University of St. Gallen

School Of Management, Economics, Law, Social Sciences, and International Affairs

Implementing a Valuation Model using Python

Helena Mühlberger
Lindenstrasse 29
9000 St. Gallen
+41 78 669 97 51
helenacorina.muehlberger@student.unisg.ch
20-612-578

Nyco Schaller
St. Jakobsstrasse 101,
9000 St. Gallen
+41 78 935 46 24
nyco.schaller@student.unisg.ch
19-612-498

Flurina Haussener
Gottfried-Keller-Strasse 22
9000 St. Gallen
+41 78 606 31 12
flurinaloana.haussener@student.unisg.ch
20-617-720

University of St. Gallen
Programming – Introduction level 4,799
Coding Language: Python
Prof. Dr. Mario Silic
Due Date 17.12.2020

Table of contents

List of figures

List of tables

List of abbreviations

API Application Programming Interface

DCF Discounted Free Cash Flow

FCF Free Cash Flow

GUI Graphical User Interface

IDE Integrated Development Environment

1. The Project Idea

For this project, we tried to find an easier solution to value company. Indeed, estimating the value of a company is often tedious and time-consuming. The latter involves building a valuation model and looking at the company fillings to extract data. We propose an automated python graphical user interface (GUI) that does all the heavy work for us.

Our coding project uses financial data from an application programming interface (API) and then computes the value of the company based on the assumptions the user enters. Our GUI allows the user to enter the ticker symbol of the company he/she wants to value. Then, as a first step, the actual price of the share and the past 4-year growth of free cash flow is printed out on the screen. With that information, the user can make more reasonable future growth assumptions. The user must also enter the discount rate. As a final step, the intrinsic value is displayed on the GUI and compared to today’s stock price.

In the first part of this paper, we will first describe in detail the valuation model used. We will explain why we choose this model and how it works. Then, in the second part, we will demonstrate how our python code functions. We have separated the coding part into three sections: the first one talks about the API used, the second one about the GUI coding and the last one about the calculation steps.

You can find the full Python code and the documentation under GitHub by scanning the QR-code below:

SCAN ME

Link: *https://github.com/nycoschlr/DCF_Model_Python*

2. Calculation of the model

2.1 Justification & Limitation

For our valuation model, we have decided to follow a discounted free cash flow model (DCF). We choose this model because, according to Jennergren (2008, p. 2), it is the most used in practice. Another advantage is that the model is quite simple to understand. The value of the company is equal to the projected free cash flows discounted for today. In practice, however, the calculation can be modeled for company specific data (e.g., computing the weighted average cost of capital), this requires an extensive model. For this project, we will focus on a simple model to implement in python.

One you should keep in mind that the DCF model, as any model, possesses limitations. One drawback of the model is its sensitivity for the input assumptions, a small shift in one’s assumptions can lead to an important shift of the equity value of the company. Furthermore, the model is based on projected assumptions but predicting the future is a complex task. In addition, the company needs to have positive cash flows, this is often not the case for growing startups. In the next part, we will explain the formula behind our model.

2.2 Formula

In this section, we will explain what formula and calculations we did to find the intrinsic value of companies. As a first step, we compute the value of future cash flows. The general formula of the DCF model is as follow:

$$\boxed{DCF = \frac{FCF_1}{(1+r)^1} + \frac{FCF_2}{(1+r)^2} + \frac{FCF_3}{(1+r)^3} + \cdots + \frac{FCF_n}{(1+r)^n}}$$

$$where:$$

$$DCF\ represents\ the\ sum\ of\ all\ future\ cash\ flows\ discounted$$

$$FCF_n\ represents\ the\ free\ cash\ flow\ in\ period\ n$$

$$r\ represents\ the\ discount\ rate$$

$$n\ represents\ the\ time\ in\ years$$

This formula gives us the value of all future cash flows reported to today's value. However, in our model, we are not going to value the cash flows indefinitely. Thus, we have decided to project the next 6 years. However, the business might still be able to generate cash flow after the 6-year period, how can one account for that factor? This is where the terminal value comes in. As a second step, we compute a value for all the future cash flows after period 6. For this calculation, we used the following formula:

$$\boxed{Terminal\ value = \frac{FCF_{n\ at\ end} * (1+g)}{r-g}}$$

$$where:$$

$$g\ represents\ the\ long-term\ growth\ rate$$

$$n\ at\ end\ represents\ the\ end\ period\ of\ the\ projected\ cash\ flows$$

At this stage, we have calculated the value of projected future free cash flows and accounted for the fact that the company might still generate cash at a growth rate of g afterwards. By adding the two, we get the enterprise value, defined as follow in our case:

$$\boxed{Entreprise\ value = DCF + Terminal\ Value}$$

Since we only want to find the equity value of the company, we must adjust the enterprise value like this:

$$\boxed{Equity\ Value = Entreprise\ value - Net\ debt + Cash\ Amount}$$

Then, as a user, we want to compare our intrinsic value to the actual share price of the stock. Therefore, we just need to divide the computed equity value by the number of share outstanding. This gives us the price we should pay for the company. From our calculated price, we can then estimate if the actual price is overvalued or undervalued.

$$\boxed{Equity\ value\ per\ share = \frac{Equity\ Value}{Number\ of\ shares\ outstanding}}$$

2.3 Excel Version

To check the sanity of our python code, we first implemented the valuation model in excel. This allows us to verify that our python code outputs the same value as our excel calculation. It also serves as a guideline to implement the calculations step by step in python. We also used a color code to represent different steps for our python code.

	Past Value	Future Value					
years	2020	**2021**	**2022**	**2023**	**2024**	**2025**	**2026**
FCF	92953000000	1.04107E+11	1.166E+11	1.30592E+11	1.46263E+11	1.63815E+11	1.83473E+11
FCF discounted		94643054545	96363837355	98115907125	99899832710	1.01716E+11	1.03566E+11
Sum of FCF	5.94304E+11						
Discount rate	10%						
Growth rate of FCF	12%						
Long-term growth rate	4%						
FCF n=6	1.83473E+11						
Terminal Value	3.18019E+12						
Discounted TV	1.79514E+12						
Entreprise Value	2.38944E+12						
Net Debt	89779000000						
Cash	34940000000						
Number of shares	16701272000						
Equity value	2.3346E+12						
Equity value per share	139.79						

Table 1: The Implementation of our model in excel. Table compiled by author.

The yellow cases represent the data that our python would need to get from a financial data provider. The orange cases represent the input assumption of our user. The bottom dark blue case represents the calculated equity value that needs to be displayed to our user. This technique helps us to develop the code in a structured manner. In the next section, we will explain our python implementation.

3. Python Code Explanation

3.1 API

Firstly, we need financial data for our python code. We have decided to use the API of financial modeling prep (accessible under https://site.financialmodelingprep.com/developer) since they have a free plan, detailed documentation, and the financial data that we are looking for. They offer balance sheets, income statements, and cash flow statement data. We signed up for free and got our API key. Our limit is 250 API calls per day. Below you can find an example of the documentation.

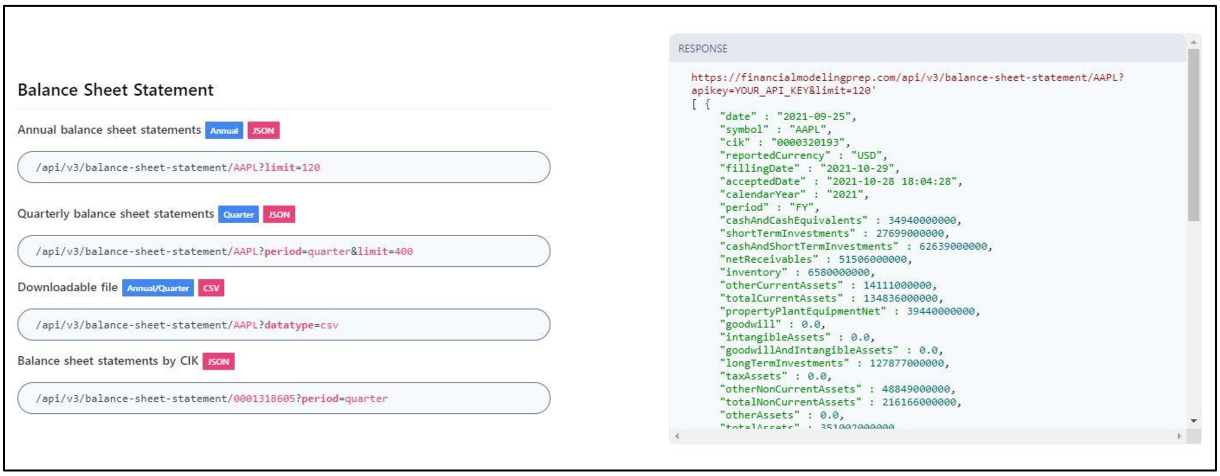

Figure 1: How to get balance sheet data using the API. Retrieved from https://site.financialmodelingprep.com/developer/docs.

As a summary, we need to find the following financial data for our model: the free cash flows, the net debt, the cash, the number of shares outstanding.

3.2 GUI

This first part of our python code is about coding the GUI. We used a video tutorial named "Tkinter Course – Create Graphical User Interface in Python Tutorial" published by freeCodeCamp.org to implement our GUI, it is accessible under the following link: *https://www.youtube.com/watch?v=YXPyB4XeYLA&list=LL&index=5*.

For our GUI, we are going to need the package Tkinter, it is useful since it is especially designed for creating GUI. First, we need to install the package in our integrated development environment (IDE). This can be done running the following command line in our terminal:

$$\boxed{pip\ install\ tk}$$

Once this is done, we can start coding. For the GUI part, we follow this structure: Tkinter Labels, then Tkinter Inputs.

```
##################################################
# Welcome to our company valuation Python code !
##################################################


#  The following code is divided in two part.
#  Part 1 is about coding the GUI.
#  Part 2 is about the DCF model.


##################################
# Part 1: Coding the GUI
##################################

# As a first step, one should install the package tkinter, it enables us to simply create GUIs.
# Run the following line in the terminal:  "pip install tk"

#Imoprting the tkinter package. Rename it as tk.

import tkinter as tk

#Creating a root widget to initialize tkinter

root = tk.Tk()

# Giving a title to our GUI and dimensions

root.title("DCF Model GUI")
root.geometry("1400x700")

```

Figure 2: Introduction part of our code. Compiled by author.

To see our GUI, we also need to add one line of code at the end which is:

$$\boxed{root.mainloop()}$$

This line is not in Figure 2 because it is at the end of the full code.

We have now initiated a simple GUI with nothing on the screen. In the next part, we will create Labels which are text elements on the screen.

```
#################
# Tkinter Labels
#################

# This section is about Labels. Labels are boxes of text that we can display on our GUIS.
# We can choose the font, the font style, and font size in the font section of the Label.
# When a label is created, we can use the grid function to position the object on the GUI.
# We inserted blank labels to create some spacing between objects on the screen. This is required
# because otherwise the objects would appear close to each others if no other objects are in the way.

# Here we create three labels by using the Label function. We then position them on the screen using the
# grid system. We can specify the row and the column where we want to place the object.

# Title Label
# We give our label a name (title_label). Then call the function Label from the package to create a label.
# The "root" word means we want to create the element in the initial root.
# We can specify the font by passing different parameters.
title_label = tk.Label(root, text="Discounted Free Cash Flow Valuation Model", font=("calibre", 20, "bold"))
title_label.grid(row=0, column=0)


#Welcome Label
welcome_label = tk.Label(root, text="Welcome to our python valuation model !", font=("calibre", 13))
welcome_label.grid(row=1, column=0)


#Blank number 0
blank = tk.Label(root, text="                                ", font=("calibre", 20, "bold"))
blank.grid(row=2, column=0)


#Ticker symbol Label
ticker_symbol_box = tk.Label(root, text="1) Enter the ticker symbol (US stock):", font=("times new roman", 14))
ticker_symbol_box.grid(row=6, column=0)


#Blank number 1
blank_1 = tk.Label(root, text="                        ", font=("calibre", 20, "bold"))
blank_1.grid(row=6, column=1)


#Stock price output Label
output_stock_price = tk.Label(root, text="Actual Stock Price: ", font=("arial", 14))
output_stock_price.grid(row=6, column=2)
```

Figure 3: Tkinter Labels Part 1. Compiled by author.

```python
#Past growth Label
output_growth_rate_fcf = tk.Label(root, text="4-yr growth rate of FCF: ", font=("arial", 14))
output_growth_rate_fcf.grid(row=7, column=2)


#Future growth rate Label
future_growth_rate_fcf = tk.Label(root, text="2) Future growth rate assumption of FCF in %: ", font=("times new roman", 14))
future_growth_rate_fcf.grid(row=16, column=0)


#Discount rate Label
future_discount_rate = tk.Label(root, text = "Discount rate (desired returns) in %: ", font=("times new roman", 14))
future_discount_rate.grid(row=18, column=0)


#Blank number 2
blank_2 = tk.Label(root, text= "              ", font=("calibre", 20, "bold"))
blank_2.grid(row=13, column=2)


#Calculated value Label
output_intrinsic_value = tk.Label(root, text="Intrinsic Value: ", font=("arial", 14))
output_intrinsic_value.grid(row= 14, column=2)


#Output potential Label
output_upside_potential = tk.Label(root, text="Upside potential: ", font=("arial", 14))
output_upside_potential.grid(row=16, column=2)

# All the texts elements are now displayed on the screen.
```

*Figure 4. Tkinter Labels Part 2. Compiled by author***.**

All the text elements are displayed on the screen.

However, we still need the input of users. We use the function Entry for this purpose.

```python
#################
# Tkinter Inputs
#################

# Here we create and position the user input elements.
# User input are created by using the entry function.
# We can vary the width of the entry box by changing the number.
# Same as for labels, we can position the element by using the grid
# function.

#Ticker symbol from user's input
input_ticker_symbol = tk.Entry(root, width=12)
input_ticker_symbol.grid(row=6, column=1)


#Future growth rate from user's input
input_future_growth_rate = tk.Entry(root, width=12)
input_future_growth_rate.grid(row=16, column=1)


#Discount rate from user's input
input_discount_rate = tk.Entry(root, width=12)
input_discount_rate.grid(row=18, column=1)



# The GUI is almost done.
# We still need to add three buttons.
# We need to define functions to tell the buttons what to do when
# they are clicked on. Thus, we will first define our functions
# and then create and place our buttons on the screen.
```

Figure 5: Tkinter User Inputs. Compiled by author.

The Tkinter buttons still need to be defined. However, functions need to be given to buttons. Thus, we will define our function and then come back to the buttons.

3.3 Calculation steps

We can now define our functions for the buttons. But, first, we need to import the requirements to communicate with the API.

```
##################################
# Part 2: Coding the DCF Model
##################################


# In this part, we define the two functions needed for our buttons.
# Function 1 retrieves the stock price and the past growth.
# Function 2 computes the equity value per share.


# API requirements
# We import requests and pandas, they will be used
# when communicating with API. We define our API key.

import requests
import pandas as pd
key = "04871ed7089e320af811c91614b80420"


# We define those two variables outside the functions.
# They will be used as global variables so the second function
# does need to call the API again for the same data.

cash_2020 = 0
price_of_ticker = 0

```

Figure 6: API requirements. Compiled by author.

Then we define our first function in figure 7 and 8. Function 1 in will display the actual stock price on the screen (part 1) as well as the past 4-year growth of the free cash flow (part 2). This is useful for the user to make more sound assumptions for the future growth rate input.

```python
#################
# Function 1
#################


# Function number 1 takes the user's input of the ticker symbol to look up the price of the stock and
# calculate the last 4-year growth of the free cash flows.

def get_financial_data():

    #Retrieve the input of the user for the ticker symbol
    ticker = input_ticker_symbol.get()

    #Convert it in upper case for the API
    ticker = ticker.upper()

    #Getting the price of the stock
    #We found the stock price in the API under the "discounted-cash-flow" document
    document = "discounted-cash-flow"

    #Passing the parameters in the url
    url = "https://financialmodelingprep.com/api/v3/{}/{}?&apikey={}".format(document, ticker, key)
    r = requests.get(url)

    #We transform the dictionary data into a dataframe, and transpose for easier readability.
    price = pd.DataFrame.from_dict(r.json()).transpose()

    #Shift the first row as a column
    price.columns = price.iloc[0]

    #We ignore the first row
    price = price.iloc[1:]

    #Making the price_of_ticker a global variable so the second function does not have to
    #do an API call again.
    global price_of_ticker

    #We locate the price data and round it
    price_of_ticker = price.iloc[2][0]
    price_of_ticker = round(price_of_ticker, 2)

    #We create a label and position it on the screen. The price will now be displayed on the GUI.
    price_label = tk.Label(root, text=str(price_of_ticker), font=("times new roman", 14))
    price_label.grid(row=6, column=3)
```

Figure 7: Function 1, part 1. Compiled by author.

The price of the stock is now displayed next to the label “Actual Stock Price:” in the GUI. The next step is to display the past growth (part 2). This is detailed in Figure 8.

```
232 #Here we will display on the screen the last 4-yr growth of the free cash flows
233
234
235 # We need to retrieve the past cash flows.
236 # his data can be found under the cash flow statements of the API
237 #Those are almost the same steps as above but with different parameters.
238 #We get the data from the API, transpose it and convert it to a dataframe.
239 document_cashflow = "cash-flow-statement"
240 url_2 = "https://financialmodelingprep.com/api/v3/{}/{}?&apikey={}".format(document_cashflow, ticker, key)
241 r2 = requests.get(url_2)
242 cf_statement = pd.DataFrame.from_dict(r2.json()).transpose()
243 cf_statement.columns = cf_statement.iloc[0]
244 cf_statement = cf_statement.iloc[1:]
245
246 #We store the cash flows in the variable cash_flows
247 cash_flows = cf_statement.loc["freeCashFlow"]
248
249 #We make the free cash flow of 2020 a global variable so the second function can
250 #directly use it.
251 global cash_2020
252 cash_2020 = cash_flows.iloc[0]
253
254 #Here we create a try/except since some stocks have 6 years of data instead of 5 years.
255 #This is to debug the mistake "index out of bound".
256 try:
257     cash_2016 = cash_flows.iloc[5]
258 except:
259     cash_2016 = cash_flows.iloc[4]
260
261
262
263 #Here we use a formula to compute the average growth rate of the last 4 years.
264 #We use the compound annual growth rate formula
265 #We convert the cash to integer for the calculation, and format the results as percentage
266 average_compound_growth = (((int(cash_2020) / int(cash_2016)) ** (1 / 4)) - 1)*100
267
268 #We round the results to two decimals place.
269 average_compound_growth = round(average_compound_growth, 2)
270
271 #We display the growth rate on the screen using label and grid. We add a "%" sign to our result.
272 growth_label = tk.Label(root, text=str(average_compound_growth)+"%", font=("times new roman", 14))
273 growth_label.grid(row=7, column=3)
274
```

Figure 8: Function 1, part 2. Compiled by author.

For the lines 242, 243, and 244, we used the help from an online video tutorial published by Spencer Pao called "SAVING TIME: Scraping Financial Data", accessible under the link *"https://www.youtube.com/watch?v=GGgNM7WanK8"*.

For the average compound growth rate, we used the following formula:

$$\boxed{CAGR = \frac{(CFCF\ 2020)^{\frac{1}{4}}}{(FCF\ 2016)} - 1}$$

Function 1 is now defined. We will now assign function 1 to a button, so when the user clicks on the button, the function gets executed. This is explained in figure 9.

```python
#######################
# Button for function 1
#######################


#We use the button function of Tkinter. Under command, we precise which function to execute for our button.
#We also name our button "Get Financial data". Using grid, we position our button. We choose the size with padx and
#pady. We selected a background color with bg.
button_get_data = tk.Button(root, text="Get financial data", command=get_financial_data, padx=50, pady=8, bg="#89CFF0")
button_get_data.grid(row=7, column=0)

```

Figure 9: Button for function 1. Compiled by author.

We can now look at function 2. The latter will import financial data, use the user's input assumptions, and calculate the intrinsic value just like our excel model did.

```python
#################
# Function 2
#################

# Function 2 computes the intrinsic value and displays it on the screen.
# To do this, the function needs the user assumptions about growth and discount rate.
# In term of financial data, the function needs the net debt, the cash position and
# the number of shares.

def intrinsic_value():

    #We retrieve the input of the ticker symbol and put in upper case.
    #We retrieve this data by using the get function on our input label.
    ticker = input_ticker_symbol.get()
    ticker = ticker.upper()

    #We retrieve the user input regarding the growth rate.
    future_growth_rate = input_future_growth_rate.get()

    #we convert the input to int.
    future_growth_rate = int(future_growth_rate)

    #We convert the growth rate in decimals.
    future_growth_rate = future_growth_rate/100

    #Retrieve, convert in int and in decimals to input discount rate.
    discount_rate = input_discount_rate.get()
    discount_rate = int(discount_rate)
    discount_rate = discount_rate/100

    #Here we do the projections of FCF for the next 6-years.
    #This is based on the assumption of the user.
    cash_2021 = cash_2020 * (1 + future_growth_rate)
    cash_2022 = cash_2021 * (1 + future_growth_rate)
    cash_2023 = cash_2022 * (1 + future_growth_rate)
    cash_2024 = cash_2023 * (1 + future_growth_rate)
    cash_2025 = cash_2024 * (1 + future_growth_rate)
    cash_2026 = cash_2025 * (1 + future_growth_rate)

    #Here we discount each cash flow to obtain the present value.
    cash_discount_2021 = cash_2021 / (1 + discount_rate)
    cash_discount_2022 = cash_2022 / ((1 + discount_rate) ** 2)
    cash_discount_2023 = cash_2023 / ((1 + discount_rate) ** 3)
    cash_discount_2024 = cash_2024 / ((1 + discount_rate) ** 4)
    cash_discount_2025 = cash_2025 / ((1 + discount_rate) ** 5)
    cash_discount_2026 = cash_2026 / ((1 + discount_rate) ** 6)

```

Figure 10: Function 2, part 1. Compiled by author.

```python
    #We sum each discounted cash flow.
    sum_of_discounted_free_cash_flow = cash_discount_2021 + cash_discount_2022 + \
                                       cash_discount_2023 + cash_discount_2024 + cash_discount_2025 + cash_discount_2026

    #We compute the terminal value. We assume 4% long term growth rate.
    #The following lines follow the terminal value formula.
    long_term_growth_rate = 0.04
    fcf_2026 = cash_2026*(1+long_term_growth_rate)
    terminal_value = fcf_2026/(discount_rate-long_term_growth_rate)
    terminal_value_discounted = terminal_value/((1+discount_rate)**6)

    #We then add upp the terminal value with the sum of discounted FCF to get the enterprise value.
    enterprise_value = sum_of_discounted_free_cash_flow + terminal_value_discounted


    #We know we need the net debt, cash position and the number of shares to find the equity value.

    #Here we ask the API to look at the balance sheet in order to retrieve the cash position and the net debt.
    #Similar as previous steps.
    document_balance_sheet = "balance-sheet-statement"
    url_3 = "https://financialmodelingprep.com/api/v3/{}/{}?&apikey={}".format(document_balance_sheet, ticker, key)
    r3 = requests.get(url_3)
    balance_sheet = pd.DataFrame.from_dict(r3.json()).transpose()
    balance_sheet.columns = balance_sheet.iloc[0]
    balance_sheet = balance_sheet.iloc[1:]

    #We locate the cash and net debt item on the balance sheet.
    cash = balance_sheet.iloc[7][0]
    net_debt_2020 = balance_sheet.iloc[48][0]
```

Figure 11: Function 2, part 2. Compiled by author.

We still need to find the number of shares outstanding. Then, we can compute the equity value per share just like our excel model did.

```python
    #Here we retrieve the number of shares from the API. The number of shares can be found in the enterprise value
    #document. The document, the ticker symbol, and our API key are passed as parameters.
    document_number_shares = "enterprise-values"
    url_4 = "https://financialmodelingprep.com/api/v3/{}/{}?limit=40&apikey={}".format(document_number_shares, ticker,
                                                                                        key)
    r4 = requests.get(url_4)

    #Similar steps as before. Convert to dataframe. Move a row to a column.
    data_shares = pd.DataFrame.from_dict(r4.json()).transpose()
    data_shares.columns = data_shares.iloc[0]
    data_shares = data_shares.iloc[1:]

    #We more precisely find the number of shares outstanding.
    number_of_shares_outstanding = data_shares.loc["numberOfShares"]
    number_shares_2020 = number_of_shares_outstanding.iloc[0]

    #In some cases we find the number of shares to be zero due to poor documents
    # if this is the case, we take the number of shares of last year.

    if number_shares_2020 == 0:
        number_shares_2020 = number_of_shares_outstanding.iloc[1]
```

Figure 12: Function 2, part 3. Compiled by author.

In figure 13, we can finally compute the equity value per share.

```python
    #We can finally compute the equity value by following the DCF formula.
    equity_value = enterprise_value - net_debt_2020 + cash
    equity_value_per_share = equity_value / number_shares_2020

    #We round the results to two decimals.
    equity_value_per_share = round(equity_value_per_share, 2)

    #We create a label for the output to be displayed on the screen. We add a "USD" symbol for better
    #understanfding. Using grid, we positioned the element on the GUI screen.
    intrinsic_value_label = tk.Label(root, text=str(equity_value_per_share)+" USD", font=("times new roman", 14))
    intrinsic_value_label.grid(row=14, column=3)

    #We define an upside potential describing the potential gain in percentage between the calculated
    #value and the actual stock price. The upside potential is negative if the stock is overvalued.
    upside_potential = (equity_value_per_share / price_of_ticker) - 1

    #Put in percentage
    upside_potential = upside_potential*100

    #Rounding to two decimals
    upside_potential = round(upside_potential, 2)

    #Creating a label to display the upside potential as text. We convert the upside_potential into a string.
    upside_potential_label = tk.Label(root, text=str(upside_potential)+"%", font=("times new roman", 14))
    upside_potential_label.grid(row=16, column=3)
```

Figure 13: Function 2, part 4. Compiled by author.

Our GUI also computes a potential upside in percentage meaning how much the stock is undervalued compared to the actual stock price. The potential upside is positive when the stock is undervalued and negative when overvalued. We just need to link function 2 to a button and everything should work. We also added an exit button to quit the GUI when the user is done.

```python
#######################
# Button for function 2
#######################

#Button for the function 2 to display the intrinsic value and the potential upside compared to the actual price.
#Using command we link the function and the button together. Padx, and pady are for the dimension of the button.
#Bg is for the background color of the button.
button_calculate_intrinsic_value = tk.Button(root, text="Calculate the intrinsic value", command=intrinsic_value,
                                             padx=50, pady=8, bg="#89CFF0")

#Postioning
button_calculate_intrinsic_value.grid(row=20, column=0)


#Blank label for spacing reasons
blank_3 = tk.Label(root, text="            ", font=("calibre", 20, "bold"))
blank_3.grid(row=19, column=0)


#Small button to quit the root GUI using the function quit.
button_quit = tk.Button(root, text="Exit Program", command=root.quit)
button_quit.grid(row=20, column=3)


#Create an infinite loop to display the GUI
root.mainloop()
```

Figure 14: Function 2, button. Compiled by author.

As a result, running the code gives us a GUI where the user can input the ticker symbol, the growth rate, the discount rate. As an output, the actual stock price, the past 4-year growth of free cash flows (FCF), the intrinsic value, and the potential upside are displayed on the screen. The final GUI can be seen in figure 15.

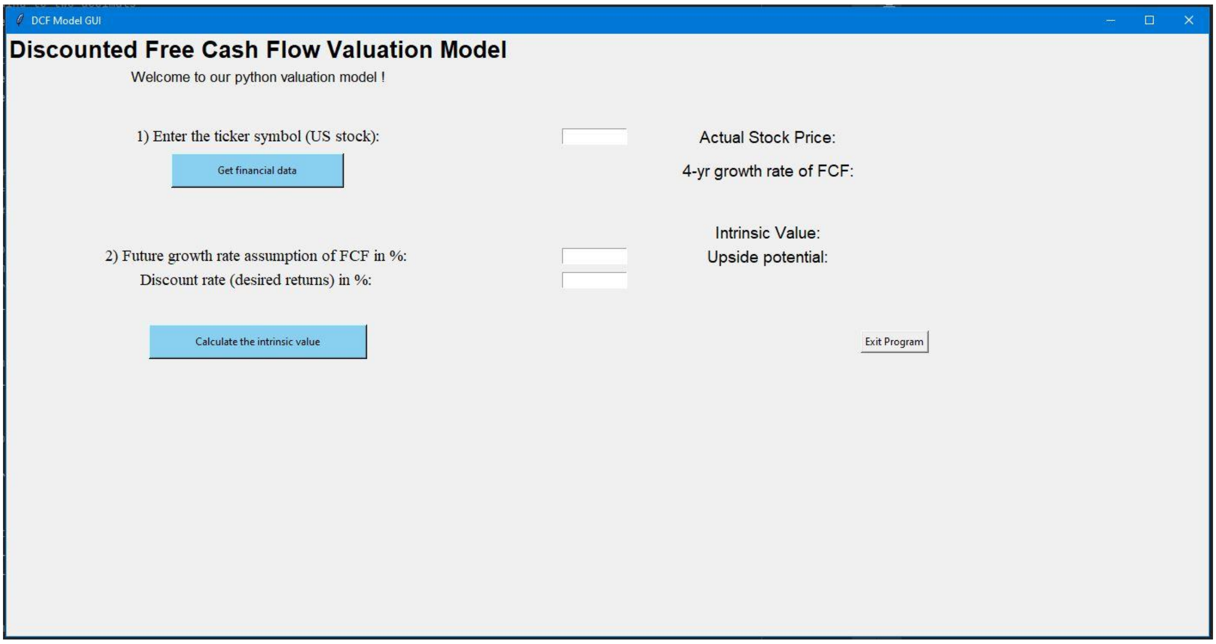

Figure 15: Final GUI. Compiled by author.

3.4 Testing & Limitations

We decided to quickly compare the strengths of our model with valuation from the website "https://www.gurufocus.com". We choose three companies to test our model: Apple (AAPL), Intel (INTC) and Meta Platforms (FB). The following assumptions were used:

Ticker	Growth rate	Discount rate	Conclusion	Gurufocus
AAPL	10	10	Overvalued by 17.89%	Overvalued by 28.45%
FB	20	10	Overvalued by 6.59%	Undervalued by 5.63%
AMZN	32	10	Undervalued by 7.29%	Undervalued by 18.53%

Table 2: Comparison of our model to gurufocus.com. Table compiled by author (Gurufocus, 2021)

Overall, our model pointed towards the same direction as guru focus with an error margin of around 10%. Thus, our model seems to be, at least, not wrong. The goal of our model is not to find a precise figure for the equity value but more to have a general idea of an attractive valuation for companies.

Regarding the limitation of our python code, some companies are not covered by the API. This could cause an error. For example, if we type in the Swiss company Swisscom, this will make our code crash. Another issue might be the different format between the different companies' documents. This could make our code lost when looking for a specific data. In addition, sometimes data points from the API are set up to zero, this might be an issue for the calculation of the intrinsic value and give biased numbers. As further research, our code could be improved to factor in all those potential errors.

References

Financial Data for every needs. Financial Modeling Prep. (n.d.). Retrieved November 17, 2021, from https://site.financialmodelingprep.com/developer.

Value investing: Market insight of investment gurus. Value Investing | Market Insight of Investment Gurus. (n.d.). Retrieved November 19, 2021, from https://www.gurufocus.com/new_index/.

Jennergren, L. P. (2008). Continuing value in firm valuation by the discounted cash flow model. *European Journal of Operational Research*, *185*(3), 1548-1563.

SAVING TIME: Scraping Financial Data. Published by Spencer Tao. Retrieved from https://www.youtube.com/watch?v=GGgNM7WanK8.

Tkinter Course - Create Graphic User Interfaces in Python Tutorial. Published by freeCodeCamp.org. Retrieved from https://www.youtube.com/watch?v=YXPyB4XeYLA&list=LL&index=6.

Declaration of authorship

"We hereby declare

- that we have written this thesis without any help from others and without the use of documents or aids other than those stated above
- that we have mentioned all the sources used and that we have cited them correctly according to established academic citation rules
- that we have acquired any immaterial rights to materials we may have used, such as images or graphs, or that we have produced such materials ourselves
- that the topic or parts of it are not already the object of any work or examination of another course unless this has been explicitly agreed to with the faculty member in advance and is referred to in the thesis
- that we will not pass on copies of this work to third parties or publish them without the university's written consent if a direct connection can be established with the University of St. Gallen or its faculty members
- that we are aware that my work can be electronically checked for plagiarism and that we hereby grant the University of St. Gallen copyright in accordance with the Examination Regulations insofar as this is required for administrative action
- that we are aware that the university will prosecute any infringement of this declaration of authorship and, in particular, the employment of a ghostwriter, and that any such infringement may result in disciplinary and criminal consequences which may result in our expulsion from the university, or my being stripped of my degree."

"By uploading this academic term paper, we confirm through my conclusive action that we are submitting the Declaration of Authorship, that we have read and understood it, and that it is true."

Nyco Schaller, Helena Mühlberger, Flurina Haussener

Characters count: 16'131