## 2. Aprendizado Supervisionado e Regressão Linear

Nesta aula você vai entender:

* O Aprendizado Supervisionado e as tarefas de Regressão e Classificação
* Como aplicar modelos de Regressão Linear Simples e Múltipla na Predição de Quantidades
* Como avaliar a eficiência dos modelos de Regressão Linear

Nesta aula vamos nos deter sobre modelos de regressão linear para a predição de quantidades. Esses modelos são úteis e empregados em uma série de problemas de predição de preços de imóveis, alugueis, demanda de produtos e muitos casos onde predomina ou é de interesse analisar uma tendência linear de evolução dos dados. Por sua simplicidade modelos lineares são talvez o primeiro modelo que experimentamos ao tratar um problema de modelagem de dados para a predição de valores e, apesar de simples, eles se mostram tremendamente úteis na predição dos valores ou no entendimento das relações entre os dados.

O modelo de regressão linear também é útil para entendermos de modo geral o funcionamento dos modelos de aprendizado supervisionado. Ele já traz em si todos os principais elementos que encontramos no aprendizado supervisionado: conjunto de treinamento, aprendizado, resultado probabilístico, métricas de eficiência do modelo etc. Assim, antes de prosseguirmos na aplicação de modelos de regressão vamos retomar e aprender mais sobre alguns conceitos do aprendizado supervisionado e das diferenças entre as tarefas de regressão e classificação.

# Regressão e Classificação

O **Aprendizado Supervisionado** normalmente envolve dois tipos de problemas, problemas de Regressão e problemas de Classificação. Em ambos os casos você possui um conjunto de dados de treinamento com exemplos de entradas e saídas. Nos problemas de **Regressão** os rótulos dos dados (as saídas) são valores contínuos como o preço de imóveis ou a quantidade de clicks em página Web, sendo essas as quantidades que você quer prever para novos valores de entrada (por exemplo, uma data futura). Já nos problemas de **Classificação** os rótulos dos dados são valores discretos como [Yes, No], [True, False], [Safe, Unsafe, Dangerous], [Fraud, Not Fraud] ou ainda [Dogs, Cats, Fishes, Others] em um conjunto de fotos de animais domésticos. Vamos empregar aqui dois exemplos de brinquedo apenas para você se familiarizar com esses modelos e entender a diferença.

## Exemplo de Regressão: Grilos e Temperatura

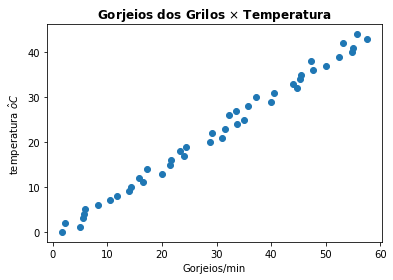
Grilos gorjeiam com mais frequência em dias quentes que em dias freios e profissionais e amadores coletaram dados sobre o gorjeio desses insetos por minuto e as temperaturas chegando a modelo linear que permite, através do número de gorjeios por minuto (a variável ou atributo preditor) estimar a temperatura (o rótulo, ou a variável objetivo). Os dados a seguir simulam esses dados.

from IPython.display import YouTubeVideo  
YouTubeVideo('ssSCGwy\_Qfo')



np.random.seed(1)  
df = pd.DataFrame()  
  
df['temperature'] = np.arange(0,45,1)  
df['chirps'] = (20 \* df['temperature'] - 4) / 16   
df['chirps'] = df['chirps'] + np.round(np.random.sample(len(df))\*5,0)

plt.scatter(df['chirps'],df['temperature'])  
  
plt.title('Gorjeios dos Grilos $\\times$ Temperatura', fontsize=14, weight='bold')  
plt.xlabel('Gorjeios/min')  
plt.ylabel('temperatura $\^{o}C$')  
plt.show()

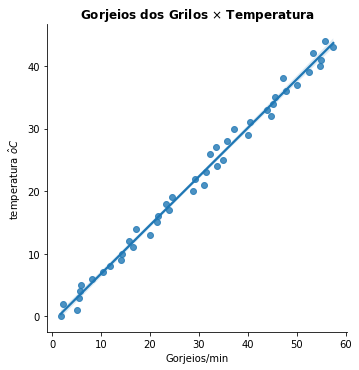


Como você pode ver os dados se aproximam bastante de uma 'reta', um modelo linear, e podemos esperar escrever algo como:

Que é a equação de uma reta.

Modelos de regressão podem ser ainda polinomiais, exponenciais etc., (diferentes classes de modelos de regressão), mas escolhemos um modelo linear para aproximar os dados por quê observando os dados *esperamos* poder aproximá-los por uma reta! Os modelos lineares é a classe de modelos de regressão que escolhemos para aproximar nossos dados.

sns.lmplot(x='chirps',y='temperature', data=df)  
  
plt.title('Gorjeios dos Grilos $\\times$ Temperatura', fontsize=14, weight='bold')  
plt.xlabel('Gorjeios/min')  
plt.ylabel('temperatura $\^{o}C$')  
plt.show()



Um modelo de regressão linear busca encontrar a reta que melhor se ajusta a esse conjunto de dados calculando os melhores coeficientes e que tornam os valores mais próximos dos valores reais.

import statsmodels.formula.api as sm  
  
# Definição do modelo  
lm = sm.ols(formula='temperature ~ chirps', data=df)  
  
# Treinamento  
lm = lm.fit()  
  
# Resultados  
print(lm.summary())

OLS Regression Results   
==============================================================================  
Dep. Variable: temperature R-squared: 0.992  
Model: OLS Adj. R-squared: 0.992  
Method: Least Squares F-statistic: 5334.  
Date: Sat, 05 Mar 2022 Prob (F-statistic): 9.94e-47  
Time: 13:14:08 Log-Likelihood: -70.585  
No. Observations: 45 AIC: 145.2  
Df Residuals: 43 BIC: 148.8  
Df Model: 1   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
Intercept -0.9595 0.361 -2.659 0.011 -1.687 -0.232  
chirps 0.7761 0.011 73.034 0.000 0.755 0.798  
==============================================================================  
Omnibus: 3.593 Durbin-Watson: 2.175  
Prob(Omnibus): 0.166 Jarque-Bera (JB): 1.946  
Skew: 0.230 Prob(JB): 0.378  
Kurtosis: 2.091 Cond. No. 69.2  
==============================================================================

Por hora, note apenas que os resultados do modelo acima mostram os coeficientes obtidos, Intercept e chirps. O Intercept é o coeficiente livre da equação da reta que não está associado a qualquer variável preditora e recebe este nome por ser o valor que 'intercepta' o eixo quando os valores de todas as variáveis preditoras é nulo. chirps é o coeficiente dos gorjeios.

E podemos empregar essa expressão para estimar a temperatura no caso de observarmos 32 gorjeios por minuto,

Gorjeios\_Observados = 32  
Temperatura\_Estimada = -0.9595 + 0.7761 \* Gorjeios\_Observados   
print(f'{Temperatura\_Estimada:.2f} oC')

23.88 oC

Ou, o que é mais comum e em geral mais simples, empregar a função predict que aplica o modelo,

Gorjeios\_Observados = pd.DataFrame({'chirps': [32]})  
Temperatura\_Estimada = lm.predict(Gorjeios\_Observados)[0]  
  
print(f'{Temperatura\_Estimada:.2f} oC')

23.88 oC

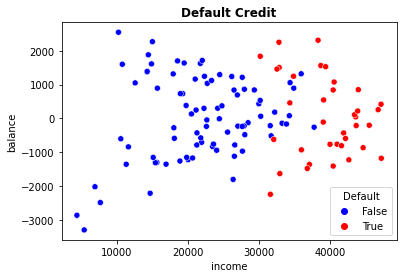
Este exemplo foi inspirado em, <https://developers.google.com/machine-learning/crash-course/descending-into-ml/linear-regression>.

## Exemplo de Classificação: Default Credit

Empresas de crédito estão bastante empenhadas e identificar possíveis 'calotes' (*default*) no pagamento do crédito cedido aos seus clientes e para isso elas buscam prever possíveis não pagamentos a partir do histórico de várias operações de crédito e dados dos seus clientes. Dados do cliente (como renda, idade, estado civil e escolaridade) e do crédito (valor, seu propósito, tempo) são os recursos empregados como variáveis preditoras e default[True,False] o rótulo dos dados que queremos prever para novas concessões de crédito ou créditos em andamento que desejamos avaliar o risco. Os dados abaixo simulam um conjunto de dados desse tipo com dados sobre renda, saldo e o pagamento ou não do crédito.

from sklearn.datasets import make\_moons, make\_circles, make\_classification  
  
X, y = make\_classification(n\_samples=120, class\_sep=0.85, weights=[0.7,0.3],  
 n\_features=2, n\_redundant=0, n\_informative=2, random\_state=0, n\_clusters\_per\_class=2  
)  
  
df = pd.DataFrame()  
df['income'] = ( X[:,0] + 3 ) \* 10000  
df['balance'] = X[:,1] \* 1000  
df['default'] = y  
df['default'] = df['default'].astype('bool')

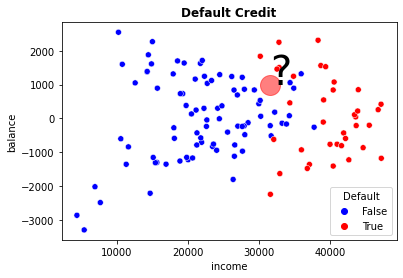
sns.scatterplot(x='income',y='balance',hue='default',data=df,palette=['blue','red'])  
plt.title('Default Credit', fontsize=14, weight='bold')  
plt.xlabel('income')  
plt.ylabel('balance')  
plt.legend(title='Default',loc='lower right')  
plt.show()  
  
plt.show()



Veja que a cor (azul ou vermelho) no gráfico acima adiciona uma dimensão dos dados ao representar o valor default[True,False] dos dados. Apesar de dois eixos temos 3 dimensões dos dados! income, balance e default!

Esse é um modo bastante comum de representarmos problemas de classificação envolvendo duas variáveis preditoras.

sns.scatterplot(x='income',y='balance',hue='default',data=df,palette=['blue','red'])  
plt.plot(31500,1000,marker='o',color='red',markersize=20,alpha=0.5)  
plt.text(31500,1000,'?',fontsize=40)  
plt.title('Default Credit', fontsize=14, weight='bold')  
plt.xlabel('income')  
plt.ylabel('balance')  
plt.legend(title='Default',loc='lower right')  
plt.show()



O propósito do modelo é então classificarmos uma nova operação de crédito, digamos para um cliente com [income, balance] = [31500,1000], como uma transação de sucesso ou não (default[True|False]).

Assim como na Regressão, existem várias classes de modelos de classificação, como Árvores de Decisão, K-Vizinhos mais Próximos etc. que empregam princípios e parâmetros diferentes para a estimar as classes. Mas por hora estamos interessados apenas em observar o resultado das classificações sendo indiferente o método que vamos aplicar. Vamos empregar aqui o modelo de K-Vizinhos mais Próximos, e você verá mais sobre ele nas aulas adiante.

from sklearn.neighbors import KNeighborsClassifier  
  
# Define as entradas e saídas do modelo  
X = df[['income','balance']]  
y = df['default']  
  
# Define o modelo a ser empregado   
clf = KNeighborsClassifier(n\_neighbors=3)  
  
# Treina o modelo  
clf.fit(X, y)

KNeighborsClassifier(n\_neighbors=3)

A forma de uso do modelo é bastante semelhante ao modelo de regressão. Definimos o modelo para em seguida buscar os parâmetros que melhor ajustam o modelo aos dados.

Obtido o melhor modelo podemos fazer a predição de novos casos.

novo\_credito = pd.DataFrame({'income':[31500], 'balance':[1000]})  
  
clf.predict(novo\_credito)

array([False])

E para o crédito a um cliente com [income, balance] = [31500,1000] concluímos que o padrão dos dados sugere que não deverá calote nessa operação e que, portanto, o crédito poderia se concedido.

# Modelos

Esses exemplos de brinquedo, embora simples, com poucos dados e atributos, já permite ter uma boa ideia de como esses modelos de aprendizado supervisionado funcionam e podemos agora prosseguir para entender melhor como eles realmente funcionam e têm em comum com muitos outros casos e modelos de aprendizado supervisionado que vamos estudar.

Diagrama

Descrição gerada automaticamente

**Figura 1. Esquema geral do Aprendizado de Máquina Supervisionado.**

Um conjunto de dados é define uma função que podemos não conhecer (mesmo não existir no sentido de uma fórmula ou expressão como conhecemos). Você pode pensar na função que leva o número de gorjeios dos grilos às temperaturas ou os valores de balanço e renda à classificação de *default* nas operações de crédito. Essa é uma função desconhecida, mas que queremos aproximar encontrando uma função que melhor se aproxima aos dados.

é um modelo que buscamos para os dados e vamos buscar inferir, ou obter essa função a partir de uma amostra dos pares , nosso conjunto de treinamento onde são os rótulos dos dados.

Além dos dados rotulados (modelos Supervisionados), o Aprendizado de Máquina também requer como entrada uma classe de modelos que queremos empregar. Como você viu, existem modelos de Regressão e de classificação, e dentro desses modelos podemos por exemplo, escolher um modelo de regressão linear ou polinomial ou, para classificação, escolher entre um modelo de Árvore de Decisão ou K-Vizinhos mais Próximos. É nossa 'hipótese' de que os dados podem ser aproximados pelo modelo que escolhemos (o que vai poder funcionar ou não, mas veremos mais adiante sobre isso).

O Aprendizado de Máquina vai consistir, então, em buscar os melhores parâmetros para a família de modelos escolhida. Isso, em geral, se resume a buscar parâmetros que buscam minimizar o erro, ou a diferença, entre os valores estimados pelo modelo para e os dados, e há diferentes formas de se medir isso (acima representamos isso por . Quando escolhemos acima um modelo linear Temperatura = a\_0 + a\_1 Gorjeios/min ele corresponde a um número infinito de modelos possíveis já que há infinitos valores para os parâmetros e , e o aprendizado consiste em encontrar os valores de e que melhor aproximam os dados.

Diagrama

Descrição gerada automaticamente

**Figura 2. Esquema geral do Aprendizado de Máquina**

**Supervisionado em poucas linhas de código.**

Isso nos dá a forma geral de como se dá o aprendizado em termos de sua programação:

1. **Definimos as Entradas e Saídas do Modelo**. Seus recursos e rótulos.
2. **Definimos a Classe de Modelos a ser empregada**.
3. **Treinamos o Modelo**. Isto é, buscam-se os parâmetros que melhor ajustam o modelo aos dados.
4. **Prevemos novos casos**. Aplicamos o modelo para novos casos.

# Modelos são Probabilísticos

Diferente de uma fórmula os modelos não são determinísticos. Quando você prevê que a temperatura será 30oC com base na medida de gorjeios dos grilos ou quando prevê que o cliente não dará calote se você conceder crédito à ele isso pode ou não acontecer. Pode ser que a temperatura seja de fato abaixo ou acima e que o cliente, independente da sua predição dê um calote e que você tenha feito um mal negócio. Mas a ideia é que os modelos buscam retornar o valor ou a classe *mais provável* e que, portanto, independente de se mostrar correta ou não a predição o modelo, buscam fornecer respostas que aumentam as suas chances de estar correto. De fato, é famoso o aforismo do estatístico George Box de que *nenhum modelo é correto, mas alguns são úteis*.

Foto em preto e branco de homem com óculos de grau

Descrição gerada automaticamente

**Figura 3. (George Box) "Nenhum modelo é correto, mas alguns são úteis" .**

**Tenha isso sempre em mente ao empregar modelos de Aprendizado de Máquina.**

**(George Box) "Nenhum modelo é correto, mas alguns são úteis" . Tenha isso sempre em mente ao empregar modelos de Aprendizado de Máquina.**

Em consequência disso existem várias abordagens ou princípios diferentes que os modelos podem adotar na buscar de uma resposta mais provável e, desse modo, diferentes modelos podem levar a diferentes respostas e não há um único modelo 'certo'!

from sklearn.neighbors import KNeighborsClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import SVC  
  
X = np.array(df[['income','balance']])  
y = df['default']  
  
models = [KNeighborsClassifier(n\_neighbors=5),  
 SVC(),  
 DecisionTreeClassifier()]  
  
fig, axes = plt.subplots(1, 3, figsize=(20, 4))  
for model, ax in zip(models, axes):  
 print(model)  
 clf = model.fit(X, y)  
 plot\_2d\_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)  
 sns.scatterplot(x=X[:,0],y=X[:,1],hue=y, ax=ax)  
 ax.set\_title(model, fontsize=14, weight='bold')  
 ax.set\_xlabel('income')  
 ax.set\_ylabel('balance')  
  
plt.show()

Tela de computador com fundo branco

Descrição gerada automaticamente

Por exemplo, no nosso exemplo de predição de crédito podemos aplicar modelos de K-Vizinhos Mais Próximos, Support Vector Machines e de Árvores de Decisão. Os gráficos acima exibem a **fronteira de decisão** dos modelos, isto é, mostram as regiões associadas a cada uma das classes previstas pelo modelo. Cada modelo emprega um princípio diferente (você pode pensar uma forma de cálculo diferente) na busca dos resultados mais prováveis. Levam, portanto, à predições diferentes das classes e não há, de modo geral, como afirmar que este ou aquele modelo é o modelo 'correto'. Nenhum é! Existem, entretanto, algumas métricas que podem ajudar na escolha de modelos *melhores* que outros.

## O Fez Classificações Erradas?

Talvez você tenha notado no exemplo acima que os modelos não classificam corretamente todos os dados do conjunto de treinamento. Você pode ver em todos eles pontos 'True' na região 'False' (área azul) e pontos 'False' na região 'True' (área vermelha do gráfico). Isso é algo comum nos modelos e que vamos estudar em detalhe. Por hora você deve lembrar que os modelos são construídos para classificação de *novos casos*. Um dado do conjunto de treinamento (não é um novo caso!) classificado errado apenas indica um caso que, apesar da maior probabilidade de pertencer a uma classe ele *ocorreu* de modo diferente. Você pode pensar estes casos como *outliers* ou exceções, que não seguiram as classes mais prováveis segundo algum critério.

# Regressão Linear

Entendido como funcionam os modelos em geral podemos agora nos concentrar unicamente em como criar e avaliar modelos de Regressão Linear.

Um modelo linear aproxima o valor de variável objetivo a partir de uma combinação linear das variáveis preditoras .

A cada variável preditora corresponde um coeficiente , havendo um coeficiente independente que corresponte ao valor de para (*intercept*). Se temos uma única variável preditora nosso modelo é uma reta e temos um modelo de **Regressão Simples**. Se temos mais dimensões temos um *hiperplano* e o modelo é uma **Regressão Múltipla**.

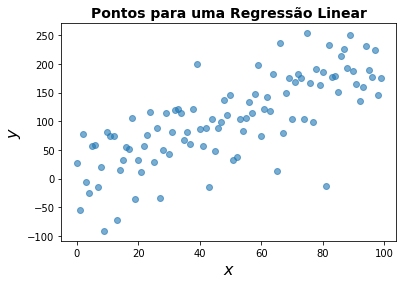
Existem também outros modelos de Regressão. Por exemplo, um modelo de Regressão Polinomial busca aproximar os dados a um polinômio de grau e seria um modelo de Regressão Polinomial Simples de grau 2. Esses são modelos de **regressão não linear**, mas nos deteremos aqui unicamente em modelos lineares.

## Calculando os Coeficientes de uma Regressão Simples

No caso mais simples, nosso problema consiste em dados um conjunto de pontos , determinar os coeficientes da reta que melhor aproxima .

# you can skip this code!  
from scipy.stats import norm  
  
x = np.arange(0, 100)  
y = 2\*x + 3  
y = y + norm.rvs(loc=0, scale=50, size=100, random\_state=1234)

plt.scatter(x,y,alpha=0.6)  
plt.title('Pontos para uma Regressão Linear Simples', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
plt.show()



Dado um conjunto de pontos podemos traçar várias retas que *aproximam* de diferentes modos o conjunto de pontos. A regressão linear simples é definida pela reta que minimiza o erro ou a distância dos pontos dos valores estimados .

Dado um conjunto de pontos queremos buscar a reta que reduz o erro das estimativas de . Esse erro pode ser medido pela distância dos pontos e , ou mais simplesmente pela distância quadrática, e podemos escrever esse erro em função de e , que ainda não conhecemos os valores:

O ponto de mínimo da função de Erro fornece os valores e dos coeficientes da reta,

e pode ser obtido a partir das derivadas com relação a esses parâmetros,

Resolvendo-se esse sistema de equações você obtêm:

onde e são a média dos valores e , e:

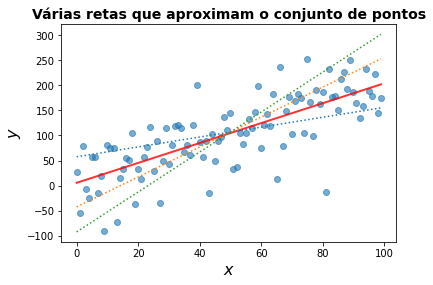
Os valores dessa expressão não devem ser estranhos a você. De fato, eles são a e a e, desse modo, podemos escrever simplesmente:

Empregando o conjunto de pontos acima podemos verificar os coeficientes produzidos:

# A diagonal da matriz contém a covariância entre cada variável e ela mesma  
  
b = np.cov(x,y)[1,0] / np.var(x); print(b)  
a = np.mean(y) - b\*np.mean(x); print(a)

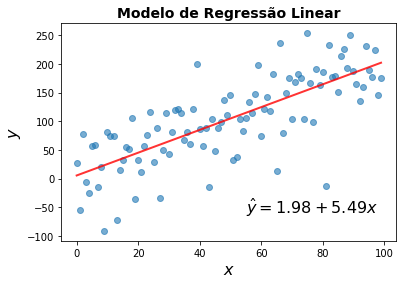
1.9850551363966769  
5.495384904636339

# you can skip this code!  
  
y2 = a + b\*x  
y3 = (y2[51] - 50\*(b-1))+(b-1)\*x  
y4 = (y2[51] - 50\*(b+1))+(b+1)\*x  
y5 = (y2[51] - 50\*(b+2))+(b+2)\*x  
  
plt.scatter(x, y, alpha=0.6)  
plt.plot(x, y2, c='red', lw=2, alpha=0.8)  
plt.plot(x, y3, linestyle='dotted')  
plt.plot(x, y4, linestyle='dotted')  
plt.plot(x, y5, linestyle='dotted')  
  
plt.title('Várias retas que aproximam o conjunto de pontos', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
plt.show()



E, assim, determinamos a reta que melhor estima os valores de :

# you can skip this code!  
  
plt.scatter(x, y, alpha=0.6)  
plt.plot(x, y2, c='red', lw=2, alpha=0.8)  
  
plt.title('Modelo de Regressão Linear', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
plt.text(55,-60,'$ \hat{y} = 1.98 + 5.49 x $', fontsize=16)  
plt.show()



# Regressão Linear Múltipla

No modelo linear mais geral o valor de variável objetivo, ou dependente, é obtido a partir de uma combinação linear de um conjunto de variáveis preditoras, ou dependentes, .

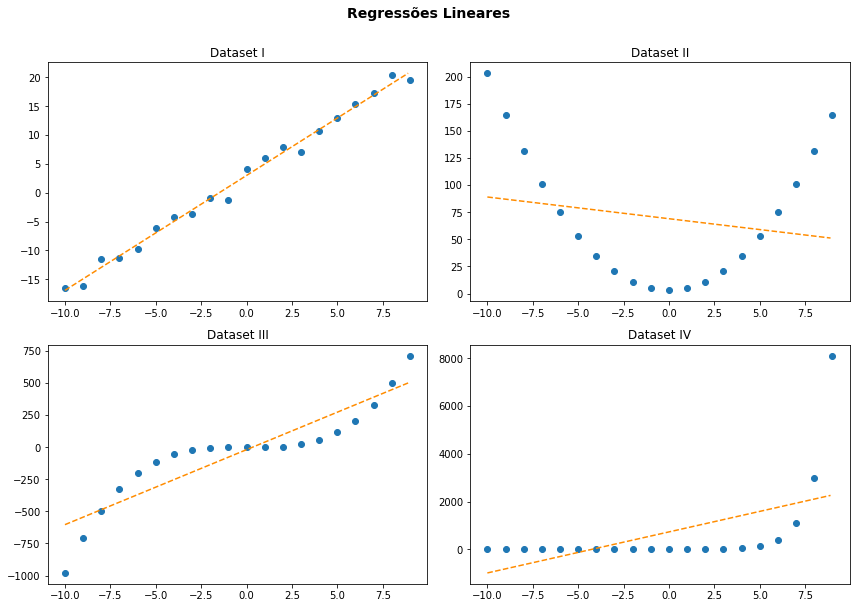
Agora, a cada variável preditora corresponde um coeficiente , havendo um coeficiente independente que corresponte ao valor de para (*intercept*).

Os coeficientes são obtidos do mesmo modo que na regressão simples, minimizando-se o erro entre os valores de nos dados e os valores estimados , e podemos escrever:

Não há, entretanto, nenhuma fórmula ou expressão para obtermos de forma algébrica os valores de que são obitidos, de modo geral, aplicando-se algum método de otimização, como o método de mínimos quadrados, sendo o erro uma função convexa que garante a existência de um único ponto de mínimo.

**Para quaisquer conjuntos de dados sempre é possível se calcular os coeficientes de uma regressão linear. Esses coeficientes fornecem sempre a melhor reta ou *hiperplano* que se ajusta os dados. A pergunta é: seria esse ajuste é suficientemente bom?**

# Avaliando a Eficiência do Modelo: Coeficiente de Determinação,

No caso de uma regressão simples podemos calcular diretamente os coeficientes e empregando apenas os valores médios de e , a e a . Assim, para quaisquer conjuntos de dados podemos *sempre* calcular um modelo de regressão *mesmo que o modelo linear não represente* *exatamente nossos dados*. Isso é ilustrado pelo exemplo a seguir.

Os quatro conjuntos de dados acima foram obtidos aplicando-se as funções:

Gráfico

Descrição gerada automaticamente com confiança média

Embora apenas o conjunto de dados se ajuste de fato a um modelo linear você pode notar que podemos calcular o modelo linear em todos os casos, *mesmo ele não se ajustando aos dados*!

No caso de uma regressão simples você pode observar o ajuste ou não do modelo fazendo uma inspeção visual como acima. Mas o mesmo pode ocorrer no caso de uma regressão múltipla e, neste caso, havendo mais dimensões você não poderá observar o ajuste dos dados ao modelo. Por isso é necessário termos uma métrica, uma medida que avalia a **eficiência do modelo**, ou o quanto os dados se ajustam ao nosso modelo proposto. A métrica mais importante para uma regressão linear é o **Coeficiente de Determinação**, ou ainda .

O **Coeficiente de Determinação** é uma medida no intervalo que indica o quanto um modelo linear explica a variância de um conjunto de dados. Quanto mais próximo de o valor do , mais os dados se ajustam ao modelo linear.

onde

é a *soma dos quadrados residuais* e,

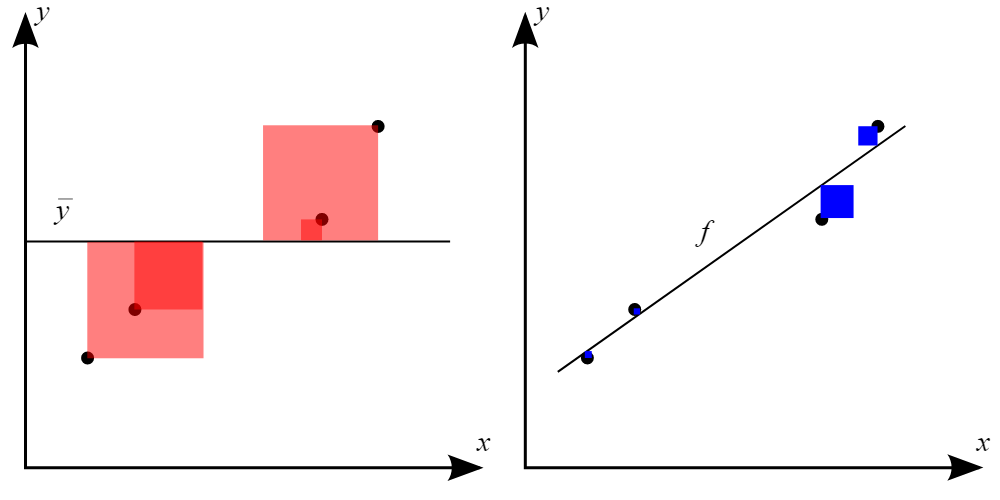
é a *soma total dos quadrados*.

De fato, como você pode observar, somente o primeiro conjunto dos quatro conjuntos de dados acima têm apresentam um coeficiente de determinação próximo de 1.

# you can skip this code!  
  
R2 = {}  
for ds\_type in df.dataset.unique():  
 ds = df[df.dataset == ds\_type]  
  
 z = np.polyfit(ds.x, ds.y, 1)  
 p = np.poly1d(z)  
  
 R2[ds\_type] = 1 - sum((ds.y - p(ds.x))\*\*2) / sum((ds.y - np.mean(y))\*\*2)  
   
for dataset, r2 in R2.items():  
 print('Dataset ' + dataset + ', R-Square = ' + str(np.round(r2,4)))

Dataset I, R-Square = 0.9999  
Dataset II, R-Square = 0.266  
Dataset III, R-Square = 0.8505  
Dataset IV, R-Square = 0.3459

Basicamente o coeficiente de determinação é uma medida de proporção que verifica o quanto variância dos dados está representada no modelo com relação ao modelo trivial .



**Figura 4. À esquerda, em vermelho, as variâncias com relação ao modelo trivial e, à direita, as variâncias do modelo de regressão (em azul). (Fonte: Wikipedia)**

O é útil para avaliarmos a adequação de um modelo de regressão linear aos dados e pode também ser empregado para compararmos modelos que disputam entre si. Por exemplo, você pode ter dois modelos de regressão múltipla para obter o preço de imóveis. Um deles considera um atributo adicional *bairro do imóvel*, o outro não. O coeficiente então pode ser empregado para avaliar qual o melhor modelo.

**Você vai encontrar essas métricas de avaliação de eficiência dos modelos para quaisquer modelos supervisionados e elas podem ser empregadas na busca de um melhor modelo, ajustando parâmetros de um modelo ou na seleção entre vários modelos que competem.**

# Ajustado

O Ajustado é uma medida alternativa para avaliação do modelo e existem ainda outras métricas.

A inclusão de inúmeras variáveis, mesmo com pouco poder explicativo sobre a variável dependente, podem sempre aumentar o valor de . Isso é um incentivo para a inclusão indiscriminada de variáveis, prejudicando o princípio da parcimônia (princípio da *navalha de Ockham*) em que o aumento de complexidade não leva a ganhos proporcionais no modelo. Para evitar isso você pode empregar o coeficiente de determinação ajustado que penaliza a inclusão de preditores pouco explicativos no modelo:

k+1 representa o número de variáveis explicativas mais a constante, e a inclusão de variáveis pouco explicativas passa a penalizar o valor do com o valor ajustado para baixo.

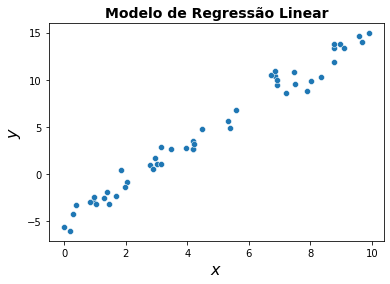
# Modelos de Regressão em Python

Você vai aprender agora como obter modelos de regressão simples e múltipla empregando o pacote statsmodels do Python. Vamos começar com uma regressão simples de valores aleatórios, algo que já fizemos antes, mas agora vamos fazer passo a passo para você se familiarizar com a construção do modelo.

import statsmodels.formula.api as sm

Vamos gerar uma amostra de 50 valores "aleatórios" a partir de uma função linear.

rng = np.random.RandomState(1)  
x = 10 \* rng.rand(50)  
y = 2 \* x - 5 + rng.randn(50)  
sns.scatterplot(x=x, y=y)  
  
plt.title('Modelo de Regressão Linear', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
plt.show()  
  
  
df = pd.DataFrame({'x':x,'y':y})  
df.head()



x y  
0 4.170220 2.653267  
1 7.203245 8.561284  
2 0.001144 -5.668959  
3 3.023326 1.033987  
4 1.467559 -3.182193

## Construindo o modelo linear, sm.ols(formula = , data= )

A função sm.ols() (ordinary least squares, se refere ao método de otimização empregado) requer um conjunto de treinamento e um parâmetro formula (*patsy* fórmula, um formato bastante empregado em modelos) que indica a variável dependente (objetivo) e as variáveis independentes (preditoras) do modelo.

formula = 'y ~ x'

significa

para um modelo

Podemos então declarar o modelo,

# Define o modelo  
model = sm.ols(formula='y ~ x', data=df)

E em seguida fazer o ajuste, *fit*, do modelo aos dados. É o treinamento ou aprendizado do modelo.

# Treinamento  
result = model.fit()

O método summary() apresenta, então, vários resultados do modelo.

print(result.summary())

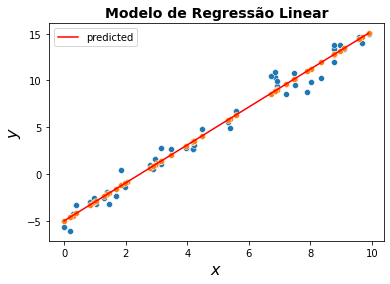
OLS Regression Results   
==============================================================================  
Dep. Variable: y R-squared: 0.979  
Model: OLS Adj. R-squared: 0.979  
Method: Least Squares F-statistic: 2246.  
Date: Sat, 05 Mar 2022 Prob (F-statistic): 5.71e-42  
Time: 18:29:23 Log-Likelihood: -65.935  
No. Observations: 50 AIC: 135.9  
Df Residuals: 48 BIC: 139.7  
Df Model: 1   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
Intercept -4.9986 0.239 -20.948 0.000 -5.478 -4.519  
x 2.0272 0.043 47.397 0.000 1.941 2.113  
==============================================================================  
Omnibus: 0.058 Durbin-Watson: 1.590  
Prob(Omnibus): 0.971 Jarque-Bera (JB): 0.057  
Skew: 0.048 Prob(JB): 0.972  
Kurtosis: 2.865 Cond. No. 10.4  
==============================================================================

Observando os coeficientes no sumário do modelo acima vemos que o modelo de aproximação linear é dado por:

E podemos empregar o modelo para estimar valores de , isto é (predicted), e comparar seus valores.

df['predicted'] = result.predict(df.x)

sns.scatterplot(x='x',y='y',data=df)  
sns.scatterplot(x='x',y='predicted',data=df)  
sns.lineplot(x='x',y='predicted',data=df,color='red', label='predicted')  
  
plt.title('Modelo de Regressão Linear', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
  
plt.legend()  
plt.show()



## Analisando a Eficiência do Modelo

O Sumário acima ainda traz o Coeficiente de Determinação e os *p-values* dos coeficientes. O primeiro você já conhece e é uma medida geral da eficiência do modelo.

Esse é um valor bastante próximo de 1 e que garante termos um modelo que explica bastante bem os dados.

Outra medida importante de se observar são os dos coeficientes. Eles são um teste de hipótese sobre os valores dos coeficientes onde a hipótese nula é de que os coeficientes são não significativos (). A hipótese alternativa é de que o coeficiente é significativo (e, portanto ). Assim, $ p-values < 0.05 $ indicarão que o coeficiente é significativo para o modelo e podemos observar que para os dois coeficientes acima (Intercept e x) os valores são significativos.



Podemos assim dizer que

é um modelo que aproxima bastante bem os dados.

Existem várias outras métricas nos resultados acima, mas para os nossos propósitos a análise acima já é suficiente.

**qual é um bom valor? Novamente não existe um modelo 'certo' e a questão é melhor formulada em termos de: qual é um bom resultado para os seus propósitos? Mas de modo geral valores já são bastante úteis mesmo para predições e com valores abaixo recomendaríamos buscar outros modelos de aproximação dos dados.**

# CASO: Estimando o Preço de Veículos

Vamos empregar agora um conjunto de dados mais interessante e estimar o Preço de veículos com base em suas características. Vamos empregar o seguinte conjunto de dados:

df = pd.read\_csv("https://vincentarelbundock.github.io/Rdatasets/csv/MASS/Cars93.csv",index\_col=0)  
df.head()

Manufacturer Model Type Min.Price Price Max.Price MPG.city \  
1 Acura Integra Small 12.9 15.9 18.8 25   
2 Acura Legend Midsize 29.2 33.9 38.7 18   
3 Audi 90 Compact 25.9 29.1 32.3 20   
4 Audi 100 Midsize 30.8 37.7 44.6 19   
5 BMW 535i Midsize 23.7 30.0 36.2 22 ...  
[5 rows x 27 columns]

## Preparação dos Dados

A **Fase de Preparação dos Dados** difere caso a caso, mas um princípio geral é nesta fase precisamos preparar o dado da melhor forma para que sejam aplicados os modelos. É uma boa prática, senão obrigatório, uma exploração dos dados para que você conheça os dados antes de aplicar os modelos.

Aqui a api statsmodels.formula.api não suporta nome de atributos com '.' e precisamos adequar o nome dos atributos antes de aplicar o modelo.

df.columns = [ x.replace('.','') for x in df.columns ]

df.head()

Manufacturer Model Type MinPrice Price MaxPrice MPGcity \  
1 Acura Integra Small 12.9 15.9 18.8 25   
2 Acura Legend Midsize 29.2 33.9 38.7 18   
3 Audi 90 Compact 25.9 29.1 32.3 20   
4 Audi 100 Midsize 30.8 37.7 44.6 19   
5 BMW 535i Midsize 23.7 30.0 36.2 22 ...   
[5 rows x 28 columns]

A avaliação de valores ausentes é uma boa prática,

df.isnull().sum()

Manufacturer 0  
Model 0  
Type 0  
Min.Price 0  
...

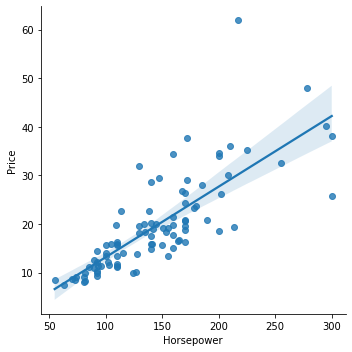
Wheelbase 0  
Width 0  
Turn.circle 0  
Rear.seat.room 2  
Luggage.room 11  
Weight 0  
Origin 0  
Make 0  
dtype: int64

Mas os valores ausentes neste caso não impactam nossos modelos (eles não serão variáveis nem dependente e nem independentes no modelo).

## Regressão Linear Simples

Inicialmente vamos empregar somente variáveis numéricas para estimar o Preço, Price, dos veículos começando por um modelo de regressão linear simples empregando apenas a potência do motor, Horsepower, como variável preditora.

sns.lmplot(x='Horsepower',y='Price',data=df)  
plt.show()



Vamos, portanto, tentar inicialmente determinar o preço a partir somente da potência dos veículos, isto é:

model = sm.ols(formula="Price ~ Horsepower", data=df)

result = model.fit()  
print(result.summary())

OLS Regression Results   
==============================================================================  
Dep. Variable: Price R-squared: 0.621  
Model: OLS Adj. R-squared: 0.617  
Method: Least Squares F-statistic: 149.3  
Date: Sat, 05 Mar 2022 Prob (F-statistic): 6.84e-21  
Time: 18:57:57 Log-Likelihood: -297.23  
No. Observations: 93 AIC: 598.5  
Df Residuals: 91 BIC: 603.5  
Df Model: 1   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
Intercept -1.3988 1.820 -0.769 0.444 -5.014 2.216  
Horsepower 0.1454 0.012 12.218 0.000 0.122 0.169  
==============================================================================  
Omnibus: 55.766 Durbin-Watson: 1.466  
Prob(Omnibus): 0.000 Jarque-Bera (JB): 308.782  
Skew: 1.805 Prob(JB): 8.89e-68  
Kurtosis: 11.164 Cond. No. 449.  
==============================================================================

Obtemos assim o modelo:

## Predição de novos valores, predict(x)

Podemos agora empregar nosso modelo para estimar o consumo desconhecido de um novo veículo na cidade, mas para o qual conhecemos a sua potência:

x = pd.DataFrame({'Horsepower': [150,180]})  
result.predict(x)

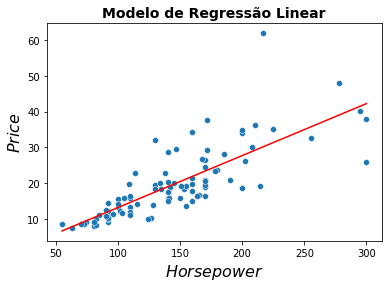
0 20.406915  
1 24.768052  
dtype: float64

## Avaliando o Modelo

Sendo uma regressão simples podemos fazer uma inspeção visual do modelo.

df['predicted'] = result.predict()

sns.scatterplot(x='Horsepower', y='Price', data=df)  
sns.lineplot(x='Horsepower', y='predicted', data=df, color='red')  
  
plt.title('Modelo de Regressão Linear', fontsize=14, weight='bold')  
plt.xlabel('$Horsepower$', fontsize=16)  
plt.ylabel('$Price$', fontsize=16)  
  
plt.show()



Os dados parecem se ajustar muito parcialmente aos dados. Mas a inspeção visual é bastante limitada e, no máximo, pode permitir a avaliação de modelos de regressão simples, em duas dimensões.

### Analisando o e *p-values*

A análise do coeficiente de determinação e do *p-value* dos coeficientes é uma forma mais efetiva de avaliação e pode ser aplicada também a modelos de regressão múltipla (com mais de uma variável preditora). Os valores também mostram que os dados se ajustam de modo bastante parcial ao modelo, que explica somente 62.1% da variação dos dados havendo ainda um coeficiente (*intercept*) não significativo para o modelo. O intervalo de confiança seria ainda uma verificação adicional e fornece o intervalo de valores de cada coeficiente, e que podemos empregar para identificar possíveis desvios nos coeficientes.

## Regressão Múltipla: Adicionando Mais Variáveis ao Modelo

Sendo o ajuste do modelo anterior parcial vamos buscar aprimorar o modelo adicionando mais variáveis preditoras. Vamos, entretanto, ainda nos atermos a entradas numéricas no modelo.

model = sm.ols(formula="Price ~ Passengers + Length + Wheelbase + \  
 Width + Turncircle + Luggageroom + \  
 Weight + Horsepower + EngineSize + \  
 RPM + Wheelbase ", data=df)  
result = model.fit()  
print(result.summary())

OLS Regression Results   
==============================================================================  
Dep. Variable: Price R-squared: 0.727  
Model: OLS Adj. R-squared: 0.689  
Method: Least Squares F-statistic: 18.94  
Date: Sat, 05 Mar 2022 Prob (F-statistic): 2.32e-16  
Time: 19:03:28 Log-Likelihood: -251.04  
No. Observations: 82 AIC: 524.1  
Df Residuals: 71 BIC: 550.6  
Df Model: 10   
Covariance Type: nonrobust   
===============================================================================  
 coef std err t P>|t| [0.025 0.975]  
-------------------------------------------------------------------------------  
Intercept 53.1792 28.749 1.850 0.069 -4.146 110.504  
Passengers -0.3768 1.317 -0.286 0.776 -3.004 2.250  
Length 0.0090 0.130 0.069 0.945 -0.251 0.269  
Wheelbase 0.6436 0.280 2.301 0.024 0.086 1.201  
Width -1.5020 0.457 -3.289 0.002 -2.413 -0.591  
Turncircle -0.5811 0.374 -1.555 0.124 -1.326 0.164  
Luggageroom 0.0801 0.349 0.230 0.819 -0.615 0.776  
Weight 0.0066 0.005 1.366 0.176 -0.003 0.016  
Horsepower 0.1430 0.046 3.123 0.003 0.052 0.234  
EngineSize -0.7457 2.409 -0.310 0.758 -5.549 4.057  
RPM -0.0025 0.002 -1.081 0.283 -0.007 0.002  
==============================================================================  
Omnibus: 28.002 Durbin-Watson: 1.869  
Prob(Omnibus): 0.000 Jarque-Bera (JB): 81.343  
Skew: 1.048 Prob(JB): 2.17e-18  
Kurtosis: 7.406 Cond. No. 2.88e+05  
==============================================================================

A inspeção visual agora não é mais possível mas podemos analisar as métricas do modelo. O modelo agora apresenta um resultado melhor, mas ainda assim parcial.

Uma imagem contendo Texto

Descrição gerada automaticamente

O , ajustado, ainda é bastante inferior a (um valor a partir do qual as predições começam a ficar interessantes) e ainda existem estimadores não sigficantes, incluindo o Intercept.

Um modelo melhor pode ser obtido. Vamos empregar agora somente os estimadores relevantes, e empregaremos o na fórmula para excluir o Intercept do modelo.

model = sm.ols(formula="Price ~ Wheelbase + Width + Horsepower - 1", data=df)  
result = model.fit()  
print(result.summary())

OLS Regression Results   
=======================================================================================  
Dep. Variable: Price R-squared (uncentered): 0.941  
Model: OLS Adj. R-squared (uncentered): 0.939  
Method: Least Squares F-statistic: 476.8  
Date: Sat, 05 Mar 2022 Prob (F-statistic): 4.19e-55  
Time: 19:08:16 Log-Likelihood: -286.90  
No. Observations: 93 AIC: 579.8  
Df Residuals: 90 BIC: 587.4  
Df Model: 3   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
Wheelbase 0.6567 0.140 4.702 0.000 0.379 0.934  
Width -1.0195 0.213 -4.789 0.000 -1.442 -0.597  
Horsepower 0.1526 0.012 12.931 0.000 0.129 0.176  
==============================================================================  
Omnibus: 44.686 Durbin-Watson: 1.663  
Prob(Omnibus): 0.000 Jarque-Bera (JB): 187.154  
Skew: 1.475 Prob(JB): 2.29e-41  
Kurtosis: 9.292 Cond. No. 89.0  
==============================================================================

Conseguimos agora um modelo que parece suficientemente bom e que explica mais de 93% da variação dos preços dos veículos.

Este é um modelo bastante bom e que podemos agora empregar para fazer predições de preços de uma forma mais ou menos segura. Podemos, por exemplo, estimar o preço de um veículo hipotético com as medidas médias de Wheelbase,Width e Horsepower.

x = pd.DataFrame({'Wheelbase': [ df.Wheelbase.mean() ],  
 'Width': [ df.Width.mean() ],  
 'Horsepower': [ df.Horsepower.mean() ]})  
preco\_estimado = result.predict(x)[0]  
  
print(f'Preço estimado (US$ 1000): {preco\_estimado:.2f}')

Preço estimado (US$ 1000): 19.48

# Adicionando Variáveis Categóricas

O pacote statsmodel permite empregar variáveis categóricas diretamente. Como o cálculo dos coeficientes requer atributos numéricos o pacote transforma esses atributos internamente fazendo o *hot encode* dos dados.

model = sm.ols(formula="Price ~ Origin + Wheelbase + Width + Horsepower - 1", data=df)  
result = model.fit()  
print(result.summary())

OLS Regression Results   
==============================================================================  
Dep. Variable: Price R-squared: 0.710  
Model: OLS Adj. R-squared: 0.697  
Method: Least Squares F-statistic: 53.85  
Date: Sat, 20 Nov 2021 Prob (F-statistic): 7.17e-23  
Time: 17:22:57 Log-Likelihood: -284.82  
No. Observations: 93 AIC: 579.6  
Df Residuals: 88 BIC: 592.3  
Df Model: 4   
Covariance Type: nonrobust   
===================================================================================  
 coef std err t P>|t| [0.025 0.975]  
-----------------------------------------------------------------------------------  
Origin[USA] -0.2091 14.536 -0.014 0.989 -29.096 28.678  
Origin[non-USA] 1.9900 13.860 0.144 0.886 -25.555 29.534  
Wheelbase 0.6152 0.140 4.380 0.000 0.336 0.894  
Width -0.9700 0.324 -2.998 0.004 -1.613 -0.327  
Horsepower 0.1530 0.015 10.457 0.000 0.124 0.182  
==============================================================================  
Omnibus: 44.115 Durbin-Watson: 1.660  
Prob(Omnibus): 0.000 Jarque-Bera (JB): 167.993  
Skew: 1.498 Prob(JB): 3.32e-37  
Kurtosis: 8.863 Cond. No. 7.11e+03  
==============================================================================

Por exemplo, ao incluir o atributo categórico Origin, que possui valores USA e non-USA, o pacote cria as variáveis binárias (*hot encode*) Origin[USA] e Origin[non-USA] para serem empregadas no modelo. Aqui o modelo apresentou um resultado pior que o modelo anterior, mas seu objetivo aqui é apenas de mostrar o uso de atributos categóricos em um modelo de regressão.

O uso de variáveis categóricas é muito importante em várias aplicações de modelos de regressão e você pode, por exemplo, pensar na importância de um atributo categórico como *bairro* ou *marca* para a estimativa de preços de imóveis ou veículos.

# Erros Comuns na Interpretação do Modelo

Dois erros são bastante comuns na intepretação de modelos de regressão linear e nos dois casos, podem levar a conclusões bastante errôneas.

## R2 ~ 0 => não há relação entre os dados.

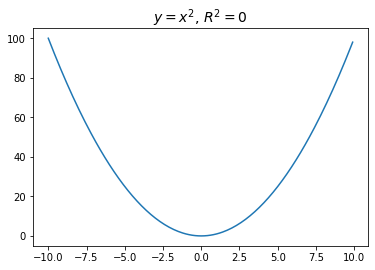
Isso não é necessariamente verdade.

data = pd.DataFrame()  
data['x'] = np.arange(-10,10,0.1)  
data['y'] = data['x']\*\*2  
  
model = sm.ols(formula="y ~ x", data=data)  
result = model.fit()  
print( result.summary() )

OLS Regression Results   
==============================================================================  
Dep. Variable: y R-squared: 0.000  
Model: OLS Adj. R-squared: -0.005  
Method: Least Squares F-statistic: 0.07426  
Date: Sat, 05 Mar 2022 Prob (F-statistic): 0.786  
Time: 19:20:23 Log-Likelihood: -962.77  
No. Observations: 200 AIC: 1930.  
Df Residuals: 198 BIC: 1936.  
Df Model: 1   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
Intercept 33.3300 2.119 15.731 0.000 29.152 37.508  
x -0.1000 0.367 -0.273 0.786 -0.824 0.624  
==============================================================================  
Omnibus: 30.369 Durbin-Watson: 0.001  
Prob(Omnibus): 0.000 Jarque-Bera (JB): 19.728  
Skew: 0.639 Prob(JB): 5.20e-05  
Kurtosis: 2.142 Cond. No. 5.77  
==============================================================================

A regressão acima tem , mas os dados tem uma clara relação como mostra o gráfico.

plt.plot(data['x'],data['y'])  
plt.title('$y = x^2$, $R^2 = 0$',fontsize=14)  
plt.show()



O coeficiente indica somente a presença de relações **lineares** entre os dados podendo haver muitas outras relações que não as relações lineares como mostra o exemplo acima.

## R2 ~ 1 => 1 então é causa e efeito.

Isso não é necessariamente verdade.

Talvez ainda mais comum que o erro anterior e até mais grave é concluírmos a partir de uma regressão linear válida uma relação de **causa-efeito** entre essas variáveis, embora em muitos casos isso possa ser verdade.

A área de um imóvel e sua idade tem, em geral, um efeito sobre o preço do imóvel, e a renda de um país tem um efeito sobre a expectativa de vida das pessoas, o que pode ser observado se implementamos um modelo, linear ou não. Mas a presença de relação linear entre os dados **não garante** que exista uma relação de causa-efeito. Uma relação linear apenas diz que observando os valores das variáveis independentes podemos estimar com alguma confiança os valores da variável dependente. Isso não permite determinar quem é a causa ou quem é o efeito e nem mesmo se existe uma relação de causa-efeito. Ambas, por exemplo, podem ter uma causa comum e, portanto, seus valores podem simplesmente *andarem* juntos e há uma série de casos de correlações espúrias entre os dados.

Gráfico, Gráfico de linhas

Descrição gerada automaticamenteGráfico, Gráfico de linhas

Descrição gerada automaticamente

**Figura 4. Relações espúrias entre dados exibindo uma correlação bastante alta ~ 0.99. Uma interpretação errada poderia levar a entendermos que os gastos com ciência são a causa de um maior número de suicídios (acima) ou que um menor consumo de margarina pode levar a uma redução do número de divórcios (abaixo). Uma nota: não nos deteremos aqui nas diferenças de correlação e coeficiente de determinação, pois não altera o observado aqui e não tem relevância para os nossos propósitos. Fonte:** [**https://www.tylervigen.com/spurious-correlations**](https://www.tylervigen.com/spurious-correlations)

# Sumário da Aula

A partir de modelos simples de regressão e classificação você pôde entender alguns conceitos importantes e comuns a vários modelos de Aprendizado Supervisionado como seu caráter **probabilístico, conjunto de treinamento, variáveis preditoras e objetivo, métricas de eficiência e classes de modelos**.

Você compreendeu também a diferença entre problemas de **regressão**, que estimam quantidades, e problemas de **classificação**, estimam categorias ou classes dos dados. Ambos são modelos supervisionados onde os casos de treinamento são previamente rotulados e sobre o qual construímos modelos para a predição de novos casos.

Você também aprendeu a criar modelos de **Regressão Linear Simples** e **Múltipla** empregando o pacote statsmodels e como avaliar esses modelos observando os valores de (coeficiente de determinação) e os **p-values** dos coeficientes. Com isso você pôde entender como podemos construir e refinar um modelo até obtermos um modelo útil para os nossos propósitos. Ao final você ainda viu alguns **erros comuns** e aos quais você deve estar atento na interpretação desses modelos.

# Para Saber Mais

* Modelos de regressão linear podem ser associados a transformações, como por exemplo a aplicação de , e podemos aplicar um modelo para aproxima Log(y) = a\_0 + a\_1 x\_1 + a\_2 x\_2 + ... no lugar . Isso expande a capacidade do modelo linear e é útil em uma série de problemas práticos como cálculo de preços de imóveis e aluguéis por exemplo. Pesquise sobre isso ou tente reproduzir o modelo de preços de veículos criado aqui obtendo um modelo de .
* Vamos relembrar o que é o *Hot Encode*? Acesse: <https://www.educative.io/blog/one-hot-encoding> e veja também como fazer o *hot encode* com o Pandas ou ainda com scikit-learn. Modelos de regressão requerem valores numéricos e essa transformação é útil quando queremos incluir no nosso modelo variáveis categóricas, como no caso de bairro ou marca do veículo para um modelo de preços de imóveis ou carros.
* Embora tenhamos usado aqui o pacote statsmodel o pacote scikit-learn é mais geral para modelos de aprendizado de máquina e é o que devemos usar daqui para diante na maior parte dos modelos. Mas ele também implementa modelos de regressão linear e você pode consultar o site <https://scikit-learn.org> e entender como implementar esses modelos ou ainda acessar Jake VanderPlas. **Python Data Science Handbook** Disponível em: <https://jakevdp.github.io/PythonDataScienceHandbook/> onde você encontra um capítulo dedicado à regressão linear.
* Acesse também <https://developers.google.com/machine-learning/crash-course/descending-into-ml/linear-regression> e assista a um vídeo de introdução aos modelos de regressão linear na visão da Google.

# Referências

* MÜLLER, Andreas C.; GUIDO, Sarah. **Introduction to machine learning with Python: a guide for data scientists.** O'Reilly Media, Inc., 2016. (parcial online: <https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/ch01.html>).
* Kotu, Vijay; Deshpande, Balachandre **Data Science: concepts and practice**. 2nd ed. Cambridge, [England]: Morgan Kaufmann, c2019. E-book (570 p.) ISBN 9780128147627 (electronic bk.). Disponível em: <http://pergamum.mackenzie.br:8080/pergamumweb/vinculos/00003c/00003cef.jpg>.
* Jake VanderPlas. **Python Data Science Handbook** O'Reilly Media, Inc. (2016). ISBN: 9781491912058. Disponível em: <https://jakevdp.github.io/PythonDataScienceHandbook/>. Acesso: 06 de Novembro de 2021.
* Larose, Chantal D.; Larose, Daniel T. **Data Science Using Python and R** Hoboken: Wiley, c2019. E-book (259 p.) (Wiley Series on Methods and Applications in Data Mining Ser.). ISBN 9781119526834 (electronic bk.). Disponível em: <https://www3.mackenzie.br/biblioteca_virtual/index.php?tipoBiblio=ebookcentral&flashObg=n>
* \_\_\_. **An introduction to machine learning with scikit-learn** Disponível em: <https://scikit-learn.org/stable/tutorial/basic/tutorial.html> Acesso em: 06 de Novembro de 2021.
* \_\_\_. **scikit-learn: machine learning in Python** Disponível em: <http://scipy-lectures.org/packages/scikit-learn/index.html> Acesso em: 06 de Novembro de 2021.
* \_\_\_.**Google, Crash Course - ML Introduction**. Disponível em: <https://developers.google.com/machine-learning/crash-course/ml-intro>. Acesso em: 03 de Março de 2022.