Tworzenie wykresów za pomocą biblioteki ggplot2

Bartosz Kozak

20 10 2019

Wykorzystanie R do generowania wykresów i ilustracji

Wykorzystując R można stworzyć dowolny wykres, za pomocą R możemy wizualizować wszystkie typy danych. Nadal korzystamy z biblioteki ggplot (wszystkie elementy omawiane na ostatnich zajęciach nadal maja zastosowanie). W dalszej części będziemy wykorzystywać plik warianty_z_asemblingu.bed, który można pobrać ze strony katedry.

Wczytanie danych

```
library(ggplot2)
theme_set(theme_gray(base_size = 18))
plik <- "warianty_z_asamblingu.bed"
my_data <- read.csv(plik, sep="\t", quote='', stringsAsFactors=TRUE,header=FALSE)
head(my_data)
```

```
##   V1        V2        V3  V4   V5 V6        V7  V8   V9
## 1  6 103832058 103832059 SV1  185  + Insertion   0  185
## 2  6 102958468 102958469 SV2  317  + Insertion -14  303
## 3  6 102741692 102741693 SV3  130  +  Deletion 130    0
## 4  6 102283759 102283760 SV4 1271  + Insertion -12 1259
## 5  6 101194032 101194033 SV5 2864  + Insertion -13 2851
## 6  6 101056644 101056645 SV6  265  + Insertion   0  265
```

```
names(my_data) <- c("chrom","start","stop","nazwa","wielkość","orientacja","typ","ref.dyst","query.dyst
head(my_data)
```

```
##   chrom     start      stop nazwa wielkość orientacja       typ ref.dyst
## 1     6 103832058 103832059   SV1      185          + Insertion        0
## 2     6 102958468 102958469   SV2      317          + Insertion      -14
## 3     6 102741692 102741693   SV3      130          +  Deletion      130
## 4     6 102283759 102283760   SV4     1271          + Insertion      -12
## 5     6 101194032 101194033   SV5     2864          + Insertion      -13
## 6     6 101056644 101056645   SV6      265          + Insertion        0
##   query.dyst
## 1        185
## 2        303
## 3          0
## 4       1259
## 5       2851
## 6        265
```

Wczytany plik BED zawiera informacje o lokalizacji wariantów genetycznych w genomie ludzkim. Warianty zlokalizowane są na chromosomach genomowych lub na scaffoldach, które do tej pory nie udało się połączyć z genomem.

```
summary(my_data$chrom)
```

```
##          1         10         11         12         13         14
##        779        481        500        435        361        277
```

```
##          15          16          17          18          19           2
##         238         248         294         274         275         702
##          20          21          22           3           4           5
##         311         190         189         565         530         507
##           6           7           8           9 GL000195.1 GL000214.1
##         568         596         488         371           1           1
## GL000219.1 GL000220.1           X
##           1           5         369
```

Ćwiczenie 1 Proszę przefiltrować tabelę `my_data`, tak aby zawierała wyłącznie wiersze z lokalizacją na chromosomach genomowych.

```
summary(my_data$chrom)
```

```
##          1          10          11          12          13          14
##        779         481         500         435         361         277
##          15          16          17          18          19           2
##         238         248         294         274         275         702
##          20          21          22           3           4           5
##         311         190         189         565         530         507
##           6           7           8           9 GL000195.1 GL000214.1
##         568         596         488         371           0           0
## GL000219.1 GL000220.1           X
##           0           0         369
```

Reorganizujemy dane w odpowiedniej kolejności (na podstawie wartości w kolumnie chrom i typ)

```
my_data$chrom <- factor(my_data$chrom, levels=c(seq(1,22),"X","Y"))
my_data$typ <- factor(my_data$typ, levels=c("Insertion","Deletion","Expansion","Contraction"))
```

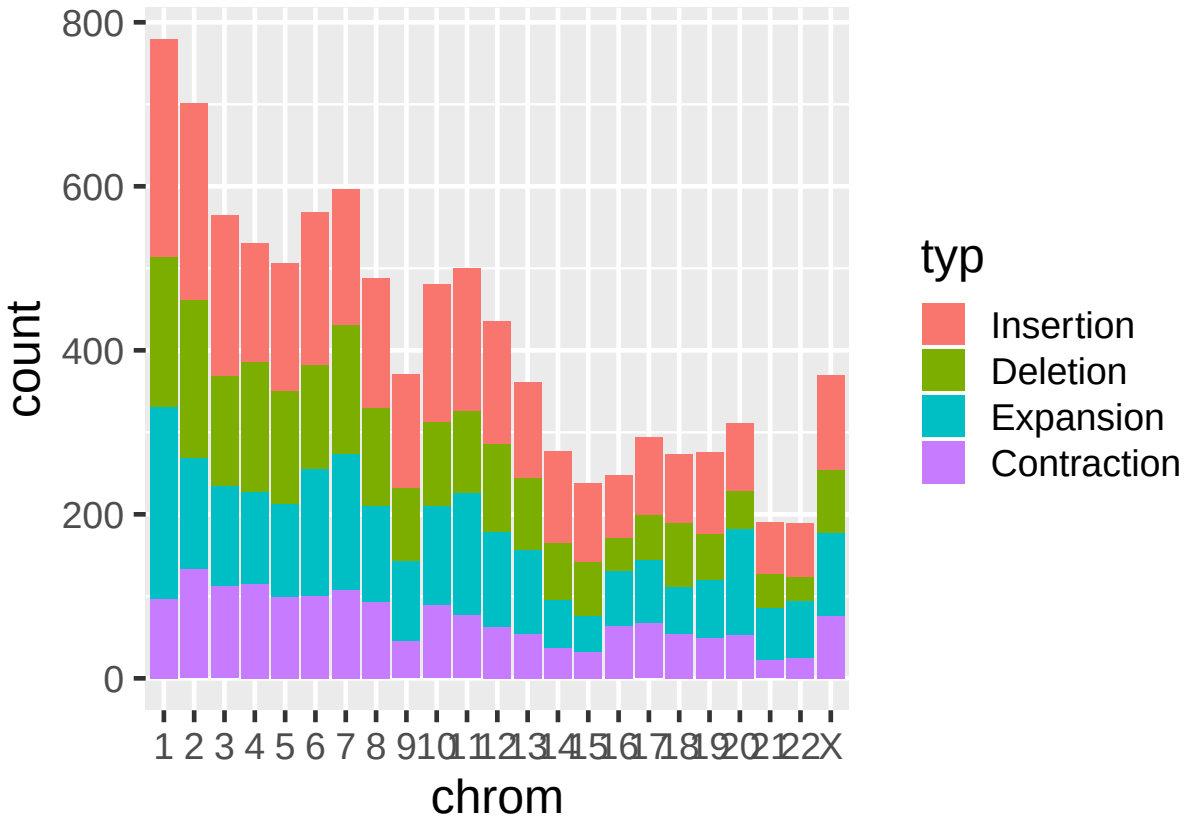
W wyniku tej operacji otrzymujemy:

```
summary(my_data$chrom)
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 779 702 565 530 507 568 596 488 371 481 500 435 361 277 238 248 294 274
##  19  20  21  22   X   Y
## 275 311 190 189 369   0
```

Wykres słupkowy

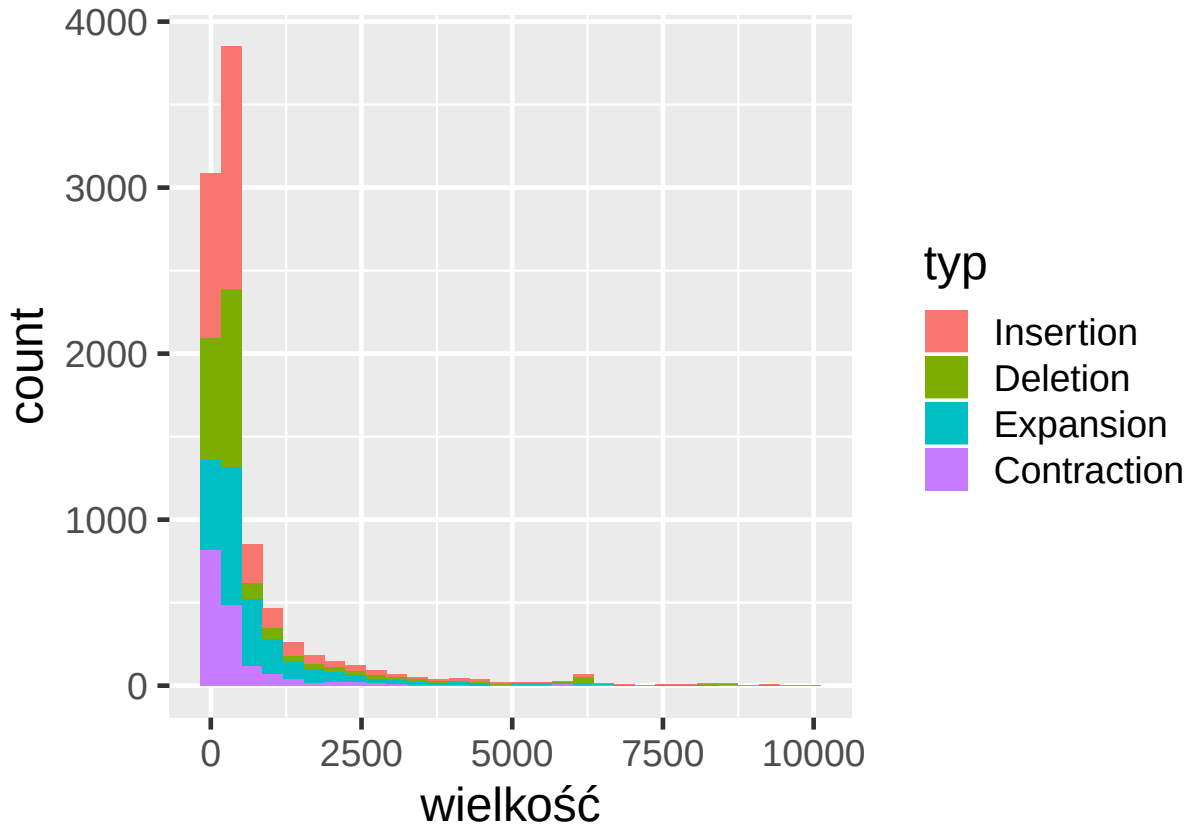
```
ggplot(my_data, aes(x=chrom,fill=typ)) + geom_bar()
```



Możemy stworzyć histogram, wystarczy że zmienimy dane na osi x z "chrom" (wartości kategoryzujące - numer chromosomu) na "wielkość" (wartości liczbowe) oraz stosując funkcję geom_histogram().

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_histogram()
```

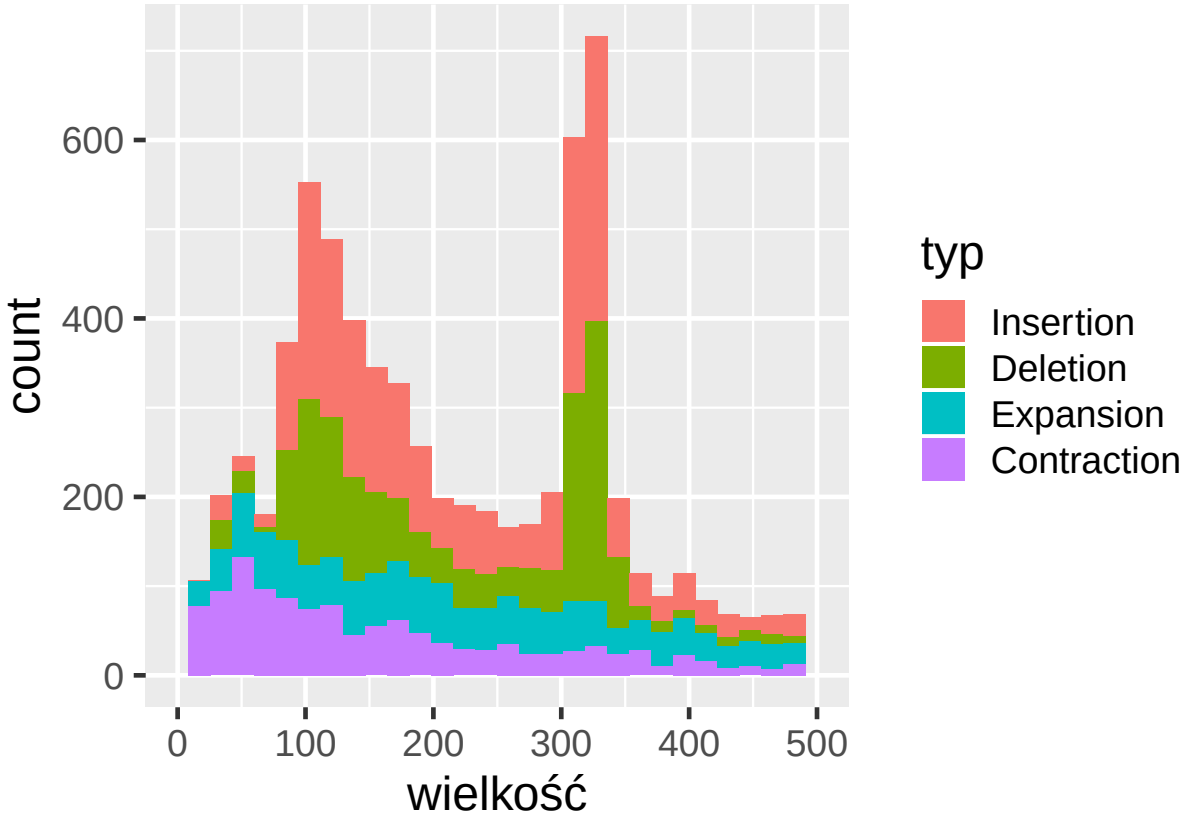
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Możemy ograniczyć wartości na osi (np. do 500), ponieważ większość danych znajduje się w tym przedziale.

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_histogram() + xlim(0,500)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 2660 rows containing non-finite values (stat_bin).
## Warning: Removed 8 rows containing missing values (geom_bar).
```

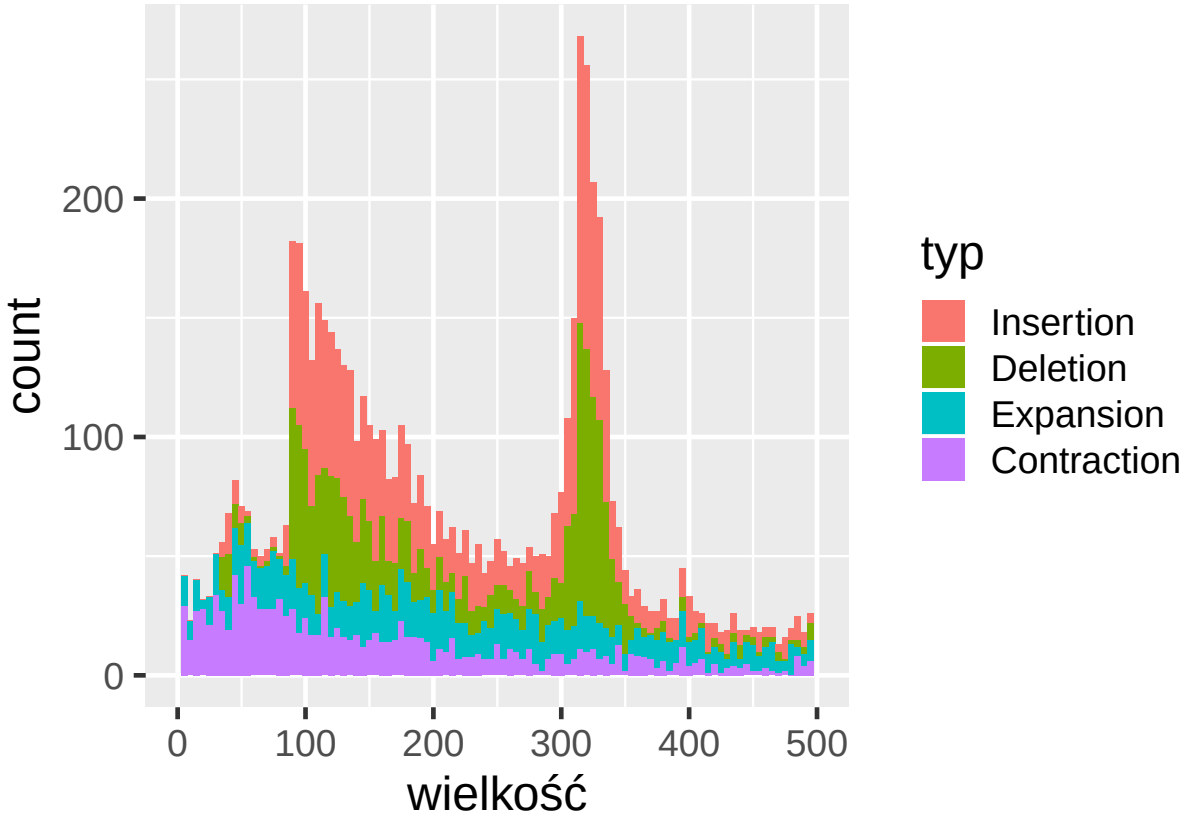


Możemy także poprawić "rozdzielczość" naszego histogramu wykorzystując parametr binwidth.

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_histogram(binwidth=5) + xlim(0,500)
```

```
## Warning: Removed 2660 rows containing non-finite values (stat_bin).
```

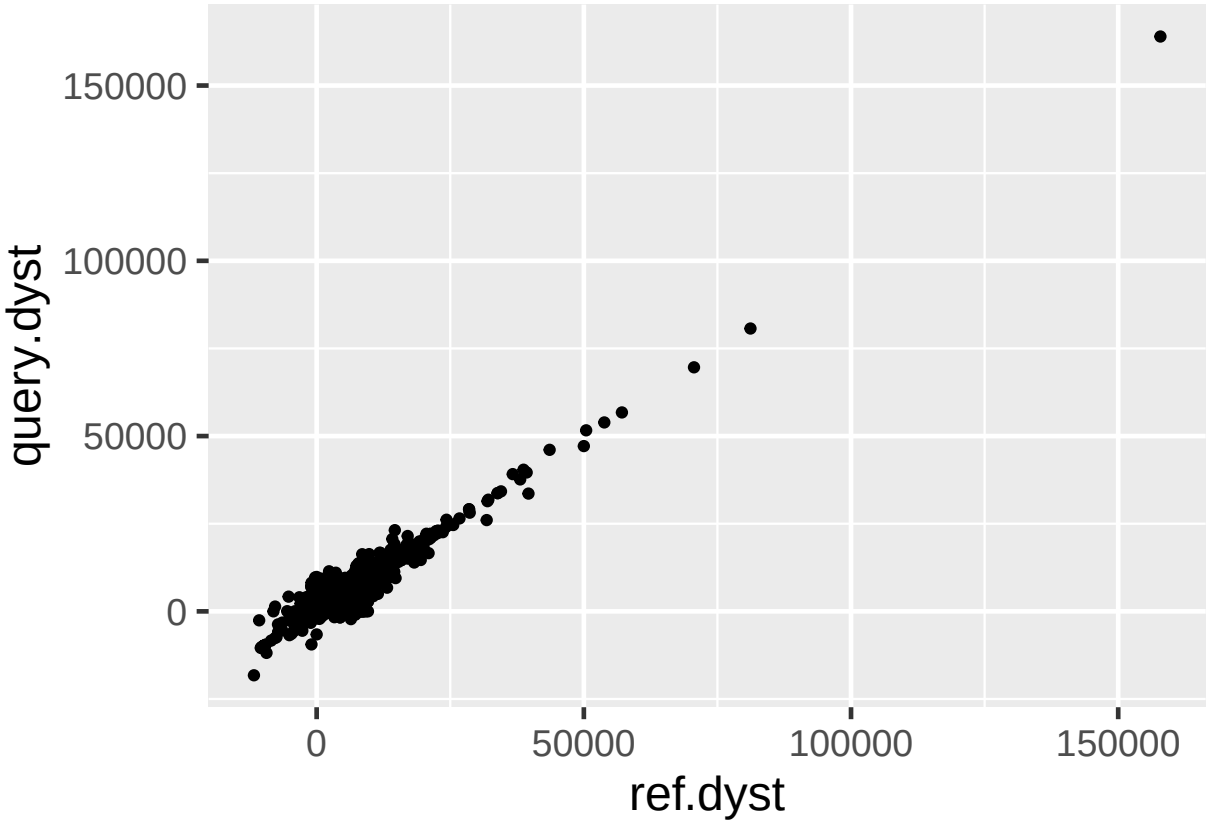
```
## Warning: Removed 8 rows containing missing values (geom_bar).
```



Wykresy punktowe

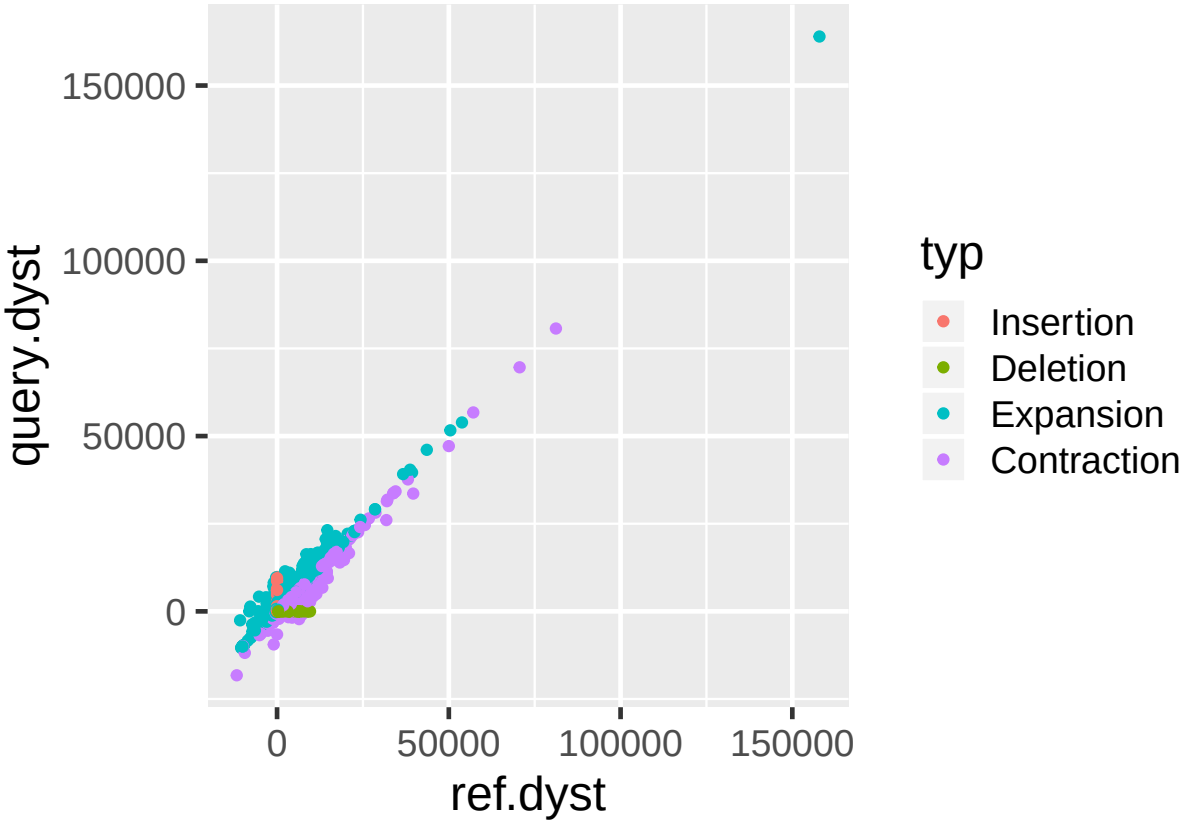
Dla danych liczbowych możemy tworzyć wykresy punktowe.

```
ggplot(my_data, aes(x=ref.dyst,y=query.dyst)) + geom_point()
```



Możemy pokolorować punkty na podstawie danych "kategoryzujących", np z kolumny "typ"

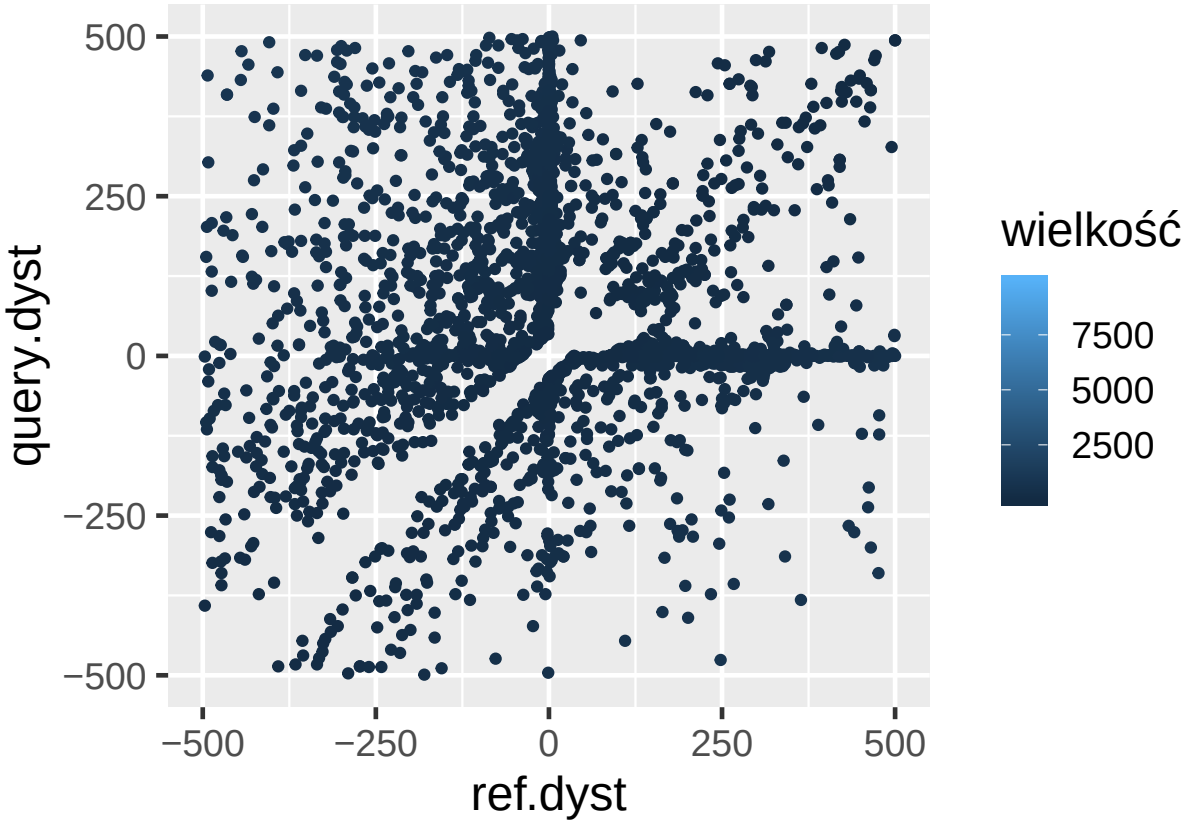
```
ggplot(my_data, aes(x=ref.dyst,y=query.dyst, color=typ)) + geom_point()
```



Możemy pokolorować punkty na podstawie danych liczbowych, np z kolumny “wielkość”, oraz ograniczyć osie (w pewnym przedziale)

```
ggplot(my_data, aes(x=ref.dyst,y=query.dyst, color=wielkość)) +
  geom_point() + xlim(-500,500) + ylim(-500,500)
```

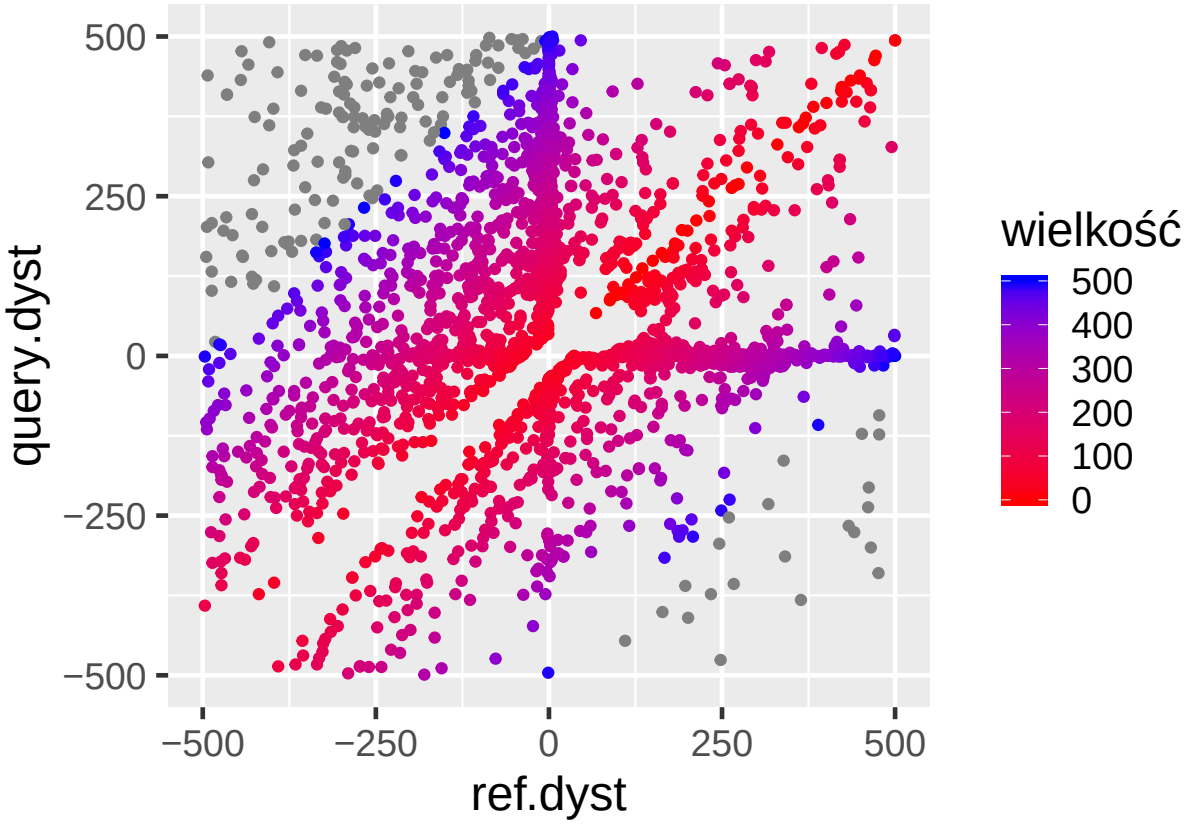
```
## Warning: Removed 3920 rows containing missing values (geom_point).
```



Możemy poprawić nasz wykres stosując funkcję `scale_colour_gradient()`.

```
ggplot(my_data, aes(x=ref.dyst,y=query.dyst, color=wielkość)) +
  geom_point() + xlim(-500,500) + ylim(-500,500) +
  scale_colour_gradient(limits=c(0,500),low = "red", high = "blue")
```

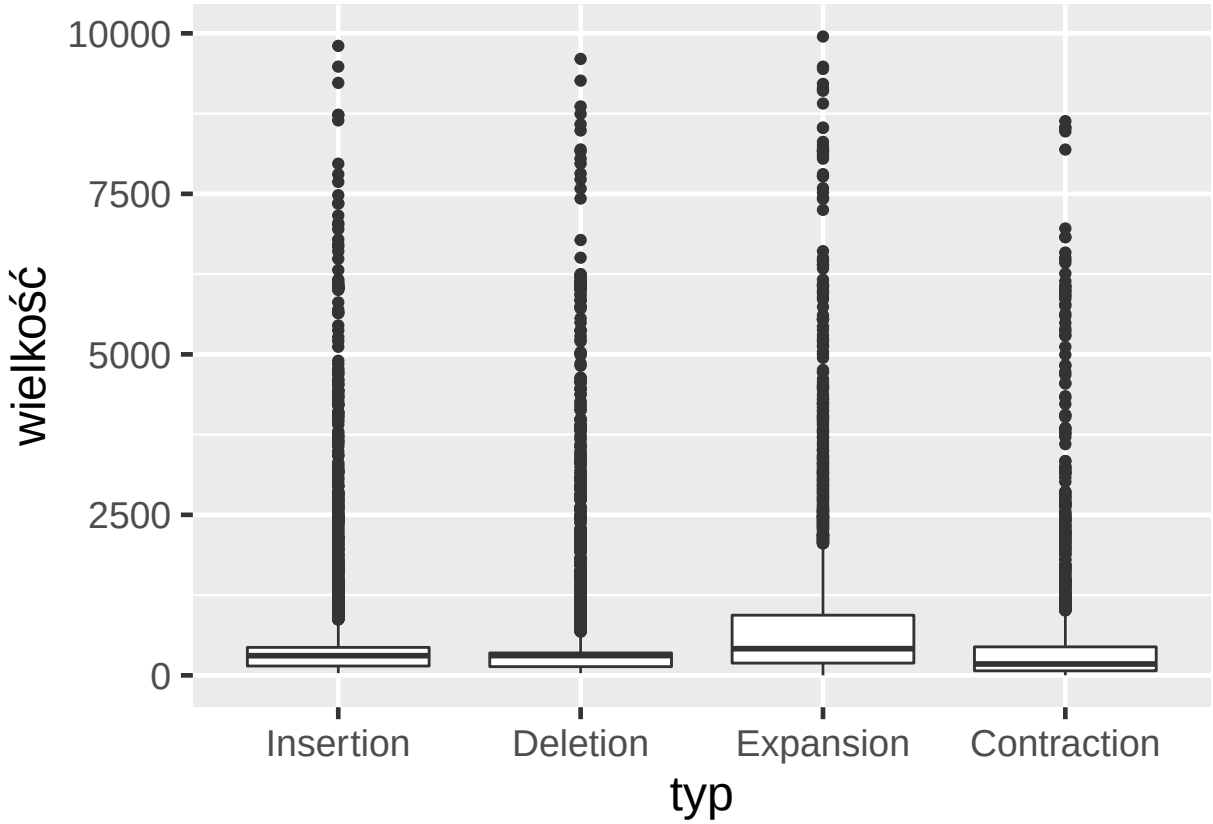
```
## Warning: Removed 3920 rows containing missing values (geom_point).
```



Tworzenie wykresów box-plot

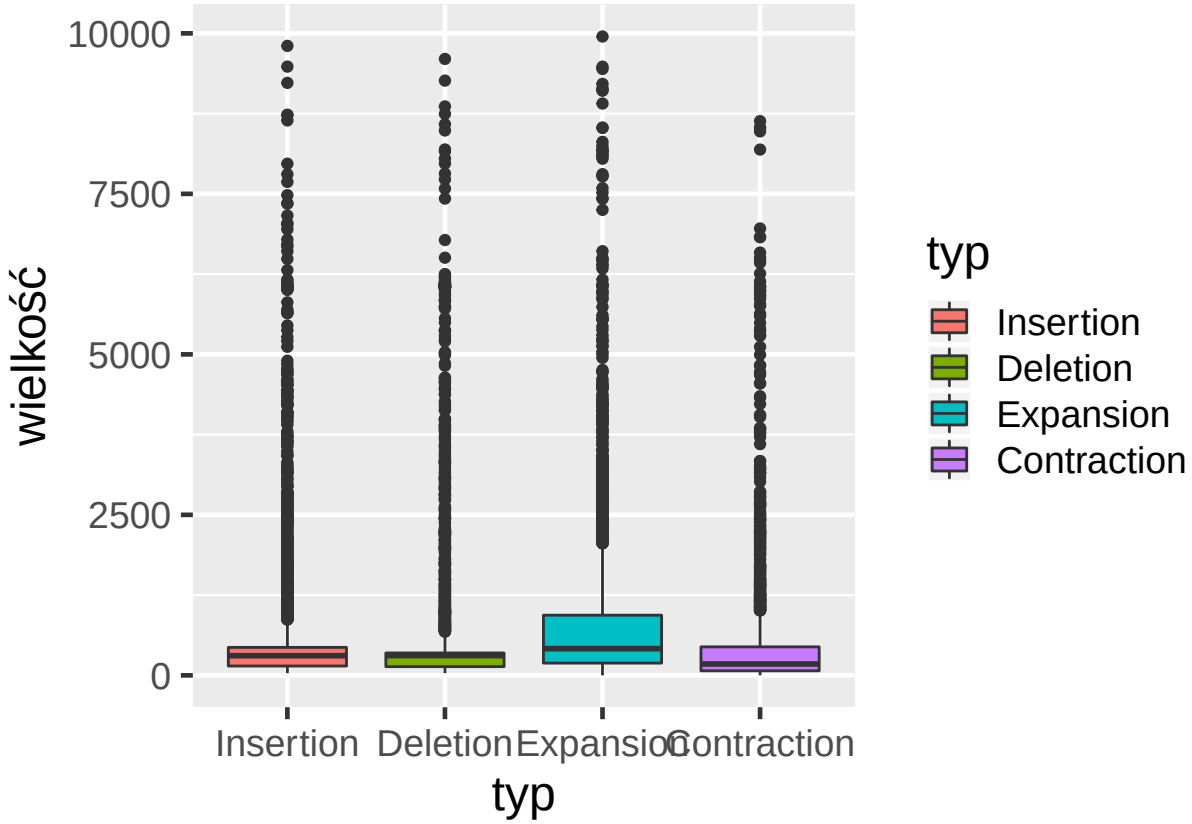
Do tworzenia wykresów box-plot wykorzystujemy funkcję `geom_boxplot()`.

```
ggplot(my_data, aes(x=typ,y=wielkość)) + geom_boxplot()
```



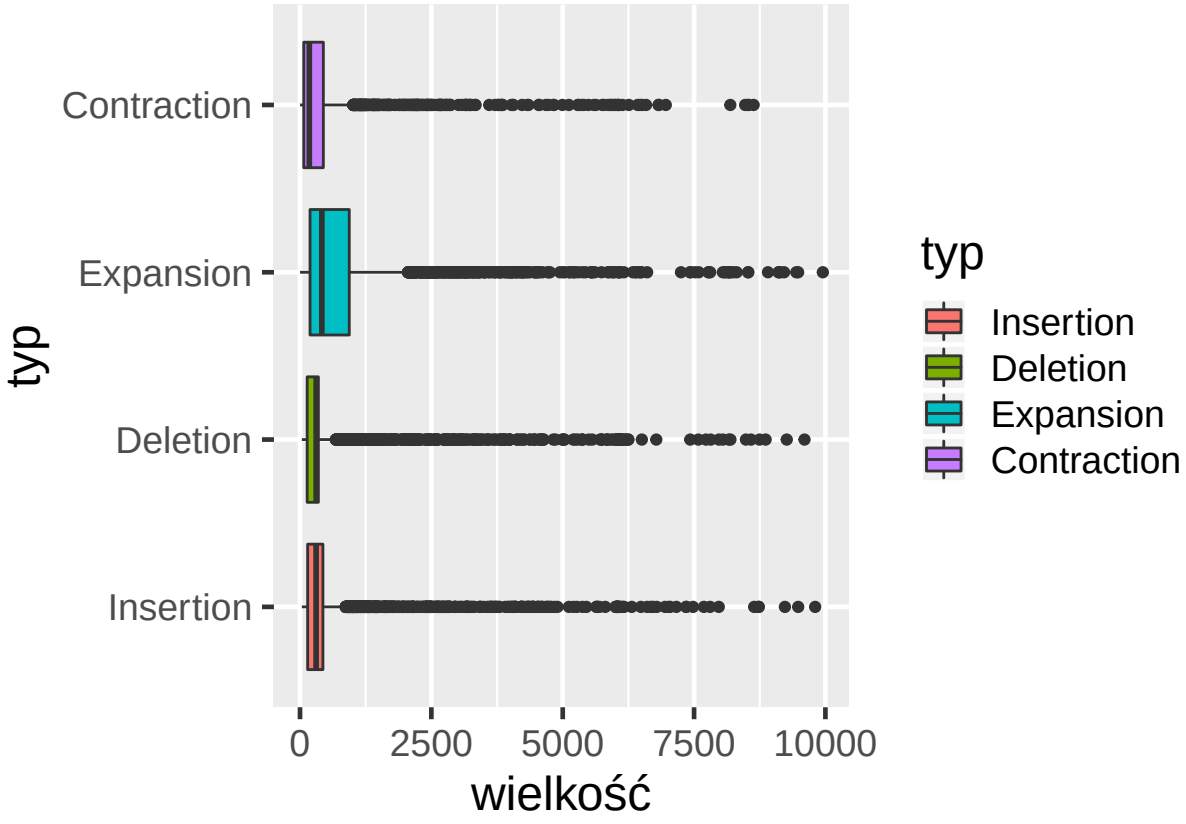
Możemy wyróżnić typy sekwencji.

```
ggplot(my_data, aes(x=typ,y=wielkość, fill=typ)) + geom_boxplot()
```



Możemy obrócić nasz wykres korzystając z funkcji `coord_flip()`

```
ggplot(my_data, aes(x=typ,y=wielkość, fill=typ)) + geom_boxplot() + coord_flip()
```



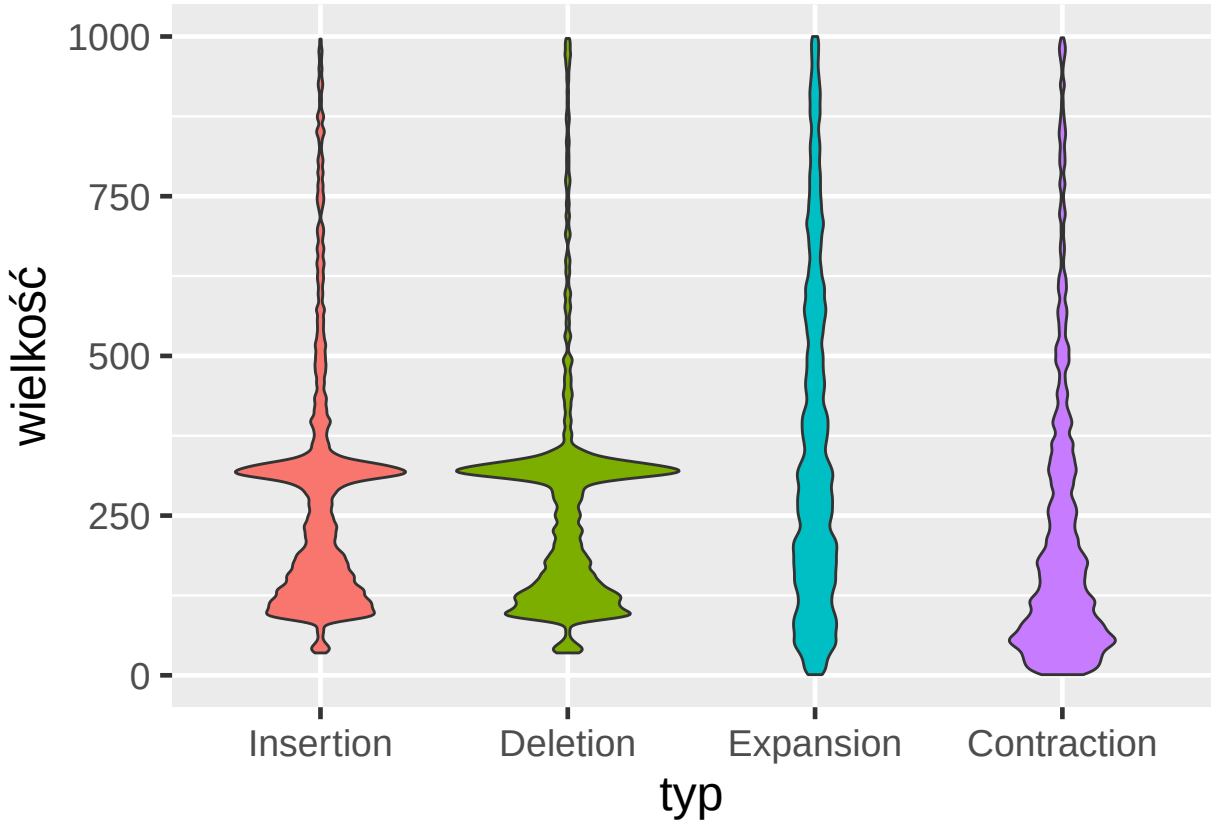
Wykres skrzypcowy (ang. violin plot)

Podobnym to wykresu "box-plot" jest wykres skrzypcowy (ang. violin plot). Ten typ wykresu ilustruje jądrowy estymator gęstości.Służy do tego funkcja geom_violin().

Jądrowy estymator gęstości - jest to rodzaj estymatora nieparametrycznego, przeznaczony do wyznaczania gęstości rozkładu zmiennej losowej, na podstawie uzyskanej próby, czyli wartości jakie badana zmienna przyjęła w trakcie dotychczasowych pomiarów. Silverman (1986). Podobnie jak wykres typu "box-plot", wykresy skrzypcowy służą do porównania rozkładu zmiennych (lub rozkładu próbek) w różnych "kategoriach".

```
ggplot(my_data, aes(x=typ,y=wielkość,fill=typ)) + geom_violin(adjust=0.2) +
  ylim(0,1000) + guides(fill=FALSE)
```

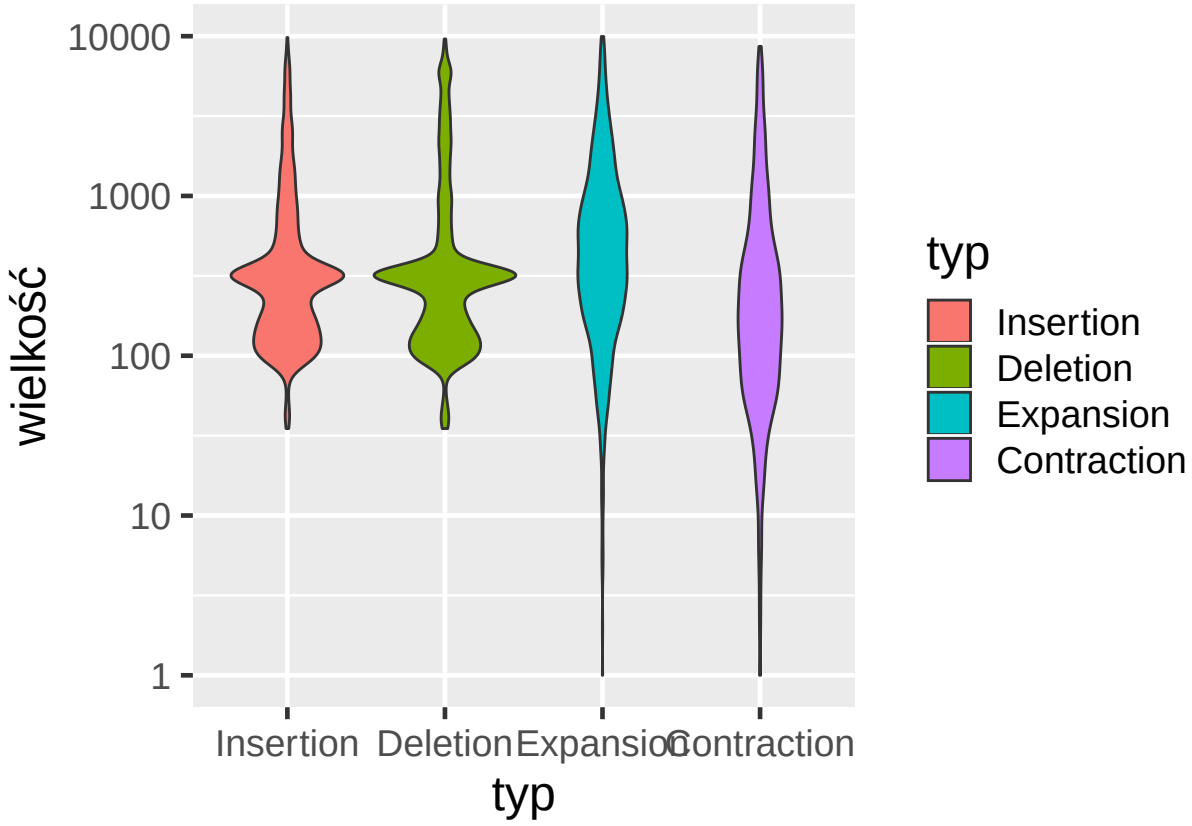
```
## Warning: Removed 1530 rows containing non-finite values (stat_ydensity).
```



Domyślna wartość adjust to 1, niższa wartość oznacza lepszą rozdzielczość.

Możemy zastosować skalę logarytmiczną korzystając z funkcji `scale_y_log10()`.

```
ggplot(my_data, aes(x=typ,y=wielkość,fill=typ)) + geom_violin() + scale_y_log10()
```

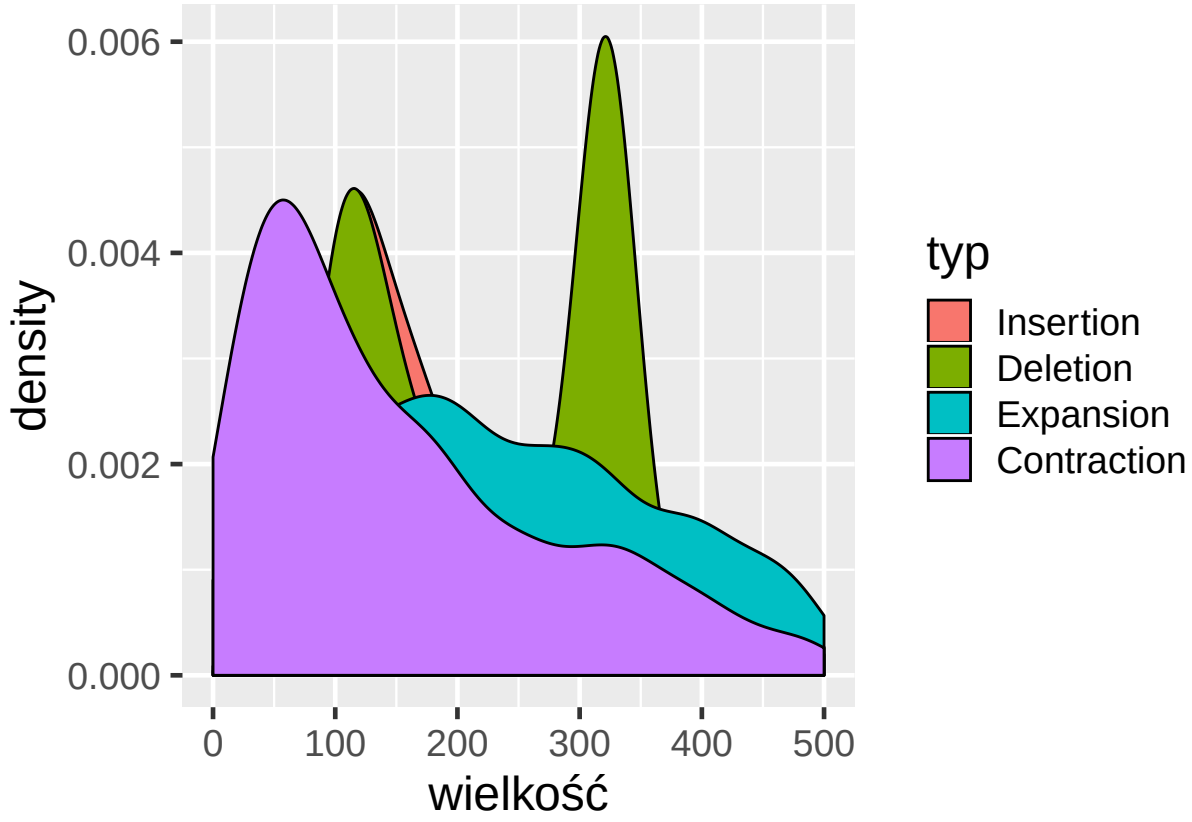


Wykres gęstości

Możemy tworzyć wykresy gęstości dzięki funkcji `geom_density()`. Wykres gęstości jest bardzo podobny do wykresu skrzypkowego, ale dane nie są podzielone na kategorie, leczy umieszczone na jednej osi. Różne kategorie mogą być reprezentowane kolorami.

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_density() + xlim(0,500)
```

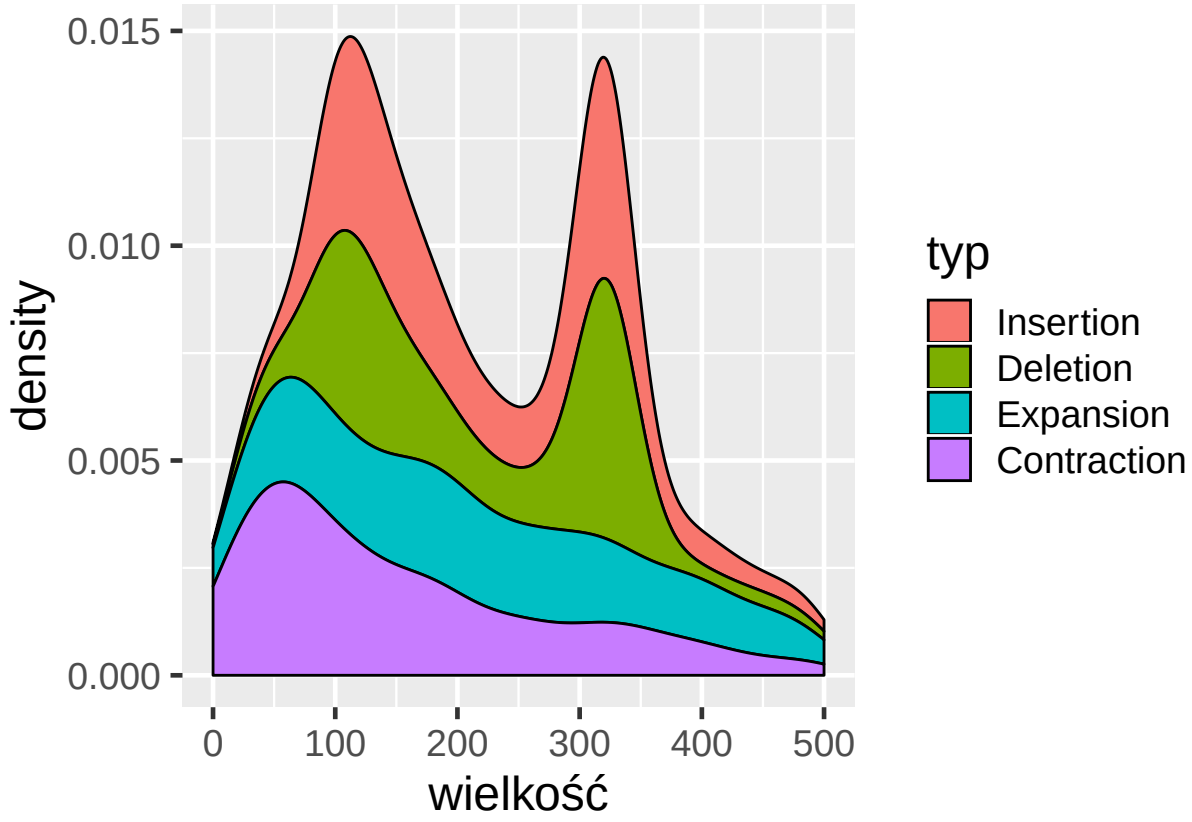
```
## Warning: Removed 2660 rows containing non-finite values (stat_density).
```



Możemy poprawić jakość naszego wykresu stosując parametr position = "stack".

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_density(position = "stack") + xlim(0,500)
```

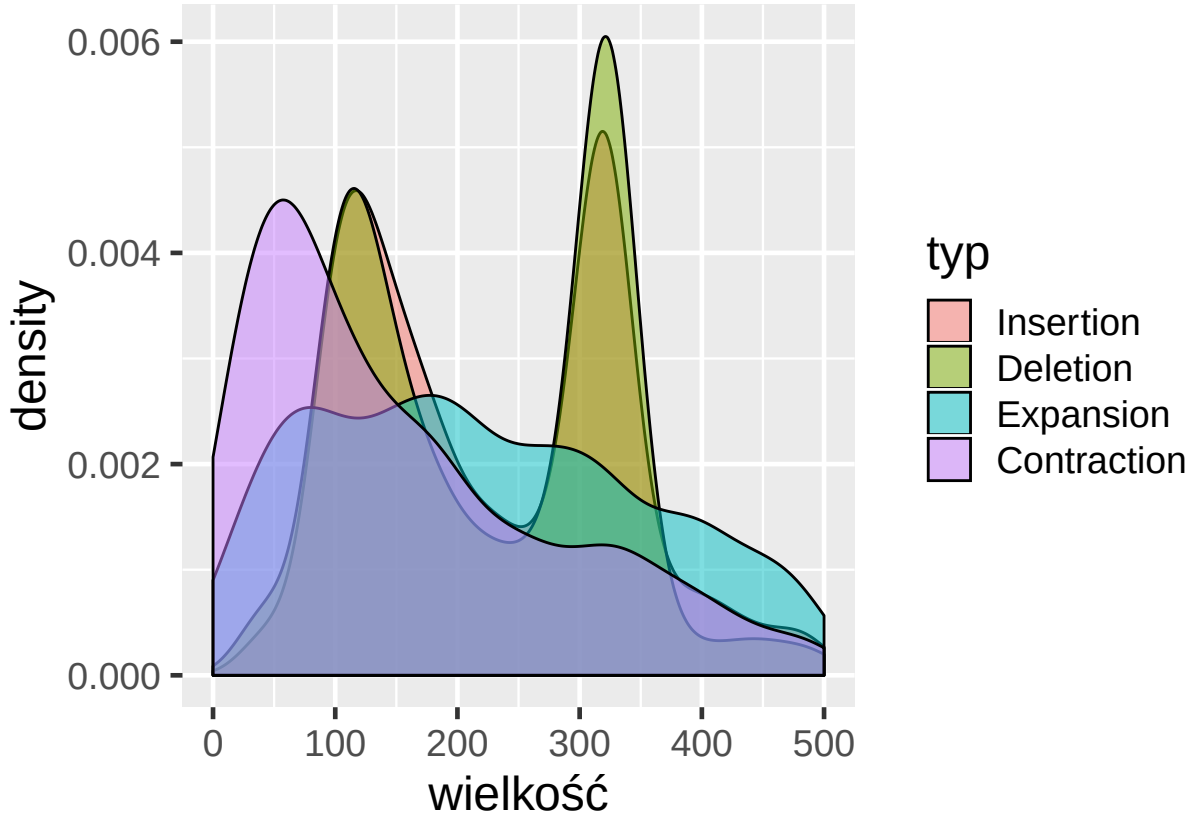
```
## Warning: Removed 2660 rows containing non-finite values (stat_density).
```



Innym rozwiązaniem jest wykorzystanie “przeźroczystości”, parametr “alpha”.

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_density(alpha=.5) + xlim(0,500)
```

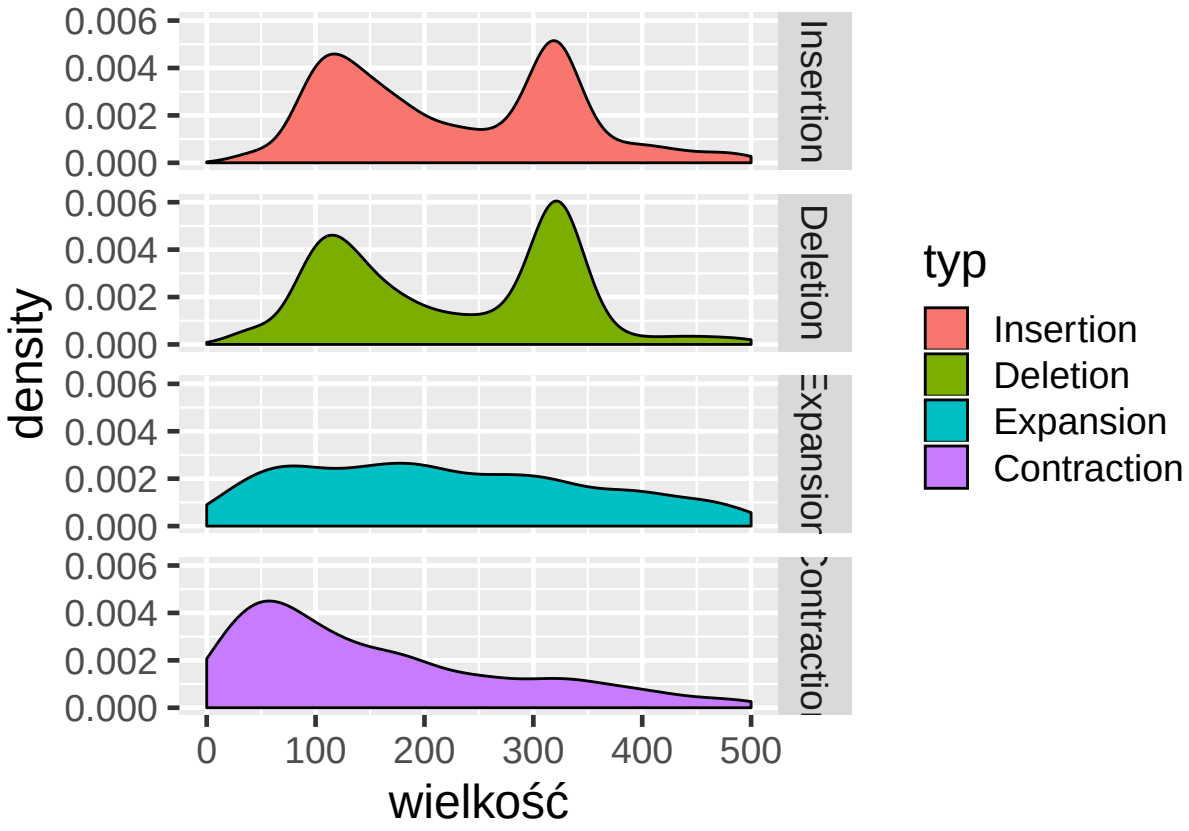
```
## Warning: Removed 2660 rows containing non-finite values (stat_density).
```



Kolejnym sposobem może być wykorzystanie funkcji `facet_grid()`

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_density() +
  xlim(0,500) + facet_grid(typ ~ .)
```

```
## Warning: Removed 2660 rows containing non-finite values (stat_density).
```

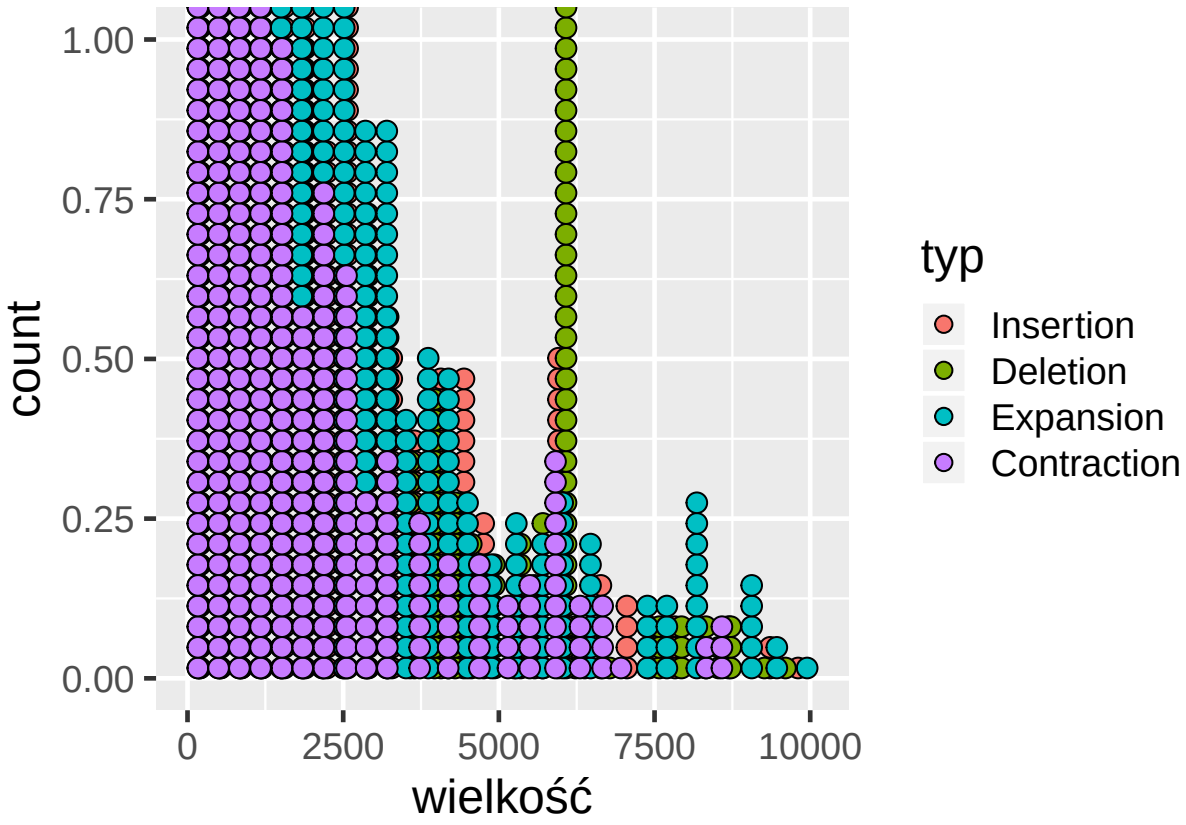


Wykres kropkowy

Kolejny przykład wykresu to wykres kropkowy. Możemy go stworzyć stosując funkcję `geom_dotplot()`. Każda kropka reprezentuje indywidualną próbkę.

```
ggplot(my_data, aes(x=wielkość,fill=typ)) + geom_dotplot()
```

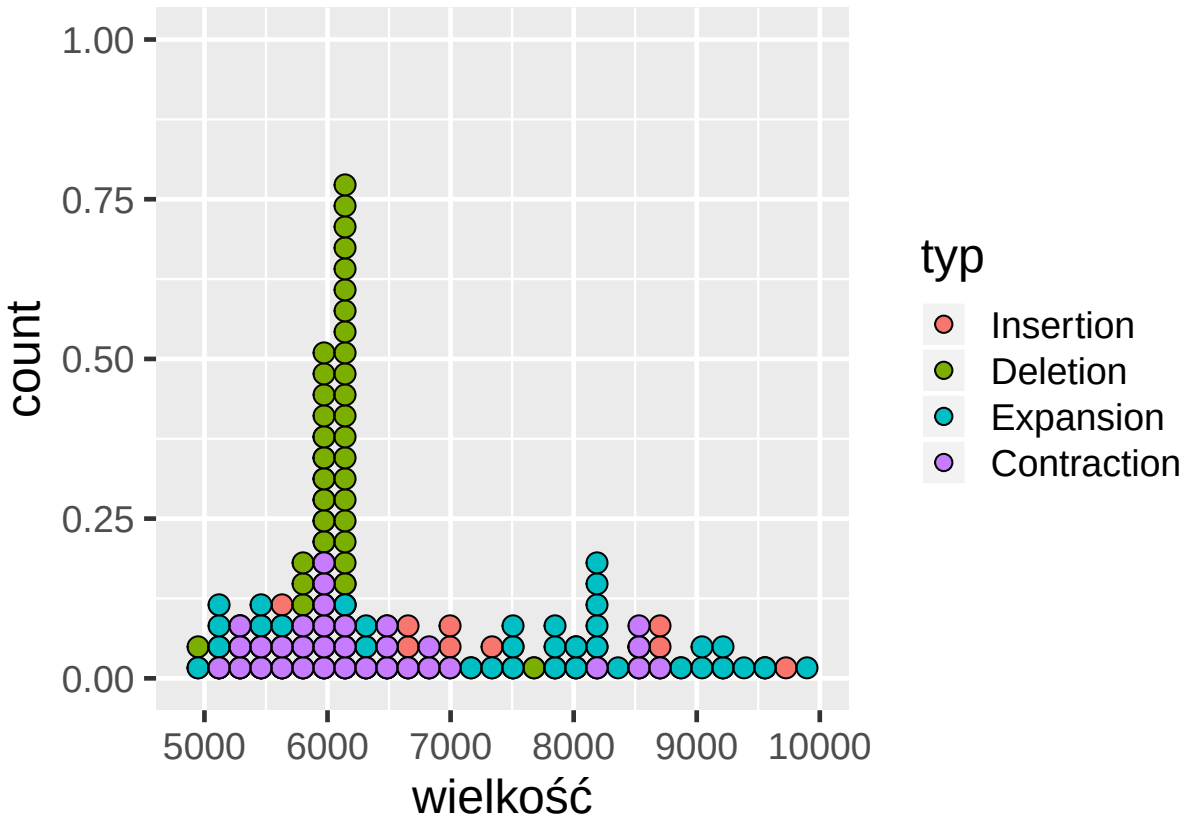
```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



Dobrym rozwiązaniem może być filtracja wyników, żeby ograniczyć ilość uzyskanych wyników. Tego typu wykres może być przydatny jeżeli analizujemy wyniki z niewielką liczbą obserwacji.

```
large_data <- my_data[my_data$wielkość>5000,  ] # [wiersze,kolumny]
ggplot(large_data, aes(x=wielkość,fill=typ)) + geom_dotplot(method="histodot")
```

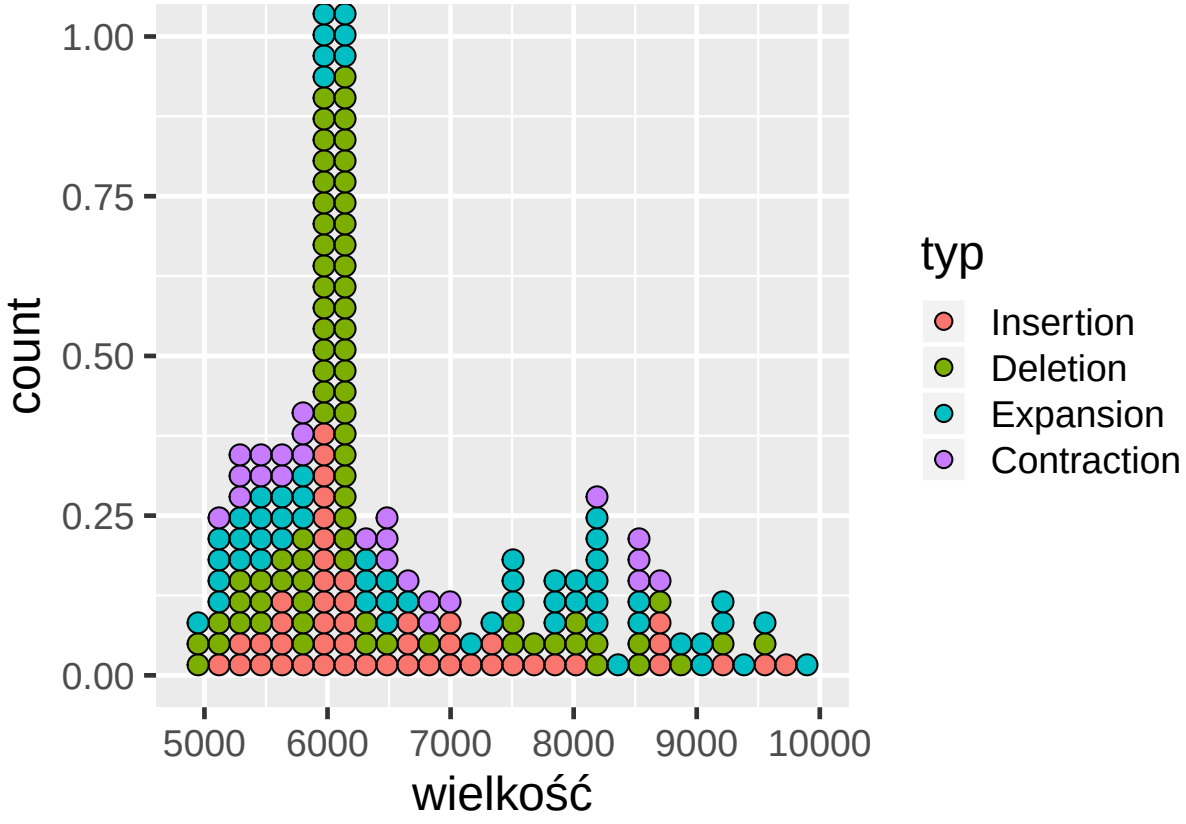
```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



Musimy być uważni ponieważ kropki (obserwacje), nie grupują się automatycznie, więc niektóre mogą być zasłonięte przez inne. Wykresy kropkowe są przydatne kiedy chcemy analizować liczby poszczególnych obserwacji, a ogólna liczba obserwacji jest niewielka. Przykładowo mamy pomiary 20 roślin i chcemy zobaczyć jak poszczególne rośliny różnią się między sobą.

```
large_data <- my_data[my_data$wielkość>5000,  ] # [wiersze,kolumny]
ggplot(large_data, aes(x=wielkość,fill=typ)) +
  geom_dotplot(method="histodot", stackgroups=T)
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



Wizualizacja serii pomiarów (np. pomiar cechy w różnych czasach)

W tej części będziemy wykorzystywać plik `seria_pomiarów_czas.txt`, który można pobrać ze strony katedry.

```
plik <- "seria_pomiarow_czas.txt"
seria <- read.csv(plik, quote='', stringsAsFactors=TRUE)
head(seria)
```

```
##   sek wartosc probka
## 1   0    0.00      A
## 2   1    5.97      A
## 3   2   13.42      A
## 4   3   56.08      A
## 5   4   98.04      A
## 6   5   27.11      A
```

Do wizualizacji tego typu danych wykorzystujemy wykres liniowy (`funkcja geom_line()`)

```
ggplot(seria, aes(x=sek,y=wartosc,colour=probka)) + geom_line(size=3)
```

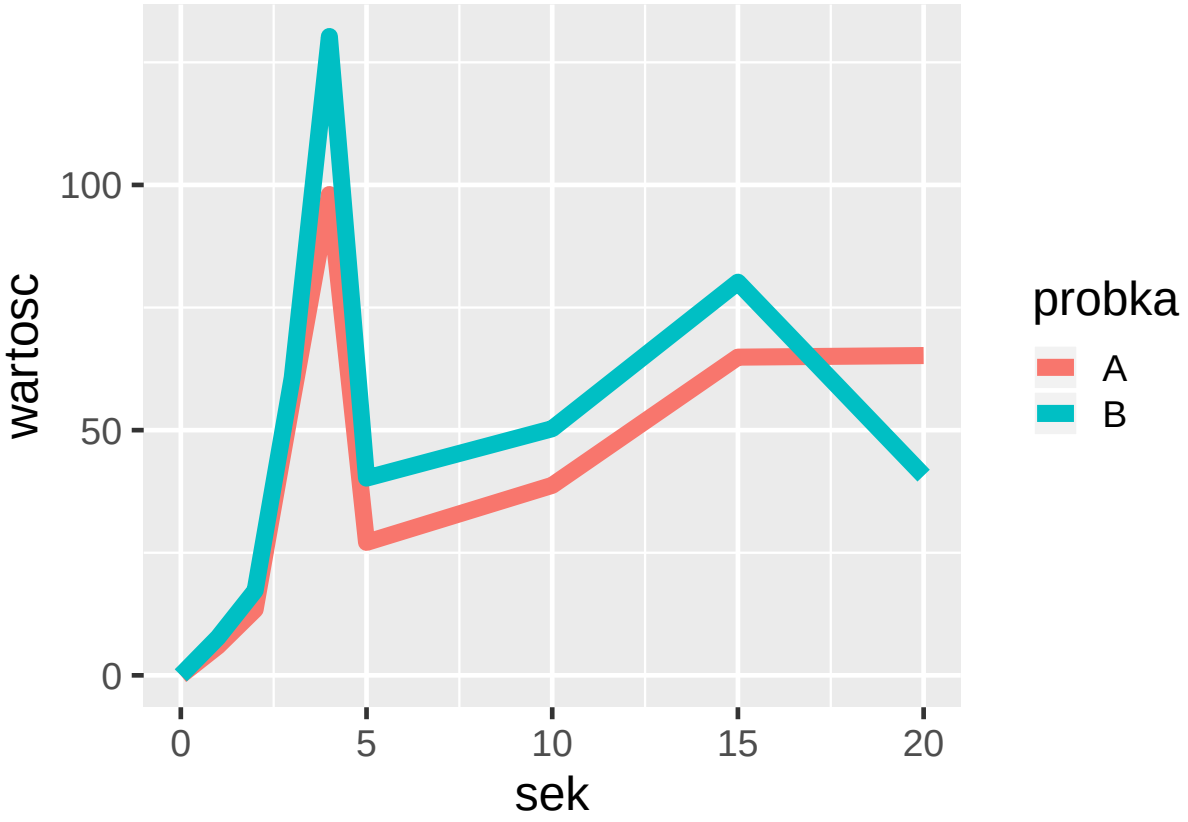


Diagram Venna

Diagram Venna – schemat, służący ilustrowaniu zależności między zbiorami. Ma postać figur geometrycznych na płaszczyźnie. Zbiory reprezentowane są na ogół przez koła lub elipsy.

W tym przykładzie wykorzystamy pliki `geny_lista_A.txt`, `geny_lista_B.txt`, `geny_lista_C.txt` oraz `geny_lista_D.txt`, które można pobrać ze strony katedry.

```
library(VennDiagram)
```

```
## Loading required package: grid
```

```
## Loading required package: futile.logger
```

```
listaA <- read.csv("geny_lista_A.txt",header=FALSE)
A <- listaA$V1
head(A)
```

```
## [1] ABI1  ABL1  ABL2  ACSL3 ACSL6 AFF1
## 537 Levels: ABI1 ABL1 ABL2 ACSL3 ACSL6 AFF1 AFF3 AFF4 AKAP9 AKT1 ... ZRSR2
```

```
listaB <- read.csv("geny_lista_B.txt",header=FALSE)
B <- listaB$V1
listaC <- read.csv("geny_lista_C.txt",header=FALSE)
C <- listaC$V1
listaD <- read.csv("geny_lista_D.txt",header=FALSE)
D <- listaD$V1
length(A)
```

```
## [1] 537
```

```
length(B)
```

```
## [1] 474
```

```
length(C)
```

```
## [1] 2273
```

```
length(D)
```

```
## [1] 1829
```

```
venn.diagram(list(A = A, B = B, C = C, D = D),
             fill = c("yellow","red","cyan","forestgreen"), cex = 1.5,
             filename="Venn_diagram_geny_4.png", imagetype = "png")
```

```
## [1] 1
```

Ćwiczenie 2

W pliku count.txt (do pobrania ze strony katedry) znajdują się liczby odczytów dla poszczególnych transkryptów w trzech tkankach łubinu wąskolistnego (kwiat, liść, łodyga). Należy stworzyć diagram Venna ilustrujący liczbę transkryptów wspólnych (ulegających ekspresji w dwóch lub trzech tkankach) oraz liczbę transkryptów charakterystycznych (z ekspresją tylko w danej tkance).

Heat Mapy - wykresy do ilustracji różnic w ekspresji genów między tkankami

Do stworzenia heatmapy wykorzystamy pakiet "ComplexHeatmap" z biblioteki Bioconductor.

Bioconductor to projekt oparty o open source open developmnet. Bioconductor jest biblioteką zawierającą powiązane ze sobą pakiety, wykorzystywane w analizie danych biologicznych (głównie genomika i transkryptomika). W skład Bioconductora wchodzi 1477 pakietów (stan luty 2018).

Więcej na temat Bioconductora w dalszej części kursu.

W dalszej części będziemy korzystać z pliku `copy_number_data.txt` do pobrania ze strony katedry.

```
library(ComplexHeatmap)
```

```
## ========================================
## ComplexHeatmap version 2.0.0
## Bioconductor page: http://bioconductor.org/packages/ComplexHeatmap/
## Github page: https://github.com/jokergoo/ComplexHeatmap
## Documentation: http://jokergoo.github.io/ComplexHeatmap-reference
##
## If you use it in published research, please cite:
## Gu, Z. Complex heatmaps reveal patterns and correlations in multidimensional
##   genomic data. Bioinformatics 2016.
## ========================================
```

```
plik <- "copy_number_data.txt"
my_data <- read.table(plik, sep="\t", quote="", stringsAsFactors=FALSE,header=TRUE)
my_data[c(1:5),c(1:5)]
```

```
##    CHR     START       END X.SRR089523. X.SRR089526.
## 1 chr1  53440429  53494914    1.1002112    1.1000844
## 2 chr1 105459037 105514187    1.1860780    0.6553897
## 3 chr1 183998520 184048557    1.3791250    1.2024487
## 4 chr1 236135655 236186012    0.8987158    1.1182392
```

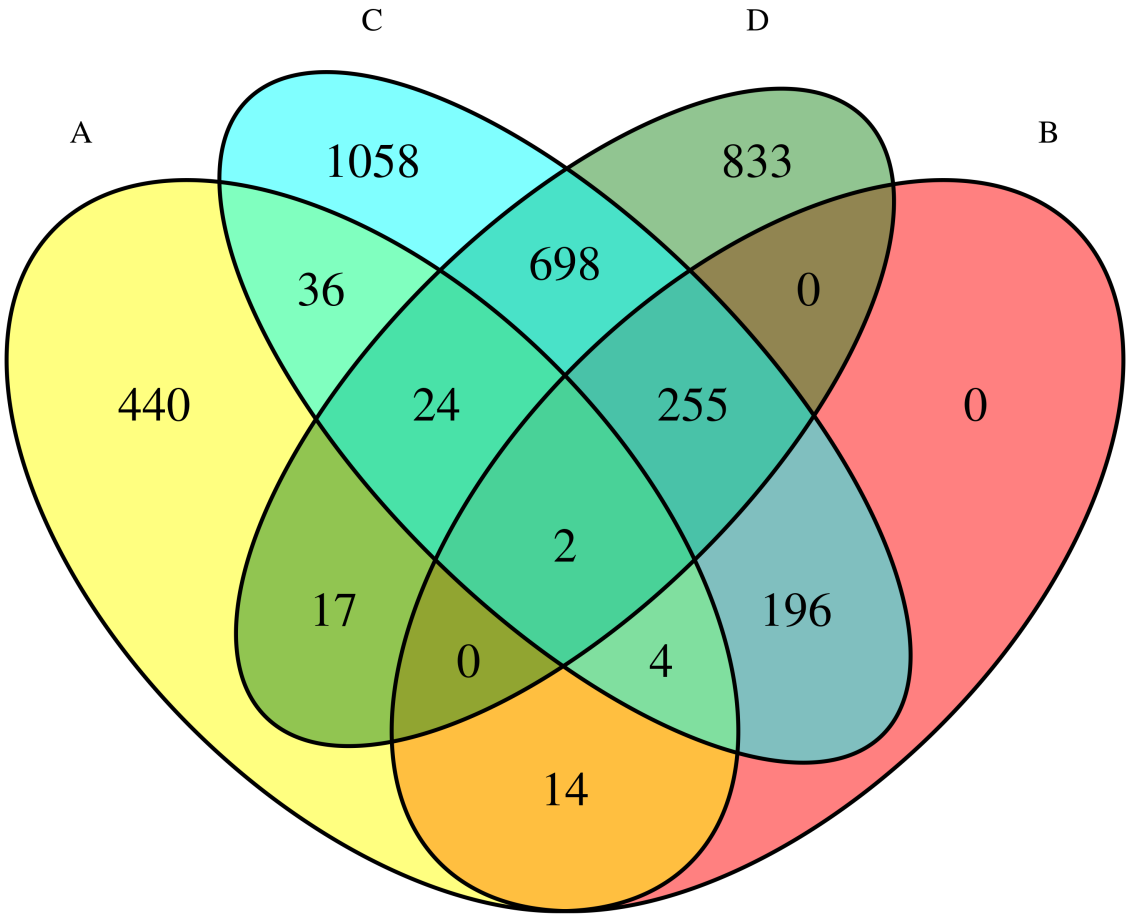


Figure 1: Diagram Venna

```
## 5 chr2  38944803  38996507    1.0788216    0.9663390
```

```
dim(my_data)
```

```
## [1]  53 103
```

```
nrow(my_data) #lokalizacje (bins) w genomie
```

```
## [1] 53
```

```
ncol(my_data) #poszczególe próbki (komórki)
```

```
## [1] 103
```

W pierwszej kolejności musimy przekształcić nasze dane na matrix, zawierającą tylko dane na temat poziomu ekspresji. Informacje z trzech pierwszych kolumn pomijamy.

```
my_matrix <- as.matrix(my_data[,c(4:103)]) # wszystkie wiersze, bez 3 pierwszych kolumn
class(my_data)
```

```
## [1] "data.frame"
```

```
class(my_matrix)
```

```
## [1] "matrix"
```

```
dim(my_matrix)
```

```
## [1]  53 100
```

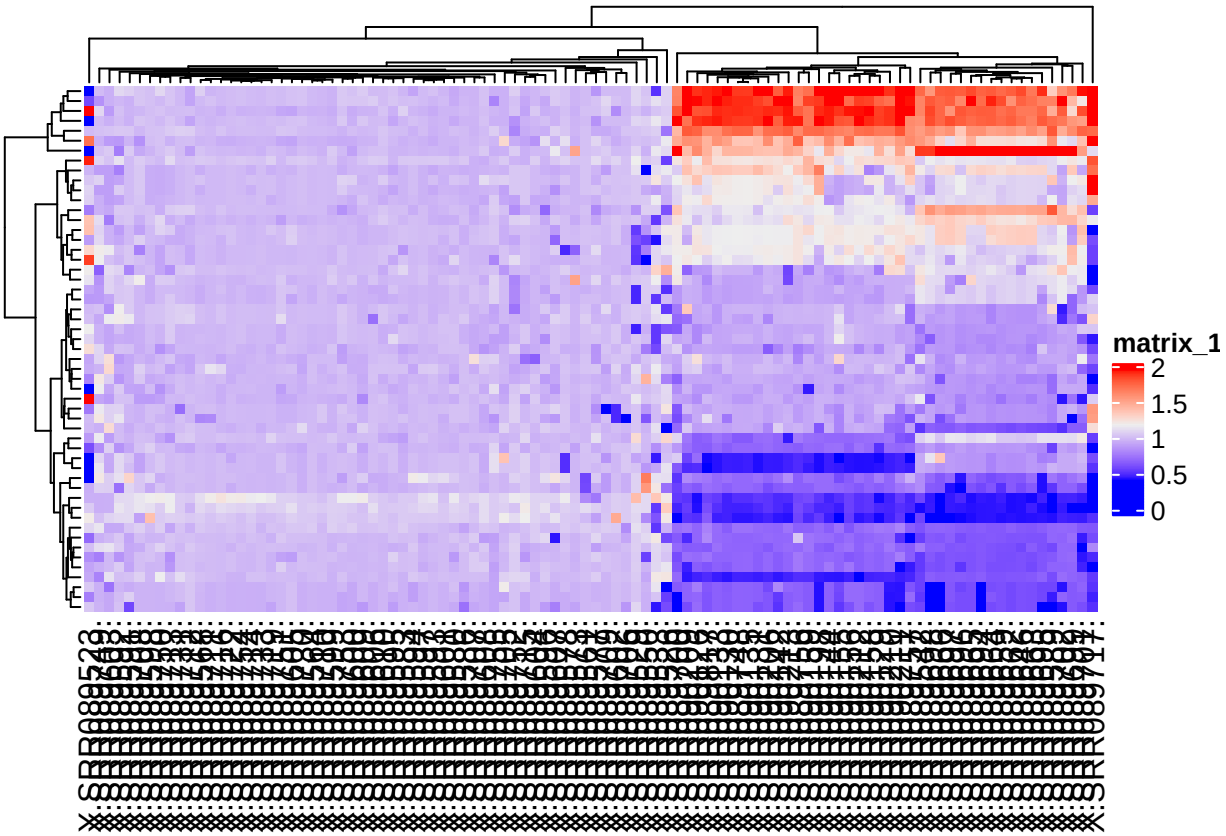
Zapisujemy informacje o lokalizacji chromosomowej, będzie nam potrzebne w dalszej części.

```
chrom <- data.frame(chrom = my_data$CHR)
```

Tworzymy pierwszą heat mape.

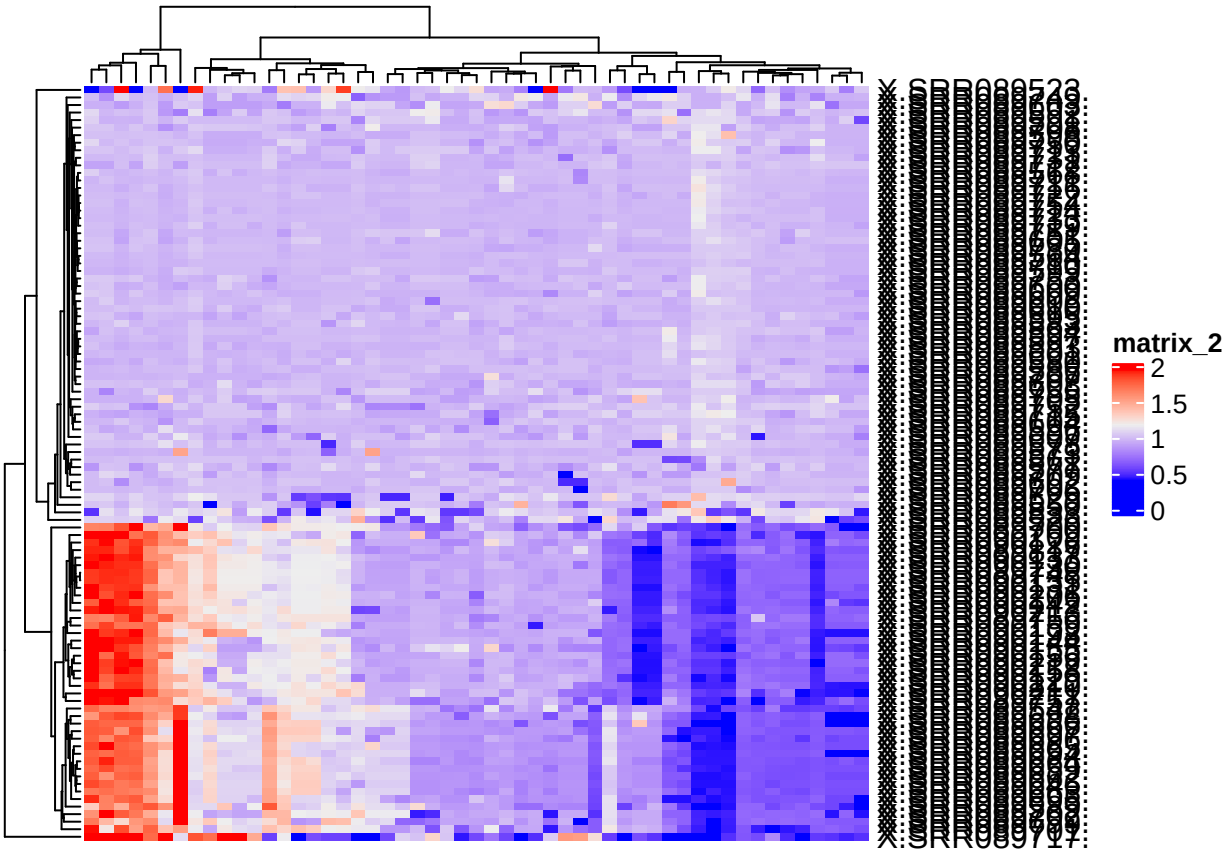
Wszystkie parametry mają wartości domyślne

```
Heatmap(my_matrix)
```



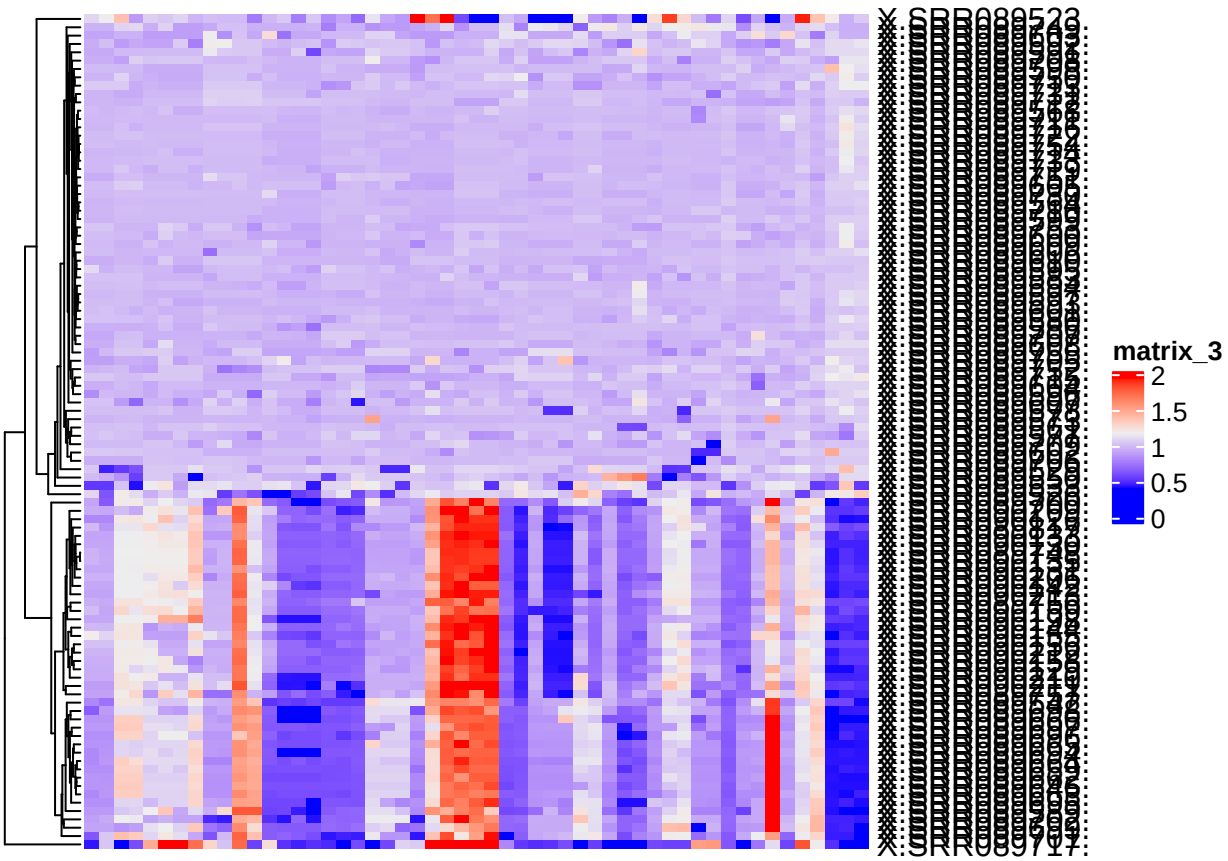
Zamieńmy wiersze z kolumnami (transpozycja)

```
my_matrix <- t(my_matrix)
Heatmap(my_matrix)
```



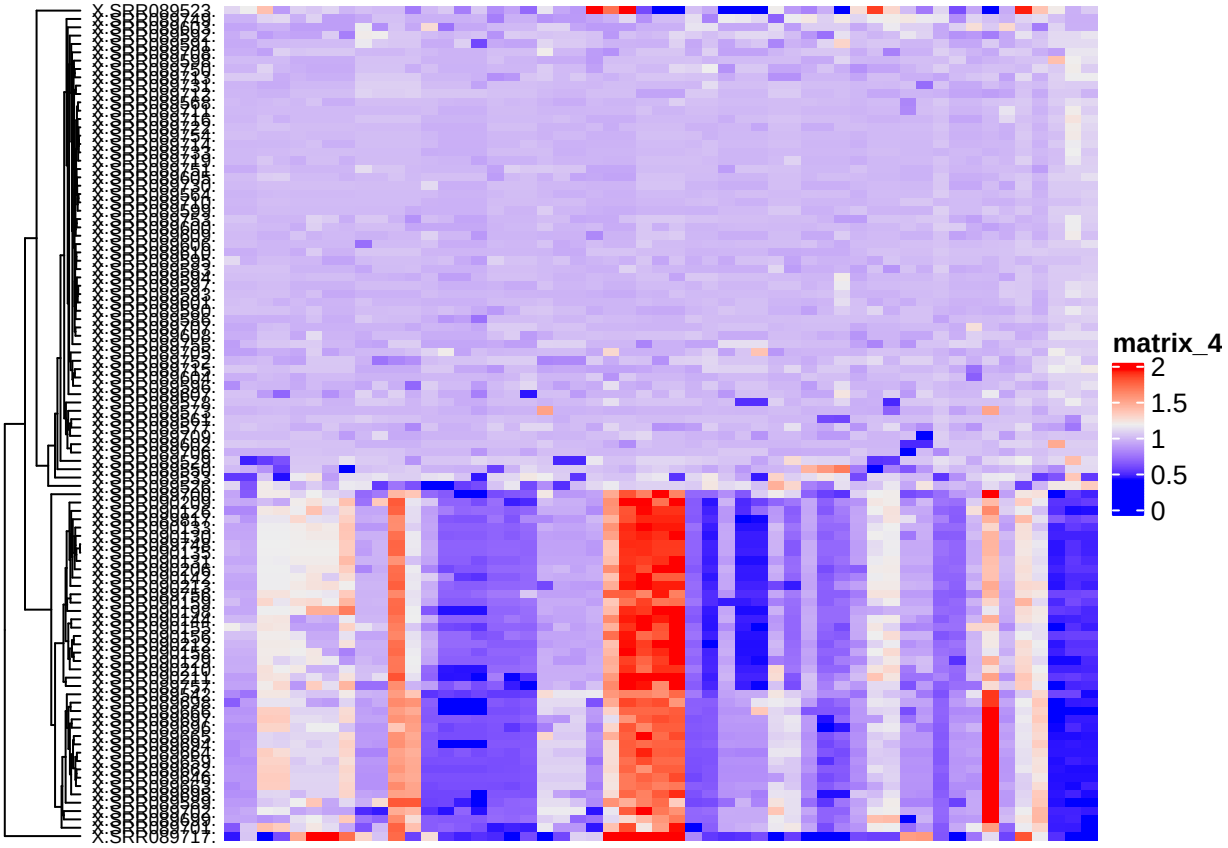
Heat mapa bez klastrów

```
Heatmap(my_matrix, cluster_columns = F)
```



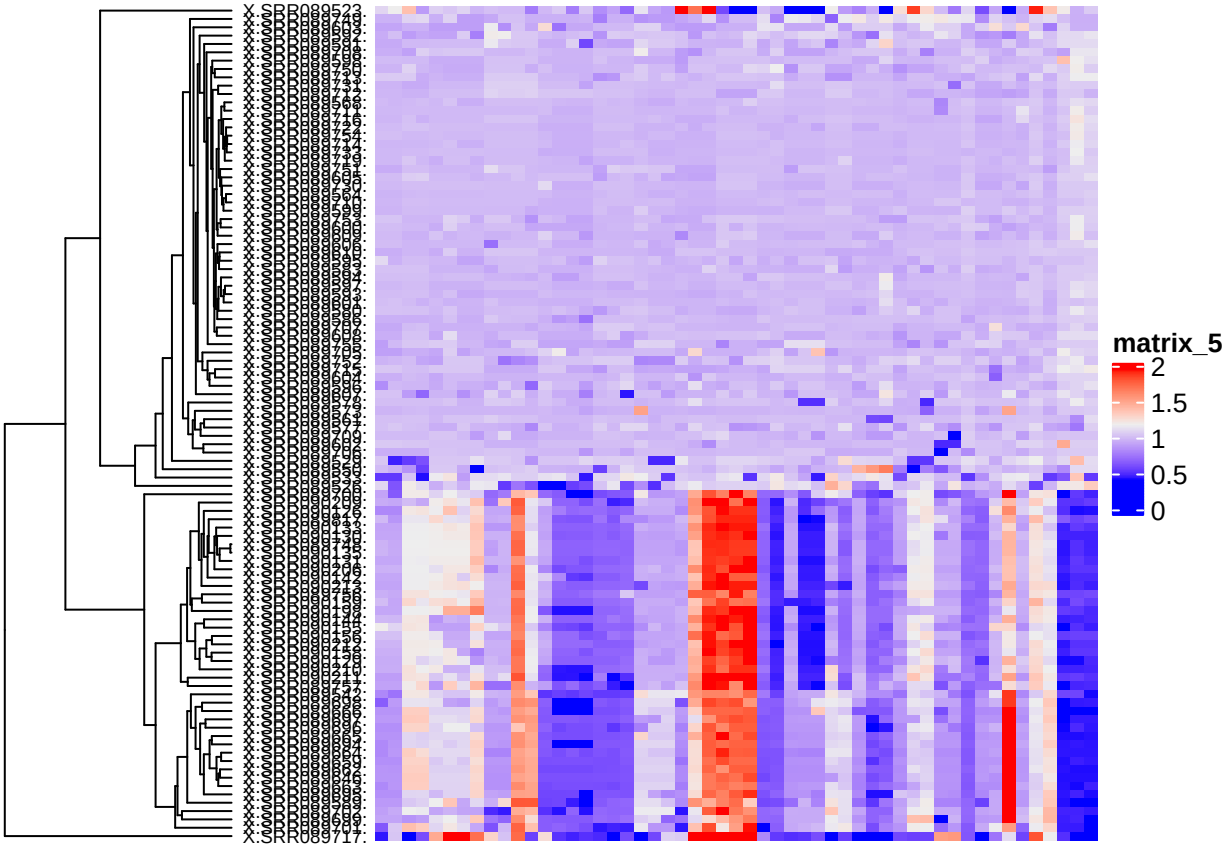
Przenosimy nazwy na lewą stronę

```
Heatmap(my_matrix, cluster_columns=FALSE,
        row_names_side = "left",
        row_dend_side = "left",
        row_names_gp=gpar(cex=0.6))
```



Rozszerzamy dendrogram

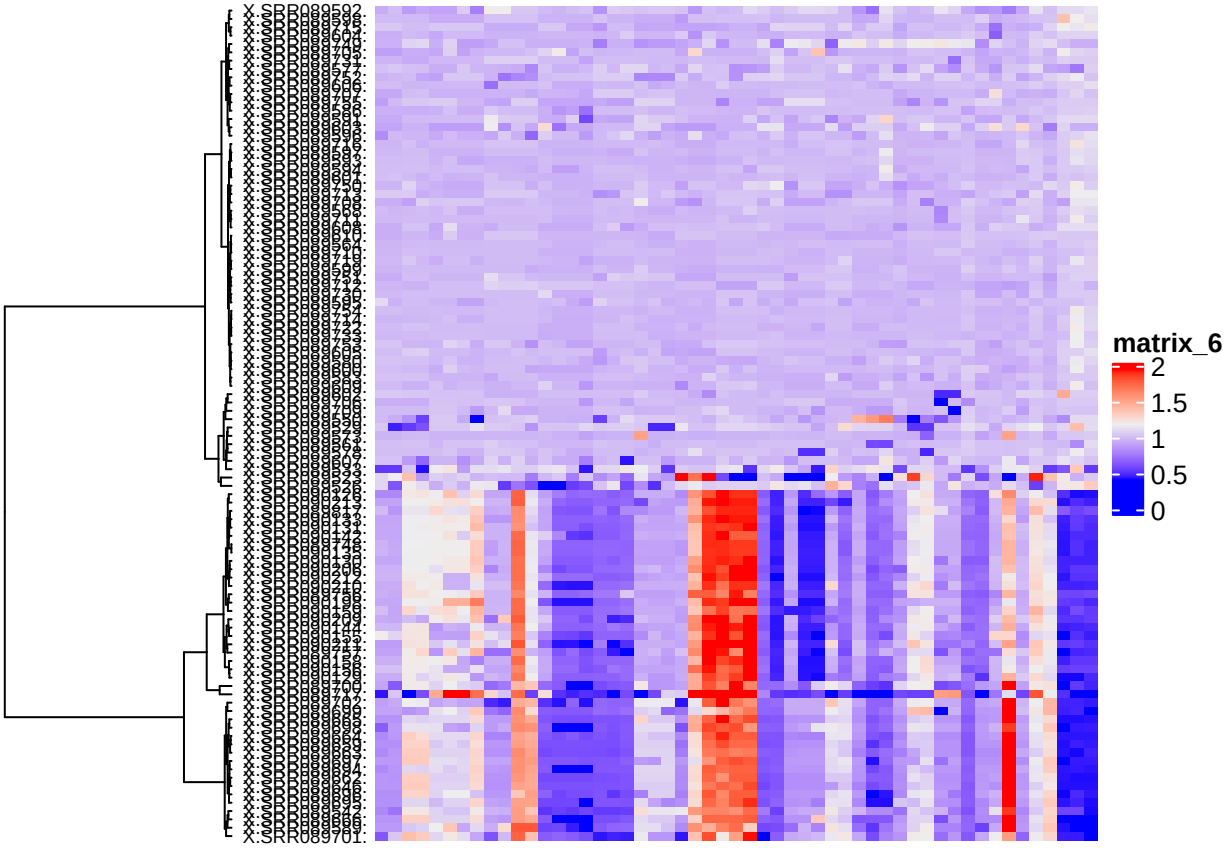
```
Heatmap(my_matrix, cluster_columns=FALSE,
        row_names_side = "left",
        row_dend_side = "left",
        row_names_gp=gpar(cex=0.6),
        row_dend_width = unit(3, "cm"))
```



Grupowanie

Możemy zmienić metodę (algorytm) grupowania (metoda obliczania dystansu, metoda grupowania)

```
Heatmap(my_matrix,
        cluster_columns=FALSE,
        row_names_side = "left",
        row_dend_side = "left",
        row_names_gp=gpar(cex=0.6),
        row_dend_width = unit(3, "cm"),
        clustering_distance_rows ="maximum",
        clustering_method_rows = "ward.D")
```



Bardziej zaawansowana modyfikacja heat mapy, kolorowanie dendrogramów, podział na klastry.

```
library(dendextend)
```

```
##
## ---------------------
## Welcome to dendextend version 1.12.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------
```

```
##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:VennDiagram':
##
##     rotate
```

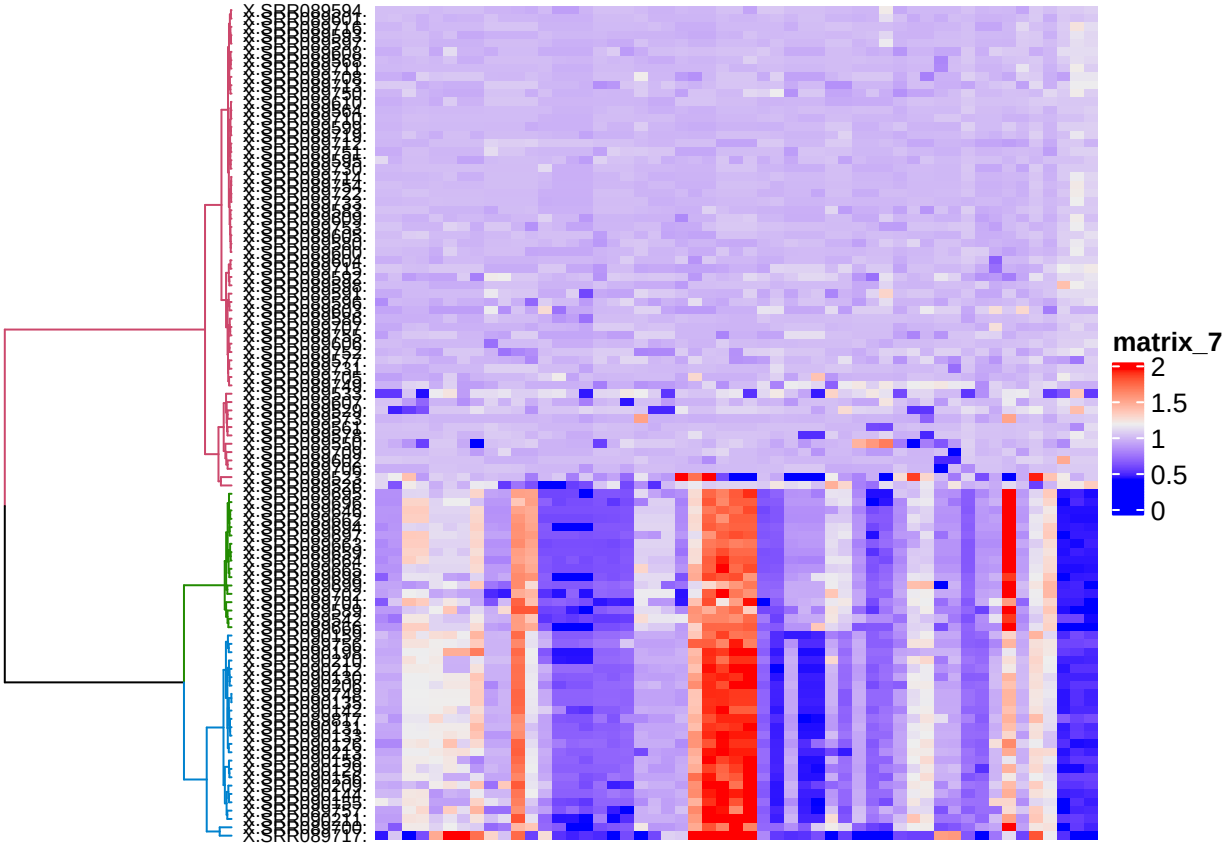
```
## The following object is masked from 'package:stats':
##
##     cutree
```

Musimy zbudować dendrogram dla naszych danych

```
dend <- hclust(dist(my_matrix, method = "maximum"), method = "ward.D")
```

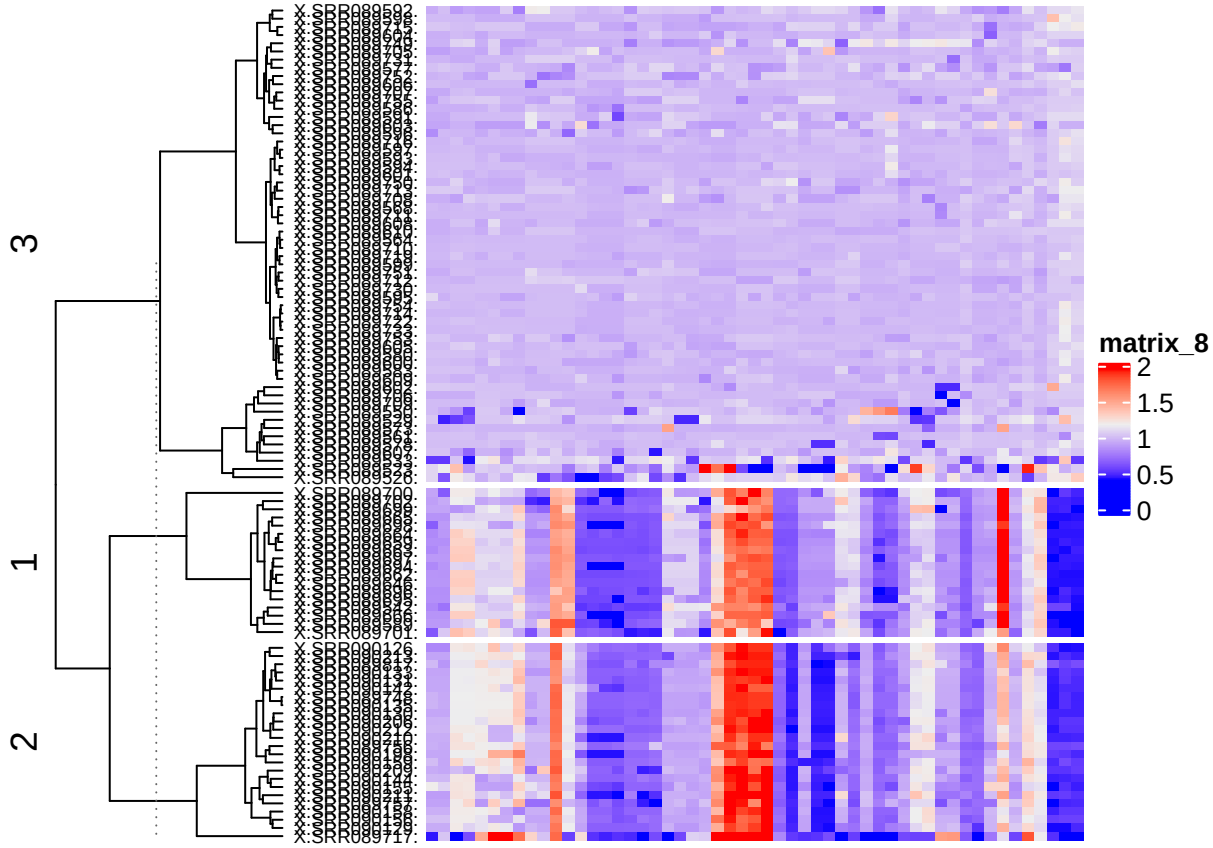
Kolorowanie dendrogramu

```
Heatmap(my_matrix,
        cluster_columns=FALSE,
        row_names_side = "left",
        row_dend_side = "left",
        row_names_gp=gpar(cex=0.6),
        row_dend_width = unit(3, "cm"),
        cluster_rows = color_branches(dend, k = 3))
```



Podział na klastry

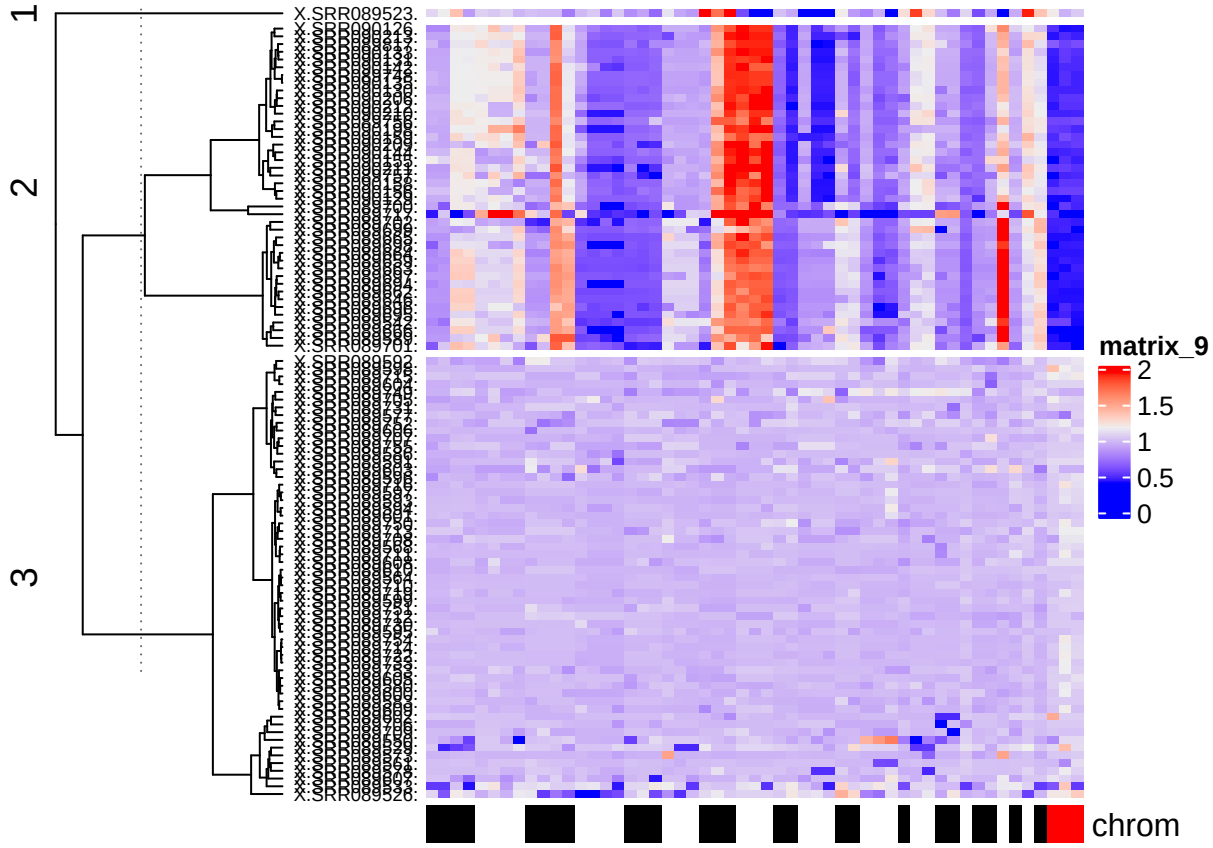
```
Heatmap(my_matrix,
        cluster_columns=FALSE,
        row_names_side = "left",
        row_dend_side = "left",
        row_names_gp=gpar(cex=0.6),
        row_dend_width = unit(3, "cm"),
        clustering_distance_rows ="maximum",
        clustering_method_rows = "ward.D",
        km=3)
```



Zaznaczamy chromosomy

```
chrom_col <- c(rep(c("black", "white"),11),"red")
names(chrom_col) <- paste0("chr",c(seq(1,22),"X"))
```

```
Heatmap(my_matrix,
        cluster_columns=FALSE,
        row_names_side = "left",
        row_dend_side = "left",
        row_names_gp=gpar(cex=0.6),
        row_dend_width = unit(3, "cm"),
        clustering_distance_rows ="maximum",
        clustering_method_rows = "ward.D",
        km=3,
        bottom_annotation = HeatmapAnnotation(df = chrom, col = list(chrom=chrom_col),
                                              show_legend=FALSE))
```


1
2
3
matrix_9
2
1.5
1
0.5
0
chrom