R Notebook

Code ▾

Chapter 4 The tidyverse Up to now we have been manipulating vectors by reordering and subsetting them through indexing. However, once we start more advanced analyses, the preferred unit for data storage is not the vector but the data frame. In this chapter we learn to work directly with data frames, which greatly facilitate the organization of information. We will be using data frames for the majority of this book. We will focus on a specific data format referred to as tidy and on specific collection of packages that are particularly helpful for working with tidy data referred to as the tidyverse.

We can load all the tidyverse packages at once by installing and loading the tidyverse package:

Hide

```
library(tidyverse)
```

```
Registered S3 methods overwritten by 'dbplyr':
  method         from
  print.tbl_lazy
  print.tbl_sql
-- Attaching packages ----------
√  ggplot2 3.3.5     √  purrr   0.3.4
√  tibble  3.1.4     √  dplyr   1.0.7
√  tidyr   1.1.3     √  stringr 1.4.0
√  readr   2.0.1     √  forcats 0.5.1
-- Conflicts -------------------
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

We will learn how to implement the tidyverse approach throughout the book, but before delving into the details, in this chapter we introduce some of the most widely used tidyverse functionality, starting with the dplyr package for manipulating data frames and the purrr package for working with functions. Note that the tidyverse also includes a graphing package, ggplot2, which we introduce later in Chapter 7 in the Data Visualization part of the book; the readr package discussed in Chapter 5; and many others. In this chapter, we first introduce the concept of tidy data and then demonstrate how we use the tidyverse to work with data frames in this format.

#4.1 Tidy data We say that a data table is in tidy format if each row represents one observation and columns represent the different variables available for each of these observations. The murders dataset is an example of a tidy data frame.

Hide

```
#>        state abb region population total
#> 1    Alabama  AL  South    4779736   135
#> 2     Alaska  AK   West     710231    19
#> 3    Arizona  AZ   West    6392017   232
#> 4   Arkansas  AR  South    2915918    93
#> 5 California  CA   West   37253956  1257
#> 6   Colorado  CO   West    5029196    65
```

Each row represent a state with each of the five columns providing a different variable related to these states: name, abbreviation, region, population, and total murders.

To see how the same information can be provided in different formats, consider the following example:

Hide

```
#>       country year fertility
#> 1     Germany 1960      2.41
#> 2 South Korea 1960      6.16
#> 3     Germany 1961      2.44
#> 4 South Korea 1961      5.99
#> 5     Germany 1962      2.47
#> 6 South Korea 1962      5.79
```

This tidy dataset provides fertility rates for two countries across the years. This is a tidy dataset because each row presents one observation with the three variables being country, year, and fertility rate. However, this dataset originally came in another format and was reshaped for the dslabs package. Originally, the data was in the following format:

Hide

```
#>       country 1960 1961 1962
#> 1     Germany 2.41 2.44 2.47
#> 2 South Korea 6.16 5.99 5.79
```

The same information is provided, but there are two important differences in the format: 1) each row includes several observations and 2) one of the variables, year, is stored in the header. For the tidyverse packages to be optimally used, data need to be reshaped into tidy format, which you will learn to do in the Data Wrangling part of the book. Until then, we will use example datasets that are already in tidy format.

Although not immediately obvious, as you go through the book you will start to appreciate the advantages of working in a framework in which functions use tidy formats for both inputs and outputs. You will see how this permits the data analyst to focus on more important aspects of the analysis rather than the format of the data.

##4.2 Exercises 1. Examine the built-in dataset co2. Which of the following is true:

a.co2 is tidy data: it has one year for each row. b.co2 is not tidy: we need at least one column with a character vector. c.co2 is not tidy: it is a matrix instead of a data frame. d.co2 is not tidy: to be tidy we would have to wrangle it to have three columns (year, month and value), then each co2 observation would have a row.

Hide

```
head(co2)
```

```
[1] 315.42 316.31 316.50 317.56
[5] 318.13 318.00
```

Hide

```
co2
```

```
        Jan    Feb    Mar
1959 315.42 316.31 316.50
1960 316.27 316.81 317.42
1961 316.73 317.54 318.38
1962 317.78 318.40 319.53
1963 318.58 318.92 319.70
1964 319.41 320.07 320.74
1965 319.27 320.28 320.73
1966 320.46 321.43 322.23
1967 322.17 322.34 322.88
1968 322.40 322.99 323.73
1969 323.83 324.26 325.47
1970 324.89 325.82 326.77
1971 326.01 326.51 327.01
1972 326.60 327.47 327.58
1973 328.37 329.40 330.14
1974 329.18 330.55 331.32
1975 330.23 331.25 331.87
1976 331.58 332.39 333.33
1977 332.75 333.24 334.53
1978 334.80 335.22 336.47
1979 336.05 336.59 337.79
1980 337.84 338.19 339.91
1981 339.06 340.30 341.21
1982 340.57 341.44 342.53
1983 341.20 342.35 342.93
1984 343.52 344.33 345.11
1985 344.79 345.82 347.25
1986 346.11 346.78 347.68
1987 347.84 348.29 349.23
1988 350.25 351.54 352.05
1989 352.60 352.92 353.53
1990 353.50 354.55 355.23
1991 354.59 355.63 357.03
1992 355.88 356.63 357.72
1993 356.63 357.10 358.32
1994 358.34 358.89 359.95
1995 359.98 361.03 361.66
1996 362.09 363.29 364.06
1997 363.23 364.06 364.61
        Apr    May    Jun
1959 317.56 318.13 318.00
1960 318.87 319.87 319.43
1961 319.31 320.42 319.61
1962 320.42 320.85 320.45
1963 321.22 322.08 321.31
1964 321.40 322.06 321.73
1965 321.97 322.00 321.71
1966 323.54 323.91 323.59
1967 324.25 324.83 323.93
1968 324.86 325.40 325.20
1969 326.50 327.21 326.54
1970 327.97 327.91 327.50
1971 327.62 328.76 328.40
1972 329.56 329.90 328.92
1973 331.33 332.31 331.90
1974 332.48 332.92 332.08
```

```
1975 333.14 333.80 333.43
1976 334.41 334.71 334.17
1977 335.90 336.57 336.10
1978 337.59 337.84 337.72
1979 338.71 339.30 339.12
1980 340.60 341.29 341.00
1981 342.33 342.74 342.08
1982 343.39 343.96 343.18
1983 344.77 345.58 345.14
1984 346.88 347.25 346.62
1985 348.17 348.74 348.07
1986 349.37 350.03 349.37
1987 350.80 351.66 351.07
1988 353.41 354.04 353.62
1989 355.26 355.52 354.97
1990 356.04 357.00 356.07
1991 358.48 359.22 358.12
1992 359.07 359.58 359.17
1993 359.41 360.23 359.55
1994 361.25 361.67 360.94
1995 363.48 363.82 363.30
1996 364.76 365.45 365.01
1997 366.40 366.84 365.68
        Jul    Aug    Sep
1959 316.39 314.65 313.68
1960 318.01 315.74 314.00
1961 318.42 316.63 314.83
1962 319.45 317.25 316.11
1963 319.58 317.61 316.05
1964 320.27 318.54 316.54
1965 321.05 318.71 317.66
1966 322.24 320.20 318.48
1967 322.38 320.76 319.10
1968 323.98 321.95 320.18
1969 325.72 323.50 322.22
1970 326.18 324.53 322.93
1971 327.20 325.27 323.20
1972 327.88 326.16 324.68
1973 330.70 329.15 327.35
1974 331.01 329.23 327.27
1975 331.73 329.90 328.40
1976 332.89 330.77 329.14
1977 334.76 332.59 331.42
1978 336.37 334.51 332.60
1979 337.56 335.92 333.75
1980 339.39 337.43 335.72
1981 340.32 338.26 336.52
1982 341.88 339.65 337.81
1983 343.81 342.21 339.69
1984 345.22 343.11 340.90
1985 346.38 344.51 342.92
1986 347.76 345.73 344.68
1987 349.33 347.92 346.27
1988 352.22 350.27 348.55
1989 353.75 351.52 349.64
1990 354.67 352.76 350.82
1991 356.06 353.92 352.05
1992 356.94 354.92 352.94
```

```
1993 357.53 355.48 353.67
1994 359.55 357.49 355.84
1995 361.94 359.50 358.11
1996 363.70 361.54 359.51
1997 364.52 362.57 360.24
        Oct    Nov    Dec
1959 313.18 314.66 315.43
1960 313.68 314.84 316.03
1961 315.16 315.94 316.85
1962 315.27 316.53 317.53
1963 315.83 316.91 318.20
1964 316.71 317.53 318.55
1965 317.14 318.70 319.25
1966 317.94 319.63 320.87
1967 319.24 320.56 321.80
1968 320.09 321.16 322.74
1969 321.62 322.69 323.95
1970 322.90 323.85 324.96
1971 323.40 324.63 325.85
1972 325.04 326.34 327.39
1973 327.02 327.99 328.48
1974 327.21 328.29 329.41
1975 328.17 329.32 330.59
1976 328.78 330.14 331.52
1977 330.98 332.24 333.68
1978 332.38 333.75 334.78
1979 333.70 335.12 336.56
1980 335.84 336.93 338.04
1981 336.68 338.19 339.44
1982 337.69 339.09 340.32
1983 339.82 340.98 342.82
1984 341.18 342.80 344.04
1985 342.62 344.06 345.38
1986 343.99 345.48 346.72
1987 346.18 347.64 348.78
1988 348.72 349.91 351.18
1989 349.83 351.14 352.37
1990 351.04 352.69 354.07
1991 352.11 353.64 354.89
1992 353.23 354.09 355.33
1993 353.95 355.30 356.78
1994 356.00 357.59 359.05
1995 357.80 359.61 360.74
1996 359.65 360.80 362.38
1997 360.83 362.49 364.34
```

Hide

```
**d
```

```
Error: unexpected '^' in "**"
```

2. Examine the built-in dataset ChickWeight. Which of the following is true: a.ChickWeight is not tidy: each chick has more than one row. b.ChickWeight is tidy: each observation (a weight) is represented by one row. The chick from which this measurement came is one of the variables. c.ChickWeight is not tidy: we are missing the year column. d.ChickWeight is tidy: it is stored in a data frame

Hide

```
ChickWeight
```

	weight <dbl>	Time <dbl>	Chick <ord>	Diet <fctr>
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1
7	106	12	1	1
8	125	14	1	1
9	149	16	1	1
10	171	18	1	1

1-10 of 578 rows Previous **1** 2 3 4 5 6 ... 58 Next

Hide

```
**b
```

```
Error: unexpected '^' in "**"
```

3. Examine the built-in dataset BOD. Which of the following is true:

a.BOD is not tidy: it only has six rows. b.BOD is not tidy: the first column is just an index. c.BOD is tidy: each row is an observation with two values (time and demand) d.BOD is tidy: all small datasets are tidy by definition.

Hide

```
BOD
```

Time <dbl>	demand <dbl>
1	8.3
2	10.3
3	19.0
4	16.0
5	15.6
7	19.8

6 rows

Hide

```
**c
```

```
Error: unexpected '^' in "**"
```

4. Which of the following built-in datasets is tidy (you can pick more than one):

a.BJsales b.EuStockMarkets c.DNase d.Formaldehyde e.Orange f.UCBAdmissions

Hide

```
BJsales
```

```
Time Series:
Start = 1
End = 150
Frequency = 1
  [1] 200.1 199.5 199.4 198.9
  [5] 199.0 200.2 198.6 200.0
  [9] 200.3 201.2 201.6 201.5
 [13] 201.5 203.5 204.9 207.1
 [17] 210.5 210.5 209.8 208.8
 [21] 209.5 213.2 213.7 215.1
 [25] 218.7 219.8 220.5 223.8
 [29] 222.8 223.8 221.7 222.3
 [33] 220.8 219.4 220.1 220.6
 [37] 218.9 217.8 217.7 215.0
 [41] 215.3 215.9 216.7 216.7
 [45] 217.7 218.7 222.9 224.9
 [49] 222.2 220.7 220.0 218.7
 [53] 217.0 215.9 215.8 214.1
 [57] 212.3 213.9 214.6 213.6
 [61] 212.1 211.4 213.1 212.9
 [65] 213.3 211.5 212.3 213.0
 [69] 211.0 210.7 210.1 211.4
 [73] 210.0 209.7 208.8 208.8
 [77] 208.8 210.6 211.9 212.8
 [81] 212.5 214.8 215.3 217.5
 [85] 218.8 220.7 222.2 226.7
 [89] 228.4 233.2 235.7 237.1
 [93] 240.6 243.8 245.3 246.0
 [97] 246.3 247.7 247.6 247.8
[101] 249.4 249.0 249.9 250.5
[105] 251.5 249.0 247.6 248.8
[109] 250.4 250.7 253.0 253.7
[113] 255.0 256.2 256.0 257.4
[117] 260.4 260.0 261.3 260.4
[121] 261.6 260.8 259.8 259.0
[125] 258.9 257.4 257.7 257.9
[129] 257.4 257.3 257.6 258.9
[133] 257.8 257.7 257.2 257.5
[137] 256.8 257.5 257.0 257.6
[141] 257.3 257.5 259.6 261.1
[145] 262.9 263.3 262.8 261.8
[149] 262.2 262.7
```

Hide

```
EuStockMarkets
```

```
Time Series:
Start = c(1991, 130)
End = c(1998, 169)
Frequency = 260
             DAX    SMI    CAC
1991.496 1628.75 1678.1 1772.8
1991.500 1613.63 1688.5 1750.5
1991.504 1606.51 1678.6 1718.0
1991.508 1621.04 1684.1 1708.1
1991.512 1618.16 1686.6 1723.1
1991.515 1610.61 1671.6 1714.3
1991.519 1630.75 1682.9 1734.5
1991.523 1640.17 1703.6 1757.4
1991.527 1635.47 1697.5 1754.0
1991.531 1645.89 1716.3 1754.3
1991.535 1647.84 1723.8 1759.8
1991.538 1638.35 1730.5 1755.5
1991.542 1629.93 1727.4 1758.1
1991.546 1621.49 1733.3 1757.5
1991.550 1624.74 1734.0 1763.5
1991.554 1627.63 1728.3 1762.8
1991.558 1631.99 1737.1 1768.9
1991.562 1621.18 1723.1 1778.1
1991.565 1613.42 1723.6 1780.1
1991.569 1604.95 1719.0 1767.7
1991.573 1605.75 1721.2 1757.9
1991.577 1616.67 1725.3 1756.6
1991.581 1619.29 1727.2 1754.7
1991.585 1620.49 1727.2 1766.8
1991.588 1619.67 1731.6 1766.5
1991.592 1623.07 1724.1 1762.2
1991.596 1613.98 1716.9 1759.5
1991.600 1631.87 1723.4 1782.4
1991.604 1630.37 1723.0 1789.5
1991.608 1633.47 1728.4 1783.5
1991.612 1626.55 1722.1 1780.4
1991.615 1650.43 1724.5 1808.8
1991.619 1650.06 1733.6 1820.3
1991.623 1654.11 1739.0 1820.3
1991.627 1653.60 1726.2 1820.3
1991.631 1501.82 1587.4 1687.5
1991.635 1524.28 1630.6 1725.6
1991.638 1603.65 1685.5 1792.9
1991.642 1622.49 1701.3 1819.1
1991.646 1636.68 1718.0 1833.5
1991.650 1652.10 1726.2 1853.4
1991.654 1645.81 1716.6 1849.7
1991.658 1650.36 1725.8 1851.8
1991.662 1651.55 1737.4 1857.7
1991.665 1649.88 1736.6 1864.3
1991.669 1653.52 1732.4 1863.5
1991.673 1657.51 1731.2 1873.2
1991.677 1649.55 1726.9 1860.8
1991.681 1649.09 1727.8 1868.7
1991.685 1646.41 1720.2 1860.4
1991.688 1638.65 1715.4 1855.9
1991.692 1625.80 1708.7 1840.5
```

```
1991.696 1628.64 1713.0 1842.6
1991.700 1632.22 1713.5 1861.2
1991.704 1633.65 1718.0 1876.2
1991.708 1631.17 1701.7 1878.3
1991.712 1635.80 1701.7 1878.4
1991.715 1621.27 1684.9 1869.4
1991.719 1624.70 1687.2 1880.4
1991.723 1616.13 1690.6 1885.5
1991.727 1618.12 1684.3 1888.4
1991.731 1627.80 1679.9 1885.2
1991.735 1625.79 1672.9 1877.9
1991.738 1614.80 1663.1 1876.5
1991.742 1612.80 1669.3 1883.8
1991.746 1605.47 1664.7 1880.6
1991.750 1609.32 1672.3 1887.4
1991.754 1607.48 1687.7 1878.3
1991.758 1607.48 1686.8 1867.1
1991.762 1604.89 1686.6 1851.9
1991.765 1589.12 1675.8 1843.6
1991.769 1582.27 1677.4 1848.1
1991.773 1567.99 1673.2 1843.4
1991.777 1568.16 1665.0 1843.6
1991.781 1569.71 1671.3 1833.8
1991.785 1571.74 1672.4 1833.4
1991.788 1585.41 1676.2 1856.9
1991.792 1570.01 1692.6 1863.4
1991.796 1561.89 1696.5 1855.5
1991.800 1565.18 1716.1 1864.2
1991.804 1570.34 1713.3 1846.0
1991.808 1577.00 1705.1 1836.8
1991.812 1590.29 1711.3 1830.4
1991.815 1572.72 1709.8 1831.6
1991.819 1572.07 1688.6 1834.8
1991.823 1579.19 1698.9 1852.1
1991.827 1588.73 1700.0 1849.8
1991.831 1586.01 1693.0 1861.8
1991.835 1579.77 1683.9 1856.7
1991.838 1572.58 1679.2 1856.7
1991.842 1568.09 1673.9 1841.5
1991.846 1578.21 1683.9 1846.9
1991.850 1573.94 1688.4 1836.1
1991.854 1582.06 1693.9 1838.6
1991.858 1610.18 1720.9 1857.6
1991.862 1605.16 1717.9 1857.6
1991.865 1623.84 1733.6 1858.4
1991.869 1615.26 1729.7 1846.8
1991.873 1627.08 1735.6 1868.5
1991.877 1626.97 1734.1 1863.2
1991.881 1605.70 1699.3 1808.3
1991.885 1589.70 1678.6 1765.1
1991.888 1589.70 1675.5 1763.5
1991.892 1603.26 1670.1 1766.0
1991.896 1599.75 1652.2 1741.3
1991.900 1590.86 1635.0 1743.3
1991.904 1603.50 1654.9 1769.0
1991.908 1589.86 1642.0 1757.9
1991.912 1587.92 1638.7 1754.9
1991.915 1571.06 1622.6 1739.7
```

```
1991.919 1549.81 1596.1 1708.8
1991.923 1549.36 1612.4 1722.2
1991.927 1554.65 1625.0 1713.9
1991.931 1557.52 1610.5 1703.2
1991.935 1555.31 1606.6 1685.7
1991.938 1559.76 1610.7 1663.4
1991.942 1548.44 1603.1 1636.9
1991.946 1543.99 1591.5 1645.6
1991.950 1550.21 1605.2 1671.6
1991.954 1557.03 1621.4 1688.3
1991.958 1551.78 1622.5 1696.8
1991.962 1562.89 1626.6 1711.7
1991.965 1570.28 1627.4 1706.2
1991.969 1559.26 1614.9 1684.2
1991.973 1545.87 1602.3 1648.5
1991.977 1542.77 1598.3 1633.6
1991.981 1542.77 1627.0 1699.1
1991.985 1542.77 1627.0 1699.1
1991.988 1542.77 1627.0 1722.5
1991.992 1564.27 1655.7 1720.7
1991.996 1577.26 1670.1 1741.9
1992.000 1577.26 1670.1 1765.7
1992.004 1577.26 1670.1 1765.7
1992.008 1598.19 1670.1 1749.9
1992.012 1604.05 1704.0 1770.3
1992.015 1604.69 1711.8 1787.6
1992.019 1593.65 1700.5 1778.7
1992.023 1581.68 1690.3 1785.6
1992.027 1599.14 1715.4 1833.9
1992.031 1613.82 1723.5 1837.4
1992.035 1620.45 1719.4 1824.3
1992.038 1629.51 1734.4 1843.8
1992.042 1663.70 1772.8 1873.6
1992.046 1664.09 1760.3 1860.2
1992.050 1669.29 1747.2 1860.2
1992.054 1685.14 1750.2 1865.9
1992.058 1687.07 1755.3 1867.9
1992.062 1680.13 1754.6 1841.3
1992.065 1671.84 1751.2 1838.7
1992.069 1669.52 1752.5 1849.9
1992.073 1686.71 1769.4 1869.3
1992.077 1685.51 1767.6 1890.6
1992.081 1671.01 1750.0 1879.6
1992.085 1683.06 1747.1 1873.9
1992.088 1685.70 1753.5 1875.3
1992.092 1685.66 1752.8 1857.0
1992.096 1678.77 1752.9 1856.5
1992.100 1685.85 1764.7 1865.8
1992.104 1683.71 1776.8 1860.6
1992.108 1686.59 1779.3 1861.6
1992.112 1683.73 1785.1 1865.6
1992.115 1679.14 1798.2 1864.1
1992.119 1685.03 1794.1 1861.6
1992.123 1680.81 1795.2 1876.5
1992.127 1676.17 1780.4 1865.1
1992.131 1688.46 1789.5 1882.1
1992.135 1696.55 1794.2 1912.2
1992.138 1690.24 1784.4 1915.4
```

```
1992.142 1711.35 1800.1 1951.2
1992.146 1711.29 1804.0 1962.4
1992.150 1729.86 1816.2 1976.5
1992.154 1716.63 1810.5 1953.5
1992.158 1743.36 1821.9 1981.3
1992.162 1745.17 1828.2 1985.1
1992.165 1746.76 1840.6 1983.4
1992.169 1749.29 1841.1 1979.7
1992.173 1763.86 1846.3 1983.8
1992.177 1762.27 1850.0 1988.1
1992.181 1762.29 1839.0 1973.0
1992.185 1746.77 1820.2 1966.9
1992.188 1753.50 1815.2 1976.3
1992.192 1753.21 1820.6 1993.9
1992.196 1739.88 1807.1 1968.0
1992.200 1723.92 1791.4 1941.8
1992.204 1734.42 1806.2 1947.1
1992.208 1723.13 1798.7 1929.2
1992.212 1732.92 1818.2 1943.6
1992.215 1729.89 1820.5 1928.2
1992.219 1725.74 1833.3 1922.0
1992.223 1730.90 1837.1 1919.1
1992.227 1714.17 1818.2 1884.6
1992.231 1716.20 1824.1 1896.3
1992.235 1719.06 1830.1 1928.3
1992.238 1718.21 1835.6 1934.8
1992.242 1698.84 1828.7 1923.5
1992.246 1714.76 1839.2 1943.8
1992.250 1718.35 1837.2 1942.4
1992.254 1706.69 1826.7 1928.1
1992.258 1723.37 1838.0 1942.0
1992.262 1716.18 1829.1 1942.7
1992.265 1738.78 1843.1 1974.8
1992.269 1737.41 1850.5 1975.4
1992.273 1714.77 1827.1 1907.5
1992.277 1724.24 1829.1 1943.6
1992.281 1733.77 1848.0 1974.1
1992.285 1729.96 1840.5 1963.3
1992.288 1734.46 1853.8 1972.3
1992.292 1744.35 1874.1 1990.7
1992.296 1746.88 1871.3 1978.2
1992.300 1746.88 1871.3 1978.2
1992.304 1746.88 1871.3 1978.2
1992.308 1747.47 1860.5 1980.4
1992.312 1753.10 1874.7 1983.7
1992.315 1745.17 1880.1 1978.1
1992.319 1745.72 1874.7 1984.9
1992.323 1742.92 1875.6 1995.7
1992.327 1731.68 1859.5 2006.6
1992.331 1731.18 1874.2 2036.7
1992.335 1728.09 1880.1 2031.1
1992.338 1728.09 1880.1 2031.1
1992.342 1731.29 1907.7 2041.6
1992.346 1733.82 1920.5 2046.9
1992.350 1745.78 1937.3 2047.2
1992.354 1752.57 1936.8 2063.4
1992.358 1748.13 1949.1 2063.4
1992.362 1750.70 1963.7 2077.5
```

```
1992.365 1747.91 1950.8 2063.6
1992.369 1745.79 1953.5 2053.2
1992.373 1735.34 1945.0 2017.0
1992.377 1719.92 1921.1 2024.0
1992.381 1763.59 1939.1 2051.6
1992.385 1766.76 1928.0 2023.1
1992.388 1785.40 1933.4 2030.8
1992.392 1783.56 1925.7 2016.8
1992.396 1804.42 1931.7 2045.1
1992.400 1812.33 1928.7 2046.3
1992.404 1799.51 1924.5 2029.6
1992.408 1792.80 1914.2 2014.1
1992.412 1792.80 1914.2 2014.1
1992.415 1806.36 1920.6 2033.3
1992.419 1798.23 1923.3 2017.4
1992.423 1800.62 1930.4 2024.9
1992.427 1786.19 1915.2 1992.6
1992.431 1791.35 1916.9 1994.9
1992.435 1789.05 1913.8 1981.6
1992.438 1789.05 1913.8 1981.6
1992.442 1784.71 1899.7 1962.2
1992.446 1789.45 1888.0 1953.7
1992.450 1779.74 1868.8 1928.8
1992.454 1786.97 1879.9 1928.3
             FTSE
1991.496 2443.6
1991.500 2460.2
1991.504 2448.2
1991.508 2470.4
1991.512 2484.7
1991.515 2466.8
1991.519 2487.9
1991.523 2508.4
1991.527 2510.5
1991.531 2497.4
1991.535 2532.5
1991.538 2556.8
1991.542 2561.0
1991.546 2547.3
1991.550 2541.5
1991.554 2558.5
1991.558 2587.9
1991.562 2580.5
1991.565 2579.6
1991.569 2589.3
1991.573 2595.0
1991.577 2595.6
1991.581 2588.8
1991.585 2591.7
1991.588 2601.7
1991.592 2585.4
1991.596 2573.3
1991.600 2597.4
1991.604 2600.6
1991.608 2570.6
1991.612 2569.4
1991.615 2584.9
1991.619 2608.8
```

```
1991.623 2617.2
1991.627 2621.0
1991.631 2540.5
1991.635 2554.5
1991.638 2601.9
1991.642 2623.0
1991.646 2640.7
1991.650 2640.7
1991.654 2619.8
1991.658 2624.2
1991.662 2638.2
1991.665 2645.7
1991.669 2679.6
1991.673 2669.0
1991.677 2664.6
1991.681 2663.3
1991.685 2667.4
1991.688 2653.2
1991.692 2630.8
1991.696 2626.6
1991.700 2641.9
1991.704 2625.8
1991.708 2606.0
1991.712 2594.4
1991.715 2583.6
1991.719 2588.7
1991.723 2600.3
1991.727 2579.5
1991.731 2576.6
1991.735 2597.8
1991.738 2595.6
1991.742 2599.0
1991.746 2621.7
1991.750 2645.6
1991.754 2644.2
1991.758 2625.6
1991.762 2624.6
1991.765 2596.2
1991.769 2599.5
1991.773 2584.1
1991.777 2570.8
1991.781 2555.0
1991.785 2574.5
1991.788 2576.7
1991.792 2579.0
1991.796 2588.7
1991.800 2601.1
1991.804 2575.7
1991.808 2559.5
1991.812 2561.1
1991.815 2528.3
1991.819 2514.7
1991.823 2558.5
1991.827 2553.3
1991.831 2577.1
1991.835 2566.0
1991.838 2549.5
1991.842 2527.8
```

```
1991.846 2540.9
1991.850 2534.2
1991.854 2538.0
1991.858 2559.0
1991.862 2554.9
1991.865 2575.5
1991.869 2546.5
1991.873 2561.6
1991.877 2546.6
1991.881 2502.9
1991.885 2463.1
1991.888 2472.6
1991.892 2463.5
1991.896 2446.3
1991.900 2456.2
1991.904 2471.5
1991.908 2447.5
1991.912 2428.6
1991.915 2420.2
1991.919 2414.9
1991.923 2420.2
1991.927 2423.8
1991.931 2407.0
1991.935 2388.7
1991.938 2409.6
1991.942 2392.0
1991.946 2380.2
1991.950 2423.3
1991.954 2451.6
1991.958 2440.8
1991.962 2432.9
1991.965 2413.6
1991.969 2391.6
1991.973 2358.1
1991.977 2345.4
1991.981 2384.4
1991.985 2384.4
1991.988 2384.4
1991.992 2418.7
1991.996 2420.0
1992.000 2493.1
1992.004 2493.1
1992.008 2492.8
1992.012 2504.1
1992.015 2493.2
1992.019 2482.9
1992.023 2467.1
1992.027 2497.9
1992.031 2477.9
1992.035 2490.1
1992.038 2516.3
1992.042 2537.1
1992.046 2541.6
1992.050 2536.7
1992.054 2544.9
1992.058 2543.4
1992.062 2522.0
1992.065 2525.3
```

```
1992.069 2510.4
1992.073 2539.9
1992.077 2552.0
1992.081 2546.5
1992.085 2550.8
1992.088 2571.2
1992.092 2560.2
1992.096 2556.8
1992.100 2547.1
1992.104 2534.3
1992.108 2517.2
1992.112 2538.4
1992.115 2537.1
1992.119 2523.7
1992.123 2522.6
1992.127 2513.9
1992.131 2541.0
1992.135 2555.9
1992.138 2536.7
1992.142 2543.4
1992.146 2542.3
1992.150 2559.7
1992.154 2546.8
1992.158 2565.0
1992.162 2562.0
1992.165 2562.1
1992.169 2554.3
1992.173 2565.4
1992.177 2558.4
1992.181 2538.3
1992.185 2533.1
1992.188 2550.7
1992.192 2574.8
1992.196 2522.4
1992.200 2493.3
1992.204 2476.0
1992.208 2470.7
1992.212 2491.2
1992.215 2464.7
1992.219 2467.6
1992.223 2456.6
1992.227 2441.0
1992.231 2458.7
1992.235 2464.9
1992.238 2472.2
1992.242 2447.9
1992.246 2452.9
1992.250 2440.1
1992.254 2408.6
1992.258 2405.4
1992.262 2382.7
1992.265 2400.9
1992.269 2404.2
1992.273 2393.2
1992.277 2436.4
1992.281 2572.6
1992.285 2591.0
1992.288 2600.5
```

```
1992.292 2640.2
1992.296 2638.6
1992.300 2638.6
1992.304 2638.6
1992.308 2625.8
1992.312 2607.8
1992.315 2609.8
1992.319 2643.0
1992.323 2658.2
1992.327 2651.0
1992.331 2664.9
1992.335 2654.1
1992.338 2659.8
1992.342 2659.8
1992.346 2662.2
1992.350 2698.7
1992.354 2701.9
1992.358 2725.7
1992.362 2737.8
1992.365 2722.4
1992.369 2720.5
1992.373 2694.7
1992.377 2682.6
1992.381 2703.6
1992.385 2700.6
1992.388 2711.9
1992.392 2702.0
1992.396 2715.0
1992.400 2715.0
1992.404 2704.6
1992.408 2698.6
1992.412 2694.2
1992.415 2707.6
1992.419 2697.6
1992.423 2705.9
1992.427 2680.9
1992.431 2681.9
1992.435 2668.5
1992.438 2645.8
1992.442 2635.4
1992.446 2636.1
1992.450 2614.1
1992.454 2603.7
 [ reached getOption("max.print") -- omitted 1610 rows ]
```

Hide

```
DNase
```

	Run <ord>	conc <dbl>	density <dbl>
1	1	0.04882812	0.017
2	1	0.04882812	0.018
3	1	0.19531250	0.121

	Run <ord>	conc <dbl>	density <dbl>
4	1	0.19531250	0.124
5	1	0.39062500	0.206
6	1	0.39062500	0.215
7	1	0.78125000	0.377
8	1	0.78125000	0.374
9	1	1.56250000	0.614
10	1	1.56250000	0.609

1-10 of 176 rows Previous **1** 2 3 4 5 6 ... 18 Next

Hide

```
Formaldehyde
```

	carb <dbl>	optden <dbl>
1	0.1	0.086
2	0.3	0.269
3	0.5	0.446
4	0.6	0.538
5	0.7	0.626
6	0.9	0.782

6 rows

Hide

```
Orange
```

	Tree <ord>	age <dbl>	circumference <dbl>
1	1	118	30
2	1	484	58
3	1	664	87
4	1	1004	115
5	1	1231	120
6	1	1372	142
7	1	1582	145
8	2	118	33

	Tree <ord>	age <dbl>	circumference <dbl>
9	2	484	69
10	2	664	111

1-10 of 35 rows Previous **1** 2 3 4 Next

Hide

```
UCBAdmissions
```

```
, , Dept = A

          Gender
Admit      Male Female
  Admitted  512     89
  Rejected  313     19

, , Dept = B

          Gender
Admit      Male Female
  Admitted  353     17
  Rejected  207      8

, , Dept = C

          Gender
Admit      Male Female
  Admitted  120    202
  Rejected  205    391

, , Dept = D

          Gender
Admit      Male Female
  Admitted  138    131
  Rejected  279    244

, , Dept = E

          Gender
Admit      Male Female
  Admitted   53     94
  Rejected  138    299

, , Dept = F

          Gender
Admit      Male Female
  Admitted   22     24
  Rejected  351    317
```

Hide

```
**b,c,d,e
```

```
Error: unexpected '^' in "**"
```

###4.3 Manipulating data frames The dplyr package from the tidyverse introduces functions that perform some of the most common operations when working with data frames and uses names for these functions that are relatively easy to remember. For instance, to change the data table by adding a new column, we use mutate. To filter the data table to a subset of rows, we use filter. Finally, to subset the data by selecting specific columns, we use select.

4.3.1 Adding a column with mutate We want all the necessary information for our analysis to be included in the data table. So the first task is to add the murder rates to our murders data frame. The function mutate takes the data frame as a first argument and the name and values of the variable as a second argument using the convention name = values. So, to add murder rates, we use:

Hide

```
library(dslabs)
data("murders")
murders <- mutate(murders, rate = total / population * 100000)
```

Notice that here we used total and population inside the function, which are objects that are not defined in our workspace. But why don't we get an error?

This is one of dplyr's main features. Functions in this package, such as mutate, know to look for variables in the data frame provided in the first argument. In the call to mutate above, total will have the values in murders$total. This approach makes the code much more readable.

We can see that the new column is added:

Hide

```
head(murders)
```

	state <chr>	abb <chr>	region <fctr>	population <dbl>	total <dbl>
1	Alabama	AL	South	4779736	135
2	Alaska	AK	West	710231	19
3	Arizona	AZ	West	6392017	232
4	Arkansas	AR	South	2915918	93
5	California	CA	West	37253956	1257
6	Colorado	CO	West	5029196	65

6 rows

4.3.3 Selecting columns with select Although our data table only has six columns, some data tables include hundreds. If we want to view just a few, we can use the dplyr select function. In the code below we select three columns, assign this to a new object and then filter the new object:

Hide

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

state	region	rate
<chr>	<fctr>	<dbl>
Hawaii	West	0.5145920
Iowa	North Central	0.6893484
New Hampshire	Northeast	0.3798036
North Dakota	North Central	0.5947151
Vermont	Northeast	0.3196211

5 rows

In the call to select, the first argument murders is an object, but state, region, and rate are variable names.

####4.4 Exercises 1. Load the dplyr package and the murders dataset.

Hide

```
library(dplyr)
library(dslabs)
data(murders)
```

You can add columns using the dplyr function mutate. This function is aware of the column names and inside the function you can call them unquoted:

Hide

```
murders <- mutate(murders, population_in_millions = population / 10^6)
```

We can write population rather than murders$population. The function mutate knows we are grabbing columns from murders.

Use the function mutate to add a murders column named rate with the per 100,000 murder rate as in the example code above. Make sure you redefine murders as done in the example code above (murders <- [your code]) so we can keep using this variable.

Hide

```
murders <- mutate(murders, rate = total / population * 100000)
murders
```

state	…	region	population	total	population_in_millions
<chr>	<chr>	<fctr>	<dbl>	<dbl>	<dbl>
Alabama	AL	South	4779736	135	4.779736
Alaska	AK	West	710231	19	0.710231
Arizona	AZ	West	6392017	232	6.392017
Arkansas	AR	South	2915918	93	2.915918
California	CA	West	37253956	1257	37.253956

state	...	region	population	total	population_in_millions
<chr>	<chr>	<fctr>	<dbl>	<dbl>	<dbl>
Colorado	CO	West	5029196	65	5.029196
Connecticut	CT	Northeast	3574097	97	3.574097
Delaware	DE	South	897934	38	0.897934
District of Columbia	DC	South	601723	99	0.601723
Florida	FL	South	19687653	669	19.687653

1-10 of 51 rows Previous **1** 2 3 4 5 6 Next

2. If rank(x) gives you the ranks of x from lowest to highest, rank(-x) gives you the ranks from highest to lowest. Use the function mutate to add a column rank containing the rank, from highest to lowest murder rate. Make sure you redefine murders so we can keep using this variable.

Hide

```
murders <- mutate(murders, rank = rank(-rate))
murders
```

state	...	region	population	total	population_in_millions
<chr>	<chr>	<fctr>	<dbl>	<dbl>	<dbl>
Alabama	AL	South	4779736	135	4.779736
Alaska	AK	West	710231	19	0.710231
Arizona	AZ	West	6392017	232	6.392017
Arkansas	AR	South	2915918	93	2.915918
California	CA	West	37253956	1257	37.253956
Colorado	CO	West	5029196	65	5.029196
Connecticut	CT	Northeast	3574097	97	3.574097
Delaware	DE	South	897934	38	0.897934
District of Columbia	DC	South	601723	99	0.601723
Florida	FL	South	19687653	669	19.687653

1-10 of 51 rows Previous **1** 2 3 4 5 6 Next

3. With dplyr, we can use select to show only certain columns. For example, with this code we would only show the states and population sizes:

Hide

```
select(murders, state, population) %>% head()
```

state	population
<chr>	<dbl>

	state <chr>	population <dbl>
1	Alabama	4779736
2	Alaska	710231
3	Arizona	6392017
4	Arkansas	2915918
5	California	37253956
6	Colorado	5029196

6 rows

Use select to show the state names and abbreviations in murders. Do not redefine murders, just show the results.

Hide

```
select(murders, state,abb)
```

state <chr>	abb <chr>
Alabama	AL
Alaska	AK
Arizona	AZ
Arkansas	AR
California	CA
Colorado	CO
Connecticut	CT
Delaware	DE
District of Columbia	DC
Florida	FL

1-10 of 51 rows Previous **1** 2 3 4 5 6 Next

4. The dplyr function filter is used to choose specific rows of the data frame to keep. Unlike select which is for columns, filter is for rows. For example, you can show just the New York row like this:

Hide

```
filter(murders, state == "New York")
```

state <chr>	... <chr>	region <fctr>	population <dbl>	total <dbl>	population_in_millions <dbl>	rate <dbl>	rank <dbl>
New York	NY	Northeast	19378102	517	19.3781	2.66796	29

1 row

You can use other logical vectors to filter rows.

Use filter to show the top 5 states with the highest murder rates. After we add murder rate and rank, do not change the murders dataset, just show the result. Remember that you can filter based on the rank column.

Hide

```
filter(murders, rank <= 5)
```

state	...	region	population	total	population_in_millions
<chr>	<chr>	<fctr>	<dbl>	<dbl>	<dbl>
District of Columbia	DC	South	601723	99	0.601723
Louisiana	LA	South	4533372	351	4.533372
Maryland	MD	South	5773552	293	5.773552
Missouri	MO	North Central	5988927	321	5.988927
South Carolina	SC	South	4625364	207	4.625364

5 rows

5. We can remove rows using the != operator. For example, to remove Florida, we would do this:

Hide

```
no_florida <- filter(murders, state != "Florida")
```

Create a new data frame called no_south that removes states from the South region. How many states are in this category? You can use the function nrow for this.

Hide

```
no_south <- filter(murders, region != "South")
nrow(no_south)
```

```
[1] 34
```

6. We can also use %in% to filter with dplyr. You can therefore see the data from New York and Texas like this:

Hide

```
filter(murders, state %in% c("New York", "Texas"))
```

state	...	region	population	total	population_in_millions	rate	rank
<chr>	<chr>	<fctr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
New York	NY	Northeast	19378102	517	19.37810	2.66796	29
Texas	TX	South	25145561	805	25.14556	3.20136	16

2 rows

Create a new data frame called murders_nw with only the states from the Northeast and the West. How many states are in this category?

Hide

```
murders_nw <- filter(murders, region %in% c("Northeast", "West"))
nrow(murders_nw)
```

```
[1] 22
```

7. Suppose you want to live in the Northeast or West and want the murder rate to be less than 1. We want to see the data for the states satisfying these options. Note that you can use logical operators with filter. Here is an example in which we filter to keep only small states in the Northeast region.

Hide

```
filter(murders, population < 5000000 & region == "Northeast")
```

state <chr>	... <chr>	region <fctr>	population <dbl>	total <dbl>	population_in_millions <dbl>	rate <dbl>	r... <dbl>
Connecticut	CT	Northeast	3574097	97	3.574097	2.7139722	25
Maine	ME	Northeast	1328361	11	1.328361	0.8280881	44
New Hampshire	NH	Northeast	1316470	5	1.316470	0.3798036	50
Rhode Island	RI	Northeast	1052567	16	1.052567	1.5200933	35
Vermont	VT	Northeast	625741	2	0.625741	0.3196211	51

5 rows

Make sure murders has been defined with rate and rank and still has all states. Create a table called my_states that contains rows for states satisfying both the conditions: it is in the Northeast or West and the murder rate is less than 1. Use select to show only the state name, the rate, and the rank.

Hide

```
my_state <- filter(murders_nw, rate < 1)
select(my_state, state, rate, rank)
```

state <chr>	rate <dbl>	rank <dbl>
Hawaii	0.5145920	49
Idaho	0.7655102	46
Maine	0.8280881	44
New Hampshire	0.3798036	50
Oregon	0.9396843	42
Utah	0.7959810	45
Vermont	0.3196211	51

state	rate	rank
<chr>	<dbl>	<dbl>
Wyoming	0.8871131	43

8 rows

#####4.5 The pipe: %>% With dplyr we can perform a series of operations, for example select and then filter, by sending the results of one function to another using what is called the pipe operator: %>%. Some details are included below.

We wrote code above to show three variables (state, region, rate) for states that have murder rates below 0.71. To do this, we defined the intermediate object new_table. In dplyr we can write code that looks more like a description of what we want to do without intermediate objects:

original data → select → filter

For such an operation, we can use the pipe %>%. The code looks like this:

Hide

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
```

```
Error: Can't subset columns that don't exist.
x Column `rate` doesn't exist.
Run `rlang::last_error()` to see where the error occurred.
```

This line of code is equivalent to the two lines of code above. What is going on here?

In general, the pipe sends the result of the left side of the pipe to be the first argument of the function on the right side of the pipe. Here is a very simple example:

Hide

```
16 %>% sqrt()
```

```
[1] 4
```

We can continue to pipe values along:

Hide

```
16 %>% sqrt() %>% log2()
```

```
[1] 2
```

The above statement is equivalent to log2(sqrt(16)).

Remember that the pipe sends values to the first argument, so we can define other arguments as if the first argument is already defined:

Hide

```
16 %>% sqrt() %>% log(base = 2)
```

```
[1] 2
```

Therefore, when using the pipe with data frames and dplyr, we no longer need to specify the required first argument since the dplyr functions we have described all take the data as the first argument. In the code we wrote:

Hide

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
```

state <chr>	region <fctr>	rate <dbl>
Hawaii	West	0.5145920
Iowa	North Central	0.6893484
New Hampshire	Northeast	0.3798036
North Dakota	North Central	0.5947151
Vermont	Northeast	0.3196211

5 rows

murders is the first argument of the select function, and the new data frame (formerly new_table) is the first argument of the filter function.

Note that the pipe works well with functions where the first argument is the input data. Functions in tidyverse packages like dplyr have this format and can be used easily with the pipe.

######4.6 Exercises 1. The pipe %>% can be used to perform operations sequentially without having to define intermediate objects. Start by redefining murder to include rate and rank.

Hide

```
murders <- mutate(murders, rate =  total / population * 100000,
                  rank = rank(-rate))
```

In the solution to the previous exercise, we did the following:

Hide

```
my_states <- filter(murders, region %in% c("Northeast", "West") &
                      rate < 1)

select(my_states, state, rate, rank)
```

state <chr>	rate <dbl>	rank <dbl>
Hawaii	0.5145920	49
Idaho	0.7655102	46
Maine	0.8280881	44
New Hampshire	0.3798036	50
Oregon	0.9396843	42
Utah	0.7959810	45

state <chr>	rate <dbl>	rank <dbl>
Vermont	0.3196211	51
Wyoming	0.8871131	43

8 rows

The pipe %>% permits us to perform both operations sequentially without having to define an intermediate variable my_states. We therefore could have mutated and selected in the same line like this:

Hide

```
mutate(murders, rate =  total / population * 100000,
       rank = rank(-rate)) %>%
  select(state, rate, rank)
```

state <chr>	rate <dbl>	rank <dbl>
Alabama	2.8244238	23
Alaska	2.6751860	27
Arizona	3.6295273	10
Arkansas	3.1893901	17
California	3.3741383	14
Colorado	1.2924531	38
Connecticut	2.7139722	25
Delaware	4.2319369	6
District of Columbia	16.4527532	1
Florida	3.3980688	13

1-10 of 51 rows Previous **1** 2 3 4 5 6 Next

Notice that select no longer has a data frame as the first argument. The first argument is assumed to be the result of the operation conducted right before the %>%.

Repeat the previous exercise, but now instead of creating a new object, show the result and only include the state, rate, and rank columns. Use a pipe %>% to do this in just one line.

Hide

```
library(dplyr)
library(dslabs)
data(murders)
mutate(murders, rate =  total / population * 100000,
       rank = rank(-rate)) %>%
  filter(region %in% c("Northeast", "West") & rate < 1) %>%
  select(state, rate, rank)
```

state <chr>	rate <dbl>	rank <dbl>
Hawaii	0.5145920	49
Idaho	0.7655102	46
Maine	0.8280881	44
New Hampshire	0.3798036	50
Oregon	0.9396843	42
Utah	0.7959810	45
Vermont	0.3196211	51
Wyoming	0.8871131	43

8 rows

Hide

```
NA
```

2. Reset murders to the original table by using data(murders). Use a pipe to create a new data frame called my_states that considers only states in the Northeast or West which have a murder rate lower than 1, and contains only the state, rate and rank columns. The pipe should also have four components separated by three %>%. The code should look something like this:

Hide

```
library(dplyr)
library(dslabs)
data(murders)

my_states <- murders %>%
  mutate(rate = total / population * 100000, rank = rank(-rate)) %>%
  filter(region %in% c("Northeast", "West"), rate < 1) %>%
  select(state, rate, rank)

my_states
```

state <chr>	rate <dbl>	rank <dbl>
Hawaii	0.5145920	49
Idaho	0.7655102	46
Maine	0.8280881	44
New Hampshire	0.3798036	50
Oregon	0.9396843	42
Utah	0.7959810	45
Vermont	0.3196211	51
Wyoming	0.8871131	43

8 rows