Machine Learning & Statistical Data Mining

- KDD Cup 2009 Customer relationship prediction -

Project Team Members: Audrey E.
Frederic M.
John D.

Table of Contents

Definition of Terms

ML: Machine Learning

Software Dependencies

R - Version 3.3.3 & R Studio - Version 0.99.903

Hardware

Laptops - Windows 10 64bits platform, supported by a Intel Core i7-67000HQ processor / RAM: DDR4 8GB.
Servers – Computing Department Goldsmith Servers

How to Run the Code

Each ML model is contained in a folder corresponding to the model family (e.g. lda) under the folder \models. Each model is declined at least in 3 version, one for each target labels (c.f. Table 1).

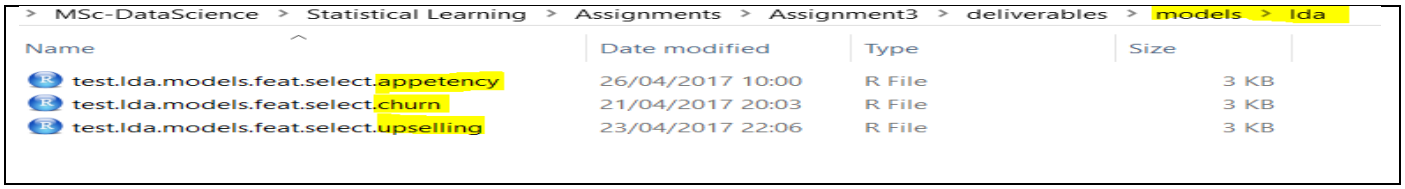


Table 1 - lda model files

Each model file contains an *isServerRun* switch. When set to True, it means the configuration is set for run on the server. When set to FALSE, it is set for a local run.
The user needs to set *working directory* path per the model accordingly, as shown in table 2.

```
#TRUE: server settings / FALSE: local settings
isServerRun = FALSE
model.name <- "LDA"
model.formula <-as.formula("appetency ~ V84+V118+V165+V224+V116+V155+V14+V90")

print(sprintf("model: %s formula: %s", model.name, deparse(model.formula)))

########################################
############## Drive Config ############
########################################
if(isServerRun){
  setwd('/host/dsm1/fmare001/stats/svm/deliverables')
}else{
  #setwd('C:/Users/audrey.ekuban/dev/goldsmiths/mlsdm/assignment3')
  #setwd('C:/Users/john/dev/goldsmiths/mlsdm/assignment3/submission')
  setwd('C:/Users/Fred/Desktop/Studies/MSc-DataScience/Statistical Learning/Assignments/Assignment3/deliverables')
}
```

Table 2 – The *isServerRun* setup and the server/local working directory configuration

A description of each R file, its implementation and interaction with the rest of the solution suite is detailed in Appendix C.

The predicted target labels

The predicted target label files are available in the \prediction folder. The CSV files contain the predicted target label for each category generated by the best model for each category.

The .RDA files contain the trained models' state. The only exception is for the NaGroups saved object. It was 50Mb large, therefore it was not possible to zip to a size that would be acceptable. Consequently, the download link is provided in the \prediction\NaGroups_ReadMe.txt

Abstract

The purpose of this experiment was build an efficient and accurate machine learning (ML) predictive tool to empower the Orange marketing department to forecast the propensity of customers to switch provider (churn), buy new products (appetency) and upgrade their existing Orange contract (upselling). Due to the nature of the training data, largely encrypted coupled with a high proportion of missing data and a high number of category levels, it was necessary to study the data in depth in attempt to discover potential pattern that could help in directing the analysis. As the results of the data analysis was inconclusive, a number of different ML approaches have been investigated in an effort maximum the AUC of the target labels, namely appetency, churn and upselling. The best models retained for each of the target labels were: the Linear Discriminant Analysis (LDA) Scenario 1 (91% training AUC) for predicting appetency, the LDA Scenario 1 (72% training AUC) for forecasting churn and the "NA Group" model (78% training AUC) for predicting upselling.

Introduction

The aim of this study is to build an efficient and accurate machine learning (ML) predictive tool to empower the Orange marketing department to forecast the propensity of customers to switch provider (churn), buy new products (appetency) and upgrade their existing Orange contract (upselling).

This project is based on a modified version of the data provided in the KDD Cup 2009. The analysis and prediction phases were challenging due to the following. First, the training data length and cardinality. It contains 33001 records spread over 230 attributes. This represents a matrix of more than 7.5 million data points. This has an impact on the server memory foot print. It also impedes ML models to complete the prediction task in a time efficient manner. Second, the schema contains a mixture of numerical and categorical attributes. This has a direct impact on feature selection and ML models to choose from, as these models need to handle both type of data.

Third, the initial data analysis phase uncovered the presence of missing data. Missing data is known to be problematic for ML models. Therefore, it had to be dealt with in advance of generating prediction. It also revealed the presence of a high category levels break down. Early prototyping showed that some models could not handle categories containing more than 53 levels. Strategies were put in place and tested to circumvent this issue.

Lastly, target labels showed the existence of a significant class unbalancing. This is known to skew ML model prediction, and therefore it was addressed accordingly.

The remainder of this study is divided in the following sections. The first part relates to the upfront data analysis. This was a central part of the project, prior to engaging into running ML models. The outcome of the analysis was an enabler to devise a strategy for coping with i) missing data, ii) category levels, iii) class unbalancing and the selection the 'best' model suite to tackle the problem at hand. The second part addresses the data pre-processing steps in detail. Data pre-processing was intentionality not applied in the same fashion to all tested ML models. The main reason was that models works in different manner and therefore have different data needs. A secondary reason was pragmatism. Testing some of the models such as Tree based or SVM models can be time consuming. The fourth section describes the tested models, their assumptions and results. The last section concludes on the 'best' proposed model for each target label, namely 'appetency', 'churn' and 'upselling'. All the models chosen in this experiment are supervised classification models and were implemented with the Caret package. The predicted target labels CSVs are available in the \prediction folder.

Data Description

The provided for the challenge is contained in three csv file.

- Train_X.csv contains the train data, a list of 33001 rows with tab delimited columns. The column data is heterogenous, i.e. it contains numerical and categorical variables and most of the data is encrypted.
- TrainY.csv contains the three target labels, namely appetency, churn and upselling.
- Test_X.csv represents the test data. It is used at the end of the experiment to generate the expected test labels from the best selected models.

Data Analysis & Visualisation

We were careful to begin with exploratory analysis of the data, starting out with a cursory visual inspection and then proceeding with plots and summaries of the variables. One thing which became immediately apparent was that although the data appeared to be reasonably clean, in the sense of being correctly formatted (e.g. no invalid numbers), there were a large number of missing values in a large number of variables. Grouping these together in R hinted at the possibility of structure behind the patterns of missing values – in that a number of variables had the same, very specific, number of missing values as some of the other variables.

Predictor	# NA	Predictor	# NA	Predictor	# NA	Predictor	# NA
V175	32884	V41	32558	V202	3294	V21	4765
V207	32884	V65	32558	V206	3294	V121	4765
V26	32643	V104	32558	V219	3294	V197	4765
V100	32643	V158	32558	V222	3294		
V173	32643	V194	32558	V230	3294		

Table 1: Extract from "N/A Count" grouping table

We additionally built a “Data Dictionary” of the training data; an Excel spreadsheet listing each variable, it’s type (character, numeric-continuous, numeric-discrete), range (min and max values), number of levels (in the case of potential factors) as well as notes on whether the variable was likely to represent e.g. monetary values or units of time. We noticed that in a large number of cases, numeric-discrete variables were multiples of a common denominator – for example, multiples of 3, 7, 12, 14, 24 or even 36. Ultimately, we were not able to successfully “decode” these variables, for example to identify which of them represented customer IDs, or the length of a customer’s contract – or even which of them represented minutes, hours, days or account balances. However, we were able to feed this information into decisions about whether to treat variables as numeric or as factors, in the case of discrete numeric values, and which types of imputation would be best suited as a result.

VAR	TYPE	#LEVELS	MIN	MAX	NOTES
V101	Numeric (continuous)	6	-267.52	12286.72	Possibly monetary, mostly 1 and 2dp - some up to 5 though.
V102	Numeric (discrete)	4	0	54	Multiples of 18
V103	Numeric (discrete)	131	0	6578865	Multiples of 15
V104	Numeric (discrete)	84	0	16784	Multiples of 16
V105	Numeric (discrete)	302	0	15235500	Multiples of 14
V106	NA				
V107	Numeric (discrete)	16	0	720	Multiples of 36
V108	Numeric (discrete)	10	0	160	Multiples of 16
V109	NA				
V110	Numeric (continuous)		0	3515.52	Possibly monetary
V111	Numeric (continuous)		0	14	Large precision (7dp), small values.
V112	Character	6			

Table 2: Extract from Data Dictionary

Plotting bar charts and histograms of each variable was also a useful exercise – and moreover plotting a variable's distribution in the training set next to its distribution in the test set. These allowed us to (loosely) identify each variable as belonging to one of six types:

- Character variables which are likely to be factors.
- Character variables which are unlikely to be factors - due to the number of unique values.
- Discrete numeric variables, which are likely to be factors.
- Discrete numeric variables, which unlikely to be factors - due to the number of unique values.
- Discrete numeric variables which are possibly factors, but more likely e.g. time - measured in discrete units.
- Continuous numeric variables.

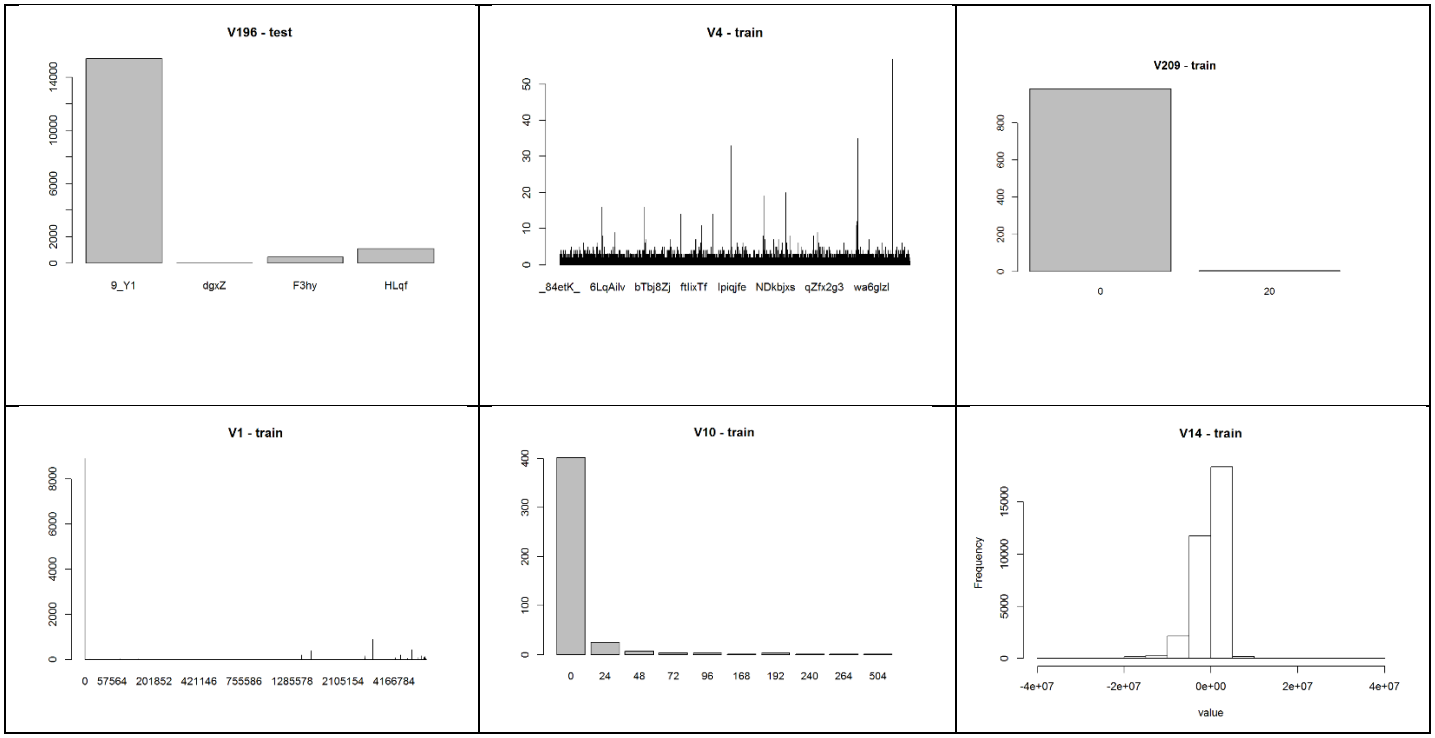


Table 3: Examples of variable types matching descriptions above (left to right, top to bottom)

By comparing plots for training vs test data, we were able to establish how representative the training sample was in terms of factor levels, ranges, and frequencies and distributions of values.

For the most part, ranges and distributions were similar across the training and test datasets. Frequencies were smaller in the test dataset, as this had a smaller number of observations – however these appeared to be in proportion to dataset size. In addition, we plotted the distributions of each variable across those observations where e.g. appetency was +1, vs those where it was -1. This was in the hope of identifying any variables which seemed important for the given classification, for example if all positive instances were in one category and all negative in another. Unsurprisingly, we did not find anything so clear-cut – and in any event, perhaps this kind of analysis is better left to the algorithms. However, it was a useful exercise in "getting to know the data".

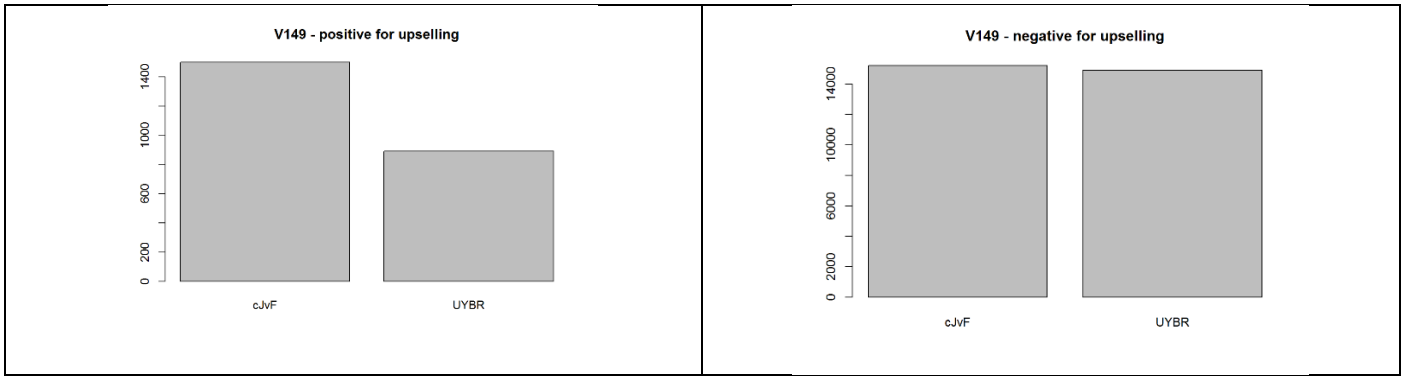


Table 4: Plot of positive vs negative responses for upselling, for V149. The differences in distribution over the levels for this variable suggest that it *may* be useful in predicting upselling.

Another approach in our analysis was to summarise factor levels; how many factors each variable had (or would have, if treated as a factor), how many levels were present in the training set but not the test set (and vice versa), and how many factor levels "tested positive" for churn, upselling and appetency. This information was intended as an aid to dimension reduction strategies, as detailed in a later section on pre-processing. We noted that of those character variables which were unlikely to be factors, nine of them had a large number of values – up to 8498 (out of 11607) – that were in the training set but not the test set, and conversely a proportionally similar number the other way around. This information allowed us to gauge whether a model might overfit to the training data, if we were to treat these variables as factors – and if they were important to the model's fit.

Data Pre-processing

Training/Validation and Test data

Depending on the ML model, one or many of these below steps are applied prior to generating the training/validation and test results. They are performed upfront for two reasons: i) they do not affect the data training/validation and test data distribution, ii) it is time efficient and iii) it provides a clear separation of concerns between the different ML tasks.

Data conversion to Factor

This is implemented in the function *convert_to_factors().* It converts all non-numerical columns to factors. This is necessary as many ML models do not work properly in R on categorical raw data (e.g. LDA, QDA, etc.).

Missing data

Missing data produce prediction bias, it was therefore necessary to clean-up the data prior to running the ML models.

- The *drop_na_cols()* function deletes any columns containing more than 50% of missing data, i.e. where the data is set to NA. This works both for categorical and numerical columns.
- The *convert_NAs_to_level() function ensures any* empty categorical attributes are replaced by a string "NA", which serves as a new level.

Correlated predictors

High multicollinearity increases the variance of the coefficient estimates and makes the estimates very sensitive to minor changes in the model. This results in unstable coefficient estimates, and could cause coefficients to switch signs. Therefore, the function *remove_correlated_predictors()* was built to find attributes that are at least 90% correlated. When found, the attributes are deleted.

Linear dependencies. The function *remove_linear_dependencies()* finds and removes linear dependencies between attributes. The recursive algorithm that takes care of the attribute deletion is natively supported by the function *findLinearCombos().* It uses the QR decomposition of a matrix to enumerate sets of linear combinations (if they exist). For each linear combination, it incrementally removes columns from the matrix and test to see if the dependencies have been resolved.

Category level aggregation. High number of levels in a category (usually greater than 53) generally causes when running ML models. Two types of level aggregation are currently supported.

- The *bin_negative_levels_[labelName]()* (e.g. bin_negative_levels_upselling) coupled with keep_top_10_levels(). The bin_negative_levels_[labelName]() moves all factor levels that *never* map to a +1 response, with reference to a specific label(e.g. appetency) into a bin called ALLNEGATIVE. The keep_top_10_levels() sorts levels by exposure and keep the X most exposed (here the default is X=10). All the other levels are stored in one unique label called BIN.
- The *create_replacement_columns()* function follows in the steps of the *keep_top_10_levels()* function. However, it uses a user defined dictionary that defines a level for a level frequency range. For example, the level named *LEV_251_500* related to category level frequency appearing between 251 and 500 (c.f. Table 2bis). In other words, the user can define the level of granularity of the level range definition.

Category level range	***Label name***
1-10	LEV_1_10
11-25	LEV_11_25
26-50	LEV_26_50
51-100	LEV_51_100
101-150	LEV_101_150
151-250	LEV_151_250
251-500	LEV_251_500
501-750	LEV_501_750
751-1000	LEV_751_1000
1001-3000	LEV_1001_3000
3001-5000	LEV_3001_5000
5001-1000000	LEV_5001_1000000

Table 2bis - Level replacement ranges

Training/Validation data

The pre-processing steps listed in this section only affect the training and validation sets, not the test sets. These transforms are distribution changing, therefore this was the preferred approached to avoid overfitting.

SMOTE. It is a package specialised in automatically rebalancing classes. It is called by the *evaluate_model()* function a part of the cross validation inner loop. The *perc.over* and *perc.under* which respectively drive the number of extra cases from the minority and majority classes to be generated are been set to their default values, i.e. 200. The number of nearest neighbours is set to 5.
The fact that SMOTE affect the validation data could have an effect on the prediction, as it may produce 'over optimistic' AUC results. However, there was no obvious solution on how to solve this problem in a nested cross-validation loop.

Missing Data. The *impute_data()* function *knn-Impute* interpolate numerical data.

Methodology

Test Harness

We wanted to be able to work both as a team, and also in parallel – to evaluate different types of model in individual streams. To support this, we needed a common methodology for evaluating the AUC scores of our models, so that we could be sure (in so far as this was practicable) that we were comparing like with like.

We developed an R script called the "test harness" to enable this. This enforced (a) a consistent framework for training and testing the code (using nested cross-validation), and (b) a consistent mechanism for evaluating AUC – based on the confusion matrix resulting from a model's predictions. It should be noted that there was still a large degree of freedom for individual models to choose e.g. pre-processing steps (and when to apply them), and so we still needed to (re)evaluate our steps when choosing final models. However, the fact that we had the test harness did help us to focus on a common goal, and to throw into relief those things which our different models did differently.

The other strength of the "test harness" came from its ability to support both Caret and non-Caret implementations of models. Our model scripts could pass in a function to replace the default call to Caret's train, and this function would return an object which supported the predict interface – allowing the test harness to be agnostic about any customisation. This was particularly useful in the case of the SVM implementation, as we had issues using Caret which proved difficult to debug (even after lengthy investigation on specialist web sites, attempting to decrypt unhelpful error messages). Using the e1071 library directly worked as expected, and so it was decided to replace the Caret SVM implementation with the e1071 one.

A snippet of the "evaluate model" function is shown in the table below. Note that when we call this "nested cross-validation", there is an assumption that inside the inner loop – where the training folds are passed to a model for fitting – then that model will itself use cross-validation. However, this need not be the case; the test harness does not (by design) enforce it, and Caret, for example, may default to bootstrap rather than cross-validation. The workflow is as follows.

- Given a data frame
  - Split the frame into a specified number of folds
  - Hold out one of those folds for validation
  - Pass the other four folds to the training function (Caret by default, but can be overridden)
  - Call predict on the resulting fitted mode, using the validation fold
  - Calculate and record the AUC for that iteration, based on the confusion matrix resulting from the predictions
  - Repeat with the next fold held out for validation
  - Calculate the average AUC when all validation folds have been used
  - Train the model on the entire dataset and return to the caller

```
evaluate_model <- function(
 data, # the dataset
 formula, # the formula to use e.g. as.formula("appetency ~ foo")
 numFolds = 5, # number of folds for the outer loop cross-validation
 trainMethod, # optional callback method to do the training (defaults to caret)
 ... # pass-through arguments to e.g. caret
 ) {

 if (missing(trainMethod)) {
  trainMethod = call_caret_train # default
 }

 # Work out how many rows go in each test fold
 numRows <- NROW(data)
 foldSize <- numRows %/% numFolds
```

```
  # Work out which of e.g. appetency, upselling and churn we're predicting
  targetColumnName <- get_target_column_name(formula)

  accuracy <- 0.
  auc <- 0.

  # Now do our cross validation
  for (i in 1:numFolds) {

    # Determine the start and end row indices for our test fold
    foldStart <- 1 + ((i-1)*foldSize)
    foldEnd = i * foldSize
    fold.rows <- foldStart:foldEnd

    # Split out the training and test folds
    train.data <- data[-fold.rows, ]
    test.data <- data[fold.rows, ]

    # Fit the model using the training folds
    model.fit <- trainMethod(formula, train.data, ...)

    # Make our predictions on the test fold
    model.pred <- predict(model.fit, test.data)

    # Build a confusion matrix of predictions vs truth
    model.table <- table(
      "Prediction" = model.pred,
      "Truth" = test.data[,targetColumnName]
    )

    # Use the confusion matrix to score for AUC this iteration
    # and increment our running total
    metrics <- calculate_metrics(model.table, debug = debug)
    accuracy <- accuracy + metrics$accuracy
    auc <- auc + metrics$auc
  }

  # Work out and return our average AUC
  accuracy <- accuracy / numFolds
  auc <- auc / numFolds

  # Retrain the model on the whole training set, ready for predicting the test data
  model.fit <- trainMethod(formula, data, ...)

  res = list("accuracy" = accuracy, "auc" = auc, "model" = model.fit)
  return (res)
}
```

We scored AUC for our models using the Balanced Accuracy (BAC) formula given on the competition website[1]. For convenience, we converted the results of each model to a confusion matrix (table) of binary responses. From this table, we were able to extract the relevant numbers of true/false negatives/positives, and plug these into the formula from the website[2]. As stated on the website, for binary responses – such as we had in this case (e.g. +1 and -1 predictions) – this gives in effect the same result as the AUC, as calculated using the trapezoid method.

A snippet of the function used to evaluate the AUC score is shown in the table below.

[1] http://www.kdd.org/kdd-cup/view/kdd-cup-2009/Tasks

[2] {(0,1), (tn/(tn+fp), tp/(tp+fn)), (1,0)}. Or, less formally, "the shaded bits" (a rectangle and two triangles)

```
calculate_metrics = function(table) {
 tn <- table[1]
 fp <- table[2]
 fn <- table[3]
 tp <- table[4]
 sens <- tp/(tp+fn)
 if (tp+fn == 0) {
  sens <- 1
 }
 spec <- tn/(tn+fp)
 if (tn+fp == 0) {
  spec <- 1
 }
 auc <- sens * spec
 auc <- auc + ((0.5*(1-sens))*spec)
 auc <- auc + ((0.5*(1-spec))*sens)

 precision <- tp / (tp+fp)
 accuracy <- (tp+tn) / (tp+fp+fn+tn)

 res = list("accuracy" = accuracy, "auc" = auc)
 return (res)
}
```

AUC = [(sensitivity*specificity)] + [0.5 * (1-sensitivity) * specificity] +[0.5 * (1- specificity) * sensitivity]

AUC Formula verification

The AUC from calculate_metrics function was verified by the using the results from a GBM model, a part of which is shown below:-

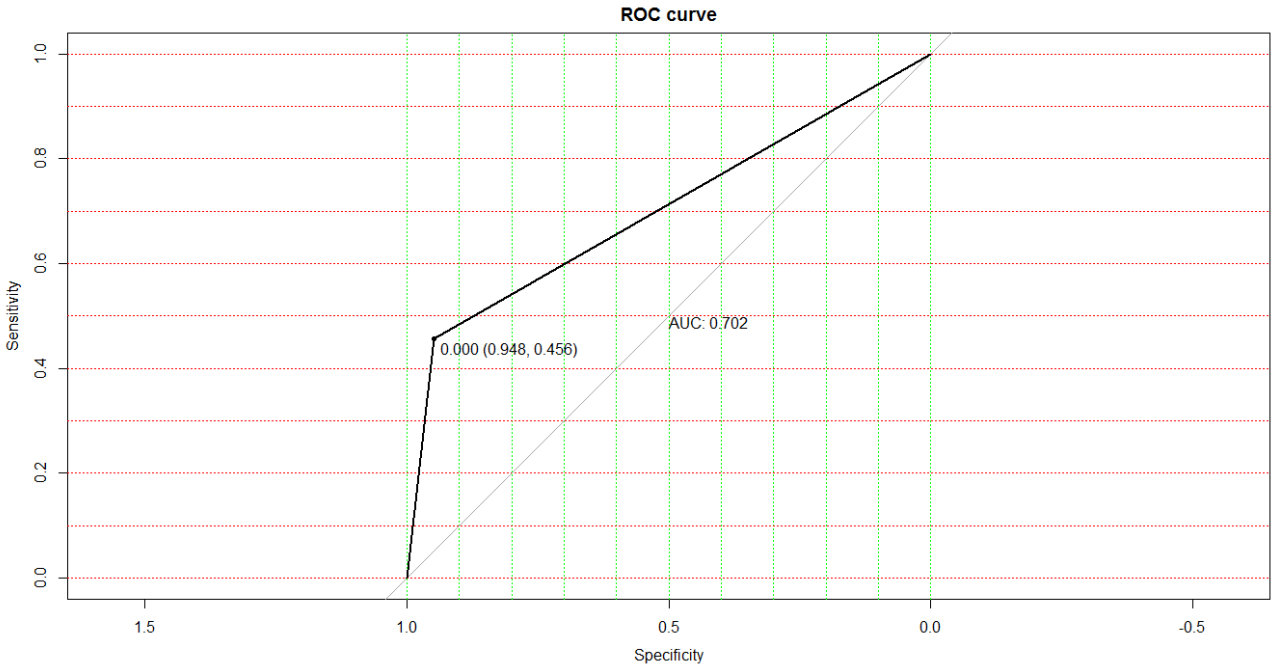
```
set.seed(2017)
gbm.tune <- train(x=model.data.train, y=model.label.train,
                  method = "gbm",
                  metric = "ROC",
                  trControl = ctrl,
                  tuneGrid=grid,
                  verbose=FALSE)


# Note that ROC was the performance criterion used to select the optimal model.

### GBM Model
# Make predictions using the test data set
gbm.pred <- predict(gbm.tune,model.data.eval)
```

The result from the calculate_metrics function is shown below:-

```
> model.table <- table(
+   "Prediction" = gbm.pred,
+   "Truth" = model.label.eval
+ )
>
> calculate_metrics(model.table)
          Truth
Prediction    N    P
         N 7245  330
         P  397  277
[1] "tn: 7245, fp: 397, fn: 330. tp: 277"
[1] "specificity: 0.948050248626014"
[1] "sensitivity: 0.456342668863262"
[1] "precision: 0.410979228486647"
[1] "accuracy: 0.911868105224876"
[1] "F1-score: 0.432474629195941"
[1] "auc: 0.702196458744638"
$accuracy
[1] 0.9118681

$auc
[1] 0.7021965
```

As can be seen below the AUC value matches with the Balanced Accuracy from the confusionMatrix function, the AUC as given by the pROC R package and the AUC from the Trapezoid Method formula as detailed in Appendix D.

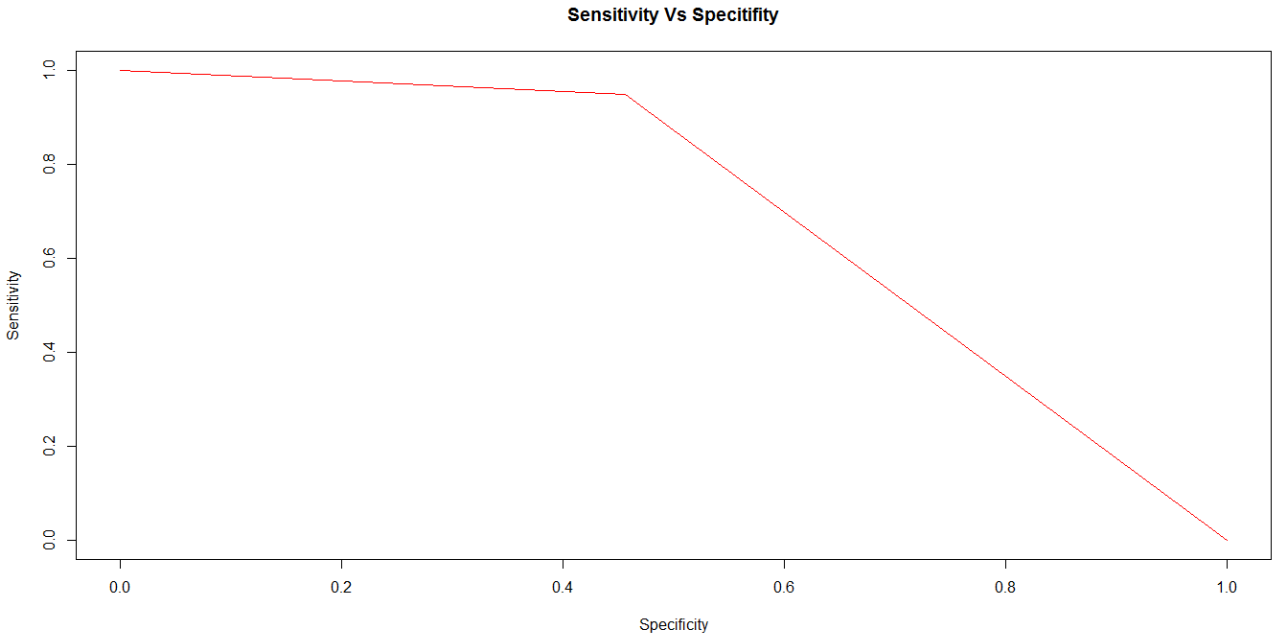
```
> #Look at the confusion matrix
> confusionMatrix(gbm.pred,model.label.eval, positive = "P")
Confusion Matrix and Statistics

          Reference
Prediction    N    P
         N 7245  330
         P  397  277

               Accuracy : 0.9119
                 95% CI : (0.9055, 0.9179)
    No Information Rate : 0.9264
    P-Value [Acc > NIR] : 1.00000

                  Kappa : 0.3848
 Mcnemar's Test P-Value : 0.01437

            Sensitivity : 0.45634
            Specificity : 0.94805
         Pos Pred Value : 0.41098
         Neg Pred Value : 0.95644
             Prevalence : 0.07358
         Detection Rate : 0.03358
   Detection Prevalence : 0.08171
      Balanced Accuracy : 0.70220

       'Positive' Class : P
```




```
> spec=c(0.0, 0.45634, 1.0)
> sens=c(1.0, 0.94805, 0.0)
> plot(spec, sens, col="red",type="l", xlab = "Specificity", ylab = "Sensitivity",
+      main = "Sensitivity Vs Specitifity")
> 0.5*(spec[2]-spec[1])*(sens[2]+sens[1]) + 0.5*(spec[3]-spec[2])*(sens[3]+sens[2])
[1] 0.702195
```

This agrees with the formula used in the calculate_metrics() function of the test harness, that is:

```
[(sensitivity*specificity)] +
[0.5 * (1-sensitivity) * specificity] +
[0.5 * (1- specificity) * sensitivity] =
0.432633137 + 0.257708432 + 0.011853432 = 0.702195.
```


Sensitivity Vs Specitifity
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
Specificity

Model Training/Optimisation and Testing

Linear Discriminant Analysis (LDA) Model

Pre-Processing

Two scenarios were carried out with pre-processing in an attempt to find the best AUC results for all label types. Table 3 shows the pre-processing scenario. Each function is called synchronously in the order provided, for each scenario. The functions behaviour is detailed in the section 'Data Processing' above. The only difference between Scenario1 and Scenario 2 lies in the usages of the categorical binning construction. Scenario 1 keeps the 10 ten levels and bins all the other levels into one single level, named BIN. Scenario 2 bins the levels into a user defined bin range, based on the analysis of level frequency across all categorical classes.

Scenario 1	***Scenario 2***
convert_to_factors drop_na_cols remove_correlated_predictors convert_NAs_to_level remove_linear_dependencies bin_negative_levels_appetency keep_top_10_levels impute_data	convert_to_factors drop_na_cols remove_correlated_predictors convert_NAs_to_level remove_linear_dependencies create_replacement_columns impute_data
The implementations lives in models\lda\test.lda.models.feat.select.appetency.r models\lda\test.lda.models.feat.select.churn.r models\lda\test.lda.models.feat.select.selling.r	The implementations lives in models\lda\test.lda.models.feat.select.replacement.appetency.r models\lda\test.lda.models.feat.select.replacement.churn.r models\lda\test.lda.models.feat.selec t.replacement.selling.r

Table 3 – Pre-processing scenarios

Feature Selection

Although it is assumed that the pre-processing of the initial training data should not impact the overall data distribution, it was necessary to run the feature selections on the two different datasets, generated by each scenario, to ensure a consistent of approach. Due to the data type mix, it was not possible to run the PCA methodology. Therefore, a simple, efficient and easy to interpret feature selection model was selected in its place, namely a Random Forest. The feature selection is performed on the entire training data. The outcome is a graph for each scenario and label showing the attributes in descending order of the 'MeanDecreaseGini' index. The general interpretation is that the higher the decreasing Gini level, the greater is the role played by the predicator in classifying the data. Once a number of attributes with significant prediction power were selected, then a backward selection process was applied starting from the variable with least prediction power. The aim was to keep the minimum number of attributes that maximise the AUC.

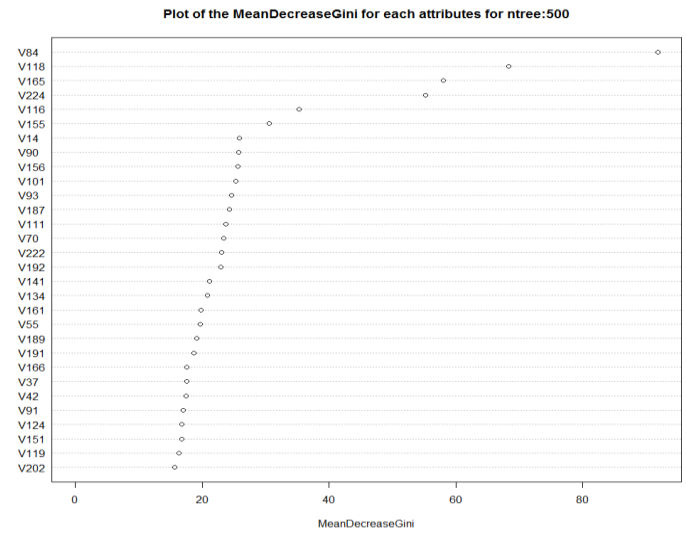
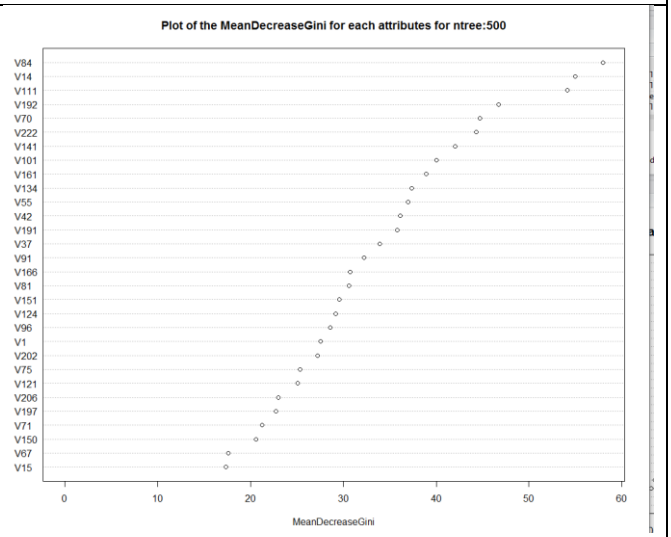
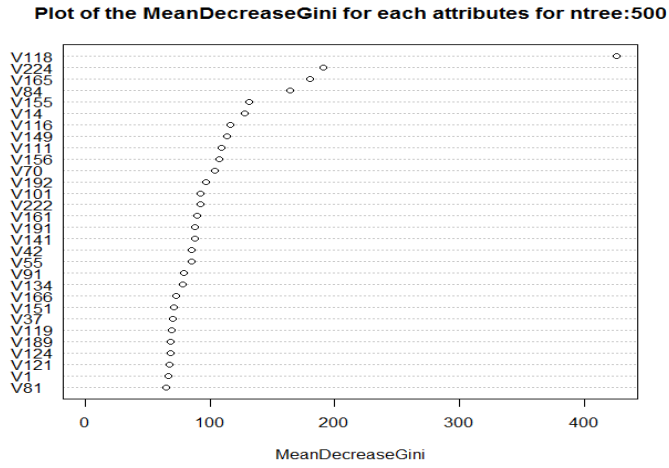
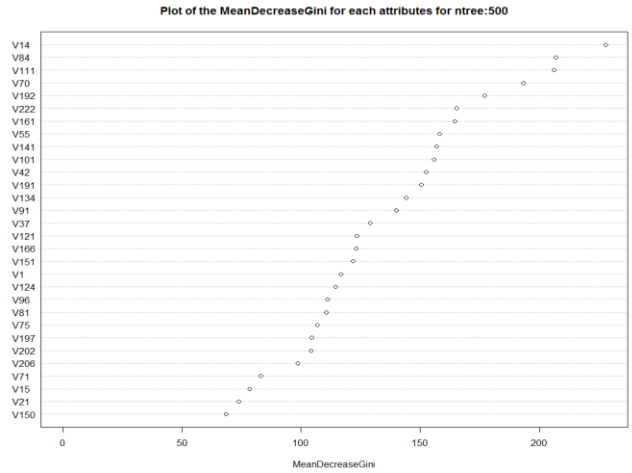
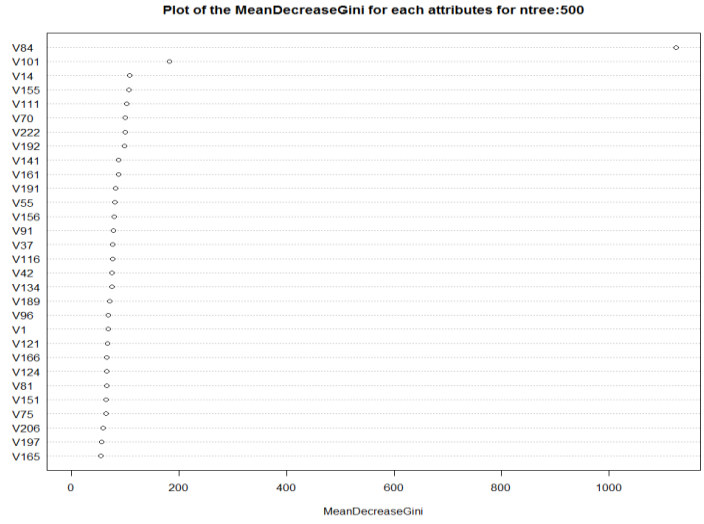
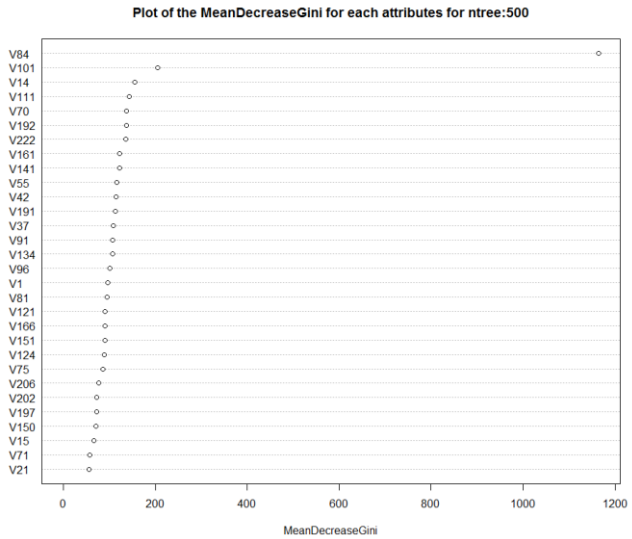
Scenario 1	Scenario 2
 **Appetency**	 **Appetency**
 **Churn**	 **Churn**
 **upselling**	 **Upselling**

Table 4 – Feature selection for scenario 1 and scenario 2

The R implementation is available in the folder *deliverables\feat.select*.

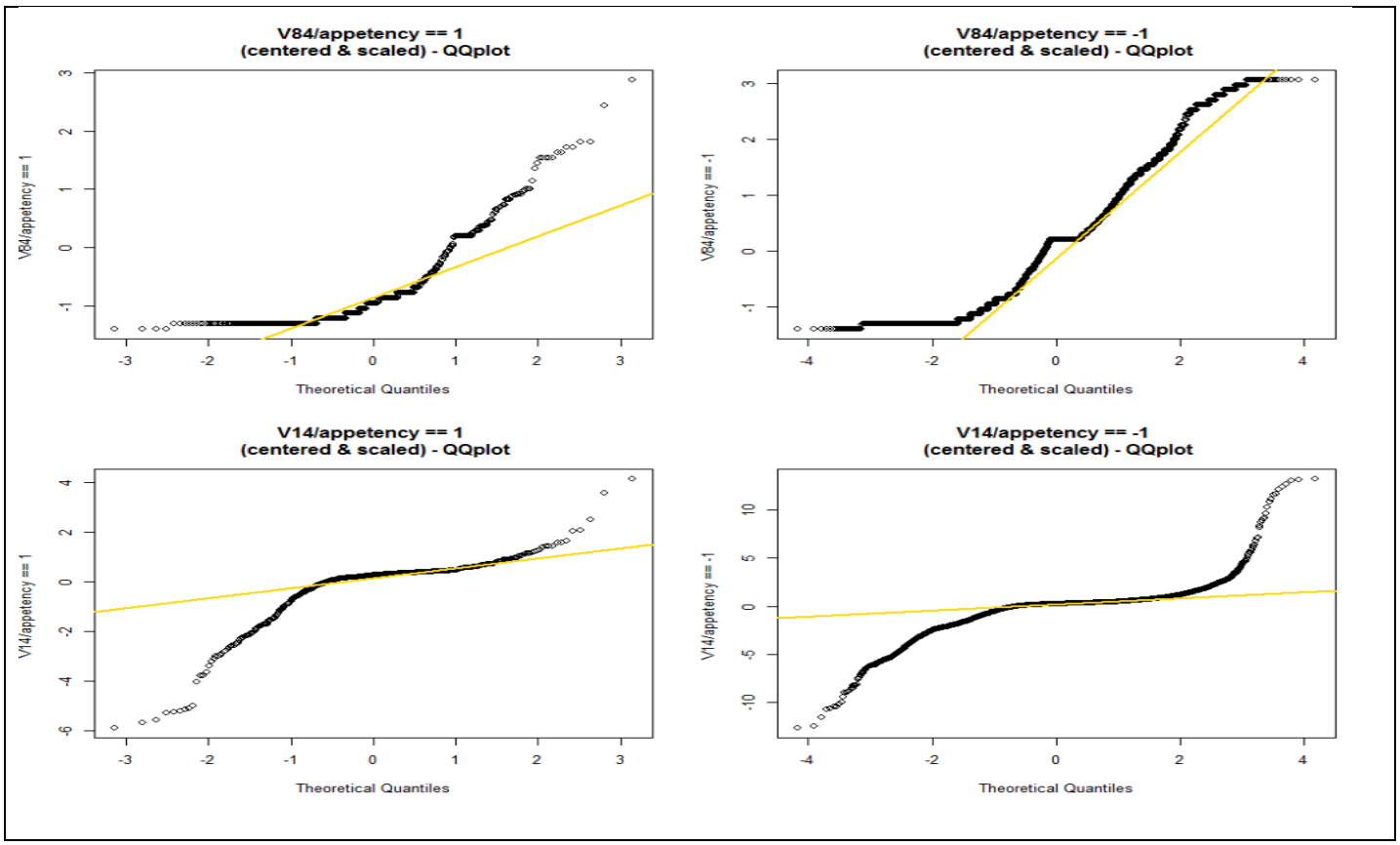
Classifier choice

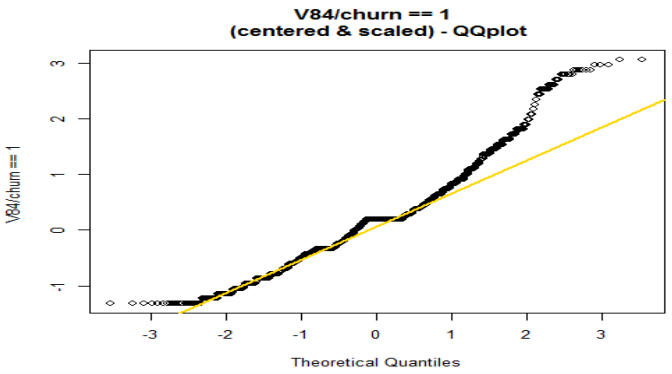
The LDA was chosen for the following reasons: i) it supports both numerical and category data (as long as it is transformed into factors), ii) it is simple (i.e. there no hyper-parameter to optimise) and iii) it is fast to execute.

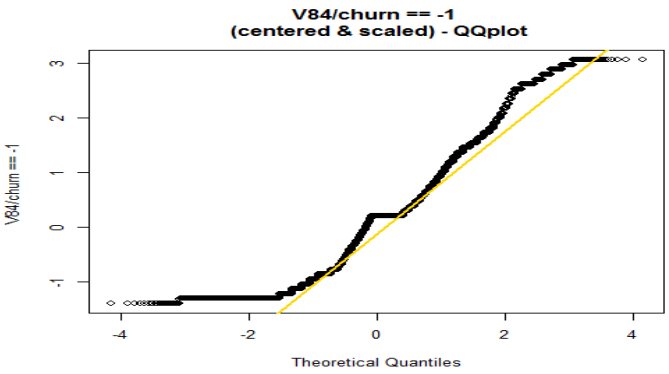
Assumption(s) Verification

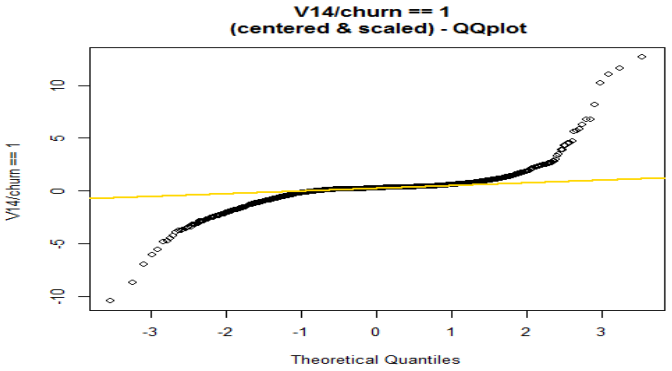
The LDA model requires that the explanatory variables follow a normal distribution. Although the lack of compliance should not prevent the use of these models for prediction purpose, it is interesting to establish whether the model is intrinsically statistically weak. If this is the case, this may have an impact of the model stability and performance accuracy.

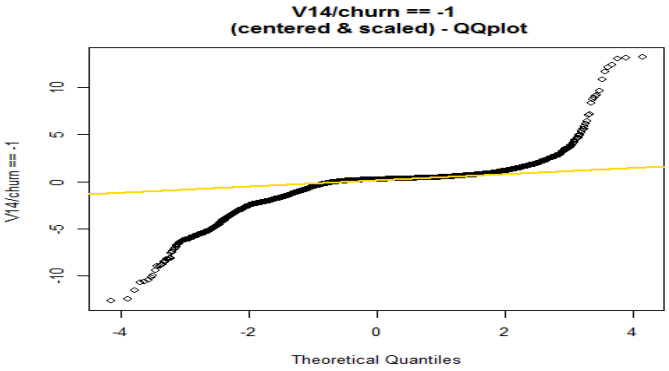
The Kolmogorov Smirnov normality test is run and the Q-plot is drawn for each numerical variable in the selected attributes list, against each label class. For example, the attribute V84 is tested against the *appetency* label for *appetency=1* and *appetency=-1*. The same test is carried out for the *churn* and *upselling* labels. The detailed results are provided in Appendix B. The results from Appendix B and the QQ plots show that none of numerical attributes seem to follow a normal distribution.

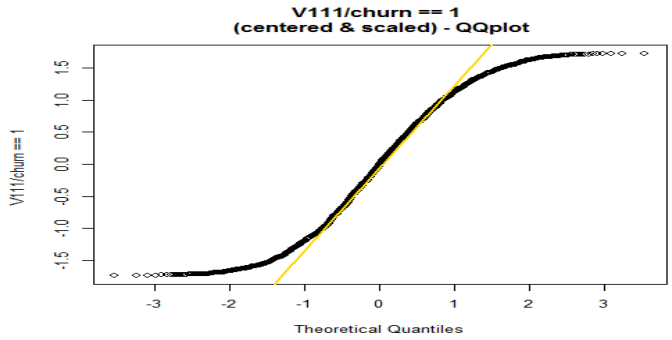


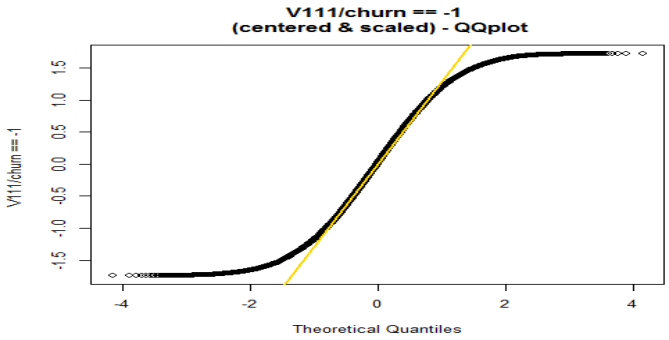

V84/churn == 1
(centered & scaled) - QQplot
V84/churn == 1
Theoretical Quantiles

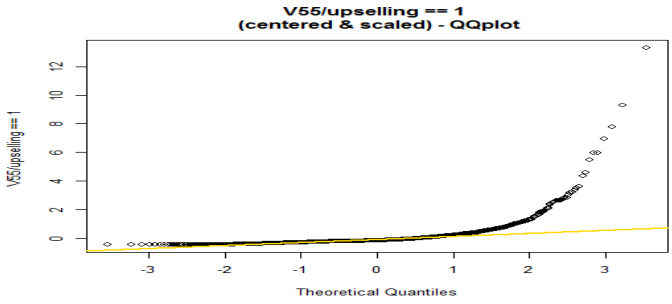

V84/churn == -1
(centered & scaled) - QQplot
V84/churn == -1
Theoretical Quantiles

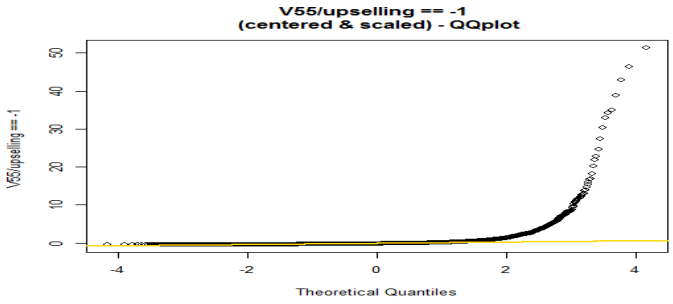

V14/churn == 1
(centered & scaled) - QQplot
V14/churn == 1
Theoretical Quantiles

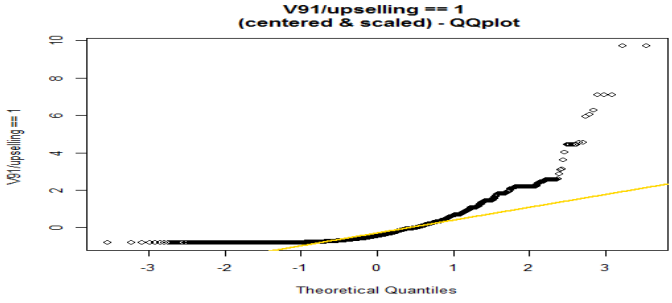

V14/churn == -1
(centered & scaled) - QQplot
V14/churn == -1
Theoretical Quantiles

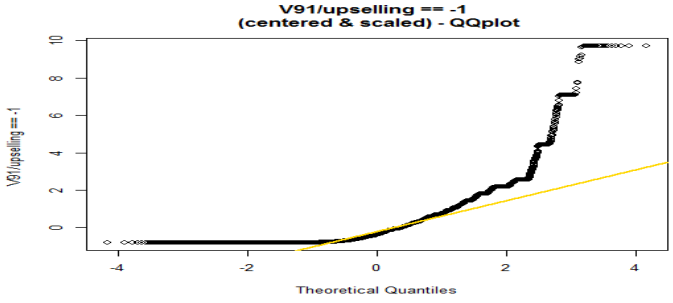

V111/churn == 1
(centered & scaled) - QQplot
V111/churn == 1
Theoretical Quantiles

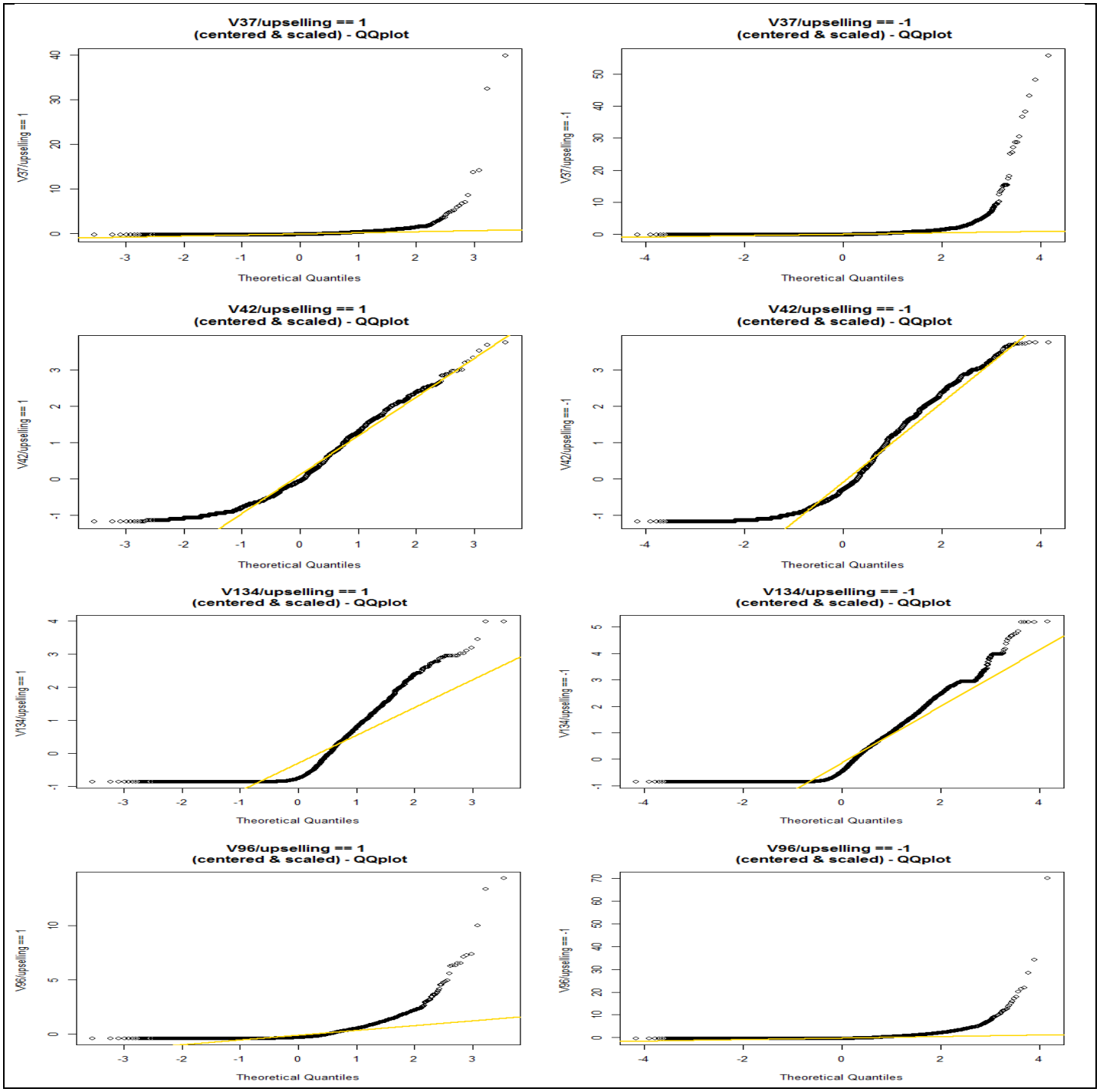

V111/churn == -1
(centered & scaled) - QQplot
V111/churn == -1
Theoretical Quantiles

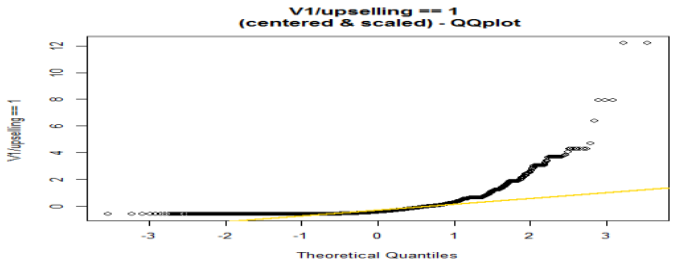

V55/upselling == 1
(centered & scaled) - QQplot
V55/upselling == 1
Theoretical Quantiles

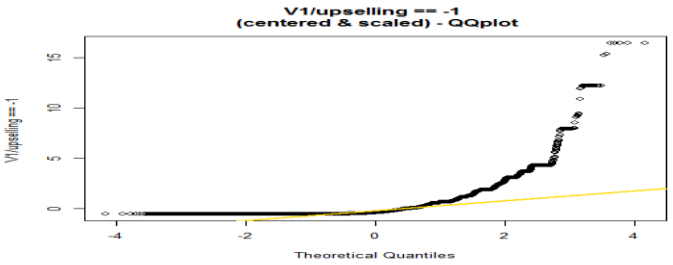

V55/upselling == -1
(centered & scaled) - QQplot
V55/upselling == -1
Theoretical Quantiles

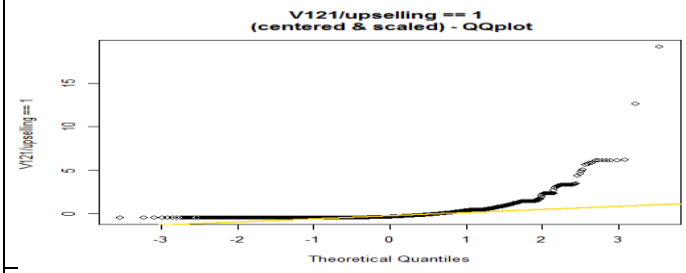

V91/upselling == 1
(centered & scaled) - QQplot
V91/upselling == 1
Theoretical Quantiles

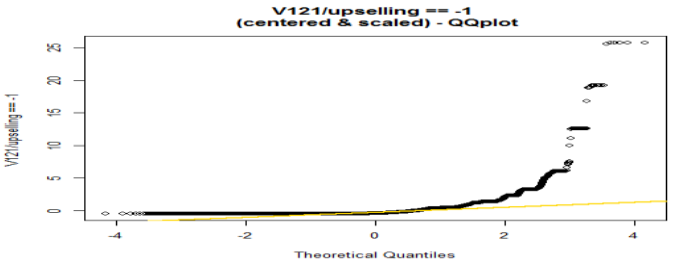

V91/upselling == -1
(centered & scaled) - QQplot
V91/upselling == -1
Theoretical Quantiles

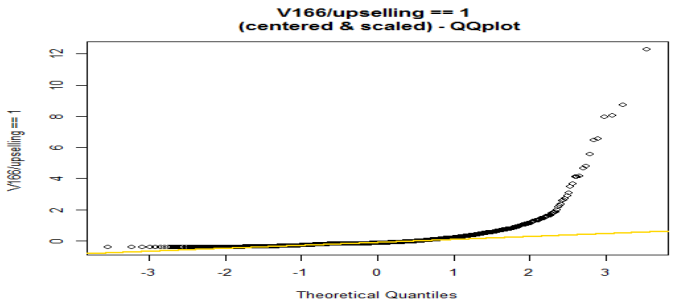

V37/upselling == 1
(centered & scaled) - QQplot
V37/upselling == 1
Theoretical Quantiles
V37/upselling == -1
(centered & scaled) - QQplot
V37/upselling == -1
Theoretical Quantiles
V42/upselling == 1
(centered & scaled) - QQplot
V42/upselling == 1
Theoretical Quantiles
V42/upselling == -1
(centered & scaled) - QQplot
V42/upselling == -1
Theoretical Quantiles
V134/upselling == 1
(centered & scaled) - QQplot
V134/upselling == 1
Theoretical Quantiles
V134/upselling == -1
(centered & scaled) - QQplot
V134/upselling == -1
Theoretical Quantiles
V96/upselling == 1
(centered & scaled) - QQplot
V96/upselling == 1
Theoretical Quantiles
V96/upselling == -1
(centered & scaled) - QQplot
V96/upselling == -1
Theoretical Quantiles

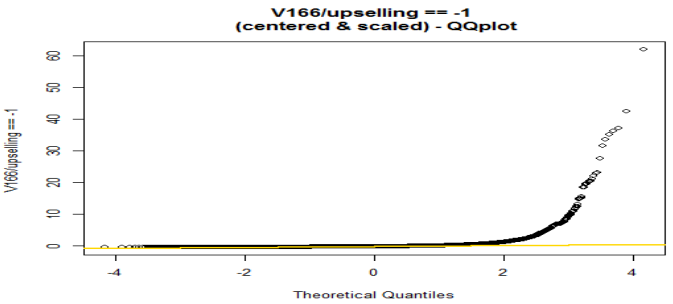

V1/upselling == 1
(centered & scaled) - QQplot
V1/upselling == 1
Theoretical Quantiles

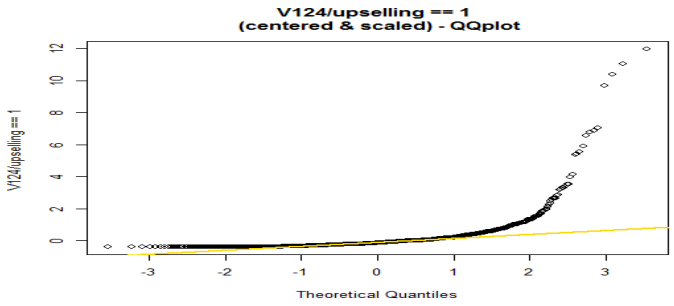

V1/upselling == -1
(centered & scaled) - QQplot
V1/upselling == -1
Theoretical Quantiles

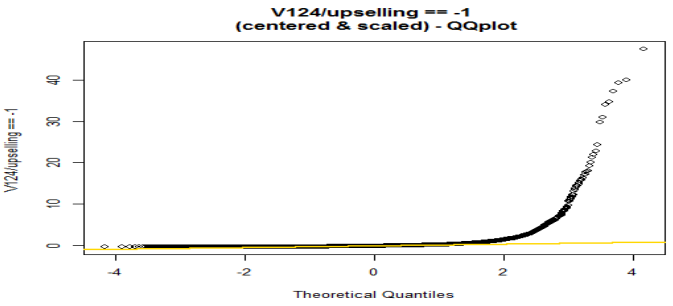

V121/upselling == 1
(centered & scaled) - QQplot
V121/upselling == 1
Theoretical Quantiles

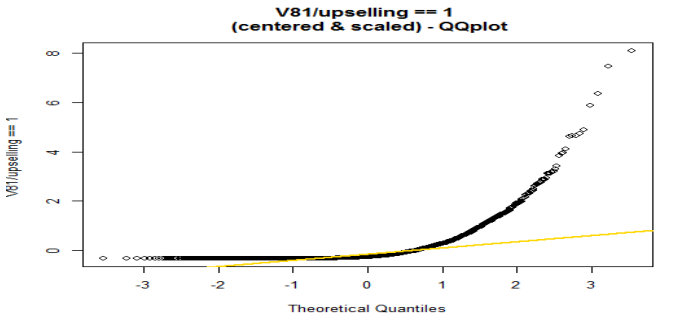

V121/upselling == -1
(centered & scaled) - QQplot
V121/upselling == -1
Theoretical Quantiles

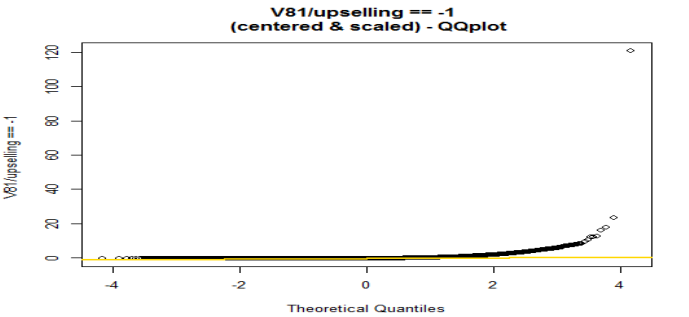

V166/upselling == 1
(centered & scaled) - QQplot
V166/upselling == 1
Theoretical Quantiles

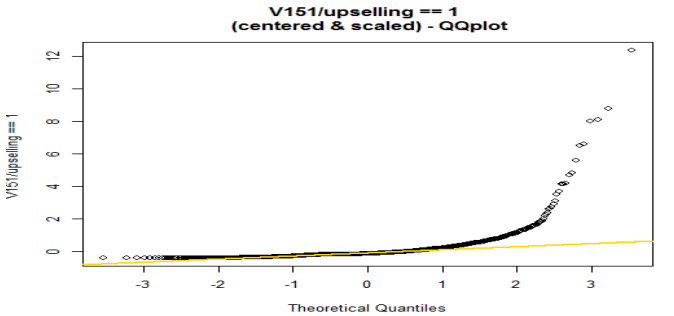

V166/upselling == -1
(centered & scaled) - QQplot
V166/upselling == -1
Theoretical Quantiles

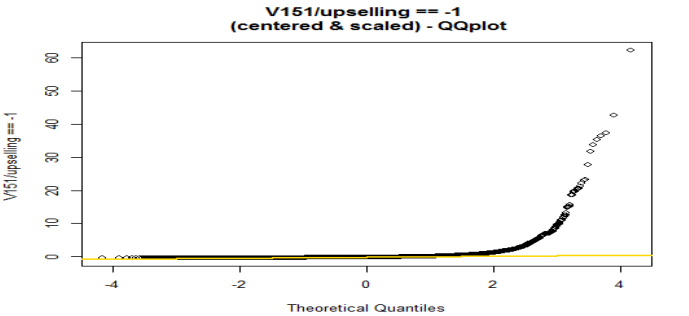

V124/upselling == 1
(centered & scaled) - QQplot
V124/upselling == 1
Theoretical Quantiles


V124/upselling == -1
(centered & scaled) - QQplot
V124/upselling == -1
Theoretical Quantiles


V81/upselling == 1
(centered & scaled) - QQplot
V81/upselling == 1
Theoretical Quantiles


V81/upselling == -1
(centered & scaled) - QQplot
V81/upselling == -1
Theoretical Quantiles


V151/upselling == 1
(centered & scaled) - QQplot
V151/upselling == 1
Theoretical Quantiles


V151/upselling == -1
(centered & scaled) - QQplot
V151/upselling == -1
Theoretical Quantiles

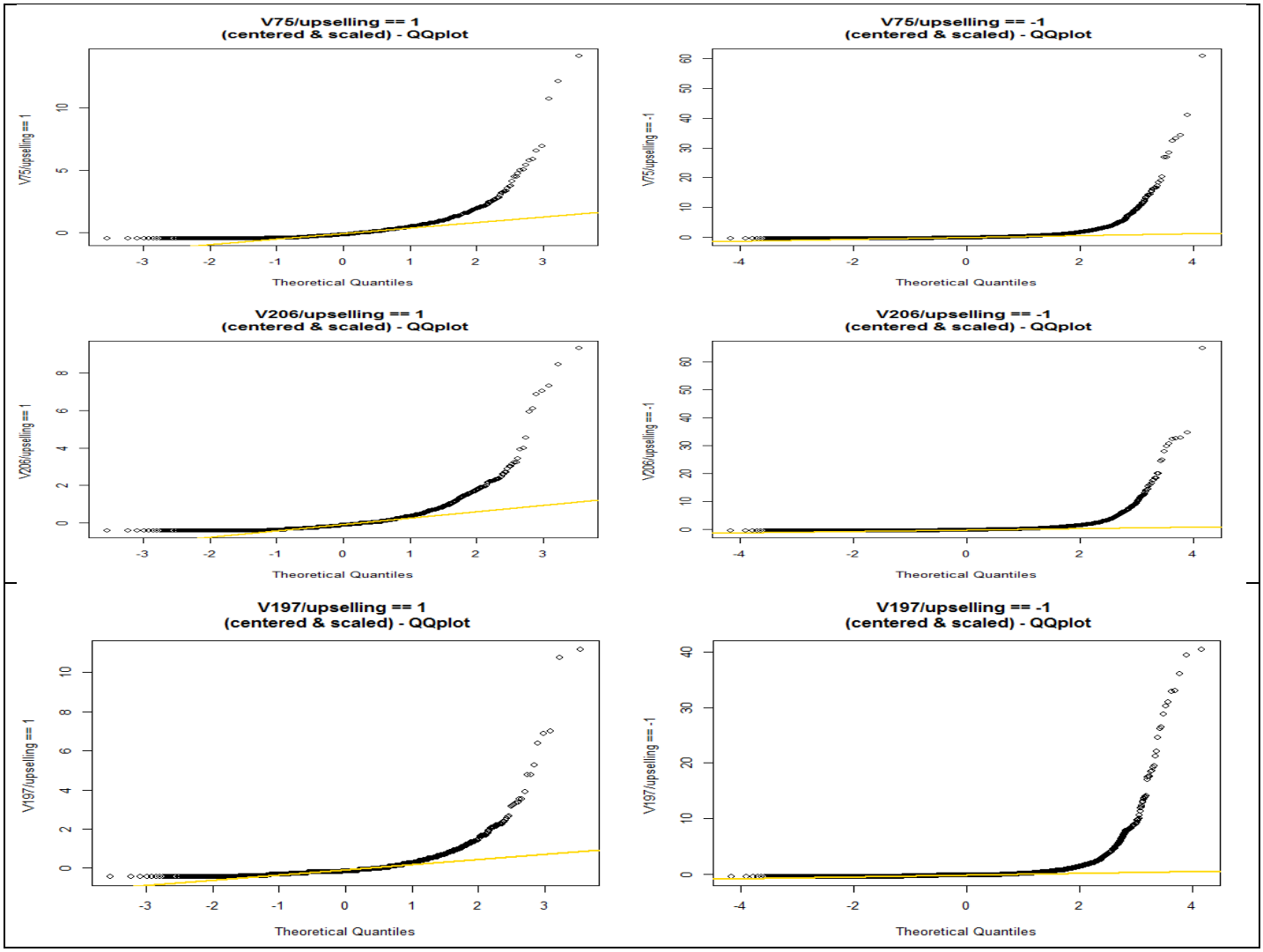


Figure 1 – Normal distribution test of numerical attributes for *appetency*, *churn* and *upselling*

The results

The results listed in table 5 correspond to the average AUC (trapezoid method) on test data using a nested cross-validation framework.

Scenario 1	***Scenario 2***
Appetency: 91.26% Churn: 71.73% Upselling: 64.37%	Appetency: 71.54% Churn: 52.39% Upselling: 66.63%
Time to completion < 30mins	Time to completion approx. 1h
The results are available in models.res\ouput.test.lda.model.feat.select.appetency. txt models.res\ouput.test.lda.model.feat.select.churn.txt models.res\ouput.test.lda.model.feat.select.upselling. txt	The results are available in ouput.test.lda.model.feat.select.replacement.appetency. txt ouput.test.lda.model.feat.select.replacement.churn. txt ouput.test.lda.model.feat.select.replacement.upselling. txt

Table 5 – LDA test results

Support Vector Machine (SVM) Model

Pre-Processing

The outcome of the LDA test results shows that the Scenario 1 pre-processing provides the best test results overall. Therefore, only this scenario is retained for the SVM test. Furthermore, there is an extra step provided by default by the *tune()* function, where all numerical data is cantered and scaled to normalise data prior to running the algorithm.

Feature Selection

The feature selection is based on scenario 1 pre-processed data (c.f. table 3). The results are provided in Table 4 Scenario 1.

Classifier choice

The SVM was chosen as this is a frequently used model in the machine learning research literature for supervised classification tasks. Furthermore, it is compatible with numerical and categorical attributes. However, it is known to be slow.

Assumption(s) Verification

There is no specific assumption made on the shape of the data.

The results

The SVM model was trained/optimised and tested under two regimes, Regime 1 and Regime 2, as described in Table 6:

Regime 1	***Regime 2***
Kernel = linear Cost: (0.0001,0.0005, 0.001, 0.1, 1 ,5 ,10 ,100) Gamma: not supported in the linear case	Kernel = radial Cost: (0.0001,0.0005, 0.001, 0.1, 1 ,5 ,10 ,100) Gamma: 0.0001,0.0005, 0.001, 0.1, 0.5, 0.7, 1)

The implementation of the linear and radial SVM models are available in:

- model\svm\linear\ test.svm.linear.model.feat.select.appetency.r (churn.r and upselling.r)
- model\svm\radial\ test.svm.radial.model.feat.select.appetency.r (churn.r and upselling.r)

The results listed in table 7 correspond to the average AUC (trapezoid method) on test data using a nested cross-validation framework.

Regime 1	***Regime 2***
Appetency: 50.0% Churn: 62.5% Upselling: 37.5%	Appetency: 50.0% Churn: 62.5% Upselling: 62.5%
Time to completion approx. 2 days	Time to completion approx. 2 days
The results are available in models.res\output.test.svm.linear.model.feat.select.upselling.txt models.res\output.test.svm.linear.model.feat.select.churn.txt models.res\output.test.svm.linear.model.feat.select.upselling.txt	The results are available in models.res\output.test.svm.radial.model.feat.select.upselling.txt models.res\output.test.svm.radial.model.feat.select.churn.txt models.res\output.test.svm.radial.model.feat.select.upselling.txt

Table 7 – SVM test results

"NA Groups" Model

Motivation

As noted in the Data Analysis & Visualisation section, earlier, it seemed to be the case that a number of variables could be grouped together by count of missing values.

For example, 27 variables had 32160 missing values, 22 had 32013, and 6 had 3640. In fact, grouping all variables by "NA count" revealed that there were potentially 23 groupings where more than one variable had the same number of missing values. This seems unlikely by chance; randomly missing values across the dataset would not be expected to align quite so precisely – although there is no guarantee that these patterns are meaningful, in terms of predictive modelling. For example, an alternative hypothesis could be that over time, Orange modified the list of attributes being collected – and so the pattern of missing values in the data represent changes to their database schema over the lifespan of collection. Nonetheless, this discovery presented an interesting opportunity for a custom model which aimed to make use of these "tables within a table"[3].

Procedure

The basic idea behind the NA Groups Model, then, was to fit a model to each group of variables with the same number of missing values, and then somehow combine the resulting predictions from each of these into an overall classification. As an example, consider the table below as an extract from a larger dataset:

A	B	C	D	E	F	G	H
		4			0		0
284076			1NjhLsz			taul	
4595634						CuXi4je	
403884			Pzu_i9p			taul	
smXZ	112			16		smXZ	
	1088			144			
		4			0		0

Let's suppose that columns A and G have the exact same number of missing values, and similarly B and E, and C, F and H. Now if we do standard imputation on missing values, then we will lose information if, in fact, these patterns of NAs are meaningful[4]. So instead, we can omit missing values from each column and group them together into three, complete, data frames:

A	G
284076	taul
4595634	CuXi4je
403884	taul
smXZ	smXZ

B	E
112	16
1088	144

C	F	H
4	0	0
4	0	0

Treating these frames as self-contained pockets of information, we can fit an individual model to each group. When it comes to making predictions, then we need to go row by row and determine which of these models can be used – based on which values are missing in each row in turn. So in our table above, for example, only model #3 (variables

[3] We were encouraged to find that the winners of the original competition, IBM Research, had also made note of this in their subsequent paper (https://researcher.ibm.com/researcher/files/us-kclang/KDDCup-jmlr09.pdf). Less so that they opted to address it with a "fast probabilistic bi-clustering algorithm" (p.29), since we didn't know what one of those was.

[4] Colloquially, we risk overlooking the fact that the data might be a mixture of apples and oranges, so that imputation would result in a generic fruit punch. Or more technically, overlooking the fact that the data might have come from different database tables, and just been munged together.

C, F and H) applies to row 1, but for row 5 we can invoke model #1 on columns A and G, and model #2 on columns B and E.
The trick then becomes how to turn this "ensemble" of predictors into a single overall prediction. A number of different strategies were attempted:

- If any one of the nested models votes +1, then the overall model predicts +1
- As above, but with a threshold of e.g. 20% positive votes to result in a +1 overall
- As above, but with an additional threshold for a model to be included in the ensemble, based on its AUC score on the training data
- A model which treats each nested predictor as a variable in a second, linear model – and then uses least squares to find the coefficients with which to weight their subsequent predictions.

Using manual thresholds produced decent results, however it was felt that this was likely to result in overfitting (since the process of manually tweaking these to get higher AUC scores was in itself a way of training the models to fit the validation folds). Eventually we arrived at a variant of the final strategy, which was simplified but actually more flexible and powerful; rather than manually extracting the weights from the uber-predictor, and using these during classification, we could just use the predictions of the uber-predictor.

In order to facilitate this, a level of indirection was inserted into the process. During the training stage, a matrix was created of the predictions of each of the nested models vs the target label – which was used to train the uber-model. This process was repeated against the validation data, and the stored uber-model used to predict the target of the matrix which resulted from the individual votes of the nested predictors.

model1	model2	model3	model4	…	model23	appetency
-1	0	+1	-1		-1	-1
+1	+1	0	-1		+1	+1
-1	+1	0	0		-1	+1

Table 5: Example matrix generated from predictions of individual models, vs the target label. Zeros in the matrix represent a predictor which cannot be used due to missing values in a row's data.

Classifier Choice

We decided to use Random Forest for all of the nested predictors. Tree based methods seem like an obvious choice given the variety and disparity of the data/types, and moreover decision trees are at least superficially more aligned with a customer's thought processes and behaviours than a parametric or mathematical model[5].

A number of models were tried for the uber-predictor. The first was lda, which gave good results – but an inspection of the coefficients showed that weightings were loading almost exclusively on model #1, which was generated from those 27 columns which had zero missing values in the overall training set. Following this discovery, a switch was made to rda – in the hope that the regularisation in this model would distribute weights more evenly over the predictors. Others were tried, including adaboost and variants of SVM, with mixed results – often dependent on which of appetency, churn and upselling was being predicted.

Feature Selection

Feature selection was not used in this model, as everything was driven by the groupings of columns, and this was automatically generated. In a sense, feature selection was built in – since any groupings of variables which did not usefully contribute to correct predictions would be down-weighted by the uber-model. There is no reason why future implementations could not drill down into these groupings, and analyse the importance of variables within, rather than just between, their models. However, this was out of scope for the assignment.

[5] That, plus the subsequent discussion from the competition organisers revealed that these were probably the best bet - http://proceedings.mlr.press/v7/guyon09/guyon09.pdf.

Pre-Processing

A number of columns did not group together with any others – that is, they had their own numbers of missing variables. As such, these were treated as randomly missing, and had missing values imputed. These columns were either character factors – in which case an explicit "NA" level was used for missing values – or else discrete, numeric, values. Using the plots of the distributions for these variables, from earlier analysis, it was decided to use the modal values in each case (most often zero).

None of the discrete, numeric, variables were converted into factors. Initially, this was because doing so resulted in a dramatic increase in processing speed, however this may have been a happy accident – since comparing the plots of these variables in the training vs the test set (proper) showed that although the test set has similar distributions, the values are often not identical (different multiples of the same common denominator, for example), and so treating these as factor levels would have been a mistake.

The absence of feature selection did mean that the model made full use of all character "factors", including the ones with thousands of levels. In order to make things computationally tractable, the number of levels in these variables had to be reduced – and therefore a number of binning strategies were applied:

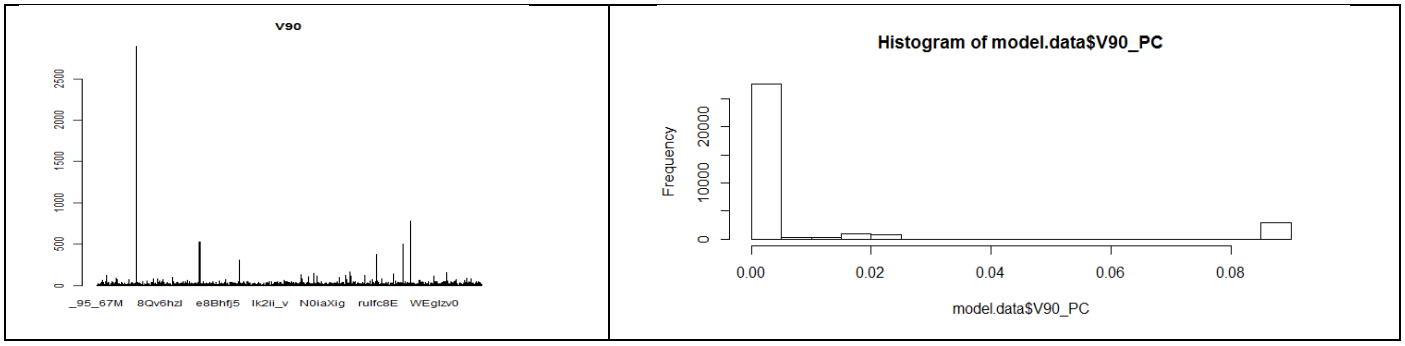
- Any levels which were not present in the (proper) test set were aggregated into a "BIN" level.

- Any levels which could never result in a +1 prediction (wrt a specified target label) were binned into an "ALL_NEGATIVE" level. This was done up-front of the nested cross-validation, on the basis that it was a re-coding of existing data rather than a derivation of anything new. In effect, it was an attempt at pro-active pruning – in the sense that any time the decision tree branched down one of these levels then it would always end up at a leaf predicting -1. This could of course be a faulty assumption, although if anything we should expect the true AUC score to be lower, as a result – since some of these levels may, in the test data, map to positive responses. But since there is no way that our models would even predict these, if all responses during training were negative, it was felt that it was worth it.

- For factors which still had more than 30 levels, the top 30 levels (by frequency) were kept, and the remainder were aggregated into a "BIN" level. (The ALL_NEGATIVE levels could have gone in here also, however we wanted to make sure that these would always map to a response of -1, which would not be the case if mixed in with BIN).

- Feature engineering: prior to the above step, we added a "numeric equivalent" of character variables, since in a lot of cases these were more likely to be IDs than factors (e.g. had way too many unique levels). We wanted to try and retain some information from these before binning them – in effect, to capture the "spikes" in the distributions in a more computationally efficient format. A number of strategies were tried, for example adding a value representing the frequency of a level as a percentage of the total number. The numeric equivalent values had to be relative (e.g. proportional) rather than absolute, as the number of rows in the test set is roughly half the number of rows in the training set – and so a simple count would be unlikely to work. Of course, for this approach to have any benefit, there would need to be some relationship between the frequency of occurrence of a factor level and the target response[6].

[6] An alternative might be to try and come up with a numeric equivalent of each individual factor level, for example the hash value of its label. Assuming that values would be mostly distinct, then this could keep the unique-ness of each factor level whilst allowing the tree to treat it as a number. Although whether the tree would be able to find useful splits along this new dimension is anyone's guess, since we didn't try it.



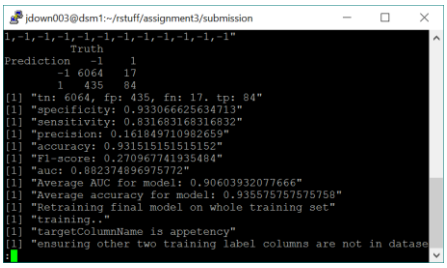
SMOTE was applied to the training folds at each iteration of the nested cross-validation, and also Near Zero Variance columns were removed. Data was centred and scaled.

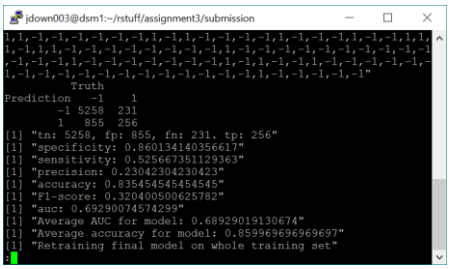
Assumption(s) Verification

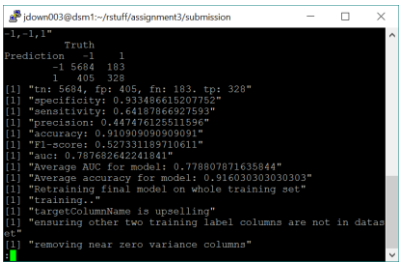
There were no specific assumptions made regarding the shape of the data, since this was a non-parametric model.

Results

Appetency: 91%
Churn: 69%
Upselling: 78%







Time to completion: This model took about 8 hours for all of them (running on dsm9) to complete. Appetency took only 30mins, however churn and upselling took the majority of the time.

Methodology Limitations

It's a little hard to know how to evaluate this model, on account of it being largely experimental. Results were encouraging; however it is difficult to know whether this was an indication of overfitting since the complexity of the model made it hard to analyse. Also, it was not especially quick to train – both as a result of having twenty-something random forests to fit for every iteration of the outer cross-validation loop, and also because predictions had to be made on a row by row basis, since the combinations of models to apply varied with the patterns of missing values in the data. On the plus side, we did not have to (by and large) impute any values for this model, since by virtue of the design each nested data frame did not have any.

"Basic RF" Model

Motivation

This model arose from an observation made during the running of "NA Groups" model, which was that when weighting each nested model in the ensemble, most of the weights appeared to be loading on the first model. This model contained those variables which had either zero missing values to begin with (27 variables), or else appeared to have a random number of missing values which were easy to impute (15 variables; a mixture of character factors and discrete numeric variables). Although attempts were made to try and regularise the weightings in the NA Groups model, as an aside we also tried a simple random forest model using just these variables as predictors.

The model used the following predictors:
V11, V14, V22, V27, V29, V42, V50, V73, V90, V111, V112, V116, V119, V128, V138, V143, V154, V155, V165, V168, V182, V190, V204, V212, V224.

Feature Selection

Feature selection was not used in this model, over and above selecting only those variables which did not appear to have a "structural" pattern to the number of missing values.

Pre-Processing

The same pre-processing steps were taken as described in the NA Groups model. The following character factors had explicit "NA" levels added: V22, V27, V50, V138, V143, V165, V182. The following discrete numeric variables had their modal values imputed: V36, V61, V84, V101, V120, V176, V183, V185. The modal values were all zero, except for V36 (=564) and V84 (=8).

Assumption(s) Verification

There were no specific assumptions made regarding the shape of the data, since this was a non-parametric model.

Results

Appetency: 92%
Churn: 65%
Upselling: 71%

```
Appetency

          Truth
Prediction   -1    1
        -1 5921    6
         1   578   95
[1] "tn: 5921, fp: 578, fn: 6. tp: 95"
[1] "specificity: 0.911063240498538"
[1] "sensitivity: 0.940594059405941"
[1] "precision: 0.141158989598811"
[1] "accuracy: 0.911515151515152"
[1] "F1-score: 0.245478036175711"
[1] "auc: 0.925828649952239"
[1] "Average AUC for model: 0.919429904884908"
[1] "Average accuracy for model: 0.918212121212121"
```

```
Upselling

          Truth
Prediction   -1    1
        -1 5402  246
         1   687  265
[1] "tn: 5402, fp: 687, fn: 246. tp: 265"
[1] "specificity: 0.887173591722779"
[1] "sensitivity: 0.518590998043053"
[1] "precision: 0.278361344537815"
[1] "accuracy: 0.858636363636364"
[1] "F1-score: 0.362269309637731"
[1] "auc: 0.702882294882916"
[1] "Average AUC for model: 0.711883741364663"
[1] "Average accuracy for model: 0.856848484848485"
```

```
Churn

          Truth
Prediction   -1    1
        -1 5589  310
         1   524  177
[1] "tn: 5589, fp: 524, fn: 310. tp: 177"
[1] "specificity: 0.914281040405693"
[1] "sensitivity: 0.363449691991786"
[1] "precision: 0.252496433666191"
[1] "accuracy: 0.873636363636364"
[1] "F1-score: 0.297979797979798"
[1] "auc: 0.63886536619874"
[1] "Average AUC for model: 0.647449356648997"
[1] "Average accuracy for model: 0.871484848484849"
```

Time to completion: This model took about 1 hour for all models to complete.

Evaluation

It's unclear why this model should score so well on appetency, and so (relatively) poorly on the other two training labels – especially given that upselling, for example, was (in the original competition) found to be the easiest of three to predict, and moreover predict using tree-based models. Possibly this is a warning that the model has overfit – although the use of the `bin_levels_if_not_in_test_set` function, at least, should have helped reduce the risk of this with the large character factors. (Without this pre-processing step, the AUC score for appetency was 96%, which seems implausible). Another benefit of this model – given that it did not have to deal with missing values – was that there was no need for imputation. This, coupled with a relatively small number of (hopefully) non-invasive pre-processing steps, suggests that the model should perform reasonably well on the test set proper.

"GBM" Model

Pre-Processing

As has been mentioned in the "Data Analysis & Visualisation" section, the discrete numerical variables were multiples of a number and missing values were not random e.g. when a value was missing for V1, there was a value for V2 and vice versa. The missing values were therefore filled in with the string "NA". The actual values were "binned" depending on what percentile they were in.

In the case of continuous variables, any columns which had more than 40% missing values were discarded. The remaining missing values were filled in (impute_data) prior to running the caret train function. The impute was done on the training data and then on the validation data.

Results:

Appetency: 58% AUC

Churn: 51% AUC

Upselling: 71.41% AUC

Appetency results

1] "Average AUC for model: 0.580495769105111"
[1] "Average accuracy for model: 0.92030303030303"
[1] "Retraining final model on whole training set"

Iter	TrainDeviance	ValidDeviance	StepSize	Improve
1	1.3584	nan	0.0100	0.0030
2	1.3509	nan	0.0100	0.0034
3	1.3432	nan	0.0100	0.0033
4	1.3362	nan	0.0100	0.0030
5	1.3289	nan	0.0100	0.0031
6	1.3222	nan	0.0100	0.0028
7	1.3160	nan	0.0100	0.0026
8	1.3089	nan	0.0100	0.0029
9	1.3024	nan	0.0100	0.0028
10	1.2960	nan	0.0100	0.0026
20	1.2327	nan	0.0100	0.0026
40	1.1258	nan	0.0100	0.0016
60	1.0373	nan	0.0100	0.0015
80	0.9637	nan	0.0100	0.0011
100	0.9014	nan	0.0100	0.0013

Churn results

[1] "tn: 5865, fp: 248, fn: 441. tp: 46"
[1] "specificity: 0.959430721413381"
[1] "sensitivity: 0.0944558521560575"
[1] "precision: 0.156462585034014"
[1] "accuracy: 0.895606060606061"
[1] "F1-score: 0.117797695262484"
[1] "auc: 0.526943286784719"
[1] "Average AUC for model: 0.515164919661482"
[1] "Average accuracy for model: 0.891818181818182"
[1] "Retraining final model on whole training set"

Iter	TrainDeviance	ValidDeviance	StepSize	Improve
1	1.3587	nan	0.0100	0.0034
2	1.3523	nan	0.0100	0.0031
3	1.3455	nan	0.0100	0.0033
4	1.3398	nan	0.0100	0.0027
5	1.3335	nan	0.0100	0.0031
6	1.3275	nan	0.0100	0.0029
7	1.3216	nan	0.0100	0.0028
8	1.3153	nan	0.0100	0.0031
9	1.3099	nan	0.0100	0.0026
10	1.3038	nan	0.0100	0.0029
20	1.2490	nan	0.0100	0.0027
40	1.1631	nan	0.0100	0.0023
60	1.0984	nan	0.0100	0.0011
80	1.0510	nan	0.0100	0.0008
100	1.0109	nan	0.0100	0.0011

Upselling results

```
> model.table
          Truth
Prediction    N    P
         N 7200  312
         P  442  295
>
> varImp(gbm.tune)
gbm variable importance

  only 20 most important variables shown (out of 200)

     Overall
V84  100.000
V101  75.413
V90   28.511
V168  19.645
V119  17.683
V40   15.461
V138  13.045
V14   11.632
V194   8.788
V221   7.198
V104   6.882
V163   3.911
V193   3.306
V86    2.694
V94    2.548
V153   2.314
V157   2.195
V3     1.992
V41    1.890
V171   1.874
> calculate_metrics(model.table)
$accuracy
[1] 0.908595

$auc
[1] 0.7140792
```

Time to completion approx. 30mins	Time to completion approx. 1h	Time to completion approx. 1h
Code files available in models\gbm\ gbm.appetency.model.r	Code files available in models\gbm\ gbm.churn.model.r	Code files available in models\gbm\ gbm.upselling.model.r

Evaluation

This section provides a summary of the results and details the choice for the final models.

Model Name	Appetency (Avg AUC)	Churn (Avg AUC)	Upselling (Avg AUC)
LDA Scenario 1	91.26%	71.73%	64.37%
LDA Scenario 2	71.54%	52.39%	66.63%
SVM Regime 1	50.0%	62.5%	37.5%
SVM Regime 2	50%	62.5%	62.5%
NA Groups	91%	69%	78%
Basic RF	92%	65%	71%
GBM	58%	51%	71%

The highlighted cells correspond to be the retained model for each target label. The selection process was based on balancing i) a relatively high training AUC, ii) a reasonable run time to completion and iii) a relatively simple implementation.

The predicted target labels CSVs are available in the \prediction folder.

Challenges & Potential Improvements

- The data headers were obfuscated and most of the data were encrypted, which made it impossible to understand the problem domain under analysis. This forced the analysis to be solely data oriented without the possibility to provide any context to it.

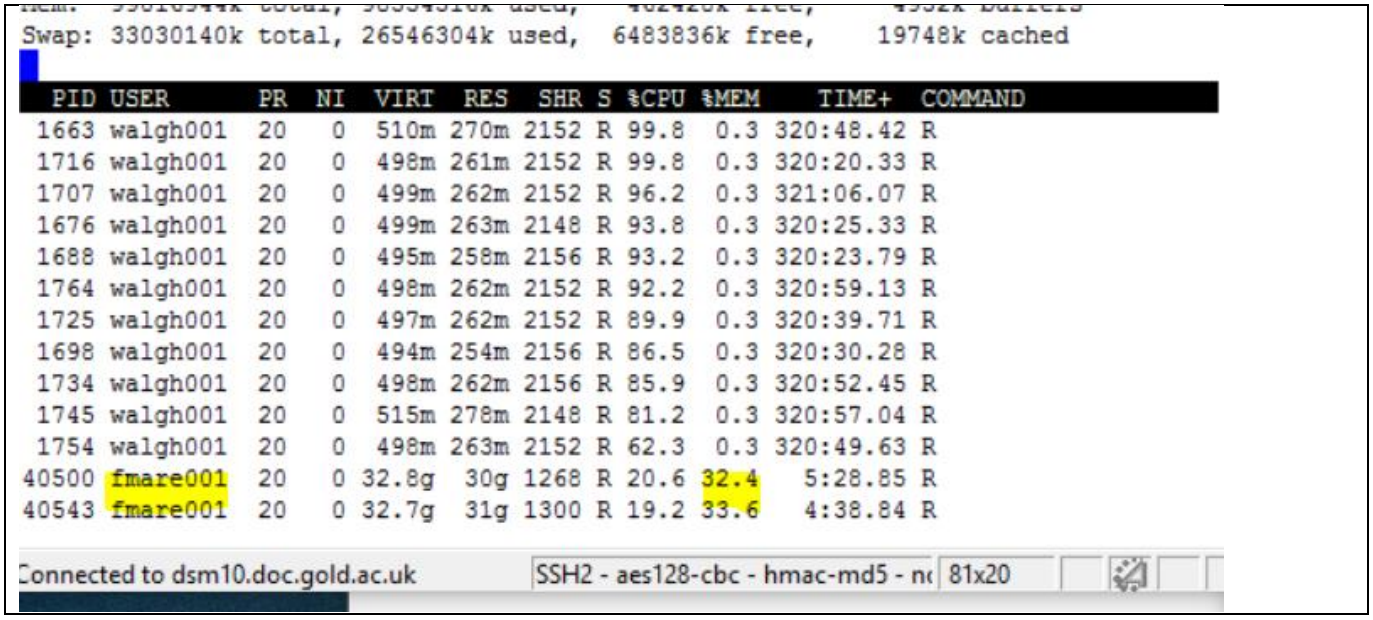
- All the algorithm used in this experiment had an issue with large amount of levels. Usually a number of levels greater than 53 would generate an error message of this nature:

  Error in randomForest.default(m, y, ...) :

  Can not handle categorical predictors with more than 53 categories

  It was therefore necessary to group labels in order to make the models run to completion. Aggregation the data has a potential to generate at training model that overfits the test data. However, there was no other choice.

- Some algorithms, such as the SVM model, are greedy when it comes to memory allocation. The below screen shot shows that 33% of the entire *dsm10* server memory was allocated per process when the entire training data set was passed in the nested cross validation harness. This had two consequences: i) it reduced the number of jobs that could be run on the server in parallel, ii) the process did not complete even 3 days or running. It was therefore necessary to reduce the dataset columns list to the list of feature selected columns prior to running the SVM model. This reduced the memory consumption to 0.3% of the server total memory.



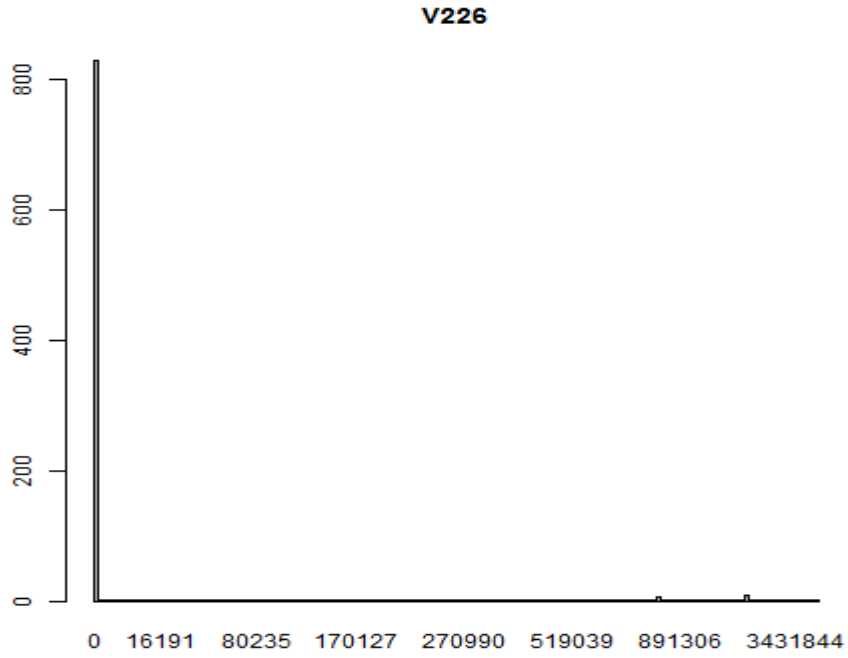
Conclusion

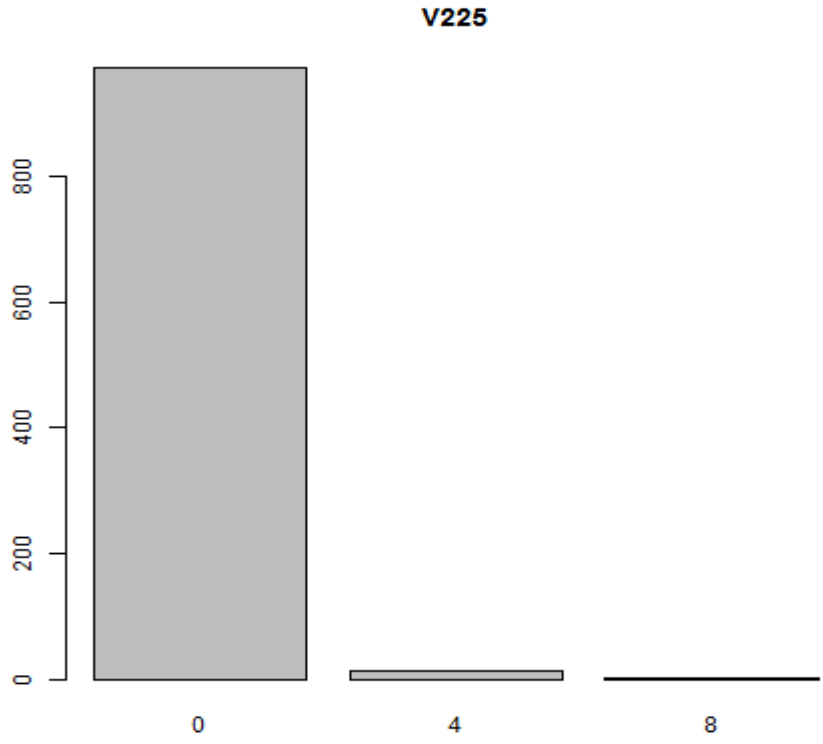
The purpose of this experiment to analyse some Orange marketing data and produce the best model for predicting the customers' propensity to switch provider (churn), buy new products (appetency) and upgrade their existing Orange contract (upselling). Due to the nature of the training data, largely encrypted coupled with a high proportion of missing data and a high number of category levels, a number of strategies were devised to maximise the AUC of each target labels. The best models retained for each of the target labels were: the Linear Discriminant Analysis (LDA) Scenario 1 (91% training AUC) for predicting appetency, the LDA Scenario 1 (72% training AUC) for forecasting churn and the "NA Group" model (78% training AUC) for predicting upselling. The predicted target labels CSVs are available in the \prediction folder.

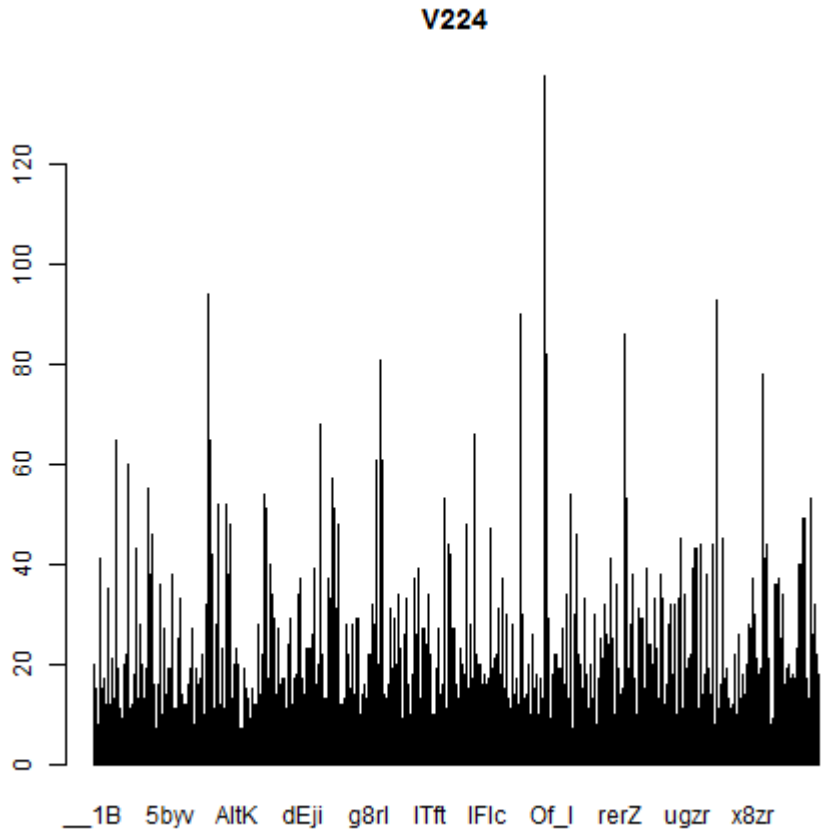
Appendix

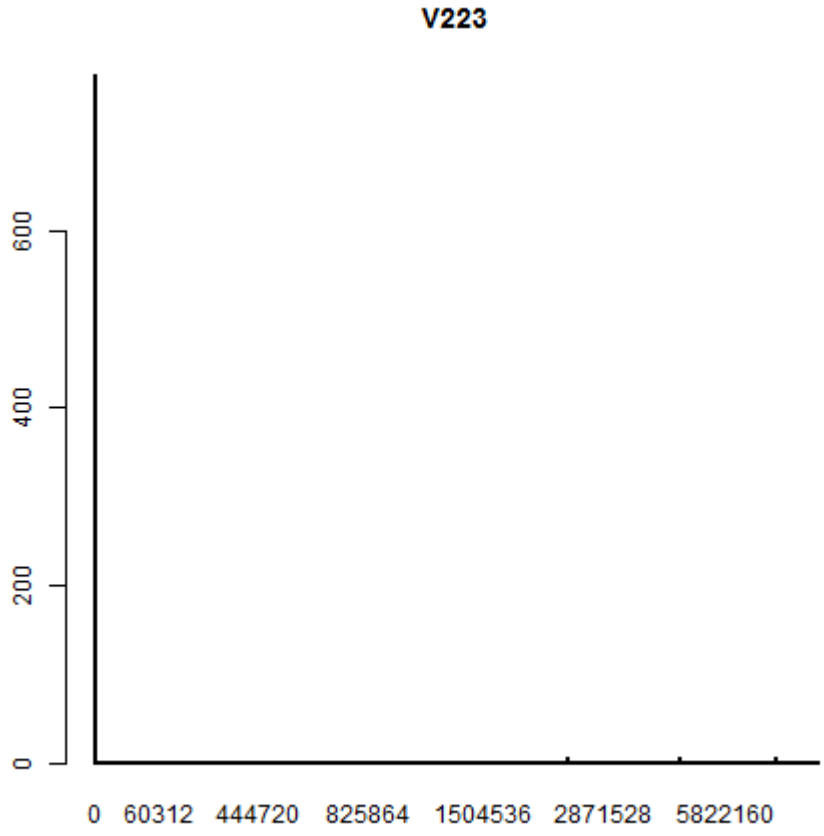
Appendix A – Level Frequency graph per category

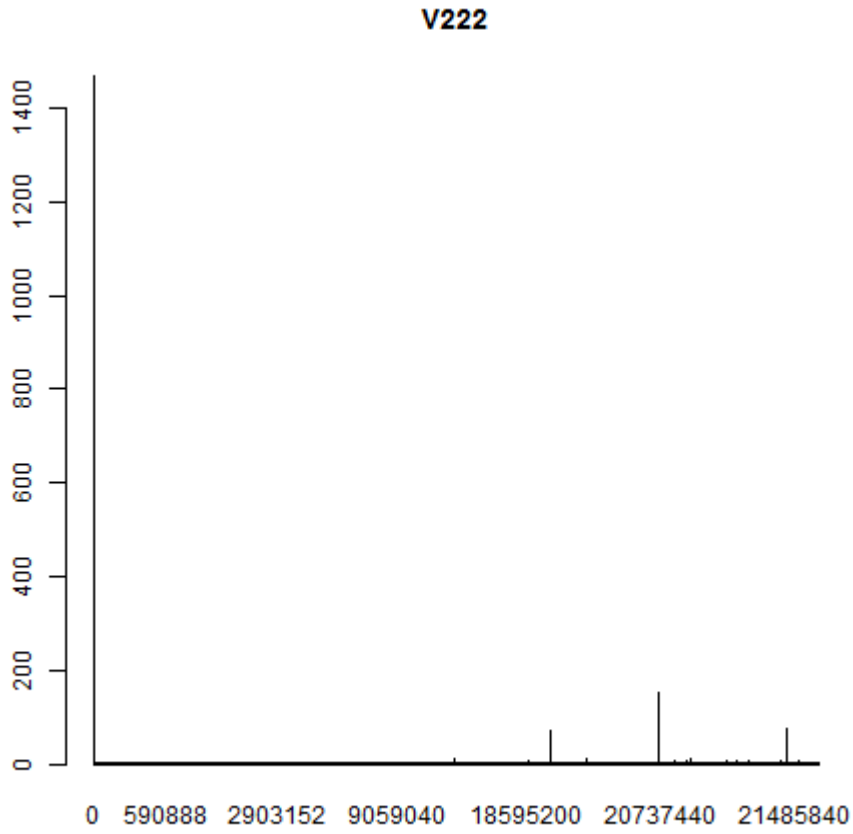
The category levels distribution images can be found in images\levels_distib

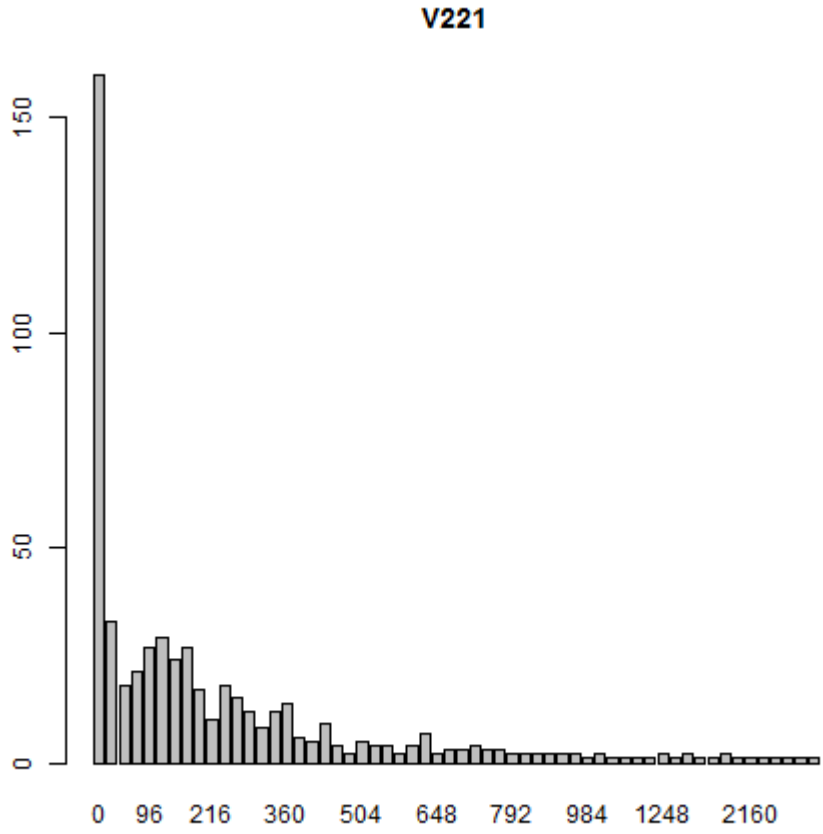


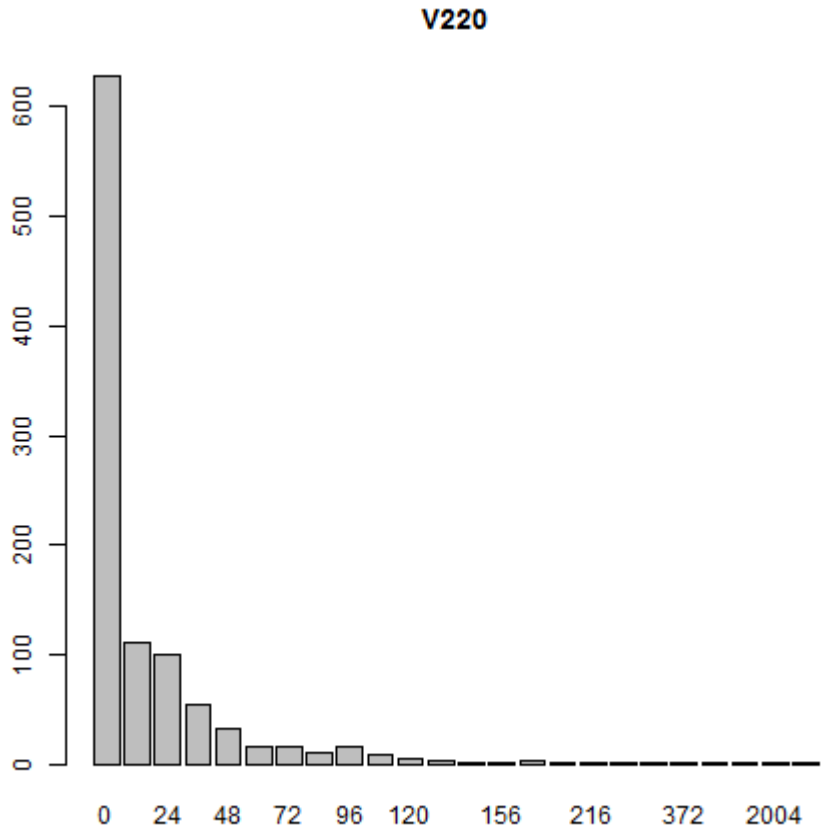


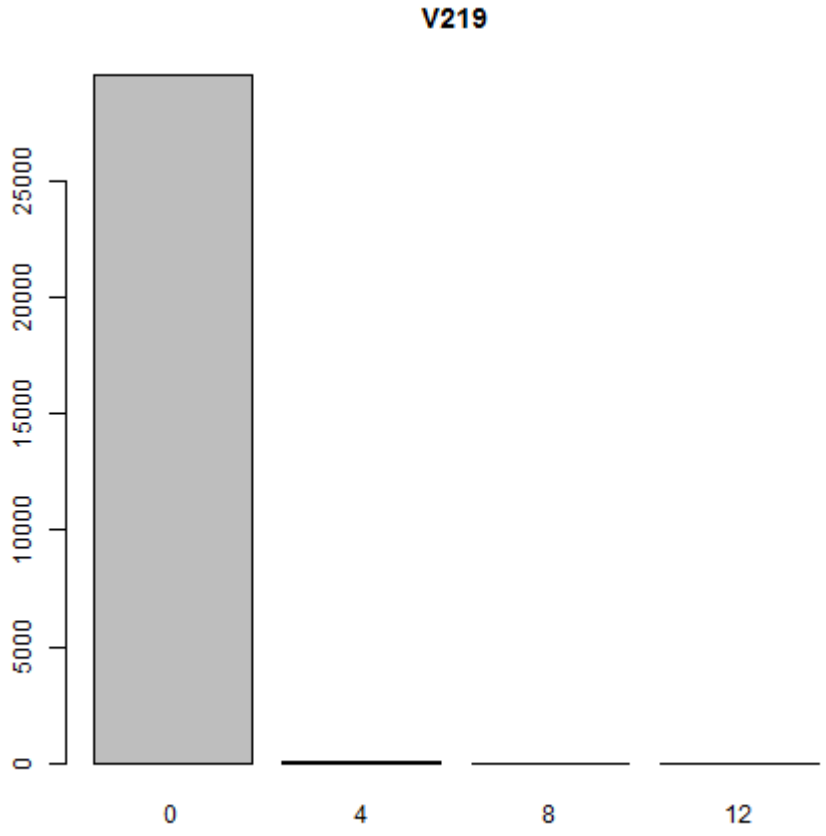

V224
0
20
40
60
80
100
120
__1B
5byv
AltK
dEji
g8rl
ITft
IFIc
Of_I
rerZ
ugzr
x8zr

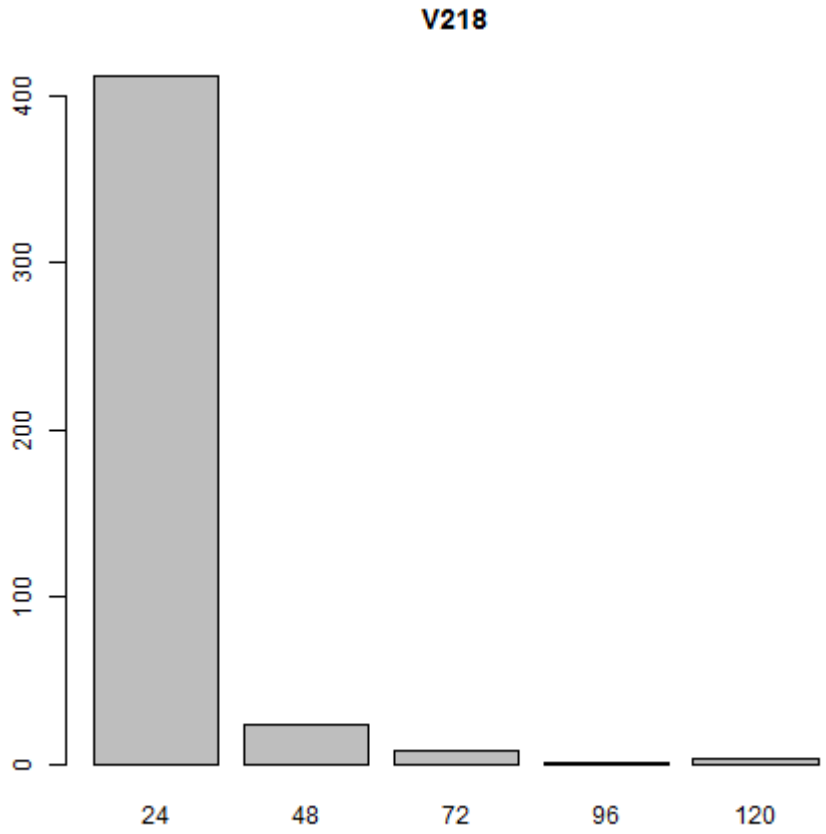

V223
0
200
400
600
0
60312
444720
825864
1504536
2871528
5822160

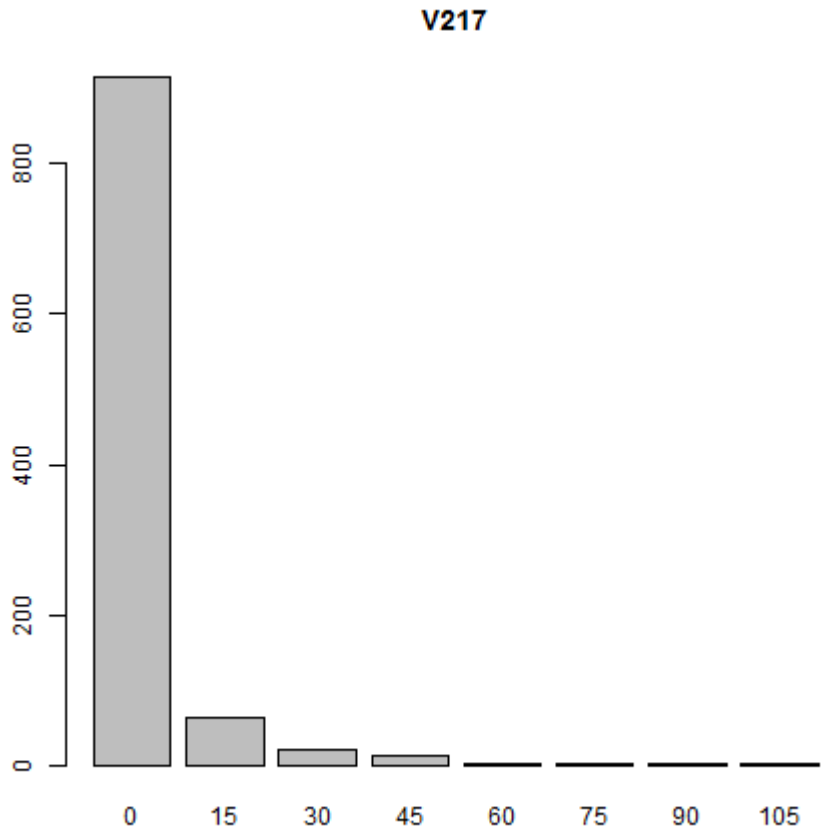

V222
0
200
400
600
800
1000
1200
1400
0
590888
2903152
9059040
18595200
20737440
21485840

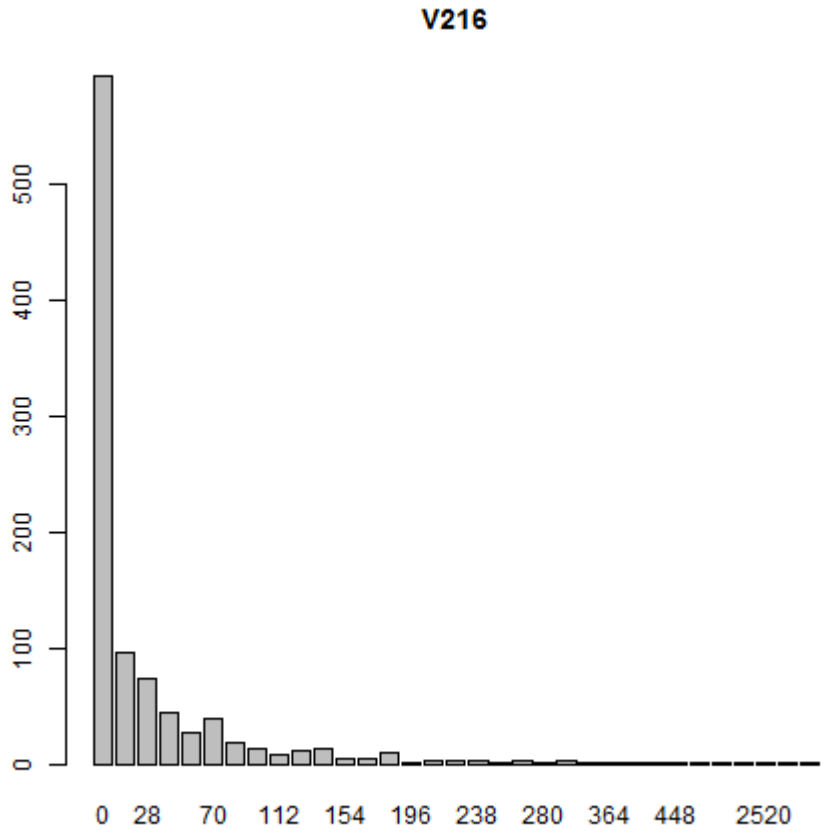

V221
0
50
100
150
0
96
216
360
504
648
792
984
1248
2160

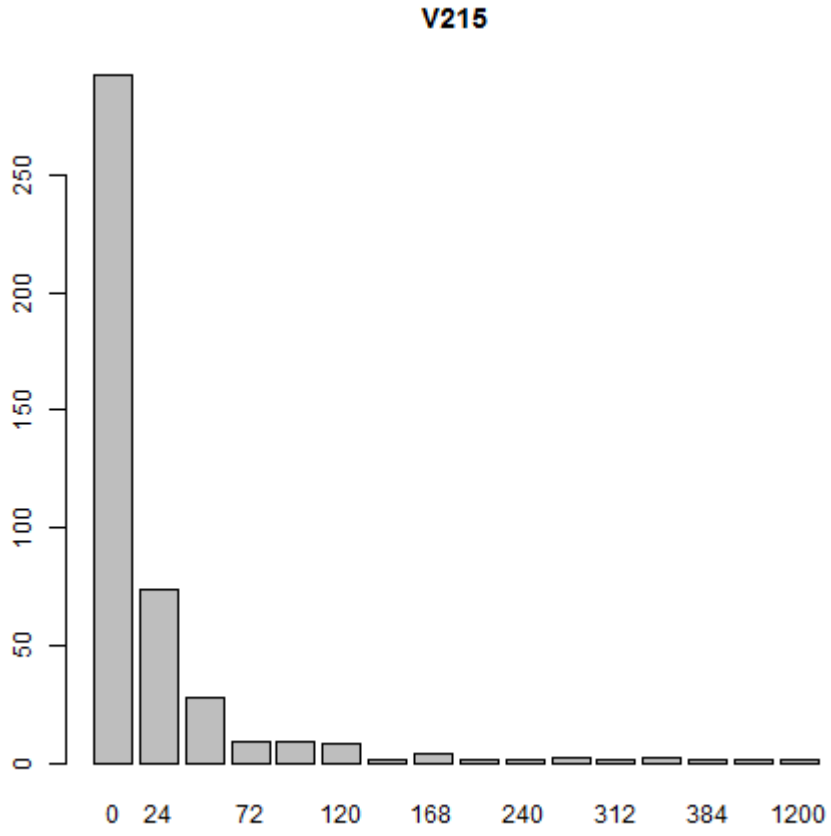

V220
0
100
200
300
400
500
600
0
24
48
72
96
120
156
216
372
2004

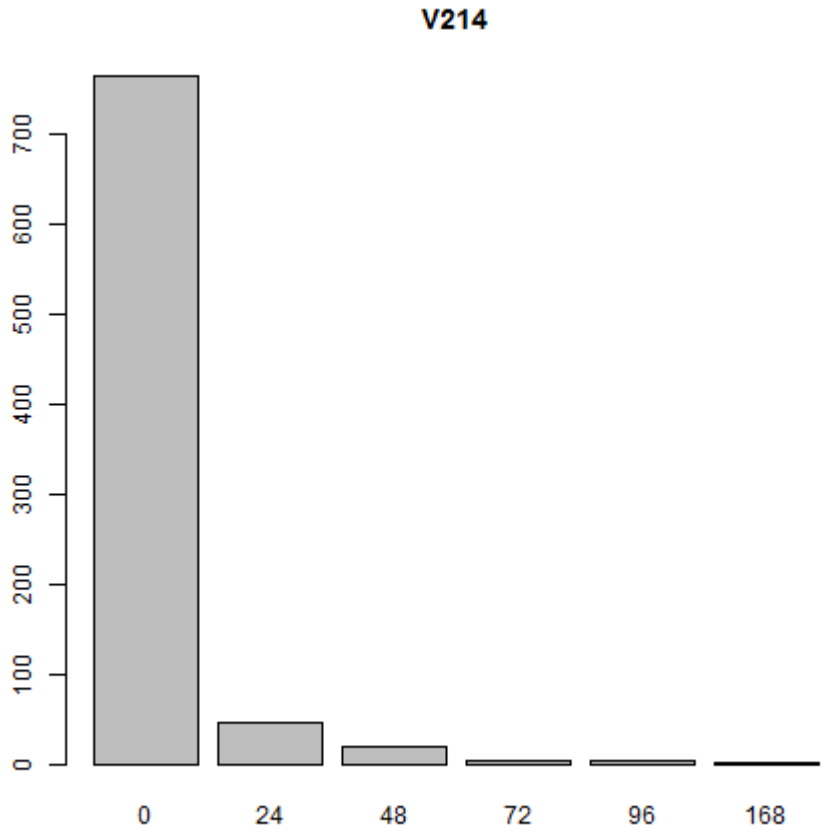

V219
0
5000
10000
15000
20000
25000
0
4
8
12

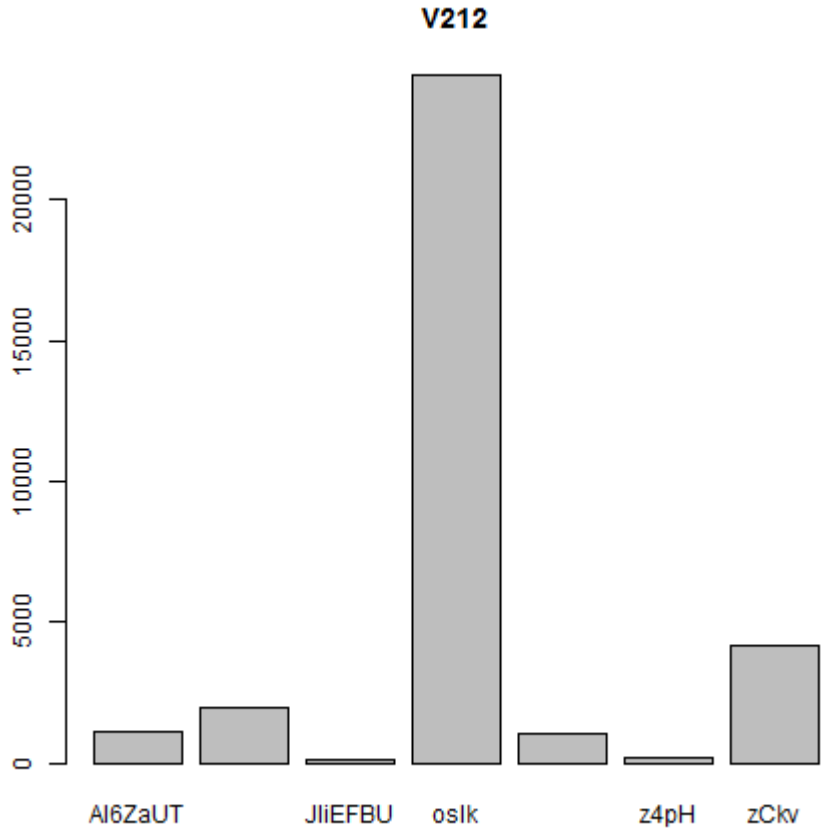

V218
0
100
200
300
400
24
48
72
96
120

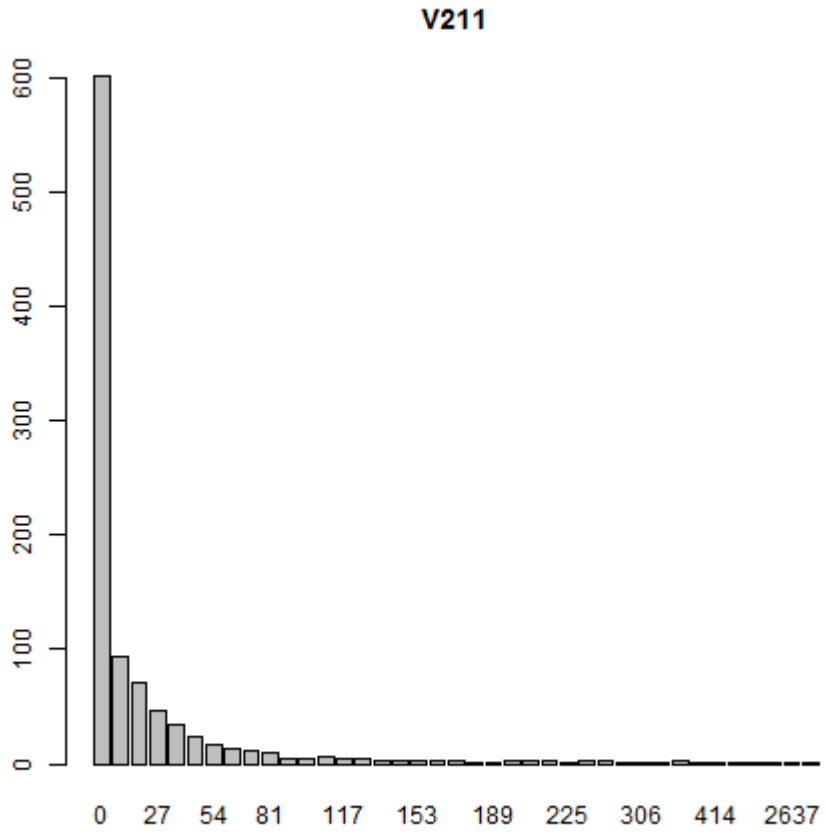

V217
0
200
400
600
800
0
15
30
45
60
75
90
105

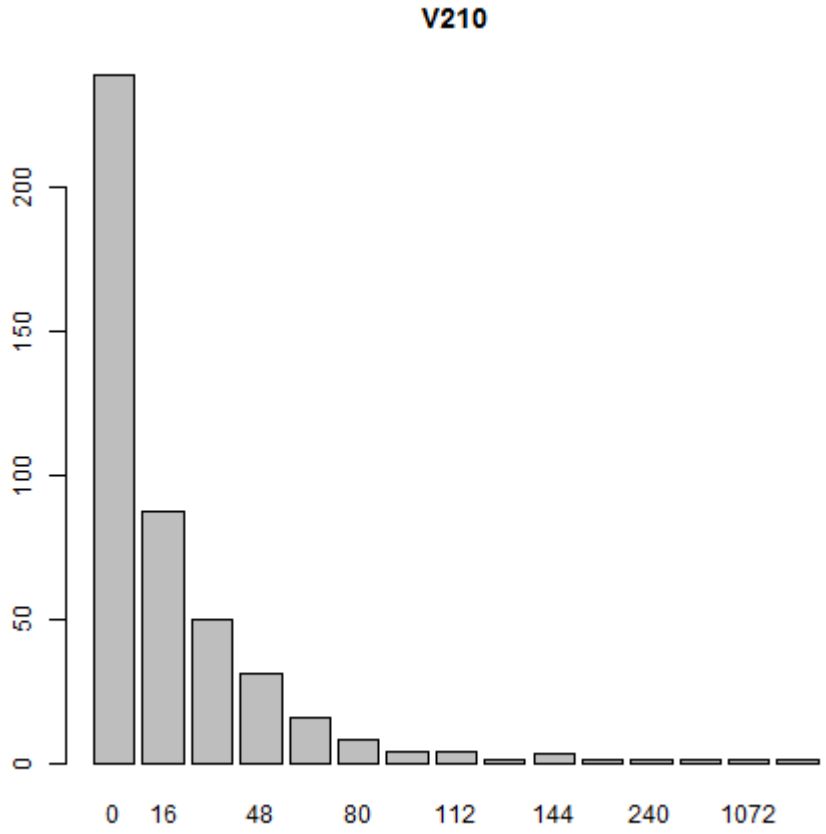

V216
0
100
200
300
400
500
0
28
70
112
154
196
238
280
364
448
2520

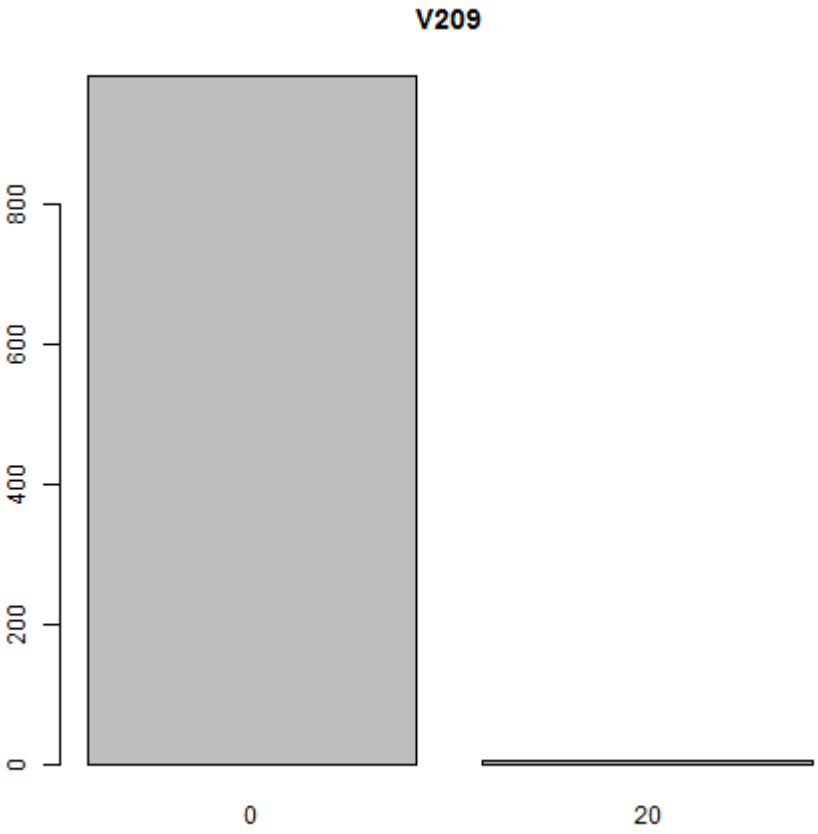

V215
0
50
100
150
200
250
0
24
72
120
168
240
312
384
1200

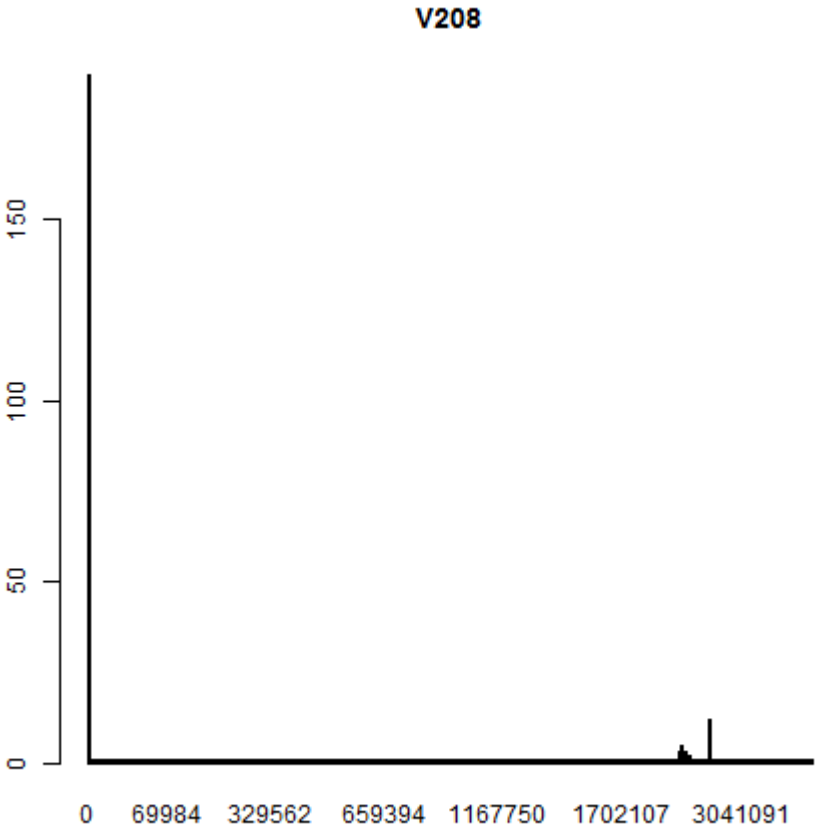

V214
0
100
200
300
400
500
600
700
0
24
48
72
96
168

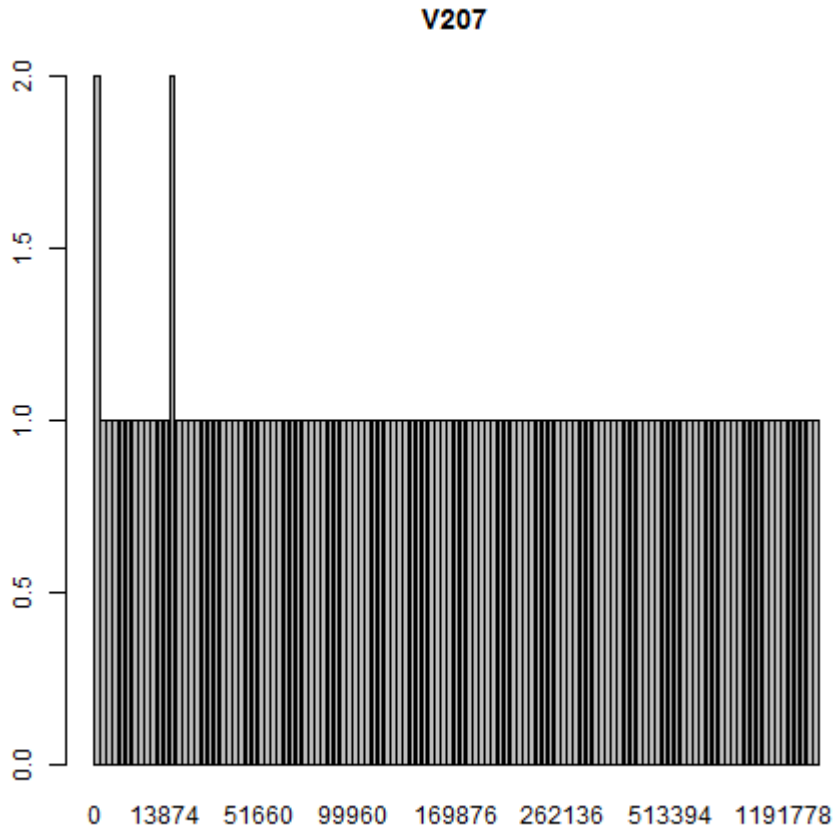

V212
0
5000
10000
15000
20000
Al6ZaUT
JliEFBU
oslk
z4pH
zCkv

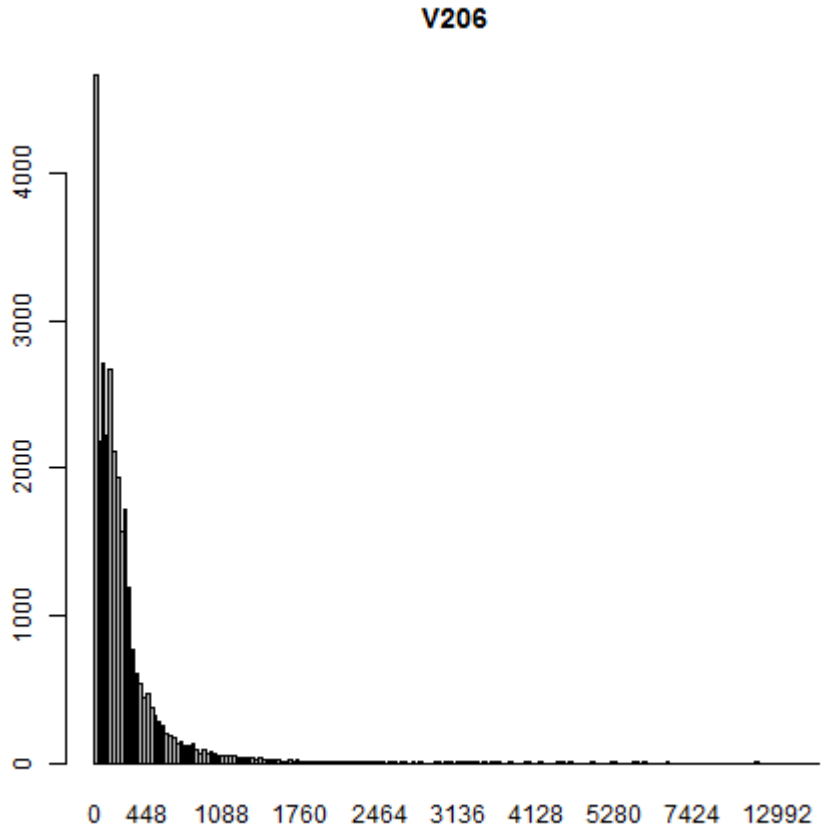

V211
600
500
400
300
200
100
0
0
27
54
81
117
153
189
225
306
414
2637

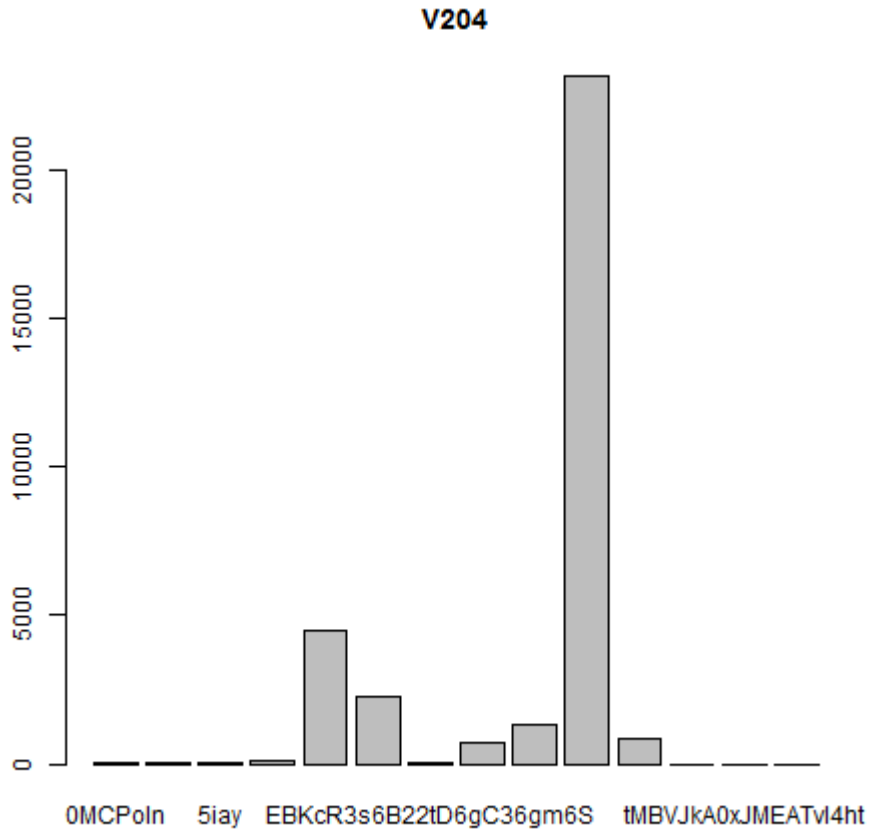

V210
200
150
100
50
0
0
16
48
80
112
144
240
1072

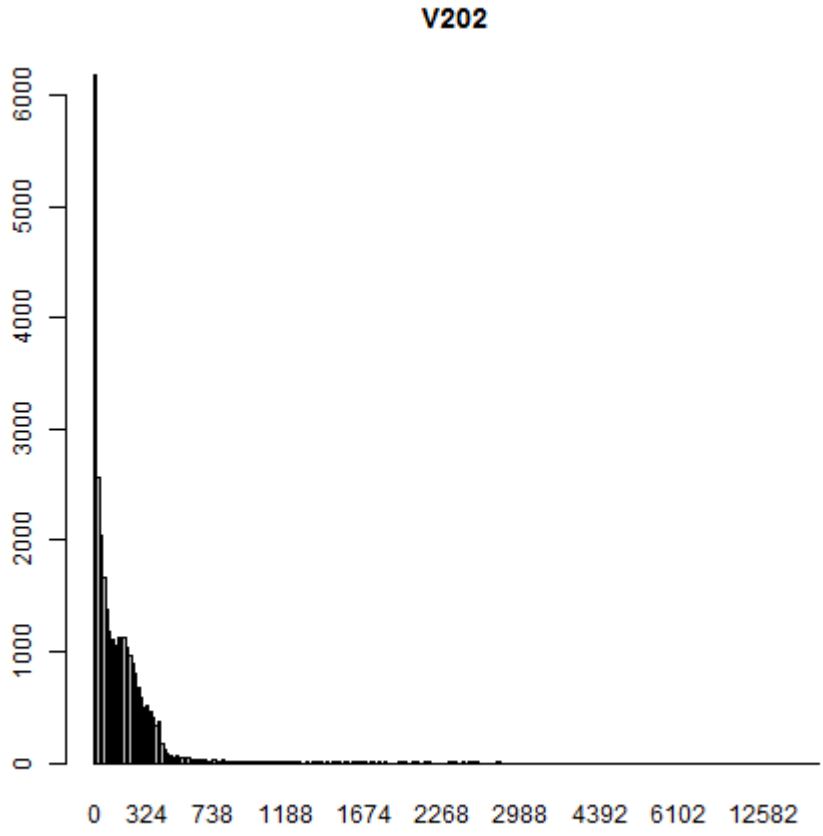

V209
800
600
400
200
0
0
20

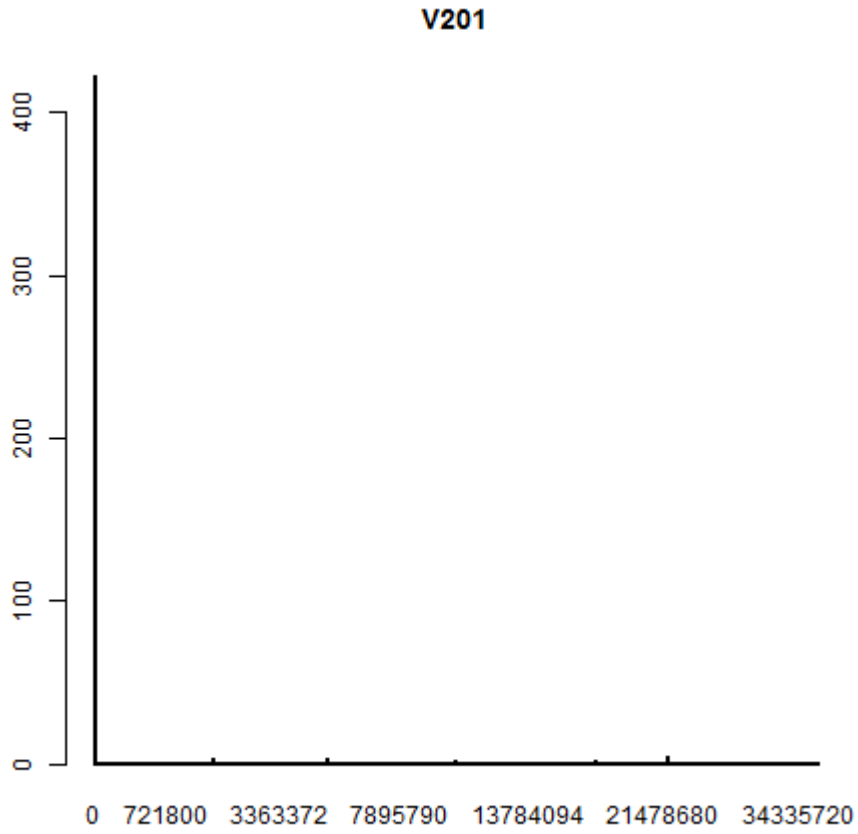

V208
150
100
50
0
0
69984
329562
659394
1167750
1702107
3041091

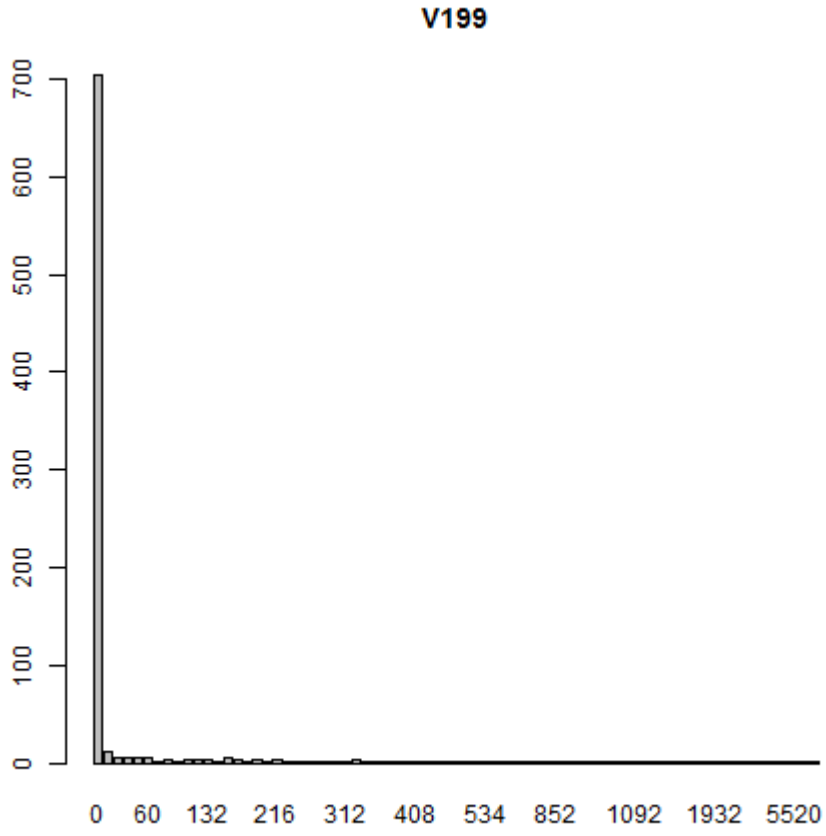

V207
2.0
1.5
1.0
0.5
0.0
0
13874
51660
99960
169876
262136
513394
1191778

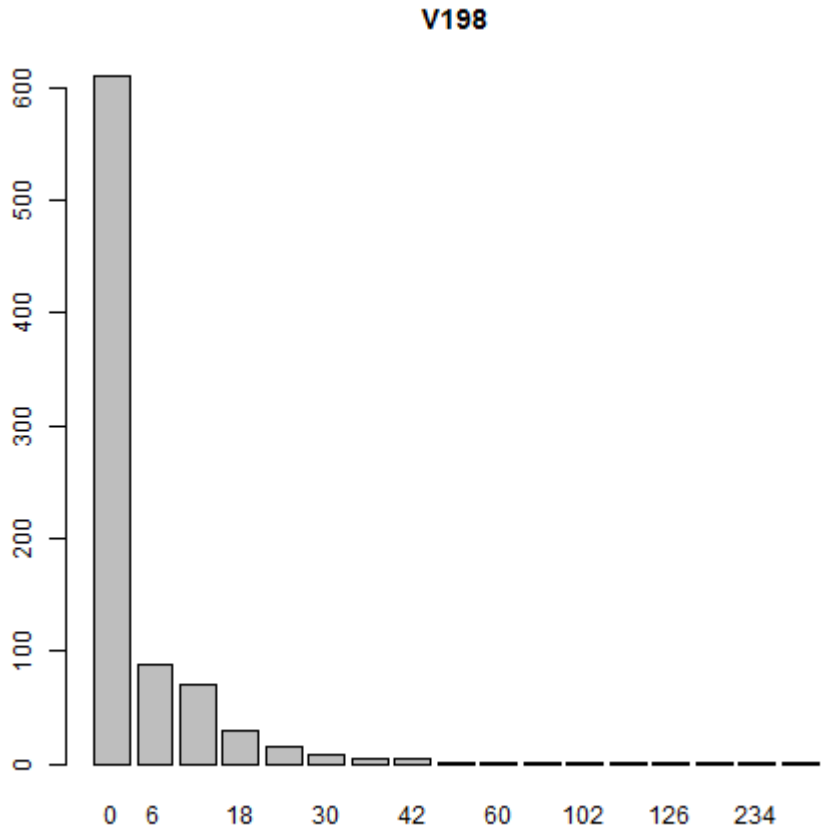

V206
4000
3000
2000
1000
0
0
448
1088
1760
2464
3136
4128
5280
7424
12992

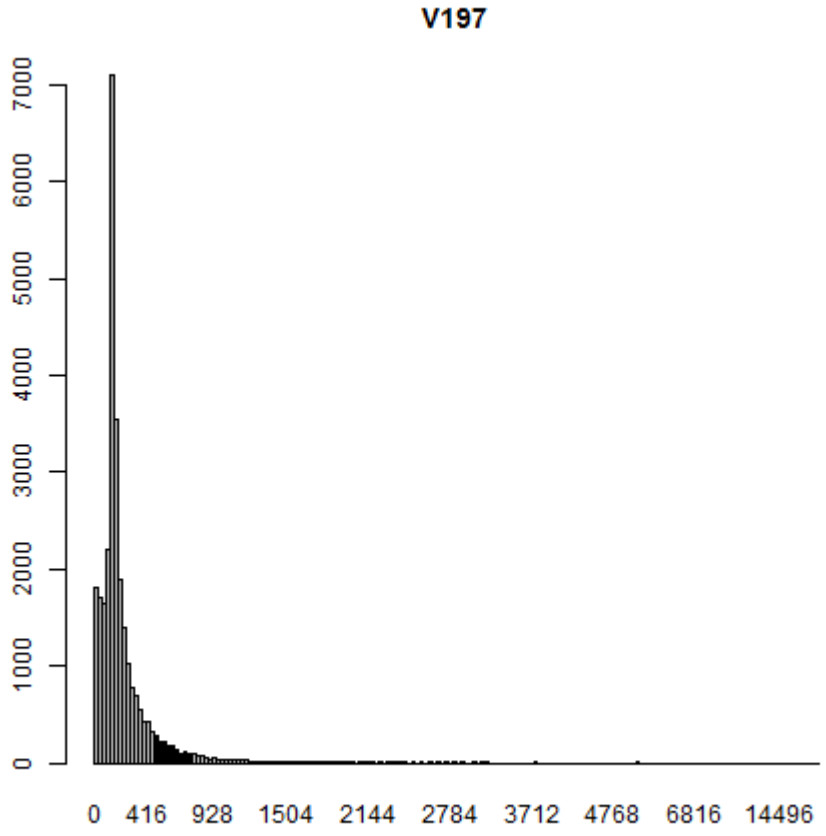

V204
0
5000
10000
15000
20000
0MCPoln
5iay
EBKcR3s6B22tD6gC36gm6S
tMBVJkA0xJMEATvl4ht

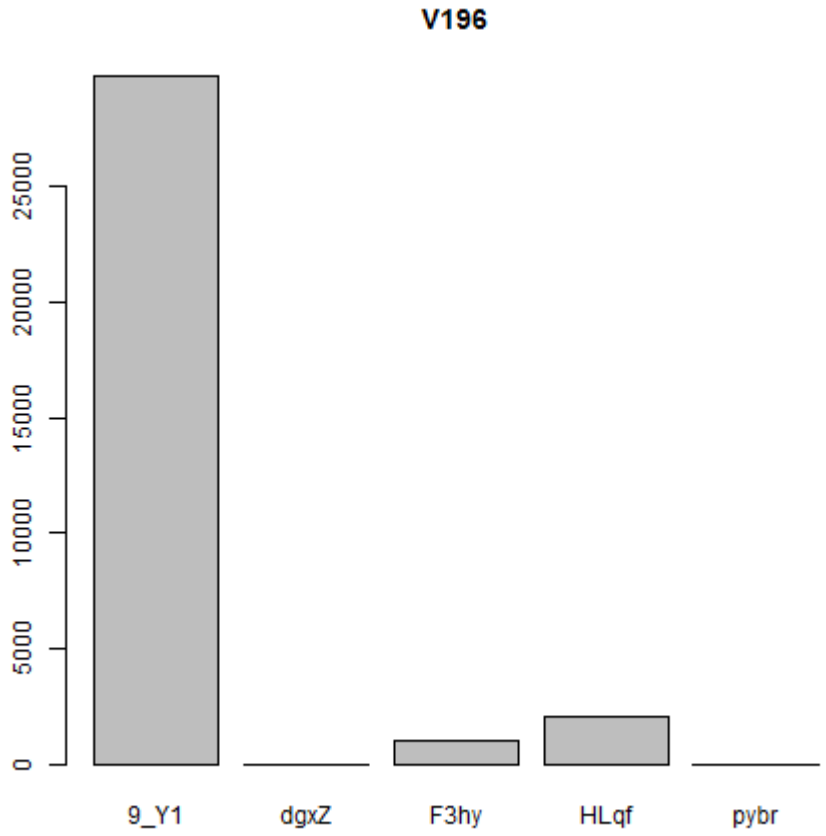

V202
0
1000
2000
3000
4000
5000
6000
0
324
738
1188
1674
2268
2988
4392
6102
12582

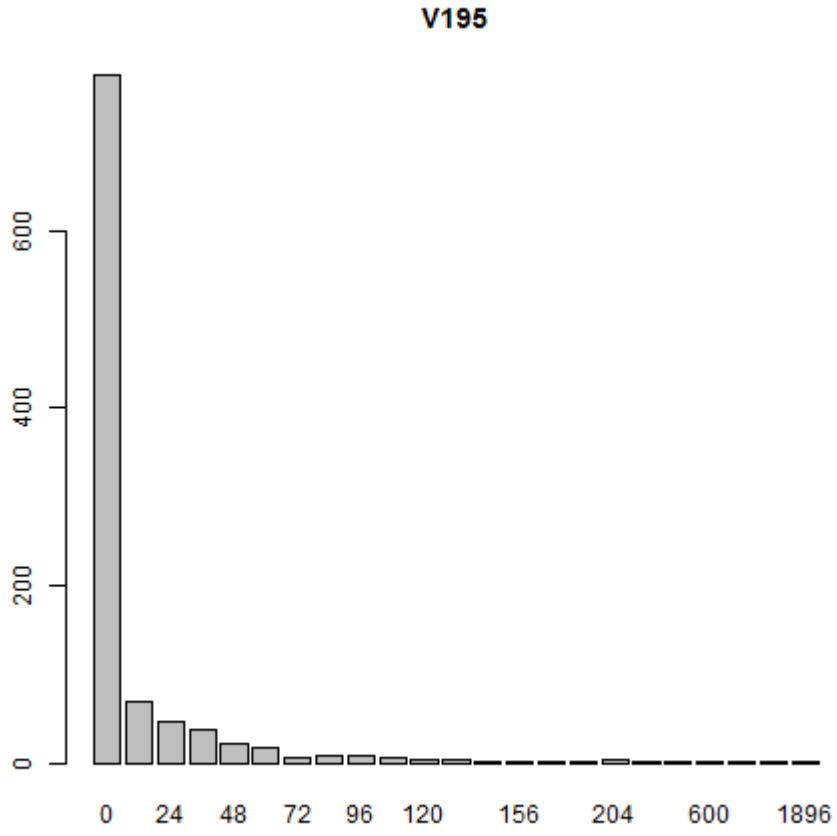

V201
400
300
200
100
0
0
721800
3363372
7895790
13784094
21478680
34335720

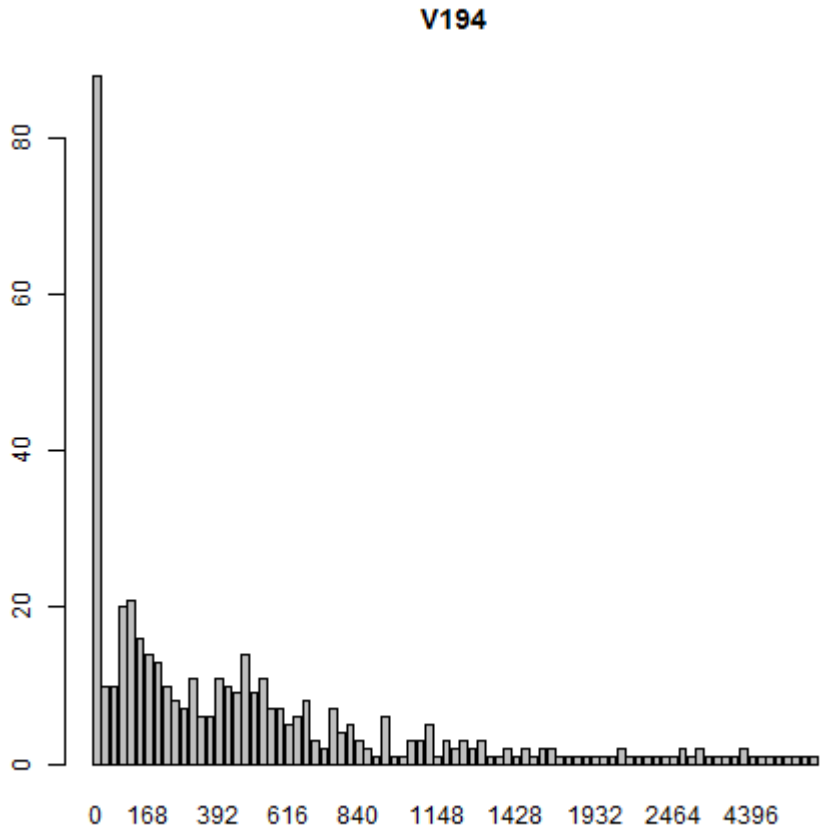

V199
700
600
500
400
300
200
100
0
0
60
132
216
312
408
534
852
1092
1932
5520

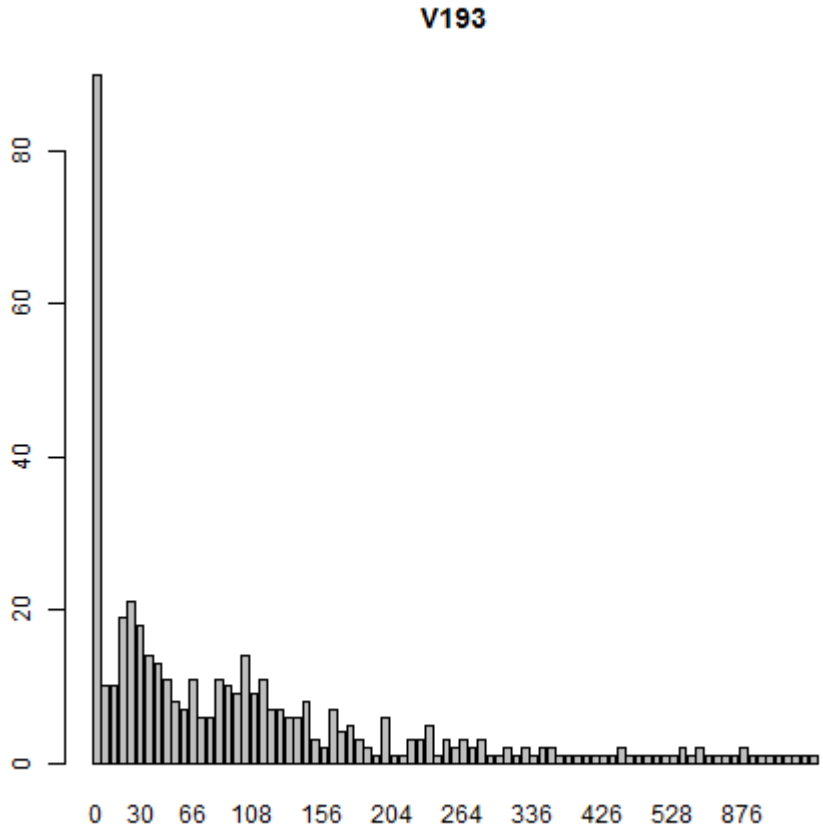

V198
0
100
200
300
400
500
600
0
6
18
30
42
60
102
126
234

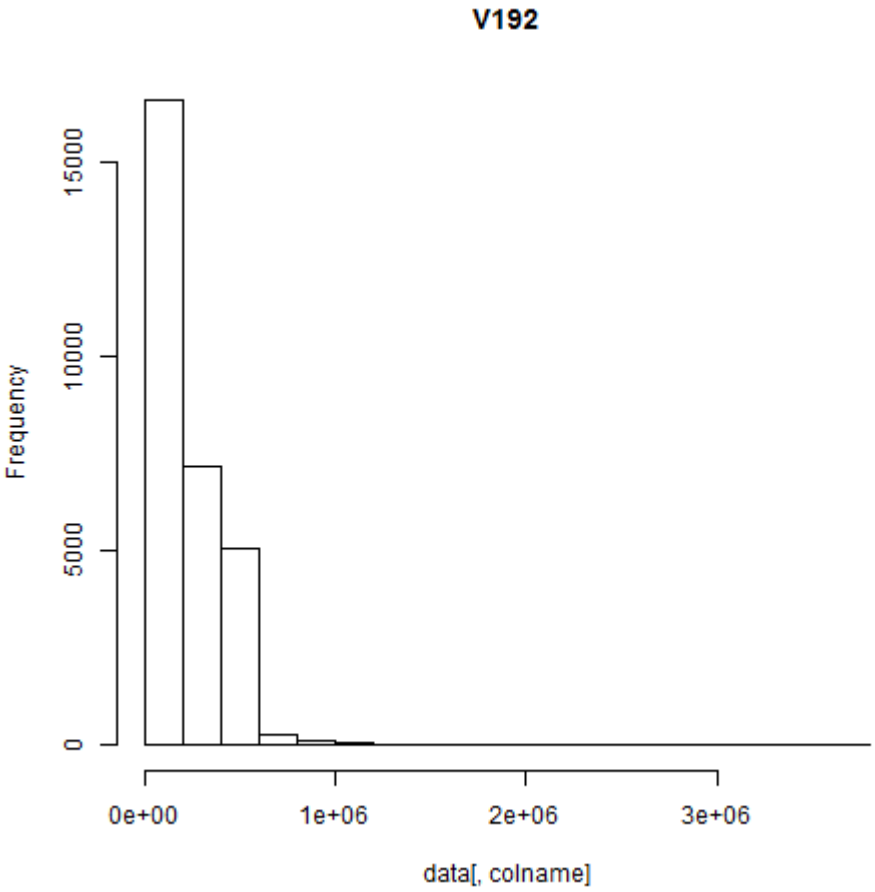

V197
0
1000
2000
3000
4000
5000
6000
7000
0
416
928
1504
2144
2784
3712
4768
6816
14496

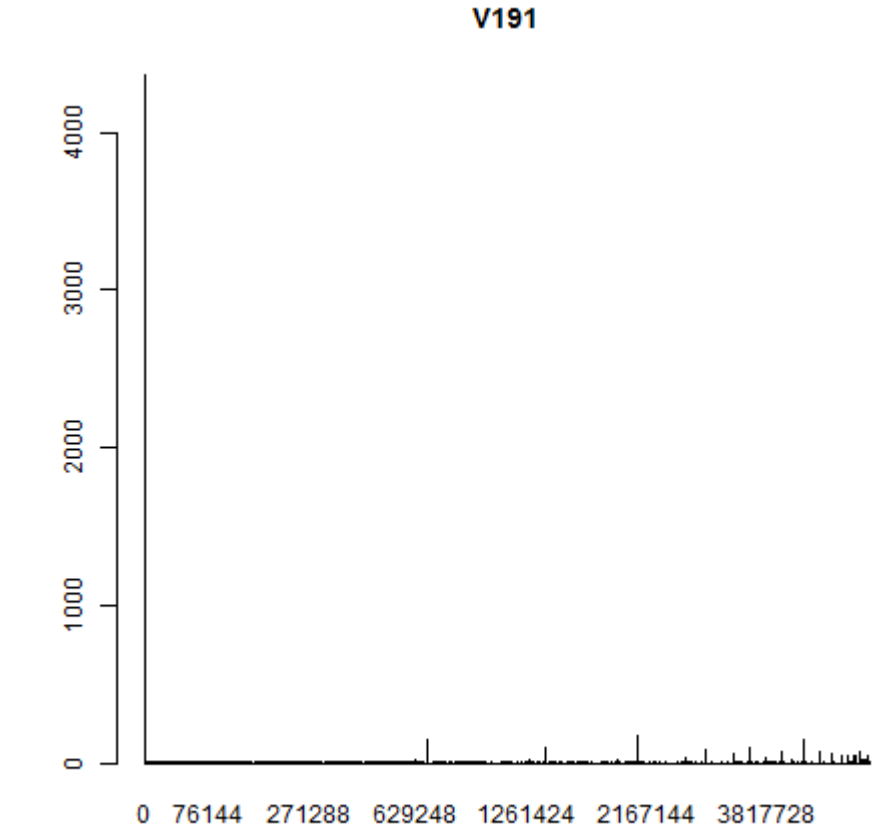

V196
25000
20000
15000
10000
5000
0
9_Y1
dgxZ
F3hy
HLqf
pybr

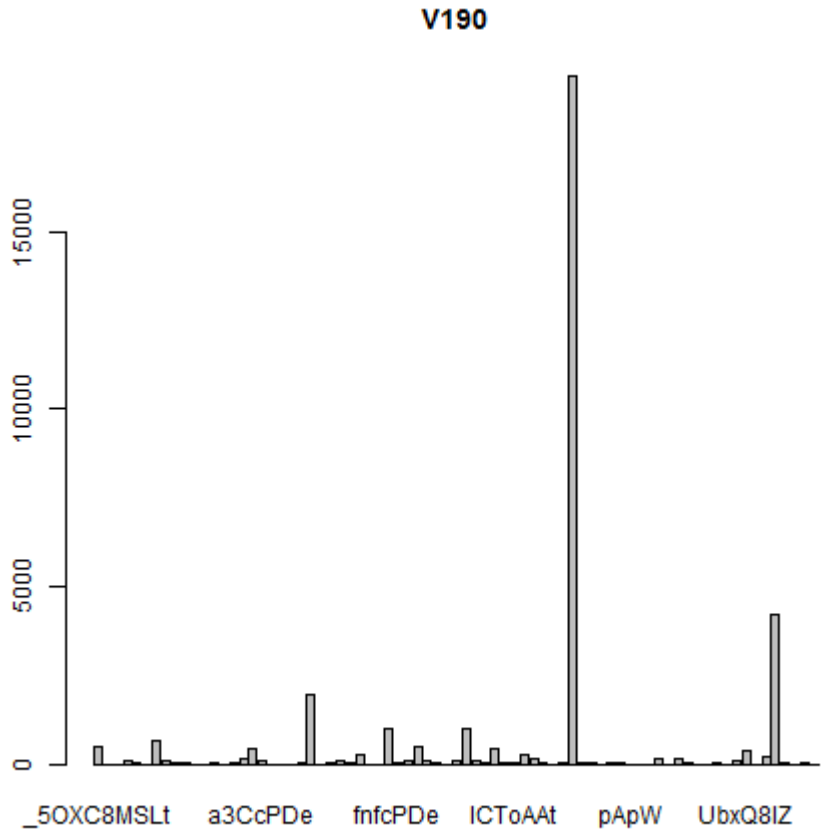

V195
600
400
200
0
0
24
48
72
96
120
156
204
600
1896

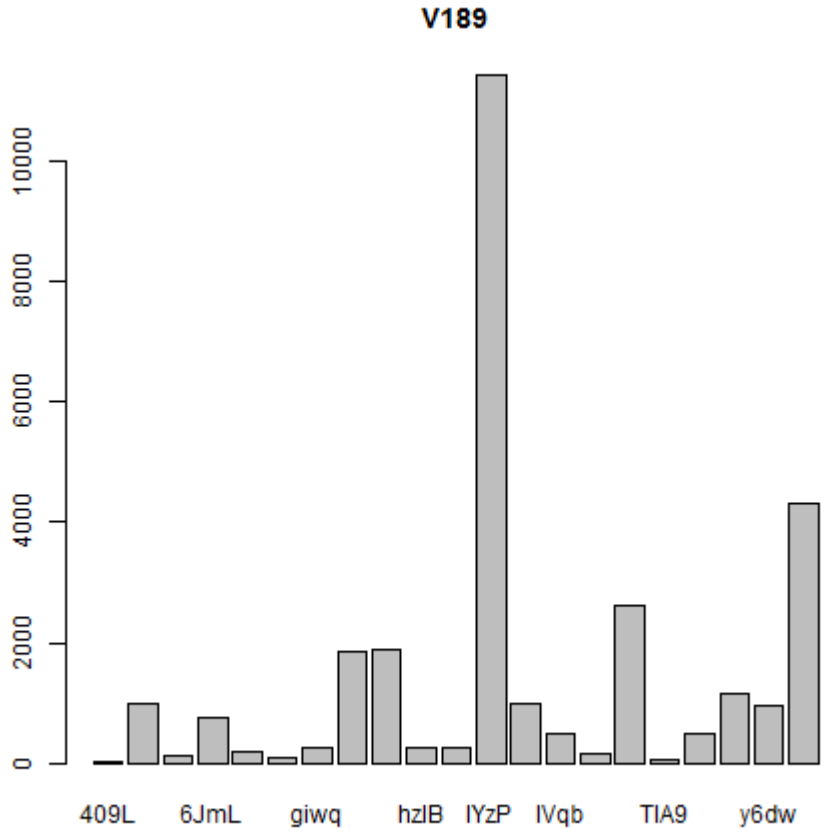

V194
0
20
40
60
80
0
168
392
616
840
1148
1428
1932
2464
4396

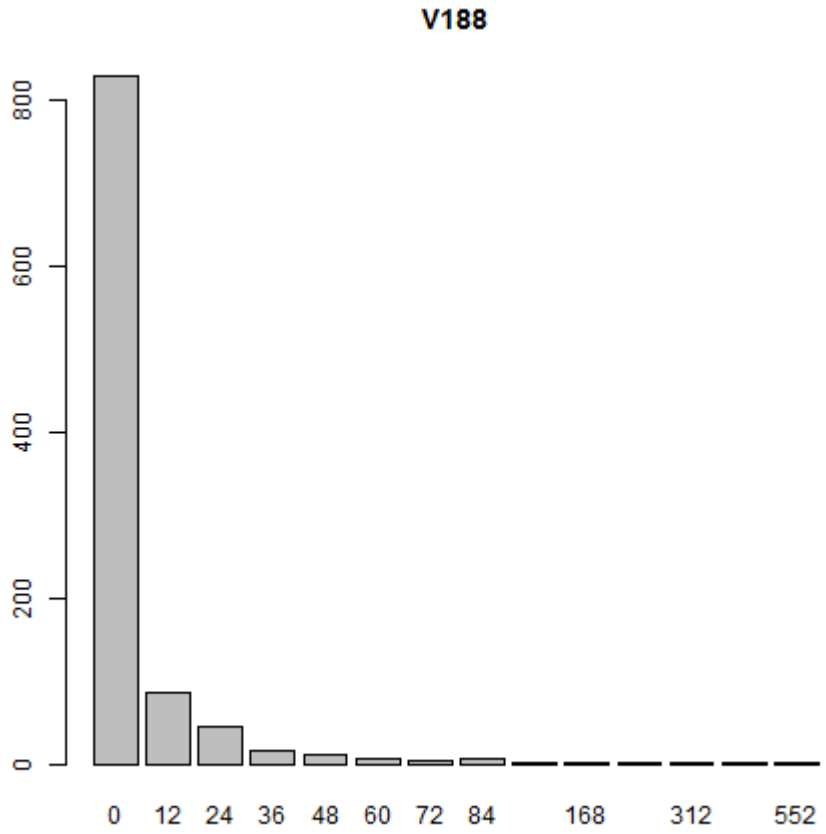

V193
0
20
40
60
80
0
30
66
108
156
204
264
336
426
528
876

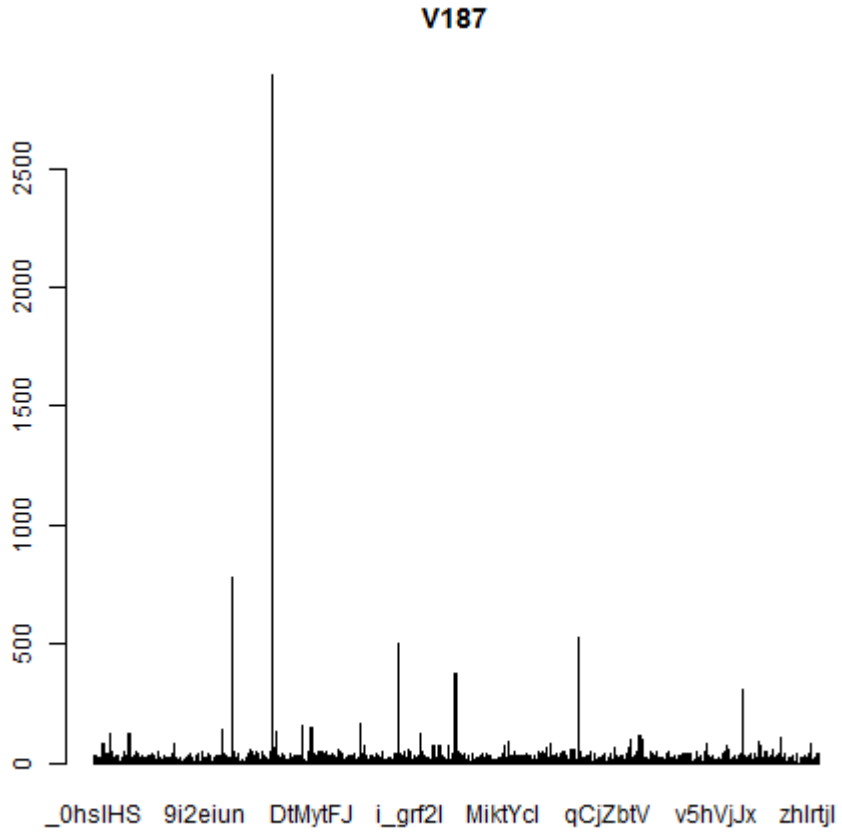

V192
Frequency
0
5000
10000
15000
0e+00
1e+06
2e+06
3e+06
data[, colname]

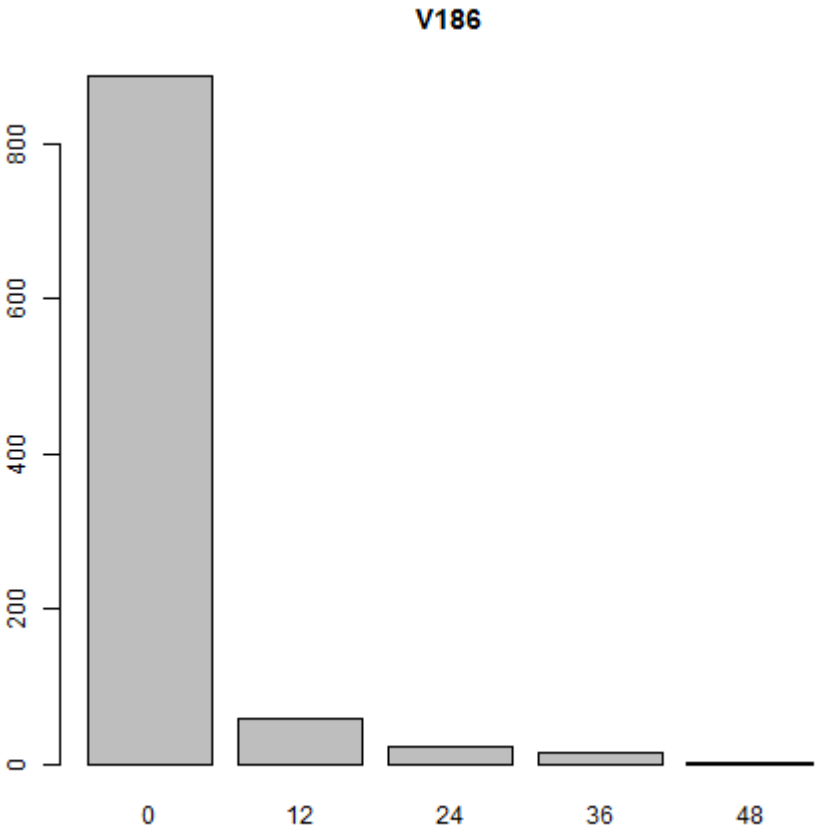

V191
0
1000
2000
3000
4000
0
76144
271288
629248
1261424
2167144
3817728

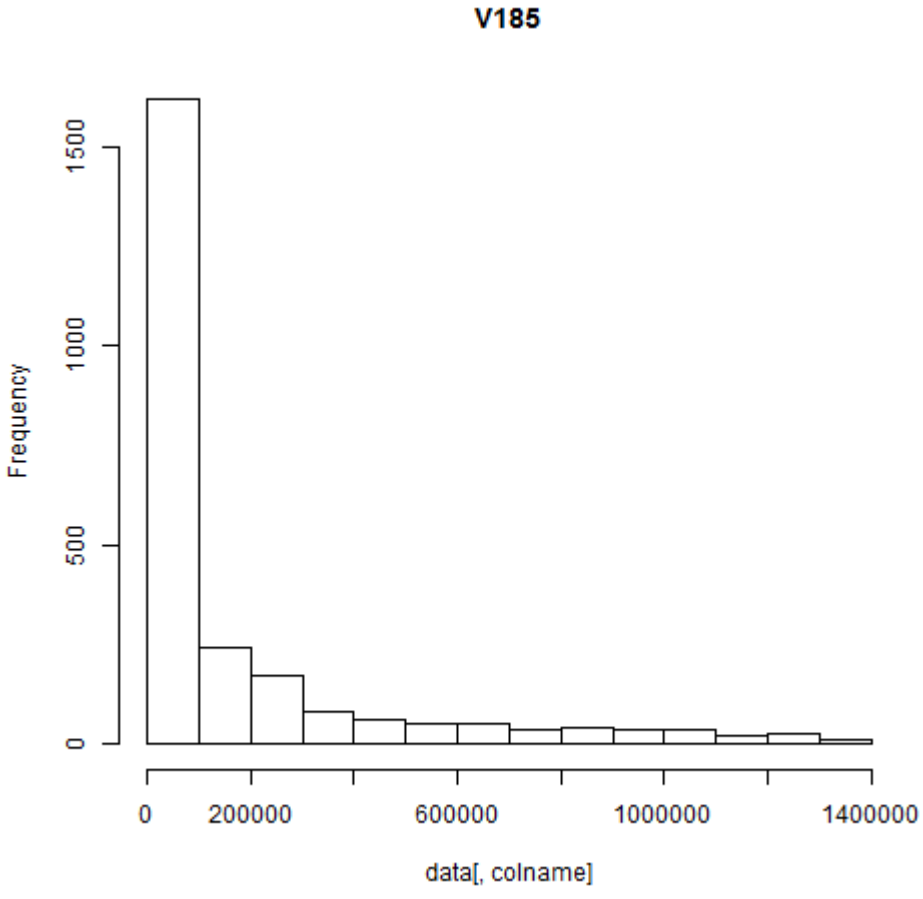

V190
15000
10000
5000
0
_5OXC8MSLt
a3CcPDe
fnfcPDe
lCToAAt
pApW
UbxQ8IZ

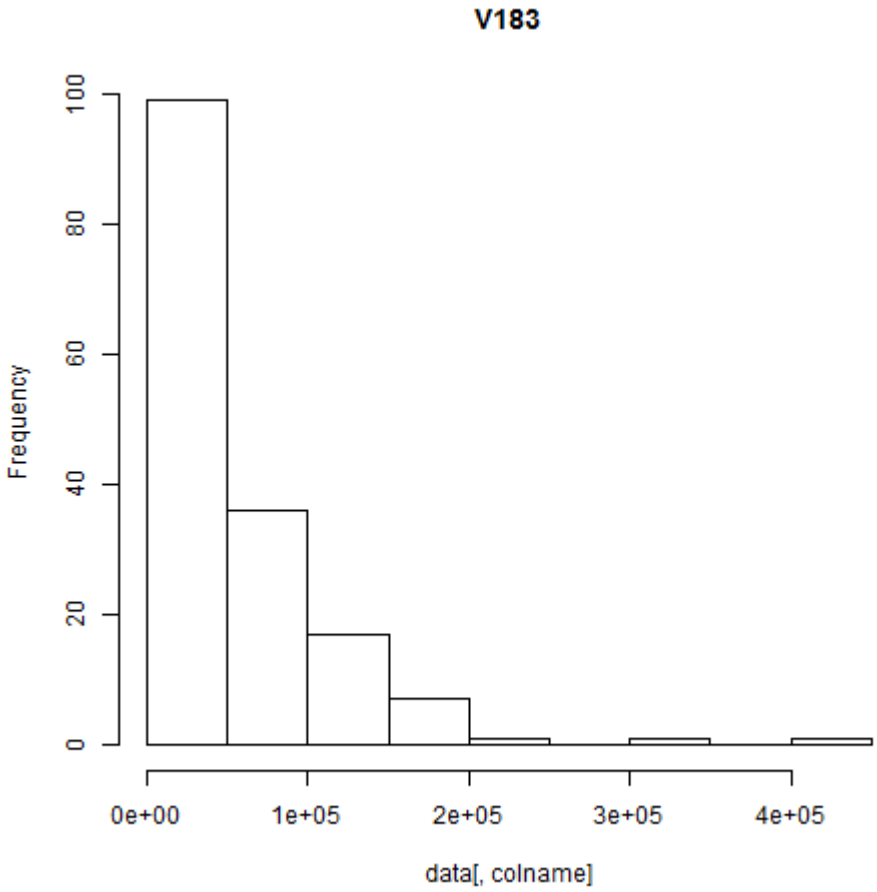

V189
10000
8000
6000
4000
2000
0
409L
6JmL
giwq
hzlB
lYzP
lVqb
TIA9
y6dw

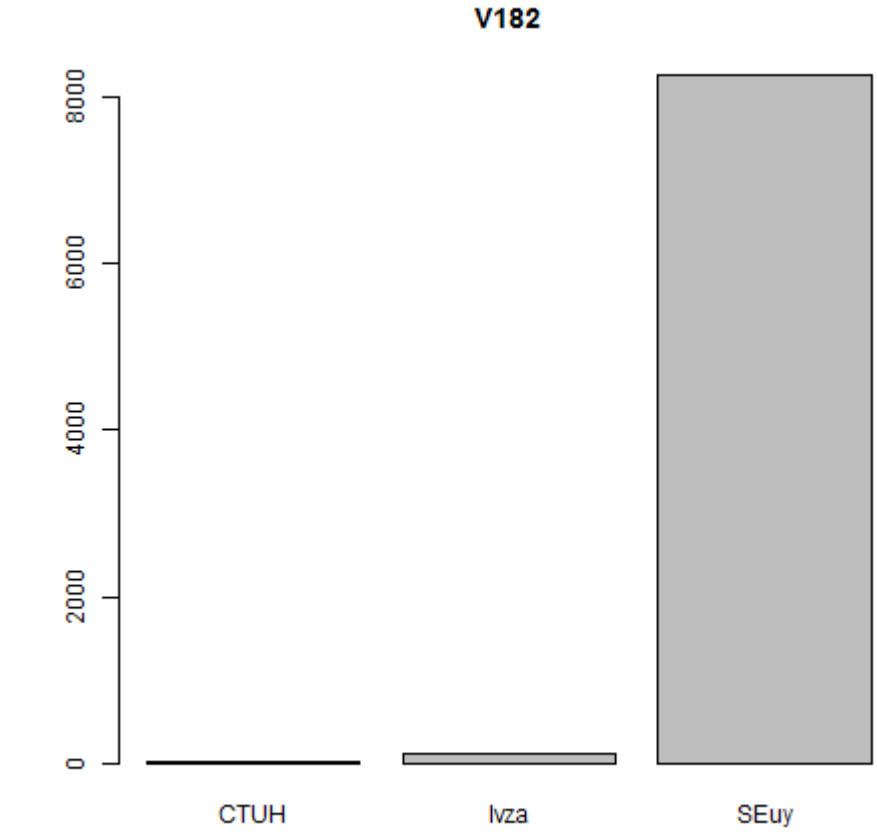

V188
800
600
400
200
0
0
12
24
36
48
60
72
84
168
312
552

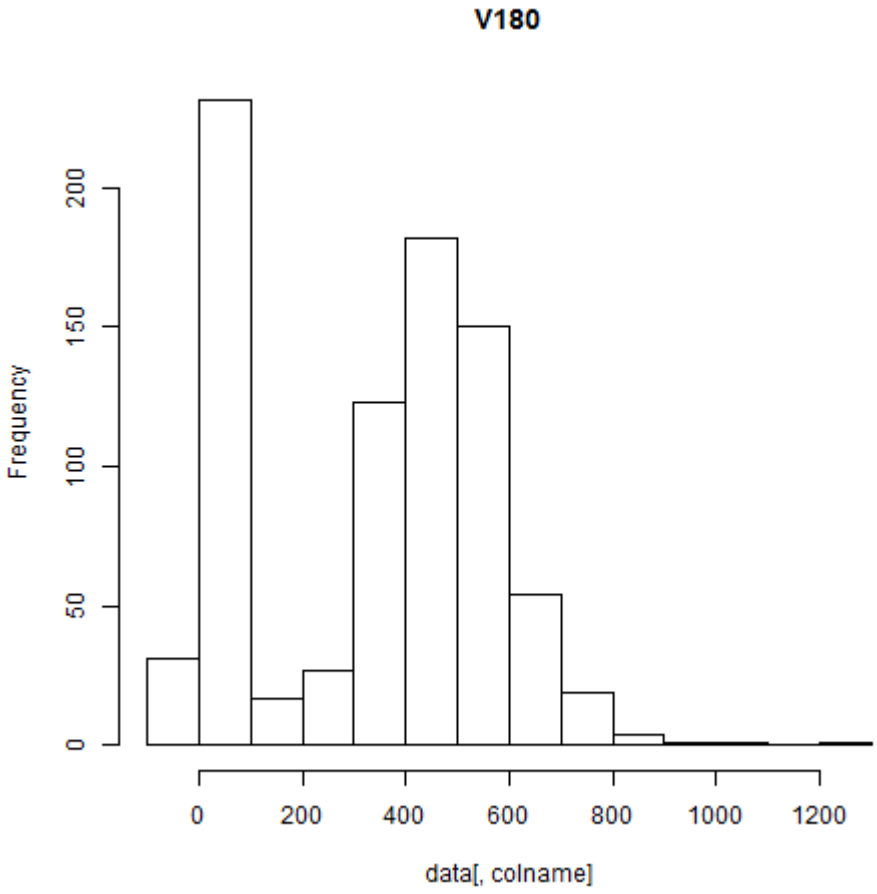

V187
2500
2000
1500
1000
500
0
_0hsIHS
9i2eiun
DtMytFJ
i_grf2l
MiktYcl
qCjZbtV
v5hVjJx
zhlrtjl

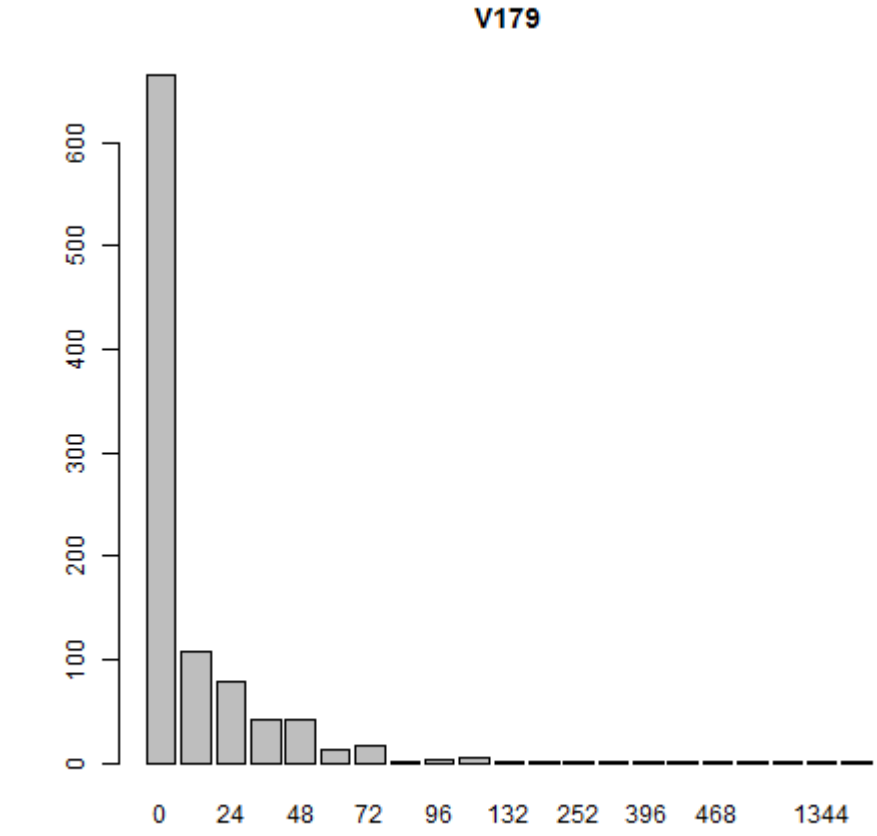

V186
800
600
400
200
0
0
12
24
36
48

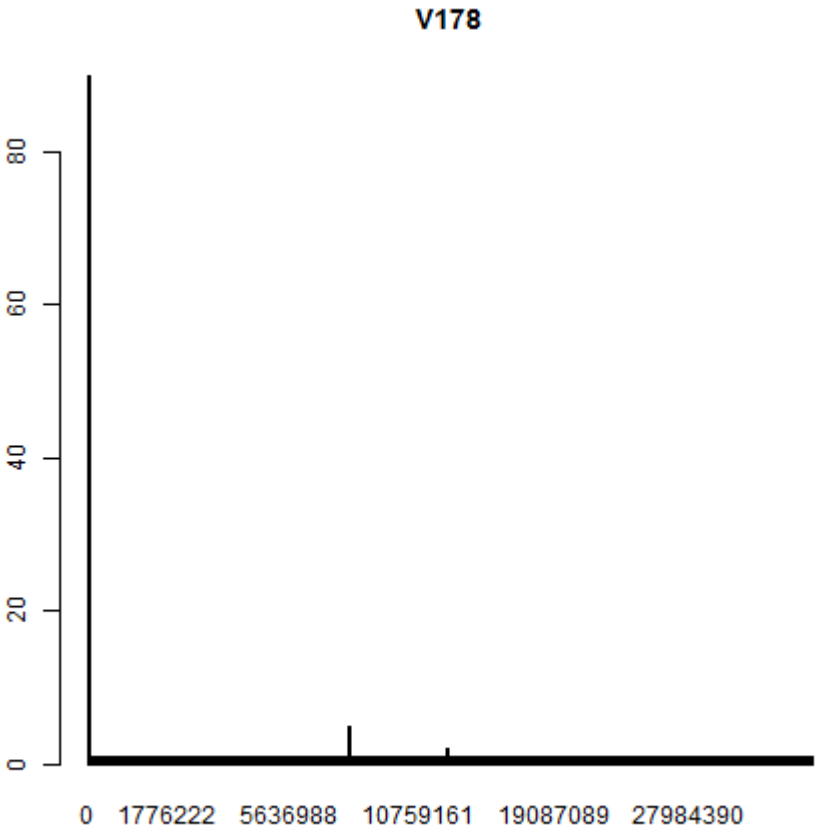

V185
Frequency
1500
1000
500
0
0
200000
600000
1000000
1400000
data[, colname]

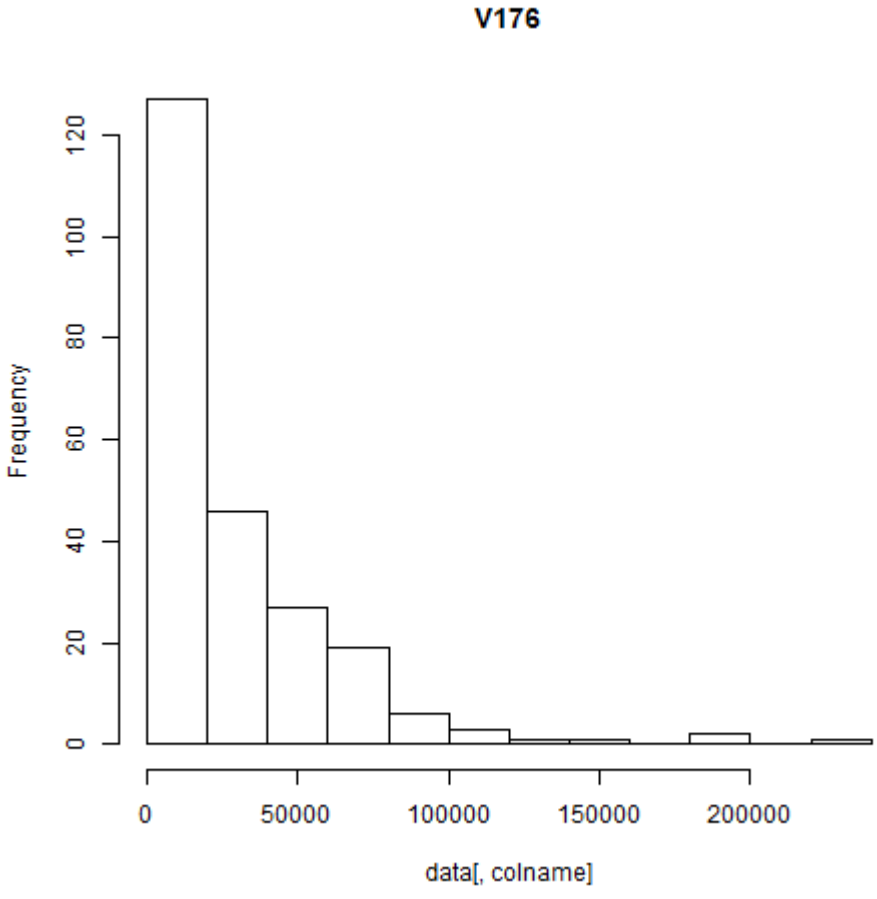

V183
Frequency
0
20
40
60
80
100
0e+00
1e+05
2e+05
3e+05
4e+05
data[, colname]

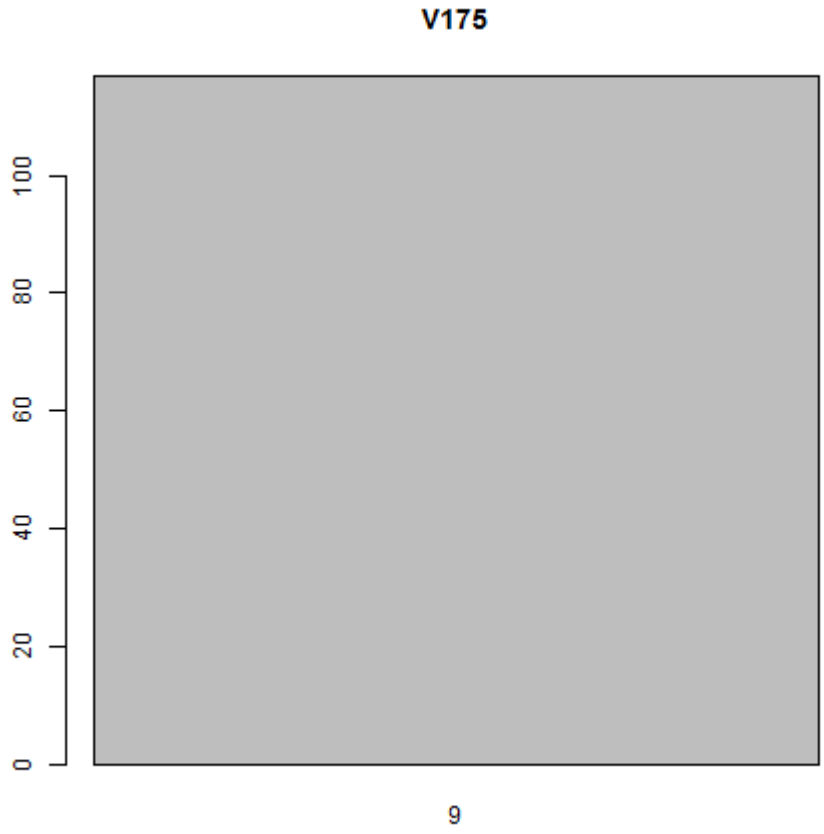

V182
0
2000
4000
6000
8000
CTUH
Ivza
SEuy

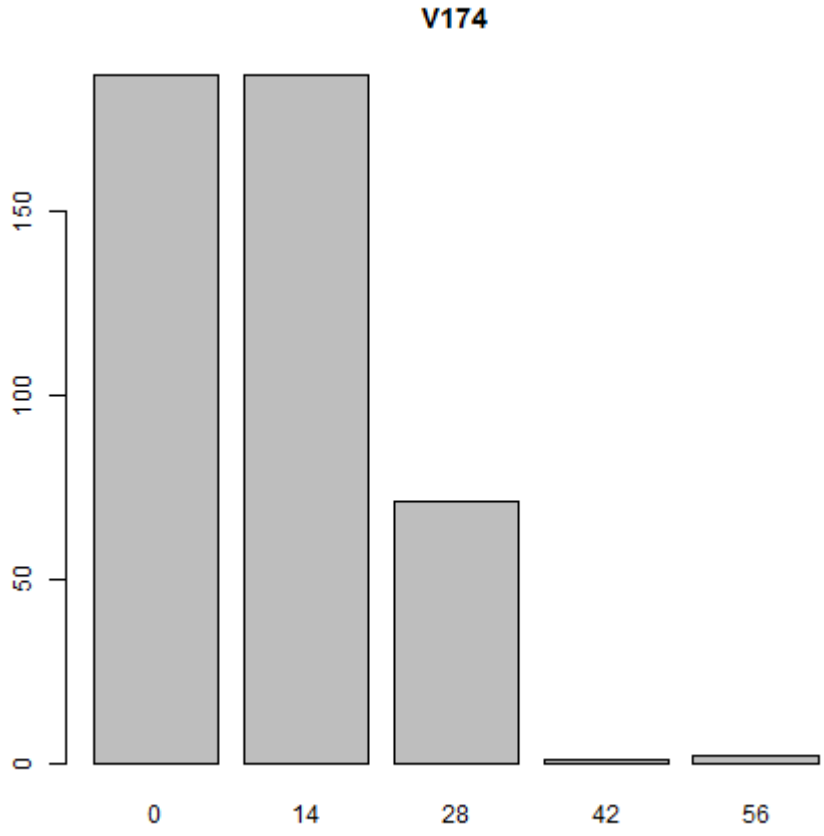

V180
Frequency
0
50
100
150
200
0
200
400
600
800
1000
1200
data[, colname]

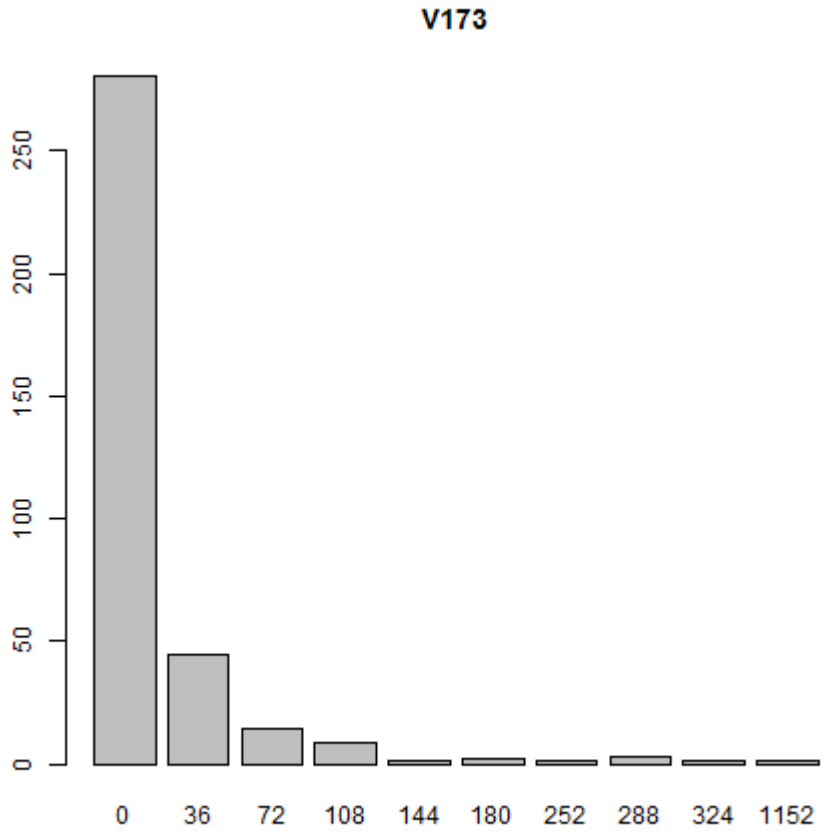

V179
0
100
200
300
400
500
600
0
24
48
72
96
132
252
396
468
1344

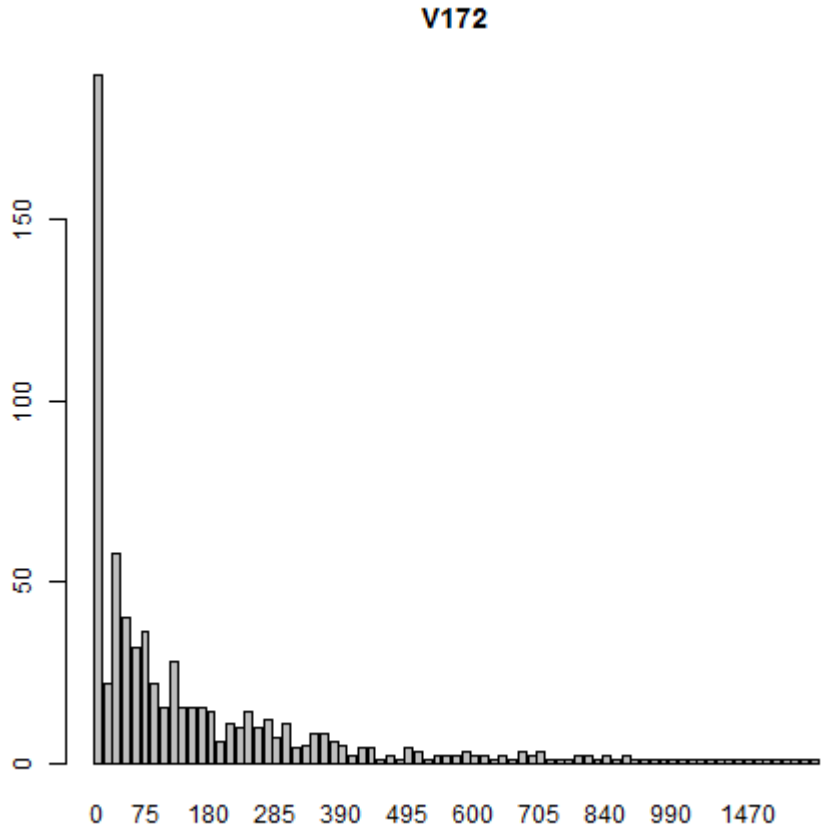

V178
0
20
40
60
80
0
1776222
5636988
10759161
19087089
27984390

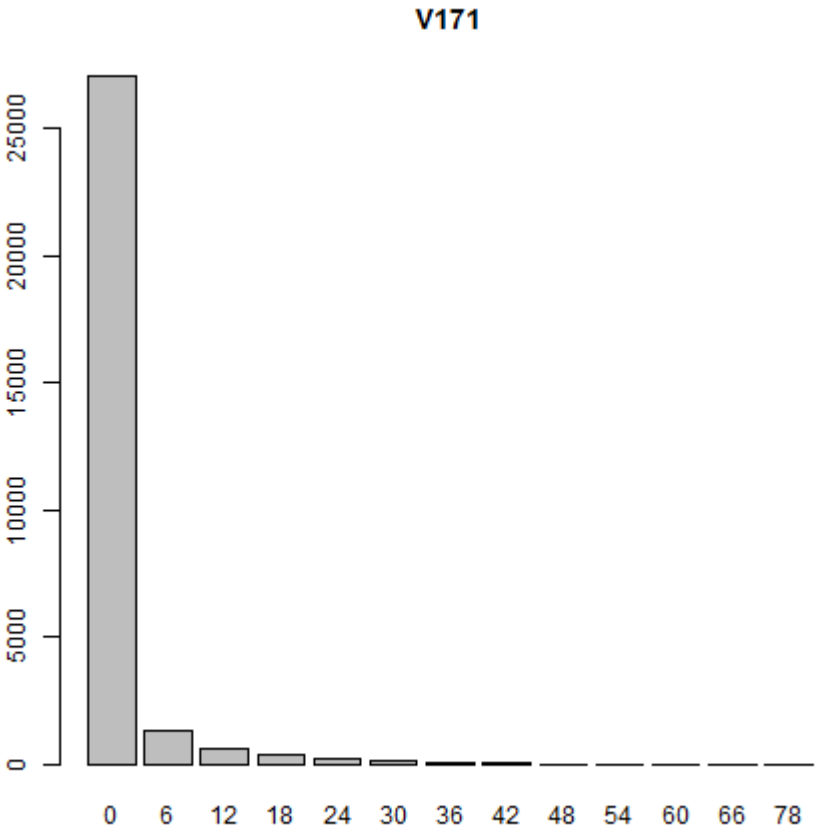

V176
Frequency
0
20
40
60
80
100
120
0
50000
100000
150000
200000
data[, colname]

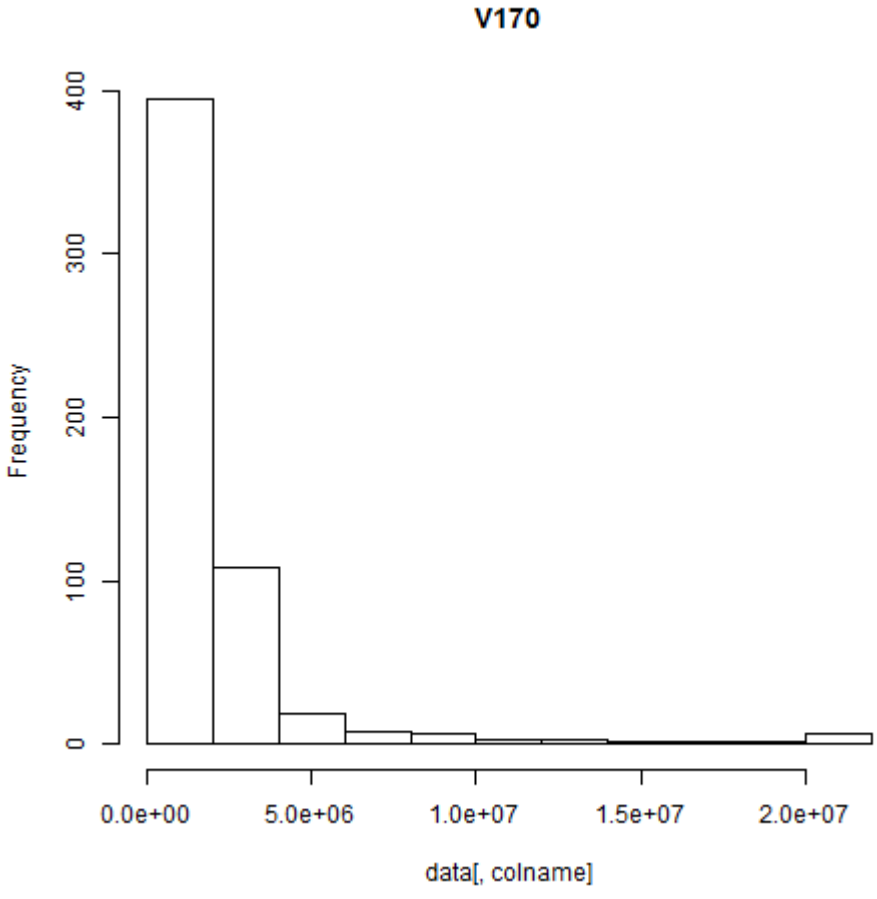

V175
0
20
40
60
80
100
9

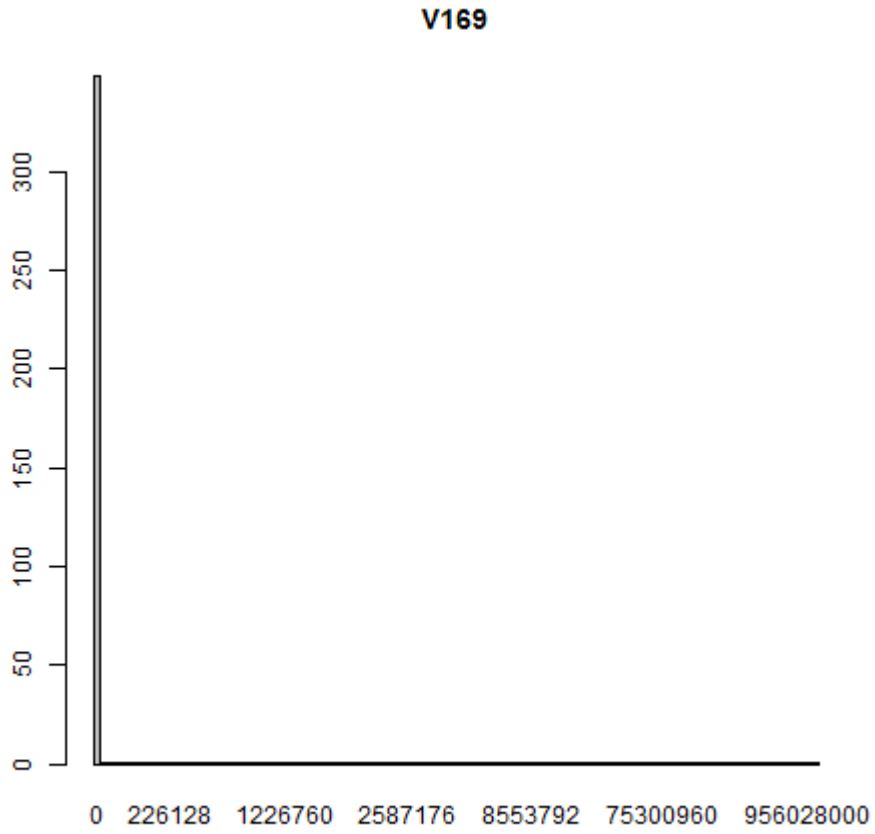

V174
0
50
100
150
0
14
28
42
56

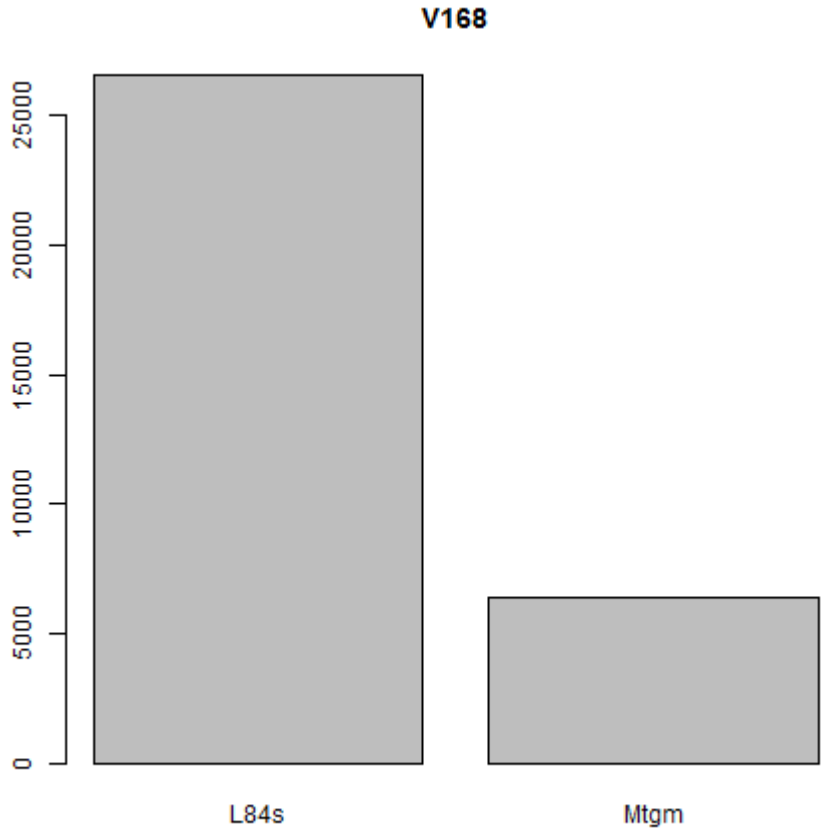

V173
0
50
100
150
200
250
0
36
72
108
144
180
252
288
324
1152

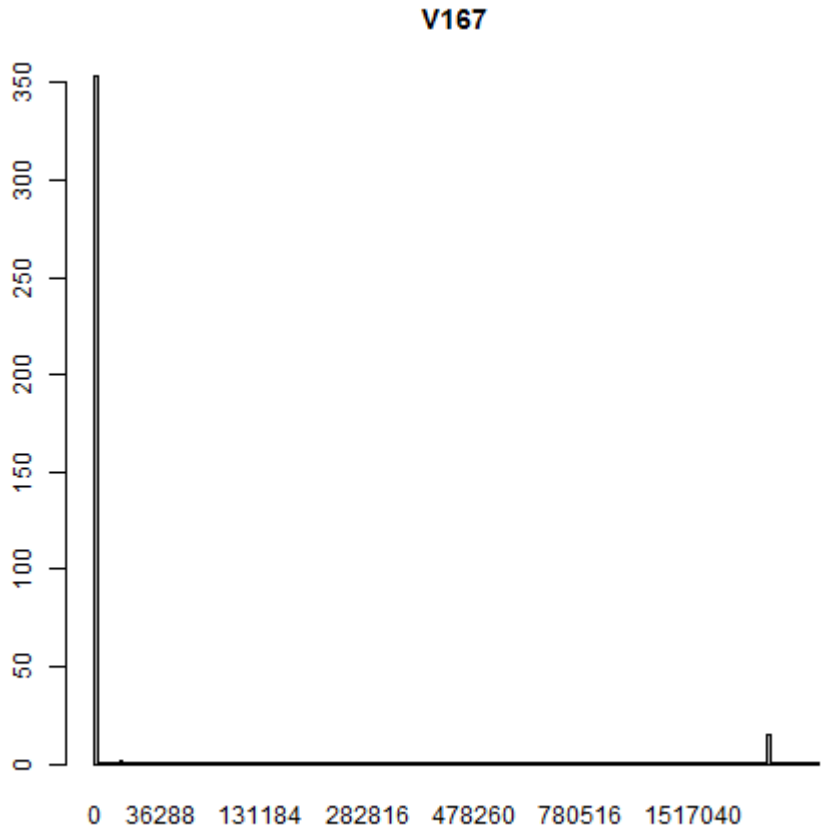

V172
0
50
100
150
0
75
180
285
390
495
600
705
840
990
1470

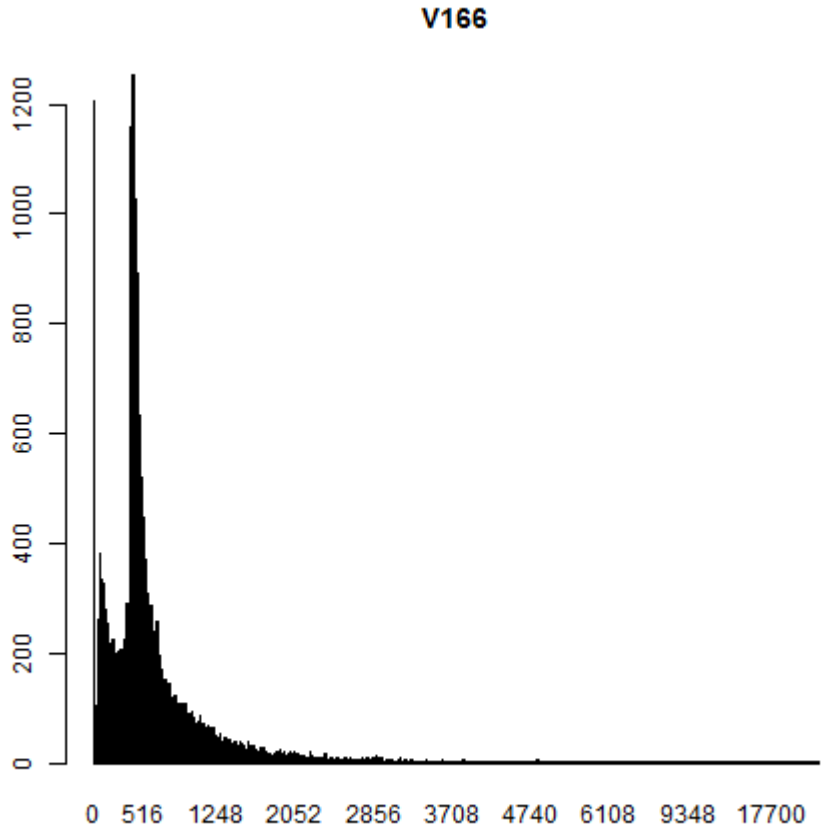

V171
25000
20000
15000
10000
5000
0
0
6
12
18
24
30
36
42
48
54
60
66
78

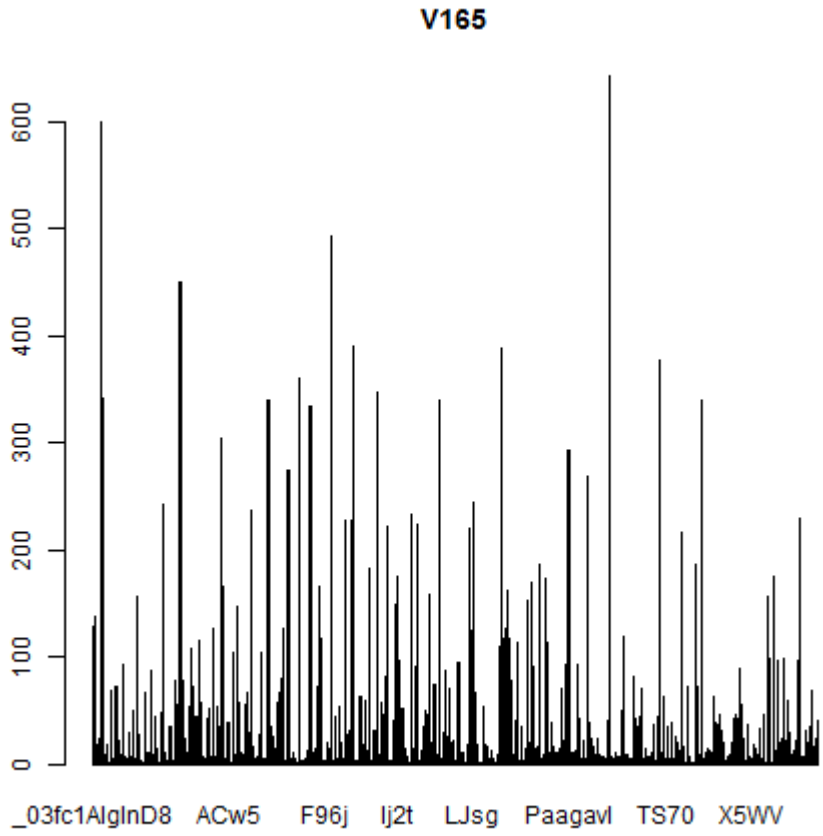

V170
400
300
200
100
0
Frequency
0.0e+00
5.0e+06
1.0e+07
1.5e+07
2.0e+07
data[, colname]

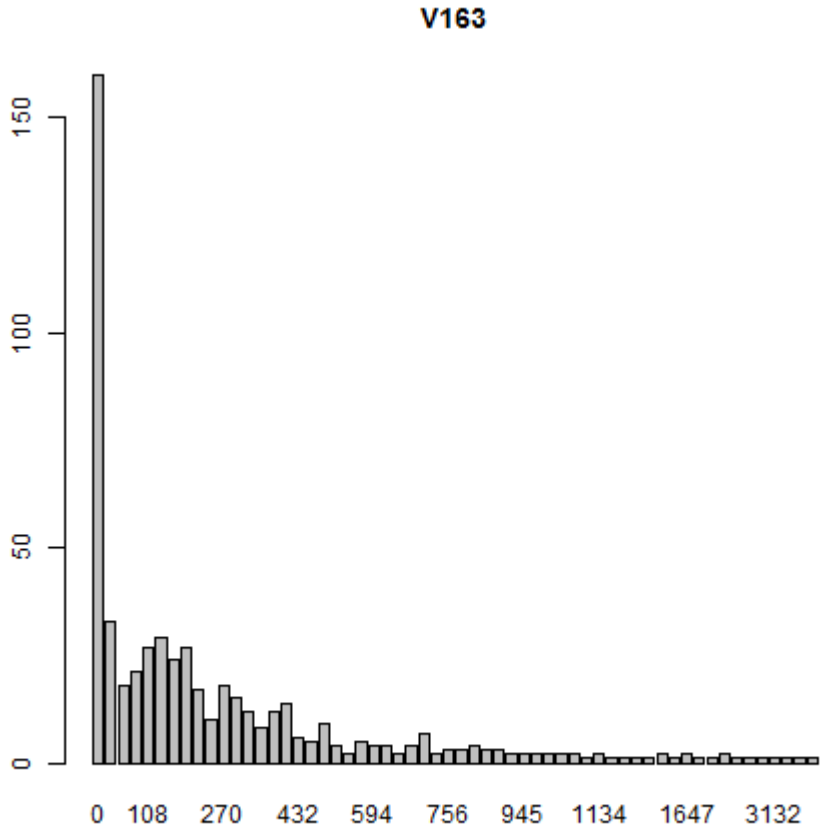

V169
0
50
100
150
200
250
300
0
226128
1226760
2587176
8553792
75300960
956028000

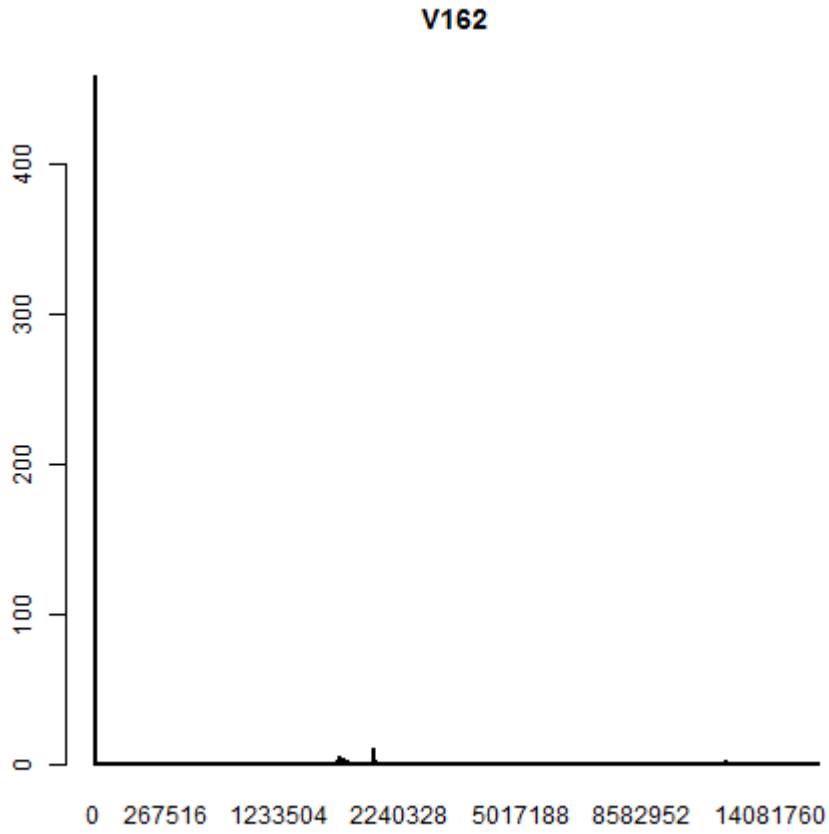

V168
0
5000
10000
15000
20000
25000
L84s
Mtgm

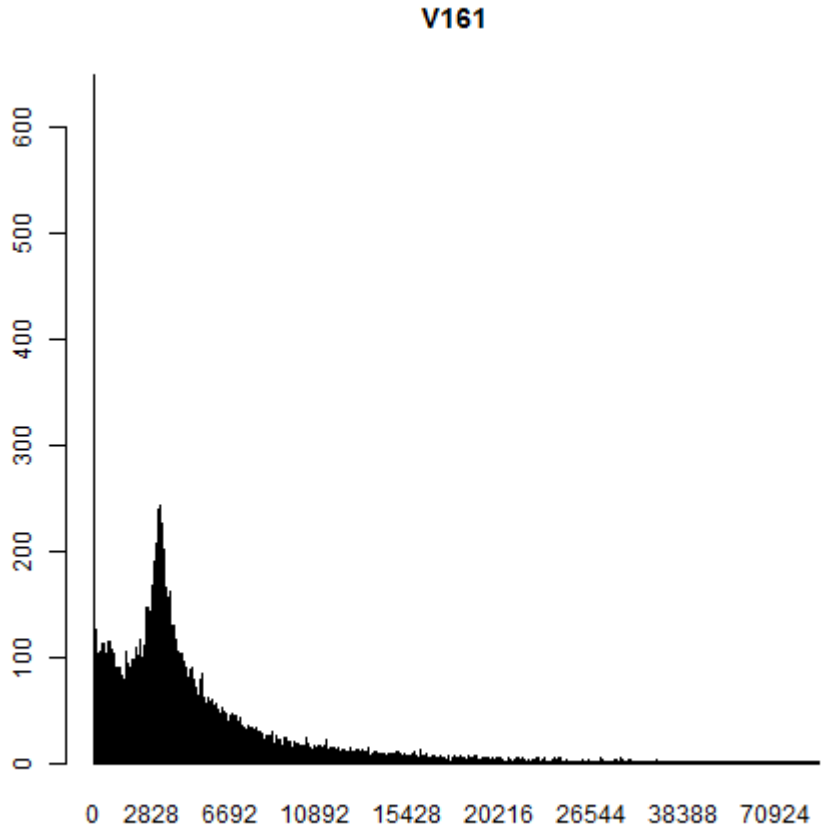

V167
0
50
100
150
200
250
300
350
0
36288
131184
282816
478260
780516
1517040

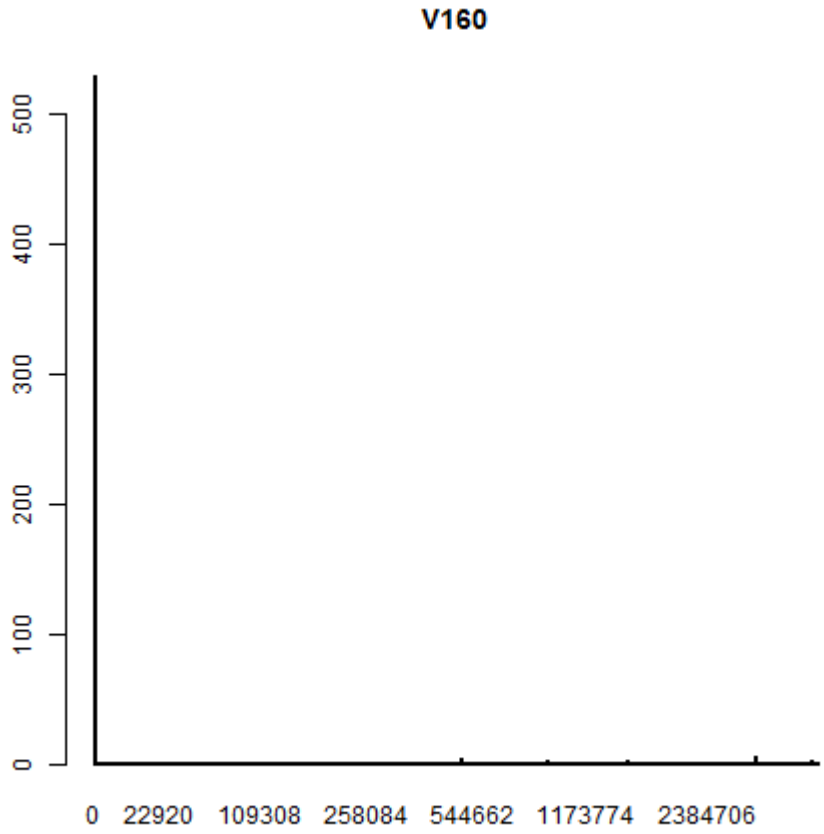

V166
0
200
400
600
800
1000
1200
0
516
1248
2052
2856
3708
4740
6108
9348
17700

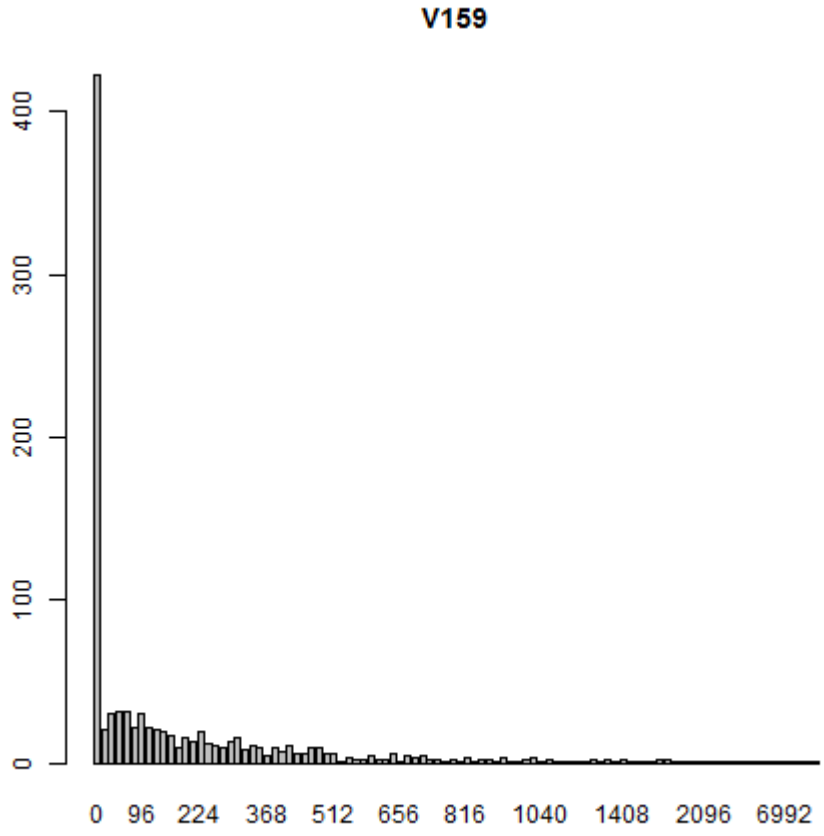

V165
0
100
200
300
400
500
600
_03fc1AlgInD8
ACw5
F96j
Ij2t
LJsg
Paagavl
TS70
X5WV

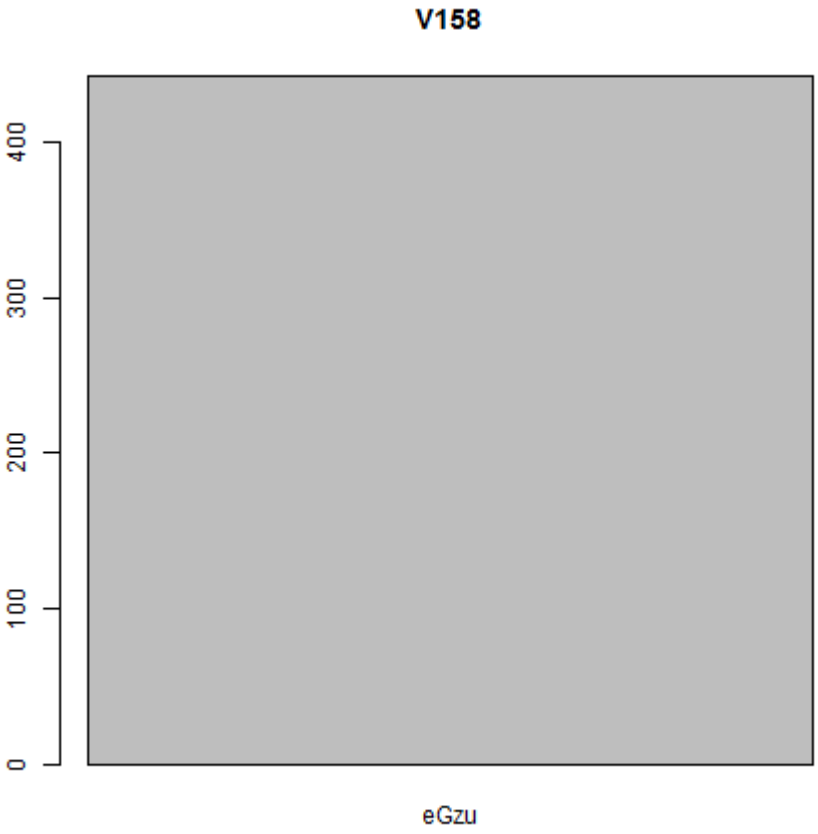

V163
0
50
100
150
0
108
270
432
594
756
945
1134
1647
3132

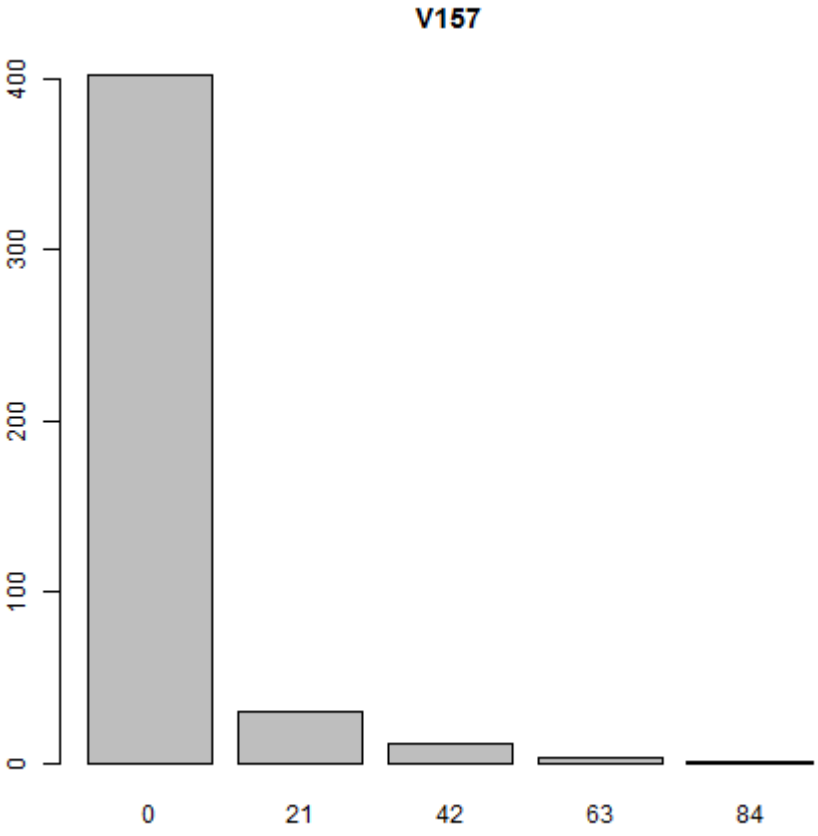

V162
0
100
200
300
400
0
267516
1233504
2240328
5017188
8582952
14081760

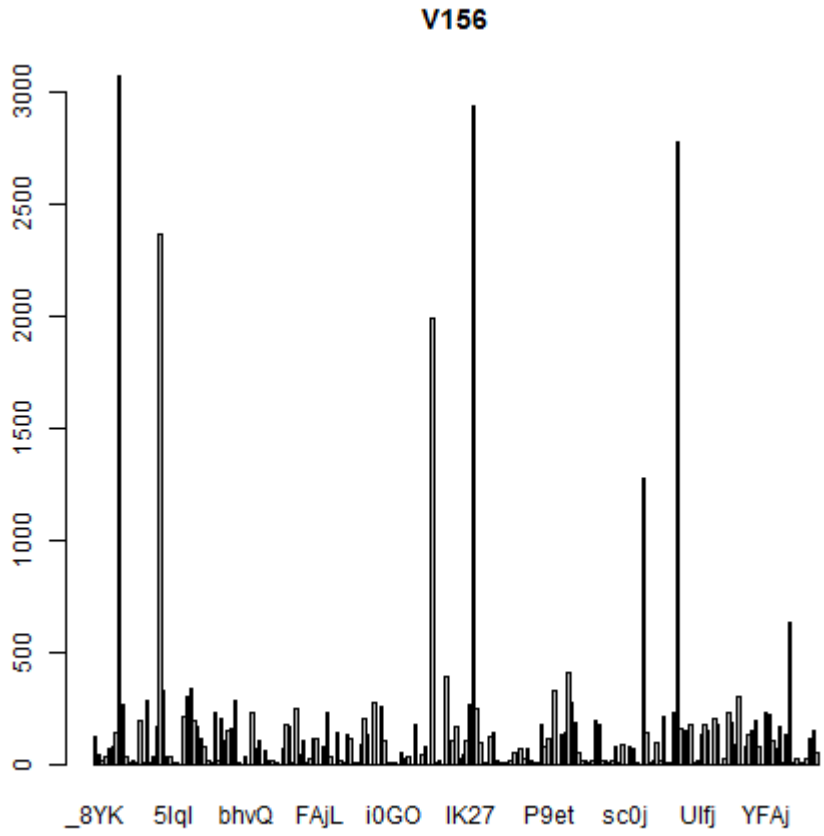

V161
0
100
200
300
400
500
600
0
2828
6692
10892
15428
20216
26544
38388
70924

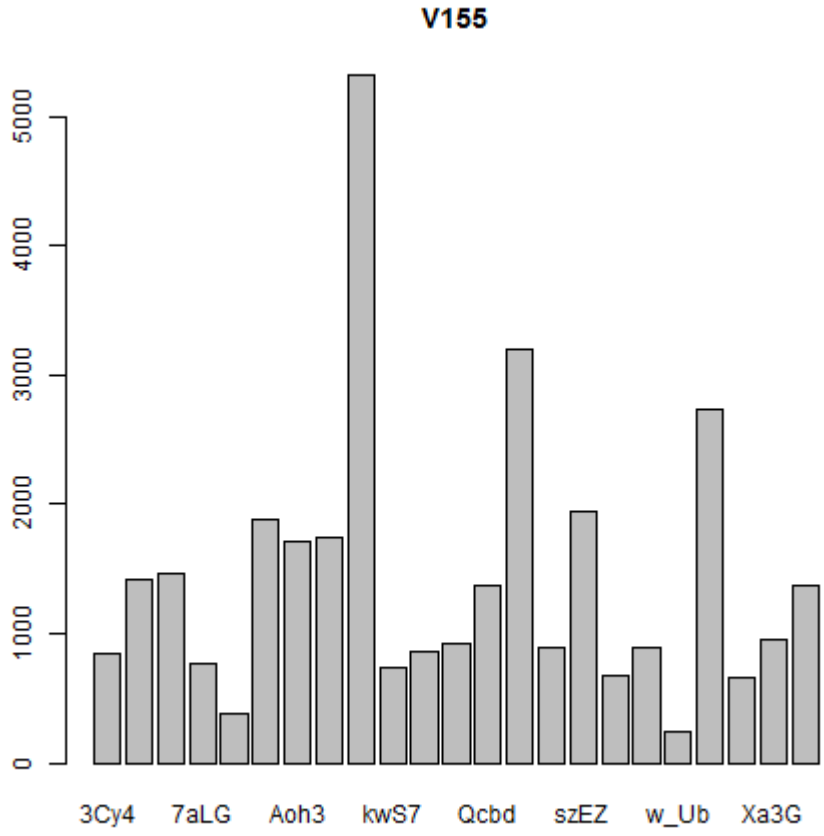

V160
0
100
200
300
400
500
0
22920
109308
258084
544662
1173774
2384706

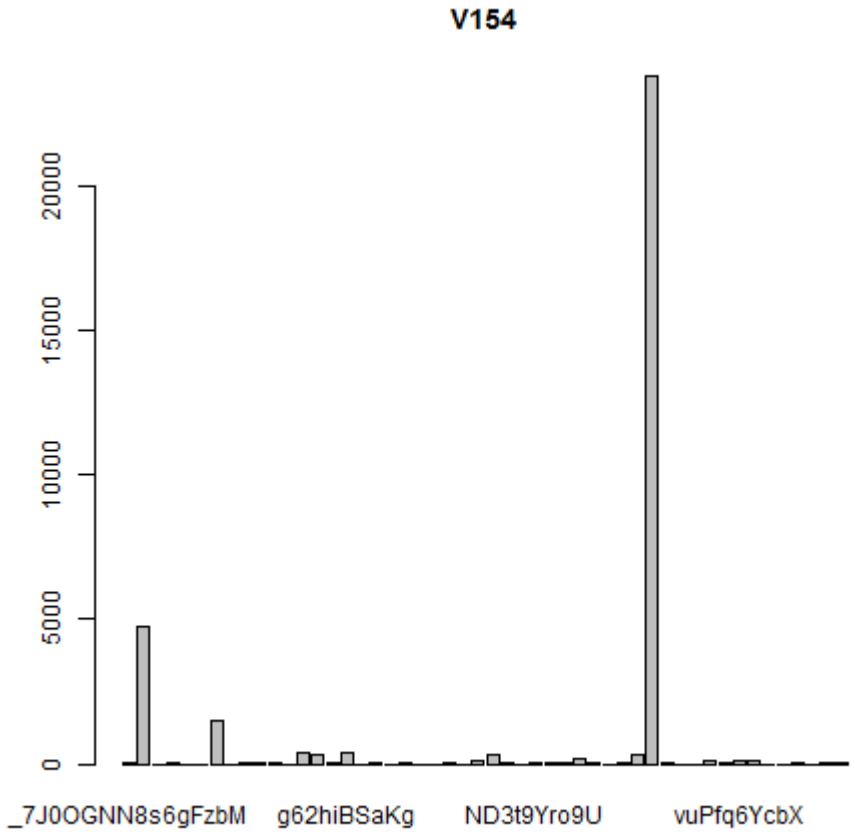

V159
0
100
200
300
400
0
96
224
368
512
656
816
1040
1408
2096
6992

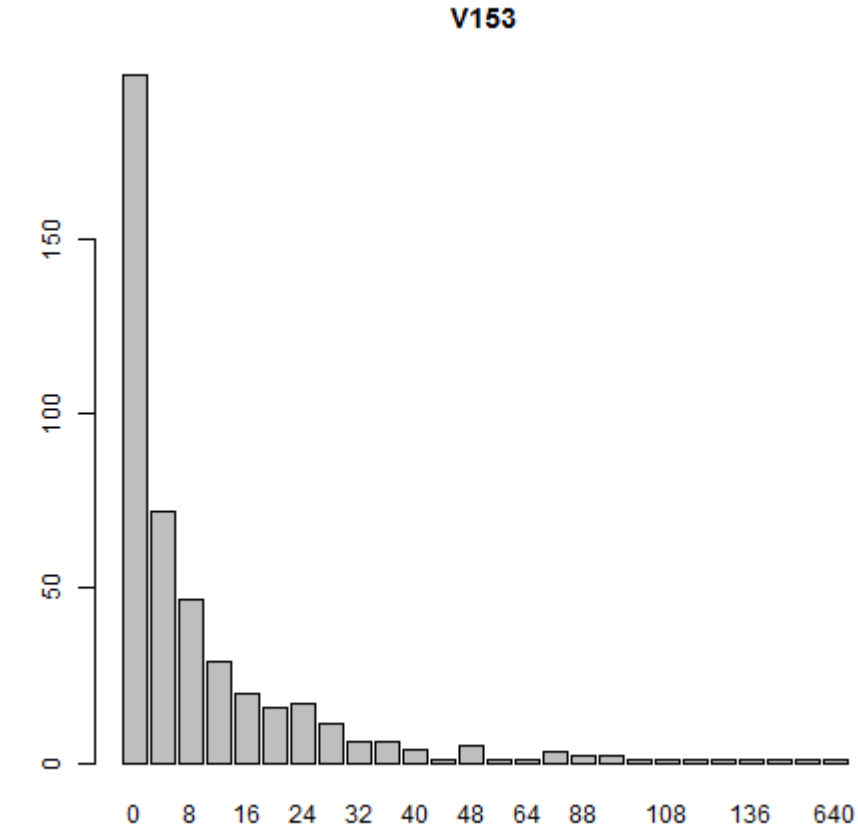

V158
400
300
200
100
0
eGzu

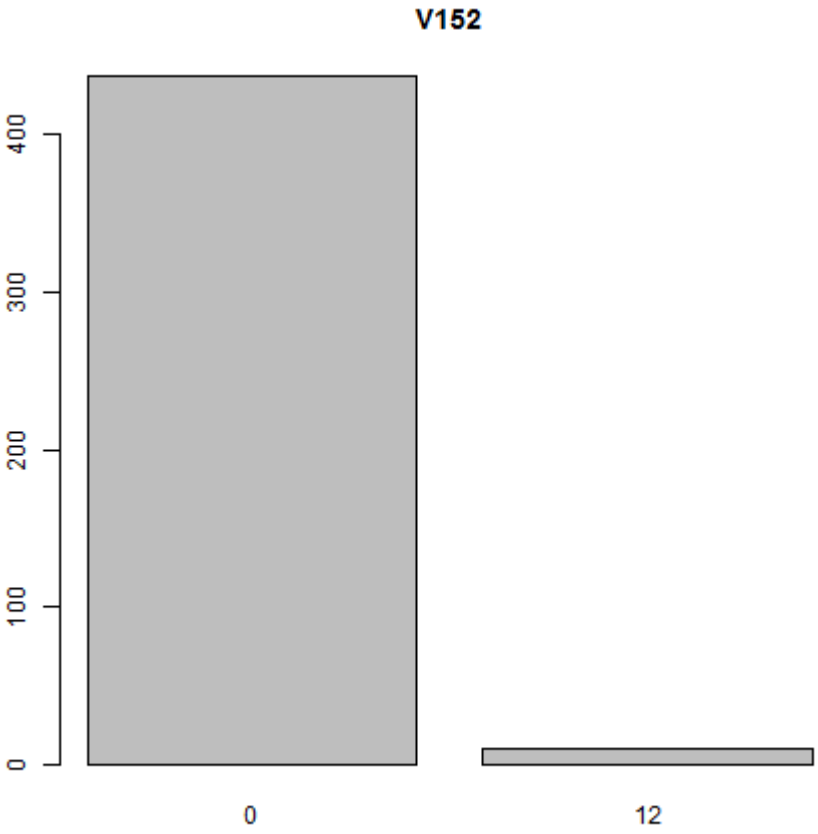

V157
400
300
200
100
0
0
21
42
63
84

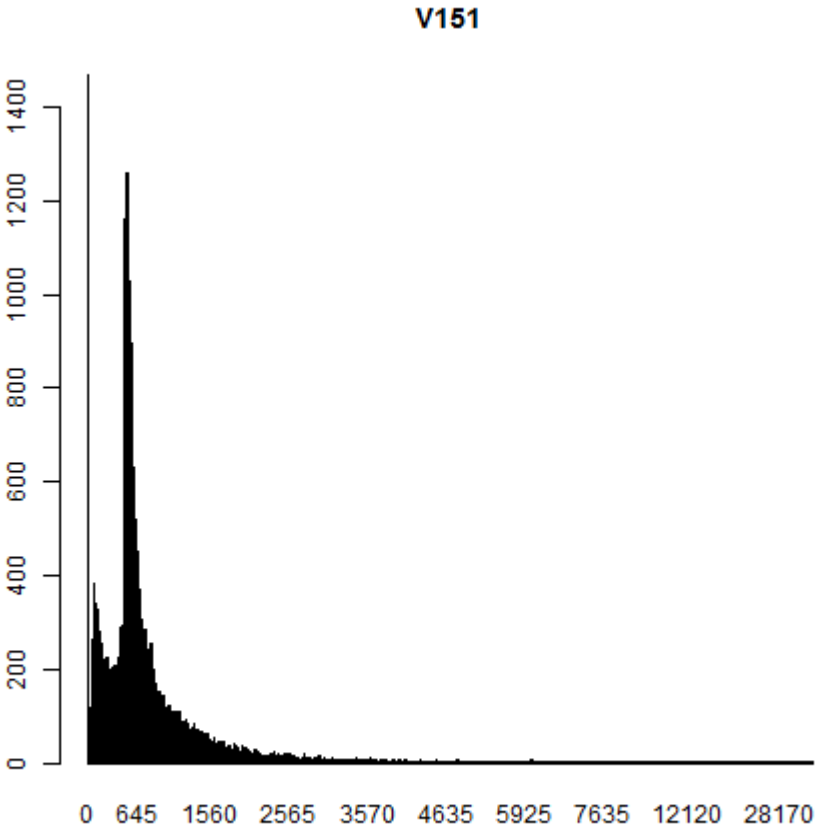

V156
0
500
1000
1500
2000
2500
3000
_8YK
5lql
bhvQ
FAjL
i0GO
lK27
P9et
sc0j
Ulfj
YFAj

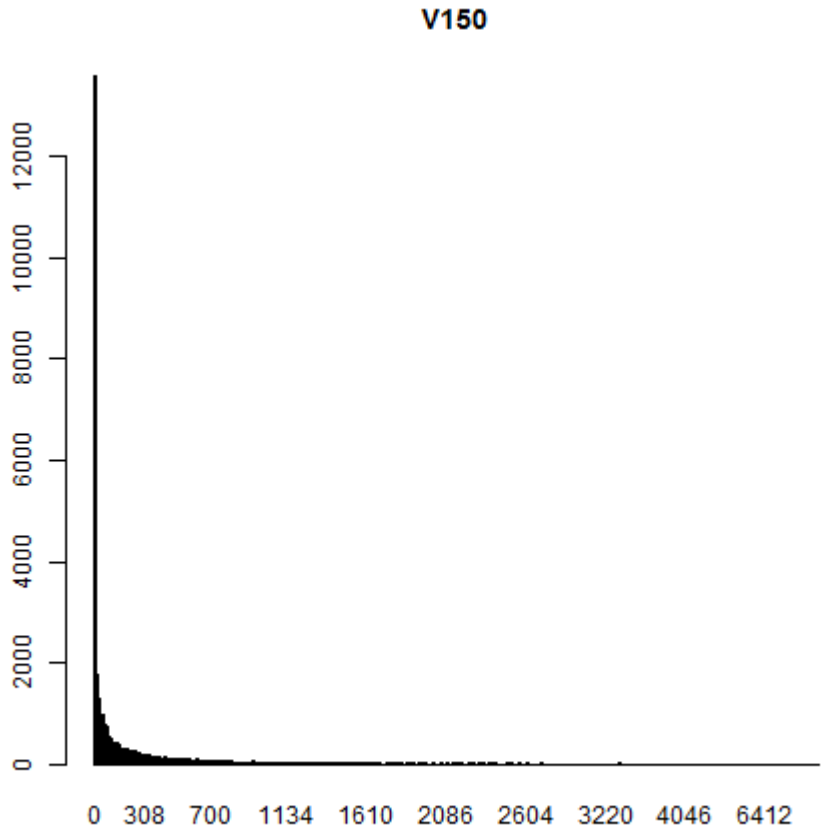

V155
0
1000
2000
3000
4000
5000
3Cy4
7aLG
Aoh3
kwS7
Qcbd
szEZ
w_Ub
Xa3G

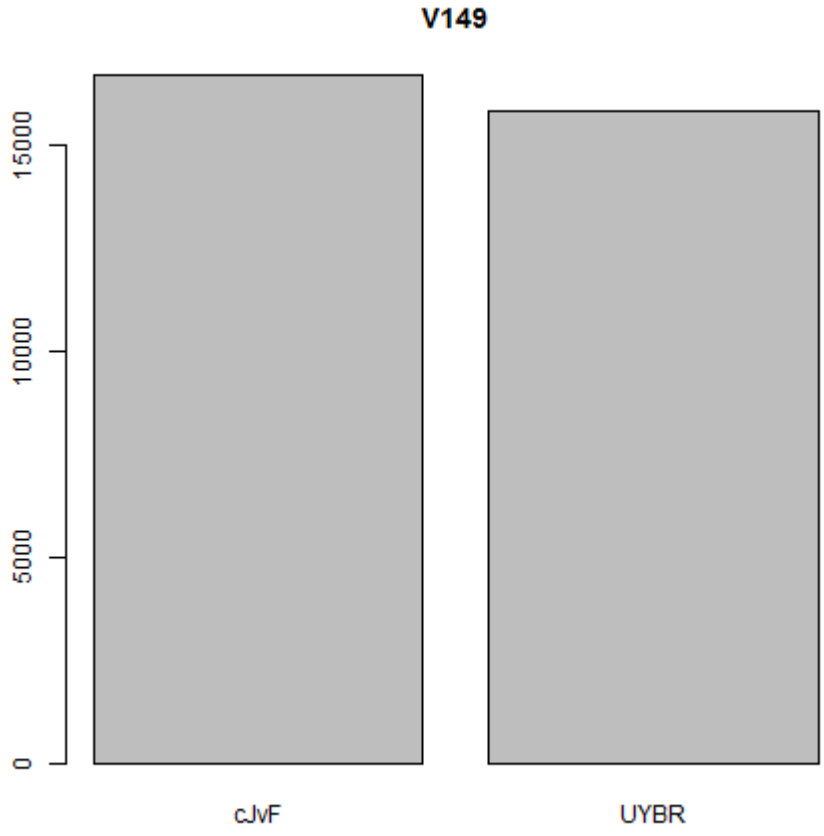

V154
20000
15000
10000
5000
0
_7J0OGNN8s6gFzbM
g62hiBSaKg
ND3t9Yro9U
vuPfq6YcbX

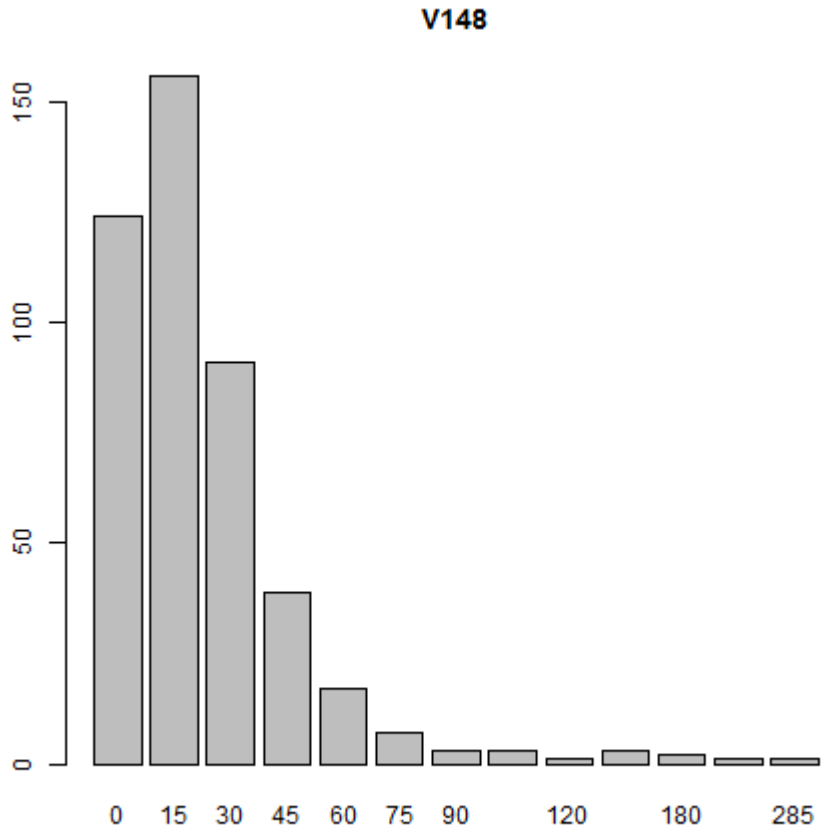

V153
150
100
50
0
0
8
16
24
32
40
48
64
88
108
136
640

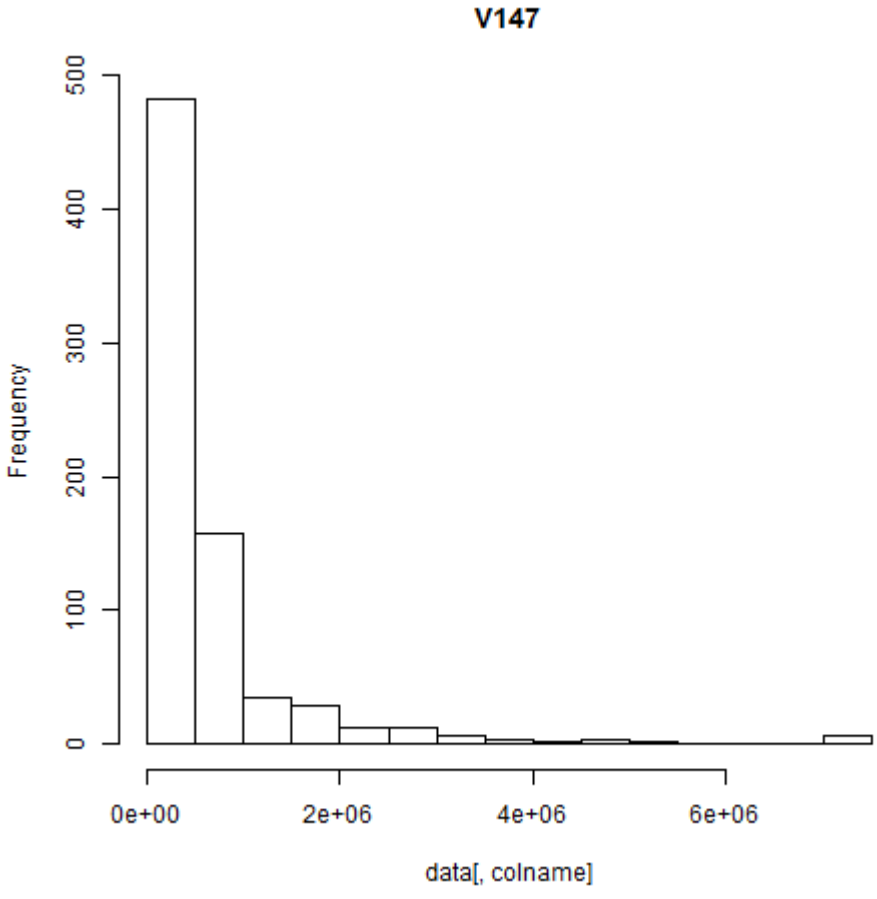

V152
400
300
200
100
0
0
12

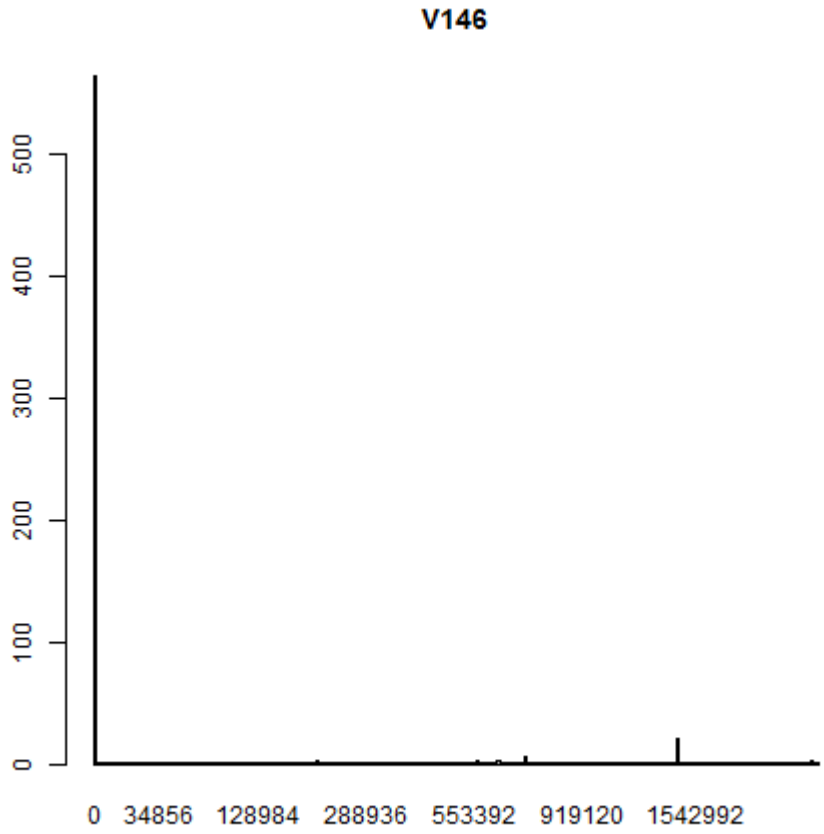

V151
1400
1200
1000
800
600
400
200
0
0
645
1560
2565
3570
4635
5925
7635
12120
28170

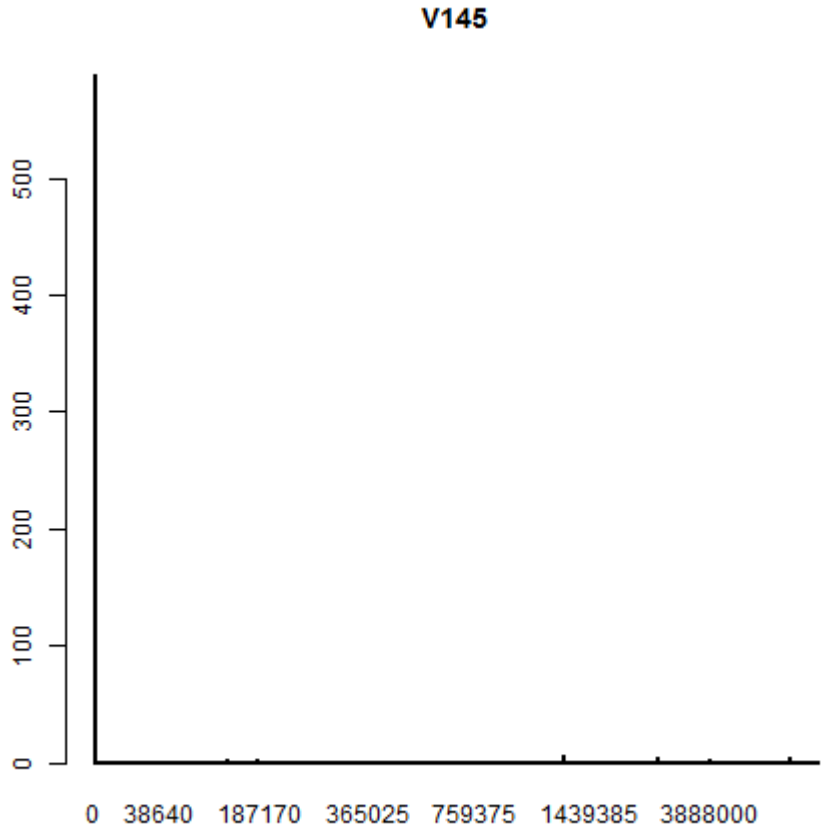

V150
0
2000
4000
6000
8000
10000
12000
0
308
700
1134
1610
2086
2604
3220
4046
6412

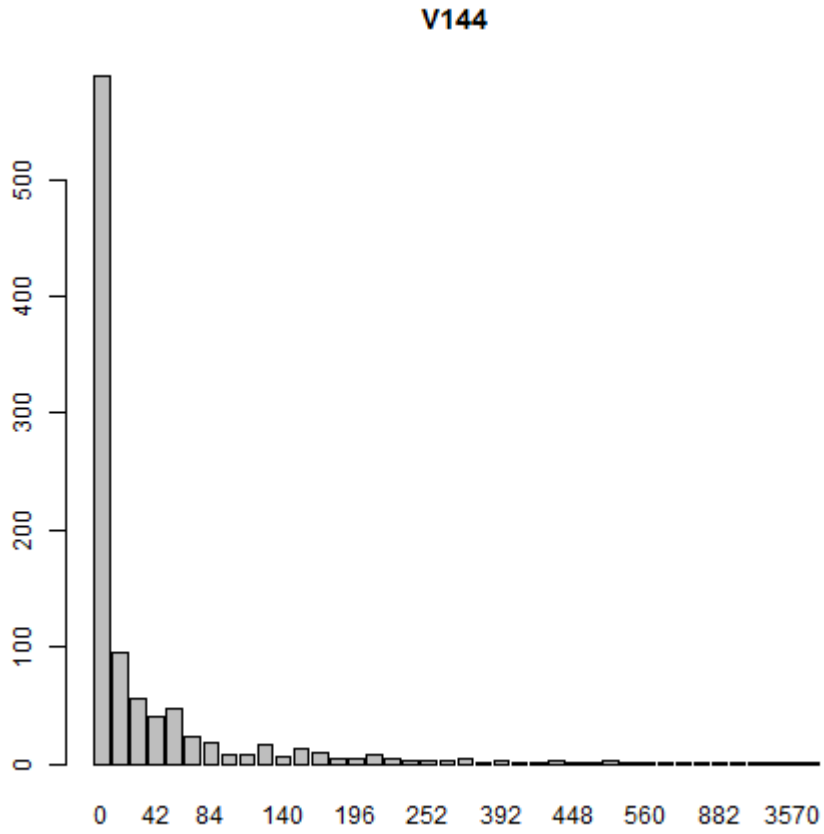

V149
0
5000
10000
15000
cJvF
UYBR

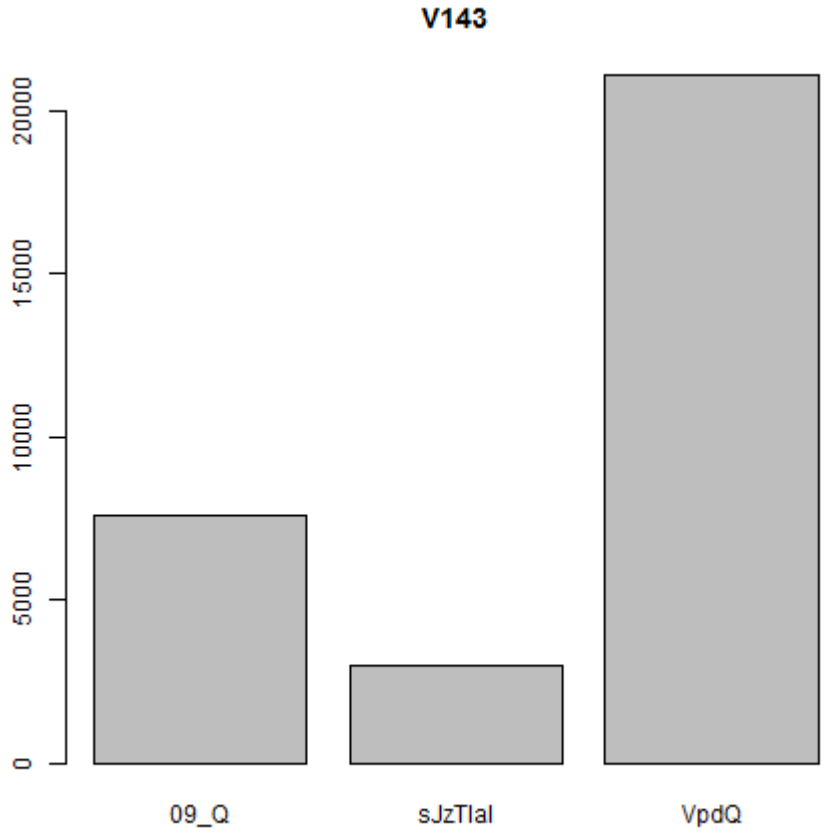

V148
150
100
50
0
0
15
30
45
60
75
90
120
180
285

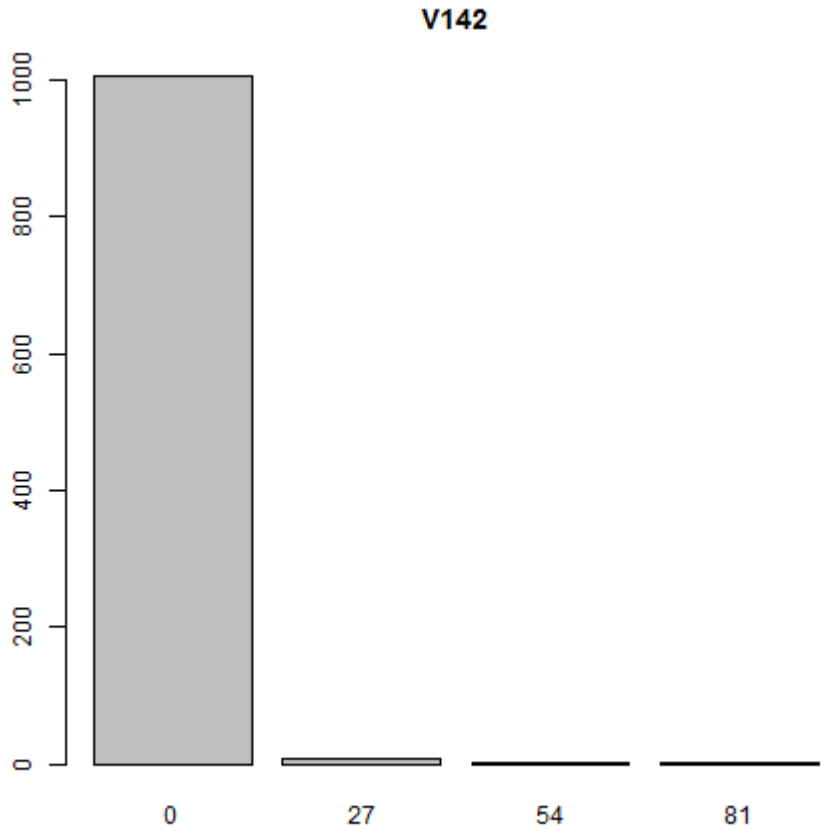

V147
Frequency
500
400
300
200
100
0
0e+00
2e+06
4e+06
6e+06
data[, colname]

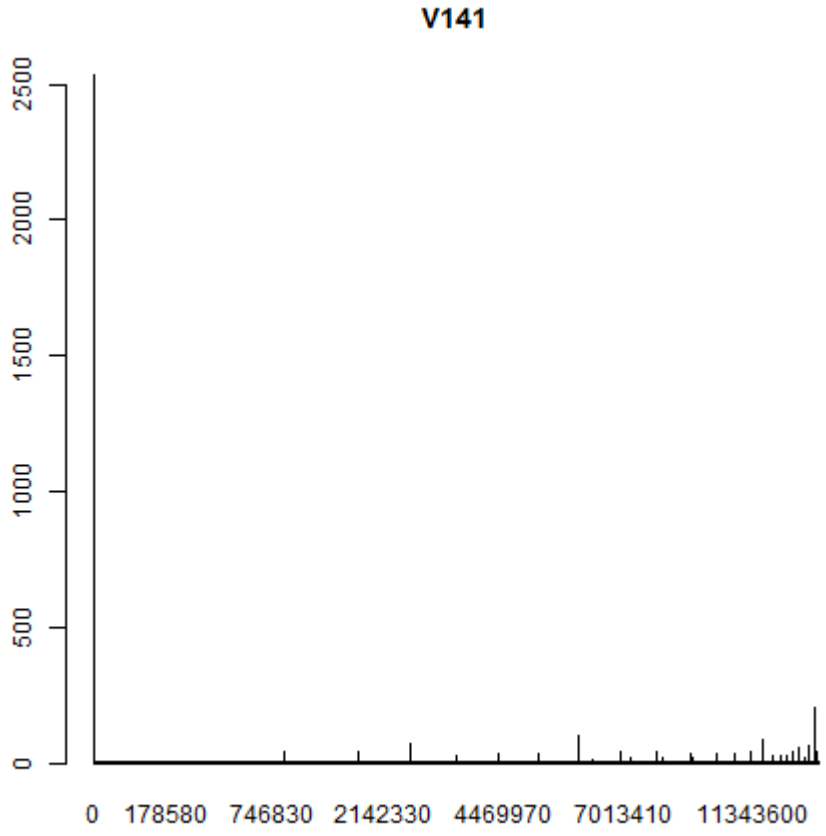

V146
0
100
200
300
400
500
0
34856
128984
288936
553392
919120
1542992

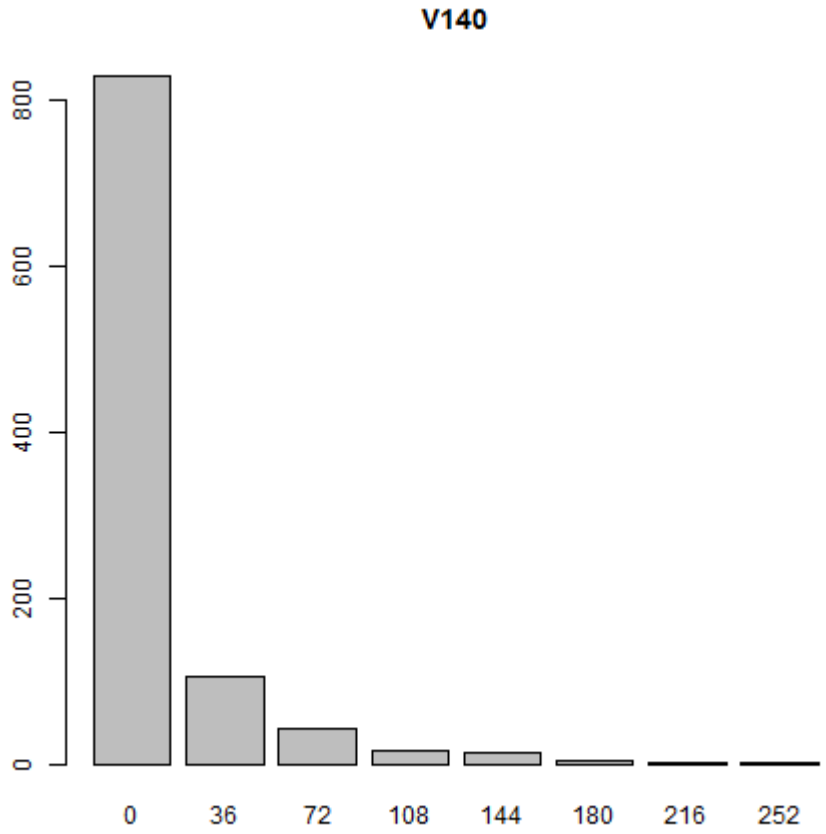

V145
0
100
200
300
400
500
0
38640
187170
365025
759375
1439385
3888000

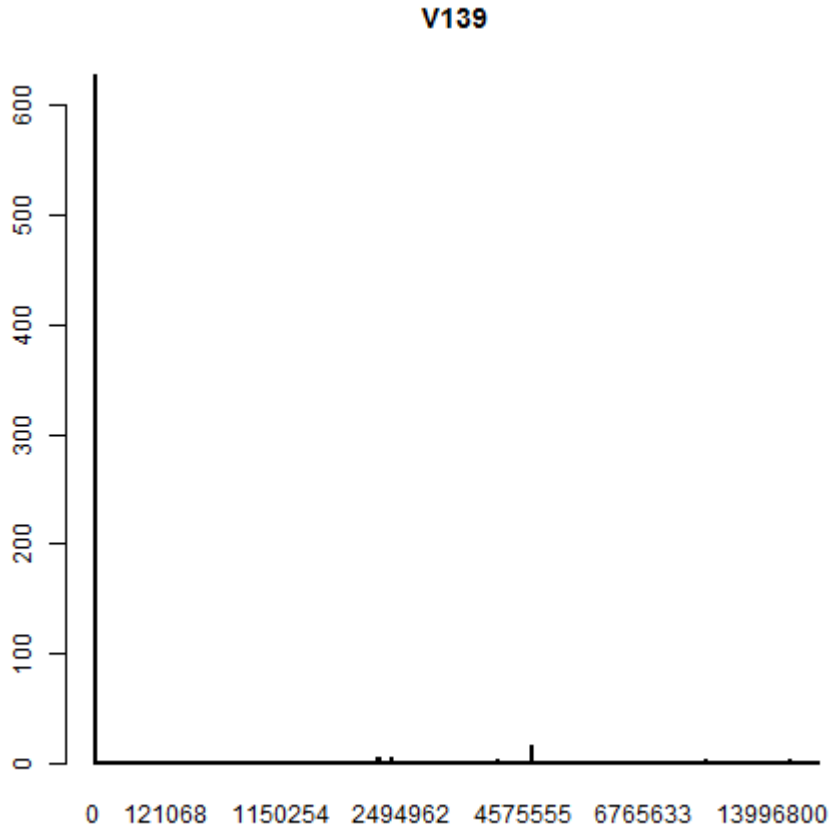

V144
0
100
200
300
400
500
0
42
84
140
196
252
392
448
560
882
3570

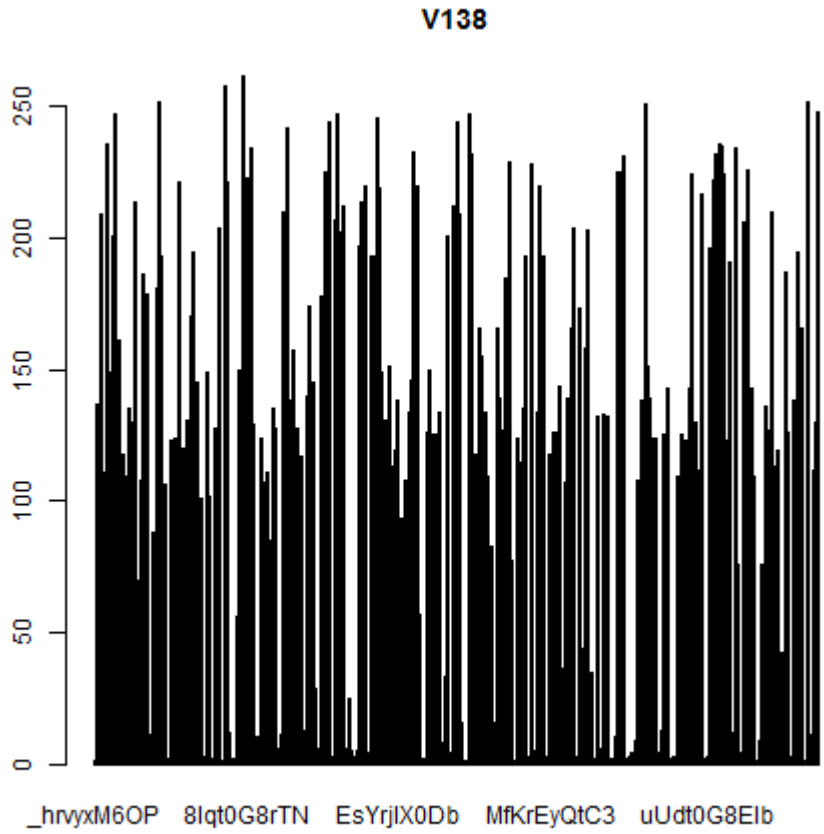

V143
0
5000
10000
15000
20000
09_Q
sJzTlal
VpdQ

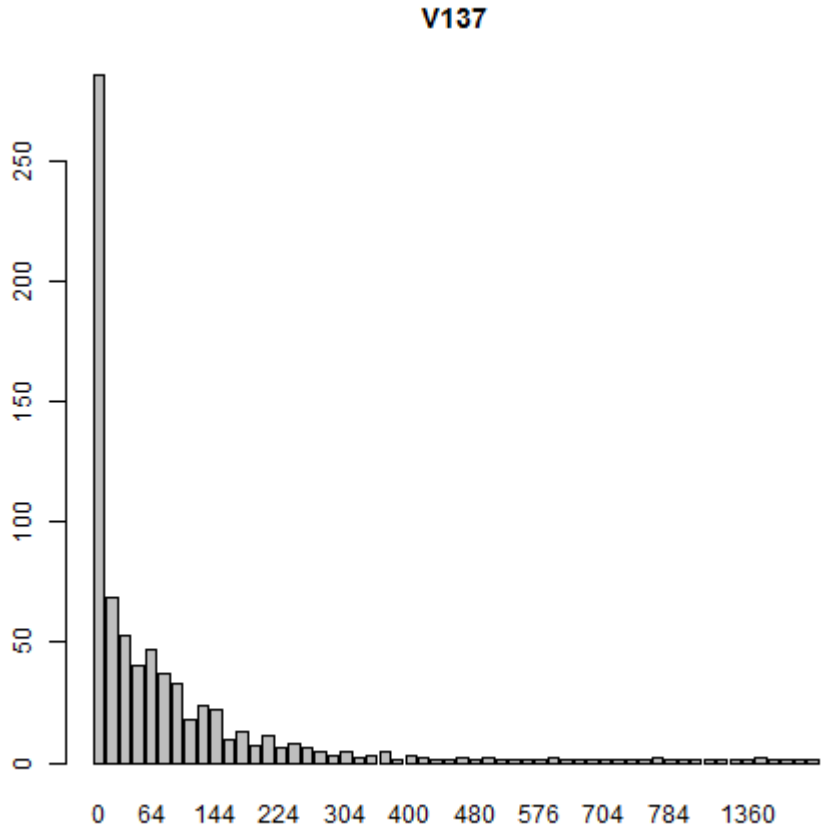

V142
1000
800
600
400
200
0
0
27
54
81

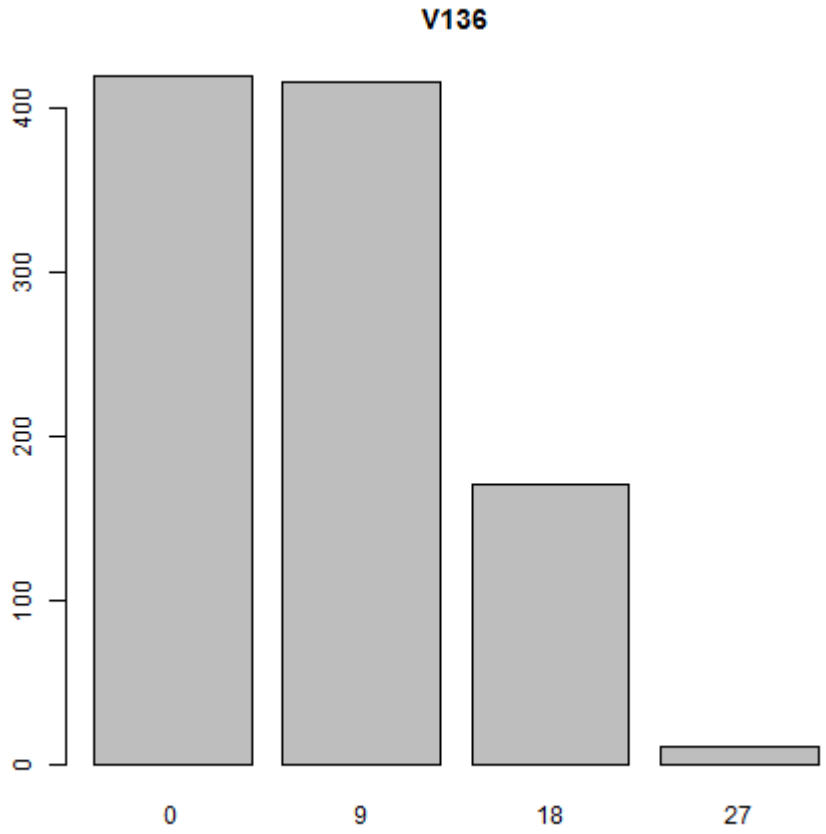

V141
2500
2000
1500
1000
500
0
0
178580
746830
2142330
4469970
7013410
11343600

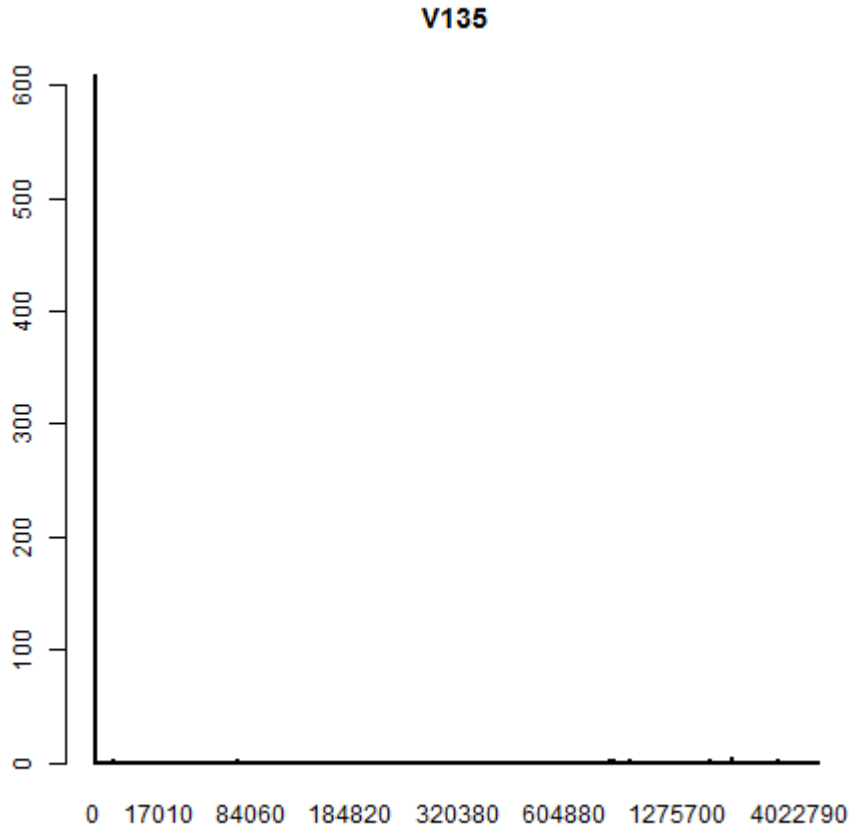

V140
800
600
400
200
0
0
36
72
108
144
180
216
252

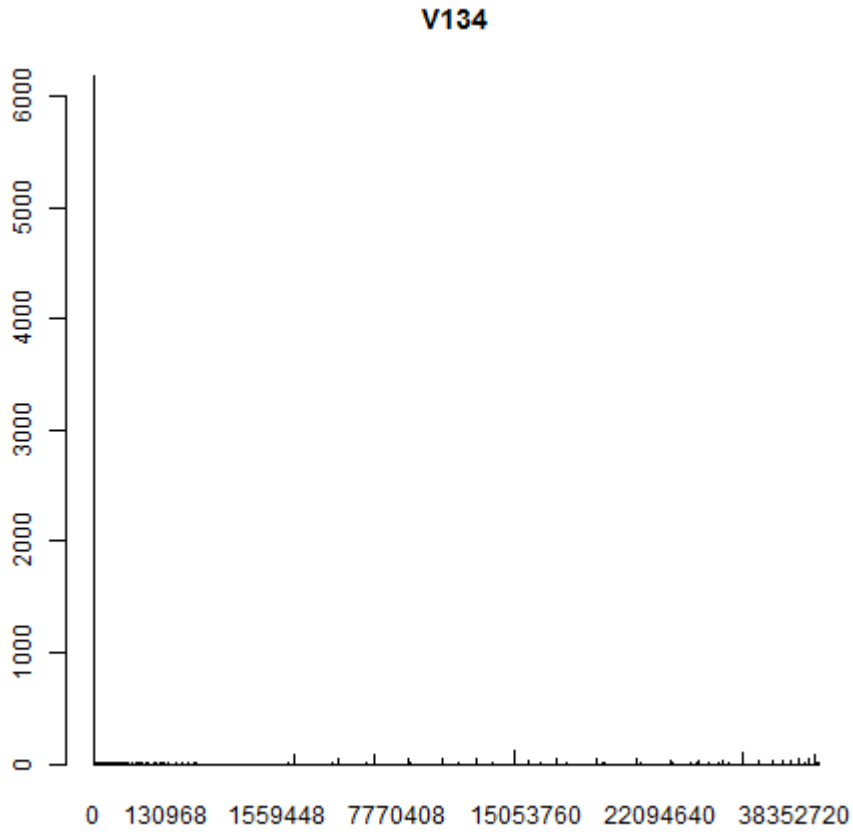

V139
600
500
400
300
200
100
0
0
121068
1150254
2494962
4575555
6765633
13996800

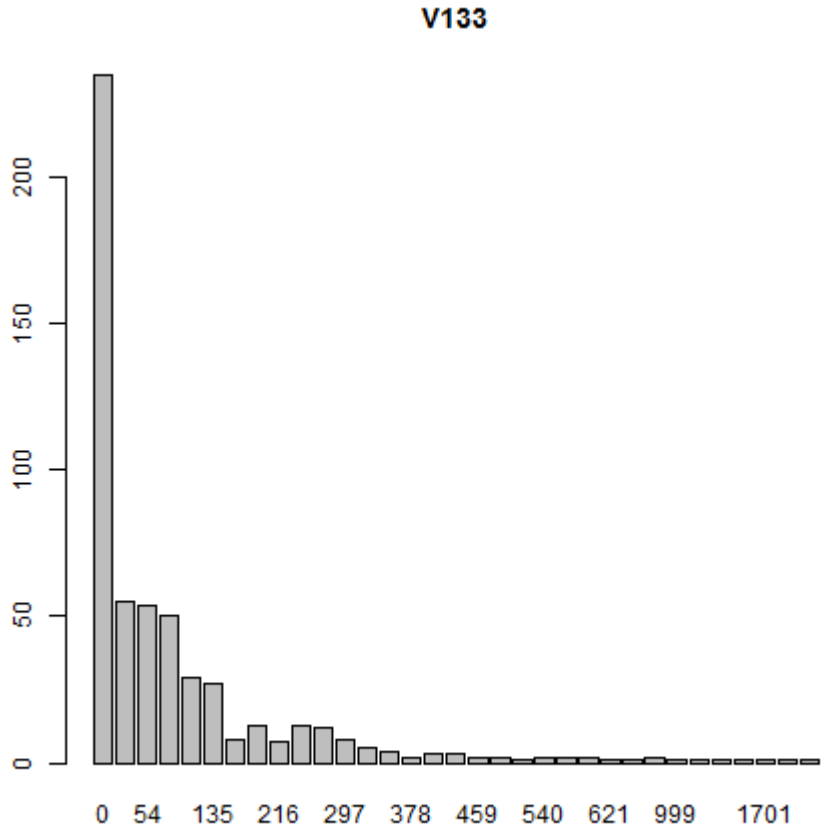

V138
0
50
100
150
200
250
_hrvyxM6OP
8Iqt0G8rTN
EsYrjlX0Db
MfKrEyQtC3
uUdt0G8Elb

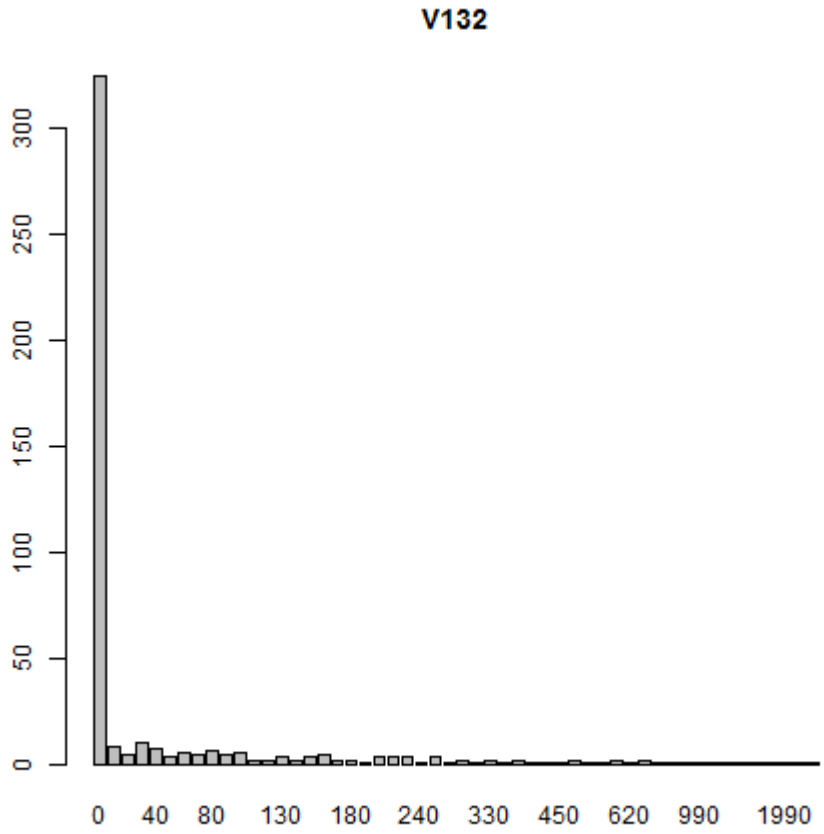

V137
0
50
100
150
200
250
0
64
144
224
304
400
480
576
704
784
1360

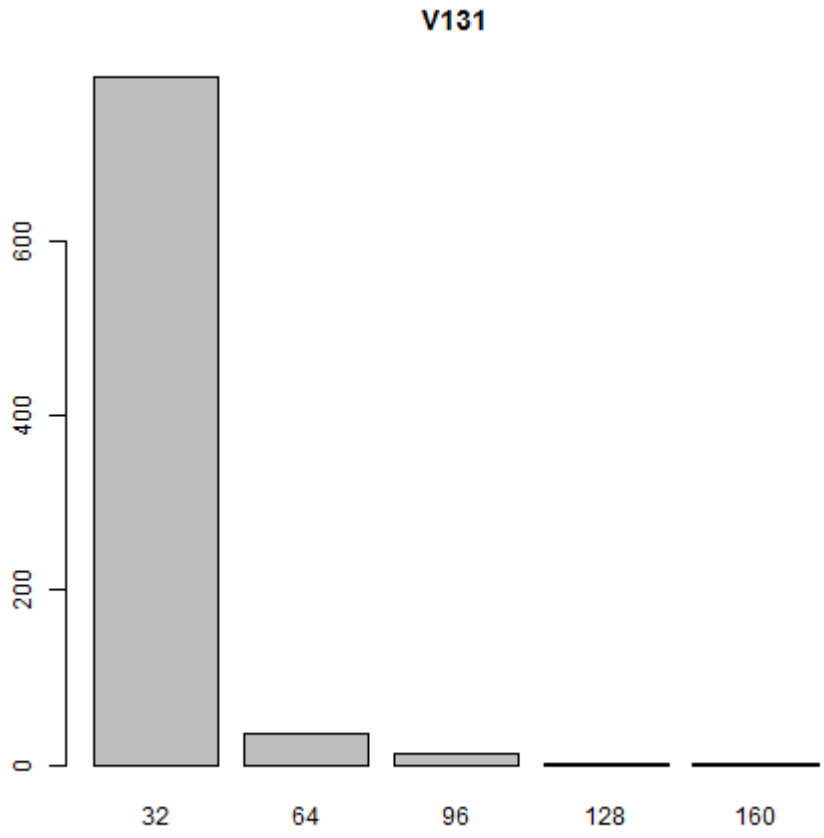

V136
400
300
200
100
0
0
9
18
27

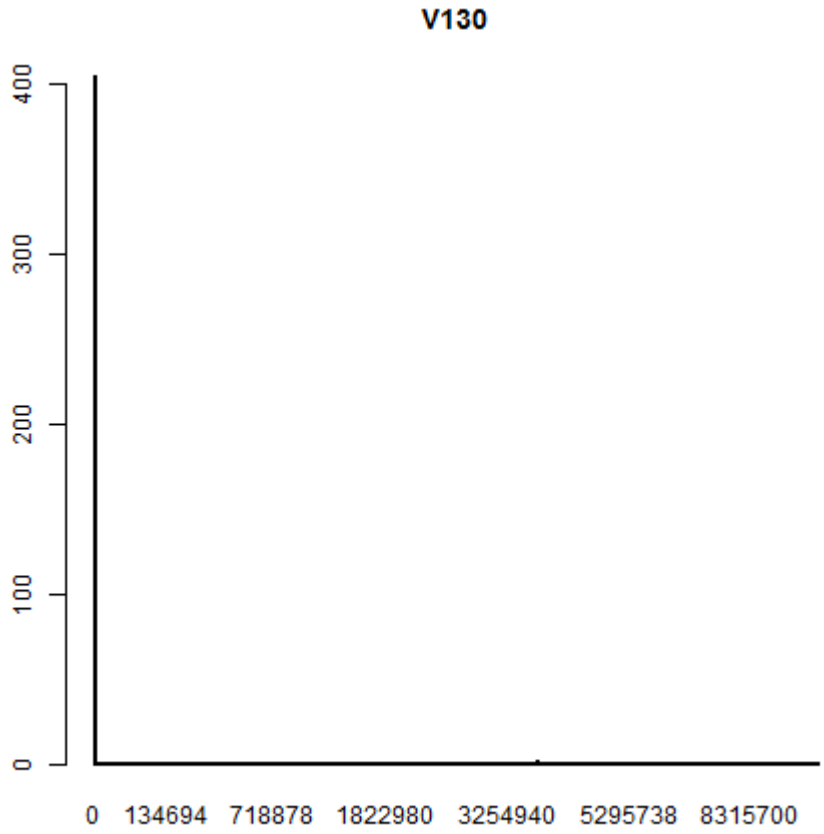

V135
600
500
400
300
200
100
0
0
17010
84060
184820
320380
604880
1275700
4022790

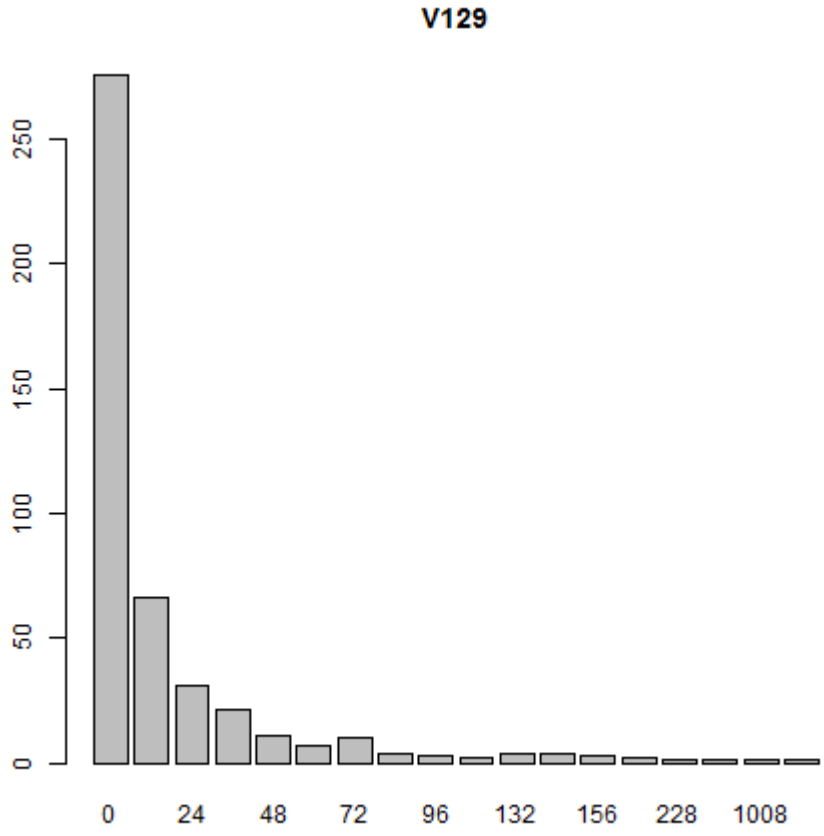

V134
6000
5000
4000
3000
2000
1000
0
0
130968
1559448
7770408
15053760
22094640
38352720

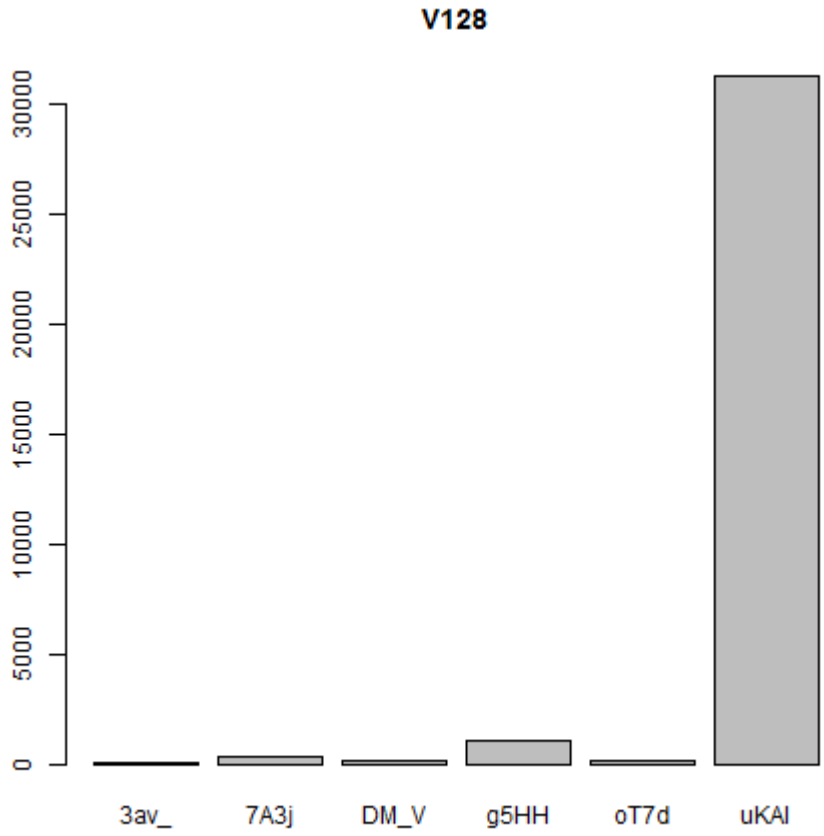

V133
200
150
100
50
0
0
54
135
216
297
378
459
540
621
999
1701

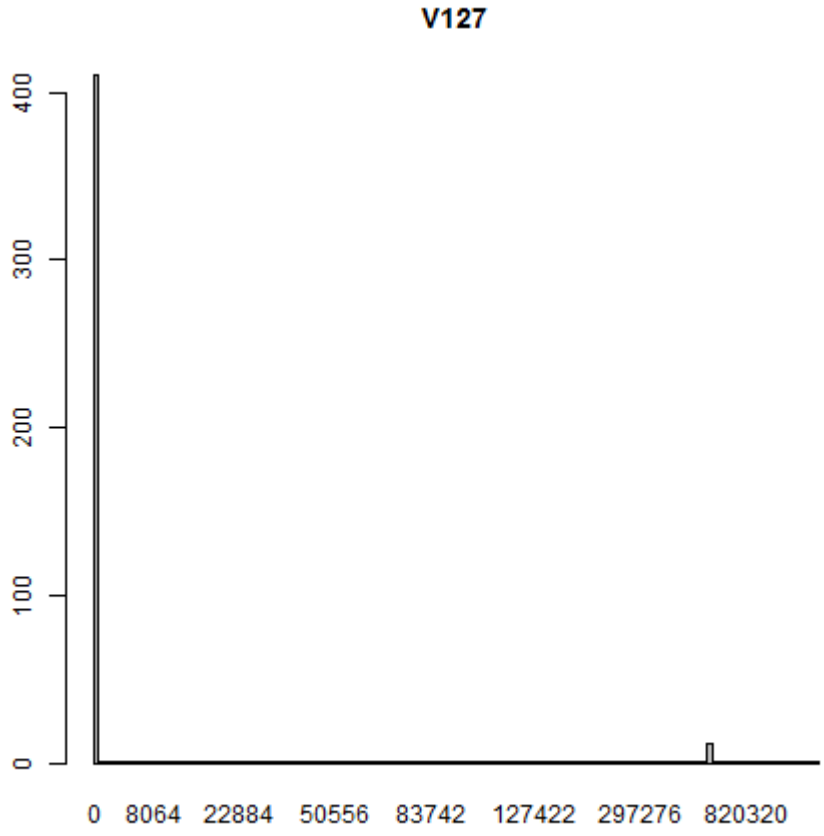

V132
0
50
100
150
200
250
300
0
40
80
130
180
240
330
450
620
990
1990

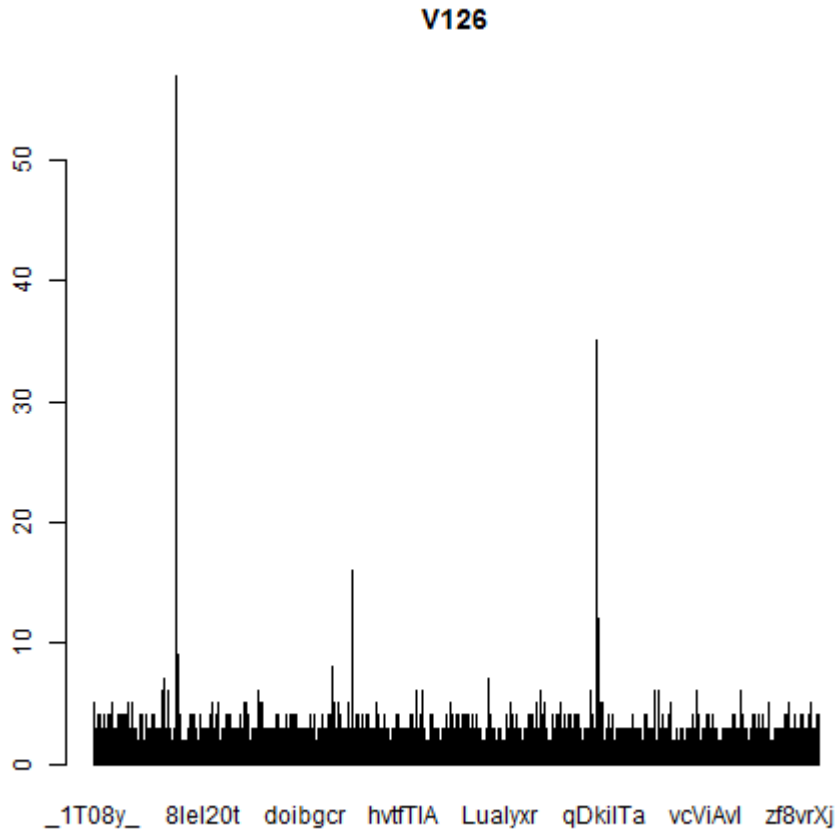

V131
0
200
400
600
32
64
96
128
160

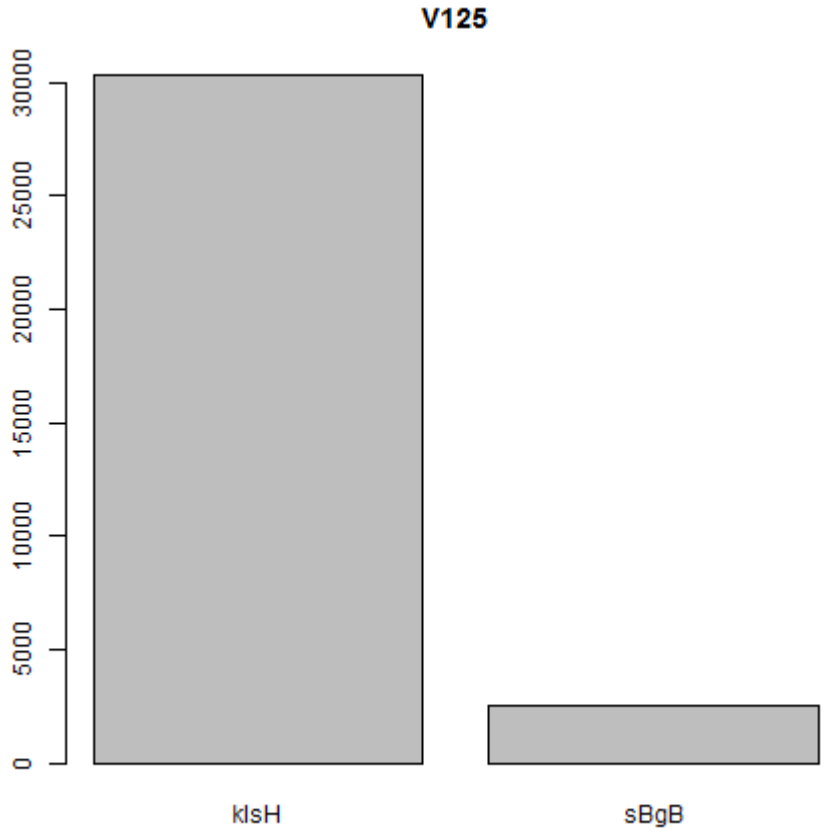

V130
400
300
200
100
0
0
134694
718878
1822980
3254940
5295738
8315700

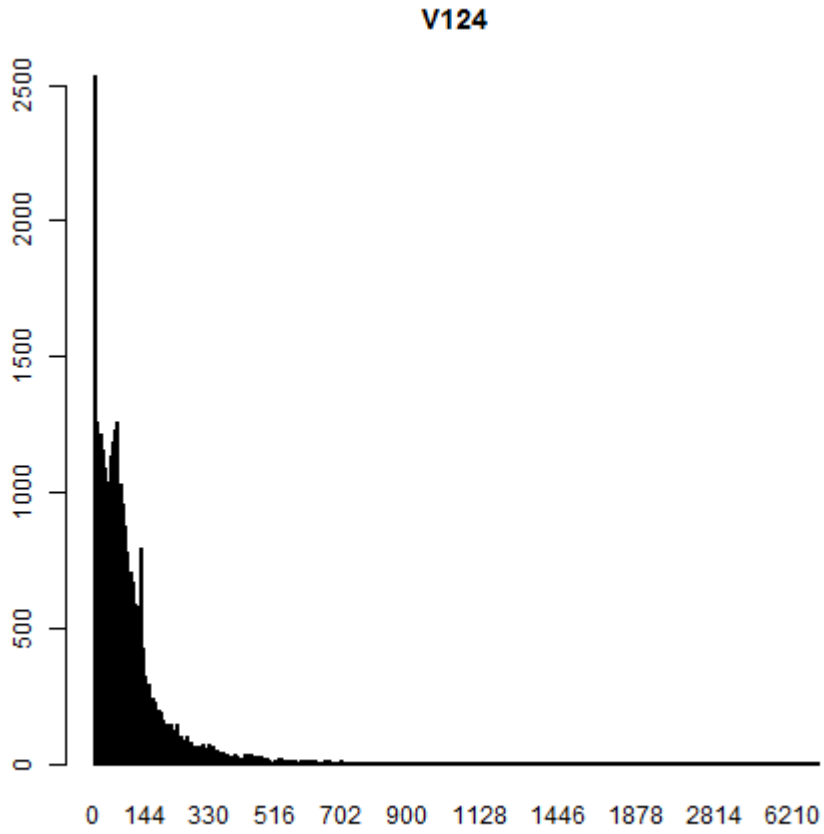

V129
250
200
150
100
50
0
0
24
48
72
96
132
156
228
1008

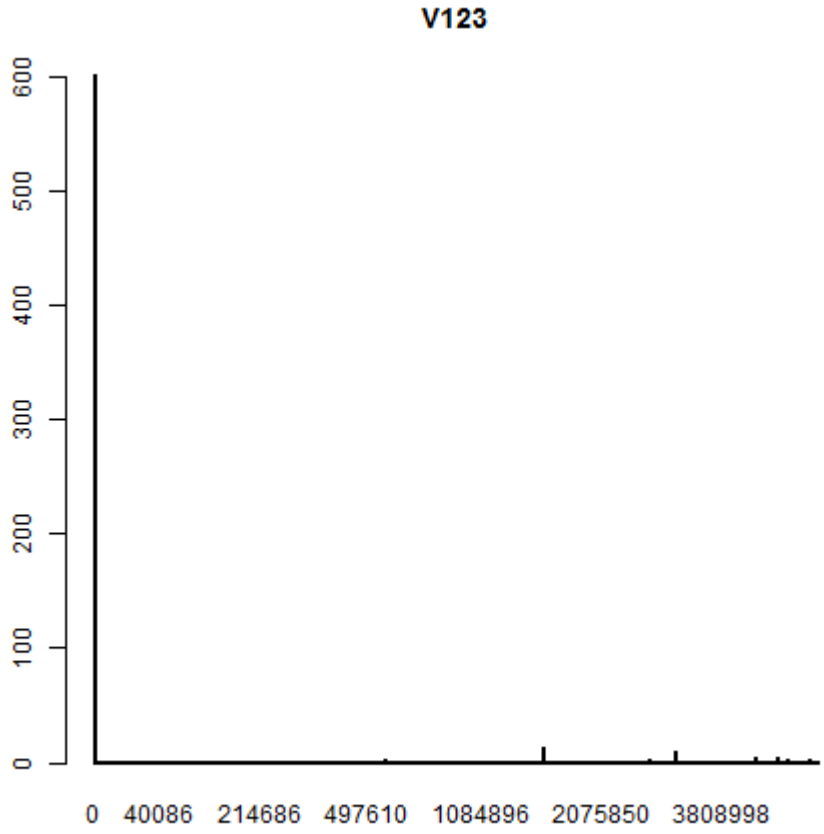

V128
30000
25000
20000
15000
10000
5000
0
3av_
7A3j
DM_V
g5HH
oT7d
uKAl

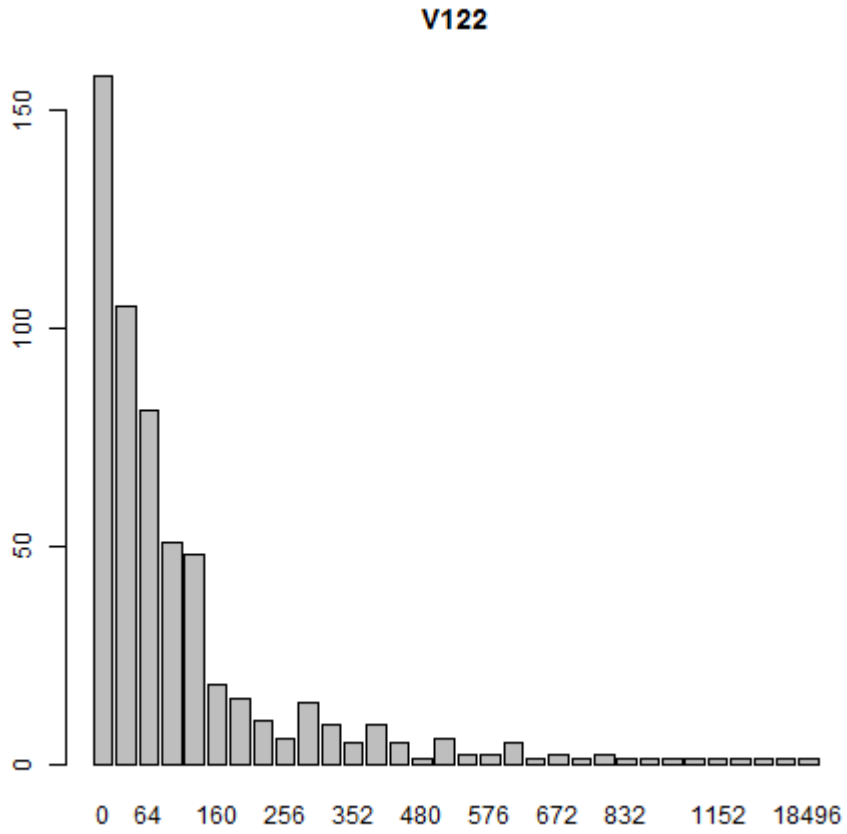

V127
400
300
200
100
0
0
8064
22884
50556
83742
127422
297276
820320

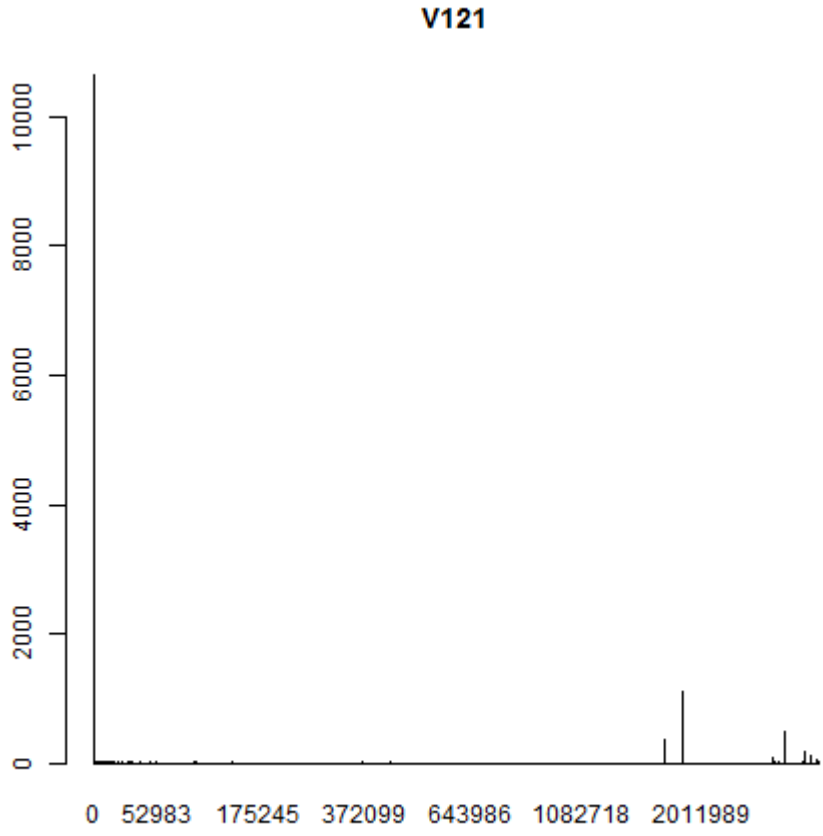

V126
0
10
20
30
40
50
_1T08y_
8lel20t
doibgcr
hvtfTlA
Lualyxr
qDkilTa
vcViAvl
zf8vrXj

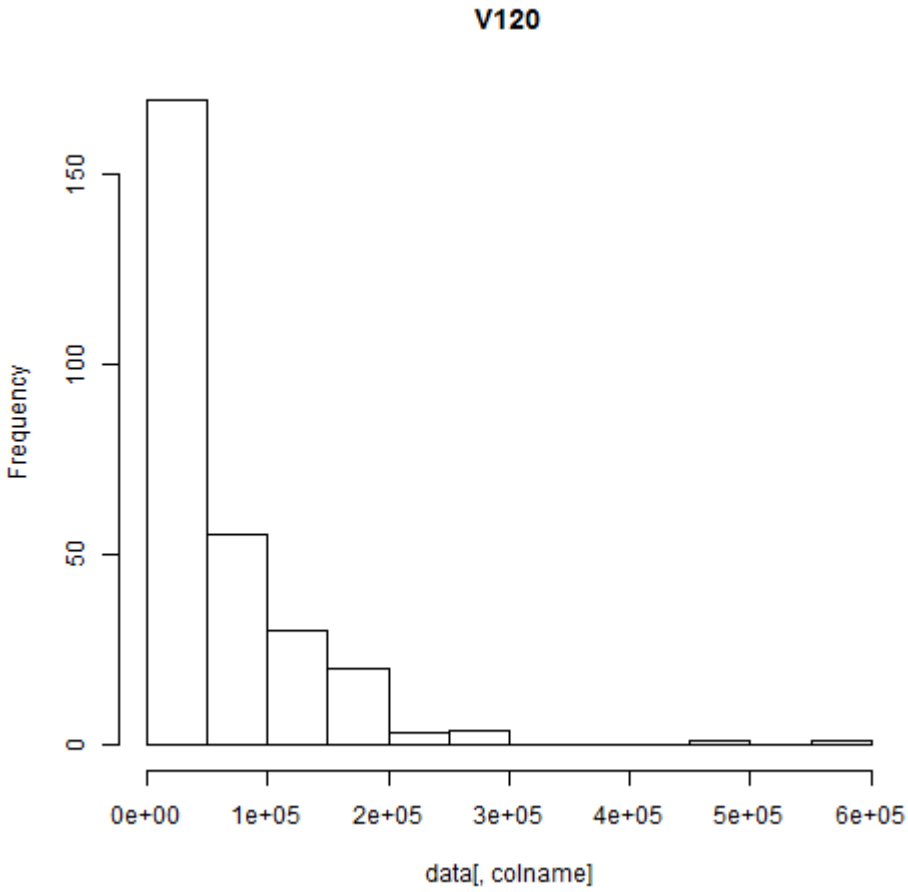

V125
0
5000
10000
15000
20000
25000
30000
klsH
sBgB

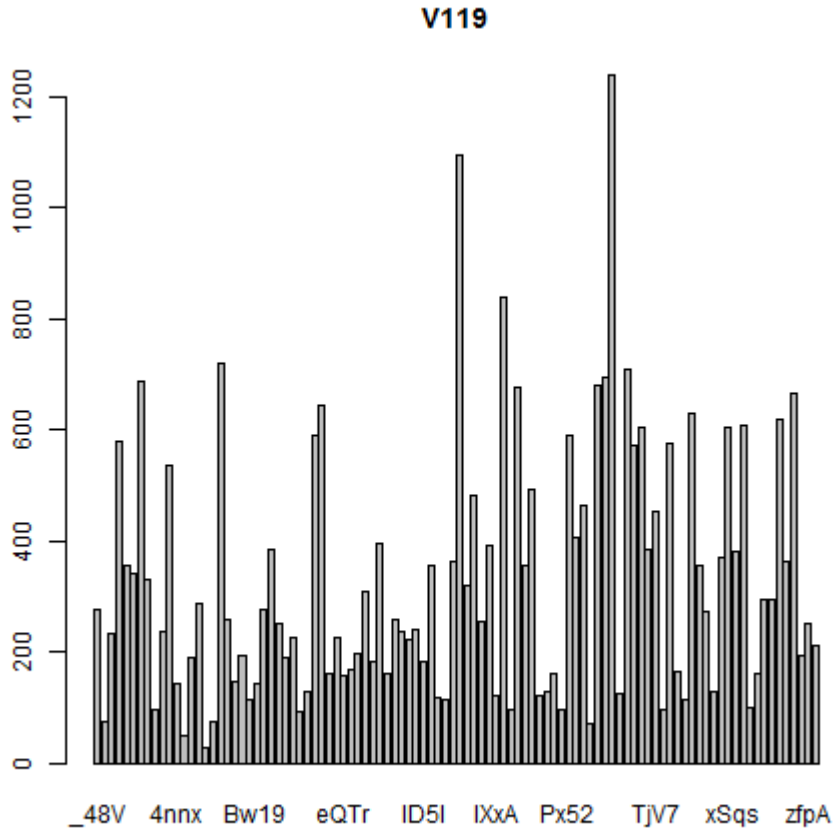

V124
2500
2000
1500
1000
500
0
0
144
330
516
702
900
1128
1446
1878
2814
6210

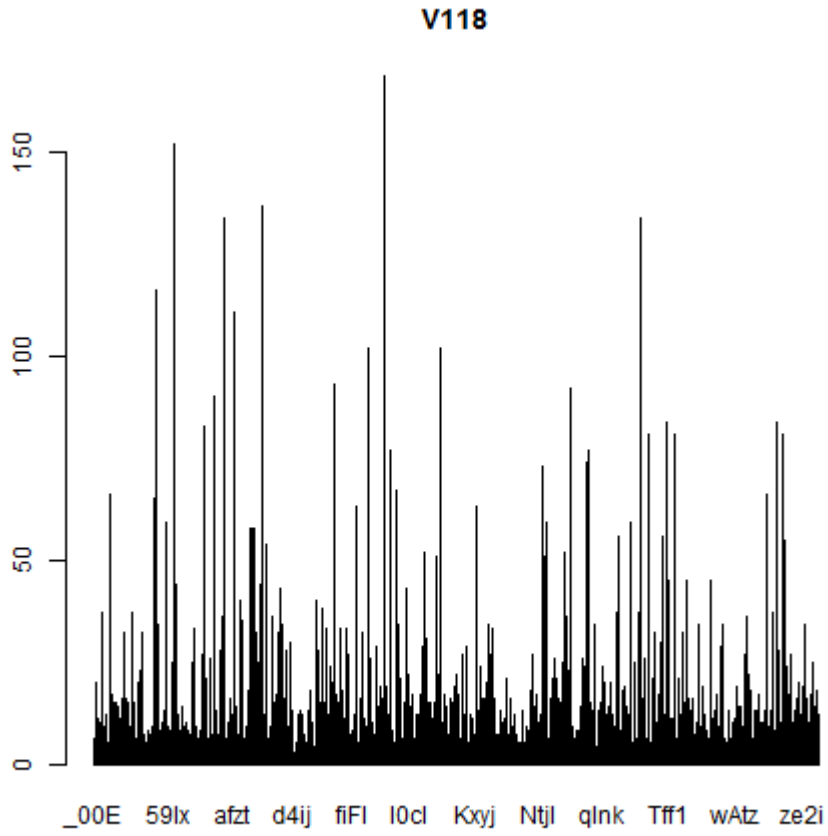

V123
600
500
400
300
200
100
0
0
40086
214686
497610
1084896
2075850
3808998

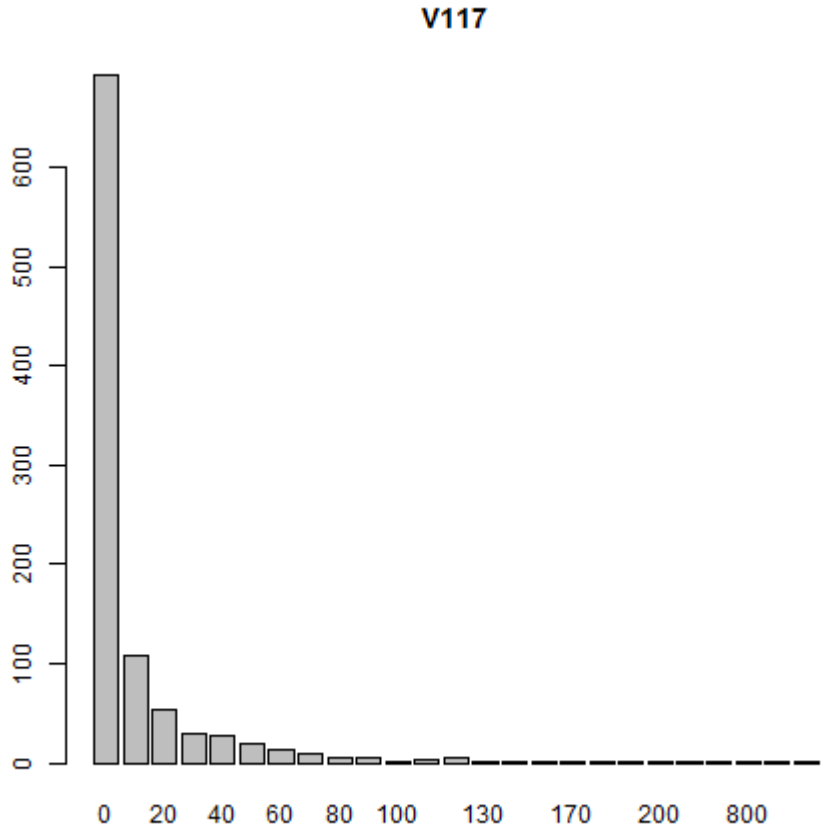

V122
150
100
50
0
0
64
160
256
352
480
576
672
832
1152
18496

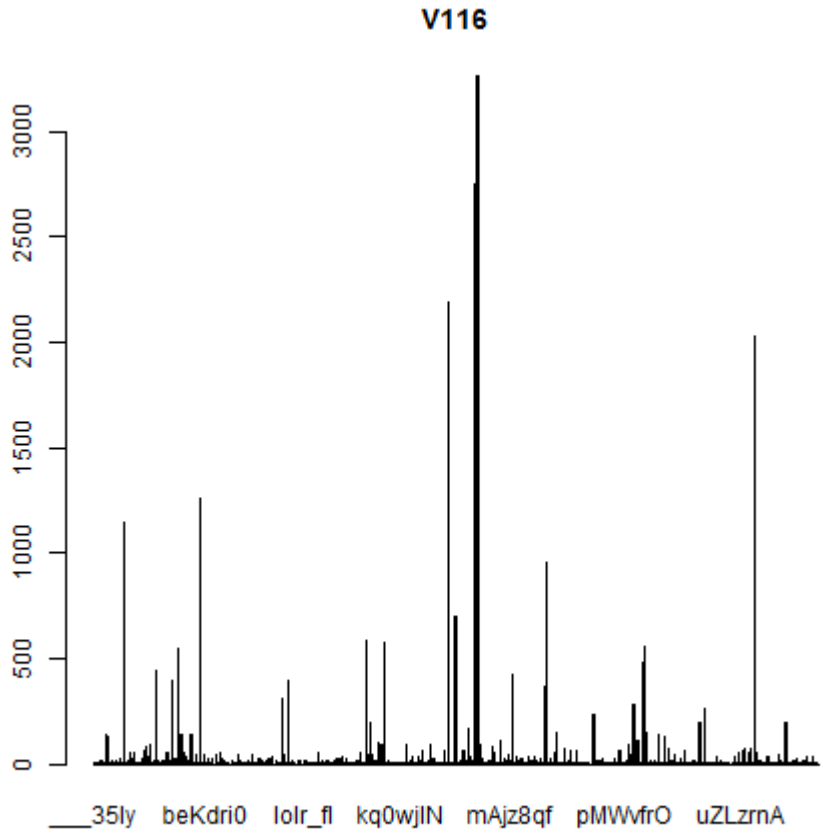

V121
10000
8000
6000
4000
2000
0
0
52983
175245
372099
643986
1082718
2011989

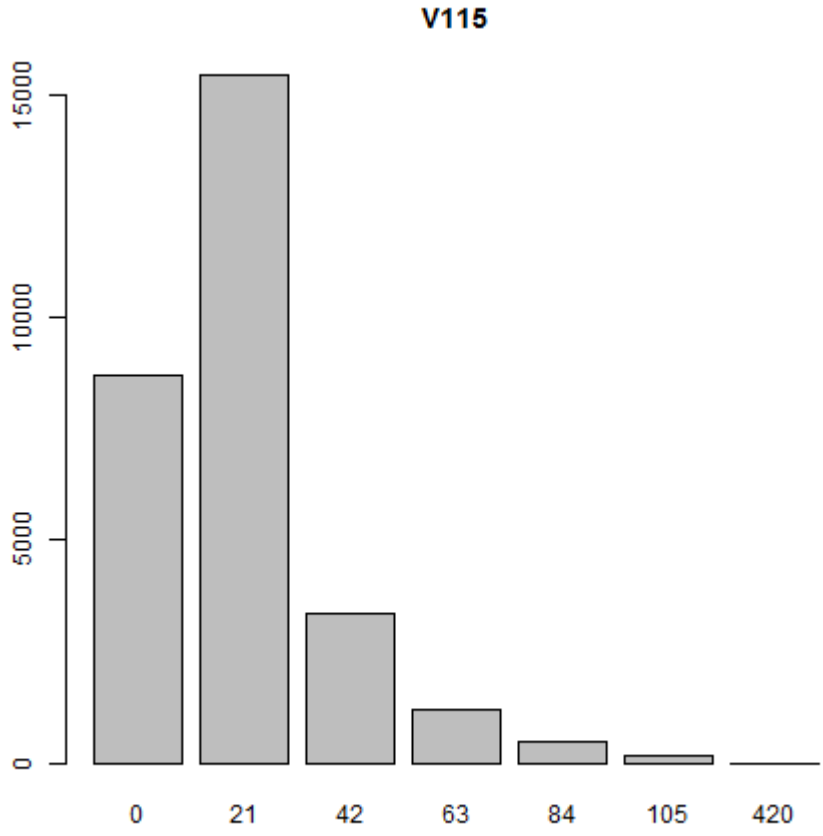

V120
Frequency
0
50
100
150
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
6e+05
data[, colname]

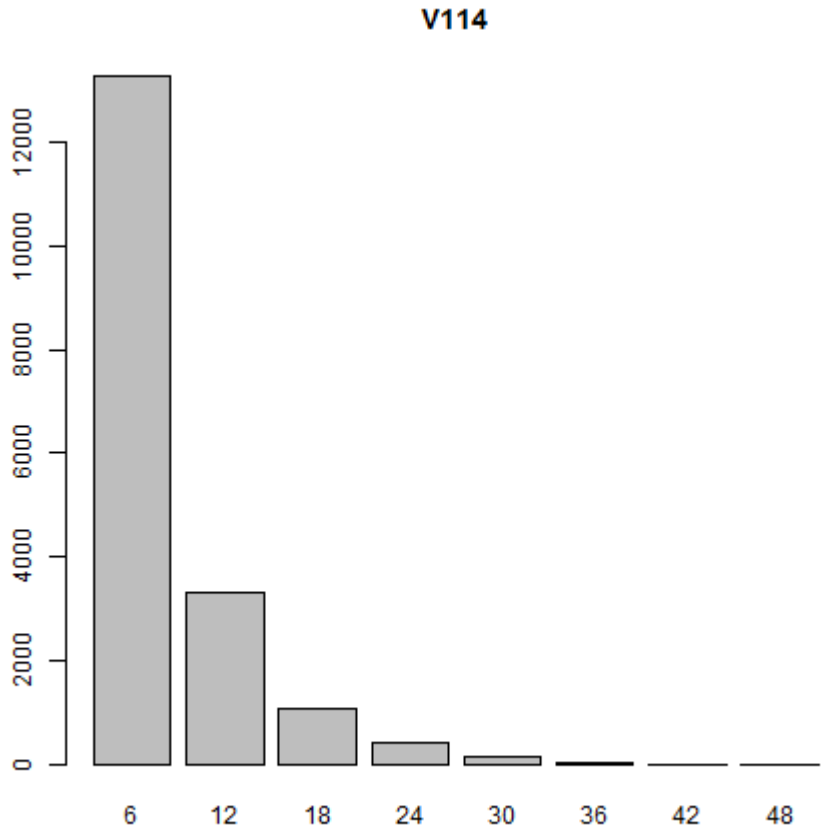

V119
0
200
400
600
800
1000
1200
_48V
4nnx
Bw19
eQTr
ID5I
IXxA
Px52
TjV7
xSqs
zfpA

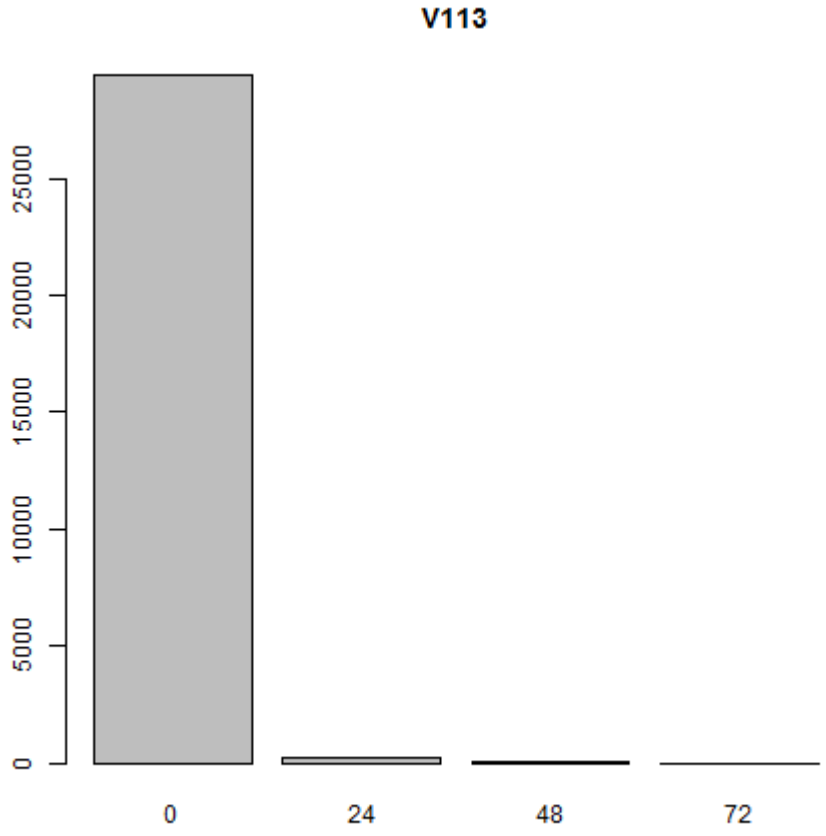

V118
150
100
50
0
_00E
59lx
afzt
d4ij
fiFl
l0cl
Kxyj
Ntjl
qlnk
Tff1
wAtz
ze2i

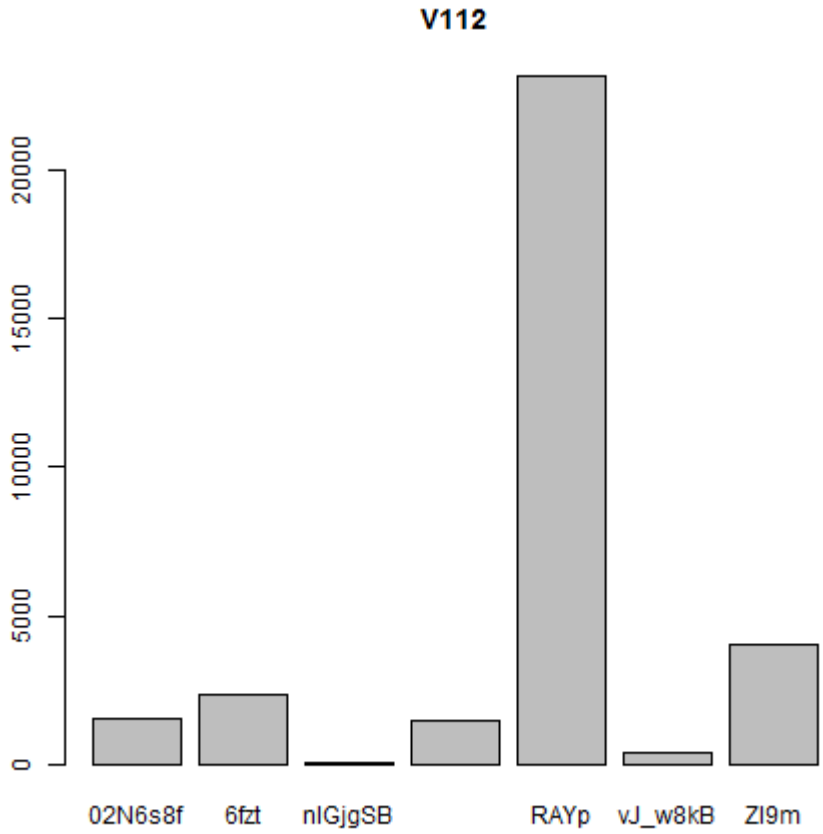

V117
600
500
400
300
200
100
0
0
20
40
60
80
100
130
170
200
800

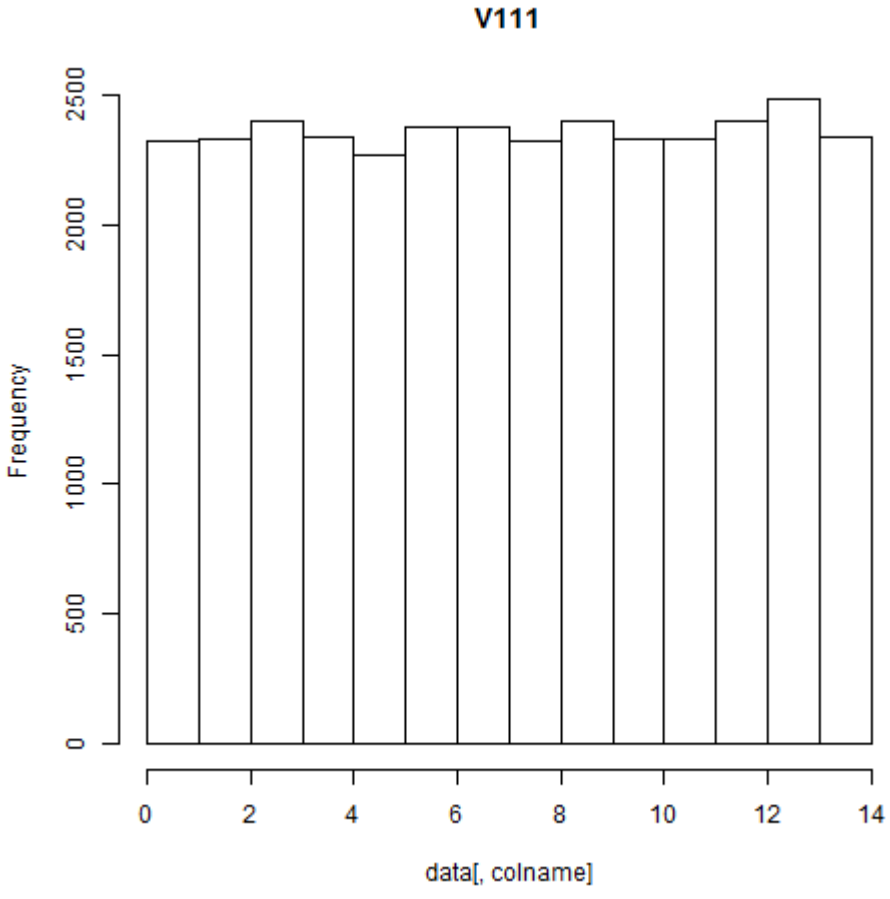

V116
3000
2500
2000
1500
1000
500
0
___35Iy
beKdri0
loIr_fl
kq0wjlN
mAjz8qf
pMWvfrO
uZLzrnA

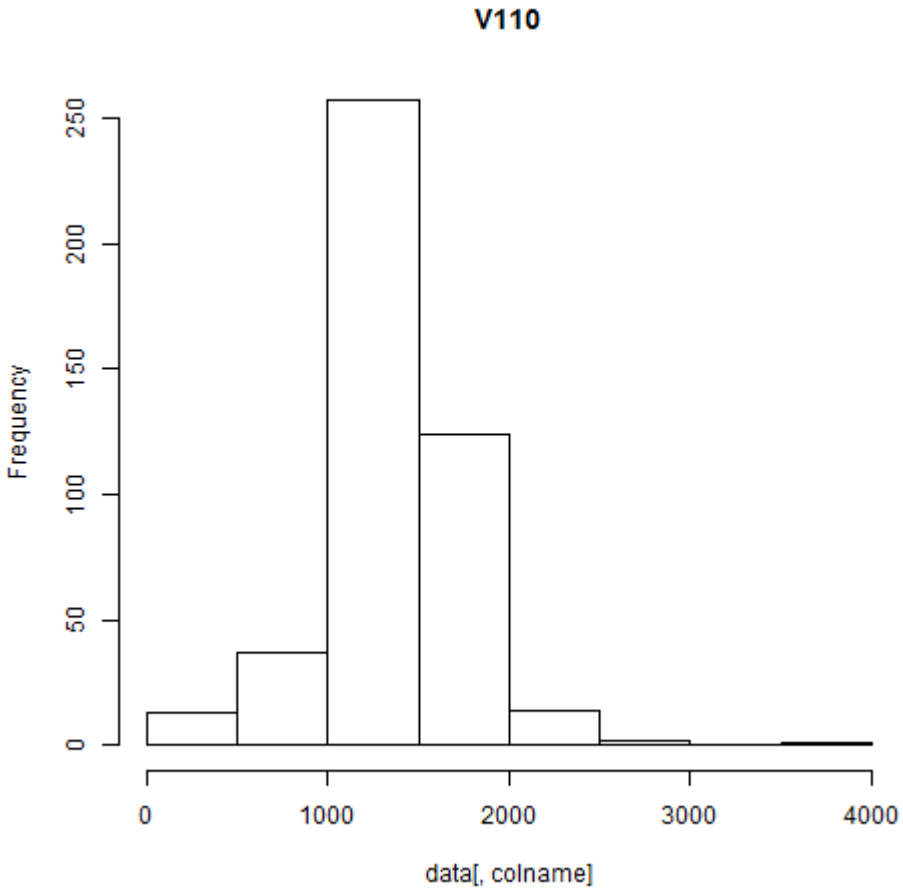

V115
15000
10000
5000
0
0
21
42
63
84
105
420

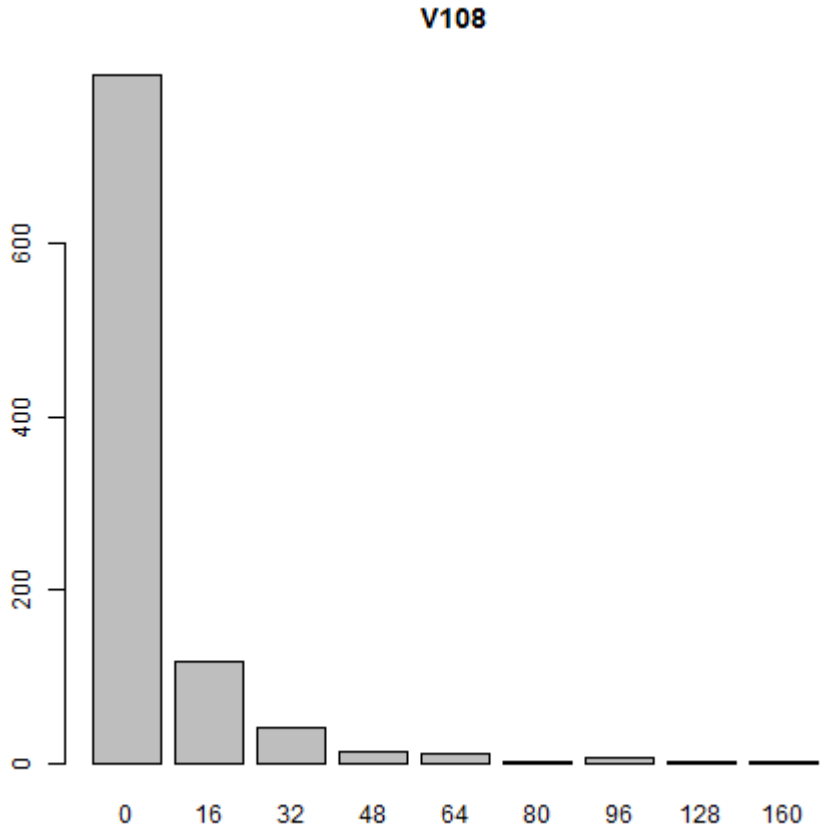

V114
0
2000
4000
6000
8000
10000
12000
6
12
18
24
30
36
42
48

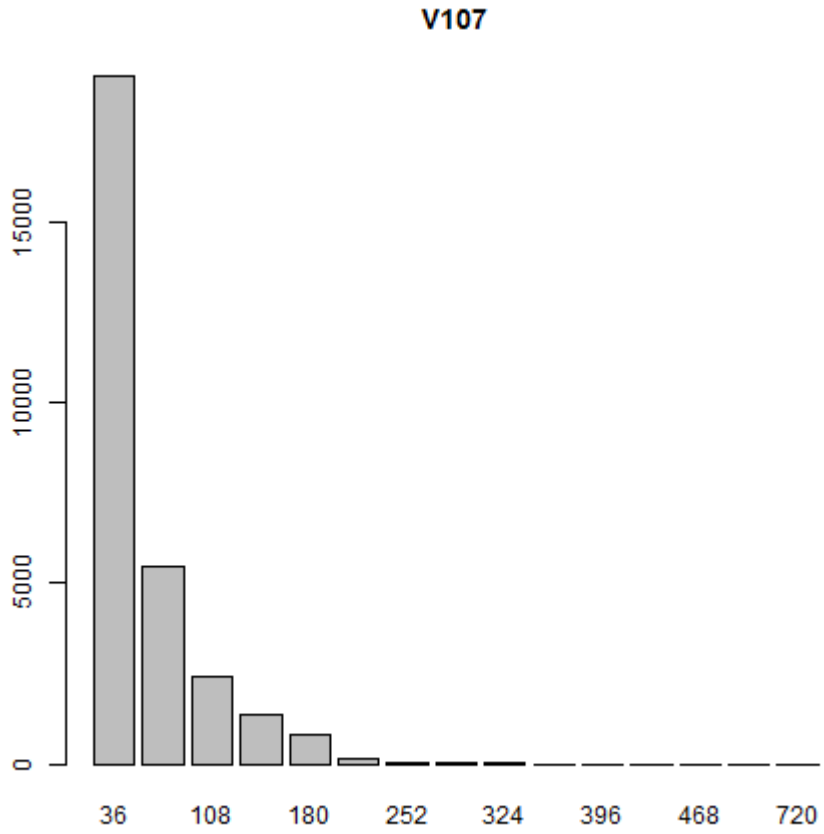

V113
0
5000
10000
15000
20000
25000
0
24
48
72

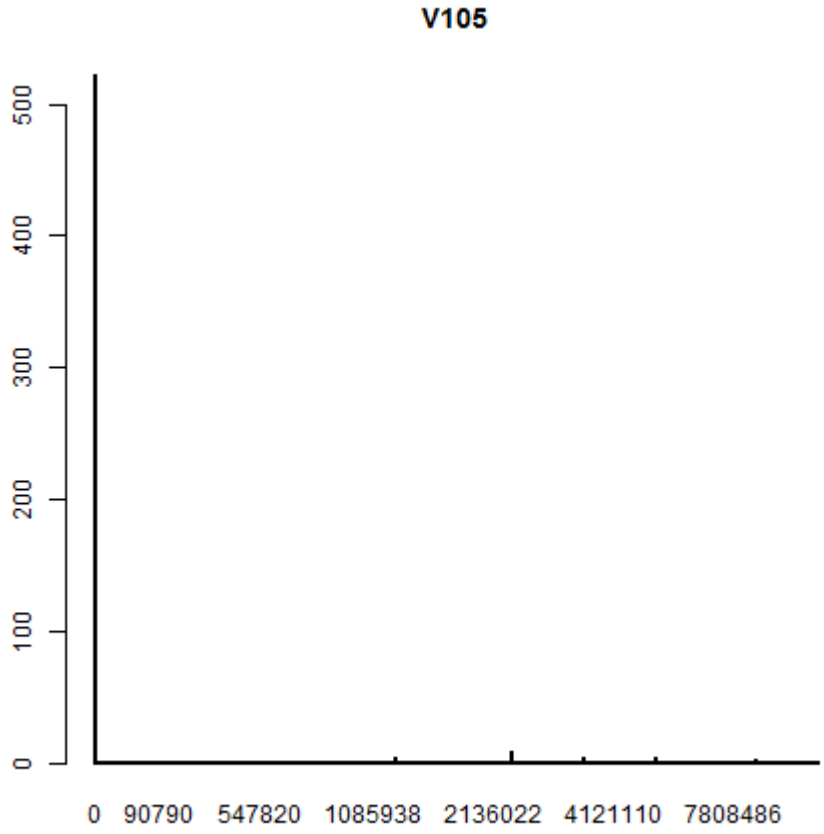

V112
20000
15000
10000
5000
0
02N6s8f
6fzt
nIGjgSB
RAYp
vJ_w8kB
ZI9m

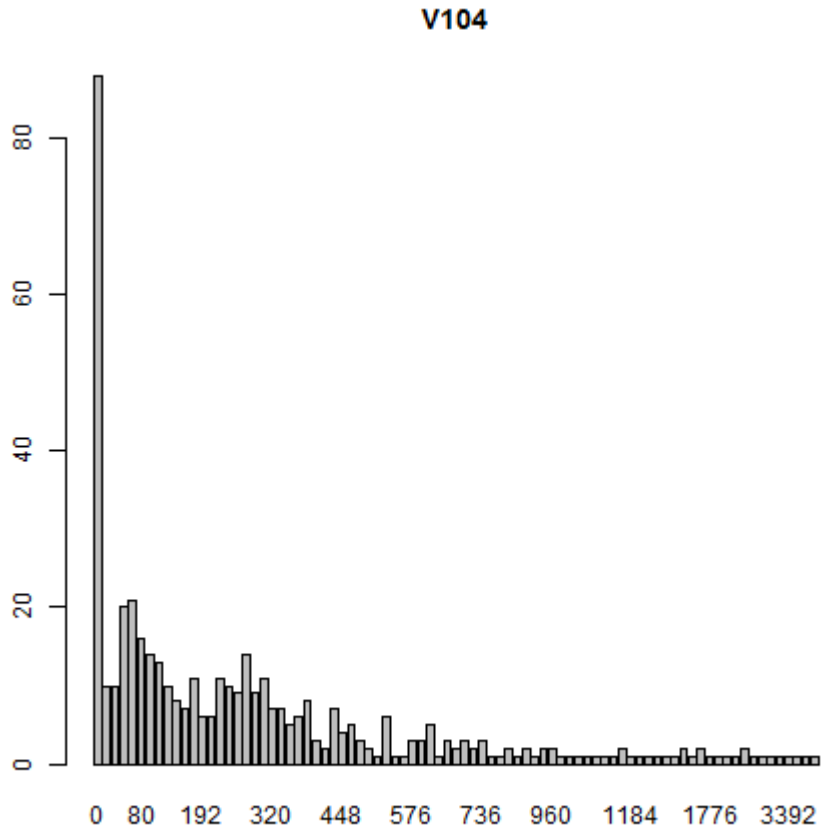

V111
Frequency
2500
2000
1500
1000
500
0
0
2
4
6
8
10
12
14
data[, colname]

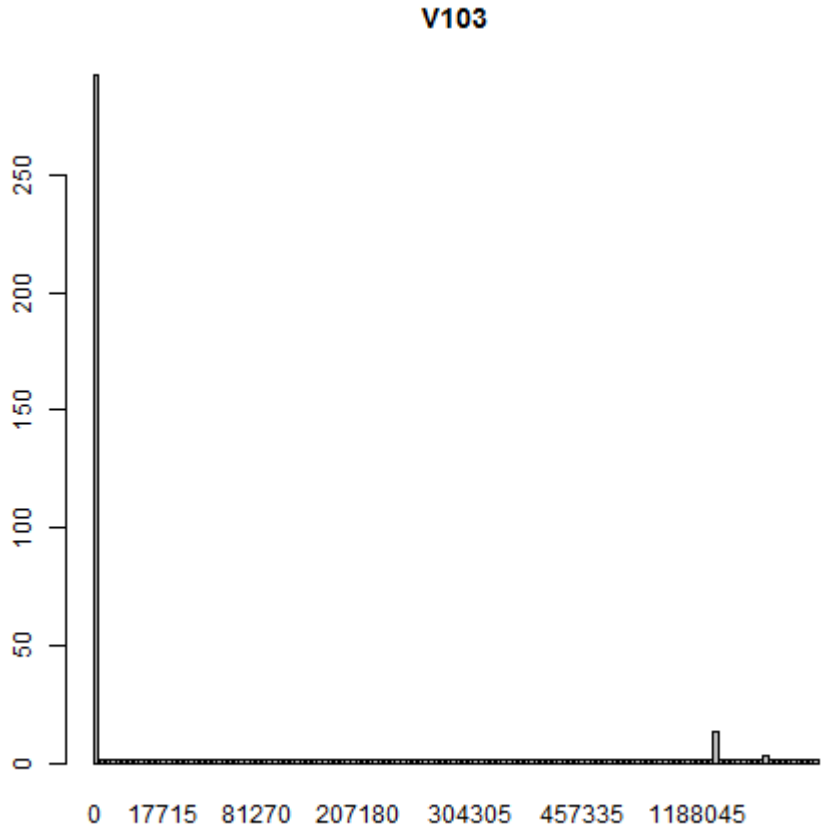

V110
Frequency
0
50
100
150
200
250
0
1000
2000
3000
4000
data[, colname]

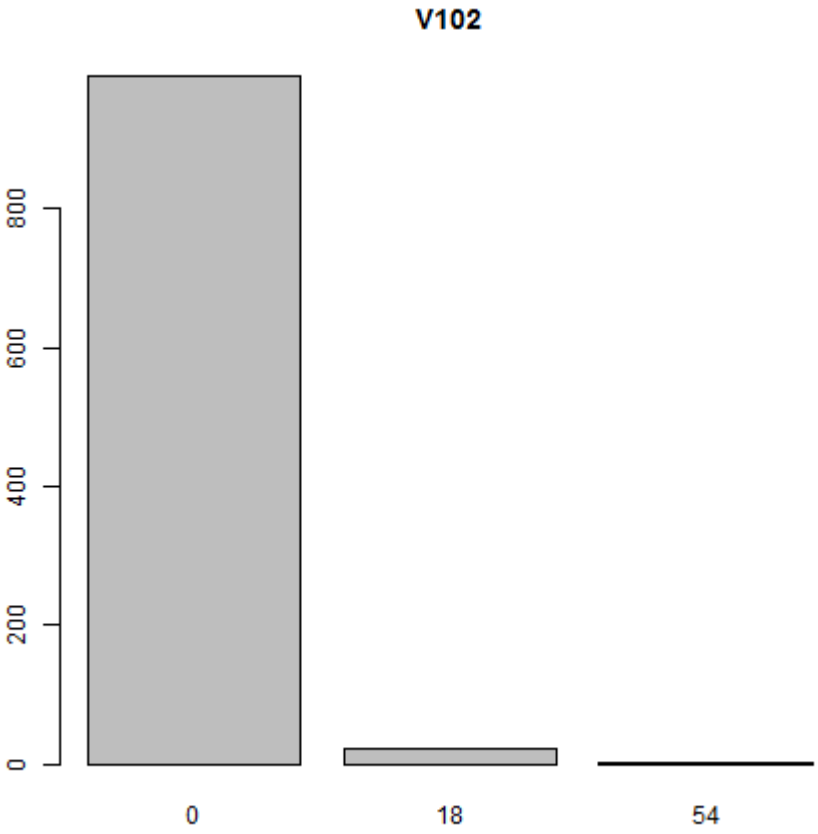

V108
0
200
400
600
0
16
32
48
64
80
96
128
160

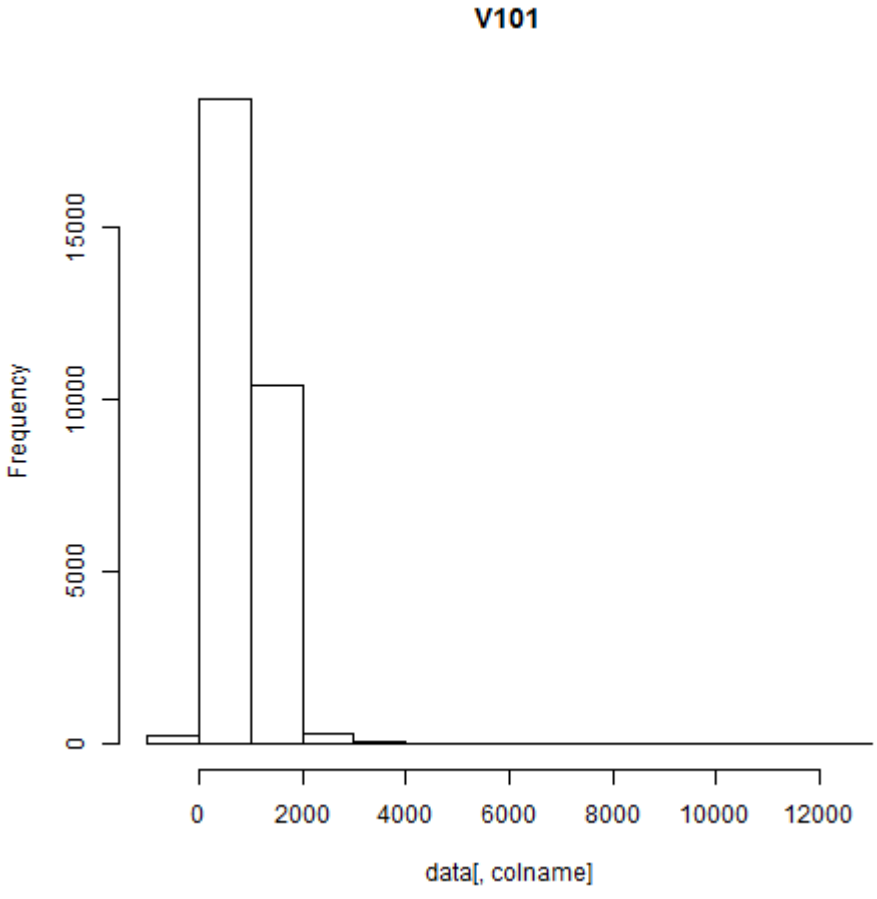

V107
15000
10000
5000
0
36
108
180
252
324
396
468
720

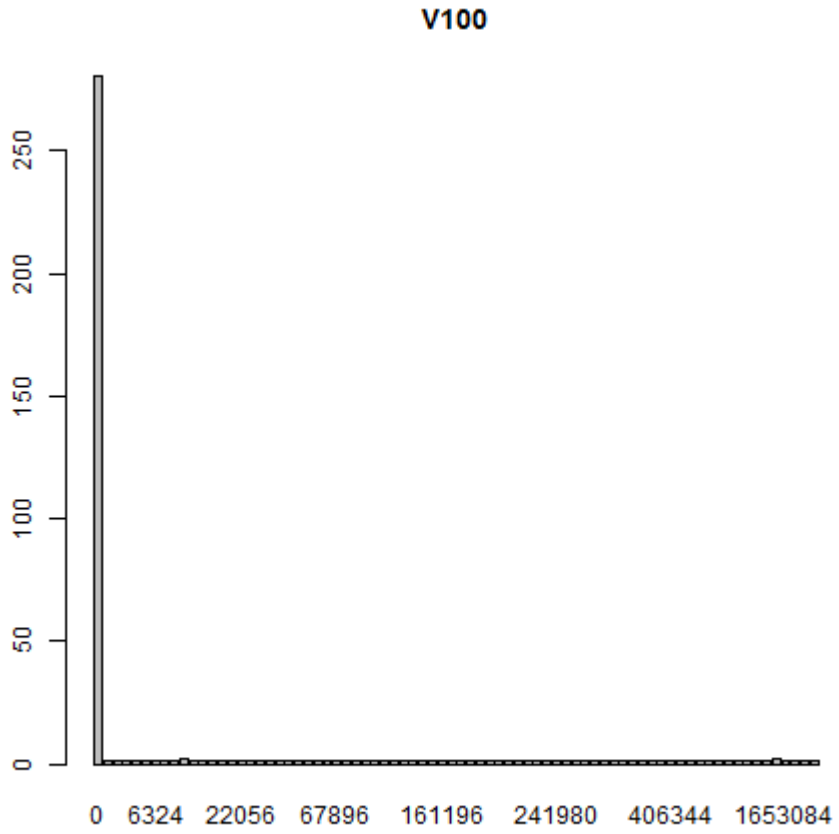

V105
500
400
300
200
100
0
0
90790
547820
1085938
2136022
4121110
7808486

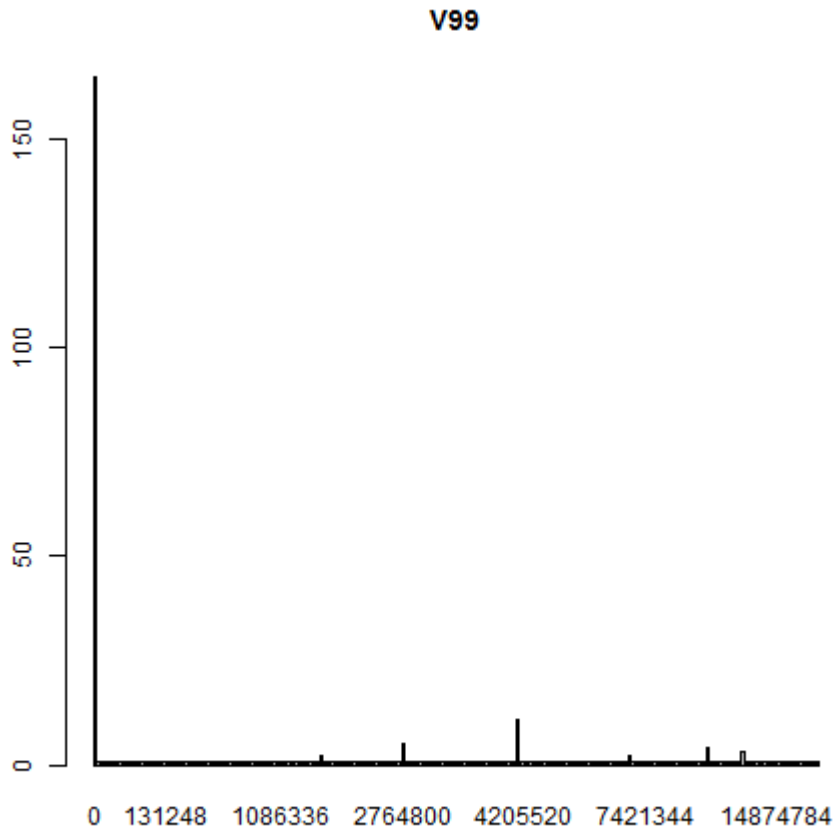

V104
0
20
40
60
80
0
80
192
320
448
576
736
960
1184
1776
3392

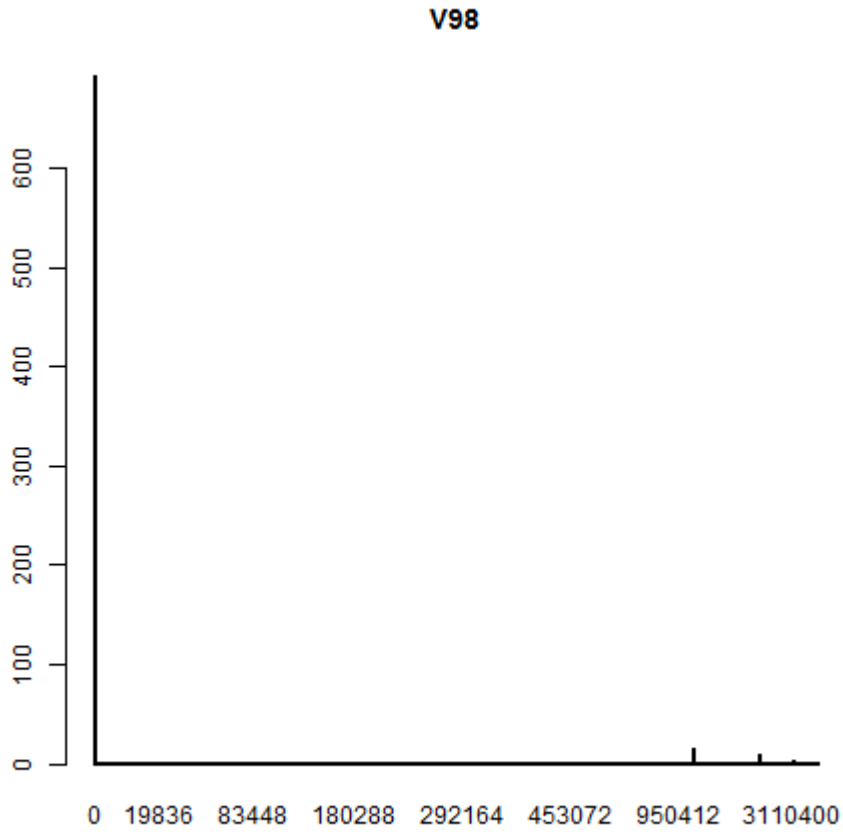

V103
0
50
100
150
200
250
0
17715
81270
207180
304305
457335
1188045

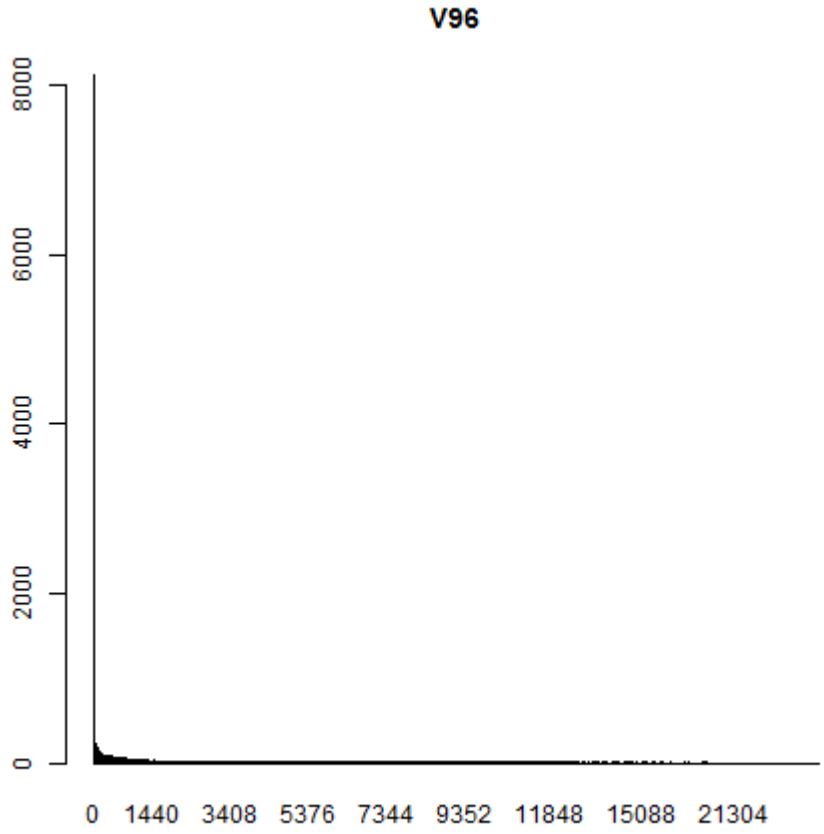

V102
800
600
400
200
0
0
18
54

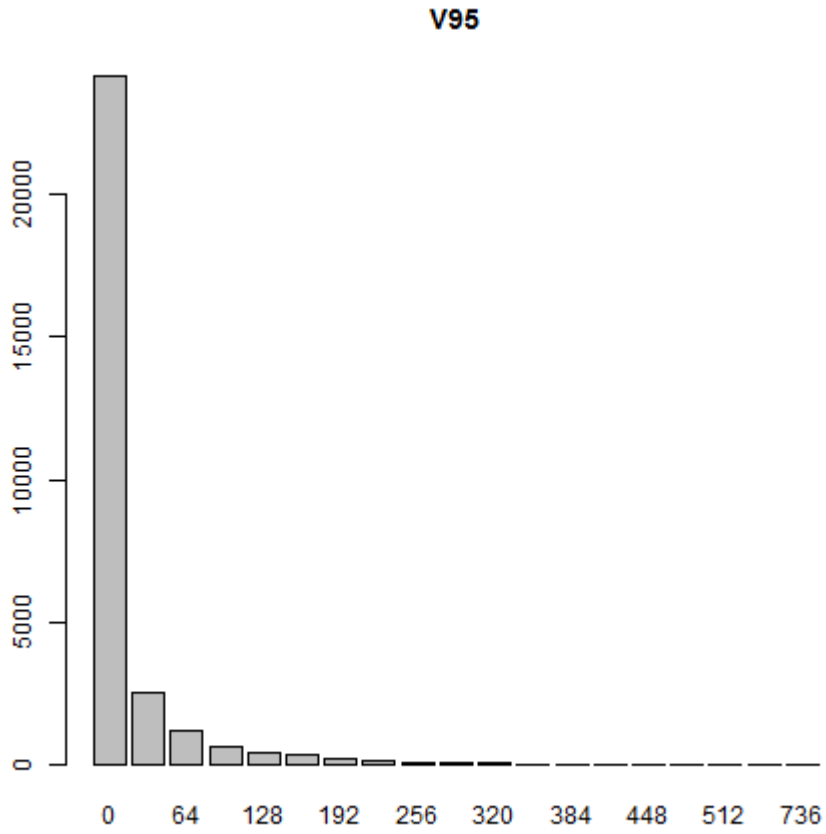

V101
Frequency
15000
10000
5000
0
0
2000
4000
6000
8000
10000
12000
data[, colname]

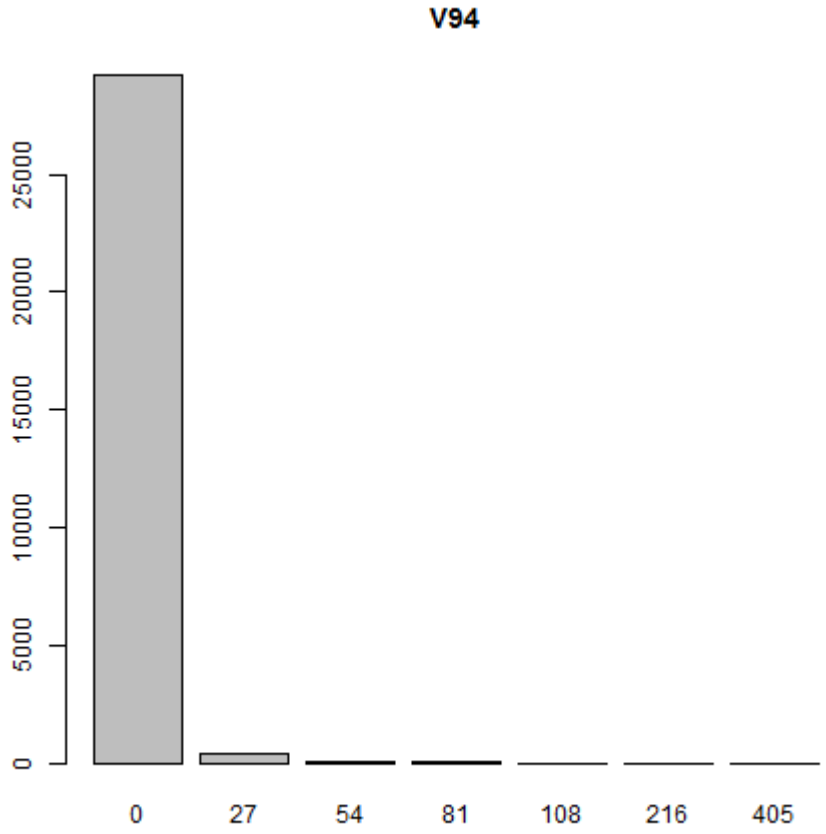

V100
0
50
100
150
200
250
0
6324
22056
67896
161196
241980
406344
1653084

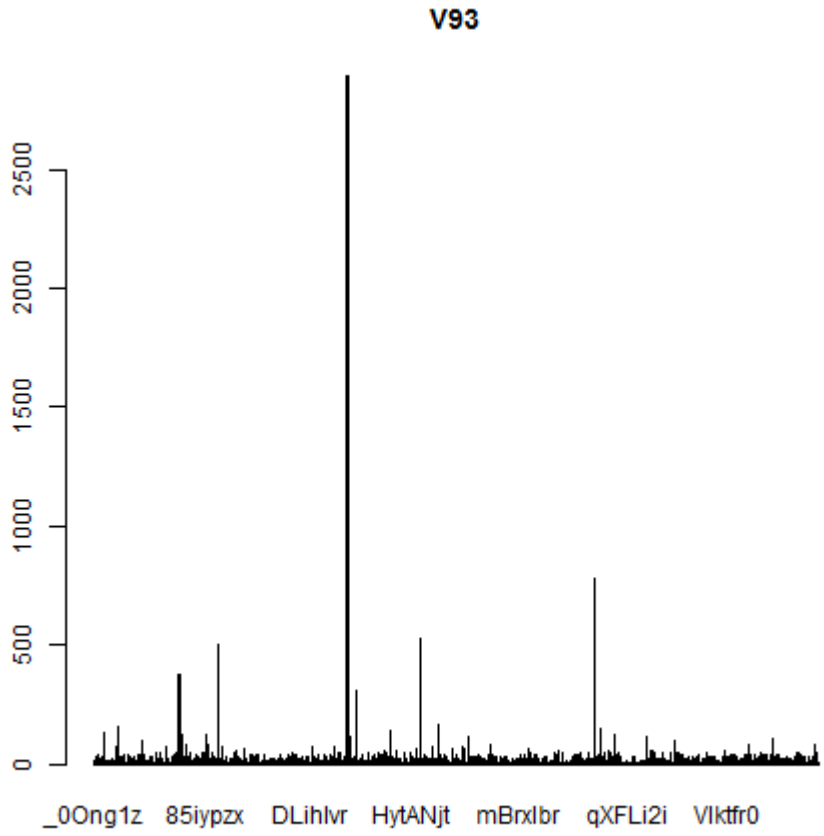

V99
0
50
100
150
0
131248
1086336
2764800
4205520
7421344
14874784

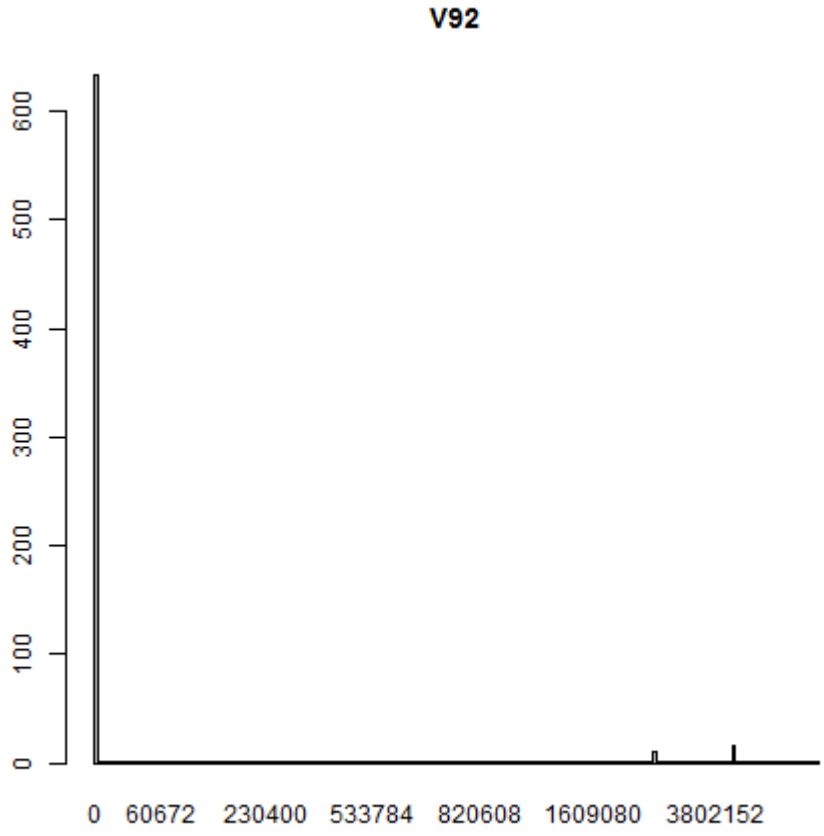

V98
0
100
200
300
400
500
600
0
19836
83448
180288
292164
453072
950412
3110400

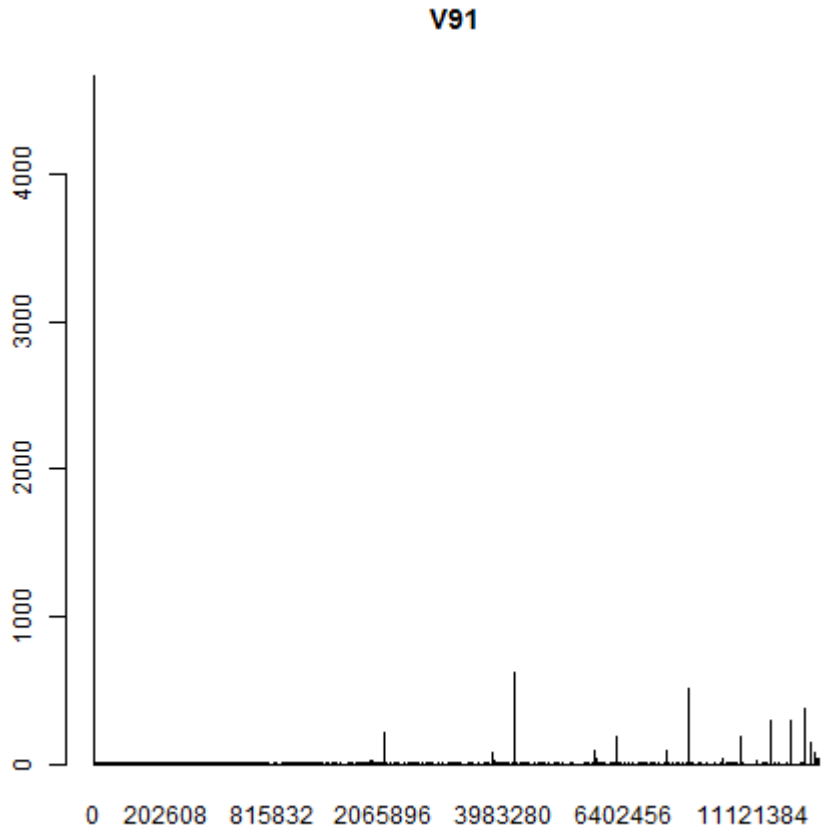

V96
0
2000
4000
6000
8000
0
1440
3408
5376
7344
9352
11848
15088
21304

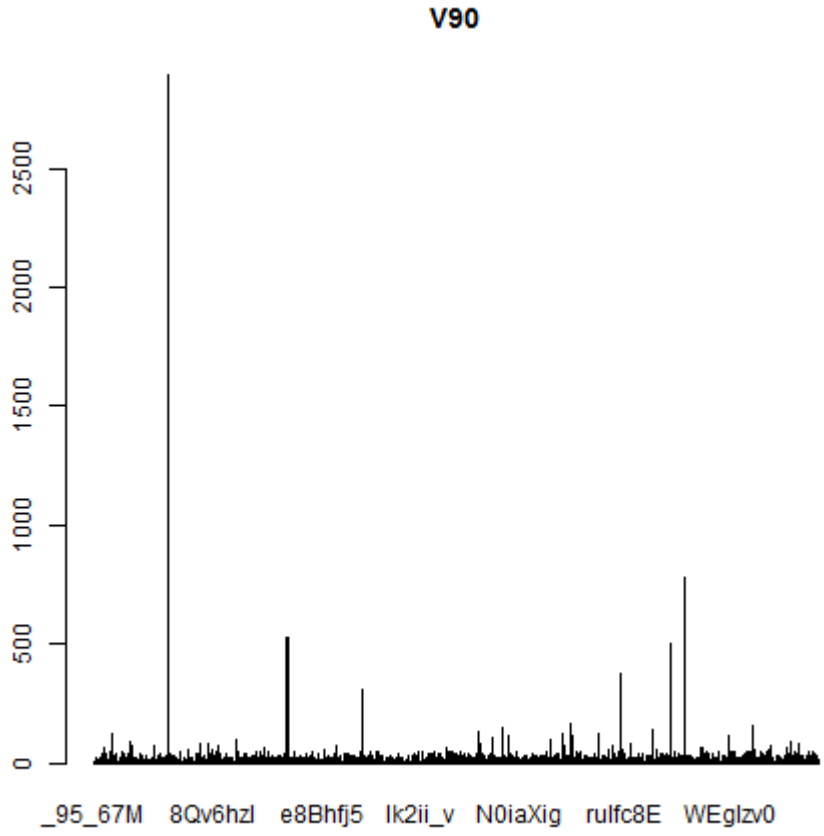

V95
0
5000
10000
15000
20000
0
64
128
192
256
320
384
448
512
736

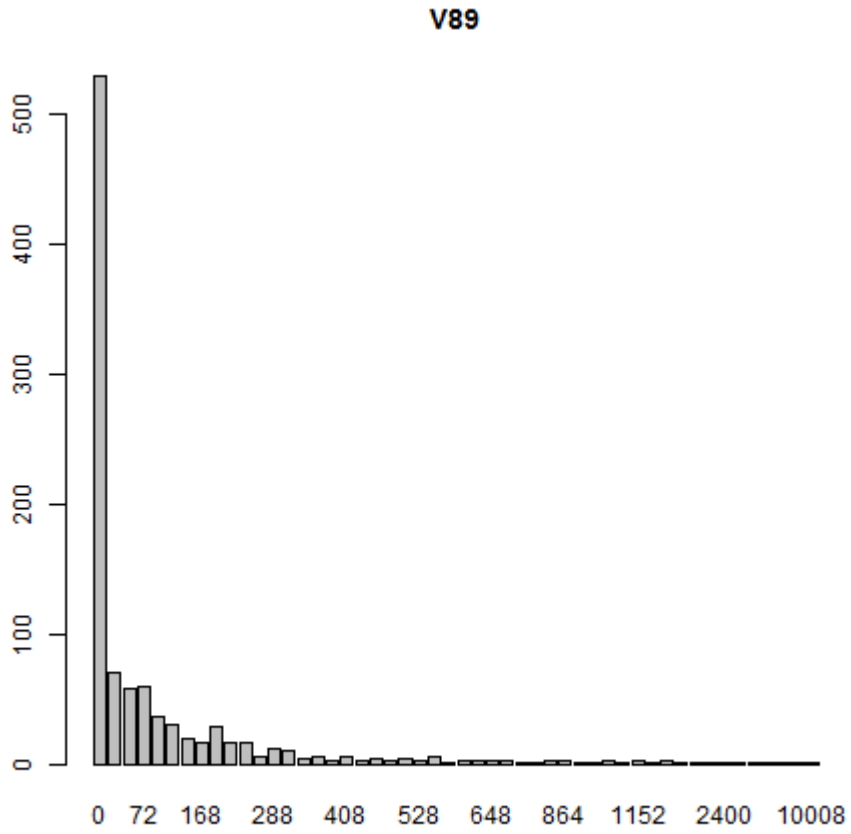

V94
0
5000
10000
15000
20000
25000
0
27
54
81
108
216
405

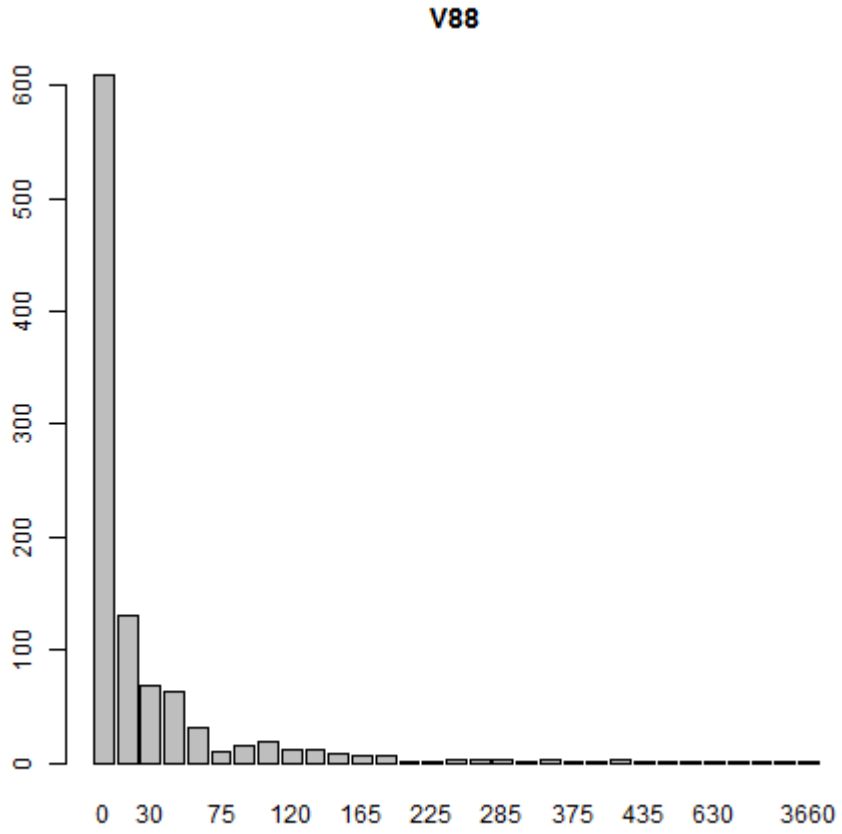

V93
0
500
1000
1500
2000
2500
_0Ong1z
85iypzx
DLihlvr
HytANjt
mBrxIbr
qXFLi2i
VIktfr0

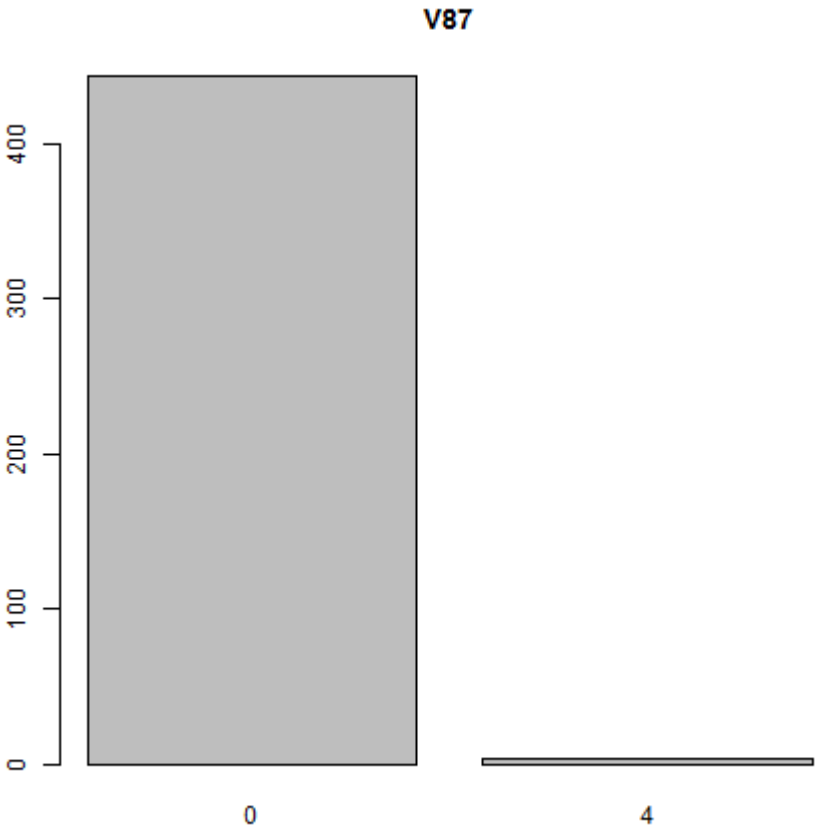

V92
0
100
200
300
400
500
600
0
60672
230400
533784
820608
1609080
3802152

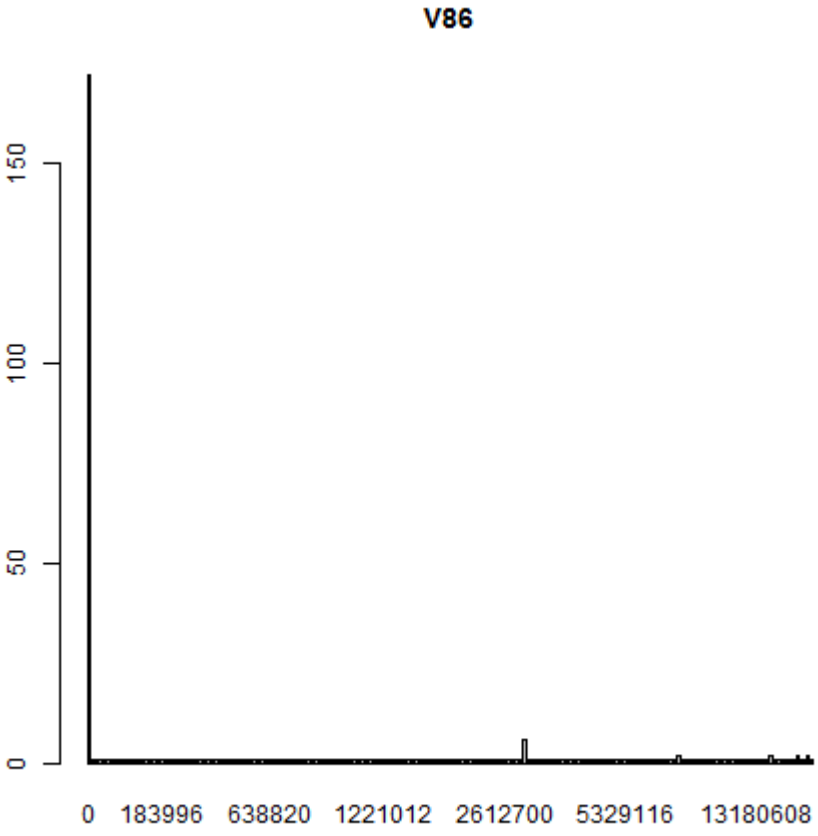

V91
4000
3000
2000
1000
0
0
202608
815832
2065896
3983280
6402456
11121384

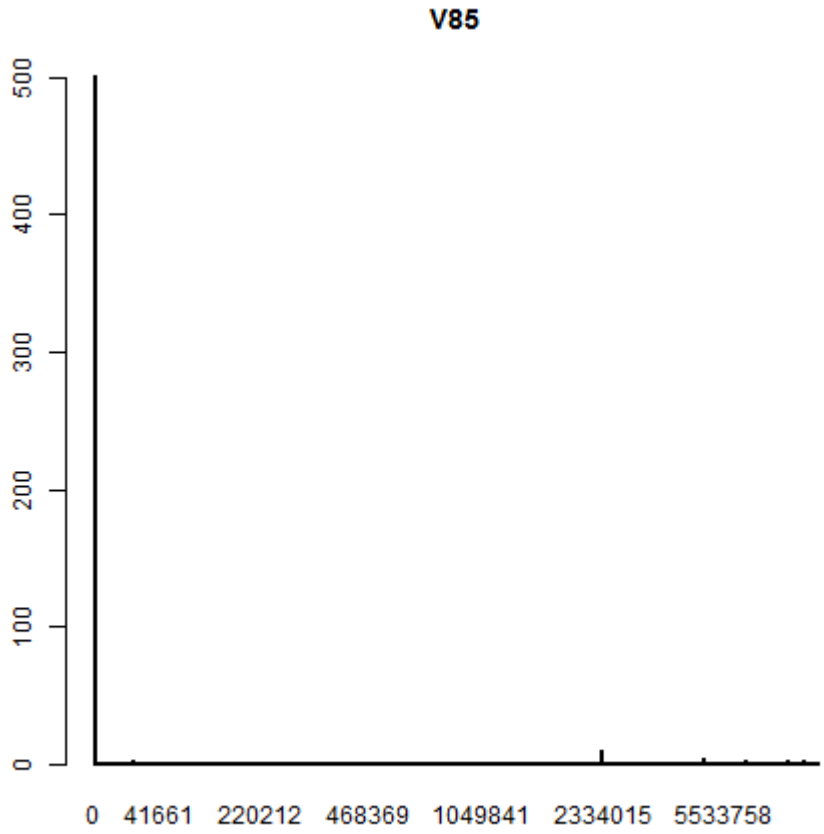

V90
2500
2000
1500
1000
500
0
_95_67M
8Qv6hzl
e8Bhfj5
Ik2ii_v
N0iaXig
rulfc8E
WEglzv0

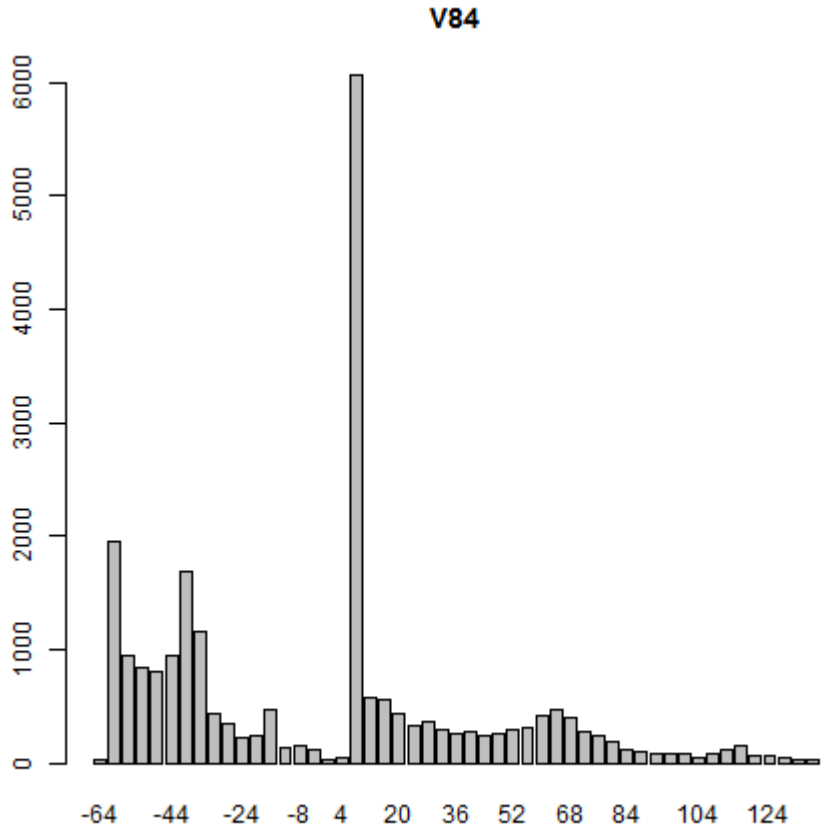

V89
0
100
200
300
400
500
0
72
168
288
408
528
648
864
1152
2400
10008

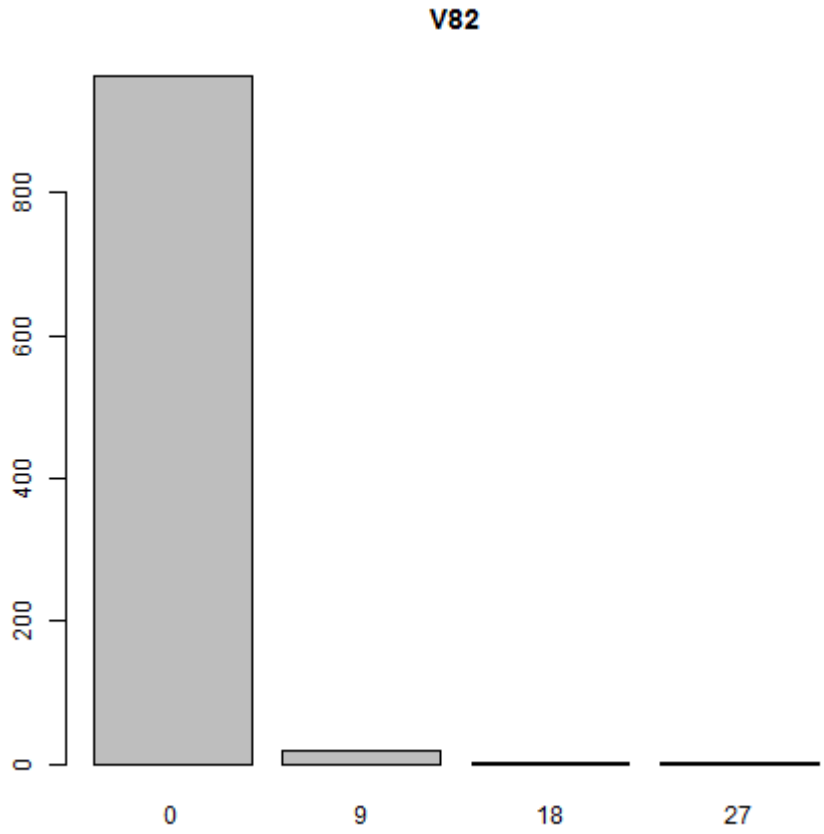

V88
0
100
200
300
400
500
600
0
30
75
120
165
225
285
375
435
630
3660

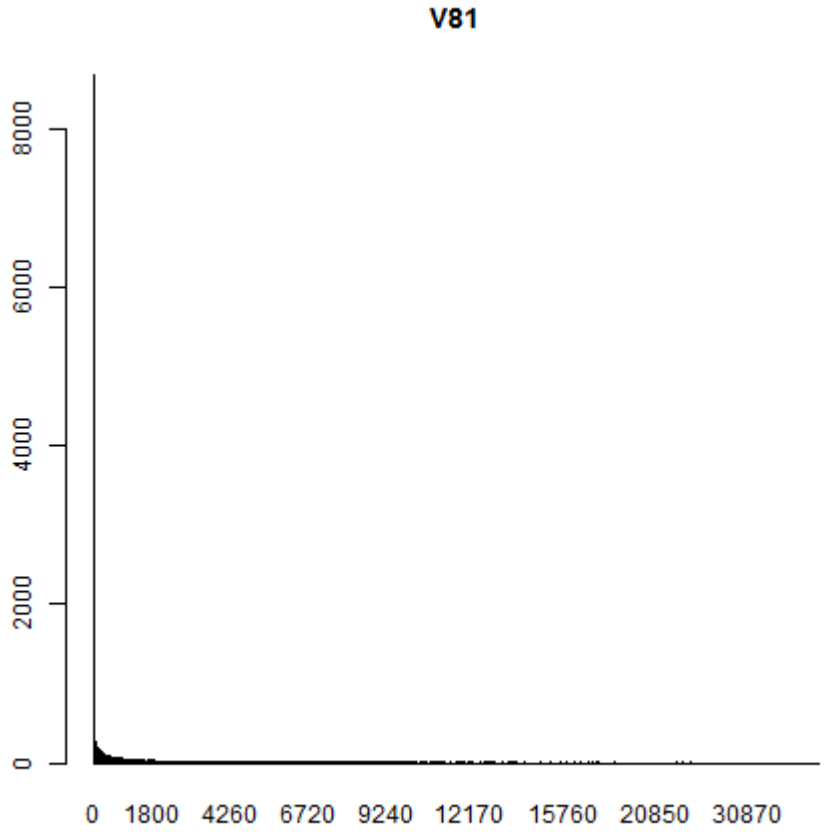

V87
400
300
200
100
0
0
4

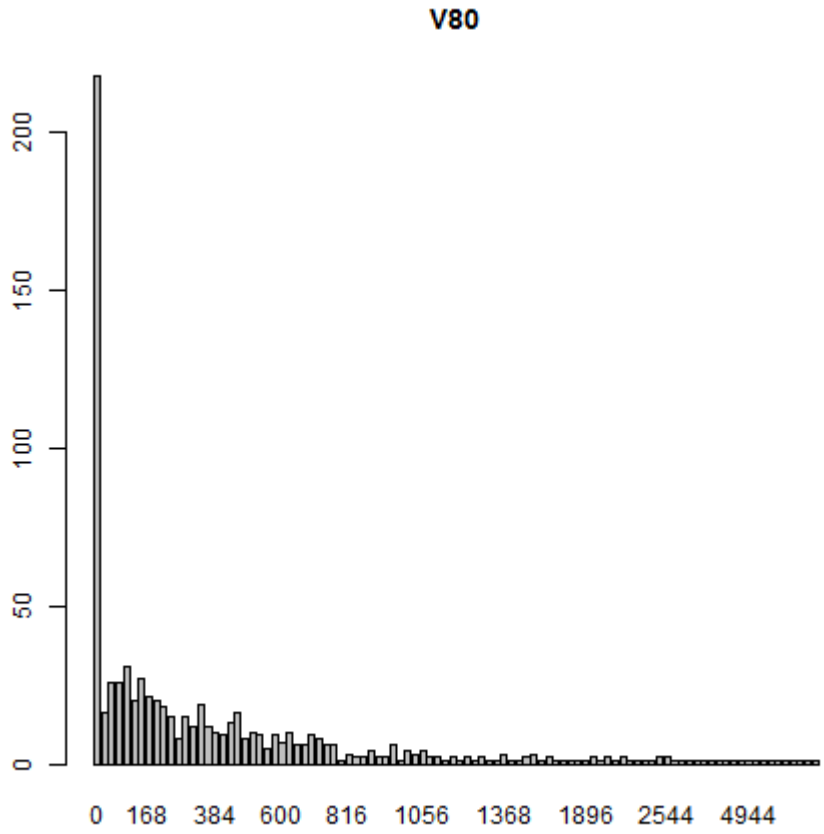

V86
150
100
50
0
0
183996
638820
1221012
2612700
5329116
13180608

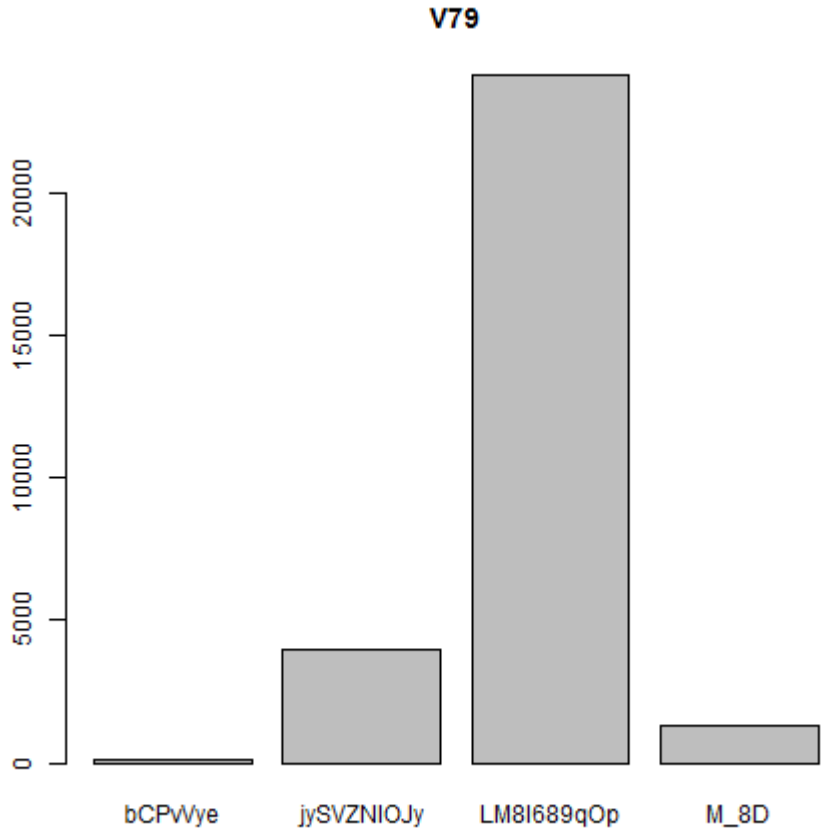

V85
0
100
200
300
400
500
0
41661
220212
468369
1049841
2334015
5533758

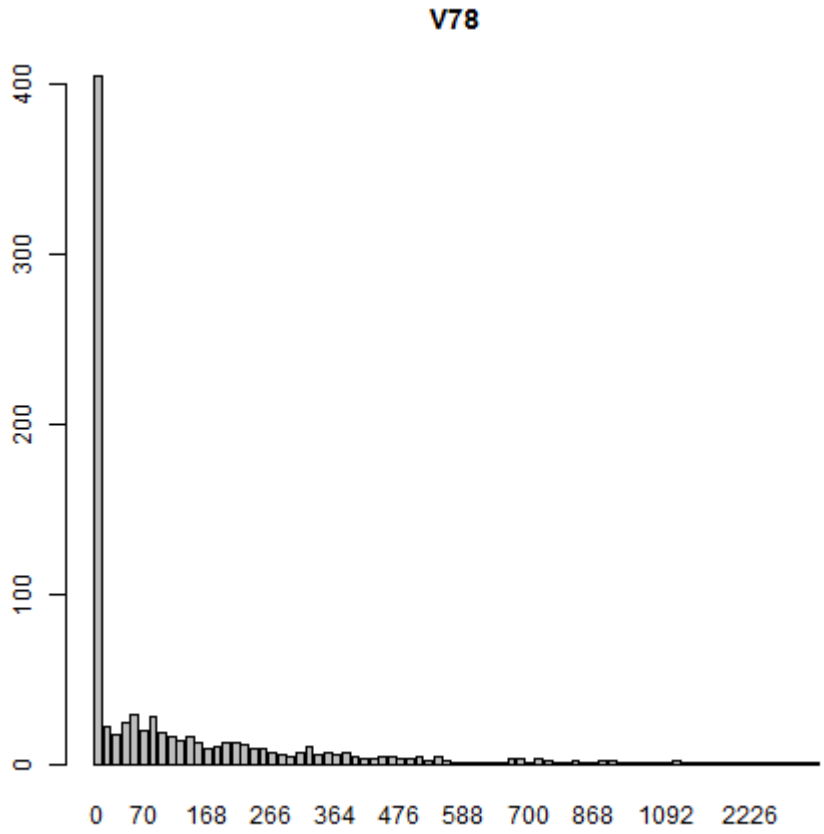

V84
0
1000
2000
3000
4000
5000
6000
-64
-44
-24
-8
4
20
36
52
68
84
104
124

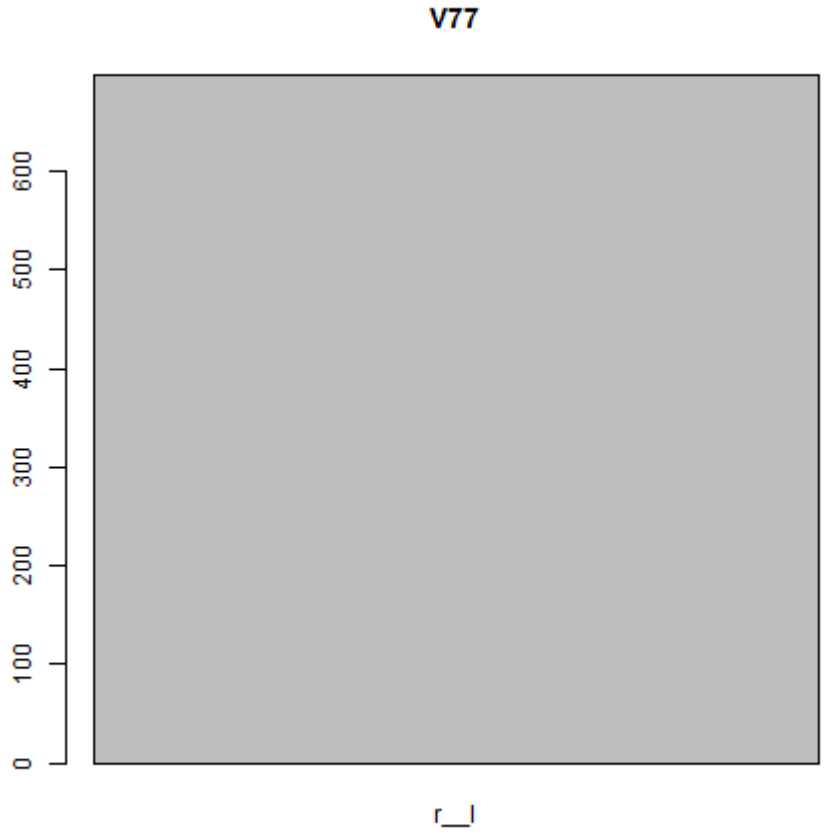

V82
800
600
400
200
0
0
9
18
27

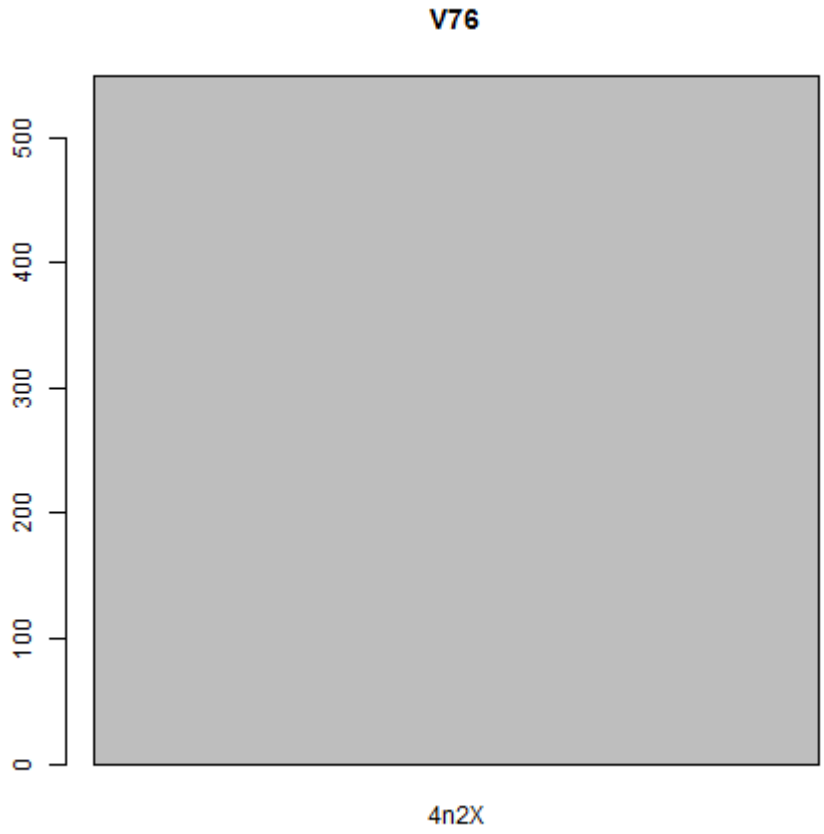

V81
8000
6000
4000
2000
0
0
1800
4260
6720
9240
12170
15760
20850
30870

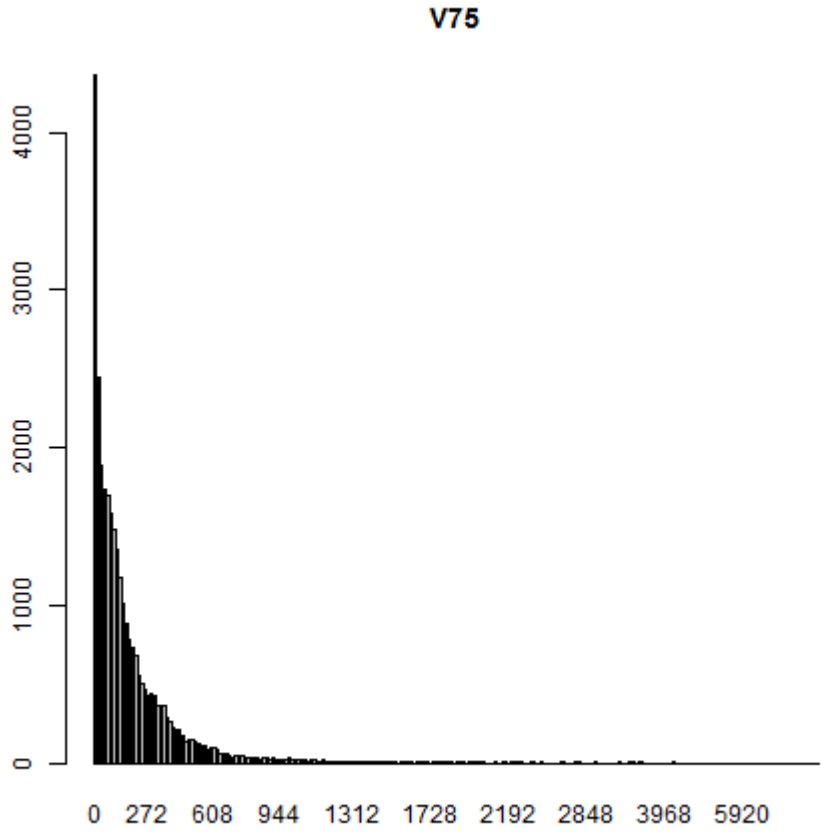

V80
200
150
100
50
0
0
168
384
600
816
1056
1368
1896
2544
4944

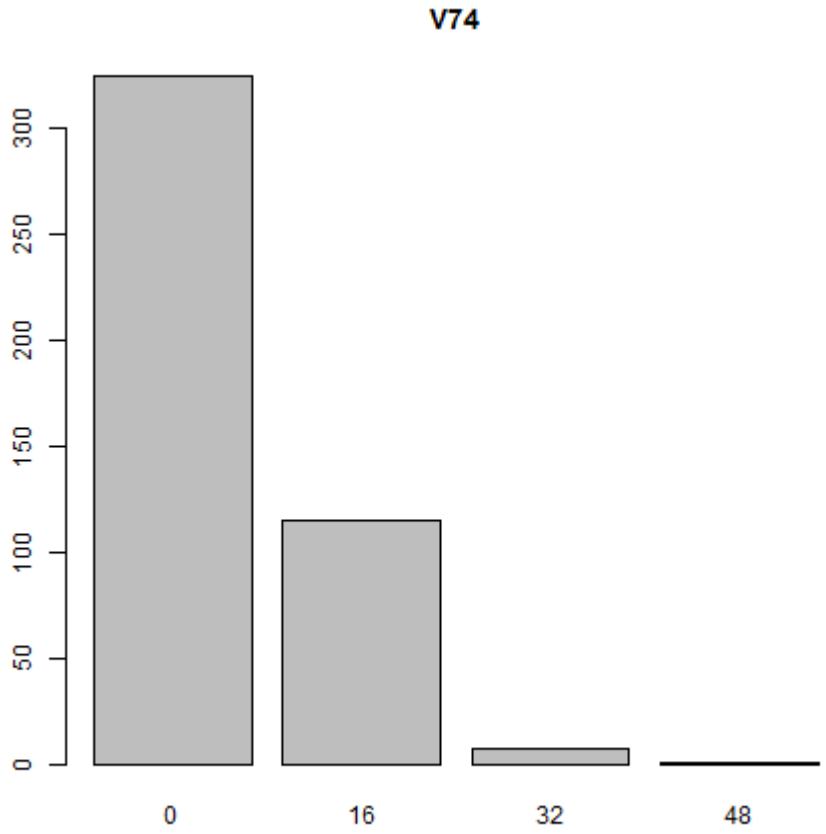

V79
20000
15000
10000
5000
0
bCPvWye
jySVZNIOJy
LM8l689qOp
M_8D

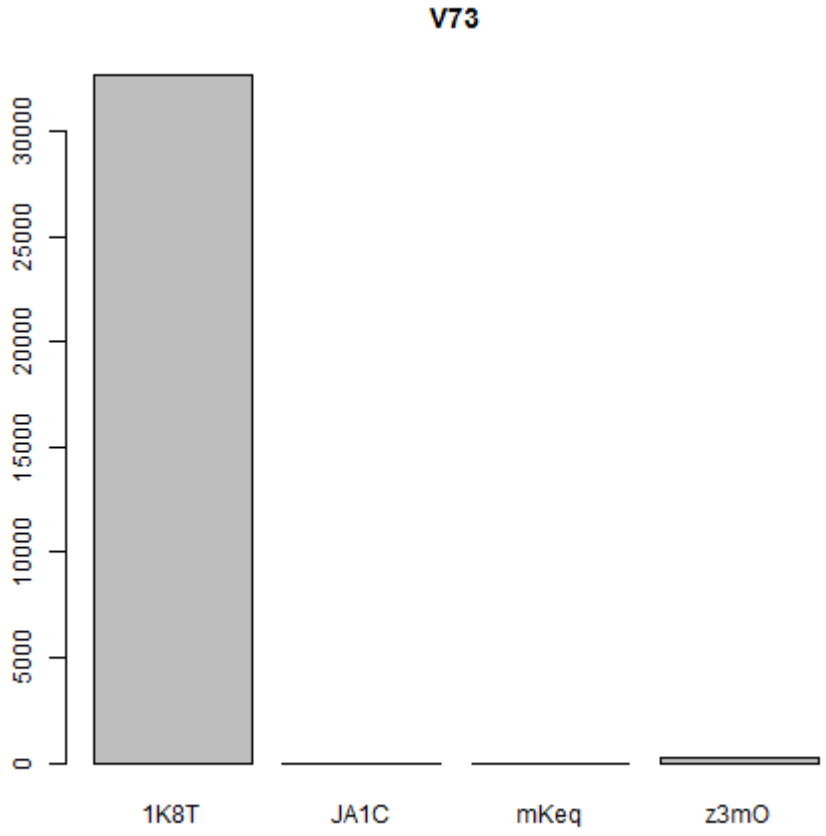

V78
400
300
200
100
0
0
70
168
266
364
476
588
700
868
1092
2226

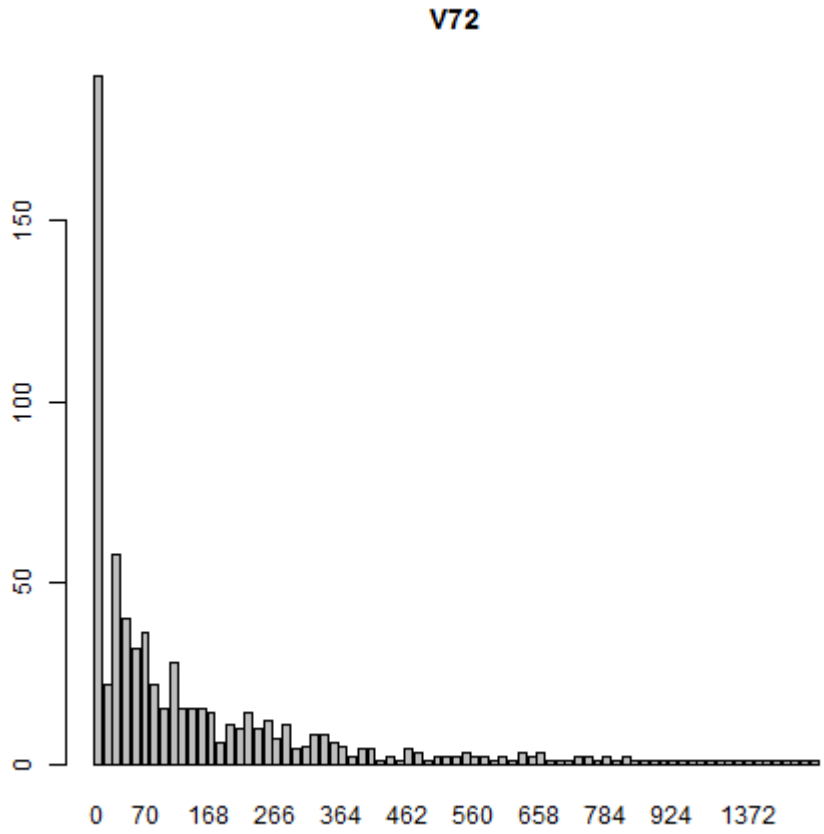

V77
600
500
400
300
200
100
0

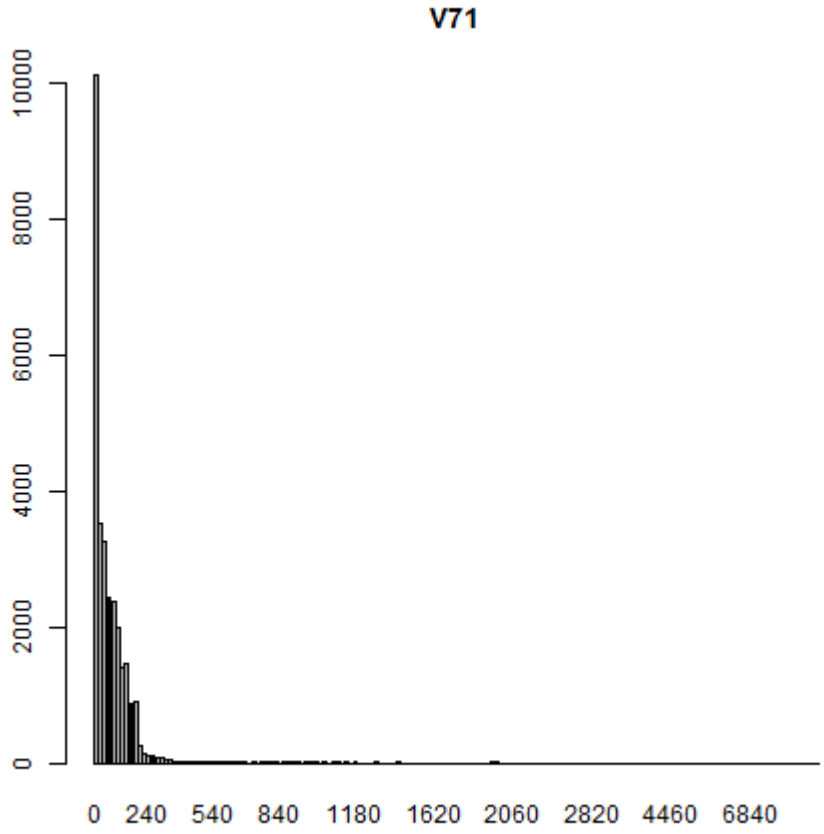

V76
0
100
200
300
400
500
4n2X

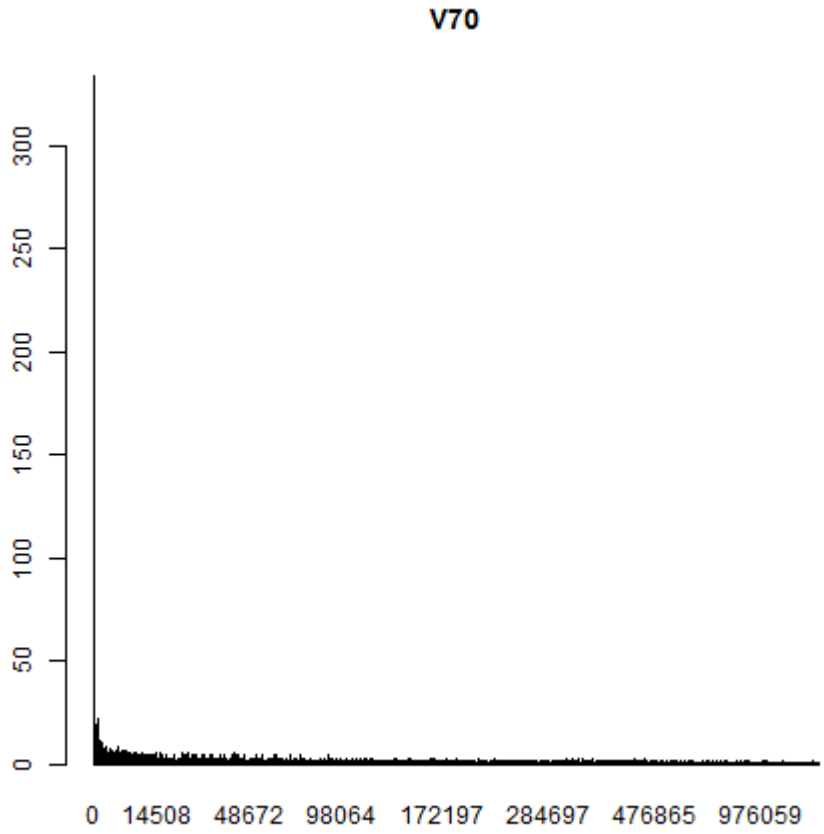

V75
0
1000
2000
3000
4000
0
272
608
944
1312
1728
2192
2848
3968
5920

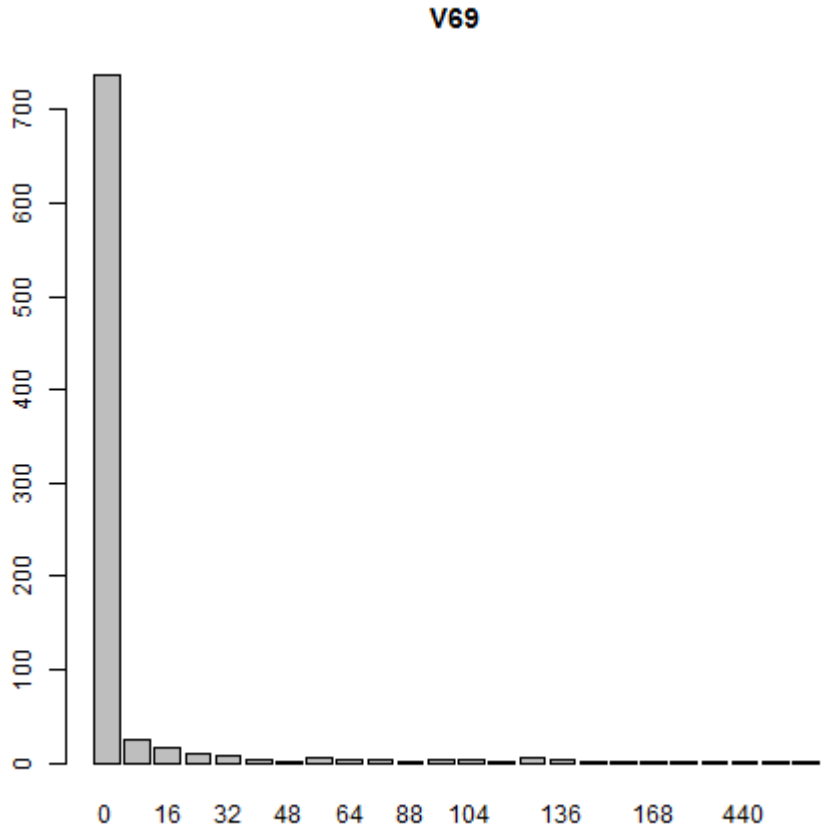

V74
0
50
100
150
200
250
300
0
16
32
48

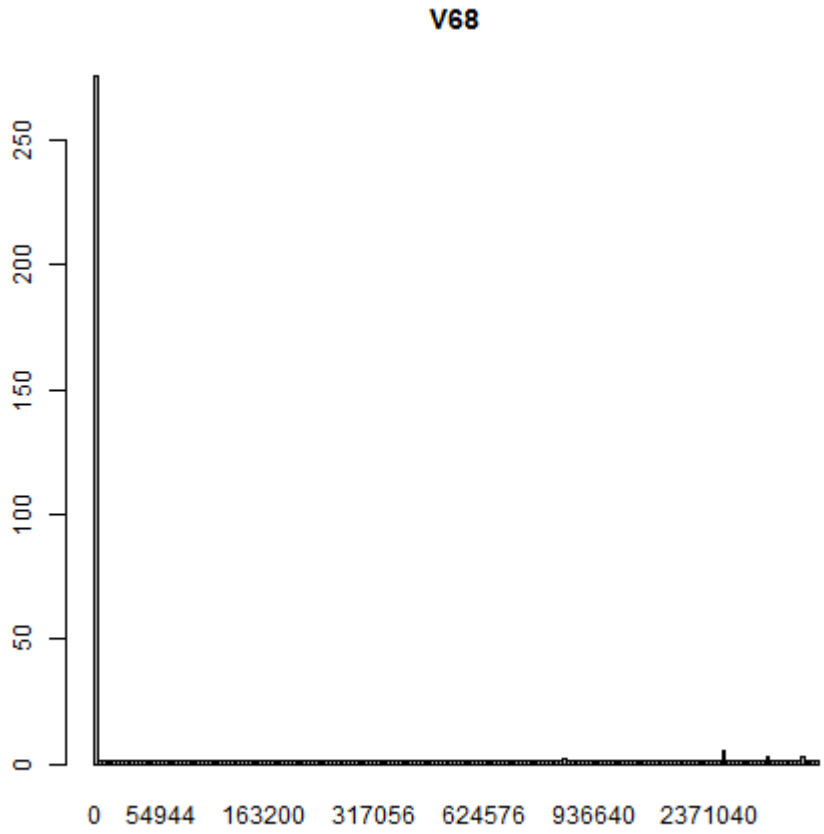

V73
0
5000
10000
15000
20000
25000
30000
1K8T
JA1C
mKeq
z3mO

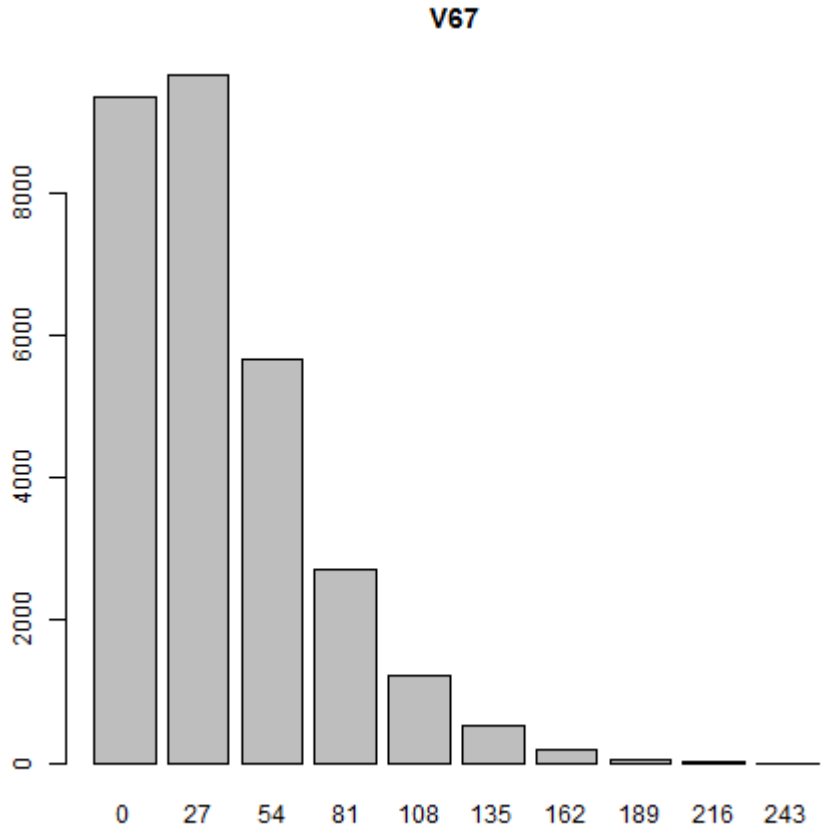

V72
150
100
50
0
0 70 168 266 364 462 560 658 784 924 1372

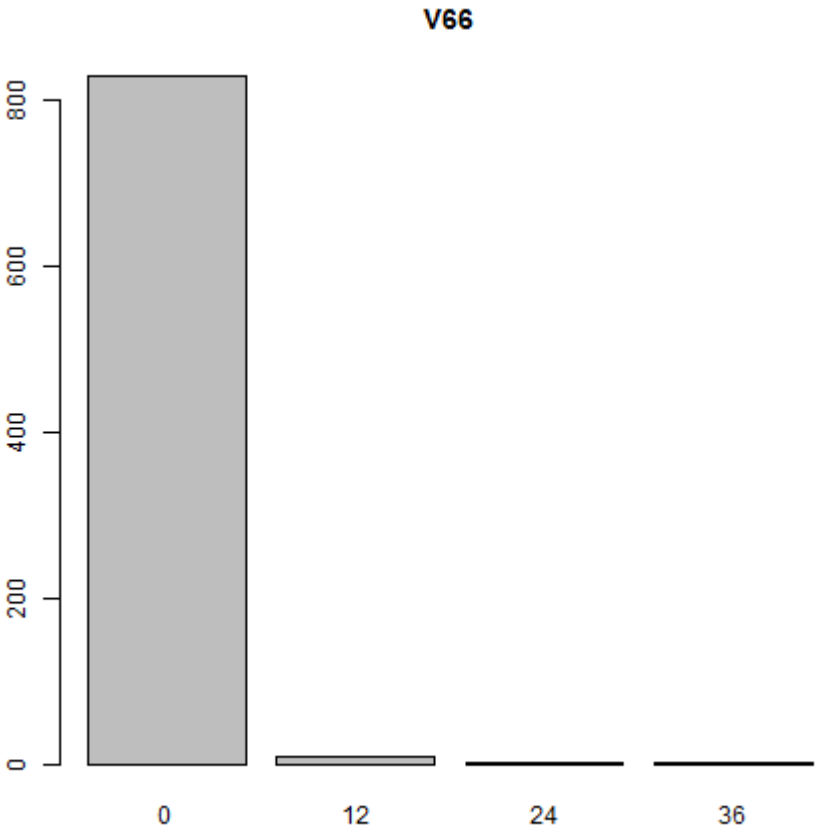

V71
10000
8000
6000
4000
2000
0
0 240 540 840 1180 1620 2060 2820 4460 6840

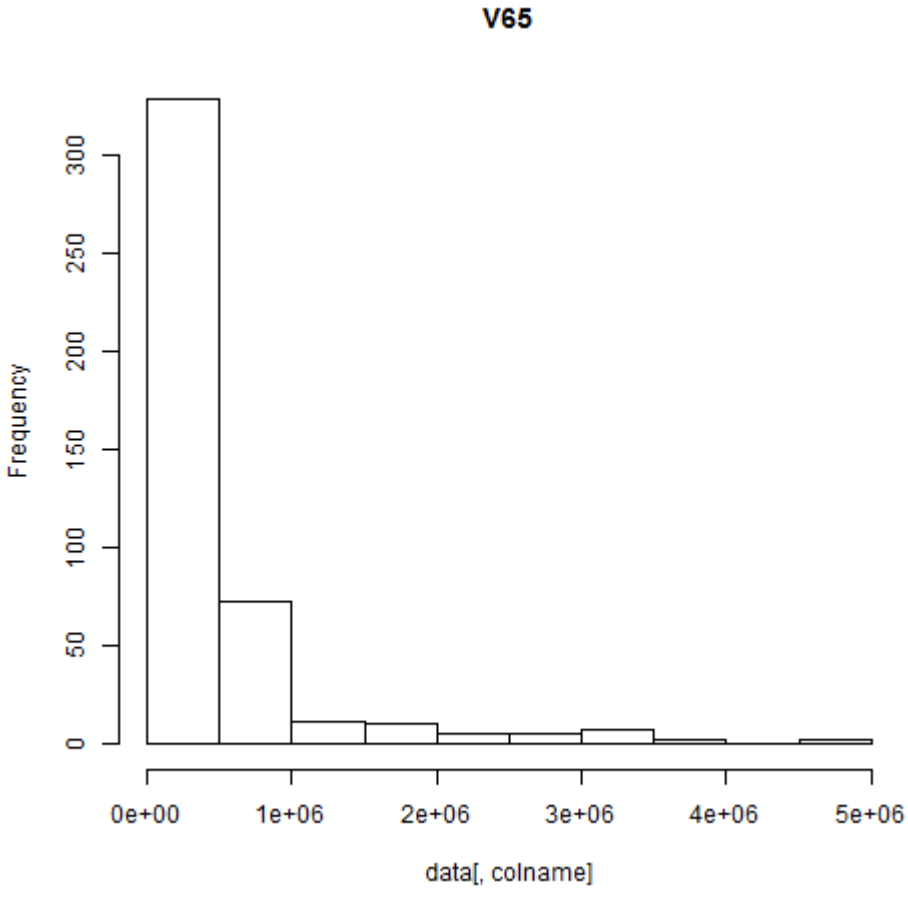

V70
0
50
100
150
200
250
300
0
14508
48672
98064
172197
284697
476865
976059

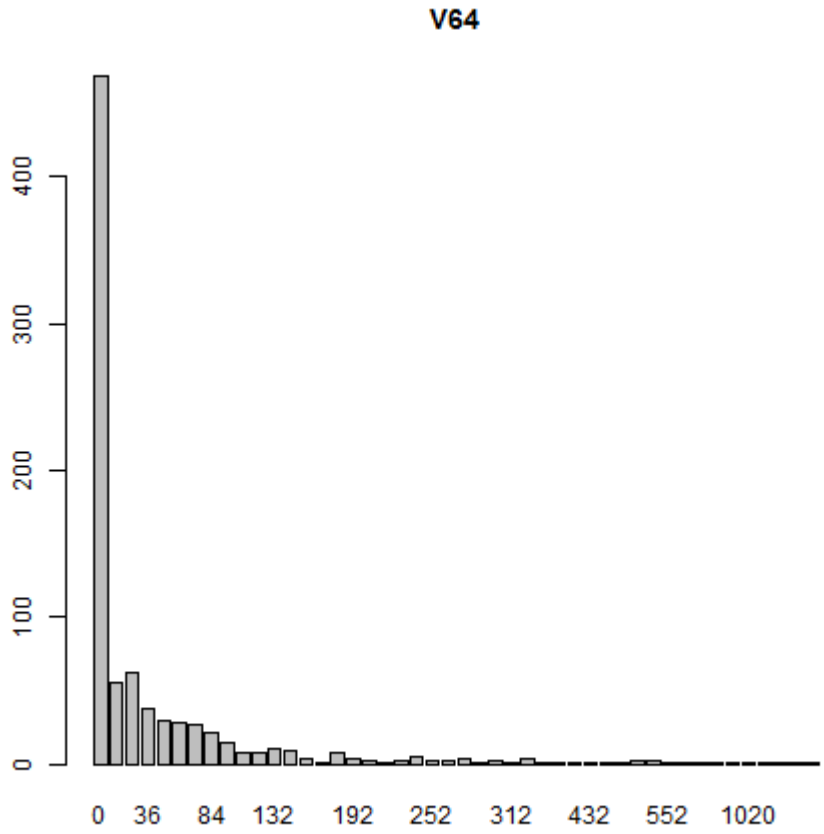

V69
0
100
200
300
400
500
600
700
0
16
32
48
64
88
104
136
168
440

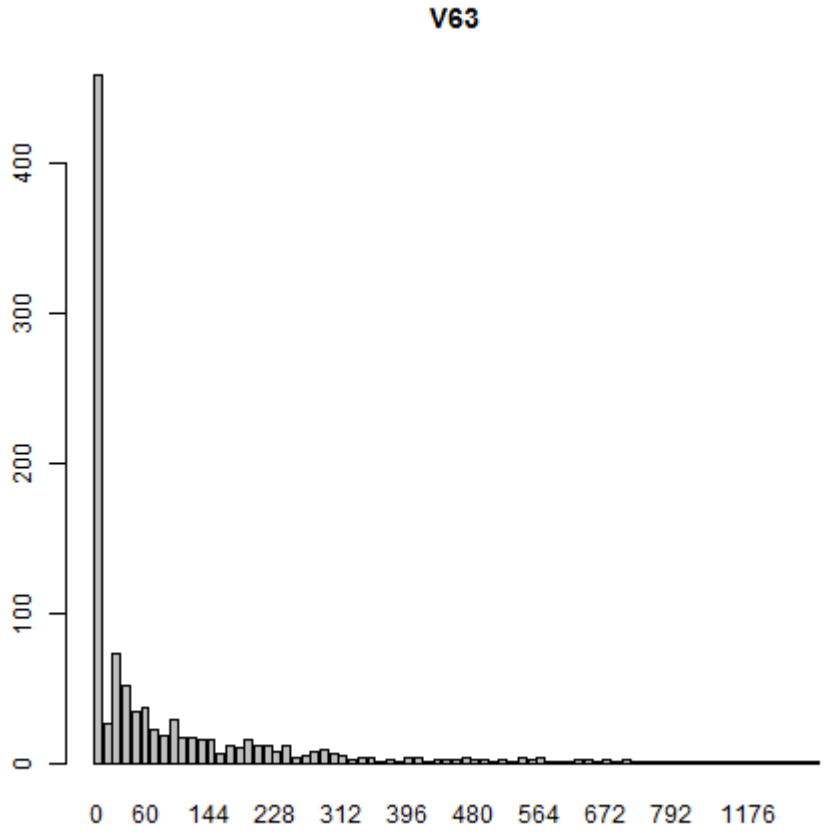

V68
0
50
100
150
200
250
0
54944
163200
317056
624576
936640
2371040

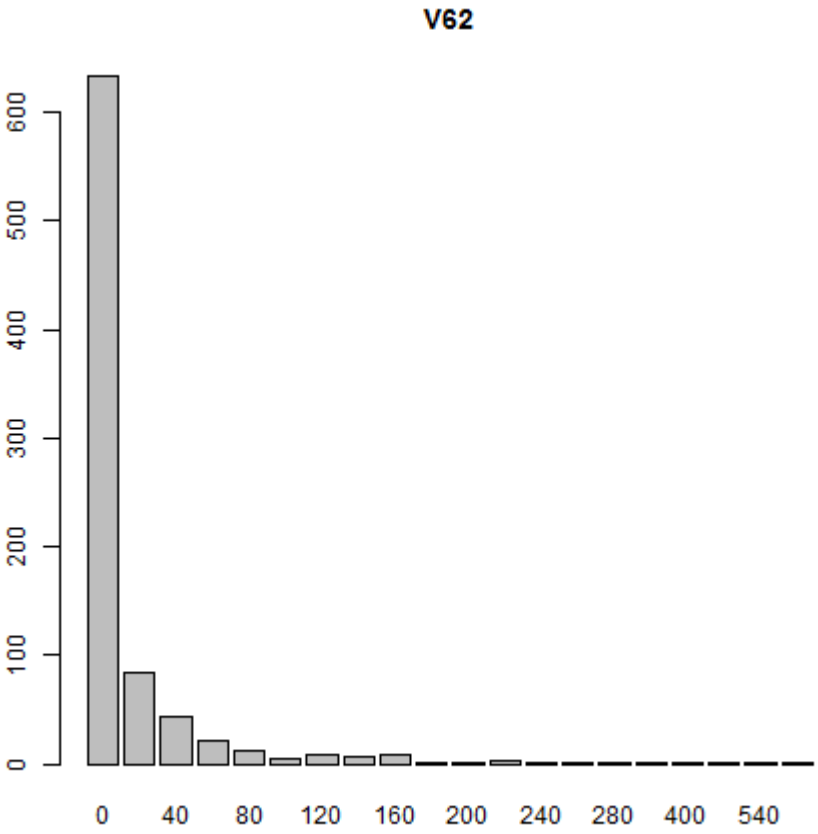

V67
0
2000
4000
6000
8000
0
27
54
81
108
135
162
189
216
243

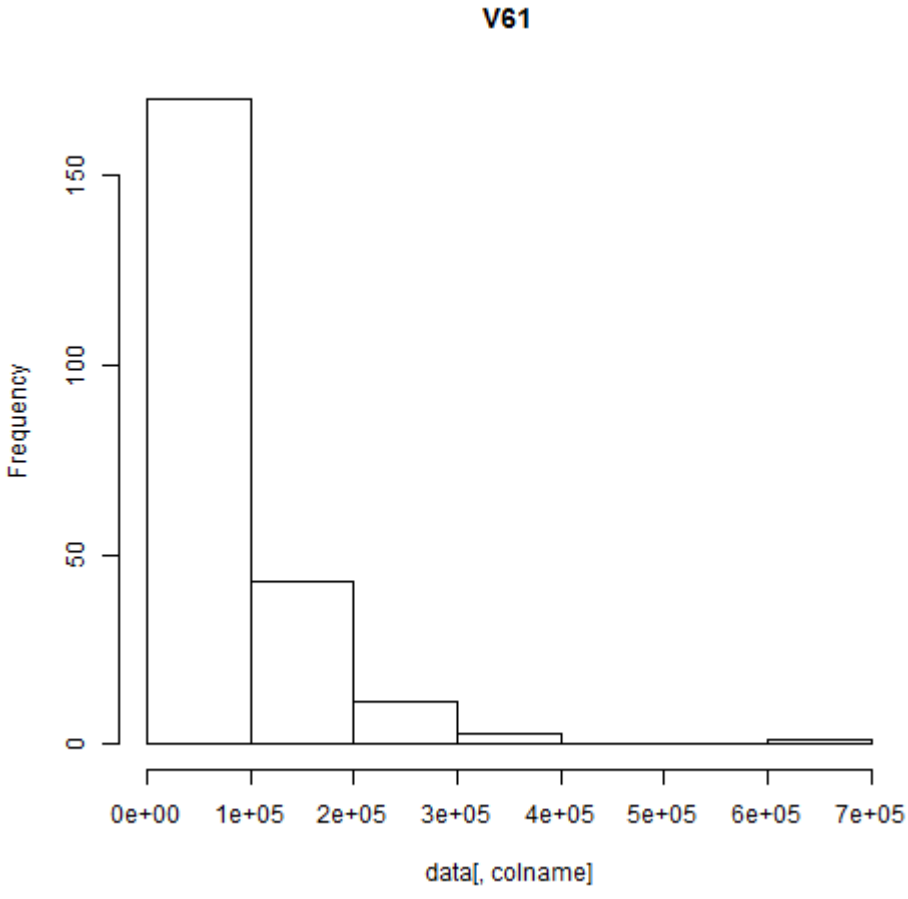

V66
800
600
400
200
0
0
12
24
36

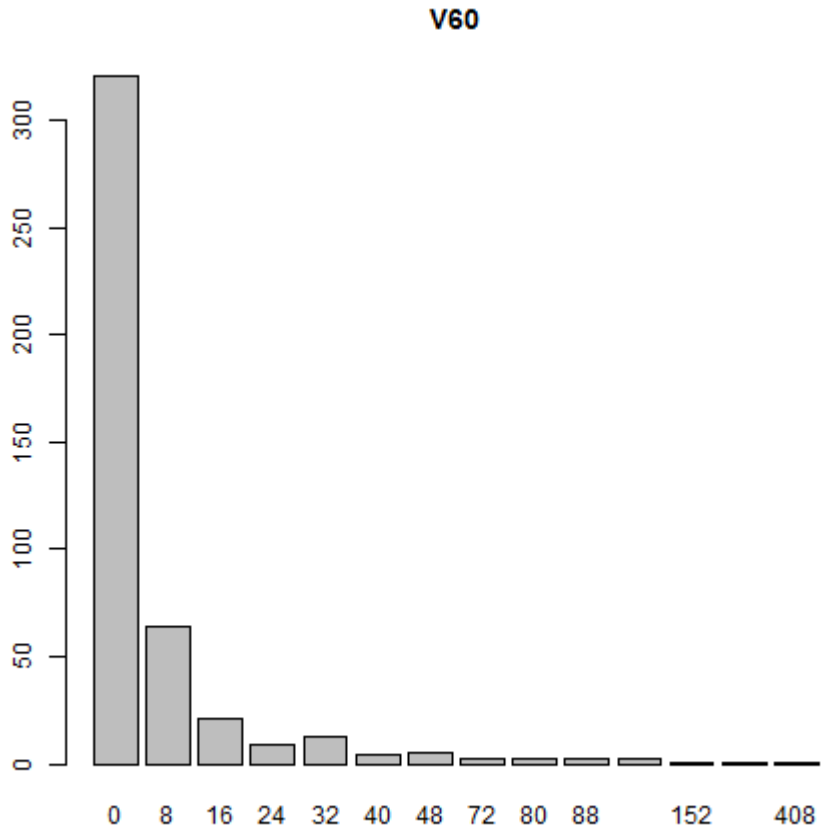

V65
Frequency
300
250
200
150
100
50
0
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06
data[, colname]

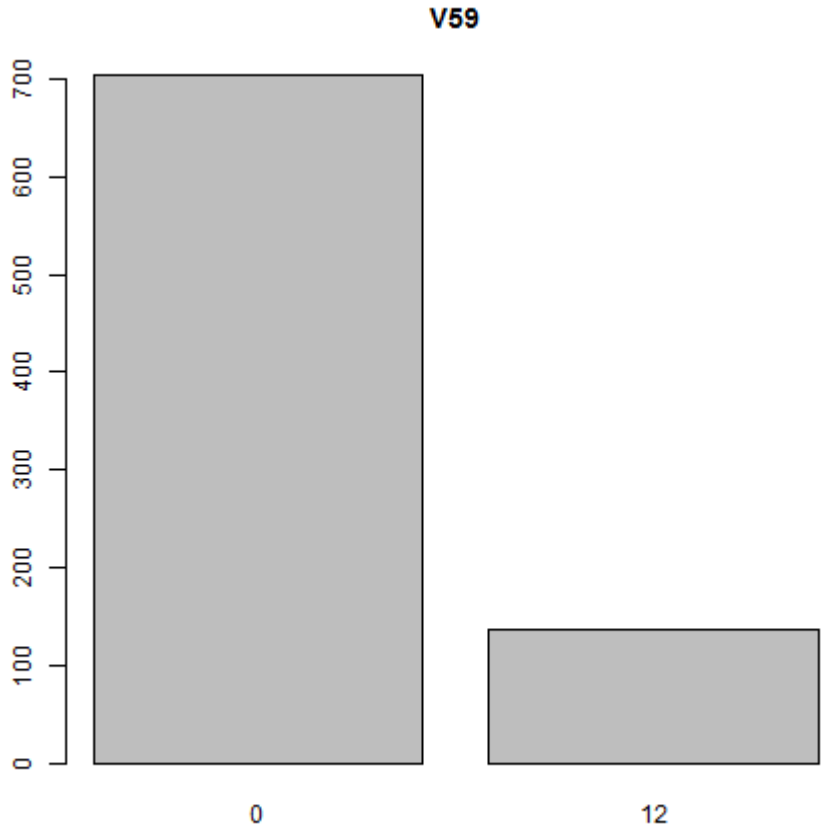

V64
400
300
200
100
0
0 36 84 132 192 252 312 432 552 1020

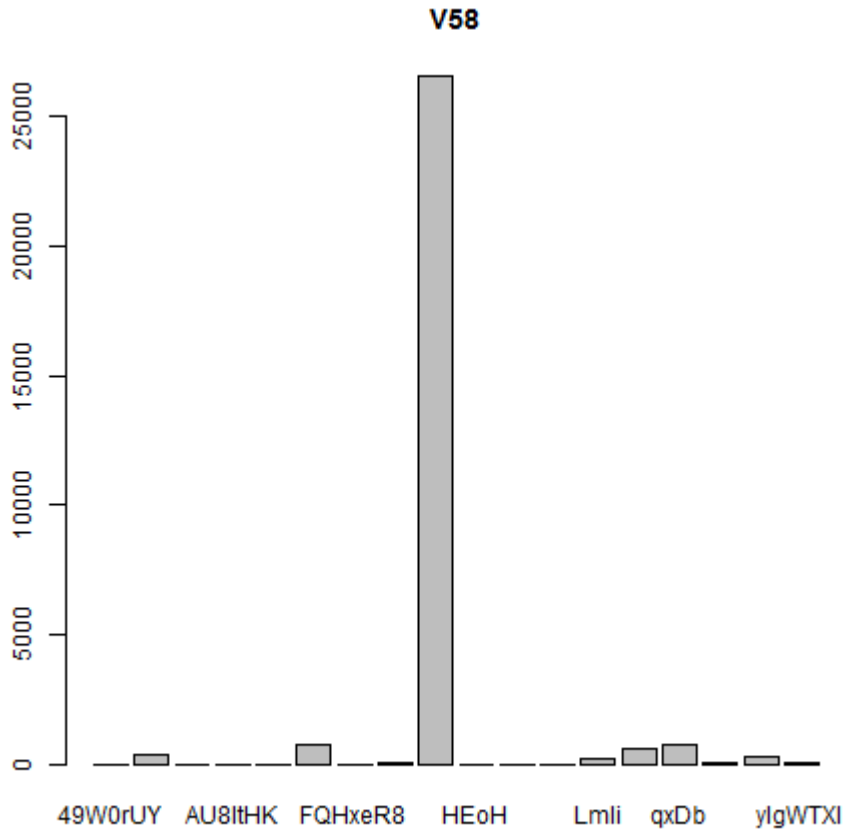

V63
400
300
200
100
0
0 60 144 228 312 396 480 564 672 792 1176

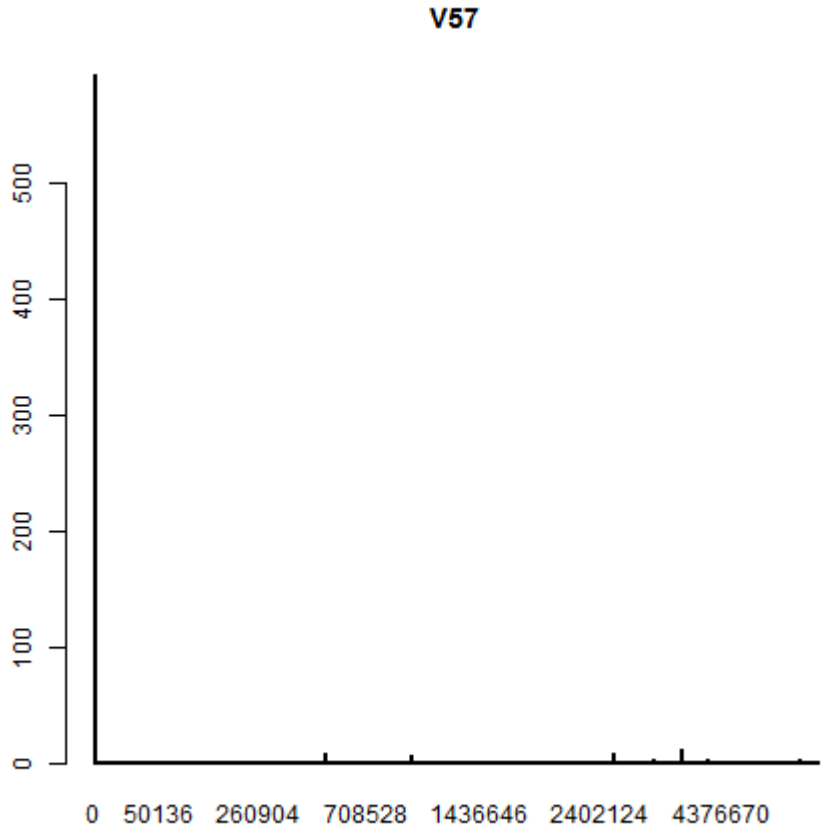

V62
600
500
400
300
200
100
0
0
40
80
120
160
200
240
280
400
540

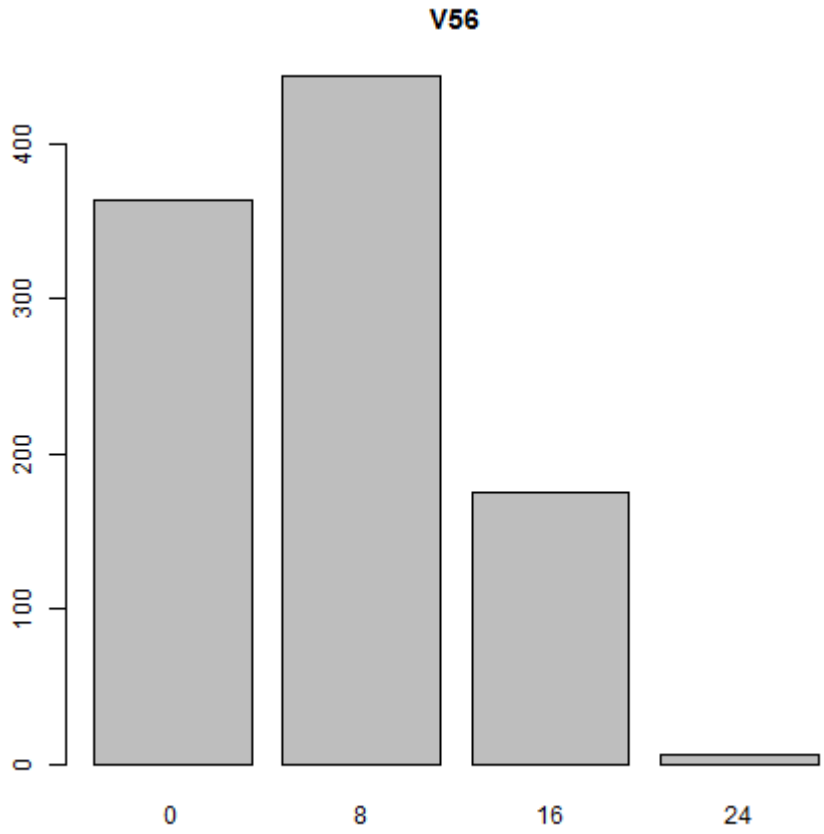

V61
Frequency
150
100
50
0
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
6e+05
7e+05
data[, colname]

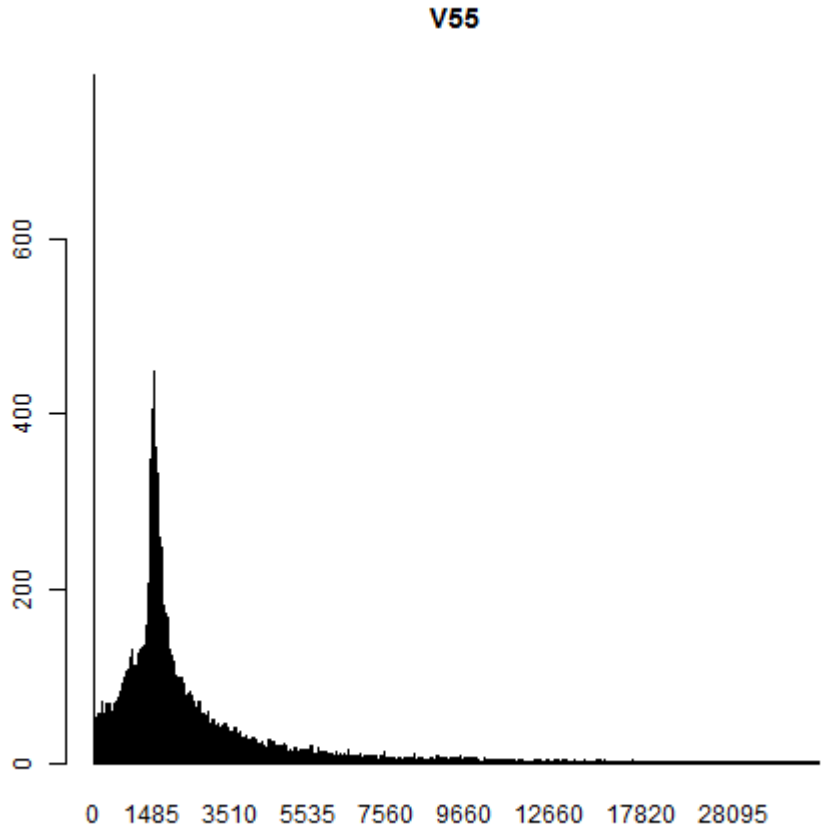

V60
0
50
100
150
200
250
300
0
8
16
24
32
40
48
72
80
88
152
408

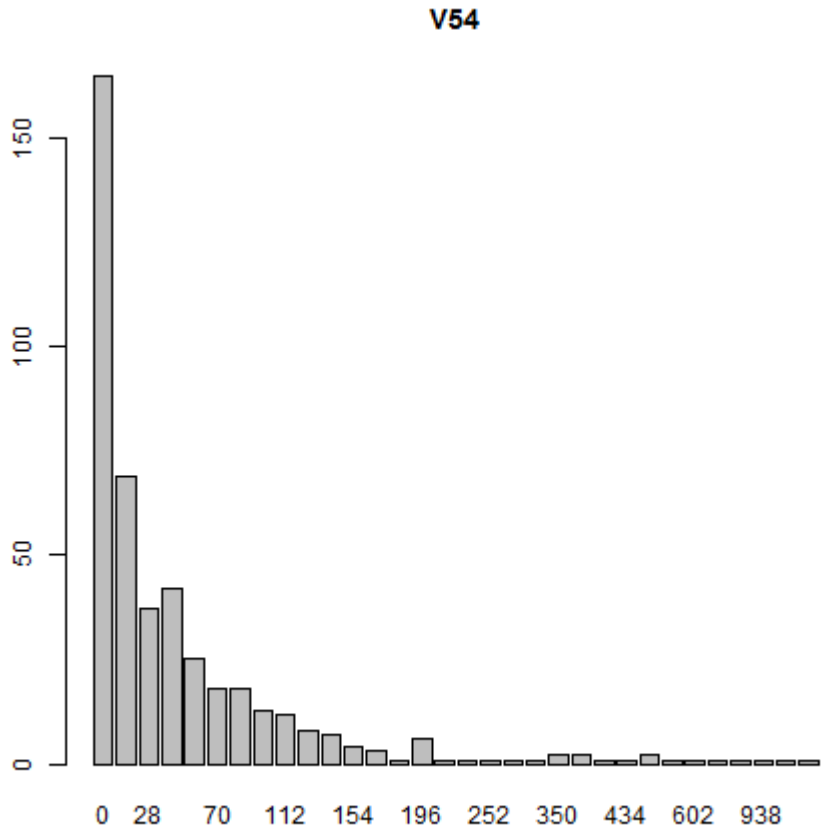

V59
0
100
200
300
400
500
600
700
0
12

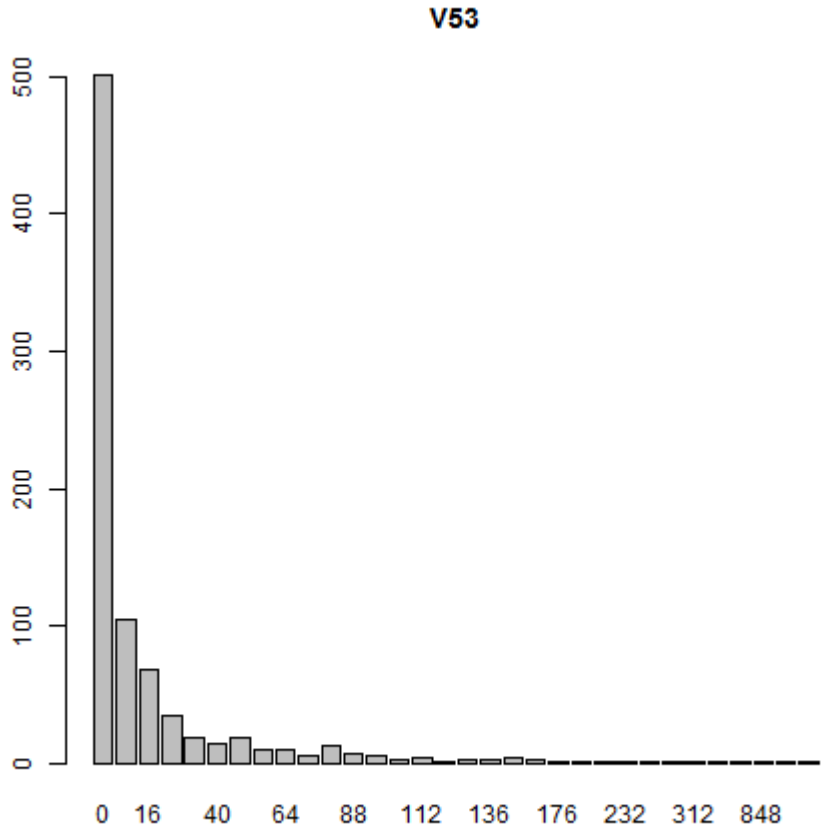

V58
25000
20000
15000
10000
5000
0
49W0rUY
AU8ltHK
FQHxeR8
HEoH
Lmli
qxDb
ylgWTXI

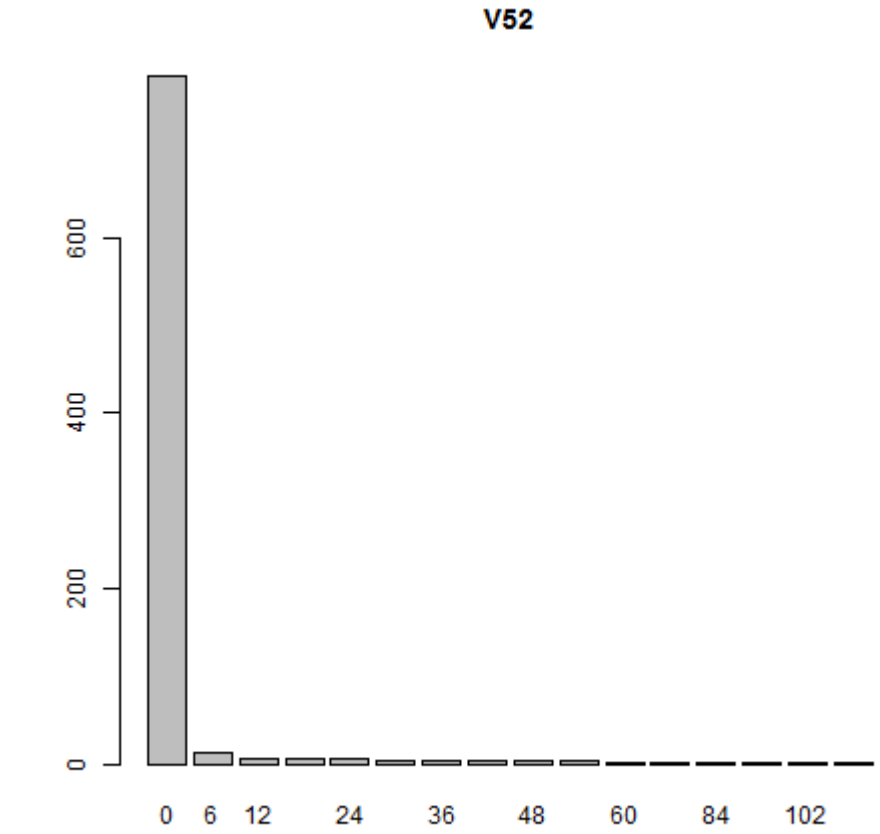

V57
500
400
300
200
100
0
0
50136
260904
708528
1436646
2402124
4376670

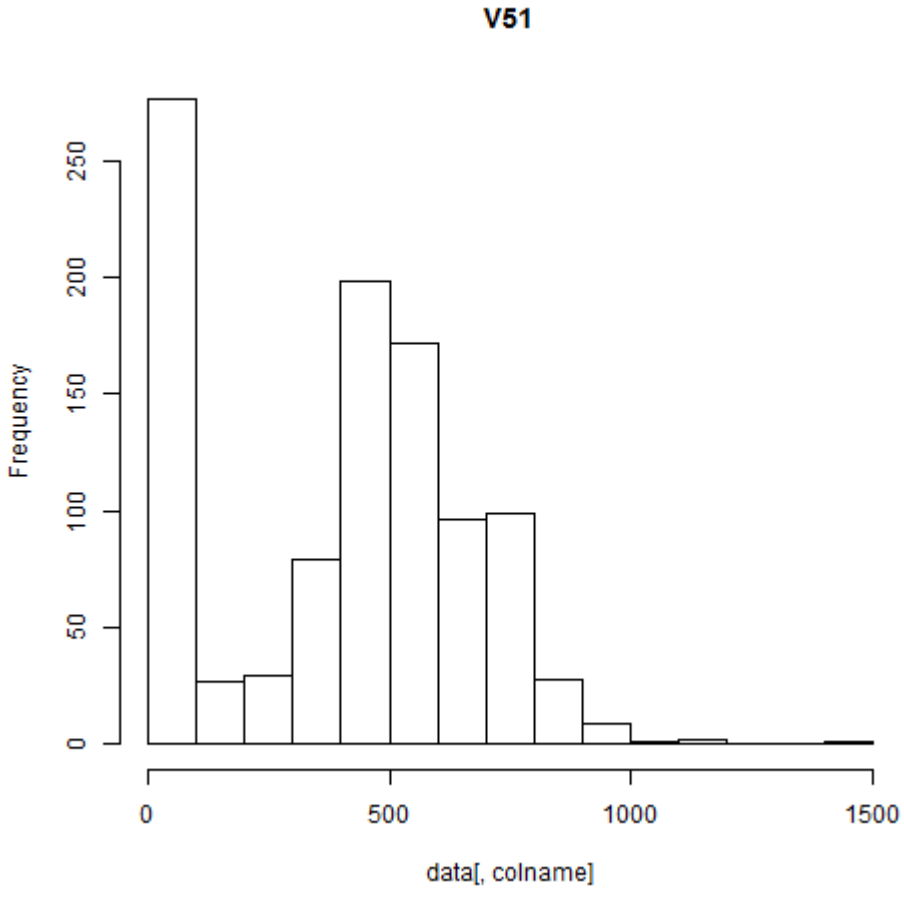

V56
400
300
200
100
0
0
8
16
24

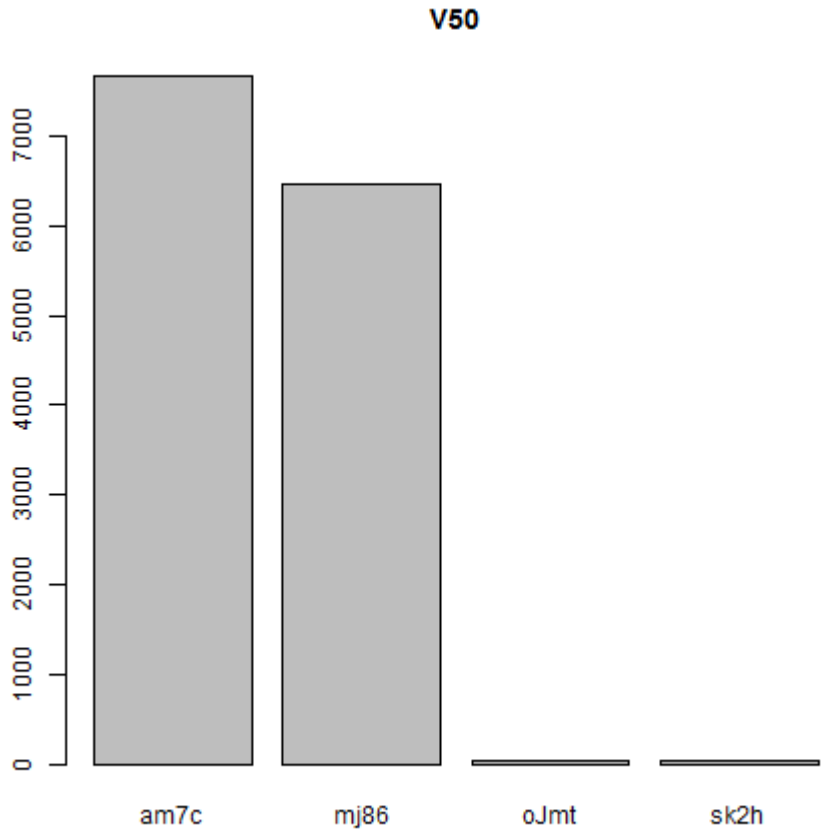

V55
600
400
200
0
0
1485
3510
5535
7560
9660
12660
17820
28095

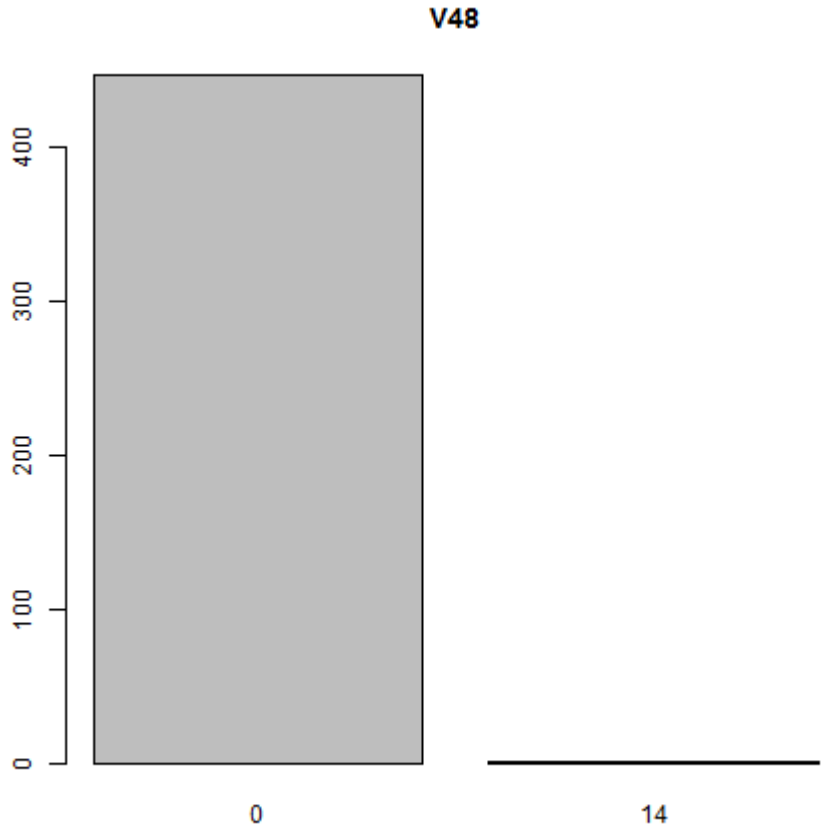

V54
150
100
50
0
0 28 70 112 154 196 252 350 434 602 938

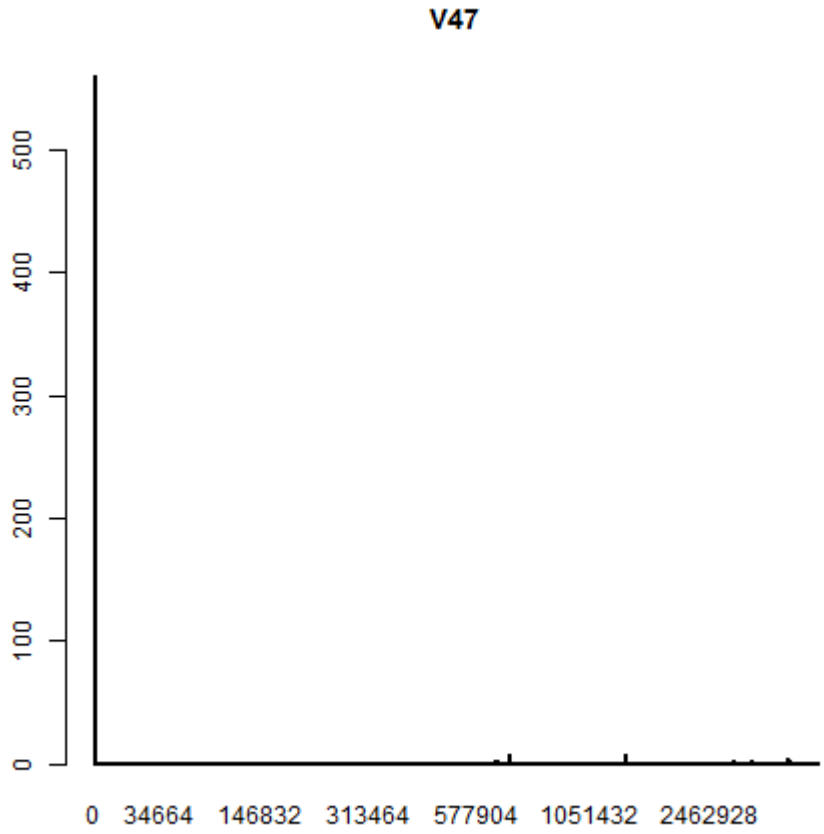

V53
500
400
300
200
100
0
0 16 40 64 88 112 136 176 232 312 848

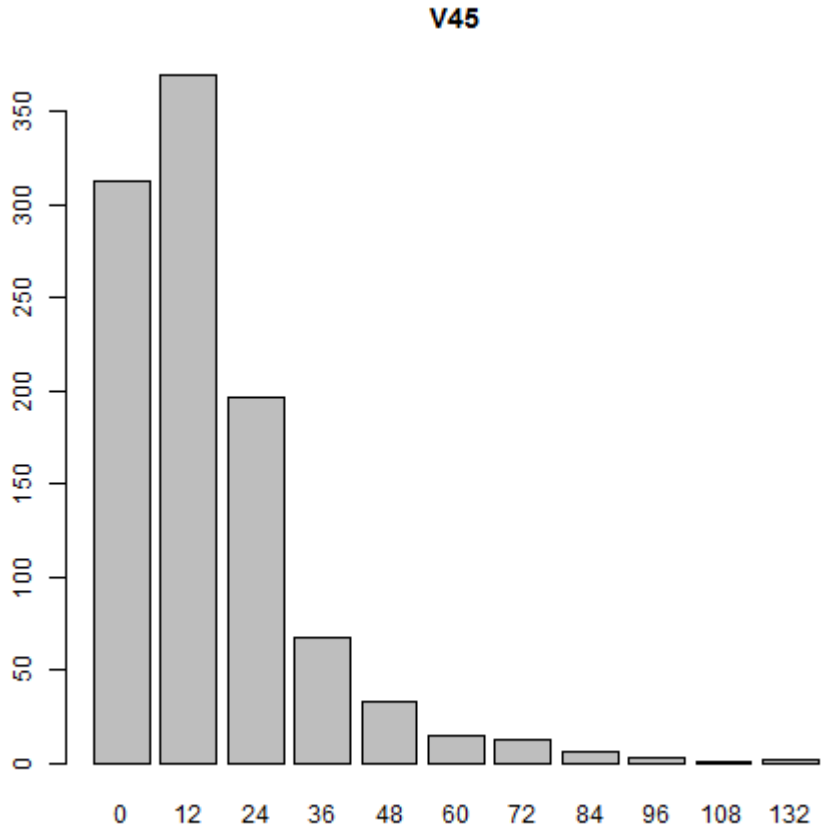

V52
0
200
400
600
0
6
12
24
36
48
60
84
102

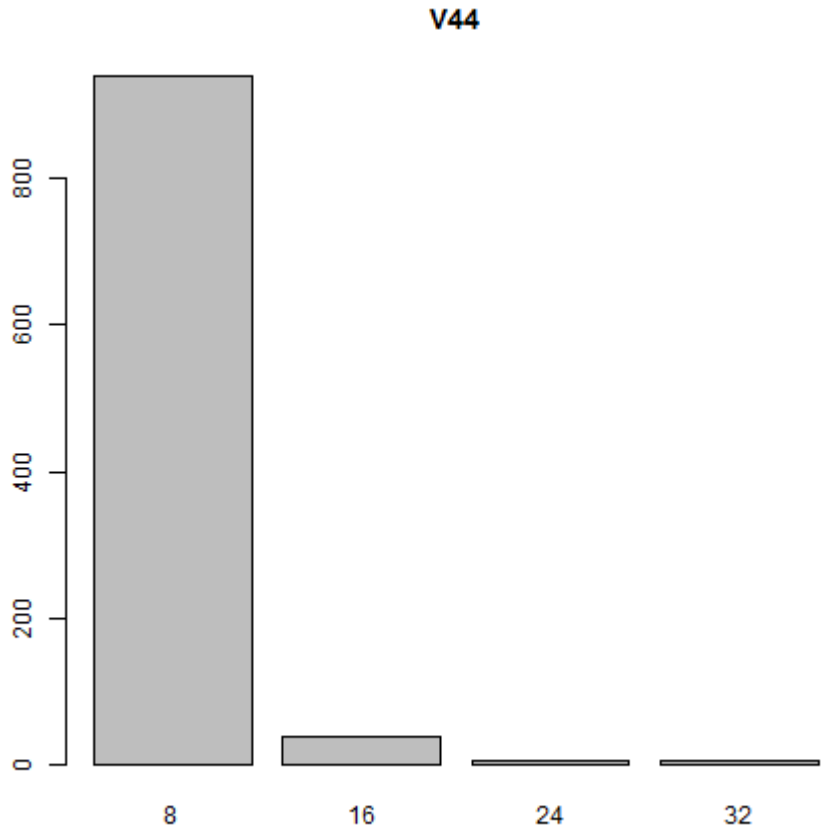

V51
Frequency
0
50
100
150
200
250
0
500
1000
1500
data[, colname]

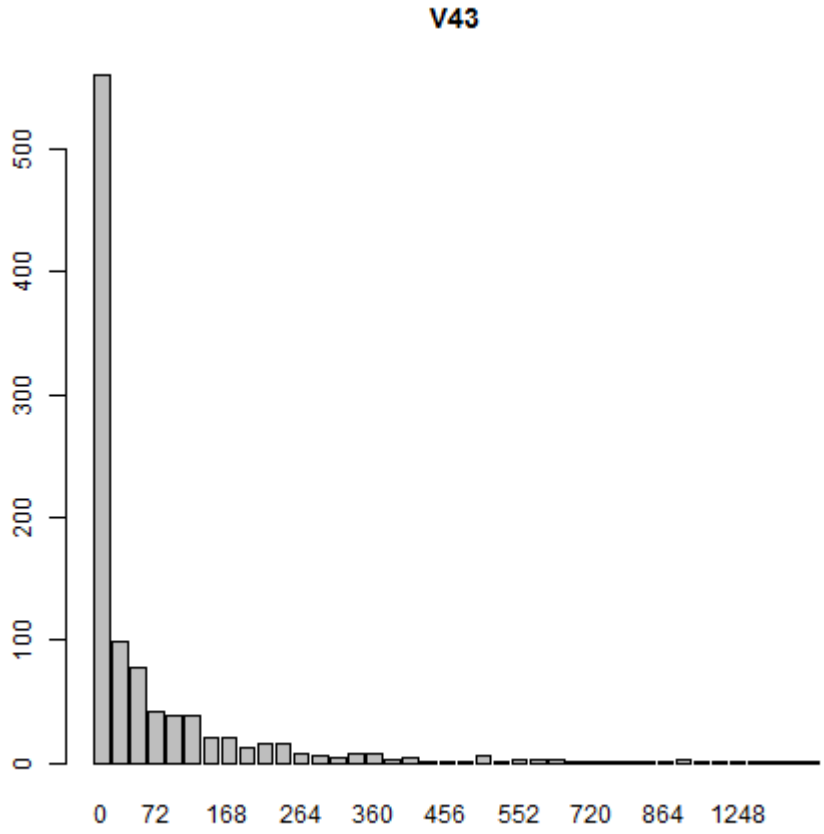

V50
0
1000
2000
3000
4000
5000
6000
7000
am7c
mj86
oJmt
sk2h

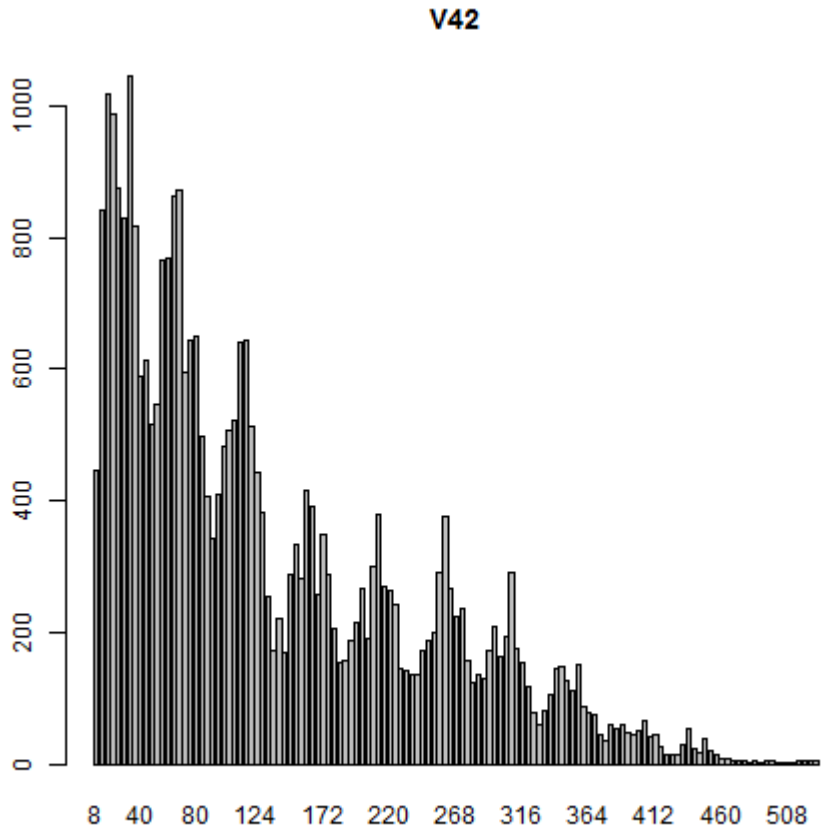

V48
0
100
200
300
400
0
14

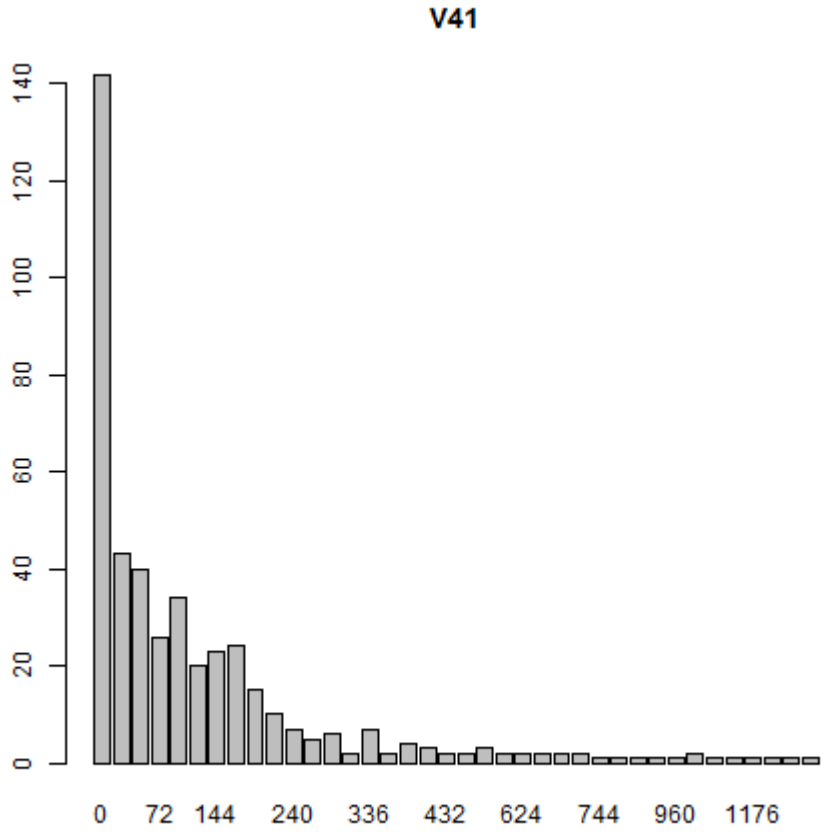

V47
0
100
200
300
400
500
0
34664
146832
313464
577904
1051432
2462928

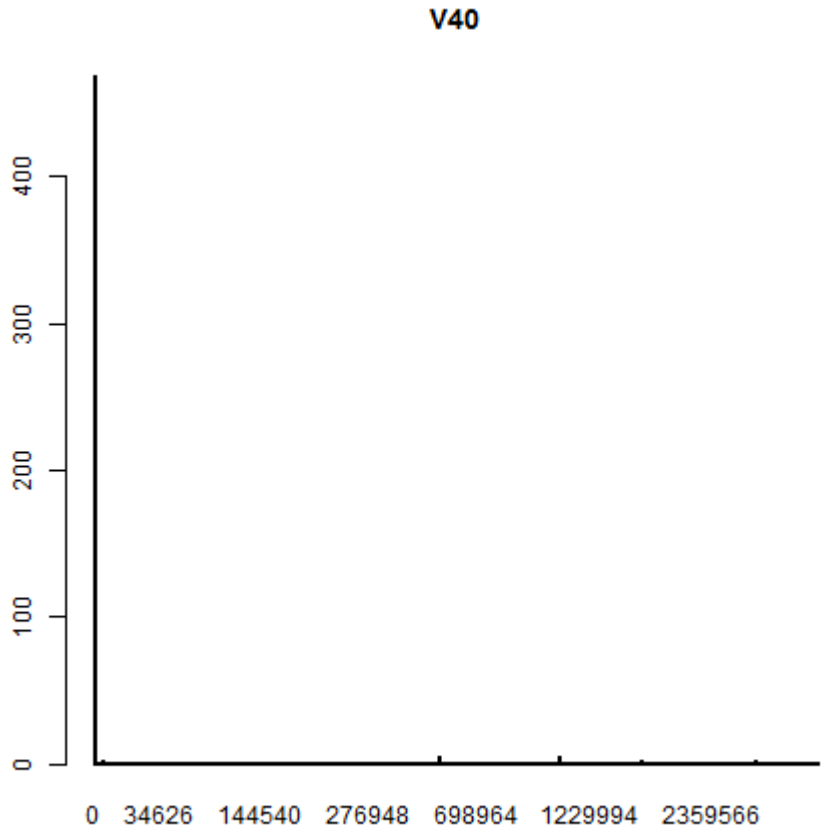

V45
0
50
100
150
200
250
300
350
0
12
24
36
48
60
72
84
96
108
132

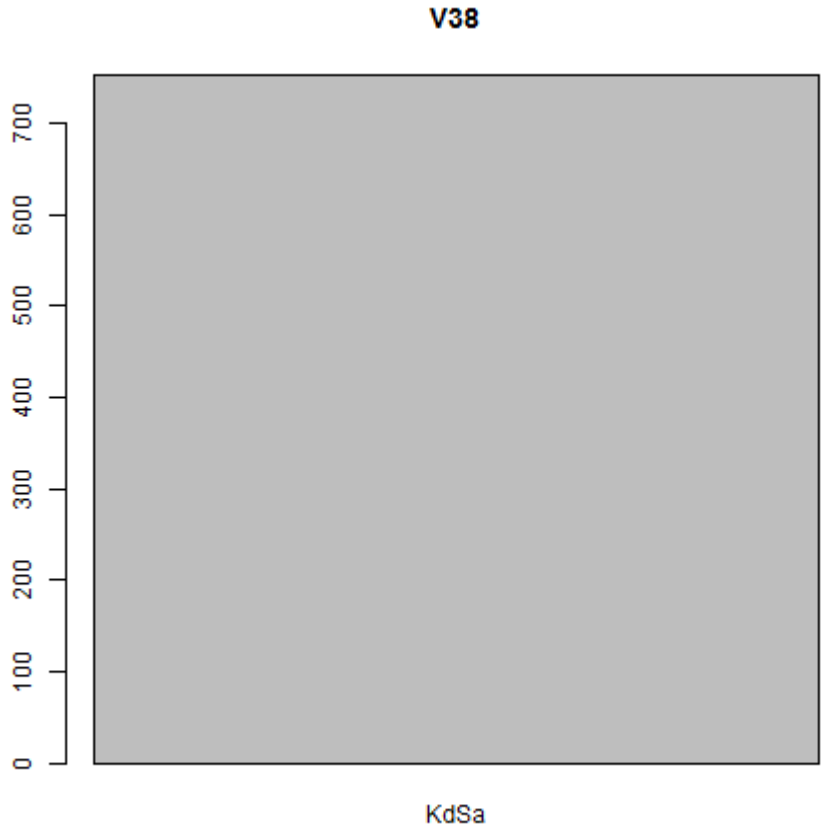

V44
0
200
400
600
800
8
16
24
32

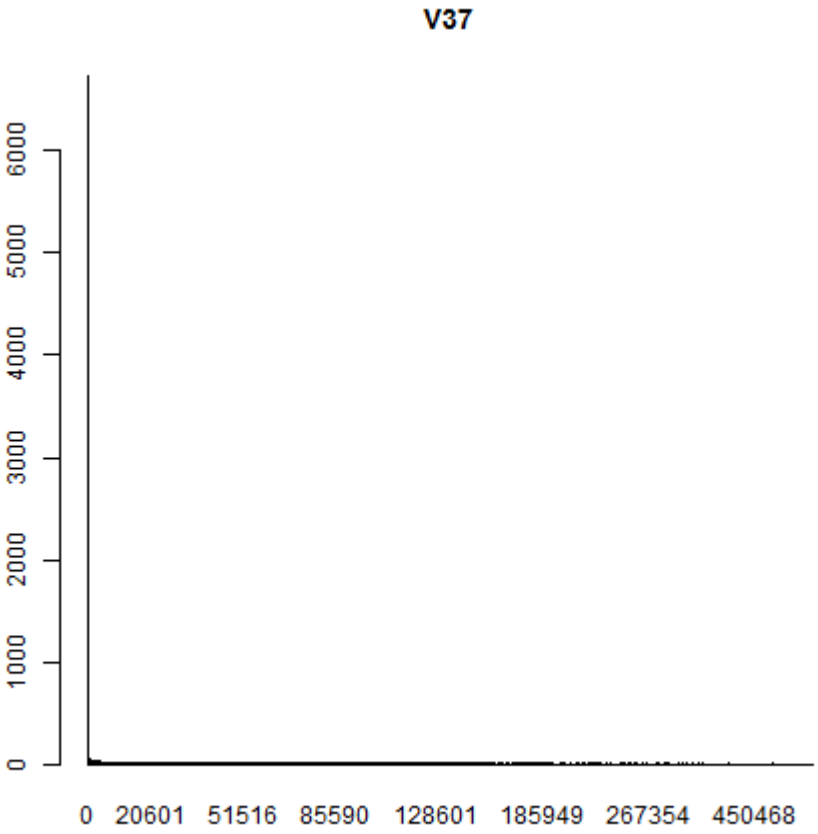

V43
0
100
200
300
400
500
0
72
168
264
360
456
552
720
864
1248

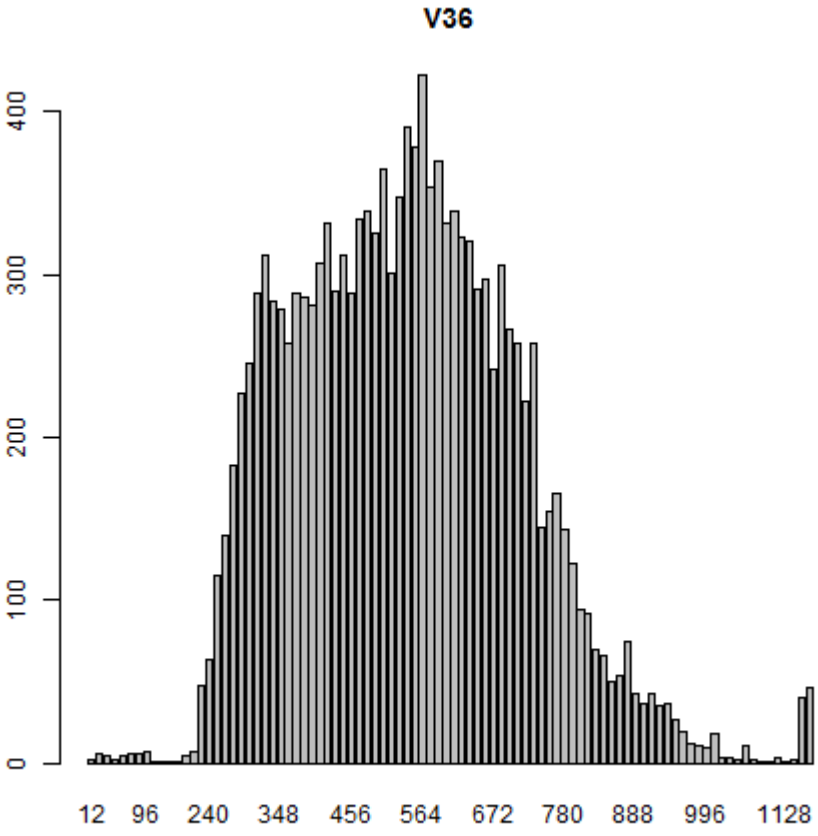

V42
0
200
400
600
800
1000
8
40
80
124
172
220
268
316
364
412
460
508

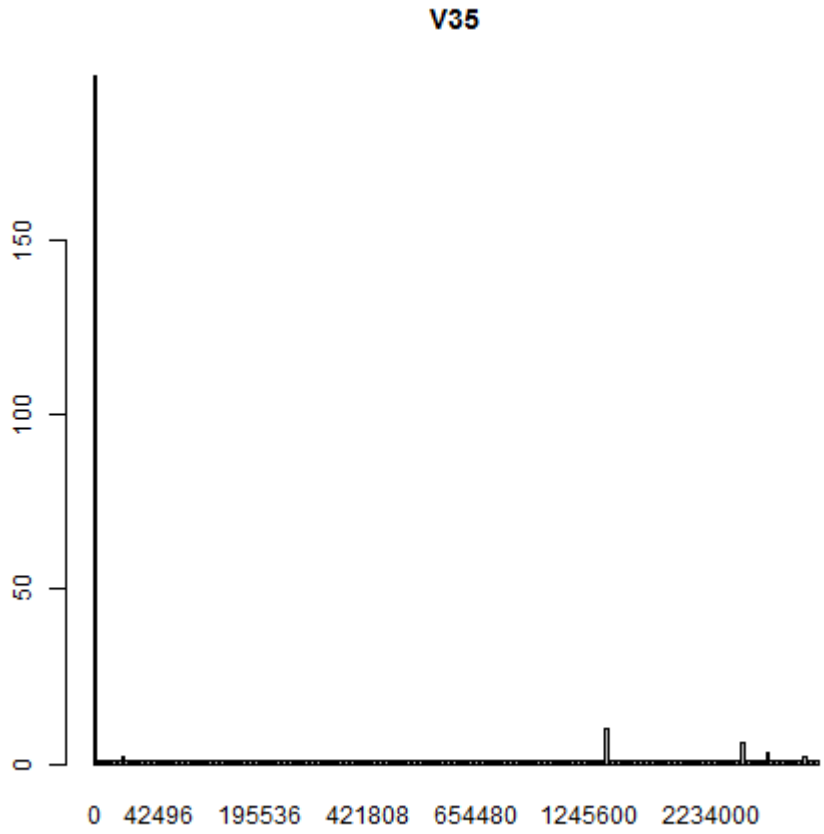

V41
0
20
40
60
80
100
120
140
0
72
144
240
336
432
624
744
960
1176

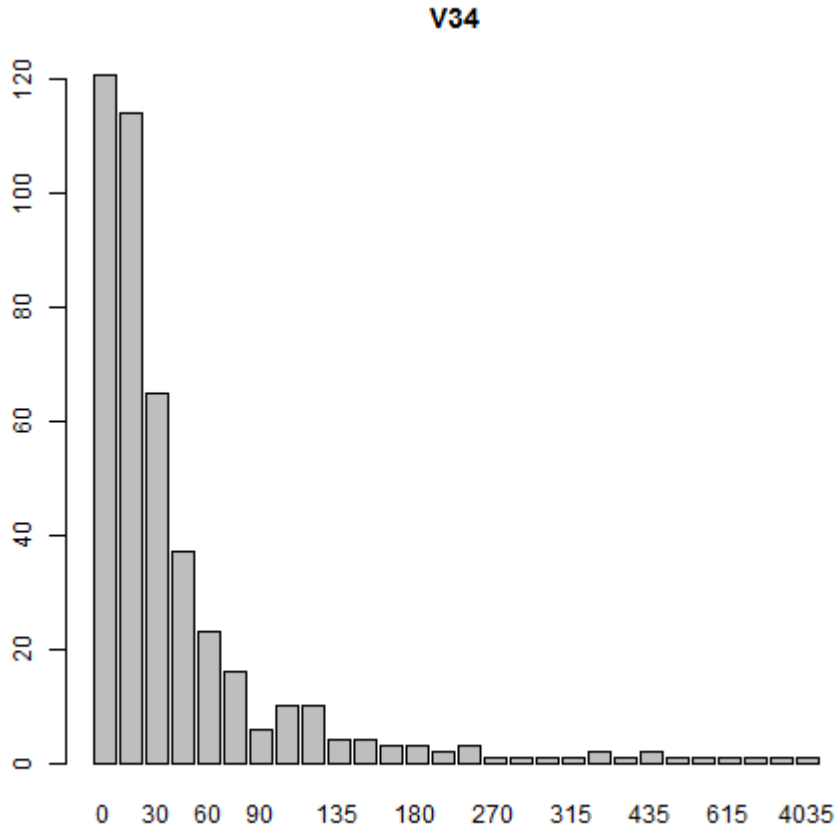

V40
0
100
200
300
400
0
34626
144540
276948
698964
1229994
2359566

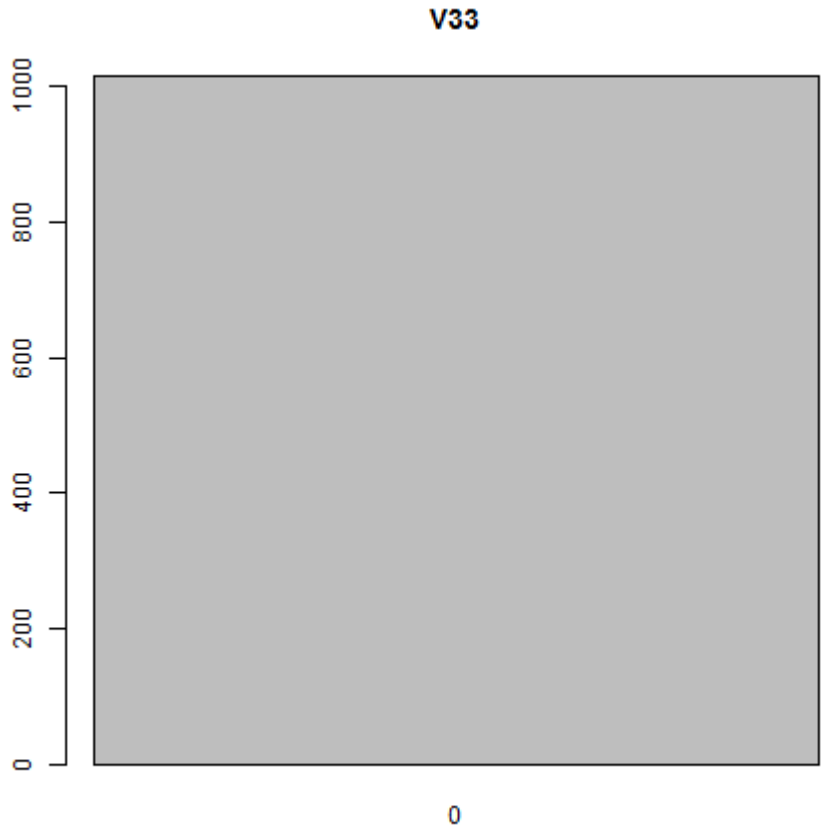

V38
0
100
200
300
400
500
600
700
KdSa

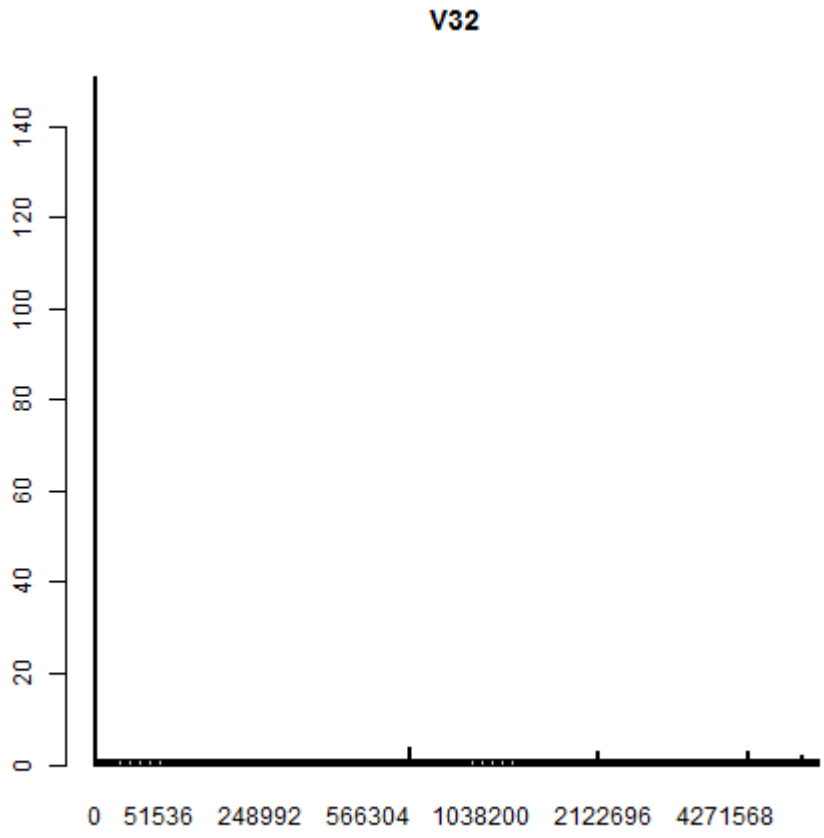

V37
0
1000
2000
3000
4000
5000
6000
0
20601
51516
85590
128601
185949
267354
450468

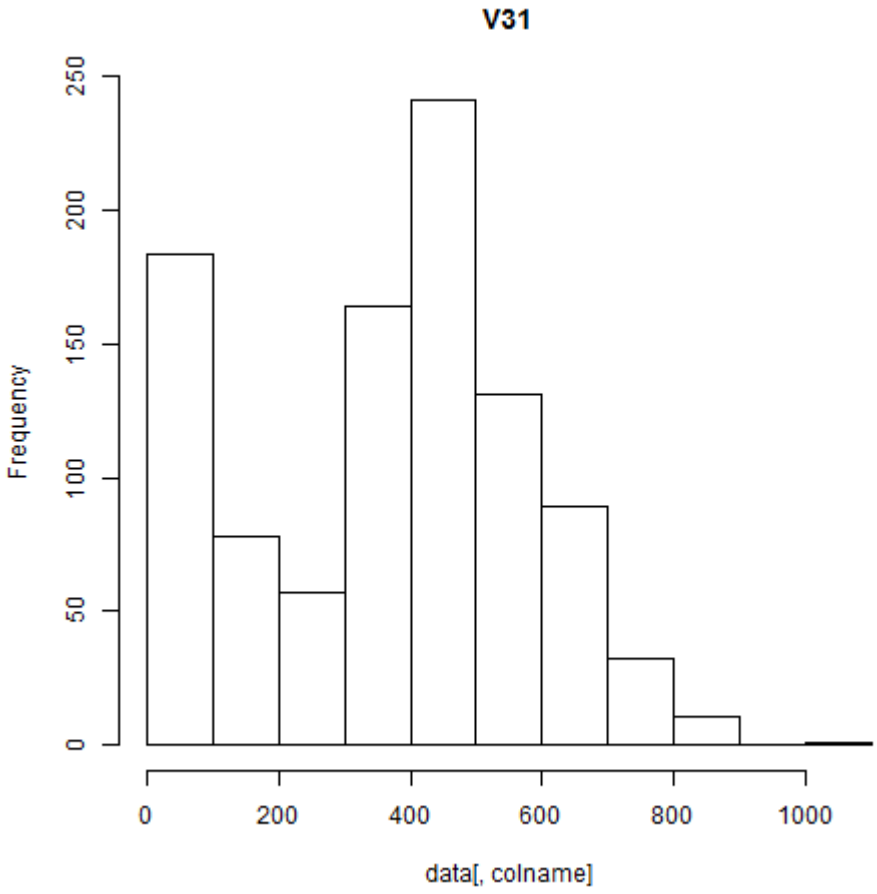

V36
0
100
200
300
400
12
96
240
348
456
564
672
780
888
996
1128

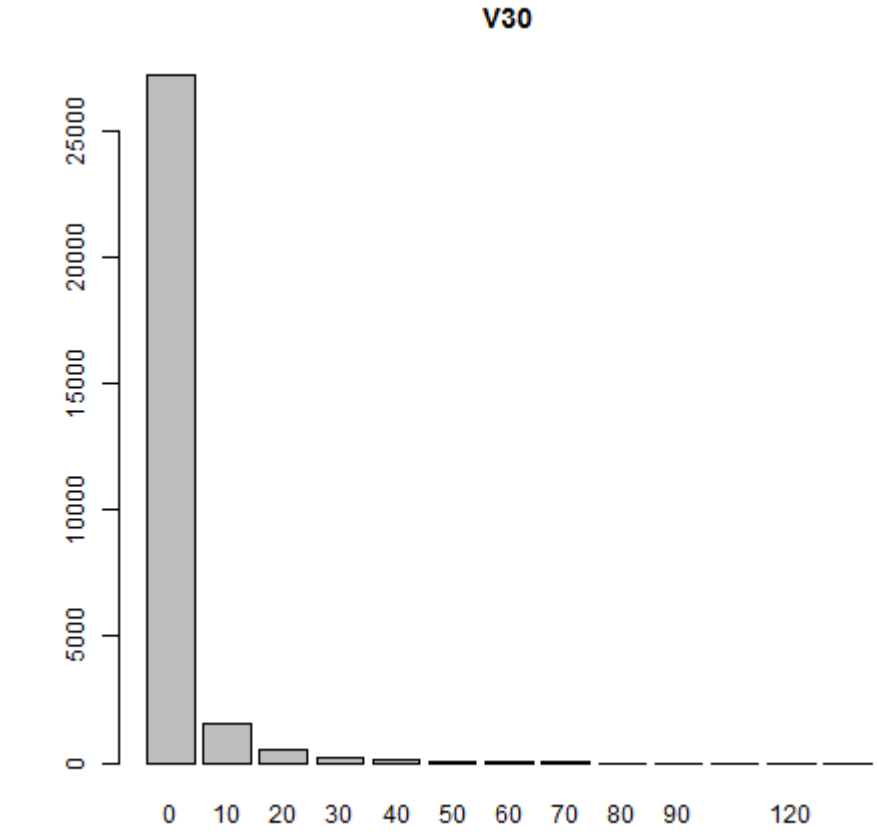

V35
0
50
100
150
0
42496
195536
421808
654480
1245600
2234000

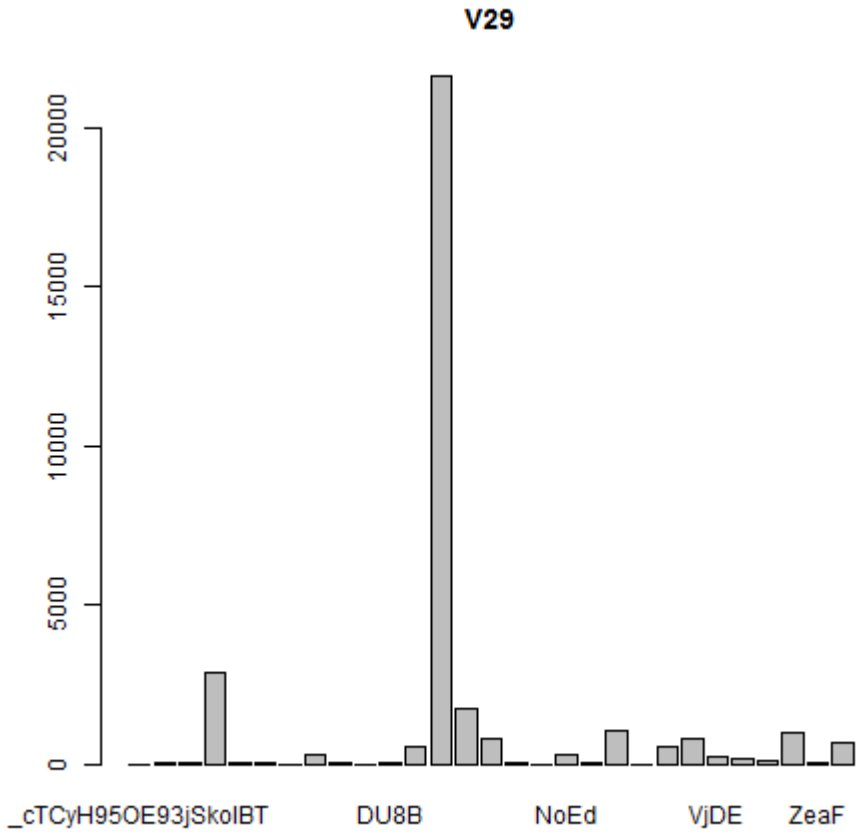

V34
0
20
40
60
80
100
120
0
30
60
90
135
180
270
315
435
615
4035

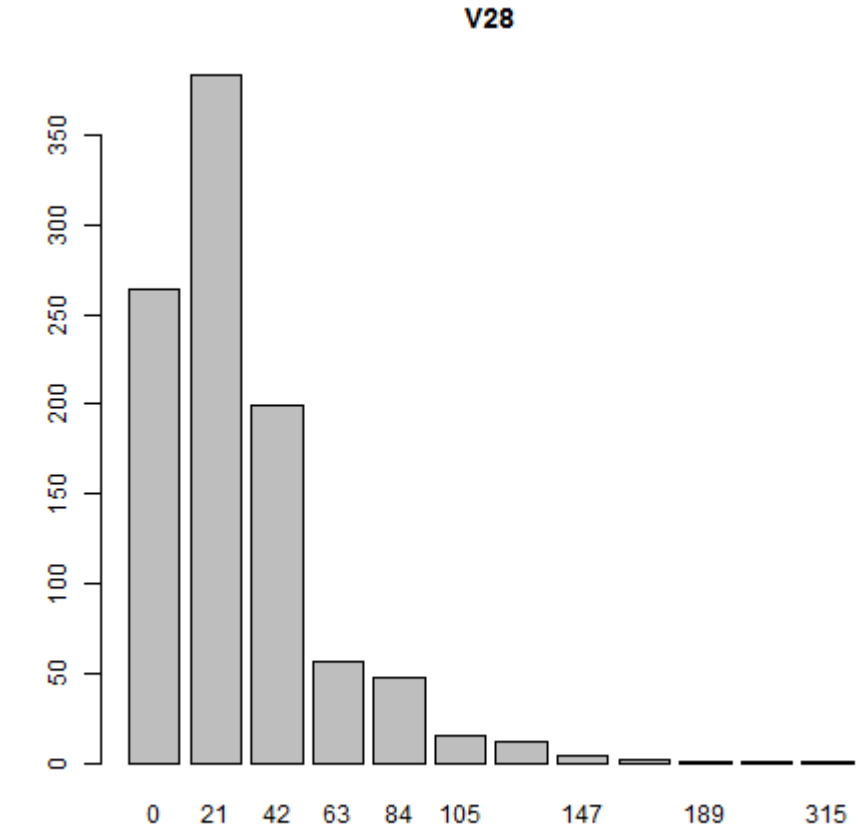

V33
1000
800
600
400
200
0
0

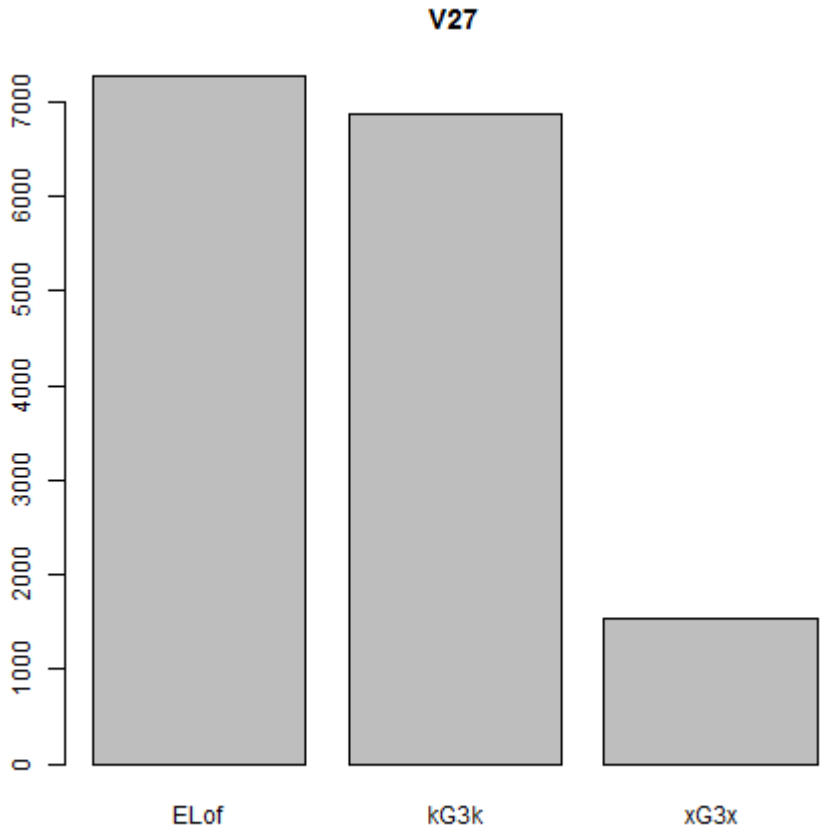

V32
140
120
100
80
60
40
20
0
0
51536
248992
566304
1038200
2122696
4271568

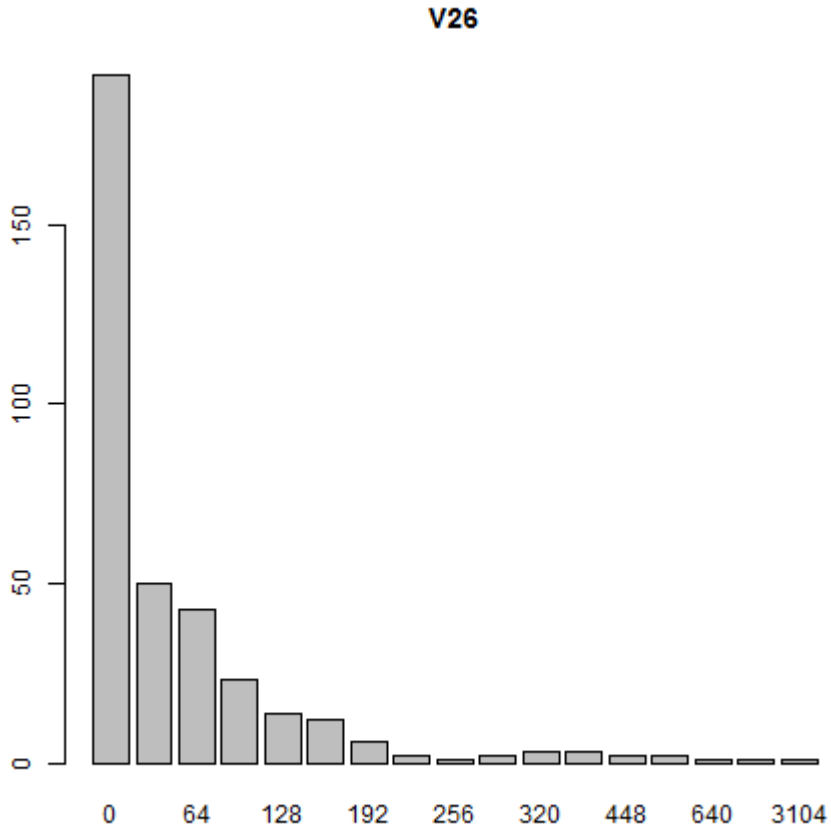

V31
Frequency
0
50
100
150
200
250
0
200
400
600
800
1000
data[, colname]

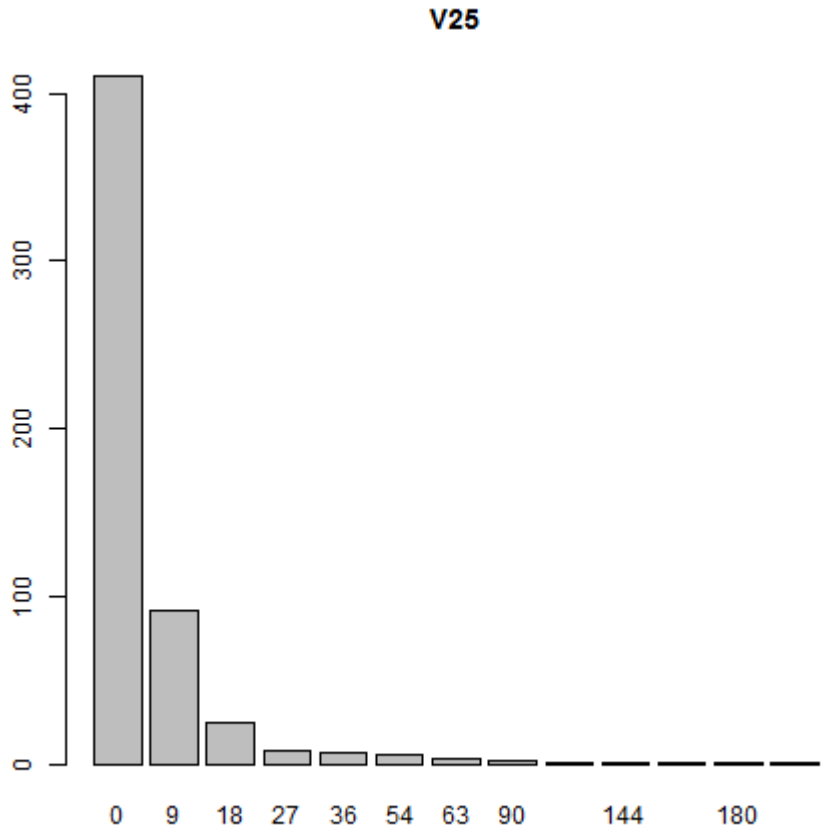

V30
0
5000
10000
15000
20000
25000
0
10
20
30
40
50
60
70
80
90
120

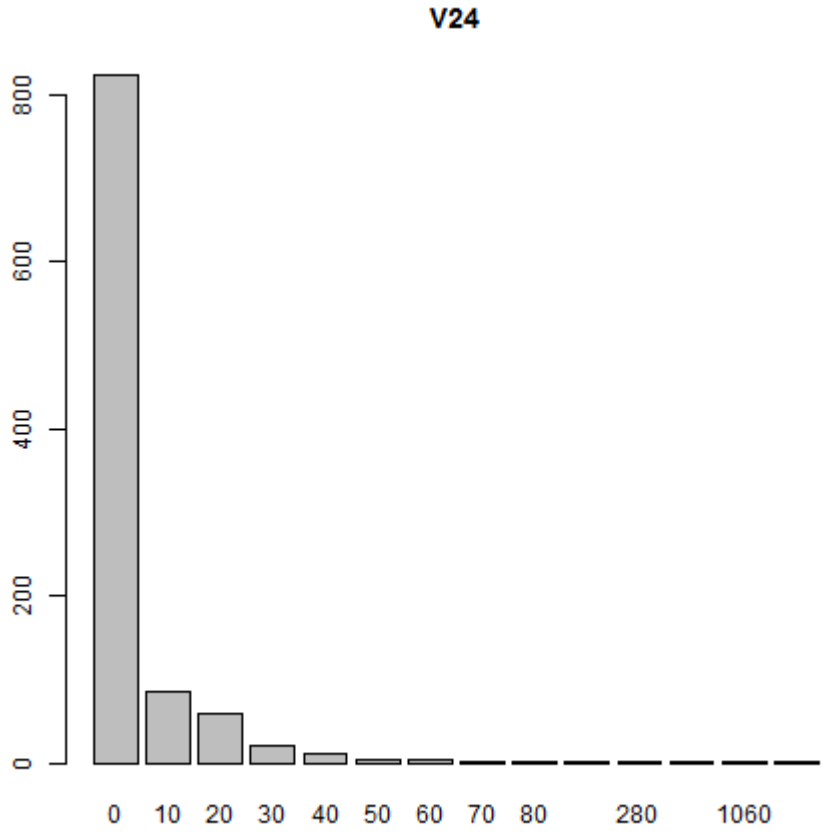

V29
20000
15000
10000
5000
0
_cTCyH95OE93jSkoIBT
DU8B
NoEd
VjDE
ZeaF

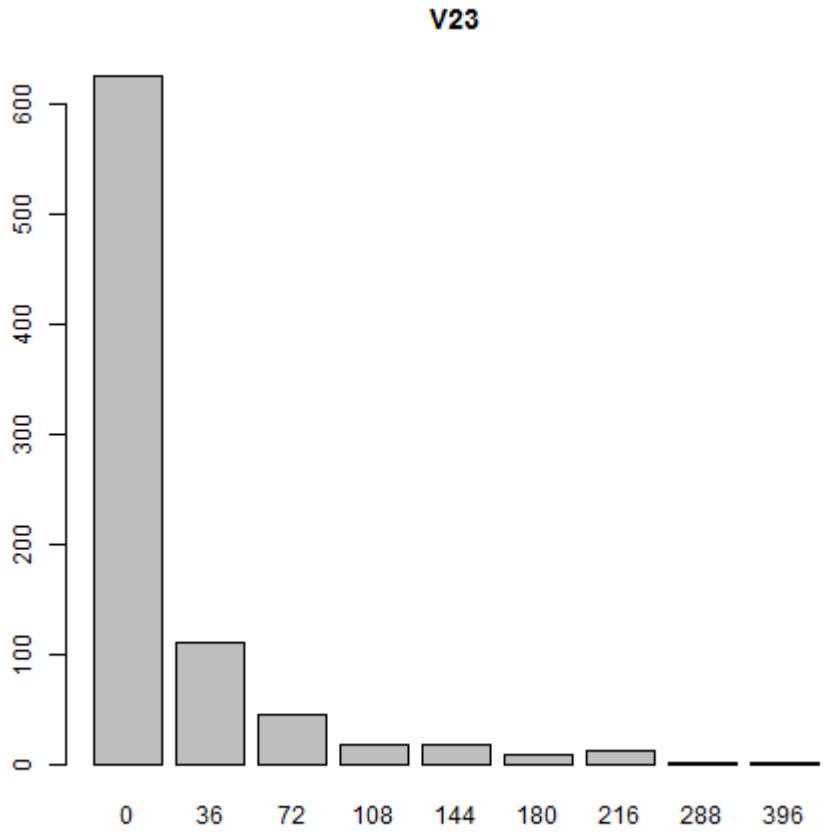

V28
350
300
250
200
150
100
50
0
0
21
42
63
84
105
147
189
315

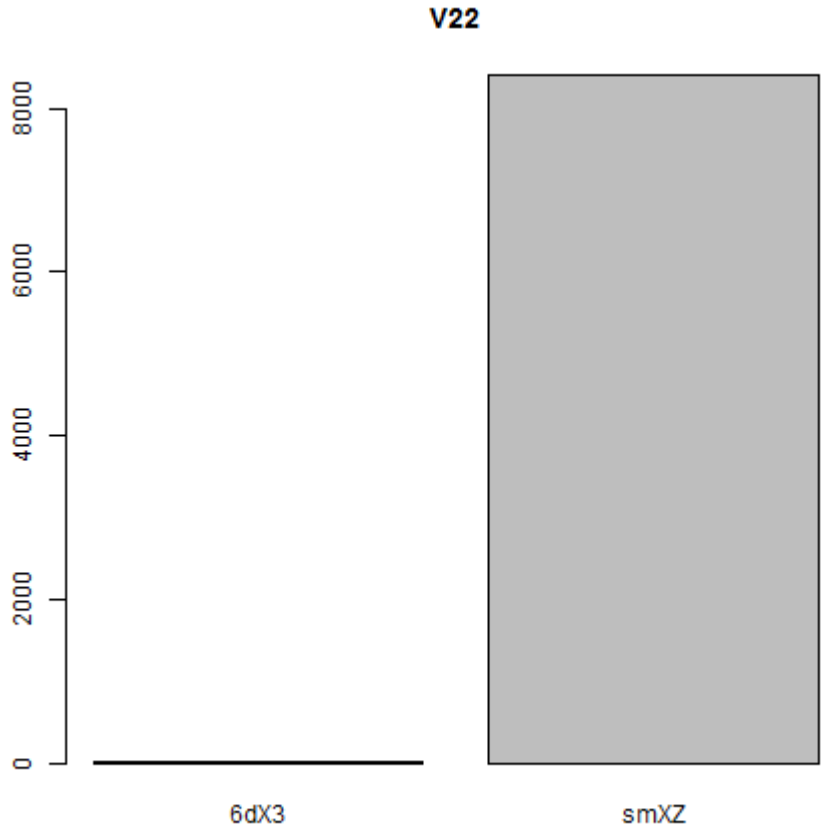

V27
0
1000
2000
3000
4000
5000
6000
7000
ELof
kG3k
xG3x

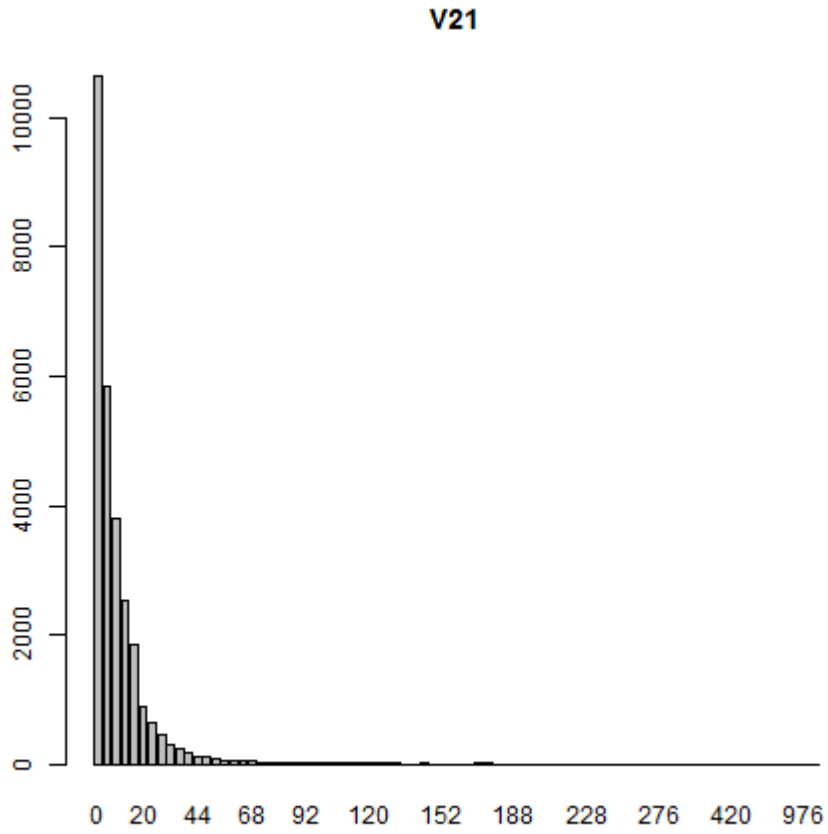

V26
0
50
100
150
0
64
128
192
256
320
448
640
3104

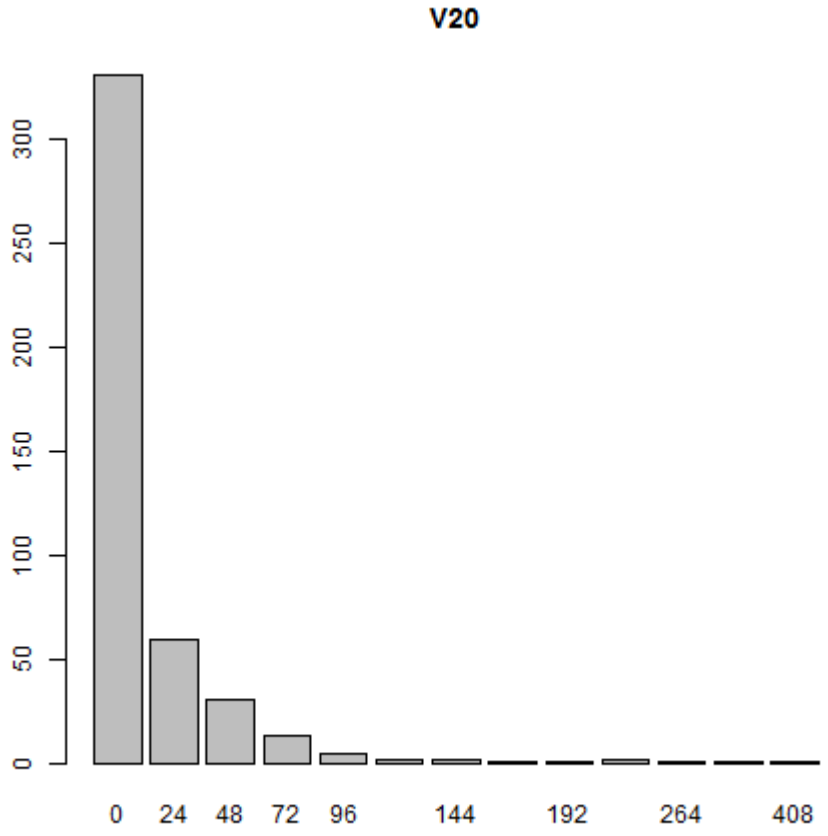

V25
400
300
200
100
0
0
9
18
27
36
54
63
90
144
180

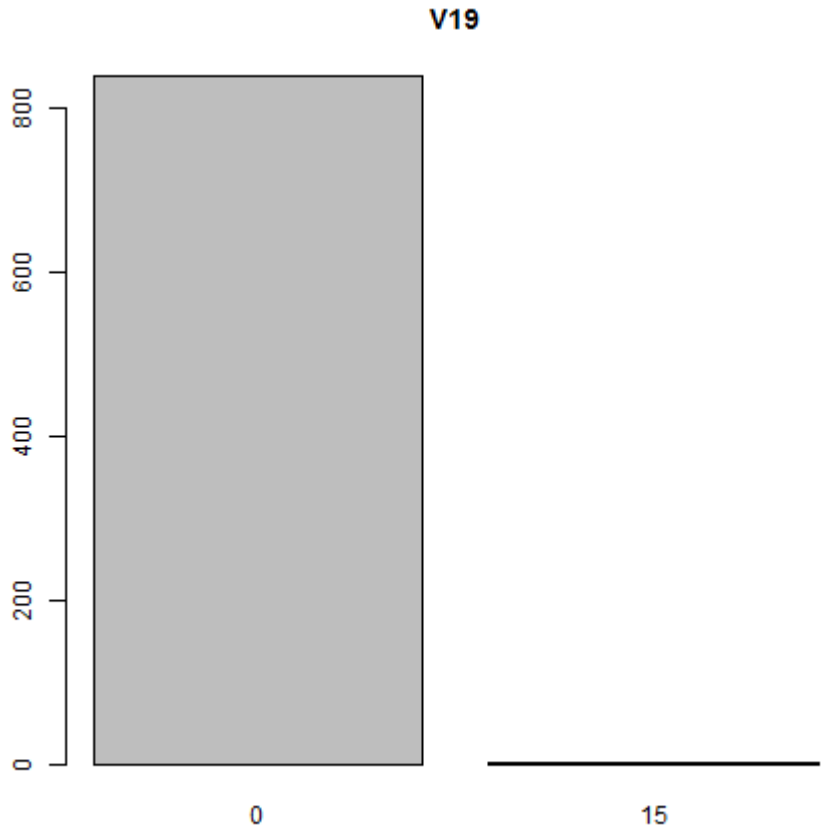

V24
800
600
400
200
0
0
10
20
30
40
50
60
70
80
280
1060

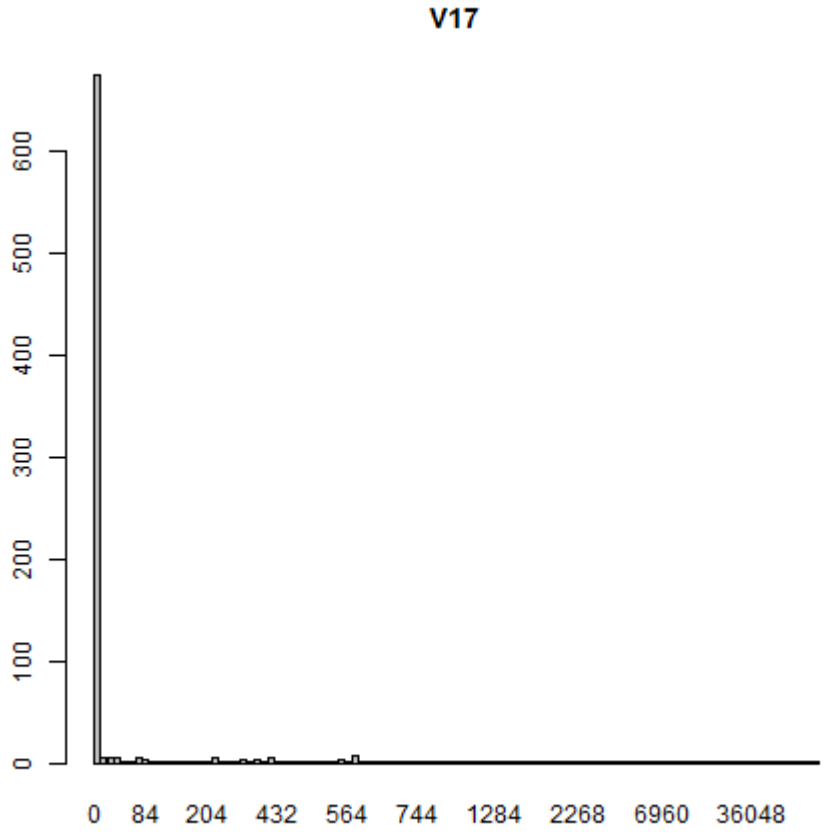

V23
0
100
200
300
400
500
600
0
36
72
108
144
180
216
288
396

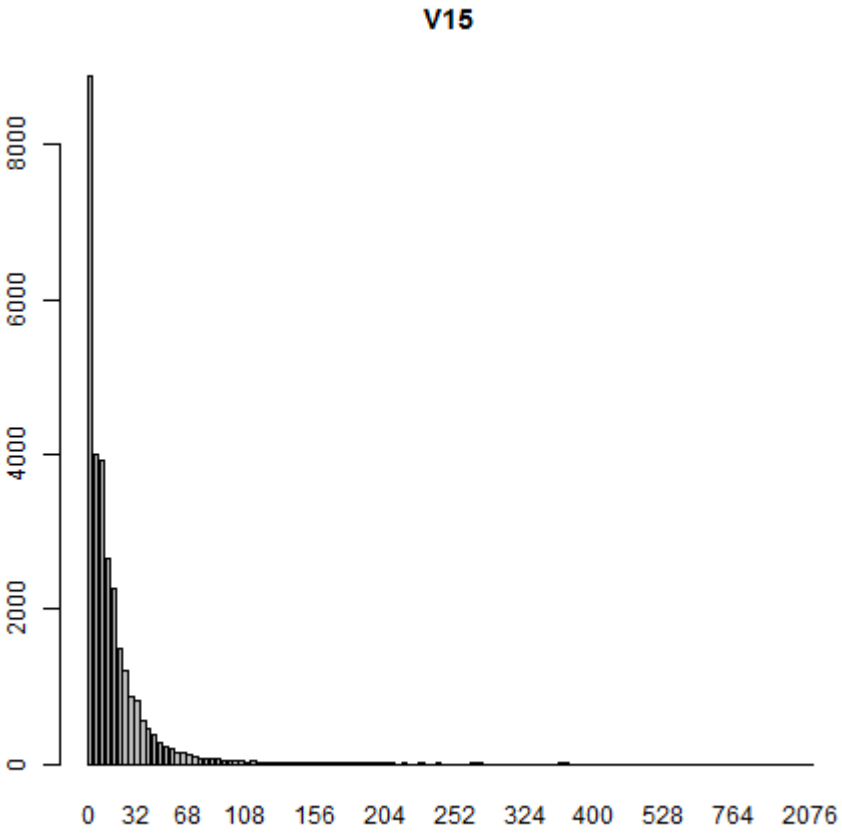

V22
0
2000
4000
6000
8000
6dX3
smXZ

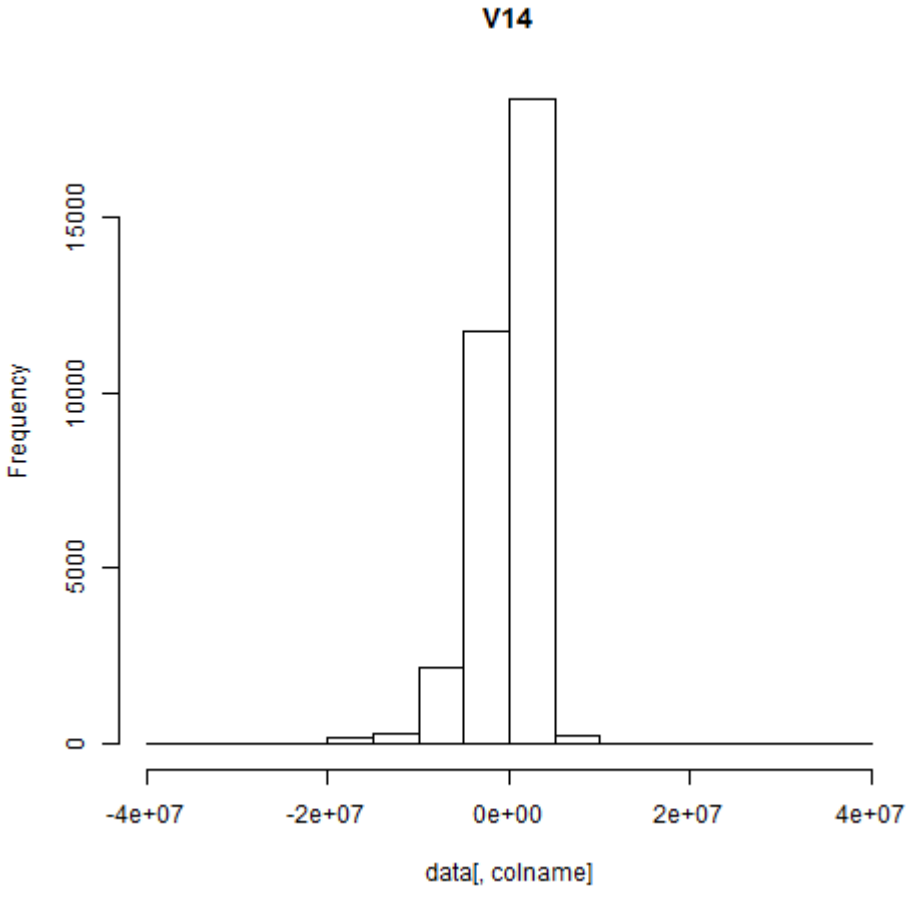

V21
0
2000
4000
6000
8000
10000
0
20
44
68
92
120
152
188
228
276
420
976

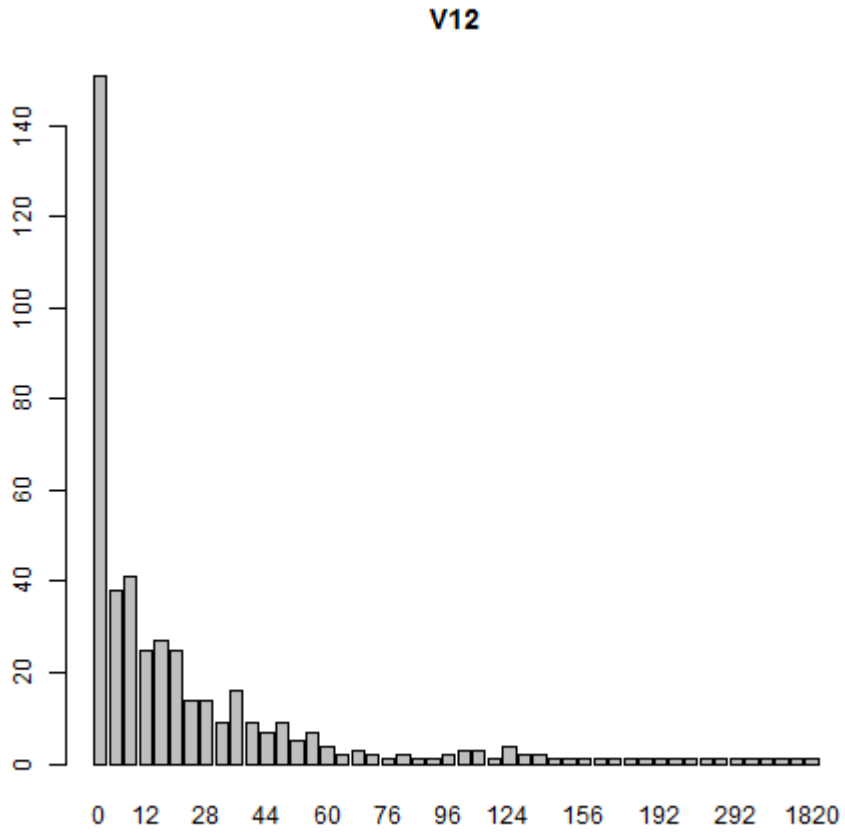

V20
0
50
100
150
200
250
300
0
24
48
72
96
144
192
264
408

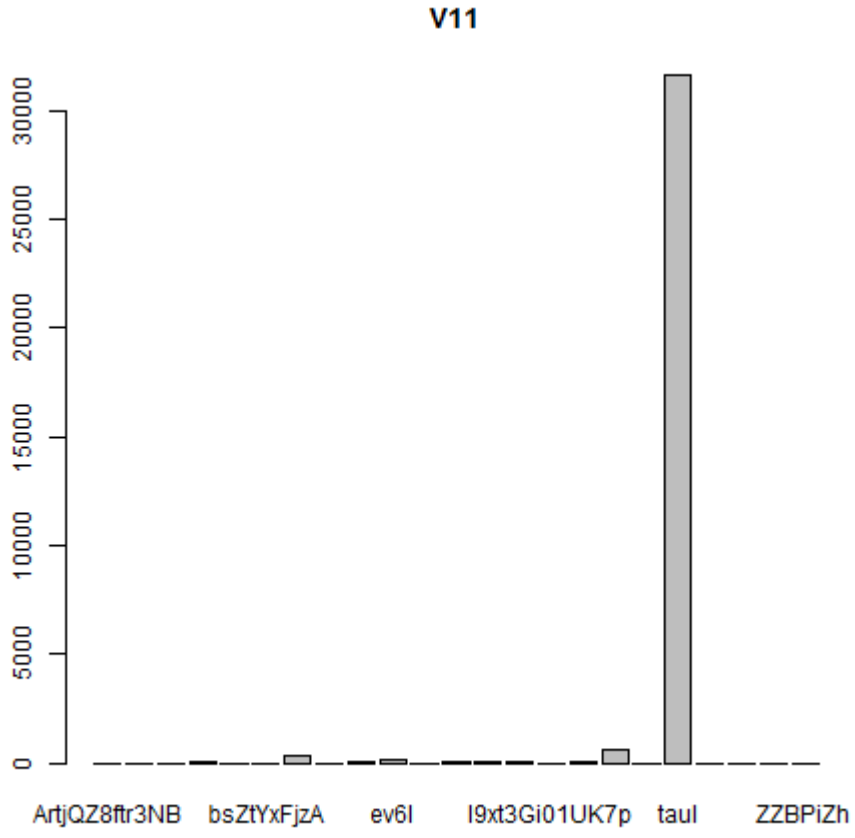

V19
800
600
400
200
0
0
15

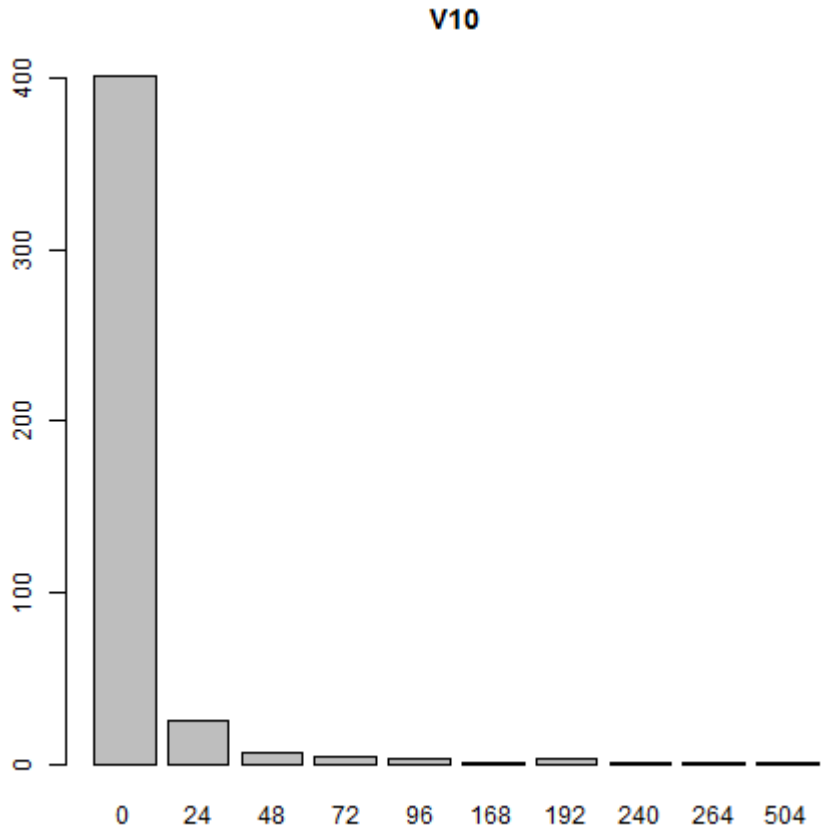

V17
600
500
400
300
200
100
0
0
84
204
432
564
744
1284
2268
6960
36048

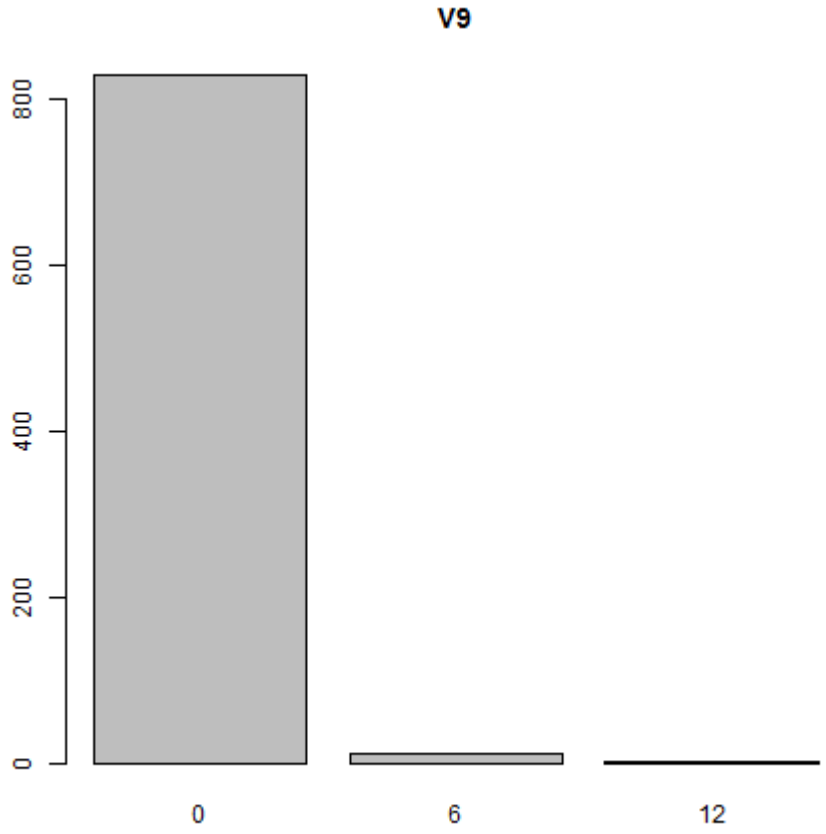

V15
8000
6000
4000
2000
0
0
32
68
108
156
204
252
324
400
528
764
2076

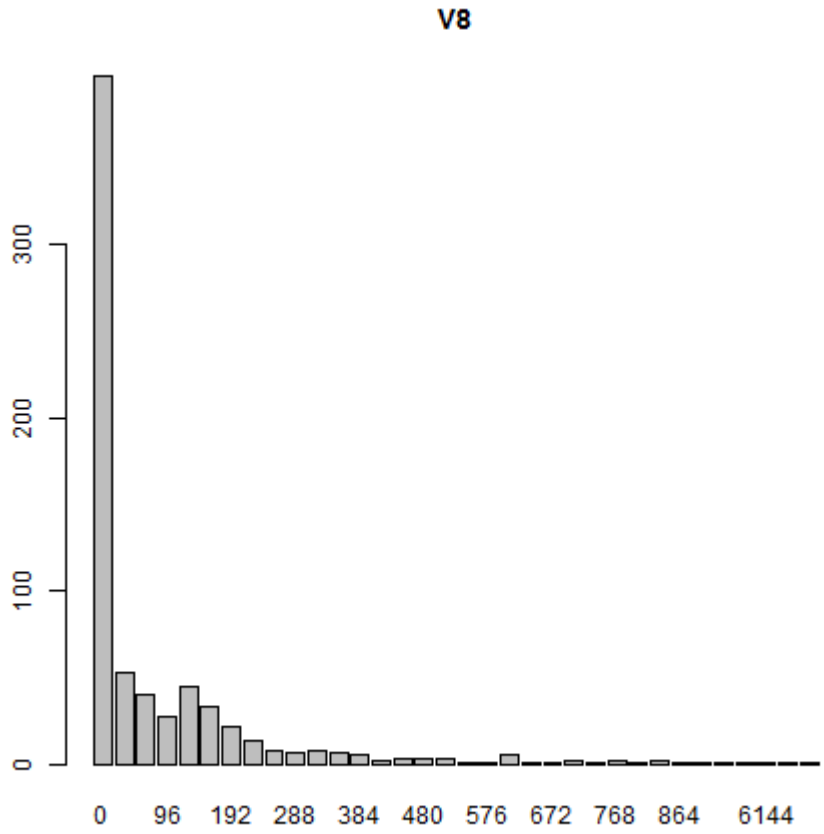

V14
Frequency
15000
10000
5000
0
-4e+07
-2e+07
0e+00
2e+07
4e+07
data[, colname]

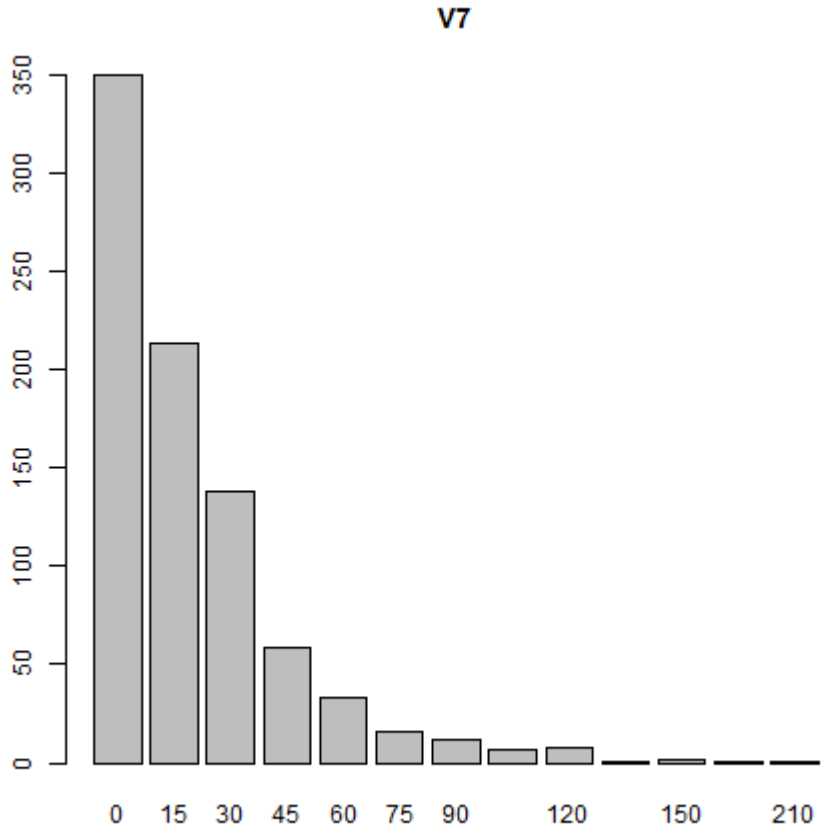

V12
0
20
40
60
80
100
120
140
0
12
28
44
60
76
96
124
156
192
292
1820

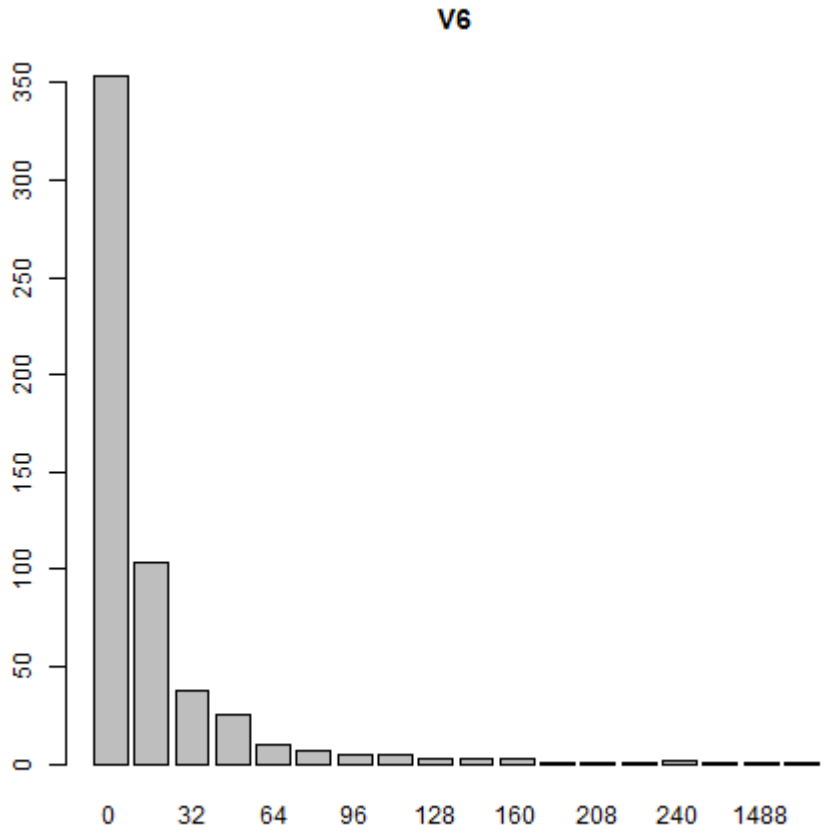

V11
0
5000
10000
15000
20000
25000
30000
ArtjQZ8ftr3NB
bsZtYxFjzA
ev6l
l9xt3Gi01UK7p
taul
ZZBPiZh

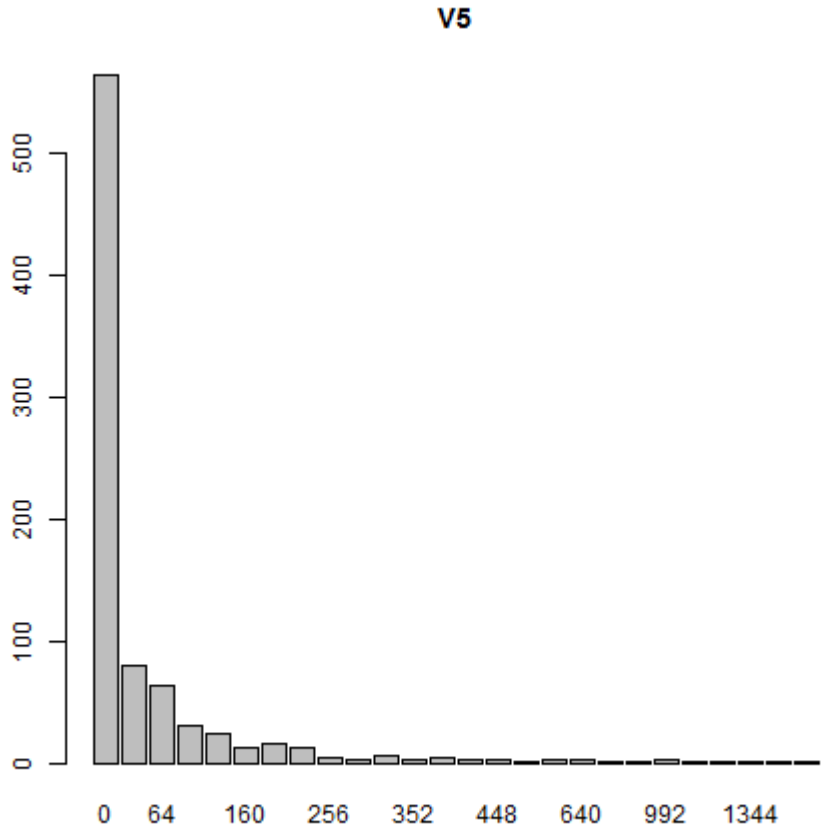

V10
400
300
200
100
0
0
24
48
72
96
168
192
240
264
504

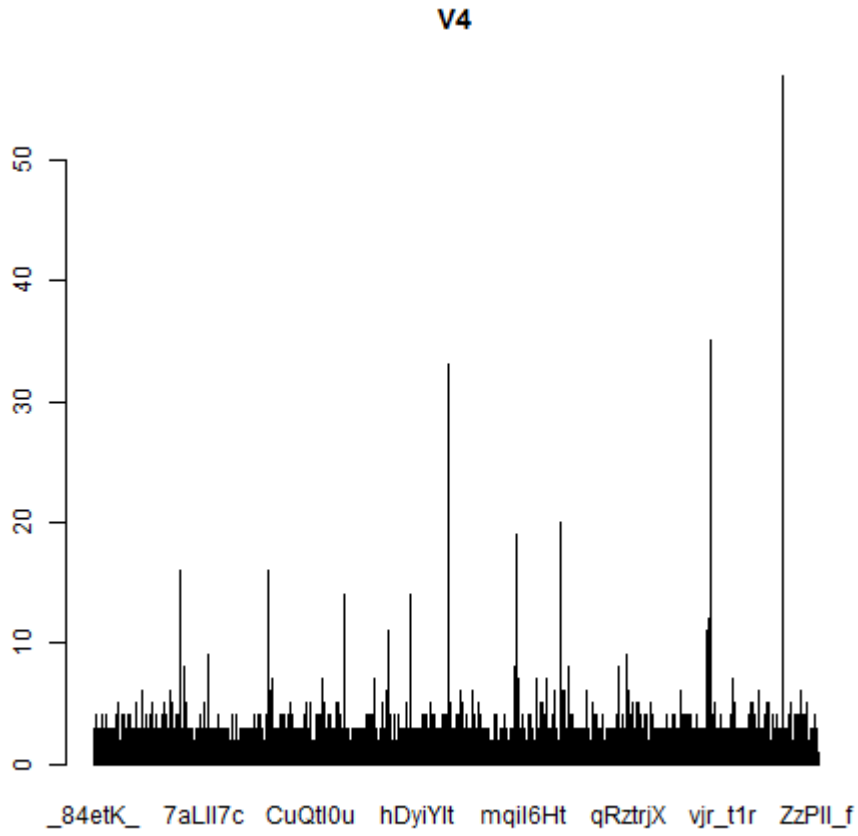

V9
800
600
400
200
0
0
6
12

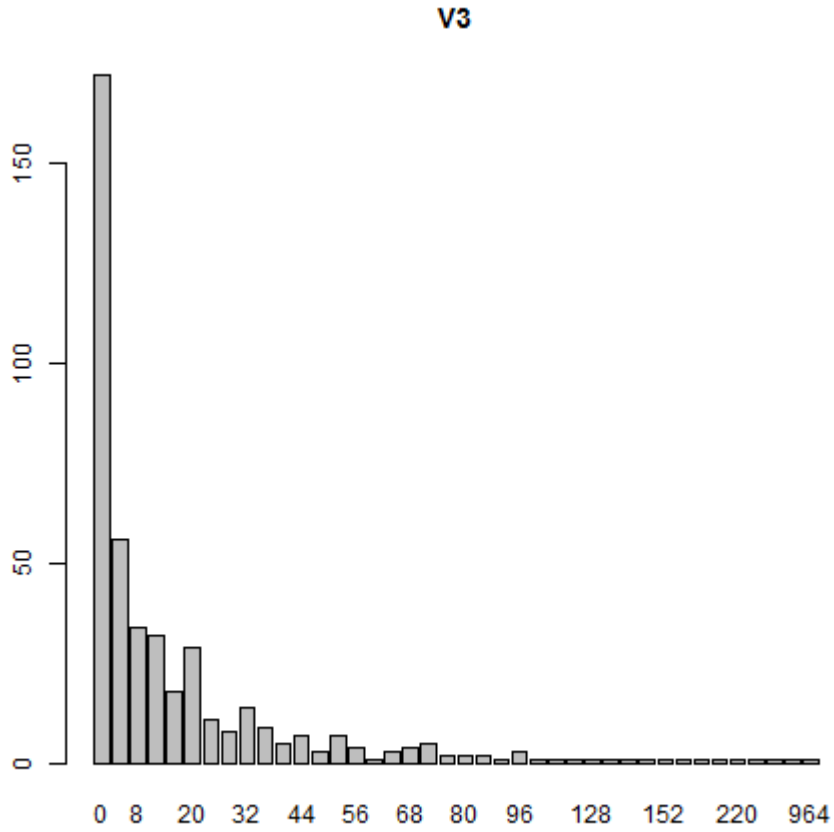

V8
0
100
200
300
0
96
192
288
384
480
576
672
768
864
6144

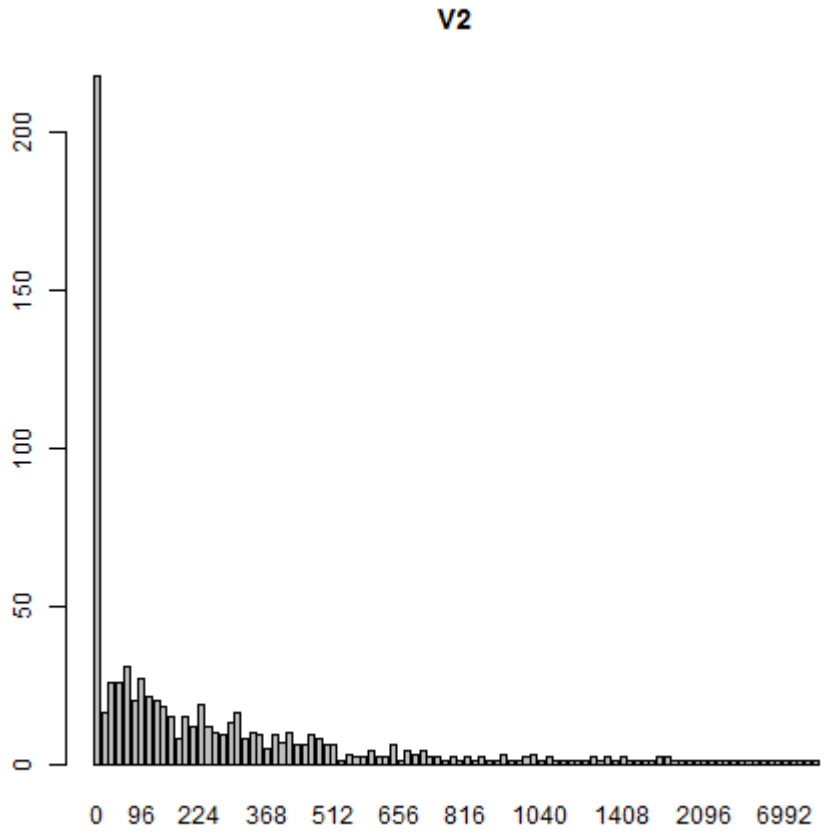

V7
0
50
100
150
200
250
300
350
0
15
30
45
60
75
90
120
150
210

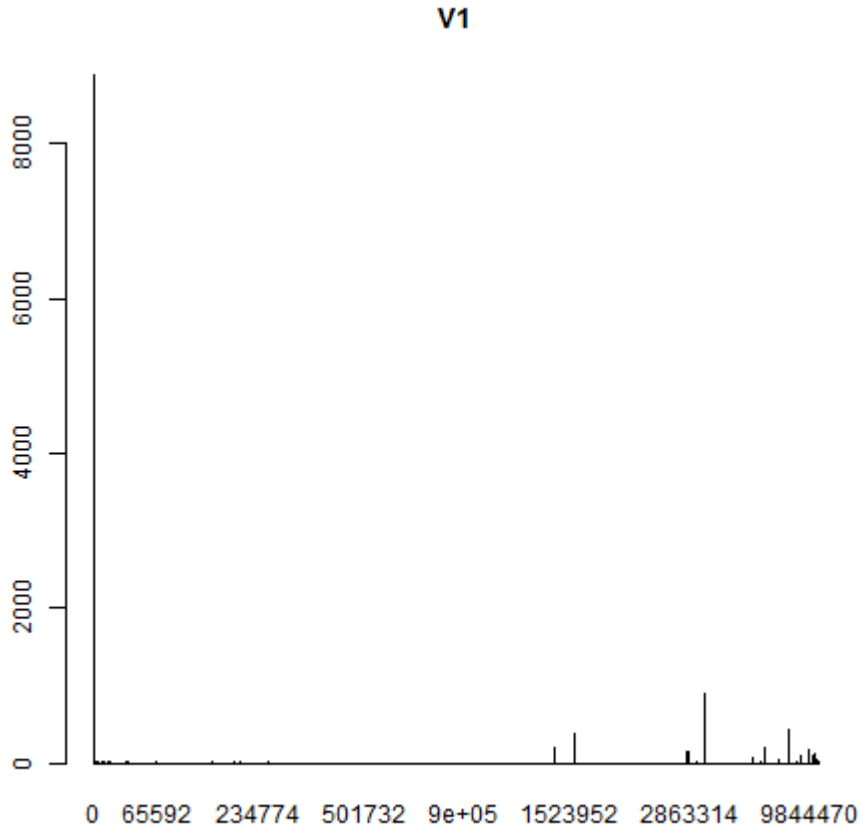

V6
0
50
100
150
200
250
300
350
0
32
64
96
128
160
208
240
1488

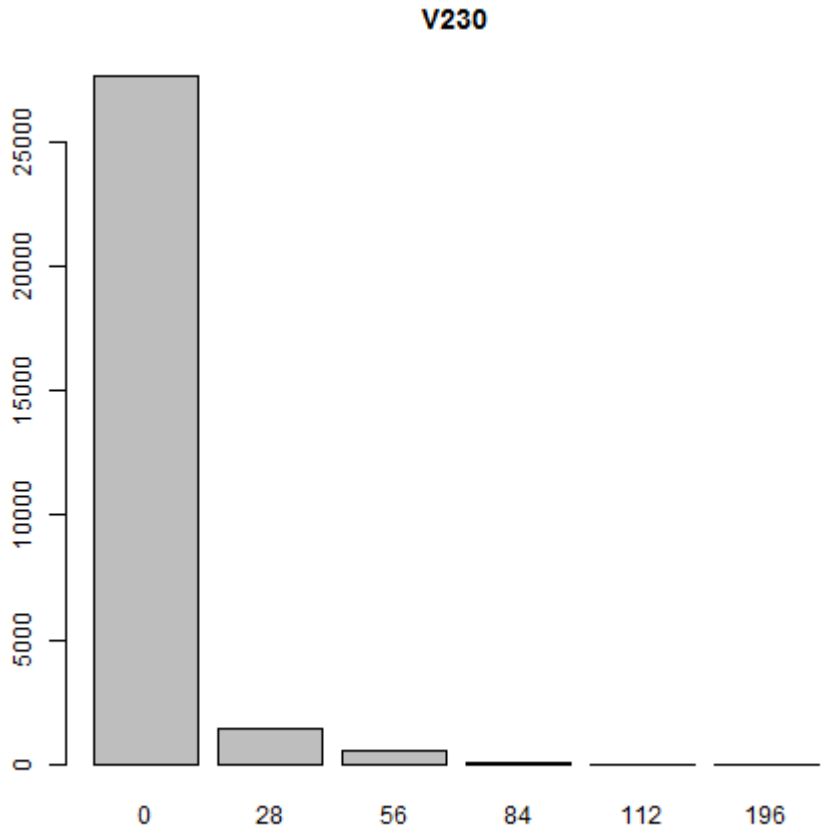

V5
0
100
200
300
400
500
0
64
160
256
352
448
640
992
1344

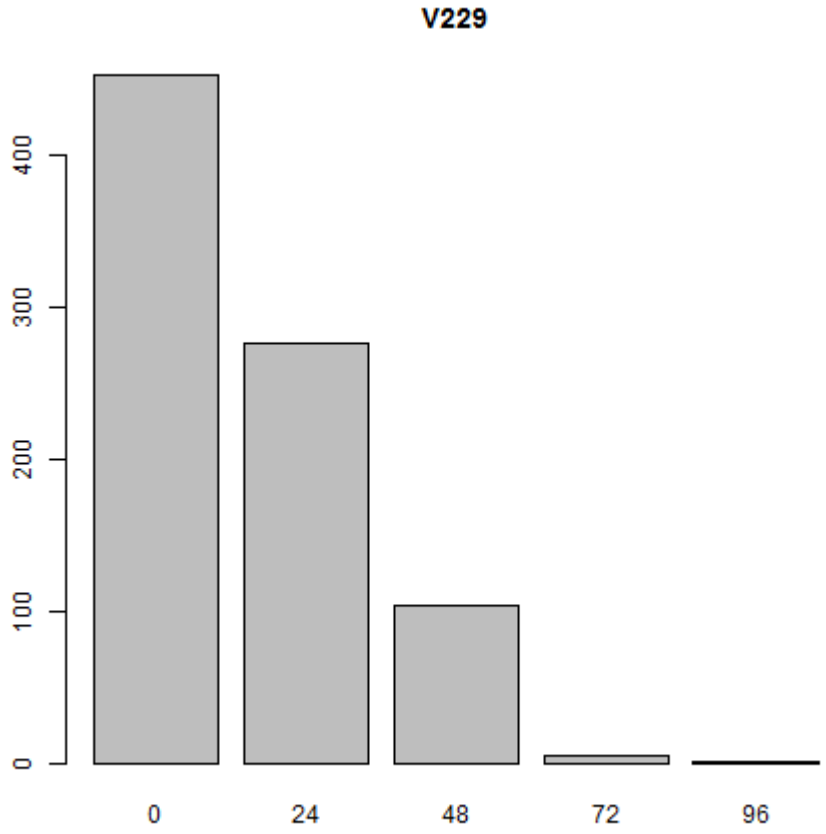

V4
0
10
20
30
40
50
84etK
7aLII7c
CuQtl0u
hDyiYlt
mqil6Ht
qRztrjX
vjr_t1r
ZzPII_f

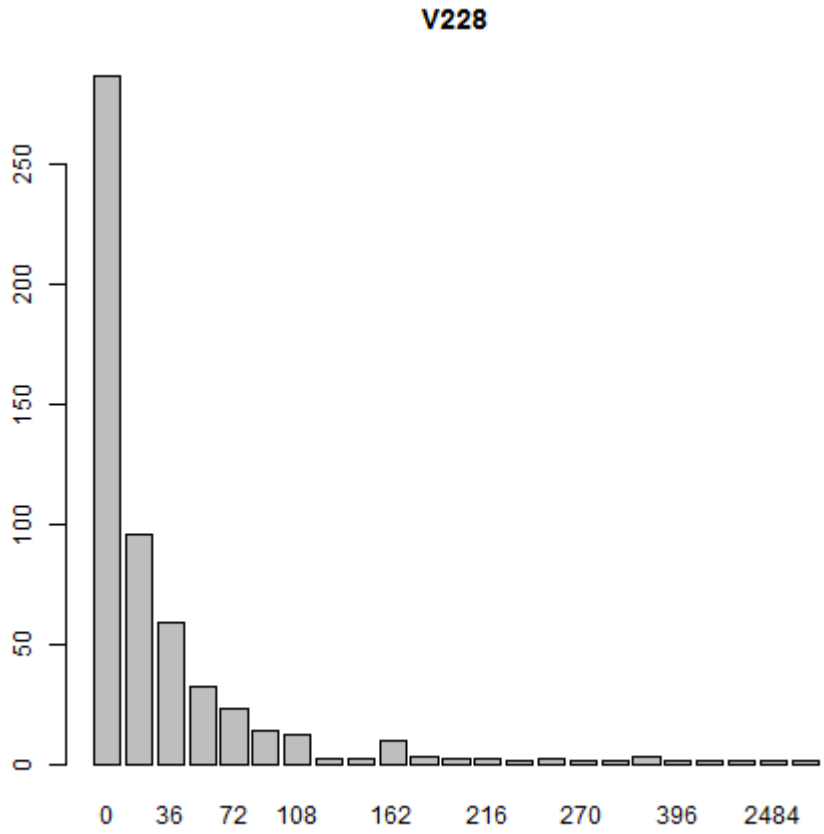

V3
0
50
100
150
0
8
20
32
44
56
68
80
96
128
152
220
964

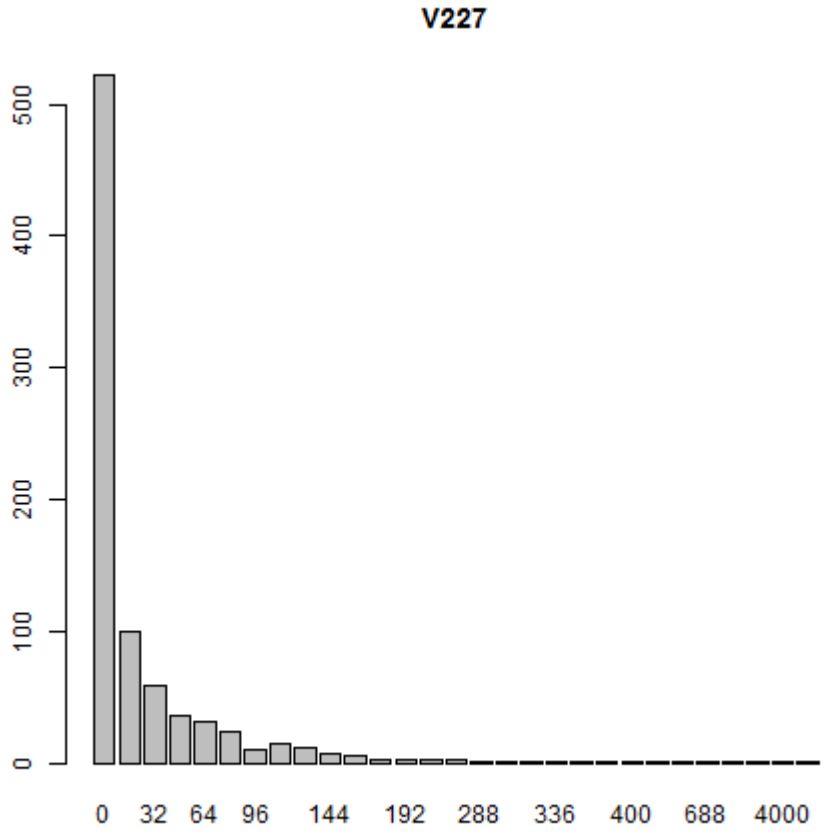

V2
0
50
100
150
200
0
96
224
368
512
656
816
1040
1408
2096
6992


V1
0
2000
4000
6000
8000
0
65592
234774
501732
9e+05
1523952
2863314
9844470


V230
0
5000
10000
15000
20000
25000
0
28
56
84
112
196


V229
0
100
200
300
400
0
24
48
72
96


V228
0
50
100
150
200
250
0
36
72
108
162
216
270
396
2484


V227
0
100
200
300
400
500
0
32
64
96
144
192
288
336
400
688
4000

Appendix B – Level Frequency graph per category

'V84/appetency == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 1.215694e-13 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V84/appetency == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 1.689765e-10 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V14/appetency == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V14/appetency == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.

'V84/churn == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0.03182046 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V84/churn == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 3.959588e-11 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V14/churn == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V14/churn == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V111/churn == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 3.728516e-10 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V111/churn == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.

'V84/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V84/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 1.798463e-09 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V14/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V14/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V101/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V101/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V111/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 8.400958e-11 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V111/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V70/upselling == 1' is normally distributed

H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V70/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V222/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V222/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V192/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V192/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V141/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V141/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V161/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V161/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V191/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V191/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V55/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V55/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V91/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V91/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V37/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V37/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V42/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 1.839214e-09 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V42/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V134/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V134/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.

'V96/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V96/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V1/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V1/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V121/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V121/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V166/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V166/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V124/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 8.881784e-15 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V124/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V81/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V81/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V151/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V151/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V75/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 1.060374e-12 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V75/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V206/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 1.125838e-09 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V206/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V197/upselling == 1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 5.790525e-09 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.
'V197/upselling == -1' is normally distributed
H0 = the data is normally distributed.
The ks p_value: 0 < 0.05 -> H0 (the null hypothesis) is rejected. The data distribution does not seem to follow a normal distribution.

Appendix C – The code structure

Data Load Code

File Path	*Description*
init_data.r	Load the trainX.csv/trainY.csv and testY.csv files. Transforms the trainX.csv categorical data into factors and bind the trainX.csv and trainY.csv files.

Data Exploration Code

File Path	*Description*
exploratory.r	Generates all the underlying data used in the section 'Data Analysis & Visualisation'
binning.exploration.r	Generates the graphs available in Appendix A. It provides the frequency analysis of each variable via a bar plot or histogram.
exploratory_functions.r	Implements the function used by exploratory.r and binning.exploration.r
\images	Contains the images generated by binning.exploration.r

Feature Selection Code

File Path	*Description*
feat.select\feat.select.lda.model.appetency.r feat.select\feat.select.lda.model.churn.r feat.select\feat.select.lda.model.upselling.r	Generates the 'MeanDecreaseGini' index, based on the pre-processing scenario1, that serves for feature selection.
feat.select\feat.select.lda.model.replacement.appetency.r feat.select\feat.select.lda.model.replacement.churn.r feat.select\feat.select.lda.model.replacement.upselling.r	Generates the 'MeanDecreaseGini' index, based on the pre-processing scenario2, that serves for feature selection.
feat.select.res\	List the table and graph output generated by the feat.select.lda files

Pre-processing Code

File Path	*Description*
pre_processing_functions.r	Implements all functions used by ML model for pre-processing the data prior to running the ML algorithms.

Test Harness Code

File Path	*Description*
test_harness.r	This is the central point of call for running the ML models. They can be run in two modes: running outside or running inside of Caret. The test harness implements the nested cross validation double loop.

ML Model Code

File Path	*Description*
models\lda\test.lda.models.feat.select.appetency.r models\lda\test.lda.models.feat.select.churn.r models\lda\test.lda.models.feat.select.upselling.r	Each file call the test harness for generating the LDA test AUC for each target label. The steps are as follows:

	1. List the scenario 1 pre-processing steps 2. Run the normal distribution Kolmogorov Smirnov test and generate the QQplots 3. Run the training/optimisation and test harness to produce the average test AUC
models\lda\test.lda.models.feat.select.replacement.appetency.r models\lda\test.lda.models.feat.select.replacement.churn.r models\lda\test.lda.models.feat.select.replacement.upselling.r	Each file call the test harness for generating the LDA test AUC for each target label. The steps are as follows: 1. List the scenario 2 pre-processing steps 2. Run the training/optimisation and test harness to produce the average test AUC
models\svm\linear\test.svm.linear.model.feat.select.appetency.r models\svm\linear\test.svm.linear.model.feat.select.churn.r models\svm\linear\test.svm.linear.model.feat.select.upselling.r	Each file call the test harness for generating the SVM (kernel=linear) test AUC for each target label. The steps are as follows: 1. List the scenario 1 pre-processing steps 2. Run the training/optimisation and test harness to produce the average test AUC
models\svm\radial\test.svm.radial.model.feat.select.appetency.r models\svm\ radial\test.svm.radial.model.feat.select.churn.r models\svm\ radial\test.svm. radial.model.feat.select.upselling.r	Each file call the test harness for generating the SVM (kernel=radial) test AUC for each target label. The steps are as follows: 3. List the scenario 1 pre-processing steps 4. Run the training/optimisation and test harness to produce the average test AUC
\models.res	Provides the models' outputs.

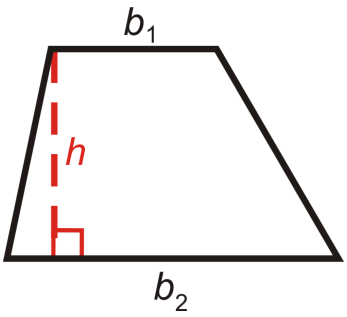
Common Code

File Path	***Description***
util_functions.r	Implements the code shared by different modules.

Appendix D – Area under a curve – Trapezoid Rule formula

Area of a Trapezoid

A trapezoid in the USA or a trapezium in the UK is a 4-sided figure (quadrilateral) in which 2 of those sides (the bases) are parallel to each other. In the UK a trapezoid is a quadrilateral with no parallel sides. In this report the USA definition of a trapezoid is used as this is standard adopted for the trapezoid method of calculating the area under a curve (AUC) of a Receiver Operating Characteristic curve (or ROC curve), also known as AUROC.



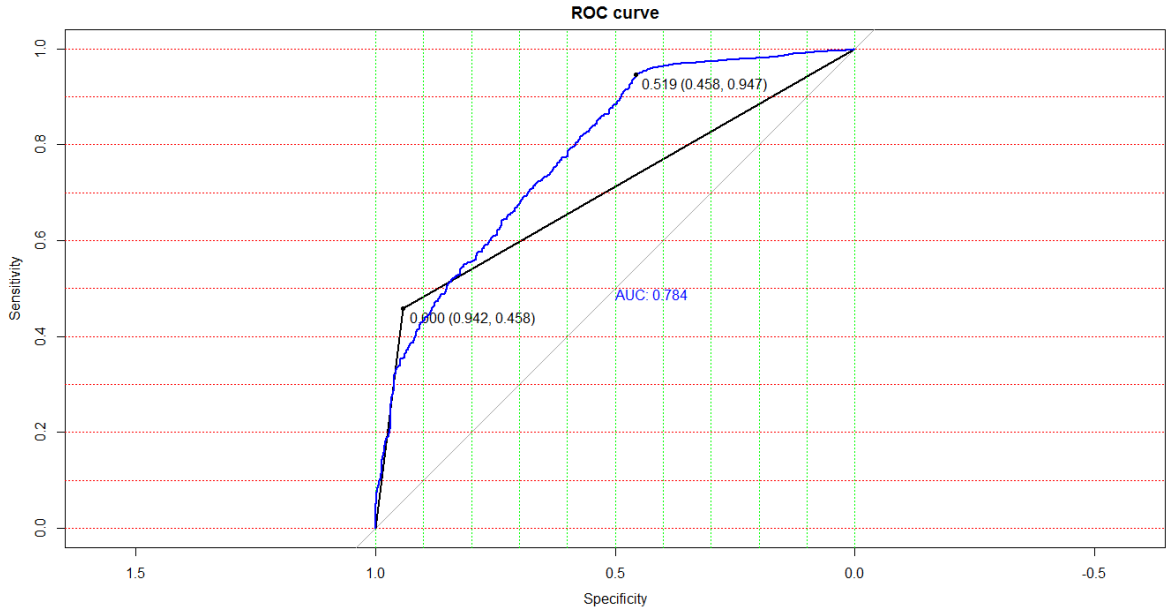
http://www.ck12.org/geometry/Area-and-Perimeter-of-Trapezoids/lesson/Area-and-Perimeter-of-Trapezoids-GEOM/

The two parallel sides of a trapezoid are called the bases. The area of a trapezoid is the average of the two bases multiplied by the height i.e. for the trapezoid shown above

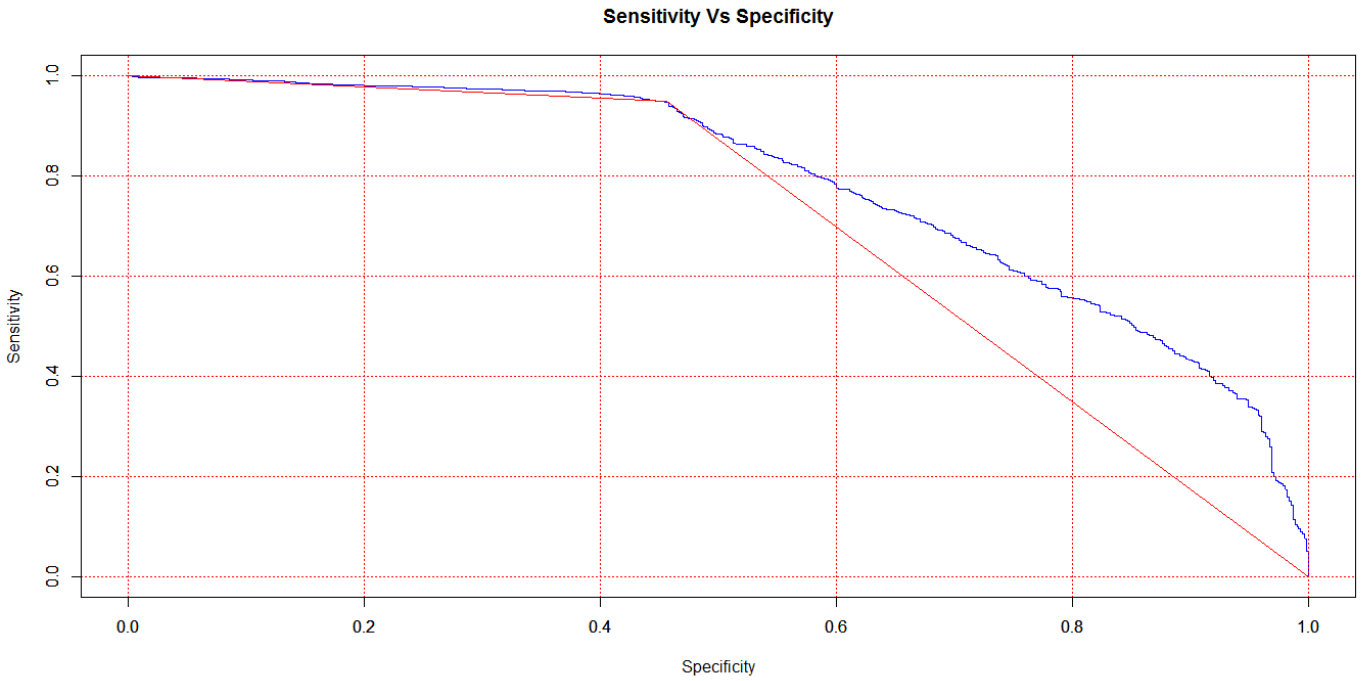
$$\text{Area} = \frac{h}{2}(b_1 + b_2)$$

ROC

The ROC is a plot of the true positive rate (sensitivity) against the false positive rate (1 – specificity) at different thresholds for a binary classifier. If a plot against specificity is drawn the specificity axis is drawn from 1 to 0.



AUROC



To calculate the AUC using trapezoids the area under the ROC curve (the blue curve) is divided into trapezoids which are adjacent to each other. The area of each trapezoid is added together to get the AUROC. The two parallel bases are parallel to the y-axis and the height is parallel to the x-axis. For the case shown above we will divide the curve into 5 trapezoids. The "height" is denoted by x and the "base" is denoted by y. The base is 0.2 units in length for all segments. But for the formula to work for Balanced Accuracy (the red curve) we will assume the lengths are unequal.

For Trapezoid 1, h = 0.2 – 0.1; b1 = 1; b2 = 0.98
For Trapezoid 2, h = 0.4 – 0.2; b1 = 0.98; b2 = 0.97
For Trapezoid 3, h = 0.6 – 0.4; b1 = 0.97; b2 = 0.8
For Trapezoid 4, h = 0.8 – 0.6; b1 = 0.8; b2 = 0.59
For Trapezoid 5, h = 1 – 0.8; b1 = 0.59; b2 = 0

We can infer from the above that the AUROC is

$$\frac{1}{2}(x_2 - x_1)(y_2 + y_1) + \frac{1}{2}(x_3 - x_2)(y_3 + y_2) + \ldots\ldots\ldots\frac{1}{2}(x_n - x_{n-1})(y_n + y_{n-1})$$

$$\Rightarrow \frac{1}{2}\sum_{i=2}^{i=n}(x_i - x_{i-1})(y_i + y_{i-1})$$

where x_i is i^{th} (1 – specificity) and y_i is the corresponding sensitivity.

Hence for the Balance Accuracy case the formula is

$$\frac{1}{2}(x_2 - 0)(y_2 + 1) + \frac{1}{2}(1 - x_2)(0 + y_2)$$