Methodology, Technical Description

Languages used: Python, SQL

Libraries from Python: Jupyter notebook (environment), Pandas, Numpy, Matplotlib, Seaborn, Re (regular expression), Collections, Pandasql (SQLite local), Apyori (apriori algorithm for association rules)

Datasets:

Jira Issues dataset: 35234 Kb, 18090 rows.

Jira Changelog dataset: 63592 Kb, 332690 rows.

Dataset for tracking the changes in the Jira issues. Key attributes are project code (`project`) and Jira Issue code (`key`), log creation date (created), username of the author of change (`author`), field that has been changed (`field`), value of the field that was set before the change (`fromString`) and value of the field that was set (`toString`).

Jira Users dataset: 245 Kb

Jira Sprints dataset: 2646 Kb

## The personality of the developers

For getting the Developers personality traits assessment, I have used IBM Watson Personality Insight. IBM Watson offers the Big 5 personality traits assessment if we provide 600 or more words written by the person. Therefore, the first task is to retrieve the texts written by each developer.

To get the text written by the developers, changelog dataset is used. The sub-goal here is to get the texts that were actually written by a developer. The columns that were used from changelog dataset:

* `author`, author of the change, can be taken as-is.
* `project`, code of the project, can be taken as-is.
* `key`, the code of the Jira issue, can be taken as-is.
* `created`, the date and time when the change was performed. Can be taken as is.
* `field` that denotes the field of Jira software that has been changed.
  + Need to detect the `field` values that represent the textual value columns, and filter them.
* `toString` column stores the values that were set to the given field.
  + Need to retrieve the textual values that have actually been written by developers

After the initial check, revealed the `field` values that contain manually written texts:

'summary', 'description', 'Acceptance Criteria', 'Comment', 'Epic Name', 'Out of Scope', 'QA Test Plan', 'Epic/Theme', 'Migration Impact', 'Business Value'

|  |  |
| --- | --- |
| **Text Field** | **Occurrences** |
| Acceptance Criteria | 97 |
| Business Value | 1 |
| Comment | 522 |
| Epic Name | 57 |
| Epic/Theme | 414 |
| Migration Impact | 6 |
| Out of Scope | 10 |
| QA Test Plan | 21 |
| description | 4227 |
| summary | 3695 |

After checking more samples of each type of field, 'summary', 'description', 'Comment', 'Acceptance Criteria', 'Migration Impact', 'QA Test Plan', 'Out of Scope' were the ones that were surely holding the useful textual data, that gives 8578 rows in total.

User may submit a change of the textual value into one column multiple times. To avoid the possibility of such duplicates, it is necessary to include only one edit on each task field for each user. Only the latest change by the action time (`created` column value) has been taken into account.

After detecting the textual `field` values and filtering the changelog respectively, analysis of the set text values has shown, that there is a need of text cleaning since the texts contain not only the manually written sentences but also the automatically generated snippets copied form the various software logs and editors.

Regular expression substitution (re.sub) and simple substring (str().replace) functions have been used to clean the texts. Common parts of the texts that have been excluded by the regular expression:

* Texts within not formatted Jira command tags;
* texts within common Jira code formatting tags;
* texts within code tags, squared bracket tags and Html tags;
* java commands / JDBC calls;
* texts of SQL command execution;
* texts of command execution;
* module calls;
* scheduled job calls;
* texts within MS SQL transaction commands;
* SQL SELECT, INSERT, DELETE statements;
* texts of System version descriptions;
* texts of deployment system technical descriptions;
* texts of system component descriptions;
* texts within the headers generated by the system;
* web-links;
* local path links with slashes and backslashes;
* system logs within the asterisks;
* texts of the word that has the length of more than 18 characters (\*explanation below);
* email addresses;
* SQL command parameters;
* cmd commands;
* texts of words with colons;
* texts of application version numbers;
* non-textual special characters;
* whitespaces;
* dates;

After applying the text cleaning, I calculated the length of the text for each of these changelog text values and created the dataset of the textual values changelog with the following structure of columns: `key`, `project`, `author`, `date`, `field`, `text`, `textlength`.

Each row in the text values dataset represents authors written text on a single task. In the end, the results should be one text row per user, and this text will combine all the texts written by the given developer on all the Jira issues in all the fields. Length of the text will be an important variable then, since the tool that will be used for personality traits assessments – `IBM Watson Personality Insights` demands at least 600 words as an input.

Before combining all the texts of each developer, it`s useful to check the length of the texts for each issue field.

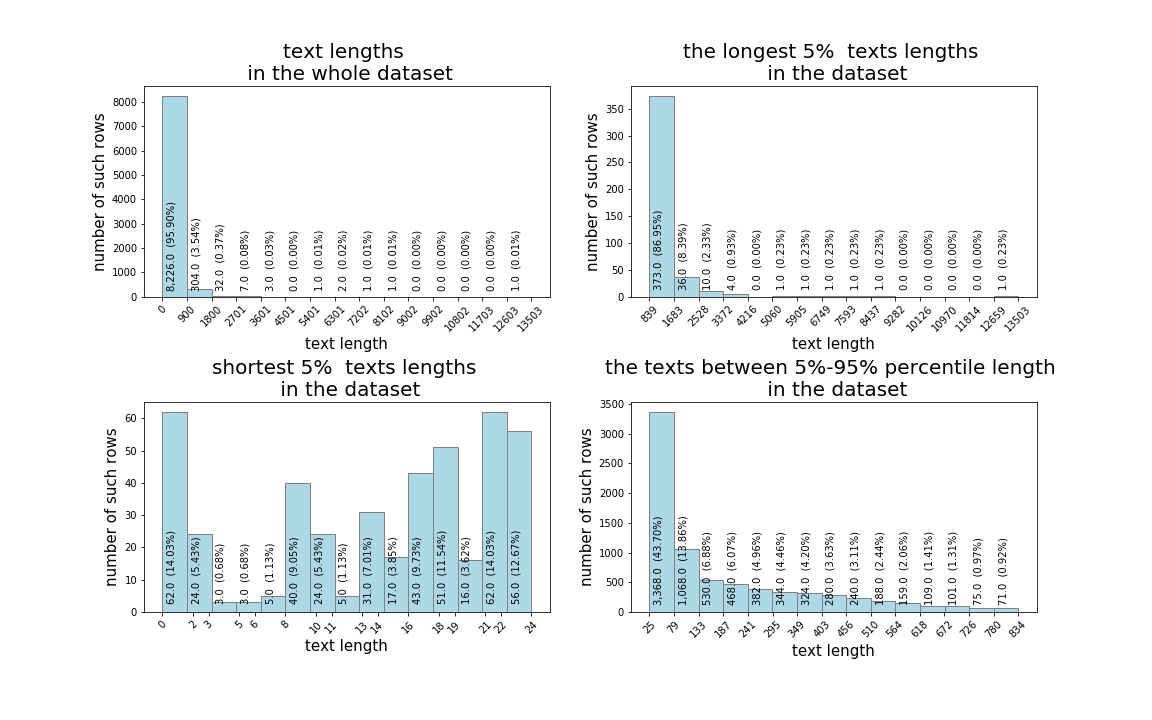


Figure 1. distribution of text lengths.

The detailed plots of text length distribution show that more than 95% of the rows have texts with the length of 900 characters or less.

Longest 5% texts plot shows that majority – more than 86% of these long texts are 1683 characters or less, while we have the rest of the long texts, more than 13% (and out of the whole dataset – 0.13\*0.05=0.065, roughly 6.5%) with an unusually long number of characters. Although the text cleaning procedure excluded the majority of the automatically generated texts, there still are the cases of the texts that are likely to be not written manually by the developers. By manually checking the records with the longest rows it was found out, that some of them still contain the system logs. Removing the top records with the relatively long texts would help to make sure that automatic texts are presented with down to the minimum in our dataset.

Shortest 5% texts rows graph shows that all of them are 24 characters or less, and as many as half of these records are even less than 15 characters long. If we consider the following facts, that each English word on average **\*\***contains 6-8 characters, and that 15 characters would make only two words, it is easy to understand that the texts with two words are less likely to contribute into forming of the 600 words threshold, while on the other hand, these two words can be system commands that are useless and even more, can affect the personality traits assessment results.

Based on the arguments mentioned above, removed the rows with longest 1% texts (86 records with more than or equal to 1504 characters), and rows with shortest 2% texts (172 records, less than or equal to 13 characters).

After cleaning and filtering the dataset, it needs to be grouped by the users. Since the same users could be present into multiple projects, the email address was additionally used to identify the actual users (whenever the email address was present in the users' dataset). As a result, in total, we have 100 unique users that have more than 600 words manually written in Jira fields, therefore, these users are capable to get the assessment of personality traits.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Project*** | tistud | timob | Dnn | xd | mesos | nexus | apstud | mule |
| ***Users*** | 33 | 30 | 20 | 15 | 12 | 11 | 10 | 5 |

Table 1.

Table 1 shows the number of users per group. Notably, the sum of the users in this table is more than the total number users mentioned above (100), because, in this table we present unique users per project, meaning that one user could work in multiple projects.

The next step performed is retrieving the results from IBM Watson Personality Insights API. The texts written by the developers was given to API in JSON format as an input. API has also returned the results in the JSON format, which then was parsed and stored into pandas dataframe.

Resulting JSON file returned by IBM Watson contains assessment of personality trait with percentile and raw score, significance indicator, percentile and raw score results of the facets that belong to the personality. Within this research we only focus on Big 5 Personality traits – Openness, Conscientiousness, Extraversion, Agreeableness, Neuroticism, therefore we ignore the results of the sub-categories of these traits.

Furthermore, IBM API has returned the normalized percentiles and the raw scores. Percentiles scores are generalized on the whole users` dataset of IBM. It is normalized based on all the users that have been requested to Watson, while raw scores are the plain results based solely on the person`s characteristics. Raw scores are the same as what the personality assessment test would return. Both, normalized and raw score percentiles are the double type numbers in range of 0 to 1.

Distribution of personality traits assessment results are shown on Figure 2. top rows on the graph represent the personality assessment percentiles of the users as normalized by IBM, while bottom row shows the respective raw scores of personality assessments percentiles.

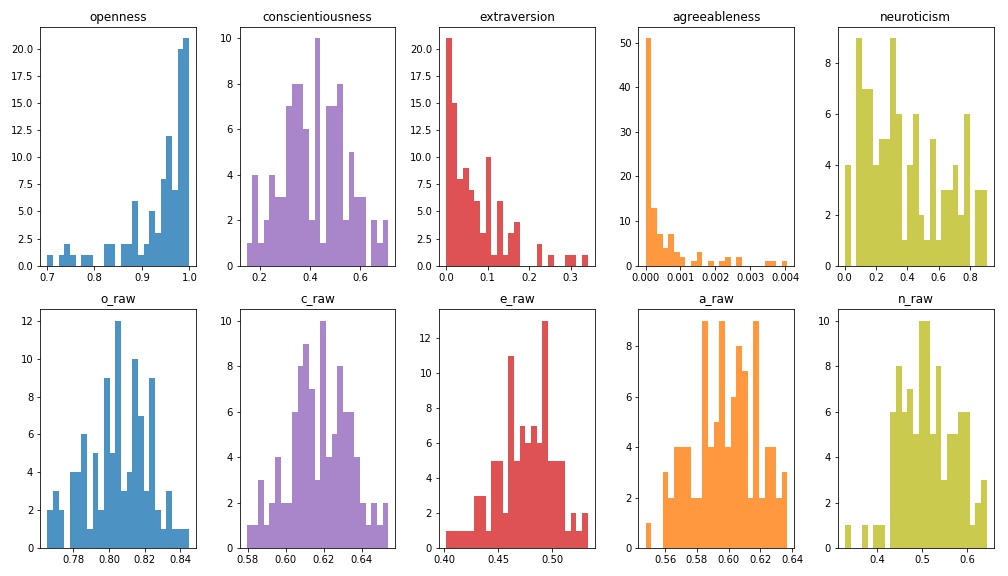
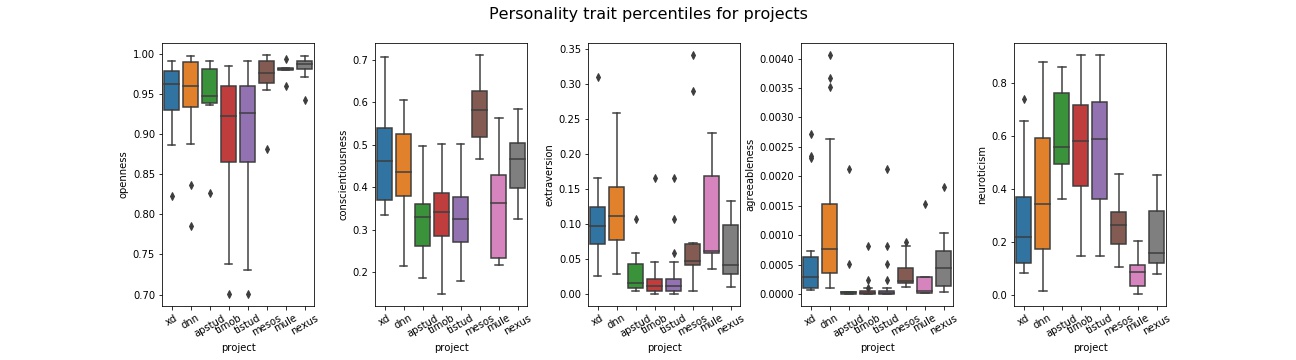


Figure 2: Histograms of users` personality trait raw scores and percentiles

To compare the result of each trait, first look makes it clear that raw scores are more normally distributed, than the normalized scores.

Openness raw score varies from 0.75 to 0.85, while on the normalized percentile it scores from 0.7 to 1, and notably the most of the results here are distributed in 0.95 to 1 percentile bin, meaning that the raw scores of the developer`s openness are higher than the sample population scores. This trait is more or less in accordance within raw and normalized score, similarly to conscientiousness and neuroticism, those have wider percentile range on IBM`s adjusted scores than on raw scores. Extraversion scores are also significantly lower on normalized percentiles, as the developers seem to be less open, than the sample users of IBM. Most distinctive difference between these two scores are returned on agreeableness. Raw score seems reasonable – varying from 0.54 to 0.64, but the normalized scores show extremely low scores - less than the 0.004, meaning that the absolute majority of the IBM sample users have more scores than all the users of our dataset.

Same results summarized within the projects are present on Figure 3. It shows that certain projects users tend to have more polar personalities, than the others, specifically project `mesos` developers are more open and conscientious, project `xd`, `dnn` and `nexus` developers are also conscientious, extravert and agreeable, and finally `timob` and `tistud` project developers share similarities in being less open, conscientious, extravert, agreeable, and more neurotic, than the other project developers. Notably, the last two projects have significantly more users checked for IBM Personality traits, than the others, as given in Table 1, giving more confidence in their results as a group of developers. This can be the reason, why these two projects have similar results in all the personality traits.



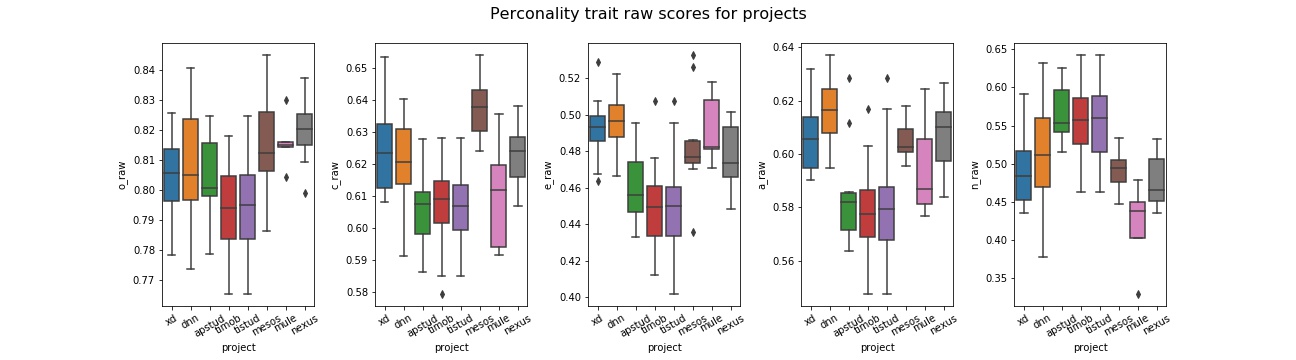


Figure 3. boxplots of users` personality trait raw scores and percentiles per project

When choosing the one measurement to use among these two, we opted to use raw scores within the research, based on the following advantages:

* IBM uses sample population for normalizing the scores, that may not be applicable and proportionate to the developers`. On the contrary, raw scores can be normalized using the results of the users of inputted dataset, resulting in the relatively more or less open/conscientious/extravert/agreeable/neurotic scores that is comparable within the given dataset.
* Results of raw scores were normally distributed, unlike the normalized scores, that were imbalanced, as shown on Figure 2.

## JIRA Metrics

This research is in accordance with the paper of E.Scott which used Felder-Silverman Learning Styles to explain the SCRUM methodology preferences and metrics, hence we use the same SCRUM metrics.

|  |  |  |
| --- | --- | --- |
| **Metric** | **Value** | **Comment** |
| State | to do | avalable |
| State | doing | available |
| State | done | available |
| Prioritization | low | available |
| Prioritization | medium | available |
| Prioritization | high | available |
| Estimation | low | available |
| Estimation | medium | available |
| Estimation | high | available |
| Time | low | available |
| Time | medium | available |
| Time | high | available |
| Role | Developer | Not Available in our JIRA dataset |
| Role | Support | Not Available in our JIRA dataset |
| ScrumMaster | Yes | Not Available in our JIRA dataset |
| ScrumMaster | No | Not Available in our JIRA dataset |
| Recommendations | OK | Not Available in our JIRA dataset |
| Recommendations | Ignored | Not Available in our JIRA dataset |

Table 2.

E. Scott has calculated the categorical variables of State, Prioritization, Estimation, Time, Role, ScrumMaster and Recommendations metrics in his paper. Out of these, State, Prioritization, Estimation and Time can be retrieved from the Jira issues and changelog datasets that we have obtained open source, however Role, ScrumMaster and Recommendations are not available.

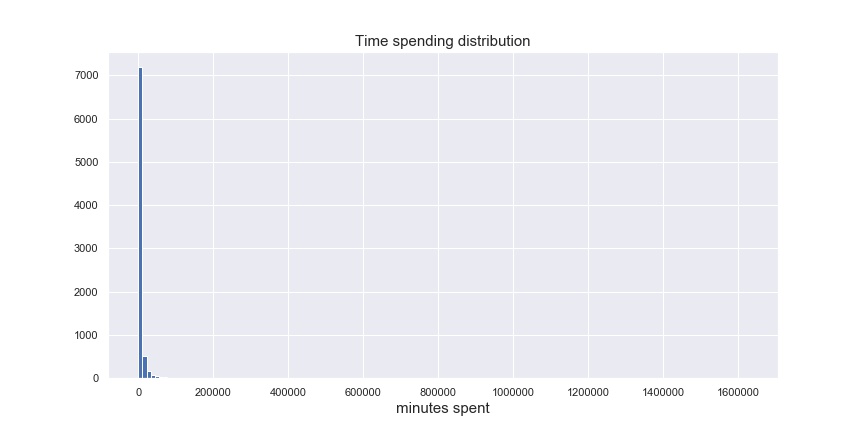
First of all, we shortened the scope of the metric calculations to take into account only the users that have been checked on IBM Personality Insights on the previous steps. All the changelog records performed by the other users than the ones mentioned previously, got filtered out.

**Time metric**

For calculating **Time** metric, we used Jira issues change log dataset. First, we defined the time spent per task by a developer: time that is passed while the task was set to 'In Progress' status. In technical terms, this is the time between the two log records when 1) the task status was set to 'In Progress' and 2) task status was changed from 'In Progress'. Therefore, we get the records from Jira change log, that have status set from In Progress to any other status, and the records that have set status set to In Progress from any other status.

In case when several developers have worked on one task – we calculate time for each of them separately. In case when one developer has set status to 'In Progress' multiple times within one task, the function calculates the sum amount, so that each developer has one number of minutes spent on one task.

The histogram on Figure 4 shows the distributions of spent minutes by the developers. Noticeably, there are outliers that puts the vast majority of the records into the first few bins.

Figure 4. Histogram of time spending by developers

To have the better picture, need to filter out the records with minutes spent greater than 5000 and less than 1, the result is on the Figure 5. Now it is visible that although the majority of the records are with the 300 minutes or less, there still are considerable amount of records with higher number of minutes spent.

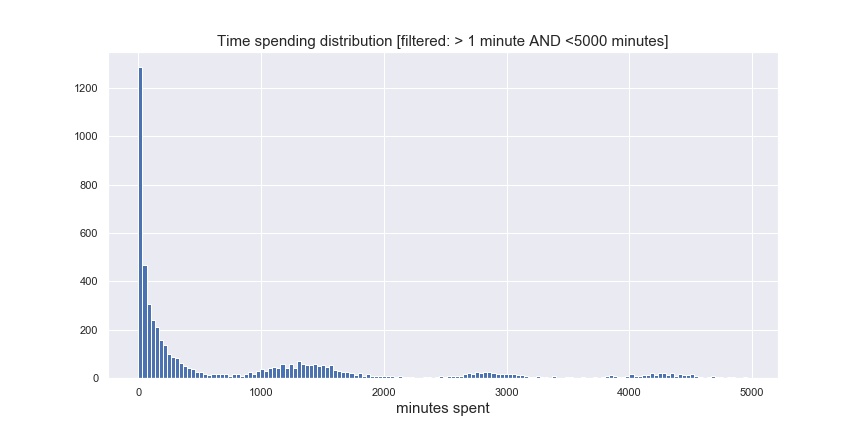


Figure 5. Histogram of time spending by developers on a subset of dataset.

The end-goal of getting the Time measurement is to have the categorical variables. we defined the top and bottom threshold for high, medium and low time spent tasks considering the number of records that would fall into each of these categories, and taking into account the reasonability of each category boundaries:

Low time spent category: tasks, that took 1 working day or less. Equal to 8 working hours, and equal to 480 minutes spent, respectively.

Medium time spent category: tasks, that took more than 1 day and up to 1 week, which is equal to 5 working days, and equal to 2400 minutes spent, respectively.

High time spent category: tasks, that took more than a working week (2400 minutes spent) to complete.

Out of the whole dataset of time metric calculated previously (8093 rows in total), the low time spent category is 4159 rows (52%), medium time spent category holds 1418 rows (17%) and high time spent category contains 2516 rows (31%).