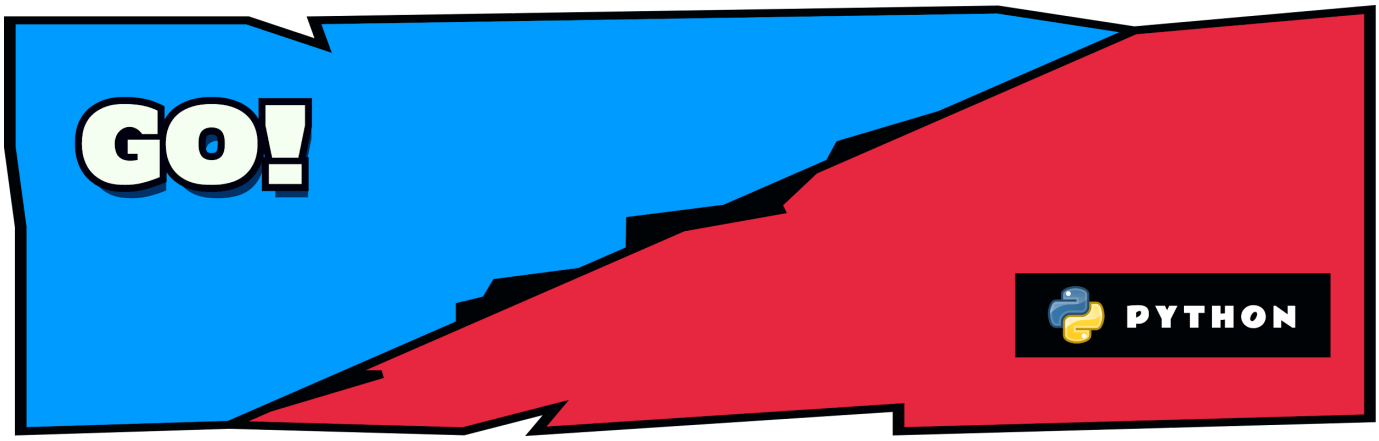


CRISP - DM | Data Science: Cross Industry Standard Process for Data Mining

Caderno de **Códigos**
Desenvolvedor Enzo Schitini

Etapa 1 CRISP - DM: Entendimento do negócio

Como primeira etapa do CRISP-DM, vamos entender do que se trata o negócio, e quais os objetivos.

Este é um problema de concessão de cartões de crédito, publicado no Kaggle, uma plataforma que promove desafios de ciência de dados, oferecendo prêmios em dinheiro para os melhores colocados. O link original está aqui.

Essa é uma base de proponentes de cartão de crédito, nosso objetivo é construir um modelo preditivo para identificar o risco de inadimplência (tipicamente definida pela ocorrência de um atraso maior ou igual a 90 em um horizonte de 12 meses) através de variáveis que podem ser observadas na data da avaliação do crédito (tipicamente quando o cliente solicita o cartão).

Atividades do CRISP-DM:

- Objetivos do negócio Note que o objetivo aqui é que o modelo sirva o mutuário (o cliente) para que avalie suas próprias decisões, e não a instituição de crédito.
- Objetivos da modelagem O objetivo está bem definido: desenvolver o melhor modelo preditivo de modo a auxiliar o mutuário a tomar suas próprias decisões referentes a crédito.

Nessa etapa também se avalia a situação da empresa/segmento/assunto de modo a se entender o tamanho do público, relevância, problemas presentes e todos os detalhes do processo gerador do fenômeno em questão, e portanto dos dados.

Também é nessa etapa que se constrói um planejamento do projeto.

Etapa 2 Crisp-DM: Entendimento dos dados

A segunda etapa é o entendimento dos dados. Foram fornecidas 15 variáveis mais a variável resposta (em negrito na tabela). O significado de cada uma dessas variáveis se encontra na tabela.

Dicionário de dados

Os dados estão dispostos em uma tabela com uma linha para cada cliente, e uma coluna para cada variável armazenando as características desses clientes. Colocamos uma cópia o dicionário de dados (explicação dessas variáveis) abaixo neste notebook:

Variable Name	Description	Tipo
sexo	M = 'Masculino'; F = 'Feminino'	M/F
posse_de_veiculo	Y = 'possui'; N = 'não possui'	Y/N
posse_de_imovel	Y = 'possui'; N = 'não possui'	Y/N
qtd_filhos	Quantidade de filhos	inteiro
tipo_renda	Tipo de renda (ex: assaliariado, autônomo etc)	texto
educacao	Nível de educação (ex: secundário, superior etc)	texto
estado_civil	Estado civil (ex: solteiro, casado etc)	texto
tipo_residencia	tipo de residência (ex: casa/apartamento, com os pais etc)	texto
idade	idade em anos	inteiro
tempo de emprego	tempo de emprego em anos	inteiro
possui_celular	Indica se possui celular (1 = sim, 0 = não)	binária
possui_fone_comercial	Indica se possui telefone comercial (1 = sim, 0 = não)	binária
possui_fone	Indica se possui telefone (1 = sim, 0 = não)	binária
possui_email	Indica se possui e-mail (1 = sim, 0 = não)	binária
qt_pessoas_residencia	quantidade de pessoas na residência	inteiro
mau	indicadora de mau pagador (True = mau, False = bom)	binária

Carregando os pacotes

É considerado uma boa prática carregar os pacotes que serão utilizados como a primeira coisa do programa.

```
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
```

Carregando os dados

O comando pd.read_csv é um comando da biblioteca pandas (pd.) e carrega os dados do arquivo csv indicado para um objeto *dataframe* do pandas.

```
# Observe que demo01.csv está na mesma pasta que este notebook
# do contrário, seria necessário indicar a pasta no nome do arquivo
df = pd.read_csv('demo01.csv')
print ("Número de linhas e colunas da tabela: {}".format(df.shape))

df.head()
```

```
Número de linhas e colunas da tabela: (70126, 16)
```

	sexo	posse_de_veiculo	posse_de_imovel	qtd_filhos	tipo_renda	educacao	estado
0	M	Y	Y	0	Working	Secondary / secondary special	I
1	F	N	Y	0	Commercial associate	Secondary / secondary special	Sing r
2	F	N	Y	0	Commercial associate	Secondary / secondary special	Sing r

Entendimento dos dados - Univariada

Nesta etapa tipicamente avaliamos a distribuição de todas as variáveis. Nesta demonstração vamos ver a variável resposta e dois exemplos de univariada apenas. Mas sinta-se à vontade para tentar observar outras variáveis.

```
grafico_barras = df['mau'].value_counts().plot.bar()

print(df['mau'].value_counts())
print("\nTaxa de inadimplentes:")
print(df['mau'].mean())
```

```
False    65040
True      5086
Name: mau, dtype: int64

Taxa de inadimplentes:
0.07252659498616776
```

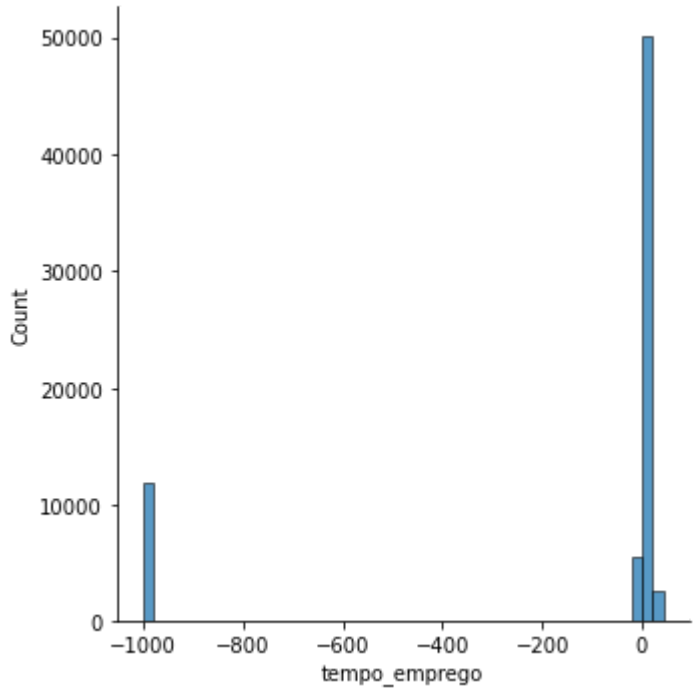
Tarefa

Construa um gráfico de barras para pelo menos duas outras variáveis. **Dica:** Não tente usar as variáveis `tempo_emprego` e `idade` pois o gráfico de barras dessa forma como construímos não é adequado para elas.


```
plt.clf()
var = "tempo_emprego"

sns.displot(df, x = var, bins = 50)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```

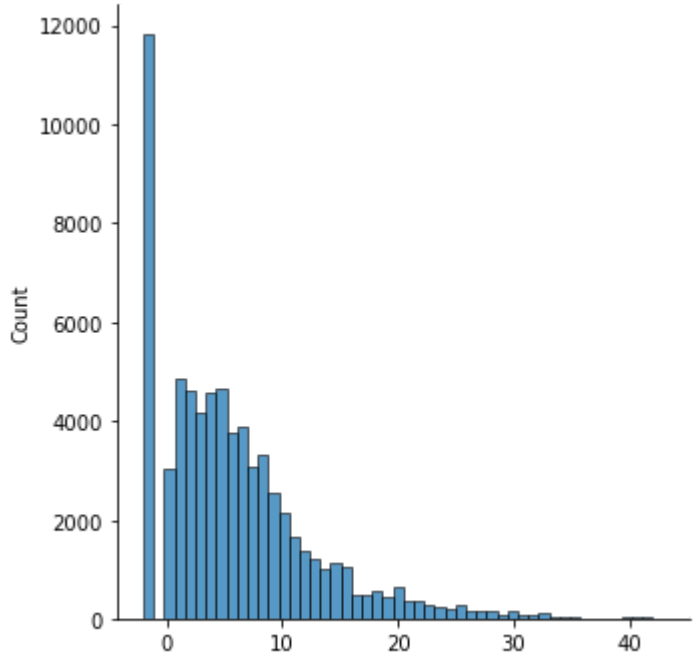



```
plt.clf()
var = "tempo_emprego"

df.loc[df[var]<0,var] = -2

sns.displot(df, x = var, bins = 50)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Tarefa

Construa o histograma da variavel `dade` .

Entendimento dos dados - Bivariadas

Entender a alteração da inadimplência indicada pela variável resposta (`AtrasoRelevante2anos`) e as variáveis explicativas (demais). Para isto, vamos calcular a taxa de inadimplentes (qtd inadimplentes / total) para diferentes grupos definidos pelas variáveis explicativas.

```
var = 'idade'
cat_srs, bins = pd.qcut(df[var], 4, retbins=True)
g = df.groupby(cat_srs)
biv = g['mau'].mean()

ax = biv.plot.line()
ax.set_ylabel("Proporção de inadimplentes")
ticks = plt.xticks(range(len(biv.index.values)), biv.index.values, rotation = 90)
```

Etapa 3 Crisp-DM: Preparação dos dados

Nessa etapa realizamos tipicamente as seguintes operações com os dados:

- seleção Neste caso, os dados já estão pré-selecionados
- limpeza Precisaremos identificar e tratar dados faltantes
- construção Neste primeiro exercício não faremos construção de novas variáveis
- integração Temos apenas uma fonte de dados, não é necessário agregação
- formatação Os dados já se encontram em formatos úteis

Os dados já estão pré-selecionados, construídos e integrados, mas há dados faltantes que serão eliminados na próxima célula

```
metadata = pd.DataFrame(df.dtypes, columns = ['tipo'])

metadata['n_categorias'] = 0

for var in metadata.index:
    metadata.loc[var,'n_categorias'] = len(df.groupby([var]).size())

metadata
```

	tipo	n_categorias
sexo	object	2
posse_de_veiculo	object	2

```
def convert_dummy(df, feature,rank=0):
    pos = pd.get_dummies(df[feature], prefix=feature)
    mode = df[feature].value_counts().index[rank]
    biggest = feature + '_' + str(mode)
    pos.drop([biggest],axis=1,inplace=True)
    df.drop([feature],axis=1,inplace=True)
    df=df.join(pos)
    return df
```

tipo_residencia	object	6

```
for var in metadata[metadata['tipo'] == 'object'].index:
    df = convert_dummy(df, var)
```

tempo_emprego	float64	3015

```
df
```

	qtd_filhos	idade	tempo_emprego	possui_celular	possui_fone_comercial	p
0	0	58.832877	3.106849	1	0	
1	0	52.356164	8.358904	1	0	
2	0	52.356164	8.358904	1	0	
3	0	46.224658	2.106849	1	1	
4	0	29.230137	3.021918	1	0	
...	...	...	...	...	...	
70121	0	54.109589	9.884932	1	0	
70122	0	43.389041	7.380822	1	1	
70123	0	30.005479	9.800000	1	1	
70124	0	30.005479	9.800000	1	1	
70125	0	33.936986	3.630137	1	0	

70126 rows × 29 columns

Etapa 4 Crisp-DM: Modelagem

Nessa etapa que realizaremos a construção do modelo. Os passos típicos são:

- Selecionar a técnica de modelagem Utilizaremos a técnica de floresta aleatória (**random forest**), pois é uma técnica bastante versátil e robusta que captura bem padrões complexos nos dados, relativamente fácil de se usar e que costuma produzir excelentes

resultados para uma classificação como estas. Vamos ver esse algoritmo em detalhes mais adiante no curso, mas pense nele por enquanto como uma regra complexa baseada nas variáveis explicativas que classifica o indivíduo como inadimplente ou não. Mais adiante no curso vamos extrair mais dessa técnica.

- Desenho do teste Antes de rodar o modelo precisamos construir um desenho do teste que será realizado. Para desenvolver um modelo como este, é considerado uma boa prática dividir a base em duas, uma chamada `treinamento`, onde o algoritmo 'aprende', e outra chamada `teste`, onde o algoritmo é avaliado. Essa prática fornece uma métrica de avaliação mais fidedigna do algoritmo, falaremos mais detalhes em lições futuras.
- Avaliação do modelo Faremos a avaliação do nosso modelo através do percentual de acerto, avaliando a classificação do modelo (inadimplente e não inadimplente) e comparando com o estado real armazenado na variável resposta (`AtrasoRelevante2anos`). Esse percentual de acerto é frequentemente chamado de acurácia (**obs:** nunca usar assertividade... a**ss**ertivo não é aquele que a**c**erta, e sim "*adj.: em que o locutor declara algo, positivo ou negativo, do qual assume inteiramente a validade; declarativo*." a**C**ertivo está errado ;)

Dividindo a base em treino e teste

```
# Tirando a v. resposta da base de treinamento
x = df.drop("mau",axis = 1)
y = df["mau"]

# Tirando ID da base de treinamento e teste
x_train, x_test, y_train, y_test = train_test_split(x, y)


x_train
```

	qtd_filhos	idade	tempo_emprego	possui_celular	possui_fone_comercial	p
8878	0	49.249315	16.726027	1	0	

Rodando o modelo

A função RandomForestClassifier gera a estrutura da floresta aleatória, e o parâmetro `n_estimator` define o número de árvores na floresta. Normalmente a acurácia do modelo tende a aumentar com o número de árvores, pelo menos até um certo limite - e aumenta também o recurso computacional demandado. Você pode alterar esse parâmetro e verificar se a acurácia do seu modelo melhora - não recomendamos valores muito altos. Vá alterando aos poucos e percebendo como o tempo aumenta com os seus recursos. Não é necessário ir muito além de umas 100 árvores.

50582	2	38.298630	4.454795	1	0	

```
# Treinar uma Random Forest com 5 árvores

clf = RandomForestClassifier(n_estimators=2)
clf.fit(x_train,y_train)
```

```
RandomForestClassifier(n_estimators=2)
```

```
# Calculando a acuracia

y_pred = clf.predict(x_test)
acc = metrics.accuracy_score(y_test, y_pred)
print('Acurácia: {0:.2f}%'.format(acc*100))
```

```
Acurácia: 97.67%
```

```
# Matriz de confusão

tab = pd.crosstab(index = y_pred, columns = y_test)
print(tab[1][0]/(tab[1][0] + tab[0][0]))
print(tab[1][1]/(tab[1][1] + tab[0][1]))
tab
```

```
0.010180786080088851
0.8166037735849057
```

mau	False	True
row_0		
False	16042	165
True	243	1082

Etapa 5 Crisp-DM: Avaliação dos resultados

A etapa final do CRISP. Neste casp, a nossa avaliação termina com a acurácia. Mas em problemas futuros aprofundaremos mais - a ideia seria avaliar o impacto do uso do modelo no negócio, ou seja, o quanto o resultado financeiro melhora em detrimento da utilização do modelo.

Como um exemplo simples, considere que um cliente bom pagador deixa (em média) 5 '*dinheiros*' de lucro, e um mau pagador deixa (em média) 100 '*dinheiros*' de prejuízo.

de acordo com a matriz de confusão:

Decisão	lucro dos bons	lucro dos maus	total
Aprovador	4042 x 5	72 x (-100)	13.010
Reprovar	27 x 5	22 x (-100)	-2.065

Estariamos evitando, portanto, um prejuízo de -2.145 '*dinheiros*' - o que na prática significa um aumento no lucro.

Etapa 6 Crisp-DM: Implantação

Nessa etapa colocamos em uso o modelo desenvolvido, normalmente implementando o modelo desenvolvido em um motor de crédito que toma as decisões com algum nível de automação - tipicamente aprovando automaticamente clientes muito bons, negando automaticamente clientes muito ruins, e enviando os intermediários para análise manual.