THE DREAM

TECHNICAL MANUAL

By Siddharth Raghavan

August 17, 2020

TABLE OF CONTENTS

PURPOSE

This document is meant for people who are interested in continuing The Dream. It is a medium-high level overview of the overall structure and code used in The Dream. I highly recommend that new developers have at least some basic Python knowledge and some basic VBA knowledge. Having some background knowledge in data frames in Python is beneficial. Additionally, it is ideal if new developers have experience with web automation.

It is important to note that an **Instruction Manual** has been written for The Dream, the purpose of which is to guide users to download and use the tool. The Instruction Manual describes the *how* of operating the tool, but the Technical Manual describes the *how come* of the tool. That is, it describes how the tool came to be – with all the underlying code.

The hope is that with this Technical Manual, the project is easily understood and the code can be worked on/altered. Due to the ambitious nature of the project, with several to-do's for the future, I expect this manual to be a guiding light amidst the many lines of code written.

PROJECT

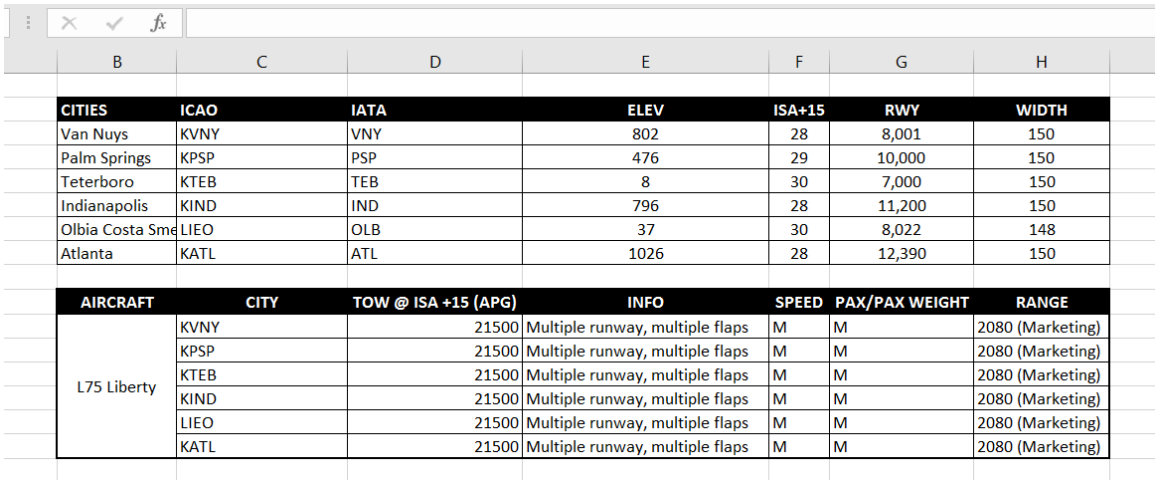
The Dream is a project undertaken to automate the bulk of range map and route study requests. These requests usually come in from Sales Directors, who pursue leads and give directives on what performance reports they think allow for a convincing narrative through which a sale can be made. Along with automation comes obvious increases in time savings, lower workloads, and greater accuracy.

The Dream is to have these two daily types of requests (range maps and route studies), of which multiple of them can take up to 3-4 hours of work during a busy work day, to be done in 1-2 hours of barely-guided non-specialist (i.e. anyone can do it) work with 100% work accuracy.

At the moment, there has been work in the project to lay the foundation for complete automation, by automating the first steps in any range map or route study request.

OVERVIEW AND WORK FLOWS

Following is a typical Excel data sheet that would be filled in while producing range maps. The first table holds information relevant to all range map and route study requests – general information about the airfields involved. The second table also involves necessary data used for all range map and route study requests – in the maximum takeoff weight (TOW) found for the airfields for a specific aircraft (or aircrafts).



CITIES	ICAO	IATA	ELEV	ISA+15	RWY	WIDTH
Van Nuys	KVNY	VNY	802	28	8,001	150
Palm Springs	KPSP	PSP	476	29	10,000	150
Teterboro	KTEB	TEB	8	30	7,000	150
Indianapolis	KIND	IND	796	28	11,200	150
Olbia Costa Sme	LIEO	OLB	37	30	8,022	148
Atlanta	KATL	ATL	1026	28	12,390	150

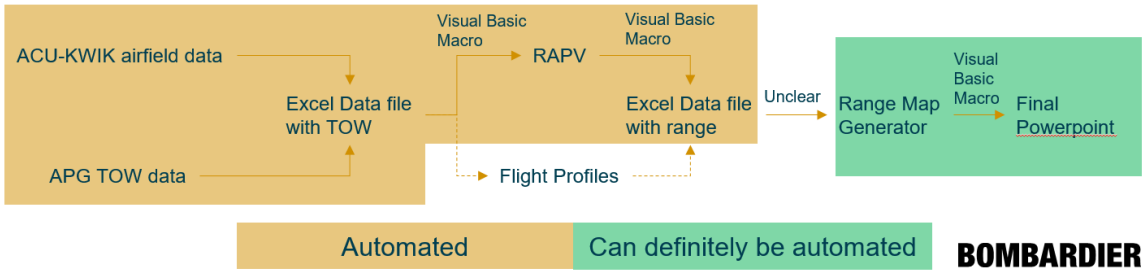
AIRCRAFT	CITY	TOW @ ISA +15 (APG)	INFO	SPEED	PAX/PAX WEIGHT	RANGE
L75 Liberty	KVNY	21500	Multiple runway, multiple flaps	M	M	2080 (Marketing)
	KPSP	21500	Multiple runway, multiple flaps	M	M	2080 (Marketing)
	KTEB	21500	Multiple runway, multiple flaps	M	M	2080 (Marketing)
	KIND	21500	Multiple runway, multiple flaps	M	M	2080 (Marketing)
	LIEO	21500	Multiple runway, multiple flaps	M	M	2080 (Marketing)
	KATL	21500	Multiple runway, multiple flaps	M	M	2080 (Marketing)

Typically, filling in all of this information is done manually, with all of the information in the first table found (generally) through ACU-KWIK, a website for general airfield information. Only the ISA+15 column is something that is usually computed on the Excel sheet, and is dependent on the elevation.

The TOW column in the second table is filled in by analyzing PDF reports that are generated after navigating through the Aircraft Performance Group (APG) website. APG's tool allows users to obtain the TOWs for a number of aircraft, for a large database of airfields, at a large range of different temperatures and for various different takeoff conditions (e.g. wet, ice, standing water, etc.) and various different takeoff inputs (e.g. static takeoff, rolling takeoff, APU on, APU off, etc.). The rest of the second table includes a Sales Analyst's inputs, to find the range of the specific aircraft(s) from those airfields. The range is found by using RAPV or Flight Profiles.

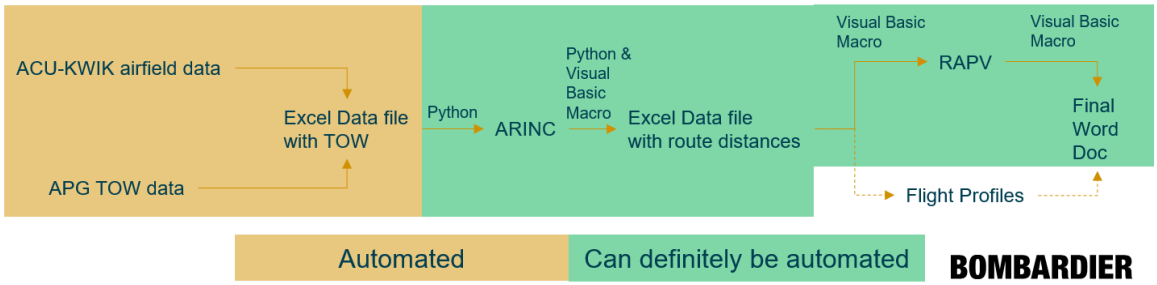
Therefore, The Dream automates all data collection from these two sources (ACU-KWIK and APG). The heart of the project is to "web scrape" and automate data collection. Along with these automations, a template Excel file is the main user interface and guides all requests to be done in a consistent manner for proper backtracking and documentation of work. For this project, **Python** was used as the web scraping and data collection tool, and **VBA** was used in conjunction with Python to format data collected and to (potentially) be used as a link with other tools to enable greater functionality.

The following flowchart describes the general steps involved for a typical range map request.



Notice how the first two steps have been automated along with some automation linking with RAPV. But, there is plenty of potential to fully automate the process.

The following flowchart describes the general steps involved for a typical route study request.

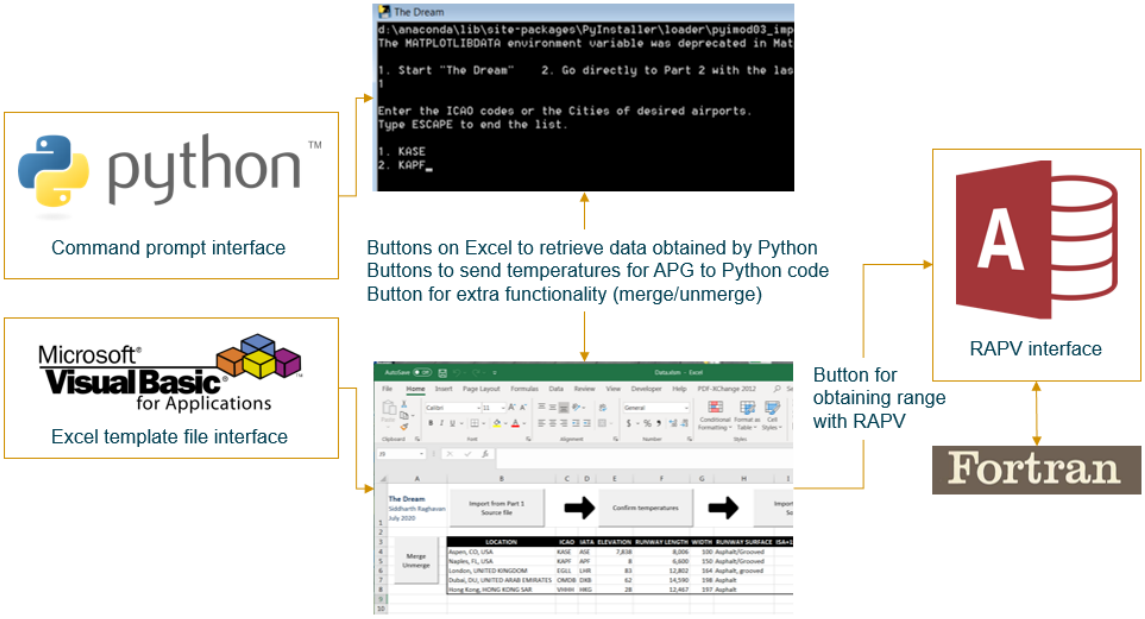


Automating route studies is more involved, and involves more specialist inputs (i.e. only an analyst can typically properly do route studies). Still, for the bulk of route studies, generalities can be described and hence automated. Notice again that the same two initial steps of automation apply to route studies too.

HIGH-LEVEL STRUCTURE

The general structure starts users off with two interfaces; the command prompt (Python) interface, and the Excel template file (VBA) interface. User on these interfaces give their inputs either through typing and hitting ENTER on the command prompt or modifying cell values and clicking on a button on the Excel file. These two interfaces are linked to each other, and feed each other with relevant information. For example, the Python script prepares data that can be formatted neatly in the Excel file, and the Excel file sends back information like the temperature at which takeoff performance is to be analyzed to the Python script.

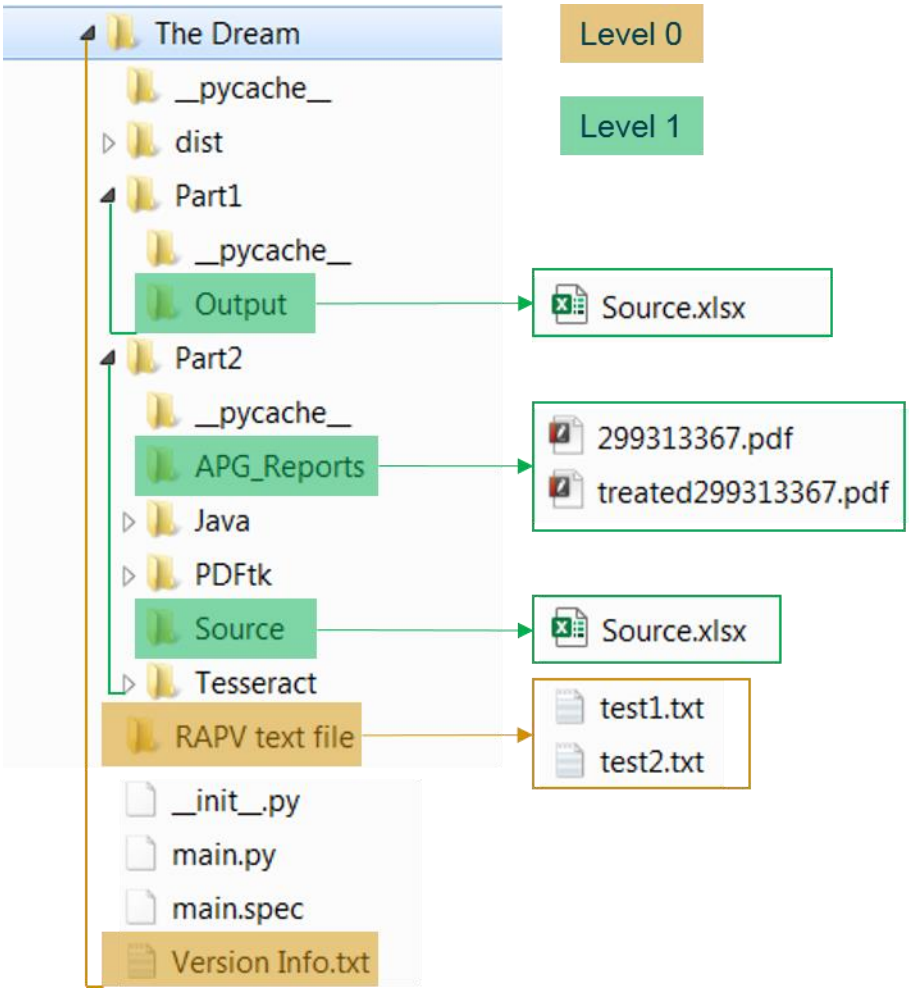
Additionally, the Excel file has additional links to the RAPV tool hosted on an MS Access Database. The link here is obvious because both the Excel file and RAPV tool have underlying VBA code that can be connected. The RAPV file invokes a Fortran executable, which performs the route analysis. This is all accessible from the Excel template file.



FOLDER FILE PATHS

The Dream is (and should be) placed in the C: drive. This is imperative because the Excel template file has macros that refer to specific file paths – for which a defined folder location is required. The Python script on the other hand does not refer to specific file paths, but relative file paths because the script is in-built in an executable file in The Dream folder.

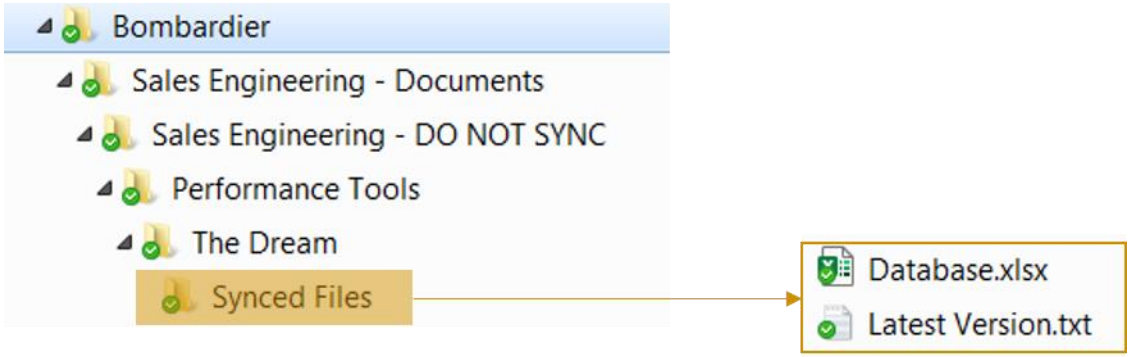
The following figure clearly shows the different folder file paths.



As seen, The Dream has both files and folders in Level 0 (i.e. these are the things you see when you open The Dream). The files are: __init__.py, main.py, main.spec, and Version Info.txt. The folders are __pycache__, dist, Part1, Part2, and RAPV text file. Level 1 files and folders are placed inside Level 0 folders. The folders inside Part1 are __pycache__ and Output. The folders inside Part2 are __pycache__, APG_Reports, Java, PDFtk, Source, and Tesseract.

Not all of these files and folders are important for normal usage. The most important ones are the ones highlighted. The Output folder has a Source (for Part 1) Excel file. The APG_Reports folder is the default download location for APG reports when using the tool. The Source folder has a Source (for Part 2) Excel file. The RAPV text file folder has text reports (test1.txt and test2.txt) for maximum ranges. These are the text file reports produced by RAPV. Finally, the Version Info.txt file is a file local to each version of The Dream. It contains information about the version number, and when that version was updated.

Additionally, it is imperative that the Synced Files folder on Sharepoint is synced. Details on how to sync the folder is described in detail in the Instruction Manual for The Dream. This folder can be either synced directly (as described in the Instruction Manual) or can be synced indirectly by syncing a parent folder. In the following figure, the file path to the Synced Files folder can be seen to be a result of syncing a parent folder. The syncing is done through OneDrive.



The Database Excel file is the file referred to whenever a city name is enter into the prompt interface. It has more than 13,000 airfields. The database updates every time a new non-duplicate (i.e. not currently in the database) ICAO code is entered into the program. The Latest Version text file holds information of the version number of the latest version of The Dream. This is compared against the information in the local Version Info text file. Hence, when a new version of The Dream is up on Sharepoint, users will be automatically notified that their version is out of date, and that they are advised to re-install the tool.

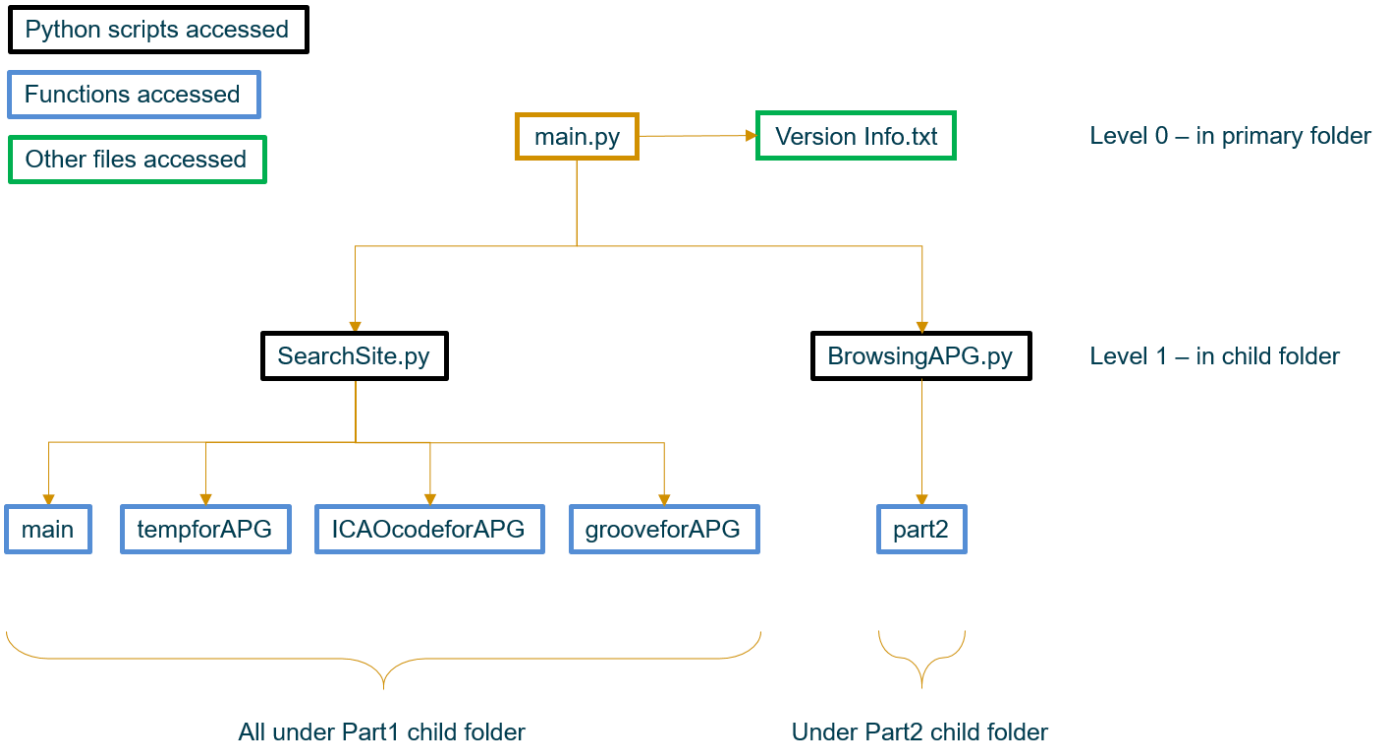
USAGE INFORMATION AND ASSUMPTIONS

1. Python code and VBA code was written under a Windows 10 Operating System (OS). On translating to a Windows 7 OS, the Python code (in .exe file) and the Excel template file (with the VBA code) transferred with no problems.
2. The Python script was written in Python Version 3.7. Code was compiled using PyCharm IDE.
3. The most important script from which the program starts up is main.py. This was converted to a .exe file using PyInstaller, with the --onedir choice for faster compiler executation. This is why we don't have a single .exe file, but rather a bunch of files and folders with the .exe file buried 2 folders deep within the dist folder (see Instruction Manual to navigate to main.exe file).
4. Apart from Python libraries, the Python scripts have some external (i.e. non Python) dependencies. These include:
    a. Sharepoint folder (Synced Files) that **needs** to be synced for proper operation of the tool.
    b. A Chromedriver.exe that is used for web automation. Version of the Google Chrome window is 75.0.3770.90. If the Chromedriver.exe file is a higher version (or lower version) than the current Google Chrome version, the tool may not work.
    c. PDFtk, which is a tool used to manipulate PDF documents. The free version's executable is used to uncompress and compress APG PDF reports. The encoding of the PDF (in UTF8) can then be manipulated to remove watermarks which makes data processing from the PDF file easier. Version 2.02 was downloaded and used.
    d. Pytesseract, which is a Python wrapper of an Optical Character Recognition (OCR) engine released by Google called Tesseract. This is used to look at images (.png files of snippets taken from the APG PDF reports) and convert them to text. Pytesseract version 0.3.5 was downloaded and used.
    e. Java jdk version 14.0.2. This is implicitly required because the code invokes the Tabula-py library, which is a Python wrapper of Tabula-java. This is used to read data from tables in PDFs.

Main Python library dependencies will be described in the next sections.

CODE BREAKDOWN - PYTHON

Initially, the project was divided into two parts: one for tackling data collection from ACU-KWIK, and one for obtaining data from the PDF reports generated by APG. Following is a figure that shows a high-level view of the Python code. In other words, the figure focuses on the direct scope (i.e. what the script directly invokes) of the main.py script in the primary folder.



Notice how the main.py script invokes the two parts as mentioned before, as seen by the branching of main.py to SearchSite.py (for Part 1) and BrowsingAPG.py (for Part 2). The main.py script has 134 lines of code.

For main.py, the Python dependencies (apart from standard libraries) were os, inspect and sys (for system file path manipulation), and Path (also for file paths).

Hence, the next sub-section focuses on obtaining data from ACU-KWIK and writing to the Excel file – Part 1. The sub-section following that explains how navigation through APG occurs and how the PDFs are processed for data that is then written to the Excel file – Part 2.

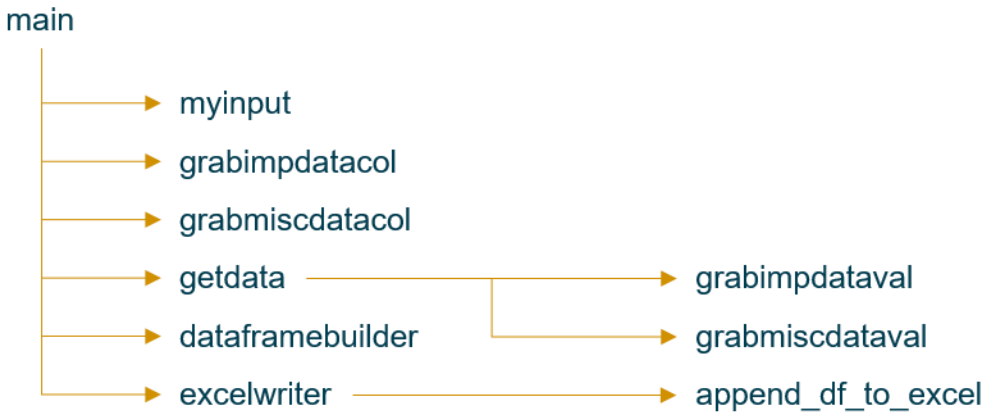
Part 1

SearchSite.py is the script that encompasses Part 1. It has functions main, tempforAPG, ICAOcodeforAPG, and grooveforAPG. However, these are only the functions that are

visible (or in the scope) of the main.py script in the primary folder. Other functions within the SearchSite.py script include: grabimpdatacol, grabmiscdatacol, grabmiscdataval, grabimpdataval, dataframebuilder, getdata, append_df_to_excel, excelwriter, and myinput. The SearchSite.py script has 615 lines of code.

For SearchSite.py, the Python dependencies (apart from standard libraries) were BeautifulSoup (for HTML parsing), os (for system file paths), urllib.request and urllib.parse (for accessing and parsing through web pages), pandas (for data frames), numpy (for efficient arrays), time (for printing the time taken to compile sections of code), and finally Path (for system paths).

The most important function is the main function, and is the function from which the other functions are invoked. It is at the end of the main function, that the compiler switches back to the main.py script in the primary folder. The following figure shows the top-down breakdown (i.e. in the order the functions were invoked) of the main function.



main
Functions as the "brain" of this script, handling all the other functions and invoking them at appropriate times. It outputs a list of the processed (or valid) ICAO codes.
Input – NA
Output – List of processed (or valid) ICAO codes

myinput
Handles the user input into the prompt. Simply adds the entered cities/codes to a list. Ends if user types "escape".
Input – NA
Output – List with user inputs

grabimpdatacol
Creates a list for the column headers of eventual data frame built for the DataEntry sheet on the Excel template file. Includes "ICAO", "IATA", etc.
Input – NA
Output – List with column headers

grabmiscdatacol
Creates a list for the column headers of eventual data frame built for the MiscEntry sheet on the Excel template file. Includes “ICAO”, “Latitude”, “Longitude”, etc.
Input – NA
Output – List with column headers

getdata
An important function that sends in the URL request and parses through the ACU-KWIK web page, using what the user had previously typed as a URL identifier. It then uses BeautifulSoup to parse through the HTML source code of the web page and grab text fields from the page. It outputs a tuple – of important data values (corresponding to the important column headers of grabimpdatacol), and of miscellaneous data values (corresponding to the miscellaneous column headers of grabmiscdatacol) by calling the relevant functions.
Input – String for the city/code. This is an element of the list from the output of myinput
Output – Tuple of important data values and miscellaneous data values (while calling those respective functions to obtain the values)

grabimpdataval
This is called by the getdata function. It takes in the parsed text fields from ACU-KWIK and gets the important text, corresponding to the values under columns of grabimpdatacol. This is returned as a list.
Input – String of text from ACU-KWIK website.
Output – List of strings that correspond to the values under column headers as returned by grabimpdatacol

grabmiscdataval
This is called by the getdata function. It takes in the parsed text fields from ACU-KWIK and gets the important text, corresponding to the values under columns of grabmiscdatacol. This is returned as a list.
Input – String of text from ACU-KWIK website.
Output – List of strings that correspond to the values under column headers as returned by grabmiscdatacol

dataframebuilder
A quick function written to create a data frame object given some column headers and some values (or data) corresponding to those columns.
Input – List of column data (headers) and list of values (data to go into the data frame)
Output – Data frame object with the column headers and data

excelwriter
Takes in a data frame object and an integer value. Depending on the value of the integer, this function either:

1. Writes data frame object into the DataEntry sheet of the Source Excel file
2. Writes data frame object into the MiscEntry sheet of the Source Excel file
3. Appends data frame object (with a helper function) to the shared database Excel on Sharepoint. It then drops duplicate rows in that database, and rewrites the database file without the duplicate rows

Input – Data frame object and specific integer value
Output – NA

append_df_to_excel
This is a helper function that is called by the excelwriter function. It mainly takes in a dataframe object to append to an existing Excel sheet, and it appends it to the Excel sheet. It takes in some other parameters which are less important.
Input – String of the filename to append to (i.e. file path to the shared database file on Sharepoint), data frame object, and some other parameters
Output – NA

Apart from these functions, there are 3 functions (excluding the main function) that are not invoked by the main function – tempforAPG, ICAOcodeforAPG, and grooveforAPG. These functions are used by the main.py script in the primary folder.

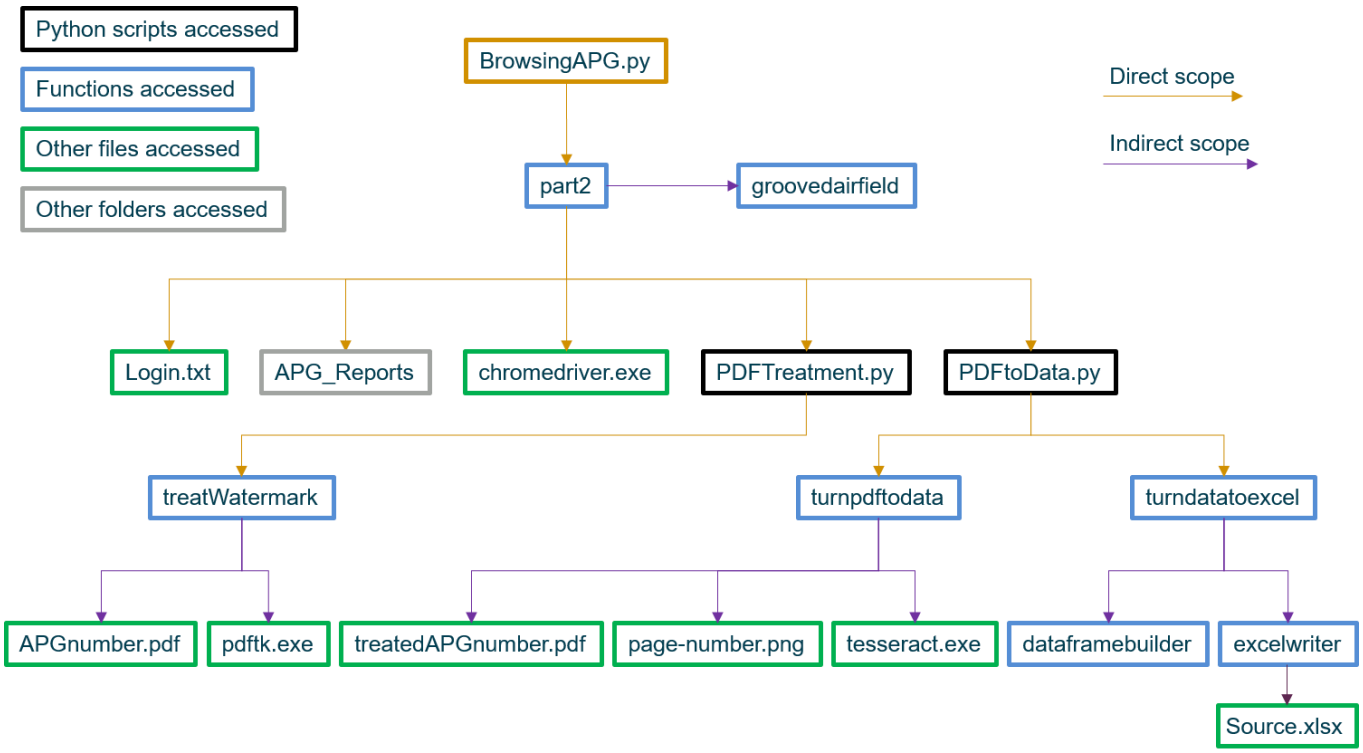
tempforAPG
This is a function that looks into the Source file (of Part 1) and obtains the temperature. If there is are no temperature values, it forces the user to click on the "Confirm Temperatures" button on the Excel template file.
Input – List of processed codes as returned by the main function
Output – Series object of the "confirmed temperature" values

grooveforAPG
A function that is later useful for wet runway analysis. It looks into the Source file (of Part 1) and divides the data frame into airfields with grooved runway surfaces, and airfields with non-grooved runway surfaces. It returns the grooved airfields' ICAO codes and chosen temperatures, and non-grooved airfields' ICAO codes and chosen temperatures.
Input – NA
Output – List of grooved airfields' ICAO codes, list of grooved airfields' temperatures, list of non-grooved airfields' ICAO codes, and list of non-grooved airfields' temperatures

ICAOcodeforAPG
A function to ensure modularity in the program. In the case that a user wanted to proceed directly to Part 2 (with a previous Part 1 source file) without having to go through Part 1 again, they can do so. This function looks into the Part 1 source file.
Input – NA
Output – List of ICAO codes present in the Part 1 Source file

Part 2

Part 2 uses multiple scripts. Namely, BrowsingAPG.py (in the scope of the main.py script in primary folder), PDFTreatment.py and PDFtoData.py (both in the scope of the BrowsingAPG.py script). The BrowsingAPG.py is the main script from which other scripts and functions are invoked. The following figure shows the direct scope and indirect scope (i.e. not all functions/files/folders are directly invoked by the script) of the BrowsingAPG.py script.



The BrowsingAPG.py script has its own two functions (part2 and groovedairfield) and invokes several other files and scripts. In fact, it is within the part2 function, that these several other files and scripts are invoked (including the groovedairfield function local to the BrowsingAPG.py script). The PDFTreatment.py script in turn has the treatWatermark function which uses pdftk.exe to uncompress and compress the APG pdf file. The PDFtoData.py script has two functions, turnpdftodata and turndatatoexcel. The turnpdftodata function works with a treated version of the APG pdf file. Tesseract.exe is also invoked to convert image (.png) files to text. It is also in the turnpdftodata function that tabula commands are invoked, which in turn necessitates a Java working environment. The turndatatoexcel function is a simple function that invokes the dataframebuilder function, and the excelwriter function (which is the function that ends up writing the data frame into the Source.xlsx Excel file).

The BrowsingAPG.py script has 1750 lines of code. The PDFTreatment.py script has 83 lines of code. Finally, the PDFtoData.py script has 318 lines of code. Totally, Part 2 uses 2151 lines of code.

For BrowsingAPG.py, the Python dependencies (apart from standard libraries) were a lot of Selenium library modules (for web automation), time (for tracking time taken for code execution), os and Path (for system file paths).

For PDFTreatment.py, the Python dependencies (apart from standard libraries) were re (for string manipulations under regular expressions), os and Path (for system file paths), time (for tracking time taken for code execution), and subprocess (for command prompt compiler executions).

For PDFtoData.py, the Python dependencies (apart from standard libraries) were fitz (for matrix building and image manipulation), os and Path (for system file paths), Image (for creating image objects for image-to-text conversion), pytesseract (to work along with tesseract.exe to actually convert image to text), tabula (a wrapper of Java code to extract data from tables in pdf documents), time (for tracking time taken for code execution), pandas (for data frames), and numpy (for efficient arrays).

To fully understand Part 2's code, it is important to understand how a user would have normally entered inputs into the APG website. The code serves to mimic these inputs by looking at the JavaScript source code and manipulating it to register as user inputs. A detailed explanation of this part of the code will not be provided as it is a bit vast in length, and a lot of the effort goes into simply navigating the nuances of the APG website. Instead, a description of functions involved will be given (similar to Part 1).

part2
This is the main function of the BrowsingAPG.py script. After part2 is done executing, the compiler switches back to the main.py script in the primary folder. This function invokes the rest of the functions/files/folders in Part 2.
Input: Series object of confirmed temperature values, list of ICAO codes, list of ICAO codes of grooved airfields, list of temperatures for the grooved airfields, list of ICAO codes of non-grooved airfields, list of temperatures for the non-grooved airfields
Output: NA

groovedairfield
This is a function that is similar to the part2 function. When a user enters that they want a wet runway analysis into takeoff performance, the airfields are split into non-grooved airfields and grooved airfields. The entire process of navigating through the APG website is then repeated for grooved airfields, encompassed by this function.
Input: String of login username, string of login password, dictionary of aircraft choices, integer of user aircraft choice, integer of a "plus" factor used to navigate through the dictionary of aircraft choices, list of ICAO codes for grooved airfields, string of user input to the default choice prompt, list of ICAO codes for non-grooved airfields, list of temperatures for the grooved airfields
Output: String of the ID number of the APG report downloaded, string of the date of the APG report downloaded, string of the aircraft name in the APG report downloaded, and list of the processed ICAO codes for grooved airfields by APG

treatWatermark
Given an ID number, date, and aircraft name as given in the APG report downloaded, this function uses pdftk to uncompress the pdf file, make changes to the pdf's encoding by finding these identifiers and removing them – thus removing the watermarks in the pdf, and compressing the pdf file back to obtain a final "treated" pdf file that is without watermarks. This "treated" pdf file is uses for data processing.
Input: String of ID number, string of date downloaded, string of aircraft name
Output: NA

turnpdftodata
This is a function that works with the "treated" pdf from the previous function, and obtains the maximum TOW value. It needs to do this in a structured manner, and does so by building a data frame of the aircraft names (as column headers), ICAO codes (as row indices), and TOW values (as the data of the data frame). This is the function that invokes a tabula dependency, which has an implicit dependency on Java. It is also the function that uses Tesseract, as it needs to read the ICAO code in the top right corner of the pdf report. This is because the maximum TOW must be found for **each** airfield, not through all the tables from all airfields. This function invokes
Input: String of ID number, list of ICAO codes processed by APG, String of whether the user chose wet runway analysis, String of aircraft name, list of cumulative maximum values of TOW, list of cumulative aircraft names
Output: List of ICAO codes as found in the APG report, list of cumulative maximum values of TOW, list of cumulative aircraft names

turndatatoexcel
The logical next step after obtaining the data from the APG pdfs is to write this data into data frames and then write those data frame to Excel sheets. This is exactly what this function does. It invokes helper functions – dataframebuilder and excelwriter – to achieve this.
Input: List of ICAO codes as found in the APG report, list of cumulative maximum values of TOW, list of cumulative aircraft names. Notably, these are exactly the output of the turnpdftodata function
Output: NA

dataframebuilder
Given index values, column values (as column headers), and data (values of maximum TOW), this function simply builds a data frame. This is invoked by turndatatoexcel.
Input: List of index values, list of values (data), list of column values (headers)
Output: Data frame object characterized by these inputs

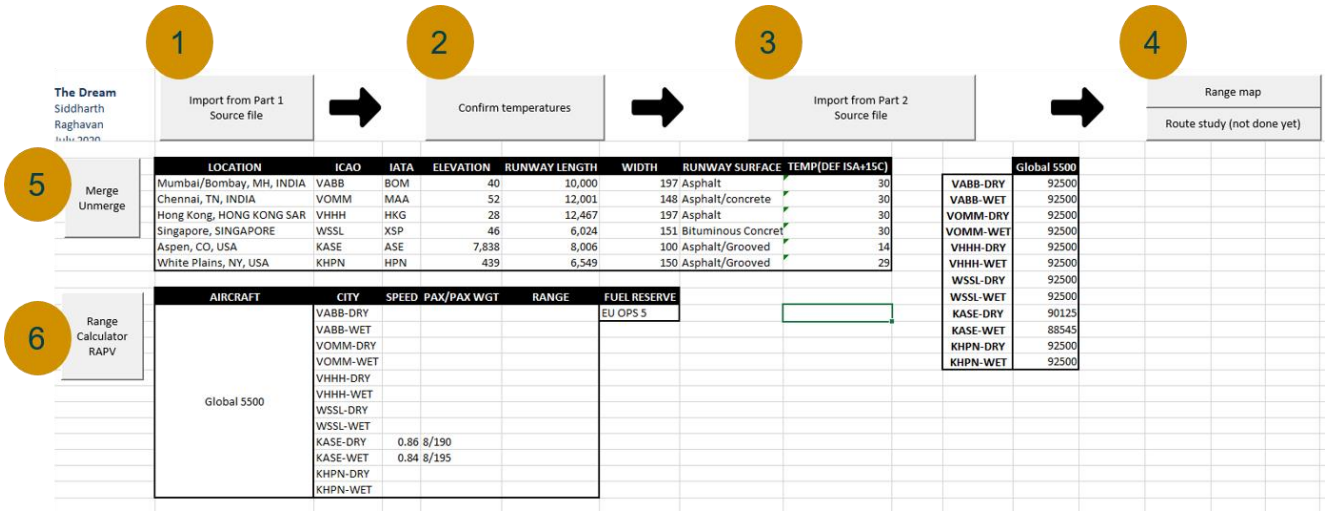
excelwriter
Given a data frame object (that is the output of the dataframebuilder function), this function writes the data frame to the Source.xlsx Excel file of Part 2.
Input: Data frame object which is the output of the dataframebuilder function, and integer value (not important in this context)
Output: NA

Summary

Throughout all the scripts and functions, a total of 2766 lines of code was written. Part 1 has a simple structure to follow and the code is not voluminous. Part 2 however is characterized by a need to navigate the nuances of the APG website. Every single input, which can be as simple as a click on a button, may be complicated to execute reliably through code. Thus, Part 2 is more challenging to understand with its web automation. As previously mentioned, it is beneficial to new developers to understand how to navigate through APG themselves (manually) before trying to understand the code to automate that navigation.

CODE BREAKDOWN - VBA

The current most up-to-date Excel template file has 6 functional buttons that are tied in to modules. This Excel template file is to be used in conjunction with the prompt window of the Python script. The Excel file is to ensure ease of formatting data collected by the Python script, and to send temperatures (by clicking on the Confirm Temperatures button) to the Python script. Additionally, as a last minute addition to this entire project, a preliminary link to RAPV has been made, further proving that complete automation indeed has a lot of potential.



LOCATION	ICAO	IATA	ELEVATION	RUNWAY LENGTH	WIDTH	RUNWAY SURFACE	TEMP(DEF ISA+15C)
Mumbai/Bombay, MH, INDIA	VABB	BOM	40	10,000	197	Asphalt	30
Chennai, TN, INDIA	VOMM	MAA	52	12,001	148	Asphalt/concrete	30
Hong Kong, HONG KONG SAR	VHHH	HKG	28	12,467	197	Asphalt	30
Singapore, SINGAPORE	WSSL	XSP	46	6,024	151	Bituminous Concret	30
Aspen, CO, USA	KASE	ASE	7,838	8,006	100	Asphalt/Grooved	14
White Plains, NY, USA	KHPN	HPN	439	6,549	150	Asphalt/Grooved	29

	Global 5500
VABB-DRY	92500
VABB-WET	92500
VOMM-DRY	92500
VOMM-WET	92500
VHHH-DRY	92500
VHHH-WET	92500
WSSL-DRY	92500
WSSL-WET	92500
KASE-DRY	90125
KASE-WET	88545
KHPN-DRY	92500
KHPN-WET	92500

AIRCRAFT	CITY	SPEED	PAX/PAX WGT	RANGE	FUEL RESERVE
Global 5500	VABB-DRY				EU OPS 5
	VABB-WET				
	VOMM-DRY				
	VOMM-WET				
	VHHH-DRY				
	VHHH-WET				
	WSSL-DRY				
	WSSL-WET				
	KASE-DRY	0.86	8/190		
	KASE-WET	0.84	8/195		
	KHPN-DRY				
	KHPN-WET				

Button/Macro Explanation

For all the buttons that are linked to the various modules, there are usually 3 base Subs. These are the Sub AllowSheetActionsWhenProtected, Sub OpenWorkbook, and Sub CloseWorkbook. The first Sub protects the DataEntry sheet of the Excel template file (so that unnecessary modifications do not occur, like deleting the buttons). The second Sub simply opens up a workbook with a given file path. And the third Sub closes that workbook. The breakdown will be provided button-by-button in the following subsections.

Import from Part 1 Source file button
This button is linked to Module 2. This module has 505 lines of code. The code in this module is quite straightforward. The Sub from which the code is compiling is called Sub Main. It first copies the data from the Part 1 Source.xlsx Excel file (located in The Dream). It copies the data in the sheet titled "DataEntry" in the source file over to a sheet titled "DataEntry" in the Excel template file. Similarly, it does the same for copying data from the "MiscEntry" sheet in the source file to the "MiscEntry" sheet in the Excel template file.

After, it properly formats all the column headers and "pretty-fies" the tables. This is done through a Sub in Module 2 called MakingPretty. Then, a new column is added called TEMP (DEF ISA+15C) which has values calculated depending on the values of elevation of the airfields. This column's values is calculated and filled by a Sub in Module 2 called ISAfifteen. Finally, the text values in the columns that are in a number format, are converted to cells with "number" type. This is done by a Sub in Module 2 called TextToNumber.

Confirm temperatures button
This button is linked to Module 9. This module has 79 lines of code. The code in this module is quite straightforward. The Sub from which the code is compiling is called Sub Main. By clicking this button, the temperatures in the TEMP (DEF ISA+15C) column are copied and sent to the Source.xlsx Excel file in Part 1 of The Dream. Thus, the Python script can read off from that source file and determine whether a user has clicked on the Confirm temperatures button. The flexibility this offers is quite profound, as users are able to choose the temperatures at which APG eventually analyzes takeoff performance from the airfields. There are no important Subs in this module to discuss.

Import from Part 2 Source file button
This button is linked to Module 7. This module has 255 lines of code. The code in this module is quite straightforward. The Sub from which the code is compiling is called Sub Main. It functions very similarly to the Import from Part 1 Source file button. It first copies the data from the Part 2 Source.xlsx Excel file (located in The Dream).

Then, if the user had chosen to analyze wet runway performance, it quickly renames the indices of the table. For example, consider that the indices of the table copied would have been "KASE" and "KASE" for the one airfield of Aspen. The Sub in Module 7 called DryandWet renames these indices as "KASE-DRY' and "KASE-WET". After, it properly formats all the column headers and "pretty-fies" the tables. This is done through a Sub in Module 7 called MakingPretty.

Range map button
This button is linked to Module 16. This module has 443 lines of code. The Sub from which the code is compiling is called Sub Main. This Main Sub works on pivoting the TOW table into another table into which Sales Analysts can input some details (like payload and speed). Then this table is formatted in the same way as the other tables, with a Sub called MakingPretty.

Then, certain cell ranges have assigned data validation restrictions. For example, the fuel reserve cell is a drop down list of 3 possible values (NBAA/IFR, EU OPS 3, and EU OPS 5). Similarly, the speed column only allows numerical values to be entered into the cells, and the value of the number must be less than or equal to 1 (i.e. the speed cannot be greater than Mach 1, because business aviation aircraft are not supersonic jets). This data validation step with restricting cell values is done by a Function called DropDownLists.

Merge Unmerge button
This button is linked to Module 18. This module has 68 lines of code. The Sub from which the code is compiling is called Sub Macro20. This is a short Sub written to merge a group of cells (if they were initially unmerged) and to unmerge a group of cells (if they were initially merged). It uses a helper Function called val to determine whether the chosen group of cells is merged or unmerged.

Range Calculator RAPV button
This button is linked to Module 1. This module currently has 274 lines of code. The Sub from which the code is compiling is called Sub LinktoRAPV. This Sub links to a Sub called TestedLinkage which is on a module on the RAPV database called The Dream. From this The Dream module, with the main important Subs TestedLinkage and MaxRange, extra code is compiled and runs RAPV to produce text reports from which range values can be obtained. The Dream module on RAPV currently has 612 lines of code. This module (or chain of modules) is by far the most complicated, because it is not just reformatting and copying data, but actually running programs through MS Access databases.

Summary

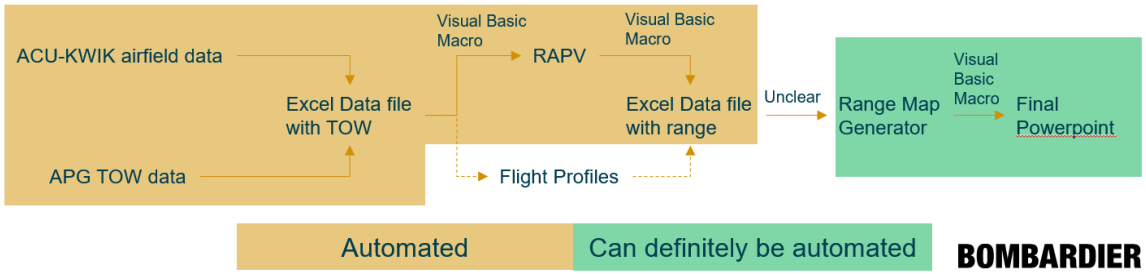
Throughout all the modules, subs, and functions, a total of 2236 lines of code was written. Most of the buttons had a simple purpose: to copy data from a source Excel sheet, and to reformat the data in a proper way. Buttons like the Confirm temperatures button copied data from the Excel template file back to the source Excel file. The Range Calculator RAPV button is an exception and involves some code that links to the RAPV database from which performance inputs are given and ranges are calculated with the help of a Fortran executable. It is beneficial to new developers to understand how to use RAPV and have different inputs to get proper range values of aircraft. It is only through this understanding that one can then alter the code and try to further work on automating this process.

GUIDELINES FOR NEXT STEPS

For future work, the following general advice is given to new developers:

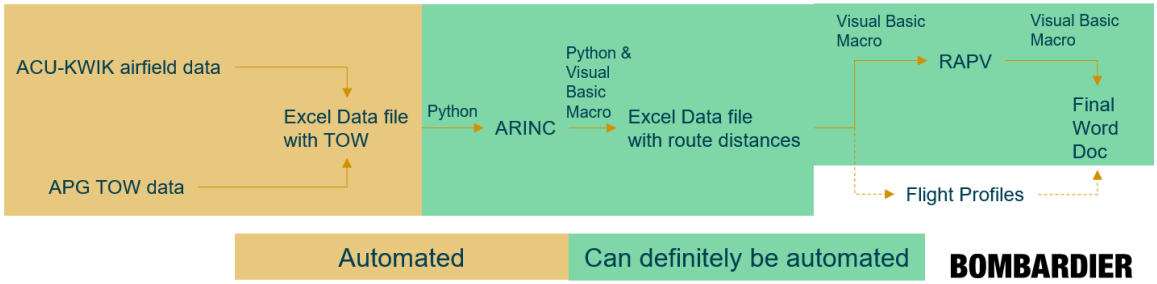
1. First get acquainted with the code (either Python or VBA) of what you are trying to alter/build up on.
2. Any part of the project you decide to work on, feel comfortable renaming variables, adding comments, implementing better code structure for more readability. I was not formally trained in Python or VBA, and I am sure that I am not ascribing to a proper industry standard for coding.
3. Refer to the flowcharts on inspiration of where the next automation steps can take place.

Let us refer back to the workflows of range maps and route studies, which shows us the current status of automation. For range maps, the following figure details the workflow involved.



Currently, the link from the Excel data file to RAPV and back to the Excel data file (with range values) is established but is not fully ready. For example, at the time of writing, the code simply passes in all aircraft in a table to RAPV, whereas in reality only some certain aircraft can be entered into RAPV for performance analysis. Additionally, no research has been done into whether a link can be established to Flight Profiles (for the non-RAPV aircraft). Also, no research has been done into whether the Excel file (with all the range values) can be automatically linked to the Range Map generator tool. However, I am quite positive from some research that the Range Map generator tool can definitely be linked to directly give a final Powerpoint file with the .png produced and with correct titles, footers, legends, and assumptions (currently, all of these are manually entered/altered each time a range map presentation is created).

Working on any of these sections would count as a solid project, and works toward the overall goal of automating a task that occurs daily. Moving on to route studies, the following figure shows the details of its workflow.



Not much has been done for automating route studies, apart from the initial couple of steps that The Dream entailed. Route studies are challenging in the sense that a Sales Analyst makes a lot of decisions, and inputs a lot of data to obtain a route study that gives Bombardier a performance edge. With decisions and variety of inputs comes increased complexity in automation. However, there is still a lot of scope to automate for route studies. For example, the process of obtaining route distances from ARINC is quite menial. A project for future developers can be to automate this with web automation yet again! Python's Selenium library is pretty reliable, and some colleagues mention Scrappy might be a good alternative. This could be linked to Excel macros coded in VBA to automatically obtain route distances between any two airfields. Then, a link can be established to RAPV (much like the previous link to RAPV to obtain ranges of aircraft), followed up with automatically obtaining a final word report of the route study. Again, the link through Flight Profiles to obtain route studies (for non-RAPV aircraft) has not been researched.

To reiterate, working on any of these sections would count as a solid project, and works toward the overall goal of automating a task that occurs daily. Another project for new developers can be fixing bugs in the sections already automated. Below is a table showing the stability of code of the different sections shown in the workflows.

ACU-KWIK airfield data	Very high stability
APG TOW data	High stability
Excel data file with TOW	Very high stability
Link from Excel to RAPV	Mediocre stability
Link from RAPV to Excel	High stability

Referring to the table can give new developers an idea on where bugs occur frequently. For example, currently, the web automation through APG has some trouble when the G450 aircraft is chosen. When the user chooses this aircraft, the code breaks. Also, there are several wait times within the web automation, which may mean that certain takeoff conditions are not chosen (e.g. APU ON is not switched to APU OFF). Nevertheless, overall the code for APG TOW data (i.e. Part 2 of Python) is pretty stable and works almost all the time.

The ACU-KWIK airfield data (i.e. Part 1 of Python) is extremely stable and seemingly does not break. The Excel file overall has very stable macros, because all it's doing is copying data and reformatting it. The link from the Excel file to RAPV is semi-stable. The underlying link is established, but more work needs to be done on making sure the

code does not break. The link from the RAPV module back to the Excel file is stable, with all of the functions in that process involving reading from RAPV text files and obtaining the range values to put into the Excel sheet.

CONCLUSION

Overall, this project was a lot of fun to work with! With a total of 5000+ lines of code written, and a solid foundation from which to work from, I am sure complete task automation is in the near future for the Sales Engineering team. This manual should hopefully answer all the questions that a new developer has. I have tried to include visual aids to ease reading and improve retention from the manual. If you have any general comments/concerns, please feel free to contact me (contact details in the last section).

FAQ

Q. How do I install the program on a new computer or on a laptop?
A. The project is hosted on the Sales Engineering Sharepoint under Performance tools. Users need access to this Sharepoint to extract the zipped folder and download the tool. The specific instructions of downloading the tool itself is explained in detail in the Instruction Manual of The Dream.

Q. How do I access the code and make changes to the Python scripts?
A. All of the code is within the zipped folder of The Dream too. I suggest reading up about what pyinstaller does. Folders like dist, and __pycache__, and files like main.spec are all generated by the pyinstaller. Everything else is relevant to the code, and can be compiled on any Python compiler.

Q. How do I access the code and make changes to the VBA code?
A. The template Excel file on Sharepoint currently is protected, meaning that you are not able to access the VBA code. This is done to prevent altering the structure of the sheet and prevent accidental deletion of buttons. To obtain the password, please contact the author of the project and I will be happy to share the password with new developers! Author contact information is provided in the last section of this document, and is also provided in the Instruction Manual.

Q. What were the steps taken to go from a Python script to a fully-distributable package?
A. First, use pyinstaller --onedir option to create an executable of your main Python script (written and tested on your personal laptop). Then copy all the files and folders into one primary folder on a USB. With the USB, transfer that primary folder over to your work laptop. Try running the .exe file to see if it works. The hardest part to debug are file paths, because .exe file executions have different file path references compared to normal Python script executions. Once your file paths are debugged and working on your work laptop, make sure all external dependencies are fulfilled. On my personal laptop, I had Java installed and pyinstaller does not wrap such dependencies within its files and folders. So, on my work laptop I downloaded Java and put it within the primary folder, and set a PATH variable so that Java is functional. There were other nuances in making sure external dependencies like chromedriver.exe was the same version as my browser's Chrome version (because my personal laptop and work laptop had different Chrome version, the code broke). Once everything works, and your .exe file runs on your work laptop fine, you are ready to zip the primary folder and upload it to Sharepoint under The Dream! Now everyone can unzip that folder, and run the .exe and it should work.

Q. If I make changes to the code, and upload a new zipped folder, should I delete the previous one? Is there anything else to do?
A. Yes, and yes! Whenever you make changes to the code, and end up having a new primary folder, make sure to change the Version Info.txt file in the folder. You

can update the version number, and update the date of that version number. Then, on Sharepoint, delete the previous zip folder in The Dream. Zip your primary folder that encompasses the new version of The Dream and upload it to The Dream on Sharepoint. The last step is to make sure that you update the Latest Version.txt file as well on Sharepoint. You will need to put the same details as you put in the Version Info.txt file of the newest version. This way, users will automatically be notified that they aren't using the latest version of The Dream because their local Version Info.txt file of a previous version of The Dream will be different from the Latest Version.txt file on Sharepoint. When users download and extract the newest zipped folder, they will no longer receive that message because the local Version Info.txt file now holds the same information as the Latest Version.txt file on Sharepoint.

AUTHOR

The project was undertaken by Siddharth Raghavan. All instruction manuals, technical documents, Visual Basic code for Excel, and underlying Python executable files are written by me. Please feel free to contact me for any questions you may have concerning the project. I will be happy to answer them if I can, even long after my internship ends (end of August 2020).

Below are some ways to contact me:

Gmail: mowglu@gmail.com
McGill email: Siddharth.raghavan@mail.mcgill.ca
Phone number: (438)-346-5135
Facebook profile: https://www.facebook.com/siddharth.raghavan.7/
LinkedIn profile: https://www.linkedin.com/in/siddharth-raghavan/
Github profile: https://github.com/mowglu?tab=repositories