# case_1.1

April 3, 2020

# 1 Pre-Case Setup: Python, Jupyter Notebook, Git

## 1.1 Python and Jupyter

Python is a general purpose programming language that allows for both simple and complex data analysis. Python is incredibly versatile, allowing analysts, consultants, engineers, and managers to obtain and analyze information for insightful decision-making.

The Jupyter Notebook is an open-source web application that allows for Python code development. Jupyter further allows for inline plotting and provides useful mechanisms for viewing data that make it an excellent resource for a variety of projects and disciplines.

The following section will outline how to install and begin working with Python and Juypter.

## 1.2 Setting up the Python Environment

Instruction guides for Windows and macOS are included below. Follow the one that corresponds with your operating system.

### 1.2.1 Windows Install:

Step 1: Open browser and go to https://www.anaconda.com/distribution/

Step 2: Click on ”Windows” and then ”Download” for Python 3.7 64-bit installer

Step 3: Run the downloaded file found in the downloads section from Step 2

Step 4: Click through the install prompts

Step 5: Go to menu (or Windows Explorer), find the Anaconda3 folder, and double-click to run

### 1.2.2 macOS Install:

Step 1: Open browser and go to https://www.anaconda.com/distribution/

Step 2: Click on ”macOS” and then ”Download” for Python 3.7 64-bit installer

Step 3: Run the downloaded file found in the downloads section from Step 2

Step 4: Click through the install prompts

Step 5: In Finder (or Launchpad), browse to the Anaconda3 folder to find the Jupyter program, and double-click to run

## 1.3 File Management with Python and Jupyter

It is common practice to have a main folder where all projects will be located (e.g. ”jupyter_research”). The following are guidelines you can use for Python projects to help keep your code organized and accessible:

1. Create subfolders for each Jupyter-related project
2. Group related .ipynb files together in the same folder
3. It can be useful to create a ”Data” folder within individual project folders if using a large number of related data files

You should now be set up and ready to begin coding in Python!

## 1.4 Setting up Git, GitHub, and cloning a repository

Git is a free and open source distributed version-control system. It is very useful for both simple software projects and large multi-functional projects that span thousands of files. The main website for Git documentation is https://git-scm.com/. We will go over a brief introduction to Git here to get up and running with the software, in addition to understanding how its version control system operates. We'll create a new repository and also cover how to clone a repository from GitHub. GitHub is an online platform that hosts repositories for users to interact with on their own projects and collaborative projects.

The following steps outline how to get started with Git, sign up for a GitHub account, create a repository, and clone a repository to your computer:

Step 1: Open your browser and go to https://git-scm.com/downloads and download Git for your operating system

Step 2: Create a new repository by browsing to the folder on your computer that you'd like to work under. From this folder, run the command: git init

Step 3: Open your browser and go to https://github.com/ and sign up for a Github account

Step 4: On GitHub, navigate to the main page of the repository you are interested in using. Note that when you create a repository on GitHub, it is a remote repository. We will clone a repository to create a local copy on your computer and we can then sync between the two locations when needed. Hence, we will have a local copy of the repository and a remote copy of the repository.

Step 5: Under the repository name on Github, click clone or download

Step 6: In the Clone with HTTPs section, copy the clone URL link

Step 7: Open your command line and navigate to the current working directory to the location where you want the cloned directory to be made. Type and run the command: git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY

You should have now created your local clone repository. Let's cover some of the utilities available with Git.

### 1.5 Understanding the Git workflow

Git is a distributed version-control system. The Git workflow is built upon three objects managed by Git: 1. Your working directory 2. The index (staging area) 3. The HEAD (points to your last commit)

The central workflow with Git is to locally edit files on your staging area (updating the index) and commit the changes to your working directory (therefore updating the HEAD) when you are ready to save this version of your work to your repository. While you are editing the files in the staging area, you can of course save these files as you progress, however the files on your working directory will not have changed since you need to commit the changes. Hence, you essentially have two versions of files, one that you are editing, and one that is a snapshot from the previous commit.

Let's go through a simple example of committing a change to some text file `hello.txt`.

Step 1: Create a text file hello.txt on your local computer in the folder you will be using Git from.

Step 2: Edit the file to add one line to the text file. Say you add "Hello World" as the first line. Save the file.

Step 3: Update your staging area (i.e. update the index). To do this, use the command: `git add hello.txt`

(If you have multiple files and you'd like to add all the files with changes in the folder to the staging area you can use the command: `git add *`)

Step 4: To commit the changes on the staging area to the working directory, use the command: `git commit -m "Commit message"`

(You should update the commit message to provide useful information. Do not neglect this as it can be very useful to have informative commit messages when debugging unexpected issues with your files)

Step 5: Step 4 commits the file to the HEAD, but the remote repository will not update, so you've updated the HEAD of your local working copy. To push the changes to your remote repository, use the command: `git push origin master`

Furthermore, useful commands to check the status and history of your repository include: 1. Command to check repository status: `git status` 2. Command to check history of repository: `git log`

This covers the basics of Git. While it takes considerable practice to master, we've covered the cloning, adding, committing, and pushing features that are the backbone of the Git version control software.

We're ready to begin the case.

# 2 Identifying Expansion Opportunities for Luxury Commercial Airline Flights

## 2.1 Case Introduction

**Business Context.** You are an employee for GrowthAir, a growing commercial airline company. In the past few years, GrowthAir has expanded luxury flight services to locations across the globe. Following your team's excellent performance in identifying new business opportunities last year, you have been tasked with identifying the top countries to further expand GrowthAir's luxury flight service.

**Business Problem.** Your manager has asked you to answer the following question: "In which countries should GrowthAir expand its luxury flight service?"

**Analytical Context.** The relevant data is a series of success estimates (i.e. probabilities of success) that your internal marketing research teams have come up with. Using your ability to conduct data analysis in Python, you will embark on summarizing the available success estimates to produce a concise recommendation to your boss.

## 2.2 Fundamentals of Python

Python is an interpreted, high-level general programming language that was first released in 1991. Python allows users to easily manipulate data and store values in what are known as objects. Everything in Python is an object and has a type. For example, if a user aims to store the integer 5 in an object named `my_int`, this can be accomplished by the statement, `my_int = 5`. This statement tells Python to assign the integer value of 5 to the **variable** `my_int` (called a variable here because it can change value). `my_int` is a Python object, and has type `int`.

Similar to how Excel distinguishes different data types (such as Text, Number, Currency, Scientific), Python offers a variety of data types. Here are a few common data types:

1. Integers, type `int`: `my_int = 1`
2. Float type `float`: `my_float = 25.5`
3. Strings, type `str`: `my_string = 'Hello'`

Here we see (1) integers and (2) floats store numeric data. The difference between the two is that floats store decimal variables, whereas the integer type can only store integer variables. (3) is the string type. Strings are used to store textual data in Python. This case will use string variables to store country names. They are often used to store identifiers such as person names, city names, and more.

There are other data types available in Python; however, these are the three fundamental types that you will see across almost every Python program. Always keep in mind that **every** object in Python has a type.

Now that we've covered the fundamentals of Python, let's take a look at GrowthAir's proprietary company data on country success estimates.

## 2.3 Exploring company data on success estimates

Let's take a look at a common data structure used to hold your company's proprietary data on estimates of probability of success for global expansion projects by country. The `success_estimates` variable below is a Python dictionary, which is being assigned certain data using the `'='` assignment operator. Each estimate here is a number (float) between 0 and 1, inclusive, which represents the probability that expanding to that country will be successful.

Python's dictionary type stores key-value pairs that allow users to quickly access information for a particular key. By specifying a key, the user can return the value corresponding to the given key. Python's syntax for dictionaries uses curly braces {},

```
user_dictionary = {'Key1':Value1, 'Key2':Value2, 'Key3':Value3}
```

The `success_estimates` dictionary has keys which are strings, and values which are of type list. A list is an incredibly useful data structure in Python that can store any number of Python objects, and are denoted by the use of square brackets `[]`. In `success_estimates` below, the list contains float types. Lists are versatile and can be expanded by adding new elements to the end of the list (the right-most side is considered the end of the list). Moreover, list elements (i.e. the objects in the list) can be accessed easily using integer indices. Interestingly, lists can also store other lists (called a lists of lists). This makes them a powerful tool for holding complex data sets.

Let's take a look at the `success_estimates` data:

```
[1]: # Data on probability of expansion success by country estimates
success_estimates = {'Australia': [0.6, 0.33, 0.11, 0.14],
                     'France': [0.66, 0.78, 0.98, 0.2],
                     'Italy': [0.6],
                     'Brazil': [0.22, 0.22, 0.43],
                     'USA': [0.2, 0.5, 0.3],
                     'England': [0.45],
                     'Canada': [0.25, 0.3],
                     'Argentina': [0.22],
                     'Greece': [0.45, 0.66, 0.75, 0.99, 0.15, 0.66],
                     'Morocco': [0.29],
                     'Tunisia': [0.68, 0.56],
                     'Egypt': [0.99],
                     'Jamaica': [0.61, 0.65, 0.71],
                     'Switzerland': [0.73, 0.86, 0.84, 0.51, 0.99],
                     'Germany': [0.45, 0.49, 0.36]}
```

Python easily allows you to print the elements stored in any variable to the screen using the `print()` statement:

```
[2]: print(success_estimates)
```

```
{'Australia': [0.6, 0.33, 0.11, 0.14], 'France': [0.66, 0.78, 0.98, 0.2],
'Italy': [0.6], 'Brazil': [0.22, 0.22, 0.43], 'USA': [0.2, 0.5, 0.3], 'England':
[0.45], 'Canada': [0.25, 0.3], 'Argentina': [0.22], 'Greece': [0.45, 0.66, 0.75,
0.99, 0.15, 0.66], 'Morocco': [0.29], 'Tunisia': [0.68, 0.56], 'Egypt': [0.99],
'Jamaica': [0.61, 0.65, 0.71], 'Switzerland': [0.73, 0.86, 0.84, 0.51, 0.99],
'Germany': [0.45, 0.49, 0.36]}
```

Notice the re-ordering of the dictonary elements when we print compared to the order in which we originally defined the dictionary. This is a key aspect of dictionary data types – they are unordered! (This is very different compared to list data types, which are ordered. More on this later.)

Now intuitively, we would like to recommend that the business put effort into the country with the highest success estimate. But what does this mean when there are multiple success estimates for some countries, and only one for others? We will explore this next.

## 2.4 Interacting with dictionaries and lists

Taking a careful look at the `success_estimates` dictionary, you notice some countries only have one success estimate, while others have many. For example, England has only one estimate contained in its list [0.45], while Jamaica has three estimates contained in its list [0.61, 0.65, 0.71]. Let's zoom in on Jamaica and take a look at some summary statistics of the estimates.

In Python, the dictionary type has built-in methods (functions, which we will discuss later) to access the dictionary keys and values. These methods are called by typing `.keys()` or `.values()` after the dictonary object. We will change the return type of calling `.keys()` and `.values()` to a list by using the `list()` method.

[3]:
```python
# Look at the keys...
list(success_estimates.keys())
```

[3]:
```
['Australia',
 'France',
 'Italy',
 'Brazil',
 'USA',
 'England',
 'Canada',
 'Argentina',
 'Greece',
 'Morocco',
 'Tunisia',
 'Egypt',
 'Jamaica',
 'Switzerland',
 'Germany']
```

[4]:
```python
# ...and their corresponding values
list(success_estimates.values())
```

[4]:
```
[[0.6, 0.33, 0.11, 0.14],
 [0.66, 0.78, 0.98, 0.2],
 [0.6],
 [0.22, 0.22, 0.43],
 [0.2, 0.5, 0.3],
 [0.45],
 [0.25, 0.3],
 [0.22],
 [0.45, 0.66, 0.75, 0.99, 0.15, 0.66],
 [0.29],
 [0.68, 0.56],
 [0.99],
 [0.61, 0.65, 0.71],
 [0.73, 0.86, 0.84, 0.51, 0.99],
 [0.45, 0.49, 0.36]]
```

We will make use of the access to keys and values of a dictionary later in the case when comparing across numerous countries' estimates. For now, just remember that you can access a dictionary's full list of keys or values simply by calling built-in methods.

We'd also like to check if a country name is one the keys in the dictionary. Python allows us to check if a key is in a dictionary through the use of the `in` keyword. The statement `key in dictionary` will return a boolean type of `True` if the key is one of the keys in the dictionary and `False` otherwise. Let's take a look at how this works.

[5]:
```python
print('Checking if Morocco key is present:')
print('Morocco' in success_estimates)

print('Checking if Japan key is present:')
print('Japan' in success_estimates)
```

```
Checking if Morocco key is present:
True
Checking if Japan key is present:
False
```

We'd now like to access the value corresponding to a specific key in the `success_estimates` dictionary. Simply type the value name in square brackets adjacent to the dictionary name. For example, `success_estimates['Jamaica']` will return Jamaica's list of estimates:

[6]:
```python
success_estimates['Jamaica']
```

[6]:
```
[0.61, 0.65, 0.71]
```

If you would like to store the result in a variable to be used later, use the assignment operator `'='`:

[7]:
```python
jamaica_list = success_estimates['Jamaica']
```

You can then view the contents of the list via the `print()` method:

[8]:
```python
print(jamaica_list)
```

```
[0.61, 0.65, 0.71]
```

Here, you'll see that the order of the elements in the Jamaica list is the same as what was originally defined above. This is because lists are ordered objects. In fact, you can access elements of a list by an index. In Python, indices start at 0 (for the first element of a given list) and increment by 1 for each successive element. For example, let's print each element of the Jamaica list:

[9]:
```
# Each successive print statement will print on a new line
print(jamaica_list[0]) # prints the first element of the list
print(jamaica_list[1]) # prints the second element of the list
print(jamaica_list[2]) # prints the third element of the list
```

```
0.61
0.65
0.71
```

Python also offers a simple way to determine the length of a list: the `len()` method. We expect the length of `jamaica_list` to be 3 since it has three elements:

[10]:
```
len(jamaica_list) # returns the length of the list
```

[10]:
```
3
```

### 2.4.1 Exercise 1:

Print the length of the success estimate lists for France, Greece, and Morocco.

**Answer.** One possible solution is shown below:

[11]:
```
print('Number of estimates for France:')
print(len(success_estimates['France']))

print('Number of estimates for Greece:')
print(len(success_estimates['Greece']))

print('Number of estimates for Morocco:')
print(len(success_estimates['Morocco']))
```

```
Number of estimates for France:
4
Number of estimates for Greece:
6
Number of estimates for Morocco:
1
```

### 2.4.2 Exercise 2:

Which of the following would be useful to store project success estimates if they were available at a regional level instead of a country level?

(a) List

(b) Dictionary

(c) Float

(d) String

**Answer.** (b).

(a) Python lists store data in a an ordered manner, where each element of a list can be of any type. There is no mapping between keys and values in Python lists, rather list elements are accessed using integer indices. Lists are commonly used to group related data together for further processing. In this case, the regional level data would still consist of key-value pairs (where the keys are the region names), so lists are not ideal.

(b) B is the correct answer since dictionaries use a key-value pair to represent data. They are fast to access and are unordered. Dictionaries are especially useful when data is being accessed by a commonly used identifier, such as country names used in this case, or region names as indicated in this example.

(c) Floats are used to represent numeric data, hence are not used for mapping keys to values.

(d) Strings are used to represent textual data. While they are often used as the keys in a dictionary, the strings themselves are not built for storing key-value pairs of any kind.

Now that we're familiar with using lists and know that lists are ordered data structures while dictionaries are unordered data structures, let's begin to compare success estimates across countries.

## 2.5 Calculating a country-specific average success estimate

Continuing our analysis on Jamaica, the list contains three numbers, [0.61, 0.65, 0.71]. Recall these number are of type `float` in Python, which stores numeric decimal values. One logical way to summarize these estimates so that they can be compared across countries is to use the arithmetic average. Let's use basic arthimetic operators to calculate the average success estimate for Jamaica, storing the result in a new variable `avg_jamaica`:

[12]:
```python
avg_jamaica = (0.61 + 0.65 + 0.71) / 3
print(avg_jamaica)
```

```
0.6566666666666666
```

We see the average probability of success estimate for Jamaica is approximately 0.657. However, we produced this estimate by hand-coding the values. If we were to do this for every country, it would take quite a long time. So we'd like to use a more automated way of producing the average.

To produce an average we can utilize a function. Functions operate on data and variables in Python to perform a desired action. Functions may have both inputs and outputs, just like familiar mathematical operators like addition, subtraction, multiplication, and division (which each have two inputs and one output). While functions in Python may still be for a mathematical purpose, such as squaring an integer, Python allows for more abstract function behaviour, such as printing to the screen. In this case, the `print()` function will print its input to the screen.

Let's use Python's built-in mathematical functions `sum()`, `min()`, and `max()` to calculate Jamaica's average success estimate, minimum success estimate, and maximum success estimate, respectively:

[13]:
```python
country_name = 'Jamaica'
jamaica_list = success_estimates[country_name] # list of the estimates for␣
 ↪Jamaica
print(jamaica_list)
```

```
[0.61, 0.65, 0.71]
```

[14]:
```python
avg_jamaica = sum(jamaica_list) / len(jamaica_list)
min_jamaica = min(jamaica_list)
max_jamaica = max(jamaica_list)
print("Country:",country_name,", Average:",avg_jamaica)
print("Country:",country_name,", Min:",min_jamaica)
print("Country:",country_name,", Max:",max_jamaica)
```

```
Country: Jamaica , Average: 0.6566666666666666
Country: Jamaica , Min: 0.61
Country: Jamaica , Max: 0.71
```

As expected, we get the same average result of approximately 0.657. Note that we could also have rounded the results to two decimal places using the `round()` method. This can improve readability.

[15]:
```python
avg_jamaica = round(sum(jamaica_list) / len(jamaica_list),2)
min_jamaica = round(min(jamaica_list),2)
max_jamaica = round(max(jamaica_list),2)
print("Country:",country_name,", Average:",avg_jamaica)
print("Country:",country_name,", Min:",min_jamaica)
print("Country:",country_name,", Max:",max_jamaica)
```

```
Country: Jamaica , Average: 0.66
Country: Jamaica , Min: 0.61
Country: Jamaica , Max: 0.71
```

Functions in Python are a very powerful tool to increase productivity and perform more complex tasks.

### 2.5.1 Exercise 3:

Write a script to calculate the average success for every country. Output (using `print()`) each country's average success estimate to the screen. The print statements should output each country on a new line, for example:

```
Country: France , Average: 0.655
Country: Brazil , Average: 0.29
```

[16]:
```python
# One possible solution
```

```
print("Country:",'France',", Average:",sum(success_estimates['France']) /␣
 ↪len(success_estimates['France']))
print("Country:",'Brazil',", Average:",sum(success_estimates['Brazil']) /␣
 ↪len(success_estimates['Brazil']))
print("Country:",'Argentina',", Average:",sum(success_estimates['Argentina']) /␣
 ↪len(success_estimates['Argentina']))
print("Country:",'Germany',", Average:",sum(success_estimates['Germany']) /␣
 ↪len(success_estimates['Germany']))
print("Country:",'Australia',", Average:",sum(success_estimates['Australia']) /␣
 ↪len(success_estimates['Australia']))
print("Country:",'Canada',", Average:",sum(success_estimates['Canada']) /␣
 ↪len(success_estimates['Canada']))
print("Country:",'Greece',", Average:",sum(success_estimates['Greece']) /␣
 ↪len(success_estimates['Greece']))
print("Country:",'USA',", Average:",sum(success_estimates['USA']) /␣
 ↪len(success_estimates['USA']))
print("Country:",'Switzerland',", Average:
 ↪",sum(success_estimates['Switzerland']) /␣
 ↪len(success_estimates['Switzerland']))
print("Country:",'Tunisia',", Average:",sum(success_estimates['Tunisia']) /
 ↪len(success_estimates['Tunisia']))
print("Country:",'Italy',", Average:",sum(success_estimates['Italy']) /␣
 ↪len(success_estimates['Italy']))
print("Country:",'Egypt',", Average:",sum(success_estimates['Egypt']) /␣
 ↪len(success_estimates['Egypt']))
print("Country:",'Jamaica',", Average:",sum(success_estimates['Jamaica']) /␣
 ↪len(success_estimates['Jamaica']))
print("Country:",'Morocco',", Average:",sum(success_estimates['Morocco']) /␣
 ↪len(success_estimates['Morocco']))
print("Country:",'England',", Average:",sum(success_estimates['England']) /␣
 ↪len(success_estimates['England']))
```

```
Country: France , Average: 0.655
Country: Brazil , Average: 0.29
Country: Argentina , Average: 0.22
Country: Germany , Average: 0.43333333333333333
Country: Australia , Average: 0.29500000000000004
Country: Canada , Average: 0.275
Country: Greece , Average: 0.61
Country: USA , Average: 0.3333333333333333
Country: Switzerland , Average: 0.7859999999999999
Country: Tunisia , Average: 0.6200000000000001
Country: Italy , Average: 0.6
Country: Egypt , Average: 0.99
Country: Jamaica , Average: 0.6566666666666666
Country: Morocco , Average: 0.29
```

```
Country: England , Average: 0.45
```

## 2.6 Systematically determine the average success estimate for all of the countries

The end goal of this analysis is a recommendation for where global expansion opportunities should be considered. To reach a conclusion, it'd be ideal to have the average success probability for each country.

To achieve this, we will use a control flow element in Python - the for loop. The `for` loop allows one to execute the same statements over and over again (i.e. looping). This saves a significant amount of time coding repetitive tasks and aids in code readability. The general structure of a for loop is:

```
for iterator_variable in some_sequence:
    statements(s)
```

The for loop iterates over `some_sequence` and performs `statements(s)` at each iteration. That is, at each iteration the `iterator_variable` is updated to the next value in `some_sequence`. As a concrete example, consider the loop:

```
for i in [1,2,3,4]:
    print(i*i)
```

Here, the for loop will print to the screen four times; that is it will print `1` on the first iteration of the loop, `4` on the second iteration, `9` on the third, and `16` on the fourth. Hence, the for loop statement will iterate over all the elements of the list `[1,2,3,4]`, and at each iteration it updates the iterator variable `i` to the next value in the list `[1,2,3,4]`.

Let's use a for loop on our country data by getting a list of all the keys in `success_estimates`:

[17]:
```
# Get all the keys from the success_estimates dictionary
country_name_list = list(success_estimates.keys())
print(country_name_list)
```

```
['Australia', 'France', 'Italy', 'Brazil', 'USA', 'England', 'Canada',
'Argentina', 'Greece', 'Morocco', 'Tunisia', 'Egypt', 'Jamaica', 'Switzerland',
'Germany']
```

Here we loop through all the elements in `country_name_list`, extract the corresponding value from `success_estimates` (which will be of type list), and subsequently take the mean of the list. Detailed printing will guide you through the for loop execution.

[18]:
```
# Loop through all countries and calculate their mean success estimate
for i in country_name_list:
    print('--Begin one iteration of loop--')
    print('Element of country_name_list, placeholder i = ' + i)
    print('Access value from dict success_estimates[i]: ', success_estimates[i])
    print('Average of list from success_estimates[i]: ',␣
 ↪sum(success_estimates[i]) / len(success_estimates[i]))
    print('--Go to next iteration of loop--')
```

```
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Australia
Access value from dict success_estimates[i]:  [0.6, 0.33, 0.11, 0.14]
Average of list from success_estimates[i]:  0.29500000000000004
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = France
Access value from dict success_estimates[i]:  [0.66, 0.78, 0.98, 0.2]
Average of list from success_estimates[i]:  0.655
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Italy
Access value from dict success_estimates[i]:  [0.6]
Average of list from success_estimates[i]:  0.6
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Brazil
Access value from dict success_estimates[i]:  [0.22, 0.22, 0.43]
Average of list from success_estimates[i]:  0.29
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = USA
Access value from dict success_estimates[i]:  [0.2, 0.5, 0.3]
Average of list from success_estimates[i]:  0.3333333333333333
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = England
Access value from dict success_estimates[i]:  [0.45]
Average of list from success_estimates[i]:  0.45
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Canada
Access value from dict success_estimates[i]:  [0.25, 0.3]
Average of list from success_estimates[i]:  0.275
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Argentina
Access value from dict success_estimates[i]:  [0.22]
Average of list from success_estimates[i]:  0.22
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Greece
Access value from dict success_estimates[i]:  [0.45, 0.66, 0.75, 0.99, 0.15,
0.66]
Average of list from success_estimates[i]:  0.61
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Morocco
```

```
Access value from dict success_estimates[i]:  [0.29]
Average of list from success_estimates[i]:  0.29
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Tunisia
Access value from dict success_estimates[i]:  [0.68, 0.56]
Average of list from success_estimates[i]:  0.6200000000000001
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Egypt
Access value from dict success_estimates[i]:  [0.99]
Average of list from success_estimates[i]:  0.99
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Jamaica
Access value from dict success_estimates[i]:  [0.61, 0.65, 0.71]
Average of list from success_estimates[i]:  0.6566666666666666
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Switzerland
Access value from dict success_estimates[i]:  [0.73, 0.86, 0.84, 0.51, 0.99]
Average of list from success_estimates[i]:  0.7859999999999999
--Go to next iteration of loop--
--Begin one iteration of loop--
Element of country_name_list, placeholder i = Germany
Access value from dict success_estimates[i]:  [0.45, 0.49, 0.36]
Average of list from success_estimates[i]:  0.43333333333333333
--Go to next iteration of loop--
```

Let's take a closer look at the above `for` loop. The `country_name_list` has 15 countries which the `for` loop is iterating over. The `for` loop uses a placeholder variable, denoted `i` in this case, to store the element of `country_name_list` that each loop iteration corresponds to. Namely, for the first iteration of the `for` loop, `i = 'Brazil'`. For the second iteration, `i = 'Canada'`. And so on until the loop reaches the final element of `country_name_list`, which it then completes and exits the looping process.

Why is this looping process useful? Well, we've performed the same calculation statements 15 times while only writing the code once! Notice that for each iteration, the corresponding value from `success_estimates` is accessed, and the mean of the returned list is calculated. The `for` loop process also enhances code readability.

### 2.6.1 Exercise 4:

Write a for loop to instead calculate the minimum and maximum of each country's list of success estimates, printing each out consecutively as in the for loop example above.

**Answer.** One possible solution is shown below:

[19]:
```
for i in country_name_list:
    print('Country',i,', Min: ', min(success_estimates[i]))
    print('Country',i,', Max: ', max(success_estimates[i]))
```

```
Country Australia , Min:  0.11
Country Australia , Max:  0.6
Country France , Min:  0.2
Country France , Max:  0.98
Country Italy , Min:  0.6
Country Italy , Max:  0.6
Country Brazil , Min:  0.22
Country Brazil , Max:  0.43
Country USA , Min:  0.2
Country USA , Max:  0.5
Country England , Min:  0.45
Country England , Max:  0.45
Country Canada , Min:  0.25
Country Canada , Max:  0.3
Country Argentina , Min:  0.22
Country Argentina , Max:  0.22
Country Greece , Min:  0.15
Country Greece , Max:  0.99
Country Morocco , Min:  0.29
Country Morocco , Max:  0.29
Country Tunisia , Min:  0.56
Country Tunisia , Max:  0.68
Country Egypt , Min:  0.99
Country Egypt , Max:  0.99
Country Jamaica , Min:  0.61
Country Jamaica , Max:  0.71
Country Switzerland , Min:  0.51
Country Switzerland , Max:  0.99
Country Germany , Min:  0.36
Country Germany , Max:  0.49
```

### 2.6.2 Exercise 5:

Using the for loop, write code to determine the country with the largest range of success estimates (that is, the largest difference between the smallest and largest estimate for a country).

**Answer.** One possible solution is shown below:

[20]:
```
for i in country_name_list:
    country_range = max(success_estimates[i]) - min(success_estimates[i])
    print('Country: ', i, ", Range: ", country_range)

# Visualizally analyzing the output, we see Greece has the largest range
```

```
Country:  Australia , Range:  0.49
Country:  France , Range:  0.78
Country:  Italy , Range:  0.0
Country:  Brazil , Range:  0.21
Country:  USA , Range:  0.3
Country:  England , Range:  0.0
Country:  Canada , Range:  0.04999999999999999
Country:  Argentina , Range:  0.0
Country:  Greece , Range:  0.84
Country:  Morocco , Range:  0.0
Country:  Tunisia , Range:  0.12
Country:  Egypt , Range:  0.0
Country:  Jamaica , Range:  0.09999999999999998
Country:  Switzerland , Range:  0.48
Country:  Germany , Range:  0.13
```

## 2.7 Using list comprehensions to determine the number of estimates for each country

Moving forward, we are interested in knowing the number of success estimates available for each country. Python offers a concise way to achieve this goal through the use of list comprehensions.

List comprehensions allow one to concisely build a list. Let's take a look at how this works.

[21]:
```
key_name_list = [i for i in success_estimates] # loop over each item i in␣
↪success_estimates and put i in the list
key_name_list
```

[21]:
```
['Australia',
 'France',
 'Italy',
 'Brazil',
 'USA',
 'England',
 'Canada',
 'Argentina',
 'Greece',
 'Morocco',
 'Tunisia',
 'Egypt',
 'Jamaica',
 'Switzerland',
 'Germany']
```

Here we see that we've looped over each key of the dictionary success_estimates (hence each country), and extracted the country name, all in one line of code. We can also access the values of each key in success_estimates.

```
[22]: value_name_list = [success_estimates[i] for i in success_estimates] # loop over␣
       ↪each item i in success_estimates and put success_estimates[i] in the list
      value_name_list
```

```
[22]: [[0.6, 0.33, 0.11, 0.14],
       [0.66, 0.78, 0.98, 0.2],
       [0.6],
       [0.22, 0.22, 0.43],
       [0.2, 0.5, 0.3],
       [0.45],
       [0.25, 0.3],
       [0.22],
       [0.45, 0.66, 0.75, 0.99, 0.15, 0.66],
       [0.29],
       [0.68, 0.56],
       [0.99],
       [0.61, 0.65, 0.71],
       [0.73, 0.86, 0.84, 0.51, 0.99],
       [0.45, 0.49, 0.36]]
```

In the list comprehension above, each value of `i` is a country name and the value is returned when `success_estimates[i]` is called. We see the list comprehension is an effective and concise way to write a for loop that creates a list.

We can the use this to quickly determine how many success estimates are available for each country.

```
[23]: # Number of estimates available for each country
      [[i,len(success_estimates[i])] for i in success_estimates]
```

```
[23]: [['Australia', 4],
       ['France', 4],
       ['Italy', 1],
       ['Brazil', 3],
       ['USA', 3],
       ['England', 1],
       ['Canada', 2],
       ['Argentina', 1],
       ['Greece', 6],
       ['Morocco', 1],
       ['Tunisia', 2],
       ['Egypt', 1],
       ['Jamaica', 3],
       ['Switzerland', 5],
       ['Germany', 3]]
```

### 2.7.1 Exercise 6:

Using list comprehensions, write a script to create a list of lists called `sum_squares_list`, where each element of the list is a two-item list [country name, value]. The value item in the list should be the sum of squares of that country's success estimates. For example, one element of `sum_squares_list` should be for Jamaica, where the two-item list is [Jamaica, 1.2987] (since 1.2987 = 0.61^2 + 0.65^2 + 0.71^2).

**Answer.** One possible answer is shown below:

```
[24]: # One possible solution
      sum_squares_list = [[i, sum([j**2 for j in success_estimates[i]])] for i in␣
       ↪success_estimates]
      sum_squares_list
```

```
[24]: [['Australia', 0.5005999999999999],
       ['France', 2.0444],
       ['Italy', 0.36],
       ['Brazil', 0.28169999999999995],
       ['USA', 0.38],
       ['England', 0.2025],
       ['Canada', 0.1525],
       ['Argentina', 0.0484],
       ['Greece', 2.6388],
       ['Morocco', 0.0841],
       ['Tunisia', 0.7760000000000001],
       ['Egypt', 0.9801],
       ['Jamaica', 1.2987],
       ['Switzerland', 3.2183],
       ['Germany', 0.5722]]
```

### 2.7.2 Exercise 7:

We'd like to determine the spread around the mean success estimate for each country. Using list comprehensions, write a script that subtracts the mean success estimate for a given country from each success estimate for that country. Store the results in a list named `removed_mean_list`. Round values to two decimal places. Your output should produce the following list of lists:

```
[['Australia', [0.3, 0.03, -0.19, -0.16]],
 ['France', [0.01, 0.12, 0.32, -0.46]],
 ['Italy', [0.0]],
 ['Brazil', [-0.07, -0.07, 0.14]],
 ['USA', [-0.13, 0.17, -0.03]],
 ['England', [0.0]],
 ['Canada', [-0.03, 0.02]],
 ['Argentina', [0.0]],
 ['Greece', [-0.16, 0.05, 0.14, 0.38, -0.46, 0.05]],
 ['Morocco', [0.0]],
 ['Tunisia', [0.06, -0.06]],
 ['Egypt', [0.0]],
 ['Jamaica', [-0.05, -0.01, 0.05]],
 ['Switzerland', [-0.06, 0.07, 0.05, -0.28, 0.2]],
 ['Germany', [0.02, 0.06, -0.07]]]
```

**Answer.** One possible answer is shown below:

```
[25]: # One possible solution
removed_mean_list = [[i, [round(j - sum(success_estimates[i])/
 ↪len(success_estimates[i]),2) for j in success_estimates[i]]] for i in␣
 ↪success_estimates]
removed_mean_list
```

```
[25]: [['Australia', [0.3, 0.03, -0.19, -0.16]],
 ['France', [0.01, 0.12, 0.32, -0.46]],
 ['Italy', [0.0]],
 ['Brazil', [-0.07, -0.07, 0.14]],
 ['USA', [-0.13, 0.17, -0.03]],
 ['England', [0.0]],
 ['Canada', [-0.03, 0.02]],
 ['Argentina', [0.0]],
 ['Greece', [-0.16, 0.05, 0.14, 0.38, -0.46, 0.05]],
 ['Morocco', [0.0]],
 ['Tunisia', [0.06, -0.06]],
 ['Egypt', [0.0]],
 ['Jamaica', [-0.05, -0.01, 0.05]],
 ['Switzerland', [-0.06, 0.07, 0.05, -0.28, 0.2]],
 ['Germany', [0.02, 0.06, -0.07]]]
```

## 2.8 Reflecting on the country-specific mean success estimate

Based on the above analysis, we see the mean country success estimates vary widely, from the lowest, Canada = 0.275, to the highest, Egypt = 0.99. However, notice that Egypt's mean is calculated from 1 success estimate. Are we confident in trusting a single estimate as a proxy for the average success estimate?

Given that the global expansion project will utilize valuable company resources, we decide it is best to restrict our analysis to countries that have two or more success estimates. To accomplish this task, we will use a control structure in Python known as the if...elif...else statement. The general structure follows.

```
if test_expression_1:
    block1_statement(s)
elif test_expression_2:
    block2_statement2(s)
else:
    block3_statement(s)
```

Here, `test_expression_1` and `test_expression_2` must evaluate to `True` or `False`, a Python boolean type. The boolean type is associated with variables that are either `True` or `False`.

If `test_expression_1` is True, `block1_statement(s)` will execute and the other block statements will not. If `test_expression_1` is False yet `test_expression_2` is True, then `block2_statement2(s)` will execute and the others will not. Finally, if `test_expression_1` and `test_expression_2` are both False, then the else section's `block3_statement(s)` will execute. This conditional structure of an if statement allows one to control the flow of Python code.

Let's use this to filter out the countries that only have one success estimate.

## 2.9 Selecting only multi-observation countries for global expansion potential

We will use the if statement above to remove countries with less than one success estimate. For convenience of viewing the result, we will store the mean estimates for each country in a new dictionary `country_means`.

[26]:
```
# Get a list of all the country names
country_name_list = list(success_estimates.keys())

# Create an empty dictionary to hold country mean estimates
country_means = {}

# Loop through all countries and calculate their mean success estimate
for i in country_name_list:
    list_country_estimates = success_estimates[i] # list of estimates for a␣
 ↪country

    # if more than one country estimate, then record the mean estimate,␣
 ↪otherwise go to next loop iteration
    if len(success_estimates[i]) > 1:
        country_mean_value = sum(list_country_estimates) /␣
 ↪len(list_country_estimates)
        country_means[i] =  country_mean_value # insert country mean value into␣
 ↪dict using country name as key
```

Let's format our results, modifying the string output to the screen to use 2 decimals when printing the float type. This is accomplished using the string type's `.format()` functionality. The `{0:s}` and `{1:.2f}` in the string indicate to the `.format()` method to format the first variable it receives as input as a string and replace the `{0:s}` placeholder, and to format the second variable it receives as input as a 2-decimal float and replace the `{1:.2f}` placeholder.

With this formatting, the `country_key` variable will be displayed as a string in place of `{0:s}`, while the `country_means[country_key]` variable will be displayed as a 2-decimal float in place of `{1:.2f}`. This advanced string formatting approach is useful to improve the clarity of the results.

[27]:
```
# Nicely format the result for printing to the screen
for country_key in country_means:
    print("Country: {0:s}, Avg Success Estimate: {1:.2f}".format(country_key,␣
↪country_means[country_key]))
```

```
Country: Australia, Avg Success Estimate: 0.30
Country: France, Avg Success Estimate: 0.66
Country: Brazil, Avg Success Estimate: 0.29
Country: USA, Avg Success Estimate: 0.33
Country: Canada, Avg Success Estimate: 0.28
Country: Greece, Avg Success Estimate: 0.61
Country: Tunisia, Avg Success Estimate: 0.62
Country: Jamaica, Avg Success Estimate: 0.66
Country: Switzerland, Avg Success Estimate: 0.79
Country: Germany, Avg Success Estimate: 0.43
```

Observing the resulting country means, we notice the country with the largest mean success estimate is Switzerland at 0.79, while the lowest mean success estimate is Canada at 0.28.

### 2.9.1 Exercise 8:

After reviewing company policy on statistical procedures, you notice the company recommends that all estimates (averages, minimums, maximums) must have at least three values contributing to the summary statistic. Write a for loop and use the `if` statement structure to select and print the average success estimates for the countries satisfying this policy. If the country does not satisfy the policy, print the country name and `"*Does not meet company policy*"`. Each country should appear on a new line.

**Answer.** One possible answer is shown below:

[28]:
```
# Get a list of all the country names
country_name_list = list(success_estimates.keys())

# Create an empty dictionary to hold country mean estimates
country_means = {}

# Loop through all countries and calculate their mean success estimate
for i in country_name_list:
    list_country_estimates = success_estimates[i] # list of estimates for a␣
↪country

    # if more than one country estimate, then record the mean estimate,␣
↪otherwise go to next loop iteration
    if len(success_estimates[i]) > 2:
        country_mean_value = sum(list_country_estimates) /␣
↪len(list_country_estimates)
        country_means[i] =  country_mean_value # insert country mean value into␣
↪dict using country name as key
```

```
        print("Country: {0:s}, Avg Success Estimate: {1:.2f}".format(i,␣
↪country_mean_value))
    else:
        print("Country: {0:s}, *Does not Meet Company Policy*".format(i))
```

```
Country: Australia, Avg Success Estimate: 0.30
Country: France, Avg Success Estimate: 0.66
Country: Italy, *Does not Meet Company Policy*
Country: Brazil, Avg Success Estimate: 0.29
Country: USA, Avg Success Estimate: 0.33
Country: England, *Does not Meet Company Policy*
Country: Canada, *Does not Meet Company Policy*
Country: Argentina, *Does not Meet Company Policy*
Country: Greece, Avg Success Estimate: 0.61
Country: Morocco, *Does not Meet Company Policy*
Country: Tunisia, *Does not Meet Company Policy*
Country: Egypt, *Does not Meet Company Policy*
Country: Jamaica, Avg Success Estimate: 0.66
Country: Switzerland, Avg Success Estimate: 0.79
Country: Germany, Avg Success Estimate: 0.43
```

### 2.9.2 Exercise 9:

What is another approach to ameliorate the one-sample problem for some countries? Think in terms of the factors that drive confidence in data-driven business decisions.

(a) Group countries together to larger regions to ensure each region has at least one estimate

(b) Only remove a country if its esimates are very large or very small compared to other estimates

(c) Use a different summary statistic for the analysis other than the average value

(d) Revisit why some countries only have one estimate and see if more data can be sourced for these countries

**Answer.** (d).

Answer (a) sounds like a good approach, however it will not accomplish the task at hand. The task is to find a specific country for global expansion, not a region. Thus, if one were to regroup countries to regions, they would first have to redefine the problem statement.

Answer (b) is an incorrect method to handle this particular data set since country statistics should be calculated independently of other countries. If one countries estimates depend on another, this could significantly complicate the analysis and would likely require a more complex and costly investigation.

Regarding answer (c), using a different summary statistic will not solve the problem of having one sample for the statistical calculation. We would like to have more confidence in our statistical estimate. Using a different measure of success such as the minimum or maximum estimate will not ameliorate the single-sample size issue.

It is common to revisit the source of the data set for a problem as an iterative analysis is performed. Hence, answer (d) is a correct approach to ameliorate the one-sample size problem. As new insight is gained throughout the analysis, one should always reflect on the data collection process and determine if any new information may help the analysis to move forward. In this case, upon observing some countries having one estimate, it would likely be beneificial to determine why some countries have such small numbers of estimates and if there is a simple, cost-effective method to obtain additional estimates for these countries before moving forward in the analysis.

## 2.10 Putting it all together

We've used for loops and control structures to calculate partial summary statistics for each of the countries. Let's put it all together to obtain a recommendation on which country we should choose to expand the luxury flight services.

### 2.10.1 Exercise 10:

Write code to print each country name and summary statistics. Each line should show one country and the corresponding summary statistics: Min Estimate (float), Average Estimate (float), Max Estimate (float), Number of Estimates (int), Meets Company Policy of at least 3 estimates (bool). For example, the line for France would appear as:

```
Country: France , Min: 0.2 , Average: 0.655 , Max: 0.98 , NumEst: 4 , MeetsPolicy: True
```

**Answer.** One possible solution is shown below:

```
[29]: country_name_list = list(success_estimates.keys())
      for i in country_name_list:
          min_stat = min(success_estimates[i])
          mean_stat = sum(success_estimates[i]) / len(success_estimates[i])
          max_stat = max(success_estimates[i])
          len_stat = len(success_estimates[i])
          meets_policy = len_stat > 2
          print('Country:',i,', Min:',min_stat,', Average:',mean_stat,', Max:
       ↪',max_stat,', NumEst:',len_stat,', MeetsPolicy:',meets_policy)
```

```
Country: Australia , Min: 0.11 , Average: 0.29500000000000004 , Max: 0.6 ,
NumEst: 4 , MeetsPolicy: True
Country: France , Min: 0.2 , Average: 0.655 , Max: 0.98 , NumEst: 4 ,
MeetsPolicy: True
Country: Italy , Min: 0.6 , Average: 0.6 , Max: 0.6 , NumEst: 1 , MeetsPolicy:
False
Country: Brazil , Min: 0.22 , Average: 0.29 , Max: 0.43 , NumEst: 3 ,
MeetsPolicy: True
Country: USA , Min: 0.2 , Average: 0.3333333333333333 , Max: 0.5 , NumEst: 3 ,
MeetsPolicy: True
Country: England , Min: 0.45 , Average: 0.45 , Max: 0.45 , NumEst: 1 ,
MeetsPolicy: False
Country: Canada , Min: 0.25 , Average: 0.275 , Max: 0.3 , NumEst: 2 ,
MeetsPolicy: False
Country: Argentina , Min: 0.22 , Average: 0.22 , Max: 0.22 , NumEst: 1 ,
MeetsPolicy: False
Country: Greece , Min: 0.15 , Average: 0.61 , Max: 0.99 , NumEst: 6 ,
MeetsPolicy: True
Country: Morocco , Min: 0.29 , Average: 0.29 , Max: 0.29 , NumEst: 1 ,
MeetsPolicy: False
Country: Tunisia , Min: 0.56 , Average: 0.6200000000000001 , Max: 0.68 , NumEst:
2 , MeetsPolicy: False
Country: Egypt , Min: 0.99 , Average: 0.99 , Max: 0.99 , NumEst: 1 ,
MeetsPolicy: False
Country: Jamaica , Min: 0.61 , Average: 0.6566666666666666 , Max: 0.71 , NumEst:
3 , MeetsPolicy: True
Country: Switzerland , Min: 0.51 , Average: 0.7859999999999999 , Max: 0.99 ,
NumEst: 5 , MeetsPolicy: True
Country: Germany , Min: 0.36 , Average: 0.4333333333333333 , Max: 0.49 , NumEst:
3 , MeetsPolicy: True
```

Now that we have summary statistics for a variety of country estimates, let's construct our actionable business recommendation.

## 2.11 Conclusions

From the analysis, Switzerland is the country with the largest chance of success for global expansion with an estimated success rate of 0.79. The summary statistic for Switzerland was calculated with an adequate number of estimates according to company policy. Thus, it is recommended that management explore opportunities in Switzerland for luxury flight services.

In addition, other countries that should be closely monitored for luxury flight services are Jamaica and France, where each held a 0.66 average success estimate. If there are additional resources in the future for further luxury service expansion, these countries may be apt choices.

## 2.12 Takeaways

In this case, we've learned the foundations of Python. Through identifying global expansion opportunities for an airline company we've covered fundamental data types, control structures, and a useful Python workflow to analyze a given set of data. You also learned about various summary statistics and how much confidence you can have in drawing conclusions from them.

Building on this knowledge, you can use these Python tools as both a foundation and a framework to build more complex projects and solve critical business problems. Python continues to be an outstanding tool to perform data-driven analysis and deliver key business insights.

[ ]:

```

```