R : Introduction à R — Analyse R | Cours 4

Ményssa Cherifa-Luron

2023-09-15

Contents

Introduction

Ce cours est conçu pour vous fournir une compréhension approfondie des techniques essentielles en R, allant de la manipulation des chaînes de caractères et la gestion des valeurs manquantes (NA), à la manipulation avancée des dates et des séries chronologiques, en passant par la visualisation des données avec `ggplot2`.

Vous découvrirez comment R, avec ses puissantes librairies telles que `lubridate` pour les dates et `xts` pour les séries chronologiques, facilite le traitement et l'analyse de données complexes. De plus, vous apprendrez à créer des graphiques expressifs et informatifs en utilisant les données `mtcars` comme cas d'étude.

Ce cours comprend également des instructions détaillées sur la sauvegarde et l'agrégation de graphiques, ainsi que des exercices pratiques pour consolider vos compétences en R.

Organisation

Contenu du Cours : - Démonstrations - Exercices

Ressources

- Rbloggers : Blog Populaire sur le langage R
- Datacamp : Plateforme d'apprentissage de la data science interactive
- Big Book of R : Edition qui rassemble des livres open-source sur le langage R
- Introduction accélérée au langage R pour la data science : L'essentiel et plus de tout ce que nous verrons

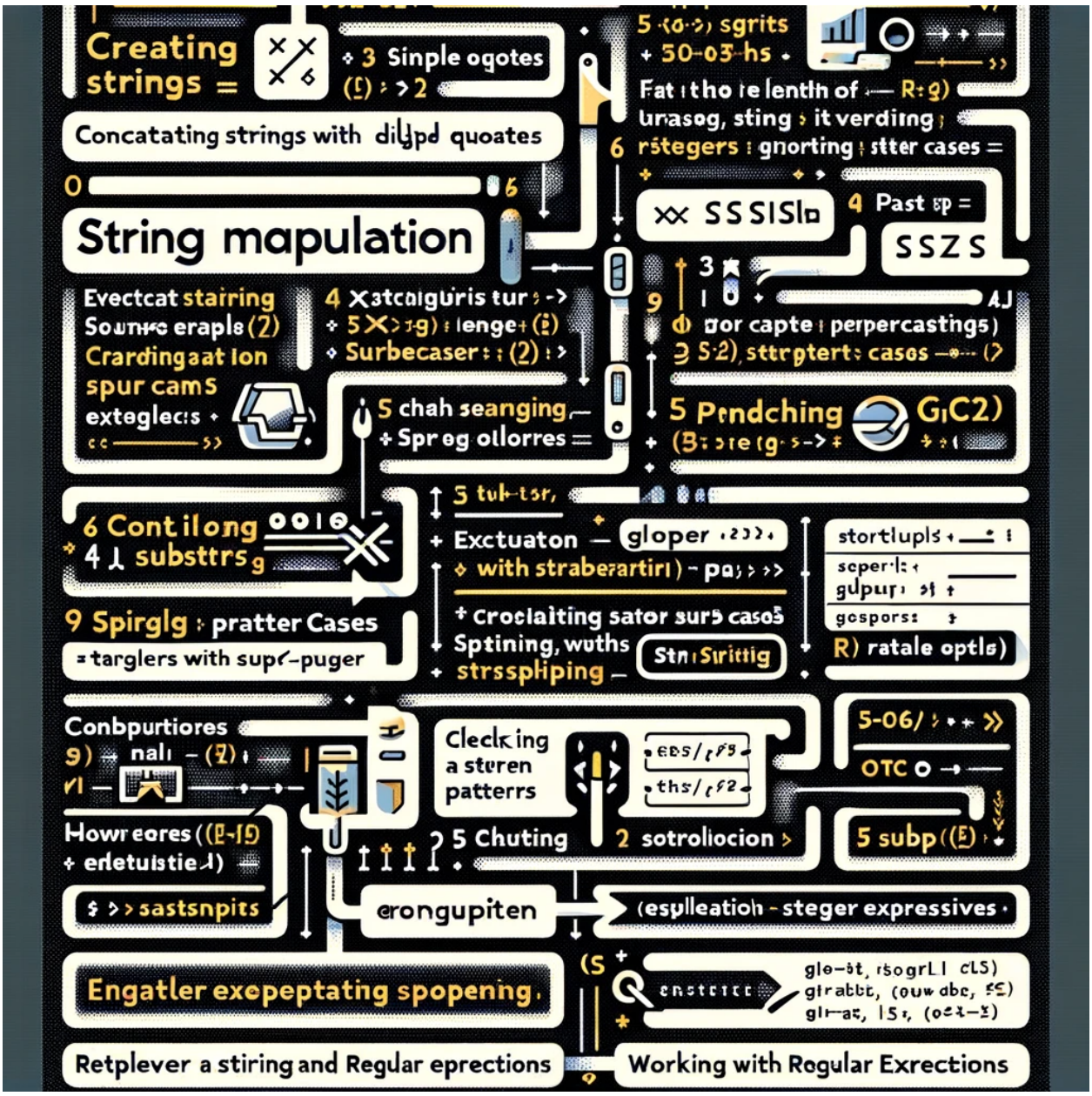


Figure 1: Illustration du langage R par DALL-E

- Regex 101
- Lise Vaudor- Regex
- Eric Gallic - Regex
- Couleurs avec R
- Couleurs avec R (2)
- Aides pour les plots

Manipulation des Chaînes de Caractères en R

Les chaînes de caractères, également appelées “strings”, sont utilisées pour stocker du texte en R. Vous pouvez créer des chaînes de caractères en utilisant des guillemets simples (’ ’) ou des guillemets doubles (” “).

Par exemple :

```
chaine_simple <- 'Ceci est une chaîne de caractères en guillemets simples.'
chaine_double <- "Ceci est une chaîne de caractères en guillemets doubles."
```

Notez que les guillemets simples et doubles sont équivalents pour créer des chaînes de caractères, mais il est important de les utiliser de manière cohérente.

Vous pouvez effectuer diverses opérations de base, telles que :

Fonctions de Manipulation de Rbase

- Concaténation : Vous pouvez joindre deux chaînes de caractères en les ajoutant ensemble à l’aide de l’opérateur + ou en utilisant la fonction `paste()`.

```
chaine1 <- "Hello"
chaine2 <- "World"
resultat <- paste(chaine1, chaine2)
```

- Longueur d’une chaîne : Vous pouvez obtenir la longueur d’une chaîne de caractères en utilisant la fonction `nchar()`.

```
chaine <- "Bonjour"
longueur <- nchar(chaine)  # longueur vaut 7
```

- Extraction de sous-chaînes : Vous pouvez extraire une partie d’une chaîne de caractères en utilisant la fonction `substr()` en spécifiant la position de départ et la longueur.

```
chaine <- "Manipulation de texte"
sous_chaine <- substr(chaine, start = 1, stop = 12)  # extrait les 12 premiers caractères
```

- Recherche de motifs : Vous pouvez vérifier si une chaîne de caractères contient un motif spécifique à l’aide de la fonction `grepl()`.

```
chaine <- "Analyse de données en R"
motif <- "R"
contient_R <- grepl(motif, chaine)  # contient_R vaut TRUE
```

- `tolower()` et `toupper()`: Ces fonctions permettent de convertir une chaîne de caractères en minuscules ou en majuscules, respectivement.

```
chaine <- "Texte à convertir"
en_minuscules <- tolower(chaine)  # en_minuscules vaut "texte à convertir"
en_majuscules <- toupper(chaine)  # en_majuscules vaut "TEXTE À CONVERTIR"
```

- `strsplit()`: Cette fonction permet de diviser une chaîne de caractères en un vecteur de sous-chaînes en utilisant un délimiteur spécifié.

```
chaine <- "Janvier,Février,Mars"
sous_chaines <- strsplit(chaine, ",")  # sous_chaines est une liste de vecteurs
```

- `gsub()`: Cette fonction permet de rechercher et de remplacer des motifs dans une chaîne de caractères.

```
chaine <- "Les chiens sont adorables, les chiens sont intelligents."
chaine_modifiee <- gsub("chiens", "chats", chaine)  # Remplace "chiens" par "chats"
```

Les Expressions Régulières (Regex)

Les expressions régulières, souvent abrégées en “regex” ou “regexp”, sont des motifs de recherche puissants et flexibles utilisés pour la recherche et la manipulation de texte dans les chaînes de caractères.

Les expressions régulières permettent de spécifier un modèle de caractères que vous souhaitez rechercher dans un texte. Elles sont largement utilisées dans la manipulation de texte, le traitement de données, la validation de formulaires, la recherche et bien d’autres domaines de l’informatique.

Voici une explication des éléments clés des expressions régulières :

1. Caractères littéraux : Les caractères littéraux dans une expression régulière correspondent exactement aux mêmes caractères dans le texte. Par exemple, l’expression régulière “chat” correspondra au mot “chat” dans un texte.
2. Méta-caractères : Les méta-caractères sont

des caractères spéciaux utilisés pour définir des modèles plus flexibles. Voici quelques méta-caractères couramment utilisés :

a. . (point) : Correspond à n’importe quel caractère, sauf le saut de ligne.

b. - (astérisque) : Correspond à zéro ou plusieurs occurrences du caractère précédent. Par exemple, “ab*c” correspondra à “ac”, “abc”, “abbc”, etc.

c. - (plus) : Correspond à une ou plusieurs occurrences du caractère précédent. Par exemple, “ab+c” correspondra à “abc”, “abbc”, etc., mais pas à “ac”.

d. ? (point d’interrogation) : Correspond à zéro ou une occurrence du caractère précédent. Par exemple, “ab?c” correspondra à “ac” et “abc”, mais pas à “abbc”.

e. [] (crochets) : Vous pouvez spécifier un ensemble de caractères possibles à cet endroit dans l’expression régulière. Par exemple, “[aeiou]” correspondra à n’importe quelle voyelle.

f. [^] (crochets inversés) : Inverse la correspondance. “[^aeiou]” correspondra à tout caractère qui n’est pas une voyelle.

g. () (parenthèses) : Permet de grouper des expressions régulières. Par exemple, “(ab)+” correspondra à “ab”, “abab”, “ababab”, etc.

3. Caractères d’échappement : Si vous souhaitez rechercher des méta-caractères littéralement, vous devez les échapper en les précédant d’un antislash.
4. Quantificateurs : Les quantificateurs (tels que *, +, ?, {n}, {n,} et {n,m}) permettent de spécifier combien de fois un motif doit se produire. Par exemple, a{2,4} correspondra à “aa”, “aaa” et “aaaa”, mais pas à “a” ou “aaaaa”.
5. Ancrages : Les ancrages (tels que ^ au début de l’expression régulière et $ à la fin) spécifient que la correspondance doit commencer ou se terminer à un endroit précis dans le texte. Par exemple, “^abc” correspondra à “abc” uniquement si “abc” est au début du texte.
6. Modificateurs : Les modificateurs permettent de définir des options pour les expressions régulières. Par exemple, i permet d’ignorer la casse, g permet de rechercher toutes les occurrences, et m permet de rechercher sur plusieurs lignes.
7. Recherches complexes : Les expressions régulières peuvent être combinées de manière complexe pour créer des modèles de recherche avancés. Par exemple, (ab|cd) correspondra à “ab” ou “cd”, et [A-Za-z]+ correspondra à une séquence de lettres en minuscules ou en majuscules.

Exemples d’utilisation des expressions régulières en R avec **stringr**

```r
library(stringr)

# Trouver un motif dans une chaîne de caractères
texte <- "Le numéro de téléphone de Jean est 555-123-4567. Le mien est 123-456-7890."
# Trouver tous les numéros de téléphone dans le texte
resultat <- str_extract_all(texte, "\\d{3}-\\d{3}-\\d{4}")
```

```r
# Remplacer un motif dans une chaîne de caractères
texte <- "Les pommes sont rouges, les bananes sont jaunes."
# Remplacer "rouges" par "vertes"
nouveau_texte <- str_replace(texte, "rouges", "vertes")
```

```r
# Extraire des parties spécifiques d'une chaîne de caractères
texte <- "Date de naissance : 25/03/1990"
# Extraire la date de naissance (xx/xx/xxxx)
date_naissance <- str_extract(texte, "\\d{2}/\\d{2}/\\d{4}")
```

Vérifions si une chaîne de caractères correspond à un motif spécifique, comme un numéro de sécurité sociale formaté en `xxx-xx-xxxx`.

```r
motif <- "\\d{3}-\\d{2}-\\d{4}"

# Chaînes à vérifier
chaine1 <- "123-45-6789"
chaine2 <- "abc-12-3456"

# Vérification du motif
est_valide1 <- str_detect(chaine1, motif)
est_valide2 <- str_detect(chaine2, motif)

# Affichage des résultats
print(est_valide1)  # Doit retourner TRUE
```

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

```
print(est_valide2)  # Doit retourner FALSE
```

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

Examinons comment extraire les noms d'utilisateur à partir d'adresses e-mail dans un texte.

```
texte <- "Les adresses e-mail de contact sont : john.doe@email.com, jane_smith@email.com."

# Extraction des adresses e-mail
utilisateurs <- str_extract_all(texte, "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}")

# Extraction des noms d'utilisateur
noms_utilisateurs <- str_extract_all(unlist(utilisateurs), "[A-Za-z0-9._%+-]+")
```

Remplaçons les caractères spéciaux dans une chaîne de caractères par des espaces.

```
texte <- "Ceci est un exemple de texte avec des caractères spéciaux : @#$%&*()_"

# Remplacement des caractères spéciaux
texte_propre <- str_replace_all(texte, "[@#$%&*()_]", " ")
```

Démontrons comment trouver des URL dans un texte en utilisant les expressions régulières.

```
texte <- "Voici quelques liens utiles : www.example.com, http://www.google.com, https://github.com."

# Extraction des URL
urls <- str_extract_all(texte, "https?://[A-Za-z0-9./]+")

# Affichage des URL trouvées
print(urls)
```

```
## [[1]]
## [1] "http://www.google.com" "https://github.com."
```

Supprimer les balises HTML d'une chaîne

```
html_text <- "<h1>Titre</h1> Ceci est un <b>texte</b> simple."
str_replace_all(html_text, "<[^>]+>", "")
```

```
## [1] "Titre Ceci est un texte simple."
```

Application : Analyse de sentiments

```
commentaires <- c("J'adore ce produit.",
                  "C'est le pire achat que j'ai jamais fait.",
                  "Pas mal, mais pourrait être mieux.",
```

```
                  "Un excellent service client !",
                  "Je ne recommanderais pas ce produit.")

commentaires <- tolower(commentaires)
commentaires <- gsub("[[:punct:]]", "", commentaires)

# 2. Analyse manuelle des sentiments
mots_positifs <- c("adore", "excellent")
mots_negatifs <- c("pire", "ne recommanderais pas")

sentiments <- c()

for(commentaire in commentaires) {
  score_positif <- sum(unlist(lapply(mots_positifs, function(word)
    grepl(word, commentaire))))
  score_negatif <- sum(unlist(lapply(mots_negatifs, function(word)
    grepl(word, commentaire))))

  if(score_positif > score_negatif) {
    sentiments <- c(sentiments, "Positif")
  } else {
    sentiments <- c(sentiments, "Négatif")
  }
}

# 3. Affichage des résultats
for(i in 1:length(commentaires)) {
  cat(paste(commentaires[i], ":", sentiments[i]), "\n")
}
```

```
## jadore ce produit : Positif
## cest le pire achat que jai jamais fait : Négatif
## pas mal mais pourrait être mieux : Négatif
## un excellent service client  : Positif
## je ne recommanderais pas ce produit : Négatif
```

Pour couvrir la gestion des données `NA` (valeurs manquantes) en R dans un document RMarkdown, voici un exemple de formatage comprenant des explications et des blocs de code R :

Gestion des Données NA (Valeurs Manquantes) en R

La gestion des valeurs manquantes est cruciale dans l'analyse de données. En R, `NA` représente une valeur manquante ou indisponible.

Détecter les NA

Utilisez `is.na()` pour identifier où se trouvent les valeurs manquantes dans vos données.

```
# Création d'un vecteur avec des valeurs NA
vecteur <- c(1, NA, 3, NA, 5)
```

```r
# Vérification des valeurs NA
na_positions <- is.na(vecteur)
print(na_positions)
```

```
## [1] FALSE  TRUE FALSE  TRUE FALSE
```

Exclure les NA dans les Calculs

Beaucoup de fonctions en R ont un argument pour gérer les NA. Par exemple, `na.rm = TRUE` permet d'exclure les NA dans les calculs.

```r
# Calcul de la moyenne en excluant les NA
moyenne_sans_na <- mean(vecteur, na.rm = TRUE)
print(moyenne_sans_na)
```

```
## [1] 3
```

Remplacement des NA

Vous pouvez remplacer les NA par une autre valeur en utilisant `na.omit()` ou `replace()`.

```r
# Remplacement des NA par une valeur spécifique (par exemple 0)
vecteur_sans_na <- replace(vecteur, is.na(vecteur), 0)
print(vecteur_sans_na)
```

```
## [1] 1 0 3 0 5
```

Utilisation de `na.omit()`

`na.omit()` retire les éléments NA d'un objet.

```r
# Retrait des NA d'un vecteur
vecteur_omit_na <- na.omit(vecteur)
print(vecteur_omit_na)
```

```
## [1] 1 3 5
## attr(,"na.action")
## [1] 2 4
## attr(,"class")
## [1] "omit"
```

Gestion des NA dans un Data Frame

Dans un data frame, vous pouvez retirer les lignes ou les colonnes contenant des NA.

```r
# Création d'un data frame avec des NA
df <- data.frame(x = c(1, 2, NA, 4), y = c("a", "b", "c", NA))

# Retrait des lignes avec des NA
df_sans_na <- na.omit(df)
print(df_sans_na)
```

```
##   x y
## 1 1 a
## 2 2 b
```

La gestion correcte des valeurs manquantes est essentielle pour assurer l'intégrité et la précision de vos analyses de données.

Dates et Heures avec lubridate

L'utilisation de la bibliothèque `lubridate` facilite le travail avec les dates et heures en R.

Conversion de Chaînes de Caractères en Dates

```r
# Installation et chargement de la bibliothèque lubridate si nécessaire
# install.packages("lubridate")
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
# Convertir une chaîne au format "année-mois-jour"
date_test <- ymd("2023-10-17")
date_test
```

```
## [1] "2023-10-17"
```

```r
class(date_test)
```

```
## [1] "Date"
```

Conversion avec Heure et Minute

```r
# Convertir une chaîne au format "jour-mois-année heure:minute"
date_test_heure <- dmy_hm("17-10-2023 14:30")
date_test_heure
```

```
## [1] "2023-10-17 14:30:00 UTC"
```

Création de Séries Chronologiques

```
# Utilisation des fonctions days, months, et years pour créer des dates
dates_jours <- as.Date("2023-01-01") + days(0:100)
dates_mois <- as.Date("2023-01-01") + months(2)
dates_years <- as.Date("2023-01-01") + years(2)
dates_jours_mois_annee <- as.Date("2023-01-01") + days(5) + months(6) + years(7)
```

Extraction de Composants Spécifiques d'une Date

```
# Extraire le jour, le mois et l'année d'une date
date <- ymd("2023-10-17")
day(date)
```

```
## [1] 17
```

```
month(date)
```

```
## [1] 10
```

```
year(date)
```

```
## [1] 2023
```

Calcul d'Intervalle entre Deux Dates

```
# Calculer l'intervalle entre deux dates
date1 <- ymd("2023-10-17")
date2 <- ymd("2022-10-17")
interval(date1, date2)
```

```
## [1] 2023-10-17 UTC--2022-10-17 UTC
```

Operations sur les Dates et Heures

```
date1 <- ymd("2023-11-06")
date2 <- ymd("2023-11-07")
plus_recente <- max(date1, date2)
plus_ancienne <- min(date1, date2)
heure1 <- hms("14:30:00")
heure2 <- hms("15:45:30")
duree <- heure2 - heure1
nouvelle_date <- date1 + days(2)
nouvelle_heure <- heure1 - hours(1)
```

```r
print(plus_recente)
```

```
## [1] "2023-11-07"
```

```r
print(plus_ancienne)
```

```
## [1] "2023-11-06"
```

```r
print(duree)
```

```
## [1] "1H 15M 30S"
```

```r
print(nouvelle_heure)
```

```
## [1] "13H 30M 0S"
```

```r
print(nouvelle_date)
```

```
## [1] "2023-11-08"
```

Synthèse

Tableau récapitulatif des fonctions de `lubridate` pour l'extraction des composants d'une donnée de type date.

Composants	Fonction pour l'extraire
Année	`year()`
Mois	`month()`
Semaine	`week()`
Jour de l'année	`yday()`
Jour du mois	`mday()`
Jours de semaine	`wday()`
Heure	`hour()`
Minute	`minute()`
Seconde	`second()`
Fuseau horaire	`tz()`

Ces fonctions sont utilisées pour extraire et manipuler les différentes parties d'une date dans le cadre de l'analyse de données en R.

Séries Chronologiques avec xts

Manipulation et visualisation des séries chronologiques en utilisant la bibliothèque `xts`.

Sélection dans une Série Chronologique

```r
# Installation et chargement de la bibliothèque xts si nécessaire
# install.packages("xts")
library(xts)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

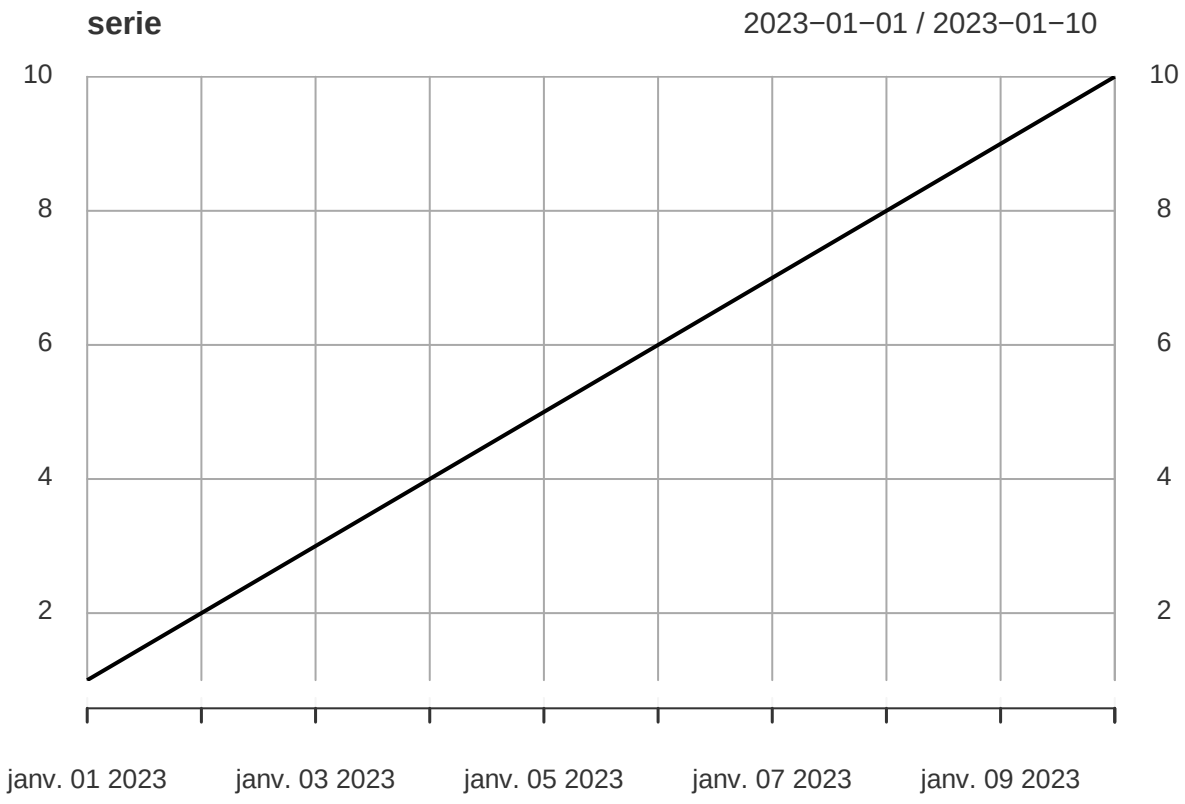
```r
# Création et sélection dans une série chronologique
dates <- as.Date("2023-01-01") + 0:9
serie <- xts(1:10, order.by=dates)
subset(serie, index(serie) >= "2023-01-05" & index(serie) <= "2023-01-08")
```

```
##            [,1]
## 2023-01-05    5
## 2023-01-06    6
## 2023-01-07    7
## 2023-01-08    8
```

Affichage Graphique de la Série Chronologique

```r
# Installation et chargement de la bibliothèque ggplot2 si nécessaire
# install.packages("ggplot2")
library(ggplot2)

# Affichage graphique de la série chronologique
plot(serie)
```



Visualisation des données mtcars avec R

• Conventions usuelles en data viz • Graphiques Usuels : Barplot, Histogramme, Scatter plot, Boxplot et Courbes • Syntaxe de R base avec et ggplot2

Conventions usuelles

La data viz est autant **un art qu'une science**.

Des conventions établies peuvent aider à garantir que les graphiques soient clairs, pertinents et accessibles.

Voici quelques conventions et recommandations générales

Titre, étiquettes, axes

Chaque graphique doit avoir un titre descriptif. Axes étiquetés avec des noms de variables ou des descriptions. Si les unités sont pertinentes (comme les années, les dollars, etc.), elles doivent être incluses. Assurez-vous que l'échelle des axes est appropriée pour vos données. Évitez les échelles tronquées qui peuvent tromper l'œil. Considérez l'ajout d'annotations pour mettre en évidence des points ou des tendances importants.

Légende

Si le graphique contient plusieurs séries ou catégories, une légende claire et bien positionnée est essentielle. Évitez d'encombrer le graphique avec une légende trop grande.

Simplicité

Évitez de surcharger le graphique avec trop d'informations. Limitez le nombre de couleurs et de motifs utilisés. Utilisez des palettes de couleurs daltoniens "friendly". Assurez-vous que la couleur ait un sens dans le contexte de vos données. Évitez d'utiliser des couleurs trop vives ou distrayantes.

Ordre des données

Dans les barplots, envisagez d'ordonner les barres par taille plutôt que de manière aléatoire ou alphabétique, sauf si un autre ordre est logique (par exemple, les mois de l'année). Clarté des données :

Consistence et sources

Si vous présentez plusieurs graphiques dans un rapport ou une présentation, essayez de maintenir une certaine cohérence en termes de couleurs, de styles et de typographie.

Si les données proviennent d'une source externe, cette source devrait être clairement indiquée.

Supprimez les grilles, les bordures et les étiquettes qui n'ajoutent pas de valeur. Évitez les "camemberts" 3D et autres graphiques qui peuvent déformer la perception des données.

Installation des packages essentiels : tidyverse

```r
# Installation et chargement du tidyverse qui contient ggplot2 et
# Installation de gridExtra pour combiner les plots
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr   1.1.2     v readr   2.1.4
## v forcats 1.0.0     v tibble  3.2.1
## v purrr   1.0.1     v tidyr   1.3.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::first()  masks xts::first()
## x dplyr::lag()    masks stats::lag()
## x dplyr::last()   masks xts::last()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

ggplot2 est une librairie spécialisée pour la visualisation de données en R. Elle utilise une syntaxe différente des graphiques de base et permet de créer des visualisations plus complexes et esthétiques.

Les données mtcars

```
# Description des données mtcars
?mtcars
```

```
## starting httpd help server ... done
```

```
mtcars <- datasets::mtcars
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
# Recodage des variables catégorielles
# transmission et moteur
mtcars$vs <- factor(mtcars$vs,levels = 0:1,labels = c("V-shaped", "Straight"))
mtcars$am <- factor(mtcars$am,levels = 0:1,labels = c("Automatic", "Manual"))

# Check
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : Factor w/ 2 levels "V-shaped","Straight": 1 1 2 2 1 2 1 2 2 2 ...
##  $ am  : Factor w/ 2 levels "Automatic","Manual": 2 2 2 1 1 1 1 1 1 1 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Barplot (diagramme à barres)

Le diagramme à barres est utilisé pour représenter graphiquement des données catégorielles ou discrètes sous forme de barres verticales ou horizontales.

Il permet de visualiser la fréquence, la distribution ou la comparaison de différentes catégories. Chaque barre représente une catégorie, et la hauteur (ou la longueur) de la barre est proportionnelle à la valeur associée à cette catégorie.

Ici, nous examinons le nombre de voitures en fonction du type de moteur.

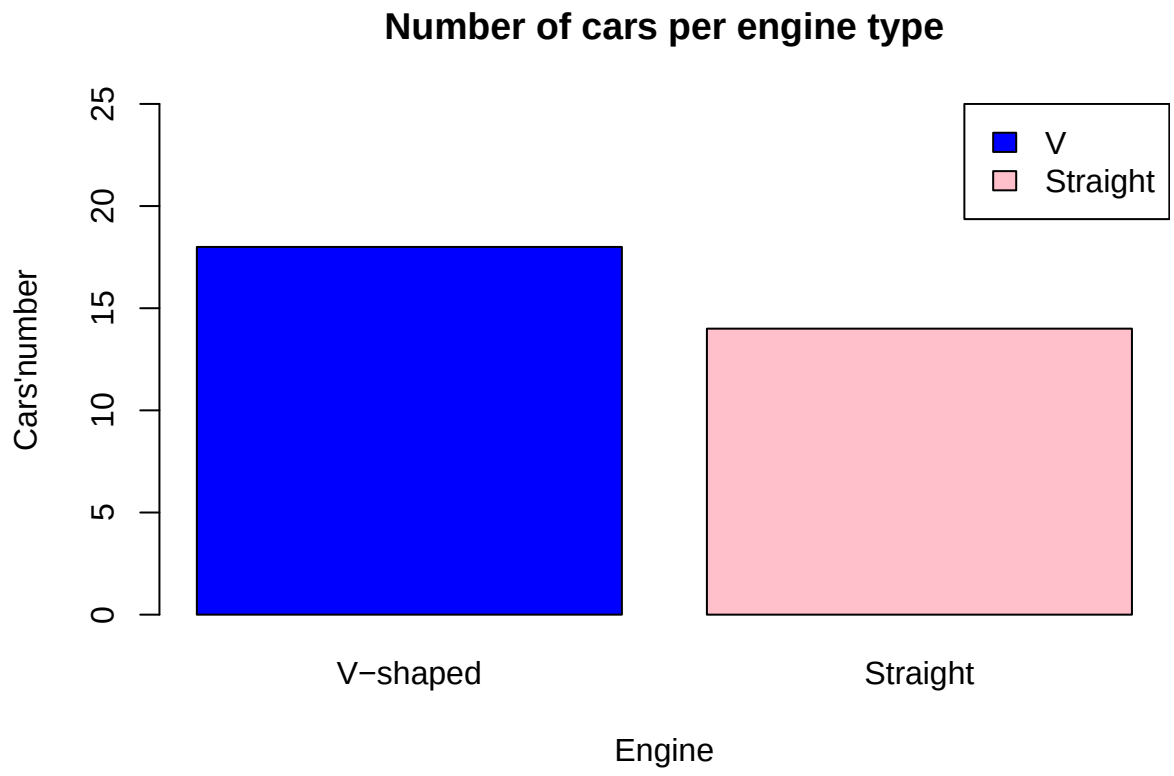
```
# Calcul de la distribution
distribution_moteur <- table(mtcars$vs)
typeof(distribution_moteur)
```

```
## [1] "integer"
```

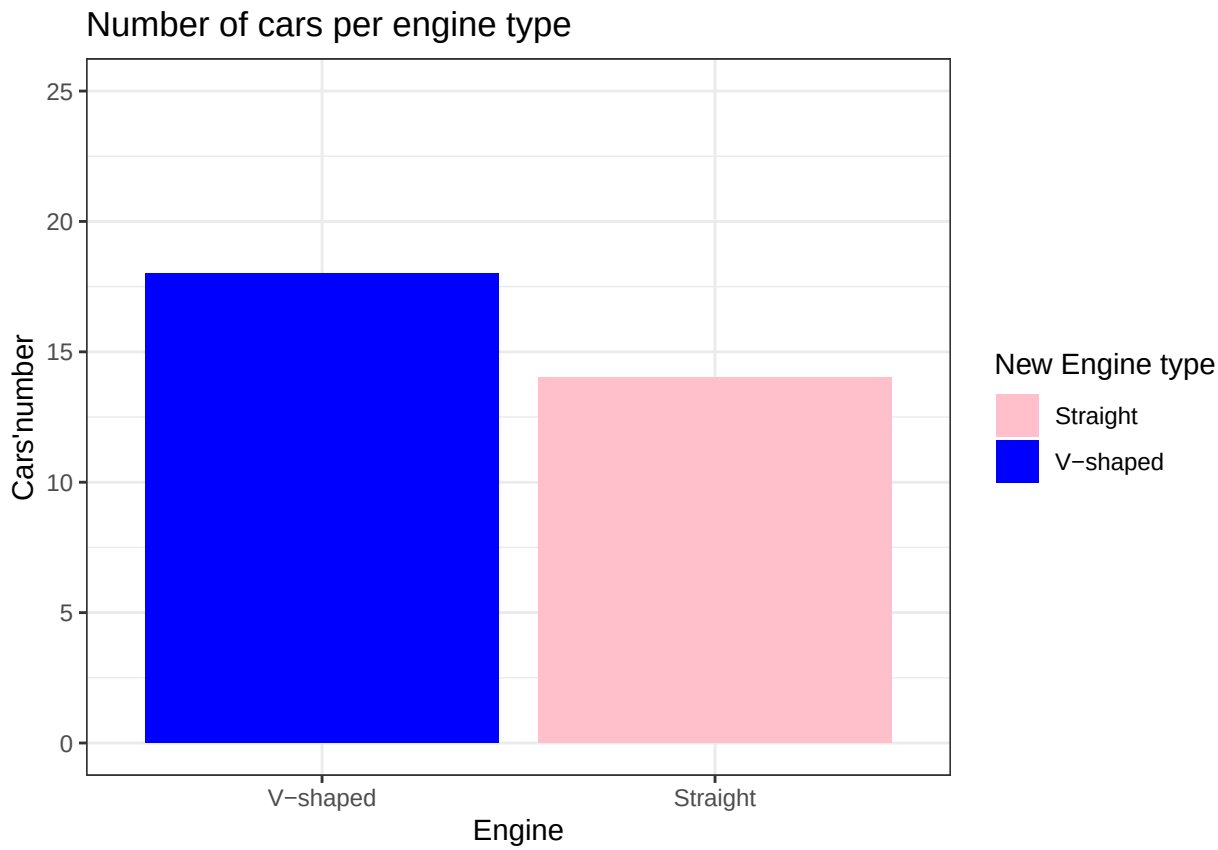
```
class(distribution_moteur)
```

```
## [1] "table"
```

```
# Barplot R base
barplot(distribution_moteur, # table
        main = "Number of cars per engine type",  # titre
        xlab = "Engine", # nom axe x
        ylab = "Cars'number",
        ylim = c(0, 25), # echelle y
        col=c("blue", "pink")) # couleur : check r color book
# Creation de la légende
legend(x="topright",legend=c("V","Straight"),
       fill=c("blue","pink"))
```



```r
# Barplot ggplot2
ggplot(mtcars, aes(x = vs, fill = vs)) +
  geom_bar() +
  labs(title = "Number of cars per engine type",
       x = "Engine",
       y = "Cars'number",
       fill = "Engine type") +
  theme_bw() + # choix du theme
  # changement manuel des couleurs
  scale_fill_manual(values = c("blue", "pink"),
                    name = "New Engine type",
                    guide = guide_legend(reverse = TRUE)) + ylim(0,25)
```



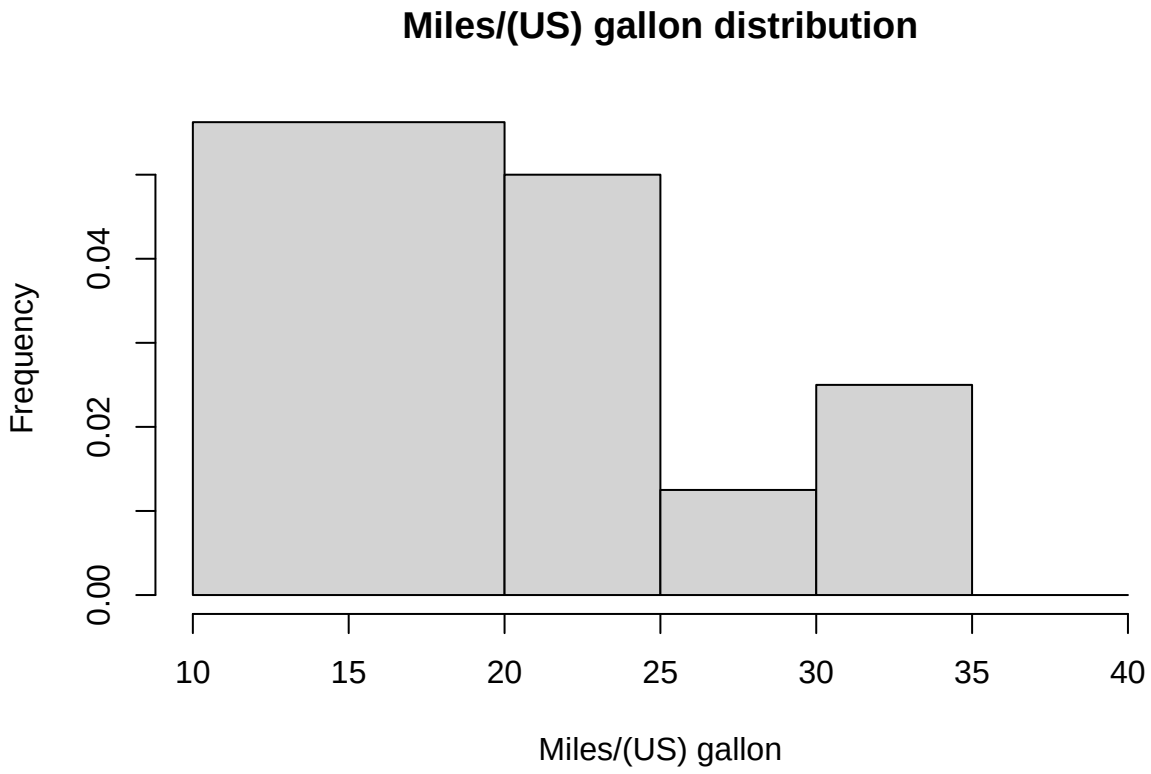
Histogramme

L'histogramme est utilisé pour représenter graphiquement la distribution des données continues en regroupant les données en classes et en montrant la fréquence ou la densité de ces classes.

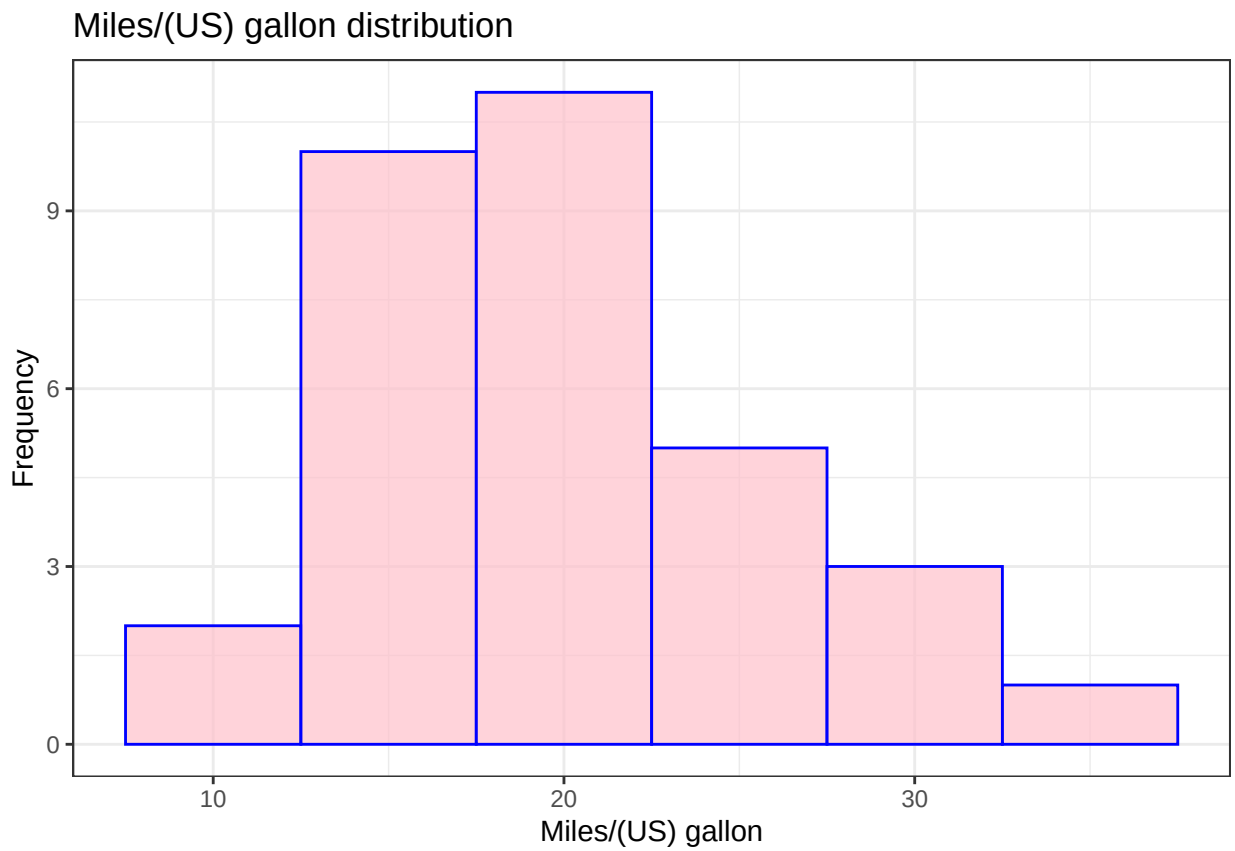
Nous examinons ici la distribution du nombre de miles par gallon.

```r
# Histogramme avec R base
hist(mtcars$mpg,
     breaks = c(10, 20, 25, 30, 35,40),
     main = "Miles/(US) gallon distribution",
```

```
     xlab = "Miles/(US) gallon",
     ylab = "Frequency")
```




```
# Histogramme avec ggplot2
ggplot(mtcars, aes(x = mpg)) +
  # spécifie la largeur et la color pour distinguer les "bins"
  # alpha permet de géré la transparence
  geom_histogram(binwidth = 5,  fill="pink", color="blue", alpha=0.7) +
  labs(title = "Miles/(US) gallon distribution",
       x = "Miles/(US) gallon",
       y = "Frequency") +
  theme_bw()
```



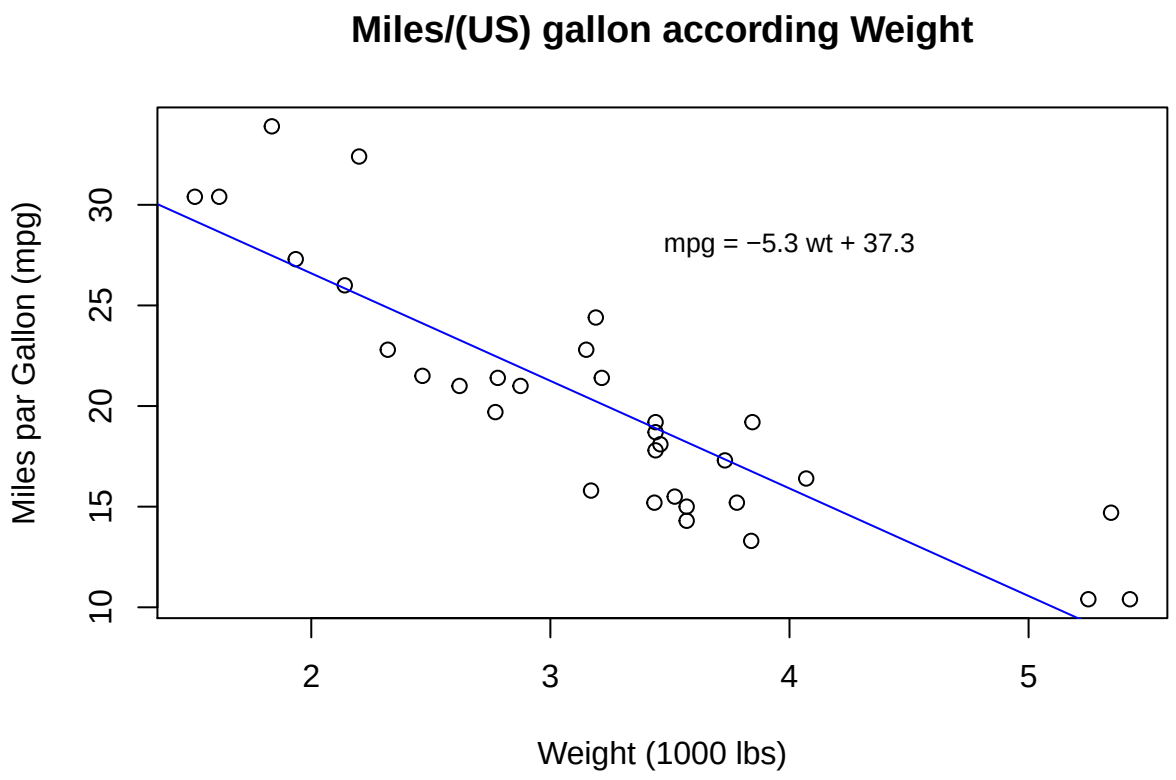
Scatter plot (nuage de points)

Le diagramme de dispersion est utilisé pour afficher la relation entre deux variables continues, généralement une variable sur l'axe horizontal et l'autre sur l'axe vertical.

Il permet d'observer la relation entre deux variables, d'identifier les tendances, les corrélations ou les groupes de données, et de repérer les valeurs aberrantes.

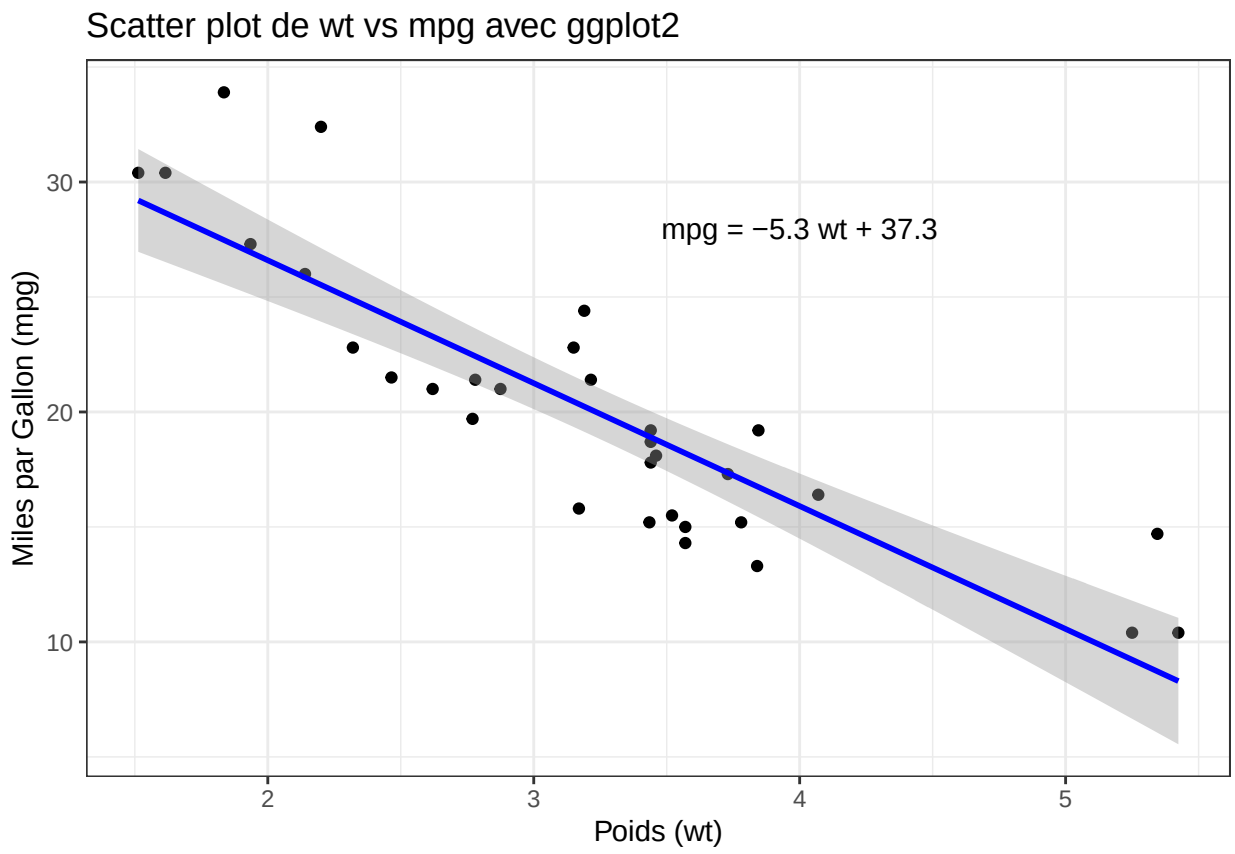
Comparons ici le poids des voitures selon leur consommation de carburant.

```
# Scatter plot avec R base
plot(x = mtcars$wt,y = mtcars$mpg,
     main = "Miles/(US) gallon according Weight",
     xlab = "Weight (1000 lbs)",
     ylab = "Miles par Gallon (mpg)")
abline(lm(mpg ~ wt, data = mtcars), col = "blue")
text(x = 4, y = 28, "mpg = -5.3 wt + 37.3", cex = .8)
```




```
# Scatter plot avec ggplot2
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method="lm", se=TRUE, color="blue") +
  labs(title = "Scatter plot de wt vs mpg avec ggplot2",
       x = "Poids (wt)",
       y = "Miles par Gallon (mpg)") +
  theme_bw() +
  annotate("text", x=4, y=28, label="mpg = -5.3 wt + 37.3")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



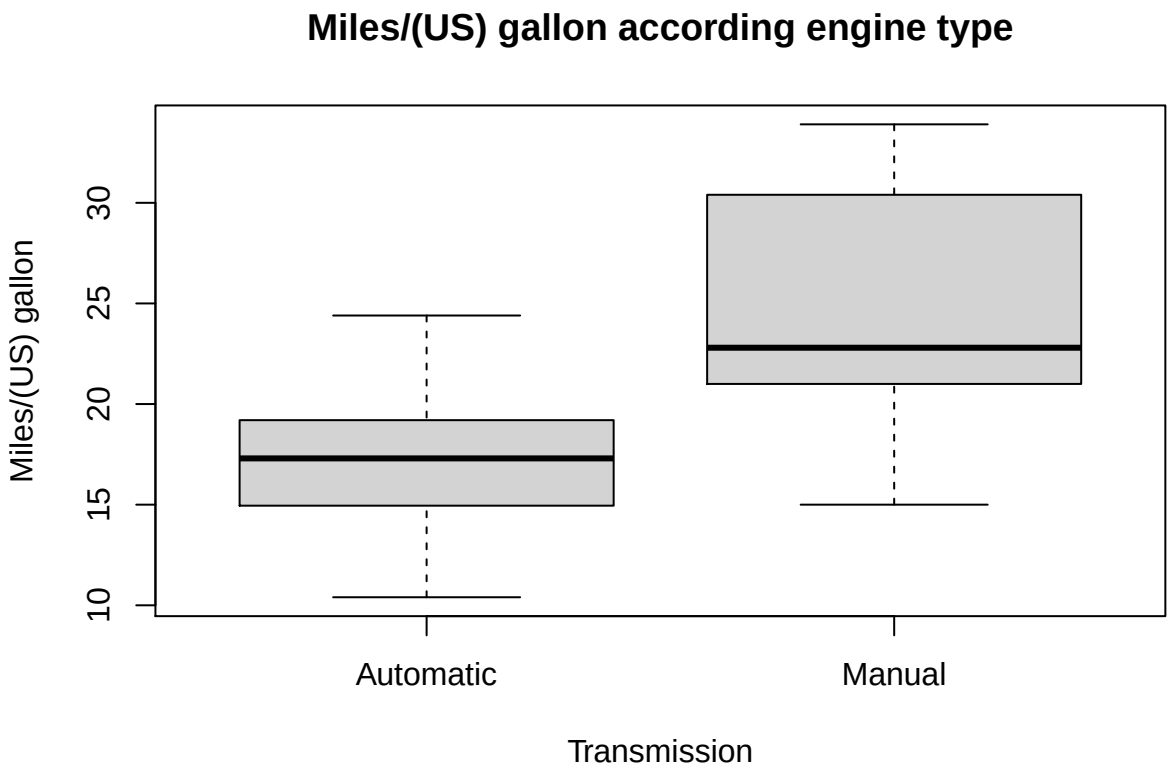
Boxplot

La boîte à moustaches est utilisée pour représenter graphiquement la distribution et les mesures de dispersion d'une variable continue, ainsi que pour détecter les valeurs aberrantes.

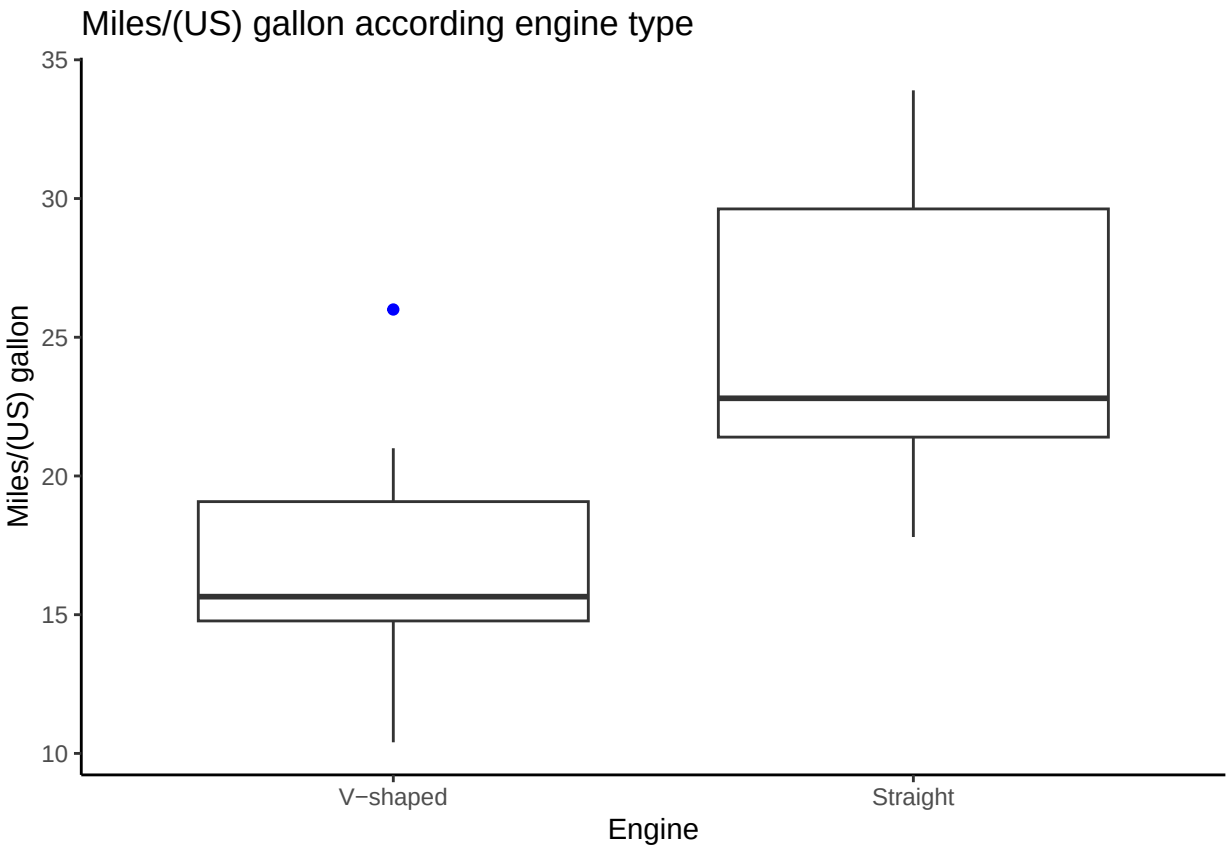
Elle affiche la médiane, les quartiles, les valeurs minimales et maximales de la distribution, ainsi que les valeurs potentielles aberrantes sous forme de points individuels.

Examinons la distribution des mpg selon le type de moteur

```
# Boxplot R base
boxplot(mpg ~ am,
        data = mtcars,
        main = "Miles/(US) gallon according engine type",
        xlab = "Transmission",
        ylab = "Miles/(US) gallon")
```




```
# Boxplot ggplot2
ggplot(mtcars, aes(y = mpg, x = vs)) +
  geom_boxplot(outlier.colour = "blue") +
  labs(title = "Miles/(US) gallon according engine type",
       x = "Engine",
       y = "Miles/(US) gallon") +
  theme_classic()
```



Courbes, droites

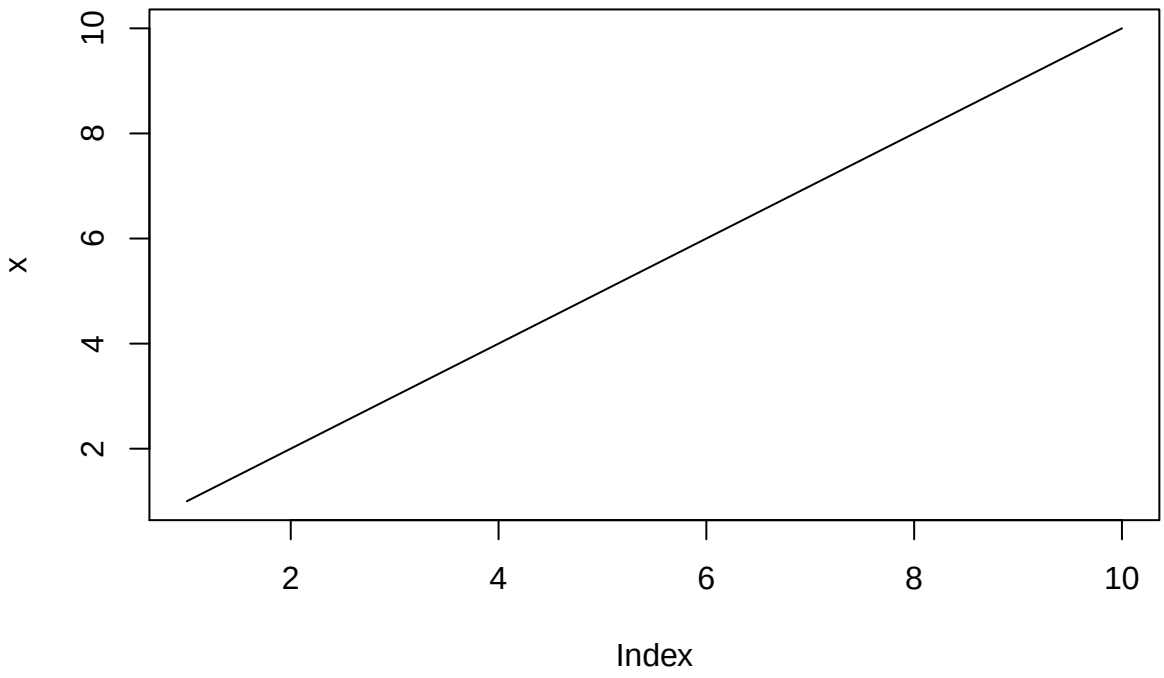
Les courbes sont utilisées pour représenter graphiquement des données continues dans des graphiques linéaires, logarithmiques ou d'autres transformations.

Elles permettent de visualiser la tendance générale des données au fil du temps ou d'autres variables indépendantes. Les courbes peuvent être utilisées pour montrer des relations, des modèles de croissance, des fluctuations, etc.

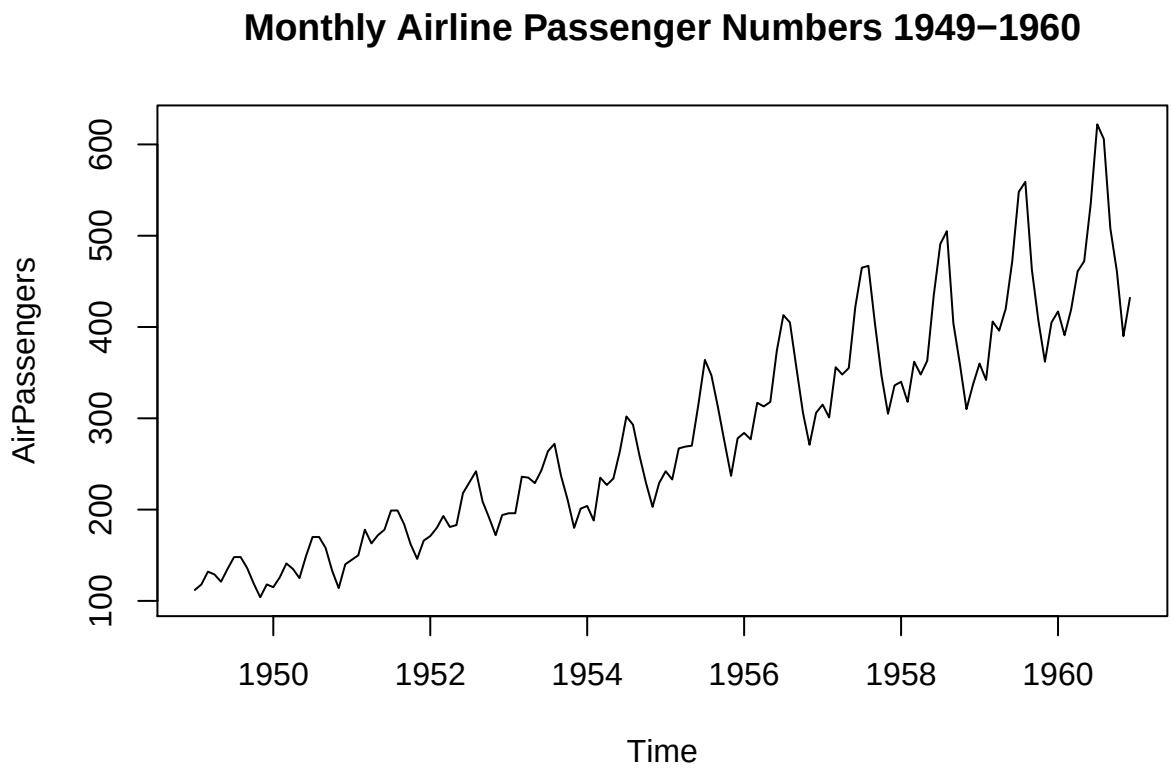
Examinons la distribution des mpg selon le nombre de cylindres.

```
?AirPassengers

x = 1:10
plot(x, type = "l")
```

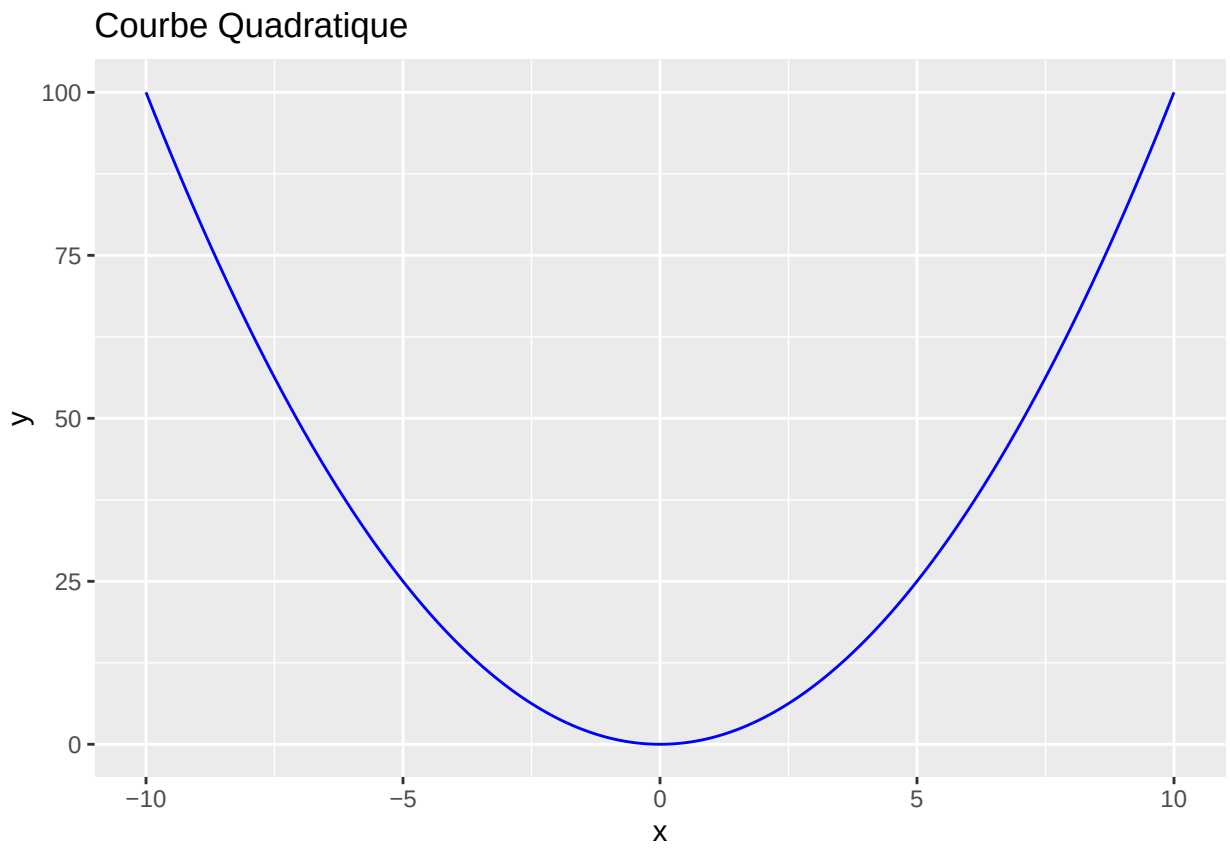



```
plot(AirPassengers,
     main="Monthly Airline Passenger Numbers 1949-1960")
```




```
data <- data.frame(x=seq(-10, 10, by=0.1))
data$y <- data$x^2

gg1 <- ggplot(data, aes(x=x, y=y)) +
  geom_line(col="blue") +
  ggtitle("Courbe Quadratique")
gg1
```



Sauvegarder et Aggréger des Graphiques

Les graphiques sont d'abord générés, puis sauvegardés dans des fichiers spécifiés, tels que des PDF ou des PNG. Pour `ggplot2`, `gridExtra` est utilisé pour combiner plusieurs graphiques en un seul affichage.

Graphiques avec R base

Sauvegarde et création de layouts pour des graphiques avec les fonctions de base de R.

```
# Enregistrement des graphiques en format PDF
pdf(file = "plotRbase.pdf")

# Configuration du layout en 2x2
par(mfrow=c(2,2))

# Premier graphique
plot(1:10, rnorm(10), main="Graphique 1")

# Deuxième graphique
plot(1:10, rnorm(10), col="blue", main="Graphique 2")

# Troisième graphique
hist(rnorm(100), main="Histogramme 1")
```

```r
# Quatrième graphique
boxplot(rnorm(100), main="Boîte à moustaches")

# Fermeture du dispositif graphique
dev.off()
```

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

```r
# Enregistrement des graphiques en format PNG
png(file = "plotRbase.png")

par(mfrow=c(2,2))

plot(1:10, rnorm(10), main="Graphique 1")

plot(1:10, rnorm(10), col="blue", main="Graphique 2")

hist(rnorm(100), main="Histogramme 1")

boxplot(rnorm(100), main="Boîte à moustaches")

dev.off()
```

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

Graphiques avec Ggplot

Création et sauvegarde de graphiques élaborés avec `ggplot2`.

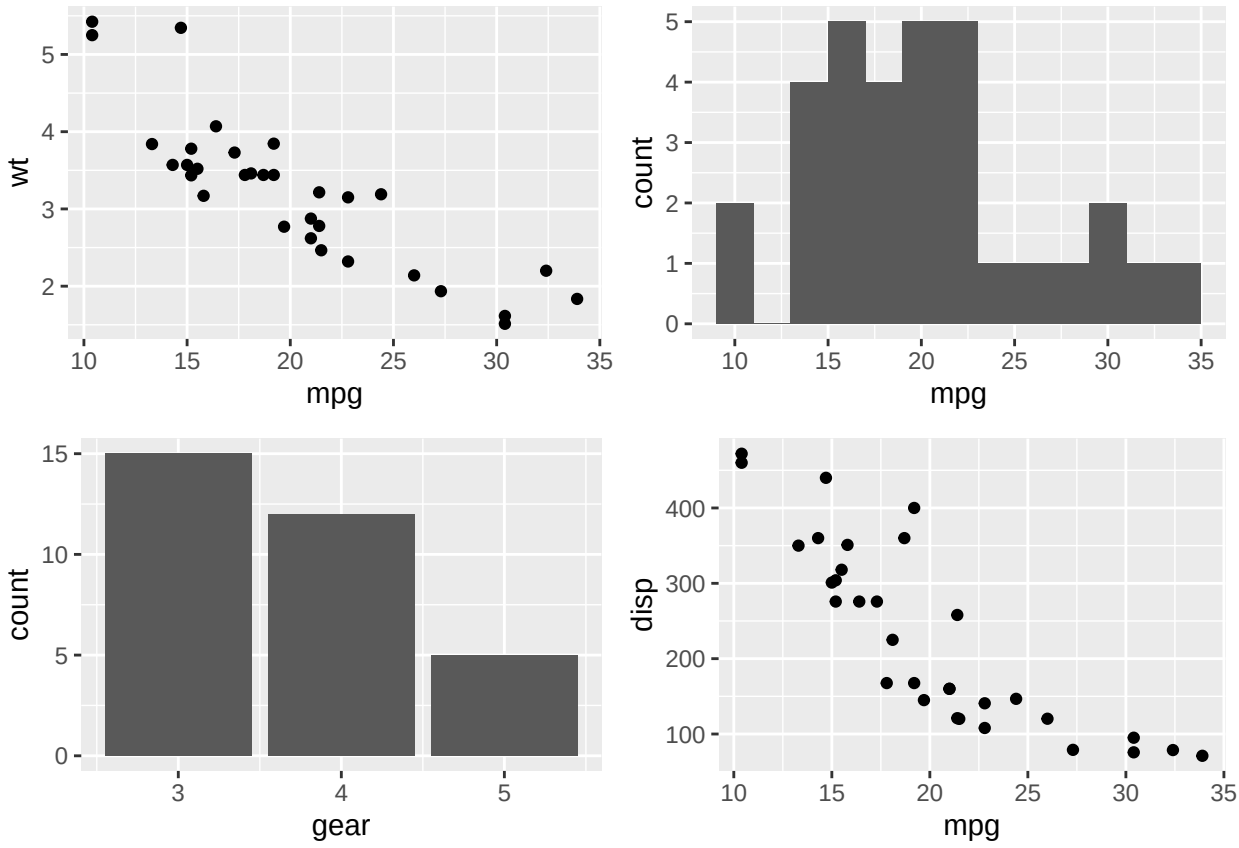
```r
library(gridExtra)

# Création d'un graphique ggplot et sauvegarde
gg1 <- ggplot(data, aes(x=x, y=y)) +
  geom_line(col="blue") +
  ggtitle("Courbe Quadratique")
ggsave("../resultats/mon_ggplot.png", plot = gg1)
```

```
## Saving 6.5 x 4.5 in image
```

```r
# Création de plusieurs ggplots
p1 <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
p2 <- ggplot(mtcars, aes(mpg)) + geom_histogram(binwidth=2)
p3 <- ggplot(mtcars, aes(gear)) + geom_bar()
p4 <- ggplot(mtcars, aes(mpg, disp)) + geom_point()

# Combinaison des ggplots dans un seul graphique
gg2 <- grid.arrange(p1, p2, p3, p4, ncol=2)
```




```
ggsave("mon_ggplot2.pdf", plot = gg2)
```

```
## Saving 6.5 x 4.5 in image
```

Exercices

Pour commencer à pratiquer, suivez ces étapes :

1. **Accédez au Dépôt GitHub :** Visitez l'URL fournie : https://github.com/universdesdonnees/Introduction-a-R pour accéder au dépôt GitHub contenant les matériaux du cours.
2. **Trouvez le Fichier des Exercices :** Dans le dépôt, localisez le fichier nommé **exercices4.txt**. Ce fichier contient les premiers exercices que vous devez pratiquer.
3. **Lisez et Essayez de Résoudre les Exercices :** Ouvrez le fichier **exercices4.txt** et lisez attentivement les exercices. Essayez de les résoudre par vous-même dans votre environnement R (comme RStudio). Il est important de pratiquer par vous-même avant de regarder les solutions pour mieux apprendre.
4. **Consultez la Correction :** Une fois que vous avez tenté de résoudre les exercices, ou si vous rencontrez des difficultés, consultez le fichier **correction_exercices3.R** pour voir les solutions. Analysez les solutions pour comprendre les méthodes et logiques utilisées.