UPC Students on Data Science in Software Teams

TAED2 2021. Group 11 - 5
September - October 2021

TechLoan

Table of contents

Authors: Emili Bonet Cervera, Pol Ruiz Farré,Anna Patrícia Orteu Irurre,Marc Rovira Burgés

Abstract:

Nowadays, many teams have decided to create software in an agile manner. Despite the demonstrated benefits of this strategy, many of these teams are still not reaching their full potential. One reason for this is that, while these teams may be focused on flawlessly implementing their agile technique of choice, they overlook the human component of those approaches. We at TechLoan are aware of this issue and think that regardless of the methodology chosen, a Balanced Team is one of the most promising aspects to lead to the success of early-stage businesses or projects, as they provide varied perspectives, create synergy and are more accountable.

Therefore, the main goal of this project is to categorize developers by how good they are and the experience they have in solving errors. This classification can be used to better balance the development squads, build a tool that automates the process of creating balanced teams or to better assign the error solving tasks to the right developers. To achieve it we have to identify the developer expertise in terms of different factors such as solving faster Jira issues.

A data preparation section with its corresponding data selection, data construction, data cleaning and data integration has been carried out. Later on, we obtained our 3 clusters (unsupervised method) of developers by first transforming the data (with the UMAP methodology) and then applying a hierarchical agglomerative clustering model (complete), after seeing that it was the one that worked better.

The results show that it actually exists a clear distinction between the clusters, where the first cluster contains developers that contribute a lot in a highly reliable way, the second cluster contains committers, not such contributors but with a high-quality code and the third group contains beginners that contribute in a remarkable way to the projects but not in such a good manner. Having all this into consideration, we state that this project has achieved information that can be implemented in future projects (business objective).

Link to project record track: TechLoan - Project Record Track

5.1. Business understanding

5.1.1. Business objectives

- *Background*

Established in 2021, TechLoan is a team formed by four students of the Data Science and Engineering degree (GCED) program of the Polytechnic University of Catalonia (UPC).

It was created with the main objective of analyzing, finding and improving the Technical Debt Dataset (TDD) as a project that was given to us in the Software Engineering part of the Advanced Topics in Data Engineering 2 (TAED2) subject. As main information, we should highlight the fact that we did not know about the dataset itself at the beginning of the project, and this is a very interesting fact because the time needed to get familiarized with it should be taken into account.

- *Business goals*

Many teams have decided to create software in an agile manner. Despite the demonstrated benefits of this strategy, many of these teams are still not reaching their full potential. One reason for this is that, while these teams may be focused on implementing their agile technique of choice, they overlook the human component of those approaches. We are aware of this issue and think that regardless of the methodology chosen, a Balanced Team (junior members, very experienced programmers, error solvers programmers…) is one of the most promising aspects to lead to the success of early-stage businesses or projects, as they provide varied perspectives, create synergy and are more accountable.

Because of the reasons explained above, the main motivation for this project is to gain valuable insight into the usual management of software engineering projects to improve the performance of development teams based on the detection of recurrent flaws in the analyzed projects from the TDD.

Specifically, our goal is to categorize developers by how good they are or the experience they have in solving errors. This way, the insights could be used to better balance the development squads in terms of having developers with different expertise. Another use of these insights could be to better assign the error solving tasks to the right developers.

- *Success criteria*

The success of this project will be able to be tested regarding how many useful insights will be obtained from the relationships of the different tables. This gain of information will enable us to better understand and classify developers among different factors.

Our analysis will be satisfactory if development teams become correctly balanced depending on their skill sets with the aim to reduce the time needed to deliver a project. At the same time, reduce the tests and code reviewing times by coding high-quality software and by consequence, reduce bugs and provide a better user experience. Therefore, the insights gained from this research will be tested regarding their usability and effect in future projects. For instance, this could be measured by monitoring the performance and teamwork level.

It is important to note that our project is a proof of concept for private enterprises of an algorithm to automatically assign their developers into different projects of that enterprise.

5.1.2. Assessment of the current situation

- *Inventory of resources*

This section will provide a summary containing a list of different points that characterize the organization's current situation, including the available resources, the requirements, the assumptions and the constraints for this project. It will also contain the risks that can be foreseen and which contingencies will be put in place to address them. Also, an explained enumeration of the terminology used in this report and the costs and benefits expected from this project will be displayed in this section.

First of all, we will start by mentioning the personnel that it counts with:

- Anna Patrícia Orteu Irurre
- Emili Bonet Cervera
- Marc Rovira Burgés
- Pol Ruiz Farré

All of us are data mining experts currently undergraduates, but very close to obtaining a degree in Data Science and Engineering. Also, although we are not business experts, this is not the first time we have dealt with a project of this dimension, so we already have the technical and organisational skills to carry it out.

Next, we will continue by introducing the Dataset we are going to work with. This, as previously mentioned, is the **Technical Debt Dataset** (TDD) created by Valentina Lenarduzzi, Nyyti Saarimäki and Davide Taibi (Tampere University, Finland) in 2019. It is a curated set of project measurements data from 33 Java projects from the Apache Software Foundation (ASF), which is available through CSV files and an SQLite database, which facilitates queries on the data. So, not only it enables researchers to compare results on common projects but also analyses more technical issues rather than only code smells.

We have at our disposal multiple **small to medium computing devices** such as laptops and desktops, which are provided by the team members themselves. Apart from that, we also have access to the **High-Performance Computing servers from UPC** in case the need arose for greater computational output.

The software used for this project will be employed to manage the project, manage the documents and folders, and create and edit code. To manage the project, we will use **WhatsApp** as the means to communicate the members of the team with each other, and **Evernote** to keep track of the project's backlog and sprint log, as well as delivery dates. Documents will be managed differently depending on their function. The project's overall report (this document) and similars will be conducted through **Google Docs**, while the readme and script documents will be held in our **Github repository**. Code files will go through two stages: the initial designs and implementation of units and small components will be done with python notebooks in **Google Colab** given the ease to debug provided by such environments, while the integration and testing to the whole software will be done in Github. To edit code already uploaded to Github, local editors such as **Visual Studio Code** or **PyCharm** will be used.

- *Requirements, assumptions, and constraints*

If we first focus on the requirements:

- The project must achieve the success criteria previously defined.
- The insights gained must be developer-based, which means that the Dataset has to be analyzed thinking about how they develop the project's stages, software used and their conditions.
- The results and what they entail must be shown clearly as the relationships among all the different tables, without the necessity of too much analysis work afterwards.

To continue, if we change our attention to the assumptions:

- The team will be able to analyze the data with different software and programming languages because data is provided both in CSV files and SQLite format.
- The team will have access to the High-Performance Computing servers from UPC if greater computational power is needed (apart from using our computers).

Finally, if we move towards the constraint's topic:

- *Time:* the whole project should be completed in 7 weeks, being the presentations date the final deadline.
- *Data:* although this dataset is a huge improvement in the topic of research, it has some limitations; for instance, the creators of TDD have relied on the developers that must report the ID of the fault to the commits that fixed it to identify the fault-fixing commit.

- *Knowledge:* None of the members of the team is fully aware of the data that we dispose of at the beginning of the project.

- *Risks and contingencies*

There is a risk of not getting enough insights to classify in a meaningful way the developers. Therefore, the task of improving coding time and quality with insights from TDD is an ambitious goal that could not be achieved due to data information restrictions (lack of insights) or due to scheduling reasons, as the project could take longer than anticipated.

To mitigate these risks, it becomes important to perform a specified project plan and continuously review what has been done and what is still pending.

- *Terminology*

  - Business terminology:

    - TDD → Technical Debt Dataset
    - SQL → Structured Query Language
    - CSV → Comma-Separated Values
    - UPC → Polytechnic University of Catalonia
    - CRISP-DM → Cross-industry standard process for data mining
    - GCED → Degree in data science and engineering

  - Data mining terminology:

    - ETL → Extraction, transformation and load
    - IQR → Interquartile range
    - ID → Identifier
    - NA → Not Available
    - NaN → Not a Number
    - t-SNE → t-distributed stochastic neighbour embedding
    - UMAP → Uniform manifold approximation and projection
    - PCA → Principal Component Analysis

- *Costs and benefits*

The benefits of achieving the goal are highly remarkable. Nevertheless, there is a high investment of time and effort to achieve this goal by analyzing the dataset and gathering information. What is more, in the end, we can find that the goal was never achievable with the dataset. Having an ambitious goal is a double-edged sword. We take a high risk by defining goals that we can not know if could be achieved. In the subsequent lists, we enumerate the costs and benefits we foresee upon the completion of the project.

Costs	Benefits
• Servers: 10€/hour of execution* • Salaries: 1050€/month and person (minimum salary) • Time investment (avg. per week per contributor): 9h/week**	• Efficient management of developers • Better quality software • Suitable to a wide range of programming projects

* Cost is approximated
** Link to project specified record track: TechLoan - Project Record Track

5.1.3. Data mining goals and success criteria

- *Data Mining goals*

Identify the developer expertise in terms of different factors such as solving faster Jira issues. To achieve this goal, we will define how much time each developer has taken to solve errors from Jira issues, how many projects has worked on and how many commits has made in each project to further apply clustering models to identify the level of each developer.

- *Success criteria*

Develop a model that can classify developers by their expertise in 3 or 4 levels. Use these predictions to define which developer should work with another, for example, by grouping developers in squads of at least 3 or 4 groups based on the classification level. We plan to analyse the classification model in terms of its intra-class and inter-class variance together with some other metrics related to performance, efficiency and reliability.

5.1.4. Project plan

Initial assessment of tools and techniques

Our project will follow the standard data mining directives outlined in the document "CRISP-DM 1.0 Step-by-step data mining guide". To be able to accomplish the objectives stated in the business goals, we are going to enumerate different steps that must be carried out for the realization of this project, with the expected duration of each in brackets.

- **Business Understanding:** understand the objective of the project and how it must be carried out. For this stage, the paper "The Technical Debt Dataset" has been very useful. [1 week]

- **Data Understanding:** understand the data, how it has been collected, how it is correlated with and more insights on them. It will be mainly done through selects against the SQLite Database available and if necessary, tables and graphics obtained through python (Google Collaboratory) where now will also be necessary the CSV files. [1 week]

- **Data preparation:** it is one of the most difficult steps in any machine learning project. First of all, we will have to determine the problems or drawbacks of the TDD regarding our final objective. Next, a data cleaning stage will have to be carried out. This stage accounts, for instance, with the NA and outliers analysis (with the corresponding deletion, if necessary).

  To continue, a feature selection will be done with the main objectives of reducing the overfitting, improving the accuracy, reducing the test and training time and improving the performance. Then, an ETL process to correlate the tables we want to focus on. Finally, feature engineering (like imputation or grouping operations) and possibly a dimensionality reduction stage will also have to be carried out. The final objective of this stage is to obtain a fully functional database which will be the base (input) of our modelling and evaluation stages. [1.5 weeks]

- **Modelling:** During this stage, we will train different machine learning algorithms to predict different clusters from the features. More concretely, we will train our data with the k-means model and hierarchical agglomerative clustering (with the complete, single, average and ward linkage types). However, before this modelling stage, a data transformation will be carried out. Different methods will be tested like UMAP, PCA and t-SNE. [1.5 weeks]

- **Evaluation:** During this phase, we will analyze the results obtained with the model. It is necessary to mention the fact that depending on the results obtained at this point, we will have to come back to the modelling step. At this step, it is very important to focus not only on if the model trains well on data but also on if it satisfies the business need and can be deployed on live data. [1 week]

- **Deployment:** Release a fully operational program and submit it to a final stress test by giving it to try to anyone interested, expecting constructive feedback to smooth out user-related details. [1 week]

The previously mentioned High-Performance Computing servers may be necessary during the data preparation and modelling phases.

5.2. Data understanding

In this phase of the CRISP-DM process, we choose the data we need to obtain for our goals and verify that it is appropriate for our needs.

5.2.1. Initial data collection

First of all, regarding our data mining goal we have decided which tables of the TDD were useful (or could be) for our project. Those tables are git_commits, szz_fault_inducing_commits (each tuple is an issue from Jira), git_commits_changes, refactoring_miner (changes done in the structure to improve the quality and understanding of the code) and jira_issues. The data format CSV has been used.

We also must mention that not every column contained in every table has been taken. This is because we have considered that they were not useful for our end or because they were already contained in another table. And so, considering that later on, we will have to build our dataset, having the data once is enough. This fact has also been helpful considering that the CSV files we are working with are too big, and so we could not load them into Google Collab, the tool we use to develop the code in.

5.2.2. Data description

During this project we are going to use the data obtained from the next webpage:
The Technical Debt Dataset 2.0

As it can be seen, we are going to work with version 2.0 of it, which is a curated dataset containing measurement data from different tools (like SonarCube or Refactoring miner) executed on all commits to enable data scientists to work on a common set of data and thus compare their results.

Although this dataset is composed of 10 tables, it is worth mentioning that we are only going to work with 5 of them, because only those tables contain information about useful data regarding our business goal. These are the tables that have been previously mentioned in the 5.2.1 Initial data collection section.

In section 5.3.1 Data selection a more detailed version of which variable we have taken from each table and what they mean can be found, also, in section 5.2.3 Data exploration we can find an insight into each meaningful variable of each table. However, here, we are going to explain with which variables we are going to work with, their types, and how they can help with our final objective, to have an initial description of the data that we are going to work with:

- COMMIT_HASH [string of size 42 with letters and numbers interspersed]: Hash that identifies every commit (unique id for each commit). They will help us to relate

the different tables and to know how many commits each developer has done. An example of a commit_hash would be: *'52fc76012c5f969145c39d3fb398a7c2c094474f'*

- PROJECT_ID [string]: Id that identifies every project (unique id for each project). It will be useful to know how many projects each committee has participated in. An example of a project_id would be: *'org.apache:batik'*.
- COMMITTER [string → normal people names, identifying each committer]: Developer that commits. This is one of the most important variables because, as said before, we want to cluster the developers, and so this variable will end up being our "id". An example of committer would be: *'James Duncan Davidson'*.
- LINES_ADDED [integer]: Number of lines of code added to the file. It will help us to know how prolific a developer is and how many lines of code she/he modifies (adds) per commit on average.
- LINES_REMOVED [integer]: Number of lines of code removed from the file. It will help us to know how prolific a developer is and how many lines of code she/he modifies (removes) per commit on average.
- FILE [string]: File identifier. Example of file identifier: *'README'*.
- REFACTORING_TYPE [string]: Refactoring activity type applied in the commit. They will enable us to know which type of refactoring is more times done for a particular cluster of committers. An example of refactoring_type would be: *'Move Class'*.
- CREATION_DATE [date]: Date when Jira issue has been created.
- RESOLUTION_DATE [date]: Date when the Jira issue has been solved.
- FAULT_FIXING_COMMIT_HASH [string of size 42 with letters and numbers interspersed]: Hash of the commit where the Jira issue was solved. They always make reference to a commit_hash of the git_commits table.
- FAULT_INDUCING_COMMIT_HASH [string of size 42 with letters and numbers interspersed]: Hash of the commit where the Jira issue was introduced. They always make reference to a commit_hash of the git_commits table.

5.2.3. Data exploration

GIT_COMMITS: Table with 81072 rows. It contains information about a total of 31 projects (not 33 as said in the paper "The Technical Debt Dataset") and 482 committers. Also, regarding the committers column, we must say that 18 lines are tagged as "No Author". Furthermore, on the one hand, we have analyzed how many commit_hashes were per project. The minimum commits per project were 153 while the maxima were 15427 and the average 2615. On the other hand, we also have computed how many commit_hashes every committer has done. This time, the minimum number of commits was 1 while the maximum was 5627 and the mean value 169.

GIT_COMMITS_CHANGES: Table with 857740 rows. On average 39 lines of code are added within each commit and 25 removed. Nevertheless, we have to take into account that the maximum number of added rows is 110607 and the number of removed rows is 84395 probably due to refactoring some files.

REFACTORING_MINER: Table with 37226 rows. It contains a total of 29 refactoring_types. Regarding this factor, we have to take into account that the refactoring miner's release 1.0.0; which was the one used to create the dataset, can detect 15 different types of refactorings from different types of code elements. It is important to notice that the types 'Move And Rename Attribute' and 'Replace Attribute' only have 2 records each.

JIRA_ISSUES: Table with 2007 rows. On average it takes 201 days to fix each issue, being its variance 470 days. The maximum time spent to fix one error has been 5170 days while the minimum is 19 hours and 18 minutes. Regarding this fact, it can be seen that there are many outliers, that is why the average time that it takes to fix an issue is 9 days. Moreover, some records have a negative duration, therefore we have considered them wrong and removed them.

SZZ_FAULT_INDUCING_COMMITS: Table with 896 rows. If we multiply this number per 2, we get 1792 hashes, while in the git_commits tables we have 81072 rows (so 81071 hashes). This fact enables us to see that at most 2,21% of the commits introduce or remove some Jira issue, while the others just introduce new functionalities or, for instance, make some refactoring activity.

5.2.4. Data quality

Regarding the quality of the hashes (that reference the git_commits table) is perfect on almost all the tables; which is a key factor considering that later on, we will have to cross the tables.

The relevant quality issues for each table that must be addressed during the following phase are the following:

GIT_COMMITS: Regarding the committer column, 18 rows have "No Author" as a value. Taking into account that it is a very low number considering the total number of lines (0.0222%), we have deleted them. Moreover, we have also found 34 NaN in this very same column, which has been deleted for the same reason.

GIT_COMMITS_CHANGES and **SZZ_FAULT_INDUCING_COMMITS**: No relevant issues of quality for our project were found.

REFACTORING_MINER: Only row 25441 has a NaN as the commit_hash value, so that row is deleted.

JIRA_ISSUES: Creation_date and resolution_date variables, which are supposed to be indicating a date, are defined as strings instead of timestamp variables. Due to this fact, we have had to change them to be able to perform operations over these dates.

5.3. Data preparation

5.3.1. Data selection

Below, we can see the data (tables and variables inside each data, that we have decided to use for our project).

GIT_COMMITS:

- COMMIT_HASH: Hash that identifies every commit (unique id for each commit).
- PROJECT_ID: Id that identifies every project (unique id for each project).
- COMMITTER: Developer that commits.

Assumptions regarding git_commits: committer = author, and so, we have just stayed with the committer column.

GIT_COMMITS_CHANGES:

- COMMIT_HASH: References git_commits (commit_hash).
- LINES_ADDED: Number of lines of code added to the file.
- LINES_REMOVED: Number of lines of code removed from the file.
- FILE: File identifier.

REFACTORING_MINER:

- COMMIT_HASH: References git_commits (commit_hash).
- REFACTORING_TYPE: Refactoring activity type applied in the commit

JIRA_ISSUES:

- HASH: References git_commits (commit_hash).
- CREATION_DATE: Date when Jira issue has been created.
- RESOLUTION_DATE: Date when the Jira issue has been solved.

SZZ_FAULT_INDUCING_COMMITS:

- FAULT_FIXING_COMMIT_HASH: Hash of the commit where the Jira issue was solved.
- FAULT_INDUCING_COMMIT_HASH: Hash of the commit where the Jira issue was introduced.

We have decided to use these variables because we think that those are the ones that can give us more valuable information regarding the business objective that we want to achieve. For instance, the jira_issues can help us to know which are the developers that solve more issues and the velocity with which they achieve it. Or, on the other hand, the refactoring_miner table can give us information about the developers that give more importance to code quality, instead of the usability and functionality of the code.

Regarding the sonar_issues table, we have to mention the fact that we also considered taking that table information (together with the information of its associated tables like sonar_rules). However, the quality was not as good as the quality of the other issues nor the way how they were displayed was as good as the jira_issues for our project and so; we preferred to focus on the tables already mentioned.

5.3.2. Data cleaning

As we can see in the data quality section, after evaluating the tables, we have just had to eliminate rows from the git_commits and the refactoring_miner tables.

Regarding the first mentioned table, 18 rows have been removed because they had “No Author” as a value. Taking into account that it is a very low number considering the total number of lines (0.0222%), we have considered it was not a huge loss. On the other hand, regarding the refactoring_miner table, we have had to eliminate row 25441 because it has a NaN as the commit_hash value.

Analysing the lines modified we have seen some commits with a high number of lines added and removed. This happens due to some refactoring made. Although these commits are not strictly outliers, we have removed them in terms of not taking into account the lines modified don’t affect the other commits modifications that have a low number of lines modified. In order to don’t lose the data of the refactorings made, we will study a way of taking into account the refactorings in the final dataset.

In a similar way to what we mentioned before, we have found a similar behaviour when we have computed the duration of the different Jira issues. Having a maximum duration of more than 5000 days can bring us problems as the median is around 9 days. Regarding this variable, we have also found 5 records with negative durations. These entries have been eliminated because the duration did not make sense. Also, we removed all those lines with a value higher or lower than the quartiles +/- 5*IQR (5 instead of the normal 1.5 because if not, we were going to delete 18% of the lines, and we thought that was too much). 2600 lines fulfilled this fact, and as they were supposed to be 11,82% of the lines of this table, we know that this is a rather high number, however, we also removed those lines to avoid possible inconsistencies while building the model.

We have continued with the detection of outliers proceeding with the same process over the lines_added and the lines_removed variables of the git_commit_changes. This time, 16% of the lines were removed, so 137265 tuples considering both of them. More concretely, 66288 tuples did not fulfil the condition in the lines_added variable (7,72 %) while 84933 did not fulfil it in the lines_removed variable (9,9%). This time, if we had considered the normal 1.5 value in the confidence interval instead of 5, 20,8% of the lines would have been eliminated.

5.3.3. Data construction

During this task, we need to derive some new fields or aggregated data.

Therefore, before integrating all the data, we have decided to compute the duration to solve each Jira issue. This will enable us to (1) find outliers regarding those durations, (2) see how much time developers take to solve a specific type of Jira issue on average (using CREATION_DATE and RESOLUTION_DATE from JIRA_ISSUES table) and (3) compute the average duration that a developer takes to solve the errors. This step has been done before the integration because we only needed the jira_issue table to compute the values, and so the computation is more computationally efficient.

Regarding this table, to later create the final dataset, we have aggregated the jira_issues by commit_hash and computed the average duration and the number of Jira issues solved in each commit as one commit is expected to solve one or more Jira issues.

To continue, we have done a similar aggregation regarding the git_commit_changes where the average of lines added and removed and the files changed was computed. Also, similar action has been done regarding the refactoring_miner table, but this time, we have computed the number of refactorings. Finally, regarding the szz_fault_inducing_commit, we have computed how many issues have been solved in each commit and how many issues have been introduced.

5.3.4. Data integration

Currently, our data is in five disparate datasets. Now we need to merge all those disparate datasets into one big dataset to get ready for the modelling phase. This will mainly be done by performing left joins over the hash_commit variable of each table (so that we do not lose the information of any commit).

As in the previous section we have already done the groupings over the commit_hash in each table, this step is rather easy with the already mentioned joins. However, once this operation was performed, we noticed that a lot of Nan’s were introduced. So, a cleaning step to remove all those Nan’s introduced was once again applied.

Finally, once we have done the integration of the different tables, as we aim to analyse developers, we have had to group by committer. This way we obtain a table with a different committer (developer) in each row. During this step, we have changed the columns project_id by the number of projects in which each developer has participated and commit_hash by the number of commits that the developer has done. We also computed the average of Jira issues solved per commit, the average duration spent solving each issue and the average number of files modified. The resulting columns are the values grouped using the average, count and sum operations.

5.3.5. Dataset description

Once the integration is done, we have our dataset, which contains the relevant information for our project and is ready to be used. The dataset contains the following information:

- COMMITTER: ID variable of our dataset, all the other variables of the dataset give specific information of committers
- COMMIT_HASH: The number of commits made
- PROJECT_ID: The number of projects that a developer has worked on.
- COUNT_SOLVED: Average Jira issues solved per commit.
- AVG_DURATION: Average duration solving a Jira issue.
- FILES_CHANGED: Average files modified per commit.
- AVG_LINES_ADDED: Average lines of code added.
- AVG_LINES_REMOVED: Average lines of code removed.
- FIXED_ISSUES: Average fixed issues.
- INDUCED_ISSUES: Average induced issues.
- COUNT_REFACTS: Average refactorings made.

The dataset has been created keeping in mind that all those variables will provide us with useful information to later divide developers into different sets.

5.4. Modeling

5.4.1. Modeling assumptions

During this project as said before, we are going to classify committers into 3-4 groups. To do it, we are going to use **clustering techniques**. However, we have to bear in mind that there does not exist a ground truth and so, it will be **unsupervised learning**. To this end, we are going to use the next clustering techniques taking into account the following assumptions:

- **K-means** → Assumptions: k-means assumes that the variance of the distribution of each attribute is spherical, that all the variables have the same variance and that the prior probability for all k clusters are the same, i.e. each cluster has a roughly equal number of observations.

- **Hierarchical agglomerative clustering** → Assumptions: all the variables must be numeric (because if not, it is quite difficult to compute a distance matrix); however, it has not been a huge deal for us, because we designed our dataset this way, without being conscious of these restrictions, since the beginning. No NA can be found because most of the agglomerative clustering models can not deal with this fact. This has not been a problem due to the data cleaning task previously performed.

5.4.2. Test design

As we are not in a supervised data mining task, but in an unsupervised classification task, we have not had to divide the dataset into train and test. Nevertheless, we continued to evaluate the different clusters made by the models to identify the clearest patterns.

In order to test the models' quality, as the ground truth labels are not known, we have made use of the Pseudo-F statistic to evaluate the resulting cluster quality; a ratio of the sum of between-clusters dispersion and of within-cluster dispersion for all clusters (being the dispersion defined as the sum of distances squared). If this score is greater, it implies the clusters formed are dense and well separated, which corresponds to the traditional definition of a cluster.

However, we have to bear in mind that in a real-world application (like the one we are dealing with) the data is typically in high dimensions and cannot be visualized on a plot like the ones we are going to use, which means that poor solutions may be found without being obvious that they are poor. Furthermore, dendrograms (diagrams representing a tree), generally used to select the number of clusters, are normally misinterpreted.

5.4.3. Model description

Before applying the models, we have to take into account that we have first transformed our data to a lower-dimensional space. First, we tried to do it with PCA (Principal Component Analysis); a process that computes the principal components and uses them to perform a change of basis on the data, sometimes using only the first few principal components (the ones that explain the majority of the variance) and ignoring the rest.

However, as the results obtained were very poor, we changed and did it with UMAP (Uniform manifold approximation and projection) that creates a high-dimensional graph representation of the data before optimizing a low-dimensional graph to be structurally comparable and preserve as much as possible the data's global structure. This technique is based on Riemannian geometry and algebraic topology.

Next, we tried with t-SNE (t-distributed stochastic neighbour embedding), which is a statistical nonlinear approach for dimensionality reduction widely used to display high-dimensional data by assigning a two- or three-dimensional map to each data point. It uses Stochastic Neighbour Embedding as its foundation and represents each dimensional point in such a way that related things are modelled by adjacent points with high probability, while different objects are modelled by distant points.

The UMAP technique is comparable to t-SNE, and it potentially maintains more of the global structure while providing better run time speed. Furthermore, as UMAP has no computational constraints on embedding dimensions, it may be used as a general-purpose dimension reduction approach in machine learning.

Next, we will precisely describe the methods and variants of them that we have used to compute the clusters:

Hierarchical agglomerative clustering: method of cluster analysis which seeks to build a hierarchy of clusters. More concretely, the agglomerative approach is a "bottom-up" approach: each observation starts in its cluster, and pairs of clusters are merged as one moves up the hierarchy. The merges are determined greedily and the results are usually presented in a dendrogram. The distance metric used has been the Euclidean distance.

Inside this method, several sub-methods can be identified:

- **Complete-linkage clustering:** clusters are sequentially combined into larger clusters using the maximum distances between all observations of the two sets. The method is also known as **maximum-linkage clustering**.
- **Single-linkage clustering:** At each step, the algorithm combines two clusters that contain the closest pair of elements not yet belonging to the same cluster as each other. However, as a drawback of this method, we see that it tends to produce long thin clusters in which nearby elements of the same cluster have small distances, while elements at opposite sides of a cluster may be much farther from each other than two elements of other clusters.

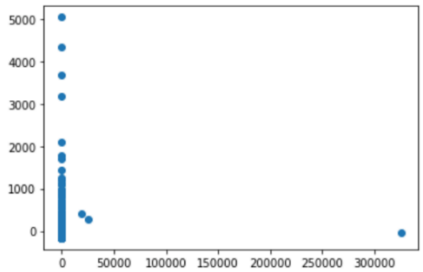
- **Average-linkage clustering:** is a method where the two nearest clusters (i and j) are combined into a higher-level cluster. This means that its distance to another cluster k is the arithmetic mean of the average distance between members of k - i and k - j.
- **Ward-linkage clustering:** agglomerative hierarchical clustering procedure, where the criterion is to minimize the variance of the clusters being merged.

k-means (linear model): is a vector quantization approach that tries to split n observations into k clusters, in which each observation belongs to the cluster with the closest mean (cluster centres or centroid), which acts as the cluster's prototype. These results minimize the within-cluster variances (squared Euclidean distances), but not regular Euclidean distances. This method is highly used in data mining.

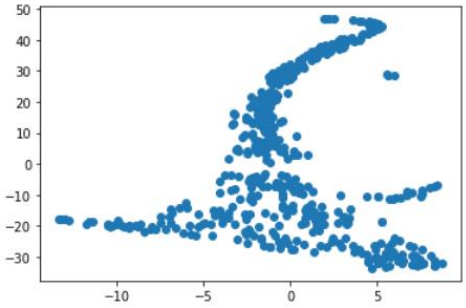
In our case, we will divide the data into 3 and 4 clusters because fewer clusters would reduce the problem too much and if more clusters were considered, they will be very similar between them, and extracting differences would be very difficult. We know this fact, due to previous data analysis performed.

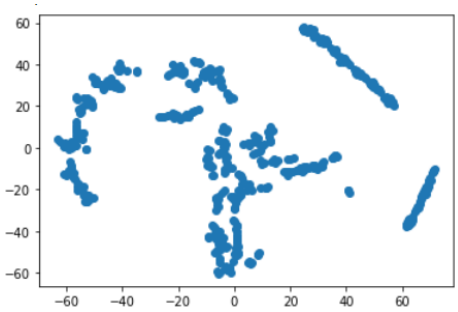
5.4.4. Model assessment

During the transformation of the data different types of dimensionality reduction techniques were carried out, more concretely, we tried PCA, t-SNE and UMAP. However, PCA was discarded because once the reduction of dimensionality was applied it was not possible to correctly separate the data. Next, we can see an example of it:

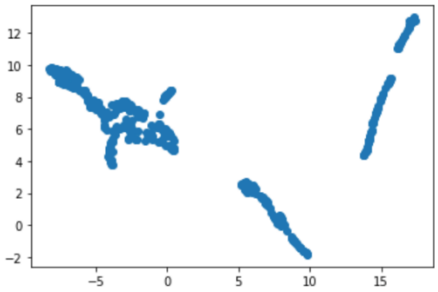


We also discarded the t-SNE method for similar reasons, because although its separation was better in comparison with the PCA transformation, the results were also poor if we compare them with the ones obtained with the UMAP method. Below we can see the plot obtained when trying to create the clusters with t-SNE (the left plot is by directly applying the technique, while the right photo is once several parameters were modified trying to obtain some visible clusters):





So, in the end, we end up using the UMAP transformation due to its good operation, even more, if we compare the results obtained with the previous ones. As stated in section 5.4.3 Model description, from the theoretical point of view, it was predictable that this method was going to be the best, obtaining the 3-4 clusters predicted. Next, we can see the result obtained with this technique:



It can be seen that after applying the UMAP transformation, we have already obtained a high-quality clusterization, which will enable us with later cluster models to perfectly discriminate between different types of developers.

To continue, as said in section 5.4.2 Test design, as the ground truth labels are not known we made use of the pseudo-F statistic to compare and evaluate the different models, where a higher score relates to a model with better-defined clusters.

We evaluated the different models and different values for the parameter of the number of clusters desired, and the results obtained for each of them were:

Pseudo-F statistic					
Cluster Model (parameters)	Hierarchical agglomerative				K-Means
	Complete	Average	Ward	Single	
K=2	839.5957	839.5957	839.5957	458.2869	839.5957
K=3	1232.8371	1232.8371	1232.8371	1232.8371	1232.8371
K=4	1482.7507	1482.7507	1482.7507	997.1103	1490.3936

It can be seen that different Hierarchical agglomerative sub-models obtained the same pseudo-F statistic except for Single linkage that had quite poor results in the case of 2 and 4 clusters. Regarding the K-Means model, it always obtained the same results as Hierarchical agglomerative models unless in the case of 4 clusters, where it performed even slightly better.

Therefore, taking into account these results and knowing that the most natural way would be to separate developers into 3 or 4 groups, this had been the number of clusters (K) chosen. Choosing the model hierarchical agglomerative (complete) for the three clusters option and the k-means for the 4 clusters one (both of them, the best options). So in the next section, 5.5 Evaluation, the best models with three and four clusters will be rigorously evaluated.

5.5. Evaluation

5.5.1. Assessment of data mining results with respect to business success criteria

During this stage, we will analyze the characteristics of each cluster obtained (both when creating 3 and 4 clusters with the models) and once we have summarized this information, we will be able to say if the model meets the business objectives or if there is any reason why the model chosen is deficient.

First of all, we will start by analyzing the approved model resulting in **three clusters (hierarchical agglomerative (complete))**. Focusing on the number of projects where each developer has participated in, we can state that cluster 0 contains those with the highest contribution with a mean of 4.12 and a maximum of projects participated per developer of 22, while these factors decrease to 1.25 and a maximum of 10 projects participated in the 1 cluster (the ones with less participation); and till 2.53 and a maximum of 20 projects participated per developer in the 2 clusters. Changing now to the commits made, a similar distribution is shown, although now the means are much more different with a value of 676.34, 11 and 78.08 with respect to the 0, 1 and 2 clusters.

This initial analysis allows us to say that the model is not deficient, as it correctly clusters the developers by how much they have participated in the different projects, which could be directly correlated with the experience they have.

However, we can obtain even more information on the elements that distinguish each class by moving on to analyzing the number of refactorings that each developer has solved. Explain before analyzing that code refactoring is the process of restructuring existing computer code without changing its external behaviour. Refactoring is intended to improve the design, structure, and/or implementation of the software while preserving its functionality. So, it can be directly correlated with how good a developer is and how much effort he/she puts into obtaining a good quality code rather than only functional code. Here, the means obtained for each cluster (0,1,2) = (0.37, 0.27, 0.21). While the maximums are (1.68, 14.61.94), as always in the same order of clusters 0, 1 and 2.

Thanks to this analysis, we can state that although group number 1 seemed to have less experience, they are fewer contributors to the projects but when they contribute they do it in a quality improvement way, which enables us to say that they are probably better developers than those that can be found in cluster 2.

This analysis can be even reinforced by looking into the issues that each commit has introduced, although not in a very reliable way as the percentiles of this variable are mostly 0 and so, not very confident. This time, the means of induced commits are: (0.0038, 0.0021, 0.0077). This means that the second group is completely the less experienced and the one that contains the worst programmers, while the zero and first group are less equally good quality talking, but that the zero one has introduced more errors, which is a pretty normal thing taking into consideration that they have contributed much more.

Finally, if we now focus on the average of lines removed and added by each committer, we see that the means and quartiles are very equal, distinguishing themselves only by the maximums and the standard deviation. That is why we state that these variables do not provide new relevant information to help us to differentiate the clusters.

Before concluding this 3 clusters analysis, say that all those means and maximums have been supported by the percentiles and the standard deviation values. And that the tables from where we have extracted this information can be found in the data_exploration.ipynb notebook.

Having all those facts into consideration we state that cluster number 0 is the one that does the main code of the projects, introducing some mistakes like every human but in general it is good quality code. While we stamp to the first cluster the tag of error solvers (working just when necessary but doing it at a high-quality level). And to the second group the tag of beginners, which to gain practice they have to participate in as many projects as possible but their code should be revised by someone to avoid mistakes, and so, wasting later time to solve them.

Next, a similar analysis but with the **k-means model with 4 clusters** is going to be carried out:

Related to the approved model with 4 clusters, we can perform a similar pattern to the other model. The main differences remain in the number of commits and the projects that a developer has worked on.

We can clearly identify the 0 cluster as the group of developers that has contributed less. The mean number of commits and projects per developer is 2.2 and 1.1, KPIs that indicate a low level of contribution. We can also analyze that in each commit, in general, the number of files changed is between 1 and 2. Along with the average number of lines removed (2.7) and added (5.2) per commit, these metrics represent the lowest values compared to the other clusters. So, we can say that the developers of k=0 are the ones with the lowest contribution and, in terms of each commit, we find that these are the commits with the lowest level of modifications (no refactors, low number of files and lines modified and no solved issues).

For the developers clustered in 1, we can see that the average number of projects has increased respectively from cluster 0 (1.1 to 2.5) as well as the number of commits (from 2.2 to 78). This can clearly indicate that the developers of this group have remarkably contributed more. In this cluster, we can see that in terms of the commits made, the developers modify (on average) more files (5.9) and lines (12 added and 5.9 removed). We can also see that in this cluster developers do more refactorings (from 0.001 to 0.22). This can indicate an increased level of seniority and contribution in the projects respectively from the ones that don't contribute at all.

Related to the developers clustered in 2, we find the developers with the highest level of contribution. The average number of commits per developer is 676.4 and the average number of projects that a developer has worked on is 4.1. We can also see that the other

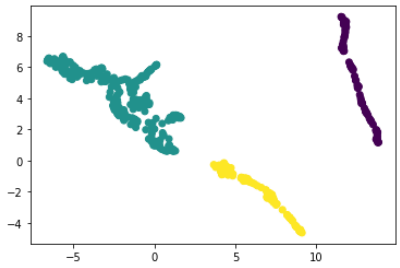
metrics don’t change a lot compared to cluster 1 (the same average of lines added, removed and files modified). These metrics indicate a high level of seniority and contribution to the projects, with a remarkably increased number of commits per developer compared to all the other clusters.

In the last cluster (3), we can clearly see a decrease in the number of projects and commits made by each developer (1.3 and 15.28). Although we can see a much higher number of files changed per commit (9) and lines added (17) we can define this group as the developers with a low level of contribution and seniority.

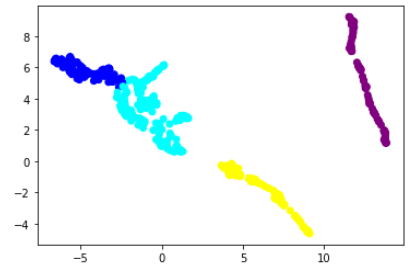
To sum up, in this model we have two clusters (0 and 3) with the developers that contribute less in the different projects, we have a cluster of medium contributors (1) and we have a cluster with high level of contributors (2).

With these conclusions, we can say that the model that clusters the developers in 4 groups takes us to the same conclusions that the model with 3 clusters: We find 3 highlighted levels of contribution and expertise. So we can say that **the model of 4 clusters doesn't make sense**. Next, the results of both models are visually shown:

Hierarchical (complete linkage), K=3



K-Means, K=4



Finally, although it was not our initial objective, we decided to create a script named auto-squading.py which assigns the developers available in the database into different balanced teams depending on the number of developers that have been assigned in each cluster during the modelling stage. In the model that we have obtained, we have a cluster that is much bigger than the other two (the one of the experienced developers). That is why if, for instance, we want to obtain teams of 5 developers, each team will have 1 senior developer (to lead the project), 1 newbie developer (so that he/she can gain experience) and 3 experienced ones so that they can correctly develop the project the fastest and with the best possible quality. And actually, the distribution obtained (1,1,3) makes a lot of sense. Below, one can see some examples of it (teams generated with three, four or five developers each, from left to right).

```
Squad #1:
    senior:
        Christopher Oliver
    experienced:
        Dean Jackson
        Michael Melhem
        Javier Puerto
    newbie:
        Kristian Rosenvold
Squad #2:
    senior:
        Carsten Ziegeler
    experienced:
        Prasanth J
        Duncan Jones
        Erik Abele
    newbie:
        Michi Mutsuzaki
```

```
Squad #1:
    senior:
        Thierry Kormann
    experienced:
        Simone Gianni
        Helder Miguel Alves Magalhães
    newbie:
        Christian Grobmeier
Squad #2:
    senior:
        Ralph Goers
    experienced:
        Sam Berlin
        Andrew Williams
    newbie:
        Kristian Rosenvold
```

```
Squad #1:
    senior:
        Henri Biestro
    experienced:
        Sean Mullan
    newbie:
        Olivier Lamy
Squad #2:
    senior:
        Niall Kegan Pemberton
    experienced:
        Christopher Piro
    newbie:
        Marc Slemko
```

5.5.2. Review of process

During this section, we are going to enumerate elements that could have improved the way or the velocity with which this project was completed. First of all, we have to say that thanks to always following the CRISP-DM guidelines no step of the project was overlooked. However, after analyzing the data, we should say that some of the variables that were taken into account, in the end they did not contribute to the discrimination between the different clusters, so they just take us time. One example of it could be the duration that a committer took to create a commit, a variable that was constructed by us during the data construction step. A more important fact that we have seen after analyzing the data is that building a model with 4 classes makes no sense at all, although of course, the analysis had to be done before being able to state that conclusion.

On the other hand, even though we strongly recommend the creation of a model to correctly classify developers into the different clusters, the most important part of the modeling step is the data transformation and dimensionality reduction (with UMAP). As we could see in the plot, it delimits pretty well the three clusters, making it quite easy to select the number of clusters and, for the unsupervised models, to determine each cluster (obtaining practically no difference between the algorithms). That is why we truly believe that although the creation of the models has been useful in order to be able to analyze them, the transformation of the data would possibly be enough if the final objective is to divide the developers into 3 groups to later use them to create squads.

Finally, mention that the fact of not following the practices of cookiecutter from the beginning has entangled us a bit towards the end of the project where the goal was not only to reach objective conclusions but also to obtain code files that could be reused.

5.5.3. List of possible actions

Now that we know more about the data that we are dealing with, we are going to list possible actions that could be performed if this project was lengthened, or continued at some other time, by ourselves, TechLoan, or some other data analysts' group:

1. Eliminate not explicative variables (from the data selection and data construction stages)
2. Introduce some other variables initially though not interesting but that could improve clusters differentiation
3. Introduce also information about other tables in the TDD
4. Implement more good practices to improve overall project quality
5. Try more transformations over the data
6. Try more models
7. Delve into how to create squads

Say, that as this project has not been financed or neither like this, the list does not take into consideration the remaining budget nor resources.

5.6. Deployment

5.6.1. Deployment plan

This stage aims at determining a strategy for deployment, documenting it for later development. It will contain the steps required and the instructions for carrying them out.

First of all, we will need an organization to introduce new information into the dataset. This information from the developers to add, must have the same data and follow the same structure as the TDD dataset.

If we know that a new developer has been introduced, we should directly apply the process described below, to assign him/her into the cluster to know to which projects or error tasks resolution is better to be assigned. Otherwise, we could wait for at least 3 or 4 introductions of new data, because if not, the clusters obtained would be very similar.

Secondly, a new extraction of the data from the database will be carried out. After this extraction has been done, the CSV's obtained needs to be passed to the data_preparation.py script (which will select the desired columns of the dataset, clean the data and perform the integration; between others). Then the data obtained will have to be passed to the commit_to_commiter.py script to obtain the dataset with which we will work during the modelling stage. This will have the next variables:

- COMMITTER: ID variable of our dataset
- COMMIT_HASH: The number of commits made
- PROJECT_ID: The number of projects that a developer has worked on.
- COUNT_SOLVED: Average Jira issues solved per commit.
- AVG_DURATION: Average duration solving a Jira issue.
- FILES_CHANGED: Average files modified per commit.
- AVG_LINES_ADDED: Average lines of code added.
- AVG_LINES_REMOVED: Average lines of code removed.
- FIXED_ISSUES: Average fixed issues.
- INDUCED_ISSUES: Average induced issues.
- COUNT_REFACTS: Average refactorings made.

This new dataset obtained, will have to be sent into the modelling.py script, which contains the data transformation with UMAP and the creation of the hierarchical agglomerative (complete) model, to create the new clusters (with more data than the last execution) and have the new developers (if exist) classified. Once we have arrived at this stage, we will have the new developers classified (if they exist) and more reliable statistical information about the distinctions between the different clusters.

5.6.2. Monitoring and maintenance plan

The careful preparation of a maintenance strategy helps to avoid unnecessarily long periods of incorrect usage of data mining results and further problems. That is the main reason why we have created a summary of monitoring and maintenance strategy, including the necessary steps and how to perform them.

Here the maintenance aims at introducing into the correct clusters new developers introduced in the TDD, and at the same time, try to obtain more insights that help to obtain better clusters (more distinctive clusters every time).

In section 5.6.1 Deployment plan, we have defined the main steps to obtain those clusters after a new introduction of developers data, however, we strongly recommend performing a more deep analysis at least once a year to see if the new developers added, have been correctly classified and if, in general, the statistical significance of the clusters obtained continue to be the same as in the previous execution. This should be done using the modelling.py script to create a new model, and analyse the results with the data_exploration.ipynb notebook provided. If this is not the case, it is possible that the transformation of the data and the choice of the model need to be changed, and adapted to due to the presence of new data with a different statistical distribution.

The most likely is that it will be the model that has to be changed, as the UMAP transformation works in general very well, due to its complexity. Great importance must be given to the different parameters used throughout the modeling stage.

Also, if more variables are introduced in the original dataset or more KPIs are taught that can add value to the model, the data_prepataion.py and commit_to_commiter.py scripts will also need to be modified in order to introduce them.

5.7. Final executive summary

- **Summary of business understanding: background, objectives, and success criteria**

Background

Techloan is a team formed by four students of the GCED program of the UPC created with the main goal of analyzing, finding and improving the TDD as a project given to us in TAED2.

Business goals

Many teams have decided to create software in an agile manner. Despite the demonstrated benefits of this strategy, many of these teams are still not reaching their full potential. We are aware of this issue and think that regardless of the methodology chosen, a Balanced Team (junior members, very experienced programmers, error solvers programmers…) is one of the most promising aspects to lead to the success of early-stage businesses or projects.

The previously mentioned fact is the reason why our business goal is to categorize developers by how good they are and the experience they have in solving errors. The insights obtained could be used to better balance the development squads in terms of having developers with different expertise or to better assign the error solving tasks to the most appropriate ones.

Success criteria

The analysis will be satisfactory if development teams become correctly balanced depending on their skill sets to reduce the time needed to deliver a project. Therefore, the insights gained from this research will be tested regarding their usability and effect in future projects. For instance, this could be measured by monitoring the performance and teamwork level.

- **Summary of data mining process**

Data Mining goals

Identify the developer expertise in terms of different factors such as solving faster Jira issues. To achieve this goal, we will define how much time each developer has taken to solve errors from Jira issues, how many projects has worked on and how many commits

has made in each project to further apply clustering models to identify the level of each developer.

Data preparation

This section is formed by different parts: the first of them is data selection, where we have selected which tables (and variables of them) of the original data set were going to be useful for our project. We ended up selecting the next tables: GIT_COMMITS, GIT_COMMITS_CHANGES, REFACTORING_MINER, JIRA_ISSUES and SZZ_FAULT_INDUCING_COMMITS.

We have decided to use the variables of these tables because we believe that those are the ones that can give us more valuable information regarding the business objective that we want to achieve. For instance, the jira_issues can help us to know which are the developers that solve more issues and the velocity with which they achieve it. Or, on the other hand, the refactoring_miner table can give us information about the developers that give more importance to code quality, instead of the usability and functionality of the code.

To continue, we have cleaned the data, so that to avoid problems when creating the model. We have eliminated rows breaking some restrictions like having “No Author” in the committer variable or having too rare numbers (outliers) like having several lines removed unexpectedly (too high).

Next, in the section of data construction, we derived some new fields or aggregated data. For instance, we have decided to compute the duration to solve each Jira issue. This enabled us to find outliers regarding those durations and to compute the average duration that a developer takes to solve the errors.

Next, we have integrated all the tables (data integration section) to create one big dataset to get ready for the modelling phase. This has been achieved by performing left joins over the hash_commit variable of each table (so as not to lose the information of any commit). In this step, we have had some problems, because once this operation was performed we noticed that a lot of Nan’s were introduced. So, a cleaning step to remove all those Nan’s introduced has been applied.

Finally, once we have done the integration of the different tables, as we aimed to analyse developers, we have had to group them by committer. This way we obtained a table with a different committer (developer) in each row. During this step, average, count and sum operations have been carried out to aggregate the rows referring to the same committer.

So in the end, the dataset contains the following information:

- COMMITTER: ID variable of our dataset, all the other variables of the dataset give specific information of committers
- COMMIT_HASH: The number of commits made
- PROJECT_ID: The number of projects that a developer has worked on.
- COUNT_SOLVED: Average Jira issues solved per commit.
- AVG_DURATION: Average duration solving a Jira issue.
- FILES_CHANGED: Average files modified per commit.
- AVG_LINES_ADDED: Average lines added.
- AVG_LINES_REMOVED: Average lines removed.
- FIXED_ISSUES: Average fixed issues.
- INDUCED_ISSUES: Average induced issues.
- COUNT_REFACTS: Average refactoring made.

Modeling

Before developing the different models, several transformations were carried out to map our data in a lower-dimensional space. We have tried PCA, t-SNE and UMAP.

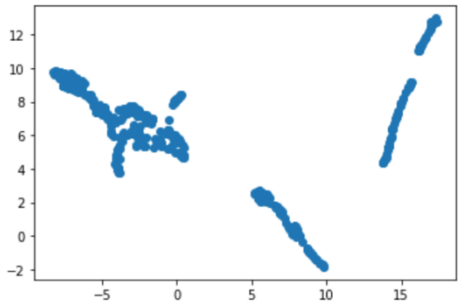
Then, we started trying different models to cluster our data with unsupervised learning. We have tried k-means (linear model), which is a vector quantization approach that tries to split n observations into k clusters, in which each observation belongs to the cluster with the closest mean, which acts as the cluster's prototype. And that assumes that the variance of the distribution of each attribute is spherical, that all the variables have the same variance and that the prior probability for all k clusters is the same.

Secondly, we have also tried hierarchical agglomerative clustering which is a method of cluster analysis that seeks to build a hierarchy of clusters. More concretely, the agglomerative approach is a "bottom-up" approach. Inside this method, several sub-methods were tried: complete-linkage, single-linkage, average-linkage and ward-linkage. These models assume that all the variables must be numeric and that they do not have NA's.

To test their quality, as the ground truth labels are not known, we made use of the Pseudo-F statistic to evaluate the resulting cluster quality in each model; a ratio of the sum of between-clusters dispersion and within-cluster dispersion for all clusters

- **Summary of data mining results**

During the transformation of the data, as said before, different types of dimensionality reduction techniques have been carried out. However, PCA was has been discarded because once the reduction of dimensionality was applied it was not possible to correctly separate the data. We also have discarded the t-SNE method for similar reasons. So, in the end, we have finished using the UMAP transformation due to its good performance. Next, we can see the results obtained with this technique:



To continue, we have made use of the pseudo-F statistic to compare and evaluate the different models, where a higher score relates to a model with better-defined clusters. We have seen that the Hierarchical agglomerative models obtained the same pseudo-F statistic except for Single linkage that had quite poor results in the case of two and four clusters. Regarding the K-Means model, it has always obtained the same results as Hierarchical agglomerative models but in the case of four clusters, it performed even slightly better:

Pseudo-F statistic					
Cluster Model (parameters)	Hierarchical agglomerative				K-Means
	Complete	Average	Ward	Single	
K=2	839.5957	839.5957	839.5957	458.2869	839.5957
K=3	1232.8371	1232.8371	1232.8371	1232.8371	1232.8371
K=4	1482.7507	1482.7507	1482.7507	997.1103	1490.3936

Therefore, taking into account the results and knowing that the most natural way would be to separate developers into three or four groups, the number of clusters (k) chosen to analyse has been 3 and 4, completely discarding the k = 2 approach. We have chosen the hierarchical agglomerative (complete) model for the three clusters option and the k-means for the 4 clusters one.

However, we have to take into account that the UMAP transformation has already obtained a high-quality clusterization, which implies that the model's discriminate task is mostly done in the data transformation stage.

- **Summary of results evaluation**

During this stage, we have analyzed the characteristics of each cluster obtained (both when creating 3 and 4 clusters with the models). The analysis has been carried out mainly with the means and maximums of each variable although it was also supported with the percentiles and the standard deviation values.

First of all, we are going to focus on the three clusters (hierarchical agglomerative (complete)) model. Regarding the number of projects where each developer had participated, we have stated that cluster 0 contains those with the highest contribution, while these factors decrease in the 1 cluster and 2 clusters. A similar distribution has been observed regarding the commits done.

More information distinguishing each class has been obtained by analyzing the number of refactorings that each developer solved. Here, the means obtained for each cluster were (0.37, 0.27, 0.21). This analysis was reinforced by looking into the issues that each commit had introduced, this time, the means of induced commits were: (0.0038, 0.0021, 0.0077).

Finally, we have focused on the average of lines removed and added by each committer, and we have found that these variables did not provide new relevant information to help us to differentiate the clusters.

Next, a similar analysis but with the 4 clusters obtained with the k-means model has been carried out. Here the main differences once again remained in the number of commits and the projects that a developer had worked on.

We have seen that developers contained in the clusters 0 and 3 were the ones that contributed less in the projects (although a higher number of files changed per commit and lines added was observed). On the other hand, the 1 cluster contained the medium contributors. It was also regarded that in terms of the commits made, these developers modified more files and lines (added and removed) and that they did more refactorings.

Finally, the 2 cluster contains the developers with more contributions (projects and commits), while the other metrics do not change a lot compared to cluster 1. With these conclusions, we have seen that the model that clustered the developers in 4 groups take us to the same conclusions that the model with 3 clusters, so it made non-sense creating it.

- **Conclusions for the business**

Once we have summarized the information found in the result evaluation, we have concluded that the models meet the business objectives.

Regarding the model with three clusters: we have stated that cluster number 0 is the one that develops the main code of the projects, introducing some mistakes like every human but in general, it achieves good quality code. While we stamped to the first cluster the tag of error solvers (working just when necessary but doing it at a high-quality level). And to the second group the tag of beginners, which to gain practice they have to participate in as many projects as possible but their code should be revised by someone to avoid mistakes.

On the other hand, regarding the model with 4 clusters (although not as discriminative as said before), we have seen that developers of k=0 were the ones with the lowest contribution, while the developers contained in the 1 cluster had remarkably contributed more, which indicates an increased level of seniority. Thirdly, regarding the committers contained in the 2 clusters, they have a high level of seniority and contribution to the projects, with a remarkably increased number of commits per developer compared to all the other clusters. Finally, we have defined cluster number 3 as the developers with a lower level of contribution and seniority (very similar to the 0 cluster).

- **Conclusions for future data mining**

Like every project, now that we know more about the data we have been analyzing these past weeks, we are going to list a few possible actions that could be implemented to improve the project:

1. Eliminate not explicative variables (from the data selection and data construction stages)
2. Introduce some other variables initially though not interesting but that could improve clusters differentiation
3. Introduce also information about other tables in the TDD
4. Implement more good practices to improve overall project quality
5. Try more transformations over the data
6. Try more models
7. Delve into how to create squads

5.8. Good engineering practices

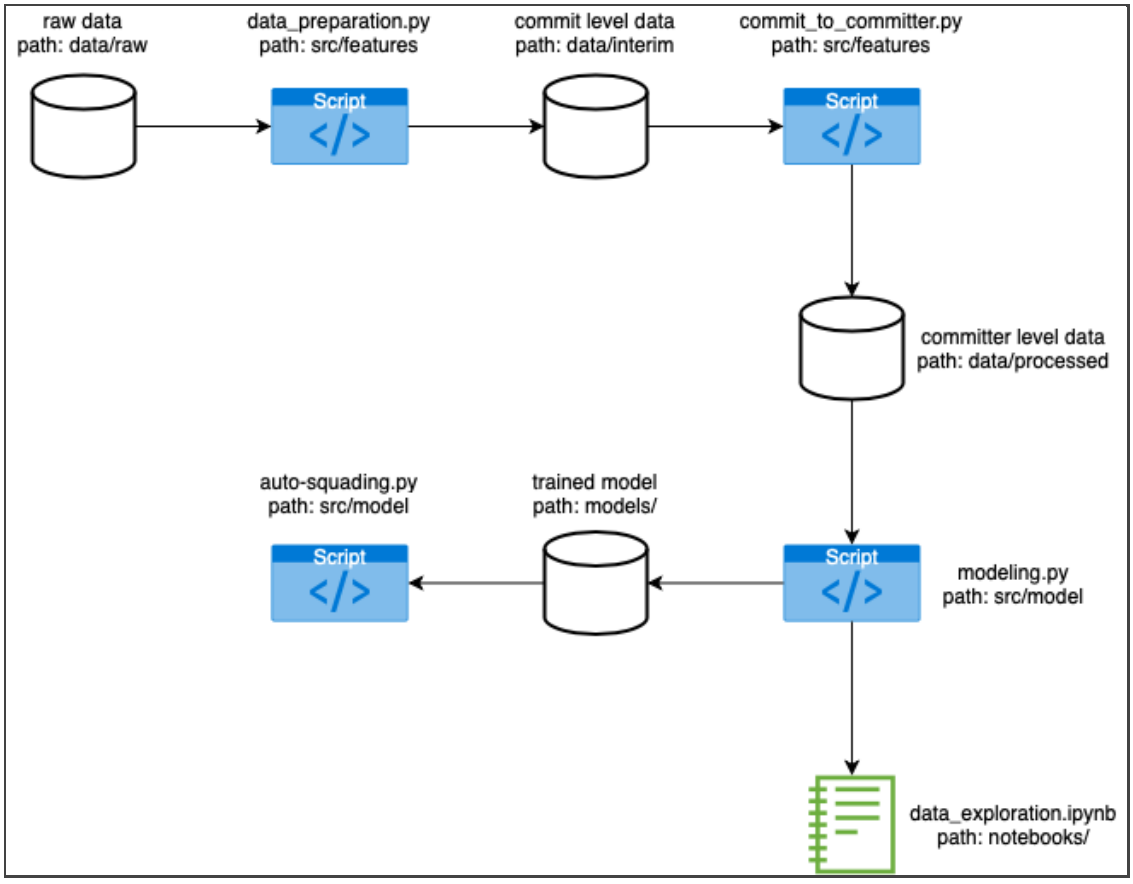
5.8.1 File structure and replication package

As suggested by the course tutor, we will use the Data Science CookieCutter template, which is a logical, reasonably standardized, but flexible project structure for doing and sharing data science work. This standardized structure was designed to allow different data science teams to produce code with good quality which means, correct and reproducible.

Also, a well-defined, standard project structure means that a newcomer can begin to understand and analyze without digging in too extensive documentation and so, they don't need to read 100% of the code before knowing where to look for very specific things.

To carry out this good practice we have copied the structure into our Github repository and changed the necessary things so that it is now specific to our project. The TechLoan Github repository can be found here: https://github.com/AnnaPaty/TechLoan.

Below is an illustration that exemplifies the usage of this template in our project:



5.8.2 Second good practice

- Use Static Analysis to Check Code Quality - Code linters

For instance, a linter is a tool that detects and reports unwanted patterns in computer code to the programmer. They can directly be used in a code editor or they can be executed directly from the command line. It was chosen because the project has four different developers, and so it was very probable that errors appeared. This tool facilitated its solving. Moreover, it contributed to achieving an optimal high-quality code style.

It was finally carried out with the VisualStudio code editor, using pylance extension which automatically diagnoses all the changes introduced into the scripts while writing the code to be able to correct quality issues immediately.

5.8.1 Third good practice

- Write Reusable Scripts for Data Cleaning and Merging

Chosen because it is one of the most (if not the most) important steps during a programming project. Also, it has to be said that it is the fifth most popular good practice, as stated in the “Adoption and Effects of Software Engineering Best Practices in Machine Learning” paper. As the objective is to cluster developers, we should write scripts that can be reused in other projects to cluster other developers.

The specific two scripts that will enable people to do so are *data_preparation.py* and if the grouping by committer instead of having the information ordered by commit is also desired the *commit_to_committer.py*. Those two files were the ones delivered in the first delivery of this project.