> Anteriormente, usamos a função de correlação para medir a força da relação linear entre duas variáveis. Para a maioria das aplicações, saber que existe tal relação linear não é suficiente. Queremos entender a natureza do relacionamento. É aqui que usaremos a **regressão linear simples**.

Modelo

Lembre-se de que estávamos investigando a relação entre o número de amigos de um usuário da *Facedata* e a quantidade de tempo que o usuário passa no site todos os dias. Vamos supor que, de fato, ter mais amigos faz com que as pessoas passem mais tempo no site.

Nossa *CEO* pede que você construa um modelo que descreva esse relacionamento. Como você encontrou um relacionamento linear bastante forte, um lugar natural para começar é um modelo linear (muito similar à uma equação de 1o grau). Em particular, você supõe que existem constantes α (alfa) e β (beta) tais que:

$$y = \alpha + \beta x + \varepsilon,$$

onde

- y é o número de minutos que um usuário gasta no site diariamente,
- x é o número de amigos (na rede) que um usuário tem e
- ε é um termo de erro aleatório que representa o fato de que existem outros fatores não contabilizados por este modelo simples.

Suposições

As suposições (ou premissas) desempenham um papel fundamental no modelo de regressão linear, pois são os pressupostos básicos que precisam ser atendidos para que os resultados obtidos sejam confiáveis e interpretações corretas possam ser feitas. Essas premissas são essenciais para autenticar a validade e a eficácia do modelo. Não iremos listar todos por conta do nível deste curso, mas podemos citar algumas como:

1. **Linearidade**: Estabelece que a relação entre a variável *output* e a variável *input* é linear. Isso significa que a relação entre as variáveis pode ser representada por uma linha reta no gráfico de dispersão. Se a relação não for linear, o modelo de regressão linear simples pode não

ser apropriado e pode ser necessário explorar outros modelos;
2. **Independência**: Assume que as observações são independentes umas das outras. Isso significa que os valores de erro para cada observação não estão correlacionados. A violação dessa premissa pode levar a resultados enviesados e interpretações incorretas das constantes (coeficientes) do modelo;

3. **Tipo de output**: Pressupõe que o *output* é um atributo **quantitativo contínuo**.

Ao considerar e verificar essas premissas, os cientista de dados garantem a robustez e a validade do modelo de regressão linear. Ao atender a essas premissas, podemos ter maior confiança nos resultados e nas interpretações feitas a partir do modelo, permitindo a tomada de decisões informadas com base na análise de regressão.

> Nem sempre todas as premissas podem ser atendidas. Por exemplo: Algumas vezes, iremos utilizar *inputs* que evoluem no tempo, indo contra a premissa de independência. Por segurança, sempre iremos verificar se o ajuste e as previsões estão representando bem a relação entre *input* e *output*.

Aprendizado

O termo **aprendizado** se refere à estimação do modelo em questão. Neste caso de regressão linear, temos interesse em descobrir qual reta (de todas as possíveis retas) se adequa melhor para descrever a relação entre *input* e *output*. Em outras palavras, o maior questionamento é apresentado na figura abaixo:

> Como escolhemos os melhores valores para α e β?

Neste curso, iremos pular os cálculos (tediosos) e veremos na prática como aplicar este modelo em nossos dados.

Prática

Em geral, sempre é seguido um mesmo protocolo para ajuste, avaliação e previsão de um modelo de aprendizado de máquina.

Organizando dados

Vamos recuperar os dados que relacionam o número de minutos que um usuário passa no site com o número de amigos que ele possui na rede:

In [ ]:
```python
# Carregando bibliotecas
import pandas as pd
import matplotlib.pyplot as plt

# Dados
num_amigos = [49, 41, 40, 25, ... ]
minutos_diarios = [68.77, 51.25, 52.08, 38.36, ... ]
```

In [ ]:
```python
# Transformando em pd.DataFrame
df_qtd_tempo = pd.DataFrame({"num_amigos": num_amigos,
                             "minutos_diarios": minutos_diarios
})
```

Verificando premissas

Estabelecemos que utilizaremos o número de amigos de um usuário para predizer o tempo (em minutos) que este mesmo usuário passa na rede. Assim:

- *input*: `num_amigos`
- *output*: `minutos_diarios`

In [ ]:
```python
# Premissa 1: Linearidade ###
# A relacao entre 'num_amigos' e 'minutos_diarios' deve ser linear
# Se a relacao e linear, o coeficiente de correlacao esta longe de zero

# Verificando a correlacao entre as variaveis
df_qtd_tempo.corr()

# Visualizando a correlacao entre as variaveis
df_qtd_tempo_sem_out.plot.scatter(x='num_amigos', y='minutos_diarios')
plt.title("Correlação com outlier")
plt.xlabel("Número de amigos")
plt.ylabel("Minutos diários")
plt.show()
```

In [ ]:
```python
# Premissa 2: Independência ###
# A independencia assumida entre os usuarios e algo mais teorico.
```

```
# Imagine que cada usuario acessa a rede em sua casa,
# de forma não programada e nao combinada...
# > Entao, a independencia e plausivel!
```

In [ ]:
```
# Premissa 3: Tipo de output ###
# A variavel 'minutos_diarios' e um atributo quantitativo continuo
df_qtd_tempo['minutos_diarios'].describe()

# Histograma
df_qtd_tempo['minutos_diarios'].plot.hist()
```

Premissas verificadas, vamos separar as bases:

Separando dados

Na aula anterior, vimos uma abordagem fundamental para evitar o sobreajuste na qual envolve separar (de forma aleatória) a base em uma parte de **treino** e outra de **teste**:

In [ ]:
```
from sklearn.model_selection import train_test_split

# Definindo conjunto de treino e teste simultaneamente
# Aqui, o conjunto de treino sera 80% dos dados
treino, teste = train_test_split(df_qtd_tempo, train_size = 0.80, random_state = 123)
```

> Note que a cada vez que você realizar esse procedimento, uma amostra diferente será obtida. Essa é a melhor forma de evitar com que sua seleção induza determinados resultados. Para efeitos de reprodução, iremos fixar um efeito para que sempre dê o mesmo resultado através do argumento `random_state` da função `train_test_split`.

Treinando o modelo

Finalmente, iremos definir e treinar nosso modelo através dos seguintes passos:

In [ ]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
import statsmodels.api as sm

# Definindo input e output - treinamento
X_treino = sm.add_constant(treino['num_amigos'])
y_treino = treino['minutos_diarios']

# Definindo o modelo de regressao linear
reg = LinearRegression()
```

In [ ]:
```
# Estimando o modelo
modelo_estimado = reg.fit(X = X_treino,
                          y = y_treino)
```

Avaliando a performance

Vimos também que a *performance* é uma métrica de medir o quão adequado um modelo está aos seus dados. No caso da regressão linear, iremos utilizar o **coeficiente de determinação**, popularmente chamado de R^2. Esta métrica varia entre 0 e 1, por vezes sendo expresso em termos percentuais. Nesse caso, expressa a quantidade da variância dos dados que é explicada pelo modelo linear. Assim, quanto maior o R^2, mais explicativo é o modelo linear, ou seja, melhor ele se ajusta aos dados. Por exemplo, um $R^2 = 0,356$ significa que o modelo linear explica 35,6% da variância do *output* a partir dos *inputs* incluídos naquele modelo:

In [ ]:
```
# Calculando o R2 - quanto maior, melhor
modelo_estimado.score(X = X_treino, y = y_treino)  # 0.356
```

> No exemplo, calculamos um R^2 de 0,356, o que nos diz que nosso modelo é **apenas razoável** em ajustar os dados e que claramente há outros fatores em jogo.
>
> Lembre-se: **Todos os modelos erram mas alguns são úteis!**

Interpretando o modelo

Parte do aprendizado para este modelo é decidir quais os melhores valores para α e β. Podemos recuperá-los utilizando:

In [ ]:
```
# alfa (tambem chamado de intercepto)
alfa = modelo_estimado.intercept_

# beta
beta = modelo_estimado.coef_[1]

print(alfa)   # 22.725
print(beta)   #  0.896
```

O modelo de regressão linear é um dos mais simples (e mais úteis - não se engane) no aprendizado de máquina. Ele nos permite uma boa interpretação do que ocorre no processo de aprendizado. Analisando os valores ajustados para $\alpha = 22,725$ e $\beta = 0,896$, nosso modelo diz que esperamos que um usuário com n amigos passe $22,725 + n \times 0,896$ minutos na rede social. Ou seja, prevemos que um usuário sem amigos na *Facedata* ainda passaria cerca de 23 minutos por dia no site. E para cada amigo adicional, esperamos que um usuário passe quase um minuto a mais no site.

Previsão

A partir da estimação de $\alpha = 22,725$ e $\beta = 0,896$, obtemos a seguinte equação:

$$y = 22,725 + 0,896x.$$

Esta será nossa regra para qualquer previsão. Utilizando `x`, ou seja, sabendo o número de amigos de um usuário, conseguiremos predizer o tempo que ele passa na rede social. Vamos utilizar o conjunto de *teste* que foi separado anteriormente:

In [ ]:
```
# Definindo input e output - teste
X_teste = sm.add_constant(teste['num_amigos'])
y_teste = teste['minutos_diarios']

# Previsao
previsao = modelo_estimado.predict(X = X_teste)
```

In [ ]:
```
# Calculando o R2 para a previsao - quanto maior, melhor
modelo_estimado.score(X = X_teste, y = y_teste)  # 0.197
```

Visualizando os resultados

Por fim, podemos visualizar os resultados - tanto o modelo resultante quanto as previsões realizadas - e comparar visualmente com os valores verdadeiros:

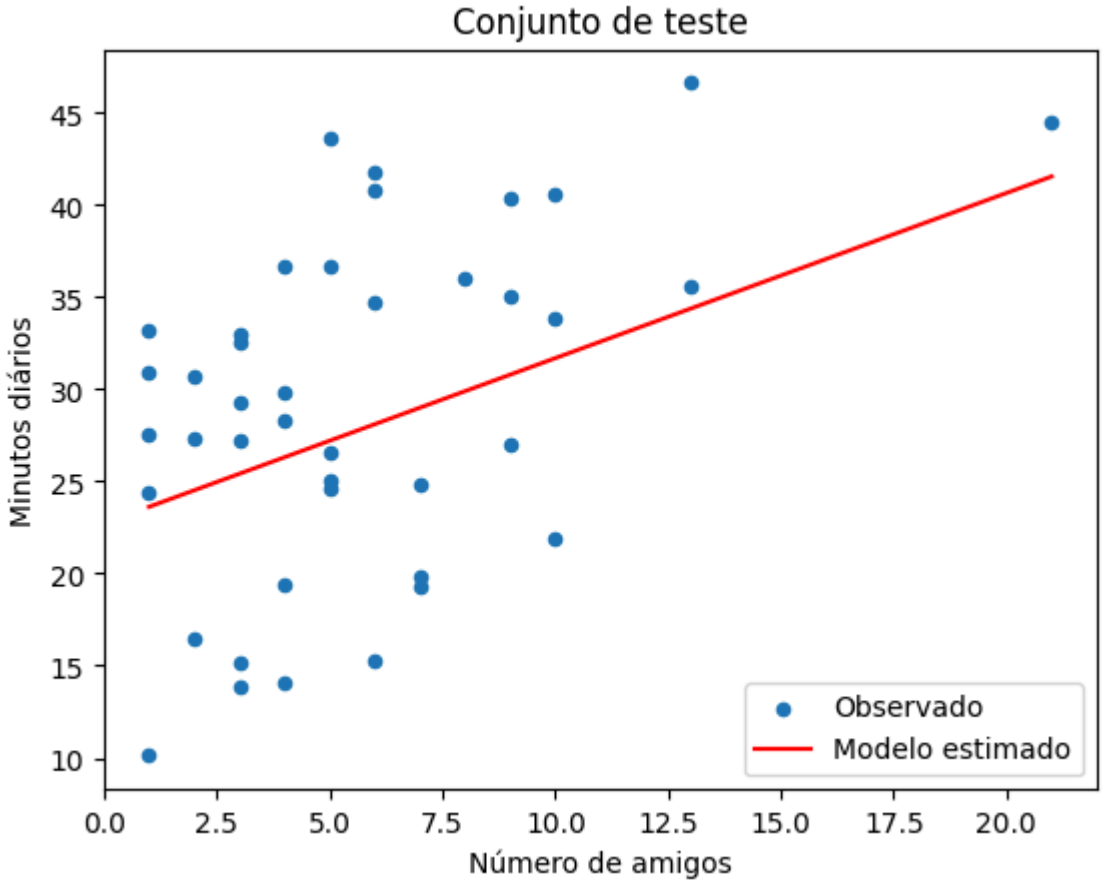
In [ ]:
```
# Grafico de dispersao - conjunto de teste
teste.plot.scatter(x = 'num_amigos', y = 'minutos_diarios', label='Observado')

# Desenhando a linha de previsao
x_prev = np.linspace(teste['num_amigos'].min(),
                     teste['num_amigos'].max(),
                     100).reshape(-1, 1)
y_prev = alfa + beta * x_prev
plt.plot(x_prev, y_prev, color='red', label='Modelo estimado')

# Rotulos e titulo
plt.xlabel("Número de amigos")
plt.ylabel("Minutos diários")
plt.title('Conjunto de teste')

# Legenda
plt.legend()

# Exibindo o grafico
plt.show()
```



A figura acima ilustra o modelo estimado onde a reta representa tanto o modelo quanto suas previsões. Ou seja, podemos estimar o *output* utilizando qualquer valor possível do *input*.

Generalização Múltipla

Imagine que cada *input* x não é mais um único número, mas sim um conjunto de k números, $x_1, \ldots, x_k$. O modelo de regressão múltipla assume que:

$$y = \alpha + \beta_1 x_1 + \ldots + \beta_k x_k + \varepsilon.$$

> Apesar de intuitiva, a generalização acima requer fundamentos de álgebra linear e está fora do escopo deste curso.

Referências Adicionais

A regressão linear é uma técnica fundamental em análise de dados e modelagem estatística, e o `Python` oferece várias bibliotecas poderosas para realizar regressão linear de forma eficiente. Além da funcionalidade básica de ajuste de modelos de regressão, essas bibliotecas também fornecem recursos avançados para análise estatística e visualização dos resultados.

- A biblioteca scikit-learn oferece uma ampla gama de algoritmos de aprendizado de máquina, incluindo o modelo de regressão linear. Com apenas algumas linhas de código, você pode ajustar um modelo de regressão linear aos seus dados e fazer previsões;
- A biblioteca statsmodels é voltada para análise estatística avançada e fornece uma ampla gama de métodos estatísticos, incluindo modelos de regressão linear. Ela permite realizar testes de hipóteses, calcular estatísticas de ajuste e realizar diagnósticos de resíduos, além de fornecer recursos para visualização dos resultados.