Chapter 2 : Section 2

**Multiple Linear Regression**

Introduction to Multiple Linear Regressions

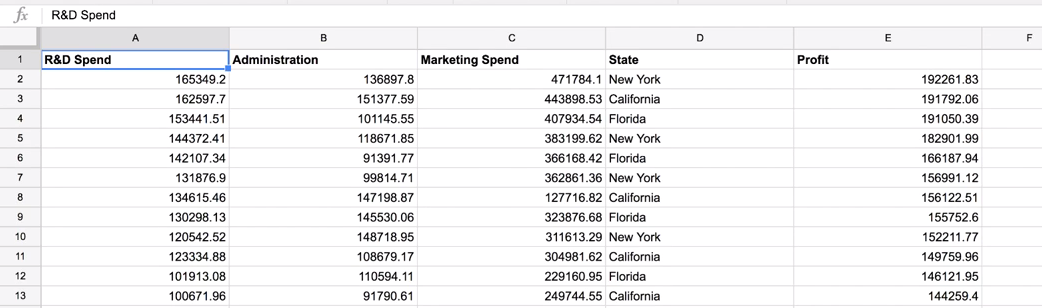
**2.1 Multiple Linear Regression**

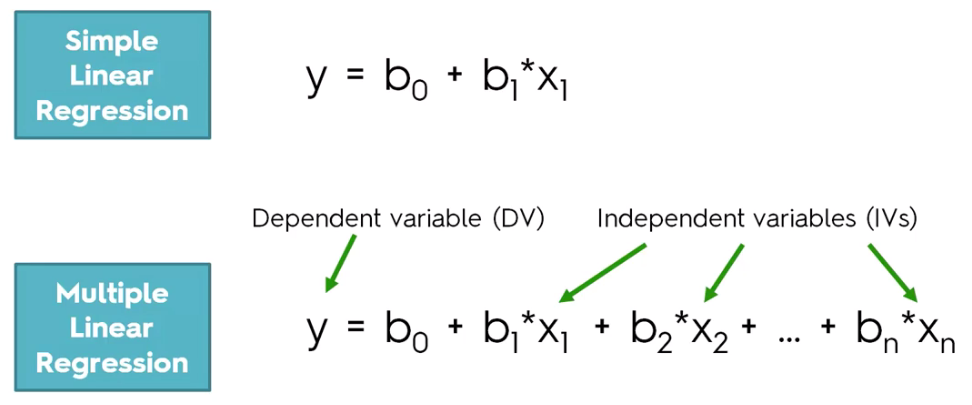
There many cases in which the *response/dependent variable* is affected by more than one *predictor/independent variable*; for such cases, the Multiple Linear Regression algorithm is used.

* Multiple Linear Regression is an extension of Simple Linear regression as it takes *more* than one *predictor variable* to predict the *response variable*.
* Multiple Linear Regression is one of the important regression algorithms which models the *linear relationship* between a *single dependent continuous variable* and *more than one independent variable*.
* Example: Prediction of CO2 emission based on engine size and number of cylinders in a car.
* Some key points about MLR:
* For ***MLR***, the *dependent or target variable* Y must be the *continuous/real*, but the *predictor or independent variable* may be of *continuous* or *categorical* form.
* Each *feature variable* must model the *linear relationship* with the *dependent variable*.
* ***MLR*** tries to ***fit*** a ***regression line*** through a ***multidimensional space of data-points***.

**2.2 Equation of MLR|**

* More than two independent variables: In multi linear regression, we deal with more than one independent variables.
* In terms of a student and his Grades is *depend* *variable*. What grade does a student get? Then in the *independent* *variables* could be how much the student has studied for the exam (***study time***), how much he has slept before the exam (***sleep time***), how many lectures he has attended throughout the course (***attendance***) and the things like that.
* The equation:
* is the dependent variable, is the constant, are coefficients and are independent variables .
* Consider the following dataset.





**2.3 Multiple Linear Regression assumptions**

|  |  |
| --- | --- |
| * ASSUMPTIONS: Before building a linear regression model you need to check that these assumptions are true. * If you ever do need to build a linear regression then make sure that you don't just blindly follow the steps presented here. * First you need to check the assumptions of the linear regression and research them and make sure that they are correct when you're building your regression model. * And then you can only proceed and be sure that you're building a good linear regression model. |  |

1. Linearity: There must be a linear relationship between the *dependent* variable and the *independent* variables.

* ***Scatterplots*** can show whether there is a linear or curvilinear relationship.

1. Homoscedasticity: This assumption states that the variance of error terms is similar across the values of the independent variables.

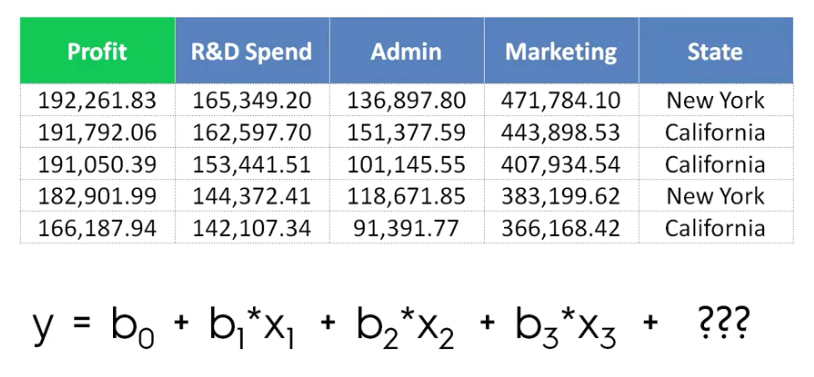
* A ***plot*** of standardized residuals versus predicted values can show whether points are equally distributed across all values of the independent variables.

1. Multivariate Normality: Multiple Linear Regression assumes that the residuals (the **differences** **between** the observed value of the dependent variable and the predicted value )are ***normally*** ***distributed***.
2. Independence of errors: Multiple Linear Regression assumes that the residuals (the **differences** **between** the observed value of the dependent variable and the predicted value ) are ***independent***.
3. Lack of multi-collinearity: Multiple Linear Regression assumes that the independent *variables* are not *highly* *correlated* with each other. This assumption is tested using Variance Inflation Factor (VIF) values.

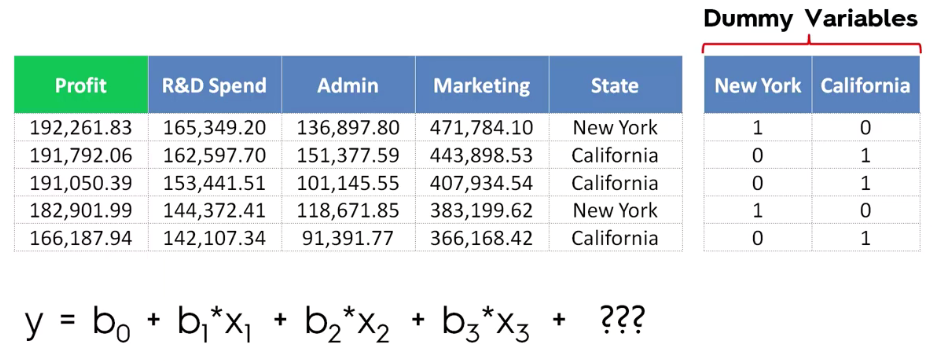
**2.4 Dummy Variables**

In following table profit is our dependent variable and the rest the blue ones are all independent variables.

* What should we place in our equation for the ***State*** column because we don't actually have numerical values here, the ***State*** is actually a ***categorical*** ***variable*** (there is categorical variables and there's numeric variables).
* First you need to go through your column and find all the different categories you have. For every single category that you find, you need to create a new column. Then in each category-column put **1** where corresponding matching category appears and **0** for unmatched.



* The approach that you need to take when you face ***categorical*** ***variables*** in ***regression*** ***models*** is you need to create dummy variables.
* So in this case we have two categories. Hence, we need to create a new column for New York and one for California. So we're kind of expanding our data set and adding some additional columns into it.
* Then we find all of rows where the state actually says New York and put a **1**.



|  |  |
| --- | --- |
|  |  |

* Do not use all the Dummy variables: This can leads us to collinearity and we eventually fall into dummy variable trap.
* All you have to do is use the ***New York*** column instead of ***States***.
* Don't use the California column (leads to dummy variable trap).
* Here all the information in our data is preserved: If we just stick to the one ***New York***, **1** means ***New York*** and **0** means ***California***. So we didn't lose any information by including only the New York. work as switches.
* Dummy variables act as switches: Actually all of the dummy variables they work as switches.
* Omit one dummy?: When you look at this approach it might seem biased. There is a coefficient for ***New York*** but for ***California*** there's no coefficient.
* In reality that's not the case because the way *regression* *models* work is that the coefficient of the dummy variable that you have not included will become the *default* *situation* for this *regression* *model*.

What that means is that the coefficient for California is going to be included in the constant by default. When is equal to zero this whole equation will turn into an equation for California.

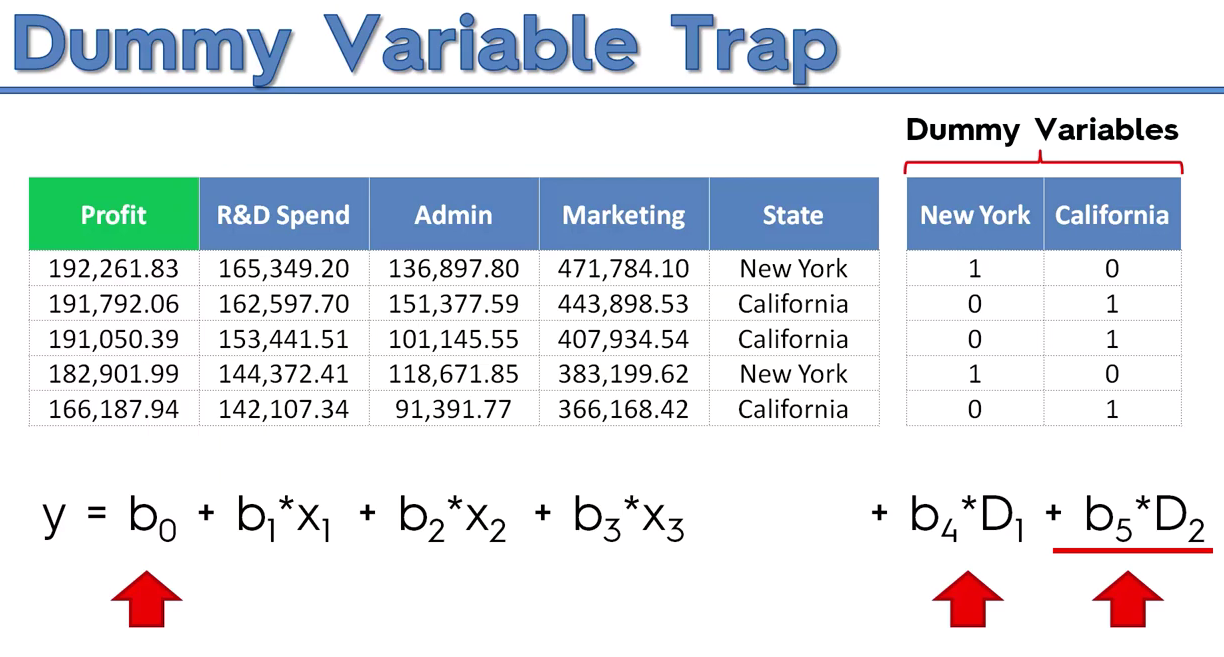
When becomes you're adding which is the difference between *New York* and *California* *coefficient*.

So basically you're altering from California to New York by flipping this light switch if it's on off. And the default state or the whole equation is working for California.

**2.5 Dummy variable trap**

**Why you should never include all of your dummy variable columns**

* Multicollinearity: Intuition here is that you're basically ***duplicating a variable***. This is because .
* The phenomenon where one or several independent variables in a linear regression predict another is called MULTICOLLINEARITY. As a result of this effect the model cannot distinguish between the effects of from the effects from of. Therefore it won't work properly. And this is the Dummy Variable Trap.
* If you do the math behind this scenario you will see that the real problem is that you cannot have these three elements in your model at the same time the constant and both the dummy variables , .

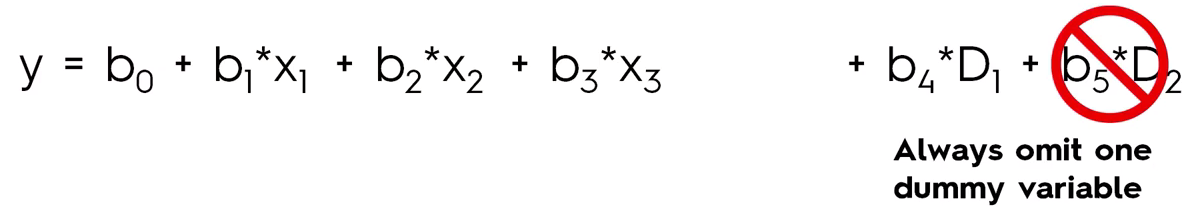


* How is the coefficient related to the dummy variable trap?

Since then if you include both and you get:

Where and

Therefore the information of the redundant dummy variable is going into the constant .



* Whenever you're building a model always omit one dummy variable and this applies to the number of dummy variables in that specific dummy set. If you have **9** then you should only include **8**, if you have **100** then you should only include **99** of them.
* Also note that if you have two sets of dummy variables then you need to apply the same rule to each set.

**2.6 p-value**

Before we get into Backward Elimination, make sure to be introduced to the ***p-value*** and have a basic understanding of how it works. By looking at almost all the explanations of the p-value on the internet

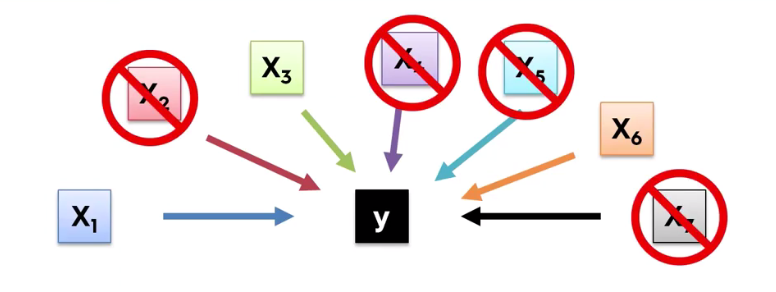
* What is the p-value?
* Null Hypothesis: To understand the P-value, we need to start by understanding the null hypothesis: the null hypothesis is the assumption that the parameters associated to your independent variables are equal to zero. Therefore under this hypothesis, your ***observations are totally random***, and don’t follow a certain pattern. For example:
* ***Does the size of a state affect population density?*** The null hypothesis is "all states have the same population density."
* ***Do cats prefer fish or milk?*** The null hypothesis is "cats have no preference; they like them the same."
* P-value: The *P-value* is the probability that the parameters associated to your independent variables have *certain nonzero values*, given that the *null hypothesis is True*. The most important thing to keep in mind about the P-Value is that it is a statistical metric: the ***lower is the P-Value, the more statistically significant is an independent variable,*** that is the better predictor it will be.
* The ***smaller*** the ***p-value***, the stronger the evidence that you should ***reject*** the ***null hypothesis***. A ***p-value*** less than ***0.05*** (typically ≤ 0.05) is statistically significant. It indicates strong evidence against the null hypothesis, as there is less than a 5% probability the *null is correct (and the results are random)*.
* The first step in backward elimination is pretty simple, you just select a *significance level*, or select the *P-value*. Usually, in most cases, a *5%* significance *level is selected*. This means the *P-value* will be *0.05*.

|  |  |
| --- | --- |
| * *P value* is a *statistical* *measure* that helps scientists determine whether or not their *hypotheses* *are correct*. *P values* are used to *determine* whether the results of their experiment are within the *normal* *range* *of values* for the events being *observed*. Usually, if the P value of a data set is below a certain pre-determined amount (like, for instance, 0.05), scientists will *reject* the *"null hypothesis"* of their experiment. * The p-value is actually the probability of getting a sample like ours, or *more* ***extreme*** *than ours IF the* ***null hypothesis*** *is true*. So, we *assume* the *null hypothesis is true* and then determine how “strange” our sample really is. If it is *not that strange* (a *large p-value*) then we *don’t change our mind about the* ***null******hypothesis***. As the **p-valu**e gets **smaller**, we start wondering if the null really is true and well maybe we should change our minds (and reject the null hypothesis). |  |

* A little more detail: A small p-value indicates that by pure luck alone, it would be unlikely to get a sample like the one we have if the null hypothesis is true. If this is small enough we start thinking that maybe we aren’t super lucky and instead our assumption about the null being true is wrong. Thats why we reject with a small p-value.
* A large p-value indicates that it would be pretty normal to get a sample like ours if the null hypothesis is true. So you can see, there is no reason here to change our minds like we did with a small p-value.
* In inferential statistics, the null hypothesis (often denoted H0)[1] is that two possibilities are the same. The null hypothesis is that the observed difference is due to chance alone. Using statistical tests, it is possible to calculate the likelihood that the null hypothesis is true.

**2.7 Feature Selection methods**

Different methods to select significant features



Among the multiple features we need to decide which ones we want to keep and which ones we want to throw out (throw out columns).

* Two common reasons:
* Garbage in garbage out: If you throw lots of stuff into your model then your model is going to be a *garbage* *model*.
* Unnecessary features increase the complexity of the model. Hence it is good to have only the most significant features and keep our model simple to get the better result.
* Don’t make it too complex to explain to Someone: At the end of the day you're going to have to explain these variables and understand the not just the math behind them but actually what it means that certain variables predict the behavior of your dependent variable and you will have to explain that to people you're presenting to.

|  |  |
| --- | --- |
| * 5 methods of building models:  1. **All-in** 2. **Backward Elimination** 3. **Forward Selection** 4. **Bidirectional Elimination** 5. **Score Comparison** |  |

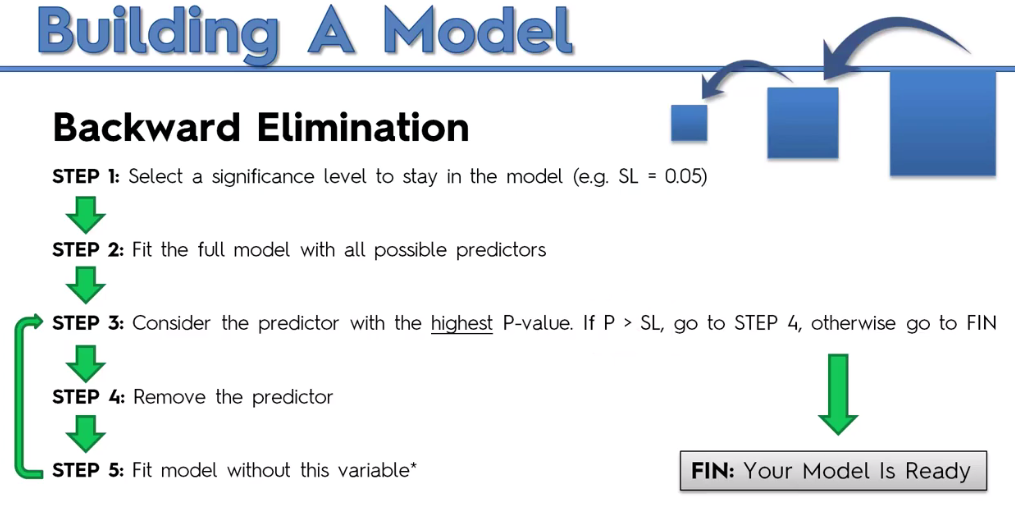
* Stepwise regression: Sometimes you'll hear stepwise regression. It's actually refers to ***2***, ***3*** and ***4*** because they are true step by step methods.
* More generally sometimes *Bidirectional* *Elimination* is called the *Stepwise* *Regression* because it is combination of *Backward* *Elimination* and *Forward* *Selection*.

1. All-in: All features are used. No specific feature is selected. When we use it:

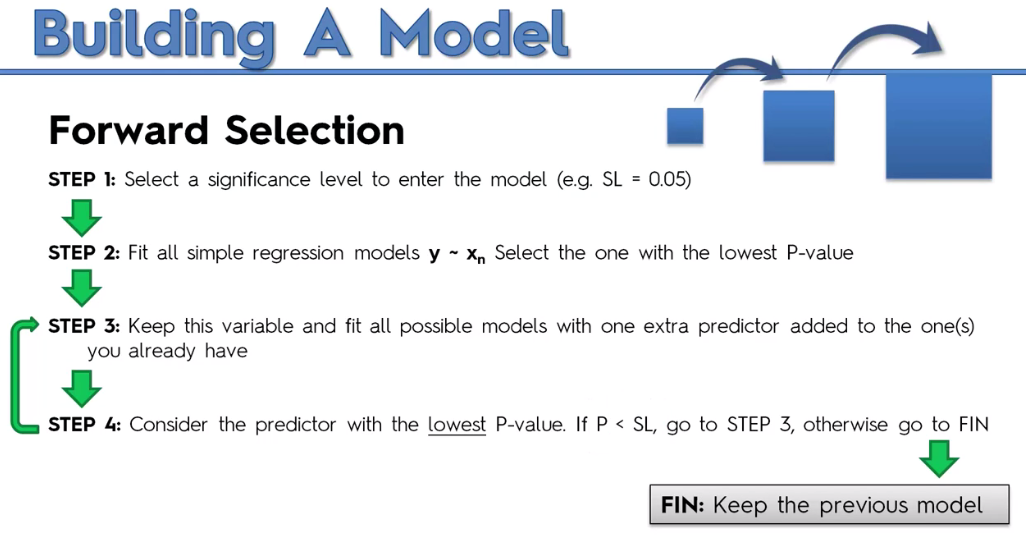
* Prior knowledge: If you have prior knowledge about those exact variables are going to be true predictors of your model (in case you build that model before)
* You have to: Somebody just gave you these variables and said please build a model. Well then you don't really have a choice. You just build the model (In case of framework or a company says that you have to use these variables.).
* Preparing for Backward Elimination: You need to use this method if you're preparing for a backward elimination.

1. Backward Elimination: Start with All Independent Variables and eliminate the rest variables one by one by checking the p-values. We keep doing the procedure until we come to a point where the highest *P values* of all the variable are still less than your *significance level*.

* STEP 1: Select a Significance Level **SL** to *stay* in the model (e.g. **SL = 0.05**). We check ***P-value*** against this ***SL***.
* STEP 2: **Fit** **the full model** with ***all possible predictors/Independent variable***
* STEP 3: Consider the predictor/Independent variable with the highest ***P-value***. If ***P > SL***, go to STEP 4, otherwise go to FIN
* STEP 4: **Remove** the predictor
* STEP 5: **Fit model again** without this variable. i.e. Rebuild and fit the model with the *remaining* *variables*.
* FIN: Your Model Is **Ready**



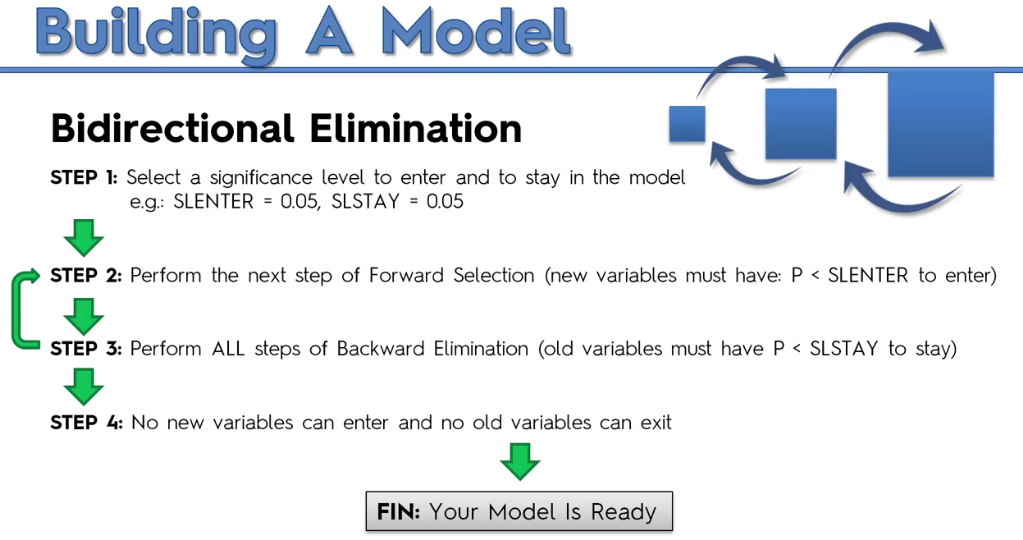
1. Forward Selection: Start with One Independent Variable and add the rest variables one by one by checking the p-values. Note that we do not keep the current model where P>SL, instead we keep the previous model.



* STEP 1: Select a Significance Level **SL** to *enter* the model (e.g. **SL = 0.05**). We check ***P-value*** against this ***SL***.
* STEP 2: Fit all Simple Linear Regression models . I.e for ***n*** predictors there will be ***n*** Simple Linear Regression models. Select the one with the lowest P-value.
* STEP 3: Keep this variable and fit all possible models again with one extra predictor (increase one independent/predictor variable added to the one(s) you already have).
* STEP 4: Consider the predictor with the lowest P-value. If , go to STEP 3, otherwise go to FIN
* FIN: Keep the previous model
* Actually what are we doing here is:
* We create Multiple Simple Regression model with *every single independent variable*. Test their P-value against SL and increase predictors one by one. And select out of all those models that has the lowest p value for the independent variable
* We keep this selected variable and we fit all other possible models with one extra predictor. That means we've selected a Simple Linear Regression with one variable. Then construct all possible Linear Regressions with two variables from other predictors.
* Now we have all possible 2 variable linear regressions. Out of all of these possible Two Variable Regressions we consider the one where the *new variable* that had the lowest p-value with  **(**means that variables a good one and it's a significant variable**)**.
* Then we moved back to Step 3.
* Means that now we have a Regression With Two Variables and now we will add a third variable. We'll try all possible variables that we have left as our third variable and then out of all of those models with three variables we will go to Step 4 and we'll select again the one of the lowest p value for that third variable. And so on.
* So basically we'll be keep growing the regression model out of the all of the possible combinations every single time and ***growing at one variable at a time***.
* We will only stop when the variable that we've added that has a p-value that is greater than our *significance level* SL. That is is false. Means that variable we just added is no longer significant.
* Note

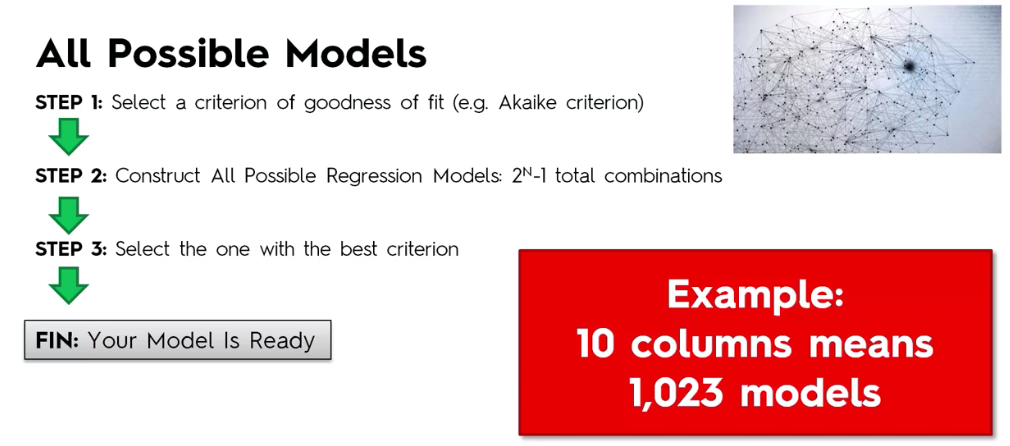
The trick is you not keep the current model but the previous one: We reject the current model with non-significant variable and pick the previous model (where all variables are significant).

1. Bidirectional Elimination:



* STEP 1: Select a significance level ***SL*** to Enter and to Stay in the model e.g.: ,
* STEP 2: Perform the next step of Forward Selection (new variables must have: to Enter)
* STEP 3: Perform ALL steps of Backward Elimination (old variables must have to Stay)
* STEP 4: No new variables can enter and no old variables can exit
* FIN: Your Model Is Ready

1. Score Comparison: Most resource consuming process.



**2.8 Implement the MLR**

Implementation of Multiple Linear Regression (MLR) model using Python. To implement MLR using Python, we have below problem:

* Problem Description:

We have a dataset of 50 start-up companies. This dataset contains five main information: *R&D* Spend (Research and development), *Administration* Spend, *Marketing* Spend, *State*, and *Profit* for a financial year.

* Goal: To create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

Since we need to find the Profit, so it is the dependent/target variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

|  |  |  |
| --- | --- | --- |
| 1. *Data Pre-processing* Steps | 1. *Fitting the MLR* model to the training set | 1. *Predicting* the result of the test set |

|  |  |
| --- | --- |
| * We analyze these 50 companies over this data set and create a model that will tell us which types of companies we should invest and our main criteria is the profit. * What we are looking for is: we want to understand for instance where companies perform better in *New York* or *California* all other things held equal or which companies perform better if you hold this "State" column equal. * We want to know: * Will a company that *spends more* on *marketing* perform *better* or a company spends less on marketing. * Also we want to understand how a company spend more on *R&D* spend or to spend more on *marketing*. | D:\100_days_comer_soln\multiple-linear-regression-in-machine-learning.png |

* It will help us to set up a set of guidelines for our own venture capitalist fund . For example, we are more interested in companies that work in *New York* and that have a very *low* *administration* spend and a very *high* *R&D* spend which is much higher than *Administration* or *Marketing* spend.
* So basically we are creating a model based off of this sample that will allow us to assess where and in which into which companies we want to invest to achieve their goal of maximizing profit.

Note:

***iloc[:, 3]*** means only 4th column.

***iloc[:, 3:]*** means all columns from 4th column

1. Data preparation:

#*Funamental Libraries*

**import** matplotlib **as** plt

**import** pandas **as** pd

**import** numpy **as** np

#*import dataset*

dataSet = pd**.read\_csv**("50\_Startups.csv")

X = dataSet**.**iloc[:, :-1] #*all rows except last*

y = dataSet**.**iloc[:, 4] #*5th row*

#*categorical to numerical*

**from** sklearn**.**compose **import** ColumnTransformer

**from** sklearn**.**preprocessing **import** OneHotEncoder

colTfrm = **ColumnTransformer**(transformers = [("encoder", **OneHotEncoder**(), [3])], remainder="passthrough" )

X\_encoded = np**.array**(colTfrm**.fit\_transform**(X))

        #*last column is now replaced with dummy colums (1st 3 colmns)*

1. Dummy Variable Trap: However, of course the Python library for ***linear regression*** is taking care of the Dummy Variable Trap, so we wouldn't need to do it manually like we do it here.

* This line just remind us about the dummy variable trap because for some software/libraries you need to do it manually.

#*Avoiding dummy-var trap: omit one dummy varable*

X\_go = X\_encoded[:, 1:]         #*select all columns starting from 2nd column*

y\_go = np**.array**(y) #*converting Dataframe to Vector/Array*

1. Test size: 40 train and 10m test.

#*split dataset to Train and Test*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_go, y\_go, test\_size = 0.2, random\_state = 0)

1. Create the model: Fitting multiple linear regression on Training set

#*Fitting multiple linear regression on traing set*

**from** sklearn**.**linear\_model **import** LinearRegression

regResor = **LinearRegression**()

regResor**.fit**(X\_train, y\_train)

1. Predicting: Cannot plot the graph for this model because it is multidimensional.

#*predict on the test-set X\_test*

y\_pred = regResor**.predict**(X\_test)

Note: We can convert a dataframe to Array obj.

|  |  |
| --- | --- |
|  |  |

Practiced version

#*Funamental Libraries*

**import** matplotlib **as** plt

**import** pandas **as** pd

**import** numpy **as** np

#*import dataset*

dataSet = pd**.read\_csv**("50\_Startups.csv")

X = dataSet**.**iloc[:, :-1] #*all rows except last*

y = dataSet**.**iloc[:, 4] #*5th row*

#*categorical to numerical*

**from** sklearn**.**compose **import** ColumnTransformer

**from** sklearn**.**preprocessing **import** OneHotEncoder

colTfrm = **ColumnTransformer**(transformers = [("encoder", **OneHotEncoder**(), [3])], remainder="passthrough" )

X\_encoded = np**.array**(colTfrm**.fit\_transform**(X))

        #*last column is now replaced with dummy colums (1st 3 colmns)*

#*Avoiding dummy-var trap: omit one dummy varable*

X\_go = X\_encoded[:, 1:]         #*select all columns starting from 2nd column*

y\_go = np**.array**(y) #*converting Dataframe to Vector/Array*

#*split dataset to Train and Test*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_go, y\_go, test\_size = 0.2, random\_state = 0)

#*Fitting multiple linear regression on traing set*

**from** sklearn**.**linear\_model **import** LinearRegression

regResor = **LinearRegression**()

regResor**.fit**(X\_train, y\_train)

#*predict on the test-set X\_test*

y\_pred = regResor**.predict**(X\_test)

#*python prctc\_mul\_lin\_rgsn.py*

Solution

#*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**('50\_Startups.csv')

X = dataset**.**iloc[:, :-1]**.**values

y = dataset**.**iloc[:, -1]**.**values *# last column*

**print**(X)

#*Encoding categorical data*

**from** sklearn**.**compose **import** ColumnTransformer

**from** sklearn**.**preprocessing **import** OneHotEncoder

ct = **ColumnTransformer**(transformers=[('encoder', **OneHotEncoder**(), [3])], remainder='passthrough')

X = np**.array**(ct**.fit\_transform**(X))

**print**(X)

#*Splitting the dataset into the Training set and Test set*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size = 0.2, random\_state = 0)

#*Training the Multiple Linear Regression model on the 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)

np**.set\_printoptions**(precision=2)

**print**(np**.concatenate**((y\_pred**.reshape**(len(y\_pred),1), y\_test**.reshape**(len(y\_test),1)),1))

* To print in the cmd for comparison

**print**(np**.concatenate**((y\_pred**.reshape**(len(y\_pred),1), y\_test**.reshape**(len(y\_test),1)),1))

**2.9 Backward Elimination**

When we built this model we actually used all the **independent** **variables**. But among these **independent** **variables** there are some that are *highly* *statistically* *significant*. That means that if we removed this *non* *statistically* *significant* *variables* from the model we would still get some amazing predictions.

* Need for Backward Elimination: An optimal Multiple Linear Regression model: In the previous section, we discussed and successfully created our Multiple Linear Regression model, where we took 4 independent variables (R&D spend, Administration spend, Marketing spend, and state (dummy variables)) and one dependent variable (Profit).
* But that model is not optimal, as we have included all the independent variables and do not know which independent variable is most affecting and which one is the least affecting for the prediction.
* Unnecessary features increase the complexity of the model. Hence it is good to have only the most significant features and keep our model simple to get the better result.
* So, in order to optimize the performance of the model, we will use the Backward Elimination method. This process is used to optimize the performance of the MLR model as it will only include the most affecting feature and **remove** the least affecting feature. Let's start to apply it to our MLR model.

#*Checking the score*

**print**(np**.concatenate**((y\_pred**.reshape**(len(y\_pred),1), y\_test**.reshape**(len(y\_test),1)),1))

**print**('\n\n------------ Train Score: ', regResor**.score**(X\_train, y\_train))

**print**('\n\n------------ Test Score: ', regResor**.score**(X\_test, y\_test))

#*building the optimal model using Backward elimination*

**import** statsmodels**.**formula**.**api **as** smf

#*X\_opt = np.append(arr = X\_go, values = np.ones(shape = (50, 1)).astype(int), axis =1)*

#*50 for row and 1 for column (row, column)*

#*convert to int type*

#*set axis = 1: column 0:row*

X\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1) #*interchange the columns*

* Step 0: Install statsmodels library

**pip install statsmodels**

Check the score:

[[103015.20159796 103282.38 ]

[132582.27760815 144259.4 ]

[132447.73845175 146121.95 ]

[ 71976.09851258 77798.83 ]

[178537.48221056 191050.39 ]

[116161.24230166 105008.31 ]

[ 67851.69209676 81229.06 ]

[ 98791.73374687 97483.56 ]

[113969.43533013 110352.25 ]

[167921.06569551 166187.94 ]]

------------ Train Score: 0.9501847627493607

------------ Test Score: 0.9347068473282436

* Note: The difference between both scores is 0.0154. On the basis of this score, we will estimate the effect of features on our model after using the Backward elimination process.
* Step: 1 Preparation of Backward Elimination:

1. Importing the library: Firstly, we need to import the statsmodels.formula.api library, which is used for the estimation of various statistical models such as OLS(Ordinary Least Square). Below is the code for it:

**import** statsmodels**.**formula**.**api **as** smf

1. Adding a column in matrix of features: As we can check in our MLR equation (a), there is one constant term , but this term is not present in our matrix of features, so we need to add it manually. We will add a column having values associated with the constant term .

X\_pre\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1)

* Library that we use here to build a linear regression models is LinearRegression, its definitely included the fact (autometically) that ***there is a constant zero*** in MLR equation but that's actually not the case for this ***statsmodels*** library which we will use to compute the values and evaluate the Statistical Significance of our independent variables.
* So that's why we need to add a ***constant column*** (Colum of ***1's***). It will correspond to our constant . That's how our ***statsmodels*** library will understand the correct MLR equn.
* To add this, we will use **append()** function of **Numpy** **library** (np which we have already imported into our code), and will assign a value of 1. Below is the code for it.

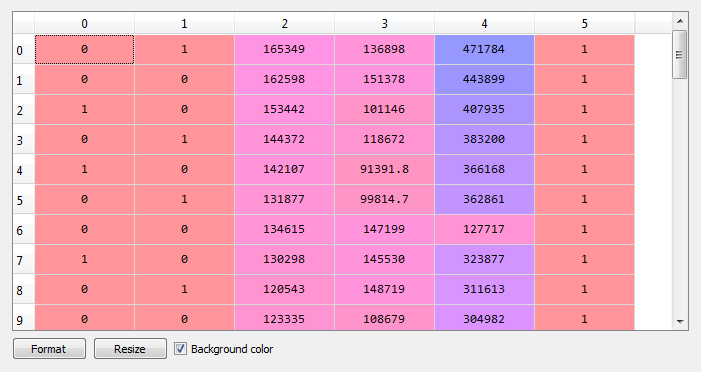
X\_pre\_opt = np**.append**(arr = X\_go, values = np**.ones**(shape = (50, 1))**.astype**(int), axis =1)

* ***"values = ":*** So the array that we're going to import here will be a matrix of 50 1's. So it will be an array of 50 1's [1, 1, 1, . . . , 1]. There is actually a trick for that. And it is

np**.ones**(shape = (50, 1))**.astype**(int)

* ***shape():*** is the shape of the matrix of what we want to create as (row, column). ***astype(int)*** is needed otherwise we will get a data type error.
* ***axis =1:*** The last argument here which is axis because you can use the ***append()*** function here to add either a line of these values or column of these values to the matrix. We need to specify if we want to add a column or line.
* If we want to add a line/row then its **axis =0**.
* If you want to column then its **axis =1** .

Notice the last column is our Constant column.



* Now we appended a column to our *feature* *matrix*. But the constant column is added to the last. We want constant column at the first column.
* So we need to interchange between ***arr*** and ***values*** attributes of **append()**: Then finally,

#*interchange the columns*

X\_pre\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1)

#*building the optimal model using Backward elimination*

**import** statsmodels**.**formula**.**api **as** smf

#*X\_opt = np.append(arr = X\_go, values = np.ones(shape = (50, 1)).astype(int), axis =1)*

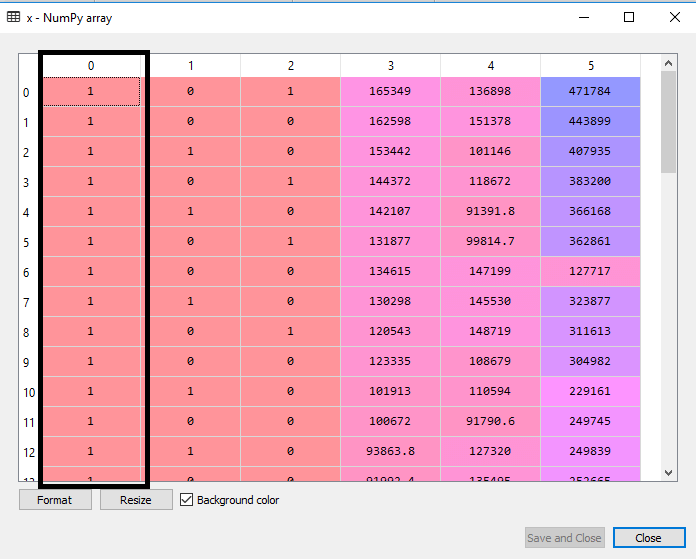
#*50 for row and 1 for column (row, column)*

#*convert to int type*

#*set axis = 1: column 0:row*

X\_pre\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1) #*interchange the columns*

* Output: By executing the above line of code, a new column will be added into our matrix of features, which will have all values equal to 1. We can check it by clicking on the x dataset under the variable explorer option.



* As we can see in the above output image, the first column is added successfully, which corresponds to the constant term of the MLR equation.
* Step 2 & 3: In this new ***X\_opt*** we specify all the indexes of the columns (all 6 columns) Explicitly. And then we eliminate one by one.

#*--------- iteration 1 ------------*

#*------------ step 2 : fit with OLS -------------------*

X\_opt = X\_pre\_opt[:, [0, 1, 2, 3, 4, 5]]  #*new vector whiuch will  be optimized*

#*Lets we set SL = 0.05 explicitly. For our learning pupose*

#*Re-fit  with new regressor. Used OLS "Ordinary Least Squares"*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*fitting with OLS*

#*------------ step 3 : inspect p-values -------------------*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

1. Firstly we will create a new feature vector ***X\_opt***, which will only contain a set of independent features that are significantly affecting the dependent variable.
2. Next, as per the **Backward Elimination process**, we need to choose a **significant level(0.5)**, and then need to fit the model with all possible predictors. So for fitting the model, we will create a ***regressor\_OLS*** object of new class OLS of statsmodels library. Then we will fit it by using the ***fit()*** method.
3. Next we need ***p-value*** to compare with ***SL*** value, so for this we will use ***summary()*** method to get the summary table of all the values.

#*================= building the optimal model using Backward elimination ====================*

#*----------- step 1 : Preprosecc for OLS ------------*

#*import statsmodels.formula.api as smf --------------------- LEGACY CODE*

**import** statsmodels**.**api **as** smf

X\_pre\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1)

#*------------ step 2 : fit with OLS -------------------*

X\_opt = X\_pre\_opt[:, [0, 1, 2, 3, 4, 5]]  #*new vector whiuch will  be optimized*

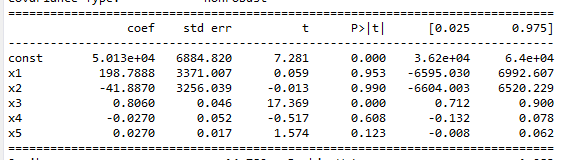
#*Lets we set SL = 0.05 explicitly. For our learning pupose*

#*Re-fit  with new regressor. Used OLS "Ordinary Least Squares"*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*fitting with OLS*

#*------------ step 3 : inspect p-values -------------------*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*



The successive iterations are given below:

#*================= building the optimal model using Backward elimination ====================*

#*----------- step 1 : Preprosecc for OLS ------------*

#*import statsmodels.formula.api as smf --------------------- LEGACY CODE*

**import** statsmodels**.**api **as** smf

#*X\_opt = np.append(arr = X\_go, values = np.ones(shape = (50, 1)).astype(int), axis =1)*

#*50 for row and 1 for column (row, column)*

#*convert to int type*

#*set axis = 1: column 0:row*

X\_pre\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1) #*interchange the columns*

#*--------- iteration 1 ------------*

#*------------ step 2 : fit with OLS -------------------*

X\_opt = X\_pre\_opt[:, [0, 1, 2, 3, 4, 5]]  #*new vector whiuch will  be optimized*

#*Lets we set SL = 0.05 explicitly. For our learning pupose*

#*Re-fit  with new regressor. Used OLS "Ordinary Least Squares"*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*fitting with OLS*

#*------------ step 3 : inspect p-values -------------------*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 2 ------------*

X\_opt = X\_pre\_opt[:, [0, 1, 3, 4, 5]]  #*removed 3rd column (x2 of iteration 1's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 3 ------------*

X\_opt = X\_pre\_opt[:, [0, 3, 4, 5]]  #*removed 2nd column (x1 of iteration 2's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 4 (can be final iteartion) ------------*

X\_opt = X\_pre\_opt[:, [0, 3, 5]]  #*removed 3rd column (x2 of iteration 3's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 5 (final iteartion) ------------*

X\_opt = X\_pre\_opt[:, [0, 3]]  #*removed 3rd column (x2 of iteration 4's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*python prctc\_mul\_lin\_rgsn.py*

|  |  |
| --- | --- |
| Iteration 4 | Iteration 5 |
|  |  |

* Final conclusion: After **4th** and **5th** iteration, we inspected the p-values. At **4th** iteration, we end-up with R&D spend is definitely a very powerful predictor of the profit and definitely has a high statistical effect impact on the dependent variable profit.
* And as the *second independent variable* is with enough *statistical* *effect* *impact* is Marketing Spend, we can see that the p-value is **0.06** that is **6%**.
* But Marketing Spend actually slightly above the ***5% significance level*** that we set. If we set another SL of 10% for example, we would have kept this independent viable.
* If you want to follow strictly the framework like the Backward Elimination Algorithm we need to remove this independent variable.
* We will use later some other metrics to make a better decision about that, when we will tack about Improving The Models Performance. In future we'll use R-squared and Adj. R-squared to calculate this significance.

Note:

We cannot have a ***0 p-value*** but it's just so small like 0.000001 type number.

All code at Once

#*Funamental Libraries*

**import** matplotlib **as** plt

**import** pandas **as** pd

**import** numpy **as** np

#*import dataset*

dataSet = pd**.read\_csv**("50\_Startups.csv")

X = dataSet**.**iloc[:, :-1] #*all rows except last*

y = dataSet**.**iloc[:, 4] #*5th row*

#*categorical to numerical*

**from** sklearn**.**compose **import** ColumnTransformer

**from** sklearn**.**preprocessing **import** OneHotEncoder

colTfrm = **ColumnTransformer**(transformers = [("encoder", **OneHotEncoder**(), [3])], remainder="passthrough" )

X\_encoded = np**.array**(colTfrm**.fit\_transform**(X))

        #*last column is now replaced with dummy colums (1st 3 colmns)*

#*Avoiding dummy-var trap: omit one dummy varable*

X\_go = X\_encoded[:, 1:]         #*select all columns starting from 2nd column*

y\_go = np**.array**(y) #*converting Dataframe to Vector/Array*

#*split dataset to Train and Test*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_go, y\_go, test\_size = 0.2, random\_state = 0)

#*Fitting multiple linear regression on traing set*

**from** sklearn**.**linear\_model **import** LinearRegression

regResor = **LinearRegression**()

regResor**.fit**(X\_train, y\_train)

#*predict on the test-set X\_test*

y\_pred = regResor**.predict**(X\_test)

#*Checking the score*

**print**(np**.concatenate**((y\_pred**.reshape**(len(y\_pred),1), y\_test**.reshape**(len(y\_test),1)),1))

**print**('\n\n------------ Train Score: ', regResor**.score**(X\_train, y\_train))

**print**('\n\n------------ Test Score: ', regResor**.score**(X\_test, y\_test))

#*================= building the optimal model using Backward elimination ====================*

#*----------- step 1 : Preprosecc for OLS ------------*

#*import statsmodels.formula.api as smf --------------------- LEGACY CODE*

**import** statsmodels**.**api **as** smf

#*X\_opt = np.append(arr = X\_go, values = np.ones(shape = (50, 1)).astype(int), axis =1)*

#*50 for row and 1 for column (row, column)*

#*convert to int type*

#*set axis = 1: column 0:row*

X\_pre\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1) #*interchange the columns*

#*--------- iteration 1 ------------*

#*------------ step 2 : fit with OLS -------------------*

X\_opt = X\_pre\_opt[:, [0, 1, 2, 3, 4, 5]]  #*new vector whiuch will  be optimized*

#*Lets we set SL = 0.05 explicitly. For our learning pupose*

#*Re-fit  with new regressor. Used OLS "Ordinary Least Squares"*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*fitting with OLS*

#*------------ step 3 : inspect p-values -------------------*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 2 ------------*

X\_opt = X\_pre\_opt[:, [0, 1, 3, 4, 5]]  #*removed 3rd column (x2 of iteration 1's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 3 ------------*

X\_opt = X\_pre\_opt[:, [0, 3, 4, 5]]  #*removed 2nd column (x1 of iteration 2's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 4 (can be final iteartion) ------------*

X\_opt = X\_pre\_opt[:, [0, 3, 5]]  #*removed 3rd column (x2 of iteration 3's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*--------- iteration 5 (final iteartion) ------------*

X\_opt = X\_pre\_opt[:, [0, 3]]  #*removed 3rd column (x2 of iteration 4's X\_opt)*

regressor\_OLS = smf**.OLS**(endog = y\_go, exog=X\_opt)**.fit**() #*Re-fit with OLS*

**print**(regressor\_OLS**.summary**()) #*to inspect the p-valuse*

#*python prctc\_mul\_lin\_rgsn.py*

***Implement Automatic Backward Elimination***

#*Fundamental Libraries*

**import** matplotlib **as** plt

**import** pandas **as** pd

**import** numpy **as** np

#*import dataset*

dataSet = pd**.read\_csv**("50\_Startups.csv")

X = dataSet**.**iloc[:, :-1] #*all rows except last*

y = dataSet**.**iloc[:, 4] #*5th row*

#*categorical to numerical*

**from** sklearn**.**compose **import** ColumnTransformer

**from** sklearn**.**preprocessing **import** OneHotEncoder

colTfrm = **ColumnTransformer**(transformers = [("encoder", **OneHotEncoder**(), [3])], remainder="passthrough" )

X\_encoded = np**.array**(colTfrm**.fit\_transform**(X))

        #*last column is now replaced with dummy colums (1st 3 colmns)*

#*Avoiding dummy-var trap: omit one dummy varable*

X\_go = X\_encoded[:, 1:]         #*select all columns starting from 2nd column*

y\_go = np**.array**(y) #*converting Dataframe to Vector/Array*

#*split dataset to Train and Test*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_go, y\_go, test\_size = 0.2, random\_state = 0)

#*Fitting multiple linear regression on traing set*

**from** sklearn**.**linear\_model **import** LinearRegression

regResor = **LinearRegression**()

regResor**.fit**(X\_train, y\_train)

#*predict on the test-set X\_test*

y\_pred = regResor**.predict**(X\_test)

#*Checking the score*

**print**(np**.concatenate**((y\_pred**.reshape**(len(y\_pred),1), y\_test**.reshape**(len(y\_test),1)),1))

**print**('\n\n------------ Train Score: ', regResor**.score**(X\_train, y\_train))

**print**('\n\n------------ Test Score: ', regResor**.score**(X\_test, y\_test))

#*================= building the optimal model using Backward elimination ====================*

"""

# ----------- step 1 : Preprosecc for OLS ------------

# import statsmodels.formula.api as smf --------------------- LEGACY CODE

import statsmodels.api as smf

# X\_opt = np.append(arr = X\_go, values = np.ones(shape = (50, 1)).astype(int), axis =1)

# 50 for row and 1 for column (row, column)

# convert to int type

# set axis = 1: column 0:row

X\_pre\_opt = np.append(arr = np.ones(shape = (50, 1)).astype(int), values =X\_go , axis =1) # interchange the columns

# --------- iteration 1 ------------

# ------------ step 2 : fit with OLS -------------------

X\_opt = X\_pre\_opt[:, [0, 1, 2, 3, 4, 5]]  # new vector whiuch will  be optimized

# Lets we set SL = 0.05 explicitly. For our learning pupose

# Re-fit  with new regressor. Used OLS "Ordinary Least Squares"

regressor\_OLS = smf.OLS(endog = y\_go, exog=X\_opt).fit() # fitting with OLS

# ------------ step 3 : inspect p-values -------------------

print(regressor\_OLS.summary()) # to inspect the p-valuse

# --------- iteration 2 ------------

X\_opt = X\_pre\_opt[:, [0, 1, 3, 4, 5]]  # removed 3rd column (x2 of iteration 1's X\_opt)

regressor\_OLS = smf.OLS(endog = y\_go, exog=X\_opt).fit() # Re-fit with OLS

print(regressor\_OLS.summary()) # to inspect the p-valuse

# --------- iteration 3 ------------

X\_opt = X\_pre\_opt[:, [0, 3, 4, 5]]  # removed 2nd column (x1 of iteration 2's X\_opt)

regressor\_OLS = smf.OLS(endog = y\_go, exog=X\_opt).fit() # Re-fit with OLS

print(regressor\_OLS.summary()) # to inspect the p-valuse

# --------- iteration 4 (can be final iteartion) ------------

X\_opt = X\_pre\_opt[:, [0, 3, 5]]  # removed 3rd column (x2 of iteration 3's X\_opt)

regressor\_OLS = smf.OLS(endog = y\_go, exog=X\_opt).fit() # Re-fit with OLS

print(regressor\_OLS.summary()) # to inspect the p-valuse

# --------- iteration 5 (final iteartion) ------------

X\_opt = X\_pre\_opt[:, [0, 3]]  # removed 3rd column (x2 of iteration 4's X\_opt)

regressor\_OLS = smf.OLS(endog = y\_go, exog=X\_opt).fit() # Re-fit with OLS

print(regressor\_OLS.summary()) # to inspect the p-valuse

"""

#*---------- implement automatic Backward Elimination: No manual iteration is needed ----------------*

**import** statsmodels**.**api **as** sm

**def** **backwardElimination**(x, sl):

    numVars = **len**(x[0])

**for** i **in** **range**(0, numVars):

        regressor\_OLS = sm**.OLS**(endog = y\_go, exog=x)**.fit**()

        #*picking the max p-value*

        maxPval = **max**(regressor\_OLS**.**pvalues)**.astype**(float)

**if** maxPval **>=** sl:

**for** j **in** **range**(0, numVars - i):

                #*deleting the clumn*

**if** (regressor\_OLS**.**pvalues[j]**.astype**(float) **==** maxPval):

                    x = np**.delete**(x, j, 1) #*deletes j-th column. "1" is used for "column". To delet "row" use "0"*

**print**(regressor\_OLS**.summary**())

**return** x

SL = 0.05

X\_pre\_opt = np**.append**(arr = np**.ones**(shape = (50, 1))**.astype**(int), values =X\_go , axis =1)

X\_opt = X\_pre\_opt[:, [0, 1, 2, 3, 4, 5]]

X\_Modeled = **backwardElimination**(X\_opt, SL)

#*python prctc\_mul\_lin\_rgsn.py*

* Here ***len(x[0])*** is the length of the first row , which is actually No. of columns. (No. of rows is ***len(x)***)

Numpy Delete

#*Python Program illustrating*

#*numpy.delete()*

**import** numpy **as** geek

#*Working on 1D*

arr = geek**.arange**(12)**.reshape**(3, 4)

**print**("arr : \n", arr)

**print**("Shape : ", arr**.**shape)

#*deletion row from 2D array*

a = geek**.delete**(arr, 1, 0)

'''

        [[ 0  1  2  3]

         [ 4  5  6  7] -> deleted

         [ 8  9 10 11]]

'''

**print**("\ndeleteing arr 2 times : \n", a)

**print**("Shape : ", a**.**shape)

#*deletion column from 2D array*

a = geek**.delete**(arr, 1, 1)

'''

        [[ 0  1\*  2  3]

         [ 4  5\*  6  7]

         [ 8  9\* 10 11]]

              ^

              Deletion

'''

**print**("\ndeleteing arr 2 times : \n", a)

**print**("Shape : ", a**.**shape)

**New model After Backward Elimination Feature selection**

#*After Backward Elimination Feature selection*

#*Building Multiple Linear Regression model by only using R&D spend*

**import** matplotlib **as** plt

**import** pandas **as** pd

**import** numpy **as** np

#*import dataset*

dataSet = pd**.read\_csv**("50\_Startups.csv")

X\_bak\_eli = dataSet**.**iloc[:, 0]**.**values #*R&D spend*

y\_bak\_eli = dataSet**.**iloc[:, 4]**.**values #*5th row*

#*converting Dataframe to Vector/Array*

X\_go = np**.array**(X\_bak\_eli)

y\_go = np**.array**(y\_bak\_eli)

#*split dataset to Train and Test*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_go, y\_go, test\_size = 0.2, random\_state = 0)

#*Fitting multiple linear regression on traing set*

**from** sklearn**.**linear\_model **import** LinearRegression

regResor = **LinearRegression**()

#*Reshape your data either using array.reshape(-1, 1) if your data has a single feature*

regResor**.fit**(X\_train**.reshape**(-1, 1), y\_train)

#*predict on the test-set X\_test*

y\_pred = regResor**.predict**(X\_test**.reshape**(-1, 1))

#*Checking the score*

**print**(np**.concatenate**((y\_pred**.reshape**(len(y\_pred),1), y\_test**.reshape**(len(y\_test),1)),1))

**print**('\n\n------------ Train Score: ', regResor**.score**(X\_train**.reshape**(-1, 1), y\_train))

**print**('\n\n------------ Test Score: ', regResor**.score**(X\_test**.reshape**(-1, 1), y\_test))

#*python prctc\_optimized\_mul\_lin\_rgsn.py*

Comparison between two models Before and after feature selection

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **[[103015.20159796** | 103282.38 ] | | **[132582.27760816** | 144259.4 ] | | **[132447.73845175** | 146121.95 ] | | **[ 71976.09851259** | 77798.83 ] | | **[178537.48221054** | 191050.39 ] | | **[116161.24230163** | 105008.31 ] | | **[ 67851.69209676** | 81229.06 ] | | **[ 98791.73374688** | 97483.56 ] | | **[113969.43533012** | 110352.25 ] | | **[167921.0656955** | 166187.94 ]] |   ------------ Train Score: 0.9501847627493607  ------------ Test Score: 0.9347068473282949 | |  |  | | --- | --- | | **[[104667.27805998** | 103282.38 ] | | **[134150.83410578** | 144259.4 ] | | **[135207.80019517** | 146121.95 ] | | **[ 72170.54428856** | 77798.83 ] | | **[179090.58602508** | 191050.39 ] | | **[109824.77386586** | 105008.31 ] | | **[ 65644.27773757** | 81229.06 ] | | **[100481.43277139** | 97483.56 ] | | **[111431.75202432** | 110352.25 ] | | **[169438.14843539** | 166187.94 ]] |   ------------ Train Score: 0.9449589778363044  ------------ Test Score: 0.9464587607787219 |
| * Difference between both scores is 0.0154 | * Difference between both scores is **.00149**. |