GabrielFigueiredoGomes_AlgoritmosSupervisionadosParaClassificacao

March 17, 2023

0.1 Escolha uma base de dados.

''A base da dados escolhida foi a do kaggle sobre preços de venda de de imóveis :https://www.kaggle.com/datasets/laotse/credit-risk-dataset

0.2 1. Explique a motivação de uso da base escolhida

Visto que trabalho no setor bancário, resolvi escolher esse Dataset pois acredito que esse estudo de caso irá agregar valor a minha rotina de trabalho, para que possa migrar de carreira com os conhecimento que já possuou e que vou adiquirir.

0.3 2. Descreva as variáveis presentes na base. Quais são as variáveis? Quais são os tipos de variáveis (discreta, categórica, contínua)? Quais são as médias e desvios padrões?

COLUNA	DESCRIÇÃO	DISC / CATEG / CONT
idade	Idade	DISC
renda_anual	Renda	CONT
tipo_imovel	Se possui casa propia ou não	CATEG
tempo_de_trabalho_anos	Tempo de serviço (em anos)	CONT
objetivo_do_emprestimo	Intenção/obejtivo do empréstimo	CATEG
grau_do_emprestimo	Grau do crédito	CATEG
valor_do_emprestimo	Valor total crédito	CONT
taxa_de_juros	Taxa de juro	CONT
inadimplente	Status do empréstimo (0 negado 1 é aprovado)	CATEG
compromentimento_renda	Percentual da renda sobre emprestimo	CONT
padrao_historico	Padrão histórico	CONT
prazo_emprestimo	Prazo do crédito	CONT

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import plotly.express as px
import seaborn as sns

from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn import metrics
```

[2]:
```
# importando dados

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

[2]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_default_on_file	cb_person_cred_hist_length
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1	0.59	Y	3
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0	0.10	N	2
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1	0.57	N	3
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1	0.53	N	2
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1	0.55	Y	4

[3]:
```
# Renomeando as colunas para facilitar o entendimento.

rename_colunas = {'person_age': 'idade',
           'person_income': 'renda_anual',
           'person_home_ownership': 'tipo_imovel',
           'person_emp_length': 'tempo_de_trabalho_anos',
           'loan_intent': 'objetivo_do_emprestimo',
```

```
            'loan_grade': 'grau_do_emprestimo',
            'loan_amnt': 'valor_do_emprestimo',
            'loan_int_rate': 'taxa_de_juros',
            'loan_status': 'inadimplente', # 0 não devedor, 1 devedor
            'loan_percent_income': 'comprometimento_renda',
            'cb_person_default_on_file': 'padrao_historico',
            'cb_person_cred_hist_length': 'prazo_emprestimo'}

df = df.rename(columns=rename_colunas)

df.head()
```

[3]:

	idade	renda_anual	tipo_imovel	tempo_de_trabalho_anos	objetivo_do_emprestimo	grau_do_emprestimo	valor_do_emprestimo	taxa_de_juros	inadimplente	comprometimento_renda	padrao_historico	prazo_emprestimo
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1	0.59	Y	3
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0	0.10	N	2
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1	0.57	N	3
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1	0.53	N	2
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1	0.55	Y	4

[4]:
```
# Renomeando os registro na coluna tipo_imovel.

tipo_imovel = {
    'RENT': 'ALUGADO',
    'OWN': 'PROPIO',
    'MORTGAGE': 'FINANCIADO',
    'OTHER': 'OUTROS'
```

```
}

df.tipo_imovel = df.tipo_imovel.map(tipo_imovel)
```

[5]:
```
# Renomeando os registro da coluna objetivo_do_emprestimo

objetivo_emprestimo = {
    'PERSONAL': 'PESSOAL',
    'EDUCATION': 'EDUCACAO',
    'MEDICAL': 'SAUDE',
    'VENTURE': 'INVESTIMENTO',
    'HOMEIMPROVEMENT': 'REFORMA',
    'DEBTCONSOLIDATION': 'DIVIDA'
}

df.objetivo_do_emprestimo = df.objetivo_do_emprestimo.map(objetivo_emprestimo)
df
```

[5]:

	idade	renda_anual	tipo_imovel	tempo_de_trabalho_anos	\
0	22	59000	ALUGADO	123.0	
1	21	9600	PROPIO	5.0	
2	25	9600	FINANCIADO	1.0	
3	23	65500	ALUGADO	4.0	
4	24	54400	ALUGADO	8.0	
…	…	…	…	…	
32576	57	53000	FINANCIADO	1.0	
32577	54	120000	FINANCIADO	4.0	
32578	65	76000	ALUGADO	3.0	
32579	56	150000	FINANCIADO	5.0	
32580	66	42000	ALUGADO	2.0	

	objetivo_do_emprestimo	grau_do_emprestimo	valor_do_emprestimo	\
0	PESSOAL	D	35000	
1	EDUCACAO	B	1000	
2	SAUDE	C	5500	
3	SAUDE	C	35000	
4	SAUDE	C	35000	
…	…	…	…	
32576	PESSOAL	C	5800	
32577	PESSOAL	A	17625	
32578	REFORMA	B	35000	
32579	PESSOAL	B	15000	
32580	SAUDE	B	6475	

	taxa_de_juros	inadimplente	comprometimento_renda	padrao_historico	\
0	16.02	1	0.59	Y	
1	11.14	0	0.10	N	

```
2              12.87             1                    0.57               N
3              15.23             1                    0.53               N
4              14.27             1                    0.55               Y
…              …            …                    …              …
32576          13.16             0                    0.11               N
32577           7.49             0                    0.15               N
32578          10.99             1                    0.46               N
32579          11.48             0                    0.10               N
32580           9.99             0                    0.15               N

       prazo_emprestimo
0                      3
1                      2
2                      3
3                      2
4                      4
…                    …
32576                 30
32577                 19
32578                 28
32579                 26
32580                 30

[32581 rows x 12 columns]
```

[6]:
```
#Informações base de dados

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   idade                   32581 non-null  int64
 1   renda_anual             32581 non-null  int64
 2   tipo_imovel             32581 non-null  object
 3   tempo_de_trabalho_anos  31686 non-null  float64
 4   objetivo_do_emprestimo  32581 non-null  object
 5   grau_do_emprestimo      32581 non-null  object
 6   valor_do_emprestimo     32581 non-null  int64
 7   taxa_de_juros           29465 non-null  float64
 8   inadimplente            32581 non-null  int64
 9   compromentimento_renda  32581 non-null  float64
 10  padrao_historico        32581 non-null  object
 11  prazo_emprestimo        32581 non-null  int64
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

[7]:
```
# Tamanho dos DataSet.

df.shape
```

[7]:
```
(32581, 12)
```

[8]:
```
# Tabela Drescritiva

df.describe()
```

[8]:
```
              idade   renda_anual  tempo_de_trabalho_anos  \
count  32581.000000  3.258100e+04            31686.000000
mean      27.734600  6.607485e+04                4.789686
std        6.348078  6.198312e+04                4.142630
min       20.000000  4.000000e+03                0.000000
25%       23.000000  3.850000e+04                2.000000
50%       26.000000  5.500000e+04                4.000000
75%       30.000000  7.920000e+04                7.000000
max      144.000000  6.000000e+06              123.000000

       valor_do_emprestimo  taxa_de_juros  inadimplente  \
count         32581.000000   29465.000000  32581.000000
mean           9589.371106      11.011695      0.218164
std            6322.086646       3.240459      0.413006
min             500.000000       5.420000      0.000000
25%            5000.000000       7.900000      0.000000
50%            8000.000000      10.990000      0.000000
75%           12200.000000      13.470000      0.000000
max           35000.000000      23.220000      1.000000

       comprometimento_renda  prazo_emprestimo
count           32581.000000      32581.000000
mean                0.170203          5.804211
std                 0.106782          4.055001
min                 0.000000          2.000000
25%                 0.090000          3.000000
50%                 0.150000          4.000000
75%                 0.230000          8.000000
max                 0.830000         30.000000
```

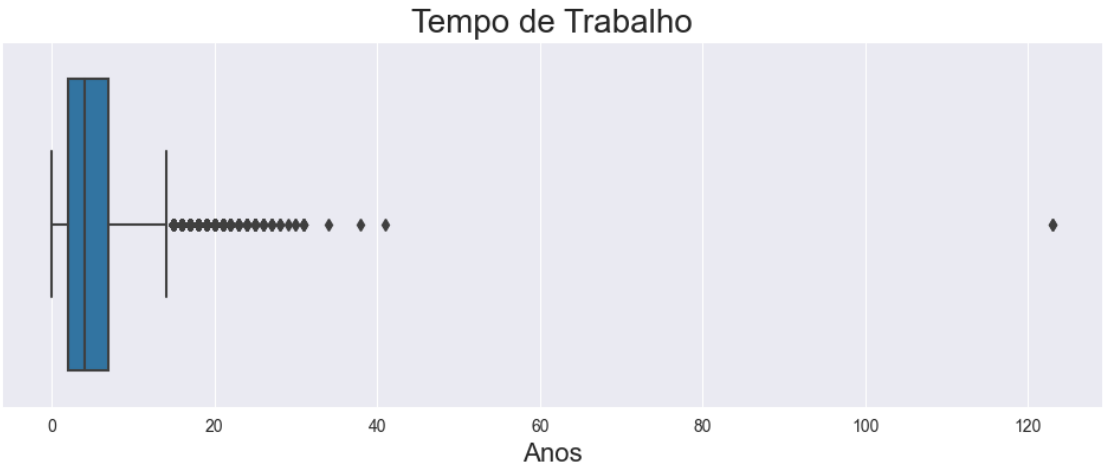
[9]:
```
# Matriz de correlação

corr = df.corr(numeric_only=True)
corr
```

[9]:
```
                          idade  renda_anual  tempo_de_trabalho_anos  \
idade                  1.000000     0.173202                0.163106
```

```
renda_anual              0.173202     1.000000                0.134268
tempo_de_trabalho_anos   0.163106     0.134268                1.000000
valor_do_emprestimo      0.050787     0.266820                0.113082
taxa_de_juros            0.012580     0.000792               -0.056405
inadimplente            -0.021629    -0.144449               -0.082489
comprometimento_renda   -0.042411    -0.254471               -0.054111
prazo_emprestimo         0.859133     0.117987                0.144699

                        valor_do_emprestimo  taxa_de_juros  inadimplente  \
idade                              0.050787       0.012580     -0.021629
renda_anual                        0.266820       0.000792     -0.144449
tempo_de_trabalho_anos             0.113082      -0.056405     -0.082489
valor_do_emprestimo                1.000000       0.146813      0.105376
taxa_de_juros                      0.146813       1.000000      0.335133
inadimplente                       0.105376       0.335133      1.000000
comprometimento_renda              0.572612       0.120314      0.379366
prazo_emprestimo                   0.041967       0.016696     -0.015529

                        comprometimento_renda  prazo_emprestimo
idade                               -0.042411          0.859133
renda_anual                         -0.254471          0.117987
tempo_de_trabalho_anos              -0.054111          0.144699
valor_do_emprestimo                  0.572612          0.041967
taxa_de_juros                        0.120314          0.016696
inadimplente                         0.379366         -0.015529
comprometimento_renda                1.000000         -0.031690
prazo_emprestimo                    -0.031690          1.000000
```

[10]:
```
# Plot Matriz de correlação

fig = px.imshow(corr, text_auto=True, width=1000, height=1000)
fig.show()
```

[11]:
```
# indetificando e removendo outliers

sns.set_style("darkgrid")
plot = sns.boxplot(x='tempo_de_trabalho_anos', data=df)
plot.figure.set_size_inches(12, 4)
plot.set_title('Tempo de Trabalho', fontsize=20)
plot.set_xlabel('Anos', fontsize=16);
```



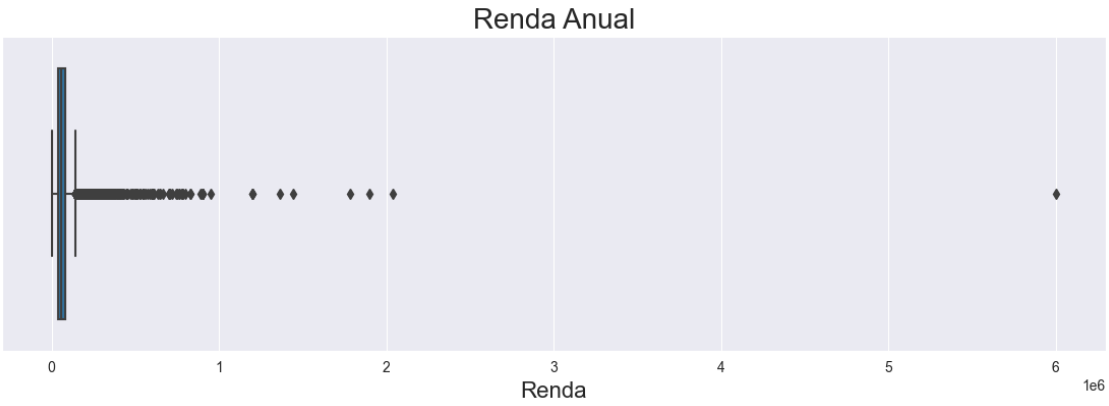
[12]:
```
# Existe outliers na coluna tempo_de_trabalho_em_anos.

df[df['tempo_de_trabalho_anos'] > 120]
```

[12]:
```
     idade  renda_anual tipo_imovel  tempo_de_trabalho_anos  \
0       22        59000     ALUGADO                   123.0
210     21       192000  FINANCIADO                   123.0

    objetivo_do_emprestimo grau_do_emprestimo  valor_do_emprestimo  \
0                 PESSOAL                  D                35000
210         INVESTIMENTO                   A                20000

     taxa_de_juros  inadimplente  comprometimento_renda padrao_historico  \
0            16.02             1                   0.59                Y
210           6.54             0                   0.10                N

     prazo_emprestimo
0                   3
210                 4
```

[13]:
```
# Revemovendo outliers

df = df.drop(df[df['tempo_de_trabalho_anos'] > 120].index)
df[df['tempo_de_trabalho_anos'] > 120]
```

[13]:
```
Empty DataFrame
Columns: [idade, renda_anual, tipo_imovel, tempo_de_trabalho_anos,
objetivo_do_emprestimo, grau_do_emprestimo, valor_do_emprestimo, taxa_de_juros,
inadimplente, comprometimento_renda, padrao_historico, prazo_emprestimo]
Index: []
```

[14]:
```
# Outliers da renda anual

plot = sns.boxplot(x='renda_anual', data=df)
plot.figure.set_size_inches(14, 4)
plot.set_title('Renda Anual', fontsize=20)
plot.set_xlabel('Renda', fontsize=16);
```



[15]:
```
df[df['renda_anual'] >= 6000000]
```

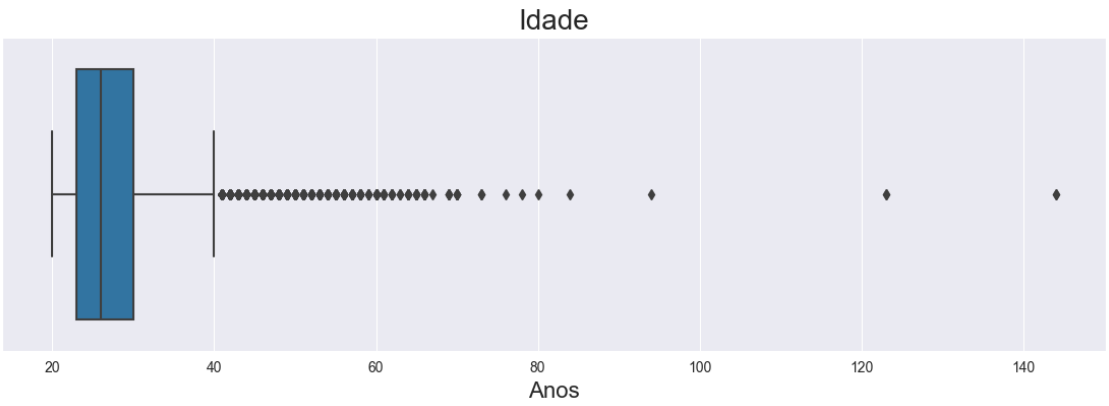
[15]:
```
       idade  renda_anual tipo_imovel  tempo_de_trabalho_anos  \
32297    144      6000000  FINANCIADO                    12.0

      objetivo_do_emprestimo grau_do_emprestimo  valor_do_emprestimo  \
32297                PESSOAL                  C                 5000

       taxa_de_juros  inadimplente  comprometimento_renda padrao_historico  \
32297          12.73             0                    0.0                N

       prazo_emprestimo
32297                25
```

[16]:
```
df.drop(df[df['renda_anual'] >= 6000000].index, inplace=True)
df[df['renda_anual'] >= 6000000]
```

[16]:
```
Empty DataFrame
Columns: [idade, renda_anual, tipo_imovel, tempo_de_trabalho_anos,
objetivo_do_emprestimo, grau_do_emprestimo, valor_do_emprestimo, taxa_de_juros,
inadimplente, comprometimento_renda, padrao_historico, prazo_emprestimo]
Index: []
```

[17]:
```
# Outliers da idade

plot = sns.boxplot(x='idade', data=df)
```

```
plot.figure.set_size_inches(14, 4)
plot.set_title('Idade', fontsize=20)
plot.set_xlabel('Anos', fontsize=16);
```



[18]:
```
df[df['idade'] > 85]
```

[18]:

	idade	renda_anual	tipo_imovel	tempo_de_trabalho_anos	\
81	144	250000	ALUGADO	4.0	
183	144	200000	FINANCIADO	4.0	
575	123	80004	ALUGADO	2.0	
747	123	78000	ALUGADO	7.0	
32416	94	24000	ALUGADO	1.0	

	objetivo_do_emprestimo	grau_do_emprestimo	valor_do_emprestimo	\
81	INVESTIMENTO	C	4800	
183	EDUCACAO	B	6000	
575	EDUCACAO	B	20400	
747	INVESTIMENTO	B	20000	
32416	SAUDE	C	6500	

	taxa_de_juros	inadimplente	comprometimento_renda	padrao_historico	\
81	13.57	0	0.02	N	
183	11.86	0	0.03	N	
575	10.25	0	0.25	N	
747	NaN	0	0.26	N	
32416	NaN	0	0.27	N	

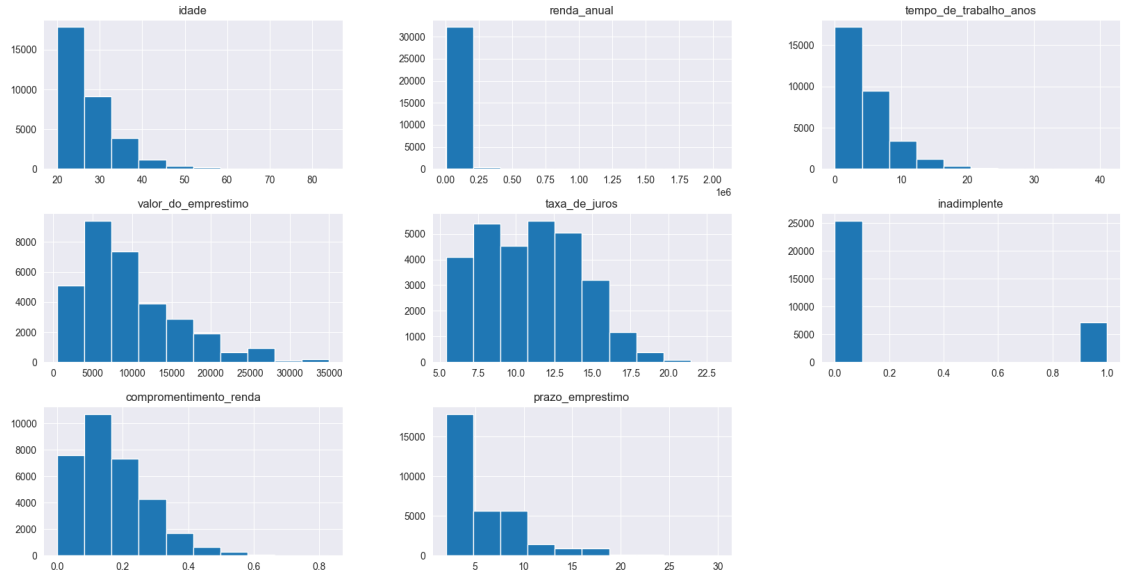
	prazo_emprestimo
81	3
183	2
575	3
747	4

```
32416                  27
```

[19]:
```
# Removendo outliers (idade > 85)

df.drop(df[df['idade'] > 85].index, inplace=True)
df.loc[df['idade'] > 85]
```

[19]:
```
Empty DataFrame
Columns: [idade, renda_anual, tipo_imovel, tempo_de_trabalho_anos,
objetivo_do_emprestimo, grau_do_emprestimo, valor_do_emprestimo, taxa_de_juros,
inadimplente, comprometimento_renda, padrao_historico, prazo_emprestimo]
Index: []
```

[20]:
```
# Gráfico distribuição

distribuicao_dados = df.hist(figsize=(20, 10))
```



[21]:
```
#Identificando valores nulos.

df.isnull().sum()
```

[21]:
```
idade                      0
renda_anual                0
tipo_imovel                0
tempo_de_trabalho_anos   895
objetivo_do_emprestimo     0
grau_do_emprestimo         0
valor_do_emprestimo        0
```

```
taxa_de_juros              3114
inadimplente                  0
comprometimento_renda         0
padrao_historico              0
prazo_emprestimo              0
dtype: int64
```

[22]:
```
# Removendo dados nulos

df = df.dropna()

df.isnull().sum()
```

[22]:
```
idade                     0
renda_anual               0
tipo_imovel               0
tempo_de_trabalho_anos    0
objetivo_do_emprestimo    0
grau_do_emprestimo        0
valor_do_emprestimo       0
taxa_de_juros             0
inadimplente              0
comprometimento_renda     0
padrao_historico          0
prazo_emprestimo          0
dtype: int64
```

[23]:
```
# Tranformando variaveis categoricas em números.

label_enconder = preprocessing.LabelEncoder()

df['tipo_imovel'] = label_enconder.fit_transform(df['tipo_imovel'])

numerico_tipo_imovel = {'categoria': ['PROPIO', 'FINANCIADO', 'ALUGADO',␣
 ↪'OUTROS'], 'numerico': [3, 1, 0, 2]}
numerico_tipo_imovel = pd.DataFrame(numerico_tipo_imovel)

df['objetivo_do_emprestimo'] = label_enconder.
 ↪fit_transform(df['objetivo_do_emprestimo'])

numerico_objetivo_do_emprestimo = {'categoria': ['EDUCACAO', 'SAUDE',␣
 ↪'INVESTIMENTO', 'PESSOAL', 'REFORMA', 'DIVIDA'], 'numerico': [1, 5, 2, 3, 4,␣
 ↪0]}
numerico_objetivo_do_emprestimo = pd.DataFrame(numerico_objetivo_do_emprestimo)
```

Visto que a coluna grau_do_emprestimo se trata do Rating do cliente e é algo que é avaliado após a análise de risco. Também Não vi necessidade e não entendi o motivo da coluna padrao_historico eirei fazer a exclução e treinar o medelo e verificar o resultado.

[24]:
```
df.drop(columns=['grau_do_emprestimo'], inplace=True)

df.drop(columns=['padrao_historico'], inplace=True)
```

0.4 3. Em relação à base escolhida:

0.4.1 a. Você irá comparar alguns modelos para prever as classes. Descreva como a validação cruzada pode ser usada para comparar modelos de maneira justa. Descreva o procedimento e como a métrica final é calculada.

[25]:
```
# Seprando X e Y

y = df['inadimplente'].values
X = df.drop(columns=['inadimplente'])
```

[26]:
```
# Separação em conjuntos de treino e teste

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=38)
```

Usando validação cruzada para comparar modelos. Não sabemos de antemão quais modelos desempenharão bem neste conjunto de dados. Será usada a validação cruzada e serão avaliados diversos modelos usando a métrica de acurácia. Utilizaremos os modelos de K-vizinhos mais próximos (KNN), , Árvores de Classificação (DTC), *Naive Bayes* (NB), Máquinas de vetores de suporte (SVM) e Regressão Logística (LR).

Vamos comparar os resultados modelos criados, treinando-os com os dados do conjunto de treino e utilizando a técnica de validação cruzada. Para cada um dos modelos criados, executaremos a validação cruzada e, em seguida, exibiremos a acurácia média e o desvio padrão de cada um.

Descrevendo processo Validação Cruzada Na validação cruzada, o dataset é dividido aleatoriamente em "K" grupos. Quando definimos um número para "k", usamos o número no lugar de "k" para fazer referência ao teste (ex: 10-fold cross-validation).

No nossa caso usaremos k=10, o processo é feito da seguinte maneira:

Dividimos o dataset em 10 grupos: Fazemos uma iteração para cada grupo: O grupo é separado para teste, enquanto os demais são utilizados para treinamento, treinamos um modelo com os dados de treinamento, testamos com os dados de teste, salvamos o valor da métrica e descartamos o modelo. Depois o próximo grupo é selecionado.

O k-fold cross validation é uma boa maneira de avaliar como o modelo se comporta diante de variações das amostras de treinamento, e ajuda a evitar algunss problemas principais como o do hold-out: influência da divisão dos dados na métrica.

Testes empíricos em machine learning aplicada mostram que valores bastante confiáveis para K são 5 e 10, pois estes valores não causam erros de teste altos nem de bias e nem de variância.

[27]:
```
# Criação dos modelos

```

```
models = []
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('LR', LogisticRegression()))
```

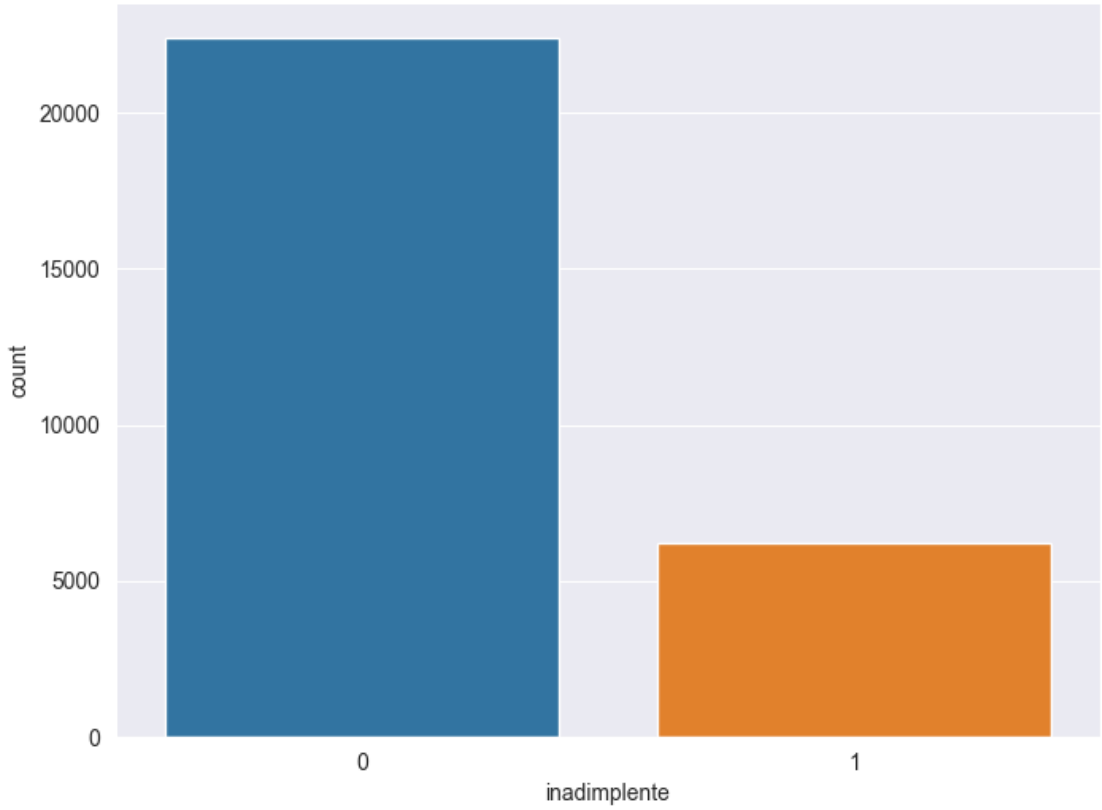
[28]:
```
# Avaliação dos modelos

results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10)
    cv_results = cross_val_score(model, X_train, y_train, cv=10,␣
 ↪scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
KNN: 0.829957 (0.006468)
CART: 0.874513 (0.006209)
NB: 0.820976 (0.002855)
SVM: 0.803363 (0.002917)
LR: 0.804760 (0.004875)
```

Estes resultados sugerem que a Árvores de Decisão (CART) , Naive Bayes (NB) e o K-vizinhos mais próximos (KNN) têm potencial de serem bons modelos, porém, vale observar que estes são apenas valores médios de acurácia, sendo prudente também observar outras métricas.

0.4.2 b. A base se encontra com as classes balanceadas? Cite uma maneira de resolver no caso das classes estarem desbalanceadas.

[29]:
```
# Inadimplentes vs adimplentes

inadim_adim = sns.countplot(x='inadimplente', data=df)
inadim_adim.figure.set_size_inches(8, 6)
```



Nossa base de dados encontra-se desbalaceadas, como podemos ver a propoção de cliente inadimplentes é bem menor que a dos clientes adimplentes e pode afeta nossa classificação, geralemente para esse tipo de problema de négocio acaba se tornando algo normal esse desbalanciamento.

Existe algumas técnicas que podem nos ajudar a ter melhores resultados. Vamos citar algumas maneiras que pode resolver o problemas.

O melhor método e que vai nos ajudar para tornar o nosso medelos mais eficiente é coletar mais dados da classe minoritaria, mas isso também leva mais tempo e muitas vezes coletar mais dados seria algo inviavel dependendo dos dados a serem coletados ou seria caro demais para coletar mais dados.

Também existem 3 técnicas bem comuns que são o Undersampling, Oversampling e SMOTE. Como elas funcionam?

Bom o Undersampling é um metodo bem simples, ele consiste em reduzir o número de dados da classe majoritária para diminuir a diferença entre as categorias. No Oversampling é justamente o contrario do Undersampling, ele consiste em criar de forma automatica novas observações da classe minoritária, assim os dados da classe minoritaria são igualados na mesma proporção. Já os SMOTE é mais sofistica, a ideia consiste em criar observações intermediárias entre dados parecidos, ou seja, se no dataset é como se ele tirasse a média entre as classes e preenchesse com um bem próxima, mas o detalhe que o SMOTE não seleciona necessariamente as médias entre as informações existente.

Algo que poderiamos fazer também é usarmos uma técnica chamada Cost-Sensitive Learning trata-

se de um aprendizado que lida com custo desiguais para realizar previsões. Nesse modelo o objetivo é buscar minimizar o erro, se considerados o custo da classificação correta ou incorreta, podemos otimizar o problema para minimizar o custo total da classificação incorreta onde podemos atribuir custos diferentes para uma classificação incorreta. Baseada na matriz de confusão, que resume as previsões corretas e incorretas para cada classe, o foco está em atribuir as penalidades para os falsos positivos e falsos negativos.

O custo total do classificador pode ser definido a partir da soma ponderada dos custos dos falsos negativos e falsos positivos. Esse será o valor a ser minimizado na modelagem. Os custos atribuídos são críticos para a modelagem, portanto devem ser feitos de forma cuidadosa e alinhada com a área de negócio.

Construir uma matriz de custos, não é simples. Pense em um problema de classificação de diagnóstico de uma doença grave. Qual o custo de um paciente doente não ser tratado? Em outro caso, qual o custo de um usuário detrator nas redes sociais? O cálculo pode se tornar muito complexo. Um bom ponto de partida ao lidar com a classificação de dados desbalanceados é usar a distribuição inversa do peso das classes.

O Cost-Sensitive Learning definem de forma explícita a penalização dos erros do modelo, a partir das regras de negócio. Existe três métodos no aprendizado Cost-Sensitive Learning

- Reamostragem: pode envolver a reamostragem dos dados de treinamento, ponderando pelo custo de cada classe. Outras abordagens são undersampling (remoção de exemplos da classe majoritária) e oversampling (duplicar ou sintetizar novos exemplos da classe minoritária).
- Algoritmo Sensível ao Custo: algoritmos existentes são alterados, passando a considerar a matriz de custos.
- Métodos Ensemble: combina modelos tradicionais que são modificados e passam a considerar a matriz de custo.

0.5 5. Com a base escolhida:

0.5.1 a. Descreva as etapas necessárias para criar um modelo de classificação eficiente.

Para a criação de um modelo de classificação eficiente precisamos realizar algumas etapas. E algumas dessas etapas mais importantes são:

Conseguir a base de dados, logíco que para treinar um modelo a primeira coisa que precisamos fazer é a coleta dos dados, nessa etapa coletamos os dados mais relevantes para nosso problema de négocio. É muito importante ter dados suficientes para que o nosso modelo possa ser treinando com precisão. Depois dos dados coletados precisamos fazer uma análise exploratória, aqui é onde endendemos melhos os dados, verificamos se existe dados faltantes, quais tipos dos dados, as categorias e identificamos possíveis padrões entre as variáveis. Depois disso seguimos para a preparação dos dados, aqui envolver a limpeza, tranformação e seleção das variáveis que serão usadas no nosso modelo, nessa etapa é onde também fazemos o tratamento dos dados faltantes por remove-los ou tratamos usando algumas técnicas possíves, codificamos as variáveis categóricas e normalizamos dados. Depois disso tudo estamos quase pronto para realizar nossa classificação.

Com os dados em mãos depois de análisar, explorar e fazer a preparação vamos dividir nosso conjunto de dados em conjuntos de treinamento e teste, nessa etapa devidimos os dados em conjuntos de treinamento e teste, enquanto uma parte é usada para treinar nosso modelo outra fica separada para testar nosso modelo, assim no modelo não fica comprometido. Em seguida vem uma etapa um

pouco mais díficil que é selecionar o algoritimo de classificação, existem vários e na hora de realizar uma classificação podemos ficar em dúvida, mas a escolha do algoritimo depende do nosso dados e problema de négocio. Isso muitas vezes pode envolver treinar vários modelos e fazer comparação entre eles para identificar qual vai resolver melhor nosso problema.

Pronto depois de escolher o algoritmo de classificação realizamos o treinamento aqui é hora de treinar o modelo. Isso envolve alimentar o conjunto de treinamento para o modelo e ajustar os parâmetros do modelo para que ele possa classificar com precisão os dados de entrada. É claro que ao terminar o treinamento precisamos saber se nosso modelo foi bom em sua performace então fazemos a Avaliação do modelo, é importante avaliar a precisão usando o conjunto de teste. Isso é feito com métricas como a precisão, recall, F1-score, matriz de confusão, entre outros.

Depois de que realizamos a avaliação podemos ideitificar que nosso modelo precise de alguns ajustes então fazemos a Melhoria do model é possível fazer melhorias, como selecionar outras variáveis ou ajustar os parâmetros do algoritmo de classificação.

E por fim realizamos a implementação do modelo, depois que um modelo é considerado eficiente, ele pode ser implantado em um ambiente de produção.

0.5.2 b. Treine um modelo de regressão logística para realizar a classificação. Qual a acurácia, a precisão, a recall e o f1-score do modelo? Treine um modelo de árvores de decisão para realizar a classificação. Qual a acurácia, a precisão, a recall e o f1-score do modelo?

Regression Logistic

[30]:
```
# Regression Logistic

logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)
y_pred = logistic_regression.predict(X_test)
```

[31]:
```
# Accuracy Score Regression Logistic

logis_regre_accuracy = metrics.accuracy_score(y_test, y_pred)
logis_regre_accuracy
```

[31]:
```
0.8142025611175786
```

[32]:
```
# Precision Regression Logistic

logis_regre_precision = metrics.precision_score(y_test, y_pred)
logis_regre_precision
```

[32]:
```
0.7637231503579952
```

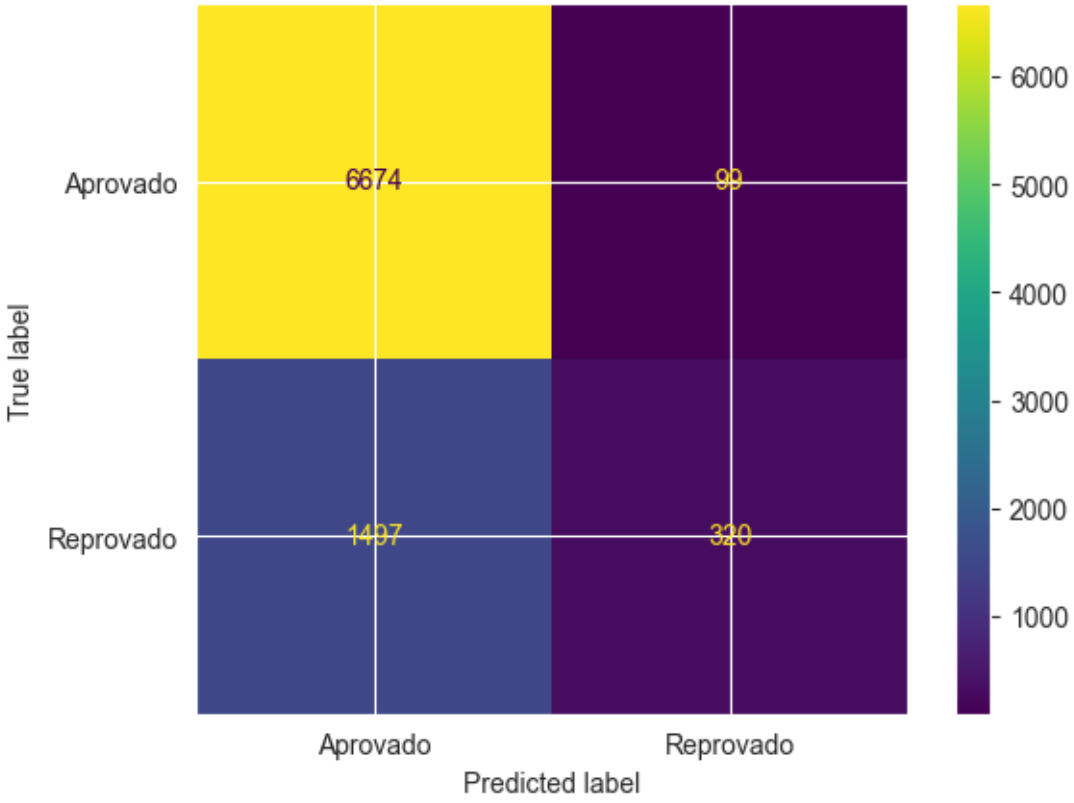
[33]:
```
# Recall Regression Logistic

logis_regre_recall = metrics.recall_score(y_test, y_pred)
logis_regre_recall
```

[33]: 0.17611447440836545

[34]:
```
# F1-Score Regression Logistic

logis_regre_f1 = metrics.f1_score(y_test, y_pred)
logis_regre_f1
```

[34]: 0.28622540250447226

[35]:
```
# Matrix Confusão Regression Logistic

cm_lr = metrics.confusion_matrix(y_test, y_pred)

labels = ['Aprovado', 'Reprovado']
px = metrics.ConfusionMatrixDisplay(cm_lr, display_labels=labels)
px.plot(values_format="d")
plt.show()
```



Decision Tree Classifier

[36]:
```
# Decision Tree Classifier
```

```
decision_tree_classifier = DecisionTreeClassifier()
decision_tree_classifier.fit(X_train, y_train)
y_pred = decision_tree_classifier.predict(X_test)
```

[37]:
```
# Accuracy Score Decision Tree Classifier

decision_tree_accuracy = metrics.accuracy_score(y_test, y_pred)
decision_tree_accuracy
```

[37]: 0.8823050058207218

[38]:
```
# Precision Decision Tree Classifier

decision_tree_precision = metrics.precision_score(y_test, y_pred)
decision_tree_precision
```

[38]: 0.7130021141649049

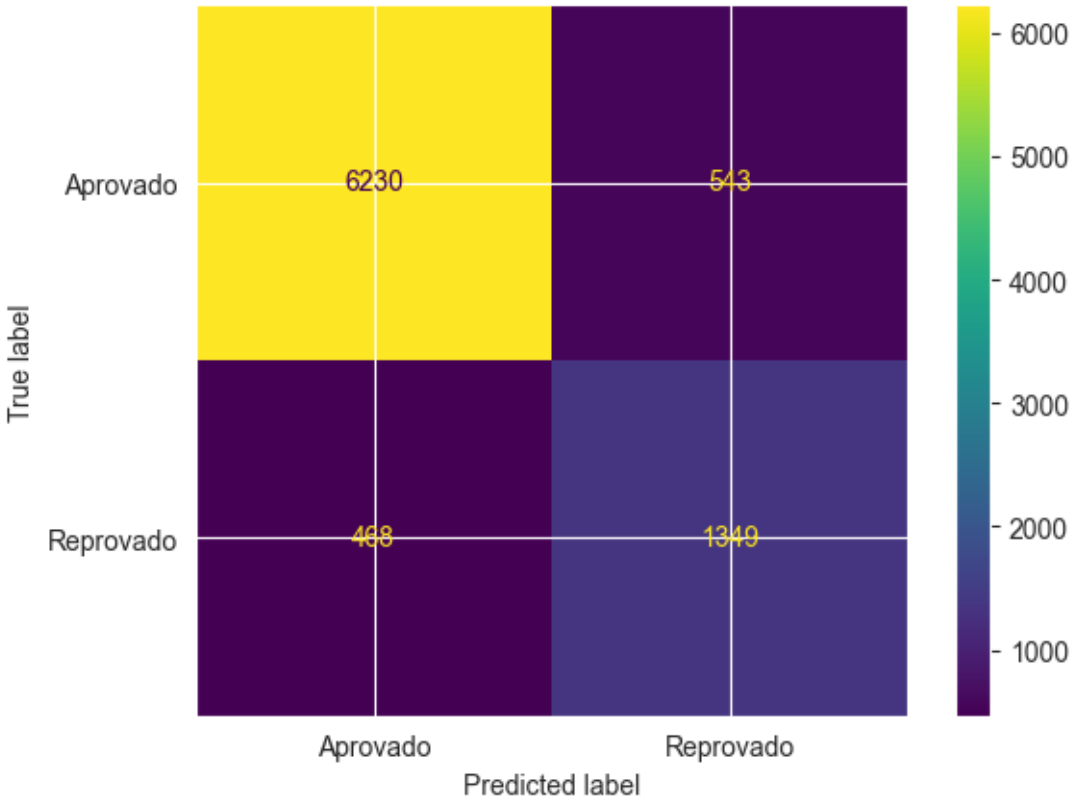
[39]:
```
# Recall Decision Tree Classifier

decision_tree_recall = metrics.recall_score(y_test, y_pred)
decision_tree_recall
```

[39]: 0.7424325811777656

[40]:
```
# F1-Score Decision Tree Classifier

decision_tree_f1 = metrics.f1_score(y_test, y_pred)
decision_tree_f1
```

[40]: 0.727419789700728

[41]:
```
# Matrix Confusão Decision Tree Classifier

cm_cart = metrics.confusion_matrix(y_test, y_pred)

labels = ['Aprovado', 'Reprovado']
px = metrics.ConfusionMatrixDisplay(cm_cart, display_labels=labels)
px.plot(values_format="d")
plt.show()
```



0.5.3 c. Treine um modelo de SVM para realizar a classificação. Qual a acurácia, a precisão, a recall e o f1-score do modelo?

SVM

[42]:
```
#SVM

svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
```

[43]:
```
# Accuracy SVM

svm_accuracy = metrics.accuracy_score(y_test, y_pred)
svm_accuracy
```

[43]: 0.8135040745052387

[44]:
```
# Precision SVM

svm_precision = metrics.precision_score(y_test, y_pred)
svm_precision
```

[44]: 0.8287461773700305

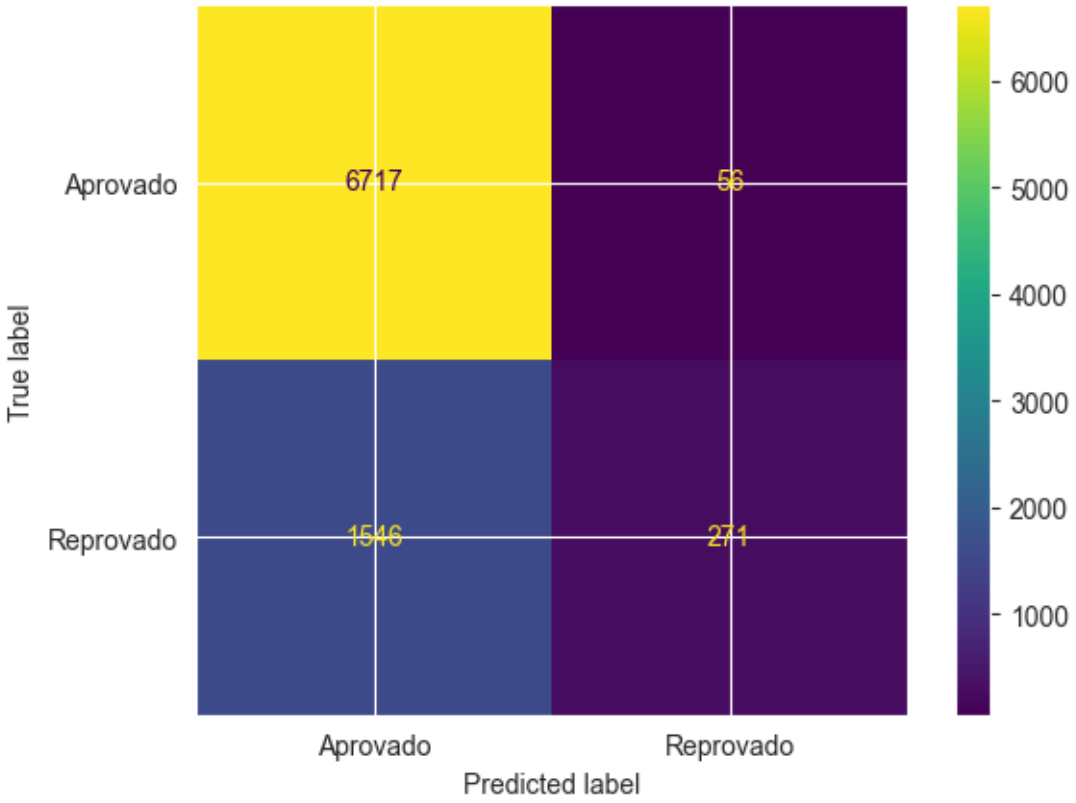
[45]:
```
# Recall SVM

svm_recall = metrics.recall_score(y_test, y_pred)
svm_recall
```

[45]: 0.1491469455145845

[46]:
```
# F1-Score SVM

svm_f1 = metrics.f1_score(y_test, y_pred)
svm_f1
```

[46]: 0.25279850746268656

[47]:
```
# Matrix Confusão SVM

cm_svm = metrics.confusion_matrix(y_test, y_pred)

labels = ['Aprovado', 'Reprovado']
px = metrics.ConfusionMatrixDisplay(cm_svm, display_labels=labels)
px.plot(values_format="d")
plt.show()
```



0.6 6. Em relação à questão anterior, qual o modelo deveria ser escolhido para uma eventual operação. Responda essa questão mostrando a comparação de todos os modelos e justifique

[48]:
```
# Criando Data Frame para comparar os modelos.

results_logis_regre = pd.DataFrame([[logis_regre_accuracy,␣
 ↪logis_regre_precision, logis_regre_recall, logis_regre_f1]],␣
 ↪index=['Logistic Regression'])
results_decision_tree = pd.DataFrame([[decision_tree_accuracy,␣
 ↪decision_tree_precision, decision_tree_recall, decision_tree_f1]],␣
 ↪index=['Decision Tree'])
results_svm = pd.DataFrame([[svm_accuracy, svm_precision, svm_recall, svm_f1]],␣
 ↪index=['SVM'])

results = pd.concat([results_logis_regre, results_decision_tree, results_svm],␣
 ↪axis=0)

results.rename(columns = {0 : 'Accuracy', 1 : 'Precission', 2: 'Recall', 3:␣
 ↪'F1'}, inplace = True)

results
```

[48]:

	Accuracy	Precission	Recall	F1
Logistic Regression	0.814203	0.763723	0.176114	0.286225
Decision Tree	0.882305	0.713002	0.742433	0.727420
SVM	0.813504	0.828746	0.149147	0.252799

Como podemos ver "Decission Tree" conseguiu uma Acuracia, Recall e F1 maior que os outros medelos, ja a "Árvore de Decisão" possui uma precisão maior que os damais. Dessa forma podemos dizer o modelos os modelos SVM e Regressão Logistica apensar de possuí uma boa acurracia o Recall consequentimente o F1 é inferior. Assim o melhor modelo escolhido pelo ver ser o Decision Tree (Árvore de Decisão).

Isso por que possui um Recall maior ou seja a possibilidade de acertar todos os verdadeiros positivos disponiveis é maior que os demais.

Por que Árvore de Decisão nosso problema de négocio seria melhor?

Vamos pensar um pouco, estamos treinado um modelo que seja capaz de decidir se um cliente ser Inadimplente ou Adimplente, então o que será melhor para a concessora do crédito? Ter uma acuracia maior e consequentimente um Recall mais visto que isso indicar que a possibilidade de falso negativos são bem menores que os demais modelos, e podemos obeservar isso na matrix confussão plotada acima.