UNICEUB - FATECS - CIÊNCIA DE DADOS

INTRODUÇÃO À LINGUAGEM R

08 novembro, 2023

1 Trabalho Principal da Disciplina

Professor: Abner Santos Belém.

Membros do Grupo:

- Bruno Abreu Cardozo RA:22306260
- Leonardo de Lima Amaral RA: 22303479
- Luis Gustavo Silva Verissimo RA:22303479
- Marcelo Saraiva Cavalcanti RA:22300690
- Matheus Graça Lira RA:22307010
- Matheus Nery RA:22309707
- Rafael Mascarenhas Brown de Andrade RA:22304454
- Tiago de Oliveira Teixeira RA:22303612

2 Bibliografia e fontes de consulta:

- Apostilas e anotações de aula do Prof. Abner Santos Belém.
- Alcoforado, Luciane Ferreira. Utilizando a Linguagem R: Conceitos, Manipulação, Visualização, Modelagem e Visualização de Relatórios. Rio de Janeiro: Alta Books, 2021.
- Morettin, Pedro Alberto. Bussad, Wilton O..Estatística Básica. 9ª Ed. São Paulo: Saraiva, 2017.
- Wickham, Hadley. Grolemund, Garret. R para Data Science: Importe, Arrume, Transforme, Visualize e Modele Dados. Rio de Janeiro: Alta Books, 2019.
- Amaral, Fernando. Introdução à Ciência de Dados: Mineração de dados e Big Data. Rio de Janeiro: Alta Books, 2016.
- OpenAI. “GPT-3”. https://openai.com (https://openai.com).
- Kaggle. https://www.kaggle.com/datasets/arnabchaki/popular-video-games-1980-2023 (https://www.kaggle.com/datasets/arnabchaki/popular-video-games-1980-2023)

3 Problema proposto a ser resolvido:

Nós estamos simulando uma situação de vida real uma empresa startup de tecnologia, com seus oito membros fundadores acima, identificaram uma ideia de negócio promissora e pretendem criar e lançar um novo game.

Iniciamos pesquisas para validar a viabilidade e potencial do mercado para identificar, principalmente, os gêneros ou áreas onde os games já lançados mais fazem sucesso.

O objetivo desta pesquisa é identificar os gêneros ou áreas temáticas presentes nos games já lançados no mercado e verificar o nível de adesão dos diversos usuários a cada um destes gêneros.

Após esta fase inicial, passaremos para a fase de desenvolvimento inicial e lançamento, quando, então, os fundadores começarão a desenvolver um protótipo ou um produto mínimo viável (MVP) para lançar no mercado e obter feedback dos primeiros usuários.

4 Busca pelos dados que tratam do assundo e Metadados:

Após diversas pesquisas nas fontes de bases de dados na internet encontramos a base de dados cujos metadados são descritos abaixo:

1. Título: Videogames populares de 1980 a 2023. Detalhes sobre os videogames mais bem avaliados de 1980 a 2023.
2. Fonte: https://www.kaggle.com/datasets/arnabchaki/popular-video-games-1980-2023 (https://www.kaggle.com/datasets/arnabchaki/popular-video-games-1980-2023)
3. Tamanho: 1512 observações (linhas) com 14 variáveis (colunas ou campos)
4. Proveniência – fonte: os dados foram coletados de um site de registro/gravação de jogos chamado backloggd < https://www.backloggd.com/ (https://www.backloggd.com/) >.
    i. Backloggd é um site de coleção de videogames misturado com elementos sociais para focar em dar vida ao seu perfil de jogo. Crie uma conta gratuita para começar a registrar os jogos que você jogou e, em seguida, avaliar e comentar à medida que avança! Entre em detalhes com plataformas de registro, tempo de jogo e até mesmo um diário para acompanhar seu progresso diário no jogo com jogadas. Tudo é adaptado ao quanto você deseja registrar, para que seu perfil se adapte a você. Além disso, você pode criar listas de jogos, tornar-se amigo de outros usuários, acompanhar suas atividades e muito mais!
5. Metodologia de coleta: ferramentas simples de coleta de dados foram usadas, como Beautiful Soup.
6. Data da coleta: estimamos que a coleta foi feita cerca de 7 meses atrás, em fevereiro ou março de 2023 (????).
7. Tipo: data.frame
8. Variáveis: tipo - descrição sumária:
    i. **X** : int – índice numérico.
    ii. **Title** : chr - título do game.
    iii. **Release.Date** : chr – data de lançamento da primeria versão do game.
    iv. **Team** : chr – empresa que desenvolveu o game.
    v. **Rating** : num – classificação média, avaliações feitas do game, notas recebidas pelos games, quanto maior melhor.
    vi. **Times.Listed** : chr – número de usuários que listaram este game.
    vii. **Number.of.Reviews** : chr - número de avaliações recebidas dos usuários.
    viii. **Genres** : chr - todos os gêneros pertencentes a um jogo específico.
    ix. **Summary** : chr – sumário fornecido pela empresa que desenvolveu o game.
    x. **Reviews** : chr – cometários dos usuários.
    xi. **Plays** : chr - número de usuários que já jogaram o jogo antes (????).
    xii. **Playing** : chr - número de usuários atuais que estão jogando.
    xiii. **Backlogs** : chr - número de usuários que têm acesso, mas ainda não iniciaram o jogo.
    xiv. **Wishlist** : chr - lista de desejos: número de usuários que desejam jogar o jogo

5 Preparando o ambiente de trabalho do RStudio:

> Observação: Breves explicações dos comandos mais importantes ou difíceis do R aparecerão em caixas de texto como esta.

5.1 Para garantir que a codificação correta (UTF-8) seja utilizada nos scripts e documentos R:

Isso define a codificação do ambiente para UTF-8, o que deve garantir que os caracteres acentuados e especiais sejam interpretados corretamente.
Adicionamos esta linha no início do nosso script ou documento RMarkdown antes de qualquer outro código para garantir que a codificação correta seja aplicada corretamente.

```
Sys.setlocale(category = "LC_ALL", locale = "en_US.UTF-8")
```

```
## [1] "LC_COLLATE=en_US.UTF-8;LC_CTYPE=en_US.UTF-8;LC_MONETARY=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=en_US.UTF-8"
```

5.2 Limpando as variáveis do ambiente para previnir a ocorrência de erros:

> **rm(list = ls())**: Este comando é usado em R para remover todos os objetos (variáveis, funções, etc.) atualmente armazenados na memória. Quando usado com `list = ls()` , ele remove todos os objetos listados no ambiente de trabalho atual, liberando memória e limpando o espaço de trabalho. É frequentemente usado para limpar o ambiente antes de executar um novo script ou para evitar conflitos entre objetos com nomes semelhantes.

```
rm(list = ls())
```

5.3 Definindo o repositório de pacotes do CRAN:

options(repos = "https://cran.r-project.org (https://cran.r-project.org)"): Este comando é usado para definir a localização do repositório CRAN (Comprehensive R Archive Network) de onde os pacotes R são baixados e instalados. Aqui, você está configurando o repositório padrão para ser "https://cran.r-project.org (https://cran.r-project.org)", que é o endereço padrão para o repositório CRAN. Isso garante que, ao instalar novos pacotes, o R procure no repositório especificado para encontrar e baixar os pacotes necessários.

```
options(repos = "https://cran.r-project.org")
```

5.4 Instalando os pacotes básicos:

O objetivo dessas linhas é verificar se o pacote "reshape2" está instalado e, se não estiver, instalá-lo automaticamente para garantir que o script tenha acesso às funcionalidades fornecidas por esse pacote.

1. `if(!require("reshape2")){}` : Esta linha verifica se o pacote "reshape2" está disponível. A função `require` tenta carregar o pacote. O ponto de exclamação `!` antes da chamada `require` é usado para verificar se o pacote não está presente. Se o pacote não estiver disponível, o código entre chaves `{}` será executado.
2. `install.packages('reshape2')` : Se o pacote "reshape2" não estiver disponível, esta linha instalará o pacote a partir do repositório CRAN (Comprehensive R Archive Network). A função `install.packages` é usada para instalar pacotes R. Ao especificar o nome do pacote entre aspas simples, você indica ao R qual pacote deve ser instalado.

```
if(!require("reshape2")){
  install.packages('reshape2')
}

if(!require("ggplot2")){
  install.packages('ggplot2')
}

if(!require("stringr")){
  install.packages('stringr')
}

if(!require("tidyverse")){
  install.packages('tidyverse')
}

if(!require("reshape2")){
  install.packages('reshape2')
}

if (!require("psych")) {
  install.packages('psych')
}


if (!require("lubridate")) {
  install.packages("lubridate")
}
```

5.5 Carregando os pacotes básicos:

1. **dplyr**: É uma ferramenta popular para manipulação de dados em R. Ele fornece uma gramática consistente para a manipulação de dados, permitindo que você filtre, transforme e resuma conjuntos de dados.
2. **car**: É um pacote utilizado para fornecer funções para o diagnóstico estatístico e visualização de modelos lineares.
3. **rstatix**: Fornece uma interface simples para executar análises estatísticas comuns e criar gráficos para explorar esses resultados.

4. **emmeans**: Este pacote é útil para realizar comparações de médias e estimativas dos efeitos marginais em modelos lineares.
5. **ggplot2**: É uma das bibliotecas mais populares para criar gráficos e visualizações em R. Ele segue a abordagem de "grammar of graphics" e é altamente personalizável.
6. **knitr**: É uma ferramenta de conversão que converte documentos RMarkdown em vários formatos, incluindo HTML e PDF.
7. **kableExtra**: É uma extensão do pacote `knitr e kable` que permite a personalização de tabelas em documentos RMarkdown e LaTeX.
8. **htmltools**: Fornece várias ferramentas para trabalhar com HTML em R, permitindo a criação e modificação de conteúdo HTML de forma programática.
9. **reshape2**: É útil para remodelar, fundir e dividir conjuntos de dados em formatos mais adequados para análise e visualização.
10. **stringr**: Fornece funções simples e eficientes para manipulação de strings em R.
11. **tidyverse**: É uma coleção de pacotes R projetados para ciência de dados. Ele inclui várias bibliotecas, incluindo dplyr e ggplot2, que são amplamente usadas para manipulação e visualização de dados.
12. **psych**: É uma biblioteca para análise psicométrica e geração de relatórios de estatísticas descritivas e inferenciais.
13. **lubridate**: É usado para trabalhar com datas e horários em R, facilitando a manipulação e cálculos de datas.

```
library(dplyr)
library(car)
library(rstatix)
library(emmeans)
library(ggplot2)
library(knitr)
library(kableExtra)
library(htmltools)
library(reshape2)
library(stringr)
library(tidyverse)
library(psych)
library(lubridate)
```

6 Lendo a base de dados:

```
data <- read.csv("G:\\Meu Drive\\RLanguage\\TrabGrupoR\\games.csv")
```

6.1 Mostrando a estrutura do dataset:

```
str(data)
```

```
## 'data.frame':    1512 obs. of  14 variables:
##  $ X               : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Title           : chr  "Elden Ring" "Hades" "The Legend of Zelda: Breath of the Wild" "Undertale" ...
##  $ Release.Date    : chr  "Feb 25, 2022" "Dec 10, 2019" "Mar 03, 2017" "Sep 15, 2015" ...
##  $ Team            : chr  "['Bandai Namco Entertainment', 'FromSoftware']" "['Supergiant Games']" "['Nintend
o', 'Nintendo EPD Production Group No. 3']" "['tobyfox', '8-4']" ...
##  $ Rating          : num  4.5 4.3 4.4 4.2 4.4 4.3 4.2 4.3 3 4.3 ...
##  $ Times.Listed    : chr  "3.9K" "2.9K" "4.3K" "3.5K" ...
##  $ Number.of.Reviews: chr  "3.9K" "2.9K" "4.3K" "3.5K" ...
##  $ Genres          : chr  "['Adventure', 'RPG']" "['Adventure', 'Brawler', 'Indie', 'RPG']" "['Adventure', 'RP
G']" "['Adventure', 'Indie', 'RPG', 'Turn Based Strategy']" ...
##  $ Summary         : chr  "Elden Ring is a fantasy, action and open world game with RPG elements such as stat
s, weapons and spells. Rise, "| __truncated__ "A rogue-lite hack and slash dungeon crawler in which Zagreus, son o
f Hades the Greek god of the dead, attempts "| __truncated__ "The Legend of Zelda: Breath of the Wild is the first
3D open-world game in the Zelda series. Link can travel an"| __truncated__ "A small child falls into the Undergrou
nd, where monsters have long been banished by humans and are hunting ever"| __truncated__ ...
##  $ Reviews         : chr  "[\"The first playthrough of elden ring is one of the best eperiences gaming can off
er you but after youve explo"| __truncated__ "['convinced this is a roguelike for people who do not like the genr
e. The art is technically good but the aesth"| __truncated__ "['This game is the game (that is not CS:GO) that I h
ave played the most ever. I have played this game for 400 h"| __truncated__ "['soundtrack is tied for #1 with nier
automata.  a super charming story and characters which have become iconic"| __truncated__ ...
##  $ Plays           : chr  "17K" "21K" "30K" "28K" ...
##  $ Playing         : chr  "3.8K" "3.2K" "2.5K" "679" ...
##  $ Backlogs        : chr  "4.6K" "6.3K" "5K" "4.9K" ...
##  $ Wishlist        : chr  "4.8K" "3.6K" "2.6K" "1.8K" ...
```

6.2 Mostrando os nomes das colunas do dataset:

```
names(data)
```

```
##  [1] "X"                "Title"            "Release.Date"
##  [4] "Team"             "Rating"           "Times.Listed"
##  [7] "Number.of.Reviews" "Genres"           "Summary"
## [10] "Reviews"          "Plays"            "Playing"
## [13] "Backlogs"         "Wishlist"
```

7 Limpando a base de dados:

7.1 Removendo as colunas com dados que não nos interessam:

Vamos eliminar as colunas “Summary” e “Reviews”, que contêm apenas textos, e a coluna “X”, que é um índice desnecessário.

O código é usado em R para criar um novo conjunto de dados (`novo_data`) que é uma versão do conjunto de dados original (`data`) com determinadas colunas removidas:

1. `subset(data, select = -c(X, Summary, Reviews))` : Este trecho de código utiliza a função `subset` para selecionar um subconjunto do conjunto de dados `data` . A parte `select = -c(X, Summary, Reviews)` indica que as colunas com os nomes “X”, “Summary” e “Reviews” devem ser removidas do subconjunto. O operador `-` antes da função `c()` indica que essas colunas devem ser excluídas.
2. `novo_data <- subset(...)` : A nova versão do conjunto de dados, que exclui as colunas especificadas, é armazenada na variável `novo_data` . Isso cria um novo conjunto de dados que contém todas as colunas do conjunto de dados original, exceto aquelas especificadas para exclusão.

O comando cria uma versão modificada do conjunto de dados original, `data` , onde as colunas “X”, “Summary” e “Reviews” são removidas, e essa nova versão é atribuída à variável `novo_data` .

```
novo_data <- subset(data, select = -c(X, Summary, Reviews))
```

7.2 Removendo as linhas com dados que não nos interessam:

As seguintes linhas serão removidas:

- 645, 650, 1253 → apresentam valores “releases on TBD” para as datas;
- 714, 1310 e 1476 → apresentam valores “[]” para os gêneros.

O código é usado em R para remover linhas específicas de um conjunto de dados existente, representado por `novo_data`:

1. `novo_data[-c(645, 650, 714, 1253, 1310, 1476), ]`: Esta expressão seleciona um subconjunto de `novo_data` excluindo as linhas com os índices especificados. Os números entre parênteses, separados por vírgulas, representam os índices das linhas que devem ser excluídas.
2. `novo_data <- novo_data[...]`: O resultado do subconjunto, que exclui as linhas especificadas, é armazenado novamente na variável `novo_data`. Isso modifica o conjunto de dados original `novo_data` para conter apenas as linhas que não foram excluídas.

O código remove as linhas dos índices 645, 650, 714, 1253, 1310 e 1476 do conjunto de dados `novo_data` e atualiza o conjunto de dados original para refletir essa exclusão.

```
novo_data <- novo_data[-c(645, 650, 714, 1253, 1310, 1476), ]
```

7.3 Este comando refaz o índice do dataset após a eliminação das linhas acima:

O comando é usado em R para remover os nomes das linhas de um conjunto de dados:

1. `row.names(novo_data)`: Esta parte do código acessa os nomes das linhas do conjunto de dados `novo_data`.
2. `<- NULL`: O operador de atribuição (`<-`) é usado para atribuir um valor a um objeto em R. Neste caso, o valor sendo atribuído é `NULL`, que é um valor especial em R que representa a ausência de valor. Atribuir `NULL` aos nomes das linhas remove esses nomes.

Portanto, o comando remove os nomes das linhas do conjunto de dados `novo_data`, resultando em um conjunto de dados sem identificadores específicos para as linhas. Isso é útil para redefinir os índices das linhas para o padrão numérico.

```
row.names(novo_data) <- NULL
```

7.3.1 Mostrando a estrutura do novo dataset:

```
str(novo_data)
```

```
## 'data.frame':    1506 obs. of  11 variables:
##  $ Title            : chr  "Elden Ring" "Hades" "The Legend of Zelda: Breath of the Wild" "Undertale" ...
##  $ Release.Date     : chr  "Feb 25, 2022" "Dec 10, 2019" "Mar 03, 2017" "Sep 15, 2015" ...
##  $ Team             : chr  "['Bandai Namco Entertainment', 'FromSoftware']" "['Supergiant Games']" "['Nintend
o', 'Nintendo EPD Production Group No. 3']" "['tobyfox', '8-4']" ...
##  $ Rating           : num  4.5 4.3 4.4 4.2 4.4 4.3 4.2 4.3 3 4.3 ...
##  $ Times.Listed     : chr  "3.9K" "2.9K" "4.3K" "3.5K" ...
##  $ Number.of.Reviews: chr  "3.9K" "2.9K" "4.3K" "3.5K" ...
##  $ Genres           : chr  "['Adventure', 'RPG']" "['Adventure', 'Brawler', 'Indie', 'RPG']" "['Adventure', 'RP
G']" "['Adventure', 'Indie', 'RPG', 'Turn Based Strategy']" ...
##  $ Plays            : chr  "17K" "21K" "30K" "28K" ...
##  $ Playing          : chr  "3.8K" "3.2K" "2.5K" "679" ...
##  $ Backlogs         : chr  "4.6K" "6.3K" "5K" "4.9K" ...
##  $ Wishlist         : chr  "4.8K" "3.6K" "2.6K" "1.8K" ...
```

7.3.2 Verificando se existem valores ausentes (NA) nas colunas do dataset:

```
colSums(is.na(novo_data))
```

```
##             Title      Release.Date              Team            Rating
##                 0                 0                 0                11
##      Times.Listed Number.of.Reviews            Genres             Plays
##                 0                 0                 0                 0
##           Playing          Backlogs          Wishlist
##                 0                 0                 0
```

7.4 Como encontranos 11 observações com NA na coluna Rating do nosso dataset, resolvemos trocar estas observações pela média, conforme abaixo:

O código é usado para substituir os valores ausentes (NA) na coluna "Rating" do conjunto de dados `novo_data` pela média dos valores não ausentes nessa coluna:

1. `novo_data$Rating` : Esta parte do código acessa a coluna "Rating" no conjunto de dados `novo_data` .
2. `[is.na(novo_data$Rating)]` : Isso verifica se os valores da coluna "Rating" são NA (ausentes) e retorna um vetor lógico indicando quais valores são NA.
3. `<- mean(novo_data$Rating, na.rm = TRUE)` : Isso substitui os valores NA na coluna "Rating" pelo valor médio dos valores não ausentes nessa coluna. A função `mean` calcula a média dos valores, e o argumento `na.rm = TRUE` indica que os valores NA devem ser excluídos do cálculo da média.

Portanto, o código assegura que os valores ausentes na coluna "Rating" sejam substituídos pela média dos valores não ausentes, permitindo que você trate os valores ausentes de maneira eficaz em seus dados.

```
novo_data$Rating[is.na(novo_data$Rating)] <- mean(novo_data$Rating, na.rm = TRUE)
```

7.4.1 Verificando, novamente se os NA foram removidos:

```
colSums(is.na(novo_data))
```

```
##             Title      Release.Date              Team            Rating
##                 0                 0                 0                 0
##      Times.Listed Number.of.Reviews            Genres             Plays
##                 0                 0                 0                 0
##           Playing          Backlogs          Wishlist
##                 0                 0                 0
```

7.5 Trocando os dados do novo dataset que estão no formato "3.5K":

Algumas colunas possuem dados numéricos no formato de texto, como por exemplo "3.5K", que significa 3500. Vamos transformar estes dados retirando a letra "K" e multiplicando o resultado por 1000.

O código é usado para transformar as colunas desejadas em formato numérico, levando em consideração a presença do sufixo "K" que indica milhares:

1. `novo_data$Times.Listed` : Esta parte do código acessa a coluna "Times.Listed" no conjunto de dados `novo_data` .
2. `ifelse(grepl("K", novo_data$Times.Listed), ...)` : Esta função condicional verifica se a coluna "Times.Listed" contém o sufixo "K" usando `grepl` para fazer uma correspondência de padrão.
3. `as.numeric(gsub("K", "", novo_data$Times.Listed)) * 1000` : Se a coluna "Times.Listed" contiver o sufixo "K", a função `gsub` é usada para remover o "K", e `as.numeric` converte a string resultante em um número. Multiplicar por 1000 converte o número para a escala correta de milhares.
4. `as.numeric(novo_data$Times.Listed)` : Se a coluna "Times.Listed" não contiver o sufixo "K", ela é convertida diretamente em um número usando a função `as.numeric` .
5. `novo_data$Times.Listed <- ifelse(...)` : O resultado da função `ifelse` é atribuído de volta à coluna "Times.Listed" no conjunto de dados `novo_data` .

Portanto, o código permite que você transforme os valores na coluna "Times.Listed" em formato numérico, considerando a presença do sufixo "K" e convertendo os valores em milhares, se necessário, para facilitar a análise e o processamento dos dados.

```
# Transformar as colunas desejadas em formato numérico, considerando a presença de "K"
novo_data$Times.Listed <- ifelse(grepl("K", novo_data$Times.Listed),
                                 as.numeric(gsub("K", "", novo_data$Times.Listed)) * 1000,
                                 as.numeric(novo_data$Times.Listed))

novo_data$Number.of.Reviews <- ifelse(grepl("K", novo_data$Number.of.Reviews),
                                      as.numeric(gsub("K", "", novo_data$Number.of.Reviews)) * 1000,
                                      as.numeric(novo_data$Number.of.Reviews))

novo_data$Plays <- ifelse(grepl("K", novo_data$Plays),
                          as.numeric(gsub("K", "", novo_data$Plays)) * 1000,
                          as.numeric(novo_data$Plays))

novo_data$Playing <- ifelse(grepl("K", novo_data$Playing),
                            as.numeric(gsub("K", "", novo_data$Playing)) * 1000,
                            as.numeric(novo_data$Playing))

novo_data$Backlogs <- ifelse(grepl("K", novo_data$Backlogs),
                             as.numeric(gsub("K", "", novo_data$Backlogs)) * 1000,
                             as.numeric(novo_data$Backlogs))

novo_data$Wishlist <- ifelse(grepl("K", novo_data$Wishlist),
                             as.numeric(gsub("K", "", novo_data$Wishlist)) * 1000,
                             as.numeric(novo_data$Wishlist))
```

```
str(novo_data)
```

```
## 'data.frame':    1506 obs. of  11 variables:
##  $ Title           : chr  "Elden Ring" "Hades" "The Legend of Zelda: Breath of the Wild" "Undertale" ...
##  $ Release.Date    : chr  "Feb 25, 2022" "Dec 10, 2019" "Mar 03, 2017" "Sep 15, 2015" ...
##  $ Team            : chr  "['Bandai Namco Entertainment', 'FromSoftware']" "['Supergiant Games']" "['Nintend
o', 'Nintendo EPD Production Group No. 3']" "['tobyfox', '8-4']" ...
##  $ Rating          : num  4.5 4.3 4.4 4.2 4.4 4.3 4.2 4.3 3 4.3 ...
##  $ Times.Listed    : num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Number.of.Reviews: num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Genres          : chr  "['Adventure', 'RPG']" "['Adventure', 'Brawler', 'Indie', 'RPG']" "['Adventure', 'RP
G']" "['Adventure', 'Indie', 'RPG', 'Turn Based Strategy']" ...
##  $ Plays           : num  17000 21000 30000 28000 21000 33000 7200 9200 25000 18000 ...
##  $ Playing         : num  3800 3200 2500 679 2400 1800 1100 759 470 1100 ...
##  $ Backlogs        : num  4600 6300 5000 4900 8300 1100 4500 3400 776 6200 ...
##  $ Wishlist        : num  4800 3600 2600 1800 2300 230 3800 3300 126 3600 ...
```

7.6 Trocando os valores de data que estão no formato caracter para data:

O dataset usa as datas em língua inglesa e nós as queremos no formato aaaa-mm-dd, ou seja transformar “Feb 25, 2022” para “2022-02-25”.

7.6.1 Testando a correção do problema com um dataset de teste:

Criei um novo dataset de teste, só com a coluna de data:

```
new_dataset <- novo_data["Release.Date"]
str(new_dataset)
```

```
## 'data.frame':    1506 obs. of  1 variable:
##  $ Release.Date: chr  "Feb 25, 2022" "Dec 10, 2019" "Mar 03, 2017" "Sep 15, 2015" ...
```

O argumento `format` na função `as.Date()` é usado para especificar o formato das datas que você está tentando converter de `character` para `Date` . Aqui estão alguns dos formatos mais comuns que podem ser usados no R:

- `%d` : Dia do mês como um número decimal (de 01 a 31).
- `%m` : Mês como um número decimal (de 01 a 12).
- `%b` : Mês abreviado em três letras (por exemplo, “Jan”, “Feb”, “Mar”).
- `%B` : Nome completo do mês (por exemplo, “January”, “February”, “March”).
- `%y` : Ano sem século (00-99).
- `%Y` : Ano com século (exemplo: 2022).

Com base na estrutura fornecida, o formato real das datas na coluna “Release.Date” parece ser `"%b %d, %Y"` (por exemplo, “Feb 25, 2022”). Portanto, para converter adequadamente as datas de `character` para `Date`, você deve usar este formato exato na função `as.Date()`. Certifique-se de ajustar o formato de acordo com o formato real das datas em sua coluna para garantir uma conversão correta.

```
# Criar a tabela com a coluna adicional
meses <- data.frame(
  Nome_Completo = c("Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho", "Julho", "Agosto", "Setembro", "Out
ubro", "Novembro", "Dezembro"),
  Abreviatura_PT = c("Jan", "Fev", "Mar", "Abr", "Mai", "Jun", "Jul", "Ago", "Set", "Out", "Nov", "Dez"),
  Abreviatura_EN = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
)

# Adicionar a coluna 'Igualdade'
meses$Igualdade <- ifelse(meses$Abreviatura_PT == meses$Abreviatura_EN, "o", "x")

# Mostrar a tabela
print(meses)
```

```
##    Nome_Completo Abreviatura_PT Abreviatura_EN Igualdade
## 1        Janeiro            Jan            Jan         o
## 2      Fevereiro            Fev            Feb         x
## 3          Março            Mar            Mar         o
## 4          Abril            Abr            Apr         x
## 5           Maio            Mai            May         x
## 6          Junho            Jun            Jun         o
## 7          Julho            Jul            Jul         o
## 8         Agosto            Ago            Aug         x
## 9       Setembro            Set            Sep         x
## 10       Outubro            Out            Oct         x
## 11      Novembro            Nov            Nov         o
## 12      Dezembro            Dez            Dec         x
```

O comando é usado para converter a coluna “Release.Date” em um formato de data específico. O comando `mdy` pertence ao pacote `lubridate` em R:

1. `new_dataset$Release.Date` : Esta parte do código acessa a coluna “Release.Date” no conjunto de dados `new_dataset` .
2. `mdy(new_dataset$Release.Date)` : A função `mdy` analisa a coluna “Release.Date” e a converte para o formato de data, assumindo o formato de mês-dia-ano. Ele reconhece automaticamente a ordem das informações de data no formato especificado e converte os valores de texto em objetos de data.

Portanto, o comando transforma os valores na coluna “Release.Date” em um formato de data adequado para facilitar a manipulação e a análise de datas no conjunto de dados `new_dataset` .

```
# Converta a coluna Release.Date para o formato de data
new_dataset$Release.Date <- mdy(new_dataset$Release.Date)
```

```
colSums(is.na(new_dataset))
```

```
## Release.Date
##            0
```

Funcionou.

7.6.2 Aplicando a mesma solução ao dataset novo_data:

Checando o dataset novamente:

```
str(novo_data)
```

```
## 'data.frame':    1506 obs. of  11 variables:
##  $ Title            : chr  "Elden Ring" "Hades" "The Legend of Zelda: Breath of the Wild" "Undertale" ...
##  $ Release.Date     : chr  "Feb 25, 2022" "Dec 10, 2019" "Mar 03, 2017" "Sep 15, 2015" ...
##  $ Team             : chr  "['Bandai Namco Entertainment', 'FromSoftware']" "['Supergiant Games']" "['Nintend
o', 'Nintendo EPD Production Group No. 3']" "['tobyfox', '8-4']" ...
##  $ Rating           : num  4.5 4.3 4.4 4.2 4.4 4.3 4.2 4.3 3 4.3 ...
##  $ Times.Listed     : num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Number.of.Reviews: num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Genres           : chr  "['Adventure', 'RPG']" "['Adventure', 'Brawler', 'Indie', 'RPG']" "['Adventure', 'RP
G']" "['Adventure', 'Indie', 'RPG', 'Turn Based Strategy']" ...
##  $ Plays            : num  17000 21000 30000 28000 21000 33000 7200 9200 25000 18000 ...
##  $ Playing          : num  3800 3200 2500 679 2400 1800 1100 759 470 1100 ...
##  $ Backlogs         : num  4600 6300 5000 4900 8300 1100 4500 3400 776 6200 ...
##  $ Wishlist         : num  4800 3600 2600 1800 2300 230 3800 3300 126 3600 ...
```

Aplicando o mesmo comando anterior ao dataset:

```
novo_data$Release.Date <- mdy(novo_data$Release.Date)
```

Checando o dataset novamente:

```
str(novo_data)
```

```
## 'data.frame':    1506 obs. of  11 variables:
##  $ Title            : chr  "Elden Ring" "Hades" "The Legend of Zelda: Breath of the Wild" "Undertale" ...
##  $ Release.Date     : Date, format: "2022-02-25" "2019-12-10" ...
##  $ Team             : chr  "['Bandai Namco Entertainment', 'FromSoftware']" "['Supergiant Games']" "['Nintend
o', 'Nintendo EPD Production Group No. 3']" "['tobyfox', '8-4']" ...
##  $ Rating           : num  4.5 4.3 4.4 4.2 4.4 4.3 4.2 4.3 3 4.3 ...
##  $ Times.Listed     : num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Number.of.Reviews: num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Genres           : chr  "['Adventure', 'RPG']" "['Adventure', 'Brawler', 'Indie', 'RPG']" "['Adventure', 'RP
G']" "['Adventure', 'Indie', 'RPG', 'Turn Based Strategy']" ...
##  $ Plays            : num  17000 21000 30000 28000 21000 33000 7200 9200 25000 18000 ...
##  $ Playing          : num  3800 3200 2500 679 2400 1800 1100 759 470 1100 ...
##  $ Backlogs         : num  4600 6300 5000 4900 8300 1100 4500 3400 776 6200 ...
##  $ Wishlist         : num  4800 3600 2600 1800 2300 230 3800 3300 126 3600 ...
```

Funcionou!!!!

8 Mostrando um resumo estatístico do dataset:

O comando `summary` em R é usado para gerar um resumo estatístico de objetos R. Quando aplicado a um conjunto de dados, como `novo_data`, o comando `summary` fornece um resumo estatístico das variáveis presentes no conjunto de dados.

Para um conjunto de dados, `summary` normalmente fornece as seguintes informações para cada variável:

1. **Mínimo e Máximo**: Os menores e maiores valores presentes na variável.
2. **Mediana**: A mediana dos valores da variável, que é o valor que divide os dados em duas partes iguais.
3. **Média**: A média dos valores da variável.
4. **Primeiro e Terceiro Quartis**: Os valores que dividem os dados em quatro partes iguais, onde o primeiro quartil é o valor abaixo do qual 25% dos dados se encontram e o terceiro quartil é o valor abaixo do qual 75% dos dados se encontram.
5. **Valores NA (ausentes)**: O número de valores ausentes para cada variável.

Portanto, quando você executa `summary(novo_data)`, o R fornece um resumo estatístico útil do conjunto de dados `novo_data`, permitindo que você entenda melhor a distribuição e as propriedades das variáveis presentes no conjunto de dados.

```
summary(novo_data)
```

```
##     Title            Release.Date            Team               Rating
##  Length:1506        Min.   :1980-05-22   Length:1506        Min.   :0.700
##  Class :character   1st Qu.:2007-09-27   Class :character   1st Qu.:3.400
##  Mode  :character   Median :2014-09-01   Mode  :character   Median :3.800
##                     Mean   :2012-09-18                      Mean   :3.718
##                     3rd Qu.:2019-09-12                      3rd Qu.:4.100
##                     Max.   :2025-03-31                      Max.   :4.600
##   Times.Listed    Number.of.Reviews    Genres              Plays
##  Min.   :   0.0   Min.   :   0.0    Length:1506        Min.   :    0
##  1st Qu.: 290.2   1st Qu.: 290.2    Class :character   1st Qu.: 1800
##  Median : 554.0   Median : 554.0    Mode  :character   Median : 4200
##  Mean   : 772.0   Mean   : 772.0                       Mean   : 6277
##  3rd Qu.:1000.0   3rd Qu.:1000.0                       3rd Qu.: 9100
##  Max.   :4300.0   Max.   :4300.0                       Max.   :33000
##     Playing          Backlogs         Wishlist
##  Min.   :   0.0   Min.   :   1.0   Min.   :   2.0
##  1st Qu.:  44.0   1st Qu.: 466.2   1st Qu.: 212.0
##  Median : 114.0   Median :1000.0   Median : 496.0
##  Mean   : 268.4   Mean   :1457.5   Mean   : 781.9
##  3rd Qu.: 301.0   3rd Qu.:2100.0   3rd Qu.:1100.0
##  Max.   :3800.0   Max.   :8300.0   Max.   :5400.0
```

8.1 Usando o pacote psych:

O comando `describe` em R fornece uma descrição estatística abrangente dos dados, incluindo medidas de tendência central, dispersão, assimetria, curtose e estatísticas relacionadas. No entanto, o comando `describe` não faz parte do R base. Geralmente, o comando `describe` é associado ao pacote `psych` em R.

Quando aplicado a um conjunto de dados, o comando `describe` fornece estatísticas descritivas detalhadas, incluindo:

1. **Média**: A média dos valores.
2. **Mediana**: A mediana dos valores.
3. **Desvio Padrão**: Uma medida de dispersão que indica a extensão em que os valores tendem a se desviar da média.
4. **Mínimo e Máximo**: Os menores e maiores valores presentes nos dados.
5. **Valores Ausentes**: O número de valores ausentes para cada variável.
6. **Estatísticas de Assimetria e Curtose**: Medidas que indicam a forma e a inclinação da distribuição dos dados.

Portanto, quando você executa `describe(novo_data)`, o R fornece uma descrição detalhada das propriedades estatísticas do conjunto de dados `novo_data`, permitindo uma compreensão mais profunda da distribuição e das propriedades dos dados.

```
describe(novo_data)
```

```
##                   vars    n    mean      sd median trimmed     mad min     max
## Title*               1 1506  551.85  315.65  558.5  552.76  411.42 1.0  1093.0
## Release.Date         2 1506     NaN      NA     NA     NaN      NA Inf    -Inf
## Team*                3 1506  385.68  213.77  402.0  386.31  266.87 1.0   763.0
## Rating               4 1506    3.72    0.53    3.8    3.77    0.44 0.7     4.6
## Times.Listed         5 1506  772.00  688.00  554.0  654.41  474.43 0.0  4300.0
## Number.of.Reviews    6 1506  772.00  688.00  554.0  654.41  474.43 0.0  4300.0
## Genres*              7 1506  131.71   64.99  125.0  130.39   60.79 1.0   253.0
## Plays                8 1506 6276.81 5895.10 4200.0 5348.99 4299.54 0.0 33000.0
## Playing              9 1506  268.36  427.01  114.0  174.24  130.47 0.0  3800.0
## Backlogs            10 1506 1457.45 1342.38 1000.0 1240.26 1014.84 1.0  8300.0
## Wishlist            11 1506  781.90  802.20  496.0  637.36  493.71 2.0  5400.0
##                     range  skew kurtosis     se
## Title*             1092.0 -0.02    -1.22   8.13
## Release.Date         -Inf    NA       NA     NA
## Team*               762.0 -0.06    -1.16   5.51
## Rating                3.9 -1.01     1.70   0.01
## Times.Listed       4300.0  1.76     3.54  17.73
## Number.of.Reviews  4300.0  1.76     3.54  17.73
## Genres*             252.0  0.30    -0.81   1.67
## Plays             33000.0  1.52     2.44 151.91
## Playing            3800.0  3.77    19.32  11.00
## Backlogs           8299.0  1.53     2.51  34.59
## Wishlist           5398.0  1.76     3.38  20.67
```

8.2 Criando gráficos Boxplot e usando o pacote reshape2 e ggplot2:

O processo de reestruturação do dataframe para o formato longo.

O comando utiliza a função `melt` do pacote `reshape2` em R.

Essa função é usada para converter um dataframe do formato largo para o formato longo, o que é útil para análise e visualização de dados.

1. `novo_data` : Este é o dataframe original que você deseja reestruturar.
2. `id.vars` : Este argumento especifica as variáveis que serão mantidas como identificadores durante a reestruturação. No seu caso, as colunas "Title", "Release.Date", "Team", "Genres", "Playing", "Backlogs" e "Wishlist" serão mantidas como identificadores.

O resultado dessa operação é um novo dataframe, chamado `novo_data_long` , que contém uma combinação dos valores das colunas especificadas em `id.vars` e uma coluna adicional chamada "variable" que armazena os nomes das colunas não especificadas em `id.vars` , bem como uma coluna "value" que armazena os valores correspondentes a essas colunas.

Essa reestruturação é útil quando você deseja comparar ou visualizar vários conjuntos de valores de diferentes variáveis em um único gráfico ou análise. Ele simplifica a tarefa de manipulação e visualização de dados, permitindo que você trabalhe com os dados de maneira mais eficiente e eficaz.

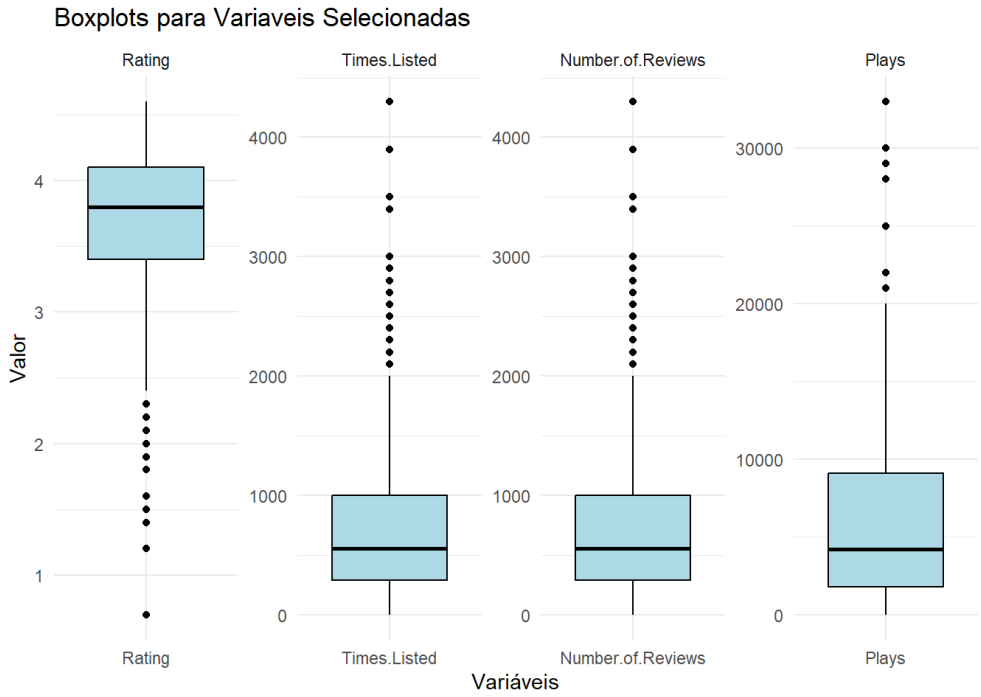
O código para criar um gráfico de boxplot lado a lado para as variáveis especificadas usando a biblioteca `ggplot2` em R:

1. `ggplot(novo_data_long, aes(x = variable, y = value))` : Esta linha inicializa o gráfico usando o dataframe `novo_data_long` . A estética (`aes`) é especificada, com a variável `variable` no eixo x e a variável `value` no eixo y.
2. `geom_boxplot(fill = "lightblue", color = "black")` : Isso adiciona os boxplots ao gráfico. A função `geom_boxplot` cria os boxplots, com a cor de preenchimento definida como "lightblue" e a cor da borda definida como "black".
3. `labs(title = "Boxplots para Variaveis Selecionadas", y = "Valor")` : Este trecho de código adiciona títulos aos gráficos e aos eixos. "Boxplots para Variaveis Selecionadas" é definido como o título do gráfico e "Valor" é definido como o rótulo do eixo y.
4. `theme_minimal()` : Isso aplica um tema minimalista ao gráfico, removendo elementos desnecessários e simplificando o visual.
5. `facet_wrap(~ variable, scales = "free", ncol = 6)` : Esta linha divide o gráfico em painéis com base na variável, permitindo a comparação lado a lado. A opção `scales = "free"` permite escalas livres para os eixos em cada painel, e `ncol = 6` especifica que deve haver 6 colunas de painéis.
6. `xlab("Variáveis")` : Isso adiciona um rótulo ao eixo x, definindo "Variáveis" como o rótulo do eixo x.

No geral, este código cria um gráfico de boxplot que permite comparar visualmente a distribuição de diferentes variáveis lado a lado, facilitando a identificação de padrões e diferenças entre as variáveis no conjunto de dados.

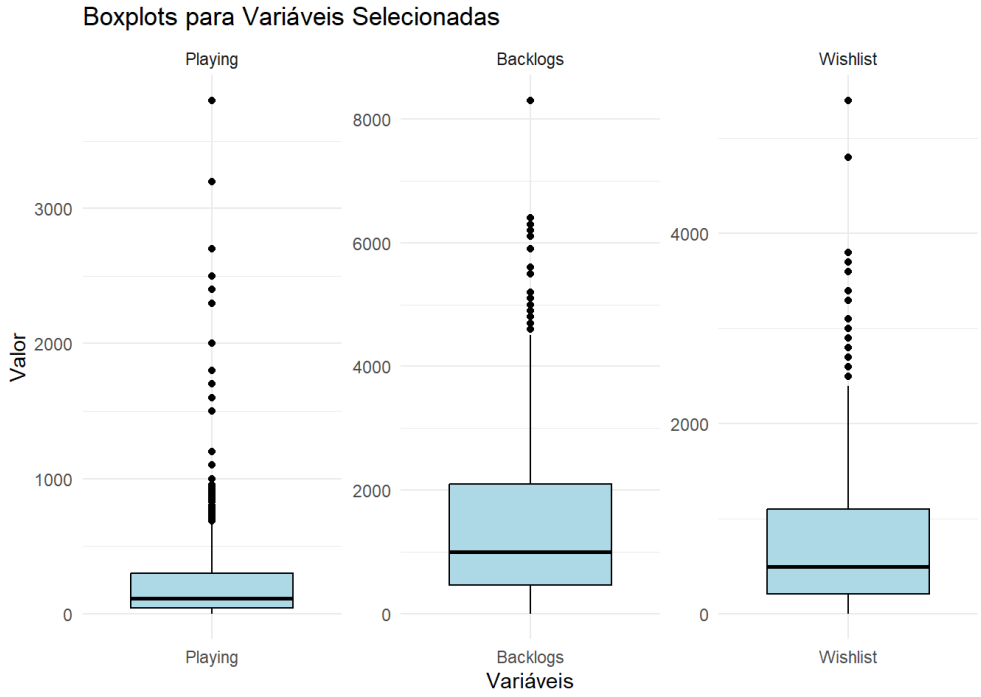
```
# Reestruturar o dataframe para o formato longo
novo_data_long <- reshape2::melt(novo_data, id.vars = c("Title", "Release.Date", "Team", "Genres", "Playing", "Bac
klogs", "Wishlist"))

# Criar um gráfico de boxplot lado a lado para as variáveis especificadas
ggplot(novo_data_long, aes(x = variable, y = value)) +
  geom_boxplot(fill = "lightblue", color = "black") +
  labs(title = "Boxplots para Variaveis Selecionadas", y = "Valor") +
  theme_minimal() +
  facet_wrap(~ variable, scales = "free", ncol = 6) +
  xlab("Variáveis")
```




```
# Reestruturar o dataframe para o formato longo
novo_data_long <- reshape2::melt(novo_data, id.vars = c("Title", "Release.Date", "Team", "Genres", "Rating", "Time
s.Listed", "Number.of.Reviews", "Plays"))

# Criar um gráfico de boxplot lado a lado para as variáveis especificadas
ggplot(novo_data_long, aes(x = variable, y = value)) +
  geom_boxplot(fill = "lightblue", color = "black") +
  labs(title = "Boxplots para Variáveis Selecionadas", y = "Valor") +
  theme_minimal() +
  facet_wrap(~ variable, scales = "free", ncol = 6) +
  xlab("Variáveis")
```



8.3 Criando gráficos de barras para os Gêneros e usando os pacotes dplyr, ggplot2, stringr e tidyverse:

8.3.1 Criando o gráfico para os gêneros combinados:

Este gráfico mostra as 25 maiores contagens de games (observações) para cada gênero, ou seja, mostra os 25 maiores gêneros com mais games.

O comando realiza a contagem de valores únicos em uma coluna específica do dataframe `novo_data` e cria um novo dataframe chamado `genre_counts`:

1. `novo_data$Genres`: Esta parte do código acessa a coluna "Genres" no dataframe `novo_data`.
2. `table(novo_data$Genres)`: A função `table` é usada para contar o número de ocorrências de valores únicos na coluna "Genres". Ele cria uma tabela de contagem dos valores únicos na coluna.
3. `as.data.frame()`: Este comando converte a tabela de contagem resultante em um dataframe. Isso cria um dataframe com duas colunas, uma para os valores únicos na coluna "Genres" e outra para a contagem de ocorrências correspondente a cada valor.

Portanto, o comando `genre_counts <- as.data.frame(table(novo_data$Genres))` cria um dataframe que contém a contagem de valores únicos na coluna "Genres" do dataframe `novo_data`, o que pode ser útil para analisar a distribuição e a frequência dos diferentes gêneros presentes no conjunto de dados.

O código as colunas de um dataframe chamado `genre_counts`:

1. `colnames(genre_counts)`: Esta parte do código acessa e modifica os nomes das colunas do dataframe `genre_counts`.
2. `<- c("Genres", "Count")`: Isso atribui novos nomes às colunas do dataframe. O nome "Genres" é atribuído à primeira coluna e "Count" é atribuído à segunda coluna.

Portanto, o comando `colnames(genre_counts) <- c("Genres", "Count")` renomeia as colunas do dataframe `genre_counts` para "Genres" e "Count", o que pode ser útil para tornar os nomes das colunas mais descritivos e legíveis durante a análise e a visualização dos dados.

```
genre_counts <- as.data.frame(table(novo_data$Genres))

# Renomeando as colunas
colnames(genre_counts) <- c("Genres", "Count")
```

O código usa o operador pipe (`%>%`) juntamente com algumas funções para realizar operações no dataframe `genre_counts`:

1. `genre_counts %>%`: O operador pipe (`%>%`) é usado para direcionar o dataframe `genre_counts` para a próxima operação, facilitando a leitura do código.
2. `arrange(desc(Count)) %>%`: A função `arrange` é usada para ordenar o dataframe com base na coluna "Count" em ordem decrescente. A função `desc` é usada para especificar que a ordenação deve ser em ordem decrescente. O operador pipe (`%>%`) direciona o dataframe resultante para a próxima operação.
3. `head(25)`: A função `head` é usada para selecionar as primeiras 25 linhas do dataframe resultante da operação `arrange`. Isso retorna as 25 principais linhas do dataframe, que correspondem aos gêneros com as contagens mais altas.

Portanto, o código `top_genres <- genre_counts %>% arrange(desc(Count)) %>% head(25)` cria um novo dataframe chamado `top_genres`, que contém os 25 principais gêneros com as contagens mais altas, com base nas contagens únicas de gêneros no dataframe original `genre_counts`. Isso é útil para identificar e analisar os gêneros mais comuns ou populares presentes nos dados.

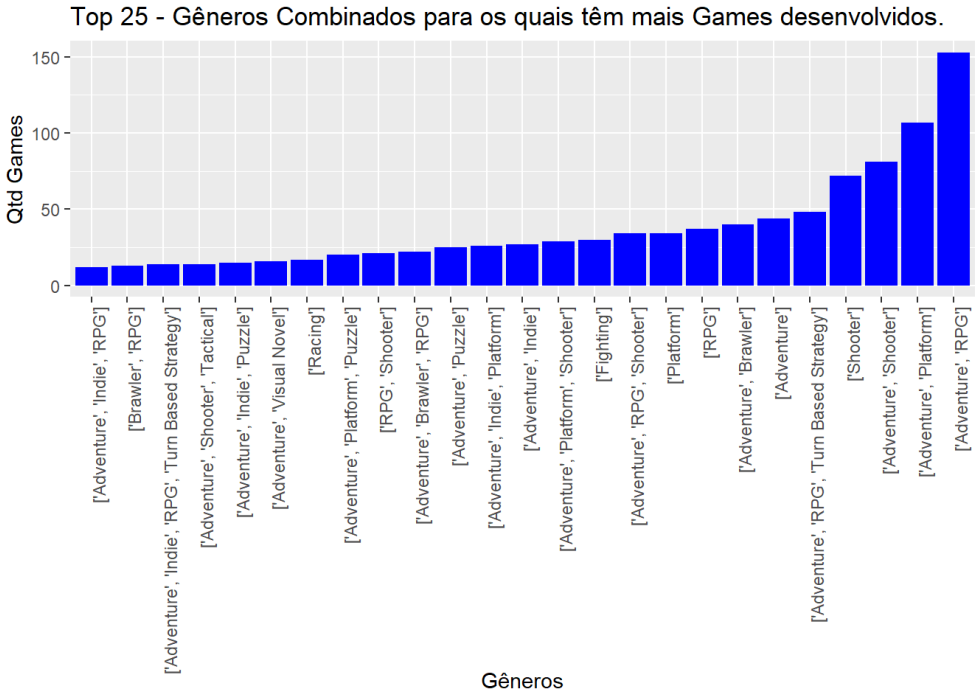
O código utiliza a biblioteca `ggplot2` em R para criar um gráfico de barras a partir do dataframe `top_genres`:

1. `ggplot(top_genres, aes(x = reorder(Genres, Count), y = Count))`: Esta linha inicializa o gráfico usando o dataframe `top_genres` e especifica a estética (`aes`) do gráfico. A função `reorder` é usada para reordenar os valores da coluna "Genres" com base na coluna "Count", para que os gêneros sejam plotados em ordem decrescente de contagem.
2. `geom_bar(stat = "identity", fill = "blue")`: Isso adiciona as barras ao gráfico. A função `geom_bar` cria um gráfico de barras, e `stat = "identity"` indica que os valores de altura das barras são fornecidos diretamente nos dados. A opção `fill = "blue"` define a cor de preenchimento das barras como azul.
3. `labs(title = "Top 25 - Gêneros Combinados para os quais têm mais Games desenvolvidos.", x = "Gêneros", y = "Qtd Games")`: Este trecho de código adiciona títulos aos gráficos e rótulos aos eixos. "Top 25 - Gêneros Combinados para os quais têm mais Games desenvolvidos." é definido como o título do gráfico, "Gêneros" é definido como o rótulo do eixo x e "Qtd Games" é definido como o rótulo do eixo y.
4. `theme(axis.text.x = element_text(angle = 90, hjust = 1))`: Isso ajusta o tema do gráfico, girando os rótulos do eixo x em um ângulo de 90 graus para evitar sobreposições, garantindo uma melhor legibilidade.

Em resumo, o código cria um gráfico de barras que representa os 25 principais gêneros com base na contagem de jogos para cada gênero, com os gêneros ordenados por contagem e representados por barras de altura proporcional à contagem correspondente. Isso ajuda a visualizar e comparar a distribuição dos gêneros de jogos mais populares.

```
# Ordenando o data frame em ordem decrescente e selecionando os 25 maiores valores
top_genres <- genre_counts %>%
  arrange(desc(Count)) %>%
  head(25)

# Criando o gráfico de barras
ggplot(top_genres, aes(x = reorder(Genres, Count), y = Count)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(title = "Top 25 - Gêneros Combinados para os quais têm mais Games desenvolvidos.", x = "Gêneros", y = "Qtd
Games") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



8.3.2 Criando gráfico para os gêneros isolados:

O código executa uma série de operações de manipulação de dados usando o pacote `dplyr` e o pacote `tidyr` em R:

1. `mutate(Genres = str_replace_all(Genres, "['\\[\\]]", ""))` : Esta linha remove caracteres indesejados da coluna "Genres" no dataframe `novo_data` . A função `str_replace_all` é usada para substituir todos os caracteres indesejados, como colchetes e aspas, por uma string vazia.
2. `separate_rows(Genres, sep = ", ")` : Isso separa os elementos na coluna "Genres" em linhas separadas, onde múltiplos gêneros separados por vírgula são divididos em linhas individuais.
3. `count(Genres)` : Isso conta o número de ocorrências de cada gênero no dataframe resultante da etapa anterior. A função `count` cria uma contagem dos valores únicos na coluna "Genres".
4. `rename(Genres.Nome = Genres, Genres.Count = n)` : Isso renomeia as colunas resultantes para "Genres.Nome" e "Genres.Count", fornecendo nomes mais descritivos para as colunas de gêneros e contagens.
5. `print(genres_data)` : Esta linha exibe o novo dataframe `genres_data` resultante das operações anteriores, mostrando os nomes de gêneros e suas respectivas contagens.

Portanto, o código em conjunto executa uma série de operações para limpar, dividir e contar os valores na coluna "Genres" do dataframe original `novo_data` , resultando em um novo dataframe chamado `genres_data` que contém os nomes dos gêneros e suas respectivas contagens.

```
# Remover caracteres indesejados e contar o número de ocorrências
genres_data <- novo_data %>%
  mutate(Genres = str_replace_all(Genres, "['\\[\\]]", "")) %>%
  separate_rows(Genres, sep = ", ") %>%
  count(Genres) %>%
  rename(Genres.Nome = Genres, Genres.Count = n)

# Mostrar o novo dataset
print(genres_data)
```

```
## # A tibble: 23 × 2
##    Genres.Nome       Genres.Count
##    <chr>                    <int>
##  1 Adventure                 1011
##  2 Arcade                      73
##  3 Brawler                    159
##  4 Card & Board Game           16
##  5 Fighting                    72
##  6 Indie                      283
##  7 MOBA                         3
##  8 Music                       24
##  9 Pinball                      1
## 10 Platform                   329
## # i 13 more rows
```

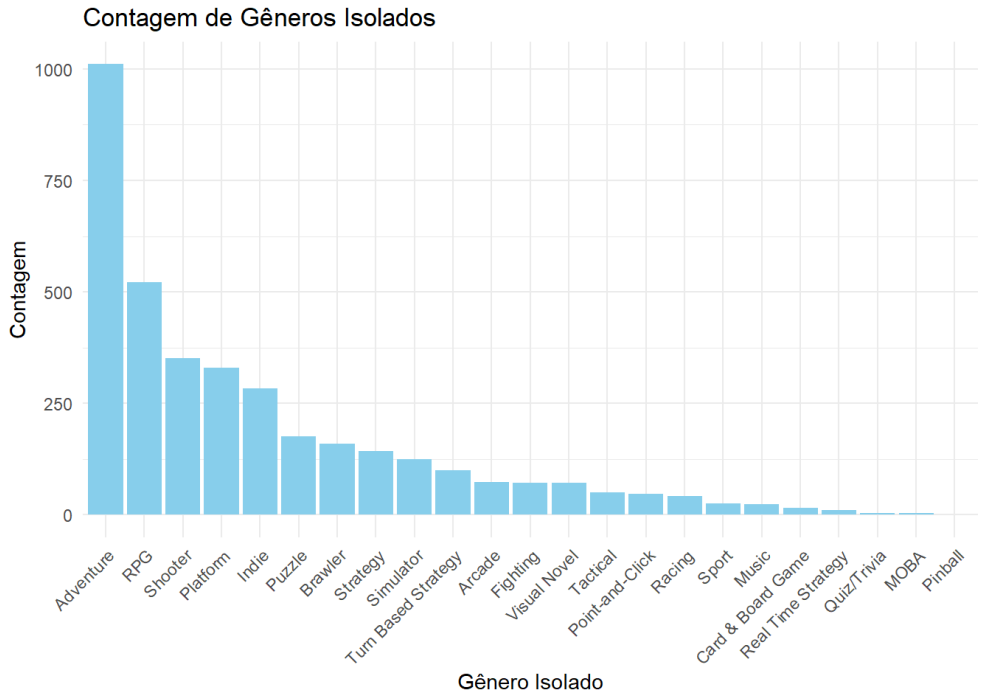
O código executa uma série de operações em um dataframe chamado `genres_data` e, em seguida, cria um gráfico de barras para visualizar as contagens de gêneros isolados:

1. `genres_data <- genres_data %>% arrange(desc(Genres.Count))` : Esta linha reordena o dataframe `genres_data` com base nas contagens de gêneros em ordem decrescente. O operador `%>%` é usado para encadear a operação.
2. `genres_data$Genres.Nome <- factor(genres_data$Genres.Nome, levels = genres_data$Genres.Nome)` : Isso converte a coluna "Genres.Nome" em um fator com níveis ordenados com base nos valores únicos presentes na coluna. Isso é útil para garantir que os valores sejam exibidos na ordem correta no gráfico.
3. `ggplot(genres_data, aes(x = Genres.Nome, y = Genres.Count))` : Esta linha inicializa o gráfico usando o dataframe `genres_data` e especifica a estética (`aes`) do gráfico, com "Genres.Nome" no eixo x e "Genres.Count" no eixo y.
4. `geom_bar(stat = "identity", fill = "skyblue")` : Isso adiciona as barras ao gráfico. A função `geom_bar` cria um gráfico de barras, e `stat = "identity"` indica que os valores de altura das barras são fornecidos diretamente nos dados. A opção `fill = "skyblue"` define a cor de preenchimento das barras como azul claro.
5. `labs(title = "Contagem de Gêneros Isolados", x = "Gênero Isolado", y = "Contagem")` : Este trecho de código adiciona títulos aos gráficos e rótulos aos eixos. "Contagem de Gêneros Isolados" é definido como o título do gráfico, "Gênero Isolado" é definido como o rótulo do eixo x e "Contagem" é definido como o rótulo do eixo y.
6. `theme_minimal()` : Isso aplica um tema minimalista ao gráfico, removendo elementos desnecessários e simplificando o visual.
7. `theme(axis.text.x = element_text(angle = 45, hjust = 1))` : Isso ajusta o tema do gráfico, girando os rótulos do eixo x em um ângulo de 45 graus para evitar sobreposições, garantindo uma melhor legibilidade.

O código resulta em um gráfico de barras que representa as contagens de gêneros isolados, ordenados de forma decrescente com os gêneros exibidos no eixo x e as contagens no eixo y, permitindo uma fácil visualização das distribuições de gêneros no conjunto de dados.

```
# Ordenar o dataset em ordem decrescente de Genres.Count
genres_data <- genres_data %>% arrange(desc(Genres.Count))
genres_data$Genres.Nome <- factor(genres_data$Genres.Nome, levels = genres_data$Genres.Nome)

# Gráfico de barras ordenado
ggplot(genres_data, aes(x = Genres.Nome, y = Genres.Count)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Contagem de Gêneros Isolados", x = "Gênero Isolado", y = "Contagem") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



9 Criando uma matriz de correlação, o heatmap e usando o pacote reshape2 e ggplot2:

Em uma matriz de correlação, os valores podem variar de -1 a 1. O sinal e a intensidade da cor em um gráfico de calor ou heatmap representam a força e a direção da correlação entre as variáveis. Estes são os significados das cores e dos valores de correlação:

1. Cores:
   - **Azul: Representa correlação negativa, ou seja, uma relação inversa entre as variáveis**.
   - **Vermelho: Representa correlação positiva, ou seja, uma relação direta entre as variáveis**.
2. Valores de correlação:
   - Próximo a -1: Indica uma correlação negativa forte.
   - Próximo a 0: Indica ausência de correlação linear.
   - Próximo a 1: Indica uma correlação positiva forte.

Portanto, nos gráficos de calor, os tons de azul indicam correlações negativas, enquanto os tons de vermelho indicam correlações positivas. Quanto mais intensa a cor, mais forte é a correlação. O valor exato da correlação é exibido no gráfico de calor para indicar a magnitude da relação entre as variáveis.

O código realiza algumas operações relacionadas à matriz de correlação em R:

1. `numeric_columns <- c("Rating", "Times.Listed", "Number.of.Reviews", "Plays", "Playing", "Backlogs", "Wishlist")`: Esta linha cria um vetor de caracteres chamado `numeric_columns`, que contém os nomes das colunas numéricas que serão usadas para calcular a matriz de correlação.
2. `correlation_matrix <- cor(novo_data[, numeric_columns], use = "pairwise.complete.obs")`: Este trecho de código calcula a matriz de correlação para as colunas numéricas específicas no dataframe `novo_data`. A função `cor` é usada para calcular as correlações entre as variáveis, e o argumento `use = "pairwise.complete.obs"` indica que a função deve usar apenas observações completas para o cálculo das correlações.

Portanto, o código seleciona colunas específicas do dataframe `novo_data` e calcula a matriz de correlação com base nessas colunas, o que é útil para analisar a relação entre as diferentes variáveis numéricas no conjunto de dados. A matriz de correlação resultante fornece informações sobre a força e a direção das relações lineares entre as variáveis numéricas especificadas.

```
# Selecionar as colunas numéricas para a matriz de correlação
numeric_columns <- c("Rating", "Times.Listed", "Number.of.Reviews", "Plays", "Playing", "Backlogs", "Wishlist")

# Criar a matriz de correlação
correlation_matrix <- cor(novo_data[, numeric_columns], use = "pairwise.complete.obs")
```

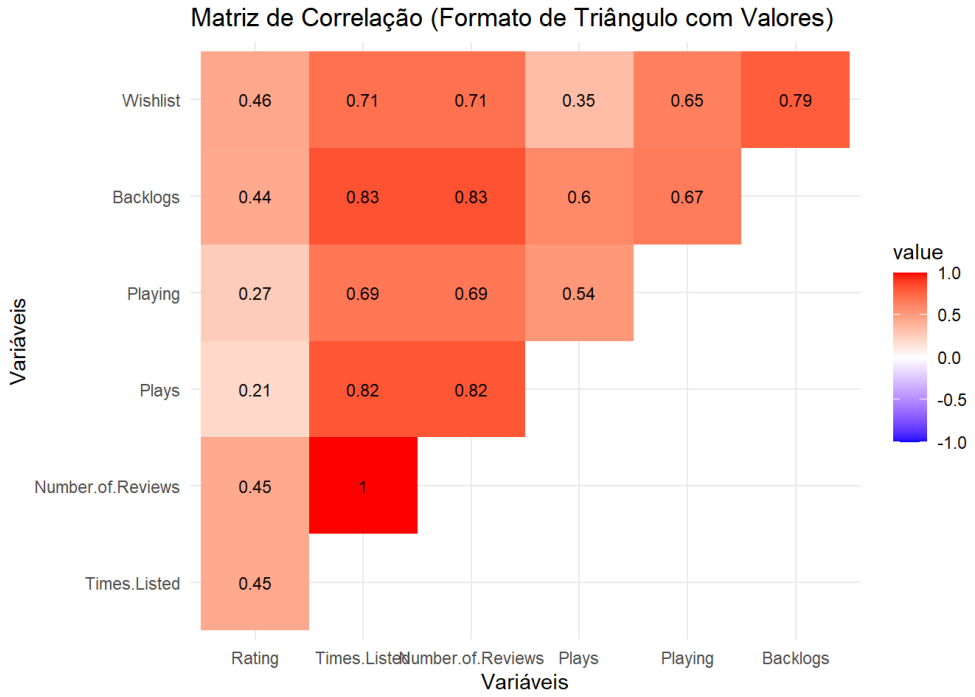
O código realiza a visualização da matriz de correlação em um gráfico de calor no formato de triângulo com valores de correlação:

1. `correlation_matrix[upper.tri(correlation_matrix, diag = TRUE)] <- NA` : Esta linha converte a parte superior da matriz de correlação em NA, para transformar a matriz em um formato de triângulo.
2. `ggplot(data = melt(correlation_matrix, na.rm = TRUE), aes(Var2, Var1, fill = value))` : Esta linha inicializa o gráfico de calor usando os dados derretidos da matriz de correlação. A função `melt` converte a matriz em um formato de dataframe adequado para o `ggplot` , e `Var2` e `Var1` representam os nomes das variáveis na matriz.
3. `geom_tile()` : Isso adiciona os azulejos ao gráfico de calor. A função `geom_tile` preenche os azulejos com cores com base nos valores de correlação.
4. `geom_text(aes(label = round(value, 2)), color = "black", size = 3)` : Isso adiciona os valores de correlação no gráfico de calor. A função `geom_text` exibe os valores de correlação arredondados para 2 casas decimais.
5. `scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0, limit = c(-1, 1))` : Isso define a escala de cores para o preenchimento dos azulejos, onde valores negativos são exibidos em azul, valores positivos em vermelho e o ponto médio em branco.
6. `theme_minimal()` : Isso aplica um tema minimalista ao gráfico, removendo elementos desnecessários e simplificando o visual.
7. `labs(title = "Matriz de Correlação (Formato de Triângulo com Valores)", x = "Variáveis", y = "Variáveis")` : Este trecho de código adiciona um título ao gráfico e rótulos aos eixos x e y.

Em resumo, o código cria um gráfico de calor que representa a matriz de correlação, exibindo as relações entre as variáveis como cores em um triângulo, com os valores de correlação exibidos dentro de cada célula correspondente. Isso é útil para visualizar padrões e relações entre as variáveis numéricas no conjunto de dados.

```
# Transformar a matriz de correlação em formato de triângulo
correlation_matrix[upper.tri(correlation_matrix, diag = TRUE)] <- NA

# Criar o gráfico de calor no formato de triângulo com valores de correlação
ggplot(data = melt(correlation_matrix, na.rm = TRUE), aes(Var2, Var1, fill = value)) +
  geom_tile() +
  geom_text(aes(label = round(value, 2)), color = "black", size = 3) +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0, limit = c(-1, 1)) +
  theme_minimal() +
  labs(title = "Matriz de Correlação (Formato de Triângulo com Valores)", x = "Variáveis", y = "Variáveis")
```



10 Listando o 20 games com maiores valores de “Rating”:

1. `top_games <- novo_data[order(-novo_data$Rating), ]` : Esta linha classifica o dataframe `novo_data` com base na coluna “Rating” em ordem decrescente, utilizando a função `order` . O sinal de menos (-) antes de `novo_data$Rating` indica que a ordenação será em ordem decrescente. O resultado é atribuído à variável `top_games` .

2. `top_games <- head(top_games, 20, )` : Esta linha seleciona as 20 primeiras linhas do dataframe `top_games` recém-ordenado. A função `head` é usada para selecionar as primeiras linhas, e 20 é o número de linhas que devem ser retornadas.
3. `top_games <- top_games[, c("Title", "Rating","Genres")]` : Esta linha seleciona apenas as colunas "Title", "Rating" e "Genres" do dataframe `top_games` , restringindo o dataframe para incluir apenas essas colunas.
4. `top_games` : Este comando final exibe o dataframe resultante `top_games` , que contém os títulos dos jogos, suas classificações e gêneros correspondentes. O resultado é uma visualização dos 20 melhores jogos com base nas classificações mais altas, juntamente com seus respectivos gêneros.

```
top_games <- novo_data[order(-novo_data$Rating), ]
top_games <- head(top_games, 20, )
top_games <- top_games[, c("Title", "Rating","Genres")]
top_games
```

```
##                                       Title Rating
## 29                Disco Elysium: The Final Cut    4.6
## 44                                  Outer Wilds    4.6
## 140                               Disco Elysium    4.6
## 253               Umineko: When They Cry Chiru    4.6
## 298                Bloodborne: The Old Hunters    4.6
## 355               Disco Elysium: The Final Cut    4.6
## 370                                 Outer Wilds    4.6
## 429                               Disco Elysium    4.6
## 540               Umineko: When They Cry Chiru    4.6
## 715              Hitman World of Assassination    4.6
## 802               Disco Elysium: The Final Cut    4.6
## 817                                 Outer Wilds    4.6
## 892                               Disco Elysium    4.6
## 955               Final Fantasy XIV: Endwalker    4.6
## 989            Metal Gear Solid 3: Subsistence    4.6
## 1033                Bloodborne: The Old Hunters    4.6
## 1071         Final Fantasy XIV: Shadowbringers    4.6
## 1088         The Great Ace Attorney 2: Resolve    4.6
## 1102 Sekiro: Shadows Die Twice - GOTY Edition    4.6
## 1136                            Half-Life: Alyx    4.6
##                                                      Genres
## 29                                ['Adventure', 'Indie', 'RPG']
## 44                ['Adventure', 'Indie', 'Puzzle', 'Simulator']
## 140                 ['Adventure', 'RPG', 'Turn Based Strategy']
## 253                               ['Adventure', 'Visual Novel']
## 298                                       ['Adventure', 'RPG']
## 355                               ['Adventure', 'Indie', 'RPG']
## 370               ['Adventure', 'Indie', 'Puzzle', 'Simulator']
## 429                 ['Adventure', 'RPG', 'Turn Based Strategy']
## 540                               ['Adventure', 'Visual Novel']
## 715                      ['Adventure', 'Shooter', 'Tactical']
## 802                               ['Adventure', 'Indie', 'RPG']
## 817               ['Adventure', 'Indie', 'Puzzle', 'Simulator']
## 892                 ['Adventure', 'RPG', 'Turn Based Strategy']
## 955                                       ['Adventure', 'RPG']
## 989                      ['Adventure', 'Shooter', 'Tactical']
## 1033                                      ['Adventure', 'RPG']
## 1071                                                  ['RPG']
## 1088 ['Adventure', 'Point-and-Click', 'Puzzle', 'Visual Novel']
## 1102                          ['Adventure', 'Brawler', 'RPG']
## 1136                        ['Adventure', 'Puzzle', 'Shooter']
```

11 Aplicação de função da família Apply:

```
str(novo_data)
```

```
## 'data.frame':    1506 obs. of  11 variables:
##  $ Title            : chr  "Elden Ring" "Hades" "The Legend of Zelda: Breath of the Wild" "Undertale" ...
##  $ Release.Date     : Date, format: "2022-02-25" "2019-12-10" ...
##  $ Team             : chr  "['Bandai Namco Entertainment', 'FromSoftware']" "['Supergiant Games']" "['Nintend
o', 'Nintendo EPD Production Group No. 3']" "['tobyfox', '8-4']" ...
##  $ Rating           : num  4.5 4.3 4.4 4.2 4.4 4.3 4.2 4.3 3 4.3 ...
##  $ Times.Listed     : num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Number.of.Reviews: num  3900 2900 4300 3500 3000 2300 1600 2100 867 2900 ...
##  $ Genres           : chr  "['Adventure', 'RPG']" "['Adventure', 'Brawler', 'Indie', 'RPG']" "['Adventure', 'RP
G']" "['Adventure', 'Indie', 'RPG', 'Turn Based Strategy']" ...
##  $ Plays            : num  17000 21000 30000 28000 21000 33000 7200 9200 25000 18000 ...
##  $ Playing          : num  3800 3200 2500 679 2400 1800 1100 759 470 1100 ...
##  $ Backlogs         : num  4600 6300 5000 4900 8300 1100 4500 3400 776 6200 ...
##  $ Wishlist         : num  4800 3600 2600 1800 2300 230 3800 3300 126 3600 ...
```

O código aplica a função `mean` a uma coluna específica do dataframe `novo_data` usando a função `apply`:

1. `mean_times_listed <- apply(novo_data["Times.Listed"], 2, mean)`: Esta linha calcula a média da coluna "Times.Listed" no dataframe `novo_data` usando a função `apply`.
    - O argumento `novo_data["Times.Listed"]` seleciona a coluna "Times.Listed" do dataframe `novo_data`.
    - O número 2 indica que a função `apply` deve ser aplicada às colunas da matriz ou dataframe.
    - A função `mean` é então aplicada a cada coluna, calculando a média dos valores da coluna "Times.Listed".
2. `mean_times_listed`: Este comando exibe o resultado do cálculo da média dos valores da coluna "Times.Listed", armazenando o resultado na variável `mean_times_listed`. Isso fornece a média dos valores presentes na coluna "Times.Listed" do dataframe `novo_data`.

```
# Aplicar a função mean a uma coluna específica usando apply
mean_times_listed <- apply(novo_data["Times.Listed"], 2, mean)
mean_times_listed
```

```
## Times.Listed
##     772.0013
```

O código aplica a função `sum` a uma coluna específica do dataframe `novo_data` usando a função `apply`:

1. `sum_plays <- apply(novo_data["Plays"], 2, sum)`: Esta linha calcula a soma dos valores na coluna "Plays" do dataframe `novo_data` usando a função `apply`.
    - O argumento `novo_data["Plays"]` seleciona a coluna "Plays" do dataframe `novo_data`.
    - O número 2 indica que a função `apply` deve ser aplicada às colunas da matriz ou dataframe.
    - A função `sum` é aplicada a cada coluna, calculando a soma dos valores na coluna "Plays".
2. `sum_plays`: Este comando exibe o resultado do cálculo da soma dos valores da coluna "Plays", armazenando o resultado na variável `sum_plays`. Isso fornece a soma total de todos os valores presentes na coluna "Plays" do dataframe `novo_data`.

```
# Aplicar a função sum a uma coluna específica usando apply
sum_plays <- apply(novo_data["Plays"], 2, sum)
sum_plays
```

```
##   Plays
## 9452870
```

O código a função `max` a uma coluna específica do dataframe `novo_data` usando a função `apply`:

1. `max_playing <- apply(novo_data["Playing"], 2, max)`: Esta linha calcula o valor máximo na coluna "Playing" do dataframe `novo_data` usando a função `apply`.
    - O argumento `novo_data["Playing"]` seleciona a coluna "Playing" do dataframe `novo_data`.
    - O número 2 indica que a função `apply` deve ser aplicada às colunas da matriz ou dataframe.
    - A função `max` é aplicada a cada coluna, identificando o valor máximo na coluna "Playing".
2. `max_playing`: Este comando exibe o resultado do cálculo do valor máximo da coluna "Playing", armazenando o resultado na variável `max_playing`. Isso fornece o valor máximo presente na coluna "Playing" do dataframe `novo_data`.

```
# Aplicar a função max a uma coluna específica usando apply
max_playing <- apply(novo_data["Playing"], 2, max)
max_playing
```

```
## Playing
##    3800
```

O código aplica a função `table` a uma coluna específica do dataframe `novo_data` usando a função `apply`:

1. `table_genres <- apply(novo_data["Genres"], 2, table)` : Esta linha cria uma tabela de contagem dos valores únicos na coluna "Genres" do dataframe `novo_data` usando a função `apply` .
    - O argumento `novo_data["Genres"]` seleciona a coluna "Genres" do dataframe `novo_data` .
    - O número 2 indica que a função `apply` deve ser aplicada às colunas da matriz ou dataframe.
    - A função `table` é aplicada a cada coluna, contando o número de ocorrências de cada valor único na coluna "Genres".
2. `head(table_genres, 20)` : Este comando exibe as primeiras 20 linhas da tabela resultante `table_genres` . Isso permite visualizar os primeiros 20 valores e suas contagens na coluna "Genres" do dataframe `novo_data` . A função `head` é usada para exibir as primeiras linhas da tabela resultante.

```
# Aplicar a função table a uma coluna específica usando apply
table_genres <- apply(novo_data["Genres"], 2, table)
head(table_genres, 20)
```

```
##                                                        Genres
## ['Adventure', 'Arcade', 'Brawler', 'Fighting']              1
## ['Adventure', 'Arcade', 'Fighting', 'Indie']                1
## ['Adventure', 'Arcade', 'Indie', 'Music', 'Platform']       1
## ['Adventure', 'Arcade', 'Indie', 'Music']                   1
## ['Adventure', 'Arcade', 'Indie', 'Platform', 'Puzzle', 'RPG']  1
## ['Adventure', 'Arcade', 'Indie', 'Platform', 'Shooter']     1
## ['Adventure', 'Arcade', 'Indie', 'Platform', 'Strategy']    3
## ['Adventure', 'Arcade', 'Indie', 'Platform']                1
## ['Adventure', 'Arcade', 'Indie', 'RPG', 'Shooter']          3
## ['Adventure', 'Arcade', 'Indie', 'Shooter']                 2
## ['Adventure', 'Arcade', 'Pinball']                          1
## ['Adventure', 'Arcade', 'Platform', 'Racing', 'Shooter']    1
## ['Adventure', 'Arcade', 'Platform', 'Shooter']              1
## ['Adventure', 'Arcade', 'Platform']                         3
## ['Adventure', 'Arcade', 'Racing', 'Shooter']                1
## ['Adventure', 'Brawler', 'Fighting', 'RPG']                 1
## ['Adventure', 'Brawler', 'Fighting']                        2
## ['Adventure', 'Brawler', 'Indie', 'Platform', 'RPG']        2
## ['Adventure', 'Brawler', 'Indie', 'Platform']               2
## ['Adventure', 'Brawler', 'Indie', 'RPG', 'Simulator', 'Strategy']  1
```

```
str(table_genres)
```

```
##  int [1:253, 1] 1 1 1 1 1 1 3 1 3 2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:253] "['Adventure', 'Arcade', 'Brawler', 'Fighting']" "['Adventure', 'Arcade', 'Fighting', 'Indi
e']" "['Adventure', 'Arcade', 'Indie', 'Music', 'Platform']" "['Adventure', 'Arcade', 'Indie', 'Music']" ...
##   ..$ : chr "Genres"
```

```
class(table_genres)
```

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

12 Exportando o dataset para um arquivo .csv:

```
# Exportar o dataframe para um arquivo CSV
write.csv(novo_data, file = "G:\\Meu Drive\\RLanguage\\TrabGrupoR\\novo_data.csv", row.names = FALSE)
```

13 Aplicando Análise de Regressão Linear Múltipla para verificar os Gêneros comercialmente mais significantes:

13.1 Criando variáveis ’dummy” (binárias) para a regressão:

```
# Preparação do dataframe que será utilizado ao final para Análise de Regressão Linear Multivariada
df_regressao <- novo_data %>%
  mutate(Generos = gsub("\\[|\\]", "", Genres),
         Genres = strsplit(Genres, ", "))
```

```
reg_novo_data <- df_regressao


reg_novo_data$Adventure <- ifelse(grepl('Adventure', reg_novo_data$Generos), 1, 0)
reg_novo_data$Indie <- ifelse(grepl('Indie', reg_novo_data$Generos), 1, 0)
reg_novo_data$Brawler <- ifelse(grepl('Brawler', reg_novo_data$Generos), 1, 0)
reg_novo_data$Platform <- ifelse(grepl('Platform', reg_novo_data$Generos), 1, 0)
reg_novo_data$Fighting <- ifelse(grepl('Fighting', reg_novo_data$Generos), 1, 0)
reg_novo_data$Arcade <- ifelse(grepl('Arcade', reg_novo_data$Generos), 1, 0)
reg_novo_data$RPG <- ifelse(grepl('RPG', reg_novo_data$Generos), 1, 0)
reg_novo_data$Shooter <- ifelse(grepl('Shooter', reg_novo_data$Generos), 1, 0)
reg_novo_data$MOBA <- ifelse(grepl('MOBA', reg_novo_data$Generos), 1, 0)
reg_novo_data$Simulator <- ifelse(grepl('Simulator', reg_novo_data$Generos), 1, 0)
reg_novo_data$Racing <- ifelse(grepl('Racing', reg_novo_data$Generos), 1, 0)
reg_novo_data$Puzzle <- ifelse(grepl('Puzzle', reg_novo_data$Generos), 1, 0)
reg_novo_data$Music <- ifelse(grepl('Music', reg_novo_data$Generos), 1, 0)
reg_novo_data$Strategy <- ifelse(grepl('Strategy', reg_novo_data$Generos), 1, 0)
reg_novo_data$Card_Board_Game <- ifelse(grepl('Card & Board Game', reg_novo_data$Generos), 1, 0)
reg_novo_data$Sport <- ifelse(grepl('Sport', reg_novo_data$Generos), 1, 0)
reg_novo_data$Visual_Novel <- ifelse(grepl('Visual Novel', reg_novo_data$Generos), 1, 0)
reg_novo_data$Point_and_Click <- ifelse(grepl('Point-and-Click', reg_novo_data$Generos), 1, 0)
reg_novo_data$Real_Time_Strategy <- ifelse(grepl('Real Time Strategy', reg_novo_data$Generos), 1, 0)
reg_novo_data$Turn_Based_Strateg <- ifelse(grepl('Turn Based Strateg', reg_novo_data$Generos), 1, 0)
reg_novo_data$Tactical <- ifelse(grepl('Tactical', reg_novo_data$Generos), 1, 0)
reg_novo_data$Quiz_Trivia <- ifelse(grepl('Quiz/Trivia', reg_novo_data$Generos), 1, 0)
reg_novo_data$Pinball <- ifelse(grepl('Pinball', reg_novo_data$Generos), 1, 0)
```

```
## Selecionando colunas para a regressão:
reg_select_novo_data <- select(reg_novo_data, Playing, Wishlist, Platform, Fighting, RPG, Shooter, Simulator, Raci
ng, Puzzle, Strategy, Sport, Visual_Novel, Point_and_Click, Real_Time_Strategy, Turn_Based_Strateg, Tactical)
```

```
## Rodando a Regressão Linear Múltipla:
modelo_reg_novo_data<- lm(formula = Playing~Wishlist+Platform+Fighting+RPG+Shooter+Simulator+Racing+Puzzle+Strateg
y+Sport+Visual_Novel+Point_and_Click+Real_Time_Strategy+Turn_Based_Strateg+Tactical, data = reg_select_novo_data)
```

13.2 Apresentando resultados da regressão, coeficientes e estatísticas de significância:

```
resumo_reg<- summary(modelo_reg_novo_data)
resumo_reg
```

```
##
## Call:
## lm(formula = Playing ~ Wishlist + Platform + Fighting + RPG +
##     Shooter + Simulator + Racing + Puzzle + Strategy + Sport +
##     Visual_Novel + Point_and_Click + Real_Time_Strategy + Turn_Based_Strateg +
##     Tactical, data = reg_select_novo_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1891.38  -129.60   -39.38    44.86  2551.84
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -56.25247   19.38731  -2.902 0.003768 **
## Wishlist             0.34165    0.01073  31.843  < 2e-16 ***
## Platform            21.88391   21.97174   0.996 0.319411
## Fighting            85.78060   40.25555   2.131 0.033261 *
## RPG                108.75334   20.63561   5.270 1.56e-07 ***
## Shooter             13.72934   21.00221   0.654 0.513400
## Simulator          139.19080   32.19492   4.323 1.64e-05 ***
## Racing              88.45950   51.42046   1.720 0.085582 .
## Puzzle             -97.58557   27.30242  -3.574 0.000362 ***
## Strategy            70.33922   34.41391   2.044 0.041137 *
## Sport               36.87776   66.25300   0.557 0.577871
## Visual_Novel        13.61358   40.78907   0.334 0.738611
## Point_and_Click    -43.87630   51.52830  -0.851 0.394629
## Real_Time_Strategy -69.88570  105.79289  -0.661 0.508978
## Turn_Based_Strateg -37.97086   46.59070  -0.815 0.415209
## Tactical           -57.37764   47.74543  -1.202 0.229655
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 317.8 on 1490 degrees of freedom
## Multiple R-squared:  0.4517, Adjusted R-squared:  0.4462
## F-statistic: 81.84 on 15 and 1490 DF,  p-value: < 2.2e-16
```

13.3 Resultados da Análise de Regressão:

Conforme quadro abaixo 'SUMMARY':

1. O modelo utiliza o Gênero de Jogo "Playing" variável dependente, como uma proxy do quanto as pessoas estão demandando determinado jogo. Sob o argumento de que quanto mais estiver sendo jogado pelas pessoas, maior o interesse delas por aquele jogo. As variáveis independentes são: Wishlist, Platform, Fighting, RPG, Shooter, Simulator, Racing, Puzzle, Strategy, Sport, Visual_Novel, Point_and_Click, Real_Time_Strategy, Turn_Based_Strateg, Tactical
2. O modelo se mostrou significante pois seu p-value ("F de Significação") de 2,2e-16 é menor que o nível de significância de 0,05;
3. O Rquadrado ajustado de 0,4464 indica que razoável parcela das variações na variável dependente são explicadas pelas variações nas variáveis independentes do modelo;
4. Os demais p-values de cada variável constantes do quadro, quando possuem valores abaixo de 0,05 do nível de significância, indicam que a o respectivo coeficiente estimado pelo modelo é significante. Ou seja, tem impacto sobre a variável dependente;
5. Em consonância com o item 'Brainstorming para análise dos dados', que indicou que concluiu que: 'devemos investir no desenvolvimento de um game que combine os gêneros 'Adventure', 'RPG' e 'Platform', a presente Análise de Regressão indica em seu modelo que a variável independente 'RPG', com coeficitente beta de 108,98 e com p-value de 1,28e-07 é bem recomendada pelo modelo dentre as três já citadas, pois 'Platform' apresenta p-value de 0,3, que é maior que o nível de significância de 0,05. E a 'Adventure' não foi tratada pelo modelo para evitar multicolinearidade com as demais variáveis.

14 Brainstorming para análise dos dados, propostas de solução e de linhas de ação a serem adotadas:

A pesquisa demonstrou que os games que fazem mais sucesso são os que combinam vários gêneros.

Os 3 gêneros combinados que fazem mais sucesso são:

- Adventure+RPG
- Adventure+Platform
- Adventure+Shooter

O 3 gêneros isolados que fazem mais sucesso são:

- Adventure
- RPG
- Platform

As variáveis que estão mais diretamente relacionadas (são diretamente proporcionais) são:

- Number.of.Reviews e Times.Listed
- Backlogs e Times.Listed
- Backlogs e Number.of.Reviews

Assim, devemos investir no desenvolvimento de um game que combine os gêneros Adventure, RPG e Platform.

Após o lançamento do game deveremos manter vigilância sobre as variáveis Number.of.Reviews, Backlogs e Times.Listed, para perceber a aceitação dele pelo mercado consumidor.