01

Trabalho 1 de AMMCI

Gabriel Colli Pavan *RA: 109882*

Matheus Augusto Schiavon Parise *RA: 107115*

02 Introdução

Os bancos incorrem em prejuízos quando os clientes não pagam seus empréstimos em dia. Por causa disso, todos os anos, os bancos têm prejuízos de milhões, e isso também afeta em grande medida o crescimento econômico do país. Foi escolhida uma base de dados de previsão de empréstimo que contém 67 mil registros fictícios, esses registros são os dados dos supostos clientes que pediram empréstimo. O objetivo do trabalho é determinar se o banco deverá conceder empréstimo de acordo com os dados do cliente. A base de dados utilizada neste trabalho pode ser obtida em: *https://www.kaggle.com/datasets/hemanthsai7/loandefault*

03.

Variáveis

Foram utilizadas para o desenvolvimento desse trabalho 4 variáveis:

Loan Amount: Valor do empréstimo" em dólares;

Valor já pago da dívida: Resultado da divisão entre o valor pago da dívida (*Funded Amount*) pelovalor total emprestado (*Loan Amount*), considerando que cada dívida possui um valor diferente, consideramos portanto a % da d´ivida paga.

Debit to Income: a % da dívida total que será paga por mês.

Interest Rate: Juros calculado em cima do valor emprestado.

Modelagem

Valor do empréstimo (em dólares)

- empréstimo <= 10.000
- 10.000 < empréstimo <= 25.000
- 25.000 < empréstimo

Valor já pago da dívida (% da dívida paga)

- valor pago < 50%
- > 50%

Débito que está para vir (valor em % pago por mês):

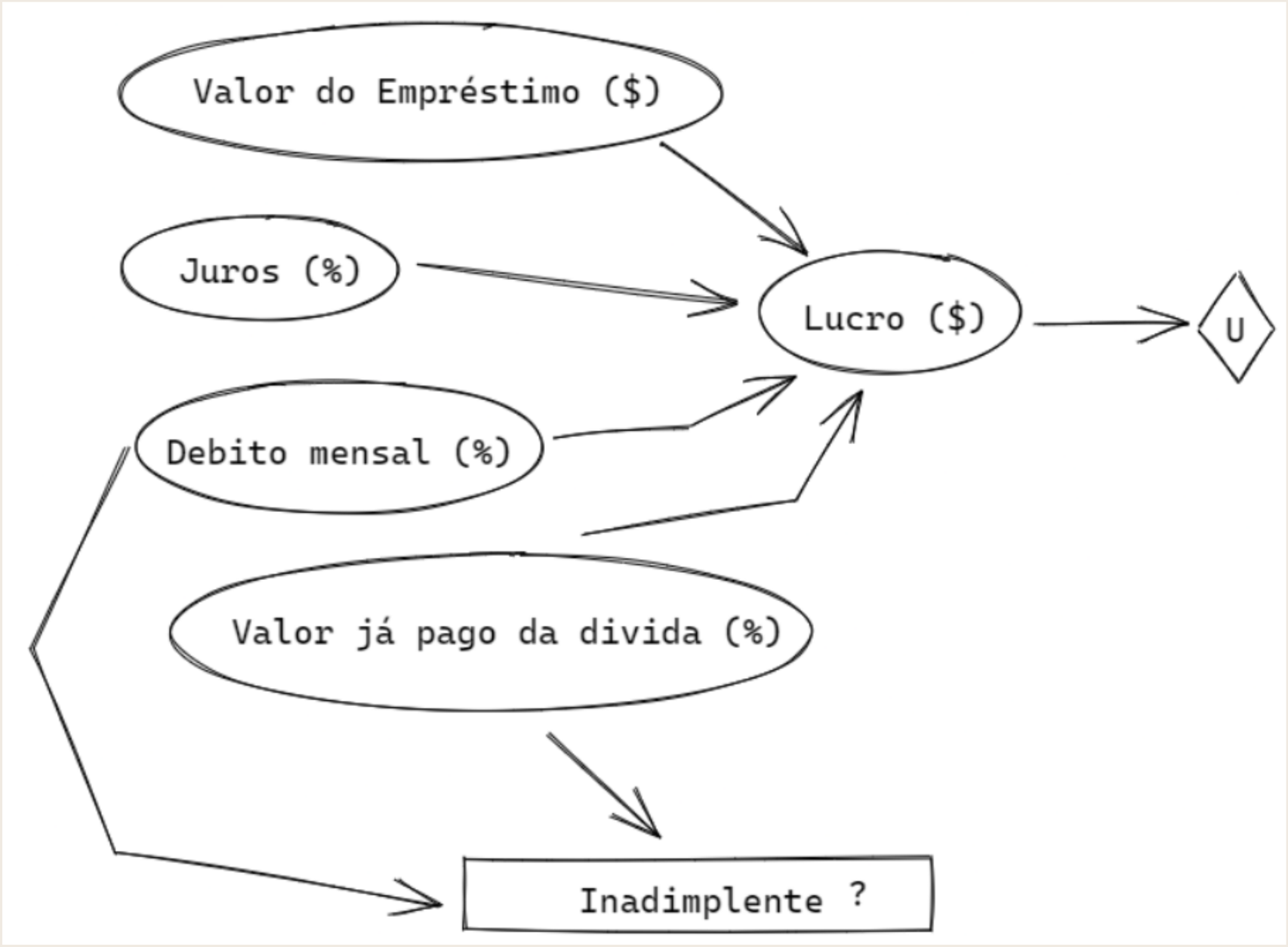
- débito <= 15%
- 15% < débito <= 30%
- débito > 30%

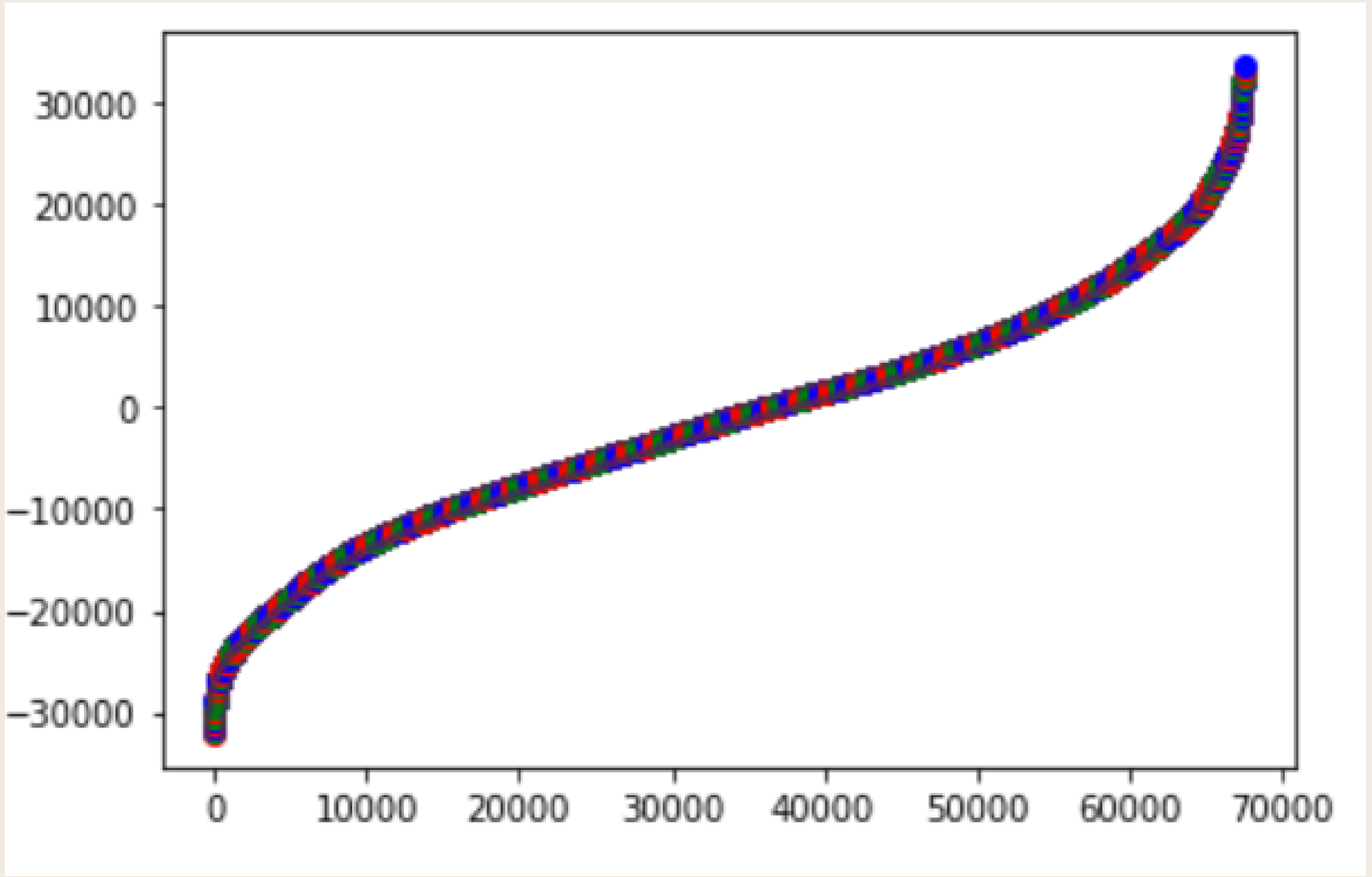
Taxa de Juros (% do valor total emprestado que é cobrado mensalmente)

- Juros <= 8%
- 8% < Juros <= 15%
- Juros > 15%

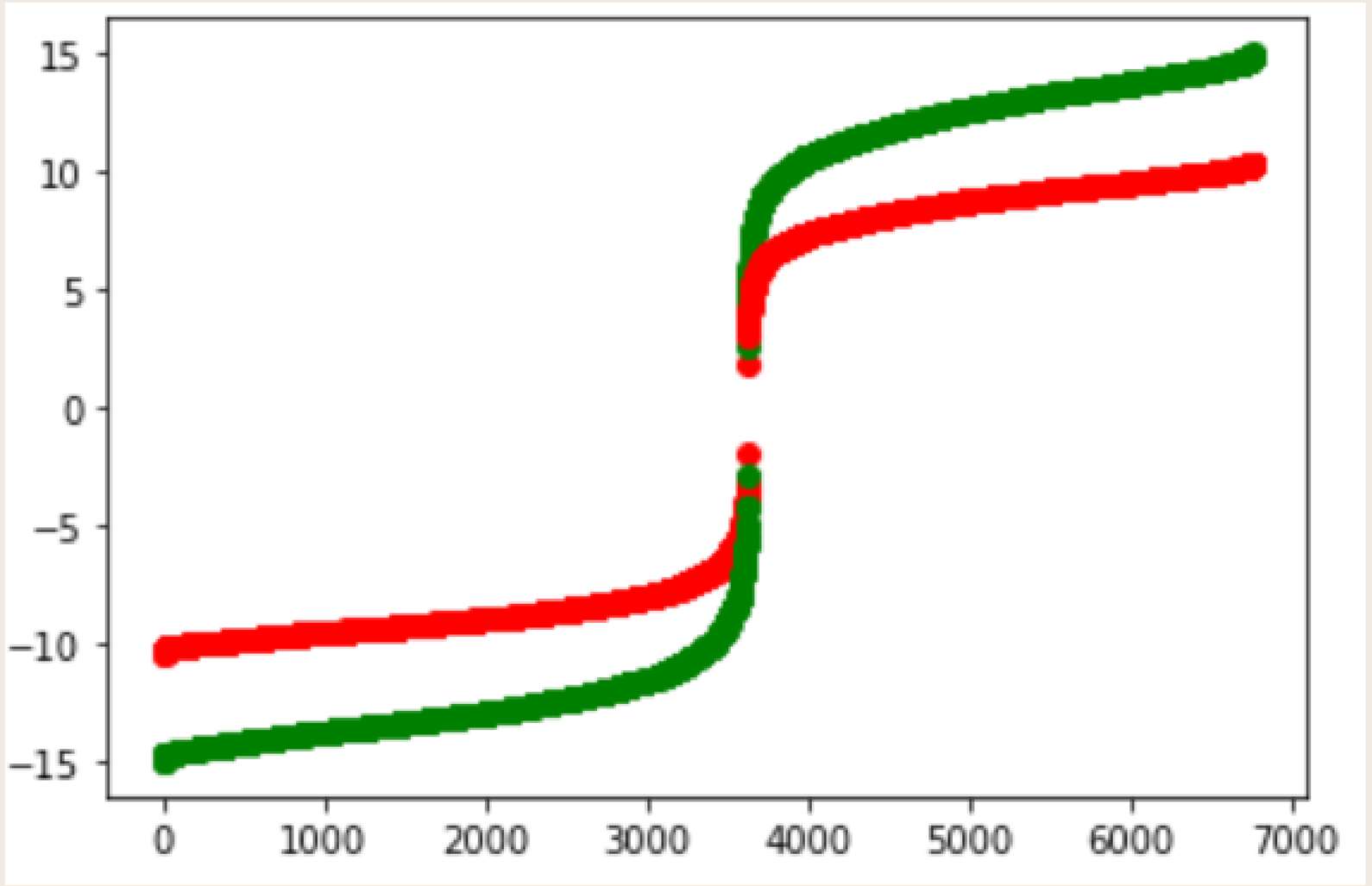
Modelo

O programa decide a porcentagem de chance dele ser ou não inadimplente e também sua utilidade. A utilidade é considerada baixa caso não tenha pago ou tenha pago muito pouco o valor investido, média utilidade se pagou uma porção próxima ao valor emprestado e alta se pagou maior que o valor emprestado.


Valor do Empréstimo ($)
Juros (%)
Lucro ($)
U
Debito mensal (%)
Valor já pago da divida (%)
Inadimplente ?



A utilidade é determinada pelo valor financeiro faturado pelo banco em relação a cada cliente.



A utilidade é determinada pelo valor financeiro faturado pelo banco em relação a cada cliente.

09

Código

```
import pandas as pd # Para manipulação de dados
import networkx as nx # Para desenhar gráficos
import matplotlib.pyplot as plt # Para desenhar gráficos
import matplotlib.cm as cm # Para as cores do plot
import itertools # Iteração entre cores # Talvez a retirar
import math # Para calculos matematicos

!pip install pybbn

# para a criação de Redes Bayesianas de Crenças (Bayesian Belief Networks (BBN))
import pybbn
from pybbn.graph.potential import Potential
from pybbn.graph.dag import Bbn
from pybbn.graph.edge import Edge, EdgeType
from pybbn.graph.jointree import EvidenceBuilder
from pybbn.graph.node import BbnNode
from pybbn.graph.variable import Variable
from pybbn.pptc.inferencecontroller import InferenceController
```

Código

```
# Definindo as opções do Pandas para exibir mais colunas
pd.options.display.max_columns=50

# Ler as informações do historico de empréstimo
df=pd.read_csv('train.csv', encoding='utf-8')

# Remover registros onde a variavel de decisão final, "Loan Status" está vazia
df=df[pd.isnull(df['Loan Status'])==False]

# Preencher as colunas com valores ausentes com a média da coluna
df=df.fillna(df.mean())

# Primeiramente vamos exibir o data frame dos clientes inadimplentes
dfi = df[df['Loan Status'] == 1] # data frame cliente inadimplentes
dfi
```

	ID	Loan Amount	Funded Amount	Funded Amount Investor	Term	Batch Enrolled	Interest Rate	Grade	Sub Grade
15	9813088	27859	33502	16545.203070	36	BAT2833642	14.848190	B	A4
19	9038338	14058	26523	9622.342364	59	BAT2003848	14.523373	A	C1
25	6688651	15330	21812	12374.390930	59	BAT4271519	8.971090	A	A5

Código

```
# É possivel observar algumas informações interessantes para usar como variaveis, como valor do emprestimo,
# porcentagem paga do emprestimo, quantidade que será paga por mês, taxa de juros, entre outros.

# Valores Medios dos casos Inadimplentes
Media_Valor_Emprestimo_Inadiplente = dfi['Loan Amount'].mean() # Valor de emprestimo do inadimplente (Dolar)
Media_Valor_Pago_Inadiplente = (dfi['Funded Amount']/dfi['Loan Amount']).mean() # Valor que já foi Pago / Valor emprestado = Valor Pago Total (%)
Media_QtdPagaNoMes_Inadiplente = dfi['Debit to Income'].mean() # Porcentagem do emprestimo paga ao mês (%)
Media_Juros_Inadiplente = dfi['Interest Rate'].mean() # Taxa de juros (% do valor total do emprestimo)

print("Valores do clientes inadimplentes:\n"
    + "Média dos valores de emprestimo: " + str(Media_Valor_Emprestimo_Inadiplente) + "\n"
    + "Média valor pago total (%): " + str(Media_Valor_Pago_Inadiplente) + "\n"
    + "Média da porcentagem do emprestimo paga ao mês (%): " + str(Media_QtdPagaNoMes_Inadiplente) + "\n"
    + "Média da taxa de juros (%): " + str(Media_Juros_Inadiplente) + "\n")

# Média de quem não é inadimplentes
dfni = df[df['Loan Status'] == 0] # data frame não inadimplente

Media_Valor_Emprestimo = dfni['Loan Amount'].mean() # Valor de emprestimo do inadimplente (Dolar)
Media_Valor_Pago = (dfni['Funded Amount']/dfni['Loan Amount']).mean() # Valor que já foi Pago / Valor emprestado = Valor Pago Total (%)
Media_QtdPagaNoMes = dfni['Debit to Income'].mean() # Porcentagem do emprestimo paga ao mês (%)
Media_Juros = dfni['Interest Rate'].mean() # Taxa de juros (% do valor total do emprestimo)

print("Valores do clientes NÃO inadimplentes:\n"
    + "Média dos valores de emprestimo: " + str(Media_Valor_Emprestimo) + "\n"
    + "Média valor pago total (%): " + str(Media_Valor_Pago) + "\n"
    + "Média da porcentagem do emprestimo paga ao mês (%): " + str(Media_QtdPagaNoMes) + "\n"
    + "Média da taxa de juros (%): " + str(Media_Juros) + "\n")

# % do valor emprestada paga por não inadimplente vs. inadimplente
[Media_Valor_Pago, Media_Valor_Pago_Inadiplente]
```

Código

```
Valores do clientes inadimplentes:
Média dos valores de emprestimo: 16731.67441115206
Média valor pago total (%): 1.4016942608785445
Média da porcentagem do emprestimo paga ao mês (%): 23.2183095565415 84
Média da taxa de juros (%): 11.880031823501682

Valores do clientes NÃO inadimplentes:
Média dos valores de emprestimo: 16860.853092025744
Média valor pago total (%): 1.3679365495468265
Média da porcentagem do emprestimo paga ao mês (%): 23.307490770173725
Média da taxa de juros (%): 11.842814966824166

[1.3679365495468265, 1.4016942608785445]
```

13.

Código

```
# Independente de conseguir pagar ou não o banco vai obter lucro pois em média seja o cliente inadimplente ou não!!!
# A função de utilidade será feita  de acordo com o lucro obtido por parte do banco em relação a determinado cliente até o momento presente.
# Agora Vamos escolher as variaveis que serão usadas no modelo e definir as categorias para elas de acordo com as medias obtidas dos casos Inadimplentes

def setar_variaveis(df):
  # Valor do emprestimo
  df['Loan Amount Cat']=df['Loan Amount'].apply(lambda x: '0. <=10000'  if x<=10000 else
                                                          '1. 10000-25000' if 10000<x<=25000 else
                                                          '2. >25000')


  # Porcentagem paga do emprestimo
  df['Porcentagem paga'] = df['Funded Amount']/df['Loan Amount']
  df['Funded Amount Cat']=df['Porcentagem paga'].apply(lambda x: '0. <=0.5' if x <= 0.5 else
                                                                 '1. >0.5') #if 0.5 < x <= 1.5 else
                                                                 #'2. >1.5')


  # Quantidade que será paga por mês
  df['Debit to Income Cat']=df['Debit to Income'].apply(lambda x: '0. <=15' if x<=15 else
                                                                  '1. 15-30' if 15<x<=30 else
                                                                  '2. >30')


  # Taxa de juros
  df['Interest Rate Cat']=df['Interest Rate'].apply(lambda x: '0. <=8' if x<=8 else
                                                              '1. 8-15' if 7<x<=15 else
                                                              '2. >15')


setar_variaveis(df)
# Mostrando Dados
df
```

Código

Loan Status	Loan Amount Cat	Porcentagem paga	Funded Amount Cat	Debit to Income Cat	Interest Rate Cat
0	0. <=10000	3.223600	1. >0.5	1. 15-30	1. 8-15
0	0. <=10000	3.308396	1. >0.5	1. 15-30	1. 8-15

15.

Código

```
# Porcentagem de clientes que estão Inadimplente (=~ 10%) e porcentagem dos não Inadimplente (=~ 90%)
LS = df['Loan Status'].value_counts(normalize=True).sort_index()
LS
```

```
0    0.90749
1    0.09251
Name: Loan Status, dtype: float64
```

```
# As probabilidades aqui são frequências normalizadas das categorias de variáveis dos dados.
# Valor do emprestimo
LAC = df['Loan Amount Cat'].value_counts(normalize=True).sort_index()
LAC
```

```
0. <=10000         0.249500
1. 10000-25000     0.559373
2. >25000          0.191127
Name: Loan Amount Cat, dtype: float64
```

```
# Porcentagem paga do emprestimo
FAC = df['Funded Amount Cat'].value_counts(normalize=True).sort_index()
FAC
```

```
0. <=0.5     0.223426
1. >0.5      0.776574
Name: Funded Amount Cat, dtype: float64
```

Código

```
# Quantidade que será paga por mês
DIC = df['Debit to Income Cat'].value_counts(normalize=True).sort_index()
DIC
```

```
0. <=15     0.179743
1. 15-30    0.568919
2. >30      0.251338
Name: Debit to Income Cat, dtype: float64
```

```
# Taxa de juros
IRC = df['Interest Rate Cat'].value_counts(normalize=True).sort_index()
IRC
```

```
0. <=8      0.145235
1. 8-15     0.656197
2. >15      0.198568
Name: Interest Rate Cat, dtype: float64
```

17.

Código

```python
# Esta função ajuda a calcular a distribuição de probabilidade, que vai para BBN (nota, pode lidar com até 4 pais)
# Há uma certa desconfiança por trás do número de possibilidades gerado por crosstab então vamos averiguar logo em seguida
# se o número bate.
def probs(data, child, parent1=None, parent2=None, parent3=None, parent4=None):
    if parent1==None:
        # Calcular probabilidades
        prob=pd.crosstab(data[child], 'Empty', margins=False, normalize='columns').sort_index().to_numpy().reshape(-1).tolist()
    elif parent1!=None:
            # Verifique se o nó filho tem 1 pai ou 2 pais
            if parent2==None:
                # Calcular probabilidades
                prob=pd.crosstab(data[parent1],data[child], margins=False, normalize='index').sort_index().to_numpy().reshape(-1).tolist()
            else:
                # Verifique se o nó filho tem 2 pai ou 3 pais
                if parent3==None:
                    # Caclucate probabilities
                    prob=pd.crosstab([data[parent1],data[parent2]],data[child], margins=False, normalize='index').sort_index().to_numpy().reshape(-1).tolist()
                else:
                    #Verifique se o nó filho tem 3 pai ou 4 pais
                    if parent4==None:
                        # Calcular probabilidades
                        prob=pd.crosstab([data[parent1],data[parent2],data[parent3]],data[child], margins=False, normalize='index').sort_index().to_numpy().reshape(-1).tolist()
                    else:
                        # Calcular probabilidades
                        prob=pd.crosstab([data[parent1],data[parent2],data[parent3],data[parent4]],data[child], margins=False, normalize='index').sort_index().to_numpy().reshape(-1).tolist()
    else: print("Erro nos cálculos de frequência de probabilidade")
    return prob
```

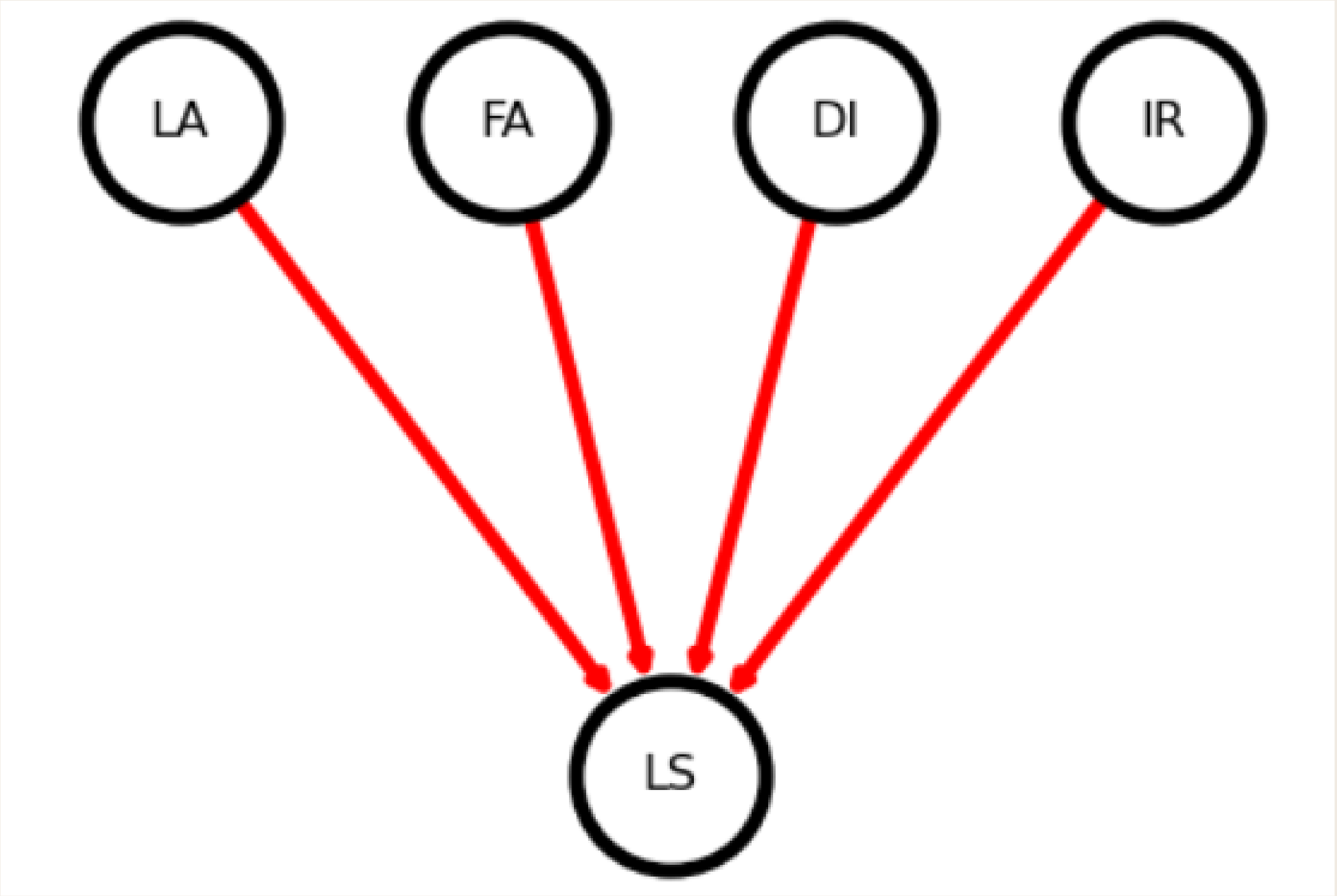
Código

```
def criar_rede(df, df2):
  # Crie nós usando nossa função anterior para calcular probabilidades automaticamente
  LA = BbnNode(Variable(0, 'LA', ['0. <=10000', '1. 10000-25000', '2. >25000']), probs(df, child='Loan Amount Cat'))
  FA = BbnNode(Variable(1, 'FA', ['0. <=0.5', '1. >0.5']), probs(df, child='Funded Amount Cat'))
  DI = BbnNode(Variable(2, 'DI', ['0. <=15', '1. 15-30', '2. >30']), probs(df, child='Debit to Income Cat'))
  IR = BbnNode(Variable(3, 'IR', ['0. <=8', '1. 8-15', '2. >15']), probs(df, child='Interest Rate Cat'))
  LS = BbnNode(Variable(4, 'LS', ['No', 'Yes']), probs(df2, child='Loan Status', parent1='Loan Amount Cat', parent2='Funded Amount Cat', parent3='Debit to Income Cat', parent4='Interest Rate Cat'))

  # Criar rede
  bbn = Bbn() \
      .add_node(LA) \
      .add_node(FA) \
      .add_node(DI) \
      .add_node(IR) \
      .add_node(LS) \
      .add_edge(Edge(LA, LS, EdgeType.DIRECTED)) \
      .add_edge(Edge(FA, LS, EdgeType.DIRECTED)) \
      .add_edge(Edge(DI, LS, EdgeType.DIRECTED)) \
      .add_edge(Edge(IR, LS, EdgeType.DIRECTED))

  # Converter o BBN em uma árvore de junção
  join_tree = InferenceController.apply(bbn)
  return  bbn, join_tree

bbn, join_tree = criar_rede(df, df)
```

Código

```
# Definir as posições dos nós (horizontal, vertical)
pos = {0: (-0.3, 1), 1: (-0.1, 1), 2: (0.1, 1), 3: (0.3, 1), 4: (0, -0.5)}

# Definir opções para a aparência do gráfico
options = {
    "font_size": 16,
    "node_size": 4000,
    "node_color": "white",
    "edgecolors": "black",
    "edge_color": "red",
    "linewidths": 5,
    "width": 5,}

# Gerar gráfico
n, d = bbn.to_nx_graph()
nx.draw(n, with_labels=True, labels=d, pos=pos, **options)

# Atualize as margens e imprima o gráfico
ax = plt.gca()
ax.margins(0.10)
plt.axis("off")
plt.show()
```

Código



Código

```
# O Grafico acima representa a dependencia e independencia entre as variaveis
# Defina uma função para imprimir as probabilidades
def print_probs():
    for node in join_tree.get_bbn_nodes():
        potential = join_tree.get_bbn_potential(node)
        print("Node:", node)
        print("Values:")
        print(potential)
        print('----------------')

# Use a função acima para imprimir probabilidades
print_probs()
```

Código

```
Node: 1|FA|0. <=0.5,1. >0.5
Values:
1=0. <=0.5|0.22343
1=1. >0.5|0.77657
----------------
Node: 2|DI|0. <=15,1. 15-30,2. >30
Values:
2=0. <=15|0.17974
2=1. 15-30|0.56892
2=2. >30|0.25134
----------------
Node: 3|IR|0. <=8,1. 8-15,2. >15
Values:
3=0. <=8|0.14524
3=1. 8-15|0.65620
3=2. >15|0.19857
----------------
Node: 4|LS|No,Yes
Values:
4=No|0.90847
4=Yes|0.09153
----------------
Node: 0|LA|0. <=10000,1. 10000-25000,2. >25000
Values:
0=0. <=10000|0.24950
0=1. 10000-25000|0.55937
0=2. >25000|0.19113
----------------
```

Código

```
# Para adicionar evidências de eventos que aconteceram para que a distribuição de probabilidade possa ser recalculada
def evidence(ev, nod, cat, val):
    ev = EvidenceBuilder() \
    .with_node(join_tree.get_bbn_node_by_name(nod)) \
    .with_evidence(cat, val) \
    .build()
    join_tree.set_observation(ev)

# Vamos supor que todos os emprestimos seja menores que 10 mil dolares, em teoria
# seria mais facil pagar as dividas se o emprestimo for menor
evidence('ev1', 'LA', '0. <=10000', 1.0)

# Imprimir probabilidades novas
print_probs()
```

```
----------------
Node: 4|LS|No,Yes
Values:
4=No|0.90846
4=Yes|0.09154
----------------
Node: 0|LA|0. <=10000,1. 10000-25000,2. >25000
Values:
0=0. <=10000|1.00000
0=1. 10000-25000|0.00000
0=2. >25000|0.00000
----------------
```

Código

```
# Vamos calcular o gráfico de utilidade. Como 67 mil registros demora de mais, então irei utilizar apenas 10%
# do dataframe de forma que eu pegue 10% dos inadimplentes e dos não inadimplentes dessa forma a proporção final não será alterada
dfi_10p = dfi.sample(frac=0.1) # 10% dos inadimplentes
dfni_10p = dfni.sample(frac=0.1) # 10% dos NÃO inadimplentes

# Juntando as duas partes
frames = [dfi_10p, dfni_10p]
df_10p = pd.concat(frames)
print("dfi + dfni = ?\n" + str(len(dfi_10p)), "+",len(dfni_10p),"=", len(dfi_10p)+len(dfni_10p))
df_10p
```

```
dfi + dfni = ?
624 + 6122 = 6746
```

Código

```python
# Categorizando a Utilidade pelo dinheiro obtido até o momento, isto é o valor emprestado menos valor pago
df_10p['Utilidade'] = df_10p['Funded Amount'] - df_10p['Loan Amount'] # Criando a coluna de utilidade
Lista_Ordenada = df_10p['Utilidade'].sort_values() # Ordenando os valores financeiros
df_10p['Index'] = [i for i in range(0, len(df_10p))] # Criando uma lista de index com base no tamanho de 10% do dataframe

Lista_Ordenada
```

```
46170   -31865
5853    -31471
5663    -30570
47102   -30280
23168   -30140
         ...
18864    30400
4582     30547
25952    30576
49780    31292
61638    31404
Name: Utilidade, Length: 6746, dtype: int64
```

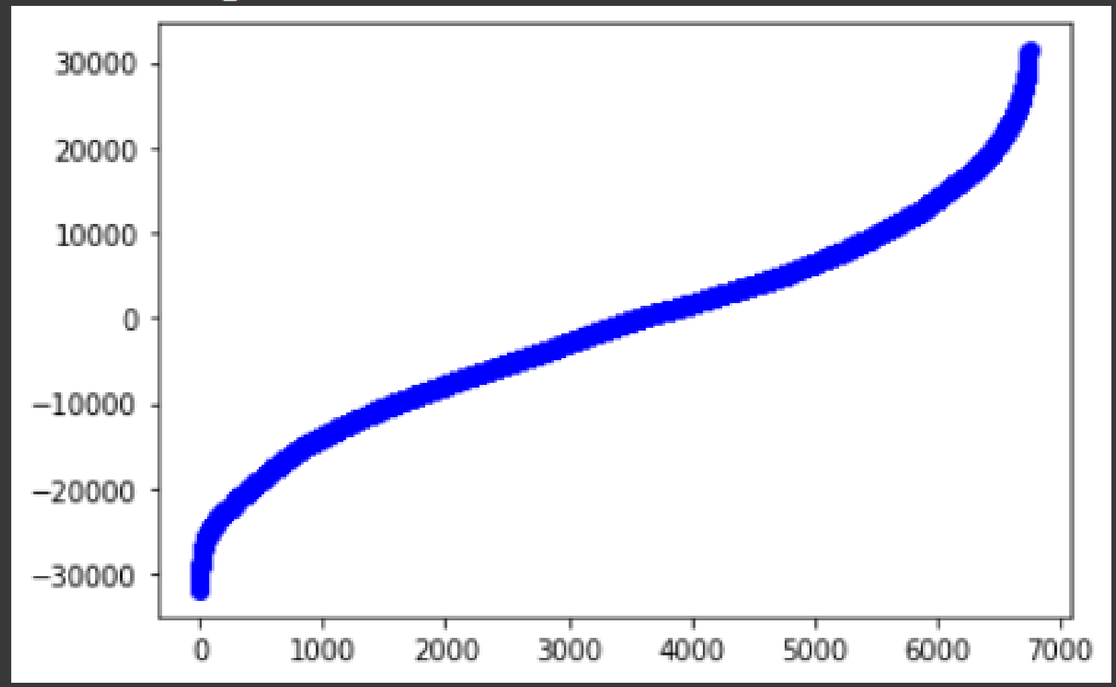
```python
# Normalizando a lista ordenada para saber proporção dos clientes o qual foi possivel obter lucro até o momento
Lista_Normalizada = Lista_Ordenada.apply(lambda x: '0. < 0'  if x < 0 else
                                                   '1. == 0' if x == 0 else
                                                   '2. > 0')
print("Clientes que deram prejuiso: ", len(Lista_Normalizada[Lista_Normalizada == '0. < 0']))
print("Clientes que pagaram exatamente o que pediram emprestado: ", len(Lista_Normalizada[Lista_Normalizada == '1. == 0']))
print("Clientes que deram lucro: ", len(Lista_Normalizada[Lista_Normalizada == '2. > 0']), "\n")
Lista_Normalizada.value_counts(normalize=True).sort_index()
```

```
Clientes que deram prejuiso:  3585
Clientes que pagaram exatamente o que pediram emprestado:  0
Clientes que deram lucro:  3161

0. < 0    0.531426
2. > 0    0.468574
Name: Utilidade, dtype: float64
```

Código

```
# Modelo gráfico inicial
xs = df_10p['Index']
ys = (df_10p['Utilidade'].sort_values())

if xs.size == ys.size:
  print("Tamanhos iguais?: ", xs.size, ys.size, xs.size == ys.size)
  # colors = cm.rainbow(np.linspace(0, 1, len(ys)))
  # for y, c in zip(ys, colors):
  #     plt.scatter(x, y, color=c)
  colors = itertools.cycle(["r", "b", "g"])
  for x,y in zip(xs,ys):
    plt.scatter(x, y, color="b") # next(colors))
```

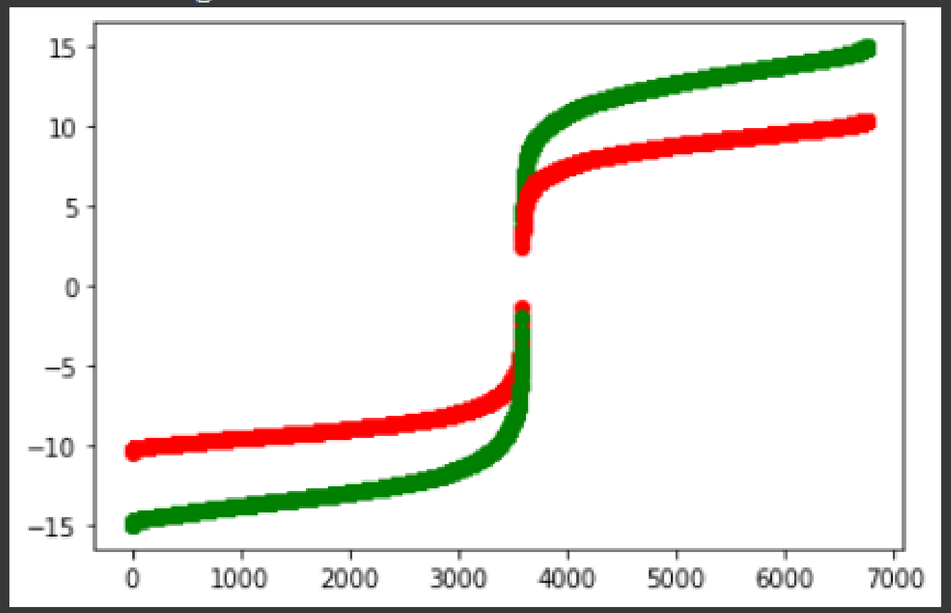
```
Tamanhos iguais?:  6746 6746 True
```



Código

```
# Caso o logaritmo na base dois seja aplicado é possivel ver algo similiar a dedução de Grayson (1960)
# em relação que a utilidade do dinheiro é proporcional ao logaritmo da quantia.
if xs.size == ys.size:
  print("Tamanhos iguais?: ", xs.size, ys.size, xs.size == ys.size)
  colors = itertools.cycle(["r", "b", "g"])
  for x,y in zip(xs,ys):
    valor_ln, valor_l2 = 0,0
    if y < 0:
      valor_ln = -(np.log(-y)) # /np.log(2))
      valor_l2 = -(np.log(-y)/np.log(2))
    else:
      valor_ln = np.log(y) # /np.log(2)
      valor_l2 = np.log(y)/np.log(2)
    plt.scatter(x, valor_ln, color="r") # Logaritmo natural
    plt.scatter(x, valor_l2, color="g") # Logaritmo na base 2
```

```
Tamanhos iguais?:  6746 6746 True
```



Código

```
# Abriando a base de teste
dft = pd.read_csv('test.csv', encoding='utf-8')
dft
```

	ID	Loan Amount	Funded Amount	Funded Amount Investor	Term	Batch Enrolled	Interest Rate	Grade
0	56492997	17120	10365	16025.082690	59	BAT2575549	12.163926	A
1	22540813	7133	11650	12615.795600	59	BAT2833642	6.564296	B

Código

Loan Status
NaN

```
# Abriando a base de teste
dft = pd.read_csv('test.csv', encoding='utf-8')
dft
```

	ID	Loan Amount	Funded Amount	Funded Amount Investor	Term	Batch Enrolled	Interest Rate	Grade
0	56492997	17120	10365	16025.082690	59	BAT2575549	12.163926	A
1	22540813	7133	11650	12615.795600	59	BAT2833642	6.564296	B

Código

```
# Pega cem amostras aleatorias
dfts = dft.sample(n=100)

# Seta as variaveis para a base de teste
setar_variaveis(dft)

# Refaz a arvore com a base de teste e calcula posteriori
bbn, join_tree = criar_rede(dft, df)

lista_porcentagem = []
for i in range(0,len(dfts)):
  # Colocando a evidencia de acordo com o cliente especifico
  evidence('ev1', 'IR', dfts.iloc[i]['Interest Rate Cat'], 1.0)
  evidence('ev2', 'DI', dfts.iloc[i]['Debit to Income Cat'], 1.0)
  evidence('ev3', 'FA', dfts.iloc[i]['Funded Amount Cat'], 1.0)
  evidence('ev4', 'LA', dfts.iloc[i]['Loan Amount Cat'], 1.0)

  # Obtém a tabela de probabilidade condicional do nó
  node = join_tree.get_bbn_nodes()[3]
  potential = join_tree.get_bbn_potential(node)

  # Tratando resposta
  r = str(potential)
  r = r.split('|')
  r = [float(r[1][:7]), float(r[2])] # No | Yes
  lista_porcentagem.append(r[0]) # Adicionando na lista de probabilidades

# Registrando a chance de não inadimplencia e fazendo a decisão final
# neste caso se a chance de não inadimplencia for maior que 90% ele faz
# o emprestimo
dfts['Chace não inadimplente'] = lista_porcentagem
dfts['Decisão']=dfts['Chace não inadimplente'].apply(lambda x: 'Não' if x < 0.9 else
                                                               'Sim')

dfts
```

Código

Chace não inadimplente	Decisão
0.92308	Sim
0.90321	Sim
0.89985	Não
0.90621	Sim

```
# Aqui está a normalização dos valores de decisão final
D = dfts['Decisão'].value_counts(normalize=True).sort_index()
D
```

```
Não    0.03
Sim    0.97
Name: Decisão, dtype: float64
```

30 Bibliografia

- *Slides da aula disponibilizados no classroom;*
- *Vídeo aula disponibilizada no classroom;*
- *Documentação da biblioteca pandas: https://pandas.pydata.org/docs*
- *Documentação da biblioteca matplotlib:*
- *https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html*