How to work on data science projects

With Shobharani.Polasa

We'll discuss how to work on data science project given in the course in this guide with a basic example on a regression problem and a classification problem. Let's start with a regression problem. I am not discussing problem details here in terms of what all variables explicitly mean, because; the purpose of this guide is to provide you with a process map to follow to work on the projects. This is an example here; follow a similar process to make a basic submission for any of the data science projects.
Please note however, that basic submission, which you do using the process mentioned here will not guarantee a score which is higher than the required threshold in individual projects. At the end of the guide, I'll also discuss how to improve on your submissions.

Regression problem

The problem here is to predict interest rate. You have been provided with train and test data separately, where test data doesn't have response values. You need to use train data to build your model and use that model to make prediction on test data and upload the csv file to LMS.
In this example we have been given ld_train as train data and ld_test as test data, Interest.Rate is our response variable which we need to make prediction model for.

CODE:

```
setwd("/Users/shobharani/Dropbox/March onwards/CBAP with R/Data/")
# you will need to use the location here for your own machine
# where ever you have put data in your machine
ld_train=read.csv("loan_data_train.csv",stringsAsFactors = F)
ld_test= read.csv("loan_data_test.csv",stringsAsFactors = F)
```

You will need same set of "vars" on both train and test, it's easier to manage that if you combine train and test in the beginning and then separate them once you are done with data prep.
Before combining however, we'll need some placeholder column which we can use to differentiate between observations coming from train and test data. Also we'll need to add a column for response to test data so that we have same columns in both train and test. We'll fill test's response column with NAs.

CODE:

```
ld_test$Interest.Rate=NA
ld_train$data='train'
ld_test$data='test'
ld_all=rbind(ld_train,ld_test)
```

ld_all now has all our data, we'll do data prep for the same . First thing to check will be, are there any variables which have come as characters but were supposed to be numbers.

CODE:

```
library(dplyr)
glimpse(ld_all)
```

OUTPUT:

```
## Observations: 2,500
## Variables: 16
## $ ID <int> 79542, 75473, 67265, 80167, 172...
## $ Amount.Requested <chr> "25000", "19750", "2100", "2800...
## $ Amount.Funded.By.Investors <chr> "25000", "19750", "2100", "2800...
## $ Interest.Rate <chr> "18.49%", "17.27%", "14.33%", "...
## $ Loan.Length <chr> "60 months", "60 months", "36 m...
## $ Loan.Purpose <chr> "debt_consolidation", "debt_con...
## $ Debt.To.Income.Ratio <chr> "27.56%", "13.39%", "3.50%", "1...
## $ State <chr> "VA", "NY", "LA", "NV", "OH", "...
## $ Home.Ownership <chr> "MORTGAGE", "MORTGAGE", "OWN", ...
## $ Monthly.Income <dbl> 8606.56, 6737.50, 1000.00, 7083...
## $ FICO.Range <chr> "720-724", "710-714", "690-694"...
## $ Open.CREDIT.Lines <chr> "11", "14", "13", "12", "6", "2...
## $ Revolving.CREDIT.Balance <chr> "15210", "19070", "893", "38194...
## $ Inquiries.in.the.Last.6.Months <int> 3, 3, 1, 1, 2, 2, 0, 1, 0, 1, 0...
## $ Employment.Length <chr> "5 years", "4 years", "< 1 year...
## $ data <chr> "train", "train", "train", "tra...
```

As you can see here, many vars have come as characters, but they should have been numbers.

For Example:
Interest.Rate and Debt.To.Income.Ratio ratio have come as character because there is a % sign in the values.

Lets get rid of that.

CODE:

```
ld_all=ld_all %>%
mutate(Debt.To.Income.Ratio=gsub("%","",Debt.To.Income.Ratio),
Interest.Rate=gsub("%","",Interest.Rate))
```

There are many other vars which have come as character columns because of some odd values in the raw csv data. We can convert them to numeric by using function as.numeric . We'll get the warnings for those odd values. Don't worry about that.

CODE:

```
col_names=c('Debt.To.Income.Ratio','Interest.Rate','Amount.Requested','Amount.Funded.By.Investors', 'Open.CREDIT.Lines','Revolving.CREDIT.Balance')
```

```
for(col in col_names){
ld_all[,col]=as.numeric(ld_all[,col])
}
```

OUTPUT:

```
## Warning: NAs introduced by coercion
## Warning: NAs introduced by coercion
## Warning: NAs introduced by coercion
## Warning: NAs introduced by coercion
```

Next we'll create dummy vars for any remaining categorical vars. I am providing a function which you can use to create dummy vars quickly. However this doesn't merge categories, number of dummies created can be huge in certain cases. It also ignores categories which have very low frequencies in the data. Default frequency cutoff is 100, you can always change that.

CODE:

```
CreateDummies=function(data,var,freq_cutoff=100){
t=table(data[,var])
t=t[t>freq_cutoff]
t=sort(t)
categories=names(t)[-1]
for( cat in categories){
name=paste(var,cat,sep="_")
name=gsub(" ","",name)
name=gsub("-","_",name)
name=gsub("\\?","Q",name)
name=gsub("<","LT_",name)
name=gsub("\\+","",name)
name=gsub(">","GT_",name)
name=gsub("=","EQ_",name)
name=gsub(",","",name)
name=gsub("/","_",name)
data[,name]=as.numeric(data[,var]==cat)
}
data[,var]=NULL
return(data)
}
```

Instead of using these functions on columns one by one, we can use sapply to get all the col names which are of character type.

CODE:

```
char_logical=sapply(ld_all,is.character)
cat_cols=names(ld_all)[char_logical]
cat_cols
```

```
## [1] "Loan.Length" "Loan.Purpose" "State"
## [4] "Home.Ownership" "FICO.Range" "Employment.Length"
## [7] "data"
```

Among these columns we don't need to create dummies for column data and our response column.

CODE:

```
cat_cols=cat_cols[!(cat_cols %in% c('data','Interest.Rate'))]
cat_cols
## [1] "Loan.Length" "Loan.Purpose" "State"
## [4] "Home.Ownership" "FICO.Range" "Employment.Length"
now we can run the function for these columns
for(col in cat_cols){
ld_all=CreateDummies(ld_all,col,50)
# we are using frequency cutoff as 50, there is no magic number here,
# lower cutoffs will simply result in more number of dummy vars
}
glimpse(ld_all)
```

OUTPUT:

```
## Observations: 2,500
## Variables: 61
## $ ID <int> 79542, 75473, 67265, 80167, 17...
## $ Amount.Requested <dbl> 25000, 19750, 2100, 28000, 242...
## $ Amount.Funded.By.Investors <dbl> 25000.00, 19750.00, 2100.00, 2...
## $ Interest.Rate <dbl> 18.49, 17.27, 14.33, 16.29, 12...
## $ Debt.To.Income.Ratio <dbl> 27.56, 13.39, 3.50, 19.62, 23....
## $ Monthly.Income <dbl> 8606.56, 6737.50, 1000.00, 708...
## $ Open.CREDIT.Lines <dbl> 11, 14, 13, 12, 6, 2, 5, 11, 2...
## $ Revolving.CREDIT.Balance <dbl> 15210, 19070, 893, 38194, 3106...
## $ Inquiries.in.the.Last.6.Months <int> 3, 3, 1, 1, 2, 2, 0, 1, 0, 1, ...
## $ data <chr> "train", "train", "train", "tr...
## $ Loan.Length_36months <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, ...
## $ Loan.Purpose_major_purchase <dbl> 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, ...
## $ Loan.Purpose_home_improvement <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ...
## $ Loan.Purpose_other <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, ...
## $ Loan.Purpose_credit_card <dbl> 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, ...
## $ Loan.Purpose_debt_consolidation <dbl> 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_CO <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_NC <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, ...
## $ State_MD <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_OH <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, ...
## $ State_MA <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_VA <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_NJ <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_PA <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_GA <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
```

```
## $ State_IL <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
## $ State_FL <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_TX <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_NY <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ State_CA <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ...
## $ Home.Ownership_RENT <dbl> 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, ...
## $ Home.Ownership_MORTGAGE <dbl> 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, ...
## $ FICO.Range_745_749 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_750_754 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_735_739 <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
## $ FICO.Range_715_719 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_725_729 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_730_734 <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...
## $ FICO.Range_710_714 <dbl> 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_720_724 <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_660_664 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_700_704 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_705_709 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_685_689 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_690_694 <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_665_669 <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ...
## $ FICO.Range_695_699 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_680_684 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_675_679 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ FICO.Range_670_674 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_n_a <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, ...
## $ Employment.Length_8years <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_7years <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_6years <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_1year <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_4years <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_5years <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_3years <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Employment.Length_2years <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ...
## $ Employment.Length_LT_1year <dbl> 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, ...
## $ Employment.Length_10years <dbl> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, ...
```

We could have avoided having too many dummy vars by converting FICO.Range and Employment.Length to numbers by some logic which wouldn't change nature of information in columns.
Now that all the columns are of numeric types we can go ahead and separate training and testing data.

But before we need to check if there are any NA values. We can drop all the obs straight away where the response itself is NA in the train data.

CODE:

```
ld_all=ld_all[!((is.na(ld_all$Interest.Rate)) & ld_all$data=='train'), ]
```

for other columns we can impute by means [you can examine individual columns with NAs and impute with some other logical values as well , in case mean doesn't makes sense in context]. We'll of course ignore columns data and response for the same. Also only obs used for calculating mean should come from train data, so that there is no information leakage.

CODE:

```
for(col in names(ld_all)){
if(sum(is.na(ld_all[,col]))>0 & !(col %in% c("data","Interest.Rate"))){
ld_all[is.na(ld_all[,col]),col]=mean(ld_all[ld_all$data=='train',col],na.rm=T)
}
}
```

Let's separate our two data sets and remove the unnecessary columns that we added while combining them.

CODE:

```
ld_train=ld_all %>% filter(data=='train') %>% select(-data)
ld_test=ld_all %>% filter(data=='test') %>% select(-data,-Interest.Rate)
```

Let's build a model on training data. I am not going to build any complex model, it's up to you to use other algorithms.

CODE:

```
for_vif=lm(Interest.Rate~.-ID,data=ld_train)
# I have excluded ID from the modeling process because it doesnt make sense to keep a serial number
# as predictor

library(car)
sort(vif(for_vif),decreasing = T)[1:3]
```

OUTPUT:

```
## Amount.Requested Amount.Funded.By.Investors
## 16.731657 16.301675
## Employment.Length_10years
## 7.232373
```

CODE:

```
for_vif=lm(Interest.Rate~.-ID-Amount.Funded.By.Investors  data=ld_train)
sort(vif(for_vif)  decreasing = T)[1:3]
```

OUTPUT:

```
## Employment.Length_10years Employment.Length_LT_1year
```

7.230869 4.115441
Employment.Length_2years
3.939294

CODE:

```
for_vif=lm(Interest.Rate~.-ID-Amount.Funded.By.Investors
-Employment.Length_10years,data=ld_train)
sort(vif(for_vif),decreasing = T)[1:3]
```

OUTPUT:

Home.Ownership_MORTGAGE Home.Ownership_RENT
3.810620 3.704167
Loan.Purpose_debt_consolidation
2.788534

now that we have taken out the vars which had redundant information, lets build our model.

CODE:

```
rm(for_vif)
fit=lm(Interest.Rate~.-ID-Amount.Funded.By.Investors
5
-Employment.Length_10years,data=ld_train)
fit=step(fit)
# there is no step function for algos such as dtree, rf or gbm
# variable selection happens there naturally as part of the process
```

in case you are building a linear model, you can further look at output of summary(fit), and remove vars which have high p-values . I am skipping that step here.
once you are done with your model building , now is the time to make prediction on test and submit

CODE:

```
test.predictions=predict(fit,newdata=ld_test)
write.csv(test.predictions,'proper_name.csv',row.names = F)
# dont be lazy chose a proper name as instructed in the problem statement page
```

this csv file is what you need to upload on LMS.

Classification problem:

In this we have been given data bd_train and bd_test and we are required to predict revenue.grid whether it takes value 1 or not.
I will carry out usual data step for this as well. no need for repeat commentary on the same

CODE:

```
setwd("/Users/lalitsachan/Dropbox/March onwards/CBAP with R/Data/")
bd_train=read.csv("bd_train.csv",stringsAsFactors = F)
bd_test=read.csv("bd_test.csv",stringsAsFactors = F)
bd_test$Revenue.Grid=NA
bd_train$data='train'
bd_test$data='test'
bd_all=rbind(bd_train,bd_test)
glimpse(bd_all)
```

OUTPUT:

```
## Observations: 10,155
## Variables: 33
## $ REF_NO <int> 4888, 8525, 3411, 692, 10726, ...
## $ children <chr> "Zero", "Zero", "3", "Zero", "...
## $ age_band <chr> "55-60", "61-65", "31-35", "51...
## $ status <chr> "Widowed", "Partner", "Partner...
## $ occupation <chr> "Retired", "Retired", "Profess...
## $ occupation_partner <chr> "Unknown", "Retired", "Housewi...
## $ home_status <chr> "Own Home", "Own Home", "Own H...
## $ family_income <chr> "<10,000, >= 8,000", ">=35,000...
## $ self_employed <chr> "No", "No", "No", "No", "No", ...
## $ self_employed_partner <chr> "No", "No", "No", "No", "No", ...
## $ year_last_moved <int> 1991, 1983, 1997, 1986, 1994, ...
## $ TVarea <chr> "Meridian", "Granada", "Granad...
## $ post_code <chr> "L15 6LL", "CW11 1RA", "PR8 6S...
## $ post_area <chr> "L15", "CW11", "PR8", "WA10", ...
## $ Average.Credit.Card.Transaction <dbl> 19.49, 9.99, 0.00, 0.00, 0.00,...
## $ Balance.Transfer <dbl> 2.99, 244.41, 0.00, 0.00, 0.00...
## $ Term.Deposit <dbl> 0.00, 152.96, 0.00, 0.00, 19.4...
## $ Life.Insurance <dbl> 436.79, 143.95, 0.00, 17.99, 0...
## $ Medical.Insurance <dbl> 0.00, 86.96, 20.97, 29.96, 58....
## $ Average.A.C.Balance <dbl> 54.88, 181.89, 0.00, 0.00, 34....
## $ Personal.Loan <dbl> 47.95, 38.48, 0.00, 0.00, 0.00...
## $ Investment.in.Mutual.Fund <dbl> 0.00, 37.97, 0.00, 0.00, 0.00,...
## $ Investment.Tax.Saving.Bond <dbl> 33.47, 45.96, 0.00, 0.00, 0.00...
## $ Home.Loan <dbl> 12.96, 28.95, 0.00, 0.00, 0.00...
## $ Online.Purchase.Amount <dbl> 4.99, 3.99, 0.00, 0.00, 0.00, ...
## $ Revenue.Grid <int> 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
## $ gender <chr> "Female", "Male", "Female", "M...
## $ region <chr> "North West", "North West", "N...
## $ Investment.in.Commudity <dbl> 91.85, 127.65, 4.19, 9.59, 15....
## $ Investment.in.Equity <dbl> 25.71, 56.21, 0.00, 0.00, 5.83...
## $ Investment.in.Derivative <dbl> 95.52, 89.20, 3.50, 7.99, 15.6...
## $ Portfolio.Balance <dbl> 249.82, 222.27, 17.05, -72.74,...
## $ data <chr> "train", "train", "train", "tr...
```

if you look at variables post_code and post_area, you will notice that they have more than 2000 unique values , having none of the individual values having frequency higher than a good number , we'll drop those vars.

CODE:

```
bd_all=bd_all %>% select(-post_code,-post_area)
```

We'll create dummies for rest [although we could avoid lot of dummies by converting children, age_band and family_income to numbers]

CODE:

```
char_logical=sapply(bd_all,is.character)
cat_cols=names(bd_all)[char_logical]
cat_cols=cat_cols[!(cat_cols %in% c('data','Revenue.Grid'))]
cat_cols
```

```
## [1] "children" "age_band"
## [3] "status" "occupation"
## [5] "occupation_partner" "home_status"
## [7] "family_income" "self_employed"
## [9] "self_employed_partner" "TVarea"
## [11] "gender" "region"
```

CODE:

```
for(col in cat_cols){
bd_all=CreateDummies(bd_all,col,500)
}
bd_all=bd_all[!((is.na(bd_all$Revenue.Grid)) & bd_all$data=='train'), ]
for(col in names(bd_all)){
if(sum(is.na(bd_all[,col]))>0 & !(col %in% c("data","Revenue.Grid"))){
bd_all[is.na(bd_all[,col]),col]=mean(bd_all[bd_all$data=='train',col],na.rm=T)
}
}
bd_train=bd_all %>% filter(data=='train') %>% select(-data)
bd_test=bd_all %>% filter(data=='test') %>% select(-data,-Revenue.Grid)
```

lets remove vars which have redundant information first on the basis of vif

CODE:

```
for_vif=lm(Revenue.Grid~.-REF_NO                data=bd_train)
sort(vif(for_vif)                               decreasing = T)[1:3]
```

```
## Investment.in.Commudity Investment.in.Derivative Investment.in.Equity
## 245581506 211745544 154274335
```

CODE:

```
for_vif=lm(Revenue.Grid~.-REF_NO-Investment.in.Commudity,data=bd_train)
sort(vif(for_vif),decreasing = T)[1:3]
```

OUTPUT:

```
## Investment.in.Derivative Investment.in.Equity Personal.Loan
## 211745470 154272444 53444347
```

CODE:

```
for_vif=lm(Revenue.Grid~.-REF_NO-Investment.in.Commudity
-Investment.in.Derivative,data=bd_train)
sort(vif(for_vif),decreasing = T)[1:3]
```

OUTPUT:

```
## Investment.in.Equity Online.Purchase.Amount Personal.Loan
## 152138342 34699149 30805247
```

CODE:

```
for_vif=lm(Revenue.Grid~.-REF_NO-Investment.in.Commudity
-Investment.in.Derivative
-Investment.in.Equity,data=bd_train)
sort(vif(for_vif),decreasing = T)[1:3]
```

OUTPUT:

```
## Portfolio.Balance TVarea_Carlton TVarea_Central
## 13.239060 5.538360 5.388449
```

CODE:

```
for_vif=lm(Revenue.Grid~.-REF_NO-Investment.in.Commudity
-Investment.in.Derivative
-Investment.in.Equity
-Portfolio.Balance,data=bd_train)
sort(vif(for_vif),decreasing = T)[1:3]
```

OUTPUT:

```
## TVarea_Carlton TVarea_Central region_SouthEast
## 5.538348 5.388446 5.048021
```

CODE:

```
bd_train$Revenue.Grid=as.numeric(bd_train$Revenue.Grid==1)
fit=glm(Revenue.Grid~.-REF_NO-Investment.in.Commudity
-Investment.in.Derivative
-Investment.in.Equity
-Portfolio.Balance, data=bd_train, family='binomial')
```

OUTPUT:

Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

Don't worry about the warning above. there will always be cases in the data for which it can be said full confidence what their outcome will be , that is what the warning means.
I am skipping the step function and dropping vars here [that doesn't mean you don't need to do that, you should if you are building a linear model]
For making prediction, usually you will be required to submit probabilities in case of classification.

CODE:

```
test.probs=predict(fit,newdata=bd_test,type='response')
# different algos/functions will have different ways of extracting probabilities
# its mentioned in the project problem statement page as well
write.csv(test.probs,'proper_name.csv',row.names = F)
```

upload this csv file to LMS under that project submission tab.

in case that particular project requires you to submit hard outcome instead of probabilities , first find a cutoff on the probability score as discussed in detail in the course. Lets say the cutoff that you have determined is 0.3.

CODE:

```
test.class=as.numeric(predict(fit,newdata=bd_test,type='response')>0.3)
test.class=ifelse(test.class==1,'Yes','No')
write.csv(test.class,'proper_name.csv',row.names = F)
```

Improving your score

• Try a nonlinear algorithm such as dtrees, rfs , extra trees or gbm , they almost always give better performance than a simple linear model.
• Tune your parameters for above mentioned algos
• Try stacking

How do I assess my model performance before submission?

You can break your given train data into two parts, build model on one check its performance on two, which will give you an idea about how your model might perform after submission. You cannot assess performance on given test data because response has been removed from it.

End Notes:

In this article we covered “How to work on data science projects” can be used to work on simple data science project. I consider this one of the most important articles. In real-life industry scenarios, you will quite often face the situation of having to explain the model’s results to the stakeholder (who is usually a non-technical person).
Your chances of getting the model approved will lie in how well you are able to explain how and why the model is behaving the way it is. Plus it’s always a good idea to always explain any model’s performance to you in a way that a layman will understand – this is always a good practice!

Hope this helps, feel free to suggest, you can connect with me in the comments section below if you have any questions or feedback on this article.
And next part is coming soon so stay tuned!

About the Author:

This is SHOBHARANI,
An avid reader and blogger, who loves exploring the endless world of data science and artificial intelligence, Fascinated by the limitless applications of ML and AI; eager to learn and discover the depths of data science.