NOVA IMS Information Management School

MDSAA

Master's degree Program in
**Data Science and Advanced Analytics**

# CLAIMS PROCESSING AUTOMATION

Modernization of an insurance company internal process

Nicola Andreatta

Internship Report

presented as partial requirement for obtaining the Master's Degree Program in Data Science and Advanced Analytics

**NOVA Information Management School**
**Instituto Superior de Estatística e Gestão de Informação**

Universidade Nova de Lisboa

**NOVA Information Management School**
**Instituto Superior de Estatística e Gestão de Informação**
Universidade Nova de Lisboa

# CLAIMS PROCESSING AUTOMATION

by

Nicola Andreatta

Internship report presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a Specialization in Data Science

**Supervisor:** Mauro Castelli

July 2023

## STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

## ABSTRACT

Deep learning and text mining are involved in the research. This work includes the project I developed together with my colleagues at SAS Institute during my internship experience. In this project we had to support an Insurance company for the automation of their existing claim processing system. In fact, as of today, the procedure of reading the incoming claim requests, selecting the useful information and extracting it to a data management software, is done manually for hundreds of claims every day. The job required by the insurance company is to substitute the existing procedure with an automated one, by implementing an OCR system to read the raw data contained in the documents sent by the customers and transform it into clean and useful information to be inserted into the data management software.

This research will show the investigation on how to deal with this problem and the objective is to automate the classification of the documents for the company, to provide them a system to prioritize the most urgent documents and to execute some technical and administrative checks on the extracted information. The automation is shown to be feasible; the completeness and accuracy of the information extracted are solid, proving that this specific task in the insurance company sector can be realized and help to reduce costs while improving time performance.

## KEYWORDS

Claim processing; Automation; Optical Character Recognition; SAS; Text interpretation

# INDEX

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| **OCR** | Optical Character Recognition |
| **CAPTCHA** | Completely Automated Public Turing test to tell Computers and Humans Apart |
| **VTA** | Visual Text Analytics |
| **LITI** | Language Interpretation and Text Interpretation |
| **FIFO** | First in first out |
| **KPI** | Key Performance Indicator |

# 1. INTRODUCTION

With the growth of Data Science techniques, the automation and enhancement of companies' internal processes have gained more relevance as a solution to reduce costs and increase efficiency. The evaluation of the process performance and the implementation of digitalized solutions are often made by other companies specialized in data analytics and information technology consulting, such as SAS Institute. I personally had the opportunity to join the Customer Intelligence and Analytics team of this company with an internship program. In this report, I will present the project in which I took part during my experience at this company.

## 1.1. COMPANY OVERVIEW

SAS Institute is a privately held American software company that specializes in analytics software and services. The company was founded in 1976 by Jim Goodnight and John Sall, who were both graduate students at North Carolina State University at the time. Originally called Statistical Analysis System, the company was renamed to SAS Institute in 1984. Today, SAS corporate headquarters is in Cary, North Carolina, and has offices in over 60 countries worldwide with more than 14,000 employees. The software is installed at more than 80,000 business, government, university sites, and 90 of the top 100 companies on the 2016 Fortune Global 500 are SAS customers.

SAS provides a wide range of analytics software and services to help businesses and organizations make data-driven decisions. Their products include data management and preparation tools, advanced analytics software, artificial intelligence and machine learning capabilities. SAS also offers consulting and training services to help customers get the most out of their analytics investments. Some of the industries that SAS serves include finance, healthcare, government, and retail. SAS's solutions are used by thousands of organizations worldwide, including many of the world's largest companies. The SAS solutions allow efficient cost in both implementation and maintenance, with the possibility to evolve gradually to handle the evolution in the requirements of business and new regulations.

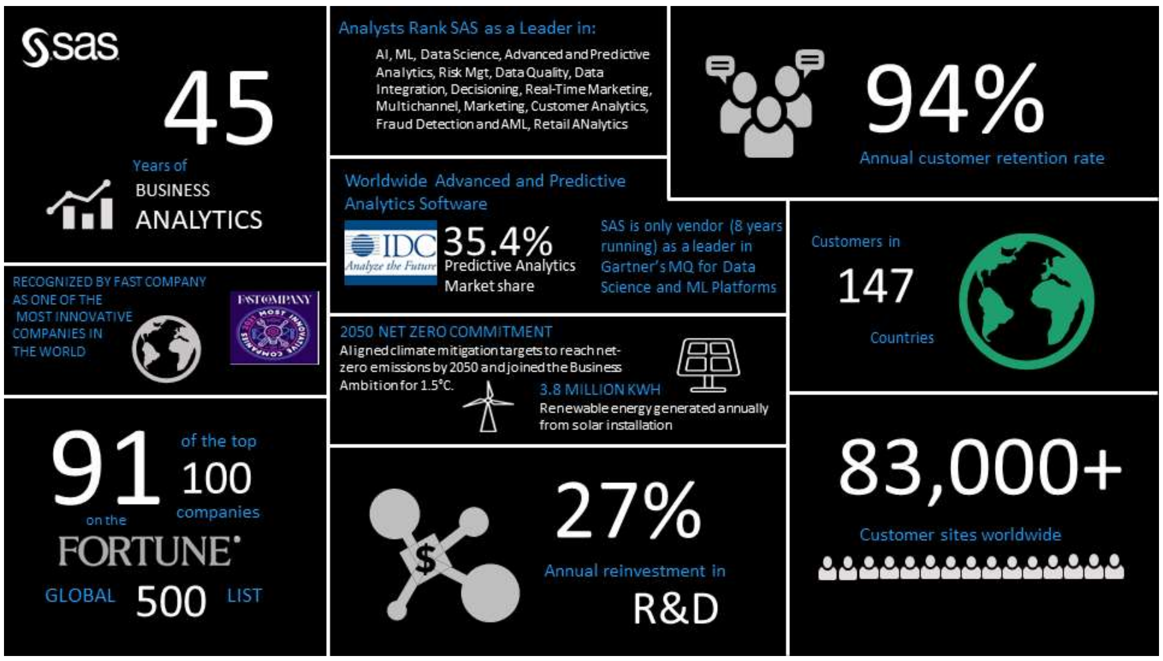

sas
45
Years of BUSINESS ANALYTICS
Analysts Rank SAS as a Leader in:
AI, ML, Data Science, Advanced and Predictive Analytics, Risk Mgt, Data Quality, Data Integration, Decisioning, Real-Time Marketing, Multichannel, Marketing, Customer Analytics, Fraud Detection and AML, Retail ANalytics
94%
Annual customer retention rate
Worldwide Advanced and Predictive Analytics Software
IDC Analyze the Future
35.4%
Predictive Analytics Market share
SAS is only vendor (8 years running) as a leader in Gartner's MQ for Data Science and ML Platforms
Customers in
147
Countries
RECOGNIZED BY FAST COMPANY AS ONE OF THE MOST INNOVATIVE COMPANIES IN THE WORLD
FAST COMPANY
2050 NET ZERO COMMITMENT
Aligned climate mitigation targets to reach net-zero emissions by 2050 and joined the Business Ambition for 1.5°C.
3.8 MILLION KWH
Renewable energy generated annually from solar installation
91 of the top 100 companies on the FORTUNE GLOBAL 500 LIST
27%
Annual reinvestment in R&D
83,000+
Customer sites worldwide

Figure 1.1 - SAS Company Infographic

Sas Institute aims to help organizations around the world turn data into intelligence. The mission of the company is to provide innovative solutions to its customers' most challenging business problems and to guarantee services that enable them to make better decisions faster.
Since 1976, commitment to employees, communities, education and the environment has been rooted in its DNA. It drives the innovation provided to the customers.

## 1.2. COMPANY SERVICES

SAS offers a range of services to help organizations implement, manage, and optimize their analytics programs. These services include:

- Consulting Services: SAS offers consulting services to help organizations define their analytics strategy, identify data sources, and develop customized analytics solutions. This service also includes the implementation of data management systems, data cleaning and data integration.
- Training: SAS offers training services for individuals and organizations looking to optimize their analytical skills. The company provides instructor-led courses online, onsite, or virtually focusing on programming, data management, and analytics solutions.
- Managed Services: SAS offers a wide range of managed services to help companies optimize their analytics program and scale up their analytic capabilities. It includes data management services, cloud services, and analytical modeling.
- Technical support: SAS provides technical support services to its clients to assist them in troubleshooting problems with software, issues with data management, and best practices for using SAS software.
- Innovation Lab: SAS has an innovation lab where data scientists, customer service, and product developers work together to develop new cutting-edge analytics solutions.

## 1.3. PROJECT OVERVIEW

The focus of this project will be the solution that SAS implemented for an insurance company to automate one of its internal processes.

The insurance company receives hundreds of documents and communications related to claims that occurred and for which the customers that took out an insurance policy with them are asking for compensation or legal assistance. The increase in the incoming number of claims and mandatory bureaucratic steps requires an increasing amount of manual work in inspecting documents, collecting the important part of information and evaluating if the request of the customer fulfils the policy contract. With the existing technology, this procedure can be automated with an OCR system and reduce the efforts to the key aspects of claim management.

The mission of the working group I took part in is to implement such a system to automate the process of management and archiving of communications regarding claims, retrieving all the relevant information."

# 2. LITERATURE REVIEW

Nowadays, OCR systems are widely used in various industries and applications. The main use of this OCR technology is for the digitization of documents, in fact, it is designed to recognize and extract text from images or scanned documents, allowing computers to interpret and process the textual information. This is particularly useful for archiving, data entry, and document management systems. In the case of the insurance sector, many companies are implementing this technology firstly to enhance operational efficiency and secondly to improve accuracy in document processing. Regarding the latter, some language and text interpretation models are applied to the text extracted from the documents to classify the information.

## 2.1. OCR SYSTEMS

Optical Character Recognition (OCR) is a technology that allows printed or handwritten text to be digitized and converted into a machine-readable format. These systems employ sophisticated algorithms and image processing techniques to extract text information from document images, enabling efficient digitization, text analysis, and information retrieval.

OCR systems have evolved significantly since their inception; with the rise of digital documents and the advent of mobile devices, they have become even more prevalent in the new century. The challenge of emulating humans in recognizing written text was already studied by Rice, Nagy and Nartker (1999) by illustrating and explaining actual OCR errors. The pictures and analysis of their research provided insights into the strengths and weaknesses of OCR systems of that time, outlining a road map to future progress. This work inspired many researchers, for instance in the identification of documents containing handwritten text (Hiremath et al., 2010) which requires more advanced techniques in recognizing characters due to the absence of a known pattern beforehand likewise computer-printed characters. Another direction of the evolution of the OCR systems involves relating word recognition to the context of the document; for instance, instead of operating on single words or characters, exploit the context through models such nearest neighbor algorithm to improve the accuracy of text recognition (Sankar et. al., 2010).

The availability of huge datasets in different languages allowed this technology to grow horizontally, including many alphabets and signs. The real-life data is fuzzy though and there is always a degree of impreciseness present in it. The book of Chaudhuri, Mandaviya, Ghoshand Badelia (2017) presents the common OCR techniques of preprocessing, character segmentation, feature extraction and classification; later, they compare them with some soft-computing techniques and assess their performance for multiple languages, showing how this approach outperforms the common method.

Although the reading capabilities gap between humans and machines will eventually shrink with the evolution of OCR systems, there is still a fascinating topic of research which is the area of CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). This widely employed test is to some degree a valuable tool in identifying the characteristics that OCR systems struggle to comprehend. (Rusu et al., 2004; 2009).

### 2.1.1. Tesseract

In this project we used Google's OCR system, Tesseract. It is an open-source OCR engine based on the Tesseract OCR engine, which uses deep learning-based methods to achieve high accuracy rates on a wide range of OCR tasks. Tesseract is designed to be easy to use and integrate into other software applications. It supports multiple languages and can handle a wide range of image formats, including TIFF, JPEG, and PNG. Tesseract also provides options for controlling the OCR engine, such as adjusting the page segmentation mode or specifying the OCR engine mode.

One of the key advantages of Tesseract is its open-source nature. This means that the code is freely available for anyone to use and modify. This has led to a large and active community of developers who have contributed to the improvement of Tesseract. For this reason, we chose it among others.

The architecture of Tesseract assumes that the input is a binary image. Therefore, the image has to be preprocessed to benefit the most from it. In figure 2.1 below, we show some basic preprocessing operations that are grayscale, binarization and deskew. The first one means turning the RGB channel to a single grayscale channel of pixel values from 0 to 255. The binarization is then setting a threshold in this scale to differentiate values into two binary options, namely black and white. Finally, the last step of deskewing is done to correct a possibly inclined image (Ma et al., 2021).

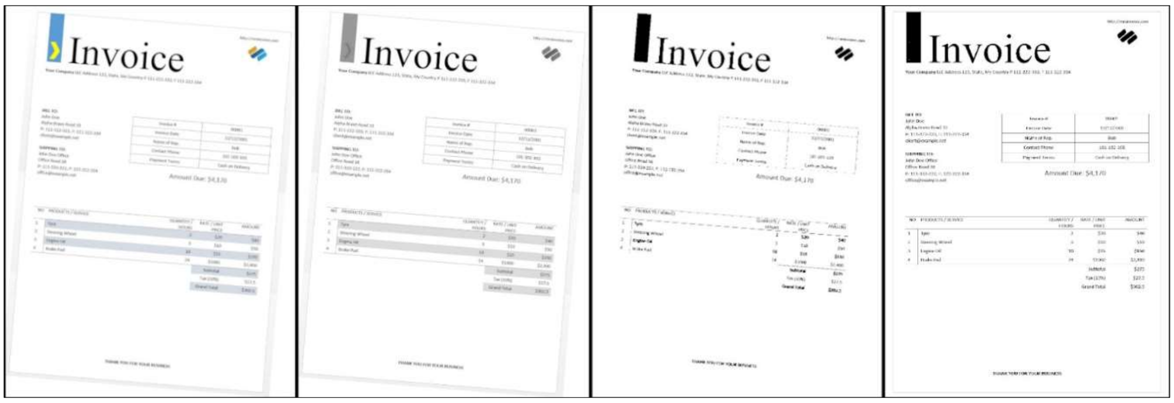
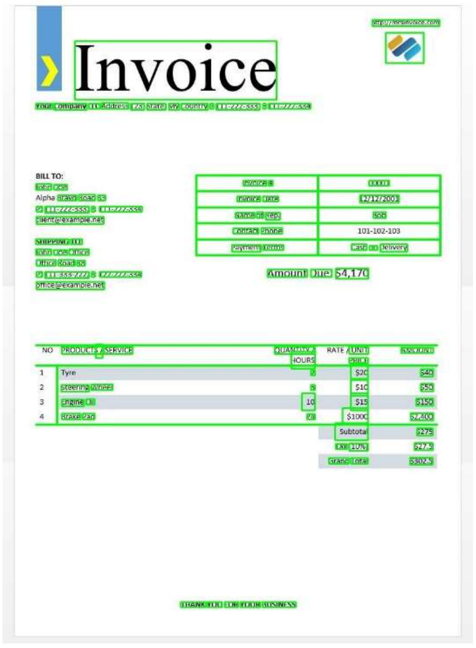


Figure 2.1 – Illustrative preprocessing steps, from the left: original image, grayscale, binarization and deskewing

Once the image is ready, the information extraction follows a step-by-step pipeline. The first step is a connected component analysis in which the outlines of the components are stored. By inspecting the nesting of outlines, and the number of child and grandchild outlines, it is simple to detect inverse text and recognize it as easily as black-on-white text. At this stage, outlines are gathered together, purely by nesting, into blobs. Blobs are organized into text lines, and the lines and regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Fixed pitch text is chopped immediately by character cells. Proportional text is broken into words using definite spaces and fuzzy spaces. Recognition then proceeds as a two-pass process. In the first pass, an attempt is made to recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to recognize more accurately the rest of the text in the page. Since the adaptive classifier may have learned some patterns at the end of the first run, a second pass is run over the page, in which words that were not recognized well enough are recognized again. The final phase resolves fuzzy spaces and checks alternative hypotheses for the x-height to locate text with different caps or fonts (Smith, 2007).



http://mrsinvoice.com

Invoice

Your Company [illegible] Address [illegible] State [illegible] Country [illegible]

BILL TO:
John Doe
Alpha Bravo Road 33
P: 111-222-333 F: 111-222-334
client@example.net

SHIPPING TO:
John Doe Office
Office Road 38
P: 111-333-222 F: 122-222-334
office@example.net

| | |
|---|---|
| Invoice # | [illegible] |
| Invoice Date | 12/12/2001 |
| Name of Rep. | Bob |
| Contact Phone | 101-102-103 |
| Payment Terms | Cash on Delivery |

Amount Due: $4,170

| NO | PRODUCTS / SERVICE | QUANTITY / HOURS | RATE / UNIT PRICE | AMOUNT |
|---|---|---|---|---|
| 1 | Tyre | [illegible] | $20 | $40 |
| 2 | Steering Wheel | 5 | $10 | $50 |
| 3 | Engine Oil | 10 | $15 | $150 |
| 4 | Brake Pad | [illegible] | $1000 | $2,400 |
| | | | Subtotal | $275 |
| | | | Tax 10% | $27.5 |
| | | | Grand Total | $302.5 |

THANK YOU FOR YOUR BUSINESS

Figure 2.2 - Example of Tesseract OCR text recognition

Beyond text detection there are some other major challenges faced by layout analysis, namely detecting text columns and table regions. When a text is split into columns it is essential to detect the edges of the layout since the OCR has a fixed orientation, commonly horizontal. This can be done with an algorithm for tab-stop detection (Smith, 2009). Table detection is also a hard problem since tables have a large variation in their layouts. This challenge can be decomposed into two steps that are: finding the boundaries of a table in an image page and recognizing the structure of rows and columns (Shafait et al., 2010).

As we mentioned before, in order to cope with the vast diversity of book content and typefaces, it is important for OCR systems to leverage the strong consistency within a book but adapt to variations across books. Lee and Smith (2012) propose a system that adapts to shapes and vocabularies within a book using the Tesseract engine as a baseline.

## 2.2. Language and text interpretation to extract information and build clusters

The digitization of documents is the first part of claim automation processing. After the information is extracted from the text it must be interpreted and categorized for business purposes. The text contains concepts with different degrees of complexity, from simple 1-word codes with fixed structure to complex sentences.

### 2.2.1. SAS Visual Studio Analytics

To implement these text analytics tasks, we will use the SAS Visual Text Analytics tool (SAS Institute Inc., 2019). This tool allows us to build a pipeline (figure 2.3), which is a process flow diagram that can be used to represent a sequence of nodes each performing analytical tasks such as processing the raw text, extracting the information and creating categories.

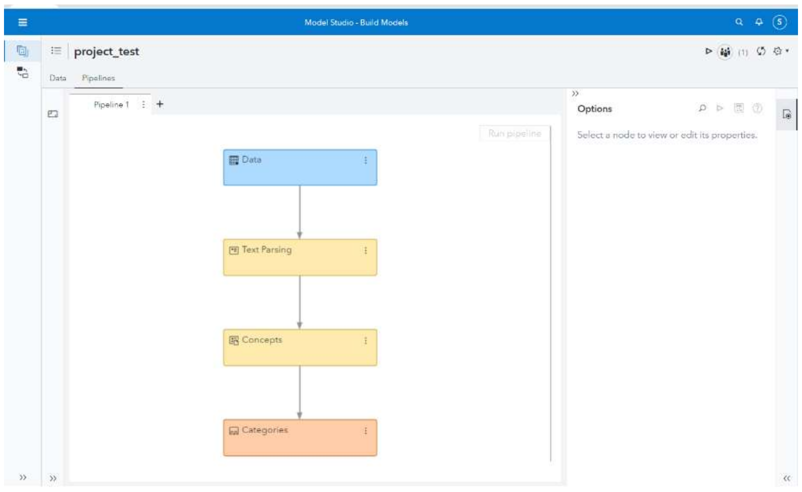
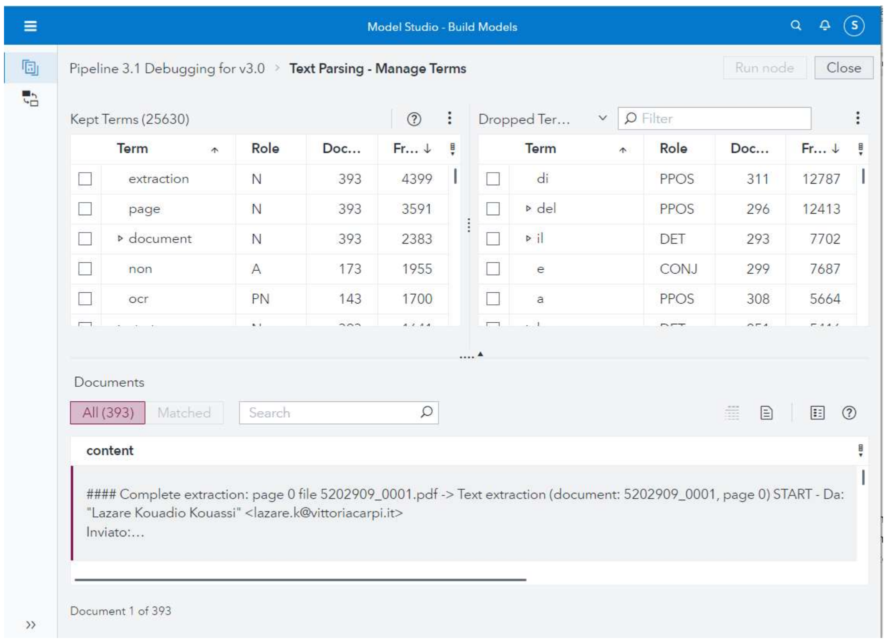


Figure 2.3 – Example of a pipeline in SAS Visual Studio Analytics

Nodes that will be involved in this project are the text parsing node, concepts node and categories nodes.

The Text Parsing node enables the user to view and explore the terms that are present in the document collection. During the parsing process, terms are either kept or dropped based on their importance. For example, terms that have the role of preposition or conjunction often provide minimal value, therefore they are dropped during text parsing. To gain a better understanding of how terms are related to each other, it is possible to generate a term map or similarity scores for a selected term to explore its relationship with other terms in your document collection. An example is shown below in figure 2.4.


Model Studio - Build Models
Pipeline 3.1 Debugging for v3.0 > Text Parsing - Manage Terms
Run node
Close

Kept Terms (25630)

| Term | Role | Doc... | Fr... |
|---|---|---|---|
| extraction | N | 393 | 4399 |
| page | N | 393 | 3591 |
| ▹ document | N | 393 | 2383 |
| non | A | 173 | 1955 |
| ocr | PN | 143 | 1700 |

Dropped Ter...

| Term | Role | Doc... | Fr... |
|---|---|---|---|
| di | PPOS | 311 | 12787 |
| ▹ del | PPOS | 296 | 12413 |
| ▹ il | DET | 293 | 7702 |
| e | CONJ | 299 | 7687 |
| a | PPOS | 308 | 5664 |

Documents

All (393) Matched Search

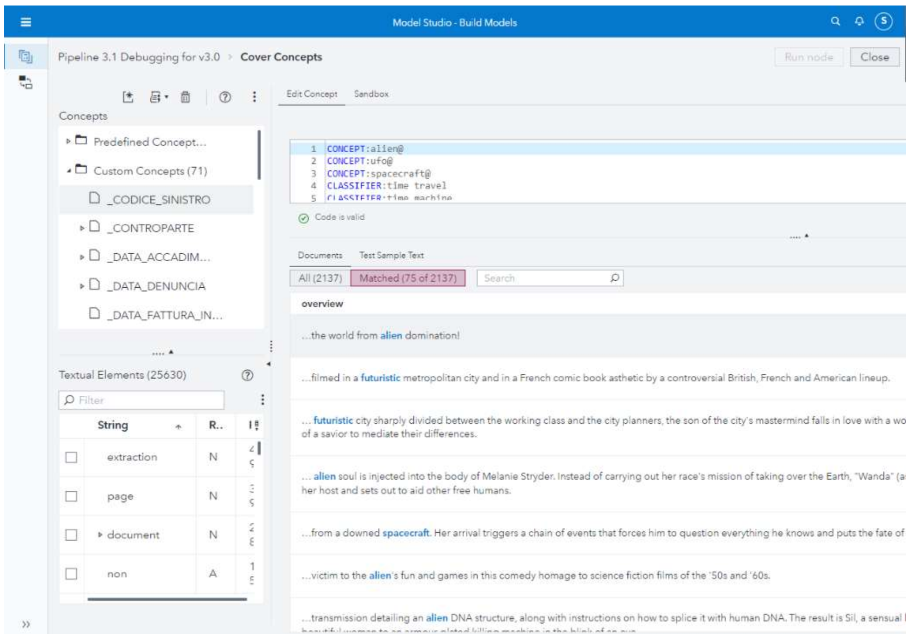
content

#### Complete extraction: page 0 file 5202909_0001.pdf -> Text extraction (document: 5202909_0001, page 0) START - Da: "Lazare Kouadio Kouassi" <lazare.k@vittoriacarpi.it>
Inviato:...

Document 1 of 393

Figure 2.4 - Example of a text parsing node

The Concepts node enables the user to work with semantic attributes, entity types, facts, or relationships and extracts pieces of the text using rules written in the language interpretation for textual information (LITI) syntax. An example is shown below in figure 2.5.



```
CONCEPT:alien@
CONCEPT:ufo@
CONCEPT:spacecraft@
CLASSIFIER:time travel
CLASSIFIER:time machine
```

| | String | R.. | I |
|---|---|---|---|
| ☐ | extraction | N | [illegible] |
| ☐ | page | N | [illegible] |
| ☐ | document | N | [illegible] |
| ☐ | non | A | [illegible] |

**overview**

...the world from **alien** domination!

...filmed in a **futuristic** metropolitan city and in a French comic book asthetic by a controversial British, French and American lineup.

... **futuristic** city sharply divided between the working class and the city planners, the son of the city's mastermind falls in love with a wo
of a savior to mediate their differences.

... **alien** soul is injected into the body of Melanie Stryder. Instead of carrying out her race's mission of taking over the Earth, "Wanda" (a
her host and sets out to aid other free humans.

...from a downed **spacecraft**. Her arrival triggers a chain of events that forces him to question everything he knows and puts the fate of

...victim to the **alien**'s fun and games in this comedy homage to science fiction films of the '50s and '60s.

...transmission detailing an **alien** DNA structure, along with instructions on how to splice it with human DNA. The result is Sil, a sensual

Figure 2.5 – Example of a concept node

Finally, a category identifies a group of documents that share a common characteristic. The Categories node enables the user to create categories in an almost identical way to the Concepts node.

# 3. METHODOLOGY

## 3.1. Process as-is and process to-be

As of today, the insurance company receives the majority of the documents via email which, besides the head and body of the mail, may contain also attachments as supporting documents to the communication, in the form of digital documents, scans and photos. All this data is converted to a pdf file identified by a unique key associated with the email. From this point, the procedure is manual: an operator reads the document and categorizes it depending on whether it is associated with an existing claim practice or if is a new request. Hence, he does the following:

- If it is an existing claim, all the new information must be added to the existing one in their data management software
- In case of a new claim request, the operator must do some administrative checks (for example check if the insurance policy is still valid when the accident happened) and create a new claim instance in the data management software with all the relevant data contained in the document

In the first meetings with the operators, several points of improvement in the streamline were identified. The main goal is to reduce the processing time of this job without a loss in the accuracy of the information extraction. Therefore, ideally, all the work of extracting the information and arranging it in categories will be done by the machine. To be more specific all the following tasks will be carried out:

- Extraction of any useful information from the documents
- Integration with all the supplementary information already existing in the database
- Calculation of the prioritization of the claims
- Check for technical and administrative coverage

### 3.1.1. Information extraction and additional data

In response to the first two points of the task list above, a table containing such information is created. This table will be defined throughout the chapter as the “cover” of the claim. It is filled mostly with data obtained through a textual analysis algorithm from the documents and the rest of the information is retrieved from the existing database through web service requests. In fact, for instance, if we know the policy identifier it is possible to retrieve all the information about the insured person and its policy contract. In table 3.1 it is reported a summary of all the features and a brief explanation of their type and meaning

Table 3.1 - List of features in the cover

| Feature ID | Feature name | Expected answer | Explanation | Source of the retrieved information |
|---|---|---|---|---|
| 1 | Date of occurrence | DD/MM/YYYY | Specific date when the fact occurred | Textual analysis |

| | | | | |
|---|---|---|---|---|
| 2 | Date of reporting | DD/MM/YYYY | First date when the insured person reported the fact | Textual analysis |
| 3 | Count of unpaid invoices | Integer number | Amount of unpaid invoices that the customer is reporting | Textual analysis |
| 4 | Date of unpaid invoices | DD/MM/YYYY | Date of each unpaid invoice that the customer is reporting | Textual analysis |
| 5 | List of unpaid invoices | List of alphanumeric ids separated by pipe character | Id of each of the unpaid invoices | Textual analysis |
| 6 | Policy ID | Alphanumeric code | Unique identifier for the policy contract between customer and company | Textual analysis |
| 7 | Other policies of the insured person | Alphanumeric codes separated by pipe character | List of other policies id that the customer may have | Web Service request |
| 8 | Retroactivity | Yes/No and years if Yes | Years of retroactivity of the policy | Web Service request |
| 9 | Motor & Debt | Yes/No | Whether the claim belongs to road accident or unpaid debt categories | Textual analysis |
| 10 | License plate | String | License plate code of the vehicle | Textual analysis |
| 11 | Cause of damage/Type of dispute | Class of a categorical variable | Label to define the nature of the dispute | Textual analysis |
| 12 | Risk type under warranty | Yes/No | Whether the claim is covered by the policy contract or not | Web Service request |
| 13 | Counterparty | String | Other parts involved in the fact | Textual analysis |
| 14 | Real estate address (in case of real estate dispute) | String | Address of the building if the claim is about a real estate dispute | Textual analysis |
| 15 | Judicial/extrajudicial phase | "Judicial" or "Extrajudicial" | Whether the claim involves violations of the penal code or not | Textual analysis |
| 16 | Number of Judicial/extrajudicial cases | Integer number | Maximum number of claims that can be refunded in a contract | Web Service request |
| 17 | Number of dispute cases already used by the insured person | Integer number | Referred to the previous, number of claims already refunded in a contract | Web Service request |
| 18 | Value of the damage/argument in object | Number with euro format | Total amount of money that are object of the dispute | Textual analysis |

| | | | | |
|---|---|---|---|---|
| 19 | Presence of legal representation | Yes/No and name if Yes | Whether there is a lawyer representing the insured person and in case its name | Textual analysis |
| 20 | Sender role | Class of a categorical variable | Role of the person reporting the fact in the first mail (insured person, mediator, lawyer…) | Textual analysis |
| 21 | Invoice directly assigned to the insurance company | Yes/No | Whether the unpaid invoice has to be paid directly by the insurance company | Textual analysis |
| 22 | New claim or follow-up | "New claim" or "Follow-up" | Whether the document belongs to a new claim or is a follow-up of an existing claim | Textual analysis |
| 23 | Priority[1] | Integer number | Number representing the hierarchy of the document in the queue | Textual analysis |
| 24 | Product code | Alphanumeric code | Code of the contract type between customer and insurance company | Web Service request |
| 25 | Regularity | Yes/No | Whether the date of occurrence and type of dispute are congruent with the covered period and damages in the contract | Web Service request |
| 26 | Completeness score | Decimal number between 0 and 1 | Index representing the proportion of features extracted from the document | Calculated index |

### 3.1.2. Calculation of the prioritization of the claims

The general rule to process the queue of documents is FIFO (First In First Out) which is to comply with the chronological order of arrival. But sometimes the queue grows and some of the practices can be postponed to prioritize more urgent documents. Therefore, the company defined some rules that establish a hierarchy among the documentation. These are enumerated from 1 to 6 and are as follows:

*Priority 1:* Top-class insurance policies that the company took out with reinsurance companies (reinsurance company is a type of insurance company that provides insurance coverage to other insurance companies. It is a practice in this sector to share risk and minimize the impact of large loss events. Reinsurance companies typically specialize in certain types of insurance coverage, such as life insurance, property and casualty insurance, or health insurance)

*Priority 2:* Other insurance policies that the company took out with reinsurance companies

*Priority 3:* Documents with keywords such as "urgent", "reminder", "complaint", "contract cancellation/termination" and documents that mention the "Institute for the Supervision of

[1] This is explained in-depth in paragraph 3.1.2.

Insurance” which is an independent authority responsible for supervising and regulating all insurance business.

*Priority 4:* Claims without the presence of legal representation

*Priority 5:* Claims of some specific legal spheres (opposition to administrative sanctions, civil mediations and others)

*Priority 6:* Any other document

Of course, if a document fulfils two or more different rules, the priority assigned is the highest one (the lowest in number)

#### 3.1.3. Check for technical and administrative coverage

The process As-is includes some checks that can be easily automated. In particular, before any monetary refund is approved by the company, it is necessary to verify if the customer paid off the insurance premium. Secondly, it must be verified whether the insurance policy is still valid, hence if the date of occurrence of the claim is between the date the contract started and the expiration date. Furthermore, the type of dispute must comply with the contract conditions, hence for the sake of the argument, a car crash can not be refunded if the customer was insured for its household. This kind of check can be automated with a web service request to the database.

The feature “regularity” was created in the cover to get these checks through a web service request. Hence, the claim is regular if the date of occurrence and type of dispute meet the requirements of the contract.

### 3.2. Data collection and storage

A data management software is used to store, organize and manage the customer’s data. The operator can fill in the important information collected from the documents through an interface. The database behind contains the documents, which are the elementary unit defined by a unique identifier associated with the email, and all the related features. The other important key besides the email identifier is the claim identifier, which is shared with all the documents related to a single claim.

Also the task of filling in the information can be automated, since from the cover an xml file is generated and sent to the company. The xml file format is suitable to be loaded on the data management system without compatibility issues.

However, every time a piece of information is needed, the operator can run a query to extract all the documents or set of values that match the request.

### 3.3. Process Flow

The structure of the process flow starts from the raw zip archives, which contain emails and attachments in several pdfs. The desired output of the pipeline is the cover table that reads the zip identifiers on the rows and the information extracted organized on the columns. Firstly, the input files need to be unzipped, and then transform all the contained information in the text. Once the text is extracted and cleaned, it is interpreted by a model that gets the information according to the features that are reported in table 3.1 and it returns the cover with the processed data.

### 3.3.1. Text extraction

For the first part of the process, we worked with Python and Pytesseract, which is the Tesseract OCR free tool made for Python. The reasons were that it was easier to handle raw files and to test many techniques locally, transforming image and text data. Moreover, these tools were open source and efficient enough for the first phase of the project.

The Sas program reported in Appendix A2. Proc python, includes the operations of ZIP archive download and pdf file extraction. Each pdf is opened with the PyMuPDF library; the metadata information, such as the number of pages, file name, author, etc. is collected in a separate csv file and the pdfs are processed to extract the text.

In the first phase of the project, we experimented with many solutions to transform the pdf files into text files. Even though the task is essentially trivial, the scan part turned out to be the most expensive operation in time and computational resources. Therefore, we evaluated to use it only when strictly necessary. For this reason, in a second moment, we worked to optimize the performance workflow. Indeed, this brought a great improvement in time performance and accuracy.

The content of the pdfs is miscellaneous and stored in many formats. This is due to the fact that the customers are of any sort: private citizens, lawyers, agents or even other companies. They are asked to comply with some guidelines when sending communications, but these are not mandatory. As a result, the variety in language and structure of the documents is very large. For this reason, we spent a great effort inspecting a sample of documents to figure out the most frequent ones. Studying the patterns of the most common documents gave us some insights to develop the workflow.

The scan of the documents is built up from each page, so here is a list of observations that we noted while looking at the single pages of the documents:

- All the text parts can be extracted without scanning the page. Direct text extraction takes an insignificant amount of time and the accuracy is perfect since the output equals the input.
- Most of the images are not optimal to be scanned by the OCR and need to be preprocessed first. In fact, they may come noisy, blurry, rotated, with incorrect brightness.
- When images are found, it occurs that the small ones do not bring relevant information, such as logos and signatures. Hence filtering those images out would prevent from spending time with the OCR extraction.
- If there is only one large image, we can run the OCR over that one instead of generating a scan of the entire page, this will save up time
- When there are many images, it occurs that all the relevant information is brought from the largest ones. Indeed, in this case, it is usually faster to generate a page scan and run the OCR once instead of repeating the task for each image.

These considerations were translated into Python code, which is reported in Appendix A below. In particular, the *text_image_page_scanner* function contains the core of those operations.

In the following Figure 3.1 we summarize the logical flow we draw to structure the code

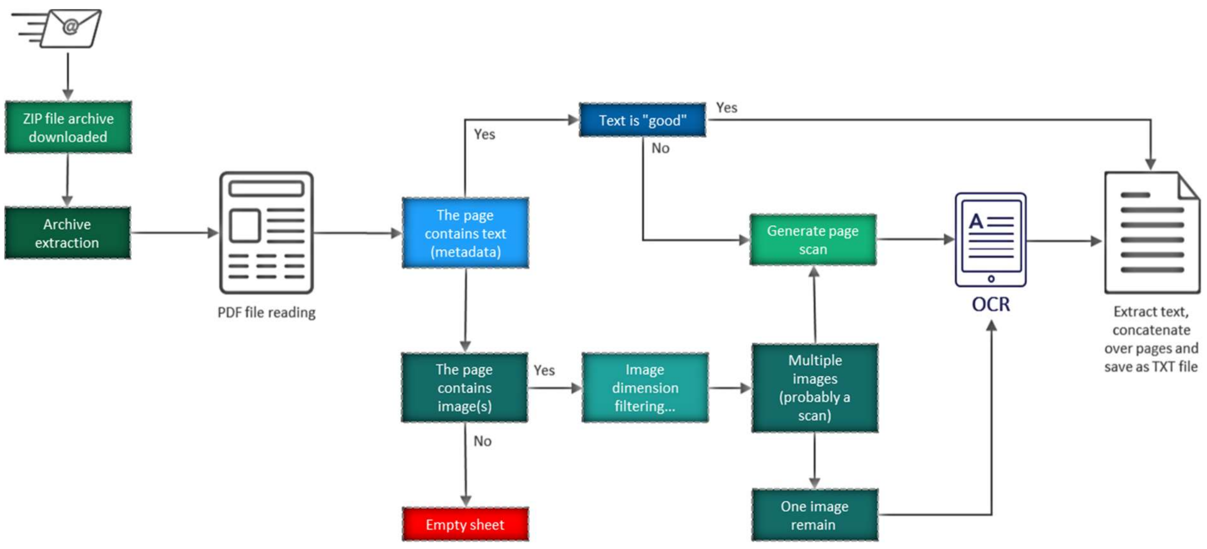

ZIP file archive downloaded
Archive extraction
PDF file reading
The page contains text (metadata)
Yes
Text is "good"
Yes
No
Generate page scan
OCR
Extract text, concatenate over pages and save as TXT file
The page contains image(s)
Yes
Image dimension filtering...
Multiple images (probably a scan)
No
Empty sheet
One image remain

Figure 3.1 – conceptual flow to extract the pdf files and transform them into Txt files

Even though the majority of the documents is being scanned correctly, some exceptions need to be faced in a further part of the code. In fact, there are few documents that may have no text and only images, among which for some reason one large white image as background and several small images with png format containing text and a transparent background layer. Hence, the algorithm that we designed filters out the images with text because of their size and reads the white large background only. As a result, the OCR scan is blank although it is clear that there is information in the page. Therefore, we added one more step for those images that are white (detected by selecting only those with an average rgb channel larger than 254) and with a blank extraction: a scan of the entire image is generated so that all the visible can be extracted.

This type of phenomenon was discovered in a further stage of the pipeline since some covers were being generated with no piece of information while the pdf documents had plenty of text. So, with the development of the project, there will be room to include eventual exceptions.

The first part shown in Figure 3.1, from the zip archive to the pdf was dealt with a Python procedure in Sas language. Hence it was possible to wrap the *text_image_page_scanner* python function within the rest of Sas code.

One additional consideration we made was about the position of the text in the document. Since the order of the information may be important, we generated a tag string in a Txt file at the beginning and the end of each page. Then for each page, we append text first and subsequently the image scan. Obviously, if a full-page scan is run then all the information contained fills one page in the txt file. Later on, we realized that this part could be improved for a future phase of the project by generating further tag strings as labels indicating either the position of the text in the page or the content of the text (i.e. “top-right corner, customer address”). In chapter 5, we will discuss this improvement and other possible ones.

The outcome of this step is then a text file for each document.

### 3.3.2. Visual Text Analytics

In this part of the flow, we built the model to interpret the information from the text and create the features that will fill the cover of each document.

Once the Txt files are collected and available, they are appended to a table with a unique identifier connected to each claim.

This data frame is then processed in the SAS Viya Visual Text Analytics (VTA). This tool can scale the human act of reading, organizing, and extracting useful information from huge volumes of textual data. In the following Figure 3.2 it is represented the structure of the flow to score the text files in concepts and categories.

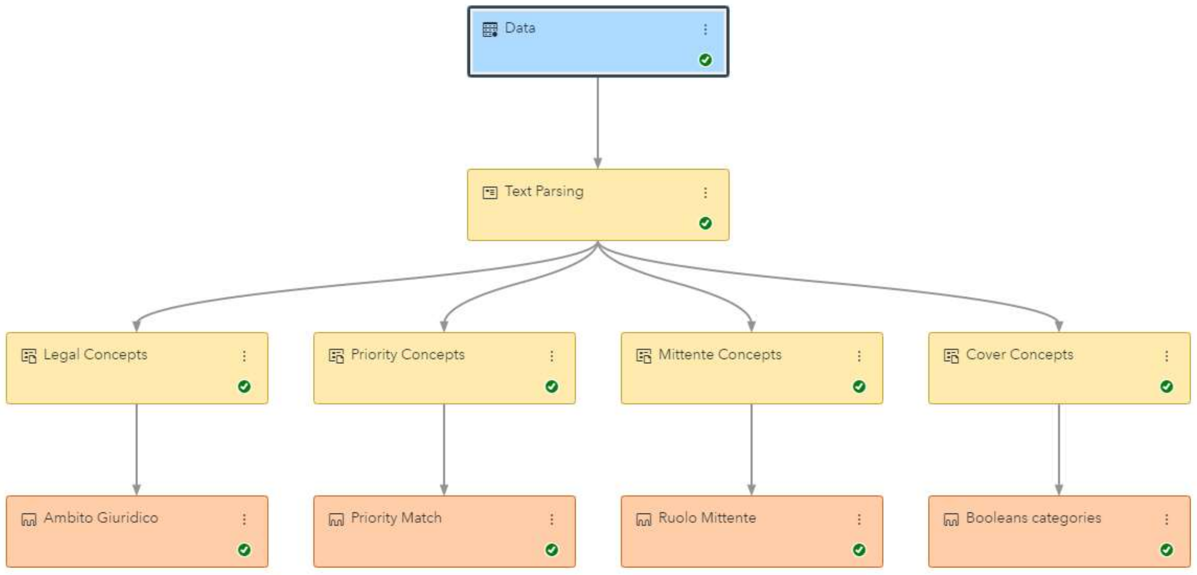


Figure 3.2 - Pipeline used for the project

In the first place, the whole text is parsed: all the non-important stop words are removed and part-of-speech filtering is applied.

Then we build the concept nodes, organized in dedicated areas of the features: general cover concepts, legal concepts, priority concepts and sender concepts. The reason was to work with slim parts of the flow separately. For each node we have several primary and supporting concepts containing all the rules written using LITI (language interpretation for textual information) syntax. Concept rules exploit semantic attributes, entity types, distance relationships and text order to recognize items in context so that you can extract only the pieces of the document that match the rule. The set of matches for each concept is then associated with the identifier of the original data frame.

The last step of the pipeline is the category node. In short, the categorical variables in the cover are made on top of some concept rules, so once a specific rule has found some matches in the text, one categorical label will be assigned to all the documents that contain those matches. For instance, the concept "Presence of legal representation" matches any lawyer name or law firm denomination; hence in the categorical node we assign the label "lawyer yes" to all those documents that contain such matches and "lawyer no" otherwise.

The training of the model was done side by side with the employees of the company who are currently doing this operation manually. Several meetings were needed to acknowledge the meaning of each concept and discuss how to match the correct information in practice. Hence, for example, we learnt that the policy number can be either a sequence of 10 digits or an alphanumeric sequence of 13 characters (5 digits + 3 letters + 5 digits). So, the first step is to search in the documents for such sequences through a regex. Of course, the matches may contain wrong information (i.e. mobile number of 10 digits), so we need a more detailed rule to match only the right sequences. Therefore, we combine this rule in a concept rule, where the result of such a regex must be matched only if it is near the keywords “policy number”, or “policy nr.”, et al.

The rule-tuning then is a step where we train the model to imitate the human process. This procedure requires a significant amount of time; rules also need to be updated over time to be consistent.

Around 150 concepts and 50 categories were created among the nodes.

### 3.3.3. Cover generation

This step of the flow concerns the creation of the cover starting from the model trained in Sas VTA. This part was done in Sas language and essentially, after a data postprocessing stage, the cover is filled with those features that can be obtained with web service requests to the company’s database. Hence, the cover is generated and all the features are stored in an xml file which is delivered to the company to add it to their database. The following Figure 3.3 represents the logical flow from the analysis of the text files to the generation of the cover and the xml output.

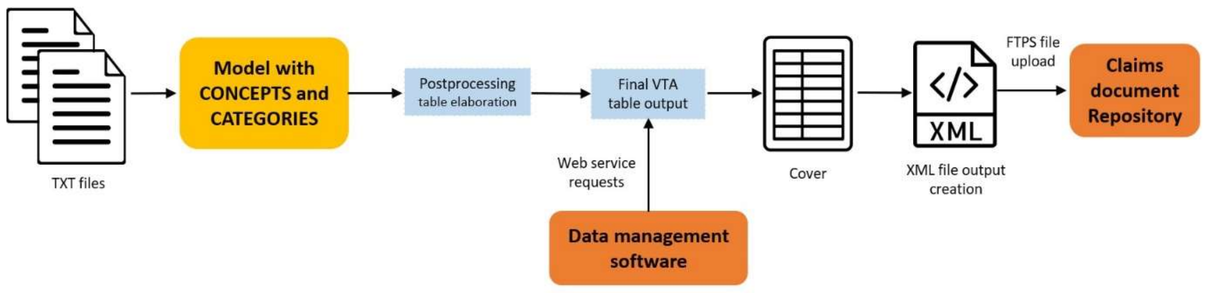


Figure 3.3 - conceptual flow to elaborate data in the txt files, create the cover and the xml file

### 3.3.4. Performance Dashboard

The whole process is being monitored in terms of time performance and statistics of the processed data. That is done through a performance dashboard where we report the main KPIs highlighting the amount of information processed, namely the number of ZIP archives, documents and pages processed. To assess the work progress, a completeness score and accuracy score are provided for the entire data. Further features are represented through bar charts and pie charts that can interact with each other. In addition, the processing time is being analyzed to assess the performance of the entire end-to-end flow. In the following figures 3.4 and 3.5 the two main dashboard tabs are shown.

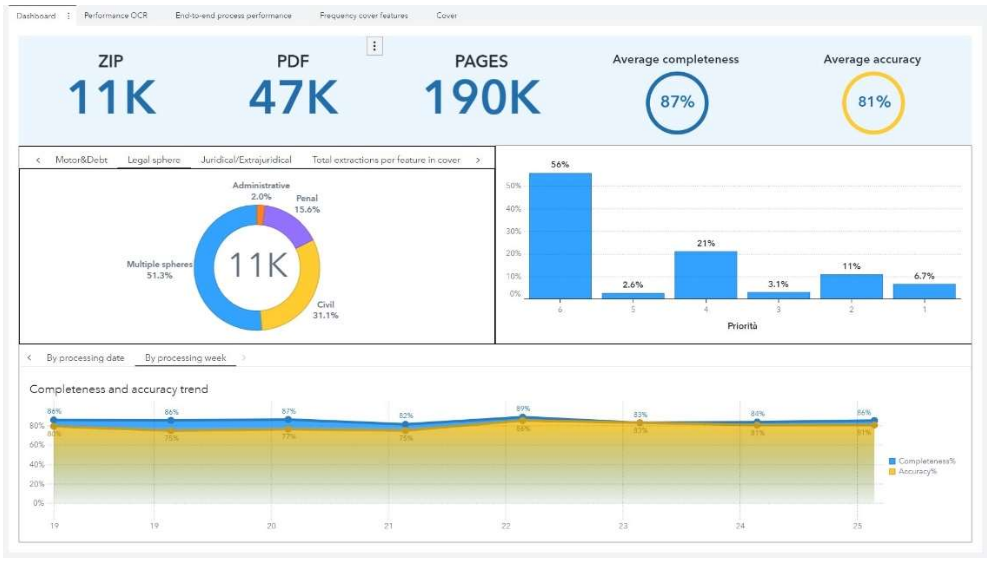


Figure 3.4 - Key performance indicators in the dashboard

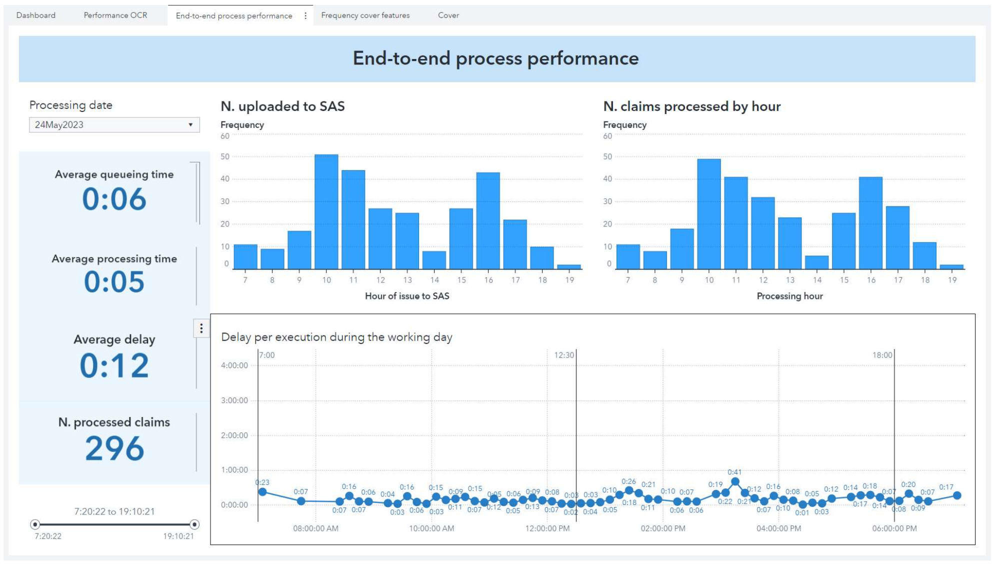


Figure 3.5 - End-to-end process performance section of the dashboard

# 4. RESULTS AND DISCUSSION

## 4.1. Completeness and accuracy

For the purpose of the project, we had to make sure that the piece of information that is extracted from each document is appropriately complete and accurate.

### 4.1.1. Completeness index

The completeness index is a feature that is included in the cover and it is calculated as the amount of extracted features over the total number of features.

$$COMPLETENESS = \frac{N° \; EXTRACTED \; FEATURES}{TOT \; FEATURES}$$

Indeed, not all the documents may have enough information to fill all the features, it may be that some of them are missing because they are not needed, or because they are not coherent with the type of claim (i.e., a list of unpaid invoices is not available for a road accident claim and license plate is not available for credit recovery claim). Therefore, if the claim type is recognized then the corresponding incoherent features are subtracted from the index in the following way:

$$COMPLETENESS = \frac{N° \; EXTRACTED \; FEATURES - INCOHERENT \; FEATURES}{TOT \; FEATURES - INCOHERENT \; FEATURES}$$

As the process will be implemented, a threshold for minimum completeness will be set in order to isolate those documents whose covers are too poor of data and analyze them manually.

### 4.1.2. Accuracy index

The accuracy index, on the other hand, is calculated in comparison to the true data. This means that to set up this index an operator is needed to check the same documents that are processed by the OCR system in order to compare their outcome. This was done over a period of 2 months to verify whether there was an improvement in accuracy while updating some parts of the process, especially the text analysis part. Since not all the features are mandatory when opening a new claim, the insurance company required to calculate it over 14 features, which are: Date of occurrence, Date of reporting, Policy ID, Contractor, Motor & Debt, Cause of damage/Type of dispute, Number of Judicial/extrajudicial cases, Number of dispute cases already used by the insured person, Presence of legal representation, Invoice directly assigned to the insurance company, New claim or follow-up, Regularity, Completeness score. Hence it is calculated as follows

$$ACCURACY = \frac{N° \; CORRECT \; MANDATORY \; FEATURES}{TOT \; MANDATORY \; FEATURES}$$

Note that if mandatory information is not in the document, the empty extraction from the OCR system will be considered as a correct answer by the operator.

## 4.2. Time performance

The time execution of the process was also monitored to certify the improvement obtained with automation instead of manual processing. Indeed, an operator can process on average 4 to 8 emails per hour, depending on the amount of content and attachments. In Figure 4.1 we represent the hourly flow of emails in the inbox of the company grouped by weekday and averaged over two months. Note that all the emails are forwarded to the operators only in between 7 am and 8 pm every day, Sunday excluded. This means that if a customer sends an email outside these time slots, it can be automatically postponed to the first available moment of the following time slot (i.e., if the company receives an email after 8 pm they do not read it before 7 am of the following day).

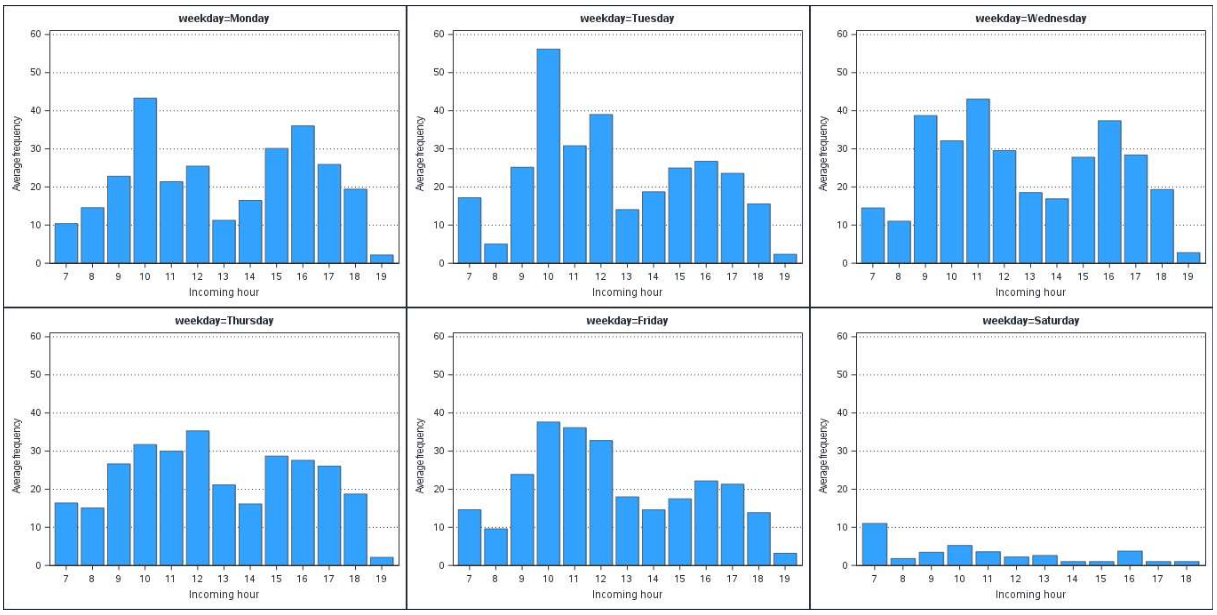


Figure 4.1 - hourly flow of emails grouped by weekday and averaged over two months

As it is now, on average for a claim the process takes on average 1 minute and 11 seconds to convert all the documentation through the OCR into a text file and process it with the concept and categories rules. When running the entire flow end-to-end there are also roughly 25 seconds of execution that are necessary steps of the pipeline (such as the creation of an execution log, creation of *cas cloud* session, creation of the ftps connection, check for the presence of zip files in the queue, creation of metadata of the execution, update of the existing tables, termination of the session).

We wanted to minimize the delay which is the time elapsed between the arriving time of the email and the upload time of the cover to the company. Assuming that as soon as the email is received it is in the queue to be processed, we can also define the delay as:

$$DELAY = QUEUEING\ TIME + PROCESSING\ TIME$$

Considering that:

- Even though it is possible to have overlapping executions (hence more than one execution per time) we wanted to avoid an overload of the machine CPU since its computational power is limited.
- An execution of a batch of claims was preferable to execute them singularly since we wanted to produce a restrained number of logs and metadata files
- The company set for the delay an indicative maximum threshold of 45 minutes

We scheduled an execution of a batch of 5 claims every 10 minutes for a processing rate of 30 claims per hour. The average processing time of such a schedule is 5 minutes and 36 seconds which is abundantly less than the scheduled interval of 10 minutes; this way there is also a margin for executions that take longer than the average processing time.

Looking again at figure Figure 4.4.1, there are still some time spans in which the arrival rate is higher than the processing rate of 30 claims per hour. Those may happen between Monday and Friday and especially in the morning (9 pm-12 pm) and in the middle of the afternoon (16 pm). So we set up a specific schedule for this time spans with a scheduled execution of a batch of 5 claims every 6 minutes. For a processing rate of 50 claims per hour. This way the average processing time is still lower than the scheduled interval of 6 minutes and even though there is less margin for executions that take longer than the average processing time the queue will not explode because the processing rate is higher than the arrival rate.

## 4.3. COMPARISON WITH SOME KPIS

Although it is not possible to measure precisely the performance of a human operation due to the variability of the documentation and the abstraction of the dynamics of this task, we would like to report a comparison of the performances of the human-driven process and the machine-driven process to point out the improvement that this project brought to the company.

In the following table 4.1 we report the comparison of the key performance indicators of the two processes.

Table 4.1 - Performance overview, manual vs. automated process

| | Manual process (per operator) | Automated process |
|---|---|---|
| range of claims processed per hour | 4 to 8 | 30 to 50 |
| Average accuracy | 95%[2] | 81% |
| Average completeness | 90% | 86% |

Certainly, the efficiency of the process has largely improved since the automated process can handle the entire load of incoming claims while there was a need for multiple operators to elaborate the same load. This led to a great achievement for the company since many resources are saved or employed in other tasks. Therefore, we can confirm that the objective of automating the reading, comprehension and extraction of all the information to the data management software was successful; the project is currently employed by the insurance company. However, the accuracy and completeness are still not as good as the ones produced by the operators, so there is still one operator checking the correctness of the covers and filling with data some blank features that are supposed to contain information. Some of the reasons were already mentioned throughout this

[2] Also in the human-driven process some mistakes are made, an estimation of 5% of errors is made by the company for a first manual elaboration which is eventually corrected by other employees in a second moment

report; considering the type of data and the limitations of the solution implemented in this project the results are good, even slightly better than expected. After a reflection, we agreed that further improvement would come from a change in the input data or the technology implemented. We will discuss the limitations of the automated process and suggest some recommendations for future work in the following chapter.

# 5. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

The research conducted in this project has provided a valuable solution for the automation of claims processing. However, like all research, it is not without its limitations and areas for improvement. It is important to acknowledge these limitations and provide recommendations for the future. This section will discuss the limitations of the present study and suggestions for future research.

## 5.1. LIMITATIONS

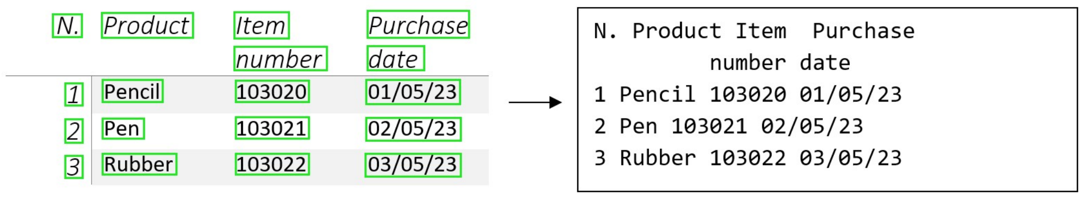
Although OCR technology carries out diligent work, it has some limitations in the effectiveness of reading the documents. Some of the lack of accuracy is due to its nature because of some intrinsic limitations of the technology. These are mainly low-quality images and handwritten text.

As a matter of fact, OCR relies on the quality of the image, meaning that if an image is not clear or has a low resolution, the OCR system may not be able to recognize the text accurately. The extent to which the image can be preprocessed sometimes it is not sufficient or it cannot be generalized for the flow a priori. In fact, similarly to other parts of the process, we established a sequence of preprocessing functions to have an optimal result for the heterogeneous pool of images that are contained in the documents while taking into consideration the trade-off with the execution time. To put it simply, it is in not worth it to spend too much time preprocessing an image that may bring anyways a bad text extraction because of its low quality.

Furthermore, OCR is still not perfect at recognizing handwritten text, especially the Tesseract engine involved in this project, so documents that contain a significant amount of handwriting need to be manually processed. Another limitation of OCR technology is in its ability to recognize non-textual elements, such as images or diagrams, making them difficult to filter and extract automatically. Finally, OCR usually handles all the text information uniformly; however, when the text is composed of mixed structures, including headers, body, and tables, and features such as font, size, and orientation, there may be a higher probability of information loss when translated to a formatted text.

For what concerns the text interpretation part, there is no empirical rule that can fully identify the correct information and isolate it from the rest of the text. In the approach implemented in this project, several rules were set up and their combination brought to a discrete accuracy. However, this method has a saturation threshold, meaning that after a certain amount of rules, the insertion of a new one may bring an improvement for a few instances but at the same time have a worse impact on the rest of the instances. Hence, we considered the combination of rules that brought the optimal accuracy for the given set of instances.

A further limitation of this part was related to the OCR and its weakness in detecting the structure of information, namely tables, font size, and position. Having a loss in the text structure means the interpretation is not as good, since for instance it is harder to relate the meaning of a date or number in a flat text when it was originally connected to its position in the document. The example documented in Figure 5.1 shows that the table information is extracted as flat text, thus it is not possible to exploit the knowledge contained in the vertical position of the column labels if the text is being examined horizontally.



| N. | Product | Item number | Purchase date |
|---|---|---|---|
| 1 | Pencil | 103020 | 01/05/23 |
| 2 | Pen | 103021 | 02/05/23 |
| 3 | Rubber | 103022 | 03/05/23 |

```
N. Product Item  Purchase
         number date
1 Pencil 103020 01/05/23
2 Pen 103021 02/05/23
3 Rubber 103022 03/05/23
```

Figure 5.1- example of weak table extraction from OCR

## 5.2. Recommendations

To address the lack of accuracy caused by OCR limitations, it is crucial to update the communication system between customers that file insurance claims and the company. According to many authors in the literature, the accuracy of OCR systems directly depends upon the quality of input (Rice et al., 1999; Chaudhuri et al., 2017). Currently, customers send emails and documents in various formats that may lead to information loss at some stage of the process. This can be resolved by implementing a standardized claim filing module, which could simplify the interpretation of information. If the essential information is required to be sent in a uniform format, it would ease the processing of data. Since this solution might not be implemented rapidly, it might be considered also a replacement of the Tesseract engine with another OCR which can better detect non-structured text and handwritten text.

Another recommendation concerns the strategy implemented to extract information from the text. Currently, the OCR engine extracts all text information from the documents. These texts are unstructured and, as we explained previously, the structure of the text sometimes has a role in its interpretation. Hence, incorporating meta-labels to differentiate the various sections of the document, or employing an alternative tool to extract structured data like tables, would facilitate the process of rule tuning. Also preprocessing the text to evaluate the meaning of its content and assign a label would help improve the interpretation of the text when dealing with concepts and categories. For instance, we had some issues excluding documents containing a scan of the insurance contract: in fact, these documents contain data that is already known by the company and have no useful information for the cover; at the same time, they are usually long and plenty of keywords that could bring some noise in the design phase of concept rules. Knowing that the document contains only a scan of the insurance contract would bring us to exclude it from the process.

# 6. CONCLUSION

As part of my internship at SAS Institute, this research was conducted for a project aimed at developing an automated system to handle the inbound claims of an insurance company. Currently, these claims are processed manually. Some literature was presented to introduce the existing technology and knowledge related to this problem and, after that, an extended explanation was given for the methodology used in practice for this project. In particular, the tasks made in the process as-is and the plan for the process to-be. The product delivered to the company is a “cover” summing up all the extracted information, the prioritization policy of the claims and a check for technical and administrative coverage. Afterward, we went through storing data and the implementation of the process flow. For this last one, we provided a detailed description of the text extraction and interpretation.

Along with the methodology, we analyzed some statistics and discussed the results of the new automated system. There is evidence for a great improvement in efficiency and reduction of spending of resources for the insurance company. Although the automation is feasible, the completeness and accuracy of the new process are still not optimal; this is due to some limitations of the OCR technology itself and the variety of the type of documents contained in the claims. To address a solution to such limitations and improve the outcomes, some recommendations for future works were given, such as considering the modernization of the current system in which the customers send the documentation to the insurance company, which may help a lot the automation process, or facilitating the interpretation of the text by adding meta-labels to the text files.

## 7. REFERENCES

Tesseract guide: https://tesseract-ocr.github.io/tessdoc/

Chaudhuri, A., Mandaviya, K., Ghosh, S. K., & Badelia, P. (2017). Optical Character Recognition Systems for Different Languages with Soft Computing. In Studies in fuzziness and soft computing. Springer Nature. https://doi.org/10.1007/978-3-319-50252-6

Lee, D., & Smith, R. F. (2012). Improving Book OCR by Adaptive Language and Image Models. https://doi.org/10.1109/das.2012.45

Lopresti, D. P. (2009). Optical character recognition errors and their effects on natural language processing. International Journal on Document Analysis and Recognition, 12(3), 141–151. https://doi.org/10.1007/s10032-009-0094-8

Ma, T., Yue, M., Yuan, C., & Yuan, H. (2021). File Text Recognition and Management System Based on Tesseract-OCR. In 2021 3rd International Conference on Applied Machine Learning (ICAML), 1-4. https://doi.org/10.1109/icaml54311.2021.00057

Miner, G. D., Elder, J. P., Hill, T. C. J., Nisbet, R. A., Delen, D., & Fast, A. S. (2012). Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications. In Elsevier eBooks, pp. 1007–1016. https://doi.org/10.1016/c2010-0-66188-8

P. S. Hiremath, J. D. Pujari, S. Shivashankar and V. Mouneswara, (2010). "Script identification in a handwritten document image using texture features," IEEE 2nd International Advance Computing Conference (IACC), Patiala, India, 2010, pp. 110-114, https://doi.org/10.1109/IADCC.2010.5423028.

Rice, S., Nagy, G., & Nartker, T. A. (1999). Optical Character Recognition: An Illustrated Guide to the Frontier. In Springer eBooks. https://doi.org/10.1007/978-1-4615-5021-1

Rusu, A., & Govindaraju, V. (2004). Handwritten CAPTCHA: Using the Difference in the Abilities of Humans and Machines in Reading Handwritten Words. https://doi.org/10.1109/iwfhr.2004.54

Rusu, A., Thomas, A. W., & Govindaraju, V. (2009). Generation and use of handwritten CAPTCHAs. International Journal on Document Analysis and Recognition, 13(1), 49–64. https://doi.org/10.1007/s10032-009-0102-z

Sankar, K. P., Jawahar, C. V., & Manmatha, R. (2010). Nearest neighbor based collection OCR. https://doi.org/10.1145/1815330.1815357

SAS Institute Inc. (2019). SAS® Visual Text Analytics 8.4: User’s Guide. Cary, NC: SAS Institute Inc., 1-172.

Shafait, F., & Smith, R. F. (2010). Table detection in heterogeneous documents, 1-9. https://doi.org/10.1145/1815330.1815339

Smith, R. J. E. (2007). An Overview of the Tesseract OCR Engine, 1-5. https://doi.org/10.1109/icdar.2007.4376991

Smith, R. F. (2009). Hybrid Page Layout Analysis via Tab-Stop Detection. https://doi.org/10.1109/icdar.2009.257

# APPENDIX

## A. CODE TO BUILD THE OCR

### A1. Python functions

```
## Import libraries
import cv2
import os
import pytesseract
from skimage import *
from deskew import determine_skew
import re
import numpy as np
import logging
import time
from PIL import Image

## Set Tesseract directory
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

## Define functions
def noise_removal_and_smoothening(img):
    img_u=img.astype(np.uint8)
    filtered = cv2.adaptiveThreshold(img_u, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 41,3)
    kernel = np.ones((1, 1), np.uint8)
    opening = cv2.morphologyEx(filtered, cv2.MORPH_OPEN, kernel)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
    #image smoothening
    ret1, th1 = cv2.threshold(img_u, 180, 255, cv2.THRESH_BINARY)
    ret2, th2 = cv2.threshold(th1, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    blur = cv2.GaussianBlur(th2, (1, 1), 0)
    ret3, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    or_image = cv2.bitwise_or(th3, closing)
    return or_image

def adjust_contrast(img, clipLimit, tileGridSize, brightness, contrast):
    try:
        lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
        l_channel, a, b = cv2.split(lab)
        clahe = cv2.createCLAHE(clipLimit=clipLimit, tileGridSize=tileGridSize)
        cl = clahe.apply(l_channel)
        limg = cv2.merge((cl,a,b))
        enhanced_img = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    except:
        img = np.int16(img)
        img = img * (contrast/127+1) - contrast + brightness
        img = np.clip(img, 0, 255)
        img = np.uint8(img)
        enhanced_img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
    return enhanced_img

def deskew(image, dimension_limit):
    '''
    Rotate the image automatically to increase the OCR precision. If the image is too big,
    the function will interrupt the process and the original image is returned
    '''
    if image.shape[0] > dimension_limit:
        print('The image has not been processed because over the dimension_limit')
        return image.astype(np.uint8)
    else:
        angle = determine_skew(image)
        if angle == None:
            rotated = rotate(image, 0, resize=True, preserve_range=False) * 255
            print('angle was not calculated by the function. Rotation is se to angle = 0')
        else:
            _t_start = time.time()
            rotated = rotate(image, angle, resize=True, preserve_range=False) * 255
            _t_end = time.time()
            print('Image rotated, angle = {a}'.format(a=angle))
            print('Deskew execution time {b}'.format(b=round(_t_end-_t_start, 2)))
        return rotated.astype(np.uint8)

def process_image_for_ocr(im, _noise_removal=True, _contrast=True, _deskew=True):
    im_arr = np.asarray(im)
    im_arr_g = cv2.cvtColor(im_arr, cv2.COLOR_RGB2GRAY)
    if _noise_removal:
        im_arr_g = noise_removal_and_smoothening(im_arr_g)
        print('Noise and Smoothness adjustments completed')
    if _contrast:
        im_arr_g = adjust_contrast(im_arr_g, 1.5, (4,4), 50, 30, False, (1400, 900), False)
        print('Contrast and Brightness adjustment completed')
    if _deskew:
        im_arr_g = deskew(im_arr_g, 4000)
        print('Image rotation adjustment completed')
    return im_arr_g

def ocr_text_extracion(img, prefix, suffix, lang):
    '''
    The function perform OCR image_to_string().
    The user could add a prefix and a suffix to the generated string.
    '''
    _t_start = time.time()
    __temp_string = pytesseract.image_to_string(img, lang=lang)
    _t_end = time.time()
    print('OCR process completed in {a} seconds'.format(a=round(_t_end-_t_start, 2)))
    return prefix + ' ' + __temp_string + ' ' + suffix

## scan document page for image generation and OCR
def text_image_page_scanner(page, doc, doc_name, num_chars, width_size, height_size, path_to_img,
image_preprocessing=True):
    '''
    - The function returns the text content of a PDF page imported using PyMuPDF.
    - The num_chars parameter controls the min lenght of characters to skip the OCR
    and extract the text directly. if there are less characters than this parameter
    the function returns the image(s) in the page. Also, if the text contains strange
    characters (bytes like), the entire page will be scanned and the OCR will be run.
    - The images are filtered according to their dimentions: (width_size, height_size)
    if the page contains 1 image, then the runs over the only image in the page.
    if the page contains >1 images larger than the 'width and height' filter,
    the function scans the entire page and returns a png of the page.
    '''
    _num_page = page.number
    _page_text = page.get_text()
    _doc_page_tag_string = ' - (document: {a}, page {b})'.format(a=doc_name, b=_num_page)
    prefix='OCR extraction start:'
    suffix='OCR extraction end'
    if len(_page_text) > num_chars:
        print('Text extraction returns more than {a} characters'.format(a=num_chars))
        print('for page {b}. OCR procedure is not necessary'.format(b=_num_page))

        if _page_text.find('\x01') + _page_text.find('\x02') + _page_text.find('\x03') > 0:
            print('The string contains unknown characters. The entire page will be scanned')
            _page_scan = page.get_pixmap(alpha=False, dpi=300, colorspace='rgb')
            img = Image.frombytes('RGB', [_page_scan.width, _page_scan.height], _page_scan.samples)

            if image_preprocessing:
                img_postprocess = process_image_for_ocr(img)
            else:
                img_postprocess = img

            _text_from_img = ocr_text_extracion(img_postprocess, prefix, suffix, 'ita')
            _tag_string ='Page OCR scan' + _doc_page_tag_string
            return _tag_string + ' - ' + _text_from_img + ' - ' + _tag_string
        else:
            _tag_string = 'Direct text extraction' + _doc_page_tag_string
            return _tag_string + ' START - ' + _page_text + ' - END ' + _tag_string
    else:
        print('The page {a} does not contain text. OCR procedure is initialized'.format(a=_num_page))
        imgpage = page.get_image_info(xrefs=True)
        print('{a} image(s) found in page {b}.'.format(a=len(imgpage), b=_num_page))
        _cleaned_images = [k for k in imgpage if (k['width'] > width_size) and (k['height'] >
height_size)]

        if len(_cleaned_images) == 0:
            print('No images found in page {a}. page scan using pixmap will start'.format(a=_num_page))
            _page_scan = page.get_pixmap(alpha=False, dpi=300, colorspace='rgb')
            img = Image.frombytes('RGB', [_page_scan.width, _page_scan.height], _page_scan.samples)

        elif len(_cleaned_images) == 1:
            print('One image found in page {a}. Image extraction using PyMuPDF'.format(a=_num_page))
            __img = _cleaned_images[0]
            _img_xref, _img_wdh, _img_hgt = __img['xref'], __img['width'], __img['height']
            print('Analyzing image with xref = {b}'.format(b=_img_xref))
            print('Image xref {c} resolution = {a}'.format(a=(_img_wdh, _img_hgt), c=_img_xref))
            _image_dict = doc.extract_image(_img_xref)
            _img_ext = _image_dict.get('ext')
            _path_to_image = path_to_img + '/' + doc_name + '_img_xref{z}_{p}.{a}'.format(a=_img_ext,
p=page.number, z=_img_xref)
            img = open(_path_to_image, "wb")
            img.write(_image_dict.get('image'))
            img.close()
            print('Image created. Path to the image is {a}'.format(a=_path_to_image))
            img = cv2.imread(_path_to_image)
            print('Image reimported')
            os.remove(_path_to_image)
            print('Image deleted')

        elif len(_cleaned_images) > 1:
            print('{c} image(s) found at the'.format(c=len(_cleaned_images)))
            print('page {a}. page scan using pixmap will start'.format(a=_num_page))
            _page_scan = page.get_pixmap(alpha=False, dpi=300, colorspace='rgb')
            img = Image.frombytes('RGB', [_page_scan.width, _page_scan.height], _page_scan.samples)

        if image_preprocessing:
            img_postprocess = process_image_for_ocr(img)
        else:
            img_postprocess = img

        if img_postprocess.shape[0] < 4000:
            _text_from_img = ocr_text_extracion(img_postprocess, prefix, suffix, 'ita')
        else:
            return 'Memory limit reached ' + _doc_page_tag_string + ' START - END'

        '''
        This further step is for some rare cases in which the text extraction from images is
        empty and the image is almost completely white (images with rgb>254). This is the case
        of some pdfs with no written text, one white image as a background and several images
        with some actual text but smaller than the size filter of (500px X 500px).
        The solution is an entire page scan
        '''
        if len(_text_from_img)-(len(prefix)+len(suffix))<=0 and img_postprocess.mean()>254:
            print('image {c} text extraction is empty.'.format(c=len(_cleaned_images)))
            print('The entire page {a} will be scanned using the pixmap'.format(a=_num_page))
            _page_scan = page.get_pixmap(alpha=False, dpi=300, colorspace='rgb')
            img = Image.frombytes('RGB', [_page_scan.width, _page_scan.height], _page_scan.samples)
            if image_preprocessing:
                img_postprocess = process_image_for_ocr(img)
            else:
                img_postprocess = img
            if img_postprocess.shape[0] <4000:
                _text_from_img = ocr_text_extracion(img_postprocess, prefix, suffix, 'ita')
            else:
                return 'Memory limit reached ' + _doc_page_tag_string + ' START - END'

        _tag_string = 'OCR extraction ' + _doc_page_tag_string
        return _tag_string + ' - ' + _text_from_img + ' - ' + _tag_string

def txt_cleaner(txt, basic_operation=True):
    regex = r'\\r|\\n|\r|\r\n'
    regex2 = r' +'
    regex3 = r'\n \n'
    if basic_operation:
        txt = re.sub(regex, '\n', txt)
        txt = re.sub(regex2, ' ', txt)
        txt = re.sub(regex3, '\n\n', txt)
        print('Text cleaned according to specified operation(s)')
    else:
        txt = txt
        print('Function returned the original text')
    return txt
```

## A2. Proc python

```
/* -- PROC PYTHON: PDF file OCR and elaborations -- */
filename pyfunc "/mnt/nfs/viya/python_functions/ocr_image_txt_processing_functions.py";
proc python infile=pyfunc restart;
submit;

## import libraries
import time
import os
import logging
import pandas as pd
import fitz
import shutil

## bring paths into python
path_to_sin_folders = SAS.symget('path_to_sin_folders')
path_to_temp_data = SAS.symget('path_to_temp_data')
path_to_metadata_out = SAS.symget('path_to_metadata_out')
path_to_sin_folders_out = SAS.symget('path_to_sin_folders_out')
path_to_log = SAS.symget('path_to_log')
path_to_sin_move = SAS.symget('path_to_sin_move')
timestamp_macro = SAS.symget('timestamp')

## log configuration
log_file = os.path.join(path_to_log, 'ocr_{a}.log'.format(a=timestamp_macro))
logging.basicConfig(filename=log_file,
                    filemode='w',
                    level=logging.DEBUG,
                    format='%(asctime)s [%(levelname)s] - %(message)s')

## PDF reading
list_folder_file = os.listdir(path_to_sin_folders)
list_folder = [k for k in list_folder_file if os.path.isdir(path_to_sin_folders+'/'+k)]

start_time_all = time.time()
for _dir in list_folder:
    start_time_dir = time.time()
    logging.debug('Working on Folder {a}'.format(a=_dir))
    _new_folder_output = os.path.join(path_to_sin_folders_out, _dir)
    if os.path.isdir(_new_folder_output):
        logging.debug('The output folder {a} already exists. Files will be
overwrited'.format(a=_dir))
        pass
    else:
        os.mkdir(_new_folder_output)
        logging.debug('Output folder {a} created'.format(a=_dir))
    _get_files = os.listdir(path_to_sin_folders+'/'+_dir)
    _get_files = [k for k in _get_files if k.lower().endswith('.pdf')]
    _text_file = []
    _medatada_file = []
    for _file in _get_files:
        start_time_file = time.time()
        _path_to_file = path_to_sin_folders+'/'+_dir+'/'+_file
        logging.debug('Working on file {a}'.format(a=_file))
        doc = fitz.open(_path_to_file)
        metadata = pd.DataFrame(doc.metadata, index=[_file])
        metadata['page_num'] = len(doc)
        metadata['fileName'] = _file.split('.')[0]
        metadata['FolderName'] = _dir
        _medatada_file.append(metadata)
        __text_page = []
        for page in doc:
            logging.debug('Working on page {a} of the file {b}'.format(a=page.number,
b=_file))
            doc_name_list =_file.split('.')[:-1]
            doc_name= ("".join(doc_name_list))
            try:
                text = text_image_page_scanner(page, doc, doc_name, 50, 500, 500,
path_to_temp_data, image_preprocessing=True)
                text = '#### Complete extraction: -> ' + text + ' ####'
                logging.debug('Text extraction completed in file {a}, page 
{b}'.format(a=_file, b=page.number))
            except:
                logging.debug('Text extraction FAILED in file {a}, page {b}'.format(a=_file, 
b=page.number))
                text = ''
                text = '#### Empty extraction: -> ' + text + ' ####'
                continue
            __text_page.append(text)
        _text_file = '\n\n'.join([k for k in __text_page])
        logging.debug('Concatenation of all pages of file {a} completed'.format(a=_file))
        logging.debug('Execution on file {a} completed in {b} seconds'.format(a=_file, 
b=round((time.time() - start_time_file), 2)))
        with open(_new_folder_output+'/{a}.txt'.format(a=doc_name),'w',encoding='utf-8') as f:
            txt = txt_cleaner(_text_file, basic_operation=True)
            f.write(txt)
            f.close()
            logging.debug('{a}.txt file created'.format(a=_file))
    logging.debug('Execution on folder {a} completed in {b} seconds'.format(a=_dir, 
b=round((time.time() - start_time_dir), 2)))
    shutil.move(os.path.join(path_to_sin_folders, _dir), os.path.join(path_to_sin_move, _dir))
    logging.debug('Folder {a} moved to path {b}\n\n'.format(a=_dir, b=path_to_sin_move))
    df_metadata_all = pd.concat(_medatada_file, axis=0)
    df_metadata_all.to_csv(path_to_metadata_out+'/metadata_{a}.csv'.format(a=_dir))
logging.debug("Execution time all SIN: {a} seconds\n".format(a=round((time.time() - 
start_time_all), 2)))

endsubmit;
quit;
```