Linear Regression

November 19, 2025

1 Linear Regression Tutorial

Author: Andrew Andrade (andrew@andrewandrade.ca)

This is part one of a series of tutorials related to regression used in data science.

Recommended reading: https://www.statlearning.com/ (Chapter 2)

https://hastie.su.domains/ISLR2/Slides/Ch3_Linear_Regression.pdf https://github.com/intro-stat-learning/ISLP_labs/blob/stable/Ch03-linreg-lab.ipynb

In this tutorial, We will first learn to fit a simple line using Least Squares Linear Regression (LSLR), plot residuals, residual distribution, statistics approach to linear regression, horizontal residuals and end with total least squares linear regression.

1.1 Fitting a line using LSLR

First let us import the necessary libraries and read the data file.

[10]:
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from math import log
from sklearn import linear_model
import seaborn as sns # Added for loading the dataset

#comment below if not using ipython notebook
%matplotlib inline
```

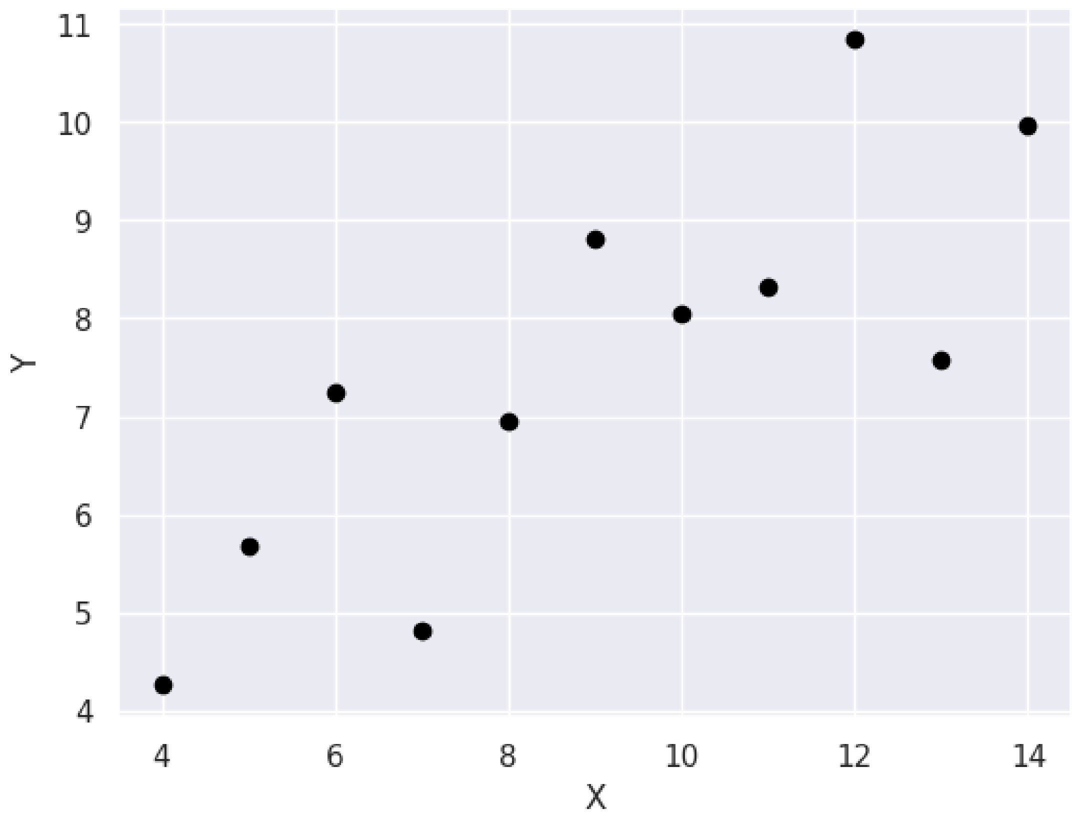
Now lets read the first set of data, take a look at the dataset and make a simple scatter plot.

[11]:
```python
# FIX: Load the dataset from seaborn instead of a missing CSV file
anscombe = sns.load_dataset('anscombe')

# Filter for dataset "I"
anscombe_i = anscombe[anscombe['dataset'] == 'I'].copy()

# Reset the index to be 0-based, which the rest of the notebook expects
anscombe_i.reset_index(drop=True, inplace=True)

anscombe_i
```

```
[11]:    dataset     x      y
      0        I  10.0   8.04
      1        I   8.0   6.95
      2        I  13.0   7.58
      3        I   9.0   8.81
      4        I  11.0   8.33
      5        I  14.0   9.96
      6        I   6.0   7.24
      7        I   4.0   4.26
      8        I  12.0  10.84
      9        I   7.0   4.82
      10       I   5.0   5.68
```

```
[12]: plt.scatter(anscombe_i.x, anscombe_i.y,  color='black')
      plt.ylabel("Y")
      plt.xlabel("X")
```

```
[12]: Text(0.5, 0, 'X')
```



Luckly for us, we do not need to implement linear regression, since scikit learn already has a very efficient implementation. The straight line can be seen in the plot below, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

The coefficients, the residual sum of squares and the variance score are also calculated.

Note: from reading the documentation this method computes the least squares solution using a singular value decomposition of X. If X is a matrix of size (n, p) this method has a cost of O(np^2), assuming that $n \geq p$. A more efficient alternative (for large number of features) is to use Stochastic Gradient Descent or another method outlined in the linear models documentation

If you do not know what BigO is, please read the background information from the notes (or take a algorithms course).

y = mx + b

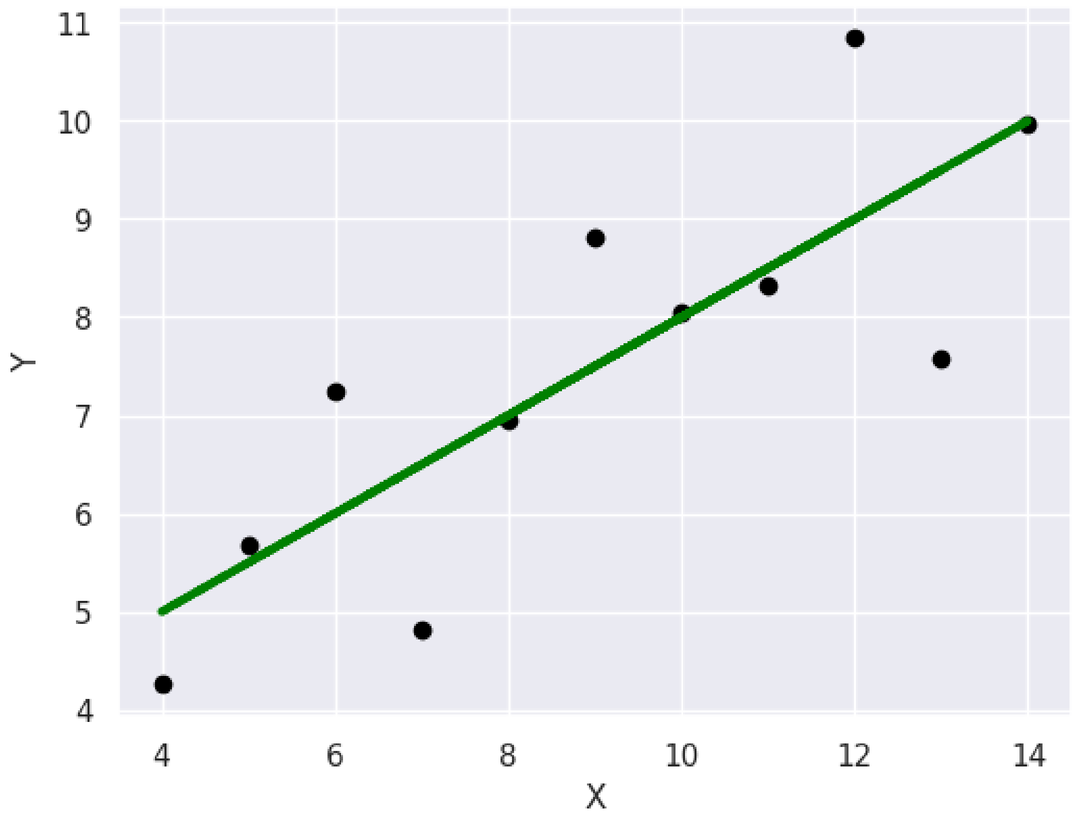
What is `m`?

That is the coefficient.

```
[13]: regr_i = linear_model.LinearRegression()

# FIX: Use .values.reshape(-1, 1) to convert pandas Series to 2D numpy array
# A pandas Series object does not have a .reshape() method.
X = anscombe_i.x.values.reshape(-1, 1)
y = anscombe_i.y.values.reshape(-1, 1)

#Fit a line
regr_i.fit(X,y)

# The coefficients
print('Coefficients: \n', regr_i.coef_)

# The mean square error
print("Residual sum of squares: %.2f"
      % np.mean((regr_i.predict(X) - y) ** 2))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr_i.score(X, y))

plt.plot(X,regr_i.predict(X), color='green',
         linewidth=3)

plt.scatter(anscombe_i.x, anscombe_i.y,  color='black')

# FIX: The x and y labels were swapped in the original code
plt.ylabel("Y")
plt.xlabel("X")
```

```
Coefficients:
 [[0.50009091]]
Residual sum of squares: 1.25
Variance score: 0.67
```

```
[13]: Text(0.5, 0, 'X')
```



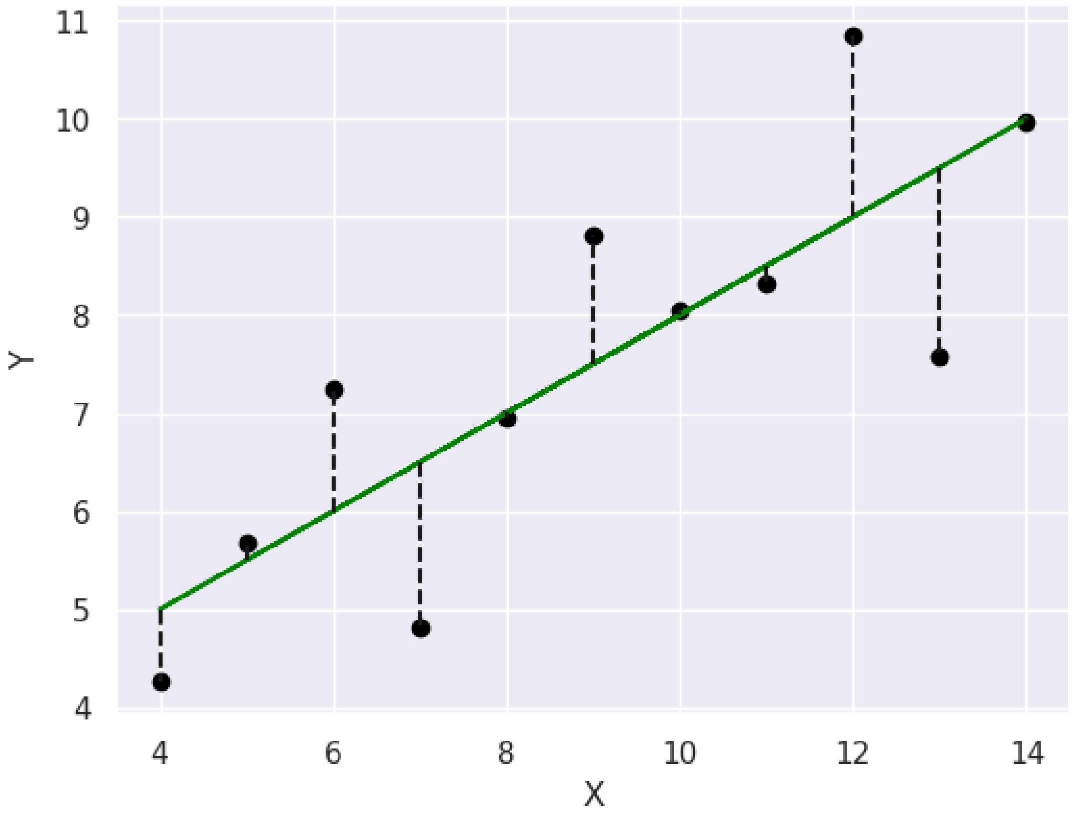
1.2 Residuals

From the notes, we learnt that we use ordinary linear regression when y is dependant on x since the algorithm reduces the vertical residual (y_observed - y predicted). The figure below outlines this using a different method for linear regression (using a polyfit with 1 polynomial).

```
[14]: # FIX: Use np.polyfit and plt. functions directly
      # Avoids 'from pylab import *'

      # determine the line-fit
      # FIX: Use the 1D pandas Series 'anscombe_i.y' for polyfit, not the 2D 'y' array
      k,d = np.polyfit(anscombe_i.x, anscombe_i.y, 1)
      yfit = k*anscombe_i.x + d

      # plot the data
      plt.figure(1)
      plt.scatter(anscombe_i.x, anscombe_i.y, color='black')
      plt.plot(anscombe_i.x, yfit, 'green')
```

```
# FIX: Replaced the slow loop with efficient plt.vlines()
# Plot vertical lines from the predicted y (yfit) to the actual y (anscombe_i.y)
plt.vlines(anscombe_i.x, ymin=yfit, ymax=anscombe_i.y, colors='k',␣
 ↪linestyles='--')

plt.xlabel('X')
plt.ylabel('Y')
```

[14]:
```
Text(0, 0.5, 'Y')
```

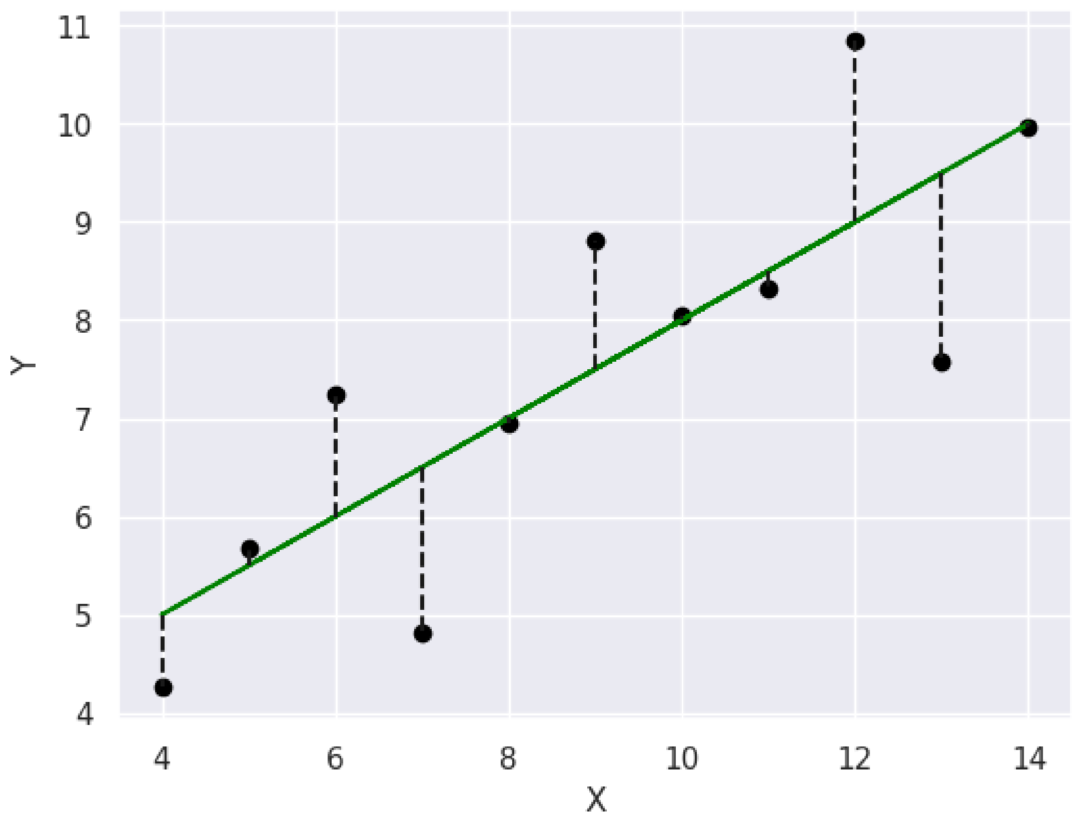


Now let us plot the residual (y - y predicted) vs x.

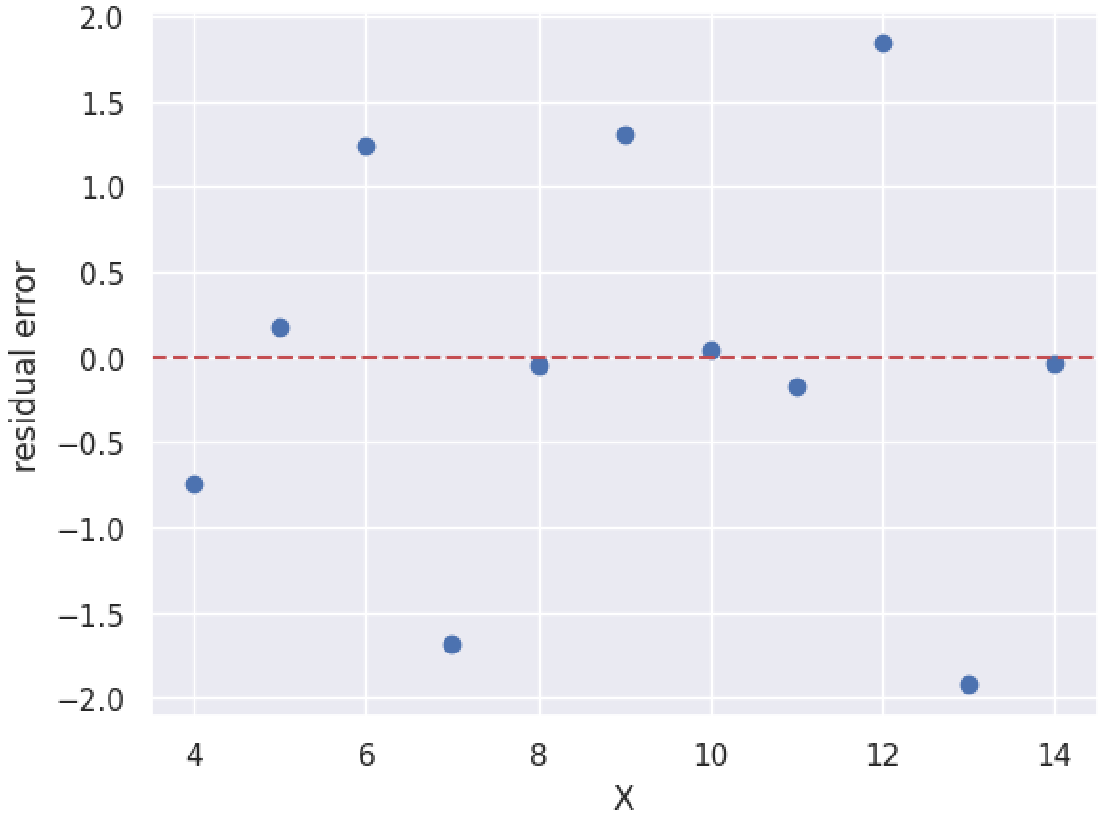
[15]:
```
# FIX: Need to import 'norm' from scipy.stats for the PDF plot
from scipy.stats import norm

# This plot is a repeat of the one above, which is fine
plt.figure(1)
plt.scatter(anscombe_i.x, anscombe_i.y, color='black')
plt.plot(anscombe_i.x, yfit, 'green')
plt.vlines(anscombe_i.x, ymin=yfit, ymax=anscombe_i.y, colors='k',␣
 ↪linestyles='--')
```

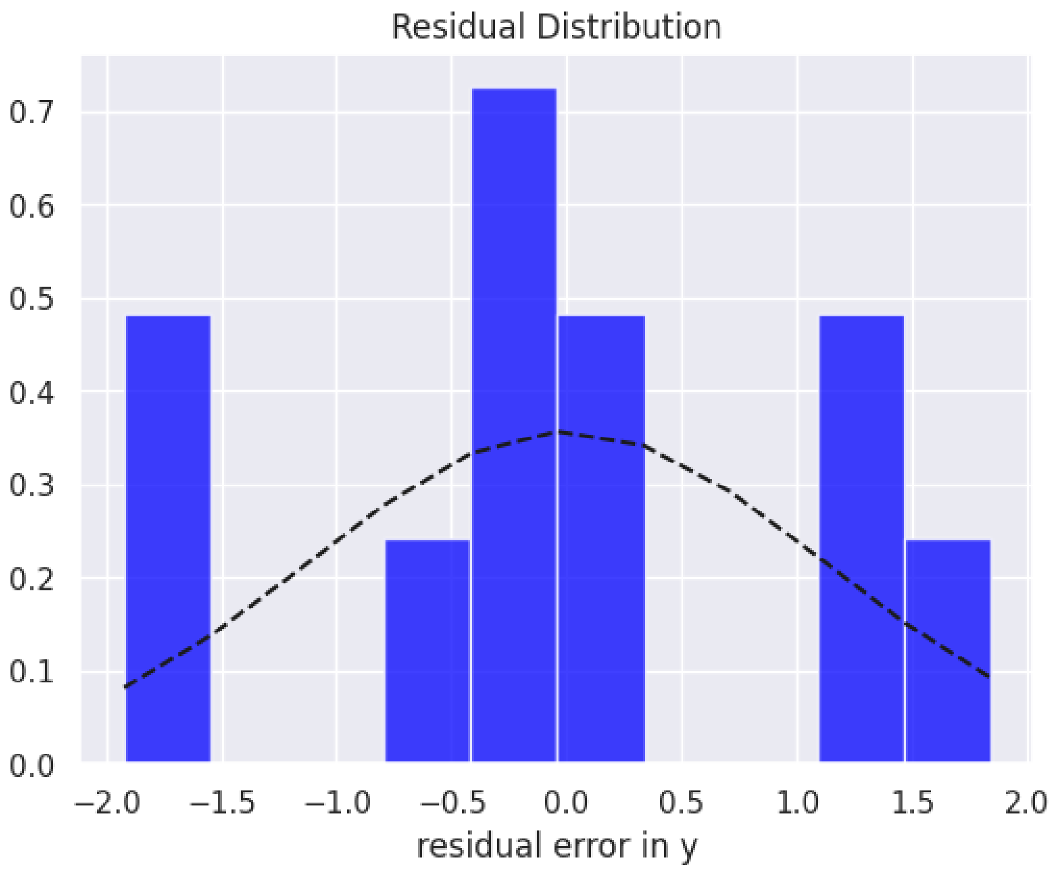
```
plt.xlabel('X')
plt.ylabel('Y')

# --- New plots from this cell ---

# Calculate residual error (y_observed - y_predicted)
residual_error = anscombe_i.y - yfit
error_mean = np.mean(residual_error)
error_sigma = np.std(residual_error)


# Plot 2: Residuals vs. X
plt.figure(2)
plt.scatter(anscombe_i.x, residual_error, label='residual error')
plt.axhline(y=0, color='r', linestyle='--') # Add a zero line for reference
plt.xlabel("X")
plt.ylabel("residual error")


# Plot 3: Histogram of Residuals
plt.figure(3)

# FIX: Use 'density=True' instead of deprecated 'normed=1'
n, bins, patches = plt.hist(residual_error, 10, density=True, facecolor='blue',␣
 ↪alpha=0.75)

# FIX: Use 'norm.pdf' from scipy.stats, not 'P.normpdf'
y_pdf = norm.pdf(bins, error_mean, error_sigma)
l = plt.plot(bins, y_pdf, 'k--', linewidth=1.5)

plt.xlabel("residual error in y")
plt.title("Residual Distribution")
```

```
[15]: Text(0.5, 1.0, 'Residual Distribution')
```



11
10
9
8
7
6
5
4
Y
4
6
8
10
12
14
X


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
residual error
4
6
8
10
12
14
X



As seen the the histogram, the residual error should be (somewhat) normally distributed and centered around zero. This post explains why.

If the residuals are not randomly distributed around zero, consider applying a transform to the data or applying non-linear regression. In addition to looking at the residuals, one could use the statsmodels library to take a statistical approach to ordinary least squares regression.

[16]:
```
# load statsmodels as alias ``sm``
import statsmodels.api as sm

y = anscombe_i.y
X = anscombe_i.x
# Adds a constant term to the predictor
# y = mx +b (where b is the constant)
X = sm.add_constant(X)

#fit ordinary least squares
est = sm.OLS(y, X)
est = est.fit()
```

```
est.summary()
```

[16]:

Dep. Variable:	y	**R-squared:**	0.667
Model:	OLS	**Adj. R-squared:**	0.629
Method:	Least Squares	**F-statistic:**	17.99
Date:	Wed, 19 Nov 2025	**Prob (F-statistic):**	0.00217
Time:	14:00:21	**Log-Likelihood:**	-16.841
No. Observations:	11	**AIC:**	37.68
Df Residuals:	9	**BIC:**	38.48
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	3.0001	1.125	2.667	0.026	0.456	5.544
x	0.5001	0.118	4.241	0.002	0.233	0.767

Omnibus:	0.082	**Durbin-Watson:**	3.212
Prob(Omnibus):	0.960	**Jarque-Bera (JB):**	0.289
Skew:	-0.122	**Prob(JB):**	0.865
Kurtosis:	2.244	**Cond. No.**	29.1

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The important parts of the summary are the:

What is the R squared?

What is the Adj. R squared?

What is the P value?

What are the 95% confidence intervals?

Helpful links: - R-squared (or coefficeient of determination

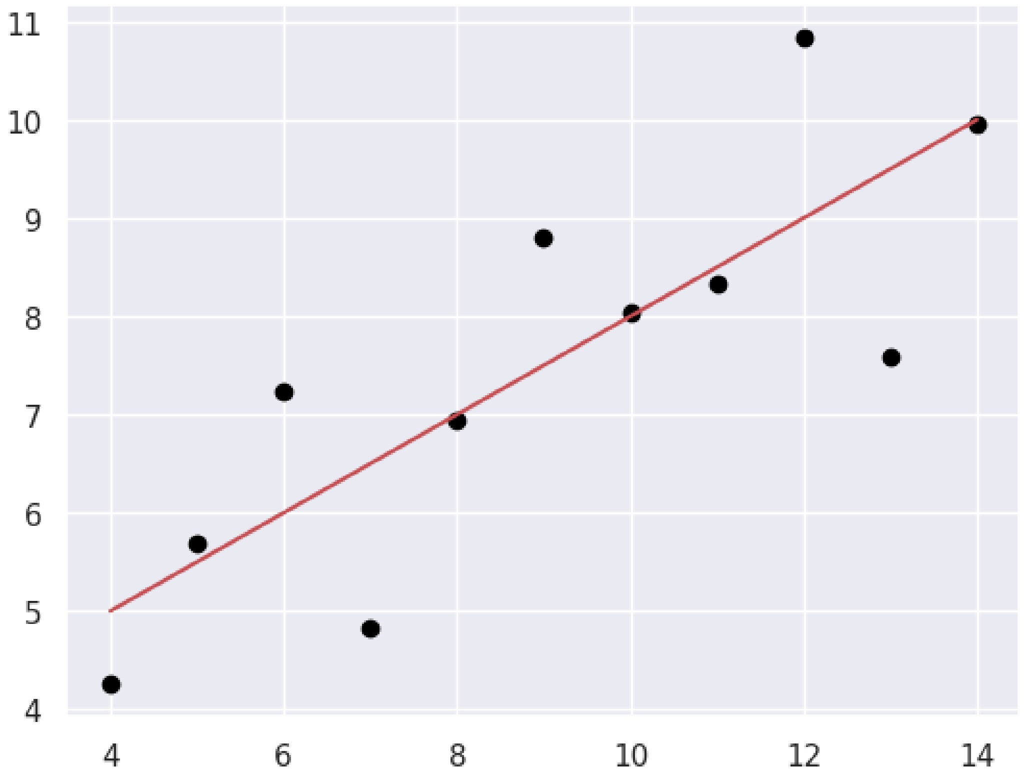
- 95.0% Conf. Interval
- http://onlinestatbook.com or http://stattrek.com/tutorials/ap-statistics-tutorial.aspx are great free resources which outlines all the necessary background to be a great statstician and data scientist. Both http://onlinestatbook.com/2/regression/inferential.html, and http://stattrek.com/regression/slope-confidence-interval.aspx?Tutorial=AP provide the specifics of confidence intervals for linear regression

We can now plot the fitted line to the data and observe the same results as the previous two methods for linear regression.

[17]:
```
plt.scatter(anscombe_i.x, anscombe_i.y,  color='black')
X_prime = np.linspace(min(anscombe_i.x), max(anscombe_i.x), 100)[:, np.newaxis]

# add constant as we did before
X_prime = sm.add_constant(X_prime)
y_hat = est.predict(X_prime)
```

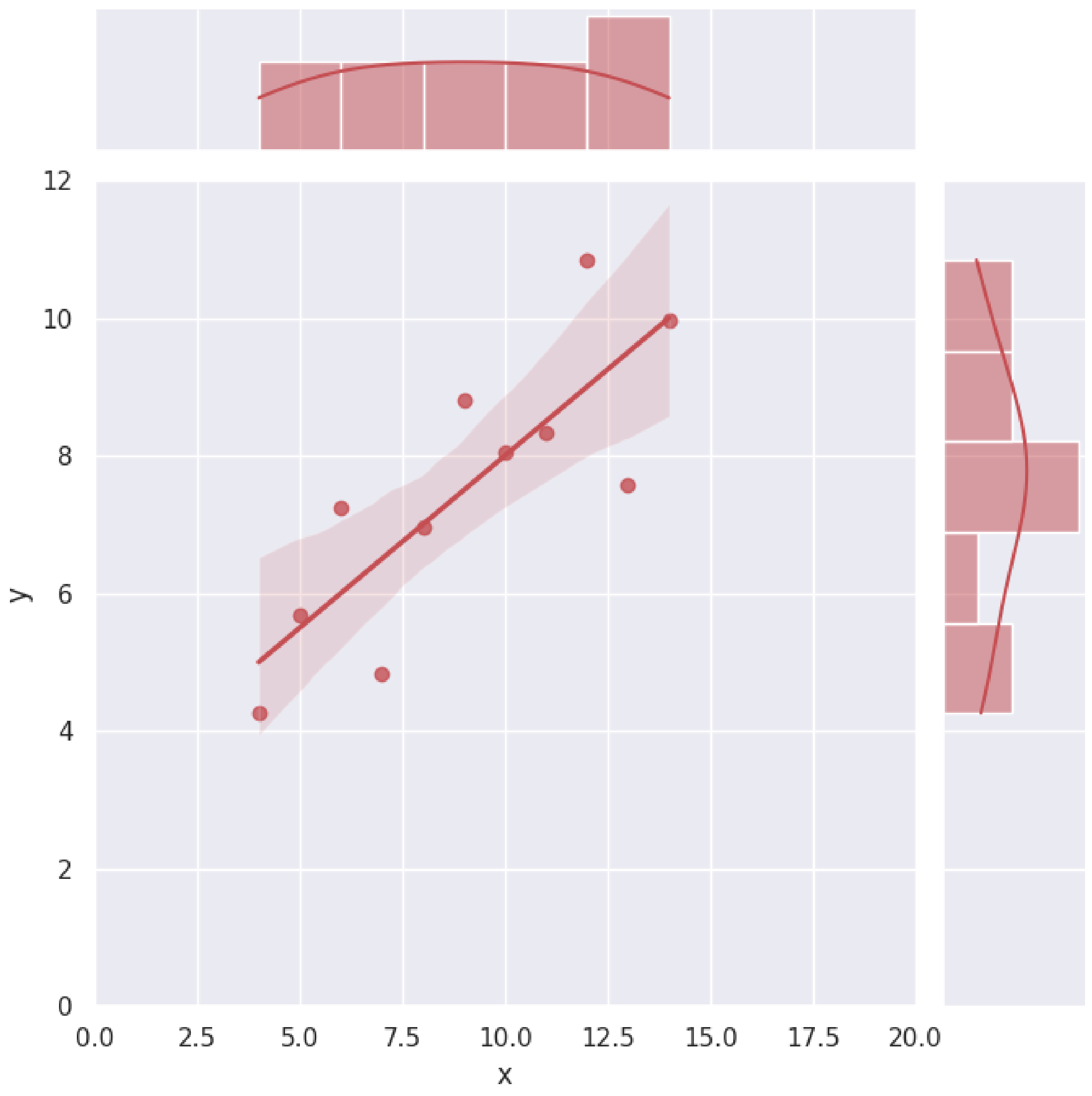
```
# Add the regression line (provides same as above)
# X_prime[:, 1] selects the original x-values, skipping the constant column [:,␣
↪0]
plt.plot(X_prime[:, 1], y_hat, 'r')
```

[17]:
```
[<matplotlib.lines.Line2D at 0x75564f9f7620>]
```



If we want to be even more fancier, we can use the seaborn library to plot Linear regression with marginal distributions which also states the pearsonr and p value on the plot. Using the statsmodels approach is more rigourous, but sns provides quick visualizations.

[18]:
```
import seaborn as sns
#this just makes the plots pretty (in my opion)
sns.set(style="darkgrid", color_codes=True)

# FIX: Use keyword arguments (data=, x=, y=) for modern seaborn compatibility
# and 'height=7' instead of deprecated 'size=7'
g = sns.jointplot(data=anscombe_i, x="x", y="y", kind="reg",
                  xlim=(0, 20), ylim=(0, 12), color="r", height=7)
```



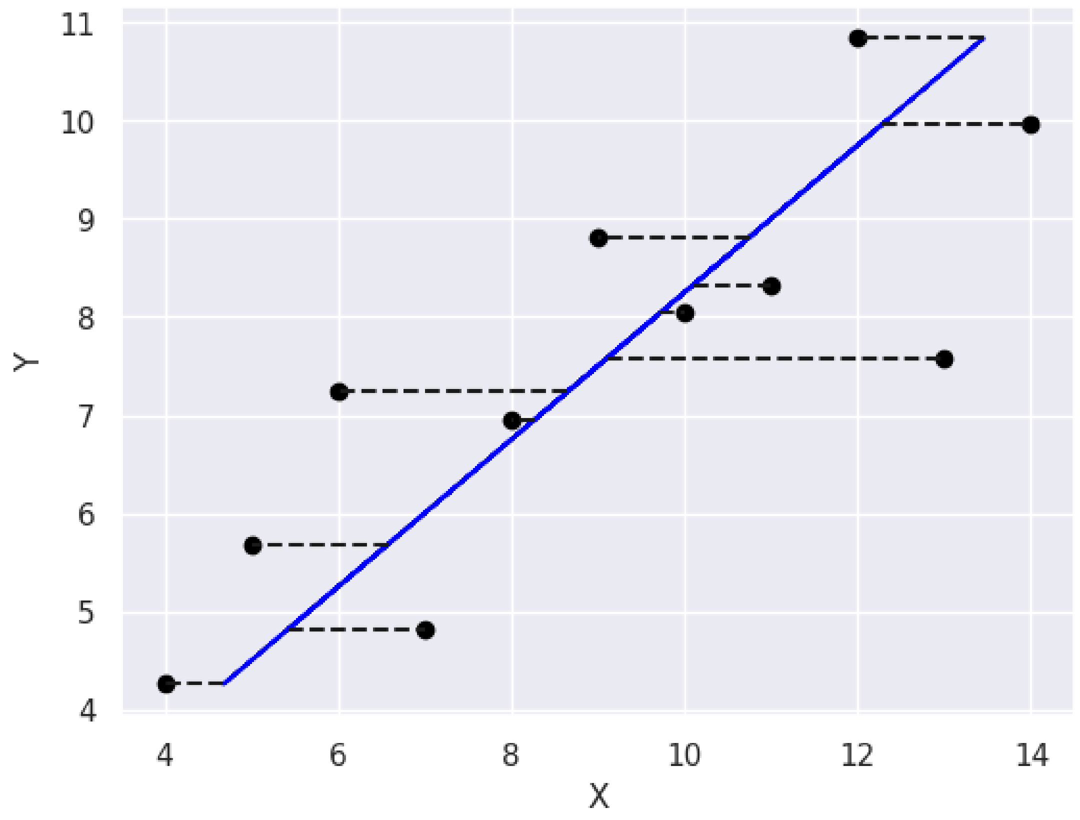
Usually we calculate the (vertical) residual, or the difference in the observed and predicted in the y. This is because "the use of the least squares method to calculate the best-fitting line through a two-dimensional scatter plot typically requires the user to assume that one of the variables depends on the other. (We caculate the difference in the y) However, in many cases the relationship between the two variables is more complex, and it is not valid to say that one variable is independent and the other is dependent. When analysing such data researchers should consider plotting the three regression lines that can be calculated for any two-dimensional scatter plot."

1.3 Regression using Horizontal Residual

If X is dependant on y, then the regression line can be made based on horizontal residuals as shown below.

[19]:
```
# FIX: No need to create 2D X and y arrays here.

# This time, we fit x as a function of y
k,d = np.polyfit(anscombe_i.y, anscombe_i.x, 1)

# FIX: Calculate xfit using the 1D Series anscombe_i.y
xfit = k*anscombe_i.y + d

plt.figure(2)
# plot the data
plt.scatter(anscombe_i.x, anscombe_i.y, color='black')
# Plot the regression line
plt.plot(xfit, anscombe_i.y, 'blue')

# FIX: Replaced loop with efficient plt.hlines()
# Plot horizontal lines from the predicted x (xfit) to the actual x (anscombe_i.
 ↪x)
plt.hlines(anscombe_i.y, xmin=xfit, xmax=anscombe_i.x, colors='k',␣
 ↪linestyles='--')

plt.xlabel('X')
plt.ylabel('Y')
```

[19]:
```
Text(0, 0.5, 'Y')
```



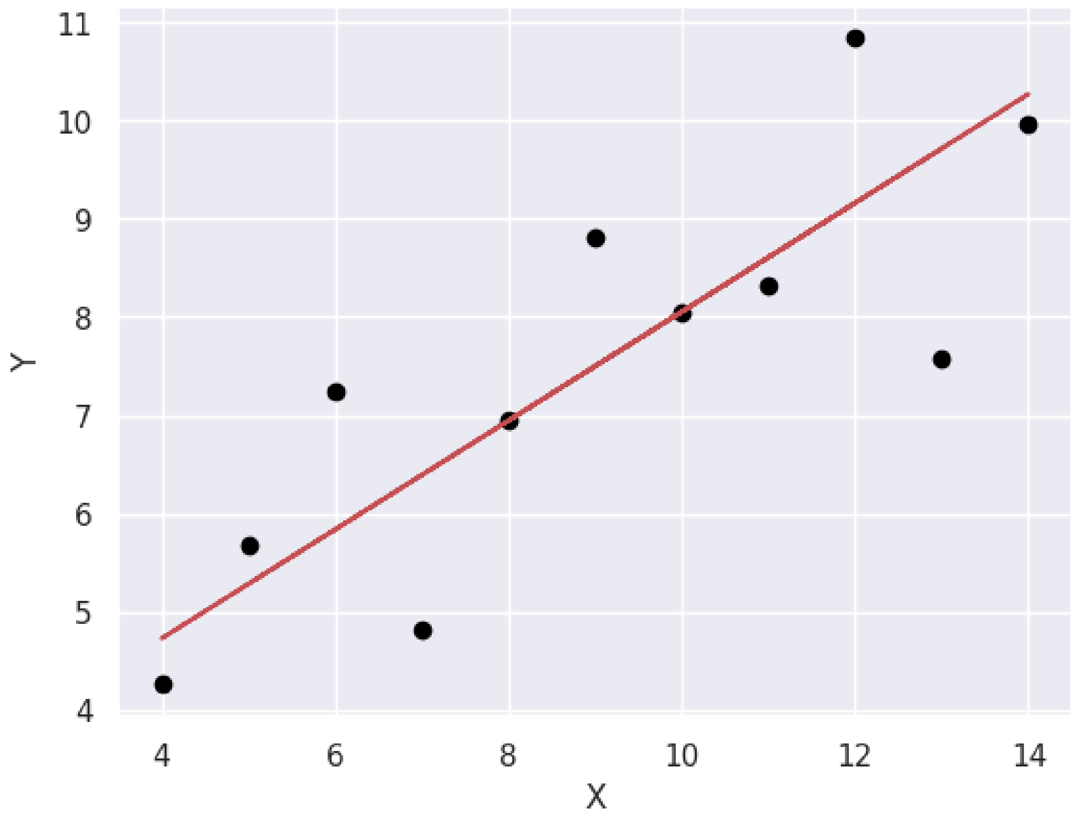
1.4 Total Least Squares Regression

Finally, a line of best fit can be made using Total least squares regression, a least squares data modeling technique in which observational errors on both dependent and independent variables are taken into account. This is done by minizing the errors perpendicular to the line, rather than just vertically. It is more complicated to implement than standard linear regression, but there is Fortran code called ODRPACK that has this efficiently implemented and wrapped scipy.odr Python module (which can beD be used out of the box). The details of odr are in the Scipy documentation and in even more detail in the ODRPACK guide.

In the code below (inspired from here uses an inital guess for the parameters, and makes a fit using total least squares regression.

[20]:
```
from scipy.odr import Model, Data, ODR
from scipy.stats import linregress
import numpy as np

def orthoregress(x, y):
    # get initial guess by first running linear regression
    linregression = linregress(x, y)

    model = Model(fit_function)

    data = Data(x, y)

    od = ODR(data, model, beta0=linregression[0:2])
    out = od.run()

    return list(out.beta)

def fit_function(p, x):
    #return y = m x + b
    return (p[0] * x) + p[1]

m, b = orthoregress(anscombe_i.x, anscombe_i.y)

# determine the line-fit
y_ortho_fit = m*anscombe_i.x + b

# plot the data
# FIX: Add plt. prefix
plt.scatter(anscombe_i.x, anscombe_i.y, color = 'black')
plt.plot(anscombe_i.x, y_ortho_fit, 'r')
plt.xlabel('X')
plt.ylabel('Y')
```

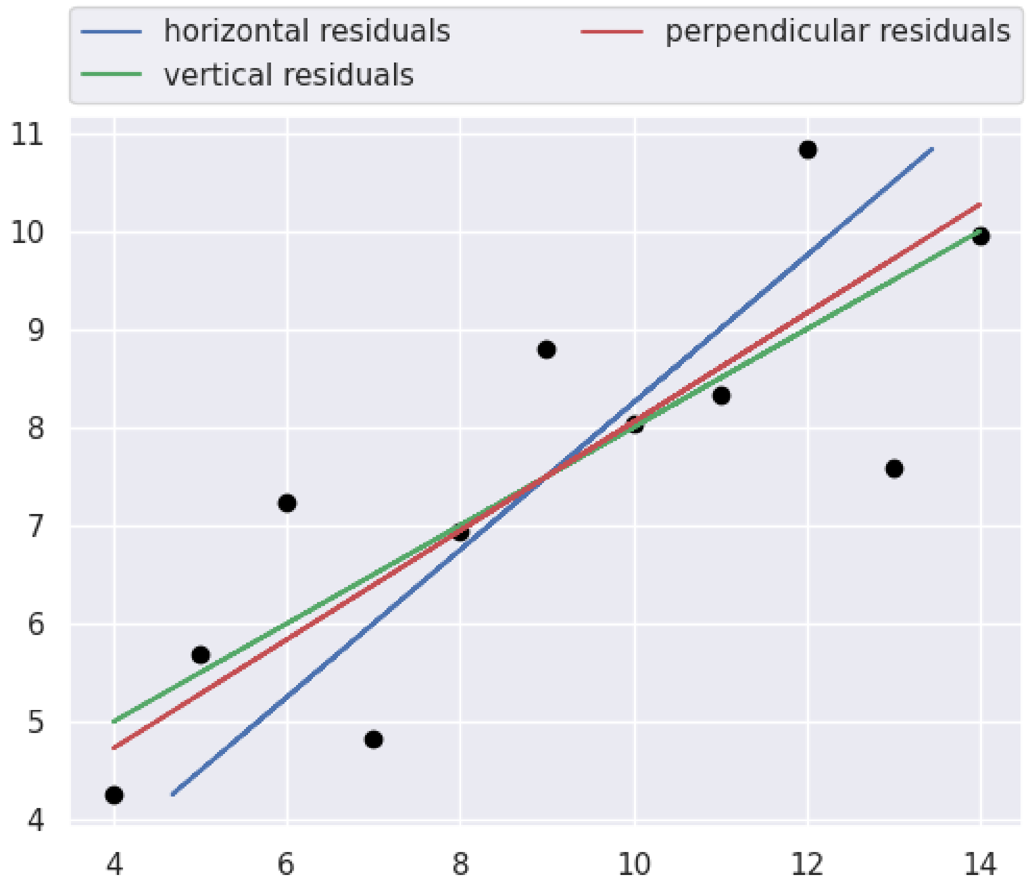
```
[20]: Text(0, 0.5, 'Y')
```



Plotting all three regression lines gives a fuller picture of the data, and comparing their slopes provides a simple graphical assessment of the correlation coefficient. Plotting the orthogonal regression line (red) provides additional information because it makes no assumptions about the dependence or independence of the variables; as such, it appears to more accurately describe the trend in the data compared to either of the ordinary least squares regression lines.

```
[21]: # FIX: Add plt. prefixes to plotting functions
      # Note: 'xfit' and 'yfit' were defined in previous cells

      plt.scatter(anscombe_i.x, anscombe_i.y, color = 'black')
      plt.plot(xfit, anscombe_i.y, 'b', label= "horizontal residuals")
      plt.plot(anscombe_i.x, yfit, 'g', label= "vertical residuals")
      plt.plot(anscombe_i.x, y_ortho_fit, 'r', label = "perpendicular residuals" )
      plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
                 ncol=2, mode="expand", borderaxespad=0.)
```

```
[21]: <matplotlib.legend.Legend at 0x75564f9d78c0>
```



1.5 Key takeaways:

1. Know the asumptions for using linear regression and ensure they are met.
2. Do not blindly apply simple linear regression, understand when to use horizonal residuals (X is dependant on y) or total least squares regression.
3. Understand the statistical significance of linear regression

1.6 Optional Further reading:

Chapter 2 (Linear regression) of Introduction to Statistical Learning

Appendix D Regression of Introduction to Data Mining

Linear models of Data Mining

Video (for using WEKA): Linear regression

1.6.1 Scikit Learn documentation:

Linear models

1.7 Homework

Now that you have seen an examples of regression using a simple linear models, see if you can predict the price of a house given the size of property from the log_regression_example.csv (found in ../datasets/log_regression_example.csv If you are unable to fit a simple linear model, try transforming variables to achieve linearity outlined in class or here

Hint: look at the log and power transform