Author :- Rakesh Ramesh

# Multiple Linear Regression

Sunday, January 15, 2017 11:22 AM

## Data

| R&D Spend | Administration | Marketing Spe | State | Profit |
|---|---|---|---|---|
| 165349.2 | 136897.8 | 471784.1 | New York | 192261.83 |
| 162597.7 | 151377.59 | 443898.53 | California | 191792.06 |
| 153441.51 | 101145.55 | 407934.54 | Florida | 191050.39 |
| 144372.41 | 118671.85 | 383199.62 | New York | 182901.99 |
| 142107.34 | 91391.77 | 366168.42 | Florida | 166187.94 |
| 131876.9 | 99814.71 | 362861.36 | New York | 156991.12 |
| 134615.46 | 147198.87 | 127716.82 | California | 156122.51 |
| 130298.13 | 145530.06 | 323876.68 | Florida | 155752.6 |
| 120542.52 | 148718.95 | 311613.29 | New York | 152211.77 |
| 123334.88 | 108679.17 | 304981.62 | California | 149759.96 |
| 101913.08 | 110594.11 | 229160.95 | Florida | 146121.95 |
| 100671.96 | 91790.61 | 249744.55 | California | 144259.4 |
| 93863.75 | 127320.38 | 249839.44 | Florida | 141585.52 |
| 91992.39 | 135495.07 | 252664.93 | California | 134307.35 |
| 119943.24 | 156547.42 | 256512.92 | Florida | 132602.65 |
| 114523.61 | 122616.84 | 261776.23 | New York | 129917.04 |
| 78013.11 | 121597.55 | 264346.06 | California | 126992.93 |
| 94657.16 | 145077.58 | 282574.31 | New York | 125370.37 |
| 91749.16 | 114175.79 | 294919.57 | Florida | 124266.9 |
| 86419.7 | 153514.11 | 0 | New York | 122776.86 |
| 76253.86 | 113867.3 | 298664.47 | California | 118474.03 |
| 78389.47 | 153773.43 | 299737.29 | New York | 111313.02 |
| 73994.56 | 122782.75 | 303319.26 | Florida | 110352.25 |

We need to understand how the columns contribute to the profit. And where should a VC invest (in which areas) in order to get the most profit and the priority order of investment.

**Multiple Linear Regression**

Dependent variable (DV) Independent variables (IVs)

$$y = b_0 + b_1{}^*x_1 + b_2{}^*x_2 + \dots + b_n{}^*x_n$$

## Assumptions of a Linear Regression:

1. Linearity
2. Homoscedasticity
3. Multivariate normality
4. Independence of errors
5. Lack of multicollinearity

Before we start building models, we need to check for these and ensure that these assumptions hold on the dataset. This won't be covered here.

Here, profit is out dependent variable and the rest of them are independent variables. We need to build a Linear regression m odel to incorporate this information into the equation and generate a model.

| Profit | R&D Spend | Admin | Marketing | State | New York | California |
|---|---|---|---|---|---|---|
| 192,261.83 | 165,349.20 | 136,897.80 | 471,784.10 | New York | 1 | 0 |
| 191,792.06 | 162,597.70 | 151,377.59 | 443,898.53 | California | ~~0~~ | 1 |
| 191,050.39 | 153,441.51 | 101,145.55 | 407,934.54 | California | ~~0~~ | 1 |
| 182,901.99 | 144,372.41 | 118,671.85 | 383,199.62 | New York | 1 | 0 |
| 166,187.94 | 142,107.34 | 91,391.77 | 366,168.42 | California | ~~0~~ | 1 |

$$y = b_0 + b_1{}^*x_1 + b_2{}^*x_2 + b_3{}^*x_3 + \ ???$$

Here, state is a categorical variable. We need to create a dummy variable for this.

**Dummy Variable Trap** - We should never include both dummy variables at the same time. That is if the possible values for state are only New York and California,

then we should never have dummy variables to represent both. Because when the value is not New York then it has to be Califor nia. So if we include both, then we are introducing another variable into the equation which does the same thing dummy variable New York would do.
D2 = 1 - D1
**To sum up, whenever we are building a model, always emit one dummy variable !!**

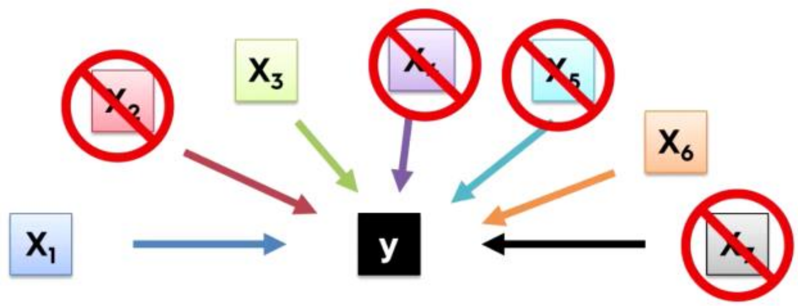
| Profit | R&D Spend | Admin | Marketing | State | Dummy Variables: New York | Dummy Variables: California |
|---|---|---|---|---|---|---|
| 192,261.83 | 165,349.20 | 136,897.80 | 471,784.10 | New York | 1 | 0 |
| 191,792.06 | 162,597.70 | 151,377.59 | 443,898.53 | California | 0 | 1 |
| 191,050.39 | 153,441.51 | 101,145.55 | 407,934.54 | California | 0 | 1 |
| 182,901.99 | 144,372.41 | 118,671.85 | 383,199.62 | New York | 1 | 0 |
| 166,187.94 | 142,107.34 | 91,391.77 | 366,168.42 | California | 0 | 1 |

$$y = b_0 + b_1{}^*x_1 + b_2{}^*x_2 + b_3{}^*x_3 + b_4{}^*D_1 + \cancel{b_5{}^*D_2}$$

**Always omit one dummy variable**

## Multi Linear Regression

Here the dependent variable is dependent on multiple independent variables. So we need to figure out which among these we nee d to keep and which one's can we remove from the dependency list. This is done to make the model reliable and make it easier for understanding and represen tation. We need to determine how the variables contribute to the dependent variables. We need to only keep the important ones.




## 5 methods of building models:

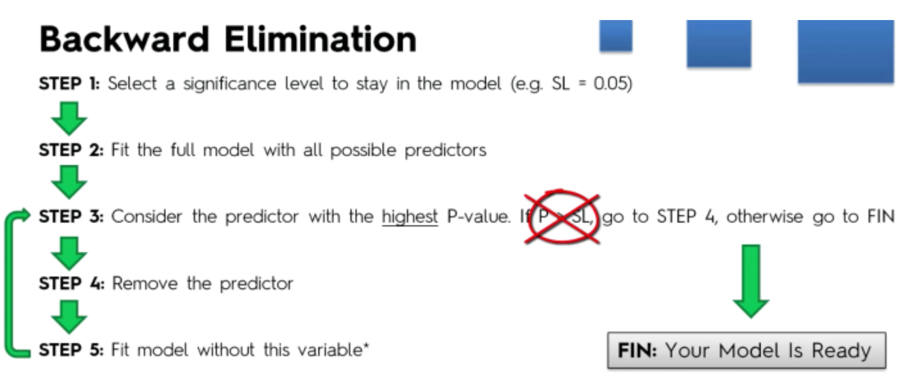
1. All-in
2. Backward Elimination
3. Forward Selection
4. Bidirectional Elimination
5. Score Comparison

(2–4: Stepwise Regression)

By default Stepwise regression means Bidirectional elimination.

## "All-in" – cases:

- Prior knowledge; OR
- You have to; OR
- Preparing for Backward Elimination



## Forward Selection

**STEP 1:** Select a significance level to enter the model (e.g. SL = 0.05)

**STEP 2:** Fit all simple regression models **y ~ $x_n$** Select the one with the lowest P-value

**STEP 3:** Keep this variable and fit all possible models with one extra predictor added to the one(s) you already have

**STEP 4:** Consider the predictor with the lowest P-value. If ~~P < SL~~, go to STEP 3, otherwise go to FIN

**FIN:** Keep the previous model


## Forward Selection

**STEP 1:** Select a significance level to enter the model (e.g. SL = 0.05)

**STEP 2:** Fit all simple regression models **y ~ $x_n$** Select the one with the lowest P-value

**STEP 3:** Keep this variable and fit all possible models with one extra predictor added to the one(s) you already have

**STEP 4:** Consider the predictor with the lowest P-value. If ~~P < SL~~, go to STEP 3, otherwise go to FIN


## Bidirectional Elimination

**STEP 1:** Select a significance level to enter and to stay in the model
e.g.: SLENTER = 0.05, SLSTAY = 0.05

**STEP 2:** Perform the next step of Forward Selection (new variables must have: P < SLENTER to enter)

**STEP 3:** Perform ALL steps of Backward Elimination (old variables must have P < SLSTAY to stay)

**STEP 4:** No new variables can enter and no old variables can exit

**FIN:** Your Model Is Ready

## All Possible Models

**STEP 1:** Select a criterion of goodness of fit (e.g. Akaike criterion)

**STEP 2:** Construct All Possible Regression Models: $2^N$-1 total combinations

**STEP 3:** Select the one with the best criterion

**FIN:** Your Model Is Ready

All possible models is not a practical approach.
We will be using Backward Elimination because this is the fastest one among all of them.

### <u>Python</u>

Data preprocessing steps. Here, there is one categorical column data we need to encode. So this needs to be taken into consideration as well while pre-processing.

```
#Multiple Linear Regression

#Importing the libraries
Import numpy as np
Import matplotlib.pyplot as plt
Import pandas as pd
```

---

```
#Importing the dataset
Dataset = pd.read_csv('Fifty_Startups.csv')
X = dataset.iloc[:,:-1].values
Y = dataset.iloc[:,4].values
```

```
#Encoding categorical variables
```

```
From sklearn.preprocessing import LabelEncoder
label_encoder_X = LabelEncoder()
X[:,3] = label_encoder_X.fit_transform(X[:,3])
```

```
From sklearn.preprocessing import OneHotEncoder
Onehotencoder = OneHotEncoder(categorical_features=[3])
X = Onehotencoder.fit_transform(X).toarray()
```

```
#Dummy Variable trap Elimination
X=X[:,1:]
```

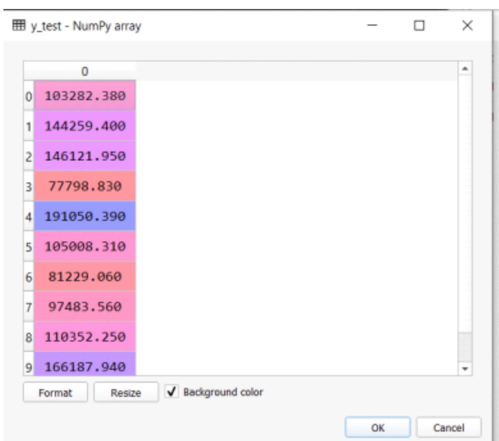
```
#Splitting the dataset into the Trainingset and Testset
From sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Fitting Multi Linear Regression model and predicting is the same as how we do it with Simple Linear Regression models. Even t he library calls are the same.

```
#No features caling required.Library will do it for us.
```

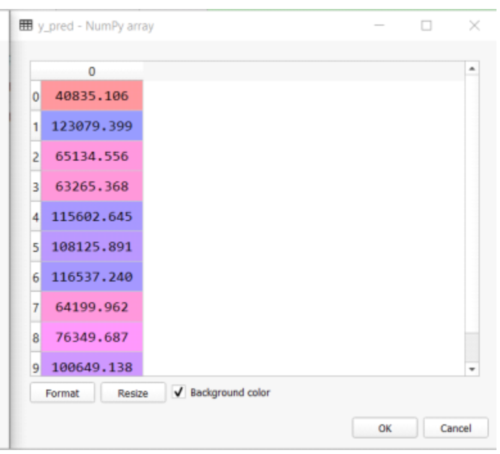
```
#Fitting Multi Linear Regression Model to Training Set
From sklearn.linear_model import LinearRegression
Regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
#Predicting the Test Set Results
y_pred = regressor.predict(X_test)
```


y_test - NumPy array

| | 0 |
|---|---|
| 0 | 103282.380 |
| 1 | 144259.400 |
| 2 | 146121.950 |
| 3 | 77798.830 |
| 4 | 191050.390 |
| 5 | 105008.310 |
| 6 | 81229.060 |
| 7 | 97483.560 |
| 8 | 110352.250 |
| 9 | 166187.940 |




y_pred - NumPy array

| | 0 |
|---|---|
| 0 | 40835.106 |
| 1 | 123079.399 |
| 2 | 65134.556 |
| 3 | 63265.368 |
| 4 | 115602.645 |
| 5 | 108125.891 |
| 6 | 116537.240 |
| 7 | 64199.962 |
| 8 | 76349.687 |
| 9 | 100649.138 |



Building the optimal model using Backward Elimination

```
# Building optimal model using Backward Elimination
# Remove variables that are not statistically significant
```

```
import statsmodels.formula.api as sm
```

Here, we need to add a column of 1's. This denotes the constant for bias. Without it, the model assumes there is no constant/ bias term.

```
X = np.append(arr = np.ones(shape=(50,1)).astype(int), values = X, axis = 1)
```

Here, we use numpy's append command. Input array is X and the appending value is a column of 1's of type Integer (Integer nee ds to be mentioned to avoid typecast errors). Axis 1 means append a column, Axis 0 means append a row.

```
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
# OLS - Ordinary Least Squares
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Summary can be used to provide details related to the variables involved. We use OLS model with endog as y (output) and exog as X_opt (optimal X with minimum number of parameters). We fit the data using these. This will result in a summary where we can find out the P -values in order to filter the columns.

```
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const       5.013e+04   6884.820      7.281      0.000      3.62e+04   6.4e+04
x1           198.7888   3371.007      0.059      0.953     -6595.030  6992.607
x2           -41.8870   3256.039     -0.013      0.990     -6604.003  6520.229
x3             0.8060      0.046     17.369      0.000         0.712     0.900
x4            -0.0270      0.052     -0.517      0.608        -0.132     0.078
x5             0.0270      0.017      1.574      0.123        -0.008     0.062
```

As seen here, the column 2 has the highest P-value and hence we start things off by removing it first.
After eliminating column 2, we get

```
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const       5.011e+04   6647.870      7.537      0.000      3.67e+04  6.35e+04
x1           220.1585   2900.536      0.076      0.940     -5621.821  6062.138
x2             0.8060      0.046     17.606      0.000         0.714     0.898
x3            -0.0270      0.052     -0.523      0.604        -0.131     0.077
x4             0.0270      0.017      1.592      0.118        -0.007     0.061
```

Now the next highest is Column 1. Which is significantly higher than 0.05 threshold for P-value. Hence we remove it.
Similarly, removing columns which exceed 0.05 we get

```
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const       4.698e+04   2689.933     17.464      0.000      4.16e+04  5.24e+04
x1             0.7966      0.041     19.266      0.000         0.713     0.880
x2             0.0299      0.016      1.927      0.060        -0.001     0.061
```

We have an option to either include x2 or not. But it's P-value is very close to 0.05 and might increase accuracy.
We can use metrics like R-squared and Adjusted R-squared to help decide whether we need to keep or not the variable x2.

## R

Data preprocessing steps are similar to what we had done previously. We have a categorical variable which needs to be converted to a numerical representation. Then, we split it into test and train sets

```
# Importing the dataset
dataset = read.csv('Fifty_Startups.csv')
# Encoding categorical data
dataset$State = factor(dataset$State,
                       levels = c('New York', 'California', 'Florida'),
                       labels = c(1, 2, 3))
# Splitting the dataset into the Training set and Test set
library(caTools)
set.seed(123)
split = sample.split(dataset$Profit, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

| R.D.Spend | Administration | Marketing.Spend | State | Profit |
|---|---|---|---|---|
| 165349.20 | 136897.80 | 471784.10 | 1 | 192261.83 |
| 162597.70 | 151377.59 | 443898.53 | 2 | 191792.06 |
| 153441.51 | 101145.55 | 407934.54 | 3 | 191050.39 |
| 144372.41 | 118671.85 | 383199.62 | 1 | 182901.99 |
| 142107.34 | 91391.77 | 366168.42 | 3 | 166187.94 |
| 131876.90 | 99814.71 | 362861.36 | 1 | 156991.12 |
| 134615.46 | 147198.87 | 127716.82 | 2 | 156122.51 |
| 130298.13 | 145530.06 | 323876.68 | 3 | 155752.60 |
| 120542.52 | 148718.95 | 311613.29 | 1 | 152211.77 |
| 123334.88 | 108679.17 | 304981.62 | 2 | 149759.96 |
| 101913.08 | 110594.11 | 229160.95 | 3 | 146121.95 |

Now for fitting Multi Linear regression model to our training set

```
# Fitting Multiple Linear Regression to the Training Set
regressor = lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend + State,
               data = training_set)
```

Here, the formula represents the relationship. That is which attributes need to be used to check for relationship to Profit. The model uses all these attributes to come up with a representation for calculating profit. We can however shorten this equation by just using a '.'. This has the same meaning as the statement above.

```
regressor = lm(formula = Profit ~ .,
```

```
                    data = training_set)
```

Now we can check the Summary of regressor to get an idea as to what is happening and how model is representing the informatio n.

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)      4.965e+04  7.637e+03   6.501 1.94e-07 ***
R.D.Spend        7.986e-01  5.604e-02  14.251 6.70e-16 ***
Administration  -2.942e-02  5.828e-02  -0.505    0.617    
Marketing.Spend  3.268e-02  2.127e-02   1.537    0.134    
State2           1.213e+02  3.751e+03   0.032    0.974    
State3           2.376e+02  4.127e+03   0.058    0.954    
```

R Automatically creates Dummy Variables and avoids Dummy Variable trap. This is what State2 and State3 represents.
Lower P-value => More impact that particular independent variable will have on the dependent variable.
We can just use R.D.Spend for predicting the Profit according to this. It gives about the same results as using all the varia bles.

```
Test Set Profit
182901.99 166187.94 155752.60 146121.95 129917.04 122776.86 118474.03 108733.99  99937.59  97483.56
Test Set Predicted Profit - Using all Variables
173981.09 172655.64 160250.02 135513.90 146059.36 114151.03 117081.62 110671.31  98975.29  96867.03
Test Set Predicted Profit - Using just R.D.Spend
172647.9  170708.2  160595.5  136288.1  147087.1  123020.5  114315.0  106846.5  102104.1  101369.2
```