**Analysis of factors influencing the formation of the provision**

* Introduction
  + Document properties
  + Connected documents
  + Purpose
  + Limitations and assumptions
  + Description of covenants and functions
* Functional requirements
  + Downloading and processing data
  + Dealing with foreign currency loans
  + Resources
  + Parameters of the function
  + Working procedure
  + Description of the fields
  + Description of the technical fields
* Non-functional requirements
* Requirements for regulations
* Data quality requirements

**Introduction**

**Document properties**

|  |  |
| --- | --- |
| **Name** | Analysis of factors influencing the formation of the provision |
| **Version of document** | 1.0 |
| **Creator** | Krauze Natalia Olegovna |
| **Status** | New |

**Connected documents**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Name of document and/or reference | Document version | Description |
| 1 | *Формула расчета резерва - задание по разработке методики расчета.docx* | 2 | Description of how provisions are calculated |

**Purpose**

The purpose of the window is to enable calculation of the dynamics of loan loss provisions for clients who conclude contracts with the bank on the basis of IFRS and Basel 2. The resulting showcase helps to dynamically trace the impact of provisioning factors over time.

**Limitations and assumptions**

1. Consider the period for calculating the imputation of reserves for the last 6 months. That is, for the last 6 months of the current year.
2. The calculation of the change in provisions is counted and analyzes by the company once a month.
3. Dataset must be in csv/xslx format to work with the function.
4. No extraneous characters are allowed in the databank - such as words, letters, sign - only numbers that can be read by the python.
5. The date format for working with libraries and functions should be strictly 'YYYYY-MM-31'.

**Description of packages and functions**

Using the python package [pycbrf.toolbox](https://pythonz.net/apps/named/pycbrf/)

With pycbrf you can retrieve information provided by the central bank of the Russian Federation, e.g.: exchange rates, bank data. We use the pycbrf library to access the Central Bank API.

You can use it as a command line utility, or you can use it from your Python application.

Example of usage:

|  |  |
| --- | --- |
| **Code** | **Output** |
| '''  c = CurrencyRates()  amount = 1  date = datetime(2021,4,6)  print(c.convert('USD','RUB', amount, date))  ''' | 76.6232644768 |

**Functional requirements**

**Downloading and processing data**

A dataset with data for the last 6 months in csv format is accepted as input.

Read out the file and process it for possible errors:

|  |  |
| --- | --- |
| Type of error in the data | Problem solution |
| Missing values | 1. for numerical values, this means in our case that the number is 0 - and this counts towards the total, so we do not delete it (see last 4 columns data)   ***SUMRESERVEMAINRPBU*** *(Amount of allowance for possible main debt losses)*  ***SUMRESERVEPRMNRPBU*** *(amount of allowance for possible losses of overdue principal)*  ***SUMRESERVEPRMN*** *(Amount of allowance for principal debt interest)*  ***SUMRESERVEPRPR*** *(Sum of Provision for Interest on Overdue Principal Debt)*   1. For categorical variables, replace with the most popular value |
| Checking for outliers | Remove outlays from the table so that they do not affect the reserve |

**Functions**

1. **Function get\_currency**

Since we have currency loans - you need to search the Central Bank website <https://www.cbr.ru/currency_base/> and pull up the currency data as of a certain date.

We do this using the pycbrf.toolbox library in python.

|  |  |
| --- | --- |
| **Code** | **Output** |
| '''  c = CurrencyRates()  amount = 1  date = datetime(2021,4,6)  print(c.convert('USD','RUB', amount, date))  ''' | 76.6232644768 |

Check result with <https://www.cbr.ru/currency_base/daily/?UniDbQuery.Posted=True&UniDbQuery.To=31.03.2020>

**Resources**

|  |  |
| --- | --- |
| **Table** | **Description** |
| Loanphys.csv | Loan portfolio of individuals |

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |
| 2 | currency | The value of the currency we want |

**Working procedure**

The function goes through the dataset, looks at the value of the currency on a certain date and pairs the Central Bank site to find out the currency, if the currency is in Rubles, the value is set to 1.

1. **Breakdown by number of credits**

As the dataset can store a different number of credit dates, it is important to correctly count the values for each number of credits. so I use pandas to split the dataset and sort it by number of credits. Deals with the same number are stored in a separate sheet for later use.

**Result**

|  |  |
| --- | --- |
| **Output** | **Description** |
| credits\_dfs[0] | Client transactions with 1 date only |
| credits\_dfs[1] | Client transactions with 2 dates only |
| credits\_dfs[2] | Client transactions with 3 dates only |
| credits\_dfs[3] | Client transactions with 4 dates only |
| credits\_dfs[4] | Client transactions with 5 dates only |
| credits\_dfs[5] | Client transactions with 6 dates only |

1. **Function that counts provision**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| Value is the sum of the reserve fields | 'SUMRESERVEMAINRPBU' + SUMRESERVEPRMNRPBU' + 'SUMRESERVEPRMN' + 'SUMRESERVEPRPR' |

1. **Function that counts delta\_provision**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| First value always is the same as the provision on that row | deltas = [tmp\_df.iloc[0]['PROVISION']] |
| All subsequent values are the amount of provision in that term minus the previous one | tmp\_df.iloc[i-1]  init\_row = tmp\_df.iloc[i]  deltas.append(init\_row['PROVISION'] - prev\_row['PROVISION']) |

1. **Function that counts dA**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| First value always is the sum of 2 values in that row | **MAINRESTCUR + PRMNRESTCUR** |
| All subsequent values are the sum of the current MAINRESTCUR + PRMNRESTCUR values (in this row) minus the previous sum of the same values | **(MAINRESTCUR\_(n+1) + PRMNRESTCUR\_(n+1)) – (MAINRESTCUR\_n + PRMNRESTCUR\_n)** |

1. **Function that counts dC**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| First value always is the same as the currency on that row | tmp\_df = df[df['DEAL\_ID'] == credit\_id]  tmp\_df.reset\_index().drop(columns = ['index'])  **dC\_s = [0]** |
| All subsequent values are the amount of currency in that term minus the currency one | tmp\_df.iloc[i-1]  tmp\_df.iloc[i-1]  init\_row = tmp\_df.iloc[i]  dR\_s.append(init\_row[' CUR\_RATE'] - prev\_row['CUR\_RATE']) |

1. **Function that counts dR**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| First value always is the same as the rate on that row | tmp\_df = df[df['DEAL\_ID'] == credit\_id]  tmp\_df.reset\_index().drop(columns = ['index'])  **dR\_s = [0]** |
| All subsequent values are the amount of rate in that term minus the rate one | tmp\_df.iloc[i-1]  init\_row = tmp\_df.iloc[i]  dR\_s.append(init\_row['RESERVERATERPBU'] - prev\_row['RESERVERATERPBU']) |

1. **Function that counts dPA**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| This value for each date is calculated separately - that is, the value in this row is multiplied by dA by the value of the currency and by the RESERVERATERPBU rate and everything is divided by 100 to get rid of percentage | next\_row = tmp\_df.iloc[i+1]  init\_row = tmp\_df.iloc[i]  dPA\_s.append(next\_row['dA'] \* (init\_row['RESERVERATERPBU'] / 100) \* init\_row['CUR\_RATE']) |

1. **Function that counts dPR**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| First value always is 0 | tmp\_df = df[df['DEAL\_ID'] == credit\_id]  tmp\_df.reset\_index().drop(columns = ['index'])  dPR\_s = [0] |
| the value equals the sum of MAINRESTCUR and PRMNRESTCUR for that date in the row, all multiplied by the currencies on that date according to the CB and multiplied by the rate change on that day and the previous day, to get rid of the percentage | init\_row = tmp\_df.iloc[i]  dPR\_s.append((init\_row['MAINRESTCUR'] + init\_row['PRMNRESTCUR']) \* (init\_row['dR'] / 100) \* init\_row['CUR\_RATE']) |

1. **Function that counts dPC**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| First value always is 0 | tmp\_df = df[df['DEAL\_ID'] == credit\_id]  tmp\_df.reset\_index().drop(columns = ['index'])  dPC\_s = [0] |
| the value equals the currency change on that day and the previous day, multiplied by RESERVERATERPBU on that date, multiplied by the sum of MAINRESTCUR and PRMNRESTCUR for that day and divided by 100, to get rid of the percentage | init\_row = tmp\_df.iloc[i]  dPC\_s.append(((init\_row['MAINRESTCUR'] + init\_row['PRMNRESTCUR']) \* init\_row['RESERVERATERPBU'] \* init\_row['dC']) / 100)] \* init\_row['dC']) / 100) |

1. **Function that counts dXR**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| The value is counted as of the specified date in the row for each separately cumulative total from the previous data frame by subtracting values such as dPA, dPR and dPC from dProvison | for column in ['dPA', 'dPR', 'dPC']:  df['dXR'] -= df[column] |

1. **Function that counts deltas**

**Parameters of the function**

|  |  |  |
| --- | --- | --- |
|  | **Parameter** | **Parameter value formula** |
| 1 | df | Dataset for the period |

**Working procedure**

|  |  |
| --- | --- |
| **Description of the fields** | **Value** |
| This is a summary function, which in turn calls the functions above in the correct order to obtain the correct values for each trade. | df = count\_provision(df)  df = count\_delta\_provision(df)  df = count\_dA(df)  df = count\_dC(df)  df = count\_dR(df)  df = count\_dPA(df)  df = count\_dPR(df)  df = count\_dPC(df)  df = count\_dXR(df) |

In this function we learn for each data frame (we have 6 for each number of days), so we call it 6 times and get 6 calculated data frames with the necessary values to calculate the factors of the reserve change and fill in the summary table.

These datasets can then be merged into one common dataset by grouping by date for each deal.

Изображение выглядит как стол

Автоматически созданное описание

**Non-functional requirements**

**Requirements for regulations**

The provisioning table is calculated monthly, the calculation for the previous 6 months is made on the 1st of the following month. The table contains monthly data cuts starting from 31.03.2020.

The report does not include information for the calendar month not completed.

**Data quality requirements**

csv / xslx table with separator ';'.