Criteria	Very good	Sufficient	Needs Improvement	Remarks
1. Github Repository				
1.1. Research Motivation				
The research question is clearly articulated and important.	The research question is clearly stated and is feasible, interesting and important.	The research question is clearly stated and is feasible and somewhat important.	The research question is not clearly articulated and/or not very feasible or relevant.	
The choice for the research method (e.g., regression analysis) is motivated well.	The chosen method is appropriate and well justified.	The chosen method is appropriate but the reasoning could be elaborated further.	The choice of method is either unsuitable and/or the justification is limited or weak.	
The way of deployment (e.g., PDF report, dashboard, …) is useful and accessible to potential knowledge users, and clearly communicates the conclusions of the analysis.	The deployment format is highly effective in communicating the conclusions of the analysis.	The deployment is functional but there is scope for improvement in its accessibility or clarity.	The deployment does not effectively communicate the findings.	
The automated and reproducible workflow is of potential use to other students and the larger scientific community.	The workflow is very relevant and useful to the broader community.	The workflow is relevant but its usefulness can be improved through clearer documentation.	The workflow is not very relevant or lacks proper documentation limitng its usefulness to the broader community.	
1.2. Repository structure and documentation (10%)				
The end-to-end workflow, kickstarted with one of the workflow templates available at Tilburg Science Hub, is made publicly available on GitHub. The repository contains a readme.md (in markdown format, so that it renders well on GitHub), which clearly explains the project's goal, and provides instructions to potential contributors/replicators on how to run the project.	The workflow template is good. The README.md is clear, well-formatted, and provides comprehensive instructions for contributors.	The workflow uses the template effectively. The README.md outlines the project's goal but could provide more detailed instructions for contributors/users (e.g. dependencies, running instructions).	Limited use of the workflow template and/or the README.md is either unclear, poorly formatted, or lacks essential instructions for contributors.	
The project has a concise and accurate name, enticing the potential user to explore the workflow. An appropriate short name for the repository's location is chosen (e.g., github. com/yourusername/investigating-airbnb).	The project name is concise, accurate, and engaging. The repository URL is appropriately short and descriptive.	The project name is relevant but could be more concise or enticing. The repository URL is suitable but lacks a bit of clarity or appeal.	The project name is uninformative or overly complex. The repository URL is too long/unclear.	
Additional metadata on GitHub is provided (e.g., a short project description), so that the repository feels and looks professional and complete.	Comprehensive metadata, including a clear project description, is provided, giving the repository a professional and complete appearance.	Basic metadata is present, such as a brief project description, but additional details would enhance the repository's professional look.	Metadata is missing or incomplete, making the repository feel unprofessional and lacking in essential information.	
1.3 Breadth of contributions and way-of-working (10%)				
Multiple team members have actively contributed ("committed") to the repository, for the entire duration of the project (i.e., do not just version your files at the end, but from beginning to end). Commit messages are accompanied by concise and clear commit messages (git log).	Multiple team members have contributed actively throughout the project. Commit messages are frequent, concise, and clearly describe the changes made.	Team members have contributed to the repository, but contributions are sporadic. Commit messages are generally clear but lack detail or consistency.	Few contributions from team members, with most commits concentrated near the end of the project. Commit messages are unclear, infrequent, or missing.	
Students have made active use of GitHub Issues and the GitHub Project Board with the "scrum"-inspired columns "backlog", and the current sprint's "to do", "in progress", and "done".	Active use of GitHub Issues and Project Board is evident, with well-maintained 'scrum'-inspired columns ('backlog', 'to do', 'in progress', 'done') that clearly track project progress.	Some use of GitHub Issues and Project Board is present. Columns are utilized but could be better organized or updated more frequently to fully reflect project status.	Limited or no use of GitHub Issues and Project Board. Columns are missing, empty, or not used effectively to manage the project's workflow.	
Students are assigning issues to one another, and integrating new features by means of pull requests from feature branches to the main branch.	Issues are actively assigned among team members, and new features are seamlessly integrated using pull requests from feature branches to the main branch.	Issues are sometimes assigned, but the process is inconsistent. Pull requests are used for integrating features, but they often lack detailed descriptions or peer review, and the workflow could be more systematic.	Issues are rarely or never assigned, and feature integration via pull requests is minimal or missing, indicating a lack of collaborative workflow.	
2. Data Preparation & Analysis				
2.1 Data exploration (10%)				
All raw data files are programmatically downloaded from the internet.	All raw data files are successfully and efficiently downloaded programmatically,	Some data files are downloaded programmatically, but the process is inefficient or requires manual intervention for certain files.	Data files are not downloaded programmatically, relying entirely on manual download, which affects the project's reproducibility.	
Meaningful RMarkdown reports for (types of) raw data/input files are created, which allow potential users of your repository to understand the content of such files, and the definition of variables.	Comprehensive RMarkdown reports are provided for all types of raw data/input files. These reports clearly explain the content, structure, and variable definitions, making it easy for users to understand and use the data.	RMarkdown reports are created for most raw data/input files, but some sections lack detail or clarity in explaining the content and variable definitions.	RMarkdown reports are missing or lack sufficient information, making it difficult for users to grasp the content of the raw data or understand the variables.	
The RMarkdown reports are properly formatted, rendered as HTML or PDF files, and feature information in a variety of modes (e.g., running text, tables, or figures).	The RMarkdown reports are well-formatted and rendered as high-quality HTML or PDF files. They effectively use a mix of text, tables, and figures to convey information in a clear and engaging manner.	The RMarkdown reports are rendered as HTML or PDF files and but could include some more variety in presentation (e.g., text, tables, figures). However, usefulness of descriptives and formatting could be improved.	The RMarkdown reports are poorly formatted, lack variety in presentation, or are not properly rendered as HTML or PDF files.	
The rendered Markdown files are "publication-ready" - i.e., code that is not relevant to understanding the data or warning messages is hidden.	The rendered Markdown files are polished and publication-ready, with non-essential code and all warning messages effectively hidden. The presentation focuses solely on relevant data insights, enhancing readability.	The rendered Markdown files are mostly publication-ready, but some non-relevant code or occasional warning messages are still visible, slightly detracting from the overall presentation.	The rendered Markdown files are not publication-ready, containing unnecessary code and visible warning messages that clutter the document and distract from the main content.	
2.2 Data preparation (20%)				
The raw data has been prepared and cleaned, using a variety of common data operations in R, involving dplyr, tidyverse, or data. table.	The raw data has been fully prepared and cleaned. The team made extensive use of various data operations in R, showcasing a strong understanding of dplyr, tidyr, and other data manipulation tools. The data is now fully ready for analysis, with no outstanding issues.	The data is prepared and ready for analysis; however, the process and code could have been more efficient, and there are minor outstanding issues that should be addressed to ensure optimal performance in the future.	The raw data preparation and cleaning process were minimal and lacked thoroughness. Several key data operations were either missing or incomplete, and the data is not yet ready for analysis.	
Common operations are merging, aggregating, de-deduplication, reshaping, converting dates, or using regular expressions.	All necessary data operations such as merging, aggregating, de-duplication, reshaping, and converting dates have been executed efficiently. Regular expressions were also applied efficiently when needed, resulting in well-structured data.	Most common data operations, such as merging and reshaping, have been successfully completed. However, there were minor inefficiencies in the code, which could have been optimized further.	Basic data operations such as merging and reshaping were attempted but were incomplete or incorrect. Handling of de-duplication, converting dates, and regular expressions was either inefficient or missing, leading to data inconsistencies.	
Basic programming concepts are made use of appropriately to increase speed and minimize errors (e.g., looping, vectorization, writing functions, handling errors/debugging).	Excellent use of basic programming concepts, such as looping and vectorization, to optimize speed and minimize errors. Several useful functions and debugging techniques are used, ensuring a robust data processing pipeline.	Basic programming concepts were applied sufficiently, with some room for improvement in optimizing the code.	Programming concepts such as looping and vectorization were either misapplied or omitted entirely. The resulting code contains errors, and insufficient debugging.	
Additional variables are created from the raw data (feature engineering).	Several additional useful variables were created. These new features add valuable insights for futher analysis.	Limited additional variables were created from the raw data, and/or they could have been more thoughtfully engineered to add greater value to the analysis.	No additional variables were created from the raw data (or) the ones created have very limited added value for further data analysis.	
2.3. Analysis and deployment				
The analysis constitutes a substantial enrichment to the raw data. By using building blocks from the course site, for example, students can conduct regression analysis on the data. Other ways of enriching the data (e.g., text analysis using textblob, or any other material from the web) can also be incorporated.	The analysis significantly enriches the raw data by applying diverse and advance methods.	The analysis enriches the data by incorporating basic techniques.	The analysis provides minimal enrichment to the raw data.	
Results of the analysis are deployed/unlocked, either in the form of a "publication-ready" PDF document (think of it as a manuscript), or in the form of other ways of knowledge dissemination (e.g., an R package with an algorithm, or a Shiny app, see building blocks on the course site). The way of deployment is well aligned with the goal of the project.	The results are deployed effectively in a professional format, such as a publication-ready PDF manuscript, a well-designed R package, or an engaging Shiny app. The method of dissemination is highly aligned with the project's goals and ensures accessibility to the intended audience.	The results are presented in a clear and functional format, such as a PDF or another dissemination tool. While the deployment is aligned with the project goals, the presentation could benefit from further refinement or enhanced usability.	The deployment of results is not effectively aligned with the project goals. The chosen format may lack professionalism, clarity, or accessibility, making it challenging to communicate the findings to the intended audience.	
3. Source code and Automation				
3.1 Source code quality (15%)				
The source code is clearly readable (e.g., variable names that are meaningful), self-documenting, and well-structured (e.g., headers, sections).	The source code is highly readable, with clear and descriptive variable names that convey the purpose of each variable. The code is self-documenting with useful comments. It is well-organized with consistent formatting, logical sections, and appropriate headers.	The source code is reasonably readable, with variable names that are generally meaningful, although some areas may benefit from more clarity. While some external comments are necessary, the code is still fairly self-explanatory. The structure of the code is adequate, with identifiable sections and headers, though improvements could be made to further enhance organization and flow.	The source code lacks readability, with variable names that are unclear or generic, making it difficult to understand the purpose of each variable. The code is not self-documenting and lacks sufficient comments to convey its logic. The structure is weak, with inconsistent formatting, poorly defined sections, and missing or insufficient headers.	
The directory structure clearly reflects the pipeline stages (e.g., data-preparation, analysis, paper/app) of the project, and subdirectories for data components (e.g., gen, src, data, and temp, input, audit, output) have been used correctly.	The directory structure is highly organized and mirrors the project's pipeline stages perfectly. Subdirectories for data components are correctly and consistently used, making it easy to navigate and understand the workflow. Each folder is appropriately labeled and logically grouped, providing clear separation of tasks and data.	The directory structure is mostly organized and generally reflects the project's pipeline stages. Subdirectories for data components are present and appropriately used, though there may be minor inconsistencies. The structure is functional but could benefit from clearer organization or labeling in some areas.	The directory structure is disorganized or incomplete, with minimal reflection of the project's pipeline stages. Subdirectories for data components are either missing or used incorrectly, making it difficult to understand the workflow. The overall structure requires significant reorganization for clarity and proper task separation.	
The code runs in a linear fashion (top to bottom execution, without errors), and adheres to the DRY principles (for-loops and functions).	The code executes smoothly from top to bottom without any errors, following a clear and logical linear flow. DRY principles are well-implemented, with minimal repetition of code. Functions and for-loops are used appropriately to streamline the code, making it efficient and easy to maintain.	The code runs without errors in a generally linear fashion, though there may be occasional deviations in flow. DRY principles are somewhat applied, but there are areas with repeated code that could be refactored. Functions and for-loops are used, but improvements could be made to enhance code efficiency and clarity.	The code does not run smoothly, with errors or issues that interrupt the linear flow. There is a significant amount of repetitive code, showing little to no adherence to DRY principles. Functions and for-loops are underutilized, leading to inefficient code.	
3.2 Degree of automation (10%)				
Code chunks follow the input-transformation-output ("modular") structure, and are "stitched" together in a makefile that runs the entire project pipeline automatically after issuing the make command in the root of the repository	Code chunks are clearly modular, consistently following the input-transformation-output structure. They are well-separated and easy to understand. The makefile is comprehensive and correctly links the entire project pipeline, allowing for a seamless execution of the full process with the make command. The pipeline runs automatically and without issues from the root of the repository, demonstrating excellent organization and automation.	Code chunks generally follow the modular structure, though there may be occasional inconsistencies. The makefile is functional and stitches the code together reasonably well, enabling the project pipeline to run with the make command. However, there might be some minor issues or areas where the process could be further streamlined.	Code chunks do not clearly follow the modular structure, with weak separation between input, transformation, and output stages. The makefile is incomplete or ineffective, leading to difficulties in running the project pipeline automatically. The make command may not execute the pipeline properly, requiring significant improvements in both modularity and automation.	
All file paths are specified relative to the current script, no absolute paths are used.	All file paths are correctly specified as relative to the current script, ensuring portability across different environments.	Most file paths are specified as relative to the current script, though a few absolute paths may still exist. While the code is generally portable, some adjustments are needed to eliminate absolute paths or correct inaccurate relative paths for full automation.	Many file paths are specified as absolute, limiting the portability of the code. The use of relative paths is minimal or inconsistent, requiring significant revisions to ensure the code runs smoothly across different environments.	
The repository only tracks the version of files that need to be tracked (i.e., source code), and not others (e.g., generated files).	The repository is well-maintained, tracking only the necessary files, such as source code. Generated or temporary files are correctly excluded through the use of .gitignore or equivalent mechanisms. The version control is clean and focused, adhering to best practices.	The repository generally tracks the correct files, with most generated or unnecessary files excluded. However, some non-essential files may still be tracked, indicating that improvements could be made in managing exclusions (e.g., refining .gitignore).	The repository tracks many unnecessary files, such as generated or temporary files, cluttering the version control history. The .gitignore or exclusion settings are poorly implemented or missing, requiring significant improvements to focus on tracking only the essential files.	