Table of Contents

Calculate linear regression, test it and plot it 1

Calculate linear regression, test it and plot it

Linear regression is a common statistical method to quantify the relationship of two quantitative variables, where one can be considered as dependent on the other. In this example, let's 1) calculate linear regression using example data and fit the regression equation, 2) predict fitted values, 3) calculate explained variation (coefficient of determination, r^2), and test the statistical significance, and 4) plot the regression line onto the scatterplot.

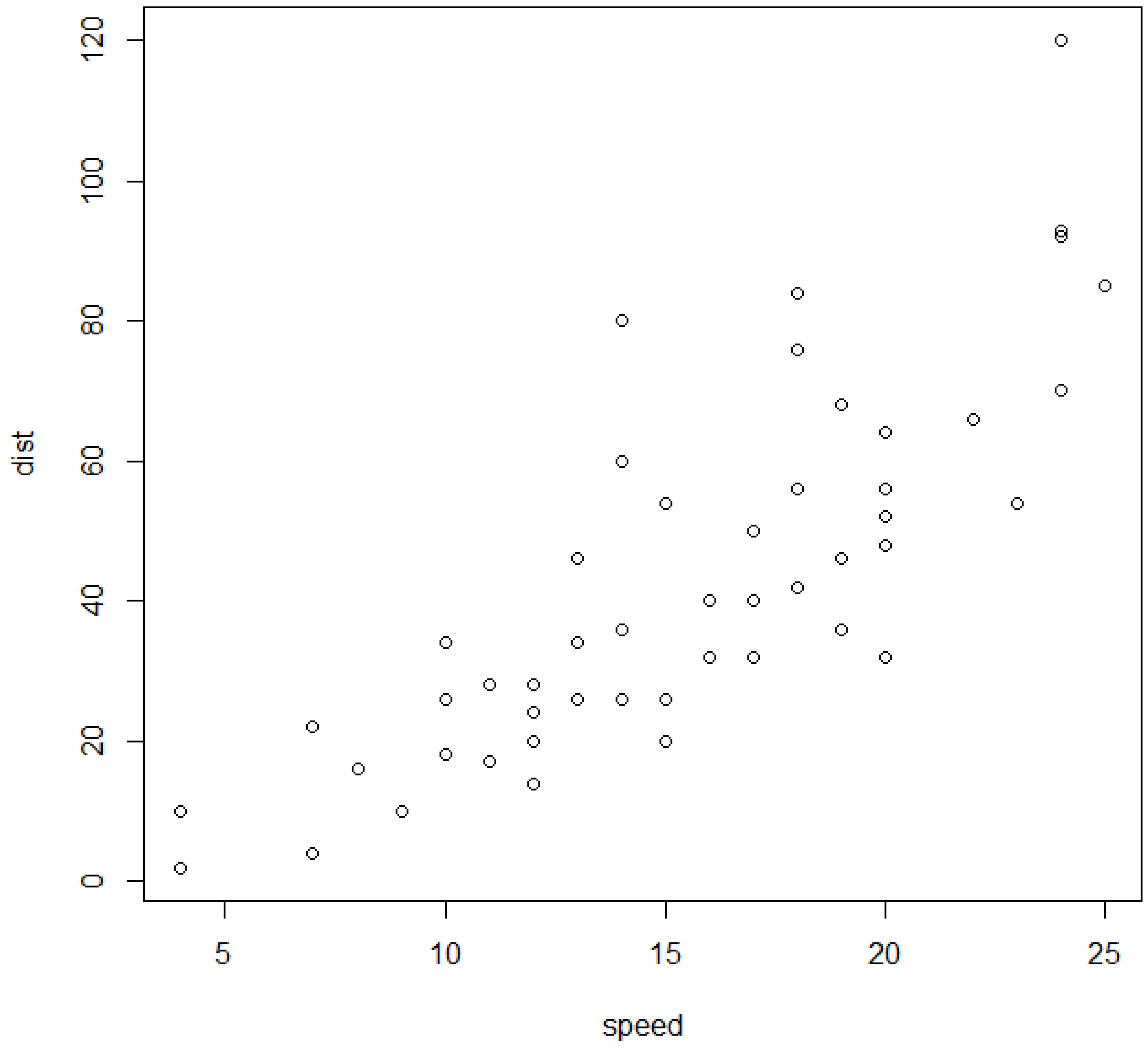
Data we will use here are in the dataset `cars`, with two variables, `dist` (the distance the car is driving after driver pressed the break) and `speed` (the speed of the car at the moment when the break was pressed).

```
head (cars)
```

```
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

Obviously, `dist` is dependent variable and `speed` is independent (increasing speed is likely increasing the breaking distance, not the other way around). We can draw the relationship, using the `plot` function with variables defined using formula interface: `dependent_variable ~ independent_variable, data = data.frame`:

```
plot (dist ~ speed, data = cars)
```



To calculate linear regression model, we use the function `lm` (linear model), which uses the same formula interface as the `plot` function:

```
lm_cars <- lm (dist ~ speed, data = dist)
lm_cars
```

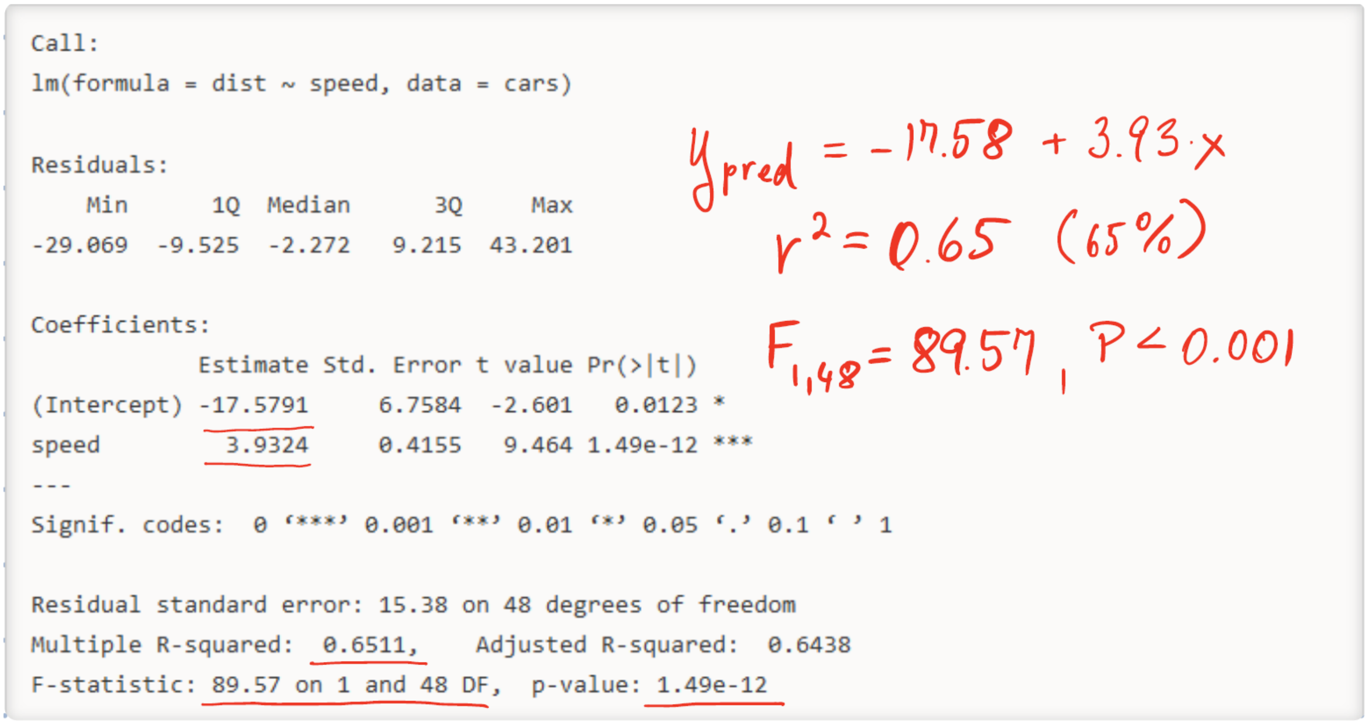
```
Call:
lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)        speed
    -17.579        3.932
```

The function 'lm' returns estimated regression coefficients, namely intercept and slope of the regression. Using these coefficients, we can create equation to calculate fitted values:

$\text{dist}_{\text{predicted}}$ = -17.579 + 3.932*speed

We can argue that the intercept coefficient does not make practical sense (at zero speed, the braking distance is obviously zero, not -17.6 feet), but let's ignore this annoying detail now and let's focus on other statistical details. We can apply generic function `summary` on the object which stores the linear model results, to get further numbers:

```
summary (lm_dist)
```



```
Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max 
-29.069  -9.525  -2.272   9.215  43.201 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -17.5791     6.7584  -2.601   0.0123 *  
speed         3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438 
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

$y_{pred} = -17.58 + 3.93 \cdot x$

$r^2 = 0.65$ (65%)

$F_{1,48} = 89.57$, $P < 0.001$

Commented output reports the most important variables: apart to the regression coefficients also the coefficient of determination (r^2), and the value of F statistic and P-value. These are usually numbers you need to include when you are reporting the results of the regression. Some details below:

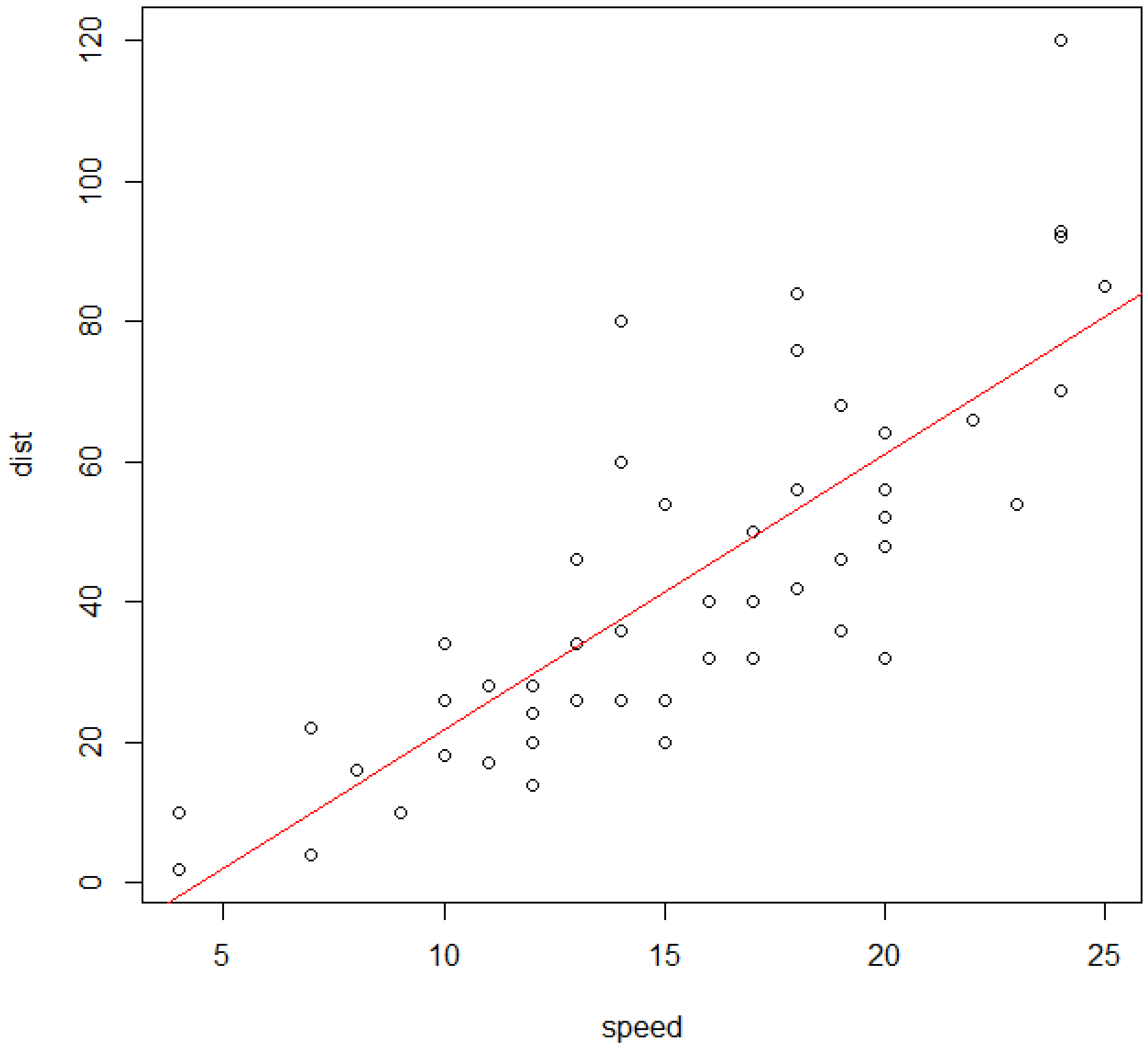
1. **Parameters of the regression equation** are important if you plan to predict the values of the dependent variable for a certain value of the explanatory variable. For example, if I want to know what would be the predicted braking distance if the car drives 15 mph, I can calculate it: $\text{dist}_{\text{predicted}}$ = -17.579 + 3.932*15 = 41.40 ft (note that the values of speed and distance are in imperial units - mph = miles per hour, where 1 mph = 1.61 km/h, and ft = feet, where 1 ft = 0.31 m).
2. **Coefficient of determination** quantifies how much variation in the dependent variable `dist` was explained by given explanatory variable (`speed`). You can imagine that braking distance, in fact, depends on many other factors than just speed of the car at the moment of pressing the brake - it also depends on how overused are wheels, whether the road is dry or wet or icy, how quickly driver pressed the brake etc. Here, speed explains 65% of the variation, which is more than half, but far from the whole - almost 35% of variation remains unexplained.
3. Before we start to interpret the results, we often want to make sure that there is something to interpret. Even if the explanatory variable is a randomly generated value, it will explain the non-zero amount of variation (you can try it to see!). This means that we need to know whether our explained variation (65%) is higher than would be variation explained by some random variable. For this, we may **test the null hypothesis** H0 that there is no relationship between the dependent and explanatory variable, and if rejected, we can confirm that this relationship is significant. In the case of a parametric test of significance for regression, we calculate F-value and use it (together with appropriate degrees of freedom) to look up the appropriate P-value. P-

value is the probability of obtaining the value of test statistic (here F-value) equal or more extreme than the observed one, even if the null hypothesis is true. The lower is the P-value, the more statistically significant is the result. In our case, P-value is very low, namely 0.00000000000149, very close to zero (you don't want to write it in this long format!). When reporting the P-value, we can either report the original value, or we report that the value is smaller than the arbitrary selected threshold, e.g. P < 0.001 (three thresholds are commonly used, P < 0.001, P < 0.01 and P < 0.05; if the P-value is e.g. 0.032, you can report P < 0.05).

It is important to report all relevant statistical values. For example, when you report P-value, you need to also include the value of the test statistic (here F-value), and also the numbers of degrees of freedom (because the same F-value with different degrees of freedom leads to different P-value).

Finally, we can plot the regression line onto the plot. The simplest way to do it is using the function `abline`, which can directly use the variables from the `lm` object. You may know that function `abline` has arguments v and h for coordinates of vertical and horizontal line, respectively, but it has also arguments a and b for regression coefficients (intercept and slope, respectively). It can extract these values directly from the result result of `lm` function:

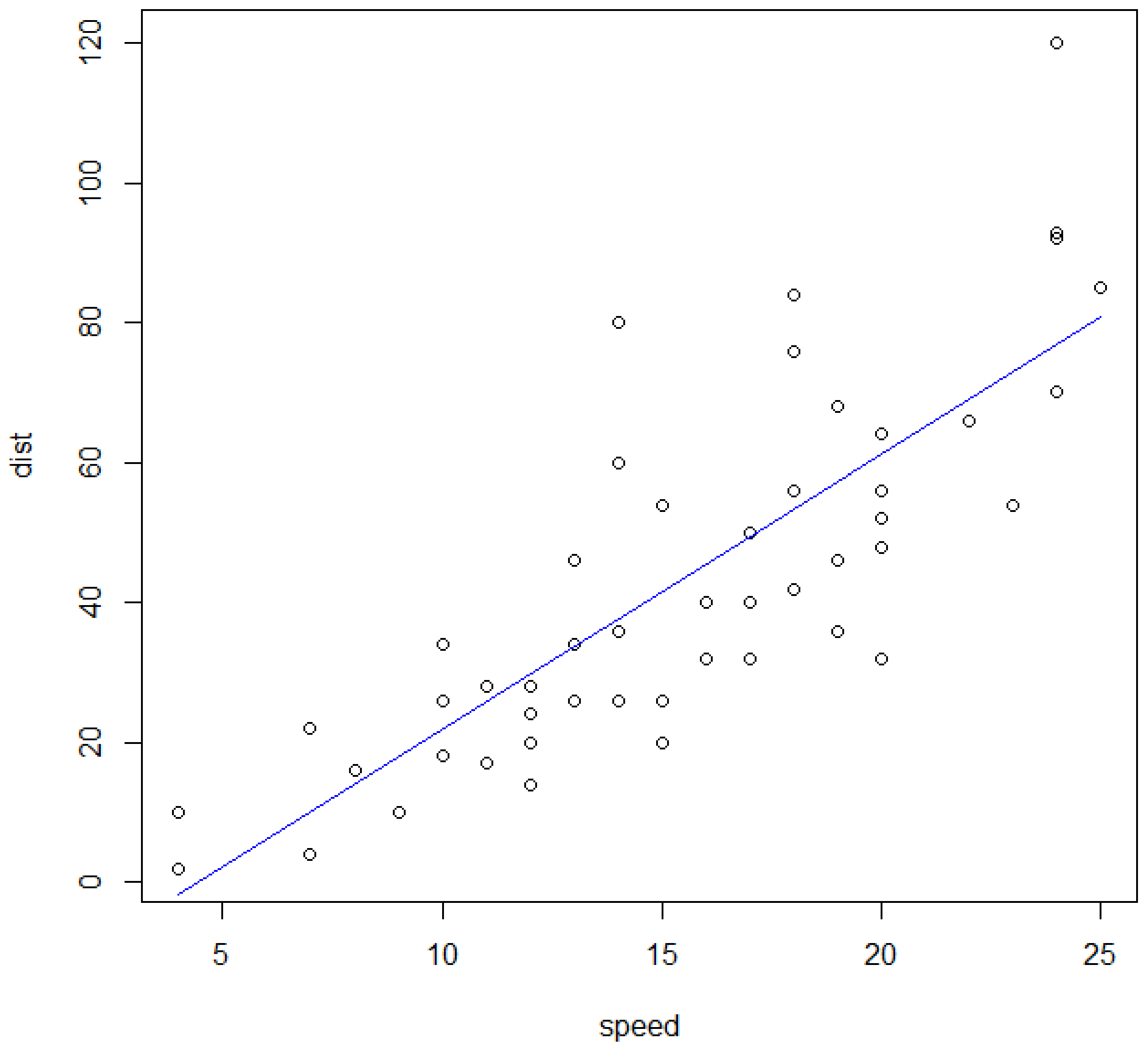
```
plot (dist ~ speed, data = cars)
lm_cars <- lm (dist ~ speed, data = cars)
abline (lm_cars, col = 'red')
```



The more general solution is to use the parameters of linear regression and predict the values of dependent variables. Since we are considering linear response here, we can just predict the values of the dependent variable for minimum and maximum of the explanatory variable (i.e. braking distance for minimum and maximum measured speed within the experiment). For this purpose, there is the function `predict`, which takes two arguments: the object generated by `lm` function (here `lm_dist`), and `newdata` with the values of the explanatory variable for which dependent variable should be predicted. Important: `newdata` must be a list, and the name of the component must be the same as the name of the explanatory variable in the formula you use in `lm` (here `speed`). If the name is different (e.g. in `lm` you use `speed`, but in `predict` you use `x`), then the predict function will ignore the argument and predict values for all numbers inside the explanatory variable (here 50 speed measurements).

```
plot (dist ~ speed, data = cars)
lm_cars <- lm (dist ~ speed, data = cars)
speed_min_max <- range (cars$speed)  # range of values, the same as: c(min
(cars$speed), max (cars$speed))
dist_pred <- predict (lm_cars, newdata = list (speed = speed_min_max))
```

```
lines (dist_pred ~ speed_min_max, col = 'blue')
```



You can see that the result (blue regression line) is very similar to the one drawn by `abline` function (red regression line above), but not identical - the red line goes from the edge to edge of the plotting region, while the blue line goes only from minimum to maximum value of `speed`.

Benefit of the `predict` solution will become more apparent if we consider that maybe the relationship between the dependent and explanatory variable is not linear, and would be better done by curvilinear shape. For this, we can use `poly` function, included in the formula of `lm` model, which allows creating polynomial regression (ie y = b0 + b1*x + b2*x^2 for polynomial of the second degree, allowing the curve to "bend" once).

```
lm_cars_2 <- lm (dist ~ poly (speed, degree = 2), data = cars)
lm_cars_2
```

```
Call:
lm(formula = dist ~ poly(speed, degree = 2), data = cars)
```

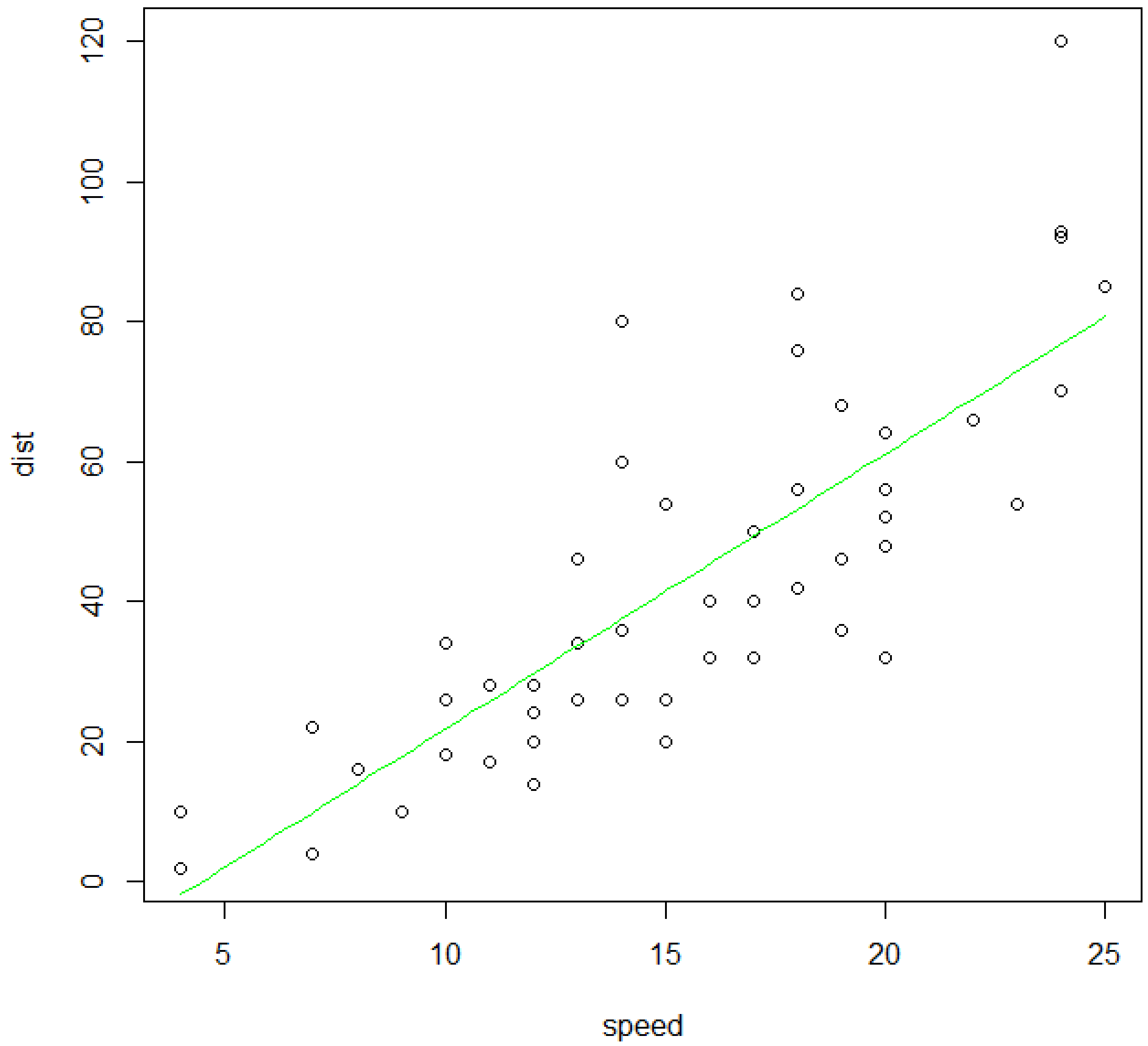
```
Coefficients:
            (Intercept)  poly(speed, degree = 2)1  poly(speed, degree = 2)2
                  42.98                    145.55                     23.00
```

The equation of this polynomial regression si then

$\text{dist}_{\text{predicted}} = 42.98 + 145.55*\text{speed} + 23.00*\text{speed}^2$

To draw the predicted polynomial regression curve, we need to use `predict` (because `abline` can draw only straight lines), and additionally we need to define enough points within the curve to make the curvature smooth.

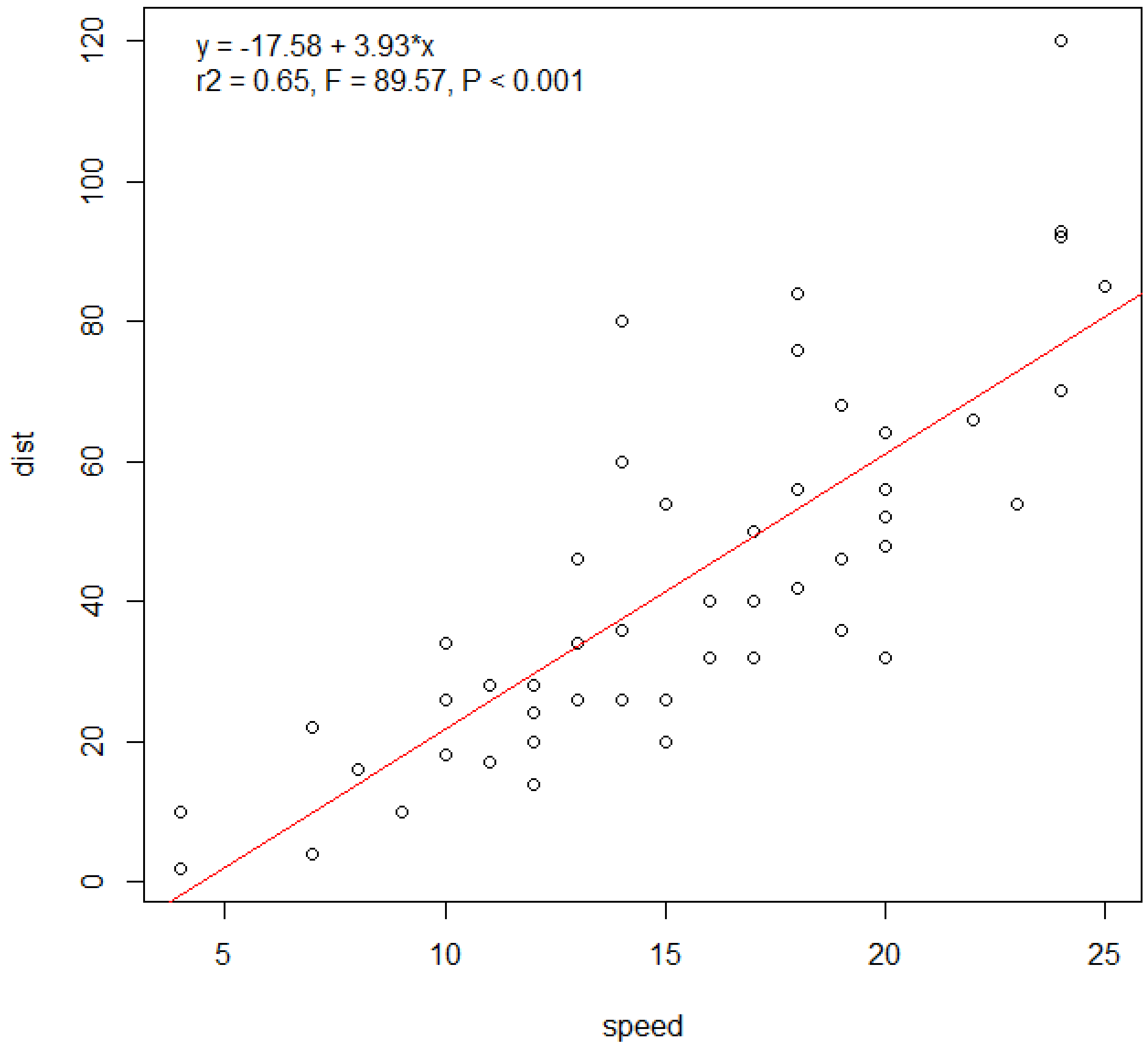
```
plot (dist ~ speed, data = cars)
speed_min_max_seq <- seq (min (cars$speed), max (cars$speed), length = 100)
dist_pred_2 <- predict (lm_cars, newdata = list (speed = speed_min_max_seq))
lines (dist_pred_2 ~ speed_min_max_seq, col = 'green')
```



You see that the green line is practically straight - in fact, the pattern is very close to linear, so even if we fit the polynomial curve, it will be almost straight.

Finally, we can report the statistical values of the linear regression as the legend inside the figure (this is suitable, however, only if there is enough space inside the drawing area). Let's do it for linear (but not polynomial) regression, and include parameters of the regression, r^2, F-value and P-value:

```
plot (dist ~ speed, data = cars)
lm_cars <- lm (dist ~ speed, data = cars)
abline (lm_cars, col = 'red')
legend ('topleft', legend = c('y = -17.58 + 3.93*x', 'r2 = 0.65, F = 89.57,
P < 0.001'), bty = 'n')
```


y = -17.58 + 3.93*x
r2 = 0.65, F = 89.57, P < 0.001
dist
speed