Machine Learning

Linear Regression

Jeff Abrahamson

août 2019

Linear models

Problem: $\{(x_i, y_i)\}$.

Given x, predict $\hat{y}$.

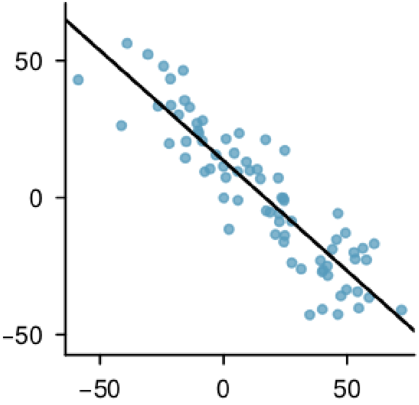
Linear models

x: **explanatory** or **predictor** variable.

y: **response** variable.

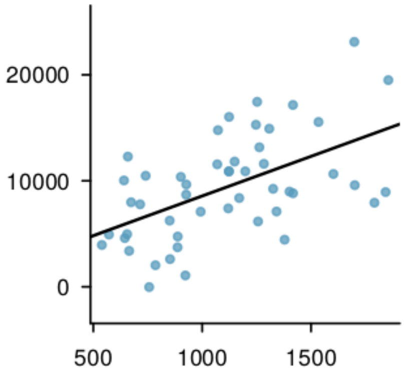
For some reason, we believe a linear model is a good idea.

Example:

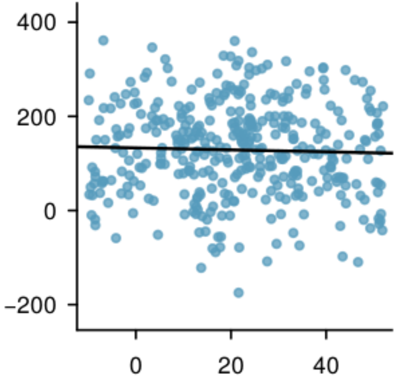


Linear models

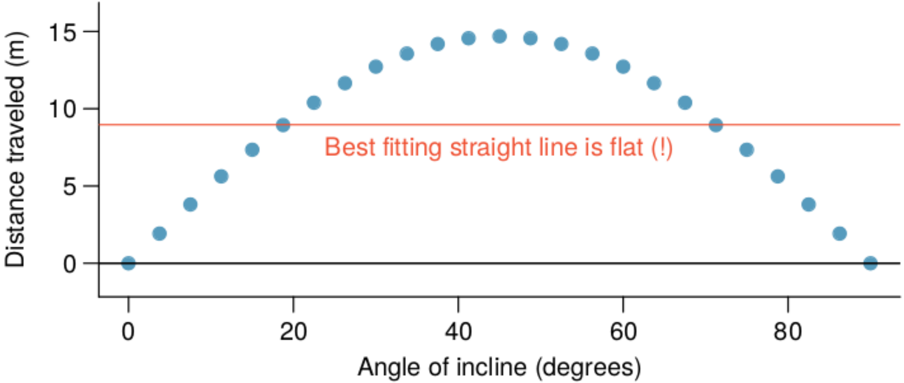
Example:



Example:



Example:


Distance traveled (m)
15
10
5
0
Best fitting straight line is flat (!)
0
20
40
60
80
Angle of incline (degrees)

Residuals

What's left over.

$$\text{data} = \text{fit} + \text{residual}$$

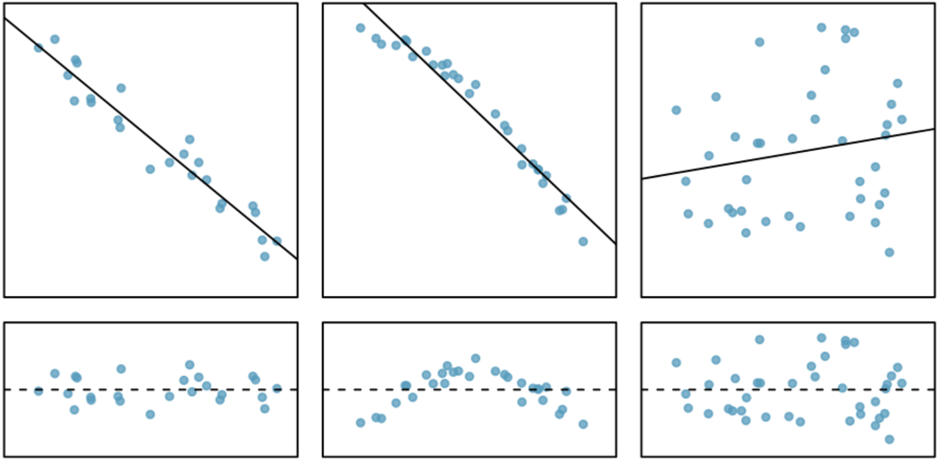
Residuals

What's left over.

$$y_i = \hat{y}_i + e_i$$

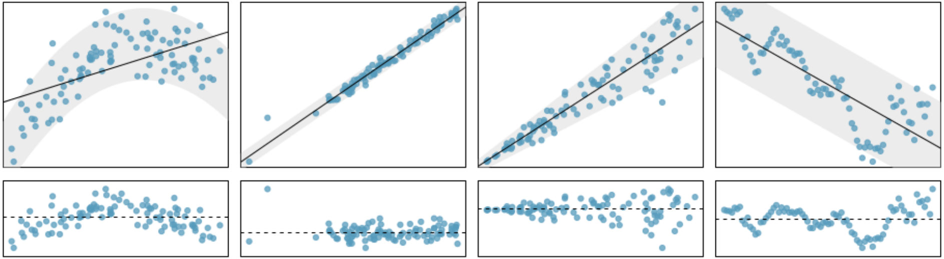
Residuals

What's left over.



Residuals

What's left over.



Residuals

What's left over.

Goal: small residuals.

$$\sum \mid e_i \mid$$

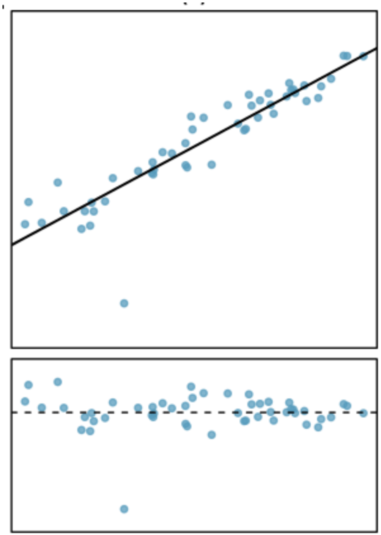
Residuals

What's left over.

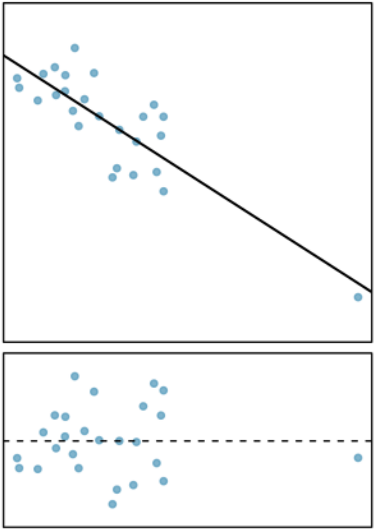
Goal: small residuals.

$$\sum e_i^2$$

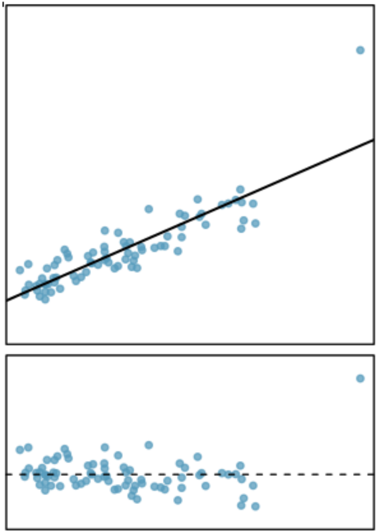
Outliers



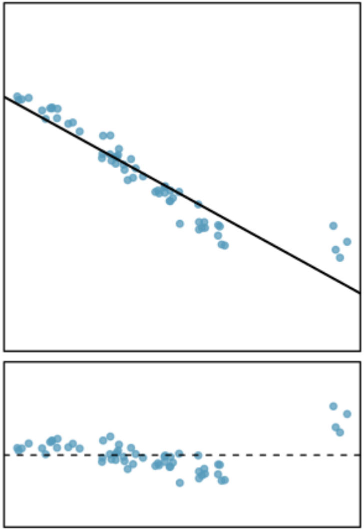
Outliers



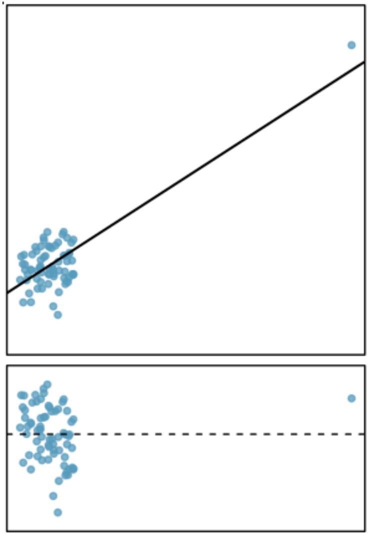
Outliers



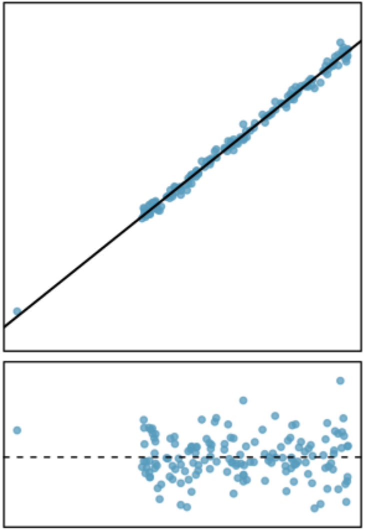
Outliers



Outliers

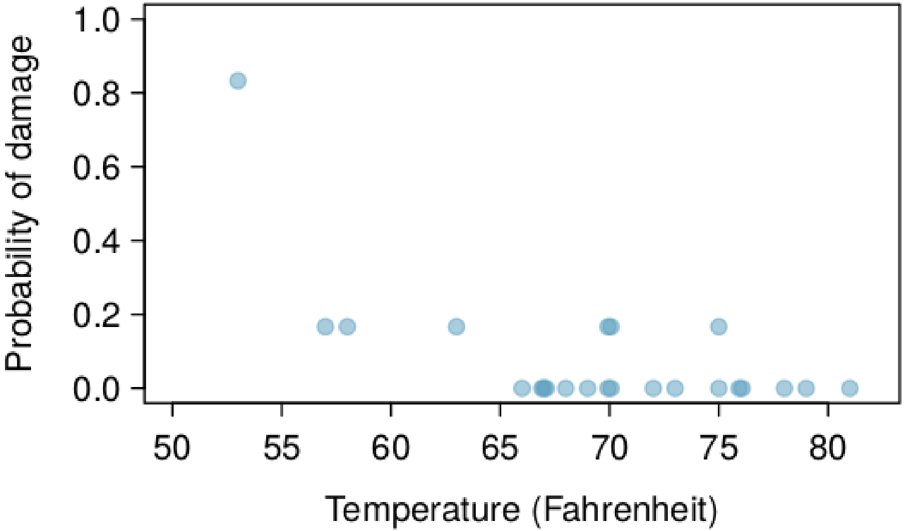


Outliers



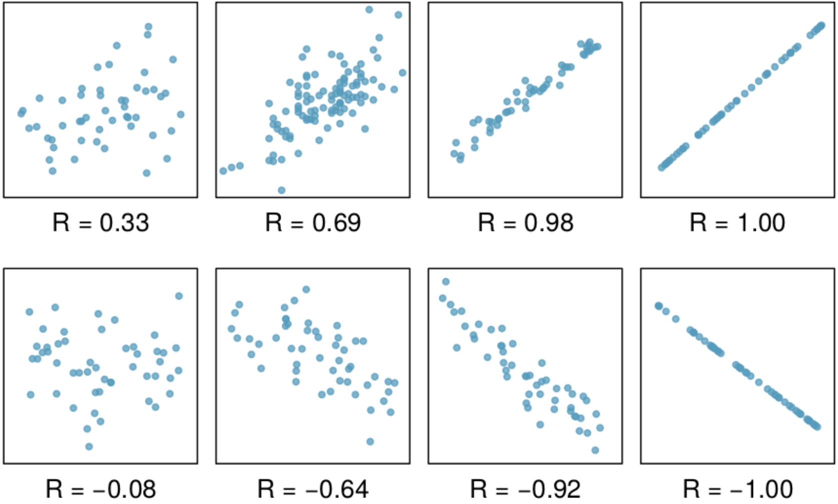
Don’t ignore outliers.

Outliers

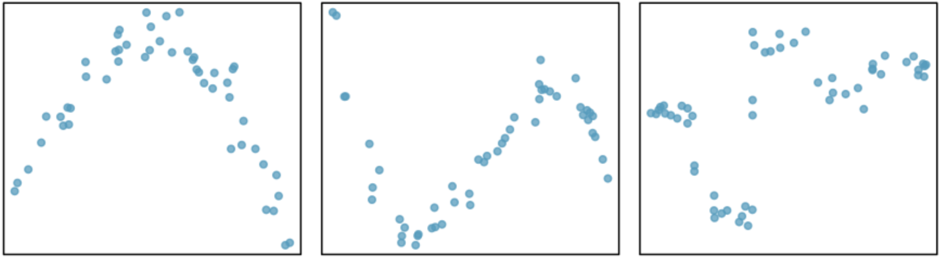


Probability of damage
1.0
0.8
0.6
0.4
0.2
0.0
50
55
60
65
70
75
80
Temperature (Fahrenheit)

Correlation



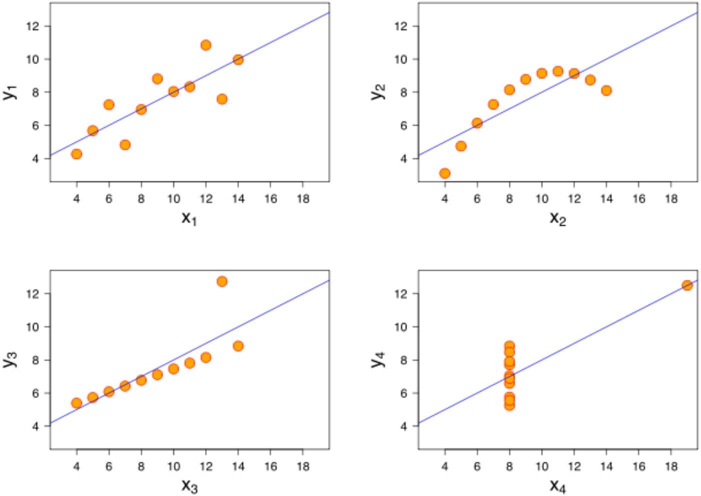
Correlation



R = −0.23 R = 0.31 R = 0.50

Correlation

Anscombe's Quartet



Correlation does not imply causation

Hypothesis (model)

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Cost function

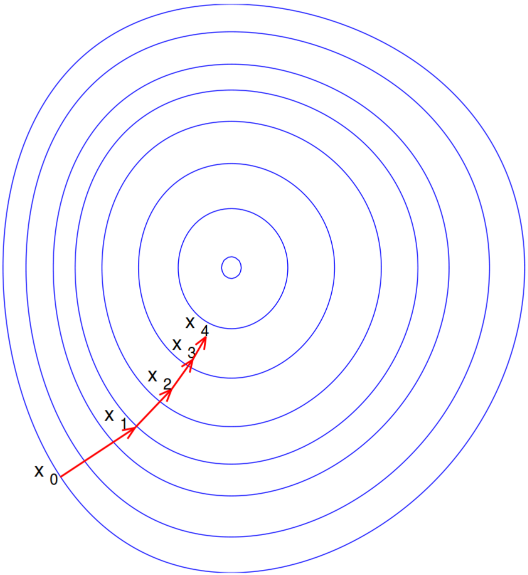
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Gradient descent

$$\begin{cases} \theta_0 & \leftarrow \theta_0 - \alpha \dfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \theta_1 & \leftarrow \theta_1 - \alpha \dfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \end{cases}$$

Gradient descent

$$\begin{cases} \theta_0 & \leftarrow \theta_0 - \dfrac{\alpha}{m} \displaystyle\sum_{i=1}^{m} (h_\theta(x_i) - y_i) \\ \theta_1 & \leftarrow \theta_1 - \dfrac{\alpha}{m} \displaystyle\sum_{i=1}^{m} ((h_\theta(x_i) - y_i) \cdot x_i) \end{cases}$$


x 0
x 1
x 2
x 3
x 4

Hypothesis again

$$
\begin{aligned}
h_\theta(x) &= \theta_0 + \theta_1 x_1 \\
&= \theta_0 + \sum_{i=1}^{1} \theta_i x_i \\
&= [\theta_0, \theta_1] \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \\
&= \theta^T x
\end{aligned}
$$

Hypothesis (multiple regression)

$$
\begin{aligned}
h_\theta(x) &= \theta_0 + \sum_{i=1}^{n} \theta_i x_i \\
&= [\theta_0, \cdots, \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \\
&= \theta^T x
\end{aligned}
$$

Hypothesis (multiple regression)

$$
\begin{aligned}
h_\theta(x) &= \theta^T x \\
&= \theta^T x^{(1)}
\end{aligned}
$$

Hypothesis (multiple regression)

$$X = \begin{bmatrix} | & | & \cdots & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \cdots & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(m)} \end{bmatrix}$$

Hypothesis (multiple regression)

$$
\begin{aligned}
h_\theta(X) &= \theta^T X \\
&= \left[h_0(x^{(1)}), h_0(x^{(2)}), \cdots, h_0(x^{(m)})\right] \\
&= \theta^T X
\end{aligned}
$$

Hypothesis (multiple regression)

or $X\theta$ if row vectors…

Cost function (multiple regression)

$$
\begin{aligned}
J(\theta) &= \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \\
&= \frac{1}{2m}(X\theta - Y)^T(X\theta - Y)
\end{aligned}
$$

Gradient descent (multiple regression)

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)}$$

$$\text{for } j = 1, \cdots, n$$

Gradient descent (multiple regression)

$$\theta \leftarrow \theta - \nabla J(\theta)$$

$$\text{where } \nabla = \begin{bmatrix} \frac{\partial}{\partial \theta_0} \\ \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_n} \end{bmatrix}$$

Scikit Learn

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0000...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```