Regressão Linear Simples

O Modelo

Lembre-se de que estávamos investigando a **relação entre o número de amigos de um usuário da DataSciencester e a quantidade de tempo que ele passou no site** todos os dias.
Suponhamos que você se convenceu de que **ter mais amigos faz com que as pessoas passem mais tempo no site**, em vez de usar uma das explicações alternativas que discutimos.
Você decide criar um modelo descrevendo esse relacionamento. Dado que você encontrou um **relacionamento linear muito forte**, é natural para começar por um **modelo linear**. Em particular, você supõe que existem constantes α (alfa) e β (beta) tais que:

$yi = \beta xi + \alpha + \varepsilon i$

onde ***yi*** é o número de minutos que o usuário ***i*** gasta no site diariamente, ***xi*** é o número de amigos que o usuário ***i*** possui e ***εi*** é um termo de erro (esperançosamente pequeno) que representa o fato de que existem outros fatores não contabilizados por este modelo simples.

Supondo que tenhamos determinado alfa e beta, fazemos previsões simplesmente com:

In [6]:
```
def predict(alpha, beta, x_i):
    return beta * x_i + alpha
```

Visualizando os dados

In [2]:
```
from collections import Counter
import matplotlib.pyplot as plt
import random
import math
import numpy as np
```

In [3]:
```
num_friends = [10, 49, 41, 40, 25, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4,
               15, 15, 15, 18, 20, 20]

daily_min = [18, 39, 37, 35, 28, 7, 9, 8, 7, 8, 10, 11, 12, 9, 13, 15, 14, 14
               25, 27, 29, 28, 30, 32]
```

Abaixo, as funções estatísticas que aprendemos em Ciência de Dados I

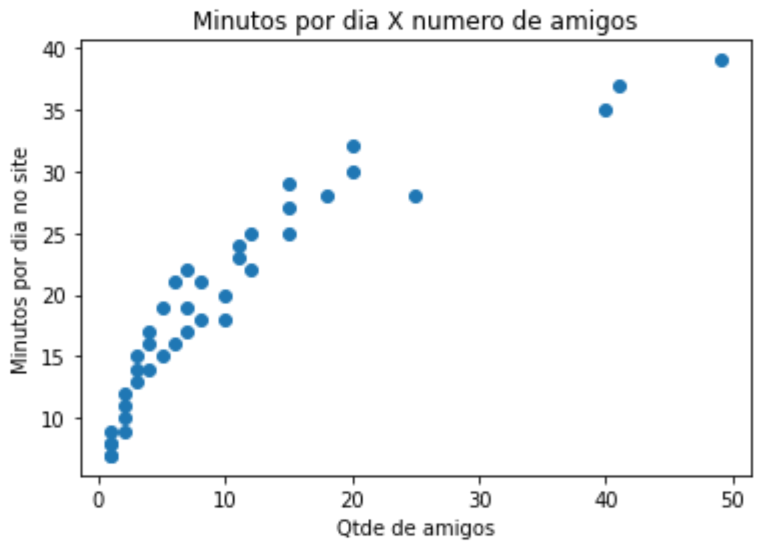
In [4]:

```
def mean(x):
    return sum(x) / len(x)

def de_mean(x):
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]

def dot(v, w):
    return sum(v_i * w_i
        for v_i, w_i in zip(v, w))

def sum_of_squares(x):
    return sum([x_i * x_i for x_i in x])

def variance(x):
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)

def standard_deviation(x):
    return math.sqrt(variance(x))

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n - 1)

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y
    else:
        return
```

Plotando o gráfico de espalhamento para visualizar os dados

In [5]:

```
plt.scatter(num_friends, daily_min)
plt.title("Minutos por dia X numero de amigos")
plt.xlabel("Qtde de amigos")
plt.ylabel("Minutos por dia no site")
plt.show()
```



Como definir as parâmetros da função

Como escolhemos alfa e beta?

Bem, qualquer escolha de alfa e beta nos dá uma saída prevista para cada entrada **x_i**.
Como sabemos a saída verdadeira **y_i**, podemos **calcular o erro** de cada par:

In [7]:
```
def error(alpha, beta, x_i, y_i):
    """the error from predicting beta * x_i + alpha when the actual value is
    return y_i - predict(alpha, beta, x_i)
```

O que realmente gostaríamos de saber é o **erro total em todo o conjunto de dados**.
Mas não queremos apenas adicionar os erros pois se a previsão para x_1 for muito alta e a previsão para x_2 for muito baixa, os erros poderão ser anulados.
Então, em vez disso, **somamos os erros quadrados**:

In [8]:
```
def sum_of_squared_errors(alpha, beta, x, y):
    return sum(error(alpha, beta, x_i, y_i) ** 2
               for x_i, y_i in zip(x, y))
```

A solução de **mínimos quadrados** visa escolher o alfa e o beta que tornam o **sum_of_squared_errors** o menor possível.
Usando o cálculo, os alfa e beta minimizadores de erros são dados por:

In [9]:
```
def least_squares_fit(x, y):
    """given training values for x and y, find the least-squares values of al
    beta = correlation(x, y) * standard_deviation(y) / standard_deviation(x)
    alpha = mean(y) - beta * mean(x)
    return alpha, beta
```

Sem passar pela matemática exata, vamos pensar no por que essa pode ser uma solução razoável:

- A escolha do **alfa** simplesmente diz que dado o valor médio da variável

independente x, predizemos o valor médio da variável dependente y.
- A escolha de **beta** significa que quando o valor de entrada aumenta por desvio_padrao(x), a previsão aumenta pela correlação(x, y) * desvio_padrao(y). No caso em que x e y estão perfeitamente correlacionados, um aumento de um desvio padrão em x resulta em um aumento de um desvio-padrão-de-y na predição.

Quando eles são perfeitamente correlacionados, o aumento no desvio padrão em x resulta em um aumento no desvio padrão de y na previsão. Quando eles são perfeitamente não correlacionados (anti-correlação) um aumento em x resulta em uma queda na previsão. E quando a correlação é zero, o beta é zero, o que significa que as alterações em x não afetam a previsão.

Aplicando a base de dados de users e tempo online, sem outliers:

In [10]:
```
alpha, beta = least_squares_fit(num_friends, daily_min)
print("Alfa=", alpha, "\nBeta=", beta)
```

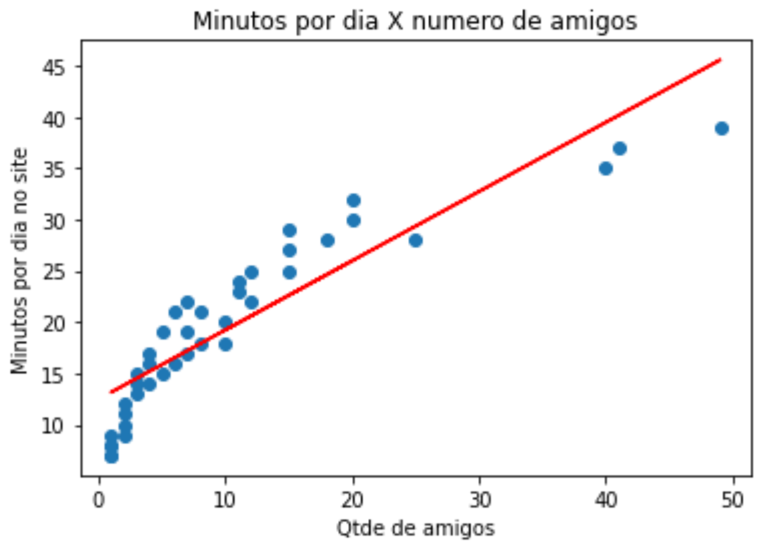
```
Alfa= 12.461446974221035
Beta= 0.6751609617054163
```

Isso fornece valores de alfa = 12.46 e beta = 0.67.

Portanto, nosso modelo diz que esperamos que um usuário com **n** amigos gaste **12.46 + n * 0.67 minutos** no site todos os dias. Ou seja, prevemos que **um usuário sem amigos** no DataSciencester ainda gastaria cerca de **12 minutos por dia** no site. E para **cada amigo adicional**, esperamos que um usuário passe quase **mais um minuto no site** todos os dias.

Na figura abaixo, plotamos a linha de previsão para ter uma noção de quão bem o modelo se ajusta aos dados observados.

In [11]:
```
pred_daily_min = [predict(alpha, beta, i) for i in num_friends]
plt.scatter(num_friends, daily_min)
plt.plot(num_friends, pred_daily_min, color='red', linestyle='solid')
plt.title("Minutos por dia X numero de amigos")
plt.xlabel("Qtde de amigos")
plt.ylabel("Minutos por dia no site")
plt.show()
```



É claro que precisamos de uma maneira melhor, do que simplesmente analisar o gráfico, de descobrir o quão bem nós ajustamos os dados.
Uma medida comum é o **coeficiente de determinação (ou R-quadrado)**, que mede a distância entre os pontos e a reta, isso determina o coeficiente de determinação ou (R2).
Em suma, **R2 é a razão entre a soma de quadrados da regressão e a soma de quadrados total.**

In [12]:

```
def total_sum_of_squares(y):
    """the total squared variation of y_i's from their mean"""
    return sum(v ** 2 for v in de_mean(y))

def r_squared(alpha, beta, x, y):
    """the fraction of variation in y captured by the model, which equals
    1 - the fraction of variation in y not captured by the model"""
    return 1.0 - (sum_of_squared_errors(alpha, beta, x, y) / total_sum_of_squ

r_squared(alpha, beta, num_friends, daily_min) # 0.329 na base do livro
```

Out[12]: 0.8121973872303889

Agora, escolhemos o alfa e o beta que minimizaram a soma dos erros de previsão ao quadrado.

A soma dos erros quadrados deve ser no mínimo 0 (melhor caso), o que significa que o **R-quadrado pode ser no máximo 1.**

Quanto maior o número, melhor o nosso modelo se ajusta aos dados.
Aqui nós calculamos um R ao quadrado de 0,812, o que nos diz que o nosso modelo está bom em ajustar os dados, contudo claramente existem outros fatores em jogo.

Atividade 1

- Usando Random crie 50 números aleatórios para as listas de quantidade de amigos e

quantidade de minutos online diariamente.
- Encontre o **alpha** e o **beta**
- Plot o gráfico de dispersão com a linha da função
- Exiba o R2 da função

Resposta:

Atividade 2

Dada uma base de dados sobre o consumo de oxigênio em função do tempo despendido para percorrer 1,5 milhas, faça:

- Carregue a base de dados em uma matriz com duas colunas: tempo e consumo de oxigênio.
- Exiba o índice de correlação dos dados
- Encontre o **alpha** e o **beta** para a função de regressão linear
- Plot o gráfico de dispersão com a linha que ilustra a função
- Exiba o R2 da função
- Informe a predição de consumo de oxigênio necessários para tempos de 15, 17 e 19 minutos.

Resposta:

In [ ]: