Simple & Multiple

Before we begin, let's open up the `regression.r` file within RStudio. See instructions below:

info

Open the `regression.r` file

Within RStudio, open the `regression.r` file by selecting: `File` —> `Open File...` —> `code` —> `describe` —> `regression.r`

Simple and Multiple Linear Regression

If you determine that your data sets are associated or correlated with each other, you can perform linear regression on them. Doing so enables you to **predict** additional data based on that regression model. There is **single linear** when you have one **independent** variable (x) and one **dependent** variable (y). And there is **multiple linear** regression when you have **multiple** independent variables (x's) but only one dependent variable (y). You can think of a *dependent* variable as an event that is **influenced** by the *independent* variable. For example, when comparing how much sunlight a plant receives versus its growth in length, it is probably better to say that the plant's growth is more likely dependent on the amount of sunlight than the reverse. Thus, the plant's length is the dependent variable and the amount of sunlight is the independent variable.

The basic syntax for simple linear regression is:

```
summary(lm(formula = y ~ x))
```

And the basic syntax for multiple linear regression is:

```
summary(lm(formula = y ~ x1 + x2))
```

Where:
* y represents the **dependent** variable or vector
* x's represent the **independent** variable(s) or vector(s)

If you have not extracted the data as vectors, you can specify each x and y as columns from the data set and then specify the data frame they come from using `data`.

```
summary(lm(formula = df$y ~ df$x1 + df$x2, data = df))
```

Simple Regression

For example, if you have the following data:

```
month <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
spend <- c(1000, 4000, 5000, 4500, 3000, 4000, 9000, 11000,
        15000, 12000, 7000, 3000)
sales <- c(9914, 40487, 54324, 50044, 34719, 42551, 94871,
        118914, 158484, 131348, 78504, 36284)
```

Note that in order to perform linear regression analysis, the data **must be represented as numerics**. Such a calculation cannot happen with categorical data.

You can use:

```
print(summary(lm(formula = sales ~ spend)))
```

To find:

```
Call:
lm(formula = sales ~ spend)

Residuals:
   Min      1Q Median      3Q     Max
 -3385  -2097    258    1726    3034

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1383.4714  1255.2404   1.102    0.296
spend         10.6222     0.1625  65.378 1.71e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2313 on 10 degrees of freedom
Multiple R-squared:  0.9977,    Adjusted R-squared:  0.9974
F-statistic:  4274 on 1 and 10 DF,  p-value: 1.707e-14
```

The `Pr(>|t|)` of `1.71e-14` suggests that `sales` is highly associated or dependent on `spend`.

Multiple Regression

To perform multiple regression by accounting for `month` as well, simply add `month` to the formula:

```
print(summary(lm(formula = sales ~ spend + month)))
```

Which returns:

```
Call:
lm(formula = sales ~ spend + month)

Residuals:
     Min       1Q   Median       3Q      Max 
-1793.73 -1558.33    -1.73  1374.19  1911.58 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -567.6098  1041.8836  -0.545  0.59913    
spend         10.3825     0.1328  78.159 4.65e-14 ***
month        541.3736   158.1660   3.423  0.00759 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1607 on 9 degrees of freedom
Multiple R-squared:  0.999, Adjusted R-squared:  0.9988 
F-statistic:  4433 on 2 and 9 DF,  p-value: 3.368e-14
```

The `Pr(>|t|)` of `0.00759` suggests that `sales` is also highly associated or dependent on `month` when considering **both** `spend` and `month`. This means that the amount of `sales` is strongly correlated with both `spend` and `month`.

Predict

Once you have your regression analysis model, you can use the `predict()` function to predict what your dependent variable will be when you have a new set of independent variable values. The basic syntax for `predict()` is:

```
predict(object = linear, newdata = new_spend)
```

Where:
* `linear_model` represents my previous regression analysis (using `lm()`)
* `new_spend` represents the new data I want to be used in the prediction analysis

For example, if I want to calculate what values I will get for `sales` given new values for `spend`, I'd first create a data frame to hold the new `spend` values:

```
new_spend <- data.frame(spend = c(1200, 5000, 4000, 3500, 5000,
        4000,
                                  8000, 9000, 10000, 13000, 6500,
        3500))
```

Then I'd copy over the previous calculated regression analysis between `sales` and `spend` and store that information in a variable called `linear_model`. And finally, I'll call the `predict()` function.

```
spend <- c(1000, 4000, 5000, 4500, 3000, 4000, 9000, 11000,
        15000, 12000, 7000, 3000)
sales <- c(9914, 40487, 54324, 50044, 34719, 42551, 94871,
        118914, 158484, 131348, 78504, 36284)

linear_model <- lm(formula = sales ~ spend)

print(predict(object = linear_model, newdata = new_spend))
```

Result:

```
        1          2          3          4          5          6
        7          8          9         10         11         12
 14130.11   54494.45   43872.25   38561.16   54494.45   43872.25
        86361.04   96983.23 107605.43 139472.01   70427.74
        38561.16
```

Scroll right to see the rest of the data. **Note** that the numbers at the top (`1` through `12`) do not correspond to the `month` data. Rather, they are there to show how many columns of data there are.

If you compare the original `sales` data with the "new" predicted sales data, you'll see that they are somewhat close in value. For better comparison, use the following code to extract the data from the prediction analysis, change the column name of the prediction analysis, store that information into a new variable called `new_sales`, and finally create a data frame `compare_data` that includes both the old and new `sales` data.

Full Code

```r
new_spend <- data.frame(spend = c(1200, 5000, 4000, 3500, 5000,
        4000,
                                  8000, 9000, 10000, 13000, 6500,
        3500))

spend <- c(1000, 4000, 5000, 4500, 3000, 4000, 9000, 11000,
        15000, 12000, 7000, 3000)
sales <- c(9914, 40487, 54324, 50044, 34719, 42551, 94871,
        118914, 158484, 131348, 78504, 36284)

linear_model <- lm(formula = sales ~ spend)

new_data <- data.frame(predict(object = linear_model, newdata =
        new_spend))
# store prediction data as data frame

colnames(new_data) <- c("new_sales")
# rename the column name of the data to "new_sales"

new_sales <- new_data$new_sales
# store this data as new variable called new_sales

compare_data <- data.frame(sales, new_sales)
# create new data frame with old sales and new sales

print(compare_data)
# print to see the data side by side
```

Result:

```
    sales new_sales
1    9914  14130.11
2   40487  54494.45
3   54324  43872.25
4   50044  38561.16
5   34719  54494.45
6   42551  43872.25
7   94871  86361.04
8  118914  96983.23
9  158484 107605.43
10 131348 139472.01
11  78504  70427.74
12  36284  38561.16
```

Logistic

Logistic Regression

Logistic regression is performed when you have **categorical** data in the mix. For example, if you want to determine if a diabetic diagnosis is dependent on (or strongly associated with) a person's blood pressure, you should use logistic regression.

The basic syntax for logistic regression is:

```
summary(glm(y ~ x, family = "binomial"))
```

Where:
* `y` represents the **dependent** variable
* `x`'s represent the **independent** variable(s)
* `"binomial"` represents that there is categorical data

Logistic Regression

For example, if you have the following data:

```
d <- read.csv("data/diabetes.csv")

outcome <- d$Outcome
bp <- d$BloodPressure
```

You can use:

```
print(summary(glm(outcome ~ bp, family = "binomial")))
```

NOTE: `tested_positive` means a person has diabetes (positive diagnosis) and `tested_negative` means a person does not have diabetes (negative diagnosis).
Data Source: https://www.kaggle.com/saurabh00007/diabetescsv

To find:

```
Call:
glm(formula = outcome ~ bp, family = "binomial")

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.0797  -0.9389  -0.9000   1.4097   1.6838

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.140092   0.299822  -3.803 0.000143 ***
bp           0.007425   0.004141   1.793 0.072994 .
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 993.48  on 767  degrees of freedom
Residual deviance: 990.13  on 766  degrees of freedom
AIC: 994.13

Number of Fisher Scoring iterations: 4
```

The `Pr(>|t|)` of `0.072994` is over `0.05` which means we fail to reject the null hypothesis. Thus, this suggests that `outcome` (diabetic diagnosis) is not very dependent on `bp` (blood pressure) which means there isn't a significant association between the two data sets.

Predict

Like with linear regression analysis, you can use the `predict()` function to determine new dependent variable values based on new independent variable values. The basic syntax is almost the same:

```
predict(object = logistic_model, newdata = new_data, type =
        "response")
```

Where:
* `logistic_model` represents my previous regression analysis (using `glm()`)
* `new_data` represents the new data I want to be used in the prediction analysis
* `"response"` is used for logistic analysis

However, due to the fact that we did not find any significant association between `outcome` and `bp`, it does not make sense to try and predict new values for these variables.