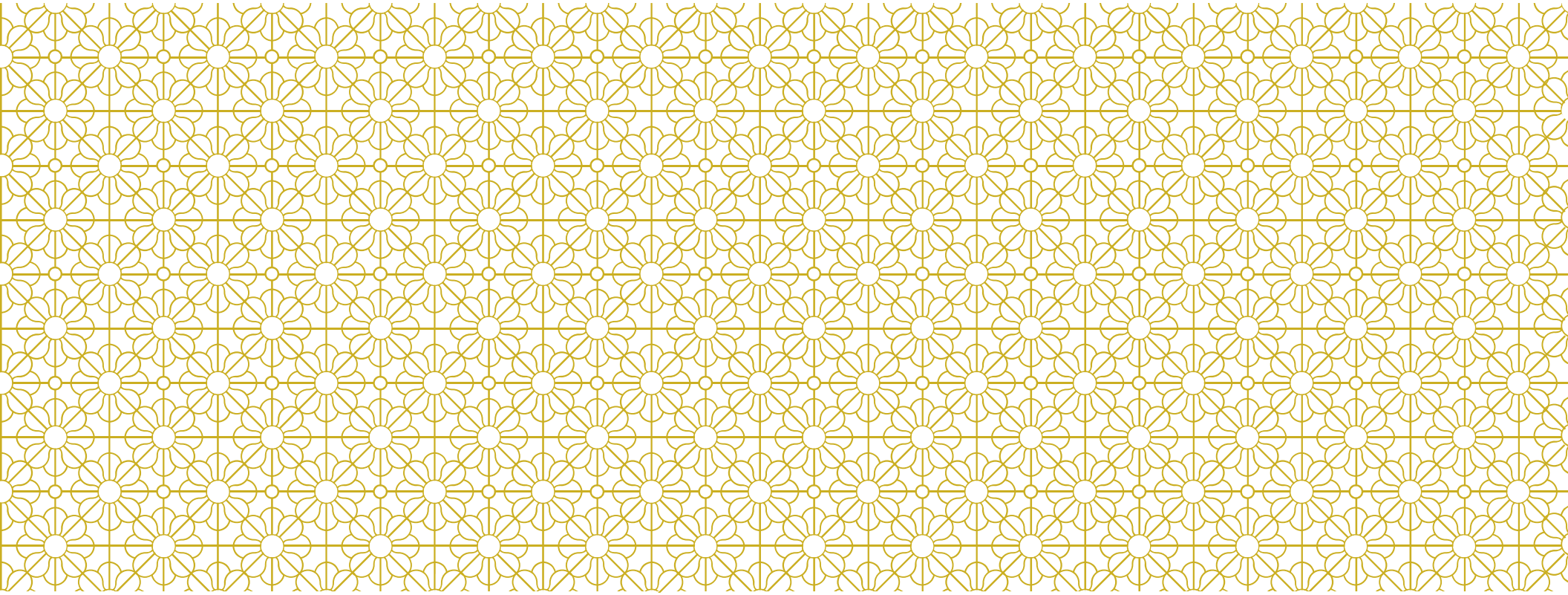


THE DATA PART OF BIG DATA: CURRENT TOPICS IN DATA PRE-PROCESSING FOR PREDICTIVE MODELING

CAS 2019 RPM Seminar

Louise Francis, FCAS, CSPA, MAAA

AGENDA

- Data Preprocessing for Predictive Modeling
- *The tidy universe: tidyverse packages*
  - The data sets we are using
  - Read data into R
  - Reshape data
  - SQL like data munging
  - Tools for Text data

Feature Engineering Topics

Please note that we plan for this to be an interactive session geared towards audience interest and participation.

CRISP DM

- Describes life cycle for data mining

See:
https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.crispdm.help/crisp_overview.htm

- Has six phases with feedback loop

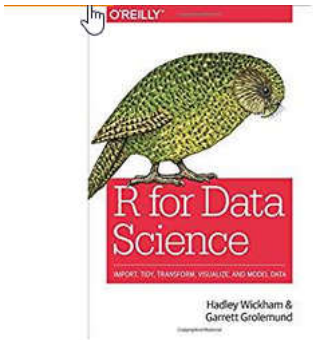
6 PHASES OF CRISP DM

1. Business understanding
2. **Data Understanding**
3. **Data Preprocessing**
4. Modeling
5. Evaluation
6. Deployment

HADLEY WICKHAM

Hadley Wickham: http://hadley.nz/

- Well known R programmer of R libraries
- Chief scientist at Rstudio
- One of his first packages was reshape package for aggregating and organizing data



R for Data Science: Import, Tidy, Transform, Visualize, and Model Data Jan 5, 2017

$18.35 - $23.62 prime

Paperback, Kindle Edition

86

HADLEY WICKHAM

- Developed *ggplot2* package
- Developed *dplyr* package
- Books include *R for Data Science* and *ggplot2*
- Proposed tidy data framework

- Described in paper *"Tidy Data" in Journal of Statistical Software*

DATA CLEANING AND TIDY DATA

- Data cleaning and preparing is the most time consuming part of most analytics projects: the rule of thumb is that it consumes 80% of the time
- Wickham notes that there has been little research into how to clean data well
- Tidy data addresses a key aspect of data cleaning: organizing the data we receive into a structure that can be used for analysis

TIDY DATA PRINCIPLES

- Provides principles on how to organize data
- To make data cleaning easier
- Which led to tools to make the process easier and more efficient

WICKHAM'S DEFINITIONS

- A data set is a collection of values
  - quantitively
  - qualitative
- A variable contains all values that measure the same attribute (such as paid losses)
- An observation contains all values measured on the same unit (such as accident year)

MULTIPLE WAYS TO ORGANIZE DATA

- Two views of data
  - Database form
  - Spread out form – columns names could represent data
- Both can be tidy depending on the application

SPREAD OUT INTO COLUMNS

A common way actuaries organize data

- Loss development example
- Rows are accident year
- Columns are development age
- Values (paid or incurred) are recorded under development age

TRIANGLE DATA

The spread out form

Cumulative Paid Losses																		
Accident	Months of Development																	
Year	12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216
1974	$267	$1,975	$4,587	$7,375	$10,661	$15,232	$17,888	$18,541	$18,937	$19,130	$19,189	$19,209	$19,234	$19,234	$19,246	$19,246	$19,246	$19,246
1975	310	2,809	5,686	9,386	14,884	20,654	22,017	22,529	22,772	22,821	23,042	23,060	23,127	23,127	23,127	23,127	23,159	
1976	370	2,744	7,281	13,287	19,773	23,888	25,174	25,819	26,049	26,180	26,268	26,364	26,371	26,379	26,397	26,397		
1977	577	3,877	9,612	16,962	23,764	26,712	28,393	29,656	29,839	29,944	29,997	29,999	29,999	30,049	30,049			
1978	509	4,518	12,067	21,218	27,194	29,617	30,854	31,240	31,598	31,889	32,002	31,947	31,965	31,986				
1979	630	5,763	16,372	24,105	29,091	32,531	33,878	34,185	34,290	34,420	34,479	34,498	34,524					
1980	1,078	8,066	17,518	26,091	31,807	33,883	34,820	35,482	35,607	35,937	35,957	35,962						
1981	1,646	9,378	18,034	26,652	31,253	33,376	34,287	34,985	35,122	35,161	35,172							
1982	1,754	11,256	20,624	27,857	31,360	33,331	34,061	34,227	34,317	34,378								
1983	1,997	10,628	21,015	29,014	33,788	36,329	37,446	37,571	37,681									
1984	2,164	11,538	21,549	29,167	34,440	36,528	36,950	37,099										
1985	1,922	10,939	21,357	28,488	32,982	35,330	36,059											
1986	1,962	13,053	27,869	38,560	44,461	45,988												
1987	2,329	18,086	38,099	51,953	58,029													
1988	3,343	239,383	52,054	66,203														
1989	3,847	34,171	59,232															
1990	6,090	33,392																
1991	5,451																	

TRIANGLE DATA -SNAPSHOT

Values spread across development month

Accident	Months of Development				
Year	12	24	36	48	60
1974	$267	$1,975	$4,587	$7,375	$10,661
1975	310	2,809	5,686	9,386	14,884
1976	370	2,744	7,281	13,287	19,773
1977	577	3,877	9,612	16,962	23,764
1978	509	4,518	12,067	21,218	27,194
1979	630	5,763	16,372	24,105	29,091

LONG DATABASE FORMAT

All paid values in one column, one variable

Year	Development Age (Years)	Cumulative paid
1974	1	267
1975	1	310
1976	1	370
1977	1	577
1978	1	509
1979	1	630
1980	1	1,078
1981	1	1,646
1982	1	1,754
1983	1	1,997
1984	1	2,164
1985	1	1,922
1986	1	1,962

NON-TRIANGULAR FORM

Loss reserving data can be in non-triangle form

- One year, one development age on each row
- One value such as incurred or paid on each row
- For modeling applications, one might want multiple values (incurred, paid, closing rates) in one row

CLAIM DATA

WC Claim dataset

From Crowd Analytics competition

Obs_ID	Dependent	Average Weekly Wage	Body Part	Body Part Code	Cause	Cause Code	Claimant Age	Claimant Gender	Claimant Gender Code	Claimant Hire Date	Claimant Marital Status
Obs_1	98679	500	Pelvis	46	Struck or Injured By	1700	21	Female	F	4/3/2001	
Obs_2	55727	1,037.00	Low Back Area	42	Strain or Injury By	1500		Male	M	5/15/2001	
Obs_3	185833	929	Low Back Area	42	Strain or Injury By	1500	63	Male	M	5/15/2001	Married
Obs_4	98615	1,226.00	Multiple Body Parts	90	Strain or Injury By	1500	49	Male	M		
Obs_5	51396		Other Facial Soft Tissue	18	Miscellaneous Causes	1900	51	Male	M		

WICKHAM'S DEFINITION OF TIDY DATA

- Each variable has its own column
- Each observation has its own row
- Each value has its own cell

TYPES OF MESSINESS

- Multiple observations are in a row spread out over columns
- Column headers contain values
- One observation is in multiple rows
- One column contains two or more variables

TIDY DATA RELATED TOOLS

- *readr* library
- read_csv
  - Reads data in as a tibble
  - Similar to a data frame
  - Note that tibbles can deal with column variables that are numbers (such as development age)
- *tidyr* library

TIDYR FUNCTIONS

tidyr library can be used to reorganize data and make it tidy

- gather
- spread
- separate

- When two variables are stored as a string in one column
  - unite

```
##  [1] "Year" "X1"   "X2"   "X3"   "X4"   "X5"   "X6"   "X7"   "X8"   "X9"
## [11] "X10"  "X11"  "X12"  "X13"  "X14"  "X15"  "X16"  "X17"  "X18"
```

```
head(paidtri)
```

```
##   Year  X1   X2    X3    X4    X5    X6    X7    X8    X9   X10   X11
## 1 1974 267 1975  4587  7375 10661 15232 17888 18541 18937 19130 19189
## 2 1975 310 2809  5686  9386 14884 20654 22017 22529 22772 22821 23042
## 3 1976 370 2744  7281 13287 19773 23888 25174 25819 26049 26180 26268
## 4 1977 577 3877  9612 16962 23764 26712 28393 29656 29839 29944 29997
## 5 1978 509 4518 12067 21218 27194 29617 30854 31240 31598 31889 32002
## 6 1979 630 5763 16372 24105 29091 32531 33878 34185 34290 34420 34479
##     X12   X13   X14   X15   X16   X17   X18
## 1 19209 19234 19234 19246 19246 19246 19246
## 2 23060 23127 23127 23127 23127 23159    NA
## 3 26364 26371 26379 26397 26397    NA    NA
## 4 29999 29999 30049 30049    NA    NA    NA
## 5 31947 31965 31986    NA    NA    NA    NA
## 6 34498 34524    NA    NA    NA    NA    NA
```

NOW USE *READR* LIBRARY

- Load *readr* package and set filename

```
library(readr)
myCSVfile<-"C:/CLRS/AutoPDVariables.csv“
```

- Read in data

```
paidtri<-read_csv(file=myCSVfile)
```

- Print names and some records

```
names(paidtri)
head(paidtri)
```

```
names(paidtri)
```

```
##  [1] "Year" "1"    "2"    "3"    "4"    "5"    "6"    "7"    "8"    "9"
## [11] "10"   "11"   "12"   "13"   "14"   "15"   "16"   "17"   "18"
```

```
head(paidtri)
```

```
## # A tibble: 6 x 19
##    Year   `1`   `2`   `3`   `4`   `5`   `6`   `7`   `8`   `9`  `10`  `11`
##   <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
## 1  1974   267  1975  4587  7375 10661 15232 17888 18541 18937 19130 19189
## 2  1975   310  2809  5686  9386 14884 20654 22017 22529 22772 22821 23042
## 3  1976   370  2744  7281 13287 19773 23888 25174 25819 26049 26180 26268
## 4  1977   577  3877  9612 16962 23764 26712 28393 29656 29839 29944 29997
## 5  1978   509  4518 12067 21218 27194 29617 30854 31240 31598 31889 32002
## 6  1979   630  5763 16372 24105 29091 32531 33878 34185 34290 34420 34479
## # ... with 7 more variables: `12` <int>, `13` <int>, `14` <int>,
## #   `15` <int>, `16` <int>, `17` <int>, `18` <int>
```

NOW GATHER THE TRIANGLE DATA

- Apply tidyr::*gather* function, specify new variable to be Devage

```
paidData<-gather(data=paidtri,'1':'18',key="Devage",value="paid")
```

- Print out top rows

```
head(paidData)
```

```
paidData<-gather(data=paidtri,'1':'18',key="Devage",value="paid")
head(paidData)
```

```
## # A tibble: 6 x 3
##    Year Devage  paid
##   <int> <chr>  <int>
## 1  1974 1        267
## 2  1975 1        310
## 3  1976 1        370
## 4  1977 1        577
## 5  1978 1        509
## 6  1979 1        630
```

TIDYVERSE

Set of related libraries that

- Read data efficiently (readr)
- Create tibble data (tibble)
- Tidy the data through reorganization (tidyr)
- Perform database management functions (diplyr)
- Perform string functions stringr()
- Perform EDA (ggplot2)
- Use code:

  >library(tidyverse)
- Make sure you have installed *tidyverse*

MORE ON *READR*: SPACES IN VARIABLE NAMES

```
[1] "Obs_ID"                        "Dependent"
 [3] "Average Weekly Wage"           "Body Part"
 [5] "Body Part Code"                "Cause"
 [7] "Cause Code"                    "Claimant Age"
 [9] "Claimant Atty Firm Name"       "Claimant Gender"
[11] "Claimant Gender Code"          "Claimant Hire Date"
[13] "Claimant Marital Status"       "Claimant Marital Status Code"
[15] "Claimant State"                "Claimant State Code"
[17] "Department Code"               "Detail Cause"
[19] "Detail Cause Code"             "Domestic vs. Foreign?"
[21] "Domestic vs. Foreign? Code"    "Dt Reported to Carrier/TPA"
[23] "Dt Reported to Employer"       "Employment Status"
[25] "Employment Status Code"        "Date of Injury/Illness"
[27] "Handling Office Name"          "How Injury Occurred"
[29] "Injury/Illness City"           "Injury/Illness Postal"
[31] "Injury/Illness State"          "Injury/Illness State Code"
[33] "Jurisdiction"                  "Jurisdiction Code"
[35] "Lost Time or Medical Only?"    "Lost Time or Medical Only? Code"
[37] "Nature of Injury/Illness"      "Nature of Injury/Illness Code"
[39] "Number of Dependents"          "OSHA Injury Type"
[41] "OSHA Injury Type Code"         "Severity Index Code"
[43] "Severity Index Code Code"      "Time of Injury/Illness"
[45] "Type of Loss"                  "Type of Loss Code"
[47] "Policy Year"                   "Reforms_dummy"
```

MAKE.NAMES

- The *make.names* function can be used to eliminate the spaces
- This makes the variables easier to work with in R

```
myfile="D:/RPM Data/CompClaimsTrain.csv“
wcdata<-read_csv(myfile)
names(wcdata)
names(wcdata)=make.names(names(wcdata),unique=TRUE)
```

VARIABLE NAMES – SPACES REPLACED

```
 1] "Obs_ID"                          "Dependent"
  [3] "Average.Weekly.Wage"             "Body.Part"
  [5] "Body.Part.Code"                  "Cause"
  [7] "Cause.Code"                      "Claimant.Age"
  [9] "Claimant.Atty.Firm.Name"         "Claimant.Gender"
 [11] "Claimant.Gender.Code"            "Claimant.Hire.Date"
 [13] "Claimant.Marital.Status"         "Claimant.Marital.Status.Code"
 [15] "Claimant.State"                  "Claimant.State.Code"
 [17] "Department.Code"                 "Detail.Cause"
 [19] "Detail.Cause.Code"               "Domestic.vs..Foreign."
 [21] "Domestic.vs..Foreign..Code"      "Dt.Reported.to.Carrier.TPA"
 [23] "Dt.Reported.to.Employer"         "Employment.Status"
 [25] "Employment.Status.Code"          "Date.of.Injury.Illness"
 [27] "Handling.Office.Name"            "How.Injury.Occurred"
 [29] "Injury.Illness.City"             "Injury.Illness.Postal"
 [31] "Injury.Illness.State"            "Injury.Illness.State.Code"
 [33] "Jurisdiction"                    "Jurisdiction.Code"
 [35] "Lost.Time.or.Medical.Only."      "Lost.Time.or.Medical.Only..Code"
 [37] "Nature.of.Injury.Illness"        "Nature.of.Injury.Illness.Code"
 [39] "Number.of.Dependents"            "OSHA.Injury.Type"
 [41] "OSHA.Injury.Type.Code"           "Severity.Index.Code"
 [43] "Severity.Index.Code.Code"        "Time.of.Injury.Illness"
 [45] "Type.of.Loss"                    "Type.of.Loss.Code"
 [47] "Policy.Year"                     "Reforms_dummy"
```

PARSING VARIABLES

- When the data was read in the wrong data type was assigned to some variables
  - Date variables read in as character
- Use one of readr's parsing functions to transform to correct data type
  - parse_date(), parse_number(),parse_factor(), parse_logical()
  - Require a string vector
- Example:
- wcdata$Dt.Reported.to.Carrier.TPA=parse_date(wcdata$Dt.Reported.to.Carrier.TPA,"%m/%d/%Y")

GET BASIC DESCRIPTIVE STATISTICS: *SUMMARY()*

- Use summary function
- summary(wcdata)

```
Obs_ID            Dependent        Average.Weekly.Wage  Body.Part
 Length:15407       Min.   :      0   Min.   :   2        Length:15407
 Class :character   1st Qu.:    152   1st Qu.: 500        Class :character
 Mode  :character   Median :    446   Median :1000        Mode  :character
                    Mean   :  10285   Mean   :1148
                    3rd Qu.:   1704   3rd Qu.:1529
                    Max.   :3774290   Max.   :9999
                                          NA's   :9535
```

GET METADATA: *STR()*

```
> str(wcdata)
Classes ‘spec_tbl_df’, ‘tbl_df’, ‘tbl’ and 'data.frame':    15407 obs. of  4
8 variables:
 $ Obs_ID                        : chr  "Obs_1" "Obs_2" "Obs_3" "Obs_4" ...
 $ Dependent                     : num  98679 55727 185833 98615 51396 ...
 $ Average.Weekly.Wage           : num  500 1037 929 1226 NA ...
 $ Body.Part                     : chr  "Pelvis" "Low Back Area" "Low Back A
rea" "Multiple Body Parts" ...
 $ Body.Part.Code                : num  46 42 42 90 18 30 42 42 42 54 ...
 $ Cause                         : chr  "Struck or Injured By" "Strain or In
jury By" "Strain or Injury By" "Strain or Injury By" ...
 $ Cause.Code                    : num  1700 1500 1500 1500 1900 1500 1500 1
500 1500 1500 ...
 $ Claimant.Age                  : num  21 NA 63 49 51 55 49 36 45 45 ...
 $ Claimant.Atty.Firm.Name       : chr  NA NA "TROBINSON & CHUR ATTORNEYS AT
LAW" "IBARRY STEVENS;;M;;" ...
 $ Claimant.Gender               : chr  "Female" "Male" "Male" "Male" ...
 $ Claimant.Gender.Code          : chr  "F" "M" "M" "M" ...
 $ Claimant.Hire.Date            : chr  "4/3/2001" "5/15/2001" "5/15/2001" N
A ...
 $ Claimant.Marital.Status       : chr  NA NA "Married" NA ...
 $ Claimant.Marital.Status.Code  : chr  NA NA "M" NA ...
 $ Claimant.State                : chr  "California" "California" "Hawaii" "
Idaho" ...
 $ Claimant.State.Code           : chr  "CA" "CA" "HI" "ID" ...
 $ Department.Code               : logi  NA NA NA NA NA NA ...
 $ Detail.Cause                  : chr  "Struck by Falling/Flying Object" "S
train/Injury by Lifting" "Strain/Injury by Carrying" "Strain/Injury by Repeti
tive Motion" ...
 $ Detail.Cause.Code             : num  75 56 55 97 90 97 60 97 97 97 ...
 $ Domestic.vs..Foreign.         : chr  "Domestic" "Domestic" "Domestic" "Do
mestic" ...
 $ Domestic.vs..Foreign..Code    : chr  "D" "D" "D" "D" ...
 $ Dt.Reported.to.Carrier.TPA    : Date, format: "2001-04-17" "2001-05-25" .
..
 $ Dt.Reported.to.Employer       : Date, format: "2001-04-13" "2001-
05-24" .
```

DPLYR USED FOR DATA MANAGEMENT

- Optimized version of plyr package
- Use to subset, summarize, filter and join data
- R has functions such as subset and operators such as [ ] that can perform data management functions but they can be difficult to use
- We will use *dplyr* on Schedule P data downloaded from CAS web site
- First read in the data
- Note that *read_csv* provides some metadata

DPLYR GRAMMAR

- select() selects variables/columns from data frame
- filter() subsets rows of data frame based on logical conditions
- arrange() reorders rows of data frame
- rename() renames variables in data frame
- mutate() performs variable transformations and adds variables
- summarize() summarizes/aggregates data from a data frame
- %>% pipe operators used to link multiple verb actions together

DPLYR FUNCTIONS

- First argument is a data frame
- Subsequent arguments describe what is to be done
- You can refer to columns without using dollar operator
- Return result is a new data frame
- The dataframe needs to be tidy

WE DO NOT NEED ALL THE COLUMNS

```
[1] "Obs_ID"                        "Dependent"
 [3] "Average Weekly Wage"           "Body Part"
 [5] "Body Part Code"                "Cause"
 [7] "Cause Code"                    "Claimant Age"
 [9] "Claimant Atty Firm Name"       "Claimant Gender"
[11] "Claimant Gender Code"          "Claimant Hire Date"
[13] "Claimant Marital Status"       "Claimant Marital Status Code"
[15] "Claimant State"                "Claimant State Code"
[17] "Department Code"               "Detail Cause"
[19] "Detail Cause Code"             "Domestic vs. Foreign?"
[21] "Domestic vs. Foreign? Code"    "Dt Reported to Carrier/TPA"
[23] "Dt Reported to Employer"       "Employment Status"
[25] "Employment Status Code"        "Date of Injury/Illness"
[27] "Handling Office Name"          "How Injury Occurred"
[29] "Injury/Illness City"           "Injury/Illness Postal"
[31] "Injury/Illness State"          "Injury/Illness State Code"
[33] "Jurisdiction"                  "Jurisdiction Code"
[35] "Lost Time or Medical Only?"    "Lost Time or Medical Only? Code"
[37] "Nature of Injury/Illness"      "Nature of Injury/Illness Code"
[39] "Number of Dependents"          "OSHA Injury Type"
[41] "OSHA Injury Type Code"         "Severity Index Code"
[43] "Severity Index Code Code"      "Time of Injury/Illness"
[45] "Type of Loss"                  "Type of Loss Code"
[47] "Policy Year"                   "Reforms_dummy"
```

SELECT()

- We do not need all the columns
- Only keep those relevant to analysis
- Use *select()* to extract only needed columns
- Let's take the first 10 columns

```
wcdata2<-select(wcdata, 1:10)
names(wcdata2)
```

```
> names(wcdata2)
 [1] "Obs_ID"                "Dependent"
 [3] "Average.Weekly.Wage"   "Body.Part"
 [5] "Body.Part.Code"        "Cause"
 [7] "Cause.Code"            "Claimant.Age"
 [9] "Claimant.Atty.Firm.Name" "Claimant.Gender"
```

SELECT BY ELIMINATION

- Leave out variables we don't want by using a "-" within select function
- In this example, we eliminate variables that are duplicates of other variables, such as Jurisdiction and Jurisdiction.Code
- We will keep the "Code" variable

```
wcdata2<-select(wcdata, -Body.Part,-Cause,
-Claimant.Hire.Date,-Detail.Cause,-Injury.Illness.State,
-Jurisdiction, -Nature.of.Injury.Illness,
-OSHA.Injury.Type,-Type.of.Loss)
```

SELECT()

Can select columns based on patterns in name

- starts_with
- ends_with
- contains
- matches

FILTER()

- Selects rows based on filter applied to values in rows
- Similar to subset but faster
- Can be used to eliminate records with clearly erroneous values
- Such as selecting either lost time or medical only claims for modeling

```
wcdata3<-filter(wcdata2, Lost.Time.or.Medical.Only..Code=="MO")
```

- Can have multiple conditions (using &, ||)

ARRANGE()

Use arrange() for sorting

```
wcdata2<-arrange(subset,desc(Dependent))
wcdata4<-arrange(wcdata3, desc(Dependent))
wcdata4[1:5,1:4]
```

It is easier to use than sort()

Use desc(varname) to sort descending

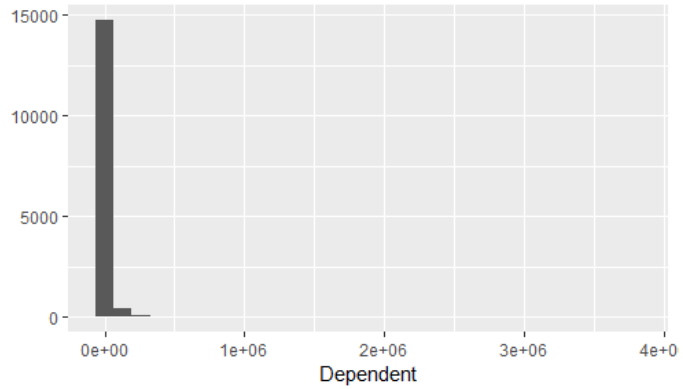
Can use .by_group to sort by a group

SORTED DATA

```
wcdata4[1:5,1:4]
# A tibble: 5 x 4
  Obs_ID    Dependent Average.Weekly.Wage Body.Part.Code
  <chr>         <dbl>               <dbl>           <dbl>
1 Obs_2673    3774290                 403              90
2 Obs_14692   1159631                3138              43
3 Obs_14673   1086522                 939              51
4 Obs_13477   1083067                2303              12
5 Obs_578     1076883                1050              41
```

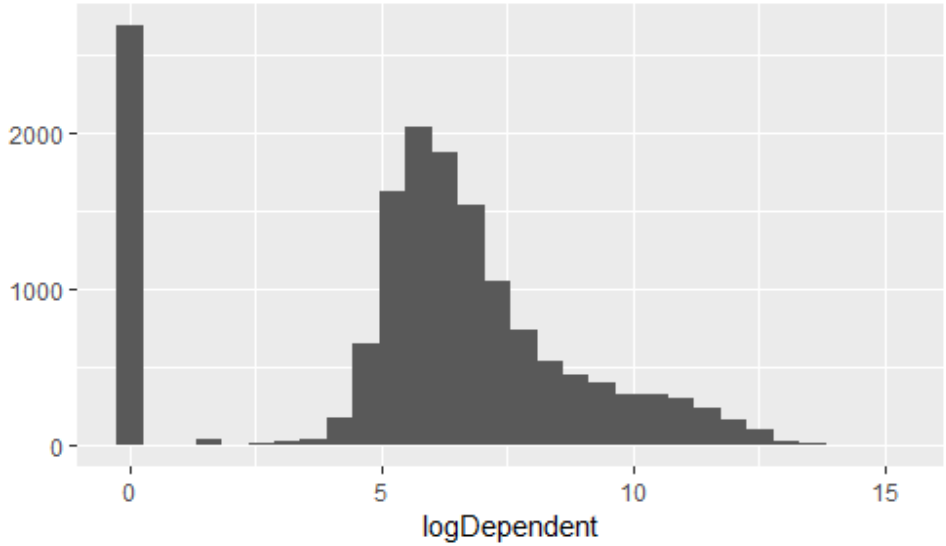
MUTATE()

- Used to perform variable transformations
- Most Predictive Modeling projects need a variety of transformations:
- For instance, we may want to log highly skewed variables



MUTATE DEPENDENT VARIABLE

```
wcdata4<-mutate(wcdata4,logDependent=log(Dependent+1))
qplot(logDependent,data=wcdata4)
```



OTHER TRANSFORMATIONS

- Indicator variables to Identify missing's in variables
- Compute time lag variables (Report lag to employer, report lag to carrier)

GROUP_BY(), SUMMARIZE()

- Can be used to group data by a variable or variables and then summarize
- Lets compute some state level statistics using claimant state:
  - Get size in database by counting the number of records for each state
  - Use result to reduce cardinality by grouping all low count states into one "small" category

```
byState=group_by(wcdata4,Claimant.State)

StateSize=summarize(byState,count=n())

StateSize=mutate(StateSize,rankState=min_rank(desc(count)))

head(StateSize)
```

```
head(StateSize)
# A tibble: 6 x 3
  Claimant.State           count rankState
  <chr>                    <int>     <int>
1 Alabama                     63        24
2 Arizona                     65        22
3 California                8912         1
4 Canada - British Columbia     4        44
5 Colorado                    65        22
6 Connecticut                 71        21
```

USING THE PIPE OPERATOR

- A different way of organizing code
- The logical sequence of the code is more natural
- A way of executing steps or nested code/ functions
- The pipe operator is %>%

REDO GROUP_BY AND SUMMARIZE WITH PIPE OPERATOR

```
StateSize<- wcdata3 %>% group_by(Claimant.State.Code) %>%
  summarize(count=n()) %>%
  mutate(r=min_rank(desc(count)))
```

- The pipe operator is denoted with "%>%"
- It moves from left to right
- The database comes first
- Then the procedures performed on it in chronological order

JOINING DATA

Used to merge datasets

Certain joins are like a VLOOKUP in Excel

inner_join()

left_join()

right_join()

full_join()

We can use a join to join the StateSize data into the data file

JOINING THE GROUPED DATA BACK INTO ORIGINAL DATA

```
wcdata5
=inner_join(wcdata4,StateSize,by="C
laimant.State.Code")

names(wcdata5)

Or

wcdata5
=right_join(wcdata4,StateSize,by="C
laimant.State.Code")

names(wcdata5)
```

```
> names(wcdata5)
 [1] "Obs_ID"
 [2] "Dependent"
 [3] "Average.Weekly.Wage"
 [4] "Body.Part.Code"
 [5] "Cause.Code"
 [6] "Claimant.Age"
 [7] "Claimant.Atty.Firm.Name"
 [8] "Claimant.Gender"
 [9] "Claimant.Gender.Code"
[10] "Claimant.Marital.Status"
[11] "Claimant.Marital.Status.Code"
[12] "Claimant.State.Code"
[13] "Department.Code"
[14] "Detail.Cause.Code"
[15] "Domestic.vs..Foreign."
[16] "Domestic.vs..Foreign..Code"
[17] "Dt.Reported.to.Carrier.TPA"
[18] "Dt.Reported.to.Employer"
[19] "Employment.Status"
[20] "Employment.Status.Code"
[21] "Date.of.Injury.Illness"
[22] "Handling.Office.Name"
[23] "How.Injury.Occurred"
[24] "Injury.Illness.City"
[25] "Injury.Illness.Postal"
[26] "Injury.Illness.State.Code"
[27] "Jurisdiction.Code"
[28] "Lost.Time.or.Medical.Only."
[29] "Lost.Time.or.Medical.Only..Code"
[30] "Nature.of.Injury.Illness.Code"
[31] "Number.of.Dependents"
[32] "OSHA.Injury.Type.Code"
[33] "Severity.Index.Code.Code"
[34] "Time.of.Injury.Illness"
[35] "Type.of.Loss.Code"
[36] "Policy.Year"
[37] "Reforms_dummy"
[38] "logDependent"
[39] "WageMissing"
[40] "count"
[41] "rankState"
```