

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS NÚCLEO DE
EDUCAÇÃO A DISTÂNCIA**

Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Alexandre Fábio de L.S Sales

**Modelo Preditivo para Aprovação de Cartão de Crédito
Utilizando Algoritmos Classificadores.**

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.

Rio de Janeiro
2023

SUMÁRIO

1. Introdução	2
1.1. Contextualização	2
1.2. O problema proposto	3
2. Coleta de Dados	4
3. Processamento/Tratamento de Dados	5
3.1. Outliers	14
4. Análise e Exploração dos Dados	17
5. Criação de Modelos de Machine Learning	34
6. Apresentação dos Resultados	49
7. Links	51
APÊNDICE	51

1. Introdução

1.1. Contextualização

Cartão de crédito é um dos meios de pagamentos mais populares no Brasil e no Mundo. Segundo a ABECS (Associação Brasileira das Empresas de Cartões de Crédito e Serviços), no Brasil, as transações em cartões foram em torno de R\$ 3,31 trilhões no ano de 2022.

Os primeiros cartões de crédito surgiram nos Estados Unidos na década de 1920 por donos de empreendimentos - hotéis, postos de gasolina, lojas de departamentos - que os davam aos clientes mais fiéis e abonados. Uma espécie de cartão de fidelidade.

Somente em 1949, o esquecimento da carteira para pagar a conta de um jantar de negócios, inspirou a ideia de cartão para restaurantes por um executivo de uma corporação de crédito - Frank Macnamara. Em 1950 foi lançado "The Diners Club" (Clube da Janta), considerado o primeiro cartão de crédito universal. Assim, surgiu um novo conceito em vez das empresas oferecerem crédito, haveria um intermediário, sendo aceito em vários estabelecimentos. No início o cartão era de papel, aceito apenas em restaurantes e pessoas influentes. As empresas que aceitavam aderir ao modelo pagavam um percentual por transação, e os assinantes uma taxa anual.

O desenvolvimento foi rápido, no 3º aniversário existiam 42 mil membros e 330 estabelecimentos no EUA cadastrados. Os associados pagavam US\$3 ano, e os estabelecimentos 7%. No Brasil chegou em 1954.

Várias outras empresas adotaram o novo modelo, e assim foi criada a indústria de cartões de crédito. Algumas bandeiras mais conhecidas do mercado, se originaram a partir do final da década de 50, por exemplo:

1958 – American Express Card

1958 – BankAmericard que se tornou VISA (1976-77)

1966 – Intercard Bank que se tornou Mastercard (1979)

Com o tempo os cartões se popularizaram. Surgem novos formatos com a evolução tecnológica, por exemplo, incorporação de chips, senhas e aproximação. Interessante verificar que ainda tem muito espaço para o crescimento. Segundo o site Statista e Banco Mundial a penetração varia com o país, função da cultura ou necessidade de investimentos das organizações. A média mundial para a população acima de 15 anos é de apenas 10%.

- O Canadá foi um dos três países do mundo em 2021, onde a posse de cartão de crédito entre consumidores com 15 anos ou mais era superior a 70%.

- No Brasil a estimativa é que em torno de 40.43% da população com 15 anos ou mais, possuía um cartão de crédito.

1.2. O problema proposto

No mundo de hoje, onde as transações online se tornaram a norma, a fraude com cartões de crédito tornou-se uma grande preocupação para bancos e instituições financeiras. É essencial que os bancos prevejam se um cliente apresenta um risco de crédito para minimizar as perdas financeiras por inadimplência ou fraude.

Um cartão de crédito é emitido por um banco ou empresa de serviços financeiros que permite aos titulares do cartão emprestar fundos para pagar bens e serviços com comerciantes que aceitam cartões como forma de pagamento. Os cartões de crédito impõem a condição de que os titulares do cartão devolvam o dinheiro emprestado, acrescido de quaisquer juros aplicáveis, bem como quaisquer encargos adicionais acordados, integralmente até a data de cobrança ou ao longo do tempo.

O problema proposto consiste na construção de um algoritmo para auxiliar na tomada de decisão do cálculo de novos clientes de cartão de crédito. O trabalho não deve ser visto como a construção de uma metodologia para definir a aquisição de novos clientes de cartão de crédito, mas sim uma ferramenta que apoia a decisão baseada em dados históricos.

Temos como objetivos dessa análise:

- Realizar a análise descritiva dos dados dos clientes de cartão de crédito;
- Verificar a correlação entre eles;
- Criar modelos preditivos para clientes que apresentem um risco de inadimplência ao adquirir um cartão de crédito, utilizando os algoritmos de classificação Árvore de Decisão, Regressão Logística, Naïve Bayes, Gradiente Descendente, KNN (K - Nearest Neighbors) e Random Forest.

Os dados utilizados neste trabalho contém transações efetuadas com cartões de crédito, para análise exploratória, extraídos do site <https://www.kaggle.com/datasets/>.

2. Coleta de Dados

Para o tratamento do problema proposto, foram utilizados dois dataframes (conjunto de dados) relativos às transações de cartões de crédito, extraídos do site <https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction/data>.

O primeiro dataframe, "application_record.csv", traz as informações cadastrais dos clientes, que podem ser utilizadas como recursos para a previsão. Esse dataframe possui os seguintes campos:

Campo	Descrição	Observação
AMT_INCOME_TOTAL	Rendimento anual	
CNT_CHILDREN	Número de filhos	
CNT_FAM_MEMBERS	Tamanho da família	
CODE_GENDER	Gênero	
DAYS_BIRTH	Aniversário	Conte regressivamente a partir do dia atual (0), -1 significa ontem
DAYS_EMPLOYED	Data de início do emprego	Conte regressivamente a partir do dia atual (0). Se positivo, significa a pessoa atualmente desempregada.
FLAG_EMAIL	Indica se possui um e-mail	
FLAG_MOBIL	Indica se possui um telefone celular	
FLAG_OWN_CAR	Indica se possui um carro	
FLAG_OWN_REALTY	Indica se possui uma propriedade	
FLAG_PHONE	Indica se possui um telefone	
FLAG_WORK_PHONE	Indica se possui um telefone comercial	
ID	Número do cliente	
NAME_EDUCATION_TYPE	Nível de educação	
NAME_FAMILY_STATUS	Estado civil	
NAME_HOUSING_TYPE	Tipo de moradia	
NAME_INCOME_TYPE	Categoria de renda	
OCCUPATION_TYPE	Ocupação	

O segundo dataframe, "credit_record.csv", registra o comportamento dos clientes de cartão de crédito. Esse dataframe possui os seguintes campos:

Campo	Descrição	Observação
ID	Número do cliente	
MONTHS_BALANCE	Meses em que a conta está aberta	0 é o mês atual, -1 é o mês anterior e assim por diante
STATUS	Indica os dias de atraso no pagamento	0: 1 a 29 dias de atraso 1: 30 a 59 dias de atraso 2: 60 a 89 dias de atraso 3: 90 a 119 dias de atraso 4: 120 a 149 dias de atraso 5: Dívidas vencidas ou incobráveis, baixas por mais de 150 dias C: quitado naquele mês X: Nenhum empréstimo no mês

3. Processamento/Tratamento de Dados

O processamento e o tratamento dos dados foram feitos utilizando a linguagem Python, versão 3.8.17, no ambiente Jupyter Notebook, versão 6.5.4. Dentro da linguagem, utilizou-se a biblioteca “pandas” que é uma poderosa ferramenta para tratamento e análise de dados.

Inicialmente foi necessário importar a biblioteca “Pandas” (Figura 8), que é uma biblioteca de código aberto de licença BSD (em inglês, Berkeley Software Distribution) que fornece estruturas de dados de alto desempenho e fáceis de usar, além de ferramentas de análise de dados para a linguagem de programação Python.

```
import pandas as pd # processamento de dados, CSV I/O
```

Em seguida, deve-se fazer a leitura e tratamento dos dataframes, que, nesse caso, será feita individualmente para cada um deles. Os dataframes “application_record.csv” e “credit_record.csv” serão processados e importados para as variáveis “cliente” e “transação” por meio do comando “pd.read.csv”, seguindo os parâmetros estabelecidos nos arquivos com as características dos dataframes.

```
cliente = pd.read_csv("application_record.csv", encoding = 'utf-8')
transacao = pd.read_csv("credit_record.csv", encoding = 'utf-8')
```

O uso do comando “pd.read.csv”, cria uma estrutura bidimensional de dados, como uma planilha, chamada dataframe. A utilização da função “info ()”, apresenta informações sobre os dataframes criados.

```
cliente.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 -
```

```
transacao.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 -
```

Os dataframes criados apresentam as seguintes dimensões:

- Dataframe “cliente” - 438557 linhas, divididas em 18 colunas.
- Dataframe “transacao” - 1048575 linhas, divididas em 3 colunas.

A coluna FLAG_MOBIL não foi mantida nesse estudo, pelo fato de que essa coluna é uma constante e por isso foi desconsiderada. O dataframe “transacao” não apresenta colunas com valores constantes.

```
cliente.nunique()
```

ID	438510
CODE_GENDER	2
FLAG_OWN_CAR	2
FLAG_OWN_REALTY	2
CNT_CHILDREN	12
AMT_INCOME_TOTAL	866
NAME_INCOME_TYPE	5
NAME_EDUCATION_TYPE	5
NAME_FAMILY_STATUS	5
NAME_HOUSING_TYPE	6
DAYS_BIRTH	16379
DAYS_EMPLOYED	9406
FLAG_MOBIL	1
FLAG_WORK_PHONE	2
FLAG_PHONE	2
FLAG_EMAIL	2
OCCUPATION_TYPE	18
CNT_FAM_MEMBERS	13

```
transacao.nunique()
```

ID	45985
MONTHS_BALANCE	61
STATUS	8

Para remover a coluna FLAG_MOBIL, foi utilizado o seguinte código:

```
cliente.drop('FLAG_MOBIL', axis=1, inplace=True)
```

```
cliente.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 17 columns):
```

O dataframe “cliente” apresenta agora 17 colunas com as 438557 entradas filtradas anteriormente. Para verificar se há dados nulos nos dataframes, utilizou-se a função “isnull()”, que faz essa análise, em conjunto com a função “sum()”, que, nesse caso, soma os valores encontrados na função anterior.

```
cliente.isna().sum()
```

```
ID                      0
CODE_GENDER               0
FLAG_OWN_CAR               0
FLAG_OWN_REALTY              0
CNT_CHILDREN               0
AMT_INCOME_TOTAL              0
NAME_INCOME_TYPE              0
NAME_EDUCATION_TYPE              0
NAME_FAMILY_STATUS              0
NAME_HOUSING_TYPE              0
DAYS_BIRTH                  0
DAYS_EMPLOYED                  0
FLAG_WORK_PHONE                  0
FLAG_PHONE                  0
FLAG_EMAIL                  0
OCCUPATION_TYPE          134203
CNT_FAM_MEMBERS                  0
```

```
transacao.isna().sum()
```

```
ID                      0
MONTHS_BALANCE               0
STATUS                      0
```

Após esse comando, verificou-se que o dataframe cliente, apresenta dados nulos na coluna OCCUPATION_TYPE que necessitam de tratamento. Nesse caso, os dados nulos serão substituídos pelo valor 'Other'. Foi utilizada a função “fillna” para a substituição.

```
cliente['OCCUPATION_TYPE'].fillna(value='Other', inplace=True)
```

O número de clientes com ID's ditintos no dataframe “cliente” deveria ser igual ao número de linhas neste dataframe . A verificação de dados duplicados foi confirmada pelo método “unique”.

```
print("Número de linhas no dataset application_record.csv: {}".format(len(cliente)))
print("Número de clientes distintos no dataset application_record.csv : {}".format(len(cliente.ID.unique())))
```

```
Número de linhas no dataset application_record.csv: 438557
Número de clientes distintos no dataset application_record.csv : 438510
```

Para remoção dos registros duplicados foi utilizado o método “drop_duplicates”, mantendo o primeiro registro encontrado.

```
cliente=cliente.drop_duplicates(subset=cliente.columns[1:], keep='first')
```

O dataframe “transação” não apresenta dados duplicados.

```
: print("Número de linhas duplicadas no dataset credit_record.csv : {}".format(transacao.duplicated().sum()))
```

```
Número de linhas duplicadas no dataset credit_record.csv : 0
```

Para modelos de machine learning supervisionados é fundamental a etapa de criação de uma coluna target no dataframe. Em problemas que isso é possível, faz com que os modelos e resultados obtidos possam ser explorados com maior facilidade.

Na maioria das situações, um algoritmo de aprendizado de máquina supervisionado é usado para derivar a variável target. Esse algoritmo usa dados históricos para aprender padrões e descobrir relacionamentos entre outras partes do seu conjunto de dados e a variável target.

O caso em análise busca avaliar a previsibilidade se o usuário é um cliente 'bom' ou 'ruim' com base em alguns parâmetros. Dessa forma, a informação que utilizaremos como resultado é a coluna STATUS, que traz os dias em atraso no pagamento.

Os clientes que se enquadram na categoria “Dados insuficientes” STATUS=X foram aqueles que não utilizaram o cartão de crédito, indicando falta de histórico de transações.

Os clientes considerados “maus pagadores” atrasaram os pagamentos por mais de 30 dias pelo menos uma vez em seu histórico de pagamentos. Isso incluiu clientes com códigos específicos (1, 2, 3, 4 ou 5) em seu histórico mensal.

Por fim, os clientes com histórico consistente de pagamentos pontuais (status “C”) ou atrasos de até 29 dias (status “0”) foram categorizados como de

baixo risco e considerados perfis adequados para aprovação de cartão de crédito.

```
transacao['STATUS'].value_counts()
```

```
STATUS
C    442031
0    383120
X    209230
1    11090
5     1693
2      868
3      320
4      223
Name: count, dtype: int64
```

```
transacao['target']=transacao['STATUS']
transacao['target'].replace('X', -1, inplace=True)
transacao['target'].replace('C', 0, inplace=True)
transacao['target']=transacao['target'].astype(int)
transacao.loc[transacao['target']>=1, 'target']=1
```

```
transacao['target'].value_counts()
```

```
target
0    825151
-1   209230
1    14194
Name: count, dtype: int64
```

O valor de target = -1 será eliminado da construção do modelo porque não ocorreu a utilização do cartão.

```
transacao = transacao[transacao['target']!= -1]
```

O objetivo agora é criar um único dataframe consolidado (new_df), gerado a partir da integração (join/merge) dos dataframes gerados. Para isso, foi necessário agrupar o dataframe “transacao”, por meio de seu ID de cliente, através dos comandos “groupby” e “agg(max)” que, em conjunto, selecionam o valor máximo de “target” e o ID para cada cliente.

```

transacao_df=pd.DataFrame(transacao.groupby(['ID'])['target'].agg(max)).reset_index()

transacao_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41449 entries, 0 to 41448
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  -- 
 0   ID      41449 non-null   int64  
 1   target  41449 non-null   int32  
dtypes: int32(1), int64(1)
memory usage: 485.9 KB

transacao_df['target'].value_counts()

target
0    36099
1    5350
Name: count, dtype: int64

# Merge dataframes
new_df=pd.merge(cliente, transacao_df, how='inner', on=['ID'])

new_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33110 entries, 0 to 33109
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype    
---  -- 
 0   ID              33110 non-null   int64    
 1   CODE_GENDER      33110 non-null   object    
 2   FLAG_OWN_CAR     33110 non-null   object    
 3   FLAG_OWN_REALTY  33110 non-null   object    
 4   CNT_CHILDREN     33110 non-null   int64    
 5   AMT_INCOME_TOTAL 33110 non-null   float64  
 6   NAME_INCOME_TYPE 33110 non-null   object    
 7   NAME_EDUCATION_TYPE 33110 non-null   object    
 8   NAME_FAMILY_STATUS 33110 non-null   object    
 9   NAME_HOUSING_TYPE 33110 non-null   object    
 10  DAYS_BIRTH       33110 non-null   int64    
 11  DAYS_EMPLOYED    33110 non-null   int64    
 12  FLAG_WORK_PHONE  33110 non-null   int64    
 13  FLAG_PHONE        33110 non-null   int64    
 14  FLAG_EMAIL        33110 non-null   int64    
 15  OCCUPATION_TYPE   33110 non-null   object    
 16  CNT_FAM_MEMBERS   33110 non-null   float64  
 17  target            33110 non-null   int32  
dtypes: float64(2), int32(1), int64(7), object(8)
memory usage: 4.4+ MB

```

O próximo passo é incluir a coluna “MONTHS_BALANCE”, que foi removida no merge anterior. Faz sentido que o número de meses em que a conta está aberta, seja correlacionado com o risco do cliente (uma vez que há mais oportunidades de perder pagamentos).

Como a coluna “MONTHS_BALANCE” apresenta valores negativos, informando o número de meses em que a conta está aberta, foi selecionado o valor mínimo da coluna para cada ID. Após a seleção, a coluna será renomeada

para “ACCOUNT_LENGTH”, os valores transformados para positivo e incluídos no dataframe “new_df”.

```
# Meses em que a conta está aberta
inicio_df=pd.DataFrame(transacao.groupby(['ID'])['MONTHS_BALANCE'].agg(min)).reset_index()

# Renomear a coluna
inicio_df.rename(columns={'MONTHS_BALANCE':'ACCOUNT_LENGTH'}, inplace=True)

# Meses como número positivo
inicio_df['ACCOUNT_LENGTH']=inicio_df['ACCOUNT_LENGTH']

# Merge dataframes
new_df=pd.merge(new_df, inicio_df, how='inner', on=['ID'])

new_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33110 entries, 0 to 33109
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   ID               33110 non-null   int64  
 1   CODE_GENDER       33110 non-null   object  
 2   FLAG_OWN_CAR     33110 non-null   object  
 3   FLAG_OWN_REALTY  33110 non-null   object  
 4   CNT_CHILDREN     33110 non-null   int64  
 5   AMT_INCOME_TOTAL 33110 non-null   float64 
 6   NAME_INCOME_TYPE 33110 non-null   object  
 7   NAME_EDUCATION_TYPE 33110 non-null   object  
 8   NAME_FAMILY_STATUS 33110 non-null   object  
 9   NAME_HOUSING_TYPE 33110 non-null   object  
 10  DAYS_BIRTH        33110 non-null   int64  
 11  DAYS_EMPLOYED    33110 non-null   int64  
 12  FLAG_WORK_PHONE  33110 non-null   int64  
 13  FLAG_PHONE        33110 non-null   int64  
 14  FLAG_EMAIL        33110 non-null   int64  
 15  OCCUPATION_TYPE  33110 non-null   object  
 16  CNT_FAM_MEMBERS  33110 non-null   float64 
 17  target            33110 non-null   int32  
 18  ACCOUNT_LENGTH   33110 non-null   int64  
dtypes: float64(2), int32(1), int64(8), object(8)
memory usage: 4.7+ MB
```

Percebe-se que o novo dataframe possui 33110 linhas. Agora iremos transformar as variáveis “DAYS_BIRTH” e “DAYS_EMPLOYED”, em variáveis que indiquem a idade, os anos de emprego e se o cliente está empregado. As antigas variáveis transformadas serão eliminadas do dataframe.

```

# Criar a variável idade
new_df['AGE_YEARS'] = new_df['DAYS_BIRTH']/365.2425
new_df.drop('DAYS_BIRTH', axis=1, inplace=True)

# Criar um indicador se está empregado ou não
new_df['EMPLOYED'] = 'S'
new_df.loc[~new_df['DAYS_EMPLOYED'] < 0, 'EMPLOYED'] = 'N'

# Criar a variável anos em que está empregado
new_df['YEARS_EMPLOYED'] = new_df['DAYS_EMPLOYED']/365.2425
new_df.loc[new_df['YEARS_EMPLOYED'] < 0, 'YEARS_EMPLOYED'] = 0
new_df.drop('DAYS_EMPLOYED', axis=1, inplace=True)

```

Como o dataframe apresenta os nomes das colunas na língua inglesa, a tradução para a língua portuguesa foi realizada para um melhor entendimento.

```

new_df = new_df.rename(columns={'CODE_GENDER': 'COD_SEXO', 'FLAG_OWN_CAR': 'COD_TEM_CARRO',
                               'FLAG_OWN_REALTY': 'COD_TEM_IMOVEL', 'CNT_CHILDREN': 'NUM_FILHOS',
                               'AMT_INCOME_TOTAL': 'RENTA', 'NAME_INCOME_TYPE': 'TIPO_RENDA',
                               'NAME_EDUCATION_TYPE': 'NIVEL_ESCOLARIDADE', 'NAME_FAMILY_STATUS': 'ESTADO_CIVIL',
                               'NAME_HOUSING_TYPE': 'TIPO_MORADIA', 'FLAG_WORK_PHONE': 'COD_TEL_TRABALHO',
                               'FLAG_PHONE': 'COD_TEM_TELEFONE', 'FLAG_EMAIL': 'COD_TEM_EMAIL', 'OCCUPATION_TYPE': 'PROFISSAO',
                               'CNT_FAM_MEMBERS': 'NUM_MEMBROS_FAMILIA', 'target': 'RESULTADO', 'ACCOUNT_LENGTH': 'TEMPO_DA_CONTA',
                               'AGE_YEARS': 'IDADE', 'EMPLOYED': 'COD_TEM_EMPREGO', 'YEARS_EMPLOYED': 'ANOS_EMPREGADO'})

```

A transformação dos valores das colunas COD_SEXO, COD_TEM_CARRO, COD_TEM_IMOVEL, TIPO_RENDA, TIPO_MORADIA, NIVEL_ESCOLARIDADE, ESTADO_CIVIL e PROFISSAO do tipo “object” para a língua portuguesa também foi realizada. A coluna NUM_MEMBROS_FAMILIA foi transformada para o tipo int, correspondendo aos valores apresentados. O uso da função “unique()” apresenta os valores distintos das colunas que serão traduzidas para língua portuguesa.

```

print(new_df['PROFISSAO'].unique())

['Other' 'Security staff' 'Sales staff' 'Accountants' 'Laborers'
 'Managers' 'Drivers' 'Core staff' 'High skill tech staff'
 'Cleaning staff' 'Private service staff' 'Cooking staff'
 'Low-skill Laborers' 'Medicine staff' 'Secretaries'
 'Waiters/barmen staff' 'HR staff' 'Realty agents' 'IT staff']

new_df['COD_SEXO'] = new_df['COD_SEXO'].replace(['M'], 'H')
new_df['COD_SEXO'] = new_df['COD_SEXO'].replace(['F'], 'M')

new_df['COD_TEM_CARRO'] = new_df['COD_TEM_CARRO'].replace(['Y'], 'S')
new_df['COD_TEM_IMOVEL'] = new_df['COD_TEM_IMOVEL'].replace(['Y'], 'S')

new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Commercial associate'], 'Sócio Comercial')
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Working'], 'Trabalhador da Área Privada')
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Pensioner'], 'Pensionista')
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['State servant'], 'Servidor Público')
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Student'], 'Estudante')

```

```

new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Security staff'], 'Equipe de segurança')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Sales staff'], 'Equipe de vendas')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Accountants'], 'Contadores')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Laborers'], 'Operários')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Managers'], 'Gerentes')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Drivers'], 'Motoristas')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Core staff'], 'Equipe principal')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['High skill tech staff'], 'Equipe técnica altamente qualificada')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Cleaning staff'], 'Pessoal de limpeza')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Private service staff'], 'Pessoal de serviço privado')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Cooking staff'], 'Pessoal de cozinha')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Low-skill Laborers'], 'Trabalhadores pouco qualificados')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Secretaries'], 'Secretários')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Medicine staff'], 'Equipe médica')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Waiters/barmen staff'], 'Equipe de garçons barmen')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['HR staff'], 'Equipe de RH')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Realty agents'], 'Corretor imobiliário')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['IT staff'], 'Equipe de TI')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Other'], 'Outro')

new_df['NUM_MEMBROS_FAMILIA']=new_df['NUM_MEMBROS_FAMILIA'].astype(int)

```

3.1 Outliers

Um outlier é uma observação que está a uma distância anormal de outros valores em uma amostra aleatória de uma população. Eles podem aparecer devido a erros na entrada de dados ou medição, ou apenas porque há variação na população que você está observando. Seja qual for o motivo pelo qual aparecem, saber como identificar e lidar com valores discrepantes é uma parte importante da limpeza de dados.

Os valores discrepantes serão removidos pelo método IQR, também chamado de "intervalo interquartil". Primeiro classificando os elementos em ordem crescente e depois encontrando Q1, Q3 e iqr pela diferença de Q3 e Q1. Finalmente, definindo o limite superior e inferior e removendo-os.

Uma das maneiras de ver se o conjunto de dados tem valores discrepantes é traçar os dados em gráfico. A função "go.Box" será utilizada para apresentar um gráfico de caixa que é uma representação estatística da distribuição de uma variável através de seus quartis. As extremidades da caixa representam os quartis inferior e superior, enquanto a mediana (segundo quartil) é marcada por uma linha dentro da caixa. As colunas que informam a renda, idade, anos de emprego, membros da família, número de filhos e tempo como cliente serão analisadas.

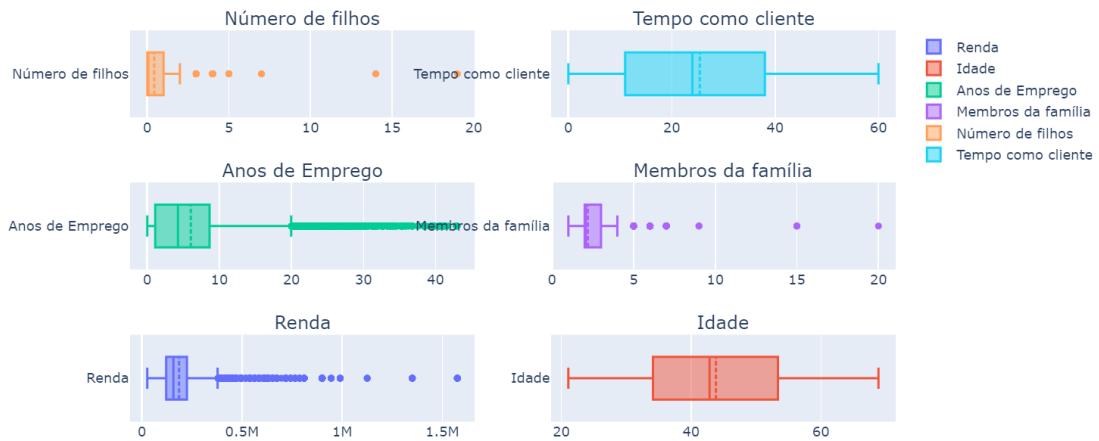
```

: fig = make_subplots(rows=3, cols=2, start_cell="bottom-left",
                      subplot_titles=("Renda", "Idade", "Anos de Emprego", "Membros da família", "Número de filhos", "Tempo como cliente"))

fig.add_trace(go.Box(x=new_df.RENDA, name='Renda', boxmean=True), row=1, col=1)
fig.add_trace(go.Box(x=new_df.IDADE, name='Idade', boxmean=True), row=1, col=2)
fig.add_trace(go.Box(x=new_df.ANOS_EMPREGADO, name='Anos de Emprego', boxmean=True), row=2, col=1)
fig.add_trace(go.Box(x=new_df.NUM_MEMBROS_FAMILIA, name="Membros da família", boxmean=True), row=2, col=2)
fig.add_trace(go.Box(x=new_df.NUM_FILHOS, name="Número de filhos", boxmean=True), row=3, col=1)
fig.add_trace(go.Box(x=new_df.TEMPO_DA_CONTA, name="Tempo como cliente", boxmean=True), row=3, col=2)

fig.show()

```



```

outliers = find_outliers_IQR(new_df.NUM_MEMBROS_FAMILIA)

print("número de outliers NUM_MEMBROS_FAMILIA : " + str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

outliers = find_outliers_IQR(new_df.ANOS_EMPREGADO)
print("número de outliers ANOS_EMPREGADO : " + str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

outliers = find_outliers_IQR(new_df.NUM_FILHOS)
print("número de outliers NUM_FILHOS : " + str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

outliers = find_outliers_IQR(new_df.RENDA)
print("número de outliers RENDA : " + str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

```

```

número de outliers NUM_MEMBROS_FAMILIA : 430
max outlier : 20
min outlier : 5
número de outliers ANOS_EMPREGADO : 1595
max outlier : 43.02073280081042
min outlier : 20.01410022108599
número de outliers NUM_FILHOS : 453
max outlier : 19
min outlier : 3
número de outliers RENDA : 1342
max outlier : 1575000.0
min outlier : 382500.0

```

Pode-se observar que existem outliers para o número de filhos, anos de emprego, membros da família e renda. A função “find_outliers_IQR” foi utilizada para descobrir os outliers e a substituição dos outliers por valores médios, será feita pela função “impute_outliers_IQR”.

```
# Função para descobrir os outliers
def find_outliers_IQR(df):

    q1=df.quantile(0.25)

    q3=df.quantile(0.75)

    IQR=q3-q1

    outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))] 

    return outliers
```

```
# Função para substituir os outliers por valores médios
def impute_outliers_IQR(df):

    q1=df.quantile(0.25)

    q3=df.quantile(0.75)

    IQR=q3-q1

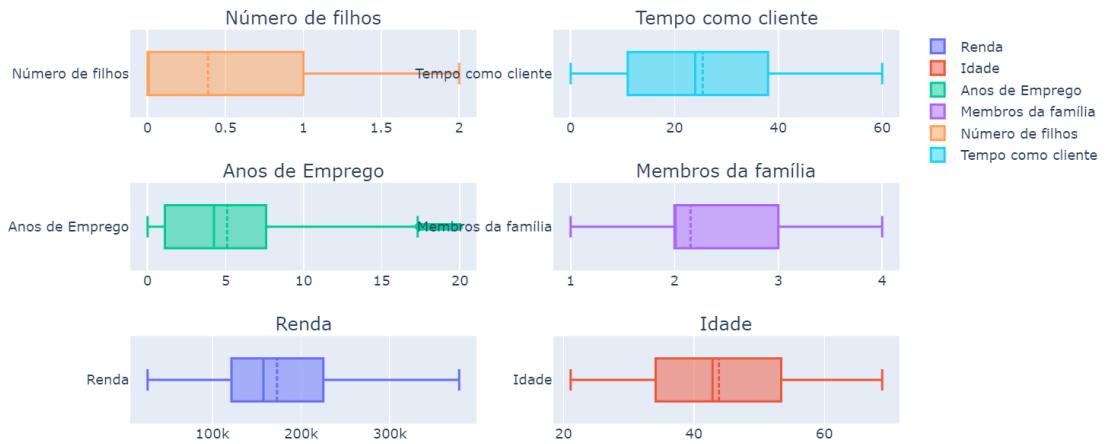
    upper = df[~(df>(q3+1.5*IQR))].max()
    lower = df[~(df<(q1-1.5*IQR))].min()

    df = np.where(df > upper,
                  df.mean(),
                  np.where(
                      df < lower,
                      df.mean(),
                      df
                  )
                )

    return df
```

```
: new_df['NUM_MEMBROS_FAMILIA'] = impute_outliers_IQR(new_df.NUM_MEMBROS_FAMILIA)
new_df['RENDAA'] = impute_outliers_IQR(new_df.RENDAA)
new_df['ANOS_EMPREGADO'] = impute_outliers_IQR(new_df.ANOS_EMPREGADO)
new_df['NUM_FILHOS'] = impute_outliers_IQR(new_df.NUM_FILHOS)
```

Após a substituição dos outliers, o gráfico de caixa não apresenta valores discrepantes.



4. Análise e Exploração de Dados

O objetivo da análise é verificar a discrepância de característica entre os clientes que possuem cartão de crédito e os que efetivamente foram considerados como confiáveis. Sempre que possível, as informações serão separadas nesses critérios e, para isso, a partir do dataframe “new_df” será criado o dataframe “clientes_confiaveis” apenas com os clientes confiáveis.

```
: #Criação de um dataframe df_consulta apenas com clientes confiáveis
clientes_confiaveis = new_df[(new_df['RESULTADO']==0)]
```

```
: clientes_confiaveis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28819 entries, 2 to 32757
Data columns (total 20 columns):
```

Observa-se que o dataframe “clientes_confiaveis” possui 28819 linhas, o que corresponde a um grande número em relação ao total de linhas no dataframe original. Inicialmente a análise será feita pelo gênero dos clientes, fazendo a contagem do total de clientes por gênero, como também, a contagem por gênero dos clientes confiáveis.

Faremos a importação da biblioteca “Collections”, com o uso da função “Counter”, que conta quantas vezes uma determinada opção aparece em uma série de dados a armazena esses valores em um dicionário.

```
#Contagem das opções da coluna COD_SEXO
from collections import Counter
genero = Counter(new_df['COD_SEXO'])
genero

Counter({'H': 10890, 'M': 22220})
```

```
#Contagem das opções da coluna COD_SEXO dos clientes confiáveis
genero_clientes_confiaveis = Counter(clientes_confiaveis['COD_SEXO'])
genero_clientes_confiaveis

Counter({'H': 9357, 'M': 19462})
```

Para facilitar a visualização das informações sobre cada gênero, através biblioteca “Matplotlib” para a criação de gráficos, serão criados os gráficos de pizza com os percentuais e gêneros.

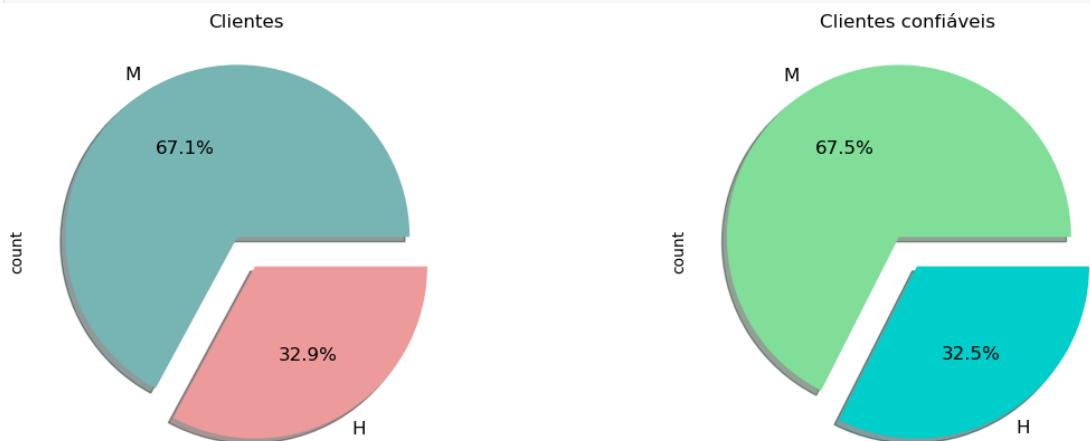
```
import matplotlib.pyplot as plt
```

```
#Plotagem das informações de gênero dos clientes
fig, axes = plt.subplots(1,2)

g1= new_df['COD_SEXO'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%.1f%%', shadow=True, colors=["#76B5B3", "#EC9B9A"])
g1.set_title("Clientes")

g2= clientes_confiaveis['COD_SEXO'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%.1f%%', shadow=True, colors=["#80DEBD", "#80DEBD"])
g2.set_title("Clientes confiáveis")

fig.set_size_inches(14,5)
```



Analizando os gráficos pode-se perceber que o percentual de cada gênero é muito semelhante entre o total dos clientes e os clientes confiáveis. Em ambos os gráficos, a maioria do percentual é do sexo feminino com valores de 67.1% e 67.5%.

O próximo item a ser analisado é a idade dos clientes. Para essa análise será utilizada a função “describe” que apresenta os principais indicadores estatísticos de uma série de dados. Na visualização da distribuição dessas

idades serão utilizados histogramas, seguindo os mesmos estilos descritos anteriormente.

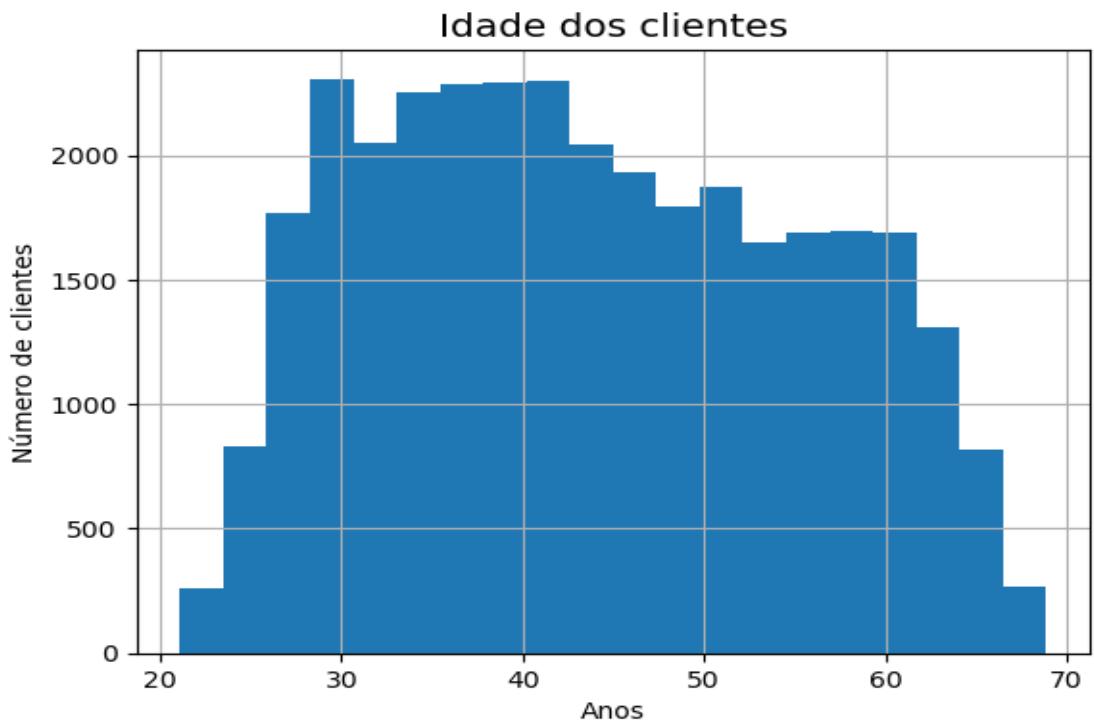
```
#Descrição estatística das idades dos clientes  
new_df['IDADE'].describe()
```

```
count    33110.000000  
mean      43.826396  
std       11.535864  
min       21.095573  
25%      34.117059  
50%      42.837293  
75%      53.375497  
max       68.863837  
Name: IDADE, dtype: float64
```

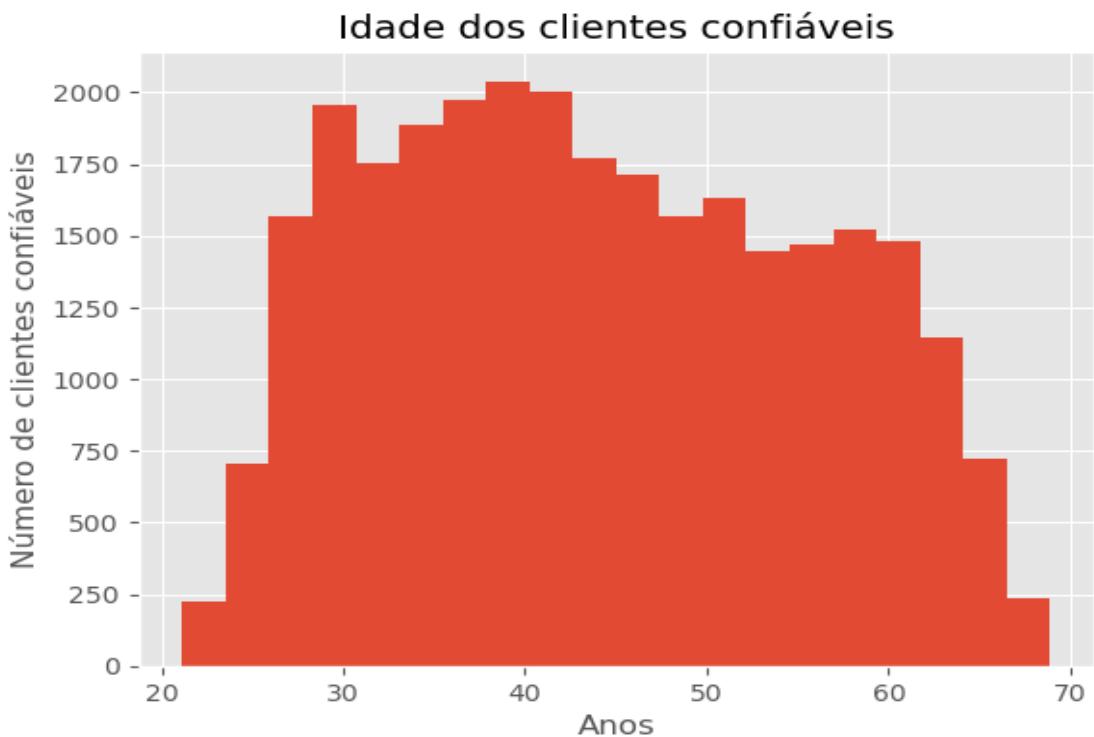
```
#Descrição estatística das idades dos clientes confiáveis  
clientes_confiaveis['IDADE'].describe()
```

```
count    28819.000000  
mean      43.975172  
std       11.531536  
min       21.144856  
25%      34.286809  
50%      43.086443  
75%      53.619171  
max       68.863837  
Name: IDADE, dtype: float64
```

```
#Plotagem de histograma com as idades dos clientes  
new_df.IDADE.hist(bins=20)  
plt.style.use('ggplot')  
plt.xlabel("Anos")  
plt.ylabel("Número de clientes")  
plt.title("Idade dos clientes")  
plt.show()
```



```
#Plotagem de histograma com as idades dos clientes confiáveis
clientes_confiaveis.IDADE.hist(bins=20)
plt.style.use('seaborn-pastel')
plt.xlabel("Anos")
plt.ylabel("Número de clientes confiáveis")
plt.title("Idade dos clientes confiáveis")
plt.show()
```

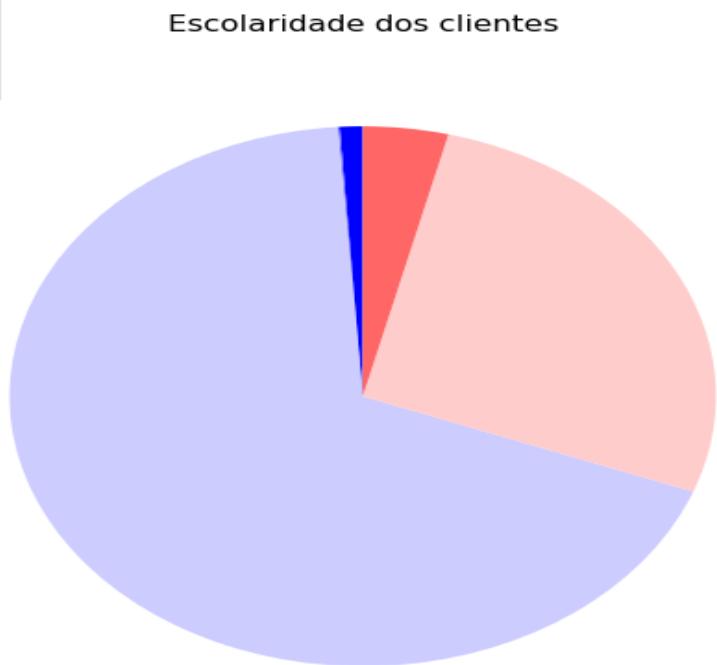


Analizando os histogramas, não se verifica nenhuma mudança significativa. Essa constatação pode ser confirmada por meio dos indicadores estatísticos das duas séries de dados que são muito similares, com variações mínimas na média de idade, por exemplo: 43,82 anos para todos os clientes e 43,97 anos para os clientes confiáveis.

A próxima análise será o nível de escolaridade dos clientes. Utilizaremos novamente a função “Counter”, para determinar quantas vezes um nível de escolaridade aparece no dataframe, e o gráfico de pizza com os valores e percentuais dos níveis de escolaridade.

```
#Contagem dos níveis de escolaridade dos clientes  
  
escolaridade_clientes = Counter(new_df['NIVEL_ESCOLARIDADE'].sort_values())  
escolaridade_clientes  
  
Counter({'Médio': 344,  
         'Pós Graduado': 29,  
         'Secundário': 22554,  
         'Superior': 8891,  
         'Superior Incompleto': 1292})
```

Médio: 1.04%
Pós Graduado: 0.09%
Secundário: 68.12%
Superior: 26.85%
Superior Incompleto: 3.9%



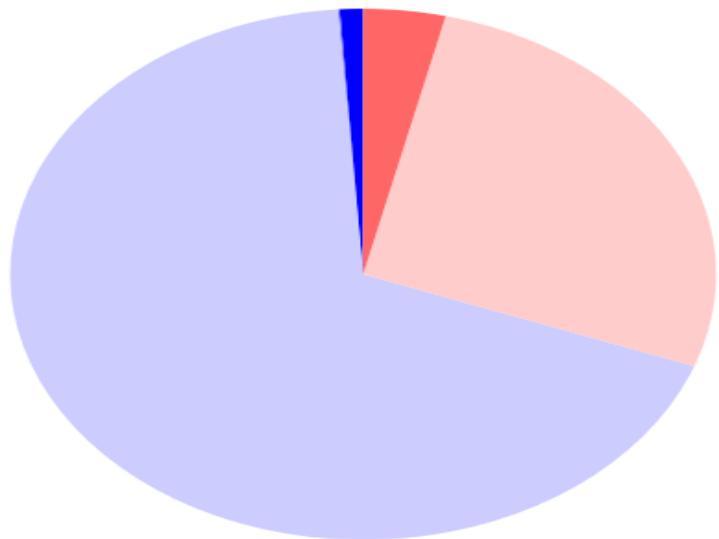
```
#Contagem dos níveis de escolaridade dos clientes confiáveis
```

```
escolaridade_clientes_confiaveis = Counter(clientes_confiaveis['NIVEL_ESCOLARIDADE'].sort_values())
escolaridade_clientes_confiaveis
```

```
Counter({'Médio': 305,
         'Pós Graduado': 22,
         'Secundário': 19664,
         'Superior': 7743,
         'Superior Incompleto': 1085})
```

■ Médio: 1.06%
■ Pós Graduado: 0.08%
■ Secundário: 68.23%
■ Superior: 26.87%
■ Superior Incompleto: 3.76%

Escolaridade dos clientes confiáveis



Percebe-se uma grande semelhança entre os níveis de escolaridade de todos os clientes e os clientes confiáveis. É importante destacar que a grande maioria dos clientes são do nível secundário.

A próxima análise será o estado civil. Os passos utilizados para a análise dos níveis de escolaridade serão semelhantes para o estado civil dos clientes, ou seja, a contagem dos valores nos dataframes e a apresentação dos gráficos de pizza. Porém, foi criado um laço de repetição “for” para apresentação dos percentuais de cada estado civil.

```
#Contagem do estado civil dos clientes
```

```
estado_civil_clientes = Counter(new_df['ESTADO_CIVIL'].sort_values())
estado_civil_clientes
```

```
Counter({'Casado': 22725,
         'Casado no civil': 2656,
         'Separado': 1908,
         'Solteiro': 4433,
         'Viúvo': 1388})
```

```
#Identificação dos percentuais do estado civil dos clientes
n_estado_civil=sum(estado_civil_clientes.values())
for x, y in estado_civil_clientes.items():
    a = y/n_estado_civil*100
    print(str(x) + ': ' + '\n' + str(round(a,2)) + '%' + '\n')
```

Casado:

68.63%

Casado no civil:

8.02%

Separado:

5.76%

Solteiro:

13.39%

Viúvo:

4.19%

```
#Contagem dos níveis do estado civil dos clientes confiáveis

estado_civil_clientes_confiaveis = Counter(clientes_confiaveis['ESTADO_CIVIL'].sort_values())
estado_civil_clientes_confiaveis

Counter({'Casado': 19811,
         'Casado no civil': 2289,
         'Separado': 1683,
         'Solteiro': 3810,
         'Viúvo': 1226})
```

```
#Identificação dos percentuais do estado civil dos clientes confiáveis
n_estado_civil_clientes_confiaveis=sum(estado_civil_clientes_confiaveis.values())
for x, y in estado_civil_clientes_confiaveis.items():
    a = y/n_estado_civil_clientes_confiaveis*100
    print(str(x) + ': ' + '\n' + str(round(a,2)) + '%' + '\n')
```

Casado:

68.74%

Casado no civil:

7.94%

Separado:

5.84%

Solteiro:

13.22%

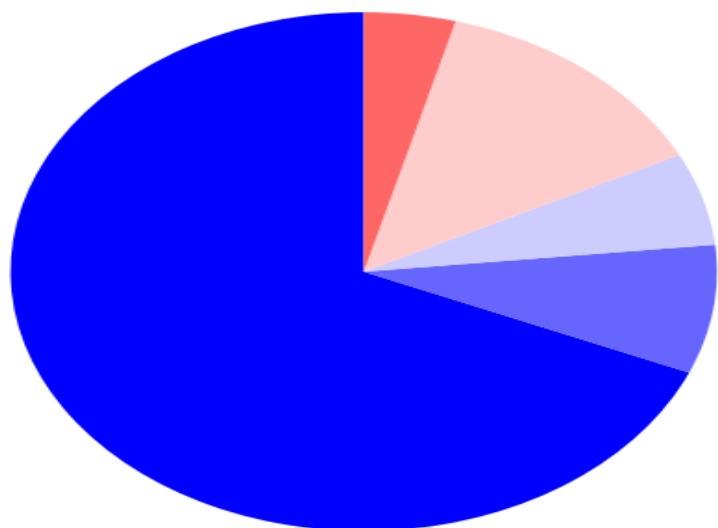
Viúvo:

4.25%

Os resultados da análise do estado civil também serão apresentados em gráficos.

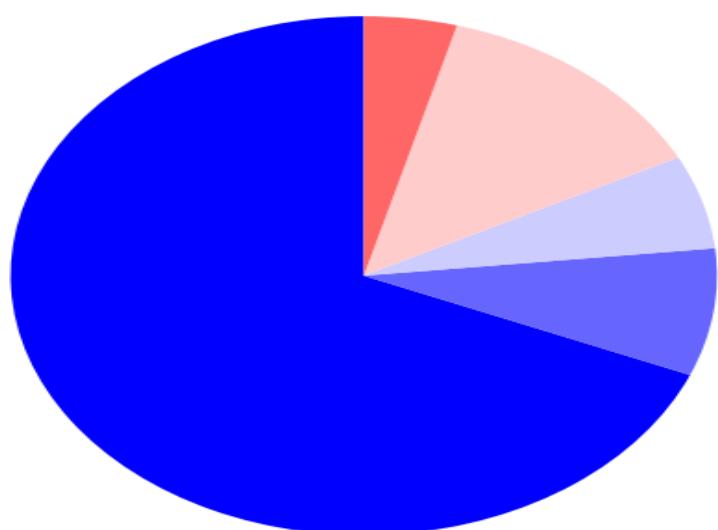
■ Casado: 68.63%
■ Casado no civil: 8.02%
■ Separado: 5.76%
■ Solteiro: 13.39%
■ Viúvo: 4.19%

Estado civil dos clientes



■ Casado: 68.74%
■ Casado no civil: 7.94%
■ Separado: 5.84%
■ Solteiro: 13.22%
■ Viúvo: 4.25%

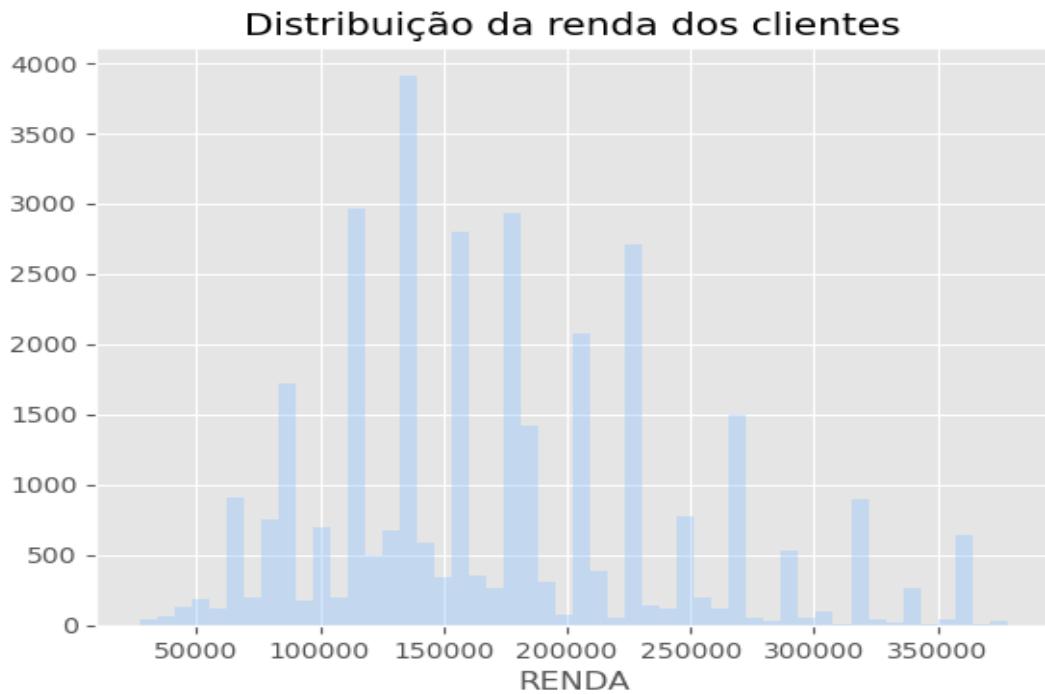
Estado civil dos clientes confiáveis



Percebe-se por meio dos gráficos e percentuais que a maioria dos clientes é casado, com 68,63%, número semelhante quando avaliamos os clientes confiáveis, com 68,74%.

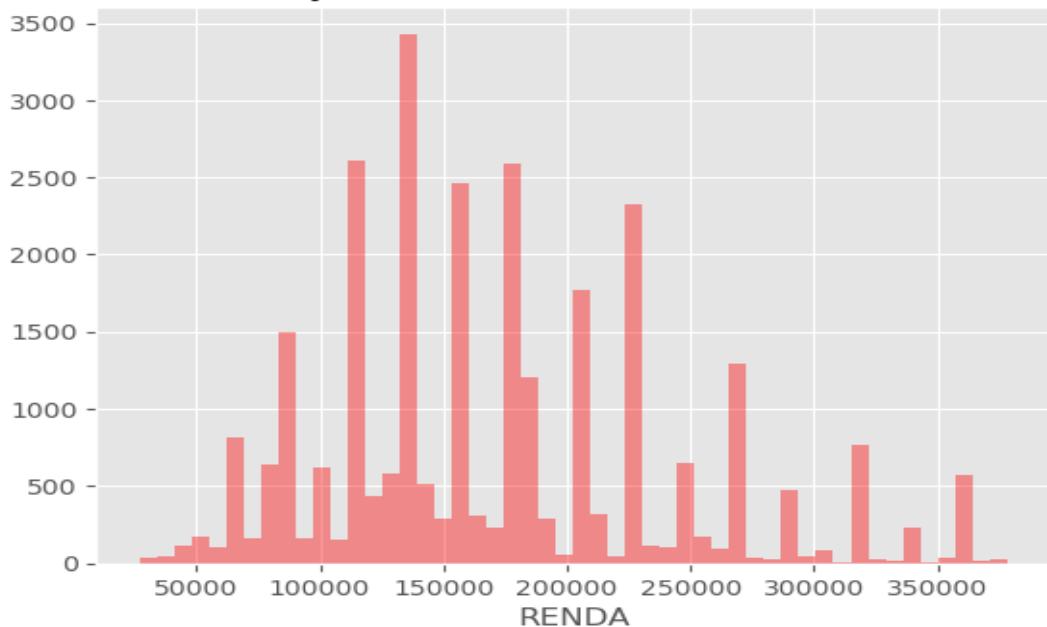
Para a distribuição de renda, anos de emprego e a posse de imóvel, carro e emprego, os percentuais entre o total de clientes e os clientes confiáveis estão também semelhantes, conforme os gráficos apresentados.

```
#Plotagem do histograma da renda dos clientes
sns.distplot(new_df.RENDA, kde=False)
plt.title('Distribuição da renda dos clientes')
```



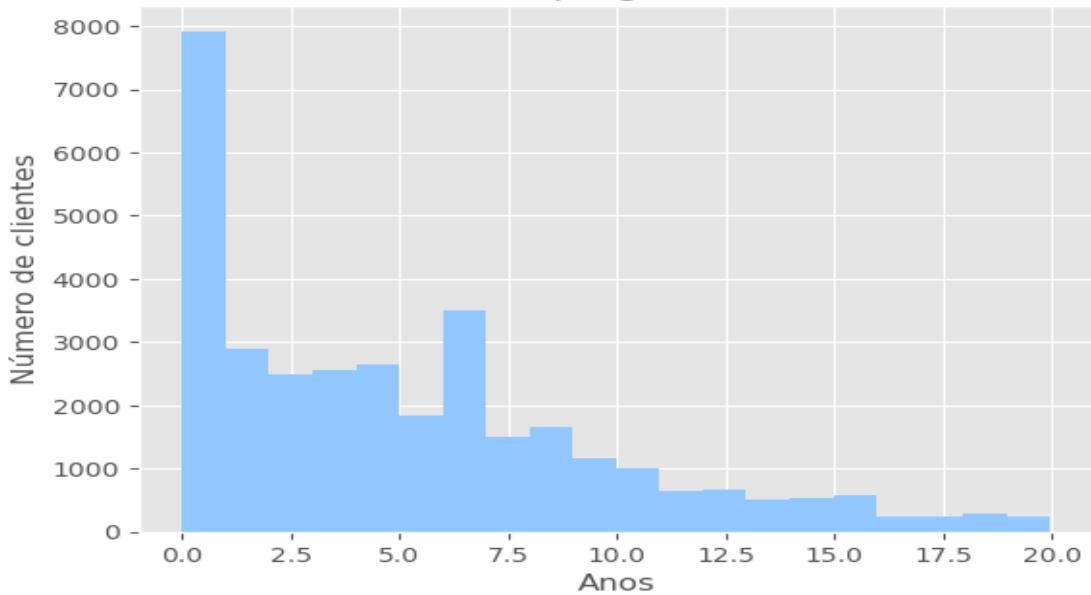
```
#Plotagem do histograma da renda dos clientes confiáveis
sns.distplot(clientes_confiaveis.RENDA, kde=False, color='red')
plt.title('Distribuição da renda dos clientes confiáveis')
```

Distribuição da renda dos clientes confiáveis

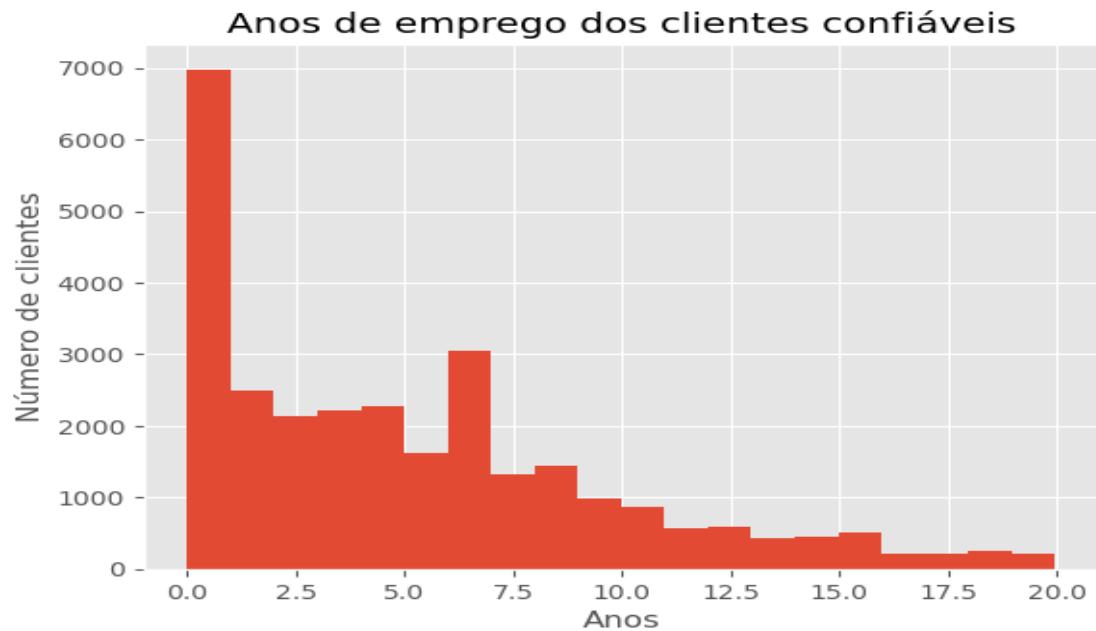


```
#Plotagem de histograma com os anos empregado dos clientes
new_df.ANOS_EMPREGADO.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Anos")
plt.ylabel("Número de clientes")
plt.title("Anos de emprego dos clientes")
plt.show()
```

Anos de emprego dos clientes



```
#Plotagem de histograma com os anos empregado dos clientes confiáveis
clientes_confiaveis.ANOS_EMPREGADO.hist(bins=20)
plt.style.use('seaborn-pastel')
plt.xlabel("Anos")
plt.ylabel("Número de clientes")
plt.title("Anos de emprego dos clientes confiáveis")
plt.show()
```



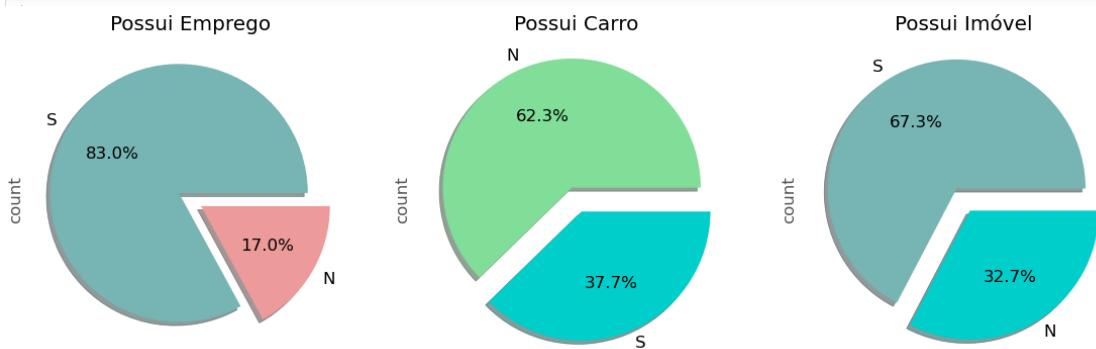
```
#Percentuais de emprego, carro e imóvel dos clientes
fig, axes = plt.subplots(1,3)

g1= new_df['COD_TEM_EMPREGO'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%1.1f%%', shadow=True, colors=["#7685B3"], g1.set_title("Possui Emprego"))

g2= new_df['COD_TEM_CARRO'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%1.1f%%', shadow=True, colors=["#80DE99", "#008080"], g2.set_title("Possui Carro"))

g3= new_df['COD_TEM_IMOVEL'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%1.1f%%', shadow=True, colors=["#7685B3", "#008080"], g3.set_title("Possui Imóvel"))

fig.set_size_inches(14,5)
```





Ao analisar a profissão, podemos perceber que há um grande percentual para as profissões que não foram categorizadas, aproximadamente 31%, para todos os clientes e os clientes confiáveis.

```
: #Contagem da profissão dos clientes
```

```
profissao_clientes = Counter(new_df['PROFISSAO'].sort_values())
profissao_clientes
```

```
Counter({'Contadores': 1106,
          'Corretor imobiliário': 67,
          'Equipe de RH': 76,
          'Equipe de TI': 57,
          'Equipe de garçons barmen': 163,
          'Equipe de segurança': 547,
          'Equipe de vendas': 3156,
          'Equipe médica': 1124,
          'Equipe principal': 3250,
          'Equipe técnica altamente qualificada': 1269,
          'Gerentes': 2695,
          'Motoristas': 1934,
          'Operários': 5601,
          'Outro': 10373,
          'Pessoal de cozinha': 587,
          'Pessoal de limpeza': 511,
          'Pessoal de serviço privado': 293,
          'Secretários': 142,
          'Trabalhadores pouco qualificados': 159})
```

```

#Percentuais da profissão dos clientes

labels = []
sizes = []
n_clientes=sum(profissao_clientes.values())
for x, y in profissao_clientes.items():
    a = y/n_clientes*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a,2))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

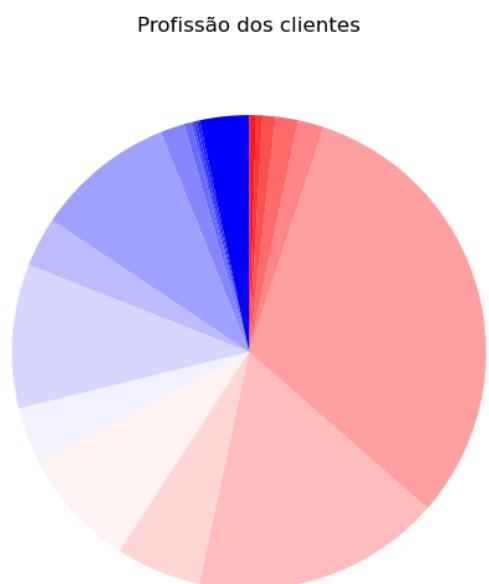
theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in range(len(sizes))])
_, _ = ax1.pie(sizes, startangle=90)

ax1.axis('equal')

total = sum(sizes)
plt.legend(
    loc='center right',
    labels=labels,
    prop={'size': 8},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
)
plt.title("Profissão dos clientes", fontsize = 12, loc = 'center')
plt.show()

```

Contadores: 3.34%
Corretor imobiliário: 0.2%
Equipe de RH: 0.23%
Equipe de TI: 0.17%
Equipe de garçons barman: 0.49%
Equipe de segurança: 1.65%
Equipe de vendas: 9.53%
Equipe médica: 3.39%
Equipe principal: 9.82%
Equipe técnica altamente qualificada: 3.83%
Gerentes: 8.14%
Motoristas: 5.84%
Operários: 16.92%
Outro: 31.33%
Pessoal de cozinha: 1.77%
Pessoal de limpeza: 1.54%
Pessoal de serviço privado: 0.88%
Secretários: 0.43%
Trabalhadores pouco qualificados: 0.48%



```

#Contagem da profissão dos clientes confiáveis

profissao_clientes_confiaveis = Counter(clientes_confiaveis['PROFISSAO'].sort_values())
profissao_clientes_confiaveis

Counter({'Contadores': 959,
         'Corretor imobiliário': 57,
         'Equipe de RH': 62,
         'Equipe de TI': 46,
         'Equipe de garçons barmen': 144,
         'Equipe de segurança': 456,
         'Equipe de vendas': 2767,
         'Equipe médica': 961,
         'Equipe principal': 2787,
         'Equipe técnica altamente qualificada': 1088,
         'Gerentes': 2305,
         'Motoristas': 1670,
         'Operários': 4871,
         'Outro': 9171,
         'Pessoal de cozinha': 501,
         'Pessoal de limpeza': 448,
         'Pessoal de serviço privado': 271,
         'Secretários': 129,
         'Trabalhadores pouco qualificados': 126})

#Percentuais da profissão dos clientes

labels = []
sizes = []
n_clientes=sum(profissao_clientes_confiaveis.values())
for x, y in profissao_clientes_confiaveis.items():
    a = y/n_clientes*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in range(len(sizes))])

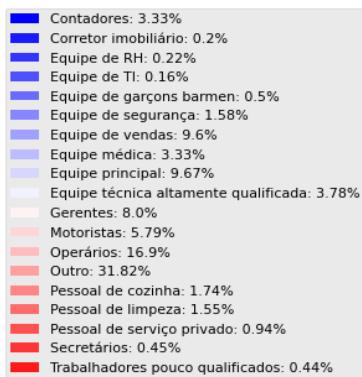
_, _ = ax1.pie(sizes, startangle=90)

ax1.axis('equal')

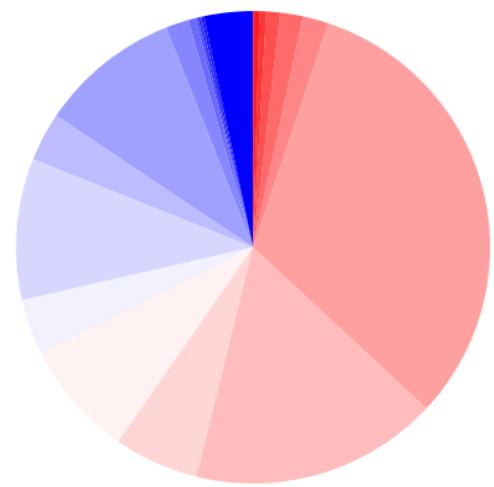
total = sum(sizes)
plt.legend(
    loc='center right',
    labels=labels,
    prop={'size': 8},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
)

plt.title("Profissão dos clientes confiáveis", fontsize = 12, loc = 'center')
plt.show()

```



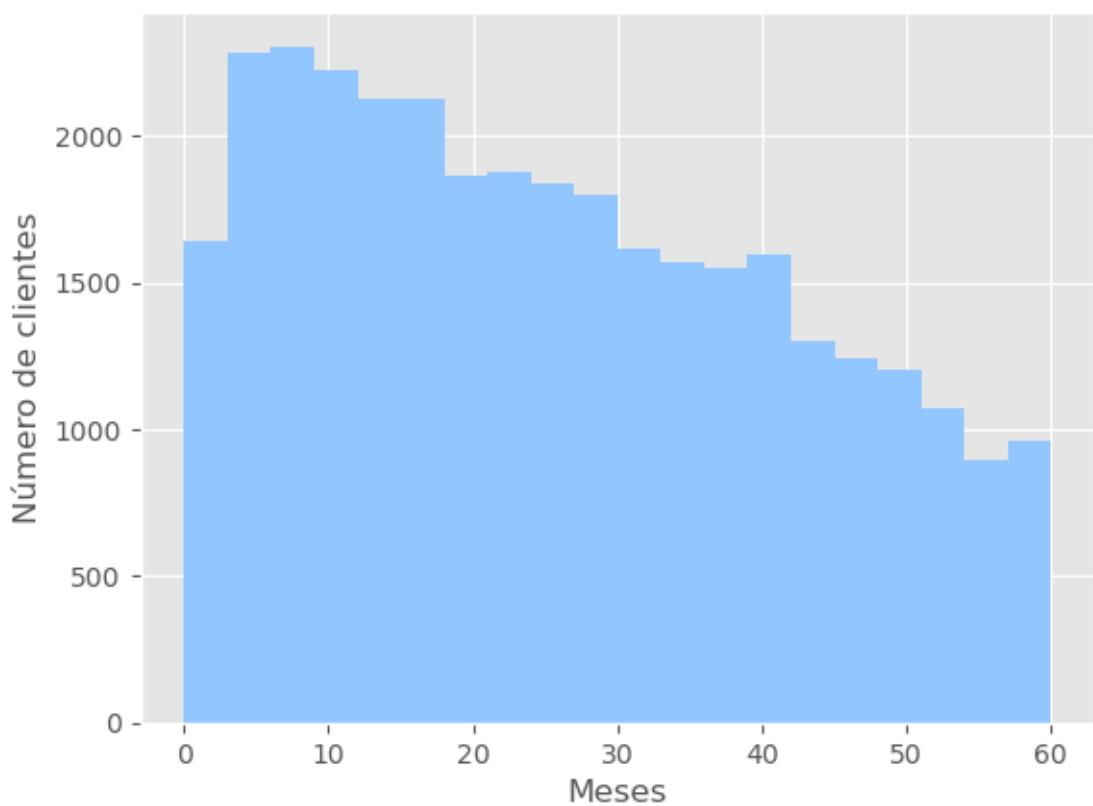
Profissão dos clientes confiáveis



A quantidade de meses em que as contas estão abertas, são analisadas através do histograma e demonstram que há semelhanças entre os clientes confiáveis e o total de clientes.

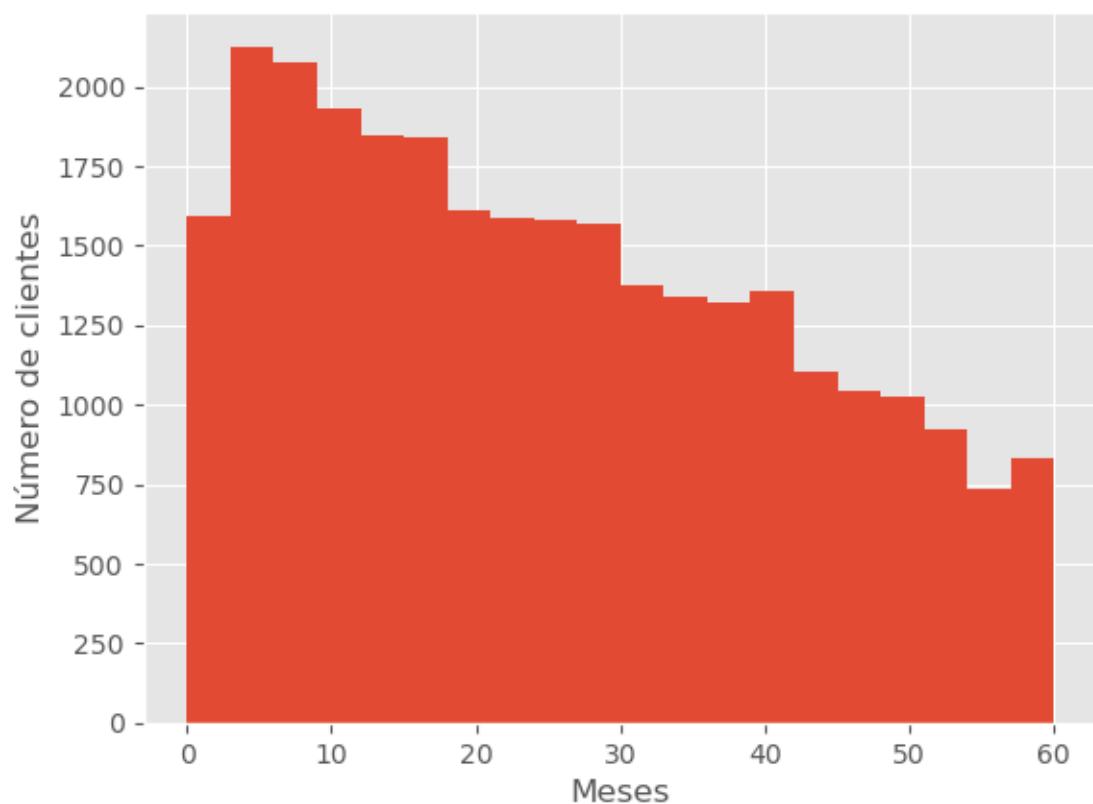
```
#Plotagem de histograma com os meses em que a contas dos clientes estão abertas
new_df.TEMPO_DA_CONTA.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Meses")
plt.ylabel("Número de clientes")
plt.title("Meses da conta dos clientes")
plt.show()
```

Meses da conta dos clientes



```
#Plotagem de histograma com os meses em que a contas dos clientes confiáveis estão abertas
clientes_confiaveis.TEMPO_DA_CONTA.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Meses")
plt.ylabel("Número de clientes")
plt.title("Meses da conta dos clientes confiáveis")
plt.show()
```

Meses da conta dos clientes confiáveis



```
new_df[ 'TEMPO_DA_CONTA' ].describe()
```

```
count      33110.000000
mean       25.429236
std        16.364444
min        0.000000
25%       11.000000
50%       24.000000
75%       38.000000
max       60.000000
Name: TEMPO_DA_CONTA, dtype: float64
```

```
clientes_confiaveis[ 'TEMPO_DA_CONTA' ].describe()
```

```
count      28819.00000
mean       24.99542
std        16.43375
min        0.00000
25%       11.00000
50%       23.00000
75%       38.00000
max       60.00000
Name: TEMPO_DA_CONTA, dtype: float64
```

Da análise de todas as informações desse dataframe, percebe-se que os dados dos clientes, confiáveis ou não, são semelhantes, o que indica um comportamento padrão nos dados do dataframe.

5. Criação de Modelos de Machine Learning

O aprendizado de máquina (Machine Learning) é o estudo de algoritmos de computador que melhoram automaticamente através da experiência e do uso de dados. Pode ser classificado em: a) aprendizado supervisionado, em que o computador é apresentado com exemplos de entradas e suas saídas desejadas, rotuladas, e o objetivo é aprender uma regra geral que mapeia entradas em saídas; b) aprendizado não supervisionado, em que nenhum rótulo é dado ao computador, deixando-o sozinho para encontrar a estrutura em sua entrada. O aprendizado não supervisionado pode ser um objetivo em si (descobrir padrões ocultos nos dados) ou um meio para um fim (aprendizado de recursos); e c) aprendizado por reforço, em que o computador interage com um ambiente dinâmico no qual deve realizar uma determinada tarefa. À medida que navega no espaço do problema, o programa recebe um feedback, que ele tenta maximizar.

O objetivo do estudo é identificar quais clientes são confiáveis dado um conjunto de características e, para isso, optou-se por utilizar algoritmos de classificação, que são cálculos preditivos usados para atribuir dados a categorias predefinidas, analisando um conjunto de dados de treinamento.

Nesse estudo serão utilizados 6 tipos de algoritmos de classificação, são eles: Árvore de Decisão, Regressão Logística, Naïve Bayes, Gradiente Descendente, KNN (K - Nearest Neighbors) e Random Forest.

Árvores de Decisão é uma tabela de decisão sob a forma de árvore com nós e folhas sequenciais e interligados que classificam as instâncias, ordenando-as com base nos valores dos recursos. Cada nó em uma árvore de decisão representa uma característica em uma instância a ser classificada, e cada ramo representa um valor que o nó pode assumir. As instâncias são classificadas começando no nó raiz com base em seus valores de recursos. Trata-se de um dos modelos mais práticos e mais utilizados em inferência por indução.

A regressão logística está entre as técnicas mais utilizadas quando se deseja entender as correlações entre variáveis de entrada e saída de um modelo, normalmente são as primeiras a serem testadas em modelos de machine learning. É uma técnica recomendada para situações em que a variável dependente é de natureza dicotômica ou binária. Quanto às independentes, tanto podem ser categóricas ou não. Ela é mais útil no entendimento da influência de diversas variáveis independentes em uma saída única variável. A desvantagem é que funciona apenas quando a variável prevista é binária, assume que todos os preditores são independentes uns dos outros e assume que os dados estão livres de valores ausentes.

O algoritmo Naive Bayes é uma técnica de classificação com uma suposição de independência entre as variáveis preditoras. O classificador Naive (do inglês ingênuo) Bayes assume que a presença de uma característica particular em uma classe não está relacionada com a presença de qualquer outra. As principais vantagens deste algoritmo é que ele requer uma pequena quantidade de dados de treinamento para estimar os parâmetros necessários e é extremamente rápido em comparação com métodos mais sofisticados. A maior desvantagem é que ele é conhecido por não ser um bom avaliador.

O algoritmo gradiente descendente é um dos algoritmos de maior sucesso em problemas de Machine Learning. O método consiste em encontrar, de forma iterativa, os valores dos parâmetros que minimizam determinada função de interesse. As vantagens desse algoritmo são sua eficiência e facilidade de implementação, mas ele requer vários hiperparâmetros e é sensível ao dimensionamento de recursos.

O algoritmo k-nearest neighbor, muitas vezes abreviado k-nn, é uma abordagem para classificação de dados que estima a probabilidade de um ponto de dados ser membro de um grupo ou de outro, dependendo de qual grupo os pontos de dados mais próximos a ele estão. Esse algoritmo é de fácil implementação, robusto a variações nos dados de treinamento e efetivo quando temos muitos dados de treinamento. O ponto negativo desse algoritmo é que exige muito recurso computacional.

Por fim, o algoritmo Random Forest que é um algoritmo de aprendizagem supervisionada que cria uma “floresta” de um modo aleatório. A “floresta” é uma combinação (ensemble) de árvores de decisão, na maioria dos casos treinados com o método de bagging. A ideia principal do método de bagging é que a combinação dos modelos de aprendizado o que aumenta o resultado geral. Dizendo de outro modo, o algoritmo cria várias árvores de decisão e as combina para obter uma predição com maior acurácia e mais estável. As dificuldades desse modelo são sua dificuldade de implementação e complexidade do algoritmo.

Para o trabalho com os algoritmos mencionados, os dados categóricos foram transformados para valores numéricos. Os valores das colunas COD_SEXO, COD_TEM_CARRO, COD_TEM_IMOVEL e COD_TEM_EMPREGO serão substituídos por 0 ou 1.

```
new_df["COD_SEXO"] = new_df['COD_SEXO'].replace(['H', 'M'], [0,1])
new_df["COD_TEM_CARRO"] = new_df["COD_TEM_CARRO"].replace(["S", "N"], [1,0])
new_df["COD_TEM_IMOVEL"] = new_df["COD_TEM_IMOVEL"].replace(["S", "N"], [1,0])
new_df["COD_TEM_EMPREGO"] = new_df["COD_TEM_EMPREGO"].replace(["S", "N"], [1,0])
```

Para as demais colunas que não permitem uma classificação binária, a escolha foi utilizar a função “get_dummies” que cria novas colunas no dataframe que terão como nome os valores de cada coluna com informações categóricas e preenche cada linha com “0” e “1” dependendo do valor que a entrada possuía.

As colunas NIVEL_ESCOLARIDADE, TIPO_MORADIA, ESTADO_CIVIL, TIPO_RENDA e PROFISSAO serão summarizadas em categorias de acordo com seus valores e transformadas para numéricas.

```
print(new_df['NIVEL_ESCOLARIDADE'].unique())
['Superior' 'Secundário' 'Superior Incompleto' 'Médio' 'Pós Graduado']

nível_escolaridade = {'Secundário':'SECUNDARIO',
                     'Médio':'SECUNDARIO',
                     'Superior':'SUPERIOR',
                     'Superior Incompleto':'SUPERIOR_INCOMPLETO',
                     'Pós Graduado':'SUPERIOR'}

new_df["NIVEL_ESCOLARIDADE"] = new_df["NIVEL_ESCOLARIDADE"].map(nível_escolaridade)
new_df= pd.get_dummies(new_df, columns=['NIVEL_ESCOLARIDADE'])
```

```
new_df["NIVEL_ESCOLARIDADE_SECUNDARIO"] = new_df['NIVEL_ESCOLARIDADE_SECUNDARIO'].replace({True: 1, False: 0})
new_df["NIVEL_ESCOLARIDADE_SUPERIOR"] = new_df['NIVEL_ESCOLARIDADE_SUPERIOR'].replace({True: 1, False: 0})
new_df["NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO"] = new_df['NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO'].replace({True: 1, False: 0})
```

```
new_df.info()
```

```
Data columns (total 22 columns):
 #   Column           Non-Null Count Dtype
 ---  -----
 0   ID               33110 non-null   int64
 1   COD_SEXO         33110 non-null   int64
 2   COD_TEM_CARRO   33110 non-null   int64
 3   COD_TEM_IMOVEL  33110 non-null   int64
 4   NUM_FILHOS       33110 non-null   float64
 5   RENDA            33110 non-null   float64
 6   TIPO_RENDA       33110 non-null   object
 7   ESTADO_CIVIL     33110 non-null   object
 8   TIPO_MORADIA    33110 non-null   object
 9   COD_TEL_TRABALHO 33110 non-null   int64
 10  COD_TEM_TELEFONE 33110 non-null   int64
 11  COD_TEM_EMAIL   33110 non-null   int64
 12  PROFISSAO        33110 non-null   object
 13  NUM_MEMBROS_FAMILY 33110 non-null   float64
 14  RESULTADO        33110 non-null   int32
 15  TEMPO_DA_CONTA  33110 non-null   int64
 16  IDADE            33110 non-null   float64
 17  COD_TEM_EMPREGO 33110 non-null   int64
 18  ANOS_EMPREGADO  33110 non-null   float64
 19  NIVEL_ESCOLARIDADE_SECUNDARIO 33110 non-null   int64
 20  NIVEL_ESCOLARIDADE_SUPERIOR    33110 non-null   int64
 21  NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO 33110 non-null   int64
dtypes: float64(5), int32(1), int64(12), object(4)
memory usage: 5.4+ MB
```

```

print(new_df['TIPO_MORADIA'].unique())

['Apartamento alugado' 'Casa / Apartamento' 'Apartamento do Município'
 'Mora com os pais' 'Apartamento Cooperativo' 'Apartamento Comercial']

tipo_moradia = {'Casa / Apartamento' : 'CASA_APARTAMENTO',
                 'Mora com os pais': 'COM_OS_PAIS',
                 'Apartamento Comercial' : 'CASA_APARTAMENTO',
                 'Apartamento alugado': 'CASA_APARTAMENTO',
                 'Apartamento do Município': 'CASA_APARTAMENTO',
                 'Apartamento Cooperativo': 'CASA_APARTAMENTO'}

new_df["TIPO_MORADIA"] = new_df['TIPO_MORADIA'].map(tipo_moradia)
new_df= pd.get_dummies(new_df, columns=['TIPO_MORADIA'])

: new_df["TIPO_MORADIA_CASA_APARTAMENTO"] = new_df['TIPO_MORADIA_CASA_APARTAMENTO'].replace({True: 1, False: 0})
new_df["TIPO_MORADIA_COM_OS_PAIS"] = new_df['TIPO_MORADIA_COM_OS_PAIS'].replace({True: 1, False: 0})

new_df.info()

Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   ID               33110 non-null   int64  
 1   COD_SEXO         33110 non-null   int64  
 2   COD_TEM_CARRO   33110 non-null   int64  
 3   COD_TEM_IMOVEL  33110 non-null   int64  
 4   NUM_FILHOS       33110 non-null   float64 
 5   RENDA            33110 non-null   float64 
 6   TIPO_RENDA       33110 non-null   object  
 7   ESTADO_CIVIL     33110 non-null   object  
 8   COD_TEL_TRABALHO 33110 non-null   int64  
 9   COD_TEM_TELEFONE 33110 non-null   int64  
 10  COD_TEM_EMAIL    33110 non-null   int64  
 11  PROFISSAO        33110 non-null   object  
 12  NUM_MEMBROS_FAMILY 33110 non-null   float64 
 13  RESULTADO        33110 non-null   int32  
 14  TEMPO_DA_CONTA   33110 non-null   int64  
 15  IDADE            33110 non-null   float64 
 16  COD_TEM_EMPREGO 33110 non-null   int64  
 17  ANOS_EMPREGADO   33110 non-null   float64 
 18  NIVEL_ESCOLARIDADE_SECUNDARIO 33110 non-null   int64  
 19  NIVEL_ESCOLARIDADE_SUPERIOR    33110 non-null   int64  
 20  NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO 33110 non-null   int64  
 21  TIPO_MORADIA_CASA_APARTAMENTO 33110 non-null   bool  
 22  TIPO_MORADIA_COM_OS_PAIS      33110 non-null   bool 

```

```

print(new_df['ESTADO_CIVIL'].unique())

['Casado no civil' 'Casado' 'Solteiro' 'Separado' 'Viúvo']

estado_civil = {'Solteiro':'SOLTEIRO',
                 'Separado':'SOLTEIRO',
                 'Viúvo':'SOLTEIRO',
                 'Casado no civil':'CASADO',
                 'Casado':'CASADO'}

new_df["ESTADO_CIVIL"] = new_df["ESTADO_CIVIL"].map(estado_civil)
new_df = pd.get_dummies(new_df, columns=['ESTADO_CIVIL'])

```

```

new_df["ESTADO_CIVIL_CASADO"] = new_df['ESTADO_CIVIL_CASADO'].replace({True: 1, False: 0})
new_df["ESTADO_CIVIL_SOLTEIRO"] = new_df['ESTADO_CIVIL_SOLTEIRO'].replace({True: 1, False: 0})

```

`new_df.info()`

#	Column	Non-Null Count	Dtype
0	ID	33110	non-null
1	COD_SEXO	33110	non-null
2	COD_TEM_CARRO	33110	non-null
3	COD_TEM_IMOVEL	33110	non-null
4	NUM_FILHOS	33110	non-null
5	RENDAS	33110	non-null
6	TIPO_RENDAS	33110	non-null
7	COD_TEL_TRABALHO	33110	non-null
8	COD_TEM_TELEFONE	33110	non-null
9	COD_TEM_EMAIL	33110	non-null
10	PROFISSAO	33110	non-null
11	NUM_MEMBROS_FAMILIA	33110	non-null
12	RESULTADO	33110	non-null
13	TEMPO_DA_CONTA	33110	non-null
14	IDADE	33110	non-null
15	COD_TEM_EMPREGO	33110	non-null
16	ANOS_EMPREGADO	33110	non-null
17	NIVEL_ESCOLARIDADE_SECUNDARIO	33110	non-null
18	NIVEL_ESCOLARIDADE_SUPERIOR	33110	non-null
19	NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO	33110	non-null
20	TIPO_MORADIA_CASA_APARTAMENTO	33110	non-null
21	TIPO_MORADIA_COM_OS_PAIS	33110	non-null
22	ESTADO_CIVIL_CASADO	33110	non-null
23	ESTADO_CIVIL_SOLTEIRO	33110	non-null

```

print(new_df['TIPO_RENDA'].unique())

['Trabalhador da Área Privada' 'Sócio Comercial' 'Pensionista'
 'Servidor Público' 'Estudante']

tipo_renda = {'Sócio Comercial':'TRABALHANDO',
              'Servidor Público':'TRABALHANDO',
              'Trabalhador da Área Privada':'TRABALHANDO',
              'Pensionista':'PENSIONISTA',
              'Estudante':'ESTUDANTE'}

new_df["TIPO_RENDA"] = new_df["TIPO_RENDA"].map(tipo_renda)
new_df= pd.get_dummies(new_df, columns=['TIPO_RENDA'])

new_df["TIPO_RENDA_ESTUDANTE"] = new_df['TIPO_RENDA_ESTUDANTE'].replace({True: 1, False: 0})
new_df["TIPO_RENDA_PENSIONISTA"] = new_df['TIPO_RENDA_PENSIONISTA'].replace({True: 1, False: 0})
new_df["TIPO_RENDA_TRABALHANDO"] = new_df['TIPO_RENDA_TRABALHANDO'].replace({True: 1, False: 0})

```

`new_df.info()`

0	ID	33110	non-null	int64
1	COD_SEXO	33110	non-null	int64
2	COD_TEM_CARRO	33110	non-null	int64
3	COD_TEM_IMOVEL	33110	non-null	int64
4	NUM_FILHOS	33110	non-null	float64
5	RENDAS	33110	non-null	float64
6	COD_TEL_TRABALHO	33110	non-null	int64
7	COD_TEM_TELEFONE	33110	non-null	int64
8	COD_TEM_EMAIL	33110	non-null	int64
9	PROFISSAO	33110	non-null	object
10	NUM_MEMBROS_FAMILIA	33110	non-null	float64
11	RESULTADO	33110	non-null	int32
12	TEMPO_DA_CONTA	33110	non-null	int64
13	IDADE	33110	non-null	float64
14	COD_TEM_EMPREGO	33110	non-null	int64
15	ANOS_EMPREGADO	33110	non-null	float64
16	NIVEL_ESCOLARIDADE_SECUNDARIO	33110	non-null	int64
17	NIVEL_ESCOLARIDADE_SUPERIOR	33110	non-null	int64
18	NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO	33110	non-null	int64
19	TIPO_MORADIA_CASA_APARTAMENTO	33110	non-null	int64
20	TIPO_MORADIA_COM_OS_PAIS	33110	non-null	int64
21	ESTADO_CIVIL_CASADO	33110	non-null	int64
22	ESTADO_CIVIL_SOLTEIRO	33110	non-null	int64
23	TIPO_RENDA_ESTUDANTE	33110	non-null	int64
24	TIPO_RENDA_PENSIONISTA	33110	non-null	int64
25	TIPO_RENDA_TRABALHANDO	33110	non-null	int64

```

print(new_df['PROFISSAO'].unique())

['Outro' 'Equipe de segurança' 'Equipe de vendas' 'Contadores' 'Operários'
 'Gerentes' 'Motoristas' 'Equipe principal'
 'Equipe técnica altamente qualificada' 'Pessoal de limpeza'
 'Pessoal de serviço privado' 'Pessoal de cozinha'
 'Trabalhadores pouco qualificados' 'Equipe médica' 'Secretários'
 'Equipe de garçons barman' 'Equipe de RH' 'Corretor imobiliário'
 'Equipe de TI']

profissao = {'Outro': 'Outro', 'Equipe de segurança': 'Equipe_de_segurança', 'Equipe de vendas': 'Equipe_de_vendas',
 'Contadores': 'Contadores', 'Operários': 'Operários', 'Gerentes': 'Gerentes', 'Motoristas': 'Motoristas',
 'Equipe principal': 'Equipe_principal', 'Equipe técnica altamente qualificada': 'Equipe_técnica_altamente_qualifica',
 'Pessoal de limpeza': 'Pessoal_de_limpeza', 'Pessoal de serviço privado': 'Pessoal_de_serviço_privado',
 'Pessoal de cozinha': 'Pessoal_de_cozinha', 'Trabalhadores pouco qualificados': 'Trabalhadores_pouco_qualificados',
 'Equipe médica': 'Equipe_médica', 'Secretários': 'Secretários', 'Equipe de garçons barman': 'Equipe_de_garçons_bar',
 'Equipe de RH': 'Equipe_de_RH', 'Corretor imobiliário': 'Corretor_imobiliário', 'Equipe de TI': 'Equipe_de_TI' }

new_df["PROFISSAO"] = new_df["PROFISSAO"].map(profissao)
new_df = pd.get_dummies(new_df, columns=['PROFISSAO'])

```

```

new_df["PROFISSAO_Contadores"] = new_df['PROFISSAO_Contadores'].replace({True: 1, False: 0})
new_df["PROFISSAO_Corretor_imobiliário"] = new_df['PROFISSAO_Corretor_imobiliário'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_RH"] = new_df['PROFISSAO_Equipe_de_RH'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_TI"] = new_df['PROFISSAO_Equipe_de_TI'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_garçons_barman"] = new_df['PROFISSAO_Equipe_de_garçons_barman'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_segurança"] = new_df['PROFISSAO_Equipe_de_segurança'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_vendas"] = new_df['PROFISSAO_Equipe_de_vendas'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_médica"] = new_df['PROFISSAO_Equipe_médica'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_principal"] = new_df['PROFISSAO_Equipe_principal'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_técnica_altamente_qualificada"] = new_df['PROFISSAO_Equipe_técnica_altamente_qualificada'].replace({True: 1, False: 0})
new_df["PROFISSAO_Gerentes"] = new_df['PROFISSAO_Gerentes'].replace({True: 1, False: 0})
new_df["PROFISSAO_Motoristas"] = new_df['PROFISSAO_Motoristas'].replace({True: 1, False: 0})
new_df["PROFISSAO_Operários"] = new_df['PROFISSAO_Operários'].replace({True: 1, False: 0})
new_df["PROFISSAO_Outro"] = new_df['PROFISSAO_Outro'].replace({True: 1, False: 0})
new_df["PROFISSAO_Pessoal_de_cozinha"] = new_df['PROFISSAO_Pessoal_de_cozinha'].replace({True: 1, False: 0})
new_df["PROFISSAO_Pessoal_de_limpeza"] = new_df['PROFISSAO_Pessoal_de_limpeza'].replace({True: 1, False: 0})
new_df["PROFISSAO_Pessoal_de_serviço_privado"] = new_df['PROFISSAO_Pessoal_de_serviço_privado'].replace({True: 1, False: 0})
new_df["PROFISSAO_Secretários"] = new_df['PROFISSAO_Secretários'].replace({True: 1, False: 0})
new_df["PROFISSAO_Trabalhadores_pouco_qualificados"] = new_df['PROFISSAO_Trabalhadores_pouco_qualificados'].replace({True: 1, Fa

```

```

new_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33110 entries, 0 to 33109
Data columns (total 44 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0    ID              33110 non-null  int64   
 1    COD_SEXO        33110 non-null  int64   
 2    COD_TEM_CARRO  33110 non-null  int64   
 3    COD_TEM_IMOVEL 33110 non-null  int64   
 4    NUM_FILHOS      33110 non-null  float64 
 5    RENDA           33110 non-null  float64 
 6    COD_TEL_TRABALHO 33110 non-null  int64   
 7    COD_TEM_TELEFONE 33110 non-null  int64   
 8    COD_TEM_EMAIL   33110 non-null  int64   
 9    NUM_MEMBROS_FAMILY 33110 non-null  float64 
 10   RESULTADO       33110 non-null  int32  
 11   TEMPO_DA_CONTA 33110 non-null  int64   
 12   IDADE           33110 non-null  float64 
 13   COD_TEM_EMPREGO 33110 non-null  int64   
 14   ANOS_EMPREGADO  33110 non-null  float64 
 15   NIVEL_ESCOLARIDADE_SECUNDARIO 33110 non-null  int64   
 16   NIVEL_ESCOLARIDADE_SUPERIOR   33110 non-null  int64   
 17   NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO 33110 non-null  int64   
 18   TIPO_MORADIA_CASA_APARTAMENTO 33110 non-null  int64   
 19   TIPO_MORADIA_COM_OS_PAIS    33110 non-null  int64   
 20   ESTADO_CIVIL_CASADO       33110 non-null  int64

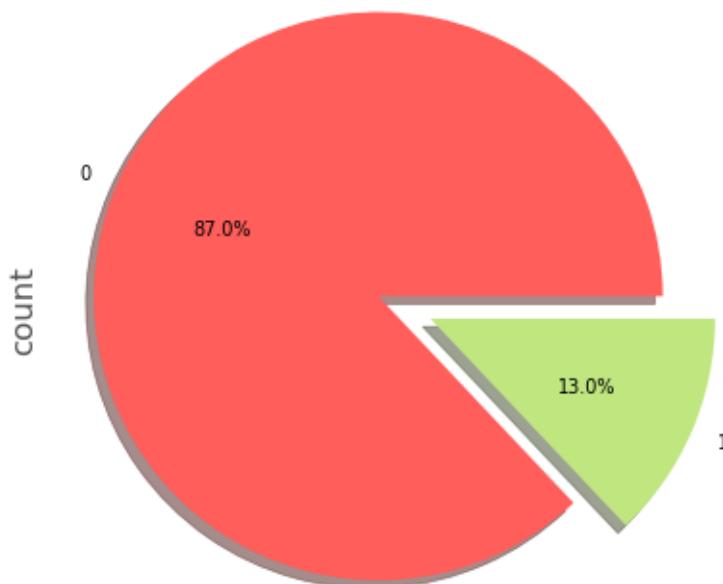
```

21	ESTADO_CIVIL_SOLTEIRO	33110	non-null	int64
22	TIPO_RENDA_ESTUDANTE	33110	non-null	int64
23	TIPO_RENDA_PENSIONISTA	33110	non-null	int64
24	TIPO_RENDA_TRABALHANDO	33110	non-null	int64
25	PROFISSAO_Contadores	33110	non-null	int64
26	PROFISSAO_Corretor_imobiliário	33110	non-null	int64
27	PROFISSAO_Equipe_de_RH	33110	non-null	int64
28	PROFISSAO_Equipe_de_TI	33110	non-null	int64
29	PROFISSAO_Equipe_de_garçons_barmen	33110	non-null	int64
30	PROFISSAO_Equipe_de_segurança	33110	non-null	int64
31	PROFISSAO_Equipe_de_vendas	33110	non-null	int64
32	PROFISSAO_Equipe_médica	33110	non-null	int64
33	PROFISSAO_Equipe_principal	33110	non-null	int64
34	PROFISSAO_Equipe_técnica_altamente_qualificada	33110	non-null	int64
35	PROFISSAO_Gerentes	33110	non-null	int64
36	PROFISSAO_Motoristas	33110	non-null	int64
37	PROFISSAO_Operários	33110	non-null	int64
38	PROFISSAO_Outro	33110	non-null	int64
39	PROFISSAO_Pessoal_de_cozinha	33110	non-null	int64
40	PROFISSAO_Pessoal_de_limpeza	33110	non-null	int64
41	PROFISSAO_Pessoal_de_serviço_privado	33110	non-null	int64
42	PROFISSAO_Secretários	33110	non-null	int64
43	PROFISSAO_Trabalhadores_pouco_qualificados	33110	non-null	int64

É possível perceber que todas as colunas agora são numéricas.

A análise do trabalho avalia a previsibilidade de um cliente ser confiável ou não com base em alguns parâmetros. Dessa forma, a informação que utilizaremos como resultado é a coluna RESULTADO, que traz se o cliente é confiável. Porém, há muito mais clientes confiáveis no dataframe conforme o gráfico, 88.2 % de clientes confiáveis.

Distribuição dos clientes confiáveis



```
#Contagem dos clientes confiáveis e não confiáveis
new_df['RESULTADO'].value_counts()
```

```
RESULTADO
0    28819
1    4291
Name: count, dtype: int64
```

Essa proporção prejudica a construção do algoritmo porque uma simples afirmação de que o cliente será confiável acertaria em 87% das vezes, nesse caso concreto. Para contornar esse problema, será utilizada a biblioteca Scikit-learn, mais especificamente sua função resample, que, após alguns passos, igualará a quantidade de amostras de clientes confiáveis ou não.

```
#Importação da função resample da biblioteca sklearn
from sklearn.utils import resample
```

Logo após, o dataframe será dividido de acordo o valor da coluna RESULTADO, que possui valor 0 para clientes confiáveis e 1 para não confiáveis.

```
#Criação de um dataframe apenas com os clientes não confiáveis
new_df_nao_confiaveis = new_df[new_df.RESULTADO==1]
new_df_nao_confiaveis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4291 entries, 0 to 33109
Data columns (total 44 columns):
```

```
#Criação de um dataframe apenas com os clientes confiáveis
new_df_confiaveis = new_df[new_df.RESULTADO==0]
new_df_confiaveis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28819 entries, 2 to 32757
Data columns (total 44 columns):
```

Em seguida, esses novos dataframes terão as mesmas quantidades de registros conforme o comando:

```
#Ajuste no número de entradas do dataframe de clientes não confiáveis
new_df_nao_confiaveis_upsampled = resample(new_df_nao_confiaveis,
                                             replace=True, n_samples=28819,
                                             random_state=123)

new_df_nao_confiaveis_upsampled.info()

<class 'pandas.core.frame.DataFrame'>
Index: 28819 entries, 29877 to 9328
Data columns (total 44 columns):
```

Utilizando a função pd.concat, os dois dataframes com os mesmos números de registros serão concatenados para uma única fonte de dados.

```
#Concatenação dos dataframes dos clientes confiáveis e não confiáveis
new_df_final_upsampled = pd.concat([new_df_confiaveis,
                                     new_df_nao_confiaveis_upsampled])

new_df_final_upsampled.info()

<class 'pandas.core.frame.DataFrame'>
Index: 57638 entries, 2 to 9328
Data columns (total 44 columns):
```

```
#Contagem dos clientes confiáveis e não confiáveis
new_df_final_upsampled['RESULTADO'].value_counts()
```

```
RESULTADO
0    28819
1    28819
Name: count, dtype: int64
```

O novo dataframe possui 57638 linhas, dividido igualmente entre clientes confiáveis e não confiáveis.

Para a aplicação dos algoritmos de classificação, o dataframe será dividido em dois, onde um dataframe consta apenas os atributos que serão analisados e outro que mostra se o cliente é confiável. O dataframe que contém todos os atributos será chamado de X_train e o que contém o resultado será chamado de y_train.

```
#Divisão para as bases de treinamento  
X_train = new_df_final_upsampled.drop(['RESULTADO'], axis = 1)  
y_train = new_df_final_upsampled.RESULTADO
```

O próximo passo para a aplicação dos algoritmos de classificação é dividir o dataframe em bases de treinamento e de teste. Para esse procedimento será utilizado a função “train_test_split”, também da biblioteca Scikit-learn. Para a divisão de percentual de dados para treinamento e para teste, será utilizado o padrão da função, que é de 75% para treinamento e 25% para teste.

```
#Importação da função train_test_split  
from sklearn.model_selection import train_test_split
```

```
#Criação das bases de teste e treinamento  
xtreinamento, xteste, ytreinamento, yteste = train_test_split(X_train, y_train, random_state = 0)
```

```
#Informação do dataframe de treinamento  
xtreinamento.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 43228 entries, 32829 to 3090  
Data columns (total 43 columns):
```

```
#Informação do dataframe de teste  
xteste.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 14410 entries, 18884 to 28317  
Data columns (total 43 columns):
```

```
#Contagem dos resultados para treinamento  
ytreinamento.count()
```

43228

```
#Contagem dos resultados para teste  
yteste.count()
```

14410

Utilizando as funções “accuracy_score” e “classification_report” da biblioteca Scikit-learn, será definida qual medida de avaliação será analisada para verificar a eficiência do modelo. Essas funções mostram as medidas de acurácia, precisão, revocação (recall) e f1-score.

A acurácia é a quantidade de acertos do modelo dividido pelo total da amostra e indica uma performance geral do modelo:
A acurácia é a quantidade de acertos do modelo dividido pelo total da amostra e indica uma performance geral do modelo:

$$Acurácia = \frac{Verdadeiros Positivos + Verdadeiros Negativos}{Total de Amostras}$$

A precisão define os chamados positivos verdadeiros, ou seja, dentre os exemplos classificados como verdadeiros, quantos eram realmente verdadeiros.

$$Precisão = \frac{Verdadeiros Positivos}{Verdadeiros Positivos + Falsos Positivos}$$

A revocação (recall) indica qual a porcentagem de dados classificados como verdadeiros comparado com a quantidade real de resultados verdadeiros que existem na amostra.

$$Revocação = \frac{Verdadeiros Positivos}{Verdadeiros Positivos + Falsos Negativos}$$

A F1-score traz a média ponderada de precisão e revocação e traz um número único que determina a qualidade geral do modelo.

$$F1 - score = \frac{2 * Precisão * Revocação}{Precisão + Revocação}$$

A medida de acurácia, será a principal medida utilizada nesta análise, porém todas outras medidas serão consideradas na avaliação. Essas medidas serão utilizadas através da importação das funções “accuracy_score” e “classification_report”.

```
#Importação das funções para as medidas de avaliação dos algoritmos
from sklearn.metrics import accuracy_score, classification_report
```

O próximo passo é a utilização dos algoritmos listados anteriormente, através dos dataframes de treinamento e de teste. Escolhidas as medidas de avaliação, o processo para todos dos algoritmos será o mesmo, começando pela importação do respectivo algoritmo, realizando o treinamento por meio da função “fit” nas duas bases de treinamento e registrando sua acurácia por meio da função score. Em seguida, será utilizada a função “predict” na base teste xteste e sua saída será comparada com a série yteste, tendo suas medidas de avaliação sendo geradas por meio das funções “accuracy_score” e “classification_report”.

```
#Criação do modelo utilizando a Árvore de decisão
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
clientes_tree = DecisionTreeClassifier(random_state=0)
clientes_tree = clientes_tree.fit(xtreinamento, ytreinamento)
print("Acurácia: ", clientes_tree.score(xtreinamento, ytreinamento))
Train_predict = clientes_tree.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, Train_predict))
print(classification_report(yteste, Train_predict))
```

Acurácia: 1.0

Acurácia de previsão: 0.9398334489937543

	precision	recall	f1-score	support
0	1.00	0.88	0.94	7168
1	0.90	1.00	0.94	7242
accuracy			0.94	14410
macro avg	0.95	0.94	0.94	14410
weighted avg	0.95	0.94	0.94	14410

```
#Criação do modelo utilizando a Regressão Logística
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr = lr.fit(xtreinamento, ytreinamento)
print("Acurácia: ", lr.score(xtreinamento, ytreinamento))
tp_lr = lr.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_lr))
print(classification_report(yteste, tp_lr))
```

Acurácia: 0.514041824743222

Acurácia de previsão: 0.5159611380985427

	precision	recall	f1-score	support
0	0.51	0.57	0.54	7168
1	0.52	0.46	0.49	7242
accuracy			0.52	14410
macro avg	0.52	0.52	0.51	14410
weighted avg	0.52	0.52	0.51	14410

```
#Criação do modelo utilizando Naïve Bayes
```

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb = nb.fit(xtreinamento, ytreinamento)
print("Acurácia: ", nb.score(xtreinamento, ytreinamento))
tp_nb = nb.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_nb))
print(classification_report(yteste, tp_nb))
```

Acurácia: 0.5459887110206347

Acurácia de previsão: 0.556557945870923

	precision	recall	f1-score	support
0	0.56	0.53	0.55	7168
1	0.56	0.58	0.57	7242
accuracy			0.56	14410
macro avg	0.56	0.56	0.56	14410
weighted avg	0.56	0.56	0.56	14410

```
#Criação do modelo utilizando Gradiente Descendente
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd = sgd.fit(xtreinamento, ytreinamento)
print("Acurácia: ", sgd.score(xtreinamento, ytreinamento))
tp_sgd = sgd.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_sgd))
print(classification_report(yteste, tp_sgd))
```

Acurácia: 0.49914407328583327
Acurácia de previsão: 0.5025676613462873

	precision	recall	f1-score	support
0	0.00	0.00	0.00	7168
1	0.50	1.00	0.67	7242
accuracy			0.50	14410
macro avg	0.25	0.50	0.33	14410
weighted avg	0.25	0.50	0.34	14410

```
#Criação do modelo utilizando KNN (K - Nearest Neighbors)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn = knn.fit(xtreinamento, ytreinamento)
print("Acurácia: ", knn.score(xtreinamento, ytreinamento))
tp_knn = knn.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_knn))
print(classification_report(yteste, tp_knn))
```

Acurácia: 0.906680855001388
Acurácia de previsão: 0.8634281748785566

	precision	recall	f1-score	support
0	0.95	0.76	0.85	7168
1	0.80	0.96	0.88	7242
accuracy			0.86	14410
macro avg	0.88	0.86	0.86	14410
weighted avg	0.88	0.86	0.86	14410

```

#Criação do modelo utilizando Random Forest
from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier()
rfm = rfm.fit(xtreinamento, ytreinamento)
print("Acurácia: ", rfm.score(xtreinamento, ytreinamento))
tp_rfm = rfm.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_rfm))
print(classification_report(yteste, tp_rfm))

```

```

Acurácia: 1.0
Acurácia de previsão: 0.9601665510062457
      precision    recall   f1-score   support
0         1.00     0.92     0.96     7168
1         0.93     1.00     0.96     7242

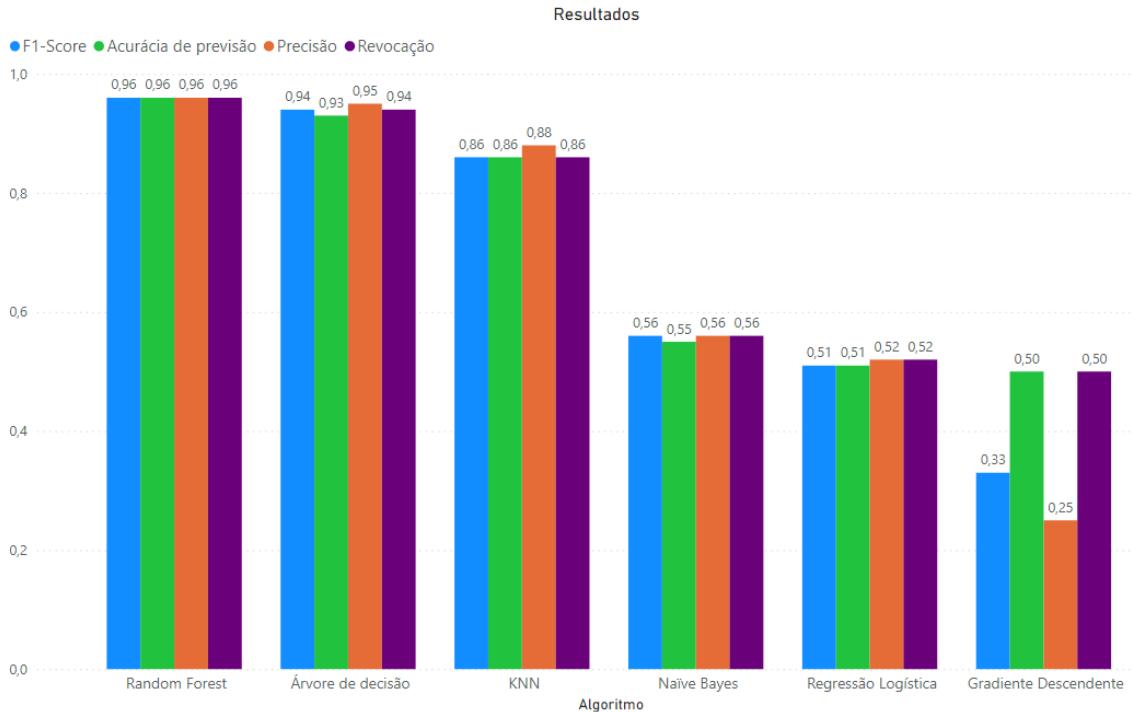
accuracy                           0.96    14410
macro avg                           0.96    14410
weighted avg                          0.96    14410

```

6. Apresentação dos Resultados

A tabela mostra os resultados médios dos algoritmos utilizados na análise:

Algoritmo	F1-Score	Acurácia de previsão	Precisão	Revocação
Random Forest	0,96	0,96	0,96	0,96
Árvore de decisão	0,94	0,93	0,95	0,94
KNN	0,86	0,86	0,88	0,86
Naïve Bayes	0,56	0,55	0,56	0,56
Regressão Logística	0,51	0,51	0,52	0,52
Gradiente Descendente	0,33	0,50	0,25	0,50



A partir dos resultados apresentados é possível perceber que, comparativamente, os modelos Random Forest e Árvore de decisão tiveram um melhor desempenho, com uma vantagem para o algoritmo Random Forest.

Ao analisar o modelo Random Forest, a precisão obtida para os clientes confiáveis apresenta um ótimo resultado com o valor 1.00, ou seja, algoritmo não identificou nenhum falso positivo no momento de identificar clientes confiáveis. O valor de 0.93 para os clientes não confiáveis também apresenta um bom resultado, demonstrando que para cada 100 previsões de um cliente não ser confiável, em apenas 7 previsões se teria um falso positivo. Porém, para os resultados de revocação o valor 1.00 é para um cliente não ser confiável, não identificando falso negativo. Para os clientes confiáveis, a cada 100 previsões, 8 seriam de falso negativo, indicando um valor de 0.92. Analisando a acurácia de previsão, temos o valor de 0,96, ou seja, de todas as amostras, o algoritmo acerta em 96% das vezes, afirmado que um cliente será confiável ou não.

O algoritmo Árvore de decisão demonstra uma precisão de 1.00 para os clientes confiáveis, sem falso positivo, e uma precisão de 0.90 para clientes não confiáveis, onde a cada 100 previsões, 10 seriam de falso negativo. Um maior valor da revocação para clientes não confiáveis, é semelhante ao que ocorreu no modelo Random Forest. A cada 100 previsões de um cliente ser confiável, em 12 previsões se teria um falso positivo e nenhum falso positivo para um cliente não ser confiável. O valor de 0.93 para acurácia de previsão, demonstra que há um acerto de 93% em afirmar se um cliente é confiável ou não.

Por fim, concluiu-se que os modelos Random Forest, Árvore de decisão e KNN apresentaram melhor capacidade de classificação, obtendo o melhor resultado nas métricas. Cabe agora testar os modelos em outros conjuntos de

dados reais para verificar se o modelo está corretamente ajustado sendo capaz de atender aos objetivos do negócio.

7. Links

Link para o vídeo :

<https://youtu.be/Y7iVP1tuo0M>

Link para o repositório Github :

<https://github.com/henrique-m-oliveira/TCC---Big-Data>

Link para o repositório Google Drive :

<https://drive.google.com/drive/folders/1VBsUZESYTF7MO9Kwi21cAWHSMzALmbIM?usp=sharing>

APÊNDICE

Programação

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

import numpy as np
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.style as style
from matplotlib import colors
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap
from IPython.display import Image
import plotly.express as px
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import plotly.offline as pyo
from plotly import tools
import seaborn as sns

import missingno as msno
```

```

from imblearn.over_sampling import SMOTE
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
confusion_matrix,precision_score,recall_score,roc_auc_score,f1_score,roc_curve

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import LabelEncoder

pyo.init_notebook_mode()

```

Processamento/Tratamento de Dados

O processamento e o tratamento dos dados foram feitos utilizando a linguagem Python, versão 3.7.0, no ambiente Jupyter Notebook, versão 5.6.0. Dentro da linguagem, utilizou-se a biblioteca “pandas” que é uma poderosa ferramenta para tratamento e análise de dados.

Em seguida, deve-se fazer a leitura e tratamento dos datasets, que, nesse caso, será feita individualmente para cada um deles

```

cliente = pd.read_csv("application_record.csv", encoding = 'utf-8') #  

é o arquivo que contém informações sobre todos os clientes quanto à  

sua situação socioeconômica  

transacao = pd.read_csv("credit_record.csv", encoding = 'utf-8')      #é  

o arquivo que contém todos os registros de pagamento/inadimplência de  

um determinado cliente

```

Para se obter informações do dataframe, pode-se utilizar a função “info()” que, no caso específico, mostrou que o dataframe possui 438557 entradas, divididas nas 18 colunas apresentadas anteriormente. falar tambem do outro dataframe

```
cliente.info()
```

```
transacao.info()
```

Interessante perceber que uma informação potencialmente útil para o estudo não foi considerada, a informação da coluna FLAG_MOBIL. Isso se deu pelo fato de que essa coluna é uma constante e por isso foi desconsiderada.

```
cliente.nunique()
transacao.nunique()
cliente.drop('FLAG_MOBIL', axis=1, inplace=True)
cliente.info()
```

Percebe-se que agora há apenas as 16 colunas escolhidas, mas ainda com as 438557 entradas filtradas anteriormente

Para verificar se há dados nulos no dataframe, utilizou-se a função “isnull()”, que faz essa análise, em conjunto com a função “sum()”, que, nesse caso, soma os valores encontrados na função anterior.

```
cliente.isna().sum()
transacao.isna().sum()
```

Após esse comando, verificou-se que o dataframe cliente, apresenta dados nulos na coluna OCCUPATION_TYPE que necessitam de tratamento. Nesse caso, os dados nulos serão substituídos pelo valor 'Other'.

```
cliente['OCCUPATION_TYPE'].fillna(value='Other', inplace=True)
cliente.isna().sum()
```

Cada linha no dataframe cliente, deve corresponder a um único ID.

```
print("Número de linhas no dataset application_record.csv:
{}".format(len(cliente)))
print("Número de clientes distintos no dataset application_record.csv
: {}".format(len(cliente.ID.unique())))
```

Para remoção dos registros duplicados foi utilizado o método “drop_duplicates”.

```
cliente=cliente.drop_duplicates(subset=cliente.columns[0],
keep='first')

print("Número de linhas no dataset application_record.csv:
{}".format(len(cliente)))
print("Número de clientes distintos no dataset application_record.csv
: {}".format(len(cliente.ID.unique())))

print("Número de linhas duplicadas no dataset credit_record.csv :
{}".format(transacao.duplicated().sum()))
```

Para modelos de machine learning supervisionados é fundamental a etapa de criação de uma coluna target no dataset. Em problemas que isso é possível, faz com que os modelos e resultados obtidos possam ser explorados com maior facilidade.

O caso em análise busca avaliar a previsibilidade se um candidato é um cliente 'bom' ou 'ruim' com base em alguns parâmetros. Dessa forma, a informação que utilizaremos como resultado é a coluna STATUS, que traz os dias em atraso no pagamento.

0: 1 a 29 dias de atraso
1: 30 a 59 dias de atraso
2: 60 a 89 dias de atraso
3: 90 a 119 dias de atraso
4: 120 a 149 dias de atraso
5: Mais de 150 dias de atraso ou dívidas inadimplentes, baixa C : Quitados no mês X: Não há empréstimos durante o mês

As linhas que possuem target=1 representam os clientes considerados “maus pagadores” QUE atrasaram os pagamentos por mais de 30 dias pelo menos uma vez em seu histórico de pagamentos. Isso incluiu clientes com códigos específicos (1, 2, 3, 4 ou 5) em seu histórico mensal.

As linhas que possuem target=0 representam os clientes com histórico consistente de pagamentos pontuais (status “C”) ou atrasos de até 29 dias (status “0”) foram categorizados como de baixo risco e considerados perfis adequados para aprovação de cartão de crédito .

As linhas que possuem target=-1 representam os clientes que não utilizaram o cartão de crédito, indicando falta de histórico de transações.

```
transacao['STATUS'].value_counts()

transacao['target']=transacao['STATUS']
transacao['target'].replace('X', -1, inplace=True)
transacao['target'].replace('C', 0, inplace=True)
transacao['target']=transacao['target'].astype(int)
transacao.loc[transacao['target']>=1,'target']=1

transacao['target'].value_counts()

# O valor de -1 deve ser eliminado da construção do modelo

transacao = transacao[transacao['target']!= -1]

transacao['target'].value_counts()
```

Integração (merge/join) dos datasets. O objetivo agora é criar um único dataset consolidado, gerado a partir da integração (join/merge) dos datasets gerados. Para isso, foi necessário agrupar o dataset transacao, por meio de seu ID de cliente, através dos comandos “groupby” e “agg(max)” que, em conjunto, selecionam o valor máximo de target para cada cliente.

```
transacao_df=pd.DataFrame(transacao.groupby(['ID'])['target'].agg(max))
).reset_index()

transacao_df.info()

transacao_df['target'].value_counts()

# Merge dataframes
new_df=pd.merge(cliente, transacao_df, how='inner', on=['ID'])

new_df.info()
```

Faz sentido que o número de meses em que a conta está aberta, seja correlacionado com o risco do cliente (uma vez que há mais oportunidades de perder pagamentos).

```

# Meses em que a conta está aberta
inicio_df=pd.DataFrame(transacao.groupby(['ID'])['MONTHS_BALANCE'].agg
(min)).reset_index()

# Renomear a coluna
inicio_df.rename(columns={'MONTHS_BALANCE':'ACCOUNT_LENGTH'},
inplace=True)

# Meses como número positivo
inicio_df['ACCOUNT_LENGTH']=-inicio_df['ACCOUNT_LENGTH']

# Merge dataframes
new_df=pd.merge(new_df, inicio_df, how='inner', on=['ID'])

new_df.info()

# Criar a variável idade
new_df['AGE_YEARS']=-new_df['DAYS_BIRTH']/365.2425
new_df.drop('DAYS_BIRTH', axis=1, inplace=True)

# Criar um indicador se está empregado ou não
new_df['EMPLOYED']='S'
new_df.loc[-new_df['DAYS_EMPLOYED']<0, 'EMPLOYED']='N'

# Criar a variável anos em que está empregado
new_df['YEARS_EMPLOYED']=-new_df['DAYS_EMPLOYED']/365.2425
new_df.loc[new_df['YEARS_EMPLOYED']<0, 'YEARS_EMPLOYED']=0
new_df.drop('DAYS_EMPLOYED', axis=1, inplace=True)

new_df.head()

```

Renamear as colunas

```

new_df = new_df.rename(columns={'CODE_GENDER': 'COD_SEXO',
'FLAG_OWN_CAR': 'COD_TEM_CARRO',
'FLAG_OWN_REALTY': 'COD_TEM_IMOVEL', 'CNT_CHILDREN': 'NUM_FILHOS',
'AMT_INCOME_TOTAL': 'RENDAS', 'NAME_INCOME_TYPE': 'TIPO_RENDAS',
'NAME_EDUCATION_TYPE': 'NIVEL_ESCOLARIDADE', 'NAME_FAMILY_STATUS': 'ESTADO_CIVIL',
'NAME_HOUSING_TYPE': 'TIPO_MORADIA', 'FLAG_WORK_PHONE': 'COD_TEL_TRABALHO',
'FLAG_PHONE': 'COD_TEM_TELEFONE', 'FLAG_EMAIL': 'COD_TEM_EMAIL', 'OCCUPATION_TYPE': 'PROFISSAO',
'CNT_FAM_MEMBERS': 'NUM_MEMBROS_FAMILIA', 'target': 'RESULTADO', 'ACCOUNT_LENGTH': 'TEMPO_DA_CONTA',
'AGE_YEARS': 'IDADE', 'EMPLOYED': 'COD_TEM_EMPREGO', 'YEARS_EMPLOYED': 'ANO_EMPREGADO'})

```

```
new_df.info()
```

```
new_df.head()
```

```
#new_df.dtypes
```

Vamos transformar os valores de algumas colunas para português

```
print(new_df['PROFISSAO'].unique())
```

```
print(new_df['TIPO_RENDA'].unique())
```

```
print(new_df['TIPO_MORADIA'].unique())
```

```
print(new_df['NIVEL_ESCOLARIDADE'].unique())
```

```
print(new_df['ESTADO_CIVIL'].unique())
```

```
new_df['COD_SEXO'] = new_df['COD_SEXO'].replace(['M'], 'H')  
new_df['COD_SEXO'] = new_df['COD_SEXO'].replace(['F'], 'M')
```

```
new_df['COD_TEM_CARRO'] = new_df['COD_TEM_CARRO'].replace(['Y'], 'S')  
new_df['COD_TEM_IMOVEL'] = new_df['COD_TEM_IMOVEL'].replace(['Y'], 'S')
```

```
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Commercial  
associate'], 'Sócio Comercial')  
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Working'],  
'Trabalhador da Área Privada')  
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Pensioner'],  
'Pensionista')  
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['State servant'],  
'Servidor Público')  
new_df['TIPO_RENDA'] = new_df['TIPO_RENDA'].replace(['Student'],  
'Estudante')
```

```
new_df['TIPO_MORADIA'] = new_df['TIPO_MORADIA'].replace(['House /  
apartment'], 'Casa / Apartamento')  
new_df['TIPO_MORADIA'] = new_df['TIPO_MORADIA'].replace(['Rented  
apartment'], 'Apartamento alugado')  
new_df['TIPO_MORADIA'] = new_df['TIPO_MORADIA'].replace(['With  
parents'], 'Mora com os pais')  
new_df['TIPO_MORADIA'] = new_df['TIPO_MORADIA'].replace(['Municipal  
apartment'], 'Apartamento do Município')  
new_df['TIPO_MORADIA'] = new_df['TIPO_MORADIA'].replace(['Office  
apartment'], 'Apartamento Comercial')  
new_df['TIPO_MORADIA'] = new_df['TIPO_MORADIA'].replace(['Co-op  
apartment'], 'Apartamento Cooperativo')
```

```
new_df['NIVEL_ESCOLARIDADE'] =  
new_df['NIVEL_ESCOLARIDADE'].replace(['Secondary / secondary  
special'], 'Secundário')  
new_df['NIVEL_ESCOLARIDADE'] =
```

```

new_df['NIVEL_ESCOLARIDADE'].replace(['Lower secondary'], 'Médio')
new_df['NIVEL_ESCOLARIDADE'] =
new_df['NIVEL_ESCOLARIDADE'].replace(['Higher education'], 'Superior')
new_df['NIVEL_ESCOLARIDADE'] =
new_df['NIVEL_ESCOLARIDADE'].replace(['Incomplete higher'], 'Superior Incompleto')
new_df['NIVEL_ESCOLARIDADE'] =
new_df['NIVEL_ESCOLARIDADE'].replace(['Academic degree'], 'Pós Graduado')

new_df['ESTADO_CIVIL'] = new_df['ESTADO_CIVIL'].replace(['Single / not married'], 'Solteiro')
new_df['ESTADO_CIVIL'] = new_df['ESTADO_CIVIL'].replace(['Separated'], 'Separado')
new_df['ESTADO_CIVIL'] = new_df['ESTADO_CIVIL'].replace(['Widow'], 'Viúvo')
new_df['ESTADO_CIVIL'] = new_df['ESTADO_CIVIL'].replace(['Civil marriage'], 'Casado no civil')
new_df['ESTADO_CIVIL'] = new_df['ESTADO_CIVIL'].replace(['Married'], 'Casado')

new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Security staff'], 'Equipe de segurança')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Sales staff'], 'Equipe de vendas')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Accountants'], 'Contadores')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Laborers'], 'Operários')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Managers'], 'Gerentes')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Drivers'], 'Motoristas')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Core staff'], 'Equipe principal')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['High skill tech staff'], 'Equipe técnica altamente qualificada')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Cleaning staff'], 'Pessoal de limpeza')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Private service staff'], 'Pessoal de serviço privado')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Cooking staff'], 'Pessoal de cozinha')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Low-skill Laborers'], 'Trabalhadores pouco qualificados')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Secretaries'], 'Secretários')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Medicine staff'], 'Equipe médica')
new_df['PROFISSAO'] = new_df['PROFISSAO'].replace(['Waiters/barmen staff'], 'Equipe de garçons barmen')

```

```

new_df[ 'PROFISSAO' ] = new_df[ 'PROFISSAO' ].replace(['HR staff'],
    'Equipe de RH')
new_df[ 'PROFISSAO' ] = new_df[ 'PROFISSAO' ].replace(['Realty agents' ],
    'Corretor imobiliário')
new_df[ 'PROFISSAO' ] = new_df[ 'PROFISSAO' ].replace(['IT staff'], 'Equipe
de TI')
new_df[ 'PROFISSAO' ] = new_df[ 'PROFISSAO' ].replace(['Other'], 'Outro')

new_df[ 'NUM_MEMBROS_FAMILIA' ]=new_df[ 'NUM_MEMBROS_FAMILIA' ].astype(int
)

new_df.info()

```

Remover outliers

```

fig = make_subplots(rows=3, cols=2, start_cell="bottom-left",
                     subplot_titles=("Renda", "Idade", "Anos de
Emprego", "Membros da família", "Número de filhos", "Tempo como
cliente"))

fig.add_trace(go.Box(x=new_df.RENDA,
name='Renda', boxmean=True), row=1, col=1)
fig.add_trace(go.Box(x=new_df.IDADE, name='Idade', boxmean=True),
row=1, col=2)
fig.add_trace(go.Box(x=new_df.ANOS_EMPREGADO, name='Anos de Emprego',
boxmean=True), row=2, col=1)
fig.add_trace(go.Box(x=new_df.NUM_MEMBROS_FAMILY, name="Membros da
família", boxmean=True), row=2, col=2)
fig.add_trace(go.Box(x=new_df.NUM_FILHOS, name="Número de filhos",
boxmean=True), row=3, col=1)
fig.add_trace(go.Box(x=new_df.TEMPO_DA_CONTA, name="Tempo como
cliente", boxmean=True), row=3, col=2)

```

```
fig.show()
```

```

# Função para descobrir os outliers
def find_outliers_IQR(df):

    q1=df.quantile(0.25)

    q3=df.quantile(0.75)

    IQR=q3-q1

    outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

    return outliers

```

```

# Função para substituir os outliers por valores médios
def impute_outliers_IQR(df):

```

```

q1=df.quantile(0.25)

q3=df.quantile(0.75)

IQR=q3-q1

upper = df[~(df>(q3+1.5*IQR))].max()

lower = df[~(df<(q1-1.5*IQR))].min()

df = np.where(df > upper,
    df.mean(),
    np.where(
        df < lower,
        df.mean(),
        df
    )
)

return df

outliers = find_outliers_IQR(new_df.NUM_MEMBROS_FAMILIA)

print("número de outliers NUM_MEMBROS_FAMILIA : " +
str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

outliers = find_outliers_IQR(new_df.ANOS_EMPREGADO)
print("número de outliers ANOS_EMPREGADO : " + str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

outliers = find_outliers_IQR(new_df.NUM_FILHOS)
print("número de outliers NUM_FILHOS : " + str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

outliers = find_outliers_IQR(new_df.RENDA)
print("número de outliers RENDA : " + str(len(outliers)))
print("max outlier : " + str(outliers.max()))
print("min outlier : " + str(outliers.min()))

new_df['NUM_MEMBROS_FAMILIA'] =
impute_outliers_IQR(new_df.NUM_MEMBROS_FAMILIA)
new_df['RENTA'] = impute_outliers_IQR(new_df.RENDA)
new_df['ANOS_EMPREGADO'] = impute_outliers_IQR(new_df.ANOS_EMPREGADO)
new_df['NUM_FILHOS'] = impute_outliers_IQR(new_df.NUM_FILHOS)

```

```

fig = make_subplots(rows=3, cols=2, start_cell="bottom-left",
                     subplot_titles=("Renda", "Idade", "Anos de
Emprego", "Membros da família", "Número de filhos", "Tempo como
cliente"))

fig.add_trace(go.Box(x=new_df.RENDA,
                     name='Renda', boxmean=True), row=1, col=1)
fig.add_trace(go.Box(x=new_df.IDADE, name='Idade', boxmean=True),
              row=1, col=2)
fig.add_trace(go.Box(x=new_df.ANOS_EMPREGADO, name='Anos de Emprego',
                     boxmean=True), row=2, col=1)
fig.add_trace(go.Box(x=new_df.NUM_MEMBROS_FAMILIA, name="Membros da
família", boxmean=True), row=2, col=2)
fig.add_trace(go.Box(x=new_df.NUM_FILHOS, name="Número de filhos",
                     boxmean=True), row=3, col=1)
fig.add_trace(go.Box(x=new_df.TEMPO_DA_CONTA, name="Tempo como
cliente", boxmean=True), row=3, col=2)

fig.show()

```

Análise e Exploração dos Dados

```

#Criação de um dataframe df_consulta apenas com clientes confiáveis
clientes_confiaveis = new_df[(new_df['RESULTADO']==0)]

clientes_confiaveis.info()

#Contagem das opções da coluna COD_SEXO
from collections import Counter
genero = Counter(new_df['COD_SEXO'])
genero

#Contagem das opções da coluna COD_SEXO dos clientes confiáveis
genero_clientes_confiaveis = Counter(clientes_confiaveis['COD_SEXO'])
genero_clientes_confiaveis

#Plotagem das informações de gênero dos clientes

```

```
fig, axes = plt.subplots(1,2)
```

```

g1=
new_df['COD_SEXO'].value_counts().plot.pie(explode=[0.1,0.1], autopct='
%1.1f%%', shadow=True, colors=["#76B5B3", "#EC9B9A"], textprops =
{'fontsize':12}, ax=axes[0])
g1.set_title("Clientes")

g2=
clientes_confiaveis['COD_SEXO'].value_counts().plot.pie(explode=[0.1,0
.1], autopct='%1.1f%%', shadow=True, colors=["#80DE99", "#00CECB"], textpro
ps = {'fontsize':12}, ax=axes[1])
g2.set_title("Clientes confiáveis")

```

```

fig.set_size_inches(14,5)

#Descrição estatística das idades dos clientes
new_df['IDADE'].describe()

#Descrição estatística das idades dos clientes confiáveis
clientes_confiaveis['IDADE'].describe()

#Plotagem de histograma com as idades dos clientes
new_df.IDADE.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Anos")
plt.ylabel("Número de clientes")
plt.title("Idade dos clientes")
plt.show()

#Plotagem de histograma com as idades dos clientes confiáveis
clientes_confiaveis.IDADE.hist(bins=20)
plt.style.use('seaborn-pastel')
plt.xlabel("Anos")
plt.ylabel("Número de clientes confiáveis")
plt.title("Idade dos clientes confiáveis")
plt.show()

#Contagem dos níveis de escolaridade dos clientes

escolaridade_clientes =
Counter(new_df['NIVEL_ESCOLARIDADE'].sort_values())
escolaridade_clientes

#Percentuais de escolaridade dos clientes

labels = []
sizes = []
n_clientes=sum(escolaridade_clientes.values())
for x, y in escolaridade_clientes.items():
    a = y/n_clientes*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a,2))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in range(len(sizes))])

_, _ = ax1.pie(sizes, startangle=90)

ax1.axis('equal')

```

```

total = sum(sizes)
plt.legend(
    loc='center left',
    labels=labels,
    prop={'size': 8},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
)
plt.title("Escolaridade dos clientes", fontsize = 12, loc = 'center')
plt.show()

#Contagem dos níveis de escolaridade dos clientes confiáveis

escolaridade_clientes_confiaveis =
Counter(clientes_confiaveis['NIVEL_ESCOLARIDADE'].sort_values())
escolaridade_clientes_confiaveis

#Percentuais de escolaridade dos clientes confiáveis

labels = []
sizes = []
n_clientes_confiaveis=sum(escolaridade_clientes_confiaveis.values())
for x, y in escolaridade_clientes_confiaveis.items():
    a = y/n_clientes_confiaveis*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a,2))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in
range(len(sizes))])

_, _ = ax1.pie(sizes, startangle=90)

ax1.axis('equal')

total = sum(sizes)
plt.legend(
    loc='center left',
    labels=labels,
    prop={'size': 8},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
)
plt.title("Escolaridade dos clientes confiáveis", fontsize = 12, loc =
'right')

```

```

plt.show()

#Contagem do estado civil dos clientes

estado_civil_clientes = Counter(new_df['ESTADO_CIVIL'].sort_values())
estado_civil_clientes

#Identificação dos percentuais do estado civil dos clientes
n_estado_civil=sum(estado_civil_clientes.values())
for x, y in estado_civil_clientes.items():
    a = y/n_estado_civil*100
    print(str(x) + ': ' + '\n' + str(round(a,2)) + '%' + '\n')

#Contagem dos níveis do estado civil dos clientes confiáveis

estado_civil_clientes_confiaveis =
Counter(clientes_confiaveis['ESTADO_CIVIL'].sort_values())
estado_civil_clientes_confiaveis

#Identificação dos percentuais do estado civil dos clientes confiáveis
n_estado_civil_clientes_confiaveis=sum(estado_civil_clientes_confiaveis.values())
for x, y in estado_civil_clientes_confiaveis.items():
    a = y/n_estado_civil_clientes_confiaveis*100
    print(str(x) + ': ' + '\n' + str(round(a,2)) + '%' + '\n')

#Percentuais do estado civil dos clientes

labels = []
sizes = []
n_clientes=sum(estado_civil_clientes.values())
for x, y in estado_civil_clientes.items():
    a = y/n_clientes*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a,2))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in range(len(sizes))])

_, _ = ax1.pie(sizes, startangle=90)
ax1.axis('equal')

total = sum(sizes)
plt.legend(
    loc='center left',
    labels=labels,

```

```

        prop={'size': 8},
        bbox_to_anchor=(0.0, 1),
        bbox_transform=fig1.transFigure
    )

plt.title("Estado civil dos clientes", fontsize = 12, loc = 'center')
plt.show()

#Percentuais do estado civil dos clientes confiáveis

labels = []
sizes = []
n_clientes=sum(estado_civil_clientes_confiaveis.values())
for x, y in estado_civil_clientes_confiaveis.items():
    a = y/n_clientes*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a,2))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in
range(len(sizes))])

_, _ = ax1.pie(sizes, startangle=90)

ax1.axis('equal')

total = sum(sizes)
plt.legend(
    loc='center left',
    labels=labels,
    prop={'size': 8},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
)

plt.title("Estado civil dos clientes confiáveis", fontsize = 12, loc =
'center')

plt.show()

#descrição estatística dos valores da renda dos clientes
new_df['RENDAM'].describe().astype('int')

#Plotagem do histograma da renda dos clientes
sns.distplot(new_df.RENDAM, kde=False)
plt.title('Distribuição da renda dos clientes')

```

```

#Descrição estatística dos valores da renda dos clientes confiáveis
clientes_confiaveis['RENDAS'].describe().astype('int')

#Plotagem do histograma da renda dos clientes confiáveis
sns.distplot(clientes_confiaveis.RENDAS, kde=False, color='red')
plt.title('Distribuição da renda dos clientes confiáveis')

#Plotagem de histograma com os anos empregado dos clientes
new_df.ANOS_EMPREGADO.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Anos")
plt.ylabel("Número de clientes")
plt.title("Anos de emprego dos clientes")
plt.show()

#Plotagem de histograma com os anos empregado dos clientes confiáveis
clientes_confiaveis.ANOS_EMPREGADO.hist(bins=20)
plt.style.use('seaborn-pastel')
plt.xlabel("Anos")
plt.ylabel("Número de clientes")
plt.title("Anos de emprego dos clientes confiáveis")
plt.show()

#Percentuais de emprego, carro e imóvel dos clientes
fig, axes = plt.subplots(1,3)

g1=
new_df['COD_TEM_EMPREGO'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%.1f%%', shadow=True, colors=["#76B5B3", "#EC9B9A"], textprops = {'fontsize':12}, ax=axes[0])
g1.set_title("Possui Emprego")

g2=
new_df['COD_TEM_CARRO'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%.1f%%', shadow=True, colors=["#80DE99", "#00CECB"], textprops = {'fontsize':12}, ax=axes[1])
g2.set_title("Possui Carro")

g3=
new_df['COD_TEM_IMOVEL'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%.1f%%', shadow=True, colors=["#76B5B3", "#00CECB"], textprops = {'fontsize':12}, ax=axes[2])
g3.set_title("Possui Imóvel")

fig.set_size_inches(14,5)

#Percentuais de emprego, carro e imóvel dos clientes confiáveis
fig, axes = plt.subplots(1,3)

g1=
clientes_confiaveis['COD_TEM_EMPREGO'].value_counts().plot.pie(explode =[0.1,0.1], autopct='%.1f%%', shadow=True,

```

```

colors=[ "#76B5B3", "#EC9B9A"],textprops = {'fontsize':12}, ax=axes[0])
g1.set_title("Possui Emprego")

g2=
clientes_confiaveis['COD_TEM_CARRO'].value_counts().plot.pie(explode=[0.1,0.1],autopct='%1.1f%%',shadow=True,colors=[ "#80DE99", "#00CECB"],textprops = {'fontsize':12}, ax=axes[1])
g2.set_title("Possui Carro")

g3=
clientes_confiaveis['COD_TEM_IMOVEL'].value_counts().plot.pie(explode=[0.1,0.1],autopct='%1.1f%%',shadow=True,colors=[ "#76B5B3", "#00CECB"],textprops = {'fontsize':12}, ax=axes[2])
g3.set_title("Possui Imóvel")

fig.set_size_inches(14,5)

#Contagem da profissão dos clientes

profissao_clientes = Counter(new_df['PROFISSAO'].sort_values())
profissao_clientes

#Percentuais da profissão dos clientes

labels = []
sizes = []
n_clientes=sum(profissao_clientes.values())
for x, y in profissao_clientes.items():
    a = y/n_clientes*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a,2))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in range(len(sizes))])

_, _ = ax1.pie(sizes, startangle=90)

ax1.axis('equal')

total = sum(sizes)
plt.legend(
    loc='center right',
    labels=labels,
    prop={'size': 8},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
)

```

```

plt.title("Profissão dos clientes", fontsize = 12, loc = 'center')

plt.show()

#Contagem da profissão dos clientes confiáveis

profissao_clientes_confiaveis =
Counter(clientes_confiaveis['PROFISSAO'].sort_values())
profissao_clientes_confiaveis

#Percentuais da profissão dos clientes

labels = []
sizes = []
n_clientes=sum(profissao_clientes_confiaveis.values())
for x, y in profissao_clientes_confiaveis.items():
    a = y/n_clientes*100
    labels.append(str(x) + ': ' + str(round(a,2)) + '%')
    sizes.append(round(a,2))

fig1, ax1 = plt.subplots(figsize=(6, 5))
fig1.subplots_adjust(0.3,0,1,1)

theme = plt.get_cmap('bwr')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes)) for i in
range(len(sizes))])

_, _ = ax1.pie(sizes, startangle=90)

ax1.axis('equal')

total = sum(sizes)
plt.legend(
    loc='center right',
    labels=labels,
    prop={'size': 8},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
)

plt.title("Profissão dos clientes confiáveis", fontsize = 12, loc = 'center')

plt.show()

#Plotagem de histograma com os meses em que a contas dos clientes
estão abertas
new_df.TEMPO_DA_CONTA.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Meses")

```

```

plt.ylabel("Número de clientes")
plt.title("Meses da conta dos clientes")
plt.show()

#Plotagem de histograma com os meses em que a contas dos clientes confiáveis estão abertas
clientes_confiaveis.TEMPO_DA_CONTA.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Meses")
plt.ylabel("Número de clientes")
plt.title("Meses da conta dos clientes confiáveis")
plt.show()

new_df[ 'TEMPO_DA_CONTA' ].describe()
clientes_confiaveis[ 'TEMPO_DA_CONTA' ].describe()

```

Criação de Modelos de Machine Learning

#Converter as varáveis categóricas para numéricas

```

new_df[ "COD_SEXO" ] = new_df[ 'COD_SEXO' ].replace(['H','M'],[0,1])
new_df[ "COD_TEM_CARRO" ] =
new_df[ "COD_TEM_CARRO" ].replace(["S","N"],[1,0])
new_df[ "COD_TEM_IMOVEL" ] =
new_df[ "COD_TEM_IMOVEL" ].replace(["S","N"],[1,0])
new_df[ "COD_TEM_EMPREGO" ] =
new_df[ "COD_TEM_EMPREGO" ].replace(["S","N"],[1,0])

```

Para que se possa avaliar os algoritmos mencionados, optou-se por realizar alguns ajustes no dataframe new_df. Como boa prática, os dados categóricos serão transformados em valores inteiros, mais especificamente as colunas NIVEL_ESCOLARIDADE, ESTADO_CIVIL, TIPO_MORADIA e PROFISSAO.

```

print(new_df[ 'NIVEL_ESCOLARIDADE' ].unique())

nivel_escolaridade = {'Secundário':'SECUNDARIO',
                      'Médio':'SECUNDARIO',
                      'Superior':'SUPERIOR',
                      'Superior Incompleto':'SUPERIOR_INCOMPLETO',
                      'Pós Graduado':'SUPERIOR'}

new_df[ "NIVEL_ESCOLARIDADE" ] =
new_df[ "NIVEL_ESCOLARIDADE" ].map(nivel_escolaridade)
new_df= pd.get_dummies(new_df, columns=[ 'NIVEL_ESCOLARIDADE' ])

new_df.info()

new_df[ "NIVEL_ESCOLARIDADE_SECUNDARIO" ] =
new_df[ 'NIVEL_ESCOLARIDADE_SECUNDARIO' ].replace({True: 1, False: 0})
new_df[ "NIVEL_ESCOLARIDADE_SUPERIOR" ] =
new_df[ 'NIVEL_ESCOLARIDADE_SUPERIOR' ].replace({True: 1, False: 0})
new_df[ "NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO" ] =

```

```

new_df[ 'NIVEL_ESCOLARIDADE_SUPERIOR_INCOMPLETO'].replace({True: 1,
False: 0})

new_df.info()

print(new_df[ 'TIPO_MORADIA'].unique())

tipo_moradia = {'Casa / Apartamento' : 'CASA_APARTAMENTO',
'Mora com os pais': 'COM_OS_PAIS',
'Apartamento Comercial' : 'CASA_APARTAMENTO',
'Apartamento alugado': 'CASA_APARTAMENTO',
'Apartamento do Município': 'CASA_APARTAMENTO',
'Apartamento Cooperativo': 'CASA_APARTAMENTO'}

new_df[ "TIPO_MORADIA"] = new_df[ 'TIPO_MORADIA'].map(tipo_moradia)
new_df= pd.get_dummies(new_df, columns=[ 'TIPO_MORADIA'])

new_df.info()

new_df[ "TIPO_MORADIA_CASA_APARTAMENTO"] =
new_df[ 'TIPO_MORADIA_CASA_APARTAMENTO'].replace({True: 1, False: 0})
new_df[ "TIPO_MORADIA_COM_OS_PAIS"] =
new_df[ 'TIPO_MORADIA_COM_OS_PAIS'].replace({True: 1, False: 0})

new_df.info()

print(new_df[ 'ESTADO_CIVIL'].unique())

estado_civil = {'Solteiro':'SOLTEIRO',
'Separado':'SOLTEIRO',
'Viúvo':'SOLTEIRO',
'Casado no civil':'CASADO',
'Casado ':'CASADO'}

new_df[ "ESTADO_CIVIL"] = new_df[ "ESTADO_CIVIL"].map(estado_civil)
new_df= pd.get_dummies(new_df, columns=[ 'ESTADO_CIVIL'])

new_df[ "ESTADO_CIVIL_CASADO"] =
new_df[ 'ESTADO_CIVIL_CASADO'].replace({True: 1, False: 0})
new_df[ "ESTADO_CIVIL_SOLTEIRO"] =
new_df[ 'ESTADO_CIVIL_SOLTEIRO'].replace({True: 1, False: 0})

new_df.info()

print(new_df[ 'TIPO_RENDAA'].unique())

tipo_renda = {'Sócio Comercial':'TRABALHANDO',
'Servidor Público':'TRABALHANDO',
'Trabalhador da Área Privada':'TRABALHANDO',
'Pensionista':'PENSIONISTA',
'Estudante ':'ESTUDANTE'}

new_df[ "TIPO_RENDAA"] = new_df[ "TIPO_RENDAA"].map(tipo_renda)
new_df= pd.get_dummies(new_df, columns=[ 'TIPO_RENDAA'])

```

```

new_df["TIPO_RENDA_ESTUDANTE"] =
new_df['TIPO_RENDA_ESTUDANTE'].replace({True: 1, False: 0})
new_df["TIPO_RENDA_PENSIONISTA"] =
new_df['TIPO_RENDA_PENSIONISTA'].replace({True: 1, False: 0})
new_df["TIPO_RENDA_TRABALHANDO"] =
new_df['TIPO_RENDA_TRABALHANDO'].replace({True: 1, False: 0})

new_df.info()

print(new_df['PROFISSAO'].unique())

profissao = {'Outro': 'Outro', 'Equipe de segurança':
'Equipe_de_segurança','Equipe de vendas' : 'Equipe_de_vendas',
'Contadores' : 'Contadores', 'Operários' : 'Operários',
'Gerentes': 'Gerentes', 'Motoristas': 'Motoristas',
'Equipe principal' : 'Equipe_principal','Equipe técnica
altamente qualificada' : 'Equipe_técnica_altamente_qualificada',
'Pessoal de limpeza' : 'Pessoal_de_limpeza', 'Pessoal de
serviço privado' : 'Pessoal_de_serviço_privado',
'Pessoal de cozinha' : 'Pessoal_de_cozinha',
'Trabalhadores pouco qualificados' :
'Trabalhadores_pouco_qualificados',
'Equipe médica': 'Equipe_médica', 'Secretários' :
'Secretários', 'Equipe de garçons barmen' :
'Equipe_de_garçons_barmen',
'Equipe de RH': 'Equipe_de_RH', 'Corretor imobiliário' :
'Corretor_imobiliário', 'Equipe de TI' : 'Equipe_de_TI' }

new_df["PROFISSAO"] = new_df["PROFISSAO"].map(profissao)
new_df= pd.get_dummies(new_df, columns=['PROFISSAO'])

new_df.info()

new_df["PROFISSAO_Contadores"] =
new_df['PROFISSAO_Contadores'].replace({True: 1, False: 0})
new_df["PROFISSAO_Corretor_imobiliário"] =
new_df['PROFISSAO_Corretor_imobiliário'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_RH"] =
new_df['PROFISSAO_Equipe_de_RH'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_TI"] =
new_df['PROFISSAO_Equipe_de_TI'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_garçons_barmen"] =
new_df['PROFISSAO_Equipe_de_garçons_barmen'].replace({True: 1, False:
0})
new_df["PROFISSAO_Equipe_de_segurança"] =
new_df['PROFISSAO_Equipe_de_segurança'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_de_vendas"] =
new_df['PROFISSAO_Equipe_de_vendas'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_médica"] =
new_df['PROFISSAO_Equipe_médica'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_principal"] =
new_df['PROFISSAO_Equipe_principal'].replace({True: 1, False: 0})
new_df["PROFISSAO_Equipe_técnica_altamente_qualificada"] =
new_df['PROFISSAO_Equipe_técnica_altamente_qualificada'].replace({True
: 1, False: 0})

```



```

new_df_final_upsampled.info()

#Contagem dos clientes confiáveis e não confiáveis
new_df_final_upsampled['RESULTADO'].value_counts()

#Divisão para as bases de treinamento
X_train = new_df_final_upsampled.drop(['RESULTADO'], axis = 1)
y_train = new_df_final_upsampled.RESULTADO

#Importação da função train_test_split
from sklearn.model_selection import train_test_split

#Criação das bases de teste e treinamento
xtreinamento, xteste, ytreinamento, yteste = train_test_split(X_train,
y_train, random_state = 0)

#Informação do dataframe de treinamento
xtreinamento.info()

#Informação do dataframe de teste
xteste.info()

#Contagem dos resultados para treinamento
ytreinamento.count()

#Contagem dos resultados para teste
yteste.count()

#Dataframe com os resultados
df_resultados = pd.DataFrame(columns=('Algoritmo','Acurácia de
previsão', 'Precisão', 'Revocação','F1-Score'))

#Importação das funções para as medidas de avaliação dos algoritmos
from sklearn.metrics import accuracy_score, classification_report

#Criação do modelo utilizando a Árvore de decisão
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
clientes_tree = DecisionTreeClassifier(random_state=0)
clientes_tree = clientes_tree.fit(xtreinamento, ytreinamento)
print("Acurácia: ", clientes_tree.score(xtreinamento, ytreinamento))
Train_predict = clientes_tree.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, Train_predict))
print(classification_report(yteste, Train_predict))

new_row = {'Algoritmo': 'Árvore de decisão', 'Acurácia de previsão':0.93, 'Precisão': 0.95,'Revocação':0.94,'F1-Score':0.94}
df_resultados.loc[len(df_resultados)] = new_row
print(df_resultados)

#Criação do modelo utilizando a Regressão Logística
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr = lr.fit(xtreinamento, ytreinamento)
print("Acurácia: ", lr.score(xtreinamento, ytreinamento))
tp_lr = lr.predict(xteste)

```

```

print("Acurácia de previsão: ", accuracy_score(yteste, tp_lr))
print(classification_report(yteste, tp_lr))

new_row = {'Algoritmo': 'Regressão Logística', 'Acurácia de previsão': 0.51, 'Precisão': 0.52, 'Revocação': 0.52, 'F1-Score': 0.51}
df_resultados.loc[len(df_resultados)] = new_row
print(df_resultados)

#Criação do modelo utilizando Naïve Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb = nb.fit(xtreinamento, ytreinamento)
print("Acurácia: ", nb.score(xtreinamento, ytreinamento))
tp_nb = nb.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_nb))
print(classification_report(yteste, tp_nb))

new_row = {'Algoritmo': 'Naïve Bayes', 'Acurácia de previsão': 0.55, 'Precisão': 0.56, 'Revocação': 0.56, 'F1-Score': 0.56}
df_resultados.loc[len(df_resultados)] = new_row
print(df_resultados)

#Criação do modelo utilizando Gradiente Descendente
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd = sgd.fit(xtreinamento, ytreinamento)
print("Acurácia: ", sgd.score(xtreinamento, ytreinamento))
tp_sgd = sgd.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_sgd))
print(classification_report(yteste, tp_sgd))

new_row = {'Algoritmo': 'Gradiente Descendente', 'Acurácia de previsão': 0.5, 'Precisão': 0.25, 'Revocação': 0.5, 'F1-Score': 0.33}
df_resultados.loc[len(df_resultados)] = new_row
print(df_resultados)

#Criação do modelo utilizando KNN (K - Nearest Neighbors)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn = knn.fit(xtreinamento, ytreinamento)
print("Acurácia: ", knn.score(xtreinamento, ytreinamento))
tp_knn = knn.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_knn))
print(classification_report(yteste, tp_knn))

new_row = {'Algoritmo': 'KNN', 'Acurácia de previsão': 0.86, 'Precisão': 0.88, 'Revocação': 0.86, 'F1-Score': 0.86}
df_resultados.loc[len(df_resultados)] = new_row
print(df_resultados)

#Criação do modelo utilizando Random Forest
from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier()
rfm = rfm.fit(xtreinamento, ytreinamento)
print("Acurácia: ", rfm.score(xtreinamento, ytreinamento))
tp_rfm = rfm.predict(xteste)

```

```
print("Acurácia de previsão: ", accuracy_score(yteste, tp_rf))
print(classification_report(yteste, tp_rf))

new_row = {'Algoritmo': 'Random Forest', 'Acurácia de previsão': 0.96,
'Precisão': 0.96, 'Revocação':0.96,'F1-Score':0.96}
df_resultados.loc[len(df_resultados)] = new_row
print(df_resultados)

df_resultados.to_excel('output.xlsx', index=False)
```