

BUSCA



Início



E-books (em breve)



Sobre



Contato

Se junte a nossa comunidade e receba atualizações de artigos, tutoriais e muito mais!



✉ Insira seu email aqui

QUERO RECEBER!



Início > 2017 > setembro > 26 > Dominando o Pandas: A Biblioteca para Análise de Dados preferida entre os Cientistas de Dados (Parte 1)



Dominando o Pandas: A Biblioteca para Análise de Dados preferida entre os Cientistas de Dados (Parte 1)



RODRIGO SANTANA



SETEMBRO 26, 2017



2 COMMENTS



DATA ANALYSIS, FERRAMENTAS E SCRIPTS



3

Para fazer Data Science com Python o **Pandas** talvez seja uma biblioteca obrigatória 😊 (eu realmente gosto dessa biblioteca).

Já falamos sobre o Pandas aqui neste blog várias vezes, mas não dedicamos nenhum artigo todo a essa lib.

Neste vamos abordar alguns recursos bem interessantes.

O Pandas é um biblioteca open source amplamente utilizada na comunidade acadêmica.

Esta se tornou extremamente útil pelo seu desempenho e pela sua capacidade de simplificar tarefas complicadas de manipulação de dados (complexo isso hein?)

Resumo da ópera:

- [Scripts e Dataset](#)
- [Mão na Massa!](#)
- [Consultando o Dataset](#)
- [Alterando um dataset](#)
- [Removendo Dados](#)
- [Missing Values](#)
- [Visualização de Dados](#)

Se você já gostou do assunto desse artigo, não deixe de compartilhar com seus amigos para que cada vez mais pessoas aprendam sobre usar o Pandas 😊

Aproveite e assine nossa lista para ficar sempre atualizado com novas postagens.

Coloque o seu e-mail abaixo para
receber gratuitamente as
atualizações do blog!

✉ Seu melhor email

QUERO RECEBER!



Scripts e Dataset

Para este artigo escolhi um dataset bem interessante que pode ser baixado gratuitamente no site do [Kaggle](#).

Essa base de dados é uma base de informações de preço e diversos outros atributos sobre imóveis nos EUA.

Para ficar fácil, disponibilizo a base aqui juntamente com os Scripts (notebook) utilizados nesse artigo.

Base de dados : [hc_house_data](#)

Scripts: [Scripts Artigo \(Parte1\)](#)

É só baixar a base de dados e os scripts para acompanhar tudo que foi feito nesse artigo 😊

Mão na massa!

Se não conhece essa biblioteca, ou não tem instalada, [AQUI](#) ensinamos como instalar

São 2 minutos, coisa rápida..:)

Pronto!

Com a biblioteca instalada, o primeiro a ser feito é a importação:

```
1 import pandas as pd
```

O segundo passo é carregar o dataset em memória. O pandas faz isso de forma muito eficiente, e como esta base está no formato **csv** basta usar o método **read_csv()**, veja:

```
1 dataset = pd.read_csv('C:\scripts\minerandodados\Pandas - Data Science\housesalespredictio
```

Com o comando acima, mandamos carregar o conteúdo do arquivo para a variável **dataset** passando o caminho do arquivo e dizendo qual é o **separador(sep=',')** utilizado entre as colunas, que nesse caso é a vírgula.

A variável dataset agora se tornou um dataframe.

O Dataframe é uma estrutura de dados com duas dimensões onde linhas podem ter colunas de tipos diferentes. Veja o tipo da variável:

```
1 type(dataset)
```

Os dataframes são objetos extremamente flexíveis, podemos armazenar listas, vetores e até outros dataframes.

Para ver se a base de dados foi carregada corretamente, é interessante imprimir algumas linhas, o método **head()** nos permite isso, veja:

```
1 dataset.head()
```

In [3]: `dataset.head()`

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_b
0	7129300520	20141013T000000	221900.0	3.0	1.00	1180	5650	1.0	0	0	...	7	1180	0
1	6414100192	20141209T000000	538000.0	3.0	2.25	2570	7242	2.0	0	0	...	7	2170	400
2	5631500400	20150225T000000	180000.0	2.0	1.00	770	10000	1.0	0	0	...	6	770	0
3	2487200875	20141209T000000	604000.0	4.0	3.00	1960	5000	1.0	0	0	...	7	1050	910
4	1954400510	20150218T000000	510000.0	3.0	2.00	1680	8080	1.0	0	0	...	8	1680	0

5 rows × 15 columns

Por padrão este exibe as colunas e as 5 primeiras linhas do dataset, caso queira imprimir mais linhas ou menos passe o número como parâmetro.

Vimos que a base foi lida corretamente, as colunas estão destacadas em negrito e as linhas em cor normal.

Se quisermos imprimir todas colunas do nosso dataset, também é fácil, veja:

1 `dataset.columns`

E se quisermos saber quantas linhas tem a base de dados? Use o método **count()**, veja:

1 `dataset.count()`

Esse método exibe a quantidade de linhas da base e de cada coluna, é bem útil.

Para fazer Data Science muitas vezes precisamos de informações estatísticas da base de dados. Informações estas como: desvio padrão, média, valor mínimo e máximo de colunas, entre outras.

Com o pandas isso é fácil de conseguir com o método **describe()**, veja:

1 `dataset.describe()`In [86]: `dataset.describe()`

Out[86]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	conc
count	2.161300e+04	2.161300e+04	21609.000000	21613.000000	21613.000000	2.161300e+04	21612.000000	21613.000000	21613.000000	21613.000000
mean	4.580302e+09	5.400881e+05	3.370910	2.114757	2079.899736	1.510697e+04	1.494332	0.007542	0.234303	3.405
std	2.876566e+09	3.671272e+05	0.930084	0.770163	918.440897	4.142051e+04	0.539991	0.086517	0.766318	0.650
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000

Veja quanta informação de forma rápida e fácil 😊

Vamos fazer algumas consultas em nosso dataset.

Consultando o Dataset

Como já visto, esse dataset contém informações de imóveis dos EUA da cidade de Seattle.

Temos 19 colunas, entre elas temos: o preço do imóvel(price), a quantidade de quartos.bedrooms), a quantidade de banheiros(bathrooms), se é beira-mar(waterfront), quantidade de andares(floors), a condição média(condition) entre outras.

Se quisermos saber a quantidade de imóveis por tamanho de quartos, o comando abaixo nos mostra isso:

```
1 pd.value_counts(dataset['bedrooms'])
```

Perceba que o método **value_counts()** agrupa as informações e faz a soma por linhas do dataset.

E se quisermos imprimir somente os imóveis que tem 3 quartos ?

```
1 dataset.loc[dataset['bedrooms']==3]
```

In [95]: dataset.loc[dataset['bedrooms']==3]

Out[95]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basem
0	7129300520	20141013T000000	221900.0	3.0	1.00	1180	5650	1.0	0	0	...	7	1180	0
1	6414100192	20141209T000000	538000.0	3.0	2.25	2570	7242	2.0	0	0	...	7	2170	400
4	1954400510	20150218T000000	510000.0	3.0	2.00	1680	8080	1.0	0	0	...	8	1680	0
6	1321400060	20140627T000000	257500.0	3.0	2.25	1715	6819	2.0	0	0	...	7	1715	0
7	2008000270	20150115T000000	291850.0	3.0	1.50	1060	9711	1.0	0	0	...	7	1060	0
8	2414600128	20150415T000000	229500.0	3.0	1.00	1780	7470	1.0	0	0	...	7	1050	730
9	3793500160	20150312T000000	323000.0	3.0	2.50	1890	6560	2.0	0	0	...	7	1890	0
10	1736800520	20150403T000000	662500.0	3.0	2.50	3560	9796	1.0	0	0	...	8	1860	1700
12	114101516	20140528T000000	310000.0	3.0	1.00	1430	19901	1.5	0	0	...	7	1430	0
13	6054650070	20141007T000000	400000.0	3.0	1.75	1370	9680	1.0	0	0	...	7	1370	0

Para consultar os dados baseado nas condições acima, usei o método **loc()**.

Esse método é usado para visualizar informações do dataset, este recebe uma lista por parâmetro para consulta. Isso é similar ao operador “where” da linguagem SQL

E se quisermos imprimir somente os imóveis que tem 3 quartos e com o número de banheiros maior que 2 ?

```
1 dataset.loc[(dataset['bedrooms']==3) & (dataset['bathrooms'] > 2)]
```

In [43]: dataset.loc[(dataset['bedrooms']==3) & (dataset['bathrooms'] > 2)]

21584	952008823	20141202T000000	380000.0	3	2.50	1260	900	2.0	0	0	...	7	940	320
21585	3832050760	20140828T000000	270000.0	3	2.50	1870	5000	2.0	0	0	...	7	1870	0
21587	6632300207	20150305T000000	385000.0	3	2.50	1520	1488	3.0	0	0	...	8	1520	0
21589	7570050450	20140910T000000	347500.0	3	2.50	2540	4760	2.0	0	0	...	8	2540	0
21592	1931300412	20150416T000000	475000.0	3	2.25	1190	1200	3.0	0	0	...	8	1190	0
21601	5100403806	20150407T000000	467000.0	3	2.50	1425	1179	3.0	0	0	...	8	1425	0
21603	7852140040	20140825T000000	507250.0	3	2.50	2270	5536	2.0	0	0	...	8	2270	0
21607	2997800021	20150219T000000	475000.0	3	2.50	1310	1294	2.0	0	0	...	8	1180	130
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0

4277 rows x 21 columns

Bastou apenas adicionar o operador **&** e colocar as condições desejadas entre parênteses.

Podemos consultar os dados de diversas formas, vejamos agora como ordenar o retorno de uma query

```
1 dataset.sort_values(by='price', ascending=False)
```

O comando acima utiliza o método **sort_values()** para ordenar a exibição do dataset pela coluna 'price' em ordem decrescente, ou seja, do maior para o menor.

Vejamos como é fácil e como o pandas é rápido.

Agora se quisermos contar a quantidade de imóveis que tem 4 quartos?

```
1 dataset[dataset['bedrooms']==4].count()
```

Basta filtrar pela coluna 'bedrooms' e utilizar o método **count()**

Alterando um dataset

É bem comum em projetos de Data Science ter a necessidade de fazer processamento dos dados.

Processamento estes tais como, remover colunas de um dataset, adicionar colunas, fazer cálculos etc.

Imagine que precisamos adicionar uma coluna que seria equivalente ao tamanho de um imóvel em metros quadrados.

Para exemplificar, vou criar uma coluna chamada "size", esta será o resultado da multiplicação do número de quartos (bedrooms) por 20.

Como podemos fazer isso?

```
1 dataset['size'] = (dataset['bedrooms'] * 20)
```

Pronto, nossa coluna "size" foi criada, veja:

```
1 dataset['size']
```

```
In [6]: dataset['size']
Out[6]: 0      60.0
        1      60.0
        2      40.0
        3      80.0
        4      60.0
        5      80.0
        6      60.0
        7      60.0
        8      60.0
        9      60.0
       10      60.0
       11      40.0
       12      60.0
       13      60.0
       14     100.0
       15      80.0
```

Outro recurso interessante do pandas é o método **apply()**. Este é um método que permite que passamos uma função que deve ser aplicada em todas as linhas ou colunas de um dataframe.

Vejamos a um exemplo..

Imagine que quero criar uma outra coluna no meu dataframe, nesta coluna quero categorizar os imóveis entre Big(Grande), Medium(Médio) e Small(Pequeno)

Para fazer isso vou criar uma função simples que testa se o tamanho for maior ou igual a 40 este imóvel é pequeno, se for maior ou igual a 60 este é médio, ou se este é maior ou igual a 80 é grande.

Criando a função 'categoriza':

```
1 def categoriza(s):
2     if s >= 80:
3         return 'Big'
4     elif s >= 60:
5         return 'Medium'
6     elif s >= 40:
7         return 'Small'
```

Ok, com a função criada, basta passar os dados para a função usando o método **apply()**, veja:

```
1 dataset['cat_size'] = dataset['size'].apply(categoriza)
```

Pronto, a coluna "cat_size" foi criada a partir da aplicação da função "categoriza" a cada linha da coluna "size"

Veja os valores da coluna nova coluna "cat_size":

```
1 dataset['cat_size']
```

```
In [9]: dataset['cat_size']
Out[9]: 0      Medium
1      Medium
2      Small
3      Big
4      Medium
5      Big
6      Medium
7      Medium
0      Medium
```

Para ver a distribuição dos valores da coluna use o método **value_counts()**, veja:

```
1 pd.value_counts(dataset['cat_size'])
```

```
In [9]: #Ver a distribuicao da coluna
pd.value_counts(dataset['cat_size'])
Out[9]: Medium    9822
Big             8816
Small          2759
Name: cat_size, dtype: int64
```

Removendo Dados

Caso queira remover uma coluna, use o método **drop()**.

Esse método usamos para apagar colunas ou linhas de um dataset, veja:

Apagando a coluna recém criada “cat_size”:

```
1 dataset.drop(['cat_size'], axis=1, inplace=True)
```

No comando acima, usamos o parâmetro ‘axis=1’ que significa que queremos apagar uma coluna e não uma linha.

O parâmetro ‘inplace’ é usado para persistir a alteração na variável dataset em memória, ou seja, sem ele por padrão, a alteração não seria efetivada no objeto dataset.

Se quisermos apagar linhas é simples, basta passar por parâmetro para o método drop a condição, veja:

```
1 dataset.drop(dataset[dataset.bedrooms==0].index ,inplace=True)
```

No exemplo acima, removemos as linhas que tem a condição de número de quartos igual a 0, ou seja, imóveis sem quartos.

A diferença aqui é o método “index”, esse método retorna a posição das linhas que o dataframe (dataset.bedrooms==0) retorna. O método drop utiliza essas posições para remover as linhas.

Se quisermos apagar os imóveis que tem mais de 30 quartos é só mudar o operador, veja:

```
1 dataset.drop(dataset[dataset.bedrooms>30].index ,inplace=True)
```

Missing Values



É muito comum pra quem trabalha com bases de dados, encontrar valores nulos ou faltantes.

Isso pode acontecer por diversas razões como erro em coleta de dados, dados exportados com erros etc.

Como lidar com esses problemas?

O pandas possui funções interessantes para lidar com valores nulos ou faltantes em bases de dados.

O Pandas nos fornece o método `isnull()` que faz uma verificação em todas as linhas e colunas do dataset em busca de valores faltantes ou nulos. Veja:

```
1 dataset.isnull().sum()
```

Checking Miss values

```
In [23]: dataset.isnull().sum()
```

```
Out[23]: id          0
         date        0
         price       0
         bedrooms    4
         bathrooms   0
         sqft_living  0
         sqft_lot     0
         floors      1
         waterfront  0
         view        0
         condition   0
         grade       0
         sqft_above   0
         sqft_basement 0
         yr_built     0
         yr_renovated 0
         zipcode      0
         lat         0
         long        0
         sqft_living15 0
         sqft_lot15   0
         dtype: int64
```

Adicionei o método **sum** pois, este nos retorna a quantidade valores nulos ou faltantes por colunas.

Veja que temos alguns dados faltantes, 4 na coluna “bedrooms” e 1 na coluna “floors”. Como resolver isso?

O pandas te dá algumas formas de resolver isso, uma delas é **remove** todas as linhas onde tem colunas com valores faltantes.

Essa alternativa pode ser ruim se tivermos muitas linhas faltantes, pois, removê-las iria talvez afetar muito o dataset. Veja como fazer isso:

```
1 dataset.dropna(inplace=True)
```

Ao usar o método **dropna()** o pandas remove todas as linhas onde tem colunas com valores faltantes, como explicado acima. O parâmetro `inplace=True` é para as modificações serem persistidas em memória.

Execute novamente o comando para checar os valores faltantes e veja que as linhas foram removidas

```
1 dataset.isnull().sum()
```

```
In [27]: dataset.isnull().sum()
```

```
Out[27]: id            0
         date          0
         price         0
         bedrooms      0
         bathrooms     0
         sqft_living    0
         sqft_lot       0
         floors         0
         waterfront     0
         view          0
         condition     0
         grade         0
         sqft_above     0
         sqft_basement  0
         yr_built       0
         yr_renovated   0
         zipcode        0
         lat            0
         long           0
         sqft_living15  0
         sqft_lot15     0
         dtype: int64
```

É possível também remover somente linhas que estejam com valores faltantes em todas as colunas, veja como fazer isso:

```
1 dataset.dropna(how='all', inplace=True)
```

Outra recurso seria preencher tais valores faltantes com média das colunas, isso talvez fosse menos radical e afetasse menos dataset em casos onde temos poucos valores faltantes. Veja um exemplo:

```
1 dataset['floors'].fillna(dataset['floors'].mean(), inplace=True)
```

No comando acima preenchemos o valor faltante da coluna “floors” com a média dos seus valores. Rode o comando com o método **isnull()** e veja que agora a coluna floors ficou 0

Outra alternativa é preencher os valores faltantes com uma constante, veja esse exemplo:

```
1 dataset['bedrooms'].fillna(1, inplace=True)
```

O comando acima, preenche os valores faltantes da coluna “bedrooms” com o valor 1.

Visualização de Dados



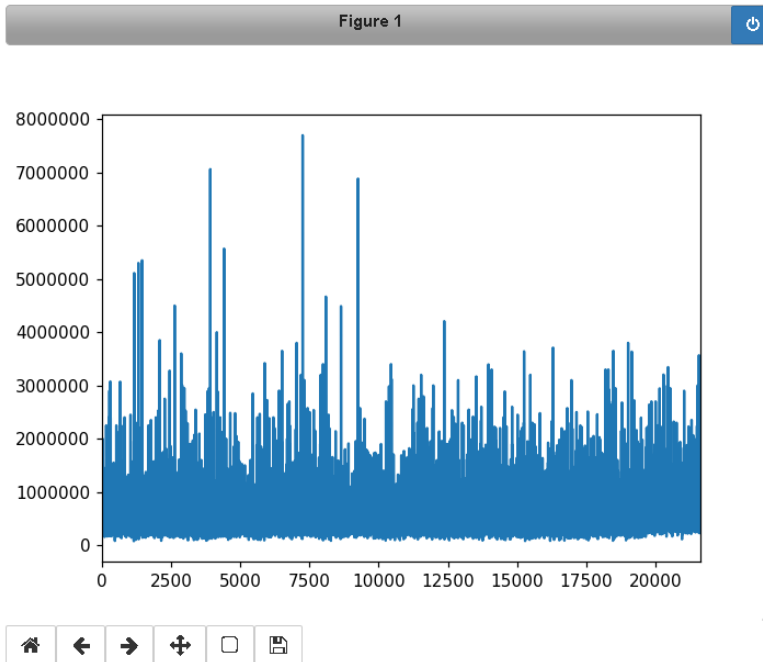
O pandas utiliza o matplotlib para permitir a plotagem de gráficos usando Dataframes e Séries.

Esse é mais uma jogada sensacional dessa lib 😊

Veja como podemos plotar gráficos simples que já nos ajudam na visualização dos nossos dados.

```
1 %matplotlib notebook
2
3 dataset['price'].plot()
```

In [36]: dataset['price'].plot()



Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0xd56e278>

No comando acima plotei a apenas a coluna preço e o pandas colocou no eixo x a quantidade de linhas do dataframe.

Isso aconteceu pois, o dataframe está indexado pela seu número de linhas.

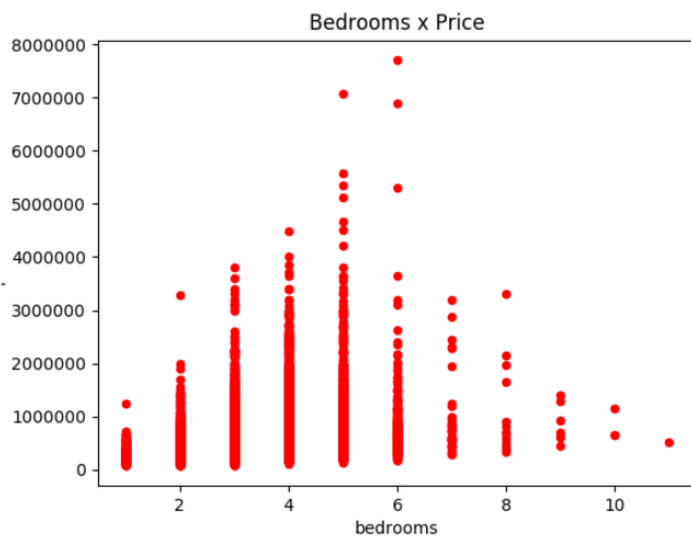
Simple né? Mas já mostra alguns insights interessantes, podemos ver que a maioria dos imóveis ficam na faixa de 100 a 250 mil.

Obs: o comando **%matplotlib notebook** é usado para permitir que o gráfico seja plotado na célula do notebook, pois, nesse caso estamos usando o **Jupyter notebook** 😊

Veja mais um exemplo abaixo:

```
1 dataset.plot(x='bedrooms',y='price',kind='scatter', title='Bedrooms x Price',color='r')
```

```
In [62]: dataset.plot(x='bedrooms',y='price',kind='scatter', title='Bedrooms x Price',color='r')
```



```
Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x10ea7400>
```

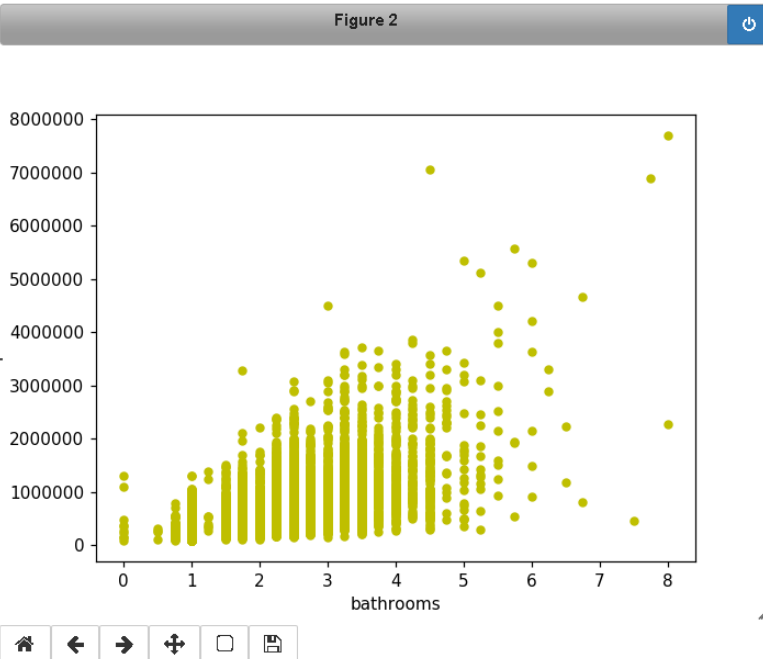
O comando acima plotei no eixo x o número de quartos (bedrooms) e no eixo y o preço dos imóveis(price), no parâmetro “kind” informo que quero um gráfico do tipo scatter.

Por fim informo um título para o gráfico (title) e defino a cor com o parâmetro color = r (red)

Vejamos outro exemplo:

```
1 dataset.plot(x='bathrooms',y='price',kind='scatter',color='y')
```

```
In [37]: dataset.plot(x='bathrooms',y='price',kind='scatter',color='y')
```



```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0xa4456a0>
```

O gráfico acima é quase idêntico ao anterior. A diferença aqui são as colunas, onde colocamos no eixo x o número de banheiros (bathrooms) e no eixo y o preço dos imóveis (price)

Bem interessante né?

Com uma linha de código podemos plotar um gráfico simples que mostra uma informação útil do nosso dataset.

Conclusão

Nesse artigo vimos o poder do Pandas.

Vimos que a biblioteca é bem poderosa e que pode nos ajudar muito em projetos de Data Science.

Se você gostou desse artigo compartilhe com seus amigos para que mais pessoas possam aprender e trabalhar com o pandas.

Deixe seu comentário abaixo e nos conte o que achou deste artigo.

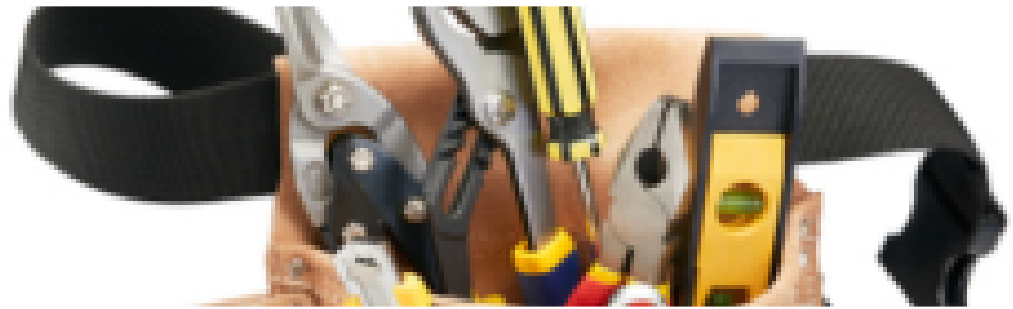
Forte Abraço!

Sobre **Rodrigo Santana**



Mestrando em Ciência da Computação, interessado em Machine Learning, NLP e Data Science.

Artigos relacionados



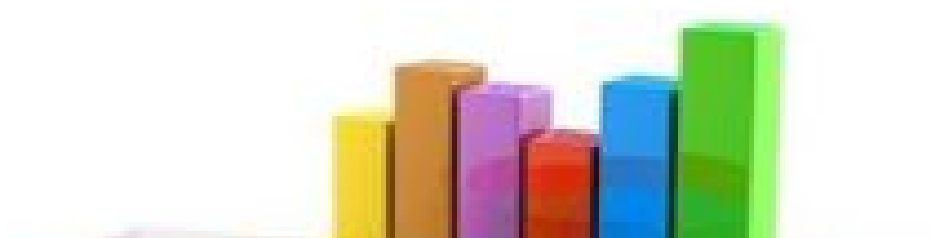
10 Melhores Ferramentas e Bibliotecas para Trabalhar com Mineração de Dados e Big Data



Exploratory Data Analysis (EDA): Aprenda Definitivamente como Extrair Valiosos Insights de Bases de Dados Reais

Text mining makes information
extraction from huge volume

Mineração de Textos: 7 Técnicas e Aplicações para Você Extrair Valor dos Dados e Alavancar Suas Análises.



7 Tipos de Gráficos que Todo Cientista de Dados Deve Conhecer



ARTIGO ANTERIOR

[Café com Código #06: Introdução a Machine Learning com Scikit-Learn](#)


PRÓXIMO ARTIGO



[Café com Código #07: RapidMiner: Data Science sem escrever uma linha de código](#)

2 COMENTÁRIOS

www.minerandodados.com.br

 Iniciar sessão ▾ Recomendar 1 Partilhar

Mostrar primeiro os mais votados ▾



Escreva o seu comentário...

INICIE SESSÃO COM O

OU REGISTE-SE NO DISQUS **Gustavo Vzfo** • há 19 dias

Parabéns Rodrigo pela iniciativa estou aprendendo muito com seus artigos.

acho que tem uma errata na parte que você explica sobre o método .apply foi definido uma função como categoriza(s) e foi colocado assim:
dataset['cat_size'] = dataset['size'].apply(set_size) o correto seria assim?

dataset['cat_size'] = dataset['size'].apply(categoreiza)

^ | ▾ • Responder • Partilhar ›

**Rodrigo Santana Ferreira** Moderador ➔ Gustavo Vzfo • há 19 dias

Grande Gustavo!

Que bom que você está gostando do blog :)

Realmente teve uma falha minha nessa linha, corrigir agora ;)

Obrigado pela observação e comentário, um abraço!

^ | ▾ • Responder • Partilhar ›

 Subscrever Acerca do DisqusAdicionar o DisqusAdicionar Privacidade

MINERADOR

Eu sou o **Minerador** e vou te ensinar sobre **Mineração de dados**, **Big Data** e muito mais...prepare suas ferramentas!

SAIBA MAIS +



Machine Learning



Data Analysis

► Café com Código



Ferramentas

🔍 Buscar por:



Minerando Dados · 2017 © Todos os direitos reservados

