MONTE CARLO SIMULATION OF DATA THAT FOLLOWS A GIVEN COVARIANCE STRUCTURE (USING R, SAS AND PYTHON)

DEE H. WU, PH.D.

ASSOCIATE PROFESSOR

DEPARTMENT OF RADIOLOGICAL SCIENCES

HTTP://MOON.OUHSC.EDU/DWU

For this presentation, we will go over how to generate multivariate simulated data given a specific covariance matrix of the data in R and SAS, as well as some Python in example.

There are two important concepts in doing

1) Setting up (in Python)
2) Cholesky Factorization (in R)
3) Factor Analysis/Principal Components (SAS)

numpy.random.multivariate_normal(mean, cov[, size])

Draw random samples from a multivariate normal distribution.

The multivariate normal, multinormal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its mean and covariance matrix. These parameters are analogous to the mean (average or “center”) and variance (standard deviation, or “width,” squared) of the one-dimensional normal distribution.

numpy.random.multivariate_normal(mean, cov[, size])

Parameters:	**mean** : 1-D array_like, of length N Mean of the N-dimensional distribution. **cov** : 2-D array_like, of shape (N, N) Covariance matrix of the distribution. Must be symmetric and positive-semidefinite for “physically meaningful” results. **size** : int or tuple of ints, optional Given a shape of, for example, (m,n,k), m*n*k samples are generated, and packed in an *m*-by-*n*-by-*k* arrangement. Because each sample is *N*-dimensional, the output shape is (m,n,k,N). If no shape is specified, a single (*N*-D) sample is returned.
Returns:	**out** : ndarray The drawn samples, of shape *size*, if that was provided. If not, the shape is (N,). In other words, each entry out[i,j,...,:] is an N-dimensional value drawn from the distribution.

```
samples = 200
r = 0.83

# Generate pearson correlated data with approximately cor(X, Y) = r
import numpy as np
data = np.random.multivariate_normal([0, 0], [[1, r], [r, 1]], size=samples)
X, Y = data[:,0], data[:,1]

# That's it! Now let's take a look at the actual correlation:
import scipy.stats as stats
print 'r=', stats.pearsonr(X, Y)[0]
```

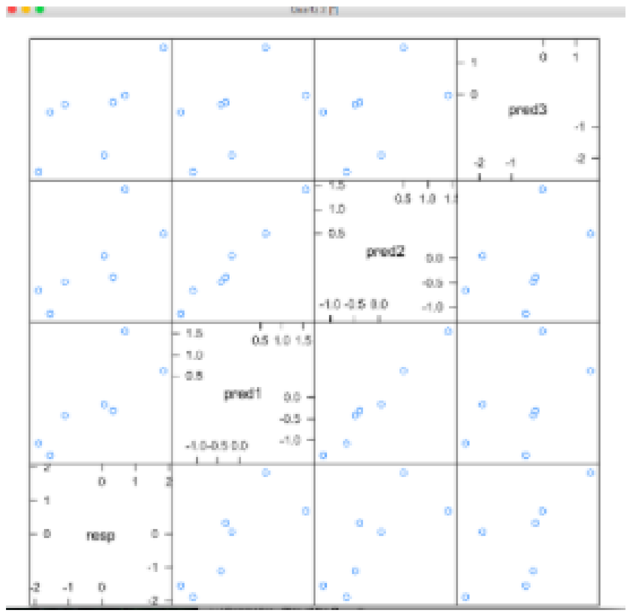
The Cholesky decomposition of a Hermitian positive-definite matrix **A** is a decomposition of the form

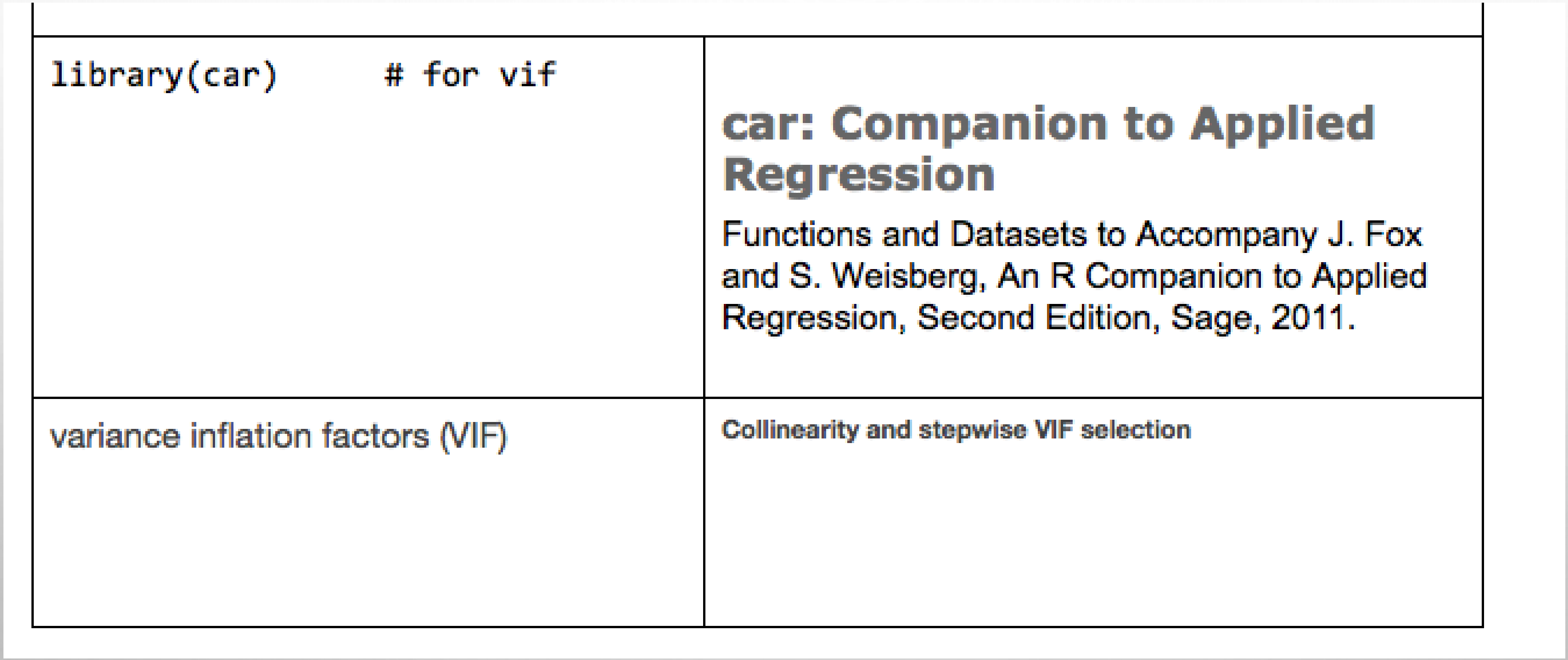
where **L** is a lower triangular matrix with real and positive diagonal entries

The Cholesky decomposition is unique when **A** is positive definite; there is only one lower triangular matrix **L** with strictly positive diagonal entries such that **A** = **LL***.

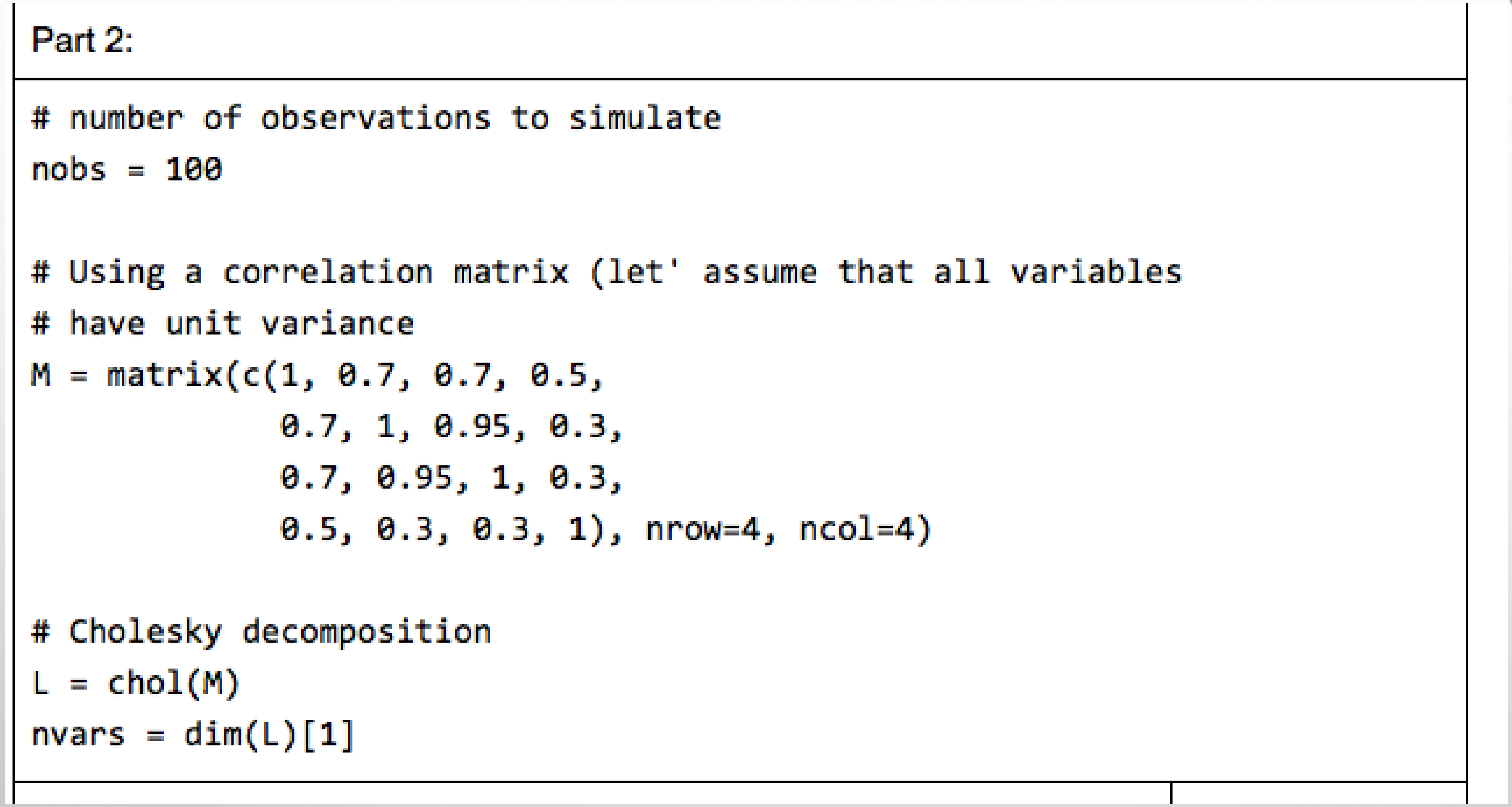
The **Cholesky algorithm,** used to calculate the decomposition matrix *L*, is a modified version of Gaussian elimination.

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} = \begin{pmatrix} 2 & & \\ 6 & 1 & \\ -8 & 5 & 3 \end{pmatrix} \begin{pmatrix} 2 & 6 & -8 \\ & 1 & 5 \\ & & 3 \end{pmatrix}$$

`library(lattice) # for splom`	**lattice also does Trellis Graphics for R** A powerful and elegant high-level data visualization system inspired by Trellis graphics, with an emphasis on multivariate data. Lattice is sufficient for typical graphics
	`splom` **Scatter Plot Matrices** **Description** Draw Conditional Scatter Plot Matrices and Parallel Coordinate Plots

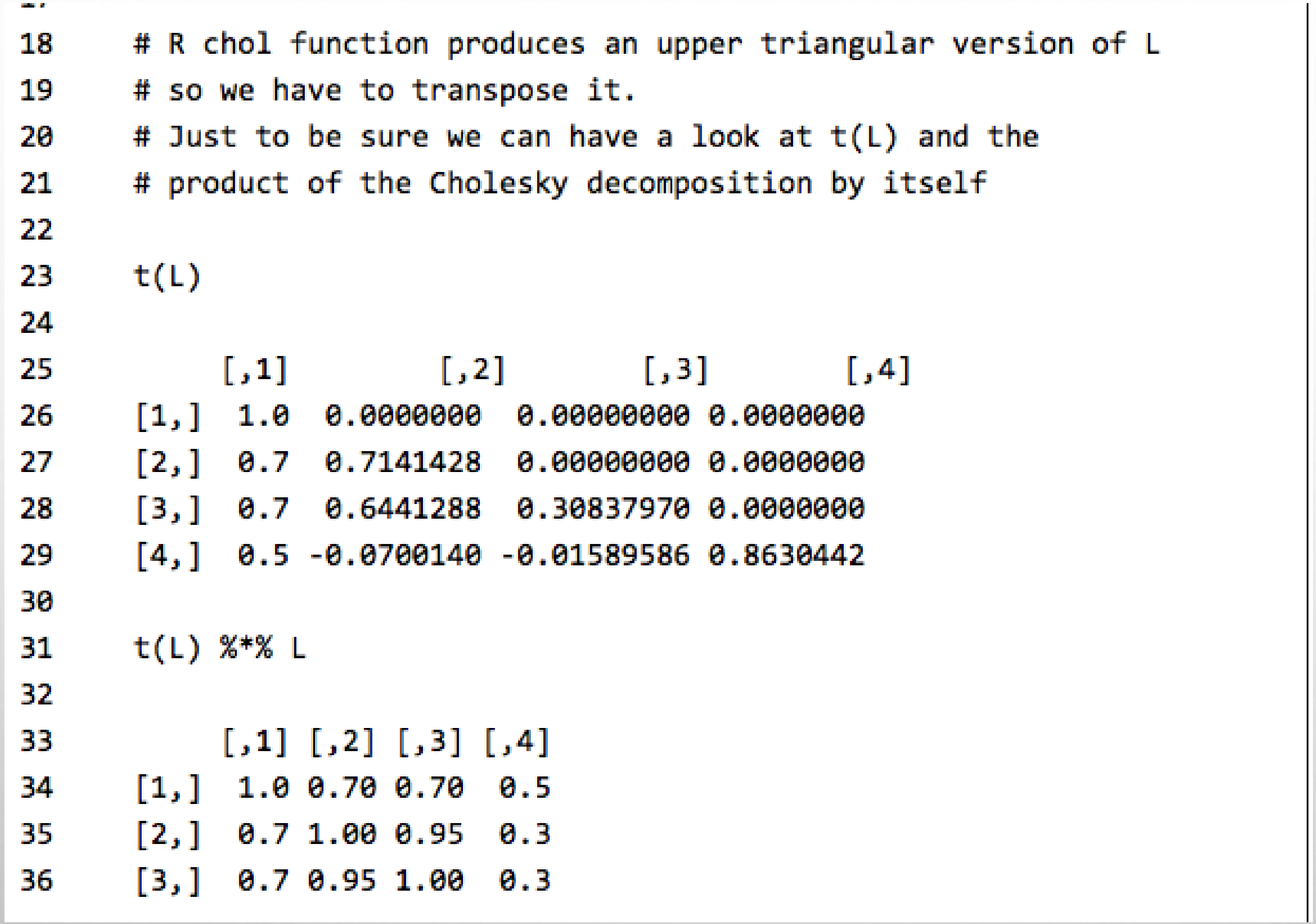


`library(car)     # for vif`	**car: Companion to Applied Regression** Functions and Datasets to Accompany J. Fox and S. Weisberg, An R Companion to Applied Regression, Second Edition, Sage, 2011.
variance inflation factors (VIF)	**Collinearity and stepwise VIF selection**

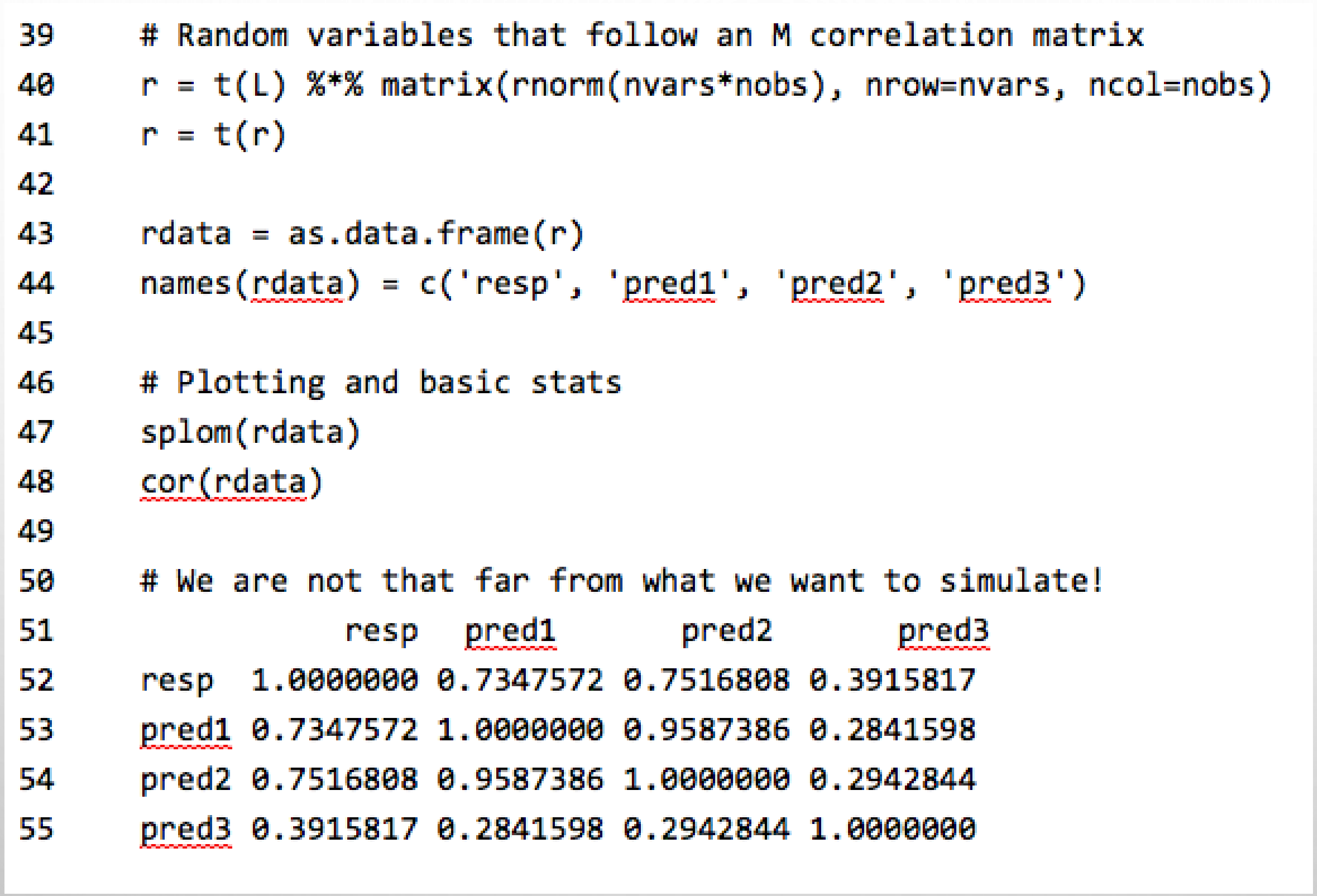


Part 2:

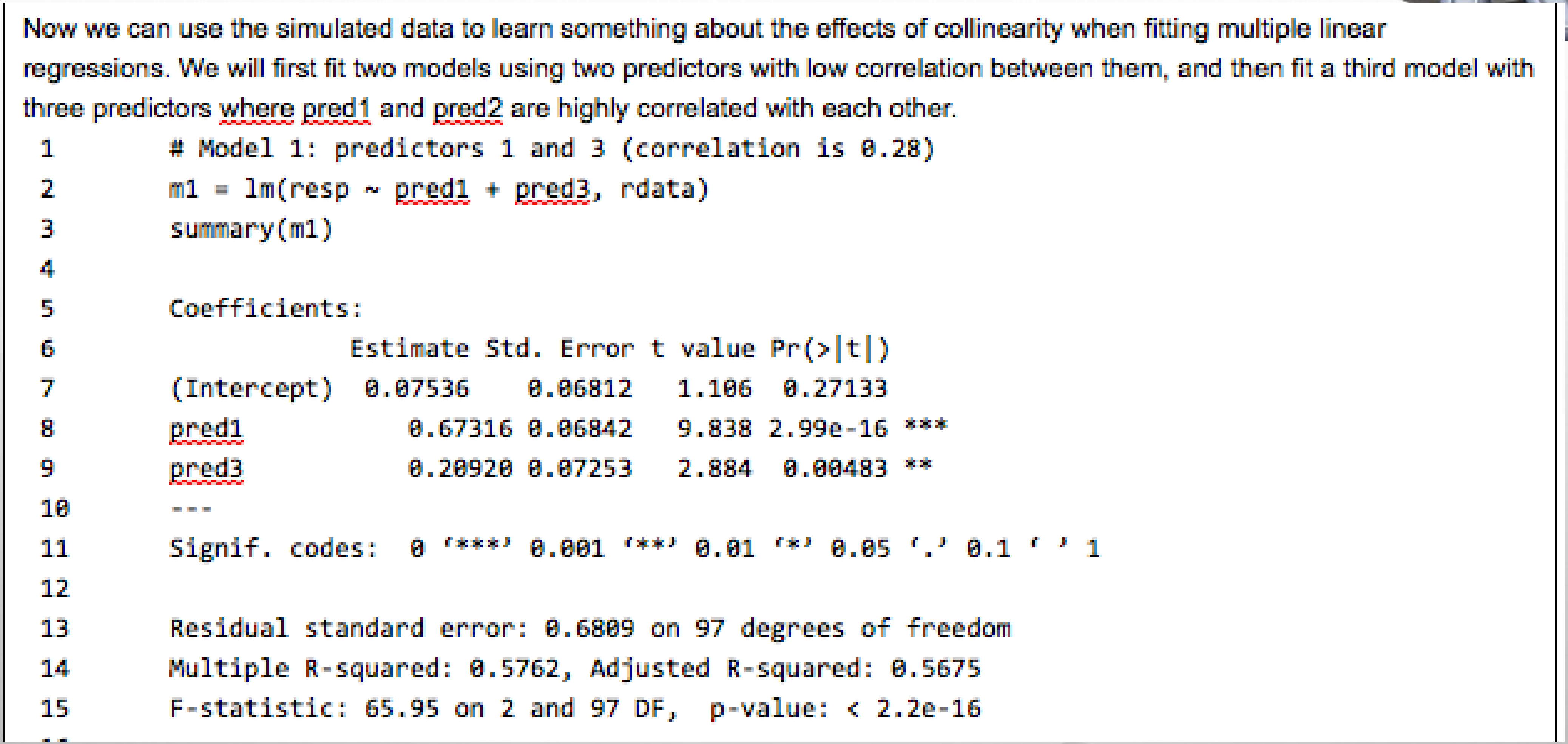
```
# number of observations to simulate
nobs = 100

# Using a correlation matrix (let' assume that all variables
# have unit variance
M = matrix(c(1, 0.7, 0.7, 0.5,
             0.7, 1, 0.95, 0.3,
             0.7, 0.95, 1, 0.3,
             0.5, 0.3, 0.3, 1), nrow=4, ncol=4)

# Cholesky decomposition
L = chol(M)
nvars = dim(L)[1]
```



```
# R chol function produces an upper triangular version of L
# so we have to transpose it.
# Just to be sure we can have a look at t(L) and the
# product of the Cholesky decomposition by itself

t(L)

     [,1]       [,2]        [,3]      [,4]
[1,]  1.0  0.0000000  0.00000000 0.0000000
[2,]  0.7  0.7141428  0.00000000 0.0000000
[3,]  0.7  0.6441288  0.30837970 0.0000000
[4,]  0.5 -0.0700140 -0.01589586 0.8630442

t(L) %*% L

     [,1] [,2] [,3] [,4]
[1,]  1.0 0.70 0.70  0.5
[2,]  0.7 1.00 0.95  0.3
[3,]  0.7 0.95 1.00  0.3
```

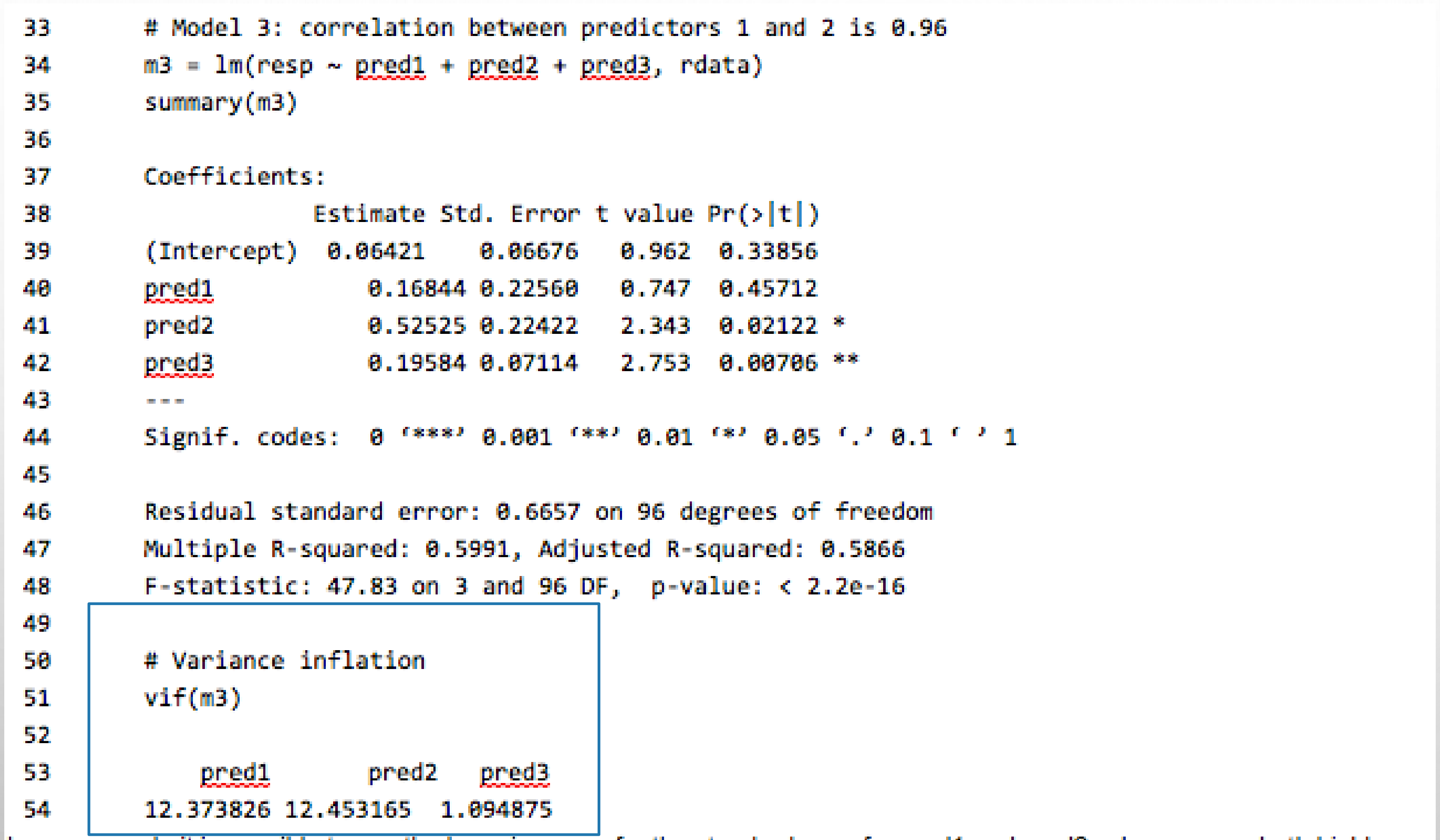


```r
# Random variables that follow an M correlation matrix
r = t(L) %*% matrix(rnorm(nvars*nobs), nrow=nvars, ncol=nobs)
r = t(r)

rdata = as.data.frame(r)
names(rdata) = c('resp', 'pred1', 'pred2', 'pred3')

# Plotting and basic stats
splom(rdata)
cor(rdata)

# We are not that far from what we want to simulate!
           resp   pred1       pred2       pred3
resp  1.0000000 0.7347572 0.7516808 0.3915817
pred1 0.7347572 1.0000000 0.9587386 0.2841598
pred2 0.7516808 0.9587386 1.0000000 0.2942844
pred3 0.3915817 0.2841598 0.2942844 1.0000000
```



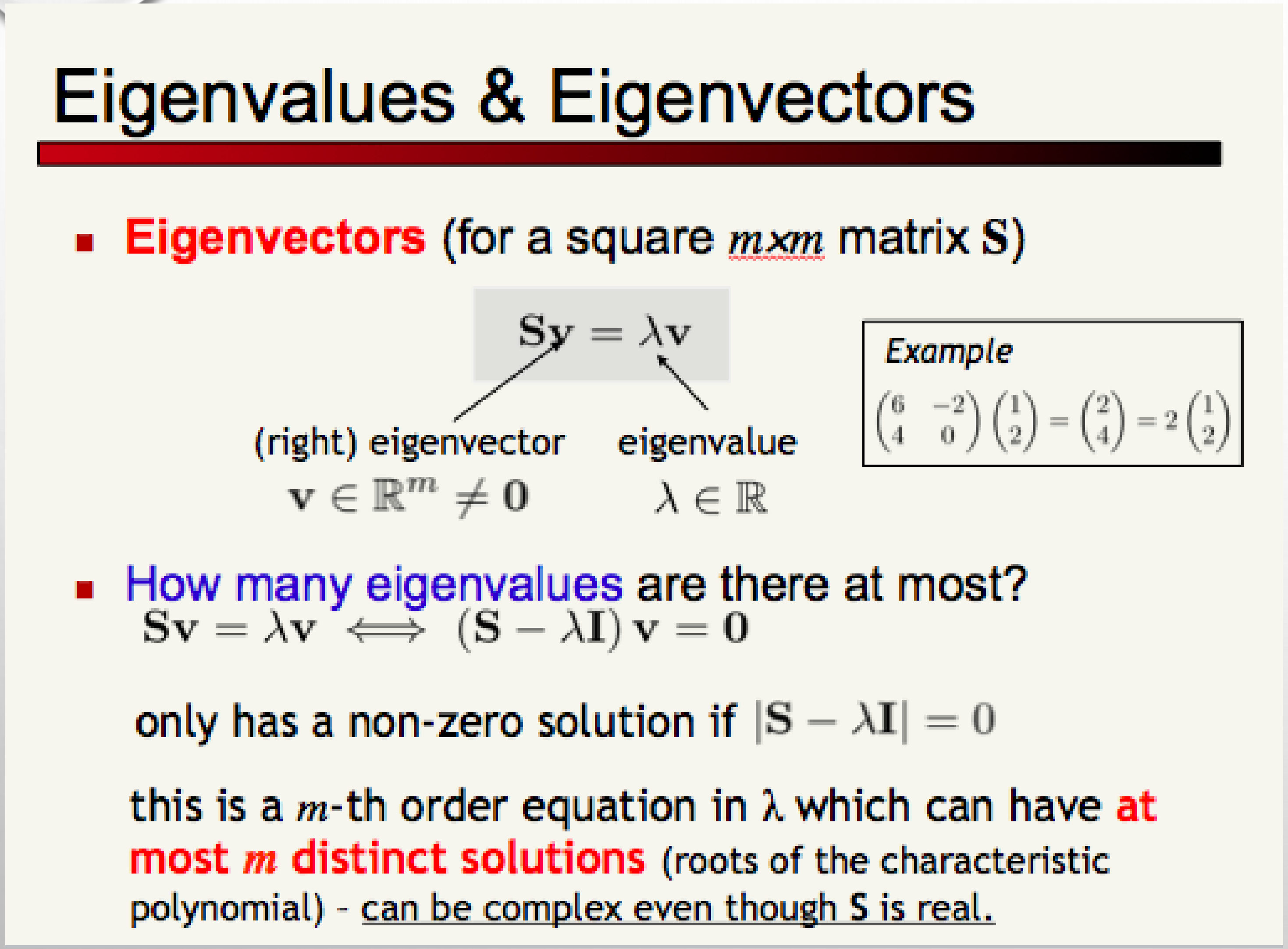
Now we can use the simulated data to learn something about the effects of collinearity when fitting multiple linear regressions. We will first fit two models using two predictors with low correlation between them, and then fit a third model with three predictors where pred1 and pred2 are highly correlated with each other.

```
# Model 1: predictors 1 and 3 (correlation is 0.28)
m1 = lm(resp ~ pred1 + pred3, rdata)
summary(m1)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.07536    0.06812   1.106  0.27133
pred1        0.67316    0.06842   9.838 2.99e-16 ***
pred3        0.20920    0.07253   2.884  0.00483 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6809 on 97 degrees of freedom
Multiple R-squared: 0.5762, Adjusted R-squared: 0.5675
F-statistic: 65.95 on 2 and 97 DF,  p-value: < 2.2e-16
```



```
# Model 3: correlation between predictors 1 and 2 is 0.96
m3 = lm(resp ~ pred1 + pred2 + pred3, rdata)
summary(m3)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.06421    0.06676   0.962  0.33856
pred1        0.16844    0.22560   0.747  0.45712
pred2        0.52525    0.22422   2.343  0.02122 *
pred3        0.19584    0.07114   2.753  0.00706 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6657 on 96 degrees of freedom
Multiple R-squared: 0.5991, Adjusted R-squared: 0.5866
F-statistic: 47.83 on 3 and 96 DF,  p-value: < 2.2e-16

# Variance inflation
vif(m3)

    pred1     pred2     pred3
12.373826 12.453165  1.094875
```

In my example it is possible to see the huge increase for the standard error for pred1 and pred2, when we use both highly correlated explanatory variables in model 3. In addition, model fit does not improve for model 3.



Eigenvalues & Eigenvectors

- **Eigenvectors** (for a square *mxm* matrix **S**)

$$\mathbf{S}\mathbf{y} = \lambda\mathbf{v}$$

(right) eigenvector $\mathbf{v} \in \mathbb{R}^m \neq \mathbf{0}$

eigenvalue $\lambda \in \mathbb{R}$

Example

$$\begin{pmatrix} 6 & -2 \\ 4 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} = 2\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

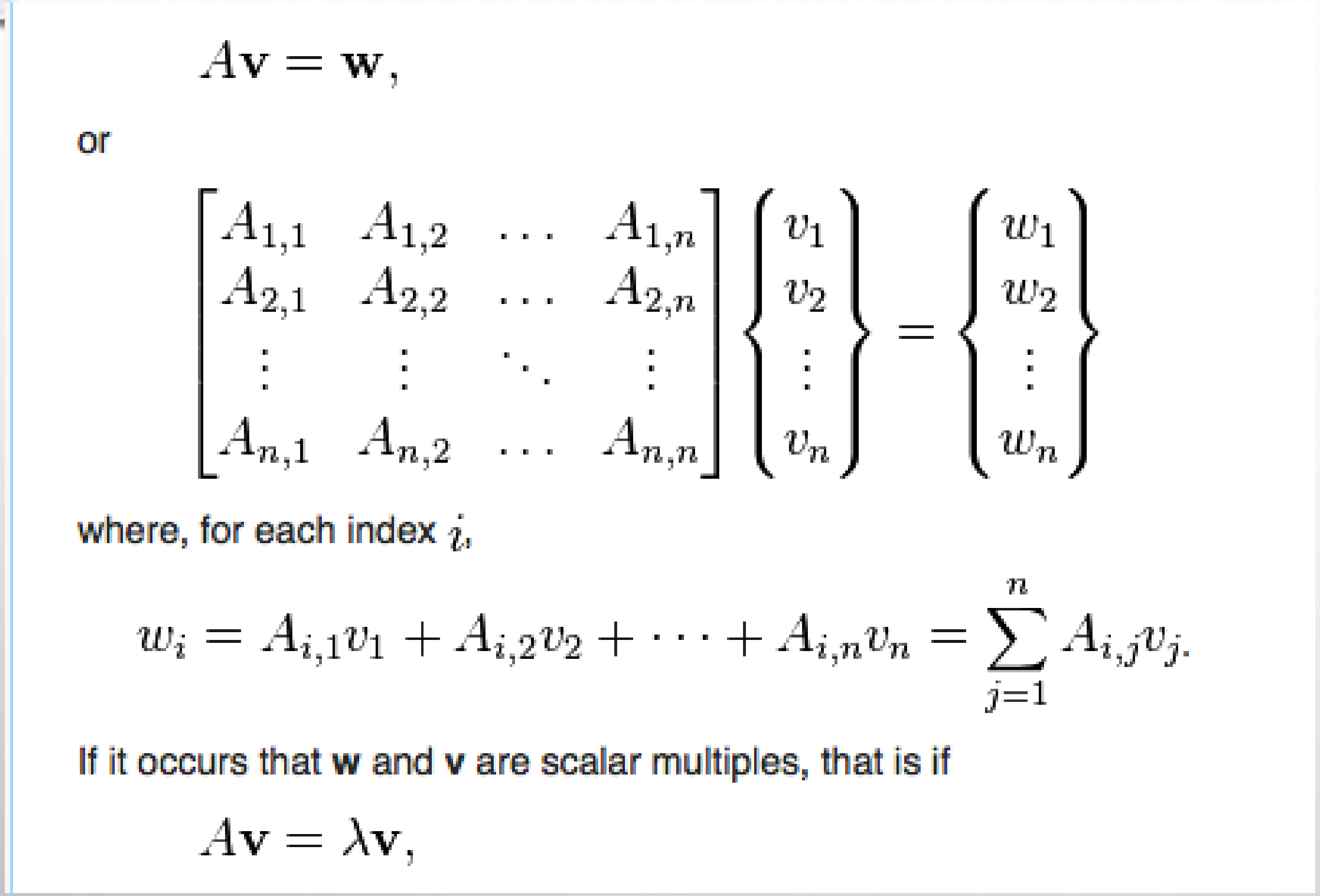
- How many eigenvalues are there at most?

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v} \iff (\mathbf{S} - \lambda\mathbf{I})\,\mathbf{v} = \mathbf{0}$$

only has a non-zero solution if $|\mathbf{S} - \lambda\mathbf{I}| = 0$

this is a m-th order equation in λ which can have **at most m distinct solutions** (roots of the characteristic polynomial) - can be complex even though S is real.

slide from: https://web.stanford.edu/class/cs276a/handouts/lecture15.ppt



$$A\mathbf{v} = \mathbf{w},$$

or

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \dots & A_{n,n} \end{bmatrix} \begin{Bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{Bmatrix} = \begin{Bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{Bmatrix}$$

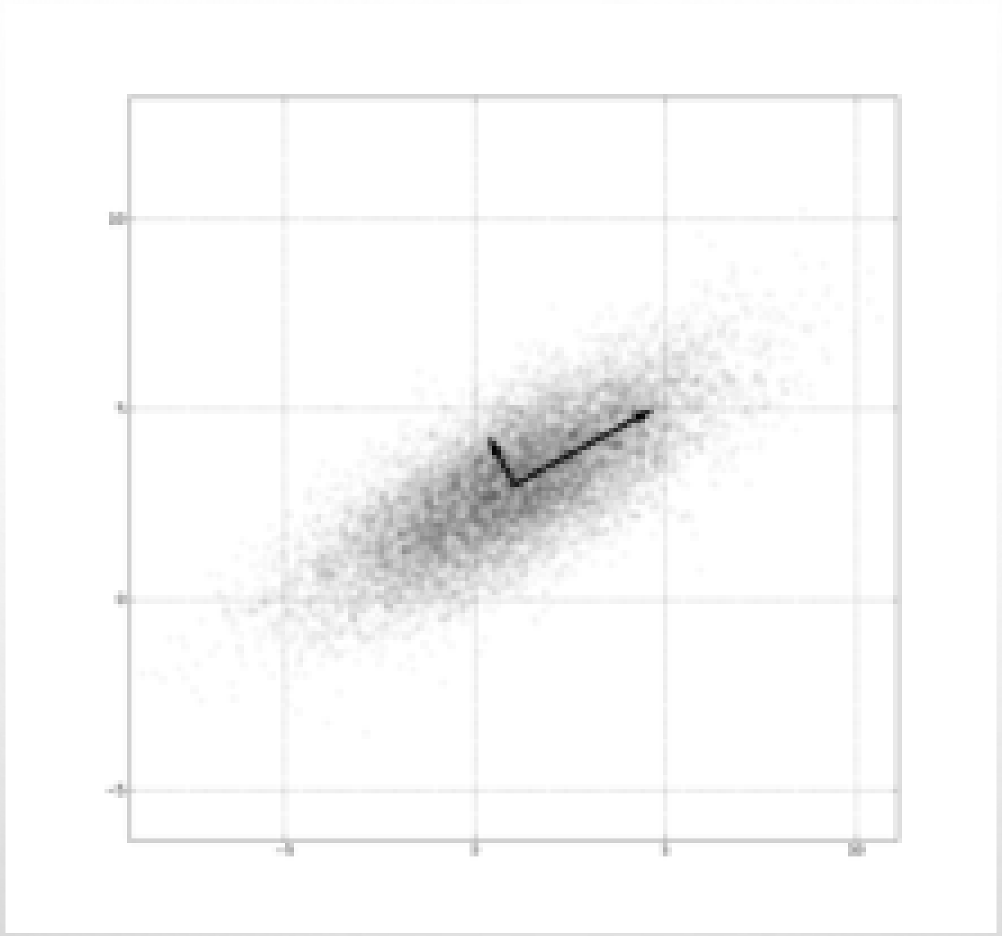
where, for each index i,

$$w_i = A_{i,1}v_1 + A_{i,2}v_2 + \cdots + A_{i,n}v_n = \sum_{j=1}^{n} A_{i,j}v_j.$$

If it occurs that **w** and **v** are scalar multiples, that is if

$$A\mathbf{v} = \lambda\mathbf{v},$$

Review of factorization (via understanding PCA) and eigenvalues



How Eigenvalues relate to Principal Components

```
data a(type=corr);
     input _name_ $ _type_ $ x1-x3;
     cards;
 .   MEAN      1     1       1    1
 .   STD       1     1       1    1
 .   N        100   100    100    100
 x1  CORR   1.00       .       .    .
 x2  CORR    .70   1.00        .     .
 x3  CORR    .70    .95   1.00    .
 x4  CORR    .50    0.3   0.3    1.0
run;
```

```
/*Remember to highlight the macro and the run together when you run the sas)
%MACRO RMNC (DATA=,OUT=,SEED=0);


 /* obtain the names of the random variables to be generated. */
 /* the names are stored in macro variables V1, V2,...        */
 /* macro variable VNAMES has all these variable names        */
 /* concatenated into one long string.                        */


 PROC CONTENTS DATA=&DATA(DROP=_TYPE_ _NAME_) OUT=_DATA_(KEEP=NAME) NOPRINT;
      RUN;
 DATA _DATA_;
      SET _LAST_ END=END;
      RETAIN N 0;
      N=N+1;
      V=COMPRESS('V'||COMPRESS(PUT(N,6.0)));
      CALL SYMPUT(V,NAME);
      IF END THEN CALL SYMPUT('NV',LEFT(PUT(N,6.)));
      RUN;
 %LET VNAMES=&V1;
 %DO I=2 %TO &NV;
     %LET VNAMES=&VNAMES &&V&I;
 %END;
```

```
/* obtain the matrix of factor patterns and other statistics. */


 PROC FACTOR DATA=&DATA NFACT=&NV NOPRINT
             OUTSTAT=_PTTRN_(
WHERE=(_TYPE_ IN ('MEAN','STD','N','PATTERN')));
      RUN;


 /* generate the random numbers.*/


 %LET NV2=%EVAL(&NV*&NV);
 DATA &OUT(KEEP=&VNAMES);


      /* rename the variables to be generated to V1, V2,... in order */
      /* to avoid any interference with the data step variables.     */


      SET _PTTRN_(KEEP=&VNAMES _TYPE_ RENAME=(
      %DO I=1 %TO &NV;
                                                 &&V&I=V&I
                                                           %END;
                                                          )) END=LASTFACT;

      RETAIN;
```

```
        /* read and store the matrix of factor patterns. */

  IF _TYPE_='PATTERN' THEN DO; DO I=1 TO &NV;


    /* here we utilize the fact that the  */
    /* observations of the factor pattern */
    /* start at observation #4.           */

                                            FPATTERN(_N_-3,I)=V(I);
                                         END;
                                     END;

        /* read and store the means. */


        IF _TYPE_='MEAN' THEN DO; DO I=1 TO &NV;
                                         VMEAN(I)=V(I);
                                     END;
                                 END;


        /* read and store the standard deviations. */


        IF _TYPE_='STD' THEN DO; DO I=1 TO &NV;
                                        VSTD(I)=V(I);
                                    END;
```

```
/* read and store the number of observations. */

     IF _TYPE_='N' THEN NNUMBERS=V(1);

     /* all necessary statistics have been read and stored. */
     /* start generating the random numbers.                */

     IF LASTFACT THEN DO;

        /* set up labels for the random variables. The labels */
        /* are stored in macro variables LBL1, LBL2,... and   */
        /* used in the subsequent PROC DATASETS.              */

        %DO I=1 %TO &NV;
            LBL="ST.NORMAL VAR., M="||COMPRESS(PUT(VMEAN(&I),BEST8.))||
                ", STD="||COMPRESS(PUT(VSTD(&I), BEST8.));
            CALL SYMPUT("LBL&I",LBL);
        %END;
```

```
      DO K=1 TO NNUMBERS;


         /* generate the initial random numbers of standard   */
         /* normal distribution. Store them in array 'VTEMP.' */


         DO I=1 TO &NV;
            VTEMP(I)=RANNOR(&SEED);
         END;


         /* impose the intercorrelation on each variable. The */
         /* transformed variables are stored in array 'V'.    */


         DO I=1 TO &NV;
            V(I)=0;
            DO J=1 TO &NV;
               V(I)=V(I)+VTEMP(J)*FPATTERN(J,I);
            END;
         END;


         /* transform the random variables so they will have */
         /* means and standard deviations as requested.      */


         DO I=1 TO &NV;
            V(I)=VSTD(I)*V(I)+VMEAN(I);
         END;
         OUTPUT;
      END;
   END;
```

```
/* rename V1,V2,... to the requested variable names. */

      RENAME           %DO I=1 %TO &NV;
             V&I=&&V&I
                       %END;
             ;
      RUN;


 /* set the label of each random variable. The label contains */
 /* the mean and standard deviation of the variable.          */


 PROC DATASETS NOLIST;
      MODIFY &OUT;
      LABEL %DO I=1 %TO &NV;
                &&V&I="&&LBL&I"
            %END;
            ;
      RUN;
%MEND;
************************************************************;
%RMNC(data=a,out=b,seed=123);
run;
```

FOR THIS PRESENTATION, WE WILL LEARN HOW TO GENERATE SIMULATED DATA IN R GIVEN A SPECIFIC COVARIANCE MATRIX OF THE DATA. I'M USING THIS TO EXAMINE POSSIBILITIES FOR EXTRA SUM OF SQUARES ANALYSIS (TO SEE IF ADDITIONAL RESIDUAL TERMS ARE WORTH ADDING TO A REGRESSION (WHEN DATA FOR A SPECIFIC COVARIANCE STRUCTURE IS GIVEN)

- The Cholesky decomposition corresponds to the unique lower triangular matrix C
  - Which I'll write as L to emphasise it's lower triangular
- Partition Σ and L as
- Then
  - $L_{11}L_{11}^T = \Sigma_{11}$, $L_{22}L_{22}^T = \Sigma_{22.1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{21}^T$
  - L^{-1} is also lower triangular
- Therefore
  - The first p elements of $L^{-1}X$ decompose the variance matrix of the first p elements of X
  - Remaining elements decompose the residual variance of the remaining elements of X conditional on the first p