Rule ID	Scrum Rule Definition	Implementation through Python Scripts, Scrum Process Mining Filter (Apromore, ProM, Disco)	Theoretical Calculation Roadmap Based on the Data Fields	Data Fields Used	Scrum Phase Affected	Scrum Event Affected	Scrum Role Affected
BEGINNING OF SCRUM RULES							
SCRUM RULES RELATED TO THE DEVELOPMENT TEAM							
#R1	Each Scrum sprint is four weeks or less in duration [Scrum Guide - Page 7 - The Sprint]	"No more than five weeks should elapse for a single sprint" (Apromore Filter) Python Script: [enter script link - to be completed]	Given the data fields from the dataset, we can check this Scrum rule using the Start Time (*created*), End Time (*resolutiondate*) or Last Updated Time (*updated*) data fields, which indicate the timeline of the issues (start, finish and last updated timestampts). Since these fields show information about the issues and not directly about the sprints, and we do not have other fields that are related directly to the sprints duration, we can still calculate the duration of the sprints through the available data fields. Firstly, we will group the issues belonging to specific sprints of specific projects. Therefore, we will need to use the columns: *key, sprint* and *project.* After we have organized the issues in the corresponding sprints, we can calculate the sprint duration by extracting the differene of time between the completion of the issue that has been reported the last (completion time of the last issue in the sprint) and the start time of the issue that has been the first issue to be reported (the issue with earliest creation date). We can therefore create another data field in the dataset, called: *sprint.duration*, where we will inject the calculated value representing the sprint duration in a weekly unit. Lastly, we can make use of this rule to support the checking of other subsequent rules.	*1. created* *2. resolutiondate* *3. updated* *4. sprint* *5. key* *6. project*	The Game Phase	Sprint	Developers, Product Owner, Scrum Master
#R2	"The Scrum Team is small enough to remain nimble and large enough to complete significant work within a Sprint, typically 10 or fewer people" [Scrum Guide - Page 5]	"The number of the Scrum Team members per project should not be **largely** less or more than 10" Python Script: [enter script link - to be completed]	Given that the data field **assignee.name** represents the developers within the team, and the **reporter.name** represents the POs, we can count the most active developers and PO from these data fields respectively, thus getting clearer insights on the approximate number of members within each project team.	*1. assignee.name* *2. reporter.name*	The Pregame Phase (planning the Scrum Team)	Sprint Planning, Sprint	Developers, Product Owner, Scrum Master
#R3	The Development team should consist of 3 - 8 members [Scrum Guide]	"No more than 8 active developers should be involved in development tasks" Python Script: [enter script link - to be completed]	Given that the data field **fields.assignee.name** we can count the unique assignees which represent the developers to whom the tasks are assigned.	*1. assignee.name*	The Pregame Phase (planning the development team)	Sprint	Developers
#R4	Every Scrum sprint has the same length Rephrasing it to: Sprints should be completed at a sustainable pace - in order to improve and retain the Scrum Team's focus and consistency [Scrum Guide - Page 5]	"The duration of all sprints should follow similar pace" Python Script: [enter script link - to be completed]	Given the data fields from the **jiradataset_issues**, we can check this rule by comparing the final output from the #R1, which checks the duration of the sprints. After the duration of the sprints has been calculated, we can later compare whether each of the sprints has reported the satisfactory weekly duration, i.e. four weeks or less. For the iterations that last more than 4 weeks, we will not consider them as agile practies, therefore deviating from the Scrum prescriptions or rules.	*1. created* *2. resolutiondate*	The Game Phase	Sprint	Developers, Product Owner, Scrum Master
#R5	An issue (bug, task) must firstly go to "in-progress" state before "done" or "completed" state [A general known practice - Rule of Thumb]	"No issue should be completed without being a work in progress first" Apromore Filter - to be completed	Given the data fields from the **jiradataset_issues**, we can check this rule by inspecting the **status.name** data field. This column indicates the different statuses that issues go through from their inception.	*1. status.name*	The Game Phase	Sprint	Developers
#R6	All development tasks must be tested before marked "done" Rephrase this to: The development lifecycle must not include any anomalies or irataional flow of tasks [A general known practice - Rule of Thumb]	"There should be a testing activity before the final end state" Apromore Filter - to be completed	Given the data fields from the dataset, we can proceed to check this rule by inspecting the columns: **status.name** and **status. statusCategory.name.** Since these two columns convey necessary information about the status of the tasks, we will check whether the development tasks reflect a status of *in-testing,* which as an agile practice eventually happens before the tasks are marked as *done*. We can investigate whether the tasks (*issuetype.name*) present in the Sprints (*sprint column*) undergo testing before marked as *complete* or *done*.	*1. status.name* *2. sprint* *3. issuetype.name*	The Game Phase	Sprint	Developers

#R7	Sprints should not overlap, rather, they should be kept within the timebox [Scrum Guide - Page 6 - SM accountabilities]	"A next Sprint execution should begin only after the previous Sprint's resolution" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by analyzing and processing the columns: *created* and *resolutiondate*. Note that these two columns correspond to the creation and resolution of individual issues, however, we can group the issues into sprints by using the *sprint* column as an indicator of which issue belongs to which sprint. Later, we can discover the first issue that was created (earliest time of *created* column) as well as the last issue (latest time of *created*) for a specific sprint. Finally, we can extract the difference between the end time of the resolution of the last issue and the start time or creation of the first issue. Using these final value which indicate the duration of the sprints, we can check whether sprints overlap with each other.	*1. created* *2. resolutiondate* *3. sprint*	The Game Phase	Sprint	Developers, Product Owner
#R8	Issues are defined concisely, thus containing a start time/date [Scrum Guide - Page 6 - Bullet Points] [Scrum Guide - Page 10 - Product Backlog - Last Line]	"There should be timestamp indicating the issue development kick-off" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting the column: *created*, as this column indicates the starting time of the issues. We will filter out the issues (regardless of their type), that do not have a starting time, i.e. an appropriate value in the *created* column is missing.	*1. created*	The Game Phase	Sprint Planning	Product Owner, Developers
#R9	Issues are defined concisely, thus containing an end time/date [Scrum Guide - Page 6 - Bullet Points] [Scrum Guide - Page 10 - Product Backlog - Last Line]	"There should be timestamp indicating the issue development completion" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting the column: *resolutiondate*, as this column indicates the completion time of the issues. We will filter out the issues (regardless of their type), that do not have a resolution/finish time, i.e. an appropriate value in the *resolutiondate* column is missing.	*1. resolutiondate*	The Game Phase	Sprint	Product Owner, Developers
#R10	Issues associated with high story points carry on bigger complexity, thus requiring more time to complete [Scrum Guide - Page 7 - Last Line]	"The duration of the sprints with higher story points should be longer than vice-versa" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting several columns, such as: *created, resolutiondate, storypoints, sprint, key*. Firstly, we will group the issues into their corresponding sprints by matching the issue ID with the sprint ID. Later on, for each issue within a particular sprint, we will calcualte the duration of the issue, i.e. how long did the issue execution took. Accordingly, we can sort the issues based on the duration in an descending order. And finally, we can compare the different story points associated with each issue, and check for output that does not comply with our rule, meaning, issues of lower complexity and story points, which take more time to complete than issues carrying higher amounts of story points and complexity.	*1. created* *2. resolutiondate* *3. storypoints* *4. sprint* *5. key*	The Game Phase	Sprint	Developers
#R11	There should not be incomplete or meaningless issues on the backlog This is rephrased to: The backlog does not contain meaningless or empty issues [Scrum Guide - Page 6 - Bullet Points] [Scrum Guide - Page 10 - Product Backlog - Last Line]	"There should be a project identifier attached to each issue" Python Script: [enter script link - to be completed]	Given all the data fields in the dataset, we can check this rule by taking into account the columns: *key, sprint* and *project.* The *project* column indicates the open-source projects that the issue belongs to, while the *sprint* column indicates the actual sprint under which the issue has been developed. As stated in other rules, the *key* column represents the identifier of the issue. This rule aims to check that all issues present in the dataset, belong to a specific project, which is a valid rule, since there might be cases that a particular issue is not associated with a corresponding project. In agile software development, it only makes sense that an issue belongs to a development iteration for a project. Therefore, we will identify those issues hat do not belong to any project (check missing values in project column), and report descriptives results from all open-source projects. We can proceed wit providing comparative analysis among the projects and the number of issues that each project has.	1. key 2. sprint 3. project	The Game Phase	Sprint Planning, Sprint	Product Owner, Developers
#R12	Issues associated with higher priority in the backlog are the first ones to be completed during the Sprints [General known practice - Rule of Thumb]	"Higher priority issues should have earlier resolution date than low priority issues" Python Script: [enter script link - to be completed]	Given all the datafields in the dataset, we can check this rule by obtaining further information from the columns: *priority.name, created, resolution.date, project* and *sprint.* We will first filter the issues that have no priority attached to them. Later, we will order the issues by the project, sprint and the priority associated to them in a descending order (higher priority issues to lower priority issues). This will make it more organized in order to inspect the issues of individual open-source projects. Later, for each issue and its priority label, we will check its creation and completion date, to verify that indeed higher priority issues have been taken care of before the other issues. The other case signigifes that developers did not correctly respect the backlog, therefore violating this Scrum rule.	1. priority.name 2. resolution.date 3. project 4. sprint	The Game Phase	Sprint	Developers

#R13	Shorter Sprints can limit the risk of cost and effort to a smaller time frame (smaller story points) [Scrum Guide - Page 8 - The Sprint - First Line]	"Shorter sprints take less time to complete and are assigned smaller story points" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by analyzing the columns: created and resolutiondate. Though these two columns correspond to the creation and resolution of individual issues, however, we can group the issues into sprints by using the sprint column as an indicator of the issues. Furthermore, we can report the first issue that was created (earliest time of created column) as well as the last issue (latest time of created) for a specific sprint and their corresponding start and end time. Finally, we can extract the difference between the end time of the resolution of the last issue and the start time or creation of the first issue. Using these final value which indicate the duration of the sprints, and by summing up the story points values for each sprint, we can verify whether shorter sprints actually add up to smaller numbers of total story points for each sprint.	1. created 2. resolutiondate 3. sprint 4. key 5. storypoints	The Game Phase	Sprint	Developers, Product Owner
#R14	A new Sprint starts as soon as the previous one ends, allowing the developers to deliver value iteratively old version: A new Scrum Sprint starts immediately after the completion of the previous Sprint, in order to preserve the pace and Sprint timeboxing [Scrum Guide - Page 7 - The Sprint] [Scrum Guide - Page 6 - SM accountabilities] There are no Breaks Between Sprints / Development team delivers products iteratively and incrementally, maximizing opportunities for feedback.	"No considerable amount of time should elapse between the finish of a sprint and the beginning of the new sprint" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by analyzing the columns: *created* and *resolutiondate*. Though these two columns correspond to the creation and resolution of individual issues, however, we can group the issues into sprints by using the *sprint* column as an indicator of the issues. Furthermore, we can report the first issue that was created (earliest time of *created* column) as well as the last issue (latest time of *created*) for a specific sprint and their corresponding start and end time. Finally, we can extract the difference between the end time of the resolution of the last issue and the start time or creation of the first issue. Using these final value which indicate the duration of the sprints, we can check whether there are breaks or considerable amount of times slacking off between the sprints.	*1. created* *2. resolutiondate* *3. sprint*	The Game Phase	Sprint	Developers
#R15	The Scrum Team does not include any other people (e.g. a manager who doesn't do tasks) [Scrum Guide - Page 5]	"There should not be any unexpected or random person involved at a rather strange point/part of the Scrum activities, phases or events." Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check for this rule by inspecting the columns of **asignee.name, creator.name, reporter. name, key, sprint,** and **project.** We will first cluster the different projects and the sprints associated with each project. The same applies for grouping the issues based on the sprint they belong to. Finally, we can list the unique names of the people who are employed in the Scrum activities and check for their contributions and obligations (who was the issue reporter, issue creator and issue assignee) in order to check whether there are persons that come into the picture in an unexpectedly uncommon and unusual point/part of the development process.	1. asignee.name 2. creator.name 3. reporter.name 4. key 5. sprint 6. project	The Game Phase	Sprint	Developers, Product Owner, Scrum Master
#R16	The top PBIs are small enough (effort) that several can fit into a single Sprint ???	"There should be a higher amount of issues reported for the first sprints as opposed to the last sprints of the development phase" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting the columns of **key** and **sprint**. For each sprint, we will firstly group the issues that belong to the corresponding sprint. Afterwards, we will count the number of issues present in each sprint, and use the final results to check against potential deviations to this rule.	1. key 2. sprint	The Game Phase	Sprint	Developers, Product Owner
#R17	Developers volunteer for a new task from the Sprint backlog as soon as they complete the current task [Scrum Guide - Page 5 - Developers]	"There should not be a considerable amount of time for a developer to volunteer and start a new issue (task) after she/he has completed the previous issue (task)" Python Script: [enter script link - to be completed]	Given the data fields from the dataset, we can check this rule by inspecting the columns of **asignee.name**, **created** and **resolutiondate.** The first column (**asignee.name**) represents the name of the developer who is assigned to a specifc issue (task). By analyzing and preprocessing the fields of **created** and **resolutiondate**, we extract the time that a developer took between the completion of one issue and the start of the successive one, therefore check for adherence or deviations to this Scrum rule.	*1. assignee.name* *2. created* *3. resolutiondate*	The Game Phase	Sprint	Developers

#R18	Developers can create Sprint Backlog Items, but they do not report issues *not sure - need to confirm*	"There should not be any issue reported or created by the developers themselves" Python Script: [enter script link - to be completed]	Given the data fields from the dataset, we can check this rule by inspecting the columns of **asignee.name, creator.name** and **reporter.name**. The first column represents the developer assigned to the issue, while the second field represents the Product Owner (i.e. the person who has created the issue), while the last column (**reporter. name**) also represents the Product Owner, as per the obligations. We can check this rule by comparing the name of the person in the **asignee.name** column against the values or name of persons found in **creator.name** and **reporter.name**. In the cases where these data fields share the same value, we can conclude that the developer has created and reported the issue, thus violating this rule of Scrum.	*1. assignee.name* *2. creator.name* *3. reporter.name* *4. key*	The Game Phase	Sprint	Developers
#R19	Developers, amid other responsibilities, are accountable for creating a plan for the Sprint, also known as the Sprint Backlog Items. [Scrum Guide - Page 5 - Developers]	"Active members of the development teams should be included in additional activities, other than development" Python Script: [enter script link - to be completed]	Given the data fields from the dataset, we can check for this rule by inspecting the active developers in the **assignee.name** field, and checking whether any of those developer names is to be found on the **creator.name** data field, which denotes the roles responsible for planning the Sprints artifacts and activities.	*1. asssignee.name* *2. creator.name*	The Pregame Phase	Sprint Planning	Developers
#R20	Issues resolved in previous increments do not appear again in future increments old version: Each Increment is additive to all prior Increments and thoroughly tested, ensuring that all Increments work together, and that issues resolved in previous Increments do not appear again in future Incerements. [Scrum Guide - Page 5]	"No previously resolved issue should reappear in a future sprint" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting the columns of **key, sprint.** Firstly, we will group the issues that belong to their specific sprints by using the aforementioned columns. Afterwards, we will check whether issues belonging to the very first sprint for instance, have reappeared in another sprint during the development phase. Identifying such abnormal phenomenon, means that there are deviations among the development teams as regards this rule of Scrum.	1. key 2. sprint	The Game Phase	Sprint	Developers
SCRUM RULES RELATED TO THE PRODUCT OWNER							
#R21	All Scrum Teams must have only one Product Owner Rephrase this to: "The Product Owner is one person, not a committee" [Scrum Guide - Page 6]	"No more than one Product Owner should be employed in the Scrum Team (resource perspective)" Apromore Filter - to be completed Python Script: [enter script link - to be completed]	Given that the data field **reporter.name** indicates the role of athe Product Owner (issue reporter), we can therefore utilize this data field and count the unique names (usernames) of the issue creators, which represent the POs responsible for managing the Product Backlog and the items in it.	*1. reporter.name*	The Pregame Phase (planning the development team)	Sprint	Product Owner
#R22	Scrum Sprints are uniquely identifiable [Scrum Guide - Page 7 - The Sprint]	"There should be a unique identifier associated with each Sprint" Python Script: [enter script link - to be completed]	Given the data fields from the dataset **jiradataset_issues**, we can check this rule by inspecting the values present in the **sprint** data field. This data field represents the identifier of the Sprints, and iterations having no such identifier cannot be considered as Sprints (agile).	*1. sprint*	The Game Phase	Sprint Planning	Product Owner
#R23	Issue have a corresponding type (bug, task, etc.) [Scrum Guide - Page 6 - Bullet Points] [Scrum Guide - Page 7 - SM serves PO]	"There should be a type, such as bug, improvement or task, associated to each issue" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by investigating the column: *issuetype.name***.** This column indicates the type of the issue that is created or reported by the Product Owner. There are several types that issues can be labelled with, for example: bug, story, task, etc. I will firstly check if there are missing values recorded for this column. The occurence of such missing values indicates that the development team does not know what type of issue they are about to deal with, and that the PO did not specify well the issue details, which as per the agile principles, is not satisfactory. After removing the missing values, I will check for the distintive values, i.e. issue type names, that have been recorded in the dataset, and report the most frequent issue types as well as the least occuring ones.	*1. issuetype.names*	The Game Phase	Sprint Planning	Product Owner

#R24	Each issue belongs to a specific Sprint [Scrum Guide - Page 6 - Bullet Points] [Scrum Guide - Page 7 - SM serves PO]	"There should be a sprint identifier attached to each issue" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule taking into account only the issues that already have a type name. We can begin by checking the column *issuetype.name*, and simultaneously inspect the *sprint* column, whihc indicates the identifier of the sprint. This way we can filter out the issues that do not belong to any of the sprints from each project.	*1. issuetypes.name* *2. sprint*	The Game Phase	Sprint Planning	Product Owner
#R25	All issues must be uniquely identifiable [Scrum Guide - Page 6 - Bullet Points] [Scrum Guide - Page 7 - SM serves PO]	"There should be a unique identifier associated with each issue" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting the column: *key*, which indicates the unique identifier of a specific issue. Since all agile-managed projects with Jira automatically create IDs for each issue (task, bug, etc.), we will firstly filter out the issues that do not contain any *key* or *ID*. This rule can be later beneficial while tracking the changing status of the issues, which is needed to check other rules as well.	*1. key*	The Game Phase	Sprint Planning	Product Owner
#R26	Scrum Sprints have a starting time [Scrum Guide - Page 7 - The Sprint]	"There should be timestamp indicating the sprints kick-off" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting the column: *created*, as this column indicates the starting time of the issues. However, before doing that, we can group all of the issues into their corresponidng Sprint. To to that, we can use the columns: *key* - which is the unique identifier of the issue and *sprint* - which represents the identifier of the sprints. By grouping the issues based on the sprints they belong to, we can check the issues that were created the earliest within each individual sprint, which at the same time indicates the starting point or starting time for the sprint itself as well.	*1. key* *2. sprint* *3. created*	The Game Phase	Sprint Planning	Product Owner
#R27	Scrum Sprints have a completion time [Scrum Guide - Page 7 - The Sprint]	"There should be timestamp indicating the sprints completion" Python Script: [enter script link - to be completed]	Given all the data fields from the dataset, we can check this rule by inspecting the column: *resolutiondate*, as this column indicates the completion time of the issues. However, before doing that, we can group all of the issues into their corresponidng Sprint. To to that, we can use the columns: *key* - which is the unique identifier of the issue and *sprint* - which represents the identifier of the sprints. By grouping the issues based on the sprints they belong to, we can check the issues that were completed or marked as *done* the latest within each individual sprint, which at the same time indicates the completion time for the sprint itself as well.	*1. key* *2. sprint* *3. resolutiondate* *4. issuetype.name*	The Game Phase	Sprint Retrospective	Product Owner
#R28	The number of PBIs selected from the Backlog for a Sprint depends on the developers, but it should not be zero PBIs per sprint [General Known Practice - Rule of Thumn]	"There should be a minimum of one issue, representing a Sprint Backlog Item, per each Sprint" Python Script: [enter script link - to be completed]	Given the data fields from the dataset, we can check this rule by inspecting the columns of **sprint, key** and **project**, which serve as unique identifiers of the sprints and issues, respectively. We will firstly group the issues corresponding to a particular sprint, and then *count* the number of issues for each of the sprints. This way, we can report on the number of issues per sprint, identify potential sprints having no issues associated to them, and also compare the total number of issues among the sprints of a particular project.	*1. key* *2. sprint* *3. project*	The Game Phase	Sprint Planning, Sprint	Product Owner
#R29	Higher Story Points means higher priority to do the task (issue.priority) ???	"Issues associated with higher story points are labeled as higher-priority issues compared to the issues with lower story points"	Given all the data fields from the dataset, we can perform checking on this rule by inspecting the columns **priority.name** and **storypoints.** Firstly, the issues will be grouped by the sprints they belong to, through the columns **sprint** and **key.** Thereafter, we will compare the issues of each sprint with regards to the story points associated to them alongside the priority name. We will extract a list of pairs (story points, priority name) for the issues, and detect anomalies or unusual associations of these two pairs. Thus, we can see the (non) compliance of the development teams to this rule of Scrum.	1. priority.name 2. storypoints 3. key 4. sprint	The Game Phase	Sprint	Product Owner

#R30	The Product Owner ensures that the Product Backlog is well-understood, visible and transparent. Therefore, the Product Backlog items should be well-written and clearly communicated to the developers with correct information. [Scrum Guide]	"Product Backlog Items should be understandable, therefore should contain a clear description, name, priority, and be identified correctly"	Given all the fields from the dataset, we can check this rule by inspecting the columns of **description, issuetype.name, priority. name, summary, storypoints, key.** Since all of these rules display information about the issues reported by the Product Owner, we can check for the existence of issues that are against the principles of transparency, visibility and understandability. Therefore, we will report the issues that are not clearly reported and understandable (by the data fields listed above).	1. description 2. issuetype.name 3. priority.name 4. summary 5. storypoints 6. key	The Game Phase	Sprint	Product Owner, Developers
SCRUM RULES RELATED TO THE SCRUM MASTER								
#R31	The Scrum Master must ensure that all Scrum events happen while respecting the timebox (if all previous rules (especially the timed ones) pass - then it means that the team complies to this rule too) [Scrum Guide]	"There should not be any Sprints that are left incomplete or whose duration differ greatly from other Sprints"	Given all the fields from the dataset, we can check this rule by inspecting the fields: **created** and **resolutiondate** for each issue within each Sprint, to firstly calculate the duration of the Sprint (as calculated in #R1). Thereafter, we can check whether there are Sprints without an end date, or Sprints that last way longer, respectively shorter, as compared to the general duration of the Sprint.	1. created 2. resolutiondate	The Game Phase	Sprint	Scrum Master
#R32	The Scrum Team includes the Product Owner, ScrumMaster and the Team Members	We cannot check the scrum master with the data we have??? We miss on this information. This aspect can be mentioned in the Implications/Future Work.		1. assignee.name, 2. creator.name 3. reporter.name			
END OF SCRUM RULES								
ADDITIONAL								
#R27	Product Owner is one person, not a committee.	Already checked above in #R19.					