The most powerful functions in Stata are `sort`, `reshape`, `collapse`, and `merge`. These functions are the bread and butter of Stata; and they justify the continued use of Stata in economic research. It seems like R is becoming more prevalent, however, as the types of analysis required for empirical research become more varied, and must rely more heavily on a wider developer base. It makes sense, then, to spend a little time learning how to replicate the `sort`, `reshape`, `collapse`, and `merge` capabilities in R.

We can first create a sample data frame for manipulation, and then print the output.

```
(X <- data.frame(id=c(1,1,2,2), t=c(1,2,1,2), x1=c(5,3,6,2), x2=c(6,5,1,4)))
```

```
  id t x1 x2
1  1 1  5  6
2  1 2  3  5
3  2 1  6  1
4  2 2  2  4
```

Sort

The difference between Stata and R is that, with Stata, the data are very rigid, which often simplifies operations for economic data. The data in R, however, can take many forms, including lists, vectors, and data frames. And there can be multiple data sets in memory simultaneously. Simple operations, then, must be more explicitly defined. Sorting data is not as simple as "sort data", but it's not too much more complicated. Instead, we arrange the data according to a list of indices.

First, as a refresher of indexing a data frame, suppose that we want to extract just the second row of X:

```
X[2, ]
```

```
  id t x1 x2
2  1 2  3  5
```

And the second through fourth rows:

```
X[2:4, ]
```

```
  id t x1 x2
2  1 2  3  5
3  2 1  6  1
4  2 2  2  4
```

And then, suppose, that we want to collect the second and fourth rows, except we ant to reverse the order:

```
X[c(4,2), ]
```

```
  id t x1 x2
4  2 2  2  4
2  1 2  3  5
```

Sorting is just a simple extension. If we collect the indices of an ordered variable, then we can sort the entire data frame. The `order` function facilitates this index ordering.

```
order(X$t)
```

```
[1] 1 3 2 4
```

The sorted data can be achieved by appropriately applying the ordered indices.

```
X[order(X$t), ]
```

```
  id t x1 x2
1  1 1  5  6
3  2 1  6  1
2  1 2  3  5
4  2 2  2  4
```

Reshape

The Stata reshape command has two basic options: wide and long. Suppose that a variable x is indexed along two dimensions, i and j. In wide format, each observation is uniquely identified by the i index, and the j indexed attributes are separate variables or columns. In long format, each observation is uniquely identified by a different i, j combination. All else equal, there will be more observations or rows in long format, and more variables or columns in wide format. Consider for example the data frame X, and let `id` be indexed by i and `t` be indexed by j. The data frame is therefore already in long format, as each observation is identified by an ID *and* time period: individual 1 in period 1 is represented as a different observation than individual 1 in period 2. Suppose that we want the unit of observation to be only the unique IDs, converting the data frame to wide format. There are many, many ways to do this in `R`, but we will use the `reshape` command:

```
(wide <- reshape(X, idvar="id", timevar="t", direction="wide"))
```

```
  id x1.1 x2.1 x1.2 x2.2
1  1    5    6    3    5
3  2    6    1    2    4
```

Note that the arguments in the `reshape` command indicate that the natural use case is for panel data, where i indexes the unit ID variable and j indexes the time variable. Reshaping data is actually pretty fun — it's like playing the Creator (see Figure 1). Even for a simple data frame like `X`, there are $2^3 = 8$ different formats, since we can view the index on the `x` characteristic as yet another index. With respect to that index, then, the data frame `X` is actually in wide format. Consider the data frame where each observation represents a different combindation of the three indices, where the third is the 1 or 2 that follows the `x` variable. Instead of reshape, we can "melt" the data to long format:

```
(llong <- melt(X, id=c("id","t")))
```

```
  id t variable value
1  1 1       x1     5
2  1 2       x1     3
3  2 1       x1     6
4  2 2       x1     2
5  1 1       x2     6
6  1 2       x2     5
7  2 1       x2     1
8  2 2       x2     4
```

So far, we've seen the same data frame `X` with dimension 4×4, 2×5, and 8×4 — all representing the exact same information. Each representation may be useful in a different context.

Collapse

Suppose that we want to transform the data frame to reflect the average value of x_1 and x_2 across time periods $t \in \{1, 2\}$. In Stata, we'd use the `collapse` command; the comparable command, both in function and in name, within `R` is `aggregate`.

Figure 1: Playing the creator with `reshape`

```
(aggdata <-aggregate(X, by=list(X$id), FUN=mean))
```

```
  Group.1 id   t x1  x2
1       1  1 1.5  4 5.5
2       2  2 1.5  4 2.5
```

The `Group.1` and `t` variables are redundant, here, for this very simple data set. The `aggregate` function supports much more complicated queries, however, and adding the group variables becomes useful. A common query in Stata is to collapse the data by multiple variables. The extension for `aggregate` is clear, since the `by` parameter is a list.

Merge

Consider another data frame `Y` with another characteristic for each unit and time period.

```
(Y <- data.frame(id=c(1,1,2,2), t=c(1,2,1,2), x3=c(4,3,4,3)))
```

```
  id t x3
1  1 1  4
2  1 2  3
3  2 1  4
4  2 2  3
```

To merge `X` and `Y` into a single data frame, we use the `merge` function. This is comparable to Stata's merge command, except that in `R`, you can have two data frames stored in memory; you do not need to save one to disk. Additionally, in `R`, the data sets do not need to be sorted before the merge. Together, these enhancements save many lines of code.

```
merge(X, Y, by=c("id", "t"))
```

```
  id t x1 x2 x3
1  1 1  5  6  4
2  1 2  3  5  3
3  2 1  6  1  4
4  2 2  2  4  3
```

A convenient aspect of `merge` in `R` is that, unlike Stata, the basis variables need not be named the same. Instead of a generic `by` parameter, we can specify the merging variables with `by.x` and `by.y`, which refer to the first and second data frames, respectively.

```
names(Y) <- c("a", "b", "x3")
merge(X, Y, by.x=c("id", "t"), by.y=c("a", "b"))
```

```
  id t x1 x2 x3
1  1 1  5  6  4
2  1 2  3  5  3
3  2 1  6  1  4
4  2 2  2  4  3
```

Merges in economic data, especially panel data, are often used to attribute static characteristics to the time series. The target data may be organized by time and unit, whereas the new data frame may be at just the unit level. The two data frames need not ber perfectly aligned, as in the previous examples. Consider a new, unit-level data frame `Z` that contains static characteristics (that do not depend on time).

```
(Z <- data.frame(id=c(1,2), x4=c("yes", "no")))
```

```
  id  x4
1  1 yes
2  2  no
```

We can merge this into the panel data frame X using roughly the same syntax.

```
merge(X, Z, by=c("id"))
```

```
  id t x1 x2  x4
1  1 1  5  6 yes
2  1 2  3  5 yes
3  2 1  6  1  no
4  2 2  2  4  no
```

Distribution simulations

The presence of an unobserved, individual effect c_i in a panel data model will create a correlation over time in the outcome variable, even if the idiosyncratic error is completely random. Still, with the standard assumptions in place, ordinary least squares will yield a consistent estimator. Consistent but inefficient. The correlated structure of the composite error, which imclused the individual effect, must be made in order to correctly identify the variance-covariance structure.

The following illustrates the variance of the linear estimator, relative to the estimate that fully accounts for the fixed effects. The data generating process is defined by $\mathbf{y} = 10\mathbf{c} + \mathbf{x} + \epsilon$, where $\epsilon \sim N(0, 1)$ and $c_i \in \{1, 2, 3, 4, 5\}$. By construction, $\mathbf{x}$ and $\mathbf{c}$ are orthogonal, so that the pooled linear estimator will be consistent. Figure 1 and 2 together show why the simple, pooled estimator will be inefficient. The following code generates a panel data set with $N = 5$ and $T = 20$, and then plots the full data set with an overall linear fit in Figure 1. It is apparent that the linear fit is subject to the spread of the covariate $\mathbf{x}$ from within a particular group. Generally, increased variation in the cofactors will yield a more precise estimator; but the variable intercept (read: fixed effect) that is relegated to the error term could potentially reverse this relationship.

```
library(ggplot2)
c <- rep(c(1,2,3,4,5), 20); x <- rnorm(100); eps <- rnorm(100)
y <- 10 * c + x + eps
p <- ggplot(data.frame(c, x, y), aes(x = x, y = y, color=c))
(p <- p + geom_point() + geom_smooth(method=lm))
```

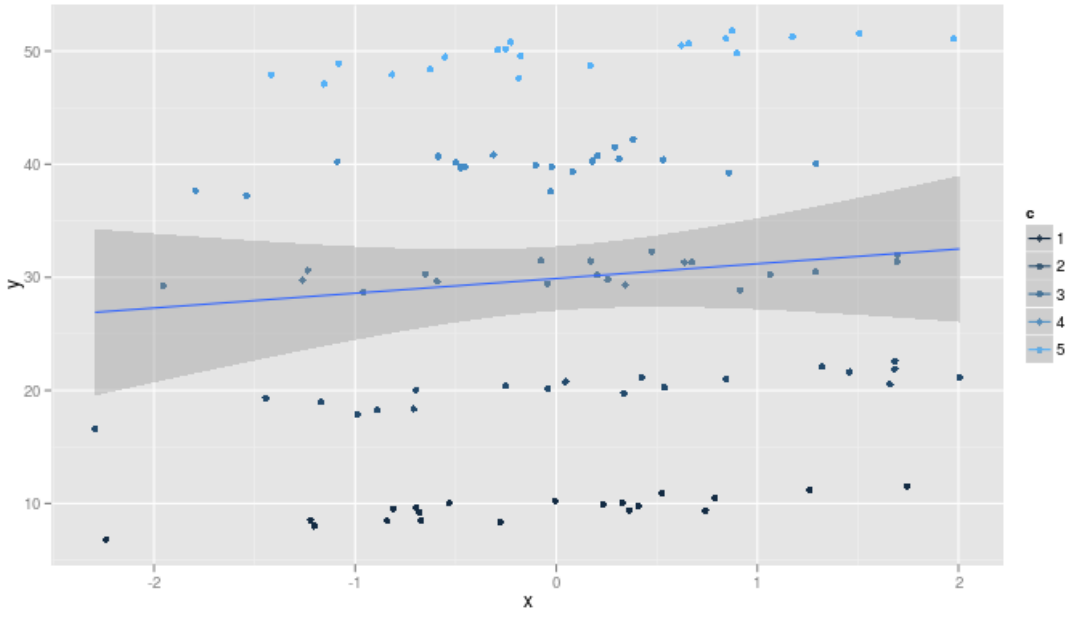

Figure 1: Fixed effect scatterplot with pooled, linear fit

We can directly calculate the variance of the coefficient on $\mathbf{x}$, but this is too much effort. Instead, we can bootstrap the distributions, since we really only care about the relative efficiency of the estimators. Figure

2 clearly illustrates that the pooled estimate is inefficient relative to directly modelling the true fixed effect. We use $B = 100$ iterations, regenerating the data each time. For this simple example, the value of the individual identifier is an argument in the data generating process. Even if this were not the case, however, the distributions in Figure 2 suggest that there must be a better way to weight the estimator to achieve a greater efficiency. And in fact, there is — clustered variance, or the `robust` option in Stata. The procedure to directly calculate the clustered variance is detailed in the lecture notes, and will not be presented again here.

```
B <- 100
fe.res <- rep(NA, B); ols.res <- rep(NA, B)

for (i in 1:B) {
  c <- rep(c(1,2,3,4,5), 20); x <- rnorm(100); eps <- rnorm(100)
  y <- 10 * c + x + eps

  ols <- lm(y ~ x)
  ols.res[i] <-  ols$coefficients[["x"]]

  fe <- lm(y ~ x + factor(c))
  fe.res[i] <- fe$coefficients[["x"]]
}
```

The geometric argument for the spread of the OLS estimates is straightforward: variation in the covariates will have differential impacts on the slope of the linear fit, depending on which strata is represented. If by chance, for example, $\mathbf{x}$ observations were disproportionately selected from the upper tail of the normal distribution when $\mathbf{c} == 1$, then the pooled linear fit will slope downward. If $\mathbf{c} == 5$, however, the linear fit will slope upwards. This alone should give pause in assessing the efficiency of the pooled estimator. Conditional on the covariates, all observations should be given equal weight. The clustered variances help to mitigate this effect by appropriately reweighting the observations.

```
labels <- c(rep("FE", B), rep("pooled", B))
sim <- data.frame(coefficient=c(fe.res, ols.res), method=labels)
ggplot(sim, aes(x = coefficient, fill=method)) + geom_density(alpha=0.2)
```

Suppose that the model was not linear, but rather characterized by a limited dependent variable. What will happen to the consistency and efficiency of the pooled estimate, without taking into account the correlated structure of the error? Figure 3 indicates that the additional variation in the composite error is not averaged away. Instead, the pooled OLS estimator is centered around the wrong estimate, suggesting that the impact of $\mathbf{x}$ on $\mathbf{y}$ is smaller than it is in truth. The value of directly modelling the error is complicated by the nonlinear Probit model. The following code first generates a binary dependent variable from the random covariates, and then esimates the generalized linear model using the Probit function as the binomial link.

```
fe.probit <- rep(NA, B); pooled.probit <- rep(NA, B)

for (i in 1:B) {
  c <- rep(c(1,2,3,4,5), 20); x <- rnorm(100); eps <- rnorm(100)
  y <- ifelse(c + x + eps > 5, 1, 0)

  pool <- glm(y ~ x, family = binomial(link = "probit"))
  pooled.probit[i] <- pool$coefficients[["x"]]

  fe <- glm(y ~ x + factor(c), family = binomial(link = "probit"))
  fe.probit[i] <- fe$coefficients[["x"]]
}
```

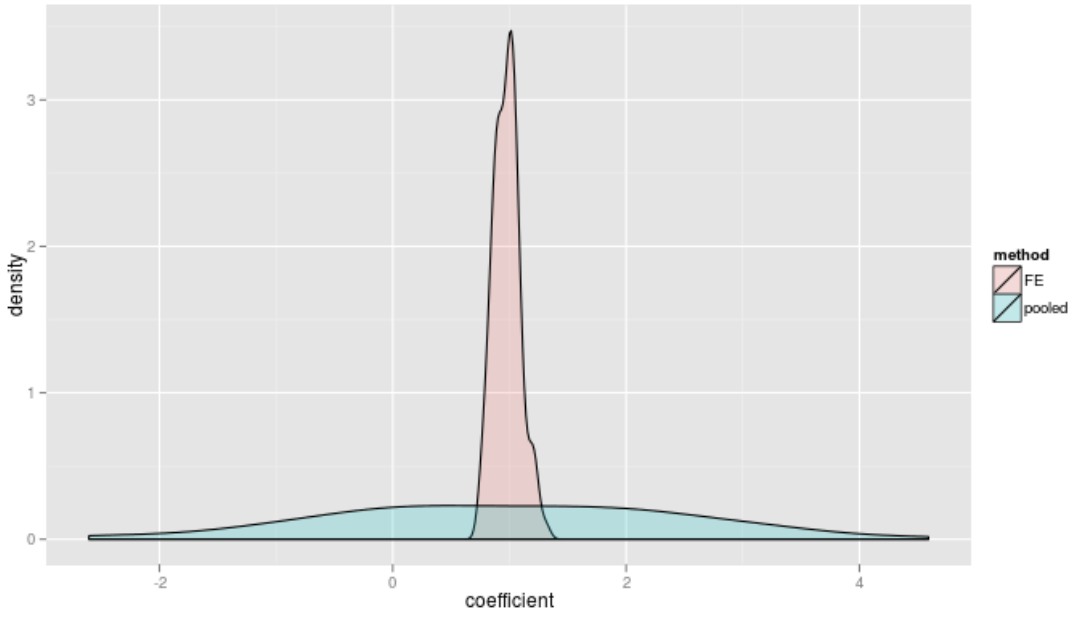

Figure 2: Pooled OLS versus fixed effects, simulated distributions

```
labels <- c(rep("FE", B), rep("pooled", B))
sim.probit <- data.frame(coefficient=c(fe.probit, pooled.probit), method=labels)
ggplot(sim.probit, aes(x = coefficient, fill=method)) + geom_density(alpha=0.2)
```

Two-way, fixed effects panel model

Consider the fixed effects, two-way component panel data model:

$$y_{it} = \alpha + x_{it}\beta + \mu_i + \lambda_t + \epsilon_{it}$$

The fixed effects estimator of β can be obtained by regressing $\mathbf{y}$ on $\mathbf{X}$, $\mathbf{Z}_1$, and $\mathbf{Z}_2$, where $\mathbf{Z}_1 = \mathbb{I}_N \otimes \kappa$ is a matrix of unit indicators and $\mathbf{Z}_2 = \iota \otimes \mathbb{I}_T$ is a matrix of time period indicators, with κ a vector of ones of dimension T and ι a vector of ones of dimension N. Note that $\dim(\mathbf{Z}_1) = TN \times N$ and $\dim(\mathbf{Z}_2) = TN \times T$, assuming a balanced panel.

The computation for this regression is daunting, however, since it requires the inversion of a $(k + N + T - 1) \times (k + N + T - 1)$ matrix. The Frisch-Waugh (FW) theorem suggests that instead of a direct regression, we can demean the variables across time and units. The FW theorem proves that a one-way within transformation will yield the same estimator as a fixed effects regression; and the theorem can be extended for both individual and time effects. The error component structure has the form $u_{it} = \mu_i + \lambda_t + \epsilon_{it}$, which is can be translated into matrix form: $u = (\mathbb{I}_N \otimes \kappa)\alpha + (\iota \otimes \mathbb{I}_T)\lambda + \epsilon$, with $\alpha = [\alpha_1, \alpha_2, \ldots \alpha_N]'$ and $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_T]'$. The error structure suggests a candidate *purging* matrix, which removes the individual- and time-specific effects, along with the overall mean. Call this matrix $\mathbf{Q} = \mathbb{I}_N \otimes \mathbb{I}_T - \mathbb{I}_N \otimes \kappa\kappa'/T - \mathbb{I}_T \otimes \iota\iota'/N + \iota\iota'/N \otimes \kappa\kappa'/T$, which will remove, in turn, the fixed time, unit, and total (through space and time) effect. If both $\mathbf{y}$ and $\mathbf{X}$ are sorted by unit and time, then a regression of $\mathbf{Qy}$ on $\mathbf{QX}$ should yield an unbiased estimate of β with a properly identified error structure.

Define $\mathbb{P}_1 = \mathbb{I}_{NT} - \mathbf{Z}_1(\mathbf{Z}_1'\mathbf{Z}_1)^{-1}\mathbf{Z}_1'$ and $\mathbb{P}_2 = \mathbb{I}_{NT} - \mathbf{Z}_2(\mathbf{Z}_2'\mathbf{Z}_2)^{-1}\mathbf{Z}_2'$ to be the projection matrices for individual and time fixed effects, respectively. It is sufficient to prove that $\mathbb{P}_1\mathbb{P}_2 = \mathbf{Q}$ to create a composition projection

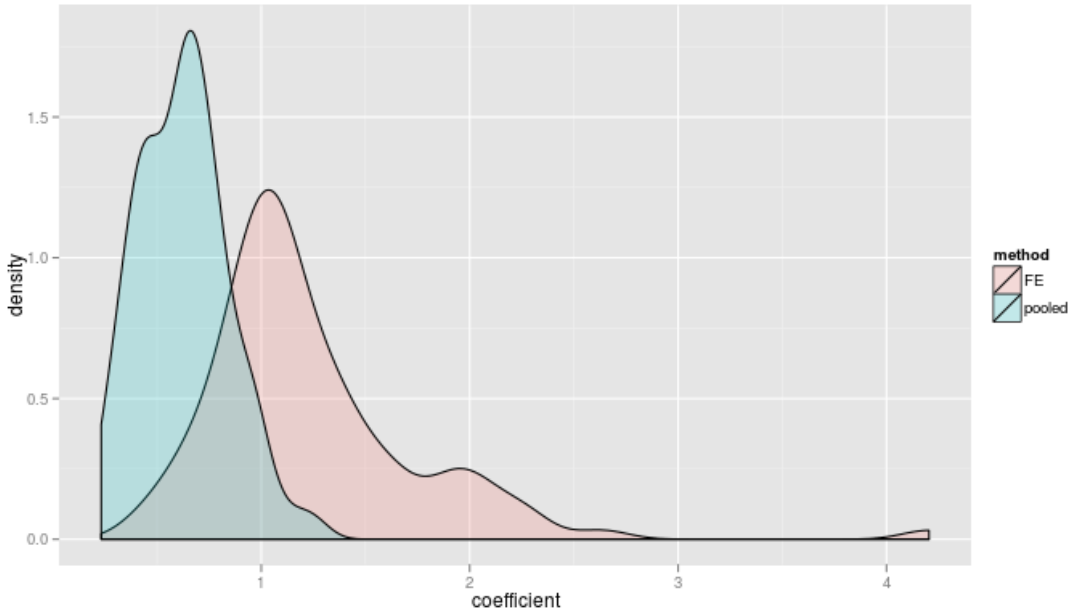

Figure 3: Limited dependent variable: inconsistency of pooled estimator

matrix: first a within transformation ignoring the time effects followed by a within transformation ignoring the individual effects. First, note that:

$$\begin{aligned}
\mathbb{P}_2 = \mathbb{I}_{NT} - \mathbf{Z}_2(\mathbf{Z}_2'\mathbf{Z}_2)^{-1}\mathbf{Z}_2' &= \mathbb{I}_{NT} - \mathbf{Z}_2((\iota \otimes \mathbb{I}_T)'(\iota \otimes \mathbb{I}_T))^{-1}\mathbf{Z}_2' \\
&= \mathbb{I}_{NT} - \mathbf{Z}_2((\iota' \otimes \mathbb{I}_T')(\iota \otimes \mathbb{I}_T))^{-1}\mathbf{Z}_2' \\
&= \mathbb{I}_{NT} - \mathbf{Z}_2(\iota'\iota \otimes \mathbb{I}_T)^{-1}\mathbf{Z}_2' \\
&= \mathbb{I}_{NT} - \mathbf{Z}_2(N \cdot \mathbb{I}_T)^{-1}\mathbf{Z}_2' \\
&= \mathbb{I}_{NT} - N^{-1}\mathbf{Z}_2\mathbf{Z}_2' \\
&= \mathbb{I}_{NT} - N^{-1}(\iota \otimes \mathbb{I}_T)(\iota \otimes \mathbb{I}_T)' \\
&= \mathbb{I}_{NT} - N^{-1}(\iota\iota' \otimes \mathbb{I}_T) \\
&= \mathbb{I}_{NT} - (\iota\iota'/N \otimes \mathbb{I}_T)
\end{aligned}$$

A similar, nearly symmetric argument can be made to show that $\mathbb{P}_1 = \mathbb{I}_{NT} - (\mathbb{I}_N \otimes \kappa\kappa'/T)$. It follows that the sequential projection using $\mathbb{P}_1$ and $\mathbb{P}_2$ is equivalent to the two-way demeaning matrix $\mathbf{Q}$:

$$\begin{aligned}
\mathbb{P}_2\mathbb{P}_1 &= (\mathbb{I}_{NT} - (\iota\iota'/N \otimes \mathbb{I}_T))\,(\mathbb{I}_{NT} - (\mathbb{I}_N \otimes \kappa\kappa'/T)) \\
&= \mathbb{I}_{NT}^2 - (\iota\iota'/N \otimes \mathbb{I}_T) - (\mathbb{I}_N \otimes \kappa\kappa'/T) + (\iota\iota'/N \otimes \mathbb{I}_T)(\mathbb{I}_N \otimes \kappa\kappa'/T) \\
&= \mathbb{I}_{NT} - (\iota\iota'/N \otimes \mathbb{I}_T) - (\mathbb{I}_N \otimes \kappa\kappa'/T) + (\iota\iota'/N \otimes \kappa\kappa'/T) \\
&= \mathbb{I}_N \otimes \mathbb{I}_T - (\iota\iota'/N \otimes \mathbb{I}_T) - (\mathbb{I}_N \otimes \kappa\kappa'/T) + (\iota\iota'/N \otimes \kappa\kappa'/T) = \mathbf{Q}
\end{aligned}$$

The sequential projection onto the fixed effect matrices is numerically equivalent to a two-way within transformation. The bilinear and associative properties of the Kronecker product in the steps above ensure that $\mathbb{P}_1\mathbb{P}_2 = \mathbb{P}_2\mathbb{P}_1 = \mathbf{Q}$, so that the ordering of the within transformations make no difference. Note that $\mathbf{Q}$ is itself a projection matrix, such that $\mathbf{Q}$ is idempotent and symmetric. The estimator can be simplified:

$$\beta = ((\mathbf{QX})'\mathbf{QX})^{-1}(\mathbf{QX})'\mathbf{Qy} = (\mathbf{X}'\mathbf{Q}'\mathbf{QX})^{-1}\mathbf{X}'\mathbf{Q}'\mathbf{Qy} = (\mathbf{X}'\mathbf{QX})^{-1}\mathbf{X}'\mathbf{Qy}$$

The results depend crucially on the panel being balanced. Otherwise, the within transformations become much, much more complicated. The non-uniform structure requires individual and special treatment for

each unit in the data set. The block diagonal matrices are of various sizes, and the dummy variable matrices must be tailored to suit the various time intervals. This is not to say that it cannot be done, however, but the demeaning process becomes complicated, circumstantial.

Extra: Some additional insight can be obtained by denoting $\mathbf{A}_k$ as the demeaning matrix of dimension k. Premultiplying $\mathbf{X}$ by $\mathbf{A}_n$ will return a matrix with deviations from column means; this is the standard case that was presented in class. If we apply the appropriately dimensioned $\mathbf{A}$ matrix to the transpose of $\mathbf{X}$, however, we can achieve row means. Note that $\mathbf{A}_k$ is symmetric and idempotent, so that $\mathbf{A}_k\mathbf{A}_k = \mathbf{A}_k$ and $\mathbf{A}'_k = \mathbf{A}_k$. If we wanted to transform the matrix toward deviations from row means, we would post-multiply by $\mathbf{A}_k$: $(\mathbf{A}_k\mathbf{X}')' = \mathbf{X}\mathbf{A}'_k = \mathbf{X}\mathbf{A}_k$. This suggests the form of $\mathbb{P}_1$ versus $\mathbb{P}_2$ above.

Preliminaries: data generating process

The first part of these section notes is a variation on Max Auffhammer's final exam for ARE212 in 2011. If you don't like this treatment of IV estimation, then blame him. Actually, I like this pattern: if you don't like something I do, then blame Max. He is probably at fault. I am faultless. Consider the following data generating process:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \eta_i \quad \text{with} \quad \eta_i \sim N(0,1) \tag{1}$$

and $\beta = [1\ 2\ \text{-}4\ 1]'$. Assume that the covariance matrix of the covariates, an additional instrument, and the idiosyncratic error (x_{1i}, x_{2i}, x_{3i}, z_i, and η_i) is defined to be

$$\Sigma = \begin{bmatrix} 1 & 0 & \rho_{13} & 0 & 0 \\ 0 & 1 & \rho_{23} & \rho_{2z} & 0 \\ \rho_{13} & \rho_{23} & 1 & 0 & 0 \\ 0 & \rho_{2z} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where each variable is assumed to have zero mean for simplicity. Note that Σ is consistent with η_i being independently and identically distributed with constant variance. We will explore the properties of IV (e.g., weak instruments and the exclusion restriction) via Monte Carlo simulation.

The first step, then, is to figure out how to generate random data with the appropriate covariance. For this, a useful function is `rmvn.chol`, which returns a random $n \times k$ multivariate normal matrix $\mathbf{X}$, based on the supplied covariance matrix `vcov.mat`. In general, $\mathbf{X} = \mathbf{Q} + \mu'$, where $\mathbf{Q}'\mathbf{Q} = \Sigma$ and μ is a vector of means for each of the k columns of $\mathbf{X}$. We have assumed away $\mu = \mathbf{0}$ and so the problem becomes a simple application of matrix decomposition.

```
rmvn.chol <- function(n, vcov.mat) {
  k <- ncol(vcov.mat)
  Q <- chol(vcov.mat)
  Z <- matrix(rnorm(k*n), nrow=n, ncol=k)
  return(Z %*% Q)
}
```

It will be handy to have a simple function to generate Σ with three arguments representing the three non-zero correlations across the covariates and the instrument.

```
vcov.fn <- function(rho.13, rho.23, rho.2z) {
  mat <- diag(5)
  mat[3,1] <- rho.13; mat[1,3] <- rho.13
  mat[2,3] <- rho.23; mat[3,2] <- rho.23
  mat[2,4] <- rho.2z; mat[4,2] <- rho.2z
  return(mat)
}
```

The result is a way to generate the random data according to the specified data generating process. The following generates the covariance matrix and a random multivariate normal matrix with 500 observations, printing Σ for reference:

```
(vcov <- vcov.fn(rho.13 = 0, rho.23 = 0.5, rho.2z = 0.5))
X <- rmvn.chol(500, vcov)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1  0.0  0.0  0.0    0
[2,]    0  1.0  0.5  0.5    0
[3,]    0  0.5  1.0  0.0    0
[4,]    0  0.5  0.0  1.0    0
[5,]    0  0.0  0.0  0.0    1
```

A quick check to ensure that the variance of each variable is 1, as specified by Σ. There will be some variation around the true variance, but even with 500 observations, it's clear that we are approaching the true variance:

```
apply(X, 2, function(i){var(i)})
```

```
[1] 1.0568253 1.0075860 1.0063271 1.0076609 0.9481866
```

Calculate bias in β and it's standard error

We will now write a couple of functions to help examine the bias of the parameter vector and it's standard error, using an array of estimation techniques, including OLS, 2SLS, and estimation by proxy variable. First, let's put together a very simple function to calculate the parameter vector and it's standard error for both direct regression and two-stage least squares, when a first-stage covariate matrix is provided.

```
ols.results <- function(y, X, first=FALSE) {
  Xt <- t(X)
  xtxi <- solve(Xt %*% X)
  beta <- xtxi %*% Xt %*% y
  if (first == FALSE) {
    e <- y - X %*% beta
  }
  else {
    e <- y - first %*% beta
  }
  s2 <- (t(e) %*% e)/(nrow(X) - ncol(X))
  se <- sqrt(diag(xtxi)*s2)
  return(cbind(beta, se))
}
```

Now comes the serious stuff, specifically, the code that is written specifically to examine IV estimation in this example. Suppose that we do not observe x_{3i}. The composite error is then $x_{3i} + \eta_i$, and we estimate the parameter vector by regressing y_i on x_{1i} and x_{2i}. If $\rho_{13} = \rho_{23} = 0$, then there is no problem: OLS will yield consistent estimates, since the regression utilizes only exogenous variation. If, however, the covariates are correlated with the composite error, the OLS estimates will be biased.

The function `est.bias` returns the simulated bias in the parameter estimates and standard errors from a Monte Carlo simulation with $B = 10,000$ repetitions. The arguments are `vcov` which is the variance-covariance matrix generated by `vcov.fn`; `n` which specifies the number of observations for each iteration, defaulted at 500; `B` is the number of iterations in the MC simulation, defaulted at 10,000; `two.stage` is a boolean argument indicating whether the simulation should employ two-stage least squares with z_i as the instrument for x_{3i}, defaulted to `FALSE`. The default behavior, then, is to run a simulation where x_{3i} is left out of the OLS regression, relegated to the error term.

```
est.bias <- function(vcov, n=500, B=10000, two.stage=FALSE) {
  true.beta <- c(1, 2, -4, 1)
  res.beta <- mat.or.vec(3,B); res.se <- mat.or.vec(3,B)
```

```
  for (i in 1:B) {

    data <- rmvn.chol(n, vcov)

    X <- cbind(1, data[,1:3]); eta <- data[,5]
    y <- X %*% true.beta + eta
    full.ols <- ols.results(y, X)

    if (two.stage==TRUE) {
      endog <- data[,2]
      first  <- cbind(1, data[,c(1,4)])
      predict <- first %*% solve(t(first) %*% first) %*% t(first) %*% endog
      exog <- cbind(1, data[,1], predict)
      limited.ols <- ols.results(y, exog, first=first)
    }
    else {
      exog <- cbind(1, data[,1:2])
      limited.ols <- ols.results(y, exog)
    }

    res.beta[,i] <- limited.ols[,1] - true.beta[1:3]
    res.se[,i]   <- limited.ols[,2] - full.ols[1:3,2]
  }
  results <- cbind(rowMeans(res.beta), rowMeans(res.se))
  colnames(results) <- c("beta bias", "se chg")
  print(results)
}
```

We can check `est.bias` by first setting $\rho_{13} = \rho_{23} = 0$ and ensuring that the bias is very low with $n = 500$ and $B = 10,000$. The following MC simulation sets $\Sigma = \mathbb{I}_5$ and runs the following regression $10,000$ times:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \epsilon_i \quad \text{with} \quad \epsilon_i = x_{3i} + \eta_i \tag{2}$$

```
vcov <- vcov.fn(rho.13 = 0, rho.23 = 0, rho.2z = 0)
est.bias(vcov)
```

```
         beta bias     se chg
[1,]  3.025788e-05 0.01850461
[2,]  7.504312e-05 0.01853065
[3,] -2.155886e-04 0.01853233
```

The parameter estimates seem fairly consistent. The simulated bias is negligible. The standard errors in the simulation are somewhat larger, however, simply because there is more variation in the composite error term. Now suppose that $\rho_{23} = \rho_{3z} = 0.5$, so that $\mathbb{E}[\epsilon_i | x_{1i}, x_{2i}] \neq 0$. The coefficient β_2 is biased upwards by 0.5 in the simulation below (the magnitude of the correlation and parameter bias are not generally the same).

```
vcov <- vcov.fn(rho.13 = 0, rho.23 = 0.5, rho.2z = 0.5)
est.bias(vcov)
```

```
         beta bias      se chg
[1,] -9.313589e-05 0.014414161
[2,] -3.125962e-04 0.014437845
[3,]  5.000045e-01 0.007484851
```

No biggie. We have a suitable instrument, by design, that we can use to cordon off exogenous variation for estimation. Note that $Cov(z_i, \epsilon_i) = 0$ since z_i is uncorrelated with the full, composite error, but that $Cov(z_i, x_{2i}) = 0.5 \neq 0$, satisfying both properties of a suitable instrument. We can now get a consistent estimate for β:

```
est.bias(vcov, two.stage=TRUE)
```

```
         beta bias    se chg
[1,]  0.0006683202 0.1239639
[2,] -0.0005169007 0.1240761
[3,] -0.0042907956 0.2877558
```

Note, however, that the standard errors are much higher. We can prove that, in general, the variance of the IV estimator is larger than the variance of an OLS estimator. Let $\mathbf{Z}$ be a matrix of instruments and $\mathbf{P} = \mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'$. Then the IV estimator is $\hat{\beta}_{iv} = (\mathbf{X}'\mathbf{P}\mathbf{X})^{-1}\mathbf{X}'\mathbf{P}\mathbf{y}$. The projection matrix $\mathbf{P}$ is symmetric and idempotent, as is the residual maker $(\mathbb{I}_N - \mathbf{P})$. Note that for any $\mathbf{X}$ and any symmetric, idempotent matrix, the following fact is true:

$$\mathbf{X}'\mathbf{P}\mathbf{X} = \mathbf{X}'\mathbf{P}\mathbf{P}\mathbf{X} = \mathbf{X}'\mathbf{P}'\mathbf{P}\mathbf{X} = (\mathbf{P}\mathbf{X})'(\mathbf{P}\mathbf{X}) \geq 0 \tag{3}$$

Note also that $\mathbb{V}(\hat{\beta}_{ols}) = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}$ and $\mathbb{V}(\hat{\beta}_{iv}) = \sigma^2(\mathbf{X}'\mathbf{P}\mathbf{X})^{-1}$. It suffices to show that $\mathbf{X}'\mathbf{X} \geq \mathbf{X}'\mathbf{P}\mathbf{X}$ to complete the proof. It follows from Equation (3) that:

$$\mathbf{X}'(\mathbb{I}_N - \mathbf{P})\mathbf{X} \geq 0 \Rightarrow \mathbf{X}'\mathbf{X} - \mathbf{X}'\mathbf{P}\mathbf{X} \geq 0 \Rightarrow \mathbf{X}'\mathbf{X} \geq \mathbf{X}'\mathbf{P}\mathbf{X},$$

thus proving that $\mathbb{V}(\hat{\beta}_{iv}) \geq \mathbb{V}(\hat{\beta}_{ols})$. The intuition is a bit more clear. The IV estimator restricts the use of covariation between $\mathbf{y}$ and $\mathbf{X}$ to *only* exogenous variation. The resulting estimator relies on less estimating variation to work with, and is therefore subject to greater variance.

Instrumental variables are by no means a cure-all, however — far from it. First of all, an argument must be made for the exclusion restriction, that indeed the instrument is uncorrelated with the composite error term. There is no way to check this empirically. Second, even if the exclusion restriction holds, the instrument must be highly correlated with the covariates. Otherwise, we run into the problem of weak instruments, which is a *big* problem, as illustrated below. The set up is exactly the same, except that $\rho_{2z} = 0.01$. Technically, both properties of a suitable instrument are satisfied, but the instrumental projection of x_{2i} leaves very little variation to work with. The result is a crazy amount of variance, yielding an almost certainly biased estimate of β. The first simulation shows the standard bias induced by a violation of strict exogeneity:

```
vcov <- vcov.fn(rho.13 = 0, rho.23 = 0.5, rho.2z = 0.01)
est.bias(vcov)
```

```
         beta bias      se chg
[1,] -5.845659e-06 0.014421682
[2,]  9.731249e-05 0.014443940
[3,]  5.000803e-01 0.007497558
```

The second simulation displays the results of trying to correct the endogeneity with a weak instrument. Not good. Way worse.

```
est.bias(vcov, two.stage=TRUE)
```

```
       beta bias     se chg
[1,]  0.18013841   982.6684
[2,] -0.02440537   745.3372
[3,]  2.06182132 13433.0801
```

The purpose of this section is to provide a numerical example for the Stable Unit Value Treatment Assumption (SUTVA) and to present an example of loess regression using `ggplot2`.

SUTVA

The very simple but fundamental assumption of stable unit treatment values can be illustrated with a numerical example. The notation is fairly simple, but the numerical example made the concept much clearer to me. Let $\mathbf{D}$ be an $N \times 1$ column of treatment values, so that D_i is an indicator of treatment for observation $i = 1, 2, \ldots, N$. Let $\mathbf{D}'$ be another vector which defines a different treatment regime. Formally, SUTVA states that if $D_i = D_i'$, then $Y_i(\mathbf{D}) = Y_i(\mathbf{D}')$. Less formally, SUTVA states that there is no spillover effect from the treatment. As stated in class, if SUTVA does not hold, then there is not a singular treatment effect, but rather the potential for an effect of a change in D_i on all observations $j = 1, 2, \ldots, N$, not just on the own observation effect. Consider the specific treatment vector $\mathbf{D}$:

```
(D <- c(1,0,1,0,0,0,0,0,0,1))
```

```
 [1] 1 0 1 0 0 0 0 0 0 1
```

Define the treatment effect to be determined by the random vector $\mathbf{x}$. This, if there were no spillover at all, $Y_i(1) = x_i$ and $Y_i(0) = 0$. However, we induce spillover by defining the treatment effect to depend on a rolling function. Specifically, define $Y_i(D_i) = \max\{D_{i+1}, D_i\} \cdot x_i$. The following applies this rule, and then prints the value of the treatment for $i = 7$.

```
library(zoo)
x <- runif(10)
Y <- rollmax(D, 2, fill = 1) * x
print(Y[7])
```

```
[1] 0
```

Now suppose we adjust the treatment regime by setting $D_8 = 1$. Under both regimes, $D_7 = D_7' = 0$, such that if SUTVA were to hold, we should observe $Y_7(\mathbf{D}) = Y_7(\mathbf{D}')$. This is not the case, however, since the value of D_8 informs the treatment effect of the seventh observation.

```
D[8] <- 1
Y <- rollmax(D, 2, fill = 1) * x
print(Y[7])
```

```
[1] 0.7799037
```

When the treatment effect of one observation depends on the treatment of another observation, then SUTVA does not hold. We are unable to estimate the treatment effect relative to the "no intervention" scenario. This is a relatively simple example, but it's conceivable that the effect across observations may be much more complicated.

Poorly defined treatments

Suppose that the value of the treatment is a function of the level of the outcome variable. This may seem a little circular, but consider a situation where the market for the outcome adjusts based on the total activity in that market. What if the individual treatment effect is a function of the level of the aggregate outcome? The treatment is poorly defined, and it may be difficult to settle on a coneptual idea of the treatment effect, let alone an empirical assessment. Suppose for example, that the treatment effect for each individual is

$x_i \cdot (\sum_i Y_i)^{-1}$. Ignore the timing of the treatment for now, and assume that the iterative adjustment process is condensed into a single time period. The higher aggregate treatment effect will induce a shift of the market supply, perhaps, and lower the individual treatment effect — a sort of congestion effect.

The following example illustrates this point. We can calculate the treatment effects for each of the ten individuals, which is a function of the aggregate treatment effect.

```
D <- c(1,0,1,0,1,0,1,1,0,1)
x <- runif(10)
(Y <- D * x * 1/sum(D * x))
```

```
 [1] 0.20911598 0.00000000 0.15488145 0.00000000 0.08860079 0.00000000
 [7] 0.05381528 0.22953690 0.00000000 0.26404959
```

If the market responds to the aggregate level of treatment — which it does by construction — then by changing the treatment regime so that observation $i = 2$ now receives treatment, the effect will be felt by all other individuals. Note that the treatment effects for the other observations are slightly lower with the restricted change in the treatment regime.

```
D[2] <- 1
(Y <- D * x * 1/sum(D * x))
```

```
 [1] 0.17523886 0.16200158 0.12979041 0.00000000 0.07424733 0.00000000
 [7] 0.04509712 0.19235156 0.00000000 0.22127314
```

Loess figures in ggplot2

We will learn more about nonparametric regression later in the course. Plotting the loess regression may suggest a model specification that may not be immediately apparent. Take, for example, the relationship between birthweight and maternal age. We can read in the data and take a random sample of 5,000, just to speed things up a bit.

```
library(ggplot2)
data <- read.csv("../../../resources/ps1.csv")
var.names <- c("dbrwt", "dmage")
idx <- sample.int(nrow(data), 5000)
sm.data <- data[idx, var.names]
```

The loess model is a specific variant of locally weighted scatterplot smoothing regression analysis. Note that this is very similar to the lowess model, which are separated only by the smoothing technique. When all observations are given equal weight over the domain, both loess and lowess are equivalent to simple linear regression. However, locally weighted regression analysis place higher weight on "nearby" observations. The specification of what constitutes "nearby" observations and the method of weighting distinguishes the various models. Mostly, the following just presents examples that are useful for learning `ggplot2`. Fig. 1 presents the loess curve on its own, and Fig. 2 presents the same curve with the scatterplot points.

```
(p <- ggplot(sm.data, aes(x=dmage, y=dbrwt)) + geom_smooth(method = "loess", size = 1.5))
```

The curve in Fig. 1 suggests that the relationship between maternal age and infant birthweight may be quadratic. Mothers that are relatively young and old may give birth to lighter babies, potentially indicating poorer infant health. Moreover, the confidence intervals are wider toward the tails of the age distribution, since there are fewer observations in these age ranges. To examine the curve in context of the actual observations, we can add the actual data as a scatterplot in Fig. 2.

```
p + geom_point()
```

The quadratic relationship is interesting, and we can show that it is statistically significant, but even for the small subsample of the data, the loess model explains only a small portion of the variation in birthweight.

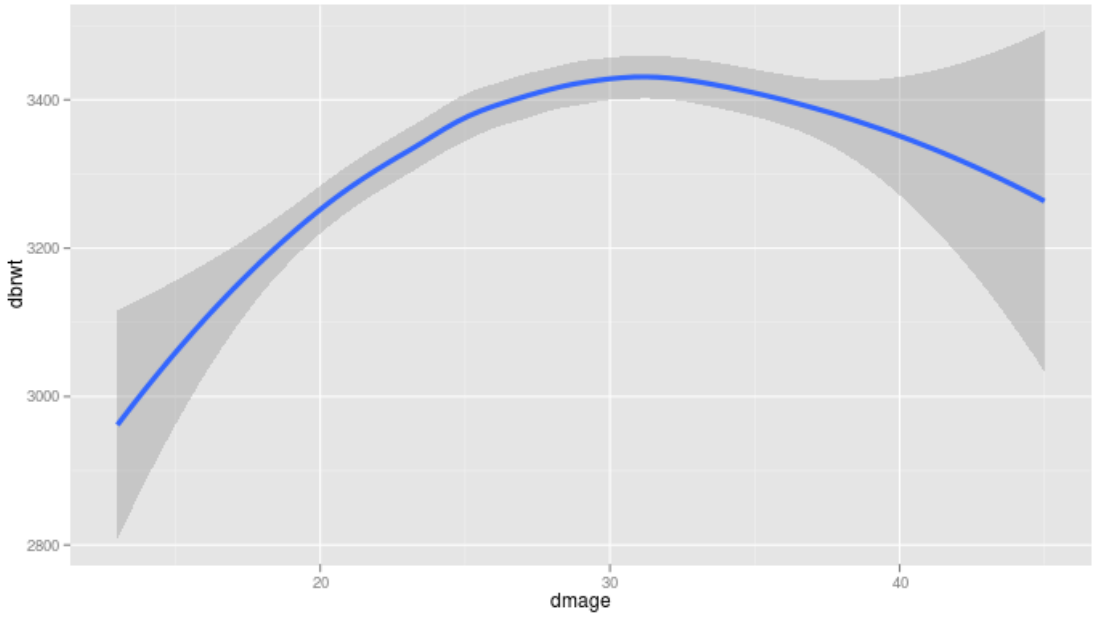

Figure 1: Loess regression

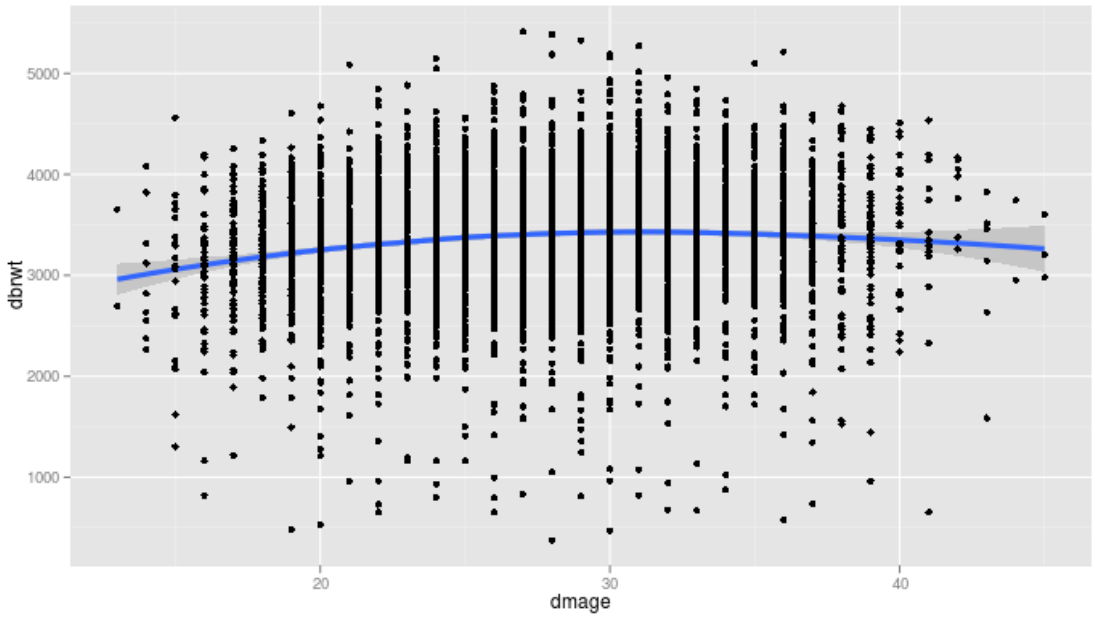

Figure 2: Loess regression with scatter plot

This is the fun (read: optional!) section to show how to manipulate and visualize spatial data in R. It seems more fun to start with a data story; and work through some basic analysis.

Data.gov is a data repository administered by the US government with over 445,000 geographic data sets. One dat set is the geographic coordinates and characteristics of 7,863 farmers marketsin the United States.[1] Suppose we are interested in examining the effect of state boundaries on the characteristics of farmers markets. Do state boundaries have a substantive impact on the character of farmers markets, or are the no better than arbitrary lines. There are rigorous ways to address this question, but we will just classify and plot farmers markets, looking for spatial patterns that follow state boundaries.

First, we export the farmers market data set as a CSV file, saving it to the data directory as `farmers-mkts.csv`. Let's just plot the distribution of farmers markets on a a base map of the United States. To do this, you will need to install and then load the following libraries:

```
library(maps)
library(maptools)
library(RColorBrewer)
library(classInt)
library(gpclib)
library(mapdata)
```

The following code plots the map, and presents the results in Figure 1.

```
data <- read.csv("../data/farmers-mkts.csv", header=TRUE)
map("state", interior = FALSE)
map("state", boundary = FALSE, col = "gray", add = TRUE)
points(data$x, data$y, cex = 0.2, col="blue")
```

The distribution of farmers markets across the US is neat to see, but there are so many points that it is difficult to visually glean any useful information, as seen in the following figure. So, instead, lets only consider the farmers markets in Colorado, Utah, New Mexico, and Arizona. There are 354 farmers markets in these four states.

```
statelist <- c("New Mexico", "Colorado", "Arizona", "Utah")
idx <- is.element(data$State, statelist)
state.data <- data[idx, ]
dim(state.data)
```

```
[1] 354  32
```

Each column of the `state.data` data frame contains information on a different attribute of the farmers markets. The last 24 columns are binary variables with entries `"Y"` or `"N"`, indicating whether the market sells cheese, for example, or accepts credit cards. These are the attributes of interest. The idea is whether we can predict the state of the farmers market, purely based on the characteristics. If so, then the state boundaries are correlated with the attributes for some reason — which is not included in the scope of this example. We can extract these characteristics into a matrix $\mathbf{X}$ and recode the string variables to 0-1 binary variables. Note that the default numeric levels of the string variables are 2 and 3, so subtracting 2 will yield the desired result.

```
X <- state.data[, 8:ncol(state.data)]
for(i in names(X)) {
  X[[i]] <- as.numeric(X[[i]]) - 2
}
```

[1] https://explore.data.gov/d/wfna-38ey

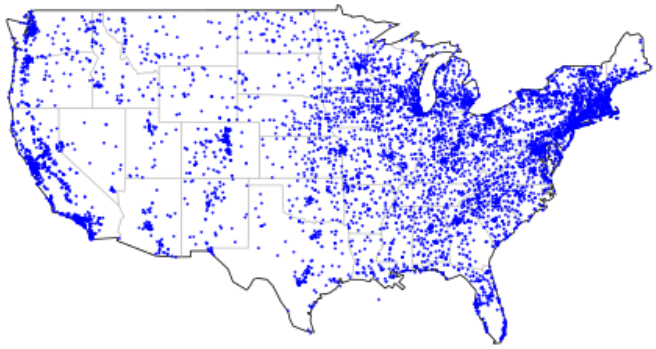

Figure 1: Distribution of US farmers markets

We want to categorize the markets based on their characteristics into four distinct buckets, indicating one of the four states. For this, we can use K-Means clustering, which is a simple call in `R`. The resulting object `cl` is a list with various output attributes of the clustering, including to which of the four clusters each market was assigned, indicated by `cl$cluster`. This is the vector we will use to color the points on the map. There are many other attributes that are collected in `cl`; three of them are listed below.

```
cl <- kmeans(X, 4)
names(cl)[1:3]
```

```
[1] "cluster" "centers" "totss"
```

The following code plots the points by zooming in on the four states, and coloring each market by the predicted state. There is slight variation in the output map, as `R` selects the exact colors on the fly. Below, we save the resulting image to `inserts/limited-mkts.png` and display one such image, produced previously. We set the number of colors to 4, one for each state; and the `brewer.pal()` function sets separate color codes for the number of supplied classes according to the color scheme `Set1`.

```
nclr <- 4
clr.set <- brewer.pal(nclr, "Set1")
```

There exactly four cluster values to match the four-color palette. However, the `classIntervals()` assigns each color in the palette to a range of values for the more general case. The `findColours()` function then maps the color codes to each point in the cluster vector; and this is the schema that is used to color the points on the map.

```
class <- classIntervals(cl$cluster, nclr, style = "pretty")
colcode <- findColours(class, cl$cluster)
```

The map is generated in much the same way as the full US map, except that the extent is limited by a latitude-longitude bounding box.

```
map("state", interior = FALSE,
    xlim = c(-117, -101), ylim = c(28, 43))
map("state", boundary = FALSE, col="gray", add = TRUE,
    xlim = c(-117, -101), ylim = c(28, 43))
points(state.data$x, state.data$y, pch = 16, col= colcode, cex = 1)
```

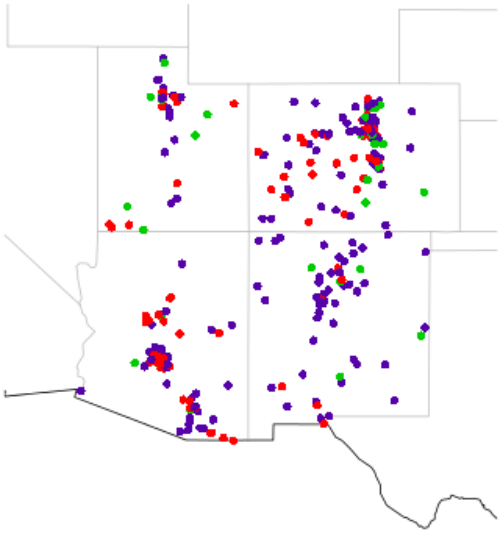

Figure 2: Distribution of US farmers markets

This is cool. It is clear that there is some separable variation in the characteristics of the markets. The red points seem to be clustered in New Mexico, the lower-right state. Even markets right across the border have characteristics that are different enough to "correctly" cluster them by state. There must be something about state regulations that are particularly imposing in New Mexico.

This is just one of many neat examples about how to use `R` to examine spatial variation; and to effectively uncover some spatial, data generating processes that are not obvious from looking at the data in a matrix or spreadsheet. More to come!