,  new_tag=0, colour="Green",hourly_wage=0)
    # children
    onchain = Activity(description="On-chain: Solidty+VRF", duration=60, new_tag=0, parent=protocol,hourly_wage=75)
    git_tellor = Activity(description="Git integration: Tellor", duration=90,  new_tag=1,parent=protocol, starts_at_child_nr_start=0,hourly_wage=75)
    git_chainlink = Activity(description="Git integration: Chainlink", duration=90,  new_tag=2,parent=protocol, starts_at_child_nr_start=0,hourly_wage=75)
    alt_chains = Activity(description="Alternative Chains", duration=90,  new_tag=3,parent=protocol,starts_at_child_nr_start=0, hourly_wage=75)

    # grandchildren
    ci = Activity(description="(Decentralised) Continuous integration ",  new_tag=0, duration=30, parent=git_chainlink)
    #git_chainlink.add_children([ci])
```

```
45      security = Activity(description="Security & Robustness", duration
           ↪ =60,  new_tag=1, parent=git_chainlink)
46      #ci.add_children([security])
47
48      # merge
49      protocol.add_children([onchain,git_tellor,git_chainlink,
           ↪ alt_chains, ci, security])
50      parents.append(protocol)
51
52
53      # parent_two
54      platform_eco = Activity(description="Platform & ecosystem",
           ↪ duration=120, new_tag=1, colour="DarkOrchid",
           ↪ starts_at_child_nr_start=0,hourly_wage=40)
55      # children
56      website = Activity(description="Website", duration=50, new_tag=0,
           ↪  parent=platform_eco)
57      marketing_platf = Activity(description="Marketing platform",
           ↪ duration=30, new_tag=1, parent=platform_eco)
58      bounties = Activity(description="Subsidize bounties", duration
           ↪ =10, new_tag=2, parent=platform_eco)
59      platform_buffer = Activity(description="Platform Planning Buffer"
           ↪ , duration=30, new_tag=3, parent=platform_eco)
60
61      # Grandchildren
62      api = Activity(description="API", duration=50, new_tag=0, parent=
           ↪ website)
63      gui = Activity(description="GUI", duration=50, new_tag=1, parent=
           ↪ website,starts_at_child_nr_start=0)
64      forum = Activity(description="Forum", duration=10, new_tag=2,
           ↪ parent=website, starts_at_child_nr_start=0)
65      website.add_children([api, gui, forum])
66
67      platform_eco.add_children([website,marketing_platf,bounties,
           ↪ platform_buffer])
68      parents.append(platform_eco)
69
70      # parent_three
71      company = Activity(description="Launch company", duration=150,
           ↪ new_tag=2, colour="Yellow",starts_at_child_nr_start=0,
           ↪ hourly_wage=35)
72      # children
73      partners = Activity(description="Qualitative partner research",
           ↪ duration=20, new_tag=0, parent=company)
74      organisation = Activity(description="Establish organisation",
           ↪ duration=80, new_tag=1, parent=company)
75      marketing_company = Activity(description="Marketing", duration
           ↪ =30, new_tag=2, parent=company)
76      company_buffer = Activity(description="Organisation Planning
           ↪ Buffer ", duration=20, new_tag=3, parent=company)
77
78      # Grandchildren
79      auditing = Activity(description="Auditing", duration=10, new_tag
           ↪ =0, parent=organisation)
80      hiring = Activity(description="Hiring", duration=20, new_tag=1,
           ↪ parent=organisation)
81      administration = Activity(description="Administration", duration
           ↪ =10, new_tag=2, parent=organisation)
82      legal = Activity(description="Legal", duration=20, new_tag=3,
           ↪ parent=organisation)
83      financial = Activity(description="Financial", duration=20,
           ↪ new_tag=4, parent=organisation)
```

```
84      organisation.add_children([auditing, hiring, administration,
           ↪ legal, financial])
85
86
87      company.add_children([partners,organisation, marketing_company,
           ↪ company_buffer])
88      parents.append(company)
89
90      return parents
91
92
93  def addTwo(self, x):
94      """adds two to the incoming integer and returns the result of the
           ↪  computation."""
95      return x + 2
```

G 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(
```

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

```
    appendices_of_all_types = [python_appendices, notebook_appendices ]

    print(f"\n\n")
    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,
    )
    # main_appendix_inclusion_lines.append(f"}}")
    print(f"main_appendix_inclusion_lines={ main_appendix_inclusion_lines}")
    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):
```

```
257     """Scans the lines of appendices in the main code, and returns
            ↪ the lines
258     of the appendices that do not contain code, in the order in which
            ↪  they were
259     included in the main latex file.
260
261     :param appendices: List of Appendix objects
262     :param appendix_tex_code: latex code from the main latex file
            ↪ that includes the appendices
263     """
264     non_code_appendices = []
265     non_code_appendix_lines = []
266     appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
267     for line in appendix_tex_code:
268         appendix_filename = get_filename_from_latex_appendix_line(
                ↪ appendices, line)
269
270         # Check if line is not commented
271         if not appendix_filename is None:
272             if not line_is_commented(line, appendix_filename):
273                 appendix = get_appendix_from_filename(appendices,
                        ↪ appendix_filename)
274                 if appendix.appendix_type == "no_code":
275                     non_code_appendices.append(appendix)
276                     non_code_appendix_lines.append(line)
277     return non_code_appendices, non_code_appendix_lines
278
279
280 def get_filename_from_latex_appendix_line(appendices, appendix_line):
281     """Returns the first filename from a list of incoming filenames
            ↪ that
282     occurs in a latex code line.
283
284     :param appendices: List of Appendix objects
285     :param appendix_line: latex code (in particular expected to be
            ↪ the code from main that is used to include appendix latex
            ↪ files.)
286     """
287     for filename in list(map(lambda appendix: appendix.
            ↪ appendix_filename, appendices)):
288         if filename in appendix_line:
289             if not line_is_commented(appendix_line, filename):
290                 return filename
291
292
293 def get_appendix_from_filename(appendices, appendix_filename):
294     """Returns the first Appendix object with an appendix filename
            ↪ that matches the incoming appendix_filename.
295     The Appendix objects are selected from an incoming list of
            ↪ Appendix objects.
296
297     :param appendices: List of Appendix objects
298     :param appendix_filename: name of a latex appendix file, ends in
            ↪ .tex,
299     """
300     for appendix in appendices:
301         if appendix_filename == appendix.appendix_filename:
302             return appendix
303
304
305 def get_compiled_notebook_paths(script_dir):
```

```
306     """Returns the list of jupiter notebook filepaths that were
            ↪ compiled successfully and that are
307     included in the same dias this script (the src directory).
308
309     :param script_dir: absolute path of this file.
310     """
311     notebook_filepaths = get_filenames_in_dir(".ipynb", script_dir)
312     compiled_notebook_filepaths = []
313
314     # check if the jupyter notebooks were compiled
315     for notebook_filepath in notebook_filepaths:
316
317         # swap file extension
318         notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf
                ↪ ")
319
320         # check if file exists
321         if os.path.isfile(notebook_filepath):
322             compiled_notebook_filepaths.append(notebook_filepath)
323     return compiled_notebook_filepaths
324
325
326 def get_list_of_appendix_files(
327     appendix_dir, absolute_notebook_filepaths,
            ↪ absolute_python_filepaths
328 ):
329     """Returns a list of Appendix objects that contain all the
            ↪ appendix files with .tex extension.
330
331     :param appendix_dir: Absolute path that contains the appendix .
            ↪ tex files.
332     :param absolute_notebook_filepaths: List of absolute paths to the
            ↪  compiled notebook pdf files.
333     :param absolute_python_filepaths: List of absolute paths to the
            ↪ python files.
334     """
335     appendices = []
336     appendices_paths = get_filenames_in_dir(".tex", appendix_dir)
337
338     for appendix_filepath in appendices_paths:
339         appendix_type = "no_code"
340         appendix_filecontent = read_file(appendix_filepath)
341         line_nr_python_file_inclusion = get_line_of_latex_command(
342             appendix_filecontent, "\pythonexternal{"
343         )
344         line_nr_notebook_file_inclusion = get_line_of_latex_command(
345             appendix_filecontent, "\includepdf[pages="
346         )
347         if line_nr_python_file_inclusion > -1:
348             appendix_type = "python"
349             # get python filename
350             line = appendix_filecontent[line_nr_python_file_inclusion
                    ↪ ]
351             filename = get_filename_from_latex_inclusion_command(
352                 line, ".py", "\pythonexternal{"
353             )
354             appendices.append(
355                 Appendix(
356                     appendix_filepath,
357                     appendix_filecontent,
358                     appendix_type,
359                     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):
```

```
412         for file in f:
413             if file.endswith(extension):
414                 if (excluded_files is None) or (
415                     (not excluded_files is None) and (not file in
                        ↪ excluded_files)
416                 ):
417                     filepaths.append(r + "/" + file)
418     return filepaths
419
420
421 def get_code_files_already_included_in_appendices(
422     absolute_code_filepaths, appendix_dir, extension, project_nr,
        ↪ root_dir
423 ):
424     """Returns a list of code filepaths that are already properly
        ↪ included the latex appendix files of this project.
425
426     :param absolute_code_filepaths: List of absolute paths to the
        ↪ code files (either python files or compiled jupyter
        ↪ notebook pdfs).
427     :param appendix_dir: Absolute path that contains the appendix .
        ↪ tex files.
428     :param extension: The file extension of the file that is sought
        ↪ in the appendix line. Either ".py" or ".pdf".
429     :param project_nr: The number  indicating which project this code
        ↪  pertains to.
430     :param root_dir: The root directory of this repository.
431     """
432     appendix_files = get_filenames_in_dir(".tex", appendix_dir)
433     contained_codes = []
434     for code_filepath in absolute_code_filepaths:
435         for appendix_filepath in appendix_files:
436             appendix_filecontent = read_file(appendix_filepath)
437             line_nr = check_if_appendix_contains_file(
438                 appendix_filecontent, code_filepath, extension,
                    ↪ project_nr, root_dir
439             )
440             if line_nr > -1:
441                 # add filepath to list of files that are already in
                    ↪ the appendices
442                 contained_codes.append(
443                     Appendix_with_code(
444                         code_filepath,
445                         appendix_filepath,
446                         appendix_filecontent,
447                         line_nr,
448                         ".py",
449                     )
450                 )
451     return contained_codes
452
453
454 def check_if_appendix_contains_file(
455     appendix_content, code_filepath, extension, project_nr, root_dir
456 ):
457     """Scans an appendix content to determine whether it contains a
        ↪ substring that
458     includes a code file (of either python or compiled notebook=pdf
        ↪ extension).
459
460     :param appendix_content: content in an appendix latex file.
```

```
461     :param code_filepath: Absolute path to a code file (either python
            ↪  files or compiled jupyter notebook pdfs).
462     :param extension: The file extension of the file that is sought
            ↪ in the appendix line. Either ".py" or ".pdf".
463     :param project_nr: The number  indicating which project this code
            ↪  pertains to.
464     :param root_dir: The root directory of this repository.
465     """
466     # convert code_filepath to the inclusion format in latex format
467     latex_relative_filepath = (
468         f"latex/project{project_nr}/../../{code_filepath[len(root_dir
                ↪ ):]}"
469     )
470     latex_command = get_latex_inclusion_command(extension,
            ↪ latex_relative_filepath)
471     return get_line_of_latex_command(appendix_content, latex_command)
472 
473 
474 def get_line_of_latex_command(appendix_content, latex_command):
475     """Returns the line number of a latex command if it is found.
            ↪ Returns -1 otherwise.
476 
477     :param appendix_content: content in an appendix latex file.
478     :param latex_command: A line of latex code. (Expected to come
            ↪ from some appendix)
479     """
480     # check if the file is in the latex code
481     line_nr = 0
482     for line in appendix_content:
483         if latex_command in line:
484             if line_is_commented(line, latex_command):
485                 commented = True
486             else:
487                 return line_nr
488         line_nr = line_nr + 1
489     return -1
490 
491 
492 def line_is_commented(line, target_substring):
493     """Returns True if a latex code line is commented, returns False
            ↪ otherwise
494 
495     :param line: A line of latex code that contains a relevant
            ↪ command (target substring).
496     :param target_substring: Used to determine whether the command
            ↪ that is found is commented or not.
497     """
498     left_of_command = line[: line.rfind(target_substring)]
499     if "%" in left_of_command:
500         return True
501     return False
502 
503 
504 def get_latex_inclusion_command(extension,
        ↪ latex_relative_filepath_to_codefile):
505     """Creates and returns a latex command that includes either a
            ↪ python file or a compiled jupiter
506     notebook pdf (whereever the command is placed). The command is
            ↪ intended to be placed in the appendix.
507 
508     :param extension: The file extension of the file that is sought
            ↪ in the appendix line. Either ".py" or ".pdf".
```

```
509     :param latex_relative_filepath_to_codefile: The latex compilation
            ↪  requires a relative path towards code files
510     that are included. Therefore, a relative path towards the code is
            ↪  given.
511     """
512     if extension == ".py":
513         left = "\pythonexternal{"
514         right = "}"
515         latex_command = f"{left}{latex_relative_filepath_to_codefile
                ↪ }{right}"
516     elif extension == ".ipynb":
517 
518         left = "\includepdf[pages=-]{"
519         right = "}"
520         latex_command = f"{left}{latex_relative_filepath_to_codefile
                ↪ }{right}"
521     return latex_command
522 
523 
524 def read_file(filepath):
525     """Reads content of a file and returns it as a list of strings,
            ↪ with one string per line.
526 
527     :param filepath: path towards the file that is being read.
528     """
529     with open(filepath) as f:
530         content = f.readlines()
531     return content
532 
533 
534 def get_code_files_not_yet_included_in_appendices(
535     code_filepaths, contained_codes, extension
536 ):
537     """Returns a list of filepaths that are not yet properly included
            ↪  in some appendix of this project.
538 
539     :param code_filepath: Absolute path to all the code files in
            ↪ this project (source directory).
540     (either python files or compiled jupyter notebook pdfs).
541     :param contained_codes: list of  Appendix objects that include
            ↪ either python files or compiled jupyter notebook pdfs,
            ↪ which
542     are already included in the appendix tex files. (Does not care
            ↪ whether those appendices are also actually
543     included in the main or not.)
544     :param extension: The file extension of the file that is sought
            ↪ in the appendix line. Either ".py" or ".pdf".
545     """
546     contained_filepaths = list(
547         map(lambda contained_file: contained_file.code_filepath,
                ↪ contained_codes)
548     )
549     not_contained = []
550     for filepath in code_filepaths:
551         if not filepath in contained_filepaths:
552             not_contained.append(filepath)
553     return not_contained
554 
555 
556 def create_appendices_with_code(
557     appendix_dir, code_filepaths, extension, project_nr, root_dir
558 ):
```

```
    """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,
        )

        print(f"content={content}")

        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.
    """
    print(f"before={content}")
    # TODO: append if exists}
    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))
    # TODO: append {}
    content.append(f"}}{{")
    # TODO: code_path_from latex line
    content.append(
        get_latex_inclusion_command(extension, code_path_from_latex_main_path)
    )
    # TODO: add closing bracket }
    content.append(f"}}")
    print(f"after={content}")
    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])
```

```
659             if index > max_index:
660                 max_index = index
661         return max_index
662
663
664     def get_auto_generated_appendix_filenames_of_specific_extension(
665         appendix_dir, extension
666     ):
667         """Returns the list of auto generated appendices of
668         a specific extension type.
669
670         :param extension: The file extension of the file that is sought
                in the appendix line. Either ".py" or ".pdf".
671         :param appendix_dir: Absolute path that contains the appendix .
                tex files.
672         """
673         appendices_of_extension_type = []
674
675         # get all appendices
676         appendix_files = get_filenames_in_dir(".tex", appendix_dir)
677
678         # get appendices of particular extention type
679         for appendix_filepath in appendix_files:
680             right_of_slash = appendix_filepath[appendix_filepath.rfind("/
                ") + 1 :]
681             if (
682                 right_of_slash[: 15 + len(extension) - 1]
683                 == f"Auto_generated_{extension[1:]}"
684             ):
685                 appendices_of_extension_type.append(appendix_filepath)
686         return appendices_of_extension_type
687
688
689     def create_section(appendix_reference_index, code_filename, content):
690         """Creates the header of a latex appendix file, such that it
                contains a section that
691         indicates the section is an appendix, and indicates which pyhon
                or notebook file is
692         being included in that appendix.
693
694         :param appendix_reference_index: A counter that is used in the
                label to ensure the appendix section labels are unique.
695         :param code_filename: file name of the code file that is included
696         :param content: A list of strings that make up the appendix, with
                 one line per element.
697         """
698         # write section
699         left = "\section{Appendix "
700         middle = code_filename.replace("_", "\_")
701         right = "}\label{app:"
702         end = "}"  # TODO: update appendix reference index
703         content.append(f"{left}{middle}{right}{appendix_reference_index}{
                end}")
704         return content
705
706
707     def overwrite_content_to_file(content, filepath, content_has_newlines
        =True):
708         """Writes a list of lines of tex code from the content argument
                to a .tex file
709         using overwriting method. The content has one line per element.
710
```

```
    :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]


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}"

```

```
764
765 def substitute_appendix_code(
766     end_index, main_tex_code, start_index,
        ↪ updated_appendices_tex_code
767 ):
768     """Replaces the old latex code that included the appendices in
        ↪ the main.tex file with the new latex
769     commands that include the appendices in the latex report.
770
771     :param end_index: Index at which the appendix section ends right
        ↪ before the latex \end{appendix} line,
772     :param main_tex_code: The code that is saved in the main .tex
        ↪ file.
773     :param start_index: Index at which the appendix section starts
        ↪ right after the latex \begin{appendix} line,
774     :param updated_appendices_tex_code: The newly created code that
        ↪ includes all the relevant appendices.
775     (relevant being (in order): manually created appendices, python
        ↪ codes, pdfs of compiled jupiter notebooks).
776     """
777     updated_main_tex_code = (
778         main_tex_code[0:start_index]
779         + updated_appendices_tex_code
780         + main_tex_code[end_index:]
781     )
782     print(f"start_index={start_index}")
783     return updated_main_tex_code
784
785
786 def get_filename_from_dir(path):
787     """Returns a filename from an absolute path to a file.
788
789     :param path: path to a file of which the name is queried.
790     """
791     return path[path.rfind("/") + 1 :]
792
793
794 def get_script_dir():
795     """returns the directory of this script regardles of from which
        ↪ level the code is executed"""
796     return os.path.dirname(__file__)
797
798
799 class Appendix_with_code:
800     """stores in which appendix file and accompanying line number in
        ↪ the appendix in which a code file is
801     already included. Does not take into account whether this
        ↪ appendix is in the main tex file or not
802     """
803
804     def __init__(
805         self,
806         code_filepath,
807         appendix_filepath,
808         appendix_content,
809         file_line_nr,
810         extension,
811     ):
812         self.code_filepath = code_filepath
813         self.appendix_filepath = appendix_filepath
814         self.appendix_content = appendix_content
815         self.file_line_nr = file_line_nr
```

```
816             self.extension = extension
817
818
819     class Appendix:
820         """stores in appendix files and type of appendix."""
821
822         def __init__(
823             self,
824             appendix_filepath,
825             appendix_content,
826             appendix_type,
827             code_filename=None,
828             appendix_inclusion_line=None,
829         ):
830             self.appendix_filepath = appendix_filepath
831             self.appendix_filename = get_filename_from_dir(self.
                    ↪ appendix_filepath)
832             self.appendix_content = appendix_content
833             self.appendix_type = appendix_type  # TODO: perform
                    ↪ validation of input values
834             self.code_filename = code_filename
835             self.appendix_inclusion_line = appendix_inclusion_line
```

H Appendix Gantt.py

```
# The bottom up model that computes the TAM and TSM
import random
import numpy as np

from .Plot_to_tex import Plot_to_tex as plt_tex
from .Create_python_gantt import create_python_gantt


class Gantt:
    def __init__(self, filepath):
        self.start_line = "@startgantt"
        self.project_start_date = "2021/07-22"
        self.closed_days = ["saturday", "sunday"]
        self.gantt_font_size="skinparam classFontSize 100"
        self.box_font_size="30"

        #self.font_size="skinparam defaultFontSize  100"

        self.parents = create_python_gantt()
        self.end_line = "@endgantt"
        self.lines = self.get_list()
        self.write_gantt(filepath, self.lines)

        self.costs = None

    def get_list(self):
        lines = []
        lines.append(self.start_line)
        lines.append(f"project starts the {self.project_start_date}")
        lines = self.add_closed_dates(lines)
        lines.append(self.gantt_font_size)
        # lines.append(f"[{parent.description}] as [{parent.get_tag()
            ↪ }] lasts {parent.duration} days")
        # for parent in self.parents:
        lines = self.loop_through_parents_printing(lines)
        lines.append(self.end_line)
        return lines

    def loop_through_parents_printing(self, lines):
        # print descriptions
        for i in range(0, len(self.parents)):
            lines = self.print_parent_descriptions(lines, self.
                ↪ parents[i])

        # print order
        for i in range(0, len(self.parents)):
            if i > 0:
                if not self.parents[i].starts_at_child_nr_start is
                    ↪ None:
                    #print(f'starting parent at:{self.parents[i].
                        ↪ starts_at_child_nr_start}')
                    lines.append(
                        f"[{self.parents[i].get_tag()}] starts at [{
                            ↪ self.parents[i].
                            ↪ starts_at_child_nr_start}]'s start"
                    )
                else:
                    print(f'parent_descr={self.parents[i].description
                        ↪ }, and starts at {self.parents[i].
                        ↪ starts_at_child_nr_start} and  tag={self.
```

```
                    ↪ parents[i].get_tag()}, writing end')
                lines.append(
                    f"[{self.parents[i].get_tag()}] starts at [{
                        ↪ self.parents[i-1].get_tag()}]'s end"
                )
            lines = self.print_parent_order(lines, self.parents[i])

        # print colour
        for i in range(0, len(self.parents)):
            lines = self.print_parent_colour(lines, self.parents[i])

        # compute costs
        total_costs = 0
        for i in range(0, len(self.parents)):
            total_costs, lines = self.print_parent_costs(total_costs,
                ↪  lines, self.parents[i])
        return lines

    def print_parent_descriptions(self, lines, parent):
        lines.append("")
        lines = self.print_descriptions(parent, lines)
        return lines

    def print_parent_order(self, lines, parent):
        lines.append("")
        lines = self.print_order(parent, lines)
        return lines

    def print_parent_colour(self, lines, parent):
        lines.append("")
        lines = self.print_colour(parent, lines)
        lines.append("")
        return lines

    def print_parent_costs(self, total_costs, lines, parent):
        lines.append("")
        total_costs, lines = self.print_costs(total_costs, parent,
            ↪ lines)
        print(f'total_costs={total_costs}')
        lines.append("")
        return total_costs, lines

    def print_descriptions(self, activity, lines):

        if not activity.font_size is None:
            font_size = activity.font_size
        else:
            font_size=self.box_font_size
        lines.append(
            f"[<size:{font_size}>{activity.description}] as [{
                ↪ activity.get_tag()}] lasts {activity.duration} days
                ↪ "
        )
        for child in activity.children:
            lines = self.print_descriptions(child, lines)
        return lines

    def print_order(self, activity, lines):
        # Write order for all children in an activity
        for i in range(0, len(activity.children)):
            if i == 0:
                lines.append(
```

```
109                     f"[{activity.children[i].get_tag()}] starts at [{
                            ↪ activity.get_tag()}]'s start"
110                 )
111             else:
112                 # check if it starts at the start or end of some
                        ↪ activity
113                 if not activity.children[i].starts_at_child_nr_start
                        ↪ is None:
114                     #print(f'activity.description={activity.
                            ↪ description}, appending at  {activity.
                            ↪ children[i].starts_at_child_nr_start}s
                            ↪ start')
115                     lines.append(
116                         f"[{activity.children[i].get_tag()}] starts
                                ↪ at [{activity.children[i].
                                ↪ starts_at_child_nr_start}]'s start"
117                     )
118                 else:
119                     #print(f'activity.description={activity.
                            ↪ description}, appending at  {activity.
                            ↪ children[i-1].get_tag()}s  end')
120                     lines.append(
121                         f"[{activity.children[i].get_tag()}] starts
                                ↪ at [{activity.children[i-1].get_tag()
                                ↪ }]'s end"
122                     )
123
124         # start recursive loop to write order of each of the
                ↪ childeren
125         for child in activity.children:
126             lines = self.print_order(child, lines)
127         return lines
128
129     def print_colour(self, activity, lines):
130         lines.append(f"[{activity.get_tag()}]  is colored in {
                ↪ activity.colour}")
131         for child in activity.children:
132             lines = self.print_colour(child, lines)
133         return lines
134
135     def print_costs(self, total_costs, activity, lines):
136         lines.append(f"'[{activity.description}]  takes: {activity.
                ↪ duration}[days] equating to:{activity.duration*activity
                ↪ .hours_per_day}[hours] and costs: {activity.hourly_wage
                ↪ } per hour, yielding activity costs: {activity.duration
                ↪ *activity.hours_per_day*activity.hourly_wage} Euros.")
137         total_costs=total_costs+activity.duration*activity.
                ↪ hours_per_day*activity.hourly_wage
138         for child in activity.children:
139             total_costs, lines = self.print_costs(total_costs, child,
                    ↪  lines)
140         return total_costs, lines
141
142     def add_closed_dates(self, lines):
143         for closed_day in self.closed_days:
144             lines.append(f"{closed_day} are closed")
145         return lines
146
147     def write_gantt(self, filepath, list):
148         with open(filepath, "w") as f:
149             for item in list:
150                 f.write("%s\n" % item)
```

```
151         f.close()
152 
153     def addTwo(self, x):
154         """adds two to the incoming integer and returns the result of
                ↪  the computation."""
155         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()
```