Stata, R与Python：我该选哪个语言

陈强 教授
山东大学经济学院
www.econometrics-stata.com
公众号：计量经济学及Stata应用

Stata, R与Python：为何讲此题目

- 在机器学习与数据科学领域，一直有R与Python之争

- 经管社科领域的学人，通常熟悉Stata，但在大数据时代，面临新的问题：

  (1) 是否应学习一门新的语言？

  (2) 究竟该学R，还是Python?

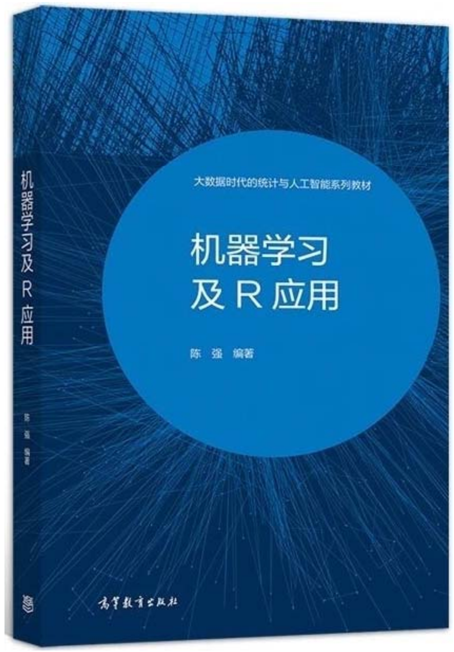
Stata, R与Python：为何由我讲此题目

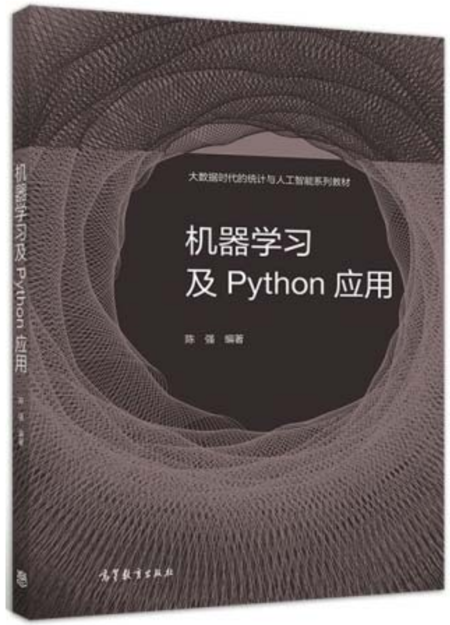
- 经管社科领域的学人，一般熟悉Stata，但很少知道R或Python
- 统计学领域的学人，熟悉R，也知道Python，但较少用Stata
- 计算机领域的学人，熟悉Python，但很少用R或Stata（当然，他们会更多的其他语言）
- 公平比较需要同时熟悉这三种语言

陈强
高教社
2015年
349页

陈强
高教社
2014年
第2版
669页

陈强
高教社
2020年
472页
双色印刷





陈强
高教社
2021年
632页
双色印刷

Stata, R与Python的起源基因

- 1976年，R语言的前身S (Statistics) 语言诞生于John M. Chambers领导的AT&T贝尔实验室统计研究部。1993年，新西兰University of Auckland的统计学家Ross Ihaka与Robert Gentleman为教学目的开发出基于S语言的R语言。

- William Gould(UCLA经济学学士、硕士与博士生)与Sean Becketti (Stanford经济学博士)花了一年时间写代码(底层使用C语言)，于1984年12月在美国经济学年会推出Stata 1.0。

Stata, R与Python的起源基因(续)

- 1989年12月，荷兰人计算机科学家Guido van Rossum为打发圣诞假期而开发的通用语言Python，并于1991年正式发布。

- van Rossum 曾参与ABC语言的开发，但该语言没有流行起来。 他彻底改进了ABC语言，使之开源，并根据其喜欢的英国室内剧Monty Python将其命名为Python。

经济学家的思维

- 在考察经济现象时，一般考虑哪些因素（变量）起作用（包括遗漏变量），以及这些变量之间的相互关系（相关、因果、逆向因果）

- 在多数“文科生”脑中，向量与矩阵可能并未扎根

- Stata的变量就是数据(二维表矩阵)的一列

统计学家的思维

- 统计学家的训练接近于数学家

- 向量与矩阵对于统计学家很自然，故R语言中原生定义了向量(vector)、矩阵(matrix)、数组(array)、列表(list)、数据框(data frame)、函数(function)等“对象”(object)

- *Everything that exists is an object. Everything that happens is a function call.* -- John M. Chambers (解读：R中的所有东西都是对象，R中的所有命令都是函数)

图 2.12　向量、矩阵与三维数组

计算机科学家的思维

- 计算机科学家一般也接受很多数学训练，但专注于计算机，类似于工程师的思维 (需处理的信息类型比统计学家更多样)

- 通用语言Python的原生对象： 数字(number)、字符串(string)、布尔型(Boolean)、列表(list)、元组(tuple)、字典(dictionary)、集合(set)

- Numpy的对象：数组(array)，包含向量(vector)与矩阵(matrix)

- Pandas的对象：序列(series)、数据框(data frame)

面向对象的编程范式

- “面向对象编程”(Object-oriented Programming，简记OOP)：同样一个命令(函数)，作用于不同类型的对象，其效果不同；比如用plot画不同的图

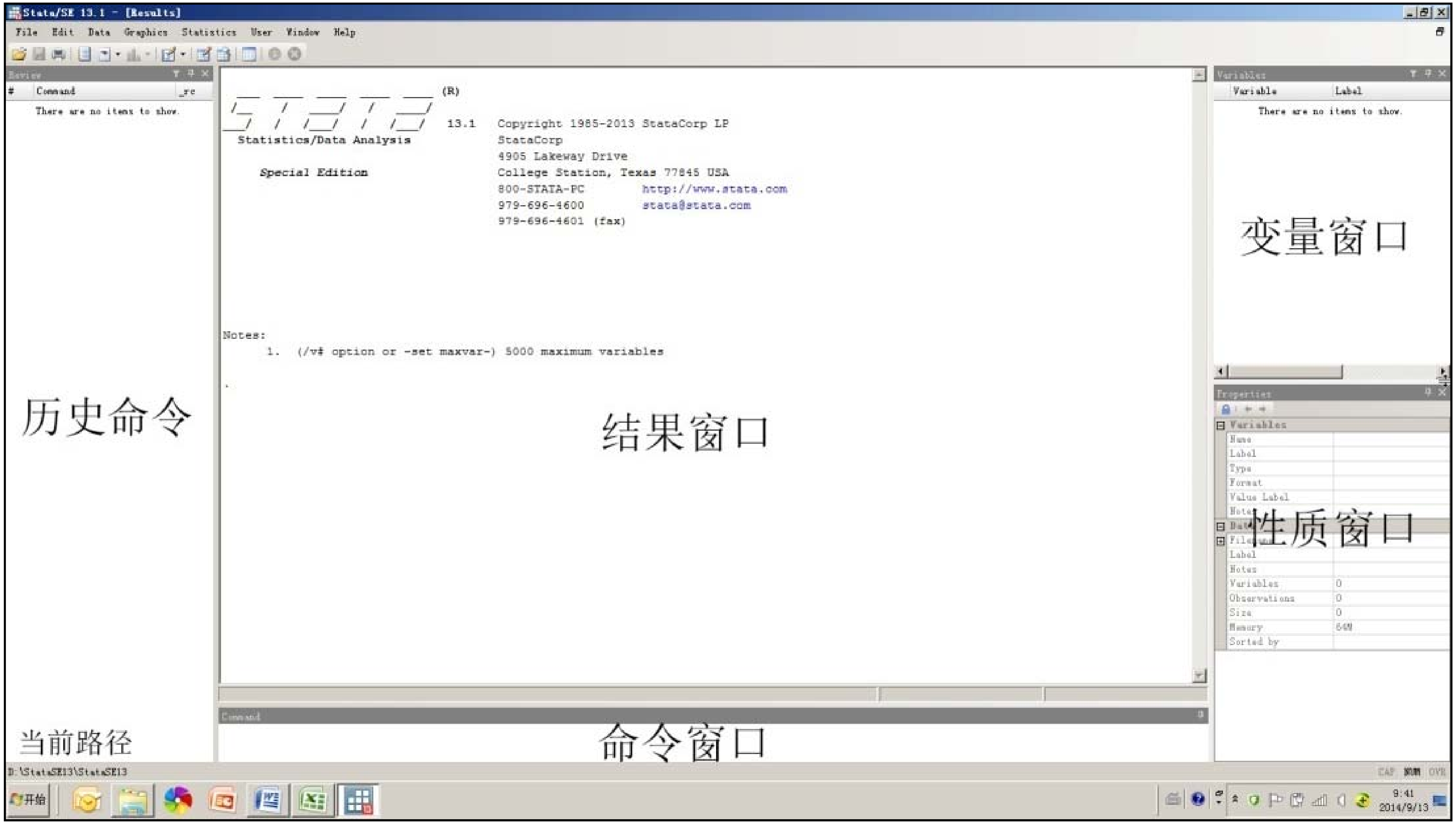
- Stata：无OOP （但mata中有OOP，类似于C或Java，直接支持矩阵编程）

- R：有OOP
- Python：更为彻底的OOP

Stata, R与Python比较

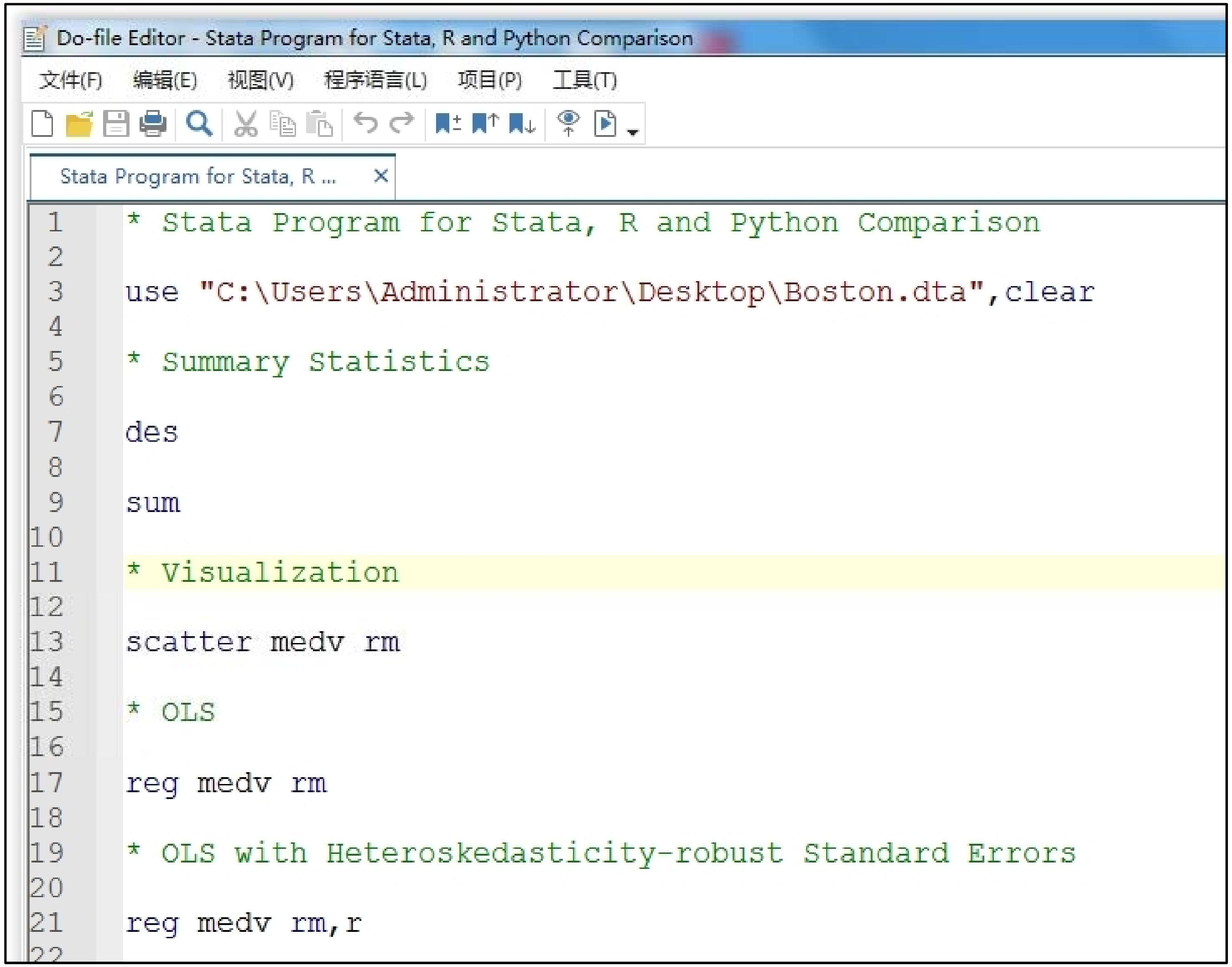
1. 界面
2. 数据管理
3. 描述性统计
4. 画图 / 可视化
5. OLS
6. Lasso
7. Decision Tree
8. Random Forest
9. Neural Network
10. 帮助文件
11. 用户手册
12. 总结

1.1 Stata的界面

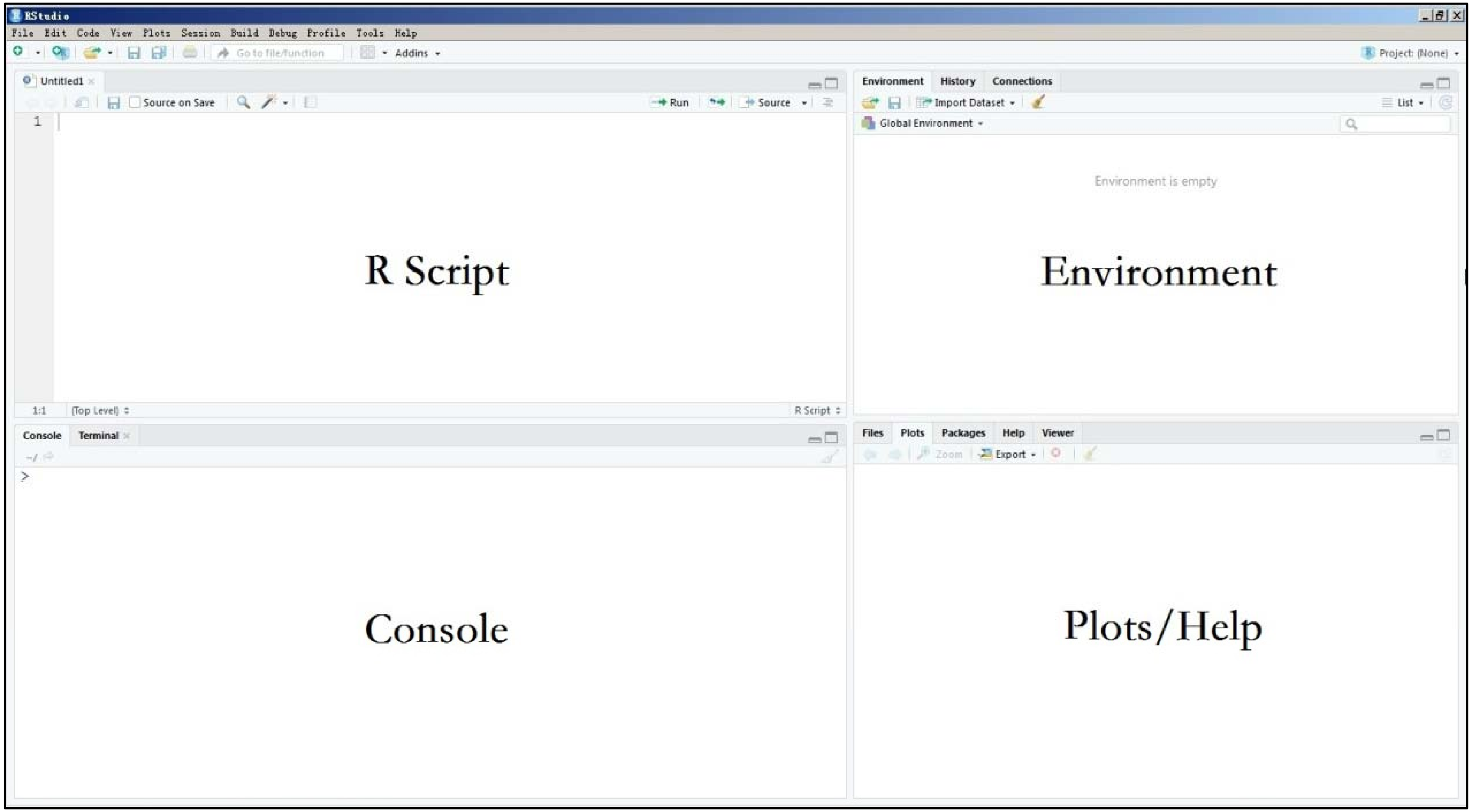


Stata/SE 13.1 - [Results]
File Edit Data Graphics Statistics User Window Help
13.1 Copyright 1985-2013 StataCorp LP
Statistics/Data Analysis
StataCorp
4905 Lakeway Drive
Special Edition
College Station, Texas 77845 USA
800-STATA-PC http://www.stata.com
979-696-4600 stata@stata.com
979-696-4601 (fax)
Notes:
1. (/v# option or -set maxvar-) 5000 maximum variables
历史命令
结果窗口
变量窗口
性质窗口
当前路径
命令窗口
D:\StataSE13\StataSE13

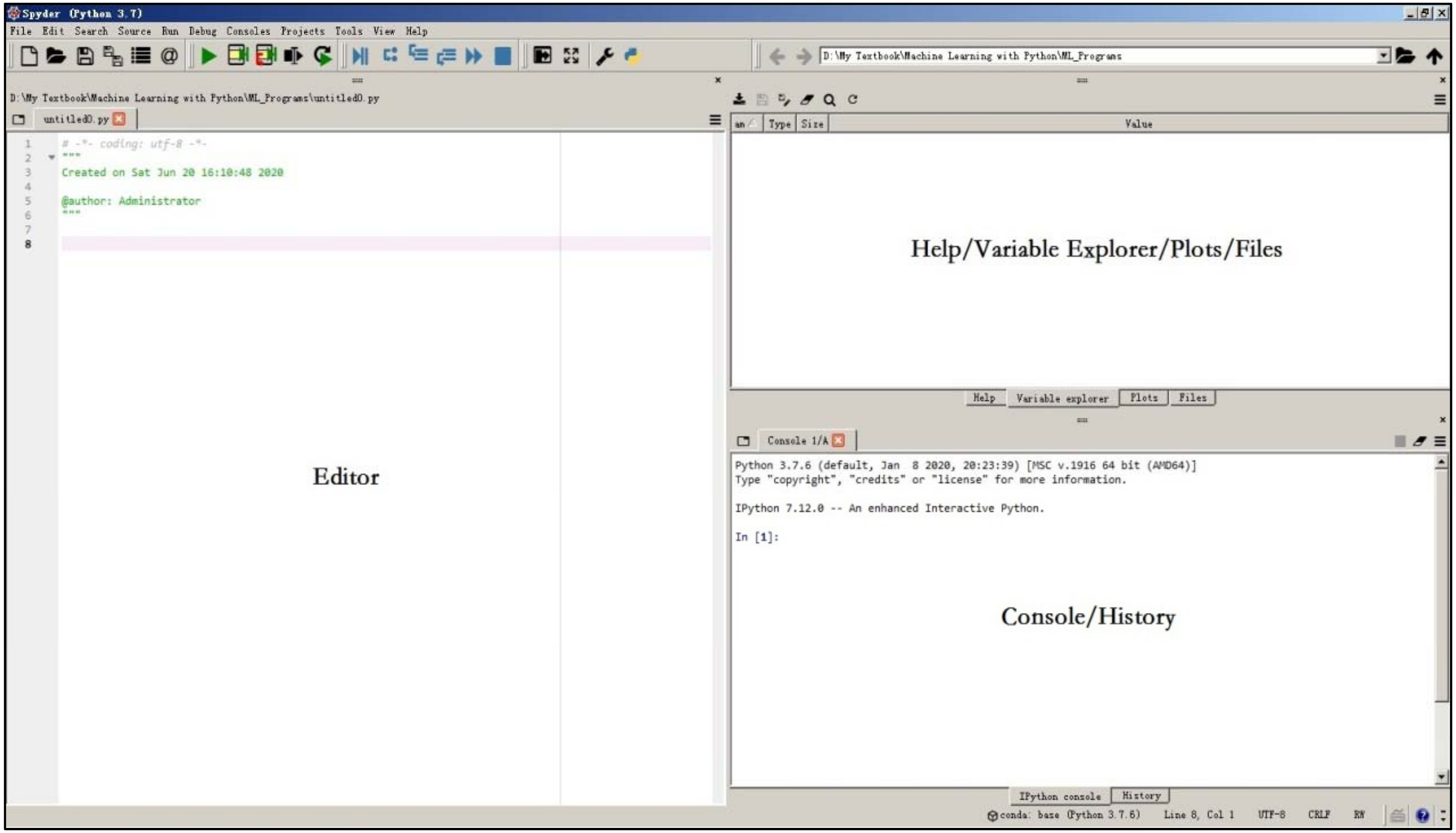
1.1 Stata的界面(续)：Do-file Editor



1.2 RStudio的界面 (R)



1.3 Spyder的界面 (Python)



2.1 Stata的数据管理

- 默认内存中只有1个数据集，特别适合经管社科类的学术研究
- 但业界通常需要同时处理不同来源的多个数据集(框)与“对象”
- 2019年，Stata 16推出“multiple datasets in memory”，但每次仍主要使用“current frame”

2.2 R与Python的数据管理

- R与Python天然地可在内存中存放不同来源的多个数据集(框)与“对象”
- 若调用某个变量，须指定这是哪个数据集的变量
- R：`Boston$medv` (数据框Boston的medv变量)
- Python：`Boston.medv` 或 `Boston[‘medv']`
  (属性, attribute)

案例：Boston Housing Data

- Harrison and Rubinfeld (1978)用此数据研究空气污染对于房价的影响。包含1970年波士顿506个社区的14个变量。响应变量为社区房价中位数medv。

- 特征变量包括平均房间数 rm、房屋年龄 age(1940年前所造房屋的比重)、黑人所占比重的平方 black、低端人口所占百分比 lstat、人均犯罪率 crim、可建25,000平方英尺以上大院的住宅用地比例 zn、非零售商业用地比例 indus、生师比 ptratio (pupil-teacher ratio)、房产税率 tax、是否毗邻查尔斯河 chas、距离波士顿五个就业中心的加权平均距离 dis、高速公路可达性指标 rad、氮氧化物浓度 nox。

3.1 描述性统计-Stata

- summarize

Variable	Obs	Mean	Std. dev.	Min	Max
crim	506	3.613524	8.601545	.00632	88.9762
zn	506	11.36364	23.32245	0	100
indus	506	11.13678	6.860353	.46	27.74
chas	506	.06917	.253994	0	1
nox	506	.5546951	.1158777	.385	.871
rm	506	6.284634	.7026171	3.561	8.78
age	506	68.5749	28.14886	2.9	100
dis	506	3.795043	2.10571	1.1296	12.1265
rad	506	9.549407	8.707259	1	24
tax	506	408.2372	168.5371	187	711
ptratio	506	18.45553	2.164946	12.6	22
black	506	356.674	91.29486	.32	396.9
lstat	506	12.65306	7.141062	1.73	37.97
medv	506	22.53281	9.197104	5	50

3.2 描述性统计-R

> summary(Boston)

```
      crim                zn             indus            chas        
 Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000  
 1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000  
 Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000  
 Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917  
 3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000  
 Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000  
      nox               rm             age              dis        
 Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130  
 1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100  
 Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207  
 Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795  
 3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188  
 Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127  
      rad              tax           ptratio          black       
 Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32  
 1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38  
 Median : 5.000   Median :330.0   Median :19.05   Median :391.44  
 Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67  
 3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23  
 Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90  
     lstat            medv      
 Min.   : 1.73   Min.   : 5.00  
 1st Qu.: 6.95   1st Qu.:17.02  
 Median :11.36   Median :21.20  
 Mean   :12.65   Mean   :22.53  
 3rd Qu.:16.95   3rd Qu.:25.00  
 Max.   :37.97   Max.   :50.00  
```

3.3 描述性统计-Python

```
>>> Boston.describe()
```

```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

Python的“方法”(method)

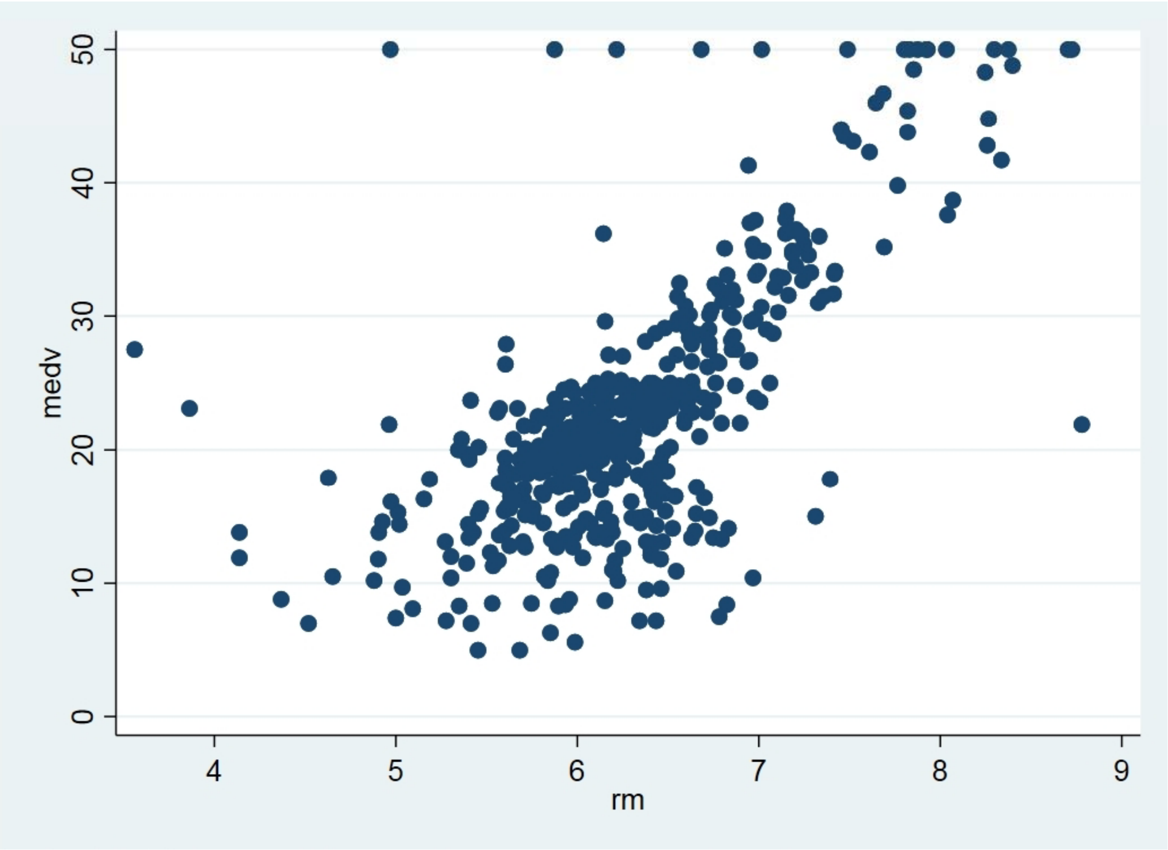
- “`Boston.describe()`”的含义可理解为“`describe(Boston)`”，这种特殊的函数写法称为“方法”(method)，在Python中很常见，便于函数的复合。

- 比如：`h(g(f(x))) versus x.f().g().h()`

- 在R中，只能写成`h(g(f(x)))`，或使用“管道算子”（pipe operator） %>%

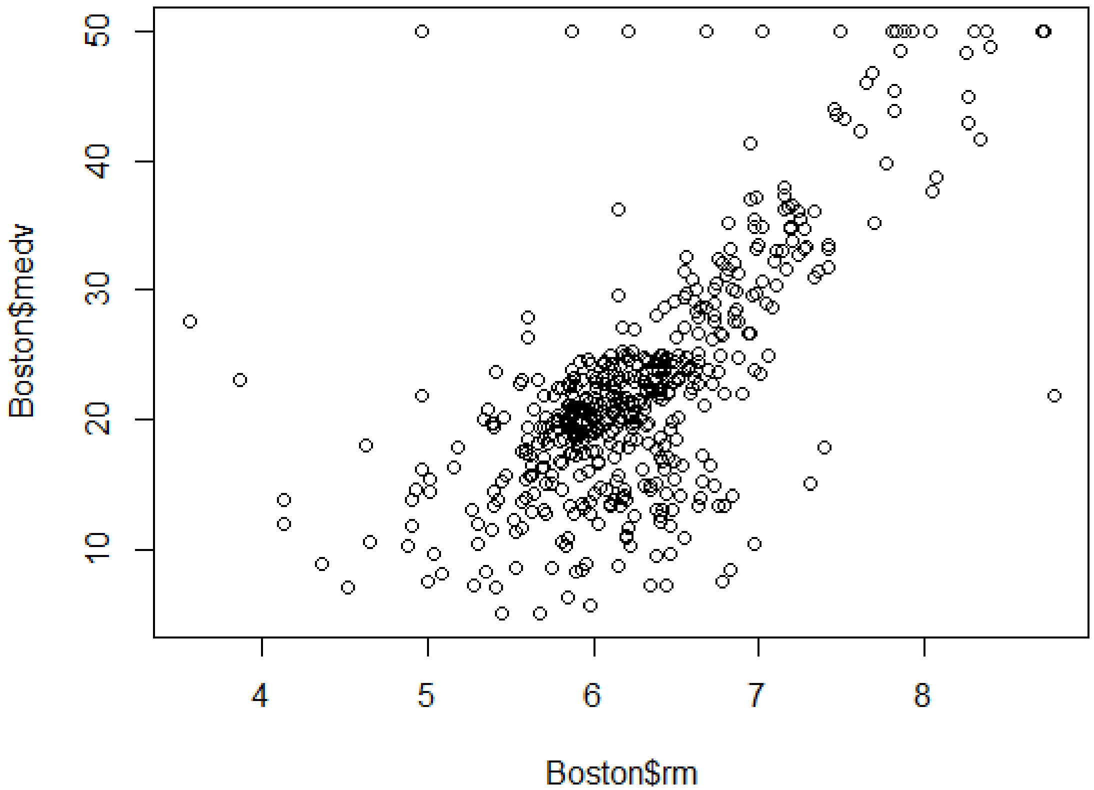
4.1 画图 - Stata

- `scatter medv rm`



4.2 画图 - R

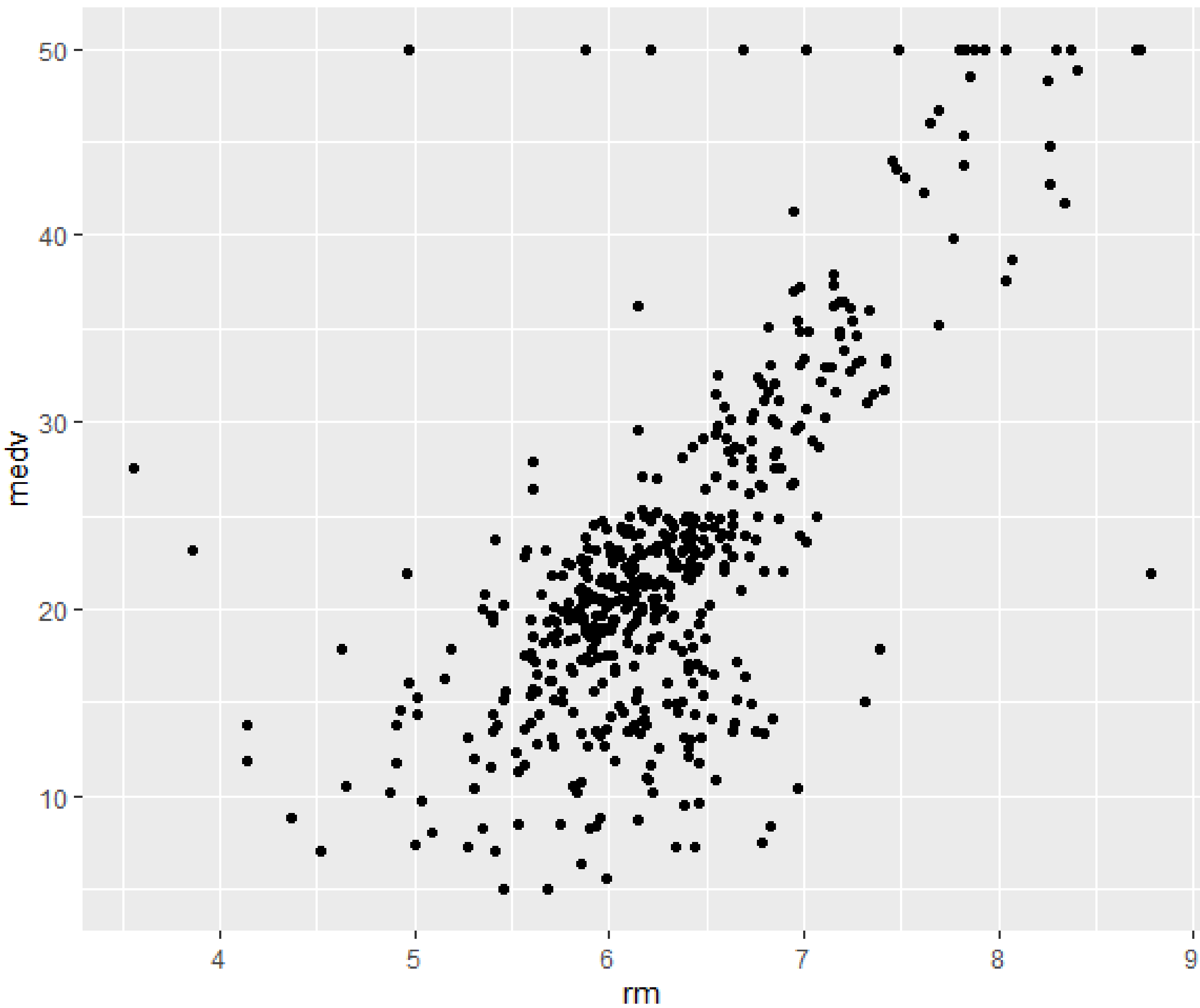
```
> plot(Boston$rm,Boston$medv)
```



4.2 画图 – R via ggplot2

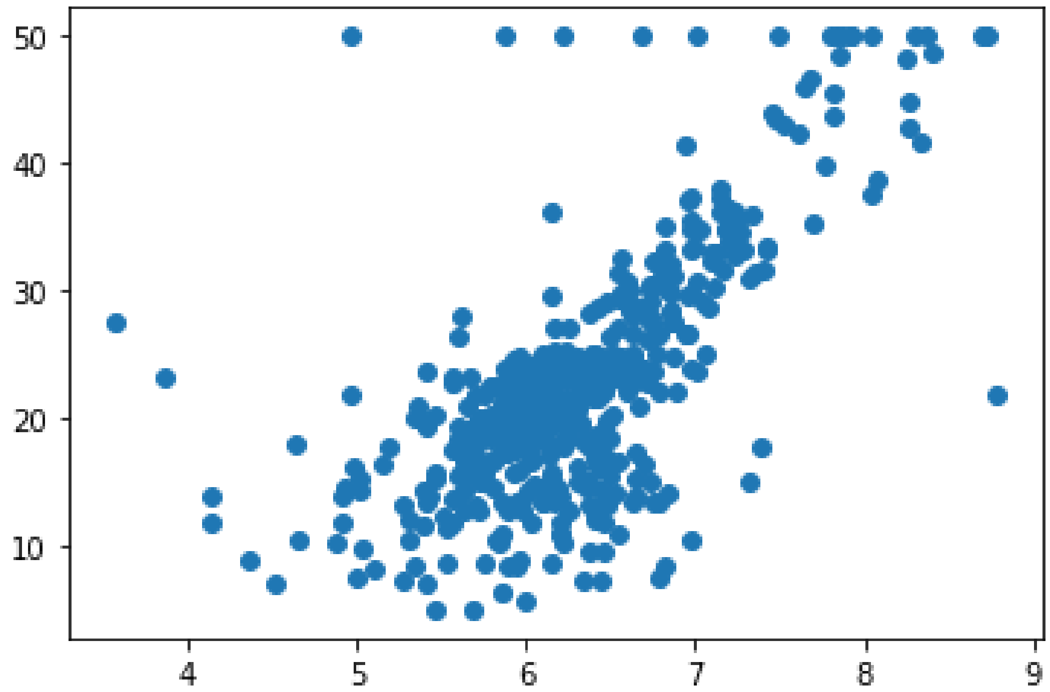
```
> library(ggplot2)
> ggplot(data=Boston) +
      geom_point(mapping = aes(x=rm,
                               y=medv))
```

- 一般认为 ggplot2 (grammar of graphics) 的画图效果最美观 (elegant)，但语句更繁琐


50
40
30
20
10
medv
4
5
6
7
8
9
rm

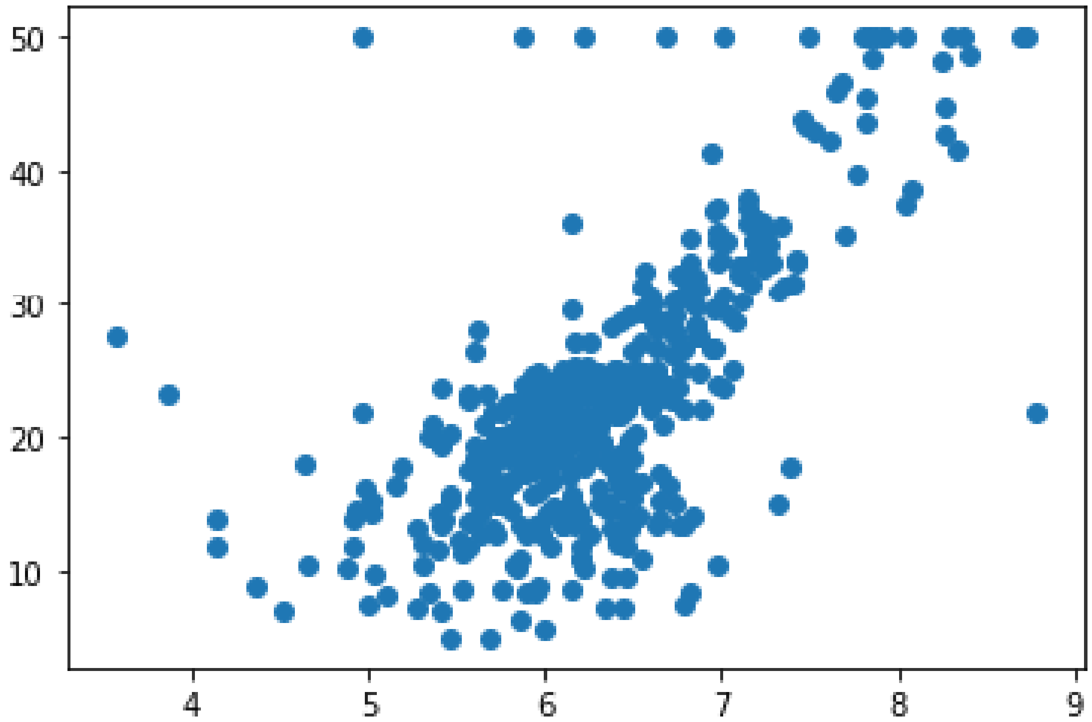
4.3 画图 – Python via Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(Boston.RM,
                Boston.MEDV)
```



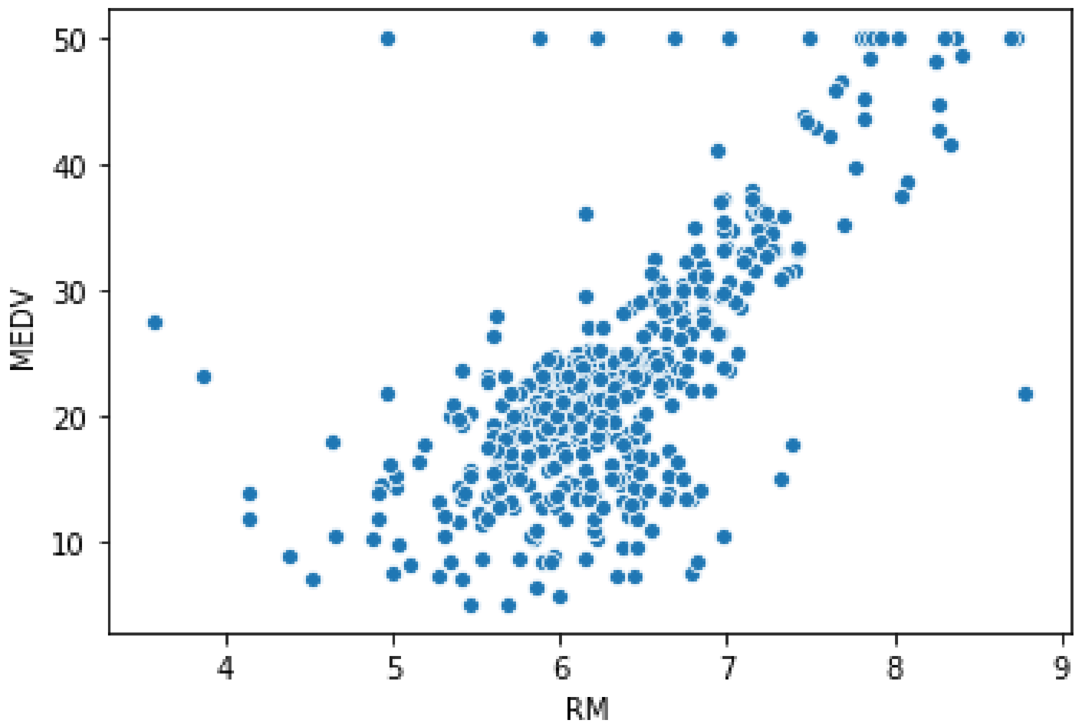
4.3 画图 – Python via Panda

```
>>> import pandas as pd
>>> Boston.plot.scatter(x = 'RM', y =
                       'MEDV')
```



4.3 画图 – Python via seaborn

```
>>> import seaborn as sns
>>> sns.scatterplot(x = 'RM',
                    y = 'MEDV', data=Boston)
```



5.1 OLS回归 - Stata

- `regress medv rm`

```
      Source |       SS           df       MS      Number of obs   =       506
-------------+----------------------------------   F(1, 504)       =    471.85
       Model |  20654.4162         1  20654.4162   Prob > F        =    0.0000
    Residual |  22061.8792       504  43.7735698   R-squared       =    0.4835
-------------+----------------------------------   Adj R-squared   =    0.4825
       Total |  42716.2954       505  84.5867236   Root MSE        =    6.6162

------------------------------------------------------------------------------
        medv | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
          rm |   9.102109   .4190266    21.72   0.000     8.278855    9.925363
       _cons |  -34.67062   2.649803   -13.08   0.000    -39.87664    -29.4646
------------------------------------------------------------------------------
```

5.1 OLS回归 – Stata with robust S.E.

- `reg medv rm,robust`

```
Linear regression                               Number of obs     =        506
                                                F(1, 504)         =     189.53
                                                Prob > F          =     0.0000
                                                R-squared         =     0.4835
                                                Root MSE          =     6.6162
```

medv	Coefficient	Robust std. err.	t	P>|t|	[95% conf.	interval]
rm	9.102109	.6611539	13.77	0.000	7.803152	10.40107
_cons	-34.67062	4.16736	-8.32	0.000	-42.85816	-26.48308

5.2 OLS回归 - R

```
> fit <- lm(medv~rm,data=Boston)
> summary(fit)
```

```
Call:
lm(formula = medv ~ rm, data = Boston)

Residuals:
    Min      1Q  Median      3Q     Max 
-23.346  -2.547   0.090   2.986  39.433 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -34.671      2.650  -13.08   <2e-16 ***
rm             9.102      0.419   21.72   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.616 on 504 degrees of freedom
Multiple R-squared:  0.4835,    Adjusted R-squared:  0.4825 
F-statistic: 471.8 on 1 and 504 DF,  p-value: < 2.2e-16
```

fit: List of 12 (包含12个成分的列表)

Name	Type	Value
fit	list [12] (S3: lm)	List of length 12
coefficients	double [2]	-34.7 9.1
residuals	double [506]	-1.18 -2.17 3.97 4.37 5.82 4.84 ...
effects	double [506]	-506.86 -143.72 4.14 4.52 5.98 4.91 ...
rank	integer [1]	2
fitted.values	double [506]	25.2 23.8 30.7 29.0 30.4 23.9 ...
assign	integer [2]	0 1
qr	list [5] (S3: qr)	List of length 5
df.residual	integer [1]	504
xlevels	list [0]	List of length 0
call	language	lm(formula = medv ~ rm, data = Boston)
terms	formula	medv ~ rm
model	list [506 x 2] (S3: data.frame)	A data.frame with 506 rows and 2 columns

5.2 OLS回归 – R with robust S.E.

```
> library(car)              # companion to
                              applied regression
> library(lmtest)           # package for
                              testing linear model
> fit <- lm(medv~rm,data=Boston)
> coeftest(fit, vcov = hccm )
```

```
t test of coefficients:

             Estimate Std. Error t value  Pr(>|t|)
(Intercept) -34.67062    4.24305 -8.1712 2.482e-15 ***
rm            9.10211    0.67302 13.5243 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

5.2 OLS回归 – R (对所有变量回归)

```
> fit <- lm(medv~.,data=Boston)
> summary(fit)
```

```
Call:
lm(formula = medv ~ ., data = Boston)

Residuals:
    Min      1Q  Median      3Q     Max 
-15.595  -2.730  -0.518   1.777  26.199 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
crim        -1.080e-01  3.286e-02  -3.287 0.001087 ** 
zn           4.642e-02  1.373e-02   3.382 0.000778 ***
indus        2.056e-02  6.150e-02   0.334 0.738288    
chas         2.687e+00  8.616e-01   3.118 0.001925 ** 
nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
age          6.922e-04  1.321e-02   0.052 0.958229    
dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
tax         -1.233e-02  3.760e-03  -3.280 0.001112 ** 
ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
```

5.3 OLS回归 - Python

```
>>> import numpy as np
>>> import pandas as pd
>>> import statsmodels.formula.api
      as smf
>>> model = smf.ols('MEDV ~ RM',
                     data=Boston)
>>> results = model.fit()
>>> results.summary()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.484
Model:                            OLS   Adj. R-squared:                  0.483
Method:                 Least Squares   F-statistic:                     471.8
Date:                Mon, 10 May 2021   Prob (F-statistic):           2.49e-74
Time:                        20:38:22   Log-Likelihood:                -1673.1
No. Observations:                 506   AIC:                             3350.
Df Residuals:                     504   BIC:                             3359.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -34.6706      2.650    -13.084      0.000     -39.877     -29.465
RM             9.1021      0.419     21.722      0.000       8.279       9.925
==============================================================================
Omnibus:                      102.585   Durbin-Watson:                   0.684
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              612.449
Skew:                           0.726   Prob(JB):                    1.02e-133
Kurtosis:                       8.190   Cond. No.                         58.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

5.3 OLS回归 – Python with robust S.E.

```
>>> results_robust =
            model.fit(cov_type='HC1')
>>> results_robust.summary()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.484
Model:                            OLS   Adj. R-squared:                  0.483
Method:                 Least Squares   F-statistic:                     189.5
Date:                Mon, 10 May 2021   Prob (F-statistic):           7.83e-37
Time:                        20:45:20   Log-Likelihood:                -1673.1
No. Observations:                 506   AIC:                             3350.
Df Residuals:                     504   BIC:                             3359.
Df Model:                           1
Covariance Type:                  HC1
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -34.6706      4.167     -8.320      0.000     -42.838     -26.503
RM             9.1021      0.661     13.767      0.000       7.806      10.398
==============================================================================
Omnibus:                      102.585   Durbin-Watson:                   0.684
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              612.449
Skew:                           0.726   Prob(JB):                    1.02e-133
Kurtosis:                       8.190   Cond. No.                         58.4
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC1)
"""
```

6.1 Lasso - Stata

- `lasso linear medv crim zn indus chas nox rm age dis rad tax ptratio black lstat, selection(cv, alllambdas) stop(0) rseed(12345) nolog`

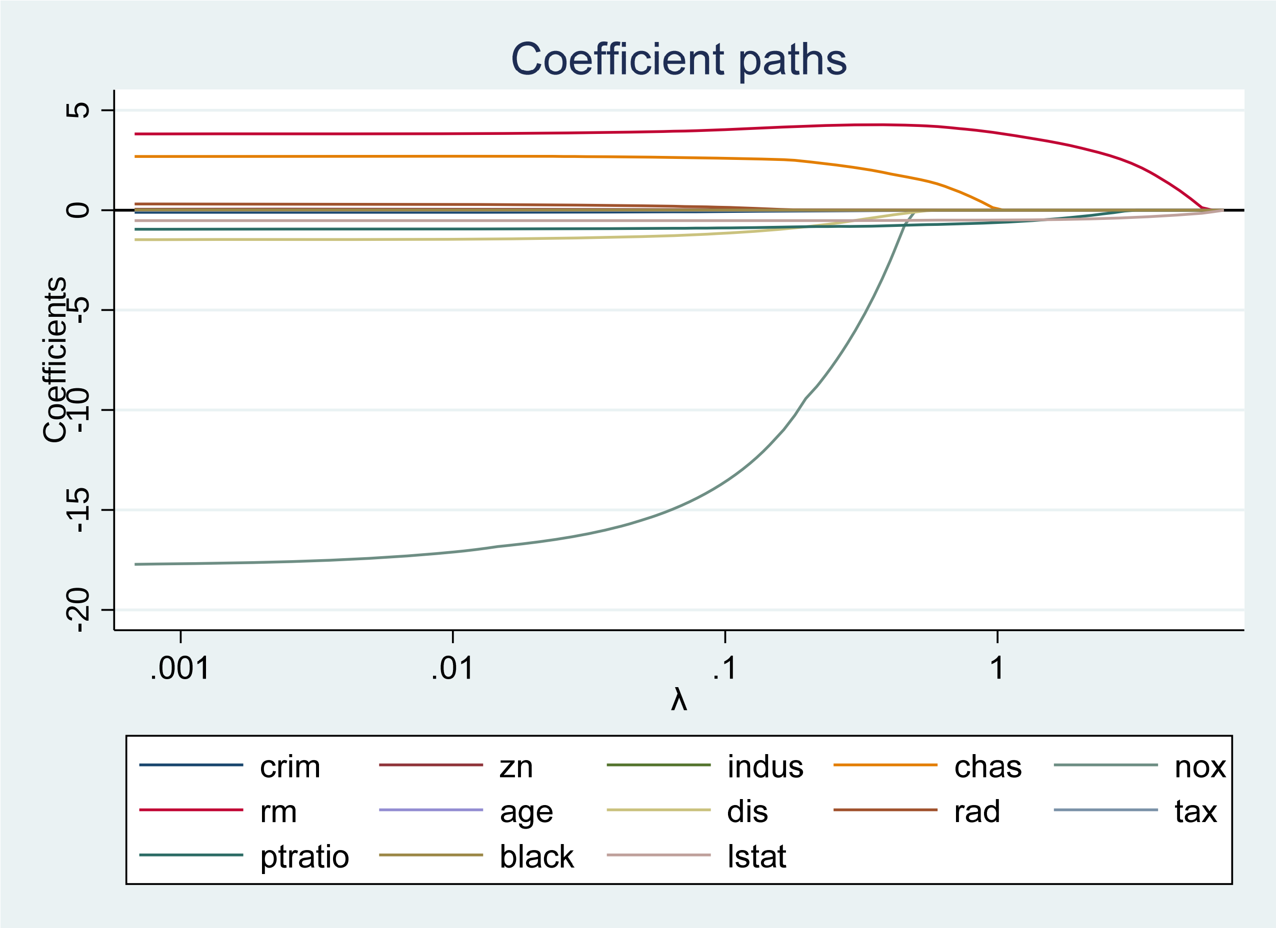
```
Lasso linear model                          No. of obs         =        506
                                            No. of covariates  =         13
Selection: Cross-validation                 No. of CV folds    =         10
```

ID	Description	lambda	No. of nonzero coef.	Out-of-sample R-squared	CV mean prediction error
1	first lambda	6.777654	0	0.0011	84.32675
59	lambda before	.0307358	11	0.7227	23.40918
* 60	selected lambda	.0280053	11	0.7227	23.40868
61	lambda after	.0255174	11	0.7227	23.40892
100	last lambda	.0006778	13	0.7221	23.46378

* lambda selected by cross-validation.

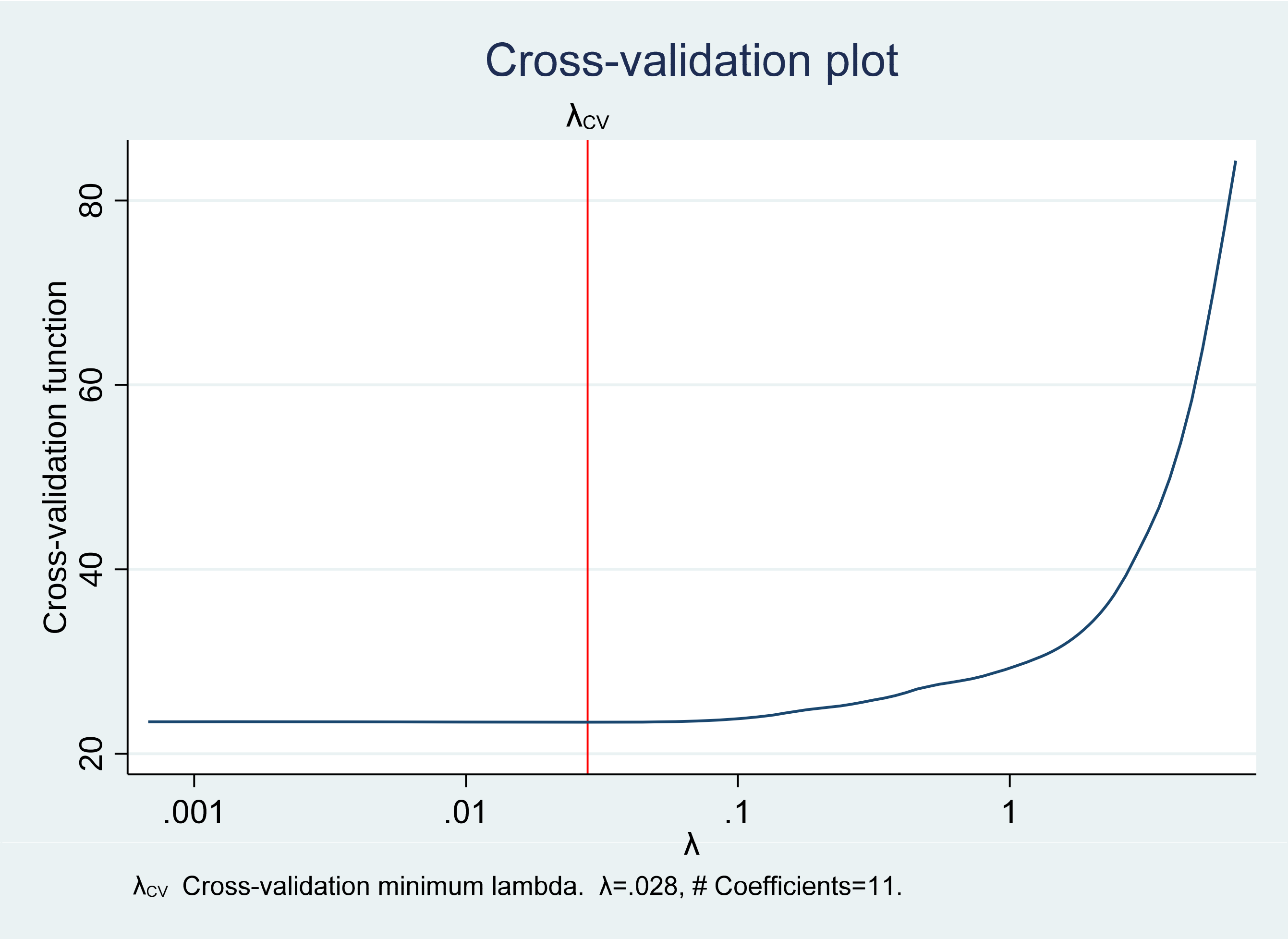
6.1 Lasso coef path – Stata

- `coefpath, legend(on position(6) cols(5)) xunits(lnlambda) rawcoefs`



6.1 Lasso CV plot – Stata

- `cvplot,xunits(lnlambda)`



6.1 Lasso coef – Stata

- `lassocoef, display(coef,penalized) sort(coef,penalized)`

	active
_cons	34.4703
nox	-16.31916
rm	3.864489
chas	2.683076
dis	-1.397161
ptratio	-.930407
lstat	-.5224817
rad	.2542623
crim	-.0986721
zn	.0415227
tax	-.0098895
black	.0090293

```
Legend:
  b - base level
  e - empty cell
  o - omitted
```

注：变量age与indus的Lasso回归系数为0，故未列出！

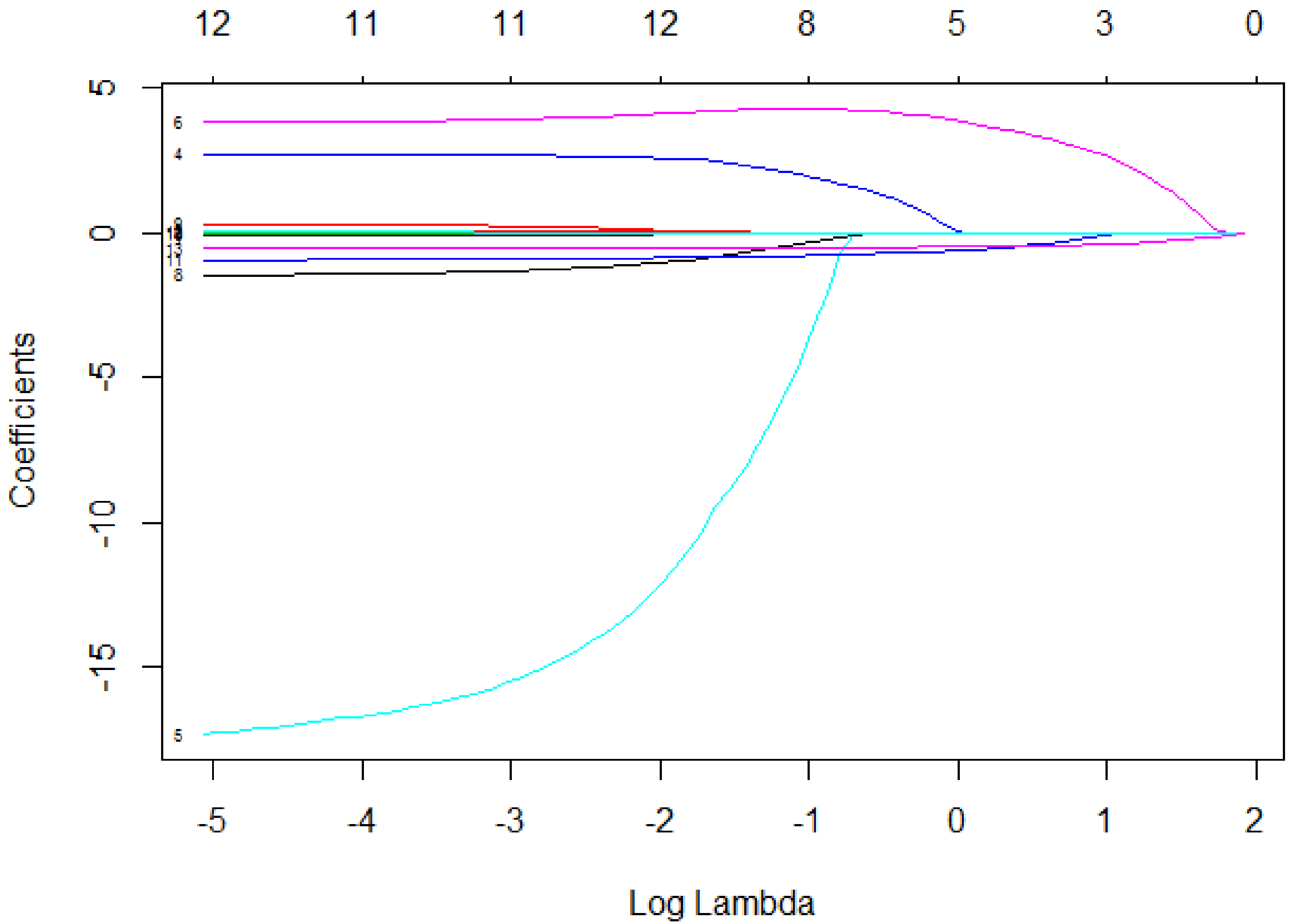
6.2 Lasso – R

```
> library(glmnet)
> x <- model.matrix(Boston$medv~.,
                          Boston[,-14])[,-1]
  # define data matrix w/o constant (去掉第1列)
> y <- Boston$medv

> fit <- glmnet(x,y,alpha=1)   #lasso
> plot(fit,xvar="lambda",label=TRUE)
```

Data (Design) Matrix

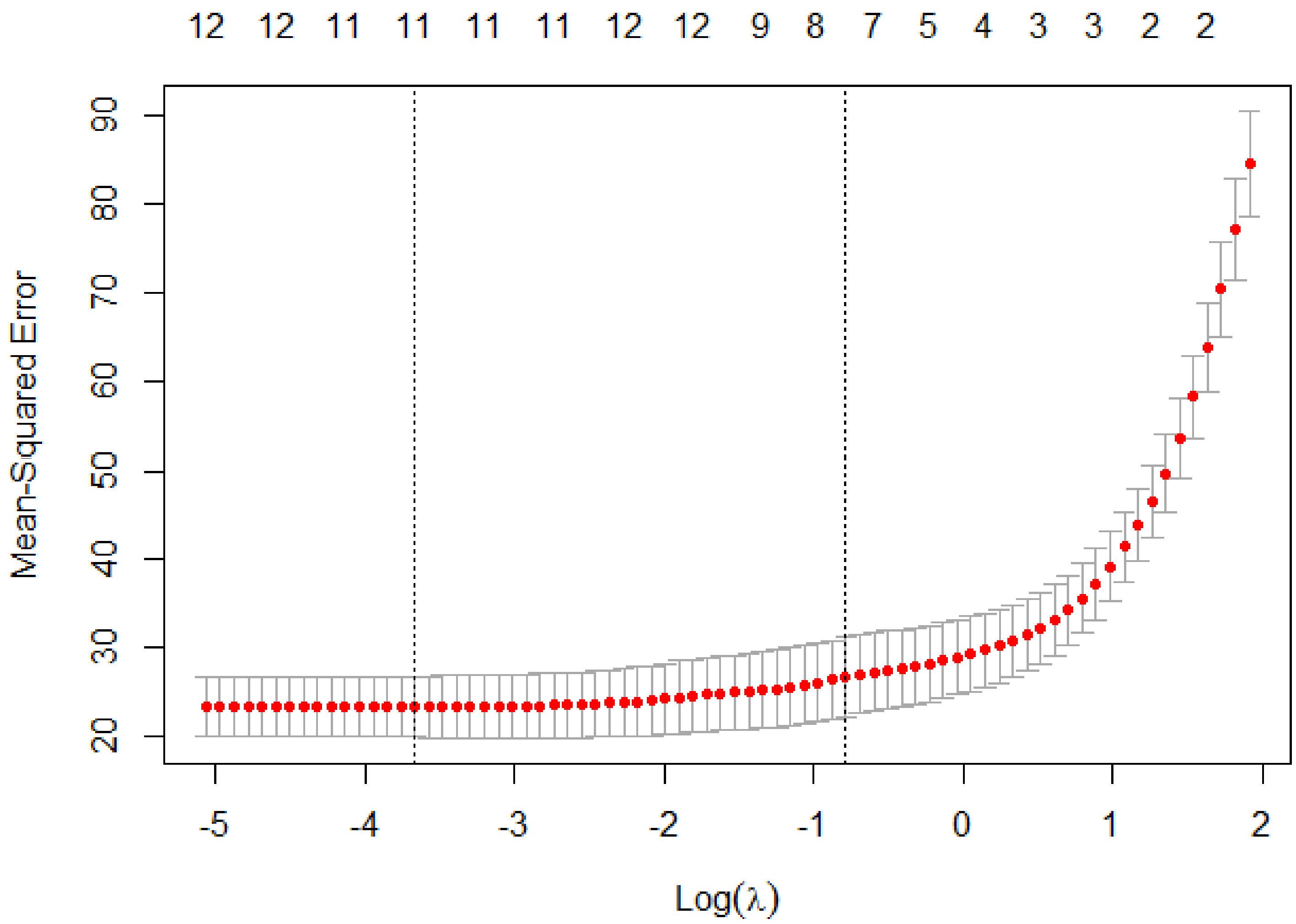
$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1K} \\ x_{21} & x_{22} & \cdots & x_{2K} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nK} \end{pmatrix}$$

行: 个体观测值(observation)；列: 变量(variable)


12
11
11
12
8
5
3
0
5
0
-5
-10
-15
Coefficients
6
4
5
-5
-4
-3
-2
-1
0
1
2
Log Lambda

6.2 Lasso CV plot – R

```
> set.seed(1)
> cvfit <-  cv.glmnet(x,y,alpha=1)
                 # 10-fold CV as default
> cvfit$lambda.min
         # lambda.min that minimizes CV
> [1] 0.02551743

> plot(cvfit)    # 同时显示MSE的正负标准差
```


12 12 11 11 11 11 12 12 9 8 7 5 4 3 3 2 2
Mean-Squared Error
20
30
40
50
60
70
80
90
-5
-4
-3
-2
-1
0
1
2
Log(λ)

6.2 Lasso coef – R

- coef(cvfit, s = "lambda.min")

```
14 x 1 sparse Matrix of class "dgCMatrix"
                         1
(Intercept)  34.594713527
crim         -0.099226869
zn            0.041830020
indus         .
chas          2.688250324
nox         -16.401122000
rm            3.861229965
age           .
dis          -1.404571749
rad           0.256788019
tax          -0.009997514
ptratio      -0.931437290
black         0.009049252
lstat        -0.522505968
```

6.3 Lasso - Python

- `from sklearn.linear_model import Lasso`
- `from sklearn.linear_model import lasso_path`
- `from sklearn.linear_model import LassoCV`
- `from sklearn.preprocessing import StandardScaler`
- `from sklearn.model_selection import Kfold`
- `from sklearn.model_selection import cross_val_score`
- `X_raw = Boston.iloc[:, :-1]`
- `y = Boston.iloc[:, -1]`
- `scaler = StandardScaler()`
- `X = scaler.fit_transform(X_raw)`
- `model = Lasso(alpha=0.2)`
- `model.fit(X, y)`

6.3 Lasso – Python (续)

- `model.coef_`

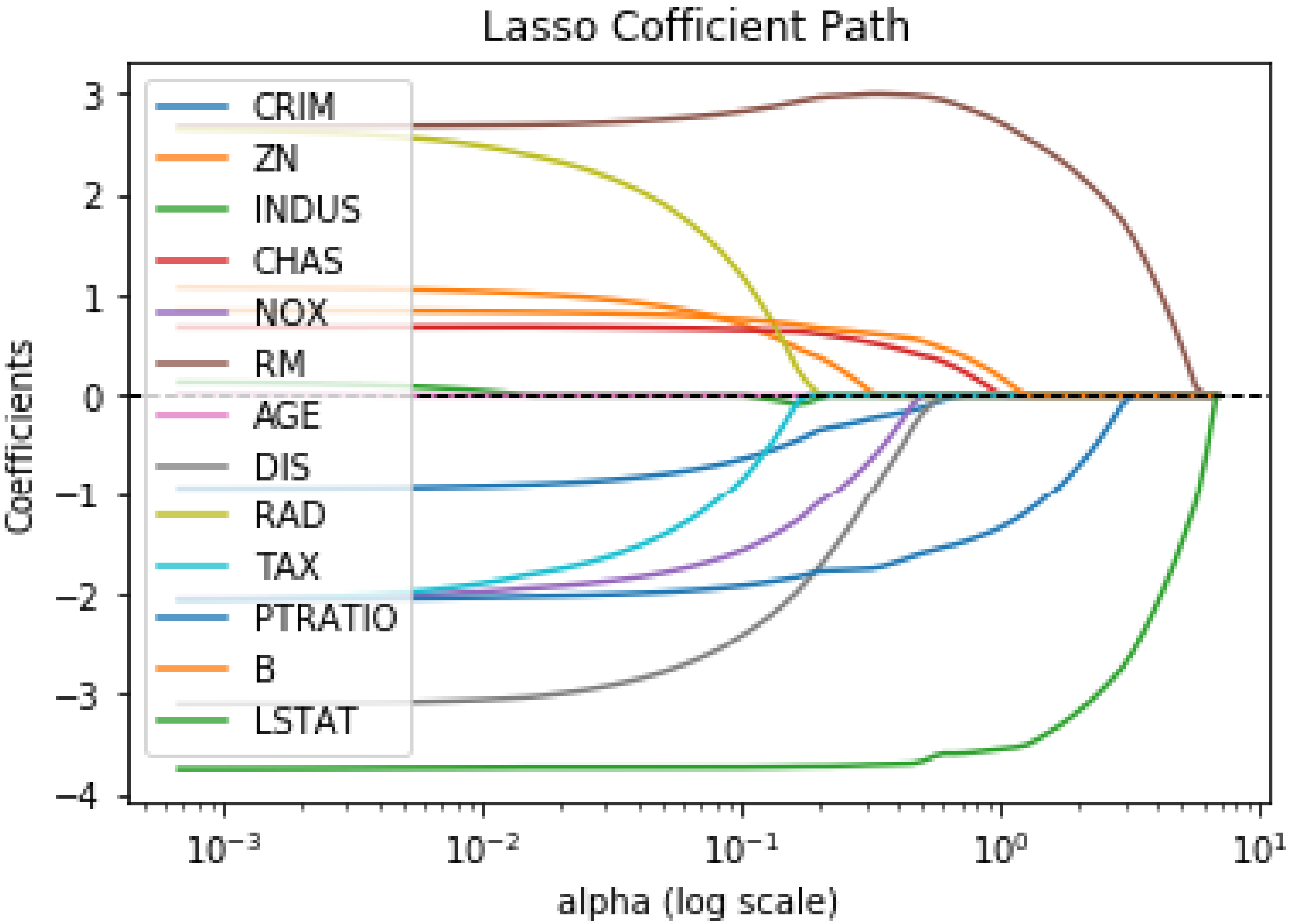
```
array([-0.33904468,  0.37945923, -0.024138  ,  0.61693339, -1.07997374,
        2.96375162, -0.        , -1.73013196,  0.00777185, -0.        ,
       -1.77372345,  0.670867  , -3.71592821])
```

- `pd.DataFrame(model.coef_, index=X_raw.columns, columns=['Coefficient'])`

	Coefficient
CRIM	-0.339045
ZN	0.379459
INDUS	-0.024138
CHAS	0.616933
NOX	-1.079974
RM	2.963752
AGE	-0.000000
DIS	-1.730132
RAD	0.007772
TAX	-0.000000
PTRATIO	-1.773723
B	0.670867
LSTAT	-3.715928

6.3 Lasso coef plot – Python

```
>>> alphas, coefs, _ = lasso_path(X, y,
                                  eps=1e-4)
>>> ax = plt.gca()
>>> ax.plot(alphas, coefs.T)
>>> ax.set_xscale('log')
>>> plt.xlabel('alpha (log scale)')
>>> plt.ylabel('Coefficients')
>>> plt.title('Lasso Cofficient Path')
>>> plt.axhline(0, linestyle='--',
                linewidth=1, color='k')
>>> plt.legend(X_raw.columns)
```


Lasso Coefficient Path
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT
Coefficients
3
2
1
0
−1
−2
−3
−4
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
alpha (log scale)

6.3 Lasso coef – Python

```
>>> kfold = KFold(n_splits=10, shuffle=True,
                random_state=1)
>>> alphas=np.logspace(-3, -1, 100)
>>> model = LassoCV(alphas=alphas, cv=kfold)
>>> model.fit(X, y)
>>> model.alpha_
0.02848035868435802
```

6.3 Lasso coef – Python (续)

```
>>> pd.DataFrame(model.coef_,
        index=X_raw.columns,
        columns=['Coefficient'])
```

```
         Coefficient
CRIM       -0.846146
ZN          0.965785
INDUS      -0.000000
CHAS        0.680701
NOX        -1.886944
RM          2.713469
AGE        -0.000000
DIS        -2.935723
RAD         2.203538
TAX        -1.658672
PTRATIO    -2.011514
B           0.823063
LSTAT      -3.727417
```

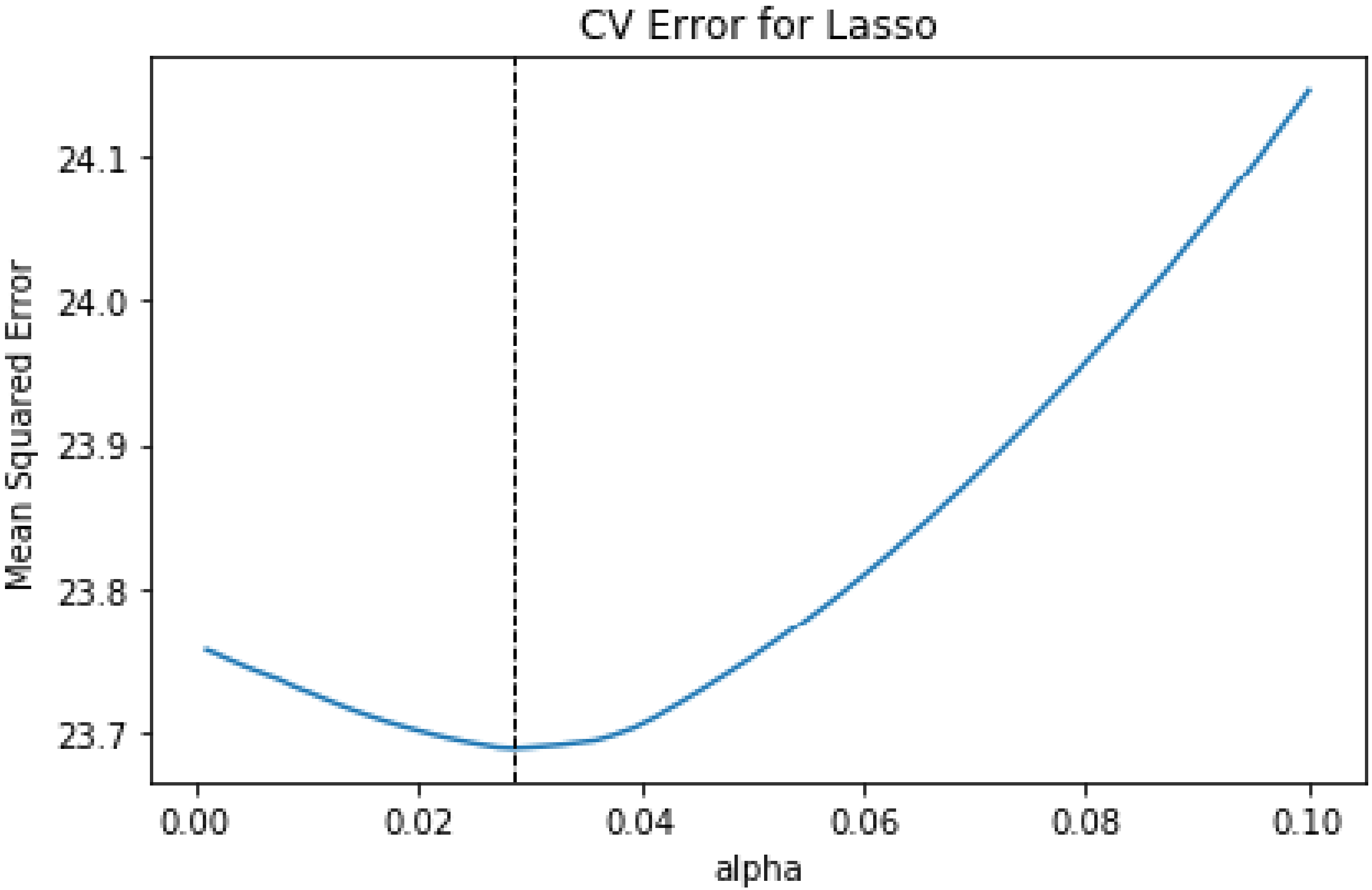
6.3 Lasso CV plot – Python

```
>>> mse = np.mean(model.mse_path ,
                  axis=1)
>>> index_min = np.argmin(mse)
>>> alphas[index_min]
0.003511191734215131
```

- 备注：model.mse_path_可能有bug，故需要手工画交叉验证图

6.3 Lasso CV plot – Python(续)

```
>>> alphas = np.logspace(-3, -1, 100)
>>> scores = []
>>> for alpha in alphas:
        model = Lasso(alpha=alpha)
        kfold = KFold(n_splits=10, shuffle=True, random_state=1)
        scores_val = -cross_val_score(model, X, y, cv=kfold,
        scoring='neg_mean_squared_error')
        score = np.mean(scores_val)
        scores.append(score)
>>> mse = np.array(scores)
>>> index_min = np.argmin(mse)
>>> alphas[index_min]

>>> plt.plot(alphas, mse)
>>> plt.axvline(alphas[index_min], linestyle='--', linewidth=1, color='k')
>>> plt.xlabel('alpha')
>>> plt.ylabel('Mean Squared Error')
>>> plt.title('CV Error for Lasso')
>>> plt.tight_layout()
```

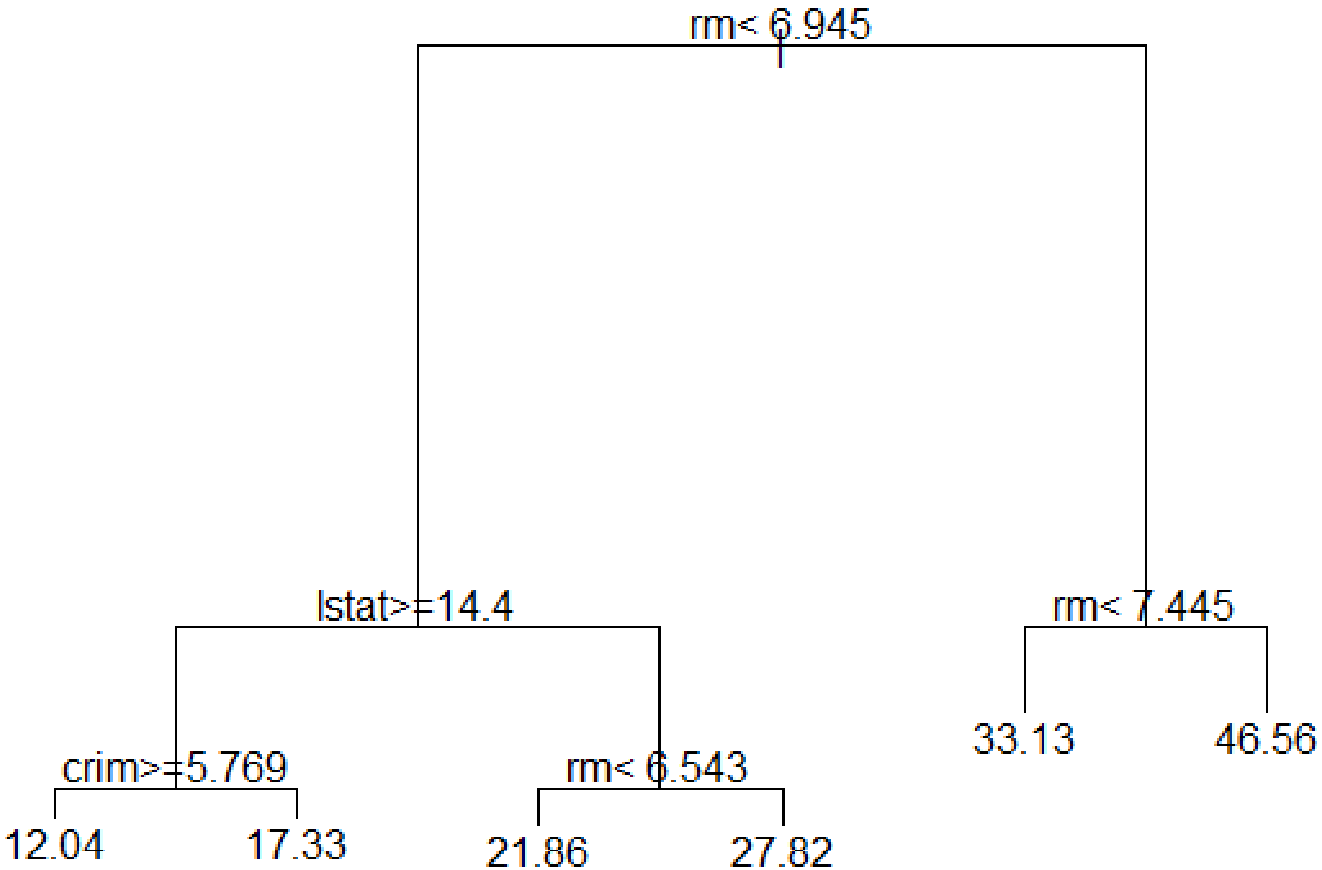

CV Error for Lasso
24.1
24.0
23.9
23.8
23.7
Mean Squared Error
0.00
0.02
0.04
0.06
0.08
0.10
alpha

7.1 Decision Tree - Stata

- 可使用非官方命令 crtrees 估计决策树
- 但功能比较原始
- 2019年才推出，算法是否靠谱？

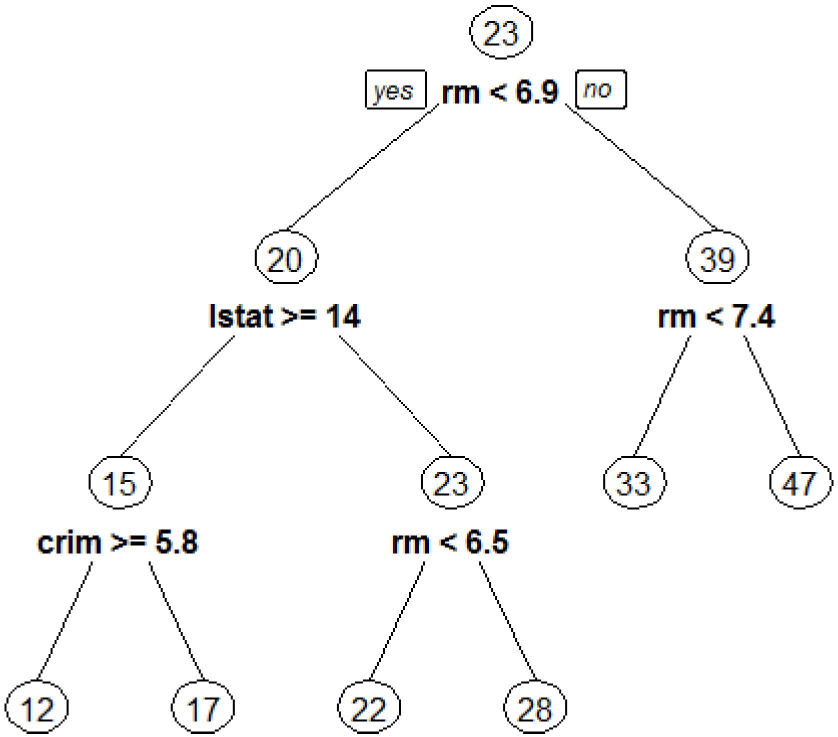
7.2 Decision Tree - R

```
> library(rpart)
> set.seed(1)    # set seed for random sampling
> train <- sample(506,354)  # 70% random sample
> set.seed(123)       # set seed for 10-fold CV
> fit <- rpart(medv~.,data=Boston,subset=train)

> op <- par(no.readonly = TRUE)
> par(mar=c(1,1,1,1))
> plot(fit,margin=0.1)
> text(fit)
> par(op)
```


rm< 6.945
lstat>=14.4
rm< 7.445
crim>=5.769
rm< 6.543
33.13
46.56
12.04
17.33
21.86
27.82

7.2 Decision Tree – R (续)

```
> library(rpart.plot)
> prp(fit,type=2)   # plot a rpart model
```



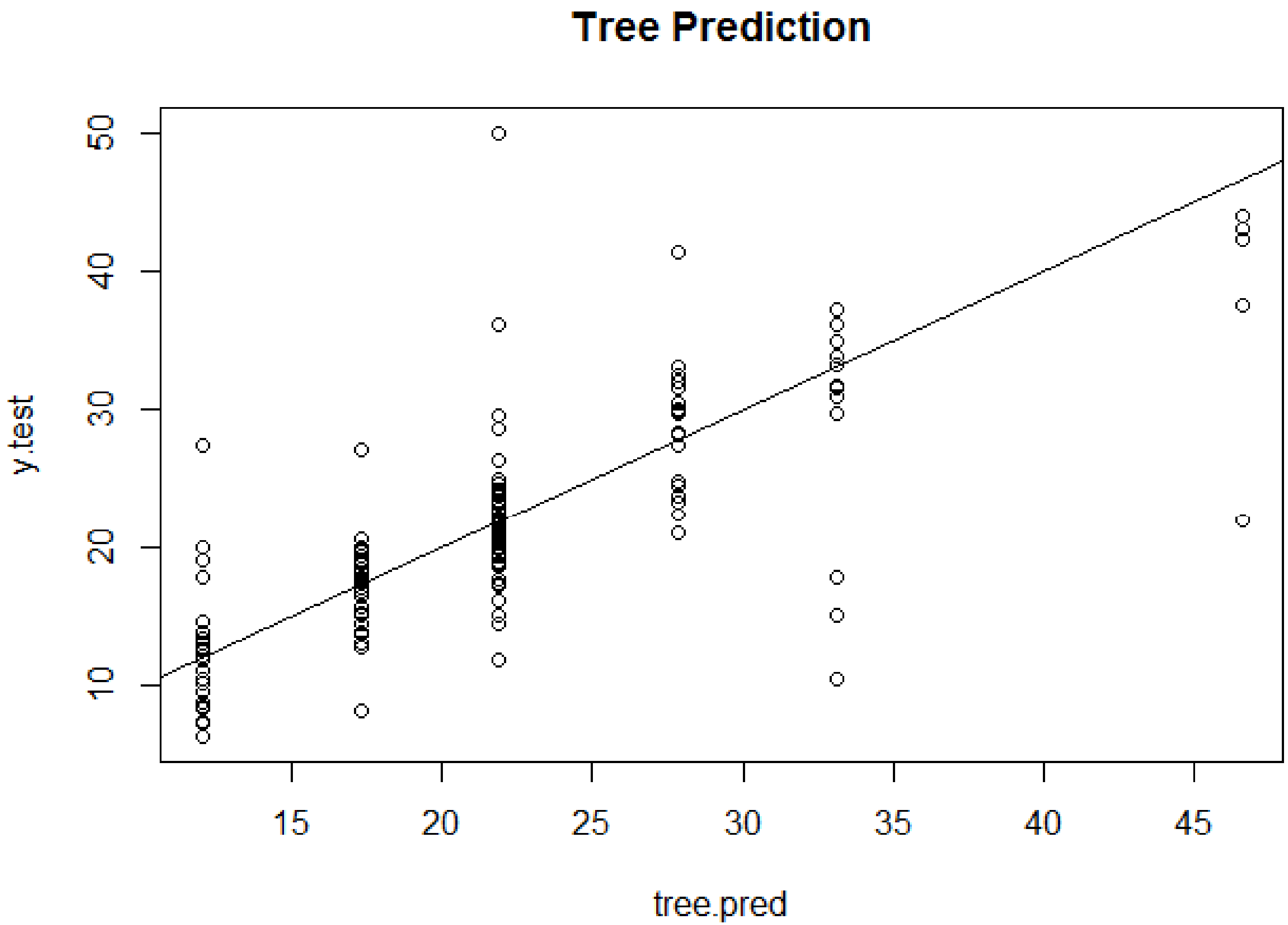
7.2 Decision Tree – R (预测)

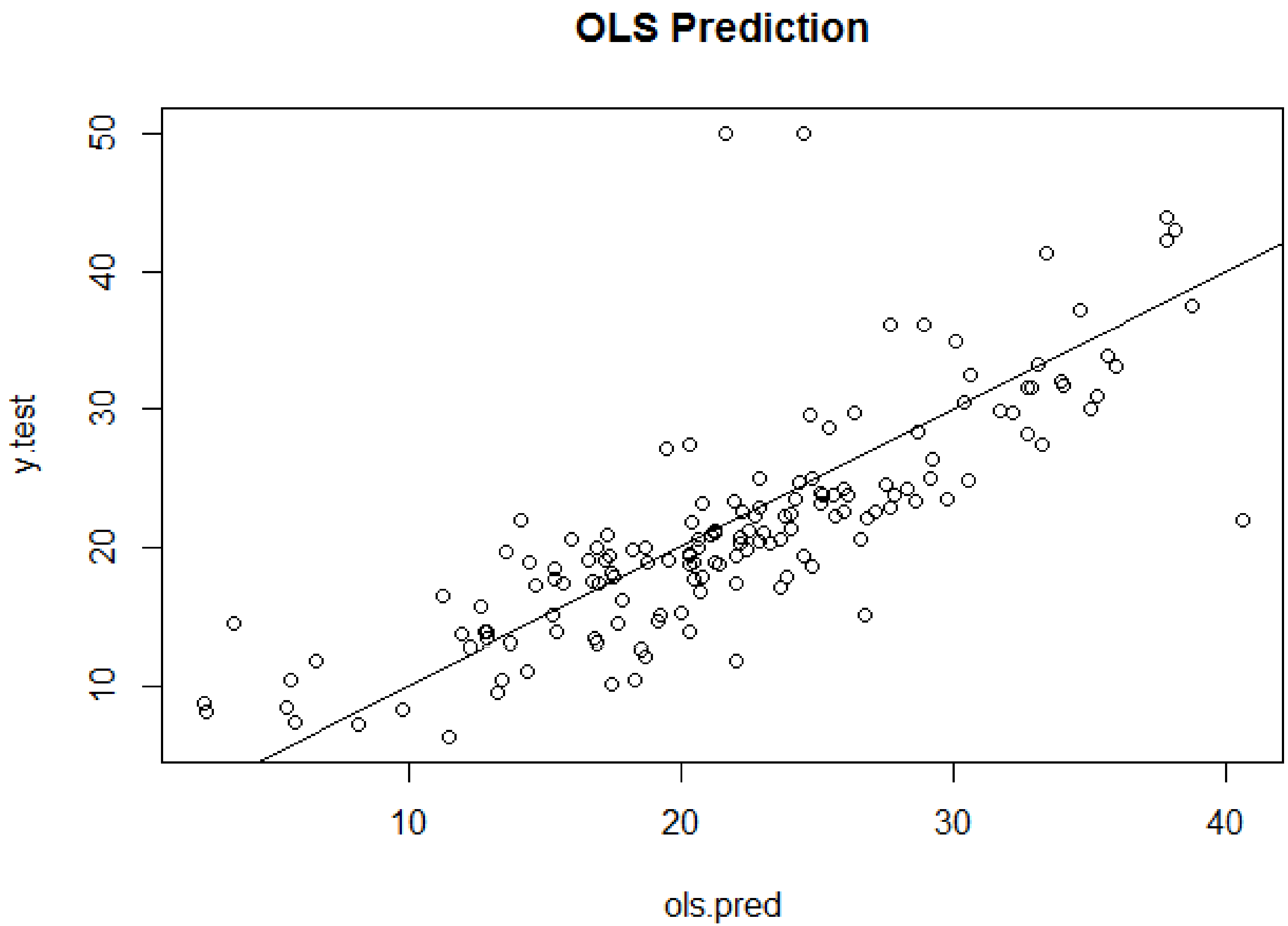
```
> tree.pred <- predict(fit,newdata=Boston[-train,])
> y.test <- Boston[-train,"medv"]
> mean((tree.pred-y.test)^2)   # MSE
  [1] 36.2319

# Compare with OLS
> ols.fit <- lm(medv~.,Boston,subset=train)
> ols.pred <- predict(ols.fit,
                      newdata=Boston[-train,])
> mean((ols.pred-y.test)^2)
  [1] 27.31196
```

7.2 Decision Tree – R (预测画图)

```
> plot(tree.pred,y.test,
       main="Tree Prediction")
> abline(0,1)
```

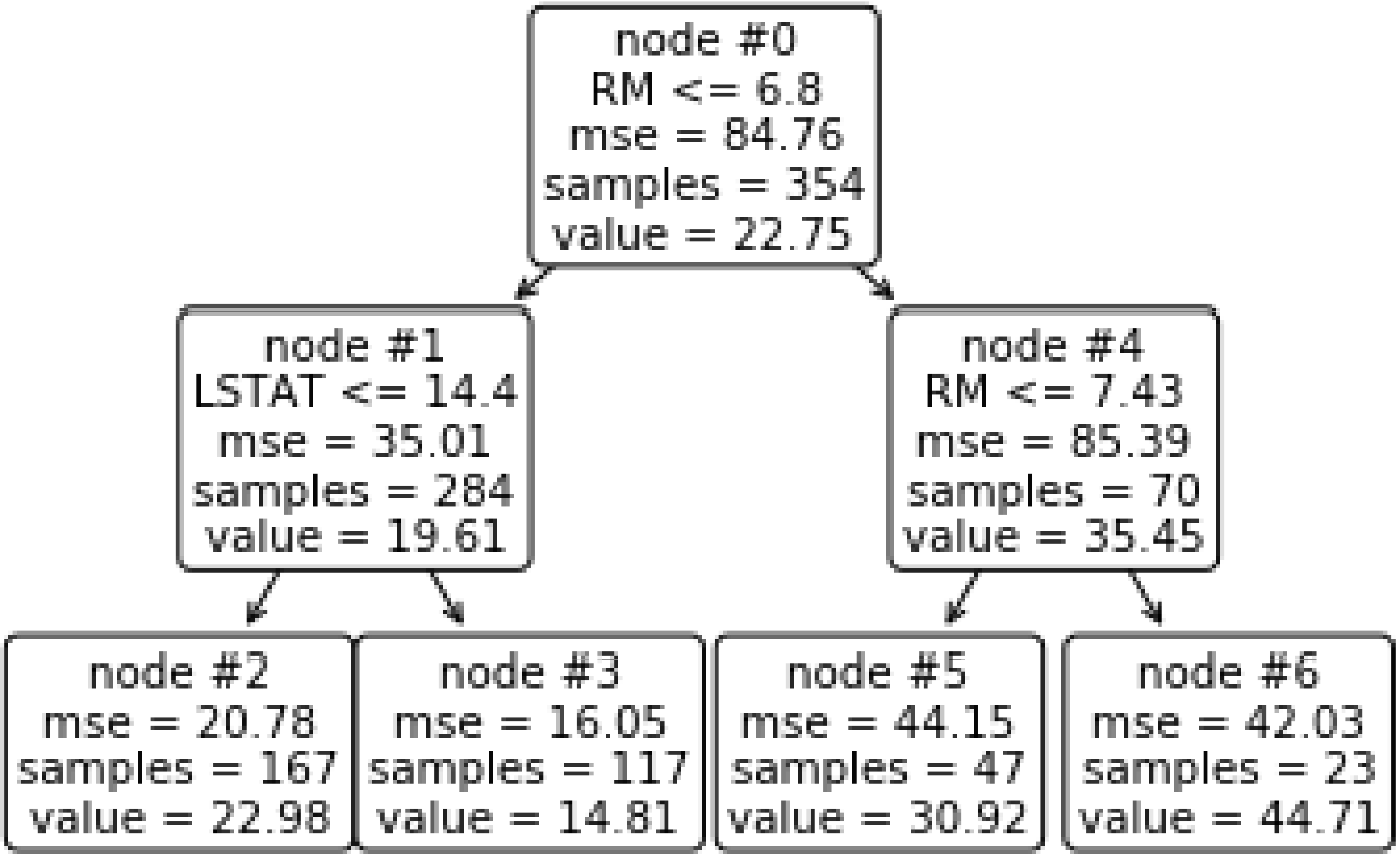
```
> plot(ols.pred,y.test,
       main="OLS Prediction")
> abline(0,1)
```


Tree Prediction
y.test
10
20
30
40
50
15
20
25
30
35
40
45
tree.pred


OLS Prediction
y.test
ols.pred
10
20
30
40
50
10
20
30
40

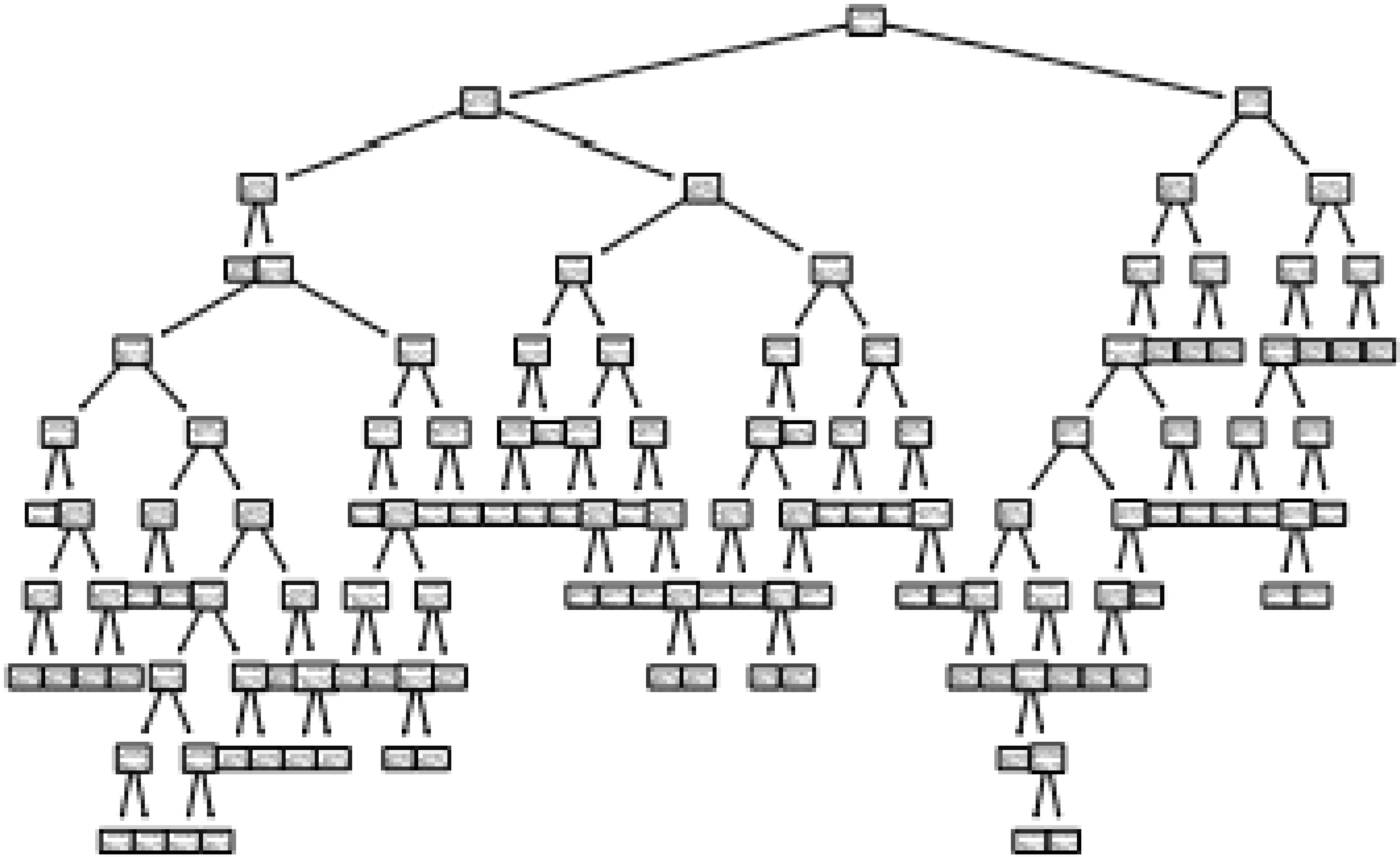
7.3 Decision Tree – Python

```
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.model_selection import KFold, StratifiedKFold
>>> from sklearn.model_selection import GridSearchCV
>>> from sklearn.tree import DecisionTreeRegressor,export_text
>>> from sklearn.tree import DecisionTreeClassifier, plot_tree
>>> from sklearn.datasets import load_boston

>>> Boston = load_boston()
>>> X_train, X_test, y_train, y_test =
        train_test_split(Boston.data, Boston.target,
                         test_size=0.3, random_state=0)
>>> model = DecisionTreeRegressor(max_depth=2, random_state=123)
>>> model.fit(X_train, y_train)
>>> plot_tree(model, feature_names=Boston.feature_names,
   node_ids=True, rounded=True, precision=2)
```


node #0
RM <= 6.8
mse = 84.76
samples = 354
value = 22.75
node #1
LSTAT <= 14.4
mse = 35.01
samples = 284
value = 19.61
node #4
RM <= 7.43
mse = 85.39
samples = 70
value = 35.45
node #2
mse = 20.78
samples = 167
value = 22.98
node #3
mse = 16.05
samples = 117
value = 14.81
node #5
mse = 44.15
samples = 47
value = 30.92
node #6
mse = 42.03
samples = 23
value = 44.71

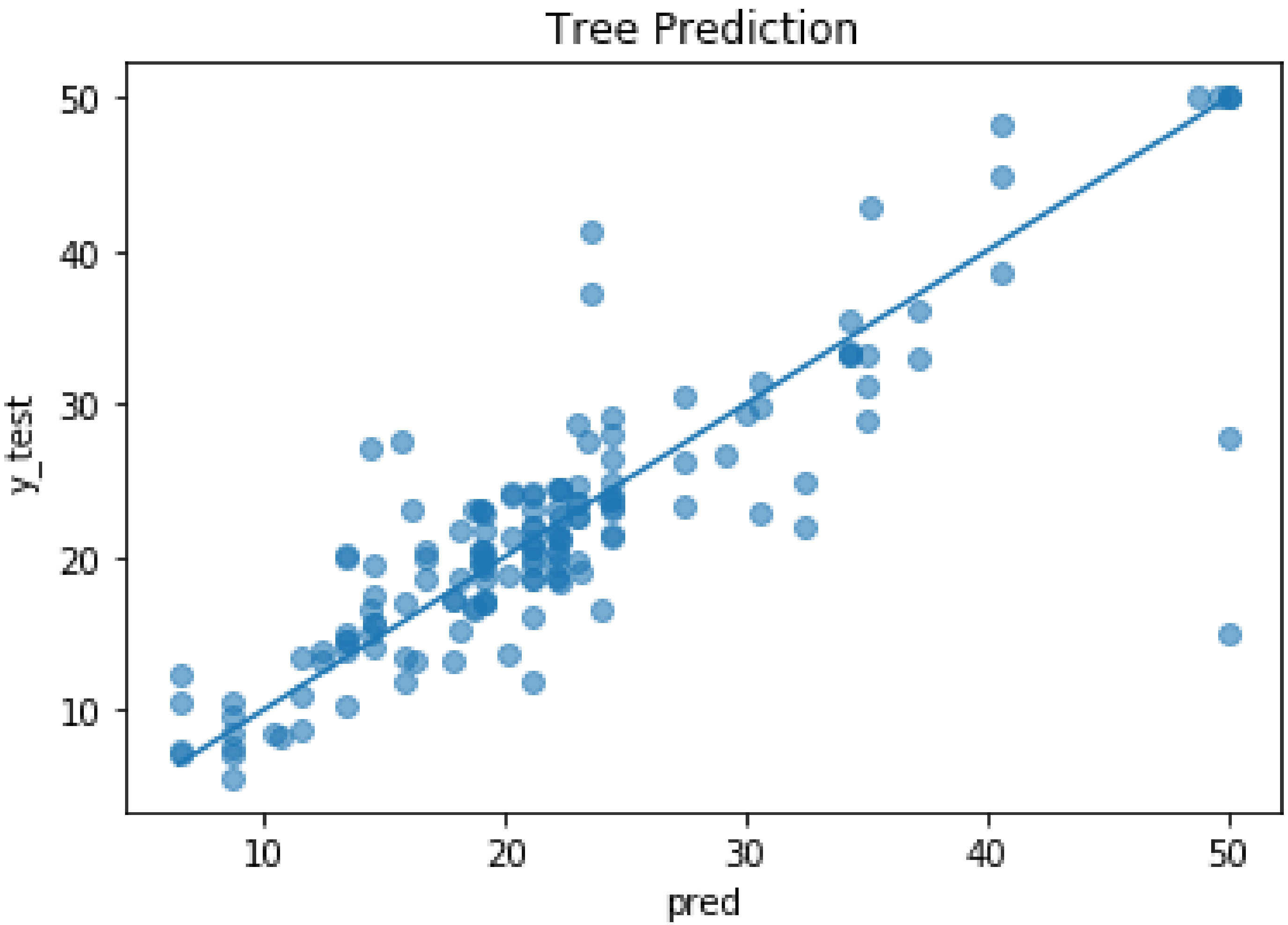
7.3 Decision Tree CV – Python

```
>>> model = DecisionTreeRegressor(random_state=123)
>>> path = model.cost_complexity_pruning_path(X_train,
                                              y_train)
>>> param_grid = {'ccp_alpha': path.ccp_alphas}
>>> kfold = KFold(n_splits=10, shuffle=True,
                  random_state=1)
>>> model =
  GridSearchCV(DecisionTreeRegressor(random_state=123),
               param_grid, cv=kfold)
>>> model.fit(X_train, y_train)
>>> model = model.best_estimator_
>>> plot_tree(model, feature_names=Boston.feature_names,
      node_ids=True, rounded=True, precision=2)
```



7.3 Decision Tree – Python (预测)

```
>>> pred = model.predict(X_test)
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, pred)
26.241625458064874

>>> plt.scatter(pred, y_test, alpha=0.6)
>>> w = np.linspace(min(pred), max(pred), 100)
>>> plt.plot(w, w)
>>> plt.xlabel('pred')
>>> plt.ylabel('y_test')
>>> plt.title('Tree Prediction')
```


Tree Prediction
50
40
30
20
10
y_test
10
20
30
40
50
pred

8.1 Random Forest - Stata

- 可使用非官方命令 crtrees 或 rforest 估计随机森林

- rforest 调用Java backend in Weka

- 功能是否全面，算法是否靠谱？

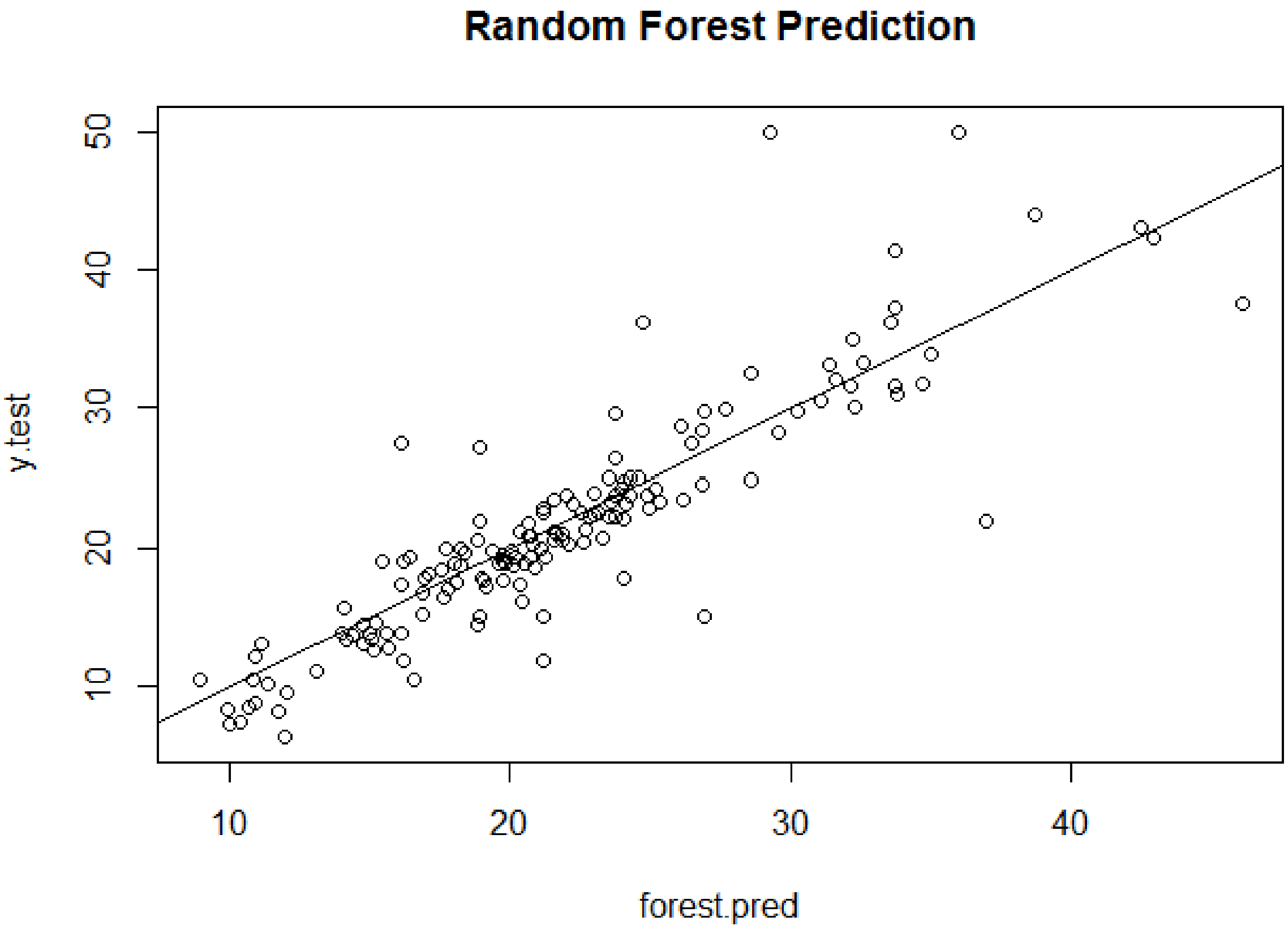
8.2 Random Forest - R

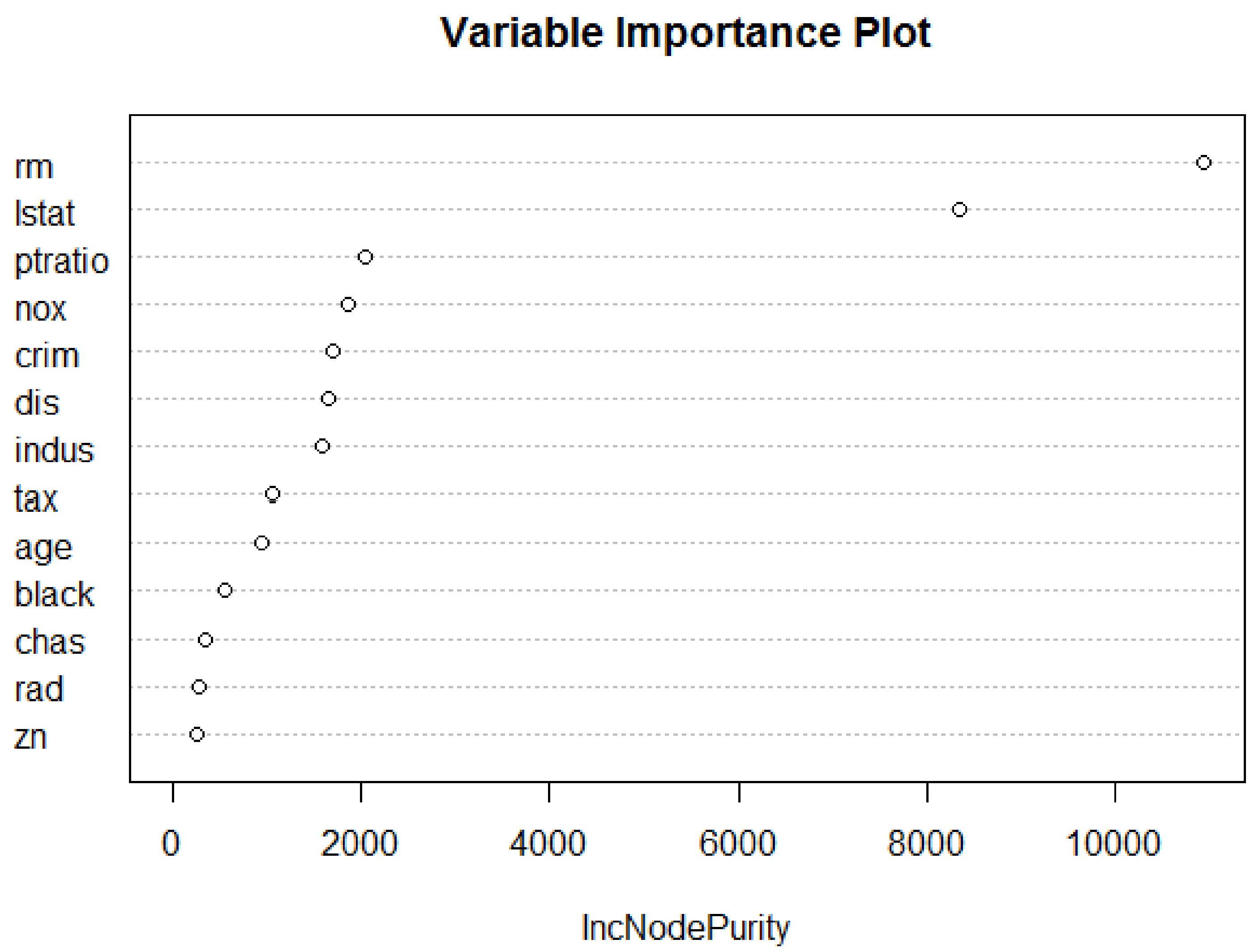
```
> library(randomForest)
> set.seed(123)
> forest.fit <- randomForest(medv~.,
                        data=Boston,subset=train)
            # Default mtry=p/3 for regression
> forest.fit
```

```
Call:
 randomForest(formula = medv ~ ., data = Boston, subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 10.33118
                    % Var explained: 88.67
```

8.2 Random Forest – R (预测)

```
> forest.pred <- predict(forest.fit,
                         newdata=Boston[-train,])
> mean((forest.pred-y.test)^2)
  [1] 14.65405

> plot(forest.pred,y.test,
       main="Random Forest Prediction")
> abline(0,1)

# 变量重要性图(Variable Importance Plot)
> varImpPlot(forest.fit,
             main="Variable Importance Plot")
```


Random Forest Prediction
y.test
forest.pred
10
20
30
40
50
10
20
30
40

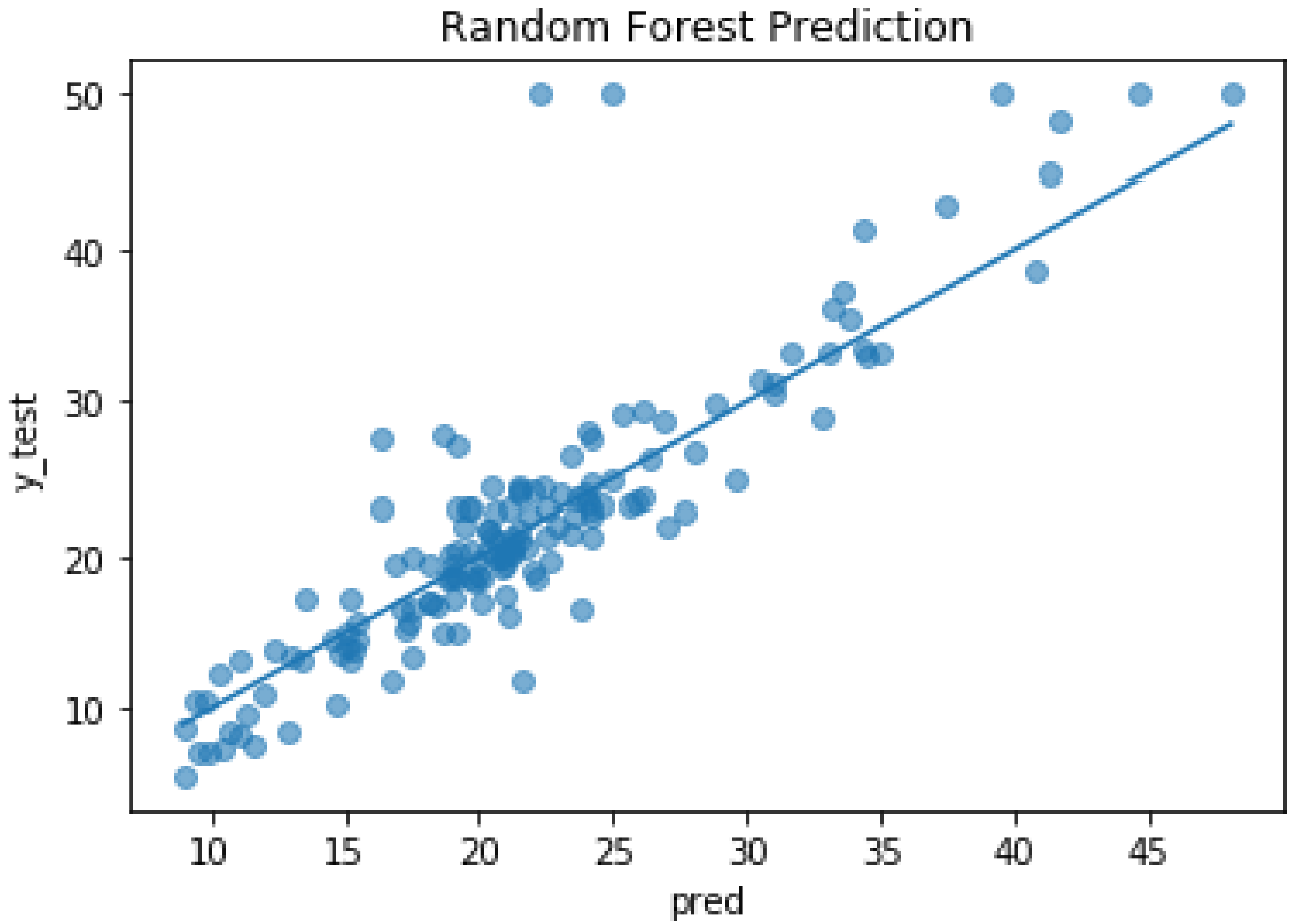

Variable Importance Plot
rm
lstat
ptratio
nox
crim
dis
indus
tax
age
black
chas
rad
zn
0
2000
4000
6000
8000
10000
IncNodePurity

8.3 Random Forest – Python

```
>>> from sklearn.ensemble import
      RandomForestRegressor

>>> max_features=int(X_train.shape[1] / 3)
>>> model = RandomForestRegressor(n_estimators=500,
          max_features=max_features, random_state=0)
>>> model.fit(X_train, y_train)

>>> pred = model.predict(X_test)
>>> mean_squared_error(pred, y_test)
18.590988882368425
```

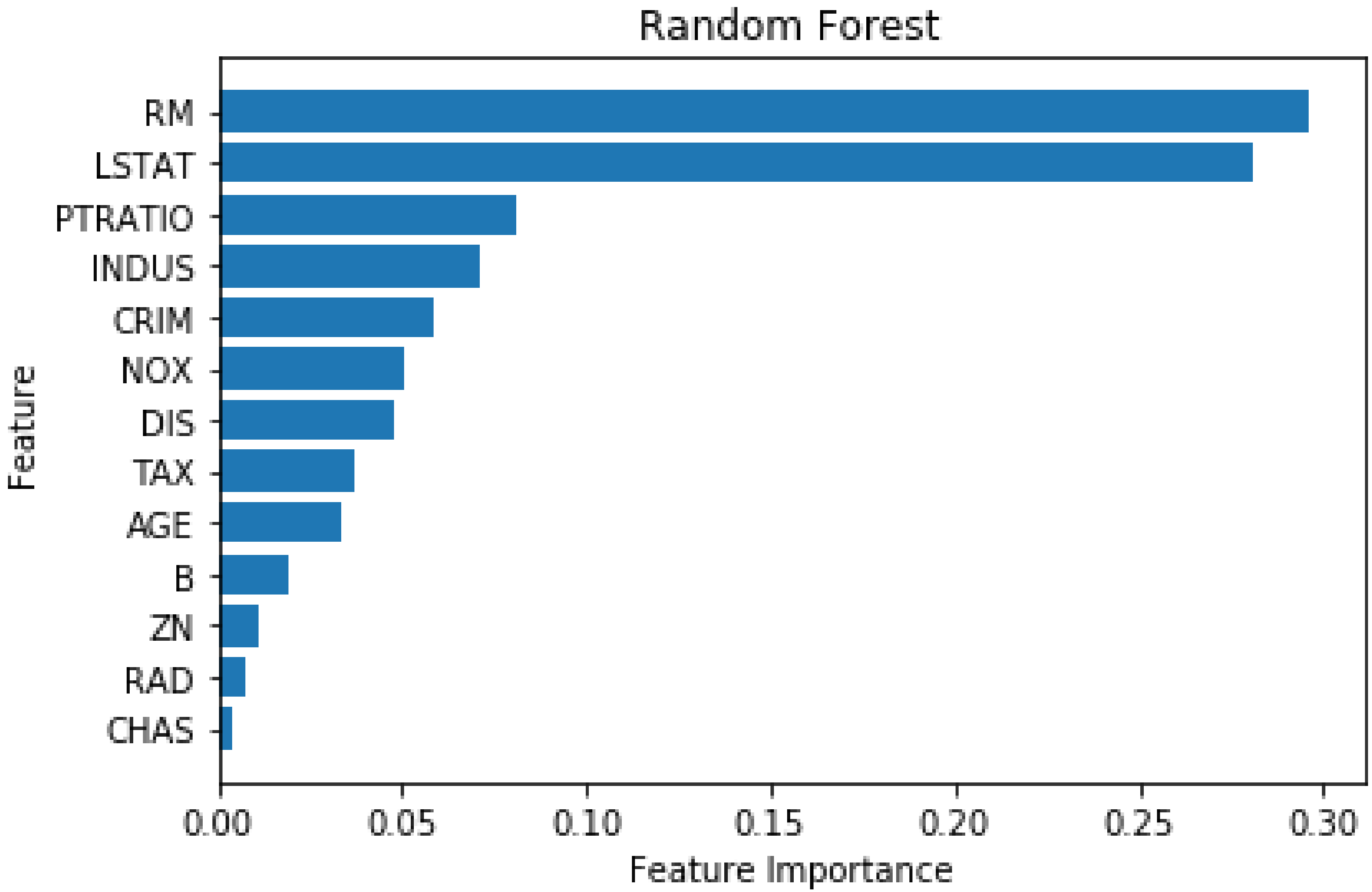
8.3 Random Forest – Python (预测画图)

```
>>> plt.scatter(pred, y_test, alpha=0.6)
>>> w = np.linspace(min(pred), max(pred),
                    100)
>>> plt.plot(w, w)
>>> plt.xlabel('pred')
>>> plt.ylabel('y_test')
>>> plt.title('Random Forest Prediction')
```


Random Forest Prediction
50
40
30
20
10
y_test
10
15
20
25
30
35
40
45
pred

8.3 Random Forest – Python (变量重要性)

```
>>> sorted_index =
        model.feature_importances_.argsort()
>>> plt.barh(range(X.shape[1]),
      model.feature_importances_[sorted_index])
>>> plt.yticks(np.arange(X.shape[1]),
      X.columns[sorted_index])
>>> plt.xlabel('Feature Importance')
>>> plt.ylabel('Feature')
>>> plt.title('Random Forest')
>>> plt.tight_layout()
```


Random Forest
RM
LSTAT
PTRATIO
INDUS
CRIM
NOX
DIS
TAX
AGE
B
ZN
RAD
CHAS
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Feature Importance

9.1 Neural Network - Stata

- 可使用非官方命令 brain 估计神经网络模型

- 但功能有限

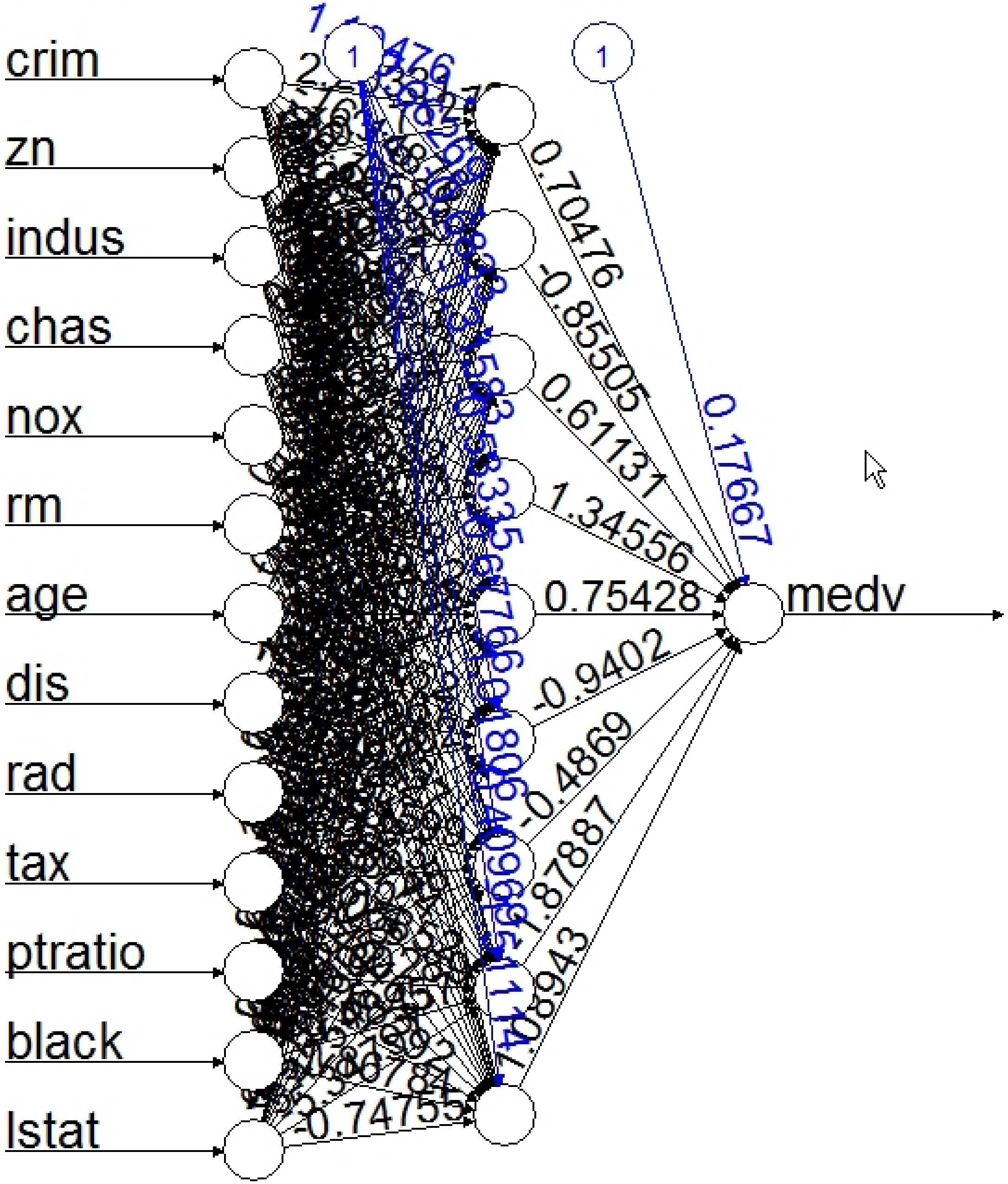
9.2 Neural Network - R

- 在R中，可使用 nnet 或 neuralnet 估计“前馈神经网络” (feedforward neural network)，但无法估计“卷积神经网络”(convolution neural network)或“循环神经网络”(recurrent neural network)等。

- 也可以通过Keras调用tensorflow，但易出错(Keras的底层语言为Python)

9.2 Neural Network – R (续)

```
> library(neuralnet)
> set.seed(123)
> fit <-
  neuralnet(medv~.,data=Boston_s[train,],
            hidden=9,act.fct="logistic",
            linear.output = TRUE)
> plot(fit,fontsize = 20)
> pred <- pred*(max(Boston$medv)-
          min(Boston$medv))+min(Boston$medv)
> mean((pred-y.test)^2)
  [1] 12.04135
```



crim
zn
indus
chas
nox
rm
age
dis
rad
tax
ptratio
black
lstat
1
1
0.70476
-0.85505
0.61131
1.34556
0.75428
-0.9402
-0.4869
1.87887
1.08943
0.17667
-0.74755
medv

9.2 Neural Network – R (续2)

```
> # Optimal number of neurons (偷看了测试集)
> MSE <- numeric(10)
> for (i in 1:10){
      set.seed(123)
      fit <- neuralnet(medv~.,data=Boston_s[train,],
                       hidden= i,linear.output = TRUE)
      pred <- predict(fit,Boston_s[-train,])
      pred <- pred*(max(Boston$medv)-
                    min(Boston$medv))+min(Boston$medv)
      MSE[i] <- mean((pred-y.test)^2)
  }
> plot(1:10,MSE,type="b",xlab="Number of Hidden
       Neurons",main="Boston Housing Data")
> abline(v=which.min(MSE),lty=2)
```


Boston Housing Data
MSE
30
25
20
15
2
4
6
8
10
Number of Hidden Neurons

9.3 Neural Network - Python

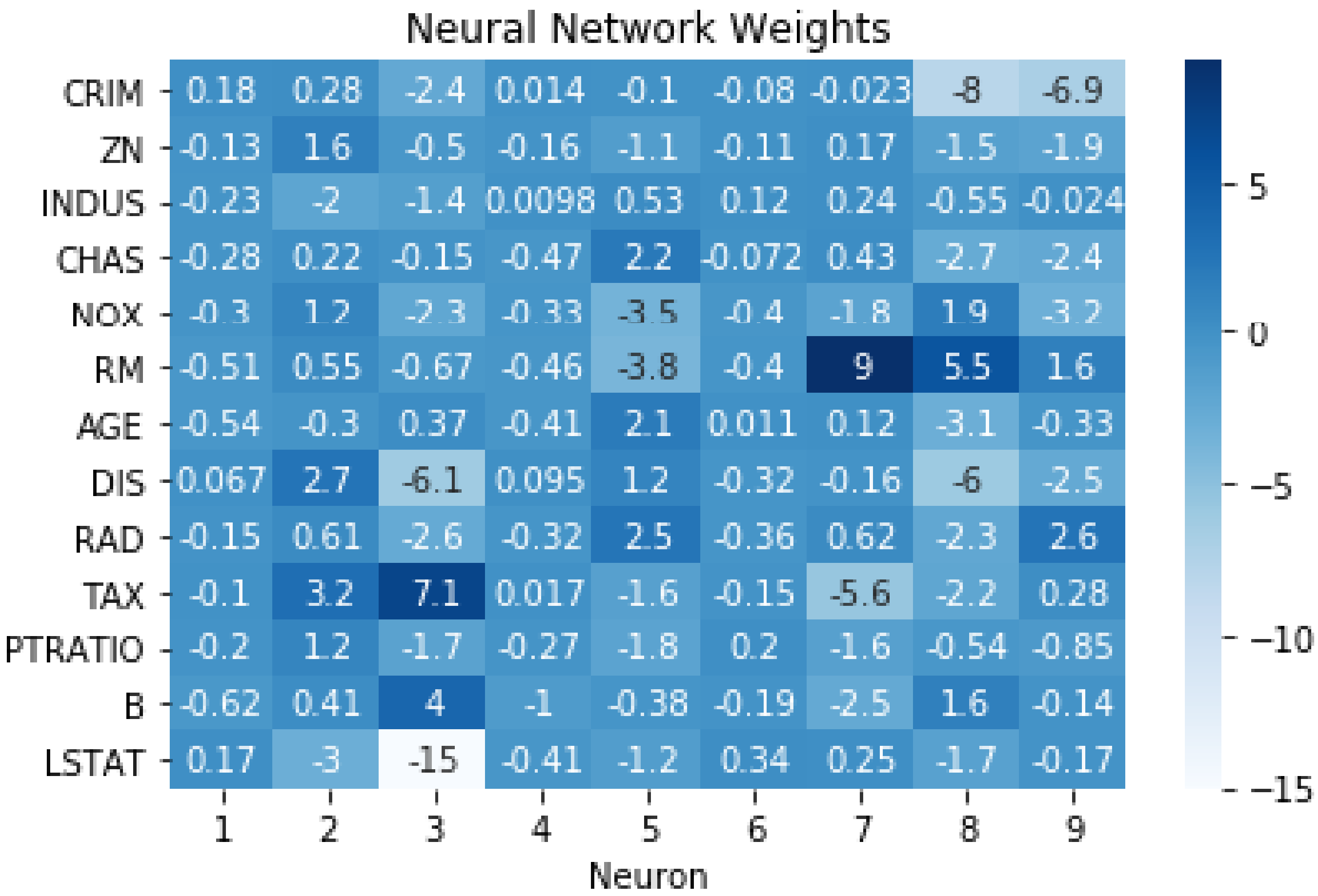
- 在Python中，可使用sklearn估计前馈神经网络模型。更专业的深度学习框架包括tensorflow，PyTorch等。
- 一个简便方法是通过Keras调用tensorflow，详见《机器学习及Python应用》
- 在此仅演示用sklearn估计神经网络

9.3 Neural Network – Python (续)

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> from sklearn.neural_network import MLPRegressor
>>> scaler = MinMaxScaler()
>>> scaler.fit(X_train)
>>> X_train_s = scaler.transform(X_train)
>>> X_test_s = scaler.transform(X_test)
>>> model = MLPRegressor(solver='lbfgs',
                hidden_layer_sizes=(9,), random_state=123,
                max_iter=10000)
>>> model.fit(X_train_s, y_train)
>>> pred = model.predict(X_test_s)
>>> mean_squared_error(pred, y_test)
19.571499847444066
```

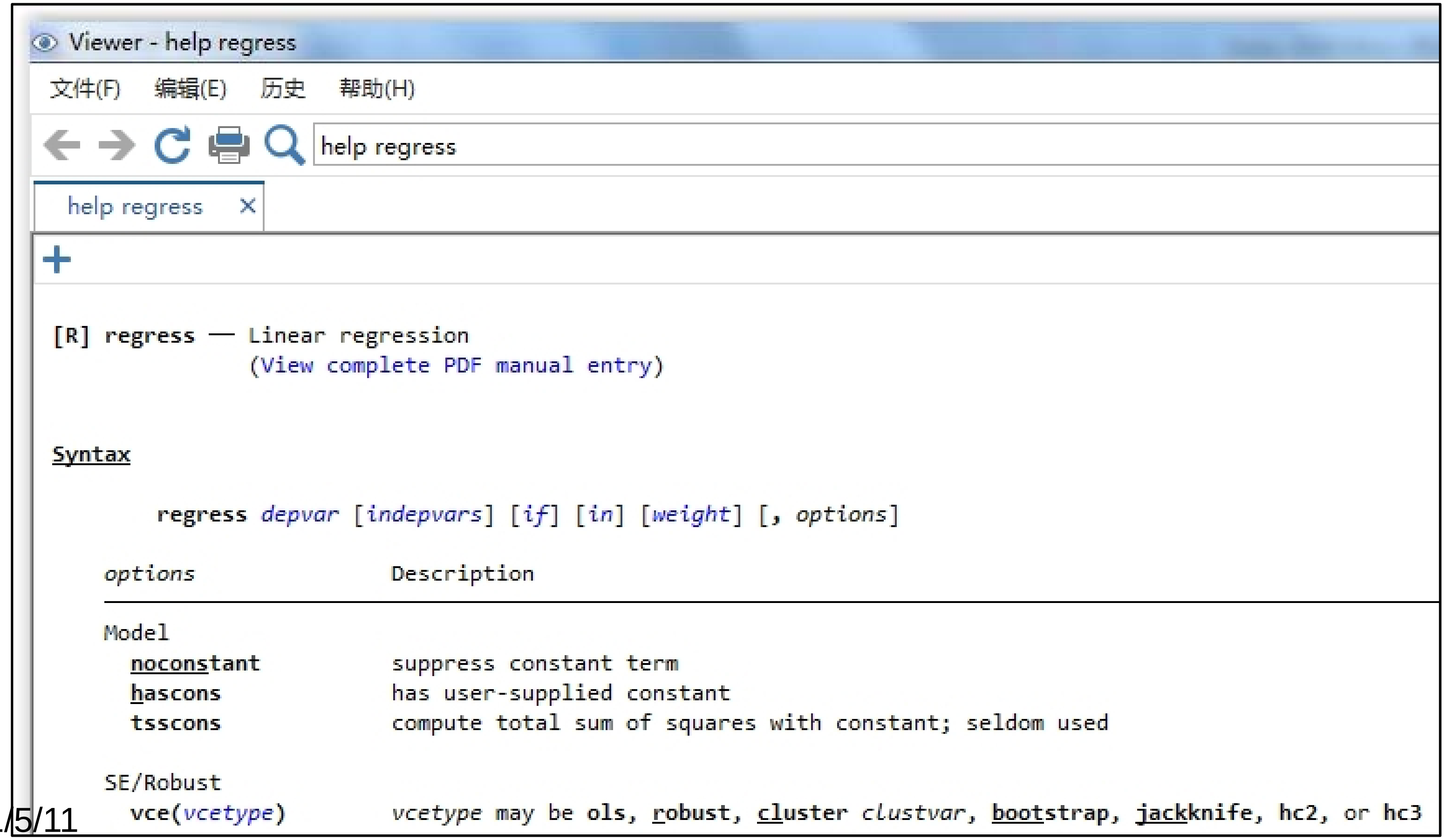
9.3 Neural Network – Python (续2)

```
# 考察神经网络的权重矩阵

>>> table = pd.DataFrame(model.coefs_[0],
                index=Boston.feature_names,
                columns=[1,2,3,4,5,6,7,8,9])
>>> sns.heatmap(table, cmap='Blues',
                annot=True)
>>> plt.xlabel('Neuron')
>>> plt.title('Neural Network Weights')
>>> plt.tight_layout()
```


Neural Network Weights
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT
Neuron
1
2
3
4
5
6
7
8
9
5
0
−5
−10
−15

10.1 帮助文件 - Stata

- `help reg`



10.2 帮助文件 - R

```
> ?lm
```

lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

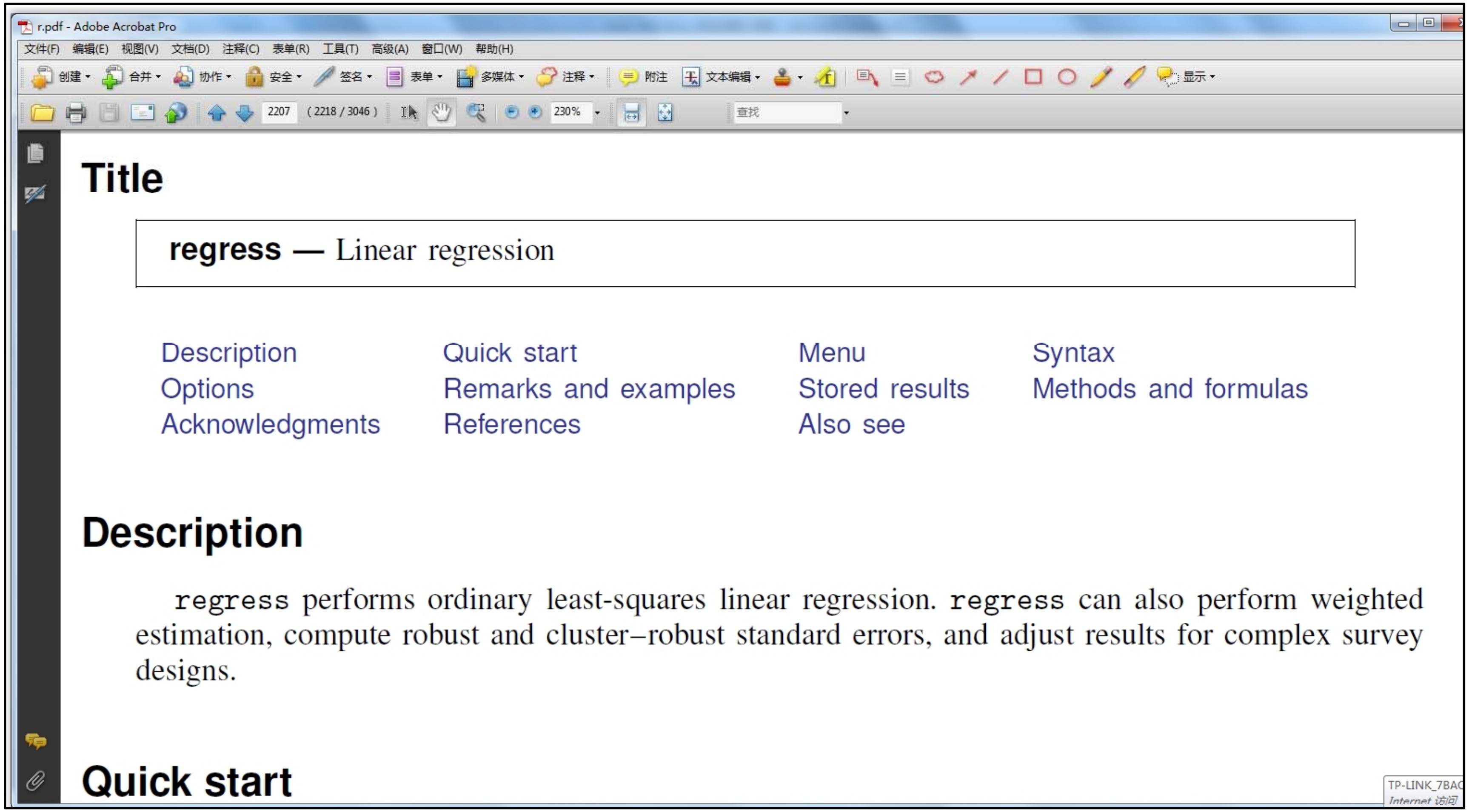
`formula`	an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.

10.3 帮助文件 - Python

```
> smf.ols?
```

```
Signature: smf.ols(formula, data, subset=None, drop_cols=None, *args, **kwargs)
Docstring:
Create a Model from a formula and dataframe.

Parameters
----------
formula : str or generic Formula object
    The formula specifying the model.
data : array_like
    The data for the model. See Notes.
subset : array_like
    An array-like object of booleans, integers, or index values that
    indicate the subset of df to use in the model. Assumes df is a
    `pandas.DataFrame`.
drop_cols : array_like
    Columns to drop from the design matrix.  Cannot be used to
    drop terms involving categoricals.
*args
    Additional positional argument that are passed to the model.
**kwargs
    These are passed to the model with one exception. The
    ``eval_env`` keyword is passed to patsy. It can be either a
    :class:`patsy:patsy.EvalEnvironment` object or an integer
    indicating the depth of the namespace to use. For example, the
    default ``eval_env=0`` uses the calling namespace. If you wish
    to use a "clean" environment set ``eval_env=-1``.
```

11.1 用户手册 - Stata



Title

regress — Linear regression

Description | Quick start | Menu | Syntax
Options | Remarks and examples | Stored results | Methods and formulas
Acknowledgments | References | Also see

Description

`regress` performs ordinary least-squares linear regression. `regress` can also perform weighted estimation, compute robust and cluster–robust standard errors, and adjust results for complex survey designs.

Quick start

11.2 用户手册 - R

- 有些R包有“小品文”(vignette)，相当于用户手册。

- 比如，打开机器学习R包caret的小品文

```
> vignette('caret')
```

- 但许多R包没有小品文，更多地依赖于网络搜索或发帖获得帮助

A Short Introduction to the caret Package ▾ Find in Topic

A Short Introduction to the caret Package

The **caret** package (short for Classification And REgression Training) contains functions to streamline the model training process for complex regression and classification problems. The package utilizes a number of R packages but tries not to load them all at package start-up (by removing formal package dependencies, the package startup time can be greatly decreased). The package "suggests" field includes 29 packages. **caret** loads packages as needed and assumes that they are installed. If a modeling package is missing, there is a prompt to install it.

Install **caret** using

```
install.packages("caret", dependencies = c("Depends", "Suggests"))
```

to ensure that all the needed packages are installed.

The **main help pages** for the package are at https://topepo.github.io/caret/ Here, there are extended examples and a large amount of information that previously found in the package vignettes.

caret has several functions that attempt to streamline the model building and evaluation process,

11.3 用户手册 - Python

- Python官网有Python Documentations
- https://www.python.org/doc/
- 机器学习的Python包sklearn的官网有很好的网页版用户手册（含案例）
- https://sklearn.org/

12. 总结

- Stata：在一定范围内(计量经济学)易学易用，但出此范围则较弱(比如机器学习)，依赖于Stata公司，成本较高。影响力局限于学术界。

- R：为统计而生，擅长统计推断，开源免费。在神经网络方面较弱。

- Python：计算机的主流语言，通用而全能，为业界所青睐。统计推断(比如标准误，*p*值)较弱。

- 建议：选择最适合你的语言，而非最时髦的语言

终极考量

- Stata：接轨(计量)经济学家
- R：接轨统计学家
- Python：接轨计算机科学家
- 进一步学习的资源(含在经管领域的应用)

www.econometrics-stata.com