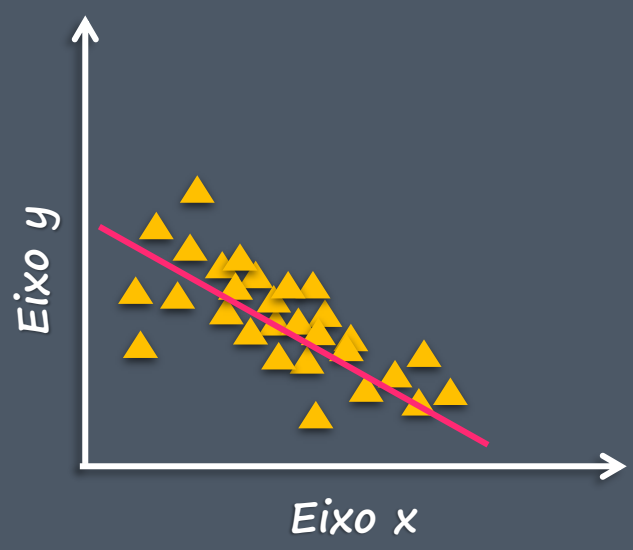
DATA Viking

Tutorial

Credit Scoring
Machine Learning



Odemir Depieri Jr
Ronisson Lucas C. da Conceição
Yan Vianna Sym

O que é Credit Scoring?

- Concessão de crédito envolve um complexo **cenário de incertezas**, em que sempre existe a possibilidade de perda. Então, se pudermos estimar a **probabilidade da perda**, poderemos tomar melhores decisões. A partir disso, podemos definir clientes bons e clientes maus (quando incorremos em perda em uma operação de crédito).

- Objetivo da **modelagem de crédito** é então prever a probabilidade da perda, tendo em vista que o crédito tenha sido concedido. A probabilidade de perda podemos denominar de risco de crédito.

Risco de crédito = probabilidade de perda

Credit score é uma medida de risco de crédito.

A partir desta medida temos decisões como conceder ou não crédito, **definir taxas**, **prazos**, **garantias**, dentre outras questões que são de atribuição dos gestores de crédito. Outras decisões que podem ser tomadas com as faixas de score são: **manutenção/adequação do limite de crédito**; **estratégias** de marketing para oferta de outros produtos a depender do perfil do cliente, dentre outras.

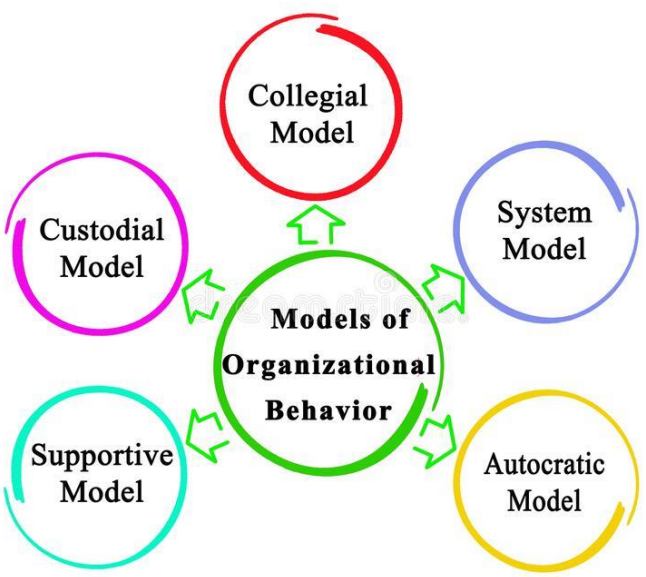
O que é Credit Scoring?

Vantagens de se mensurar o risco de crédito de forma objetiva (com técnicas quantitativas):

- **Consistência nas decisões**: o escore é o mesmo independentemente do analista ou da agência, dado que as características da solicitação sejam inalteradas.
- Decisões rápidas: centenas/milhares de decisões podem ser tomadas.
- **Decisões adequadas**: definições de taxas a depender do grau de risco do cliente.
- **Decisão à distância**: após a imputação de dados obtemos uma decisão de crédito a ser repassada ao solicitante.
- Aplicações:
- **Decisão de crédito em massa** (alta demanda de solicitação de crédito em um pequeno intervalo de tempo).
- **Processo automatizado de decisão**.

Modelos de crédito

- Podemos aplicar modelos de credit scoring para **solicitantes de crédito** que sejam clientes ou ex-clientes, estes são denominados de behavioral scoring. Por outro lado, também podemos desenvolver modelos para solicitantes que não sejam clientes ou que não tenhamos um histórica.
- Nos **modelos de behavioral** usamos o histórico de uso de crédito, além de variáveis consideradas nos modelos de application scoring.
- Por exemplo, no caso de clientes que honram seus empréstimos temos que essa informação contribuirá positivamente. Já clientes que tiveram atraso teremos uma contribuição negativa para o escore.
- Os **modelos de behavioral** tendem a apresentar um grau maior de discriminação em comparação com os modelos de application.



Importação dos dados

```
# Importando libs
import pandas as pd
import pickle
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cro
ss_validate, StratifiedKFold, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')

tb_job = pd.read_csv('dados/tb_job.csv').set_index('id')
tb_cadastro = pd.read_csv('dados/tb_cadastro.csv').set_i
ndex('id')
tb_atraso = pd.read_csv('dados/tb_atraso.csv').set_index
('id')

tb_cadastro.head()
```

id	tp_sexo	uf	status_civil	n_dependentes	casa_propria
8927457	M	SP	SOLTEIRO	0	1
22009007	M	SP	SOLTEIRO	0	0
30794041	F	SP	SOLTEIRO	0	0
13342495	M	RJ	SOLTEIRO	0	0
73110407	M	SP	C	0	1

```
tb_job.head()
```

id	escolaridade	tempo_empregado	modelo_renda_v2	classe_renda_inf
42543947	Ensino Fundamental	9	1517	B
49524184	Ensino Fundamental	9	1654	B
29697248	Ensino Superior	5	1468	A
61147127	Ensino Fundamental	4	1535	B
25601658	Ensino Médio	14	1399	A

Importação dos dados

```
tb_atraso.head()
```

id	max_dias_atraso_6m	programa_reneg
59692101	59	NÃO
3621563	64	NÃO
17013828	71	SIM
56633600	52	NÃO
34502897	56	NÃO

Cruzamento dos dados

```
# junção das tabelas
data = pd.concat([tb_job,
                  tb_cadastro,
                  tb_atraso],
                 axis = 1)

data.head()
```

id	escolaridade	tempo_empregado	modelo_renda_v2	classe_renda_inf	tp_sexo	uf	status_civil	n_dependentes	casa_propria	max_
1000279	Ensino Médio	16	2015	C	M	RJ	CASADO	0	0	
1000558	Ensino Fundamental	8	1715	B	F	RJ	SOLTEIRO	0	0	
1000841	Ensino Médio	8	2065	C	F	RJ	SOLTEIRO	0	1	
1000923	Ensino Médio	7	1703	B	M	SP	CASADO	0	0	
1002651	Ensino Médio	7	2226	C	M	RJ	CASADO	1	0	

```
# dimensionalidade dos dados
data.shape
```

```
(249678, 11)
```

```
# informações da base de dados
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 249678 entries, 1000279 to 99999909
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   escolaridade        249678 non-null  object
 1   tempo_empregado     249678 non-null  int64
 2   modelo_renda_v2     249678 non-null  int64
 3   classe_renda_inf    249678 non-null  object
 4   tp_sexo             249678 non-null  object
 5   uf                  249678 non-null  object
 6   status_civil        249678 non-null  object
 7   n_dependentes       249678 non-null  int64
 8   casa_propria        249678 non-null  int64
 9   max_dias_atraso_6m  249678 non-null  int64
 10  programa_reneg      249678 non-null  object
dtypes: int64(5), object(6)
memory usage: 22.9+ MB
```

Limpeza dos Dados

```
def contar_linhas_duplicadas(data_frame):
    "Recebe um DataFrame e retorna se o DataFrame possui
 linhas duplicadas ou não."
    contar_duplicidade = data_frame.duplicated().sum()
    if contar_duplicidade > 0:
        return True
    return False

def colunas_constantes(data_frame):
    "Retorna uma lista com as colunas constantes do DataFr
ame."
    return data_frame.columns[data_frame.nunique() == 1].t
olist()

def low_variance(data_frame, threshold):
    "Conta a variância das variáveis do DataFrame, mapeand
o low variance."
    minmax_scaler = MinMaxScaler()
    numerical_columns = data.select_dtypes('number')
    data_frame_scaled = pd.DataFrame(minmax_scaler.fit_
transform(numerical_columns),
                                     columns = numerical_columns.colum
ns.tolist()
                                    )

    low_variance = list()
    for column in data_frame_scaled:
        var = data_frame_scaled[column].var()
        if var < threshold:
            low_variance.append(column)

    return low_variance

def pct_nan(data_frame):
    "Mensura o percentual de NaN."
    return np.multiply(data_frame.isna().sum() / data_fram
e.shape[0], 100).round(2)
```

Limpeza dos Dados

```
# sumário estatístico das variáveis
data.describe().T.round(1)
```

	count	mean	std	min	25%	50%	75%	max
tempo_empregado	249678.0	10.0	3.2	0.0	8.0	10.0	12.0	27.0
modelo_renda_v2	249678.0	1769.2	351.7	1200.0	1513.0	1704.0	2037.0	9841.0
n_dependentes	249678.0	0.5	0.9	0.0	0.0	0.0	1.0	4.0
casa_propria	249678.0	0.2	0.4	0.0	0.0	0.0	0.0	1.0
max_dias_atraso_6m	249678.0	60.0	7.7	28.0	55.0	60.0	65.0	95.0

```
# check de missing
data.isna().sum()
```

```
escolaridade          0
tempo_empregado       0
modelo_renda_v2       0
classe_renda_inf      0
tp_sexo               0
uf                    0
status_civil          0
n_dependentes         0
casa_propria          0
max_dias_atraso_6m    0
programa_reneg        0
dtype: int64
```

```
# valores únicos em cada coluna
data.nunique()
```

```
escolaridade             4
tempo_empregado         27
modelo_renda_v2       1550
classe_renda_inf         3
tp_sexo                  2
uf                       2
status_civil             4
n_dependentes            5
casa_propria             2
max_dias_atraso_6m      68
programa_reneg           2
dtype: int64
```

```
# check de linhas duplicadas
contar_linhas_duplicadas(data)
```

```
True
```

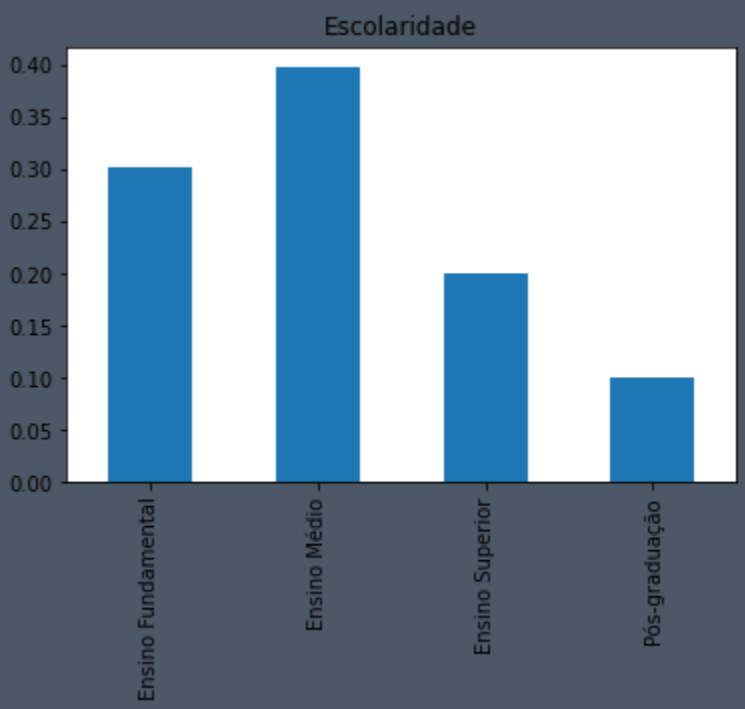
```
# remove linhas duplicadas
data.drop_duplicates(inplace = True)

# check de linhas duplicadas
contar_linhas_duplicadas(data)
```

```
False
```

Data Visualization

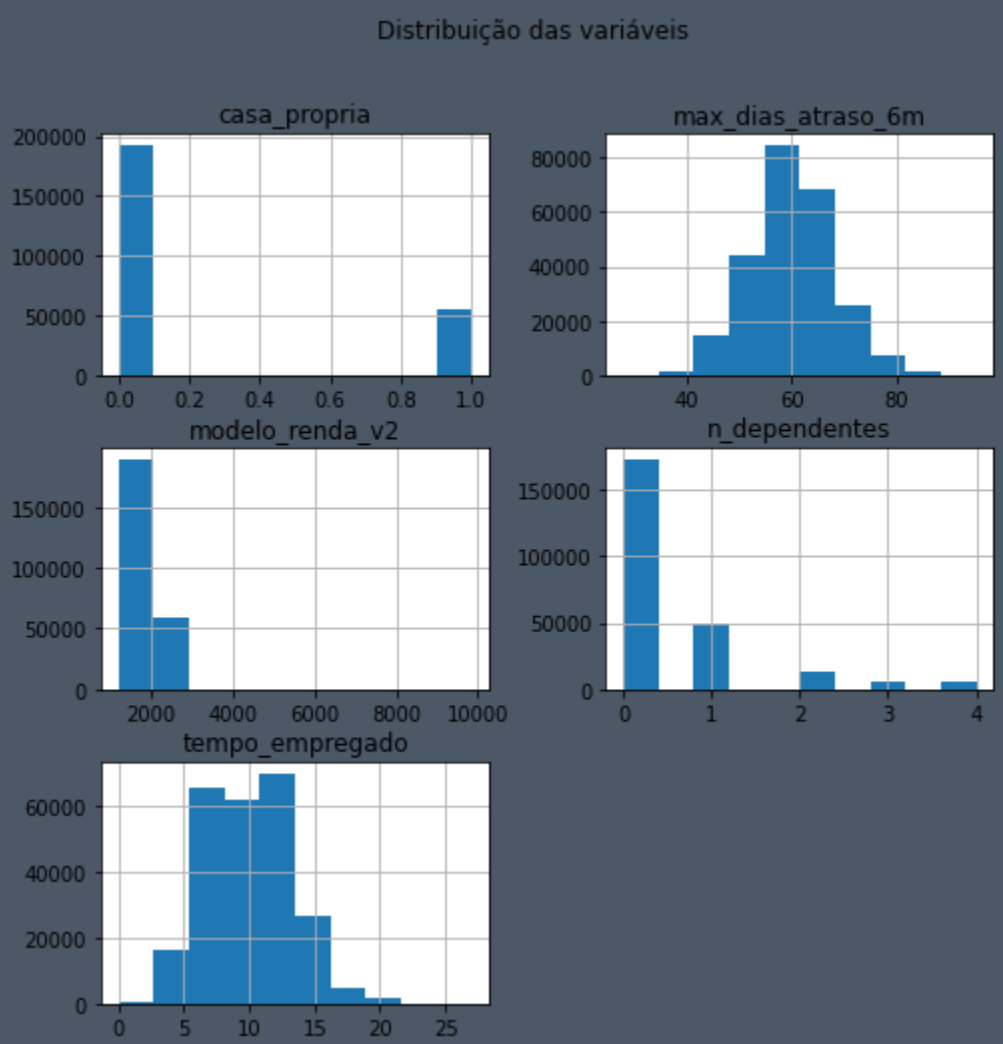
```
data.escolaridade.value_counts(normalize = True).sort_index().plot(kind = 'bar')
plt.title('Escolaridade')
```




```
data['classe_renda_inf'].value_counts(normalize = True).sort_index().plot(kind = 'barh')
plt.title('Clientes por nível de renda')
```

Data Visualization

```
fig, ax = plt.subplots(figsize = (8,8))
data[['tempo_empregado', 'modelo_renda_v2', 'n_dependentes', 'casa_propria', 'max_dias_atraso_6m']].hist(ax = ax)
fig.suptitle('Distribuição das variáveis')
```



Feature Engineering

```
# seleciona colunas do tipo object
cat_cols = data.select_dtypes(include = 'object')
```

Feature Engineering

```
# verifica valores únicos de cada coluna do tipo object
for col in cat_cols:
    print(col)
    print(cat_cols[col].unique());print()
```

```
escolaridade
['Ensino Médio' 'Ensino Fundamental' 'Ensino Superior' 'Pós-graduação']

classe_renda_inf
['C' 'B' 'A']

tp_sexo
['M' 'F']

uf
['RJ' 'SP']

status_civil
['CASADO' 'SOLTEIRO' 'C' 'CAsado']

programa_reneg
['NÃO' 'SIM']
```

```
# tratamento das variáveis
data['escolaridade'] = data.escolaridade.map({'Ensino Fundamental': 1, 'Ensino Médio': 2, 'Ensino Superior': 3, 'Pós-graduação': 4 })

data['classe_renda_inf'] = data.classe_renda_inf.map({'A': 1, 'B': 2, 'C': 3 })

data['tp_sexo'] = data.tp_sexo.map({'M': 1, 'F': 0})

data['uf'] = data.uf.map({'SP': 1, 'RJ': 0})

data['status_civil'] = data.status_civil.str.lower().map({'solteiro': 0, 'c': 1, 'casado': 1})

data['programa_reneg'] = data.programa_reneg.map({'SIM': 1, 'NÃO': 0})

data['target'] = data.max_dias_atraso_6m.apply(lambda x: 1 if x >= 60 else 0)
```

Divisão dos Dados

```
# features: variáveis explicativas do modelo
X = data.drop(columns = ['target', 'max_dias_atraso_6m']
)

# target: variável-alvo da previsão
y = data.target

y.value_counts(normalize = True)

SEED = 1321
# separa os dados em conjunto de treinamento e conjunto
de teste
X_train, X_test, y_train, y_test = train_test_split(X, y, r
andom_state = SEED,
                                                    test_size = 0.3
                                                   )
```

Validação Cruzada

```
# define validação cruzada
cv = StratifiedKFold(n_splits=5, shuffle=True, random_st
ate = SEED)
reglog = LogisticRegression()
cv_scores_reglog = cross_validate(
    estimator = reglog,
    X = X_train,
    y = y_train,
    scoring = 'roc_auc',
    cv = cv,
    return_train_score = True
)

cv_scores_reglog
```

```
{'fit_time': array([2.36624789, 1.97572827, 2.0905726 , 1.98471737, 1.96068835]),
 'score_time': array([0.04176831, 0.02595019, 0.0269289 , 0.02587056, 0.02590609]),
 'test_score': array([0.83072235, 0.84672394, 0.82364489, 0.82002474, 0.82032518]),
 'train_score': array([0.83074953, 0.84625701, 0.82220496, 0.82207345, 0.82038599])}
```

Validação Cruzada

```
cv_scores_reglog['test_score'].mean()
```

```
0.8283341855096479
```

```
cv_scores_reglog['train_score'].mean()
```

```
0.8282882197487667
```

```
# define validação cruzada
cv = StratifiedKFold(n_splits=5, shuffle=True, random_st
ate = SEED)
dt_clf = DecisionTreeClassifier()
cv_scores_dt = cross_validate(
    estimator = dt_clf,
    X = X_train,
    y = y_train,
    scoring = 'roc_auc',
    cv = cv,
    return_train_score = True
)
cv_scores_dt
```

```
{'fit_time': array([0.56239104, 0.42903972, 0.44313836, 0.45825958, 0.45307231]),
 'score_time': array([0.0188961 , 0.0191915 , 0.02073073, 0.01926613, 0.01917267]),
 'test_score': array([0.75174578, 0.7542887 , 0.74863133, 0.75417795, 0.75047237]),
 'train_score': array([0.99898415, 0.99903513, 0.99908578, 0.99902691, 0.99902914])}
```

```
cv_scores_dt['train_score'].mean()
```

```
0.9990322223373267
```

```
cv_scores_dt['test_score'].mean()
```

```
0.7518632238522052
```

Validação Cruzada

```
# define validação cruzada
cv = StratifiedKFold(n_splits=5, shuffle=True, random_st
ate = SEED)
dt_clf = DecisionTreeClassifier(max_depth = 5)
cv_scores_dt = cross_validate(
    estimator = dt_clf,
    X = X_train,
    y = y_train,
    scoring = 'roc_auc',
    cv = cv,
    return_train_score = True
)

cv_scores_dt['train_score'].mean(), cv_scores_dt['test_s
core'].mean()
```

```
(0.8884426294672212, 0.8870216815201222)
```

Random Forest

```
# define validação cruzada
cv = StratifiedKFold(n_splits=5, shuffle=True, random_st
ate = SEED)
rf_clf = RandomForestClassifier(max_depth = 5)
cv_scores_rf = cross_validate(
    estimator = rf_clf,
    X = X_train,
    y = y_train,
    scoring = 'roc_auc',
    cv = cv,
    return_train_score = True
)

cv_scores_rf['train_score'].mean(), cv_scores_rf['test_s
core'].mean()
```

```
0.8891260908183252, 0.8864660762766142
```

Hiper parâmetros

```
# define validação cruzada
cv = StratifiedKFold(n_splits=2, shuffle=True, random_st
ate = SEED)

param_grid = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : list(range(1, 11)),
    'min_samples_leaf' : range(2, 21, 2),
    'min_samples_split' : range(2, 41, 5),
}

rf_model = RandomForestClassifier(max_depth = 5)
rf_model = RandomizedSearchCV(
    estimator = rf_model,
    param_distributions = param_grid,
    scoring = 'roc_auc',
    random_state = SEED,
    n_jobs = -1,
    verbose = 1,
    n_iter = 3
)

rf_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits

RandomizedSearchCV(estimator=RandomForestClassifier(max_depth=5), n_iter=3,
                   n_jobs=-1,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                      10],
                                        'min_samples_leaf': range(2, 21, 2),
                                        'min_samples_split': range(2, 41, 5)},
                   random_state=1321, scoring='roc_auc', verbose=1)
```

```
y_pred_proba_train = rf_model.predict_proba(X_train)
y_pred_proba_test = rf_model.predict_proba(X_test)
y_pred_train = rf_model.predict(X_train)
y_pred_test = rf_model.predict(X_test)

roc_auc_train = roc_auc_score(y_train, y_pred_proba_tr
ain[:, 1])
roc_auc_test = roc_auc_score(y_test, y_pred_proba_tes
t[:, 1])
roc_auc_train, roc_auc_test
```

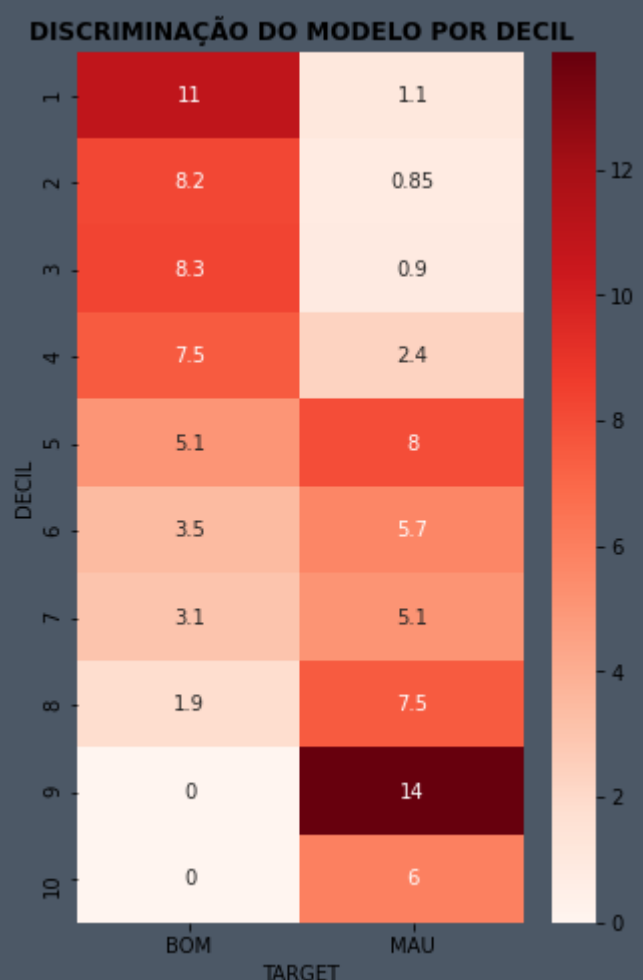
```
0.8901682483972364, 0.8877400538895949
```

Hiper parâmetros

```
result = X_test.reset_index()[['id']]
result['SCORE'] = np.multiply(y_pred_proba_test[:, 1], 10
00).astype(int)
result['TARGET'] = y_test.values
result['DECIL'] = pd.qcut(result.SCORE, 10, labels = [i for
i in range(1, 11)])

decile_df = pd.crosstab(
    result['DECIL'],
    result['TARGET'].map({1:'MAU', 0: 'BOM'}),
    normalize = True
) * 100

fig, ax = plt.subplots(figsize = (5, 8))
ax.set_title('DISCRIMINAÇÃO DO MODELO POR DECIL
', weight = 'bold')
sns.heatmap(decile_df, cmap = 'Reds', annot = True, ax = a
x)
plt.show()
```



Régua de decisão

* SCORE < 300: A
  * Decisão: conceder produto
* SCORE >= 300 e SCORE <= 600: B
  * Decisão: mesclar decisão com outro modelo ou decisão do analista
* SCORE > 600: C
  * Decisão: não conceder produto

```
def classificar_cliente(score):
    if score < 300: return 'A'
    elif 300 <= score <= 600: return 'B'
    elif score > 600: return 'C'
result['DECISION'] = result.SCORE.apply(classificar_clie
nte)
result
```

	id	SCORE	TARGET	DECIL	DECISION
0	45630330	96	0	1	A
1	56114485	601	0	4	C
2	32029382	612	0	8	C
3	97109191	985	1	9	C
4	28706788	110	0	3	A
...	...	...	...	...	...
74391	81523025	606	1	5	C
74392	10088950	143	0	4	A
74393	56613498	609	1	6	C
74394	26006330	99	1	1	A
74395	99525827	96	0	1	A

74396 rows × 5 columns

Código (script)

Para acessar esse script acesse o *qrcode* abaixo.