Giulia Secoli

Student ID: 13707617

gsecol01@student.bbk.ac.uk

BSc Computer Science Project

Data Science and Computing

“Automated Risk Assessment For Italian Companies Applying For a Surety Bond to Trade, on the Italian Market”

Introduction

Surety Bond Trading (S.B.T.) is a project that aims to automate the Insurance underwriting process by providing underwriters with a tool that can autonomously assess the risk level of a requested bond. S.B.T. considers various factors such as bond duration, bond value, and the size of the applicant's company, as well as their financial performances and trends, to determine the maximum credit limit that the company can borrow. S.B.T. then assigns a risk score ranging from 0 to 100, with higher scores indicating higher risk. Additionally, the software suggests the appropriate level of experience of the underwriter based on the final risk score, with more experienced underwriters being assigned to higher-risk quotations. As the risk level increases, the underwriting approach shifts from junior underwriters to board meetings to ultimately not approving the bond.

S.B.T. offers the functionality to assess the creditworthiness of a co-obligor, providing an adjusted security value that represents the amount in Euro that the Insurance company can get back from the co-obligor if the primary obligor fails to fulfill their obligation. This functionality helps to determine what is the real value of a co-obligor in relation to the requested quotation. This information can be saved on the entity's profile for future use or to gain insight into the entity's financial worth.

Background

Italy boasts one of the world's largest surety markets in the world, with approximately €650M of premium per annum. Surety bonds are a legal requirement for Italian companies trading in the market, making them an essential guarantee. The market has a long-standing demand that exceeds supply. The 'Surety' line of business' showed an impressive 14.5% increase in 2021, compared to a 2.8% increase in the entire insurance market. Post-pandemic, Italy is receiving a significant amount of grants from the EU, and as a result, the demand for surety bonds is expected to grow faster in the next few years than in previous years, as was already observed in 2021.

However, the Italian surety market remains opaque, with inconsistent risk assessment methodologies and underwriting review processes. This is where S.B.T. can make a significant impact by providing a consistent and objective approach to risk assessment and underwriting review. S.B.T. can reduce potential biases and subjectivity in the process, ensuring that all companies are evaluated fairly and accurately based on their financial health and risk profile. S.B.T. model can be tailored to different underwriter needs and risk appetite. This flexibility makes it an ideal solution for insurance companies looking to improve their risk assessment methodologies, ultimately creating a fair and equal opportunity for all companies seeking a Surety bond in the Italian market.

As a Surety Underwriter working in the London Insurance industry for over six years, I can state that the current underwriting process is inefficient and prone to errors. This is because it requires a human underwriter to sift through documents to determine if an applicant is qualified to borrow. This involves analyzing various data, such as obligor's financial statements, co-obligor Land Registry report and Hypothecary Registry report.

In practice, the work of an insurance underwriter typically involves reviewing applications. Underwriters receive applications from brokers or directly from applicants. The papers required to submit a quotation can be submitted by either the broker or the applicant directly. The underwriter examines the application to ensure that it is complete and that all relevant information has been provided. However, it is common for the documentation to be incomplete, which often requires the underwriter to communicate with the correspondent party to ask for missing documentation or further document integration. Documentation is often delayed, or sometimes, also being manipulated. Malpractice aimed at increasing the likelihood of the applicant to be approved for the requested coverage.

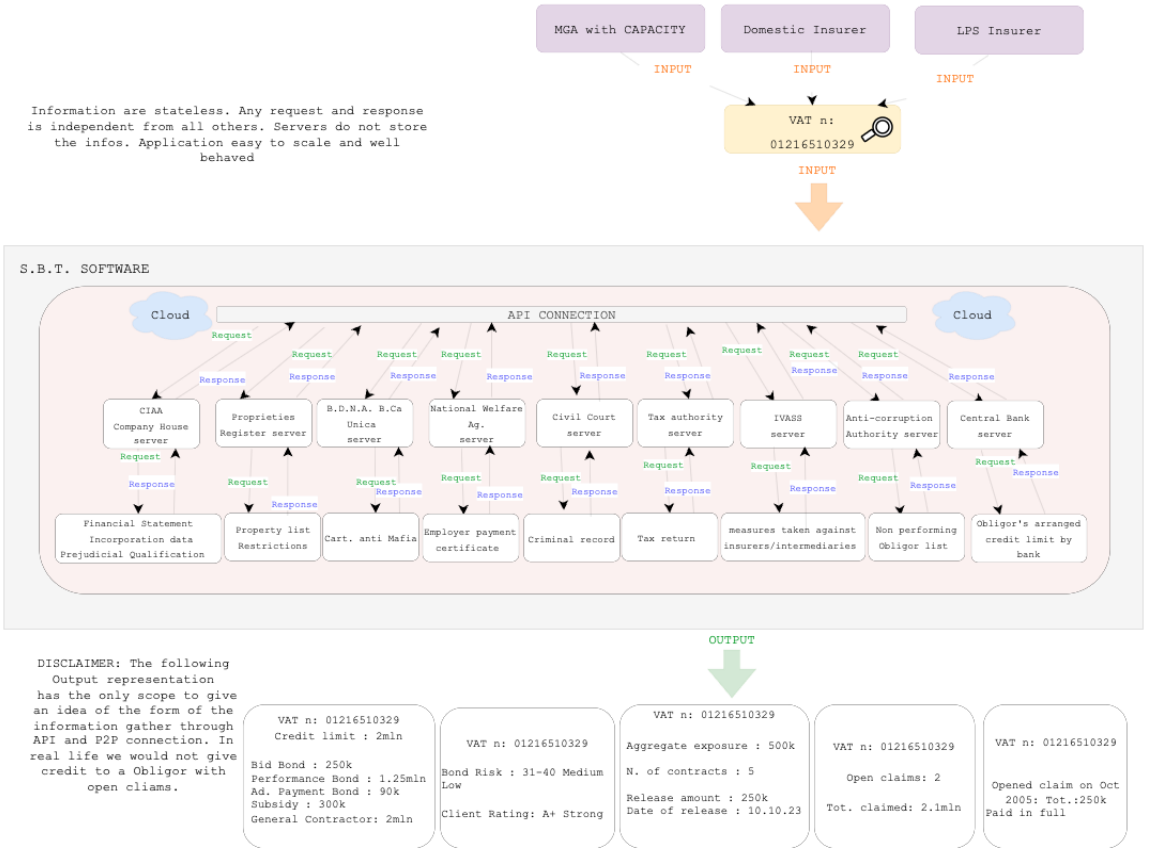
As a result, technology can help reduce the risk of documentation inconsistency and transparency. One such tool is the use of application program interfaces (APIs) which can allow for automated and secure transfer of information between different systems, reducing the likelihood of errors and delays in manual data entry. To achieve these goals, the idea behind S.B.T. is to use API connections that connect the software with major Italian data providers where relevant financial information is held. When an underwriter initiates an API call, the application retrieves all the available information for the given VAT number. The information retrieved would be stateless, meaning that any request and response is independent from all others, and the servers do not store the information. This approach makes the application easy to scale and well-behaved. To retrieve official data on Italian companies, their legal status, ownership, and financial statements, the Italian Company House (Camera di Commercio) is the public institution that stores this info.

However, I have requested access to their servers but as a student or non-corporate entity, unfortunately I do not have the necessary authorization and permission to connect to their servers, resulting in my request being rejected. As a solution, the project will retrieve data from official XML files provided by the Italian Company House, stored locally for Python to read.

The insurance underwriter is a crucial player in the risk management process. An underwriter should ensure that only eligible candidates receive the requested coverage. Failing to accurately assess risk can lead to financial losses for both the insurer and the insured, as well as the possibility of legal issues and reputational damages. Bigger losses can have broader economic implications, such as creating market instability and hindering investment opportunities, as well as disruptions to national projects. Also, granting EU subsidies to performing companies over non-performing companies.

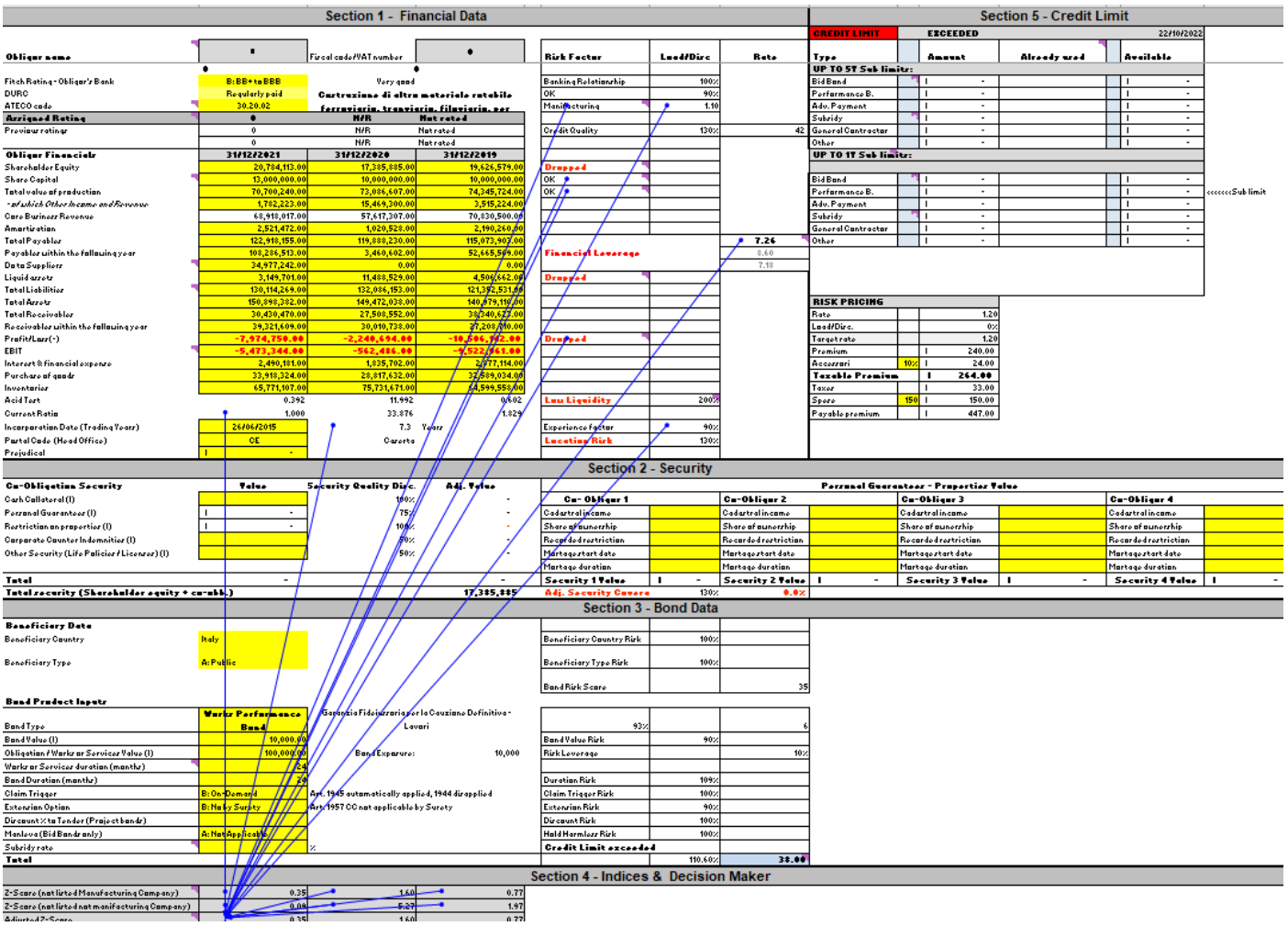
Therefore, the work of an underwriter is critical in maintaining low market loss ratios for the sector. The average loss ratio for the Italian market, for the Surety business, is roughly 45%. The future of underwriting involves risk-pricing models that replace most of the work that underwriters do, by reducing the need for paperwork and increasing automation through more sophisticated computational techniques.

Method: The method of insurance underwriting involves assessing the risk exposure and determining the premium that should be charged to insure the risk. This process requires a significant amount of data to evaluate the financial position of the applicant. Traditionally, this information has been provided by brokers or by the applicant themselves. Data include, financial statements, tax returns, property or asset values, employer payment certificates, and more.



An example of the current evaluation model is the excel-based model shown below. From the picture the interconnections between each field is visible. The current excel-based model is divided into sections. The yellow fields ate input data that needed to be entered manually. Specifically:

- **Obligor name :** Company name, description and Ateco code, Incorporation date
- **Obligor Financials :** Financials for the last three years of account
- **Co-Obligation security :** Cadastral Income, share of property ownership, recorded restrictions on properties, mortgage start date, mortgage duration (we will not consider cash collateral in this project. Typically, cash collateral is valued at 100% of its value)
- **Bond Product Inputs** : Bond type, Bond value, Bond duration and Works value



1.1. Scope: Surety Guarantees

A Surety bond is a unique type of insurance that guarantees a payment to the creditor (beneficiary) if the policyholder/obligor fails to do so for any reason, essentially taking on the risk of the Obligor's default. Three parties are involved in this contract:

- The **policyholder** or obligor, typically a company, who requests the surety bond.
- The **beneficiary** or creditor, usually a public entity, to whom the service or project must be provided, who can request a guarantor's presence for added security and can call the bond (make a claim)
- The **co-obligor** (guarantor), usually an individual, that acts as a guarantor to the contractor and assumes responsibility for paying the debt in case the main obligor defaults on its debt or its obligations.

The Insurance company is responsible for evaluating the financial stability of the policyholder by verifying their income capacity and economic solvency through a review of relevant documents. If the applicant has other outstanding loans or debts, the request will be deemed more risky and it may be rejected altogether. Vice-versa if the contractor's reliability has been confirmed, a surety policy will be issued. One popular type of surety insurance is for public tender projects.

The Beneficiary is the entity who can call the bond. When a bond is called, the insurance company is responsible for paying out the losses to the beneficiary, who has suffered damages due to the obligor's failure to fulfill their obligations under the bond terms and conditions. The insurance company may then seek to recover the amount paid out to the Beneficiary, from the co-obligor, due to the primary obligor's default. Typically, the action to recover the losses is directed towards the co-obligor because the primary obligor, who is responsible for the proper execution of the contract, has defaulted and is unable to repay

its debts. Typically, the guarantor is an individual who takes on the responsibility of paying off the debt if the contractor is unable to do so, thus providing an additional layer of security for the beneficiary.

To determine the financial capability of a guarantor, we conduct an evaluation of their assets and calculate their adjusted value by subtracting any debts or outdated valuations on properties.

1.2. Current Model

The current underwriting model is an Excel file that continuously evolves. The Excel model has been developed over the course of several years through extensive analysis of past insurance policies, claims, past policyholder's performance. In addition, I regularly participate in weekly underwriting and claims sessions with senior underwriters at our company to learn from past claims and identify factors with a higher correlation to the likelihood of a company defaulting on a surety bond. As a result, it has become a valuable asset that confers a notable competitive advantage in the surety sector.

The presented project will closely simulate the Excel-based model functionalities while taking an important step forward in automating the risk assessment and improving efficiency, performance and transparency. One of the key objectives is to create a program that elevates the existing Excel-based model, which is considered 'pretty' revolutionary in the industry I work in (at the time this paper was written) by developing a full-stack application that integrates both back-end and front-end functionalities. The objective is to achieve the same results as the Excel model but with the added efficiency and benefits of using programming languages. To maintain the confidentiality of the proprietary knowledge and expertise that forms the core of the existing Excel-based model, the new model will not exactly replicate its functions. Instead, it will incorporate similar or slightly incomplete functions to safeguard the original formula.

S.B.T. is a project idea that originated while I was studying for the CII's Insurance London Market exam, through my workplace. Through this experience, I became aware of the big differences in process automation between the Italian and London insurance markets. In London, Lloyd's market uses a software called Crystal, to issue policies, manage claims, payments, and to access relevant documentation. This does not exist in Italy yet. However, both the London and Italian market still rely heavily on manual underwriting, which presents a significant deficiency.

S.B.T. aims to bridge this gap by automating the underwriting process and introducing efficiencies to the Italian insurance market.

The purpose of S.B.T. is to provide the software to all licensed 'market players' worldwide, who are authorized by the 'Insurance Italian regulator, IVASS', to underwrite Surety bonds in the Italian market.

The excel model, covers and consists of six main sections:

1. Information about the Obligor
2. Financial statements of the Obligor
3. Information about the type of co-obligations offered (Co-obligation Security)
4. Information about the specific type of Surety guarantee required
5. Assessment of the decision maker's level of competence (Underwriter Assessment)
6. Assignment of Credit Limit

The excel model prioritizes creditworthiness when evaluating the type of guarantee requested. A low-risk policyholder is more attractive, even at a lower premium rate, as they have a lower risk of default while a high-risk obligor who is willing to pay a higher premium rate is considered unattractive business as they don't meet the protocol standards.

To this end, the examination of financial information aims to identify the following:

- The variation in revenue, operating results, and shareholders equity over the past two years of account
- The extent of financial leverage
- The level of liquidity and immediate availability of funds
- Key performance indicators that reveal important information about a company's financial health

The model evaluates all information entered and generates an output that can have a positive or negative effect on the final risk of the bond. The model

generates a final score for the quotation being requested, which subsequently determines the overall risk level of the request and the level of internal decision-making assigned by competence to those who can approve the risk. The decision-making levels range from Junior Underwriter to Underwriter, Senior Underwriter, Underwriter Manager, and ultimately the Board.

In addition, the model assigns a Credit Limit which includes:

- The maximum Credit Limit that can be granted to the Policyholder (Obligor).
- 5 Sub-limits each representing a major sub-category of the Surety bonds, such as Bid Bond, Housing, Advance Payment Bond, Subsidy, General Contractor, and Others.

1.3. Requirements

To eliminate potential errors and inefficiencies, the current manual input of data into the Excel file needs to be automated. To achieve this goal, the program must fulfill specific requirements that have been identified for this purpose.

1.3.1. Development of a Backend System for Data Analysis

The first requirement to automate the process is the development of an efficient backend system capable of processing large volumes of data with speed and accuracy. The system must also be designed for easy maintenance. In addition, a relational database must be created to store the data from multiple sources and those entered manually from the front-end. For this purpose, MySQL has been selected for its ability to connect and manage data effectively. The backend system will be developed using the Python language, which offers a clear syntax and the ability to execute complex logic with minimal instructions. It also facilitates seamless communication with the database.

1.3.2. Retrieval of data

The front-end interface and the Python computations are integrated to work together in retrieving and outputting data. When a user clicks on the "submit" button in the front-end interface, it initiates a call to the backend. The algorithms reads in multiple XML files that contains information about the applicant, such as the Company's balance sheet, Visura camerale (Chamber of Commerce report), Visura Catastale (Land Registry report), and Visura Ipocatastale (Hypothecary Registry report). The back-end searches for additional data from the database where tables in MySQL have been created to add an additional layer of complexity to the model.

Overall, this system allows for the efficient retrieval and analysis of important data from multiple sources. Once the computations are complete, the system outputs the results to the front-end interface, where the user can view and interact with the data. The front-end initiates the entire process by triggering a submit button that calls the back-end.

1.3.3. Data Processing and Result generation

The user is presented with three different input sections on the index.html page. Once the data is sent from the front-end to the back-end, the Python computations replicate similarly the Excel file's logic. The back-end sends the output result of the risk assessment to the front-end on the result.html page. and based on the information provided, the software will generate three different outputs.

Input sections and data required:

1. **Obligor Data Section:** VAT Number
2. **Bond Information Section:** Type of Bond, Bond Value, Bond Duration
3. **Co-Obligor Section:** TAX Code

Output section:

1. If only the **VAT Number** is provided, the software will display the financial statements for the last two years of account, over 10 key financial ratios , the Company name, type of business and assigned rating .

2. If both **VAT Number and the Bond information** are provided, the result page will also display the maximum credit limit the Obligor can borrow, along with six sub-limits indicating the maximum amount the applicant can borrow per each type of bond. Additionally, the result page will provide a suggested Decision Maker to sign off the quotation, if applicable, and the Final Risk Score associated with the requested quotation.
3. If **VAT Number, Bond information and TAX Code** is entered (or only TAX Code alone) the result page will display the details of the properties owned by the co-obligor, any restrictions on properties, mortgage amount, mortgage duration, mortgage start date. The page will also display an adjusted value called 'final_result' that estimates the actual amount that could be recovered from that asset, in the event of a claim and a recovery action..

1.3.4 Transmit the output to the front-end as a structured JSON object

Once the computations are complete, the program sends the results to the front-end to provide a comprehensive overview of the situation. The output is transmitted as a JSON object. This output will include information such as, key performance indicators, main financial ratios, final risk score, applicant's assigned rating, max suggested credit limit, 5 sub limits, person entitled for signing off on the guarantee, if applicable, the co-obligor credit worthiness and any mortgages detailed about the co-obligor.

1.4. Data to be analyzed

1.4.1. Italian Chamber of Commerce: The Obligor

The first section of the analysis requires financial information about the Obligor. The front-end has been designed to only ask for the VAT number to retrieve the balance-sheet information. The data being retrieved will be displayed in the front-end result.html page in two columns corresponding to the last two financial years of the account, starting from the most recent. The data should include Shareholder Equity, Total value of production, Total Payables, Payables within the following year, Liquid assets, Total Assets, Total Receivables, Receivables within the following year, Profit / Loss (-), Interest & financial expense, Purchase of goods, Inventories, EBIT and more. It will also include important liquidity ratios data: Acid test, Current ratio, Leverage (financial leverage), ROE, ROI, Interest ratio and more.

Because the XML (sample file) received by the Chamber of Commerce includes only the last two financial years, the software, for this purpose, will only analyze the last two financial statements, throughout the project.

1.4.2. Surety Bond Requirements via Front-end

This section requires the input of the data pertaining to the specific bond for which a quote is requested, with three subsections to be completed:

Bond Type: The type of bond requested, for example, Performance Bond or Bid bond. The Client can choose from the 5 main classes of Surety bonds.

Bond Value: The bond value refers to the maximum amount of money that the Surety bond will cover in case the Obligor (the party that purchased the bond) fails to fulfill their obligations as agreed in the bond's Terms and Conditions. The bond value is typically specified in the bond agreement or it can be derived. If the information is not available, the underlying contract between the Beneficiary and the Obligor, that the surety bond guarantees, needs to be analyzed to identify the amount of guarantees required.

Bond Duration: The duration refers to the length of time that a Surety bond provides coverage. The duration is expressed in months and includes the policy period within which the Beneficiary has the right to claim. It is an important factor to consider when evaluating the risk associated with a particular bond, as longer bond durations may carry a higher risk of default.

Obligation/Works Value: The backend will automatically calculate this section. It represents the total amount of obligation assumed by the Obligor towards the Beneficiary. For instance, for works awarded by a tender, for which a definitive deposit is required (Performance bond), the obligation will correspond to the 10% of the award value of the contract, or more commonly, 10 times the sum insured.

1.4.3. The Italian Land Registry : The Co-Obligor

The next step pertains to the Co-obligor section. In this section the front-end is designed to request the TAX code. Once the submit button is clicked, the

back-end is triggered to retrieve the relevant information from the Italian Land Registry file. The Excel model required manual entry of the various fields, which are now retrieved and manipulated automatically .As part of the assessment, only real estate properties are evaluated. Lands are excluded due to their nature, which requires a longer period of time to receive cash from the sale

To assess the Co-obligor the model is interested in any properties owned, percentage of ownership, any mortgage on property, cadastral value and any relevant information about any restrictions on the properties. Specifically, mortgage data such as the mortgage-start date, total mortgage value and mortgage duration that were entered manually, are now retrieved automatically to estimate the adjusted security value for each property. The backend calculation ensures a prudent assessment of the real recoverable value in the event of a claim and if a recovery action needs to be initiated. In principle, properties weighted down by mortgages are not attractive.

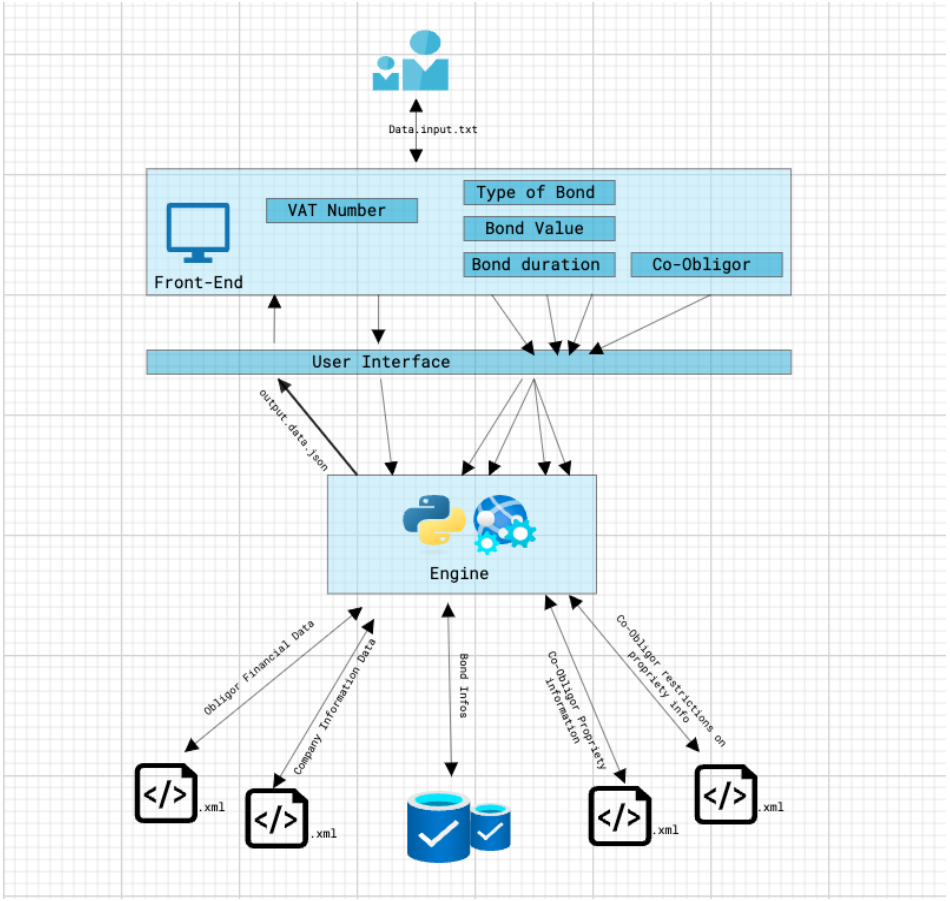
The value of the personal guarantee (co-obligor) is calculated using the following formula:

- Personal Guarantee Value = 170 x cadastral income x % of ownership of the property

Both the cadastral income and the percentage of ownership must be acquired from the cadastral certificate.

Lastly, in the excel model also the incorporation date had to be entered manually.

2. System Architecture



2.1.1. Input: the XBRL file

To allow the processing of all information, the program extracts the financial data from the XBRL file, specifically for the last two years of account. Now, data can be easily manipulated and analyzed. All the needed mathematical computations can take place.

2.1.2. The core of the application

The application is the main execution part of the system: the request in the front-end triggers a call to the back-end that retrieves the inputs from the front-end to the back-end. Once the data has been collected, the algorithms can process the data and retrieve all the Company's data from the various xml files and the database. To add an additional layer of complexity to the project, instead of creating variables in Python code, I've created a database in MySQL where to store simple Bonds information with assigned aggravating/mitigating corrective factors. This decision should result to be optimal, especially when future circumstances will need the database to be edited or modified.

The Excel-model, for its nature, is constantly evolving to refine the risk assessment based on new data, and therefore there is a clear need to create a separation of concerns between the data layer and the logical level. In this way, if only one adjustment of one coefficient is required, it will be sufficient to modify some records in the database while leaving the structure of the code intact. Viceversa, if it is necessary to correct a bug, the data and evaluation metrics, in the database, will not be touched.

3. Back-end design/development

3.1. MySQL Database

A database or "db" is a structured and homogeneous set of data. MySQL databases support various types of backend and different application-program-interfaces APIs. Users or Python Script interact with databases through the so-called query languages, which can be used for querying, inserting, deleting, updating data, etc.
Typically, databases can have different structures, but currently the most common is the relational structure, which represents the database using tables that group data based on their type and relationships between them. An important requirement for a good database is to avoid unnecessarily duplicating the information contained in it.
Through relationships, data from one table can be searched for in another table. In database management we have seen that rows of a table are called tuples or records, each of which is composed of at least one attribute (which represents the category of data, such as an email address or a characteristic), and for each table, a primary key is defined, which is a minimal set of attributes that uniquely identifies each record of the entity or association.
Example of different type of keys:

- One of the attributes: for a table that stores Clients, the "Fiscal Code" of the person will be the key attribute used.
- A combination of attributes, called a composite key: for the students of a class, name and surname can be used.
- An attribute that is added specifically for its purpose: for example, in a store, a bar-code is generated for each product.

SQL is a programming language solution that overcomes the expensive necessity of executing a loop to collect different records, an operation that depended directly on the number of data present in the table, we call it of complexity O(n). SQL is capable of performing all normal administration operations of databases (inserting, modifying, deleting data) and also allows for a simple tool to extract and read data or groups of data through a single, more efficient operation: the query.

The functionality of a database depends essentially on its design: the correct identification of the purposes of the database allows for faster data extraction and, in general, more efficient data management. In the field of engineering design of databases, three independent and consecutive levels of design are distinguished: conceptual design, logical design, and physical design.

Note: all corrective factors are represented as floating-point numbers that indicate the appropriate corrective factor needed to assess the level of risk associated with the Client's request. In the Python program, they will be transformed into actual corrective factors by multiplying the desired value times the expressed adjusting factor. For example, if the percentage taken into consideration is 90%, the corresponding attribute in the record will be filled with the value 0.90.

3.2. Python Algorithm/Program

The language chosen for writing this program is Python . Additionally, JSON, primarily used to manage the transmission of data from the backend to the frontend and database.

3.2.1. Input: file txt

As mentioned before, the program receives inputs in .txt format from the front-end, and triggers a call to the back-end.

3.2.2. Output: JSON object

The output JSON object contains all the data that has been calculated by the program and displayed on the result.htm page. The output information include:

- Company and type of business, refers to the company name and type of business matching the specified VAT number
- Company rating, generated with the help of the function AZScore(), which has been obtained by taking into account different financial ratios. The higher the AZ Score, the lower the likelihood of bankruptcy, positively weighting on the Company' rating.
- Maximum suggested credit limit, which is the maximum amount of money the Client has the potential to borrow, remaining into an acceptable medium /low risk area.
- 5 subclasses of credit limit, represents the maximum amount the Company can borrow, for each main class of bond.
- Financial indices, calculated for the last two years of account for the specified VAT number
- Main financial income statements
- The final decision-maker, which represents the level of experience required to sign off/bound the bond requested.
- The final adjusted risk score that takes into account multiple parameters from the database stored in MYSQL relating the type of business, ATECO code, company trading years, and the details of the requested bond.

```
# This function communicates between backend and frontend. Stores the text input variable from the front-end and saves it in
dictionary data
@app.route('/result', methods=['GET','POST'])
def result():
   if request.method == "GET":
       return " not valid"
   if request.method == 'POST':
   # Set the input variables to None. Retrieve input variables from index.html if not empty
       vat_no, bond_value, bond_duration, type_of_bond, tax_code = None, None, None, None, None
       # If insert-text field in index.html is not empty retrieve value and assign it to variable vat_no
       if request.form.get('insert-text') is not None and request.form.get('insert-text') != '':
           vat_no = request.form.get('insert-text')
       # If bond value field in index.html is not empty retrieve value and assign it to variable bond_value
       if request.form.get('bond_value') is not None and request.form.get('bond_value') != '':
           bond_value = request.form.get('bond_value')
       # If bnd duration field in index.html is not empty retrieve value and assign it to variable bond_duration
       if request.form.get('bond_duration') is not None and request.form.get('bond_duration') != '':
           bond_duration = request.form.get('bond_duration')
       # If type of bond field in index.html is not empty retrieve value and assign it to variable type_of_bond
       if request.form.get('type-of-bond') is not None and request.form.get('type-of-bond') != '':
           type_of_bond = request.form.get('type-of-bond')
       # If tax code field in index.html is not empty retrieve value and assign it to variable tax_code
       if request.form.get('tax_code') is not None and request.form.get('tax_code') != '':
           tax_code = request.form.get('tax_code')
       # Initialize dictionaries to store data and error messages
       data = {}
       error = {}
```

When the result() function receives POST request, it checks that the input data from index.html are not empty, specifically it checks for vat_no, bond_value, bond_duration, type_of_bond, tax_code. Throughout the execution of the function, if any None value is found, an error message is redirected to the index.html page.

The function also retrieves company name and assigned rating using the get_data_for_key() and get_assigned_rating() functions, respectively. If the bond value, bond duration, and bond type are also provided, the function calls additional functions to calculate the credit limit, sub-limits, adjusted final score, and the suggested level of seniority to approve the quotation based on the risk assessed for the quotation. The results are stored in the data dictionary. If the tax code is not None, the function retrieves the information related to the properties/buildings owned by the Co-Obligor. In addition, any information related to any potential restriction on those properties, is retrieved. To do so, the function uses multiple functions such as get_number_proprieties(), get_cadastral_income(), get_mortgage_info(), and get_percentage_ownership_buildings(). The function then calculates the value of the co-obligation offered using the calculate_restriction_value() function. The results are stored in the data dictionary. The final output tells us the amount in Euro we can recover from the co-obligor, for each property, in the event the main Obligor defaults.

4. Implementation

The first step in the implementation process is to load the libraries required for the code to run smoothly. The following is a list of the libraries used:

- **re:** Library to match tags with regular expression
- **copy:** Library for creating copies of objects
- **crypt.methods**: Library for encrypting and decrypting data
- **flask:** Web framework for handling web requests and responses.
- **pymysql:** Library to connect to MySQL using Python
- **SQLAlchemy:** Library used to perform operations on MYSQL database

```
import re

import copy

from crypt import methods

from flask import Flask, render_template, request, jsonify

import pymysql

from flask_sqlalchemy import SQLAlche
```

4.1. Facilitation Function Creation

In order to extract and manipulate data from the Client's balance sheets, Python uses regular expressions to extract the financial data tags with their corresponding values that are then stored in a Python dictionary called 'data'. The 're' module has been imported to allow the writing, finding and matching of regular expressions, that is, the management of the retrieval of financial tags and respective values from the balance sheet in XBRL format.

The python script defines two dictionaries. The tags dictionary contains the names of the financial data tags to be extracted from the xml file as keys, with their corresponding tag_values. The tags_types dictionary defines the data type for each extracted tag. (e.g. currency, float, string). The tag_value is expressed in floating point numbers representing currency, Euro. Here an example:

```
tags = {
   'patrimonio_netto': 'TotalePatrimonioNetto',
   'totale_crediti': 'TotaleCrediti',
   'crediti_esigibili_entro_esercizio': 'CreditiEsigibiliEntroEsercizioSuccessivo',
   'disponibilita_liquide': 'TotaleDisponibilitaLiquide',
   'rimanenze': 'TotaleRimanenze',
   'debiti_esigibili_entro_esercizio': 'DebitiEsigibiliEntroEsercizioSuccessivo',
   'totale_valore_produzione': 'TotaleValoreProduzione',
}
tags_types = {

  'patrimonio_netto' : 'currency',
   'totale_crediti' : 'currency',
   'crediti_esigibili_entro_esercizio' : 'currency',
   'disponibilita_liquide' : 'currency',
   'rimanenze' : 'currency',
  'valore_produzione_altri_ricavi' : 'float',
   'turnover_impegni_ratios': 'float',

}
```

The main function in the calculation.py file is get_financial_data(tags, file_path) function that lays the foundations for retrieving all necessary data. It extracts the financial data and it is stored so that it can be easily manipulated later.

```
file_path = '/Users/giuls/Desktop/SBT/download-file-api.txt'

# Function takes in file path and tags
def get_financial_data(tags, file_path):
   # Read the file
   with open(file_path, 'r') as file:
       content = file.read()
   # Create empty dictionary where all data will be stored
   data = {}
   # Loop through each tag in tags dictionary created above
   for tag_name, tag in tags.items():
       # Regular expression that finds matches for any given requested tag
       tag_matches = re.findall(fr'<(?:itcc-ci:|itccci:){tag} [^>]*contextRef="([id])
([^"]+)"[^>]*>([\d,.]+)</(?:itcc-ci:|itccci:){tag}>',content)
       # Regular expression to retrieve Company name
       if tag_name == 'company_name':
           tag_matches = re.findall(fr'<(?:itcc-ci:|itccci:){tag}[^>]*contextRef="                    ([id])
_([^"]+)">([^<]+)</ (?:itcc-ci:|itccci:){tag}>',content)
       # If there are matches, stored them separately
       if tag_matches:
           tag_values = {}
           for match in tag_matches:
```

```
                context_type, context_value, tag_value = match
                #  Replace 'd' with 'i' for all instances. Helps algorithm not to crash
                context_type = 'i' if context_type == 'd' else context_type
                   # Output float numbers with commas, for currencies
                if isinstance(tag_value, float):
                    tag_value = float(tag_value.replace(',', '.'))
                  else:
                    tag_value = tag_value
                # Store type and value in dictionary tag_value
                tag_values[f"{context_type}_{context_value}"] = tag_value
            # Store the result into data dictionary, with key the tag name
            data[tag_name] = tag_values
        else:
            print(f"Error: {tag} tag not found")
    # Return copy to avoid algorithm crashing
    return copy.deepcopy(data)
data = get_financial_data(tags, file_path)
```

The extracted tag values are stored in a nested dictionary called tag_values, with keys context_type, context_value, and tag_value. Finally, the script adds the tag_values dictionary to the data dictionary tag_name and returns the data dictionary.

Example:

data = { '**totale_attivo_circolante**': { *# tag_name = financial statement*
'**i_31-12-2005': '727919**', *# year of account followed by the matching tag_value for the given tag_name and year of account*
'**i_31-12-2004': '593811**'
}
}

The output is a dictionary data structure denoted by curly braces {} that allows storing key-value pairs. In this case, the dictionary has one key-value pair where the key is 'totale_attivo_circolante' and the value is another inner dictionary that has two key-value pairs where the keys are the year of account, which in this case they correspond to year of account 2005 and 2004 respectively 'i_31-12-2005' and 'i_31-12-2004', and the values are '727919' and '593811', respectively. The values are ordered first by parameter index and then by reference year from the most recent to the least recent. For each parameter, there are therefore two sets of data. Following is an example of a 'tag' , as it's written in the XML file :

<itcc-ci:TotaleAttivoCircolante contextRef="i_31-12-2005" unitRef="eur" decimals="6780" >727919</itcc-ci:TotaleAttivoCircolante>

- tag_name: **TotaleAttivoCircolante** represents Total Assets, and its used to find a match in the xml file ;
- context_type : **"i"** or "**d" .** represents whether the tag_name is of type 'instant'or 'duration'. Instant represents the asset section of the balance sheet and duration represents the liability, equity section of the balance sheet;
- context_value : **31-12-2005** represents the year of account for the corresponding client / tag_name;
- tag_value: **727919** is the value associated with the tag_name. It's of type currency € 727,919 ;

```
def return_result():
  result = "\n <br>".join([f"{key} ({context_ref}): {value}"
                     for key, values in data.items()
                     for context_ref, value in values.items()])
  return result
```

The return_result() function on calculation.py sheet, stores all the data retrieved from the balance sheet, and it manipulates the data here. The resturn_result() function takes no arguments and returns a string created by iterating over the dictionary data {} created in get_financial_data(tags,file_path) function, explained above. For each key-value pair in the data dictionary, the code iterates over the inner dictionary values and retrieves the context_ref and value. These two variables are then formatted into a string storing the key, context_ref, and value. The process continues until all the key-value pairs in data have been

processed.

In summary, the return_result() function creates a formatted string that represents the data in the data dictionary, with each key-value pair on a new line separated by an HTML line break tag.

Once the data has been correctly retrieved from the balance sheet, formatted, stored in the corresponding dictionary and the algorithms have performed their tasks, another helper function get_adjusted_result(bond_type,bond_value,months), has been designed to perform the adjusted risk calculations.

```
# This function communicates from backend to database to calculate the adjusted risk of a quotation based on the text input bond type, bond value, and bond duration and retrieves the corrective factors stored in the db. Function render_template() pushes the data to the result.html page in the front-end
def get_adjusted_result(bond_type,bond_value,months):
   #If not bond has been specified, return zero
   if bond_type is None:
       return 0
   else:
       # Get bond risk score and impact factor from Adjusted_risk_table from MYSQL
       bond_type_query = "SELECT bond_risk_score, impact_factor FROM Adjusted_risk_table WHERE bond_type = '{}'".format(bond_type)
       result = engine.execute(bond_type_query).fetchone()
       # If bond type not found, return error message
       if result is None:ce('"', '')
           # Check for input values, bond value, bond duration and bond type. Display error message if any values is None and redirect to index.html
           if bond_value is not None or bond_duration is not None or type_of_bond is not None:
               print('bond_value: ', bond_value)
               print('bond_duration: ', bond_duration)
               print('type_of_bond: ', type_of_bond)
               if bond_value is None or bond_value == '':
           return jsonify({'error': 'Not Valid Bond Type'})
       # Get the Ateco code and impact factor from ateco_code_table from MYSQL
       bond_risk_score, impact_factor = result
       ateco_code_query = "SELECT code, impact_factor FROM ateco_code_table WHERE code = '{}'".format(get_ateco_code())
       print('ateco_code_query: ', ateco_code_query)
       result_ateco = engine.execute(ateco_code_query).fetchone()
       print('result_ateco: ', result_ateco)
       # If the Ateco code is not found, return error message
       if result_ateco is None:
           return jsonify({'error': 'Not Valid Ateco Code'})
       ateco_code, ateco_impact_factor = result_ateco
       # Get the adjusting factor from capacity_track_record_table based on years of experience of the Client
       incorporation_query = "SELECT factor FROM capacity_track_record_table WHERE years_experience <= {} ORDER BY years_experience DESC LIMIT 1 ".format(get_incorporation())
       result_track_record = engine.execute(incorporation_query).fetchone()
       # If incorporation is not found, return an error message. Past performance not found.
       if result_track_record is None:
           return jsonify({'error': 'Not Valid Incorporation'})
       factor = result_track_record[0]
       # Get bond value and corrective factor from bond_value_table based on the selected bond value, from MYSQL.
       bond_value_query = "SELECT bond_value, factor FROM bond_value_table WHERE bond_value <= '{}' ORDER BY bond_value DESC LIMIT 1".format(bond_value)
       result = engine.execute(bond_value_query).fetchone()
       print('bond_value_query: ', bond_value_query)
       print('result: ', result)
       # If bond value is not found, return error message
       if result is None:
           return jsonify({'error': 'Not Valid Bond Value'})
       bond_value, bond_value_factor = result
       # Get loading factor from bond_duration_table based on bond duration, from MYSQL
       bond_duration_query = "SELECT months, loading FROM bond_duration_table WHERE months <= {} ORDER BY months DESC LIMIT
```

```
1".format(months)
        result = engine.execute(bond_duration_query).fetchone()

        print('bond_duration_query: ', bond_duration_query)
        print('result: ', result)
        # If bond duration is not found then return error message
        if result is None:
            return jsonify({'error': 'Not Valid Duration'})
        months, loading = result
        # Calculate the adjusted result based on the corrective factors retrieved from MYSQL
        adjusted_result = bond_risk_score * impact_factor
        adjusted_result = adjusted_result * ateco_impact_factor
        adjusted_result = adjusted_result * factor
        adjusted_result = adjusted_result * bond_value_factor
        adjusted_result = adjusted_result * loading
        return adjusted_result
```

The above function provides a simplified way to calculate the adjusted risk of a quotation based on the three parameters: bond_type, bond_value, and months. The 'bond_type' parameter specifies the type of bond, which is used to look up the bond's adjusted_risk_table in the MySQL database.

The Adjusted_risk_table : assigns a bond risk score and an impact factor for each type of bond selectable from the user interface homage of the website. The bond risk score values range from 10 to 50 where 10 represents a very low risk and 50 represents a high risk. The impact factor column, represented by floating point numbers, has values ranging from 0.8 to 1.15, where 0.8 is a multiplicative factor that reduces the overall risk by multiplying the risk calculated by 0.8 .

The code queries the database to retrieve the bond risk score and impact factor. If the bond type is not found in the database, the function returns 0. The 'bond_value' parameter specifies the value of the bond, which is used to look up the bond value factor in the database.

Bond_value_table : Similarly to the table seen above, the bond value table stores the bond values ranging from 1 to over 10 million, with their corresponding aggravating / mitigating factor ranging from 0.9 to 1.6.

The 'months' parameter specifies the duration of the bond expressed in months, which is used to look up the loading factor in the database in MySQL. Next, the function queries the database to retrieve the Ateco code and impact factor. If the Ateco code is not found in the database, the function returns an error message.

Ateco_code_table : Ateco is the Italian version of the NACE classification system used to classify economic activity. The table stores the Ateco codes (in real life we have almost 1000 different codes) , together with the Ateco description, type of business and the impact factor. In this case we will see that the impact factor is set to 1 for all the records, in other words it will not change the previous risk score, based on the Ateco code.

The function queries the database to retrieve the adjusting factor based on the client's years of experience. If the incorporation year is not found, the function returns an error message.

Capacity_track_record_table : stores the years of experience from 0 to greater than 10 with their corresponding aggravating / mitigating factor ranging from 0.85 to 2.

Next, the function queries the database to retrieve the 'bond value factor' based on the value of the bond selected. If the bond value is not found in the database, the function returns an error message.

Bond_duration_table : stores the duration of the bond in months and the associated loading factor. For bonds whose duration is less or equal to 2 years, the loading factor is equal to 1. On the other hand, for duration equal or greater than 50 months, the loading factor is equal to 1.15, which also represents the maximum value for the column.

Higher the bond value, higher the risk associated with the transaction. Finally, the function queries the database to retrieve the loading factor based on the bond duration. If the bond duration is not found in the database, the function returns an error message. Similarly as above, the higher the duration of the bond, higher the risk associated with the transaction, as the insurance Company remains exposed for longer, the risk increases.

Once all necessary factors have been retrieved from the database, the function calculates the resulting 'adjusted_result' .

As we have seen above, the data extrapolated from the balance-sheet is divided into two dictionaries, tag_name and tags_types.. The below function formats the data according to the data_type.

```
def format_result(data):
   for key, value in data.items():
       if isinstance(value, dict):
           for context_ref, context_value in value.items():
               # get type for key
               format_type = tags_types[key]
               if format_type == "currency":
                   value[context_ref] = currency_format(float(context_value))
               if format_type == "float":
                   value[context_ref] = f"{float(context_value):.2f}"
               if format_type == "str":
                   value[context_ref] = context_value.replace('"', "")
       else:
           data[key] = f"{value:.2f}"
```

4.3. Connecting to the database

In order to access the information stored in the database, the connection is established using the pymysql library, introduced earlier.

```
app = Flask(__name__)

 db_config = {

   'user': 'root',
   'password' : '1234asd!',
   'host': 'localhost',
   'port': 3306,
   'database': 'sbt'
 }

 # Connection string for the database
 app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://%(user)s:%(password)s@%(host)s:%(port)s/%(database)s' %
 db_config
 db_string = 'mysql+pymysql://%(user)s:%(password)s@%(host)s:%(port)s/%(database)s' % db_config
 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
 # Initialization
 db = SQLAlchemy(app)

 # Create connections. Length 1h
 engine = db.create_engine(db_string, engine_opts={'pool_recycle': 3600})
```

The Flask application is named "app" and creates a dictionary named "db_config" that contains the configuration information for connecting to a MySQL database. Specifically, the dictionary specifies the username, password, host, port number, and database name for the MySQL database.

To perform common database operations we use Flask-SQLAlchemy SQLALCHEMY_DATABASE_URI . SQLALCHEMY_TRACK_MODIFICATIONS is set to False, so that it does not track modifications and will not generate signals for every change. The pool_recycle option is set to 3600, which means that database connection is recycled every hour. This means that any connections that have been established more than 1 hour before, will be closed and replaced with a new connection.

4.4. Retrieving data from the database

To retrieve data from the database, we query the database simply using Python code. The function has error messages to test the validity of the data returned (the same checks have been done throughout the software) If the bond type is not found, the script returns an error message. The functionality is to validate the user inputs and retrieve the relevant data from the database to calculate the adjusted risk of the type of bond requested.

```
# Get bond risk score and impact factor from Adjusted_risk_table from MYSQL
       bond_type_query = "SELECT bond_risk_score, impact_factor FROM Adjusted_risk_table WHERE bond_type =
'{}'".format(bond_type)
       result = engine.execute(bond_type_query).fetchone()
       # If bond type not found, return error message
       if result is None:
           return jsonify({'error': 'Not Valid Bond Type'})
       # Get the Ateco code and impact factor from ateco_code_table from MYSQL
       bond_risk_score, impact_factor = result
       ateco_code_query = "SELECT code, impact_factor FROM ateco_code_table WHERE code = '{}'".format(get_ateco_code())
       print('ateco_code_query: ', ateco_code_query)
       result_ateco = engine.execute(ateco_code_query).fetchone()
       print('result_ateco: ', result_ateco)
       # If the Ateco code is not found, return error message
       if result_ateco is None:
           return jsonify({'error': 'Not Valid Ateco Code'})
        ateco_code, ateco_impact_factor = result_ateco
       # Get the adjusting factor from capacity_track_record_table based on years of experience of the Client from MYSQL
       incorporation_query = "SELECT factor FROM capacity_track_record_table WHERE years_experience <= {} ORDER BY
years_experience DESC LIMIT 1 ".format(get_incorporation())
       result_track_record = engine.execute(incorporation_query).fetchone()
       # If incorporation is not found, return error message. Past performance not found.
       if result_track_record is None:
           return jsonify({'error': 'Not Valid Incorporation'})
       factor = result_track_record[0]
       # Get bond value and corrective factor from bond_value_table based on the selected bond value, from MYSQL.
       bond_value_query = "SELECT bond_value, factor FROM bond_value_table WHERE bond_value <= '{}' ORDER BY bond_value DESC
LIMIT 1".format(bond_value)
       result = engine.execute(bond_value_query).fetchone()
       print('bond_value_query: ', bond_value_query)
       print('result: ', result)
       # If bond value is not found, return error message
       if result is None:
           return jsonify({'error': 'Not Valid Bond Value'})
        bond_value, bond_value_factor = result
       # Get loading factor from bond_duration_table based on bond duration, from MYSQL
       bond_duration_query = "SELECT months, loading FROM bond_duration_table WHERE months <= {} ORDER BY months DESC LIMIT
1".format(months)
       result = engine.execute(bond_duration_query).fetchone()
```

4.6. Final decision and criteria used

The get_assigned_rating() function is designed to assign a rating for any Company based on its financial data, only by inserting the VAT_number. The function first checks if the "AZ_manufacturing" value exists in the data{} dictionary. The AZ_Manufacturing() function is an adjusted calculation of the AZ-Score, which is a financial metric developed by Cerved, an Italian Credit Information Company, to predict the likelihood of a company's bankruptcy within two years. The get_assigned_rating() function performs a series of if-elif statements to determine the appropriate assigned rating based on the AZ_score value for the last year of account.If the AZ_score_M_1 falls within a certain range, the assigned_rating variable is set to a tuple consisting of a string rating, e.g. 'CCC', and an integer value, e.g. 20. The value assigned corresponds to the risk score associated with that rating. If the value of AZ_score_M_1 is greater than 8.5, the assigned_rating is

set to a tuple consisting of 'AAA', and 100. If an error occurs, the function will return an error message.

```
def get_assigned_rating():
   assigned_rating = ""
   print(data, "data")
   if 'AZ_manufacturing' not in data:
       return "Error: AZ_manufacturing value not found"
   AZ_score_M_1 = round(list(AZ_manufacturing.values())[1],2)
    print(AZ_score_M_1,"AZ_score_M_1")
    if AZ_score_M_1 <= 1.75:
       assigned_rating = ('D', 0)
   elif AZ_score_M_1 <= 2.0:
       assigned_rating = ('C', 1)
   elif AZ_score_M_1 <= 2.2:
       assigned_rating = ('CC', 10)
   elif AZ_score_M_1 <= 2.5:
       assigned_rating = ('CCC-', 15)
   elif AZ_score_M_1 <= 3.2:
       assigned_rating = ('CCC', 20)
   elif AZ_score_M_1 <= 3.75:
       assigned_rating = ('CCC+', 20)
   elif AZ_score_M_1 <= 4.15:
       assigned_rating = ('B-', 25)
   elif AZ_score_M_1 <= 4.4:
       assigned_rating = ('B', 30)
   elif AZ_score_M_1 <= 4.75:
       assigned_rating = ('B+', 35)
   elif AZ_score_M_1 <= 4.95:
       assigned_rating = ('BB-', 40)
   elif AZ_score_M_1 <= 5.25:
       assigned_rating = ('BB', 45)
    elif AZ_score_M_1 <= 5.65:
       assigned_rating = ('BB+', 50)
   elif AZ_score_M_1 <= 5.83:
       assigned_rating = ('BBB-', 60)
   elif AZ_score_M_1 <= 6.25:
       assigned_rating = ('BBB', 60)
   elif AZ_score_M_1 <= 6.4:
       assigned_rating = ('BBB+', 65)
   elif AZ_score_M_1 <= 6.65:
       assigned_rating = ('A-', 70)
   elif AZ_score_M_1 <= 6.85:
       assigned_rating = ('A', 70)
   elif AZ_score_M_1 <= 7.2:
       assigned_rating = ('A+', 75)
   elif AZ_score_M_1 <= 7.6:
       assigned_rating = ('AA-', 80)
   elif AZ_score_M_1 <= 8.0:
       assigned_rating = ('AA', 85)
   elif AZ_score_M_1 <= 8.5:
       assigned_rating = ('AA+', 90)
   else:
       assigned_rating = ('AAA', 100)
       print(assigned_rating, "here inside")
   return assigned_rating
```

The get_experience_factor() function calculates the experience factor of the applicant combining the bond value requested and the trading years/experience of the applicant. The function first checks whether the parameters and variables exist in the data dictionary. If not, it returns an error message. The function first sets the works_value as ten times the bond_value (public tenders) . After that, it cross-checks the trading years, values of past performance and it assigns an experience factor accordingly and it returns the calculated experience factor. For example, the first if statement that checks the bond value, incorporation years and total value of production first checks if incorporation_year is greater than 10 (years) then it checks if total value of production 'valore_prod_1' for the past year of account and the value of production 'valore_prod_2 ' for the second last year of account are both greater than twice the works_value. If these conditions are met, then it sets the experience_factor to 85. In other words, if the company has been incorporated for more than 10 years and its total production value is more than twice the works value for two consecutive years, it is considered to have a high level of experience in the industry and will be assigned an experience factor of 85, which multiply the final risk score by 0.85. For example, if 10 means high risk and 0 means no risk, a Company applying for a Surety bond and the bond requested is high in value (high sum insured) but the applicant has demonstrated significant past performance for similar projects in the past, the overall risk will go down to 8.5 from 10. This is the role of the adjusting factors.

```
def get_experience_factor(bond_value, incorporation_year):
  if bond_value <= 0 or incorporation_year <= 0:
       return "bond_value and incorporation_year must be greater than zero"

   if 'valore_produzione_altri_ricavi' not in data:
       return "valore_produzione_altri_ricavi value not found"

   experience_factor = 0
   works_value = 10 * bond_value
   print( works_value , "works_value")
   if 'valore_produzione_altri_ricavi' in data :
       valore_prod_1 = float(list(data['valore_produzione_altri_ricavi'].values())[0])
       valore_prod_2 = float(list(data['valore_produzione_altri_ricavi'].values())[1])

  if incorporation_year>10 and valore_prod_1 > works_value * 2 and valore_prod_2> works_value * 2 :
           experience_factor = 85
   elif  5 <incorporation_year <= 10 and valore_prod_1 >= works_value * 2  :
           experience_factor = 90
   elif incorporation_year <= 5 and valore_prod_1 >= works_value:
           experience_factor = 95
  elif incorporation_year>5 and valore_prod_1> works_value*0.5 and valore_prod_2>works_value * 0.5:
           experience_factor = 100
   elif incorporation_year > 1 and valore_prod_1 > works_value:
           experience_factor = 120
   elif incorporation_year <= 1:
       experience_factor = 200
   else:
       # no condition satisfied
       return None

   return experience_factor
```

The function get_credit_limit() takes three arguments: experience_factor, assigned_rating, and data. It calculates the maximum credit limit that can be granted to the applicant based on the provided parameters. First, it calculates the acid test value by calling the calculate_acid_test() function and then it retrieves the values for total_value_of_production and shareholders_equity from the data dictionary. After that, the function checks the assigned experience_factor that has been introduced just before and based on the assigned value, it calculates the credit limit.

If the experience_factor is equal to 85, which indicates an expert company, it checks if the shareholders equity is greater than 20% of total_value_of_production for the last two years, if the acid_test is greater than 1, and if the assigned_rating is less than or equal to 'CCC'. If all of these conditions are met, then it calculates the credit limit as total_value_of_production * 0.20, otherwise, it calculates it as 10% of the total_value_of_production of the last year of account. The same process is repeated in a series of if/elif/else conditional statements that checks the criteria for True condition.

```
def get_credit_limit(experience_factor, assigned_rating, data=main_data):
   acid_test = calculate_acid_test(data)
   acid_test = float(list(acid_test.values())[0])
   print(acid_test, "acid_test")
   if 'totale_valore_produzione' in data and 'patrimonio_netto' in data :
       valore_produzione_1 = float(list(data['totale_valore_produzione'].values())[0])       #1900
       valore_produzione_2 = float(list(data['totale_valore_produzione'].values())[1])
       patrimonio_netto_1 = float(list(data['patrimonio_netto'].values())[0])       #80
       patrimonio_netto_2 = float(list(data['patrimonio_netto'].values())[1])
   print(experience_factor, "experience_factor")
   print(assigned_rating, "assigned_rating")
   if experience_factor == None:
       return 0
   if experience_factor == 85:                      # 'expert'
        if patrimonio_netto_1 > valore_produzione_1 * 0.20  and patrimonio_netto_2 > valore_produzione_2 * 0.20 and acid_test > 1
and assigned_rating <= 'CCC':
           credit_limit = valore_produzione_1 * 0.20
       else:
           credit_limit = valore_produzione_1 * 0.10
   elif experience_factor == 90:                    # 'experienced'
       if valore_produzione_1 * 0.10 < patrimonio_netto_1 and acid_test > 1 and assigned_rating  <= 'CCC':
           credit_limit = valore_produzione_1 * 0.10
       else:
           credit_limit = valore_produzione_1 * 0.05
   elif experience_factor == 95:                   # 'demonstrated':
       if valore_produzione_1 * 0.05 < patrimonio_netto_1 and acid_test > 1 and assigned_rating  <= 'CCC':
           credit_limit = valore_produzione_1 * 0.05
       else:
           credit_limit = valore_produzione_1 * 0.025
   elif experience_factor > 95:                     # 'on the market'
       if valore_produzione_1 * 0.025 < patrimonio_netto_1 and acid_test > 1 and assigned_rating  <= 'CCC':
           credit_limit = valore_produzione_1 * 0.025
       else:
           return 0
   return credit_limit
```

The function get_sublimits(), takes get_credit_limit() function output, as input parameter, and returns a dictionary of sublimits, each with a value and a label. The dictionary includes 6 sublimits for the 6 main categories of type of bonds, such as:

- Bid bond
- Advance payment bond
- Subsidy
- General Contractor
- Housing &
- Other

The get_sub_limits() function, calculates the value of each sublimit based on a pre-set percentage of the credit limit output, generated above. For example, if the maximum credit_limit assigned to an applicant is 1.000.000,00 the bid bond sublimit is 0.25 times the credit_limit, thus 250.000,00 Euro. The function uses the currency_format() function to format the output values as currency. The output is a string representation with a comma separator at two decimal places. The function returns the dictionary of the sublimits with the values and labels for each sublimit.

```
def get_sublimits(credit_limit):
   sublimits = {
       "bid_bond": {
           "value": None,
           "label": "Bid Bond"
       },
```

```
        "advance_payment_bond": {
            "value": None,
            "label": "Advance Payment Bond"
         },
        "subsidy": {
            "value": None,
            "label": "Subsidy"
        },
        "general_contractor": {
            "value": None,
            "label": "General Contractor"
        },
          "housing": {
            "value": None,
            "label": "Housing"
        },
        "other": {
             "value": None,
            "label": "Other"
        }
}

  sublimits["bid_bond"]["value"] = currency_format(0.25 * credit_limit)
  sublimits["advance_payment_bond"]["value"] = currency_format(0.5 * credit_limit)
  sublimits["subsidy"]["value"] = currency_format(0.8 * credit_limit)
  sublimits["general_contractor"]["value"] = currency_format(credit_limit)
  sublimits["housing"]["value"] = currency_format(0.9 * credit_limit)
  sublimits["other"]["value"] = currency_format(0.6 * credit_limit)


  return sublimits
```

The get_adjusted_result(bond_type, bond_value, months) function seen earlier retrieves data from the database and it returns the 'adjusted result' based on the corrective aggravating / mitigating factors.

The get_underwriter(adjusted_risk_score) function takes in as a parameter the adjusted risk score based on the corrective factors stored in the database and returns the underwriting approach based on a set of rules. The function first creates a dictionary underwriting_approach {} where each key represents a range of adjusted risk scores and the corresponding underwriting approach for that range of risk. For example, if the adjusted risk score is between 0 and 20 inclusive,, the underwriting approach will be 'JUNIOR UNDERWRITER'. If the score is between 21 and 30 inclusive, the approach will be 'UNDERWRITER', and so on. *The function uses the adjusted risk score to determine which underwriting approach is most appropriate.*

This function helps determine the appropriate level of experience for an underwriter based on the level of risk associated with the requested quotation. In general, as the risk level increases, the underwriting approach shifts from junior underwriter to board meeting to do not write, with more experienced underwriters needed to handle higher-risk quotations.

```
def get_underwriter(adjusted_risk_score):
      underwriting_approach = {
          0 <= adjusted_risk_score <= 20: 'JUNIOR UNDERWRITER',
          21 <= adjusted_risk_score <= 30: 'UNDERWRITER',
          31 <= adjusted_risk_score <= 40: 'UNDERWRITER',
          41 <= adjusted_risk_score <= 50: 'SENIOR UNDERWRITER',
          51 <= adjusted_risk_score <= 60: 'DAILY BRIEFING',
          61 <= adjusted_risk_score <= 70: 'BOARD MEETING',
          71 <= adjusted_risk_score <= 100: 'DO NOT WRITE !'
      }.get(True, 'Invalid score')
      return underwriting_approach
```

5. Conclusions

This project has shown that it is possible to offer an easy-to-use system that can assess an applicant's risk, the bond requested, and the creditworthiness of a co-obligor with just a few inputs. The project demonstrated the potential of using Python, JSON, MySQL, HTML, and CSS to develop a user-friendly software that allows for efficient communication between the front-end, back-end, and database. The software automates the risk assessment for Italian companies applying for a Surety bond in the Italian market, by significantly reducing errors, improving the efficiency of the underwriting process. S.B.T.N can help Insurance Companies improve their loss ratios and reduce operational costs.

The system is a simple solution for clients because it reduces the amount of information required when applying for a quotation, and it provides underwriters with a powerful tool to make informed decisions quickly and efficiently. S.B.T. offers a more objective and consistent approach to risk assessment and underwriting review, which can result in significant improvements in efficiency and profitability for the industry.

This program also avoids the criticalities of Excel spreadsheets, which consist of the possibility of deleting the formulas present in the cells or accidentally modifying them: the only thing that the user using the application needs to do is enter a handful of data from the front-end and wait for processing, without worrying about compromising the evaluation mechanism. This software, within the intended application context, yields results and helps solve day-to-day business problems and optimize time.cThis thesis has shown how working together with different technologies can lead to more effective and innovative solutions. Despite the size and complexity of the project, the overall experience was very constructive. I learned a lot and through hands-on practice, I was able to deepen my understanding of topics covered more lightly in the courses.

The result of this project is just a framework for developing a fully automated software. The future software incorporates API calls that retrieve data from all major Italian servers, to establish a marketplace where applicants can request a bond, and it will be directly visible to insurance companies via the SBT platform. SBT will facilitate the process by making available quotations on the marketplace, providing pre-assessments, and showing the risk level assigned to each quotation made available for the insurers on the ‘marketplace’. Future developments include creating a re-insurance section, where re-insurers can offer facultative coverage to insurance companies willing to take on a good risk, but that exceeds their capacity. The software will also offer a co-insurance section where multiple insurance companies can pool resources to insure a single risk. These are just a few of the many features that I see for the future SBT platform.

To test the software for errors I have inserted error messages throughout the code. I’ve done so by identifying the possible scenarios where errors may occur in the software, such as incorrect user inputs, missing data, invalid configurations or division by zero. Formulas have been checked verifying that the actual results match the expected results, or that appropriate error messages are displayed when errors occur. When an error is detected, the error message ensures a clear understanding of where the error generated. By following this approach, I’ve identified and fixed errors in the software

To conclude, what I found most stimulating throughout this journey, in which difficulties were not lacking, was that S.B.T. is a real project, and it was the main motivation that inspired me to pursue a Bachelor's degree in Computer Science.