

---

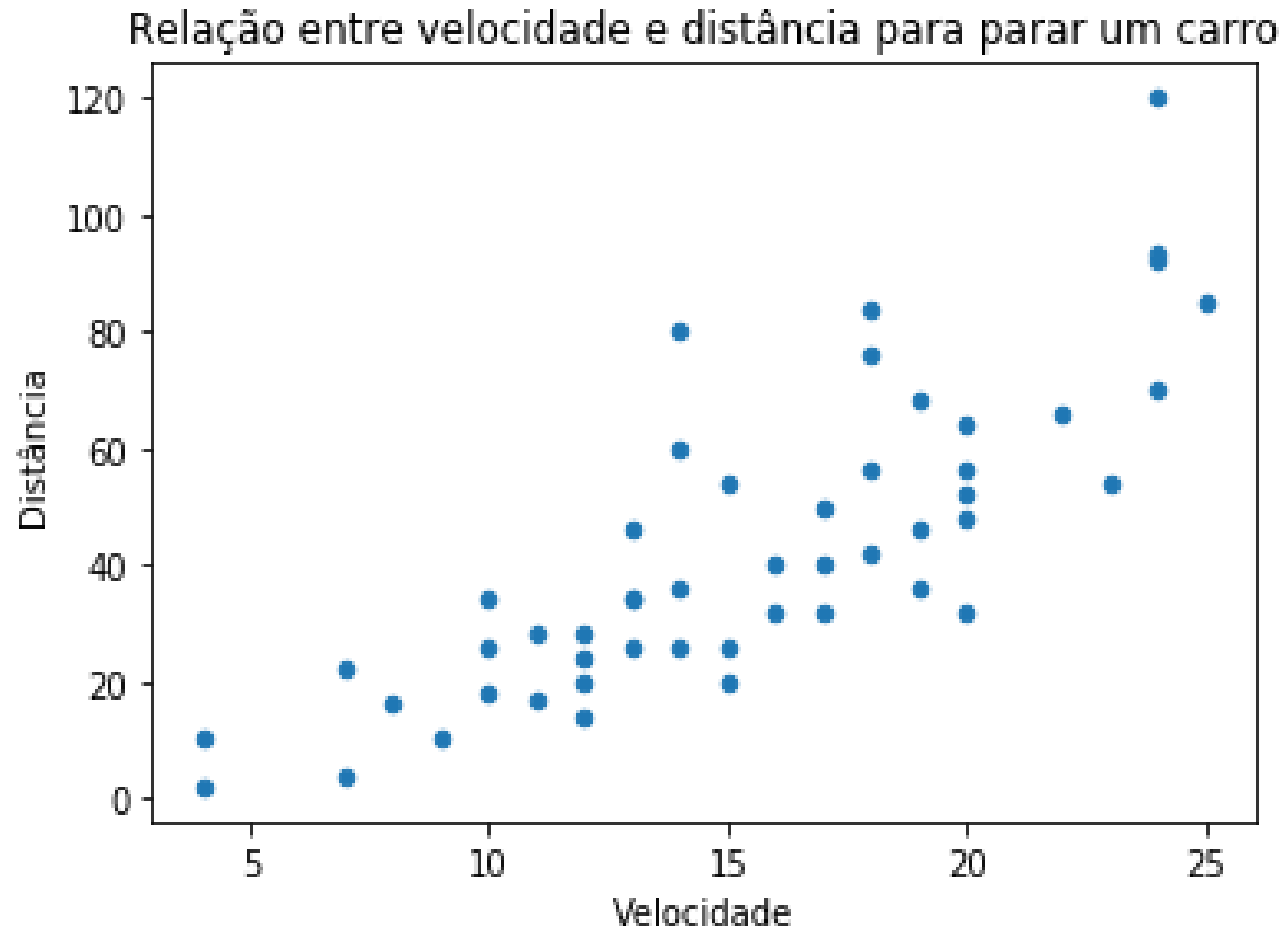
# Regressão Linear

Fabrício Barth

Setembro de 2019

---

# Dados sobre carros



---

## Código para plotar o exemplo anterior

```
import matplotlib.pyplot as plt
cars.plot(kind='scatter', x='speed', y='dist',
          style='o')
plt.title('Relação entre velocidade e distância')
plt.xlabel('Velocidade')
plt.ylabel('Distância')
plt.show()
```

a

---

<sup>a</sup>O DataFrame cars já foi carregado anteriormente.

---

# Relações entre variáveis

- Será que existe relação entre a distância com que um carro consegue parar e a velocidade com que ele estava no momento da freada?
- Métodos de regressão tentam identificar se existe uma relação entre a variável dependente (o valor que precisa ser predito) e a variável independente.
- Distância = variável dependente
- Velocidade = variável independente

---

## Definindo linhas

- Uma linha pode ser definida na forma de
$$y = \alpha + \beta \times x$$
- onde  $x$  é a variável independente e  $y$  a variável dependente.
- $\beta$  indica quanto a linha cresce a cada incremento de  $x$ .
- A variável  $\alpha$  indica o valor de  $y$  quando  $x = 0$ .

---

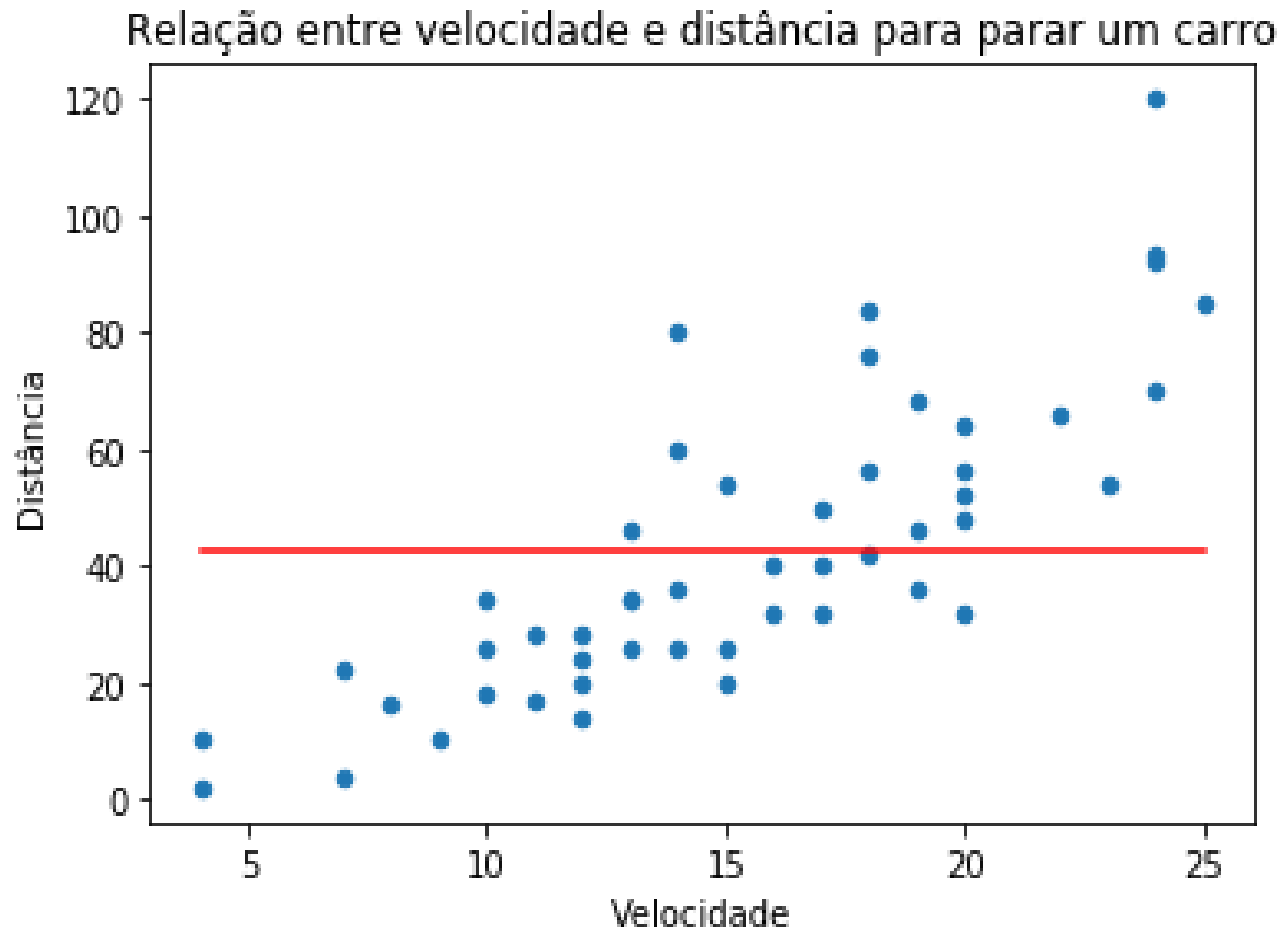
## Definindo modelos de regressão linear

- O objetivo de um algoritmo que cria este tipo de função é definir valores para  $\alpha$  e  $\beta$  de tal maneira que a linha consiga representar o conjunto de dados.
- Esta linha pode não representar o conjunto de dados perfeitamente. Portanto, é necessário calcular o erro de alguma forma.

---

$$distancia = 42.3 + 0 \times velocidade$$

É uma função válida. Mas é uma função boa?



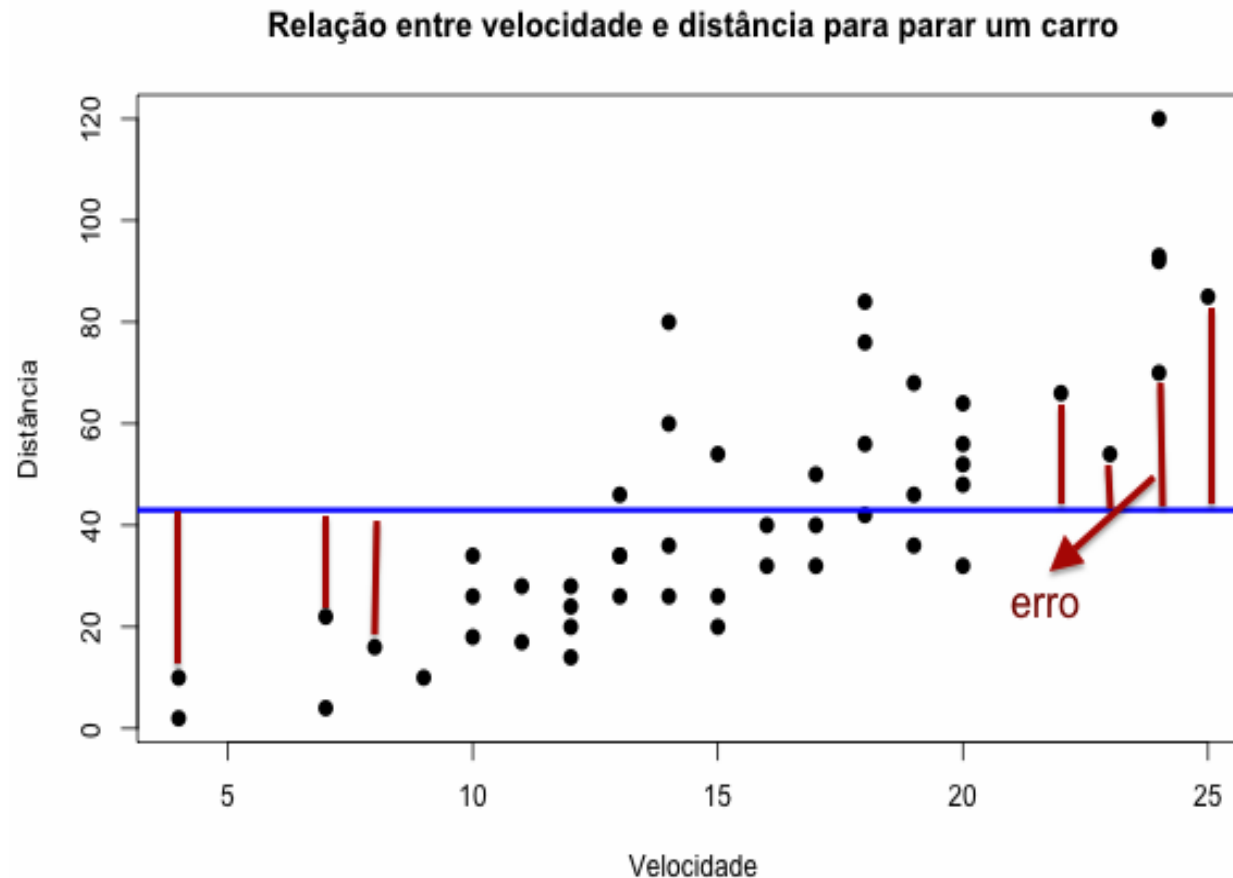
---

$distancia = 42.3 + 0 \times velocidade$  em Python

```
y_predicted = cars['speed'].apply(  
    lambda x : 42.3 + 0 * x)  
  
cars.plot(kind='scatter', x='speed', y='dist',  
          style='o')  
plt.title('Relação entre velocidade e distância')  
plt.xlabel('Velocidade')  
plt.ylabel('Distância')  
plt.plot(cars['speed'], y_predicted, color='r')  
plt.show()
```



# Erro de

$$distancia = 42.3 + 0 \times velocidade$$


---

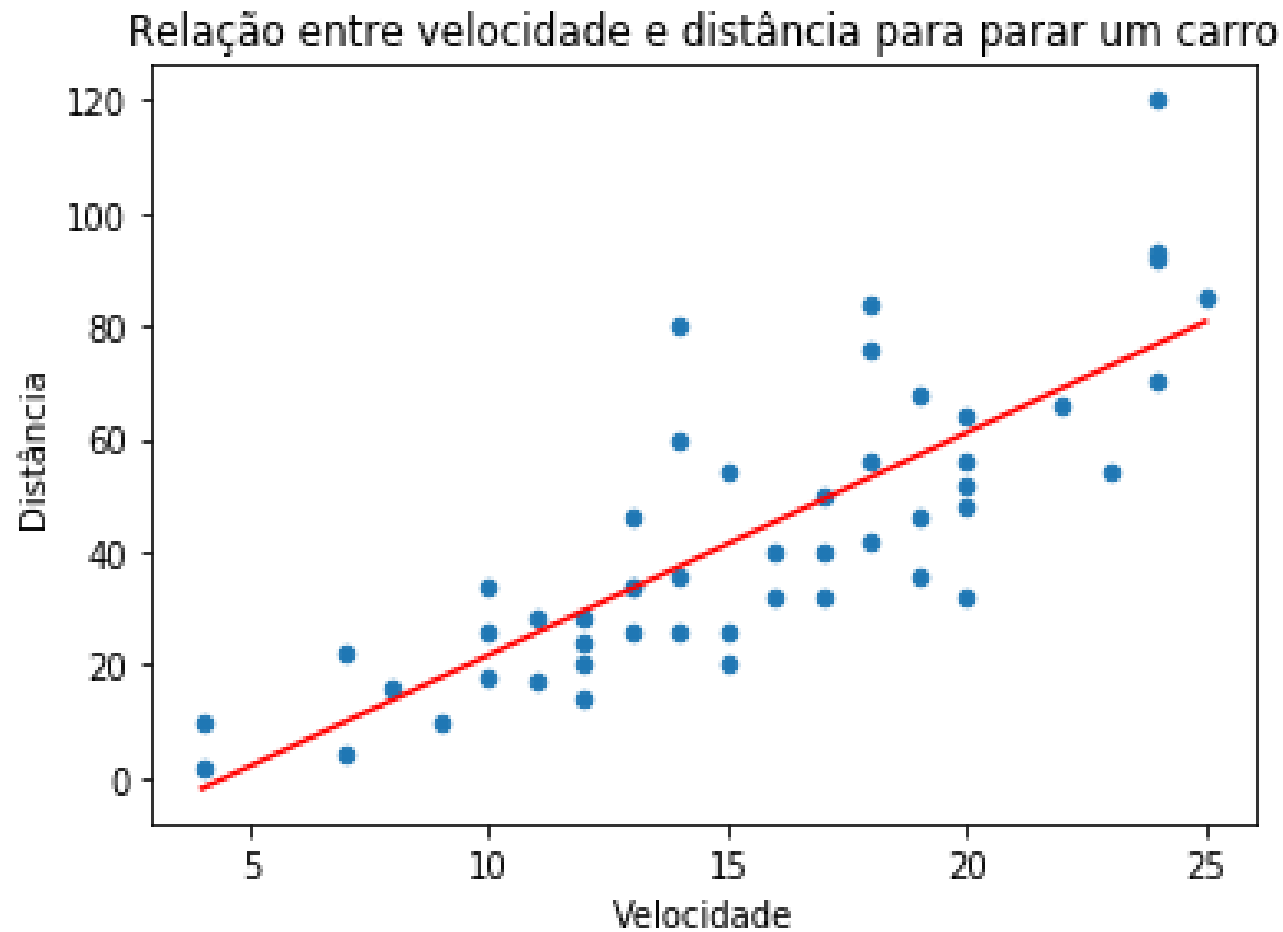
## Determinando o valor de $\alpha$ e $\beta$ em uma regressão linear simples

- Para estimar os melhores valores para  $\alpha$  e  $\beta$  é utilizado método chamado de **ordinary least squares (OLS)**.
- Com este método, os valores de  $\alpha$  e  $\beta$  são escolhidos para minimizar a soma dos erros ao quadrado, ou seja, a distância vertical entre o valor predito e o valor real.

$$erro = \sum (y_i - \hat{y}_i)^2 \quad (1)$$

onde,  $y_i$  é o valor real e  $\hat{y}_i$  é o valor predito.

# Uma função com um erro menor



---

## Código em *Python* para o slide anterior

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(
    cars['speed'].values.reshape(-1,1),
    cars['dist'].values.reshape(-1,1))
y_predicted = model.predict(
    cars['speed'].values.reshape(-1,1))
```

---

```
cars.plot(kind='scatter', x='speed', y='dist',
          style='o')
plt.title('Relação entre velocidade e distância')
plt.xlabel('Velocidade')
plt.ylabel('Distância')
plt.plot(cars['speed'], y_predicted, color='r')
plt.show()
```

---

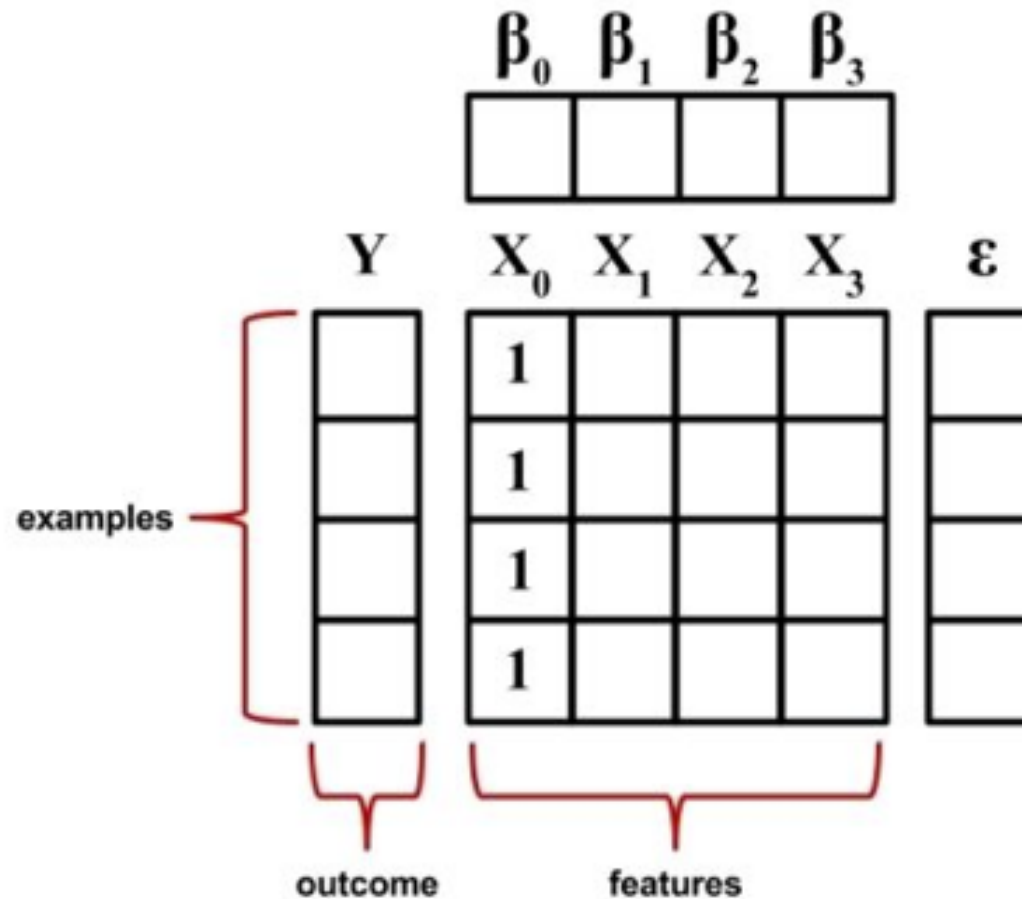
## Regressão linear múltipla

$$y = \alpha + \beta_1 \times x_1 + \beta_2 \times x_2 + \cdots + \beta_i \times x_i + e \quad (2)$$

Podemos utilizar uma equação compactada:

$$Y = X \times \beta + e \quad (3)$$

# Regressão linear múltipla



---

## Regressão linear múltipla

Agora o objetivo é resolver  $\hat{\beta}$ :

$$\beta = (X^T X)^{-1} X^T Y \quad (4)$$

onde:

- $X^T$  é matriz transposta de  $X$ , e;
- $X^{-1}$  a matriz inversa de  $X$ .



---

## Encontrando os coeficientes para o problema do carro

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(
    cars['speed'].values.reshape(-1,1),
    cars['dist'].values.reshape(-1,1))
```

Este dataset tem apenas uma variável independente. Por isso é necessário fazer o reshape da mesma.

---

## Exemplo de regressão linear simples usando *LinearRegression*

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(
    cars['speed'].values.reshape(-1,1),
    cars['dist'].values.reshape(-1,1))

print(model.intercept_)
print(model.coef_)
```

---

# Avaliação de modelos

```
from sklearn.metrics import mean_squared_error,
    r2_score
rmse = mean_squared_error(cars['dist'],
    y_predicted)
r2 = r2_score(cars['dist'], y_predicted)

print(rmse)
print(r2)
```

Da forma como é calculado é bom para avaliar o comportamento dos seus dados no passado, mas não é adequado para avaliar a sua capacidade de generalização.

---

## Exemplo de regressão linear múltipla

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
    r2_score, mean_absolute_error
from sklearn.datasets import load_boston
boston = load_boston()
model = LinearRegression().fit(
    boston['data'],
    boston['target'])
```

---

## Exemplo de regressão linear múltipla

```
print(model.intercept_)
```

```
print(model.coef_)
```

```
36.45948838508987
```

```
[-1.08011358e-01  4.64204584e-02  2.05586264e-02  
 2.68673382e+00 -1.77666112e+01  3.80986521e+00  
 6.92224640e-04 -1.47556685e+00  3.06049479e-01  
-1.23345939e-02 -9.52747232e-01  9.31168327e-03  
-5.24758378e-01]
```

---

# Exemplo de regressão linear múltipla

```
y_predicted = model.predict(boston['data'])

rmse = mean_squared_error(boston['target'],
                           y_predicted)

r2 = r2_score(boston['target'], y_predicted)

print(rmse)
print(r2)
print('Mean Absolute Error:',
      mean_absolute_error(
          boston['target'], y_predicted))
```

---

## Exemplo de regressão linear múltipla

21.894831181729206

0.7406426641094094

Mean Absolute Error: 3.270862810900314

---

# Exemplos didático para regressão linear e polinomial



---

# Criação dados

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
print(x)
```

```
[[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]]
```

```
print(y)
```

```
[ 5 20 14 32 22 38]
```

---

# Criação do modelo e uso

```
model = LinearRegression()
model.fit(x, y)
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('coefficients:', model.coef_)
```

```
coefficient of determination: 0.715875613747954
intercept: 5.633333333333329
coefficients: [0.54]
```

---

```
y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')
```

```
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

---

## Outra forma de utilização do modelo

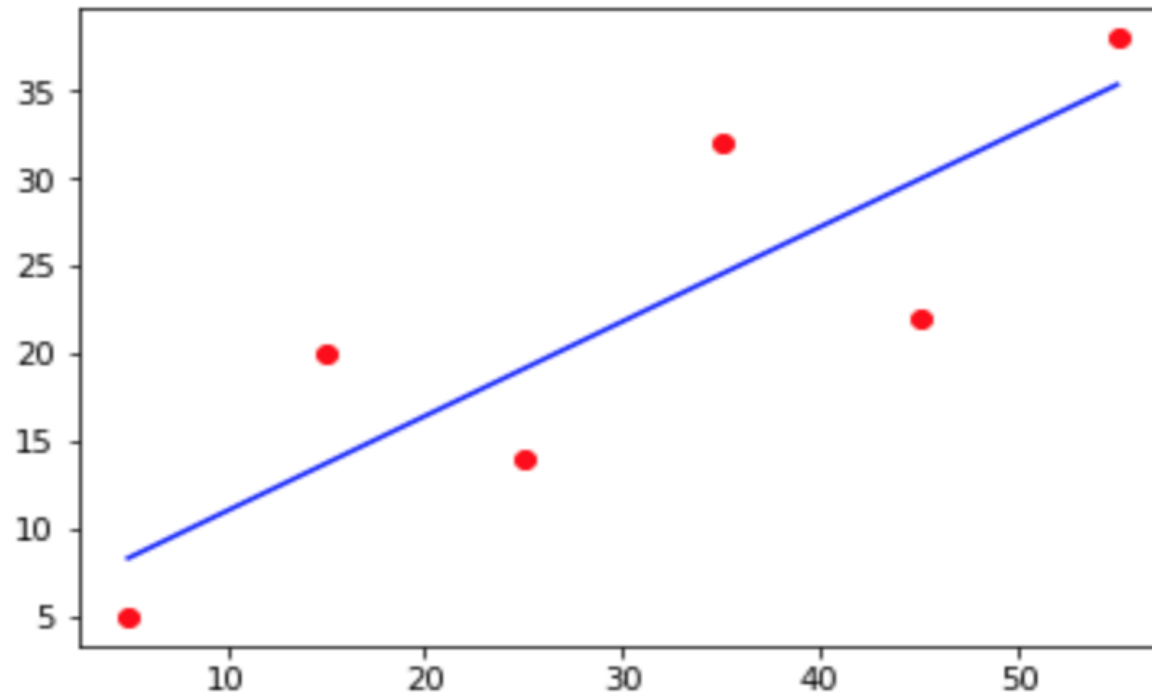
```
y_pred = model.intercept_ + model.coef_ * x
print('predicted response:', y_pred, sep='\n')
```

```
predicted response:
```

```
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
```

# Plot do modelo

```
plt.plot(x, y, 'ro')  
plt.plot(x, y_pred, color='b')  
plt.show()
```



---

# Regressão polinomial

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([15, 11, 2, 8, 25, 32])
```

```
transformer = PolynomialFeatures(degree=2, include_bias=False)
transformer.fit(x)
x_ = transformer.transform(x)
```

---

## Atributos gerados utilizado grau 2

```
print(x_)
```

```
[[ 5.  25.]  
 [15. 225.]  
 [25. 625.]  
 [35. 1225.]  
 [45. 2025.]  
 [55. 3025.]]
```

---

# Utilização do modelo

```
model = LinearRegression().fit(x_, y)
```

```
r_sq = model.score(x_, y)
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('coefficients:', model.coef_)
```

```
coefficient of determination: 0.8908516262498564
intercept: 21.372321428571425
coefficients: [-1.32357143  0.02839286]
```

---

# Utilização do modelo

```
y_pred = model.predict(x_)  
print('predicted response:', y_pred, sep='\n')
```

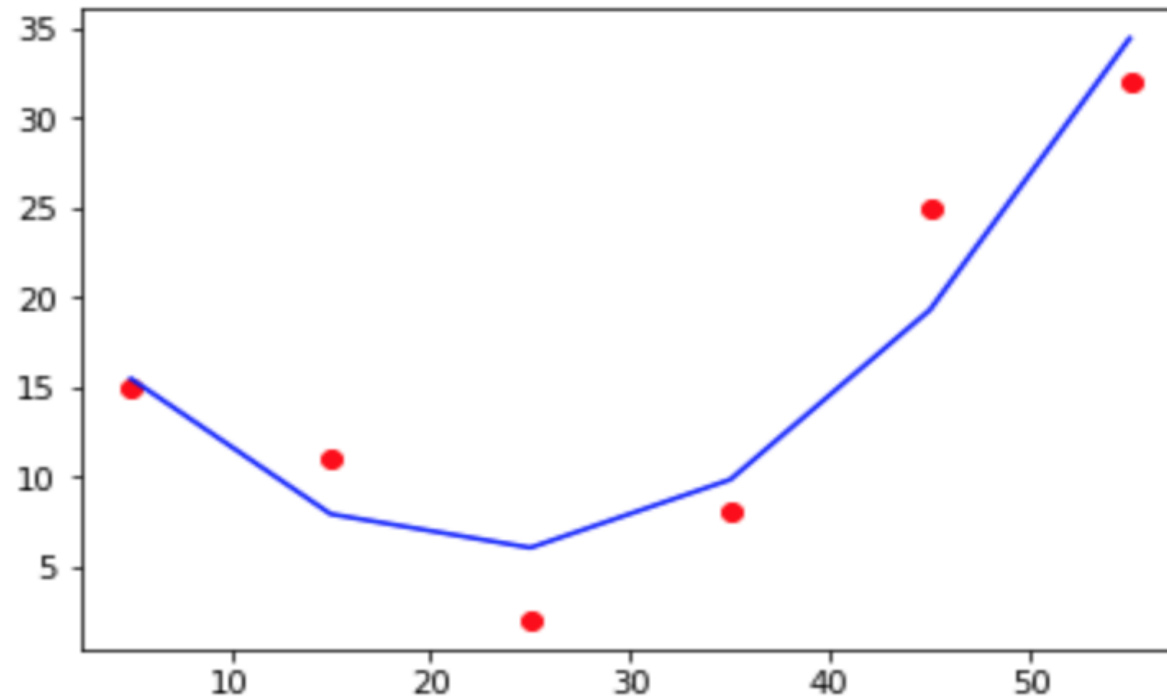
predicted response:

[15.46428571 7.90714286 6.02857143 9.82857143 19.30714286 34.46428571]



# Plot do modelo polinomial

```
plt.plot(x, y, 'ro')  
plt.plot(x, y_pred, color='b')  
plt.show()
```



---

# Modelo grau 4

```
transformer = PolynomialFeatures(degree=4, include_bias=False)
transformer.fit(x)
x_ = transformer.transform(x)
print(x_)
model = LinearRegression().fit(x_, y)
r_sq = model.score(x_, y)
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('coefficients:', model.coef_)
y_pred = model.predict(x_)
print('predicted response:', y_pred, sep='\n')
plt.plot(x, y, 'ro')
plt.plot(x, y_pred, color='b')
plt.show()
```

---

## Modelo grau 4: resultados

```
[[5.000000e+00 2.500000e+01 1.250000e+02 6.250000e+02]
 [1.500000e+01 2.250000e+02 3.375000e+03 5.062500e+04]
 [2.500000e+01 6.250000e+02 1.562500e+04 3.906250e+05]
 [3.500000e+01 1.225000e+03 4.287500e+04 1.500625e+06]
 [4.500000e+01 2.025000e+03 9.112500e+04 4.100625e+06]
 [5.500000e+01 3.025000e+03 1.663750e+05 9.150625e+06]]
```

coefficient of determination: 0.9996871368552787

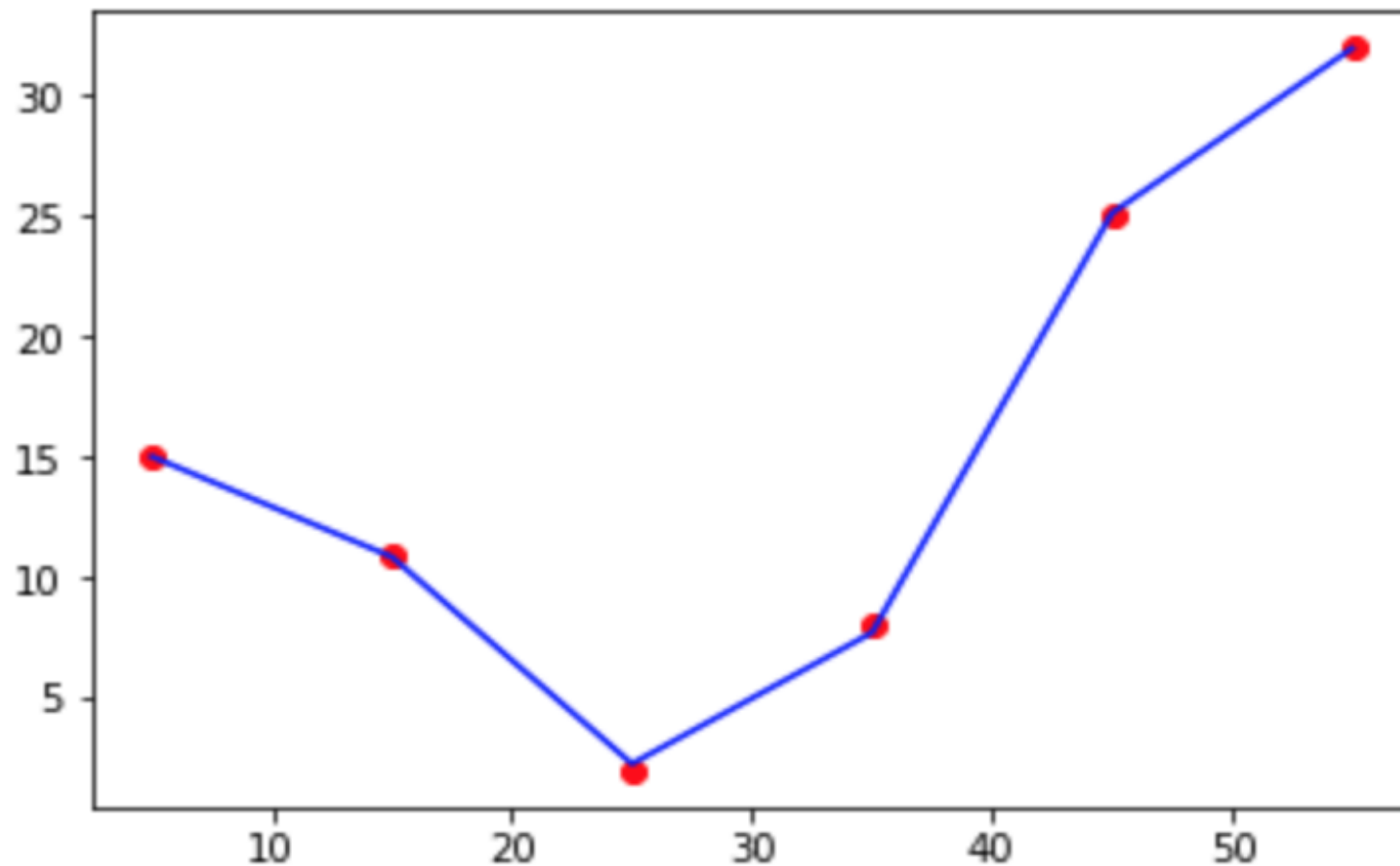
intercept: 4.085503472844266

coefficients: [ 3.67175926e+00 -3.44062500e-01 9.90740741e-03 -8.54166667e-05]

predicted response:

```
[15.02777778 10.86111111  2.27777778  7.72222222 25.13888889 31.97222222]
```

## Modelo grau 4: plot



---

## Material de **consulta**

- [https://github.com/fbarth/ml-espmm/blob/master/scripts/02\\_01\\_regressao\\_linear.ipynb](https://github.com/fbarth/ml-espmm/blob/master/scripts/02_01_regressao_linear.ipynb)
- <https://realpython.com/linear-regression-in-python/>