

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Weberton Rodrigues Costa

**Aplicação de *Machine Learning* no
Processo de Avaliação de Crédito**

Belo Horizonte
2022

Weberton Rodrigues Costa

**Aplicação de *Machine Learning* no
Processo de Avaliação de Crédito**

Trabalho de Conclusão de Curso apresentado
ao Curso de Especialização em Ciência de
Dados e Big Data como requisito parcial à
obtenção do título de especialista.

Belo Horizonte
2022

SUMÁRIO

1. Introdução	4
1.1. Contextualização	4
1.2. O problema proposto	5
2. Coleta de Dados	7
3. Processamento/Tratamento de Dados	14
4. Análise e Exploração dos Dados	23
5. Criação de Modelos de Machine Learning.....	30
7. Apresentação dos Resultados	40
8. Links	42
REFERÊNCIAS	43
Bibliografia.....	43
APÊNDICE	44

1. Introdução

1.1. Contextualização

O setor financeiro, assim como muitos outros segmentos, tem experimentado um expressivo crescimento da quantidade de dados disponíveis para análise e auxílio na tomada de decisão. Nesse contexto a utilização de técnicas para tratamento de dados e a extração de conhecimento deles se mostra fundamental para as empresas.

A avaliação do risco associado à concessão de crédito a um determinado cliente é um ponto central e complexo nas instituições financeiras. Nesse processo são avaliados os chamados C's do crédito:

- **Capacidade:** refere-se à capacidade do solicitante de arcar com os compromissos e efetivamente pagar o empréstimo;
- **Colateral:** refere-se às garantias oferecidas;
- **Caráter:** histórico financeiro do cliente;
- **Condições:** avaliar as perspectivas para o mercado em que está inserido o solicitante; e
- **Capital:** patrimônio líquido do solicitante.

Uma avaliação assertiva é fundamental para uma instituição financeira considerando o setor na qual está inserida em que a perenidade e a saúde financeira da empresa depende do efetivo retorno dos valores concedidos como crédito. Assim, as abordagens de análise que utilizam *machine learning* têm sido cada vez mais empregadas para avaliar os clientes, principalmente, como uma resposta aos ambientes dinâmicos, nos quais decisões devem ser tomadas rapidamente.

Machine Learning ou Aprendizado de Máquina é uma subárea de Inteligência artificial que se ocupa com o estudo de algoritmos de computacionais que podem melhorar automaticamente por meio da experiência e do uso de dados. Os algoritmos de aprendizado de máquina constroem um modelo com base em dados para fazer previsões ou decisões sem serem explicitamente programados para isso.

No setor financeiro além da utilização de *Machine Learning* na parte de análise de crédito há diversas outras aplicações como:

- **Atendimento ao cliente;**
- **Detecção e prevenção de fraudes; e**
- **Automação de operações.**

1.2. O problema proposto

Esse trabalho de conclusão de curso se propõe a empregar modelos de *Machine Learning* no contexto instituições financeiras com dois conjuntos de dados: um relativo a avaliação de aprovação de empréstimo e outro relativo a avaliação da aprovação de cartão de crédito.

Conforme mencionado acima, a avaliação de crédito de crédito do cliente é uma tarefa central no setor financeiro e com a disponibilização cada vez maior de dados e ferramentas para o seu tratamento é de se esperar um crescimento da aplicação de *Machine Learning* para essa tarefa.

Aqui se utilizará dois conjuntos de dados disponíveis no repositório da UCI (University of California Irvine): o primeiro possui dados sobre clientes de um banco Alemão e a avaliação do crédito solicitado como bom ou ruim, o segundo possui dados de clientes e avaliação da aprovação de cartão de crédito de um Banco Australiano.

Esse trabalho tem como objetivo o emprego de três algoritmos de *Machine Learning* com abordagens diferentes no contexto da aprovação de crédito, problema central de instituições financeiras, e a comparação do desempenho dos algoritmos empregados.

Para o desenvolvimento desse trabalho utilizamos a linguagem programação *Python* Versão 3.7.6 e o ambiente de *Jupyter Notebook* com as bibliotecas abaixo.

```
# importação das bibliotecas
import pandas as pd

import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold, cross_val_score

import statistics

from scipy.stats import randint

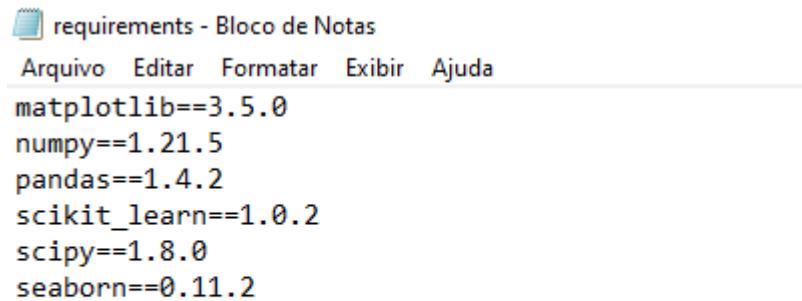
import matplotlib.pyplot as plt

import numpy as np

import warnings
warnings.filterwarnings("ignore")
```

Figura 1 - Bibliotecas utilizadas

Usamos o pacote *pipreqs* para gerar o arquivo *requirements.txt* com as versões das bibliotecas utilizadas.



```
requirements - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
matplotlib==3.5.0
numpy==1.21.5
pandas==1.4.2
scikit_learn==1.0.2
scipy==1.8.0
seaborn==0.11.2
```

Figura 2 - Requirements.txt

2. Coleta de Dados

Nesse trabalho optou-se pela utilização de dois conjuntos de dados disponíveis no repositório público da *UCI (University of California Irvine)*.

A descrição dos conjuntos de dados utilizados segue abaixo.

Conjunto de dados 1 - *South German Credit Data Set*

Os dados são de uma amostra com 1 mil classificações de crédito (entre “bom” e “ruim”) coletados entre 1973 e 1975 de um grande banco do sul da Alemanha. O conjunto de dados originalmente foi doado pelo professor alemão *Hans Hofmann* em 1994 para o repositório da UCI.

Entretanto, a Profa. Dra. *Ulrike Grömping* da Universidade de Ciências Aplicadas de Berlim verificou algumas inconsistências nos dados disponibilizados e promoveu a correção dos mesmos. Disponibilizando novamente no repositório da UCI.

O conjunto de dados está disponível no link: [South German Credit](#) e o artigo com as correções promovidas está disponível em: [South German Credit Data: Correcting a Widely Used Data Set.](#)

O *dataset* corresponde a um conjunto de variáveis descrevendo aspectos do proponente de um empréstimo e as características da solicitação. Temos, além da variável objetivo *risco_credito* que utiliza 0 para risco bom ou 1 para risco ruim, 20 atributos sendo 3 variáveis numéricas e 17 variáveis categóricas.

UCI 

Machine Learning Repository
Center for Machine Learning and Intelligent Systems

About Citation Policy Donate a Data Set Contact
 Repository Web Search

[View ALL Data Sets](#)

Check out the [beta version](#) of the new UCI Machine Learning Repository we are currently testing! [Contact us](#) if you have any issues, questions, or concerns. [Click here to try out the new site.](#) X

South German Credit (UPDATE) Data Set

[Download](#) [Data Folder](#) [Data Set Description](#)

Abstract: 700 good and 300 bad credits with 20 predictor variables. Data from 1973 to 1975. Stratified sample from actual credits with bad credits heavily oversampled. A cost matrix can be used.

Data Set Characteristics:	Multivariate	Number of Instances:	1000	Area:	Business
Attribute Characteristics:	Integer, Real	Number of Attributes:	21	Date Donated	2020-06-20
Associated Tasks:	Classification, Regression, Clustering	Missing Values?	N/A	Number of Web Hits:	42336

Source:

Ulrike Groemping

Beuth University of Applied Sciences Berlin

Website with contact information: <https://prof.beuth-hochschule.de/groemping/>

Figura 3 - Data Set: South German Credit - Disponível em:

<https://archive.ics.uci.edu/ml/datasets/South+German+Credit+%28UPDATE%29>

Tabela 1: Variáveis numéricas

Nome da coluna/campo Original	Nome da coluna/campo após renomear	Descrição
laufzeit	duracao	Duração do empréstimo em meses.
hoehe	montante	O valor solicitado do empréstimo em DM (Deutsche Mark - Marco alemão).
alter	idade	Idade do proponente em anos.

Tabela 2: Variáveis categóricas

Nome da coluna/campo Original	Nome da coluna/campo após renomear	Descrição
laufkont	status_conta_corrente	Situação da conta do proponente com o Banco. 1 = Sem conta corrente; 2 = Saldo menor que 0 (zero) DM. 3 = Saldo maior ou igual a zero e menor que 200 DM.

		4 = Saldo maior que 200 DM.
moral	historico_credito	<p>Histórico de crédito do proponente.</p> <p>0 = com pagamento em atraso no passado.</p> <p>1 = Conta crítica/ outros empréstimos em outros lugares.</p> <p>2 = Sem empréstimos tomados/todos os empréstimos pagos em dia.</p> <p>3 = Empréstimos ativos pagos em dia até agora.</p> <p>4 = Todos os empréstimos no banco pagos em dia.</p>
verw	objetivo_proposta	<p>Destinação do crédito pleiteado.</p> <p>0 = Outros.</p> <p>1 = Carro novo.</p> <p>2 = Carro usado.</p> <p>3 = Móveis/Equipamentos.</p> <p>4 = Rádio/Televisão.</p> <p>5 = Aparelhos domésticos.</p> <p>6 = Reparos.</p> <p>7 = Educação.</p> <p>8 = Férias.</p> <p>9 = Treinamento.</p> <p>10 = Negócios.</p>
sparkont	poupanca	Valor que o proponente possui em poupança/títulos.

		<p>1 = Desconhecido/ Não possui.</p> <p>2 = Menor que 100 DM.</p> <p>3 = Maior ou igual a 100 e menor que 500 DM.</p> <p>4 = Maior ou igual a 500 e menor que 1000 DM.</p> <p>5 = Maior ou igual a 1000 DM.</p>
beszeit	tempo_emprego	<p>Tempo em que o proponente está no emprego atual.</p> <p>1 = Sem emprego.</p> <p>2 = Menos que 1 ano.</p> <p>3 = Maior ou igual a 1 ano e menor que 4.</p> <p>4 = Maior ou igual a 4 e menor que 7 anos.</p> <p>5 = Maior ou igual a 7 anos.</p>
rate	percentual_renda	<p>Percentual da renda do proponente que seria comprometida com a parcela do empréstimo.</p> <p>1 = Maior ou igual a 35%;</p> <p>2 = Maior ou igual a 25% e menor que 35%.</p> <p>3 = Maior ou igual a 20% e menor que 25%.</p> <p>4 = Menor que 20%.</p>
famges	sexo_estado_civil	<p>Sexo e estado civil.</p> <p>1 = Homem: divorciado/separado.</p>

		<p>2 = Mulher: não solteira ou Homem: solteiro.</p> <p>3 = Homem: casado/viúvo.</p> <p>4 = Mulher: solteira.</p>
buerge	garantia	<p>Garantia do empréstimo.</p> <p>1 = Sem garantia.</p> <p>2 = Co-requerente.</p> <p>3 = Fiador.</p>
wohnzeit	tempo_residencia	<p>A quanto tempo do proponente mora em sua atual residência.</p> <p>1 = Menos que 1 ano.</p> <p>2 = Maior ou igual a 1 anos e menor que 4 anos.</p> <p>3 = Maior ou igual a 4 anos e menor que 7 anos.</p> <p>4 = Maior ou igual a 7 anos.</p>
verm	propriedade	<p>Propriedades em nome do proponente.</p> <p>1 = Desconhecido/Sem propriedades.</p> <p>2 = Carro ou outro.</p> <p>3 = Contrato de poupança com uma sociedade de construção/Seguro de vida.</p> <p>4 = Imóveis.</p>
weitkred	outro_fornecedor_credito	Outros fornecedores de crédito que não o Banco.

		<p>1 = Banco.</p> <p>2 = Lojas.</p> <p>3 = Nenhum.</p>
wohn	tipo_residencia	<p>O tipo da residência do proponente.</p> <p>1 = Grátis (sem pagamento de aluguel).</p> <p>2 = Alugada.</p> <p>3 = Própria.</p>
bishkred	num_credito_tomado	<p>Número de empréstimos que o proponente possui no Banco.</p> <p>1 = Um.</p> <p>2 = De 2 a 3.</p> <p>3 = De 4 a 5.</p> <p>4 = Maior ou igual a 6.</p>
beruf	qualificacao_emprego	<p>Qualificação do emprego do proponente.</p> <p>1 = Desempregado.</p> <p>2 = Sem qualificação.</p> <p>3 = Com qualificação.</p> <p>4 = Altamente qualificado.</p>
pers	dependentes	<p>Número de dependentes do proponente.</p> <p>1 = 3 ou mais.</p> <p>2 = 0 até 2.</p>
telef	tem_telefone	<p>O proponente tem linha telefônica em seu nome?</p>

		1 = Não. 2 = Sim.
Gastarb	empregado_estrangeiro	O proponente é trabalhador estrangeiro? 1 = Sim. 2 = Não.

Conjunto de Dados 2: *Statlog (Australian Credit Approval) Data Set*

Este conjunto de dados diz respeito a aprovação ou não de solicitações de cartão de crédito. Todos os nomes e valores originais foram disponibilizados com alterações para símbolos para fins de confidencialidade dos dados.

O conjunto de dados está disponível no seguinte link: [Australian Credit Approval](#).

O conjunto de dados possui 14 atributos sendo 6 numéricos e 8 categóricos. Havia valores ausentes, sendo que em 37 casos (5%) tinha-se um ou mais valores ausentes que foram substituídos pela moda do atributo, no caso de atributos categóricos, e pela média, no caso de atributo numérico.



Check out the [beta version](#) of the new UCI Machine Learning Repository we are currently testing! [Contact us](#) if you have any issues, questions, or concerns. [Click here to try out the new site.](#)

Statlog (Australian Credit Approval) Data Set

[Download](#) | [Data Folder](#) | [Data Set Description](#)

Abstract: This file concerns credit card applications. This database exists elsewhere in the repository (Credit Screening Database) in a slightly different form

Data Set Characteristics:	Multivariate	Number of Instances:	690	Area:	Financial
Attribute Characteristics:	Categorical, Integer, Real	Number of Attributes:	14	Date Donated	N/A
Associated Tasks:	Classification	Missing Values?	Yes	Number of Web Hits:	201123

Source:

(confidential)

Submitted by quinlan.%40.cs.su.oz.au

Figura 4 - Data Set: Australian Credit Approval - Disponível em:
<https://archive.ics.uci.edu/ml/datasets/Statlog+Australian+Credit+Approval%29>

Tabela 3: Variáveis numéricas

Nome da coluna/campo
col2
col3
col7
col10
col13
col14

Tabela 4: Variáveis categóricas

Nome da coluna/campo
col1
col4
col5
col6
col8
col9
col11
col12

3. Processamento/Tratamento de Dados

Conjunto de dados 1 - *South German Credit Data Set*

O dataset *alemao_df* apresenta 1000 linhas e 21 colunas:

```
alemao_df = pd.read_csv("SouthGermanCredit.asc", sep=' ')
alemao_df
```

	laufkont	laufzeit	moral	verw	hoehe	sparkont	beszeit	rate	famges	buerge	...	verm	alter	weitkred	wohn	bishkred	beruf	pers	telef	gastarb	kr
0	1	18	4	2	1049	1	2	4	2	1	...	2	21	3	1	1	3	2	1	2	
1	1	9	4	0	2799	1	3	2	3	1	...	1	36	3	1	2	3	1	1	2	
2	2	12	2	9	841	2	4	2	2	1	...	1	23	3	1	1	2	2	1	2	
3	1	12	4	0	2122	1	3	3	3	1	...	1	39	3	1	2	2	1	1	1	
4	1	12	4	0	2171	1	3	4	3	1	...	2	38	1	2	2	2	1	1	1	
...	
995	1	24	2	3	1987	1	3	2	3	1	...	1	21	3	1	1	2	1	1	2	
996	1	24	2	0	2303	1	5	4	3	2	...	1	45	3	2	1	3	2	1	2	
997	4	21	4	0	12680	5	5	4	3	1	...	4	30	3	3	1	4	2	2	2	
998	2	12	2	3	6468	5	1	2	3	1	...	4	52	3	2	1	4	2	2	2	
999	1	30	2	2	6350	5	5	4	3	1	...	2	31	3	2	1	3	2	1	2	

1000 rows × 21 columns

Figura 5 - Conjunto de dados alemao_df

As colunas estão com os nomes originais em alemão, então renomeamos para português.

```
renomea = {"laufkont" : "status_conta_corrente", "laufzeit" : "duracao", "moral" : "historico_credito", "verw" : "objetivo_proposta",
           "hoehe" : "montante", "sparkont" : "poupanca", "beszeit" : "tempo_emprego", "rate" : "percentual_renda",
           "famges" : "sexo_estado_civil", "buerge" : "garantia", "wohnzeit" : "tempo_residencia", "verm" : "propriedade",
           "alter" : "idade", "weitkred" : "outro_fornecedor_credito", "wohn" : "tipo_residencia", "bishkred" : "num_credito_total",
           "beruf" : "qualificacao_emprego", "pers" : "dependentes", "telef" : "tem_telefone", "gastarb" : "empregado_estrangeiro",
           "kredit" : "risco_credito"}
alemao_df.rename(columns=renomea, inplace=True)
alemao_df
```

	status_conta_corrente	duracao	historico_credito	objetivo_proposta	montante	poupanca	tempo_emprego	percentual_renda	sexo_estado_civil	garantia
0	1	18	4	2	1049	1	2	4	2	1	...	2	4	2	1	2	1	1	2	1	
1	1	9	4	0	2799	1	3	2	3	1	...	1	3	2	1	3	1	1	2	1	
2	2	12	2	9	841	2	4	2	2	1	...	1	2	2	1	2	2	1	1	2	
3	1	12	4	0	2122	1	3	3	3	1	...	1	3	3	1	3	3	1	1	1	
4	1	12	4	0	2171	1	3	4	3	1	...	2	3	1	3	4	3	1	1	1	
...	
995	1	24	2	3	1987	1	3	2	3	1	...	1	3	2	1	3	1	1	2	1	
996	1	24	2	0	2303	1	5	4	3	2	...	1	4	3	2	1	3	2	1	2	
997	4	21	4	0	12680	5	5	4	3	1	...	4	3	4	4	3	1	4	3	1	
998	2	12	2	3	6468	5	1	2	3	1	...	4	2	2	3	1	4	2	2	2	
999	1	30	2	2	6350	5	5	4	3	1	...	2	3	1	4	3	1	1	2	1	

1000 rows × 21 columns

Figura 6 - Renomeação do cabeçalho das colunas do conjunto de dados alemao_df

Em seguida verificamos as informações do *DataFrame* (índices, colunas, valores não nulos, os tipos de dados e a utilização de memória).

```
alemao_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   status_conta_corrente    1000 non-null   int64  
 1   duracao                  1000 non-null   int64  
 2   historico_credito        1000 non-null   int64  
 3   objetivo_proposta        1000 non-null   int64  
 4   montante                 1000 non-null   int64  
 5   poupanca                 1000 non-null   int64  
 6   tempo_emprego            1000 non-null   int64  
 7   percentual_renda         1000 non-null   int64  
 8   sexo_estado_civil        1000 non-null   int64  
 9   garantia                 1000 non-null   int64  
 10  tempo_residencia         1000 non-null   int64  
 11  propriedade               1000 non-null   int64  
 12  idade                     1000 non-null   int64  
 13  outro_fornecedor_credito 1000 non-null   int64  
 14  tipo_residencia          1000 non-null   int64  
 15  num_credito_tomado       1000 non-null   int64  
 16  qualificacao_emprego     1000 non-null   int64  
 17  dependentes               1000 non-null   int64  
 18  tem_telefone              1000 non-null   int64  
 19  empregado_estrangeiro    1000 non-null   int64  
 20  risco_credito             1000 non-null   int64  
dtypes: int64(21)
memory usage: 164.2 KB
```

Figura 7 - Informações sobre o conjunto de dados alemao_df

O dataset não possui valores ausentes ou duplicados e as inconsistências nos dados foram corrigidas pela Profa. Dra. *Ulrike Grömping* da Universidade de Ciências Aplicadas de Berlim, conforme disponível em: [South German Credit Data: Correcting a Widely Used Data Set](#). Todas os atributos, inclusive os categóricos, estão representados por números e são do tipo *int64*.

A maioria algoritmos de *Machine Learning* não conseguem trabalhar tendo com entrada variáveis categóricas, sendo necessário algum tipo de transformação nesse tipo de dado.

Uma opção é utilizar o *LabelEncoding* que transforma cada variável em um valor numérico. Entretanto com a utilização dessa abordagem é possível que surja outros problemas uma vez que utilizando de uma sequência de números induz-se o algoritmo a trabalhar como se existe uma hierarquia/ordem entre as categorias.

Uma outra opção é utilizar o *One-Hot Encoder* que representa as varáveis categóricas em um array binário. Apesar dessa abordagem eliminar o problema de hierarquia/ordem ela

tem a desvantagem de adicionar mais colunas do *dataset*, especialmente nos casos em que há muitos valores diferentes dentro de uma categoria.

No caso optou-se pela utilização do *One-Hot Encoder*. O *SkLearn* disponibiliza a função *OneHotEncoder* que implementa essa transformação e foi utilizada nesse trabalho, conforme abaixo.

```
categoricas = ["status_conta_corrente", "historico_credito", "objetivo_proposta", "poupanca",
               "sexo_estado_civil", "garantia", "outro_fornecedor_credito", "tipo_residencia", "propriedade",
               "qualificacao_emprego", "tempo_emprego", "empregado_estrangeiro", "percentual_renda", "tempo_residencia"]

one_hot = OneHotEncoder(handle_unknown='ignore', categories='auto', drop='first', sparse=False)

cat_df = pd.DataFrame(one_hot.fit_transform(alemao_df[categoricas]), columns = one_hot.get_feature_names_out(categoricas))
cat_df.head()

< IPython.core.display.HTML>

```

	status_conta_corrente_2	status_conta_corrente_3	status_conta_corrente_4	historico_credito_1	historico_credito_2	historico_credito_3	historico_credito_4
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	1.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 50 columns

Figura 8 - Aplicação do *OneHotEncoder* no conjunto de dados *alemao_df*

Após essa transformação nos dados, retiramos as variáveis categóricas originais do *dataframe alemao_df* e incluímos as novas variáveis no *dataframe alemao_df_copia* juntamente com as variáveis numéricas.

```
alemao_df_copia = alemao_df.drop(categoricas, axis=1)
alemao_df_copia = pd.concat([cat_df, alemao_df_copia], axis=1)
alemao_df_copia

< IPython.core.display.HTML>
```

	status_conta_corrente_2	status_conta_corrente_3	status_conta_corrente_4	historico_credito_1	historico_credito_2	historico_credito_3	historico_credito_4
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	1.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0
...
995	0.0	0.0	0.0	0.0	1.0	0.0	0.0
996	0.0	0.0	0.0	0.0	1.0	0.0	0.0
997	0.0	0.0	1.0	0.0	0.0	0.0	1.0
998	1.0	0.0	0.0	0.0	1.0	0.0	0.0
999	0.0	0.0	0.0	0.0	1.0	0.0	0.0

1000 rows × 55 columns

Figura 9 - Exclusão das variáveis originais e inclusão das variáveis após a transformação *OneHotEncoder*

No caso das variáveis numéricas para o melhor desempenho dos modelos de *machine learning* é necessário que os dados estejam na mesma escala. Para isso empregou-se a função *MinMaxScaler* do *SkLearn* que transforma os dados conforme equação abaixo:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Onde x_{scaled} representa o valor na nova escala, x_{max} o maior valor observado do atributo e x_{min} o menor observado da atributo. Obtendo assim os valores no intervalo entre 0 e 1. Abaixo segue o código utilizado.

```
numericas = ["duracao", "montante", "idade"]
mms = MinMaxScaler()

num_df = pd.DataFrame(mms.fit_transform(alemao_df[numericas]), columns = mms.get_feature_names_out(numericas))
num_df.head()
```

	duracao	montante	idade
0	0.205882	0.043964	0.035714
1	0.073529	0.140255	0.303571
2	0.117647	0.032519	0.071429
3	0.117647	0.103004	0.357143
4	0.117647	0.105700	0.339286

Figura 10 - Aplicação do *MinMaxScaler* no conjunto de dados alemao_df

Em seguida retiramos as variáveis numéricas originais do *dataframe alemao_df_copia* e armazenamos os novos valores no mesmo *dataframe*.

```
alemao_df_copia = alemao_df_copia.drop(numericas, axis=1)
alemao_df_copia = pd.concat([num_df, alemao_df_copia], axis=1)
alemao_df_copia
```

	duracao	montante	idade	tem_telefone	risco_credito	status_conta_corrente_2	status_conta_corrente_3	status_conta_corrente_4	historico_credito_1
0	0.205882	0.043964	0.035714	1	1	0.0	0.0	0.0	0.0
1	0.073529	0.140255	0.303571	1	1	0.0	0.0	0.0	0.0
2	0.117647	0.032519	0.071429	1	1	1.0	0.0	0.0	0.0
3	0.117647	0.103004	0.357143	1	1	0.0	0.0	0.0	0.0
4	0.117647	0.105700	0.339286	1	1	0.0	0.0	0.0	0.0
...
995	0.294118	0.095576	0.035714	1	0	0.0	0.0	0.0	0.0
996	0.294118	0.112964	0.464286	1	0	0.0	0.0	0.0	0.0
997	0.250000	0.683944	0.196429	2	0	0.0	0.0	1.0	0.0
998	0.117647	0.342137	0.589286	2	0	1.0	0.0	0.0	0.0
999	0.382353	0.335644	0.214286	1	0	0.0	0.0	0.0	0.0

1000 rows × 55 columns

Figura 11 - Exclusão das variáveis originais e inclusão das variáveis após a transformação *MinMaxScaler*

Conjunto de dados 2: Statlog (Australian Credit Approval) Data Set

O dataset *australiano_df* apresenta 690 linhas e 15 colunas:

```
australiano_df = pd.read_csv("australian.dat", sep=' ', header=None)
australiano_df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	22.08	11.460	2	4	4	1.585	0	0	0	1	2	100	1213	0
1	0	22.67	7.000	2	8	4	0.165	0	0	0	0	2	160	1	0
2	0	29.58	1.750	1	4	4	1.250	0	0	0	1	2	280	1	0
3	0	21.67	11.500	1	5	3	0.000	1	1	11	1	2	0	1	1
4	1	20.17	8.170	2	6	4	1.960	1	1	14	0	2	60	159	1
...
685	1	31.57	10.500	2	14	4	6.500	1	0	0	0	2	0	1	1
686	1	20.67	0.415	2	8	4	0.125	0	0	0	0	2	0	45	0
687	0	18.83	9.540	2	6	4	0.085	1	0	0	0	2	100	1	1
688	0	27.42	14.500	2	14	8	3.085	1	1	1	0	2	120	12	1
689	1	41.00	0.040	2	10	4	0.040	0	1	1	0	1	560	1	1

690 rows × 15 columns

Figura 12 - Conjunto de dados *australiano_df*

As colunas não apresentam nomes então as nomeamos conforme abaixo.

```
australiano_df.rename(columns={0:"col1", 1:"col2", 2:"col3",
                               3:"col4", 4:"col5", 5:"col6",
                               6:"col7", 7:"col8", 8:"col9",
                               9:"col10", 10:"col11", 11:"col12",
                               12:"col13", 13:"col14", 14:"col15"}, inplace=True)
australiano_df.head()
```

	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11	col12	col13	col14	col15
0	1	22.08	11.46	2	4	4	1.585	0	0	0	1	2	100	1213	0
1	0	22.67	7.00	2	8	4	0.165	0	0	0	0	2	160	1	0
2	0	29.58	1.75	1	4	4	1.250	0	0	0	1	2	280	1	0
3	0	21.67	11.50	1	5	3	0.000	1	1	11	1	2	0	1	1
4	1	20.17	8.17	2	6	4	1.960	1	1	14	0	2	60	159	1

Figura 13 - Nomeação do cabeçalho das colunas do conjunto de dados *australiano_df*

Em seguida verificamos as informações do *DataFrame* (índices, colunas, valores não nulos, os tipos de dados e a utilização de memória).

```
australiano_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   col1    690 non-null    int64  
 1   col2    690 non-null    float64 
 2   col3    690 non-null    float64 
 3   col4    690 non-null    int64  
 4   col5    690 non-null    int64  
 5   col6    690 non-null    int64  
 6   col7    690 non-null    float64 
 7   col8    690 non-null    int64  
 8   col9    690 non-null    int64  
 9   col10   690 non-null    int64  
 10  col11   690 non-null    int64  
 11  col12   690 non-null    int64  
 12  col13   690 non-null    int64  
 13  col14   690 non-null    int64  
 14  col15   690 non-null    int64  
dtypes: float64(3), int64(12)
memory usage: 81.0 KB
```

Figura 14 - Informações sobre o conjunto de dados *australiano_df*

O conjunto de dados não possui valores ausentes ou duplicados. Originalmente havia valores ausentes, sendo que em 37 casos (5%) tinha-se um ou mais valores ausentes que foram substituídos pela moda do atributo, no caso de atributos categóricos, e pela média, no caso de atributo numérico. Tantos os valores numéricos como os categóricos estão representados por números e são dos tipos *int64* e *float64*.

Empregamos a função *OneHotEncoder* do *SkLearn* para tratamento dos atributos categóricos conforme abaixo.

```
categoricas = ["col1", "col4", "col5", "col6", "col8", "col9", "col11", "col12"]
one_hot = OneHotEncoder(handle_unknown='ignore', categories='auto', drop='first', sparse=False)
cat_df = pd.DataFrame(one_hot.fit_transform(australiano_df[categoricas]), columns = one_hot.get_feature_names_out(categoricas))
cat_df.head()
```

	col1_1	col4_2	col4_3	col5_2	col5_3	col5_4	col5_5	col5_6	col5_7	col5_8	...	col6_4	col6_5	col6_7	col6_8	col6_9	col8_1	col9_1	col11_1	col12_1
0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.
4	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.

5 rows × 28 columns

Figura 15 - Aplicação do *OneHotEncoder* no conjunto de dados *australiano_df*

Após essa transformação nos dados retiramos as variáveis categóricas originais do *dataframe australiano_df* e incluímos as novas variáveis no *dataframe australiano_df_copia* juntamente com as variáveis numéricas.

```
australiano_df_copia = australiano_df.drop(categoricas, axis=1)
australiano_df_copia = pd.concat([cat_df, australiano_df_copia], axis=1)
australiano_df_copia
```

	col1_1	col4_2	col4_3	col5_2	col5_3	col5_4	col5_5	col5_6	col5_7	col5_8	...	col11_1	col12_2	col12_3	col2	col3	col7	col10	col13	col14
0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	1.0	0.0	22.08	11.460	1.585	0	100	1213
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	1.0	0.0	22.67	7.000	0.165	0	160	1
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	1.0	0.0	29.58	1.750	1.250	0	280	1
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	1.0	1.0	0.0	21.67	11.500	0.000	11	0	1
4	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	20.17	8.170	1.960	14	60	159
...	
685	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	31.57	10.500	6.500	0	0	1
686	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	1.0	0.0	20.67	0.415	0.125	0	0	45
687	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	18.83	9.540	0.085	0	100	1
688	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	27.42	14.500	3.085	1	120	12
689	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	41.00	0.040	0.040	1	560	1

690 rows × 35 columns

< >

Figura 16 - Exclusão das variáveis originais e inclusão das variáveis após a transformação OneHotEncoder

Para normalização dos valores dos atributos numéricos empregamos a função *MinMaxScaler* do *SkLearn* conforme abaixo.

```
numericas = ["col2", "col3", "col7", "col10", "col13", "col14"]
mms = MinMaxScaler()
num_df = pd.DataFrame(mms.fit_transform(australiano_df[numericas]), columns = mms.get_feature_names_out(numericas))
num_df.head()
```

	col2	col3	col7	col10	col13	col14
0	0.125263	0.409286	0.055614	0.000000	0.05	0.01212
1	0.134135	0.250000	0.005789	0.000000	0.08	0.00000
2	0.238045	0.062500	0.043860	0.000000	0.14	0.00000
3	0.119098	0.410714	0.000000	0.164179	0.00	0.00000
4	0.096541	0.291786	0.068772	0.208955	0.03	0.00158

Figura 17 - Aplicação do MinMaxScaler no conjunto de dados australiano_df

Em seguida retiramos as variáveis numéricas originais do *dataframe australiano_df_copia* e armazenamos os novos valores no mesmo *dataframe*.

```

australiano_df_copia = australiano_df_copia.drop(numericas, axis=1)
australiano_df_copia = pd.concat([num_df, australiano_df_copia], axis=1)
australiano_df_copia

```

	col2	col3	col7	col10	col13	col14	col1_1	col4_2	col4_3	col5_2	...	col6_5	col6_7	col6_8	col6_9	col8_1	col9_1	col11_1	col
0	0.125263	0.409286	0.055614	0.000000	0.05	0.01212	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
1	0.134135	0.250000	0.005789	0.000000	0.08	0.00000	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.238045	0.062500	0.043860	0.000000	0.14	0.00000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
3	0.119098	0.410714	0.000000	0.164179	0.00	0.00000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
4	0.096541	0.291786	0.068772	0.208955	0.03	0.00158	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	1.0	0.0	
...	
685	0.267970	0.375000	0.228070	0.000000	0.00	0.00000	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
686	0.104060	0.014821	0.004386	0.000000	0.00	0.00044	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
687	0.076391	0.340714	0.002982	0.000000	0.05	0.00000	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
688	0.205564	0.517857	0.108246	0.014925	0.06	0.00011	0.0	1.0	0.0	0.0	...	0.0	0.0	1.0	0.0	1.0	1.0	0.0	
689	0.409774	0.001429	0.001404	0.014925	0.28	0.00000	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	

690 rows × 35 columns

Figura 18 - Exclusão das variáveis originais e inclusão das variáveis após a transformação *MinMaxScaler*

4. Análise e Exploração dos Dados

Conjunto de dados 1 - *South German Credit Data Set*

Verificamos o balanceamento entre as classes.

```
# Checando a proporção de registros por classe
bom = alemao_df.loc[alemao_df['risco_credito'] == 1]
ruim = alemao_df.loc[alemao_df['risco_credito'] == 0]
print("Temos", len(bom), "classificados como crédito bom e", len(ruim), "classificados como crédito ruim.")

Temos 700 classificados como crédito bom e 300 classificados como crédito ruim.
```

Figura 19 - Balanceamento de classes do conjunto *alemao_df*

A classe com risco de crédito “bom” apresenta 70% dos casos e o risco de crédito “ruim” 30%. Há desbalanceamento, mas pequeno entre as classes e deve afetar pouco o desempenho do modelo e por isso resolvemos não empregar algum dos métodos para tratamento de problemas com desbalanceamento.

Abaixo segue um resumo das informações estatísticas das colunas do *dataset* com o número de valores não nulos, valores mínimos e máximos, média, desvio padrão e os valores de quartis.

	count	mean	std	min	25%	50%	75%	max
status_conta_corrente	1000.0	2.577	1.257638	1.0	1.0	2.0	4.00	4.0
duracao	1000.0	20.903	12.058814	4.0	12.0	18.0	24.00	72.0
historico_credito	1000.0	2.545	1.083120	0.0	2.0	2.0	4.00	4.0
objetivo_proposta	1000.0	2.828	2.744439	0.0	1.0	2.0	3.00	10.0
montante	1000.0	3271.248	2822.751760	250.0	1365.5	2319.5	3972.25	18424.0
poupanca	1000.0	2.105	1.580023	1.0	1.0	1.0	3.00	5.0
tempo_emprego	1000.0	3.384	1.208306	1.0	3.0	3.0	5.00	5.0
percentual_renda	1000.0	2.973	1.118715	1.0	2.0	3.0	4.00	4.0
sexo_estado_civil	1000.0	2.682	0.708080	1.0	2.0	3.0	3.00	4.0
garantia	1000.0	1.145	0.477706	1.0	1.0	1.0	1.00	3.0
tempo_residencia	1000.0	2.845	1.103718	1.0	2.0	3.0	4.00	4.0
propriedade	1000.0	2.358	1.050209	1.0	1.0	2.0	3.00	4.0
idade	1000.0	35.542	11.352670	19.0	27.0	33.0	42.00	75.0
outro_fornecedor_credito	1000.0	2.675	0.705601	1.0	3.0	3.0	3.00	3.0
tipo_residencia	1000.0	1.928	0.530186	1.0	2.0	2.0	2.00	3.0
num_credito_tomado	1000.0	1.407	0.577654	1.0	1.0	1.0	2.00	4.0
qualificacao_emprego	1000.0	2.904	0.653614	1.0	3.0	3.0	3.00	4.0
dependentes	1000.0	1.845	0.362086	1.0	2.0	2.0	2.00	2.0
tem_telefone	1000.0	1.404	0.490943	1.0	1.0	1.0	2.00	2.0
empregado_estrangeiro	1000.0	1.963	0.188856	1.0	2.0	2.0	2.00	2.0
risco_credito	1000.0	0.700	0.458487	0.0	0.0	1.0	1.00	1.0

Figura 20 - Informações estatísticas do conjunto alemao_df

A matriz de correlação entre as variáveis:

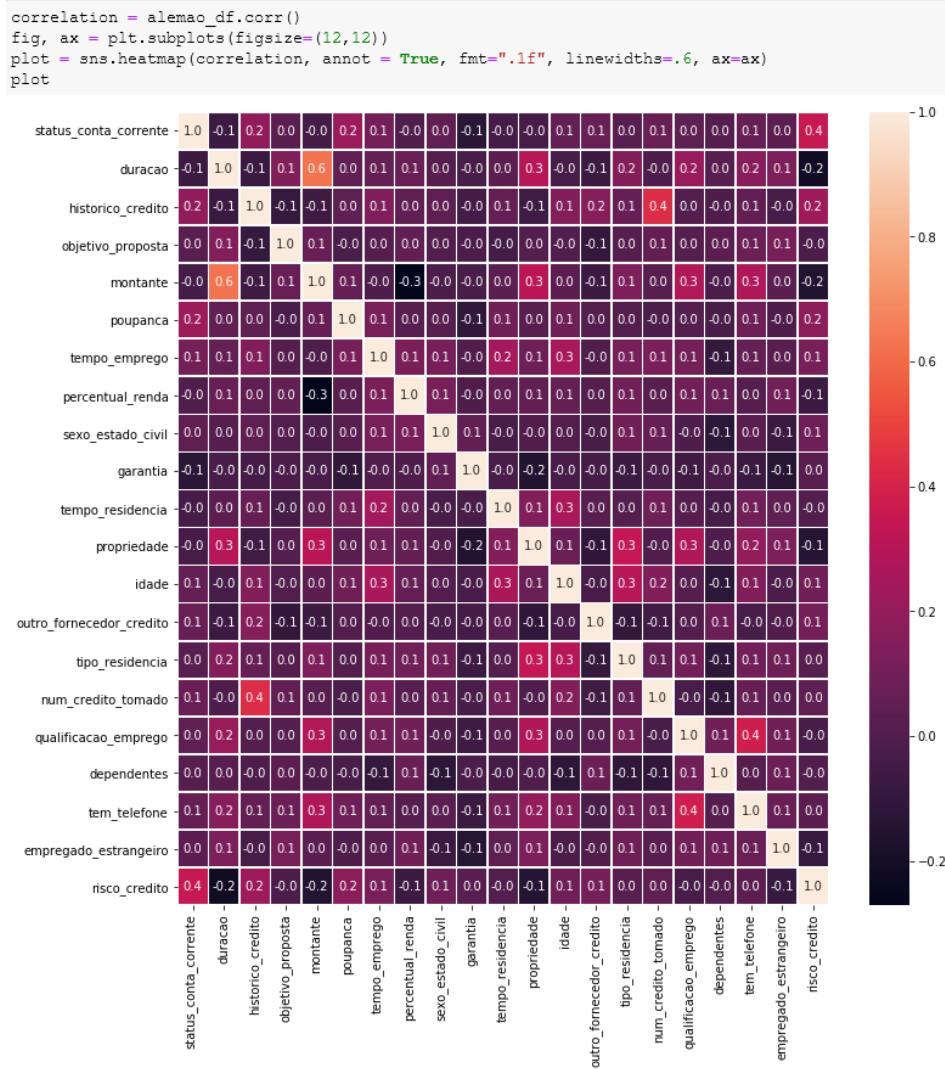
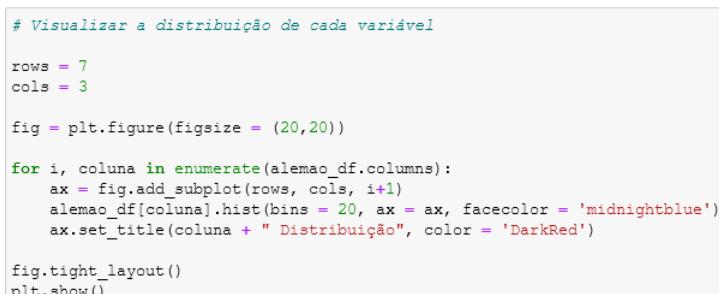


Figura 21 - Heatmap com os valores de correlação das variáveis do conjunto alemao_df

Como podemos pelo heatmap acima os atributos possuem correlação fraca, em geral abaixo de 0.5, exceto pelos atributos *montante* e *duracao* que possuem correlação de 0.6.

Plotamos em um gráfico a distribuição das variáveis.



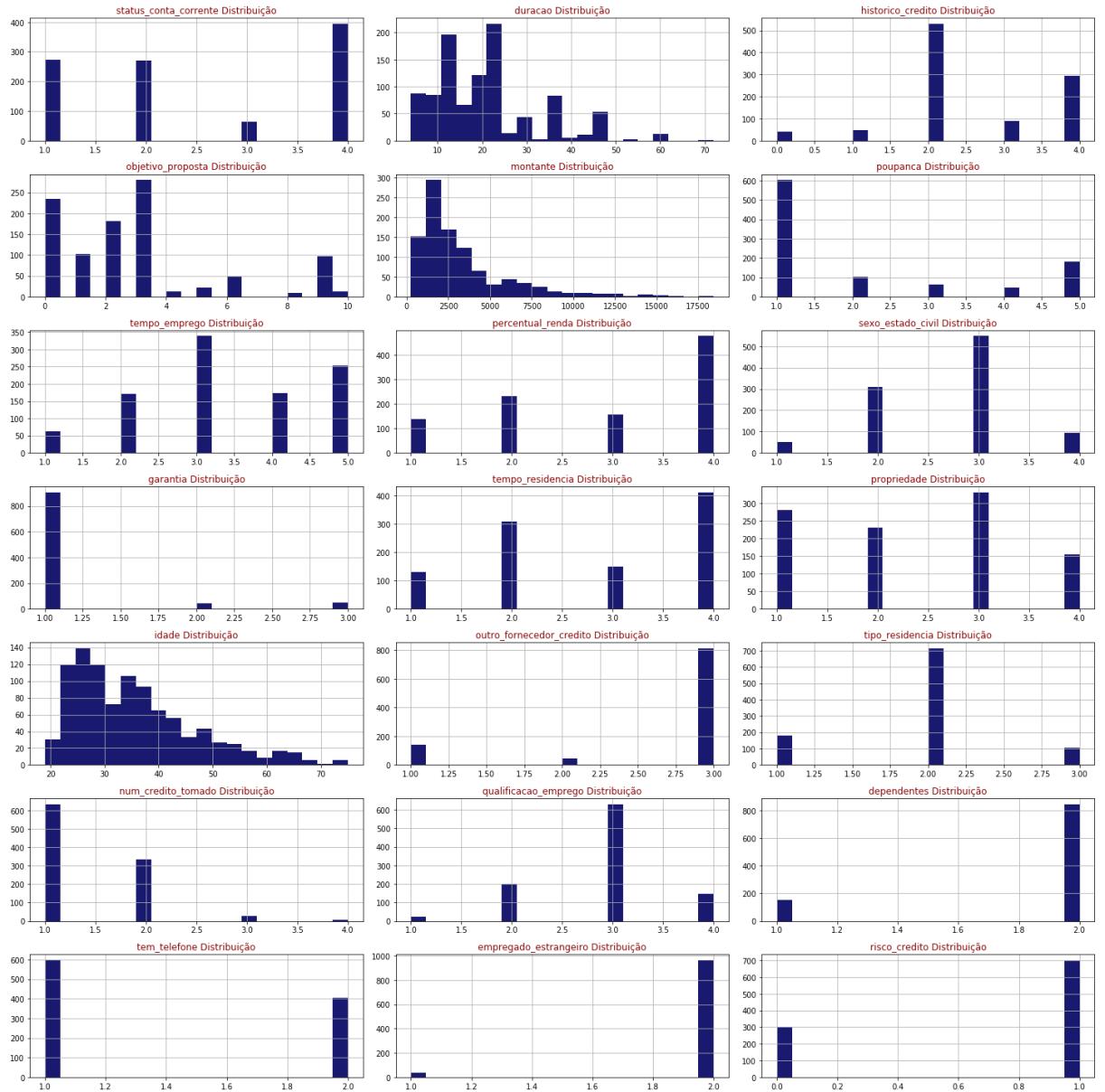


Figura 22 - Distribuição dos valores das variáveis do conjunto alemao_df

Conjunto de dados 2: Statlog (Australian Credit Approval) Data Set

Verificamos o balanceamento entre as classes.

```
# Checando a proporção de registros por classe
aprovado = australiano_df.loc[australiano_df['col15'] == 1]
nao_aprovado = australiano_df.loc[australiano_df['col15'] == 0]
print("Temos", len(aprovado), "solicitações de cartão de crédito aprovadas e", len(nao_aprovado),
      "solicitações de cartão de crédito não aprovadas.")
```

Temos 307 solicitações de cartão de crédito aprovadas e 383 solicitações de cartão de crédito não aprovadas.

Figura 23 - Balanceamento das classes no conjunto australiano_df

As classes estão平衡adas, com 44% representada pela classe 1 e 54% pela classe 0.

Abaixo segue um resumo das informações estatísticas das colunas do conjunto de dados com o número de valores não nulos, valores mínimos e máximos, média, desvio padrão e os valores de quartis.

	count	mean	std	min	25%	50%	75%	max
col1	690.0	0.678261	0.467482	0.00	0.000	1.000	1.0000	1.00
col2	690.0	31.568203	11.853273	13.75	22.670	28.625	37.7075	80.25
col3	690.0	4.758725	4.978163	0.00	1.000	2.750	7.2075	28.00
col4	690.0	1.766667	0.430063	1.00	2.000	2.000	2.0000	3.00
col5	690.0	7.372464	3.683265	1.00	4.000	8.000	10.0000	14.00
col6	690.0	4.692754	1.992316	1.00	4.000	4.000	5.0000	9.00
col7	690.0	2.223406	3.346513	0.00	0.165	1.000	2.6250	28.50
col8	690.0	0.523188	0.499824	0.00	0.000	1.000	1.0000	1.00
col9	690.0	0.427536	0.495080	0.00	0.000	0.000	1.0000	1.00
col10	690.0	2.400000	4.862940	0.00	0.000	0.000	3.0000	67.00
col11	690.0	0.457971	0.498592	0.00	0.000	0.000	1.0000	1.00
col12	690.0	1.928986	0.298813	1.00	2.000	2.000	2.0000	3.00
col13	690.0	184.014493	172.159274	0.00	80.000	160.000	272.0000	2000.00
col14	690.0	1018.385507	5210.102598	1.00	1.000	6.000	396.5000	100001.00
col15	690.0	0.444928	0.497318	0.00	0.000	0.000	1.0000	1.00

Figura 24 - Informações estatísticas do conjunto australiano_df

A matriz de correlação entre as variáveis:

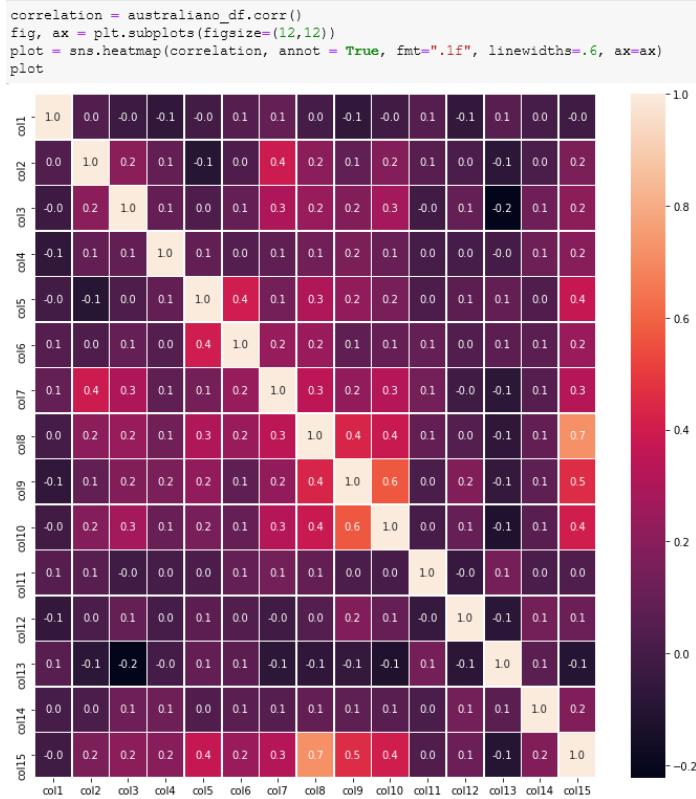


Figura 25 - Heatmap da correção entre as variáveis do conjunto australiano_df

Como podemos pelo *heatmap* acima os atributos possuem correlação fraca, em geral entre 0.0 a 0.5, exceto pelos atributos que apresentaram correlação moderada: *col9* e *col10* com 0.6 e *col8* e *col15* com 0.7.

Plotamos em um gráfico a distribuição das variáveis.

```

# Visualizar a distribuição de cada variável

rows = 7
cols = 3

fig = plt.figure(figsize = (20,20))

for i, coluna in enumerate(australiano_df.columns):
    ax = fig.add_subplot(rows, cols, i+1)
    australiano_df[coluna].hist(bins = 20, ax = ax, facecolor = 'midnightblue')
    ax.set_title(coluna + " Distribuição", color = 'DarkRed')

fig.tight_layout()
plt.show()

```

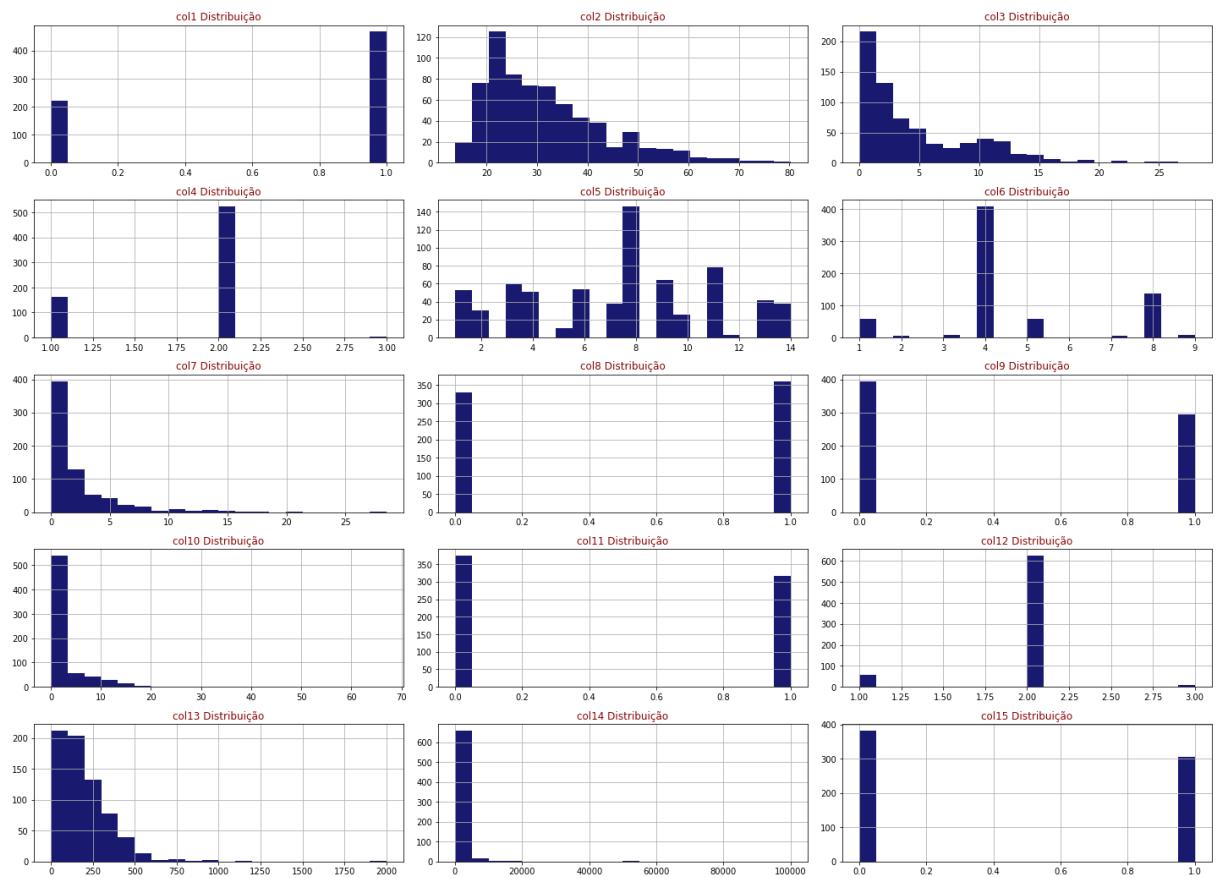


Figura 26 - Gráficos com a distribuição dos valores das variáveis do conjunto *australiano_df*

5. Criação de Modelos de *Machine Learning*

Utilizamos nesse trabalho para comparação três algoritmos de *Machine Learning*: *Random Forest* (Floresta Aleatória), *SVM* (*Support Vector Machine*) e Regressão Logística. Mais abaixo apresentamos o básico sobre o funcionamento dos três algoritmos.

Para busca e otimização dos hiperparâmetros utilizamos a função *RandomizedSearchCV* da *SkLearn* que aleatoriamente explora o espaço de hiperparâmetros de modo a encontrar a melhor configuração para o modelo. Enquanto a função *GridSearchCV* passa por todas as combinações de hiperparâmetros, o que torna a busca em grade computacionalmente muito cara, a *RandomizedSearchCV* passa por apenas um número fixo de configurações. Ele se move dentro da grade de maneira aleatória para encontrar os melhores valores reduzindo a computação desnecessária.

Utilizamos a abordagem de validação cruzada aninhada (*Nested Cross-Validation*) em conjunto com a função *RandomizedSearchCV* para busca no espaço de hiperparâmetros e avaliação dos modelos. A validação cruzada aninhada faz um melhor trabalho em garantir uma medida de generalização mais robusta, confiável e que é mais próxima de como o modelo se sairia em ambiente de produção. Na validação cruzada aninhada utiliza-se a validação cruzada interna para seleção dos hiperparâmetros e a validação cruzada externa para avaliar o desempenho do modelo preditivo.

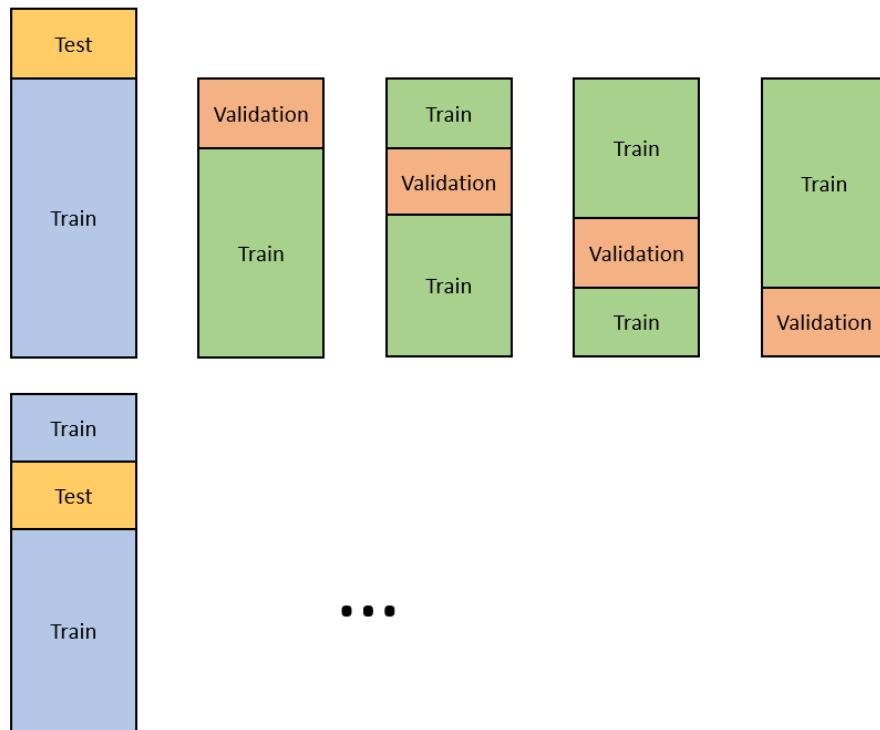


Figura 27 - Ilustração da validação cruzada aninhada (adaptado de (Castilla, 2021))

Na ilustração acima o loop externo é repetido 5 vezes gerando 5 diferentes conjuntos de testes e para cada iteração, o conjunto de treinamento externo será dividido em 4 partes.

Para implementar a validação cruzada aminha utilizando o *SkLearn* usamos um estimador da classe *RandomizedSearchCV* que efetua internamente uma validação cruzada e retorna o melhor estimador encontrado para a função *cross_val_score()*.

O que ocorre é que *cross_val_score()* funciona como uma validação cruzada comum porém ao invés de ajustar um único modelo aos $(n-1)$ folds separados para treino ela chama o objeto *RandomizedSearchCV* que usa esses $(n-1)$ folds para realizar uma outra validação cruzada tradicional, criando internamente seus próprio k-folds, validando o modelo e retornando o de menor erro.

Árvore de decisão

Uma árvore de decisão usa a estratégia dividir para conquistar de modo a resolver um problema de decisão. Um problema complexo é dividido em problemas mais simples, aos quais recursivamente é aplicada a mesma estratégia. As soluções dos subproblemas podem ser combinadas, na forma de uma árvore, para produzir uma solução do problema complexo.

Os modelos em árvore são designados *árvores de decisão* no caso de problemas de classificação e *árvores de regressão* nos problemas de regressão. (Faceli, 2021)

As árvores de decisão são construídas utilizando dois tipos de elementos: nós e galhos. Em cada nó, um dos atributos do conjunto dos dados é testado de modo a dividir as observações.

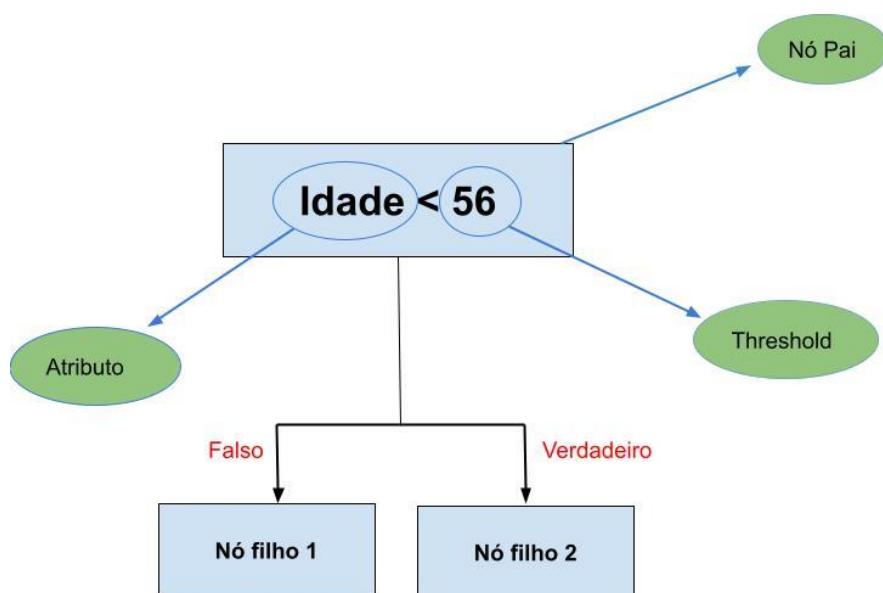


Figura 28: Em cada nó um atributo é testado para decidir qual caminho seguir. (Adaptado de ZORNOZA, 2020)

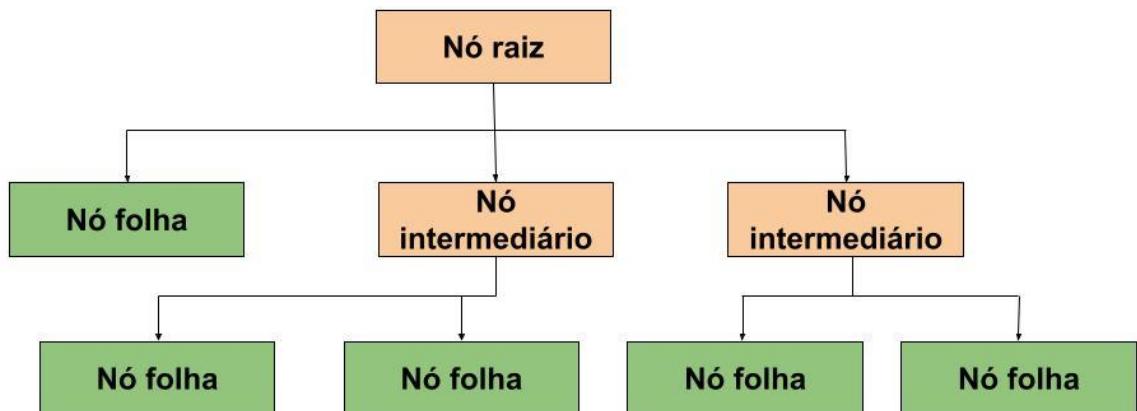


Figura 29: Uma árvore de decisão (Adaptado de ZORNOZA, 2020)

Na figura anterior podemos distinguir 3 tipos nós:

- **Um nó raiz:** o nó onde começa a árvore e que testa um atributo;
- **Nós internos:** nós onde os atributos são testados, mas não são os nós finais;
- **Nós folhas:** os nós finais da árvore que não possuem filhos e onde as predições dos valores categóricos os numéricos são feitos.

O conhecimento em uma árvore de decisão é representado por cada nó, que ao ser testado conduz a busca para um nó filho, até chegar a um nó folha.

Entre os aspectos positivos das árvores de decisão estão (Faceli, 2021):

- Flexibilidade: Árvores de decisão não assumem nenhuma distribuição para os dados. Elas são métodos não paramétricos. O espaço de objetos é dividido em subespaços e cada subespaço é ajustado com diferentes modelos. Uma árvore de decisão fornece uma cobertura exaustiva do espaço de instâncias.
- Robustez: Árvores univariáveis são invariantes a transformações (estritamente) monótonas de variáveis de entrada. Por exemplo, usar x_j , $\log x_j$, ou e^{x_j} como a j-ésima variável de entrada produz árvores com a mesma estrutura. Como uma consequência dessa invariância, a sensibilidade a distribuições com grande cauda e outliers é também reduzida.
- Seleção de atributos: O processo de construção de uma árvore de decisão seleciona os atributos a usar no modelo de decisão. Essa seleção de atributos produz modelos que tendem a ser bastante robustos contra a adição de atributos irrelevantes e redundantes.
- Interpretabilidade: Decisões complexas e globais podem ser aproximadas por uma série de decisões mais simples e locais. Todas as decisões são baseadas nos valores dos atributos usados para descrever o problema. Ambos os aspectos contribuem para a popularidade das árvores de decisão.

Entre os aspectos negativos das árvores de decisão estão (Faceli, 2021):

- Valores ausentes: Uma árvore de decisão é uma hierarquia de testes. Se o valor de um atributo é desconhecido, isso causa problemas em decidir que ramo seguir. Algoritmos devem empregar mecanismos especiais para abordar falta de valores.
- Instabilidade: Pequenas variações no conjunto de treinamento podem produzir grandes variações na árvore final. A cada nó, o critério de mérito de divisão classifica os atributos, e o melhor atributo é escolhido para dividir os dados. Se dois ou mais atributos são classificados similarmente, pequenas variações da classificação dos dados podem alterar a classificação. Todas as subárvore abaixo desse nó mudam. A estratégia da partição recursiva implica que a cada divisão que é feita o dado é dividido com base no atributo de teste. Depois de algumas divisões, há usualmente muito poucos dados nos quais a decisão se baseia. Há uma forte tendência a inferências feitas próximo das folhas serem menos confiáveis que aquelas feitas próximas da raiz.

Florestas aleatórias

A floresta aleatória, como o próprio nome indica, consiste em um grande número de árvores de decisão individuais que operam como um conjunto (ensemble). Cada árvore individual na floresta aleatória gera uma previsão de classe e a classe com mais votos se torna a previsão do seu modelo (veja a figura abaixo).

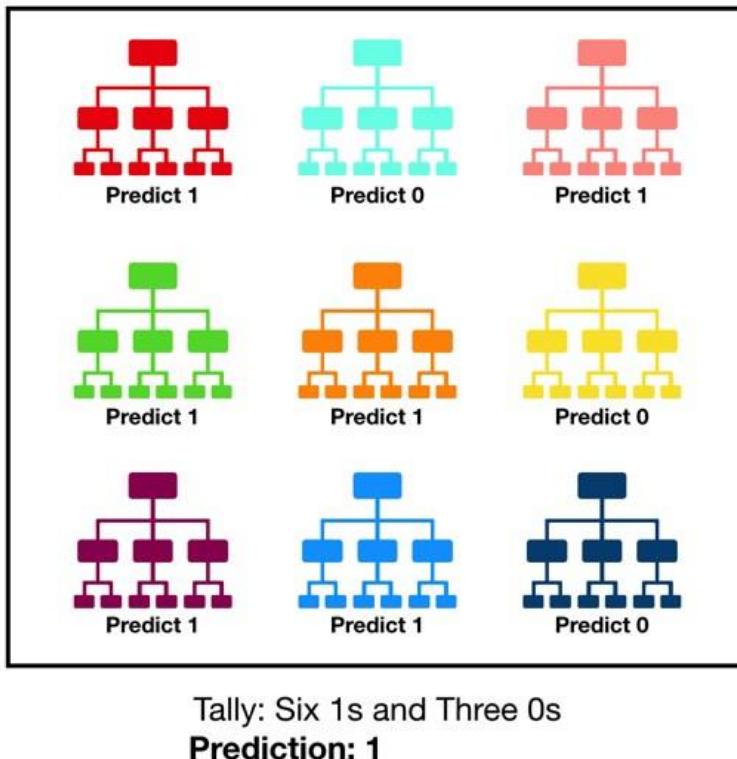


Figura 30 - Visualização de um modelo de floresta aleatória fazendo uma previsão. (YIU, 2019)

Cada uma das árvores é construída usando uma reamostragem dos dados e, em cada divisão, o conjunto de variáveis utilizado é um subconjunto aleatório das variáveis, o que resulta em baixa correlação das árvores individuais. Esta técnica também pode ser aplicada tanto para a classificação quanto para a regressão.

A implementação utilizando do modelo utilizando a função *RandomForestClassifier* do *SkLearn*, juntamente com as funções *RandomSearchCV* para busca no espaço de hiperparâmetros e *StratifiedKFold* para divisão em conjunto de dados.

Os valores para busca no espaço de hiperparâmetros foram definidos de modo a contemplar os valores padrão do algoritmo definido pelo *SkLearn*.

```

espaço_de_parametros = {
    "n_estimators": range(70, 130, 10),
    "max_depth": randint(0, 15),
    "min_samples_split": randint(1, 10),
    "min_samples_leaf": randint(0, 10),
    "bootstrap": [True, False],
    "criterion": ["gini", "entropy"]
}

# cross validation interno (variando conjunto de validação)
modelo_tree = RandomizedSearchCV(RandomForestClassifier(), espaço_de_parametros, n_iter=10, return_train_score=True,
cv = StratifiedKFold(n_splits = 10, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_tree, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

```

Figura 31 - Código para aplicação da validação cruzada aninhada com o algoritmo de floresta aleatória.

SVM (*Support Vector Machine*)

É um algoritmo aplicado tanto para classificação quanto para regressão assim como para problemas lineares quanto não lineares. O algoritmo procura criar uma linha ou hiperplano que separa os dados entre classes.

A SVM faz uso de uma ferramenta denominada SVC, ou classificador de suporte vetorial (em inglês, *support vector classifier*). O classificador SVC separa um conjunto de dados buscando a melhor relação viés/variância. Espera-se obter tanto um viés preditivo quanto variâncias baixas a fim de classificar corretamente não só o maior número possível de amostras de treinamento, mas também de testes. (Lenz, 2020)

Em um problema de classificação com duas classes, o SVM procura encontrar um linha que maximiza a separação entre as classes no espaço bidimensional. Os pontos das duas classes com as distâncias mínimas para o hiperplano são chamados de vetores de suporte. A distância entre os vetores de suporte e o hiperplano é chamado de margem e a hiperplano que maximiza a margem é considerado o ótimo.

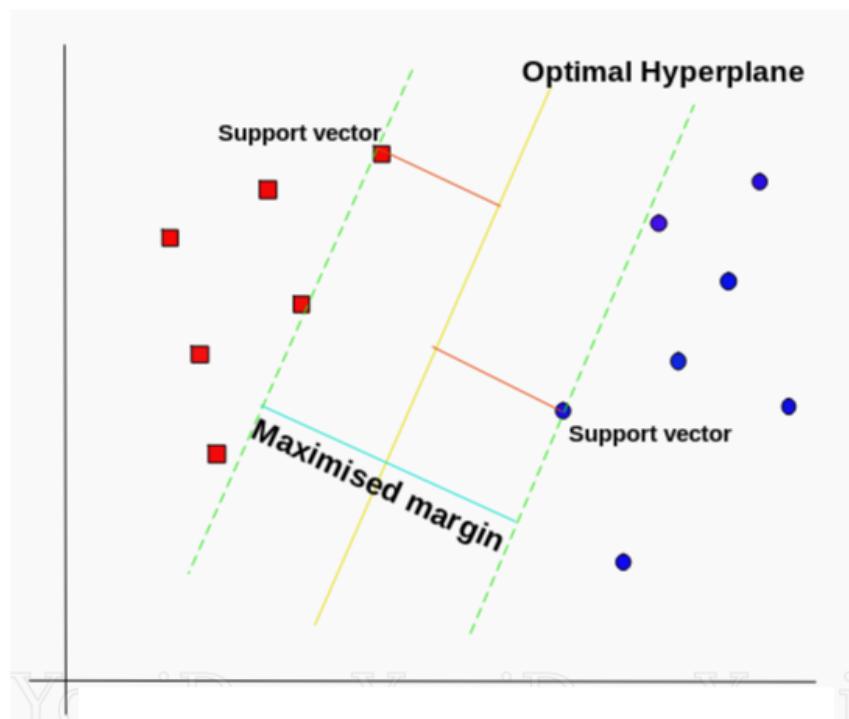


Figura 32 - Hiperplano ótimo com SVM (Pupale, 2018)

Os aspectos positivos das SVM's são (Faceli, 2021):

- Boa capacidade de generalização e robustez diante de objetos de grandes dimensões, sobre os quais outras técnicas de aprendizado comumente obtêm classificadores super ou subajustados.
- O uso de funções kernel na não linearização das SVMs torna o algoritmo eficiente, pois permite a construção de simples hiperplanos em um espaço de alta dimensão de forma tratável do ponto de vista computacional.
- É determinístico e retorna os mesmos resultados independentemente da ordem de apresentação de um mesmo conjunto de treinamento.

Os aspectos negativos das SVM's são (Faceli, 2021):

- Sensibilidade a escolhas de valores de parâmetros e a dificuldade de interpretação do modelo gerado por essa técnica.
- São consideradas “caixas-pretas”, no sentido de que o conhecimento extraído dos objetos por elas se encontra codificado em equações de difícil interpretação, em contraste com os modelos gerados por técnicas simbólicas como as árvores de decisão.

A implementação utilizando do modelo utilizando a função `svm.SVC` do `SkLearn`, juntamente com as funções `RandomSearchCV` para busca no espaço de hiperparâmetros e `StratifiedKFold` para divisão em conjunto de dados.

Os valores para busca no espaço de hiperparâmetros foram definidos de modo a contemplar os valores padrão do algoritmo definido pelo `SkLearn`.

```

espaço_de_parametros = {
    'C': [0.1, 0.5, 1, 2, 5],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
}

# cross validation interno (variando conjunto de validação)
modelo_svm = RandomizedSearchCV(svm.SVC(probability=True), espaço_de_parametros, n_iter=10, return_train_score=True,
                                  cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_svm, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

```

Figura 33 - Código para aplicação da validação cruzada aninhada com o algoritmo de SVM.

Regressão Logística

A Regressão Logística é uma técnica estatística que tem como objetivo produzir, a partir de conjunto de observações, um modelo que permita a predição de valores tomados por uma variável categórica, frequentemente binária, em função de uma ou mais variáveis independentes contínuas e/ou binárias. (Gonzalez, 2018).

Na regressão logística, a probabilidade de ocorrência de um evento pode ser estimada diretamente. No caso da variável dependente Y assumir apenas dois possíveis estados (1 ou 0) e haver um conjunto de p variáveis independentes X_1, X_2, \dots, X_p , o modelo de regressão logística pode ser escrito da seguinte forma:

$$P(Y = 1) = \frac{1}{1 + e^{-g(x)}}$$

Onde:

$$g(x) = B_0 + B_1X_1 + B_2X_2 + \dots + B_pX_p$$

Os coeficientes B_0, B_1, \dots, B_p são estimados a partir do conjunto dados, pelo método da máxima verossimilhança, em que se encontra uma combinação de coeficientes que maximiza a probabilidade da amostra ter sido observada. Considerando uma certa combinação de coeficientes B_0, B_1, \dots, B_p e variando os valores de X , observa-se que a curva logística tem um comportamento probabilístico no formato da letra S, o que é uma característica da regressão logística (edisciplinas usp, 2022).

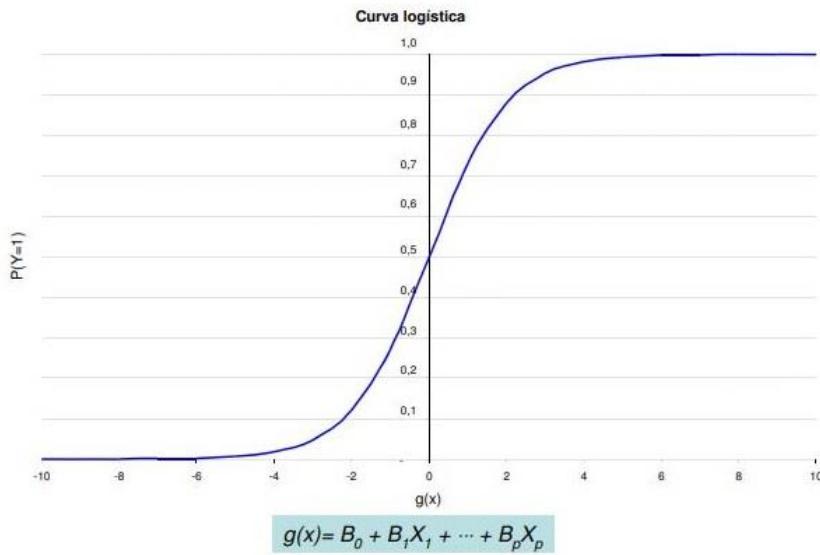


Figura 34 - Curva da Regressão Logística (Adaptado de edisciplinas usp, 2022)

A implementação utilizando do modelo utilizando a função *LogisticRegression* do SkLearn, juntamente com as funções *RandomSearchCV* para busca no espaço de hiperparâmetros e *StratifiedKFold* para divisão em conjunto de dados.

Os valores para busca no espaço de hiperparâmetros foram definidos de modo a contemplar os valores padrão do algoritmo definido pelo *SkLearn*.

```

espaco_de_parametros = {
    'C': [0.1, 0.5, 1, 2, 5],
    'penalty': ['l1', 'l2'],
    'solver': ['lbfgs', 'liblinear'],
    'max_iter': [50, 100, 200, 300, 500]
}

# cross validation interno (variando conjunto de validação)
modelo_logistic = RandomizedSearchCV(LogisticRegression(), espaco_de_parametros, n_iter=10, return_train_score=True,
                                         cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_logistic, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

```

Figura 35 - Código para aplicação da validação cruzada aninhada com o algoritmo de Regressão Logística.

7. Apresentação dos Resultados

Abaixo segue as duas tabelas com o desempenho dos algoritmos nos dois conjuntos de dados.

Tabela 5

Conjunto de dados Alemão		
ALGORITMO	ACURÁCIA MÉDIA (%)	DESVIO PADRÃO (%)
Floresta Aleatória	73,50	3,00
SVM	74,80	3,10
Regressão Logística	75,70	2,30

Tabela 6

Conjunto de dados Australiano		
ALGORITMO	ACURÁCIA MÉDIA (%)	DESVIO PADRÃO (%)
Floresta Aleatória	86,88	1,10
SVM	84,50	1,60
Regressão Logística	86,10	2,00

Podemos perceber que o desempenho dos algoritmos foi pior no conjunto de dados Alemão, isso pode resultado de ruído nos dados, a discretização de variáveis numéricas ou condensação de diferentes informações em uma única atributo como, por exemplo, utiliza-se uma única variável para representar o sexo e o estado civil. Quanto ao conjunto de dados Australiano, não temos maiores informações pois a descrição das variáveis foi suprimida na sua disponibilização.

Quanto ao desempenho dos algoritmos entre si, no conjunto de dados Alemão o algoritmo de Regressão Logística apresentou os melhores resultados e no caso do conjunto de dados Australiano o algoritmo de Floresta Aleatória apresentou os melhores resultados.

8. Links

Aqui você deve disponibilizar os links para o vídeo com sua apresentação de 5 minutos e para o repositório contendo os dados utilizados no projeto, scripts criados, etc.

Link para o vídeo: <https://youtu.be/YYWa0dvluHg>

Link para o repositório: <https://github.com/webertonrc/tcc-puc-2022>

REFERÊNCIAS

Bibliografia

- Baranauskas, J. A. (2022). *Departamento de Física e Matemática – FFCLRP-USP*. Fonte:
<https://dcm.ffclrp.usp.br/~augusto/teaching/ami/AM-I-Conceitos-Definicoes.pdf>.
- Faceli, K. (2021). *Inteligência Artificial: Uma abordagem de Aprendizado de Máquina* (2ª Edição ed.). LTC.
- Gonzalez, L. d. (2018). *Regressão Logistica e suas aplicações* .
- Languages, O. (2022). <https://languages.oup.com/google-dictionary-en/>. Fonte: Oxford Languages.
- Lenz, M. L. (2020). *Fundamentos de aprendizagem de máquina*. Sagah.
- Pupale, R. (16 de 06 de 2018). *Towards Data Science*. Fonte: Towards Data Science:
<https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989>
- YIU, T. (2019). *Towards Data Science*. Fonte: Towards Data Science:
<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
- ZORNOZA, J. (08 de Março de 2020). *Towards Data Science*. Fonte: Towards Data Science:
<https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6>

APÊNDICE

Programação/Scripts

Jupyter Notebook: Alemano

```
# importação das bibliotecas
import pandas as pd

import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold, cross_val_score

import statistics

from scipy.stats import randint

import matplotlib.pyplot as plt

import numpy as np

import warnings
warnings.filterwarnings("ignore")

alemao_df = pd.read_csv("SouthGermanCredit.asc", sep=' ')
alemao_df

      laufkont  laufzeit  moral  verw  hoehe  sparkont  beszeit  rate  famg
es \
0          1        18      4      2    1049         1        2      4
2
1          1        9      4      0    2799         1        3      2
3
2          2       12      2      9     841         2        4      2
2
3          1       12      4      0    2122         1        3      3
3
4          1       12      4      0    2171         1        3      4
3
..        ...
..        ...
995         1       24      2      3    1987         1        3      2
3
996         1       24      2      0    2303         1        5      4
3
997         4       21      4      0   12680         5        5      4
3
```

```

998      2      12      2      3    6468      5      1      2
3
999      1      30      2      2    6350      5      5      4
3

f \
  buerge ... verm alter weitkred wohn bishkred beruf pers tele
0      1 ...     2     21      3      1      1      3      2
1
1      1 ...     1     36      3      1      2      3      1
1
2      1 ...     1     23      3      1      1      2      2      2
1
3      1 ...     1     39      3      1      2      2      1
1
4      1 ...     2     38      1      2      2      2      2
1
...
...   ... ... ... ...
995      1 ...     1     21      3      1      1      2      1
1
996      2 ...     1     45      3      2      1      3      2
1
997      1 ...     4     30      3      3      1      4      2
2
998      1 ...     4     52      3      2      1      4      2
2
999      1 ...     2     31      3      2      1      3      2
1

gastarbkredit
0      2      1
1      2      1
2      2      1
3      1      1
4      1      1
...
995      2      0
996      2      0
997      2      0
998      2      0
999      2      0

```

[1000 rows x 21 columns]

```

renomea = {"laufkont" : "status_conta_corrente", "laufzeit" : "duracao", "moral" : "historico_credito", "verw" : "objetivo_proposta", "hoehe" : "montante", "sparkont" : "poupanca", "beszeit" : "tempo_emprego", "rate" : "percentual_renda", "famges" : "sexo_estado_civil", "buerge" : "garantia", "wohnzeit" : "tempo_residencia", "verm" : "propriedade", "alter" : "idade", "weitkred" : "outro_fornecedor_credito", "wohn" : "tipo_residencia", "bishkred" : "num_credito_tomado", "beruf" : "qualificacao_emprego", "pers" : "dependentes", "tele"

```

```

" : "tem_telefone", "gastarb" : "empregado_estrangeiro",
    "kredit" : "risco_credito"}
alemao_df.rename(columns=renomea, inplace=True)
alemao_df

      status_conta_corrente duracao historico_credito objetivo_proposta
\ 
0                  1          18             4              2
1                  1           9             4              0
2                  2          12             2              9
3                  1          12             4              0
4                  1          12             4              0
..                 ...
995                 1          24             2              3
996                 1          24             2              0
997                 4          21             4              0
998                 2          12             2              3
999                 1          30             2              2

      montante poupanca tempo_emprego percentual_renda sexo_estado_civi
1 \
0          1049         1            2             4
2
1          2799         1            3             2
3
2          841          2            4             2
2
3          2122         1            3             3
3
4          2171         1            3             4
3
..         ...
995        1987         1            3             2
3
996        2303         1            5             4
3
997        12680        5            5             4
3
998        6468          5            1             2
3
999        6350          5            5             4
3

      garantia ... propriedade idade outro_fornecedor_credito \
0          1 ...          2   21               3
1          1 ...          1   36               3
2          1 ...          1   23               3
3          1 ...          1   39               3
4          1 ...          2   38               1
..         ...
995        1 ...          1   21               3
996        2 ...          1   45               3
997        1 ...          4   30               3

```

```

998      1 ...      4    52      3
999      1 ...      2    31      3

  tipo_residencia  num_credito_tomado  qualificacao_emprego  dependente
s \
0          1            1                  1                  3
2
1          1            2                  2                  3
1
2          1            1                  1                  2
2
3          1            2                  2                  2
1
4          2            2                  2                  2
2
..        ...
.
995      1            1                  1                  2
1
996      2            1                  1                  3
2
997      3            1                  1                  4
2
998      2            1                  1                  4
2
999      2            1                  1                  3
2

  tem_telefone  empregado_estrangeiro  risco_credito
0          1            2                  1
1          1            2                  1
2          1            2                  1
3          1            1                  1
4          1            1                  1
..
995      1            2                  0
996      1            2                  0
997      2            2                  0
998      2            2                  0
999      1            2                  0

[1000 rows x 21 columns]

```

Exploração dos dados

```

alemao_df.describe()\
    .transpose()

           count      mean       std      min    25% \
status_conta_corrente  1000.0    2.577  1.257638  1.0    1.0
duracao                1000.0   20.903 12.058814  4.0   12.0
historico_credito     1000.0    2.545  1.083120  0.0    2.0
objetivo_proposta     1000.0    2.828  2.744439  0.0    1.0
montante                1000.0 3271.248 2822.751760 250.0 1365.5
poupanca               1000.0    2.105  1.580023  1.0    1.0

```

tempo_emprego	1000.0	3.384	1.208306	1.0	3.0
percentual_renda	1000.0	2.973	1.118715	1.0	2.0
sexo_estado_civil	1000.0	2.682	0.708080	1.0	2.0
garantia	1000.0	1.145	0.477706	1.0	1.0
tempo_residencia	1000.0	2.845	1.103718	1.0	2.0
propriedade	1000.0	2.358	1.050209	1.0	1.0
idade	1000.0	35.542	11.352670	19.0	27.0
outro_fornecedor_credito	1000.0	2.675	0.705601	1.0	3.0
tipo_residencia	1000.0	1.928	0.530186	1.0	2.0
num_credito_tomado	1000.0	1.407	0.577654	1.0	1.0
qualificacao_emprego	1000.0	2.904	0.653614	1.0	3.0
dependentes	1000.0	1.845	0.362086	1.0	2.0
tem_telefone	1000.0	1.404	0.490943	1.0	1.0
empregado_estrangeiro	1000.0	1.963	0.188856	1.0	2.0
risco_credito	1000.0	0.700	0.458487	0.0	0.0
		50%	75%	max	
status_conta_corrente	2.0	4.00	4.0		
duracao	18.0	24.00	72.0		
historico_credito	2.0	4.00	4.0		
objetivo_proposta	2.0	3.00	10.0		
montante	2319.5	3972.25	18424.0		
poupanca	1.0	3.00	5.0		
tempo_emprego	3.0	5.00	5.0		
percentual_renda	3.0	4.00	4.0		
sexo_estado_civil	3.0	3.00	4.0		
garantia	1.0	1.00	3.0		
tempo_residencia	3.0	4.00	4.0		
propriedade	2.0	3.00	4.0		
idade	33.0	42.00	75.0		
outro_fornecedor_credito	3.0	3.00	3.0		
tipo_residencia	2.0	2.00	3.0		
num_credito_tomado	1.0	2.00	4.0		
qualificacao_emprego	3.0	3.00	4.0		
dependentes	2.0	2.00	2.0		
tem_telefone	1.0	2.00	2.0		
empregado_estrangeiro	2.0	2.00	2.0		
risco_credito	1.0	1.00	1.0		

```
# Checando a proporção de registros por classe
bom = alemao_df.loc[alemao_df['risco_credito'] == 1]
ruim = alemao_df.loc[alemao_df['risco_credito'] == 0]
print("Temos", len(bom), "classificados como crédito bom e", len(ruim), "classificados como crédito ruim.")
```

Temos 700 classificados como crédito bom e 300 classificados como crédito ruim.

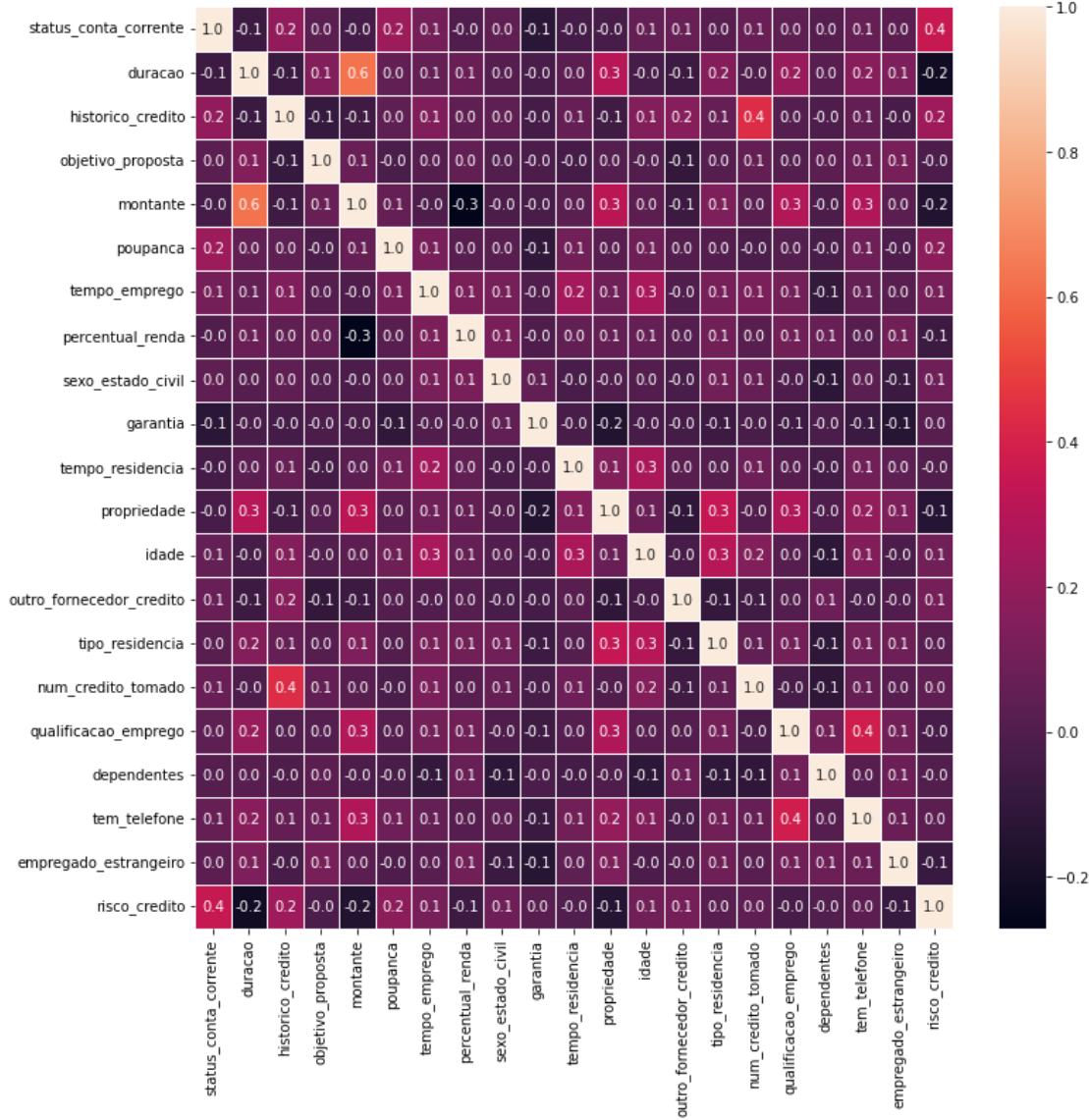
```
alemao_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   ID              1000 non-null    int64  
 1   RISCO_CREDITO  1000 non-null    int64  
 2   ANO             1000 non-null    float64
 3   MINGO          1000 non-null    float64
 4   SEXTA           1000 non-null    float64
 5   SABADO          1000 non-null    float64
 6   DOMINGO         1000 non-null    float64
 7   CUSTO           1000 non-null    float64
 8   INVESTIMENTO    1000 non-null    float64
 9   SALARIO          1000 non-null    float64
 10  CUSTO_ALIMENTACAO 1000 non-null    float64
 11  CUSTO_HABITACAO 1000 non-null    float64
 12  CUSTO_EDUCACAO  1000 non-null    float64
 13  CUSTO_LIGACAO  1000 non-null    float64
 14  CUSTO_TEL      1000 non-null    float64
 15  CUSTO_ALUGUEL  1000 non-null    float64
 16  CUSTO_ALIMENTACAO 1000 non-null    float64
 17  CUSTO_HABITACAO 1000 non-null    float64
 18  CUSTO_EDUCACAO  1000 non-null    float64
 19  CUSTO_LIGACAO  1000 non-null    float64
 20  CUSTO_TEL      1000 non-null    float64
```

```
0    status_conta_corrente      1000 non-null   int64
1    duracao                  1000 non-null   int64
2    historico_credito        1000 non-null   int64
3    objetivo_proposta         1000 non-null   int64
4    montante                  1000 non-null   int64
5    poupanca                 1000 non-null   int64
6    tempo_emprego             1000 non-null   int64
7    percentual_renda          1000 non-null   int64
8    sexo_estado_civil         1000 non-null   int64
9    garantia                 1000 non-null   int64
10   tempo_residencia          1000 non-null   int64
11   propriedade               1000 non-null   int64
12   idade                     1000 non-null   int64
13   outro_fornecedor_credito 1000 non-null   int64
14   tipo_residencia           1000 non-null   int64
15   num_credito_tomado        1000 non-null   int64
16   qualificacao_emprego     1000 non-null   int64
17   dependentes                1000 non-null   int64
18   tem_telefone               1000 non-null   int64
19   empregado_estrangeiro     1000 non-null   int64
20   risco_credito              1000 non-null   int64
dtypes: int64(21)
memory usage: 164.2 KB

correlation = alemao_df.corr()
fig, ax = plt.subplots(figsize=(12,12))
plot = sns.heatmap(correlation, annot = True, fmt=".1f", linewidths=.6, ax=ax)
plot

<AxesSubplot:>
```



Trata variáveis categóricas

```
categoricas = ["status_conta_corrente", "historico_credito", "objetivo_proposta",
               "poupanca", "sexo_estado_civil", "garantia", "outro_fornecedor_credito",
               "tipo_residencia", "propriedade",
               "qualificacao_emprego", "tempo_emprego", "empregado_estrangeiro",
               "percentual_renda", "tempo_residencia", "num_credito_tomado",
               "dependentes"]
```

```
one_hot = OneHotEncoder(handle_unknown='ignore', categories='auto', drop='first', sparse=False)
```

```
cat_df = pd.DataFrame(one_hot.fit_transform(alemao_df[categoricas]), columns=one_hot.get_feature_names_out(categoricas))
cat_df.head()
```

```
status_conta_corrente_2  status_conta_corrente_3  status_conta_corrente
-4 \                                         0.0                                         0.0                                         0
```

```

.0
1          0.0          0.0          0
.0
2          1.0          0.0          0
.0
3          0.0          0.0          0
.0
4          0.0          0.0          0
.0

    historico_credito_1  historico_credito_2  historico_credito_3 \
0          0.0          0.0          0.0
1          0.0          0.0          0.0
2          0.0          1.0          0.0
3          0.0          0.0          0.0
4          0.0          0.0          0.0

    historico_credito_4  objetivo_proposta_1  objetivo_proposta_2 \
0          1.0          0.0          1.0
1          1.0          0.0          0.0
2          0.0          0.0          0.0
3          1.0          0.0          0.0
4          1.0          0.0          0.0

    objetivo_proposta_3  ...  percentual renda_2  percentual renda_3 \
0          0.0  ...          0.0          0.0
1          0.0  ...          1.0          0.0
2          0.0  ...          1.0          0.0
3          0.0  ...          0.0          1.0
4          0.0  ...          0.0          0.0

    percentual renda_4  tempo_residencia_2  tempo_residencia_3 \
0          1.0          0.0          0.0
1          0.0          1.0          0.0
2          0.0          0.0          0.0
3          0.0          1.0          0.0
4          1.0          0.0          0.0

    tempo_residencia_4  num_credito_tomado_2  num_credito_tomado_3 \
0          1.0          0.0          0.0
1          0.0          1.0          0.0
2          1.0          0.0          0.0
3          0.0          1.0          0.0
4          1.0          1.0          0.0

    num_credito_tomado_4  dependentes_2
0          0.0          1.0
1          0.0          0.0
2          0.0          1.0
3          0.0          0.0
4          0.0          1.0

```

[5 rows x 50 columns]

```

alemao_df_copia = alemao_df.drop(categoricas, axis=1)
alemao_df_copia = pd.concat([cat_df, alemao_df_copia], axis=1)
alemao_df_copia

    status_conta_corrente_2  status_conta_corrente_3  \
0                  0.0                  0.0
1                  0.0                  0.0
2                  1.0                  0.0
3                  0.0                  0.0
4                  0.0                  0.0
..                  ...
995                 0.0                  0.0
996                 0.0                  0.0
997                 0.0                  0.0
998                 1.0                  0.0
999                 0.0                  0.0

    status_conta_corrente_4  historico_credito_1  historico_credito_2  \
0                  0.0                  0.0                  0.0
1                  0.0                  0.0                  0.0
2                  0.0                  0.0                  1.0
3                  0.0                  0.0                  0.0
4                  0.0                  0.0                  0.0
..                  ...
995                 0.0                  0.0                  1.0
996                 0.0                  0.0                  1.0
997                 1.0                  0.0                  0.0
998                 0.0                  0.0                  1.0
999                 0.0                  0.0                  1.0

    historico_credito_3  historico_credito_4  objetivo_proposta_1  \
0                  0.0                  1.0                  0.0
1                  0.0                  1.0                  0.0
2                  0.0                  0.0                  0.0
3                  0.0                  1.0                  0.0
4                  0.0                  1.0                  0.0
..                  ...
995                 0.0                  0.0                  0.0
996                 0.0                  0.0                  0.0
997                 0.0                  1.0                  0.0
998                 0.0                  0.0                  0.0
999                 0.0                  0.0                  0.0

    objetivo_proposta_2  objetivo_proposta_3  ...  tempo_residencia_4  \
0                  1.0                  0.0  ...                  1.0
1                  0.0                  0.0  ...                  0.0
2                  0.0                  0.0  ...                  1.0
3                  0.0                  0.0  ...                  0.0
4                  0.0                  0.0  ...                  1.0
..                  ...
995                 0.0                  1.0  ...                  1.0
996                 0.0                  0.0  ...                  0.0
997                 0.0                  0.0  ...                  1.0
998                 0.0                  1.0  ...                  0.0

```

```

999          1.0          0.0  ...
num_credito_tomado_2  num_credito_tomado_3  num_credito_tomado_4  \
0            0.0            0.0            0.0            0.0
1            1.0            0.0            0.0            0.0
2            0.0            0.0            0.0            0.0
3            1.0            0.0            0.0            0.0
4            1.0            0.0            0.0            0.0
..           ...
995          0.0            0.0            0.0            0.0
996          0.0            0.0            0.0            0.0
997          0.0            0.0            0.0            0.0
998          0.0            0.0            0.0            0.0
999          0.0            0.0            0.0            0.0

dependentes_2  duracao  montante  idade  tem_telefone  risco_credito
0            1.0        18       1049     21            1            1
1            0.0         9       2799     36            1            1
2            1.0        12       841      23            1            1
3            0.0        12      2122     39            1            1
4            1.0        12      2171     38            1            1
..           ...
995          0.0        24      1987     21            1            0
996          1.0        24      2303     45            1            0
997          1.0        21     12680     30            2            0
998          1.0        12      6468     52            2            0
999          1.0        30      6350     31            1            0

```

[1000 rows x 55 columns]

Trata variáveis numéricas

```
numericas = ["duracao", "montante", "idade"]

mms = MinMaxScaler()

num_df = pd.DataFrame(mms.fit_transform(alemao_df[numericas]), columns = mms.get_feature_names_out(numericas))
num_df.head()

      duracao  montante      idade
0  0.205882  0.043964  0.035714
1  0.073529  0.140255  0.303571
2  0.117647  0.032519  0.071429
3  0.117647  0.103004  0.357143
4  0.117647  0.105700  0.339286

alemao_df_copia = alemao_df_copia.drop(numericas, axis=1)
alemao_df_copia = pd.concat([num_df, alemao_df_copia], axis=1)
alemao_df_copia

      duracao  montante      idade  status_conta_corrente_2 \
0  0.205882  0.043964  0.035714                      0.0
1  0.073529  0.140255  0.303571                      0.0
2  0.117647  0.032519  0.071429                     1.0
```

3	0.117647	0.103004	0.357143	0.0
4	0.117647	0.105700	0.339286	0.0
..
995	0.294118	0.095576	0.035714	0.0
996	0.294118	0.112964	0.464286	0.0
997	0.250000	0.683944	0.196429	0.0
998	0.117647	0.342137	0.589286	1.0
999	0.382353	0.335644	0.214286	0.0
\\				
0		0.0	0.0	0.0
1		0.0	0.0	0.0
2		0.0	0.0	0.0
3		0.0	0.0	0.0
4		0.0	0.0	0.0
..
995		0.0	0.0	0.0
996		0.0	0.0	0.0
997		0.0	1.0	0.0
998		0.0	0.0	0.0
999		0.0	0.0	0.0
historico_credito_2 historico_credito_3 historico_credito_4 ... \\				
0	0.0	0.0	1.0	...
1	0.0	0.0	1.0	...
2	1.0	0.0	0.0	...
3	0.0	0.0	1.0	...
4	0.0	0.0	1.0	...
..
995	1.0	0.0	0.0	...
996	1.0	0.0	0.0	...
997	0.0	0.0	1.0	...
998	1.0	0.0	0.0	...
999	1.0	0.0	0.0	...
percentual_renda_4 tempo_residencia_2 tempo_residencia_3 \\				
0	1.0	0.0	0.0	
1	0.0	1.0	0.0	
2	0.0	0.0	0.0	
3	0.0	1.0	0.0	
4	1.0	0.0	0.0	
..	
995	0.0	0.0	0.0	
996	1.0	0.0	0.0	
997	1.0	0.0	0.0	
998	0.0	0.0	0.0	
999	1.0	0.0	0.0	
tempo_residencia_4 num_credito_tomado_2 num_credito_tomado_3 \\				
0	1.0	0.0	0.0	
1	0.0	1.0	0.0	
2	1.0	0.0	0.0	
3	0.0	1.0	0.0	

```

4           1.0           1.0           0.0
..
995        ...          0.0           0.0
996        0.0           0.0           0.0
997        1.0           0.0           0.0
998        0.0           0.0           0.0
999        1.0           0.0           0.0

  num_credito_tomado_4  dependentes_2  tem_telefone  risco_credito
0            0.0           1.0           1             1
1            0.0           0.0           1             1
2            0.0           1.0           1             1
3            0.0           0.0           1             1
4            0.0           1.0           1             1
..
995         0.0           0.0           1             0
996         0.0           1.0           1             0
997         0.0           1.0           2             0
998         0.0           1.0           2             0
999         0.0           1.0           1             0

[1000 rows x 55 columns]

X = alemao_df_copia.loc[:,alemao_df_copia.columns != 'risco_credito']
y = alemao_df_copia['risco_credito']

# Visualizar a distribuição de cada variável

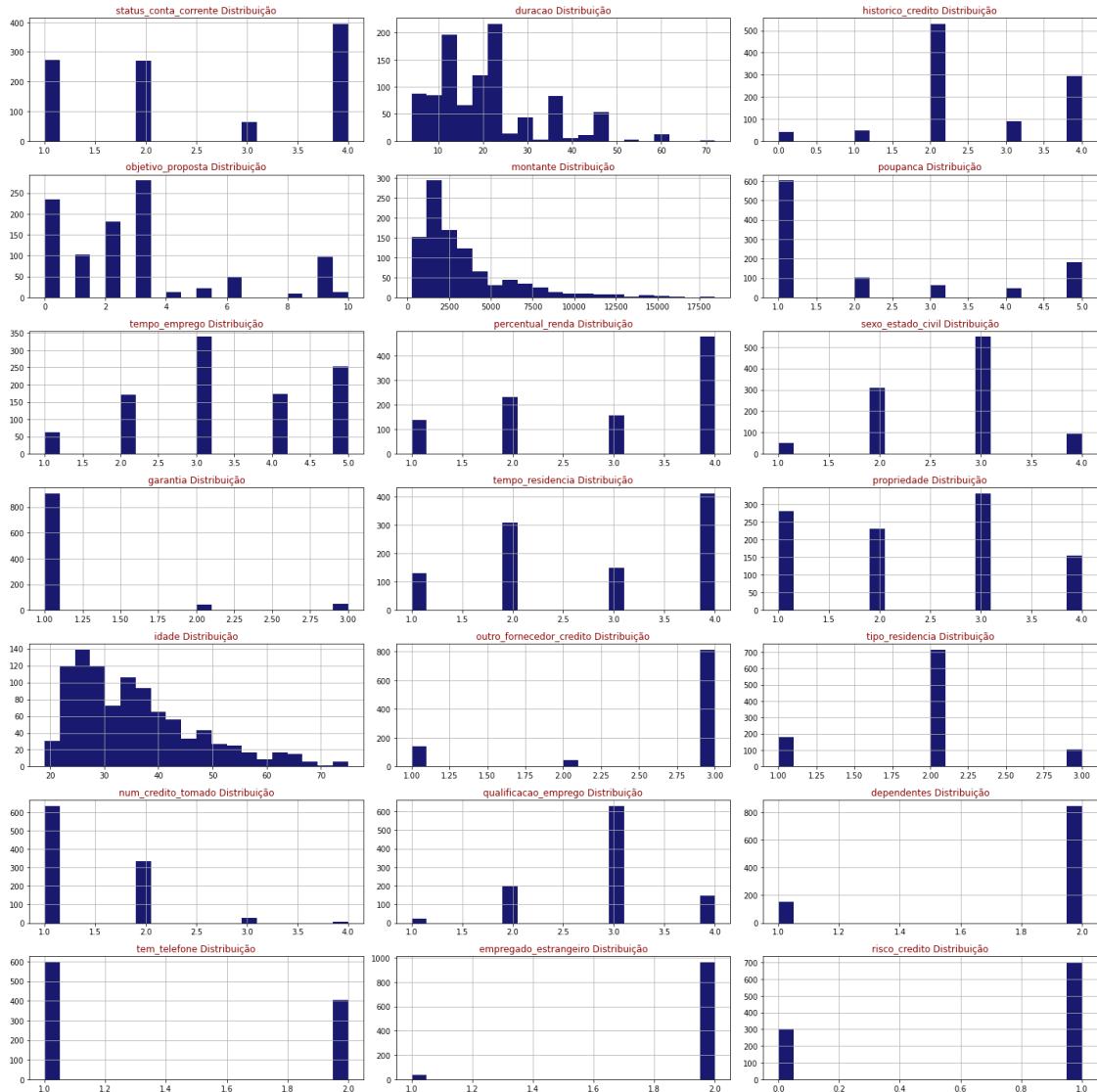
rows = 7
cols = 3

fig = plt.figure(figsize = (20,20))

for i, coluna in enumerate(alemao_df.columns):
    ax = fig.add_subplot(rows, cols, i+1)
    alemao_df[coluna].hist(bins = 20, ax = ax, facecolor = 'midnightblue')
    ax.set_title(coluna + " Distribuição", color = 'DarkRed')

fig.tight_layout()
plt.show()

```



Seleção de hiperparametros

Random Forest - Nested Cross validation

```

espaco_de_parametros = {
    "n_estimators" : range(70, 130, 10),
    "max_depth" : randint(0, 15),
    "min_samples_split" : randint(1, 10),
    "min_samples_leaf" : randint(0, 10),
    "bootstrap" : [True, False],
    "criterion" : ["gini", "entropy"]
}

# cross validation interno (variando conjunto de validação)
modelo_tree = RandomizedSearchCV(RandomForestClassifier(), espaco_de_parametros, n_iter=10, return_train_score=True,
                                  cv = StratifiedKFold(n_splits = 10, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_tree, X, y, cv = StratifiedKFold(n_splits

```

```

= 5, shuffle=True, random_state=0))
scores
array([0.75 , 0.76 , 0.695, 0.71 , 0.76 ])

print("Acurácia Média: {:.2%}".format(round(scores.mean(), 3)), "Desvio padrão: {:.2%}".format(round(statistics.stdev(scores), 3)))
Acurácia Média: 73.50% Desvio padrão: 3.00%

SVM - Nested Cross Validation

espaço_de_parametros = {
    'C': [0.1, 0.5, 1, 2, 5],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
}

# cross validation interno (variando conjunto de validação)
modelo_svm = RandomizedSearchCV(svm.SVC(probability=True), espaço_de_parametros, n_iter=10, return_train_score=True,
                                  cv = StratifiedKFold(n_splits = 10, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_svm, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

array([0.7 , 0.775, 0.76 , 0.735, 0.77 ])

print("Acurácia Média: {:.2%}".format(round(scores.mean(), 3)), "Desvio padrão: {:.2%}".format(round(statistics.stdev(scores), 3)))
Acurácia Média: 74.80% Desvio padrão: 3.10%

Regressão logística - Nested Cross Validation

espaço_de_parametros = {
    'C': [0.1, 0.5, 1, 2, 5],
    'penalty': ['l1', 'l2'],
    'solver': ['lbfgs', 'liblinear'],
    'max_iter': [50, 100, 200, 300, 500]
}

# cross validation interno (variando conjunto de validação)
modelo_logistic = RandomizedSearchCV(LogisticRegression(), espaço_de_parametros, n_iter=10, return_train_score=True,
                                       cv = StratifiedKFold(n_splits = 10, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_logistic, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

array([0.755, 0.76 , 0.77 , 0.72 , 0.78 ])

```

```
print("Acurácia Média: {:.2%}".format(round(scores.mean(), 3)), "Desvio padrão: {:.2%}".format(round(statistics.stdev(scores), 3)))
```

Acurácia Média: 75.70% Desvio padrão: 2.30%

Jupyter Notebook: Australiano

```
# importação das bibliotecas
import pandas as pd

import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold, cross_val_score

import statistics

from scipy.stats import randint

import matplotlib.pyplot as plt

import numpy as np

import warnings
warnings.filterwarnings("ignore")

australiano_df = pd.read_csv("australian.dat", sep=' ', header=None)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
14	0	1	22.08	11.460	2	4	4	1.585	0	0	0	1	2	100	1213
0	0	1	22.67	7.000	2	8	4	0.165	0	0	0	0	2	160	1
0	2	0	29.58	1.750	1	4	4	1.250	0	0	0	1	2	280	1
0	3	0	21.67	11.500	1	5	3	0.000	1	1	11	1	2	0	1
1	4	1	20.17	8.170	2	6	4	1.960	1	1	14	0	2	60	159
1
..	685	1	31.57	10.500	2	14	4	6.500	1	0	0	0	2	0	1
1	686	1	20.67	0.415	2	8	4	0.125	0	0	0	0	2	0	45
0	687	0	18.83	9.540	2	6	4	0.085	1	0	0	0	2	100	1

```

1
688   0  27.42  14.500    2  14    8  3.085    1    1    1    0    2  120    12
1
689   1  41.00  0.040    2  10    4  0.040    0    1    1    0    1  560    1
1

[690 rows x 15 columns]

australiano_df.rename(columns={0:"col1", 1:"col2", 2:"col3",
                               3:"col4", 4:"col5", 5:"col6",
                               6:"col7", 7:"col8", 8:"col9",
                               9:"col10", 10:"col11", 11:"col12",
                               12:"col13", 13:"col14", 14:"col15"}, inplace=True)
australiano_df.head()

      col1   col2   col3   col4   col5   col6   col7   col8   col9   col10  col11
\ 
0     1  22.08  11.46    2     4     4  1.585    0     0     0     0     1
1     0  22.67   7.00    2     8     4  0.165    0     0     0     0     0
2     0  29.58   1.75    1     4     4  1.250    0     0     0     0     1
3     0  21.67  11.50    1     5     3  0.000    1     1     1    11     1
4     1  20.17   8.17    2     6     4  1.960    1     1    14     0

      col12  col13  col14  col15
0       2    100   1213     0
1       2    160     1     0
2       2    280     1     0
3       2      0     1     1
4       2     60   159     1

```

Exploração dos dados

```

australiano_df.describe()\
    .transpose()

      count      mean        std       min      25%      50%      75%
\ 
col1    690.0  0.678261  0.467482  0.00  0.000  1.000  1.0000
col2    690.0  31.568203 11.853273 13.75 22.670 28.625 37.7075
col3    690.0  4.758725  4.978163  0.00  1.000  2.750  7.2075
col4    690.0  1.766667  0.430063  1.00  2.000  2.000  2.0000
col5    690.0  7.372464  3.683265  1.00  4.000  8.000 10.0000
col6    690.0  4.692754  1.992316  1.00  4.000  4.000  5.0000
col7    690.0  2.223406  3.346513  0.00  0.165  1.000  2.6250
col8    690.0  0.523188  0.499824  0.00  0.000  1.000  1.0000
col9    690.0  0.427536  0.495080  0.00  0.000  0.000  1.0000
col10   690.0  2.400000  4.862940  0.00  0.000  0.000  3.0000
col11   690.0  0.457971  0.498592  0.00  0.000  0.000  1.0000
col12   690.0  1.928986  0.298813  1.00  2.000  2.000  2.0000
col13   690.0  184.014493 172.159274  0.00  80.000 160.000 272.0000
col14   690.0  1018.385507 5210.102598  1.00  1.000  6.000 396.5000
col15   690.0  0.444928  0.497318  0.00  0.000  0.000  1.0000

      max

```

```

col1      1.00
col2     80.25
col3    28.00
col4     3.00
col5   14.00
col6     9.00
col7   28.50
col8     1.00
col9     1.00
col10   67.00
col11   1.00
col12     3.00
col13  2000.00
col14 100001.00
col15     1.00

```

```

# Checando a proporção de registros por classe
aprovado = australiano_df.loc[australiano_df['col15'] == 1]
nao_aprovado = australiano_df.loc[australiano_df['col15'] == 0]
print("Temos", len(aprovado), "solicitações de cartão de crédito aprovadas"
e", len(nao_aprovado),
      "solicitações de cartão de crédito não aprovadas.")

```

Temos 307 solicitações de cartão de crédito aprovadas e 383 solicitações de cartão de crédito não aprovadas.

```
australiano_df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   col1    690 non-null   int64  
 1   col2    690 non-null   float64 
 2   col3    690 non-null   float64 
 3   col4    690 non-null   int64  
 4   col5    690 non-null   int64  
 5   col6    690 non-null   int64  
 6   col7    690 non-null   float64 
 7   col8    690 non-null   int64  
 8   col9    690 non-null   int64  
 9   col10   690 non-null   int64  
 10  col11   690 non-null   int64  
 11  col12   690 non-null   int64  
 12  col13   690 non-null   int64  
 13  col14   690 non-null   int64  
 14  col15   690 non-null   int64  
dtypes: float64(3), int64(12)
memory usage: 81.0 KB

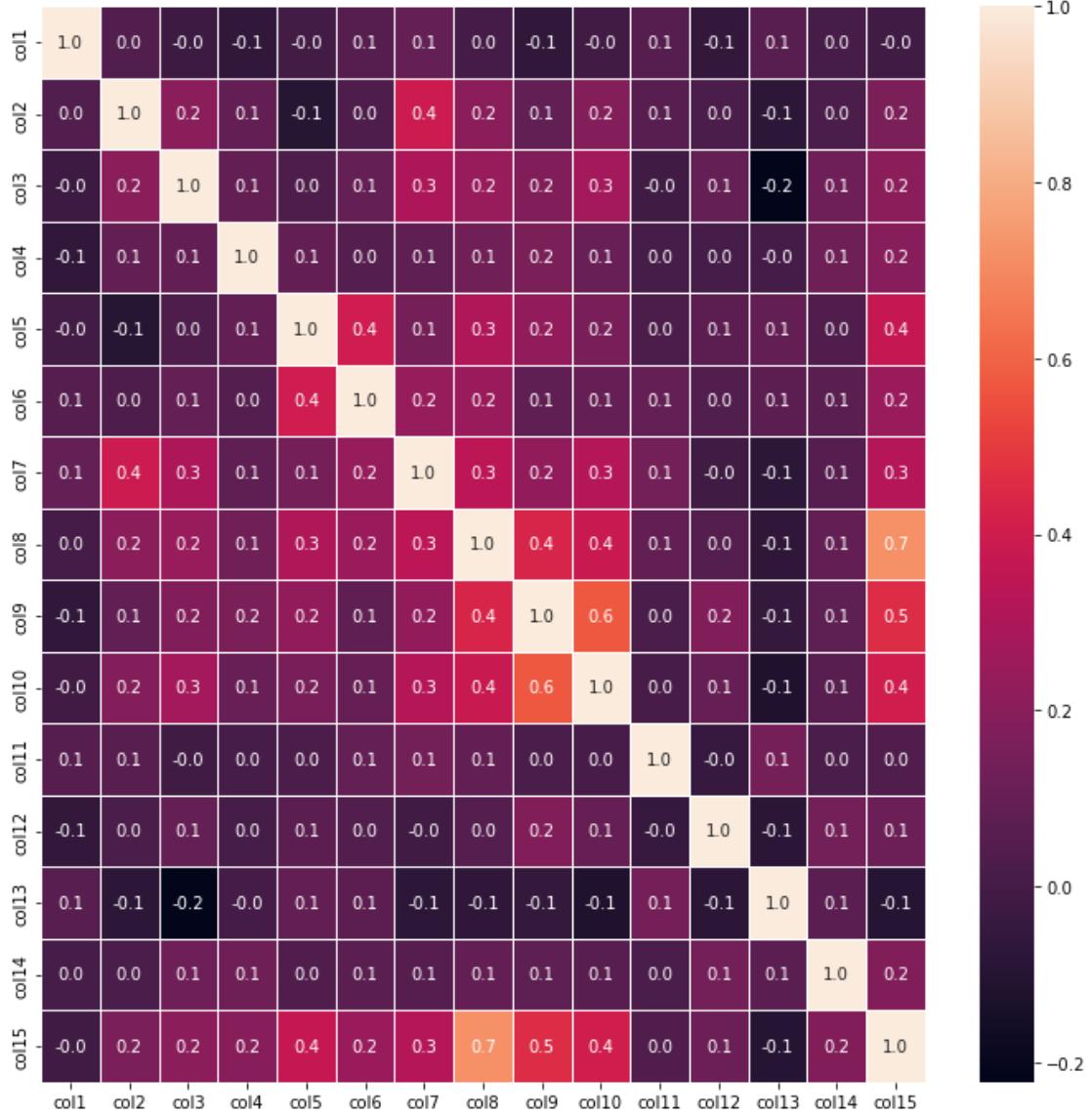
```

```

correlation = australiano_df.corr()
fig, ax = plt.subplots(figsize=(12,12))
plot = sns.heatmap(correlation, annot = True, fmt=".1f", linewidths=.6, ax

```

```
=ax)
plot
<AxesSubplot:>
```



Trata variáveis categóricas

```
categoricas = ["col1", "col4", "col5", "col6", "col8", "col9", "col11", "col12"]

one_hot = OneHotEncoder(handle_unknown='ignore', categories='auto', drop='first', sparse=False)

cat_df = pd.DataFrame(one_hot.fit_transform(australiano_df[categoricas]),
columns = one_hot.get_feature_names_out(categoricas))
cat_df.head()

  col1_1  col4_2  col4_3  col5_2  col5_3  col5_4  col5_5  col5_6  col5_7
0      1.0      1.0      0.0      0.0      0.0      0.0      1.0      0.0      0.0
```

```

1   0.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2   0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
3   0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
4   1.0    1.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0

  col5_8 ... col6_4  col6_5  col6_7  col6_8  col6_9  col8_1  col9_1 \
0   0.0 ...   1.0    0.0    0.0    0.0    0.0    0.0    0.0
1   1.0 ...   1.0    0.0    0.0    0.0    0.0    0.0    0.0
2   0.0 ...   1.0    0.0    0.0    0.0    0.0    0.0    0.0
3   0.0 ...   0.0    0.0    0.0    0.0    0.0    0.0    1.0
4   0.0 ...   1.0    0.0    0.0    0.0    0.0    1.0    1.0

  col11_1  col12_2  col12_3
0     1.0    1.0    0.0
1     0.0    1.0    0.0
2     1.0    1.0    0.0
3     1.0    1.0    0.0
4     0.0    1.0    0.0

[5 rows x 28 columns]

australiano_df_copia = australiano_df.drop(categoricas, axis=1)
australiano_df_copia = pd.concat([cat_df, australiano_df_copia], axis=1)
australiano_df_copia

  col1_1  col4_2  col4_3  col5_2  col5_3  col5_4  col5_5  col5_6  col5_7 \
0       1.0    1.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
0
1       0.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
0
2       0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
0
3       0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
0
4       1.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    1.0
0
..     ...
.      ...
685     1.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
0
686     1.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
0
687     0.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    1.0
0
688     0.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
0
689     1.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
0

  col5_8 ... col11_1  col12_2  col12_3  col2  col3  col7  col10 \
0       0.0 ...   1.0    1.0    0.0  22.08  11.460  1.585   0
1       1.0 ...   0.0    1.0    0.0  22.67  7.000   0.165   0

```

```

2      0.0 ...     1.0      1.0      0.0   29.58    1.750  1.250      0
3      0.0 ...     1.0      1.0      0.0   21.67   11.500  0.000     11
4      0.0 ...     0.0      1.0      0.0   20.17    8.170  1.960     14
...
685     0.0 ...     0.0      1.0      0.0   31.57   10.500  6.500      0
686     1.0 ...     0.0      1.0      0.0   20.67    0.415  0.125      0
687     0.0 ...     0.0      1.0      0.0   18.83    9.540  0.085      0
688     0.0 ...     0.0      1.0      0.0   27.42   14.500  3.085     1
689     0.0 ...     0.0      0.0      0.0   41.00    0.040  0.040     1

      col13  col14  col15
0      100   1213    0
1      160     1    0
2      280     1    0
3       0     1    1
4       60   159    1
...
685      0     1    1
686      0    45    0
687     100     1    1
688     120    12    1
689     560     1    1

[690 rows x 35 columns]

```

```

import numpy as np
print(np.__version__)

```

1.21.5

Trata variáveis numéricas

```

numericas = ["col2", "col3", "col7", "col10", "col13", "col14"]

mms = MinMaxScaler()

num_df = pd.DataFrame(mms.fit_transform(australiano_df[numericas]), column
s = mms.get_feature_names_out(numericas))
num_df.head()

      col2      col3      col7      col10     col13     col14
0  0.125263  0.409286  0.055614  0.000000   0.05  0.01212
1  0.134135  0.250000  0.005789  0.000000   0.08  0.00000
2  0.238045  0.062500  0.043860  0.000000   0.14  0.00000
3  0.119098  0.410714  0.000000  0.164179   0.00  0.00000
4  0.096541  0.291786  0.068772  0.208955   0.03  0.00158

australiano_df_copia = australiano_df_copia.drop(numericas, axis=1)
australiano_df_copia = pd.concat([num_df, australiano_df_copia], axis=1)
australiano_df_copia

      col2      col3      col7      col10     col13     col14  col1_1  col4_
2 \
0   0.125263  0.409286  0.055614  0.000000   0.05  0.01212      1.0     1.
0

```

1	0.134135	0.250000	0.005789	0.000000	0.08	0.00000	0.0	1.	
0									
2	0.238045	0.062500	0.043860	0.000000	0.14	0.00000	0.0	0.	
0									
3	0.119098	0.410714	0.000000	0.164179	0.00	0.00000	0.0	0.	
0									
4	0.096541	0.291786	0.068772	0.208955	0.03	0.00158	1.0	1.	
0									
..	
.									
685	0.267970	0.375000	0.228070	0.000000	0.00	0.00000	1.0	1.	
0									
686	0.104060	0.014821	0.004386	0.000000	0.00	0.00044	1.0	1.	
0									
687	0.076391	0.340714	0.002982	0.000000	0.05	0.00000	0.0	1.	
0									
688	0.205564	0.517857	0.108246	0.014925	0.06	0.00011	0.0	1.	
0									
689	0.409774	0.001429	0.001404	0.014925	0.28	0.00000	1.0	1.	
0									
	col4_3	col5_2	...	col6_5	col6_7	col6_8	col6_9	col8_1	col9_1
\									
0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	1.0
4	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	1.0
..
685	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0
686	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
687	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0
688	0.0	0.0	...	0.0	0.0	1.0	0.0	1.0	1.0
689	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0
	col11_1	col12_2	col12_3	col15					
0	1.0	1.0	0.0	0					
1	0.0	1.0	0.0	0					
2	1.0	1.0	0.0	0					
3	1.0	1.0	0.0	1					
4	0.0	1.0	0.0	1					
..					
685	0.0	1.0	0.0	1					
686	0.0	1.0	0.0	0					
687	0.0	1.0	0.0	1					
688	0.0	1.0	0.0	1					
689	0.0	0.0	0.0	1					

[690 rows x 35 columns]

```
X = australiano_df_copia.loc[:, australiano_df_copia.columns != 'col15']
y = australiano_df_copia['col15']
```

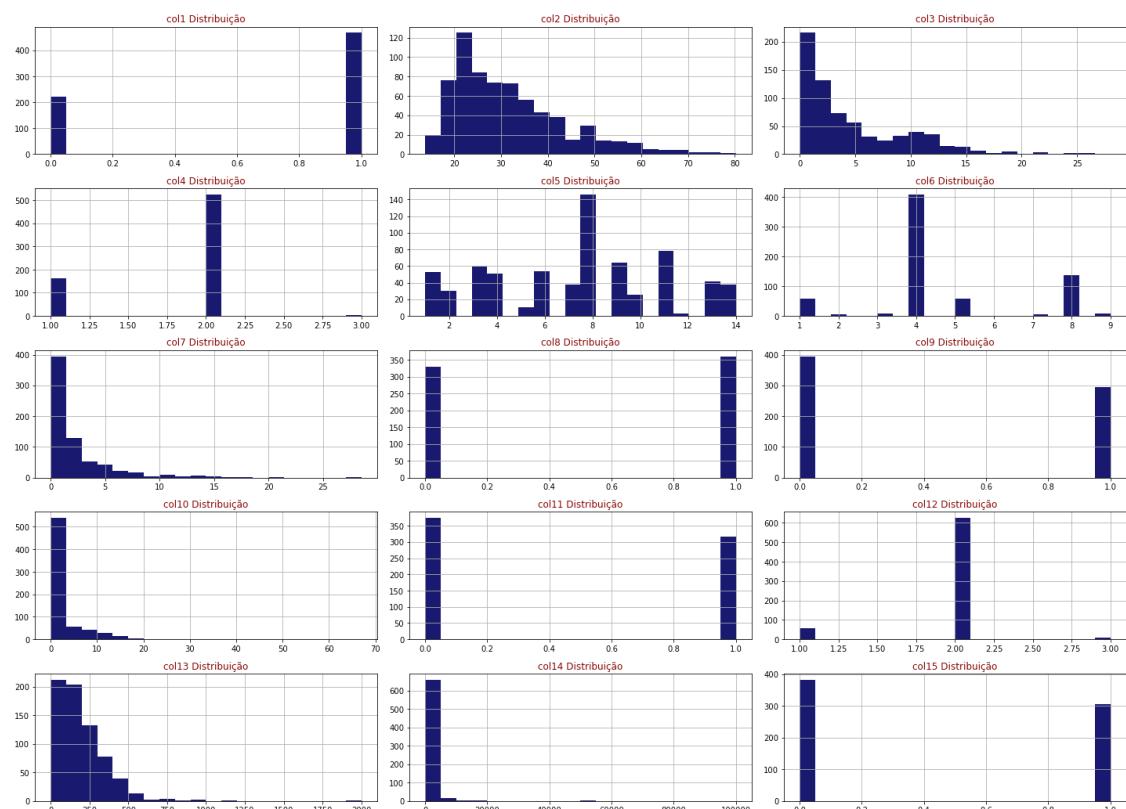
Visualizar a distribuição de cada variável

```
rows = 7
cols = 3

fig = plt.figure(figsize = (20,20))

for i, coluna in enumerate(australiano_df.columns):
    ax = fig.add_subplot(rows, cols, i+1)
    australiano_df[coluna].hist(bins = 20, ax = ax, facecolor = 'midnightblue')
    ax.set_title(coluna + " Distribuição", color = 'DarkRed')

fig.tight_layout()
plt.show()
```



australiano_df.head()

	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
0	1	22.08	11.46	2	4	4	1.585	0	0	0	1
1	0	22.67	7.00	2	8	4	0.165	0	0	0	0
2	0	29.58	1.75	1	4	4	1.250	0	0	0	1
3	0	21.67	11.50	1	5	3	0.000	1	1	11	1
4	1	20.17	8.17	2	6	4	1.960	1	1	14	0

	col12	col13	col14	col15
0	2	100	1213	0
1	2	160	1	0
2	2	280	1	0

3	2	0	1	1
4	2	60	159	1

Seleção de hiperparametros

Random Forest - Nested Cross validation

```
espaco_de_parametros = {
    "n_estimators" : range(70, 130, 10),
    "max_depth" : randint(0, 15),
    "min_samples_split" : randint(1, 10),
    "min_samples_leaf" : randint(0, 10),
    "bootstrap" : [True, False],
    "criterion" : ["gini", "entropy"]
}

# cross validation interno (variando conjunto de validação)
modelo_tree = RandomizedSearchCV(RandomForestClassifier(), espaco_de_parametros, n_iter=10, return_train_score=True,
                                   cv = StratifiedKFold(n_splits = 10, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_tree, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

array([0.86956522, 0.86231884, 0.85507246, 0.86956522, 0.88405797])

print(" Acurácia Média: {:.2%}".format(round(scores.mean(), 3)), "Desvio padrão: {:.2%}".format(round(statistics.stdev(scores), 3)))

Acurácia Média: 86.80% Desvio padrão: 1.10%
```

SVM - Nested Cross Validation

```
espaco_de_parametros = {
    'C': [0.1, 0.5, 1, 2, 5],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
}

# cross validation interno (variando conjunto de validação)
modelo_svm = RandomizedSearchCV(svm.SVC(probability=True), espaco_de_parametros, n_iter=10, return_train_score=True,
                                   cv = StratifiedKFold(n_splits = 10, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_svm, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

array([0.82608696, 0.86956522, 0.84782609, 0.84057971, 0.84057971])

print(" Acurácia Média: {:.2%}".format(round(scores.mean(), 3)), "Desvio padrão: {:.2%}".format(round(statistics.stdev(scores), 3)))
```

Acurácia Média: 84.50% Desvio padrão: 1.60%

Regressão logística - Nested Cross Validation

```
espaço_de_parametros = {
    'C': [0.1, 0.5, 1, 2, 5],
    'penalty': ['l1', 'l2'],
    'solver': ['lbfgs', 'liblinear'],
    'max_iter': [50, 100, 200, 300, 500]
}

# cross validation interno (variando conjunto de validação)
modelo_logistic = RandomizedSearchCV(LogisticRegression(), espaço_de_parametros, n_iter=10, return_train_score=True,
                                         cv = StratifiedKFold(n_splits = 10, shuffle=True, random_state=0))

# cross validation externo (variando conjunto de teste)
scores = cross_val_score(modelo_logistic, X, y, cv = StratifiedKFold(n_splits = 5, shuffle=True, random_state=0))
scores

array([0.86956522, 0.85507246, 0.84782609, 0.84057971, 0.89130435])

print("Acurácia Média: {:.2%}".format(round(scores.mean(), 3)), "Desvio padrão: {:.2%}".format(round(statistics.stdev(scores), 3)))
```

Acurácia Média: 86.10% Desvio padrão: 2.00%