Spis treści

Wprowadzenie

Według szacunków International Data Corporation (IDC) do 2020 roku ilość cyfrowych danych wyniesie 44 zettabajtów (44 tryliony gigabajtów), natomiast w 2025 roku ilość danych wzrośnie do 175 zettabajtów. Z raportu IDC wynika, że ludzkość produkuje i magazynuje coraz większe ilości danych, co wiąże się z trudnością przetwarzania coraz większych zbiorów danych oraz wymusza poszukiwanie nowych rozwiązań w dziedzinie analizy danych [1,2].

Dane same w sobie to tylko forma reprezentacji informacji, w związku z czym wydobycie wiedzy z ich zbioru jest możliwe dopiero po przeprowadzeniu niezbędnej analizy. Wiedza wydobyta ze zbioru danych może służyć do opisu procesów i stworzenia modeli otaczających nas zjawisk. Modeli opisujące przyrodę przyczyniły się do rozwoju nauk przyrodniczych oraz matematyki. W społeczeństwie podobnie jak w przyrodzie występują zjawiska i procesy które kształtują nasze otoczenie. Zrozumienie tych zjawisk jest niezwykle istotne dla takich nauk takich jak socjologia i ekonomia. Procesy społeczne możemy opisywać za pomocą badań opinii publicznej, w których odpowiedzi na nurtujące pytania zbierane są za pomocą ankiet. Badania ankietowane to zbiór odpowiedzi respondentów wykorzystując wystandaryzowany kwestionariusz. W badaniu możemy zebrać informację na temat demografii, opinii społecznej lub poziomu zadowolenia z usług a także dane wspomagające opis zjawisk występujących w społeczeństwie. [3].

Motywacją do napisania niniejszej pracy była potrzeba stworzenia narzędzia wspomagającego analizę danych ankietowych. Obecnie dostępne na rynku narzędzia umożliwiają tworzenie i analizę ankiety wewnątrz danego systemu (np. Webankieta, Surveymonkey, Google forms). Aplikacje te pozwalają na tworzenie złożonych ankiet jednak nie dostarczają rozbudowanych funkcji statystycznych. Brakuje aplikacji do analizy danych ankietowych która byłaby niezależna od innych systemów. Umożliwiłoby to badaczom analizę ankiet pochodzących z różnych źródeł przy pomocy dedykowanego narzędzia.

1. Cel i założenia pracy

Celem pracy jest stworzenie aplikacji webowej, która wspomaga analizę danych ankietowych.

Głównym zadaniem było wykonanie aplikacji, która wspomaga analizę danych ankietowych poprzez wizualizację danych i różnego rodzaju metody statystyczne. Dodatkowo powinna być to aplikacja webowa napisana w języku *R*, która umożliwiłaby wprowadzenie danych ankietowych jako pliku i doboru parametrów dla formatu CSV. Użytkownik miałby możliwość wizualizacji wyników ankiet poprzez tworzenie różnego rodzaju wykresów dla wybranych danych. Aplikacja powinna posiadać implementacje metod statystycznych wspomagających analizę ankiet.

Dodatkowym celem pracy było zbadanie możliwości i zastosowania pakietu *Shiny* w tworzeniu interaktywnych aplikacji webowych w języku *R*. *Shiny* to nowy pakiet *R*, który pozwala budować interaktywne aplikacje internetowe. Pakiet zyskuje na popularności i jest ciągle rozwijany przez społeczność open source, jednak wciąż brakuje publikacji na temat wykorzystania pakietu *R* w złożonej aplikacji webowej. W wersji 0.13.0 zaimplementowano możliwość dzielenia aplikacji na moduły w postaci View-Controller, brakuje jednak opisu wzorców i zaleceń dotyczących zarządzania rosnącą złożonością kodu aplikacji *Shiny*. W poniższej pracy zawarto wnioski dotyczące wykorzystania pakietu *Shiny* i *R* w aplikacjach internetowych. Opisano również użyte praktyki w tworzeniu komponentów aplikacji wykorzystując moduły i wzorce [4,5].

2. Teoria

2.1 Język R

R jest językiem programowania oraz wolnym (otwartym i darmowym) zaawansowanym środowiskiem nastawionym na analizę i eksplorację danych. Oprogramowanie objęte jest licencją GNU GPL (ang. *General Public License*) [6].

Środowisko *R* zostało utworzone w 1997 roku przez Ross Ihaka i Roberta Gentlemana na Uniwersytecie w Auckland w Nowej Zelandii. Jest alternatywą dla pakietu *S-PLUS* zaprojektowanego w laboratoriach przez zespół prowadzony przez Johna Chambersa. *R* był inspirowany środowiskiem *S*, jądro *R* składa się z napisanej w *C* i *Fortran* implementacji znanego od 1976 roku języka *S*. Język jest dynamicznie rozwijany, kolejne wydania pojawiają się kilka razy w roku uwzględniając nowe funkcje i poprawki błędów [6,7].

Warto zauważyć że język *R* jest składnikiem większego ekosystemu, w którego skład wchodzą pakiety rozszerzające możliwości bazowe języka. Pakiety znajdują się w repozytorium *CRAN* (*Comprehensive R Archive Network*), warto wspomnieć, że na początku 2019 roku w repozytorium dostępnych było ponad 13600 pakietów tworzonych i rozwijanych przez społeczność *R* [8].

2.2 Shiny

Shiny to pakiet języka *R* oraz framework pozwalający budować aplikacje internetowe. Pakiet rozszerza możliwości standardowego środowiska *R* i umożliwia budowę aplikacji typu klient - serwer. Pakiet zaprezentowano po raz pierwszy w wersji beta na konferencji JSM w 2012 roku w San Diego [9].

Oprogramowanie rozwijane jest przez RStudio oraz zespół pod przewodnictwem Winstona Changa, dzięki ich pracy pakiet ulega ciągłym modyfikacjom, a ostatnie wydanie pochodzi z 2 listopada 2018 roku - wersja 1.2.0 [10]. *Shiny* pozwala na budowę interaktywnych aplikacji internetowych przy pomocy wyrażeń reaktywnych, które tworzą powiązanie między wejściem a wyjściem. Funkcjonalność ta realizowana jest za pomocą *WebSocket* - jest to technologia, która zapewnia dwukierunkowy kanał komunikacyjny. Proces *R* uruchomiony po stronie serwera renderuje odpowiedni widok w przeglądarce po stronie użytkownika. Program *Shiny* domyślnie używa surowego *WebSocket*, korzystanie z paczki *Shiny Server* rozszerza mechanizm i umożliwia korzystanie z *SockJS*. Platforma *Shiny Server* umożliwia zarządzanie procesami *R* z różnymi aplikacjami *Shiny* na różnych adresach URL i portach [11].

2.3 Dane ankietowe

Ankiety to specyficzna forma wywiadu i narzędzie badawcze służąca zebraniu dużej liczby informacji na temat zjawisk występujących w społeczeństwie [3]. Dane zbierane są za pomocą kwestionariusza z pytaniami, na które odpowiadają respondenci. Istnieje kilka rodzajów pytań w badaniach ankietowych [12]:

- otwarte - pozwalają respondentowi na swobodną wypowiedź;
- półotwarte - respondent ma możliwość wybrania odpowiedzi zdefiniowanej
  w kwestionariuszu lub udzieleniu własnej odpowiedzi;
- zamknięte - rodzaj pytań z ograniczoną pulą odpowiedzi gdzie respondent ma możliwość wyboru. Wyróżniamy tutaj następujące typy odpowiedzi [13]:
  - koniunktywny – pozwala na wybranie kilku możliwych odpowiedzi,
  - dysjunktywny – pozwala na wybranie tylko jednej odpowiedzi,
  - alternatywny – pozwala na wybranie jednej z dwóch możliwości np. TAK/NIE.

W niniejszej pracy skupiono się na analizie danych ankietowych. Poznanie ich charakterystyki jest niezwykle istotne pod względem dostarczenia użytkownikowi odpowiedniego zestawu narzędzi do analizy tego zbioru danych.

3. Implementacja

3.1 Wymagania

Do korzystania z aplikacji niezbędne jest połączenie z internetem. Użytkownik za pomocą przeglądarki internetowej łączy się z aplikacją dostępną pod adresem https://dyzio18.shinyapps.io/survey-app/. Program jest aplikacją internetową, w związku z tym jest niezależną od systemu operacyjnego i może być używana na wielu platformach. W procesie deweloperskim wymagana jest instalacja *R* w wersji 3.0.2 lub wyższej. Dodatkowo należy zainstalować zależności projektu czyli zewnętrzne pakiety z których korzysta aplikacja.

3.2 Narzędzia programistyczne

Zgodnie z założeniami pracy projekt został zaimplementowany w języku *R*. Kolejnym założeniem była dostępność aplikacji na różnych typach urządzeń niezależnie od systemu operacyjnego. W tym celu wykorzystano pakiet *Shiny* oraz pakiet *Shinydashboard*. Narzędzia te pozwalają na implementację aplikacji internetowej wraz z zapewnieniem responsywności. Aplikacje można obsługiwać na urządzeniach o różnych rozdzielczościach, takich jak telefony komórkowe, tablety oraz PC.

Ze względu na szeroki zestaw funkcji jaką oferuje aplikacja, użyto następujących pakietów *R*:

- *cluster*,
- *dplyr*,
- *dt*,
- *ggplot2*,
- *shiny,*
- *shiny.i18n*,
- *shinydashboard*,
- *stringr*,
- *tidyr*,
- *tidytext*,
- *wordcloud.*

Łącznie w projekcie występuje 70 zależności, ilość ta jest adekwatna biorąc pod uwagę złożoność aplikacji oraz to, że część pakietów wymaga własnych zależności.

W procesie tworzenia oprogramowania wykorzystano edytor tekstowy *Visual Studio Code* z wtyczkami wspomagającymi pracę z *R* oraz system kontroli wersji *git*, wdrożono również automatyzację procesu wdrożenia aplikacji na środowisko produkcyjne korzystając z *Shinyapps.io* i pakietu *Rsconnect*.

3.3 Struktura projektu

Struktura programu została podzielona na część klienta oraz serwera. Aplikacja składa się z niezależnych komponentów do których przekazywane są dane z modułu odpowiadającego za wczytanie danych. Podział ten ułatwia rozbudowę aplikacji oraz niweluje zależności między poszczególnymi komponentami do minimum. Każdy komponent posiada w pełni określoną funkcjonalność i jest niezależny od pozostałych zgodnie z zasadą pojedynczej odpowiedzialności. Zgodnie z zaleceniami twórców, w głównym katalogu projektu umieszczono następujące pliki:

- *app.R* - jest to główny plik inicjalizujący aplikację, łączy obiekty serwera oraz interfejsu użytkownika zwanego dalej UI (ang. *user interface*). Obiekty te przekazywane są do funkcji *shinyApp* (Lis. 1).

```
source('./ui.R')
source('./server.R')
shinyApp(ui, server)
```

Listing 1: Wywołanie funkcji *shinyApp* w *app.R*

- *ui.R* - w pliku tym zdefiniowano obiekt UI, który zawiera elementy widoków takie jak nagłówek, panel boczny oraz główny panel aplikacji.
- s*erver.R* - w pliku tym znajduje się implementacja funkcji, w której wczytywane są moduły i obsługiwane są żądania po stronie serwera.
- *global.R* - w pliku tym umieszczane zostały obiekty globalne w tym obiekt i18n, który odpowiada za wielojęzyczność aplikacji.

W projekcie wydzielono również następujące katalogi (Lis.2) :

- *ui* - obiekty poszczególnych widoków
- *modules* - pojedyncze komponenty aplikacji wczytywane jako moduły po stronie serwera i klienta
- *packrat* - katalog systemu zarządzania zależnościami
- *i18n* - umieszczono tutaj plik *translation.json,* który pełni rolę słownika
- *rsconnect* - w katalogu tym umieszczone są meta-dane odpowiadające za wdrożenie aplikacji na środowisko produkcyjne
- *www* - jest to katalog w którym umieszczane są arkusze stylów CSS, skrypty JavaScript oraz statyczne pliki

```
|   .gitignore
|   app.R
|   functions.R
|   global.R
|   README.md
|   server.R
|   ui.R
|
+---i18n
|       translation.json
+---modules
+---packrat
+---rsconnect
+---ui
|   |
|   \---panels
|
\---www
    +---css
    +---img
    +---js
    \---resources
```

Listing 2: Struktura projektu z wyróżnieniem głównych plików i katalogów

3.4 Moduły w aplikacji Shiny

Ponieważ aplikacje Shiny stają się coraz bardziej rozbudowane i skomplikowane, autorzy pakietu wprowadzili w wersji 0.13.0 *Shiny* możliwość dzielenia kodu aplikacji na moduły [4]. Przed wprowadzeniem tego rozwiązania autorzy zalecają używanie funkcji, które zwracały odpowiednie obiekty UI, lub wykonywały akcje po stronie serwera. Napotkano jednak na problem związany z przestrzenią nazw, każdy identyfikator elementu UI musi być unikalny, gdyż Shiny korzysta z globalnej przestrzeni nazw. Po stronie serwera i klienta framework odwołuje się do identyfikatora, który przypisany jest do globalnych obiektów wejścia i wyjścia.

W informatyce tradycyjnym rozwiązaniem problemu kolizji nazw są przestrzenie nazw (ang. *namespace*). Dopóki nazwy są unikalne w obrębie przestrzeni nazw, a żadne dwie przestrzenie nazw nie mają tej samej nazwy, wówczas każda kombinacja przestrzeni nazw jest unikalna. Wiele systemów pozwala zagnieżdżać przestrzenie nazw, nie potrzebują więc one identyfikatora, który jest globalnie unikalny, wystarczające jest to, że jest unikalna w macierzystej przestrzeni nazw [5].

Budowa modułu w Shiny składa się z dwóch funkcji. Jedna zawiera fragment interfejsu użytkownika a druga fragment logiki po stronie serwera.

Logika odpowiadająca za renderowanie interfejsu użytkownika zawarta jest w funkcji *clusteringModuleUI,* której parametrem wywołania jest unikatowy identyfikator (Lis. 3).

```
###################
# UI
###################
clusteringModuleUI <- function(id) {
    ns <- NS(id)
    fluidRow(
        h2(class="panel__title", i18n$t("Clustering")),
        box( width = 12,
            column(12,uiOutput(ns("setup")))
        ),
        plotOutput(ns("clusteringPlot"))
    )
}
```

Listing 3: Funkcja renderująca UI w pliku *clusteringModule.R*

Cała logika komponentu została zawarta w funkcji, która po stronie serwera przypisana jest do *clusteringModule* (Lis. 5). Funkcja modułu po stronie serwera przyjmuje obowiązkowo 3 argumenty takie jak obiekty wejścia i wyjścia oraz obiekt sesji, przez który przypisana zostanie przestrzeń nazw. Funkcja może przyjmować dodatkowe argumenty, w tym przypadku ramkę danych z danymi ankietowanymi.

Fragment modułu odpowiedzialny za UI wywoływany jest po stronie klienta w sposób przedstawiony na listingu (Lis. 4).

```
source('./modules/clusteringModule.R')
clusteringPanelUI <- tabItem(
    tabName = "clustering",
    clusteringModuleUI("clustering")
)
```

Listing 4: Wywołanie modułu po stronie klienta, plik *clusteringPanelUI.R*

```
###################
# Server
###################
clusteringModule <- function(input, output, session, data) {
    ns <- session$ns
       output$setup <- renderUI({
              column(6,
                     selectInput(
                            ns("columnY"),  "Choose a column Y:",
                            choices = colnames(data)
                     ),
                     selectInput(
                            ns("columnX"), "Choose a column X:",
                            choices = colnames(data)
                     ),
                     numericInput(ns("clustersNumber"), 'Cluster count', 3,
                            min = 1, max = 9
                     ),
                     actionButton(
                            inputId = ns("submit"), label = "Submit"
                     )
              )
       })

       clusteringData <- eventReactive( input$submit, {
        cleanData <- data[data==""]<-NA
              cleanData <- data[, c(input$columnY, input$columnX )] %>%
            filter(complete.cases(.)) %>% na.omit
       })

       clusters <- reactive({
              kmeans(clusteringData(), input$clustersNumber)
       })

       output$clusteringPlot <- renderPlot({
              palette <- c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999")

        dataClustered <- data.frame(
            clusteringData(),
            cluster=factor(clusters()$cluster)
        )
        names(dataClustered) <- c("Y_col", "X_col", "cluster")

        ggplot(dataClustered)+
        geom_point(
            aes(dataClustered[["X_col"]],dataClustered[["Y_col"]],
                col=dataClustered[["cluster"]], shape = dataClustered[["cluster"]]
            ), size=6
        ) + xlab(input$columnX) + ylab(input$columnY)
       })
}
```

Listing 5: Funkcja modułu po stronie serwera w pliku *clusteringModule.R*

Moduł po stronie serwera wywoływany jest za pomocą funkcji *callModule,* która jako pierwszy argument przyjmuje funkcje modułu, następnie identyfikator, który powiąże instancje modułu z UI. Do funkcji jako ostatni parametr przekazano wyrażenie reaktywne, które zawiera ramkę danych z wczytanego pliku. Dodatkowo wywołanie opakowano w obserwator nasłuchujący zmian wyrażenia reaktywnego *df*, moduł zostanie zaktualizowany jeśli dane ulegną zmianie (Lis.6).

```
observe({
        df()
        callModule(
                clusteringModule,
                "clustering",
                df()
        )
})
```

Listing 6: Wywołanie modułu po stronie serwera

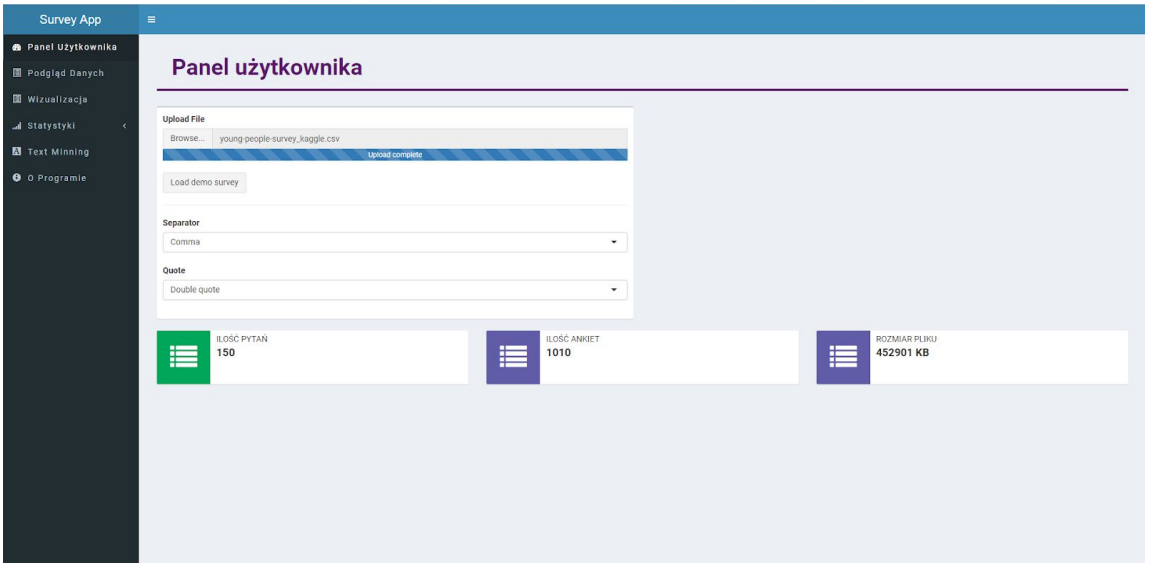
3.5 Interfejs użytkownika

Interfejs użytkownika został podzielony ze względu na możliwość realizowanych funkcji przez odbiorcę. W panelu bocznym użytkownik może przełączać się między panelami i jest on prowadzony ścieżką, która pozwala mu przeanalizować odpowiedni zbiór danych. Żeby korzystać z aplikacji nie jest potrzebne konto, dane użytkownika trzymane są w obiekcie sesji. W przyszłości planowana jest rozbudowa programu o konta użytkowników oraz moduł administratora. Widok aplikacji jest w pełni responsywny i dostosowuje się do rozdzielczości urządzenia. Program został podzielony na 6 widoków zwanych dalej panelami. Wyróżniono następujące panele:

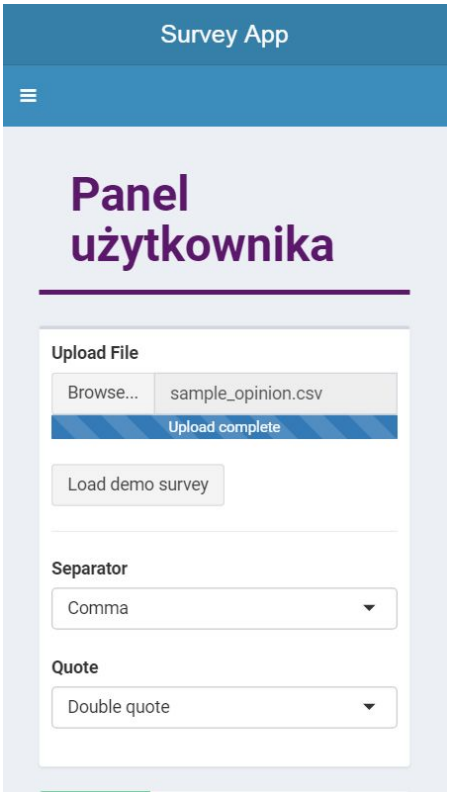
- Panel wczytania danych,
- Panel podglądu danych,
- Panel wizualizacji,
- Panel statystyk,
- Panel analizy języka,
- Panel ze szczegółami dotyczącymi aplikacji.

35.1 Panel wczytania danych

W tym komponencie użytkownik ma możliwość wczytania danych w postaci pliku CSV oraz konfigurację parametrów wczytania zależnie od formatu pliku. Po wczytaniu danych użytkownikowi wyświetlana jest ilość pytań, odpowiedzi oraz informacje na temat pliku (Rys. 1a, Rys. 1b).



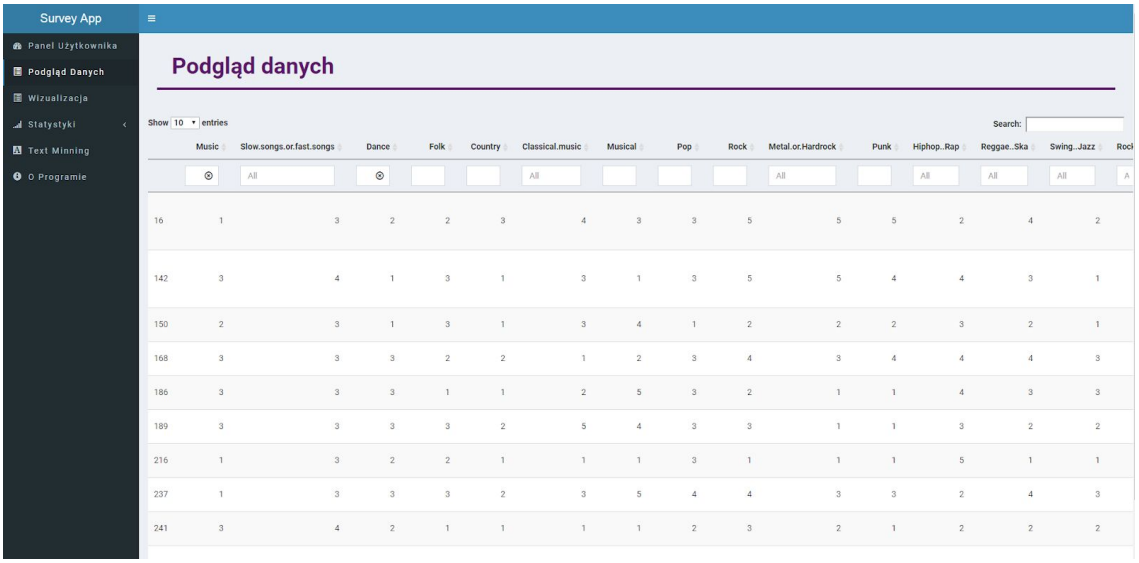
Rysunek 1a: Widok panelu wczytania danych w rozdzielczości 1920 px



Rysunek 1b: Widok panelu wczytania danych w rozdzielczości mobile

3.5.2 Panel podglądu danych

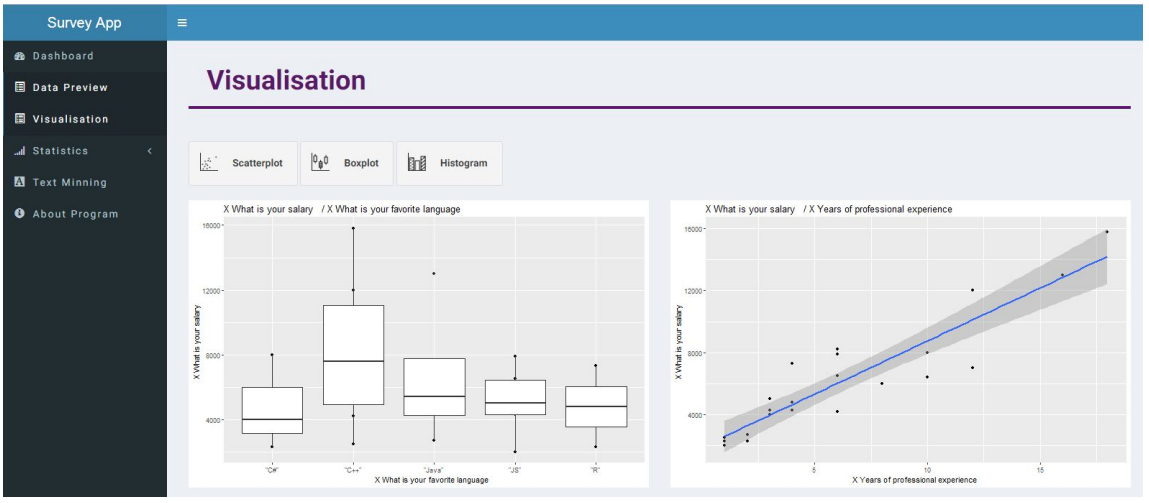
W panelu tym użytkownik ma możliwość podglądu wczytanych danych, filtracji oraz wyszukiwania interesujących go informacji w zbiorze.



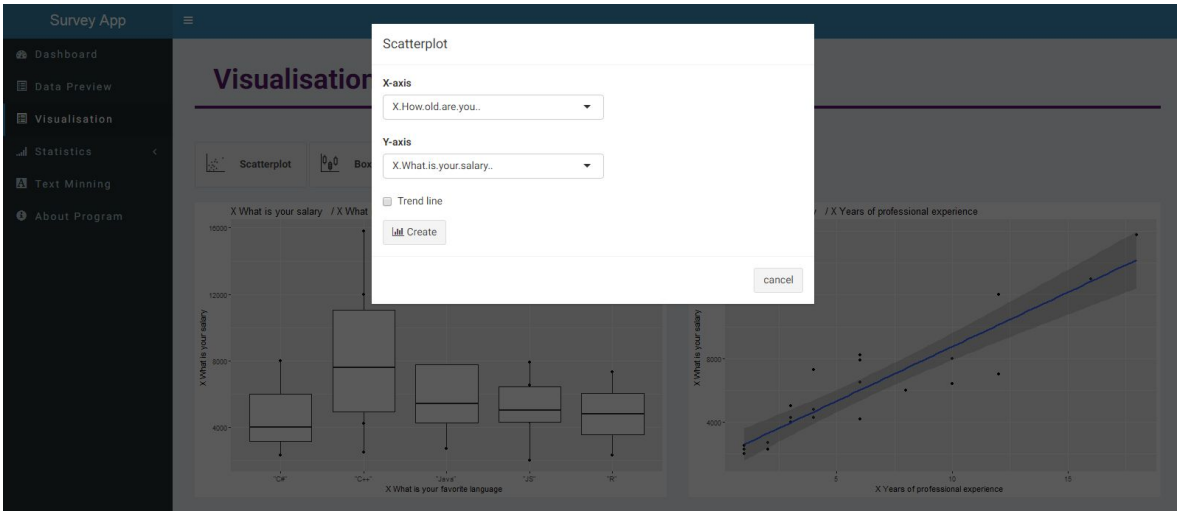
Rysunek 2: Widok panelu podglądu danych w rozdzielczości 1920 px

3.5.3 Panel wizualizacji danych

W tym komponencie użytkownik może generować wykresy w celu wizualizacji danych. Na początku wybierany jest rodzaj wykresu, następnie użytkownik dobiera dane i dodatkowe parametry wykresu. W module wykorzystano pakiet *ggplot2* [14]. Użytkownik ma możliwość wykonania wykresu rozrzutu, ramka-wąsy oraz histogramu (Rys. 2a, Rys. 2b).



Rysunek 2a: Widok panelu wizualizacji wraz z przykładowymi wykresami



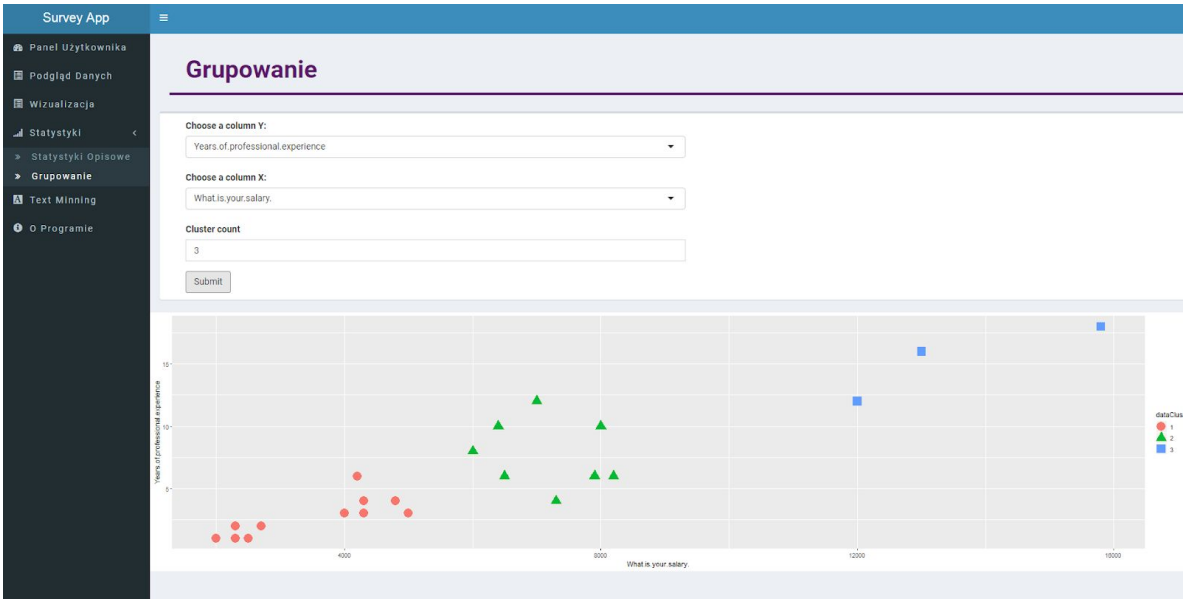
Rysunek 2b: Widok panelu wizualizacji i okno kreacji wykresu

3.5.4 Panel statystyk

Po kliknięciu w zakładkę statystyk rozwinięty zostanie pasek z listą dostępnych metod statystycznych, zaimplementowano następujące metody:

- Statystyki opisowe,
- Klasteryzacja k-średnich,
- Badanie korelacji.

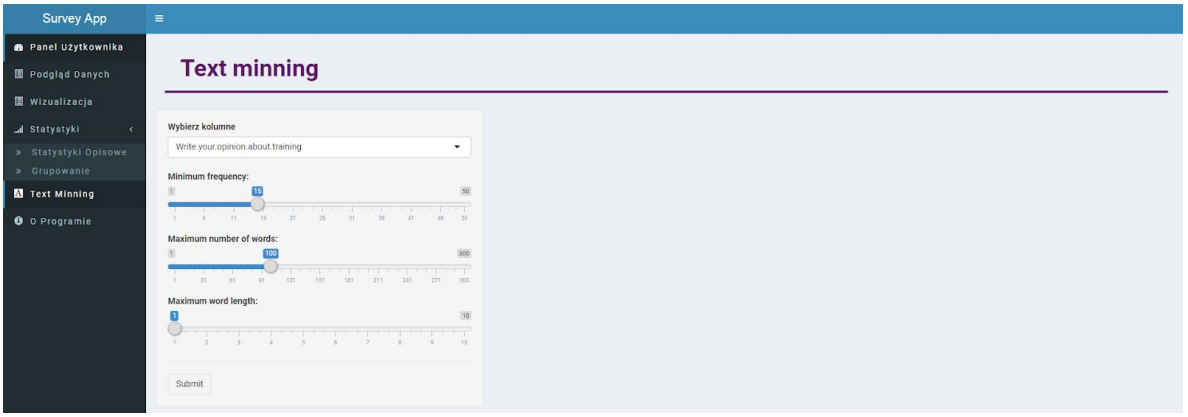
Przykładowy widok okna panelu statystyk znajduje się na rysunku (Rys. 3).



Rysunek 3: Widok panelu klasteryzacji w zakładce statystyki

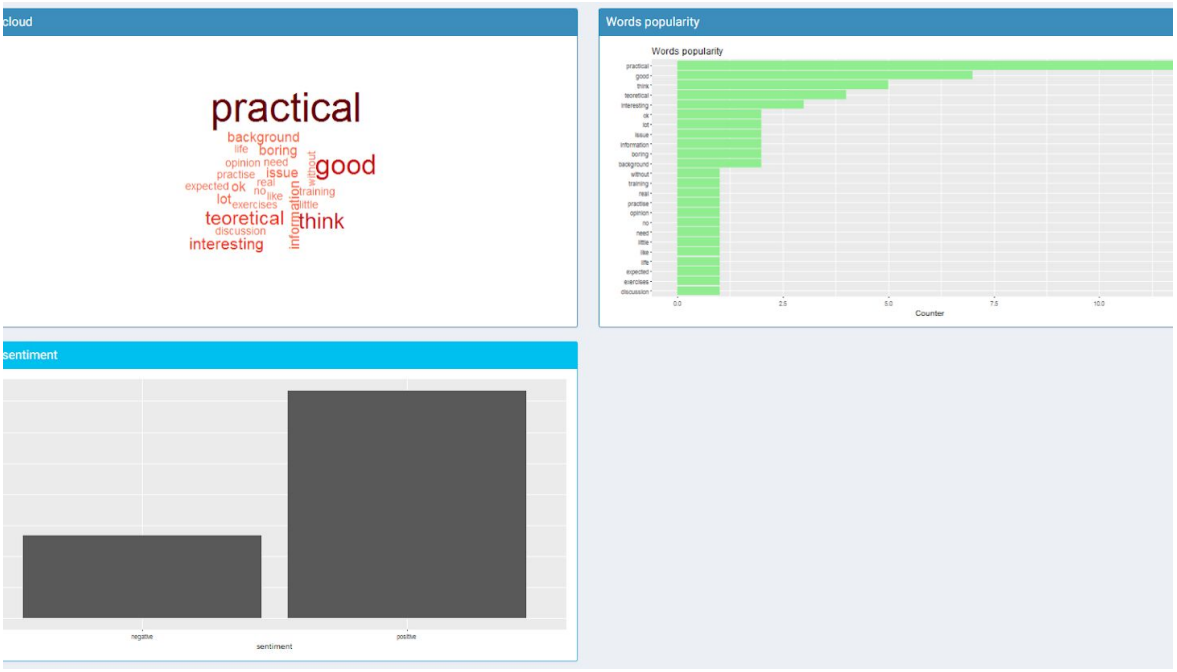
3.5.5 Panel analizy języka

Ważnym elementem ankiet są pytania otwarte, niestety są one stosunkowo problematyczne, gdyż trudniej jest przeprowadzić ich analizę. W aplikacji zaimplementowano metody text miningu, pozwalające analizować pytania otwarte w ankiecie za pomocą wizualizacji i badania sentymentu danych słów. Użytkownik ma możliwość wybrania odpowiedniej kolumny oraz dodatkowych parametrów (Rys. 4a).



Rysunek 4a: Widok panelu text miningu w rozdzielczości 1920 px

Wynikiem analizy jest chmura najczęściej używanych słów, wykres przedstawiający wystąpienia odpowiednich fraz oraz badanie sentymentu (*sentiment analysis*) słów w ankiecie (Rys 4b).



Rysunek 4a: Widok wyników analizy w panelu text miningu

Analiza sentymentu odbywa się za pomocą połączenia występujących słów oraz słownika z opisem czy dane słowo ma pozytywny czy negatywny charakter. Skorzystano tutaj ze słowników dostępnych w pakiecie *tidytext* [15]. Implementacja metody renderującej wykres znajduje listingu (Lis. 7)

```
text <- reactive({
    textRaw <- data[[selectedColumn()]] %>% toString() %>% tolower()
    textRaw <- gsub("[^0-9A-Za-z///' ]","" , textRaw ,ignore.case = TRUE)
    textRaw <- strsplit(textRaw, ",| ")
    textRaw <- data.frame(lapply(textRaw, as.character), stringsAsFactors=FALSE)
    colnames(textRaw) <- c("word")
    result <- textRaw %>% filter(!word %in% stopWords) %>% filter(nchar(word) >=
input$minLengthWord)
})

textCount <- reactive({
    result <- text() %>% count(word, sort = TRUE)
})

output$wordSentiment <- renderPlot({
    textSentiments <- textCount() %>%
        inner_join(get_sentiments("bing"))

    ggplot(textSentiments, aes(sentiment, n)) +
        geom_col(show.legend = FALSE)
})
```

Listing 7: Renderowanie wykresu sentymentu wraz z filtracją tekstu

4. Podsumowanie i wnioski

Cel pracy, którym było zaimplementowanie aplikacji do analizy danych ankietowych został osiągnięty. Wdrożono szereg funkcji wspomagających badanie, wizualizację i analizę danych, a aplikacja w przejrzysty i intuicyjny sposób pozwala na analizę odpowiedzi respondentów. Podczas implementacji aplikacji sprawdzono możliwości pakietu *Shiny* w zastosowaniu do budowy interaktywnych i rozbudowanych aplikacji.

Pakiet *Shiny* oraz *R* bardzo dobrze nadają się do prototypowania aplikacji nastawionych na analizę i wizualizację danych oraz wspierają *rapid application development (RAD)*. Shiny dostarcza wiele gotowych komponentów oraz rozwiązań, które przyśpieszają proces tworzenia aplikacji. Najciekawszą rzeczą w pakiecie *Shiny* jest przepływ danych, który nastawiony jest na tworzenie interaktywnych aplikacji i manipulacji danymi. Całość oparta jest o wyrażenia i wartości reaktywne aktywujące się podczas zmiany, powodując aktualizacje elementów, które się do nich odwołują.

Niezwykle ważna jest jednak znajomość ograniczeń i przemyślana architektura aplikacji, gdyż drobne zmiany w przepływie danych mogą powodować trudne do

wykrycia błędy. W środowisku *R* wciąż brakuje narzędzi deweloperskich wspomagających rozwój aplikacji. Obecnie najpopularniejszym IDE jest *RStudio* jednak brakuje w nim wtyczek wspomagających pracę ze złożoną aplikacją.

Podczas pisania pracy korzystano z *VSC*, który sprawdza się lepiej w zarządzaniu projektem o złożonej strukturze, napotkano również na problemy związane z debugowaniem kodu, komunikaty o błędach były bardzo skromne i często nie wskazywały poprawnego miejsca błędu. Podczas debugowania kodu za pomocą funkcji *browser()* nie można było podejrzeć wartości wyrażeń reaktywnych.

Dokumentacja pakietu *Shiny* oparta jest na prostych przykładach i brakuje opisu bardziej złożonego przepływu danych, lub wykorzystania dobrych praktyk i wzorców w większej aplikacji. Poza oficjalną dokumentacją ciężko znaleźć inne źródła wiedzy na temat pakietu *Shiny*, pakiet dopiero zyskuje na popularności wśród społeczności *R* i data science, wkrótce więc sytuacja w tej kwestii może się polepszyć.

Podsumowując, pakiet *Shiny* może stanowić doskonały interfejs dla aplikacji napisanych w języku *R*. Framework umożliwia tworzenie interaktywnych aplikacji internetowych, co może zainteresować środowisko biznesowe. Wzrost popularności dziedzin związanych z przetwarzaniem oraz analizą danych, a także możliwości języka *R* w tej dziedzinie mogą powodować intensywny rozwój frameworku w najbliższej przyszłości.

4.1 Dalsze możliwości rozwoju aplikacji

Projekt ma duże możliwości rozwoju i rozbudowy, aplikacja składa się z luźno powiązanych modułów, co daje możliwość dodawania nowych funkcji bez ingerencji w aktualny stan już działających. W przyszłości planowany jest dalszy rozwój aplikacji poprzez dodanie modułu administratora oraz umożliwienie użytkownikom logowanie się do aplikacji i zapisywanie ankiet na swoim koncie. Ciekawym pomysłem jest również możliwość analizy pytań otwartych, w związku z czym planowane są dalsze prace nad rozbudową tego modułu, a także dodanie słowników umożliwiających analizę w innych językach niż angielski.

Literatura

[1] Raport IDC z 2014 roku [@:] https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[2] Raport IDC z 2018 roku [@:] https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[3] Sztumski J.: Wstęp do metod i technik badań społecznych, 1995, s. 130-133

[4] Nota dla wydania pakietu Shiny w wersji 0.13.0 [@:] https://shiny.rstudio.com/reference/shiny/0.13.0/upgrade.html

[5] Artykuł na temat modułów w pakiecie Shiny [@:] https://shiny.rstudio.com/articles/modules.html

[6] Strona projektu R [@:] https://www.r-project.org/

[7] Gąglewski M.: Programowanie w języku R, 2016, s. 3-6

[8] Strona repozytorium CRAN [@:] https://cran.r-project.org/

[9] Smith D. [@:] https://blog.revolutionanalytics.com/2012/11/rstudio-releases-shiny.html

[10] Nota dla wydania pakietu Shiny w wersji 1.2.0 [@:] https://shiny.rstudio.com/reference/shiny/1.2.0/upgrade.html

[11] Dokumentacja techniczna pakietu Shiny [@:] https://shiny.rstudio.com

[12] Dutkiewicz W.: Podstawy metodologii badań, 2000, s. 85-86

[13] Nowak S.: Metodologia badań społecznych, 2007, s. 31-36

[14] Repozytorium pakietu ggplot2 [@:] https://github.com/tidyverse/ggplot2

[15] Repozytorium pakietu tidytext [@:] https://github.com/juliasilge/tidytext