# Estatística e Experimentação no Ambiente R

## 1. Introdução ao R

### 1.1 O que é o R?

O R é uma linguagem de programação e ambiente de computação estatística amplamente utilizado para análise de dados e gráficos. Ele é uma ferramenta de código aberto que fornece uma variedade de funções estatísticas e recursos para manipular, visualizar e modelar dados. O R é apreciado por sua flexibilidade e extensibilidade, permitindo aos usuários desenvolver seus próprios pacotes e funções. É uma escolha popular entre estatísticos, cientistas de dados e pesquisadores de diversas áreas devido à sua capacidade de lidar com uma ampla gama de tarefas analíticas e sua comunidade ativa de desenvolvedores e usuários. Durante esta aula, vamos explorar os conceitos fundamentais do R e como utilizá-lo para realizar análises estatísticas e experimentações.

### 1.2 Instalação e configuração básica.

Antes de começar a utilizar o R, é necessário instalar o software e configurar o ambiente. A instalação do R é simples e está disponível para várias plataformas (Windows, macOS, Linux). Após a instalação, é comum utilizar um ambiente de desenvolvimento integrado (IDE) como o RStudio, que facilita a interação com o R. A configuração básica envolve a definição de diretórios de trabalho, opções de exibição e a instalação de pacotes adicionais. Com um ambiente de trabalho bem configurado, você estará pronto para começar a escrever código em R, realizar análises de dados e executar experimentações estatísticas.

### 1.3 Interface do RStudio.

O RStudio é uma das interfaces mais populares para o ambiente R. Ele oferece um ambiente integrado que simplifica o uso do R, tornando-o mais amigável e produtivo. A interface do RStudio é composta por várias janelas e painéis:

Console R: O console é onde você digita e executa comandos R. Você pode ver a saída imediata dos comandos aqui.  
  
Editor de Script: Permite que você escreva, edite e salve scripts R. É útil para criar código reutilizável e documentado.  
  
Ambiente e Histórico: Mostra uma lista de objetos carregados na sessão atual e o histórico de comandos usados.  
  
Painel de Plots: Exibe gráficos e visualizações gerados a partir dos comandos R.  
  
Painel de Pacotes: Gerencia a instalação e carregamento de pacotes R adicionais.  
  
Painel de Arquivos e Ajuda: Permite navegar em diretórios, acessar ajuda online e visualizar arquivos.  
  
Configurações Globais: Aqui, você pode personalizar configurações, temas e atalhos do RStudio.

A interface do RStudio é altamente personalizável para atender às suas necessidades e fluxo de trabalho. Ele oferece recursos avançados, como depuração de código, controle de versão integrado e suporte a projetos. Dominar a interface do RStudio é essencial para se tornar produtivo na análise de dados e programação em R.

### 1.4 Instalação de pacotes.

Os pacotes são extensões de funcionalidades que podem ser adicionadas ao ambiente R. Eles contêm funções, conjuntos de dados e documentação relacionada a tarefas específicas. Para instalar um pacote em R, você pode usar a função install.packages(). Veja como funciona:

Seleção do Pacote: Primeiro, você precisa decidir qual pacote deseja instalar. Escolha pacotes que atendam às necessidades da sua análise de dados ou experimentação.  
  
Instalação do Pacote: Utilize o comando install.packages("nome\_do\_pacote"), substituindo "nome\_do\_pacote" pelo nome do pacote desejado. Por exemplo, para instalar o pacote "ggplot2", você digitaria install.packages("ggplot2").  
  
Seleção do Repositório: O R oferece vários repositórios de pacotes. Você pode selecionar um repositório de onde deseja baixar o pacote ou simplesmente pressionar Enter para usar o repositório padrão, o CRAN (Comprehensive R Archive Network).  
  
Aprovação de Dependências: O RStudio geralmente perguntará se você deseja instalar pacotes dependentes necessários para o funcionamento do pacote selecionado. Confirme a instalação dessas dependências, se solicitado.  
  
Conclusão: Após a instalação bem-sucedida, o pacote estará pronto para ser carregado e usado em sua sessão R. Você pode carregá-lo usando a função library().

A instalação de pacotes é fundamental, pois permite estender as capacidades do R para atender a necessidades específicas. Essa flexibilidade é uma das razões pelas quais o R é tão poderoso para análise de dados e estatísticas.

Vamos instalar alguns pacotes

install.packages("tidyverse")  
install.packages("agricolae")  
install.packages("ExpDes.pt")

Vamos carregar o pacotes ao ambiente

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 stringr 1.5.0  
## v ggplot2 3.4.2 v tibble 3.2.1  
## v lubridate 1.9.2 v tidyr 1.3.0  
## v purrr 1.0.1   
## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(agricolae)  
library(ExpDes.pt)

##   
## Attaching package: 'ExpDes.pt'  
##   
## The following objects are masked from 'package:agricolae':  
##   
## lastC, order.group, tapply.stat

## 2. Conceitos Básicos do R

### 2.1 Tipos de objetos (vetores, matrizes, listas, data frames).

R oferece diversos tipos de objetos para armazenar e manipular dados. Aqui estão alguns dos tipos mais comuns:

#### **Vetores**:

Os vetores são sequências unidimensionais de elementos do mesmo tipo (por exemplo, números inteiros, números reais, caracteres). Eles podem ser criados usando a função c() (combine) e são a estrutura de dados fundamental em R.

OBS: a atribuição de valores a um objeto no R será sempre feita pelo operador de atribuição <-.

x <- 1  
is.vector(x)

## [1] TRUE

length(x)

## [1] 1

Observe que um escalar para o R é um vetor com dimensão de elementos.

y <- c(pi, 2, 5)  
y

## [1] 3.141593 2.000000 5.000000

Vamos dar um exemplo de vetor com caracteres:

nome <- c("Arthur", "Beatriz", "Carlos", "Davi")  
nome

## [1] "Arthur" "Beatriz" "Carlos" "Davi"

Agora observe o retorno das funções

is.character(nome)

## [1] TRUE

mode(nome)

## [1] "character"

class(nome)

## [1] "character"

#### **Matrizes**:

As matrizes são estruturas bidimensionais que consistem em elementos organizados em linhas e colunas. Todos os elementos de uma matriz devem ter o mesmo tipo de dado. Você pode criar matrizes com a função matrix().

fornecido o vetor x, contendo 9 elementos, podemos criar a matriz A, com 3 linhas e 3 colunas. O preenchimento da matriz é controlado pelo argumento byrow da função matrix(), sendo por linhas (byrow = TRUE) ou por colunas (byrow = FALSE).

x <- c(1,2,3,4,5,6,7,8,10)  
A <- matrix(x, ncol=3, byrow = FALSE)  
A

## [,1] [,2] [,3]  
## [1,] 1 4 7  
## [2,] 2 5 8  
## [3,] 3 6 10

A determinante de A é dada por:

det(A)

## [1] -3

Podemos trasnpor a matriz A com a função t().

t(A)

## [,1] [,2] [,3]  
## [1,] 1 2 3  
## [2,] 4 5 6  
## [3,] 7 8 10

Podemos inverter a matriz A com a função solve().

solve(A)

## [,1] [,2] [,3]  
## [1,] -0.6666667 -0.6666667 1  
## [2,] -1.3333333 3.6666667 -2  
## [3,] 1.0000000 -2.0000000 1

Podemos multiplicar a matriz A pela sua inversa e termos a matriz identidade.

round(solve(A) %\*% A, 2)

## [,1] [,2] [,3]  
## [1,] 1 0 0  
## [2,] 0 1 0  
## [3,] 0 0 1

OBS: A função round() foi utilizada para aproximação das casas decimais dos valores.

#### **Listas**:

Listas são estruturas de dados que podem conter elementos de tipos diferentes, incluindo vetores, matrizes, listas e objetos individuais. Listas são criadas com a função list().

Vamos criar uma lista com os vetores nome e x e a matriz A.

minha\_lista <- list(  
 nomes = nome,  
 elementos = x,  
 matrix = A  
)  
minha\_lista

## $nomes  
## [1] "Arthur" "Beatriz" "Carlos" "Davi"   
##   
## $elementos  
## [1] 1 2 3 4 5 6 7 8 10  
##   
## $matrix  
## [,1] [,2] [,3]  
## [1,] 1 4 7  
## [2,] 2 5 8  
## [3,] 3 6 10

#### **Data Frames**:

Os data frames são semelhantes a matrizes, mas têm uma flexibilidade adicional. Eles são estruturas bidimensionais onde as colunas podem conter diferentes tipos de dados. Data frames são frequentemente usados para representar dados tabulares, como planilhas. Você pode criar data frames usando a função data.frame().

Esses tipos de objetos são fundamentais para armazenar e organizar dados em R. Eles desempenham um papel essencial na manipulação e análise de dados, permitindo que você realize operações estatísticas, visualizações e experimentações de maneira eficaz. Cada tipo de objeto tem suas próprias propriedades e métodos específicos para operações e análises.

Vamos crias um Data frame com nome dos alunos, data de nascimentos, e notas das provas 1 e 2.

meu\_df <- data.frame(  
 nome = nome,  
 data\_nascimento = c("25/05/2003","26/08/2006","27/07/2003","28/08/2003"),  
 nota\_p1 = c(5,3,8,10),  
 nota\_p2 = c(2,1,3,6)  
)  
meu\_df

## nome data\_nascimento nota\_p1 nota\_p2  
## 1 Arthur 25/05/2003 5 2  
## 2 Beatriz 26/08/2006 3 1  
## 3 Carlos 27/07/2003 8 3  
## 4 Davi 28/08/2003 10 6

### 2.2 Operações básicas com objetos.

O acesso aos elementos da lista pode ser realizado por meio da indexação de sua posição entre colchetes, declarado após o nome do objeto. Tal indexação funciona para todos os elementos até aqui criados.

O terceiro nome é:

nome[3]

## [1] "Carlos"

O primeiro e o quarto nomes são:

nome[c(1,4)]

## [1] "Arthur" "Davi"

A terceira coluna de A é:

A[,3]

## [1] 7 8 10

A primeira e terceira colunas de A são:

A[,c(1,3)]

## [,1] [,2]  
## [1,] 1 7  
## [2,] 2 8  
## [3,] 3 10

As segunda e terceira linhas de A são:

A[c(2,3),]

## [,1] [,2] [,3]  
## [1,] 2 5 8  
## [2,] 3 6 10

O primeiro elemento da lista é:

minha\_lista[1]

## $nomes  
## [1] "Arthur" "Beatriz" "Carlos" "Davi"

Retirar a segunda coluna da matriz da lista:

minha\_lista[[3]][,2]

## [1] 4 5 6

Agora, podemos aproveitar o operador de pipe (%>%) para efetuar alterações no Data Frame que construímos. Esse operador é uma implementação do pacote magrittr, que é carregado no ambiente R quando utilizamos o meta-pacote tidyverse. A seguir, serão apresentadas algumas funções para a manipulação de Data Frames. A grande vantagem desse método de manipulação é que o objeto original a ser trabalhado (meu\_df) é sempre preservado, tornando-o ideal para transformações sem destruir os dados originais.

Vamos criar a média das provas, nota final dos alunos

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2)

## nome data\_nascimento nota\_p1 nota\_p2 nota\_final  
## 1 Arthur 25/05/2003 5 2 3.5  
## 2 Beatriz 26/08/2006 3 1 2.0  
## 3 Carlos 27/07/2003 8 3 5.5  
## 4 Davi 28/08/2003 10 6 8.0

Vamos criar a média das provas, nota final dos alunos, e ordenar da maior para a menor média

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2) %>%   
 arrange(desc(nota\_final))

## nome data\_nascimento nota\_p1 nota\_p2 nota\_final  
## 1 Davi 28/08/2003 10 6 8.0  
## 2 Carlos 27/07/2003 8 3 5.5  
## 3 Arthur 25/05/2003 5 2 3.5  
## 4 Beatriz 26/08/2006 3 1 2.0

Vamos criar a média das provas, nota final dos alunos, e ordenar da maior para a menor média, e selecionar somente as colunas nome e nota\_final.

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2) %>%   
 arrange(desc(nota\_final)) %>%   
 select(nome, nota\_final)

## nome nota\_final  
## 1 Davi 8.0  
## 2 Carlos 5.5  
## 3 Arthur 3.5  
## 4 Beatriz 2.0

Vamos criar a média das provas, nota final dos alunos, e ordenar da maior para a menor média, e selecionar somente as colunas nome e nota\_final e adicionar a situação do aluno, se a nota\_final por maior ou igual a 5 ele estará aprovado, caso contrário, estará reprovado.

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2) %>%   
 arrange(desc(nota\_final)) %>%   
 select(nome, nota\_final) %>%   
 mutate(situação = ifelse(nota\_final >=5, "Aprovado","Reprovado"))

## nome nota\_final situação  
## 1 Davi 8.0 Aprovado  
## 2 Carlos 5.5 Aprovado  
## 3 Arthur 3.5 Reprovado  
## 4 Beatriz 2.0 Reprovado

Qual a média geral das notas?:

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2) %>%   
 arrange(desc(nota\_final)) %>%   
 select(nome, nota\_final) %>%   
 mutate(situação = ifelse(nota\_final >=5, "Aprovado","Reprovado")) %>%   
 summarise(media\_geral = mean(nota\_final))

## media\_geral  
## 1 4.75

Qual a média geral das notas por situação?:

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2) %>%   
 arrange(desc(nota\_final)) %>%   
 select(nome, nota\_final) %>%   
 mutate(situação = ifelse(nota\_final >=5, "Aprovado","Reprovado")) %>%   
 group\_by(situação) %>%   
 summarise(media\_geral = mean(nota\_final))

## # A tibble: 2 x 2  
## situação media\_geral  
## <chr> <dbl>  
## 1 Aprovado 6.75  
## 2 Reprovado 2.75

Quais os desvios de nota\_final em relação à média geral?:

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2) %>%   
 arrange(desc(nota\_final)) %>%   
 select(nome, nota\_final) %>%   
 mutate(situação = ifelse(nota\_final >=5, "Aprovado","Reprovado")) %>%   
 mutate(desvio = nota\_final - mean(nota\_final))

## nome nota\_final situação desvio  
## 1 Davi 8.0 Aprovado 3.25  
## 2 Carlos 5.5 Aprovado 0.75  
## 3 Arthur 3.5 Reprovado -1.25  
## 4 Beatriz 2.0 Reprovado -2.75

Quais os desvios de nota\_final em relação à média de nota\_final por grupo de situação?:

meu\_df %>%   
 mutate(nota\_final = 0.5\* nota\_p1 + 0.5\*nota\_p2) %>%   
 arrange(desc(nota\_final)) %>%   
 select(nome, nota\_final) %>%   
 mutate(situação = ifelse(nota\_final >=5, "Aprovado","Reprovado")) %>%   
 group\_by(situação) %>%   
 mutate(desvio = nota\_final - mean(nota\_final))

## # A tibble: 4 x 4  
## # Groups: situação [2]  
## nome nota\_final situação desvio  
## <chr> <dbl> <chr> <dbl>  
## 1 Davi 8 Aprovado 1.25  
## 2 Carlos 5.5 Aprovado -1.25  
## 3 Arthur 3.5 Reprovado 0.75  
## 4 Beatriz 2 Reprovado -0.75

### 2.3 Funções e operadores.

#### Funções Básicas:

Operações Matemáticas:

Soma:

Subtração: 5 - 1

Multiplicação: 4 \* 7

Divisão: 11 / 2

Divisão inteira: 11 %/% 3

Resto da divisão inteira: 11 %% 3

#### Funções Matemáticas:

Raiz quadrada: sqrt(25)

Potência: 2^3 ou 2\*\*

Valor absoluto: abs(-10)

Funções trigonométricas seno: sin(30\*pi/180)

Função arco-seno: asin(0.5)\*180/pi

Função exponencial: exp(1)

Função logarítmo natual: log(2.718282)

Função log na base 10: log10(100)

Função log definição da base: log(1024,2)

#### Operadores Lógicos e relacionais:

Igualdade:

> 3 == 3 (verdadeiro)  
  
> 3 == 4 (falso)  
  
> 3 != 4 (verdadeiro)

Maior e Menor (ou igual):

> 5 > 3 (verdadeiro)  
  
 > 2 <= 1 (falso)

Operadores Lógicos Compostos:

> TRUE && FALSE (AND lógico, falso)  
  
> TRUE || FALSE (OR lógico, verdadeiro)

#### Criando uma função no R

Como exemplo vamos criar uma funão que calcule o coeficiente de variação que será dado pela relação percentual do desvio-padrão amostral e a média amostral:

Criando a função no R.

meu\_cv <- function(x){  
 media <- mean(x)  
 desv\_pad <- sd(x)  
 valor\_cv <- 100\*desv\_pad/media  
 return(valor\_cv)  
}

Agora podemos chamar a função.

y <- c(8,5.5,3.5,2)  
meu\_cv(y)

## [1] 54.69634

## 3. Manipulação de Dados no R

### 3.1 Importação e exportação de dados (leitura de CSV, Excel, etc.).

Suponha que você tenha um arquivo de texto chamado “dados.txt” com o seguinte conteúdo:

Nome Idade João 30 Maria 28 Pedro 35 Ana 25

Você pode usar a função read.table() para importar esses dados em R da seguinte forma:

# Especifique o caminho para o arquivo de texto  
caminho\_do\_arquivo <- "caminho/para/seu/arquivo/dados.txt"  
  
# Use a função read.table() para importar os dados  
seu\_dataframe <- read.table(caminho\_do\_arquivo, header = TRUE)

Para importar dados de um arquivo CSV (Comma-Separated Values), você pode usar a função read.csv():

meu\_dataframe <- read.csv("seu\_arquivo.csv")

Para importar dados de uma planilha Excel, você pode usar a biblioteca readxl e a função read\_excel(), o argumento sheet especifica o nome da planilha que se quer importar:

library(readxl)  
meu\_dataframe <- read\_excel("seu\_arquivo.xlsx",  
 sheet = "Plan1")

Leitura de Outros Formatos:

O R suporta a importação de dados de muitos outros formatos, como arquivos JSON, SQL, HDF5, e mais. Você pode usar pacotes específicos para cada formato.

### 3.1 Limpeza e transformação de dados.

Vamos simular um conjunto de dados:

meus\_dados <- data.frame(  
 FCO2 = sort(rnorm(100, 4, 0.95)),  
 Temp = rnorm(100, 24,2),  
 umidade = sort(rnorm(100, 15,5),decreasing = TRUE)  
)

Padronização dos nomes das variáveis:

meus\_dados %>%   
 janitor::clean\_names() %>%   
 head()

## fco2 temp umidade  
## 1 1.425698 23.95987 25.70015  
## 2 1.449699 26.93935 25.42715  
## 3 1.567257 22.74233 25.30713  
## 4 1.895866 23.93879 23.13553  
## 5 2.225544 21.09140 22.51078  
## 6 2.526564 27.24548 22.15193

Renomeando a variável temp para ts e a variável umidade para us.

meus\_dados\_pad <- meus\_dados %>%   
 janitor::clean\_names() %>%   
 rename(ts = temp, us = umidade)

Valores perdidos poderão ser descartados da base de dados pela função drop\_na do pacote tidyr.

meus\_dados\_pad %>%   
 drop\_na() %>%   
 head()

## fco2 ts us  
## 1 1.425698 23.95987 25.70015  
## 2 1.449699 26.93935 25.42715  
## 3 1.567257 22.74233 25.30713  
## 4 1.895866 23.93879 23.13553  
## 5 2.225544 21.09140 22.51078  
## 6 2.526564 27.24548 22.15193

Vamos transformar os dados de fco2, passando-os para a escala logarítmica:

meus\_dados\_pad %>%   
 drop\_na() %>%   
 mutate(fco2\_log = log10(fco2)) %>%   
 head()

## fco2 ts us fco2\_log  
## 1 1.425698 23.95987 25.70015 0.1540274  
## 2 1.449699 26.93935 25.42715 0.1612779  
## 3 1.567257 22.74233 25.30713 0.1951402  
## 4 1.895866 23.93879 23.13553 0.2778076  
## 5 2.225544 21.09140 22.51078 0.3474361  
## 6 2.526564 27.24548 22.15193 0.4025303

Criar as categorais para emissão em “baixa”, “média” e “alta”, por meio da função cut. Mude a posição das colunas fco2\_log e categoria para logo depois da coluna fco2.

meus\_dados\_pad <- meus\_dados\_pad %>%   
 drop\_na() %>%   
 mutate(fco2\_log = log10(fco2),  
 categoria = cut(fco2,  
 breaks = 3,  
 labels = c("baixa","média","alta")  
 )  
 ) %>%   
 relocate(fco2, fco2\_log, categoria)   
meus\_dados\_pad %>% head

## fco2 fco2\_log categoria ts us  
## 1 1.425698 0.1540274 baixa 23.95987 25.70015  
## 2 1.449699 0.1612779 baixa 26.93935 25.42715  
## 3 1.567257 0.1951402 baixa 22.74233 25.30713  
## 4 1.895866 0.2778076 baixa 23.93879 23.13553  
## 5 2.225544 0.3474361 baixa 21.09140 22.51078  
## 6 2.526564 0.4025303 baixa 27.24548 22.15193

### 3.3 Resumo estatístico dos dados.

Vamos agora criar um resumo estatístico simples para os dados

Emissão de CO2 do solo

meus\_dados\_pad %>%   
 pull(fco2) %>%   
 summary()

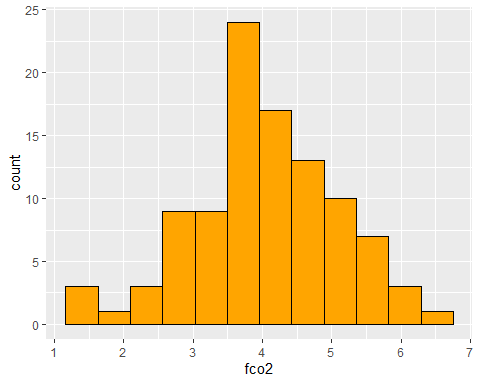
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.426 3.513 4.051 4.027 4.685 6.549

## 4. Visualização de Dados

### 4.1 Gráficos básicos (histogramas, boxplot, gráficos de barras, gráficos de dispersão).

Histograma da emissão de CO2 do solo

meus\_dados\_pad %>%   
 ggplot(aes(x=fco2)) +   
 geom\_histogram(bins = 12, color="black",fill="orange")



Boxplot da emissão de CO2 do solo por categoria

meus\_dados\_pad %>%   
 ggplot(aes(x= categoria, y=fco2, fill = categoria)) +   
 geom\_boxplot()

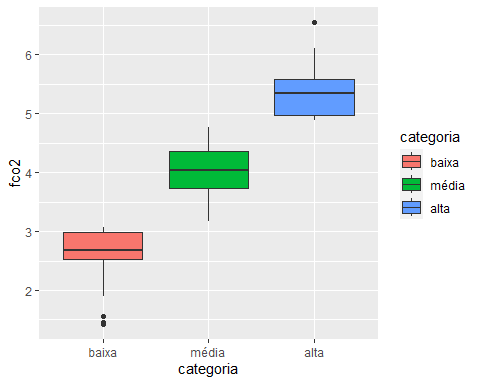


Gráfico de colunas da emissão de CO2 do solo

meus\_dados\_pad %>%   
 group\_by(categoria) %>%   
 summarise(media = mean(fco2)) %>%   
 ggplot(aes(x=categoria, y=media, fill=categoria)) +  
 geom\_col()

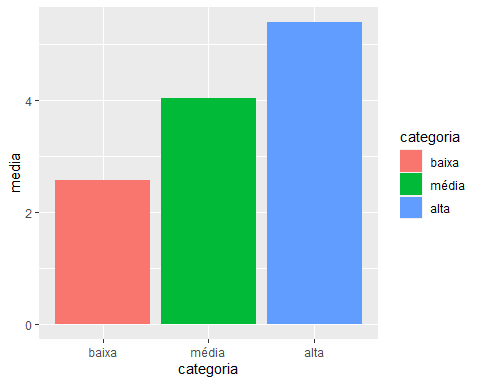
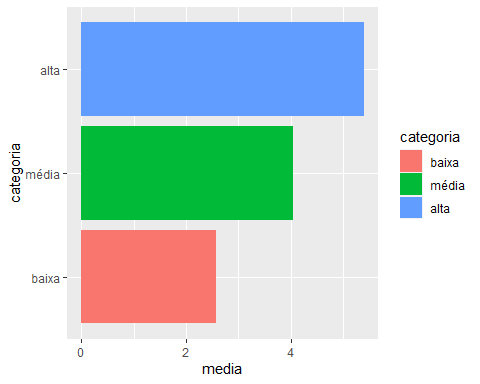


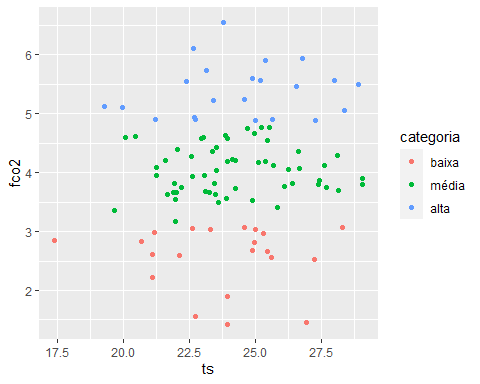
Gráfico de barras da emissão de CO2 do solo

meus\_dados\_pad %>%   
 group\_by(categoria) %>%   
 summarise(media = mean(fco2)) %>%   
 ggplot(aes(x=categoria, y=media, fill=categoria)) +  
 geom\_col() +  
 coord\_flip()

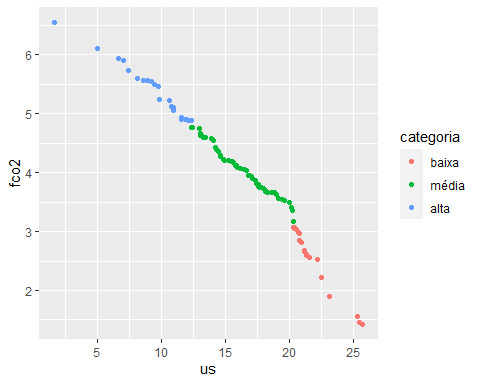


Plotar os dados de emissão de co2 em função da temperatura do solo, mapeando as categorias

meus\_dados\_pad %>%   
 ggplot(aes(x=ts, y=fco2, color=categoria)) +  
 geom\_point()

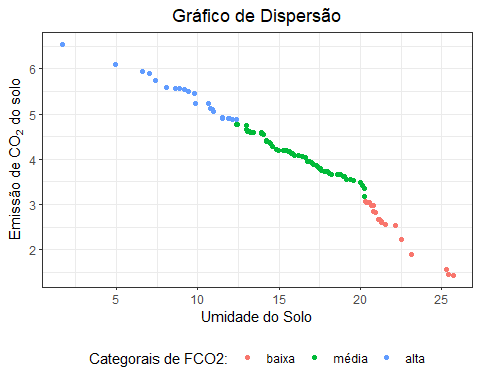
 Plotar os dados de emissão de co2 em função da umidade do solo, mapeando as categorias

meus\_dados\_pad %>%   
 ggplot(aes(x=us, y=fco2, color=categoria)) +  
 geom\_point()



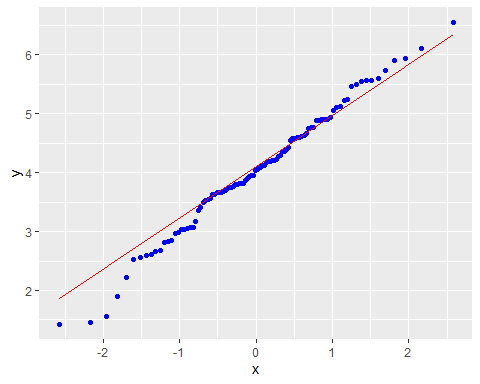
### 4.2 Personalização de gráficos.

meus\_dados\_pad %>%   
 ggplot(aes(x=us, y=fco2, color=categoria)) +  
 geom\_point() +  
 theme\_bw() +  
 labs(title="Gráfico de Dispersão",  
 x= "Umidade do Solo",  
 y=expression(paste("Emissão de ", CO[2]," do solo")),  
 color = "Categorais de FCO2:"  
 ) +   
 theme(legend.position = "bottom",  
 plot.title = element\_text(hjust = 0.5))



### 4.3 Verificação de normalidade (teste de Shapiro-Wilk, gráfico Q-Q).

meus\_dados\_pad %>%   
 ggplot(aes(sample = fco2)) +   
 stat\_qq(color="blue") +  
 stat\_qq\_line(color = "red")

 Teste de Normalidade

library(nortest)  
meus\_dados\_pad %>% pull(fco2) %>% shapiro.test()

##   
## Shapiro-Wilk normality test  
##   
## data: .  
## W = 0.99025, p-value = 0.685

meus\_dados\_pad %>% pull(fco2) %>% lillie.test()

##   
## Lilliefors (Kolmogorov-Smirnov) normality test  
##   
## data: .  
## D = 0.068995, p-value = 0.2859

meus\_dados\_pad %>% pull(fco2) %>% ad.test()

##   
## Anderson-Darling normality test  
##   
## data: .  
## A = 0.28426, p-value = 0.6236

meus\_dados\_pad %>% pull(fco2) %>% cvm.test()

##   
## Cramer-von Mises normality test  
##   
## data: .  
## W = 0.045511, p-value = 0.5772

## 5. Estatística Descritiva

### 5.1 Medidas de tendência central (média, mediana, moda).

### 5.2 Medidas de dispersão (desvio padrão, variância, intervalos).

### 5.3 Visualização de estatísticas descritivas.

## 6. Ajuste de Regressão

### 6.1 Regressão linear simples e múltipla.

### 6.2 Análise de variância (ANOVA).

### 6.3 Interpretação de resultados.

### 6.4 Gráficos de regressão.

## 7. Delineamentos Experimentais

### 7.1 Delineamento Inteiramente Casualizados

### 7.2 Teste de comparação de médias

### 7.3 Delineamento em Blocos Casualizados

### 7.4 Delineamento Quadrado Latino

### 7.5 Desdobamento de Graus de Liberdade

### 7.6 Experimentos Fatoriais

### 7.7 Experimentos em Parcelas Subdivididas

### 7.8 Análise de Regressão