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

JOHREL MARTINS PEREIRA

MACHINE LEARNING PARA ANÁLISE DE RISCO DE CRÉDITO EM EMPRÉSTIMO BANCÁRIO

Belo Horizonte
2023

JOHREL MARTINS PEREIRA

MACHINE LEARNING PARA ANÁLISE DE RISCO DE CRÉDITO EM EMPRÉSTIMO BANCÁRIO

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

Belo Horizonte
2023

SUMÁRIO

RESUMO

No contexto atual da economia brasileira, as instituições financeiras enfrentam o desafio de avaliar o risco de crédito em um cenário de aumento da inadimplência. Para isso, as instituições financeiras utilizam modelos de avaliação de crédito para classificar seus clientes como prováveis pagadores ou inadimplentes. Historicamente, essa avaliação era realizada pelos gerentes, baseando-se em sua intuição e experiência. No entanto, com o surgimento de novas técnicas de *Data Mining* e *Machine Learning (ML)*, é possível tomar decisões mais precisas e relevantes para a concessão de crédito. Dentre as técnicas mais utilizadas estão as árvores de decisão, florestas aleatórias, máquinas de vetores de suporte e redes neurais. Essas técnicas ajudam a encontrar insights valiosos para embasar as tomadas de decisão e consequentemente diminuir o risco de perda das instituições financeiras. Diante desse cenário, este estudo tem como objetivo analisar o uso de técnicas de ML na avaliação de risco de crédito e sua aplicação no contexto da inadimplência no Brasil. O estudo realizou experimentos utilizando três modelos de ML, sendo eles o *Logistic Regression, Random Forest* e *LightGBM*. Os dados são de um dataset disponível no Kaggle, no qual avaliamos a acurácia média dos modelos de ML mencionados acima. Os resultados demonstraram que o classificador *Random Forest* superou as outras técnicas, apresentando a maior acurácia média e sendo a mais consistente nos três modelos analisados. Embora a modelo de *LightGBM* tenha obtido um bom resultado, ele não superou a mencionada anteriormente.

Palavras-chave: Instituições Financeiras; Análise de risco de crédito; Análise de Dados; Inadimplência; Machine Learning.

ABSTRACT

In the current context of the Brazilian economy, financial institutions face the challenge of evaluating credit risk in a scenario of increasing delinquency. To do so, financial institutions use credit scoring models to classify their clients as likely payers or delinquent. Historically, this evaluation was done by managers, based on their intuition and experience. However, with the emergence of new Data Mining and Machine Learning (ML) techniques, it is possible to make more accurate and relevant decisions for credit granting. Among the most used techniques are decision trees, random forests, support vector machines, and neural networks. These techniques help to find valuable insights to support decision-making and consequently reduce the risk of loss for financial institutions. In this context, this study aims to analyze the use of ML techniques in credit risk evaluation and its application in the context of delinquency in Brazil. The study conducted experiments using three ML models, namely Logistic Regression, Random Forest, and LightGBM. The data is from a dataset available on Kaggle, in which we evaluated the average accuracy of the mentioned ML models. The results demonstrated that the Random Forest classifier outperformed the other techniques, presenting the highest average accuracy and being the most consistent among the three models analyzed. Although the LightGBM model obtained a good result, it did not surpass the previously mentioned model.

Keywords: Financial Institutions; Credit Risk Analysis; Data Analysis; Delinquency; Machine Learning.

1. Introdução

1.1. Contextualização

A pandemia de COVID-19 causou impactos significativos na economia brasileira e, consequentemente, na inadimplência no país. Com a crise econômica e o aumento do desemprego, muitas pessoas e empresas tiveram dificuldades para cumprir com suas obrigações financeiras, como o pagamento de empréstimos, financiamentos e cartões de crédito.

De acordo com dados do Banco Central do Brasil (BACEN), a taxa de inadimplência do crédito para pessoas físicas e jurídicas aumentou durante a pandemia, atingindo seu pico em meados de 2020. No entanto, em 2022, a taxa de inadimplência voltou a subir, atingindo um novo recorde de famílias endividadas no país. Ainda que o governo brasileiro tenha adotado diversas medidas para mitigar os impactos da crise econômica, como o auxílio emergencial, que ajudou a manter o poder de compra dos brasileiros e a estimular a economia, os efeitos da pandemia continuam a aumentar a inadimplência do país.

As instituições financeiras, por sua vez, devem continuar a adotar medidas para avaliar e gerenciar o risco de crédito em seus empréstimos e financiamentos, garantindo uma concessão responsável e segura de crédito. E com o aumento da inadimplência, é natural que as instituições financeiras busquem novas técnicas para amenizarem suas perdas. Uma destas técnicas é a utilização do aprendizado de máquina, a fim de encontrar *insights* que possam embasar as tomadas de decisão e consequentemente diminuir seu risco de perda.

1.2. O problema proposto

O problema proposto resume-se na utilização de técnicas de *Machine Learning* (ML) para ajudar na tomada de decisão sobre empréstimos bancários. Este estudo busca avaliar um conjunto de dados que contém colunas que simulam dados de uma instituição financeira, a fim de determinar se alguém terá seu empréstimo aprovado ou não.

Com o objetivo de simplificar a compreensão do problema e da resposta proposta, adotamos a técnica dos 5Ws, que consiste em responder às seguintes indagações:

Why? Diminuir os riscos de inadimplência em empréstimos bancários.

Who? Os efeitos posteriores à pandemia tiveram repercussões diversas na economia e nas atitudes dos consumidores em geral.

What? Utilizar técnicas de ML para analisar dados bancários e buscar prever quais pessoas seriam um risco de crédito para as instituições financeiras.

Where? A aplicação da metodologia e as informações apresentadas neste estudo são realizados através da web.

When? Como o conjunto de dados é fictício, ele não possui dados históricos. Mas a ideia é que o método aplicado neste estudo possa ser replicado aos dados reais de qualquer instituição financeira.

De acordo com o Banco Central do Brasil (BACEN), atualmente o país possui cerca de 155 bancos em atividade. No entanto, esse número pode variar ao longo do tempo devido a fusões, aquisições e encerramentos de instituições financeiras.

Os estudos sobre risco de crédito em empréstimos financeiros são importantes para as instituições financeiras avaliarem a probabilidade de um tomador de empréstimo não cumprir com suas obrigações de pagamento. Esses estudos geralmente envolvem a análise do perfil de crédito do tomador, sua capacidade de pagamento, histórico de crédito, garantias oferecidas e outras variáveis relevantes.

Com base nessa análise, as instituições financeiras podem determinar o risco de crédito associado ao empréstimo e, a partir disso, definir as condições do empréstimo, como prazo, juros e garantias exigidas. Isso é importante para garantir que a instituição não assuma riscos excessivos e minimize as perdas em caso de inadimplência.

Existem diversas metodologias e modelos estatísticos utilizados para avaliar o risco de crédito em empréstimos financeiros, como análise discriminante, análise de

regressão, modelos de redes neurais, entre outros. Além disso, as instituições financeiras podem usar ferramentas de análise de dados e tecnologias avançadas, como inteligência artificial e aprendizado de máquina, para aprimorar a avaliação de risco de crédito em empréstimos financeiros, o que justifica a elaboração deste estudo.

1.3. Objetivos

O objetivo deste TCC é desenvolver um modelo de ML para análise de risco de crédito em empréstimo bancário utilizando técnicas de pré-processamento de dados, seleção de parâmetros e automatização do processo de treinamento e validação em diferentes algoritmos de classificação. O modelo será avaliado por meio da plotagem das curvas de *Precision and Recall* e *Learning Curve*, a fim de verificar o desempenho em relação à precisão, recall, viés e variância. O resultado final do trabalho será um modelo de ML capaz de identificar de forma precisa e eficiente o risco de inadimplência em empréstimos bancários, contribuindo para a redução de perdas das instituições financeiras e para a melhoria do acesso ao crédito para os clientes.

2. Coleta de Dados

Neste estudo, utilizamos uma base disponível no Kaggle, que se chama *Credit Risk Dataset.* e como mencionado anteriormente, é um conjunto de dados que contém colunas que simulam dados de uma instituição financeira.

O link e a data de acesso estarão disponíveis nas referências deste estudo. Abaixo estão descritos os campos, descrição e os tipos de formato de cada campo do *dataset*.

Campo	**Descrição**	**Tipo**
person_age	Age	int64
person_income	Annual Income	int64
person_home_ownership	Home ownership	object
person_emp_length	Employment length (in years)	float64
loan_intent	Loan intent	object
loan_grade	Loan grade	object
loan_amnt	Loan amount	int64
loan_int_rate	Interest rate	float64
loan_status	Loan status (0 is non default 1 is default)	int64
loan_percent_income	Percent income	float64
cb_person_default_on_file	Historical default	object
cb_preson_cred_hist_length	Credit history length	int64

O *dataset* contém informações sobre solicitações de empréstimos de clientes, incluindo detalhes como idade, renda, histórico de crédito, dívidas existentes, finalidade do empréstimo e outros fatores relevantes. O conjunto de dados também inclui uma coluna indicando se a solicitação de empréstimo foi aprovada ou não, o que permite o desenvolvimento de modelos para prever se uma nova solicitação de empréstimo será aprovada ou não.

3. Processamento/Tratamento de Dados

Para a análise de risco de crédito usando ML, é necessário realizar algumas etapas de processamento e tratamento de dados que serão abordadas a seguir. Uma das etapas mais importante é a realização da limpeza e organização dos dados. Este passo é indispensável para garantir a qualidade dos dados que serão utilizados na análise. Isso inclui tratar valores ausentes, corrigir inconsistências e padronizar as variáveis, se necessário. Para isto, utilizaremos os comandos a seguir, através da plataforma Google Colab.

Antes de tudo, temos que importar algumas bibliotecas que iremos utilizar em todo nosso estudo. Abaixo temos cada uma delas:

```
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, learning_curve, RandomizedSearchCV
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.metrics import ConfusionMatrixDisplay, PrecisionRecallDisplay, precision_recall_curve
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import confusion_matrix, classification_report
from lightgbm import LGBMClassifier
```

Agora vamos acessar o Google Drive, para importar a base de estudo para o ambiente onde o mesmo será analisado. Em seguida vamos ler o conteúdo e mostrar as primeiras linhas da base.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
file = "/content/drive/My Drive/Colab Notebooks/TCC/credit_risk_dataset.csv"
df = pd.read_csv(file)
df.head()
```

Após a leitura da base, verificamos se existem campos nulos ou *missing,* utilizando o comando a seguir.

```
# Buscando valores nulos ou missing
df.isnull().sum()

# Os resultados mostram que as colunas person_emp_length e loan_int_rate contém valores nulos
# Esses valores precisam ser contabilizados em nosso modelo
```

```
person_age                     0
person_income                  0
person_home_ownership          0
person_emp_length            895
loan_intent                    0
loan_grade                     0
loan_amnt                      0
loan_int_rate               3116
loan_status                    0
loan_percent_income            0
cb_person_default_on_file      0
cb_person_cred_hist_length     0
dtype: int64
```

A forma que decidimos tratar os campos nulos foi adicionar a média aritmética nos mesmos, pois acreditamos que manter as linhas utilizando a média, seria mais vantajoso para o modelo do que apenas excluí-las.

```
media_pel = df["person_emp_length"].mean()
media_lir = df["loan_int_rate"].mean()
df["person_emp_length"] = df["person_emp_length"].fillna(media_pel)
df["loan_int_rate"] = df["loan_int_rate"].fillna(media_lir)
df.isnull().sum()
```

```
person_age                  0
person_income               0
person_home_ownership       0
person_emp_length           0
loan_intent                 0
loan_grade                  0
loan_amnt                   0
loan_int_rate               0
loan_status                 0
loan_percent_income         0
cb_person_default_on_file   0
cb_person_cred_hist_length  0
dtype: int64
```

O próximo passo foi verificar a quantidade de linhas e colunas da base.

```
# Mostrar número de colunas e linhas
print("Número de linhas = ", len(df))
print("Número de colunas = ", len(df.columns))
```

```
Número de linhas =  32581
Número de colunas =  12
```

Em seguida geramos um resumo estatístico rápido para buscar alguns outliers. Podemos notar que as colunas “person_age” e “person_emp_length”, contém números muito destoantes da realidade, afinal, não é comum encontramos pessoas com 144 anos de idade, e também com um tempo de serviço prestado de 123 anos.

```
# Obtendo um resumo estatístico rápido do conjunto de dados
df.describe()

# Olhando o dataset podemos verificar alguns outliers
# person_age = 144, obviamente alguém não viveria tanto, certo?! hahah
# person_emp_length = 123, tempo de serviço de 123 anos? tem algo errado!!
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	32581.000000	32581.000000	32581.000000
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	0.218164	0.170203	5.804211
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	0.413006	0.106782	4.055001
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000	0.000000	2.000000
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	0.000000	0.090000	3.000000
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	0.000000	0.150000	4.000000
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	0.000000	0.230000	8.000000
max	144.000000	6.000000e+06	123.000000	35000.000000	23.220000	1.000000	0.830000	30.000000

Após encontramos alguns outliers usando o método *describe*, verificou-se a necessidade de se aprofundar mais e buscar mais detalhes sobre os mesmos.

```
# Detectando mais outliers usando scatter plot
fig = px.scatter_matrix(df, dimensions=
            ["person_age", "person_income", "person_emp_length",
             "loan_amnt","loan_int_rate"],
            labels={col:col.replace('_', ' ') for col in df.columns},
            height=900, color='loan_status',
            color_continuous_scale=px.colors.diverging.Temps)
fig.show()

# Analisando o scatter plot, podemos ver que a coluna "person_income" tem um outlier de $6M
```

Para isso usamos o *scatter plot,* que cria um gráfico de dispersão entre variáveis numéricas. Após analisar os gráficos abaixo, notou-se que a coluna “*person_income*” tem um valor que se destaca do restante.

Agora que verificamos a presença de alguns *outliers* em nossa base de estudo, iremos tratar os mesmos e gerar um novo resumo estatístico, com o intuito de comparar com o resumo anterior.

```
# Vamos remover os outliers para assegurar a qualidade do modelo
df = df[df['person_age'] <= 100]   # Manter apenas dados onde a idade for igual ou menor a 100 anos
df = df[df['person_emp_length'] <= 50] # Manter apenas dados onde o tempo de serviço for igual ou menor a 50 anos
df = df[df['person_income'] <= 3000000] # Manter apenas dados onde a receita for igual ou menor a $3 Milhões

# Verificar o dataset e buscar se existem mais outliers presentes
df.describe()
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
count	31679.000000	3.167900e+04	31679.000000	31679.000000	28632.000000	31679.000000	31679.000000	31679.000000
mean	27.730673	6.649010e+04	4.782064	9659.962436	11.039701	0.215442	0.169610	5.809211
std	6.213427	5.276879e+04	4.034948	6334.360554	3.229409	0.411135	0.106269	4.059710
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000	0.000000	2.000000
25%	23.000000	3.936600e+04	2.000000	5000.000000	7.900000	0.000000	0.090000	3.000000
50%	26.000000	5.600000e+04	4.000000	8000.000000	10.990000	0.000000	0.150000	4.000000
75%	30.000000	8.000000e+04	7.000000	12500.000000	13.480000	0.000000	0.230000	8.000000
max	94.000000	2.039784e+06	41.000000	35000.000000	23.220000	1.000000	0.830000	30.000000

Depois que tratamos os *outliers,* vamos buscar por dados duplicados e analisar se realmente existem duplicidades usando uma query e alguns filtros.

```
dups = df.duplicated()
```

```
df[dups]
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate
15975	23	42000	RENT	5.0	VENTURE	B	6000	9.99
15989	23	90000	MORTGAGE	7.0	EDUCATION	B	8000	10.36
15995	24	48000	MORTGAGE	4.0	MEDICAL	A	4000	5.42
16025	24	10000	RENT	8.0	PERSONAL	A	3000	7.90
16028	23	100000	MORTGAGE	7.0	EDUCATION	A	15000	7.88
...	...	...	...	...	...	...	...	...
32010	42	39996	MORTGAGE	2.0	HOMEIMPROVEMENT	A	2500	5.42
32047	36	250000	RENT	2.0	DEBTCONSOLIDATION	A	20000	7.88
32172	49	120000	MORTGAGE	12.0	MEDICAL	B	12000	10.99
32259	39	40000	OWN	4.0	VENTURE	B	1000	10.37
32279	43	11340	RENT	4.0	EDUCATION	C	1950	NaN

157 rows × 12 columns

Após fazermos a *query*, identificamos linhas duplicadas que mostraremos a seguir. Também vamos verificar a quantidade de registros antes, e depois de removê-las do *dataset*.

```
df.query("person_age==23 & person_income==42000 &\
person_home_ownership=='RENT' & loan_int_rate==9.99")
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate
6464	23	42000	RENT	5.0	VENTURE	B	6000	9.99
15975	23	42000	RENT	5.0	VENTURE	B	6000	9.99

```
# Mostrar número de colunas e linhas
print("Número de linhas = ", len(df))
print("Número de colunas = ", len(df.columns))
```

```
Número de linhas =  32574
Número de colunas =  12
```

```
# Excluindo as duplicidades
df.drop_duplicates(inplace=True)
```

```
# Mostrar número de colunas e linhas
print("Número de linhas = ", len(df))
print("Número de colunas = ", len(df.columns))
```

```
Número de linhas =  32409
Número de colunas =  12
```

Após a realização das etapas de limpeza, organização e transformação dos dados, obtivemos um *dataset* mais consistente e preparado para a análise e modelagem de ML. O próximo passo é a análise e exploração dos dados, que permitirá uma compreensão mais profunda do comportamento das variáveis e a identificação de possíveis padrões e correlações relevantes para o modelo.

4. Análise e Exploração dos Dados

A etapa de análise e exploração de dados é importante para identificar possíveis correlações entre variáveis, padrões nos dados e possíveis limitações ou vieses que devem ser considerados ao construir um modelo de ML. A análise exploratória de dados geralmente envolve a visualização e descrição das variáveis, a identificação de valores extremos e a análise de correlações e distribuições. Com base nessas análises, tentaremos levantar hipóteses sobre o comportamento dos dados e possíveis padrões.

Ao iniciarmos a exploração dos dados, vamos criar alguns gráficos para analisar as correlações entre as variáveis e suas distribuições em todo *dataset*. Mas primeiro vamos separar as variáveis numéricas das categóricas e descrevê-las usando o método *describe*.

```
#Aqui separamos as variáveis numéricas das categóricas. E abaixo as descrevemos.
df_num = df.select_dtypes(include='number')
df_cat = df.select_dtypes(include=['object','category'])
```

```
df_num.describe().T.round(2)
```

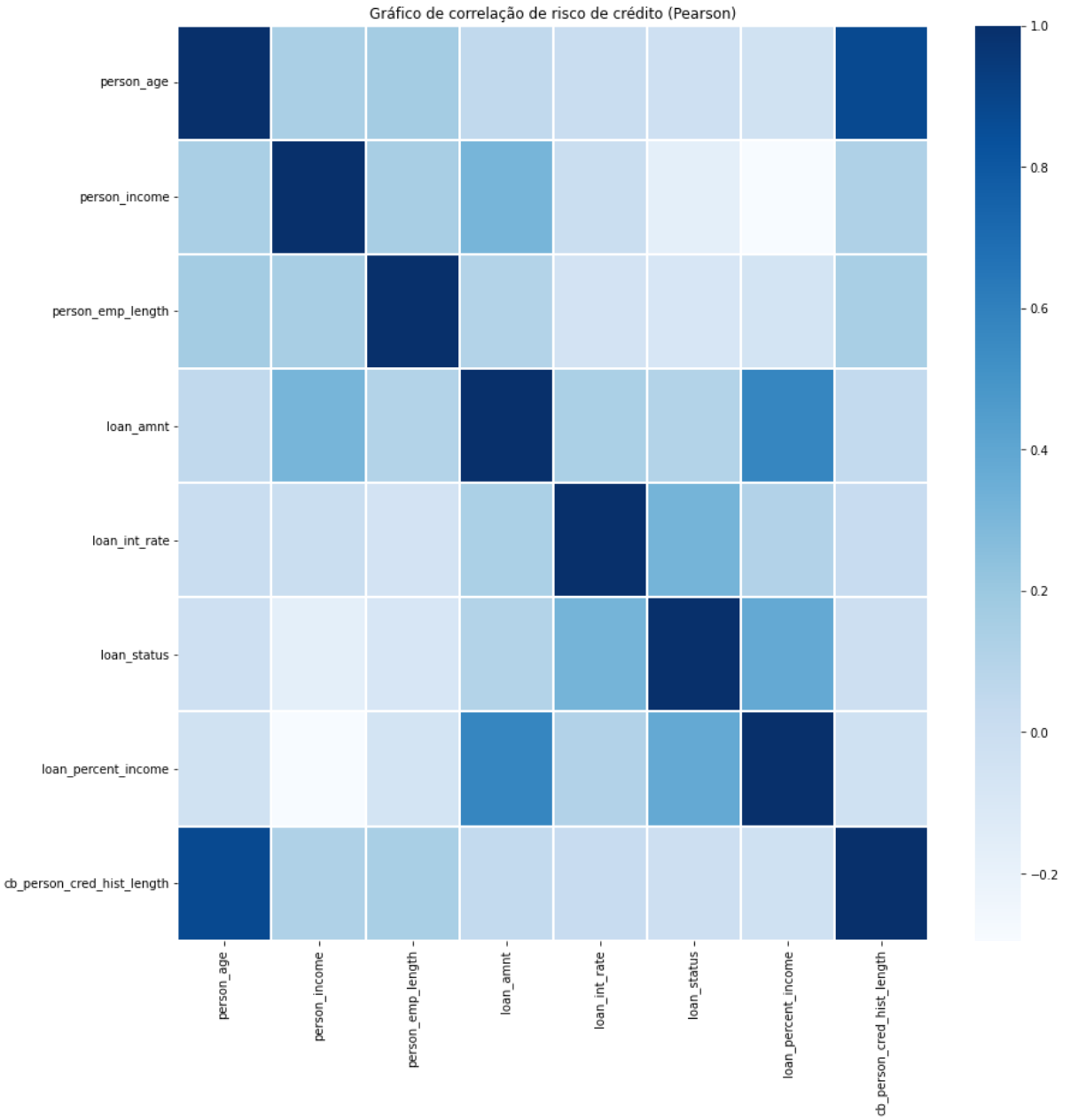
	count	mean	std	min	25%	50%	75%	max
person_age	32409.0	27.73	6.21	20.00	23.00	26.00	30.00	94.00
person_income	32409.0	65894.28	52517.87	4000.00	38500.00	55000.00	79200.00	2039784.00
person_emp_length	32409.0	4.78	3.98	0.00	2.00	4.00	7.00	41.00
loan_amnt	32409.0	9592.49	6320.89	500.00	5000.00	8000.00	12250.00	35000.00
loan_int_rate	32409.0	11.02	3.08	5.42	8.49	11.01	13.11	23.22
loan_status	32409.0	0.22	0.41	0.00	0.00	0.00	0.00	1.00
loan_percent_income	32409.0	0.17	0.11	0.00	0.09	0.15	0.23	0.83
cb_person_cred_hist_length	32409.0	5.81	4.06	2.00	3.00	4.00	8.00	30.00

```
df_cat.describe().T
```

	count	unique	top	freq
person_home_ownership	32409	4	RENT	16374
loan_intent	32409	6	EDUCATION	6409
loan_grade	32409	7	A	10702
cb_person_default_on_file	32409	2	N	26680

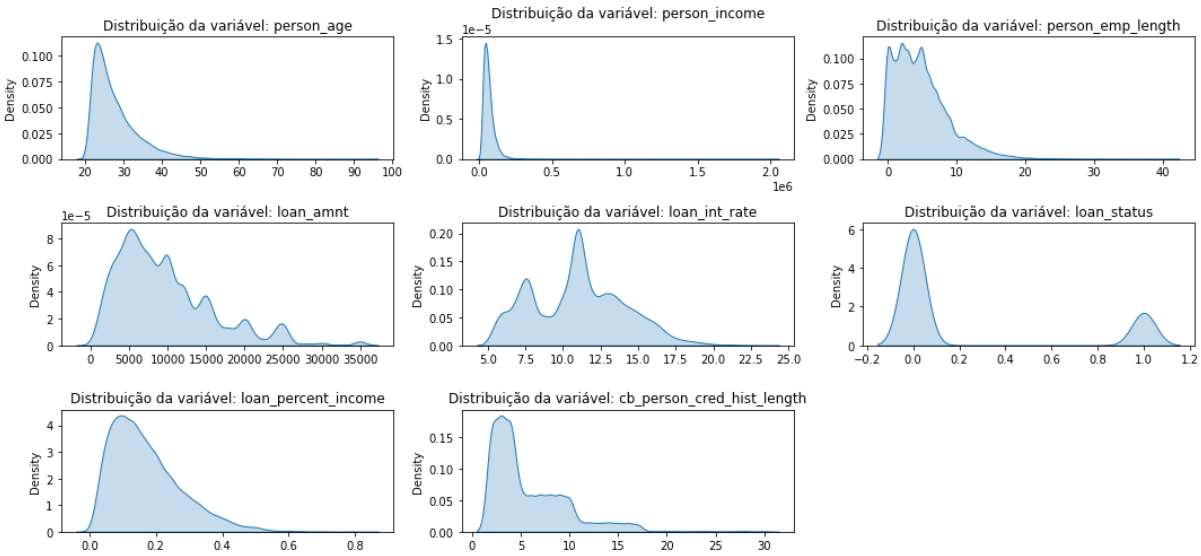
Agora que separamos as colunas númericas, das categóricas, vamos plotar alguns gráficos para melhor visualização dos dados. Abaixo temos um gráfico de correlação entre as variáveis númericas.

```
#Abaixo criamos um gráfico de calor para mostrar as correlações entre as variáveis numéricas de um conjunto de dados.
plt.figure(figsize = (14,14))
plt.title('Gráfico de correlação de risco de crédito (Pearson)')
corr = df_num.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.1,cmap="Blues")
plt.show()
```



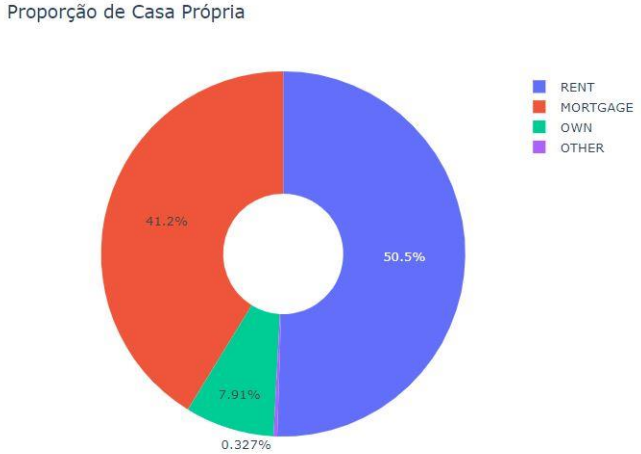
A seguir temos alguns gráficos com as distribuições de cada variável.

```
#Abaixo criamos alguns gráficos de densidade para mostrar a distribuição de cada variável numérica no conjunto de dados.
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 7), sharex = False, sharey = False)
axes = axes.ravel()
cols = df_num.columns[:]

for col, ax in zip(cols, axes):
    data = df_num
    sns.kdeplot(data=data, x=col, fill=True, ax=ax)
    ax.set(title=f'Distribuição da variável: {col}', xlabel=None)

fig.delaxes(axes[8])
fig.tight_layout()
plt.show()
```

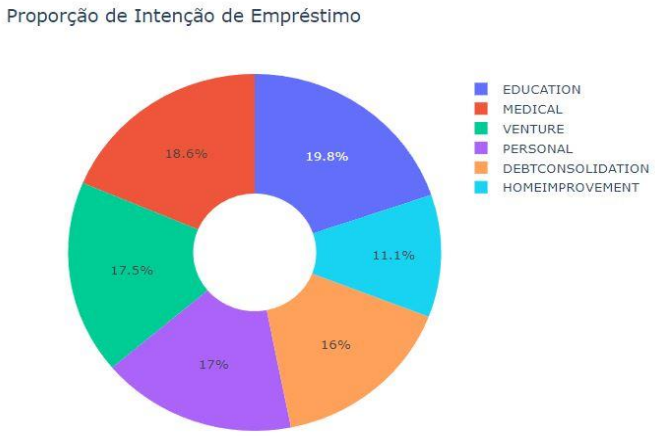


Se tratando das variáveis categóricas, criamos alguns gráficos para analisar sua distribuição percentual em todo dataset.

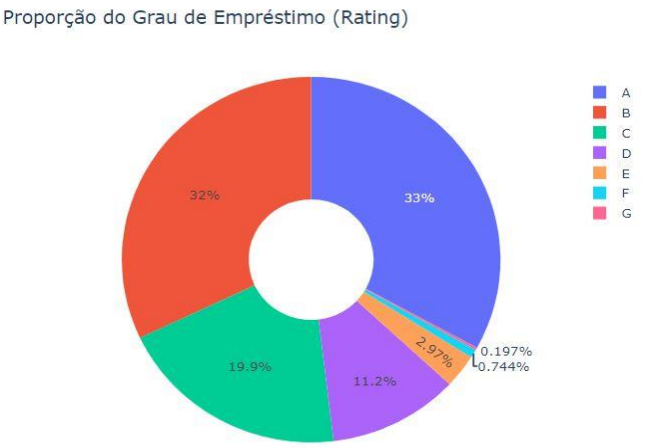
```
#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção de casa própria
ex.pie(df_cat,names='person_home_ownership',title='Proporção de Casa Própria',hole=0.33)
```



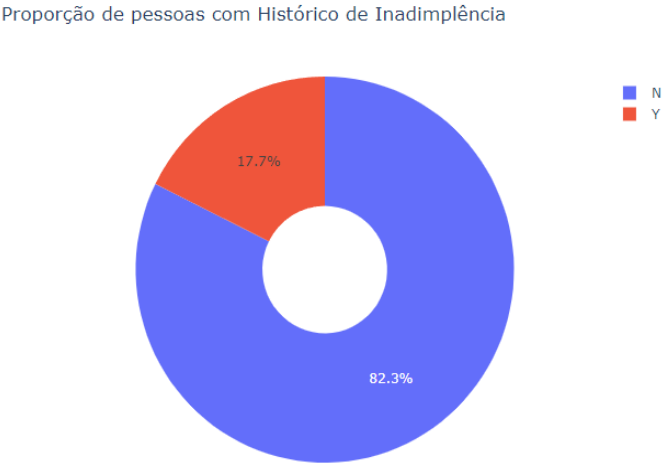
```
#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção de intenção de empréstimo
ex.pie(df_cat,names='loan_intent',title='Proporção de Intenção de Empréstimo',hole=0.33)
```




```
#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção do rating
ex.pie(df_cat,names='loan_grade',title='Proporção do Grau de Empréstimo (Rating)',hole=0.33)
```




```
#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção do histórico de inadimplência
ex.pie(df_cat,names='cb_person_default_on_file',title='Proporção de pessoas com Histórico de Inadimplência',hole=0.33)
```



No gráfico anterior, podemos reparar que 17,7% das pessoas contidas no *dataset* possuem histórico de inadimplência em suas fichas. A seguir, vamos verificar a diferença do histórico de inadimplência e o status atual de cada pessoa contida no dataset. A *feature "loan_status"* é a nossa variável *target,* é ela quem nos mostra se o empréstimo foi aprovado ou não. O código abaixo mostra a diferença percentual entre as aprovações.

```
# Vamos ver o número de empréstimos aprovados ou não, contidos no dataset
Negado = df[df.loan_status == 1].loan_status.count()
Aprovado = df[df.loan_status == 0].loan_status.count()

print("Total de pessoas no dataset: {:,}".format(len(df)))
print("Empréstimos negados = {:,}, Empréstimos aprovados = {:,}".format(Negado, Aprovado))
print("Percentual de Empréstimos negados: {:.2f}%".format((Negado / len(df)) * 100))
print("Percentual de Empréstimos aprovados: {:.2f}%".format((Aprovado / len(df)) * 100))
```

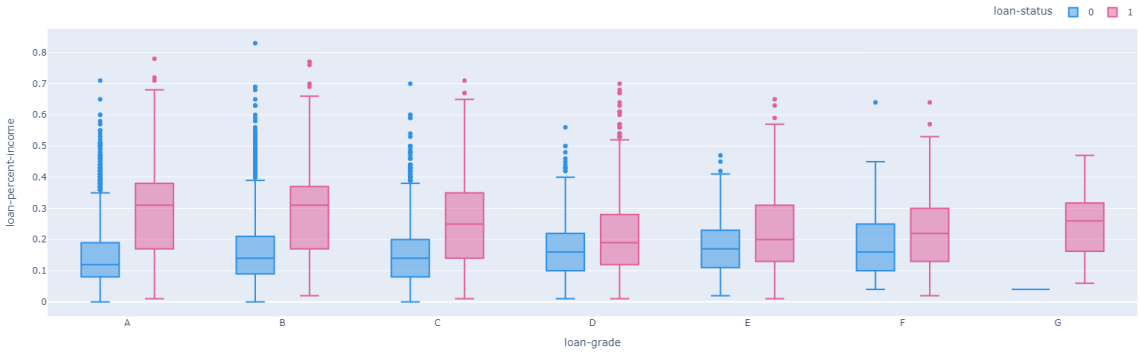
```
Total de pessoas no dataset: 32,409
Empréstimos negados = 7,088, Empréstimos aprovados = 25,321
Percentual de Empréstimos negados: 21.87%
Percentual de Empréstimos aprovados: 78.13%
```

Após analisar os dados, podemos concluir que pessoas que não tinham um histórico de inadimplência, também tiveram seu empréstimo negado. Já que o percentual de empréstimos aprovados foi apenas de 78,13%. Abaixo temos um gráfico de pizza onde podemos visualizar mais facilmente essa distribuição.

```
# Gráfico de pizza mostrando a relação de pessoas que tiveram o empréstimo aprovado ou não
values = [21.87, 78.13]
colors = ['r', 'g']
explode = [0, 0.05]
labels = ['Empréstimos negados', 'Empréstimos aprovados']
textprops = {'color': 'white', 'weight': 'bold'}
plt.pie(values, colors=colors, labels=labels, explode=explode, autopct='%1.2f%%', textprops=textprops)
plt.title('Representação de empréstimos aprovados e não aprovados')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), labels=labels, fontsize=12)
plt.show()
```

Analisando essa distribuição, verificamos a necessidade de plotar mais algumas análises onde fazemos a comparação e relação entre algumas variáveis que consideramos importante.

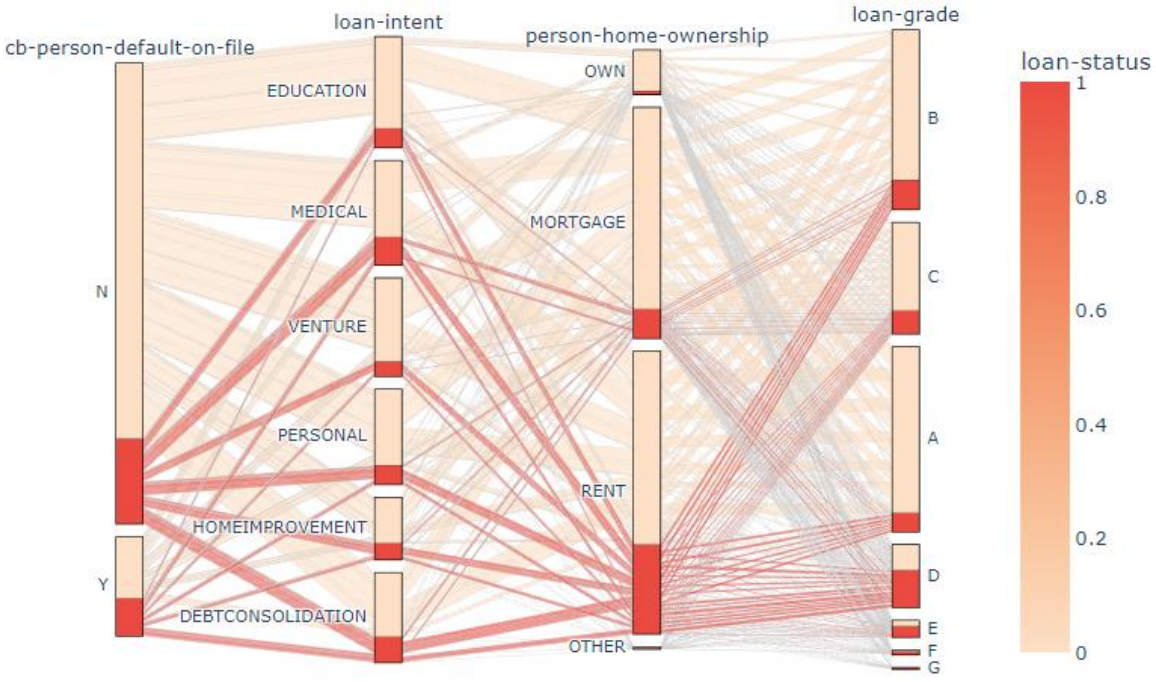
```
# A seguir, vamos ver a relação entre loan_status em relação a loan_grade e loan_percent_income
fig = px.box(df, x="loan_grade", y="loan_percent_income",
color="loan_status",
color_discrete_sequence=px.colors.qualitative.Dark24,
labels={col:col.replace('_','-') for col in df.columns},
category_orders={'loan_grade':['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']})
fig.update_layout(legend=dict(orientation="h", yanchor="bottom",
                y=1.02, xanchor="right", x=1))
fig.show()
```



As informações apresentadas no gráfico, mostram que aqueles que tiveram seu empréstimo negado, tinham uma relação entre empréstimo e renda mais alta, o que nos indica que esses indivíduos estavam se endividando além de sua capacidade de pagamento. Além disso, os mutuários na categoria G, que representam o maior risco de inadimplência, provavelmente não conseguiriam pagar o empréstimo, o que aumentaria o risco de prejuízo financeiro para a instituição.

A seguir temos um diagrama que nos ajuda a entender como as diferentes variáveis em nosso conjunto de dados estão relacionadas entre si e também com o mapa de status de empréstimo.

```
# O gráfico abaixo mostra a relação entre diferentes categorias, incluindo o status do empréstimo,
# histórico de inadimplência, finalidade do empréstimo, posse de casa e o rating.
fig = px.parallel_categories(df,
    color_continuous_scale=px.colors.sequential.Peach,
    color="loan_status",
    dimensions=['cb_person_default_on_file', 'loan_intent','person_home_ownership','loan_grade'],
    labels={col:col.replace('_', '-') for col in df.columns})
fig.show()
```



No gráfico acima, podemos notar os *insights* comentados anteriormente em nosso estudo, como por exemplo, sobre as pessoas que não tinham histórico de inadimplência e ainda sim tiveram seu empréstimo negado. Outra informação interessante é que a maioria das pessoas que não receberam o empréstimo moram de aluguel. Já a distribuição do *rating (Loan Grade)* e da intenção de empréstimo *(Loan Intent)*, são dados bem pulverizados no *dataset*.

Com base nas análises e insights obtidos durante a exploração dos dados, o próximo passo é construir modelos de ML capazes de prever se um empréstimo será aprovado ou não, com base nas informações do *dataset*. Para isso, será utilizado algoritmos de classificação, que aprenderam a distinguir as características dos clientes que tiveram seus empréstimos aprovados dos que não tiveram. O objetivo é obter um modelo com alta acurácia e precisão na predição de novos casos, que possa ser utilizado pelas instituições financeiras para avaliar a concessão de empréstimos de forma mais eficiente e segura. No próximo tópico, serão apresentados detalhes sobre os modelos de ML escolhidos, bem como os resultados obtidos a partir de sua aplicação no conjunto de dados.

5. Criação de Modelos de Machine Learning

O código em Python abaixo foi utilizado para criar um modelo de *Logistic Regression* para prever se um empréstimo será aprovado ou não, com base em variáveis disponíveis em um conjunto de dados.

Primeiro, verificamos os tipos de dados presentes no conjunto de dados usando a função dtypes do pandas. Em seguida, separamos a variável *target* (*loan_status*) do conjunto de dados e transformamos as variáveis categóricas em variáveis numéricas usando a função *get_dummies* do pandas.

Em seguida, separamos os dados em um conjunto de treinamento e um conjunto de teste usando a função *train_test_split* do *scikit-learn*, com uma proporção de 70% para o conjunto de treinamento e 30% para o conjunto de teste.

```
# Verificar os tipos de dados
print(df.dtypes)

# Separar a variável target do conjunto de dados
X = df.drop(['loan_status'], axis=1)
y = df['loan_status']

# Transformar as variáveis categóricas em variáveis numéricas
X = pd.get_dummies(X)

# Separar os dados de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Após a divisão dos dados, definimos os parâmetros a serem otimizados, usando a classe *GridSearchCV* do *scikit-learn*. Em seguida, treinamos o modelo com os dados de treinamento e exibimos os melhores parâmetros encontrados usando a função *best_params_* do objeto *GridSearchCV*.

```
# Definir os parâmetros a serem otimizados
params = {'penalty': ['l1', 'l2'],
          'C': [0.01, 0.1, 1, 10, 100],
          'solver': ['liblinear', 'saga']}

# Criar o objeto GridSearchCV
grid_search = GridSearchCV(LogisticRegression(random_state=42), params, cv=5, scoring='accuracy')

# Treinar o modelo com os dados de treinamento
grid_search.fit(X_train, y_train)

# Mostrar os melhores parâmetros encontrados
print(grid_search.best_params_)
```

Com os melhores parâmetros encontrados, criamos um novo modelo e treinamos com os dados de treinamento. Em seguida, fizemos a previsão com os

dados de teste usando o método *predict* e avaliamos a precisão do modelo usando a função *accuracy_score* do *scikit-learn*.

```
# Criar o modelo com os melhores parâmetros encontrados
model = LogisticRegression(random_state=42, penalty=grid_search.best_params_['penalty'],
                           C=grid_search.best_params_['C'], solver=grid_search.best_params_['solver'])

# Treinar o modelo com os dados de treinamento
model.fit(X_train, y_train)

# Fazer a previsão com os dados de teste
y_pred = model.predict(X_test)

# Avaliar a precisão do modelo com os dados de teste
print("Acurácia:", accuracy_score(y_test, y_pred))
```

```
Acurácia: 0.8619767561452226
```

O resultado do modelo de *Logistic Regression* foi uma acurácia de 0.86, o que significa que ele é capaz de prever corretamente se um empréstimo será aprovado ou não em 86% das vezes.

Uma outra opção para melhorar ainda mais a acurácia do modelo é utilizar outros algoritmos em vez da *Logistic Regression*. O *Random Forest* é um algoritmo de aprendizado de máquina que utiliza múltiplas árvores de decisão para realizar a classificação, o que pode ser especialmente útil para lidar com variáveis categóricas e interações entre variáveis. Além disso, o *Random Forest* é menos suscetível a *overfitting* do que a regressão logística, o que pode resultar em uma melhor performance do modelo em novos dados. Portanto, acreditamos que a acurácia atual do modelo de *Logistic Regression* ainda não é satisfatória, sendo assim, vamos testar os algoritmos *Random Forest* e *LGBMClassifier* para comparar os resultados obtidos.

Para essa comparação entre o modelo anterior, com os algoritmos *Random Forest* e *LGBMClassifier*. O primeiro passo foi fazer uma cópia do *DataFrame* utilizado anteriormente no *Logistic Regression*, para garantir que a análise e o tratamento dos dados seguiram o mesmo padrão. Em seguida, utilizamos a função *train_test_split* para separar o conjunto de dados em treinamento e teste.

```
# Cópia do Dataframe usado na regressão logística
df2 = df.copy()

# X e y serão considerados como todos os dados de treinamento
# X_test2 e y_test2 serão considerados como dados fora da amostra para avaliação do modelo

X, X_test2, y, y_test2 = train_test_split(df2.drop('loan_status', axis=1), df2['loan_status'],
                                          random_state=0,  test_size=0.2, stratify=df2['loan_status'],
                                          shuffle=True)
```

Na sequência, utilizamos algumas técnicas de pré-processamento, como a remoção da variável *'loan_percent_income'* e a normalização dos dados numéricos através da classe *StandardScaler*. Além disso, utilizamos a classe *ColumnTransformer* para aplicar o pré-processamento de forma específica em cada tipo de dado (numérico e categórico).

```
# Aqui vamos aplicar uma técnica de processamento ao remover a coluna do percentual de receita,
# para tentar melhorar a acurácia dos modelos que usaremos a seguir.
X.drop('loan_percent_income', axis=1, inplace=True)
X_test2.drop('loan_percent_income', axis=1, inplace=True)
```

```
# Abaixo imprimimos o número de valores únicos em cada coluna do dataset,
# com o objetivo do gráfico abaixo é ajudar a entender a distribuição dos valores em cada coluna.

for col in X:
    print(col, '--->', X[col].nunique())
    if X[col].nunique()<20:
        print(X[col].value_counts(normalize=True)*100)
    print()
```

```
# Aqui novamente separamos as colunas númericas com outro nome para não alterar o que fizemos acima.
num_cols = [col for col in X if X[col].dtypes != 'O']
num_cols
```

```
# Abaixo plotamos alguns histogramas utilizando os dados das colunas númericas
for col in num_cols:
    sns.histplot(X[col])
    plt.show()
```

```
# já que dropamos alguns dados de X, precisamos passar essas atualizações para y também
y = y[X.index]
```

```
# Aqui novamente separamos as colunas categóricas com outro nome para não alterar o que fizemos acima.
cat_cols = [col for col in X if X[col].dtypes == 'O']
cat_cols
```

```
# O código abaixo cria um pipeline de pré-processamento de dados numéricos,
# com o objetivo de tratar os dados antes de aplicar um modelo de aprendizado de máquina.
num_pipe = Pipeline([
    ('impute', IterativeImputer()),
    ('scale', StandardScaler()),
])
```

```
# O código abaixo usa o ColumnTransformer que aplica o pipeline que contém tanto dados numéricos como categóricos.
ct = ColumnTransformer([
    ('num_pipe', num_pipe, num_cols),
    ('cat_cols', OneHotEncoder(sparse=False, handle_unknown='ignore'), cat_cols)
], remainder='passthrough')
```

Em seguida, definimos um grid de hiperparâmetros para os algoritmos e utilizamos a função *RandomizedSearchCV* para selecionar o melhor conjunto dos mesmos. Abaixo temos os códigos utilizados e explicaremos os passos com mais detalhe.

```
# O código abaixo define um grid de hiperparâmetros para os modelos de classificação, RandomForestClassifier e LGBMClassifier.
# A grade especifica diferentes valores para vários parâmetros do modelo, como o número de estimadores,
# a taxa de aprendizado e o tipo de reforço para o LGBMClassifier.
# O objetivo é procurar pelos melhores hiperparâmetros que otimizam o desempenho do modelo em um conjunto de dados de treinamento.

grid = {
    RandomForestClassifier(random_state=0, n_jobs=-1, class_weight='balanced'):
    {'model__n_estimators':[300,400,500],
     'coltf__num_pipe__impute__estimator': [LinearRegression(), RandomForestRegressor(random_state=0),
                                    KNeighborsRegressor()]},

    LGBMClassifier(class_weight='balanced', random_state=0, n_jobs=-1):
    {'model__n_estimators':[300,400,500],
     'model__learning_rate':[0.001,0.01,0.1,1,10],
     'model__boosting_type': ['gbdt', 'goss', 'dart'],
     'coltf__num_pipe__impute__estimator':[LinearRegression(), RandomForestRegressor(random_state=0),
                                    KNeighborsRegressor()]},
}
```

```
for clf, param in grid.items():
    print(clf)
    print('-'*50)
    print(param)
    print('\n')
```

Para selecionarmos os melhores hiperparâmetros, foi criado um pipeline para cada modelo, que contém dois passos: o ‘coltf’ ct e o ‘model’ clf. Em seguida, a função *RandomizedSearchCV* é criada para cada pipeline usando os hiperparâmetros especificados no dicionário grid. Essa função realiza uma busca aleatória pelos hiperparâmetros, avaliando cada combinação de cada um deles usando validação cruzada. Os melhores são selecionados com base na pontuação de acurácia, conforme especificado pelo parâmetro *scoring*.

Após o ajuste do objeto *RandomizedSearchCV* para cada pipeline, os resultados são armazenados em um *dataframe* chamado *full_df*. Ele contém os hiperparâmetros e as pontuações médias de teste para cada combinação testada.

O melhor estimador para cada modelo também é armazenado em um dicionário chamado *best_algos*, onde a chave é o nome do modelo e o valor é o melhor estimador encontrado durante a busca.

```
# O código abaixo realiza uma busca aleatória de hiperparâmetros usando RandomizedSearchCV em um grid de modelos e parâmetros predefinidos,
# treina os modelos resultantes no conjunto de dados X e y, e armazena os resultados da validação cruzada em um DataFrame.
# Ele também armazena o melhor modelo encontrado para cada algoritmo em um dicionário chamado "best_algos".

full_df = pd.DataFrame()
best_algos = {}

for clf, param in grid.items():
    pipe = Pipeline([
    ('coltf', ct),
    ('model', clf)
])

    gs = RandomizedSearchCV(estimator=pipe, param_distributions=param, scoring='accuracy',
                            n_jobs=-1, verbose=3, n_iter=4, random_state=0)

    gs.fit(X, y)

    all_res = pd.DataFrame(gs.cv_results_)

    temp = all_res.loc[:, ['params', 'mean_test_score']]
    algo_name = str(clf).split('(')[0]
    temp['algo'] = algo_name

    full_df = pd.concat([full_df, temp], ignore_index=True)
    best_algos[algo_name] = gs.best_estimator_
```

```
# Aqui ordenamos de forma crescente os parâmetros de acordo com o seu score médio
full_df.sort_values('mean_test_score', ascending=False)
```

```
# Aqui retornamos o valor do primeiro hiperparâmetro selecionado pelo modelo que obteve
# a melhor pontuação de precisão média nos resultados do processo de validação cruzada.
full_df.sort_values('mean_test_score', ascending=False).iloc[0, 0]
```

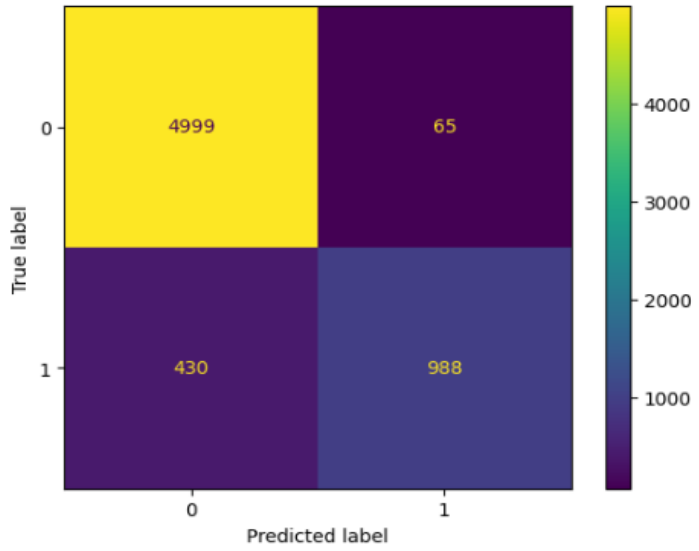
```
# Aqui selecionamos o melhor algoritmo de acordo com sua pontuação
be = best_algos['RandomForestClassifier']
be
```

```
# Aqui ajustamos os dados treinados para as variáveis X e Y
be.fit(X, y)
```

Também utilizamos a função *ConfusionMatrixDisplay* para visualizar a matriz de confusão e a função *classification_report* para obter as métricas de avaliação do modelo.

```
# Aqui criamos uma matriz de confusão para identificar os falsos negativos e positivos,
# permitindo a visualização dos acertos e erros do modelo
ConfusionMatrixDisplay.from_estimator(be, X_test2, y_test2)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f08f0628250>
```




```
# Aqui imprimimos o relatório de classificação
print(classification_report(y_test2, preds))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95      5064
           1       0.94      0.70      0.80      1418

    accuracy                           0.92      6482
   macro avg       0.93      0.84      0.88      6482
weighted avg       0.92      0.92      0.92      6482
```

```
# Aqui mostramos o score atingido pelo algoritmo
be.score(X_test2, y_test2)
```

```
0.9236346806541191
```

Com a criação dos modelos de *Logistic Regression*, *Random Forest e LGBMClassifier*, conseguimos prever a probabilidade de um cliente de um banco conseguir um empréstimo pessoal. Todos modelos tiveram desempenho satisfatório, com o *Random Forest* apresentando uma performance ligeiramente superior.

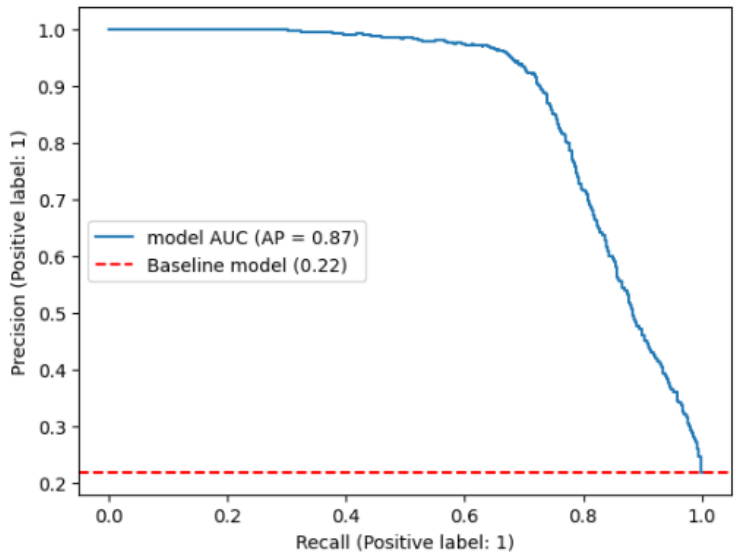
No entanto, após analisar a curva de aprendizagem, notamos a ocorrência de *overfitting*, que ocorre quando o modelo se ajusta demais aos dados de treinamento e não generaliza bem para novos dados. Por isso, no próximo capítulo, discutiremos como identificar e lidar com o *overfitting*. Uma das abordagens é ajustar os hiperparâmetros do modelo ou modificar o pré-processamento dos dados. Veremos como realizar essas tarefas e como avaliar a performance do modelo ajustado.

6. Interpretação dos Resultados

Apesar do bom desempenho dos modelos nos dados de treinamento, tivemos a ocorrência de *overfitting*, O código abaixo foi utilizado para identificar a ocorrência do mesmo e para plotar a curva de aprendizado do modelo de classificação.

```
# O gráfico abaixo mostra a curva precision-recall e a linha de base para um modelo de classificação treinado,
# representando a qualidade das previsões do modelo em diferentes níveis de threshold.
PrecisionRecallDisplay.from_estimator(estimator=be, X=X_test2, y=y_test2, name='model AUC')
baseline = y_test2.sum() / len(y_test2)
plt.axhline(baseline, ls='--', color='r', label=f'Baseline model ({round(baseline,2)})')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7f70b9f2b760>
```



Inicialmente, é criado um objeto *PrecisionRecallDisplay* que recebe o modelo de classificação be, os dados de teste X_test2 e as suas respectivas classes y_test2. Esse objeto serve para plotar a curva de *Precision and Recall* do modelo, também conhecida como curva AUC (área sob a curva). Em seguida, é calculado o valor médio da taxa de acerto esperado para um modelo ingênuo que simplesmente chutaria a classe majoritária. Esse valor é plotado como uma linha horizontal pontilhada na cor vermelha, indicando a referência para avaliar a performance do modelo.

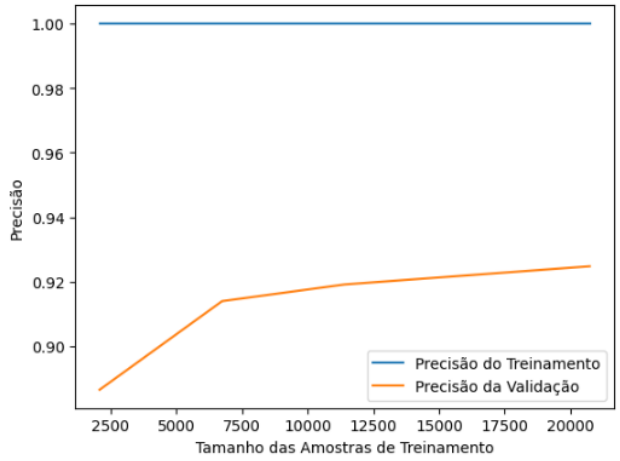
Em seguida, é calculada a curva de aprendizado do modelo. Para isso, é utilizada a função *learning_curve*, que recebe o modelo de classificação be, os dados X e as suas respectivas classes y, além de outras opções como o número de *jobs* para paralelização (*n_jobs*) e a métrica de avaliação (*scoring*). A função *learning_curve* retorna três *arrays*: “a” representa o tamanho do conjunto de

treinamento, “b” contém as pontuações de treinamento para cada tamanho de conjunto e “c” contém as pontuações de validação cruzada para cada tamanho de conjunto. Essas pontuações são medidas utilizando a métrica definida na função *learning_curve,* que é a acurácia.

```
# Logo abaixo fazemos a curva de aprendizagem a fim de visualizar o desempenho do modelo em relação ao tamanho do conjunto de treinamento.
# Os dados gerados podem ajudar identificar se o modelo está sofrendo de overfitting ou underfitting.
a, b, c = learning_curve(be, X, y, n_jobs=-1, scoring='accuracy')
```

```
# Aqui plotamos os dados da curva de aprensizagem e descubrimos a ocorrência do overfitting
plt.plot(a, b.mean(axis=1), label='Precisão do Treinamento')
plt.plot(a, c.mean(axis=1),  label='Precisão da Validação')
plt.xlabel('Tamanho das Amostras de Treinamento')
plt.ylabel('Precisão')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f08e74a9340>
```



Após criar os *arrays*, plotamos as curvas de precisão do treinamento e da validação em função do tamanho do conjunto de treinamento. A precisão média é calculada para cada tamanho de conjunto, levando em consideração a validação cruzada. Essas curvas permitem verificar se o modelo está sofrendo de *overfitting* (baixa precisão na validação) ou de *underfitting* (baixa precisão no treinamento).

O *overfitting* é um problema comum em modelos de ML, e que ocorre quando o modelo se ajusta muito bem aos dados de treinamento e, consequentemente, não consegue generalizar bem para novos dados, e que pode levar a uma alta precisão no treinamento (baixo viés), mas baixa precisão nos dados de teste ou validação (alta variância). Outro indicativo é um grande espaço entre as curvas de treinamento e validação, pois indica que o modelo está muito complexo e está aprendendo até mesmo o "ruído" nos dados, o que é indesejável.

Para corrigir o problema de *overfitting*, uma das primeiras medidas seria tentar obter mais amostras de treinamento, para que o modelo pudesse aprender melhor. Mas como não temos mais amostras, usaremos outras medidas, como por exemplo, simplificar o modelo ou reduzir sua complexidade. Isso pode ser feito reduzindo o número de características ou aumentando a regularização (lambda), além de podar as árvores de decisão.

O código abaixo mostra as correções feitas no modelo a fim de evitar *overfitting*. Nele estamos criando um novo grid de parâmetros apenas para o *Random Forest*, pois acreditamos que o modelo anterior se tornou muito complexo ao se utilizar mais de um algoritmo ao mesmo tempo. O novo grid inclui diferentes combinações de hiperparâmetros, como o número de estimadores, a profundidade máxima e o número mínimo de amostras necessárias para dividir um nó.

```
#Novo Grid usando apenas o algoritmo Randon Forest
grid = {

    RandomForestClassifier(random_state=0, n_jobs=-1, class_weight='balanced'):
    {'model__n_estimators':[100,200,300],
     'model__max_depth':[5, 9, 13],
     'model__min_samples_split':[4,6,8],
     'coltf__num_pipe__impute__estimator': [LinearRegression(), RandomForestRegressor(random_state=0),
                                            KNeighborsRegressor()]},
}
```

```
for clf, param in grid.items():
    print(clf)
    print('-'*50)
    print(param)
    print('\n')
```

Também estamos usando um pipeline para pré-processar os dados antes de ajustar o modelo, incluindo a imputação de valores ausentes usando diferentes estimadores. Em seguida, usamos o *RandomizedSearchCV* para encontrar os melhores hiperparâmetros para o modelo. Isso envolve a validação cruzada para avaliar o desempenho do modelo em dados que não foram usados para treiná-lo. Usando os melhores hiperparâmetros encontrados, ajustamos o modelo e fazemos previsões nos dados de teste.

```
# Novamente fazemos a busca aleatória de hiperparâmetros no novo grid
full_df = pd.DataFrame()
best_algos = {}

for clf, param in grid.items():
    pipe = Pipeline([
    ('coltf', ct),
    ('model', clf)
])

    gs = RandomizedSearchCV(estimator=pipe, param_distributions=param, scoring='accuracy',
                            n_jobs=-1, verbose=3, n_iter=4)

    gs.fit(X, y)

    all_res = pd.DataFrame(gs.cv_results_)

    temp = all_res.loc[:, ['params', 'mean_test_score']]
    algo_name = str(clf).split('(')[0]
    temp['algo'] = algo_name

    full_df = pd.concat([full_df, temp])
    best_algos[algo_name] = gs.best_estimator_
```

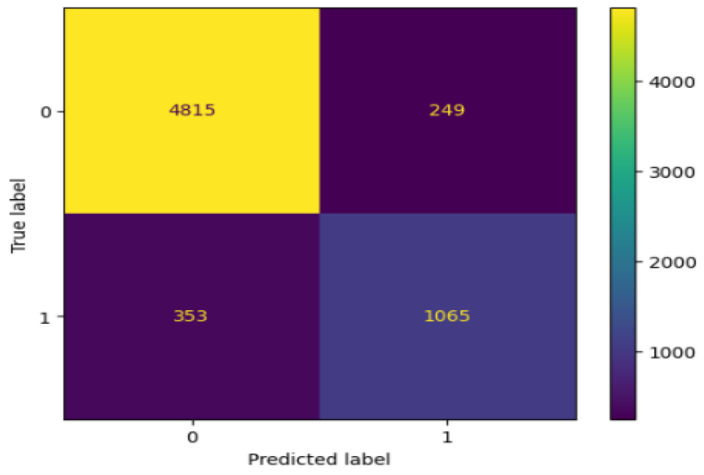
```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
# Aqui selecionamos o melhor algoritmo de acordo com sua pontuação
be = best_algos['RandomForestClassifier']
be
```

```
# Aqui ajustamos os dados treinados para as variáveis X e Y
be.fit(X, y)
```

```
# Aqui criamos uma matriz de confusão para identificar os falsos negativos e positivos,
# permitindo a visualização dos acertos e erros do modelo
ConfusionMatrixDisplay.from_estimator(be, X_test2, y_test2)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f08e7388640>
```




```
# Aqui imprimimos o relatório de classificação
print(classification_report(y_test2, preds))
```

```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      5064
           1       0.81      0.75      0.78      1418

    accuracy                           0.91      6482
   macro avg       0.87      0.85      0.86      6482
weighted avg       0.91      0.91      0.91      6482
```

```
# Aqui mostramos o score atingido pelo algoritmo
be.score(X_test2, y_test2)
```

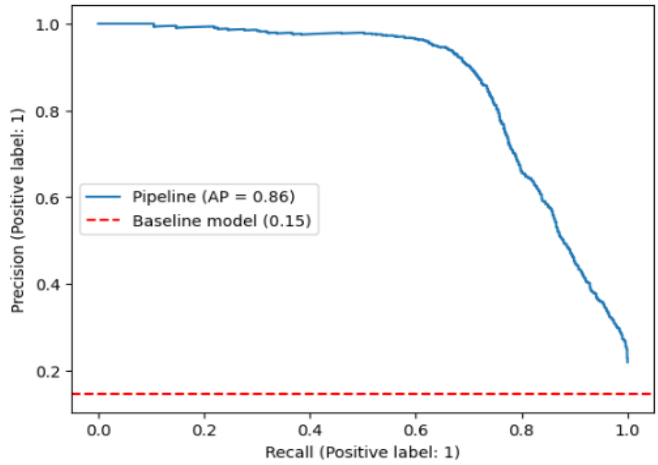
```
0.9071274298056156
```

```
# Aqui mostramos o score atingido pelo algoritmo
be.score(X_test2, y_test2)
```

```
0.9071274298056156
```

```
# O gráfico abaixo mostra a curva precision-recall e a linha de base para um modelo de classificação treinado,
# representando a qualidade das previsões do modelo em diferentes níveis de threshold.
PrecisionRecallDisplay.from_estimator(be, X_test2, y_test2)
baseline = y_test2.sum() / len(y_test)
plt.axhline(baseline, ls='--', color='r', label=f'Baseline model ({round(baseline,2)})')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7f08e7451310>
```

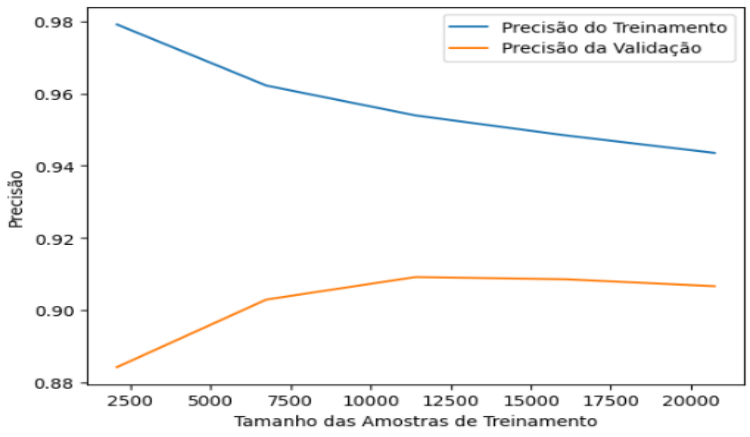


Finalmente, avaliamos o desempenho do modelo usando diferentes métricas, como a matriz de confusão, o relatório de classificação, o *Precision and Recall*. Também plotamos as curvas de aprendizado para avaliar como o modelo está se ajustando aos dados de treinamento e validação em diferentes tamanhos de amostra.

```
# Logo abaixo fazemos a nova curva de aprendizagem
a, b, c = learning_curve(be, X, y, n_jobs=-1, cv=5)
```

```
# Aqui plotamos os dados da nova curva de aprensizagem
plt.plot(a, b.mean(axis=1), label='Precisão do Treinamento')
plt.plot(a, c.mean(axis=1),  label='Precisão da Validação')
plt.xlabel('Tamanho das Amostras de Treinamento')
plt.ylabel('Precisão')
plt.legend()
```

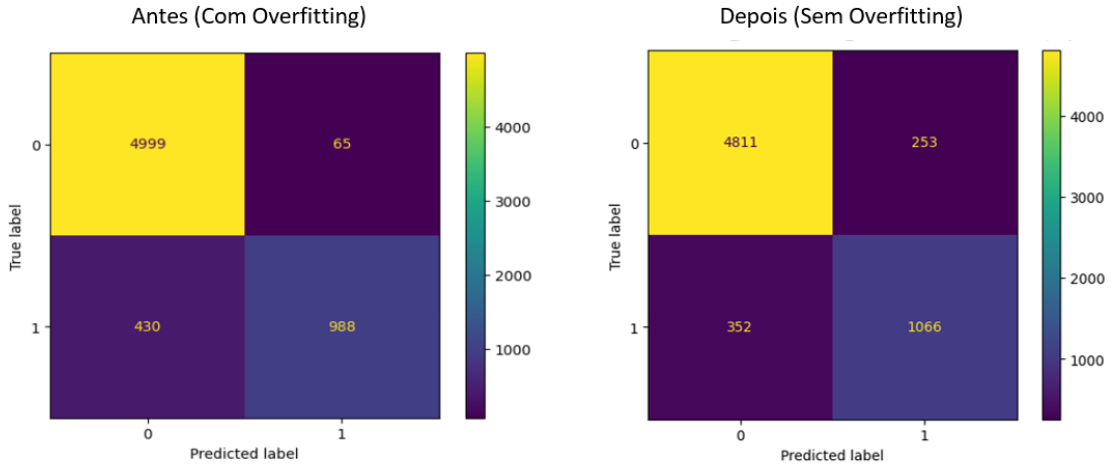
```
<matplotlib.legend.Legend at 0x7f08e72679a0>
```



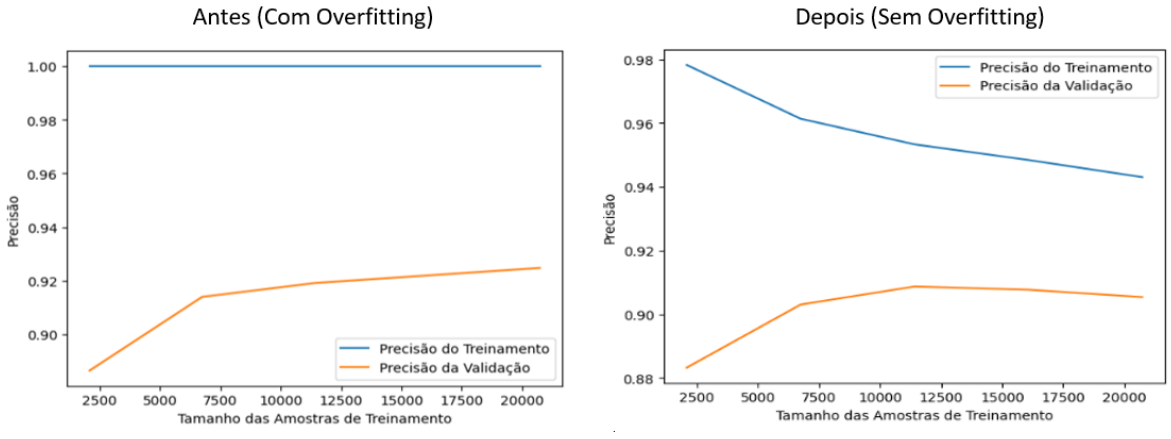
7. Apresentação dos Resultados

Após a realização do treinamento e teste do modelo de classificação, utilizando a técnica de *Random Forest* com otimização de hiperparâmetros, encontramos resultados promissores. Para avaliar a precisão do modelo, utilizamos a métrica de acurácia e o modelo apresentou uma acurácia de 0,90, o que indica que o modelo é capaz de classificar corretamente 90% das amostras.

Ao analisar a nova matriz de confusão, podemos verificar que o modelo apresentou poucas ocorrências de falsos positivos e falsos negativos, o que significa que ele tem uma boa capacidade de generalização e não está sofrendo de *overfitting*, ou seja, de ajuste excessivo aos dados de treinamento.



Para evitar o *overfitting*, foram utilizadas algumas medidas de prevenção, como a redução do número de *features*, a utilização de regularização e a poda de árvores de decisão.



Com base nos resultados obtidos através da aplicação dos modelos de *Logistic Regression, Random Forest* e *LightGBM*, podemos concluir que todos apresentaram uma alta capacidade de prever se um empréstimo será aprovado ou não, com acurácias entre 86% e 92%, respectivamente.

Se compararmos os modelos de *Logistic Regression* e *Random Forest*, iremos notar uma melhora significativa na precisão do modelo. Antes, a precisão estava em torno de 86%, e agora, após a otimização dos hiperparâmetros e a implementação de medidas de prevenção para evitar o *overfitting*, a precisão aumentou para 90%.

No entanto, é importante destacar que o modelo anterior que utilizava o *Random Forest e LightGBM* precisou passar por um tratamento de *overfitting* através da redução de *features*, utilização de regularização e poda de árvores de decisão, a fim de evitar ajuste excessivo aos dados de treinamento e melhorar sua capacidade de generalização.

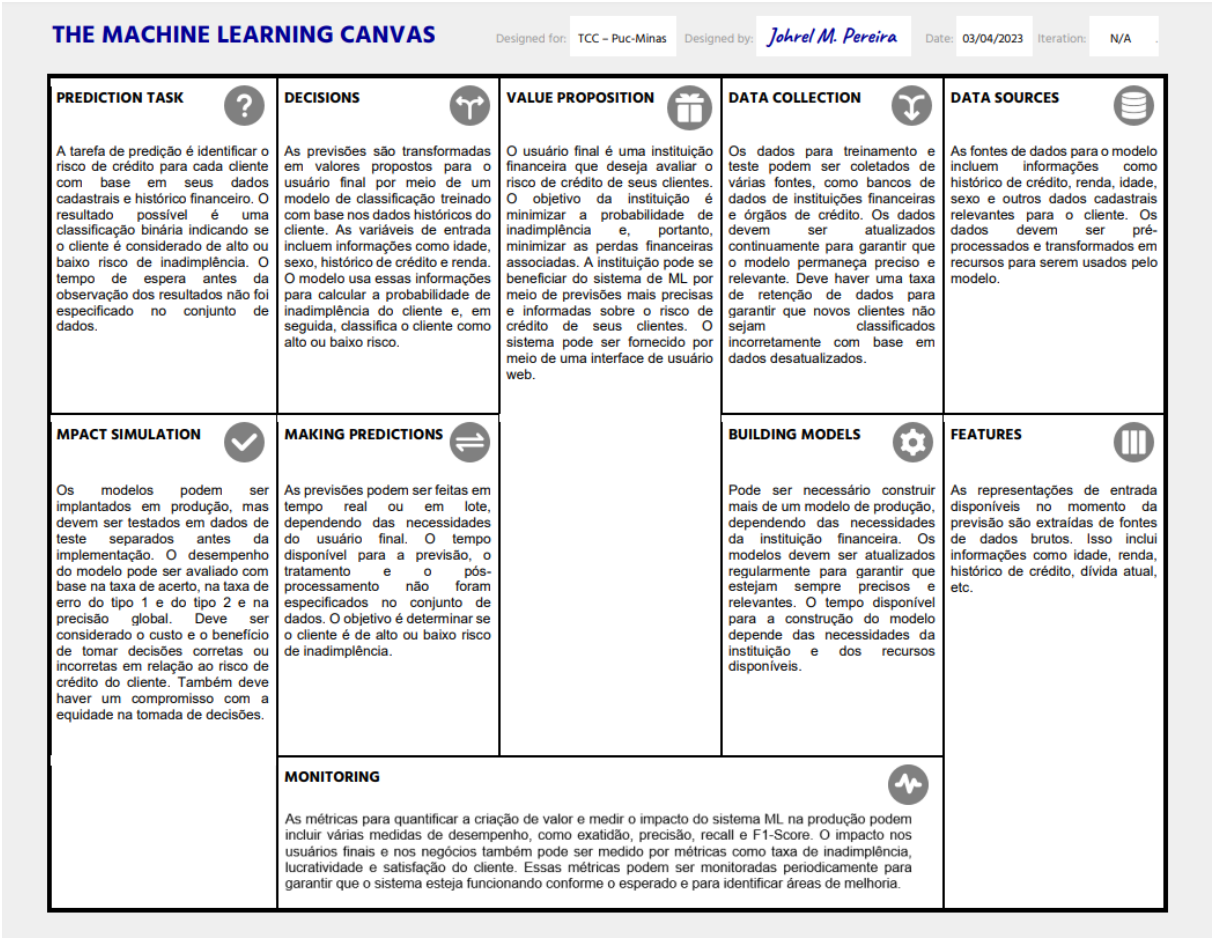
Comparando o modelo anterior, que apresentava uma acurácia de 92%, com o modelo otimizado que fizemos o tratamento de *overfitting*, podemos concluir que o novo modelo se manteve preciso, caindo apenas 2 pontos percentuais e atingindo 90% de acurácia.

Dessa forma, concluímos que o modelo de *Random Forest* otimizado é o mais adequado para a tarefa de prever se um empréstimo será aprovado ou não, e pode ser utilizado como uma ferramenta auxiliar para a tomada de decisões em instituições financeiras.

Por fim, abaixo temos o quadro “The Machine Learning Canvas”, sugerido por Dorard (2014), que, segundo o mesmo, tem como propósito ajudar a mapear os diversos aspectos envolvidos no desenvolvimento de um projeto de ML, incluindo as etapas de coleta e pré-processamento de dados, escolha do algoritmo de aprendizado de máquina, treinamento do modelo, avaliação de desempenho, e implantação do modelo em produção.

O canvas é uma espécie de "mapa" visual que organiza essas informações de forma clara e objetiva, permitindo que toda a equipe envolvida no projeto tenha uma

visão geral do processo de desenvolvimento e possa tomar decisões mais informadas e estratégicas. Além disso, o canvas pode ser utilizado como uma ferramenta de comunicação e colaboração entre as diferentes áreas envolvidas no projeto, como cientistas de dados, engenheiros de software e especialistas em negócios.



THE MACHINE LEARNING CANVAS

Designed for: TCC – Puc-Minas | Designed by: Johrel M. Pereira | Date: 03/04/2023 | Iteration: N/A

PREDICTION TASK

A tarefa de predição é identificar o risco de crédito para cada cliente com base em seus dados cadastrais e histórico financeiro. O resultado possível é uma classificação binária indicando se o cliente é considerado de alto ou baixo risco de inadimplência. O tempo de espera antes da observação dos resultados não foi especificado no conjunto de dados.

DECISIONS

As previsões são transformadas em valores propostos para o usuário final por meio de um modelo de classificação treinado com base nos dados históricos do cliente. As variáveis de entrada incluem informações como idade, sexo, histórico de crédito e renda. O modelo usa essas informações para calcular a probabilidade de inadimplência do cliente e, em seguida, classifica o cliente como alto ou baixo risco.

VALUE PROPOSITION

O usuário final é uma instituição financeira que deseja avaliar o risco de crédito de seus clientes. O objetivo da instituição é minimizar a probabilidade de inadimplência e, portanto, minimizar as perdas financeiras associadas. A instituição pode se beneficiar do sistema de ML por meio de previsões mais precisas e informadas sobre o risco de crédito de seus clientes. O sistema pode ser fornecido por meio de uma interface de usuário web.

DATA COLLECTION

Os dados para treinamento e teste podem ser coletados de várias fontes, como bancos de dados de instituições financeiras e órgãos de crédito. Os dados devem ser atualizados continuamente para garantir que o modelo permaneça preciso e relevante. Deve haver uma taxa de retenção de dados para garantir que novos clientes não sejam classificados incorretamente com base em dados desatualizados.

DATA SOURCES

As fontes de dados para o modelo incluem informações como histórico de crédito, renda, idade, sexo e outros dados cadastrais relevantes para o cliente. Os dados devem ser pré-processados e transformados em recursos para serem usados pelo modelo.

MPACT SIMULATION

Os modelos podem ser implantados em produção, mas devem ser testados em dados de teste separados antes da implementação. O desempenho do modelo pode ser avaliado com base na taxa de acerto, na taxa de erro do tipo 1 e do tipo 2 e na precisão global. Deve ser considerado o custo e o benefício de tomar decisões corretas ou incorretas em relação ao risco de crédito do cliente. Também deve haver um compromisso com a equidade na tomada de decisões.

MAKING PREDICTIONS

As previsões podem ser feitas em tempo real ou em lote, dependendo das necessidades do usuário final. O tempo disponível para a previsão, o tratamento e o pós-processamento não foram especificados no conjunto de dados. O objetivo é determinar se o cliente é de alto ou baixo risco de inadimplência.

BUILDING MODELS

Pode ser necessário construir mais de um modelo de produção, dependendo das necessidades da instituição financeira. Os modelos devem ser atualizados regularmente para garantir que estejam sempre precisos e relevantes. O tempo disponível para a construção do modelo depende das necessidades da instituição e dos recursos disponíveis.

FEATURES

As representações de entrada disponíveis no momento da previsão são extraídas de fontes de dados brutos. Isso inclui informações como idade, renda, histórico de crédito, dívida atual, etc.

MONITORING

As métricas para quantificar a criação de valor e medir o impacto do sistema ML na produção podem incluir várias medidas de desempenho, como exatidão, precisão, recall e F1-Score. O impacto nos usuários finais e nos negócios também pode ser medido por métricas como taxa de inadimplência, lucratividade e satisfação do cliente. Essas métricas podem ser monitoradas periodicamente para garantir que o sistema esteja funcionando conforme o esperado e para identificar áreas de melhoria.

8. Links

Link para o vídeo: https://youtu.be/3f9fd34Q_YM

Link para o repositório: https://github.com/Johrel/TCC-Puc_Minas.git

REFERÊNCIAS

AGTON, P. Análise de Sentimentos: Logistic Regression x Random Forest. Disponível em: <https://patrickagton.medium.com/análise-de-sentimentos-logistic-regression-x-random-forest-994492566f26>. Acesso em: 02/04/2023.

BANCO CENTRAL DO BRASIL. Relatório de Inflação. Disponível em: <https://www.bcb.gov.br/content/ri/relatorioinflacao/202303/ri202303p.pdf>. Acesso em: 22/03/2023.

BHATTIPROLU. Sreenivas. Tutorial 87 - Comparing Random Forest, XGBoost and LGBM for semantic image segmentation. [Vídeo]. Canal Apeer_micro, 1 de março de 2021. Disponível em: <https://www.youtube.com/watch?v=pbdseWOKG1U>. Acesso em: 02/04/2023.

CNC. (2023). Endividamento e inadimplência no Brasil. Disponível em: <https://static.poder360.com.br/2023/01/cnc-endividamento.pdf>. Acesso em: 22/03/2023.

DORARD, L. The Machine Learning Canvas. Disponível em: <https://www.louisdorard.com/machine-learning-canvas>. Acesso em: 03/04/2023.

LAOTSE. Credit Risk Dataset. Disponível em: <https://www.kaggle.com/datasets/laotse/credit-risk-dataset>. Acesso em: 22/03/2023.

LIGHTGBM. LGBMModel Python API. Disponível em: <https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMModel.html>. Acesso em: 02/04/2023.

APÊNDICE

```
# -*- coding: utf-8 -*-
"""Aprovação em Empréstimos.ipynb

Automatically generated by Colaboratory.

## **Usando a API do Kaggle qualquer pessoa pode reproduzir esse estudo.**

1.   Vá para a página do Kaggle e faça login na sua conta.
2.   Clique na sua foto de perfil e selecione "Conta"
3.   Role para baixo até a seção "API" e clique no botão "Criar nova chave da API".
4.   Um arquivo JSON com suas credenciais de API será baixado automaticamente.
5.   No Google Colab, faça o upload do arquivo JSON para sua sessão usando o código abaixo:
"""

from google.colab import files
files.upload()

"""6. Instale a API do Kaggle no Google Colab:"""

!pip install kaggle

"""7. Configure suas credenciais de API com a seguinte linha de código:"""

import os
os.environ['KAGGLE_USERNAME'] = "seu nome de usuário do Kaggle"
os.environ['KAGGLE_KEY'] = "sua chave da API do Kaggle"

"""8. Finalmente, você pode baixar o conjunto de dados desejado com o seguinte comando:"""

!kaggle datasets download -d laotse/credit-risk-dataset

"""O conjunto de dados será baixado no formato zip. Para descompactar, você pode usar o seguinte comando:"""

!unzip credit-risk-dataset.zip

"""Depois de descompactar o arquivo, é possível ler o conjunto de dados usando a biblioteca pandas no Python. Você pode fazer o seguinte:"""

df = pd.read_csv('credit_risk_dataset.csv')



"""## **Eu utilizei o Google Drive, então fica a seu critério!**
```

```
1. Para usá-lo, basta seguir o código abaixo:


"""

from google.colab import drive
drive.mount('/content/drive')

"""## **Para ler o dataset, você pode iniciar importando as bibliotecas que
irá utilizar. As usadas neste estudo seguem abaixo:**"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
import plotly.express as ex
import matplotlib.pyplot as plt
# %matplotlib inline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, learning_curve, Random-
izedSearchCV
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.metrics import ConfusionMatrixDisplay, PrecisionRecallDisplay,
precision_recall_curve
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import confusion_matrix, classification_report
from lightgbm import LGBMClassifier

"""2. Aqui você precisa alterar a pasta. Coloque a pasta onde seu arquivo está
para que consiga ler o mesmo"""

file = "/content/drive/My Drive/Colab Notebooks/TCC/credit_risk_dataset.csv"
df = pd.read_csv(file)

df.head()

# Buscando valores nulos ou missing
df.isnull().sum()
```

```
# Os resultados mostram que as colunas person_emp_length e loan_int_rate con-
tém valores nulos
# Esses valores precisam ser contabilizados em nosso modelo

media_pel = df["person_emp_length"].mean()
media_lir = df["loan_int_rate"].mean()
df["person_emp_length"] = df["person_emp_length"].fillna(media_pel)
df["loan_int_rate"] = df["loan_int_rate"].fillna(media_lir)
df.isnull().sum()

# Mostrar número de colunas e linhas
print("Número de linhas = ", len(df))
print("Número de colunas = ", len(df.columns))

# Obtendo um resumo estatístico rápido do conjunto de dados
df.describe()

# Olhando o dataset podemos verificar alguns outliers
# person_age = 144, obviamente alguém não viveria tanto, certo?! hahah
# person_emp_length = 123, tempo de serviço de 123 anos? tem algo errado!!

# Detectando mais outliers usando scatter plot
fig = px.scatter_matrix(df, dimensions=
            ["person_age", "person_income", "person_emp_length",
             "loan_amnt","loan_int_rate"],
            labels={col:col.replace('_', ' ') for col in df.columns},
            height=900, color='loan_status',
            color_continuous_scale=px.colors.diverging.Temps)
fig.show()

# Analisando o scatter plot, podemos ver que a coluna "person_income" tem um
outlier de $6M

# Vamos remover os outliers para assegurar a qualidade do modelo
df = df[df['person_age'] <= 100]   # Manter apenas dados onde a idade for
igual ou menor a 100 anos
df = df[df['person_emp_length'] <= 50] # Manter apenas dados onde o tempo de
serviço for igual ou menor a 50 anos
df = df[df['person_income'] <= 3000000] # Manter apenas dados onde a receita
for igual ou menor a $3 Milhões

# Verificar o dataset e buscar se existem mais outliers presentes
df.describe()

dups = df.duplicated()

df[dups]
```

```
df.query("person_age==23 & person_income==42000 &\
person_home_ownership=='RENT' & loan_int_rate==9.99")

# Mostrar número de colunas e linhas
print("Número de linhas = ", len(df))
print("Número de colunas = ", len(df.columns))

# Excluindo as duplicidades
df.drop_duplicates(inplace=True)

# Mostrar número de colunas e linhas
print("Número de linhas = ", len(df))
print("Número de colunas = ", len(df.columns))

# Vamos ver o número de empréstimos aprovados ou não, contidos no dataset
Negado = df[df.loan_status == 1].loan_status.count()
Aprovado = df[df.loan_status == 0].loan_status.count()

print("Total de pessoas no dataset: {:,}".format(len(df)))
print("Empréstimos negados = {:,}, Empréstimos aprovados =
{:,}".format(Negado, Aprovado))
print("Percentual de Empréstimos negados: {:.2f}%".format((Negado / len(df)) *
100))
print("Percentual de Empréstimos aprovados: {:.2f}%".format((Aprovado /
len(df)) * 100))

# Gráfico de pizza mostrando a relação de pessoas que tiveram o empréstimo
aprovado ou não
values = [21.87, 78.13]
colors = ['r', 'g']
explode = [0, 0.05]
labels = ['Empréstimos negados', 'Empréstimos aprovados']
textprops = {'color': 'white', 'weight': 'bold'}
plt.pie(values, colors=colors, labels=labels, explode=explode, au-
topct='%1.2f%%', textprops=textprops)
plt.title('Representação de empréstimos aprovados e não aprovados')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), labels=labels, font-
size=12)
plt.show()

# A seguir, vamos ver a relação entre loan_status em relação a loan_grade e
loan_percent_income
fig = px.box(df, x="loan_grade", y="loan_percent_income",
color="loan_status",
color_discrete_sequence=px.colors.qualitative.Dark24,
labels={col:col.replace('_','-') for col in df.columns},
category_orders={'loan_grade':['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']})
fig.update_layout(legend=dict(orientation="h", yanchor="bottom",
```

```
                y=1.02, xanchor="right", x=1))
fig.show()

# O gráfico abaixo mostra a relação entre diferentes categorias, incluindo o
status do empréstimo,
# histórico de inadimplência, finalidade do empréstimo, posse de casa e o ra-
ting.
fig = px.parallel_categories(df,
    color_continuous_scale=px.colors.sequential.Peach,
    color="loan_status",
    dimensions=['cb_person_default_on_file',
'loan_intent','person_home_ownership','loan_grade'],
    labels={col:col.replace('_', '-') for col in df.columns})
fig.show()

#Aqui separamos as variáveis numéricas das categóricas. E abaixo as descreve-
mos.
df_num = df.select_dtypes(include='number')
df_cat = df.select_dtypes(include=['object','category'])

df_num.describe().T.round(2)

df_cat.describe().T

#Abaixo criamos um gráfico de calor para mostrar as correlações entre as vari-
áveis numéricas de um conjunto de dados.
plt.figure(figsize = (14,14))
plt.title('Gráfico de correlação de risco de crédito (Pearson)')
corr = df_num.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=
.1,cmap="Blues")
plt.show()

#Abaixo criamos alguns gráficos de densidade para mostrar a distribuição de
cada variável numérica no conjunto de dados.
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 7), sharex = False,
sharey = False)
axes = axes.ravel()
cols = df_num.columns[:]

for col, ax in zip(cols, axes):
    data = df_num
    sns.kdeplot(data=data, x=col, fill=True, ax=ax)
    ax.set(title=f'Distribuição da variável: {col}', xlabel=None)

fig.delaxes(axes[8])
fig.tight_layout()
plt.show()
```

```
df.select_dtypes(include=['object','category']).columns.tolist()

df_cat = df.select_dtypes(include=['object','category'])
df_cat.count()

#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção de casa própria
ex.pie(df_cat,names='person_home_ownership',title='Proporção de Casa Própria',hole=0.33)

#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção de intenção de empréstimo
ex.pie(df_cat,names='loan_intent',title='Proporção de Intenção de Empréstimo',hole=0.33)

#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção do rating
ex.pie(df_cat,names='loan_grade',title='Proporção do Grau de Empréstimo (Rating)',hole=0.33)

#Abaixo criamos um gráfico de Pizza com dados percentuais sobre a proporção do histórico de inadimplência
ex.pie(df_cat,names='cb_person_default_on_file',title='Proporção de pessoas com Histórico de Inadimplência',hole=0.33)

"""**Algoritmo de Regressão Logística**"""

# Verificar os tipos de dados
print(df.dtypes)

# Separar a variável target do conjunto de dados
X = df.drop(['loan_status'], axis=1)
y = df['loan_status']

# Transformar as variáveis categóricas em variáveis numéricas
X = pd.get_dummies(X)

# Separar os dados de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Definir os parâmetros a serem otimizados
params = {'penalty': ['l1', 'l2'],
          'C': [0.01, 0.1, 1, 10, 100],
          'solver': ['liblinear', 'saga']}

# Criar o objeto GridSearchCV
```

```
grid_search = GridSearchCV(LogisticRegression(random_state=42), params, cv=5,
scoring='accuracy')

# Treinar o modelo com os dados de treinamento
grid_search.fit(X_train, y_train)

# Mostrar os melhores parâmetros encontrados
print(grid_search.best_params_)

# Criar o modelo com os melhores parâmetros encontrados
model = LogisticRegression(random_state=42, penal-
ty=grid_search.best_params_['penalty'],
                           C=grid_search.best_params_['C'], solv-
er=grid_search.best_params_['solver'])

# Treinar o modelo com os dados de treinamento
model.fit(X_train, y_train)

# Fazer a previsão com os dados de teste
y_pred = model.predict(X_test)

# Avaliar a precisão do modelo com os dados de teste
print("Acurácia:", accuracy_score(y_test, y_pred))

"""**Algoritmo de Floresta aleatória (Random Forest Classifier)**"""

# Cópia do Dataframe usado na regressão logística
df2 = df.copy()

# X e y serão considerados como todos os dados de treinamento
# X_test2 e y_test2 serão considerados como dados fora da amostra para avalia-
ção do modelo

X, X_test2, y, y_test2 = train_test_split(df2.drop('loan_status', axis=1),
df2['loan_status'],
                                          random_state=0,  test_size=0.2, strat-
ify=df2['loan_status'],
                                          shuffle=True)

df2['loan_status'].value_counts(normalize=True)

X[['person_income', 'loan_amnt', 'loan_percent_income']].head()

# Aqui vamos aplicar uma técnica de processamento ao remover a coluna do per-
centual de receita,
# para tentar melhorar a acurácia dos modelos que usaremos a seguir.
X.drop('loan_percent_income', axis=1, inplace=True)
X_test2.drop('loan_percent_income', axis=1, inplace=True)
```

```
# Abaixo imprimimos o número de valores únicos em cada coluna do dataset,
# com o objetivo do gráfico abaixo é ajudar a entender a distribuição dos va-
lores em cada coluna.

for col in X:
    print(col, '--->', X[col].nunique())
    if X[col].nunique()<20:
        print(X[col].value_counts(normalize=True)*100)
    print()

# Aqui novamente separamos as colunas númericas com outro nome para não alte-
rar o que fizemos acima.
num_cols = [col for col in X if X[col].dtypes != 'O']
num_cols

# Abaixo plotamos alguns histogramas utilizando os dados das colunas númericas
for col in num_cols:
    sns.histplot(X[col])
    plt.show()

# já que dropamos alguns dados de X, precisamos passar essas atualizações para
y também
y = y[X.index]

# Aqui novamente separamos as colunas categóricas com outro nome para não al-
terar o que fizemos acima.
cat_cols = [col for col in X if X[col].dtypes == 'O']
cat_cols

# O código abaixo cria um pipeline de pré-processamento de dados numéricos,
# com o objetivo de tratar os dados antes de aplicar um modelo de aprendizado
de máquina.
num_pipe = Pipeline([
    ('impute', IterativeImputer()),
    ('scale', StandardScaler()),
])

# O código abaixo usa o ColumnTransformer que aplica o pipeline que contém
tanto dados numéricos como categóricos.
ct = ColumnTransformer([
    ('num_pipe', num_pipe, num_cols),
    ('cat_cols', OneHotEncoder(sparse=False, handle_unknown='ignore'),
cat_cols)
], remainder='passthrough')

# O código abaixo define um grid de hiperparâmetros para os modelos de classi-
ficação, RandomForestClassifier e LGBMClassifier.
```

```
# A grade especifica diferentes valores para vários parâmetros do modelo, como
o número de estimadores,
# a taxa de aprendizado e o tipo de reforço para o LGBMClassifier.
# O objetivo é procurar pelos melhores hiperparâmetros que otimizam o desempe-
nho do modelo em um conjunto de dados de treinamento.

grid = {
    RandomForestClassifier(random_state=0, n_jobs=-1,
class_weight='balanced'):
    {'model__n_estimators':[300,400,500],
     'coltf__num_pipe__impute__estimator': [LinearRegression(), Random-
ForestRegressor(random_state=0),
                                           KNeighborsRegressor()]},

    LGBMClassifier(class_weight='balanced', random_state=0, n_jobs=-1):
    {'model__n_estimators':[300,400,500],
     'model__learning_rate':[0.001,0.01,0.1,1,10],
     'model__boosting_type': ['gbdt', 'goss', 'dart'],
     'coltf__num_pipe__impute__estimator':[LinearRegression(), Random-
ForestRegressor(random_state=0),
                                           KNeighborsRegressor()]},
}

for clf, param in grid.items():
    print(clf)
    print('-'*50)
    print(param)
    print('\n')

# O código abaixo realiza uma busca aleatória de hiperparâmetros usando Rando-
mizedSearchCV em um grid de modelos e parâmetros predefinidos,
# treina os modelos resultantes no conjunto de dados X e y, e armazena os re-
sultados da validação cruzada em um DataFrame.
# Ele também armazena o melhor modelo encontrado para cada algoritmo em um
dicionário chamado "best_algos".

full_df = pd.DataFrame()
best_algos = {}

for clf, param in grid.items():
    pipe = Pipeline([
    ('coltf', ct),
    ('model', clf)
])

    gs = RandomizedSearchCV(estimator=pipe, param_distributions=param, scor-
ing='accuracy',
                            n_jobs=-1, verbose=3, n_iter=4, random_state=0)
```

```
    gs.fit(X, y)

    all_res = pd.DataFrame(gs.cv_results_)

    temp = all_res.loc[:, ['params', 'mean_test_score']]
    algo_name = str(clf).split('(')[0]
    temp['algo'] = algo_name

    full_df = pd.concat([full_df, temp], ignore_index=True)
    best_algos[algo_name] = gs.best_estimator_

# Aqui ordenamos de forma crescente os parâmetros de acordo com o seu score
médio
full_df.sort_values('mean_test_score', ascending=False)

# Aqui retornamos o valor do primeiro hiperparâmetro selecionado pelo modelo
que obteve
# a melhor pontuação de precisão média nos resultados do processo de validação
cruzada.
full_df.sort_values('mean_test_score', ascending=False).iloc[0, 0]

# Aqui selecionamos o melhor algoritmo de acordo com sua pontuação
be = best_algos['RandomForestClassifier']
be

# Aqui ajustamos os dados treinados para as variáveis X e Y
be.fit(X, y)

preds = be.predict(X_test2)

confusion_matrix(y_test2, preds)

# Aqui criamos uma matriz de confusão para identificar os falsos negativos e
positivos,
# permitindo a visualização dos acertos e erros do modelo
ConfusionMatrixDisplay.from_estimator(be, X_test2, y_test2)

# Aqui imprimimos o relatório de classificação
print(classification_report(y_test2, preds))

# Aqui mostramos o score atingido pelo algoritmo
be.score(X_test2, y_test2)

"""###### **Precision Recall Curve**"""

# O gráfico abaixo mostra a curva precision-recall e a linha de base para um
modelo de classificação treinado,
```

```
# representando a qualidade das previsões do modelo em diferentes níveis de
threshold.
PrecisionRecallDisplay.from_estimator(estimator=be, X=X_test2, y=y_test2,
name='model AUC')
baseline = y_test2.sum() / len(y_test2)
plt.axhline(baseline, ls='--', color='r', label=f'Baseline model
({round(baseline,2)})')
plt.legend(loc='best')

"""###### **Learning Curve**"""

# Logo abaixo fazemos a curva de aprendizagem a fim de visualizar o desempenho
do modelo em relação ao tamanho do conjunto de treinamento.
# Os dados gerados podem ajudar identificar se o modelo está sofrendo de over-
fitting ou underfitting.
a, b, c = learning_curve(be, X, y, n_jobs=-1, scoring='accuracy')

# Aqui plotamos os dados da curva de aprensizagem e descubrimos a ocorrência
do overfitting
plt.plot(a, b.mean(axis=1), label='Precisão do Treinamento')
plt.plot(a, c.mean(axis=1),  label='Precisão da Validação')
plt.xlabel('Tamanho das Amostras de Treinamento')
plt.ylabel('Precisão')
plt.legend()

"""##### Overfitting:

1. Alta precisão de treinamento (--- viés baixo)
2. Baixa precisão de teste/validação (--- alta variância)
3. Grande lacuna entre as curvas de treinamento e validação (--- alta variân-
cia)
4. O overfitting torna um modelo muito complexo e aprende até o "ruído" nos
dados, o que é indesejável




"""



""" Medidas corretivas:

1. Adicione mais amostras de treinamento, se possível, para permitir que o
modelo aprenda melhor


2. Faça um modelo mais simples / reduza a complexidade do modelo:
```

```
  * tente reduzir o número de recursos
  * tente aumentar a regularização (lambda)
  * tente podar as árvores de decisão

"""

#Novo Grid usando apenas o algoritmo Random Forest

grid = {

    RandomForestClassifier(random_state=0, n_jobs=-1,
class_weight='balanced'):
    {'model__n_estimators':[100,200,300],
     'model__max_depth':[5, 9, 13],
     'model__min_samples_split':[4,6,8],
     'coltf__num_pipe__impute__estimator': [LinearRegression(), Random-
ForestRegressor(random_state=0),
                                            KNeighborsRegressor()]},
}

for clf, param in grid.items():
    print(clf)
    print('-'*50)
    print(param)
    print('\n')

# Novamente fazemos a busca aleatória de hiperparâmetros no novo grid
full_df = pd.DataFrame()
best_algos = {}

for clf, param in grid.items():
    pipe = Pipeline([
    ('coltf', ct),
    ('model', clf)
])

    gs = RandomizedSearchCV(estimator=pipe, param_distributions=param, scor-
ing='accuracy',
                            n_jobs=-1, verbose=3, n_iter=4)

    gs.fit(X, y)

    all_res = pd.DataFrame(gs.cv_results_)

    temp = all_res.loc[:, ['params', 'mean_test_score']]
    algo_name = str(clf).split('(')[0]
    temp['algo'] = algo_name
```

```
    full_df = pd.concat([full_df, temp])
    best_algos[algo_name] = gs.best_estimator_

# Aqui selecionamos o melhor algoritmo de acordo com sua pontuação
be = best_algos['RandomForestClassifier']
be

# Aqui ajustamos os dados treinados para as variáveis X e Y
be.fit(X, y)

preds = be.predict(X_test2)

confusion_matrix(y_test2, preds)

# Aqui criamos uma matriz de confusão para identificar os falsos negativos e
positivos,
# permitindo a visualização dos acertos e erros do modelo
ConfusionMatrixDisplay.from_estimator(be, X_test2, y_test2)

# Aqui imprimimos o relatório de classificação
print(classification_report(y_test2, preds))

# Aqui mostramos o score atingido pelo algoritmo
be.score(X_test2, y_test2)

# O gráfico abaixo mostra a curva precision-recall e a linha de base para um
modelo de classificação treinado,
# representando a qualidade das previsões do modelo em diferentes níveis de
threshold.
PrecisionRecallDisplay.from_estimator(be, X_test2, y_test2)
baseline = y_test2.sum() / len(y_test)
plt.axhline(baseline, ls='--', color='r', label=f'Baseline model
({round(baseline,2)})')
plt.legend(loc='best')

# Logo abaixo fazemos a nova curva de aprendizagem
a, b, c = learning_curve(be, X, y, n_jobs=-1, cv=5)

# Aqui plotamos os dados da nova curva de aprensizagem
plt.plot(a, b.mean(axis=1), label='Precisão do Treinamento')
plt.plot(a, c.mean(axis=1),  label='Precisão da Validação')
plt.xlabel('Tamanho das Amostras de Treinamento')
plt.ylabel('Precisão')
plt.legend()
```