**Scales**

On a vu qu’avec ggplot2 on définissait des mappages entre des attributs graphiques (position, taille, couleur, etc.) et des variables d’un tableau de données. Ces mappages sont définis, pour chaque geom, via la fonction aes().

Les *scales* dans ggplot2 permettent de modifier la manière dont un attribut graphique va être relié aux valeurs d’une variable, et dont la légende correspondante va être affichée. Par exemple, pour l’attribut color, on pourra définir la palette de couleur utilisée. Pour size, les tailles minimales et maximales, etc.

Pour modifier une *scale* existante, on ajoute un nouvel élément à notre objet ggplot2 avec l’opérateur +. Cet élément prend la forme scale\_<attribut>\_<type>.

Voyons tout de suite quelques exemples.

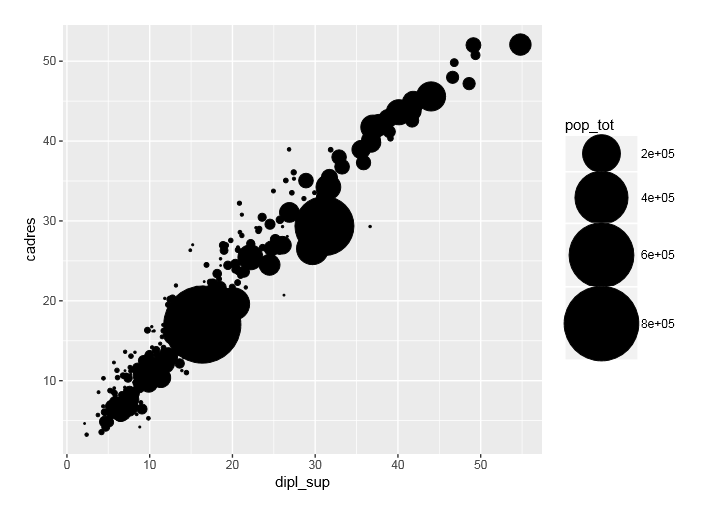
### scale\_size

Si on souhaite modifier les tailles minimales et maximales des objets quand on a effectué un mappage de type size, on peut utiliser la fonction scale\_size et son argument range :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, size = pop\_tot)) +

scale\_size(range = c(0,20))

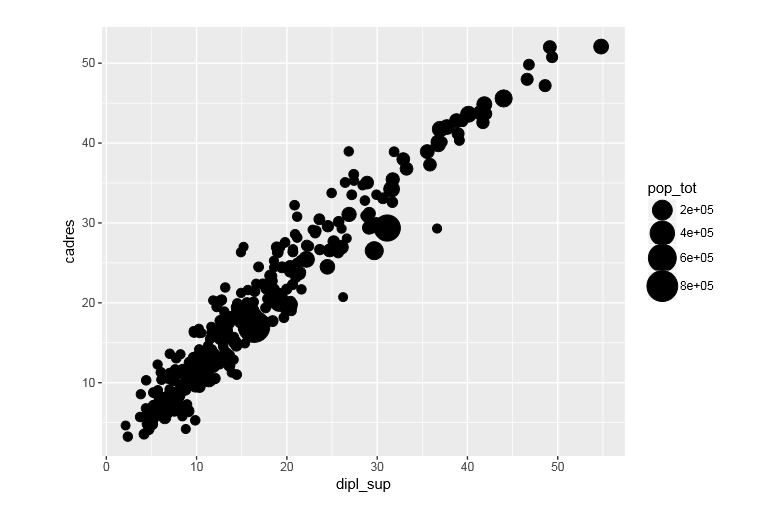


À comparer par exemple à :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, size = pop\_tot)) +

scale\_size(range = c(2,8))

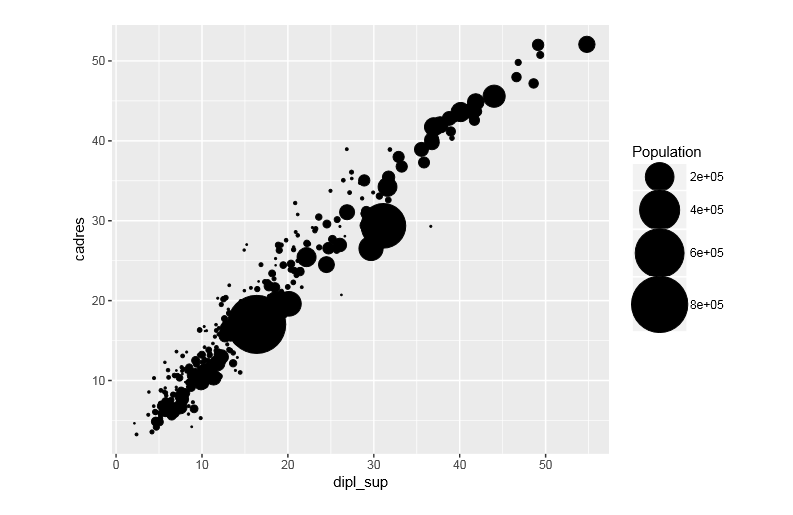


On peut ajouter d’autres paramètres à scale\_size. Le premier argument est toujours le titre donné à la légende :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, size = pop\_tot)) +

scale\_size("Population", range = c(0,15))

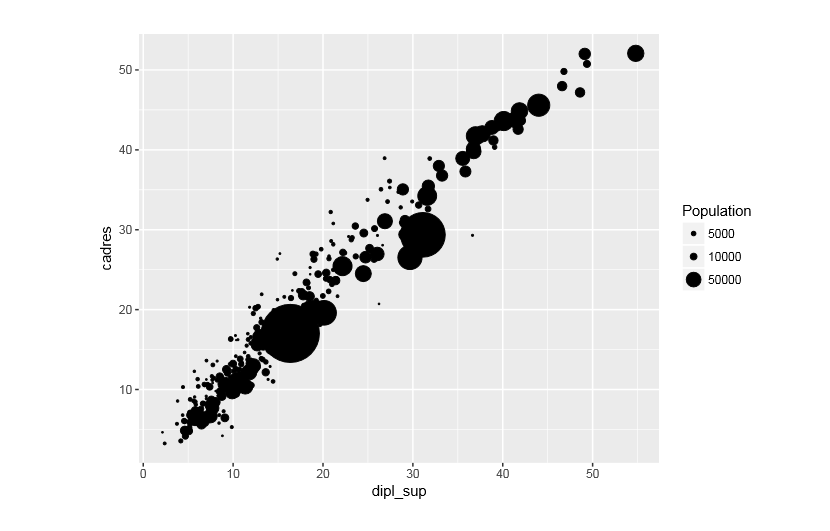


On peut aussi définir manuellement les éléments de légende représentés :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, size = pop\_tot)) +

scale\_size("Population", range = c(0,15), breaks = c(1000,5000,10000,50000))



### scale\_x, scale\_y

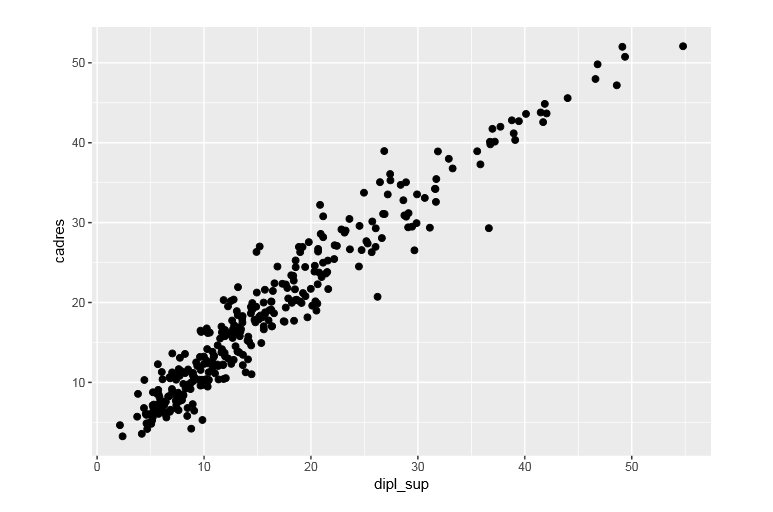
Les scales scale\_x et scale\_y modifient les axes x et y du graphique.

scale\_x\_continuous et scale\_y\_continuous s’appliquent lorsque la variable x ou y est numérique (quantitative).

C’est le cas de notre nuage de points croisant part de cadres et part de diplômés du supérieur :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres))



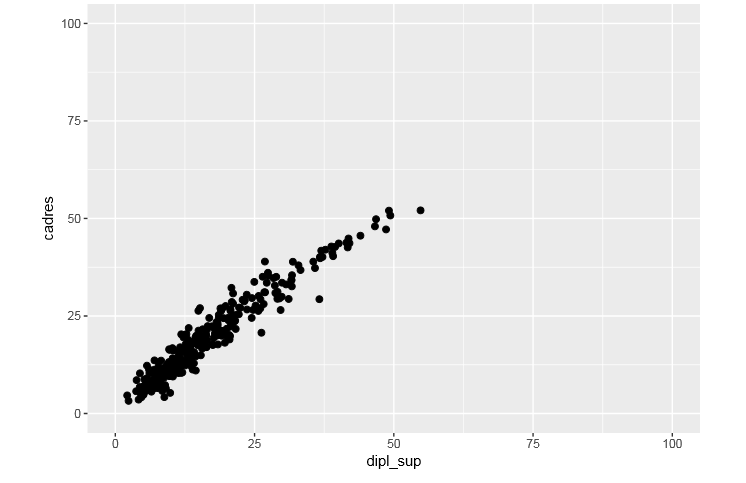
Comme on représente des pourcentages, on peut vouloir forcer les axes x et y à s’étendre des valeurs 0 à 100. On peut le faire en ajoutant un élément scale\_x\_continuous et un élément scale\_y\_continuous, et en utilisant leur argument limits :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres)) +

scale\_x\_continuous(limits = c(0,100)) +

scale\_y\_continuous(limits = c(0,100))

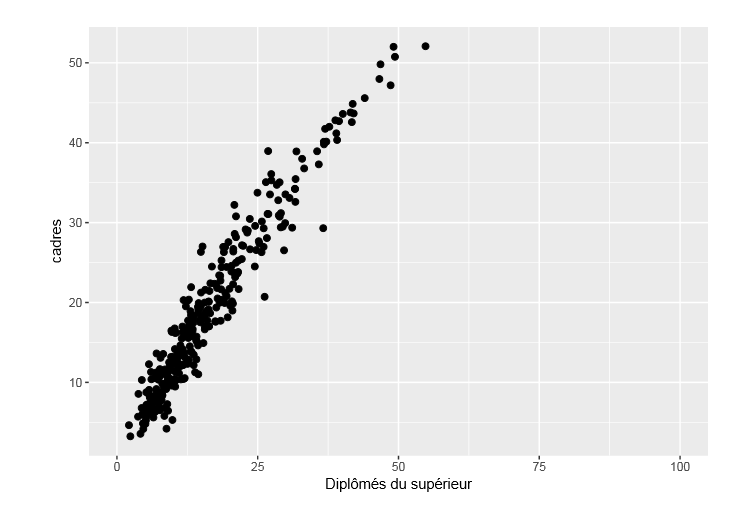


Là aussi, on peut modifier les étiquettes des axes en indiquant une chaîne de caractères en premier argument :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres)) +

scale\_x\_continuous("Diplômés du supérieur", limits = c(0,100))

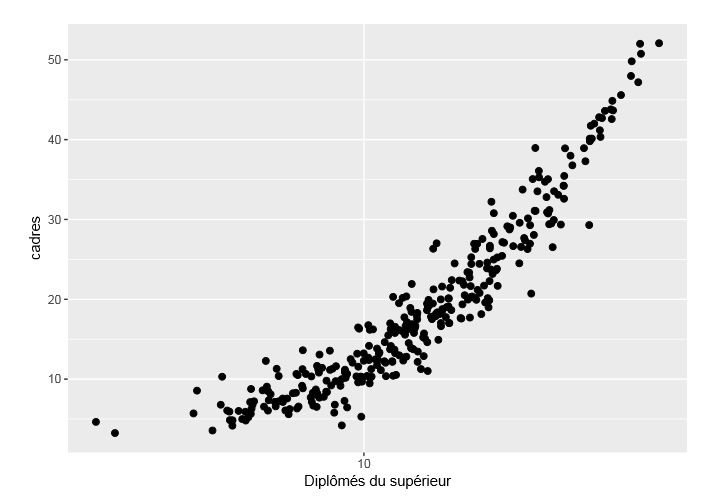


On peut utiliser scale\_x\_log10 et scale\_y\_log10 pour passer un axe à une échelle logarithmique :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres)) +

scale\_x\_log10("Diplômés du supérieur")

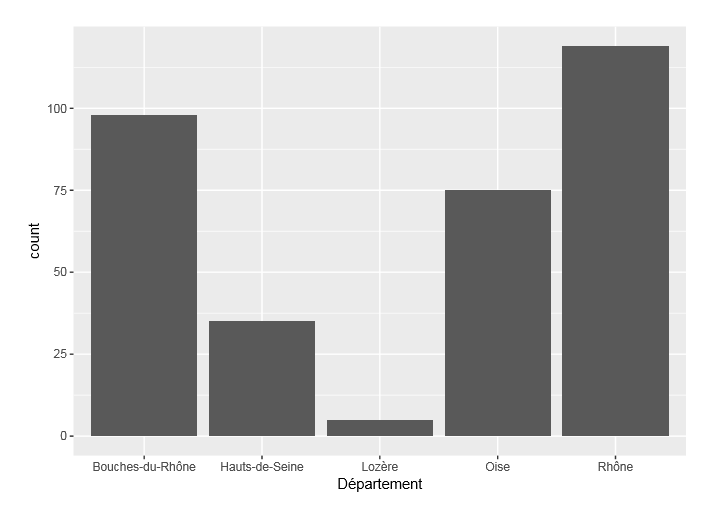


scale\_x\_discrete et scale\_y\_discrete s’appliquent quant à elles lorsque l’axe correspond à une variable discrète (qualitative). C’est le cas de l’axe des x dans un diagramme en bâtons :

ggplot(rp) +

geom\_bar(aes(x = departement)) +

scale\_x\_discrete("Département")



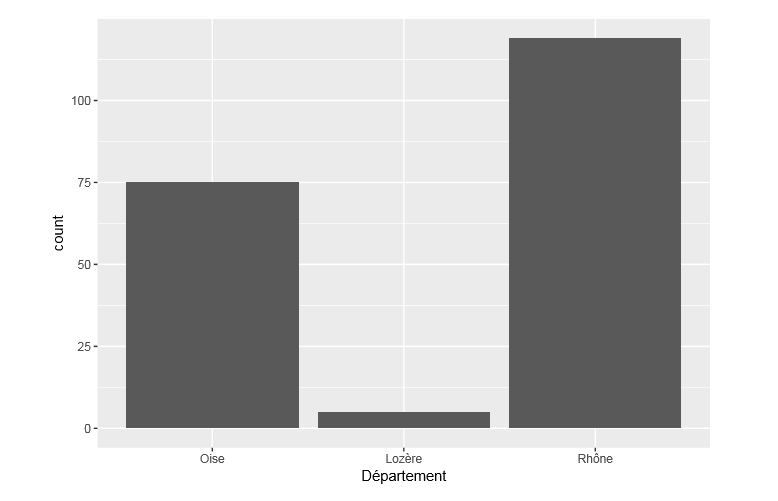
L’argument limits de scale\_x\_discrete permet d’indiquer quelles valeurs sont affichées et dans quel ordre.

ggplot(rp) +

geom\_bar(aes(x = departement)) +

scale\_x\_discrete("Département", limits = c("Oise", "Lozère", "Rhône"))

Warning: Removed 133 rows containing non-finite values (stat\_count).



### scale\_color, scale\_fill

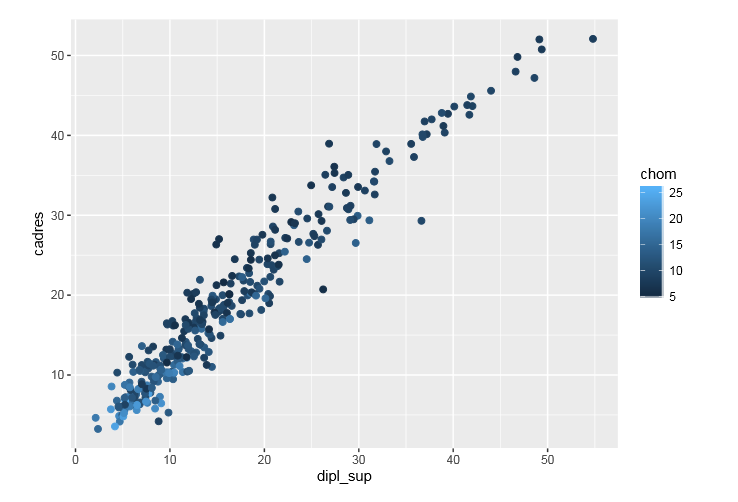
Ces scales permettent, entre autre, de modifier les palettes de couleur utilisées pour le dessin (color) ou le remplissage (fill) des éléments graphiques. Dans ce qui suit, pour chaque fonction scale\_color présentée il existe une fonction scale\_fill équivalente et avec en général les mêmes arguments.

#### 8.7.3.1 Variables quantitatives

Le graphique suivant colore les points selon la valeur d’une variable numérique quantitative (ici la part de chômeurs) :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, color = chom))

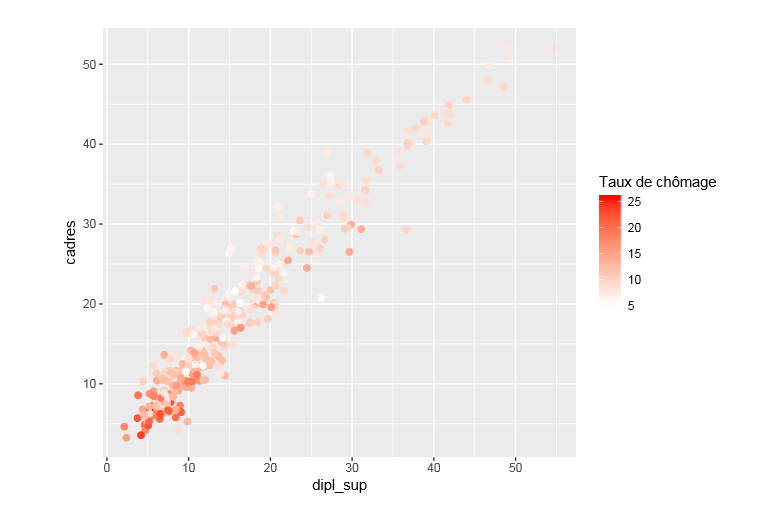


On peut modifier les couleurs utilisées avec les arguments low et high de la fonction scale\_color\_gradient. Ici on souhaite que la valeur la plus faible soit blanche, et la plus élevée rouge :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, color = chom)) +

scale\_color\_gradient("Taux de chômage", low = "white", high = "red")



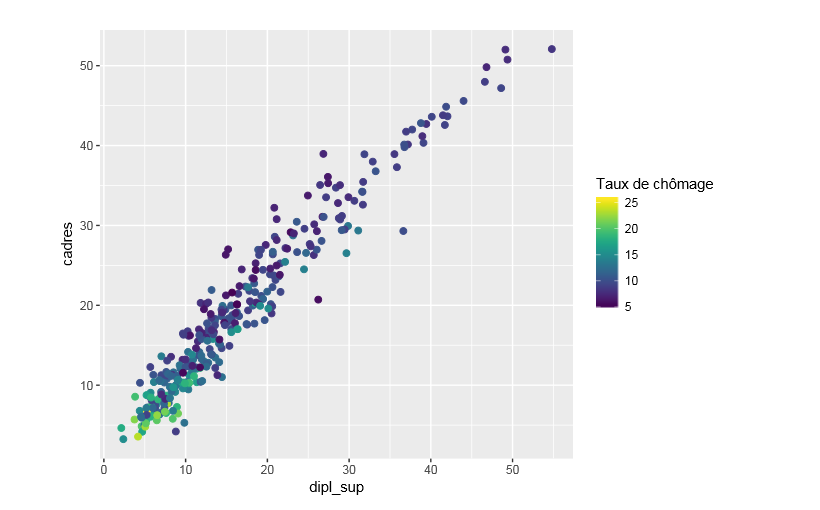
On peut aussi utiliser des palettes prédéfinies. L’une des plus populaires est la palette viridis, accessible depuis l’extension du même nom. On l’ajoute en utilisant scale\_color\_viridis :

library(viridis)

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, color = chom)) +

scale\_color\_viridis("Taux de chômage")

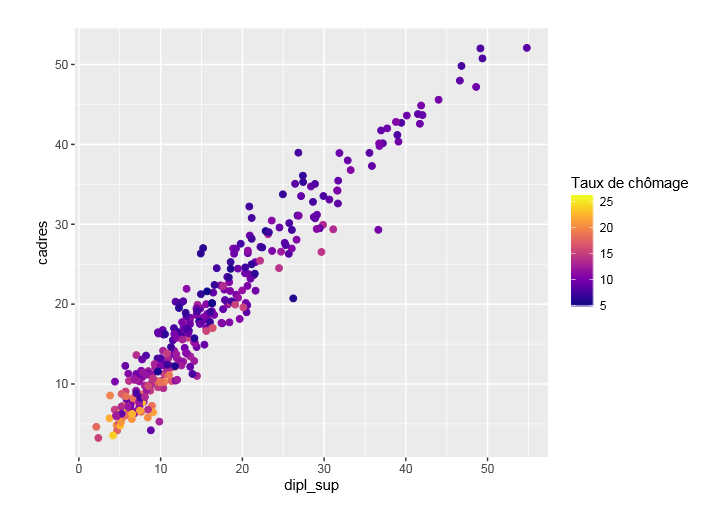


L’extension viridis propose également trois autres palettes, magma, inferno et plasma, accessibles via l’argument option :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, color = chom)) +

scale\_color\_viridis("Taux de chômage", option = "plasma")

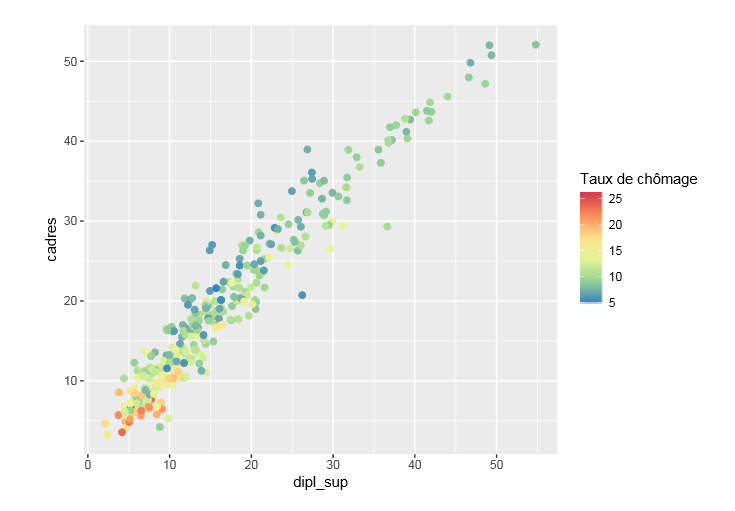


On peut aussi utiliser scale\_color\_distiller, qui transforme une des palettes pour variable qualitative de scale\_color\_brewer en palette continue pour variable numérique :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, color = chom)) +

scale\_color\_distiller("Taux de chômage", palette = "Spectral")



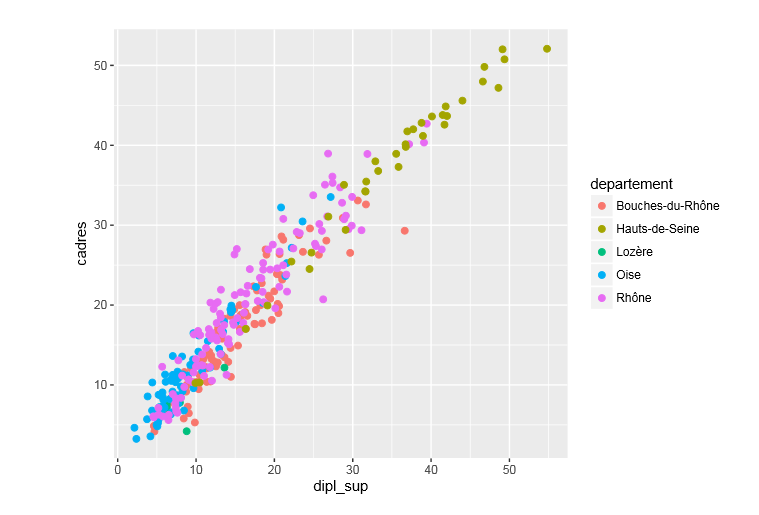
La liste des palettes de scale\_color\_brewer est indiquée en fin de section suivante.

#### 8.7.3.2 Variables qualitatives

Si on a fait un mappage avec une variable discrète (qualitative), comme ici avec le département :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, color = departement))



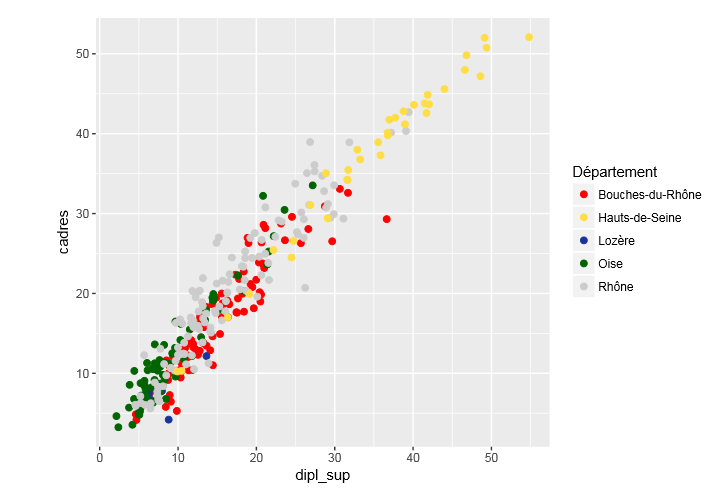
Une première possibilité est de modifier la palette manuellement avec scale\_color\_manual et son argument values :

ggplot(rp) +

geom\_point(aes(x = dipl\_sup, y = cadres, color = departement)) +

scale\_color\_manual("Département",

values = c("red", "#FFDD45", rgb(0.1,0.2,0.6), "darkgreen", "grey80”))



L’exemple précédent montre plusieurs manières de définir manuellement des couleurs dans R :

* Par code hexadécimal : “#FFDD45”
* En utilisant la fonction rgb et en spécifiant les composantes rouge, vert, bleu par des nombres entre 0 et 1 (et optionnellement une quatrième composante d’opacité, toujours entre 0 et 1) : rgb(0.1,0.2,0.6)
* En donnant un nom de couleur : “red”, “darkgreen”

La liste complète des noms de couleurs connus par R peut être obtenu avec la fonction colors(). Vous pouvez aussi retrouver en ligne [la liste des couleurs et leur nom](http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf) (PDF).

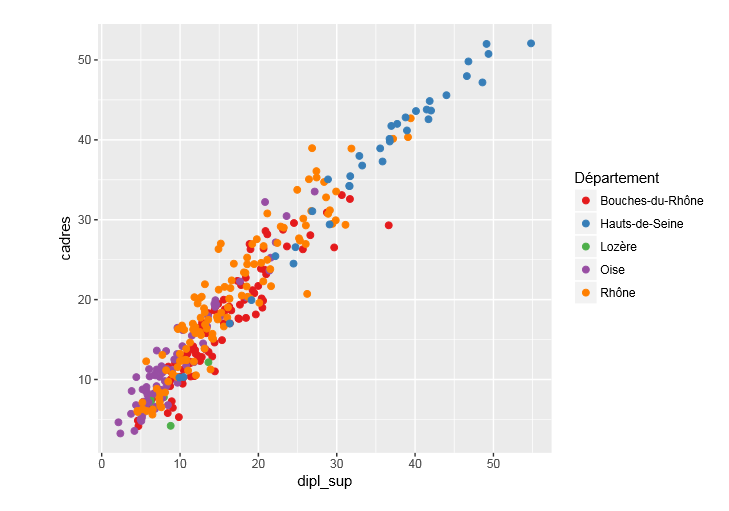
Il est cependant souvent plus pertinent d’utiliser des palettes prédéfinies. Celles du site [Colorbrewer](http://colorbrewer2.org/), initialement prévues pour la cartographie, permettent une bonne lisibilité, et peuvent être adaptées pour certains types de daltonisme.

Ces palettes s’utilisent via la fonction scale\_color\_brewer, en passant le nom de la palette via l’argument palette. Par exemple, si on veut utiliser la palette Set1 :

ggplot(rp) +

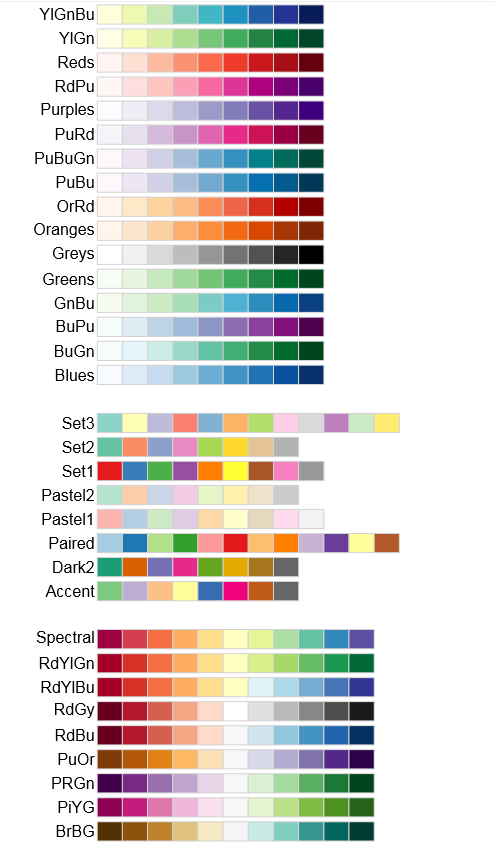
geom\_point(aes(x = dipl\_sup, y = cadres, color = departement)) +

scale\_color\_brewer("Département", palette = "Set1")



Le graphique suivant, accessible via la fonction display.brewer.all(), montre la liste de toutes les palettes disponibles via scale\_color\_brewer. Elles sont réparties en trois familles : les palettes séquentielles (pour une variable quantitative), les palettes qualitatives, et les palettes divergentes (typiquement pour une variable quantitative avec une valeur de référence, souvent 0, et deux palettes continues distinctes pour les valeurs inférieures et pour les valeurs supérieures).

RColorBrewer::display.brewer.all()



Il existe d’autres méthodes pour définir les couleurs : pour plus d’informations on pourra se reporter à [l’article de la documentation officielle sur ce sujet](http://ggplot2.tidyverse.org/articles/ggplot2-specs.html#colour).

## 8.8 Thèmes

Les thèmes permettent de contrôler l’affichage de tous les éléments du graphique qui ne sont pas reliés aux données : titres, grilles, fonds, etc.

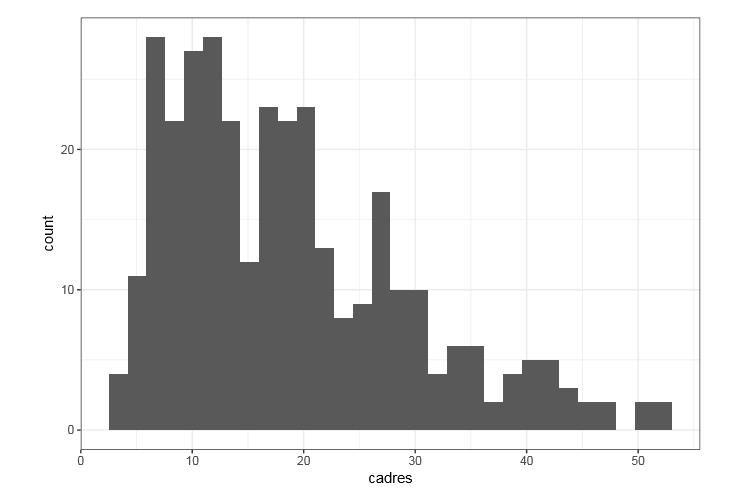
Il existe un certain nombre de thèmes préexistants, par exemple le thème theme\_bw :

ggplot(data = rp) +

geom\_histogram(aes(x = cadres)) +

theme\_bw()

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



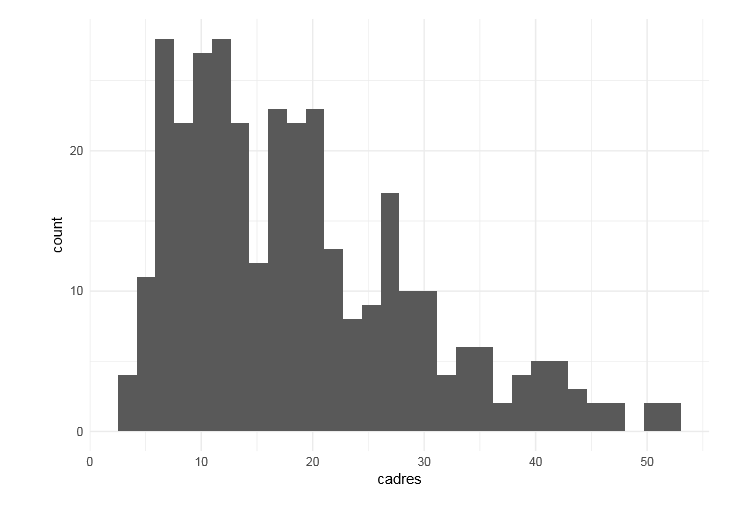
Ou le thème theme\_minimal :

ggplot(data = rp) +

geom\_histogram(aes(x = cadres)) +

theme\_minimal()

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



On peut cependant modifier manuellement les différents éléments. Par exemple, les fonctions ggtitle, xlab et ylab permettent d’ajouter ou de modifier le titre du graphique, ainsi que les étiquettes des axes x et y :

ggplot(data = rp) +

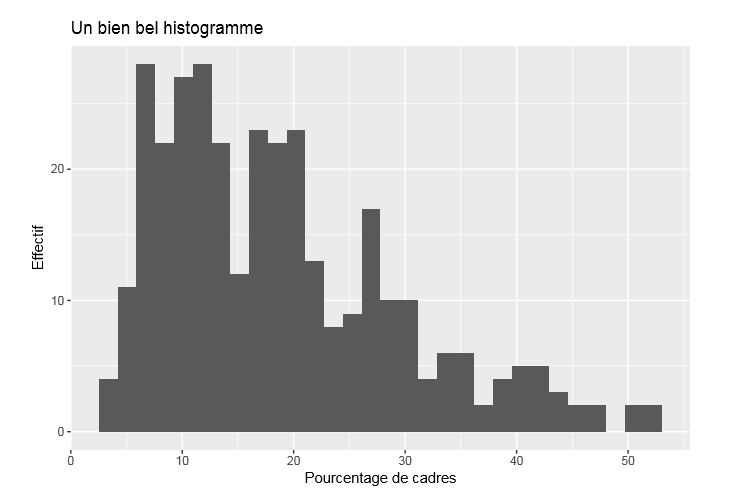
geom\_histogram(aes(x = cadres)) +

ggtitle("Un bien bel histogramme") +

xlab("Pourcentage de cadres") +

ylab("Effectif")

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



Les éléments personnalisables étant nombreux, un bon moyen de se familiariser avec tous les arguments est sans doute l’addin RStudio ggThemeAssist. Pour l’utiliser il suffit d’installer le package du même nom, de sélectionner dans son script RStudio le code correspondant à un graphique ggplot2, puis d’aller dans le menu Addins et choisir ggplot Theme Assistant. Une interface graphique s’affiche alors permettant de modifier les différents éléments. Si on clique sur Done, le code sélectionné dans le script est alors automatiquement mis à jour pour correspondre aux modifications effectuées.

Ce qui permet d’obtenir très facilement des résultats extrêmement moches :

ggplot(data = rp) + geom\_histogram(aes(x = cadres)) +

theme(panel.grid.major = element\_line(colour = "dodgerblue",

size = 0.5, linetype = "dotdash"), axis.title = element\_text(family = "serif",

size = 18, face = "italic", colour = "white"),

axis.text = element\_text(family = "serif",

size = 15, face = "bold"), axis.text.x = element\_text(family = "mono"),

plot.title = element\_text(family = "serif"),

legend.text = element\_text(family = "serif"),

legend.title = element\_text(family = "serif"),

panel.background = element\_rect(fill = "coral"),

plot.background = element\_rect(fill = "blueviolet"))

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



[La documentation officielle](http://ggplot2.tidyverse.org/index.html) (en anglais) de ggplot2 est très complète et accessible en ligne.

Une “antisèche” (en anglais) résumant en deux pages l’ensemble des fonctions et arguments et disponible soit directement depuis RStudio (menu Help > Chetasheets > Data visualization with ggplot2) ou [directement en ligne](https://www.rstudio.com/resources/cheatsheets/)

Les parties [Data visualisation](http://r4ds.had.co.nz/data-visualisation.html) et [Graphics for communication](http://r4ds.had.co.nz/graphics-for-communication.html) de l’ouvrage en ligne R for data science, de Hadley Wickham, sont une très bonne introduction à ggplot2.

Plusieurs ouvrages, toujours en anglais, abordent en détail l’utilisation de ggplot2, en particulier [ggplot2: Elegant Graphics for Data Analysis](http://www.amazon.fr/ggplot2-Elegant-Graphics-Data-Analysis/dp/0387981403/), toujours de Hadley Wickham, et le [R Graphics Cookbook](http://www.amazon.fr/R-Graphics-Cookbook-Winston-Chang/dp/1449316956) de Winston Chang.

Le [site associé](http://www.cookbook-r.com/Graphs/) à ce dernier ouvrage comporte aussi pas mal d’exemples et d’informations intéressantes.

Enfin, si ggplot2 présente déjà un très grand nombre de fonctionnalités, il existe aussi un système d’extensions permettant d’ajouter des geom, des thèmes, etc. Le site [ggplot2 extensions](http://www.ggplot2-exts.org/) est une très bonne ressource pour les parcourir et les découvrir, notamment grâce à sa [galerie](http://www.ggplot2-exts.org/gallery/).

## Représentation de plusieurs geom

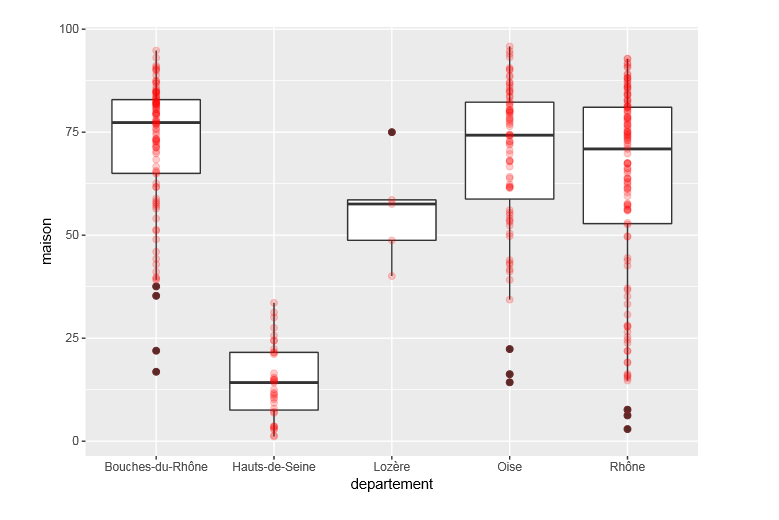
On peut représenter plusieurs geom simultanément sur un même graphique, il suffit de les ajouter à tour de rôle avec l’opérateur +.

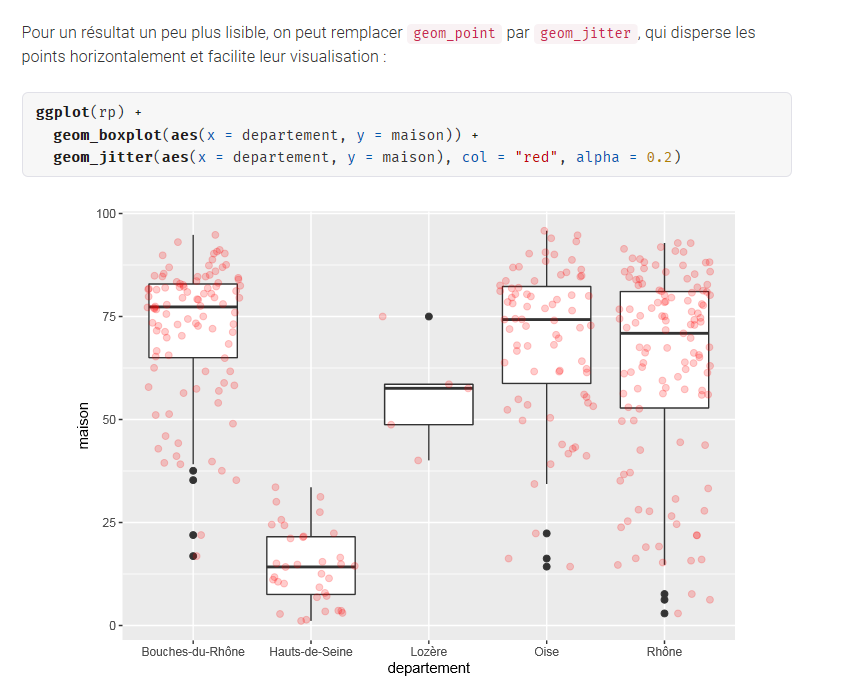
Par exemple, on peut superposer la position des points au-dessus d’un graphique en boîtes à moustaches. On va pour cela ajouter un geom\_point après avoir ajouté notre geom\_boxplot :

ggplot(rp) +

geom\_boxplot(aes(x = departement, y = maison)) +

geom\_point(aes(x = departement, y = maison), col = "red", alpha = 0.2)



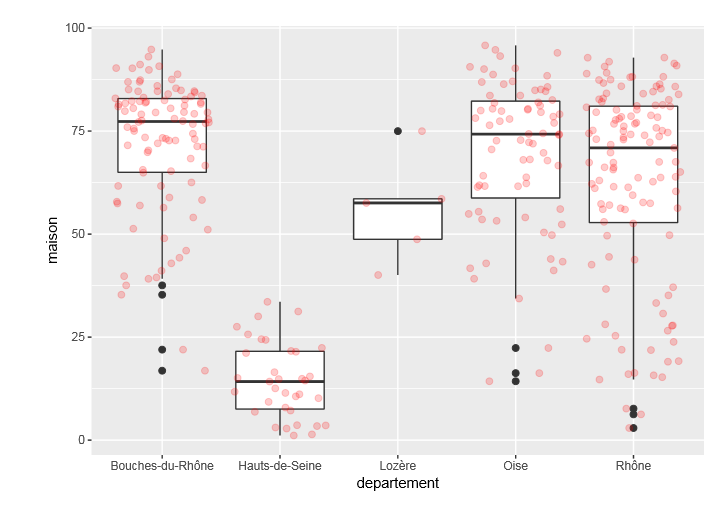


Dans ce cas, plutôt que de déclarer les mappages dans chaque geom, on peut les déclarer dans l’appel à ggplot(). Ils seront automatiquement “hérités” par les geom ajoutés (sauf s’ils redéfinissent les mêmes mappages) :

ggplot(rp, aes(x = departement, y = maison)) +

geom\_boxplot() +

geom\_jitter(color = "red", alpha = 0.2)

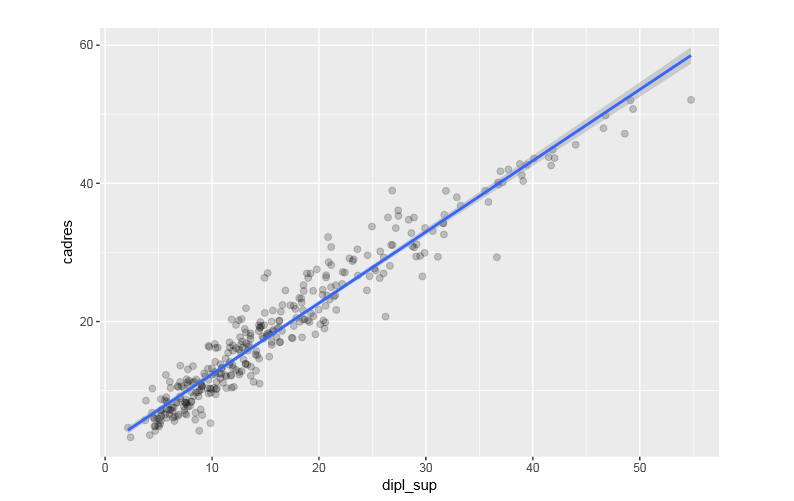


Autre exemple, on peut vouloir ajouter à un nuage de points une ligne de régression linéaire à l’aide de geom\_smooth :

ggplot(rp, aes(x = dipl\_sup, y = cadres)) +

geom\_point(alpha = 0.2) +

geom\_smooth(method = "lm")



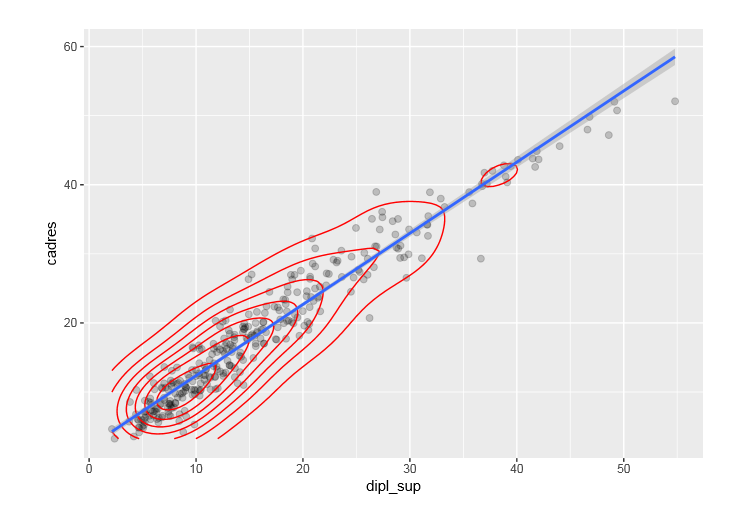
Et on peut même superposer une troisième visualisation de la répartition des points dans l’espace avec geom\_density2d :

ggplot(rp, aes(x = dipl\_sup, y = cadres)) +

geom\_point(alpha = 0.2) +

geom\_density2d(color = "red") +

geom\_smooth(method = "lm")



### Plusieurs sources de données

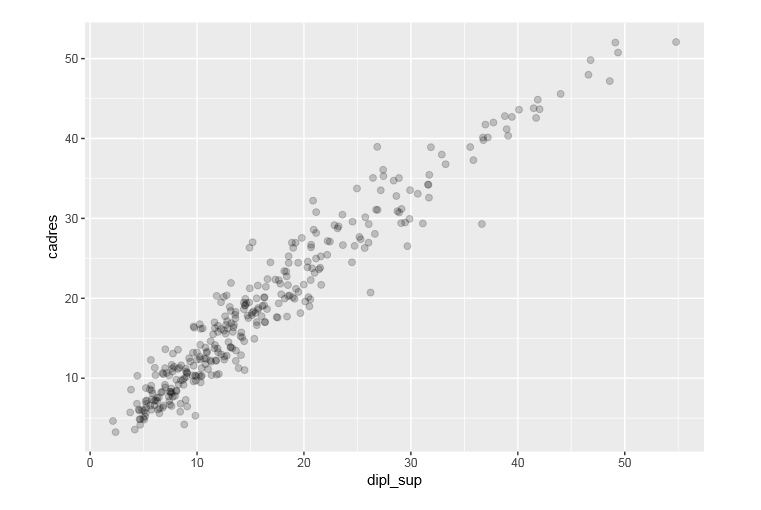
On peut aussi associer à différents geom des sources de données différentes. Supposons qu’on souhaite afficher sur un nuage de points les noms des communes de plus de 50000 habitants. On peut commencer par créer un tableau de données avec seulement ces communes à l’aide de la fonction filter :

com50 <- filter(rp, pop\_tot >= 50000)

On fait ensuite le nuage de points comme précédemment :

ggplot(data = rp, aes(x = dipl\_sup, y = cadres)) +

geom\_point(alpha = 0.2)



Pour superposer les noms de communes de plus de 50 000 habitants, on peut ajouter un geom\_text, mais en spécifiant que les données proviennent du nouveau tableau com50 et non de notre tableau initial rp. On le fait en passant un argument data spécifique à geom\_text :

ggplot(data = rp, aes(x = dipl\_sup, y = cadres)) +

geom\_point(alpha = 0.2) +

geom\_text(data = com50, aes(label = commune), color = "red", size = 3)

