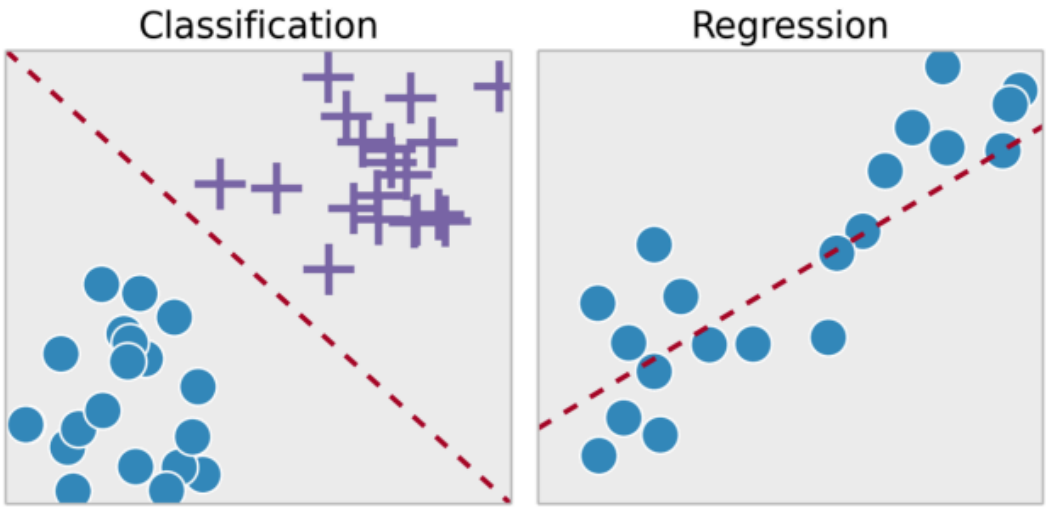
1.Regressão Linear

Modelagem da relação entre variáveis numéricas (variável dependente y e variáveis explanatórias x)



Relações matemáticas entre variáveis

No exemplo abaixo vamos visualizar como o tamanho de um imóvel em m² influencia o valor de venda do mesmo imóvel.

Tamanho (m²)	Preço
30	57.000
39	69.000
49	77.000
60	90.000

Existem três fatores matemáticos que podem determinar essa relação.

Covariância

Indica a relação linear entre duas variáveis

$$C(x,y) = \frac{\sum(xi - \bar{x}) * (yi - \bar{y})}{n - 1}$$

Onde o resultado:

- Maior que 0, variáveis se movem juntas
- Menor que zero 0, variáveis se movem em direções opostas
- Igual 0, variáveis são independentes

Vamos calcular com base no exemplo:

Tamanho (m^2)	Preço	$x_i - \bar{x}$	$y_i - \bar{y}$	$(xi - \bar{x}) * (y_i - \bar{y})$
30	57.000	-14,5	-16.250	235.625
39	69.000	-5,5	-4.250	23.375
49	77.000	4,5	3.750	16.875
60	90.000	15,5	16.750	259.625
44,5 (média) 12,92 (dp)	73.250 (média) 13.865,42 (dp)			**535.500** (soma)

$$C(x,y) = \frac{535.500}{3} = 178500{,}00$$

Assim percebemos que quanto maior o tamanho maior será o valor do imóvel.

Coeficiente de Correlação

Padroniza a covariância com valores entre -1 e 1. Quanto mais próximo dos extremos maior a relação entre as variáveis.

$$Cr(x,y) = \frac{Cov(x,y)}{Std(x) * Std(y)}$$

$$Cr(x,y) = \frac{178500{,}00}{12{,}92 * 13865{,}42} = 0{,}99$$

Coeficiente de Determinação

Mede a proporção da variância em uma variável que pode ser explicada pela outra, sendo o quadrado do coeficiente de correlação.

$$Cd(x,y) = Cr^2$$
$$Cd(x,y) = 0{,}99^2$$
$$Cd(x,y) = 0{,}98$$

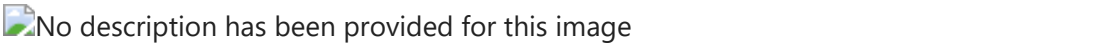
Dessa forma vemos que no exemplo 98% da variável dependente pode ser explicada pela outra.

Regressão Linear Simples

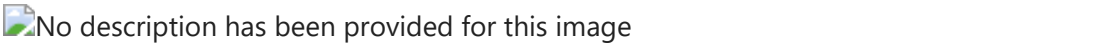
Trata-se de um modelo de regrassão que trabalha com uma única variável explanatória.

Exemplo relação linear entre atributos:

Podemos observar que quanto maior a idade do cliente, maior o custo do plano de saúde, conforme o gráfico de dispersão abaixo.



Assim podemos determinar que existe uma linearidade entre as duas variáveis. A partir disso, pode-se definir uma linha de regressão e fazer previsões de valores que não estão relacionados nos dados atuais.



Essa linha é construida a partir de dois fatores.

Intersecção

O ponto de encontro da linha no eixo Y, onde X = 0.

Inclinação

Fator que determina a inclinação da linha onde a cada unidade que aumenta a variável Independente (x), a variável de resposta (y) sobe o valor da inclinação.

Por fim podemos determinar a previsão de novos valores dado a fórmula:

P = b + m * v

Onde:

P = previsão

b (Constante) = Intersecção

m (Coeficiente) = Inclinação

v = Valor a ser previsto em X

A função de um algoritmo de regressão é encontrar os valores de B0 e B1 que tragam os melhores resultados.

O fluxo matemático para chegarmos na previsão se dá da seguinte forma:

Vamos calcular novos valores para o exemplo anterior, conforme já vimos a correlação se da por **Cr = Cov(X,Y)/Std(X) x Std(Y)**. Assim:

Cr = 11869,71 / 15,985 * 751,62

Cr = 0,9879

A **Inclinação** se da pela fórmula:

m = Cr(Std(Y)/Std(X))

Assim temos:

m = 0,9879 (751,62/15,985)

m = 46,45

Já a **Intersecção** :

$$b = \bar{y} - m\bar{x}$$

$$\bar{y}: média\ de\ y$$
$$\bar{x}: média\ de\ x$$
$$m: Inclinação\ (46{,}45)$$

b = 2104,182 – 46,45 * 42,09

b = 149,0577

Com esses valores conseguimos fazer novas previsões. Por exemplo para v = 54 anos:

P = = 149,0577 + 46,45 * 54

P = 2657,35

Vamos testar isso na prática!

Sempre iniciamos importando as bibliotecas

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

A base plano de saúde possui os dados que vimos anteriormente.

In [4]:
```
base_plano = pd.read_csv('plano_saude.csv')
base_plano
```

Out[4]:

	idade	custo
0	18	871
1	23	1100
2	25	1393
3	33	1654
4	34	1915
5	43	2100
6	48	2356
7	51	2698
8	58	2959
9	63	3000
10	67	3100

Vamos criar uma regressão manualmente apenas aplicando todos os cálculos

In [5]:
```
def PrevisaoRegressaoLinear(x,y,v):
    x = np.array(x)
    y = np.array(y)
```

```
    cov_xy = np.corrcoef(x, y)[0,1]
    inclinacao = cov_xy*(np.std(y)/np.std(x))
    interseccao = np.mean(y) - inclinacao * np.mean(x)
    previsao = interseccao + inclinacao * v
    return float(previsao)
```

In [6]:
```
x = base_plano['idade']
y = base_plano['custo']

lr = PrevisaoRegressaoLinear(x,y,54)
lr
```

Out[6]:
```
2657.3594760672704
```

Utilizando sklearn

O exemplo anterior mostra como podemos fazer previsões apenas utilizando cálculos no entanto em python existem bibliotecas com algoritmos prontos que podem fazer o mesmo.

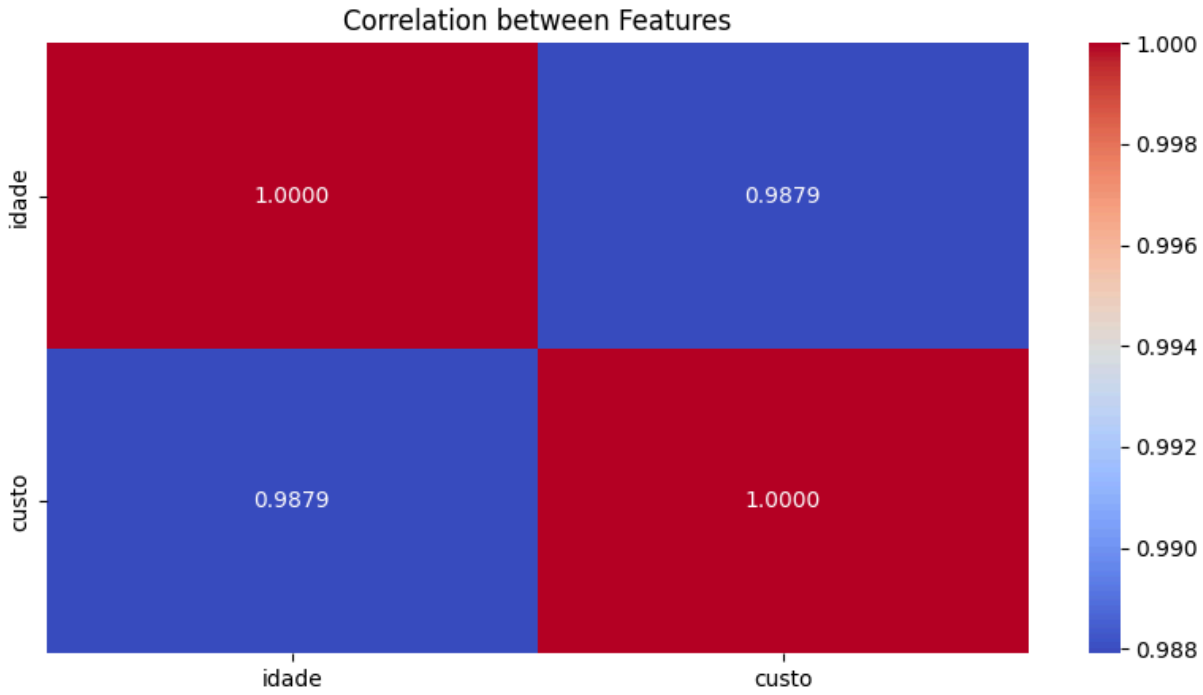
Para calcularmos o coeficiente de correlação utilizamos a função **corrcoef** da biblioteca numpy.

In [7]:
```
np.corrcoef(x, y)
```

Out[7]:
```
array([[1.        , 0.98790395],
       [0.98790395, 1.        ]])
```

Conseguimos essa mesma correlação entre uma ou mais features de um dataframe utilizando a função corr da biblioteca pandas, para melhor visualização utilizamos um mapa de calor da biblioteca seaborn

In [8]:
```
plt.figure(figsize=(10,5))
sns.heatmap(base_plano.corr(), annot=True, cmap='coolwarm', fmt=".4f")
plt.title('Correlation between Features')
plt.show()
```



Para aplicarmos o algoritmo de regressão precisamos tranformar o X em um array bidimensional, mesmo que tenha apenas uma característica. Então vamos ajustar o formato da variável.

In [9]:
```
x = x.values.reshape(-1,1)
x.shape
```

Out[9]:
```
(11, 1)
```

Agora com a biblioteca sklearn vamos importar a classe LinearRegression e instanciar numa variável um objeto dessa classe na sequência utilizamos a função **fit** para treinarmos o algoritmo com os dados reais, nesse treinamento ele irá encontrar os melhores valores para a intersecção e inclinação da linha de regressão.

In [10]:
```
from sklearn.linear_model import LinearRegression
regressor_plano_saude = LinearRegression()
regressor_plano_saude.fit(x, y)
```

Out[10]:
```
LinearRegression
▶ Parameters
```

Com a função abaixo podemos visualizar o valor de intersecção encontrado.

In [11]:
```
regressor_plano_saude.intercept_
```

Out[11]:
```
np.float64(149.05772962483752)
```

E na sequência encontramos a inclinação da linha.

In [12]:
```
regressor_plano_saude.coef_
```

Out[12]:
```
array([46.45003234])
```

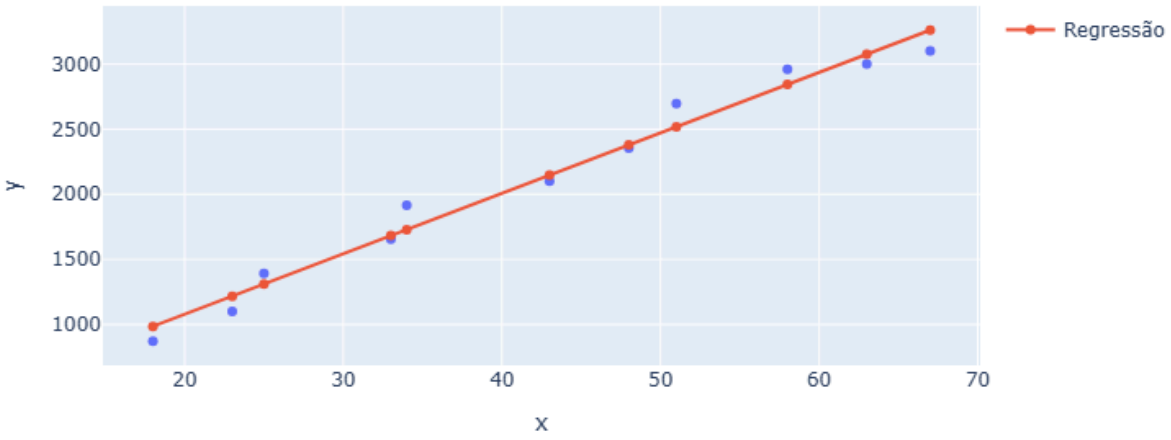
Agora fazermos as previsões de Y com base nos valores de X

In [13]:
```
previsoes = regressor_plano_saude.predict(x)
previsoes
```

Out[13]:
```
array([ 985.15831177, 1217.40847348, 1310.30853816, 1681.9087969 ,
       1728.35882924, 2146.40912031, 2378.65928202, 2518.00937904,
       2843.15960543, 3075.40976714, 3261.20989651])
```

Para compararmos os valores reais com os previstos retornamos podemos utilizar um gráfico de dispersão da biblioteta plotly.

In [14]:
```
grafico = px.scatter(x = x.ravel(), y = y)
grafico.add_scatter(x = x.ravel(), y = previsoes, name = 'Regressão')
grafico.show()
```



Por fim com o método **predict** para prevermos novos valores.

In [15]:
```
previsao_54_anos = regressor_plano_saude.predict([[54]])

print(f"A previsão para 54 anos é: {previsao_54_anos[0]:.2f}")
```

```
A previsão para 54 anos é: 2657.36
```

Para verificarmos quão bom foi o algoritmo nessa aplicação utilizamos a função **score**.

In [17]:
```
regressor_plano_saude.score(x.reshape(-1,1), y)
```

Out[17]:
```
0.9759542217686598
```

Regressão Linear Múltipla

Na regressão múltipla o algoritmo funciona da mesma forma porém considerando mais de uma variável exploratória para determinar valores da variável dependente. Nessa regessão o cálculo é diferente unicamente por considerar um coeficiente para cada característica do dataset conforme a fórmula abaixo.

P = b + m1 * v1 + m2 * v2 + ... + mn * vn

Vamos testar com um novo dataset referente aos valores de imóveis.

In [18]:
```
base_casas = pd.read_csv('house_prices.csv')
base_casas
```

Out[18]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080
...	...	...	...	...	...	...	...
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076

21613 rows × 21 columns

Vamos estudar as características do dataset.

In [19]:
```
#a versão atual do Pandas não está considerando o formato do dado como data, vamos
base_casas.drop('date', axis=1, inplace = True)
base_casas
```

Out[19]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
0	7129300520	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	180000.0	2	1.00	770	10000	1.0	
3	2487200875	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	510000.0	3	2.00	1680	8080	1.0	
...	...	...	...	...	...	...	...	
21608	263000018	360000.0	3	2.50	1530	1131	3.0	
21609	6600060120	400000.0	4	2.50	2310	5813	2.0	
21610	1523300141	402101.0	2	0.75	1020	1350	2.0	
21611	291310100	400000.0	3	2.50	1600	2388	2.0	
21612	1523300157	325000.0	2	0.75	1020	1076	2.0	

21613 rows × 20 columns

In [20]:
```
base_casas.describe()
```

Out[20]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lo
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+0
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+0
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+0
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+0
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+0
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+0
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+0
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+0

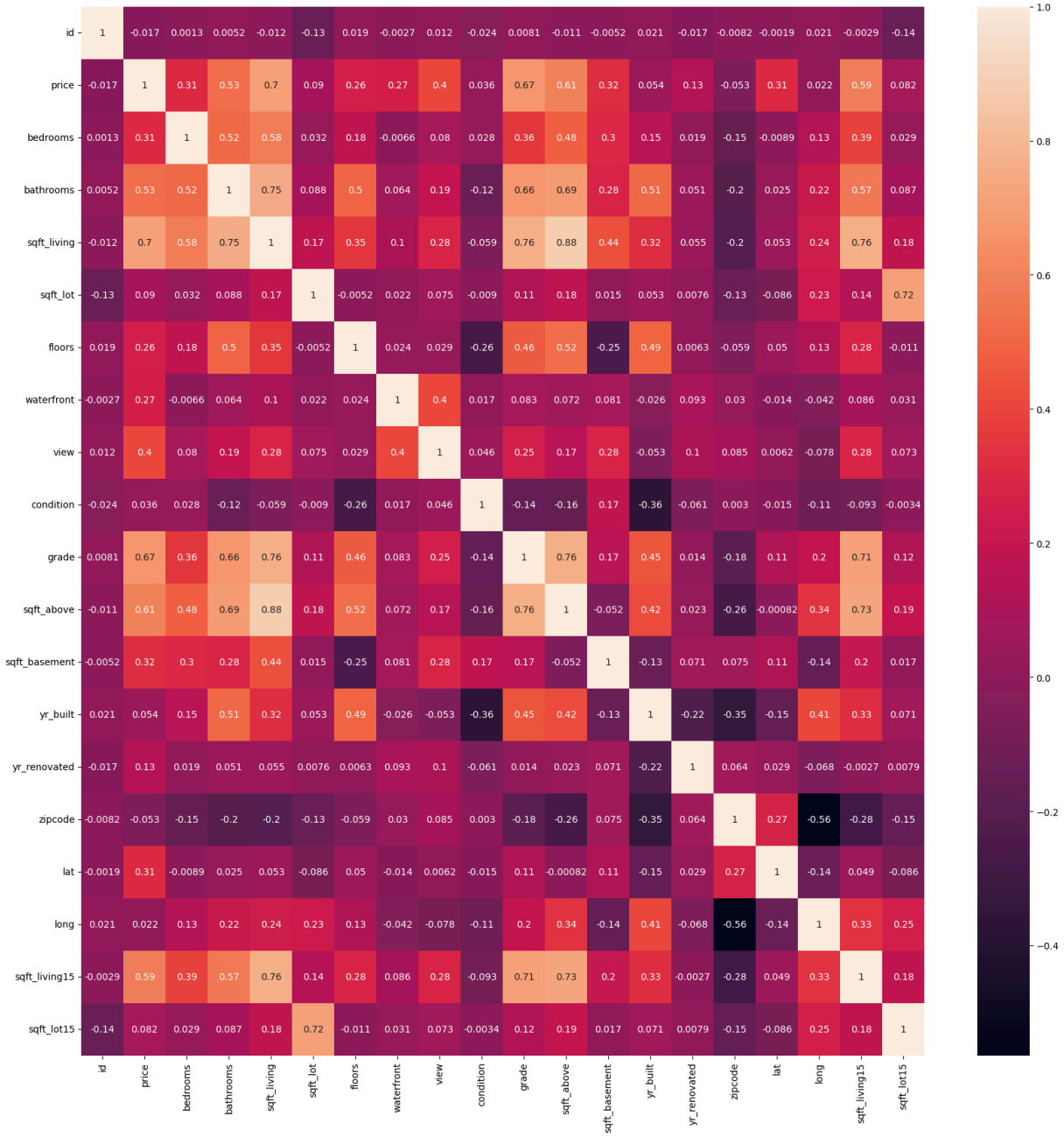
Para verificarmos se existem valores vazios no dataset utilizamos a função abaixo. Caso existam precisam ser tratados.

In [21]:
```
base_casas.isnull().sum()
```

Out[21]:
```
id               0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

Vamos criar o mapa de calor para analisar as correlações das features.

In [22]:
```
figura = plt.figure(figsize=(20,20))
sns.heatmap(base_casas.corr(),  annot=True);
```



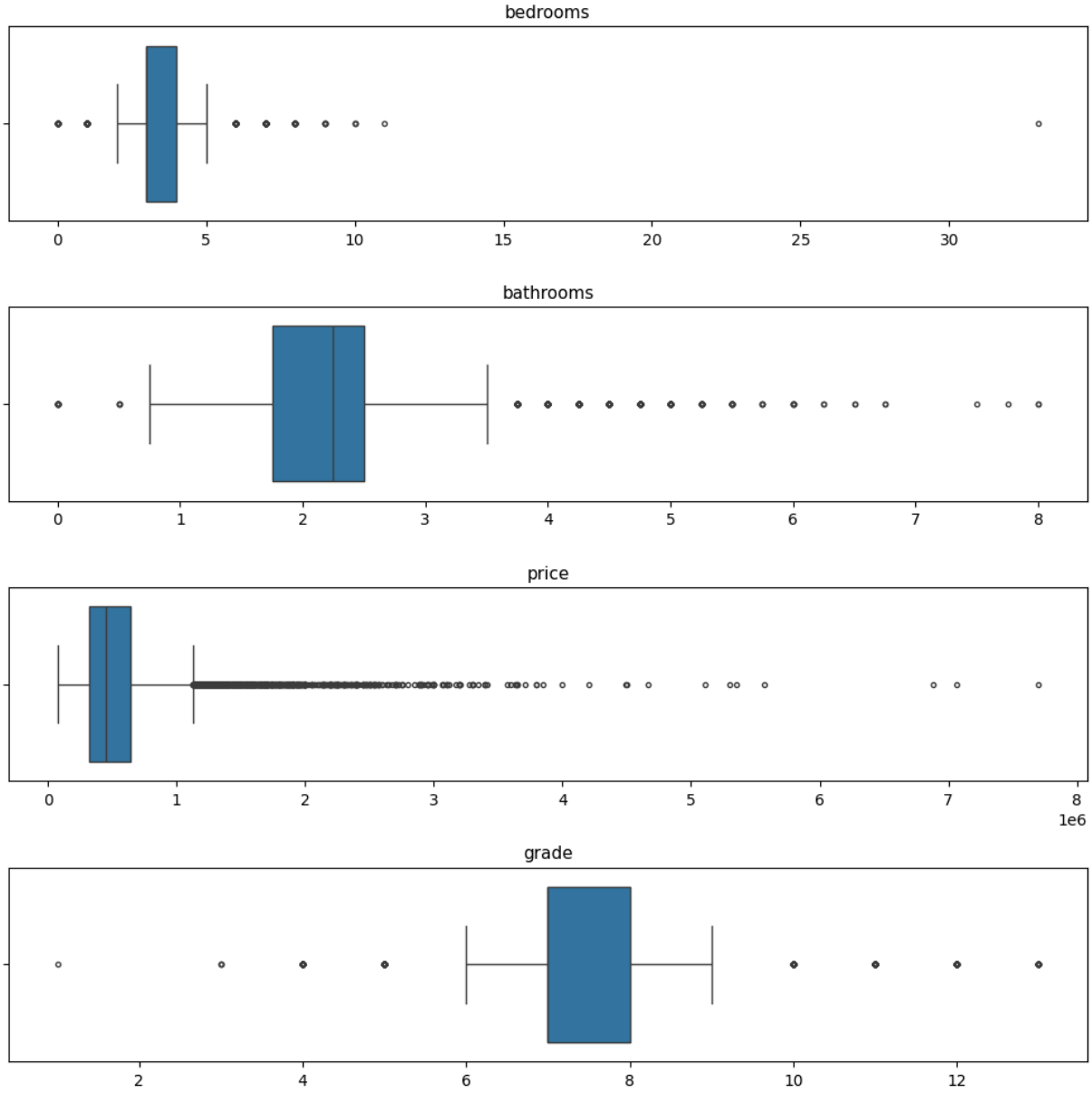
In [ ]:

In [23]:
```
colunas = ['bedrooms', 'bathrooms', 'price', 'grade']

plt.figure(figsize=(10, 10))

for i, coluna in enumerate(colunas, 1):
    plt.subplot(len(colunas), 1, i)  # cada boxplot ocupa uma linha
    sns.boxplot(data=base_casas, x=coluna, fliersize=3, linewidth=1)
    plt.title(coluna, fontsize=11)
    plt.xlabel('')
    plt.grid(axis='y', linestyle='--', alpha=0)

plt.tight_layout()
```



Como o foco nesse caso é prever o preço, observamos as features que tem mais relação com a coluna "price". Podemos concluir que as variáveis que mais influenciam o valor do imóvel são: "sqft_living" (tamanho do imóvel), "sqft_lot" (tamanho do terreno), "grade" (nota do imóvel), "bathrooms" (número de banheiros). Vamos selecionar as colunas em x que utilizaremos para prever.

In [24]:
```
X_casas = base_casas.iloc[:, 2:18].values
```

E em y vamos selecionar a coluna "price" que queremos prever.

In [25]:
```
y_casas = base_casas.iloc[:, 1].values
```

Vamos importar a função para separarmos os dados de treinamento e teste, separando 30% dos dados para treinamento. O parâmetro random_state é como se fosse uma semente que você planta antes de usar a sorte (aleatoriedade) no seu código. Ele serve para congelar a

aleatoriedade toda vez que você rodar o código, a divisão dos dados ou o resultado do seu modelo será diferente.

In [76]:
```
from sklearn.model_selection import train_test_split
X_casas_treinamento, X_casas_teste, y_casas_treinamento, y_casas_teste = train_test
```

In [77]:
```
X_casas_treinamento.shape, X_casas_teste.shape
```

Out[77]:
```
((17290, 16), (4323, 16))
```

Agora vamos treinar o algoritmo.

In [78]:
```
regressor_multiplo_casas = LinearRegression()
regressor_multiplo_casas.fit(X_casas_treinamento, y_casas_treinamento)
```

Out[78]:

▾ LinearRegression

▶ Parameters

Podemos medir a qualidade do treinamento dos dados com o algoritmo.

In [79]:
```
regressor_multiplo_casas.score(X_casas_treinamento, y_casas_treinamento)
```

Out[79]:
```
0.6983188500566084
```

E também a qualidade dos dados de teste.

In [80]:
```
regressor_multiplo_casas.score(X_casas_teste, y_casas_teste)
```

Out[80]:
```
0.6999397683935713
```

Podemos fazer as previsões com os dados que separamos para teste e comparar com os valores reais.

In [81]:
```
previsoes = regressor_multiplo_casas.predict(X_casas_teste)
previsoes
```

Out[81]:
```
array([ 452543.74330989,  747488.69498968, 1233807.42966791, ...,
        415902.13479391,  622706.21412473,  435461.0654793 ])
```

In [82]:
```
y_casas_teste
```

Out[82]:
```
array([ 365000.,  865000., 1038000., ...,  285000.,  605000.,  356500.])
```

Métricas de erro

Mean absolute error (MAE)

Diferenças absolutas entre as previsões e os valores reais

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean squared error (MSE)

Diferenças elevadas ao quadrado (erros penalizados)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root mean squared error (RMSE)

Interpretação facilitada, pois no MSE os erros podem ter valores muito altos assim tiramos a raiz para facilitar.

$$\boldsymbol{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}$$

Vamos aplicar todos e ver as diferenças.

In [83]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
mae = mean_absolute_error(y_casas_teste, previsoes)
print(f"MAE: {mae}")
```

```
MAE: 127744.42981458819
```

Em média, a previsão do modelo de preço está errada por 123.888,44 para mais ou menos

In [84]:
```
mse = mean_squared_error(y_casas_teste, previsoes)
print(f"MSE: {mse}")
```

```
MSE: 45362119293.29081
```

Este é o erro quadrático médio, o número é muito grande porque os erros foram elevados ao quadrado, o número não está na unidade de preço e é muito grande. Sua principal função é penalizar erros maiores drasticamente. Se o modelo comete um erro de 500.000, o MSE o transforma em 250 bilhões. Isso torna o MSE sensível a outliers ou erros extremos.

In [85]:
```
rmse = np.sqrt(mean_squared_error(y_casas_teste, previsoes))
print(f"RMSE: {rmse}")
```

```
RMSE: 212983.84749386704
```

Este é a raiz quadrada do MSE, ele traz a métrica de volta para as unidades originais de preço. O RMSE é geralmente maior que o MAE (206.786,74 vs. 123.888,44). Essa diferença é crucial: O RMSE é maior que o MAE porque a raiz quadrada não consegue desfazer totalmente a penalização que o MSE aplicou aos erros grandes. Conclusão: O erro médio é de ≈124.000 (MAE), mas o erro que mais pesa no modelo é ≈207.000 (RMSE), indicando que as grandes falhas estão puxando o RMSE para cima. Na prática, o RMSE é frequentemente preferido porque equilibra a interpretabilidade (unidades originais) com a penalização de erros grandes.

O objetivo é sempre que esses erros sejam os menores possíveis.

Regressão Linear Polinomial

A Regressão Polinomial é um método de regressão utilizado quando a relação entre a variável independente (X) e a dependente (y) é não linear, ou seja, curva, e não pode ser representada por uma linha reta simples. Embora modele essa relação curva como um polinômio de n graus, ela ainda é classificada como um modelo linear porque a relação é linear em seus coeficientes (m), que são os parâmetros que o modelo aprende, como na fórmula abaixo.

$P = C + m1 * v1 + m2 * v1^2 + ... + mn * v1^n$

Vamos importar a classe do algoritmo e criar um objeto transformador, definimos o parâmetro do grau do polinômio a ser criado. Um grau 2 (quadrático) significa que o transformador adicionará as colunas originais elevadas ao quadrado e as combinações cruzadas das colunas como novas features.

In [86]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2)
```

Precisamos adequar os dados para modelar relações não-lineares, ou seja, curvas, mesmo utilizando um algoritmo que é fundamentalmente linear. O objetivo é permitir que o modelo aprenda curvas. Para fazer isso, o transformador cria novas colunas (novas features):

In [87]:
```
X_casas_treinamento_poly = poly.fit_transform(X_casas_treinamento)
X_casas_teste_poly = poly.transform(X_casas_teste)
```

Em seguida, usamos os dados transformados para treinar o modelo de regressão linear.

In [88]:
```
regressor_casas_poly = LinearRegression()
regressor_casas_poly.fit(X_casas_treinamento_poly, y_casas_treinamento)
```

Out[88]:

▾ LinearRegression

▶ Parameters

In [89]:
```
regressor_casas_poly.score(X_casas_treinamento_poly, y_casas_treinamento)
```

Out[89]:
```
0.7983246387758883
```

In [90]:
```
regressor_casas_poly.score(X_casas_teste_poly, y_casas_teste)
```

Out[90]:
```
0.7752461260971509
```

In [91]:
```
previsoes = regressor_casas_poly.predict(X_casas_teste_poly)
previsoes
```

Out[91]:
```
array([ 433114.85649762,  747478.18812404, 1230945.10005174, ...,
        391716.34638009,  589479.88327298,  410396.90534148])
```

In [92]:
```
y_casas_teste
```

Out[92]:
```
array([ 365000.,  865000., 1038000., ...,  285000.,  605000.,  356500.])
```

In [93]:
```
mean_absolute_error(y_casas_teste, previsoes)
```

Out[93]:
```
109563.62719685062
```

In [94]:
```
mse = mean_squared_error(y_casas_teste, previsoes)
rmse = np.sqrt(mean_squared_error(y_casas_teste, previsoes))
print(f"RMSE: {rmse}")
```

```
RMSE: 184330.0076591569
```

Regressão por Arvore de Decisão

O algorítimo decision tree é utilizado para fazer previsões usando um principio de “separar” os dados em grupos de forma hierárquica. O objetivo é dividir os dados em grupos (nós) onde os valores da variável alvo y sejam o mais parecidos possível. Cada folha faz uma previsão = média dos valores dentro dela.

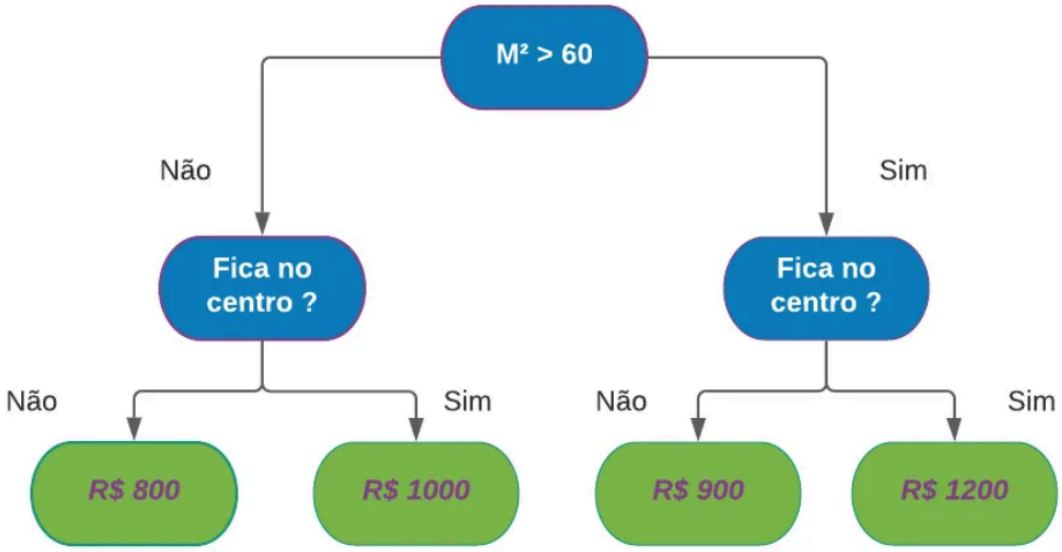
O que se mede é a dispersão dos valores numéricos, geralmente com Erro Quadrático Médio (MSE). Passo a passo:

- Divide o conjunto em dois.
- Calcula o MSE antes e depois da divisão.
- Mede o ganho.
- Escolhe o corte com melhor ganho.
- O processo se repete recursivamente em cada novo nó, até: a variância ficar pequena, ou atingir limites (profundidade máxima, mínimo de exemplos, etc.).

Como o exemplo a seguir.

	Quartos	Banheiros	M2	vaga_estac	valor_mes	fica no centro
0	1	1	40	Sim	800	Não
1	2	2	75	Sim	1700	Não
2	1	1	40	Sim	1000	Sim
3	1	2	60	Não	1200	Sim
4	2	1	60	Não	900	Não

Observamos que apartamentos que tem acima de 60 m^2 são em media mais caros, não só isso, aparentemente os imóveis que ficam no centro tem um aluguel mais caro. Sendo assim podemos separar os dados em dois grupos os que tem mais de 60 m^2 e os que são menores. Depois, dentro de cada um desses grupos,separamos os que ficam no centro e os que não. No final temos 4 grupos e um valor médio de aluguel para cada um deles.



O exemplo ilustra a dinâmica de como funciona uma árvore de decisão, onde se passam os dados, o algorítimo separa os grupos e subgrupos ate chegar nas "folhas" e nelas teremos o valor estimado para a previsão. Esse processo é o "treinamento" do algorítimo, pois é onde o modelo aprende os padrões que existem nos seus dados.

In [95]:
```
from sklearn.tree import DecisionTreeRegressor
regressor_arvore_casas = DecisionTreeRegressor()
regressor_arvore_casas.fit(X_casas_treinamento, y_casas_treinamento)
```

Out[95]:
```
DecisionTreeRegressor
▶ Parameters
```

In [96]:
```
regressor_arvore_casas.score(X_casas_treinamento, y_casas_treinamento)
```

Out[96]:
```
0.9992343746075667
```

In [97]:
```
regressor_arvore_casas.score(X_casas_teste, y_casas_teste)
```

Out[97]:
```
0.7078756492774958
```

A segunda parte, conhecida como "predição" é quando comparamos as informações de dados que o algorítimo ainda não viu com os padrões que a árvore encontrou.

In [98]:
```
previsoes = regressor_arvore_casas.predict(X_casas_teste)
previsoes
```

Out[98]:
```
array([ 435000.,  696000., 1225000., ...,  280000.,  620000.,  346300.])
```

In [99]:
```
y_casas_teste
```

Out[99]:
```
array([ 365000.,  865000., 1038000., ...,  285000.,  605000.,  356500.])
```

In [100…
```
mean_absolute_error(y_casas_teste, previsoes)
```

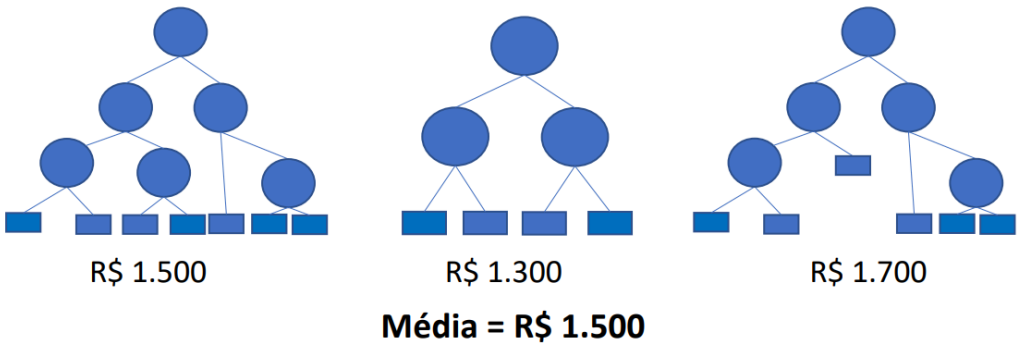
Out[100…
```
104398.26347443905
```

In [101…
```
mse = mean_squared_error(y_casas_teste, previsoes)
rmse = np.sqrt(mean_squared_error(y_casas_teste, previsoes))
print(f"RMSE: {rmse}")
```

```
RMSE: 210148.51632214076
```

Random Forest

Esse algoritmo é baseado em árvores de decisão, no entanto diferente do decision tree ele utiliza inúmeras árvores diferentes para chegar a conclusão final. Utiliza o chamado ensemble learning (aprendizagem em conjunto), como para "Consultar diversos profissionais para tomar uma decisão" no caso vários algoritmos juntos para construir um algoritmo mais "forte".



Conforme a imagem ele realiza difersas consultas e usa a média (regressão) para dar a resposta final.

In [102…
```
from sklearn.ensemble import RandomForestRegressor
regressor_random_forest_casas = RandomForestRegressor(n_estimators=100)
regressor_random_forest_casas.fit(X_casas_treinamento, y_casas_treinamento)
```

Out[102…
```
▾ RandomForestRegressor
▶ Parameters
```

In [103…
```
regressor_random_forest_casas.score(X_casas_treinamento, y_casas_treinamento)
```

Out[103…
```
0.9820120985970889
```

In [104…
```
regressor_random_forest_casas.score(X_casas_teste, y_casas_teste)
```

Out[104…
```
0.852328882126942
```

previsoes = regressor_random_forest_casas.predict(X_casas_teste) previsoes

In [105…
```
y_casas_teste
```

Out[105…
```
array([ 365000.,  865000., 1038000., ...,  285000.,  605000.,  356500.])
```

In [106…
```
mean_absolute_error(y_casas_teste, previsoes)
```

Out[106…
```
104398.26347443905
```

In [107…
```
mse = mean_squared_error(y_casas_teste, previsoes)
rmse = np.sqrt(mean_squared_error(y_casas_teste, previsoes))
print(f"RMSE: {rmse}")
```

```
RMSE: 210148.51632214076
```

Algoritimos Regularizadores

O que a regressão linear comum faz é tentar achar uma reta (ou plano) que melhor se encaixa nos dados, ou seja, encontrar os pesos (coeficientes - intersecção) que multiplicam cada variável para minimizar o erro entre o valor previsto e o valor real. Mas às vezes acontece um problema. Quando há muitas variáveis ou elas estão muito parecidas entre si, o modelo "se confunde" e acaba dando pesos muito grandes ou não coerentes. Isso faz o modelo funcionar bem nos dados de treino, mas errar muito em novos dados (problema de overfitting). Para evitar isso, usamos regularização, que é como colocar uma "penalidade" nos pesos muito grandes.

Ridge Regression

Ridge penaliza o quadrado dos coeficientes. Isso "puxa" os coeficientes para valores pequenos (mas não zera exatamente), reduzindo variância do modelo e ajudando quando há multicolinearidade em modelos de regressão onde as variáveis independentes estão altamente correlacionadas entre si, o que pode dificultar a interpretação dos resultados do modelo

$$\min_{w} \sum_{i=1}^{N} (\vec{w} \cdot \vec{x}_i + \beta - y_i)^2 + \alpha w^2$$

onde $\alpha \geq 0$ controla a força da regularização e w^2 coloca cada coeficiente ao quadrado → isso é chamado de norma L2. Ridge mantém todos os coeficientes, mas os diminui.

Imagine que o modelo está dizendo:

preço = 200 × quartos + 0.3 × tamanho + 500 × localização

Esses pesos (200, 0.3, 500) podem ser exagerados. O Ridge “puxa” esses números para valores menores, tipo:

preço = 120 × quartos + 0.2 × tamanho + 300 × localização

Assim, o modelo fica mais equilibrado e mais estável. Pense no Ridge como um elástico preso nos pesos — quanto mais eles tentam crescer, mais o elástico puxa de volta.

Lasso Regression

Sua ideia é punir pesos grandes mas também puni-los igualmente os pequenos, possibilitando que valores zero apareçam nos pesos e indiretamente realizando uma remoção de colunas que terão valores desconsiderados no processo. Seu termo é semelhante, mas ao invés de usar um termo quadrático, usa um linear em módulo:

$$\min_{w} \sum_{i=1}^{N} (\vec{w} \cdot \vec{x}_i + \beta - y_i)^2 + \alpha |w|$$

Ou seja, ele descarta variáveis que não ajudam muito.

preço = 200 × quartos + 0.3 × tamanho + 500 × localização + 0 × idade_da_casa

Perceba que “idade_da_casa” ficou com peso zero. Isso significa que o Lasso eliminou essa variável do modelo. Pense no Lasso como uma tesoura: corta fora as variáveis menos importantes.

Elastic Net

Por fim, mas não menos importante, é o Elastic-Net que mistura tanto a abordagem do Ridge quanto do Lasso:

$$\min_{w} \sum_{i=1}^{N} (\vec{w} \cdot \vec{x}_i + \beta - y_i)^2 + \alpha\rho |w| + \alpha(1-\rho) w^2$$

No caso introduzimos um parâmetro pho. Quando rho é igual a 0 temos Ridge, quando é igual a 1 temos Lasso, e quando é um valor intermediário temos uma composição entre os dois métodos.

Usa o “elástico” do Ridge para controlar exageros Usa a “tesoura” do Lasso para eliminar variáveis inúteis

preço = 150 × quartos + 0.2 × tamanho + 0 × idade_da_casa + 400 × localização

Aqui ele reduziu alguns pesos e zerou outros. O Elastic Net encontra o equilíbrio entre os dois anteriores.

O α (alfa) é o nível da penalização — ou seja, o "quanto apertado" está o elástico ou a "força do corte".

Se α = 0, é uma regressão normal (sem penalização). Se α é alto, o modelo fica mais simples, com pesos menores e menos variáveis.

Padronização dos Dados

O dimensionamento dos dados é um recurso muito utilizado na fase inicial de todo pipeline de Machine Learning. Olhar para as escalas das features do conjunto de dados é importante, porque essa diferença de escalas diferença de escala pode "degradar" o desempenho preditivo de muitos algoritmos.

StandardScaler

Esse método age sobre as colunas dos dados que estamos pré-processando.A documentação oficial nos traz que o StandardScaler "padroniza as features removendo a média e a escala a uma variância a uma unidade". Em outras palavras, isso significa que para cada feature, a média seria 0, e o Desvio Padrão seria 1. Em resumo, o método subtrai do valor em questão a média da coluna e divide o resultado pelo desvio padrão:

$$\frac{x_i - \mu(x)}{\sigma} \qquad \frac{x_i - \mu(x)}{\sigma}$$

Porque para o computador (que chamamos de "algoritmo"), é mais fácil e justo comparar coisas quando elas estão em uma escala parecida. Se uma f1 tem tamanho 1 e f2 tem tamanho 1000, o computador pode pensar que f2 é muito mais importante só por ser maior.

Exemplo:

Pessoa	Idade (Anos)	Salário (R$)
A	25	3.000
B	60	50.000
C	40	6.000

O StandardScaler é como um "tradutor de escalas" que diz para o computador:

- Achar o Ponto Médio (Média):

Ele calcula a idade média de todos (digamos que seja 42 anos). Ele calcula o salário médio de todos (digamos que seja R$ 19.666).

- Centralizar no Zero:

Para a Idade, ele move os 42 anos para a posição 0. Pessoas mais jovens que 42 terão um número negativo, e pessoas mais velhas terão um número positivo. Para o Salário, ele move os R$ 19.666 para a posição 0. Salários menores que a média terão um número negativo, e salários maiores terão um número positivo.

- Ajustar a "Esticada" (Desvio Padrão):

Ele garante que a "distância" entre a idade mais nova e a mais velha (ou o salário mais baixo e o mais alto) não seja gigante. Ele aperta e estica as duas colunas para que ambas tenham a mesma "variedade" ou "esticada" (chamada de desvio-padrão igual a 1).

Pessoa	Idade (Transformada)	Salário (Transformado)
A	-1,2	-0,8
B	1,8	2,1
C	-0,1	-0,4

O uso do StandardScaler é crucial para o Elastic Net e qualquer modelo que use regularização. A razão principal é que o Elastic Net usa uma técnica de "punição" nos seus cálculos que é muito sensível à escala dos dados. O Elastic Net (e seus primos Lasso e Ridge)

funcionam minimizando uma função de custo, onde eles tentam não só minimizar o erro de previsão, mas também punir os coeficientes grandes. A punição (ou regularização) é o que faz o modelo ser chamado de Elastic Net e tem o objetivo de evitar o overfitting. Se você não padronizar os dados: Uma variável com números grandes (como Salário em R$) e outra com números pequenos (como Idade em Anos) terão pesos muito diferentes só por causa da escala. O modelo pode ser forçado a dar um coeficiente muito pequeno para o Salário (que tem números grandes) e um coeficiente muito grande para a Idade (que tem números pequenos) para que o efeito das duas variáveis no resultado final seja equilibrado. A penalidade da regularização vai "achar" que o coeficiente da Idade é muito grande e vai puni-lo mais, mesmo que a Idade não seja necessariamente mais importante que o Salário. Com todas as variáveis na mesma escala, o Elastic Net pode aplicar a punição de forma justa e igualitária em todos os coeficientes.

Assim valor padronizar as nossas features importando a função StandardScaler, e testar com a regressão comum.

In [150…
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_casas_treinamento = scaler.fit_transform(X_casas_treinamento)
X_casas_teste = scaler.transform(X_casas_teste)

modelo_lr = LinearRegression()
modelo_lr.fit(X_casas_treinamento, y_casas_treinamento)

y_pred_lr = modelo_lr.predict(X_casas_treinamento)
score = modelo_lr.score(X_casas_treinamento, y_casas_treinamento)
score
```

Out[150…
```
0.698318850056608
```

In [151…
```
score = modelo_lr.score(X_casas_teste, y_casas_teste)
score
```

Out[151…
```
0.6999397683935455
```

Agora vamos aplicar com o Elastic Net.

In [157…
```
from sklearn.linear_model import ElasticNet

modelo_en = ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=0)
modelo_en.fit(X_casas_treinamento, y_casas_treinamento)

y_pred_en = modelo_en.predict(X_casas_treinamento)
```

As duas coisas mais importantes que podemos alterar para melhorar o algoritmo são o alpha e o l1_ratio. Eles são chamados de hiperparâmetros e controlam como o modelo Elastic Net funciona.

O alpha é o hiperparâmetro que controla a força total da regularização (a punição) aplicada ao modelo.

- Alpha pequeno(0,001): O modelo é quase uma Regressão Linear Simples. Ele tenta se ajustar o máximo possível aos dados de treino, mas corre o risco de overfitting (ficar bom demais no treino e ruim em dados novos).
- Aplha grande (10): O modelo "encolhe" muito os pesos das variáveis, tornando-o mais simples. Pode levar ao underfitting (ficar ruim tanto no treino quanto em dados novos) se for muito alto.
- Alpha zero: Nenhuma regularização.

Você deve testar uma variedade de valores [0.001, 0.01, 0.1, 1.0, 10] para ver qual dá o melhor resultado em seus dados de teste.

O l1_ratio é o hiperparâmetro que controla o equilíbrio entre os dois tipos de regularização: L1 (Lasso) e L2 (Ridge). Ele é um número que vai de 0 a 1.

- l1_ratio = 0: Punição 100% L2 (Ridge)
- l1_ratio = 1: Punição 100% L1 (Lasso)
- l1_ratio = 0.5: Punição 50% L1 e 50% L2

Você pode variar esse valor para determinar se seus dados se beneficiam mais da seleção de variáveis do Lasso (valores mais próximos de 1) ou da estabilidade do Ridge (valores mais próximos de 0).

In [158…
```
score = modelo_en.score(X_casas_treinamento, y_casas_treinamento)
score
```

Out[158…
```
0.6974582676453411
```

In [159…
```
score = modelo_en.score(X_casas_teste, y_casas_teste)
score
```

Out[159…
```
0.6977591397217584
```

In [2]:
```
!export PATH=/Library/TeX/texbin:$PATH
```

```
'export' is not recognized as an internal or external command,
operable program or batch file.
```

In [ ]: