

Name: Kuek Yu Fei

Analysing financial statement:

In this project, we will build a prototype to look at a company's sales and the projected sales aimed to achieve for the different financial years.

```
In [32]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

We will need to import the appropriate libraries first before embarking on the project. I have used numpy, pandas, and matplotlib. Numpy is a base N-dimensional array package, pandas are used for data analysis and matplotlib for comprehensive plotting in this project. This is important as it will help with data manipulation and visualisation.

Creating financial statements:

```
In [ ]: # // not for comments
#no ; is required

In [52]: years = ['2018A', '2019B', '2020P', '2021P', '2022P', '2023P']
sales = pd.Series(index=years)
sales['2018A'] = 31.0
sales

<ipython-input-52-a4e0cd0137e1>:2: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64'
in a future version. Specify a dtype explicitly to silence this warning.
sales = pd.Series(index=years)

Out[52]: 2018A    31.0
2019B     NaN
2020P     NaN
2021P     NaN
2022P     NaN
2023P     NaN
dtype: float64
```

The comments are just for my own references. *

The first three lines that was created to form a data structure with an index consisting of years.

A stand for Actual.

B stands for Budget.

P stands for Projected.

The 'NaN' shown here is "Not a Number", empty cells for the projection. The fourth line of this cells (sales) will print a representation of the data. This will enable us to have a sensible representation of it by typing the variable in interactive interpreter.

```
In [53]: growth_rate = 0.1
for year in range(1, 6):
    sales[year] = sales[year - 1] * (1 + growth_rate)

sales

Out[53]: 2018A    31.00000
2019B    34.10000
2020P    37.51000
2021P    41.26100
2022P    45.38710
2023P    49.92581
dtype: float64
```

Next, I have declared a variable to represent the projected annual sales growth. At this point, it is only an estimate and might not be the actual values. Reason being is because I want to use the same input and make sure the software performs the same result as the excel data sheet, before replacing estimates with my probability distributions. By having this variable, I can create a loop that calculates the sales of each year projections with respect to the previous year and growth rate. I have the set the year in range (1, 6) as what I have intended to in the first cell. So, instead of NaN, I have the projected sales.

```
In [54]: #ebit stands for earning before interest and taxes
#ebitda stands for earning before interest, taxes, depreciation and amortization

ebitda_margin = 0.14
depr_percent = 0.032
ebitda = sales * ebitda_margin
depreciation = sales * depr_percent
ebit = ebitda - depreciation
```

```
In [55]: #nwc stands for net working capital
#capex stands for capital expenditure

nwc_percent = 0.24
nwc = sales * nwc_percent
change_in_nwc = nwc.shift(1) - nwc
capex_percent = depr_percent
capex = -(sales * capex_percent)
tax_rate = 0.25
tax_payment = -ebit * tax_rate
tax_payment = tax_payment.apply(lambda x: min(x, 0))
free_cash_flow = ebit + depreciation + tax_payment + capex + change_in_nwc
free_cash_flow
```

```
Out[55]: 2018A    NaN
2019B    2.018100
2020P    2.219910
2021P    2.441901
2022P    2.686091
2023P    2.954700
dtype: float64
```

Using the same approach and method as the previous cells, we will be following through the financial statements and declaring all the variables needed. This is to perform the necessary calculations to eventually arrive at free cash flow. Once we have gotten the values, we can cross check if it corresponds to the excel sheet.

For clarity of the tax payment, I applied a small function to ensure that in scenarios where profit before tax becomes negative, we won't have a positive tax payment.

Performing DCF (discount_factor) value:

Arriving at projected cash flows, we can now calculate a simple terminal value and discount all cash flows back to the present to get the DCF result. I will be using indexing and slicing in this section, which will allow access to one or more elements in data structure.

I have accessed the elements by writing square brackets directly after the name of the structure. Simple indexing accesses elements by their position, starting with 0, meaning 'free_cash_flow' will give us the second element. [-1] is for accessing the last element, for the last year's cash flow is used to calculate terminal value. Using a colon gives us a slice, so [1:] gives us all elements except the first one, since I do not want to include historical year '2018A' in DCF valuation projection.

```
In [56]: cost_of_capital = 0.12
terminal_growth = 0.02
terminal_value = ((free_cash_flow[-1] * (1 + terminal_growth)) /
(cost_of_capital - terminal_growth))
discount_factors = [(1 / (1 + cost_of_capital)) ** i for i in range (1,6)]
dcf_value = (sum(free_cash_flow[1:] * discount_factors) +
terminal_value * discount_factors[-1])
dcf_value
```

Out[56]: 25.79438401113792

Exporting data:

```
In [57]: output = pd.DataFrame([sales, ebit, free_cash_flow],
index=['Sales', 'EBIT', 'Free Cash Flow']).round(1)
output.to_excel('Python DCF Model Output.xlsx')
output
```

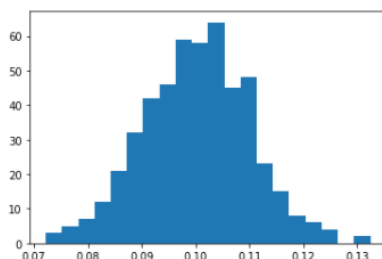
Out[57]:

	2018A	2019B	2020P	2021P	2022P	2023P
Sales	31.0	34.1	37.5	41.3	45.4	49.9
EBIT	3.3	3.7	4.1	4.5	4.9	5.4
Free Cash Flow	NaN	2.0	2.2	2.4	2.7	3.0

Creating probability distributions:

The first step I took was to decide how many iterations I want to run in the simulation. I have decided that 500 as a starting point as it strikes a balance between getting enough data points to get sensible output plots, versus having the simulation finish within a certain time frame. Now, I must generate the actual distributions. After deciding on distributions, I have specify the parameters required to describe the shape such as mean and standard deviation, and also the number of desired outcomes.

```
In [58]: iterations = 500
sales_growth_dist = np.random.normal(loc=0.1, scale=0.01, size=iterations)
ebitda_margin_dist = np.random.normal(loc=0.14, scale=0.02, size=iterations)
nwc_percent_dist = np.random.normal(loc=0.24, scale=0.01, size=iterations)
plt.hist(sales_growth_dist, bins=20)
plt.show()
```



There may be arguments that EBITDA should not be a separate random variable independent from sales but correlated to sales in some degree. However, that would be too much complexity and I will be doing it this way for the sake of clarity.

The less data of my inform choice of distribution and parameters, the more I will have to rely on the outcome of my various due diligence workstreams, combined with experience, to form a consensus view on ranges of likely scenarios.

In this project, with cash flow projections, there will be a large subjective component, which means that visualizing the probability distributions becomes important. I can get a basic visualization, showing the sales growth distribution, with only two short lines of code. This way I can quickly view any distribution to eyeball one that best reflects a view.

Now that I have all the building blocks needed to run the simulation, but they are not in a convenient format for running the simulation.

Here is the same code I have used thus far but all gathered in one cell and rearranged into a function for convenience:

```
In [59]: def run_mcs():

    # Create probability distributions
    sales_growth_dist = np.random.normal(loc=0.1, scale=0.01, size=iterations)
    ebitda_margin_dist = np.random.normal(loc=0.14, scale=0.02, size=iterations)
    nwc_percent_dist = np.random.normal(loc=0.24, scale=0.01, size=iterations)

    # Calculate DCF value for each set of random inputs
    output_distribution = []
    for i in range(iterations):
        for year in range(1, 6):
            sales[year] = sales[year - 1] * (1 + sales_growth_dist[0])
            ebitda = sales * ebitda_margin_dist[i]
            depreciation = (sales * depr_percent)
            ebit = ebitda - depreciation
            nwc = sales * nwc_percent_dist[i]
            change_in_nwc = nwc.shift(1) - nwc
            capex = -(sales * capex_percent)
            tax_payment = -ebit * tax_rate
            tax_payment = tax_payment.apply(lambda x: min(x, 0))
            free_cash_flow = ebit + depreciation + tax_payment + capex + change_in_nwc

        # DCF valuation
        terminal_value = (free_cash_flow[-1] * 1.02) / (cost_of_capital - 0.02)
        free_cash_flow[-1] += terminal_value
        discount_factors = [(1 / (1 + cost_of_capital)) ** i for i in range(1,6)]
        dcf_value = sum(free_cash_flow[1:] * discount_factors)
        output_distribution.append(dcf_value)

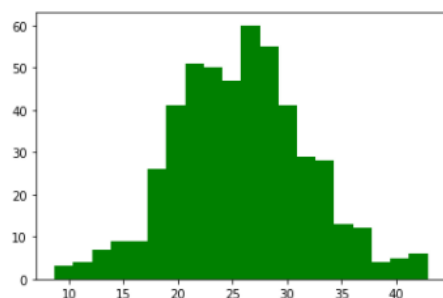
    return output_distribution
```

I can now run the whole simulation and plot the output distribution, which will be the discounted cash flow value of this company in each of the 500 iterations, with the following code.

The %time command is not Python code but a notebook shorthand that measures the time to run. *

```
In [60]: %time plt.hist(run_mcs(), bins=20, color='g')
plt.show()
```

Wall time: 668 ms



Refining the prototype:

I will be rewriting existing codes to improve structure without changing functionality.

- Organise the different parts in a more sensible way.
- Renaming variables and functions to make purpose and working clearer.
- Allow future features.

- Improve execution speed.

I have cleaned up the prototype by collecting initial variables in one place rather than scattering it throughout the prototype script, and optimised execution speed.

```
In [61]: # Key inputs from DCF model
years = 5
starting_sales = 31.0
capex_percent = depr_percent = 0.032
sales_growth = 0.1
ebitda_margin = 0.14
nwc_percent = 0.24
tax_rate = 0.25
# DCF assumptions
r = 0.12
g = 0.02
# For MCS model
iterations = 500
sales_std_dev = 0.01
ebitda_std_dev = 0.02
nwc_std_dev = 0.01
```

```
In [62]: def run_mcs():

    # Generate probability distributions
    sales_growth_dist = np.random.normal(loc=sales_growth,
                                          scale=sales_std_dev,
                                          size=(years, iterations))

    ebitda_margin_dist = np.random.normal(loc=ebitda_margin,
                                          scale=ebitda_std_dev,
                                          size=(years, iterations))

    nwc_percent_dist = np.random.normal(loc=nwc_percent,
                                        scale=nwc_std_dev,
                                        size=(years, iterations))

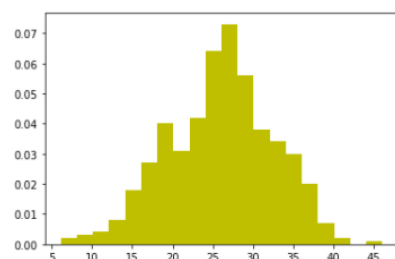
    # Calculate free cash flow
    sales_growth_dist += 1
    for i in range(1, len(sales_growth_dist)):
        sales_growth_dist[i] *= sales_growth_dist[i-1]
    sales = sales_growth_dist * starting_sales
    ebitda = sales * ebitda_margin_dist
    ebit = ebitda - (sales * depr_percent)
    tax = -(ebit * tax_rate)
    np.clip(tax, a_min=None, a_max=0)
    nwc = nwc_percent_dist * sales
    starting_nwc = starting_sales * nwc_percent
    prev_year_nwc = np.roll(nwc, 1, axis=0)
    prev_year_nwc[0] = starting_nwc
    delta_nwc = prev_year_nwc - nwc
    capex = -(sales * capex_percent)
    free_cash_flow = ebitda + tax + delta_nwc + capex
    # Discount cash flows to get DCF value
    terminal_value = free_cash_flow[-1] * (1 + g) / (r - g)
    discount_rates = [(1 / (1 + r)) ** i for i in range(1,6)]
    dcf_value = sum((free_cash_flow.T * discount_rates).T)
    dcf_value += terminal_value * discount_rates[-1]

    return dcf_value
```

The main difference in the versions is the absence of “for i in range(iterations)” loop. Using NumPy’s array, this new version runs faster compared to the old one. (Look at wall time)

```
In [63]: %time plt.hist(run_mcs(), bins=20, density=True, color="y")
plt.show()
```

Wall time: 23.7 ms



€m	Actual			Projections				
	2016	2017	2018	2019	2020	2021	2022	2023
Income Statement								
Sales	25.6	28.1	31.0	34.1	37.5	41.3	45.4	49.9
% growth	11.3%	9.8%	10.3%	10.0%	10.0%	10.0%	10.0%	10.0%
EBITDA	3.4	4.1	4.6	4.8	5.3	5.8	6.4	7.0
EBITDA %	13.3%	14.6%	14.8%	14.0%	14.0%	14.0%	14.0%	14.0%
Depr. & Amort.	0.8	0.9	1.0	1.1	1.2	1.3	1.5	1.6
EBIT	2.6	3.2	3.6	3.7	4.1	4.5	4.9	5.4
EBIT %	10.1%	11.4%	11.6%	10.8%	10.8%	10.8%	10.8%	10.8%
Balance Sheet								
Net Working Capital	5.6	6.7	7.4	8.2	9.0	9.9	10.9	12.0
NWC % of Sales	22%	24%	24%	24%	24.0%	24.0%	24.0%	24.0%
Cash-Flow Statement								
EBIT	2.6	3.2	3.6	3.7	4.1	4.5	4.9	5.4
Adj. Taxes	(0.6)	(0.8)	(0.9)	(0.9)	(1.0)	(1.1)	(1.2)	(1.3)
Depr. & Amort.	0.8	0.9	1.0	1.1	1.2	1.3	1.5	1.6
Capex	(0.8)	(0.9)	(1.1)	(1.1)	(1.2)	(1.3)	(1.5)	(1.6)
Capex % of Sales	3.0%	3.2%	3.4%	3.2%	3.2%	3.2%	3.2%	3.2%
Change in NWC	(0.4)	(1.1)	(0.7)	(0.7)	(0.8)	(0.9)	(1.0)	(1.1)
Free Cash Flow	1.6	1.3	1.9	2.0	2.2	2.4	2.7	3.0