**CODE BOOK – SACI (Suicide Analytics for Counter Incidence)**

Instituição:

Colégio Visconde de Porto de Seguro – Unidade Panamby

R. Itapaiúna, 1355 - Panamby, São Paulo - SP, CEP: 05707-001

Estudante Autor do Projeto:

Andres Philipp Aeschlimann

Arthur Sandre de Almeida

Orientador:

Prof. Dr. Francisco Tupy

Coorientador: João Pedro Sassi Sandre

**Sumário**

**Código de tratamento dos dados**

1. Importação dos dados------------------------------------ p. 4
2. Normalização de cada base de dados------------------------------------ p. 5

2.1. Inflação------------------------------------ p. 5

2.2. PIB per capita------------------------------------ p. 7

2.3. Taxa de Desemprego------------------------------------ p. 8

2.4. IDH------------------------------------ p. 9

2.5. HFI------------------------------------ p. 10

2.6. Temperatura média------------------------------------ p. 12

2.7. Variáveis Religiosas------------------------------------ p. 13

1. Junção dos dados em uma base de dados final------------------------------------ p. 15

**Código de Análise dos dados**

1. Código Teste de Shapiro Wilk------------------------------------ p. 20
2. Código Teste de Kruskal-Wallis------------------------------------ p. 23
3. Código de Correlação de Spearmann------------------------------------ p. 26
4. Código de Regressões Individuais------------------------------------ p. 29
5. Código Regressão de Ridge------------------------------------ p. 48
6. Código do Mapa de Risco------------------------------------ p. 53

**Lista de Figuras**

Fig. 1 Base de dados inflação ajustada ------------------------------------p. 5

Fig. 2 Base de dados PIB per capita ajustada------------------------------------p. 7

Fig. 3 Base de dados Desemprego ajustada-----------------------------------p. 8

Fig. 4 Base de dados IDH ajustada -----------------------------------p. 9

Fig. 5 Base de dados HFI ajustada ------------------------------------p. 11

Fig. 6 Base de dados Temperatura Média ajustada ------------------------------------p. 12

Fig. 7 Base de dados Variáveis Religiosas ajustada ------------------------------------p. 14

Fig. 8 Junção das Variáveis Socioeconômicas------------------------------------p. 16

Fig. 9 Junção IDH, HFI e Temperatura Média-----------------------------------p. 18

Fig. 10 Junção Variáveis Religiosas. Data Frame Final------------------------------------p. 19

Fig. 11 Resultados Teste de Shapiro-Wilk------------------------------------p. 22

Fig. 12 Resultados Teste de Kruskal-Wallis------------------------------------p. 24

Fig. 13, 14, 15, 16 Resultados Correlação de Spearmann--------------p. 28

Fig. 17 Resultados Regresões Individuais: Temperatura Média e IDH----------p. 44

Fig. 18 Resultados Regresões Individuais: Temperatura Média e PIB per Capita---p. 46

Fig. 19 Resultados Regresões Individuais: Temperatura Média e HFI------p. 47

Fig. 20 Resultados Regresão de Ridge------------------------------------p. 51

**Código de Tratamento dos Dados**

**1. Importação dos dados**

import pandas as pd

from google.colab import files

uploaded = files.upload()

data\_1 = pd.read\_csv("resultado.csv")

data\_2 = pd.read\_csv("rounded\_percentage.csv")

data\_3 = pd.read\_csv("hfi\_cc\_2021.csv")

data\_4 = pd.read\_csv("final\_result\_r.csv")

data\_5 = pd.read\_csv("data\_2020\_filtrada (2).csv")

data\_6 = pd.read\_csv("resultadho.csv")

data\_7 = pd.read\_csv("global\_inflation\_data.csv")

data\_8 = pd.read\_csv("rounded\_percentage (1).csv")

data\_9 = pd.read\_csv("final\_result\_tem.csv")

data\_10 = pd.read\_csv("the weather of 187 countries in 2020.csv")

data\_11 = pd.read\_csv("hfi\_cc\_2022.csv")

data\_12 = pd.read\_csv("Human Development Index - Full.csv")

**2. Normalização de cada base de dados**

**2.1 Inflação:**

data\_7\_long = pd.melt(data\_7,

id\_vars=['country\_name', 'indicator\_name'],

var\_name='year',

value\_name='inflation\_rate')

# Remover a coluna 'indicator\_name', se não for necessária

data\_7\_long = data\_7\_long.drop(columns=['indicator\_name'])

# Exibir as primeiras linhas do dataframe reorganizado

data\_7\_2019 = data\_7\_long[data\_7\_long['year'] == '2019']

# Remover valores faltantes

data\_7\_2019\_clean = data\_7\_2019.dropna()

# Exibir as primeiras linhas do dataframe limpo

print(data\_7\_2019\_clean.head())

print(data\_7\_2019\_clean.head())

data\_7\_2019\_clean = data\_7\_2019\_clean.rename(columns={'country\_name': 'Pais', 'year': 'Ano'})

# Garantir que as colunas de ambos os dataframes estejam prontas para o merge

data\_7\_2019\_clean['Ano'] = data\_7\_2019\_clean['Ano'].astype(int)

data\_6['Ano'] = data\_6['Ano'].astype(int)

# Realizar o merge das duas bases de dados baseando-se no País e Ano

data\_merged = data\_7\_2019\_clean.merge(data\_6[['Pais', 'Ano', 'Taxa\_de\_inflacao']],

on=['Pais', 'Ano'],

suffixes=('', '\_novo'),

how='left') # Usar left join para manter todas as linhas de data\_7\_2019\_clean

# Verificar se a nova coluna foi criada corretamente

print(data\_merged.head())

# Substituir os valores de 'Taxa\_de\_inflacao' pelos novos valores, se existirem

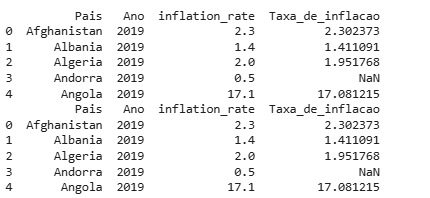
if 'Taxa\_de\_inflacao\_novo' in data\_merged.columns:

data\_merged['Taxa\_de\_inflacao'] = data\_merged['Taxa\_de\_inflacao\_novo'].combine\_first(data\_merged['Taxa\_de\_inflacao'])

# Remover a coluna extra 'Taxa\_de\_inflacao\_novo' criada no merge

data\_merged = data\_merged.drop(columns=['Taxa\_de\_inflacao\_novo'])

Figura 1: Base de dados Inflação ajustada



Fonte: Autores.

**2.2. PIB per capita:**

year\_columns = [str(year) for year in range(1960, 2025) if str(year) in data\_4.columns]

# Derretendo (melting) a base de dados para o PIB per capita

gdp\_per\_capita\_melt = pd.melt(data\_4, id\_vars=['Country Name', 'Code'],

value\_vars=year\_columns,

var\_name='Year',

value\_name='GDP\_per\_Capita') # Usando "\_" para evitar espaços

# Limpando a coluna 'Year' para deixá-la como int

gdp\_per\_capita\_melt['Year'] = gdp\_per\_capita\_melt['Year'].astype(int)

# Removendo duplicatas e valores NaN da coluna 'GDP per Capita'

gdp\_per\_capita\_melt = gdp\_per\_capita\_melt.drop\_duplicates()

gdp\_per\_capita\_melt = gdp\_per\_capita\_melt[gdp\_per\_capita\_melt['GDP\_per\_Capita'].notna()] # Usando "\_" aqui também

# Mostrando o DataFrame final

print(gdp\_per\_capita\_melt)

Figura 2: Base de dados PIB per capita ajustada



Fonte: Autores.

**2.3 Taxa de Desemprego:**

year\_columns = [str(year) for year in range(1991, 2022) if str(year) in data\_5.columns]

# Derretendo (melting) a base de dados para o indicador

desemprego\_melt = pd.melt(data\_5, id\_vars=['Country Name', 'Country Code'],

value\_vars=year\_columns,

var\_name='Year',

value\_name='Desemprego') # Usando 'Value' como nome genérico

# Limpando a coluna 'Year' para deixá-la como int

desemprego\_melt['Year'] = desemprego\_melt['Year'].astype(int)

# Removendo duplicatas e valores NaN da coluna 'Value'

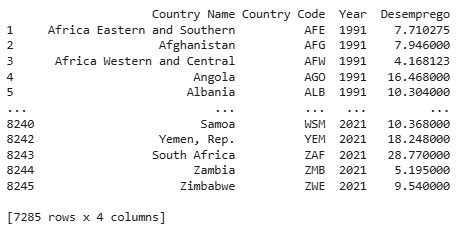
desemprego\_melt = desemprego\_melt.drop\_duplicates()

desemprego\_melt = desemprego\_melt[desemprego\_melt['Desemprego'].notna()]

# Mostrando o DataFrame final

print(desemprego\_melt)

Figura 3: Base de dados Taxa de Desemprego ajustada



Fonte: Autores.

**2.4 IDH:**

year\_columns = [str(year) for year in range(1990, 2020) if str(year) in data\_1.columns]

# Derretendo (melting) a base de dados para o IDH

hdi\_melt = pd.melt(data\_1, id\_vars=['Coverage', 'HDI Rank', 'Country'],

value\_vars=year\_columns,

var\_name='Year',

value\_name='HDI\_Value') # Nome da coluna para os valores de IDH

# Limpando a coluna 'Year' para deixá-la como int

hdi\_melt['Year'] = hdi\_melt['Year'].astype(int)

# Removendo duplicatas e valores NaN da coluna 'HDI\_Value'

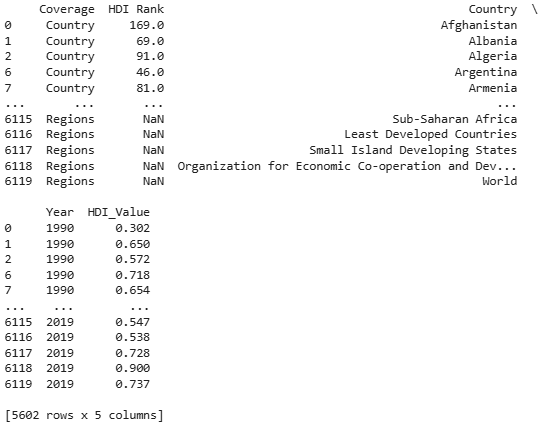
hdi\_melt = hdi\_melt.drop\_duplicates()

hdi\_melt = hdi\_melt[hdi\_melt['HDI\_Value'].notna()]

# Mostrando o DataFrame final

print(hdi\_melt)

Figura 4: Base de dados IDH ajustada



Fonte: Autores.

**2.5 HFI:**

data\_3\_filtered = data\_3[data\_3['year'] == 2019]

data\_merged['ISO3'] = data\_merged['ISO3'].str.strip().str.upper()

data\_3\_filtered['ISO'] = data\_3\_filtered['ISO'].str.strip().str.upper()

# 2. Verificar e remover valores nulos

print("Valores nulos em data\_merged:")

print(data\_merged['ISO3'].isnull().sum())

print("Valores nulos em data\_3\_filtered:")

print(data\_3\_filtered['ISO'].isnull().sum())

# Remover linhas com códigos nulos

data\_merged.dropna(subset=['ISO3'], inplace=True)

data\_3\_filtered.dropna(subset=['ISO'], inplace=True)

# 3. Verificar duplicatas

print("Duplicatas em data\_merged:")

print(data\_merged['ISO3'].duplicated().sum())

print("Duplicatas em data\_3\_filtered:")

print(data\_3\_filtered['ISO'].duplicated().sum())

# Remover duplicatas (caso necessário)

data\_merged.drop\_duplicates(subset=['ISO3'], inplace=True)

data\_3\_filtered.drop\_duplicates(subset=['ISO'], inplace=True)

# 4. Verificar códigos ausentes

missing\_in\_filtered = set(data\_merged['ISO3']) - set(data\_3\_filtered['ISO'])

print("Códigos ausentes em data\_3\_filtered:", missing\_in\_filtered)

missing\_in\_merged = set(data\_3\_filtered['ISO']) - set(data\_merged['ISO3'])

print("Códigos ausentes em data\_merged:", missing\_in\_merged)

# 5. Fazer a mesclagem com base nos códigos dos países

data\_merged = data\_merged.merge(

data\_3\_filtered[['ISO', 'hf\_score']],

left\_on='Código', # coluna em 'data\_merged'

right\_on='ISO', # coluna em 'data\_3\_filtered'

how='left',

suffixes=('', '\_by\_code')

)

# 6. Renomear a coluna hf\_score\_by\_code para um nome comum

data\_merged.rename(columns={'hf\_score\_by\_code': 'hf\_score\_merged\_final'}, inplace=True)

# 7. Remover valores faltantes na coluna resultante

data\_merged.dropna(inplace=True)

# 8. Exibir as primeiras linhas do DataFrame atualizado

print(data\_merged.head())

**2.6 Temperatura média:**

data\_2020 = data\_10[data\_10['Year'] == 2020]

# Filtrando dados que contêm 2020 na coluna DATE

# Calculando a média de TAVG por país

average\_tavg = data\_2020.groupby('Country/Region')['TAVG'].mean().reset\_index()

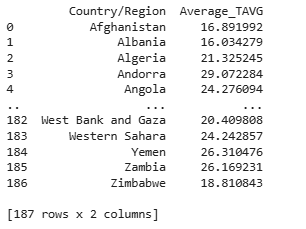
# Renomeando a coluna para Average\_TAVG

average\_tavg.rename(columns={'TAVG': 'Average\_TAVG'}, inplace=True)

# Exibindo a nova base de dados

print(average\_tavg)

Figura 6: Base de dados Temperatura Média ajustada



Fonte: Autores.

**2.7 Variáveis Religiosas**

mapping\_dict = {

'Bahamas, The': 'Bahamas',

'Bosnia and Herzegovina': 'Bosnia-Herzegovina',

'Brunei Darussalam': 'Brunei',

'Cabo Verde': 'Cape Verde',

'Gambia, The': 'Gambia',

'Kyrgyz Republic': 'Kyrgyzstan',

'Myanmar': 'Burma (Myanmar)',

'Russian Federation': 'Russia',

'Slovak Republic': 'Slovakia',

'Eswatini': 'Swaziland'

}

merged\_final['Pais\_Normalizado'] = merged\_final['Pais'].replace(mapping\_dict)

# 2. Criar listas de países a partir de data\_merged (normalizados e não normalizados)

paises\_normalizados = merged\_final['Pais\_Normalizado'].unique()

paises\_nao\_normalizados = merged\_final['Pais'].unique()

# 3. Combinar ambas as listas

todos\_paises = set(paises\_normalizados).union(set(paises\_nao\_normalizados))

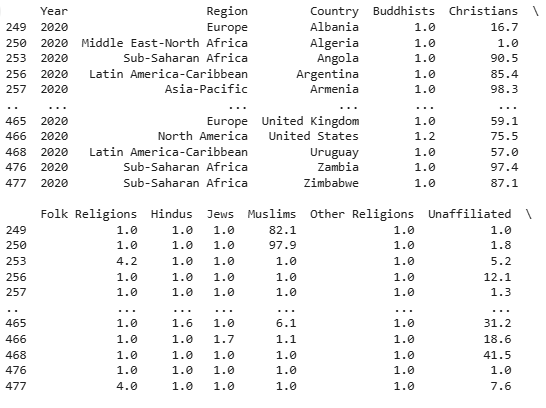
# 4. Filtrar average\_tavg com base em todos os países

df\_2020 = df\_2020[df\_2020['Country'].isin(todos\_paises)]

# Exibir o DataFrame filtrado

print(df\_2020)

Figura 7: Base de dados Variáveis Religiosas ajustada



Fonte: Autores.

**3. Junção dos dados em uma base de dados final:**

gdp\_per\_capita\_melt.rename(columns={'Country Name': 'Country', 'Year': 'Year'}, inplace=True)

desemprego\_melt.rename(columns={'Country Name': 'Country', 'Year': 'Year'}, inplace=True)

hdi\_melt.rename(columns={'Country': 'Country', 'Year': 'Year'}, inplace=True)

# Convertendo 'Year' para inteiro

for df in [gdp\_per\_capita\_melt, desemprego\_melt, hdi\_melt]:

df['Year'] = df['Year'].astype(int)

# Verificando os nomes das colunas

print("GDP per Capita Melted Columns:", gdp\_per\_capita\_melt.columns.tolist())

print("Desemprego Melted Columns:", desemprego\_melt.columns.tolist())

print("HDI Melted Columns:", hdi\_melt.columns.tolist())

# Realizando a junção dos DataFrames

merged\_df = pd.merge(merged\_df, gdp\_per\_capita\_melt, on=['Country', 'Year'], how='outer')

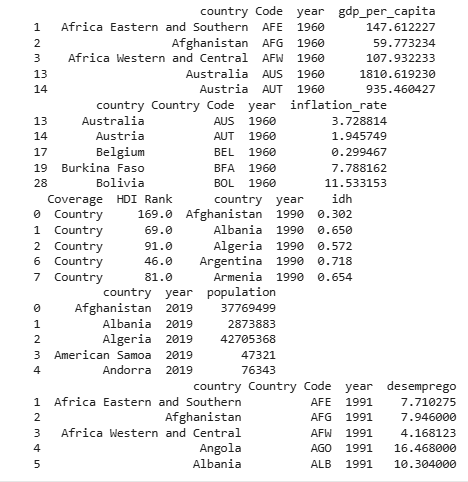
merged\_df = pd.merge(merged\_df, desemprego\_melt, on=['Country', 'Year'], how='outer')

merged\_df = pd.merge(merged\_df, hdi\_melt, on=['Country', 'Year'], how='outer')

# Visualizando o DataFrame final

print(merged\_df.head())

Figura 8: Junção das Variáveis Socioeconômicas



Fonte: Autores.

country\_mapping = {

"US": "United States",

"United Arab Emirates": "United Arab Emirates",

"Korea, Rep.": "South Korea",

"North Macedonia": "North Macedonia",

"Comoros": "Comoros",

"Venezuela": "Venezuela",

"Guinea": "Guinea",

"Burkina Faso": "Burkina Faso",

"Bahamas": "Bahamas",

"Albania": "Albania",

"Brazil": "Brazil"

}

# Invertendo o dicionário

inverted\_country\_mapping = {v: k for k, v in country\_mapping.items()}

# Renomear os países em data\_merged usando o dicionário

data\_merged['Pais'] = data\_merged['Pais'].replace(inverted\_country\_mapping)

# Exibir o DataFrame atualizado

print(data\_merged)

# Renomear os países em data\_merged

# Criar um set de países da segunda base, incluindo mapeamentos

lista\_paises = data\_merged['Pais'].unique().tolist()

# Adicionando países mapeados

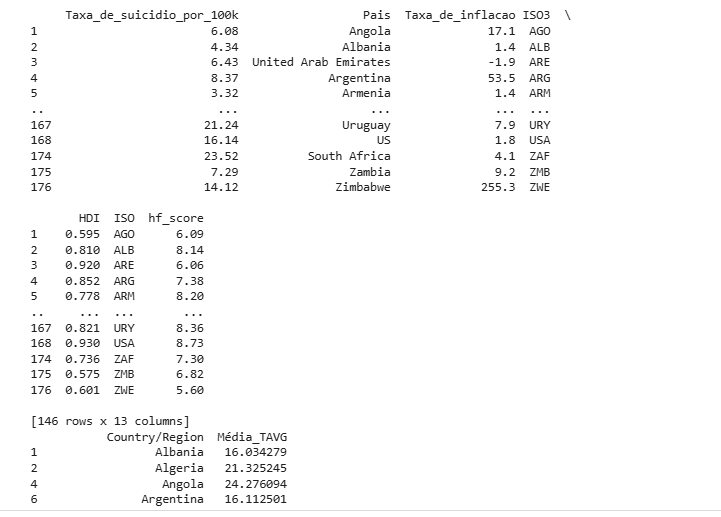
# Filtrar a primeira base de dados

media\_tavg\_por\_pais = media\_tavg\_por\_pais[media\_tavg\_por\_pais['Country/Region'].isin(lista\_paises)]

# Exibir o DataFrame filtrado

print(media\_tavg\_por\_pais)

Figura 9: Junção IDH, HFI e Temperatura Média

Fonte: Autores.

merged\_final['Buddhists'] = df\_2020\_atualizado['Buddhists']

merged\_final['Christians'] = df\_2020\_atualizado['Christians']

merged\_final['Folk Religions'] = df\_2020\_atualizado['Folk Religions']

merged\_final['Hindus'] = df\_2020\_atualizado['Hindus']

merged\_final['Jews'] = df\_2020\_atualizado['Jews']

merged\_final['Muslims'] = df\_2020\_atualizado['Muslims']

merged\_final['Other Religions'] = df\_2020\_atualizado['Other Religions']

merged\_final['Unaffiliated'] = df\_2020\_atualizado['Unaffiliated']

merged\_final = merged\_final.drop(columns=['Country/Region'])

# Reorganizar as colunas para que 'Pais' seja a segunda coluna

# Supondo que 'Pais' seja o nome da coluna que você deseja mover

colunas = merged\_final.columns.tolist() # Obter lista de colunas

colunas.remove('Pais') # Remover 'Pais' da lista

colunas.insert(1, 'Pais') # Inserir 'Pais' na segunda posição

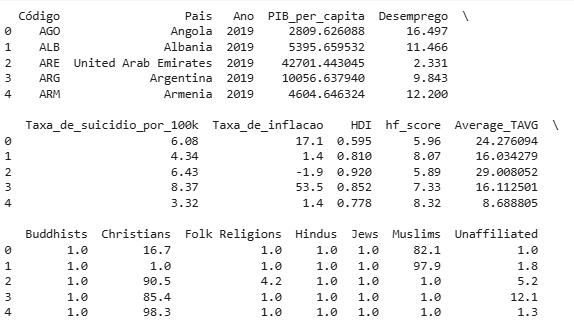
# Reorganizar o DataFrame com as novas colunas

df\_final = merged\_final[colunas]

# Exibir as primeiras linhas do DataFrame resultante

print(df\_final.head())

Figura 10: Junção Variáveis Religiosas. Data Frame Final



Fonte: Autores.

**Código de Análise dos dados**

**4. Código Teste de Shapiro Wilk**

# #Carregar as bibliotecas necessárias

library(dplyr) library(ggplot2)

# #Exibir o DataFrame original para conferência

print("DataFrame original:") print(df\_final)

# #Definir as variáveis numéricas que serão analisadas no teste de normalidade

variables\_to\_test <- c("PIB\_per\_capita", "Desemprego", "Taxa\_de\_inflacao", "HDI", "hf\_score", "Average\_TAVG", "Buddhists", "Christians", "Folk Religions", "Hindus", "Jews", "Muslims", "Unaffiliated", "Other Religions")

# #Criar um DataFrame para armazenar os resultados do teste Shapiro-Wilk

shapiro\_results <- data.frame( Variable = character(), W = numeric(), p\_value = numeric(), stringsAsFactors = FALSE )

# #Loop para realizar o teste Shapiro-Wilk em cada variável

for (var in variables\_to\_test) {

# #Remover valores ausentes antes do teste

data\_without\_na <- na.omit(df\_final[[var]])

# #Verifica se a variável é numérica antes de executar o teste

if (is.numeric(data\_without\_na)) { # Realiza o teste de normalidade Shapiro-Wilk test\_result <- shapiro.test(data\_without\_na)

# Armazena os resultados no DataFrame  
shapiro\_results <- rbind(shapiro\_results, data.frame(  
 Variable = var,  
 W = test\_result$statistic,  
 p\_value = test\_result$p.value  
))

} else { # Caso a variável não seja numérica, adiciona uma linha com valores NA shapiro\_results <- rbind(shapiro\_results, data.frame( Variable = var, W = NA, p\_value = NA )) } }

# #Exibir os resultados do teste Shapiro-Wilk

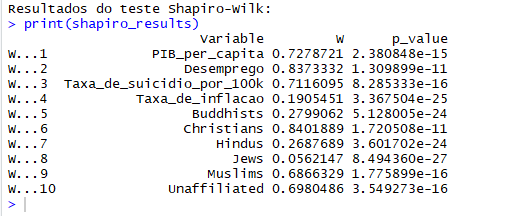
print("\nResultados do teste Shapiro-Wilk:") print(shapiro\_results)

# #Visualizar a distribuição de cada variável

for (var in variables\_to\_test) { p <- ggplot(df\_final, aes\_string(x = var)) + geom\_histogram(bins = 30, fill = "blue", alpha = 0.6) + theme\_minimal() + ggtitle(paste("Distribuição de", var)) print(p) }

Resultados:

Figura 11: Resultados Teste de Shapiro-Wilk



Fonte: Autores.

**5. Código Teste de Kruskal-Wallis**

# #Carregar as bibliotecas necessárias

library(dplyr) library(ggplot2) library(kruskal.test)

# #Carregar o dataset (substituir pelo caminho correto)

df\_final <- read.csv("df\_final.csv")

# #Criar diretório para salvar os gráficos

if (!dir.exists("plots\_kruskal")) { dir.create("plots\_kruskal") }

# #Definir as variáveis a serem testadas

variables\_to\_test <- c("PIB\_per\_capita", "Desemprego", "Taxa\_de\_inflacao", "HDI", "hf\_score", "Average\_TAVG", "Buddhists", "Christians", "Folk Religions", "Hindus", "Jews", "Muslims", "Unaffiliated", "Other Religions")

# #Criar um DataFrame para armazenar os resultados do teste Kruskal-Wallis

kruskal\_results <- data.frame( Variable = character(), H\_statistic = numeric(), p\_value = numeric(), stringsAsFactors = FALSE )

# #Loop para realizar o teste Kruskal-Wallis para cada variável

for (var in variables\_to\_test) {

# #Criar quartis (4 grupos)

df\_final[[paste0(var, "\_group")]] <- cut(df\_final[[var]], breaks = quantile(df\_final[[var]], probs = seq(0, 1, 0.25), na.rm = TRUE), include.lowest = TRUE, labels = c("Q1", "Q2", "Q3", "Q4"))

# #Realizar o teste de Kruskal-Wallis

test\_result <- kruskal.test(Taxa\_de\_suicidio\_por\_100k ~ df\_final[[paste0(var, "\_group")]], data = df\_final)

# #Armazenar os resultados

kruskal\_results <- rbind(kruskal\_results, data.frame( Variable = var, H\_statistic = test\_result$statistic, p\_value = test\_result$p.value ))

# #Criar e salvar o boxplot

p <- ggplot(df\_final, aes(x = get(paste0(var, "\_group")), y = Taxa\_de\_suicidio\_por\_100k)) + geom\_boxplot(fill = "lightblue", color = "black", outlier.color = "red", outlier.shape = 16) + theme\_classic() + labs( title = paste("Distribuição da Taxa de Suicídio por", var), x = var, y = "Taxa de Suicídio (por 100k)" ) + theme( plot.title = element\_text(hjust = 0.5, face = "bold", size = 14), axis.title.x = element\_text(size = 12, face = "bold"), axis.title.y = element\_text(size = 12, face = "bold"), axis.text = element\_text(size = 10) )

# #Salvar o gráfico na pasta "plots\_kruskal"

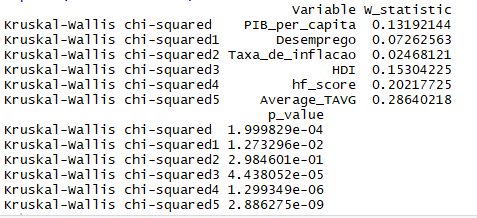
ggsave(filename = paste0("plots\_kruskal/", var, "\_kruskal.png"), plot = p, width = 8, height = 6, dpi = 300) }

# #Exibir os resultados do teste Kruskal-Wallis

print("Resultados do Teste de Kruskal-Wallis:") print(kruskal\_results)

Resultados:

Figura 12: Resultados Teste de Kruskal-Wallis



Fonte: Autores.

**6. Código de Correlação de Spearmann**

#Carregar os dados

df\_final <- read.csv("df\_final.csv")

#Definir variáveis para análise de correlação

variaveis\_interesse <- c("Desemprego" "HDI", "hf\_score", "Average\_TAVG", "Muslims", "Unaffiliated", "Taxa\_de\_suicidio\_por\_100k")

#Criar matriz de correlação de Spearman

spearman\_corr <- cor(df\_final[, variaveis\_interesse], method = "spearman", use = "complete.obs")

#Exibir matriz de correlação de Spearman

cat("\nMatriz de Correlação de Spearman:\n") print(spearman\_corr)

#Visualizar a matriz de correlação de Spearman

corrplot(spearman\_corr, method = "circle", type = "upper", tl.col = "black", tl.srt = 45, title = "Matriz de Correlação - Spearman", mar = c(0, 0, 1, 0))

#Criar lista para armazenar os resultados de Spearman

spearman\_results <- list()

#Criar um dataframe com os resultados

correlation\_df <- data.frame( Variavel = names(spearman\_results), Correlacao\_Spearman = sapply(spearman\_results, function(x) x$estimate), P\_Value = sapply(spearman\_results, function(x) x$p\_value) )

#Exibir os resultados

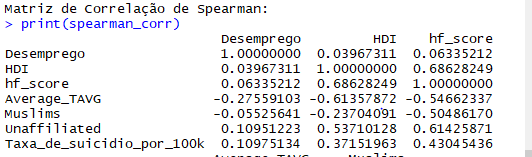
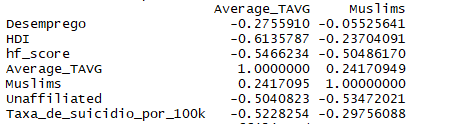
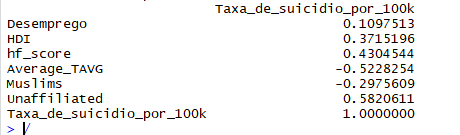
print(correlation\_df)

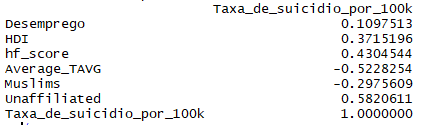
#Salvar a matriz de correlação em um arquivo CSV

write.csv(correlation\_df, "correlation\_results.csv", row.names = FALSE)

Resultados:

Figura 11,12,13 e 14: Resultados Correlação de Spearmann

Fonte: Autores.  


Fonte: Autores.

**7 Código de Regressões Individuais**

# Carregar bibliotecas necessárias

library(glmnet)

library(dplyr)

y <- df\_final$Taxa\_de\_suicidio\_por\_100k

# criar um data frame para armazenar resultados

resultados <- data.frame(

Variavel = character(),

Modelo = character(),

AIC = numeric(),

stringsAsFactors = FALSE

)

# Inicializar um data frame para armazenar p-values dos modelos com peso

p\_values\_com\_peso <- data.frame(

Variavel = character(),

P\_Value = numeric(),

stringsAsFactors = FALSE

)

# Função para ajustar modelos e calcular AIC

ajustar\_modelos <- function(y, x, nome\_variavel) {

# Modelo Linear

modelo\_linear <- lm(y ~ x)

aic\_linear <- AIC(modelo\_linear) # Calcular AIC

resultados <<- rbind(resultados, data.frame(

Variavel = nome\_variavel,

Modelo = "Linear",

AIC = aic\_linear

))

# Modelo Quadrático

modelo\_quadratico <- lm(y ~ poly(x, 2, raw = TRUE))

aic\_quadratico <- AIC(modelo\_quadratico) # Calcular AIC

resultados <<- rbind(resultados, data.frame(

Variavel = nome\_variavel,

Modelo = "Quadrático",

AIC = aic\_quadratico

))

# Modelo Cúbico

modelo\_cubico <- lm(y ~ poly(x, 3, raw = TRUE))

aic\_cubico <- AIC(modelo\_cubico) # Calcular AIC

resultados <<- rbind(resultados, data.frame(

Variavel = nome\_variavel,

Modelo = "Cúbico",

AIC = aic\_cubico

))

}

# Ajustar modelos para cada variável independente com dados originais

for (i in colnames(df\_final[, c(

"PIB\_per\_capita",

"Desemprego",

"Taxa\_de\_inflacao",

"IDH",

"hfi",

"temperatura\_media",

"Budistas",

"Cristaos",

"Religioes\_Folcloricas",

"Hindus",

"Judeus",

"Muculmanos",

"Não\_Afilhados",

"Outras\_Religioes"

)])) {

# Chamar a função ajustando o modelo apenas com a variável atual

ajustar\_modelos(y, df\_final[[i]], i)

}

# Printar todos os resultados dos modelos ajustados

print(resultados)

# Encontrar o melhor modelo para cada variável com base no menor AIC

melhores\_modelos <- resultados[ave(resultados$AIC, resultados$Variavel, FUN=min) == resultados$AIC, ]

# Printar o resumo do melhor modelo para cada variável e indicar qual é o melhor modelo

for (variavel in unique(melhores\_modelos$Variavel)) {

cat("\nResumo do Melhor Modelo para:", variavel, "\n")

modelo\_resumo <- melhores\_modelos[melhores\_modelos$Variavel == variavel, ]

# Imprimir os detalhes do melhor modelo encontrado

print(modelo\_resumo)

# Indicar qual é o melhor modelo com base no menor AIC

melhor\_modelo <- modelo\_resumo[which.min(modelo\_resumo$AIC), ]

cat("Melhor Modelo com base no menor AIC:", melhor\_modelo$Modelo, "\n")

# Ajustar o melhor modelo normal e imprimir resumo e AIC

if (melhor\_modelo$Modelo == "Linear") {

modelo\_normal <- lm(y ~ df\_final[[variavel]])

cat("\nResumo do Modelo Linear Normal:\n")

print(summary(modelo\_normal))

cat("AIC do Modelo Linear Normal:", AIC(modelo\_normal), "\n")

# Identificar outliers usando Z-score

residuos <- residuals(modelo\_normal)

z\_scores <- (residuos - mean(residuos)) / sd(residuos)

limite\_superior <- 3

# Atribuir peso zero aos outliers na modelagem sem removê-los do conjunto de dados

pesos <- ifelse(abs(z\_scores) > limite\_superior, 0, 1)

# Ajustar novamente o modelo linear com pesos e imprimir resumo e AIC

modelo\_com\_peso <- lm(y ~ df\_final[[variavel]], weights=pesos)

# Armazenar p-value do modelo com peso no DataFrame

p\_value <- summary(modelo\_com\_peso)$coefficients[2,4] # P-value da variável independente

p\_values\_com\_peso <<- rbind(p\_values\_com\_peso, data.frame(

Variavel = variavel,

P\_Value = p\_value

))

cat("\nResumo do Modelo Linear com Pesos:\n")

print(summary(modelo\_com\_peso))

cat("AIC do Modelo Linear com Pesos:", AIC(modelo\_com\_peso), "\n")

} else if (melhor\_modelo$Modelo == "Quadrático") {

modelo\_normal <- lm(y ~ poly(df\_final[[variavel]],2))

cat("\nResumo do Modelo Quadrático Normal:\n")

print(summary(modelo\_normal))

cat("AIC do Modelo Quadrático Normal:", AIC(modelo\_normal), "\n")

# Identificar outliers usando Z-score

residuos <- residuals(modelo\_normal)

z\_scores <- (residuos - mean(residuos)) / sd(residuos)

# Atribuir peso zero aos outliers na modelagem sem removê-los do conjunto de dados

pesos <- ifelse(abs(z\_scores) > limite\_superior,0,1)

# Ajustar novamente o modelo quadrático com pesos e imprimir resumo e AIC

modelo\_com\_peso <- lm(y ~ poly(df\_final[[variavel]],2), weights=pesos)

# Armazenar p-value do modelo com peso no DataFrame

p\_value <- summary(modelo\_com\_peso)$coefficients[2,4] # P-value da variável independente

p\_values\_com\_peso <<- rbind(p\_values\_com\_peso,data.frame(

Variavel = variavel,

P\_Value = p\_value

))

cat("\nResumo do Modelo Quadrático com Pesos:\n")

print(summary(modelo\_com\_peso))

cat("AIC do Modelo Quadrático com Pesos:", AIC(modelo\_com\_peso), "\n")

} else if (melhor\_modelo$Modelo == "Cúbico") {

modelo\_normal <- lm(y ~ poly(df\_final[[variavel]],3))

cat("\nResumo do Modelo Cúbico Normal:\n")

print(summary(modelo\_normal))

cat("AIC do Modelo Cúbico Normal:", AIC(modelo\_normal), "\n")

# Identificar outliers usando Z-score

residuos <- residuals(modelo\_normal)

z\_scores <- (residuos - mean(residuos)) / sd(residuos)

# Atribuir peso zero aos outliers na modelagem sem removê-los do conjunto de dados

pesos <- ifelse(abs(z\_scores) > limite\_superior,0,1)

# Ajustar novamente o modelo cúbico com pesos e imprimir resumo e AIC

modelo\_com\_peso <- lm(y ~ poly(df\_final[[variavel]],3), weights=pesos)

# Armazenar p-value do modelo com peso no DataFrame

p\_value <- summary(modelo\_com\_peso)$coefficients[2,4] # P-value da variável independente

p\_values\_com\_peso <<- rbind(p\_values\_com\_peso,data.frame(

Variavel = variavel,

P\_Value = p\_value

))

cat("\nResumo do Modelo Cúbico com Pesos:\n")

print(summary(modelo\_com\_peso))

cat("AIC do Modelo Cúbico com Pesos:", AIC(modelo\_com\_peso), "\n")

}

}

# Printar os p-values dos modelos ajustados com peso em um novo DataFrame

print(p\_values\_com\_peso)

Regressões cruzadas

# Carregar bibliotecas necessárias

library(dplyr)

# Definindo as variáveis dependentes

y\_temperatura <- df\_final$temperatura\_media

y\_nao\_afiliados <- df\_final$Não\_Afilhados

# Variáveis independentes a serem testadas

variaveis <- c("IDH", "PIB\_per\_capita", "hfi")

# Função para ajustar e comparar modelos

ajustar\_modelos <- function(y, x, nome\_variavel) {

# Ajustando modelos

modelo\_linear <- lm(y ~ x)

modelo\_quadratico <- lm(y ~ poly(x, 2, raw = TRUE))

modelo\_cubico <- lm(y ~ poly(x, 3, raw = TRUE))

# Calculando AICs

aic\_resultados <- data.frame(

Modelo = c("Linear", "Quadrático", "Cúbico"),

AIC = c(AIC(modelo\_linear), AIC(modelo\_quadratico), AIC(modelo\_cubico)),

Tipo = "Normal",

Variavel = nome\_variavel,

stringsAsFactors = FALSE

)

return(aic\_resultados)

}

# Função para ajustar modelos ponderados (sem outliers)

ajustar\_modelos\_ponderados <- function(y, x, nome\_variavel) {

# Ajuste do modelo normal para calcular resíduos

modelo\_normal <- lm(y ~ x)

# Cálculo dos resíduos e pesos

residuos <- residuals(modelo\_normal)

# Identificação de outliers usando z-scores

z\_scores <- (residuos - mean(residuos)) / sd(residuos)

pesos <- ifelse(abs(z\_scores) > 3, NA, 1) # Remove outliers

# Ajustando modelos ponderados

modelo\_linear\_ponderado <- lm(y ~ x, weights = pesos)

modelo\_quadratico\_ponderado <- lm(y ~ poly(x, 2), weights = pesos)

modelo\_cubico\_ponderado <- lm(y ~ poly(x, 3), weights = pesos)

# Calculando AICs ponderados

aic\_resultados\_ponderados <- data.frame(

Modelo = c("Linear", "Quadrático", "Cúbico"),

AIC = c(AIC(modelo\_linear\_ponderado), AIC(modelo\_quadratico\_ponderado), AIC(modelo\_cubico\_ponderado)),

Tipo = "Ponderado",

Variavel = nome\_variavel,

stringsAsFactors = FALSE

)

return(aic\_resultados\_ponderados)

}

# Inicializar data frames para armazenar resultados

resultados\_temperatura <- data.frame()

resultados\_nao\_afiliados <- data.frame()

# Ajustar modelos para Temperatura Média

for (variavel in variaveis) {

cat("\nAjustando modelos para Temperatura Média e", variavel, "\n")

resultados\_temperatura <- rbind(resultados\_temperatura, ajustar\_modelos(y\_temperatura, df\_final[[variavel]], variavel))

}

# Ajustar modelos para Taxa de Não Afilhados

for (variavel in variaveis) {

cat("\nAjustando modelos para Taxa de Não Afilhados e", variavel, "\n")

resultados\_nao\_afiliados <- rbind(resultados\_nao\_afiliados, ajustar\_modelos(y\_nao\_afiliados, df\_final[[variavel]], variavel))

}

# Comparar AICs dos modelos ponderados para Temperatura Média

for (variavel in variaveis) {

cat("\nAjustando modelos ponderados para Temperatura Média e", variavel, "\n")

resultados\_temperatura <- rbind(resultados\_temperatura, ajustar\_modelos\_ponderados(y\_temperatura, df\_final[[variavel]], variavel))

}

cat("\nResultados dos Modelos Ajustados para Temperatura Média (Incluindo Ponderados):\n")

print(resultados\_temperatura)

# Comparar AICs dos modelos ponderados para Taxa de Não Afilhados

for (variavel in variaveis) {

cat("\nAjustando modelos ponderados para Taxa de Não Afilhados e", variavel, "\n")

resultados\_nao\_afiliados <- rbind(resultados\_nao\_afiliados, ajustar\_modelos\_ponderados(y\_nao\_afiliados, df\_final[[variavel]], variavel))

}

cat("\nResultados dos Modelos Ajustados para Taxa de Não Afilhados (Incluindo Ponderados):\n")

print(resultados\_nao\_afiliados)

# Resumo dos melhores modelos com base no menor AIC

melhores\_modelos\_temperatura <- resultados\_temperatura %>%

group\_by(Variavel) %>%

filter(AIC == min(AIC))

melhores\_modelos\_nao\_afiliados <- resultados\_nao\_afiliados %>%

group\_by(Variavel) %>%

filter(AIC == min(AIC))

cat("\nMelhores Modelos para Temperatura Média:\n")

print(melhores\_modelos\_temperatura)

cat("\nMelhores Modelos para Taxa de Não Afilhados:\n")

print(melhores\_modelos\_nao\_afiliados)

# Resumo dos melhores modelos ajustados

for (i in seq\_len(nrow(melhores\_modelos\_temperatura))) {

variavel <- melhores\_modelos\_temperatura$Variavel[i]

modelo\_tipo <- melhores\_modelos\_temperatura$Modelo[i]

if (modelo\_tipo == "Linear") {

melhor\_modelo\_ajustado <- lm(y\_temperatura ~ df\_final[[variavel]])

} else if (modelo\_tipo == "Quadrático") {

melhor\_modelo\_ajustado <- lm(y\_temperatura ~ poly(df\_final[[variavel]],2))

} else if (modelo\_tipo == "Cúbico") {

melhor\_modelo\_ajustado <- lm(y\_temperatura ~ poly(df\_final[[variavel]],3))

}

cat("\nResumo do Melhor Modelo para Temperatura Média e:", variavel, "\n")

print(summary(melhor\_modelo\_ajustado))

}

for (i in seq\_len(nrow(melhores\_modelos\_nao\_afiliados))) {

variavel <- melhores\_modelos\_nao\_afiliados$Variavel[i]

modelo\_tipo <- melhores\_modelos\_nao\_afiliados$Modelo[i]

if (modelo\_tipo == "Linear") {

melhor\_modelo\_nao\_afiliado\_ajustado<-lm(y\_nao\_afiliados ~ df\_final[[variavel]])

} else if (modelo\_tipo == "Quadrático") {

melhor\_modelo\_nao\_afiliado\_ajustado<-lm(y\_nao\_afiliados ~ poly(df\_final[[variavel]],2))

} else if (modelo\_tipo == "Cúbico") {

melhor\_modelo\_nao\_afiliado\_ajustado<-lm(y\_nao\_afiliados ~ poly(df\_final[[variavel]],3))

}

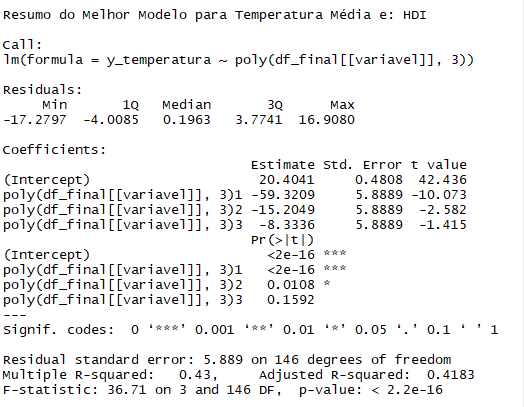
cat("\nResumo do Melhor Modelo para Taxa de Não Afilhados e:", variavel, "\n")

print(summary(melhor\_modelo\_nao\_afiliado\_ajustado))

}

Resultados:

Figura 13: Resultados Regressões Individuais: Temperatura Média e IDH



Fonte: Autores.

Figura 14: Resultados Regressões Individuais: Temperatura Média e PIB per capita

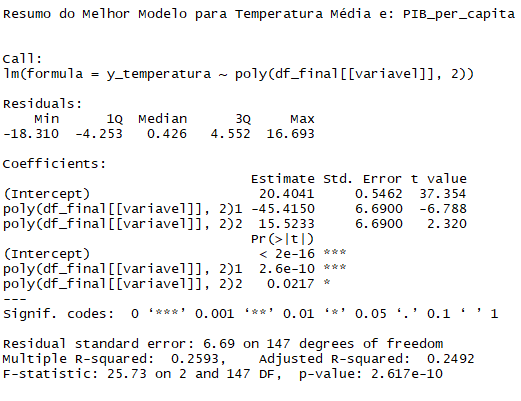
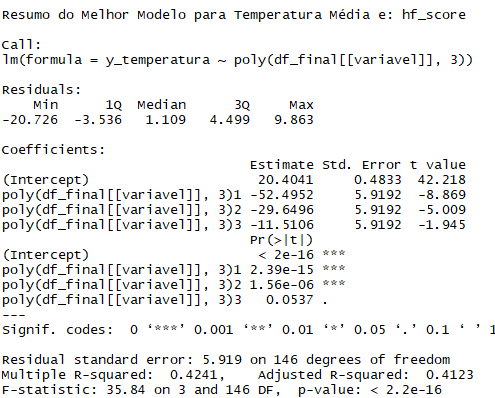


Figura 13: Resultados Regressões Individuais: Temperatura Média e HFI



Fonte: Autores.

**8. Código Regressão de Ridge**

#Carregar bibliotecas necessarias

library(glmnet) library(dplyr)

#Definir a variavel dependente

y <- df\_final$Taxa\_de\_suicidio\_por\_100k

#Identificar outliers usando Z-score

residuos <- residuals(lm(y ~ df\_final$Desemprego)) z\_scores <- (residuos - mean(residuos)) / sd(residuos) limite\_superior <- 3

#Filtrar dados para remover outliers

df\_sem\_outliers <- df\_final[abs(z\_scores) <= limite\_superior, ]

#Definir a matriz de variaveis preditoras sem outliers

x\_sem\_outliers <- data.matrix(df\_sem\_outliers[, c( "Desemprego", "HDI", "hf\_score", "Average\_TAVG", "Unaffiliated", "Muslims" )])

#Ajustar o modelo linear sem outliers

modelo\_linear\_sem\_outliers <- lm(Taxa\_de\_suicidio\_por\_100k ~ x\_sem\_outliers, data = df\_sem\_outliers) summary\_modelo\_linear\_sem\_outliers <- summary(modelo\_linear\_sem\_outliers)

#Imprimir resumo do modelo linear sem outliers

cat("\nResumo do Modelo Linear Sem Outliers:\n") print(summary\_modelo\_linear\_sem\_outliers)

#Ajustar modelo Ridge Regression sem outliers

modelo\_sem\_outliers <- glmnet(x\_sem\_outliers, df\_sem\_outliers$Taxa\_de\_suicidio\_por\_100k, alpha = 0)

#Exibir coeficientes do modelo sem outliers

cat("\nCoeficientes do Modelo Sem Outliers (Ridge Regression):\n") coeficientes\_sem\_outliers <- coef(modelo\_sem\_outliers) print(coeficientes\_sem\_outliers)

#Validacao cruzada para encontrar o lambda otimo

cv\_modelo\_sem\_outliers <- cv.glmnet(x\_sem\_outliers, df\_sem\_outliers$Taxa\_de\_suicidio\_por\_100k, alpha = 0) lambda\_otimo\_sem\_outliers <- cv\_modelo\_sem\_outliers$lambda.min cat("Lambda otimo (sem outliers):", lambda\_otimo\_sem\_outliers, "\n")

#Ajustar o melhor modelo

melhor\_modelo\_sem\_outliers <- glmnet(x\_sem\_outliers, df\_sem\_outliers$Taxa\_de\_suicidio\_por\_100k, alpha = 0, lambda = lambda\_otimo\_sem\_outliers)

#Fazer previsoes

y\_previsto\_sem\_outliers <- predict(melhor\_modelo\_sem\_outliers, s = lambda\_otimo\_sem\_outliers, newx = x\_sem\_outliers)

#Calcular R-quadrado

sst\_sem\_outliers <- sum((df\_sem\_outliers$Taxa\_de\_suicidio\_por\_100k - mean(df\_sem\_outliers$Taxa\_de\_suicidio\_por\_100k))^2) sse\_sem\_outliers <- sum((y\_previsto\_sem\_outliers - df\_sem\_outliers$Taxa\_de\_suicidio\_por\_100k)^2) rsq\_sem\_outliers <- 1 - sse\_sem\_outliers/sst\_sem\_outliers cat("R-quadrado (sem outliers):", rsq\_sem\_outliers, "\n")

#Ajustar o modelo com todos os dados (com outliers)

x\_com\_outliers <- data.matrix(df\_final[, c( "Desemprego", "HDI", "hf\_score", "Average\_TAVG", "Muslims", "Unaffiliated" )])

modelo\_linear\_com\_outliers <- lm(Taxa\_de\_suicidio\_por\_100k ~ Desemprego + Christians + Folk Religions + HDI + hf\_score + Average\_TAVG + Muslims + Unaffiliated + Other Religions, data = df\_final) summary\_modelo\_linear\_com\_outliers <- summary(modelo\_linear\_com\_outliers)

#Imprimir resumo do modelo linear com outliers

cat("\nResumo do Modelo Linear Com Outliers:\n") print(summary\_modelo\_linear\_com\_outliers)

#Ajustar modelo Ridge Regression com outliers

modelo\_com\_outliers <- glmnet(x\_com\_outliers, y, alpha = 0)

#Exibir coeficientes do modelo com outliers

cat("\nCoeficientes do Modelo Com Outliers (Ridge Regression):\n") coeficientes\_com\_outliers <- coef(modelo\_com\_outliers) print(coeficientes\_com\_outliers)

#Validacao cruzada para encontrar o lambda otimo

cv\_modelo\_com\_outliers <- cv.glmnet(x\_com\_outliers, y, alpha = 0) lambda\_otimo\_com\_outliers <- cv\_modelo\_com\_outliers$lambda.min cat("Lambda otimo (com outliers):", lambda\_otimo\_com\_outliers, "\n")

#Ajustar o melhor modelo

melhor\_modelo\_com\_outliers <- glmnet(x\_com\_outliers, y, alpha = 0, lambda = lambda\_otimo\_com\_outliers)

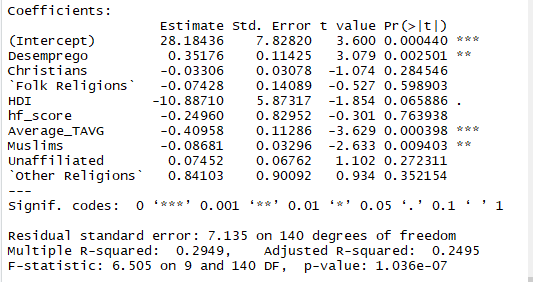
#Fazer previsoes

y\_previsto\_com\_outliers <- predict(melhor\_modelo\_com\_outliers, s = lambda\_otimo\_com\_outliers, newx = x\_com\_outliers)

#Calcular R-quadrado

sst\_com\_outliers <- sum((y - mean(y))^2) sse\_com\_outliers <- sum((y\_previsto\_com\_outliers - y)^2) rsq\_com\_outliers <- 1 - sse\_com\_outliers/sst\_com\_outliers cat("R-quadrado (com outliers):", rsq\_com\_outliers, "\n")

Resultados:

Figura 16: Resultados Regressão de Ridge 

Fonte: Autores.

**9 Código do Mapa de risco**

#Carregar os pacotes necessários

library(dplyr)

#Carregar a base de dados

df\_final <- read.csv("df\_final.csv", stringsAsFactors = FALSE)

#Função para normalizar os valores entre 0 e 100, garantindo que o maior seja 100

normalize <- function(x) { max\_val <- max(x, na.rm = TRUE)

#Se o maior valor for 0, retorna um vetor de zeros para evitar erro de divisão

if (max\_val == 0) return(rep(0, length(x)))

return ((x / max\_val) \* 100) }

#Definir as variáveis que serão analisadas

variaveis\_interesse <- c("Desemprego", "IDH", "PIB\_per\_capita", "hfi", "temperatura\_media", "Muculmanos", "Não\_Afilhados")

#Criar um DataFrame para armazenar os resultados

resultados <- df\_final %>% select(Pais) %>% mutate(across(all\_of(variaveis\_interesse), ~ NA\_real\_)) # Inicializa com valores NA

#Criar colunas adicionais para armazenar coeficientes e produtos

for (variavel in variaveis\_interesse) { resultados[[paste0("Coef\_Mult\_", variavel)]] <- NA\_real\_ resultados[[paste0("Produto\_", variavel)]] <- NA\_real\_ }

#Iterar sobre as variáveis para normalizar e calcular coeficientes

for (variavel in variaveis\_interesse) {

#Verifica se a variável existe no dataset antes de prosseguir

if (variavel %in% colnames(df\_final)) {

valor\_original <- df\_final[[variavel]]

# Se houver valores não nulos, realiza a normalização if (any(!is.na(valor\_original))) { valor\_normalizado <- normalize(valor\_original) # Obtém o p-value da tabela correspondente e ajusta para não ser menor que 0.01 p\_value <- p\_values\_com\_peso$P\_Value[p\_values\_com\_peso$Variavel == variavel] p\_value <- ifelse(p\_value < 0.01, 0.01, round(p\_value, 2)) # Obtém a correlação de Spearman entre a variável e a taxa de suicídio correlacao\_spearman <- correlation\_df %>% filter(Variavel == variavel) %>% pull(Taxa\_de\_suicidio\_por\_100k) # Calcula o coeficiente de multiplicação coeficiente\_multiplicacao <- correlacao\_spearman / p\_value # Armazena os resultados no DataFrame final resultados[[variavel]] <- valor\_normalizado resultados[[paste0("Coef\_Mult\_", variavel)]] <- coeficiente\_multiplicacao resultados[[paste0("Produto\_", variavel)]] <- valor\_normalizado \* coeficiente\_multiplicacao # Criar uma categorização para os valores normalizados resultados[[paste0(variavel, "\_Grupo")]] <- cut( valor\_normalizado, breaks = seq(0, 100, by = 25), include.lowest = TRUE, labels = c("Baixo 0-25", "Médio Baixo 25-50", "Médio Alto 50-75", "Alto 75-100") ) } else { warning(paste("A variável", variavel, "contém apenas valores NA.")) }

} else { warning(paste("A variável", variavel, "não está presente no dataset.")) } }

#Verifica se todas as colunas Produto\_ existem antes de calcular a soma total

produtos\_existentes <- paste0("Produto\_", variaveis\_interesse)

if (all(produtos\_existentes %in% names(resultados))) { resultados$Soma\_Total <- rowSums(resultados[, produtos\_existentes], na.rm = TRUE) } else { warning("Algumas colunas de produtos não existem no DataFrame.") }

#Normalizar a soma total caso existam valores válidos

if (!all(is.na(resultados$Soma\_Total))) { max\_soma\_total <- max(resultados$Soma\_Total, na.rm = TRUE) min\_soma\_total <- min(resultados$Soma\_Total, na.rm = TRUE)

resultados$Soma\_Total\_Normalizada <- (resultados$Soma\_Total - min\_soma\_total) / (max\_soma\_total - min\_soma\_total) \* 100

#Criar grupos para a soma total normalizada

resultados$Grupo <- cut( resultados$Soma\_Total\_Normalizada, breaks = seq(0, 100, by = 25), include.lowest = TRUE, labels = c("Baixo 0-25", "Médio Baixo 25-50", "Médio Alto 50-75", "Alto 75-100") ) } else { warning("A coluna Soma\_Total contém apenas valores NA.") }

#Exibir os primeiros resultados para conferir

print(head(resultados))

#Salvar os resultados atualizados em um arquivo CSV

write.csv(resultados, file = "resultados\_atualizados.csv", row.names = FALSE)