# Programowanie i analiza danych w R

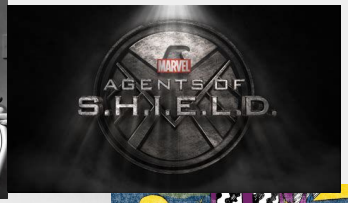
Laboratorium I: podstawy R

# O mnie

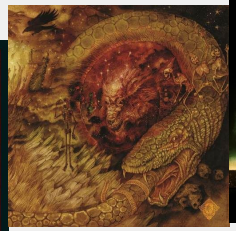
- doktorant - biostatystyka,
- badania:
  - wcześniej - interpretable machine learning (PW)
  - teraz - biostatystyka (metody statystyczne w proteomice/spektrometrii mas) (UWr + UHasselt),
- praktyka:
  - analiza danych w ramach studiów/doktoratu + współpraca z kilkoma firmami
  - aktualnie: freelancer

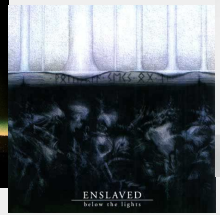
# O mnie

- R:
  - 3 pakiety na CRAN-ie
  - 2 kursy dla studentów z wykorzystaniem R
  - warsztaty i wystąpienia na konferencjach R (eRum + Why R? x 2)
  - warsztaty w ramach spotkań STWUR-a
  - analizy i rozwój kodu naukowo i komercyjnie
- Oprócz tego można ze mną porozmawiać o





# Konsultacje

poniedziałek,
dogodna dla Was godzina

# O czym będzie ten kurs (i jak go zaliczyć)

# Zasady zaliczenia

Punkty zostaną przyznane za 3 grupy zadań:

1. Aktywność na zajęciach i bonusowe punkty z zadań domowych: 7 punktów.

2. Zadania domowe: 14 zadań po 3 punkty (razem 42 punkty). Do zaliczenia konieczne jest oddanie każdego zadania. Opóźnienie skutkuje utratą punktów (1 punkt za każdy tydzień opóźnienia, ale nie więcej niż trzy punkty).

3. Projekt całosemestralny (do oddania na ostatnich zajęciach): 51 punktów. Oddanie projektu jest konieczne do zaliczenia.

# Zasady zaliczenia - szczegóły

1. Nie ma się czego bać.
2. Zadania domowe: najpierw Data Camp, potem krótkie fragmenty kodu do napisania. Kod będzie czytany przez innych i dyskutowany na zajęciach.
3. Projekt: pakiet do napisania - szczegóły na trzecim laboratorium.

# Analiza danych

## Implementacja

```
# xgboost:
set.seed(17, kind = "L'Ecuyer")
mod_xgb <- train(xgboost, subsetTask(hlh_task_xgboost, train_set))
# logreg:
mod_logreg <- train(logreg, subsetTask(hlh_task_sym, train_set))
# lasso:
set.seed(17, kind = "L'Ecuyer")
mod_lasso <- train(lasso, subsetTask(hlh_task_sym, train_set))
# decision tree
set.seed(17, kind = "L'Ecuyer")
mod_tree <- train(tree, subsetTask(hlh_task_sym, train_set))


roc_lasso_plot <- draw_roc_curve(mod_lasso,
                                 subsetTask(hlh_task_sym, test_set),
                                 "Lasso")
roc_logreg_plot <- draw_roc_curve(mod_logreg,
                                 subsetTask(hlh_task_sym, test_set),
                                 "Logistic regression")
```

## Interpretacja

- “poziom bilirubiny wpływa na ryzyko zgonu”
- “starsi pacjenci mają mniejszą szansę przeżycia, bo...”


14:05 - 14:50 Always Be Deploying. How to make R great for machine learning in (not only) Enterprise.

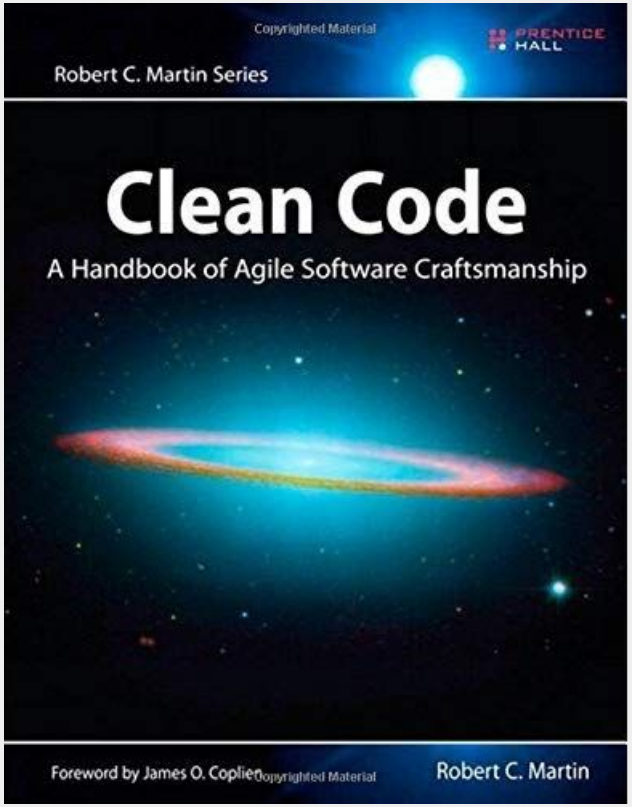
Author: Wit Jakuczun (WLOG Solutions)

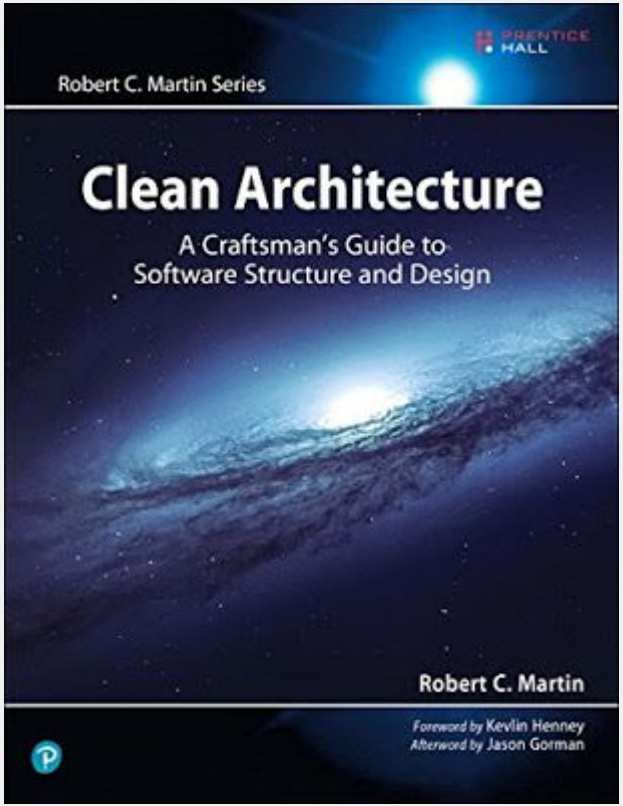
For many years software engineers have put enormous effort to develop best practices to deliver stable and maintainable software. How R users can benefit from this experience? I will try to answer this question going through several concepts and tools that are natural for software engineers but are often undervalued by R users.
I will start with a description of the deployment process because this is the ultimate step that exposes all weaknesses. You will learn about structuring R project, using abstractions to manage model's features, automating models building process, optimizing the performance of the solution and the challenges of the deployment process itself.

- Why R 2019 (wkrótce online)

- Analiza danych wymaga tworzenia kodu
  - automatyzacja,
  - wiarygodność,
  - reprodukowalność
- Dobry kod dla analizy danych musi być **dobrym kodem** - obowiązują powszechne w programowaniu zasady.

**Na tym kursie będziemy się uczyć tworzenia dobrego kodu w R.**


Copyrighted Material
PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Copyrighted Material
Robert C. Martin


PRENTICE HALL
Robert C. Martin Series
Clean Architecture
A Craftsman's Guide to Software Structure and Design
Robert C. Martin
Foreword by Kevlin Henney
Afterword by Jason Gorman

# SOLID principles (Uncle Bob)

- **Single responsibility principle**: a class should have one, and only one, reason to change. Or the new version: a module should be responsible to one, and only one, actor.
- **Open-closed principle**: a class should be open for extension but closed for modification.
- **Liskov's substitution principle**: objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- **Interface segregation principle**: many client-specific interfaces are better than one general-purpose interface.
- **Dependency inversion principle**: one should depend upon abstractions, not concretions.

(Podsumowanie z https://clevercoder.net/2018/09/08/clean-architecture-summary-review/)

# Dobry kod

- jest **czytelny**
- jest zmodularyzowany
- jest dobrze udokumentowany
- jest przetestowany
- łatwo poddaje się zmianom

“Code is read much more often than it is written”

- Guido van Rossum

“If you do not communicate your ideas, you are not doing science. You are only engaging in a hobby”

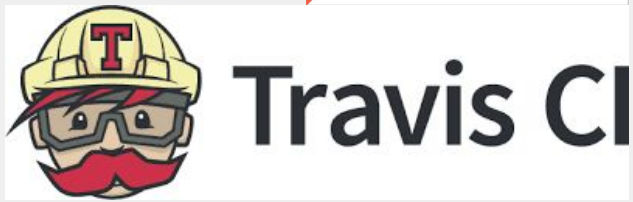
- Kurt W. Kolasinski

(To samo dotyczy Data Science)

# Pomocne narzędzia programisty

- Version control


- Continuous Integration




- Code Coverage


- IDE

# R jako język programowania

# R: bardzo ogólnie

- Interpretowany (jak Python, inaczej niż C++)
-

# Cechy funkcjonalne

For our purposes, the main principles of functional programming can be summarized as follows:

1. Programming consists largely of defining *functions*.
2. A function definition in the language, like a function in mathematics, implies that a function call returns a unique value corresponding to each valid set of arguments, but *only* dependent on these arguments.
3. A function call has no side effects that could alter other computations.

https://arxiv.org/pdf/1409.3531.pdf

# Cechy obiektowe

The main ideas of object-oriented programming are also quite simple and intuitive:

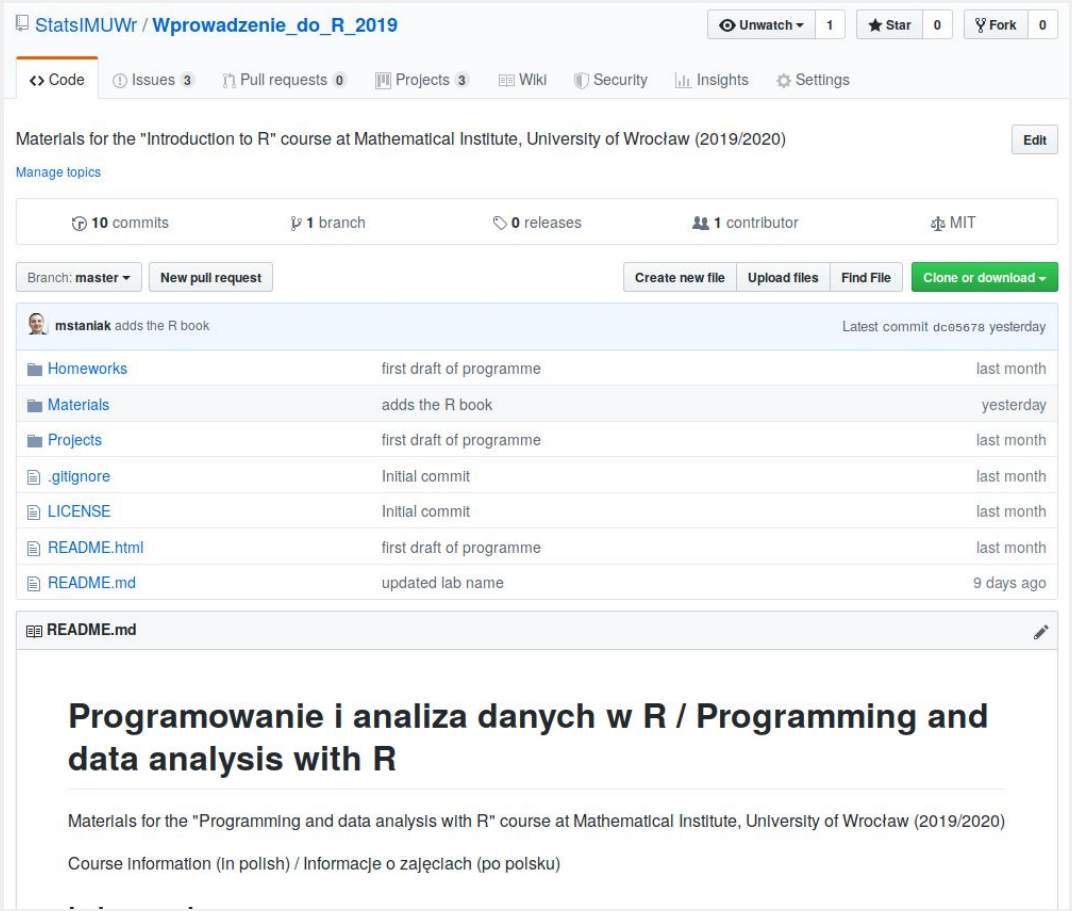
1. Everything we compute with is an *object*, and objects should be structured to suit the goals of our computations.
2. For this, the key programming tool is a *class* definition saying that objects belonging to this class share structure defined by *properties* they all have, with the properties being themselves objects of some specified class.
3. A class can *inherit* from (contain) a simpler superclass, such that an object of this class is also an object of the superclass.
4. In order to compute with objects, we can define *methods* that are only used when objects are of certain classes.

# Przechodzimy do praktyki

# Problemy

1. O co w ogóle chodzi?
    a. Co jest co?
    b. Skąd się biorą funkcje?
    c. Jak szukać pomocy?
    d. Skąd brać informacje nt. obiektów?
2. Jak obliczyć wariancję zbioru obserwacji?
    a. Podstawowe struktury danych w R
    b. Podstawowe operacje na wektorach
3. Jak obliczyć liczności poziomów zmiennych jakościowych?
    a. Dwuwymiarowe struktury danych w R
    b. Operacje na macierz i listach, ramki danych

# Podsumowanie




https://github.com/StatsIMUWr/Wprowadzenie_do_R_2019

# Zadanie domowe

(A dokładniej trzy zadania domowe)

1. Na 14 X - pierwsze dwa rozdziały kursu Introduction to R
2. Na 21 X - kolejne dwa rozdziały kursu Introduction to R
3. Na 28 X - kurs z Gita (lub wybrane rozdziały - do ustalenia)