Lesson 2: Importing and Exploring Data

AUTHOR
John Fariss

PUBLISHED
April 3, 2025

Introduction

Welcome to Lesson 2! In this lesson, you will: - Import real estate data - Clean and standardize column names to **BigCamelCase** - Explore the dataset - Understand how to handle different date formats

Step 1: Importing Data

We will load our dataset and ensure that dates are correctly formatted **on import**. We can do this with .csv files, but Excel (.xlsx) files will not behave correctly, so there is an "Extra" section below on handling dates from Excel files.

In this instance, we load an entire **ADF**. This is typically how we want to start in **Evidence Based Valuation**. We can later narrow to a **CMS**, but for simplicity, you may want to only import your CMS while getting accustomed to RStudio.

```
# Load required libraries
library(readr)

# Import the dataset and save as ADF (assignment data frame)
ADF0 <- read_csv("data/ADF.csv", col_types = cols(
  `Selling Date` = col_date(format = "%m/%d/%y"),
  `Pending Date` = col_date(format = "%m/%d/%y"),
  `Listing Date` = col_date(format = "%m/%d/%y")
))

# View the first few rows of data
head(ADF0)
```

Step 2: Standardizing Field Names

The dataset comes with column names that need to be cleaned and renamed to follow a **standard format (BigCamelCase)**. Why do we want to standardize names? Two reasons: transparency and reproducibility. When we always use the same names, its easier to follow and repeat the process.

```
# Load needed packages
library(janitor) # Use janitor for cleaning column names
library(dplyr) # Use dplyr for rename and mutate functions

# Clean data frame
```

```r
ADF1 <- ADF0 %>% # This is the pipe operator, it pipes one line of code into the next
  # Clean all variable names to PascalCase/DoubleCamel/BigCamel
  clean_names(case = "big_camel", # specify case to use
              abbreviations = c("APN", "DOM")) %>% # specify abbreviations to keep
  # Rename specified variables using the rename function, NewName = OldName
  # This requires the variable names after the clean_names function
  rename(Address = StreetNumberNameDirection,
         City = AddressCity,
         Zip = AddressZip,
         MLS = MlNumber,
         Status = Status,
         DateSale = SellingDate,
         DatePend = PendingDate,
         DateList = ListingDate,
         PriceSale = SellingPrice,
         GLA = SquareFootage,
         YearBuilt = YearBuilt,
         LotSF = LotSizeSqFt
         )

# View cleaned variable names
names(ADF1)
```

Notice that when we ran this function, we renamed the data frame from ADF0 to ADF1. The reason we do this is to keep track of our changes. Again, transparency and reproducibility.

Step 3: Handling Missing Values

This is a modeling decision that you as an appraiser must make. This is where your domain expertise intersects with data science. When values are missing, we must make modeling decisions on how to handle them. In this project, instead of removing all missing values, we will **only** remove missing values in the **GLA (gross living area) field**.

```r
# Remove rows where GLA is missing
ADF <- ADF1 %>% filter(!is.na(GLA))
```

We also saved the ADF1 as ADF which will be our final version used throughout the rest of our code.

Step 4: Cleaning the Data

Now, let's check the structure and summary of our dataset.

```r
# Check structure of dataset
str(ADF)

# Summary statistics of key fields
summary(ADF)
```

Extra: Handling Date Fields in Excel

When importing an Excel (.xlsx) instead of comma separate values (.csv), dates can get mixed up, even if we change them on import. Let's take a look at an example. First, we'll import an Excel file using the **readxl** package.

```
# Import an Excel file
ExcelData <- readxl::read_excel("Data/ExcelData.xlsx")
# This temporarily loads readxl and uses a function from the package

# Didn't work? You may need to install the readxl package
```

Checking for Incorrect Date Formats

If you suspect that Excel misformatted your dates, run:

```
# Check the data type of only our date fields using the str (structure function)
str(ExcelData[c("DateSale","DatePend","DateList")])
# OR, use the class function
class(ExcelData$DateSale)
# If the result is "POSIXct" instead of "Date", use lubridate to fix it.
```

Introducing lubridate

As you can see in the example above, when we import data from an **Excel file**, date fields may not be properly formatted. Often, Excel imports dates as **POSIXct** (timestamp format) instead of standard date formats. To correct this, we use the **lubridate** package.

Choosing the Right lubridate Function

- `mdy()` → If dates are formatted as **"MM/DD/YYYY"** (Month-Day-Year)
- `ymd()` → If dates are formatted as **"YYYY-MM-DD"** (Year-Month-Day) - used by POSIXct
- `dmy()` → If dates are formatted as **"DD/MM/YYYY"** (Day-Month-Year)
- `ymd_hms()` → If timestamps include **hours, minutes, seconds**

The standard format for POSIXct uses the **ymd** function, but occasionally you may need to try one of the other functions above. These functions can also be used to convert date fields in other data classes (character or numeric) to the date data class.

Example: Converting Date Fields from Excel

```
# Load lubridate for date conversion
library(lubridate)
# Convert from Excel-imported POSIXct to Date format (if necessary)
ExcelData$DateSale <- ymd(ExcelData$DateSale)  # If using "YYYY-MM-DD"
```

Step 5: Exploratory Data Analysis (EDA)

This is where we review the ADF to help us narrow to the CMS. We see how the subject fits in the ADF. There are other tools available, for this lesson, we'll build a histogram and scatter plot.

Histogram of Sale Prices

```
# load the ggplot2 package for data visualizations
library(ggplot2)
# create the histogram
ggplot(ADF, aes(PriceSale)) + # specify data and variable
  geom_histogram(binwidth=10000, fill="blue", color = "gray") + # specify plot type
  labs(title="Histogram of Sale Prices", # Add labels
       x="Sale Price",
       y="Volume")
```

Scatterplot of Sale Price vs. Date

```
# the ggplot2 library was already loaded in the chunk above
ggplot(ADF, aes(DateSale, PriceSale)) + # specify data and variables
  geom_point(color = "blue") + # specify plot type
  geom_smooth(method = "lm", # add a smoother - linear method
              color = "red", # make the line red
              se = FALSE) + # set standard error shading off
  labs(title="ADF Price Index", # add our titles
       x="Date",
       y="Sale Price")
```

In the next lesson, we will **filter the dataset to a Comparable Market Segment (CMS)**.