The GSqwsr R package

Laura De Cicco[1], Steve Corsi[1], and Austin Baldwin[1]

[1] *United States Geological Survey*

January 22, 2014

Contents

1 Introduction to GSqwsr package

The GSqwsr (USGS water quality surrogate regressions) package was designed to simplify the process of gathering water quality sample data and unit surrogate data, running a stepwise regression using the USGSwsQW censReg function, and analyzing those results. This vignette will first show a general overview workflow (2), then a more detailed description of the workflow with working examples (3).

2 General Workflow

```
library("GSqwsr")

#Sample data included with package:
DTComplete <- StLouisDT
UV <- StLouisUV
QWcodes <- StLouisQWcodes
siteINFO <- StLouisInfo

investigateResponse <- "SuspSed"
transformResponse <- "lognormal"

DT <- DTComplete[c(investigateResponse,
                   getPredictVariables(names(UV)),
                   "decYear","sinDY","cosDY","datetime")]
DT <- na.omit(DT)

predictVariables <- names(DT)[-which(names(DT)
                  %in% c(investigateResponse,"datetime","decYear"))]


#Check predictor variables
predictVariableScatterPlots(DT,investigateResponse)

# Create 'kitchen sink' formula:
kitchenSink <- createFullFormula(DT,investigateResponse)

#Run stepwise regression with "kitchen sink" as upper bound:
returnPrelim <- prelimModelDev(DT,investigateResponse,kitchenSink,
                               "BIC", #Other option is "AIC"
                               transformResponse)

steps <- returnPrelim$steps
modelResult <- returnPrelim$modelInformation
```

```
modelReturn <- returnPrelim$DT.mod

# Analyze steps found:
plotSteps(steps,DT,transformResponse)
analyzeSteps(steps, investigateResponse,siteINFO)

# Analyze model produced from stepwise regression:
resultPlots(DT,modelReturn,siteINFO)
resultResidPlots(DT,modelReturn,siteINFO)

# Create prediction plots
predictionPlot(UV,DT,modelReturn,siteINFO=siteINFO)
```

3 Workflow Details

In this section, we will step through the basic workflow.

3.1 Data Retrieval

Data retrieval is currently supported by web service calls to the National Water Information Service (NWIS). The first step is to get the discrete sample data that the regressions are modeling. In this example, we will look at the St Louis River at Scanlon (USGS site ID 04024000). If we don't know the sample data that is available, we can use the whatQW function to discover that information.

```
library(GSqwsr)
```

```
site <- "04024000"
QWcodes <- whatQW(site, minCount=20)
head(QWcodes)
```

```
  parameter_cd  startDate    endDate count service
1        00010 1964-10-28 2014-01-21   266      qw
2        00020 1974-10-30 2014-01-21   155      qw
3        00025 1981-10-20 2014-01-21   136      qw
4        00041 2010-10-07 2013-06-11    55      qw
5        00055 2010-10-07 2013-04-27    35      qw
6        00061 1960-07-28 2013-07-17   310      qw
```

Most likely, there will be a known set of parameters that are to be modeled. If the parameter codes for these analytes are known, the data from NWIS can be accessed directly with the function

importNWISqw. The following example shows the process, and then lists the column names returned in the QW dataframe.

```
pCodeQW <- c("00608","00613","00618","00631","00665",
             "00671","00940","62855","80154")
startDate <- "2011-03-17"
endDate <- ""
QW <- importNWISqw(site, params=pCodeQW,
                   begin.date=startDate, end.date=endDate)
```

```
names(QW)
```

```
 [1] "site_no"              "sample_dt"
 [3] "sample_tm"            "tzone_cd"
 [5] "medium_cd"            "Ammonia.N"
 [7] "Nitrite.N"            "Nitrate.N"
 [9] "NO2PlusNO3.N"         "Phosphorus_WW.P"
[11] "OrthoPhosphate.P"     "Chloride"
[13] "NitrogenTotal_WW.sum" "SuspSed"
[15] "datetime"
```

This brings the data in automatically as a 'qw' object. This means that censoring information is embeded within each data point. If any processing needs to be done to the data, it might be easier to import the raw data first, then convert to 'qw' objects with the makeQWObjects function.

```
QWRaw <- retrieveNWISqwData(site,pCodeQW,startDate,
                            endDate,expanded=TRUE)
QW <- makeQWObjects(QWRaw)
```

Next, the unit value data that will be used as surrogates for the analytes should be retrieved. If the parameters are not known, they can be discovered using the getDataAvailability function, filtering just the 'uv' (unit value) data:

```
UVcodes <- getDataAvailability(site)
UVcodes <- UVcodes[UVcodes$service == "uv",]
names(UVcodes)
```

```
 [1] "parameter_cd"       "statCd"
 [3] "startDate"          "endDate"
 [5] "count"              "service"
 [7] "parameter_group_nm" "parameter_nm"
 [9] "casrn"              "srsname"
[11] "parameter_units"
```

```
UVcodes$parameter_cd


[1] "00010" "00021" "00060" "00065" "00095" "00300" "00301"
[8] "00400" "63680"
```

Finally, the unit value data can be retrieved with the getMultipleUV function. Because of the potentially large amount of data being returned, the web service call is automatically split into individual parameter codes.

```
UVpCodes <- c("00010","00060","00095","00300","00400","63680")
UV <- getMultipleUV(site, startDate, endDate, UVpCodes)
```

```
names(UV)


 [1] "agency_cd"    "site_no"      "datetime"     "tz_cd"
 [5] "Wtemp"        "Wtemp_cd"     "Flow"         "Flow_cd"
 [9] "SpecCond"     "SpecCond_cd" "DO"           "DO_cd"
[13] "pH"           "pH_cd"        "Turb"         "Turb_cd"
```

3.2 Data Merging

We now need to merge the sample and continuous data into one dataframe. This is accomplished using the mergeDatasets function.

```
mergeReturn <- mergeDatasets(QW, UV, QWcodes)
DTComplete <- mergeReturn$DTComplete
QWcodes <- mergeReturn$QWcodes
```

The dataframe DTComplete contains a column of each of the discrete samples, and a column of the nearest (temporaly) unit value data. The function mergeDatasets has an argument called 'max.diff'. The default is set to '2 hours', meaning that if the sample and continious data timestamps do not match, the merge will take the closest continious data within 2 hours. This value can be changed, see ?mergeNearest for more options.

3.3 Data Investigation

3.3.1 Narrow down investigation

We now want to narrow our investigation down to one analyte. Let us look at Chloride. First we will want a dataframe DT with just chloride and the unit values. We will call these the 'prediciton values' because they will eventually be used to predict chloride.

```
investigateResponse <- "Chloride"
predictionVariables <- getPredictVariables(names(UV))

DT <- DTComplete[c(investigateResponse,
                   predictionVariables,
                   "decYear","sinDY","cosDY","datetime")]

names(DT)
```

```
 [1] "Chloride" "Wtemp"    "Flow"     "SpecCond" "DO"
 [6] "pH"       "Turb"     "decYear"  "sinDY"    "cosDY"
[11] "datetime"
```

For the regression, there can be no NA values in any of the columns. There are many ways in R to deal with this requirement. The easiest way to do it is remove any row that has any NA. This can be done as follows:

```
DT <- na.omit(DT)
```

There may be other situations where you want to remove a column that contains the majority of the missing data.

3.3.2 Plot variables

There are a few tools included in this package to explore the data before performing the regression.

```
plotQQTransforms(DT,investigateResponse)
```

```
predictVariableScatterPlots(DT,investigateResponse)
```

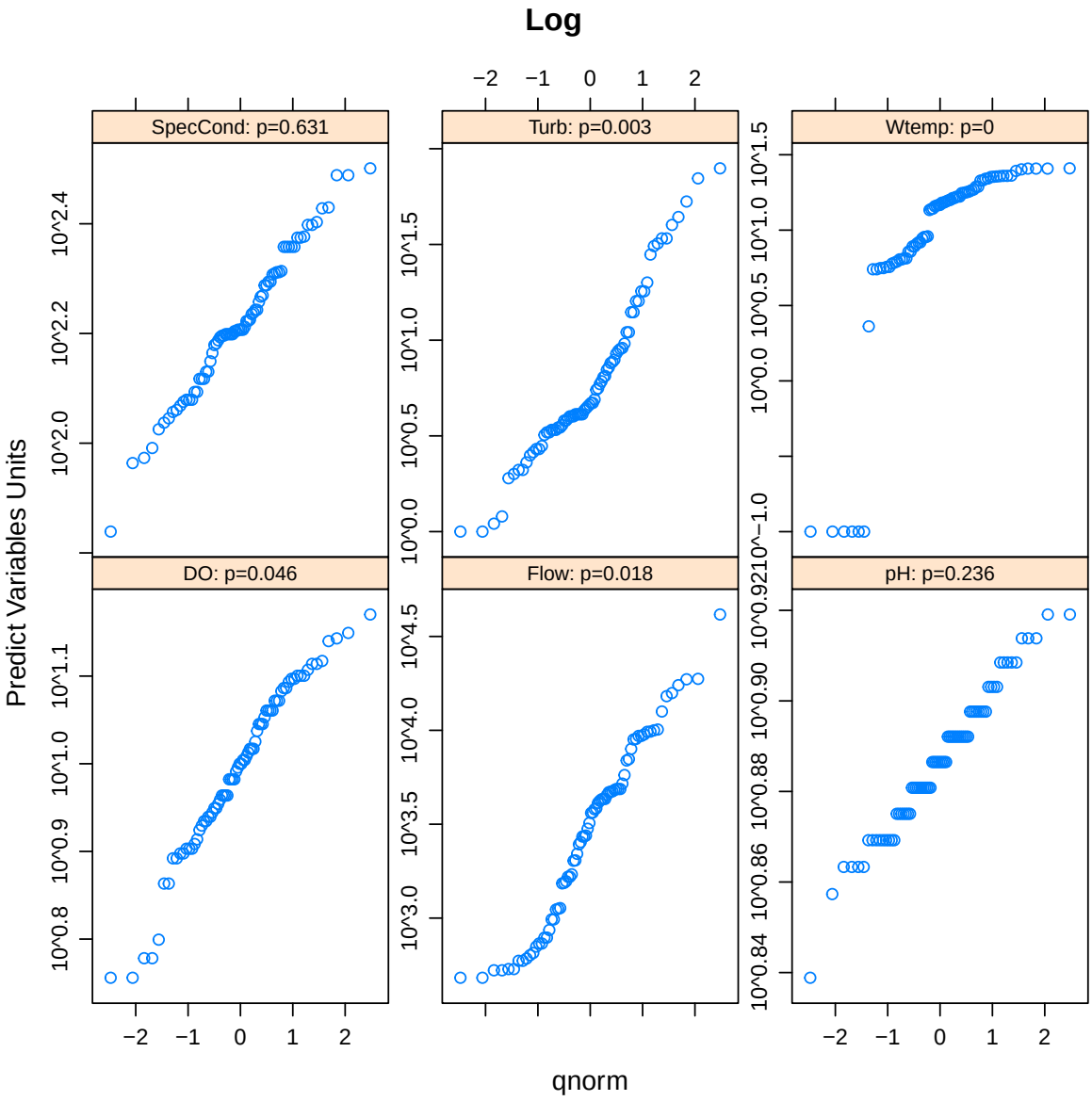


Figure 1: plotQQTransforms

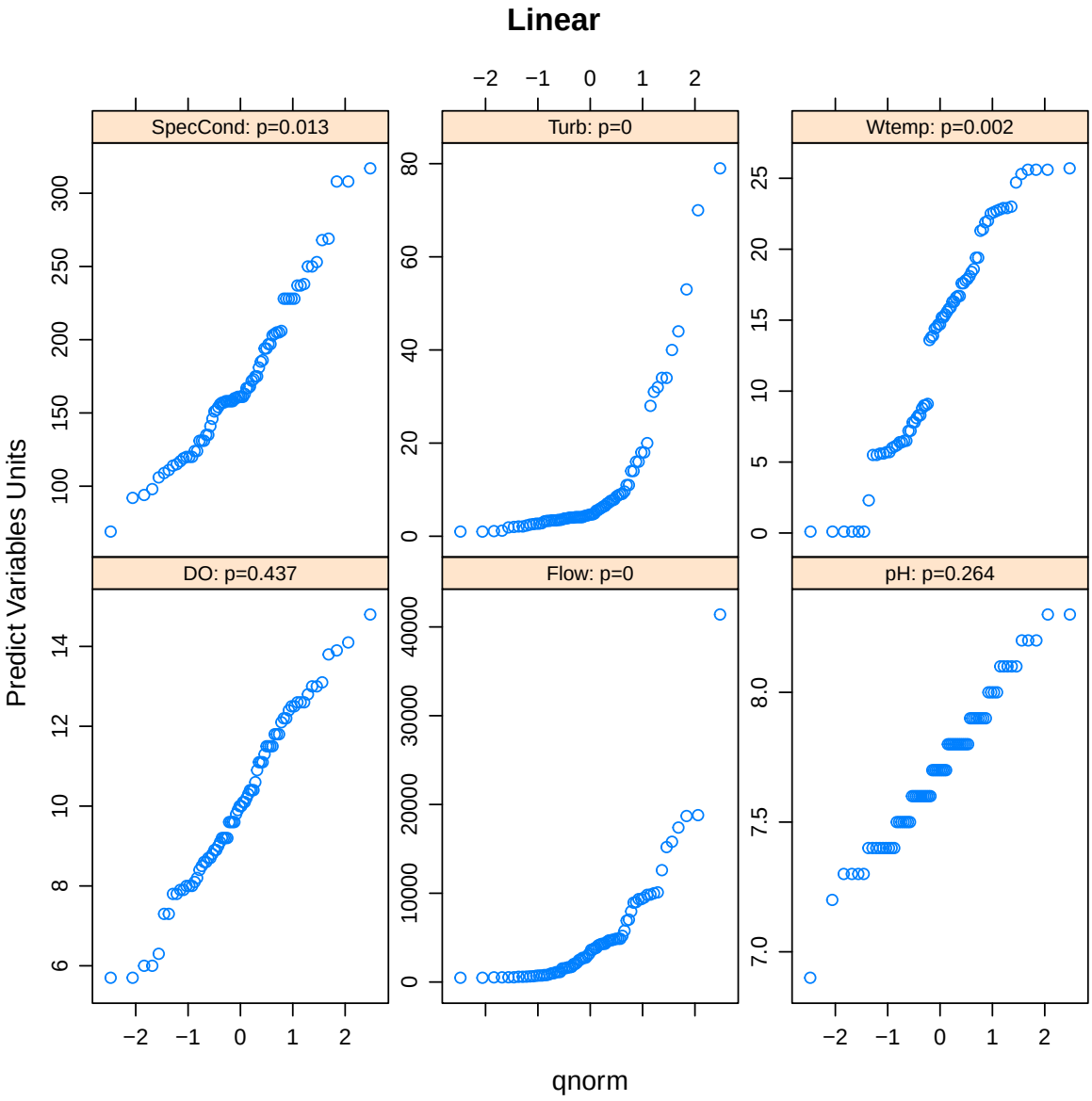


Figure 2: plotQQTransforms

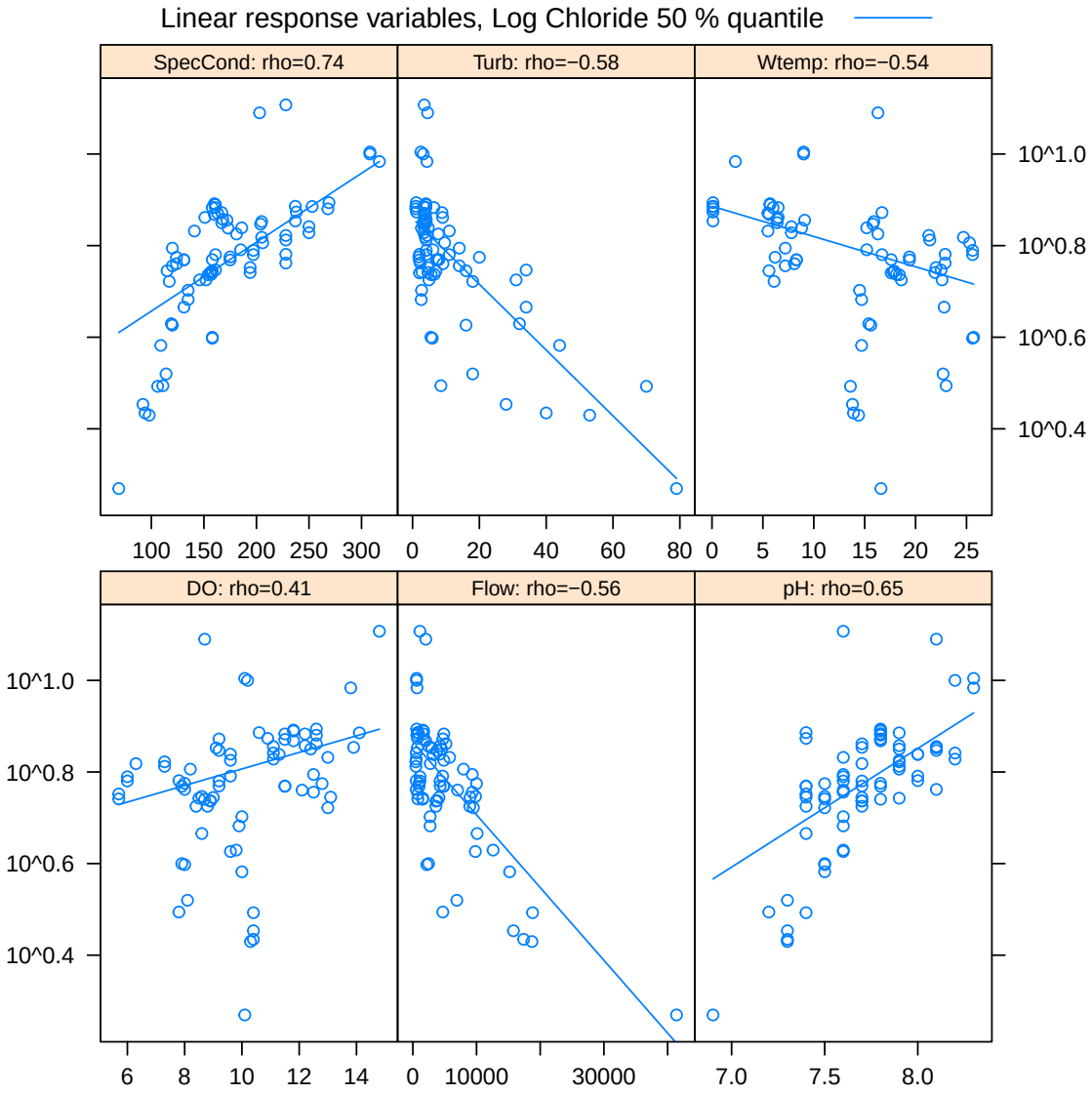


Figure 3: predictVariableScatterPlots

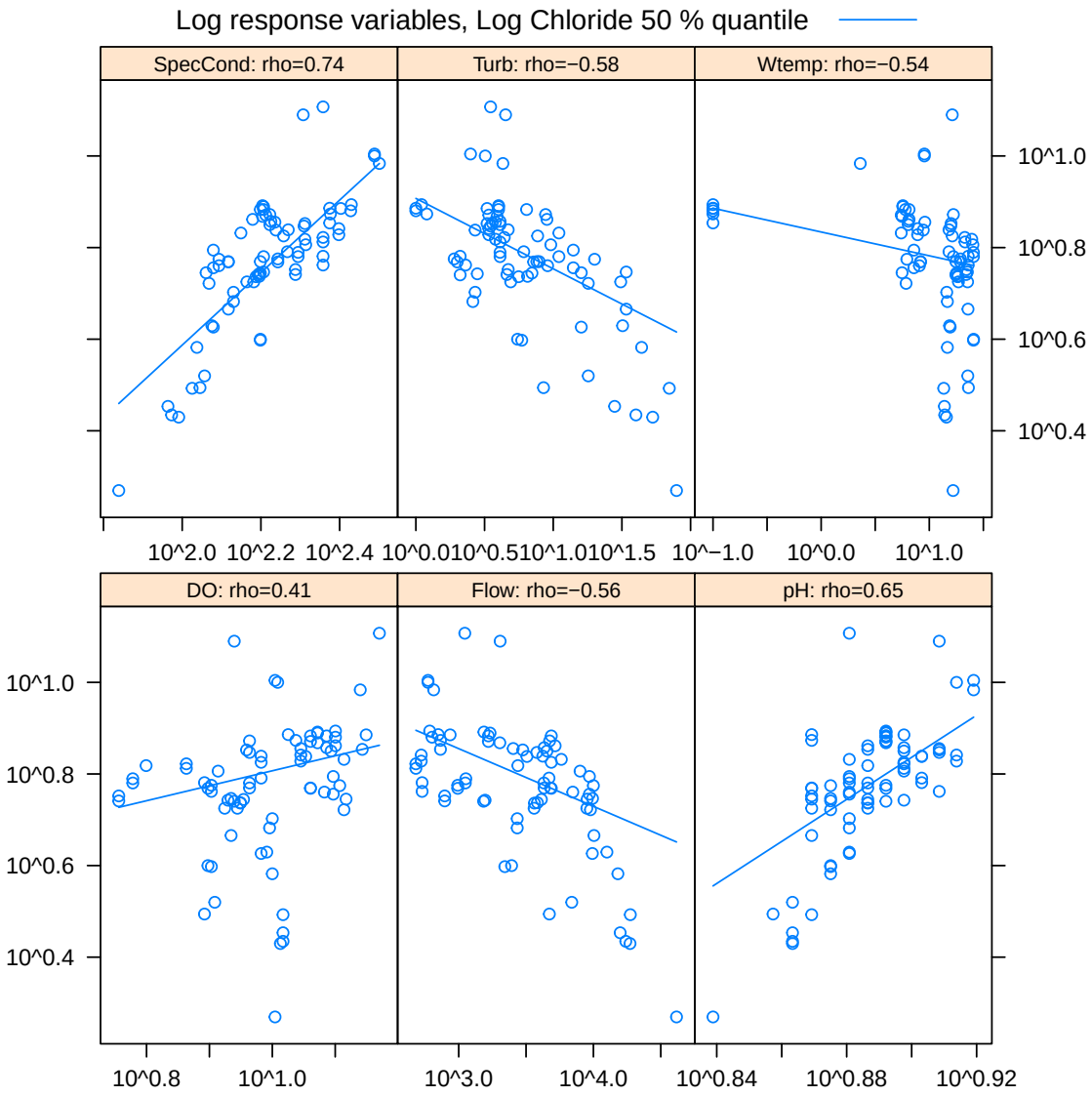


Figure 4: predictVariableScatterPlots

3.4 Stepwise Regression

We are ready to perform a stepwise regression of the data to find the most significant variables to use in the model. This is accomlished with the prelimModelDev function. There are several inputs to this function. DT is the dataframe with all the predictor variables as well as the response we are investigating. We also need to define an upper bound for the stepwise regression to test. This is an equation with the most predictor variables, along with their transforms. If we want to use all possible variables, and all available transforms, we can use the equation createFullFormula (continuing with our Chloride example):

```
upperBoundFormula <- createFullFormula(DT,investigateResponse)
```

```
[1] "Wtemp + Flow + SpecCond + DO + pH + Turb + sinDY + cosDY + "
[1] "log(Flow) + log(SpecCond) + log(DO) + log(Turb)"
```

The function will check if any data in DT has less than or equal to zero values. If so, a log transform is not included.

Now to use the stepwise regression within the prelimModelDev function:

```
transformResponse <- "lognormal"

returnPrelim <- prelimModelDev(DT,
                  investigateResponse,
                  upperBoundFormula,
                  "BIC", #Other option is "AIC"
                  transformResponse)


Start:  AIC=56.2
Chloride ~ 1

                Df   AIC
+ log(SpecCond)  1 -11.6
+ Flow           1   1.6
+ SpecCond       1   4.7
+ Turb           1   4.8
+ pH             1   7.0
+ log(Turb)      1  15.6
+ log(Flow)      1  22.5
+ cosDY          1  38.6
+ Wtemp          1  43.8
+ DO             1  53.5
+ log(DO)        1  55.2
```

```
<none>              56.2
+ sinDY          1  60.5

Step:  AIC=-11.56
Chloride ~ log(SpecCond)

                Df   AIC
+ sinDY          1 -60.6
+ DO             1 -30.5
+ Wtemp          1 -29.4
+ log(DO)        1 -29.4
+ Turb           1 -21.2
+ SpecCond       1 -18.8
+ Flow           1 -16.4
+ pH             1 -15.3
<none>             -11.6
+ log(Flow)      1  -9.6
+ log(Turb)      1  -9.2
+ cosDY          1  -8.6
- log(SpecCond)  1  56.2

Step:  AIC=-60.59
Chloride ~ log(SpecCond) + sinDY

                Df   AIC
+ log(Turb)      1 -62.7
+ pH             1 -61.5
+ SpecCond       1 -60.9
+ log(Flow)      1 -60.6
<none>             -60.6
+ cosDY          1 -60.4
+ Turb           1 -59.7
+ Flow           1 -59.0
+ DO             1 -57.0
+ log(DO)        1 -56.7
+ Wtemp          1 -56.3
- sinDY          1 -11.6
- log(SpecCond)  1  60.5

Step:  AIC=-62.74
Chloride ~ log(SpecCond) + sinDY + log(Turb)

                Df   AIC
+ Turb           1 -89.0
```

```
+ Flow           1 -73.8
+ SpecCond       1 -67.6
+ cosDY          1 -64.2
+ pH             1 -63.2
<none>             -62.7
- log(Turb)      1 -60.6
+ Wtemp          1 -59.3
+ log(Flow)      1 -58.6
+ log(DO)        1 -58.6
+ DO             1 -58.5
- sinDY          1  -9.2
- log(SpecCond)  1  19.1

Step:  AIC=-89.02
Chloride ~ log(SpecCond) + sinDY + log(Turb) + Turb

                Df   AIC
<none>             -89.0
+ pH             1 -86.2
+ cosDY          1 -85.7
+ Flow           1 -84.8
+ SpecCond       1 -84.8
+ log(Flow)      1 -84.8
+ Wtemp          1 -84.8
+ DO             1 -84.7
+ log(DO)        1 -84.7
- Turb           1 -62.7
- log(Turb)      1 -59.7
- sinDY          1 -20.9
- log(SpecCond)  1  11.5
Analyzing steps (total= 5 ): 2
3
4
5


steps <- returnPrelim$steps
modelResult <- returnPrelim$modelInformation
modelReturn <- returnPrelim$DT.mod
```

The output during the function shows the steps that the stepwise regression determined were ideal.

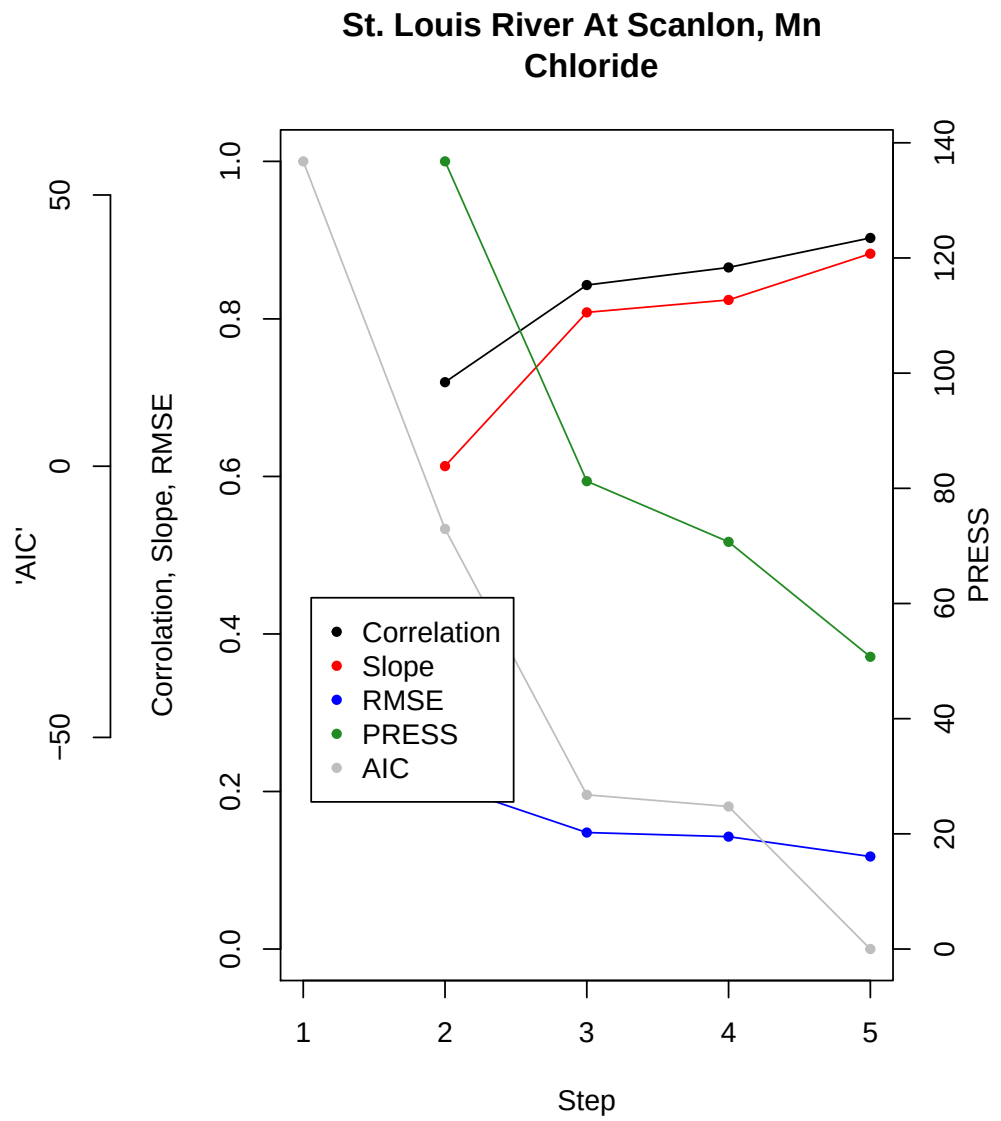


Figure 5: analyzeSteps

3.5 Stepwise Regression Analysis

It might be a good idea here to verify that the results from the stepwise regression are indeed what you want. The process can be observed using two functions: plotSteps and analyzeSteps.

```
siteINFO <- getSiteFileData(site, interactive=FALSE)
analyzeSteps(steps, investigateResponse,siteINFO)
```

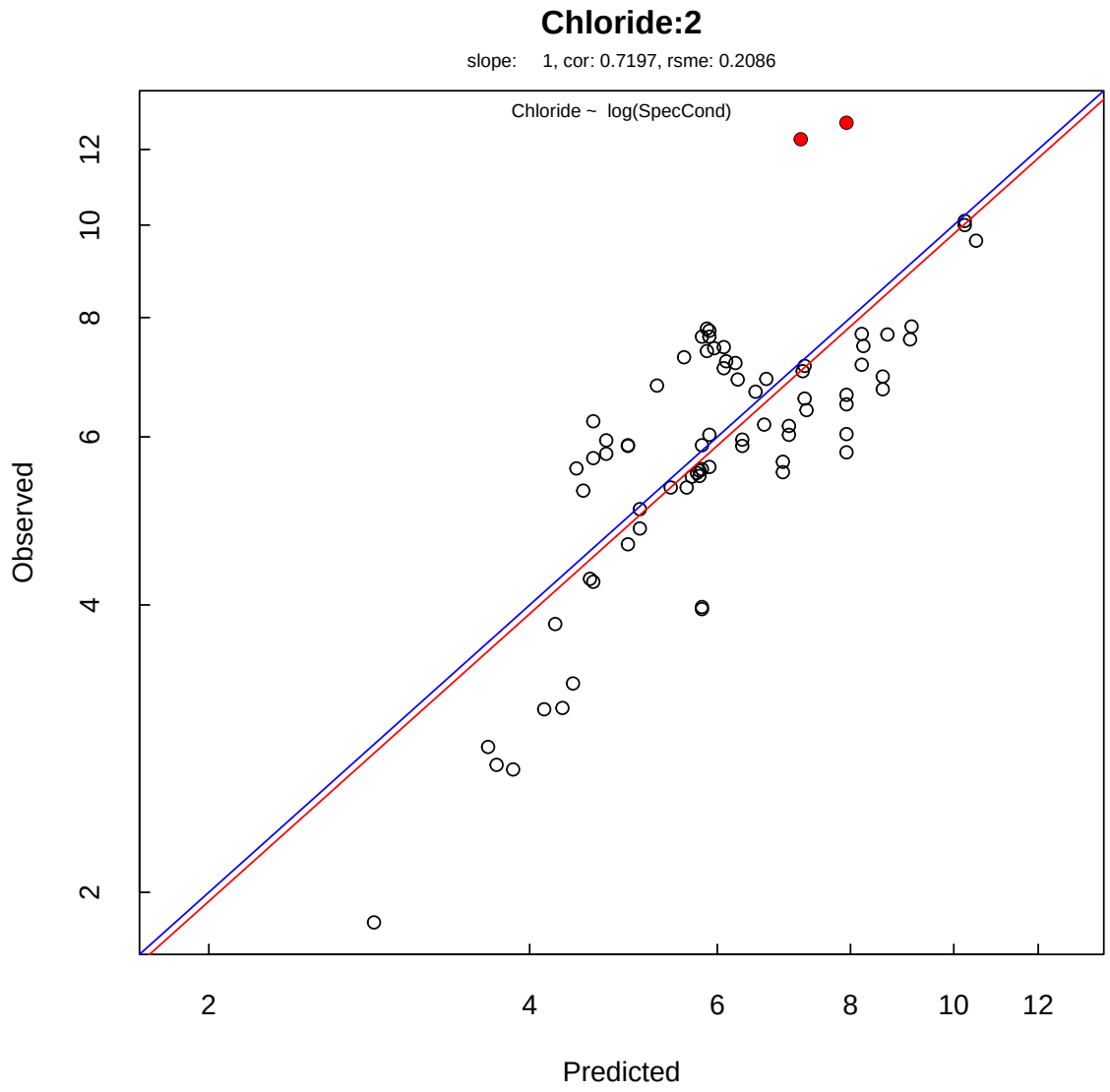


Figure 6: plotSteps

```
plotSteps(steps,DT,transformResponse)
```

```
Chloride ~  log(SpecCond)
```

```
Chloride ~  log(SpecCond) + sinDY
```

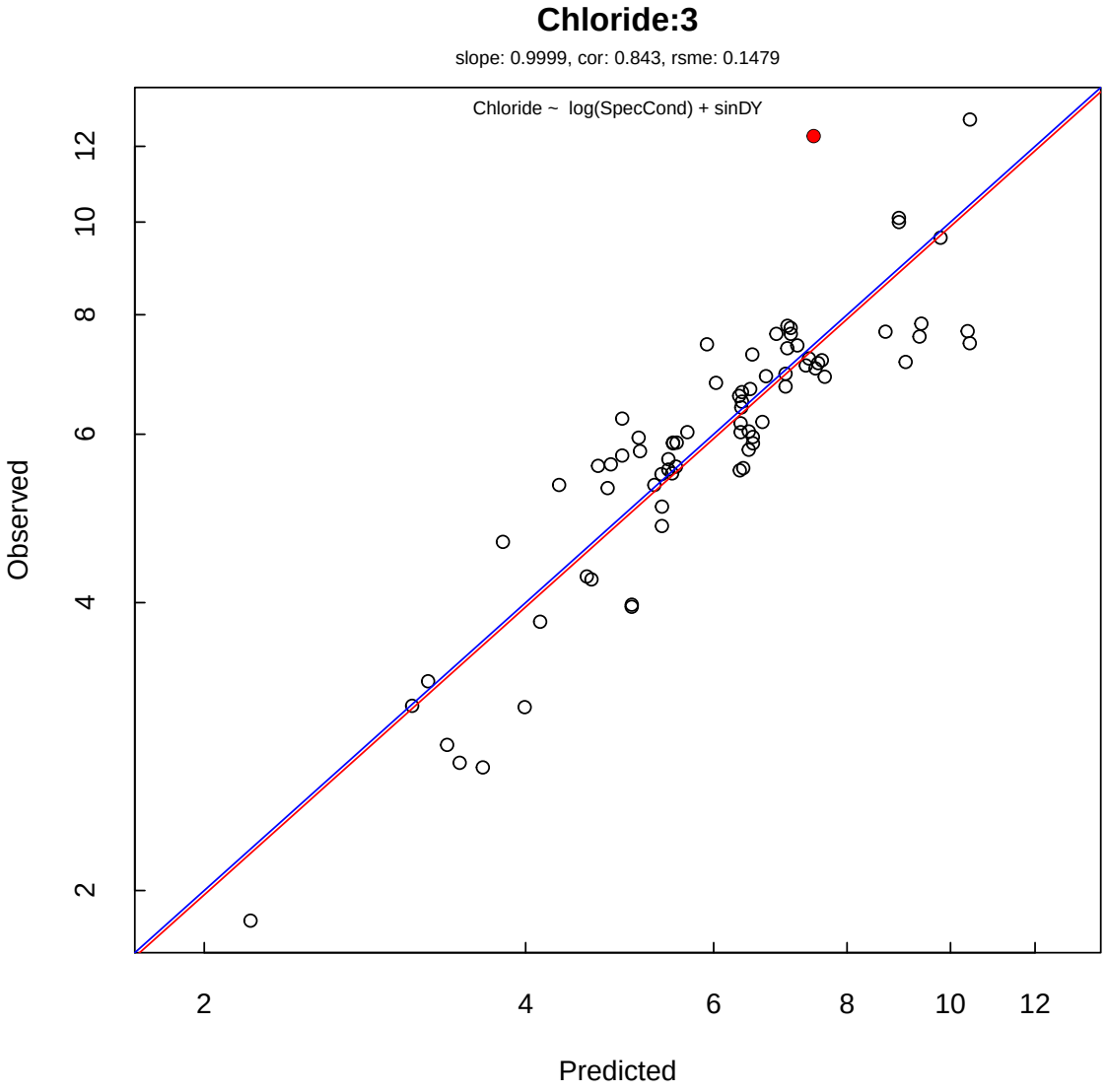


Figure 7: plotSteps

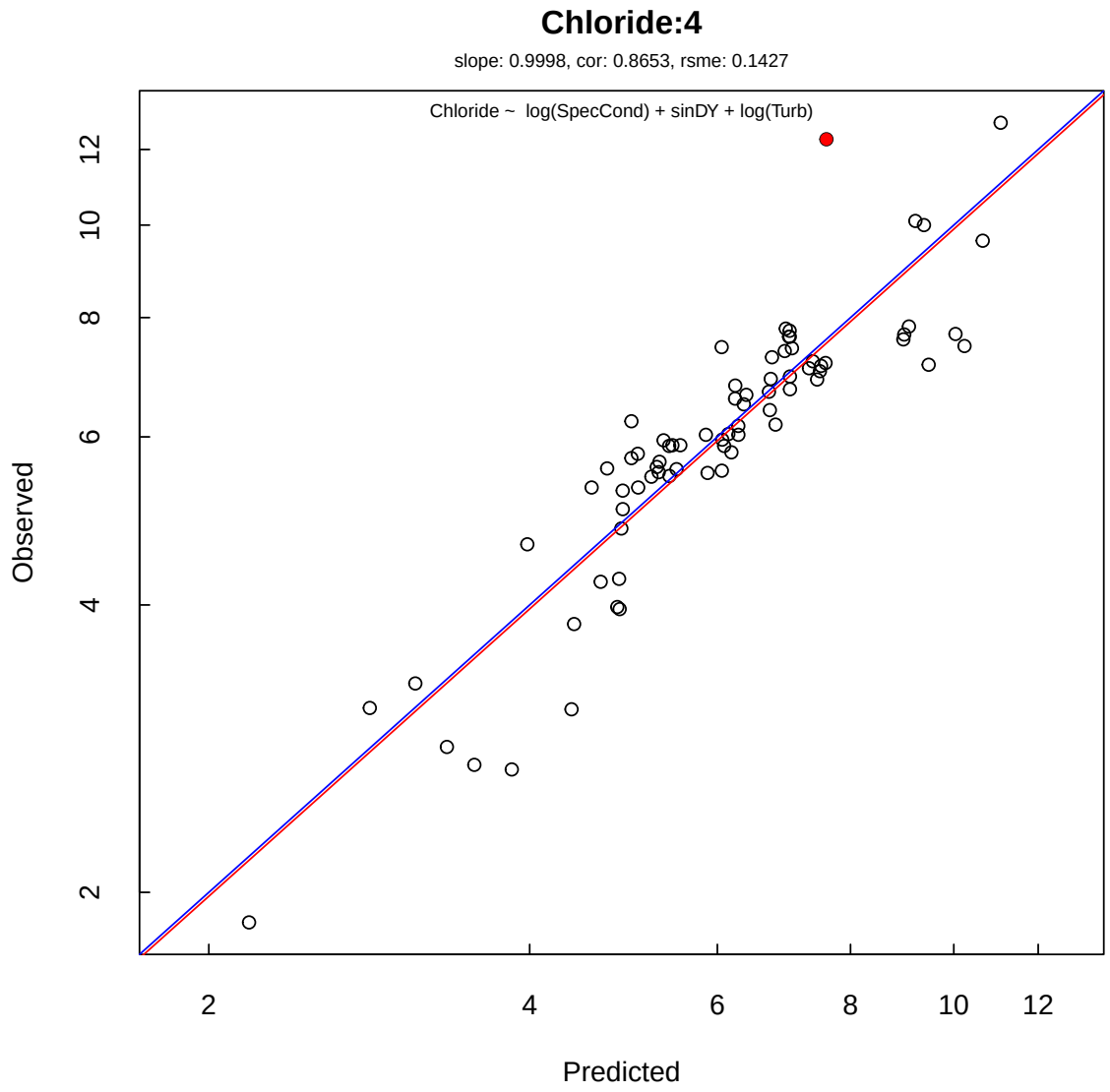


Figure 8: plotSteps

```
Chloride ~  log(SpecCond) + sinDY + log(Turb)
```

```
Chloride ~  log(SpecCond) + sinDY + log(Turb) + Turb
```

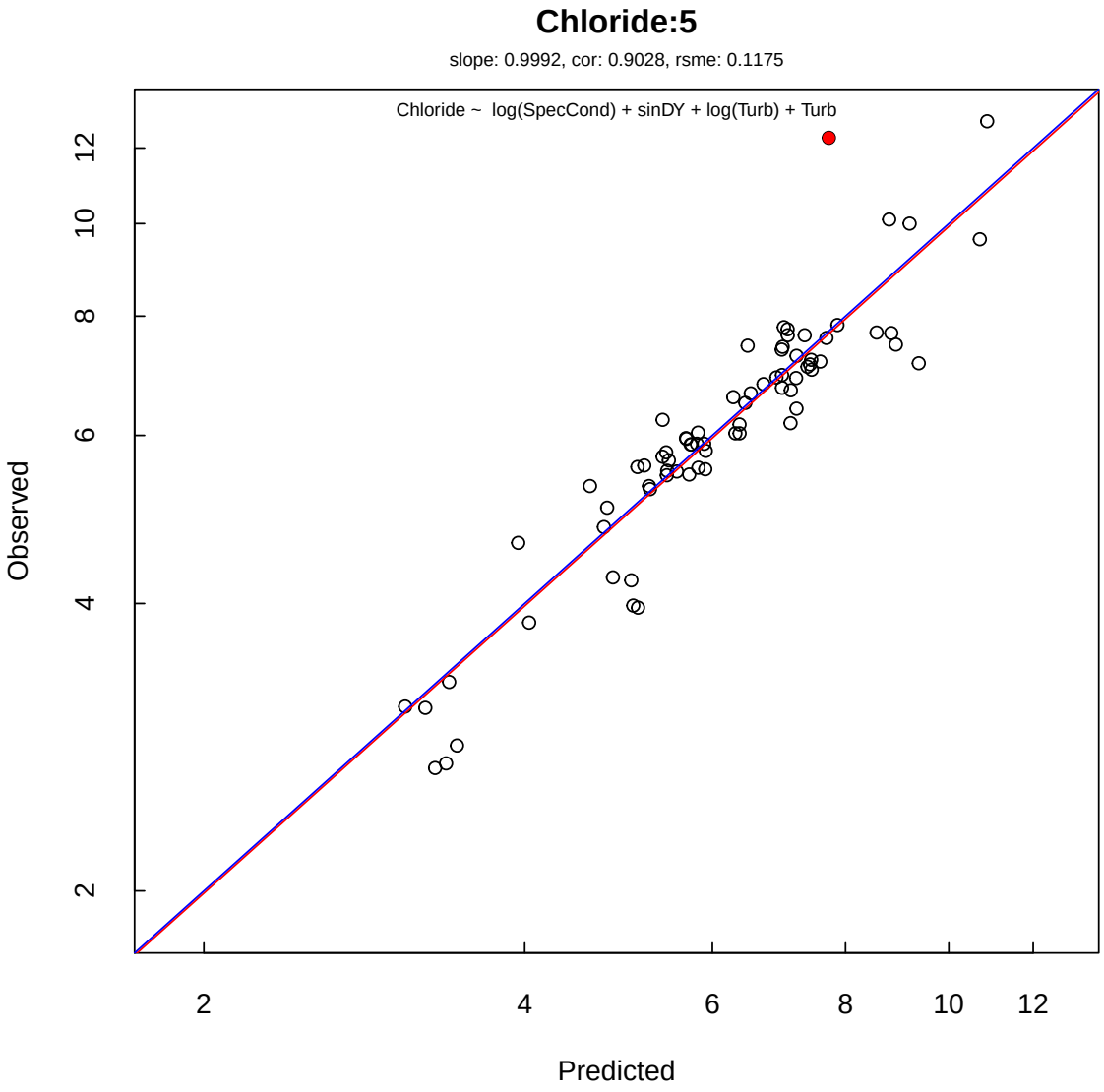
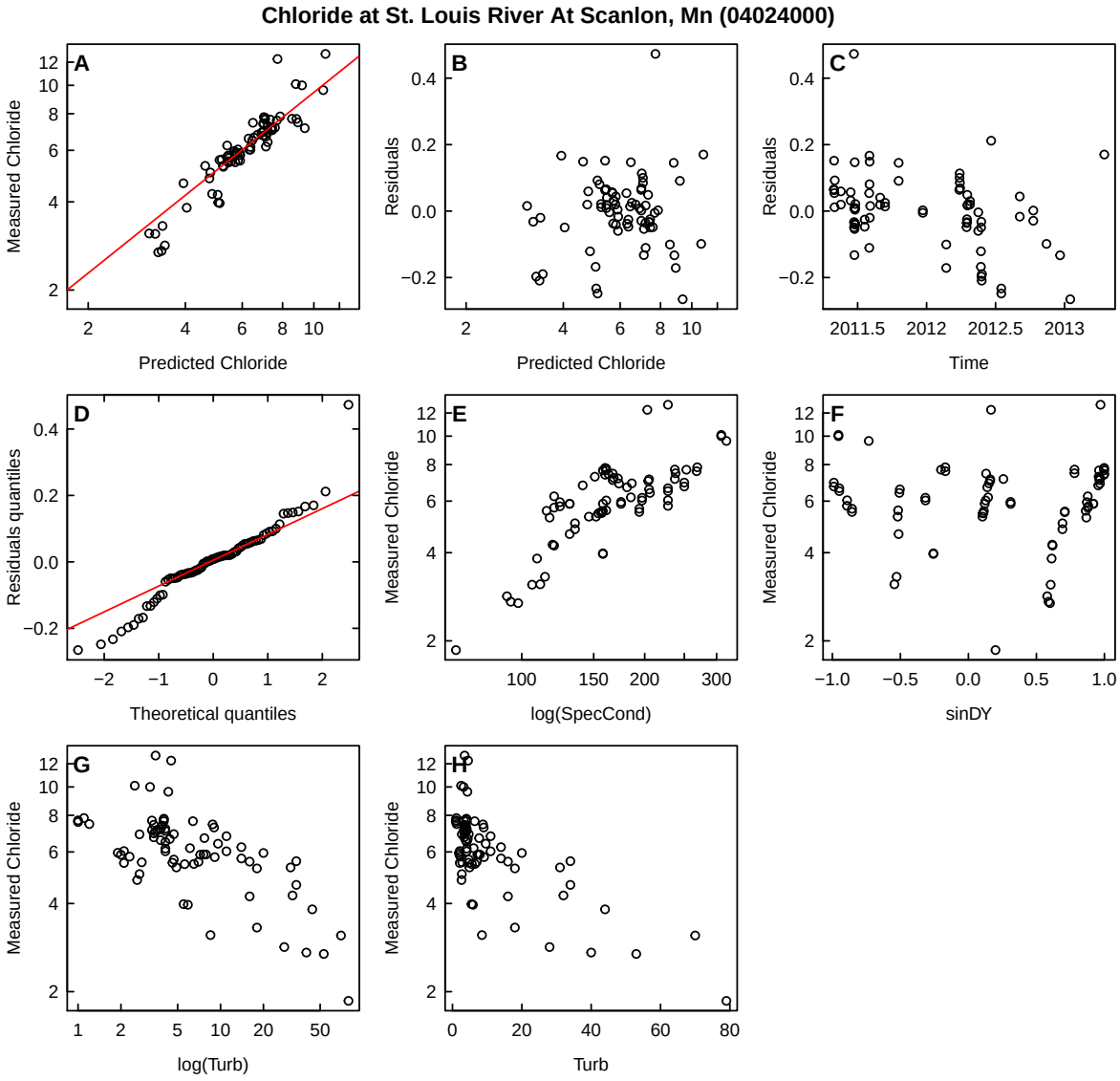


Figure 9: plotSteps

3.6 Model Analysis

Finally, the package offers several ways to analyze the model results.

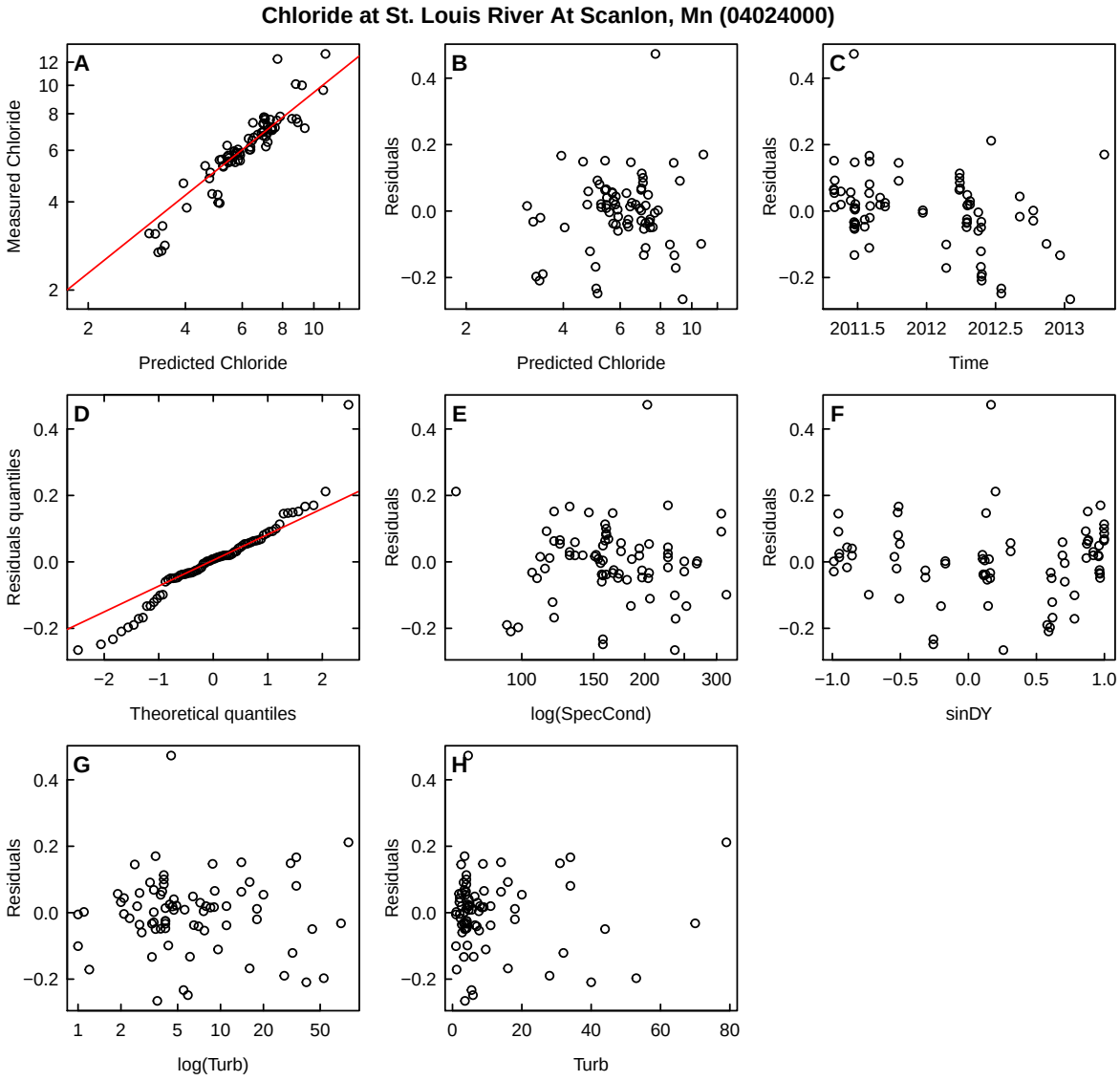
```
resultPlots(DT,modelReturn,siteINFO)
```



ln(Chloride) = −5.328 (Intercept) + 1.327 log(SpecCond) + 0.2861 sinDY + 0.2096 log(Turb) + −0.01082 Turb

Figure 10: resultPlots

```
resultResidPlots(DT,modelReturn,siteINFO)
```



ln(Chloride) = −5.328 (Intercept) + 1.327 log(SpecCond) + 0.2861 sinDY + 0.2096 log(Turb) + −0.01082 Turb

Figure 11: resultResidPlots

```
predictionPlot(UV,DT,modelReturn,siteINFO=siteINFO)
```

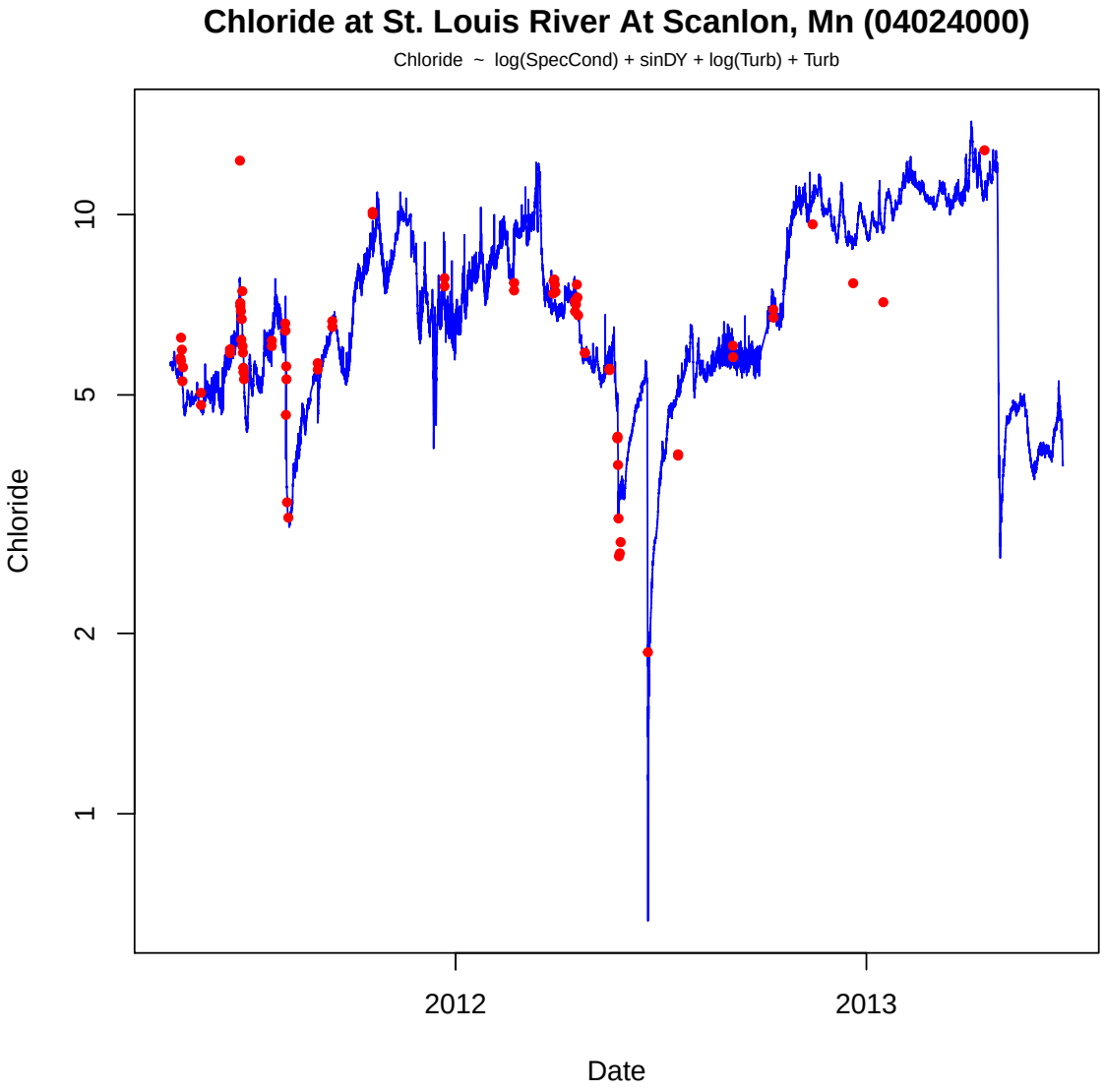


Figure 12: predictionPlot

```
summaryPrintout(modelReturn, siteINFO, saveOutput=FALSE,fileName)


Chloride  at  St. Louis River At Scanlon, Mn  ( 04024000 )
Number of observations:  76
Distribution:  lognormal
Method:  AMLE
Degrees of freedom:  12
RMSE:  0.1175
RSQ:  88.22
Number of censored values:  0
Chloride ~ (Intercept) + log(SpecCond) + sinDY + log(Turb) + Turb

           Term Coefficient StdError pValue  StCoef
1   (Intercept)      -5.328    0.507      0 -10.507
2 log(SpecCond)       1.327    0.091      0  14.532
3         sinDY       0.286    0.027      0  10.642
4     log(Turb)       0.210    0.033      0   6.287
5          Turb      -0.011    0.002      0  -5.934
6      logSigma       0.014    0.002      0   6.245

Correlation matrix of coefficients:
              (Intercept) log(SpecCond)   sinDY log(Turb)
(Intercept)        1.0000       -0.9973 -0.6424   -0.6119
log(SpecCond)     -0.9973        1.0000  0.6432    0.5596
sinDY             -0.6424        0.6432  1.0000    0.3089
log(Turb)         -0.6119        0.5596  0.3089    1.0000
Turb              -0.0203        0.0672  0.0584   -0.6791
logSigma           0.0000        0.0000  0.0000    0.0000
                  Turb logSigma
(Intercept)   -0.0203        0
log(SpecCond)  0.0672        0
sinDY          0.0584        0
log(Turb)     -0.6791        0
Turb           1.0000        0
logSigma       0.0000        1
```

A Getting Started in R

This section describes the options for downloading and installing the dataRetrieval package.

A.1 New to R?

If you are new to R, you will need to first install the latest version of R, which can be found here: `http://www.r-project.org/`.

There are many options for running and editing R code, one nice environment to learn R is RStudio. RStudio can be downloaded here: `http://rstudio.org/`. Once R and RStudio are installed, the dataRetrieval package needs to be installed as described in the next section.

At any time, you can get information about any function in R by typing a question mark before the functions name. This will open a file (in RStudio, in the Help window) that describes the function, the required arguments, and provides working examples.

```
library(GSqwsr)
?plotSteps
```

To see the raw code for a particular code, type the name of the function:

```
plotSteps
```

A.2 R User: Installing QWSR

Before installing GSqwsr, the dependent packages must be first be installed:

```
install.packages(c("XML", "lubridate", "akima", "KernSmooth",
                   "leaps", "car", "mvtnorm", "digest",
                   "relimp", "BSDA", "RODBC","memoise",
                   "boot","survival","splines","RColorBrewer",
                   "lattice","MASS"), dependencies=TRUE)
install.packages(c("USGSwsBase","USGSwsData","dataRetrieval",
                   "USGSwsGraphs","USGSwsStats",
                   "USGSwsQW","GSqwsr"),
                 repos="http://usgs-r.github.com")
```

After installing the package, you need to open the library each time you re-start R. This is done with the simple command:

```r
library(GSqwsr)
```