Projekt i implementacja aplikacji do kodowania odpowiedzi z pytań otwartych w badaniach ankietowych

Spis treści

1. Wstęp.

Przedmiotem niniejszej pracy jest opis wykonanej – w ramach pracy dyplomowej - aplikacji webowej służącej do kodowania odpowiedzi z pytań otwartych (wykorzystywanych w badaniach społecznych lub badaniach rynku do uwzględnienia tych odpowiedzi respondentów, które nie zostały zdefiniowane przez badacza na etapie projektowania kwestionariusza wywiadu), a także przedstawienie jej projektu, w tym dalszych możliwych kierunków rozwoju.

Tłem dla zrealizowanego pomysłu była praca przy kodowaniu odpowiedzi respondentów z badań rynku, która odbywała się z użyciem arkuszy kalkulacyjnych i to właśnie narzędzie było głównym punktem odniesienia. Ma to swoje konsekwencje – po pierwsze, starano się zachować pewną spójność i zgodność, jeśli chodzi o strukturę plików czy możliwość wymiany danych między aplikacją a arkuszem kalkulacyjnym; po drugie, skoncentrowano się na usprawnieniu procesu ręcznego kodowania, a nie zastąpieniu go przez kodowanie automatyczne, szczególny nacisk kładąc na wyszukiwanie (filtrowanie) oraz zarządzanie kluczem kodowym.

Od strony technicznej, aplikacja została przygotowana z wykorzystaniem języka R, a w jego ramach (głównie) frameworku (pakietu) *shiny*, który stanowi zbiór funkcji tłumaczonych na HTML, CSS i JavaScript. R to jeden z najbardziej oczywistych wyborów w kontekście celów aplikacji, ze względu na jego przeznaczenie do pracy z danymi (zwłaszcza w formie tabelarycznej) oraz względnie dużą popularność w obszarze nauk społecznych [A]. Podobnie, *shiny* to najpopularniejszy pakiet (w ramach R) umożliwiający udostępnianie użytkownikom rozwiązań bez konieczności pisania przez nich kodu [B].

Struktura pracy dyplomowej, oprócz rozdziałów poświęconych wybranej technologii oraz obszaru problemowego, jakim jest kodowanie pytań z odpowiedzi otwartych, obejmuje przedstawienie założeń aplikacji, jej funkcji, opis bazy danych, testów oraz funkcji, których nie zaimplementowano, a które znacznie poprawiłyby skuteczność kodowania odpowiedzi i przyśpieszyłyby je. Wybór technologii oraz konstrukcja samej aplikacji (w tym również bazy danych) tworzona była również z myślą o tych pominiętych funkcjonalnościach, dlatego wątek ten jest dość istotny, przy czym ograniczono go głównie – w myśl przyjętych założeń, na których miała opierać się budowa aplikacji – do problematyki dalszego usprawnienia procesu wyszukiwania odpowiedzi, w których może znajdować się dany kod (tj. do wyszukiwania odpowiedzi, które są do siebie na tyle podobne, że w ramach przyjętego klucza kodowego, traktowane są jako tożsame). Problem ten sprowadza się, oczywiście, do określenia podobieństwa łańcuchów znaków i jako taki jest niezwykle szerokim, interdyscyplinarnym zagadnieniem, który funkcjonuje pod terminem przetwarzania języka naturalnego (NLP).

Pracę zamyka podsumowanie, które stara się zebrać najważniejsze z poruszonych wątków, przedstawiając je w możliwie najbardziej skondensowanej, spójnej postaci.

1. Założenia projektowe.
   1. Kodowanie odpowiedzi z pytań otwartych jako niezbędny element procesu badawczego.

Kodowanie to proces, w którym danym fragmentom tekstu (wypowiedzi) przyporządkowuje się kategorie. Kategorie to zazwyczaj liczby wraz z określoną etykietą [C]. Proces ten jest konieczny wtedy, gdy w badaniach ankietowych zastosowano pytania otwarte lub półotwarte, tj. takie, w których wypowiedź respondenta jest wpisywana w całości. Z dużym prawdopodobieństwem bowiem, każda taka wypowiedź strukturalnie (gramatycznie) lub semantycznie jest odmienna – respondenci w badaniach posługują się różnymi sformułowaniami (popełniają również błędy) i, dodatkowo, nie zawsze to samo rozumieją pod danym sformułowaniem. Trudno sobie zatem wyobrazić jakąkolwiek analizę (nie tylko ilościową), w której nie doprowadzono by surowych danych do postaci zbioru wspólnych kategorii (kodów właśnie). Dane w postaci zakodowanej pozwalają następnie na różnego rodzaju analizy (np. analizę częstości występowania danych odpowiedzi) i wizualizacje.

Największym wyzwaniem podczas kodowania jest niewątpliwie tworzenie kodów – istnieje napięcie między tym, na ile kod powinien być szczegółowy, a nie ile ogólny. Żadna z tych skrajności nie jest pożądana – zbyt duża szczegółowość doprowadzi do braku możliwości uchwycenia różnic między respondentami (różnice będą tak małe, że staną się statystycznie nieistotne), podobnie zbyt duża ogólność. Przyjmuje się, że kody powinny odpowiadać celom badawczym – postawionym pytaniom szczegółowym i hipotezom. Nie powinny być także w całości oparte o uprzednio stworzony klucz kodowy (tj. zbiór kodów wraz z etykietami) – to w skrajnym ujęciu doprowadziłoby do absurdalnej sytuacji, kiedy swobodna wypowiedź respondenta musi być zamknięta w z góry narzucone ramy skonstruowane przez badacza. Kodowanie wymagałoby zatem wiedzy na temat pytań szczegółowych i hipotez badawczych przy jednoczesnym oparciu klucza kodowego na rzeczywistej treści wypowiedzi [C].

Tak opisaną procedurę trudno zautomatyzować. Tym trudniej, im mniej ścisłe jest określenie, *co* badacz ma zamiar sprawdzić. I tak np. o wiele prostsze (choć nietrywialne) wydaje się – myśląc o automatyzacji kodowania – sprawdzenie, czy respondenci mają pozytywny, neutralny czy negatywny sentyment wobec Mazur niż określenie, jakie skojarzenia mają respondenci z Mazurami. Z tych powodów oprogramowania do tak rozumianego kodowania odpowiedzi z pytań otwartych opierają głównie na przeszukiwaniu danych tekstowych [D]. Taki też był główny cel wykonanej aplikacji – miała ona ułatwić (przyspieszyć) proces ręcznego kodowania, a nie go zastąpić.

* 1. Przegląd istniejącego oprogramowania dedykowanego do kodowania tekstu.

W podrozdziale tym głównie omówione zostaną dwa programy służące do kodowania tekstu – MAXQDA (CAQDAS) oraz RQDA. Ten pierwszy z racji swojej dominującej pozycji na rynku, a drugi jako oprogramowanie stworzone w języku R, a zatem w technologii wybranej w tej pracy do napisania aplikacji.

MAXQDA [E] należy do rodziny oprogramowania CAQDAS, które zostało stworzone do analizy danych jakościowych [D]. Jest to oprogramowanie płatne, umożliwiające – to jest niewątpliwie ważny czynnik wyróżniający to narzędzie – analizę nie tylko danych tekstowych, ale również danych w postaci audio i wideo (takie dane – ich wyodrębnione części – również można kodować). Z punktu widzenia niniejszej pracy najważniejsze jednak są funkcje odnoszące się do kodowania danych tekstowych. MAXQDA umożliwia tworzenie klucza kodowego, w tym grupowanie kodów w pewne ogólniejsze kategorie (z punktu widzenia analitycznego nie jest to konieczne, ale umożliwia łatwiejsze zarządzanie kluczem kodowym osobie kodującej), przekodowywanie, usuwanie kodów, tworzenie etykiet oraz dłuższych opisów przypominających, co kod oznacza (często, zwłaszcza po dłuższym czasie lub gdy kodowaniem zajmuje się więcej niż jedna osoba, sama etykieta jest niewystarczająca, ponieważ zazwyczaj jest krótka, przez co może być niejednoznaczna, gdy zapomni się lub nie zna się kontekstu, w którym została stworzona). Możliwe jest także otrzymanie raportu wizualnego i w formie tabeli, pokazującego rozkład użytych kodów – jest to przydatne zwłaszcza wtedy, gdy badacz nie chce, aby występowanie kodów było zbyt rzadkie – możliwe jest wtedy stworzenie ogólniejszej kategorii na podstawie istniejących już kodów, a więc zastosowanie jednej z form przekodowywania.

Proces kodowania odbywa się poprzez zaznaczenie fragmentu tekstu i wpisanie lub przeciągnięcie kodu. Możliwe jest także wyświetlenie chmury słów (nie muszą być to pojedyncze słowa, jeśli wcześniej określi się, ile słów współwystępujących wziąć pod uwagę) jako ilościowej reprezentacji materiału tekstowego – wybranie pojedynczego słowa spowoduje przeszukanie danych i wyświetlenie tych fragmentów, w których ono się znajduje (dla wybranych języków możliwe jest przeprowadzenie procesu lematyzacji, a wtedy program wyszukuje słowo nie dosłownie, ale również np. bierze pod uwagę odmianę przez przypadki czy osoby, ponieważ wyznaczony jest korpus słowa). Wreszcie, użytkownik może skorzystać także z kodowania automatycznego – zakodowane będą te fragmenty, w których dane słowo (lub kilka słów) występuje. Taki rodzaj kodowania automatycznego wymaga jednak przejrzenia kodów i sprawdzenia, czy – biorąc pod uwagę szerszy kontekst – kod został przypisany poprawnie. Jest to więc dość proste rozwiązanie i w literaturze nie poświęca się tej akurat funkcji wiele miejsca [E].

RQDA – w zakresie kodowania danych tekstowych – dzieli wiele funkcji z MAXQDA [F, G], będąc przy tym rozwiązaniem darmowym. Trzeba jednak już na początku powiedzieć, że projekt nie jest wspierany, a ostatnia aktualizacja pojawiła się w 2018 roku. I tak w RQDA możliwe jest tworzenie klucza kodowego, w tym przypisywanie kodów do ogólniejszych kategorii (liczba kategorii nadrzędnych jest jednak ograniczona względem MAXQDA), zarządzanie nim, a także dopisywanie dłuższych notatek odnoszących się do całego projektu czy pojedynczego kodu. Proces kodowania również wygląda podobnie – zaznacza się fragment tekstu i przyporządkowuje się dany kod. Możliwe jest zastosowanie kodowania automatycznego, ale wymaga to już pewnych umiejętności programistycznych (nie ma graficznego interfejsu użytkownika dla tej funkcji) oraz jest bardziej uproszczone niż w MAXQDA – wyszukuje jedynie wpisany fragment tekstu, nie dokonując lematyzacji. Z perspektywy użytkownika ważne może być także to, że instalacja RQDA jest nieco trudniejsza (oprogramowanie to nie zostało napisane jako aplikacja webowa) i nie jest to oprogramowanie tak atrakcyjne wizualnie, jak MAXQDA. Z drugiej strony, użytkownicy bardziej zaawansowani, w tym szczególnie użytkownicy znający język R, będą mogli korzystać z dużo bardziej zaawansowanych funkcji (wymagających pisania skryptu) i zewnętrznych bibliotek, np. do text miningu.

MAXQDA oraz RQDA to narzędzia mające o wiele szersze zastosowanie niż oprogramowanie tylko do kodowania odpowiedzi z pytań otwartych w badaniach ankietowych. Nawet w zakresie samego kodowania danych tekstowych mają szersze zastosowanie, ponieważ zostały stworzone z myślą o kodowaniu jakichkolwiek tekstów, tj. nie tylko odpowiedzi udzielonych w ankiecie, ale też np. tekstów publicystycznych, pamiętników, ustaw. I choć były punktem odniesienia w procesie tworzenia prezentowanej w tej pracy aplikacji, to jednak założenia projektowe w największym stopniu odnosiły się do alternatywnego sposobu kodowanie wobec wykorzystania arkuszy kalkulacyjnych.

* 1. Wymagania funkcjonalne i niefunkcjonalne.

Odniesieniem dla stworzonej aplikacji nie były jednak wymienione wyżej narzędzia, ale używanie arkuszy kalkulacyjnych do kodowania odpowiedzi z ankiet. Takie odpowiedzi, w przeciwieństwie do innych danych tekstowych, są krótsze i do każdej z nich przypisany jest identyfikator respondenta oraz identyfikator pytania. Przeszukiwanie odpowiedzi pod kątem danych słów polega na ręcznym wpisywaniu szukanego słowa przez badacza i filtrowaniu tych odpowiedzi, w których słowo się znalazło. Najważniejszym celem aplikacji było przyspieszenie przeszukiwania danych.

Wymagania funkcjonalne:

1. aplikacja webowa (dostęp przez przeglądarkę internetową);
2. import i eksport danych w określonym formacie i o określonej strukturze (pliki .xlsx zawierające kolumny: identyfikator respondenta, identyfikator pytania, treść odpowiedzi, kody oddzielane spacją);
3. tworzenie projektów, powrót do stanu, w którym przerwało się pracę i usuwanie projektów;
4. kodowanie i tworzenie planów kodowych dla wybranych identyfikatorów pytań naraz;
5. dodawanie nowych kodów, usuwanie kodów i przekodowywanie (zmiana kodów lub łączenie kodów) dla klucza kodowego, a także zmiany te powinny mieć odbicie w już zakodowanych danych, których dotyczą;
6. tworzenie i modyfikacja etykiet dla kodów;
7. wprowadzanie kodów dla danych odpowiedzi;
8. wyświetlanie chmury słów na podstawie danych tekstowych dla wybranych identyfikatorów pytań;
9. wyszukiwanie zaznaczonych słów w chmurze słów w danych do zakodowania;
10. usuwanie wybranych słów z chmury słów.

Wymagania niefunkcjonalne:

1. aplikacja działa na przeglądarkach Chrome, Firefox, Edge;
2. aplikacja działa na systemie Windows 10.
   1. Diagram przypadków użycia.

Diagram przypadków użycia pozwala zobrazować działanie systemu, abstrahując od szczegółów implementacyjnych, ukazuje aktorów (użytkowników) i funkcjonalności systemu przez nich używane [H]. Poniżej zaprezentowano diagram przypadków użycia dla trzech najważniejszych funkcji – zarządzania kluczem kodowym, kodowania oraz wyszukiwania konkretnego słowa.



*Rys. 2.1.: Diagram przypadków użycia.*

**Przypadek użycia:** Zarządzanie kluczem kodowym.

**Aktor:** Badacz.

**Opis:** Dodawanie nowego kodu i etykiety, modyfikowanie kodu, etykiety, usuwanie kodu i etykiety.

**Warunki wstępne:** Wybrany projekt oraz wybrane zmienne (pytania w kwestionariuszu).

**Przebieg:**

1. Badacz wybiera zakładkę z kluczem kodowym.
2. Wyświetla się tabela, w której można dodawać nowe wiersze z kolumnami odpowiadającymi kodom i etykietom, usuwać wiersze oraz modyfikować treść komórek.

**Przypadek użycia:** Przypisanie kodu do odpowiedzi.

**Aktor:** Badacz.

**Opis:** Dodawanie kodu do odpowiedzi, usuwanie kodu, zmienianie kodu.

**Warunki wstępne:** Wybrany projekt oraz wybrane zmienne (pytania w kwestionariuszu).

**Przebieg:**

1. Badacz wybiera zakładkę z odpowiedziami.
2. Wyświetla się tabela z kolumnami odpowiadającymi numerowi identyfikacyjnemu, treści odpowiedzi oraz kodom. Komórki dla kodów można edytować, tj. wpisywać w nie kod, zmieniać kod, usuwać kod.

**Przypadek użycia:** Wyszukanie słowa.

**Aktor:** Badacz.

**Opis:** Wyszukanie wybranego słowa we wszystkich odpowiedziach przynależnych do wybranych zmiennych w danym projekcie.

**Warunki wstępne:** Wybrany projekt, wybrane zmienne (pytania w kwestionariuszu) oraz wyświetlona chmura słów.

**Przebieg:**

1. Badacz wybiera zakładkę z odpowiedziami – obok wyświetla się chmura słów.
2. Pojedyncze kliknięcie na słowo w chmurze słów powoduje posortowanie tabeli z odpowiedziami tak, że na początku tabeli pojawiają się odpowiedzi, w których znajduje się wybrane słowo.
3. Implementacja i wdrożenie.
   1. Wybrana technologia.

Aplikacja została napisana w języku R [I] i stworzona jako aplikacja webowa dzięki wykorzystaniu pakietu *shiny* [J], przy czym korzystano także z innych rozszerzeń (pakietów) tego języka: *glue* [K], *shinyalert* [L], *DBI* [M], *stringi* [N], *dplyr* [O], *tibble* [P], *DT* [R], *purrr* [S], *wordcloud2* [T], *tidyr* [U], *bslib* [W], *RSQLite* [X].

R to język używany w obszarze eksploracji danych, analizy, uczenia maszynowego, wizualizacji, ale także raportowania i prezentacji wyników. Pakiet *shiny* powstał właśnie w tym ostatnim celu, stając się pierwszym wyborem dla użytkowników, którzy chcieliby tworzyć aplikacje webowe w R [Z]. Wyróżnia się zastosowaniem paradygmatu programowania reaktywnego – stan aplikacji jest stale monitorowany i jeśli nastąpi zmiana, która wymuszałaby ponowne przeprowadzanie obliczeń, żeby wyświetlić zaktualizowane wyniki (np. użytkownik nacisnął przycisk, wybrał inną wartość itp.), to te obliczenia są wykonywane automatycznie [Y]. Jest to rozwiązanie bardzo wygodne dla programisty, ale rodzi też niebezpieczeństwo przeciążenia systemu – zbyt daleko idąca reaktywność w najlepszym przypadku doprowadzi do niepotrzebnych obliczeń (np. użytkownik, testując aplikację, zmienia różne opcje, które są ze sobą powiązane, ale każda opcja uruchamia obliczenia na nowo, mimo że mógłby istnieć przycisk dedykowany ponownemu przeliczeniu po wprowadzeniu przez użytkownika wszystkich opcji), a w najgorszym sprawi, że korzystanie z aplikacji stanie się niemożliwe z powodu zbyt długiego czasu reakcji [AA].

Pakiet *shiny* oparty jest na HTML, CSS i JavaScript, umożliwiając jednak posługiwanie się wyłącznie składnią języka R jako wystarczającą umiejętnością do pisania nawet złożonych aplikacji. Można jednak dzięki temu mówić o tym, że możliwe jest połączenie zalet R i tych trzech wcześniej wymienionych technologii. Samo takie połączenie (tj. pisanie w czystym HTML, CSS lub JavaScripcie) nie jest jednak trywialne i w toku prac nad aplikacją nie skorzystano z tej możliwości. Niewątpliwie jednak, chociażby funkcjonalności wspominane na końcu niniejszej pracy wymagałyby zastosowania JavaScriptu.

Mimo że R bywa postrzegany jako język nieużywany w profesjonalnym tworzeniu oprogramowania, a bardziej jako język służący do interaktywnej pracy z danymi, należy uznać, że przeświadczenie to jest fałszywe – np. pakiet *shiny* pozwala na pisanie aplikacji z użyciem modularnego systemu oddzielonych od siebie, możliwych do ponownego użycia komponentów, sprawiając, że kod aplikacji jest dużo bardziej przejrzysty niż gdyby miał znajdować się w jednym pliku na przestrzeni kilkuset czy kilku tysięcy linii kodu [AA]. To był także jeden z powodów wyboru właśnie tej technologii.

* 1. Zastosowana baza danych.

Baza danych to jeden z najważniejszych elementów napisanej aplikacji – nie służy tylko do trwałego zapisu danych, ale odpowiada również za logikę biznesową poprzez odpowiednią strukturę i nałożone ograniczenia. Posłużono się bazą SQLite, która jest relacyjną bazą danych, łatwo przenoszoną pomiędzy różnymi systemami operacyjnymi czy programami, łatwo implementowaną [AB]. Schemat bazy danych wygląda następująco:



*Rys. 3.2.: Schemat bazy danych.*

W relacji *projects* odkładane są projekty tworzone przez użytkownika – usunięcie projektu oznacza jednocześnie usunięcie wszystkich danych powiązanych z tym projektem w pozostałych relacjach. W relacji *verbatims* znajdują się dane zaimportowane przez użytkownika oraz – dodatkowo – nadany identyfikator dla danego cytatu (odpowiedzi respondenta). Co prawda identyfikator taki nie jest konieczny do jednoznacznego zidentyfikowania odpowiedzi (do tego wystarczy id projektu, id respondenta oraz zmienna, ale nadanie identyfikatora uprościło operacje wykonywane w programie). W relacji *codes* odkładają się kody nadawane przez użytkownika – do danego stwierdzenia może być przypisanych wiele kodów i każdy kod odkłada się w osobnym wierszu. W relacji *code\_frame* znajduje się klucz kodowy – kod wraz z etykietą i informacją, do której zmiennej ma zastosowanie.

Szczególnie istotny jest związek relacji *codes* i *code\_frame* – kluczem obcym tej pierwszej jest klucz podstawowy (kompozytowy) tej drugiej, który składa się z *project\_id*, *variable*, *value*. Usunięcie jakiegokolwiek kodu z klucza kodowego sprawia, że kody te (wiersze) zostaną usunięte z relacji *codes* – podobnie będzie w przypadku zmiany kodu w kluczu kodowym. Nie może istnieć także więcej niż jeden taki sam kod w kluczu kodowym dla danych zmiennych, choć nie ma takiego ograniczenia między zmiennymi – powszechną praktyką jest stosowanie tych samych numerów (kodów) dla zmiennych (pytań w kwestionariuszu), które koduje się osobno.

* 1. Aplikacja – wygląd i funkcje.

Bibliografia

A: Robinson D., *The Impressive Growth of R*, Stack Overflow Blog, 10.11.2017, dostęp: <https://stackoverflow.blog/2017/10/10/impressive-growth-r/> (14.12.2021).

B: King R., *Shiny vs. Dash: A Side-by-side Comparison*, R. King Data Consulting, 6.03.2019, dostęp: https://www.rkingdc.com/blog/2019/3/6/shiny-vs-dash-a-side-by-side-comparison (14.12.2021).

C: Haczkowska A., *Analiza trudności związanych z kodowaniem odpowiedzi na pytania otwarte i próba ich rozstrzygnięcia z pomocą psychologii poznawczej na przykładzie badania skojarzeń internautów z regionami Polski*, „Nauki o Zarządzaniu. Management Sciences”, 2012, 11, 2, s. 55-73.

D: Bryda G., *CAQDAS a badania jakościowe w praktyce*, „Przegląd Socjologii Jakościowej”, 2014, 10, 2, s. 12-39.

E: Kuckartz U., Rädiker S., *Analyzing Qualitative Data with MAXQDA. Text, Audio, and Video*, Springer, 2019.

F: Chandra Y., Shang L., *Qualitative Research Using R: A Systematic Approach*, Springer, 2019.

G: Huang R., *RQDA: R-based qualitative data analysis.* R package version 0.2-8., dostęp: http://rqda.r-forge.r-project.org/ (15.12.2021).

H: Klimek R., Szwed P., *Formal analysis of use case diagrams*, „Computer Science”, 2010, 11, 1, s. 115-131.

I: R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, dostęp: https://www.R-project.org/ (22.12.2021).

J: Chang, W. et al., *shiny: Web Application Framework for R*, R package version 1.7.1., dostęp: https://CRAN.R-project.org/package=shiny (22.12.2021).

K: Hester J., Bryan J., *glue: Interpreted String Literals,* R package version 1.5.1., dostęp: https://CRAN.R-project.org/package=glue (22.12.2021).

L: Attali D., Edwards T., *shinyalert: Easily Create Pretty Popup Messages (Modals) in ‘Shiny’*, R package version 2.0.0., dostęp: https://CRAN.R-project.org/package=shinyalert (22.12.2021).

M: R Special Interest Group on Databases (R-SIG-DB), Wickham H., Müller K., *DBI: R Database Interface*, R package version 1.1.1., dostęp: https://CRAN.R-project.org/package=DBI (22.12.2021).

N: Gągolewski M., *stringi: Fast and portable character string processing in R*, R package version 1.7.6., dostęp: https://stringi.gagolewski.com/ (22.12.2021).

O: Wickham H., Romain F., Henry L., Müller K., *dplyr: A Grammar of Data Manipulation*, R package version 1.0.7., dostęp: https://CRAN.R-project.org/package=dplyr (22.12.2021).

P: Müller K.., Wickham H., *tibble: Simple Data Frames*, R package version 3.1.6., dostęp: https://CRAN.R-project.org/package=tibble (22.12.2021).

R: Xie Y., Cheng J., Tan X., *DT: A Wrapper of the JavaScript Library 'DataTables'*, R package version 0.20., dostęp: https://CRAN.R-project.org/package=DT (22.12.2021).

S: Henry L., Wickham H., *purrr: Functional Programming Tools*, R package version 0.3.4., dostęp: https://CRAN.R-project.org/package=purrr (22.12.2021).

T: Lang D., *wordcloud2: Create Word Cloud by htmlWidget*, R package version 0.2.2., dostęp: https://github.com/lchiffon/wordcloud2 (22.12.2021).

U: Wickham H., *tidyr: Tidy Messy Data*, R package version 1.1.4., dostęp: https://CRAN.R-project.org/package=tidyr (22.12.2021).

W: Sievert C., Cheng J., *bslib: Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'*, R package version 0.3.1., dostęp: https://CRAN.R-project.org/package=bslib (22.12.2021).

X: Müller K., Wickham H. James D., Falcon S., RSQLite: SQLite Interface for R, R package version 2.2.9., dostęp: https://CRAN.R-project.org/package=RSQLite (22.12.2021).

Y: Wickham H., *Mastering Shiny. Build Interactive Apps, Reports & Dashboards Powered by R*, dostęp: https://mastering-shiny.org/ (22.12.2021).

Z: Coene J., *JavaScript for R*, 19.04.2021, dostęp: https://book.javascript-for-r.com/ (22.12.2021).

AA: Fay C., Rochette S., Guyader V., Girard C., *Engineering Production-Grade Shiny Apps*, dostęp: https://engineering-shiny.org/ (22.12.2021).

AB: Allen G., Owens M., *The Definitive Guide to SQLite*, Apress, 2010.