KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

IBE entuzjaści edukacji

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Monitorowanie losów absolwentów uczelni
z wykorzystaniem danych administracyjnych
Zakładu Ubezpieczeń Społecznych

Dokumentacja techniczna

Pakietu do generowania raportów MLAK

Raport przygotowany przez Pracownię Ewaluacji Jakości Kształcenia na Uniwersytecie Warszawskim w ramach projektu systemowego Badanie jakości i efektywności edukacji oraz instytucjonalizacja zaplecza badawczego, współfinansowanego przez Unię Europejską ze środków Europejskiego Funduszu Społecznego, realizowanego przez Instytut Badań Edukacyjnych

Warszawa, 21 Kwietnia 2015

Autorzy:
dr Mikołaj Jasiński
Marek Bożykowski
Mateusz Żółtak
dr Marek Styczeń

Wydawca:
Instytut Badań Edukacyjnych
ul. Górczewska 8
01-180 Warszawa
tel. (22) 241 71 00; www.ibe.edu.pl

Publikacja opracowana w ramach projektu systemowego „Badanie jakości i efektywności edukacji oraz instytucjonalizacja zaplecza badawczego”, współfinansowanego przez Unię Europejską ze środków Europejskiego Funduszu Społecznego, realizowanego przez Instytut Badań Edukacyjnych.

Spis Treści

1. Wprowadzenie

W poniższym opracowaniu przedstawiamy zarówno zagadnienia techniczne oprogramowania służącego przetwarzaniu informacji z rejestrów wielu uczelni i ZUS oraz udostępnianiu wyników w postaci raportów automatycznych, jak i zasadnicze wnioski metodologiczne pozwalające na zrozumienie założeń i rozwiązań funkcjonalnych opracowanego narzędzia. Załączone raporty automatyczne prezentują typowe zastosowanie przygotowanego narzędzia. Natomiast w części aneksowej opisane są testy oprogramowania. Elementem poniższego opracowania jest także opis formatu danych wejściowych i wyjściowych.

2. Koncepcja techniczna

W niniejszym rozdziale opisane zostały wymagania postawione tworzonemu oprogramowaniu, czynniki, które zadecydowały o wyborze wykorzystanej technologii, oraz przyjęta operacjonalizacja wymagań w kontekście tej technologii. Na końcu rozdziału umieszczono syntetyczne podsumowanie przyjętego procesu generowania raportów.

2.1. Wymagania

Przy tworzeniu systemu generowania raportów postawiono sobie następujące cele:

1. Powinien zapewniać jak największą łatwość korzystania początkującym użytkownikom, w szczególności:
    - pozwalać na łatwe obliczanie podstawowych statystyk takich jak średnia, mediana, krotności, odsetki, kwantyle, korelacja liniowa i rangowa;
    - pozwalać na łatwe generowanie podstawowych wykresów: rozkładów, słupkowych oraz kołowych;
    - pozwalać na łatwe wczytywanie danych przygotowanych w zewnętrznym oprogramowaniu (np. programie eksportu danych z IRK i USOS, arkuszu kalkulacyjnym).
2. Powinien zapewniać jak największą elastyczność analiz użytkownikom zaawansowanym.
3. Powinien zapewniać automatyczną anonimizację danych zgodnie z przyjętymi regułami.
4. Powinien zapewniać automatyczne generowanie raportów dla wielu odbiorców.
5. Na podstawie tego samego szablonu raportu i danych, na podstawie których generowany jest raport w formacie PDF, powinna istnieć możliwość wygenerowania raportu także w innych formatach (np. HTML, DOCX, itp.).
6. Dodatkowym atutem byłaby możliwość łatwego przygotowania interaktywnych raportów w postaci stron WWW.

7. Powinien być wieloplatformowy (najlepiej, aby działał zarówno w systemie Windows, MacOs, jak i Linux).

8. O ile to możliwe, powinien opierać się na już dostępnych, wypróbowanych rozwiązaniach, przy czym powinny to być rozwiązania darmowe z punktu widzenia użytkownika pakietu.

Motywacją dla ostatniego z wyżej wymienionych celów było zredukowanie czasu potrzebnego na przygotowanie systemu oraz dążenie do zapewnienia jak największej niezawodności działania.

2.2. Wybór technologii i operacjonalizacja wymagań

W wyniku analizy dostępnego oprogramowania zdecydowano oprzeć się na środowisku programu statystycznego *R*. Istniejące w ramach tego środowiska oprogramowanie (w szczególności sam program *R* oraz pakiet *rmarkdown*) zapewniało już realizację wymagań 2, 5, 6, 7 i 8, tak więc niezbędne było jedynie uzupełnienie go o moduły pozwalające na spełnienie wymagań 1, 3 oraz 4. Dodatkowo środowisko *R* umożliwiało szybkie i łatwe wytworzenie takich modułów poprzez mechanizm *pakietów*.

Aby zoperacjonalizować wymagania tworzonego oprogramowania, niezbędne jest zapoznanie się z procesem generowania raportu w ramach wybranej technologii bazowej (pakiet *rmarkdown* dla programu statystycznego *R*):

1. Użytkownik przygotowuje szablon raportu w języku składu *markdown*.

2. Dodatkowo w szablonie można umieszczać dowolny kod R (tzw. *wstawki R*).

    - Podczas generowania raportu, pakiet *rmarkdown* (za pośrednictwem pakietu *knitr*) najpierw wykona wstawki R, a następnie zastąpi kod *wstawki* w szablonie raportu wynikiem jej działania.

    - W wyniku tej operacji powstaje spersonalizowany raport składający się już tylko z kodu w języku składu *markdown*.

3. Spersonalizowany raport w języku składu *markdown* konwertowany jest na wybrany format wyjściowy (PDF, DOCX lub HTML) za pomocą programu Pandoc.

    - W wypadku konwersji do PDF w pierwszej kolejności wykonywana jest konwersja na LaTeX, a następnie za pomocą dystrybucji LaTeX-a generowany jest PDF. Dlatego generowanie raportów w PDF wymaga obecności w systemie dystrybucji LaTeX-a.

Mechanizm ten umożliwia także wyświetlanie interaktywnych raportów w postaci stron WWW. W takim wypadku trzeba dodatkowo przygotować skrypt R opisujący kontrolki, za pomocą których użytkownik strony WWW będzie parametryzował treść raportu (np. wybór kierunku studiów, rocznika itp.). Wybrane przez użytkownika parametry przekazywane są na etapie wykonywania *wstawek R* (punkt 2 powyżej), co pozwala na wykonanie odpowiednio spersonalizowanego raportu. W tym wypadku eksport raportu, z racji wyświetlania go w postaci strony WWW, następuje zawsze do formatu HTML.

Powyższy proces nie zapewnia realizacji wymagań 1, 3 i 4, w szczególności:

- (*wymaganie 1*) Obliczanie statystyk i generowanie wykresów, które mają się znaleźć w raporcie, wymaga znajomości języka *R* i ewentualnie jego dodatkowych pakietów (np. pakietów do rysowania wykresów). Z jednej strony dla użytkowników, którzy nie mieli dotąd

styczności z *R* ani z programowaniem, może to stanowić bardzo poważną barierę. Z drugiej strony pisanie pełnej składni poleceń w języku *R* może czynić szablon raportu mniej czytelnym i udostępnienie najczęściej wykorzystywanych statystyk i wykresów (wymienionych w wymaganiu 1) w postaci gotowych, krótkich w zapisie funkcji będzie korzystne także w wypadku użytkowników zaawansowanych.

- Np. jeśli chcemy policzyć średnią ze zmiennej *Z* i wyświetlić jej wynik zaokrąglony do dwóch miejsc po przecinku, pełny kod w języku *R* wyglądałby następująco:

  ```
  sprintf('%.2f', mean(Z, na.rm = TRUE))
  ```

  Jest to zapis długi i nieczytelny. Korzystnie byłoby móc zastąpić go krótszym wywołaniem, które wewnętrznie wykona wszystkie potrzebne czynności, a dodatkowo samo przyjmie najczęściej wykorzystywane wartości formatowań (a więc np. domyślnie przyjmie zaokrąglanie wartości do 2 miejsc po przecinku):

  ```
  E(Z)
  ```

- Jeszcze bardziej jaskrawo różnicę tę widać w wypadku wykresów, gdzie wyświetlenie elegancko sformatowanego wykresu w *R* wymaga wpisania wielu linii kodu oraz znajomości rozlicznych parametrów graficznych sterujących wyglądem wykresu. Możliwość zastąpienia ich krótkim wywołaniem, w którym przekazywałoby się jedynie niezbędne informacje (dane, tytuł wykresu i ewentualnie osi), znacznie upraszczałoby przygotowanie szablonu raportu oraz poprawiało jego czytelność.

- (*wymaganie 1*) Wczytywanie danych przygotowanych w zewnętrznym oprogramowaniu, podobnie jak obliczanie statystyk, wymaga znajomości języka *R*. Także w tym wypadku korzystnie byłoby obudować proces wczytywania danych prostymi funkcjami pomocniczymi, którym wystarczyłoby jedynie przekazać nazwę pliku, a one same rozpoznawałyby jego format i w odpowiedni sposób wczytywały dane.

- (*wymaganie 1*) Istnieje jeszcze kilka innych aspektów związanych z koniecznością znajomości języków *R* oraz *markdown* i dobrego zrozumienia relacji między nimi w procesie generowania raportu, które mogą być nadmiernie trudne dla użytkownika. W szczególności dotyczą one tworzenia tabel w połączeniu ze stosowaniem *wstawek R* wewnątrz kolumn, warunkowego zamieszczania treści w raporcie oraz sposobu przetwarzania *wstawek R* w szablonie raportu. Również tutaj warto byłoby wytworzyć funkcje, które dawałyby użytkownikowi możliwość łatwego i intuicyjnego tworzenia szablonu raportu bez konieczności zagłębiania się w szczegóły techniczne.

- (*wymaganie 3*) Nie zapewnia mechanizmów automatycznej anonimizacji danych. Najwygodniejszym sposobem byłoby zintegrowanie anonimizacji danych z opisywanymi wyżej funkcjami obliczania statystyk oraz generowania wykresów. Pozwoliłoby to na stosowanie reguł anonimizacji wyników w sposób całkowicie przezroczysty i niewymagający podejmowania żadnych dodatkowych działań przez osoby przygotowujące szablon raportu i/lub generujące raporty.

- (*wymaganie 4*) Nie zapewnia mechanizmów automatycznego generowania raportów.

Pozwala to zoperacjonalizować wymagania pakietu *MLAK* jako:

- Wytworzenie funkcji obliczających statystyki: średnią, medianę, dowolne kwantyle, krotności, odsetki, korelację liniową (i jej kwadrat) oraz korelację rangową. Funkcje te powinny automatycznie dbać o anonimizację danych, jak również formatować wynik (uwzględniając wymogi stawiane formatowaniu przy osadzaniu wyniku w kolumnie tabeli). Przy formatowaniu powinny zakładać domyślnie najczęściej stosowany sposób formatowania, umożliwiając jednak dostosowanie go na życzenie użytkownika. Powinny zapewniać jak najbardziej skrótowy i czytelny zapis w szablonie raportu.
    - E(), Me(), N(), P(), Pw(), Q(), R(), R2(), Tau(), W()
- Wytworzenie funkcji wspomagającej obliczanie statystyk dla podgrup ze względu na kwantyle (np. analizy w podziale na grupy ze względu na wyniki studentów).
    - G()
- Wytworzenie funkcji umożliwiających generowanie wykresów: histogramu, słupkowego i kołowego. Funkcje te powinny zapewnić możliwie krótki i przejrzysty zapis w szablonie raportu (w wypadku konieczności radzenia sobie z różnymi rodzajami danych lepiej więc stworzyć kilka wariantów funkcji niż komplikować jej parametryzację). Powinny domyślnie formatować wykres zgodnie z przekazanymi danymi, ale także umożliwiać ewentualną ręczną zmianę parametrów wykresu przez użytkownika. Powinny automatycznie dbać o anonimizację danych.
    - wykresHistogram(), wykresSlupkowy(), wykresKolowy(), wykresKolowyNorm(), wykresKolowyZlicz()
- Wytworzenie funkcji umożliwiającej warunkowe wyświetlanie danego fragmentu raportu.
    - wstawTresc()
- Wytworzenie funkcji ułatwiających wczytywanie danych przygotowanych w oprogramowaniu zewnętrznym. Powinny odczytywać dane w formatach CSV (zapisanym zgodnie z MS Excel dla polskich ustawień językowych – separator pola: średnik, kodowanie znaków: Windows-1250) oraz RData. Powinny umożliwiać łatwe wczytywanie kompletu danych niezbędnych do wygenerowania raportu (zarówno dane o odbiorcy, jak i o jednostkach obserwacji, które typowo znajdują się w oddzielnych plikach).
    - wczytajOdbiorce(), wczytajDane()
- Wytworzenie funkcji umożliwiającej automatyczne wygenerowanie raportów dla wielu odbiorców.
    - generujRaporty()
- Wytworzenie i zintegrowanie z funkcjami do automatycznego generowania raportów oraz wczytywania danych odbiorcy funkcji ustawiających właściwą domyślną parametryzację przetwarzania *wstawek R* przez pakiet *rmarkdown* (a właściwie przez używany przez *rmarkdown* pakiet *knitr*).
    - konfigurujKnitr ()

2.3. Podsumowanie architektury rozwiązania

- Jako środowisko pracy wykorzystywany jest interpreter języka *R*.
- W ramach tego środowiska generowanie raportów odbywa się za pośrednictwem pakietu *rmarkdown* i polega na:
    - odczytaniu szablonu raportu zapisanego jako kombinacja języka składu *markdown* oraz wstawek w języku *R*;
    - wykonaniu wszystkich wstawek w języku *R* i podstawieniu wyników ich działania w miejsca, gdzie znajdowały się w szablonie raportu. W rezultacie otrzymywany jest raport w postaci czystego kodu *markdown*;
    - konwersji raportu w formacie *markdown* do wybranego formatu wynikowego (DOCX, HTML, PDF) za pomocą programu *Pandoc*.
        - W wypadku PDF najpierw następuje konwersja raportu na format LaTeX, a dopiero potem, za pośrednictwem dystrybucji LaTeX-a, konwersja na PDF.

  Istnieje również wariant powyższej procedury umożliwiający generowanie raportów w postaci interaktywnych stron WWW.
- Pakiet *MLAK* zawiera funkcje (w języku *R*) umożliwiające krótszy, bardziej czytelny oraz niewymagający głębszej znajomości języka *R* zapis wstawek w języku *R* w szablonie raportu.
    - Jednocześnie funkcje te dbają o zachowanie anonimowości danych.
- Pakiet *MLAK* zawiera funkcję (w języku *R*) umożliwiającą automatyczne wygenerowanie raportów dla wielu odbiorców.

3. Środowisko pracy i kompilacji

Podstawowym środowiskiem pracy i kompilacji pakietu *MLAK* jest interpreter języka *R*, nie wyczerpuje on jednak listy programów niezbędnych do działania pakietu. Poniżej opisane zostały wszystkie niezbędne zależności, oddzielnie dla środowiska pracy oraz środowiska kompilacji.

3.1. Środowisko pracy

Do działania pakietu *MLAK* niezbędne są:

- program statystyczny *R* w wersji 3.0 lub nowszej;
- program *Pandoc* w wersji 1.12.3 lub nowszej;
- dowolna dystrybucja LaTeX-a;
- pakiety programu *R* (wraz z pakietami *R*, od których zależą):

- *rmarkdown* w wersji 0.5.1 lub nowszej;
- *knitr* w wersji 1.6 lub nowszej;
- *ggplot2* w wersji 1.0.0 lub nowszej;
- *reshape2* w wersji 1.4.0 lub nowszej.

Wymienione powyżej pakiety *R* instalowane są automatycznie podczas instalacji pakietu *MLAK*, nie wymagają więc oddzielnej instalacji.

3.2. Środowisko programistyczne

Z pakietu *MLAK* można korzystać za pośrednictwem dowolnego środowiska programistycznego dla języka *R* (w szczególności domyślnego środowiska instalowanego wraz z programem statystycznym *R*), jednak największą wygodę, dzięki ścisłej integracji z funkcjonalnością pakietu *rmarkdown*, daje środowisko RStudio.

3.3. Środowisko kompilacji

Do skompilowania pakietu *MLAK* niezbędne są:

- program statystyczny R w wersji 3.0 lub nowszej;
- pakiety *devtools* programu *R* w wersji 1.7.0 lub nowszej (wraz z pakietami *R*, od których zależy, przy czym zależności te instalują się automatycznie przy instalacji pakietu *devtools*).

Sama kompilacja odbywa się w standardowy dla pakietów *R* sposób (komendą *R CMD build*).

Uwagi:

- Aby możliwe było wykonanie wszystkich testów (patrz rozdział 2.4), niezbędne są wszystkie programy opisane powyżej w części *środowisko pracy*.
- Do skorzystania z przygotowanej dla pakietu *MLAK* infrastruktury ciągłej integracji (zapewniającej automatyczne uruchomienie wszystkich testów po wprowadzeniu zmiany w repozytorium kodu źródłowego) niezbędne jest zainstalowanie klienta git lub GitHub.
- Jakkolwiek kompilacji pakietu można dokonać z wykorzystaniem różnych narzędzi (z wywołaniem programu *R* z linii komend systemu operacyjnego włącznie), polecane jest skorzystanie ze środowiska programistycznego RStudio. Zapewnia ono integrację z funkcjonalnością pakietu *devtools* oraz obsługuje repozytoria kodu źródłowego, co znacznie ułatwia przeprowadzenie procesu kompilacji.

4. Organizacja kodu źródłowego pakietu

Kod źródłowy pakietu zorganizowany jest zgodnie z wytycznymi dla pakietów *R*. Dodatkowo zawiera dwa pliki niezbędne do poprawnej współpracy z repozytorium kodu źródłowego (*.gitignore*) oraz systemem ciągłej integracji (*.travis.yml*).

- plik *.Rbuildignore* zawiera listę plików, które znajdują się w repozytorium kodu źródłowego, ale powinny zostać pominięte przy instalacji pakietu na komputerze użytkownika;
- plik *.gitignore* zawiera listę plików, które nie powinny być przesyłane do repozytorium kodu źródłowego;
- plik *.travis.yml* zawiera konfigurację systemu ciągłej integracji (Travis CI);
- plik *DESCRIPTION* zawiera opis pakietu i jego zależności od innych pakietów *R*;
    - każdy pakiet R musi posiadać ten plik;
- plik *LICENSE* zawiera licencję pakietu (licencja MIT);
- plik *NEWS* zawiera opis zmian zachodzących w kolejnych wersjach pakietu;
- plik *NAMESPACE* zawiera opis funkcji udostępnianych i importowanych przez pakiet;
    - każdy pakiet R musi posiadać ten plik;
    - w wypadku pakietu *MLAK* jest on generowany automatycznie na podstawie anotacji Roxygen2 w plikach z kodem źródłowym;
- plik *README.md* zawiera treść strony powitalnej wyświetlanej w repozytorium pakietu pod adresem https://github.com/zozlak/MLAK;
- katalog *R* zawiera kod źródłowy wszystkich funkcji pakietu (patrz niżej);
    - każda funkcja zdefiniowana jest w oddzielnym pliku, którego nazwa pokrywa się z nazwą funkcji;
    - argumenty, zwracana wartość i sposób działania funkcji udokumentowane są w postaci anotacji pakietu Roxygen2;
- katalog *man* zawiera pliki pomocy środowiska *R* dla wszystkich funkcji pakietu;
    - pliki pomocy generowane są automatycznie na podstawie anotacji Roxygen2 w plikach z kodem źródłowym;
- katalog *tests/testthat* zawiera testy automatyczne pakietu (patrz rozdział 4);
    - katalog test/testthat/dane zawiera dane (szablony raportów, pliki danych, pliki definicji odbiorców) używane w procesie testowania automatycznego.

Wyczerpujący opis sposobu organizacji kodu źródłowego pakietów R można znaleźć w:

- oficjalnym podręczniku;
- książce Hadleya Wickhama *R packages*;

przy czym pakiet *MLAK* nie zawiera niektórych opisywanych tam elementów, jak np. winietki, zbiory danych dołączane do pakietu czy kod źródłowy w językach innych niż *R*.

4.1. Organizacja funkcji pakietu

Zgodnie z wymaganiami organizacji kodu źródłowego pakietów *R* wszystkie funkcje pakietu *MLAK* znajdują się w jednym katalogu. Jednak z uwagi na rolę, jaką pełnią w pakiecie (i dla użytkownika pakietu), podzielić można je na kilka grup:

- **Funkcje do obliczania statystyk.** Odgrywają dwojaką rolę – z jednej strony ułatwiają przygotowywanie raportów użytkownikom nieznającym składni języka *R*, z drugiej strony zapewniają anonimizację danych, jeśli statystyka byłaby obliczana dla zbyt małej liczby obserwacji. Do funkcji tych zaliczają się:
  - E() – średnia,
  - Me() – mediana,
  - N() – liczba obserwacji lub liczba obserwacji o zadanej wartości zmiennej,
  - P() – odsetek obserwacji o zadanej wartości zmiennej,
  - Pw() – odsetek obserwacji o zadanej wartości zmiennej procentowany z pominięciem braków danych,
  - Q() – dowolnie określone kwantyle,
  - R(), R2() – korelacja liniowa Pearsona i jej kwadrat,
  - Tau() – korelacja rangowa Kendalla (tau-b).
- **Funkcje rysujące wykresy:**
  - wykresHistogram() – rozkład wartości zmiennej,
  - wykresSlupkowy() – wykres słupkowy,
  - wykresKolowy(), wykresKolowyNorm(), wykresKolowyZlicz() – różne warianty wykresu kołowego.
- **Pomocnicze funkcje do wykorzystania w szablonie raportu:**
  - W() – formatowanie przekazanej wartości, tak by pasowała ona do układu tabeli (patrz rozdział o tabelach w dokumentacji użytkownika),
  - wstawTresc() – warunkowe wstawianie fragmentów raportu (w połączeniu z konstrukcją *if* języka *R*),
  - G() – odfiltrowanie obserwacji należących do wskazanego kwantyla (patrz rozdział o obliczaniu statystyk w podziale na grupy w dokumentacji użytkownika).

- **Funkcja automatycznie generująca raporty dla wielu odbiorców** – generujRaporty().
- **Funkcja wczytująca dane wskazanego odbiorcy** – wczytajOdbiorce().
- **Pomocnicza funkcja do wczytywania zbiorów danych:**
    - wczytajDane() – wykorzystywana przy wołaniu wczytajOdbiorce() i generujRaporty() lub wewnętrznie przez obydwie te funkcje.
- **Wewnętrzne funkcje do obliczania statystyk**. Funkcje do obliczania statystyk tak naprawdę wywołują jedynie (z odpowiednią parametryzacją) jedną z dwóch funkcji zewnętrznych. Ułatwia to zapewnienie jednorodnego sposobu obliczania wszystkich statystyk (w tym także standardów anonimizacji i formatowania wyników):
    - statKorelacja() – wywoływana przez funkcje R(), R2() oraz Tau(),
    - statWektor() – wywoływana przez wszystkie pozostałe funkcje obliczające statystyki.
- **Wewnętrzne funkcje pomocnicze:**
    - wczytajCSV() – wczytywanie danych z plików CSV,
    - konfigurujKnitr() – ustawienie domyślnych parametrów konwersji szablonu raportu (wołana przez generujRaporty() i wczytajOdbiorce()),
    - giodo() – anonimizacja wyniku, jeśli statystyka liczona dla zbyt małej liczby obserwacji (wołana przez statKorelacja() i statWektor()),
    - naLiczbe(), wyrownajDl() – formatowanie wartości zwracanych przez funkcje obliczające statystyki, wołane przez statKorelacja() i statWektor(),
    - polamTekst(), wykresDefaultTheme() – formatowanie wykresów, wołane przez wszystkie funkcje rysujące wykresy,
    - zainstaluj() – lokalna instalacja pakietu ze źródeł (używana przy testowaniu wprowadzanych zmian przed zatwierdzeniem ich do repozytorium kodu źródłowego).

5. Testy

Kod źródłowy pakietu *MLAK* jest w 100% pokryty testami. Oznacza to, że każda z funkcjonalności pakietu jest w procesie testowania przynajmniej raz uruchamiana i mamy pewność, że skorzystanie z niej nie powoduje błędu. Testowanie odbywa się na dwóch poziomach:

- **Testów jednostkowych**
    - Kontrolujących poprawność wyników – koncentrują się one na bardzo dokładnej weryfikacji poprawności działania pojedynczych funkcji. Sprawdzają nie tylko to, czy wywołanie danej funkcji nie generuje błędu, ale także czy wynik jej działania jest

zgodny z oczekiwaniami. Tego typu testy przygotowane zostały dla wszystkich funkcji obliczających wartości statystyk (N(), E(), G(), Me(), P(), Pw(), Q(), R(), R2(), Tau()), wczytujących dane (wczytajDane(), wczytajOdbiorce()), a także wewnętrznych funkcji pomocniczych pakietu (wyrownajDl(), naLiczbe(), polamTekst(), statWektor(), statKorelacja()).

- Niekontrolujących poprawności wyników – weryfikują one jedynie, czy wywołanie funkcji nie zakończyło się błędem, nie sprawdzają natomiast poprawności wygenerowanego wyniku. Tego typu testy stosowane są np. w stosunku do funkcji rysujących wykresy, gdzie nie daje się w prosty sposób automatycznie porównać zgodności wygenerowanego wykresu z wykresem wzorcowym.

- **Testów wysokiego poziomu.** W trakcie testów tego typu generowane są próbnie przykładowe raporty. Testy te skupiają się na tym, czy poszczególne funkcje pakietu poprawnie współpracują między sobą i czy daje się z ich pomocą poprawnie wytworzyć realny raport. W raportach testowych użyte zostały wszystkie funkcje obliczające statystyki (N(), E(), G(), Me(), P(), Pw(), Q(), R(), R2(), Tau()) i generujące wykresy (wykresSlupkowy(), wykresKolowy(), wykresKolowyNorm(), wykresKolowyZlicz(), wykresHistogram()), funkcje pomocnicze (W(), G(), wstawTresc()), jak również funkcje wspomagające wczytywanie danych oraz automatyczne generowanie raportów dla wielu odbiorców.

Wszystkie przygotowane testy uruchamiane są automatycznie po dokonaniu każdej zmiany w kodzie źródłowym pakietu (dokładnie przy zatwierdzeniu zmian w repozytorium kodu źródłowego), co spełnia funkcję testów regresji, tzn. pozwala na natychmiastowe stwierdzenie, że zmiana dokonana w pakiecie spowodowała błąd w funkcjonalnościach, które dotychczas działały poprawnie.

Testy znajdują się w katalogu tests/testthat kodu źródłowego pakietu.

W wyniku przeprowadzenia testów automatycznie generowane są raporty pokrycia kodu źródłowego testami, co pozwala na łatwe zidentyfikowanie funkcjonalności, które nie są jeszcze testowane i odpowiednie uzupełnienie testów. Raporty pokrycia dostępne są na stronie https://coveralls.io/r/zozlak/MLAK. Ich wyniki dla wersji 0.5.2 pakietu można znaleźć w aneksie.

6. Wnioski metodologiczne

W toku prac nad modyfikacją oprogramowania do generowania raportów automatycznych na podstawie danych administracyjnych opracowane zostały ważne wnioski metodologiczne. Przygotowane ustalenia pozwolą na rozszerzenie funkcjonalności oprogramowania o cechy wymienione w opisie przedmiotu zamówienia oraz złożonej przez nas ofercie, tj.

- obsługę danych z wielu źródeł oraz selektywne generowanie raportów tylko dla absolwentów wybranej uczelni;

- możliwość definiowania i generowania raportów z analizami ze względu na region geograficzny oraz specyficzne cechy uczelni.

Ponadto w raporcie przedstawimy:

- wnioski metodologiczne, które pozwolą na rozszerzenie koncepcji bezpiecznej analizy statystycznej zawartości rejestrów danych osobowych z uwzględnieniem postulatu prowadzenia pogłębionych analiz losów absolwentów na wielu uczelniach;

- wykaz technicznych modyfikacji zawartości plików pomocniczych;
- wykaz zmian merytorycznej zawartości wzorców raportów automatycznych.

6.1. Obsługa danych z wielu źródeł oraz selektywne generowanie raportów tylko dla absolwentów wybranej uczelni

W ramach realizacji projektu powstało oprogramowanie służące do eksportu informacji z systemu USOS. Stanowi ono część tego systemu i jest dostępne dla wszystkich uczelni wykorzystujących USOS. Było już z powodzeniem przetestowane, a następnie wykorzystane do eksportu informacji z systemów Uniwersytetu Warszawskiego i Uniwersytetu Śląskiego. Szczegółowy opis wspomnianego rozwiązania informatycznego znajduje się w osobnej dokumentacji. Pozwala ono na łatwy eksport potrzebnych danych zawierających łatwo interpretowalne zmienne, które mogą być podstawą do analiz procesów kształcenia – w szczególności sukcesu edukacyjnego na studiach.

Zbiory wyeksportowane za pomocą opracowanego oprogramowania daje się w prosty sposób połączyć w jedną bazę danych w formie prostokątnej tablicy. Jest to zapewnione przez ustalony, jednolity standard eksportu danych, jednakowy dla wszystkich uczelni posługujących się USOS. Dokumentacja tego oprogramowania zawiera opis zmiennych pozwalający na przygotowanie analogicznych rozwiązań informatycznych przez inne uczelnie, nie posługujące się USOS.

Oprogramowanie służące do generowania raportów automatycznych, rozwinięte w niniejszym projekcie, posługuje się plikami pomocniczymi – podobnie jak w przypadku oprogramowania pierwotnego. Plik zawierający listę odbiorców raportów automatycznych pozwala na scharakteryzowanie poziomu agregacji wyników oraz wskazanie, dla których konkretnie grup mają zostać wygenerowane raporty automatyczne. Możliwe jest zatem wygenerowanie raportów na przykład wyłącznie dla wybranej grupy kierunków dla wybranej uczelni lub wyselekcjonowanej grupy uczelni.

Warto przypomnieć, że pewne analizy mają sens tylko dla określonych poziomów agregacji. Zatem poziom agregacji determinuje niejednokrotnie zakres merytoryczny raportów. Przykładowo, analiza rozmaitych wskaźników ze względu na niepełnosprawność powinna być prowadzona z zasady na poziomie całej uczelni – z powodu niewielkich liczebności osób niepełnosprawnych w jednostkach niższego rzędu.

6.2. Możliwość definiowania i generowania raportów z analizami ze względu na region geograficzny oraz specyficzne cechy uczelni

W poprzedniej wersji oprogramowania stworzonego na potrzeby Uniwersytetu Warszawskiego na podstawie kodu pocztowego wyznaczano powiat pochodzenia oraz powiat zamieszkania absolwenta. Na tej podstawie klasyfikowano absolwentów w jednej z czterech kategorii geograficznych: powiatu warszawskiego, Obszaru Metropolitalnego Warszawy, województwa mazowieckiego oraz obszaru poza województwem mazowieckim. Taka klasyfikacja ma jednak ograniczone zastosowanie – jest na przykład mało użyteczna dla uczelni znajdujących się poza Warszawą. Podobne – poczwórne – klasyfikacje można by utworzyć dla innych największych centrów akademickich Polski (np. dla Trójmiasta, Krakowa, Katowic, czy Łodzi). Przeciwko utrzymaniu tego rozwiązania przemawiają dwie przesłanki:

- Wyniki prowadzonych w ostatnich latach przez PEJK UW badań rekrutacyjnych wykazały, że wzorce decyzji edukacyjnych kandydatów zamieszkałych na terenie Obszaru Metropolitalnego Warszawy nie różnią się od decyzji osób zamieszkałych w Warszawie. Należy się spodziewać, że zamieszkanie w mieście, w którym znajduje się uczelnia, nie będzie odróżniało zasadniczo sposobu kształcenia się osób mieszkających w bliskiej odległości od miasta. Zaliczenie danego obszaru do okolic miasta wynika z istnienia dobrych połączeń komunikacyjnych z miastem, a zatem i łatwego dostępu jego infrastruktury, w tym uczelni.

- Dla ośrodków akademickich związanych z mniejszymi miastami nie da się sensownie wyznaczyć odpowiednika Obszaru Metropolitalnego Warszawy, gdyż wpływ tej uczelni nie wykracza poza powiat, w którym się ona znajduje.

Dlatego zdecydowaliśmy się posługiwać trójwartościową klasyfikacją. Wyróżniamy osoby zamieszkałe

- w mieście, w którym znajduje się uczelnia, lub w jego okolicy,
- w województwie, w którym znajduje się uczelnia (z wyjątkiem osób z pierwszej grupy),
- poza województwem, w którym znajduje się uczelnia.

Ustalenie listy powiatów należących do obszaru traktowanego jako okolica miasta, w którym znajduje się uczelnia, należy pozostawić w gestii uczelni. W niniejszym projekcie, na użytek przykładowych analiz dotyczących Uniwersytetu Warszawskiego i Uniwersytetu Śląskiego, za okolice miast przyjęliśmy, odpowiednio, Obszar Metropolitalny Warszawy (powiaty: Warszawa, grodziski, legionowski, piaseczyński, pruszkowski, warszawski zachodni, żyrardowski, grójecki, miński, nowodworski, otwocki, sochaczewski, wołomiński, wyszkowski) oraz Konurbację Górnośląską (Bytom, Chorzów, Dąbrowa Górnicza, Jaworzno, Gliwice, Katowice, Mysłowice, Piekary Śląskie, Ruda Śląska, Siemianowice Śląskie, Sosnowiec, Świętochłowice, Tychy, Zabrze).

Przygotowane oprogramowanie umożliwia także tworzenie fragmentów raportów automatycznych, które pojawiałyby się jedynie dla wybranych uczelni. Ze względu na liczne podobieństwa pomiędzy UW i UŚ nie zaistniała potrzeba tworzenia osobnych fragmentów raportów dla każdej z uczelni.

Dotychczas we wzorcach raportów automatycznych nazwa uczelni była stałym elementem raportu. Obecnie jest to zmienna tekstowa, której wartości oprogramowanie pobiera z pliku odbiorców.

6.3. Wnioski metodologiczne, które pozwolą na rozszerzenie koncepcji bezpiecznej analizy statystycznej zawartości rejestrów danych osobowych z uwzględnieniem postulatu prowadzenia pogłębionych analiz losów absolwentów na wielu uczelniach

- W toku prac nad projektem, spośród czterech wariantów dostosowania analizy danych statystycznych w badaniach losów absolwentów do wymogów ochrony danych osobowych, za najwłaściwszy uznaliśmy wariant d, który przewiduje uniemożliwienie bezpośredniego dostępu do danych – przetwarzanie „wsadowe” (off-line) pełnych baz danych statystycznych. Rozwiązanie to zostało szczegółowo opisane w punkcie 4.9 złożonej przez nas oferty. Niezbędnym elementem niniejszego rozwiązania jest istnienie ośrodka nazwanego przez nas roboczo „Bezpiecznym Centrum Obliczeniowym X”. Funkcję tego ośrodka może pełnić jednostka obliczeniowa wyodrębniona przez Ośrodek Przetwarzania Informacji – Państwowy Instytut Badawczy (OPI). Ośrodek Przetwarzania Informacji – Państwowy Instytut Badawczy ściśle współpracuje z Ministerstwem Nauki i Szkolnictwa Wyższego, które wspiera w procesie podejmowania ważnych decyzji. W swoich działaniach badawczych wykorzystuje procedury zapewniające bezpieczeństwo danych osobowych. Usytuowanie tam zbioru danych statystycznych eksportowanych przez uczelnie pozwoliłoby na bezpieczne przetwarzanie tych informacji bez potrzeby tworzenia osobnej instytucji przeznaczonej do celu badań losów absolwentów uczelni. Docelowo rozszerzone informacje eksportowane z systemów uczelni powinny być kierowane do systemu POL-on (lub podobnego) administrowanego przez OPI. Kluczową kwestią, zarówno ze względów merytorycznych, organizacyjnych, jak i wizerunkowych, jest precyzyjne określenie zasad współpracy między uczelniami i OPI, zapewniających wzajemne poszanowanie autonomii tych instytucji przy możliwie szerokim zakresie badań. Bardziej szczegółowy opis uwarunkowań prawnych i administracyjnych tego rozwiązania znajduje się w raporcie końcowym z projektu *Monitorowanie losów absolwentów uczelni z wykorzystaniem danych administracyjnych Zakładu Ubezpieczeń Społecznych.*

Część 2: Wdrożenie systemu monitorowania losów absolwentów i popularyzacja technik analiz informacji pochodzących z rejestrów opracowanych w ramach projektów badawczych IBE.

- Ważną funkcjonalnością oprogramowania generującego raporty automatyczne zapewniającą anonimowość wyników analiz jest funkcja uzależniająca włączanie wyników analiz statystycznych do treści raportów od liczebności zbiorowości, których dotyczy dany wynik. W dokumentacji technicznej oprogramowania została opisana jako funkcja „giodo()". Liczebność graniczną prowadzonych analiz można ustalić dowolnie restrykcyjnie.

6.4. Wykaz technicznych modyfikacji zawartości plików pomocniczych

- W pierwotnej wersji oprogramowania w plikach odbiorców znajdowały się definicje wszystkich podgrup, w odniesieniu do których były prowadzone jakiekolwiek obliczenia w ramach generowania raportu automatycznego. Aktualnie definicje te znajdują się we wzorcach raportów.

- Zawartość pliku odbiorców została rozszerzona o zmienne będące funkcją uczelni, których dotyczy dany rekord, w tym nazwa uczelni odmieniona przez wszystkie przypadki, skrót nazwy uczelni oraz zmienne definiujące nazwy i zakres obszarów geograficznych używanych dla danego odbiorcy.

6.5. Wykaz zmian merytorycznej zawartości wzorców raportów automatycznych

- W zakończonym projekcie „Monitorowanie losów absolwentów uczelni wyższych z wykorzystaniem danych administracyjnych Zakładu Ubezpieczeń Społecznych" przewidziano analizy z użyciem tzw. przedmiotów kluczowych, tj. obowiązkowych, jednolicie sprawdzanych i merytorycznie istotnych z punktu widzenia programu studiów. W celu selekcji przedmiotów wykładanych na uczelni, na każdym etapie studiów i dla każdego etapu studiów należałoby corocznie przeprowadzać rozległe badania terenowe. Takie badania zostały przeprowadzone na Uniwersytecie Warszawskim w ramach wspomnianego projektu. Nawet jednorazowe przeprowadzenie podobnych badań jest czasochłonnym i kosztownym przedsięwzięciem. Pozornie jego ponowne przeprowadzenie sprowadzałoby się do aktualizacji istniejących danych. Jednakże na uczelniach naszego kraju swobodnie pojawiają się nowe programy studiów, zaś inne zanikają. W konsekwencji tego nawet na pojedynczej uczelni pojawiłaby się konieczność określania przedmiotów kluczowych na pewnych programach studiów od zera. Podobnie miałaby się rzecz ze wszystkimi programami studiów szkoły wyższej, która włączałaby się do tworzonego systemu monitorowania losów absolwentów uczelni z wykorzystaniem danych administracyjnych. Dodatkową trudnością (omówioną przez nas w *Raporcie badawczym ze studium współzależności procesów dydaktycznych oraz funkcjonowania elektronicznego systemu obsługi studiów*) jest brak zasad odnoszących się do „dziedziczenia" kodów. Nie ma jasnych reguł dotyczących tego, kiedy przedmiot powinien otrzymać odrębny kod. Oczywiste jest, że dwa różne przedmioty powinny mieć odmienne kody, ale nie zawsze jest zupełnie jasne, jaka zmiana w obrębie jednego przedmiotu w kolejnych okresach powinna prowadzić do nadania nowego kodu. Zmiana prowadzącego lub koordynatora przy niezmienionym programie może być traktowana różnie. Podobnie jest w przypadku niewielkich uaktualnień w opisie przedmiotu. Czasami, mimo niewielkich zmian, przyjmuje się, że przedmiot powinien otrzymać nowy kod. Do tego dochodzi zjawisko zmieniania przez prowadzących wyłącznie nazw przedmiotów. Przedstawione problemy skłoniły nas w niniejszym projekcie do zastąpienia analiz odwołujących się do tzw. przedmiotów kluczowych analizami bazującymi na średniej rang ze wszystkich przedmiotów zaliczanych na danym etapie studiów. Przyjęte rozwiązanie umożliwia automatyzację procesu przygotowywania danych do eksportu z systemu

elektronicznego uczelni i nie wymaga ponoszenia systematycznych, znacznych nakładów na aktualizację cząstkowych informacji o przedmiotach. Nie generuje dodatkowych kosztów początkowych dla kolejnych uczelni. Zamiast corocznych, kosztownych badań terenowych wystarczy przy zaproponowanym rozwiązaniu przeprowadzać okresowe (w cyklu kilkuletnim) studia współzależności systemu informatycznego i procesów dydaktycznych na uczelni, które pozwalają na określenie skali i struktury niedopasowań między wspomnianymi systemami.

- W zakończonym projekcie planowaliśmy prowadzenie analiz na poziomie poszczególnych specjalizacji w ramach jednego kierunku. Wyniki studium homomorfizmu systemu edukacyjnego i informatycznego skłoniły nas do rezygnacji z posługiwania się tak drobną kategorią analityczną. Sposób zapisywania specjalności w systemach informatycznych uczelni jest kłopotliwy. Specjalności niekiedy mogłyby być traktowane niemal jak odrębne programy studiów, z innymi wymaganiami, przedmiotami obowiązkowymi itd. Jednak ze względu na fakt, że otrzymują jeden kod programu, w ramach prowadzonych przez nas badań nie możemy ich traktować jako osobnych programów, mimo że w wielu wypadkach mogłoby się to wydawać merytorycznie uzasadnione. Zakładamy bowiem, że nie ingerujemy w definicje przyjęte przez uczelnie. W konsekwencji może się okazać, że w przypadku pewnych kierunków będziemy mieli do czynienia z wieloma podzbiorowościami absolwentów, którzy będą się różnić w istotny sposób kompetencjami, a zatem i szansą powodzenia na rynku pracy, co będzie wynikać ze studiów w ramach innych specjalizacji. Niestety w takich przypadkach nie będzie możliwości odróżnienia tych osób, a więc wychwycenia, które specjalizacje spośród oferowanych w ramach danego kierunku pozwalają w największym stopniu odnieść sukces na rynku pracy. Do powyższego należy dodać potencjalne trudności związane z przygotowaniem algorytmów bazujących na dostępnych kodach i służących wychwyceniu, w ramach których kierunków zostały wyróżnione specjalności. Identyfikowanie specjalizacji i uwzględnianie jej w analizach na równi z programem studiów wymagałoby stworzenia dodatkowego słownika „programo-specjalności", a jednym z założeń naszych badań jest rezygnacja z tworzenia nowych słowników.

- W raportach z funkcjonowania absolwentów na rynku pracy przewidziano podawanie średniej i wybranych kwantyli średniorocznej liczby pracodawców. Liczba ta uwzględniała jedynie pracodawców zatrudniających badanych na umowę o pracę, a parametry wyliczane były tylko dla absolwentów pracujących etatowo. Idea średniorocznej liczby etatowych pracodawców jest dość skomplikowana i może okazać się niezrozumiała dla części czytelników. Skłoniło nas to do zastąpienia średniorocznej liczby pracodawców prostszym wskaźnikiem – liczbą pracodawców. Nadal uwzględniani są tylko pracodawcy zatrudniający na umowę o pracę, a parametry są wyznaczane dla absolwentów pracujących etatowo. Wspomniany wskaźnik wyliczany będzie wyłącznie dla kohort kończących studia w tym samym roku, nie będzie zaś wyliczany w analizach odnoszących się do kohort równocześnie rozpoczynających studia. Osoby rozpoczynające studia w tym samym roku mogą je kończyć w różnych, niekiedy odległych momentach, a przez to mogą się znacznie różnić czasem funkcjonowania na rynku pracy jako absolwenci. Podawanie w takim razie miar określających liczbę pracodawców dla osób różniących się długością okresu objętego analizą mogłoby być dezinformujące.

- W zakończonym projekcie planowaliśmy wykluczać z analiz funkcjonowania na rynku pracy tych absolwentów, którzy nie figurują w rejestrach Zakładu Ubezpieczeń Społecznych. Obecnie zdecydowaliśmy się włączyć ich do analiz, gdyż uznaliśmy, że brak informacji o osobie w ZUS należy traktować nie tyle jako niedoskonałość systemu bądź lukę w rejestrze, ile raczej jako przesłankę o braku aktywności badanego na rynku pracy. Jesteśmy świadomi tego, że brak informacji o badanych nie musi oznaczać braku zatrudnienia. W celu rozstrzygnięcia tej kwestii pomocne byłyby oczywiście informacje pochodzące z innych rejestrów (np. Kasy Rolniczego Ubezpieczenia Społecznego, urzędów skarbowych). Możliwości rozszerzenia bazy do badań losów absolwentów wykorzystujących rejestry administracyjne zostały opisane w *Raporcie badawczym ze studium możliwości wykorzystania danych zastanych jako źródła w badaniach edukacyjno-zawodowych losów absolwentów szkół wyższych* powstałym w ramach projektu „Monitorowanie losów absolwentów uczelni z wykorzystaniem danych administracyjnych Zakładu Ubezpieczeń Społecznych. Część 2:

Wdrożenie systemu monitorowania losów absolwentów i popularyzacja technik analiz informacji pochodzących z rejestrów opracowanych w ramach projektów badawczych IBE”.

7. Aneks – raporty pokrycia kodu pakietu testami

Plik	Nr linii	Kod	Liczba wywołań w testach
E.R	7	`E = function(x, wyrownaj = T, dokl = 2){`	
	8	`  return(statWektor(x, mean, sys.call(), wyrownaj, dokl))`	142×
	9	`}`	
G.R	16	`G = function(x, q, n, filtr = NULL){`	
	17	`  if(is.null(filtr)){`	12×
	18	`    filtr = rep(T, length(x))`	4×
	19	`  }`	12×
	20	`  stopifnot(`	12×
	21	`    is.numeric(x), is.vector(x),`	12×
	22	`    is.numeric(q), is.vector(q), length(q) == 1, all(!is.na(q)),`	12×
	23	`    is.numeric(n), is.vector(n), length(n) == 1, all(!is.na(n)), n >= 1,`	12×
	24	`    is.logical(filtr), is.vector(filtr),`	12×
	25	`    length(filtr) == length(x),`	12×
	26	`    n >= q`	12×
	27	`  )`	12×
	28		
	29	`  tmp = quantile(x[filtr], seq(0, 1, length.out = n + 1), na.rm = TRUE)`	12×
	30	`  if(q == 1){`	12×
	31	`    tmp[q] = tmp[q] - 1 # aby nie pomijać najmniejszej obserwacji`	3×
	32	`  }`	12×
	33	`  return(x <= tmp[q + 1] & x > tmp[q] & filtr & !is.na(x))`	12×
	34	`}`	
Me.R	7	`Me = function(x, wyrownaj = T, dokl = 2){`	
	8	`  return(statWektor(x, median, sys.call(), wyrownaj, dokl))`	10×
	9	`}`	
N.R	12	`N = function(x, w = NULL, wyrownaj = T){`	
	13	`  stopifnot(`	344×
	14	`    is.vector(x),`	344×
	15	`    is.numeric(x) | is.character(x) | is.logical(x),`	344×
	16	`    is.vector(w) | is.null(w)`	344×
	17	`  )`	344×
	18	`  if(!is.null(w)){`	344×
	19	`    stopifnot(`	278×
	20	`      is.numeric(w) | is.character(w) | is.logical(w)`	278×
	21	`    )`	278×
	22	`  }`	344×
	23		
	24	`  if(!is.null(w)){`	344×
	25	`    # obejście umożliwiające zliczanie NA`	344×

Plik	Nr linii	Kod	Liczba wywołań w testach
	26	`    if(any(is.na(w))){`	278×
	27	`      if(all(is.na(x))){`	28×
	28	`        tmp = 1`	3×
	29	`      }else if(is.numeric(x)){`	28×
	30	`        tmp = max(x, na.rm = T) + 1`	24×
	31	`      }else{`	28×
	32	`        tmp = paste0('_', max(x, na.rm = T))`	1×
	33	`      }`	28×
	34	`      x[is.na(x)] = tmp`	28×
	35	`      w[is.na(w)] = tmp`	28×
	36	`    }`	278×
	37		344×
	38	`    # obejście funkcji giodo() przy braku wartości w wektorze`	344×
	39	`    f = function(d){`	278×
	40	`      return(sum(d %in% w))`	278×
	41	`    }`	278×
	42	`    if(f(x) == 0){`	278×
	43	`      x = c(x, rep(NA, 50))`	92×
	44	`    }`	278×
	45		344×
	46	`    return(statWektor(x, f, sys.call(), wyrownaj, 0))`	278×
	47	`  }else{`	344×
	48	`    return(statWektor(x, length, sys.call(), wyrownaj, 0))`	66×
	49	`  }`	344×
	50	`}`	
P.R	13	`P = function(x, w, wyrownaj = T, dokl = 1, znakProcent = TRUE){`	
	14	`  stopifnot(`	222×
	15	`    is.vector(dokl), is.numeric(dokl), length(dokl) == 1, all(!is.na(dokl)),`	222×
	16	`    is.vector(znakProcent), is.logical(znakProcent), length(znakProcent) == 1, all(!is.na(znakProcent))`	222×
	17	`  )`	222×
	18		
	19	`  wynik = N(x, w, F)`	222×
	20	`  if(is.numeric(wynik)){`	222×
	21	`    wynik = round(100 * wynik / length(x), dokl)`	148×
	22	`  }else`	222×
	23	`  if(wyrownaj){`	222×
	24	`    wynik = wyrownajDl(wynik, sys.call(), dokl, 3)`	72×
	25	`  }`	222×
	26	`  if(znakProcent){`	222×
	27	`    wynik = paste0(wynik, '%')`	220×
	28	`  }`	222×
	29	`  return(wynik)`	222×

Plik	Nr linii	Kod	Liczba wywołań w testach
	30	}	
Pw.R	13	Pw = function(x, w, wyrownaj = T, dokl = 1, znakProcent = TRUE){	
	14	return(P(x[!is.na(x)], w[!is.na(w)], wyrownaj, dokl, znakProcent))	5×
	15	}	
Q.R	13	Q = function(x, q, n, wyrownaj = T, dokl = 2){	
	14	stopifnot(	21×
	15	is.numeric(x),	21×
	16	is.numeric(q),	21×
	17	is.numeric(n),	21×
	18	length(n) == 1,	21×
	19	all(n >= q),	21×
	20	n >= 1	21×
	21	)	21×
	22		
	23	f = function(x){	18×
	24	tmp = quantile(x, seq(0, 1, length.out = n + 1))	16×
	25	return(tmp[q + 1])	16×
	26	}	18×
	27	return(statWektor(x, f, sys.call(), wyrownaj, dokl))	18×
	28	}	
R.R	8	R = function(x, y, wyrownaj = T, dokl = 2){	
	9	return(statKorelacja(x, y, wyrownaj, dokl, 'pearson', FALSE, sys.call()))	19×
	10	}	
R2.R	8	R2 = function(x, y, wyrownaj = T, dokl = 2){	
	9	return(statKorelacja(x, y, wyrownaj, dokl, 'pearson', TRUE, sys.call()))	13×
	10	}	
Tau.R	8	Tau = function(x, y, wyrownaj = T, dokl = 2){	
	9	return(statKorelacja(x, y, wyrownaj, dokl, 'kendall', FALSE, sys.call()))	14×
	10	}	
W.R	11	W = function(x){	
	12	dl = 4 + nchar(deparse(sys.call()))	2×
	13	return(sprintf(paste0('% ', dl, 's'), x))	2×
	14	}	
generujRaporty.R	29	generujRaporty = function(plikSzablonu, dane, grupyOdbiorcow, katalogWy = '', prefiksPlikow = ''){	
	30	konfigurujKnitr()	2×
	31		
	32	if(is.character(dane)){	2×
	33	stopifnot(	1×
	34	length(dane) == 1,	1×
	35	file.exists(dane)	1×
	36	)	1×

Plik	Nr linii	Kod	Liczba wywołań w testach
	37	`    dane = wczytajDane(dane)`	1×
	38	`  }`	2×
	39	`  if(is.character(grupyOdbiorcow)){`	2×
	40	`    stopifnot(`	1×
	41	`      length(grupyOdbiorcow) == 1,`	1×
	42	`      file.exists(grupyOdbiorcow)`	1×
	43	`    )`	1×
	44	`    grupyOdbiorcow = wczytajDane(grupyOdbiorcow)`	1×
	45	`  }`	2×
	46	`  stopifnot(`	2×
	47	`    is.data.frame(dane),`	2×
	48	`    is.data.frame(grupyOdbiorcow) | is.list(grupyOdbiorcow)`	2×
	49	`  )`	2×
	50	`  if(katalogWy == ''){`	2×
	51	`    katalogWy = '.'`	2×
	52	`  }`	2×
	53		
	54	`   # przerabianie grup odbiorców podanych jako ramka danych na listę`	
	55	`  if(is.data.frame(grupyOdbiorcow)){`	2×
	56	`    tmp = list()`	2×
	57	`    for(i in 1:nrow(grupyOdbiorcow)){`	2×
	58	`      tmp[[i]] = as.list(grupyOdbiorcow[i, ])`	2×
	59	`    }`	2×
	60	`    names(tmp) = grupyOdbiorcow[, 1]`	2×
	61	`    grupyOdbiorcow = tmp`	2×
	62	`  }`	2×
	63		
	64	`  for(i in 1:length(grupyOdbiorcow)){`	2×
	65	`    odbiorca = wczytajOdbiorce(grupyOdbiorcow, dane, i)`	2×
	66	`    with(`	2×
	67	`      odbiorca,`	2×
	68	`      {`	2×
	69	`        render(`	2×
	70	`          input = plikSzablonu,`	2×
	71	`          output_format = pdf_document(),`	2×
	72	`          output_file = paste0(`	2×
	73	`            prefiksPlikow,`	2×
	74	`            names(grupyOdbiorcow)[i],`	2×
	75	`            '.pdf'`	2×
	76	`          ),`	2×
	77	`          output_dir = katalogWy`	2×
	78	`        )`	2×
	79	`      }`	2×

Plik	Nr linii	Kod	Liczba wywołań w testach
	80	)	2×
	81	}	2×
	82	}	
giodo.R	10	giodo = function(x){	
	11	stopifnot(	555×
	12	is.vector(x) | is.data.frame(x)	555×
	13	)	555×
	14		
	15	min = 5	555×
	16		
	17	if(is.vector(x)){	555×
	18	if(length(x) < min){	520×
	19	x[] = NA	130×
	20	}	520×
	21	}	555×
	22	if(is.data.frame(x)){	555×
	23	if(nrow(x) < min){	35×
	24	x[] = NA	3×
	25	}	35×
	26	}	555×
	27		
	28	return(x)	555×
	29	}	
konfiguru jKnitr.R	4	konfigurujKnitr = function(){	
	5	opts_chunk$set(	34×
	6	'error' = FALSE, 'warnings' = FALSE, 'message' = FALSE,	34×
	7	'echo' = FALSE, 'results' = 'asis'	34×
	8	)	34×
	9		
	10	if(!is.null(opts_knit$get('rmarkdown.pandoc.to'))){	34×
	11	if(opts_knit$get('rmarkdown.pandoc.to') == 'latex'){	13×
	12	cairo = capabilities()['cairo']	13×
	13	if(cairo %in% TRUE){	13×
	14	opts_chunk$set('dev' = 'cairo_pdf')	13×
	15	}	13×
	16	# Bez tego na Mac-u koniec koncow produkuja sie poprawne wykresy,	13×
	17	# ale najpierw nastepuje litania bledow (tak jakby mimo wskazania	13×
	18	# cairo_pdf() probowal najpierw wyprodukowac wykres za pomoca pdf(),	13×
	19	# a dopiero po bledzie przechodzil do cairo_pdf())	13×
	20	pdf.options(encoding = 'CP1250')	13×
	21	}	13×
	22	}	34×

Plik	Nr linii	Kod	Liczba wywołań w testach
	23	}	
naLiczbe.R	7	naLiczbe = function(dane){	
	8	stopifnot(	75×
	9	is.vector(dane)	75×
	10	)	75×
	11	if(is.numeric(dane)){	73×
	12	return(dane)	1×
	13	}	73×
	14	if(is.character(dane)){	72×
	15	dane = sub('%$', '', dane)	72×
	16	}	72×
	17	dane = setNames(suppressWarnings(as.numeric(dane)), names(dane))	72×
	18	return(dane)	72×
	19	}	
polamTekst.R	9	polamTekst = function(wektor, n = 15, wymus = FALSE){	
	10	stopifnot(	81×
	11	is.vector(wektor) | is.factor(wektor),	81×
	12	is.vector(n), is.numeric(n), length(n) == 1, all(!is.na(n)),	81×
	13	is.vector(wymus), is.logical(wymus), length(wymus) == 1, all(!is.na(wymus))	81×
	14	)	81×
	15		
	16	dodajOdstepy = function(x){	81×
	17	x = sub('^\n+', '', x)	81×
	18	x = sub('\n+$', '', x)	81×
	19	return(paste0('\n', x, '\n'))	81×
	20	}	81×
	21		
	22	if(any(grepl('\n', wektor))){	81×
	23	if(wymus){	24×
	24	wektor = sub('\n', ' ', wektor)	1×
	25	}else{	24×
	26	return(dodajOdstepy(wektor))	23×
	27	}	24×
	28	}	81×
	29		
	30	wektor = gsub(' +', ' ', wektor)	58×
	31	wyrazy = strsplit(wektor, ' ')	58×
	32	wynik = sapply(wyrazy, function(x){	58×
	33	suma = 0	58×
	34	wynik = ''	58×
	35	for(i in seq_along(x)){	58×

Plik	Nr linii	Kod	Liczba wywołań w testach
	36	`    dl = nchar(x[i])`	58×
	37	`    if(suma > 0 & suma + 1 + dl > n){`	58×
	38	`      wynik = paste0(wynik, '\n')`	58×
	39	`      suma = 0`	58×
	40	`    }else if(suma > 0){`	58×
	41	`      wynik = paste0(wynik, ' ')`	58×
	42	`      suma = suma + 1`	58×
	43	`    }`	58×
	44	`    wynik = paste0(wynik, x[i])`	58×
	45	`    suma = suma + dl`	58×
	46	`  }`	58×
	47	`  return(dodajOdstepy(wynik))`	58×
	48	`})`	58×
	49	`return(dodajOdstepy(wynik))`	58×
	50	`}`	
statKorelacja.R	24	`statKorelacja = function(x, y, wyrownaj, dokl, metoda, kwadrat, call){`	
	25	`  suppressWarnings(stopifnot(`	46×
	26	`    is.vector(x) | is.factor(x), is.numeric(x) | is.character(x) | is.logical(x) | is.factor(x),`	46×
	27	`    is.vector(y) | is.factor(y), is.numeric(y) | is.character(y) | is.logical(y) | is.factor(y),`	46×
	28	`    length(x) == length(y),`	46×
	29	`    any(!is.na(x) & !is.na(y)),`	46×
	30	`    is.vector(wyrownaj), is.logical(wyrownaj), length(wyrownaj) == 1, all(!is.na(wyrownaj)),`	46×
	31	`    is.vector(dokl), is.numeric(dokl), length(dokl) == 1, all(!is.na(dokl)),`	46×
	32	`    is.vector(metoda), is.character(metoda), length(metoda) == 1, all(metoda %in% c('pearson', 'spearman', 'kendall')),`	46×
	33	`    is.vector(kwadrat), is.logical(kwadrat), length(kwadrat) == 1, all(!is.na(kwadrat)),`	46×
	34	`    is.call(call)`	46×
	35	`  ))`	46×
	36	`  if(metoda %in% 'pearson'){`	43×
	37	`    stopifnot(`	30×
	38	`      is.numeric(x), is.numeric(y)`	30×
	39	`    )`	30×
	40	`  }`	43×
	41	`  if(!is.numeric(x)){`	35×
	42	`    x = xtfrm(x)`	3×
	43	`  }`	35×
	44	`  if(!is.numeric(y)){`	35×
	45	`    y = xtfrm(y)`	1×
	46	`  }`	35×
	47	`  dane = giodo(data.frame(x = x, y = y))`	35×
	48	`  dane = dane[!is.na(dane$x) & !is.na(dane$y), ]`	35×

Plik	Nr linii	Kod	Liczba wywołań w testach
	49	`  if(nrow(dane) == 0){`	35×
	50	`    wynik = '-'`	3×
	51	`  }else{`	35×
	52	`    wynik = cor(dane$x, dane$y, method = metoda)`	32×
	53	`    if(kwadrat){`	32×
	54	`      wynik = wynik^2;`	8×
	55	`    }`	32×
	56	`    if(!is.numeric(wynik) | is.na(wynik) | is.nan(wynik)){`	32×
	57	`      wynik = '-'`	1×
	58	`    }`	32×
	59	`  }`	35×
	60	`  if(wyrownaj){`	35×
	61	`    return(wyrownajDl(wynik, call, dokl))`	18×
	62	`  }else if(is.character(wynik)){`	35×
	63	`    return(wynik)`	4×
	64	`  }`	35×
	65	`  return(round(wynik, dokl))`	13×
	66	`}`	
statWektor.R	18	`statWektor = function(x, f, call, wyrownaj = T, dokl = 2){`	
	19	`  stopifnot(`	520×
	20	`    is.vector(x),`	520×
	21	`      is.numeric(x) | is.character(x) | is.logical(x),`	520×
	22	`    is.vector(wyrownaj), is.logical(wyrownaj), length(wyrownaj) == 1, all(!is.na(wyrownaj)),`	520×
	23	`    is.vector(dokl), is.numeric(dokl), length(dokl) == 1, all(!is.na(dokl))`	520×
	24	`  )`	520×
	25	`  x = giodo(x)`	520×
	26	`  x = x[!is.na(x)]`	520×
	27	`  if(length(x) == 0){`	520×
	28	`    wynik = '-'`	204×
	29	`  }else{`	520×
	30	`    wynik = f(x)`	316×
	31	`    if(!is.numeric(wynik)){`	316×
	32	`      wynik[] = '-'`	4×
	33	`    }`	316×
	34	`    if(any(is.na(wynik)) | any(is.nan(wynik)) | any(is.infinite(wynik))){`	316×
	35	`      wynik[is.na(wynik) | is.nan(wynik) | is.infinite(wynik)] = '-'`	2×
	36	`    }`	316×
	37	`  }`	520×
	38		
	39	`  if(wyrownaj){`	520×
	40	`    return(wyrownajDl(wynik, call, dokl))`	234×
	41	`  }else if(is.character(wynik)){`	520×

Plik	Nr linii	Kod	Liczba wywołań w testach
	42	return(wynik)	85×
	43	}	520×
	44	return(round(wynik, dokl))	201×
	45	}	
wczytajCSV.R	15	wczytajCSV = function(	
	16	plik,	
	17	sep = ';',	
	18	quote = '"',	
	19	dec = ',',	
	20	fileEncoding = 'Windows-1250',	
	21	header = TRUE,	
	22	fill = T,	
	23	comment.char = '',	
	24	stringsAsFactors = FALSE,	
	25	...	
	26	){	
	27	return(read.table(	5×
	28	plik,	5×
	29	header = header,	5×
	30	sep = sep,	5×
	31	quote = quote,	5×
	32	dec = dec,	5×
	33	fill = fill,	5×
	34	comment.char = comment.char,	5×
	35	stringsAsFactors = stringsAsFactors,	5×
	36	fileEncoding = fileEncoding,	5×
	37	...	5×
	38	))	5×
	39	}	
wczytajDane.R	12	wczytajDane = function(plik){	
	13	stopifnot(	17×
	14	file.exists(plik),	17×
	15	grepl('[.](csv|rdata)$', tolower(plik))	17×
	16	)	17×
	17		
	18	plikL = tolower(plik)	17×
	19	if(grepl('csv$', plikL)){	17×
	20	dane = wczytajCSV(plik)	5×
	21	}else if(grepl('rdata$', plikL)){	17×
	22	srodowisko = new.env()	12×
	23	load(plik, srodowisko, FALSE)	12×
	24	dane = list()	12×
	25	for(i in ls(, envir = srodowisko)){	12×

Plik	Nr linii	Kod	Liczba wywołań w testach
	26	`      tmp = get(i, envir = srodowisko)`	12×
	27	`      if(is.data.frame(tmp)){`	12×
	28	`        dane[[i]] = tmp`	11×
	29	`      }`	12×
	30	`    }`	12×
	31	`    if(length(dane) == 0){`	12×
	32	`      dane = data.frame()`	2×
	33	`    }else if(length(dane) == 1){`	12×
	34	`      dane = dane[[1]]`	9×
	35	`    }`	12×
	36	`  }`	17×
	37		
	38	`  return(dane)`	17×
	39	`}`	
wczytajOdbiorce.R	24	`wczytajOdbiorce = function(grupy, dane = data.frame(), n = 1){`	
	25	`  # aby nie było potrzebne oddzielne wywolywanie przy generowaniu raportu`	
	26	`  # wprost z RStudio`	
	27	`  konfigurujKnitr()`	32×
	28		
	29	`  # pobieramy stos wywołań; uwaga:`	
	30	`  # - pozycja 1 to funkcja najwyższego poziomu`	
	31	`  # - pozycja ostatnia to funkcja, w której właśnie jesteśmy`	
	32	`  stos = unlist(lapply(sys.calls(), function(x){`	32×
	33	`    return(deparse(x)[1])`	32×
	34	`  }))`	32×
	35	`  pozGenRap = suppressWarnings(max(seq_along(stos)[grepl('^generujRaporty', stos)]))`	32×
	36	`  if(length(pozGenRap) > 0 & pozGenRap != length(stos) - 1 & !is.infinite(pozGenRap)){`	32×
	37	`    return(list())`	13×
	38	`  }`	32×
	39		
	40	`  if(is.character(grupy)){`	19×
	41	`    stopifnot(`	6×
	42	`      length(grupy) == 1,`	6×
	43	`      file.exists(grupy)`	6×
	44	`    )`	6×
	45	`    grupy = wczytajDane(grupy)`	5×
	46	`  }`	19×
	47	`  if(is.character(dane)){`	18×
	48	`    stopifnot(`	1×
	49	`      length(dane) == 1,`	1×
	50	`      file.exists(dane)`	1×

Plik	Nr linii	Kod	Liczba wywołań w testach
	51	)	1×
	52	dane = wczytajDane(dane)	1×
	53	}	18×
	54	stopifnot(	18×
	55	is.data.frame(dane),	18×
	56	is.list(grupy) | is.data.frame(grupy)	18×
	57	)	18×
	58		
	59	if(is.data.frame(grupy)){	18×
	60	odbiorca = grupy[n,]	5×
	61	}else{	18×
	62	odbiorca = grupy[[n]]	13×
	63	}	18×
	64		
	65	odbiorca = as.list(odbiorca)	18×
	66	kolEval = grep('^[.]', names(odbiorca), value = T)	18×
	67		
	68	if(length(kolEval) > 0){	18×
	69	odbiorca = with(	11×
	70	dane,	11×
	71	{	11×
	72	for(i in kolEval){	11×
	73	odbiorca[[i]] = eval(parse(text = odbiorca[[i]]))	11×
	74	}	11×
	75	return(odbiorca)	10×
	76	}	11×
	77	)	11×
	78	}	18×
	79		
	80	names(odbiorca) = sub('^[.]', '', names(odbiorca))	17×
	81		
	82	odbiorca = append(odbiorca, dane)	17×
	83	return(odbiorca)	17×
	84	}	
wstawTresc.R	7	wstawTresc = function(kod){	
	8	stopifnot(	16×
	9	is.character(kod),	16×
	10	length(kod) == 1	16×
	11	)	16×
	12		
	13	if(file.exists(kod)){	16×
	14	kod = readChar(kod, 10^6)	8×
	15	}	16×

Plik	Nr linii	Kod	Liczba wywołań w testach
	16		
	17	`  plik = tempfile(fileext = '.Rmd')`	16×
	18	`  writeChar(kod, plik)`	16×
	19	`  plikMd = knit(plik, tempfile(fileext = '.md'), quiet = TRUE, envir = parent.frame())`	16×
	20	`  kodMd = readChar(plikMd, 10^6)`	16×
	21	`  unlink(c(plik, plikMd))`	16×
	22	`  cat(kodMd)`	16×
	23	`  return(invisible(kodMd))`	16×
	24	`}`	
wykresDefaultTheme.R	10	`wykresDefaultTheme = function(wykres, tytul = NULL, tytulX = NULL, tytulY = NULL, rozmiarTekstu = NULL){`	
	11	`  if(is.null(rozmiarTekstu)){`	85×
	12	`    rozmiarTekstu = 10`	85×
	13	`  }`	85×
	14		
	15	`  wykres = wykres +`	85×
	16	`    ylab('') +`	85×
	17	`    xlab('') +`	85×
	18	`    scale_fill_grey() +`	85×
	19	`    labs(fill = '') +`	85×
	20	`    theme_bw(base_size = rozmiarTekstu) +`	85×
	21	`    theme(`	85×
	22	`      panel.border      = element_rect(fill = NA, colour=NA),`	85×
	23	`      legend.position   = 'right',`	85×
	24	`      legend.text       = element_text(size = rozmiarTekstu),`	85×
	25	`      axis.text.x       = element_text(size = rozmiarTekstu),`	85×
	26	`      axis.text.y       = element_text(size = rozmiarTekstu),`	85×
	27	`      axis.title.x      = element_text(size = rozmiarTekstu * 1.2),`	85×
	28	`      axis.title.y      = element_text(size = rozmiarTekstu * 1.2),`	85×
	29	`      plot.title        = element_text(size = rozmiarTekstu * 1.3)`	85×
	30	`    )`	85×
	31		
	32	`  if(!is.null(tytul)){`	85×
	33	`    wykres = wykres + ggtitle(tytul)`	85×
	34	`  }`	85×
	35	`  if(!is.null(tytulX)){`	85×
	36	`    wykres = wykres + xlab(tytulX)`	5×
	37	`  }`	85×
	38	`  if(!is.null(tytulY)){`	85×
	39	`    wykres = wykres + ylab(tytulY)`	20×

Plik	Nr linii	Kod	Liczba wywołań w testach
	40	}	85×
	41		
	42	return(wykres)	85×
	43	}	
wykresHistogram.R	14	wykresHistogram = function(dane, n = 9, tytul = '', tytulX = NULL, tytulY = NULL, rozmiarTekstu = NULL, opcjeWykresu = NULL){	
	15	stopifnot(	9×
	16	is.vector(dane) | is.factor(dane),	9×
	17	is.numeric(dane) | is.character(dane) | is.logical(dane) | is.factor(dane)	9×
	18	)	9×
	19		
	20	dane = na.exclude(dane)	9×
	21		
	22	wykres = ggplot(data = data.frame(d = dane)) +	9×
	23	aes(x = get('d'))	9×
	24		
	25	if(is.numeric(dane)){	9×
	26	breaks = seq(min(dane), max(dane), length.out = n + 1)	7×
	27		9×
	28	wykres = wykres +	7×
	29	geom_histogram(	7×
	30	breaks = breaks,	7×
	31	colour = '#000000',	7×
	32	fill = '#FFFFFF'	7×
	33	)	7×
	34	}else{	9×
	35	wykres = wykres +	2×
	36	geom_histogram(	2×
	37	colour = '#000000',	2×
	38	fill = '#FFFFFF'	2×
	39	)	2×
	40	}	9×
	41		
	42	wykres = wykresDefaultTheme(wykres, tytul = tytul, tytulX = tytulX, tytulY = tytulY, rozmiarTekstu = rozmiarTekstu) +	9×
	43	theme(axis.line = element_line(colour = '#000000', linetype = 'solid'), axis.line.x = element_blank())	9×
	44		
	45	if(is.numeric(dane)){	9×
	46	wykres = wykres + scale_x_continuous(breaks = round(breaks, 2))	7×
	47	}	9×
	48		
	49	if(!is.null(opcjeWykresu)){	9×
	50	wykres = wykres + opcjeWykresu	1×

Plik	Nr linii	Kod	Liczba wywołań w testach
	51	}	9×
	52		
	53	return(wykres)	9×
	54	}	
wykresKolowy.R	21	wykresKolowy = function(dane, tytul = '', rozmiarTekstu = NULL, opcjeWykresu = NULL){	
	22	stopifnot(	51×
	23	is.vector(dane), is.numeric(dane) | is.character(dane)	51×
	24	)	51×
	25		
	26	dane = na.exclude(dane)	51×
	27	etykiety = dane	51×
	28	if(is.character(dane)){	51×
	29	dane = sub('^ +', '', dane)	46×
	30	etykiety = dane	46×
	31	dane = naLiczbe(dane)	46×
	32	if(any(is.na(dane))){	46×
	33	stop('Dane wykresu zawieraly wartosci niebedace liczbami ani procentami')	1×
	34	}	46×
	35	}	51×
	36		
	37	# upewnij się, że wektor danych ma nazwy	
	38	if(is.null(names(dane))){	50×
	39	names(dane) = dane	1×
	40	}	50×
	41		
	42	dane = data.frame(	50×
	43	e = polamTekst(paste(names(dane), '-', etykiety)),	50×
	44	y = as.numeric(dane),	50×
	45	stringsAsFactors = FALSE	50×
	46	)	50×
	47	dane$e = factor(dane$e, levels = dane$e, labels = dane$e)	50×
	48		
	49	wykres = ggplot(data = dane) +	50×
	50	aes(x = factor(1), y = get('y'), fill = get('e')) +	50×
	51	geom_bar(width = 1, stat = 'identity') +	50×
	52	coord_polar(theta = 'y') +	50×
	53	scale_x_discrete(breaks = NULL) +	50×
	54	scale_y_continuous(breaks = NULL)	50×
	55	wykres = wykresDefaultTheme(wykres, tytul = tytul, rozmiarTekstu = rozmiarTekstu) +	50×
	56	theme(	50×
	57	panel.grid.major = element_blank()	50×
	58	)	50×

Plik	Nr linii	Kod	Liczba wywołań w testach
	59		
	60	`  if(!is.null(opcjeWykresu)){`	50×
	61	`    wykres = wykres + opcjeWykresu`	3×
	62	`  }`	50×
	63		
	64	`  return(wykres)`	50×
	65	`}`	
wykresKolowyNorm.R	14	`wykresKolowyNorm = function(dane, dokl = 1, tytul = '', rozmiarTekstu = NULL, opcjeWykresu = NULL){`	
	15	`  stopifnot(`	36×
	16	`    is.vector(dane), is.numeric(dane) | is.character(dane)`	36×
	17	`  )`	36×
	18		
	19	`  dane = na.exclude(dane)`	36×
	20	`  if(is.character(dane)){`	36×
	21	`    dane = sub('%$', '', dane)`	10×
	22	`    dane = sub('^ +', '', dane)`	10×
	23	`    tmp = names(dane)`	10×
	24	`    dane = setNames(suppressWarnings(as.numeric(dane)), tmp)`	10×
	25	`    if(any(is.na(dane))){`	10×
	26	`      stop('Dane wykresu zawieraly wartosci niebedace liczbami ani procentami')`	1×
	27	`    }`	10×
	28	`  }`	36×
	29		
	30	`  dane = round(dane * 100 / sum(dane), dokl)`	35×
	31	`  if(length(dane) > 1){`	35×
	32	`    dane[1] = 100 - sum(dane[-1])`	35×
	33	`  }`	35×
	34	`  tmp = names(dane)`	35×
	35	`  dane = setNames(paste0(dane, '%'), tmp)`	35×
	36		
	37	`  wykres = wykresKolowy(dane, tytul = tytul, rozmiarTekstu = rozmiarTekstu, opcjeWykresu = opcjeWykresu)`	35×
	38	`  return(wykres)`	35×
	39	`}`	
wykresKolowyZlicz.R	28	`wykresKolowyZlicz = function(dane, etykiety = NULL, tytul = '', rozmiarTekstu = NULL, opcjeWykresu = NULL){`	
	29	`  stopifnot(`	24×
	30	`    is.vector(dane) | is.factor(dane),`	24×
	31	`    is.numeric(dane) | is.character(dane) | is.logical(dane) | is.factor(dane)`	24×
	32	`  )`	24×
	33		
	34	`  # logical -> factor`	
	35	`  if(is.logical(dane)){`	24×

Plik	Nr linii	Kod	Liczba wywołań w testach
	36	`    dane = factor(dane, levels = c(T, F), labels = c('TAK', 'NIE'))`	1×
	37	`  }`	24×
	38		
	39	`  # etykietowanie wartości`	
	40	`  if(is.character(etykiety)){`	24×
	41	`    wartosci = names(etykiety)`	16×
	42	`    if(is.null(wartosci)){`	16×
	43	`      wartosci = seq_along(etykiety)`	16×
	44	`    }`	16×
	45	`    dane = factor(dane, wartosci, etykiety)`	16×
	46	`  }`	24×
	47		
	48	`  dane = table(dane)`	24×
	49	`  tmp = names(dane)`	24×
	50	`  dane = setNames(as.vector(dane), tmp)`	24×
	51		
	52	`  wykres = wykresKolowyNorm(dane, tytul = tytul, rozmiarTekstu = rozmiarTekstu, opcjeWykresu = opcjeWykresu)`	24×
	53	`  return(wykres)`	24×
	54	`}`	
wykresSlupkowy.R	16	`wykresSlupkowy = function(dane, skumulowany = F, tytul = '', tytulX = NULL, tytulY = NULL, sufiksY = '', rozmiarTekstu = NULL, opcjeWykresu = NULL){`	
	17	`  stopifnot(`	27×
	18	`    is.vector(dane) | is.matrix(dane)`	27×
	19	`  )`	27×
	20	`  # ew. konwersja na liczby`	
	21	`  if(is.character(dane)){`	27×
	22	`    tmp = sum(is.na(dane))`	23×
	23	`    dane = naLiczbe(dane)`	23×
	24	`    if(sum(is.na(dane)) != tmp){`	23×
	25	`      stop('dane nie są liczbami')`	1×
	26	`    }`	23×
	27	`  }`	27×
	28	`  stopifnot(`	26×
	29	`    is.numeric(dane)`	26×
	30	`  )`	26×
	31		
	32	`  if(is.vector(dane)){`	26×
	33	`    tmp = names(dane)`	26×
	34	`    dane = matrix(dane, length(dane), 1)`	26×
	35	`    rownames(dane) = tmp`	26×
	36	`  }`	26×
	37		

Plik	Nr linii	Kod	Liczba wywołań w testach
	38	`  dane = melt(dane, varnames = c('y', 'x'))`	26×
	39	`  dane$y = polamTekst(dane$y)`	26×
	40	`  dane$y = factor(dane$y, levels = dane$y, labels = dane$y)`	26×
	41	`  pozycja = ifelse(skumulowany, position_stack, position_dodge)`	26×
	42	`  wykres = ggplot(data = dane) +`	26×
	43	`    aes(x = factor(get('x')), y = get('value'), fill = get('y')) +`	26×
	44	`    geom_bar(`	26×
	45	`      stat = 'identity',`	26×
	46	`      position = pozycja()`	26×
	47	`    )`	26×
	48	`  wykres = wykresDefaultTheme(wykres,  tytul = tytul, tytulX = tytulX, tytulY = tytulY, rozmiarTekstu = rozmiarTekstu) +`	26×
	49	`    theme(axis.line = element_line(colour = '#000000', linetype = 'solid'), axis.line.x = element_blank()) +`	26×
	50	`    scale_y_continuous(labels = function(x){`	26×
	51	`      return(paste0(x, sufiksY))`	15×
	52	`    })`	26×
	53		
	54	`  if(length(unique(dane$x)) == 1){`	26×
	55	`    wykres = wykres + scale_x_discrete(breaks = NULL)`	26×
	56	`  }`	26×
	57		
	58	`  if(!is.null(opcjeWykresu)){`	26×
	59	`    wykres = wykres + opcjeWykresu`	1×
	60	`  }`	26×
	61		
	62	`  return(wykres)`	26×
	63	`}`	
wyrownajDl.R	12	`wyrownajDl = function(wynik, call, dokl, delta = 4){`	
	13	`  dl = delta + nchar(deparse(call))`	332×
	14	`  if(max(nchar(wynik)) > dl){`	332×
	15	`    warning('Wyrównanie długości nie było możliwe')`	1×
	16	`  }`	332×
	17	`  format = ifelse(`	332×
	18	`    is.character(wynik),`	332×
	19	`    paste0('% ', dl, 's'),`	332×
	20	`    paste0('% ', dl, '.', dokl, 'f')`	332×
	21	`  )`	332×
	22	`  return(setNames(sprintf(format, wynik), names(wynik)))`	332×
	23	`}`	
zainstaluj.R	5	`zainstaluj = function(){`	
	6	`  suppressMessages(devtools::document(roclets = c('rd', 'collate', 'namespace')))`	1x

Plik	Nr linii	Kod	Liczba wywołań w testach
	7	`  devtools::build(quiet = TRUE)`	1x
	8	`  tmp = list.files('../', '^MLAK.*tar.gz$')`	1x
	9	`  tmp = tmp[order(tmp, decreasing = TRUE)]`	1x
	10	`  install.packages(paste0('../', tmp[1]), repos = NULL, quiet = TRUE)`	1x
	11	`  return(invisible(TRUE))`	1x
	12	`}`	