



ENGENHARIA DE *SOFTWARE* PARA INTELIGÊNCIA ARTIFICIAL

UNIDADE IV VISUALIZAÇÃO DE DADOS

Elaboração

Diogo Santos Silva da Costa

Atualização

Alexander Francisco Vargas Salgado

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

SUMÁRIO

UNIDADE IV

VISUALIZAÇÃO DE DADOS.....	5
----------------------------	---

CAPÍTULO 1

MATPLOTLIB.....	5
-----------------	---

CAPÍTULO 2

BOKEH.....	8
------------	---

REFERÊNCIAS	15
-------------------	----

CAPÍTULO 1

MATPLOTLIB

Conhecendo a biblioteca

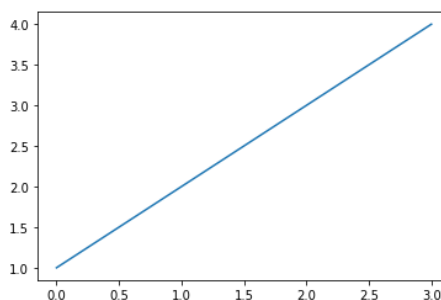
Nesse capítulo, vamos falar sobre visualização de dados iniciando o assunto com uma biblioteca desenvolvida para Python bastante simples de se entender e principalmente de usar, nesse momento o que queremos é plotar simples gráficos 2-D para entender os princípios aqui utilizados.

Matplotlib é uma coleção de funções no estilo de comando que faz o matplotlib funcionar como o MATLAB. Cada função pyplot faz alguma alteração em uma figura: por exemplo, cria uma figura, cria uma área de plotagem em uma figura, plota algumas linhas em uma área de plotagem, decora a plotagem com rótulos, etc.

A geração de visualizações com o pyplot é muito rápida:

```
1 import matplotlib.pyplot as plt
2 plt.plot([1, 2, 3, 4])
3 plt.show()
```

Gráfico 2. Plotagem de reta simples.



Fonte: Elaboração própria do autor, 2020.

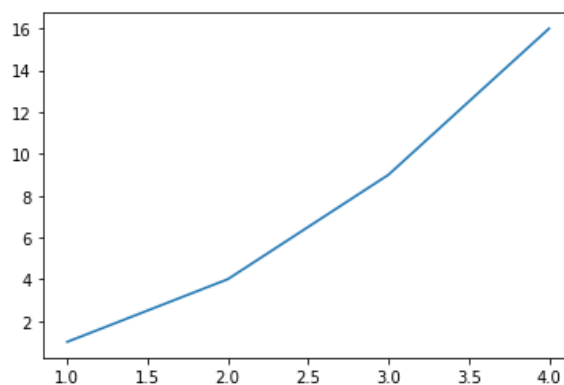
Você pode estar se perguntando por que o eixo x varia de 0 a 3 e o eixo y de 1 a 4. Se você fornecer uma única lista ou matriz ao plot() comando, o matplotlib assume que é

uma sequência de valores y e gera automaticamente os valores x para você. Como os intervalos python começam com 0, o vetor x padrão tem o mesmo comprimento que y , mas começa com 0. Portanto, os dados x são $[0,1,2,3]$.

Assim, `plot()` é um comando versátil e levará um número arbitrário de argumentos. Por exemplo, para plotar x *versus* y , você pode emitir o comando:

```
1 plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
[<matplotlib.lines.Line2D at 0x1ffb429eec8>]
```

Gráfico 3 – Plotagem de reta composta.

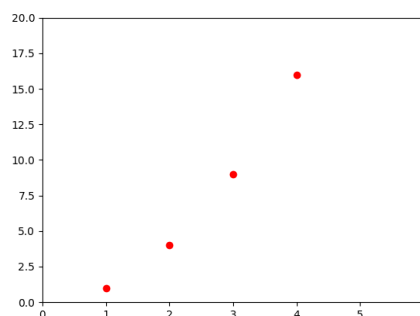


Fonte: Elaboração própria do autor, 2020.

Para cada par de argumentos x , y , existe um terceiro argumento opcional, que é a *string* de formato que indica a cor e o tipo de linha do gráfico. As letras e os símbolos da sequência de formatação são do MATLAB e você concatena uma sequência de cores com uma sequência de estilo de linha. A sequência de formato padrão é 'b-', que é uma linha azul sólida. Por exemplo, para plotar o esquema acima com círculos vermelhos, você emitiria:

```
1 plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
2 plt.axis([0, 6, 0, 20])
3 plt.show()
```

Gráfico 4 – Plotagem de pontos destacados.



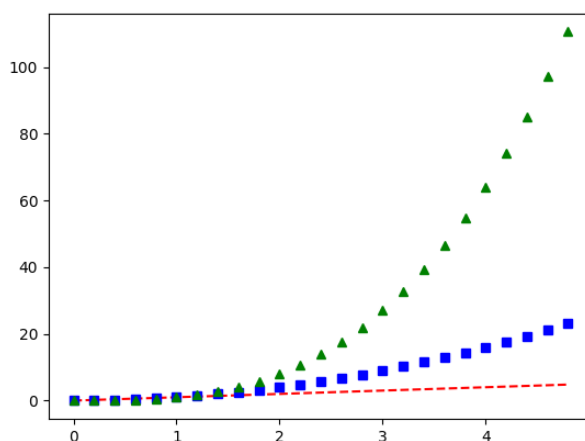
Fonte: Elaboração própria do autor, 2020.

Consulte a documentação da função `plot()` para obter uma lista completa dos estilos de linha e sequências de formato. O `axis()` comando no exemplo acima pega uma lista e especifica a janela de exibição dos eixos [`xmin`, `xmax`, `ymin`, `ymax`].

Se o `matplotlib` estivesse limitado ao trabalho com listas, seria bastante inútil para o processamento numérico. Geralmente, você usará matrizes `numpy`. De fato, todas as sequências são convertidas em matrizes `numpy` internamente. O exemplo a seguir ilustra uma plotagem de várias linhas com diferentes estilos de formato em um comando usando matrizes:

```
1 import numpy as np
2
3 # tempo de amostragem uniforme em intervalos de 200 ms
4 t = np.arange(0., 5., 0.2)
5
6 # traços vermelhos, quadrados azuis e triângulos verdes
7 plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
8 plt.show()
```

Gráfico 5. Plotagem de sequencias múltiplas.



Fonte: Elaboração própria do autor, 2020.

CAPÍTULO 2

BOKEH

Gráficos

A melhor forma de analisar os dados com os quais trabalharemos antes de escolher os parâmetros que nos interessam é com uma análise gráfica. Claro, muitas vezes combinamos com análises estatísticas, mas os gráficos ainda assim estão presentes, já que, dentro de um universo, permitem-nos uma visão do todo, incluindo o comportamento geral.


Não há muita teoria que possamos explorar sobre os gráficos neste curso, portanto é mais interessante irmos direto para as técnicas de construção deles. Vamos a um exemplo.

Como sempre, começamos importando as bibliotecas necessárias, aqui vamos trabalhar com uma biblioteca interativa chamada Bokeh. Assim:

```
1 import pandas as pd
2 from bokeh.io import output_notebook
3 from bokeh.plotting import figure, show
```

Como queremos visualizar os gráficos no nosso notebook, iniciaremos a biblioteca indicando que todos os gráficos devem ser gerados de forma embutida, para isso usamos a função `output_notebook`.

```
1 output_notebook()
```

 BokehJS 1.4.0 successfully loaded.

A seguir, carregaremos os dados com os quais vamos trabalhar e verificaremos a saída na tabela 22:

```
1 df = pd.read_excel('D:\Documentos\consolidado.xlsx')
2 df.tail(10)
```

Tabela 22. Saída dos dados com o método `head`.

	Unnamed: 0	ANO	AREA	CLASSE	OBJECTID	RESUMO	LARGURA
0	10811	2001	41666620000	Campo Antrópico	2057	Artificial	1195228908
1	10812	2001	10708700000	Solo Exposto	2058	Artificial	443936564
2	10813	2001	31352480000	Campo Antrópico	2059	Artificial	2510363662
3	10814	2001	13082380000	Campo Antrópico	2060	Artificial	474466226
4	10815	2001	57329300000000	Floresta	2061	Natural	239096878528
5	10816	2001	255663300000	Urb não consolid	2062	Artificial	2606700649
6	10817	2001	97423800000	Área Urbana	2063	Artificial	1900314385
7	10818	2001	631521000000	Campo Antrópico	2064	Artificial	5372474447
8	10819	2001	25192190000	Área Urbana	2065	NaN	778061236
9	10820	2001	1129063000000	Campo Antrópico	2066	NaN	10948839735

Fonte: Elaboração própria do autor, 2020.

Esses dados são da ocupação de solo anteriormente ocupado por áreas vegetais no município do Rio de Janeiro. Temos registros dos anos de 1988, 1992, 1996, 1999 e 2001. Os campos presentes são:

- » ano: ano da coleta de informações;
- » area: área ocupada pela cobertura vegetal em questão;
- » classe: forma de uso do solo;
- » objectid: um identificador do elemento;
- » resumo: indica se a ocupação do solo é natural, artificial ou vazia, quando a informação não é fornecida;
- » largura: largura da área em questão.

Vamos começar explorando a distribuição de registros ao longo dos anos.

Primeiro passo é contar o número de elementos agrupados por ano. Vamos separar apenas um campo de interesse, o campo `object_id`, e, claro, o ano, que é o campo em questão. Observamos a saída na Tabela 23:

```
1 registros_por_ano = df[["OBJECTID", "ANO"]].groupby("ANO").count().reset_index()
2 registros_por_ano.columns = ["ANO", "TOTAL"]
3 registros_por_ano.head()
```

Tabela 23. Saída dos dados com o método `head`.

	ANO	TOTAL
0	2001	10

Fonte: Elaboração própria do autor, 2020.

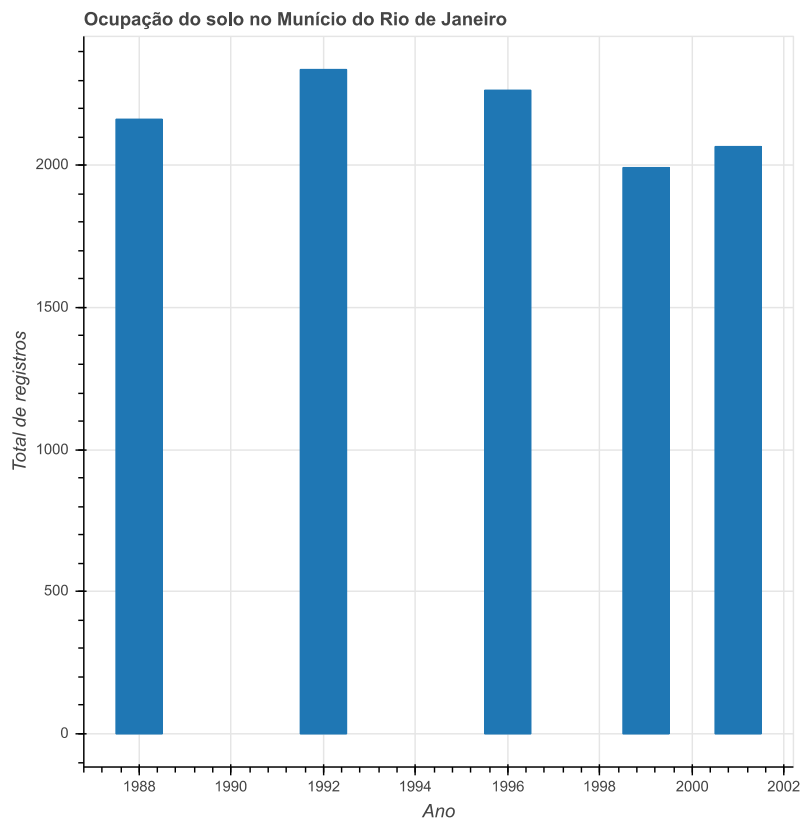
Agora estamos prontos para começar um histograma. Primeiro passo é criar uma figura. A figura é o espaço em que o gráfico será impresso. Nela, indicamos parâmetros como título, tamanho, etc. Nesse sentido:

```
1 p = figure(plot_height = 600,
2 plot_width = 600,
3 title = 'Ocupação do solo no Município do Rio de Janeiro',
4 x_axis_label = 'Ano',
5 y_axis_label = 'Total de registros')
```

Agora usamos o gráfico indicando que queremos criar um histograma com os dados que temos em mãos. O nosso histograma será um conjunto de retângulos criados com o método `vbar`, indicamos que:

```
1 p.vbar(top=registros_por_ano["TOTAL"], x=registros_por_ano["ANO"], width=1)
2 show(p)
```

Gráfico 6. Ocupação do solo no município do Rio de Janeiro. Gráfico do total de registros x ano.



Fonte: Elaboração própria do autor, 2019.

Com o histograma, é possível visualizar a evolução dos registros, mas não é suficiente, vamos avaliar a evolução de alguns tipos de ocupação.

```
1 df.CLASSE.unique()
array(['Campo Antrópico', 'Solo Exposto', 'Floresta', 'Urb não consolid',
      'Área Urbana'], dtype=object)
```

Começamos, por exemplo, com o tipo `Floresta`, cujos dados estão exibidos na tabela 24.

```
1 df_Floresta = df[df.CLASSE == 'Floresta']
2 df_Floresta.head(10)
```

Tabela 24. Saída dos dados com o método `head`.

Unnamed: 0	ANO	AREA	CLASSE	OBJECTID	RESUMO	LARGURA	
4	10815	2001	57329300000000	Floresta	2061	Natural	239096878528

Fonte: Elaboração própria do autor, 2020.

Vamos agrupar os registros ao longo dos anos, exibidos na tabela 25.

```
1 registros_por_ano = df_Floresta[["ANO", "OBJECTID"]].groupby("ANO").count().reset_index()
2 registros_por_ano.columns = ["ANO", "TOTAL"]
3 registros_por_ano.head()
```

Tabela 25. Saída dos dados com o método *head*.

	ANO	TOTAL
0	2001	1

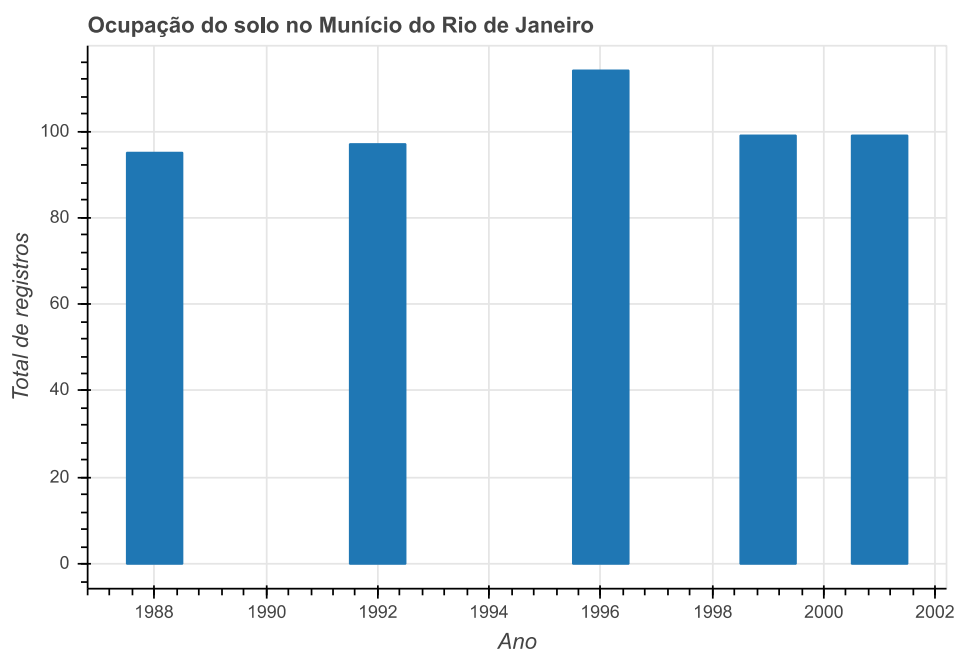
Fonte: Elaboração própria do autor, 2020.

```

1 p = figure(plot_height = 400,
2   plot_width = 600,
3   title = 'Ocupação do solo no Município do Rio de Janeiro',
4   x_axis_label = 'Ano',
5   y_axis_label = 'Total de registros')
6 p.vbar(top=registros_por_ano["TOTAL"], x=registros_por_ano["ANO"], width=1)
7 show(p)

```

Gráfico 7. Ocupação do solo no município do Rio de Janeiro. Gráfico do total de registros x ano.



Fonte: Elaboração própria do autor, 2020.

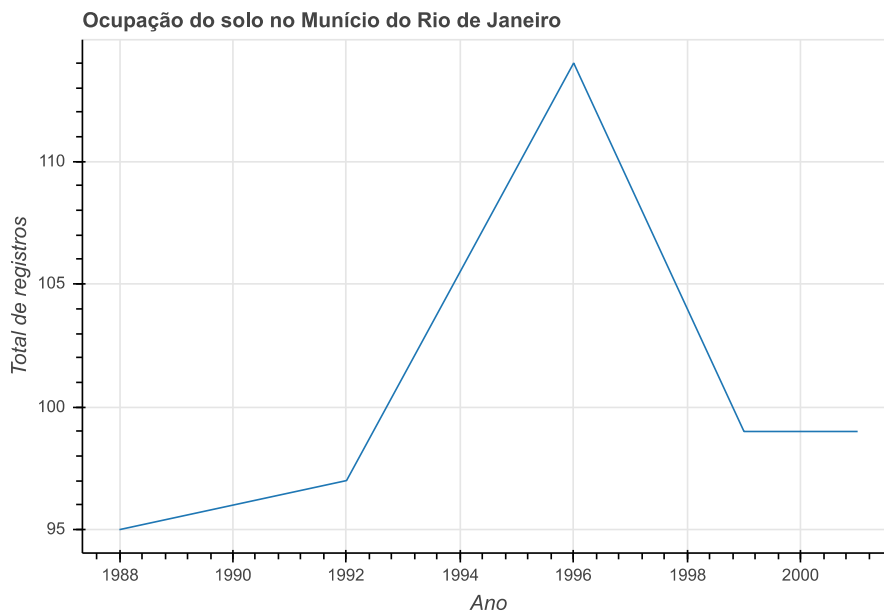
Uma outra forma de analisar essa evolução é com um gráfico de linhas. Dessa maneira, veja o código utilizado para tanto:

```

1 p = figure(plot_height = 400,
2   plot_width = 600,
3   title = 'Ocupação do solo no Município do Rio de Janeiro',
4   x_axis_label = 'Ano',
5   y_axis_label = 'Total de registros')
6 p.line(y=registros_por_ano["TOTAL"], x=registros_por_ano["ANO"])
7 show(p)

```

Gráfico 8. Ocupação do solo no município do Rio de Janeiro. Gráfico do total de registros x ano.

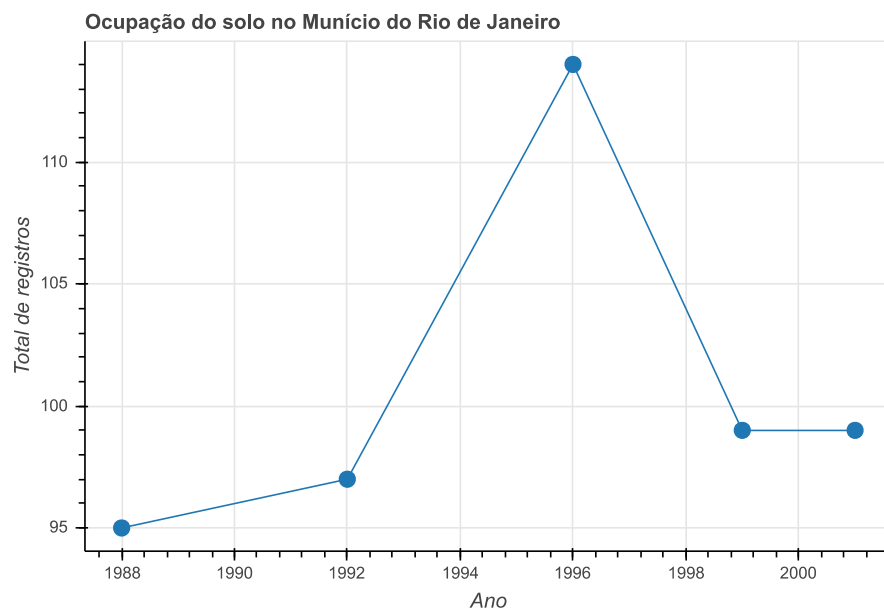


Fonte: Elaboração própria do autor, 2020.

Vamos melhorar incluindo os pontos correspondentes, para isso apenas adicionamos um novo gráfico na mesma figura.

```
1 p.circle(y=registros_por_ano["TOTAL"], x=registros_por_ano["ANO"], size=10)
2 show(p)
```

Gráfico 9. Ocupação do solo no município do Rio de Janeiro. Gráfico do total de registros x ano.



Fonte: Elaboração própria do autor, 2020.

Percebemos um crescimento do número de registros, que acelera bastante, contudo também é interessante analisar a área envolvida. Para isso, criaremos uma nova agregação,

somando as áreas em vez de contar identificadores e, em seguida, poderemos visualizá-los na Tabela 26.

```
1 area_por_ano = df_Floresta[["ANO", "AREA"]].groupby("ANO").sum().reset_index()
2 area_por_ano.head()
```

Tabela 26. Saída dos dados com o método *head*.

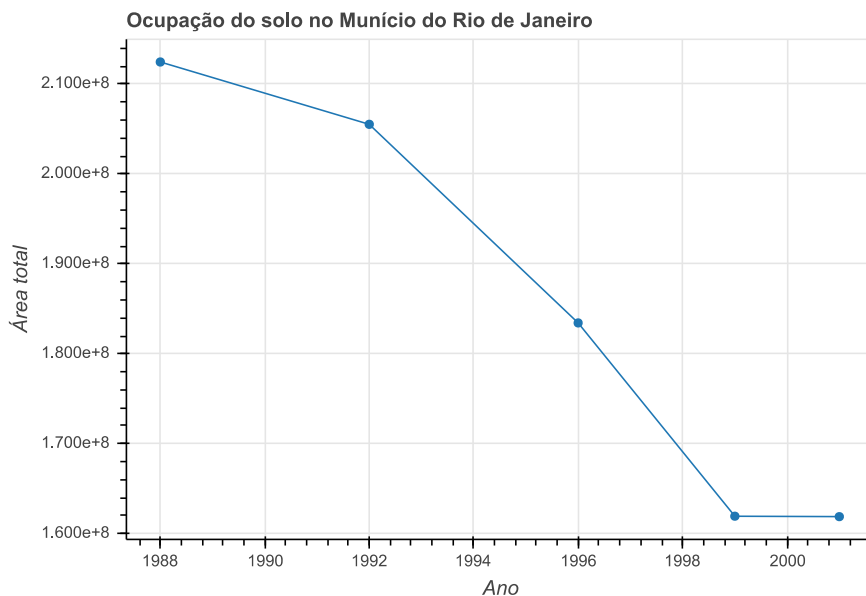
	ANO	AREA
0	2001	57329300000000

Fonte: Elaboração própria do autor, 2020.

Fazendo o gráfico:

```
1 p = figure(plot_height = 400,
2 plot_width = 600,
3 title = 'Ocupação do solo no Município do Rio de Janeiro',
4 x_axis_label = 'Ano',
5 y_axis_label = 'Área total')
6 p.line(y=area_por_ano["AREA"], x=area_por_ano["ANO"])
7 p.circle(y=area_por_ano["AREA"], x=area_por_ano["ANO"], size=5)
8 show(p)
```

Gráfico 10. Ocupação do solo no município do Rio de Janeiro. Gráfico da área total x ano.



Fonte: Elaboração própria do autor, 2020.

A área de ocupação florestal diminuiu consideravelmente enquanto o número de registros cresceu, mas se estabilizou quando o número de registros caiu. Vamos analisar o comportamento relacionado às duas variáveis e sua saída na tabela 27.

```
1 area_por_total = registros_por_ano.join(area_por_ano, rsuffix='_area')
2 area_por_total
```

Tabela 26. Saída dos dados com o método `head`.

	ANO	TOTAL	ANO_area	AREA
0	2001	1	2001	57329300000000

Fonte: Elaboração própria do autor, 2020.

Utilizaremos apenas as colunas *total* e *area*. A Tabela 28 demonstra a saída dos dados.

```
1 area_por_total = area_por_total[["TOTAL", "AREA"]]
2 area_por_total
```

Tabela 28. Saída dos dados com o método `head`.

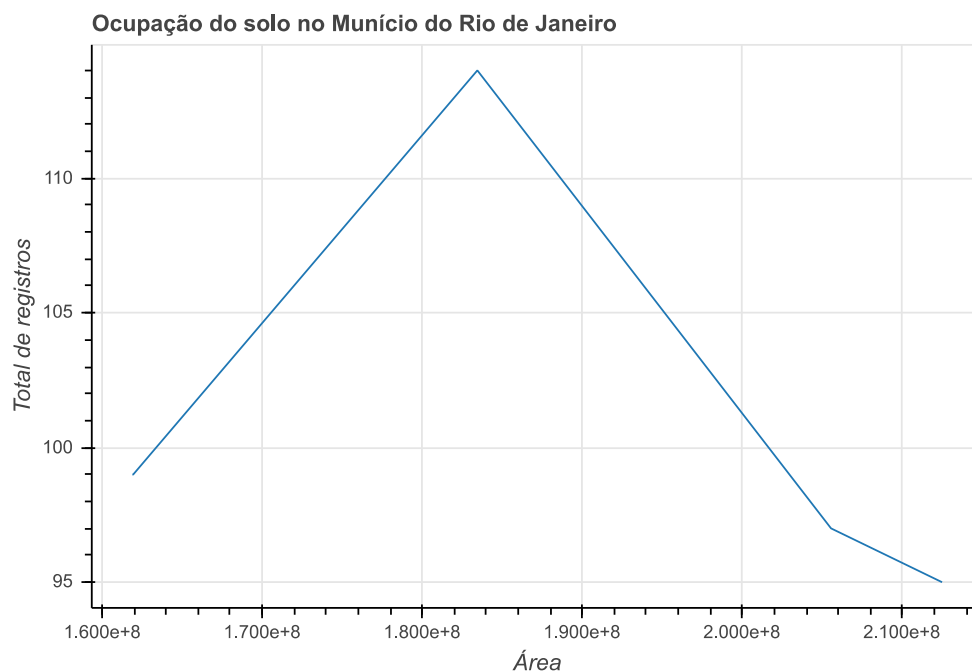
	TOTAL	AREA
0	1	57329300000000

Fonte: Elaboração própria do autor, 2020.

Por fim, analisaremos a relação entre as duas variáveis das duas áreas em questão.

```
1 p = figure(plot_height = 400,
2 plot_width = 600,
3 title = 'Ocupação do solo no Município do Rio de Janeiro',
4 x_axis_label = 'Área',
5 y_axis_label = 'Total de registros')
6 p.line(y=area_por_total["TOTAL"], x=area_por_total["AREA"])
7 show(p)
```

Gráfico 11. Ocupação do solo no município do Rio de Janeiro. Total de registro x área.



Fonte: Elaboração própria do autor, 2020.

REFERÊNCIAS

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to Algorithms**. 3. ed. Massachusetts, MA: MIT Press, 1.292 p. 2009.
- ELKNER, J.; DOWNEY, A.; MEYERS, C. **How to think like a computer scientist: learning with Python**. Wickford, UK: Samurai Media Limited, 306 p. 2016.
- GRUS, J. **Data science from scratch: first principles with python**. Sebastopol, CA: O'Reilly, 330 p. 2015.
- HERSTEIN, N.; WINTER, D. J. **Matrix Theory and Linear Algebra**. USA: Macmillan Pub. Co., 508 p. 1988.
- JOLLY, K. **Hands-on data visualization with Bokeh: interactive web plotting for Python using Bokeh**. Packt Publishing Ltd, 174 p. 2018.
- MADHAVAN, S. **Mastering Python for Data Science**. USA: Packt Publishing, 294 p. 2015.
- MÜLLER, A. C.; GUIDO, S. **Introduction to machine learning with Python: a guide for data scientists**. 1. ed. O'Reilly Media, 392 p. 2016.
- SEEDGEWICK, R.; WAYNE, K. **The textbook Algorithms**. 4, USA: Ed. Addison-Wesley Professional, 992 p. 2011.
- TATSUOKA, Maurice M.; LOHNES, Paul R. **Multivariate analysis: Techniques for educational and psychological research**. USA: Macmillan Publishing Co. Inc, 479 p. 1988.