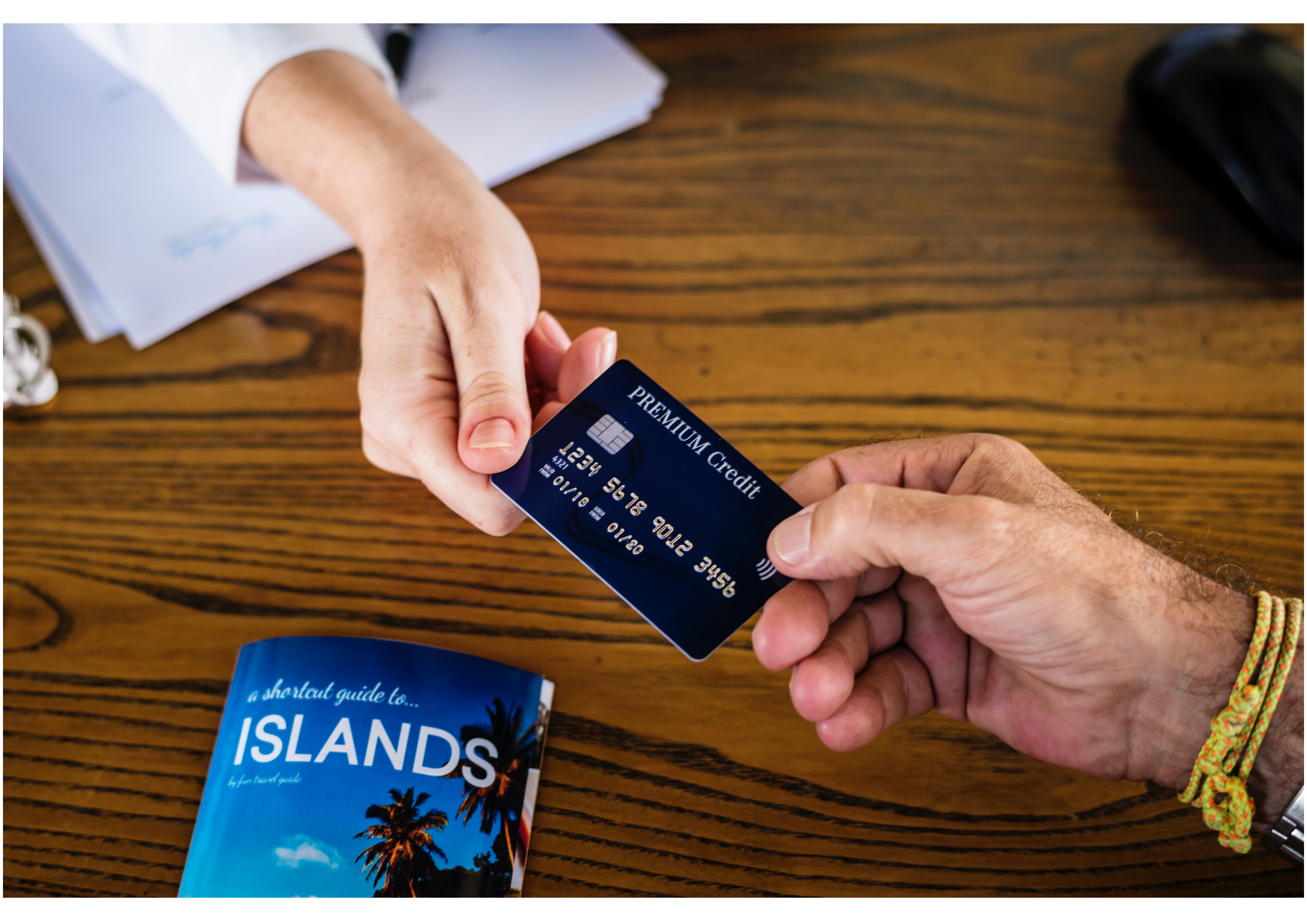
1. Pedidos de crédito

Bancos recebem *muitos* pedidos de crédito e muitos são rejeitados por vários motivos. Analizar manualmente esses pedidos é exaustivo, muito sucetível a falhas e demorado (e tempo é dinheiro). Felizmente essa tarefa pode ser automatizada com aprendizado de máquina (*machine learning*) e praticamente todo banco faz isso hoje em dia. Neste notebook veremos a construção de um preditor automático de aprovação de crédito utilizando técnicas de *machine learning*, assim como bancos reais fazem.



Utilizaremos o Credit Card Approval dataset do repositório de dados "UCI Machine Learning Repository". Segue a organização deste notebook:

- Primeiro, vamos carregar e visualizar o dataset.
- Veremos que o dataset é uma mistura tanto de características numéricas e não numéricas, que contém valores de diferentes amplitudes, além de valores faltantes.
- Teremos que pré processar o dataset para garantir que o modelo que escolhermos possa fazer boas previsões
- Na sequencia faremos uma análise exploratória dos dados
- Finalmente, contruiremos um modelo de machine learning que seja capaz de prever se o pedido de crédito de um cliente será aprovado ou não.

Primeiro, carregaremos o dataset. Já que os dados são confidenciais, o contribuidor do dataset anonimizou os nomes das características. Trataremos cada coluna como um número. Isso não deverá ser um problema para este projeto.

A coluna que queremos prever (o nosso *target*) é a coluna 15

In [1]:

```python
import pandas as pd
cc_apps = pd.read_csv("datasets/cc_approvals.data", header=None)
cc_apps.head()
```

Out[1]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

2. Inspecionando os pedidos

Os dados podem parecer confusos a primeira vista. Como já mencionado, as features deste dataset foram anonimizadas para proteger a privacidade dos clientes, mas este blog nos dá uma boa idéia de o que as features provavelmente são. Provavelmente temos `Sexo`, `Idade`, `Débito`, `Estado civil`, `Cliente Tipo`, `Escolaridade`, `Etinia`, `Anos empregado`, `Dívida anterior`, `Empregado`, `Score de crédito`, `Carteira de Habilitação`, `Cidadania`, `CEP`, `Renda` e finalmente o `Status de Aprovação`. Isso nos dá um bom ponto de partida caso precisemos mapear essas features no resultado.

O dataset é uma mistura de dados numéricos e categóricos. Isso pode ser corrigido com algum pré-processamento, mas antes disso, vamos aprender mais sobre o dataset para descobrir o que mais precisa ser corrigido.

In [2]:

```python
# Imprimindo describe() do dataset
cc_apps_description = cc_apps.describe()
print(cc_apps_description)

print("\n")

# Imprimindo info() do dataset
cc_apps_info = cc_apps.info()
print(cc_apps_info)

print("\n")

#total de valores nulos

cc_apps.isnull().sum()
```

```
                2           7          10             14
count  690.000000  690.000000  690.00000     690.000000
mean     4.758725    2.223406    2.40000    1017.385507
std      4.978163    3.346513    4.86294    5210.102598
min      0.000000    0.000000    0.00000       0.000000
25%      1.000000    0.165000    0.00000       0.000000
50%      2.750000    1.000000    0.00000       5.000000
75%      7.207500    2.625000    3.00000     395.500000
max     28.000000   28.500000   67.00000  100000.000000


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       690 non-null    object 
 1   1       690 non-null    object 
 2   2       690 non-null    float64
 3   3       690 non-null    object 
 4   4       690 non-null    object 
 5   5       690 non-null    object 
 6   6       690 non-null    object 
 7   7       690 non-null    float64
 8   8       690 non-null    object 
 9   9       690 non-null    object 
 10  10      690 non-null    int64  
 11  11      690 non-null    object 
 12  12      690 non-null    object 
 13  13      690 non-null    object 
 14  14      690 non-null    int64  
 15  15      690 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 86.4+ KB
None
```

Out[2]:

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
dtype: int64
```

3. Tratando valores faltantes (parte i)

Alguns problemas que vão afetar nosso modelo caso não sejam tratados:

- Nosso dataset contém valores numéricos e categóricos
- O data set contém valores de várias amplitures. Algumas características variam entre 0 - 28, algumas entre 2 - 67, e outras tem uma variação de 1017 - 100000. Tirando essas, nós conseguimos dados estatísticas úteis (como `média`, `máximo`, e `mínimo`) sobre as características que tem valores numéricos.
- Finalmente, o dataset contém valores faltantes, o qual nós trataremos aqui. Apesar de não existirem valores de fato nulos (NaN), existem valores rotulados com '?'.

Primeiro, vamos temporariamente subtituir '?' por NaN.

Abaixo podemos observar uma ocorrencia na coluna 0

In [3]:

```python
import numpy as np

# Inspecionando valores faltantes
print(cc_apps.tail(20))

# Verificando novamente...
print("\n##########\n"),
cc_apps = cc_apps.replace("?", np.nan)
print(cc_apps.tail(20))
```

```
    0      1       2  3  4   5   6      7  8  9   10 11 12     13   14 15
670  b  47.17   5.835  u  g   w   v  5.500  f  f   0  f  g  00465  150  -
671  b  25.83  12.835  u  g  cc   v  0.500  f  f   0  f  g  00000    2  -
672  a  50.25   0.835  u  g  aa   v  0.500  f  f   0  t  g  00240  117  -
673  ?  29.50   2.000  y  p   e   h  2.000  f  f   0  f  g  00256   17  -
674  a  37.33   2.500  u  g   i   h  0.210  f  f   0  f  g  00260  246  -
675  a  41.58   1.040  u  g  aa   v  0.665  f  f   0  f  g  00240  237  -
676  a  30.58  10.665  u  g   q   h  0.085  f  t  12  t  g  00129    3  -
677  b  19.42   7.250  u  g   m   v  0.040  f  t   1  f  g  00100    1  -
678  a  17.92  10.210  u  g  ff  ff  0.000  f  f   0  f  g  00000   50  -
679  a  20.08   1.250  u  g   c   v  0.000  f  f   0  f  g  00000    0  -
680  b  19.50   0.290  u  g   k   v  0.290  f  f   0  f  g  00280  364  -
681  b  27.83   1.000  y  p   d   h  3.000  f  f   0  f  g  00176  537  -
682  b  17.08   3.290  u  g   i   v  0.335  f  f   0  t  g  00140    2  -
683  b  36.42   0.750  y  p   d   v  0.585  f  f   0  f  g  00240    3  -
684  b  40.58   3.290  u  g   m   v  3.500  f  f   0  t  s  00400    0  -
685  b  21.08  10.085  y  p   e   h  1.250  f  f   0  f  g  00260    0  -
686  a  22.67   0.750  u  g   c   v  2.000  f  t   2  t  g  00200  394  -
687  a  25.25  13.500  y  p  ff  ff  2.000  f  t   1  t  g  00200    1  -
688  b  17.92   0.205  u  g  aa   v  0.040  f  f   0  f  g  00280  750  -
689  b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000    0  -

##########

       0      1       2  3  4   5   6      7  8  9   10 11 12     13   14 15
670    b  47.17   5.835  u  g   w   v  5.500  f  f   0  f  g  00465  150  -
671    b  25.83  12.835  u  g  cc   v  0.500  f  f   0  f  g  00000    2  -
672    a  50.25   0.835  u  g  aa   v  0.500  f  f   0  t  g  00240  117  -
673  NaN  29.50   2.000  y  p   e   h  2.000  f  f   0  f  g  00256   17  -
674    a  37.33   2.500  u  g   i   h  0.210  f  f   0  f  g  00260  246  -
675    a  41.58   1.040  u  g  aa   v  0.665  f  f   0  f  g  00240  237  -
676    a  30.58  10.665  u  g   q   h  0.085  f  t  12  t  g  00129    3  -
677    b  19.42   7.250  u  g   m   v  0.040  f  t   1  f  g  00100    1  -
678    a  17.92  10.210  u  g  ff  ff  0.000  f  f   0  f  g  00000   50  -
679    a  20.08   1.250  u  g   c   v  0.000  f  f   0  f  g  00000    0  -
680    b  19.50   0.290  u  g   k   v  0.290  f  f   0  f  g  00280  364  -
681    b  27.83   1.000  y  p   d   h  3.000  f  f   0  f  g  00176  537  -
682    b  17.08   3.290  u  g   i   v  0.335  f  f   0  t  g  00140    2  -
683    b  36.42   0.750  y  p   d   v  0.585  f  f   0  f  g  00240    3  -
684    b  40.58   3.290  u  g   m   v  3.500  f  f   0  t  s  00400    0  -
685    b  21.08  10.085  y  p   e   h  1.250  f  f   0  f  g  00260    0  -
686    a  22.67   0.750  u  g   c   v  2.000  f  t   2  t  g  00200  394  -
687    a  25.25  13.500  y  p  ff  ff  2.000  f  t   1  t  g  00200    1  -
688    b  17.92   0.205  u  g  aa   v  0.040  f  f   0  f  g  00280  750  -
689    b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000    0  -
```

4. Tratando valores faltantes (parte ii)

Substituímos '?' com NaNs.

Usaremos a estratégia de imputação da média para os valores faltantes.

In [4]:

```python
# Impute the missing values with mean imputation
cc_apps.fillna(cc_apps.mean(), inplace=True)

# Contagem dos valores faltantes
cc_apps.isnull().sum()
```

Out[4]:

```
0     12
1     12
2      0
3      6
4      6
5      9
6      9
7      0
8      0
9      0
10     0
11     0
12     0
13    13
14     0
15     0
dtype: int64
```

5. Tratando valores faltantes (parte iii)

Tratamos os valores falantes com sucesso, porém somente nas colunas numéricas. Ainda existem valores faltantes nas colunas 0, 1, 3, 4, 5, 6 e 13. Todas essas colunas contém dados não numéricos. Portanto a imputação da média não funcionaria aqui.

Uma boa prática é imputar com o valor mais frequente, como faremos em seguida.

In [5]:

```python
for col in cc_apps.columns:
    if cc_apps[col].dtype == 'object':
        cc_apps = cc_apps.fillna(cc_apps[col].value_counts().index[0])

# Contagem dos valores faltantes
cc_apps.isnull().sum()
```

Out[5]:

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
dtype: int64
```

6. Pré-processamento dos dados (parte i)

Resolvemos os valores faltantes.

Ainda existem outras etapas de pré-processamento que devemos executar antes de partirmos para o modelo. Dividiremos essas etapas em três partes:

1. Converter dados não numéricos para numéricos
2. Dividir os dados em treino e teste
3. Escalonar os dados para um intervalo uniforme

Primeiro, vamos converter todos valores categóricos em numéricos. Nós faremos isso não somente por resultar em uma computação mais rápida mas também porque muitos modelos de *machine learning* (como o XGBoost) (e especialemnte aqueles que foram desenvolvidos usando scikit-learn) requerem que os dados sejam numéricos. Faremos isso usando uma técnica chamada label encoding.

In [6]:

```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in cc_apps.columns.values:
    if cc_apps[col].dtype =='object':
        cc_apps[col] = le.fit_transform(cc_apps[col])
```

In [7]:

```python
cc_apps.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	156	0.000	2	1	13	8	1.25	1	1	1	0	0	68	0	0
1	0	328	4.460	2	1	11	4	3.04	1	1	6	0	0	11	560	0
2	0	89	0.500	2	1	11	4	1.50	1	0	0	0	0	96	824	0
3	1	125	1.540	2	1	13	8	3.75	1	1	5	1	0	31	3	0
4	1	43	5.625	2	1	13	8	1.71	1	0	0	0	2	37	0	0

`Sexo`, `Idade`, `Débito`, `Estado civil`, `Cliente Tipo`, `Escolaridade`, `Etinia`, `Anos empregado`, `Dívida anterior`, `Empregado`, `Score de crédito`, `Carteira de Habilitação`, `Cidadania`, `CEP`, `Renda` e finalmente o `Status de Aprovação`

7. Dividindo os dados em sets de treino e teste

Agora vamos dividir os dados em treino e teste. Normalmente, nenhuma informação dos dados de teste deve ser usada para escalonar os dados de treino ou usada para direcionar a fase de treino de uma modelo de machine learning. Portanto, primeiro diviremos os dados e então aplicaremos o escalonamento.

Além disso, características like `Carteira de Habilitação` e `CEP` não são tão importantes quanto outras carterísticas para prever aprovação de crédito. Nõs devemos remove-las para fornecer ao modelo o melhor conjunto de características. Na literatura sobre Ciência de Dados, isso é comumente chamado de *feature selection*.

In [8]:

```python
from sklearn.model_selection import train_test_split

# Removendo as características 11 e 13 e convertendo o DataFrame em um array NumPy
cc_apps = cc_apps.drop([11, 13], axis=1)

cc_apps = cc_apps.values

# Separandoa s características e labels em diferentes variáveis
X,y = cc_apps[:,0:-1] , cc_apps[:,-1]

# Dividindo entre treino e teste, sendo teste 33% dos dados.
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33, random_state=42)
```

8. Pré-processamento dos dados (parte ii)

Só nos restou mais uma etapa de pré-processamento que é escalonamento dos dados antes que treinemos o modelo.

Agora, vamos tentar entender o que esses dados escalonados representam no mundo real. Vamos usar `Score de crédito` como exemplo. O score de crédito é dado com base no histórico de crédito do cliente. Quanto maior esse número, mais financeiramente confiável a pessoa é considerada ser. Logo, um `Score de crédito` de 1 é o maior uma vez que estamos escalonando os valores entre 0 e 1

In [9]:

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)
```

9. Treinando os dados em um modelo de regressão logística.

Essencialmente, prever se um pedido de crédito será aprovado ou não é uma tarefa de classificação. De acordo com a UCI, nosso dataset contém mais instâncias que correspondem a "Negado" ("Denied") do que "Aprovado" ("Approved"). Especificamente, de 690 instancias, existem 383 (55.5%) pedidos que foram negados e 307 (44.5%) pedidos foram aprovados.

Isso nos dá um benchmark. Um bom modelo de machine learning deve ser capaz de prever corretamente o status dos clientes de acordo com essas estatísticas.

Quais modelo devemos selecionar? A pergunta deve ser: *quais características que afetam a decisão de aprovação de crédito são correlacionadas uma com a outra?* Apesar de podermos medir a correlação, isso está fora do escopo deste notebook, portanto confiaremos que elas estão de fato correlacionadas por enquanto. Por causa dessa correlação, aproveitaremos do fato de que modelos lineares performam bem nesses casos. Vamos começar com um modelo de Regressão Logística.

In [10]:

```python
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(rescaledX_train, y_train)
```

Out[10]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

10. Fazendo previsões e avaliando performance

Mas quão bom é o modelo?

Agora vamos avaliar o modelo de classificação no conjunto de teste quanto a sua Acurácia (classification accuracy). Após realizarmos a previsão, também olharemos para o seu f1-score Também olharemos para a sua matriz de confusão. Também é importante vermos se nosso modelo é capaz de prever o status de aprovação como negados e que de fato foram originalmente negados. Se nosso modelo não estiver perfomando bem neste aspecto, ele pode acabar aprovando pedidos que deveriam ter sido negados. A matriz de confusão nos ajudará a avaliar a performance do modelo nesses aspectos.

In [11]:

```python
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.metrics import classification_report

print("Acurácia do classificador: ", logreg.score(rescaledX_test, y_test))

y_pred = logreg.predict(rescaledX_test)

print("F1-Score: ", f1_score(y_test, y_pred))

confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
Acurácia do classificador:  0.8421052631578947
F1-Score:  0.8448275862068965
              precision    recall  f1-score   support

         0.0       0.78      0.91      0.84       103
         1.0       0.92      0.78      0.84       125

    accuracy                           0.84       228
   macro avg       0.85      0.85      0.84       228
weighted avg       0.85      0.84      0.84       228
```

11. Grid Search - melhorando a performance do modelo

O modelo foi capaz de prever com uma acurácia de 84%. Mas acurácia não é uma medida confiável de desempenho.

Na matriz de confusão, o primeiro elemento da primeira linha representa os Verdadeiros Positivos (Pode parecer confuso, mas aqui, "positivo" significa que o resultado é 1, ou seja que o pedido de crédito foi **negado**), sendo o numero de previsões corretas em que o modelo previu como pedido **negado**. E o segundo elemento da segunda linha são os Verdadeiros Negativos, ou seja, o numero de previsões corretas para pedido **aprovado**.

Vejamos se conseguimos melhorar esses resultados. Podemos realizar um grid search nos hiperparâmetros do modelo para melhorar a habilidade de prever aprovações de pedidos de crédito.

A implementação do scikit-learn para regressões logísticas consiste de diferentes hiperparâmetros, mas vamos buscar a melhor configuração para os seguintes:

- tol
- max_iter
- C

In [12]:

```python
from sklearn.model_selection import GridSearchCV

# Define os valores a serem testados em tol e max_iter
tol = [0.01,0.001,0.0001, 1, 10, 100]
max_iter = [100,150,200]
C = np.linspace(1, 100 , 5)

# Cria um dicionario com os valores a serem testados
param_grid = {"tol" : tol, "max_iter" : max_iter, "C": C}
```

12. Sintonia de hiperparâmetros e validação cruzada

Definimos o *grid* de hiperparâmetros e os convertemos para um único dicionário o qual `GridSearchCV()` receberá como um dos seus parâmetros. Agora, iniciaremos o *grid search* para ver quais valores performam melhor.

Criaremos uma instancia de `GridSearchCV()` com o nosso modelo `logreg` com todos os nossos dados. Ao invés de passar treino e teste separadamente, vamos oferecer todo o `X` (versão escalonada) e o `y`. Também vamos instruir o `GridSearchCV()` para realizar cross-validation (validação cruzada) em dez *folds* ou dobras.

Terminaremos este notebook armazenando o melhor score obtido, assim como os melhores parâmetros.

Na construção deste classificador, abordamos algumas das etapas mais conhecidade de pré-processamento como **escalonamento**, **label encoding**, e **imputação de valores faltantes**. Terminamos com um modelo de **machine learning** para prever se um pedido de crédito será aprovado ou não com base nos dados do cliente.

In [14]:

```python
grid_model = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=10, scoring='f1')

rescaledX = scaler.fit_transform(X)
grid_model_result = grid_model.fit(rescaledX, y)

scorer = grid_model_result.scorer_
print('Scorer: ', scorer)
best_score, best_params= grid_model_result.best_score_, grid_model_result.best_params_
print("Melhor score  %f usando %s" % (best_score, best_params))
```

```
Scorer:  make_scorer(f1_score, average=binary)
Melhor score  0.829388 usando {'C': 1.0, 'max_iter': 100, 'tol': 10}
```