**Linear Regression**[**¶**](http://localhost:8888/nbconvert/html/LR.ipynb?download=false#Linear-Regression)

1. Linear regression through code
2. Linear regression through scikit learn

In [25]:

xs=[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]*#input value of x*

ys=[10,12,20,22,21,25,30,21,32,34,35,30,50,45,55,60,66,64,67,72,74,80,79,84]*#input value of y*

In [26]:

**import** **matplotlib.pyplot** **as** **plt***#used for plotting*

**import** **numpy** **as** **np**

In [27]:

len(xs),len(ys)

Out[27]:

(24, 24)

In [28]:

plt.scatter(xs,ys)*#1. Plot the scatter plot of x and y to visualize the arrangement of datapoints*

plt.ylabel("dependent variable")

plt.xlabel("independent variable")

plt.show()

In [43]:

**def** slope\_intercept(x\_val,y\_val):*#Compute the slope and y coordinate*

x=np.array(x\_val)

y=np.array(y\_val)

m = ((np.mean(x)\*np.mean(y))-np.mean(x\*y))/((np.mean(x)\*np.mean(x))-np.mean(x\*x))

b = (np.mean(y)-np.mean(x)\*m)

**return** round(m,2),round(b,2)

In [47]:

print (slope\_intercept(xs,ys))

m,b = slope\_intercept(xs,ys)

(3.29, 0.88)

In [48]:

reg\_line = [(m\*x) + b **for** x **in** xs]*#plotting of the regression line*

plt.scatter(xs,ys,color="red")

plt.plot(xs,reg\_line)

plt.ylabel("dependent variable")

plt.xlabel("independent variable")

plt.title("Making a regression line")

plt.show()

In [53]:

**def** rmse(y1,y\_hat):*#error is the difference between the actual and the predicted. The root mean square error is predicted*

y\_actual=np.array(y1)

y\_pred=np.array(y\_hat)

error=(y\_actual-y\_pred)\*\*2

error\_mean=round(np.mean(error))

**import** **math**

err\_sq=math.sqrt(error\_mean)

**return** err\_sq

In [54]:

rmse(ys,reg\_line)

Out[54]:

4.58257569495584

**2. Linear regression using scikit learn**[**¶**](http://localhost:8888/nbconvert/html/LR.ipynb?download=false#2.-Linear-regression-using-scikit-learn)

In [65]:

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**from** **sklearn** **import** datasets, linear\_model

**from** **sklearn.metrics** **import** mean\_squared\_error, r2\_score

In [66]:

*# Load the diabetes dataset*

diabetes = datasets.load\_diabetes()

In [67]:

*# Use only one feature*

diabetes\_X = diabetes.data[:, np.newaxis, 2]

In [68]:

*# Split the data into training/testing sets*

diabetes\_X\_train = diabetes\_X[:-20]

diabetes\_X\_test = diabetes\_X[-20:]

In [69]:

*# Split the targets into training/testing sets*

diabetes\_y\_train = diabetes.target[:-20]

diabetes\_y\_test = diabetes.target[-20:]

In [70]:

*# Create linear regression object*

regr = linear\_model.LinearRegression()

In [71]:

*# Train the model using the training sets*

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

Out[71]:

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

In [72]:

*# Make predictions using the testing set*

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

In [73]:

*# The coefficients*

print('Coefficients: **\n**', regr.coef\_)

*# The mean squared error*

print("Mean squared error: **%.2f**"

% mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred))

*# Explained variance score: 1 is perfect prediction*

print('Variance score: **%.2f**' % r2\_score(diabetes\_y\_test, diabetes\_y\_pred))

*# Plot outputs*

plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color='black')

plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color='blue', linewidth=3)

plt.xticks(())

plt.yticks(())

plt.show()

Coefficients:

[ 938.23786125]

Mean squared error: 2548.07

Variance score: 0.47

In [ ]: