## 2. Regressão e Classificação: Regressão Linear e Logística

Nesta trilha você vai aprender:

* Princípios básicos de Aprendizado Supervisionado com modelos de Regressão e Classificação
* Criar e aplicar modelos de Regressão Linear Simples e Múltipla em Python para predição de valores
* Criar e aplicar um modelo de Regressão Logística para predição de Classes de Dados

Aqui você aprender dois modelos simples de aprendizado de máquina supervisionado mas que trazem vários elementos que são comuns a vários outros modelos. Os modelos de regressão linear e logística fazem parte dos modelos de inferência estatística que dizem respeito a predição de valores e classes de conjuntos de dados, e surgiram muito antes de começarmos a empregar de modo mais geral termos como ciência de dados e aprendizado de máquina para eles. Mas você pode entendê-los como dois modelos típicos de aprendizado supervisionado e entender esses modelos permitirá você compreender os princípios básicos por traz da maior parte dos modelos supervisionados.

# Regressão Linear

Em estatística, para a predição de valores, os métodos os mais empregados são os chamados métodos de regressão e vamos nos deter aqui aos métodos de regressão linear, tanto simples como múltipla, aprendendo como fazer a estimativa de valores a partir de uma aproximação de linear e como avaliar a qualidade desses modelos.

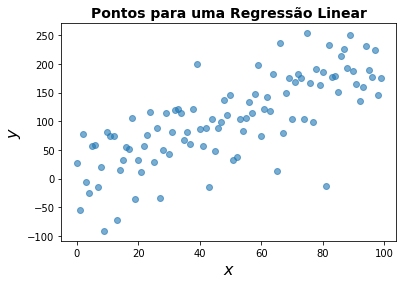
Modelos de regressão são modelos estatísticos para predição de uma variável dependente como uma função de uma ou mais variáveis independentes . Esses modelos podem ser divididos em modelos de **regressão simples** e **múltipla**, conforme empreguem uma única variável preditora (simples) ou várias variáveis preditoras (múltipla). Ainda podemos encontrar modelos de **regressão linear** e **não linear**. Enquanto modelos lineares aproximam a função por uma reta (regressão simples) ou um *hiperplano* (regressão múltipla)as regressões não lineares podem ser de vários tipos, como polinomial e exponencial. Aqui nos limitaremos a criar modelos de regressão linear simples e múltiplos, sendo modelos já bastante poderosos para predição.

## Calculando os Coeficientes de uma Regressão Simples

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

# you can skip this code!  
  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline  
import seaborn as sns  
from scipy.stats import norm  
  
x = np.arange(0, 100)  
y = 2\*x + 3  
y1 = y + norm.rvs(loc=0, scale=50, size=100, random\_state=1234)

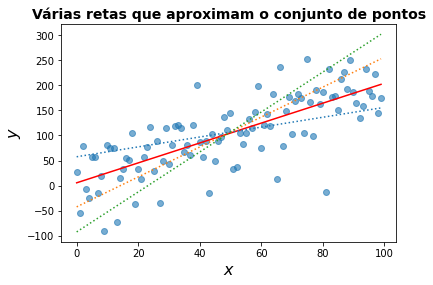
plt.scatter(x,y1,alpha=0.6)  
plt.title('Pontos para uma Regressão Linear', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
plt.show()



# A diagonal da matriz contém a covariância entre cada variável e ela mesma  
b = np.cov(x,y1)[1,0] / np.var(x); print(b)  
a = np.mean(y1) - 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, y1, alpha=0.6)  
plt.plot(x, y2, 'r')  
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()



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 .

De modo geral, 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 (valores e dos coeficientes da reta) da função de Erro pode ser então obtido a partir das derivadas.

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:

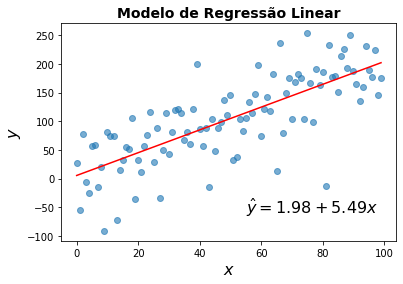
Empregando o conjunto de pontos acima podemos verificar os coeficientes produzidos:

# A diagonal da matriz contém a covariância entre cada variável e ela mesma  
b = np.cov(x,y1)[1,0] / np.var(x); print(b)  
a = np.mean(y1) - b\*np.mean(x); print(a)

1.9850551363966769  
5.495384904636339

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

# you can skip this code!  
  
plt.scatter(x, y1, alpha=0.6)  
plt.plot(x, y2, 'r')  
  
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

Um modelo linear mais geral aproxima o valor de variável objetivo, ou dependente, por uma combinação linear de um conjunto de variáveis preditoras, ou dependentes, .

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 os valores reais de do conjunto de dados e os valores estimados , e podemos escrever:

E podemos obter os valores de para o ponto de mínimo aplicando algum método de otimização, como método de mínimos quadrados, e sendo uma função convexa garante-se a existência de um único ponto de mínimo.

# Avaliando o Modelo: Coeficiente de Determinação,

No caso de uma única variável o cálculo dos pontos de mínimo pode ser obtido diretamente e não é necessário aplicarmos qualquer método de otimização. Como você viu, podemos calcular diretamente os coeficientes e apenas empregando os valores médios de e , a e a . Assim, para quaisquer conjunto de dados podemos *sempre* calcular um modelo de regressão *mesmo que o modelo linear não represente exatamente os dados*. Isso é ilustrado pelo exemplo a seguir.

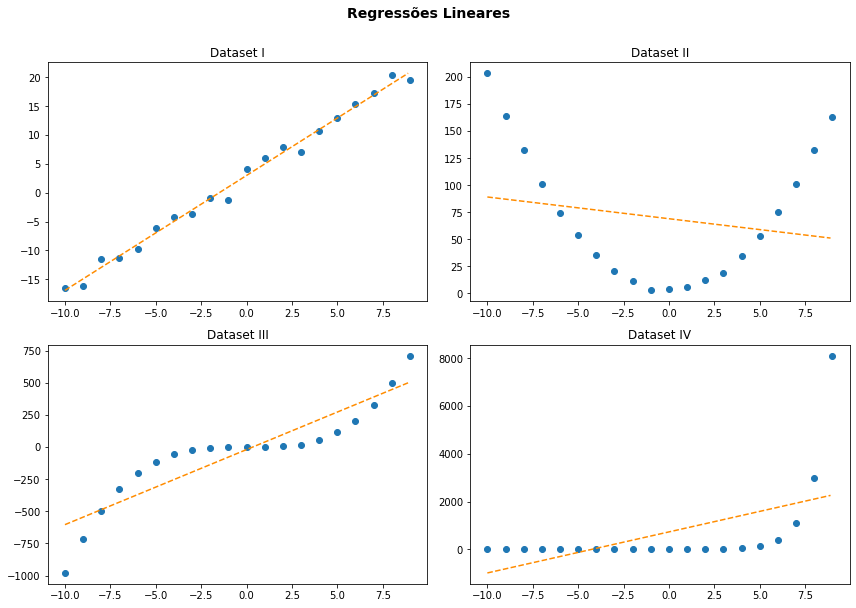
# you can skip this code!  
  
x = np.arange(-10,10,1)  
ruido = norm.rvs(loc=0, scale=1, size=len(x), random\_state=1234)  
df = pd.DataFrame({'dataset':'I','x':x,'y': 2\*x + 3 + ruido})  
df = pd.concat( [df, pd.DataFrame({'dataset':'II','x':x,'y': 2\*x\*\*2 + 3 + ruido}) ])  
df = pd.concat( [df, pd.DataFrame({'dataset':'III','x':x,'y': x\*\*3 - 2\*x + ruido}) ])  
df = pd.concat( [df, pd.DataFrame({'dataset':'IV','x':x,'y': np.exp(1)\*\*x + ruido}) ])

df

dataset x y  
0 I -10 -16.528565  
1 I -9 -16.190976  
2 I -8 -11.567293  
3 I -7 -11.312652  
4 I -6 -9.720589  
.. ... .. ...  
15 IV 5 148.415277  
16 IV 6 403.834247  
17 IV 7 1096.922250  
18 IV 8 2982.279145  
19 IV 9 8101.537022  
  
[80 rows x 3 columns]

# you can skip this code!  
  
fig, ax = plt.subplots(2,2,figsize=(12,8))  
  
fig.suptitle('Regressões Lineares', fontsize=14, weight='bold', y=1.05)  
  
i = 0  
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)  
 print('Modelo de Regressão, Dataset:', ds\_type, p)  
  
 ax[i//2,i%2].scatter(ds.x,ds.y)  
  
 x = np.arange(ds.x.min(), ds.x.max(), 0.1)  
 ax[i//2,i%2].plot(x,p(x),color='darkorange', linestyle='dashed')  
  
 ax[i//2,i%2].set\_title('Dataset ' + ds\_type)  
  
 i += 1  
  
plt.tight\_layout()  
plt.show()

Modelo de Regressão, Dataset: I   
1.991 x + 2.984  
Modelo de Regressão, Dataset: II   
-2.009 x + 68.98  
Modelo de Regressão, Dataset: III   
58.39 x - 19.82  
Modelo de Regressão, Dataset: IV   
171.9 x + 726.9



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

Texto, Carta

Descrição gerada automaticamente

acrescidas de uma parcela de erro . Embora apenas o conjunto de dados se ajuste de fato a um modelo linear você pôde calcular o modelo linear de todos os conjuntos, *mesmo ele não se ajustando aos dados*.

Neste caso você pode observar o ajuste pela simples inspeção visual, mas o caso será pior nos casos de regressão múltipla em que você não pode observar graficamente o ajuste dos dados ao modelo. É neste ponto que é útil termos uma métrica para avaliarmos o quanto os dados se ajustam ao modelo proposto, uma vez que sempre poderemos obtê-lo. A métrica mais importante para isso é *coeficiente de determinação* ou .

O Coeficiente de Determinação, ou ainda Square, é uma medida de 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 de dados tem 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 = 0.2481  
Dataset III, R-Square = 0.85  
Dataset IV, R-Square = 0.3463

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

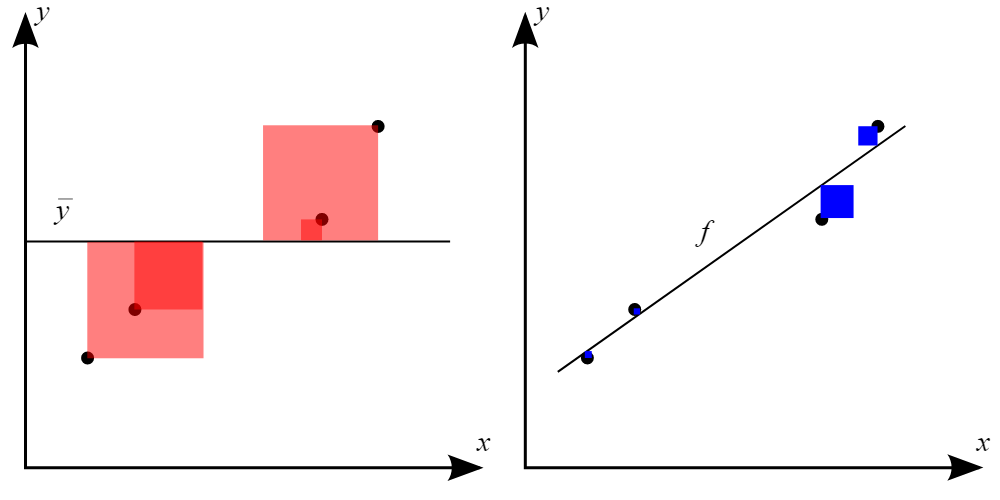


Figura 1. À esquerda, em vermelho, as variâncias com relação ao modelo trivial e, à direita, as variâncias do modelo de regressão (em azul). (Fonte: Wikipedia)

*O é útil não só avaliarmos a adequação de um modelo aos dados, mas pode ser também empregado para compararmos diferentes modelos entre si. Por exemplo, você pode ter dois modelos de regressão múltipla para obter o preço de imóveis. Um considera um atributo adicional bairro do imóvel, outro não. O coeficiente então pode ser empregado para avaliar qual o melhor modelo. Você encontra esse uso de métricas de avaliação dos modelos para quaisquer modelos supervisionados e elas são empregadas para a seleção de diferentes modelos ou para ajustes de parâmetros de um mesmo modelo.*

# Ajustado

O Ajustado é uma medida alternativa para avaliação do modelo. A inclusão de inúmeras variáveis, mesmo com pouco poder explicativo sobre a variável dependente, podem sempre aumentar o valor de . Isto incentiva a inclusão indiscriminada de variáveis, prejudicando o princípio da parcimônia (princípio da *navalha de Ockhan*) em que o aumento de complexidade não leva a um ganho correspondente do modelo. Para evitar isso você pode então empregar o coeficiente de determinação ajustado que penaliza a inclusão de preditores pouco explicativos:

$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

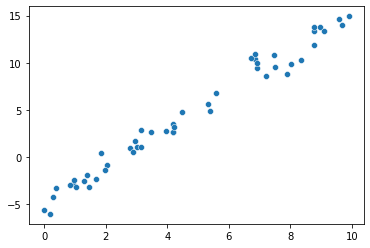
Você vai aprender agora como obter modelos de regressão simples e múltipla empregando o pacote statsmodels do Python. Vamos começar com uma regressão simples de valores aleatórios apenas para você se familiarizar com a construção do modelo.

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline  
import seaborn as sns  
import statsmodels.formula.api as sm

/usr/local/lib/python3.7/dist-packages/statsmodels/tools/\_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.  
 import pandas.util.testing as tm

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.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() requer um conjunto de dados e o parâmetro formula (*patsy* fórmula, um formato bastante empregado em modelos) indicará as variáveis objetivo e preditoras a serem empregadas.

formula = 'y ~ x'

significa

para um modelo

model = sm.ols(formula='y ~ x', data=df)

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

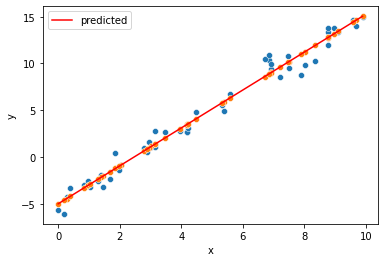
OLS Regression Results   
==============================================================================  
Dep. Variable: y R-squared: 0.979  
Model: OLS Adj. R-squared: 0.979  
Method: Least Squares F-statistic: 2246.  
Date: Sat, 20 Nov 2021 Prob (F-statistic): 5.71e-42  
Time: 17:22:55 Log-Likelihood: -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  
==============================================================================  
  
Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

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.legend()  
plt.show()



## Analisando o modelo obtido

Observando o Coeficiente de Determinação e os *p-values* dos coeficientes,

Podemos ver que $ y = -4.9986 + 2.0272 x $ é um modelo que aproxima bastante bem os 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 ... Weight Origin Make  
1 Acura Integra Small ... 2705 non-USA Acura Integra  
2 Acura Legend Midsize ... 3560 non-USA Acura Legend  
3 Audi 90 Compact ... 3375 non-USA Audi 90  
4 Audi 100 Midsize ... 3405 non-USA Audi 100  
5 BMW 535i Midsize ... 3640 non-USA BMW 535i  
  
[5 rows x 27 columns]

## Preparação dos Dados

A preparação de dados na maior parte dos casos difere caso a caso, mas um princípio geral é de que precisamos preparar o dado da melhor forma para sejam aplicados os modelos.

Neste caso a api statsmodels.formula.api não suporta atributos nomeados com '.' e precisamos, então, adequar o nomde dos atributos antes de aplicar o modelo.

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

df.head()

Manufacturer Model Type ... Weight Origin Make  
1 Acura Integra Small ... 2705 non-USA Acura Integra  
2 Acura Legend Midsize ... 3560 non-USA Acura Legend  
3 Audi 90 Compact ... 3375 non-USA Audi 90  
4 Audi 100 Midsize ... 3405 non-USA Audi 100  
5 BMW 535i Midsize ... 3640 non-USA BMW 535i  
  
[5 rows x 27 columns]

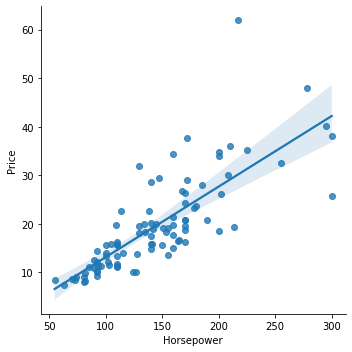
df.describe(include='all')

Manufacturer Model Type ... Weight Origin Make  
count 93 93 93 ... 93.000000 93 93  
unique 32 93 6 ... NaN 2 93  
top Ford Spirit Midsize ... NaN USA Chevrolet Corvette  
freq 8 1 22 ... NaN 48 1  
mean NaN NaN NaN ... 3072.903226 NaN NaN  
std NaN NaN NaN ... 589.896510 NaN NaN  
min NaN NaN NaN ... 1695.000000 NaN NaN  
25% NaN NaN NaN ... 2620.000000 NaN NaN  
50% NaN NaN NaN ... 3040.000000 NaN NaN  
75% NaN NaN NaN ... 3525.000000 NaN NaN  
max NaN NaN NaN ... 4105.000000 NaN NaN  
  
[11 rows x 27 columns]

## 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, 20 Nov 2021 Prob (F-statistic): 6.84e-21  
Time: 17:22:57 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.  
==============================================================================  
  
Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

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 que sabemos o seu peso:

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

0 20.406915  
1 24.768052  
dtype: float64

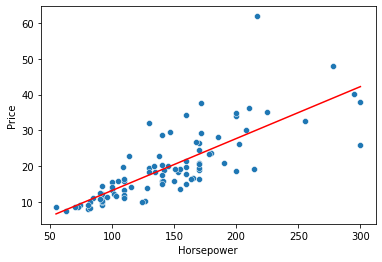
## Avaliando o Modelo

### Inspeção Visual

Sendo uma regressão simples podemos fazer uma inspeção visual do modelo e ver se de fato os dados se ajustam ao modelo.

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

sns.scatterplot(x='Horsepower', y='Price', data=df)  
sns.lineplot(x='Horsepower', y='predicted', data=df, color='red')  
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). O 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.

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, 20 Nov 2021 Prob (F-statistic): 6.84e-21  
Time: 17:22:57 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.  
==============================================================================  
  
Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

## Regressão Múltipla: Adicionando Mais Variáveis ao Modelo

Sendo o ajuste do modelo anterior parcial vamos buscar aprimorar o mdeolo 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, 20 Nov 2021 Prob (F-statistic): 2.32e-16  
Time: 17:22:57 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  
==============================================================================  
  
Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  
[2] The condition number is large, 2.88e+05. This might indicate that there are  
strong multicollinearity or other numerical problems.

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

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, então, empregando-se somente os estimadores relevante, e empregamos 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, 20 Nov 2021 Prob (F-statistic): 4.19e-55  
Time: 17:22:57 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  
==============================================================================  
  
Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

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

Este é assim o modelo que deveríamos empregar e podemos fazer agora uma predição mais segura do preço dos veículos com base nessas características. Por exemplo, podemos 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() ]})  
result.predict(x)

0 19.48232  
dtype: float64

# 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  
==============================================================================  
  
Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  
[2] The condition number is large, 7.11e+03. This might indicate that there are  
strong multicollinearity or other numerical problems.

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.

# Modelo de Regressão o Aprendizado Supervisionado

O modelo de Regressão Linear que você aprendeu aqio já carrega os elementos essenciais de qualquer Aprendizado de Máquina Supervisionado e, por isso, vale a pena revisarmos o que aprendeu até aqui.

1. **Conjunto de Treinamento**. O conjunto de dados corresponde ao conjunto de treinamento em que os dados são rotulados com valores para cada conjunto de dados .
2. **Aprendizado**. O aprendizado consiste e obter os coeficientes do modelo linear que minimizam o erro entre os valores do conjunto de treinamento e os valores obtidos pelo modelo. A otimização de uma função de custo é um conceito chave no aprendizado de máquina e você poderá encontrar essa mesma ideia até em modelos sofisticados de redes neurais profundas para o ajuste dos *pesos* e o aprendizado da rede.
3. **Busca de um Padrão**. A regressão linear produz um padrão que aproxima o conjunto de dados e a partir do qual podemos fazer inferências úteis (reveja aqui a definição de Ciência de Dados da trilha anterior!)
4. **Avaliação do Modelo**. Outro conceito chave está em definirmos uma métrica para avaliação do modelo (aqui, o coeficiente de determinação ). Outros modelos empregarão outras métricas, mas a ideia é a mesma, uma métrica que avalia o quanto o modelo de fato representa o padrão dos dados. Isso é útil para compararmos modelos concorrentes, como por exemplo, no caso da regressão, dois modelos de regressão que empregam diferentes variáveis preditoras.

# Regressão Logística

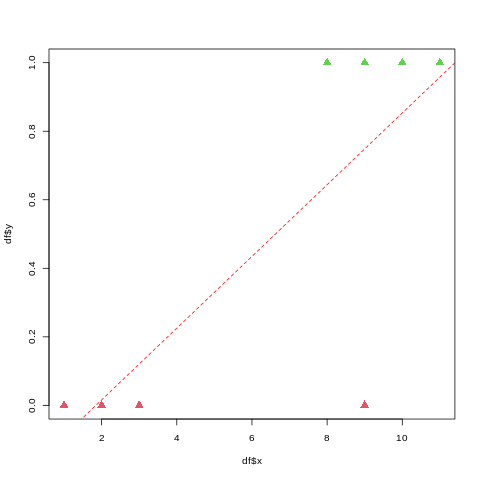
Antes de começarmos a falar de Regressão Logística um ponto de atenção: você não deve confundir modelos de regressão em geral, lineares ou não, que preveem valores com a Regressão Logística que é um modelo de Classificação. A Regressão Logística modela as probabilidades para problemas de classificação binários, com dois resultados possíveis, como *yes/no*, *true/false*, *fraude/não fraude*, *spam/não spam* ou *0/1*, e pode ser entendido como uma extensão dos modelos de regressão linear para problemas de classificação.

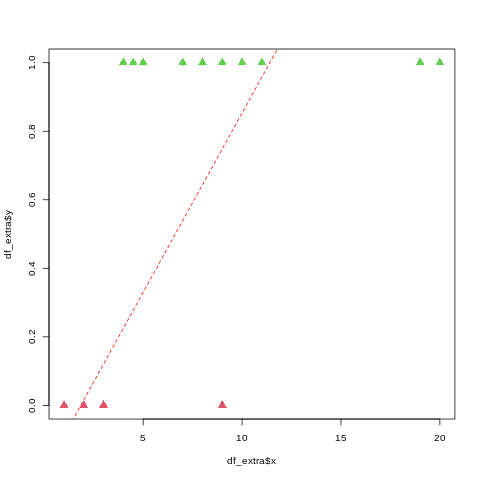
O modelo de regressão linear pode funcionar bem para regressão, mas falha na maior parte dos casos quando é aplicado para classificação. Imagine, você poderia rotular uma das classes com 0 e a outra com 1 e usar a regressão linear. Tecnicamente isso funciona e certamente você obterá os coeficientes da regressão linear, afinal para quaisquer conjuntos de pontos podemos calcular os coeficientes de um modelo linear. Mas para classificação essa abordagem apresenta muitos problemas. Como seu resultado não é uma probabilidade, mas a interpolação linear dos pontos, não há um limite significativo no qual você possa distinguir uma classe da outra.

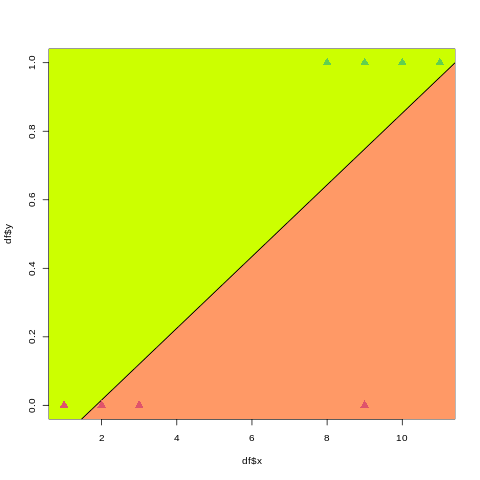
Um modelo simples pode mostrar a ineficiência da regressão linear para a classificação. Veja a abaixo o uso de regressão linear para a classificação dos pontos verdes e vermelhos abaixo.

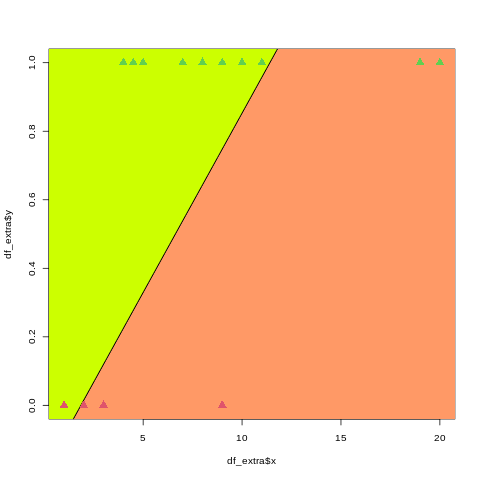
%load\_ext rpy2.ipython

# you can skip this code!  
  
%%R  
# adaptado de https://christophm.github.io/interpretable-ml-book/logistic.html  
# e https://stats.stackexchange.com/questions/22381/why-not-approach-classification-through-regression  
  
df = data.frame(x = c(1,2,3,8,9,10,11,9),  
 y = c(0,0,0,1,1,1,1, 0))  
  
df\_extra = data.frame(x=c(df$x, 7, 7, 7, 20, 19, 5, 5, 4, 4.5),  
 y=c(df$y, 1,1,1,1, 1, 1, 1, 1, 1))  
  
# par(mfrow = c(1, 3))  
layout.matrix <- matrix(c(1, 1, 1, 0), nrow = 2, ncol = 2)  
  
layout(mat = layout.matrix,  
 heights = c(2, 2), # Heights of the two rows  
 widths = c(2, 2)) # Widths of the two columns  
  
plot(df$x, df$y, col=df$y+2, pch=17)  
fit = lm(y ~ x,data=df)  
abline(coefficients(fit),col='red',lty=2)  
points(df$x, df$y, col=df$y+2, pch=17, cex=1.5)  
  
plot(df\_extra$x, df\_extra$y, col=df\_extra$y+2)  
fit = lm(y ~ x,data=df)  
abline(coefficients(fit),col='red',lty=2)  
points(df\_extra$x, df\_extra$y, col=df\_extra$y+2, pch=17, cex=1.5)  
  
plot(df$x, df$y, col=df$y+2, pch=17)  
fit = lm(y ~ x,data=df)  
abline(coefficients(fit),col='red',lty=2)  
polygon(c(0,12,12),c(predict(fit,data.frame(x = c(0,12))),-0.1), col='#FF9966')  
polygon(c(0,12,0), c(predict(fit,data.frame(x = c(0,12))),1.1), col='#CCFF00')  
points(df$x, df$y, col=df$y+2, pch=17, cex=1.5)  
  
plot(df\_extra$x, df\_extra$y, col=df\_extra$y+2)  
fit = lm(y ~ x,data=df)  
abline(coefficients(fit),col='red',lty=2)  
polygon(c(0,21,21),c(predict(fit,data.frame(x = c(0,21))),-0.1), col='#FF9966')  
polygon(c(0,21,0), c(predict(fit,data.frame(x = c(0,21))),1.1), col='#CCFF00')  
points(df\_extra$x, df\_extra$y, col=df\_extra$y+2, pch=17, cex=1.5)









O primeiro conjunto de pontos é linearmente separável e a regressão linear permite classificar os dados com apenas um erro de classificação (o ponto vermelho de menor valor ). Mas à medida que inserirmos novos dados, o modelo passa a apresentar falhas cada vez maiores.

## Modelo logístico

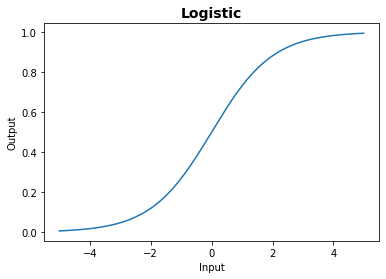
Para resolver esse problema, a regressão logística modela o que é conhecido como log de probabilidades:

Os resultados da regressão logística vão então informar sobre as chances ('Odds') e razão das chances ('Odds Ratio') e não exatamente probabilidades (por exemplo, em alguns casos os valores podem não ter soma 1), mas podemos entender de qualquer modo que uma *chance* reflete em uma probabilidade maior daquele evento ocorrer. Se um evento tem probabilidade , suas chances são , e é por isso que a esquerda lado é chamado de "odds logarítmicas" ou "logit", e podemos obter a probabilidade de chances invertendo a função acima:

Essa função é conhecida como função logística ou sigmóide, , é definida como:

Ela tem o seguinte gráfico:

x = np.arange(-5, 5, 0.01)  
y = 1 / (1 + np.exp(-x))  
  
plt.plot(x,y)  
plt.title('Logistic', fontsize=14, weight='bold')  
plt.xlabel('Input')  
plt.ylabel('Output')  
  
plt.show()



e desempenha um papel importante em outros modelos, como os modelos neurais. Note que os valores dessa função variam de 0 a 1 e é portanto, possível empregar essa função como uma medida de probabilidade, ou das chances.

Esse procedimento funciona muito melhor para classificação que a regressão linear e podemos usar 0.5 como o valor limite de probabilidades entre as classes, isto é, valores para uma classe e valores para outra.

## Estimando os parâmetros

Estimar os valores dos coeficientes . pode seguir a mesma estratégia empregada para calcular os coeficientes de uma regressão múltipla e empregamos um otimizador, como o método de mínimos quadrados.

Do mesmo modo que na regressão linear você deseja minimizar o erro ou maximizar os *acertos* do modelo. Para cada amostra que pertence à classe '1', você irá querer que fique mais próximo de enquanto, para cada amostra que não pertence à classe '1', você quer que seja o mais próximo de possível. As diferenças consistem no erro que queremos minimizar e podemos então, a partir de valores iniciais de dos coeficientes empregar um método de otimização como o de mínimos quadrados, ou um método gradiente, para obter os coeficientes que fornecem o menor erro.

## Exemplo

Você pode pensar em um sistema de detecção de fraude de transações de cartão de crédito com base no histórico de ocorrências. Valores da operação e uso do cartão nas últimas 24h são duas variáveis conhecidas como preditoras de fraude de cartões. Para isso, depois de estimados os coeficientes com base nas amostras conhecidas, tanto de operações de fraude como de operações normais de crédito, você pode aplicar dados de uma nova operação (Valor,Trans24h) no modelo logístico para estimar a chance da operação ser ou não uma fraude:

## Probabilidades e Classificação Binária

O modelo de regressão logística não é apenas um modelo de classificação, mas também fornece as *chances* de cada valor. Esta é uma grande vantagem em relação a modelos que fornecem apenas a classificação final, pois faz uma grande diferença saber se a *chance* daquela classe é 99% ou 51%. Para usar o nosso exemplo, pense em uma transação de cartão classificada como fraude, ser uma fraude com 99% de chance parece algo bem mais crítico que uma fraude com 51% de chance!

A regressão logística ainda é um modelo de **classificação binária**. Mas ele pode ser estendido para classificação multiclasse. O procedimento é simples e consiste em criarmos vários classificadores. Se você tem 3 classes, 'Y', 'N', 'O', pode então ter um classificador logístico 'Y' e 'não Y', outro 'N' e 'não Y', ainda 'O' e 'não O' e, do mesmo modo, buscar o maior valor de p entre esses classificadores. Em geral os pacotes de Software fazem uma implementação multiclasse da classificação logística e, portanto, fica transparente para você a regressão logística ser um procedimento de classificação binária.

*Um cuidado a ser tomado no uso da regressão logística é que ela pode sofrer de separação completa. Isso ocorre quando uma variável preditora separa perfeitamente as duas classes e, neste caso, o modelo de regressão logística não pode ser treinado pois o coeficiente desse atributo seria infinito e convergiria. Observar isso é importante, mas em termos práticos você não precisaria de um modelo se tivesse uma regra simples, como uma única variável para separar as duas classes, e existem técnicas de penalização dos pesos ou definição de uma distribuição de probabilidade dos pesos para minimizar esse problema.*

# CASO: Estimando o tipo de Transmissão dos Veículos

Você vai empregar agora o mesmo conjunto de dados anterior para prever a classe de transmissão dos veículos, isto é, se os veículos tem ou não transmissão manual. Para isso vamos empregar aqui o pacote scikit-learn, um pacote que será empregado para muitos outros modelos de aprendizado de máquina.

df.head()

Manufacturer Model Type ... Origin Make predicted  
1 Acura Integra Small ... non-USA Acura Integra 18.953203  
2 Acura Legend Midsize ... non-USA Acura Legend 27.675476  
3 Audi 90 Compact ... non-USA Audi 90 23.605082  
4 Audi 100 Midsize ... non-USA Audi 100 23.605082  
5 BMW 535i Midsize ... non-USA BMW 535i 28.838446  
  
[5 rows x 28 columns]

df.Mantransavail.value\_counts()

Yes 61  
No 32  
Name: Mantransavail, dtype: int64

## Preparação dos Dados

Mantransavail é um atributo binário e portanto podemos aplicar a regressão logística para obter um modelo de classificação. Mas para a regressão logística com o scikit-lean precisamos ter os valores de classe como ou , e vamos medir a probabilidade do valor se *verdadeiro*, isto é, . Vamos, portanto, criar um novo atributo em que o valor vai designar os veículos manuais e os veículos de transmissão exclusivamente automática.

df['Manual'] = df['Mantransavail'].apply(lambda x: 1 if 'Yes' in x else 0)

df.head()

Manufacturer Model Type ... Make predicted Manual  
1 Acura Integra Small ... Acura Integra 18.953203 1  
2 Acura Legend Midsize ... Acura Legend 27.675476 1  
3 Audi 90 Compact ... Audi 90 23.605082 1  
4 Audi 100 Midsize ... Audi 100 23.605082 1  
5 BMW 535i Midsize ... BMW 535i 28.838446 1  
  
[5 rows x 29 columns]

df.Manual.value\_counts()

1 61  
0 32  
Name: Manual, dtype: int64

Vamos então definir algumas variáveis independentes e a variável dependente do treinamento, lembrando que você deve empregar variáveis numéricas no modelo logístico.

df.columns

Index(['Manufacturer', 'Model', 'Type', 'MinPrice', 'Price', 'MaxPrice',  
 'MPGcity', 'MPGhighway', 'AirBags', 'DriveTrain', 'Cylinders',  
 'EngineSize', 'Horsepower', 'RPM', 'Revpermile', 'Mantransavail',  
 'Fueltankcapacity', 'Passengers', 'Length', 'Wheelbase', 'Width',  
 'Turncircle', 'Rearseatroom', 'Luggageroom', 'Weight', 'Origin', 'Make',  
 'predicted', 'Manual'],  
 dtype='object')

X\_train = df[['EngineSize', 'Horsepower', 'RPM', 'Price', 'Weight']]  
y\_train = df['Manual']

from sklearn.linear\_model import LogisticRegression # para configurar o modelo..   
  
# criando o modelo  
logreg = LogisticRegression()  
  
# treinando o modelo  
logreg.fit(X\_train,y\_train)

LogisticRegression()

O treinamento do modelo fornece então os seguintes coeficientes:

logreg.coef\_

array([[-0.84623734, 0.00713032, 0.00141883, -0.01854519, -0.00161397]])

## Avaliando o Modelo

Existem muitas métricas e formas de avaliação e você vai estudar mais sobre isso adiante. Por hora vamos no deter apenas em verificar o número de *acertos* do nosso preditor.

y\_pred = logreg.predict(X\_train)  
  
df['Manual\_predict'] = y\_pred  
  
df[['Manual','Manual\_predict']]

Manual Manual\_predict  
1 1 1  
2 1 1  
3 1 1  
4 1 1  
5 1 1  
.. ... ...  
89 1 0  
90 1 1  
91 1 1  
92 1 1  
93 1 1  
  
[93 rows x 2 columns]

Podemos fazer isso como acima, fazendo a predição dos valores do nosso conjunto e comparando com os valores *reais*, e você pode ainda obter o percentual de acertos do modelo:

sum( df['Manual'] == df['Manual\_predict'] ) / len(df)

0.8279569892473119

Nada mal, não é mesmo? Esse é o *score* do modelo ou a acuracidade, o percentual de acertos do nosso modelo, e pode ser também obtido como:

logreg.score(X\_train, y\_train)

0.8279569892473119

## Empregando Atributos Categóricos

Do mesmo modo que na regressão linear pode haver preditores categóricos importantes que queremos incluir no nosso modelo. O scikit-learn, entretanto, não fornece um *hot encode* implícito dos atributos categóricos como no pacote *statsmodels*. Você precisará, portanto, fazer isso de modo explícito.

### Preparando os Dados

Vamos incluir aqui o atributo Type que, esperamos, tenha alguma influência sobre o tipo de transmissão disponível nos veículos. Precisamos, desse modo, fazer o *hot encode* desse atributo.

df.Type.value\_counts()

Midsize 22  
Small 21  
Compact 16  
Sporty 14  
Large 11  
Van 9  
Name: Type, dtype: int64

df = pd.get\_dummies(df, columns=['Type'])  
df.head()

Manufacturer Model MinPrice ... Type\_Small Type\_Sporty Type\_Van  
1 Acura Integra 12.9 ... 1 0 0  
2 Acura Legend 29.2 ... 0 0 0  
3 Audi 90 25.9 ... 0 0 0  
4 Audi 100 30.8 ... 0 0 0  
5 BMW 535i 23.7 ... 0 0 0  
  
[5 rows x 35 columns]

df.columns

Index(['Manufacturer', 'Model', 'MinPrice', 'Price', 'MaxPrice', 'MPGcity',  
 'MPGhighway', 'AirBags', 'DriveTrain', 'Cylinders', 'EngineSize',  
 'Horsepower', 'RPM', 'Revpermile', 'Mantransavail', 'Fueltankcapacity',  
 'Passengers', 'Length', 'Wheelbase', 'Width', 'Turncircle',  
 'Rearseatroom', 'Luggageroom', 'Weight', 'Origin', 'Make', 'predicted',  
 'Manual', 'Manual\_predict', 'Type\_Compact', 'Type\_Large',  
 'Type\_Midsize', 'Type\_Small', 'Type\_Sporty', 'Type\_Van'],  
 dtype='object')

X\_train = df[['EngineSize', 'Horsepower', 'RPM', 'Price', 'Weight',  
 'Type\_Compact', 'Type\_Large', 'Type\_Midsize',   
 'Type\_Small', 'Type\_Sporty', 'Type\_Van']]  
y\_train = df['Manual']

from sklearn.linear\_model import LogisticRegression # para configurar o modelo..   
  
# criando o modelo  
logreg = LogisticRegression(max\_iter=1000) # aumentamos o número de iterações para obter a convergência do modelo  
  
# treinando o modelo  
logreg.fit(X\_train,y\_train)

LogisticRegression(max\_iter=1000)

Agora você já sabe como obter a acuracidade (o *score*) do modelo diretamente, e podemos fazer:

logreg.score(X\_train, y\_train)

0.8602150537634409

Um aumento considerável no percentual de acerto do nosso modelo! Essa é métrica de acuracidade é bom resultado e um bom começo para você entender as métricas de um modelo de classificação, mas é uma medida extremamente simples e mesmo modelos ruins podem ter um acuracidade alta em alguns casos. Outras métricas e formas de avaliação mais elaboradas em modelos de classificação serão vistas mais adiante.

# Síntese

Nesta trilha aprendeu alguns princípios comuns e importantes dos modelos de Aprendizado Supervisionado como conjunto de treinamento, variáveis preditoras e objetivo e métricas de avaliação dos modelos. Você deve ter compreendido também a diferença de solução de problemas regressão, que estimam valores, e de classificação, que fazem predição de categorias dos dados. De qualquer modo, ambos são modelos supervisionados onde os casos de treinamento são previamente rotulados com as saídas (variável objetivo) através da qual construímos o modelo para a predição de novos casos.

Você também aprendeu a criar modelos de Regressão Linear Simples e Múltipla e de Regressão Logística em Python, empregando os pacotes statsmodels e scikit-learn.O scikit-learn é um pacote que implementa a maior parte dos modelos que empregaremos à seguir e você mais aprender ainda muito mais sobre ele.

# 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.
* Entenda melhor por que modelo de regressão linear não são adequados para classificação, acesse Molnar Christoph, **Interpretable Machine Learning: A Guide for Making Black Box Models Explainable** em: <https://christophm.github.io/interpretable-ml-book/> cap. 5.2 Logistic Regression.
* Aprenda mais sobre a criação de modelos com o pacote scikit-learn em **An introduction to machine learning with scikit-learn** Disponível em: <https://scikit-learn.org/stable/tutorial/basic/tutorial.html>. Ou você ainda pode preferir acessar Jake VanderPlas. **Python Data Science Handbook** Disponível em: <https://jakevdp.github.io/PythonDataScienceHandbook/>, no capítulo *Introducing SciKit-Learn*, <https://jakevdp.github.io/PythonDataScienceHandbook/05.02-introducing-scikit-learn.html>.

# Referências

Molnar Christoph, **Interpretable Machine Learning: A Guide for Making Black Box Models Explainable** Disponível em: <https://christophm.github.io/interpretable-ml-book/> Acesso em: 06 de Novembro de 2021.

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>.

Oliveira, Rogério de. **Probabilidade e Estatística Aplicada com R** Disponível em: <https://github.com/Rogerio-mack/Probabilidade-Estatistica-Aplicada-R> Acesso em: 06 de Novembro de 2021.

Manapat, Michael. **Introduction to Machine Learning with Python**. Em An Introduction to Machine Learning. EMag Edição 50 (Abr 2017)

Navarro, Danielle, **Learning Statistics with R** disponível em: <https://learningstatisticswithr.com/book/> Acesso: 06 de Novembro de 2021.

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.