Namir’s R 203 Best Regression Tutorial

by

Namir Shammas

Table of Contents

Introduction 5
The Rationale 5
The Root Concept 5
Expanding the Search 6
The Proposed R Functions 6
Shifting and Scaling Data 8
Helper Functions 8
The has.zero() Function 8
The has.neg() Function 9
The say.fy() Function 9
The say.fx() Function 9
The show.results() Function 10
Using the Helper Functions 12
The best.lr() Function 12
The Function Declaration 12
The Source Code 13
A Sample Run 14
The best.mlr1() Function 15
The Function Declaration 15
The Source Code 16
A Sample Run 18
The best.mlr1b() Function 18

The Function Declaration 18
The Source Code 19
A Sample Run 21
The best.mlr2() Function 22
The Function Declaration 22
The Source Code 22
A Sample Run 24
The best.mlr2b() Function 25
The Function Declaration 25
The Source Code 26
A Sample Run 28
A Second Generation 29
The Functions write.best.lr() and read.best.lr() 30
The Declaration of Function write.best.lr() 30
The Source Code 30
The Declaration of Function read.best.lr() 31
The Source Code 31
A Sample Run 33
The Functions write.best.mlr1() and read.best.mlr1() 35
The Declaration of Function write.best.mlr1() 35
The Source Code 35
The Declaration of Function read.best.mlr1() 35
The Source Code 36
A Sample Run 38
The Functions write.best.mlr1b() and read.best.mlr1b() 39
The Declaration of Function write.best.mlr1b() 39
The Source Code 39
The Declaration of Function read.best.mlr1b() 40
The Source Code 40
A Sample Run 42
The Functions write.best.mlr2() and read.best.mlr2() 43

The Declaration of Function write.best.mlr2() 43
The Source Code 43
The Declaration of Function read.best.mlr2() 44
The Source Code 44
A Sample Run 46
The Functions write.best.mlr2b() and read.best.mlr2b() 48
The Declaration of Function write.best.mlr2b() 48
The Source Code 48
The Declaration of Function read.best.mlr2b() 49
The Source Code 49
A Sample Run 51
Best-Model Test Cases 52
The Best Model to Approximate the Inverse Normal Distribution 53
Tested Models 53
Test Code 53
The Results 53
The Best Model to Approximate the Inverse Student-t Distribution 55
Tested Models 55
Test Code 55
The Results 56
The Best Model to Approximate the Inverse Chi-square Distribution 58
Tested Models 58
Test Code 59
The Results 59
The Best Model to Approximate the Inverse F Distribution 62
Tested Models 62
Test Code 62
The Results 62
Using the Significance Levels 64
The Inverse Normal Distribution 65
The Inverse Student-t Distribution 66

The Inverse Chi-square Distribution 66

The Inverse F Distribution 66

Massaging the Best-Model Search Output 67

The VBA Code 67

The Main Tasks of the VBA Code 72

Sample Input and Output 73

$\mathcal{R} > \mathcal{S}$

"The better is the enemy of the good."

Voltaire

Introduction

This tutorial complements *Namir's R 101 Tutorial*, *Namir's R 102 Plotting Tutorial*, and *Namir's 201 Regression Tutorial*. This tutorial focuses on automating model selection for two, three, and four variables. The R functions presented in this tutorial excel in quickly wading through tens, hundreds, and even thousands of regression models.

The Rationale

The Root Concept

The idea of obtaining the best fit goes back to the days of programmable HP calculators several decades ago. These calculators offered programs that worked with data for a dependent variable y and an independent variable x. By using linear (basically no transformation) and logarithmic transformations on each of these variables, the calculator programs tested the following models and reported the model with the highest coefficient of determination:

$$y = a + b\,x$$

$$y = a + b\ln(x)$$

$$\ln(y) = a + b\,x$$

$$\ln(y) = a + b\ln(x)$$

So by just using two transformations on each variable, the calculator programs searched among 4 (equals 2^2) models.

Expanding the Search

Moving up from 2 to n transformations, a similar R function can search among n^2 models for two regression variables. In the case of three and four regression variables you have n^3 and n^4 models!

The following table shows the suggested transformation set and numeric code associated with them.

Table 1. The list of transformations.

Transformation	*Numeric Code for the Transformation*
x	1
x^2	2
x^3	3
sqrt(x)	4
ln(x)	0
1/x	–1
$1/x^2$	–2
1/x3	–3
1/sqrt(x)	–4

A best-model searching R function has to apply up to 9 transformations on each regression variable. The key word here is *up to*. If the regression data has zeros and negative numbers, then the model selection process has to bypass using certain transformations, such as the square root and the natural logarithm, to name a few. Thus, the best-model searching R functions have to scan the data for negative values and for zeros and flag them.

The Proposed R Functions

The next table lists the set of best-model searching R functions that I present in this tutorial and indicate the scope of their model search.

Table 2. The best-model searching R functions.

Name of R Function	*Total Number of Variables*	*Scope of Model Search*
best.lr()	2	Searches for the best model: fy(y) = a + b fx(x) Among a maximum of 81 models.
best.mlr1()	3	Searches for the best model: fy(y) = a + b fx1(x1) + c fx2(x2) Among a maximum of 729 models.
best.mlr1b()	2	Searches for the best model:

Name of R Function	***Total Number of Variables***	***Scope of Model Search***
		fy(y) = a + b fx1(x) + c fx2(x) Among a maximum of 324 distinct models. The functions fx1 and fx2 are different transformations.
best.mlr2()	4	Searches for the best model: fy(y) = a + b fx1(x1) + c fx2(x2) + d fx3(x3) Among a maximum of 6561 models.
best.mlr2b()	2	Searches for the best model: fy(y) = a + b fx1(x) + c fx2(x) + d fx3(x) Among a maximum of 756 distinct models. The functions fx1, fx2, and fx3 are different transformations.

In the case of functions mlr1b() and mlr2b(), the regression occurs between two variables. The independent variable appears in multiple terms. When these functions apply transformations to the independent variable, they need to avoid applying the same transformation twice. Also these functions need to avoid redundant symmetry. Symmetric terms generate basically the same model by swapping the transformations for the same variable. For example, the models

fy(y) = A + B fx1(x) + C fx2(x)

fy(y) = a + b fx2(x) + c fx1(x)

Are symmetric, since regression on these two models yields B = c and C = b. The two models are essentially the same, with terms simply shuffled around. While symmetrical terms generate no runtime error, they tend to make you see many *doubles* when you look at the best-model results.

The functions in the above table perform linearized regression on numerous models and sort the result based on the value of the F statistics for each model. I chose to use the F statistic since it has no upper limit as does the coefficient of determination. The functions return the following data for each regression:

- The F statistic. This value is used as the key in sorting the results.
- The coefficient of determination, R^2.
- The regression intercept.
- The regression slopes for each independent variable/term.
- The R-style formula used to specify the regression model.

Keep in mind that each function returns results that share the same degrees of freedom.

Shifting and Scaling Data

What about shifting and scaling the regression data? The answer is that such an operation, where you multiply a regression variable by a scale and then add a shift value, is a double edged sword. When you apply the same scale and shift values to a regression variable and then feed the results to a best-model seeking function, the results can be mixed. Such data massaging can give certain models an advantage while put other models at a disadvantage. So when it comes to scaling and shifting data, I will leave that to you. You can perform such operations very easily on the regression data and right before you call a best-model searching function.

The tools I offer do not offer a magical path to the best empirical model or confirming theoretical models through regression. These tools significantly reduce the time for searching through tens, hundreds, and even thousands of regression models. Experience shows that the errors in the data may well favor certain regression models over others. It is a good practice to prudently look at the list of leading models and not solely focus on (more like obsess with) the very best model. If you can afford the luxury of having additional data sets to work with, then the best empirical model selection may benefit from that.

Pitting one regression model against another is nothing short of stirring a war of curvatures, so to speak. The fittest model is the one that accommodates the data (and errors in the data) with the best curvature. May the best regression model win!

Helper Functions

Before I discuss the best-model searching R functions, I present their helper functions. These functions, which play a vital supporting role, are has.zero(), has.neg(), say.fx(), say.fy(), and show.results().

The has.zero() Function

The has.zero(x) function which returns TRUE if the vector x contains a zero. Here is the source code for the function:

```
has.zero = function(x)
{
  # return TRUE if vector x has a zero value
  for (i in 1:length(x)) {
    if (x[i] == 0) return (TRUE)
  }
  return (FALSE)
}
```

The function searches all of the elements in vector x and does not need that the vector be sorted.

The has.neg() Function

The has.neg(x) function which returns TRUE if the vector x contains a negative value. Here is the source code for the function:

```
has.neg = function(x)
{
  # return TRUE if vector x has a negative value
  for (i in 1:length(x)) {
    if (x[i] < 0) return (TRUE)
  }
  return (FALSE)
}
```

The say.fy() Function

The say.fy(index, varname) function builds part of the regression formula that involves the dependent variable. The parameter **index** represents the transformation code that appears in Table 1. The parameter **varname** is the name of the dependent variable. It MUST match the name of the dependent variable vector that is passed to the best-model searching functions. Here is the source code for the function:

```
say.fy = function(index, varname)
{
  # translate the transformation index into a term with the
  # specified dependent variable
  if (index == 1)
    return (varname)
  else if (index == 2)
    return (paste(varname, "^2", sep=""))
  else if (index == 3)
    return (paste(varname, "^3", sep=""))
  else if (index == 0)
    return (paste("log(", varname, ")", sep=""))
  else if (index == -1)
    return (paste("1/", varname, "", sep=""))
  else if (index == -2)
    return (paste("1/", varname, "^2", sep=""))
  else if (index == -3)
    return (paste("1/", varname, "^3", sep=""))
  else if (index == 4)
    return (paste("sqrt(", varname, ")", sep=""))
  else if (index == -4)
    return (paste("1/sqrt(", varname, ")", sep=""))
  else
    return (varname)
}
```

The say.fx() Function

The say.fx(index, varname) function builds part of the regression formula that involves the independent variable. The parameter **index** represents the transformation code that appears in Table 1. The parameter **varname** is the name of the independent variable. It MUST match the name of the independent variable vector that is passed to the best-model searching functions. Here is the source code for the function:

```
say.fx = function(index, varname)
{
  # translate the transformation index into a term with the
  # specified independent variable
  if (index == 1)
    return (varname)
  else if (index == 2)
    return (paste("I(", varname, "^2)", sep=""))
  else if (index == 3)
    return (paste("I(", varname, "^3)", sep=""))
  else if (index == 0)
    return (paste("I(log(", varname, "))", sep=""))
  else if (index == -1)
    return (paste("I(1/", varname, ")", sep=""))
  else if (index == -2)
    return (paste("I(1/", varname, "^2)", sep=""))
  else if (index == -3)
    return (paste("I(1/", varname, "^3)", sep=""))
  else if (index == 4)
    return (paste("I(sqrt(", varname, "))", sep=""))
  else if (index == -4)
    return (paste("I(1/sqrt(", varname, "))", sep=""))
  else
    return (varname)
}
```

The show.results() Function

The function show.results(list.res, outfile, show.best=–1,) displays results to the R Console window and possibly send the same output to a text file. The parameter **list.res** is a list that contains the results for the best-model search. The parameter **outfile** specifies the output file. The default value is an empty string which suppresses file output. Passing an invalid filename generates a runtime error. The parameter **show.best** tells the function how many models to display. The default of –1 tells the function to show all the models. Here is the source code for the function:

```
show.results = function(list.res, show.best=-1, outfile="",)
{
  # show the results
  mat.res = list.res$mat
  formula.arr = list.res$form
  # get the number of models obtained and the number of results
  n.models = nrow(mat.res)
  n.res = ncol(mat.res)
  # remove the output file if it exists
  if (file.exists(outfile)) {
    file.remove(outfile)
  }
  # determine how many values to display
  if (show.best == -1) {
    n = n.models
  }
  else {
    if (show.best > n.models)
      n = n.models
    else
      n = show.best
  }

  # show header if few models are selected
  if (n < n.models) {
    cat("Best ", n, " models\n", sep="")
    if (outfile != "") {
      cat("Best ", n, " models\n", file=outfile, append="TRUE", sep="")
    }
  }

  nobs = list.res$nobs
  cat("Number of observations = ", nobs, "\n", sep="")
  if (outfile != "" & file.exists(outfile)) {
    cat("Number of observations = ", nobs, "\n", file=outfile,
         append="TRUE", sep="")
  }

  # show the best models
  for (i in 1:n) {
    if (mat.res[i,1] > 0) {
      cat("F = ", mat.res[i,1] , ", R^2 = ",  mat.res[i,2],
        ", Model is ",  formula.arr[i], ", Intercept = ",
         mat.res[i,3], sep="")
      for (j in 4:n.res)
        cat(ifelse(j==n.res, ", and Slope", ", Slope"), j-3,
            " = ", mat.res[i,j], sep="")
      cat("\n")
    }
  }

  # output models to a text file
  if (outfile != "" & file.exists(outfile)) {
    for (i in 1:n) {
      if (mat.res[i,1] > 0) {
        cat("F = ", mat.res[i,1] , ", R^2 = ",  mat.res[i,2],
          ", Model is ",  formula.arr[i], ", Intercept = ",
           mat.res[i,3], file=outfile, append="TRUE", sep="")
        for (j in 4:n.res)
          cat(ifelse(j==n.res, ", and Slope", ", Slope"), j-3,
              " = ", mat.res[i,j],
              file=outfile, append="TRUE", sep="")
        cat("\n", file=outfile, append="TRUE", sep="")
      }
    }
  }
}
```

The parameter list.res is a list that contains the following elements:

- The tag mat contains the matrix of results. The columns for the matrix store the values for the F statistic, coefficient of determination, the regression intercept, and the regression slopes--each slope in a separate column.
- The tag form contains the vector of formulas that specify the regression models.

Using the Helper Functions

There are several ways that you can work with the helper functions:

- Include their code in the script file of each best-model seeking function. You end up with one script that contains all of the functions.
- Save the helper functions and the best-model seeking functions in separate script files and manually load each script at the command line by calling the source() function.
- In the script file for the best-model seeking function, include a call to function source() to load the helper functions. Make sure that the call to function source() has the correct path and script file name for the helper functions.
- Include all of the helper functions and all of the best-model seeking functions in a single script file. You then have the entire code in a single file that you can easily load from the command line prompt.

The best.lr() Function

The Function Declaration

The function bes.lr() performs the best-model search for two variables. The declaration for this function is:

```
best.lr = function(x, y, name.x="x", name.y="y", quiet=FALSE,
                   show.best=-1, outfile="C:/best.lr.txt")
```

The parameters **x** and **y** are the data vectors. The parameters **name.x** and **name.y** represent the string names for the vectors x and y, and MUST match the names of these vectors. The Boolean parameter **quiet** is a flag that tells the function best.lr() not to display any results or write any results to an output file. The parameter **show.best** specified the number of best models to display. The default value of -1 tells the function to display all of the results. The parameter **outfile** specifies the output file. If you pass an empty string to this parameter, you suppress file output.

The Source Code

Here is the source code for the function best.lr():

```
best.lr = function(x, y, name.x="x", name.y="y", quiet=FALSE,
                   show.best=-1, outfile="C:/best.lr.txt")
{
  # x and y are data vectors.
  # name.x is the name for argument for parameter x
  # name.y is the name for argument for parameter y
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #   Default shows all results
  # outfile is the output filename

  max.models=81
  max.res=4
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  v = rep(0,max.res) # dummy vector used for swapping rows in mat.res
  # initialize number of models found
  n.models = 0
  # check for zero and negative values in arrays
  x.has.zero = has.zero(x)
  x.has.neg = has.neg(x)
  y.has.zero = has.zero(y)
  y.has.neg = has.neg(y)
  for (iy in -4:4) {
    # check for zero value restraint
    if (!(y.has.zero & iy <= 0)) {
      # check for negative value restraint
      if (!(y.has.neg & (iy == 4 | iy == -4 | iy == 0))) {
        for (ix in -4:4) {
          # check for zero value restraint
          if (!(x.has.zero & ix <= 0)) {
            # check for negative value restraint
            if (!(x.has.neg & (ix == 4 | ix == -4 | ix == 0))) {
              # increment number of models
              n.models = n.models + 1
              # construct formula
              formula = paste(say.fy(iy,name.y),"~",say.fx(ix,name.x),sep="")
              # perform linearized regression
              lr = lm(formula)
              # get the summary
              slr = summary(lr)
              # build matrix of results
              mat.res[n.models,1] = slr$fstatistic[1]
              mat.res[n.models,2] = slr$r.squared
              mat.res[n.models,3] = slr$coefficients[1]
              mat.res[n.models,4] = slr$coefficients[2]
              formula.arr[n.models] = formula
            }
          }
        }
      }
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Initializes the variables that store the results.
2. Scans the values in vectors x and y for zeros and negative values. This task stores the results of these scans to use in the next task.
3. Uses nested for loops to apply all of the transformations on the regression variables. The loops include several if statements that check for zeros and negative values and how they can prevent applying certain transformations.
4. The innermost if statement builds the model for the regression and performs a linearized regression by calling function lm(). The function stores the results of functions lm() and summary(). With this information at hand, the function stores a subset of the total results in matrix mat.res and vector formula.arr.
5. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
6. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
7. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
8. Returns the list list.res.

A Sample Run

Once you store the function best.lr() and the helper functions, load them using the source() function. To test the best.lr() function, execute the following commands:

```
> x=runif(20,1,10)
> y=x^2-4
> lr.list = best.lr(x,y,show.best=10)
```

```
Best 10 models
Number of observations = 20
F = 4.418392e+32, R^2 = 1, Model is y~I(x^2), Intercept = -4, and Slope1 = 1
F = 6969.695, R^2 = 0.997424, Model is 1/y^2~I(1/x^2), Intercept = -
0.002144087, and Slope1 = 0.1494861
F = 3150.522, R^2 = 0.9943191, Model is 1/y^3~I(1/x^3), Intercept =
0.000460766, and Slope1 = -0.05667038
F = 1861.481, R^2 = 0.9904229, Model is 1/y^2~I(1/x^3), Intercept =
0.001092446, and Slope1 = 0.1580089
F = 1077.338, R^2 = 0.9835667, Model is y^2~I(x^3), Intercept = -786.1159,
and Slope1 = 9.361907
F = 849.3212, R^2 = 0.9792464, Model is 1/y^3~I(1/x^2), Intercept =
0.001580754, and Slope1 = -0.05301868
F = 847.2644, R^2 = 0.9791971, Model is y~I(x^3), Intercept = 8.901373, and
Slope1 = 0.09281814
F = 455.8737, R^2 = 0.9620152, Model is 1/y^2~I(1/x), Intercept = -
0.02149645, and Slope1 = 0.1518563
F = 357.5352, R^2 = 0.9520684, Model is y~x, Intercept = -35.24081, and
Slope1 = 12.17854
F = 233.012, R^2 = 0.9282903, Model is y^2~I(x^2), Intercept = -1889.593, and
Slope1 = 96.9631
```

The function best.lr() succeeds in identifying the correct model:

y = x^2 – 4

Experiment with introducing random errors in the data and then call the best-model selection function. Notice whether or not other models start to compete with the original model as you increase the level of errors.

The best.mlr1() Function

The Function Declaration

The function bes.mlr1() performs the best-model search for three variables. The declaration for this function is:

```
best.mlr1 = function(x1, x2, y, name.x1="x1", name.x2="x2", name.y="y",
                     quiet=FALSE, show.best=20, outfile="C:/best.mlr1.txt")
```

The parameters **x1**, **x2**, and **y** are the data vectors. The parameters **name.x1**, **name.x2,** and **name.y** represent the string names for the vectors x1, x2, and y, and MUST match the names of these vectors. The Boolean parameter **quiet** is a flag that tells the function best.mlr1() not to display any results or write any results to an output file. The parameter **show.best** specified the number of best models to display. The default value of -1 tells the function to display all of the results. The parameter **outfile** specifies the output file. If you pass an empty string to this parameter, you suppress file output.

The Source Code

Here is the source code for the function best.mlr1():

```
best.mlr1 = function(x1, x2, y, name.x1="x1", name.x2="x2", name.y="y",
                     quiet=FALSE, show.best=20, outfile="C:/best.mlr1.txt")
{
  # x1, x2, and y are data vectors.
  # name.x1 is the name for argument for parameter x1
  # name.x2 is the name for argument for parameter x2
  # name.y is the name for argument for parameter y
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #  Default shows 20 best
  # outfile is the output filename

  max.models=9^3
  max.res=5
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  v = rep(0,max.res) # dummy vector used for swapping rows in mat.res
  # initialize number of models found
  n.models = 0
  # check for zero and negative values in arrays
  x1.has.zero = has.zero(x1)
  x1.has.neg = has.neg(x1)
  x2.has.zero = has.zero(x2)
  x2.has.neg = has.neg(x2)
  y.has.zero = has.zero(y)
  y.has.neg = has.neg(y)
  for (iy in -4:4) {
    # check for zero value restraint
    if (!(y.has.zero & iy <= 0)) {
      # check for negative value restraint
      if (!(y.has.neg & (iy == 4 | iy == -4 | iy == 0))) {
        for (ix1 in -4:4) {
          # check for zero value restraint
          if (!(x1.has.zero & ix1 <= 0)) {
            # check for negative value restraint
            if (!(x1.has.neg & (ix1 == 4 | ix1 == -4 | ix1 == 0))) {
              for (ix2 in -4:4) {
                # check for zero value restraint
                if (!(x2.has.zero & ix2 <= 0)) {
                  # check for negative value restraint
                  if (!(x2.has.neg & (ix2 == 4 | ix2 == -4 | ix2 == 0))) {
                    # increment number of models
                    n.models = n.models + 1
                    # construct formula
                    formula = paste(say.fy(iy,name.y), "~",
                      say.fx(ix1,name.x1), "+", say.fx(ix2,name.x2), sep="")
                    # perform linearized regression
                    mlr = lm(formula)
                    # get the summary
                    smlr = summary(mlr)
                    # build matrix of results
                    mat.res[n.models,1] = smlr$fstatistic[1]
                    mat.res[n.models,2] = smlr$r.squared
                    for(i in 1:3)
                      mat.res[n.models,i+2] = smlr$coefficients[i]
                    formula.arr[n.models] = formula
                  }
                }
              }
            }
          }
        }
      }
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y)))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Initializes the variables that store the results.
2. Scans the values in vectors x1, x2, and y for zeros and negative values. This task stores the results of these scans to use in the next task.
3. Uses nested for loops to apply all of the transformations on the regression variables. The loops include several if statements that check for zeros and negative values and how they can prevent applying certain transformations.
4. The innermost if statement builds the model for the regression and performs a linearized regression by calling function lm(). The function stores the results of functions lm() and summary(). With this information at hand, the function stores a subset of the total results in matrix mat.res and vector formula.arr.
5. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
6. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.

7. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
8. Returns the list list.res.

A Sample Run

Once you store the function best.mlr1() and the helper functions, load them using the source() function. To test the best.mlr1() function, execute the following commands:

```
> x1=runif(20,1,10)
> x2=runif(20,1,10)
> y=10+x1^2-4/x2
> mlr1.lst = best.mlr1(x1,x2,y,show.best=10)

Best 10 models
Number of observations = 20
F = 2.217171e+32, R^2 = 1, Model is y~I(x1^2)+I(1/x2), Intercept = 10, Slope1
= 1, and Slope2 = -4
F = 1083037, R^2 = 0.9999922, Model is y~I(x1^2)+I(1/sqrt(x2)), Intercept =
11.07334, Slope1 = 1.000045, and Slope2 = -4.315609
F = 383183.6, R^2 = 0.9999778, Model is y~I(x1^2)+I(1/x2^2), Intercept =
9.49005, Slope1 = 0.9996174, and Slope2 = -5.324091
F = 272828.2, R^2 = 0.9999688, Model is y~I(x1^2)+I(log(x2)), Intercept =
7.391099, Slope1 = 0.9999082, and Slope2 = 1.053988
F = 143839.4, R^2 = 0.999941, Model is y~I(x1^2)+I(1/x2^3), Intercept =
9.363164, Slope1 = 0.9992464, and Slope2 = -7.619819
F = 129808.9, R^2 = 0.9999345, Model is y~I(x1^2)+I(sqrt(x2)), Intercept =
6.950108, Slope1 = 0.9995863, and Slope2 = 0.9366696
F = 81844.68, R^2 = 0.9998962, Model is y~I(x1^2)+x2, Intercept = 8.061064,
Slope1 = 0.9991268, and Slope2 = 0.1920718
F = 49263.76, R^2 = 0.9998275, Model is y~I(x1^2)+I(x2^2), Intercept =
8.687222, Slope1 = 0.998057, and Slope2 = 0.01362967
F = 38581.06, R^2 = 0.9997797, Model is y~I(x1^2)+I(x2^3), Intercept =
8.929396, Slope1 = 0.997082, and Slope2 = 0.001129781
F = 2859.942, R^2 = 0.9970367, Model is log(y)~I(sqrt(x1))+I(1/x2), Intercept
= 0.8217708, Slope1 = 1.220513, and Slope2 = -0.007348296
```

The function best.mlr1() succeeds in identifying the correct model. Experiment with introducing random errors in the data and then call the best-model selection function. Notice whether or not other models start to compete with the original model as you increase the level of errors.

The best.mlr1b() Function

The Function Declaration

The function bes.mlr1b() performs the best-model search for two variables. The declaration for this function is:

```
best.mlr1b = function(x, y, name.x="x", name.y="y",
                     quiet=FALSE, show.best=20, outfile="C:/best.mlr1b.txt")
```

The parameters **x** and **y** are the data vectors. The parameters **name.x** and **name.y** represent the string names for the vectors x and y, and MUST match the names of these vectors. The Boolean parameter **quiet** is a flag that tells the function best.mlr1b() not to display any results or write any results to an output file. The parameter **show.best** specified the number of best models to display. The default value of -1 tells the function to display all of the results. The parameter **outfile** specifies the output file. If you pass an empty string to this parameter, you suppress file output.

The Source Code

Here is the source code for the function best.mlr1b():

```
best.mlr1b = function(x, y, name.x="x", name.y="y",
                     quiet=FALSE, show.best=20, outfile="C:/best.mlr1b.txt")
{
  # x and y are data vectors.
  # name.x is the name for argument for parameter x
  # name.y is the name for argument for parameter y
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #   Default shows 20 best
  # outfile is the output filename

  max.models=9^3
  max.res=5
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  v = rep(0,max.res) # dummy vector used for swapping rows in mat.res
  # initialize number of models found
  n.models = 0
  # check for zero and negative values in arrays
  x.has.zero = has.zero(x)
  x.has.neg = has.neg(x)
  y.has.zero = has.zero(y)
  y.has.neg = has.neg(y)
  for (iy in -4:4) {
    # check for zero value restraint
    if (!(y.has.zero & iy <= 0)) {
      # check for negative value restraint
      if (!(y.has.neg & (iy == 4 | iy == -4 | iy == 0))) {
        for (ix1 in -4:4) {
          # check for zero value restraint
          if (!(x.has.zero & ix1 <= 0)) {
            # check for negative value restraint
            if (!(x.has.neg & (ix1 == 4 | ix1 == -4 | ix1 == 0))) {
              for (ix2 in ix1:4) {
                # avoid the same transformations
                if (ix1 != ix2) {
                  # check for zero value restraint
                  if (!(x.has.zero & ix2 <= 0)) {
                    # check for negative value restraint
                    if (!(x.has.neg & (ix2 == 4 | ix2 == -4 | ix2 == 0))) {
                      # increment number of models
                      n.models = n.models + 1
                      # construct formula
                      formula = paste(say.fy(iy,name.y), "~",
                       say.fx(ix1,name.x),
                       "+", say.fx(ix2,name.x), sep="")
                      # perform linearized regression
                      mlr = lm(formula)
                      # get the summary
                      smlr = summary(mlr)
                      # build matrix of results
                      mat.res[n.models,1] = smlr$fstatistic[1]
                      mat.res[n.models,2] = smlr$r.squared
                      for(i in 1:3)
                        mat.res[n.models,i+2] = smlr$coefficients[i]
                      formula.arr[n.models] = formula
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y)))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Initializes the variables that store the results.
2. Scans the values in vectors x and y for zeros and negative values. This task stores the results of these scans to use in the next task.
3. Uses nested for loops to apply all of the transformations on the regression variables. The loops include several if statements that check for zeros and negative values and how they

can prevent applying certain transformations. In addition, the code avoids redundant and symmetric terms for the vector x. Redundant will have two terms with the same transformation—not a good idea if you want to avoid runtime error. The innermost if statement builds the model for the regression and performs a linearized regression by calling function lm(). The function stores the results of functions lm() and summary(). With this information at hand, the function stores a subset of the total results in matrix mat.res and vector formula.arr.

4. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
5. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
6. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
7. Returns the list list.res.

A Sample Run

Once you store the function best.mlr1b() and the helper functions, load them using the source() function. To test the best.mlr1b() function, execute the following commands:

```
> x=runif(20,1,10)
> y=10+x^2-4/x
> mlr1b.lst = best.mlr1b(x,y,show.best=10)

Best 10 models
Number of observations = 20
F = 2.252852e+32, R^2 = 1, Model is y~I(1/x)+I(x^2), Intercept = 10, Slope1 =
-4, and Slope2 = 1
F = 1387419, R^2 = 0.9999939, Model is y~I(1/sqrt(x))+I(x^2), Intercept =
12.07094, Slope1 = -5.882417, and Slope2 = 0.9926577
F = 593573, R^2 = 0.9999857, Model is y~I(1/x^2)+I(x^2), Intercept = 8.99292,
Slope1 = -3.155545, and Slope2 = 1.008332
F = 330034.9, R^2 = 0.9999742, Model is y~I(log(x))+I(x^2), Intercept =
6.452223, Slope1 = 2.040739, and Slope2 = 0.9820683
F = 251955.2, R^2 = 0.9999663, Model is 1/y^3~I(1/sqrt(x))+I(1/x^2),
Intercept = 0.0001569481, Slope1 = -0.0005883813, and Slope2 = 0.003325193
F = 243637.1, R^2 = 0.9999651, Model is y~I(1/x^3)+I(x^2), Intercept =
8.718859, Slope1 = -2.914116, and Slope2 = 1.011822
F = 199658.5, R^2 = 0.9999574, Model is 1/y^3~I(1/x^2)+I(log(x)), Intercept =
-0.0003042402, Slope1 = 0.003194018, and Slope2 = 0.0001231346
F = 174971.1, R^2 = 0.9999514, Model is 1/y^2~I(1/x)+I(log(x)), Intercept = -
0.01118981, Slope1 = 0.031434, and Slope2 = 0.003573156
F = 171422.6, R^2 = 0.9999504, Model is sqrt(y)~x+I(x^2), Intercept =
1.945761, Slope1 = 0.7125468, and Slope2 = 0.01429142
F = 150571.7, R^2 = 0.9999436, Model is y~I(x^2)+I(sqrt(x)), Intercept =
4.020623, Slope1 = 0.9659033, and Slope2 = 2.736114
```

The function best.mlr1b() succeeds in identifying the correct model. Experiment with introducing random errors in the data and then call the best-model selection function. Notice whether or not other models start to compete with the original model as you increase the level of errors.

The best.mlr2() Function

The Function Declaration

The function bes.mlr2() performs the best-model search for four variables. The declaration for this function is:

```
best.mlr2 = function(x1, x2, x3, y, name.x1="x1", name.x2="x2", name.x3="x3",
                     name.y="y", quiet=FALSE, show.best=20,
                     outfile="C:/best.mlr2.txt")
```

The parameters **x1**, **x2**, **x3**, and **y** are the data vectors. The parameters **name.x1**, **name.x2, name.x3,** and **name.y** represent the string names for the vectors x1, x2, x3, and y, and MUST match the names of these vectors. The Boolean parameter **quiet** is a flag that tells the function best.mlr2() not to display any results or write any results to an output file. The parameter **show.best** specified the number of best models to display. The default value of -1 tells the function to display all of the results. The parameter **outfile** specifies the output file. If you pass an empty string to this parameter, you suppress file output.

The Source Code

Here is the source code for the function best.mlr2():

```
best.mlr2 = function(x1, x2, x3, y, name.x1="x1", name.x2="x2", name.x3="x3",
                     name.y="y", quiet=FALSE, show.best=20,
outfile="C:/best.mlr2.txt")
{
  # x1, x2, x3, and y are data vectors.
  # name.x1 is the name for argument for parameter x1
  # name.x2 is the name for argument for parameter x2
  # name.x3 is the name for argument for parameter x3
  # name.y is the name for argument for parameter y
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #   Default shows 20 best
  # outfile is the output filename

  max.models=9^4
  max.res=6
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  v = rep(0,max.res) # dummy vector used for swapping rows in mat.res
  # initialize number of models found
  n.models = 0
  # check for zero and negative values in arrays
  x1.has.zero = has.zero(x1)
  x1.has.neg = has.neg(x1)
  x2.has.zero = has.zero(x2)
  x2.has.neg = has.neg(x2)
  x3.has.zero = has.zero(x3)
  x3.has.neg = has.neg(x3)
  y.has.zero = has.zero(y)
  y.has.neg = has.neg(y)
  for (iy in -4:4) {
    # check for zero value restraint
    if (!(y.has.zero & iy <= 0)) {
      # check for negative value restraint
      if (!(y.has.neg & (iy == 4 | iy == -4 | iy == 0))) {
        for (ix1 in -4:4) {
          # check for zero value restraint
          if (!(x1.has.zero & ix1 <= 0)) {
            # check for negative value restraint
            if (!(x1.has.neg & (ix1 == 4 | ix1 == -4 | ix1 == 0))) {
              for (ix2 in -4:4) {
                # check for zero value restraint
                if (!(x2.has.zero & ix2 <= 0)) {
                  # check for negative value restraint
                  if (!(x2.has.neg & (ix2 == 4 | ix2 == -4 | ix2 == 0))) {
                    for (ix3 in -4:4) {
                      # check for zero value restraint
                      if (!(x3.has.zero & ix3 <= 0)) {
                        # check for negative value restraint
                        if (!(x3.has.neg & (ix3 == 4 | ix3 == -4 |
                            ix3 == 0))) {
                          # increment number of models
                          n.models = n.models + 1
                          # construct formula
                          formula = paste(say.fy(iy,name.y), "~",
                            say.fx(ix1,name.x1),
                            "+", say.fx(ix2,name.x2),
                            "+", say.fx(ix3,name.x3), sep="")
                          # perform linearized regression
                          mlr = lm(formula)
                          # get the summary
                          smlr = summary(mlr)
                          # build matrix of results
                          mat.res[n.models,1] = smlr$fstatistic[1]
                          mat.res[n.models,2] = smlr$r.squared
                          for(i in 1:4)
                            mat.res[n.models,i+2] = smlr$coefficients[i]
                          formula.arr[n.models] = formula
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y)))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Initializes the variables that store the results.
2. Scans the values in vectors x1, x2, x3, and y for zeros and negative values. This task stores the results of these scans to use in the next task.
3. Uses nested for loops to apply all of the transformations on the regression variables. The loops include several if statements that check for zeros and negative values and how they can prevent applying certain transformations.
4. The innermost if statement builds the model for the regression and performs a linearized regression by calling function lm(). The function stores the results of functions lm() and summary(). With this information at hand, the function stores a subset of the total results in matrix mat.res and vector formula.arr.
5. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
6. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
7. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
8. Returns the list list.res.

A Sample Run

Once you store the function best.mlr2() and the helper functions, load them using the source() function. To test the best.mlr2() function, execute the following commands (keep in mind that the process may last several seconds):

```
> x1=runif(20,1,10)
> x2=runif(20,1,10)
> x3=runif(20,10,100)
> y=10+x1^2-4/x2+log(x3)
> mlr2.lst = best.mlr2(x1,x2,x3,y,show.best=10)

Best 10 models
Number of observations = 20
F = 3.929575e+32, R^2 = 1, Model is y~I(x1^2)+I(1/x2)+I(log(x3)), Intercept =
10, Slope1 = 1, Slope2 = -4, and Slope3 = 1
F = 4437158, R^2 = 0.9999988, Model is y~I(x1^2)+I(1/x2)+I(sqrt(x3)),
Intercept = 11.94097, Slope1 = 0.9997764, Slope2 = -3.894320, and Slope3 =
0.2709474
F = 4414091, R^2 = 0.9999988, Model is y~I(x1^2)+I(1/x2)+I(1/sqrt(x3)),
Intercept = 15.96884, Slope1 = 1.000049, Slope2 = -4.084705, and Slope3 = -
14.13160
F = 1247116, R^2 = 0.9999957, Model is y~I(x1^2)+I(1/sqrt(x2))+I(log(x3)),
Intercept = 11.17688, Slope1 = 0.9996687, Slope2 = -4.150386, and Slope3 =
0.9562958
F = 1166913, R^2 = 0.9999954, Model is y~I(x1^2)+I(1/x2)+x3, Intercept =
12.92946, Slope1 = 0.999409, Slope2 = -3.777364, and Slope3 = 0.01759749
F = 1154382, R^2 = 0.9999954, Model is y~I(x1^2)+I(1/x2)+I(1/x3), Intercept =
14.98382, Slope1 = 0.9999235, Slope2 = -4.142409, and Slope3 = -47.86012
F = 1139778, R^2 = 0.9999953, Model is y~I(x1^2)+I(1/sqrt(x2))+I(1/sqrt(x3)),
Intercept = 16.91325, Slope1 = 0.9997394, Slope2 = -4.243904, and Slope3 = -
13.54852
F = 877326.9, R^2 = 0.999994, Model is y~I(x1^2)+I(1/sqrt(x2))+I(sqrt(x3)),
Intercept = 13.00907, Slope1 = 0.9994353, Slope2 = -4.037534, and Slope3 =
0.2585578
F = 745153, R^2 = 0.9999928, Model is y~I(x1^2)+I(1/sqrt(x2))+I(1/x3),
Intercept = 15.98892, Slope1 = 0.9996456, Slope2 = -4.311318, and Slope3 = -
46.01706
F = 543471.6, R^2 = 0.9999902, Model is y~I(x1^2)+I(1/sqrt(x2))+x3, Intercept
= 13.92392, Slope1 = 0.99907, Slope2 = -3.915408, and Slope3 = 0.01676484
```

The function best.mlr2() succeeds in identifying the correct model. Experiment with introducing random errors in the data and then call the best-model selection function. Notice whether or not other models start to compete with the original model as you increase the level of errors.

The best.mlr2b() Function

The Function Declaration

The function bes.mlr2b() performs the best-model search for two variables. The declaration for this function is:

```
best.mlr2b = function(x, y, name.x="x", name.y="y", quiet=FALSE,
                      show.best=20, outfile="C:/best.mlr1.txt")
```

The parameters **x** and **y** are the data vectors. The parameters **name.x** and **name.y** represent the string names for the vectors x and y, and MUST match the names of these vectors. The Boolean parameter **quiet** is a flag that tells the function best.mlr2b() not to display any results or write any results to an output file. The parameter **show.best** specified the number of best models to display. The default value of -1 tells the function to display all of the results. The parameter **outfile** specifies the output file. If you pass an empty string to this parameter, you suppress file output.

The Source Code

Here is the source code for the function best.mlr2b():

```
best.mlr2b = function(x, y, name.x="x", name.y="y", quiet=FALSE,
                      show.best=20, outfile="C:/best.mlr2b.txt")
{
  # x and y are data vectors
  # name.x is the name for argument for parameter x
  # name.y is the name for argument for parameter y
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #   Default shows 20 best
  # outfile is the output filename

  max.models=9^4
  max.res=6
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  v = rep(0,max.res) # dummy vector used for swapping rows in mat.res
  # initialize number of models found
  n.models = 0
  # check for zero and negative values in arrays
  x.has.zero = has.zero(x)
  x.has.neg = has.neg(x)
  y.has.zero = has.zero(y)
  y.has.neg = has.neg(y)
  for (iy in -4:4) {
    # check for zero value restraint
    if (!(y.has.zero & iy <= 0)) {
      # check for negative value restraint
      if (!(y.has.neg & (iy == 4 | iy == -4 | iy == 0))) {
        for (ix1 in -4:4) {
          # check for zero value restraint
          if (!(x.has.zero & ix1 <= 0)) {
            # check for negative value restraint
            if (!(x.has.neg & (ix1 == 4 | ix1 == -4 | ix1 == 0))) {
              for (ix2 in ix1:4) {
                if (ix1 != ix2) {
                # check for zero value restraint
                if (!(x.has.zero & ix2 <= 0)) {
                  # check for negative value restraint
                  if (!(x.has.neg & (ix2 == 4 | ix2 == -4 | ix2 == 0))) {
                    for (ix3 in ix2:4) {
                      if (ix2 != ix3) {
                      # check for zero value restraint
                      if (!(x.has.zero & ix3 <= 0)) {
                        # check for negative value restraint
                        if (!(x.has.neg & (ix3 == 4 | ix3 == -4 |
                            ix3 == 0))) {
                          # increment number of models
                          n.models = n.models + 1
                          # construct formula
                          formula = paste(say.fy(iy,name.y), "~",
                            say.fx(ix1,name.x),
                            "+", say.fx(ix2,name.x),
                            "+", say.fx(ix3,name.x), sep="")
                          # perform linearized regression
                          mlr = lm(formula)
                          # get the summary
                          smlr = summary(mlr)
                          # build matrix of results
                          mat.res[n.models,1] = smlr$fstatistic[1]
                          mat.res[n.models,2] = smlr$r.squared
                          for(i in 1:4)
                            mat.res[n.models,i+2] = smlr$coefficients[i]
                          formula.arr[n.models] = formula
                        }
                      }
                      }
                    }
                  }
                }
                }
              }
            }
          }
        }
      }
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y)))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Initializes the variables that store the results.
2. Scans the values in vectors x and y for zeros and negative values. This task stores the results of these scans to use in the next task.

3. Uses nested for loops to apply all of the transformations on the regression variables. The loops include several if statements that check for zeros and negative values and how they can prevent applying certain transformations. In addition, the code avoids redundant and symmetric terms for the vector x. Redundant will have two or three terms with the same transformation—not a good idea if you want to avoid runtime error. Symmetric terms generate basically the same model by swapping the transformations for vector x.
4. The innermost if statement builds the model for the regression and performs a linearized regression by calling function lm(). The function stores the results of functions lm() and summary(). With this information at hand, the function stores a subset of the total results in matrix mat.res and vector formula.arr.
5. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
6. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
7. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
8. Returns the list list.res.

A Sample Run

Once you store the function best.mlr2b() and the helper functions, load them using the source() function. To test the best.mlr2b() function, execute the following commands (keep in mind that the process may last several seconds)::

```
> x=runif(20,1,10)
> y=10+x^2-4/x+log(x)
> mlr2b.lst = best.mlr2b(x,y,show.best=10)

Best 10 models
Number of observations = 20
F = 2.326808e+32, R^2 = 1, Model is y~I(1/x)+I(log(x))+I(x^2), Intercept =
10, Slope1 = -4, Slope2 = 1, and Slope3 = 1
F = 5940365075, R^2 = 1, Model is y~I(1/sqrt(x))+I(1/x^2)+I(x^2), Intercept =
14.17140, Slope1 = -7.525488, Slope2 = -0.6526698, and Slope3 = 1.001219
F = 557110918, R^2 = 1, Model is y~I(1/sqrt(x))+I(1/x^3)+I(x^2), Intercept =
14.32506, Slope1 = -7.886738, Slope2 = -0.4705604, and Slope3 = 1.000688
F = 258499620, R^2 = 1, Model is y~I(1/sqrt(x))+I(1/x)+I(x^2), Intercept =
13.69992, Slope1 = -5.692199, Slope2 = -1.990817, and Slope3 = 1.002158
F = 255565991, R^2 = 1, Model is y~I(1/x)+I(x^2)+I(sqrt(x)), Intercept =
9.701272, Slope1 = -4.66813, Slope2 = 0.9964237, and Slope3 = 0.952308
F = 66660631, R^2 = 1, Model is y~I(1/sqrt(x))+I(log(x))+I(x^2), Intercept =
17.25022, Slope1 = -11.21245, Slope2 = -0.9478255, and Slope3 = 1.004052
F = 65981677, R^2 = 1, Model is y~I(1/x)+x+I(x^2), Intercept = 10.70256,
Slope1 = -5.000611, Slope2 = 0.2737962, and Slope3 = 0.9893375
F = 43257973, R^2 = 0.9999999, Model is y~I(1/sqrt(x))+I(x^2)+I(sqrt(x)),
Intercept = 16.54472, Slope1 = -9.836476, Slope2 = 1.005920, and Slope3 = -
0.6621007
```

```
F = 30963365, R^2 = 0.9999998, Model is y~I(1/sqrt(x))+x+I(x^2), Intercept =
15.59766, Slope1 = -9.380408, Slope2 = -0.1655957, and Slope3 = 1.009618
F = 20474489, R^2 = 0.9999997, Model is y~I(1/x^2)+I(log(x))+I(x^2),
Intercept = 7.8443, Slope1 = -1.925660, Slope2 = 1.967620, and Slope3 =
0.9950322
```

The function best.mlr2b() succeeds in identifying the correct model. Experiment with introducing random errors in the data and then call the best-model selection function. Notice whether or not other models start to compete with the original model as you increase the level of errors.

A Second Generation

The next sections present a family of R functions that also perform best-model search but with a different twist. This new generation of functions, which is a spinoff from the older ones, comes in pairs. The first function in each pair writes the regression formulas to a text file. You can then open that text file and edit its contents as follows:

- Eliminate models that you know are not suitable for the regression selection. This task speeds up the search and eliminates the possibility of wading through ineligible (based on your own definition and case study) models that might make it to the top of the best models list.
- Add new models. You need to make sure that the new models conform to the style appearing in the file. You must follow the same names of variables and should use the same number of terms.
- Include custom values for scaling and shifting for specific models. Earlier in the tutorial I mentioned that imposing values to shift and scale regression variables, across the board, may put some of the models at a disadvantage. By targeting particular models and injecting specific values for shifting and scaling, you have better chances in enhancing these models. Even if your assessment for shifting and scaling variables is off, they will affect only those models that you target.

The next table lists the set of functions that I present in the second half of this tutorial.

Table 3. The paired set of best-model search functions.

Name of R Function	*Total Number of Variables*	*Scope of Model Search*
write.best.lr() and read.best.lr()	2	Searches for the best model: fy(y) = a + b fx(x)
write.best.mlr1() and read.best.mlr1()	3	Searches for the best model:

Name of R Function	***Total Number of Variables***	***Scope of Model Search***
		fy(y) = a + b fx1(x1) + c fx2(x2)
write.best.mlr1b() and read.best.mlr1b()	2	Searches for the best model: fy(y) = a + b fx1(x) + c fx2(x) The functions fx1 and fx2 are different transformations.
write.best.mlr2() and read.best.mlr2()	4	Searches for the best model: fy(y) = a + b fx1(x1) + c fx2(x2) + d fx3(x3)
write.best.mlr2b() and read.best.mlr2b()	2	Searches for the best model: fy(y) = a + b fx1(x) + c fx2(x) + d fx3(x) The functions fx1, fx2, and fx3 are different transformations.

So here we go!

The Functions write.best.lr() and read.best.lr()

The Declaration of Function write.best.lr()

The function write.best.lr() writes the regression models to a files. The function read.best.lr() reads these regression models and uses them with data to perform regression calculations. The declaration for the write.best.lr() function is:

```
write.best.lr = function(name.x="x", name.y="y", outfile="C:/best.lr.dat")
```

The parameters **name.x** and **name.y** represent the names of the dependent and independent variables, respectively. The parameter **outfile** is the output file. The function writes the list of regression models (used later in calling function lm()) to the output file. It is important to point out that the names specified by parameters name.x and name.y must be the same names of the vectors that supply function read.best.lr() with data.

The Source Code

Here is the code for the function write.best.lr():

```
write.best.lr = function(name.x="x", name.y="y", outfile="C:/best.lr.dat")
{
  # setup the models for linearized regression.

  # write header
  cat("", sep="", file=outfile, append=FALSE)
  # write entries
  for (iy in -4:4) {
    for (ix in -4:4) {
      cat(say.fy(iy,name.y), "~", say.fx(ix,name.x),
          "\n", file=outfile, append=TRUE, sep="")
    }
  }
  cat("\n", file=outfile, append=TRUE)

  return (TRUE)
}
```

The Declaration of Function read.best.lr()

The declaration for function read.best.lr():

```
read.best.lr = function(x, y, quiet=FALSE,
                       show.best=-1, infile=" C:/best.lr.dat",
                       outfile="C:/best.lr.txt")
```

The parameters x and y are the data vectors. The arguments for these parameters must match the names specified in calling function write.best.lr(). The parameters **quiet** is the quiet mode flag. The parameter **show.best** tells the function how many best regression models to show. The parameter **infile** specified the model input filename. The parameter **outfile** is the output filename.

The Source Code

Here is the source code for function read.best.lr():

```
read.best.lr = function(x, y, quiet=FALSE,
                       show.best=-1, infile=" C:/best.lr.dat",
                       outfile="C:/best.lr.txt")
{
  # x and y are data vectors.
  # name.x is the name for argument for parameter x
  # name.y is the name for argument for parameter y
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #  Default shows all results
  # infile is the name of the file containing the transformations, shifting,
  #   and scaling data
  # outfile is the output filename

  if (infile=="" | !file.exists(infile)) {
    cat("Input file does not exist. Function aborted.\n")
    return (FALSE)
  }
  # read the formulas from the file
  formula.vect = scan(file=infile, what=character(0))
  max.models=length(formula.vect)
  max.res=4
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  # initialize number of models found
  n.models = 0
  # iterate over the number of elements in vector index.x
  for (i in 1:max.models) {
    lr = 1 # assign a dummy value
    formula = as.character(formula.vect[i])
    # perform linearized regression
    lr = lm(formula)
    # call to function lm() succeeded?
    if (class(lr) == "lm") {
      # increment number of models
      n.models = n.models + 1
      # get the summary
      slr = summary(lr)
      # build matrix of results
      mat.res[n.models,1] = slr$fstatistic[1]
      mat.res[n.models,2] = slr$r.squared
      mat.res[n.models,3] = slr$coefficients[1]
      mat.res[n.models,4] = slr$coefficients[2]
      formula.arr[n.models] = formula
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Exits with a warning message if the parameter outfile is an empty string or refers to a nonexistent file.
2. Reads the array of models and stores the array in vector formula.vect. This task uses function scan() to read the formulas from the input file.
3. Calculates the number of models and stores it in variable max.model.
4. Assigns the number of results to variable max.res.

5. Initializes the matrix mat.res and vector formula.arr that store the results.
6. Initializes the number of models.
7. Starts a loop to iterate over all the available models. The loop performs the following subtasks:
   7.1. Assigns a dummy scalar to variable lr.
   7.2. Stores the current formula in the variable formula.
   7.3. Performs the linearized regression for the current model. This task calls function lm(formula) and stores the result in variable lr.
   7.4. Determines if the class of variable lr is "lm". If this condition is true, then the last task was successful. The function increments the model counter (variable n.models) and then obtains the summary of the regression variables lr and stores the F statistics, coefficient of determination, the regression intercept, and the regression slope in the matrix mat.res. The function also stores the current formula in vector formula.arr.
8. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
9. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
10. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
11. Returns the list list.res.

A Sample Run

Save the write.best.lr() and read.best.lr() functions to a script file and then load that file using the source() function. Invoke the function write.best.lr() to write the regression models to file C:\best.lr.dat by executing the following command:

```
> write.best.lr()
```

Open the file C:\best.lr.dat with a text editor. The first few lines should look like the following:

```
1/sqrt(y)~I(1/sqrt(x))
1/sqrt(y)~I(1/x^3)
1/sqrt(y)~I(1/x^2)
1/sqrt(y)~I(1/x)
1/sqrt(y)~I(log(x))
1/sqrt(y)~x
1/sqrt(y)~I(x^2)
1/sqrt(y)~I(x^3)
1/sqrt(y)~I(sqrt(x))
1/y^3~I(1/sqrt(x))
1/y^3~I(1/x^3)
1/y^3~I(1/x^2)
```

Locate the following line:

```
y~I(x^2)
```

And edit that line to be as follows:

```
y~I(3*(x-1)^2)
```

The above task injects a shift values of -1 and a scale factor of 3 to the model y = a + b x^2, making it y = a + b (3(x-1)^2).

Save the contents of the text file and close your text editor. Type in the following commands to trigger the search for the best model using the models stored in file C:\best.lr.dat:

```
> x=runif(20,1,10)
> y=3*(x-1)^2-5
> lr.list = read.best.lr(x, y, show.best=10, infile="C:/best.lr.dat")
Read 81 items
Best 10 models
Number of observations = 20
F = 1.688182e+33, R^2 = 1, Model is y~I(3*(x-1)^2), Intercept = -5, and
Slope1 = 1
F = 6857.008, R^2 = 0.9978172, Model is sqrt(y)~x, Intercept = -2.964978, and
Slope1 = 1.854829
F = 5210.383, R^2 = 0.9971294, Model is sqrt(y)~I(sqrt(x)), Intercept = -
13.17484, and Slope1 = 8.930317
F = 1691.171, R^2 = 0.9894686, Model is y~I(x^3), Intercept = 5.849399, and
Slope1 = 0.2457168
F = 1372.676, R^2 = 0.9891906, Model is log(y)~I(1/x), Intercept = 7.113554,
and Slope1 = -16.83487
F = 694.4031, R^2 = 0.9747334, Model is y^2~I(x^3), Intercept = -3486.318,
and Slope1 = 54.39975
F = 672.9775, R^2 = 0.978197, Model is sqrt(y)~I(log(x)), Intercept = -
9.063916, and Slope1 = 10.23069
F = 583.2273, R^2 = 0.974926, Model is log(y)~I(1/x^2), Intercept = 5.480176,
and Slope1 = -33.29122
F = 554.0981, R^2 = 0.9736425, Model is log(y)~I(1/sqrt(x)), Intercept =
10.56956, and Slope1 = -15.75582
F = 361.1927, R^2 = 0.9601268, Model is sqrt(y)~I(x^2), Intercept = 2.220991,
and Slope1 = 0.1418674
There were 27 warnings (use warnings() to see them)
```

The first two commands create the data vectors. The model used to create the y vector is:

$$y = 3(x-1)^2 - 5$$

The third command invoke function read.best.lr(). Notice that the results show that the model y~I(3*(x-1)^2) is the best one. This model has an intercept of -5 and a slope of 1. The value of 1 is correct since the best model includes a term with $3(x-1)^2$.

The Functions write.best.mlr1() and read.best.mlr1()

The Declaration of Function write.best.mlr1()

The function write.best.mlr1() writes the regression models for three variables to a files. The function read.best.mlr1() reads these regression models and uses them with data to perform regression calculations. The declaration for the write.best.mlr1() function is:

```
write.best.mlr1 = function(name.x1="x1", name.x2="x2",
                  name.y="y", outfile="C:/best.mlr1.dat")
```

The parameters **name.x1**, **name.x2**, and **name.y** represent the names of the dependent and independent variables. The parameter **outfile** is the output file. The function writes the list of regression models (used later in calling function lm()) to the output file. It is important to point out that the names specified by parameters name.x1, name.x2, and name.y must be the same names of the vectors that supply function read.best.mlr1() with data.

The Source Code

Here is the code for the function write.best.mlr1():

```
write.best.mlr1 = function(name.x1="x1", name.x2="x2",
                  name.y="y", outfile="C:/best.mlr1.dat")
{
  # setup the models for linearized regression.

  # write header
  cat("", sep="", file=outfile, append=FALSE)
  # write entries
  for (iy in -4:4) {
    for (ix1 in -4:4) {
      for (ix2 in -4:4) {
        cat(say.fy(iy,name.y), "~", say.fx(ix1,name.x1),
             "+", say.fx(ix2,name.x2),
             "\n", file=outfile, append=TRUE, sep="")
      }
    }
  }
  cat("\n", file=outfile, append=TRUE)

  return (TRUE)
}
```

The Declaration of Function read.best.mlr1()

The declaration for function read.best.mlr1():

```
read.best.mlr1 = function(x1, x2, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr1.dat",
                   outfile="C:/best.mlr1.txt")
```

The parameters x1, x2, and y are the data vectors. The arguments for these parameters must match the names specified in calling function write.best.mlr1(). The parameters **quiet** is the quiet mode flag. The parameter **show.best** tells the function how many best regression models to show. The parameter **infile** specified the model input filename. The parameter **outfile** is the output filename.

The Source Code

Here is the source code for function read.best.mlr1():

```
read.best.mlr1 = function(x1, x2, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr1.dat",
outfile="C:/best.mlr1.txt")
{
  # x1, x2, and y are data vectors.
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #  Default shows all results
  # infile is the name of the file containing the transformations, shifting,
  #   and scaling data
  # outfile is the output filename

  if (infile=="" | !file.exists(infile)) {
    cat("Input file does not exist. Function aborted.\n")
    return (FALSE)
  }
  # read the formulas from the file
  formula.vect = scan(file=infile, what=character(0))
  max.models=length(formula.vect)
  max.res=5
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  # initialize number of models found
  n.models = 0
  # iterate over the number of elements in vector index.x
  for (i in 1:max.models) {
    mlr = 1 # assign a dummy value
    formula = as.character(formula.vect[i])
    # perform linearized regression
    mlr = lm(formula)
    # call to function lm() succeeded?
    if (class(mlr) == "lm") {
      # increment number of models
      n.models = n.models + 1
      # get the summary
      slr = summary(mlr)
      # build matrix of results
      mat.res[n.models,1] = slr$fstatistic[1]
      mat.res[n.models,2] = slr$r.squared
      for (i in 1:3)
        mat.res[n.models,i+2] = slr$coefficients[i]
      formula.arr[n.models] = formula
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Exits with a warning message if the parameter outfile is an empty string or refers to a nonexistent file.
2. Reads the array of models and stores the array in vector formula.vect. This task uses function scan() to read the formulas from the input file.
3. Calculates the number of models and stores it in variable max.model.
4. Assigns the number of results to variable max.res.
5. Initializes the matrix mat.res and vector formula.arr that store the results.
6. Initializes the number of models.
7. Starts a loop to iterate over all the available models. The loop performs the following subtasks:
   - 7.1. Assigns a dummy scalar to variable mlr.
   - 7.2. Stores the current formula in the variable formula.
   - 7.3. Performs the linearized regression for the current model. This task calls function lm(formula) and stores the result in variable mlr.
   - 7.4. Determines if the class of variable mlr is "lm". If this condition is true, then the last task was successful. The function increments the model counter (variable n.models) and then obtains the summary of the regression variables mlr and stores the F statistics, coefficient of determination, the regression intercept, and the regression slopes in the matrix mat.res. The function also stores the current formula in vector formula.arr.
8. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
9. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.

10. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
11. Returns the list list.res.

A Sample Run

Save the write.best.mlr1() and read.best.mlr1() functions to a script file and then load that file using the source() function. Invoke the function write.best.mlr1() to write the regression models to file C:\best.mlr1.dat by executing the following command:

```
> write.best.mlr1()
```

Open the file C:\best.mlr1.dat with a text editor. The first few lines should look like the following:

```
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^2)
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2)
1/sqrt(y)~I(1/sqrt(x1))+I(log(x2))
1/sqrt(y)~I(1/sqrt(x1))+x2
1/sqrt(y)~I(1/sqrt(x1))+I(x2^2)
1/sqrt(y)~I(1/sqrt(x1))+I(x2^3)
1/sqrt(y)~I(1/sqrt(x1))+I(sqrt(x2))
1/sqrt(y)~I(1/x1^3)+I(1/sqrt(x2))
1/sqrt(y)~I(1/x1^3)+I(1/x2^3)
1/sqrt(y)~I(1/x1^3)+I(1/x2^2)
```

Type in the following commands to trigger the search for the best model using the models stored in file C:\best.mlr1.dat:

```
> x1=runif(20,2,10)
> x2=runif(20,0.1,1)
> y=10+2*log(x1)-5/x2
> mlr1.list = read.best.mlr1(x1,x2,y,show.best=10, infile="C:/best.mlr1.dat")
Read 729 items
Best 10 models
Number of observations = 20
F = 1.933720e+32, R^2 = 1, Model is y~I(log(x1))+I(1/x2), Intercept = 10,
Slope1 = 2, and Slope2 = -5
F = 182338.3, R^2 = 0.9999534, Model is y~I(sqrt(x1))+I(1/x2), Intercept =
9.299606, Slope1 = 1.722137, and Slope2 = -4.998122
F = 181608.6, R^2 = 0.9999532, Model is y~I(1/sqrt(x1))+I(1/x2), Intercept =
17.28138, Slope1 = -8.905054, and Slope2 = -5.002794
F = 47129.23, R^2 = 0.9998197, Model is y~x1+I(1/x2), Intercept = 11.31914,
Slope1 = 0.355858, and Slope2 = -4.997079
F = 46880.65, R^2 = 0.9998187, Model is y~I(1/x1)+I(1/x2), Intercept =
15.28458, Slope1 = -9.504124, and Slope2 = -5.006428
F = 13321.16, R^2 = 0.9993623, Model is y~I(1/x1^2)+I(1/x2), Intercept =
14.31233, Slope1 = -19.29295, and Slope2 = -5.015226
F = 13234.67, R^2 = 0.9993582, Model is y~I(x1^2)+I(1/x2), Intercept =
12.36905, Slope1 = 0.02721633, and Slope2 = -4.996697
```

```
F = 7177.315, R^2 = 0.9988171, Model is y~I(1/x1^3)+I(1/x2), Intercept =
14.02298, Slope1 = -46.1872, and Slope2 = -5.024046
F = 6926.138, R^2 = 0.9987743, Model is y~I(x1^3)+I(1/x2), Intercept =
12.75296, Slope1 = 0.002461737, and Slope2 = -4.997191
F = 1631.578, R^2 = 0.9966404, Model is sqrt(y)~I(log(x1))+I(1/x2^2),
Intercept = 2.46595, Slope1 = 0.4297847, and Slope2 = -0.3398041
There were 50 or more warnings (use warnings() to see the first 50)
```

The first two commands create the data vectors. The third command invoke function read.best.mlr1(). The function locates the best model based on the actual data as defined in the third command.

The Functions write.best.mlr1b() and read.best.mlr1b()

The Declaration of Function write.best.mlr1b()

The function write.best.mlr1b() writes the regression models for two variables to a files. The function read.best.mlr1b() reads these regression models and uses them with data to perform regression calculations. The declaration for the write.best.mlr1b() function is:

```
write.best.mlr1b = function(name.x="x", name.y="y",
                            outfile="C:/best.mlr1b.dat")
```

The parameters **name.x** and **name.y** represent the names of the dependent and independent variables. The parameter **outfile** is the output file. The function writes the list of regression models (used later in calling function lm()) to the output file. It is important to point out that the names specified by parameters name.x and name.y must be the same names of the vectors that supply function read.best.mlr1b() with data.

The Source Code

Here is the code for the function write.best.mlr1b():

```
write.best.mlr1b = function(name.x="x", name.y="y",
                            outfile="C:/best.mlr1b.dat")
{
  # setup the models for linearized regression.

  # write header
  cat("", sep="", file=outfile, append=FALSE)
  # write entries
  for (iy in -4:4) {
    for (ix1 in -4:4) {
      for (ix2 in ix1:4) {
        if (ix1 != ix2) {
          cat(say.fy(iy, name.y), "~", say.fx(ix1, name.x),
             "+", say.fx(ix2, name.x),
             "\n", file=outfile, append=TRUE, sep="")
        }
      }
    }
  }
  cat("\n", file=outfile, append=TRUE)

  return (TRUE)
}
```

The Declaration of Function read.best.mlr1b()

The declaration for function read.best.mlr1b():

```
read.best.mlr1b = function(x, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr1b.dat",
                   outfile="C:/best.mlr1b.txt")
```

The parameters **x** and **y** are the data vectors. The arguments for these parameters must match the names specified in calling function write.best.mlr1b(). The parameters **quiet** is the quiet mode flag. The parameter **show.best** tells the function how many best regression models to show. The parameter **infile** specified the model input filename. The parameter **outfile** is the output filename.

The Source Code

Here is the source code for function read.best.mlr1b():

```
read.best.mlr1b = function(x, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr1b.dat",
                   outfile="C:/best.mlr1b.txt")
{
  # x and y are data vectors.
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #  Default shows all results
  # infile is the name of the file containing the transformations, shifting,
  #   and scaling data
  # outfile is the output filename

  if (infile=="" | !file.exists(infile)) {
    cat("Input file does not exist. Function aborted.\n")
    return (FALSE)
  }
  # read the formulas from the file
  formula.vect = scan(file=infile, what=character(0))
  max.models=length(formula.vect)
  max.res=5
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  # initialize number of models found
  n.models = 0
  # iterate over the number of elements in vector index.x
  for (i in 1:max.models) {
    mlr = 1 # assign a dummy value
    formula = as.character(formula.vect[i])
    # perform linearized regression
    mlr = lm(formula)
    # call to function lm() succeeded?
    if (class(mlr) == "lm") {
      # increment number of models
      n.models = n.models + 1
      # get the summary
      slr = summary(mlr)
      # build matrix of results
      mat.res[n.models,1] = slr$fstatistic[1]
      mat.res[n.models,2] = slr$r.squared
      for (i in 1:3)
        mat.res[n.models,i+2] = slr$coefficients[i]
      formula.arr[n.models] = formula
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Exits with a warning message if the parameter outfile is an empty string or refers to a nonexistent file.
2. Reads the array of models and stores the array in vector formula.vect. This task uses function scan() to read the formulas from the input file.
3. Calculates the number of models and stores it in variable max.model.
4. Assigns the number of results to variable max.res.
5. Initializes the matrix mat.res and vector formula.arr that store the results.
6. Initializes the number of models.
7. Starts a loop to iterate over all the available models. The loop performs the following subtasks:
   7.1. Assigns a dummy scalar to variable mlr.
   7.2. Stores the current formula in the variable formula.

   7.3. Performs the linearized regression for the current model. This task calls function lm(formula) and stores the result in variable mlr.
   7.4. Determines if the class of variable mlr is "lm". If this condition is true, then the last task was successful. The function increments the model counter (variable n.models) and then obtains the summary of the regression variables mlr and stores the F statistics, coefficient of determination, the regression intercept, and the regression slopes in the matrix mat.res. The function also stores the current formula in vector formula.arr.
8. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
9. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
10. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
11. Returns the list list.res.

A Sample Run

Save the write.best.mlr1b() and read.best.mlr1b() functions to a script file and then load that file using the source() function. Invoke the function write.best.mlr1b() to write the regression models to file C:\best.mlr1b.dat by executing the following command:

```
> write.best.mlr1b()
```

Open the file C:\best.mlr1b.dat with a text editor. The first few lines should look like the following:

```
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^2)
1/sqrt(y)~I(1/sqrt(x))+I(1/x)
1/sqrt(y)~I(1/sqrt(x))+I(log(x))
1/sqrt(y)~I(1/sqrt(x))+x
1/sqrt(y)~I(1/sqrt(x))+I(x^2)
1/sqrt(y)~I(1/sqrt(x))+I(x^3)
1/sqrt(y)~I(1/sqrt(x))+I(sqrt(x))
1/sqrt(y)~I(1/x^3)+I(1/x^2)
1/sqrt(y)~I(1/x^3)+I(1/x)
1/sqrt(y)~I(1/x^3)+I(log(x))
1/sqrt(y)~I(1/x^3)+x
1/sqrt(y)~I(1/x^3)+I(x^2)
1/sqrt(y)~I(1/x^3)+I(x^3)
```

Type in the following commands to trigger the search for the best model using the models stored in file C:\best.mlr1b.dat:

```
> x=runif(20,2,10)
> y=10+2*log(x)-5/x
> mlr1b.list = read.best.mlr1b(x, y,show.best=10, infile="C:/best.mlr1b.dat")
Read 324 items
Best 10 models
```

```
Number of observations = 20
F = 2.55444e+31, R^2 = 1, Model is y~I(1/x)+I(log(x)), Intercept = 10, Slope1
= -5, and Slope2 = 2
F = 26773360, R^2 = 0.9999997, Model is y^2~I(1/sqrt(x))+I(log(x)), Intercept
= 76.63747, Slope1 = -56.40767, and Slope2 = 60.82328
F = 26553765, R^2 = 0.9999997, Model is y^3~I(1/sqrt(x))+I(log(x)), Intercept
= -1927.299, Slope1 = 1964.51, and Slope2 = 1785.043
F = 22149818, R^2 = 0.9999996, Model is y^2~I(1/x)+I(log(x)), Intercept =
46.66905, Slope1 = -29.41601, and Slope2 = 67.41334
F = 7996046, R^2 = 0.999999, Model is y~I(1/sqrt(x))+x, Intercept = 17.30283,
Slope1 = -12.05312, and Slope2 = 0.06149608
F = 7013914, R^2 = 0.9999988, Model is log(y)~I(1/x)+I(sqrt(x)), Intercept =
2.549257, Slope1 = -0.8931135, and Slope2 = 0.05921979
F = 3728393, R^2 = 0.9999977, Model is sqrt(y)~I(1/x)+I(log(x)), Intercept =
3.32907, Slope1 = -1.006947, and Slope2 = 0.2287091
F = 1864052, R^2 = 0.9999954, Model is 1/sqrt(y)~I(1/sqrt(x))+I(1/x^3),
Intercept = 0.2199143, Slope1 = 0.1462053, and Slope2 = 0.1018720
F = 1842571, R^2 = 0.9999954, Model is y^3~I(1/x)+I(log(x)), Intercept = -
882.4722, Slope1 = 1022.53, and Slope2 = 1555.099
F = 1716155, R^2 = 0.999995, Model is log(y)~I(1/sqrt(x))+I(1/x^3), Intercept
= 2.982848, Slope1 = -1.06629, and Slope2 = -0.3540008
```

The first two commands create the data vectors. The third command invoke function read.best.mlr1b(). The function locates the best model based on the actual data as defined in the second command.

The Functions write.best.mlr2() and read.best.mlr2()

The Declaration of Function write.best.mlr2()

The function write.best.mlr2() writes the regression models for four variables to a files. The function read.best.mlr2() reads these regression models and uses them with data to perform regression calculations. The declaration for the write.best.mlr2() function is:

```
write.best.mlr2 = function(name.x1="x1", name.x2="x2", name.x3="x3",
                  name.y="y", outfile="C:/best.mlr2.dat")
```

The parameters **name.x1**, **name.x2**, **name.x3**, and **name.y** represent the names of the dependent and independent variables. The parameter **outfile** is the output file. The function writes the list of regression models (used later in calling function lm()) to the output file. It is important to point out that the names specified by parameters name.x1, name.x2, name.x3, and name.y must be the same names of the vectors that supply function read.best.mlr2() with data.

The Source Code

Here is the code for the function write.best.mlr2():

```
write.best.mlr2 = function(name.x1="x1", name.x2="x2", name.x3="x3",
                     name.y="y", outfile="C:/best.mlr2.dat")
{
  # setup the models for linearized regression.

  # write header
  cat("", sep="", file=outfile, append=FALSE)
  # write entries
  for (iy in -4:4) {
    for (ix1 in -4:4) {
      for (ix2 in -4:4) {
        for (ix3 in -4:4) {
          cat(say.fy(iy,name.y), "~", say.fx(ix1,name.x1),
             "+", say.fx(ix2,name.x2),
             "+", say.fx(ix3,name.x3),
             "\n", file=outfile, append=TRUE, sep="")
        }
      }
    }
  }
  cat("\n", file=outfile, append=TRUE)

  return (TRUE)
}
```

The Declaration of Function read.best.mlr2()

The declaration for function read.best.mlr2():

```
read.best.mlr2 = function(x1, x2, x3, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr2.dat",
                   outfile="C:/best.mlr2.txt")
```

The parameters x1, x2, x3, and y are the data vectors. The arguments for these parameters must match the names specified in calling function write.best.mlr2(). The parameters **quiet** is the quiet mode flag. The parameter **show.best** tells the function how many best regression models to show. The parameter **infile** specified the model input filename. The parameter **outfile** is the output filename.

The Source Code

Here is the source code for function read.best.mlr2():

```
read.best.mlr2 = function(x1, x2, x3, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr2.dat",
                   outfile="C:/best.mlr2.txt")
{
  # x1, x2, x3, and y are data vectors.
  # name.x1 is the name for argument for parameter x1
  # name.x2 is the name for argument for parameter x2
  # name.x3 is the name for argument for parameter x2
  # name.y is the name for argument for parameter y
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show.
  #  Default shows all results
  # infile is the name of the file containing the transformations, shifting,
  #   and scaling data
  # outfile is the output filename

  if (infile=="" | !file.exists(infile)) {
    cat("Input file does not exist. Function aborted.\n")
    return (FALSE)
  }
  # read the formulas from the file
  formula.vect = scan(file=infile, what=character(0))
  max.models=length(formula.vect)
  max.res=6
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  # initialize number of models found
  n.models = 0
  # iterate over the number of elements in vector index.x
  for (i in 1:max.models) {
    mlr = 1 # assign a dummy value
    formula = as.character(formula.vect[i])
    # perform linearized regression
    mlr = lm(formula)
    # call to function lm() succeeded?
    if (class(mlr) == "lm") {
      # increment number of models
      n.models = n.models + 1
      # get the summary
      slr = summary(mlr)
      # build matrix of results
      mat.res[n.models,1] = slr$fstatistic[1]
      mat.res[n.models,2] = slr$r.squared
      for (i in 1:4)
        mat.res[n.models,i+2] = slr$coefficients[i]
      formula.arr[n.models] = formula
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Exits with a warning message if the parameter outfile is an empty string or refers to a nonexistent file.
2. Reads the array of models and stores the array in vector formula.vect. This task uses function scan() to read the formulas from the input file.
3. Calculates the number of models and stores it in variable max.model.
4. Assigns the number of results to variable max.res.
5. Initializes the matrix mat.res and vector formula.arr that store the results.
6. Initializes the number of models.
7. Starts a loop to iterate over all the available models. The loop performs the following subtasks:
   7.1. Assigns a dummy scalar to variable mlr.
   7.2. Stores the current formula in the variable formula.
   7.3. Performs the linearized regression for the current model. This task calls function lm(formula) and stores the result in variable mlr.
   7.4. Determines if the class of variable mlr is "lm". If this condition is true, then the last task was successful. The function increments the model counter (variable n.models) and then obtains the summary of the regression variables mlr and stores the F statistics, coefficient of determination, the regression intercept, and the regression slopes in the matrix mat.res. The function also stores the current formula in vector formula.arr.
8. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
9. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
10. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
11. Returns the list list.res.

A Sample Run

Save the write.best.mlr2() and read.best.mlr2() functions to a script file and then load that file using the source() function. Invoke the function write.best.mlr2() to write the regression models to file C:\best.mlr2.dat by executing the following command:

```
> write.best.mlr2()
```

Open the file C:\best.mlr2.dat with a text editor. The first few lines should look like the following:

```
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(1/sqrt(x3))
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(1/x3^3)
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(1/x3^2)
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(1/x3)
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(log(x3))
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+x3
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(x3^2)
```

```
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(x3^3)
1/sqrt(y)~I(1/sqrt(x1))+I(1/sqrt(x2))+I(sqrt(x3))
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(1/sqrt(x3))
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(1/x3^3)
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(1/x3^2)
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(1/x3)
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(log(x3))
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+x3
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(x3^2)
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(x3^3)
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^3)+I(sqrt(x3))
1/sqrt(y)~I(1/sqrt(x1))+I(1/x2^2)+I(1/sqrt(x3))
```

Type in the following commands to trigger the search for the best model using the models stored in file C:\best.mlr2.dat:

```
> x1=runif(20,1,10)
> x2=runif(20,1,10)
> x3=runif(20,10,100)
> y=10+x1^2-4/x2+log(x3)
> mlr2.lst = read.best.mlr2(x1,x2,x3,y,show.best=10)
Read 6561 items
Best 10 models
Number of observations = 20
F = 2.489478e+32, R^2 = 1, Model is y~I(x1^2)+I(1/x2)+I(log(x3)), Intercept =
10, Slope1 = 1, Slope2 = -4, and Slope3 = 1
F = 3134918, R^2 = 0.9999983, Model is y~I(x1^2)+I(1/x2)+I(1/sqrt(x3)),
Intercept = 15.84792, Slope1 = 1.000514, Slope2 = -4.038379, and Slope3 = -
13.30377
F = 2967565, R^2 = 0.9999982, Model is y~I(x1^2)+I(1/x2)+I(sqrt(x3)),
Intercept = 11.82381, Slope1 = 0.9995311, Slope2 = -3.969937, and Slope3 =
0.2874713
F = 1854936, R^2 = 0.9999971, Model is y~I(x1^2)+I(1/sqrt(x2))+I(log(x3)),
Intercept = 10.89389, Slope1 = 1.000469, Slope2 = -4.118693, and Slope3 =
1.016618
F = 1682527, R^2 = 0.9999968, Model is y~I(x1^2)+I(1/sqrt(x2))+I(1/sqrt(x3)),
Intercept = 16.86046, Slope1 = 1.000987, Slope2 = -4.170913, and Slope3 = -
13.56957
F = 881258, R^2 = 0.999994, Model is y~I(x1^2)+I(1/sqrt(x2))+I(sqrt(x3)),
Intercept = 12.74156, Slope1 = 0.9999996, Slope2 = -4.075698, and Slope3 =
0.2913848
F = 833095, R^2 = 0.9999936, Model is y~I(x1^2)+I(1/x2)+I(1/x3), Intercept =
14.86380, Slope1 = 1.001032, Slope2 = -4.08376, and Slope3 = -42.39287
F = 817006, R^2 = 0.9999935, Model is y~I(x1^2)+I(1/sqrt(x2))+I(1/x3),
Intercept = 15.87669, Slope1 = 1.001514, Slope2 = -4.230354, and Slope3 = -
43.39327
F = 749183.9, R^2 = 0.9999929, Model is y~I(x1^2)+I(1/x2)+x3, Intercept =
12.81644, Slope1 = 0.999143, Slope2 = -3.947637, and Slope3 = 0.01975666
F = 734049.5, R^2 = 0.9999927, Model is y~I(x1^2)+I(1/x2^2)+I(sqrt(x3)),
Intercept = 11.44787, Slope1 = 0.9985698, Slope2 = -6.434313, and Slope3 =
0.2828081
```

The first two commands create the data vectors. The third command invoke function read.best.mlr2(). The function locates the best model based on the actual data as defined in the fourth command.

The Functions write.best.mlr2b() and read.best.mlr2b()

The Declaration of Function write.best.mlr2b()

The function write.best.mlr2b() writes the regression models for two variables to a files. The function read.best.mlr2b() reads these regression models and uses them with data to perform regression calculations. The declaration for the write.best.mlr2b() function is:

```
write.best.mlr2b = function(name.x="x", name.y="y",
                            outfile="C:/best.mlr2b.dat")
```

The parameters **name.x** and **name.y** represent the names of the dependent and independent variables. The parameter **outfile** is the output file. The function writes the list of regression models (used later in calling function lm()) to the output file. It is important to point out that the names specified by parameters name.x and name.y must be the same names of the vectors that supply function read.best.mlr2b() with data.

The Source Code

Here is the code for the function write.best.mlr2b():

```
write.best.mlr2b = function(name.x="x", name.y="y",
                            outfile="C:/best.mlr2b.dat")
{
  # setup the models for linearized regression.

  # write header
  cat("", sep="", file=outfile, append=FALSE)
  # write entries
  for (iy in -4:4) {
    for (ix1 in -4:4) {
      for (ix2 in ix1:4) {
        if (ix1 != ix2) {
          for (ix3 in ix2:4) {
            if (ix3 != ix2) {
              cat(say.fy(iy, name.y), "~", say.fx(ix1, name.x),
                 "+", say.fx(ix2, name.x),
                 "+", say.fx(ix3, name.x),
                 "\n", file=outfile, append=TRUE, sep="")
            }
          }
        }
      }
    }
  }
  cat("\n", file=outfile, append=TRUE)

  return (TRUE)
}
```

The Declaration of Function read.best.mlr2b()

The declaration for function read.best.mlr2b():

```
read.best.mlr2b = function(x, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr2b.dat",
                   outfile="C:/best.mlr2b.txt")
```

The parameters x and y are the data vectors. The arguments for these parameters must match the names specified in calling function write.best.mlr2b(). The parameters **quiet** is the quiet mode flag. The parameter **show.best** tells the function how many best regression models to show. The parameter **infile** specified the model input filename. The parameter **outfile** is the output filename.

The Source Code

Here is the source code for function read.best.mlr2b():

```
read.best.mlr2b = function(x, y, quiet=FALSE,
                   show.best=-1, infile="C:/best.mlr2b.dat",
outfile="C:/best.mlr2b.txt")
{
  # x and y are data vectors.
  # quiet is flag--when TRUE function generates no output
  # show.best specifies the number of best results to show. Default shows all
results
  # infile is the name of the file containing the transformations, shifting,
  #   and scaling data
  # outfile is the output filename

  if (infile=="" | !file.exists(infile)) {
    cat("Input file does not exist. Function aborted.\n")
    return (FALSE)
  }
  # read the formulas from the file
  formula.vect = scan(file=infile, what=character(0))
  max.models=length(formula.vect)
  max.res=6
  # initialize variables used for results
  mat.res = matrix(rep(0,max.res*max.models), nrow=max.models, ncol=max.res)
  formula.arr=rep("", max.models)
  # initialize number of models found
  n.models = 0
  # iterate over the number of elements in vector index.x
  for (i in 1:max.models) {
    mlr = 1 # assign a dummy value
    formula = as.character(formula.vect[i])
    # perform linearized regression
    mlr = lm(formula)
    # call to function lm() succeeded?
    if (class(mlr) == "lm") {
      # increment number of models
      n.models = n.models + 1
      # get the summary
      slr = summary(mlr)
      # build matrix of results
      mat.res[n.models,1] = slr$fstatistic[1]
      mat.res[n.models,2] = slr$r.squared
      for (i in 1:4)
        mat.res[n.models,i+2] = slr$coefficients[i]
      formula.arr[n.models] = formula
    }
  }

  # sort the results
  sort.order = order(mat.res[,1], decreasing=TRUE)
  for (i in 1:max.res)
     mat.res[,i] = mat.res[sort.order,i]
  formula.arr = formula.arr[sort.order]

  # store the results in a list
  list.res = list(mat=mat.res, form=formula.arr, nobs=length(y))

  # display the results?
  if(!quiet) show.results(list.res, show.best, outfile)

  return (list.res)
}
```

The function performs the following tasks:

1. Exits with a warning message if the parameter outfile is an empty string or refers to a nonexistent file.
2. Reads the array of models and stores the array in vector formula.vect. This task uses function scan() to read the formulas from the input file.
3. Calculates the number of models and stores it in variable max.model.
4. Assigns the number of results to variable max.res.
5. Initializes the matrix mat.res and vector formula.arr that store the results.
6. Initializes the number of models.
7. Starts a loop to iterate over all the available models. The loop performs the following subtasks:
   7.1. Assigns a dummy scalar to variable mlr.
   7.2. Stores the current formula in the variable formula.
   7.3. Performs the linearized regression for the current model. This task calls function lm(formula) and stores the result in variable mlr.

7.4. Determines if the class of variable mlr is "lm". If this condition is true, then the last task was successful. The function increments the model counter (variable n.models) and then obtains the summary of the regression variables mlr and stores the F statistics, coefficient of determination, the regression intercept, and the regression slopes in the matrix mat.res. The function also stores the current formula in vector formula.arr.

8. Sorts the data matrix mat.res and vector formula.arr, using the first column of matrix mat.res as the sort key values.
9. Stores the matrix mat.res, the vector formula.arr, and the number of observations in the list list.res.
10. Calls function show.results() to display the results if the value in parameter quiet is FALSE.
11. Returns the list list.res.

A Sample Run

Save the write.best.mlr2b() and read.best.mlr2b() functions to a script file and then load that file using the source() function. Invoke the function write.best.mlr2b() to write the regression models to file C:\best.mlr2b.dat by executing the following command:

```
> write.best.mlr2b()
```

Open the file C:\best.mlr2b.dat with a text editor. The first few lines should look like the following:

```
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)+I(1/x^2)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)+I(1/x)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)+I(log(x))
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)+x
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)+I(x^2)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)+I(x^3)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^3)+I(sqrt(x))
1/sqrt(y)~I(1/sqrt(x))+I(1/x^2)+I(1/x)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^2)+I(log(x))
1/sqrt(y)~I(1/sqrt(x))+I(1/x^2)+x
1/sqrt(y)~I(1/sqrt(x))+I(1/x^2)+I(x^2)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^2)+I(x^3)
1/sqrt(y)~I(1/sqrt(x))+I(1/x^2)+I(sqrt(x))
1/sqrt(y)~I(1/sqrt(x))+I(1/x)+I(log(x))
1/sqrt(y)~I(1/sqrt(x))+I(1/x)+x
```

Type in the following commands to trigger the search for the best model using the models stored in file C:\best.mlr2b.dat:

```
> x=runif(20,1,10)
> y=10+2*x^2-4/x-5*log(x)
> mlr2b.lst = read.best.mlr2b(x,y,show.best=10)
Read 756 items
Best 10 models
Number of observations = 20
```

```
F = 3.574931e+32, R^2 = 1, Model is y~I(1/x)+I(log(x))+I(x^2), Intercept =
10, Slope1 = -4, Slope2 = -5, and Slope3 = 2
F = 1379518187, R^2 = 1, Model is y~I(1/sqrt(x))+I(log(x))+I(x^2), Intercept
= 16.15292, Slope1 = -10.01027, Slope2 = -6.572589, and Slope3 = 2.002408
F = 370119276, R^2 = 1, Model is y~I(1/x^2)+I(log(x))+I(x^2), Intercept =
8.120245, Slope1 = -2.528048, Slope2 = -4.227334, and Slope3 = 1.997126
F = 215145200, R^2 = 1, Model is y~I(x^2)+I(x^3)+I(sqrt(x)), Intercept =
10.86291, Slope1 = 2.020463, Slope2 = -0.0004533956, and Slope3 = -4.540419
F = 182362886, R^2 = 1, Model is y~x+I(x^2)+I(x^3), Intercept = 8.055372,
Slope1 = -2.074806, Slope2 = 2.169127, and Slope3 = -0.006176907
F = 172616379, R^2 = 1, Model is y~x+I(x^2)+I(sqrt(x)), Intercept = 11.00627,
Slope1 = 0.1281340, Slope2 = 2.009855, and Slope3 = -4.796334
F = 155926296, R^2 = 1, Model is y~I(log(x))+I(x^2)+I(sqrt(x)), Intercept =
10.48292, Slope1 = -0.2221863, Slope2 = 2.012420, and Slope3 = -4.143856
F = 149296675, R^2 = 1, Model is y~I(1/sqrt(x))+I(x^2)+I(sqrt(x)), Intercept
= 10.33505, Slope1 = 0.3102624, Slope2 = 2.012910, and Slope3 = -4.304906
F = 143827519, R^2 = 1, Model is y~I(1/x)+I(x^2)+I(sqrt(x)), Intercept =
10.55570, Slope1 = 0.1437788, Slope2 = 2.013223, and Slope3 = -4.357676
F = 137847112, R^2 = 1, Model is y~I(1/sqrt(x))+I(1/x)+I(x^2), Intercept = -
9.512804, Slope1 = 31.66601, Slope2 = -16.59026, and Slope3 = 1.992203
```

The first two commands create the data vectors. The third command invoke function read.best.mlr2b(). The function locates the best model based on the actual data as defined in the second command.

Best-Model Test Cases

This section is an optional bonus and marks the end of the tutorial part of this document. Here I apply the functions best.lr(), best.mlr1(), best.mlr1b(), best.mlr2(), and best.ml2b() to determine the best models that can approximate the inverse statistical distributions for the Normal, Student-t, Chi-square, and F distributions. Please keep in mind that the results are not the fruit of a thorough search that involves experimenting with a wide variety of empirical relations. The result that you see are what the best of functions best.lr(), best.mlr1(), best.mlr1b(), best.mlr2(), and best.ml2b() can produce. My experience in finding new empirical approximations to popular inverse distributions shows that for these approximations to be taken seriously, their curves have to yield R^2 values with several nines after the decimal places! Otherwise, these approximations may fail when a calculated statistic comes very close to an inverse distribution value--the error from the approximation may throw off a researcher in accepting or rejecting a hypothesis, when the reverse conclusion is true!

Before we start make sure that the functions best.lr(), best.mlr1(),best.mlr1b(), best.mlr2(), and best.ml2b(), along with their helper functions are already loaded in the workspace. You may want to copy all of these functions and store them in a single script file. It becomes easier to load just one script file before you follow the instructions in this section. It is also easier to append to such a script file the test commands that you will see in the next sections.

The Best Model to Approximate the Inverse Normal Distribution

Tested Models

The first test tackles the inverse Normal distribution. The test covers the confidence level values in the range of 0.8 to 0.99, in increments of 0.01. The categories of models tested are:

$$f_0(\text{Qinv}) = A + B\, f_1(p) \qquad (1)$$

$$f_0(\text{Qinv}) = A + B\, f_1(p) + C\, f_2(p) \qquad (2)$$

$$f_0(\text{Qinv}) = A + B\, f_1(p) + C\, f_2(p) + D\, f_3(p) \qquad (3)$$

Where p is the confidence level.

Test Code

Execute the following commands to initialize the data and perform the best-model selection. I suggest that you type in the commands in a script window, select these commands, and then execute them by pressing the CTRL+R keys:

```
options(digits=10)
p = seq(0.8, 0.99, 0.01)
y = qnorm(p)
dummy = best.lr(p, y, name.x="p", name.y="y", show.best=10)
dummy = best.mlr1b(p, y, name.x="p", name.y="y", show.best=10)
dummy = best.mlr2b(p, y, name.x="p", name.y="y", show.best=10)
```

The Results

Here is the output generated by the above commands:

```
(output from function best.lr())
Best 10 models
Number of observations = 20
F = 10314.54378, R^2 = 0.9982579314, Model is 1/y~I(log(p)), Intercept =
0.423373524, and Slope1 = -3.378677125
F = 9989.276755, R^2 = 0.9982013089, Model is 1/y~I(sqrt(p)), Intercept =
7.571500237, and Slope1 = -7.155908388
F = 8637.8479, R^2 = 0.997920481, Model is 1/y~I(1/sqrt(p)), Intercept = -
5.943308725, and Slope1 = 6.37442545
F = 7988.721481, R^2 = 0.9977518888, Model is 1/y~p, Intercept = 4.192868613,
and Slope1 = -3.785088904
F = 7948.597025, R^2 = 0.997740566, Model is 1/y^2~I(1/p^3), Intercept = -
1.140407526, and Slope1 = 1.284048809
F = 6548.893259, R^2 = 0.9972589778, Model is 1/sqrt(y)~I(p^3), Intercept =
1.533961682, and Slope1 = -0.8889127028
F = 6385.626878, R^2 = 0.997189093, Model is 1/y~I(1/p), Intercept = -
2.564686528, and Slope1 = 3.00350002
F = 5902.203954, R^2 = 0.9969595642, Model is 1/sqrt(y)~I(p^2), Intercept =
1.852050529, and Slope1 = -1.197615078
```

```
F = 4155.148934, R^2 = 0.9956867104, Model is 1/y~I(p^2), Intercept =
2.503611584, and Slope1 = -2.11151556
F = 3673.553842, R^2 = 0.9951240045, Model is 1/sqrt(y)~p, Intercept =
2.80641054, and Slope1 = -2.14263877

(output from function best.mlr1b())
Best 10 models
Number of observations = 20
F = 12651.48548, R^2 = 0.9993285932, Model is 1/y^2~I(1/p^3)+I(1/p^2),
Intercept = 0.1393372974, Slope1 = 3.069456956, and Slope2 = -3.032364921
F = 11976.44002, R^2 = 0.9992907766, Model is 1/y^2~I(1/p^3)+I(1/p),
Intercept = 1.388431219, Slope1 = 2.163838563, and Slope2 = -3.375396666
F = 11659.94897, R^2 = 0.9992715399, Model is 1/y^2~I(1/sqrt(p))+I(1/p^3),
Intercept = 3.886505106, Slope1 = -5.692130262, and Slope2 = 1.982714881
F = 11356.74326, R^2 = 0.9992521058, Model is 1/y^2~I(1/p^3)+I(log(p)),
Intercept = -1.684664896, Slope1 = 1.861967469, and Slope2 = 2.498040813
F = 11066.28667, R^2 = 0.9992324909, Model is 1/y^2~I(1/p^3)+I(sqrt(p)),
Intercept = -6.105599489, Slope1 = 1.775721878, and Slope2 = 4.507389099
F = 10788.05700, R^2 = 0.999212712, Model is 1/y^2~I(1/p^3)+p, Intercept = -
3.607533132, Slope1 = 1.711040880, and Slope2 = 2.074208556
F = 10543.56227, R^2 = 0.9991944703, Model is 1/y^2~I(1/p^2)+I(1/p),
Intercept = 4.431463649, Slope1 = 7.29130827, and Slope2 = -11.54429577
F = 10266.26461, R^2 = 0.9991727304, Model is 1/y^2~I(1/p^3)+I(p^2),
Intercept = -2.358469297, Slope1 = 1.620498177, and Slope2 = 0.9160841338
F = 10098.17828, R^2 = 0.999158972, Model is 1/y^2~I(1/sqrt(p))+I(1/p^2),
Intercept = 10.86944406, Slope1 = -16.26115167, and Slope2 = 5.570690454
F = 9787.496437, R^2 = 0.9991322986, Model is 1/y^2~I(1/p^3)+I(p^3),
Intercept = -1.942096031, Slope1 = 1.560152400, and Slope2 = 0.5604351551

(output from function best.mlr2b())
Best 10 models
Number of observations =
F = 847199.089, R^2 = 0.9999937048, Model is 1/y^3~I(1/p^3)+I(1/p^2)+I(1/p),
Intercept = -63.8411425, Slope1 = 58.53066569, Slope2 = -177.5904702, and
Slope3 = 182.9468583
F = 751832.5366, R^2 = 0.9999929063, Model is
1/y^3~I(1/sqrt(p))+I(1/p^3)+I(1/p^2), Intercept = -132.3309625, Slope1 =
206.8528267, Slope2 = 48.91298144, and Slope3 = -123.3893471
F = 668769.4216, R^2 = 0.9999920252, Model is
1/y^3~I(1/p^3)+I(1/p^2)+I(log(p)), Intercept = 53.83315534, Slope1 =
42.50151169, Slope2 = -96.28958316, and Slope3 = -68.49046874
F = 596702.0452, R^2 = 0.999991062, Model is
1/y^3~I(1/p^3)+I(1/p^2)+I(sqrt(p)), Intercept = 141.6322162, Slope1 =
37.92216972, Slope2 = -80.03037219, and Slope3 = -99.47934654
F = 576493.0223, R^2 = 0.9999907487, Model is
1/y^3~I(1/sqrt(p))+I(1/p^3)+I(1/p), Intercept = -288.1929139, Slope1 =
677.6287815, Slope2 = 27.01431287, and Slope3 = -416.4056129
F = 534253.7517, R^2 = 0.9999900173, Model is 1/y^3~I(1/p^3)+I(1/p^2)+p,
Intercept = 73.14240232, Slope1 = 34.48791207, Slope2 = -69.19144521, and
Slope3 = -38.39462045
F = 511082.2623, R^2 = 0.9999895647, Model is
1/y^3~I(1/p^3)+I(1/p)+I(log(p)), Intercept = 193.1564557, Slope1 =
23.51283238, Slope2 = -216.6251728, and Slope3 = -149.5705827
```

```
F = 455146.3132, R^2 = 0.9999882823, Model is
1/y^3~I(1/p^3)+I(1/p)+I(sqrt(p)), Intercept = 310.0921794, Slope1 =
21.01192408, Slope2 = -150.0329548, and Slope3 = -181.0274877
F = 455108.9046, R^2 = 0.9999882813, Model is 1/y^2~p+I(p^2)+I(p^3),
Intercept = 92.4303304, Slope1 = -280.4860626, Slope2 = 289.1706894, and
Slope3 = -100.9769778
F = 450365.0935, R^2 = 0.9999881579, Model is
1/y^3~I(1/sqrt(p))+I(1/p^3)+I(log(p)), Intercept = 715.0530223, Slope1 = -
734.7247027, Slope2 = 19.71530782, and Slope3 = -311.735117
```

The function best.lr() returns the following best model 1 (with F = 10314.54378 and R^2 = 0.9982579314):

Qinv = 1/[0.423373524 – 3.378677125 ln(p)]

The function best.mlr1b() returns the following best model 2 (with F = 12651.48548 and R^2 = 0.9993285932):

Qinv = 1/√[0.1393372974 + 3.069456956 / p^3 – 3.032364921 / p^2]

The function best.mlr2b() returns the following best model 3 (with F = 847199.089 and R^2 = 0.9999937048):

Qinv = 1/[–63.8411425 + 58.53066569 / p^3 – 177.5904702 / p^2 + 182.9468583 / p]^(1/3)

I expect that the above models, and especially the last one, to give a good overall approximation for the inverse Normal distribution.

The Best Model to Approximate the Inverse Student-t Distribution

Tested Models

The second test handles the inverse Student-t distribution. The test covers the confidence levels of 0.85, 0.90, 0.95, and 0.99, as well as the degrees of freedom in the range of 5 to 50, in increments of 1. The categories of models tested are:

$$f_0(Tinv) = A + B f_1(p) + C f_2(df) \quad (4)$$

$$f_0(Tinv) = A + B f_1(p) + C f_2(df) + D f_3(p / df) \quad (5)$$

$$f_0(Tinv) = A + B f_1(p) + C f_2(df) + D f_3(df / p) \quad (6)$$

Where p is the confidence level.

Test Code

Execute the following commands to initialize the data and perform the best-model selection. I suggest that you type in the commands in a script window, select these commands, and then execute them by pressing the CTRL+R keys:

```
options(digits=10)
p = c(0.85, 0.90, 0.95, 0.99)
num.p = length(p)
df1 = 5
df2 = 50
num.df = df2 - df1 + 1
p.vect = c()
df.vect = c()
for (i in 1:num.p) {
  p.vect = c(p.vect, rep(p[i], num.df))
  df.vect = c(df.vect, df1:df2)
}
y = abs(qt(p.vect, df.vect))
x = p.vect / df.vect
z = p.vect * df.vect
dummy = best.mlr1(p.vect, df.vect, y, name.x1="p.vect", name.x2="df.vect",
name.y="y", show.best=10)
dummy = best.mlr2(p.vect, df.vect, x, y, name.x1="p.vect", name.x2="df.vect",
name.x3="x", name.y="y", show.best=10)
dummy = best.mlr2(p.vect, df.vect, z, y, name.x1="p.vect", name.x2="df.vect",
name.x3="z", name.y="y", show.best=10)
```

The Results

Here is the output generated by the above commands:

```
(output from function best.mlr1())
Best 10 models
Number of observations = 184
F = 85332.72615, R^2 = 0.9989405692, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect), Intercept = -1.041795169, Slope1 =
1.203351118, and Slope2 = -0.696798663
F = 72884.06763, R^2 = 0.998759842, Model is
1/y^2~I(1/p.vect^3)+I(1/sqrt(df.vect)), Intercept = -0.9975400673, Slope1 =
1.203351118, and Slope2 = -0.3731081525
F = 72444.75289, R^2 = 0.9987523308, Model is
1/y^2~I(1/p.vect^2)+I(1/df.vect), Intercept = -1.827769738, Slope1 =
1.979730501, and Slope2 = -0.696798663
F = 64215.57831, R^2 = 0.998592668, Model is 1/y~I(p.vect^3)+I(1/df.vect),
Intercept = 1.892130896, Slope1 = -1.509551568, and Slope2 = -0.5601569163
F = 63267.26108, R^2 = 0.9985716036, Model is
1/y^2~I(1/p.vect^2)+I(1/sqrt(df.vect)), Intercept = -1.783514636, Slope1 =
1.979730501, and Slope2 = -0.3731081525
F = 53129.97481, R^2 = 0.9982995266, Model is
1/y~I(p.vect^3)+I(1/sqrt(df.vect)), Intercept = 1.927484175, Slope1 = -
1.509551568, and Slope2 = -0.2989108208
F = 47451.77014, R^2 = 0.9980964308, Model is 1/y~I(p.vect^2)+I(1/df.vect),
Intercept = 2.477253543, Slope1 = -2.086900878, and Slope2 = -0.5601569163
F = 46539.07778, R^2 = 0.9980591718, Model is
1/y^2~I(1/p.vect^3)+I(log(df.vect)), Intercept = -1.217668567, Slope1 =
1.203351118, and Slope2 = 0.04409368605
F = 44915.3537, R^2 = 0.9979891505, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect^2), Intercept = -1.063038754, Slope1 =
1.203351118, and Slope2 = -3.504196932
```

```
F = 42416.3909, R^2 = 0.9978709334, Model is
1/y^2~I(1/p.vect^2)+I(log(df.vect)), Intercept = -2.003643136, Slope1 =
1.979730501, and Slope2 = 0.04409368605

(output from function best.mlr2(p.vect, df.vect, x, y, …))
Best 10 models
Number of observations = 184
F = 105928.1895, R^2 = 0.9994338992, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect)+x, Intercept = -1.086868115, Slope1 =
1.238042388, Slope2 = -3.318147302, and Slope3 = 2.84157034
F = 91277.1574, R^2 = 0.9993430932, Model is
1/y^2~I(1/p.vect^2)+I(1/df.vect)+x, Intercept = -1.902228151, Slope1 =
2.042472105, Slope2 = -3.573390316, and Slope3 = 3.118256535
F = 79336.58196, R^2 = 0.9992443, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect^2)+I(x^3), Intercept = -1.069613422, Slope1 =
1.213593431, Slope2 = -8.190212887, and Slope3 = 33.08381466
F = 73792.4377, R^2 = 0.9991875691, Model is
1/y^2~I(1/p.vect^3)+I(1/sqrt(df.vect))+I(sqrt(x)), Intercept = -1.089508881,
Slope1 = 1.274136706, Slope2 = -2.956464195, and Slope3 = 2.690782426
F = 73400.62317, R^2 = 0.9991832359, Model is
1/y^2~I(1/p.vect^2)+I(1/sqrt(df.vect))+I(sqrt(x)), Intercept = -1.95096451,
Slope1 = 2.120830386, Slope2 = -3.498385178, and Slope3 = 3.255238674
F = 66577.62819, R^2 = 0.9990996078, Model is
1/y^2~I(1/p.vect^2)+I(1/df.vect^2)+I(x^3), Intercept = -1.862563604, Slope1 =
1.996920230, Slope2 = -8.271526835, and Slope3 = 33.65790053
F = 63940.96147, R^2 = 0.999062514, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect)+I(x^2), Intercept = -1.038669979, Slope1 =
1.206258031, Slope2 = -0.9389629962, and Slope3 = 1.555004637
F = 63630.72568, R^2 = 0.9990579476, Model is
1/y^2~I(1/p.vect^3)+df.vect+I(1/sqrt(x)), Intercept = -1.244541389, Slope1 =
1.150244234, Slope2 = -0.00662909918, and Slope3 = 0.07909737872
F = 62181.29152, R^2 = 0.9990360097, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect)+I(x^3), Intercept = -1.039507944, Slope1 =
1.204882586, Slope2 = -0.817514405, and Slope3 = 4.946814764
F = 57899.93964, R^2 = 0.9989648022, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect)+I(1/x^3), Intercept = -1.043159707, Slope1 =
1.202109312, Slope2 = -0.6708168474, and Slope3 = 3.514130185e-08

(output from function best.mlr2(p.vect, df.vect, z, y, …))
Best 10 models
Number of observations = 184
F = 110630.7760, R^2 = 0.9994579494, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect)+I(1/z), Intercept = -1.088313020, Slope1 =
1.239154488, Slope2 = 1.976017956, and Slope3 = -2.457584847
F = 87048.14812, R^2 = 0.999311201, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect^3)+I(1/z^2), Intercept = -1.080102503, Slope1
= 1.221541185, Slope2 = 25.86184805, and Slope3 = -6.833267405
F = 86656.15267, R^2 = 0.9993080874, Model is
1/y^2~I(1/p.vect^2)+I(1/df.vect)+I(1/z), Intercept = -1.900365256, Slope1 =
2.040902356, Slope2 = 2.076724655, and Slope3 = -2.550182018
F = 76673.8505, R^2 = 0.9992180765, Model is
1/y^2~I(1/p.vect^3)+I(1/sqrt(df.vect))+I(1/sqrt(z)), Intercept = -
1.093994835, Slope1 = 1.277589408, Slope2 = 2.319889493, and Slope3 = -
2.583359227
```

```
F = 76228.98167, R^2 = 0.9992135168, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect^2)+I(1/z), Intercept = -1.052129740, Slope1 =
1.214672583, Slope2 = 0.7801479834, and Slope3 = -0.7771185233
F = 74748.73533, R^2 = 0.9991979546, Model is
1/y^2~I(1/p.vect^3)+I(1/df.vect^3)+I(1/z), Intercept = -1.052596293, Slope1 =
1.213589772, Slope2 = 1.987987787, and Slope3 = -0.7027931
F = 73603.08311, R^2 = 0.9991854807, Model is
1/y^2~I(1/p.vect^3)+I(1/sqrt(df.vect))+I(1/z), Intercept = -1.063795812,
Slope1 = 1.214387088, Slope2 = 0.06615519616, and Slope3 = -0.7575218136
F = 72883.4298, R^2 = 0.9991774448, Model is
1/y^2~I(1/p.vect^3)+I(df.vect^2)+I(1/z), Intercept = -1.054534031, Slope1 =
1.212666156, Slope2 = 6.030627484e-07, and Slope3 = -0.6393950267
F = 72883.0648, R^2 = 0.9991774406, Model is
1/y^2~I(1/p.vect^3)+I(df.vect^3)+I(1/z), Intercept = -1.054298753, Slope1 =
1.212689327, Slope2 = 1.097472589e-08, and Slope3 = -0.6409855167
F = 72843.99299, R^2 = 0.9991769998, Model is
1/y^2~I(1/p.vect^3)+df.vect+I(1/z), Intercept = -1.054984221, Slope1 =
1.212645362, Slope2 = 3.483767528e-05, and Slope3 = -0.6379677173
```

The function best.mlr1() yields the following best model 4 (with F = 85332.72615 and R^2 = 0.9989405692):

$$Tinv = 1/\sqrt{[-1.041795169 + 1.203351118 / p^3 - 0.696798663 / df]}$$

The first call to function best.mlr2() yields the following best model 5 (with F = 105928.1895 and R^2 = 0.9994338992):

$$Tinv = 1/\sqrt{[-1.086868115 + 1.238042388 / p^3 - 3.318147302 / df + 2.84157034 (p / df)]}$$

The second call to function best.mlr2() yields the following best model 6 (with F = 110630.7760 and R^2 = 0.9994579494):

$$Tinv = 1/\sqrt{[-1.088313020 + 1.239154488 / p^3 + 1.976017956 / df - 2.457584847 / (p * df)]}$$

The last model is more promising than the other two models.

The Best Model to Approximate the Inverse Chi-square Distribution

Tested Models

The third test deals with the inverse Chi-square distribution. The test covers the confidence levels of 0.85, 0.90, 0.95, and 0.99, as well as the degrees of freedom in the range of 5 to 50, in increments of 1. The categories of models tested are:

$$f_0(ChiSqrInv) = A + B f_1(p) + C f_2(df) \qquad (7)$$

$$f_0(\text{ChiSqrInv}) = A + B\, f_1(p) + C\, f_2(df) + D\, f_3(p / df) \tag{8}$$

$$f_0(\text{ChiSqrInv}) = A + B\, f_1(p) + C\, f_2(df) + D\, f_3(df / p) \tag{9}$$

Where p is the confidence level.

Test Code

Execute the following commands to initialize the data and perform the best-model selection. I suggest that you type in the commands in a script window, select these commands, and then execute them by pressing the CTRL+R keys:

```
options(digits=10)
p = c(0.85, 0.90, 0.95, 0.99)
num.p = length(p)
df1 = 5
df2 = 50
num.df = df2 - df1 + 1
p.vect = c()
df.vect = c()
for (i in 1:num.p) {
  p.vect = c(p.vect, rep(p[i], num.df))
  df.vect = c(df.vect, df1:df2)
}
y = qchisq(p.vect, df.vect)
x = p.vect / df.vect
z = p.vect * df.vect
dummy = best.mlr1(p.vect, df.vect, y, name.x1="p.vect", name.x2="df.vect",
name.y="y", show.best=10)
dummy = best.mlr2(p.vect, df.vect, x, y, name.x1="p.vect", name.x2="df.vect",
name.x3="x", name.y="y", show.best=10)
dummy = best.mlr2(p.vect, df.vect, z, y, name.x1="p.vect", name.x2="df.vect",
name.x3="z", name.y="y", show.best=10)
```

The Results

Here is the output generated by the above commands:

```
(output from function best.mlr1())
Best 10 models
Number of observations = 184
F = 22278.00566, R^2 = 0.995954133, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect)), Intercept = -1.084352387, Slope1 =
2.675016518, and Slope2 = 1.008922006
F = 19044.87883, R^2 = 0.9952705405, Model is
sqrt(y)~I(p.vect^2)+I(sqrt(df.vect)), Intercept = -2.104816121, Slope1 =
3.678895095, and Slope2 = 1.008922006
F = 16468.07489, R^2 = 0.9945345538, Model is
sqrt(y)~p.vect+I(sqrt(df.vect)), Intercept = -5.170229354, Slope1 =
6.727763569, and Slope2 = 1.008922006
F = 15373.95526, R^2 = 0.9941478702, Model is
sqrt(y)~I(sqrt(p.vect))+I(sqrt(df.vect)), Intercept = -11.30419306, Slope1 =
12.85348621, and Slope2 = 1.008922006
```

```
F = 14387.91256, R^2 = 0.9937493147, Model is
sqrt(y)~I(log(p.vect))+I(sqrt(df.vect)), Intercept = 1.541081676, Slope1 =
6.135039429, and Slope2 = 1.008922006
F = 13496.81216, R^2 = 0.9933393743, Model is
sqrt(y)~I(1/sqrt(p.vect))+I(sqrt(df.vect)), Intercept = 13.23815799, Slope1 =
-11.70525162, and Slope2 = 1.008922006
F = 12689.40385, R^2 = 0.9929185696, Model is
sqrt(y)~I(1/p.vect)+I(sqrt(df.vect)), Intercept = 7.104234865, Slope1 = -
5.579461091, and Slope2 = 1.008922006
F = 11288.28598, R^2 = 0.9920466032, Model is
sqrt(y)~I(1/p.vect^2)+I(sqrt(df.vect)), Intercept = 4.038983577, Slope1 = -
2.530321027, and Slope2 = 1.008922006
F = 10375.14810, R^2 = 0.9913526617, Model is
log(y)~I(p.vect^3)+I(log(df.vect)), Intercept = 0.2849832164, Slope1 =
0.9201003463, and Slope2 = 0.8091213202
F = 10149.01418, R^2 = 0.99116169, Model is y~I(p.vect^3)+df.vect, Intercept
= -20.26493449, Slope1 = 33.10174898, and Slope2 = 1.229571544

(output from function best.mlr2(p.vect, df.vect, x, y, …))
Best 10 models
Number of observations = 184
F = 20332.45745, R^2 = 0.9970577357, Model is
log(y)~I(p.vect^3)+df.vect+I(1/sqrt(x)), Intercept = -0.334941322, Slope1 =
1.680748411, Slope2 = -0.03570613198, and Slope3 = 0.6735229184
F = 16496.46882, R^2 = 0.9963760388, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(1/sqrt(x)), Intercept = -1.701128048,
Slope1 = 3.453086224, Slope2 = 0.2907337611, and Slope3 = 0.6889490728
F = 16352.73329, R^2 = 0.9963443018, Model is
sqrt(y)~I(p.vect^3)+I(1/sqrt(df.vect))+I(log(x)), Intercept = -14.89368444,
Slope1 = 4.49896018, Slope2 = 16.79714232, and Slope3 = -4.267714299
F = 16284.89907, R^2 = 0.99632913, Model is
sqrt(y)~I(p.vect^3)+I(log(df.vect))+I(1/sqrt(x)), Intercept = -1.988981590,
Slope1 = 3.73333816, Slope2 = 0.0721702623, and Slope3 = 0.9371007617
F = 16255.71438, R^2 = 0.9963225637, Model is
sqrt(y)~I(p.vect^3)+I(1/sqrt(df.vect))+I(1/sqrt(x)), Intercept = -
1.819849140, Slope1 = 3.753890008, Slope2 = -0.2469050707, and Slope3 =
0.9552985871
F = 16246.39939, R^2 = 0.996320463, Model is
sqrt(y)~I(p.vect^3)+I(1/df.vect)+I(1/sqrt(x)), Intercept = -1.888917115,
Slope1 = 3.75952112, Slope2 = -0.2896816843, and Slope3 = 0.9602847077
F = 16236.79507, R^2 = 0.9963182945, Model is
sqrt(y)~I(p.vect^3)+I(1/df.vect^2)+I(1/sqrt(x)), Intercept = -1.921527929,
Slope1 = 3.763465756, Slope2 = -0.9504536465, and Slope3 = 0.9637775223
F = 16229.78538, R^2 = 0.9963167102, Model is
sqrt(y)~I(p.vect^3)+I(1/df.vect^3)+I(1/sqrt(x)), Intercept = -1.931543885,
Slope1 = 3.764931655, Slope2 = -4.106569837, and Slope3 = 0.965075516
F = 16206.82735, R^2 = 0.9963115118, Model is
sqrt(y)~I(p.vect^3)+I(df.vect^3)+I(1/sqrt(x)), Intercept = -1.967713520,
Slope1 = 3.772289196, Slope2 = -1.756225674e-07, and Slope3 = 0.9715903194
F = 16206.33779, R^2 = 0.9963114008, Model is
sqrt(y)~I(p.vect^3)+I(df.vect^2)+I(1/sqrt(x)), Intercept = -1.974489229,
Slope1 = 3.774429544, Slope2 = -1.193482499e-05, and Slope3 = 0.9734855104
Best 10 models

(output from function best.mlr2(p.vect, df.vect, z, y, …))
```

```
Number of observations = 184
F = 15816.86647, R^2 = 0.9962209168, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(sqrt(z)), Intercept = -1.584809496,
Slope1 = 3.306349012, Slope2 = 1.593508245, and Slope3 = -0.6088956962
F = 14933.64285, R^2 = 0.995998304, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+z, Intercept = -1.234914745, Slope1 =
2.732766426, Slope2 = 1.056413284, and Slope3 = -0.005347349765
F = 14843.54254, R^2 = 0.9959741115, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(z^2), Intercept = -1.139702411, Slope1
= 2.69402451, Slope2 = 1.021312115, and Slope3 = -2.819390712e-05
F = 14819.77767, R^2 = 0.9959676817, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(z^3), Intercept = -1.118660273, Slope1
= 2.68775192, Slope2 = 1.01569845, and Slope3 = -3.603933998e-07
F = 14805.92074, R^2 = 0.995963923, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(1/z^2), Intercept = -1.059460735,
Slope1 = 2.671745831, Slope2 = 1.005270908, and Slope3 = -0.7336253668
F = 14803.59042, R^2 = 0.9959632903, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(1/z^3), Intercept = -1.067620686,
Slope1 = 2.672389416, Slope2 = 1.006432228, and Slope3 = -2.99039817
F = 14800.77773, R^2 = 0.9959625262, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(1/z), Intercept = -1.038128998, Slope1
= 2.669911204, Slope2 = 1.002945232, and Slope3 = -0.2086381287
F = 14790.93968, R^2 = 0.9959598516, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(1/sqrt(z)), Intercept = -1.007159865,
Slope1 = 2.668087539, Slope2 = 1.001155657, and Slope3 = -0.1433452943
F = 14771.86154, R^2 = 0.9959546548, Model is
sqrt(y)~I(p.vect^3)+I(sqrt(df.vect))+I(log(z)), Intercept = -1.090051273,
Slope1 = 2.670197565, Slope2 = 1.003950670, and Slope3 = 0.0112755217
F = 14674.40355, R^2 = 0.9959278976, Model is
log(y)~I(p.vect^3)+I(sqrt(df.vect))+z, Intercept = -0.07455766806, Slope1 =
1.342879144, Slope2 = 0.705789474, and Slope3 = -0.03914718139
```

The function best.mlr1() yields the following best model 7 (with F = 22278.00566 and R^2 = 0.995954133):

$$\text{ChiSqrInv} = [-1.084352387 + 2.675016518\, p^3 + 1.008922006 \sqrt{df}]^2$$

The first call to function best.mlr2() yields the following best model 8 (with F = 20332.45745 and R^2 = 0.9970577357):

$$\text{ChiSqrInv} = \exp[-0.334941322 + 1.680748411\, p^3 - 0.03570613198\, df$$
$$+ 0.6735229184 / \sqrt{(p / df)}]$$

The second call to function best.mlr2() yields the following best model 9 (with F = 15816.86647 and R^2 = 0.9962209168):

$$\text{ChiSqrInv} = [-1.584809496 + 3.306349012\, p^3 + 1.593508245 \sqrt{df}$$
$$- 0.6088956962 \sqrt{(p * df)}]^2$$

The first model is more promising than the other two models.

The Best Model to Approximate the Inverse F Distribution

Tested Models

The last test tackles the inverse F distribution. The test covers the confidence levels of 0.85, 0.90, 0.95, and 0.99, as well as the two degrees of freedom in the range of 5 to 50, in increments of 1. The category of models tested is:

$$f_0(\text{Finv}) = A + B\ f_1(p) + C\ f_2(df_1) + D\ f_3(df_2) \qquad (10)$$

Where α is the confidence level.

Test Code

Execute the following commands to initialize the data and perform the best-model selection. I suggest that you type in the commands in a script window, select these commands, and then execute them by pressing the CTRL+R keys:

```
options(digits=10)
p = c(0.85, 0.90, 0.95, 0.99)
num.p = length(p)
df11 = 5
df21 = 50
df12 = 5
df22 = 50
num.df1 = df21 - df11 + 1
num.df2 = df22 - df12 + 1
p.vect = c()
df1.vect = c()
df2.vect = c()
for (i in 1:num.p) {
    p.vect = c(p.vect, rep(p[i], num.df2 * num.df1))
    for (j in df11:df21) {
      df1.vect = c(df1.vect, rep(j, num.df2))
      df2.vect = c(df2.vect, df12:df22)
  }
}
y = qf(p.vect, df1.vect, df2.vect)
dummy = best.mlr2(p.vect, df1.vect, df2.vect, y, name.x1="p.vect",
name.x2="df1.vect", name.x3="df2.vect", name.y="y", show.best=10)
```

The Results

Here is the output generated by the above commands:

```
Best 10 models
Number of observations = 8464
F = 109495.8443, R^2 = 0.9748922334, Model is
1/y~I(p.vect^3)+I(1/sqrt(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.452746260, Slope1 = -0.7549204893, Slope2 = -0.4726944735, and Slope3 = -
1.124248917
F = 104935.9859, R^2 = 0.9738297601, Model is
1/y~I(p.vect^3)+I(log(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.169084883, Slope1 = -0.7549204893, Slope2 = 0.05737550302, and Slope3 = -
1.124248917
F = 99877.83648, R^2 = 0.9725408042, Model is
1/y~I(p.vect^3)+I(1/df1.vect)+I(1/sqrt(df2.vect)), Intercept = 1.395449242,
Slope1 = -0.7549204893, Slope2 = -0.8593652257, and Slope3 = -1.124248917
F = 94697.83516, R^2 = 0.9710822129, Model is
1/y~I(p.vect^2)+I(1/sqrt(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.742429573, Slope1 = -1.040213869, Slope2 = -0.4726944735, and Slope3 = -
1.124248917
F = 91733.97327, R^2 = 0.9701757641, Model is
1/y^2~I(p.vect^3)+I(1/sqrt(df1.vect))+I(log(df2.vect)), Intercept =
0.5743216997, Slope1 = -0.7403353229, Slope2 = -0.4758895650, and Slope3 =
0.1244083215
F = 91241.89124, R^2 = 0.9700197395, Model is
1/y~I(p.vect^2)+I(log(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.458768196, Slope1 = -1.040213869, Slope2 = 0.05737550302, and Slope3 = -
1.124248917
F = 91221.61667, R^2 = 0.970013276, Model is
1/y^2~I(p.vect^3)+I(log(df1.vect))+I(log(df2.vect)), Intercept =
0.2874519906, Slope1 = -0.7403353229, Slope2 = 0.05817202802, and Slope3 =
0.1244083215
F = 90749.55379, R^2 = 0.9698619916, Model is
1/y~I(p.vect^3)+I(sqrt(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.224827980, Slope1 = -0.7549204893, Slope2 = 0.02478626458, and Slope3 = -
1.124248917
F = 87364.54333, R^2 = 0.9687307836, Model is
1/y~I(p.vect^2)+I(1/df1.vect)+I(1/sqrt(df2.vect)), Intercept = 1.685132555,
Slope1 = -1.040213869, Slope2 = -0.8593652257, and Slope3 = -1.124248917
F = 85559.94591, R^2 = 0.968092309, Model is
1/y^2~I(p.vect^2)+I(1/sqrt(df1.vect))+I(log(df2.vect)), Intercept =
0.8598958186, Slope1 = -1.021859090, Slope2 = -0.4758895650, and Slope3 =
0.1244083215
```

The best model 10 generated by function best.mlr2() is (with F = 109495.8443 and R^2 = 0.9748922334):

$$\text{Finv} = 1/[1.452746260 - 0.7549204893\, p^3 - 0.4726944735 / \sqrt{df_1} - 1.124248917/ \sqrt{df_2}]$$

The above model is not very reliable since the value if R^2 is below 0.99000. In the world of high expectations from this kind of approximations, the above model would yield disappointing results! A good empirical approximation that I was able to build has more terms and more elaborate transformations.

Using the F statistics to asses various models is better than the coefficient of determination, as the F statistic does not bow to the constraining value of 1, as does R^2.

Using the Significance Levels

It may come as a surprise for some readers that replacing the confidence level p with the significance level α (equals (1-p)/2) in the tested models gives generally better results. Here is the test code for the four common distributions:

```
options(digits=10)
p = seq(0.8, 0.99, 0.01)
a = (1-p)/2
y = qnorm(a)
dummy = best.lr(a, y, name.x="a", name.y="y", show.best=10)
dummy = best.mlr1b(a, y, name.x="a", name.y="y", show.best=10)
dummy = best.mlr2b(a, y, name.x="a", name.y="y", show.best=10)

# inverse Student-t
options(digits=10)
p = c(0.85, 0.90, 0.95, 0.99)
a = (1-p)/2
num.a = length(a)
df1 = 5
df2 = 50
num.df = df2 - df1 + 1
a.vect = c()
df.vect = c()
for (i in 1:num.a) {
  a.vect = c(a.vect, rep(a[i], num.df))
  df.vect = c(df.vect, df1:df2)
}
y = qt(a.vect, df.vect)
x = a.vect / df.vect
z = a.vect * df.vect
dummy = best.mlr1(a.vect, df.vect, y, name.x1="a.vect", name.x2="df.vect",
name.y="y", show.best=10)
dummy = best.mlr2(a.vect, df.vect, x, y, name.x1="a.vect", name.x2="df.vect",
name.x3="x", name.y="y", show.best=10)
dummy = best.mlr2(a.vect, df.vect, z, y, name.x1="a.vect", name.x2="df.vect",
name.x3="z", name.y="y", show.best=10)

# inverse Chi-square
options(digits=10)
p = c(0.85, 0.90, 0.95, 0.99)
a = (1-p)/2
num.a = length(a)
df1 = 5
df2 = 50
num.df = df2 - df1 + 1
a.vect = c()
df.vect = c()
for (i in 1:num.a) {
  a.vect = c(a.vect, rep(a[i], num.df))
  df.vect = c(df.vect, df1:df2)
}
y = qchisq(a.vect, df.vect)
x = a.vect / df.vect
```

```
z = a.vect * df.vect
dummy = best.mlr1(a.vect, df.vect, y, name.x1="a.vect", name.x2="df.vect",
name.y="y", show.best=10)
dummy = best.mlr2(a.vect, df.vect, x, y, name.x1="a.vect", name.x2="df.vect",
name.x3="x", name.y="y", show.best=10)
dummy = best.mlr2(a.vect, df.vect, z, y, name.x1="a.vect", name.x2="df.vect",
name.x3="z", name.y="y", show.best=10)

# inverse F
p = c(0.85, 0.90, 0.95, 0.99)
a = (1-p)/2
num.a = length(a)
df11 = 5
df21 = 50
df12 = 5
df22 = 50
num.df1 = df21 - df11 + 1
num.df2 = df22 - df12 + 1
a.vect = c()
df1.vect = c()
df2.vect = c()
for (i in 1:num.a) {
    a.vect = c(a.vect, rep(a[i], num.df2 * num.df1))
    for (j in df11:df21) {
      df1.vect = c(df1.vect, rep(j, num.df2))
      df2.vect = c(df2.vect, df12:df22)
  }
}
y = qf(a.vect, df1.vect, df2.vect)
dummy = best.mlr2(a.vect, df1.vect, df2.vect, y, name.x1="a.vect",
name.x2="df1.vect", name.x3="df2.vect", name.y="y", show.best=10)
```

The above code is very similar to the one I presented in the previous sections. Without listing the output, which looks very similar, I will simply list the best models that use the significance levels:

The Inverse Normal Distribution

The function best.lr() returns the following best model 1 (with F = 23275.46994 and R^2 = 0.9992272512):

$$Qinv^2 = –2.224502594 + 1.654856747 \ln(\alpha)$$

The function best.mlr1b() returns the following best model 2 (with F = 21576151.67 and R^2 = 0.999999606):

$$Qinv = –1.186252673 + 0.2861352962 \ln(\alpha) + 1.781900821 \sqrt{\alpha}$$

The function best.mlr2b() returns the following best model 3 (with F = 1824316363 and R^2 = 0.999999997):

$$Qinv^2 = –3.393022312 – 1.861790587 \ln(\alpha) + 1.763562059 \alpha^2 + 2.310890670 \sqrt{\alpha}$$

The Inverse Student-t Distribution

The function best.mlr1() yields the following best model 4 (with F = 47085.04163 and R^2 = 0.998081633):

$$Tinv = 1/[0.2685566466 – 1.549320225 \sqrt{\alpha} + 0.6019495052 / df]$$

The first call to function best.mlr2() yields the following best model 5 (with F = 112231.5646 and R^2 = 0.9994656767):

$$Tinv^2 = 1/[0.2230986339 + 4.169541863\ \alpha + 0.3786612402 / \sqrt{df} – 0.0005397186582 / \sqrt{(\alpha/df)}]$$

The second call to function best.mlr2() yields the following best model 6 (with F = 64901.34386 and R^2 = 0.9990763738):

$$Tinv = 1/[–0.2776183787 – 1.882129943 \sqrt{\alpha} + 0.5222885533 / \sqrt{df} + 0.02491058733 \ln(\alpha * df)$$

The Inverse Chi-square Distribution

The function best.mlr1() yields the following best model 7 (with F = 240568.7114 and R^2 = 0.9996239496):

$$ChiSqrInv = [–0.3666832465 + 0.2547098825 \ln(\alpha) + 0.992050952 \sqrt{df}]^2$$

The first call to function best.mlr2() yields the following best model 8 (with F = 887044.909 and R^2 = 0.9999323643):

$$ChiSqrInv = [–1.744640735 + 2.478192331 \sqrt{\alpha} + 1.018618948 \sqrt{df} – 0.003717062919 / \sqrt{(p / df)}]^2$$

The second call to function best.mlr2() yields the following best model 9 (with F = 928697.7014 and R^2 = 0.9999353976):

$$ChiSqrInv = [–0.3857586016 + 0.3103911880 \ln(\alpha) + 1.016090808 \sqrt{df} + 0.0647128734 / \sqrt{(p*df)}]^2$$

The Inverse F Distribution

The best model 10 generated by function best.mlr2() is (with F = 180844.1118 and R^2 = 0.9846458844):

$$Finv = 0.6155558551 + 1.024845007 \sqrt{\alpha} – 1.177836477 / \sqrt{df_1} – 0.5065293412 / \sqrt{df_2}$$

The following table compares the two sets of best models for the various inverse distributions. The table lists the F and R^2 values. Granted that the best models we are comparing may or may not be the same. What is important is how well the best model fit the data. The set of the significance level outperformed the one for confidence level in all cases, expect the one for the result of best.mlr1() for the inverse Student-t. The reason might be that the variation in the significance levels values represents a bigger spread than the variations in the confidence levels values.

Inverse Distribution	***Model Number***	***F (confidence level)***	***R^2 (confidence level)***	***F (significance level)***	***R^2 (significance level)***
Normal	1	10314	0.9982579314	23275	0.9992272512
	2	12651	0.9993285932	21576151	0.9999996060
	3	847199	0.9999937048	1824316363	0.9999999970
Student-t	4	85332	0.9989405692	47085	0.9980816330
	5	105928	0.9994338992	112231	0.9994656767
	6	110630	0.9994579494	64901	0.9990763738
Chi-square	7	22278	0.9959541330	240568	0.9996239496
	8	20332	0.9970577357	887044	0.9999323643
	9	15816	0.9962209168	928697	0.9999353976
F	10	109495	0.9748922334	180844	0.9846458844

Massaging the Best-Model Search Output

The text output of the best-model search can be massaged to look more readable. This section presents Excel VBA code which reads what the R functions generated, displays the result in Excel, and also rewrites the information in a more readable form to a text file.

The VBA Code

The following VBA macro Parse and its helper functions can assist in reformatting the output from the R functions:

```
Option Explicit

Const BAD_RESULT = 1E+100

Function GetValueIn(ByRef sStr As String) As Double
  Dim sVar As Variant
  Dim I As Integer

  ' split sStr at spaces
  sVar = Split(sStr, " ")
  GetValueIn = 1.1 * BAD_RESULT ' default function result
  ' Find a numerical value in array sVar
  For I = LBound(sVar) To UBound(sVar)
    ' found numeric value?
    If IsNumeric(sVar(I)) Then
      GetValueIn = CDbl(sVar(I)) ' assign value to function's return value
      Exit Function
    End If
  Next I
End Function
```

```
Function GetValueAfter(ByRef sStr As String, ByVal sFind As String) As Double
  Dim I As Integer, J As Integer, K As Integer
  Dim sVal As String, sStr1 As String

  On Error GoTo HandleErr
  ' make a local copy
  sStr1 = sStr
  ' remove internal spaces
  Do While InStr(sStr1, " ") > 0
    sStr1 = Replace(sStr1, " ", "")
  Loop

  ' find the marker in variable sFind
  I = InStr(sStr1, sFind)
  If I > 0 Then
    ' find location of = sign
    J = InStr(I, sStr1, "=")
    ' find next comma or end of string
    K = InStr(J, sStr1, ",")
    If K = 0 Then K = Len(sStr1)
    sVal = Mid(sStr1, J + 1, K - J - 1)
    GetValueAfter = CDbl(sVal)
  Else
    GetValueAfter = 1.1 * BAD_RESULT
  End If
ExitProc:
  Exit Function

HandleErr:
  MsgBox "Error " & Err.Description & " String is " & sVal, _
     vbOKOnly + vbCritical, "Error"
  End ' Die!!!
'''  GetValueAfter = 1.1E+100
'''  Resume ExiProct
End Function

Function ExSplit(ByRef sStr As String, ByVal sSplitChar As String) As Variant
  Const DUMMY_PLUS = "?"
  Const PLUS = "+"

  Dim I As Integer, nNumOpenPar As Integer
  Dim C As String
  Dim vTerm As Variant

  nNumOpenPar = 0 ' initial state
  ' Scan string for open and close parentheses and plus characters
  For I = 1 To Len(sStr)
    C = Mid(sStr, I, 1) ' get the current character
    If C = "(" Then
      nNumOpenPar = nNumOpenPar + 1
    ElseIf C = ")" Then
      nNumOpenPar = nNumOpenPar - 1
    ElseIf C = PLUS Then
      ' plus sign inside parentheses?
      If nNumOpenPar > 0 Then
```

```
        Mid(sStr, I, 1) = DUMMY_PLUS ' temporarely change to dummy character
      Else
        ' do nothing
      End If
    Else
      ' do nothing
    End If
  Next I

  ' now split the string at the high-level plus characters
  vTerm = Split(sStr, PLUS)

  ' restore the lower-level plus characters
  For I = LBound(vTerm) To UBound(vTerm)
    vTerm(I) = Replace(vTerm(I), DUMMY_PLUS, PLUS)
  Next I

  ExSplit = vTerm

End Function

Function SaySignAndVal(ByVal X As Double) As String
  ' return sign and absolute value as a string
  SaySignAndVal = IIf(X < 0, " - " & CStr(Abs(X)), " + " & CStr(X))
End Function

Function PrintModel(ByRef sStr As String, ByVal Row As Integer, _
                    ByVal Col As Integer) As String
  Dim N As Integer, I As Integer
  Dim sModel As String, sRes As String
  Dim sTerm As String
  Dim X As Double
  Dim vTerm As Variant

  ' extract the model
  sModel = GetModel(sStr)
  ' remove spces in the model
  Do While InStr(sModel, " ") > 0
    sModel = Replace(sModel, " ", "")
  Loop

  ' find the tilde character
  I = InStr(sModel, "~")
  ' get term for Y and append the intercept
  sRes = Left(sModel, I - 1) & " = " & Cells(Row, Col)

  N = 0 ' initialize number of terms
  ' remove Y term up to and including tilde character
  sModel = Mid(sModel, I + 1)
  ' split the rest of the the model into tokens
  vTerm = ExSplit(sModel, "+")

  ' iterate over each term
  For I = LBound(vTerm) To UBound(vTerm)
    N = N + 1 ' increment term counter
    sTerm = vTerm(I) ' get the next term
    ' Is the variable transformed using I()?
    If Left(sTerm, 2) = "I(" Then
      sTerm = Mid(sTerm, 3) ' chop leading 'I('
      sTerm = Left(sTerm, Len(sTerm) - 1) ' chop trailing ')'
    End If

    ' Check for shifting and scaling
    If (InStr(sTerm, "+") + InStr(sTerm, "-") + _
        InStr(sTerm, "*") + InStr(sTerm, "/")) > 0 Then
      sTerm = "(" & sTerm & ")" ' enclose in parentheses
    End If

    ' get the regression coefficient
    X = Cells(Row, Col + N)
    ' append regression coefficient AND term
    If Left(sTerm, 2) = "1/" Then
      sRes = sRes & SaySignAndVal(X) & " / " & Mid(sTerm, 3)
    Else
      sRes = sRes & SaySignAndVal(X) & " * " & sTerm
    End If
  Next I

  ' return function result
  PrintModel = sRes

End Function

Function GetModel(ByRef sStr As String) As String
  Const MARKER = "Model is "
  Dim I As Integer, J As Integer, K As Integer

  GetModel = ""
  ' find the ~ character
  I = InStr(sStr, "~")
  If I < 1 Then Exit Function
  K = Len(MARKER)
  J = InStr(sStr, MARKER)
  If J < 1 Then Exit Function
  J = J + Len(MARKER)
  ' find the comma that comes after the model
  K = InStr(J, sStr, ",")
  GetModel = Mid(sStr, J, K - J)
End Function

Sub Parse()
  Const OUTPUT_EXT = ".res.txt"
  Dim inFile As Integer, sInFilename As String
  Dim outFile As Integer, sOutFilename As String
  Dim sLine As String
  Dim I As Integer, J As Integer, K As Integer, N As Integer, M As Integer
  Dim C1 As Integer, C2 As Integer, R As Integer
  Dim X As Double

  ' clear the spreadsheet
```

```
ActiveSheet.Range("A:Z").Clear

' prompt user for input filename
With Application.FileDialog(msoFileDialogOpen)
  '.Filters(1) = "All files|*.*"
  .Show
  If .SelectedItems.Count > 0 Then
    sInFilename = .SelectedItems(1)
  End If
End With

' find the last dot in the input filename
I = InStrRev(sInFilename, ".")
' make up the output filename
If I > 0 Then
  sOutFilename = Left(sInFilename, I - 1) & OUTPUT_EXT
Else
  sOutFilename = sInFilename & OUTPUT_EXT
End If
' tell the user about the output filename
MsgBox "Output file is:" & vbCrLf & sOutFilename, _
       vbOKOnly + vbInformation, "For Your Information"

On Error GoTo HandleErr
' open input filr
inFile = FreeFile
Open sInFilename For Input As #inFile
' open output file
outFile = FreeFile
Open sOutFilename For Output As #outFile

' write some of the headers
Range("A1").Value = "Best # of Models"
Range("B1").Value = "Num # of Obs"
Range("C1").Value = "F"
Range("D1").Value = "R^2"
Range("E1").Value = "Model"
Range("F1").Value = "Intercept"

R = 2 ' start at second row
' loop for all input lines
Do While Not EOF(inFile)
  ' read the next line
  Line Input #inFile, sLine
  ' process non-bank lines
  If Trim(sLine) <> "" Then
    ' Line starts with the word "Bes "?
    If InStr(LCase(sLine), "best ") = 1 Then
      Print #outFile, sLine
      Cells(R, 1) = GetValueIn(sLine)
    ' line starts with the words "Number of "?
    ElseIf InStr(LCase(sLine), "number of ") = 1 Then
      Print #outFile, sLine
      Cells(R, 2) = GetValueIn(sLine)
    Else ' found line with regression statistics and model
```

```
        Cells(R, 3) = GetValueAfter(sLine, "F")
        Cells(R, 4) = GetValueAfter(sLine, "R^2")
        Cells(R, 5) = GetModel(sLine)
        Cells(R, 6) = GetValueAfter(sLine, "Intercept")

        I = 1 ' initialize variable counter
        C1 = 6 ' initialize column for first variable
        C2 = C1 ' intiialize column for next variable
        Do
          ' get teh value for the next slope
          X = GetValueAfter(sLine, "Slope" & I)
          ' Exit loop if no slope is found
          If X > BAD_RESULT Then Exit Do
          C2 = C2 + 1 ' increment next-variable coulmn index
          ' adjust header
          Cells(1, C2) = "Slope" & I
          ' store regression coefficient in cell
          Cells(R, C2) = X
          I = I + 1
        Loop

        ' now output to file
        Print #outFile, "F = " & Cells(R, 3) & ", R^2 = " & Cells(R, 4)
        Print #outFile, PrintModel(sLine, R, C1)
        R = R + 1
      End If
    End If
  Loop

ExitProc:
  Close #inFile
  Close #outFile
  Exit Sub

HandleErr:
  MsgBox "Error " & Err.Description, vbOKOnly + vbCritical, "Error"
  Resume ExitProc

End Sub
```

The Main Tasks of the VBA Code

The VBA code works within an Excel spreadsheet that is initially empty. When you run the Parse macro, it performs the following tasks:

1. Prompts you to select an input file. This task uses the File Open common dialog box.
2. Displays the name of the output file, based on the name of the input file you select. The code replaces the input filename extension with the extension .res.txt to come up with the output filename. For example, if you chose in step 1 the input filename best.fit.dat, the macro generates the output filename as best.fit.res.txt.
3. Reformats the input data and displays results in the spreadsheet and also writes an easy to read version to the output file. The output to Excel places in individual cells each of the

F statistic, coefficient of determination, regression model, regression intercept, and the regression slopes. The output to a text file writes the in an easily readable form

Thus, the VBA macro gives you two forms of the output that you can incorporate in your reports and presentations.

Sample Input and Output

The above VBA code converts the following sample output of the R functions:

```
Best 10 models
Number of observations = 8464
F = 109495.8443, R^2 = 0.9748922334, Model is
1/y~I(p.vect^3)+I(1/sqrt(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.452746260, Slope1 = -0.7549204893, Slope2 = -0.4726944735, and Slope3 = -
1.124248917
F = 104935.9859, R^2 = 0.9738297601, Model is
1/y~I(p.vect^3)+I(log(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.169084883, Slope1 = -0.7549204893, Slope2 = 0.05737550302, and Slope3 = -
1.124248917
F = 99877.83648, R^2 = 0.9725408042, Model is
1/y~I(p.vect^3)+I(1/df1.vect)+I(1/sqrt(df2.vect)), Intercept = 1.395449242,
Slope1 = -0.7549204893, Slope2 = -0.8593652257, and Slope3 = -1.124248917
F = 94697.83516, R^2 = 0.9710822129, Model is
1/y~I(p.vect^2)+I(1/sqrt(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.742429573, Slope1 = -1.040213869, Slope2 = -0.4726944735, and Slope3 = -
1.124248917
F = 91733.97327, R^2 = 0.9701757641, Model is
1/y^2~I(p.vect^3)+I(1/sqrt(df1.vect))+I(log(df2.vect)), Intercept =
0.5743216997, Slope1 = -0.7403353229, Slope2 = -0.4758895650, and Slope3 =
0.1244083215
F = 91241.89124, R^2 = 0.9700197395, Model is
1/y~I(p.vect^2)+I(log(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.458768196, Slope1 = -1.040213869, Slope2 = 0.05737550302, and Slope3 = -
1.124248917
F = 91221.61667, R^2 = 0.970013276, Model is
1/y^2~I(p.vect^3)+I(log(df1.vect))+I(log(df2.vect)), Intercept =
0.2874519906, Slope1 = -0.7403353229, Slope2 = 0.05817202802, and Slope3 =
0.1244083215
F = 90749.55379, R^2 = 0.9698619916, Model is
1/y~I(p.vect^3)+I(sqrt(df1.vect))+I(1/sqrt(df2.vect)), Intercept =
1.224827980, Slope1 = -0.7549204893, Slope2 = 0.02478626458, and Slope3 = -
1.124248917
F = 87364.54333, R^2 = 0.9687307836, Model is
1/y~I(p.vect^2)+I(1/df1.vect)+I(1/sqrt(df2.vect)), Intercept = 1.685132555,
Slope1 = -1.040213869, Slope2 = -0.8593652257, and Slope3 = -1.124248917
F = 85559.94591, R^2 = 0.968092309, Model is
1/y^2~I(p.vect^2)+I(1/sqrt(df1.vect))+I(log(df2.vect)), Intercept =
0.8598958186, Slope1 = -1.021859090, Slope2 = -0.4758895650, and Slope3 =
0.1244083215
```

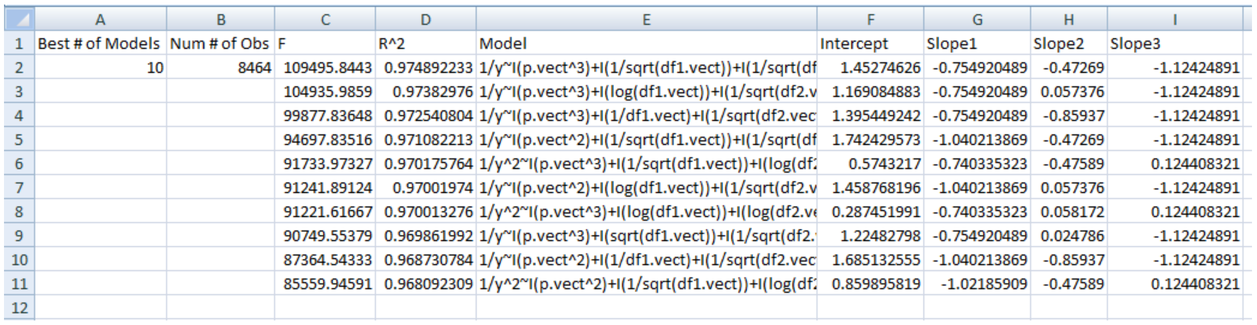
Into the following more readable output:

```
Best 10 models
```

```
Number of observations = 8464
F = 109495.8443, R^2 = 0.9748922334
1/y = 1.45274626 - 0.7549204893 * p.vect^3 - 0.4726944735 / sqrt(df1.vect) -
1.12424891 / sqrt(df2.vect)
F = 104935.9859, R^2 = 0.9738297601
1/y = 1.169084883 - 0.7549204893 * p.vect^3 + 0.05737550302 * log(df1.vect) -
1.12424891 / sqrt(df2.vect)
F = 99877.83648, R^2 = 0.9725408042
1/y = 1.395449242 - 0.7549204893 * p.vect^3 - 0.8593652257 / df1.vect -
1.12424891 / sqrt(df2.vect)
F = 94697.83516, R^2 = 0.9710822129
1/y = 1.742429573 - 1.040213869 * p.vect^2 - 0.4726944735 / sqrt(df1.vect) -
1.12424891 / sqrt(df2.vect)
F = 91733.97327, R^2 = 0.9701757641
1/y^2 = 0.5743216997 - 0.7403353229 * p.vect^3 - 0.475889565 / sqrt(df1.vect)
+ 0.124408321 * log(df2.vect)
F = 91241.89124, R^2 = 0.9700197395
1/y = 1.458768196 - 1.040213869 * p.vect^2 + 0.05737550302 * log(df1.vect) -
1.12424891 / sqrt(df2.vect)
F = 91221.61667, R^2 = 0.970013276
1/y^2 = 0.2874519906 - 0.7403353229 * p.vect^3 + 0.05817202802 *
log(df1.vect) + 0.124408321 * log(df2.vect)
F = 90749.55379, R^2 = 0.9698619916
1/y = 1.22482798 - 0.7549204893 * p.vect^3 + 0.02478626458 * sqrt(df1.vect) -
1.12424891 / sqrt(df2.vect)
F = 87364.54333, R^2 = 0.9687307836
1/y = 1.685132555 - 1.040213869 * p.vect^2 - 0.8593652257 / df1.vect -
1.12424891 / sqrt(df2.vect)
F = 85559.94591, R^2 = 0.968092309
1/y^2 = 0.8598958186 - 1.02185909 * p.vect^2 - 0.475889565 / sqrt(df1.vect) +
0.124408321 * log(df2.vect)
```

The above text output shows the models in a mathematical form that integrates the regression terms and the values for the regression slopes.

In addition, the VBA macro generates the following data in the Excel spreadsheet:



	A	B	C	D	E	F	G	H	I
1	Best # of Models	Num # of Obs	F	R^2	Model	Intercept	Slope1	Slope2	Slope3
2	10	8464	109495.8443	0.974892233	1/y~I(p.vect^3)+I(1/sqrt(df1.vect))+I(1/sqrt(df	1.45274626	-0.754920489	-0.47269	-1.12424891
3			104935.9859	0.97382976	1/y~I(p.vect^3)+I(log(df1.vect))+I(1/sqrt(df2.v	1.169084883	-0.754920489	0.057376	-1.12424891
4			99877.83648	0.972540804	1/y~I(p.vect^3)+I(1/df1.vect)+I(1/sqrt(df2.vec	1.395449242	-0.754920489	-0.85937	-1.12424891
5			94697.83516	0.971082213	1/y~I(p.vect^2)+I(1/sqrt(df1.vect))+I(1/sqrt(df	1.742429573	-1.040213869	-0.47269	-1.12424891
6			91733.97327	0.970175764	1/y^2~I(p.vect^3)+I(1/sqrt(df1.vect))+I(log(df	0.5743217	-0.740335323	-0.47589	0.124408321
7			91241.89124	0.97001974	1/y~I(p.vect^2)+I(log(df1.vect))+I(1/sqrt(df2.v	1.458768196	-1.040213869	0.057376	-1.12424891
8			91221.61667	0.970013276	1/y^2~I(p.vect^3)+I(log(df1.vect))+I(log(df2.ve	0.287451991	-0.740335323	0.058172	0.124408321
9			90749.55379	0.969861992	1/y~I(p.vect^3)+I(sqrt(df1.vect))+I(1/sqrt(df2.	1.22482798	-0.754920489	0.024786	-1.12424891
10			87364.54333	0.968730784	1/y~I(p.vect^2)+I(1/df1.vect)+I(1/sqrt(df2.vec	1.685132555	-1.040213869	-0.85937	-1.12424891
11			85559.94591	0.968092309	1/y^2~I(p.vect^2)+I(1/sqrt(df1.vect))+I(log(df	0.859895819	-1.02185909	-0.47589	0.124408321
12									