Example to plot directly into latex

19-10-2019

1 Introduction

Welcome, this document presents our market analysis for the TruCol consultancy. 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.

This market analysis estimates the total adressable market (TAM) as well as the total servicable market (TSM) for a consultancy service that is being developed to help companies get the most out of the TruCol protocol. Since this market analysis consists of a rough estimate, three different estimation methods are used for generating the TAM and TSM estimates. The redundancy is introduced to establish some overview/reference 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

To illustrate how the python code exports the figures directly into the report, this second "hw2" is included. Below are the pictures that are created by the code listed in **??** and **??**.

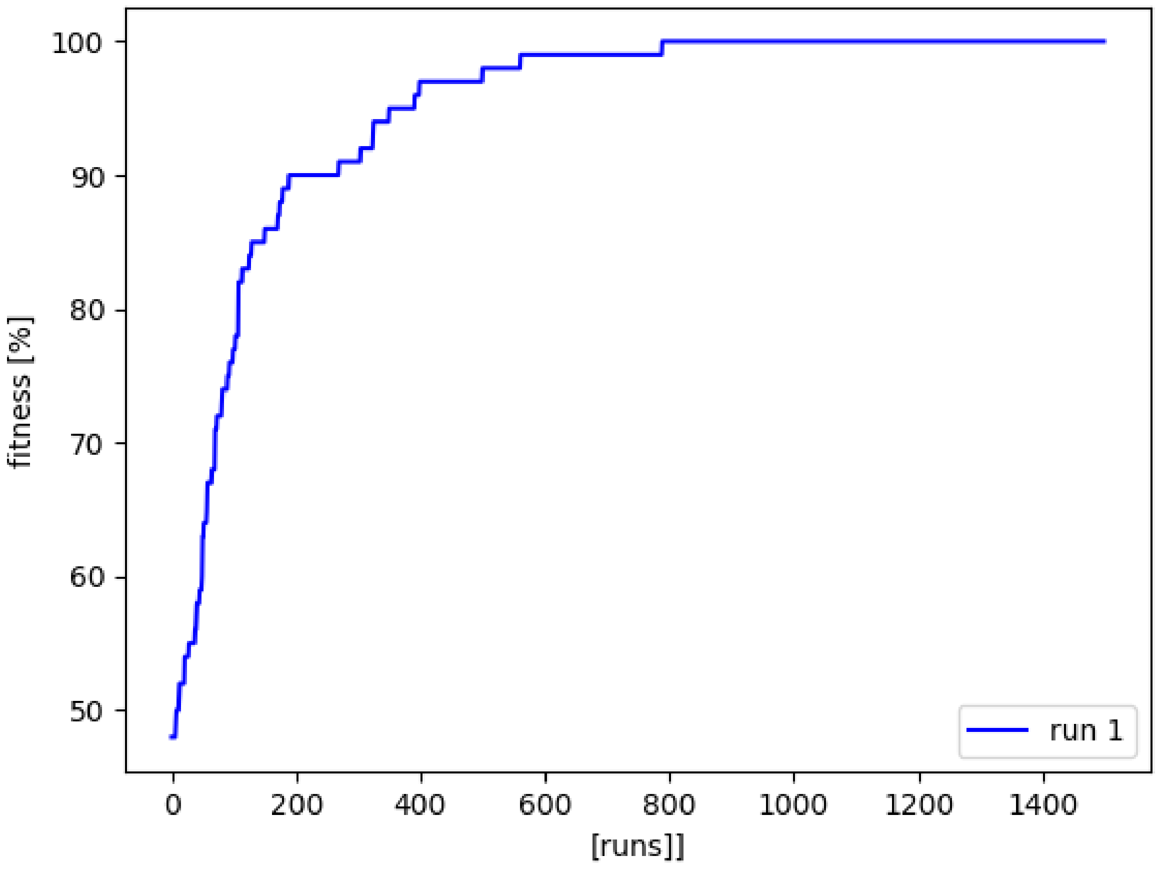

Figure 1: Performance of some genetic algorithm

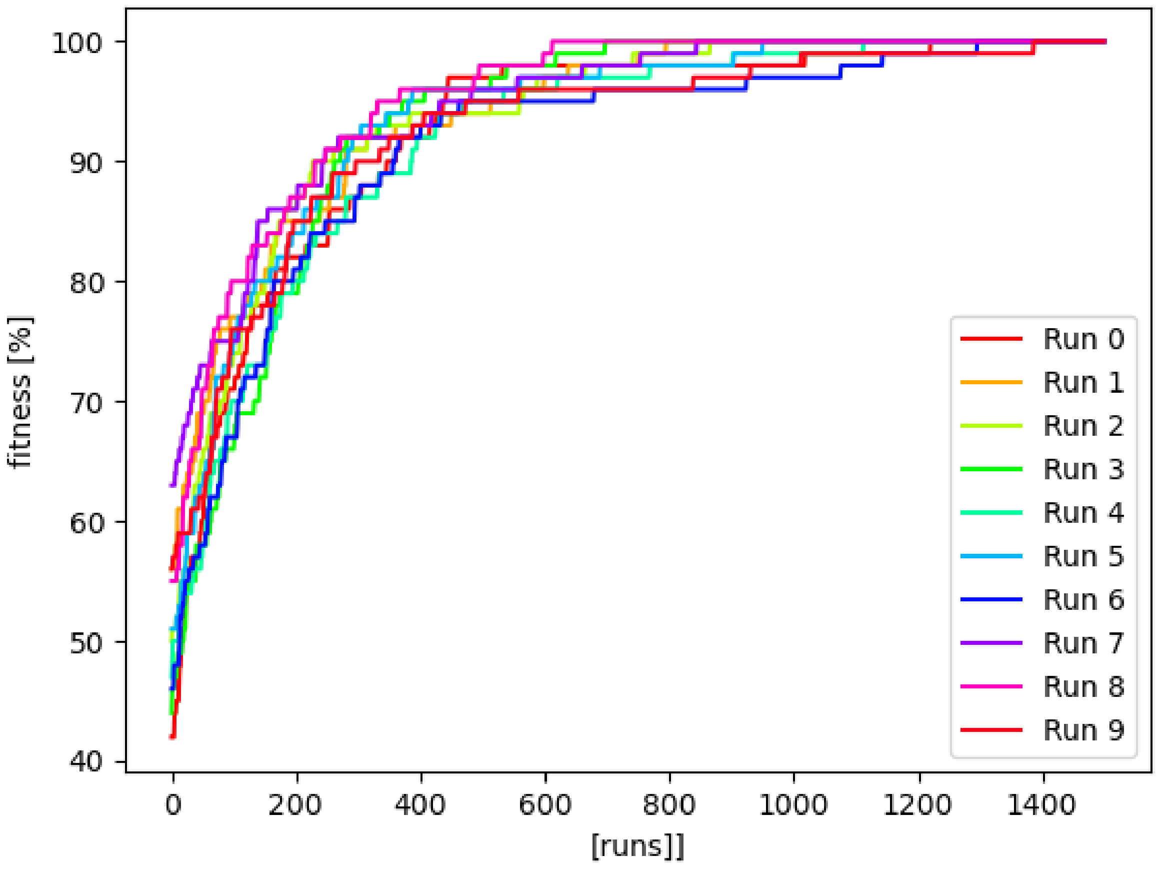

Figure 2: Performance of some genetic algorithm

3 Model Description

3.1 Top Down

3.2 Bottom Up

3.3 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

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()

# 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 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}')]

    python_filepaths = get_filenames_in_dir('py',script_dir, ['
      ↪ __init__.py'])
    compiled_notebook_pdf_filepaths = get_compiled_notebook_paths(
      ↪ script_dir)

    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)

    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')

    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:
    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
    for appendix in python_appendices:
        line = update_appendix_tex_code(appendix.appendix_filename, project_nr)
        main_appendix_inclusion_lines.append(line)

    for appendix in notebook_appendices:
        line = update_appendix_tex_code(appendix.appendix_filename, project_nr)
        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):
                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


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).
    :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 (whereever 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 = 0

    for code_filepath in code_filepaths:
        latex_relative_filepath = f'latex/project{project_nr}/../../{code_filepath[len(root_dir):]}'
        content = []
        filename = get_filename_from_dir(code_filepath)
        content = create_section(appendix_reference_index, filename, content)
        inclusion_command = get_latex_inclusion_command(extension, latex_relative_filepath)
        content.append(inclusion_command)
        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 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]


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, 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.
    """
    left = "\input{latex/project"
    middle = "/Appendices/"
    right = "} \\newpage\n"
    return f'{left}{project_nr}{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


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
```

E Appendix Plot_to_tex.py

```python
### 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.'''
        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()
```