Machine Learning

Mean, Median e Mode - Média, mediana e modo

O que podemos aprender observando um grupo de números?

No Machine Learning (e na matemática) existem frequentemente três valores que nos interessam:

- **Média** - O valor médio
- **Mediana** - O valor do ponto médio
- **Modo** - O valor mais comum

Exemplo: Registramos a velocidade de 13 carros:

```
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Qual é o valor de velocidade médio, intermediário ou mais comum?

Mean

Para calcular a média, encontre a soma de todos os valores e divida a soma pelo número de valores:

```
(99+86+87+88+111+86+103+87+94+78+77+85+86) / 13 = 89.77
```

O módulo NumPy possui um método para isso.

Exemplo

Use o `mean()` do método NumPy para encontrar a velocidade média:

```
import numpy
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = numpy.mean(speed)
print(x)
```

Mediana

O valor mediano é o valor intermediário, depois de classificar todos os valores:

77, 78, 85, 86, 86, 86, 87, 87, 88, 94, 99, 103, 111

É importante que os números sejam classificados antes de você encontrar a mediana.

Use o método NumPy `median()` para encontrar o valor médio:

```
import numpy
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = numpy.median(speed)
print(x)
```

Se houver dois números no meio, divida a soma desses números por dois.

77, 78, 85, 86, 86, 86, 87, 87, 94, 98, 99, 103

(86 + 87) / 2 = 86.5

```
import numpy
speed = [99,86,87,88,86,103,87,94,78,77,85,86]
x = numpy.median(speed)
print(x)
```

Mode

O valor Mode é o valor que aparece o maior número de vezes:

99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86 = 86

Use o método SciPy `mode()` para encontrar o número que aparece mais:

```
from scipy import stats
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = stats.mode(speed)
print(x)
```

Desvio Padrão

O desvio padrão é um número que descreve a distribuição dos valores.

Um desvio padrão baixo significa que a maioria dos números está próxima do valor médio (médio).

Um desvio padrão alto significa que os valores estão espalhados por uma faixa mais ampla.

Exemplo: Desta vez registramos a velocidade de 7 carros:

```
speed = [86,87,88,86,87,85,86]
```

O desvio padrão é:

```
0.9
```

O que significa que a maioria dos valores está na faixa de 0,9 do valor médio, que é 86,4.

Façamos o mesmo com uma seleção de números com um intervalo mais amplo:

```
speed = [32,111,138,28,59,77,97]
```

O desvio padrão é:

```
37.85
```

O que significa que a maioria dos valores está na faixa de 37,85 do valor médio, que é 77,4.

Como você pode ver, um desvio padrão mais alto indica que os valores estão espalhados por uma faixa mais ampla.

O módulo NumPy possui um método para calcular o desvio padrão:

Use o `std()`método NumPy para encontrar o desvio padrão:

```
import numpy
speed = [86,87,88,86,87,85,86]
x = numpy.std(speed)
print(x)
```

```
import numpy
speed = [32,111,138,28,59,77,97]
x = numpy.std(speed)
print(x)
```

Variância (var)

A variância é outro número que indica o quão dispersos estão os valores.

Na verdade, se você tirar a raiz quadrada da variância, obterá o desvio padrão!

Ou vice-versa, se você multiplicar o desvio padrão por ele mesmo, obtém a variância!

Para calcular a variância você deve fazer o seguinte:

1. Encontre a média:

```
(32+111+138+28+59+77+97) / 7 = 77.4
```

2. Para cada valor: encontre a diferença da média:

```
 32 - 77.4 = -45.4
111 - 77.4 =  33.6
138 - 77.4 =  60.6
 28 - 77.4 = -49.4
 59 - 77.4 = -18.4
 77 - 77.4 = - 0.4
 97 - 77.4 =  19.6
```

3. Para cada diferença: encontre o valor quadrado:

```
(-45.4)² = 2061.16
 (33.6)² = 1128.96
 (60.6)² = 3672.36
(-49.4)² = 2440.36
(-18.4)² =  338.56
(- 0.4)² =    0.16
 (19.6)² =  384.16
```

4. A variância é o número médio destas diferenças quadradas:

```
(2061.16+1128.96+3672.36+2440.36+338.56+0.16+384.16) / 7 = 1432.2
```

Felizmente, NumPy tem um método para calcular a variação:

Use o `var()`método NumPy para encontrar a variação:

```
import numpy
speed = [32,111,138,28,59,77,97]
x = numpy.var(speed)
print(x)
```

Desvio padrão

Como aprendemos, a fórmula para encontrar o desvio padrão é a raiz quadrada da variância:

√1432.25 = 37.85

Ou, como no exemplo anterior, use o NumPy para calcular o desvio padrão:

Use o `std()` método NumPy para encontrar o desvio padrão:

```
import numpy
speed = [32,111,138,28,59,77,97]
x = numpy.std(speed)
print(x)
```

Símbolos

O Desvio Padrão é frequentemente representado pelo símbolo Sigma: σ

A variância é frequentemente representada pelo símbolo Sigma Squared: σ^2

Percentiles

Os percentis são usados nas estatísticas para fornecer um número que descreve o valor inferior a uma determinada porcentagem dos valores.

Exemplo: Digamos que temos um conjunto de idades de todas as pessoas que moram na rua.

```
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
```

Qual é o percentil 75? A resposta é 43, o que significa que 75% das pessoas têm 43 anos ou menos.

O módulo NumPy possui um método para encontrar o percentil especificado:

Use o método NumPy `percentile()`para encontrar os percentis:

```
import numpy
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
x = numpy.percentile(ages, 75)
print(x)
```

Qual é a idade em que 90% das pessoas são mais jovens?

```
import numpy
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
x = numpy.percentile(ages, 90)
print(x)
```

Distribuição de Dados

No mundo real, os conjuntos de dados são muito maiores, mas pode ser difícil recolher dados do mundo real, pelo menos numa fase inicial de um projeto.

Como podemos obter conjuntos de Big Data?

Para criar conjuntos de big data para teste, usamos o módulo Python NumPy, que vem com vários métodos para criar conjuntos de dados aleatórios, de qualquer tamanho.

Crie um array contendo 250 números flutuantes aleatórios entre 0 e 5:

```
import numpy
x = numpy.random.uniform(0.0, 5.0, 250)
print(x)
```

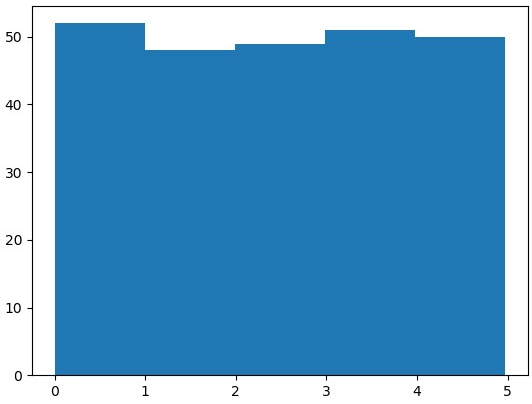
Histograma

Para visualizar o conjunto de dados podemos desenhar um histograma com os dados que coletamos.

Usaremos o módulo Python Matplotlib para desenhar um histograma.

Desenhe um histograma:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 250)

plt.hist(x, 5)
plt.show()
```



Histograma explicado

Usamos o array do exemplo acima para desenhar um histograma com 5 barras.

A primeira barra representa quantos valores no array estão entre 0 e 1.

A segunda barra representa quantos valores estão entre 1 e 2.

Etc.

O que nos dá este resultado:

- 52 valores estão entre 0 e 1
- 48 valores estão entre 1 e 2
- 49 valores estão entre 2 e 3
- 51 valores estão entre 3 e 4
- 50 valores estão entre 4 e 5

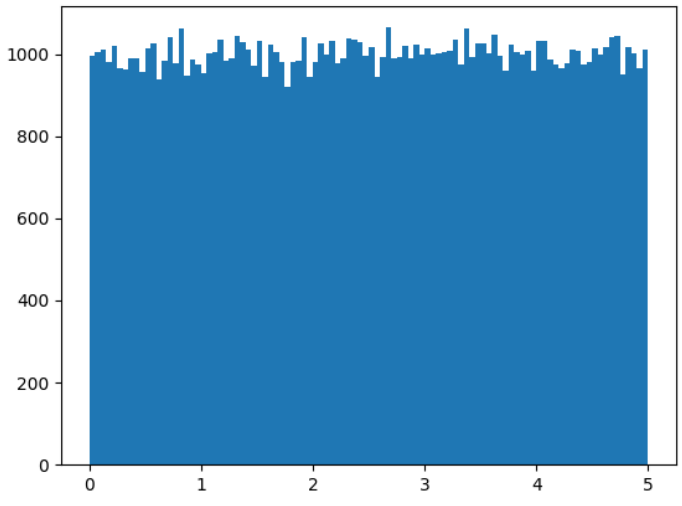
Nota: Os valores da matriz são números aleatórios e não mostrarão exatamente o mesmo resultado no seu computador.

Distribuições de Big Data

Um array contendo 250 valores não é considerado muito grande, mas agora você sabe como criar um conjunto aleatório de valores e, alterando os parâmetros, pode criar o conjunto de dados do tamanho que desejar.

Crie um array com 100.000 números aleatórios e exiba-os usando um histograma com 100 barras:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 100000)

plt.hist(x, 100)
plt.show()
```



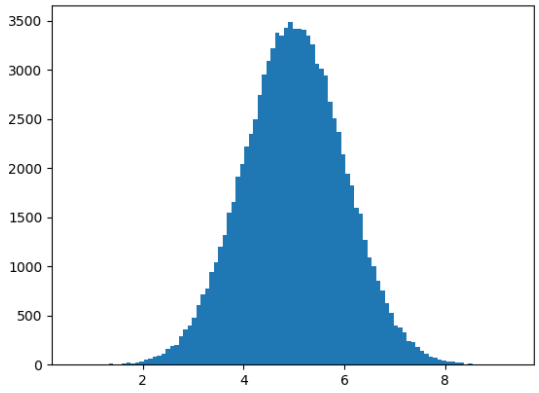
Distribuição Normal de Dados

Normal Data Distribution

Na teoria das probabilidades, esse tipo de distribuição de dados é conhecido como *distribuição normal de dados* , ou *distribuição gaussiana de dados* , em homenagem ao matemático Carl Friedrich Gauss que criou a fórmula dessa distribuição de dados.

Uma distribuição normal típica de dados:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100000)

plt.hist(x, 100)
plt.show()
```



Nota: Um gráfico de distribuição normal também é conhecido como curva em sino devido ao seu formato característico de sino.

Histograma explicado

Utilizamos o array do `numpy.random.normal()` método, com 100.000 valores, para desenhar um histograma com 100 barras.

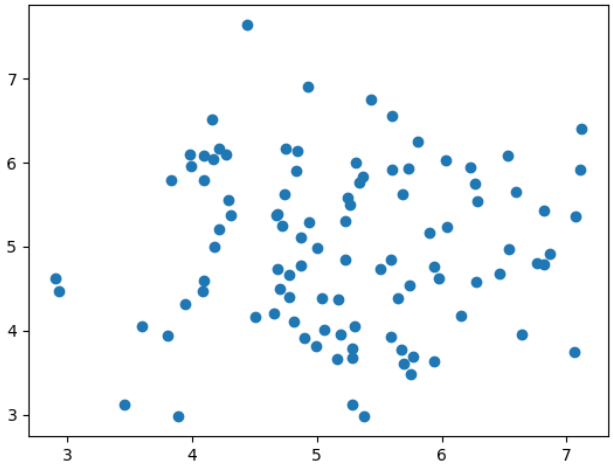
Especificamos que o valor médio é 5,0 e o desvio padrão é 1,0.

O que significa que os valores devem estar concentrados em torno de 5,0, e raramente mais distantes que 1,0 da média.

E como você pode ver no histograma, a maioria dos valores está entre 4,0 e 6,0, com um máximo em aproximadamente 5,0.

Gráfico de Dispersão - Scatter Plot

Um gráfico de dispersão é um diagrama onde cada valor no conjunto de dados é representado por um ponto.



O módulo Matplotlib possui um método para desenhar gráficos de dispersão, ele precisa de dois arrays do mesmo comprimento, um para os valores do eixo x e outro para os valores do eixo y:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

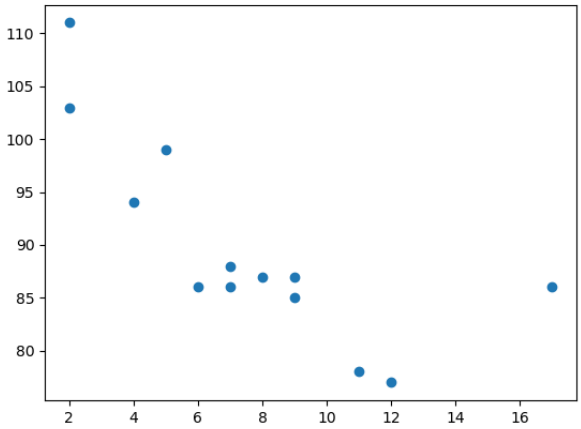
```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

A `x`matriz representa a idade de cada carro.

A `y`matriz representa a velocidade de cada carro.

Use o `scatter()`método para desenhar um diagrama de dispersão:

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```



Gráficos de dispersão explicados

O eixo x representa idades e o eixo y representa velocidades.

O que podemos ler no diagrama é que os dois carros mais rápidos tinham ambos 2 anos e o carro mais lento tinha 12 anos.

Nota: Parece que quanto mais novo o carro, mais rápido ele anda, mas isso pode ser coincidência, afinal registramos apenas 13 carros.

Distribuições aleatórias de dados

Random Data Distributions

No Machine Learning, os conjuntos de dados podem conter milhares ou até milhões de valores.

Talvez você não tenha dados do mundo real ao testar um algoritmo; talvez seja necessário usar valores gerados aleatoriamente.

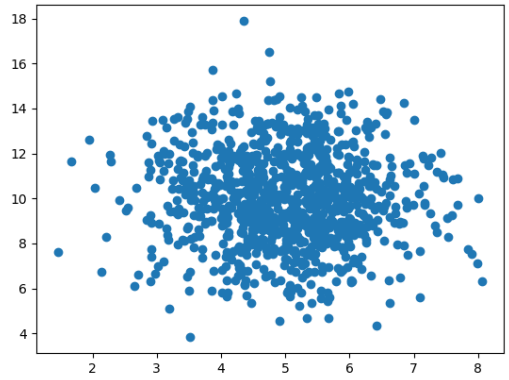
Vamos criar duas matrizes preenchidas com 1.000 números aleatórios de uma distribuição normal de dados.

A primeira matriz terá a média definida como 5,0 com desvio padrão de 1,0.

A segunda matriz terá a média definida como 10,0 com um desvio padrão de 2,0:

Um gráfico de dispersão com 1000 pontos:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

plt.scatter(x, y)
plt.show()
```



Gráficos de dispersão explicados

Podemos ver que os pontos estão concentrados em torno do valor 5 no eixo x e 10 no eixo y.

Também podemos ver que o spread é maior no eixo y do que no eixo x.

Regressão Linear

Regressão

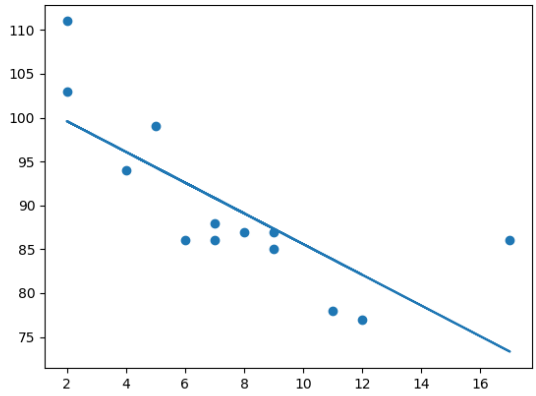
O termo regressão é usado quando você tenta encontrar a relação entre variáveis.

No Machine Learning e na modelagem estatística, esse relacionamento é usado para prever o resultado de eventos futuros.

Regressão linear

A regressão linear usa o relacionamento entre os pontos de dados para traçar uma linha reta através de todos eles.

Esta linha pode ser usada para prever valores futuros.



No Machine Learning, prever o futuro é muito importante.

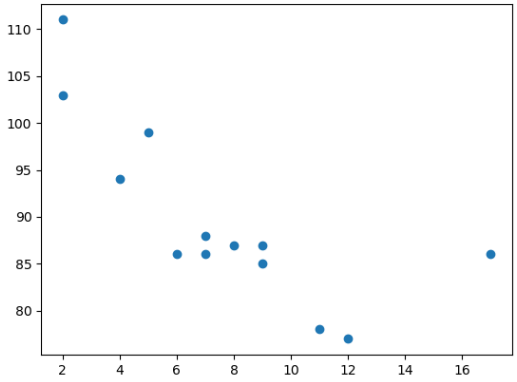
Como funciona?

Python possui métodos para encontrar um relacionamento entre pontos de dados e traçar uma linha de regressão linear. Mostraremos como usar esses métodos em vez de seguir a fórmula matemática.

No exemplo abaixo, o eixo x representa a idade e o eixo y representa a velocidade. Registramos a idade e a velocidade de 13 carros ao passarem por um pedágio. Vejamos se os dados que coletamos poderiam ser usados em uma regressão linear:

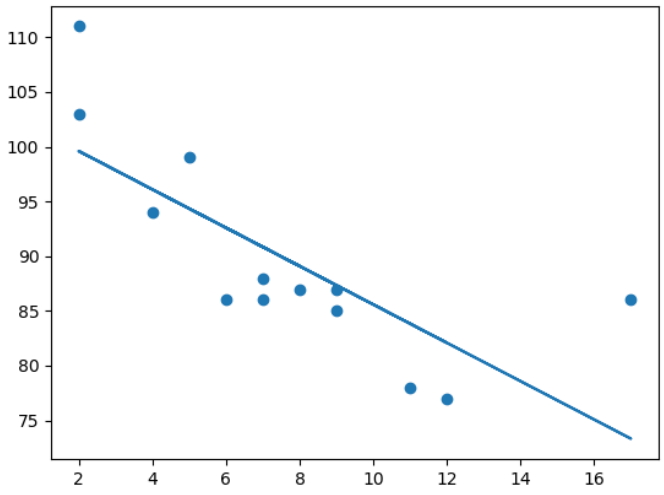
Comece desenhando um gráfico de dispersão:

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```



Importe `scipy` e desenhe a linha de Regressão Linear:

```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



Exemplo explicado

Importe os módulos que você precisa.

```
import matplotlib.pyplot as plt
from scipy import stats
```

Crie os arrays que representam os valores dos eixos x e y:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Execute um método que retorne alguns valores-chave importantes da regressão linear:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Crie uma função que use os valores `slope`e `intercept`para retornar um novo valor. Este novo valor representa onde no eixo y o valor x correspondente será colocado:

```
def myfunc(x):
  return slope * x + intercept
```

Execute cada valor do array x por meio da função. Isso resultará em um novo array com novos valores para o eixo y:

```
mymodel = list(map(myfunc, x))
```

Desenhe o gráfico de dispersão original:

```
plt.scatter(x, y)
```

Desenhe a linha de regressão linear:

```
plt.plot(x, mymodel)
```

Exiba o diagrama:

```
plt.show()
```

R para relacionamentos

É importante saber como é a relação entre os valores do eixo x e os valores do eixo y, se não houver relação a regressão linear não pode ser usada para prever nada.

Essa relação – o coeficiente de correlação – é chamada r.

O rvalor varia de -1 a 1, onde 0 significa nenhum relacionamento e 1 (e -1) significa 100% relacionado.

Python e o módulo Scipy calcularão esse valor para você, tudo que você precisa fazer é alimentá-lo com os valores xey.

Até que ponto meus dados se ajustam a uma regressão linear?

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

print(r)
```

Nota: O resultado -0,76 mostra que existe uma relação, não perfeita, mas indica que poderíamos usar a regressão linear em previsões futuras.

Prever Valores Futuros

Agora podemos usar as informações que reunimos para prever valores futuros.

Exemplo: Vamos tentar prever a velocidade de um carro de 10 anos.

Para fazer isso, precisamos da mesma `myfunc()`função do exemplo acima:

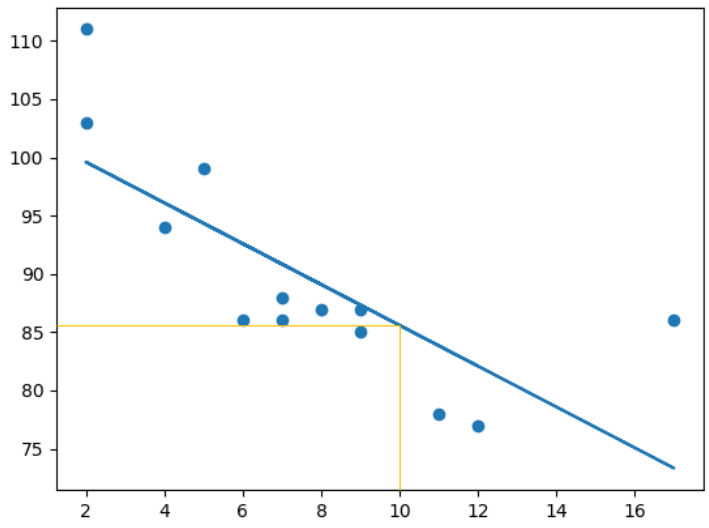
```
def myfunc(x):
  return slope * x + intercept
```

Exemplo

Preveja a velocidade de um carro de 10 anos:

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

speed = myfunc(10)

print(speed)
```

O exemplo previu uma velocidade de 85,6, que também podemos ler no diagrama:



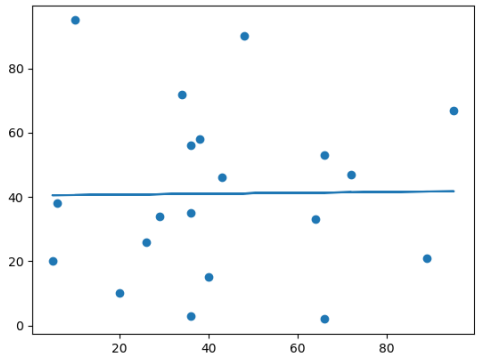
Ajuste ruim?

Vamos criar um exemplo onde a regressão linear não seria o melhor método para prever valores futuros.

Exemplo

Esses valores para os eixos x e y devem resultar em um ajuste muito ruim para regressão linear:

```
import matplotlib.pyplot as plt
from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



E o r para relacionamento?

Exemplo

Você deve obter um `r` valor muito baixo.

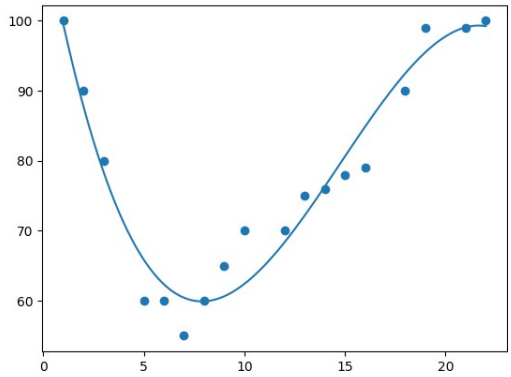
```
import numpy
from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

slope, intercept, r, p, std_err = stats.linregress(x, y)

print(r)
```

O resultado: 0,013 indica um relacionamento muito ruim e nos diz que este conjunto de dados não é adequado para regressão linear.

Regressão Polinomial

Se seus pontos de dados claramente não se ajustarem a uma regressão linear (uma linha reta que passa por todos os pontos de dados), pode ser ideal para regressão polinomial.

A regressão polinomial, assim como a regressão linear, usa a relação entre as variáveis xey para encontrar a melhor maneira de traçar uma linha através dos pontos de dados.



Como funciona?

Python possui métodos para encontrar um relacionamento entre pontos de dados e desenhar uma linha de regressão polinomial. Mostraremos como usar esses métodos em vez de seguir a fórmula matemática.

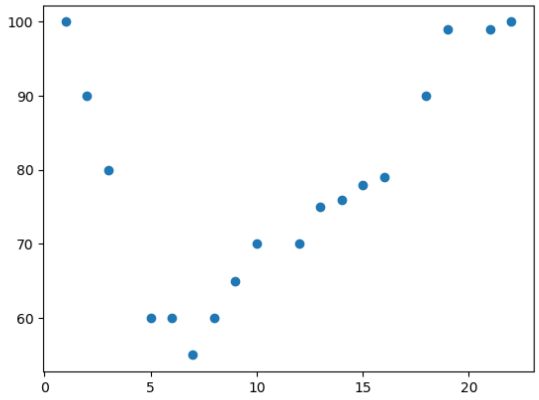
No exemplo abaixo, registramos 18 carros ao passarem por um determinado pedágio.

Registramos a velocidade do carro e a hora do dia (hora) em que ocorreu a ultrapassagem.

O eixo x representa as horas do dia e o eixo y representa a velocidade:

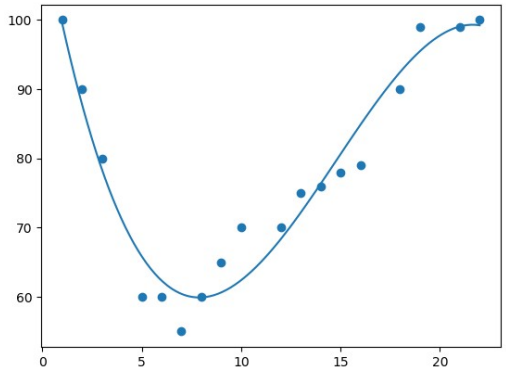
Comece desenhando um gráfico de dispersão:

```
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

plt.scatter(x, y)
plt.show()
```



Importe numpye matplotlibdesenhe a linha de Regressão Polinomial:

```
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



Exemplo explicado

Importe os módulos que você precisa.

```
import numpy
import matplotlib.pyplot as plt
```

Crie os arrays que representam os valores dos eixos x e y:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

NumPy possui um método que nos permite fazer um modelo polinomial:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Em seguida, especifique como a linha será exibida, começamos na posição 1 e terminamos na posição 22:

```
myline = numpy.linspace(1, 22, 100)
```

Desenhe o gráfico de dispersão original:

```
plt.scatter(x, y)
```

Desenhe a linha de regressão polinomial:

```
plt.plot(myline, mymodel(myline))
```

Exiba o diagrama:

```
plt.show()
```

R-Squared

R-quadrado

É importante saber quão bem está a relação entre os valores dos eixos xey, se não houver relação a regressão polinomial não pode ser usada para prever nada.

A relação é medida com um valor denominado r-quadrado.

O valor de r ao quadrado varia de 0 a 1, onde 0 significa nenhum relacionamento e 1 significa 100% relacionado.

Python e o módulo Sklearn calcularão esse valor para você, tudo o que você precisa fazer é alimentá-lo com os arrays x e y:

Exemplo

Quão bem meus dados se ajustam em uma regressão polinomial?

```
import numpy
from sklearn.metrics import r2_score

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

Nota: O resultado 0,94 mostra que existe uma relação muito boa, e podemos usar regressão polinomial em previsões futuras.

Prever Valores Futuros

Agora podemos usar as informações que reunimos para prever valores futuros.

Exemplo: Vamos tentar prever a velocidade de um carro que passa no pedágio por volta das 17h:

Para fazer isso, precisamos do mesmo `mymodel`array do exemplo acima:

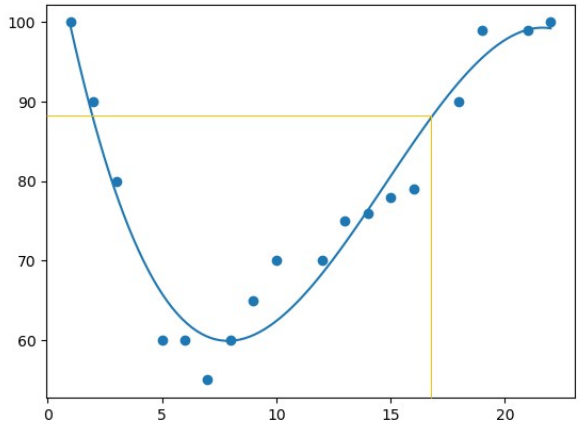
```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Exemplo

Preveja a velocidade de um carro passando às 17h:

```
import numpy
from sklearn.metrics import r2_score

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

speed = mymodel(17)
print(speed)
```

O exemplo previu uma velocidade de 88,87, que também podemos ler no diagrama:



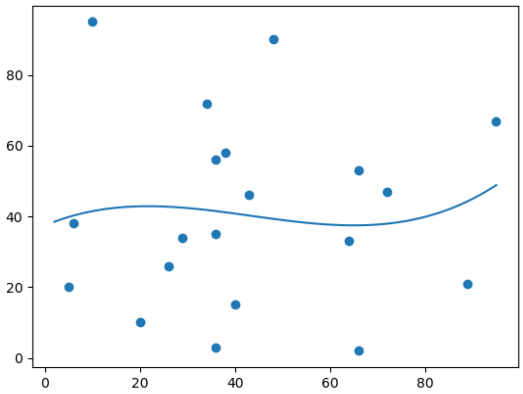
Ajuste ruim?

Vamos criar um exemplo onde a regressão polinomial não seria o melhor método para prever valores futuros.

Exemplo

Esses valores para os eixos x e y devem resultar em um ajuste muito ruim para a regressão polinomial:

```
import numpy
import matplotlib.pyplot as plt

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(2, 95, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



E o valor de r ao quadrado?

Exemplo

Você deve obter um valor de r ao quadrado muito baixo.

```
import numpy
from sklearn.metrics import r2_score

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

O resultado: 0,00995 indica um relacionamento muito ruim e nos diz que este conjunto de dados não é adequado para regressão polinomial.

Regressão Múltipla

A regressão múltipla é como a regressão linear , mas com mais de um valor independente, o que significa que tentamos prever um valor com base em **duas ou mais** variáveis.

Dê uma olhada no conjunto de dados abaixo, ele contém algumas informações sobre carros.

Caracteres	Modelo	Volume	Peso	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Estrela Espacial	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Tanoeiro	1500	1140	105
VW	Acima!	1000	929	105
Skoda	Fábia	1400	1109	90
Mercedes	Uma aula	1500	1365	noventa e dois
Ford	Festa	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Rápido	1300	990	101
Ford	Festa	1000	1112	99
Honda	Cívico	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rápido	1600	1119	104
Ford	Foco	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insígnia	2000	1428	99
Mercedes	Classe C	2100	1365	99
Skoda	Otávia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	Classe E	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Máx.	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

Podemos prever a emissão de CO2 de um carro com base no tamanho do motor, mas com a regressão múltipla podemos incluir mais variáveis, como o peso do carro, para tornar a previsão mais precisa.

Como funciona?

Em Python temos módulos que farão o trabalho para nós. Comece importando o módulo Pandas.

```
import pandas
```

O módulo Pandas nos permite ler arquivos csv e retornar um objeto DataFrame.

O arquivo destina-se apenas para fins de teste: data.csv.

```
df = pandas.read_csv("data.csv")
```

Depois faça uma lista dos valores independentes e chame essa variável `X`.

Coloque os valores dependentes em uma variável chamada `y`.

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

Dica: É comum nomear a lista de valores independentes com X maiúsculo, e a lista de valores dependentes com y minúsculo.

Usaremos alguns métodos do módulo sklearn, então teremos que importar esse módulo também:

```
from sklearn import linear_model
```

No módulo sklearn usaremos o `LinearRegression()`método para criar um objeto de regressão linear.

Este objeto possui um método chamado `fit()`que toma os valores independentes e dependentes como parâmetros e preenche o objeto de regressão com dados que descrevem o relacionamento:

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Agora temos um objeto de regressão pronto para prever os valores de CO2 com base no peso e volume de um carro:

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])
```

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

Resultado:

```
[107.2087328]
```

Coeficiente

O coeficiente é um fator que descreve a relação com uma variável desconhecida.

Exemplo: se xé uma variável, então $2x$é $x$duas vezes. xé a variável desconhecida e o número 2é o coeficiente.

Neste caso, podemos solicitar o valor do coeficiente de peso em relação ao CO2 e de volume em relação ao CO2. A(s) resposta(s) que obtemos dizem-nos o que aconteceria se aumentássemos ou diminuíssemos um dos valores independentes.

Exemplo

Imprima os valores dos coeficientes do objeto de regressão:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

Resultado:

```
[0,00755095 0,00780526]
```

Resultado explicado

A matriz de resultados representa os valores dos coeficientes de peso e volume.

Peso: 0,00755095
Volume: 0,00780526

Estes valores dizem-nos que se o peso aumentar em 1kg, a emissão de CO2 aumenta em 0,00755095g.

E se o tamanho do motor (Volume) aumentar em 1 cm^3 , a emissão de CO2 aumenta em 0,00780526 g.

Acho que é um palpite justo, mas vamos testar!

Já previmos que se um carro com motor de 1300cm3 pesar 2300kg a emissão de CO2 será de aproximadamente 107g.

E se aumentarmos o peso em 1000kg?

Exemplo

Copie o exemplo anterior, mas altere o peso de 2300 para 3300:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Resultado:

```
[114.75968007]
```

Previmos que um carro com motor de 1,3 litro e peso de 3.300 kg liberará aproximadamente 115 gramas de CO2 por cada quilômetro percorrido.

O que mostra que o coeficiente de 0,00755095 está correto:

107,2087328 + (1000 * 0,00755095) = 114,75968

Escala

Quando seus dados têm valores diferentes e até mesmo unidades de medida diferentes, pode ser difícil compará-los. O que são quilogramas em comparação com metros? Ou altitude em comparação com o tempo?

A resposta para esse problema é a escala. Podemos dimensionar os dados em novos valores que são mais fáceis de comparar.

Dê uma olhada na tabela abaixo, é o mesmo conjunto de dados que usamos em regressão múltipla , mas desta vez a coluna **de volume** contém valores em litros em vez de cm ^{3} (1,0 em vez de 1000).

Caracteres	Modelo	Volume	Peso	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Estrela Espacial	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Tanoeiro	1500	1140	105
VW	Acima!	1000	929	105
Skoda	Fábia	1400	1109	90
Mercedes	Uma aula	1500	1365	noventa e dois
Ford	Festa	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Rápido	1300	990	101
Ford	Festa	1000	1112	99
Honda	Cívico	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rápido	1600	1119	104
Ford	Foco	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insígnia	2000	1428	99
Mercedes	Classe C	2100	1365	99
Skoda	Otávia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	Classe E	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Máx.	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

Pode ser difícil comparar o volume 1.0 com o peso 790, mas se escalarmos ambos em valores comparáveis, poderemos ver facilmente o quanto um valor é comparado ao outro.

Existem diferentes métodos para dimensionar dados, usaremos um método chamado padronização.

O método de padronização usa esta fórmula:

`z = (x - u) / s`

Onde `z` está o novo valor, `x` é o valor original, `u` é a média e `s` é o desvio padrão.

Se você pegar a coluna **de peso** do conjunto de dados acima, o primeiro valor será 790 e o valor dimensionado será:

`(790 - 1292.23) / 238.74 = -2.1`

Se você pegar a coluna **de volume** do conjunto de dados acima, o primeiro valor será 1,0 e o valor dimensionado será:

`(1.0 - 1.61) / 0.38 = -1.59`

Agora você pode comparar -2,1 com -1,59 em vez de comparar 790 com 1,0.

Você não precisa fazer isso manualmente, o módulo sklearn do Python possui um método chamado `StandardScaler()` que retorna um objeto Scaler com métodos para transformar conjuntos de dados.

Dimensione todos os valores nas colunas Peso e Volume:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]

scaledX = scale.fit_transform(X)

print(scaledX)
```

Resultado:

Observe que os dois primeiros valores são -2,1 e -1,59, o que corresponde aos nossos cálculos:

```
[[-2.10389253 -1.59336644]
 [-0,55407235 -1,07190106]
 [-1,52166278 -1,59336644]
 [-1.78973979 -1.85409913]
 [-0,63784641 -0,28970299]
 [-1,52166278 -1,59336644]
 [-0,76769621 -0,55043568]
 [0,3046118 -0,28970299]
 [-0,7551301 -0,28970299]
 [-0,59595938 -0,0289703]
 [-1.30803892 -1.33263375]
 [-1,26615189 -0,81116837]
 [-0,7551301 -1,59336644]
 [-0,16871166 -0,0289703]
 [0,14125238 -0,0289703]
 [0,15800719 -0,0289703]
 [0,3046118 -0,0289703]
 [-0,05142797 1,53542584]
 [-0,72580918 -0,0289703]
 [0,14962979 1,01396046]
 [1,2219378 -0,0289703]
 [0,5685001 1,01396046]
 [0,3046118 1,27469315]
 [0,51404696 -0,0289703]
 [0,51404696 1,01396046]
 [0,72348212 -0,28970299]
 [0,8281997 1,01396046]
 [1,81254495 1,01396046]
 [0,96642691 -0,0289703]
 [1,72877089 1,01396046]
 [1.30990057 1.27469315]
 [1.90050772 1.01396046]
 [-0,23991961 -0,0289703]
 [0,40932938 -0,0289703]
 [0,47215993 -0,0289703]
 [0,4302729 2,31762392]]
```

Prever valores de CO2

A tarefa de Regressão Múltipla era prever a emissão de CO2 de um carro quando você conhecia apenas seu peso e volume.

Quando o conjunto de dados for dimensionado, você terá que usar a escala ao prever valores:

Exemplo

Preveja a emissão de CO2 de um carro de 1,3 litro que pesa 2.300 kg:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

scaledX = scale.fit_transform(X)

regr = linear_model.LinearRegression()
regr.fit(scaledX, y)

scaled = scale.transform([[2300, 1.3]])

predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

Resultado:

```
[107.2087328]
```

Treinar/Teste - Train/Test

Avalie seu modelo

No Machine Learning criamos modelos para prever o resultado de determinados eventos, como no capítulo anterior, onde previmos a emissão de CO2 de um carro quando sabíamos o peso e o tamanho do motor.

Para medir se o modelo é bom o suficiente, podemos usar um método chamado Train/Test.

O que é treinamento/teste

Treinar/Teste é um método para medir a precisão do seu modelo.

É chamado de Treinamento/Teste porque você divide o conjunto de dados em dois conjuntos: um conjunto de treinamento e um conjunto de teste.

80% para treinamento e 20% para teste.

Você treina o modelo usando o conjunto de treinamento.

Você testa o modelo usando o conjunto de testes.

Treinar o modelo significa criar o modelo.

Testar o modelo significa testar a precisão do modelo.

Comece com um conjunto de dados

Comece com um conjunto de dados que você deseja testar.

Nosso conjunto de dados ilustra 100 clientes em uma loja e seus hábitos de compra.

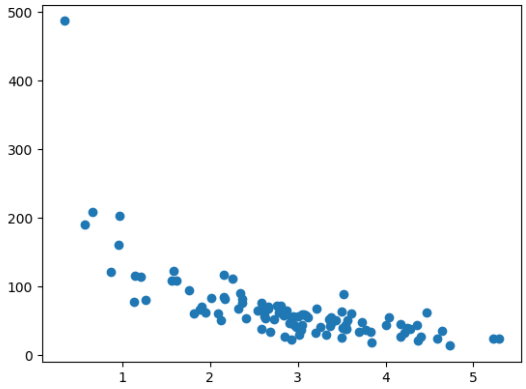
Exemplo

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)
plt.show()
```

Resultado:

O eixo x representa o número de minutos antes de fazer uma compra.

O eixo y representa a quantidade de dinheiro gasto na compra.



Dividir em treinamento/teste

O conjunto de treinamento deve ser uma seleção aleatória de 80% dos dados originais.

O conjunto de teste deve ser os 20% restantes.

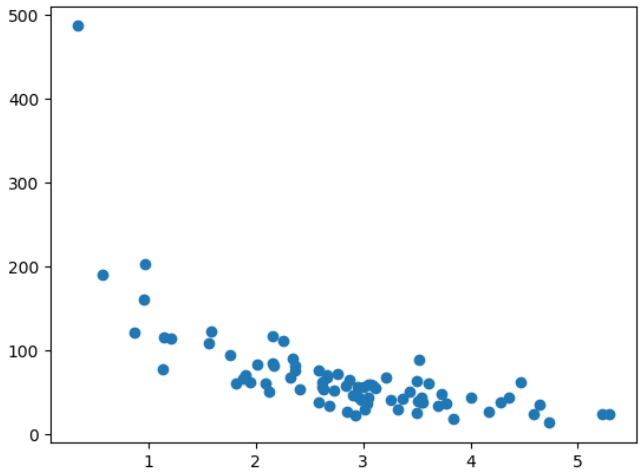
```
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]
```

Exibir o conjunto de treinamento

Exiba o mesmo gráfico de dispersão com o conjunto de treinamento:

Exemplo

```
plt.scatter(train_x, train_y)
plt.show()
```



Exibir o conjunto de testes

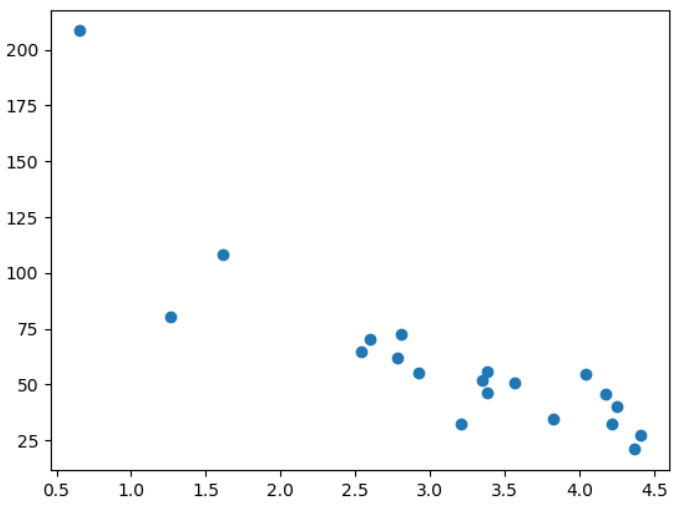
Para garantir que o conjunto de testes não seja completamente diferente, daremos uma olhada no conjunto de testes também.

Exemplo

```
plt.scatter(test_x, test_y)
plt.show()
```

Resultado:

O conjunto de testes também se parece com o conjunto de dados original:



Ajustar o conjunto de dados

Qual é a aparência do conjunto de dados? Na minha opinião, acho que o melhor ajuste seria uma regressão polinomial , então vamos traçar uma linha de regressão polinomial.

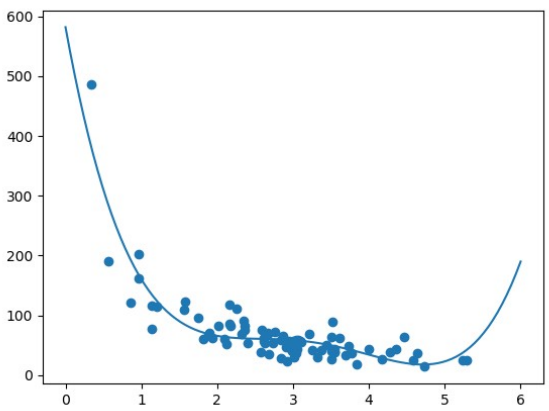
Para traçar uma linha através dos pontos de dados, usamos o `plot()`método do módulo matplotlib:

Exemplo

Desenhe uma linha de regressão polinomial através dos pontos de dados:

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

myline = numpy.linspace(0, 6, 100)

plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Resultado:



O resultado pode apoiar a sugestão de que o conjunto de dados se ajuste a uma regressão polinomial, embora isso nos desse alguns resultados estranhos se tentarmos prever valores fora do conjunto de dados. Exemplo: a linha indica que um cliente que passasse 6 minutos na loja faria uma compra no valor de 200. Isso é provavelmente um sinal de overfitting.

Mas e quanto à pontuação R ao quadrado? A pontuação R ao quadrado é um bom indicador de quão bem meu conjunto de dados se ajusta ao modelo.

R2

Lembra do R2, também conhecido como R ao quadrado?

Mede a relação entre o eixo x e o eixo y, e o valor varia de 0 a 1, onde 0 significa nenhum relacionamento e 1 significa totalmente relacionado.

O módulo sklearn possui um método chamado `r2_score()`que nos ajudará a encontrar esse relacionamento.

Neste caso gostaríamos de medir a relação entre os minutos que um cliente permanece na loja e quanto dinheiro gasta.

Exemplo

Até que ponto meus dados de treinamento se ajustam a uma regressão polinomial?

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(train_y, mymodel(train_x))

print(r2)
```

Nota: O resultado 0,799 mostra que existe um relacionamento OK.

Traga o conjunto de testes

Agora criamos um modelo que está OK, pelo menos quando se trata de dados de treinamento.

Agora queremos testar o modelo também com os dados de teste, para ver se ele nos dá o mesmo resultado.

Exemplo

Vamos encontrar a pontuação R2 ao usar dados de teste:

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(test_y, mymodel(test_x))

print(r2)
```

Nota: O resultado 0,809 mostra que o modelo também se ajusta ao conjunto de testes e estamos confiantes de que podemos usar o modelo para prever valores futuros.

Prever Valores

Agora que estabelecemos que nosso modelo está OK, podemos começar a prever novos valores.

Exemplo

Quanto dinheiro um cliente comprador gastará se permanecer na loja por 5 minutos?

```
print(mymodel(5))
```

O exemplo previu que o cliente gastaria 22,88 dólares, como parece corresponder ao diagrama: