Um tutorial completo para aprender Data Science com Python do zero

adminvooo 16 de agosto de 2016 7

Traduzido de: A Complete Tutorial to Learn Data Science with Python from Scratch

Autor: KUNAL JAIN

Introdução

Depois de trabalhar com SAS por mais de 5 anos, decidi que era hora de sair da minha zona de conforto. Sendo um cientista de dados, minha busca por outras ferramentas úteis foi ótima! Felizmente, não demorou muito tempo para eu me decidir: Python foi minha escolha.

Eu sempre tive inclinação para a codificação. Este era o momento de fazer o que eu realmente sempre amei: código. E o melhor é que codificar ficou tão fácil!

Eu aprendi as noções básicas de Python em uma semana e, desde então, não só exploro profundamente essa linguagem como ajudo muitos outros a aprende-la. Python teve origem como linguagem de propósito geral, mas ao longo dos anos e com forte apoio da comunidade, passou a ter bibliotecas dedicadas para análise de dados e modelagem preditiva.

Como existem poucas informações sobre Python para Data Science, decidi criar este tutorial para ajudar as pessoas a aprenderem Python mais rápido. Neste tutorial vamos dar uma boa quantidade de informações sobre como usar Python para análise de dados e vamos revisa-la até que se esteja confortável para desenvolver suas próprias aplicações.

Índice

- Regressão logística
- Árvore de decisão
- "Random forest" – Floresta aleatória

Vamos começar!

1. **Fundamentos do Python para Análise de Dados**

Por que aprender Python para análise de dados?

Python tem atraído muito interesse como linguagem para análise de dados. Eu comparei Python com SAS & R há algum tempo atrás.

Aqui estão algumas razões a favor do Python:

- Código aberto – livre para instalar
- Comunidade online impressionante
- Muito fácil de aprender
- Pode se tornar uma linguagem comum para Data Science e a produção de produtos analíticos de base web.

Deve-se dizer que ele também tem algumas desvantagens:

- É uma linguagem interpretada, em vez de linguagem compilada – pode até gastar mais recursos de CPU. No entanto, dada a economia de tempo de programação (devido à facilidade de aprendizagem), ainda pode ser uma boa opção.

Python 2.7 v / s 3.4

Este é um dos temas mais debatidos em Python. Você vai sempre cruzar com esse tema, especialmente enquanto você for um iniciante. Não há escolha certa ou errada. Depende totalmente da situação e da sua necessidade. Vou tentar dar algumas dicas para ajudá-lo a fazer uma escolha bem informada.

Por Que Python 2.7?

- Impressionante apoio da comunidade! Isso é algo que você vai precisar em seus primeiros dias. Python 2 foi lançado no final de 2000 e tem sido usado por mais de 15 anos.
- Infinidade de bibliotecas de terceiros! Embora muitas bibliotecas tenham apoio 3.x, ainda há um grande número de módulos que funcionam apenas em versões

2.x. Se você pretende usar o Python para aplicações específicas, como web-development com alta dependência de módulos externos, 2.7 pode ser melhor.

- Algumas das características de versões 3.x tem compatibilidade com versões anteriores e pode trabalhar com versão 2,7.

Por Que Python 3.4?

- Mais limpo e mais rápido! Desenvolvedores de Python corrigiram algumas falhas inerentes e inconvenientes para estabelecer uma base mais forte para o futuro. Isso pode não ser muito relevante inicialmente, mas pode se tornar importante.
- É o futuro! 2.7 é a última versão para a família 2.x e, eventualmente, todo mundo terá que mudar para versões 3.x. Python 3 lançou versões estáveis nos últimos 5 anos e vai continuar assim.

Não há um vencedor claro, mas suponho que o pano de fundo seja que você deve se concentrar em aprender Python como uma linguagem. Transitar entre as versões deve ser apenas uma questão de tempo. Fique ligado para um artigo específico sobre Python 2.x vs 3.x no futuro próximo!

Como instalar o Python?

Existem 2 abordagens para instalar Python:

- Baixe o Python diretamente do site do projeto e instale componentes e bibliotecas que você quiser individualmente.
- Alternativamente, você pode baixar e instalar um pacote que vem com bibliotecas pré-instaladas. Eu recomendo baixar Anaconda. Outra opção poderia ser Enthought Canopy Express

O segundo método proporciona uma instalação mais livre de aborrecimentos e, portanto, eu a recomendo para iniciantes.

A limitação dessa abordagem é que você tem que esperar por todo o pacote a ser atualizado mesmo que esteja interessado somente na versão mais recente de uma única biblioteca. Isso não deve ser importante a menos que você esteja fazendo pesquisa com estatística de ponta.

Escolhendo um ambiente de desenvolvimento

Uma vez instalado o Python, existem várias opções para a escolha de um ambiente. Aqui estão as 3 opções mais comuns:

- Terminal / Shell baseados

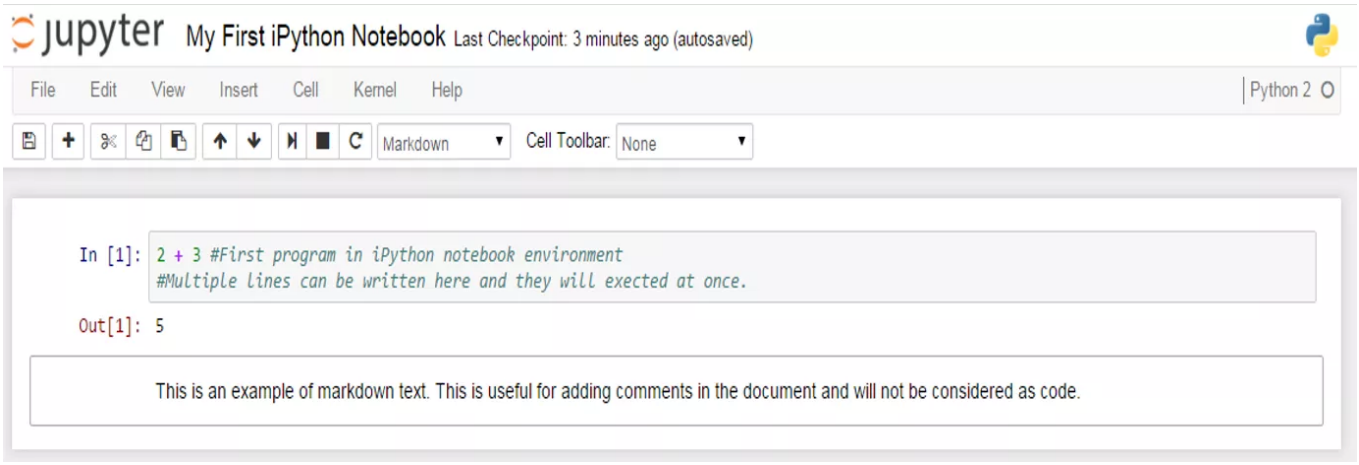
- IDLE (ambiente padrão)
- ipython notebook – semelhante ao 'markdown' do R

Embora a escolha do ambiente certo dependa de sua necessidade, pessoalmente prefiro iPython Notebooks. Ele fornece várias boas características para a documentação ao escrever o código em si e você pode escolher executar o código em blocos (em vez da execução linha a linha)

Usaremos ambiente ipython para este tutorial completo.

Aquecendo: Executando o seu primeiro programa de Python

Vamos usar Python como uma calculadora simples para começar:



Algumas coisas para se observar

- Para começar ipython notebook escreva "ipython notebook" em seu terminal / cmd, dependendo do sistema operacional que você está trabalhando
- Para nomear um caderno ipython, basta clicar sobre o nome – Untitled na imagem acima
- A interface mostra In [*] para entradas e out [*] para a saídas.
- Para executar um código pressione "Shift + Enter" ou "ALT + Enter", se desejar inserir uma linha adicional depois.

Antes de mergulhar mais profundamente na resolução de problemas, vamos dar um passo atrás e entender os conceitos básicos de Python. Como sabemos, estruturas de dados, derivações e construções condicionais formam o cerne de qualquer linguagem. Em Python, incluem-se listas, strings, tuplas, dicionários, for-loop, while-loop, if-else, etc..Vamos dar uma olhada em alguns deles.

2. Bibliotecas Python e Estruturas de Dados

Estruturas de Dados Python

A seguir algumas estruturas de dados que são utilizados em Python. Você precisa se familiarizar com elas para utiliza-las da forma apropriada.

- **Listas –** Listas são uma das estruturas de dados mais versáteis em Python. Uma lista pode ser simplesmente definida como uma lista de valores separados por vírgulas entre colchetes. Listas podem conter itens de diferentes tipos, mas geralmente todos os itens têm o mesmo tipo. As listas Python são mutáveis e elementos individuais de uma lista podem ser alterados.

Aqui está um exemplo rápido para definir uma lista e depois acessá-la:

```
#LISTAS
#Uma lista pode ser definida como valores separados por vírgulas entre colchetes

lista_quadrados = [0,1,4,9,16,25]
lista_quadrados
>>> [0, 1, 4, 9, 16, 25]

#Elementos individuais da lista podem ser acessados ao se colocar o
#número de índice entre colchetes
#Fique atento: o primeiro índice de uma lista é 0 e não 1

lista_quadrados[0] #Indexação retorna o item
>>> 0

#Um intervalo pode ser acessado ao especificar o primeiro e o último índice

lista_quadrados[2:4] #Corte retorna uma nova lista
>>> [4, 9]

#Índice negativo acessa a lista a partir do final

lista_quadrados[-2] #Acessa o segundo elemento a partir do final da lista
>>> 16

#Outros métodos aplicáveis a listas incluem:
#append(), extend(), insert(), remove(), pop(), count(), reverse()
```

Strings – Strings podem ser definidos simplesmente pelo uso de aspas simples (‘), duplas (”) ou triplas (”’). Strings entre aspas triplas (”’) podem ser divididas em várias linhas e são usados frequentemente em docstrings (modo do Python de documentar funções). \ é usado como um caractere de escape. Note por favor que as strings Python são imutáveis, então você não pode mudar parte de strings.

```
#STRINGS
#Uma string é definida pelo uso de aspas simples ( ' ), duplas ( " ) ou triplas ( '

greeting = 'Hello'
print greeting[1]
print len(greeting)
print greeting + 'World'
>>> e
>>> 6
>>> HelloWorld

#'Raw strings' podem ser usadas para passar uma string como ela é.
#O interpretador Python não altera a string, caso seja especificada como 'raw'
#'Raw strings' são aquelas que começam com r

stmt = r'\n é um caracter de nova linha por padrão.'
print stmt
>>>\n é um caracter de nova linha por padrão.

#Strings Python são imutáveis. Tentar mudar uma string vai retornar erro.

greeting[1:] = 'i' #Tentando converter Hello em Hi. Vai retornar erro.
>>> TypeError: 'str' object does not support item assignment

#Métodos comuns de manipulação de strings incluem:
#lower(), upper(), strip(), isdigit(), isspace(), find(), replace()
#split() e join(). São muito úteis na manipulação de dados e limpeza de campos texto
```

Tuplas – Uma tupla é representada por um número de valores separados por vírgulas. Tuplas são imutáveis e a saída é cercada por parênteses para que as tuplas sejam processadas corretamente. Além disso, mesmo sendo imutáveis, elas podem armazenar dados mutáveis, se necessário.

Como as tuplas são imutáveis e não podem mudar, elas são mais rápidas no processamento em comparação com listas. Por isso, se a sua lista tem pouca probabilidade de mudar, use tuplas.

```
#Tuplas
#Uma tupla é representada por valores separados por vírgulas

exemplo_tupla = 0, 1, 4, 9, 16, 25
exemplo_tupla #resultado é dado em parenteses
>>> (0, 1, 4, 9, 16, 25)
```

```
exemplo_tupla[2] #elementos individuais podem ser acessados de forma similar
>>> 4

exemplo_tupla[2] = 6 #Tuplas são imutáveis e aqui vai retornar erro
>>> TypeError: 'tuple' object does not support item assignment
```

Dicionário – Dicionário é um conjunto desordenado de chaves: pares de valores, com a exigência de que as chaves sejam únicas (dentro de um dicionário). Um par de chaves cria um dicionário vazio: {}.

```
#Dicionário

extensao = {'Kunal': 9073, 'Tavish': 9128, 'Sunil': 9223, 'Nitin': 9330}
extensao
>>> {'Kunal': 9073, 'Nitin': 9330, 'Sunil': 9223, 'Tavish': 9128}

extensao['Mukesh'] = 9150
extensao
>>> {'Kunal': 9073, 'Mukesh': 9150, 'Nitin': 9330, 'Sunil': 9223, 'Tavish': 9128}

extensao.keys()
{'Sunil', 'Tavish', 'Kunal', 'Mukesh', 'Nitin'}
```

Derivação Python e Construções Condicionais

Como a maioria das linguagens, Python também tem um FOR-loop que é o método mais utilizado para derivações com uma sintaxe simples:

```
for i in [Python Iterable]:
  expression(i)
```

Aqui "Python Iterable (derivado)" pode ser uma lista, tupla ou outras estruturas de dados avançados que vamos explorar nas seções seguintes. Vamos dar uma olhada em um exemplo simples, determinando o fatorial de um número.

```
fact = 1
for i in range(1, N+1):
  fact *= i
```

As declarações condicionais são usadas para executar fragmentos de código com base em uma condição. A construção mais comumente utilizado é if-else, com a seguinte sintaxe:

```
if [condition]:
  __execution if true__
else:
  __execution if false__
```

Por exemplo, se quisermos saber se o número N é par ou ímpar:

```
if N%2 == 0:
  print 'Par'
else:
  print 'Ímpar'
```

Agora que você está familiarizado com os fundamentos do Python, vamos dar um passo adiante e realizar as seguintes tarefas:

1. Multiplicar 2 matrizes
2. Encontrar a raiz de uma equação quadrática
3. Gráficos de barras, plots e histogramas
4. Modelos estatísticos
5. Acesso a páginas Web

Tentar escrever os códigos a partir do zero pode ser um pesadelo e fará você desistir de Python em 2 dias! Mas você não precisa se preocupar com isso. Felizmente, existem muitas bibliotecas com coisas pré-definidas que podem ser importadas diretamente para o nosso código e que tornam a nossa vida muito mais fácil.

Por exemplo, considere o exemplo fatorial que acabamos de ver. Podemos fazer isso em uma única linha com:

```
math.factorial(N)
```

Claro que precisamos importar a biblioteca de matemática para isso. Vamos explorar as várias bibliotecas a seguir.

Vamos dar um passo à frente e aprender Python nos familiarizando com algumas bibliotecas úteis. O primeiro passo é, obviamente, aprender a importá-las para o nosso

ambiente. Existem várias maneiras de fazer isso em Python:

```
import math as m
```

```
from math import *
```

Na primeira forma, definimos um alias m à biblioteca matemática. Podemos agora usar várias funções de biblioteca de matemática (por exemplo fatorial) referenciando-a através do alias m.factorial().

Na segunda forma, importa-se todo o conjunto de nomes de math, ou seja, pode-se usar fatorial() diretamente, sem se referir à math.

Dica: o Google recomenda que você use o primeiro estilo de importadores de bibliotecas, pois assim você pode saber de onde as funções vieram.
A seguir uma lista de bibliotecas necessárias para quaisquer cálculos científicos e análise de dados:

1. **NumPy** vem de Python numérico. A característica mais poderosa de NumPy é a matriz n-dimensional. Esta biblioteca também contém funções básicas de álgebra linear, transformações de Fourier, capacidades avançadas de números aleatórios e ferramentas para integração com outras linguagens de baixo nível como Fortran, C e C ++
2. **SciPy** vem de Scientific Python. SciPy é construída sobre NumPy e é uma das bibliotecas mais úteis para uma variedade de ciências de alto nível e engenharia como transformação de Fourier discreta, Álgebra Linear e matrizes de otimização e esparsas.
3. **Matplotlib** para traçar grande variedade de gráficos, desde histogramas até gráficos de calor. Pode-se usar o recurso Pylab no notebook ipython (ipython notebook -pylab = inline). Se você ignorar a opção inline, então Pylab converte o ambiente ipython em um ambiente muito semelhante ao Matlab. Pode-se também usar comandos de Latex para adicionar matemática aos seus gráficos.
4. **Pandas** para operação e manipulação de dados estruturados. É amplamente utilizado para preparação de dados. Pandas foram adicionados há relativamente pouco tempo no Python e têm sido fundamentais para impulsionar o uso do Python na comunidade de cientistas de dados.
5. **Scikit Learn** para a 'machine learning'. Construído sobre NumPy, SciPy e matplotlib, esta biblioteca contém uma grande quantidade de ferramentas

eficientes para machine learning e modelagem estatística, incluindo classificação, regressão, clustering e redução de dimensionalidade.

6. **Statsmodels** para modelagem estatística. Statsmodels é um módulo Python que permite aos usuários explorar os dados, estimar modelos estatísticos, e realizar testes estatísticos. Uma extensa lista de estatística descritiva, testes estatísticos, funções de plotagem e estatísticas de resultados estão disponíveis para diferentes tipos de dados.
7. **Seaborn** para visualização de dados estatísticos. Seaborn é uma biblioteca para fazer gráficos estatísticos atraentes e informativos em Python. Baseia-se em matplotlib. Seaborn visa tornar a visualização uma parte central da exploração e compreensão de dados.
8. **Bokeh** para a criação de gráficos interativos, dashboards e aplicações de dados em modernos navegadores web. Permite que o usuário gere gráficos elegantes e concisos no estilo D3.js. Além disso, ele tem a capacidade de interatividade de alto desempenho em conjuntos de dados de grandes volumes ou de streaming.
9. **Blaze** para estender a capacidade de Numpy e Pandas a conjuntos de dados distribuídos e streaming. Pode ser usado para acessar dados de uma variedade de fontes, incluindo Bcolz, MongoDB, SQLAlchemy, Apache Spark, PyTables, etc. Juntamente com Bokeh, Blaze pode atuar como uma ferramenta muito poderosa para a criação e visualizações de dashboards eficazes em pedaços enormes de dados.
10. **Scrapy** para rastreamento web. É um framework muito útil para obter padrões específicos de dados. Ele tem capacidade de iniciar em uma url de uma home de website e, em seguida, cavar através de páginas web dentro do website para pegar informações.
11. **SymPy** para computação simbólica. Tem capacidades abrangentes de aritmética simbólica básica para cálculo, álgebra, matemática discreta e física quântica. Outro recurso útil é a capacidade de formatar o resultado dos cálculos como o código LaTeX.
12. **Requests** para acesso à web. Funciona de forma semelhante à biblioteca Python padrão urllib2, mas é muito mais fácil de codificar. Há diferenças sutis com relação a urllib2, mas para iniciantes, Requests pode ser mais conveniente.

Bibliotecas adicionais que podem ser necessárias:

1. **OS** para operações com sistema operacional e de arquivos
2. **NetworkX** e **IGRAPH** para manipulações de dados gráficos
3. **regular expressions** para encontrar padrões em dados de texto
4. **BeautifulSoup** para fazer scraping de websites. É inferior ao Scrapy uma vez que irá extrair informações a partir de uma única página corrida.

Agora que estamos familiarizados com os fundamentos do Python e bibliotecas adicionais, vamos dar um mergulho profundo na resolução de problemas através do Python. Sim, eu quero dizer fazer um modelo preditivo! No processo, usaremos algumas bibliotecas poderosas e também iremos nos deparar com o próximo nível de estruturas de dados. Vamos levá-los através das 3 fases principais:

- Exploração de Dados – descobrir mais sobre os dados que temos
- Munging de dados – limpar os dados e jogar com eles para torná-los melhores na modelagem estatística
- Modelagem preditiva – fazer funcionar os algoritmos reais e se divertir 🙂

3. A análise exploratória em Python usando Pandas

Para explorar ainda mais os nossos dados, vamos apresentar um outro animal (como se Python não foi suficiente!) – Pandas

Pandas é uma das bibliotecas de análise de dados mais úteis em Python (eu sei que estes nomes soam estranhos, mas tudo bem!). Elas têm sido fundamentais para aumentar o uso de Python na comunidade científica de dados. Vamos agora usar Pandas para ler um conjunto de dados a partir de uma competição da Analytics Vidhya, executar análise exploratória e construir nosso primeiro algoritmo básico de categorização para resolver este problema.

Antes de carregar os dados, vamos compreender as duas principais estruturas de dados em pandas – Séries e Dataframes.

Introdução a Séries e Dataframes

Série pode ser entendida como uma matriz unidimensional rotulada e indexada. Pode-se acessar elementos individuais desta série através destes rótulos.

Um dataframe 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 do número de linha. A diferença essencial é que os nomes das colunas e os números de linha são conhecidos como colunas e índices de linha, no caso dos dataframes.

Séries e dataframes formam o núcleo dos modelos de dados do Pandas em Python. Os conjuntos de dados são lidos primeiro nos dataframes e, em seguida, várias operações (por exemplo: group by, agregações, etc.) podem ser aplicadas muito facilmente às suas colunas.

Veja mais: 10 Minutes to Pandas

Prática com conjunto de dados – Problema de previsão de Empréstimo

Faça o download da base aqui e aqui. Estas são as descrições das variáveis:

```
DESCRIÇÃO DAS VARIÁVEIS:
Variável                  Descrição
Loan_ID                   ID único do empréstimo (loan)
Gender                    Masculino/Feminino (Male/Female)
Married                   Casado - sim ou não (Y/N)
Dependents                Neumero de dependentes
Education                 Grau escolar (Graduate/ Under Graduate)
Self_Employed             Auto empregado - sim ou não (Y/N)
ApplicantIncome           Renda do aplicante
CoapplicantIncome         Renda do co-aplicante
LoanAmount                Montante do empréstimo (loan), em milhares
Loan_Amount_Term          Prazo do empréstimo (loan), em meses
Credit_History            Histórico de crédito corresponde aos critérios
Property_Area             Localização da propriedade (Urban/Semi Urban/ Rural)
Loan_Status               Empréstimo (loan) aprovado - sim ou não (Y/N)
```

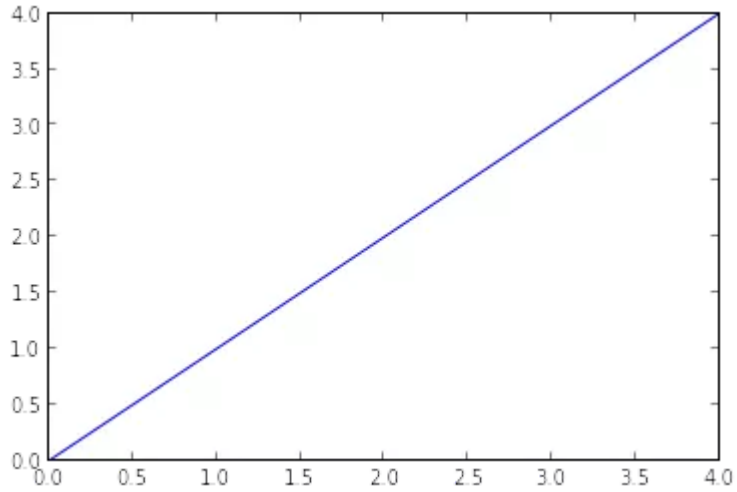
Vamos começar com axploração dos dados

Para começar, inicie a interface de ipython em modo inline Pylab digitando o seguinte em seu terminal/prompt de comando:

```
ipython notebook --pylab=inline
```

Isso abre ipython notebook em ambiente Pylab, que tem algumas bibliotecas úteis já importadas. Além disso, você será capaz de gerar gráficos pela linha de comando, o que torna este um bom ambiente para análise de dados interativos. Você pode verificar se o ambiente foi carregado corretamente digitando o seguinte comando (e recebendo a saída como visto na figura abaixo):

```
plot(arange(5))
```



Atualmente trabalho com o Linux, e ele armazenou os dados indicados no seguinte local:

- Arquivo-Treino
- Arquivo-Teste

Importando bibliotecas e o conjunto de dados:

Abaixo estão as bibliotecas que usaremos neste tutorial:

1. numpy
2. matplotlib
3. pandas

Note por favor que não é preciso importar matplotlib e numpy no ambiente Pyplot. Eu ainda mantive a importação no código para o caso de você usar um ambiente diferente.

Depois de importar a biblioteca, 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

df = pd.read_csv("/home/kunal/Downloads/Loan_Prediction/train.csv") #Lendo o conjunt
```

Rápida Exploração de dados

Depois de ler o conjunto de dados, você pode dar uma olhada nas linhas iniciais usando a função head().

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

In [3]:
```
df.head(10)  #Printing first 10 rows of dataset
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001002	Male	No	0	Graduate	No	5849	0	NaN	360	1
1	LP001003	Male	Yes	1	Graduate	No	4583	1508	128	360	1
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0	66	360	1
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1
4	LP001008	Male	No	0	Graduate	No	6000	0	141	360	1
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516	95	360	1
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504	158	360	0
8	LP001018	Male	Yes	2	Graduate	No	4006	1526	168	360	1
9	LP001020	Male	Yes	1	Graduate	No	12841	10968	349	360	1

10 linhas devem aparecer. Alternativamente, você também pode olhar para mais linhas do conjunto de dados.

Em seguida, você pode olhar para um resumo dos campos numéricos usando a função describe().

```
df.describe()
```

In [4]:
```
df.describe()  #Get summary of numerical variables
```

Out[4]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

A função describe() fornece contagem, média, desvio padrão (STD), mínimo, quartis e máximo (Leia este artigo para atualizar-se em estatísticas básicas para compreender a distribuição de uma população).

Aqui estão algumas inferências, você pode desenhar olhando para a saída de 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 candidatos têm histórico de crédito. Como? A média do campo Credit_History é 0,84 (Lembre-se, Credit_History tem um valor 1 para aqueles que têm um histórico de crédito e 0 caso contrário).
- A distribuição ApplicantIncome parece estar em linha com a expectativa. O mesmo com a CoapplicantIncome.

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

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

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

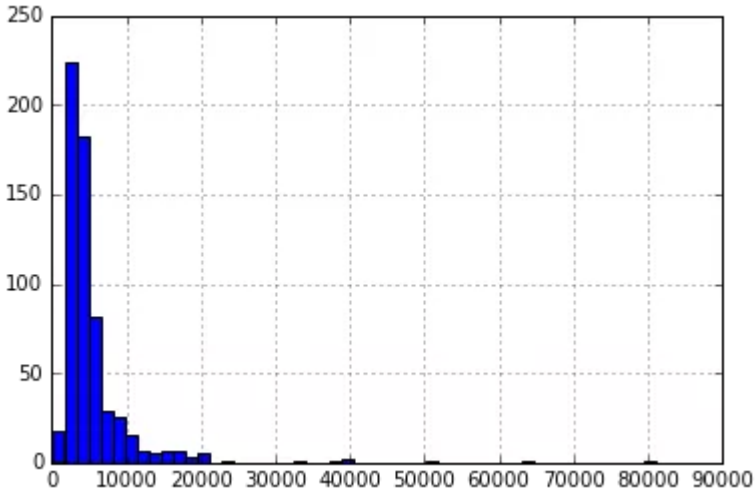
Da mesma forma, podemos olhar para valores exclusivos de histórico de crédito. Note que o dfname[‘column_name’] é uma técnica básica de indexação para acessar uma determinada coluna do dataframe. Ele pode ser uma lista de colunas também. Para mais informações, consulte o “10 Minutes to Pandas”, recurso compartilhado acima.

Análise da Distribuição

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

Vamos começar plotando o histograma do ApplicantIncome usando o seguinte comando:

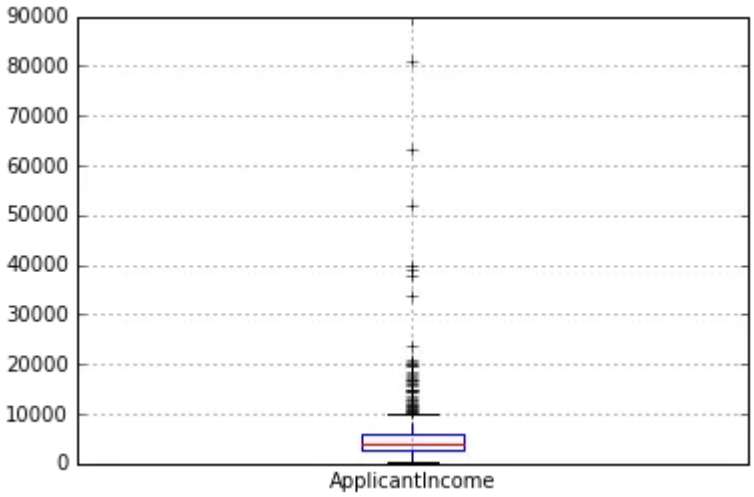
```
df['ApplicantIncome'].hist(bins=50)
```



Observamos que há poucos valores extremos. Esta é também a razão pela qual 50 caixas (bins) são necessários para representar a distribuição claramente.

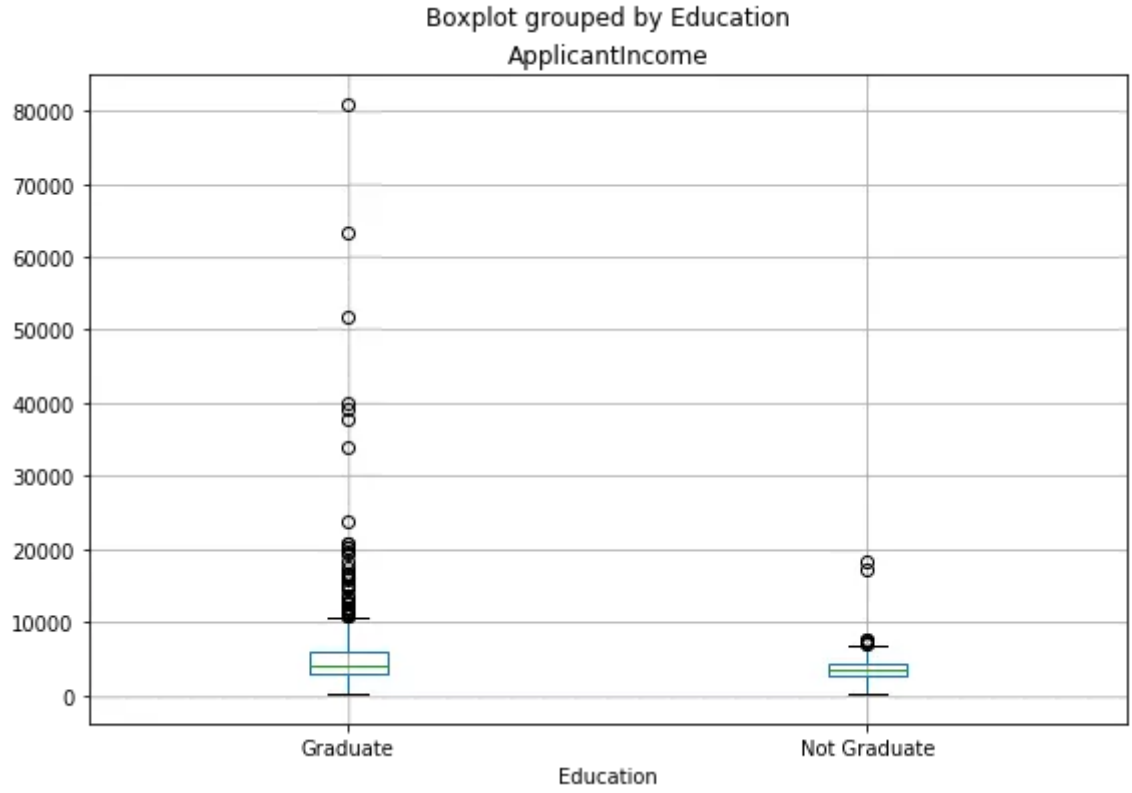
Em seguida, olhemos para o box plots para compreender as distribuições. Box plots para renda do aplicante pode ser traçado por:

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



Isso confirma a presença de uma grande quantidade de outliers/valores extremos. Isso pode ser atribuído à disparidade de renda na sociedade. Parte disso pode ser impulsionado pelo fato de que nós estamos olhando para as pessoas com diferentes níveis de ensino. Vamos segregá-los por Educação:

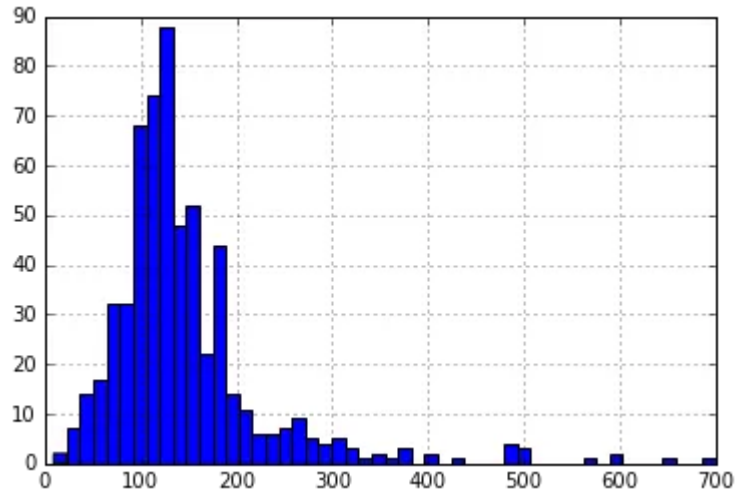
```
df.boxplot(column='ApplicantIncome', by = 'Education')
```



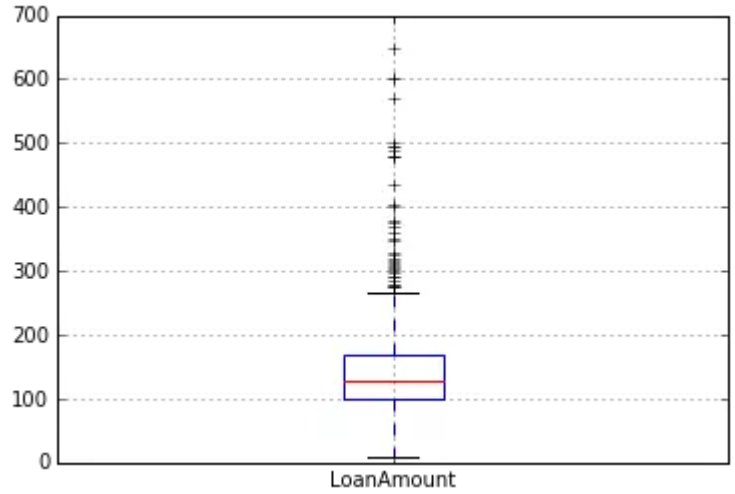
Podemos ver que não há nenhuma diferença substancial entre a renda média de pós-graduação e de não-graduados. Mas há um maior número de diplomados com rendimentos muito elevados que parecem ser os outliers.

Agora, vamos olhar para o histograma e o box plot do valor do empréstimo usando o seguinte comando:

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



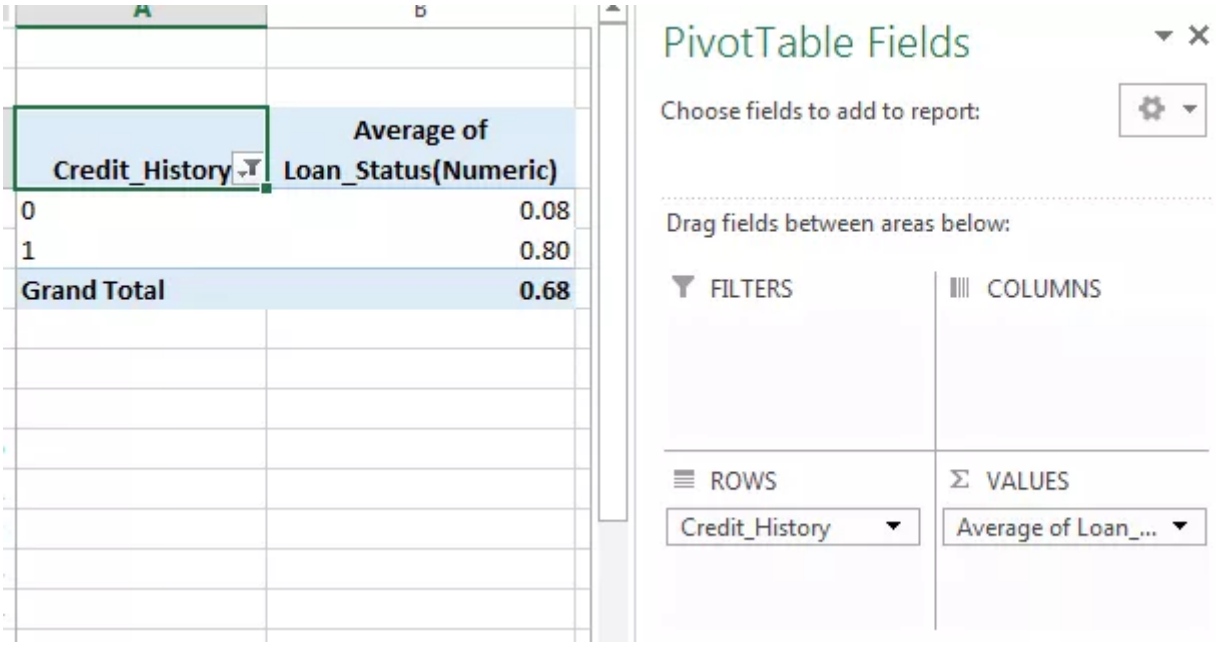

```
df.boxplot(column='LoanAmount')
```



Mais uma vez, há alguns valores extremos. Claramente, tanto ApplicantIncome e LoanAmount exigem uma certa quantidade de '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 seções.

Análise de Variáveis Categóricas

Agora que entendemos as distribuições do ApplicantIncome e do LoanIncome, vamos entender as variáveis categóricas com mais detalhes. Usaremos tabela estilo pivot table do Excel e tabulação cruzada. Por exemplo, vamos olhar para as chances de conseguir um empréstimo com base no histórico de crédito. Isto pode ser conseguido no Microsoft Excel utilizando uma pivot table como:



Nota: aqui, o status de empréstimo foi codificado como 1 para Sim e 0 para Não. Assim, a média representa a probabilidade de obtenção de empréstimo.

Agora, vamos olhar para os passos necessários para gerar uma visão semelhante usando Python. Por favor, consulte este artigo para obter um jeito de as diferentes técnicas de

manipulação de dados em Pandas.

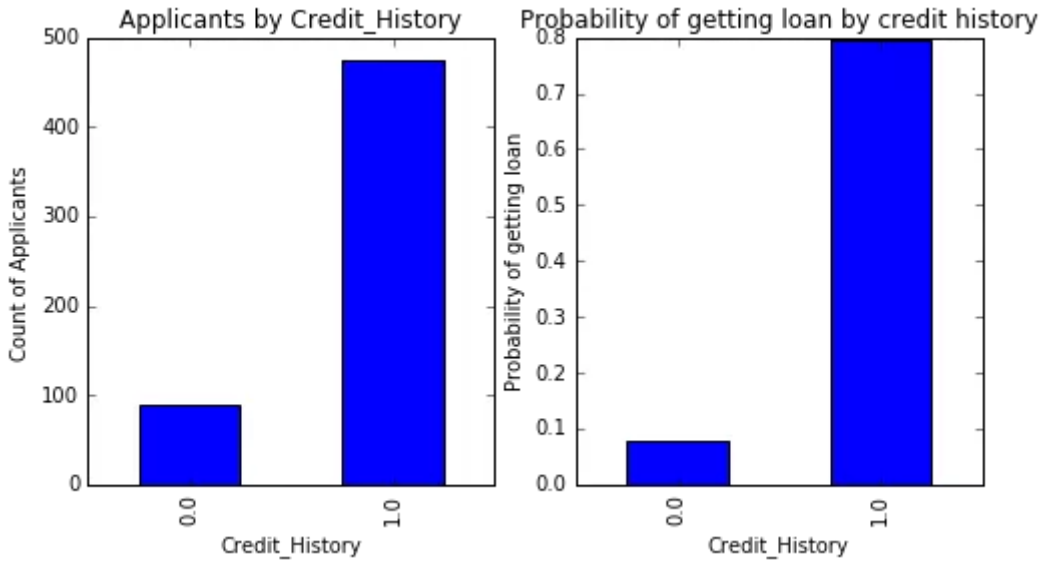
```
temp1 = df['Credit_History'].value_counts(ascending=True)
temp2 = df.pivot_table(values='Loan_Status',index=['Credit_History'],aggfunc=lambda
print('Frequency Table for Credit History:')
print(temp1)
print('\nProbability of getting loan for each Credit History class:')
print(temp2)
```

```
Frequency Table for Credit History:
0     89
1    475
Name: Credit_History, dtype: int64

Probility of getting loan for each Credit History class:
Credit_History
0    0.078652
1    0.795789
Name: Loan_Status, dtype: float64
```

Agora, podemos observar que temos uma tabela_dinâmica semelhante ao MS Excel. Isto pode ser plotado como um gráfico de barras usando a biblioteca “matplotlib” com o seguinte código:

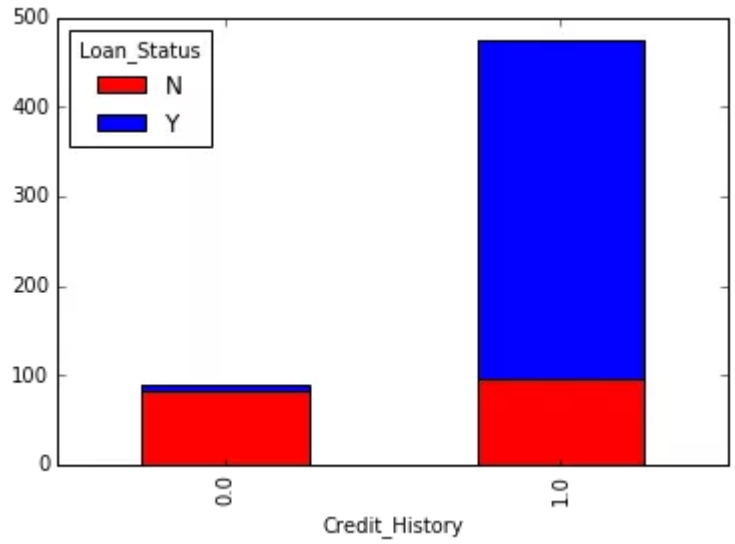
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)
ax1.set_xlabel('Credit_History')
ax1.set_ylabel('Count of Applicants')
ax1.set_title("Applicants by Credit_History")
temp1.plot(kind='bar')
ax2 = fig.add_subplot(122)
temp2.plot(kind = 'bar')
ax2.set_xlabel('Credit_History')
ax2.set_ylabel('Probability of getting loan')
ax2.set_title("Probability of getting loan by credit history")
```



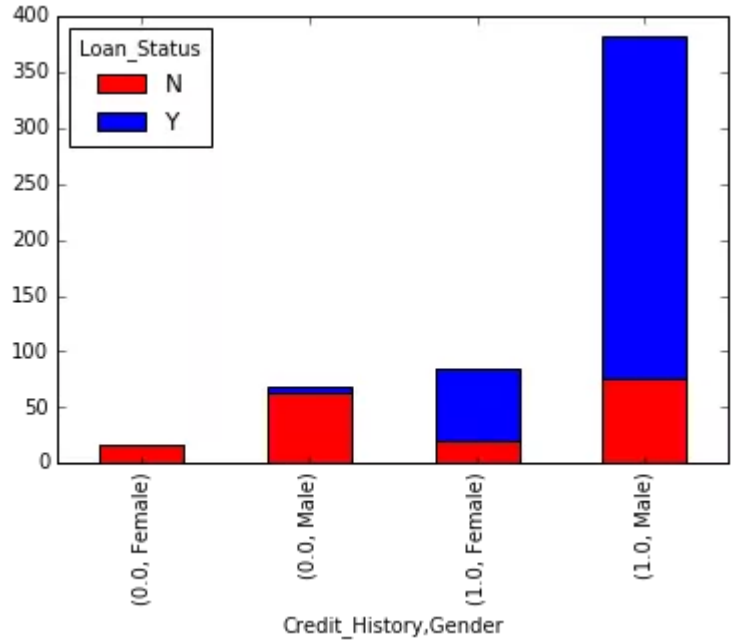
Isso mostra que a chance de conseguir um empréstimo é oito vezes se o requerente tiver um histórico de crédito válido. Pode-se traçar gráficos semelhantes por Estado civel, Profissional Autônomo, Localização de propriedade, etc.

Como alternativa, estes dois plots também podem ser visualizados por combinação em um gráfico empilhado.

```
temp3 = pd.crosstab(df['Credit_History'], df['Loan_Status']) temp3.plot(kind='bar',
```



Você também pode adicionar género na mistura (semelhante a tabela dinâmica em Excel):



Caso ainda não tenha percebido, acabamos de criar dois algoritmos de classificação básicos aqui, um baseado no histórico de crédito e outro em 2 variáveis categóricas (incluindo sexo). Você pode codificar isso para criar sua primeira submissão ao AV Datahacks.

Nós acabamos de ver como podemos fazer análise exploratória em Python utilizando Pandas. Espero que o seu amor pelos Pandas (o animal) tenha aumentado agora – dada a quantidade de ajuda que a biblioteca pôde fornecer na análise de conjuntos de dados.

Em seguida, vamos explorar variáveis ApplicantIncome e LoanStatus e ainda, realizar munging dos dados e criar um conjunto de dados para a aplicação de várias técnicas de modelagem. Eu recomendo fortemente que você pegue um outro conjunto de dados e outro problema e faça uma exploração de dados independente, como exemplo, antes de continuar lendo.

4. Munging de dados com Python: Usando Pandas

Para os que chegaram até aqui, coloquem seus tênis, vamos começar a correr!

Data munging – por quê da necessidade

Durante a exploração dos dados, encontramos alguns problemas que precisam ser resolvidos antes que os dados estejam prontos para serem modelados. Este exercício é normalmente referido como “Data Munging”. Aqui estão os problemas, já estamos cientes de que:

1. Há valores faltando em algumas variáveis. Devemos estimar esses valores, dependendo da quantidade de valores faltantes e da importância esperada das

variáveis.

2. Quando se olha para 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.

Além desses problemas com campos numéricos, devemos também olhar para os campos não-numéricos. Ou seja, Gênero, Property_Tree, Estado vil, Educação e Dependentes, para ver se eles contêm alguma informação útil.

Se você é novo em pandas, eu recomendo ler este artigo antes de prosseguir. Ele detalha algumas técnicas úteis de manipulação de dados.

Verifique os Valores Ausentes no Conjunto de Dados

Vejamos os valores ausentes em todas as variáveis, porque a maioria dos modelos não funciona com dados ausentes. E mesmo se funcionarem, isso ajuda mais frequentemente do que não. Então, vamos verificar o número de nulos e de NaNs no 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, pois isnull () retorna 1 se o valor é nulo.

```
In [14]: df.apply(lambda x: sum(x.isnull()),axis=0)
Out[14]: Loan_ID              0
         Gender              13
         Married              3
         Dependents          15
         Education            0
         Self_Employed       32
         ApplicantIncome      0
         CoapplicantIncome    0
         LoanAmount          22
         Loan_Amount_Term    14
         Credit_History      50
         Property_Area        0
         Loan_Status          0
         dtype: int64
```

Embora os valores ausentes não sejam muito elevados em número, muitas variáveis têm valores ausentes e cada um deles deve ser estimado e adicionado aos dados. Obtenha uma visão detalhada sobre diferentes técnicas de imputação através deste artigo.

Nota: Lembre-se que os valores ausentes podem nem sempre ser NaNs. Por exemplo, se o valor de prazo do empréstimo for 0, isso faz sentido? Ou você considera que está

ausente? Suponho que sua resposta seja que é ausente e você está certo. Assim, devemos verificar se há valores que não fazem sentido.

Como preencher valores ausentes no LoanAmount, valor do empréstimo?

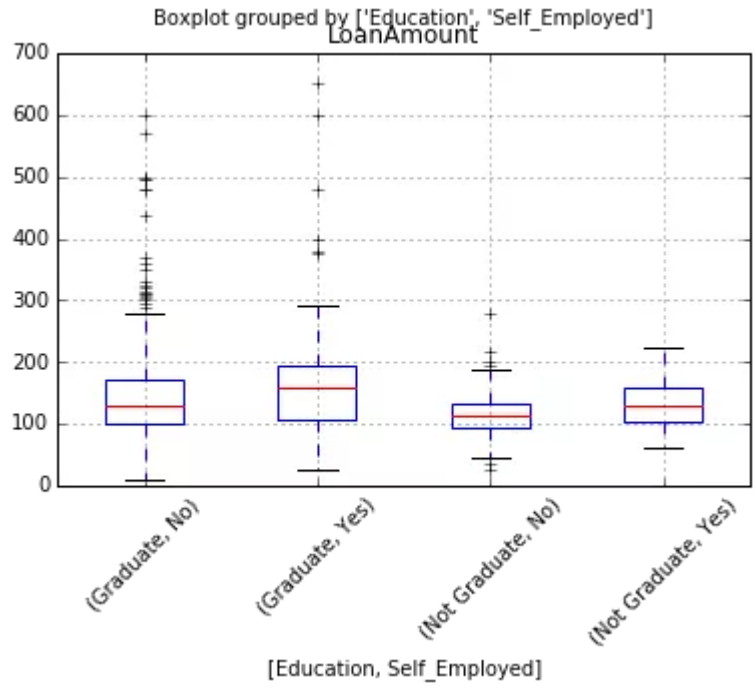
Existem inúmeras maneiras de preencher os valores ausentes dos empréstimos – a mais simples é substituí-los pela média, o que pode ser feito pelo seguinte código:

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

O outro extremo poderia ser a construção de um modelo de aprendizagem supervisionada para prever o montante do empréstimo com base em outras variáveis e, em seguida, utilizar a idade, juntamente com outras variáveis, para prever a sobrevida.

Como o objetivo agora é trazer as etapas do munging de dados, eu vou usar uma abordagem que se encontra em algum ponto entre esses 2 extremos. A hipótese principal é que se o nível de escolaridade ou o trabalho por conta própria podem se combinar para dar uma boa estimativa do montante do empréstimo.

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



Assim, vemos algumas variações na quantidade média de empréstimo para cada grupo e este pode ser utilizado para imputar os valores. Mas, primeiro, temos de 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:

```
In [40]: df['Self_Employed'].value_counts()

Out[40]: No     500
         Yes     82
         Name: Self_Employed, dtype: int64
```

Como 86% dos valores são “não”, 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 Educação. Em seguida, vamos definir uma função, que retorna os valores dessas células e aplicá-los para preencher os valores ausentes de valor do empréstimo:

```
table = df.pivot_table(values='LoanAmount', index='Self_Employed' ,columns='Educatio

# Define a função que retorna o valor da tabela pivot
def fage(x):
   return table.loc[x['Self_Employed'],x['Education']] # Substitui valoes faltantes
   df['LoanAmount'].fillna(df[df['LoanAmount'].isnull()].apply(fage, axis=1), inplac
```

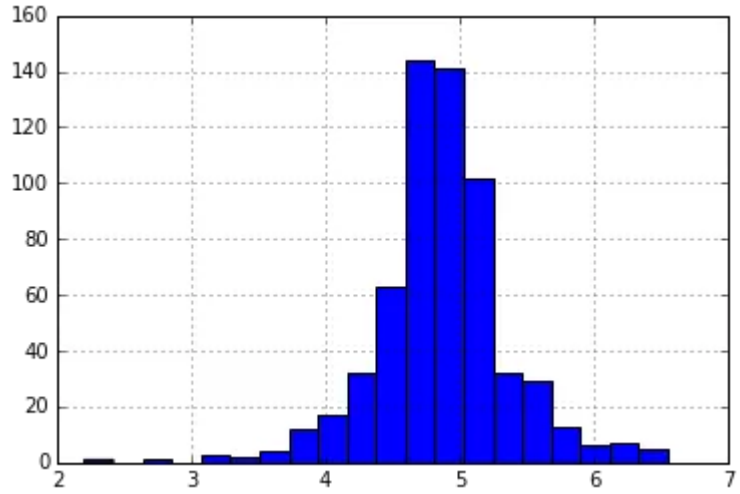
Esta é uma boa maneira de imputar os valores ausentes do montante do empréstimo.

Como tratar valores extremos na distribuição de valor do empréstimo (LoanAmount) e Solicitante de renda (ApplicantIncome)?

Vamos analisar os valores do empréstimo em primeiro lugar. 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, vamos tentar uma transformação log para anular os seus efeitos:

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

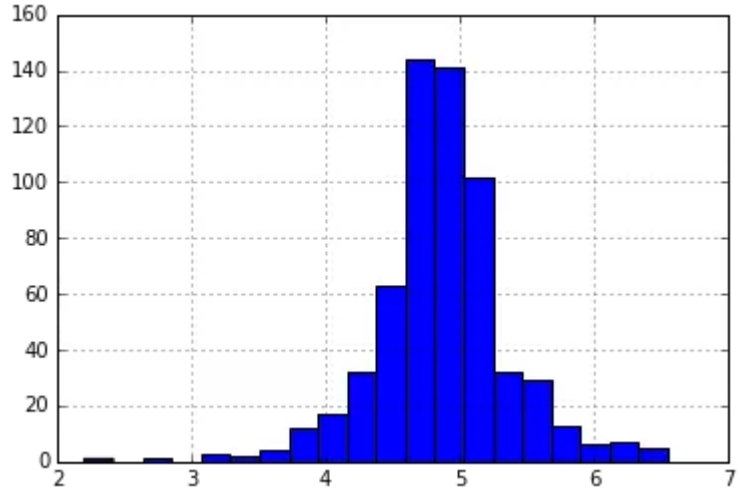
Olhando novamente para o Histograma:



Agora, a distribuição parece muito mais próxima ao normal e os efeitos dos valores extremos foi significativamente diminuído.

Chegando à Renda do Solicitante (ApplicantIncome). Uma intuição pode ser que alguns candidatos tenham renda baixa, mas apoio de co-candidatos fortes. Assim, pode ser uma boa ideia combinar ambos os rendimentos como renda total e fazer a mesma transformação log.

```
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome'] df['TotalIncome_
```



Vemos que a distribuição ficou muito melhor que antes. Vou deixar para você imputar os valores ausentes para Sexo, Casado, Dependentes, Prazo do valor do empréstimo e Histórico de crédito. Além disso, encorajo você a pensar sobre possíveis informações adicionais que podem ser derivadas a partir dos dados. Por exemplo, criar uma coluna para LoanAmount / TotalIncome pode fazer sentido, uma vez que dá uma ideia se o candidato tem condições adequadas para pagar seu empréstimo.

Em seguida, vamos olhar para criação de modelos preditivos.

5. Construção de um Modelo Preditivo em Python

Após tornarmos os dados úteis para a modelagem, vamos agora olhar para 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 vamos seguir a trilha. Convido você a fazerem uma reciclagem sobre sklearn através deste artigo.

Uma vez que sklearn exige que todas as entradas estejam numéricas, devemos converter todas as variáveis categóricas em numéricas através da codificação das categorias. Isso pode ser feito usando o seguinte código:

```
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
```

Em seguida vamos importar os módulos necessários. Então, vamos definir uma função de classificação genérica, o que leva um modelo como entrada e determina a pontuação de precisão e de validação cruzada. Uma vez que este é um artigo introdutório, não vou entrar em detalhes de codificação. Por favor, consulte este artigo para obter detalhes sobre os algoritmos com códigos R e Python. Além disso, seria bom obter também uma reciclagem sobre validação cruzada através deste artigo, pois é uma medida muito importante do desempenho de performance.

```
#Importa os modelos da biblioteca scikit learn:

from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold #For K-fold cross validation
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics

#Função genérica para fazer um modelo de classificação para avaliar performance
def classification_model(model, data, predictors, outcome):
  #Acerta 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("Accuracy : %s" % "{0:.3%}".format(accuracy))

#Performa validação cruzada k-fold com 5 folds
kf = KFold(data.shape[0], n_folds=5)
error = []
for train, test in kf:
  # 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("Cross-Validation Score : %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 fazer o primeiro modelo de regressão logística. Uma forma seria pegar todas as variáveis no modelo, mas isso pode resultar em overfitting (não se preocupe se você desconhece essa terminologia). Em palavras simples, utilizar todas as variáveis pode resultar em um modelo de compreensão das relações complexas específicas para estes dados, mas difícil de ser generalizado. Leia mais sobre Regressão Logística.

Podemos facilmente fazer algumas hipóteses intuitivas para fazer a bola rolar. A chance de conseguir um empréstimo será maior para:

1. Os candidatos que têm um histórico de crédito (lembra que observamos isso na exploração?)
2. Os candidatos com rendimento maior do solicitante e co-candidatos
3. Os candidatos com nível superior
4. Imóveis em áreas urbanas com perspectivas de alto crescimento

Então, vamos fazer o nosso primeiro modelo com 'histórico de crédito ".

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

Precisão: 80,945% Cross-Validação Score: 80,946%

```
#Podemos tentar diferentes combinações de variáveis:
predictor_var = ['Credit_History','Education','Married','Self_Employed','Property_A
```

Precisão: 80,945% Cross-Validação Score: 80,946%

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 modo. Temos duas opções agora:

1. Engenharia de Recursos: derivam-se novas informações e tenta-se prever esses. Vou deixar isso para a sua criatividade.
2. Técnicas de modelagem melhor. Vamos explorar esse lado.

Arvore de Decisão

Árvore de decisão é um outro método para de modelo preditivo conhecido por proporcionar maior precisão do que o modelo de regressão logística. Leia mais sobre Árvores de Decisão.

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

Precisão: 81,930% Score de Cross-Validação: 76,656%

Aqui, o modelo baseado em variáveis categóricas é incapaz de ter um impacto porque o histórico de crédito os está dominando. Vamos tentar algumas variáveis numéricas:

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

Precisão: 92,345% Score de Cross-Validação: 71,009%

Aqui observamos que, embora a precisão tenha subido ao adicionar as variáveis, o erro de validação cruzada caiu. Este é o resultado do modelo de sobre-ajustamento dos dados. 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. Leia mais sobre Floresta Aleatória.

Uma vantagem com a Floresta Aleatória é que podemos fazê-la funcionar com todas as características e retorna uma matriz de importância de recurso que pode ser usada para selecionar os recursos.

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

Precisão: 100,000% Score de Cross-Validação: 78,179%

Aqui vemos que a precisão é de 100% para o grupo de treinamento. Este é o caso de superajuste final 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 dar as características mais importantes.

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

```
Credit_History      0.273094
TotalIncome_log     0.264433
LoanAmount_log      0.229032
Dependents          0.050138
Property_Area       0.048979
Loan_Amount_Term    0.042681
Married             0.025823
Education           0.022426
Gender              0.021895
Self_Employed       0.021500
dtype: float64
```

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, r
predictor_var = ['TotalIncome_log','LoanAmount_log','Credit_History','Dependents','
classification_model(model, df,predictor_var,outcome_var)
```

Precisão: 82,899% Score de Cross-Validação: 81,461%

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

Mesmo depois de alguns ajustes nos parâmetros fundamentais na floresta aleatória, atingimos uma precisão de validação cruzada apenas ligeiramente melhor do que o modelo de regressão logística original. Este exercício nos dá 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 características para atender melhor o modelo.

Então você está pronto para assumir o desafio? Comece sua viagem a ciência de dados com o Problema de previsão de empréstimos.

Notas Finais

Espero que este tutorial tenha ajudado a maximizar a sua eficiência no início da atividade de Data Science com Python. Estou certo de que isso não só lhe deu uma ideia sobre métodos básicos de análise de dados, mas também mostrou como implementar algumas das técnicas mais sofisticadas disponíveis hoje.

Python é realmente uma ótima ferramenta e está se tornando uma linguagem cada vez mais popular entre os cientistas de dados. A razão de ser é que fácil de aprender e se integra bem com outras bases de dados e ferramentas como Spark e Hadoop. Majoritariamente, tem grande intensidade computacional e poderosas bibliotecas de análise de dados.

Assim, aprender Python serve para realizar o ciclo de vida completo de qualquer projeto de Data Science, incluindo leitura, análise, visualização e finalmente previsões.

Veja também:

- Guia de acesso rápido – Python para Data Science – Matplotlib
- Python PrettyTable
- Participantes do mercado de pagamentos

PUBLICADO EM Python

MARCADO bokeh Data Science pandas python scypy seaborn

7 comentários em “Um tutorial completo para aprender Data Science com Python do zero”

osmar diz:
27 de fevereiro de 2017 às 01:46

Obrigado, foi muito útil e pratico.

Responder

Jefferson de Vasconcelos diz:
5 de julho de 2017 às 18:53

Osmar, parabéns pelo conteúdo. Porém não consegui fazer download do dataset no link que você indicou, mesmo realizando minha inscrição no site. Você poderia fornecer um link direto para download desse data set?

Responder

adminvooo diz:
15 de julho de 2017 às 12:19

Olá Jefferson,
Infelizmente não temos o link.

Responder

adminvooo diz:
3 de maio de 2018 às 20:46

Link para o arquivo está postado. Abs.

Responder

ELAINE LIMA A. XAVIER diz:
19 de setembro de 2017 às 22:00

Não consegui baixar o arquivo, não o encontrei. Tem como passa-lo de uma outra forma? Parabéns pelo post.

Responder

adminvooo diz:
21 de setembro de 2017 às 18:10

Olá,
Este texto é uma tradução do A Complete Tutorial to Learn Data

Science with Python from Scratch (link no artigo). Você pode tentar obter lá.

Responder

adminvooo diz:
3 de maio de 2018 às 20:46

Link para o arquivo está postado. Abs.

Responder

Deixe uma resposta

COMENTÁRIO

NOME *

E-MAIL *

SITE

NOTIFIQUE-ME SOBRE NOVOS COMENTÁRIOS POR E-MAIL.

NOTIFIQUE-ME SOBRE NOVAS PUBLICAÇÕES POR E-MAIL.

PUBLICAR COMENTÁRIO

Pesquisa

Posts recentes

Tutorial – Compreensão de listas Python, com exemplos

Expressões regulares (regex)

BeautifulSoup

Regressão logística

Analytics

Arquivos

- março 2019
- janeiro 2019
- novembro 2018
- setembro 2018
- junho 2018
- maio 2018
- abril 2018
- janeiro 2018
- dezembro 2017
- setembro 2017
- agosto 2017
- junho 2017
- maio 2017
- abril 2017
- março 2017

- dezembro 2016
- novembro 2016
- outubro 2016
- setembro 2016
- agosto 2016
- julho 2016
- maio 2016
- abril 2016