Analysis in R

Instructor: Eric Dunford

Email: edunford@gmail.com

Overview

Today, we are going to go in the weeds regarding some statistical manipulations that can be performed in `R`. The goal is to offer a general idea of

1. the types of summary statistics available,
2. visualizing data summaries (most of which w are already familiar with)
3. how to deal with statistical objects in `R`, and
4. to go into the particulars on various types of analyses in `R`

Descriptive Statistics

There are a wealth of useful summary operators that are built into `R`.

```
mean() # Mean
sd() # Standard Deviation
var() # Variance
range() # Range of a varible
min() # Minimum
max() # Maximum
median() # Median
quantile() # Distribution Quantiles
fivenum() # Five Number Summary
colMeans() # Means by Column
rowMeans() # Means by Row
table() # Counts by Category
```

. . . to name a few!

Summaries

As we've learned the `summary()` function in R, which offers a **concise summary of the distribution of a variable**.

```
summary(iris$Sepal.Length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
```

The `summary()` function works with a range of different objects (along with `plot()`).

We can also run this on a `data.frame` to get a summary of **each variable**.

```
summary(iris[,1:3])
```

```
##   Sepal.Length    Sepal.Width     Petal.Length
##  Min.   :4.300   Min.   :2.000   Min.   :1.000
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600
##  Median :5.800   Median :3.000   Median :4.350
##  Mean   :5.843   Mean   :3.057   Mean   :3.758
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100
##  Max.   :7.900   Max.   :4.400   Max.   :6.900
```

Correlations

We can retrieve the correlation between two variables using the `cor()` base function.

```
cor(iris$Sepal.Length,iris$Sepal.Width)
```

```
## [1] -0.1175698
```

To test if the correlation is statistically significant, we can use the `cor.test()` function.

```
cor.test(iris$Sepal.Length,iris$Sepal.Width)
```

```
##
##  Pearson's product-moment correlation
##
## data:  iris$Sepal.Length and iris$Sepal.Width
## t = -1.4403, df = 148, p-value = 0.1519
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.27269325  0.04351158
## sample estimates:
##        cor
## -0.1175698
```

Correlation Matrix

If we have a `data.frame` of numeric values, we can run `cor()` to get a **correlation matrix** – which tells us to what degree all the variables are correlated with every other variable.

```
iris %>% select(-Species) %>% cor(.)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

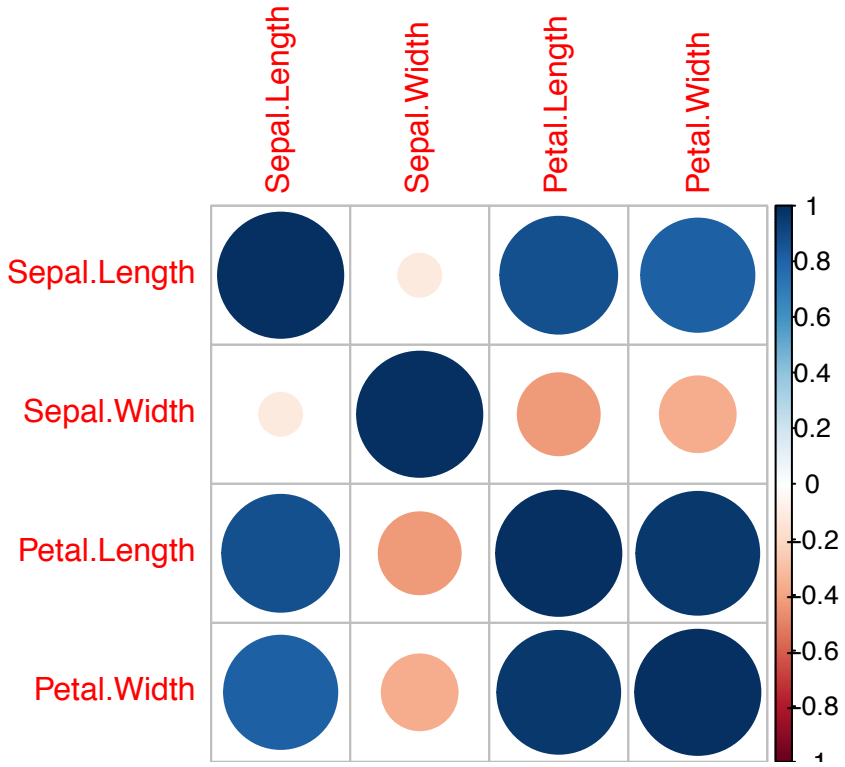
Visualizing the Correlation Matrix

The package `corrplot` offers a the function `corrplot()` which provides quick visualization of a correlation matrix.

```
require(corrplot)
```

```
## Loading required package: corrplot
```

```
iris %>% select(-Species) %>%
  cor(.) %>% corrplot(.)
```



Group Comparisons

We often need to deal with comparing the differences between groups. To examine the difference between two groups on some dimension we normally compare the means and see if they're different.

Here let's look at the difference in the `Sepal.Length` between the `setosa` and the `virginica` species.

```
iris %>% filter(Species!="versicolor") %>%
  group_by(Species) %>%
  summarize(mu=mean(Sepal.Length))
```

```
## # A tibble: 2 × 2
##     Species    mu
##      <fctr> <dbl>
## 1    setosa 5.006
## 2 virginica 6.588
```

Is that difference statistically meaningful? That is, is it not just random chance that we observe this difference? The `t.test()` function can help us answer this question by performing a difference of means test on our two groups.

```
# Group 1
g1 <- iris %>% filter(Species=="setosa")
# Group 2
g2 <- iris %>% filter(Species=="virginica")
```

Two-Sample T-Test

```
t.test(g1$Sepal.Length,g2$Sepal.Length)
```

```
##
##  Welch Two Sample t-test
##
## data:  g1$Sepal.Length and g2$Sepal.Length
## t = -15.386, df = 76.516, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.78676 -1.37724
## sample estimates:
## mean of x mean of y
##     5.006     6.588
```

We can alter the confidence level, if need be.

```
t.test(g1$Sepal.Length,g2$Sepal.Length,
       conf.level = .999)
```

```
##
##  Welch Two Sample t-test
##
## data:  g1$Sepal.Length and g2$Sepal.Length
## t = -15.386, df = 76.516, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 99.9 percent confidence interval:
##  -1.933876 -1.230124
## sample estimates:
## mean of x mean of y
##     5.006     6.588
```

One-Sample T-Test

Here let's say that we believe the "true" population mean to equal 5.1. How different is our sample mean from this value?

```
t.test(g1$Sepal.Length,mu=5.1)
```

```
##
##  One Sample t-test
##
## data:  g1$Sepal.Length
## t = -1.8857, df = 49, p-value = 0.06527
## alternative hypothesis: true mean is not equal to 5.1
## 95 percent confidence interval:
##  4.905824 5.106176
## sample estimates:
## mean of x
##     5.006
```

Cross Tabulation

A **cross tab** is a concise way of presenting the **joint distributions** of two or more variables. The information is usually displayed as a table (or contingency matrix).

We've already encountered quick ways of offering these types of summaries with the `table()` function. However, there are packages out there that offer more expansive cross tabs.

Here we'll look at one of these packages: `gmodels`

```
require(gmodels)
```

```
## Loading required package: gmodels
```

First, let's generate an indicator variable from the `iris` data which breaks the data up into "big" values (by some arbitrary condition).

```
df <- iris %>%
  mutate(big=as.numeric(Sepal.Length >= 5 & Petal.Length>=1.5))

table(df$big)
```

```
##
##   0   1
##  33 117
```

```
CrossTable(x=df$big,df$Species,
           prop.r = T,prop.c = F,prop.t = F,
           prop.chisq = F,
           chisq = T,format="SPSS")
```

```
##
##    Cell Contents
## |-------------------------|
## |                   Count |
## |             Row Percent |
## |-------------------------|
##
## Total Observations in Table:  150
##
##              | df$Species
##       df$big |     setosa  | versicolor  |  virginica  |  Row Total |
## -------------|-------------|-------------|-------------|------------|
##            0 |         31  |          1  |          1  |         33  |
##              |     93.939% |      3.030% |      3.030% |     22.000% |
## -------------|-------------|-------------|-------------|------------|
##            1 |         19  |         49  |         49  |        117  |
##              |     16.239% |     41.880% |     41.880% |     78.000% |
## -------------|-------------|-------------|-------------|------------|
## Column Total |         50  |         50  |         50  |        150  |
## -------------|-------------|-------------|-------------|------------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
```

```
## Chi^2 =  69.93007     d.f. =  2     p =  6.529475e-16
##
##
##
##        Minimum expected frequency: 11
```

Visualizing Relationships

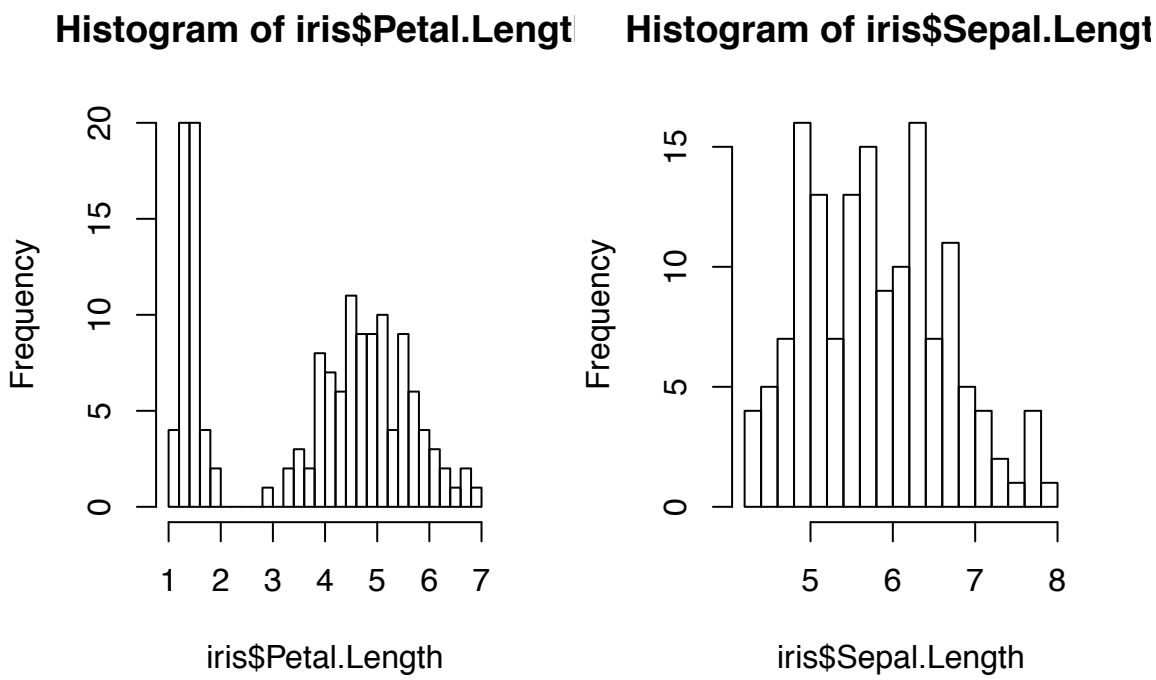
Lastly, some of the most powerful descriptive summaries are visual. Using the `hist()`, `barplot()` and `plot()` functions to specify relationships between variables are useful ways to visualize univariate and bivariate relationships.

Linear Regression in R

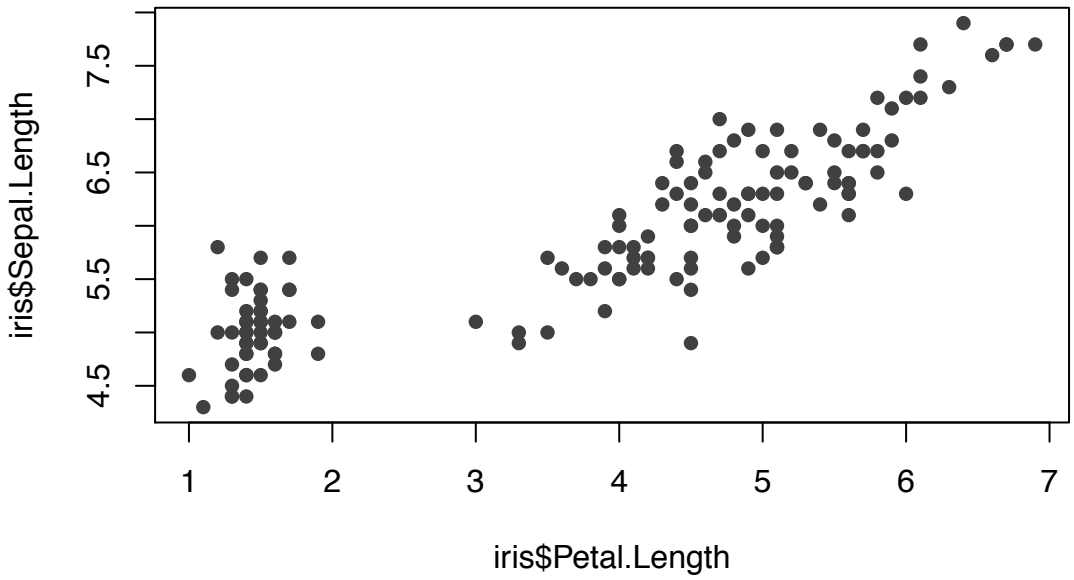
`R` was built for statistics, and to that end, it does its job well. The `lm()` function is a base function in `R` and offers everything one could need when dealing with the linear model.

For an example, let's look at the relationship between `Sepal.Length` and `Petal.Length`.

```
par(mfrow=c(1,2))
hist(iris$Petal.Length,breaks=25)
hist(iris$Sepal.Length,breaks=25)
```




```
plot(iris$Petal.Length,iris$Sepal.Length,
     col="grey25",pch=16)
```

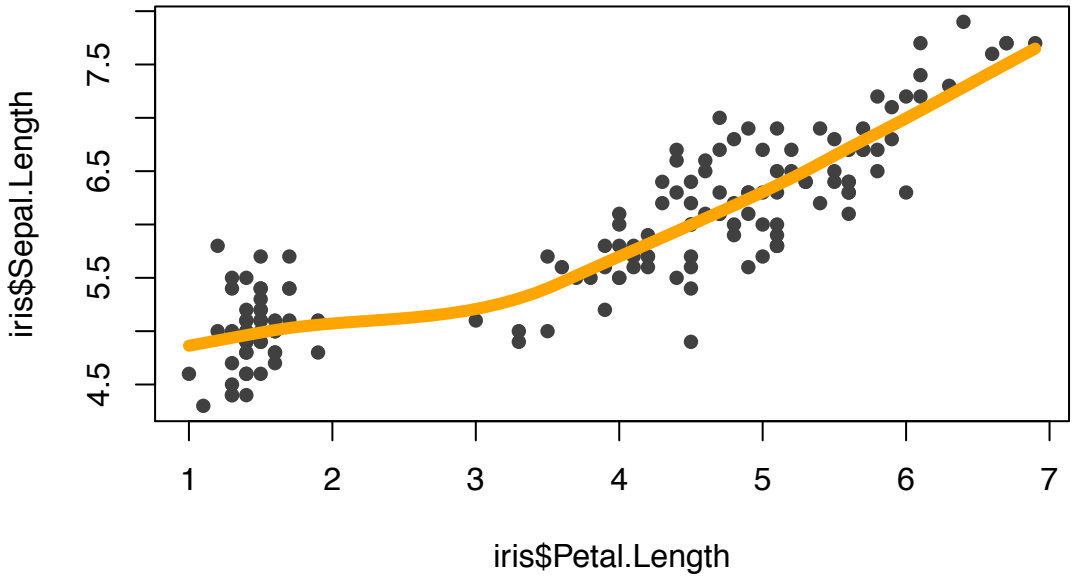


Fitting a loess curve

When dealing with ***bivariate*** relationships, it can be useful to visualize the relationship between two variables. One powerful way of doing this is by using a **loess curve**.

LOESS stands for "Locally Weighted Scatterplot Smoothing". It generates a smooth line through a scatter plot to help see relationships between two variables.

```
smoother <- loess.smooth(iris$Petal.Length,iris$Sepal.Length)
plot(iris$Petal.Length,iris$Sepal.Length,
     col="grey25",pch=16)
lines(smoother,lwd=6,col="orange")
```



Visually, it looks as if there is a strong correlation between the length of a flower's petals and sepal.

Let's look at the bivariate relationship between these two variables by modeling it linearly.

The **linear model** has a number of assumptions built into it (which we won't cover here), but in essence,

we're saying that the relationship between x and y is linear with a consistent rate of change (i.e. a line) and can be modeled as such.

$$Sepal.Length = \alpha + \beta * (Petal.Length) + \epsilon$$

Setting up the lm()

The `lm()` function requires a **formula**, which we specify using a tilde (`~`).

We set this up by saying `y` (outcome) is a function of `x` (predictor): `y ~ x`. We then specify our data object.

```
lm(Sepal.Length ~ Petal.Length, data=iris)
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Length, data = iris)
##
## Coefficients:
##  (Intercept)  Petal.Length
##       4.3066        0.4089
```

Much like everything else in `R`. We can allocate the output from a model to an object, which we can then reference later. This object is **loaded** with relevant information produced by `lm()`.

```
model1 <- lm(Sepal.Length ~ Petal.Length, data=iris)
str(model1)
```

```
## List of 12
##  $ coefficients : Named num [1:2] 4.307 0.409
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "Petal.Length"
##  $ residuals    : Named num [1:150] 0.2209 0.0209 -0.1382 -0.32 0.1209 ...
##   ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:150] -71.566 8.812 -0.155 -0.337 0.104 ...
##   ..- attr(*, "names")= chr [1:150] "(Intercept)" "Petal.Length" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:150] 4.88 4.88 4.84 4.92 4.88 ...
##   ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr   : num [1:150, 1:2] -12.2474 0.0816 0.0816 0.0816 0.0816 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:150] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "(Intercept)" "Petal.Length"
##   .. ..- attr(*, "assign")= int [1:2] 0 1
##   ..$ qraux: num [1:2] 1.08 1.1
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 1e-07
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 148
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = Sepal.Length ~ Petal.Length, data = iris)
##  $ terms        :Classes 'terms', 'formula'  language Sepal.Length ~ Petal.Length
##   .. ..- attr(*, "variables")= language list(Sepal.Length, Petal.Length)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
```

```
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "Sepal.Length" "Petal.Length"
##   .. .. .. ..$ : chr "Petal.Length"
##   .. ..- attr(*, "term.labels")= chr "Petal.Length"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(Sepal.Length, Petal.Length)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "Sepal.Length" "Petal.Length"
##  $ model        :'data.frame':   150 obs. of  2 variables:
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language Sepal.Length ~ Petal.Length
##   .. .. ..- attr(*, "variables")= language list(Sepal.Length, Petal.Length)
##   .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:2] "Sepal.Length" "Petal.Length"
##   .. .. .. .. ..$ : chr "Petal.Length"
##   .. .. ..- attr(*, "term.labels")= chr "Petal.Length"
##   .. .. ..- attr(*, "order")= int 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(Sepal.Length, Petal.Length)
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. .. ..- attr(*, "names")= chr [1:2] "Sepal.Length" "Petal.Length"
##  - attr(*, "class")= chr "lm"
```

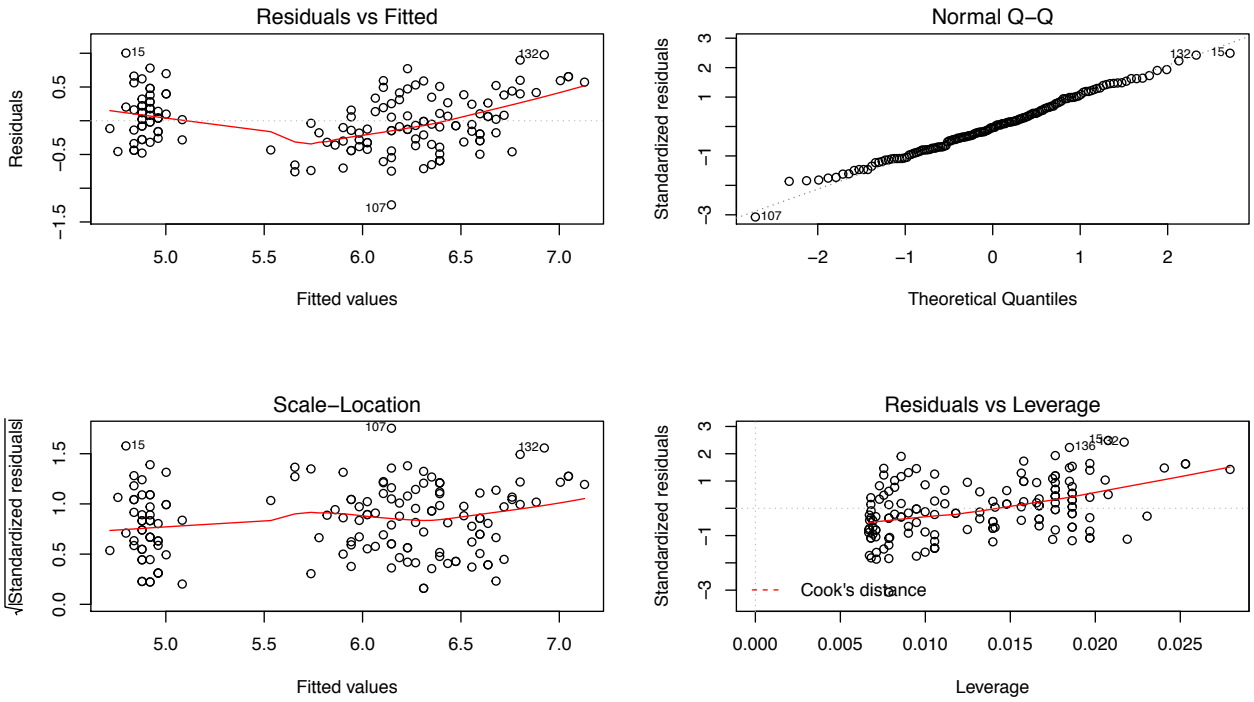
Assessing output

`lm()` overloads the `plot()` and `summary()` functions to produce grahics specific to the object output.

Specifically, `plot()` will print visual information regarding how well the data conforms to the assumptions that underpin the linear model. The function produces a total of four plots:

1. linearity test,
2. normality assumuption test,
3. equal variance test (to check for homoskedasticity)
4. pinpointing influencial cases.

```
par(mfrow=c(2,2))
plot(model1)
```



We use `summary()` with `lm` objects to get a concise report of our model results.

```
summary(model1)
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.24675 -0.29657 -0.01515  0.27676  1.00269
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.30660    0.07839   54.94   <2e-16 ***
## Petal.Length  0.40892    0.01889   21.65   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4071 on 148 degrees of freedom
## Multiple R-squared:   0.76,  Adjusted R-squared:  0.7583
## F-statistic: 468.6 on 1 and 148 DF,  p-value: < 2.2e-16
```

Auxilary Extraction Functions

There are a range of functions to help us extract key pieces of information from an `lm` object, such as our

- linear prediction (what the model predicts the value of y to be),
- residuals (how wrong our prediction is from the observed data),
- coefficients (constant and slope),
- goodness of fit (information criteria)

```
pred <- predict(model1)
head(as.matrix(pred))
```

```
##       [,1]
## 1 4.879095
## 2 4.879095
## 3 4.838202
## 4 4.919987
## 5 4.879095
## 6 5.001771
```

```
residuals <- resid(model1)
head(as.matrix(residuals))
```

```
##         [,1]
## 1  0.2209054
## 2  0.0209054
## 3 -0.1382024
## 4 -0.3199868
## 5  0.1209054
## 6  0.3982287
```

```
coefficients(model1)
```

```
##  (Intercept) Petal.Length
##    4.3066034    0.4089223
```

There are also manual ways of extracting the same information.

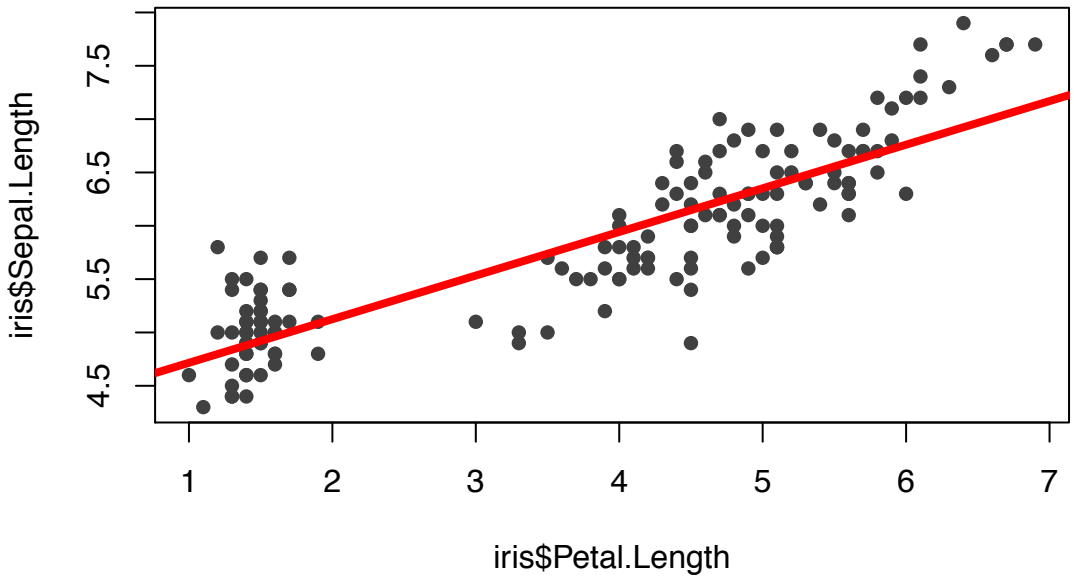
```
model1$coefficients
```

```
##  (Intercept) Petal.Length
##    4.3066034    0.4089223
```

```
coefs <- model1$coefficients
```

Let's see how well our "best fitting line" did in explaining the relationship.

```
plot(iris$Petal.Length,iris$Sepal.Length,
     col="grey25",pch=16)
abline(a=coefs[1],b=coefs[2],lwd=4,col="red")
```



Goodness of Fit

```
sum1 <- summary(model1)
sum1$r.squared # R Squared
```

```
## [1] 0.7599546
```

```
AIC(model1) # Akaike's Information Criterion
```

```
## [1] 160.0404
```

```
BIC(model1) # Bayesian Information Criterion
```

```
## [1] 169.0723
```

Scaling

Scaling methods (or **latent variable models**) are a ***data reduction technique*** designed to collapse a wide range of similarly-composed indicators down into a smaller set underlying components.

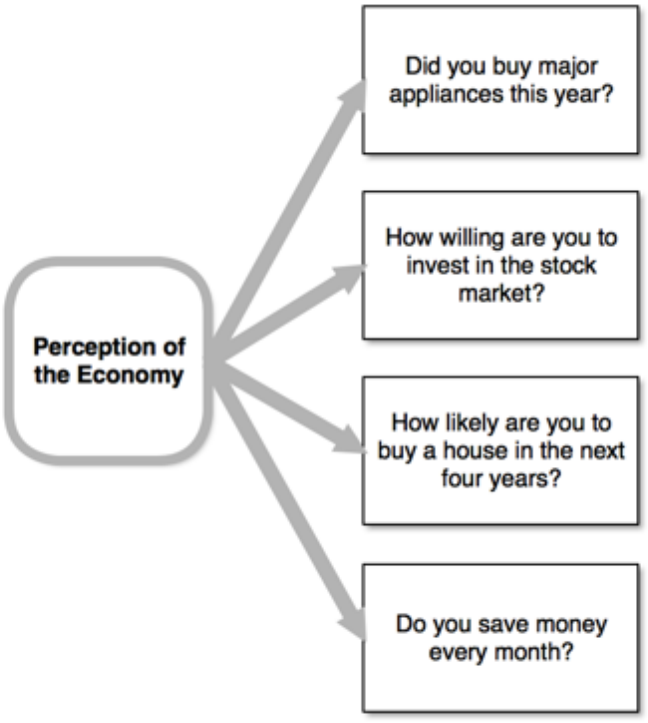
We do this to:

1. reduce a data matrix down to its **unique sources of variation** as a way of incorporating of a broader array of measures without sacrificing degrees of freedom.
2. assuming that the observed variables are **jointly dependent on some unobserved variable** and that unobserved variable corresponses with a **theoretical concept**, we can uncover that latent dimension.

Figure one captures this idea conceptually.

Where "perception of the economy" is latent but one can elicit information about investment, spending, and saving to recover respondent perceptions.

Figure 1: Latent Perception of the Economy



Data Type

Assume we've surveyed 100 respondents and have asked them 8 questions about the economy. Each question is ordinal and provides use information on each individuals spending, saving, and investment patters.

```
str(econ_survey)
```

```
## 'data.frame':    100 obs. of  9 variables:
##  $ repondID: chr  "e1" "e2" "e3" "e4" ...
##  $ econ1   : num  1 2 3 2 2 3 2 0 1 2 ...
##  $ econ2   : num  3 2 3 2 2 3 2 1 2 2 ...
##  $ econ3   : num  1 1 3 2 2 4 1 1 2 3 ...
##  $ econ4   : num  1 1 3 2 2 3 2 1 1 2 ...
##  $ econ5   : num  1 1 4 2 1 3 2 0 1 2 ...
##  $ econ6   : num  2 2 3 1 3 3 2 0 1 1 ...
##  $ econ7   : num  1 1 3 2 1 4 3 0 2 2 ...
##  $ econ8   : num  2 1 4 3 2 4 2 2 2 2 ...
```

Standardizing Variables

Often before running any form of latent variable analysis, we want to **standardize** the variables so that they are on the **same scale**.

We can do this easily in `R` with the `scales()` function. Here we'll drop the character ID differentiating between respondents.

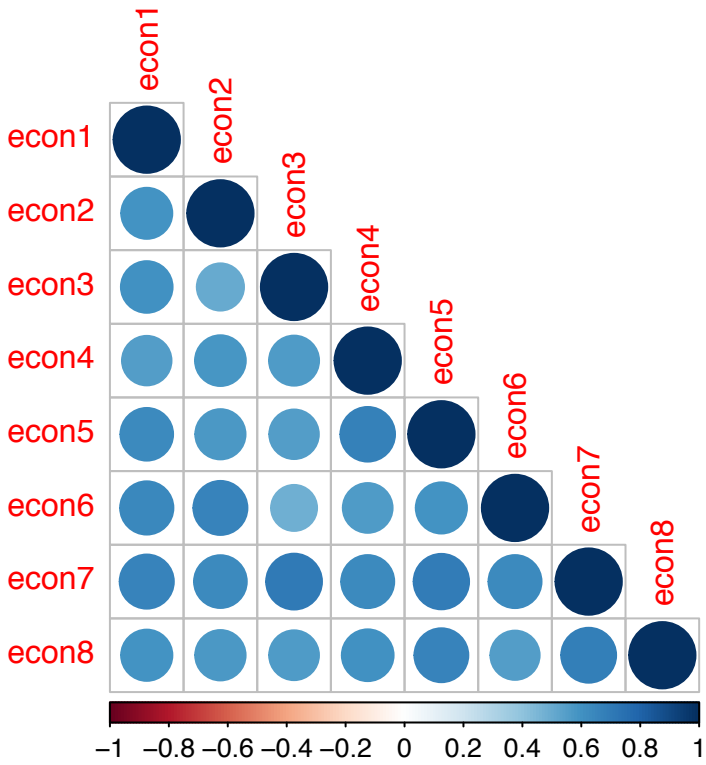
```
econStan <- scale(econ_survey[,-1])
econStan[1:3,1:3] # 1st 4 rows and columns
```

```
##            econ1      econ2      econ3
## [1,] -1.07485280  1.0373705 -1.0837153
## [2,] -0.03130639 -0.1688743 -1.0837153
## [3,]  1.01224002  1.0373705  0.9419208
```

Assessing Dimensionality

For a latent variable model to work, all the input indicators must be highly **correlated**.

```
corrplot(cor(econStan),type = "lower")
```



Second, we need to know how many **unique sources of variation** there are in the data matrix. We can do this by looking at the **eigen values** of the decomposed data matrix.

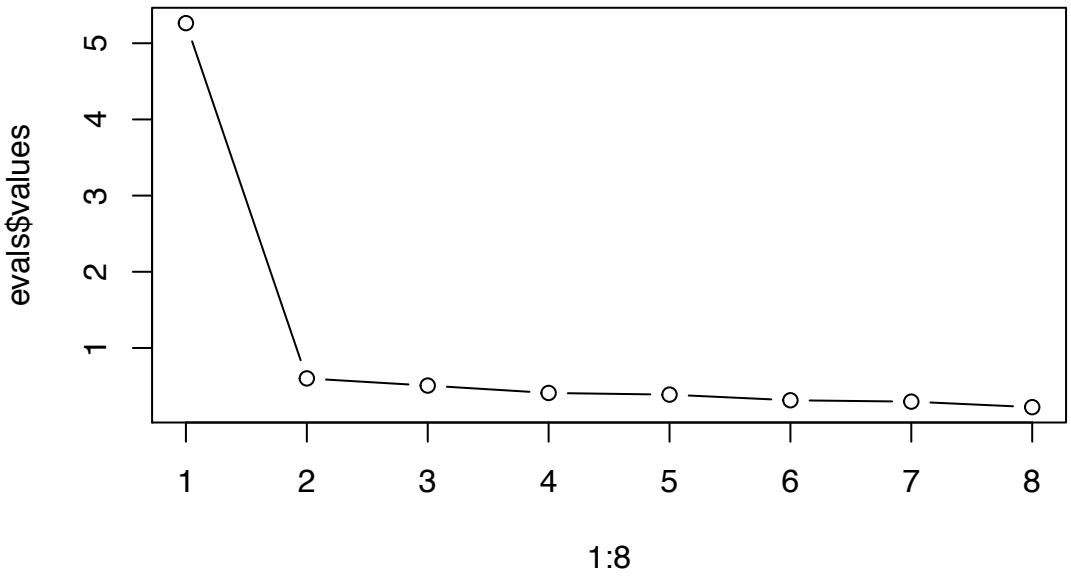
```
evals <- eigen(cor(econStan))
evals$values
```

```
## [1] 5.2639906 0.6004294 0.5062099 0.4095089 0.3876655 0.3135801 0.2954771
## [8] 0.2231385
```

We can plot these values to generate a **Scree Plot**. What we are looking for is an "elbow". The more unique numbers of variation there are, the more latent dimensions we have.

```
plot(1:8,evals$values,type="both")
```

```
## Warning in plot.xy(xy, type, ...): plot type 'both' will be truncated to
## first character
```



Summated Rating Model

The **summated rating model** (SRM) is a commonly employed data reduction technique within social science. At its core, SRM uses error laden measures ("dirty" indicators) to recapture more "fine grained" data of the unique dimension that underpins it.

The model assumes that ***only one underlying dimension exists*** in the data, and requires a ***large number of measures*** to work effectively. As the number of items in the scale increases, the variation in the error will become smaller. Thus, the more items we use, the smaller the variance and the closer the estimated score comes to the true dimension.

SRM is easy to implement. To get the latent score for the underlying dimension, one sums up the K number of variables in the data structure by each unique observation and then divides by K.

Put differently, the SRM simply **takes the row mean** of the data to estimate a score for that observation.

In `R` we can do this easily with the function `rowMeans()`.

```
srm = rowMeans(econStan)
```

Principal Component Analysis

Principal component analysis (PCA) is a data reduction technique that makes no underlying assumptions regarding the latency of some unobserved variable.

PCA is not a model of the data, but rather a transformation of a data matrix that reduces it down to its unique sources of variation to create a smaller set of independent composite variables.

There are number of functions (from various paackage) that will perform PCA, here we'll use the base function `princomp()`.

```
pca = princomp(econStan)
pca$loadings
```

```
##
## Loadings:
##       Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
```

```
## econ1 -0.356        -0.416  0.442 -0.339  0.596         0.161
## econ2 -0.344  0.496        -0.442  0.435  0.319 -0.379
## econ3 -0.335 -0.566 -0.491 -0.347                      -0.443
## econ4 -0.349 -0.116  0.527 -0.465 -0.373  0.229  0.385  0.176
## econ5 -0.363         0.404  0.283 -0.309 -0.171 -0.622 -0.321
## econ6 -0.345  0.580 -0.167        -0.199 -0.532  0.382 -0.212
## econ7 -0.382 -0.192 -0.144         0.135 -0.408 -0.191  0.759
## econ8 -0.353 -0.172  0.307  0.435  0.635         0.367 -0.139
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125
## Cumulative Var  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
```

To return the estimated "scores" of the latent dimension, we'll draw out the value and save it to an object.

```
pca_scale <- pca$scores[,1] # Only the first
```

Factor Analysis

PCA and **Factor Analysis** (FA) are often employed inter-changeably. However, there is are unique conceptual differences that sets them apart.

PCA, put simply, is a technique to reduce a correlated set of observed variables down to a smaller set of independent composite variables. PCA is not a model of the data, it is a tool to compress it.

FA, on the other hand, **assumes a theoretical model of latent factors that cause the observed variables**. The difference is most pronounced in how the latent component is understood theoretically in relation to the observed variables.

A factor analysis is easy to implement in `R`. Though there is a base function `factanal()` that runs factor analysis, the `fa()` function from the `psych` package is by far the most robust.

```
require(psych)
```

```
## Loading required package: psych
```

```
fac_analysis = fa(econStan)
fac_scale <- as.numeric(fac_analysis$scores)
```

Comparing Scales

As we can see, all scales produce scores that are highly correlated with one another.

```
all_scales <- data.frame(srm,pca_scale,fac_scale)
cor(all_scales)
```

```
##                  srm  pca_scale  fac_scale
## srm        1.0000000 -0.9999598  0.9980299
## pca_scale -0.9999598  1.0000000 -0.9985196
## fac_scale  0.9980299 -0.9985196  1.0000000
```

If we look at the distributions of each of the scales, we can see that they are highly similar.

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following objects are masked from 'package:psych':
##
##     %+%, alpha
```