The Wheel

Diego Aguilar-Ramirez

2020-04-08

The following script produces a circular plot showing associations between NMR measured biomarkers and an outcome of interest (in this example, incident diabetes). These associations can be product of any generalised linear model that is of relevance for epidemiology such as linear regression, logistic regression, or Cox regression.

The general structure of the SAS output datasource (usually in .csv) should go as follows:

id_name_s	text1	Estimate	StdErr	WaldChiSq
1	XXL_VLDL_P	0.3709	0.0358	107.6032

Data should be sorted by `id_name_s`, which corresponds to the following biomarkers array as per `text1`:

```
##   [1] XXL_VLDL_P  XL_VLDL_P   L_VLDL_P    M_VLDL_P    S_VLDL_P    XS_VLDL_P
##   [7] IDL_P       L_LDL_P     M_LDL_P     S_LDL_P     XL_HDL_P    L_HDL_P
##  [13] M_HDL_P     S_HDL_P     XXL_VLDL_C  XL_VLDL_C   L_VLDL_C    M_VLDL_C
##  [19] S_VLDL_C    XS_VLDL_C   IDL_C       L_LDL_C     M_LDL_C     S_LDL_C
##  [25] XL_HDL_C    L_HDL_C     M_HDL_C     S_HDL_C     XXL_VLDL_FC XL_VLDL_FC
##  [31] L_VLDL_FC   M_VLDL_FC   S_VLDL_FC   XS_VLDL_FC  IDL_FC      L_LDL_FC
##  [37] M_LDL_FC    S_LDL_FC    XL_HDL_FC   L_HDL_FC    M_HDL_FC    S_HDL_FC
##  [43] XXL_VLDL_CE XL_VLDL_CE  L_VLDL_CE   M_VLDL_CE   S_VLDL_CE   XS_VLDL_CE
##  [49] IDL_CE      L_LDL_CE    M_LDL_CE    S_LDL_CE    XL_HDL_CE   L_HDL_CE
##  [55] M_HDL_CE    S_HDL_CE    XXL_VLDL_TG XL_VLDL_TG  L_VLDL_TG   M_VLDL_TG
##  [61] S_VLDL_TG   XS_VLDL_TG  IDL_TG      L_LDL_TG    M_LDL_TG    S_LDL_TG
##  [67] XL_HDL_TG   L_HDL_TG    M_HDL_TG    S_HDL_TG    XXL_VLDL_PL XL_VLDL_PL
##  [73] L_VLDL_PL   M_VLDL_PL   S_VLDL_PL   XS_VLDL_PL  IDL_PL      L_LDL_PL
##  [79] M_LDL_PL    S_LDL_PL    XL_HDL_PL   L_HDL_PL    M_HDL_PL    S_HDL_PL
##  [85] XXL_VLDL_L  XL_VLDL_L   L_VLDL_L    M_VLDL_L    S_VLDL_L    XS_VLDL_L
##  [91] IDL_L       L_LDL_L     M_LDL_L     S_LDL_L     XL_HDL_L    L_HDL_L
##  [97] M_HDL_L     S_HDL_L     VLDL_D      LDL_D       HDL_D       ApoA1
## [103] ApoB        ApoB_ApoA1  PUFA        MUFA        SFA         DHA
## [109] LA          FAw3        FAw6        TotFA       PUFA_FA     MUFA_FA
## [115] SFA_FA      DHA_FA      LA_FA       FAw3_FA     FAw6_FA     TotCho
## [121] PC          SM          Lac         Cit         Glc         Ala
## [127] Gln         His         Ile         Leu         Val         Phe
## [133] Tyr         Ace         AcAce       bOHBut      Alb         Crea_n
## [139] Gp
## 139 Levels: AcAce Ace Ala Alb ApoA1 ApoB ApoB_ApoA1 bOHBut Cit Crea_n ... XXL_VLDL_TG
```

NOTE: to keep the y-axis in the log-scale (to preserve the estimates in symmetrical and proportional distance from the null-hypothesis), parameters are (perhaps sometimes unsatisfactorily) constantly log transformed

to then be back-transformed by exponentiating such parameters. This not only happens with the values contained in the SAS output datasets but also when defining axes, ticks, labels (as characters), and other situations. This might be confusing and I do apologise for that. Further versions will aim to clean an homogenise such inconsistencies.

Install circlize

You can find the documentation for the package here.
And download from here.

```
library(circlize)
```

1. Prep to call data

Define dataset that will be called into R for plotting.

```
ROOTDIR <- params$ROOTDIR
PREFIX <- "LR"
OUTCOME <- "PRDM"
GROUP <- "ALL"
file.out <- "PRDM"
fact <- "mets"
```

`ROOTDIR` = path to where data with results to plot is located. For this example, ROOTDIR is parametrised, and it should be adjusted to where datasource is located.

`PREFIX` = Prefix from SAS output file name. I use LR = Logistic regression.

`OUTCOME` = Substring from SAS output file name. I use PRDM = prospective or incident diabetes.

`GROUP` = Substring from SAS output file name. I use ALL = as in all individuals included, but could be used to label stratified analyses or other subgroups of interest.

`file.out` = Substring for the output file name.

`fact` = String that can be changed. I randomly chose "mets".

2. Import datasets

Using the objects define above, we now call the datasource with SAS output to create `datasub`.

1. We exponentiate `Estimate` to create `RR`.
2. We then create `RR_1<-RR` if `RR< 1` (*i.e. those with negative associations*), else `RR_1<-1`.
3. We also then create `RR_2<-RR` if `RR>1` (*i.e. those with positive associations*), else `RR_2<-1`.

```
datasub <<- read.csv(paste("", ROOTDIR ,"data\\",PREFIX,"_",OUTCOME,"_",GROUP,".csv", sep=""), skip=0, h

# 1.
datasub$RR <- exp(datasub$Estimate)

# 2.
```

```
datasub$RR_1 <- datasub$RR
datasub$RR_1[datasub$RR>1] <- 1

# 3.
datasub$RR_2 <- datasub$RR
datasub$RR_2[datasub$RR<1] <- 1
```

3. Estimating *p-values*.

1. From SAS output now imported into `datasub`, estimate p-values from chisq statistics `datasub$RawP`.
2. Using the false discovery rate adjustment by Benjamini & Hochberg, `p.adjust` estimates adjusted p-values `datasub$AdjP`.
3. Then, adds flags for the metabolites with evidence against the null hypothesis bellow the fdr-adjusted "significance level".
4. We then create new vectors for estimates that are significant (suffix = `_s`). *NB, suffix `_1` is used for estimates with negative associations and suffix `_2` for estimates with positive associations.* We will add colours later (red for positive and blue for negative, darker shade for those below the significance threshold).
5. If flagged as *"non-significant"* then newly created vectors are transformed into 1 (the value for the null hypothesis).
6. If flagged as *"significant"* then original vectors are transformed into 1 (the value for the null hypothesis).
7. Estimates the number of metabolites based on de dimension of the dataset, necessary later.

```
# 1.
datasub$RawP <- pchisq(datasub$WaldChiSq, 1, lower.tail=FALSE)

# 2.
datasub$AdjP <- p.adjust(datasub$RawP, method = "fdr")

# 3.
datasub$Sig<-NA
datasub$Sig[datasub$AdjP< 0.05] <- 1
datasub$Sig[datasub$AdjP>= 0.05] <- 0

# 4.
datasub$RR_1_s <- datasub$RR_1
datasub$RR_2_s <- datasub$RR_2

# 5.
datasub$RR_1_s[datasub$Sig==0] <- 1
datasub$RR_2_s[datasub$Sig==0] <- 1

# 6.
datasub$RR_1[datasub$Sig==1] <- 1
datasub$RR_2[datasub$Sig==1] <- 1

# 7.
len.data <<-  as.numeric(dim(datasub)[1])
```

4. Plotting parameters

In this section we input the parameters for the plotting areas and steps are taken to keep proportions. Importantly, **the measures to keep proportionality could be substantially improved.**

Y-axis

1. `YLIM YCUTS` and `YCUTS.LABS` define the Y-axis. *Parameters here are defined manually but could be automated by extracting `MIN` and `MAX` and using the pretty function to define cuts and labels.*
2. Alternatively, one could define labels as percentage instead of relative risks, if desired.
3. `ylab` 1:3 define the levels for labels around the circular plot that are relative and proportional to the MAX and MIN of the axis.
4. If estimate is off limits from `YLIM` then estimates are trimmed. **Currently, the plot doesn't flag this transformation,** although it should be evident as the bar ends precisely at the limit of the axis and user should be aware as the axis limits are currently defined manually.

```
# 1.
YLIM <- c(log(0.6), log(1.7))
YCUTS <- c(log(0.6), log(0.75), log(1), log(1.3), log(1.7))
YCUTS.LABS <- as.character(exp(YCUTS))
YMAX <- exp(max(YLIM))
YMIN <- exp(min(YLIM))

# 2.
#YCUTS.LABS <- c("-40%", "-20%", "0%", "30%", "60%")

# 3.
ylab1 <- exp(log(YMAX)+log(YMAX)*0.05)
ylab2 <- exp(log(YMAX)+log(YMAX)*0.3)
ylab3 <- exp(log(YMAX)+log(YMAX)*0.7)
ylab3b <- exp(log(YMAX)+log(YMAX)*0.55)

# 4.
ADJ <- 0.01

datasub$RR_1[datasub$RR_1<=YMIN] <- YMIN+YMIN*ADJ
datasub$RR_2[datasub$RR_2>=YMAX] <- YMAX-YMAX*ADJ

datasub$RR_1_s[datasub$RR_1_s<=YMIN] <- YMIN+YMIN*ADJ
datasub$RR_2_s[datasub$RR_2_s>=YMAX] <- YMAX-YMAX*ADJ

datasub$Estimate[datasub$Estimate<=log(YMIN)] <- log(YMIN)+log(YMIN)*ADJ
datasub$Estimate[datasub$Estimate>=log(YMAX)] <- log(YMAX)-log(YMAX)*ADJ
```

X-axis

IMPORTANT the x-axis is defined by the number of metabolic biomarkers. This number is currently **139** derived from `id_name_s`. All the labels are mapped around this number, and in this especific order. If the user decides a different array of biomarkers is needed (*i.e. only include lipids, or by lipid types instead of by lipoprotein sizes*), then this change can only currently be implemented in SAS and the mapping for labels should also be changed manually.

```
XLIM <- c(min(as.numeric(datasub$id_name_s)), max(as.numeric(datasub$id_name_s)))
```

Labels

1. `labs1` Contains the lipoprotein subclass size acronyms. This is repeated 7 times, once per each measurement of interest (*i.e. lipoprotein particle number, cholesterol, free cholesterol, esterified cholesterol, triglycerides, phospholipids, and total lipids*).
2. `labs4` Vector with additional labels for the rest of biomarkers besides lipids within lipoproteins.
3. `CEX` states a vector to use for sizing. If user changes `CEX` (*with upper case*), then all those functions using CEX will be proportionally re-sized.
   NOTE if the user changes the array defining `id_name_s`, then this section should be changed accordingly.

```
labs1 <- c(rep (c("XXL", "XL", "L", "M", "S", "XS", "IDL", "L", "M", "S", "XL", "L", "M", "S"), 7))

labs4 <- c("VLDL-D", "LDL-D", "HDL-D", "Apo-AI", "Apo-B", "Apo-B/Apo-AI",
           "PUFA", "MUFA", "SFA", "DHA", "LA", "FAw3", "FAw6", "TotFA",
           "PUFA/FA", "MUFA/FA", "SFA/FA", "DHA/FA", "LA/FA", "FAw3/FA", "FAw6/FA",
           "TotCho", "PC", "SM",
           "Lac", "Cit", "Glc",
           "Ala", "Gln", "His", "Ile", "Leu", "Val", "Phe", "Tyr",
           "Ace", "AcAce", "bOHBut",
           "Alb", "Crea", "Glyc-A")

CEX <- (8)
```

Graphic device

We decided to use the png graphic device, but others such as pdf or tiff do the trick as well.
1. The line we would use produces a filename that includes the `file.out` substring defined above as well as `GROUP` and the date.
2. For this document, have named the output file `"foo.png"`.

```
# 1.
#png(paste("",ROOTDIR,"tables and figures\\" , file.out ," " , GROUP ," ", format(Sys.time(), " %Y-%m-%
# 2.
name <- "foo.png"
png(filename = name, height=6000,width=6000, bg = "white")
```

Margins

The outer margins `OMA` are quite large (*29 spaces, in the 4 margins*), as we need space to place our labels.

```
par(xpd = NA, oma = rep(29,4))
```

5. Circos function

This represents the core of the script, although most of the job is done above. It uses the *circlize* package but, as you will see, most of the basic R plot functions are preserved and only slightly changed.

Importantly, this plot uses only very limitedly the applications of the *circlize* package. Some of the approaches I have had to make the plot are probably clumsy or redundant.

I **highly** recommend to have a quick look into the documentation. It is simpler than it looks and relatively easy to work with.

Parameters

Circlize transforms a "Cartesian plane" with x and y axis into a *circle* of y radius and x circumference. Basically, a traditional rectangular plot is twisted into a donut. The *donut* is called *sector*. Sectors can be split in several tracks. You can add additional sectors or *donuts* in ever more central levels.

Our example only has 1 sector with 1 track.
`track.height` determines the proportion of the radius of the circle the track (where we are going to plot) is going to use. The circle used by circlize always has a radius of 1, so a height of 0.1 means 10% of the circle radius.
`gap.degree` determines the space between the end of the track and the start of the track.
`start.degree` determines the place to start the track at in degrees (count starts at the *West*).

```
circos.par("track.height" = 0.6,
           cell.padding = c(0, 0, 0, 0),
           gap.degree = 45,
           start.degree = 90,
           unit.circle.segments=50000)
```

Initialize the circle

1. `circos.initialize` is the core function that determines the basic parameters. I am still not entirely sure how it works. However, a character object in the `factors` option (in this example `fact`) does the trick and this becomes the name of our *sector*.

2. `xlim` is defined by the length of the `id_name_s` column, as noted above.

```
# 1.
circos.initialize(factors = fact, xlim = c(0,len.data))

# 2.
circos.track(factors = fact, ylim = YLIM, bg.border = NA)
```

Draw shades for metabolic subgroups

To highlight specific regions use *circlize()* to calculate the positions in the polar coordinate. Always keep in mind that *x-axis* in the cell are always clock wise.

The highlight region to be calculated by `circlize()`needs coordinates in x and y, a `sector.index` (in this case `"mets"`), and a `track.index` (in this case `1`).

NOTE: In this example, the coordinates were imputed manually and correspond to the array defined by `id_names_s`. If changed, this section must also be changed to preserve meaningful highlight regions.

Unless the user wants to change the order of the biomarkers, this section needs no further details explained.

```
pos1 = circlize(c(0.5, 6.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos1[1, "theta"], pos1[2, "theta"], pos1[1, "rou"], pos1[2, "rou"], clock.wise = TRUE, col =

pos2 = circlize(c(10.5, 14.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos2[1, "theta"], pos2[2, "theta"], pos2[1, "rou"], pos2[2, "rou"], clock.wise = TRUE, col =

pos3 = circlize(c(20.5, 24.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos3[1, "theta"], pos3[2, "theta"], pos3[1, "rou"], pos3[2, "rou"], clock.wise = TRUE, col =

pos4 = circlize(c(28.5, 34.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos4[1, "theta"], pos4[2, "theta"], pos4[1, "rou"], pos4[2, "rou"], clock.wise = TRUE, col =

pos5 = circlize(c(38.5, 42.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos5[1, "theta"], pos5[2, "theta"], pos5[1, "rou"], pos5[2, "rou"], clock.wise = TRUE, col =

pos6 = circlize(c(48.5, 52.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos6[1, "theta"], pos6[2, "theta"], pos6[1, "rou"], pos6[2, "rou"], clock.wise = TRUE, col =

pos7 = circlize(c(56.5, 62.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos7[1, "theta"], pos7[2, "theta"], pos7[1, "rou"], pos7[2, "rou"], clock.wise = TRUE, col =

pos8 = circlize(c(66.5, 70.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos8[1, "theta"], pos8[2, "theta"], pos8[1, "rou"], pos8[2, "rou"], clock.wise = TRUE, col =

pos9 = circlize(c(76.5, 80.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos9[1, "theta"], pos9[2, "theta"], pos9[1, "rou"], pos9[2, "rou"], clock.wise = TRUE, col =

pos10 = circlize(c(84.5, 90.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos10[1, "theta"], pos10[2, "theta"], pos10[1, "rou"], pos10[2, "rou"], clock.wise = TRUE,

pos11 = circlize(c(94.5, 98.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos11[1, "theta"], pos11[2, "theta"], pos11[1, "rou"], pos11[2, "rou"], clock.wise = TRUE,

pos12 = circlize(c(104.5, 112.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos12[1, "theta"], pos12[2, "theta"], pos12[1, "rou"], pos12[2, "rou"], clock.wise = TRUE,

pos13 = circlize(c(119.5, 122.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos13[1, "theta"], pos13[2, "theta"], pos13[1, "rou"], pos13[2, "rou"], clock.wise = TRUE,

pos14 = circlize(c(125.5, 133.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos14[1, "theta"], pos14[2, "theta"], pos14[1, "rou"], pos14[2, "rou"], clock.wise = TRUE,

pos15 = circlize(c(136.5, 139.5), c(min(YLIM), max(YLIM)), sector.index = "mets", track.index = 1)
draw.sector(pos15[1, "theta"], pos15[2, "theta"], pos15[1, "rou"], pos15[2, "rou"], clock.wise = TRUE,
```

Plotting region

1. Using `circos.track`, we select track `1`, using factors defined in object `fact`, and the `YLIM` defined above.

2. We use `circos.segmets` exactly as `segments` would be used to create:

   i) Start and end of plot lines.

ii) Outer and inner lines.

iii) Lines at null hypotesis and other cuts.

```
# 1.
circos.track(track.index = 1, bg.border = "white", factors = fact, ylim = YLIM, panel.fun = function(x,

# i)
# Start
circos.segments(x0=min(XLIM)-0.5, y0=max(YLIM), x1=min(XLIM)-0.5, y1=min(YLIM), col = "black", lwd=2)
# End
circos.segments(x0=max(XLIM)+0.5, y0=max(YLIM), x1=max(XLIM)+0.5, y1=min(YLIM), col = "black", lwd=2)

# ii)
# Outer
circos.segments(x0=min(XLIM)-.75, y0=max(YLIM), x1=max(XLIM)+0.5, y1=max(YLIM), col = "black", lwd=2)
# Inner
circos.segments(x0=min(XLIM)-.75, y0=min(YLIM), x1=max(XLIM)+0.5, y1=min(YLIM), col = "black", lwd=2)

# iii)
# Lines at YCUTS
# Null Hypothesis
circos.segments(x0=min(XLIM)-.75, y0=0, x1=max(XLIM)+0.5, y1=0, col = "black", lwd=2)
circos.segments(x0=min(XLIM)-.75, y0=(YCUTS[2]), x1=max(XLIM)+0.5, y1=(YCUTS[2]), col = "gray75", lwd=2
circos.segments(x0=min(XLIM)-.75, y0=(YCUTS[4]), x1=max(XLIM)+0.5, y1=(YCUTS[4]), col = "gray75", lwd=2
```

Draw bars with estimates

`circos.rect` draws a rectangle of `xleft`, `xright`, `ytop`, and `ybottom` dimentions.

Each bar is defined in the x axis by its position withing `is_name_s`. Width is defined by simply substracting or adding 0.35 to the coordinates in `xleft` and `xright`, respectively.

Each bar of the 4 types of bars are defined in the y axis by the value in one of the four `RR` vectors created above, based on the following:

1. Positive and "significant", in dark red (i.e. `RR_2_s`).
2. Positive and not "significant", in light red (i.e. `RR_2`).
3. Negative and "significant", in dark blue (i.e. `RR_1_s`).
4. Negative and "non-significant", in light blue (i.e. `RR_1`).

Colours are defined in hex with the last 2 digits defining transparency.

```
# Bars
# 1.
circos.rect(xleft=(as.numeric(datasub$id_name_s)-.35), xright=(as.numeric(datasub$id_name_s)+.35), ytop=
# 2.
circos.rect(xleft=(as.numeric(datasub$id_name_s)-.35), xright=(as.numeric(datasub$id_name_s)+.35), ytop=
# 3.
```

```
circos.rect(xleft=(as.numeric(datasub$id_name_s)-.35), xright=(as.numeric(datasub$id_name_s)+.35),ytop=
# 4.
circos.rect(xleft=(as.numeric(datasub$id_name_s)-.35), xright=(as.numeric(datasub$id_name_s)+.35), ytop=
```

Draw confidence intervals

Also using `circos.rect` draw confidence intervals out of `StdErr`.

NOTE: This chunk must be plotted after the bars, so the graphic device can draw the confidence intervals on top.

```
# Confidence intervals
circos.rect(xleft=(as.numeric(datasub$id_name_s)), xright=(as.numeric(datasub$id_name_s)), ytop=(as.num
```

Another useful option is `circos.points`, which allows to draw blobs with the basic R `pch` options.

Draw y-axis

Similar to basic R plotting axis options.

```
circos.yaxis(at=YCUTS, labels = YCUTS.LABS, labels.cex = CEX*1, tick = FALSE, col = "white")
```

Draw labels

We use the function `circos.text` to paste the labels at the margins of the plot (remember we left a lot of space at the margins when defining the plot `par` above).

The option `facing` defines how to paste the labels. The package has several options that make text look nicely, including `niceFacing`, which makes text flip so it can be read easily.

The positions for labels are, unfortunately, very inefficiently defined manually.

All the `.shift` objects were used to manually adjust the labels. These work now, but maybe play with them to see how the labels move.

```
VLDL.shift <- 2
LDL.shift <- 1.5
HDL.shift <- 1.5

circos.text(x=0.5+5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Lipoprotein
circos.text(x=0.5+VLDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "VLD
circos.text(x=6.5+LDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "LDL"
circos.text(x=10.5+HDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "HDL

circos.text(x=14.5+5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Cholesterol
circos.text(x=14.5+VLDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "VL
circos.text(x=20.5+LDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "LDL
circos.text(x=24.5+HDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "HDL

circos.text(x=28.5+5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Free choles
circos.text(x=28.5+VLDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "VL
circos.text(x=34.5+LDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "LDL
```

```
circos.text(x=38.5+HDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = FALSE, labels = "HDL

circos.text(x=42.5+5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Esterified C
circos.text(x=42.5+VLDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "VLD
circos.text(x=48.5+LDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "LDL
circos.text(x=52.5+HDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "HDL

circos.text(x=56.5+5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Triglyceride
circos.text(x=56.5+VLDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "VLD
circos.text(x=62.5+LDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "LDL
circos.text(x=66.5+HDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "HDL

circos.text(x=70.5+5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Phospholipid
circos.text(x=70.5+VLDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "VLD
circos.text(x=76.5+LDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "LDL
circos.text(x=80.5+HDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "HDL

circos.text(x=84.5+5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Total lipids
circos.text(x=84.5+VLDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "VLD
circos.text(x=90.5+LDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "LDL
circos.text(x=94.5+HDL.shift, y=log(ylab2), facing = "bending.inside", niceFacing = TRUE, labels = "HDL

circos.text(x=98.5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Sizes & Apo-LI

circos.text(x=104.5+6, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Fatty acids

circos.text(x=119.5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "Cholines, gly

circos.text(x=133.5, y=log(ylab3), facing = "bending.inside", niceFacing = TRUE, labels = "  Ketone bodi
circos.text(x=133.5, y=log(ylab3b), facing = "bending.inside", niceFacing = TRUE, labels = "& fluid bala

circos.text(x=c(1:98)+0.25,   y = log(ylab1), labels = labs1, facing = "clockwise", niceFacing = TRUE, c
circos.text(x=c(99:139)+0.25, y = log(ylab1), labels = labs4, facing = "clockwise", niceFacing = TRUE, c


            }

)
```

Title

Paste the title at the centre of the circle (at `x=0, y=0`).

Use `circos.clear` to reset the circular layout parameters.

Close the plotting device with `dev.off()`.

```
text(0, 0, paste("Increase or decrease in\nodds of incident diabetes\nassociated with 1SD\nhigher levels

circos.clear()

dev.off()
```

6. Output

```
## pdf
##   2
```