Financial Modeling with Python

Part 1 – Mortgage Payment and Amortization Schedule

In this session, we will learn how to calculate the monthly payments of a mortgage and generate an amortization schedule using **NumPy** package and **Pandas** package, and create charts using **Matplotlib** package.

By finishing this session, you will be able to:

- Use the **pmt()**, **ppmt()**, **ipmt()** functions to calculate the monthly payment, principal payment and interest payment of the mortgage.
- Apply the basic features of Pandas' data frame.

Importing Necessary Packages

We have to import them in our python code. Add the following code to the beginning of your python file:

```
import pandas as pd
import numpy as np
from datetime import date
```

Calculating Mortgage Payment and Interest

We are going to learn how to calculate the monthly payment of a 30 year $5 million mortgage with annual interest rate of 2.5%.

First, we define the variables for the mortgage. Assume that the starting date of the periods is 1st July 2018.

```
interest_rate = 2.5 / 100 / 12
nper = 30 * 12
pv = 5000000
start_date = date(2018, 7, 1)
```

Monthly Payment

To calculate the monthly payment, the following formula is used:

$$Payment = Principal\_value \times \left(\frac{interest\_rate \times (1 + interest\_rate)^{num\_of\_periods}}{(1 + interest\_rate)^{num\_of\_periods} - 1}\right)$$

But, **Numpy** provides a function named **pmt()** that allows us to get the result directly. The syntax is:

```
result = np.pmt(rate, num_of_periods, principal_value)
```

Let’s use our example to calculate the monthly payment using the **pmt()** function. The following statement is as follows:

```
pmt = np.pmt(interest_rate, nper, pv)
```

To make the result clearer, we round the result off to 2 decimal places:

```
pmt = round(np.pmt(interest_rate, nper, pv), 2)
```

We get *-19756.04*. It indicates that the monthly payment of *$19,756.04* would be required.

The monthly payment includes principal payment and interest payment. The monthly payment stays constant over the time but the amount applied to principal increases and the interest decreases as we move forward in time.

Principal Payment and Interest Payment

The principal payment can be calculated using the **Numpy’s ppmt()** function. The syntax is:

```
result = np.ppmt(rate, i, num_of_periods, principal_value)
```

Where *i* indicates the specific period.

The following is the example to calculate the principal payment of the 1st period:

```
ppmt = np.ppmt(interest_rate, 1, nper, pv)
```

By the way, we round the result off to 2 decimal places:

```
ppmt = round(np.ppmt(interest_rate, 1, nper, pv), 2)
```

Then, we get *-9339.38*. That means only $9,339.38 in the payment of the 1st period belongs to the principal payment. Others would be the interest payment.

The interest payment would be equal to:

$$Interest\_Payment = Payment - Principal\_Payment$$

We can get the interest payment using a simple subtraction as the formula above, we do:

```
ipmt = pmt - ppmt
```

However, if you do not know the amounts of the payment and the principal payment, you can compute it by using **Numpy’s ipmt()** function directly. The syntax is:

```
ipmt = np.ipmt(rate, i, periods, principal_value)
```

*Note: the **np.ipmt()** function returns the result as an **ndarray**. You may convert the result by using the **float()** function.*

Let’s put everything together to get the amounts of the payment, principal payment and interest payment of **the first month**.

```
#%%
import pandas as pd
import numpy as np
from datetime import date

interest_rate = 2.5 / 100 / 12
nper = 30 * 12
pv = 5000000
start_date = date(2018, 7, 1)

pmt = round(np.pmt(interest_rate, nper, pv), 2)

ppmt = round(np.ppmt(interest_rate, 1, nper, pv), 2)

ipmt = round(float(np.ipmt(interest_rate, 1, nper, pv)), 2)

print("Payment: ", pmt)
print("Principal Payment: ", ppmt)
print("Interest Payment: ", ipmt)
```

Creating an amortization schedule

An amortization schedule is a table detailing each periodic payment on an amortizing loan, such as a mortgage. It includes payment numbers, payment dates, monthly payment amounts, principal and interest payment amount of each period, balances, cumulative principal amounts, etc.

The following is the sample of amortization schedule:

Period	Payment Date	Payment	Principal	Interest	Balance	Cumulative Principal
1	2018-07-01	-19756.04	-9339.38	-10416.67	4990660.62	-9339.38
2	2018-08-01	-19756.04	-9358.84	-10397.21	4981301.79	-18698.21
3	2018-09-01	-19756.04	-9378.33	-10377.71	4971923.45	-28076.55
4	2018-10-01	-19756.04	-9397.87	-10358.17	4962525.58	-37474.42
5	2018-11-01	-19756.04	-9417.45	-10338.59	4953108.13	-46891.87
6	2018-12-01	-19756.04	-9437.07	-10318.98	4943671.06	-56328.94
7	2019-01-01	-19756.04	-9456.73	-10299.31	4934214.33	-65785.67
8	2019-02-01	-19756.04	-9476.43	-10279.61	4924737.90	-75262.10
…	…	…	…	…	…	…

Pandas' DataFrame can be used to build this kind of table. We now follow the steps below to create an amortization schedule:

1. We create a code cell and define some parameters:

```
#%%
import pandas as pd
import numpy as np
```

```
from datetime import date

interest_rate = 2.5 / 100 / 12
nper = 30 * 12
pv = 5000000
start_date = date(2018, 7, 1)
```

2. We first create a list of payment dates. The payment starting date is 1st July 2018, and the mortgage duration is 30 years. Assume that we make a payment every month, there are total 360 payments. Therefore, the list of payment dates contains 360 dates from 1st July 2018 to 1st June 2048. We create the list using the following statement:

```
date_range = pd.date_range(start_date, periods=nper, freq='MS')
```

3. The table has six columns – Payment Date, Payment, Principal, Interest, Balance, and Cumulative Principal. The "Period" is the index of the table, we will ignore this column at this moment. We create the data frame (the table) by using the following statement:

```
df = pd.DataFrame(columns=['Payment Date', 'Payment', 'Principal', 'Interest', 'Balance',
'Cumulative Principal'])
```

4. Then, we assign the list date_range to the data frame.

```
df['Payment Date'] = date_range
```

Now, the data frame contains the data as follows:

	Payment Date	Payment	Principal	Interest	Balance	Cumulative Principal
0	2018-07-01	NaN	NaN	NaN	NaN	NaN
1	2018-08-01	NaN	NaN	NaN	NaN	NaN
2	2018-09-01	NaN	NaN	NaN	NaN	NaN
3	2018-10-01	NaN	NaN	NaN	NaN	NaN
4	2018-11-01	NaN	NaN	NaN	NaN	NaN
5	2018-12-01	NaN	NaN	NaN	NaN	NaN
…	…	…	…	…	…	…

5. The first column is the index column that starts from 0. We reset the indices in the range from 1 to 360 (30 years x 12 months = 360 periods). The following lines are used to a new column named "Period" and set it as index:

```
df['Period'] = range(1, nper + 1)
df.set_index('Period', inplace=True)
```

The following is the updated view of the data frame:

Period	Payment Date	Payment	Principal	Interest	Balance	Cumulative Principal
1	2018-07-01	NaN	NaN	NaN	NaN	NaN
2	2018-08-01	NaN	NaN	NaN	NaN	NaN

3	2018-09-01	NaN	NaN	NaN	NaN	NaN
4	2018-10-01	NaN	NaN	NaN	NaN	NaN
5	2018-11-01	NaN	NaN	NaN	NaN	NaN
6	2018-12-01	NaN	NaN	NaN	NaN	NaN
…	…	…	…	…	…	…

6. We assign the values for the Payment, Principal and Interest columns by using **Numpy's pmt(), ppmt()** and **ipmt()** function:

```
df['Payment'] = np.pmt(interest_rate, nper, pv)
df['Principal'] = np.ppmt(interest_rate, 1, nper, pv)
df['Interest'] = float(np.ipmt(interest_rate, 1, nper, pv))
df = np.around(df, 2)
```

Then, we get the following data frame:

Period	Payment Date	Payment	Principal	Interest	Balance	Cumulative Principal
1	2018-07-01	-19756.04	-9339.38	-10416.67	NaN	NaN
2	2018-08-01	-19756.04	-9339.38	-10416.67	NaN	NaN
3	2018-09-01	-19756.04	-9339.38	-10416.67	NaN	NaN
4	2018-10-01	-19756.04	-9339.38	-10416.67	NaN	NaN
5	2018-11-01	-19756.04	-9339.38	-10416.67	NaN	NaN
6	2018-12-01	-19756.04	-9339.38	-10416.67	NaN	NaN
…	…	…	…	…	…	…

7. But, we know that the amounts of Principal and Interest should not be constant over the time. Therefore, we rewrite the code with a loop for computing the Principal Payment and Interest Payment for each row.

```
df['Payment'] = round(np.pmt(interest_rate, nper, pv), 2)

for i in range(1, nper + 1):
   df.loc[i, 'Principal'] = round(np.ppmt(interest_rate, i, nper, pv), 2)
   df.loc[i, 'Interest'] = df.loc[i, 'Payment'] - df.loc[i, 'Principal']
```

8. We can simplify the code above as follow as:

```
df['Payment'] = np.pmt(interest_rate, nper, pv)
df['Principal'] = np.ppmt(interest_rate, df.index, nper, pv)
df = np.around(df, 2)
df['Interest'] = df['Payment'] - df['Principal']
```

The line **df = np.around(df, 2)** is used to round the float down to two decimal places.

9. Then, we compute the cumulative principal. Consider the following formula for computing the cumulative principal for each row:

$$Cum\_Principal_{\ i} = Cum\_Principal_{i-1} - Principal\_Payment_i$$

$$Cum\_Principal_0 = 0$$

But, we use the **cumsum()** function instead.

```
df['Cumulative Principal'] = df['Principal'].cumsum()
```

10. Next, we compute the balance for each row. It can be computed using a simple subtraction as follows:

$$Balance_{\ i} = Principal\_value - CumPrincipal_{\ i}$$

The python code is as follows:

```
df['Balance'] = pv + df['Cumulative Principal']
```

We then have the data frame as follows:

Period	Payment Date	Payment	Principal	Interest	Balance	Cumulative Principal
1	2018-07-01	-19756.04	-9339.38	-10416.67	4990660.62	-9339.38
2	2018-08-01	-19756.04	-9358.84	-10397.21	4981301.78	-18698.22
3	2018-09-01	-19756.04	-9378.33	-10377.71	4971923.45	-28076.55
…	…	…	…	…	…	…
359	2048-05-01	-19756.04	-19673.98	-82.06	19714.9	-4980285.12
360	2048-06-01	-19756.04	-19714.97	-41.07	-0.09	-5000000.09

11. We find that the balance shown in the last row is a negative value and the cumulative principal is more than the principal value. We need to correct them by adding additional statements after and outside the for-loop.

```
df.loc[nper, 'Principal'] -= df.loc[nper, 'Balance']
df.loc[nper, 'Balance'] = 0
df.loc[nper, 'Cumulative Principal'] = df.loc[nper - 1, 'Cumulative Principal'] + \
df.loc[nper, 'Principal']
df.loc[nper, 'Payment'] = df.loc[nper, 'Principal'] + df.loc[nper, 'Interest']
```

Now, we get a corrected data:

Period	Payment Date	Payment	Principal	Interest	Balance	Cumulative Principal
1	2018-07-01	-19756.04	-9339.38	-10416.67	4990660.62	-9339.38
2	2018-08-01	-19756.04	-9358.84	-10397.21	4981301.78	-18698.22
3	2018-09-01	-19756.04	-9378.33	-10377.71	4971923.45	-28076.55
…	…	…	…	…	…	…
359	2048-05-01	-19756.04	-19673.98	-82.06	19714.9	-4980285.12
360	2048-06-01	-19756.04	-19714.88	-41.07	0	-5000000.00

12. Add the **print()** function to print the final data frame, and check the complete code below:

```
#%%
import pandas as pd
import numpy as np
from datetime import date
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

interest_rate = 2.5 / 100 / 12
nper = 30 * 12
pv = 5000000
start_date = date(2018, 7, 1)

date_range = pd.date_range(start_date, periods=nper, freq='MS')

df = pd.DataFrame(columns=['Payment Date', 'Payment', 'Principal', 'Interest',
   'Balance', 'Cumulative Principal'])

df['Payment Date'] = date_range

df['Period'] = range(1, 361)
df.set_index('Period', inplace=True)

df['Payment'] = np.pmt(interest_rate, nper, pv)

df['Principal'] = np.ppmt(interest_rate, df.index, nper, pv)

df = np.around(df, 2)

df['Interest'] = df['Payment'] - df['Principal']

df['Cumulative Principal'] = df['Principal'].cumsum()
df['Balance'] = pv + df['Cumulative Principal']

df.loc[nper, 'Principal'] -= df.loc[nper, 'Balance']
df.loc[nper, 'Balance'] = 0
df.loc[nper, 'Cumulative Principal'] = df.loc[nper - 1, 'Cumulative Principal'] + \
df.loc[nper, 'Principal']
df.loc[nper, 'Payment'] = df.loc[nper, 'Principal'] + df.loc[nper, 'Interest']

print(df)
```

Exercise: Mortgage Schedule Function

In the previous part, we have written a python code to build an amortization schedule for the specific principal value, annual interest rate, and duration.

To make it reusable and flexible, you need to convert the code to a function that accepts the principal value, annual interest, and duration then returns a schedule in **Pandas' DataFrame** format.