**7-Steps Predictive Modeling Process**

Step 1: Understand Business Objective

Step 2: Define Modeling Goals

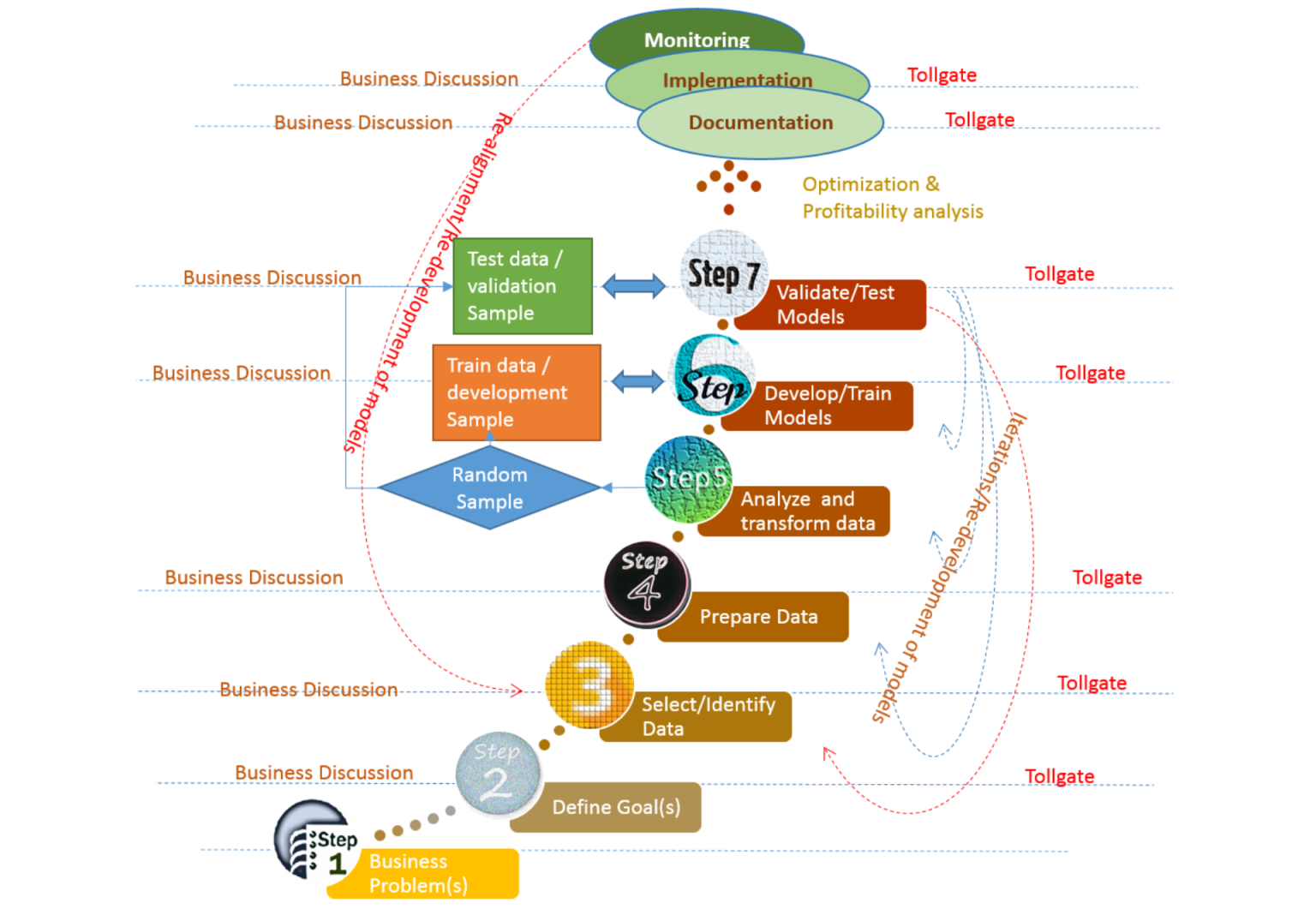
Step 3: Select/Get Data

Step 4: Prepare Data

Step 5: Analyze and Transform Variables. Random Sampling

Step 6: Model Selection and Develop Models (Training)

Step 7: Validate Models (Testing), Optimize and Profitability



**Why Standard Process?**

Organizations today are increasing their use of advance analytics and predictive modeling. The processes of generating predictive models involve data preparation, checking of data quality, reduction, modelling, prediction, and analysis of results. Generating high-quality predictive models is a time consuming activity because of the tuning process in finding optimum model parameters and often required to redevelop, tweak or reuse the models in the future. Thus it is important to

* Follow standard methodologies and industry best practices
* Establish Governance in Modeling building
* Maintain Quality standards across the board
* Ensure Re-usability of codes
* Save Time and Cost for Future Development
* Best Practices
* Comply with Internal and External Reviews
* Audit Requirements

**For Whom?**

* Advanced Analytics Professionals
* Predictive Modelers
* Data Scientists
* Business Analysts
* Change Managers
* CxOs

**Key Stake Holders**

* Business Managers, Product Managers, Risk Managers, Marketing Managers
* Operation Managers, IT Managers, Data Managers, Change Managers
* Business Analysts, Data Scientist, Consultants
* IT Developers, Information Security Risk Managers
* Independent Reviewers
* Internal and External Auditors

**Step 1: Business Objective(s)**

* Target Marketing
* Risk & Fraud Management
* Strategy Implementation and Change Management
* Operational Efficiency
* Increase Customer Experience
* Manage Marketing Campaigns
* Forecast Revenue or Loss
* Workforce Management
* Financial Modeling
* Churn Management
* Social Media Influencers

**Step 1.1: Business Objectives - Asking Right Questions!**

It is very important to clearly define the goals based on business objective.

* Do you want to understand the characteristics of customers?
* Do you want to make unprofitable customers profitable?
* Do you want to understand what driving sales?
* Do you want to win-back lost customers?
* Do you want to increase sales?
* Do you want to reduce customer churn?
* Do you want to reduce cost of production or operation?
* Do you want to target new customers?
* Do you want to identify probable credit default customers?
* Do you want to know X-sell/Upsell opportunities?

Businesses want to find answers all such important questions and make decisions based on data…

**Step 1.2: Business Objective(s) - Target Modeling Opportunities**

| **Industry** | **Response** | **Risk Mitigation** | **Attrition** | **Cross-Sell/Upsell** | **Net Present Value** | **Life Time Value** |
| --- | --- | --- | --- | --- | --- | --- |
| Retail | X |  | X | X | X | X |
| Banking | X | X | X | X | X | X |
| Insurance | X | X | X | X | X | X |
| Telecom | X | X | X | X | X | X |
| Utilities | X | X | X | X | X | X |
| Hospitality | X |  | X | X | X | X |
| Catalog | X |  |  | X | X | X |
| Publishing | X |  | X | X | X | X |

**Step 2: Define Goals - translate business objective into analytics goal**

Based on the business questions we want to answer, translate the business objective into Analytic terms

* Profile Analysis
* Segmentations
* Response Modeling
* Risk Modeling
* Activation
* Cross-Sell and Upsell
* Attrition/Churn Modeling
* Net Present Value(NPV)
* Customer Life Time Value (CLTV)
* etc.

**Step 3: Selecting Data for Modeling**

Selecting best data for target modeling requires thorough understanding of the market, business and the objective. The model is only as good and relevant as the underlying data:

**Data Types**

| **Data Type** | **Predictive Power** | **Stability** | **Cost** |
| --- | --- | --- | --- |
| Demographic | Medium | High | Low |
| Behavioural | High | Low | High |
| Psychographic | Medium | Medium | High |

**Sources of Data**

| **Internal Sources** | **External Sources** |
| --- | --- |
| Customer Data, Transaction Data | Survey Data, Research Data, Suppliers, Ratings |
| Other History | Credit Bureau Data, Third Party data, Sellers, Compilers |

**Step3.1: A Case Study - Target Marketing**

Typical data required for Target Marketing

| **Demographic Data** | **Behaviour Data** | **External Data** |
| --- | --- | --- |
| Customer Demographic | Transaction | Customer Survey |
| Income | Loyalty | Market Research |
| Purchasing Power | - | Macro Economic Factors |
| - | - | Competitions |

**Step 4: Prepare Data**

1. In this step we need prepare data into right format for analysis and the tool you may want use.
2. Do initial cleaning up
3. Define Variables and Create Data Dictionary
4. Joining/Appending multiple datasets
5. Validate for correctness
6. Produce Basic Summary Reports

**Step 5: Analyze and Transform Variables**

Once data is in right shape and perform

* univariate analysis: to check the distribution of each of the variables and features
* multivariate analyses: to check relationships with other variables and with dependent variables

Based on type of model you are going to use, you may need to transform the variables using one of the approaches

1. Bining approach: create distinct groups
2. Transformation:

* Logarithmic, Polynomial
* Square Root, Inverse, Square, boxCox

1. Extreme value (outlier) treatments
2. Missing Value Treatment
3. Dimension Reduction - Information Value(IV) and Weight of Evidence(WoE), Variable Clustering, PCA, Factor Analysis, etc.

**Step 5.1: Random Sampling (Train and Test)**

* Training Sample: Model will be developed on this sample. Typically 50%, 60%, 70% or 80% of the data goes here.
* Test Sample: Model performances will be validated on this sample. Typically 50%, 40%, 30% or 20% of the data goes here

**Step 6.1: Model Selection**

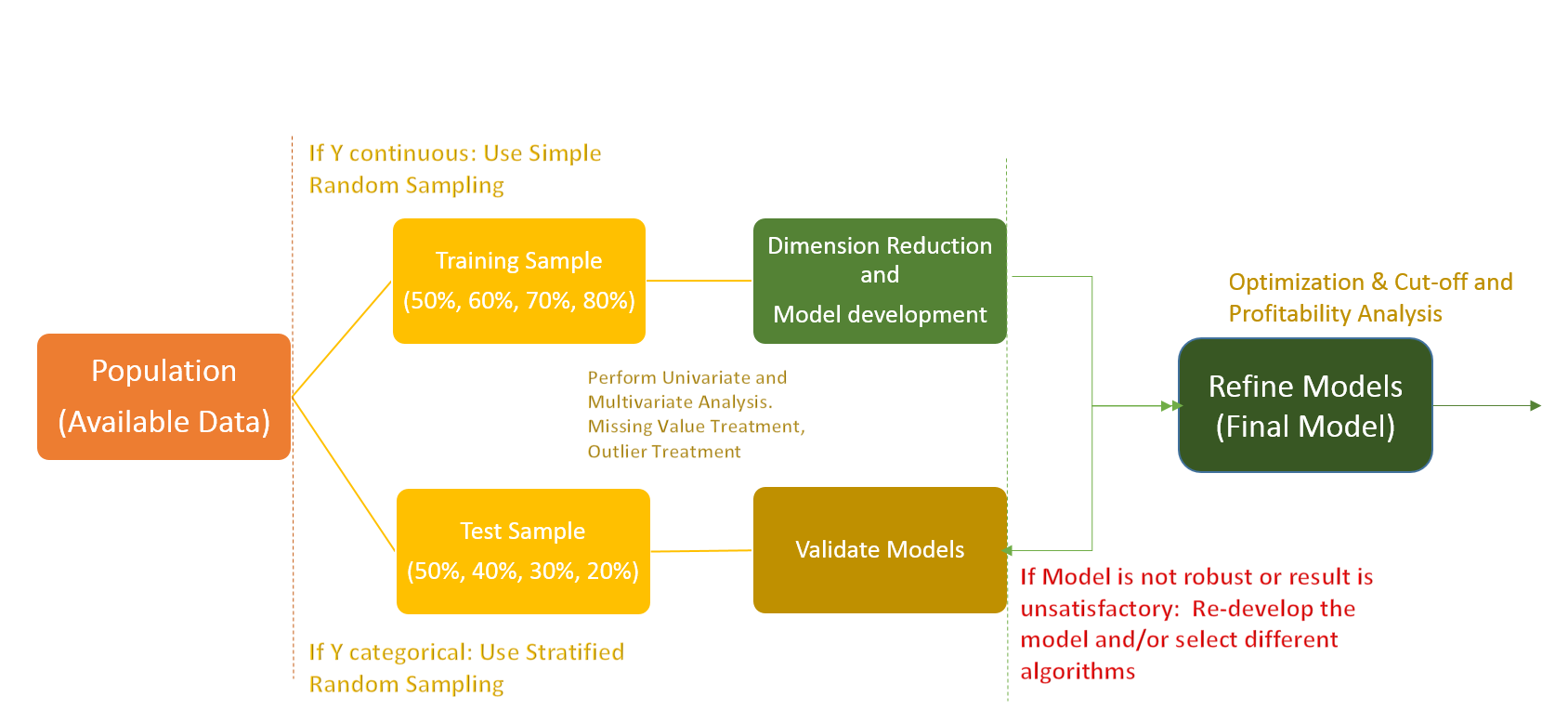
Based on the defined goal(s) (supervised or unsupervised) we have to select one of or combinations of modeling techniques. Such as

* General linear model
* Non-Linear Regression
* Linear Regression
* Lasso Regression
* Ridge Regression
* Non-Negative Garrotte Regression
* Percentage Regression
* Quantile Regression
* Non-parametric regression
* Logistic Regression
* Tobit Regression
* Probit Regression
* Classification/Decision Trees
* Random Forest
* Support Vector Machine (SVM)
* Distance metric learning
* Bayesian methods
* Graphical Models
* Neural Networks
* Genetic Algorithm
* The Hazard and Survival Functions
* Time Series Models
* Signal Processing
* Clustering Techniques
* Market Basket Analysis
* Frequent Itemset Mining
* Association Rule Mining etc.

There are wide variety of choices available outside this list.

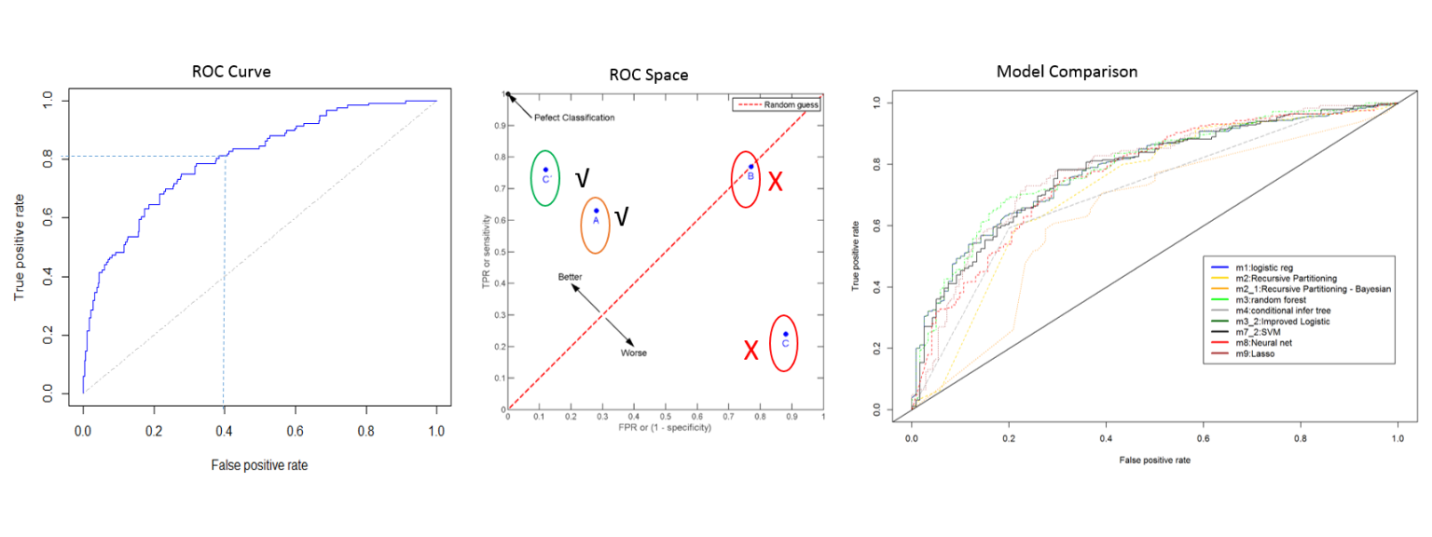
**Step 6.2: Build/Develop/Train Models**

* Validate the assumptions of the chosen algorithm
* Check for Multicollinearity and Redundancies of Independent Variables (Features). Sometime in Machine Learning, we are keen on accuracies of the models and hence we may not perform these checks!
* Develop/Train Model on Training Sample, which is 80%/70%/60%/50% of the available data(Population)
* Check Model performance - Error, Accuracy, ROC, KS, Gini



**Step 7: Validate/Test Models**

* Score and Predict using Test Sample
* Check for the robustness and stability of the model
* Check Model Performance: Accuracy, ROC, AUC, KS, GINI etc.



* Perform Cross Validation to increase accuracy/performance of the models

**Example-1**

**Step 1** : Import required libraries and read test and train data set. Append both.

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

import random

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

train=pd.read\_csv('C:/Users/Analytics Vidhya/Desktop/challenge/Train.csv')

test=pd.read\_csv('C:/Users/Analytics Vidhya/Desktop/challenge/Test.csv')

train['Type']='Train' #Create a flag for Train and Test Data set

test['Type']='Test'

fullData = pd.concat([train,test],axis=0) #Combined both Train and Test Data set

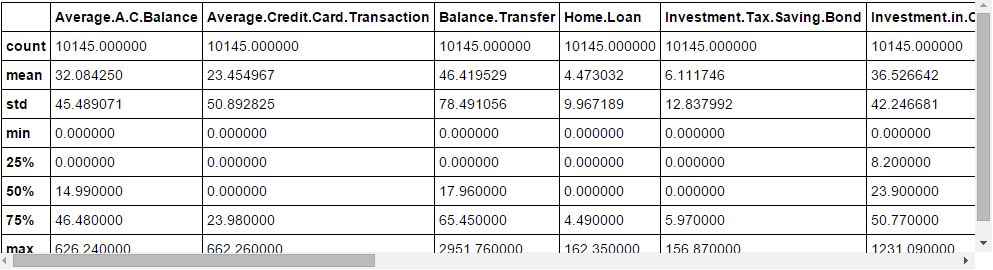
**Step 2**: Step 2 of the framework is not required in Python. On to the next step.

**Step 3**: View the column names / summary of the dataset

fullData.columns # This will show all the column names

fullData.head(10) # Show first 10 records of dataframe

fullData.describe() #You can look at summary of numerical fields by using describe() function

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/09/Capture_10.png)

**Step 4**: Identify the a) ID variables b)  Target variables c) Categorical Variables d) Numerical Variables e) Other Variables

ID\_col = ['REF\_NO']

target\_col = ["Account.Status"]

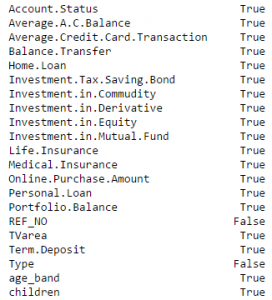
cat\_cols = ['children','age\_band','status','occupation','occupation\_partner','home\_status','family\_income','self\_employed', 'self\_employed\_partner','year\_last\_moved','TVarea','post\_code','post\_area','gender','region']

num\_cols= list(set(list(fullData.columns))-set(cat\_cols)-set(ID\_col)-set(target\_col)-set(data\_col))

other\_col=['Type'] #Test and Train Data set identifier

**Step 5** : Identify the variables with missing values and create a flag for those

fullData.isnull().any()#Will return the feature with True or False,True means have missing value else False



num\_cat\_cols = num\_cols+cat\_cols # Combined numerical and Categorical variables

#Create a new variable for each variable having missing value with VariableName\_NA

# and flag missing value with 1 and other with 0

for var in num\_cat\_cols:

if fullData[var].isnull().any()==True:

fullData[var+'\_NA']=fullData[var].isnull()\*1

**Step 6** : Impute Missing values

#Impute numerical missing values with mean

fullData[num\_cols] = fullData[num\_cols].fillna(fullData[num\_cols].mean(),inplace=True)

#Impute categorical missing values with -9999

fullData[cat\_cols] = fullData[cat\_cols].fillna(value = -9999)

**Step 7** : Create a label encoders for categorical variables and split the data set to train & test, further split the train data set to Train and Validate

#create label encoders for categorical features

for var in cat\_cols:

number = LabelEncoder()

fullData[var] = number.fit\_transform(fullData[var].astype('str'))

#Target variable is also a categorical so convert it

fullData["Account.Status"] = number.fit\_transform(fullData["Account.Status"].astype('str'))

train=fullData[fullData['Type']=='Train']

test=fullData[fullData['Type']=='Test']

train['is\_train'] = np.random.uniform(0, 1, len(train)) <= .75

Train, Validate = train[train['is\_train']==True], train[train['is\_train']==False]

**Step 8** : Pass the imputed and dummy (missing values flags) variables into the modelling process. I am using random forest to predict the class

features=list(set(list(fullData.columns))-set(ID\_col)-set(target\_col)-set(other\_col))

x\_train = Train[list(features)].values

y\_train = Train["Account.Status"].values

x\_validate = Validate[list(features)].values

y\_validate = Validate["Account.Status"].values

x\_test=test[list(features)].values

random.seed(100)

rf = RandomForestClassifier(n\_estimators=1000)

rf.fit(x\_train, y\_train)

**Step 9** : Check performance and make predictions

status = rf.predict\_proba(x\_validate)

fpr, tpr, \_ = roc\_curve(y\_validate, status[:,1])

roc\_auc = auc(fpr, tpr)

print roc\_auc

final\_status = rf.predict\_proba(x\_test)

test["Account.Status"]=final\_status[:,1]

test.to\_csv('C:/Users/Analytics Vidhya/Desktop/model\_output.csv',columns=['REF\_NO','Account.Status'])

**Example-2**

Load Dataset — *Data Understanding*

**import** **pandas** **as** **pd**  
  
df = pd.read\_excel("bank.xlsx")

Data Transformation — *Data Preparation*

Now, we have our dataset in a pandas dataframe. Next, we look at the variable descriptions and the contents of the dataset using df.info() and df.head() respectively. The target variable (‘Yes’/’No’) is converted to (1/0) using the code below.

df['target'] = df['y'].apply(**lambda** x: 1 **if** x == 'yes' **else** 0)

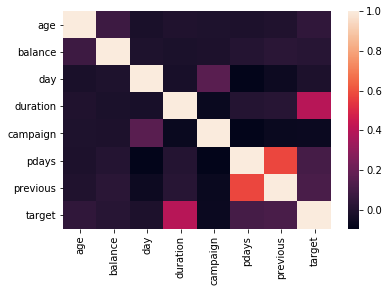
Descriptive Stats — *Data Understanding*

Exploratory statistics help a modeler understand the data better. A couple of these stats are available in this framework. First, we check the missing values in each column in the dataset by using the below code.

df.isnull().mean().sort\_values(ascending=**False**)\*100

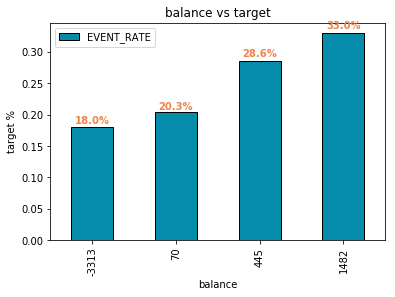
Second, we check the correlation between variables using the code below.

**import** **seaborn** **as** **sns**  
**import** **matplotlib.pyplot** **as** **plt**  
%matplotlib inline  
corr = df.corr()  
sns.heatmap(corr,   
 xticklabels=corr.columns,  
 yticklabels=corr.columns)



Finally, in the framework, I included a binning algorithm that automatically bins the input variables in the dataset and creates a bivariate plot (inputs vs target).

bar\_color = '#058caa'  
num\_color = '#ed8549'  
  
final\_iv,\_ = data\_vars(df1,df1['target'])  
final\_iv = final\_iv[(final\_iv.VAR\_NAME != 'target')]  
grouped = final\_iv.groupby(['VAR\_NAME'])  
**for** key, group **in** grouped:  
 ax = group.plot('MIN\_VALUE','EVENT\_RATE',kind='bar',color=bar\_color,linewidth=1.0,edgecolor=['black'])  
 ax.set\_title(str(key) + " vs " + str('target'))  
 ax.set\_xlabel(key)  
 ax.set\_ylabel(str('target') + " %")  
 rects = ax.patches  
 **for** rect **in** rects:  
 height = rect.get\_height()  
 ax.text(rect.get\_x()+rect.get\_width()/2., 1.01\*height, str(round(height\*100,1)) + '%',   
 ha='center', va='bottom', color=num\_color, fontweight='bold')



The values in the bottom represent the start value of the bin.

Variable Selection — *Data Preparation*

Please read my article below on variable selection process which is used in this framework. The variables are selected based on a voting system. We use different algorithms to select features and then finally each algorithm votes for their selected feature. The final vote count is used to select the best feature for modeling.

Model — *Modeling*

80% of the predictive model work is done so far. To complete the rest 20%, we split our dataset into train/test and try a variety of algorithms on the data and pick the best one.

**from** **sklearn.cross\_validation** **import** train\_test\_split  
  
train, test = train\_test\_split(df1, test\_size = 0.4)  
train = train.reset\_index(drop=**True**)  
test = test.reset\_index(drop=**True**)  
  
features\_train = train[list(vif['Features'])]  
label\_train = train['target']  
features\_test = test[list(vif['Features'])]  
label\_test = test['target']

We apply different algorithms on the train dataset and evaluate the performance on the test data to make sure the model is stable. The framework includes codes for Random Forest, Logistic Regression, Naive Bayes, Neural Network and Gradient Boosting. We can add other models based on our needs. The Random forest code is provided below.

**from** **sklearn.ensemble** **import** RandomForestClassifier  
clf = RandomForestClassifier()  
  
clf.fit(features\_train,label\_train)  
  
pred\_train = clf.predict(features\_train)  
pred\_test = clf.predict(features\_test)  
  
**from** **sklearn.metrics** **import** accuracy\_score  
accuracy\_train = accuracy\_score(pred\_train,label\_train)  
accuracy\_test = accuracy\_score(pred\_test,label\_test)  
  
**from** **sklearn** **import** metrics  
fpr, tpr, \_ = metrics.roc\_curve(np.array(label\_train), clf.predict\_proba(features\_train)[:,1])  
auc\_train = metrics.auc(fpr,tpr)  
  
fpr, tpr, \_ = metrics.roc\_curve(np.array(label\_test), clf.predict\_proba(features\_test)[:,1])  
auc\_test = metrics.auc(fpr,tpr)

Hyper parameter Tuning — *Modeling*

In addition, the hyperparameters of the models can be tuned to improve the performance as well. Here is a code to do that.

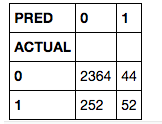
**from** **sklearn.model\_selection** **import** RandomizedSearchCV  
**from** **sklearn.ensemble** **import** RandomForestClassifier  
  
n\_estimators = [int(x) **for** x **in** np.linspace(start = 10, stop = 500, num = 10)]  
max\_features = ['auto', 'sqrt']  
max\_depth = [int(x) **for** x **in** np.linspace(3, 10, num = 1)]  
max\_depth.append(**None**)  
min\_samples\_split = [2, 5, 10]  
min\_samples\_leaf = [1, 2, 4]  
bootstrap = [**True**, **False**]  
  
random\_grid = {'n\_estimators': n\_estimators,  
 'max\_features': max\_features,  
 'max\_depth': max\_depth,  
 'min\_samples\_split': min\_samples\_split,  
 'min\_samples\_leaf': min\_samples\_leaf,  
 'bootstrap': bootstrap}  
  
rf = RandomForestClassifier()  
  
rf\_random = RandomizedSearchCV(estimator = rf, param\_distributions = random\_grid, n\_iter = 10, cv = 2, verbose=2, random\_state=42, n\_jobs = -1)  
rf\_random.fit(features\_train, label\_train)

Final Model and Model Performance — *Evaluation*

The final model that gives us the better accuracy values is picked for now. However, we are not done yet. We need to evaluate the model performance based on a variety of metrics. The framework contain codes that calculate cross-tab of actual vs predicted values, ROC Curve, Deciles, KS statistic, Lift chart, Actual vs predicted chart, Gains chart. We will go through each one of them below.

1. **Crosstab**

pd.crosstab(label\_train,pd.Series(pred\_train),rownames=['ACTUAL'],colnames=['PRED'])



Crosstab of Actual vs Predicted values

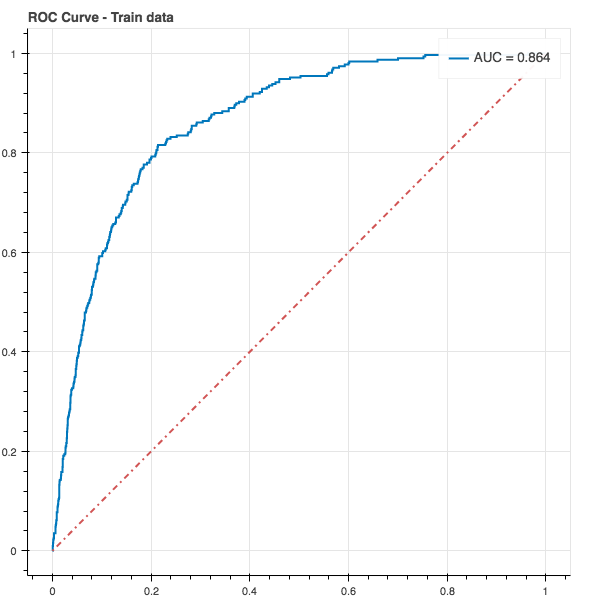
**2. ROC/AUC curve or c-statistic**

from bokeh.charts import Histogram  
from ipywidgets import interact  
from bokeh.plotting import figure  
from bokeh.io import push\_notebook, show, output\_notebook  
output\_notebook()

from sklearn import metrics  
preds = clf.predict\_proba(features\_train)[:,1]

fpr, tpr, \_ = metrics.roc\_curve(np.array(label\_train), preds)  
auc = metrics.auc(fpr,tpr)

p = figure(title="ROC Curve - Train data")  
r = p.line(fpr,tpr,color='#0077bc',legend = 'AUC = '+ str(round(auc,3)), line\_width=2)  
s = p.line([0,1],[0,1], color= '#d15555',line\_dash='dotdash',line\_width=2)  
show(p)



Save Model for future use — *Deployment*

Finally, we developed our model and evaluated all the different metrics and now we are ready to deploy model in production. The last step before deployment is to save our model which is done using the code below.

**import** **pandas**  
**from** **sklearn.externals** **import** joblib  
  
filename = 'final\_model.model'  
i = [d,clf]  
joblib.dump(i,filename)

Here, “clf” is the model classifier object and “d” is the label encoder object used to transform character to numeric variables.

Score New data — *Deployment*

For scoring, we need to load our model object (clf) and the label encoder object back to the python environment.

*# Use the code to load the model*  
filename = 'final\_model.model'  
  
**from** **sklearn.externals** **import** joblib  
d,clf=joblib.load(filename)

Then, we load our new dataset and pass to the scoring macro.

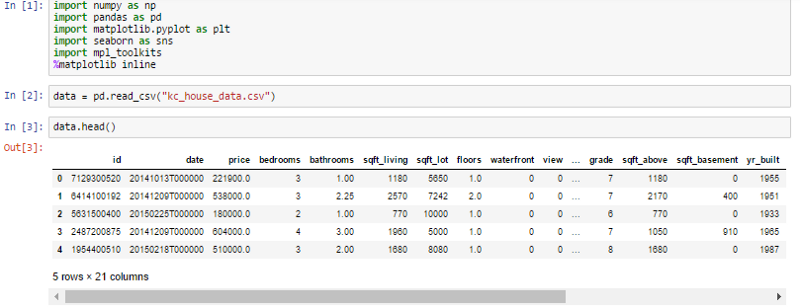
**def** score\_new(features,clf):  
 score = pd.DataFrame(clf.predict\_proba(features)[:,1], columns = ['SCORE'])  
 score['DECILE'] = pd.qcut(score['SCORE'].rank(method = 'first'),10,labels=range(10,0,-1))  
 score['DECILE'] = score['DECILE'].astype(float)  
 **return**(score)

And we call the macro using the code below

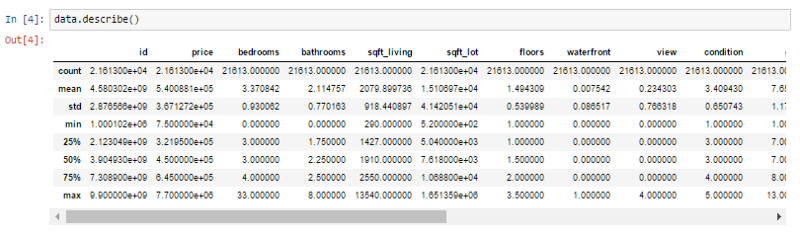
scores = score\_new(new\_score\_data,clf)

**Example-3**

First thing first , we import our libraries and dataset and then we see the head of the data to know how the data looks like and use describe function to see the percentile’s and other key statistics.



Starting , by importing libraries and reading dataset



Knowing more about the dataset

What can we infer from the above describe function ?

1. Look at the bedroom columns , the dataset has a house where the house has 33 bedrooms , seems to be a massive house and would be interesting to know more about it as we progress.
2. Maximum square feet is 13,450 where as the minimum is 290. we can see that the data is distributed.

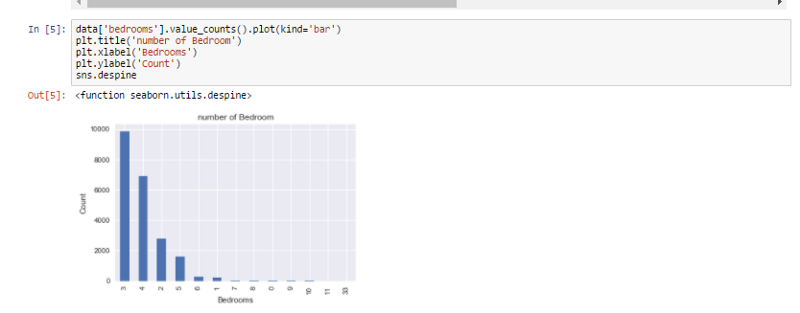
Similarly , we can infer so many things by just looking at the describe function.

Now , we are going to see some visualization and also going to see how and what can we infer from visualization.

#### ****Which is the most common house (Bedroom wise) ?****

Let’s see which is most common bedroom number. You may wonder why is it important ? Let’s look at this problem from a builder’s perspective, sometimes it’s important for a builder to see which is the highest selling house type which enables the builder to make house based on that. Here in India , for a good locality a builder opts to make houses which are more than 3 bedrooms which attracts the higher middle class and upper class section of the society.

Let’s see how this pans out for this data ?



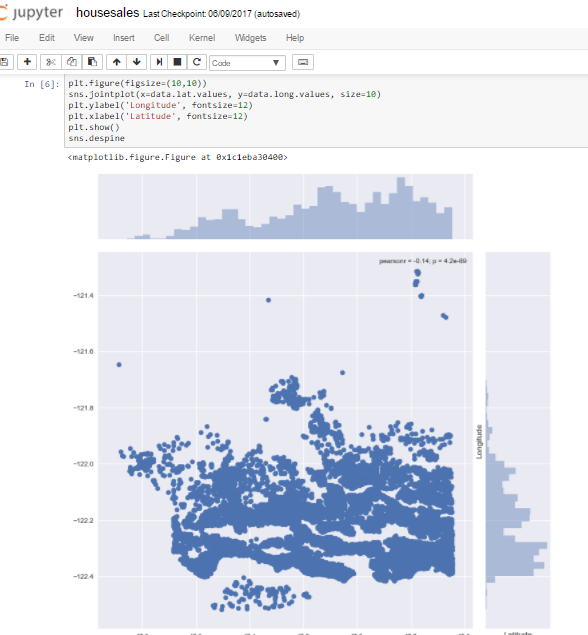
House bedrooms and count

As we can see from the visualization 3 bedroom houses are most commonly sold followed by 4 bedroom. So how is it useful ? For a builder having this data , He can make a new building with more 3 and 4 bedroom’s to attract more buyers.

So now we know that 3 and 4 bedroom’s are highest selling. But at which locality ?

#### Visualizing the location of the houses based on latitude and longitude.

So according to the dataset , we have latitude and longitude on the dataset for each house. We are going to see the common location and how the houses are placed.



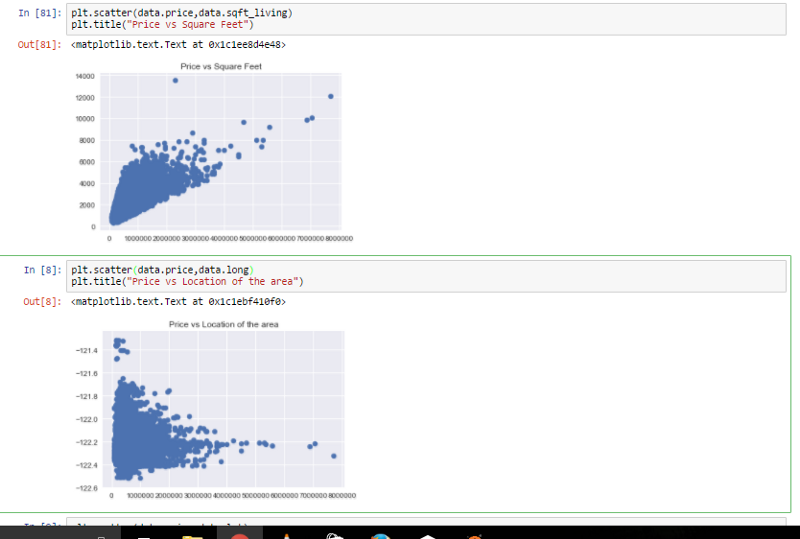
How houses are placed ?

We use seaborn , and we get his beautiful visualization. Joinplot function helps us see the concentration of data and placement of data and can be really useful. Let us see what we can infer from this visualization. For latitude between -47.7 and -48.8 there are many houses , which would mean that maybe it’s an ideal location isn’t it ? But when we talk about longitude we can see that concentration is high between -122.2 to -122.4. Which would mean that most of the buy’s has been for this particular location.

#### ****How common factors are affecting the price of the houses ?****

We saw the common locations and now we’re going to see few common factors affecting the prices of the house and if so ? then by how much ?

Let us start with , If price is getting affecting by living area of the house or not ?



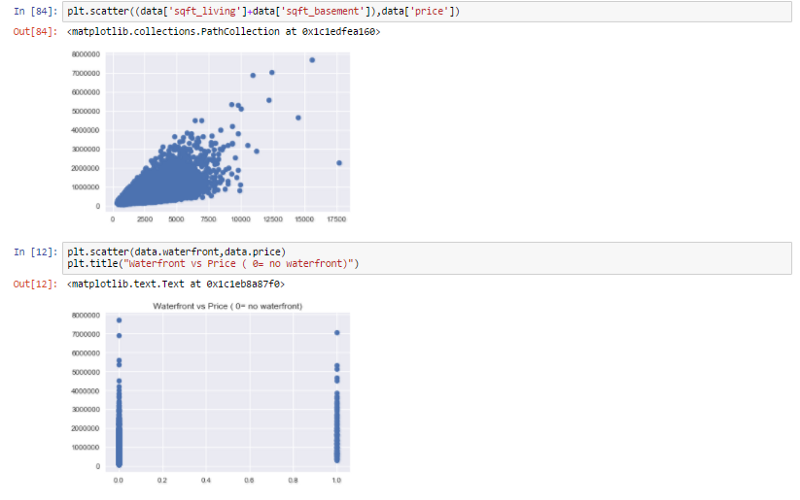
Price vs Square feet and Price vs Longitude

The plot that we used above is called scatter plot , scatter plot helps us to see how our data points are scattered and are usually used for two variables. From the first figure we can see that more the living area , more the price though data is concentrated towards a particular price zone , but from the figure we can see that the data points seem to be in linear direction. Thanks to scatter plot we can also see some irregularities that the house with the highest square feet was sold for very less , maybe there is another factor or probably the data must be wrong. The second figure tells us about the location of the houses in terms of longitude and it gives us quite an interesting observation that -122.2 to -122.4 sells houses at much higher amount.

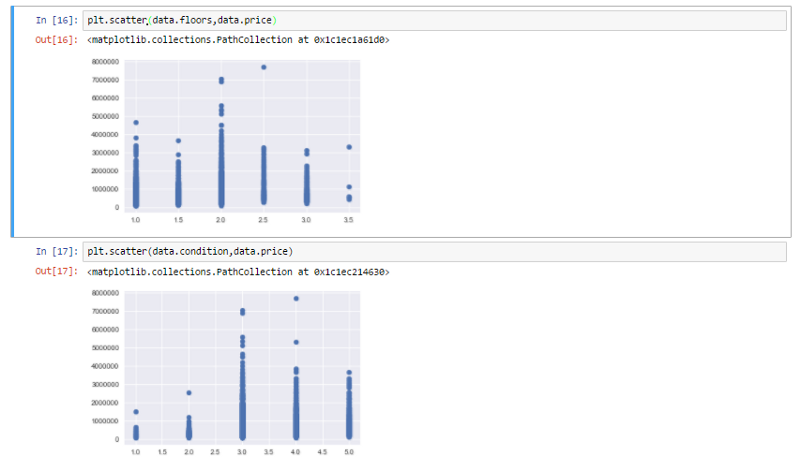


Similarly we compare other factors

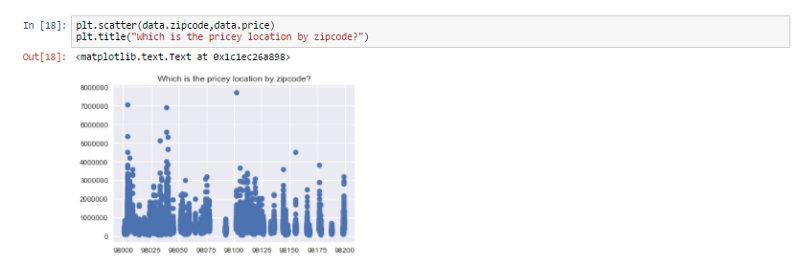
We can see more factors affecting the price



Total sqft including basement vs price and waterfront vs price



Floors vs Price and condition vs Price



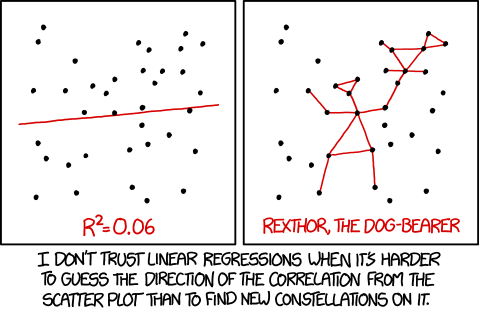
Which location by zipcode is pricey ?

As we can see from all the above representation that many factors are affecting the prices of the house , like square feet which increases the price of the house and even location influencing the prices of the house.

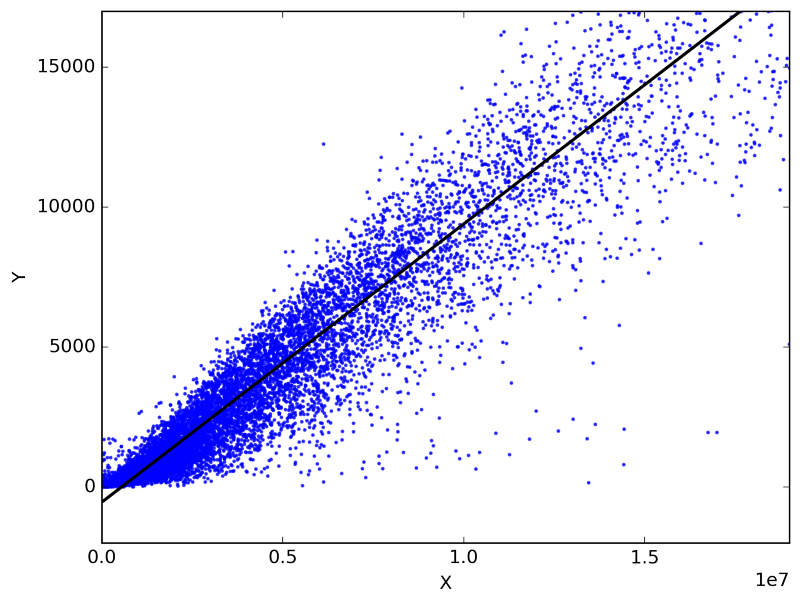
Now that we are familiar with all these representation and can tell our own story let us move and create a model to which would predict the price of the house based upon the other factors such as square feet , water front etc . We are going to see what is linear regression and how do we do it ?

#### Linear Regression :-

In easy words a model in statistics which helps us predicts the future based upon past relationship of variables. So when you see your scatter plot being having data points placed linearly you know regression can help you!



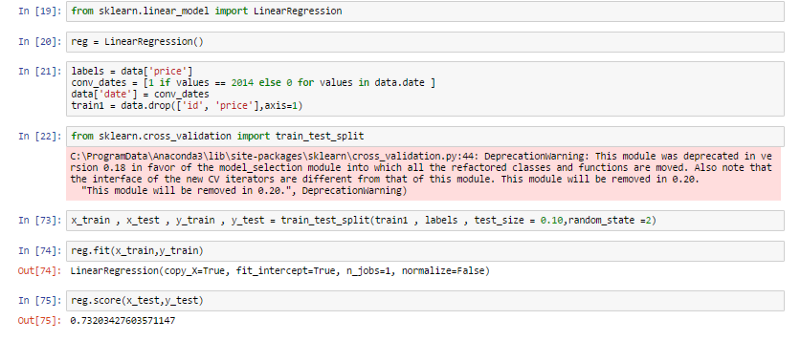
Regression works on the line equation , y=mx+c , trend line is set through the data points to predict the outcome.



Fitting line on the basis of scatter

The variable we are predicting is called the criterion variable and is referred to as Y. The variable we are basing our predictions on is called the predictor variable and is referred to as X. When there is only one predictor variable, the prediction method is called **Simple Regression. and if multiple predictor variable are present then multiple regression.**

Let’s look at the code ,



Linear regression on the data to predict prices

We use train data and test data , train data to train our machine and test data to see if it has learnt the data well or not. Before anything , I want everyone to **remember that the machine is the student and train data is the syllabus and test data is the exam. we see how much the machine has scored and if it scores well are model is successful.**

So what did we do ? Let’s go step by step.

1. We import our dependencies , for linear regression we use sklearn (built in python library) and import linear regression from it.
2. We then initialize Linear Regression to a variable reg.
3. Now we know that prices are to be predicted , hence we set labels (output) as price columns and we also convert dates to 1’s and 0’s so that it doesn’t influence our data much . We use 0 for houses which are new that is built after 2014.
4. We again import another dependency to split our data into train and test.
5. I’ve made my train data as 90% and 10% of the data to be my test data , and randomized the splitting of data by using random\_state.
6. So now , we have train data , test data and labels for both let us fit our train and test data into linear regression model.
7. After fitting our data to the model we can check the score of our data ie , prediction. in this case the prediction is **73%**

The accuracy of the model is lower than our aim of 85. So how do we achieve that 85% target ?

We use a different method , which is very important for weak prediction models such as this.

This might seem to be a bit advanced but if understood is a really brilliant tool to enable better predictions.

For building a prediction model , many experts use **gradient boosting regression ,** so what is gradient boosting ? It is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Now to make it easy , remember how we mapped machine as a student , train data as the syllabus and test data as the exam. let’s try to understand gradient boosting method using the same. So , let’s analyse why our student (machine) didn’t get above 85% ? there could be many reasons few such reasons could be :-

1. Our student forgot few of the topics before giving the exam , Similarly data read by machine can be lost.
2. It could be a weak learner who doesn’t learn by reading but needs visualization. Our machine can be a weak learner and may require decision tree.
3. Even after using newer technique our student may not remember the syllabus so we give our student time to read and understand. Similarly for machine .

Hence for all this problems there is one fix , Gradient descent boosting.