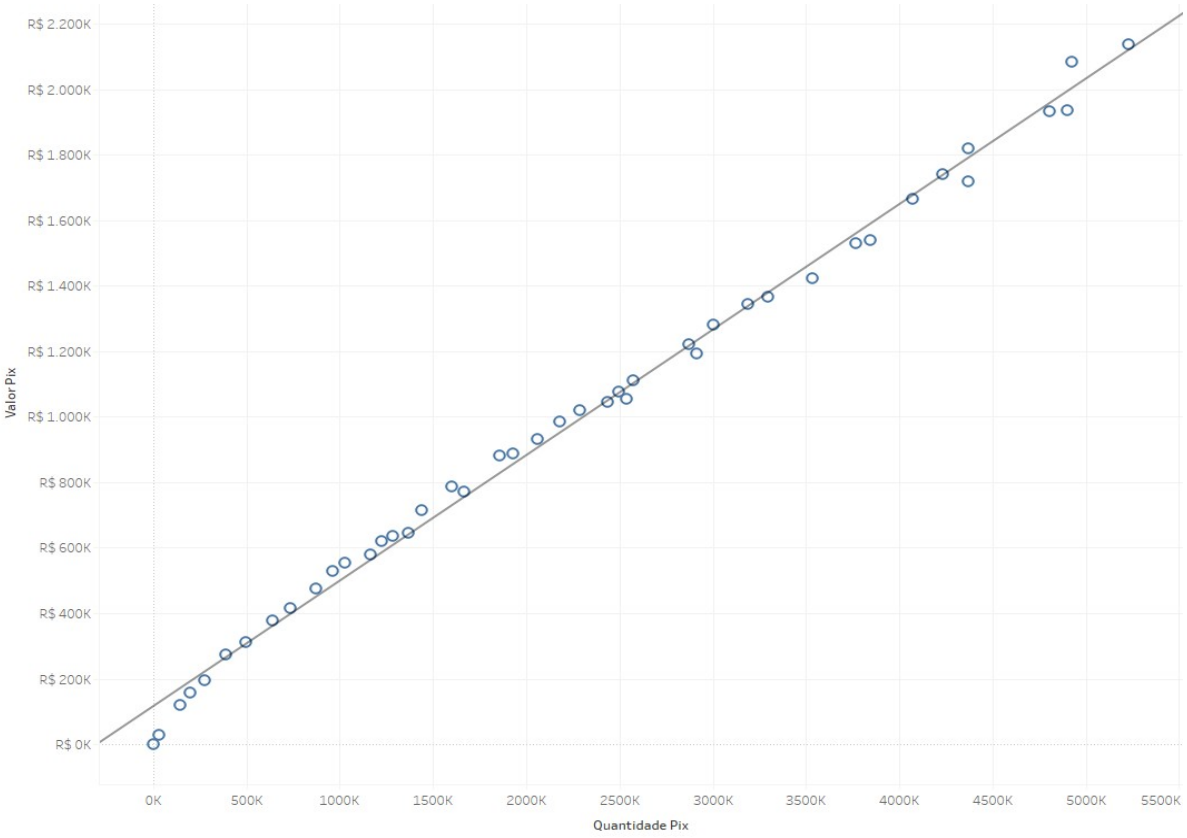
REGRESSÃO LINEAR

Simples e Múltipla



Rafael Oliveira & ChatGPT

REGRESSÃO LINEAR

SIMPLES E MÚLTIPLA

BY

Rafael Oliveira & ChatGPT

Este ebook foi escrito inteiramente pelo ChatGPT e com edição feita por mim no MS Word, não há nenhum objetivo comercial ou didático. Trabalho realizado como um projeto proposto pelo Bootcamp Microsoft Copilot AI.

Dedico este ebook para todos.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. INTRODUÇÃO

A regressão linear é uma das técnicas de análise estatística mais comuns e utilizadas para prever relações entre variáveis. Ela é amplamente aplicada em diversas áreas, como economia, finanças, ciências sociais e biologia.

Neste eBook, exploraremos os conceitos básicos, a aplicação prática e as nuances dessa poderosa ferramenta estatística.

2. REGRESSÃO LINEAR

A regressão linear é um método estatístico utilizado para modelar a relação entre uma variável dependente e uma ou mais variáveis independentes. É uma técnica que busca estimar a linha reta que melhor se ajusta aos dados, de forma a prever ou explicar o comportamento da variável dependente com base nas variáveis independentes.

A regressão linear pode ser chamada de "simples" quando envolve apenas uma variável independente e "múltipla" quando envolve mais de uma variável independente. A ideia básica é que a variável dependente pode ser expressa como uma combinação linear das variáveis independentes, multiplicadas pelos seus respectivos coeficientes.

O objetivo da regressão linear é estimar os coeficientes de regressão, que representam a contribuição das variáveis independentes na variação da variável dependente. Esses coeficientes permitem entender a relação entre as variáveis e realizar previsões ou inferências.

Através da análise dos resíduos, que são as diferenças entre os valores observados e os valores estimados pela regressão, é possível diagnosticar a qualidade do ajuste do modelo. Se os resíduos apresentarem um padrão aleatório, significa que a regressão linear é adequada para os dados. Caso contrário, podem existir problemas como heteroscedasticidade (variância não constante dos resíduos) ou multicolinearidade (alta correlação entre as variáveis independentes).

A regressão linear é uma ferramenta poderosa para análise de dados e tomada de decisões. Ela permite quantificar e entender a relação entre as variáveis, fazer previsões e testar hipóteses. No entanto, é importante ter em mente que a regressão linear pressupõe certas condições e pode não ser adequada para todos os tipos de dados e problemas. É fundamental entender os fundamentos teóricos e realizar análises cuidadosas antes de interpretar os resultados.

3. REGRESSÃO LINEAR SIMPLES

A regressão linear simples é uma técnica estatística que busca modelar a relação entre duas variáveis, sendo uma considerada a variável dependente (y) e a outra a variável independente (x). A ideia principal é encontrar a linha reta que melhor se ajusta aos pontos no gráfico de dispersão dos dados, de forma a prever ou explicar o comportamento da variável dependente com base na variável independente.

A equação da regressão linear simples é dada por:

$$yi = \beta 0 + \beta 1 * xi + \varepsilon$$

Onde:

- yi é valor observável da variável dependente.
- xi é o valor observável da variável independente.
- β0 é o intercepto, que representa o ponto em que a linha de regressão intersecta o eixo y (valor de y quando x é igual a zero).
- β1 é o coeficiente de inclinação, que representa a mudança em y para cada unidade de mudança em x.
- ε é o termo de erro, que representa a variação no valor de y não explicada pelo modelo de regressão.

O objetivo da regressão linear simples é encontrar os valores de β0 e β1 que melhor se ajustam aos dados. Esses valores são estimados utilizando métodos de mínimos quadrados, que minimizam a soma dos quadrados das diferenças entre os valores observados de y e os valores estimados pela regressão.

Com base nos coeficientes de regressão estimados, é possível fazer previsões para valores futuros de y com base em valores conhecidos ou observados da variável independente x. Além disso, os coeficientes também permitem entender a relação entre as variáveis, ou seja, como uma mudança em x afeta o valor de y.

É importante destacar que a regressão linear simples pressupõe certas condições, como linearidade da relação entre as variáveis, homoscedasticidade dos erros e independência dos erros. É necessário verificar se essas condições são atendidas antes de fazer inferências ou interpretações baseadas no modelo de regressão.

3.1. MÉTODO DOS MÍNIMOS QUADRADOS

O método dos mínimos quadrados é uma técnica estatística usada para ajustar um modelo de regressão linear aos dados observados. O objetivo do método é encontrar os coeficientes da equação de regressão que minimizam a soma dos quadrados dos erros (resíduos) entre os valores observados e os valores previstos pelo modelo.

SSE – Sum of Squared Errors - (Soma dos quadrados dos erros):

$$SSE = \sum_{i=1}^{n} (yi - \hat{y}\,i)^2$$

Onde ŷi é o valor previsto pelo modelo para a observação i no eixo y.

O método consiste em calcular as derivadas parciais das diferenças entre os valores observados e os valores estimados em relação a cada um dos coeficientes, e igualar essas derivadas a zero. Isso resulta em um sistema de equações lineares, chamadas de equações normais, que podem ser resolvidas para encontrar os valores dos coeficientes.

Para encontrar os valores ótimos de β0 e β1, derivamos as equações normais a partir da função objetivo (SSE).

As equações normais são:

$$\sum_{i=1}^{n} (yi \; - \; \beta 0 \; - \; \beta 1\, xi) \; = \; 0$$

$$\sum_{i=1}^{n} xi\,(yi \; - \; \beta 0 \; - \; \beta 1\, xi) \; = \; 0$$

Resolvendo as equações normais, obtemos as fórmulas para β0 e β1:

$$\beta 1 = \frac{\sum_{i=1}^{n} (xi \; - \; \bar{x})(yi \; - \; \bar{y})}{\sum_{i=1}^{n} (xi \; - \; \bar{x})^2}$$

$$\beta 0 = \bar{y} \; - \; \beta 1\, \bar{x}$$

Onde:

- $\bar{x}$ é a média da variável x.
- $\bar{y}$ é a média da variável y.
- β1 representa a inclinação da linha de regressão.
- β0 representa o valor esperado de y quando x = 0, chamado de intercepto.

3.2. EXEMPLO: MODELO DE REGRESSÃO LINEAR SIMPLES

Um exemplo prático de regressão linear simples pode ser a análise da relação entre a experiência de trabalho (em anos) e o salário anual (em milhares de reais) de funcionários em uma empresa. Vamos imaginar que temos os seguintes dados de uma amostra de 10 funcionários:

Anos de Experiência	Salário Anual (R$ mil)
1	40
2	42
3	45
4	47
5	50
6	52
7	54
8	57
9	60
10	62

Tabela 1: Exemplo de Regressão Linear Simples
Fonte: ChatGPT, 2024

Podemos utilizar a regressão linear simples para modelar a relação entre os anos de experiência (variável independente X) e o salário anual (variável dependente Y).

3.2.1. PASSOS PARA REALIZAR A REGRESSÃO LINEAR SIMPLES

I. Plotar os dados:

Primeiramente, podemos visualizar os dados em um gráfico de dispersão para verificar se há uma relação linear aparente entre as variáveis.

II. Calcular os coeficientes da regressão:

Utilizamos a fórmula da regressão linear simples $y = \beta 0 + \beta 1 * x + \varepsilon$, onde:

- y é a variável dependente (Salário Anual).
- x é a variável independente (Anos de Experiência).
- β0 é o intercepto.
- β1 é o coeficiente de inclinação.
- ϵ é o erro.

Podemos calcular β0 e β1 utilizando as fórmulas do método dos mínimos quadrados:

$$\beta 1 = \frac{\sum_{i=1}^{n} (xi - \bar{x})(yi - \bar{y})}{\sum_{i=1}^{n} (xi - \bar{x})^2}$$

$$\beta 0 = \bar{y} - \beta 1 \bar{x}$$

- β1 = 2.46667.
- β0 = 37.33333.

III. Estimar o modelo e interpretar os resultados:

Com os coeficientes calculados, podemos prever o valor de y dado um valor de x.

- Anos de Experiência = 12
- $\hat{y} = \beta_0 + (\beta_1 \cdot x) \rightarrow \hat{y} = 37.33333 + (2.46667 \cdot 12) + (2.46667 \cdot 12) \rightarrow \hat{y} \approx 66.94$

3.2.2. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Dados
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
Y = np.array([40, 42, 45, 47, 50, 52, 54, 57, 60, 62])

# Modelo de Regressão Linear
model = LinearRegression()
model.fit(X, Y)

# Coeficientes
beta_1 = model.coef_[0]
beta_0 = model.intercept_

print(f"Coeficiente de inclinação (beta_1): {beta_1}")
print(f"Intercepto (beta_0): {beta_0}")

# Previsão
Y_pred = model.predict(X)

# Plotar os dados e a linha de regressão
plt.scatter(X, Y, color='blue', label='Dados reais')
plt.plot(X, Y_pred, color='red', label='Linha de Regressão')
plt.xlabel('Anos de Experiência')
plt.ylabel('Salário Anual (R$ mil)')
plt.title('Regressão Linear Simples')
plt.legend()
plt.show()
```

Figura 1: Implementação de Regressão Linear Simples no Python
Fonte: ChatGPT, 2024

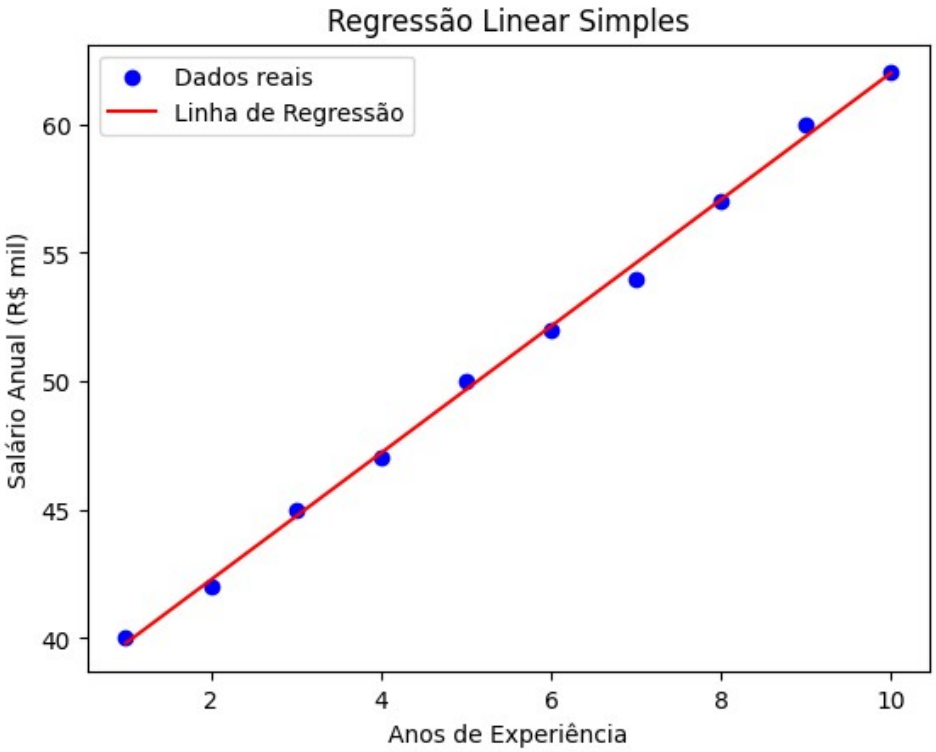


Figura 2: Gráficos de Regressão Linear Simples
Fonte: Autor

4. DIAGNÓSTICO DE MODELOS DE REGRESSÃO

O diagnóstico de modelos de regressão é um conjunto de técnicas utilizadas para avaliar a adequação e a validade de um modelo de regressão. Essas técnicas ajudam a identificar problemas e melhorar o modelo para garantir que ele faça previsões precisas e confiáveis. Aqui estão alguns dos principais aspectos e ferramentas usadas no diagnóstico de modelos de regressão:

- **Resíduos (Erros) do Modelo:**
  - Análise de Resíduos: Avaliar os resíduos (diferença entre os valores observados e os valores previstos) pode revelar se há padrões não capturados pelo modelo.
  - Gráficos de Resíduos: Gráficos como resíduos vs valores ajustados, resíduos vs variáveis independentes e histogramas dos resíduos ajudam a identificar heterocedasticidade, não linearidade e outliers.
- **Suposições do Modelo:**
  - Linearidade: Verificar se a relação entre as variáveis independentes e a variável dependente é linear.
  - Independência: Checar se os resíduos são independentes.
  - Homoscedasticidade: Verificar se os resíduos têm variância constante.
  - Normalidade dos Resíduos: Usar testes estatísticos e gráficos (como o Q-Q plot) para verificar se os resíduos seguem uma distribuição normal.
- **Multicolinearidade:**
  - VIF (Variance Inflation Factor): Medir a inflação da variância de um coeficiente devido à colinearidade entre as variáveis independentes.
  - Matriz de Correlação: Analisar a correlação entre as variáveis independentes para identificar multicolinearidade.
- **Influência e Outliers:**
  - Leverage: Identificar pontos que têm um grande impacto nos coeficientes de regressão.
  - DFFITS, DFBETAS, Cook's Distance: Medir a influência de observações individuais nos coeficientes do modelo.

- **Ajuste do Modelo:**
  - R^2 e R^2 Ajustado: Medir a proporção da variância explicada pelo modelo.
  - AIC (Akaike Information Criterion) e BIC (Bayesian Information Criterion): Avaliar a qualidade do modelo levando em consideração o número de parâmetros.
- **Validação do Modelo:**
  - Cross-Validation: Usar técnicas de validação cruzada para avaliar a capacidade de generalização do modelo.
  - Análise de Sensibilidade: Testar a robustez do modelo a variações nos dados.

Essas ferramentas e técnicas permitem uma avaliação detalhada do desempenho do modelo de regressão, ajudando a identificar e corrigir problemas que podem comprometer a validade das previsões.

4.1. R-QUADRADO

O coeficiente de determinação, também conhecido como R-quadrado, é uma medida estatística utilizada na análise de regressão linear para avaliar o ajuste do modelo aos dados. Ele fornece uma medida de quanta variabilidade na variável dependente é explicada pelas variáveis independentes incluídas no modelo.

O valor do R-quadrado varia de 0 a 1 e é interpretado como a proporção da variabilidade total na variável dependente que é explicada pelas variáveis independentes no modelo. Um R-quadrado de 1 indica que todas as variações na variável dependente são explicadas pelas variáveis independentes, enquanto um R-quadrado de 0 indica que nenhuma das variações é explicada.

Em termos mais simples, o R-quadrado é uma medida de quão bem o modelo se ajusta aos dados. Quanto maior o valor do R-quadrado, melhor é o ajuste do modelo. No entanto, é importante observar que o R-quadrado por si só não indica se as relações entre as variáveis são significativas ou se o modelo é válido.

O R-quadrado pode ser interpretado como a proporção da variabilidade total na variável dependente que é explicada pelas variáveis independentes incluídas no modelo. Por exemplo, se o R-quadrado for de 0,80, isso significa que 80% da variabilidade na variável dependente é explicada pelas variáveis independentes no modelo, enquanto os 20% restantes são atribuídos a outros fatores não incluídos no modelo.

É importante notar que o R-quadrado tem algumas limitações. Por exemplo, ele tende a aumentar com o número de variáveis independentes incluídas no modelo, mesmo que essas variáveis tenham pouco ou nenhum efeito real nas variações na variável dependente. Além disso, o R-quadrado não indica a direção ou magnitude das relações entre as variáveis, apenas a proporção de variabilidade explicada.

Portanto, o R-quadrado é uma medida importante na análise de regressão linear, pois fornece uma medida de quão bem o modelo se ajusta aos dados e explica a variabilidade na variável dependente. No entanto, é fundamental interpretar o R-quadrado em conjunto com outras estatísticas, realizar análises de diagnóstico e considerar a relevância e validade das variáveis independentes incluídas no modelo.

- Cálculo da Soma Total dos Quadrados (SST):

$$SST = \sum_{i=1}^{n} (yi - \bar{y})^2$$

Onde *yi* são os valores observados e $\bar{y}$ é a média dos valores observados.

- Cálculo da Soma dos Quadrados dos Resíduos (SSE):

$$SSE = \sum_{i=1}^{n} (yi - \hat{y})^2$$

Onde $\hat{y}i$ são os valores previstos pelo modelo.

- Cálculo da Soma dos Quadrados da Regressão (SSR):

$$SSR = \sum_{i=1}^{n} (\hat{y}i - \bar{y})^2$$

- Relação entre SST, SSR e SSE:

$$SST = SSR + SSE$$

- Cálculo do R^2:

$$R^2 = \frac{SSR}{SST}$$

- Cálculo do R^2 usando SSE:

$$R^2 = 1 - \frac{SSE}{SST}$$

4.1.1. EXEMPLO DE CÁLCULO DE R²

Suponha que temos os seguintes dados observados e previstos:

- Dados observados y = [3,4,5,6]
- Dados previstos $\hat{y}$ = [2.8; 4.2; 4.9; 6.1]s
- Calcular a média de y:

$$\bar{y} = \frac{3+4+5+6}{4} = 4.5$$

- Calcular SST:

$$SST = (3-4.5)^2 + (4-4.5)^2 + (5-4.5)^2 + (6-4.5)^2 = 5$$

- Calcular SSE:

$$SSE = (3-2.8)^2 + (4-4.2)^2 + (5-4.9)^2 + (6-6.1)^2 = 0.1$$

- Calcular SSR:

$$SSR = SST - SSE = 5 - 0.1 = 4.9$$

- Calcular R²:

$$R^2 = \frac{SSR}{SST} = \frac{4.9}{5} = 0.98$$

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{0.1}{5} = 0.98$$

Portanto, com SST = 5, SSE = 0.1 e R² = 0.98, podemos confirmar que 98% da variação na variável dependente é explicada pelo modelo de regressão.

4.2. ERRO PADRÃO DE REGRESSÃO

O erro padrão de regressão é uma medida da precisão das estimativas dos coeficientes de regressão em um modelo de regressão linear. Ele quantifica a variabilidade dos estimadores dos coeficientes em torno dos seus valores verdadeiros.

O erro padrão de um coeficiente de regressão (*βi*) é a estimativa da variabilidade do estimador de *βi*. Formalmente, se *bi* é o estimador de *βi*, então o erro padrão de *bi* é dado por:

$$SE(bi)=\sqrt{\frac{\sigma^2}{\sum(xi\ -\ \bar{x})^2}}$$

Onde σ^2 é a variância dos resíduos do modelo e $\sum(xi-\bar{x})^2$ é a soma dos quadrados das diferenças entre as observações da variável independente x e sua média $\bar{x}$.

- **Inferência:** O erro padrão é usado para construir intervalos de confiança e realizar testes de hipóteses sobre os coeficientes de regressão. Um erro padrão menor indica que o coeficiente é estimado com maior precisão.
- **Significância Estatística:** Junto com o valor do coeficiente estimado e a estatística t, o erro padrão ajuda a determinar se o coeficiente é significativamente diferente de zero.

Para um modelo de regressão linear simples:

$$y=\beta 0+\beta 1*x+\varepsilon$$

Onde ε são os erros ou resíduos do modelo.

A variância dos resíduos σ^2 é estimada pela soma dos quadrados dos resíduos (SSR) dividida pelos graus de liberdade (n-2):

$$\hat{\sigma^2}=\frac{SSR}{n-2}\ =\ \frac{\sum(yi-\hat{yi})^2}{n-2}$$

Então, o erro padrão de b1 (coeficiente da variável independente x) é:

$$SE(b1)\ =\ \sqrt{\frac{\hat{\sigma^2}}{\sum(xi-\bar{x})^2}}$$

No software estatístico, os erros padrões dos coeficientes são geralmente apresentados junto com os coeficientes estimados na tabela de regressão. Eles são fundamentais para interpretar a confiabilidade das estimativas dos coeficientes.

5. REGRESSÃO LINEAR MÚLTIPLA

A regressão linear múltipla é uma extensão da regressão linear simples, que permite analisar a relação entre uma variável dependente e várias variáveis independentes simultaneamente. Ao contrário da regressão linear simples, em que apenas uma variável independente é considerada, a regressão linear múltipla leva em conta múltiplas variáveis independentes que podem influenciar na variável dependente.

A equação da regressão linear múltipla pode ser expressa da seguinte forma:

$$Y = \beta 0\, X 0 + \beta 1\, X 1 + \beta 2\, X 2 + \ldots + \beta\, kXk + \epsilon$$

Onde:

- Y é a variável dependente;
- X são as variáveis independentes;
- β são os coeficientes de regressão que representam o efeito das variáveis independentes sobre a variável dependente, necessário calcular cada uma;
- ε é o termo de erro, que representa a parte não explicada pela relação linear entre as variáveis.

A análise de regressão linear múltipla oferece várias vantagens em relação à regressão linear simples, pois permite controlar o efeito de múltiplas variáveis independentes na variável dependente. Além disso, ela pode ser usada para fazer previsões mais precisas e analisar o efeito combinado das variáveis independentes.

No entanto, é necessário tomar cuidado na interpretação dos coeficientes de regressão múltipla, pois eles podem ser influenciados pela presença de multicolinearidade, que é a existência de correlação entre as variáveis independentes. É importante também realizar análises de diagnóstico para verificar a validade dos supostos da regressão linear múltipla, como a normalidade dos resíduos e a homoscedasticidade.

Em resumo, a regressão linear múltipla é uma técnica estatística utilizada para analisar a relação entre uma variável dependente e várias variáveis independentes, permitindo controlar e quantificar o efeito de cada variável independente sobre a variável dependente. É uma ferramenta útil para fazer previsões e entender a relação complexa entre variáveis em diferentes áreas de estudo.

5.1. EXEMPLO REGRESSÃO LINEAR MÚLTIPLA

Suponhamos que queremos prever o preço de casas com base em vários fatores, como área da casa, número de quartos e idade da casa. Temos os seguintes dados de exemplo:

Área (m²)	Número de Quartos	Idade (anos)	Preço (milhares de R$)
150	3	10	400
200	4	5	550
250	4	20	600
300	5	15	700
350	6	2	850

Tabela 2: Exemplo de Regressão Linear Múltipla
Fonte: ChatGPT, 2024

Podemos usar esses dados para ajustar um modelo de regressão linear múltipla. A equação geral da regressão linear múltipla é:

$$Preço = \beta 0 + \beta 1 \cdot Área + \beta 2 \cdot Números\,de\,Quartos + \beta 3 \cdot Idade$$

$$\beta k = \frac{\sum_{i=1}^{n} (Xi - \bar{X}k)(Yi - \bar{Y})}{\sum_{i=1}^{n} (Xi - \bar{X}k)}$$

- β0 (Intercepto) = 76.5625
- β1 (Área) = 1.46875
- β2 (Números de Quartos) = 43.75
- β3 (Idade) = -1.5625

Para prever o preço de uma casa com 280 m² de área, 5 quartos e 8 anos de idade, podemos usar a equação ajustada:

$$Preço = 76.5625 + (1.46875 \cdot 280) + (43.75 \cdot 5) + (-1.5625 \cdot 8)$$

$$Preço = R\$\,694.06$$

5.2. COEFICIENTE DE REGRESSÕES PARCIAIS

O coeficiente de regressões parciais é uma medida estatística utilizada na análise de regressão linear múltipla. Ele fornece uma medida do efeito individual de uma variável independente na variável dependente, controlando o efeito das outras variáveis independentes no modelo.

Em uma regressão linear múltipla, cada variável independente contribui para a previsão da variável dependente. No entanto, essas contribuições podem ser confundidas pela presença de correlação entre as variáveis independentes, chamada de multicolinearidade. O coeficiente de regressões parciais ajuda a resolver esse problema, medindo o efeito isolado de cada variável independente na variável dependente, mantendo as outras variáveis constantes.

O coeficiente de regressões parciais é calculado como a diferença entre os coeficientes de regressão estimados quando uma determinada variável independente é incluída no modelo e quando essa variável é excluída. Essa diferença representa o efeito único da variável independente na variável dependente, controlando o efeito das outras variáveis independentes.

O coeficiente de regressões parciais fornece informações valiosas sobre o efeito individual de cada variável independente na variável dependente, permitindo analisar o impacto específico de cada variável controlando as demais. Isso é especialmente útil quando há correlação entre as variáveis independentes e é necessário identificar o efeito exclusivo de cada variável.

É importante ressaltar que o coeficiente de regressões parciais não deve ser interpretado isoladamente, mas sim em conjunto com os demais coeficientes de regressão estimados no modelo. Além disso, é fundamental realizar análises de diagnóstico para verificar a validade dos supostos da regressão linear múltipla, como a normalidade dos resíduos e a homogeneidade de variância.

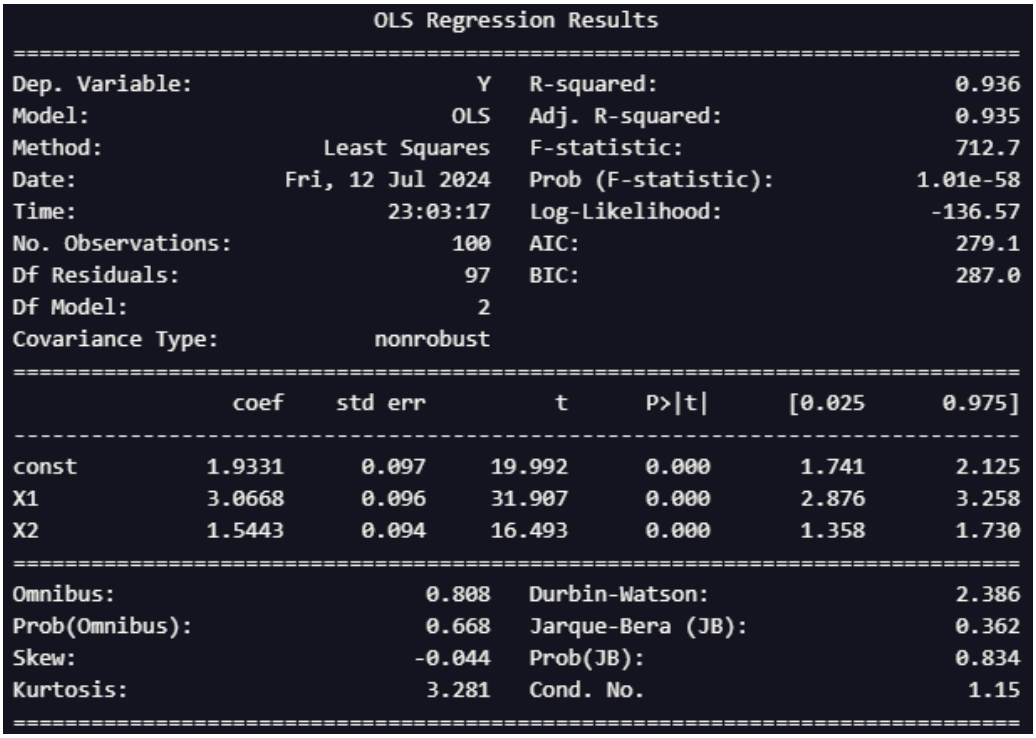
Em resumo, o coeficiente de regressões parciais é uma medida utilizada na análise de regressão linear múltipla que mede o efeito individual de uma variável independente na variável dependente, controlando o efeito das outras variáveis independentes no modelo. Ele ajuda a resolver o problema da multicolinearidade e fornece informações valiosas sobre o impacto específico de cada variável independente.

5.2.1. EXEMPLO DE COEFICIENTE DE REGRESSÕES PARCIAIS

Suponha que temos um conjunto de regressão parcial de X_1 em Y, mantendo X_2 constante. Implementando em Python:

```python
# Importando bibliotecas necessárias
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Gerando dados fictícios
np.random.seed(0)  # Para reprodutibilidade

# Variáveis independentes
X1 = np.random.normal(size=100)
X2 = np.random.normal(size=100)

# Variável dependente (Y) com relação linear com X1 e X2, e um termo de erro
Y = 2 + 3*X1 + 1.5*X2 + np.random.normal(size=100)

# Criando um DataFrame
data = pd.DataFrame({'Y': Y, 'X1': X1, 'X2': X2})

# Adicionando uma constante ao modelo (termo intercepto)
X = sm.add_constant(data[['X1', 'X2']])

# Ajustando o modelo de regressão múltipla
model = sm.OLS(data['Y'], X).fit()

# Exibindo o resumo dos resultados
print(model.summary())
```

Figura 3: Implementação de Regressão Linear Múltipla no Python
Fonte: ChatGPT, 2024



OLS Regression Results

Dep. Variable:	Y	R-squared:	0.936
Model:	OLS	Adj. R-squared:	0.935
Method:	Least Squares	F-statistic:	712.7
Date:	Fri, 12 Jul 2024	Prob (F-statistic):	1.01e-58
Time:	23:03:17	Log-Likelihood:	-136.57
No. Observations:	100	AIC:	279.1
Df Residuals:	97	BIC:	287.0
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.9331	0.097	19.992	0.000	1.741	2.125
X1	3.0668	0.096	31.907	0.000	2.876	3.258
X2	1.5443	0.094	16.493	0.000	1.358	1.730

Omnibus:	0.808	Durbin-Watson:	2.386
Prob(Omnibus):	0.668	Jarque-Bera (JB):	0.362
Skew:	-0.044	Prob(JB):	0.834
Kurtosis:	3.281	Cond. No.	1.15

Figura 4: Resumo do Resultado da Regressão Linear Múltipla
Fonte: ChatGPT, 2024

O resumo dos resultados da regressão múltipla fornecerá várias informações importantes:

- Para cada acréscimo no valor de X1, o valor de Y aumenta em 3.0668.
- Para cada acréscimo no valor de X2, o valor de Y aumenta 1.5443.
- Usando os coeficientes do resultado da regressão, podemos escrever a equação de regressão linear múltipla estimada:

$$Y = 1.9331 + (3.0668 \cdot X1) - (1.5443 \cdot X2)$$

5.3. TESTES DE SIGNIFICÂNCIA

Os testes de significância da regressão linear múltipla são utilizados para avaliar a relevância estatística das variáveis independentes incluídas no modelo de regressão. Esses testes ajudam a determinar se as variáveis independentes têm um efeito significativo na variável dependente.

Existem diferentes testes de significância que podem ser aplicados na regressão linear múltipla. Os principais testes incluem:

• **Teste F para Significância Global do Modelo:** O teste F é usado para testar a significância conjunta das variáveis independentes como um todo. Ele compara o ajuste do modelo completo (com todas as variáveis independentes) com um modelo restrito (sem as variáveis independentes) e avalia se a inclusão das variáveis independentes melhora significativamente o modelo. Um valor de p associado ao teste F menor que um valor de significância pré-determinado (por exemplo, 0.05) indica que o modelo é estatisticamente significativo.

- $H_0: \beta_1 = \beta_2 = ... = \beta_k = 0$ (nenhuma variável independente é significativa.
- H_1: pelo menos um $\beta_i \neq 0$ (pelo menos uma variável independente é significativa).

A fórmula para o cálculo do teste F na regressão linear múltipla é calculado da seguinte forma:

$$F = \frac{\frac{SSR}{k}}{\frac{SSE}{(n-k-1)}}$$

Onde:

- *SSR* é a soma dos quadrados de regressão, SSE é a soma dos quadrados dos erros.
- *SSE* é a soma dos quadrados dos erros.
- k é o número de variáveis independentes.
- n é o número de observações.

Esse teste avalia se o modelo de regressão como um todo é estatisticamente significativo, comparando a regressão com o modelo nulo (sem variáveis independentes). Se o valor de F for maior que o valor crítico correspondente, podemos rejeitar a hipótese nula e concluir que o modelo é significativo.

• **Teste t para coeficientes individuais:** Os testes t são usados para testar a significância individual de cada variável independente no modelo. Eles avaliam se cada variável independente tem um efeito significativo na variável dependente, controlando os outros fatores no modelo. Cada teste t fornece um valor de p que indica a probabilidade de obter os resultados observados, supondo que a variável independente não tenha um efeito significativo. Valores de p menores que o valor de significância pré-determinado (por exemplo, 0.05) indicam que a variável independente é estatisticamente significativa.

A fórmula para calcular o teste t individual na regressão linear múltipla é a seguinte:

$$t = \frac{\hat{\beta}_i}{SE(\hat{\beta}_i)}$$

Onde:

- $\hat{\beta}_i$ é o coeficiente de regressão da variável independente i.
- $SE(\hat{\beta}i_i)$ é o erro padrão do coeficiente.

O teste t é utilizado para avaliar se o coeficiente de regressão de uma variável independente específica é estatisticamente significativo. Se o valor absoluto do t for maior que o valor crítico correspondente (com base no nível de significância desejado e nos graus de liberdade do modelo), podemos rejeitar a hipótese nula de que o coeficiente de regressão é igual a zero e concluir que ele é estatisticamente diferente de zero.

• **Estatísticas de ajuste do modelo:** Além dos testes de significância, outras estatísticas são comumente usadas na regressão linear múltipla para avaliar o ajuste e a qualidade do modelo. Entre elas estão o coeficiente de determinação (R-quadrado), o

erro padrão das estimativas (SCE) e o critério de informação Akaike (AIC) ou critério de informação bayesiano (BIC).

OLS Regression Results

Dep. Variable:	Y	R-squared:	0.936
Model:	OLS	Adj. R-squared:	0.935
Method:	Least Squares	F-statistic:	712.7
Date:	Fri, 12 Jul 2024	Prob (F-statistic):	1.01e-58
Time:	23:03:17	Log-Likelihood:	-136.57
No. Observations:	100	AIC:	279.1
Df Residuals:	97	BIC:	287.0
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.9331	0.097	19.992	0.000	1.741	2.125
X1	3.0668	0.096	31.907	0.000	2.876	3.258
X2	1.5443	0.094	16.493	0.000	1.358	1.730

Omnibus:	0.808	Durbin-Watson:	2.386
Prob(Omnibus):	0.668	Jarque-Bera (JB):	0.362
Skew:	-0.044	Prob(JB):	0.834
Kurtosis:	3.281	Cond. No.	1.15

Figura 5: Resumo do Resultado da Regressão Linear Múltipla
Fonte: ChatGPT, 2024

- **Coeficientes individuais:**
  - O coeficiente para X1 é 3.0668 com um p-valor de 0.000, indicando que X1 é significativamente diferente de zero.
  - O coeficiente para X2 é 1.5443 com um p-valor de 0.000, indicando que X2 também é significativamente diferente de zero.
- **Modelo Global:**
  - A estatística F é 712.7 com um p-valor de 1.01e-58, indicando que o modelo é globalmente significativo.

Esses resultados mostram que tanto X1 quanto X2 têm um impacto significativo sobre Y e que o modelo como um todo é significativo.

5.4. PRESSUPOSTOS BÁSICOS DA REGRESSÃO LINEAR

Os pressupostos básicos da regressão linear são fundamentais para garantir a validade dos resultados obtidos. Eles incluem:

- **Linearidade:** A relação entre as variáveis independentes e a variável dependente deve ser linear. Isso significa que a mudança na variável dependente é proporcional à mudança na variável independente.
- **Independência dos Erros:** Os resíduos (erros) devem ser independentes entre si. Isso implica que não deve haver correlação entre os erros. Em séries temporais, isso é conhecido como ausência de autocorrelação.
- **Homoscedasticidade:** A variância dos erros deve ser constante ao longo dos valores preditos. Em outras palavras, a dispersão dos resíduos deve ser a mesma para todos os valores de predição da variável dependente. O contrário é chamado de heteroscedasticidade.
- **Normalidade dos Erros:** Os resíduos da regressão devem seguir uma distribuição normal. Este pressuposto é importante principalmente para a construção de intervalos de confiança e testes de hipótese.
- **Ausência de Multicolinearidade:** As variáveis independentes não devem ser altamente correlacionadas entre si. A multicolinearidade pode inflar as variâncias das estimativas dos coeficientes, tornando-os instáveis e difíceis de interpretar.
- **Variáveis Independentes Exatamente Medidas:** Assume-se que as variáveis independentes são medidas sem erro. Se houver erro nas variáveis independentes, os coeficientes estimados podem ser enviesados.

Estes pressupostos são importantes para a validade das inferências que podem ser feitas a partir do modelo de regressão linear. Quando algum desses pressupostos é violado, as estimativas dos coeficientes de regressão podem ser enviesadas ou ineficientes, e os testes de hipótese podem não ser válidos.

5.5. PROBLEMAS COMUNS

• **Violação da Linearidade:** pode ocorrer quando a relação entre a variável dependente e uma ou mais variáveis independentes não é linear. Isso pode ser detectado examinando gráficos de resíduos. Soluções incluem a transformação das variáveis ou o uso de modelos não lineares.

• **Autocorrelação dos Erros:** erros que não são independentes uns dos outros, comum em dados de séries temporais. A autocorrelação pode ser detectada usando o teste de Durbin-Watson. Modelos ARIMA ou a adição de termos autorregressivos podem ser utilizados para corrigir esse problema.

• **Heteroscedasticidade:** quando a variância dos erros não é constante, os intervalos de confiança e testes de hipóteses podem ser inválidos. Testes como o teste de Breusch-Pagan podem detectar heteroscedasticidade. Transformações como a transformação logarítmica ou modelos ponderados podem ser utilizados para corrigir esse problema.

• **Erros Não Normais:** a não normalidade dos resíduos pode afetar a validade das inferências estatísticas. Gráficos de probabilidade normal ou testes como o teste de Shapiro-Wilk podem ser usados para verificar a normalidade. Transformações de dados ou técnicas robustas podem ser aplicadas para lidar com essa violação.

• **Multicolinearidade:** pode ser detectada examinando a matriz de correlação das variáveis independentes ou calculando o fator de inflação da variância (VIF). Para mitigar a multicolinearidade, pode-se remover variáveis altamente correlacionadas, combinar variáveis ou usar técnicas de regularização como Ridge ou Lasso.

5.5.1. DIAGNÓSTICO E CORREÇÃO

• **Análise de Resíduos:** gráficos de resíduos vs valores ajustados podem ajudar a identificar problemas de linearidade e homoscedasticidade.

• **Teste de Durbin-Watson:** para detectar autocorrelação.

• **Gráficos de Probabilidade Normal:** para verificar a normalidade dos resíduos.

• **Fatores de Inflação da Variância (VIF):** para diagnosticar multicolinearidade.

6. APLICAÇÕES AVANÇADAS DA REGRESSÃO LINEAR

Em aplicações avançadas, ela pode ser usada para resolver problemas complexos e em combinações com outras técnicas para melhorar a precisão e a utilidade das previsões.

6.1. REGRESSÃO POLINOMIAL

É uma forma de regressão linear múltipla na qual a relação entre a variável dependente e a variável independente é modelada como um polinômio de ordem n. Em outras palavras, em vez de uma relação linear simples entre as variáveis, a regressão polinomial permite modelar uma relação não linear, incluindo termos elevados a potências maiores.

A forma geral de um modelo de regressão polinomial de ordem n é:

$$y = \beta_0, \beta_1 x, \beta_2 x^2, \beta_3 x^3, \ldots, \beta_n x^n + \epsilon$$

Onde y é a variável dependente, x é a variável independente, β_n são os coeficientes do modelo e o ϵ é o termo de erro.

- **Passos para realizar a regressão polinomial:**
  - **Escolher a ordem do polinômio:** a ordem n do polinômio deve ser escolhida com cuidado. Ordens muito altas podem levar ao overfitting, onde o modelo se ajusta demais aos dados de treinamento, mas não generaliza bem para novos dados.
  - **Transformação de dados:** criar termos de entrada elevados às potências de 2 até n. Por exemplo, para um polinômio de ordem 2, incluir x^2 como uma nova variável independente no modelo.
  - **Ajuste do modelo:** usar métodos de mínimos quadrados para ajustar o modelo polinomial aos dados.
  - **Avaliação do modelo:** avaliar o modelo ajustado usando métricas como R-quadrado, RMSE (Root Mean Square Error) e gráficos de resíduos para verificar a qualidade do ajuste e detectar possíveis problemas de overfitting ou underfitting.
- **Vantagens**
  - Captura relações não lineares entre as variáveis.
  - Flexível, pois permite ajustar a ordem do polinômio para melhor ajuste.

- **Desvantagens:**
  - Pode levar ao overfitting se a ordem do polinômio for muito alta.
  - A interpretação dos coeficientes do modelo se torna mais complexa à medida que a ordem aumenta.
  - Pode ser sensível a outliers.
- **Aplicações:**
  - Economia: modelar a relação entre variáveis econômicas que não seguem uma tendência linear simples.
  - Engenharia: análise de dados experimentais onde as relações entre variáveis podem ser não lineares.
  - Ciências Naturais: modelar fenômenos naturais que seguem relações curvas, como crescimento populacional e reações químicas.

6.1.1. Implementação em Python

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Gerar dados de exemplo
np.random.seed(0)
x = np.sort(np.random.rand(100, 1) * 10, axis=0)
y = 2 - 1.5 * x + 0.5 * x**2 + np.random.randn(100, 1)

# Transformar dados para incluir termos polinomiais (por exemplo, ordem 2)
poly_features = PolynomialFeatures(degree=2)
x_poly = poly_features.fit_transform(x)

# Ajustar o modelo de regressão linear nos dados transformados
model = LinearRegression()
model.fit(x_poly, y)

# Fazer previsões
y_poly_pred = model.predict(x_poly)

# Visualizar os resultados
plt.scatter(x, y, color='blue', label='Dados de Treinamento')
plt.plot(x, y_poly_pred, color='red', label='Modelo Polinomial (ordem 2)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

Figura 6: Implementação de Regressão Polinomial em Python
Fonte: ChatGPT, 2024

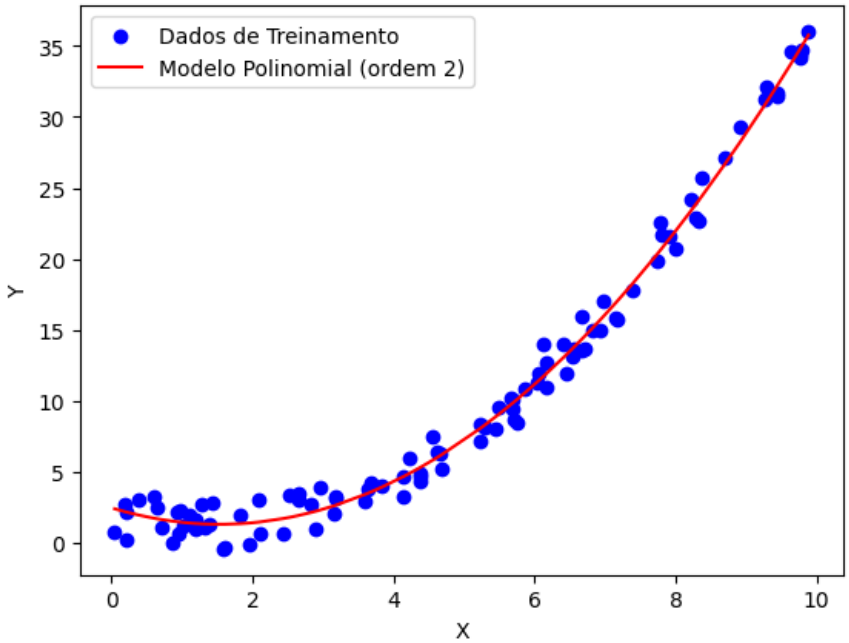


Figura 7: Gráfico de Regressão Polinomial
Fonte: ChatGPT, 2024

6.2. REGRESSÃO COM INTERAÇÕES

A regressão com interações é uma extensão da regressão linear que inclui termos de interação entre variáveis independentes no modelo. Esses termos de interação permitem que o efeito de uma variável independente na variável dependente dependa do valor de outra variável independente. Isso é particularmente útil quando há suspeita de que as variáveis independentes não atuam de maneira independente umas das outras.

O modelo de regressão com interação para duas variáveis independentes seria:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 (x_1 \cdot x_2) + \epsilon$$

O coeficiente do termo de interação β_3 indica o quanto o efeito de x_1 na variável dependente y varia dependendo do valor de x_2 e vice-versa.

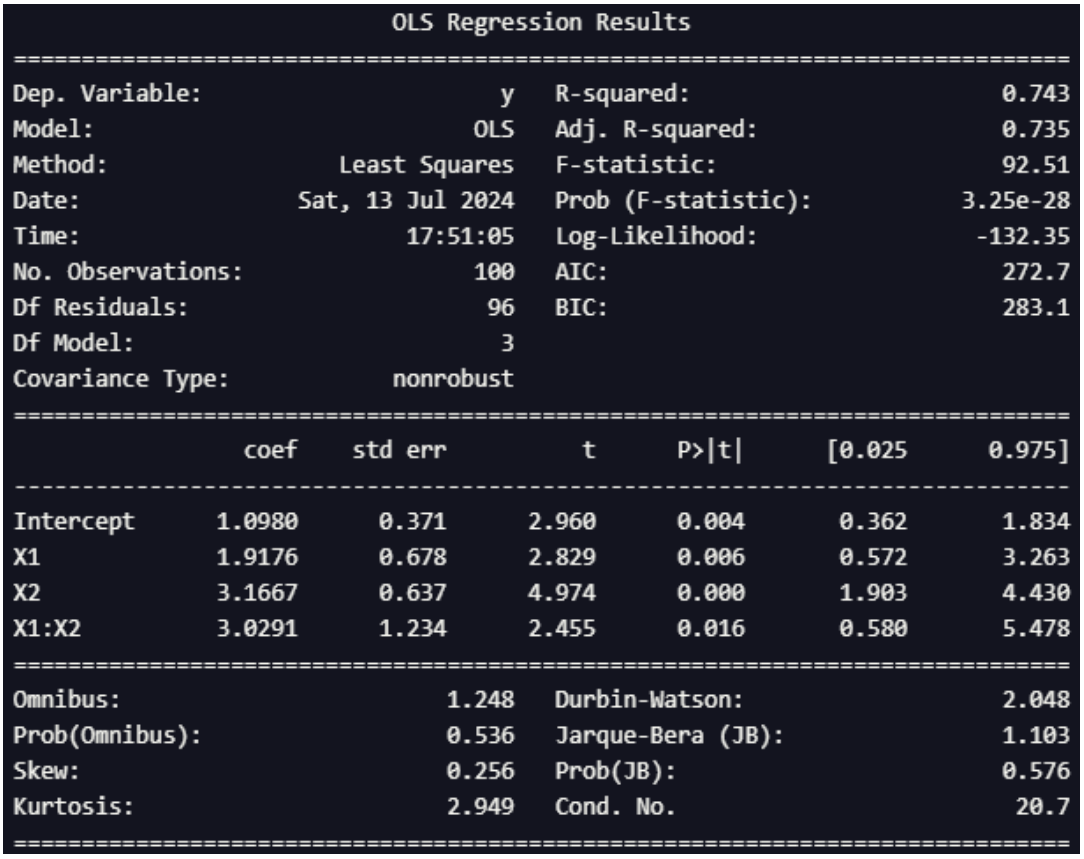
- **Passos para implementar Regressão com Interações:**
  - Identificar as variáveis de interação: escolher quais variáveis independentes podem ter interações significativas.
  - Incluir termos de interação no modelo: criar termos que representam o produto das variáveis de interação.
  - Ajustar o modelo: usar métodos de regressão linear para ajustar o modelo que inclui os termos de interação.
  - Avaliar o modelo: avaliar a significância dos termos de interação e o ajuste geral do modelo usando métricas apropriadas e testes estatísticos.
- **Vantagens:**
  - Captura efeitos combinados de variáveis independentes, proporcionando uma compreensão mais rica das relações entre as variáveis.
  - Pode melhorar o ajuste do modelo ao considerar interações significativas.
- **Desvantagens:**
  - A interpretação dos coeficientes se torna mais complexa com a inclusão de termos de interação.
  - Pode levar ao overfitting se muitos termos de interação forem incluídos sem justificativa teórica ou empírica.

- **Aplicações:**
  - Psicologia: analisar como diferentes fatores psicossociais interagem para afetar o comportamento.
  - Economia: modelar como diferentes políticas econômicas interagem para impactar indicadores econômicos.
  - Medicina: investigar como diferentes tratamentos combinados afetam a saúde dos pacientes.

6.2.1. IMPLEMENTAÇÃO EM PYTHON

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Gerar dados de exemplo
np.random.seed(0)
x1 = np.random.rand(100)
x2 = np.random.rand(100)
y = 1 + 2 * x1 + 3 * x2 + 4 * x1 * x2 + np.random.randn(100)

# Criar DataFrame
data = pd.DataFrame({'x1': x1, 'x2': x2, 'y': y})

# Ajustar o modelo de regressão com interação
model = ols('y ~ x1 * x2', data=data).fit()

# Resumo do modelo
print(model.summary())
```

Figura 8: Implementação Regressão com Interações em Python
Fonte: ChatGPT, 2024



OLS Regression Results

Dep. Variable:	y	R-squared:	0.743
Model:	OLS	Adj. R-squared:	0.735
Method:	Least Squares	F-statistic:	92.51
Date:	Sat, 13 Jul 2024	Prob (F-statistic):	3.25e-28
Time:	17:51:05	Log-Likelihood:	-132.35
No. Observations:	100	AIC:	272.7
Df Residuals:	96	BIC:	283.1
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.0980	0.371	2.960	0.004	0.362	1.834
X1	1.9176	0.678	2.829	0.006	0.572	3.263
X2	3.1667	0.637	4.974	0.000	1.903	4.430
X1:X2	3.0291	1.234	2.455	0.016	0.580	5.478

Omnibus:	1.248	Durbin-Watson:	2.048
Prob(Omnibus):	0.536	Jarque-Bera (JB):	1.103
Skew:	0.256	Prob(JB):	0.576
Kurtosis:	2.949	Cond. No.	20.7

Figura 9: Resumo do Resultado da Regressão com Interações
Fonte: ChatGPT, 2024

Um coeficiente de interação significativo sugere que há uma interação entre x_1 e x_2, ou seja, o efeito de x_1 em y depende do valor de x_2.

6.3. REGULARIZAÇÃO

A regularização em regressão linear é uma técnica usada para prevenir overfitting ao introduzir um termo de penalização no cálculo dos coeficientes do modelo. Isso é particularmente útil quando se trabalha com conjuntos de dados que possuem um grande número de variáveis independentes ou multicolinearidade. As duas formas mais comuns de regularização são a Regressão Ridge e a Lasso.

6.3.1. REGRESSÃO RIDGE (REGULARIZAÇÃO L2)

A Regressão Ridge adiciona um termo de penalização baseado na soma dos quadrados dos coeficientes (também conhecido como penalização L2) à função de custo da regressão linear. O objetivo é minimizar a seguinte função:

$$J(\beta) = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Onde:

- λ é o parâmetro de regularização que controla a magnitude da penalização. Um valor maior de λ aumenta a penalização, reduzindo os coeficientes e prevenindo overfitting.
- β_j são os coeficientes do modelo.

6.3.2. REGRESSÃO LASSO (REGULARIZAÇÃO L1)

A Regressão Lasso (Least Absolute Shrinkage and Selection Operator) adiciona um termo de penalização baseado na soma dos valores absolutos dos coeficientes (penalização L1) à função de custo:

$$J(\beta) = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

A Regressão Lasso tem a propriedade adicional de poder reduzir alguns coeficientes exatamente a zero, efetivamente selecionando um subconjunto de variáveis. Isso é útil para a seleção de variáveis em modelos de alta dimensão.

6.3.3. ELASTIC NET

O Elastic Net é uma combinação de Ridge e Lasso, utilizando tanto penalizações L1 quanto L2:

$$J(\beta)=\sum_{i=1}^{n}(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij})^2+\lambda_1\sum_{j=1}^{p}|\beta_j|+\lambda_2\sum_{j=1}^{p}\beta_j^2$$

6.3.4. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Gerar dados de exemplo
np.random.seed(0)
X = np.random.rand(100, 10)
y = 5 + np.dot(X, np.array([1.5, -2, 0, 3, 0, -1, 0, 0, 2, -1])) + np.random.randn(100)

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Ajustar o modelo de Regressão Ridge
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)

# Ajustar o modelo de Regressão Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)

# Avaliar o desempenho dos modelos
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)

print(f'MSE Regressão Ridge: {mse_ridge}')
print(f'MSE Regressão Lasso: {mse_lasso}')

# Visualizar os coeficientes dos modelos
plt.figure(figsize=(10, 6))
plt.plot(ridge.coef_, label='Ridge Coefficients')
plt.plot(lasso.coef_, label='Lasso Coefficients')
plt.legend()
plt.show()
```

Figura 10: Implementação Regressão Ridge e Lasso em Python
Fonte: ChatGPT, 2024

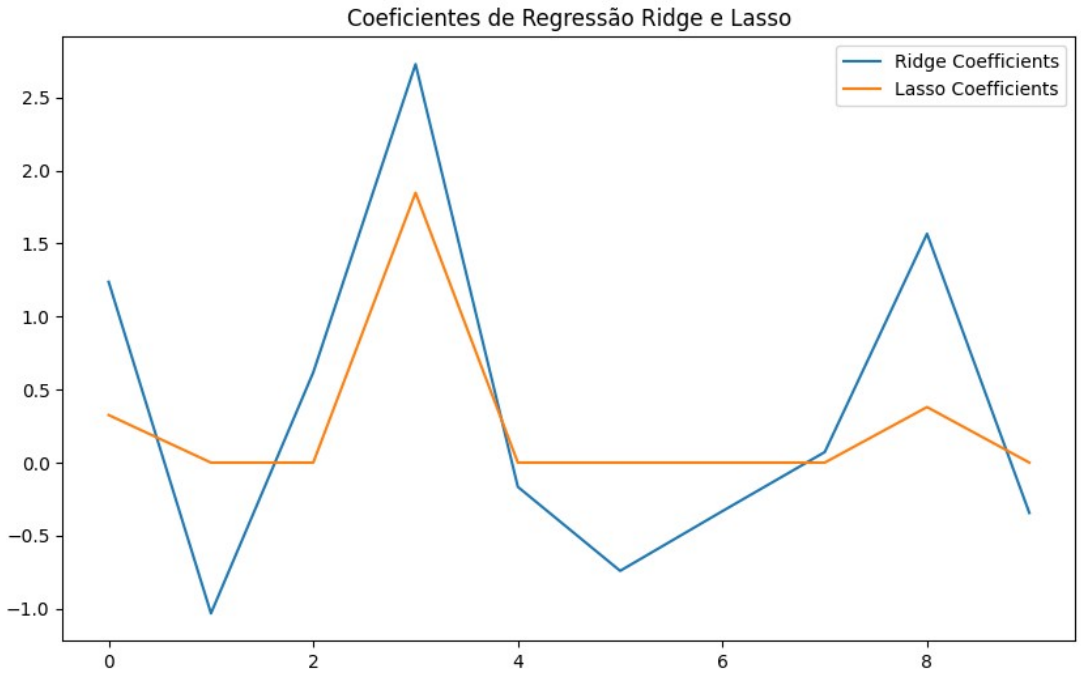


Figura 11: Gráfico dos coeficientes dos modelos da Regressão Ridge e Lasso
Fonte: Autor

6.4. REGRESSÃO PONDERADA

A regressão ponderada, também conhecida como regressão de mínimos quadrados ponderados (WLS – Weighted Least Squares), é uma técnica de regressão linear onde diferentes observações têm diferentes níveis de importância (pesos). É útil quando a variância dos erros não é constante (heteroscedasticidade) ou quando algumas observações são mais confiáveis ou precisas que outras.

6.4.1. Características da Regressão Ponderada:

- **Pesos:**
  - Cada observação i tem um peso w_i, que reflete sua importância ou a precisão da observação.
  - A função de custo minimizada na regressão ponderada é:

$$J(\beta)=\sum_{i=1}^{n} w_i\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2$$

  Onde w_i são os pesos, y_i são as observações da variável dependente, x_{ij} são as observações das variáveis independentes, e β são os coeficientes do modelo.

- **Heteroscedasticidade:**
  - A regressão ponderada é particularmente útil quando a variância dos erros não é constante. Os pesos podem escolhidos como o inverso da variância dos erros, de forma que as observações com menor variância tenham maior peso.

- **Observações influentes:**
  - Se algumas observações são mais confiáveis ou precisas, podem ser atribuídos pesos maiores a essas observações.

- **Vantagens:**
  - Heteroscedasticidade: pode lidar com a heteroscedasticidade, melhorando a precisão dos coeficientes estimados.
  - Observações influentes: permite considerar a variabilidade na precisão das observações, dando maior importância a observações mais confiáveis.
  - Flexibilidade: pode ser usada em uma variedade de contextos onde a regressão linear simples falha devido à variabilidade na qualidade dos dados.

- **Desvantagens:**

  • Escolha dos Pesos: requer uma escolha criteriosa dos pesos. A seleção inadequada dos pesos pode resultar em um modelo enviesado.

  • Complexidade Computacional: pode ser mais complexa computacionalmente do que a regressão linear simples, especialmente para grandes conjuntos de dados.

- **Aplicações:**

  • Econometria: lidar com dados econômicos que apresentam heteroscedasticidade, como renda versus consumo.

  • Engenharia: análise de dados experimentais onde algumas medições são mais precisas que outras.

  • Medicina: modelagem de dados clínicos onde diferentes estudos ou medições têm diferentes níveis de precisão.

6.4.2. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
import statsmodels.api as sm

# Gerar dados de exemplo
np.random.seed(0)
x = np.random.rand(100)
y = 2 + 3 * x + np.random.randn(100)

# Pesos (por exemplo, variância inversa)
w = 1 / (np.random.rand(100) + 0.1)

# Adicionar uma constante (termo de intercepto)
X = sm.add_constant(x)

# Ajustar o modelo de regressão ponderada
model = sm.WLS(y, X, weights=w)
results = model.fit()

# Resumo do modelo
print(results.summary())
```

Figura 12: Implementação de Regressão Ponderada em Python
Fonte: ChatGPT, 2024

6.5. REGRESSÃO ROBUSTA

A regressão robusta é uma técnica usada para modelar dados que são influenciados por outliers ou apresentam desvios da normalidade que podem distorcer os resultados de uma regressão linear clássica. A regressão robusta é projetada para ser menos sensível a essas anomalias, proporcionando estimativas mais confiáveis e robustas.

- **Vantagens:**
  - Robustez a Outliers: menor sensibilidade a outliers, proporcionando estimativas mais confiáveis.
  - Flexibilidade: pode ser aplicada em uma variedade de contextos onde os dados apresentam desvios da normalidade.
- **Desvantagens:**
  - Complexidade Computacional: pode ser mais computacionalmente intensiva do que a regressão linear clássica.
  - Interpretação: a interpretação dos coeficientes pode ser mais complexa, dependendo do método de regressão robusta utilizado.
- **Aplicações:**
  - Economia: modelagem de dados econômicos com outliers devido a eventos extremos.
  - Engenharia: análise de dados experimentais que podem conter medições anômalas.
  - Ciências Sociais: estudo de dados comportamentais onde outliers podem surgir devido a respostas extremas.

6.5.1. PRINCIPAIS MÉTODOS DE REGRESSÃO ROBUSTA

6.5.1.1. MÉTODO DOS MÍNIMOS QUADRADOS PONDERADOS ITERATIVOS

O método dos mínimos quadrados ponderados iterativos (IRLS – Iteratively Reweighted Least Squares) é uma técnica utilizada para resolver problemas de regressão, especialmente quando os dados possuem heterocedasticidade ou outliers. Esse método ajusta modelos lineares ao reponderar iterativamente as observações, atribuindo pesos diferentes a cada ponto de dados em cada iteração.

- **Procedimento IRLS:**
  - Iniciação: começa com uma estimativa inicial dos parâmetros do modelo.
  - Cálculo dos Resíduos: calcula os resíduos (a diferença entre os valores observados e os valores ajustados pelo modelo).
  - Cálculo dos Pesos: calcula os pesos com base nos resíduos. Diferentes esquemas de pesos podem ser utilizados, como o inverso do módulo dos resíduos ou funções mais complexas.
  - Reajuste do Modelo: ajusta o modelo usando os mínimos quadrados ponderados com os pesos calculados na etapa anterior.
  - Iteração: repete os passos 2 a 4 até que os parâmetros do modelo convergam, ou seja, até que as mudanças nos parâmetros entre as iterações sejam menores que um limiar pré-estabelecido.
- **Vantagens:**
  - Robustez a Outliers: a ponderação pode minimizar a influência de outliers nos parâmetros do modelo.
  - Ajuste para Heterocedasticidade: pode lidar melhor com variância não constante dos resíduos.
- **Desvantagens:**
  - Complexidade Computacional: pode ser computacionalmente intensivo devido à necessidade de múltiplas iterações.
  - Escolha dos Pesos: a performance do método depende da escolha adequada da função de pesos.

6.5.1.1.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
from scipy.linalg import lstsq

def irls(X, y, tol=1e-6, max_iter=100):
    m, n = X.shape
    w = np.ones(m)
    beta = np.zeros(n)

    for i in range(max_iter):
        W = np.diag(w)
        X_w = np.dot(W, X)
        y_w = np.dot(W, y)

        beta_new, _, _, _ = lstsq(X_w, y_w)
        residuals = y - np.dot(X, beta_new)

        if np.linalg.norm(beta_new - beta) < tol:
            break

        beta = beta_new
        w = 1 / np.maximum(np.abs(residuals), tol)

    return beta

# Exemplo de uso:
X = np.array([[1, 1], [1, 2], [1, 3], [1, 4], [1, 5]])
y = np.array([1, 2, 1.3, 3.75, 2.25])
beta = irls(X, y)
print(beta)
```

Figura 13: Implementação do IRLS em Python
Fonte: ChatGPT, 2024

Este exemplo ajusta um modelo linear simples aos dados fornecidos usando IRLS. O vetor “beta” resultante contém os coeficientes do modelo ajustado.

6.5.1.2. REGRESSÃO DE M-ESTIMADORES

A regressão de M-estimadores é uma técnica robusta de regressão que visa minimizar a influência de outliers e outros tipos de desvios que podem afetar negativamente a precisão dos modelos de regressão tradicionais. Diferente dos mínimos quadrados ordinários (OLS), que minimizam uma função de perda mais geral, que pode ser menos sensível a grandes desvios nos dados.

- **Procedimento de M-Estimadores:**
  - Definição da Função de Perda: escolha uma função de perda $\rho(u)$ que seja menos influenciada por outliers. Algumas funções comuns incluem a função Huber, a função bisquare, e outras funções robustas.
  - Estimativa Inicial: comece com uma estimativa inicial dos parâmetros do modelo.
  - Cálculo dos Resíduos: calcule os resíduos (diferença entre os valores observados e os valores ajustados pelo modelo).
  - Reponderação dos Resíduos: use uma função de ponderação derivada da função de perda para atribuir pesos aos resíduos. O peso geralmente diminui à medida que o resíduo aumenta.
  - Reajuste do Modelo: ajuste o modelo ponderado para minimizar a soma ponderada da função de perda dos resíduos.
  - Iteração: repita os passos 3 a 5 até que os parâmetros do modelo convergam, ou seja, até que as mudanças nos parâmetros entre as iterações sejam menores que um limiar pré-estabelecido.
- **Funções de Perda Comuns:**
  - Função Huber: combina mínimos quadrados para pequenos resíduos e mínimos absolutos para grandes resíduos.
  - Função Bisquare (ou Tukey's biweight): dá peso zero a resíduos grandes, reduzindo sua influência.
- **Vantagens:**
  - Robustez a outliers: minimiza a influência de outliers nos parâmetros do modelo.

  - Flexibilidade: diferentes funções de perda podem ser escolhidas para diferentes tipos de problemas.

- **Desvantagens:**
  - Complexidade Computacional: pode ser mais computacionalmente intensivo do que os mínimos quadrados ordinários.
  - Escolha da Função de Perda: a performance do modelo depende da escolha adequada da função de perda.

6.5.1.2.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np

# Calcula a perda Huber para os resíduos. A perda Huber combina a perda
# quadrática para resíduos pequenos e a perda linear para resíduos grandes.
def huber_loss(residuals, delta=1.0):
    return np.where(np.abs(residuals) <= delta,
                    0.5 * residuals**2,
                    delta * (np.abs(residuals) - 0.5 * delta))

# Calula os pesos de acordo com a função de perda Huber. Para resíduos pequenos,
# o peso é 1, para resíduos grandes, o peso diminui inversamente proporcional ao
# tamanho do resíduo.
def huber_weight(residuals, delta=1.0):
    return np.where(np.abs(residuals) <= delta,
                    1,
                    delta / np.abs(residuals))

# Implementa o algoritmo iterativo de M-estimadores:
# - inicializa os coeficientes beta como zeros;
# - calcula os resíduos e os pesos usando a função de perda Huber;
# - ajusta o modelo ponderado utilizando "np.linalg.lstsq" para resolver o
#   sistema de equações linear ponderado;
# - itera até que a mudança nos coeficientes seja menor que um limite de
#  tolerância "tol"
def m_estimator(X, y, delta=1.0, tol=1e-6, max_iter=100):
    m, n = X.shape
    beta = np.zeros(n)

    for i in range(max_iter):
        residuals = y - np.dot(X, beta)
        weights = huber_weight(residuals, delta)
        W = np.diag(weights)
        X_w = np.dot(W, X)
        y_w = np.dot(W, y)

        beta_new, _, _, _ = np.linalg.lstsq(X_w, y_w, rcond=None)

        if np.linalg.norm(beta_new - beta) < tol:
            break

        beta = beta_new

    return beta

# Exemplo de uso:
X = np.array([[1, 1], [1, 2], [1, 3], [1, 4], [1, 5]])
y = np.array([1, 2, 1.3, 3.75, 2.25])
beta = m_estimator(X, y)
print(beta)
```

Figura 14: Implementação de Regressão de M-Estimadores em Python
Fonte: ChatGPT, 2024

6.5.1.3. RANSAC (RANDOM SAMPLE CONSENSUS)

RANSAC (Random Sample Consensus) é um algoritmo iterativo utilizado para estimar os parâmetros de um modelo a partir de um conjunto de dados que contém outliers. É particularmente útil em problemas de visão computacional e processamento de imagens, onde os dados podem estar contaminados por ruído significativo ou outliers.

- **Passos do Algoritmo RANSAC:**
  - Seleção Aleatória: selecionar aleatoriamente um subconjunto mínimo de pontos do conjunto de dados.
  - Modelo Inicial: ajustar um modelo ao subconjunto selecionado.
  - Contagem de Inliers: determinar quantos pontos do conjunto de dados se ajustam bem ao modelo (ou seja, são inliers) dentro de um certo limiar de tolerância.
  - Armazenar o Melhor Modelo: se o número de inliers for maior que um limiar predefinido, considerar este modelo como o melhor modelo até o momento.
  - Iteração: repetir os passos de 1 a 4 um número pré-determinado de vezes ou até que um modelo satisfatório seja encontrado.
  - Refinamento do Modelo: ajustar o modelo final utilizando todos os inliers encontrados.
- **Vantagens:**
  - Robustez a Outliers: é eficaz na presença de muitos outliers.
  - Simplicidade: é relativamente simples de implementar e entender.
- **Desvantagens:**
  - Complexidade Computacional: pode ser computacionalmente intensivo se o número de iterações necessário for grande.
  - Parâmetros Sensíveis: a eficácia do algoritmo depende da escolha apropriada dos parâmetros, como o número de iterações e o limiar de tolerância.

6.5.1.3.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np

def fit_line_ransac(data, n, k, t, d):
    """RANSAC algoritmo para ajustar uma linha

    Parâmetros:
        data: um conjunto de pontos de dados observados
        n: o número mínimo de pontos de dados necessários para ajustar o modelo
        k: o número máximo de iterações permitidas no algoritmo
        t: um valor limite para determinar se um ponto se ajusta bem ao modelo
        d: o número de pontos de dados próximos necessários para afirmar que um
            modelo se ajusta bem aos dados

    Retorna:
        best_model - parâmetros do melhor modelo encontrado (inclinação e intercepto)
        best_inliers - inliers correspondentes ao melhor modelo
    """

    best_model = None
    best_inliers = []
    iterations = 0

    while iterations < k:
        # Aleatoriamente escolhe um subconjunto dos dados
        subset = data[np.random.choice(data.shape[0], n, replace=False)]

        # Ajusta o modelo para o subconjunto
        x_subset = subset[:, 0]
        y_subset = subset[:, 1]
        A = np.vstack([x_subset, np.ones(len(x_subset))]).T
        model, _, _, _ = np.linalg.lstsq(A, y_subset, rcond=None)

        # Determina inliers
        slope, intercept = model
        distances = np.abs(slope * data[:, 0] + intercept - data[:, 1])
        inliers = data[distances < t]

        # Atualiza o melhor modelo se o número de inliers é maior do que o limiar d
        if len(inliers) > d and len(inliers) > len(best_inliers):
            best_model = model
            best_inliers = inliers

        iterations += 1

    return best_model, best_inliers
```

Figura 15: Algoritmo RANSAC em Python
Fonte: ChatGPT, 2024

```python
# Criar alguns dados com ruído
np.random.seed(0)
x = np.linspace(0, 10, 100)
y = (3 * x) + 10 + np.random.normal(0, 1, x.shape)
data = np.column_stack((x, y))

# Adiciona alguns outliers
outliers = np.array([[5, 30], [7, 40], [8, 20]])
data = np.vstack([data, outliers])

# Aplicar o RANSAC
best_model, best_inliers = fit_line_ransac(data, n=2, k=100, t=2, d=50)
slope, intercept = best_model

print("Melhor modelo: y = {:.2f}x + {:.2f}".format(slope, intercept))
print("Número de inliers: ", len(best_inliers))
```

Figura 16: Implementação do Algoritmo RANSAC em um modelo de regressão linear
Fonte: ChatGPT, 2024

6.5.2. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Gerar dados de exemplo
np.random.seed(0)
n_samples = 100
X = np.random.rand(n_samples) * 100
y = 2 * X + np.random.randn(n_samples) * 10

# Adicionar outliers
n_outliers = 10
X[:n_outliers] = X[:n_outliers] * 10
y[:n_outliers] = y[:n_outliers] * 10

# Criar DataFrame
data = pd.DataFrame({'X': X, 'y': y})

# Variáveis independentes (com termo de intercepto)
X = sm.add_constant(data['X'])

# Variável dependente
y = data['y']

# Ajustar o modelo de regressão linear clássica
model_ols = sm.OLS(y, X).fit()

# Ajustar o modelo de regressão robusta com M-Estimadores
model_robust = sm.RLM(y, X, M=sm.robust.norms.HuberT()).fit()

# Resumo dos modelos
print("Resumo da Regressão Linear Clássica:")
print(model_ols.summary())

print("\nResumo da Regressão Robusta:")
print(model_robust.summary())

# Plotar os dados e as regressões ajustadas
plt.scatter(data['X'], data['y'], label='Dados com Outliers', alpha=0.6)
plt.plot(data['X'], model_ols.fittedvalues, label='OLS', color='red')
plt.plot(data['X'], model_robust.fittedvalues, label='Robusta', color='green')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Figura 17: Implementação de Regressão Robusta em Python
Fonte: ChatGPT, 2024

```
Resumo da Regressão Linear Clássica:
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.989
Model:                            OLS   Adj. R-squared:                  0.989
Method:                 Least Squares   F-statistic:                     9028.
Date:                Sun, 14 Jul 2024   Prob (F-statistic):           2.65e-98
Time:                        17:09:28   Log-Likelihood:                -506.84
No. Observations:                 100   AIC:                             1018.
Df Residuals:                      98   BIC:                             1023.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.7196      4.455      0.610      0.543      -6.122      11.561
X              2.0188      0.021     95.016      0.000       1.977       2.061
==============================================================================
Omnibus:                       31.499   Durbin-Watson:                   1.957
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              426.093
Skew:                           0.269   Prob(JB):                     2.99e-93
Kurtosis:                      13.098   Cond. No.                         241.
==============================================================================
```

Figura 18: Resumo do Resultado da Regressão Linear Clássica com Outliers
Fonte: ChatGPT, 2024

```
Resumo da Regressão Robusta:
                    Robust linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  100
Model:                            RLM   Df Residuals:                       98
Method:                          IRLS   Df Model:                           1
Norm:                           HuberT
Scale Est.:                        mad
Cov Type:                           H1
Date:                Sun, 14 Jul 2024
Time:                        17:09:28
No. Iterations:                    50
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2434      1.441      0.169      0.866      -2.582       3.068
X              2.0407      0.007    296.893      0.000       2.027       2.054
==============================================================================
```

Figura 19: Resumo do Resultado da Regressão Linear Robusta com Outliers
Fonte: ChatGPT, 2024

É possível visualizar a diferença nos valores dos coeficientes.

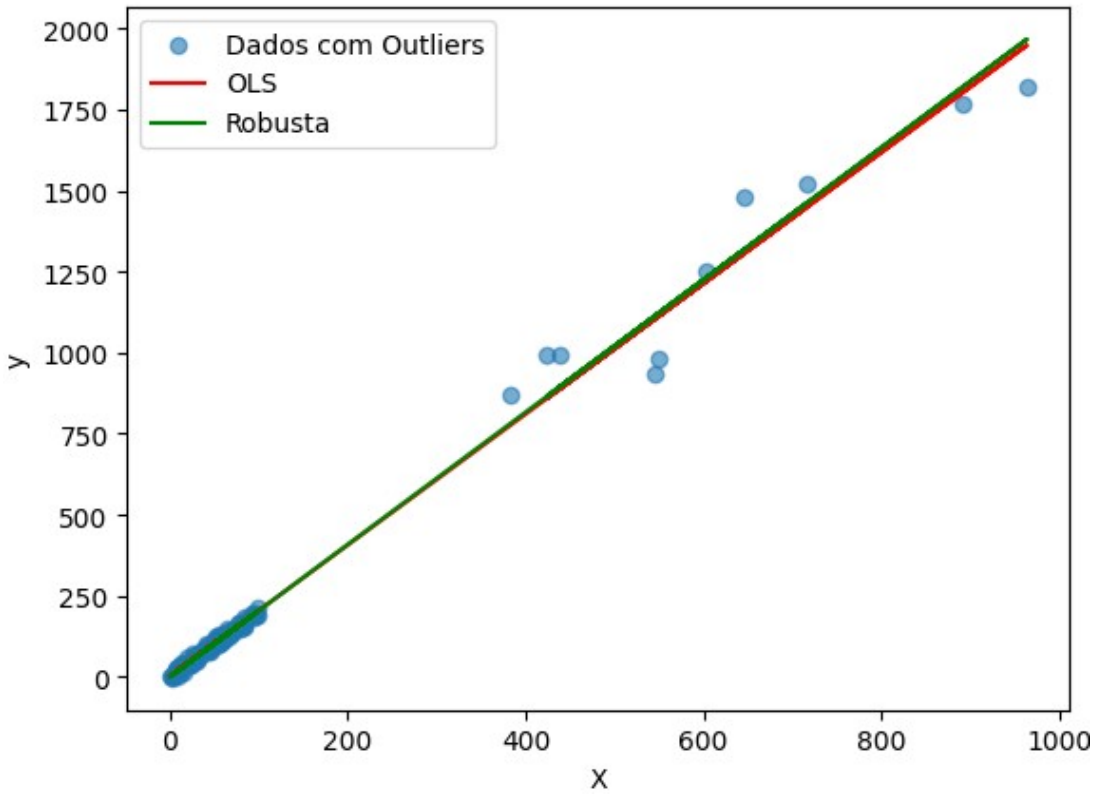


Figura 20: Gráfico Regressão Linear Clássica vs Robusta
Fonte: ChatGPT, 2024

6.6. REGRESSÃO DE COMPONENTES PRINCIPAIS (PCR)

A Regressão de Componentes Principais (PCR, do inglês Principal Component Regression) é uma técnica estatística que combina a Análise de Componentes Principais (PCA) e a regressão linear. Esta abordagem é útil quando se trabalha com dados multicolineares, ou seja, quando as variáveis preditoras estão altamente correlacionadas entre si.

- **Passos da PCR:**
  - Padronização dos Dados: normalizar os dados para que cada variável tenha média zero e variância um.
  - Análise de Componentes Principais: aplicar PCA aos dados padronizados para obter componentes principais, que são combinações lineares ortogonais das variáveis originais.
  - Seleção de Componentes: selecionar um número adequado de componentes principais, geralmente os que explicam a maior parte da variância nos dados.
  - Regressão Linear: realizar regressão linear usando os componentes principais selecionados como variáveis preditoras.
- **Vantagens:**
  - Redução de Dimensionalidade: reduz a dimensionalidade dos dados, o que pode melhorar a performance do modelo e reduzir o risco de overfitting.
  - Lida com Multicolinearidade: componentes principais são ortogonais, eliminando o problema da multicolinearidade entre variáveis preditoras.
- **Desvantagens:**
  - Interpretação: os componentes principais podem ser difíceis de interpretar em termos das variáveis originais.
  - Perda de Informação: se poucos componentes são selecionados, pode haver perda de informação relevante.

6.6.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Gerar dados de exemplo
np.random.seed(0)
X = np.random.rand(100, 5)
y = 3 * X[:, 0] + 2 * X[:, 1] + np.random.normal(0, 0.1, 100)

# Padronizar os dados
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=0)

# Aplicar PCA
pca = PCA()
X_train_pca = pca.fit_transform(X_train)

# Selecionar número adequado de componentes principais (explicando, por exemplo,
# 95% da variância)
explained_variance = np.cumsum(pca.explained_variance_ratio_)
num_components = np.argmax(explained_variance >= 0.95) + 1

print(f"Número de componentes selecionados: {num_components}")

# Ajustar regressão linear usando os componentes principais selecionados
pca = PCA(n_components=num_components)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

regressor = LinearRegression()
regressor.fit(X_train_pca, y_train)

# Fazer previsões
y_pred = regressor.predict(X_test_pca)

# Avaliar o modelo
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Coeficientes do modelo
print("Coeficientes do modelo:", regressor.coef_)
```

Figura 21: Implementação de Regressão de Componentes Principais em Python
Fonte: ChatGPT, 2024

6.7. REGRESSÃO DE MÍNIMOS QUADRADOS PARCIAIS (PLS)

A Regressão de Mínimos Quadrados Parciais (PLS, do inglês Partial Least Squares) é uma técnica que combina características de regressão linear e análise de componentes principais. É especialmente útil quando se trabalha com conjuntos de dados que têm muitas variáveis preditoras que são altamente correlacionadas, ou quando o número de observações é menor do que o número de variáveis preditoras.

- **Passos da PLS:**
  - Padronização dos Dados: normalizar os dados para que cada variável tenha média zero e variância um.
  - Decomposição em Componentes: encontrar componentes latentes que capturam a maior parte da covariância entre as variâncias preditoras e a variável resposta.
  - Construção do Modelo: ajustar um modelo de regressão linear utilizando as componentes latentes.
- **Vantagens:**
  - Lida com Multicolinearidade: como os componentes latentes são extraídos para maximizar a covariância com a variável resposta, a multicolinearidade entre variáveis preditoras não é um problema.
  - Redução de Dimensionalidade: pode reduzir a dimensionalidade dos dados, o que pode melhorar a performance do modelo e reduzir o risco de overfitting.
  - Interpretação: as componentes latentes podem ser mais interpretáveis em termos de variáveis originais.
- **Desvantagens:**
  - Complexidade Computacional: para conjuntos de dados muito grandes ou com um grande número de variáveis preditoras, a PLS pode ser computacionalmente intensiva, especialmente se muitas componentes forem necessárias.
  - Número de Componentes: a escolha errada pode levar a um modelo subajustado (underfitting) ou sobreajustado (overfitting). A validação cruzada é geralmente usada, mas pode ser demorada.

• Sensibilidade a Outliers: pode ser sensível a outliers nos dados, o que pode afetar a robustez do modelo. Métodos robustos de PLS existem, mas são mais complexos.

6.7.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.cross_decomposition import PLSRegression

# Gerar dados de exemplo
np.random.seed(0)
X = np.random.rand(100, 10)
y = 3 * X[:, 0] + 2 * X[:, 1] + np.random.normal(0, 0.1, 100)

# Padronizar os dados
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(
  X_scaled, y, test_size=0.2,random_state=0)

# Aplicar PLS
pls = PLSRegression(n_components=2)
pls.fit(X_train, y_train)

# Fazer previsões
y_pred = pls.predict(X_test)

# Avaliar o modelo
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Coeficientes do modelo
print("Coeficientes do modelo:", pls.coef_)
```

Figura 22: Implementação de Regressão de Mínimos Quadrados Parciais em Python
Fonte: ChatGPT, 2024

6.8. REGRESSÃO QUANTÍLICA

A Regressão Quantílica é uma técnica estatística que permite modelar a relação entre variáveis independentes e percentis específicos da variável dependente, ao contrário da regressão linear tradicional que se concentra na média da variável dependente. É especialmente útil para dados heterocedásticos ou quando a distribuição dos erros não é normal.

A escolha dos quantis a serem modelados depende do interesse específico da análise. Quantis mais baixos (como 0.1) modelam a cauda inferior da distribuição, enquanto quantis mais altos (como 0.9) modelam a cauda superior.

- **Vantagens:**
  - Flexibilidade: permite a modelagem de diferentes quantis da distribuição da variável dependente, proporcionando uma visão mais completa das relações entre variáveis.
  - Robustez a Outliers: a regressão quantílica é mais robusta a outliers do que a regressão linear, pois não depende de suposições sobre a distribuição dos erros.
  - Heterocedasticidade: funciona bem quando a variância dos erros muda com o nível das variáveis independentes, ao contrário da regressão linear que assume homocedasticidade.
  - Informação Adicional: fornece informações detalhadas sobre a distribuição condicional da variável dependente, ajudando a entender melhor a variabilidade dos dados.
- **Desvantagens:**
  - Sensibilidade a Outliers: embora a regressão quantílica seja robusta a outliers em muitos casos, quantis extremos podem ainda ser afetados por outliers. A detecção e tratamento de outliers pode ser necessária antes da análise.
  - Menor Eficiência em Pequenas Amostras: em amostras pequenas, a estimativa dos quantis pode ser menos eficiente e mais variável, o que pode resultar em maior incerteza nos coeficientes estimados.

6.8.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt

# Gerar dados de exemplo
np.random.seed(0)
n = 100
X = np.random.rand(n, 1)
y = 5 + 2 * X.squeeze() + np.random.randn(n)

# Adicionar alguns outliers
y[np.random.choice(n, 5)] += 10

# Ajustar modelo de regressão quantílica
model = smf.quantreg('y ~ X', {'X': X.squeeze(), 'y': y})
quantiles = [0.1, 0.5, 0.9]
results = [model.fit(q=q) for q in quantiles]

# Plotar os resultados
plt.scatter(X, y, alpha=0.5, label='Dados')
x_vals = np.linspace(X.min(), X.max(), 100)
for i, q in enumerate(quantiles):
    y_vals = results[i].params['Intercept'] + results[i].params['X'] * x_vals
    plt.plot(x_vals, y_vals, label=f'Quantil {q}')

plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Regressão Quantílica')
plt.show()

# Exibir os parâmetros dos modelos ajustados
for i, q in enumerate(quantiles):
    print(
      f"Quantil {q}: Intercepto = {results[i].params['Intercept']}",
      f"Coeficiente de X = {results[i].params['X']}")
```

Figura 23: Implementação de Regressão Quantílica em Python
Fonte: ChatGPT, 2024

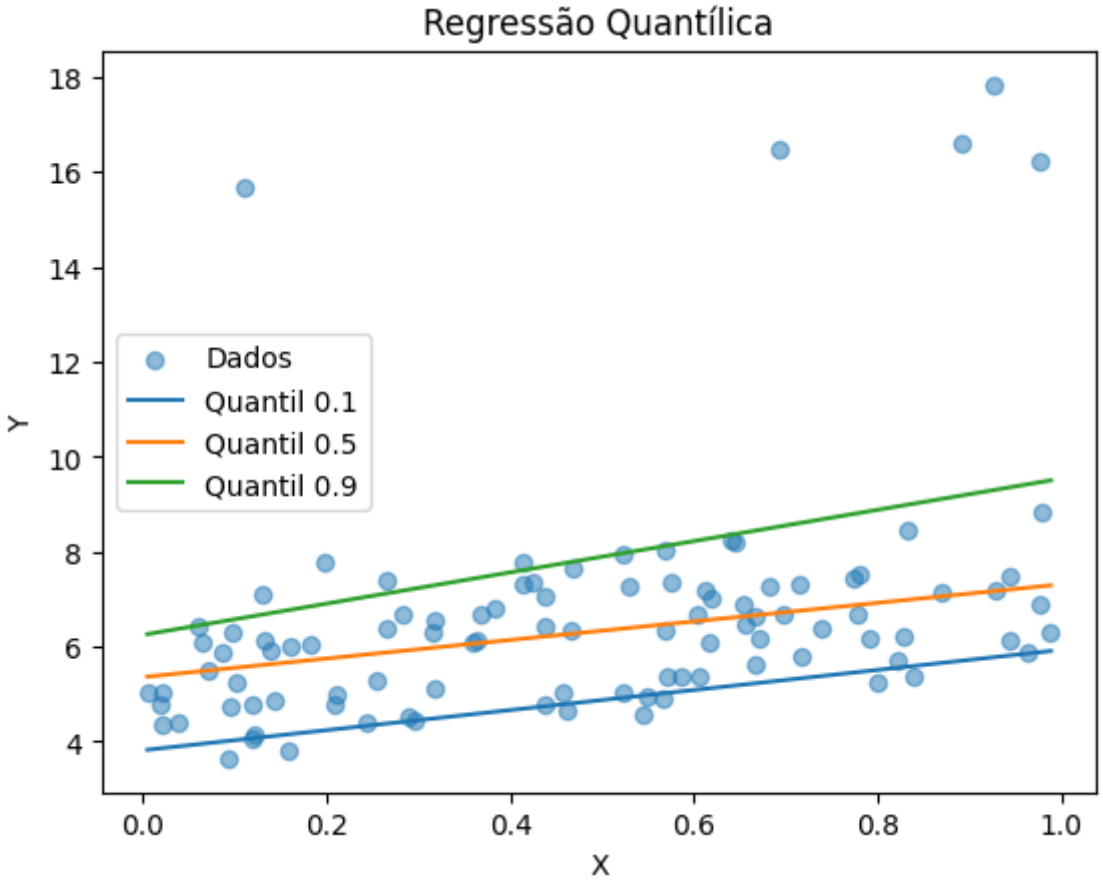


Figura 24: Gráfico da Regressão Quantílica
Fonte: ChatGPT, 2024

- Quantil 0.1: *Intercepto* ≈ 3.8185, Coeficiente de $X \approx 2.1149$
- Quantil 0.5: *Intercepto* ≈ 5.3596, Coeficiente de $X \approx 1.9541$
- Quantil 0.9: *Intercepto* ≈ 6.2485, Coeficiente de $X \approx 3.2923$

6.9. REGRESSÃO BAYESIANA

A Regressão Bayesiana é uma abordagem estatística que aplica os princípios da inferência bayesiana à regressão. Em vez de encontrar estimativas pontuais para os coeficientes de regressão, como na regressão linear tradicional, a regressão bayesiana trata os coeficientes como variáveis aleatórias e busca suas distribuições posteriores com base nos dados observados e nas distribuições a priori.

- **Princípios Básicos:**
  - Prior (Priori): representa o conhecimento ou suposições sobre os parâmetros antes de observar os dados. Pode ser informativa (baseada em conhecimento prévio) ou não informativa (pouca ou nenhuma informação prévia).
  - Likelihood (Verossimilhança): representa a probabilidade dos dados observados dado um conjunto de parâmetros.
  - Posterior (Posteriori): a distribuição posterior dos parâmetros é obtida atualizando a priori com a verossimilhança dos dados observados usando o Teorema de Bayes:

$$P(\theta|y)=\frac{P(y|\theta)P(\theta)}{P(y)}$$

  Onde θ são os parâmetros e y são os dados observados.

- **Vantagens:**
  - Incorporação de Conhecimento Prévio: permite incorporar conhecimento prévio sobre os parâmetros através das distribuições a priori.
  - Estimativas Intervalares: fornece distribuições completas dos parâmetros, permitindo a construção de intervalos de credibilidade e uma melhor quantificação da incerteza.
  - Flexibilidade: pode lidar com modelos complexos e incorporar diferentes tipos de incertezas.
  - Robustez: pode ser mais robusta a multicolinearidade e pequenos tamanhos de amostra.

- **Desvantagens:**
  - Complexidade Computacional: pode ser computacionalmente intensiva, especialmente para modelos complexos ou grandes conjuntos de dados, devido ao uso de métodos como Markov Chain Monte Carlo (MCMC).
  - Escolha das Prioris: a escolha das distribuições a priori pode ser subjetiva e influenciar os resultados, especialmente com dados limitados.
  - Interpretação: requer uma compreensão da inferência bayesiana e das distribuições de probabilidade, o que pode ser mais complexo do que a interpretação da regressão linear tradicional.

6.9.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
import pymc as pm

# Gerar dados de exemplo
np.random.seed(0)
n = 100
X = np.random.rand(n)
y = 3 + 2 * X + np.random.normal(0, 0.5, n)

# Ajustar modelo de regressão bayesiana
with pm.Model() as model:
    # Priori para os coeficientes
    intercept = pm.Normal('Intercept', mu=0, sigma=10)
    slope = pm.Normal('Slope', mu=0, sigma=10)

    # Priori para o erro
    sigma = pm.HalfNormal('Sigma', sigma=1)

    # Modelo linear
    mu = intercept + slope * X

    # Verossimilhança
    y_obs = pm.Normal('Y_obs', mu=mu, sigma=sigma, observed=y)

    # Amostragem posterior
    trace = pm.sample(2000, tune=1000, return_inferencedata=True)

# Sumário das estimativas
pm.summary(trace).round(2)

# Plotar os resultados
pm.plot_trace(trace)
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0, hspace=0.35)
plt.show()
```

Figura 25: Implementação da Regressão Bayesiana em Python
Fonte: ChatGPT, 2024

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	3.11	0.10	2.92	3.29	0.0	0.0	3773.0	4279.0	1.0
Sigma	0.51	0.04	0.44	0.58	0.0	0.0	4770.0	4292.0	1.0
Slope	1.97	0.18	1.65	2.31	0.0	0.0	3690.0	4074.0	1.0

Figura 26: Resumo dos Resultados da Regressão Bayesiana
Fonte: ChatGPT, 2024

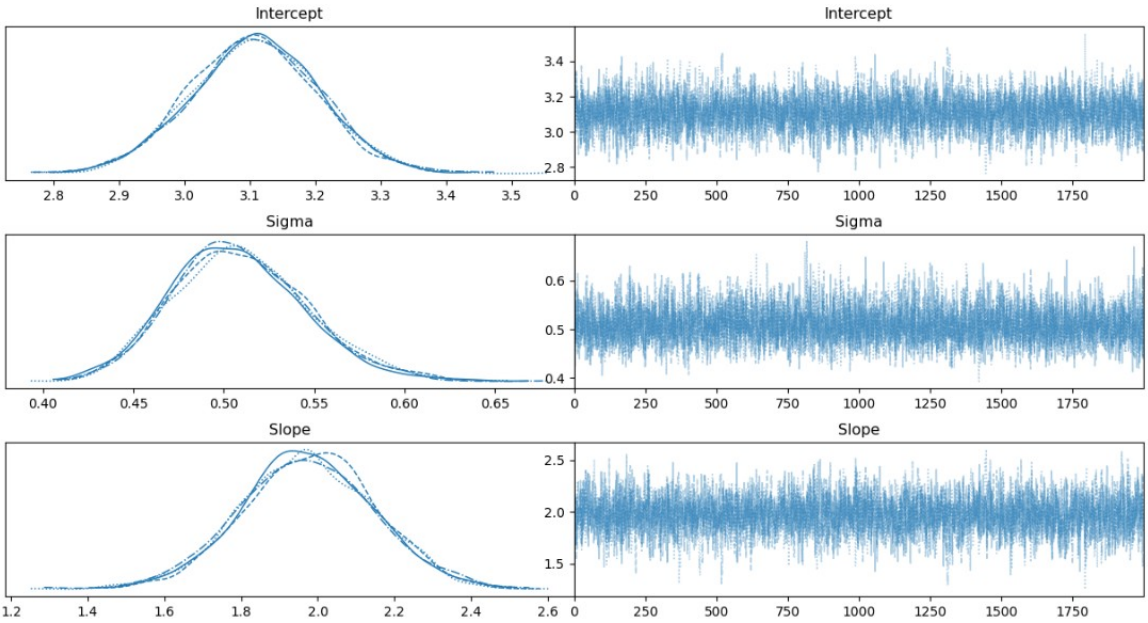


Figura 27: Gráfico dos Resultados da Regressão Bayesiana
Fonte: ChatGPT, 2024

6.10. MODELOS LINEARES GENERALIZADOS (GLM)

Modelos Lineares Generalizados (GLM) são uma extensão da regressão linear tradicional que permitem a modelagem de uma variável de respostas que não segue necessariamente uma distribuição normal. Eles são usados para modelar a relação entre uma variável dependente (ou resposta) e uma ou mais variáveis independentes (ou preditoras).

- **Principais Componentes do GLM:**
  - Função de Ligação (Link Function): relaciona a esperança condicional da variável dependente ao preditor linear. Permite que a variável resposta tenha uma distribuição diferente da normal. Exemplo: Logit, Probit, Log, Identidade.
  - Distribuição da Família Exponencial: a variável resposta Y é assumida como pertencente à família de distribuições exponenciais, que inclui normal, binomial, Poisson, gama, etc.
  - Preditor Linear: uma combinação linear das variáveis independentes.

$$\eta = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p$$

  - Função de Ligação Inversa: conecta a esperança da variável dependente à predição linear.

$$g(\mathrm{E}(Y)) = \eta$$

6.10.1. REGRESSÃO LOGÍSTICA

Modelo estatístico usado para prever a probabilidade de um resultado binário (com duas categorias, como 0 e 1, sucesso ou falha) com base em uma ou mais variáveis independentes (ou preditoras). Diferentemente da regressão linear, a regressão logística usa a função logística para garantir que as previsões estejam no intervalo [0, 1].

A fórmula básica para a regressão logística é:

$$p(X) = \frac{1}{1 + \mathrm{e}^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p)}}$$

Onde:

- p(X) é a probabilidade do evento de interesse.
- β_0 é o intercepto.
- β_p são os coeficientes das variáveis independentes X_p

A função de ligação (logit) é a função logística inversa:

$$logit(p) = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p$$

6.10.1.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Gerar dados de exemplo
np.random.seed(0)
n = 100
X = np.random.rand(n, 1)
y = (X[:, 0] > 0.5).astype(int)  # Variável dependente binária

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)

# Ajustar o modelo de regressão logística
model = LogisticRegression()
model.fit(X_train, y_train)

# Previsões
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]

# Avaliação do modelo
print("Relatório de Classificação:\n", classification_report(y_test, y_pred))
print("Matriz de Confusão:\n", confusion_matrix(y_test, y_pred))
print("Previsões de Probabilidade:", y_prob[:10])
```

Figura 28: Implementação de Regressão Logística em Python
Fonte: ChatGPT, 2024

6.10.2. REGRESSÃO DE POISSON

A Regressão de Poisson é um tipo de modelo linear generalizado (GLM) usado para modelar contagens de eventos que ocorrem em um intervalo de tempo fixo ou espaço fixo. É apropriado quando a variável dependente é uma contagem (ou seja, assume valores inteiros não negativos).

Fórmula da Regressão de Poisson:

$$\log(\lambda) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p$$

Onde:

- λ é a medida (e a variância) da distribuição de Poisson.
- β_0 é o intercepto.
- β_p são os coeficientes das variáveis independentes X_p.

A função de ligação é o logaritmo natural, que garante que a média λ seja sempre positiva.

É amplamente usada em áreas como epidemiologia, finanças e ciências sociais.

6.10.2.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
from sklearn.linear_model import PoissonRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Gerar dados de exemplo
np.random.seed(0)
n = 100
X = np.random.rand(n, 1)
y = np.random.poisson(lam=np.exp(1 + 2 * X)).flatten()

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.3, random_state=0)

# Ajustar o modelo de regressão de Poisson
model = PoissonRegressor()
model.fit(X_train, y_train)

# Previsões
y_pred = model.predict(X_test)

# Avaliação do modelo
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Previsões:", y_pred[:10])
```

Figura 29: Implementação da Regressão de Poisson em Python
Fonte: ChatGPT, 2024

6.10.3. REGRESSÃO GAMMA

A Regressão Gamma é uma técnica utilizada quando a variável dependente é contínua e positiva, e segue uma distribuição gamma. É útil para modelar dados que são assimetricamente distribuídos à direita, como tempos de espera, custos, ou durações de eventos.

Fórmula da Regressão Gamma:

$$g(\mathrm{E}(Y)) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

Onde:

- g é a função de ligação, frequentemente o logaritmo natural (log).
- E(Y) é a média da distribuição gamma da variável dependente Y.

- β_0 é o intercepto.
- β_p são os coeficientes das variáveis independentes X_p.

A função de ligação logarítmica garante que as previsões sejam sempre positivas, o que é apropriado para dados que seguem uma distribuição gamma.

É amplamente utilizada em áreas como economia, finanças, ciências atuariais e pesquisa médica, onde os tempos de espera, custos e durações de eventos precisam ser modelados de forma precisa.

6.10.3.1. IMPLEMENTAÇÃO EM PYTHON

```python
import numpy as np
import statsmodels.api as sm
from statsmodels.genmod.families import Gamma
from statsmodels.genmod.families.links import Log

# Gerar dados de exemplo
np.random.seed(0)
n = 100
X = np.random.rand(n, 1)
# Variável dependente com distribuição gamma
y = np.random.gamma(shape=2.0, scale=1 + 2 * X).flatten()

# Adicionar uma constante aos preditores (intercepto)
X = sm.add_constant(X)

# Ajustar o modelo de regressão gamma
# Usar a classe `Log` explicitamente
gamma_model = sm.GLM(y, X, family=Gamma(link=Log()))
result = gamma_model.fit()

# Sumário dos resultados
print(result.summary())

# Previsões
y_pred = result.predict(X)
print("Previsões:", y_pred[:10])
```

Figura 30: Implementação de Regressão Gamma em Python
Fonte: ChatGPT, 2024

6.11. ANÁLISE DE SOBREVIVÊNCIA COM REGRESSÃO

A Análise de Sobrevivência é uma área da estatística que trata da análise do tempo até que um evento de interesse ocorra, como a morte, a falha de um equipamento, ou a recidiva de uma doença. Na análise de sobrevivência, frequentemente utilizamos a regressão de Cox (modelo de riscos proporcionais de Cox) para avaliar o efeito de várias covariáveis sobre a taxa de falha ou de risco.

O modelo de Cox é semi-paramétrico e assume que a razão de riscos (hazard ratio) entre dois indivíduos é constante ao longo do tempo, independente do valor das covariáveis.

A fórmula básica do modelo de Cox é:

$$h(t|X) = h_o(t) \exp(\beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_P)$$

Onde:

- $h(t|X)$ é a função de risco no tempo t dada as covariáveis X.
- $h_o(t)$ é a função de risco base.
- β_p são os coeficientes das covariáveis X_p.

6.11.1. IMPLEMENTAÇÃO EM PYTHON

```python
import pandas as pd
import numpy as np
from lifelines import CoxPHFitter
from lifelines.datasets import load_rossi

# Carregar dados de exemplo
df = load_rossi()

# Ver os primeiros registros
print(df.head())

# Ajustar o modelo de Cox
cph = CoxPHFitter()
cph.fit(df, duration_col='week', event_col='arrest')

# Sumário dos resultados
cph.print_summary()

# Previsões de risco parcial
df['risk'] = cph.predict_partial_hazard(df)
print(df[['week', 'arrest', 'risk']].head())
```

Figura 31: Implementação da Análise de Sobrevivência em Python
Fonte: ChatGPT, 2024

7. BIBLIOGRAFIA

CHATGPT, 2024.