

Ciência de dados com




Professor Rodrigo Bossini

Conteúdo

1	Introdução	1
2	Pré-processamento de dados	2
2.1	Bibliotecas	2
2.2	Importando os dados	3
2.3	Especificando as variáveis independentes e a variável dependente . .	3
2.4	Dados faltantes	4
2.5	Codificação para variáveis categóricas	5
2.6	Especificando coleções de dados para o treinamento e para o teste .	8
2.7	Feature scaling - Normalização	8
2.8	Exercícios	11
3	Regressão Linear Simples	13
3.1	Definição geral	13
3.2	Expressão da Regressão Linear Simples	14
3.3	Importando os dados	14
3.4	Especificando as variáveis independente e dependente	14
3.5	Dados para treinamento e para teste	15
3.6	Executando o treinamento	15
3.7	Calculando os valores de classe para as instâncias de teste	15
3.8	Visualizando os dados de treinamento	16
3.9	Visualizando os dados de teste	16
3.10	Predição para uma única instância	16
3.11	Visualizando a equação da reta	17
3.12	Exercícios	17

Capítulo 1

Introdução

Conforme previu Moore em 1965, ao longo das décadas seguintes, a capacidade computacional dos computadores cresceu exponencialmente. Em contraste, o preço das memórias neste mesmo período cairia drasticamente. Tais acontecimentos tornaram viável o armazenamento e processamento de grandes volumes de dados, fenômeno esse muitas vezes denominado "Big Data". Além disso, estima-se que, ao longo da próxima década, a quantidade de dados disponíveis e acessíveis por meio da Internet irá dobrar a cada quinze minutos. Esses dados incluem interações de usuários em redes sociais, hábitos de motoristas coletados por seus carros inteligentes, hábitos de consumo de pessoas coletados por suas casas inteligentes, variações de temperatura e umidade no campo coletados por sensores etc. Diante da constatação de que tais dados possuem informações neles a princípio ocultas e que são de alto valor para a sociedade, fica evidente a necessidade de técnicas computacionais e matemáticas apropriadas para sua devida extração. Neste material abordam-se tópicos de estatística, probabilidade e algoritmos de mineração de dados com foco na extração de informações a partir de grandes volumes de dados brutos. Tais técnicas, quando aplicadas ao cenário descrito, constituem parte daquilo que hoje é conhecido como Ciência de Dados. Os algoritmos serão implementados utilizando-se a linguagem  python, uma das mais utilizadas para este fim atualmente.


Capítulo 2

Pré-processamento de dados

As técnicas que estudaremos consistem em analisar grandes coleções de dados a fim de encontrar possíveis padrões de interesse. Antes de implementá-las, entretanto, é comum a necessidade de alguns tipos de preparos prévios.

- O que fazer com dados faltantes?
- Como lidar com variáveis categóricas?
- Uma vez construído um modelo, qual coleção de dados utilizar para testá-lo?
- O que fazer quando o intervalo de uma variável é muito maior que o das demais?

Neste capítulo veremos algumas técnicas comumente utilizadas para resolver esses problemas.

2.1 Bibliotecas  possui muitas bibliotecas próprias para o processamento de dados.

- **numpy** - Disponibiliza objetos multidimensionais e uma coleção de funções eficientes para a sua manipulação, o que inclui ordenação, entrada e saída de dados, álgebra linear, operações estatísticas básicas etc.
- **matplotlib** - Viabiliza a visualização de dados por meio da criação de gráficos estáticos, animados e interativos.
- **pandas** - Fornece a implementação de estruturas de dados eficientes, em particular para dados relacionais e/ou rotulados.
- **scikit-learn** - Possui implementações de diversos algoritmos de aprendizado de máquina, além de ferramentas para pré-processamento de dados, validação etc.

O Bloco de Código 2.1.1 mostra como importá-las.

Bloco de Código 2.1.1

```
#utilizamos "as" para dar um "apelido"  
# às bibliotecas e simplificar seu uso  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

2.2 Importando os dados Faça o download do arquivo com os dados. Para importá-los, usaremos a biblioteca **pandas**, em particular a sua função **read_csv**. Veja o Bloco de Código 2.2.1.

Bloco de Código 2.2.1

```
#o prefixo r vem de raw  
#caracteres de escape são ignorados  
#pode ser útil caso queira usar \  
dataset = pd.read_csv(r'Data.csv')
```

2.3 Especificando as variáveis independentes e a variável dependente

Algoritmos de classificação têm como finalidade gerar um modelo que descreve como o valor de uma variável pode ser descrito em função dos valores das demais variáveis. Muitas vezes, essa variável é chamada de variável **dependente** ou de **classe**. As demais são chamadas de variáveis **independentes** ou **features**. Vamos utilizar a biblioteca **pandas** para, a partir da coleção de dados completa, especificar quais são as variáveis independentes e qual é a variável dependente. A Figura 2.3.1 mostra a coleção de dados com que estamos trabalhando no momento.

Figura 2.3.1

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Os dados incluem algumas características de consumidores de uma loja. A ideia é obter um modelo que prevê se um determinado consumidor irá comprar um produto ou não, dadas as demais características. O Bloco de Código 2.3.1 mostra como construir uma estrutura contendo as variáveis independentes e uma outra estrutura contendo a variável dependente.

Bloco de Código 2.3.1

```

#iloc: integer ou index location
#primeiro range: todas as linhas
#segundo range: todas as colunas, exceto a última
#iloc devolve instâncias associadas a índices
#values pega somente os valores
features = dataset.iloc[:, :-1].values
classe = dataset.iloc[:, -1].values

```

2.4 Dados faltantes Note que algumas instâncias (cada linha é uma instância) não possuem valores para algumas de suas variáveis. Isso pode ter impacto significativo no resultado final. Por isso, é preciso aplicar algum tipo de processamento para lidar com esses casos. Quando a coleção de dados é suficientemente grande e a quantidade de instâncias com valores faltantes é pequena, a sua simples **remoção** pode ser suficiente. Uma outra técnica simples consiste em **substituir o valor faltante pela média dos valores das demais instâncias**. O Bloco de Código

2.4.1 mostra como implementar essa última utilizando a classe `SimpleImputer` da biblioteca `scikit-learn`.

Bloco de Código 2.4.1

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer

. . .

imputer = SimpleImputer(missing_values=np.nan,
                        strategy="mean")
# todas as linhas
# somente as colunas numéricas
# manter as categóricas pode resultar em erros
# fit: sobre quais colunas operar?
imputer.fit(features[:, 1:3])
# transform: faz a operação e gera um objeto alterado
features[:, 1:3] = imputer.transform(features[:, 1:3])
```

2.5 Codificação para variáveis categóricas Algoritmos de aprendizado de máquina podem lidar com variáveis categóricas. No entanto, muitos deles requerem que elas sejam convertidas para valores numéricos, o que pode ser feito utilizando-se um **sistema de codificação**. Um sistema muito simples consiste em **substituir os valores textuais por números naturais em sequência**, como na Figura 2.5.1. Ele leva o nome de **Codificação por Rótulo**.

Figura 2.5.1

Country	Code
France	0
Spain	1
Germany	2

Embora possa funcionar em alguns casos, esse método pode dar origem a interpretações errôneas por parte de alguns algoritmos, devido à ordem natural dos números. No exemplo que estamos utilizando, podemos dizer que um país é, de alguma forma, “maior” do que o outro e, por isso, recebe um número maior? Isso depende da natureza dos dados e da análise que se deseja fazer. Outra técnica consiste em **converter a variável categórica para n variáveis**, sendo n igual ao número de valores existentes nela. Para cada instância, cada nova variável re-

cebe o valor 0 ou o valor 1. O valor 1 aparece somente na coluna referente ao valor existente naquela instância. Veja o exemplo da Figura 2.5.2.

Figura 2.5.2

Country	France	Spain	Germany
France	1	0	0
Spain	0	1	0
Germany	0	0	1
Spain	0	1	0
Germany	0	0	1
France	1	0	0
Spain	0	1	0
France	1	0	0
Germany	0	0	1
France	1	0	0

Com esse método nos livramos da ordenação imposta pelo método anterior. Entretanto, ele pode levar alguma desvantagem pelo fato de ter um número de colunas potencialmente muito maior, o que pode ter impacto no desempenho dos algoritmos. Ele leva nomes diferentes, com **One Hot Encoding** sendo bastante comum. O Bloco de Código 2.5.1 mostra a sua aplicação à variável *Country*.

Bloco de Código 2.5.1

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

. . .

# transformers: queremos fazer codificação
# do tipo One Hot Encoding
# na coluna de índice zero
# remainder: o que fazer com as demais colunas
# sem esse parâmetro elas seriam excluídas
# np.array: converte o resultado para um array numpy,
#           esperado pelos algoritmos que usaremos
columnTransformer = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(), [0])],
    remainder='passthrough')
features =
    np.array(columnTransformer.fit_transform(features))

```

A variável dependente - *Purchased* - também é categórica e também será convertida. Utilizaremos a Codificação por Rótulo. Veja o Bloco de Código 2.5.2.

Bloco de Código 2.5.2

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

. . .

labelEncoder = LabelEncoder()
classe = labelEncoder.fit_transform(classe)

```

2.6 Especificando coleções de dados para o treinamento e para o teste

Uma vez que um modelo tenha sido obtido, é muito importante testar o seu desempenho. Não estamos nos referindo ao seu tempo de execução ou consumo de memória, por exemplo. Estamos nos referindo, neste momento ainda informalmente, ao seu percentual de acerto quando seus resultados são comparados com uma coleção de dados composta por instâncias já classificadas. O teste, obviamente, não pode ser realizado utilizando-se a mesma base utilizada durante a fase de construção do modelo (o treinamento). Afinal, o modelo incorpora informações desta base e seu percentual de acerto seria provavelmente muito alto neste caso. Precisamos de uma outra coleção de dados para realizar os testes. Uma técnica bastante comum consiste em separar a coleção de dados em duas sub-coleções: uma que será utilizada somente na fase de treinamento e uma outra que será utilizada sobre na fase de testes. O Bloco de Código 2.6.1 mostra uma possível implementação.

Bloco de Código 2.6.1

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

. . .

#obtemos bases para treinamento e teste
#test_size: 20% das instâncias serão usadas para teste
#random_state: a cada execução as bases geradas são iguais
features_treinamento, features_teste, classe_treinamento,
    classe_teste = train_test_split(features, classe,
    test_size = 0.2, random_state=1)
```

2.7 Feature scaling - Normalização Cada variável independente tem um intervalo próprio. Devido à própria natureza dos dados, pode acontecer de uma delas ter um intervalo muito maior do que o das demais. Isso pode ser um problema para a construção de modelos de predição, pois variáveis com intervalo maior tendem a dominar as demais. Alguns algoritmos utilizam, por exemplo, medidas de distância entre instâncias para encontrar aquelas que são mais similares. Caso uma das variáveis independentes tenha intervalo muito maior do que o das demais, a distância entre duas instâncias será definida quase que exclusivamente em

função desta variável específica. Para resolver esse problema, aplicamos técnicas de **normalização**. A ideia consiste em fazer com que o intervalo de cada variável independente seja igual. Ele pode ser $[0, 1]$, $[-1, 1]$ etc. Tudo depende da natureza dos dados. Um dos métodos de **normalização** mais simples se chama **min-max**. Ele pode ser implementado como mostra a Equação 2.7.1, em que *norm* é o valor normalizado e *real* é o valor original.

$$norm = \frac{real - \min(real)}{\max(real) - \min(real)} \quad (2.7.1)$$

A Figura 2.7.1 mostra um exemplo em que as variáveis *Age* e *Salary* são normalizadas. As instâncias com valores faltantes para alguma dessas variáveis foram removidas.

Figura 2.7.1

Country	Age	Salary	Purchased	Age_Norm	Salary_Norm
France	44	72000	No	0.74	0.69
Spain	27	48000	Yes	0.00	0.00
Germany	30	54000	No	0.13	0.17
Spain	38	61000	No	0.48	0.37
France	35	58000	Yes	0.35	0.29
France	48	79000	Yes	0.91	0.89
Germany	50	83000	No	1.00	1.00
France	37	67000	Yes	0.43	0.54

Uma outra técnica de normalização bastante utilizada leva o nome de **standardização** ou **padronização**. Ela pode ser implementada como mostra a Equação 2.7.2, em que *norm* é o valor normalizado, *real* é o valor original, μ é a média da variável e σ é o seu desvio-padrão. O valor obtido indica a quantidade de desvios-padrão o valor original está acima (quando positivo) ou abaixo (quando negativo) da média.

$$norm = \frac{real - \mu}{\sigma} \quad (2.7.2)$$

A Figura 2.7.2 mostra um exemplo em que a padronização foi aplicada às variáveis *Age* e *Salary*.

Figura 2.7.2

Country	Age	Salary	Purchased	Age_Std	Salary_Std
France	44	72000	No	0.699858	0.589891
Spain	27	48000	Yes	-1.51365	-1.5075
Germany	30	54000	No	-1.12303	-0.98315
Spain	38	61000	No	-0.08138	-0.37141
France	35	58000	Yes	-0.472	-0.63359
France	48	79000	Yes	1.220683	1.20163
Germany	50	83000	No	1.481095	1.551195
France	37	67000	Yes	-0.21158	0.152935

Lembre-se que o objetivo da padronização é ajustar o intervalo de variáveis evitando o domínio de uma sobre as demais. As variáveis binárias que adicionamos para codificar as variáveis categóricas não devem, portanto, passar por esse processo. O Bloco de Código 2.7.1 mostra o uso da classe `StandardScaler` da biblioteca `scikit-learn`.

Bloco de Código 2.7.1

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

. . .

standardScaler = StandardScaler()
# padronização na coleção usada para o treinamento
# fit: calcula a média e o desvio padrão de cada variável
# transform: faz o cálculo

features_treinamento[:, 3:] = standardScaler.fit_transform(
    features_treinamento[:, 3:])
#fazemos somente transform para a coleção de teste
#assim a média e desvio-padrão da população completa são
usados
#Em produção, os dados de teste serão novos dados
#Suas variáveis devem ser padronizadas usando as mesmas
#medidas da população
features_teste[:, 3:] = standardScaler.transform(
    features_teste[:, 3:])
```

2.8 Exercícios Faça o download do arquivo de dados próprio para os exercícios. A ideia é ajustar os dados para que, no futuro, seja possível elaborar um modelo que decide se haverá jogo ou não em função das variáveis independentes. Escreva

um programa com  python que

- Faz a leitura do arquivo *CSV* usando *pandas*.
- Cria uma base contendo as variáveis independentes e uma base contendo a variável dependente.
- Substitui dados faltantes pela média da respectiva variável.
- Codifica todas as variáveis categóricas independentes com **One Hot Encoding**.
- Codifica a variável dependente com **Codificação por Rótulo**.
- Separa a base em duas partes: uma para treinamento e outra para testes. Use 85% das instâncias para o treinamento.
- Normaliza as variáveis *temperatura* e *humidade* usando padronização.

Capítulo 3

Regressão Linear Simples

Esta técnica é aplicada quando desejamos explicar a variação de uma variável dependente em função de uma única variável independente. Pressupõem-se que há uma relação linear (a qual pode ser verificada por diferentes métodos, os quais não são parte do escopo deste capítulo) entre as variáveis, daí o nome **linear**. O modelo é dito **simples** pelo fato de basear-se somente em uma única variável independente.

3.1 Definição geral A expressão geral de uma modelo de regressão linear é exibida pela Equação 3.1.1. , em que x é a variável dependente, a_1, a_2, \dots, a_k são as variáveis independentes e w_0, w_1, \dots, w_k são os coeficientes que desejamos encontrar, calculados na fase de treinamento. Note que $k = 1$ quando o modelo é “simples”. O peso w_0 está sendo utilizado para simplificar a notação. Considere que ele está associado a um atributo a_0 cujo valor é sempre 1.

$$x = w_0 + w_1 a_1 + w_2 a_2 + \dots + w_k a_k \quad (3.1.1)$$

Suponha que $a_1^{(i)}, a_2^{(i)}, \dots, a_k^{(i)}$ são os valores das variáveis independentes da i -ésima instância da coleção de dados e que a sua classe ou variável dependente é $x^{(i)}$. O valor de $x^{(i)}$ obtido pelo método de regressão linear é exibido na Equação 3.1.2.

$$w_0 a_0^{(i)} + w_1 a_1^{(i)} + \dots + w_k a_k^{(i)} = \sum_{j=0}^k w_j a_j^{(i)} \quad (3.1.2)$$

A intenção é calcular coeficientes de modo que a diferença entre o valor obtido pelo modelo e o valor real existente nos dados de treinamento sejam minimizado. A Equação 3.1.3 mostra o valor a ser minimizado, considerando que n é o número de instâncias na coleção de dados.

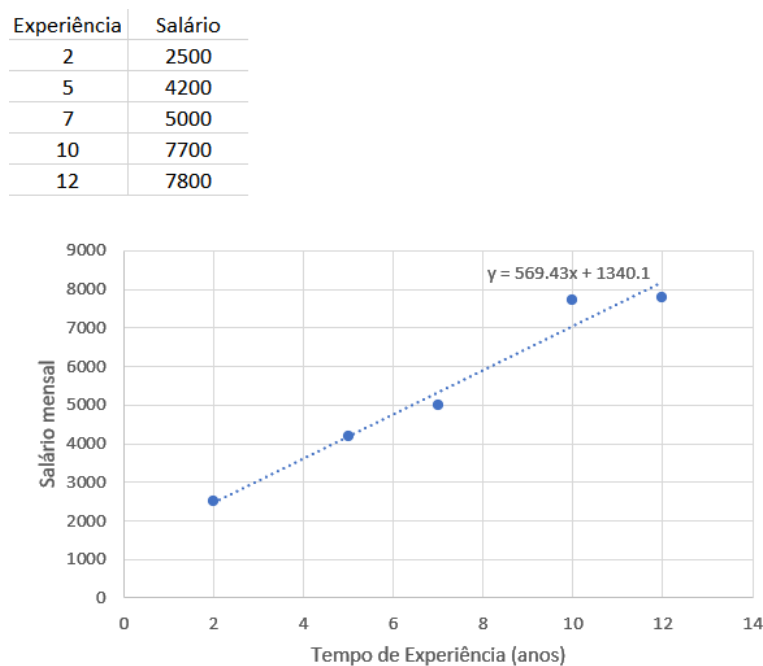
$$\sum_{i=1}^n \left(x^{(i)} - \sum_{j=0}^k w_j a_j^{(i)} \right)^2 \quad (3.1.3)$$

3.2 Expressão da Regressão Linear Simples Utilizaremos a expressão exibida pela Equação 3.2.1, em que y é a variável dependente, b_0 é o coeficiente linear a ser encontrado e b_1 é o coeficiente angular a ser encontrado.

$$y = b_0 + b_1x_1 \quad (3.2.1)$$

Graficamente, a Equação 3.2.1 dá origem a uma reta. Como mostra a Figura 3.2.1, o objetivo é traçar-la minimizando a soma dos quadrados das diferenças entre os valores preditos e os valores reais para cada instância.

Figura 3.2.1



3.3 Importando os dados Faça o download do arquivo de dados do ambiente da disciplina. A seguir, importe as bibliotecas necessárias, como mostra o Bloco de Código 3.3.1.

Bloco de Código 3.3.1

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

3.4 Especificando as variáveis independente e dependente Nossa coleção de dados possui somente duas variáveis. O Bloco de código 3.4.1 mostra como importar os dados e separar cada variável em uma estrutura de dados independente.

Bloco de Código 3.4.1

```
dataset = pd.read_csv ('dados.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

3.5 Dados para treinamento e para teste Uma vez obtido um modelo, desejamos verificar o seu desempenho quando aplicado a uma base de dados diferente daquela utilizada no treinamento. Por isso, vamos separar a base de dados em duas: uma será utilizada para o treinamento e a outra para os testes. O Bloco de Código 3.5.1 mostra como fazê-lo.

Bloco de Código 3.5.1

```
. . .
from sklearn.model_selection import train_test_split
. . .
x_treinamento, x_teste, y_treinamento, y_teste =
    train_test_split(x, y, test_size=0.2,
                    random_state=0)
```

3.6 Executando o treinamento A biblioteca `scikit-learn` possui uma classe que nos permite obter modelos de regressão linear simples. Ela se chama `LinearRegression`. Uma vez importada, construímos um objeto e utilizamos o método `fit` para realizar o treinamento, ou seja, a obtenção do vetor de coeficientes (neste caso temos apenas um). Veja o Bloco de Código 3.6.1.

Bloco de Código 3.6.1

```
. . .
from sklearn.linear_model import LinearRegression
. . .
linearRegression = LinearRegression()
linearRegression.fit(x_treinamento, y_treinamento)
```

3.7 Calculando os valores de classe para as instâncias de teste Uma vez obtido o modelo, desejamos verificar os valores de classe que ele irá calcular para cada instância de teste. Isso pode ser feito utilizando-se o método `predict`. Veja o Bloco de Código 3.7.1.

Bloco de Código 3.7.1

```
. . .
y_pred = linearRegression.fit(x_treinamento, y_treinamento)
```

3.8 Visualizando os dados de treinamento A biblioteca `matplotlib` viabiliza a visualização de dados. Utilizaremos métodos de `pyplot` para visualizar os dados de treinamento e a reta obtida em função dos coeficientes calculados. Os dados podem ser visualizados utilizando-se um gráfico de dispersão, que pode ser obtido com o método `scatter`. A reta será construída com o método `plot`. Veja o Bloco de Código 3.8.1.

Bloco de Código 3.8.1

```
. . .
plt.scatter(x_treinamento, y_treinamento, color="red")
plt.plot(x_treinamento,
         linearRegression.predict(x_treinamento),
         color="blue")
plt.title("Salário x Tempo de Experiência (Treinamento)")
plt.xlabel("Anos de Experiência")
plt.ylabel("Salário")
plt.show()
```

3.9 Visualizando os dados de teste A visualização dos dados de teste é análoga à visualização dos dados de treinamento. Basta trocar os dados. Veja o Bloco de Código .

Bloco de Código 3.9.1

```
. . .
plt.scatter(x_teste, y_teste, color="red")
#os coeficientes são únicos, assim não faz diferença
#trocar as coleções na hora de exibir
plt.plot(x_treinamento,
         linearRegression.predict(x_treinamento),
         color="blue")
plt.title("Salário x Tempo de Experiência (Teste)")
plt.xlabel("Anos de Experiência")
plt.ylabel("Salário")
plt.show()
```

3.10 Predição para uma única instância Podemos realizar a predição para uma única instância usando o método `predict`. Note que ele espera um vetor de vetores, já que pode operar sobre uma coleção de instâncias, cada qual com uma coleção de atributos. Veja o Bloco de Código 3.10.1.

Bloco de Código 3.10.1

```

. . .
# qual o salario de alguem com 15.7 anos de experiencia
print(linearRegression.predict([[15.7]]))
#quanto deve ganhar alguem que acabou de entrar na empresa
print(linearRegression.predict([[0]]))

```

3.11 Visualizando a equação da reta Podemos obter os coeficientes angular e linear obtidos utilizando os métodos `coef_` e `intercept_`, respectivamente. Veja o Bloco de Código 3.11.1.

Bloco de Código 3.11.1

```


. . .
print(
f'y = {linearRegression.coef_[0]:.2f}x +
      {linearRegression.intercept_:.2f}')

```

3.12 Exercícios Vasilhe o portal de dados brasileiro (Link ??) e encontre uma base de dados interessante, própria para a obtenção de um modelo de regressão linear simples. Caso a base tenha mais de duas variáveis independentes, escolha uma delas e remova as demais.

Link 3.12.1

<https://dados.gov.br/>

Escreva um programa com  python que

- Faz a leitura do arquivo *CSV* usando *pandas*.
- Cria uma base contendo as variáveis independentes e uma base contendo a variável dependente.
- Separa a base em duas partes: uma para treinamento e outra para testes. Use 85% das instâncias para o treinamento.
- Constrói um modelo de regressão linear simples em função dos dados de treinamento.
- Exibe a reta obtida e os dados de treinamento em um mesmo gráfico. Inclua a equação obtida no título do gráfico.
- Exibe a reta obtida e os dados de teste em um mesmo gráfico. Inclua a equação obtida no título do gráfico.

