



MBA-USP Data Science and Analytics

Linguagem Python



Laboratório Experimental de Análise de Dados

Prof. Antonio Geraldo da Rocha Vidal

2021

Sumário

Laboratório de Experimental de Análise de Dados	4
Google Colab	4
Introdução	4
Notebook Colab.....	4
Células de Código	6
Iniciando com o Colab Notebook	6
Usando Python no Google Colab	8
Análise de Dados com Python.....	10
Introdução.....	10
Análise Exploratória de Dados Usando Pandas.....	11
Introdução a Séries e Dataframes Pandas	11
Problema de Prático de Previsão de Empréstimo.....	11
Preparando o Colab.....	12
Importando Bibliotecas e Dados	13
Rápida Exploração dos Dados	14
Análise da Distribuição	15
Análise de Variáveis Categóricas.....	17
Tratamento de Dados com Pandas	20
Introdução.....	20
Verifique os Valores Ausentes no Conjunto de Dados.....	20
Tratamento de Valores Ausentes no Valor do Empréstimo	21
Tratando Valores Extremos.....	23
Continue a Explorar os Dados como Exercício	24
Construção de um Modelo Preditivo	24
Introdução.....	24
Regressão Logística	27
Árvore de Decisão	28
Floresta Aleatória	29
Etapa Final: Relatório de Elaboração do Laboratório	31
Conclusões Finais	31
Referências.....	31
Bibliografia	31
Vídeo Aulas.....	32

Você deve entregar um relatório com os resultados das etapas elaboradas neste laboratório no **alunoWeb**, para formatá-lo siga estas orientações:

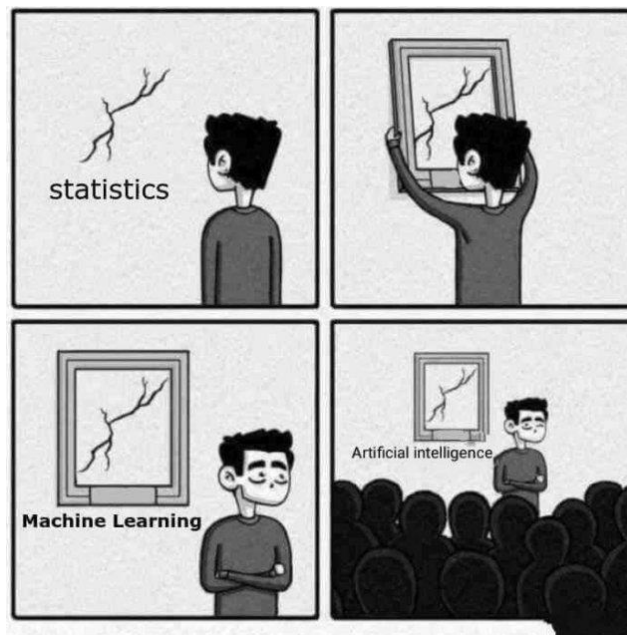
1. Crie um documento Word e identifique-o com o nome do laboratório, data de elaboração e o seu nome ou da dupla (ou trio) que o elaborou;
2. Crie um tópico para cada resultado que você considerar relevante (manipulação de dados ou resultado de algum processamento) identificando-o com um título e uma breve explicação. Os resultados podem ser imagens de gráficos gerados ou de listas de valores ou dados de resultados obtidos. Não devem ser incluídos os *scripts* ou instruções de processamento utilizados, inclua apenas os resultados que você considerar relevantes.
3. No final do relatório crie um último tópico denominado “Conclusões” e elabore comentários, sugestões e conclusões sobre o que você pode aprender com a elaboração deste laboratório.

Esta apostila introdutória pode conter erros, falhas ou imprecisões. Se você identificar algum problema por favor informe através do e-mail vidal@usp.br para que a correção possa ser providenciada.

Esta apostila não é autoral, tem objetivo estritamente didático como material de apoio para a disciplina. Foi desenvolvida através da compilação dos diversos textos e materiais citados na bibliografia.



Obrigado!



Marketing Inteligente

Laboratório de Experimental de Análise de Dados

Google Colab

Introdução

O *Google Colaboratory*, ou apenas "Colab" para abreviar (<https://colab.research.google.com/>), é baseado no projeto Jupyter e permite que você escreva e execute a linguagem Python diretamente em seu navegador Web, com a vantagem de:

- Não ser necessário instalar o Python no seu equipamento, pois ele estará disponível na nuvem hospedado nos servidores do Google;
- Nenhuma necessidade de configuração do Python, pois o Google faz isso para você;
- Acesso "gratuito" a todo o potencial de GPUs do Google para machine learning;
- Compartilhamento fácil do seu código através de *notebooks* do Google Drive.

Se você for um estudante, um cientista de dados ou um pesquisador de IA, o Colab pode tornar seu trabalho mais fácil. Assista introdução ao Colab em <https://www.youtube.com/watch?v=inN8seMm7UI> para ouvir mais, ou apenas começar a utilizá-lo.

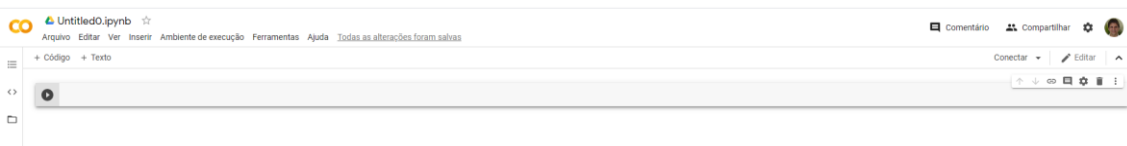
Notebook Colab

O Colab trabalha com um ambiente interativo chamado **Colab Notebook**, que permite escrever e executar código. Por exemplo, a seguir está uma célula de código com um script em Python que calcula um valor, armazena-o em uma variável e imprime o resultado:

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

86400

Para executar o código na célula acima, selecione-o com um clique e pressione o **botão play** à esquerda do código ou use o atalho de teclado [Ctrl + Enter]. Para editar o código, basta clicar na célula e começar a editar.



As variáveis que você define em uma célula podem ser usadas posteriormente em outras células:

```
[ ] seconds_in_a_week = 7 * segundos_in_a_day
seconds_in_a_week
```

604800

Os notebooks do Colab permitem combinar código executável e *rich text* em um único documento, junto com imagens, gráficos, HTML, LaTeX e muito mais. Quando você cria seus próprios notebooks do Colab, eles são armazenados na sua conta do [Google Drive](#).

Google Drive é um serviço de armazenamento e sincronização de arquivos que foi apresentado pelo Google em 24 de abril de 2012. O Google Drive abriga o Google Docs, um leque de aplicações de produtividade, que oferece a edição de documentos, folhas de cálculo, apresentações, e muito mais.

O Google Drive também abriga o Google Colab e para utilizar o Colab você precisa ter uma conta Google e estar familiarizado com o Google Drive. Espero que esteja, pois não abordaremos este assunto neste laboratório, mas você pode consultar a documentação disponibilizada pelo [Google a respeito](#).

Você pode compartilhar facilmente seus notebooks Colab com colegas de trabalho ou amigos, permitindo que eles comentem ou editem seus notebooks. Para criar um notebook Colab, você pode usar o menu Arquivo acima ou usar o seguinte link: crie um novo notebook Colab.

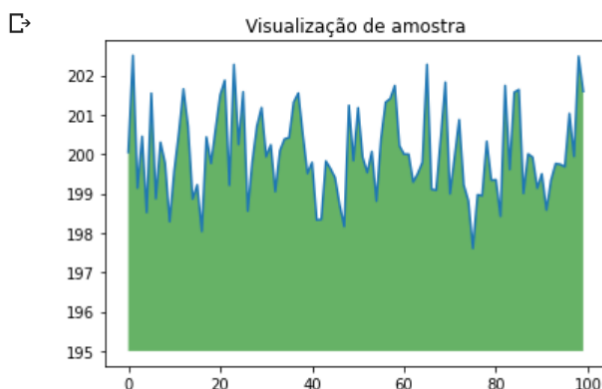
O Colab está integrado ao Google Drive. Permite compartilhar, comentar e colaborar no mesmo documento com várias pessoas:

- O botão COMPARTILHAR (canto superior direito da barra de ferramentas) permite compartilhar o bloco de notas e controlar as permissões definidas nele.
- Arquivo-> Criar uma cópia cria uma cópia do notebook no Drive.
- Arquivo-> Salvar salva o arquivo no Drive. Arquivo-> Salvar e ponto de verificação fixa a versão para que ela não seja excluída do histórico de revisões.
- Arquivo-> Histórico de revisões mostra o histórico de revisões do notebook.

Os notebooks da Colab são notebooks Jupyter hospedados pelo Colab. Se você quiser saber mais sobre o projeto Jupyter, consulte [jupyter.org](#).

Com o Colab, você pode aproveitar todo o poder das bibliotecas Python para analisar e visualizar dados. A célula de código abaixo usa a biblioteca **numpy** para gerar alguns dados aleatórios e depois usa a biblioteca **matplotlib** para visualizá-los. Para editar o código, basta clicar na célula e começar a digitar.

```
import numpy as np
from matplotlib import pyplot as plt
ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]
plt.plot(x,ys,'-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor = 'g', alpha=0.6)
plt.title("Visualização de amostra")
plt.show()
```



Você pode importar seus próprios dados para os notebooks Colab a partir da sua conta do Google Drive, inclusive de planilhas, do Github e de muitas outras fontes de dados. Para saber mais sobre a importação de dados e como o Colab pode ser usado para ciência de dados consulte a [Ajuda do Colab](#) sobre este tema.

Com o Colab, você pode importar um conjunto de dados de imagens, treinar um classificador de imagens e avaliar o modelo, tudo com algumas poucas linhas de código. Os notebooks Colab executam códigos nos servidores em nuvem do Google, o que significa que você pode aproveitar o poder do hardware do Google, incluindo GPUs e TPUs, independentemente do poder da sua máquina. Tudo que você precisa é de um navegador e um bom acesso à Internet.

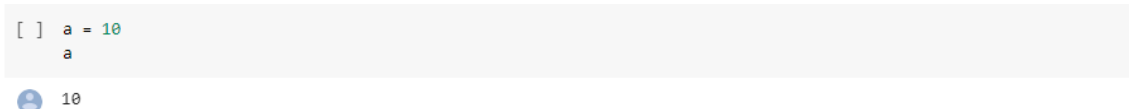
Um notebook é uma lista de células, como se fossem as linhas de um caderno de anotações ou escolar. As células contêm texto explicativo ou código executável e sua saída. Clique em uma célula para selecioná-la.

Células de Código

Abaixo está uma célula de código. Depois que o botão da barra de ferramentas indicar CONECTADO, clique na célula para selecioná-la e execute o conteúdo das seguintes maneiras:

- Clique no ícone **Play** na extremidade esquerda da célula;
- Digite **[Ctrl + Enter]** para executar a célula atual;
- Digite **[Shift + Enter]** para executar a célula e mover o foco para a próxima célula (adicionando uma se não houver); ou
- Digite **[Alt + Enter]** para executar a célula e inserir uma nova célula de código imediatamente abaixo dela.

Além destas, existem opções adicionais para executar algumas ou todas as células no menu Ambiente de execução.



Iniciando com o Colab Notebook

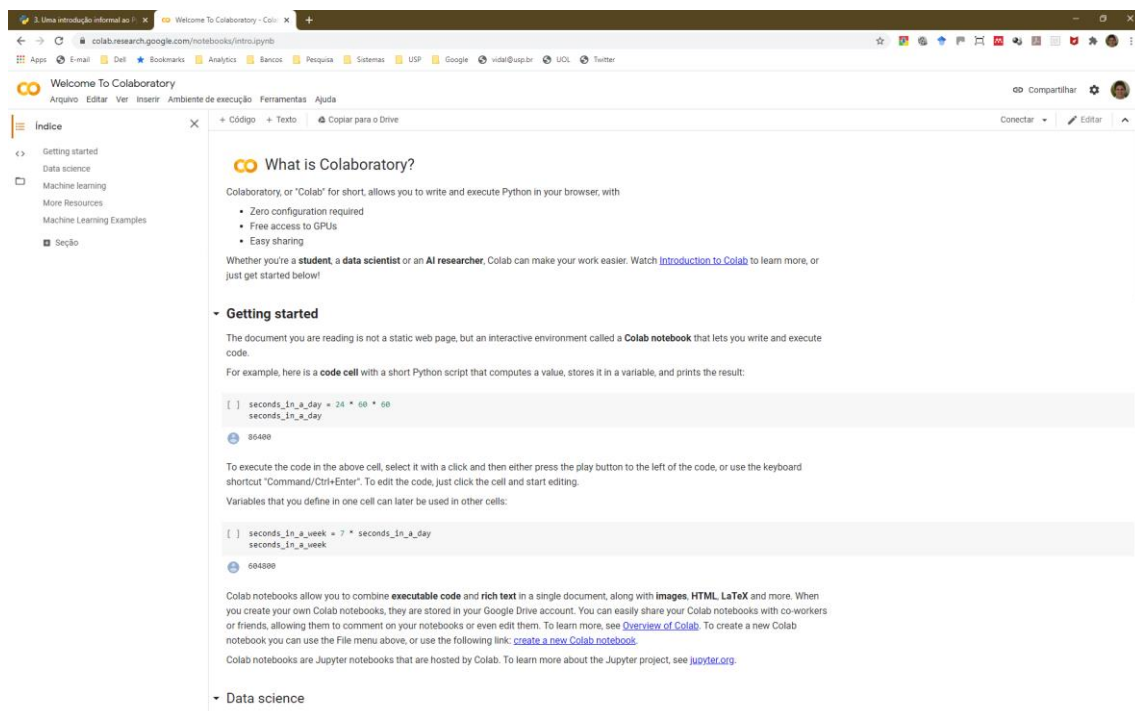
Para iniciar a utilização do Python através do projeto Google Colaboratory você deve ter uma conta Google e configurado o Google Drive no seu equipamento. Caso ainda não tenha, por favor providencie, pois são totalmente gratuitos e neste laboratório utilizaremos os recursos disponibilizados pelo Google para trabalhar com o Python.

Para acessar o Google Colab utilize o seguinte link: <https://colab.research.google.com/>.

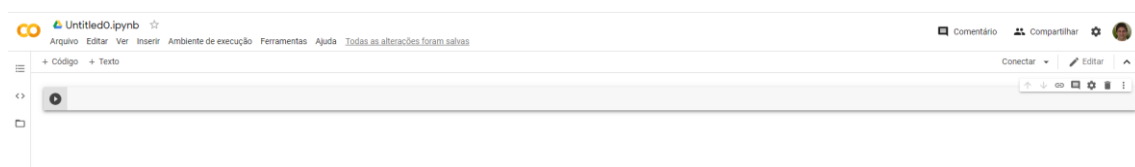
Caso você já tenha utilizado o Colab será apresentada a janela ilustrada a seguir, através da qual você poderá continuar um trabalho (notebook) anterior ou criar um notebook novo.

Exemplos	Recente	Google Drive	GitHub	Upload
Filtrar notebooks				
Título		Primeiro acesso	Aberto pela última vez	
Welcome To Colaboratory		há 1 hora	há 0 minuto	
Cópia de Untitled0.ipynb		há 30 minutos	há 30 minutos	
PECE.ipynb		há 58 minutos	há 58 minutos	
Overview of Colaboratory Features		28 de abr. de 2020	há 1 hora	
Untitled0.ipynb		há 1 hora	há 1 hora	
NOVO NOTEBOOK CANCELAR				

Supondo que você esteja utilizando o Google Colab pela primeira vez, verá a tela apresentada a seguir, através da qual você poderá criar seu primeiro notebook. Para isso, no menu Arquivo escolha a opção Novo Notebook.



Após escolher Arquivo e Novo Notebook, seu notebook será criado e apresentado como ilustra a figura a seguir. Note que ele ainda está com um nome genérico **Untitled0.ipynb**, portanto, você não deve se esquecer de renomeá-lo quando for salvá-lo através do menu Arquivo/File.



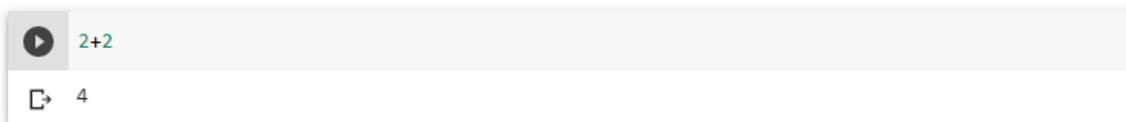
Em um Colab Notebook você pode digitar diretamente o código Python nas “linhas do seu caderno” (células) e depois clicar o botão Play ou Executar para obter diretamente os resultados. Essencialmente neste laboratório utilizaremos o Python desta forma, pois é, de longe, muito vantajosa para um aprendizado inicial em *Analytics* com Python.

Você pode estar se perguntando “Onde está o Python? Quem está processando o código do meu Notebook? Onde o meu Notebook está sendo salvo? Onde estão as bibliotecas Python para Data Science? E mais um monte de outros Quem, Onde e Como? A resposta será sempre a mesma: Google!!

Usando Python no Google Colab

Vamos iniciar experimentando os comandos mais simples de Python que podemos imaginar. Inicie o seu Colab Notebook e aguarde a apresentação da linha ou célula para digitação do seu primeiro código Python, como ilustrado na figura anterior.

Digite o seguinte código e clique o botão Play ou Executar à esquerda para ver o resultado, conforme ilustrado a seguir.



```
2+2
```

```
4
```

Agora digite o cálculo abaixo, mas em vez de clicar o botão Play, execute teclando [Shift + Enter]. Além de executar o código [Shift + Enter] automaticamente abrirá a próxima linha do seu Notebook, conforme ilustrado a seguir, de forma que você possa continuar codificando.



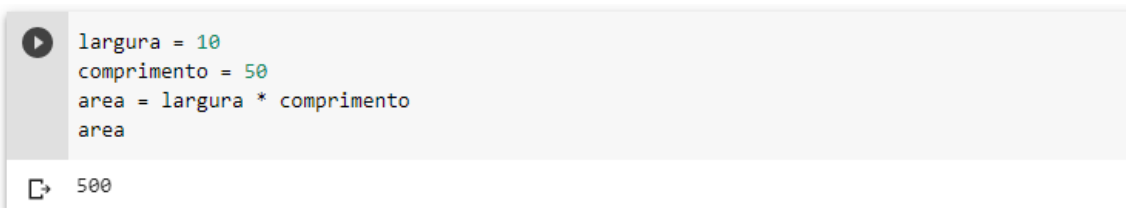
```
[10] 50 - 5*6
```

```
20
```

Todas as linhas e resultados do seu notebook estão sendo gravados pelo Google Colab, mas a partir de agora será uma boa providência nomear o seu notebook. Para isso, no menu Arquivo escolha a opção Renomear e perceba que o Colab editará o nome genérico inicial do seu notebook permitindo que você defina um nome mais apropriado para ele. No meu caso escolhi PECE.ipynb. A extensão .ipynb, como você já deve ter percebido significa *interactive python notebook*.

O sinal de igual ('=') é usado para atribuir um valor a uma variável. Depois de uma atribuição, nenhum resultado é exibido antes do próximo comando. Por exemplo, execute esta nova linha no seu notebook.

Atenção, após digitar as linhas você deve clicar o botão Play ou teclar [Ctrl + Enter] para executá-las com o cursor posicionado na última posição, caso contrário o seu código não funcionará.



```
largura = 10  
comprimento = 50  
area = largura * comprimento  
area
```

```
500
```

Agora vamos simular um erro para ver como o Python reage no Colab, pois ao longo do seu aprendizado e mesmo depois quando você já for um expert em Python, erros serão

comuns e precisarão ser corrigidos. No exemplo a seguir utilizamos uma variável ainda não definida para realizar um cálculo. Vamos ver o que o Python e o Colab nos dizem sobre isso.



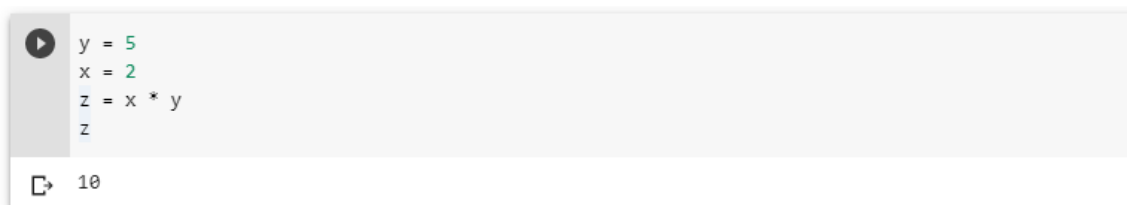
```
y = 5
x = 2
z = x * a
z
```

NameError Traceback (most recent call last)
<ipython-input-20-4311bac23b92> in <module>()
 1 y = 5
 2 x = 2
----> 3 z = x * a
 4 z

NameError: name 'a' is not defined

SEARCH STACK OVERFLOW

Bacana, não? Quase que o Python poderia corrigir o erro automaticamente para nós, pois ele identificou o erro e mostrou onde ocorreu. Basta então você corrigir, editando a linha apontada com o código correto e executando-a novamente.



```
y = 5
x = 2
z = x * y
z
```

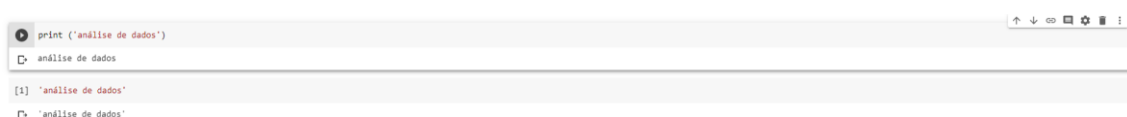
10

Há suporte completo para ponto flutuante (*float*); operadores com operandos de diferentes tipos convertem o inteiro para ponto flutuante.

Além de int e float, o Python suporta outros tipos de números, tais como Decimal e Fraction. O Python também possui suporte nativo a complex numbers 1, e usa os sufixos j ou J para indicar a parte imaginária (por exemplo, 3+5j).

Além de números, o Python também pode manipular *strings* (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas.

Na interpretação interativa, a string de saída é delimitada com aspas e caracteres especiais são escapados com barras invertidas. Embora isso possa às vezes parecer diferente da entrada (as aspas podem mudar), as duas strings são equivalentes. A string é delimitada com aspas duplas se a string contiver uma única aspa simples e nenhuma aspa dupla, caso contrário, ela é delimitada com aspas simples. A função **print()** produz uma saída mais legível, ao omitir as aspas e ao imprimir caracteres escapados e especiais:



```
print('análise de dados')
```

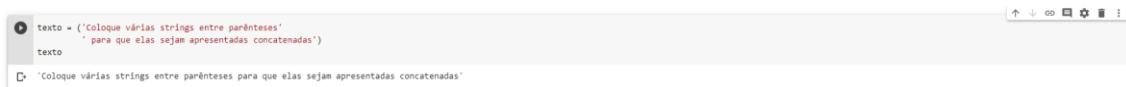
análise de dados

[1] 'análise de dados'

'análise de dados'

As strings literais podem abranger várias linhas. Uma maneira é usar as aspas triplas: `"""..."""` ou `'''...'''`. O fim das linhas é incluído automaticamente na string, mas é possível evitar isso adicionando uma `\` no final.

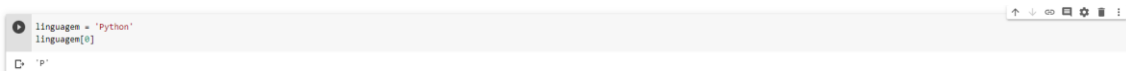
Strings podem ser concatenadas (coladas) com o operador `+`, e repetidas com `*`. Duas ou mais strings literais (ou seja, entre aspas) ao lado da outra são automaticamente concatenadas.



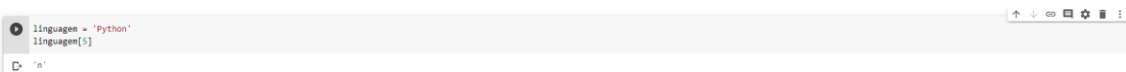
```
texto = ('Coloque várias strings entre parênteses'
        ' para que elas sejam apresentadas concatenadas')
texto
'Coloque várias strings entre parênteses para que elas sejam apresentadas concatenadas'
```

Esse recurso é particularmente útil quando você quer quebrar strings longas.

As strings podem ser indexadas (subscritas), com o primeiro caractere como índice 0. Não existe um tipo específico para caracteres; um caractere é simplesmente uma string cujo tamanho é 1. Índices também podem ser números negativos para iniciar a contagem pela direita.

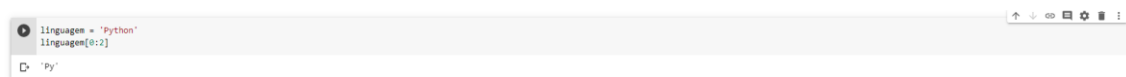


```
linguagem = 'Python'
linguagem[0]
'P'
```



```
linguagem = 'Python'
linguagem[5]
'n'
```

Além da indexação, o fatiamento também é permitido. Embora a indexação seja usada para obter caracteres individuais, fatiar permite que você obtenha uma substring:



```
linguagem = 'Python'
linguagem[0:2]
'Py'
```

Observe como o início sempre está incluído, e o fim sempre é excluído. Além disso, os índices do fatiamento possuem padrões úteis; um primeiro índice omitido padrão é zero, um segundo índice omitido é por padrão o tamanho da string sendo fatiada.

As strings do Python não podem ser alteradas — uma string é imutável. Portanto, atribuir a uma posição indexada na sequência resulta em um erro. Se você precisar de uma string diferente, deverá criar uma nova. A função `len()` devolve o comprimento de uma string.

Análise de Dados com Python

Introdução

Neste último laboratório, considerando que agora você esteja familiarizado com os fundamentos do [Python](#) e das bibliotecas adicionais que ele permite agregar, vamos elaborar alguns exemplos sofisticados de análise de dados. As técnicas e algoritmos de aprendizado de máquina (*machine learning*) que iremos utilizar não serão abordadas em detalhes neste laboratório, pois serão objeto de disciplinas do curso. Entretanto, considere este laboratório como uma demonstração para você poder sentir o "poder do Python" para aplicações de *Data Science e Analytics*.

Neste processo, usaremos algumas bibliotecas poderosas e iremos nos deparar com o próximo nível de estruturas de dados. Vamos conduzir esta análise de dados experimental através das 3 fases principais:

1. **Exploração de Dados** – descobrir mais sobre os dados que temos;
2. **Tratamento de Dados** – limpar os dados e prepará-los para torná-los melhores para a análise de dados;
3. **Modelagem Preditiva** – fazer funcionar os algoritmos de machine learning reais e se “impressionar”.

Análise Exploratória de Dados Usando Pandas

Para explorar os nossos dados, como se apenas o Python não fosse suficiente, vamos utilizar a biblioteca Pandas.

Pandas é uma das bibliotecas de análise de dados mais úteis em Python. Elas têm sido fundamentais para aumentar o uso de Python na comunidade científica de dados. Vamos usar Pandas para ler um conjunto de dados, executar análise exploratória sobre estes dados e construir o nosso primeiro modelo analítico básico de classificação para resolver um problema preditivo de análise de crédito em uma instituição financeira.

Antes de carregar os dados, vamos recordar para compreender melhor as duas principais estruturas de dados em **Pandas** – Séries e Dataframes.

Introdução a Séries e Dataframes Pandas

Um objeto **Series** de Pandas pode ser entendido como uma matriz unidimensional rotulada e indexada. Pode-se acessar elementos individuais desta série através destes rótulos.

Um objeto **Dataframe** de Pandas é semelhante a uma planilha do Excel – você tem nomes de colunas referentes a colunas e você tem linhas, que podem ser acessadas com o uso de nomes ou do número de cada linha. A diferença essencial é que os nomes das colunas e os números das linhas são conhecidos como colunas e índices de linha, no caso dos Dataframes. Um Dataframe também pode ser semelhante à uma tabela de um banco de dados relacional (SQL). Numa tabela de banco de dados você tem colunas nomeadas, que podem armazenar diferentes tipos de dados, e linhas numeradas, o que a torna muito semelhante à um Dataframe de Pandas.

Séries e Dataframes formam o núcleo dos modelos de dados do Pandas em Python. Os conjuntos de dados que você precisa analisar são lidos primeiro para os Dataframes e, em seguida, várias operações (por exemplo: agrupamento, agregações, estatísticas etc.) podem ser aplicadas muito facilmente às suas colunas e linhas do Dataframe.

Problema de Prático de Previsão de Empréstimo

Para iniciarmos você deve fazer o *download* dos arquivos Arquivo_Treino.csv e Arquivo_Testes.csv, que contém dados sobre pessoas que pediram empréstimo à instituição financeira do nosso problema. Estes arquivos deverão estar disponíveis no site da disciplina, caso não os encontre, por favor, solicite orientação para o professor. Estas são as descrições das variáveis (ou colunas) contidas nestes arquivos; algumas são numéricas (contínuas) e outras categóricas (discretas).

Como você poderá observar o arquivo contém um conjunto de dados de clientes de uma instituição financeira que solicitaram empréstimos (loan) para aquisição de imóveis. Alguns foram aprovados (loan_status = Y) outros não (loan_status = N).

Variável	Descrição
Loan_ID	ID único do empréstimo (numérico)
Gender	Masculino/Feminino (Male/Female)
Married	Casado - sim ou não (Y/N)
Dependents	Número de dependentes
Education	Escolaridade (Graduate / Under Graduate)
Self_Employed	Auto empregado - sim ou não (Y/N)
ApplicantIncome	Renda do aplicante
CoapplicantIncome	Renda do fiador
LoanAmount	Montante do empréstimo, em milhares
Loan_Amount_Term	Prazo do empréstimo, em meses
Credit_History	Histórico de crédito corresponde aos critérios (1 ou 0)
Property_Area	Localização da propriedade (Urban/Semi Urban/ Rural)
Loan_Status	Empréstimo aprovado - sim ou não (Y/N)

Com base nestes dados, neste laboratório vamos treinar algoritmos que utilizam aprendizagem de máquina do tipo supervisionada (*supervised machine learning*) de forma a construir um modelo analítico preditivo que consiga prever para a instituição financeira se um novo cliente deveria ter o seu empréstimo aprovado ou não.

Para isso precisamos de pelo menos dos dois conjuntos de dados que baixamos. Os dados do **Arquivo_Treino.csv** (dados separados por ponto-e-vírgula para compatibilidade com o Excel em português) serão utilizados para treinar os algoritmos, ou seja, para que a máquina aprenda a classificar os clientes da instituição com estes dados. Os dados do arquivo **Arquivo_Teste.csv** (separados por ponto-e-vírgula para compatibilidade com o Excel em português) serão utilizados para testar os algoritmos de forma a verificar se o aprendizado da máquina foi satisfatório e o modelo preditivo é suficientemente genérico e preciso para sua aplicação em casos reais de análise de crédito da instituição financeira em questão.

Preparando o Colab

Para começar, inicie o **Google Colab**:

<https://colab.research.google.com/notebooks/intro.ipynb#recent=true>

Isso abre o **Colab Notebook** em ambiente Python, que tem muitas bibliotecas úteis já importadas. Além disso, você será capaz de gerar gráficos pela linha de comando do notebook, o que torna este um bom ambiente para análise de dados interativos.

Para poder ler os arquivos que contém os dados que iremos utilizar temos que seguir os seguintes passos:

1. Copiar os arquivos **Arquivo_Treino.csv** e **Arquivo_Teste.csv** para o seu Google Drive; pode ser em qualquer pasta, mas sugiro que você crie uma pasta específica, por exemplo denominada PECE, e copie estes arquivos para lá, assim será mais fácil localizá-los quando necessário.

Portanto, armazene os arquivos de dados indicados no seguinte local:

- a. Dados para Treino: Google Drive:/PECE/Arquivo_Treino.csv

b. Dados para Teste: Google Drive:/PECE/Arquivo_Testes.csv

2. Montar o seu Google Drive no **Colab** para que você tenha acesso a ele através do Colab. Fazemos isso executando as seguintes instruções no notebook:

```
from google.colab import drive
drive.mount('/content/drive')
```

O **Colab** irá responder a você da seguinte forma:

```
Go to this URL in a browser:
https://accounts.google.com/o/oauth2/auth?client\_id=chave
```

```
Enter your authorization code: cole o código obtido aqui
```

```
Mounted at /content/drive
```

Para se certificar que o seu Google Drive foi acessado pelo **Colab**, liste o seu conteúdo executando a instrução a seguir:

```
!ls "/content/drive/My Drive"
!ls "/content/drive/My Drive/PECE"
# Este último verifica se os arquivos que utilizaremos
# estão lá
```

Você também poderá consultar o Google Drive montado no Colab clicando o botão “Arquivos” (com o ícone de pasta) no canto esquerdo, para que seja apresentada a janela de gerenciamento de arquivos conforme ilustrado na figura apresentada a seguir.

Importando Bibliotecas e Dados

Abaixo estão as bibliotecas Python que usaremos neste laboratório experimental de análise preditiva de dados:

1. numpy
2. matplotlib
3. pandas

Vale ressaltar que não é obrigatório importar **matplotlib** e **numpy** no ambiente **Colab**. Mantivemos a importação no código para o caso de você usar em um ambiente diferente. Note que estamos utilizando os aliases clássicos para cada uma das bibliotecas.

Depois de importar as bibliotecas, leia o conjunto de dados usando a função **read_csv()**. O código até esta fase fica assim:

```
import pandas as pd
import numpy as np
```

```
import matplotlib as plt
```

```
#Lendo o conjunto de dados para um dataframe (df) de Pandas
df = pd.read_csv("/content/drive/My
Drive/PECE/Arquivo_Treino.csv", set=';')
```

Rápida Exploração dos Dados

Depois de ler o conjunto de dados, você pode dar uma olhada nas linhas iniciais usando a função **head()** do Dataframe **df** que foi criado a partir da instrução **pd.read_csv()**.

```
df.head(10)
```

Devem ser apresentadas as 10 primeiras linhas. Alternativamente, você também pode consultar mais linhas do conjunto de dados aumentando o valor do parâmetro. Na figura a seguir você pode ver:

- As instruções para montar o Google Drive no Colab;
- O gerenciador de arquivos do Google Drive no Colab;
- As instruções para importar as bibliotecas Python;
- A instrução para ler o conjunto de dados para um Dataframe Pandas;
- As dez primeiras linhas do Dataframe **df** apresentando os dados lidos.

The screenshot shows a Google Colab notebook interface. On the left, the 'Arquivos' (Files) pane shows the file structure, including 'drive' and 'PECE' folders. The main code area contains the following steps:

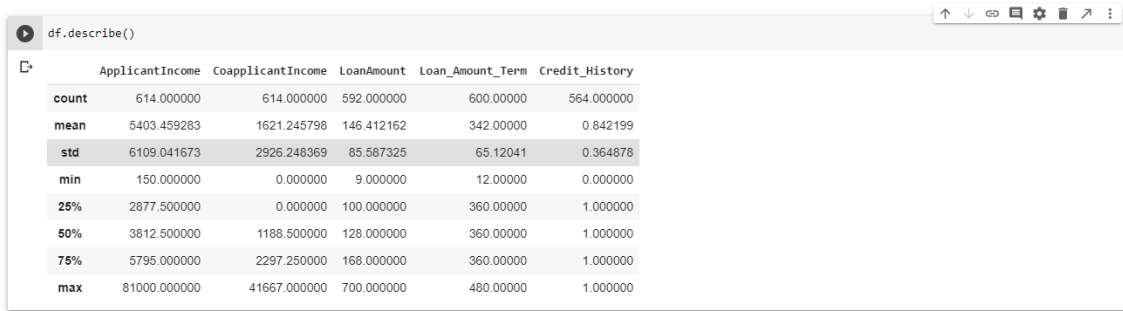
- Mounting the Google Drive: `from google.colab import drive; drive.mount('/content/drive')`
- Importing libraries: `import pandas as pd; import numpy as np; import matplotlib as plt`
- Loading the CSV file: `df = pd.read_csv('/content/drive/My Drive/PECE/ArquivoTreino.csv')`
- Displaying the first 10 rows: `df.head(10)`

The output shows the first 10 rows of the DataFrame, with columns: Loan_ID, Gender, Married, Dependents, Education, Self_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term, Credit_History, and Property_Area.

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urb
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Ru
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urb
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urb
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urb
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	1.0	Urb
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	1.0	Urb
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158.0	360.0	0.0	Semiurb
8	LP001018	Male	Yes	2	Graduate	No	4006	1526.0	168.0	360.0	1.0	Urb
9	LP001020	Male	Yes	1	Graduate	No	12841	10968.0	349.0	360.0	1.0	Semiurb

Em seguida, você pode olhar para um resumo das variáveis ou colunas numéricas do Dataframe usando a função **describe()**.

```
df.describe()
```



	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.000000	564.000000
mean	5403.459283	1621.245798	146.412162	342.000000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	3612.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	168.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

A função `describe()` mostra as principais estatísticas descritivas dos seus dados, entre elas, contagem, média, desvio padrão (STD), mínimo, quartis e máximo.

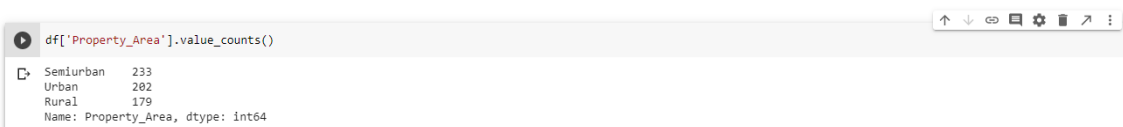
Aqui estão algumas descobertas importantes, você já pode fazer analisando a saída da função `describe()`:

- `LoanAmount` tem 22 valores faltantes (614 – 592).
- `Loan_Amount_Term` tem 14 valores faltantes (614 – 600).
- `Credit_History` tem 50 valores faltantes (614 – 564).
- Podemos também verificar que cerca de 84% dos clientes têm histórico de crédito. Como? A média da variável `Credit_History` é 0,84 (`Credit_History` tem um valor 1 para aqueles que têm um histórico de crédito e 0 em caso contrário).
- A distribuição da variável `ApplicantIncome` (renda do cliente) parece estar em linha com a expectativa. O mesmo ocorre com `CoapplicantIncome` (renda do fiador).

Note que podemos ter uma ideia de uma possível distorção nos dados comparando a média com a mediana, isto é, o valor de mediano de 50% dos clientes.

Para os valores não-numéricos (por exemplo, localização da propriedade, escolaridade etc.), podemos olhar para a distribuição de frequência para entender se elas fazem sentido ou não. A tabela de frequências pode ser exibida utilizando o seguinte comando:

```
df['Property_Area'].value_counts()
```



Semiurban	233
Urban	202
Rural	179

Name: Property_Area, dtype: int64

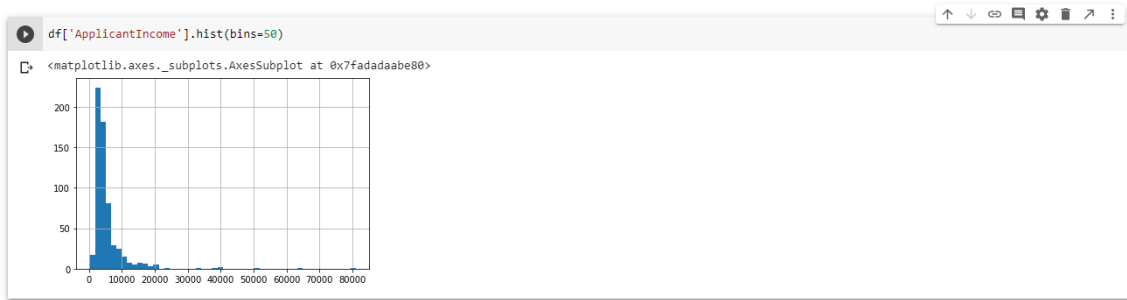
Da mesma forma, podemos olhar para valores exclusivos de histórico de crédito. Lembre-se que `dfname['column_name']` é uma técnica básica de indexação para acessar uma determinada coluna do Dataframe. Ela pode ser uma lista de colunas também.

Análise da Distribuição

Agora que estamos um pouco familiarizados com as características básicas dos dados, vamos procurar estudar a distribuição de diversas variáveis. Vamos começar com as variáveis numéricas – `ApplicantIncome` e `LoanAmount`.

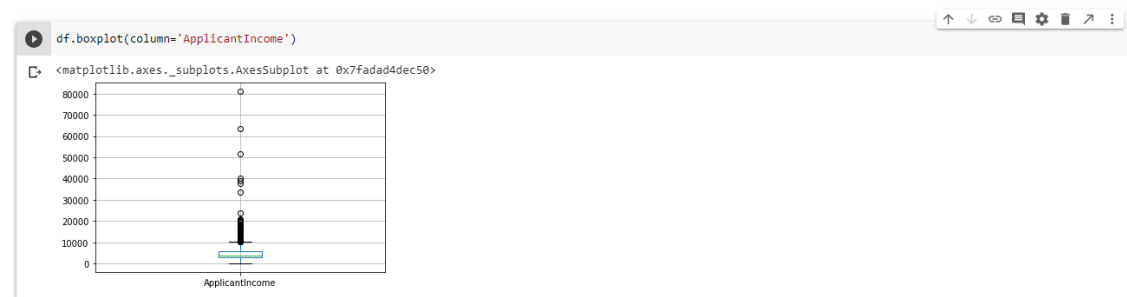
Vamos inicialmente plotar o histograma do `ApplicantIncome` usando o seguinte comando, separando em faixas (bins) de 50:

```
df['ApplicantIncome'].hist(bins=50)
```



Observamos pelo histograma que não há muitos valores extremos (*outliers*). Em seguida, vamos analisar pelo boxplot para compreender as distribuições. O boxplot para renda do cliente pode ser traçado por:

```
df.boxplot(column='ApplicantIncome')
```



O boxplot confirma a presença *outliers* (valores extremos). Isso pode ser atribuído à disparidade de renda na sociedade. Parte disso pode ser explicado, por exemplo, pelo fato de que estamos olhando para pessoas com diferentes níveis de escolaridade. Vamos segregá-los, portanto, por Educação, para verificar esta hipótese:

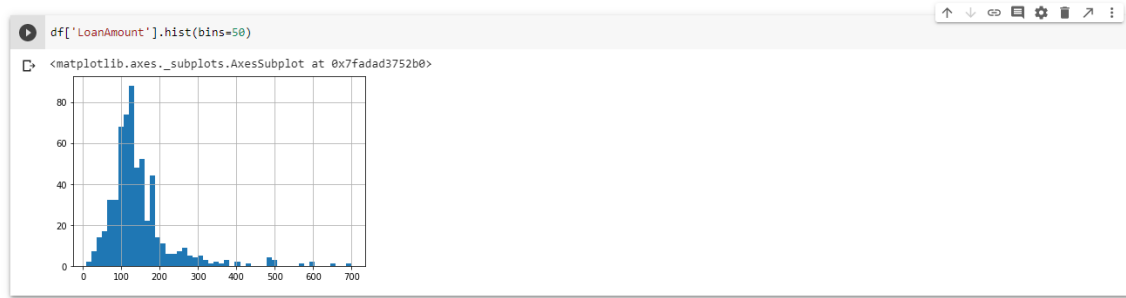
```
df.boxplot(column='ApplicantIncome', by = 'Education')
```



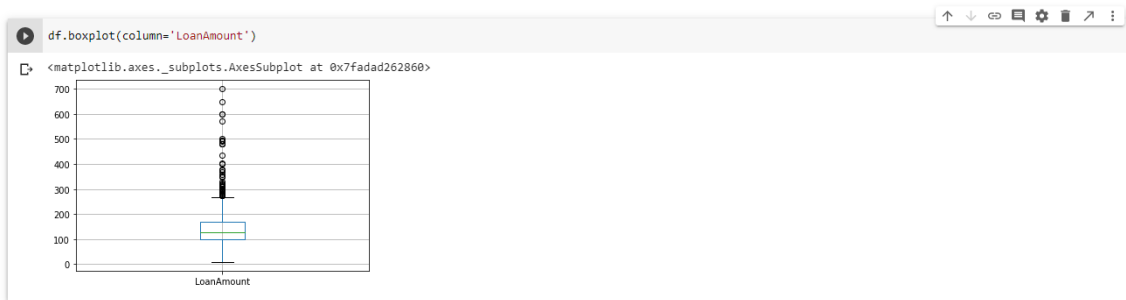
Podemos observar que não há uma diferença substancial entre a renda média de graduados e de não-graduados. Por outro lado, podemos observar que há um maior número de graduados com rendimentos muito elevados que parecem ser os *outliers*.

De forma equivalente, vamos agora analisar o histograma e o boxplot do valor do empréstimo usando os seguintes comandos:

```
df['LoanAmount'].hist(bins=50)
```

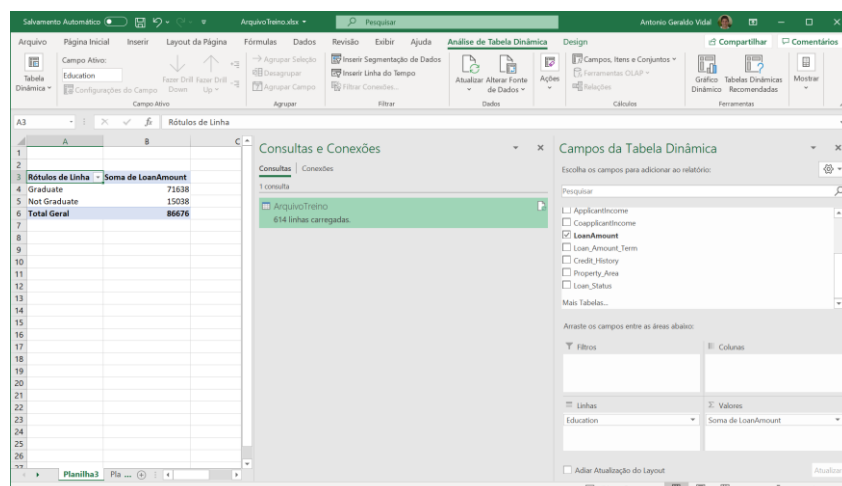
```
df.boxplot(column='LoanAmount')
```



Mais uma vez, podemos observar que há alguns valores extremos. Claramente, tanto ApplicantIncome e LoanAmount exigem uma certa quantidade de tratamento de dados (*'data munging'*). LoanAmount tem valores faltantes e valores extremos, enquanto ApplicantIncome tem alguns valores extremos, que exigem compreensão mais profunda. Vamos avaliar isto nas próximas etapas.

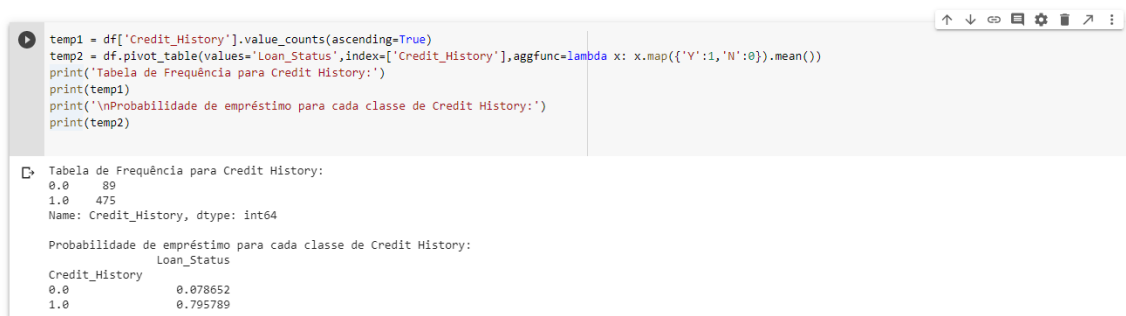
Análise de Variáveis Categóricas

Agora que já exploramos um pouco as distribuições das principais variáveis relacionadas ao problema, ApplicantIncome e LoanIncome, vamos explorar as variáveis categóricas com mais detalhes. Usaremos uma tabela estilo *pivot table* do Excel e tabulação cruzada. Por exemplo, vamos olhar para as chances de se conseguir um empréstimo com base no histórico de crédito. Isto pode ser conseguido no Microsoft Excel lendo o arquivo ArquivoTreino.csv e utilizando uma *pivot table* ou tabela dinâmica como exemplificado a seguir. Se você não tiver familiaridade com o Excel não se preocupe, não é necessário executar este passo, pois vamos executá-lo aqui aplicando algumas instruções em Python que abordamos nos laboratórios anteriores quando conhecemos a linguagem.



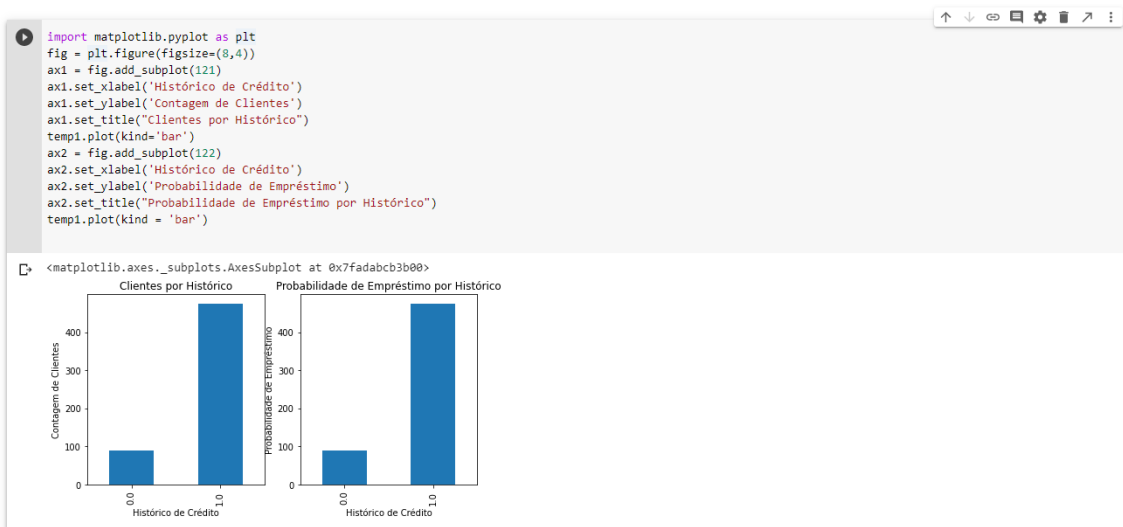
Vamos ver os passos necessários para gerar uma visão semelhante usando Python.

```
temp1 = df['Credit_History'].value_counts(ascending=True)
temp2 =
df.pivot_table(values='Loan_Status',index=['Credit_History'],aggfunc=lambda x: x.map({'Y':1, 'N':0}).mean())
print('Frequency Table for Credit History:')
print(temp1)
print('\nProbability of getting loan for each Credit
History class:')
print(temp2)
```



Agora, podemos observar que temos uma tabela dinâmica semelhante à do MS Excel. Isto pode ser plotado como um gráfico de barras usando a biblioteca **matplotlib** através do seguinte código:

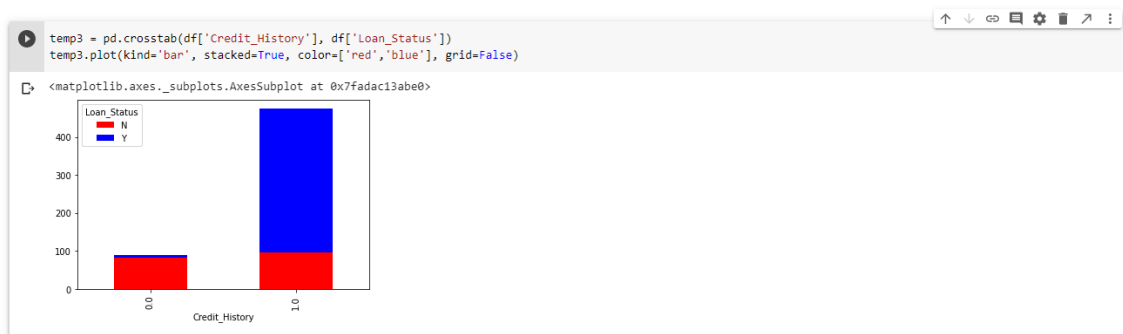
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)
ax1.set_xlabel('Histórico de Crédito')
ax1.set_ylabel('Contagem de Clientes')
ax1.set_title('Clientes por Histórico')
temp1.plot(kind='bar')
ax2 = fig.add_subplot(122)
ax2.set_xlabel('Histórico de Crédito')
ax2.set_ylabel('Probabilidade de Empréstimo')
ax2.set_title('Probabilidade de Empréstimo por Histórico')
temp2.plot(kind='bar')
```



Isso mostra que a chance de conseguir um empréstimo é muito maior se o requerente tiver um histórico de crédito válido. Para aprofundar a sua análise você pode plotar gráficos semelhantes por Estado Civil, Profissional Autônomo, Localização de Propriedade etc. se desejar.

Como alternativa, estes dois gráficos também podem ser visualizados combinados em um único gráfico empilhado.

```
temp3 = pd.crosstab(df['Credit_History'],
df['Loan_Status'])
temp3.plot(kind='bar', stacked=True, color=['red','blue'],
grid=False)
```



Você também pode adicionar o gênero na combinação (semelhante a tabela dinâmica em Excel):



Acabamos de criar dois critérios de classificação básicos aqui, um baseado no histórico de crédito e outro em 2 variáveis categóricas (incluindo gênero). Também começamos

a estudar como podemos fazer análise exploratória em Python utilizando Pandas. Você deve ter percebido que as bibliotecas Pandas e Matplotlib podem fornecer uma quantidade quase ilimitada de recursos para análise de conjuntos de dados, portanto, estamos apenas apresentando neste texto. A partir deste primeiro contato você poderá aprofundar o seu estudo, pois há farto material disponível na Web, para poder realizar suas análises com proficiência e profissionalismo.

Em seguida, vamos explorar as variáveis ApplicantIncome e LoanStatus e ainda, realizar algum tratamento nos dados para criar um conjunto de dados adequado para a aplicação de várias técnicas de modelagem que utilizam aprendizagem de máquina. Com este conhecimento você pode obter um outro conjunto de dados, relacionado a um outro problema de seu interesse, e fazer uma exploração de dados independente. Assim realmente você poderá verificar o que aprendeu.

Tratamento de Dados com Pandas

Introdução

Durante a nossa exploração dos dados, encontramos alguns problemas que precisam ser resolvidos antes que os dados estejam prontos para serem utilizados em algoritmos analíticos de dados. Esta atividade é normalmente referida como “*Data Munging*” ou Tratamento de Dados. Aqui estão alguns problemas que já percebemos com os dados que estamos manipulando:

1. Há valores faltando em algumas variáveis. Devemos estimar esses valores, dependendo da quantidade de valores faltantes e da importância esperada das variáveis para o problema que estivermos tratando.
2. Quando se observa as distribuições, vê-se que ApplicantIncome e LoanAmount parecem conter valores extremos em cada ponta. Embora intuitivamente eles possam fazer sentido, devem ser tratados de forma adequada, pois podem distorcer resultados da análise a ser elaborada.

Além desses problemas com variáveis ou campos numéricos, devemos também olhar para os dados não-numéricos. Ou seja, Gender, Property_Tree, Married, Education e Dependents, para ver se eles contêm alguma informação útil para o nosso objetivo.

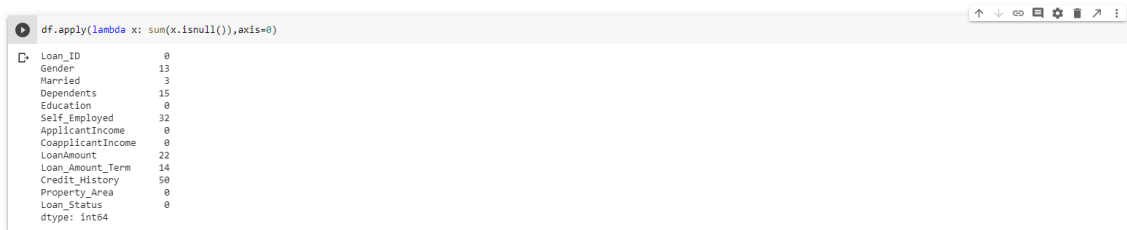
Sendo um iniciante em **Pandas**, eu recomendo consultar a bibliografia que apresentamos, especialmente o livro “Python para Análise de Dados” que foi escrito por ninguém menos que o autor da biblioteca **Pandas**. Nele você encontrará detalhes de muitas técnicas úteis de manipulação de dados, além das que abordamos em nossos laboratórios.

Verifique os Valores Ausentes no Conjunto de Dados

Vamos analisar os valores ausentes em todas as variáveis, pois a maioria dos modelos analíticos não funciona com dados ausentes (nan). E mesmo se funcionassem, tratar adequadamente dados ausentes é importante, pois podem causar distorções nos resultados das análises. Vamos verificar o número de valores nulos e **NaN** no nosso conjunto de dados.

```
df.apply(lambda x: sum(x.isnull()), axis=0)
```

Este comando deve nos dizer o número de valores faltantes em cada coluna ou variável, pois **isnull()** retorna 1 se o valor é nulo.



Embora os valores ausentes não sejam muito elevados em número, muitas variáveis têm valores ausentes e idealmente cada um deles deve ser estimado e adicionado aos dados.

Observação: valores ausentes podem nem sempre ser nulos ou **NaNs**. Por exemplo, se o valor de prazo do empréstimo for **0** ou muito próximo disso, faz sentido? Ou você considera que está ausente? Suponho que sua resposta seja que é ausente e você está certo. Assim, devemos verificar também se há valores que não fazem sentido e considerá-los como ausentes, pois provavelmente são erros ou ruídos contidos nos dados.

Tratamento de Valores Ausentes no Valor do Empréstimo

Existem inúmeras maneiras de tratar os valores ausentes dos empréstimos – a mais simples é substituí-los pela média, o que pode ser feito pelo seguinte código, **mas não o execute ainda**, pois ainda estamos analisando as melhores estratégias para isso:

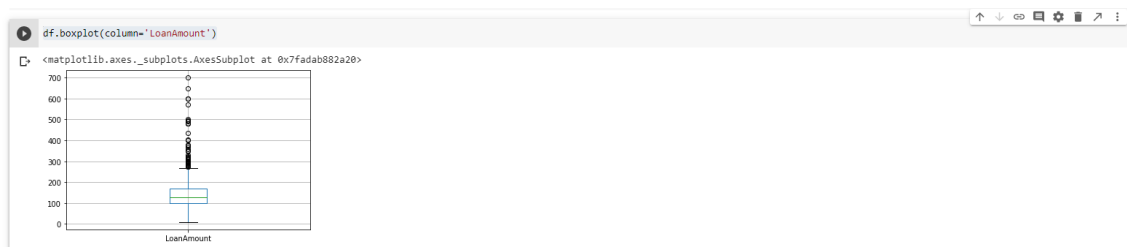
```
df['LoanAmount'].fillna(df['LoanAmount'].mean(),  
inplace=True)
```

O outro extremo para resolver esse problema poderia ser a construção de um modelo de aprendizagem de máquina supervisionada para prever o montante do empréstimo com base em outras variáveis.

Como o nosso principal objetivo neste laboratório é apresentar apenas algumas possibilidades básicas de tratamento de dados, vamos utilizar uma abordagem que se encontra entre estas duas alternativas extremas. A hipótese que admitiremos é que o nível de escolaridade e o trabalho por conta própria podem se combinar para fornecer uma boa estimativa do montante do empréstimo.

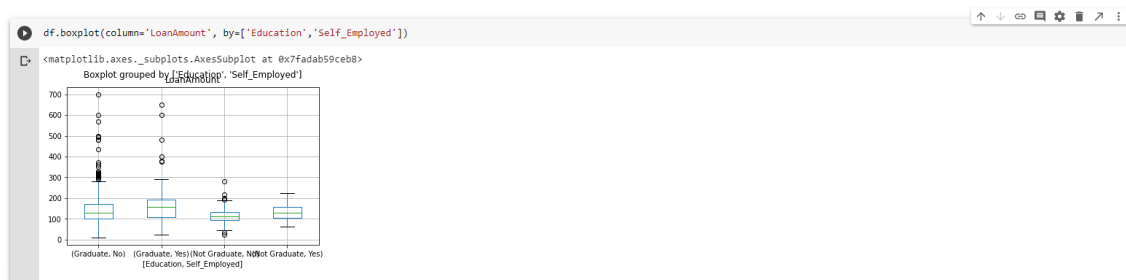
Primeiro, vamos olhar para o gráfico boxplot para ver se existe uma tendência:

```
df.boxplot(column='LoanAmount')
```



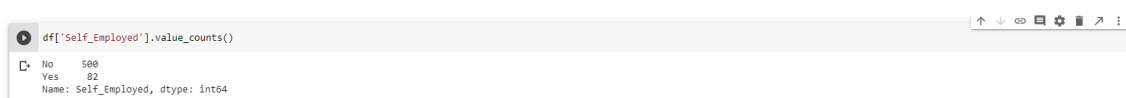
Agora vamos agrupar por Escolaridade (*Education*) e Autônomo (*Self_Employed*).

```
df.boxplot(column='LoanAmount',  
by=['Education', 'Self_Employed'])
```



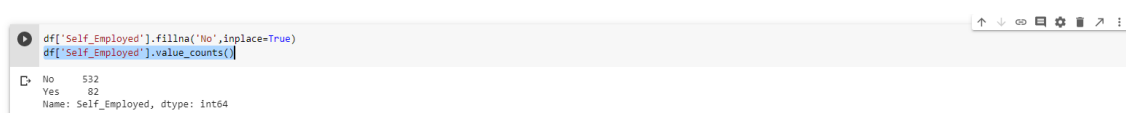
Podemos observar algumas variações na quantidade média de empréstimo para cada grupo e isto pode ser utilizado para preencher os valores. Mas, primeiro, temos que garantir que as variáveis `Self_Employed` e `Education` não tenham valores ausentes.

Como dissemos anteriormente, `Self_Employed` tem alguns valores ausentes. Vamos olhar para a tabela de frequência:



Como 86% dos valores são “não”, podemos assumir que é seguro calcular os valores ausentes como “Não” pois há uma alta probabilidade de sucesso. Isso pode ser feito usando o seguinte código:

```
df['Self_Employed'].fillna('No', inplace=True)
```



Agora, vamos criar uma tabela dinâmica que nos forneça valores médios para todos os grupos de valores exclusivos de características de `Self_Employed` e `Education`. Em seguida, vamos definir uma função, que retorna os valores dessas células e aplicá-los para preencher os valores ausentes do valor do empréstimo `LoanAmount`:

```
table = df.pivot_table(values='LoanAmount',
index='Self_Employed', columns='Education',
aggfunc=np.median)
table
```



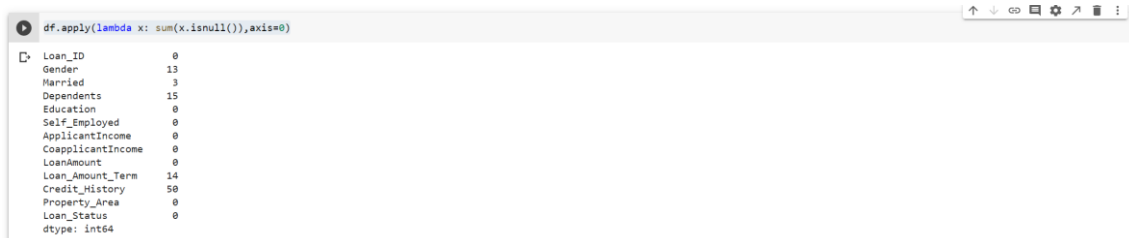
```
# Define a função que retorna o valor da tabela pivot
def fage(x):
    return table.loc[x['Self_Employed'], x['Education']]
# Substitui valores faltantes

df['LoanAmount'].fillna(df[df['LoanAmount'].isnull()].apply(
    fage, axis=1), inplace=True)
```

Esta é uma boa maneira de preencher os valores ausentes do montante do empréstimo (*LoanAmount*).

Verifique novamente os valores faltantes e compare com o resultado anterior para ter certeza de que agora não há mais valores faltantes para *LoanAmount*. Antes havia 22 valores faltantes.

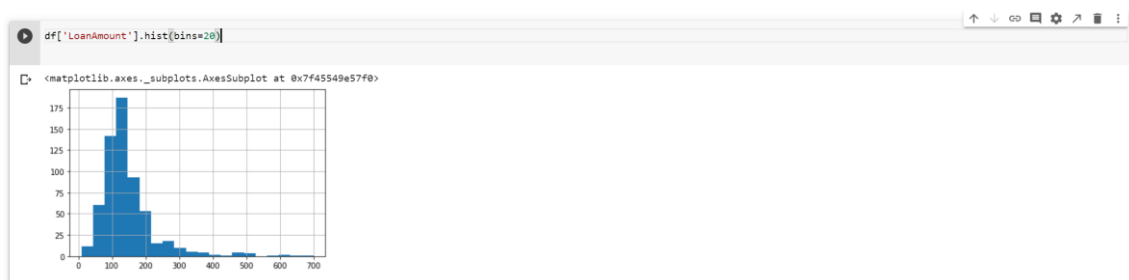
```
df.apply(lambda x: sum(x.isnull()),axis=0)
```



Tratando Valores Extremos

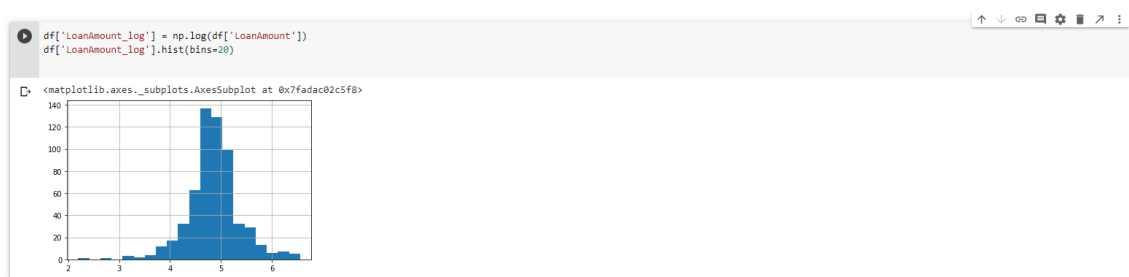
Vamos analisar os valores do empréstimo – *LoanAmount* - em primeiro lugar.

```
df['LoanAmount'].hist(bins=20)
```



Valores extremos são possíveis, ou seja, algumas pessoas podem solicitar empréstimos de alto valor devido a necessidades específicas. Então, ao invés de tratá-los como valores atípicos, podemos tentar uma transformação log para reduzir os seus efeitos:

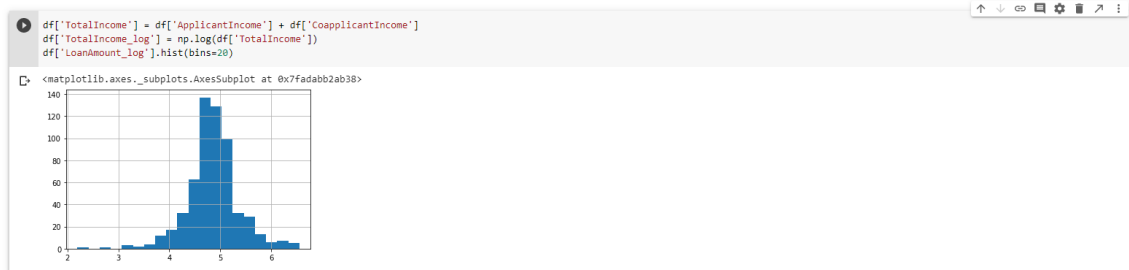
```
df['LoanAmount_log'] = np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins=20)
```



Olhando para o Histograma agora, a distribuição parece muito mais próxima ao normal e os efeitos dos valores extremos foi significativamente reduzido.

Chegando à Renda do Cliente (*ApplicantIncome*). Uma intuição pode ser que alguns clientes tenham renda baixa, mas apoio de fiadores fortes. Assim, pode ser uma boa ideia combinar ambos os rendimentos como sendo a renda total considerada para avaliar o empréstimo e fazer a mesma transformação log.

```
df['TotalIncome'] = df['ApplicantIncome'] +  
df['CoapplicantIncome']  
df['TotalIncome_log'] = np.log(df['TotalIncome'])  
df['LoanAmount_log'].hist(bins=20)
```



Podemos observar que a distribuição ficou ainda melhor que antes.

Continue a Explorar os Dados como Exercício

Pararemos por aqui neste exemplo, mas você deveria continuar a analisar e preencher os valores ausentes para **Gender** (sexo), **Married** (estado civil), **Dependents** (dependentes), **Loan_Amount_Term** (prazo do empréstimo), **Credit_History** (histórico de crédito). Pois é, ser um cientista de dados é bastante trabalhoso, mesmo dispondo de ferramentas fantásticas como o Python e suas poderosas bibliotecas.

Além disso, pense sobre possíveis informações adicionais que podem ser derivadas a partir dos dados. Por exemplo, criar uma coluna para **LoanAmount / TotalIncome** pode fazer muito sentido, uma vez que dá uma ideia se o candidato tem condições adequadas para pagar pelo empréstimo.

Construção de um Modelo Preditivo

Introdução

Utilizando os dados agora já tratados sobre os empréstimos vamos criar um modelo para prever os clientes que estarão ou não aptos para receber crédito desta Instituição Financeira.

Após tornarmos os dados úteis para criar um modelo analítico, vamos apresentar o código Python para criar um modelo preditivo em nosso conjunto de dados. **Scikit-learn (sklearn)** é a biblioteca mais comumente usada em Python para este fim e seguiremos esta trilha. Se você estiver muito ansioso para aprender, convindo-o(a) a consultar a bibliografia indicada, especialmente o livro “Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow”. Porém, ao longo do curso haverá outras disciplinas que abordarão adequadamente estes temas e técnicas.

Além disso, é bom deixar claro que estas mesmas técnicas e praticamente os mesmos recursos que utilizaremos neste laboratório experimental com a linguagem Python, também estão todos disponíveis na linguagem R. Em geral, não há melhor nem pior ferramenta, mas aquela que você (pessoalmente) gosta mais ou está mais habituado ou conformável para utilizar.

Devido à utilização de algoritmos de *machine learning*, **sklearn** exige que todas as entradas sejam numéricas e não ausentes. Portanto, inicialmente devemos eliminar todos os valores faltantes e converter todas as variáveis categóricas em numéricas através da codificação das categorias. Isso pode ser feito usando o seguinte código, que utiliza a função **LabelEncoder** da própria biblioteca **sklearn**:

```
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0],
inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()
[0], inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0],
inplace=True)
from sklearn.preprocessing import LabelEncoder
var_mod
=['Gender', 'Married', 'Dependents', 'Education', 'Self_Employe
d', 'Property_Area', 'Loan_Status']
le = LabelEncoder()
for i in var_mod: df[i] = le.fit_transform(df[i])
df.dtypes
```



```
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
from sklearn.preprocessing import LabelEncoder
var_mod = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area', 'Loan_Status']
le = LabelEncoder()
for i in var_mod: df[i] = le.fit_transform(df[i])
df.dtypes
```

Loan_ID	object
Gender	int64
Married	int64
Dependents	int64
Education	int64
Self_Employed	int64
ApplicantIncome	int64
CoapplicantIncome	float64
LoanAmount	float64
Loan_Amount_Term	float64
Credit_History	float64
Property_Area	int64
Loan_Status	int64
LoanAmount_log	float64
TotalIncome	float64
TotalIncome_log	float64
dtype:	object

Espero que você analise as instruções que utilizamos para entender como fizemos esta transformação nos dados e verificar acima que todos os dados agora são numéricos. Em seguida vamos importar os módulos necessários para aplicar diversos algoritmos de classificação disponíveis em **sklearn**: *Logistic Regression*, *Random Forests* e *Decision Trees*.

Em seguida, vamos definir uma função de classificação genérica, que leva um modelo como argumento de entrada e determina a pontuação de precisão e de validação cruzada obtida. Uma vez que este é um laboratório experimental, não vamos entrar em detalhes de sua codificação, mas incentivamos a analisá-la, como aprendizado.

```
#Importa os modelos da biblioteca scikit learn:
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier,
export_graphviz
from sklearn import metrics
```

```
#Função para fazer um modelo de classificação para avaliar
performance
def classification_model(model, data, predictors, outcome):
    #Ajusta o modelo:
    model.fit(data[predictors],data[outcome])
    #Faz previsões nos dados de treino:
    predictions = model.predict(data[predictors])
    #Mostra a acurácia
    accuracy =
metrics.accuracy_score(predictions,data[outcome])
    print("Acurácia : %s" % "{0:.3%}".format(accuracy))

    #Realiza validação cruzada k-fold com 5 folds
    kf = KFold(5,shuffle=False)
    error = []
    for train, test in kf.split(data):
        # Filtra dados de treino
        train_predictors = (data[predictors].iloc[train,:])

        # O alvo que estamos usando para treinar o algoritmo
        train_target = data[outcome].iloc[train]

        # Treinando o algoritmo com previsores e alvo
        model.fit(train_predictors, train_target)

        #Grava erros de cada loop de validação cruzada
        error.append(model.score(data[predictors].iloc[test,:],
data[outcome].iloc[test]))
```

```
print("Score da Validação Cruzada : %s" %  
      "{0:.3%}".format(np.mean(error)))  
  
#Acerta de novo o modelo para que possa se referir fora  
da função  
model.fit(data[predictors], data[outcome])
```

Regressão Logística

Vamos aplicar o primeiro algoritmo de regressão logística. Uma forma seria incluir todas as variáveis no modelo, mas isso pode resultar em *overfitting* ou super ajuste (não se preocupe se você ainda desconhece essa terminologia, ao longo do curso ficará bem familiarizado com ela). Em palavras simples, utilizar todas as variáveis pode resultar em um modelo de compreensão das relações complexas específico e exclusivo para os dados de treinamento que dispomos, mas difícil de ser generalizado, e, portanto, de pouca utilidade prática.

Podemos tecer algumas hipóteses intuitivas para iniciar o nosso modelo. Podemos, por exemplo, supor que a chance de conseguir um empréstimo será maior para:

1. Os clientes que têm um histórico de crédito (lembre-se que observamos isso na exploração dos dados);
2. Os clientes e fiadores com maiores rendimentos (parece razoável a instituição considerar que há menor risco em emprestar para que tem maiores rendimentos);
3. Os clientes com maior nível de escolaridade (parece razoável a instituição considerar que clientes com maior escolaridade avaliem melhor sua capacidade de pagamento do empréstimo);
4. Imóveis em áreas urbanas com alta perspectiva de crescimento (parece razoável imaginar que imóveis em áreas urbanas com alta expectativa de crescimento se valorizarão mais e reduzirão o risco da instituição, caso o cliente não pague o empréstimo).

Como você pode perceber, a ciência de dados não se limita ao domínio da estatística e da computação, ou das linguagens Python e R, mas o cientista de dados deve investigar tudo, inclusive aspectos do mercado, do negócio e qualquer outro aspecto que possa ter significativa influência nos objetivos e resultados da análise, criando, assim, um modelo analítico válido, eficiente e robusto.

Vamos iniciar o nosso primeiro modelo com o algoritmo de Regressão Logística utilizando a variável “histórico de crédito” (*Credit_History*) como variável preditora e status do empréstimo (*Loan_Status*) como variável de saída (dependente).

```
outcome_var = 'Loan_Status'  
model = LogisticRegression()  
predictor_var = ['Credit_History']
```

```
classification_model(model, df, predictor_var, outcome_var)
```

```
outcome_var = 'loan_Status'
model = LogisticRegression()
predictor_var = ['Credit_History']
classification_model(model, df, predictor_var, outcome_var)
```

Acurácia : 88.945%
Score da Validação Cruzada : 88.946%

A acurácia e o *score* (pontuação) da validação cruzada obtidos parecem bons. Mas agora podemos tentar diferentes combinações de variáveis:

```
predictor_var =
['Credit_History', 'Education', 'Married', 'Self_Employed', 'Pr
operty_Area']
classification_model(model, df, predictor_var, outcome_var)
```

```
predictor_var = ['Credit_History', 'Education', 'Married', 'Self_Employed', 'Property_Area']
classification_model(model, df, predictor_var, outcome_var)
```

Acurácia : 88.945%
Score da Validação Cruzada : 88.946%

Geralmente esperamos que a precisão aumente adicionando variáveis. Mas este é um caso mais desafiador. A precisão e a pontuação de validação cruzada não estão sendo impactadas por variáveis menos importantes. Histórico de crédito está dominando o modelo. Temos duas opções:

1. Engenharia de Recursos: derivam-se novas informações (variáveis) e tenta-se prever com elas. Vamos deixar isso para a sua criatividade... e futura pesquisa.
2. Utilizar outras técnicas de modelagem com algoritmos melhores do que a Regressão Logística. Vamos explorar essa alternativa a seguir.

Árvore de Decisão

Árvore de decisão é uma outra técnica ou algoritmo de modelo preditivo conhecido por proporcionar maior precisão do que o modelo de regressão logística. Você certamente estudará mais sobre árvores de decisão ao longo do curso.

```
model = DecisionTreeClassifier()
predictor_var =
['Credit_History', 'Gender', 'Married', 'Education']
classification_model(model, df, predictor_var, outcome_var)
```

```
model = DecisionTreeClassifier()
predictor_var = ['Credit_History', 'Gender', 'Married', 'Education']
classification_model(model, df, predictor_var, outcome_var)
```

Acurácia : 88.945%
Score da Validação Cruzada : 88.946%

Novamente o modelo baseado em variáveis categóricas foi incapaz de ter um impacto porque o histórico de crédito os está dominando, como aconteceu na Regressão Logística. Vamos tentar algumas variáveis numéricas:

```
#Podemos tentar diferentes combinações de variáveis:
predictor_var = ['Credit_History', 'Loan_Amount_Term']
classification_model(model, df, predictor_var, outcome_var)
```

```
#Podemos tentar diferentes combinações de variáveis:
predictor_var = ['Credit_History', 'Loan_Amount_Term']
model = DecisionTreeClassifier()
classification_model(model, df, predictor_var, outcome_var)
```

Acurácia : 81.270%
Score da Validação Cruzada : 80.295%

Nesta tentativa observamos que, embora a precisão tenha subido ao adicionar as variáveis diferentes, o erro de validação cruzada caiu. Este pode ser o resultado do modelo de sobre ajustamento dos dados (*overfitting*). Vamos tentar um algoritmo ainda mais sofisticado e ver se isso ajuda.

Floresta Aleatória

Floresta aleatória é outro algoritmo para resolver o problema de classificação que você estudará mais adiante no curso. Consiste basicamente em uma combinação aleatória de várias árvores de decisão para se obter melhor precisão de quando se utiliza uma única árvore como no caso anterior; daí o nome do algoritmo.

Uma vantagem da Floresta Aleatória é que podemos fazê-la funcionar com todas as características ou variáveis, pois ela retorna uma matriz de importância de recurso (variável) que pode ser usada para selecionar os recursos (variáveis) mais relevantes para a solução do problema.

```
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender', 'Married', 'Dependents',
'Education', 'Self_Employed', 'Loan_Amount_Term',
'Credit_History', 'Property_Area',
'LoanAmount_log', 'TotalIncome_log']
classification_model(model, df, predictor_var, outcome_var)
```

```
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Loan_Amount_Term', 'Credit_History', 'Property_Area']
classification_model(model, df, predictor_var, outcome_var)
```

Acurácia : 86.156%
Score da Validação Cruzada : 77.529%

Aqui podemos ver que a precisão melhorou para o grupo de treinamento, mas a validação cruzada piorou. Este parece ser o caso de super ajuste e pode ser resolvido de duas maneiras:

1. A redução do número de preditores;
2. Ajustando os parâmetros do modelo.

Vamos tentar ambos. Primeiro vemos a matriz importância de recurso a partir do qual vamos obter as características mais importantes.

```
#Criar uma série:
featimp = pd.Series(model.feature_importances_,
index=predictor_var).sort_values(ascending=False)
print(featimp)
```

```
#Criar uma série:
featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_values(ascending=False)
print(featimp)
```

Credit_History	0.471468
Dependents	0.128418
Property_Area	0.114974
Loan_Amount_Term	0.109190
Education	0.050286
Married	0.047425
Self_Employed	0.043728
Gender	0.034601

dtype: float64

Antes de mais nada, conforme já estávamos suspeitando, note a predominância da variável ou recursos **Credit_History**; isto claramente quer dizer que os clientes que já receberam crédito tendem a receber novamente. Porém, será que esta é realmente a melhor política de crédito a ser adotada pela nossa Instituição? Parece que isso seria o "óbvio", mas certamente, como cientistas, temos que tentar ir além através dos dados e das ferramentas analíticas de que dispomos.

Vamos usar as 5 principais variáveis para a criação do modelo. Além disso, vamos modificar um pouco os parâmetros do modelo de floresta aleatória:

```
model = RandomForestClassifier(n_estimators=25,
min_samples_split=25, max_depth=7, max_features=1)

predictor_var =
['TotalIncome_log', 'LoanAmount_log', 'Credit_History', 'Depen
dents', 'Property_Area']

classification_model(model, df, predictor_var, outcome_var)
```

```
model = RandomForestClassifier(n_estimators=25, min_samples_split=25, max_depth=7, max_features=1)
predictor_var = ['Credit_History', 'Dependents', 'Property_Area', 'Loan_Amount_Term', 'Education']
classification_model(model, df, predictor_var, outcome_var)
```

Acurácia : 81.107%
Score da Validação Cruzada : 80.457%

Note que, embora a precisão tenha sido reduzida, a pontuação de validação cruzada está melhor, mostrando que o modelo está generalizando bem. Devemos destacar que os modelos de floresta aleatória não são exatamente repetíveis. Rodadas diferentes irão resultar em variações ligeiramente diferentes devido à randomização desta técnica. Mas as saídas devem ser próximas.

Mesmo depois de alguns ajustes nos parâmetros fundamentais na floresta aleatória, atingimos uma precisão de validação cruzada praticamente idêntica ao do modelo de regressão logística original.

Vamos parar por aqui neste primeiro laboratório experimental de análise de dados. Porém, este exercício nos trouxe algumas aprendizagens muito interessantes e originais:

1. Usar um modelo mais sofisticado não garante resultados melhores.
2. Evite o uso de técnicas de modelagem complexas como uma “caixa preta” sem compreender os conceitos subjacentes. Fazê-lo seria aumentar a tendência de *overfitting* tornando seus modelos menos interpretáveis.
3. Engenharia de Recursos é a chave para o sucesso. Todo mundo pode usar um modelo "Xgboost" mas a verdadeira arte e criatividade encontra-se em melhorar suas **variáveis** ou recursos (os dados, portanto) para atender melhor o modelo e alcançar seus objetivos.
4. Ciência de dados é multidisciplinar e vai além das técnicas estatísticas e computacionais. Procure não se contentar com resultados óbvios ou simplesmente consistentes, mas, como cientista, tente ir além através dos dados

e das ferramentas analíticas de que dispomos para inovar descobrindo o que poucos ou ainda ninguém descobriu.

Etapa Final: Relatório de Elaboração do Laboratório

Você deve entregar um relatório com os resultados das etapas elaboradas neste laboratório no e-Disciplinas, para formatá-lo siga estas orientações:

1. Crie um documento Word e identifique-o com o nome do laboratório, data de elaboração e o seu nome ou da dupla que o elaborou;
2. Crie um tópico para cada resultado que você considerar relevante (manipulação de dados ou resultado de algum processamento) identificando-o com um título e uma breve explicação. Os resultados podem ser imagens de gráficos gerados ou de listas de valores ou dados de resultados obtidos. Não devem ser incluídos os scripts ou instruções de processamento utilizados, inclua apenas os resultados que você considerar relevantes.
3. No final do relatório crie um último tópico denominado “Conclusões” e elabore comentários, sugestões e conclusões sobre o que você pode aprender com a elaboração deste laboratório.

Conclusões Finais

Espero que este laboratório tenha ajudado o início da sua atividade de *Data Science* com Python. Acredito que ela não só lhe deu uma ideia sobre a linguagem Python e os métodos básicos de análise de dados, mas também mostrou como será possível implementar algumas das técnicas mais sofisticadas atualmente disponíveis.

Python é realmente uma ótima ferramenta e está se tornando uma linguagem cada vez mais popular entre os cientistas de dados. A razão disso é ser relativamente fácil de aprender, possuir centenas de funções e se integrar bem com bases de dados e ferramentas como *Spark* e *Hadoop*. Fundamentalmente, Python oferece facilidade exploratória, capacidade computacional e poderosas bibliotecas de análise de dados.

Com certeza você percebeu que aprender Python leva tempo, pois os recursos que oferece são quase ilimitados. Se alguém disser a você que “domina” Python, desconfie, pois acho que ninguém seja capaz disso; pode apenas conhecer bastante.

Desta forma, aprender Python serve para realizar o ciclo de vida completo de qualquer projeto de Data Science, incluindo leitura, análise, visualização e finalmente previsões e obtenção de conhecimento a partir dos dados.

Parabéns! Você concluiu com sucesso o último Laboratório!

Referências

Bibliografia

- Python Tutorial em <https://docs.python.org/pt-br/3.7/tutorial/index.html>.
- Python para Análise de Dados – Tratamento de Dados com Pandas, NumPy e IPython, McKinney, Wes, Novatec, 2018.
- Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow, Géron, Aurélien, Alta Books, 2019

- Tutorial Completo para Aprender Data Science com Python, acessível em <https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/>

Vídeo Aulas

- Vídeo Aula Conhecendo Python:
<https://www.youtube.com/watch?v=BrV6QNFuVZg>
- Vídeo Aula Introdutória de Python:
<https://www.youtube.com/watch?v=Gojgw9BQ5qY&t=69s>
- Vídeo Aula Python Data Science:
https://www.youtube.com/watch?v=F608hzn_ygo&t=129s
- Vídeo Curso de Python:
https://www.youtube.com/playlist?list=PLHz_AreHm4dIKP6QQCekuIPky1Ciwmdl6