tharunte_Homework2

2024-09-26

8. In Section 12.2.3, a formula for calculating PVE was given in Equation 12.10. We also saw that the PVE can be obtained using the sdev output of the prcomp() function. On the USArrests data, calculate PVE in two ways:

(a) Using the sdev output of the prcomp() function, as was done in Section 12.2.3.

LAB 12.5.1 & Applied 8. a)

Loading the dataset

```
data("USArrests")
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

Row names of the data set

```
states <- row.names(USArrests)
states
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
## [33] "North Carolina" "North Dakota"   "Ohio"           "Oklahoma"
```

```
## [37] "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
## [41] "South Dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West Virginia"
## [49] "Wisconsin"      "Wyoming"
```

Column names

```
names(USArrests)
```

```
## [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

Finding the mean of all columns

```
apply(USArrests, 2, mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

Finding variance for the columns

```
apply(USArrests, 2, var)
```

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

Applying PCA and scaling

```
pr.out <- prcomp(USArrests, scale. =  TRUE)
pr.out
```

```
## Standard deviations (1, .., p=4):
## [1] 1.5748783 0.9948694 0.5971291 0.4164494
##
## Rotation (n x k) = (4 x 4):
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995 -0.4181809  0.3412327  0.64922780
## Assault  -0.5831836 -0.1879856  0.2681484 -0.74340748
## UrbanPop -0.2781909  0.8728062  0.3780158  0.13387773
## Rape     -0.5434321  0.1673186 -0.8177779  0.08902432
```

```
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
pr.out$center
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
pr.out$scale
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

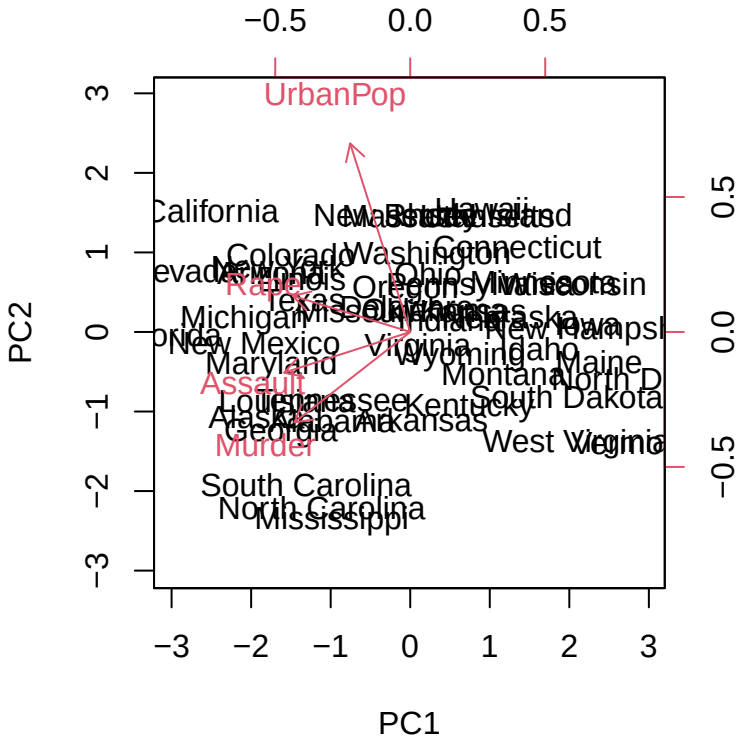
```
pr.out$rotation
```

```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995 -0.4181809  0.3412327  0.64922780
## Assault  -0.5831836 -0.1879856  0.2681484 -0.74340748
## UrbanPop -0.2781909  0.8728062  0.3780158  0.13387773
## Rape     -0.5434321  0.1673186 -0.8177779  0.08902432
```

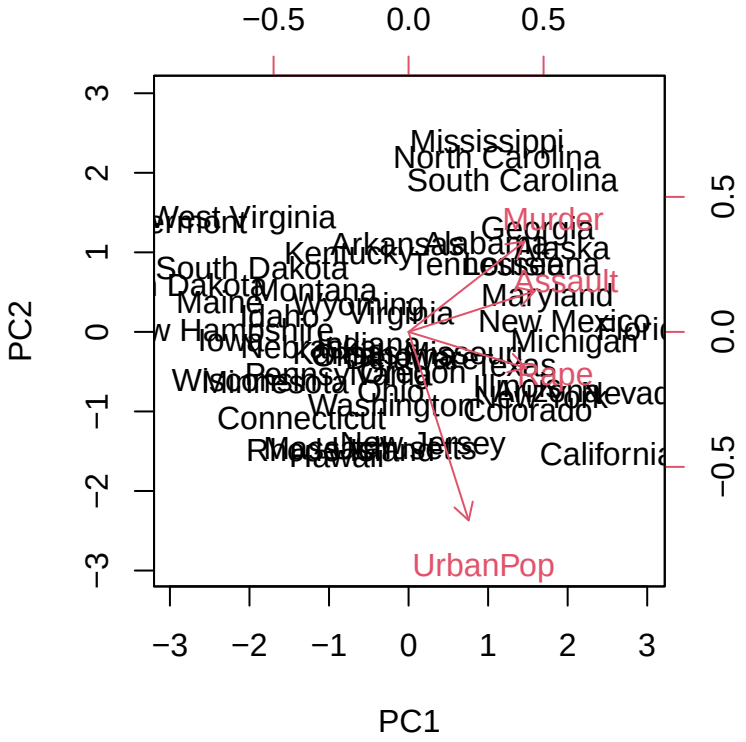
```
dim(pr.out$x)
```

```
## [1] 50  4
```

```
biplot(pr.out, scale =0)
```



```
pr.out$rotation = - pr.out$rotation
pr.out$x = - pr.out$x
biplot(pr.out , scale = 0)
```




```
pr.out$sdev
```

```
## [1] 1.5748783 0.9948694 0.5971291 0.4164494
```

```
pr.var <- pr.out$sdev^2
pr.var
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

Calculating PVE

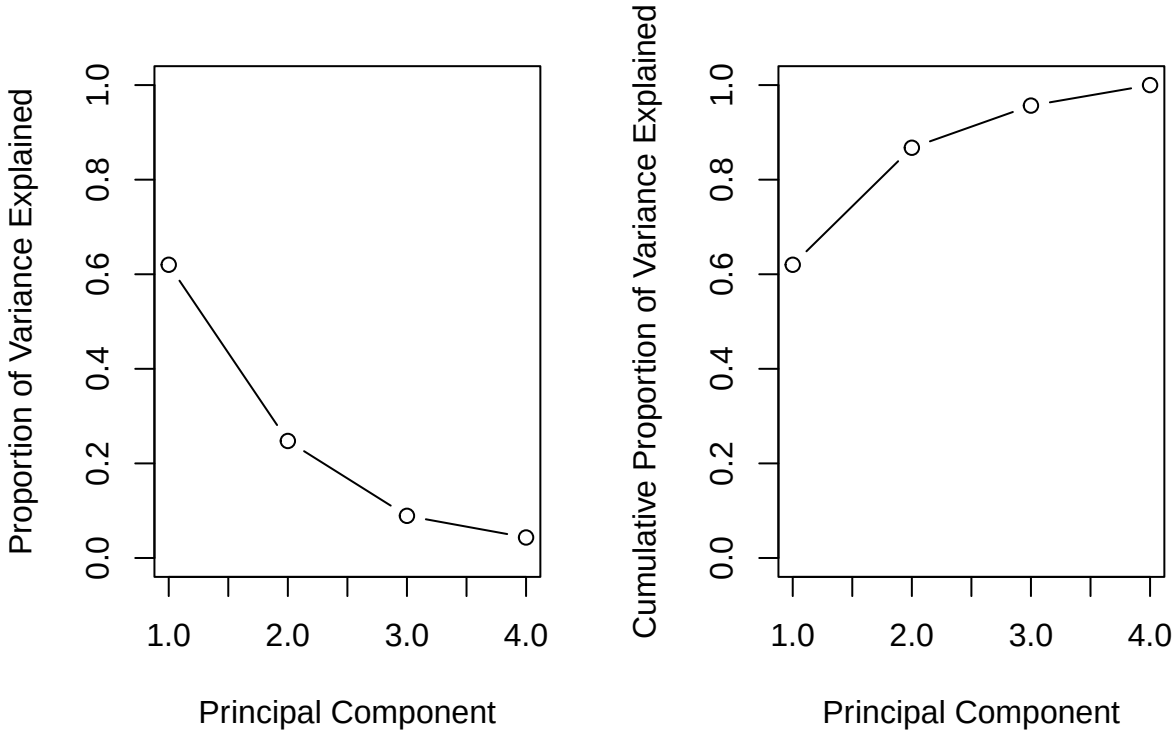
```
PVE <- pr.var/sum(pr.var)
PVE
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

```
par ( mfrow = c (1 , 2) )
plot ( PVE , xlab = "Principal Component" ,
 ylab = "Proportion of Variance Explained" , ylim = c (0 , 1) ,
 type = "b" )
 3
```

```
## [1] 3
```

```
plot ( cumsum ( PVE ) , xlab = "Principal Component" ,
 ylab = "Cumulative Proportion of Variance Explained" ,
 ylim = c (0 , 1) , type = "b" )
```



8b.By applying Equation 12.10 directly. That is, use the prcomp() function to compute the principal component loadings. Then, use those loadings in Equation 12.10 to obtain the PVE.

Get the principal component loadings (rotation matrix)

```
loadings <- pr.out$rotation
loadings
```

```
##                PC1        PC2        PC3         PC4
## Murder   0.5358995  0.4181809 -0.3412327 -0.64922780
## Assault  0.5831836  0.1879856 -0.2681484  0.74340748
## UrbanPop 0.2781909 -0.8728062 -0.3780158 -0.13387773
## Rape     0.5434321 -0.1673186  0.8177779 -0.08902432
```

Center and scale the dataset

```
scaled_data <- scale(USArrests)
```

```
p <- ncol(scaled_data)  # Number of variables
```

Sum of squared values of the scaled data

```
ss_data <- sum(scaled_data^2)
ss_data
```

```
## [1] 196
```

Calculate PVE for each principal component using the loadings (manual calculation)

pve_b contains the PVE values calculated using Equation 12.10.

```
pve_b <- numeric(p)

for (m in 1:p) {
  z_im <- scaled_data %*% loadings[,m]
  ss_scores_m <- sum(z_im^2)
  pve_b[m] <- ss_scores_m / ss_data
}

# Output the calculated PVE
pve_b
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

The two methods giving the same results, confirming that both approaches are valid for calculating the proportion of variance explained.

LAB 12.5.2 Matrix Completion

```r
X<-data.matrix(scale(USArrests))
pcob <- prcomp(X)
summary(pcob)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.5749 0.9949 0.59713 0.41645
## Proportion of Variance 0.6201 0.2474 0.08914 0.04336
## Cumulative Proportion  0.6201 0.8675 0.95664 1.00000
```

Applying Singular Value Decomposition

```r
sX <- svd(X)
names(sX)
```

```
## [1] "d" "u" "v"
```

```r
round(sX$v, 3)
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,] -0.536 -0.418  0.341  0.649
## [2,] -0.583 -0.188  0.268 -0.743
## [3,] -0.278  0.873  0.378  0.134
## [4,] -0.543  0.167 -0.818  0.089
```

Loadings

```r
pcob$rotation
```

```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995 -0.4181809  0.3412327  0.64922780
## Assault  -0.5831836 -0.1879856  0.2681484 -0.74340748
## UrbanPop -0.2781909  0.8728062  0.3780158  0.13387773
## Rape     -0.5434321  0.1673186 -0.8177779  0.08902432
```

```r
t(sX$d * t(sX$u))
```

```
##              [,1]        [,2]        [,3]         [,4]
##  [1,] -0.97566045 -1.12200121  0.43980366  0.154696581
##  [2,] -1.93053788 -1.06242692 -2.01950027 -0.434175454
##  [3,] -1.74544285  0.73845954 -0.05423025 -0.826264240
##  [4,]  0.13999894 -1.10854226 -0.11342217 -0.180973554
##  [5,] -2.49861285  1.52742672 -0.59254100 -0.338559240
##  [6,] -1.49934074  0.97762966 -1.08400162  0.001450164
##  [7,]  1.34499236  1.07798362  0.63679250 -0.117278736
##  [8,] -0.04722981  0.32208890  0.71141032 -0.873113315
```

```
##  [9,] -2.98275967 -0.03883425  0.57103206 -0.095317042
## [10,] -1.62280742 -1.26608838  0.33901818  1.065974459
## [11,]  0.90348448  1.55467609 -0.05027151  0.893733198
## [12,]  1.62331903 -0.20885253 -0.25719021 -0.494087852
## [13,] -1.36505197  0.67498834  0.67068647 -0.120794916
## [14,]  0.50038122  0.15003926 -0.22576277  0.420397595
## [15,]  2.23099579  0.10300828 -0.16291036  0.017379470
## [16,]  0.78887206  0.26744941 -0.02529648  0.204421034
## [17,]  0.74331256 -0.94880748  0.02808429  0.663817237
## [18,] -1.54909076 -0.86230011  0.77560598  0.450157791
## [19,]  2.37274014 -0.37260865  0.06502225 -0.327138529
## [20,] -1.74564663 -0.42335704  0.15566968 -0.553450589
## [21,]  0.48128007  1.45967706  0.60337172 -0.177793902
## [22,] -2.08725025  0.15383500 -0.38100046  0.101343128
## [23,]  1.67566951  0.62590670 -0.15153200  0.066640316
## [24,] -0.98647919 -2.36973712  0.73336290  0.213342049
## [25,] -0.68978426  0.26070794 -0.37365033  0.223554811
## [26,]  1.17353751 -0.53147851 -0.24440796  0.122498555
## [27,]  1.25291625  0.19200440 -0.17380930  0.015733156
## [28,] -2.84550542  0.76780502 -1.15168793  0.311354436
## [29,]  2.35995585  0.01790055 -0.03648498 -0.032804291
## [30,] -0.17974128  1.43493745  0.75677041  0.240936580
## [31,] -1.96012351 -0.14141308 -0.18184598 -0.336121113
## [32,] -1.66566662  0.81491072  0.63661186 -0.013348844
## [33,] -1.11208808 -2.20561081  0.85489245 -0.944789648
## [34,]  2.96215223 -0.59309738 -0.29824930 -0.251434626
## [35,]  0.22369436  0.73477837  0.03082616  0.469152817
## [36,]  0.30864928  0.28496113  0.01515592  0.010228476
## [37,] -0.05852787  0.53596999 -0.93038718 -0.235390872
## [38,]  0.87948680  0.56536050  0.39660218  0.355452378
## [39,]  0.85509072  1.47698328  1.35617705 -0.607402746
## [40,] -1.30744986 -1.91397297  0.29751723 -0.130145378
## [41,]  1.96779669 -0.81506822 -0.38538073 -0.108470512
## [42,] -0.98969377 -0.85160534 -0.18619262  0.646302674
## [43,] -1.34151838  0.40833518  0.48712332  0.636731051
## [44,]  0.54503180  1.45671524 -0.29077592 -0.081486749
## [45,]  2.77325613 -1.38819435 -0.83280797 -0.143433697
## [46,]  0.09536670 -0.19772785 -0.01159482  0.209246429
## [47,]  0.21472339  0.96037394 -0.61859067 -0.218628161
## [48,]  2.08739306 -1.41052627 -0.10372163  0.130583080
## [49,]  2.05881199  0.60512507  0.13746933  0.182253407
## [50,]  0.62310061 -0.31778662  0.23824049 -0.164976866
```

```
pcob$x
```

```
##                       PC1         PC2         PC3          PC4
## Alabama        -0.97566045 -1.12200121  0.43980366  0.154696581
## Alaska         -1.93053788 -1.06242692 -2.01950027 -0.434175454
## Arizona        -1.74544285  0.73845954 -0.05423025 -0.826264240
## Arkansas        0.13999894 -1.10854226 -0.11342217 -0.180973554
## California     -2.49861285  1.52742672 -0.59254100 -0.338559240
## Colorado       -1.49934074  0.97762966 -1.08400162  0.001450164
## Connecticut     1.34499236  1.07798362  0.63679250 -0.117278736
## Delaware       -0.04722981  0.32208890  0.71141032 -0.873113315
```

```
## Florida        -2.98275967 -0.03883425  0.57103206 -0.095317042
## Georgia        -1.62280742 -1.26608838  0.33901818  1.065974459
## Hawaii          0.90348448  1.55467609 -0.05027151  0.893733198
## Idaho           1.62331903 -0.20885253 -0.25719021 -0.494087852
## Illinois       -1.36505197  0.67498834  0.67068647 -0.120794916
## Indiana         0.50038122  0.15003926 -0.22576277  0.420397595
## Iowa            2.23099579  0.10300828 -0.16291036  0.017379470
## Kansas          0.78887206  0.26744941 -0.02529648  0.204421034
## Kentucky        0.74331256 -0.94880748  0.02808429  0.663817237
## Louisiana      -1.54909076 -0.86230011  0.77560598  0.450157791
## Maine           2.37274014 -0.37260865  0.06502225 -0.327138529
## Maryland       -1.74564663 -0.42335704  0.15566968 -0.553450589
## Massachusetts   0.48128007  1.45967706  0.60337172 -0.177793902
## Michigan       -2.08725025  0.15383500 -0.38100046  0.101343128
## Minnesota       1.67566951  0.62590670 -0.15153200  0.066640316
## Mississippi    -0.98647919 -2.36973712  0.73336290  0.213342049
## Missouri       -0.68978426  0.26070794 -0.37365033  0.223554811
## Montana         1.17353751 -0.53147851 -0.24440796  0.122498555
## Nebraska        1.25291625  0.19200440 -0.17380930  0.015733156
## Nevada         -2.84550542  0.76780502 -1.15168793  0.311354436
## New Hampshire   2.35995585  0.01790055 -0.03648498 -0.032804291
## New Jersey     -0.17974128  1.43493745  0.75677041  0.240936580
## New Mexico     -1.96012351 -0.14141308 -0.18184598 -0.336121113
## New York       -1.66566662  0.81491072  0.63661186 -0.013348844
## North Carolina -1.11208808 -2.20561081  0.85489245 -0.944789648
## North Dakota    2.96215223 -0.59309738 -0.29824930 -0.251434626
## Ohio            0.22369436  0.73477837  0.03082616  0.469152817
## Oklahoma        0.30864928  0.28496113  0.01515592  0.010228476
## Oregon         -0.05852787  0.53596999 -0.93038718 -0.235390872
## Pennsylvania    0.87948680  0.56536050  0.39660218  0.355452378
## Rhode Island    0.85509072  1.47698328  1.35617705 -0.607402746
## South Carolina -1.30744986 -1.91397297  0.29751723 -0.130145378
## South Dakota    1.96779669 -0.81506822 -0.38538073 -0.108470512
## Tennessee      -0.98969377 -0.85160534 -0.18619262  0.646302674
## Texas          -1.34151838  0.40833518  0.48712332  0.636731051
## Utah            0.54503180  1.45671524 -0.29077592 -0.081486749
## Vermont         2.77325613 -1.38819435 -0.83280797 -0.143433697
## Virginia        0.09536670 -0.19772785 -0.01159482  0.209246429
## Washington      0.21472339  0.96037394 -0.61859067 -0.218628161
## West Virginia   2.08739306 -1.41052627 -0.10372163  0.130583080
## Wisconsin       2.05881199  0.60512507  0.13746933  0.182253407
## Wyoming         0.62310061 -0.31778662  0.23824049 -0.164976866
```

we are putting some random values in the data set

```
nomit <-  20
set.seed(15)
ina <- sample(seq(50), nomit)
inb <- sample(1:4, nomit, replace=TRUE)
Xna <- X
index.na <- cbind(ina, inb)
```

```r
Xna[index.na] <- NA
#index.na
Xna
```

```
##                    Murder     Assault    UrbanPop         Rape
## Alabama         1.24256408          NA -0.52090661 -0.003416473
## Alaska          0.50786248  1.10682252          NA  2.484202941
## Arizona         0.07163341  1.47880321  0.99898006  1.042878388
## Arkansas        0.23234938  0.23086801 -1.07359268 -0.184916602
## California      0.27826823          NA  1.75892340  2.067820292
## Colorado        0.02571456  0.39885929  0.86080854  1.864967207
## Connecticut    -1.03041900 -0.72908214  0.79172279 -1.081740768
## Delaware       -0.43347395  0.80683810  0.44629400 -0.579946294
## Florida         1.74767144  1.97077766  0.99898006  1.138966691
## Georgia         2.20685994  0.48285493 -0.38273510           NA
## Hawaii         -0.57123050 -1.49704226  1.20623733 -0.110181255
## Idaho          -1.19113497 -0.60908837          NA -0.750769945
## Illinois        0.59970018  0.93883125  1.20623733  0.295524916
## Indiana        -0.13500142 -0.69308401 -0.03730631 -0.024769429
## Iowa           -1.28297267 -1.37704849 -0.58999237 -1.060387812
## Kansas         -0.41051452 -0.66908525  0.03177945 -0.345063775
## Kentucky        0.43898421 -0.74108152 -0.93542116 -0.526563903
## Louisiana       1.74767144  0.93883125  0.03177945  0.103348309
## Maine          -1.30593210 -1.05306531 -1.00450692 -1.434064548
## Maryland        0.80633501  1.55079947  0.10086521           NA
## Massachusetts  -0.77786532          NA  1.34440885 -0.526563903
## Michigan        0.99001041  1.01082751  0.58446551  1.480613993
## Minnesota              NA -1.18505846  0.03177945 -0.676034598
## Mississippi     1.90838741  1.05882502 -1.48810723 -0.441152078
## Missouri        0.27826823  0.08687549          NA  0.743936999
## Montana        -0.41051452          NA -0.86633540 -0.515887425
## Nebraska       -0.80082475 -0.82507715 -0.24456358 -0.505210947
## Nevada          1.01296983  0.97482938  1.06806582  2.644350114
## New Hampshire  -1.30593210 -1.36504911 -0.65907813 -1.252564419
## New Jersey     -0.08908257 -0.14111267  1.62075188 -0.259651949
## New Mexico      0.82929443  1.37080881  0.30812248  1.160319648
## New York        0.76041616  0.99882813          NA  0.519730957
## North Carolina  1.19664523  1.99477641 -1.41902147 -0.547916860
## North Dakota   -1.60440462 -1.50904164 -1.48810723           NA
## Ohio           -0.11204199 -0.60908837  0.65355127  0.017936483
## Oklahoma       -0.27275797 -0.23710769  0.16995096 -0.131534211
## Oregon         -0.66306820 -0.14111267          NA  0.861378259
## Pennsylvania   -0.34163624          NA  0.44629400 -0.676034598
## Rhode Island   -1.00745957  0.03887798  1.48258036 -1.380682157
## South Carolina  1.51807718  1.29881255 -1.21176419  0.135377743
## South Dakota   -0.91562187 -1.01706718 -1.41902147 -0.900240639
## Tennessee       1.24256408          NA -0.45182086  0.605142783
## Texas                  NA  0.36286116  0.99898006  0.455672088
## Utah           -1.05337842 -0.60908837          NA  0.178083656
## Vermont        -1.28297267 -1.47304350 -2.31713632 -1.071064290
## Virginia        0.16347111 -0.17711080          NA -0.056798864
## Washington             NA -0.30910395  0.51537975  0.530407436
## West Virginia  -0.47939280 -1.07706407 -1.83353601 -1.273917376
```

```
## Wisconsin      -1.19113497 -1.41304662  0.03177945 -1.113770203
## Wyoming        -0.22683912 -0.11711392          NA -0.601299251
## attr(,"scaled:center")
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
## attr(,"scaled:scale")
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

```
fit.svd <- function(X, M = 1){
 svdob <- svd(X)
 with(svdob,
    u[, 1:M, drop = FALSE ] %*%
    (d[1:M] * t(v[, 1:M, drop = FALSE]))
)
}
```

Removing Na with the mean of the data

```
Xhat <- Xna
xbar <- colMeans(Xna,na.rm = TRUE)
Xhat[index.na] <- xbar[inb]
```

```
thresh <- 1e-7
rel_err <- 1
iter <- 0
ismiss <- is.na(Xna)  # placing True at NA and False at remaining places
mssold <- mean (( scale ( Xna , xbar , FALSE ) [! ismiss ]) ^2)
mss0 <- mean ( Xna [! ismiss ]^2)
```

mssold thus stores the mean of the squared deviations of the non-missing data points from their corresponding column means. mss0 stores the mean of the squared values of the non-missing entries in Xna.

```
while ( rel_err > thresh ) {
iter <- iter + 1
# Step 2( a )
Xapp <- fit.svd ( Xhat , M = 1)
# Step 2( b )
Xhat [ ismiss ] <- Xapp [ ismiss ]
# Step 2( c )
mss <- mean ((( Xna - Xapp ) [! ismiss ]) ^2)
rel_err <- ( mssold - mss ) / mss0
mssold <- mss
cat ( "Iter :" , iter , "MSS :" , mss ,
"Rel.Err :" , rel_err , "\ n" ) }
```

```
## Iter : 1 MSS : 0.3821695 Rel.Err : 0.6194004  nIter : 2 MSS : 0.3705046 Rel.Err : 0.01161265  nIter
```

```
cor(Xapp[ismiss],X[ismiss])
```

```
## [1] 0.6535043
```

Notes: # **Matrix Completion Explained**

Matrix completion is a technique used to fill in missing entries in a data matrix, often using a method that captures the underlying patterns and relationships within the data. Here's a simple explanation of the matrix completion process, as illustrated by the code you provided.

Step 1: Understanding Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)

Principal Component Analysis (PCA)

PCA is a technique that reduces the dimensionality of a data matrix by finding the main components (directions) that capture the most variation in the data. When data is complete, we can use PCA directly to identify these main components.

Singular Value Decomposition (SVD)

SVD is a mathematical method similar to PCA. It decomposes a matrix X into three matrices:

$$X = U \cdot D \cdot V^T$$

- U: A matrix containing left singular vectors - D: A diagonal matrix with singular values - V: A matrix containing right singular vectors

This decomposition helps us understand the structure of the matrix X.

Step 2: Dealing with Missing Data

When data has missing values, we can't directly apply PCA or SVD. Instead, we use an iterative approach to estimate the missing values.

The Iterative Algorithm for Matrix Completion

The goal of matrix completion is to estimate missing values by finding an approximate version of the original matrix using a specified number of principal components.

Detailed Steps in the Algorithm:

1. **Initialization:**
   - Create a matrix `Xna` with some missing values. We replace the missing values with the mean of the corresponding column to create an initial matrix `Xhat`.
2. **Fitting a Low-Rank Matrix (SVD-based Approximation):**
   - Define the `fit.svd` function: This function uses SVD to find an approximation of `Xhat` using only the first M principal components. It reconstructs a low-rank version of `Xhat` that captures the most important patterns.
3. **Iterative Refinement:**
   - **Step 2(a)**: Apply `fit.svd` to `Xhat` to get an approximate matrix `Xapp` using the first principal component.
   - **Step 2(b)**: Replace the missing values in `Xhat` with the corresponding values from `Xapp`.

- **Step 2(c)**: Calculate the Mean Squared Error (MSS) to monitor the difference between the current approximation `Xapp` and the original observed values `Xna`. Compute the relative error to see how much improvement has been made.

4. **Repeat the Iterative Steps Until Convergence:**
    - The loop continues until the relative error falls below a small threshold, indicating that the matrix has been completed to a satisfactory level.

Final Evaluation

- After the iterations stop, you can compare the filled-in values in `Xapp` (the reconstructed matrix) with the original data to see how well the missing values have been imputed.

Conclusion

The matrix completion method leverages the idea that many datasets can be effectively approximated by a lower-dimensional structure (i.e., a few principal components). By iteratively refining the estimates of missing values using the dominant patterns in the data, we arrive at a completed matrix that best represents the underlying structure of the original data. This approach is particularly useful when dealing with datasets that have missing entries, and it's a powerful technique for handling incomplete data in real-world scenarios.