Exploratory Data Analysis in R

Advice for Getting Started on a Data Analysis

Dan Hall, Director of the SCC

Department of Statistics
Franklin College of Arts and Sciences
Statistical Consulting Center
UNIVERSITY OF GEORGIA

Table of Contents

Related Resources

- An R Markdown document in HTML format called `EDANotes.html` accompanies these slides and is available at https://tinyurl.com/2s4fkuas and via the QR code below.

- The talk will be given from this document as well as from `EDANotes.html`, the document linked above.

Related Resources

- An R Markdown document in HTML format called `EDANotes.html` accompanies these slides and is available at https://tinyurl.com/2s4fkuas and via the QR code below.

- The talk will be given from this document as well as from `EDANotes.html`, the document linked above.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Goals of EDA

- Identifying the scale of each variable. Determining the location, spread, and distributional shape of each variable.
- Detecting gross outliers, invalid data. Verifying the data import.
- Determining the extent and pattern of missingness.
- Detecting nonlinear relationships. Suggesting transformations.
- Detecting bivariate associations, potential interactions.
- For categorical data, finding categories with few observations that perhaps should be combined with others.
- Suggesting zero-inflation.

Not a goal:

- Normality is best assessed from the residuals of a model, *not* on a univariate distribution.
- Inference. EDA suggests; formal analysis confirms.

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn...

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn…

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn...

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn...

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in “wide” or “tall” format;
- are the data delimited (and what’s the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don’t know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn...

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn…

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn...

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn…

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn...

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Looking at the Data File

Unless the data set is very large, it is always a good idea to look at the file containing the data.
We typically want to learn...

- how the data are organized;
- what types of variables are there (e.g., character, numeric, date);
- does the data file have a header with variable names;
- are there extra rows or columns of non-data (e.g., comments, tables);
- how have missing values been coded;
- how have categorical variables/non-numeric data been coded;
- how have dates been formatted (12/27/1966, 27-12-66, etc.);
- are there inconsistencies in the organization or formatting of data;
- if relevant, are the data in "wide" or "tall" format;
- are the data delimited (and what's the delimiter?), in fixed-width columns, or something else.

Often we have a data dictionary or documentation file of some sort. But you don't know if such documentation is accurate until *you look at the data!*

Reading Data into R

- `read.table()` for space delimited,
- `read.delim()` for tab delimited,
- `read.csv()` for comma-delimited,
- `read.fwf()` for data in fixed-width columns.
- `readr` package (part of the tidyverse) has alternate versions of the functions above (e.g., `read_table()`) that offer speed advantage, other minor improvements.
- `haven` package has `read_sas()`, `read_spss()`, `read_stata()`.
- `readxl::read_excel()` for reading data from Excel files.

Reading Data into R

- `read.table()` for space delimited,
- `read.delim()` for tab delimited,
- `read.csv()` for comma-delimited,
- `read.fwf()` for data in fixed-width columns.
- `readr` package (part of the tidyverse) has alternate versions of the functions above (e.g., `read_table()`) that offer speed advantage, other minor improvements.
- `haven` package has `read_sas()`, `read_spss()`, `read_stata()`.
- `readxl::read_excel()` for reading data from Excel files.

Reading Data into R

- `read.table()` for space delimited,
- `read.delim()` for tab delimited,
- `read.csv()` for comma-delimited,
- `read.fwf()` for data in fixed-width columns.
- `readr` package (part of the tidyverse) has alternate versions of the functions above (e.g., `read_table()`) that offer speed advantage, other minor improvements.
- `haven` package has `read_sas()`, `read_spss()`, `read_stata()`.
- `readxl::read_excel()` for reading data from Excel files.

Reading Data into R

- `read.table()` for space delimited,
- `read.delim()` for tab delimited,
- `read.csv()` for comma-delimited,
- `read.fwf()` for data in fixed-width columns.
- `readr` package (part of the tidyverse) has alternate versions of the functions above (e.g., `read_table()`) that offer speed advantage, other minor improvements.
- `haven` package has `read_sas()`, `read_spss()`, `read_stata()`.
- `readxl::read_excel()` for reading data from Excel files.

Reading Data into R

- `read.table()` for space delimited,
- `read.delim()` for tab delimited,
- `read.csv()` for comma-delimited,
- `read.fwf()` for data in fixed-width columns.
- `readr` package (part of the tidyverse) has alternate versions of the functions above (e.g., `read_table()`) that offer speed advantage, other minor improvements.
- haven package has `read_sas()`, `read_spss()`, `read_stata()`.
- `readxl::read_excel()` for reading data from Excel files.

Reading Data into R

- `read.table()` for space delimited,
- `read.delim()` for tab delimited,
- `read.csv()` for comma-delimited,
- `read.fwf()` for data in fixed-width columns.
- `readr` package (part of the tidyverse) has alternate versions of the functions above (e.g., `read_table()`) that offer speed advantage, other minor improvements.
- `haven` package has `read_sas()`, `read_spss()`, `read_stata()`.
- `readxl::read_excel()` for reading data from Excel files.

Reading Data into R

- `read.table()` for space delimited,
- `read.delim()` for tab delimited,
- `read.csv()` for comma-delimited,
- `read.fwf()` for data in fixed-width columns.
- `readr` package (part of the tidyverse) has alternate versions of the functions above (e.g., `read_table()`) that offer speed advantage, other minor improvements.
- `haven` package has `read_sas()`, `read_spss()`, `read_stata()`.
- `readxl::read_excel()` for reading data from Excel files.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`, ...
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`,. . . .
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`, ...
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`,. . . .
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`,. . . .
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
    - `dead` better than `status`;
    - never use `x1`, `x2`,. . . .
- consistent
    - `insPlan`, `dataSource`, `ageGroup`
    - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`,. . . .
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`,. . . .
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - not `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Data files often have headers with variable names.

- This is convenient, but those names are not always good choices.
- Renaming the variables will avoid much inconvenient typing and/or confusion from non-descriptive variable names.

Names should be

- short and easy to type (no spaces, not in all CAPS);
- suggestive of the variable content
  - `dead` better than `status`;
  - never use `x1`, `x2`,....
- consistent
  - `insPlan`, `dataSource`, `ageGroup`
  - **not** `InsurancePlan`, `data.source`, `AGE_GROUP`.

Naming Variables

Factors:

- For factors, keep two versions: a numeric or character version, and a factor. Name them appropriately. E.g., `ageGroupNum` and `ageGroupFac`.
- For factors with ordered levels, use `levels=` to put them in proper order.
  - There is an ordered factor class, but it is rarely needed. Use it sparingly.
- Use `labels=` to attach labels to a factor whose levels are not self-explanatory.

```
# suppose we have opinions from a survey with the following responses from n=5 subjects:
opinNum <- c(1,3,3,2,1)
(opinFac <- factor(opinNum,levels=1:3,labels=c("disagree","neutral","agree")))
```

```
[1] disagree agree    agree    neutral  disagree
Levels: disagree neutral agree
```

- Avoid creating factors "on the fly" in function calls and model formulas, but do take transformations on the fly.

```
# Don't do this:
m1 <- lm(y~factor(trt)+logAge,data=myData)
# Do this:
myData$trtFac <- factor(myData$trt,levels=1:3,labels=c("Ctrl","A","B"))
m1 <- lm(y~trtFac+log(Age),data=myData)
```

Naming Variables

Factors:

- For factors, keep two versions: a numeric or character version, and a factor. Name them appropriately. E.g., `ageGroupNum` and `ageGroupFac`.
- For factors with ordered levels, use `levels=` to put them in proper order.
  - There is an `ordered` factor class, but it is rarely needed. Use it sparingly.
- Use `labels=` to attach labels to a factor whose levels are not self-explanatory.

```
# suppose we have opinions from a survey with the following responses from n=5 subjects:
opinNum <- c(1,3,3,2,1)
(opinFac <- factor(opinNum,levels=1:3,labels=c("disagree","neutral","agree")))
```

```
[1] disagree agree    agree    neutral  disagree
Levels: disagree neutral agree
```

- Avoid creating factors “on the fly” in function calls and model formulas, but do take transformations on the fly.

```
# Don't do this:
m1 <- lm(y~factor(trt)+logAge,data=myData)
# Do this:
myData$trtFac <- factor(myData$trt,levels=1:3,labels=c("Ctrl","A","B"))
m1 <- lm(y~trtFac+log(Age),data=myData)
```

Naming Variables

Factors:

- For factors, keep two versions: a numeric or character version, and a factor. Name them appropriately. E.g., `ageGroupNum` and `ageGroupFac`.
- For factors with ordered levels, use `levels=` to put them in proper order.
  - There is an `ordered` factor class, but it is rarely needed. Use it sparingly.
- Use `labels=` to attach labels to a factor whose levels are not self-explanatory.

```
# suppose we have opinions from a survey with the following responses from n=5 subjects:
opinNum <- c(1,3,3,2,1)
(opinFac <- factor(opinNum,levels=1:3,labels=c("disagree","neutral","agree")))
```

```
[1] disagree agree    agree    neutral  disagree
Levels: disagree neutral agree
```

- Avoid creating factors “on the fly” in function calls and model formulas, but do take transformations on the fly.

```
# Don't do this:
m1 <- lm(y~factor(trt)+logAge,data=myData)
# Do this:
myData$trtFac <- factor(myData$trt,levels=1:3,labels=c("Ctrl","A","B"))
m1 <- lm(y~trtFac+log(Age),data=myData)
```

Naming Variables

Factors:

- For factors, keep two versions: a numeric or character version, and a factor. Name them appropriately. E.g., `ageGroupNum` and `ageGroupFac`.
- For factors with ordered levels, use `levels=` to put them in proper order.
    - There is an `ordered` factor class, but it is rarely needed. Use it sparingly.
- Use `labels=` to attach labels to a factor whose levels are not self-explanatory.

```
# suppose we have opinions from a survey with the following responses from n=5 subjects:
opinNum <- c(1,3,3,2,1)
(opinFac <- factor(opinNum,levels=1:3,labels=c("disagree","neutral","agree")))
```

```
[1] disagree agree    agree    neutral  disagree
Levels: disagree neutral agree
```

- Avoid creating factors “on the fly” in function calls and model formulas, but do take transformations on the fly.

```
# Don't do this:
m1 <- lm(y~factor(trt)+logAge,data=myData)
# Do this:
myData$trtFac <- factor(myData$trt,levels=1:3,labels=c("Ctrl","A","B"))
m1 <- lm(y~trtFac+log(Age),data=myData)
```

Naming Variables

Factors:

- For factors, keep two versions: a numeric or character version, and a factor. Name them appropriately. E.g., `ageGroupNum` and `ageGroupFac`.
- For factors with ordered levels, use `levels=` to put them in proper order.
  - There is an `ordered` factor class, but it is rarely needed. Use it sparingly.
- Use `labels=` to attach labels to a factor whose levels are not self-explanatory.

```
# suppose we have opinions from a survey with the following responses from n=5 subjects:
opinNum <- c(1,3,3,2,1)
(opinFac <- factor(opinNum,levels=1:3,labels=c("disagree","neutral","agree")))
```

```
[1] disagree agree    agree    neutral  disagree
Levels: disagree neutral agree
```

- Avoid creating factors "on the fly" in function calls and model formulas, but do take transformations on the fly.

```
# Don't do this:
m1 <- lm(y~factor(trt)+logAge,data=myData)
# Do this:
myData$trtFac <- factor(myData$trt,levels=1:3,labels=c("Ctrl","A","B"))
m1 <- lm(y~trtFac+log(Age),data=myData)
```

Summary Statistics

There are many functions in R that produce summary statistics for many variables quickly.

- Running functions like `mean()` and `fivenum()` on each variable separately is too slow and doesn't produce compact results for a report.

Better tools:

1. `base::summary()`

- Not just for summarizing models.
- Applied to a data frame, it produces a compact summary of each variable.
- For numeric variables it gives a five-number summary, the mean, and a count of `NA`s (if any).
- For factors it gives a frequency distribution and a count of `NA`s (if any).

Summary Statistics

There are many functions in R that produce summary statistics for many variables quickly.

- Running functions like `mean()` and `fivenum()` on each variable separately is too slow and doesn't produce compact results for a report.

Better tools:

1. `base::summary()`

- Not just for summarizing models.
- Applied to a data frame, it produces a compact summary of each variable.
- For numeric variables it gives a five-number summary, the mean, and a count of `NA`s (if any).
- For factors it gives a frequency distribution and a count of `NA`s (if any).

Summary Statistics

There are many functions in R that produce summary statistics for many variables quickly.

- Running functions like `mean()` and `fivenum()` on each variable separately is too slow and doesn't produce compact results for a report.

Better tools:

1. `base::summary()`

- Not just for summarizing models.
- Applied to a data frame, it produces a compact summary of each variable.
- For numeric variables it gives a five-number summary, the mean, and a count of `NA`s (if any).
- For factors it gives a frequency distribution and a count of `NA`s (if any).

Summary Statistics

There are many functions in R that produce summary statistics for many variables quickly.

- Running functions like `mean()` and `fivenum()` on each variable separately is too slow and doesn't produce compact results for a report.

Better tools:

1. `base::summary()`

- Not just for summarizing models.
- Applied to a data frame, it produces a compact summary of each variable.
- For numeric variables it gives a five-number summary, the mean, and a count of `NA`s (if any).
- For factors it gives a frequency distribution and a count of `NA`s (if any).

Summary Statistics

There are many functions in R that produce summary statistics for many variables quickly.

- Running functions like `mean()` and `fivenum()` on each variable separately is too slow and doesn't produce compact results for a report.

Better tools:

1. `base::summary()`

- Not just for summarizing models.
- Applied to a data frame, it produces a compact summary of each variable.
- For numeric variables it gives a five-number summary, the mean, and a count of `NA`s (if any).
- For factors it gives a frequency distribution and a count of `NA`s (if any).

Summary Statistics

There are many functions in R that produce summary statistics for many variables quickly.

- Running functions like `mean()` and `fivenum()` on each variable separately is too slow and doesn't produce compact results for a report.

Better tools:

1. `base::summary()`

- Not just for summarizing models.
- Applied to a data frame, it produces a compact summary of each variable.
- For numeric variables it gives a five-number summary, the mean, and a count of `NA`s (if any).
- For factors it gives a frequency distribution and a count of `NA`s (if any).

Summary Statistics

2. `DescTools::Desc()`

- Like `summary()`, this function is generic and is useful for several classes of R objects.
- When applied to a data frame, it gives a more extensive summary of the data frame and each variable than does `summary()`.
- By default, a plot is produced for each variable, but these plots can be suppressed (use `plotit=FALSE`).
- The results of `Desc()` are excellent, but too voluminous for some purposes.

Summary Statistics

2. `DescTools::Desc()`

- Like `summary()`, this function is generic and is useful for several classes of R objects.
- When applied to a data frame, it gives a more extensive summary of the data frame and each variable than does `summary()`.
- By default, a plot is produced for each variable, but these plots can be suppressed (use `plotit=FALSE`).
- The results of `Desc()` are excellent, but too voluminous for some purposes.

Summary Statistics

2. `DescTools::Desc()`

- Like `summary()`, this function is generic and is useful for several classes of R objects.
- When applied to a data frame, it gives a more extensive summary of the data frame and each variable than does `summary()`.
- By default, a plot is produced for each variable, but these plots can be suppressed (use `plotit=FALSE`).
- The results of `Desc()` are excellent, but too voluminous for some purposes.

Summary Statistics

2. `DescTools::Desc()`

- Like `summary()`, this function is generic and is useful for several classes of R objects.
- When applied to a data frame, it gives a more extensive summary of the data frame and each variable than does `summary()`.
- By default, a plot is produced for each variable, but these plots can be suppressed (use `plotit=FALSE`).
- The results of `Desc()` are excellent, but too voluminous for some purposes.

Summary Statistics

2. `DescTools::Desc()`

- Like `summary()`, this function is generic and is useful for several classes of R objects.
- When applied to a data frame, it gives a more extensive summary of the data frame and each variable than does `summary()`.
- By default, a plot is produced for each variable, but these plots can be suppressed (use `plotit=FALSE`).
- The results of `Desc()` are excellent, but too voluminous for some purposes.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
  - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
  - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
    - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
  - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
    - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
    - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
  - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
  - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

3. `skimr::skim()`

- Compactly summarizes a data frame and each variable in it.
- Different summaries depending on variable class.
- Yields a data frame that can be further processed.
- Works well with tidyverse methods.
- Prints nicely in documents rendered by `knitr` (e.g., R Markdown documents).
    - Chunk option `skimr_include_summary=FALSE` can be used to suppress the summary of the data frame (used in some cases below).
- Customizable.
- Default summaries for numeric variables include spark plots that don't render to pdf easily, so use `skim_without_charts()` instead of `skim()` to suppress those plots.

Summary Statistics

4. `Hmisc::describe()`

- Produces a compact and thorough summary of each variable, but it includes obscure statistics and is not customizable.

5. Others:

- `psych::describe()` produces a very compact set of summary statistics and will give statistics by group, but doesn't handle factors well;
- `summarytools::dfSummary()` produces nice results for html and Word, but is glacially slow when converting its results to pdf format.

Summary Statistics

4. `Hmisc::describe()`

- Produces a compact and thorough summary of each variable, but it includes obscure statistics and is not customizable.

5. Others:

- `psych::describe()` produces a very compact set of summary statistics and will give statistics by group, but doesn't handle factors well;
- `summarytools::dfSummary()` produces nice results for html and Word, but is glacially slow when converting its results to pdf format.

Summary Statistics

4. `Hmisc::describe()`

- Produces a compact and thorough summary of each variable, but it includes obscure statistics and is not customizable.

5. Others:

- `psych::describe()` produces a very compact set of summary statistics and will give statistics by group, but doesn't handle factors well;
- `summarytools::dfSummary()` produces nice results for html and Word, but is glacially slow when converting its results to pdf format.

Summary Statistics

4. `Hmisc::describe()`

- Produces a compact and thorough summary of each variable, but it includes obscure statistics and is not customizable.

5. Others:

- `psych::describe()` produces a very compact set of summary statistics and will give statistics by group, but doesn't handle factors well;
- `summarytools::dfSummary()` produces nice results for html and Word, but is glacially slow when converting its results to pdf format.

Summary Statistics

4. `Hmisc::describe()`

- Produces a compact and thorough summary of each variable, but it includes obscure statistics and is not customizable.

5. Others:

- `psych::describe()` produces a very compact set of summary statistics and will give statistics by group, but doesn't handle factors well;
- `summarytools::dfSummary()` produces nice results for html and Word, but is glacially slow when converting its results to pdf format.

Grouped Data Summaries

Often we want summary statistics separated by the levels of one or more grouping factors.

- E.g., we may wish to obtain summary statistics separately for male and female respondents. That is, we want results *by gender.*
- Such operations are sometimes referred to as *by-group processing.*

There are many ways to do by-group processing in R.

- The `doBy` package is devoted to tasks of this sort. And the function `doBy::summaryBy` is particularly useful.
- But the most powerful set of tools for by-group processing is in the `dplyr` package, part of the tidyverse.
- Currently, `dplyr` handles by-group processing through the use of *grouped data frames.*
- These are of class `grouped_df` and can be created using the `dplyr::group_by()` function.
    - See `vignette("grouping",package="dplyr")` for details, but we illustrate with several examples in `EDANotes.html`.

Grouped Data Summaries

Often we want summary statistics separated by the levels of one or more grouping factors.

- E.g., we may wish to obtain summary statistics separately for male and female respondents. That is, we want results *by gender.*
- Such operations are sometimes referred to as *by-group processing.*

There are many ways to do by-group processing in R.

- The `doBy` package is devoted to tasks of this sort. And the function `doBy::summaryBy` is particularly useful.
- But the most powerful set of tools for by-group processing is in the `dplyr` package, part of the tidyverse.
- Currently, `dplyr` handles by-group processing through the use of *grouped data frames.*
- These are of class `grouped_df` and can be created using the `dplyr::group_by()` function.
  - See `vignette("grouping",package="dplyr")` for details, but we illustrate with several examples in `EDANotes.html`.

Grouped Data Summaries

Often we want summary statistics separated by the levels of one or more grouping factors.

- E.g., we may wish to obtain summary statistics separately for male and female respondents. That is, we want results *by gender.*
- Such operations are sometimes referred to as *by-group processing.*

There are many ways to do by-group processing in R.

- The `doBy` package is devoted to tasks of this sort. And the function `doBy::summaryBy` is particularly useful.
- But the most powerful set of tools for by-group processing is in the `dplyr` package, part of the tidyverse.
- Currently, `dplyr` handles by-group processing through the use of *grouped data frames.*
- These are of class `grouped_df` and can be created using the `dplyr::group_by()` function.
  - See `vignette("grouping",package="dplyr")` for details, but we illustrate with several examples in `EDANotes.html`.

Grouped Data Summaries

Often we want summary statistics separated by the levels of one or more grouping factors.

- E.g., we may wish to obtain summary statistics separately for male and female respondents. That is, we want results *by gender*.
- Such operations are sometimes referred to as *by-group processing*.

There are many ways to do by-group processing in R.

- The `doBy` package is devoted to tasks of this sort. And the function `doBy::summaryBy` is particularly useful.
- But the most powerful set of tools for by-group processing is in the `dplyr` package, part of the tidyverse.
- Currently, `dplyr` handles by-group processing through the use of *grouped data frames*.
- These are of class `grouped_df` and can be created using the `dplyr::group_by()` function.
  - See `vignette("grouping",package="dplyr")` for details, but we illustrate with several examples in `EDANotes.html`.

Grouped Data Summaries

Often we want summary statistics separated by the levels of one or more grouping factors.

- E.g., we may wish to obtain summary statistics separately for male and female respondents. That is, we want results *by gender.*
- Such operations are sometimes referred to as *by-group processing.*

There are many ways to do by-group processing in R.

- The `doBy` package is devoted to tasks of this sort. And the function `doBy::summaryBy` is particularly useful.
- But the most powerful set of tools for by-group processing is in the `dplyr` package, part of the tidyverse.
- Currently, `dplyr` handles by-group processing through the use of *grouped data frames.*
- These are of class `grouped_df` and can be created using the `dplyr::group_by()` function.
    - See `vignette("grouping",package="dplyr")` for details, but we illustrate with several examples in `EDANotes.html`.

Grouped Data Summaries

Often we want summary statistics separated by the levels of one or more grouping factors.

- E.g., we may wish to obtain summary statistics separately for male and female respondents. That is, we want results *by gender.*
- Such operations are sometimes referred to as *by-group processing.*

There are many ways to do by-group processing in R.

- The `doBy` package is devoted to tasks of this sort. And the function `doBy::summaryBy` is particularly useful.
- But the most powerful set of tools for by-group processing is in the `dplyr` package, part of the tidyverse.
- Currently, `dplyr` handles by-group processing through the use of *grouped data frames.*
- These are of class `grouped_df` and can be created using the `dplyr::group_by()` function.
  - See `vignette("grouping",package="dplyr")` for details, but we illustrate with several examples in `EDANotes.html`.

Grouped Data Summaries

Often we want summary statistics separated by the levels of one or more grouping factors.

- E.g., we may wish to obtain summary statistics separately for male and female respondents. That is, we want results *by gender.*
- Such operations are sometimes referred to as *by-group processing.*

There are many ways to do by-group processing in R.

- The `doBy` package is devoted to tasks of this sort. And the function `doBy::summaryBy` is particularly useful.
- But the most powerful set of tools for by-group processing is in the `dplyr` package, part of the tidyverse.
- Currently, `dplyr` handles by-group processing through the use of *grouped data frames.*
- These are of class `grouped_df` and can be created using the `dplyr::group_by()` function.
    - See `vignette("grouping",package="dplyr")` for details, but we illustrate with several examples in `EDANotes.html`.

Plotting the Data

There is much to say about the design and implementation of effective graphics, but here we concentrate on the main types of plots to use when doing EDA and how to construct them in R.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Univariate plots:

- Produce univariate plots of each continuous variable. All of the following are useful:
  - Box plots
  - Density plots
  - Histograms
  - Frequency polygons
  - Dot plots
- For factors, univariate plots of the frequency distribution (e.g., bar charts) are nice, but often add little over a numeric frequency distribution.
  - For the latter, use `DescTools::PercTable()` and include percentages instead of just getting counts with `base::table()`.
- Functions like `DescTools::Desc` make it easy to get univariate plots quickly, but you may want to re-plot some variables differently or with more polish.
  - See examples in `EDANotes.html`.

Bivariate plots:

- If there is a Y vs X distinction, plot the response variable versus each X variable in a pair-wise manner. Repeat for additional responses if present.
- The most useful bivariate plots depend on the scale of the variables involved.
  - See Table 1 below and examples in `EDANotes.html`.

Table 1: Plots for bivariate relationships between Y (response) and X (explanatory)

Scale of Y	Scale of X	Plot Type(s)
Continuous	Continuous	Scatter plot
Continuous	Categorical	Side-by-side box, violin, or dot plots; Faceted histograms; Faceted or overlaid density plots or frequency polygons
Dichotomous	Continuous	Conditional density plots, scatter plots with binned averages
Dichotomous	Categorical	Mosaic plots
Polytomous	Continuous	Conditional density plots
Polytomous	Categorical	Mosaic plots

Bivariate plots:

- If there is a Y vs X distinction, plot the response variable versus each X variable in a pair-wise manner. Repeat for additional responses if present.
- The most useful bivariate plots depend on the scale of the variables involved.
  - See Table 1 below and examples in `EDANotes.html`.

Table 1: Plots for bivariate relationships between Y (response) and X (explanatory)

Scale of Y	Scale of X	Plot Type(s)
Continuous	Continuous	Scatter plot
Continuous	Categorical	Side-by-side box, violin, or dot plots; Faceted histograms; Faceted or overlaid density plots or frequency polygons
Dichotomous	Continuous	Conditional density plots, scatter plots with binned averages
Dichotomous	Categorical	Mosaic plots
Polytomous	Continuous	Conditional density plots
Polytomous	Categorical	Mosaic plots

Bivariate plots:

- If there is a Y vs X distinction, plot the response variable versus each X variable in a pair-wise manner. Repeat for additional responses if present.
- The most useful bivariate plots depend on the scale of the variables involved.
  - See Table 1 below and examples in `EDANotes.html`.

Table 1: Plots for bivariate relationships between Y (response) and X (explanatory)

Scale of Y	Scale of X	Plot Type(s)
Continuous	Continuous	Scatter plot
Continuous	Categorical	Side-by-side box, violin, or dot plots; Faceted histograms; Faceted or overlaid density plots or frequency polygons
Dichotomous	Continuous	Conditional density plots, scatter plots with binned averages
Dichotomous	Categorical	Mosaic plots
Polytomous	Continuous	Conditional density plots
Polytomous	Categorical	Mosaic plots

Plot matrices

Scatterplot matrices are useful for getting all pairwise scatter plots between several variables.

- The `GGally::ggpairs()` function extends this concept to get pairwise plots of various types, depending on the scales of the variables involved.
  - The diagonal typically shows univariate distribution plots.
  - This function is customizable to control the types of plots that it produces on the diagonal and in each triangle of the matrix.
- For small sets of variables, plot matrices are a very useful tool to plot the data quickly and compactly.

Plot matrices

Scatterplot matrices are useful for getting all pairwise scatter plots between several variables.

- The `GGally::ggpairs()` function extends this concept to get pairwise plots of various types, depending on the scales of the variables involved.
  - The diagonal typically shows univariate distribution plots.
  - This function is customizable to control the types of plots that it produces on the diagonal and in each triangle of the matrix.
- For small sets of variables, plot matrices are a very useful tool to plot the data quickly and compactly.

Plot matrices

Scatterplot matrices are useful for getting all pairwise scatter plots between several variables.

- The `GGally::ggpairs()` function extends this concept to get pairwise plots of various types, depending on the scales of the variables involved.
  - The diagonal typically shows univariate distribution plots.
  - This function is customizable to control the types of plots that it produces on the diagonal and in each triangle of the matrix.
- For small sets of variables, plot matrices are a very useful tool to plot the data quickly and compactly.

Plot matrices

Scatterplot matrices are useful for getting all pairwise scatter plots between several variables.

- The `GGally::ggpairs()` function extends this concept to get pairwise plots of various types, depending on the scales of the variables involved.
  - The diagonal typically shows univariate distribution plots.
  - This function is customizable to control the types of plots that it produces on the diagonal and in each triangle of the matrix.
- For small sets of variables, plot matrices are a very useful tool to plot the data quickly and compactly.

Plots for visualizing conditional association

Bivariate relationships often differ across the levels of one or more additional variables.

- A two-way relationship can be stronger or weaker—or even qualitatively different—depending on a third variable.
- When this is the case, a bivariate plot may be simplistic or misleading.

When we suspect that the `Y` vs `X` relationship depends on `Z`, plotting the conditional relationship may give us important insight. This arises most commonly when `Z` is a factor.

- In this case, we can stratify the `Y` by `X` plot into different panels corresponding to the values of `Z`. This is known as *faceting*.
- Alternatively, we can use different plotting symbols at each level of `Z` (e.g., scatter plots) or examine the distribution of `Y` at combinations of the levels of `X` and `Z` (e.g., grouped side-by-side box plots, mosaic plots)
- See Table 2 below and examples in `EDANotes.html`.

Plots for visualizing conditional association

Bivariate relationships often differ across the levels of one or more additional variables.

- A two-way relationship can be stronger or weaker—or even qualitatively different—depending on a third variable.
- When this is the case, a bivariate plot may be simplistic or misleading.

When we suspect that the `Y` vs `X` relationship depends on `Z`, plotting the conditional relationship may give us important insight. This arises most commonly when `Z` is a factor.

- In this case, we can stratify the `Y` by `X` plot into different panels corresponding to the values of `Z`. This is known as *faceting*.
- Alternatively, we can use different plotting symbols at each level of `Z` (e.g., scatter plots) or examine the distribution of `Y` at combinations of the levels of `X` and `Z` (e.g., grouped side-by-side box plots, mosaic plots)
- See Table 2 below and examples in `EDANotes.html`.

Plots for visualizing conditional association

Bivariate relationships often differ across the levels of one or more additional variables.

- A two-way relationship can be stronger or weaker—or even qualitatively different—depending on a third variable.
- When this is the case, a bivariate plot may be simplistic or misleading.

When we suspect that the `Y` vs `X` relationship depends on `Z`, plotting the conditional relationship may give us important insight. This arises most commonly when `Z` is a factor.

- In this case, we can stratify the `Y` by `X` plot into different panels corresponding to the values of `Z`. This is known as *faceting*.
- Alternatively, we can use different plotting symbols at each level of `Z` (e.g., scatter plots) or examine the distribution of `Y` at combinations of the levels of `X` and `Z` (e.g., grouped side-by-side box plots, mosaic plots)
- See Table 2 below and examples in `EDANotes.html`.

Plots for visualizing conditional association

Bivariate relationships often differ across the levels of one or more additional variables.

- A two-way relationship can be stronger or weaker—or even qualitatively different—depending on a third variable.
- When this is the case, a bivariate plot may be simplistic or misleading.

When we suspect that the `Y` vs `X` relationship depends on `Z`, plotting the conditional relationship may give us important insight. This arises most commonly when `Z` is a factor.

- In this case, we can stratify the `Y` by `X` plot into different panels corresponding to the values of `Z`. This is known as *faceting*.
- Alternatively, we can use different plotting symbols at each level of `Z` (e.g., scatter plots) or examine the distribution of `Y` at combinations of the levels of `X` and `Z` (e.g., grouped side-by-side box plots, mosaic plots)
- See Table 2 below and examples in `EDANotes.html`.

Plots for visualizing conditional association

Bivariate relationships often differ across the levels of one or more additional variables.

- A two-way relationship can be stronger or weaker—or even qualitatively different—depending on a third variable.
- When this is the case, a bivariate plot may be simplistic or misleading.

When we suspect that the `Y` vs `X` relationship depends on `Z`, plotting the conditional relationship may give us important insight. This arises most commonly when `Z` is a factor.

- In this case, we can stratify the `Y` by `X` plot into different panels corresponding to the values of `Z`. This is known as *faceting*.
- Alternatively, we can use different plotting symbols at each level of `Z` (e.g., scatter plots) or examine the distribution of `Y` at combinations of the levels of `X` and `Z` (e.g., grouped side-by-side box plots, mosaic plots)
- See Table 2 below and examples in `EDANotes.html`.

Plots for visualizing conditional association

Table 2: Plots for bivariate relationships between Y and X, conditional on Z

Scale of Y	Scale of X	Scale of Z	Plot Type(s)
Continuous	Continuous	Categorical	Scatter plots with different plotting symbols and different fits, faceted scatter plots
Continuous	Categorical	Categorical	Grouped or faceted side-by-side box, violin, or dot plots; doubly-faceted histograms, density plots, or frequency polygons; faceted and superimposed density plots or frequency polygons
Categorical	Continuous	Categorical	Faceted conditional density plots, scatter plots with binned averages, or mosaic plots with binned values of X
Categorical	Categorical	Categorical	Mosaic plots or faceted mosaic plots
		Continuous	Bin Z and use one of the methods above

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.,, `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate**.
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.,, `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate**.
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.„ `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate.**
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.„ `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate.**
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.,, `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate.**
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.,, `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate**.
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.„ `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate**.
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plotting correlations

Correlation heatmaps are a good way to summarize pairwise correlations between variables. An example can be found in `EDANotes.html`.

- Such plots can be produced with, e.g.„ `corrplot::corrplot.mixed()`.
- It is easier to quickly understand patterns, magnitudes, and directions of association from such plots than from numeric correlation matrices.

Warnings:

- Don't rely on heatmaps without examining scatter plots with, e.g., `ggpairs()`.
  - If variables are related nonlinearly, transform to linearity before computing Pearson correlations or use Spearman (rank) correlations.
  - Spearman correlations and partial correlations can also be summarized with heatmaps.
- **Do not include variables for which correlations are inappropriate**.
  - E.g., correlations are inappropriate for dichotomous and nominal polytomous variables, so leave them out of the heatmap.
  - Ordinal polytomous variables can be included, but use Spearman correlations in that case.

Plots for exploring missingness

- Plots of the extent and pattern of missingness in a data set are often helpful.
  - Which variables have missing data and how much?
  - How many cases have missing data on at least one variable?
  - Which pairs or groups of variables tend to be missing together?
- Good tools for addressing these questions can be found in the `visdat`, `naniar`, `mice` and `VIM` packages.
  - See `vignette("using_visdat",package="visdat")`, `vignette("naniar-visualization",package="naniar")`, and `vignette("VisualImp",package="VIM")` for details.
- Examples can be found in `EDANotes.html`.

Plots for exploring missingness

- Plots of the extent and pattern of missingness in a data set are often helpful.
  - Which variables have missing data and how much?
  - How many cases have missing data on at least one variable?
  - Which pairs or groups of variables tend to be missing together?
- Good tools for addressing these questions can be found in the `visdat`, `naniar`, `mice` and `VIM` packages.
  - See `vignette("using_visdat",package="visdat")`, `vignette("naniar-visualization",package="naniar")`, and `vignette("VisualImp",package="VIM")` for details.
- Examples can be found in `EDANotes.html`.

Plots for exploring missingness

- Plots of the extent and pattern of missingness in a data set are often helpful.
  - Which variables have missing data and how much?
  - How many cases have missing data on at least one variable?
  - Which pairs or groups of variables tend to be missing together?
- Good tools for addressing these questions can be found in the `visdat`, `naniar`, `mice` and `VIM` packages.
  - See `vignette("using_visdat",package="visdat")`, `vignette("naniar-visualization",package="naniar")`, and `vignette("VisualImp",package="VIM")` for details.
- Examples can be found in `EDANotes.html`.

Plots for exploring missingness

- Plots of the extent and pattern of missingness in a data set are often helpful.
  - Which variables have missing data and how much?
  - How many cases have missing data on at least one variable?
  - Which pairs or groups of variables tend to be missing together?
- Good tools for addressing these questions can be found in the `visdat`, `naniar`, `mice` and `VIM` packages.
  - See `vignette("using_visdat",package="visdat")`, `vignette("naniar-visualization",package="naniar")`, and `vignette("VisualImp",package="VIM")` for details.
- Examples can be found in `EDANotes.html`.

Plots for exploring missingness

- Plots of the extent and pattern of missingness in a data set are often helpful.
  - Which variables have missing data and how much?
  - How many cases have missing data on at least one variable?
  - Which pairs or groups of variables tend to be missing together?
- Good tools for addressing these questions can be found in the `visdat`, `naniar`, `mice` and `VIM` packages.
  - See `vignette("using_visdat",package="visdat")`, `vignette("naniar-visualization",package="naniar")`, and `vignette("VisualImp",package="VIM")` for details.
- Examples can be found in `EDANotes.html`.

Plots for exploring missingness

- Plots of the extent and pattern of missingness in a data set are often helpful.
  - Which variables have missing data and how much?
  - How many cases have missing data on at least one variable?
  - Which pairs or groups of variables tend to be missing together?
- Good tools for addressing these questions can be found in the `visdat`, `naniar`, `mice` and `VIM` packages.
  - See `vignette("using_visdat",package="visdat")`, `vignette("naniar-visualization",package="naniar")`, and `vignette("VisualImp",package="VIM")` for details.
- Examples can be found in `EDANotes.html`.

Plots for exploring missingness

- Plots of the extent and pattern of missingness in a data set are often helpful.
  - Which variables have missing data and how much?
  - How many cases have missing data on at least one variable?
  - Which pairs or groups of variables tend to be missing together?
- Good tools for addressing these questions can be found in the `visdat`, `naniar`, `mice` and `VIM` packages.
  - See `vignette("using_visdat",package="visdat")`, `vignette("naniar-visualization",package="naniar")`, and `vignette("VisualImp",package="VIM")` for details.
- Examples can be found in `EDANotes.html`.

R Packages Used in the Talk

```
library(tidyverse) # ggplot2, forcats, dplyr, tidyr, etc.
library(DescTools) # Desc() function for univariate summaries
library(skimr)     # descriptive stats
library(Hmisc)     # describe() function for descriptives.
library(doBy)      # by-group processing
library(car)       # for brief() function
library(ggmosaic)  # mosaic plots with gpplot2
library(gridExtra) # grid.arrange() puts multiple ggplots on a page
library(GGally)    # for ggpairs()
library(corrplot)  # correlation heatmaps
library(naniar)    # for missing data visualizations and stats
library(visdat)    # for missing data visualizations
library(VIM)       # for missing data visualizations
library(kableExtra)# nice tables
```

General R Programming and Graphics:

- `tidyverse` includes many packages developed by Hadley Wickham and the team at Posit. W.r.t. this talk, its most important packages are
  - `ggplot2` Graphics. See my SCC Seminar on ggplot2.
  - `forcats` a collection of useful tools for working with factors in R. See my SCC Seminar on factors.
  - `dplyr` various tools for programming in R.
  - `magrittr` the incredibly useful pipe operator (%>%).
  - `tidyr` tools for organizing data sets in R.

R Packages Used in the Talk

```
library(tidyverse) # ggplot2, forcats, dplyr, tidyr, etc.
library(DescTools) # Desc() function for univariate summaries
library(skimr)     # descriptive stats
library(Hmisc)     # describe() function for descriptives.
library(doBy)      # by-group processing
library(car)       # for brief() function
library(ggmosaic)  # mosaic plots with ggplot2
library(gridExtra) # grid.arrange() puts multiple ggplots on a page
library(GGally)    # for ggpairs()
library(corrplot)  # correlation heatmaps
library(naniar)    # for missing data visualizations and stats
library(visdat)    # for missing data visualizations
library(VIM)       # for missing data visualizations
library(kableExtra)# nice tables
```

General R Programming and Graphics:

- `tidyverse` includes many packages developed by Hadley Wickham and the team at Posit. W.r.t. this talk, its most important packages are
  - `ggplot2` Graphics. See my SCC Seminar on ggplot2.
  - `forcats` a collection of useful tools for working with factors in R. See my SCC Seminar on factors.
  - `dplyr` various tools for programming in R.
  - `magrittr` the incredibly useful pipe operator (%>%).
  - `tidyr` tools for organizing data sets in R.

R Packages Used in the Talk

```
library(tidyverse) # ggplot2, forcats, dplyr, tidyr, etc.
library(DescTools) # Desc() function for univariate summaries
library(skimr)     # descriptive stats
library(Hmisc)     # describe() function for descriptives.
library(doBy)      # by-group processing
library(car)       # for brief() function
library(ggmosaic)  # mosaic plots with gpplot2
library(gridExtra) # grid.arrange() puts multiple ggplots on a page
library(GGally)    # for ggpairs()
library(corrplot)  # correlation heatmaps
library(naniar)    # for missing data visualizations and stats
library(visdat)    # for missing data visualizations
library(VIM)       # for missing data visualizations
library(kableExtra)# nice tables
```

General R Programming and Graphics:

- `tidyverse` includes many packages developed by Hadley Wickham and the team at Posit. W.r.t. this talk, its most important packages are
  - `ggplot2` Graphics. See my SCC Seminar on ggplot2.
  - `forcats` a collection of useful tools for working with factors in R. See my SCC Seminar on factors.
  - `dplyr` various tools for programming in R.
  - `magrittr` the incredibly useful pipe operator (%>%).
  - `tidyr` tools for organizing data sets in R.

R Packages Used in the Talk

```
library(tidyverse) # ggplot2, forcats, dplyr, tidyr, etc.
library(DescTools) # Desc() function for univariate summaries
library(skimr)     # descriptive stats
library(Hmisc)     # describe() function for descriptives.
library(doBy)      # by-group processing
library(car)       # for brief() function
library(ggmosaic)  # mosaic plots with gpplot2
library(gridExtra) # grid.arrange() puts multiple ggplots on a page
library(GGally)    # for ggpairs()
library(corrplot)  # correlation heatmaps
library(naniar)    # for missing data visualizations and stats
library(visdat)    # for missing data visualizations
library(VIM)       # for missing data visualizations
library(kableExtra)# nice tables
```

General R Programming and Graphics:

- `tidyverse` includes many packages developed by Hadley Wickham and the team at Posit. W.r.t. this talk, its most important packages are
  - `ggplot2` Graphics. See my SCC Seminar on ggplot2.
  - `forcats` a collection of useful tools for working with factors in R. See my SCC Seminar on factors.
  - `dplyr` various tools for programming in R.
  - `magrittr` the incredibly useful pipe operator (%>%).
  - `tidyr` tools for organizing data sets in R.

R Packages Used in the Talk

```
library(tidyverse) # ggplot2, forcats, dplyr, tidyr, etc.
library(DescTools) # Desc() function for univariate summaries
library(skimr)     # descriptive stats
library(Hmisc)     # describe() function for descriptives.
library(doBy)      # by-group processing
library(car)       # for brief() function
library(ggmosaic)  # mosaic plots with gpplot2
library(gridExtra) # grid.arrange() puts multiple ggplots on a page
library(GGally)    # for ggpairs()
library(corrplot)  # correlation heatmaps
library(naniar)    # for missing data visualizations and stats
library(visdat)    # for missing data visualizations
library(VIM)       # for missing data visualizations
library(kableExtra)# nice tables
```

General R Programming and Graphics:

- `tidyverse` includes many packages developed by Hadley Wickham and the team at Posit. W.r.t. this talk, its most important packages are
  - `ggplot2` Graphics. See my SCC Seminar on ggplot2.
  - `forcats` a collection of useful tools for working with factors in R. See my SCC Seminar on factors.
  - `dplyr` various tools for programming in R.
  - `magrittr` the incredibly useful pipe operator (`%>%`).
  - `tidyr` tools for organizing data sets in R.

R Packages Used in the Talk

```
library(tidyverse) # ggplot2, forcats, dplyr, tidyr, etc.
library(DescTools) # Desc() function for univariate summaries
library(skimr)     # descriptive stats
library(Hmisc)     # describe() function for descriptives.
library(doBy)      # by-group processing
library(car)       # for brief() function
library(ggmosaic)  # mosaic plots with gpplot2
library(gridExtra) # grid.arrange() puts multiple ggplots on a page
library(GGally)    # for ggpairs()
library(corrplot)  # correlation heatmaps
library(naniar)    # for missing data visualizations and stats
library(visdat)    # for missing data visualizations
library(VIM)       # for missing data visualizations
library(kableExtra)# nice tables
```

General R Programming and Graphics:

- `tidyverse` includes many packages developed by Hadley Wickham and the team at Posit. W.r.t. this talk, its most important packages are
  - `ggplot2` Graphics. See my SCC Seminar on ggplot2.
  - `forcats` a collection of useful tools for working with factors in R. See my SCC Seminar on factors.
  - `dplyr` various tools for programming in R.
  - `magrittr` the incredibly useful pipe operator (`%>%`).
  - `tidyr` tools for organizing data sets in R.

R Packages Used in the Talk

Descriptive Statistics:

- `DescTools` Many tools for descriptive statistics, EDA, categorical data analysis, and one- and two-sample inference. The function `Desc()` is featured here.
- `skimr` Contains `skim()` function and others useful for obtaining univariate summary statistics.
- `Hmisc` Mentioned here for the `describe()` function that reports univariate summaries.
- `doBy` Contains `summaryBy()` and other functions for by-group processing. Useful and simple, but newer tools in `dplyr` are more powerful.

R Packages Used in the Talk

Descriptive Statistics:

- `DescTools` Many tools for descriptive statistics, EDA, categorical data analysis, and one- and two-sample inference. The function `Desc()` is featured here.
- `skimr` Contains `skim()` function and others useful for obtaining univariate summary statistics.
- `Hmisc` Mentioned here for the `describe()` function that reports univariate summaries.
- `doBy` Contains `summaryBy()` and other functions for by-group processing. Useful and simple, but newer tools in `dplyr` are more powerful.

R Packages Used in the Talk

Descriptive Statistics:

- `DescTools` Many tools for descriptive statistics, EDA, categorical data analysis, and one- and two-sample inference. The function `Desc()` is featured here.
- `skimr` Contains `skim()` function and others useful for obtaining univariate summary statistics.
- `Hmisc` Mentioned here for the `describe()` function that reports univariate summaries.
- `doBy` Contains `summaryBy()` and other functions for by-group processing. Useful and simple, but newer tools in `dplyr` are more powerful.

R Packages Used in the Talk

Descriptive Statistics:

- `DescTools` Many tools for descriptive statistics, EDA, categorical data analysis, and one- and two-sample inference. The function `Desc()` is featured here.
- `skimr` Contains `skim()` function and others useful for obtaining univariate summary statistics.
- `Hmisc` Mentioned here for the `describe()` function that reports univariate summaries.
- `doBy` Contains `summaryBy()` and other functions for by-group processing. Useful and simple, but newer tools in `dplyr` are more powerful.

R Packages Used in the Talk

Graphics:

- `ggmosaic` tools for producing mosaic plots with `ggplot2`.
- `gridExtra` Used here for the `grid.arrange()` function for displaying multiple ggplots on a page.
- `GGally` Used here for the `ggpairs()` function that produces a nice matrix of pairwise bivariate plots.
- `corrplot` For correlation heatmaps.
- `naniar`, `visdat`, and `VIM` for visualizing missingness.

Miscellaneous:

- `car` Has many tools for working with regression results. Used here for the `brief()` function that gives a compact print-out of a data frame.

R Packages Used in the Talk

Graphics:

- `ggmosaic` tools for producing mosaic plots with `ggplot2`.
- `gridExtra` Used here for the `grid.arrange()` function for displaying multiple ggplots on a page.
- `GGally` Used here for the `ggpairs()` function that produces a nice matrix of pairwise bivariate plots.
- `corrplot` For correlation heatmaps.
- `naniar`, `visdat`, and `VIM` for visualizing missingness.

Miscellaneous:

- `car` Has many tools for working with regression results. Used here for the `brief()` function that gives a compact print-out of a data frame.

R Packages Used in the Talk

Graphics:

- `ggmosaic` tools for producing mosaic plots with `ggplot2`.
- `gridExtra` Used here for the `grid.arrange()` function for displaying multiple ggplots on a page.
- `GGally` Used here for the `ggpairs()` function that produces a nice matrix of pairwise bivariate plots.
- `corrplot` For correlation heatmaps.
- `naniar`, `visdat`, and `VIM` for visualizing missingness.

Miscellaneous:

- `car` Has many tools for working with regression results. Used here for the `brief()` function that gives a compact print-out of a data frame.

R Packages Used in the Talk

Graphics:

- `ggmosaic` tools for producing mosaic plots with `ggplot2`.
- `gridExtra` Used here for the `grid.arrange()` function for displaying multiple ggplots on a page.
- `GGally` Used here for the `ggpairs()` function that produces a nice matrix of pairwise bivariate plots.
- `corrplot` For correlation heatmaps.
- `naniar`, `visdat`, and `VIM` for visualizing missingness.

Miscellaneous:

- `car` Has many tools for working with regression results. Used here for the `brief()` function that gives a compact print-out of a data frame.

R Packages Used in the Talk

Graphics:

- `ggmosaic` tools for producing mosaic plots with `ggplot2`.
- `gridExtra` Used here for the `grid.arrange()` function for displaying multiple ggplots on a page.
- `GGally` Used here for the `ggpairs()` function that produces a nice matrix of pairwise bivariate plots.
- `corrplot` For correlation heatmaps.
- `naniar`, `visdat`, and `VIM` for visualizing missingness.

Miscellaneous:

- `car` Has many tools for working with regression results. Used here for the `brief()` function that gives a compact print-out of a data frame.

R Packages Used in the Talk

Graphics:

- `ggmosaic` tools for producing mosaic plots with `ggplot2`.
- `gridExtra` Used here for the `grid.arrange()` function for displaying multiple ggplots on a page.
- `GGally` Used here for the `ggpairs()` function that produces a nice matrix of pairwise bivariate plots.
- `corrplot` For correlation heatmaps.
- `naniar`, `visdat`, and `VIM` for visualizing missingness.

Miscellaneous:

- `car` Has many tools for working with regression results. Used here for the `brief()` function that gives a compact print-out of a data frame.

Thanks

- If you need assistance with EDA or with any statistical design or analysis task, please contact the SCC.
  - www.stat.uga/consulting
- We can help!

Thank you!

Thanks

- If you need assistance with EDA or with any statistical design or analysis task, please contact the SCC.
  - www.stat.uga/consulting
- We can help!

Thank you!

Thanks

- If you need assistance with EDA or with any statistical design or analysis task, please contact the SCC.
  - www.stat.uga/consulting
- We can help!

Thank you!

Questions?

Questions?