#Load Packages

```
library(readr)
library(caret)
library(pROC)
library(dplyr)
library(h2o)
h2o.init(nthreads=-1)
library(rattle.data)
```

#Import Data

```
c_card <- h2o.importFile("E:/creditcard.csv")
dim(c_card)
head(c_card, n=5)
h2o.table(c_card$Class)
```

	Class <dbl>	Count <dbl>
1	0	284315
2	1	492

#Convert data to h2o and split them

```
#convert data to h2o
c_card <- as.h2o(c_card)
```

#Split data into classes

```
set.seed(123)
training <- twoClassSim(floor(0.001*nrow(c_card)), intercept = -13)
testing  <- twoClassSim(floor(0.75*nrow(c_card)), intercept = -13)

table(training$Class)

nmin <- sum(training$Class == "Class2")
nmin
```

```
Class1 Class2
   249     35
[1] 35
```

#Train two random forest models: one using down-sampling and another with the standard sampling procedure

```
ctrl <- trainControl(method = "cv",
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary)
```

#Down-sampling

```
set.seed(2)
rfDownsampled <- train(Class ~ ., data = training,
```

```
                method = "rf",
                ntree = 1500,
                tuneLength = 5,
                metric = "ROC",
                trControl = ctrl,
                #sample by class
                strata = training$Class,
                #specify that the number of samples selected within each class
                sampsize = rep(nmin, 2))
```

#Standard sampling

```
set.seed(2)
rfUnbalanced <- train(Class ~ ., data = training,
                method = "rf",
                ntree = 1500,
                tuneLength = 5,
                metric = "ROC",
                trControl = ctrl)
```

#Compute the test set ROC curves for both procedures

#Compute down-sampled ROC test set

```
downProbs <- predict(rfDownsampled, testing, type = "prob")[,1]

downsampledROC <- roc(response = testing$Class,
                predictor = downProbs,
                levels = rev(levels(testing$Class)))
```

#Compute unbalanced ROC test set

```
unbalProbs <- predict(rfUnbalanced, testing, type = "prob")[,1]
unbalROC <- roc(response = testing$Class,
            predictor = unbalProbs,
            levels = rev(levels(testing$Class)))
```

#Plot the curves and determine the area under each curve

#Plot down-sample ROC curve chart

```
plot(downsampledROC, col = rgb(1, 0, 0, .5), lwd = 2,
    response = testing$Class,
    predictor = downProbs,
    levels = rev(levels(testing$Class)))
```

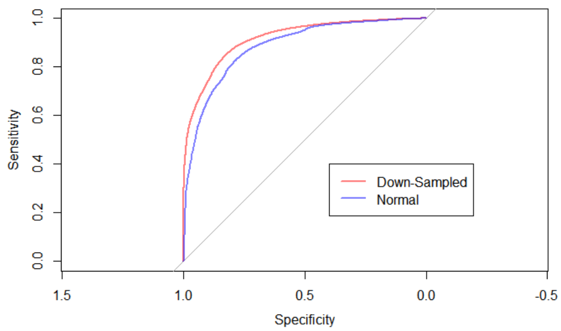
#Plot unbalanced ROC curve chart

```
plot(unbalROC, col = rgb(0, 0, 1, .5), lwd = 2,
    add= TRUE,
    response = testing$Class,
    predictor = unbalProbs,
    levels = rev(levels(testing$Class)))
```

```
legend(.4, .4,
    c("Down-Sampled", "Normal"),
    lwd = rep(2, 1),
    col = c(rgb(1, 0, 0, .5), rgb(0, 0, 1, .5)))

#Training performance
getTrainPerf(rfDownsampled)

#Area under the curve for down-sample
auc(downsampledROC)
```



TrainROC <dbl>	TrainSens <dbl>	TrainSpec <dbl>	method <chr>
0.9321667	0.9756667	0.5166667	rf

```
Area under the curve: 0.9159
```

```
#Splitting the dataframe
# the ratios should sum up to to be less than 1.0.
cc_splits <- h2o.splitFrame(data = c_card,
                    ratios = c(0.6,0.2),
                    destination_frames = c("train", "valid", "test"),
                    seed = 1234)

train <- cc_splits[[1]]
valid <- cc_splits[[2]]
test  <- cc_splits[[3]]
valid


# Identify predictors and response
y <- "Class"
x <- setdiff(names(train), y)

# For binary classification, response should be a factor
train[,y] <- as.factor(train[,y])
test[,y] <- as.factor(test[,y])
valid[,y] <- as.factor(valid[,y])
```

 # Number of CV folds (to generate level-one data for stacking)

```
nfolds <- 5
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	0	-1.359807	-0.07278117	2.53634674	1.3781552	-0.3383208	0.46238778	0.23959855	0.0986979	0.3637870	0.09079417	-0.55159953
2	1	-1.358354	-1.34016307	1.77320934	0.3797796	-0.5031981	1.80049938	0.79146096	0.2476758	-1.5146543	0.20764287	0.62450146
3	12	1.103215	-0.04029621	1.26733209	1.2890915	-0.7359972	0.28806916	-0.58605679	0.1893797	0.7823329	-0.26797507	-0.45031128
4	18	1.166616	0.50212009	-0.06730031	2.2615692	0.4288042	0.08947352	0.24114658	0.1380817	-0.9891624	0.92217497	0.74478579
5	22	-1.946525	-0.04490051	-0.40557007	-1.0130573	2.9419677	2.95505340	-0.06306315	0.8555463	0.0499669	0.57374251	-0.08125651
6	25	1.114009	0.08554609	0.49370249	1.3357600	-0.3001886	-0.01075378	-0.11876002	0.1886167	0.2056868	0.08226226	1.13355567

V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
-0.61780086	-0.9913898	-0.3111694	1.4681770	-0.4704005	0.2079712	0.02579058	0.4039930	0.25141210	-0.018306778	0.277837576
1.06523531	0.4890950	-0.1437723	0.6355581	0.4639170	-0.1148047	-0.18336127	-0.1457830	-0.06908314	-0.225775248	-0.638671953
0.06608369	0.7172927	-0.1659459	2.3458649	-2.8900832	1.1099694	-0.12135931	-2.2618571	0.52497973	0.247998153	0.771679402
0.17822823	0.5077569	-0.2879237	-0.6314181	-1.0596472	-0.6840928	1.96577500	-1.2326220	-0.20803778	-0.108300452	0.005273597
0.53819555	1.3458516	-1.1196698	0.1751211	-0.4514492	-0.2370332	-0.03819479	0.8034869	0.40854236	-0.009430697	0.798278495

ws | 14-24 of 31 columns

V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0.4039930	0.25141210	-0.018306778	0.277837576	-0.1104739	0.06692807	0.1285394	-0.1891148	0.133558377	-0.02105305	149.62	0
-0.1457830	-0.06908314	-0.225775248	-0.638671953	0.1012880	-0.33984648	0.1671704	0.1258945	-0.008983099	0.01472417	2.69	0
-2.2618571	0.52497973	0.247998153	0.771679402	0.9094123	-0.68928096	-0.3276418	-0.1390966	-0.055352794	-0.05975184	378.66	0
-1.2326220	-0.20803778	-0.108300452	0.005273597	-0.1903205	-1.17557533	0.6473760	-0.2219288	0.062722849	0.06145763	123.50	0
0.8034869	0.40854236	-0.009430697	0.798278495	-0.1374581	0.14126698	-0.2060096	0.5022922	0.219422230	0.21515315	69.99	0

5 rows | 21-32 of 31 columns

```
[56904 rows x 31 columns]
```

#Build RF Model

#2000 is recommended

#30 is recommended

```
cc_rfmodel <- h2o.randomForest(
      training_frame = train,
      validation_frame = valid,
      x=x,
      y= 'Class',
      model_id = "cc_rfmodel",
      ntrees = 2000,
      max_depth = 20 ,
      stopping_rounds = 2,
      stopping_tolerance = 1e-2,
      score_each_iteration = T,
      seed=1234)
```

Get the AUC on the validation set

```
h2o.auc(h2o.performance(cc_rfmodel , newdata = test))
```

```
Rf_predictions<-h2o.predict(object = cc_rfmodel, newdata = valid)
```

```
  |=========================================================================| 100%
[1] 0.935577
  |=========================================================================| 100%
```

#Hyper parameters and h2o grid search

```
hyper_params = list( ntrees = seq(100,1000,200),
             max_depth=seq(5, 12, 8))

cc.grid <- h2o.grid(hyper_params = hyper_params,
           search_criteria = list(strategy = "Cartesian"),
           algorithm="randomForest",
           grid_id="rf.grid",
           x = x,
           y = 'Class',
           training_frame = train,
           validation_frame = valid,
           seed = 123456,
           stopping_rounds = 5,
           stopping_tolerance = 1e-8,
           stopping_metric = "AUC",
           score_tree_interval = 10)
cc.grid
summary(cc.grid)
```

```
H2O Grid Details
================

Grid ID: rf.grid
Used hyper parameters:
  -  max_depth
  -  ntrees
Number of models: 20
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss
H2O Grid Summary
================

Grid ID: rf.grid
Used hyper parameters:
  -  max_depth
  -  ntrees
Number of models: 20
  -  rf.grid_model_19
  -  rf.grid_model_15
  -  rf.grid_model_7
  -  rf.grid_model_11
  -  rf.grid_model_3
  -  rf.grid_model_6
  -  rf.grid_model_10
  -  rf.grid_model_14
  -  rf.grid_model_18
  -  rf.grid_model_2
  -  rf.grid_model_9
  -  rf.grid_model_13
  -  rf.grid_model_5
  -  rf.grid_model_17
  -  rf.grid_model_1
  -  rf.grid_model_4
  -  rf.grid_model_8
  -  rf.grid_model_16
  -  rf.grid_model_12
  -  rf.grid_model_0

Number of failed models: 0
```

	max_depth <chr>	ntrees <chr>	model_ids <chr>	logloss <chr>
1	11	900	rf.grid_model_19	0.003519959285996883
2	11	700	rf.grid_model_15	0.003519959285996883
3	11	300	rf.grid_model_7	0.003519959285996883
4	11	500	rf.grid_model_11	0.003519959285996883
5	11	100	rf.grid_model_3	0.0035363662701978233
6	8	300	rf.grid_model_6	0.003710169187427133
7	8	500	rf.grid_model_10	0.003710169187427133
8	8	700	rf.grid_model_14	0.003710169187427133
9	8	900	rf.grid_model_18	0.003710169187427133
10	8	100	rf.grid_model_2	0.0037291343992123006
11	5	500	rf.grid_model_9	0.003973746740935194
12	5	700	rf.grid_model_13	0.003973746740935194
13	5	300	rf.grid_model_5	0.003973746740935194
14	5	900	rf.grid_model_17	0.003973746740935194
15	5	100	rf.grid_model_1	0.00399425533640768
16	2	300	rf.grid_model_4	0.004827279370480342
17	2	500	rf.grid_model_8	0.004827279370480342
18	2	900	rf.grid_model_16	0.004827279370480342
19	2	700	rf.grid_model_12	0.004827279370480342
20	2	100	rf.grid_model_0	0.004864453690512188

20 rows

 # Grid search top 5 models

rf.grid.sorted <- h2o.getGrid("rf.grid", sort_by="auc", decreasing = TRUE)
print(rf.grid.sorted)

f <- function(i) g(i-1)
g <- function(i) if(i <= 0) 0 else g(i-1) + f(i)

for (f in 1:5)
 {topModels <- h2o.getModel(rf.grid.sorted@model_ids[[f]])
 print(h2o.auc(h2o.performance(topModels, valid = TRUE)))}

```
H2O Grid Details
================

Grid ID: rf.grid
Used hyper parameters:
  -  max_depth
  -  ntrees
Number of models: 5
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
[1] 0.9638342
[1] 0.9638342
[1] 0.9638342
[1] 0.9638342
[1] 0.961974
```

	max_depth <chr>	ntrees <chr>	model_ids <chr>	auc <chr>
1	5	500	rf.grid_model_2	0.9638341984814208
2	5	700	rf.grid_model_3	0.9638341984814208
3	5	300	rf.grid_model_1	0.9638341984814208
4	5	900	rf.grid_model_4	0.9638341984814208
5	5	100	rf.grid_model_0	0.9619739752901754

5 rows

#Best model details

best.rfmodel <- h2o.getModel(rf.grid.sorted@model_ids[[1]])
summary(best.rfmodel)
scoring_history <- as.data.frame(best.rfmodel@model$scoring_history)
ntrees <- best.rfmodel@model$model_summary$number_of_trees

```
Model Details:
==============

H2OBinomialModel: drf
Model Key:  rf.grid_model_2
Model Summary:

H2OBinomialMetrics: drf
** Reported on training data. **
** Metrics reported on Out-Of-Bag training samples **

MSE:  0.000458891
RMSE:  0.02142174
LogLoss:  0.002912689
Mean Per-Class Error:  0.107216
AUC:  0.9586499
Gini:  0.9172998

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

Maximum Metrics: Maximum metrics at their respective thresholds

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`
H2OBinomialMetrics: drf
** Reported on validation data. **

MSE:  0.000617044
RMSE:  0.02484037
LogLoss:  0.003973747
Mean Per-Class Error:  0.09648874
AUC:  0.9638342
Gini:  0.9276684

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

Maximum Metrics: Maximum metrics at their respective thresholds

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`


Scoring History:

---

Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:

---
```

	number_of_trees <chr>	number_of_internal_trees <chr>	model_size_in_bytes <chr>	min_depth <chr>	max_depth <chr>	mean_depth <chr>	min_leaves <chr>	max_leaves <chr>	mean_leaves <chr>
1	300	300	92138	5	5	5.00000	11	26	19.43667

1 row

	0 <chr>	1 <chr>	Error <chr>	Rate <chr>
0	170876	25	0.000146	=25/170901
1	60	220	0.214286	=60/280
Totals	170936	245	0.000497	=85/171181

3 rows

	0 <chr>	1 <chr>	Error <chr>	Rate <chr>
0	56777	18	0.000317	=18/56795
1	21	88	0.192661	=21/109
Totals	56798	106	0.000685	=39/56904

3 rows

	metric <chr>	threshold <chr>	value <chr>	idx <chr>
1	max f1	0.353303	0.838095	176
2	max f2	0.196998	0.822604	204
3	max f0point5	0.528018	0.881579	148
4	max accuracy	0.377136	0.999503	172
5	max precision	0.988820	1.000000	0
6	max recall	0.000274	1.000000	399
7	max specificity	0.988820	1.000000	0
8	max absolute_mcc	0.353303	0.839722	176
9	max min_per_class_accuracy	0.000487	0.915074	384
10	max mean_per_class_accuracy	0.001359	0.933481	365

10 rows

	metric <chr>	threshold <chr>	value <chr>	idx <chr>
1	max f1	0.153753	0.818605	103
2	max f2	0.153753	0.811808	103
3	max f0point5	0.457783	0.871965	83
4	max accuracy	0.457783	0.999350	83
5	max precision	0.986130	1.000000	0
6	max recall	0.000282	1.000000	399
7	max specificity	0.986130	1.000000	0
8	max absolute_mcc	0.153753	0.818341	103
9	max min_per_class_accuracy	0.000503	0.908257	360
10	max mean_per_class_accuracy	0.000503	0.916872	360

10 rows

	timestamp <chr>	duration <chr>	number_of_trees <chr>	training_rmse <chr>	training_logloss <chr>	training_auc <chr>	training_lift <chr>
1	2018-08-01 20:04:18	33.464 sec	0				
2	2018-08-01 20:04:18	33.849 sec	10	0.02322	0.00346	0.94424	84.99057
3	2018-08-01 20:04:19	34.299 sec	20	0.02215	0.00308	0.93597	85.70477
4	2018-08-01 20:04:19	34.789 sec	30	0.02187	0.00304	0.93907	85.70477
5	2018-08-01 20:04:20	35.308 sec	40	0.02167	0.00296	0.96158	85.70477

5 rows | 1-8 of 13 columns

training_classification_error <chr>	validation_rmse <chr>	validation_logloss <chr>	validation_auc <chr>	validation_lift <chr>	validation_classification_error <chr>
0.00063	0.02509	0.00401	0.93116	79.15315	0.00074
0.00054	0.02489	0.00403	0.94458	76.79818	0.00072
0.00054	0.02469	0.00401	0.94289	81.51386	0.00069
0.00050	0.02457	0.00396	0.94366	81.22884	0.00067

5 rows | 9-14 of 13 columns

	timestamp <chr>	duration <chr>	number_of_trees <chr>	training_rmse <chr>	training_logloss <chr>	training_auc <chr>	training_lift <chr>
26	2018-08-01 20:04:39	54.023 sec	250	0.02141	0.00291	0.95982	86.06188
27	2018-08-01 20:04:40	55.317 sec	260	0.02142	0.00292	0.95864	86.77608
28	2018-08-01 20:04:41	56.618 sec	270	0.02145	0.00292	0.95930	86.77608
29	2018-08-01 20:04:43	57.960 sec	280	0.02149	0.00292	0.95807	86.41898
30	2018-08-01 20:04:44	59.333 sec	290	0.02144	0.00291	0.95825	86.41898
31	2018-08-01 20:04:45	1 min 0.741 sec	300	0.02142	0.00291	0.95865	86.41898

6 rows | 1-8 of 13 columns

training_classification_error <chr>	validation_rmse <chr>	validation_logloss <chr>	validation_auc <chr>	validation_lift <chr>	validation_classification_error <chr>
0.00048	0.02484	0.00397	0.96419	81.85532	0.00069
0.00049	0.02486	0.00397	0.96400	82.42974	0.00069
0.00049	0.02489	0.00398	0.96421	82.42974	0.00069
0.00049	0.02491	0.00398	0.96374	82.42974	0.00069
0.00050	0.02487	0.00398	0.96387	82.42974	0.00069
0.00050	0.02484	0.00397	0.96383	82.42974	0.00069

6 rows | 9-14 of 13 columns

	variable <chr>	relative_importance <chr>	scaled_importance <chr>	percentage <chr>
1	V17	10258.659180	1.000000	0.207514
2	V12	8509.955078	0.829539	0.172141
3	V14	7414.674316	0.722772	0.149986
4	V11	4565.718262	0.445060	0.092356
5	V10	4333.534668	0.422427	0.087660

5 rows

	variable <chr>	relative_importance <chr>	scaled_importance <chr>	percentage <chr>
25	V19	136.827942	0.013338	0.002768
26	V25	113.561798	0.011070	0.002297
27	V28	101.732544	0.009917	0.002058
28	Amount	80.560020	0.007853	0.001630
29	V23	74.316055	0.007244	0.001503
30	V13	61.761089	0.006020	0.001249

6 rows

#Performance measures
rforest.pred <- h2o.predict(best.rfmodel,test)
head(rforest.pred)

performance<-h2o.performance(best.rfmodel, test)
performance

```
  |======================================================================| 100%
H2OBinomialMetrics: drf

MSE:  0.0005650092
RMSE:  0.02376992
LogLoss:  0.003419752
Mean Per-Class Error:  0.08755526
AUC:  0.9655486
Gini:  0.9310973

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

Maximum Metrics: Maximum metrics at their respective thresholds

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`
```

	predict <fctr>	p0 <dbl>	p1 <dbl>
1	0	0.9996989	0.0003010813
2	0	0.9997177	0.0002822872
3	0	0.9996117	0.0003883483
4	0	0.9997177	0.0002822872
5	0	0.9996765	0.0003235141
6	0	0.9997136	0.0002863868

6 rows

	0 <chr>	1 <chr>	Error <chr>	Rate <chr>
0	56599	20	0.000353	=20/56619
1	18	85	0.174757	=18/103
Totals	56617	105	0.000670	=38/56722

rows

	metric <chr>	threshold <chr>	value <chr>	idx <chr>
1	max f1	0.166998	0.817308	100
2	max f2	0.166998	0.822050	100
3	max f0point5	0.596819	0.882353	68
4	max accuracy	0.383777	0.999365	78
5	max precision	0.979503	1.000000	0
6	max recall	0.000282	1.000000	399
7	max specificity	0.979503	1.000000	0
8	max absolute_mcc	0.166998	0.817010	100
9	max min_per_class_accuracy	0.000493	0.902913	364
10	max mean_per_class_accuracy	0.036898	0.926699	137

10 rows

#Plot ROC
tpr<-as.data.frame(h2o.tpr(performance))
fpr<-as.data.frame(h2o.fpr(performance))

ROC<-merge(tpr,fpr,by='threshold')
head(ROC)

	threshold <dbl>	tpr <dbl>	fpr <dbl>
1	0.0002823373	1.0000000	1.0000000
2	0.0002837246	0.9805825	0.6595842
3	0.0002861136	0.9805825	0.5299458
4	0.0002895072	0.9805825	0.4537170
5	0.0002932610	0.9708738	0.4052703
6	0.0002974800	0.9708738	0.3584309

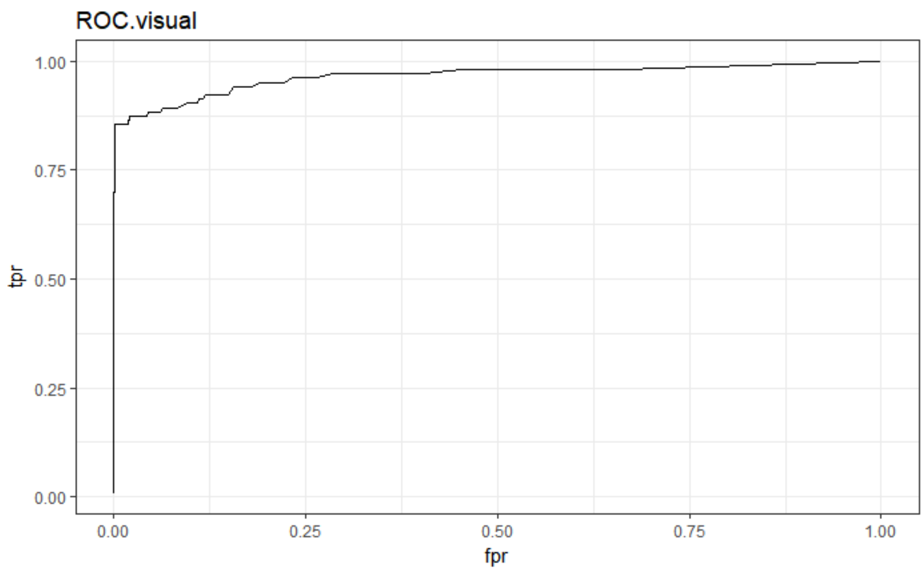
6 rows

#Plot ROC curve and save output file

```
pdf(file="C:/Users/daveh/Documents/APAN 5420/EDA VIIII/eda9ROCcurve.pdf")

ggplot(data = ROC, aes(x = fpr, y = tpr)) +
  theme_bw() +
  geom_line() +
  ggtitle("ROC.visual")

while (!is.null(dev.list()))  dev.off()
```



#Plotting Random Forest Model Precision-Threshold Curve

```
h2o.F1(performance)

rf.precision <- as.data.frame(h2o.precision(performance))
rf.recall <- as.data.frame(h2o.recall(performance))
rf.pre.recall <-merge(precision,recall,by='threshold')

head(rf.pre.recall)
plot(rf.pre.recall)

ggplot(rf.pre.recall, aes(x = precision, y = threshold)) +
  theme_bw() +
  geom_line() +
  ggtitle("thres-hold & precision")
```

	threshold <dbl>	f1 <dbl>
1	0.9795034	0.01923077
2	0.9744131	0.03809524
3	0.9711338	0.05660377
4	0.9705332	0.09259259
5	0.9601453	0.11009174

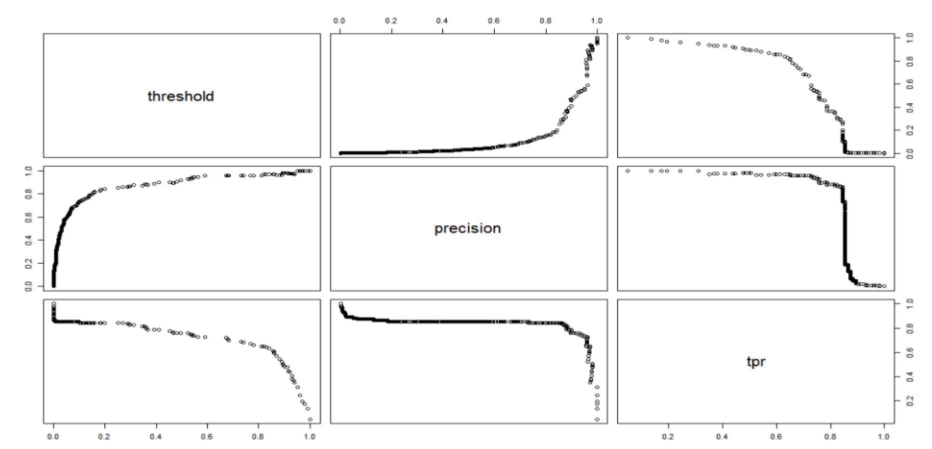
5 rows

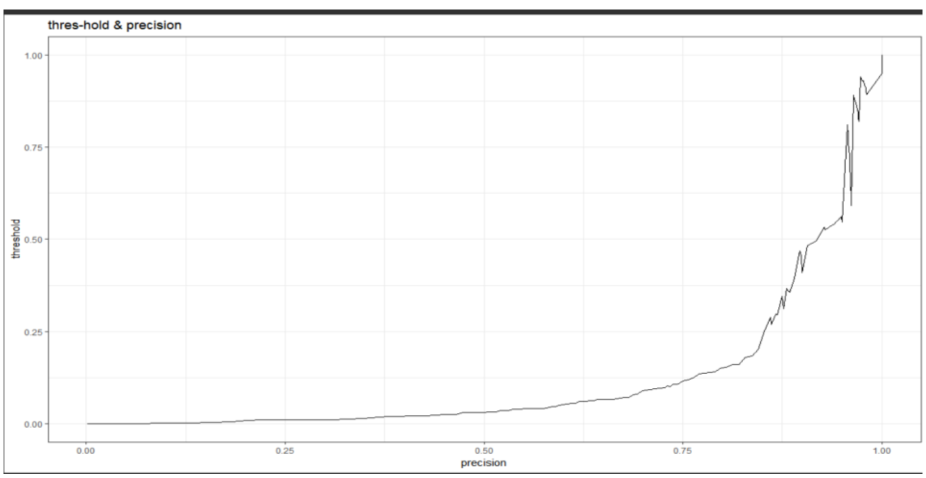
	threshold <dbl>	f1 <dbl>
395	0.0002974800	0.009757525
396	0.0002932610	0.008639682
397	0.0002895072	0.007801336
398	0.0002861136	0.006686749
399	0.0002837246	0.005379637
400	0.0002823373	0.003625165

6 rows

	threshold <dbl>	precision <dbl>	tpr <dbl>
1	0.000000e+00	0.001815874	1
2	8.806628e-05	0.001815970	1
3	9.181361e-05	0.001847600	1
4	9.469812e-05	0.001886309	1
5	9.715304e-05	0.001968466	1
6	9.915449e-05	0.002028318	1

6 rows





#pdf output file name
```
pdf(file="C:/Users/daveh/Documents/APAN 5420/EDA VIIII/eda9pre.pdf")

ggplot(rf.pre.recall, aes(x = tpr, y = precision)) +
  theme_bw() +
```

```
  geom_line() +
  ggtitle("precision and recall")

while (!is.null(dev.list()))dev.off()
```