UCLA
UCLA Electronic Theses and Dissertations

Title
RegTools: A Julia Package for Assisting Regression Analysis

Permalink
https://escholarship.org/uc/item/9zc0q602

Author
Liang, Muzhou

Publication Date
2015

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California

Los Angeles

RegTools: A Julia Package for Assisting Regression Analysis

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Muzhou Liang

2015

© Copyright by
Muzhou Liang
2015

Abstract of the Thesis

RegTools: A Julia Package for Assisting Regression Analysis

by

Muzhou Liang

Master of Science in Statistics

University of California, Los Angeles, 2015

Professor Yingnian Wu, Chair

The **RegTools** package for Julia provides tools to select models, detect outliers and diagnose problems in regression models. The current tools include AIC, AICc and BIC based model selection methods, outlier detection methods and multicollinearity detection methods. This article briefly outlines the methodologies behind these techniques, and tests the functions by comparing with corresponding functions in R. The identical conclusions drawn from Julia and R prove the validity of **RegTools**.

The thesis of Muzhou Liang is approved.

Jingyi Li

Qing Zhou

Yingnian Wu, Committee Chair

University of California, Los Angeles

2015

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Julia is a high-level, high-performance programming language designed specifically for scientific computing, which appeared in 2012. Its performance can approach that of statically-compiled languages like C. As a young language, Julia still lacks many basic and useful functions and packages existed in other computing languages like R and Matlab. The existing Julia package **GLM** provides simple linear model and generalized linear model fitting functions, but it has no functions for post-fitting process such as model selection, outlier detection and model diagnostics, which usually take most of time in regression modeling. Therefore, a package that helps regression modeling is in need among the Julia community.

RegTools can be accessed at https://github.com/joemliang/RegTools.jl

The main aims of the **RegTools** are:

- Measure goodness of fit by R^2, adjusted R^2, AIC, AIC$_c$, BIC, etc.
- Select variables or models by AIC-based stepwise methods.
- Detect outliers using Cook's distance, jackknife methods, etc.
- Diagnose potential model problems such as multicollinearity and heteroscedasticity.

During the development, I have been mainly referencing to two books to decide which regression tools should be included in **RegTools** – *A Modern Approach to Regression with R* by Simon J. Sheather (2009) , and *Linear Models with R* by Julian J. Faraway (2004) .

CHAPTER 2

Methodolgy

2.1 Notation

In this chapter, we are looking at some methods for regression modeling. The regression model involve is defined as below. We will use the same way of notation throughout this article.

Define the $(n \times 1)$ vector $\mathbf{Y}$ of response and the $n \times (p+1)$ matrix $\mathbf{X}$ of intercept and predictors by

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix} \tag{2.1}$$

Also define the $(p+1) \times 1$ vector β of regression coefficients and the $(n \times 1)$ vector $\mathbf{e}$ of random errors by

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix} \tag{2.2}$$

We write the linear regression model in matrix notation as

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{e} \tag{2.3}$$

2.2 R^2 and Adjusted R^2

R^2 and adjusted R^2 have been widely used to indicate the goodness of fit to a statistical model. With a dataset of n observations, and $\mathbf{Y}$ as the response, R^2 is given as below:

$$R^2 = 1 - \frac{RSS}{SST} \tag{2.4}$$

where $RSS = \sum_i^n (y_i - \hat{y}_i)^2$, $SST = \sum_i^n (y_i - \bar{y})^2$. Adjusted R^2 is given by

$$R^2_{adj} = 1 - \frac{RSS/(n-p-1)}{SST/(n-1)} \tag{2.5}$$

where p is the number of predictors in the model not counting in the intercept if any.

2.3 Half-Normal Plots

Half-normal plots visualize the leverage against the positive normal quantiles, which are useful to identify the outliers (Faraway, 2004) . To be specific, on the vertical axis of a half-normal plot are $h_{[i]}$'s, while on the horizontal axis are u_i's. $h_{[i]}$'s are h_i's in ascending order, which are the diagonal elements of the hat matrix

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{2.6}$$

u_i's are given by

$$u_i = \Phi^{-1}\left(\frac{n+i}{2n+1}\right) \tag{2.7}$$

2.4 Jackknife Residuals

In addition to the visualization way above, another quantitative way to detect outliers is called Jackknife residuals by doing a t-test on it. The jackknife residual

$$\begin{aligned} t_i &= \frac{y_i - \hat{y}_{(i)}}{\hat{\sigma}_{(i)}(1 + \mathbf{x}_i^T(\mathbf{X}_{(i)}^T\mathbf{X}_{(i)})^{-1}\mathbf{x}_i)^{1/2}} \\ &= r_i\left(\frac{n-p-1}{n-p-r_i^2}\right)^{1/2} \end{aligned} \tag{2.8}$$

where r_i's are the studentized residuals, which is given by

$$r_i = \frac{\hat{e}_i}{\hat{\sigma}\sqrt{1-h_i}} \tag{2.9}$$

where $\hat{\sigma} = \sqrt{RSS/(n-p)}$.

The jackknife residual $t_i \sim t_{n-p-1}$. Thus, we can test whether case i is an outlier or not by doing a t-test on the corresponding jackknife residual.

2.5 Cook's Distance

Cook (1977, 1979) proposed a widely used measure of the influence of individual cases, which is given by

$$D_i = \frac{r_i^2}{p}\frac{h_i}{1-h_i} \tag{2.10}$$

after simplification, where r_i is the ith studentized residual.

There is no significance test for D_i. However, Fox (2002) is among many authors who recommend $4/(n-p)$ as a rough cutoff.

2.6 Akaike's Information Criterion (AIC)

Akaike's (1973, 1985) information criterion (AIC) is proposed to balance the goodness of fit and a penalty for model complexity. The goodness of fit is measured by the expected information loss, which is asymptotically equivalent to negative log-likelihood of the candidate model; while the model complexity is measured by K the number of parameters put in the candidate model. The AIC is defined as

$$\text{AIC} = 2\left[-\log\left(L\big(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_p, \hat{\sigma}^2|Y\big)\right) + K\right] \tag{2.11}$$

2.7 Corrected AIC (AIC_c)

Corrected AIC (AIC_c) (Sugiura,1978 , Hurvich and Tsai, 1989) adds a bias correction term for small sample size, which is recommended when $n/K < 40$ (Burnham and Anderson, 2002, p. 445) .

$$\text{AIC}_c = \text{AIC} + \frac{2K(K+1)}{n-K+1} \tag{2.12}$$

where n is the sample size.

2.8 Bayesian Information Criterion (BIC)

Schwarz (1978) derived the Bayesian information criterion as

$$\text{BIC} = -2log\left(L\big(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_p, \hat{\sigma}^2|Y\big)\right) + K\log(n) \tag{2.13}$$

Burnham and Anderson (2004) pointed out that "BIC is a misnomer as it is not related to information theory", and "most applications of BIC use it in a frequentist spirit and hence ignore issues of prior and posterior model probabilities".

2.9 Stepwise Model Selection

Based on AIC, AIC_c and BIC, a stepwise model selection (Hastie, Trevor J. and Daryl Pregibon, 1992) can be implemented on regression models. Since the smaller the value of AIC, AIC_c or BIC the better the model, we can add or drop one predictor and compare the criterion values at each step. There are three main approaches:

- **Forward selection**, starting with the existing model, tests the addition of each variable by a chosen criterion, then adds the variable of the best improvement if any. Repeat this process until none of the candidate variables can improve the model.
- **Backward elimination**, starting with the existing model, tests the deletion of each

variable by a chosen criterion, then deletes the variable of the best improvement if any. Repeat this process until none improves the model.

- **Bidirectional adaptation**, a combination of the above, tests if a addition or a deletion should be done at each step.

2.10 Added-Variable Plots

Added-variable plots (Mosteller and Tukey, 1977) help assess the effect of each predictor, having adjusted for the effects of the other predictors, in a multiple linear regression model,

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{e}. \tag{2.14}$$

Suppose we are considering the introduction of an additional predictor variable $\mathbf{Z}$ to the model, i.e., we are considering the model

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{Z}\alpha + \mathbf{e}. \tag{2.15}$$

The added-variable plot is obtained by plotting on the vertical axis the residuals from model (2.14) $\hat{\mathbf{e}}_{\mathbf{Y}.\mathbf{X}}$ against on the horizontal axis the residuals $\hat{\mathbf{e}}_{\mathbf{Z}.\mathbf{X}}$ from model

$$\mathbf{Z} = \mathbf{X}\delta + \mathbf{e} \tag{2.16}$$

Thus, the added-variable plot model is given by

$$\hat{\mathbf{e}}_{\mathbf{Y}.\mathbf{X}} = \hat{\mathbf{e}}_{\mathbf{Z}.\mathbf{X}}\alpha + \mathbf{e}^* \tag{2.17}$$

where $\mathbf{e}^* = (\mathbf{I} - \mathbf{H})\mathbf{e}$.

Hence, the effects brought by $\mathbf{X}$ are eliminated from both axes. $\hat{\alpha}$ will indicate if a predictor should be added into the model given other predictors.

Besides, Velleman and Welsch (1981) listed some useful properties of this plot:

- The least squares linear fit to this plot has the slope β_i and intercept zero.

- The influences of individual data values on the coefficient estimation are easy to identify.
- In addition, it enables us to find other kinds of model failures or violations of the underlying assumptions like nonlinearity, heteroscedasticity, etc.

2.11 Variance Inflation Factor (VIF)

Multicollinearity can inflate the variance amongst the variables in the model, which is problematic since some variables add very little independent information to the model (Belsly, et al. 1980) . Consider a multiple regression model

$$\mathbf{Y} = \beta_0 + \beta_1 \mathbf{x_1} + \beta_2 \mathbf{x_2} + ... + \beta_p \mathbf{x_p} + \mathbf{e} \tag{2.18}$$

The variance inflation factor of one variable x_j is the value of R^2 from the regression of x_j on the other predictors,

$$VIF_j = \frac{1}{1 - R_j^2} \tag{2.19}$$

To keep a predictor in the model, its VIF value needs to be small. Generally, we consider 5 as the cut-off, over which there would be a big chance that multicollinearity would cause a poor estimation.

CHAPTER 3

Comparison with R: A Real World Example

3.1 LifeCycleSavings Data

We want to compare **RegTools** results with R results to validate Julia package. Let's use LifeCycleSavings data, a dataset on the savings ratio over 1960–1970 in R, which can also be accessed in Julia through the package **RDatasets**, shown in Table 3.1. The meaning of each variable is listed as below:

- sr: numeric aggregat personal savings
- pop15: % of population under 15
- pop75: % of population over 75
- dpi: real per-capita disposable income
- ddpi: % growth rate of dpi

Table 3.1: An overview of the dataset LifeCycleSavings

Country	sr	pop15	pop75	dpi	ddpi
Australia	11.43	29.35	2.87	2329.68	2.87
Austria	12.07	23.32	4.41	1507.99	3.93
Belgium	13.17	23.80	4.43	2108.47	3.82
Bolivia	5.75	41.89	1.67	189.13	0.22
Brazil	12.88	42.19	0.83	728.47	4.56
Canada	8.79	31.72	2.85	2982.88	2.43

We use *sr* as response to fit a regression model in R and Julia respectively. Regression model fitting in Julia is done by the function from package **GLM**.

```
R> lm1 = lm(sr ~ pop15+pop75+dpi , data = LifeCycleSavings)
R> lm1

Call:
lm(formula = sr ~ pop15+pop75+dpi, data = LifeCycleSavings)
Coefficients:
(Intercept)        pop15        pop75          dpi
 31.4573811   -0.4921418   -1.5676746   -0.0008336

julia> lm2 = fit(LinearModel, SR ~ Pop15+Pop75+DPI, LifeCycleSavings)
julia> lm2

DataFrameRegressionModel{LinearModel{DensePredQR{Float64}},Float64}:
Coefficients:
                 Estimate   Std.Error   t value Pr(>|t|)
(Intercept)       31.4574     7.48219    4.2043   0.0001
Pop15           -0.492142    0.149044  -3.30199   0.0019
Pop75            -1.56767      1.1208  -1.39871   0.1686
DPI          -0.000833645 0.000932509 -0.893981   0.3760
```

3.2 Goodness of Fit – Function rsquared and adjrsquared

First, we check how well the model fits the dataset. I omit the beginning part of the model summary of coefficients and t-values, only showing the R^2 and adjusted R^2 part for R output.

```
R> summary(lm1)

......
```

```
Residual standard error: 3.939 on 46 degrees of freedom
Multiple R-squared:  0.2744,  Adjusted R-squared:  0.227
F-statistic: 5.797 on 3 and 46 DF,  p-value: 0.001898
```

```
julia> rsquared(lm2)

0.27435285763662065

julia> adjrsquared(lm2)

0.22702804400422627
```

3.3 Outliers Detection – Function halfnorm, rstudent, jackknife

, and cooksdistance Half-normal plots are designed to detect outliers by visualization. Function halfnorm draws a half-normal plot given a regression model object, and marks potential outliers with the corresponding case number. There is no function for half-normal plot in common R packages.

An example is shown by Figure 3.2, from which we can see that Case 44 and 21 are probably two outliers.

```
julia> halfnorm(lm2)
```

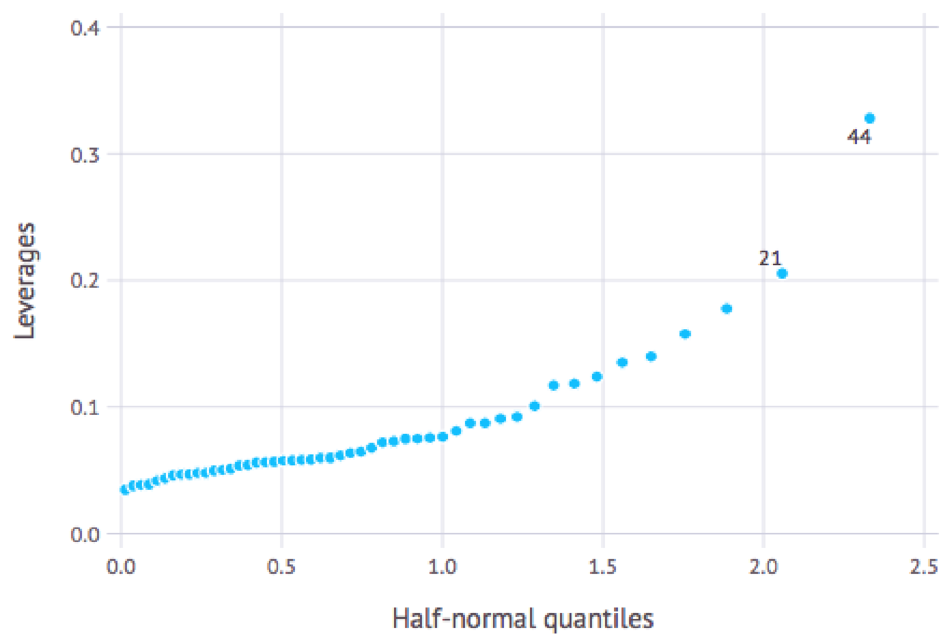

Figure 3.1: The half-normal plot with two potential outliers marked

Function rstudent gives the studentized residuals of a regression model, in the same order as the case number.

```
julia> rstudent(lm2)

50-element Array{Float64,1}:
  0.228114
  0.0710395
  0.571627
 -0.609063
  1.08655
 -0.0285486
 -2.25827
  0.991539
 -0.431082
  1.32022
  .
  .
  .
 -1.0164
 -0.998823
 -0.160697
  0.715791
  2.74097
 -0.123284
 -0.925846
  0.623287
 -0.603056
```

Function jackknife returns the jackknife residuals of a regression model, but does not do the test. To determine whether a case is an outlier or not, one needs to look up to the t-table. There is no function for jackknife in common R packages, so I only listed **RegTools** function here.

```
julia> jackknife(lm2)

50-element Array{Float64,1}:
  0.225799
  0.0702834
  0.567489
 -0.60494
  1.08868
 -0.0282435
 -2.36618
  0.991358
 -0.427317
```

```
  1.33101
  .
  .
  .
 -1.01676
 -0.998797
 -0.159022
  0.712027
  2.9584
 -0.121985
 -0.924412
  0.619185
 -0.598928
```

Function cooksdistance generates the Cook's distance for each case, and can output a graph and mark the cases of which the Cook's distances are greater than the cutoff $(4/(n-p)$. The example graph suggests that case 46 and 23 might be outliers.

```
julia> cooksdistance(lm2)

50-element Array{Float64,1}:
 0.000944858
 0.000169391
 0.00781867
 0.00467338
 0.0204861
 3.81398e-5
 0.0458411
 0.01411
 0.00279714
 0.0353162
 0.0368696
 0.00961817
 0.0035392
 .
 .
 .
 0.046377
 0.0072154
 0.00591886
 0.015858
 0.0279255
 0.00315124
 0.00995356
 0.116378
 0.000149496
 0.0173407
 0.0128716
 0.00556643
```

```
julia> cooksdistance(lm2, plotit = true)
```

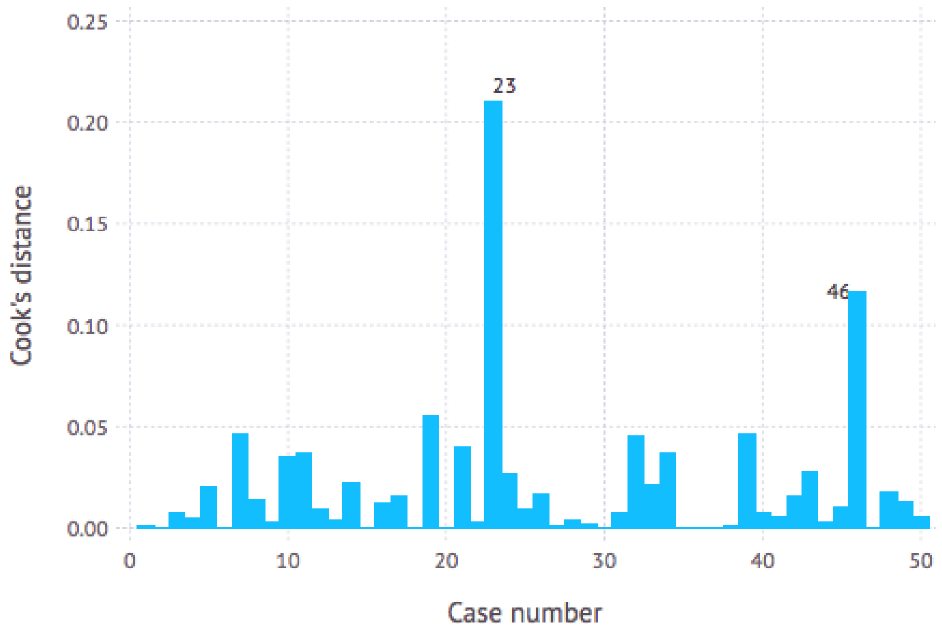

Figure 3.2: The plot for Cook's distance

3.4 Model Comparison Criterions

Besides checking R^2 and adjusted R^2, we can also examine AIC, AIC$_c$ and BIC of the model. In **RegTools**, we use function AIC, AICc and BIC, all taking a RegressionModel argument. For R, we only show AIC and BIC since R has no function for AIC$_c$. We can see that R and **RegTools** yield identical results.

```
R> extractAIC(lm1)

[1]   4.0000 140.9267

R> n = length(lm1$residuals)
R> extractAIC(lm1, k=log(n)) # BIC

[1]   4.0000 148.5748

julia> AIC(lm2)
```

```
140.926680670782

julia> AICc(lm2)

141.81556955967088

julia> BIC(lm2)

148.57477269249458
```

3.5 Functions add1 and drop1

Functions add1 and drop1 is to add one variable from the candidate variables and to eliminate one predictor from the current model, respectively. add1 takes three arguments: a RegressionModel, a scope of String type, and a full dataset of DataFrame type.

R results suggest we should add ddpi as AIC lowers from 140.93 to 138.30; in terms of drop1, we should drop dpi as AIC would go down to 139.79.

```
R> add1(lm1, ~ ddpi + I(ddpi^2) + .)

Single term additions
Model:
sr ~ pop15 + pop75 + dpi
          Df Sum of Sq    RSS    AIC
<none>                 713.77 140.93
ddpi       1    63.054 650.71 138.30
I(ddpi^2)  1    32.946 680.82 140.56
R> drop1(lm1)
Single term deletions
Model:
sr ~ pop15 + pop75 + dpi
```

```
       Df Sum of Sq    RSS    AIC
<none>              713.77 140.93
pop15   1   169.181 882.95 149.56
pop75   1    30.357 744.12 141.01
dpi     1    12.401 726.17 139.79
```

RegTools gives the same conclusions as R.

```
julia> add1(lm2, "DDPI+DDPI&DDPI", LifeCycleSavings)

Add DDPI with AIC = 138.3022838270998

julia> drop1(lm2)

Drop DPI with AIC = 139.78791765459198
```

3.6 Stepwise Model Selection

We test the stepwise model selection function here. Stepwise method is just a combination of add1 and drop1 at each step, and will keep going until no action can improve the model. Function step takes one argument - RegressionModel.

Again, the results of R and **RegTools** agree with each other. Also, running time is shown here - 0.020 seconds in R and 0.0032 seconds in Julia.

```
R> system.time(step(lm1))

Start:  AIC=140.93
sr ~ pop15 + pop75 + dpi
        Df Sum of Sq    RSS    AIC
- dpi    1    12.401 726.17 139.79
<none>              713.77 140.93
```

```
- pop75  1    30.357 744.12 141.01
- pop15  1   169.181 882.95 149.56

Step:  AIC=139.79
sr ~ pop15 + pop75
        Df Sum of Sq    RSS    AIC
<none>                726.17 139.79
- pop75  1    53.343 779.51 141.33
- pop15  1   158.915 885.08 147.68

   user  system elapsed
  0.018   0.001   0.020
```

```
julia> @time step(fm3, "both", false)
```

```
Drop DPI with AIC = 139.78791765459198
elapsed time: 0.003172594 seconds (373644 bytes allocated)
```

3.7 Function avPlot for Added-Variable Plots

The added-variable plots can be used to identify which variable adds little information to the model with the existence of other predictors. The function avPlot generate Figure 3.3 for corresponding predictors, which suggests that *Pop15* adds the most information, while *DPI* adds little. The code is as following:

```
julia> avPlot(lm2, :Pop15)
julia> avPlot(lm2, :Pop75)
julia> avPlot(lm2, :DPI)
```

Figure 3.4 is generated by R function avPlots from the package **car**.

```
R> avPlots(lm1)
```

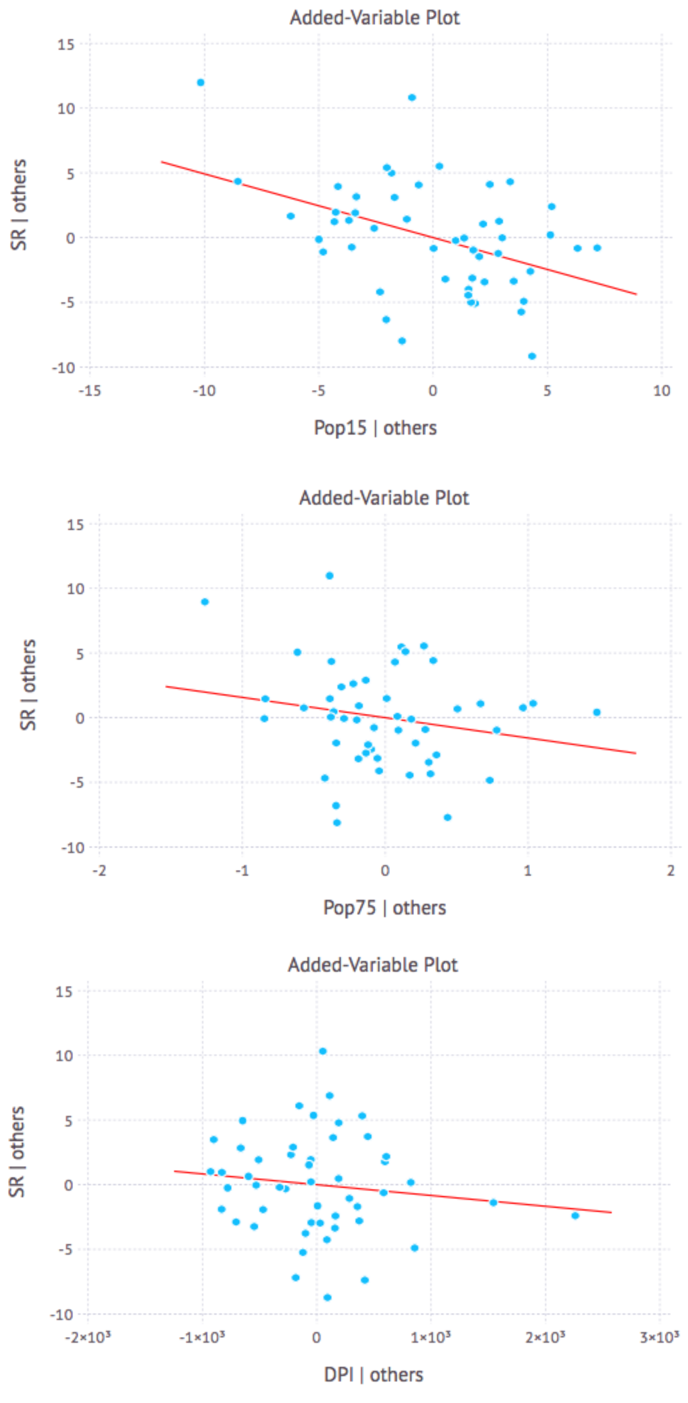

Figure 3.3: Added-variable plots generated by Julia function avplot: Pop15, Pop75, DPI

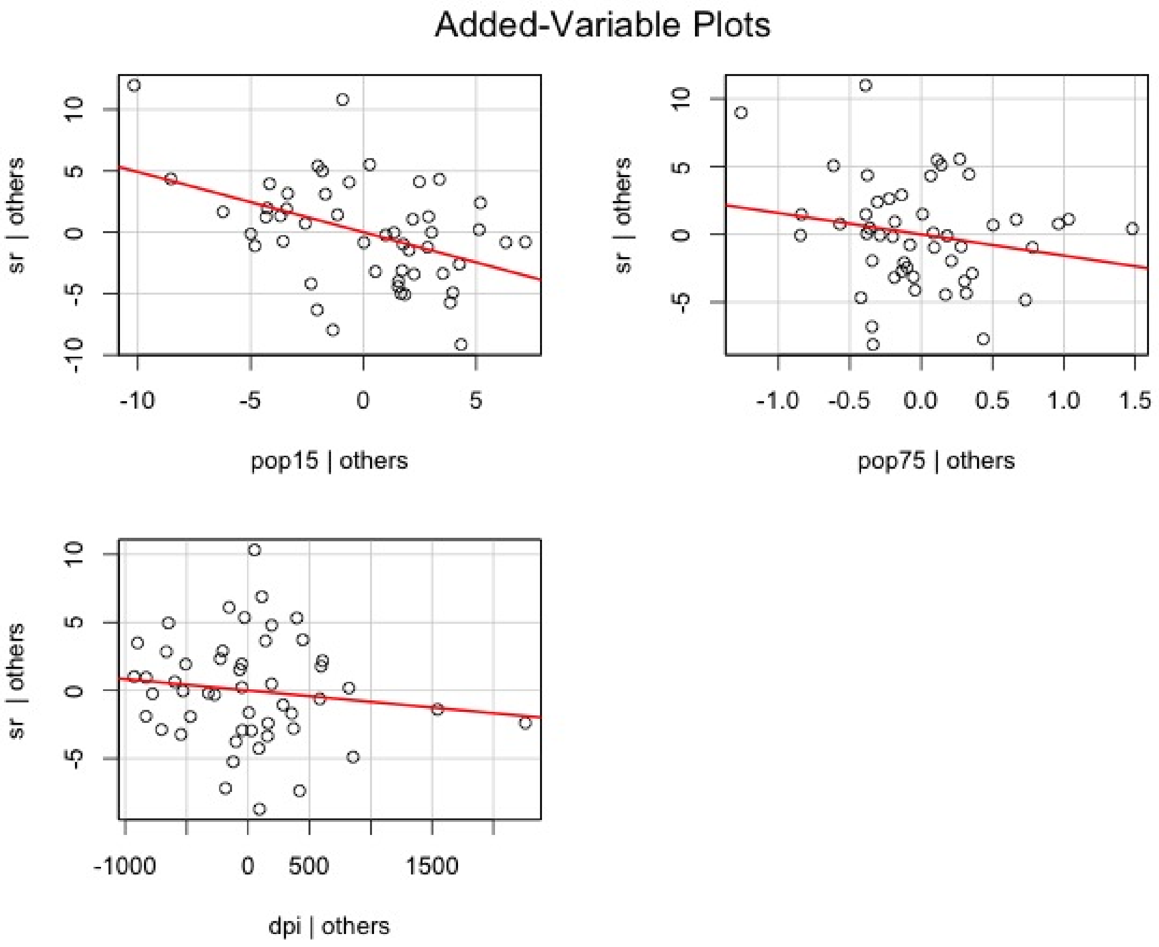

Figure 3.4: Added-variable plots generated by R

3.8 Function vif for Variance Inflation Factor (VIF)

As stated previously, variance inflation factor tests multicollinearity in a model. Function vif accepts a linear regression object, return a VIF for each predictor in the model. In the example, the VIFs of *Pop15* and *Pop75* are both greater than 5. We may consider remove one of them from the model.

```
julia> vif(lm2)

3x2 DataFrame
| Row | variable | vif      |
|-----|----------|---------|
| 1   | "Pop15"  | 5.87532 |
| 2   | "Pop75"  | 6.60925 |
| 3   | "DPI"    | 2.6961  |
```

Below is the output of VIF function in R.

```
vif(lml)
```

```
   pop15    pop75      dpi
5.875320 6.609254 2.696100
```

CHAPTER 4

Conclusions and Thoughts for Future Development

4.1 Conclusions

The Julia package **RegTools** aims at providing various tools for regression modeling, including measures of goodness of fit, outlier detection, variable and model selection, failure of underlying assumption detection, which are familiar to R users.

Through the LifeCycleSavings example, we see that Julia package **RegTools** gives similar result as R with a faster speed, which is one of the major advantages Julia holds. In addition, **RegTools** covers some methods that are not included in common R packages, such as half-normal plots and jackknife residuals.

4.2 Future Development

So far, still some widely used tools for regression modeling have not been included in **RegTools**. I will include more outlier detection and model diagnostics functions like ANOVA for model selection, Park test for heteroscedasticity, Durbin-Watson test for serial correlations, etc.

CHAPTER 5

Appendix

Some key portion of the code are attached here.

```
abstract VarSel


type add1 <: VarSel
    aic::Float64
    add::String
    model::RegressionModel
end


type drop1 <: VarSel
    aic::Float64
    drop::String
    model::RegressionModel
end


function add1(dfrm::RegressionModel, scope::String, data::DataFrame)
    # e.g. scope = "X1+X2+X1&X2"
    replace(scope, " ", "") # remove whitespace in scope
    scope = split(scope, "+")
    index = find(notnull, scope)
    scope = scope[index]
```

```
lhs = dfrm.mf.terms.eterms[1] # response
rhs = extractrhs(dfrm)

add = ""
aic = AIC(dfrm)
model = dfrm
for var in scope
    if var in rhs.rhsarray
        continue # if var already in model, try next var
    end
    rhsnew = rhs.rhsstring * "+" * var
    rhsnew = rhsnew[2:end]
    rhsnew = parse(rhsnew)
    fnew = Formula(lhs, rhsnew)
    newfit = fit(LinearModel, fnew, data)
    newaic = AIC(newfit)
    if newaic < aic
        aic = newaic
        add = var
        f = fnew
        model = newfit
    end
end
if add == ""
    println("No term added")
    return add1(aic, "N/A", dfrm)
else
    println("Add $add with AIC = $aic")
    return add1(aic, add, model)
    end
end



function drop1(dfrm::RegressionModel, scope::String)
    # e.g. scope = "X1+X2+X1&X2"
    replace(scope, " ", "") # remove whitespace in scope
    scope = split(scope, "+")
    index = find(notnull, scope)
    scope = scope[index]

    lhs = dfrm.mf.terms.eterms[1] # response
    rhs = extractrhs(dfrm) # extract rhs

    drop = ""
    aic = AIC(dfrm)
    model = dfrm
    for var in scope
        if var in rhs.rhsarray
            var = "+"*var
            rhsnew = replace(rhs.rhsstring, var, "")
        else
            error("$var is not in the original model,
            please modify scope")
        end
        rhsnew = rhsnew[2:end]
        rhsnew = parse(rhsnew)
        fnew = Formula(lhs, rhsnew)
        newfit = fit(LinearModel, fnew, dfrm.mf.df)
        newaic = AIC(newfit)
        if newaic < aic
            aic = newaic
            drop = var
            f = fnew
            model = newfit
        end
    end
    if drop == ""
        println("No term dropped")
        return drop1(aic, "N/A", dfrm)
    else
        println("Drop $(drop[2:end]) with AIC = $aic")
        return drop1(aic, drop, model)
    end
end

drop1(dfrm::RegressionModel) = drop1(dfrm, extractrhs(dfrm).rhsstring)



type step <: VarSel
    aic::Float64
    model::RegressionModel
end

function step(dfrm::RegressionModel, scope::String, data::DataFrame,
              direction::String, trace::Bool=false)
    directions = ["backward", "forward", "both"]
    if ~ (direction in directions)
        error("direction must be one of ’backward’, ’forward’,
        or ’both’")
    end
    # turn scope to a string array
    scope = split(scope, "+")
    index = find(notnull, scope)
    scope = scope[index]
    # initialize variables
    todrop = ""
    aic = AIC(dfrm)
    model = dfrm
    if direction == "backward"
        drop = drop1(dfrm, scope)
        while drop.drop != ""
            scope = replace(scope, drop.drop, "")
            drop = drop1(drop.model, scope)
        end
        if drop == ""
            println("No term dropped")
            return step(drop.aic, dfrm)
        else
            return step(drop.aic, model)
        end

    elseif direction == "forward"
        add = add1(dfrm, scope, data)
        while add.add != ""
            scope = replace(scope, add.add, "")
            add = add1(add.model, scope, data)
        end
        if add == ""
            println("No term dropped")
            return step(add.aic, dfrm)
        else
            return step(add.aic, model)
        end

    else # both
        drop = drop1(dfrm, scope)
        add = add1(dfrm, scope)
        while drop.drop != "" | add.add != ""
            if drop.aic < add.aic
                model = drop.model
                aic = drop.aic
                scopenew = replace(scope, drop.drop, "")
            else
                model = add.model
                aic = add.aic
                scopenew = replace(scope, add.add, "")
            end
            drop = drop1(model, scopenew)
            add = add1(model, scopenew)
        end
        if scopenew == scope
            println("No term changed")
            return step(AIC(dfrm), dfrm)
        else
            return step(aic, model)
        end
    end
end

function step(dfrm::RegressionModel, direction::String;
     trace::Bool=false)
    directions = ["backward", "both"]
    if ˜ (direction in directions)
        error("If no scope specified, direction must be ’backward’
        or ’both’")
    end
    todrop = ""
    aic = AIC(dfrm)
    model = dfrm
    if direction == "backward"
        drop = drop1(dfrm)
        while drop.aic < aic
            aic = drop.aic
            todrop = drop.drop
            model = drop.model
            drop = drop1(model)
        end
        if todrop == ""
            println("No term dropped")
            return step(aic, dfrm)
        else
            return step(aic, model)
        end
    else # both
        scope = extractrhs(dfrm).rhsstring
        data = dfrm.mf.df
        toadd = ""
        drop = drop1(dfrm)
        add = add1(dfrm, scope, data)
        while drop.aic < aic || add.aic < aic
            if drop.aic < add.aic
                aic = drop.aic
                todrop = drop.drop
                model = drop.model
            else
                aic = add.aic
                toadd = add.add
                model = add.model
            end
            drop = drop1(model)
            add = add1(model, scope, data)
        end
        if todrop == "" && toadd == ""
            println("No term changed")
            return step(AIC(dfrm), dfrm)
        else
            return step(aic, model)
        end
    end
end
```

```
function avPlot(dfrm::RegressionModel, variable::Symbol)
    varNames = names(dfrm.mf.df)
    # get e_yx
    lhs = dfrm.mf.terms.eterms[1]
    rhs = extractrhs(dfrm)
    rhsnew = replace(rhs.rhsstring, "+"*string(variable), "")
    rhsnew = rhsnew[2:end]
    rhsnew = parse(rhsnew)
    fnew = Formula(lhs, rhsnew)
    newfit = fit(LinearModel, fnew, dfrm.mf.df)
    e_yx = residuals(newfit)
    # get e_zx
    X = dfrm.mf.df[2:end]
    Z = X[variable]
    X = X[~[(name in [variable]) for name in names(X)]]
    intercept = ones(Int, nrow(X), 1)
    X = convert(Array, X)
    X = hcat(intercept, X)
    H = *(*(X, inv(*(transpose(X), X))), transpose(X))
    e_zx = *((eye(size(H)[1])-H), Z)
    df = DataFrame(e_yx=e_yx, e_zx=e_zx)
    # get alpha
    avlm = fit(LinearModel, e_yx~0+e_zx, df)
    alpha = coef(avlm)
    # plot
    xs_range = abs(maximum(df[:e_zx]) - minimum(df[:e_zx]))
    xs = linspace(minimum(df[:e_zx])-xs_range/10,
      maximum(df[:e_zx])+xs_range/10)
    ys = alpha .* xs
    xy = DataFrame(xs=xs, ys=ys)
    plot(layer(df, x="e_zx", y="e_yx", Geom.point),
         Guide.xlabel(string(variable)*" | others"),
         Guide.ylabel(string(varNames[1])*" | others"),
         Guide.title("Added-Variable Plot"),
         layer(xy, x="xs", y="ys", Theme(default_color=color("red")),
                 Geom.line))
end

function cooksdistance(dfrm::RegressionModel; plotit::Bool=false)
    r = rstudent(dfrm)
    X = dfrm.mm.m
    H = *(*(X, inv(*(transpose(X), X))), transpose(X))
    h = diag(H)
    p = size(dfrm.mm.m, 2)
    d = (r.^2 .* h) ./ (p*(1-h))
    if !plotit
        return d
    else
        # identify outliers
        n = length(d)
        cutoff = 4/(n-p)
        labels = 1:n
        outlierlabel = labels[d.>cutoff]
        df = DataFrame(n=labels, d=d, label="")
        for i in outlierlabel
            df[:label][i] = string(i)
        end
        plot(df, x="n", y="d", label="label",
            Guide.xlabel("Case number"),
            Guide.ylabel("Cook’s distance"),
    Geom.bar, Geom.label)
    end
end

function vif(dfrm::RegressionModel)
    X = dfrm.mf.df[2:end]
    rhs = extractrhs(dfrm)
    result = DataFrame(variable=rhs.rhsarray, vif=0.0)
    i = 1
    for (var in rhs.rhsarray)
        lhs = parse(var)
        rhsnew = replace(rhs.rhsstring, "+"*var, "")
        rhsnew = rhsnew[2:end]
        rhsnew = parse(rhsnew)
        fnew = Formula(lhs, rhsnew)
        newfit = fit(LinearModel, fnew, X)
        r2 = rsquared(newfit)
        result[:vif][i] = 1 / (1-r2)
        i = i + 1
    end
    result
end


function rsquared(dfrm::RegressionModel)
    SStot = sum((dfrm.model.rr.y - mean(dfrm.model.rr.y)).^2)
    SSres = sum((dfrm.model.rr.y - dfrm.model.rr.mu).^2)
    return (1-(SSres/SStot))
end



function adjrsquared(dfrm::RegressionModel)
    SStot = sum((dfrm.model.rr.y - mean(dfrm.model.rr.y)).^2)
    SSres = sum((dfrm.model.rr.y - dfrm.model.rr.mu).^2)
    n = size(dfrm.model.rr.y, 1)  #number of samples
    if dfrm.mf.terms.intercept
        p = size(dfrm.mm.m, 2) - 1
    else
        p = size(dfrm.mm.m, 2)
    end
    return 1- ( (SSres/(n-p-1)) / (SStot/(n-1)) )
end




function rstudent(dfrm::RegressionModel)
    SSres = sum((dfrm.model.rr.y - dfrm.model.rr.mu).^2)
    n = size(dfrm.model.rr.y, 1)  #number of samples
    if dfrm.mf.terms.intercept
        p = size(dfrm.mm.m, 2) - 1
    else
        p = size(dfrm.mm.m, 2)
    end
    sigma2 = SSres / (n-p)
    X = dfrm.mm.m
    H = *(*(X, inv(*(transpose(X), X))), transpose(X))
    h = diag(H)
    r = residuals(dfrm) ./ (sqrt(sigma2 .* (1 - h)))
    return r
end

function jackknife(dfrm::RegressionModel)
    r = rstudent(dfrm)
    n = size(dfrm.model.rr.y, 1)  #number of samples
    if dfrm.mf.terms.intercept
        p = size(dfrm.mm.m, 2) - 1
    else
        p = size(dfrm.mm.m, 2)
    end
    t = r .* sqrt((n-p-1) ./ (n-p-r.^2))
    return t
end


function halfnorm(dfrm::RegressionModel)
    N = size(dfrm.model.rr.y, 1)
    n = 1:N
    X = dfrm.mm.m
    H = *(*(X, inv(*(transpose(X), X))), transpose(X))
    h = diag(H)
    labels = sortperm(h)
    h = h[labels]
    U = (N+n) / (2*N+1)
    d = Normal()
    u = quantile(d, U)
    # prepare labels for potential outliers
    df = DataFrame(u=u, h=h, label="")
    outlierlabel = labels[u.>2]
    no_outliers = length(outlierlabel)
    for i in (length(labels)-no_outliers+1):length(labels)
        df[:label][i] = string(labels[i])
    end
    plot(df, x="u", y="h", label="label", Geom.point, Geom.label,
         Guide.xlabel("Half-normal quantiles"),
         Guide.ylabel("Leverages"))
end
```

References

[1] Akaike, Hirotugu. 1973. "Information Theory as an Extension of the Maximum Likelihood Principle." Pp. 267-81 in *Second International Symposium on Information Theory*, edited by B. N. Petrov and F. Csaki. Budapest: Akademiai Kiado.

[2] Akaike, Hirotugu. 1985. "Prediction and Entropy." Pp. 1-24 in *A Celebration of Statistics*, edited by Anthony C. Atkinson and Stephen E. Fienberg. New York: Springer-Verlag.

[3] Belsley, D. A., Kuh, E. and R. E. Welsch. 1980. "Regression Diagnostics: Identifying Influential Data and Sources of Collinearity." New York: John Wiley.

[4] Burnham, Kenneth P. and David R. Anderson. 2002. *Model selection and multimodel inference: A practical information-theoretic approach*, New York: Springer-Verlag.

[5] Burnham, Kenneth P. and David R. Anderson. 2004. "Multimodel Inference: Understanding AIC and BIC in Model Selection." *Sociological Methods Research* 33:261-304

[6] Cook, R. Dennis. 1977. "Detection of Influential Observation in Linear Regression." *Techometrics* 19:15-18.

[7] Cook, R. Dennis. 1979. "Influential Observations in Linear Regression." *Journal of the American Statistical Association* 74:169-174.

[8] Faraway, Julian J. 2004. *Linear Models with R*, Taylor & Francis, Ltd.

[9] Fox, John 2002. *An R and S-PLUS Companion to Applied Regression*. Sage, California.

[10] Hastie, Trevor J. and Daryl Pregibon. 1992. "Generalized linear models." Chapter 6 of *Statistical Models in S*, edited by John M. Chambers and Trevor Hastie. Wadsworth & Brooks/Cole.

[11] Hurvich, Clifford M. and Chih-Ling Tsai. 1989. "Regression and Time Series Model Selection in Small Samples." *Biometrika* 76:297-307.

[12] Mosteller, Frederick and John W. Tukey. 1977. "Mathematical justification for added-variable plots." *Data Analysis and Regression: A Second Course in Statistics*, Addison-Wesley Publishing Company

[13] Schwarz, Gideon. 1978. Estimating the Dimension of a Model. *Annals of Statistics* 6:461-464.

[14] Sheath, Simon J. 2009. *A Modern Approach to Regression with R*, New York: Springer.

[15] Sugiura, Nariaki. 1978. "Further Analysis of the Data by Akaike's Information Criterion and the Finite Corrections." *Communications in Statistics, Theory and Methods* A7:13-26.

[16] Velleman, Paul F. and Roy E. Welsch. 1981. "Efficient Computing of Regression Diagnostics." *The American Statistician* 35:234-242 Taylor & Francis, Ltd.