

Exploratory Data Analysis: How To Take The First Steps To Prep Data For Machine Learning



Pierre DeBois

Follow

Mar 12 · 7 min read ★

Data is essential for creating an analysis. But not every dataset is useful right off the bat. Raw entries with unfamiliar column labels, mixed field types within column, and omitted fields require consolidation. Discussions about data have evolved from gathering big data to examining the data obtained. Determining how to set up data is at the heart of Exploratory Data Analysis (EDA).

EDA drives the quality of every model you want to build. The steps in a EDA refines data from its unstructured appearance in datasets into a format that a model can read. EDA is necessary to prepare for a regression model or machine learning initiative.

Start With Forming The Question Behind The Data Being Selected

To get started with EDA, you should form an hypothesis for your business case. A hypothesis is the premise or claim to be tested. The data provides the observations to be tested, with the hypothesis claiming that given two samples, the means are not equal. The business case must address the outcome of this comparison.

The hypothesis (and thus the business question being tested) must be a unambiguous statement, where the results may be binary. In addition, the hypothesis proposes that no significant difference exists in a given set of observations.

To support that binary and difference qualities in the hypothesis, you would want a null hypothesis, a counter-instance to the hypothesis that proves that given two samples the means of each sample are equal.

Outlining a hypothesis may sound overly technical, but it has some essential advantages for a business case (beyond being statistically necessary for your model). Creating a hypothesis positions your business objective as an unambiguous question to investigate, one with results that can then be expressed against the data you have.

The end result is that better assumptions and questions can be formed. This leads to better decisions to organize data cleaning tasks, and in some instances, better decisions on whether or not seeking more data is needed.

Select Data For Your Analysis

Speaking of data, your next step is examining the data itself. Datasets usually contain unstructured observations — various field types, unusual characters, and categories labels.

Either R Programming or Python will work well for EDA— I'm using R programming for this post. Both have a few built-in functions to figure out how to adjust the dataset. But you will want to use additional functions to gain more nuanced details or visualizations that aid exploration decisions.

Libraries — dependency functions used in R programming — can provide the additional functionality. Libraries are held within packages, a dependency file that the user downloads into their program to access the functions. For example, there are libraries designed to import data from different databases, saving you time from typing out syntax for API calls.

So to import your data, you'll have to place it in a object— R is object oriented, just like many programming languages. In general there are specific kinds of table objects you can set to your data — a matrix, data frame or a data table. Data frames and data tables are typically used by functions and libraries. When you are planning your program, you can always check the library reference manual — it's a pdf document that comes with every package — to verify what objects are being called. The document explains the library and the functions available.

When exploring data you should consider how the following points impact your planned models and data visualization choices.

- Identify data types for the value — are they true numeric or just characters? How does your model generally recognize those value types?
- Identify variables as continuous or categorical.
- Identifying if numerical values are normally distributed — is the data skewed? Are there outliers? Statistical metrics such as skewness can describe how normal the data is (Skewness indicates data distribution around a mean, median, and mode of data in a column, impacting an algorithmic choice for a model).
- Decide if a program line should address how data is imported. Decision such as which columns should be joined or separated may require library functions to automate these tasks when new datasets are used.
- Use domain knowledge about the data to determine how to treat special characters of filter for intended phrases.

Select Functions That Can Examine The Observations

To explore, you can start with a few select functions. A few useful ones are already built into R programming.

One common one is **head()**, which displays the first few rows of a dataset. Invoking it means typing “**head(x, y)**” where x is the variable containing the data and y is an optional argument — the number of rows to display. A complimentary function, **tail()**, yield the last rows of a dataset.

Systematic missing values must be identified because most machine learning models do not process missing values well. In R you can use the function **is.na()** to display the observations in a dataset. In the example, the function is returning the phrase FALSE for each row in the data frame. The column names are called out for the dataframe. You can also use a nested function **any(is.na())** to return a single FALSE or TRUE value for a given set of observations.

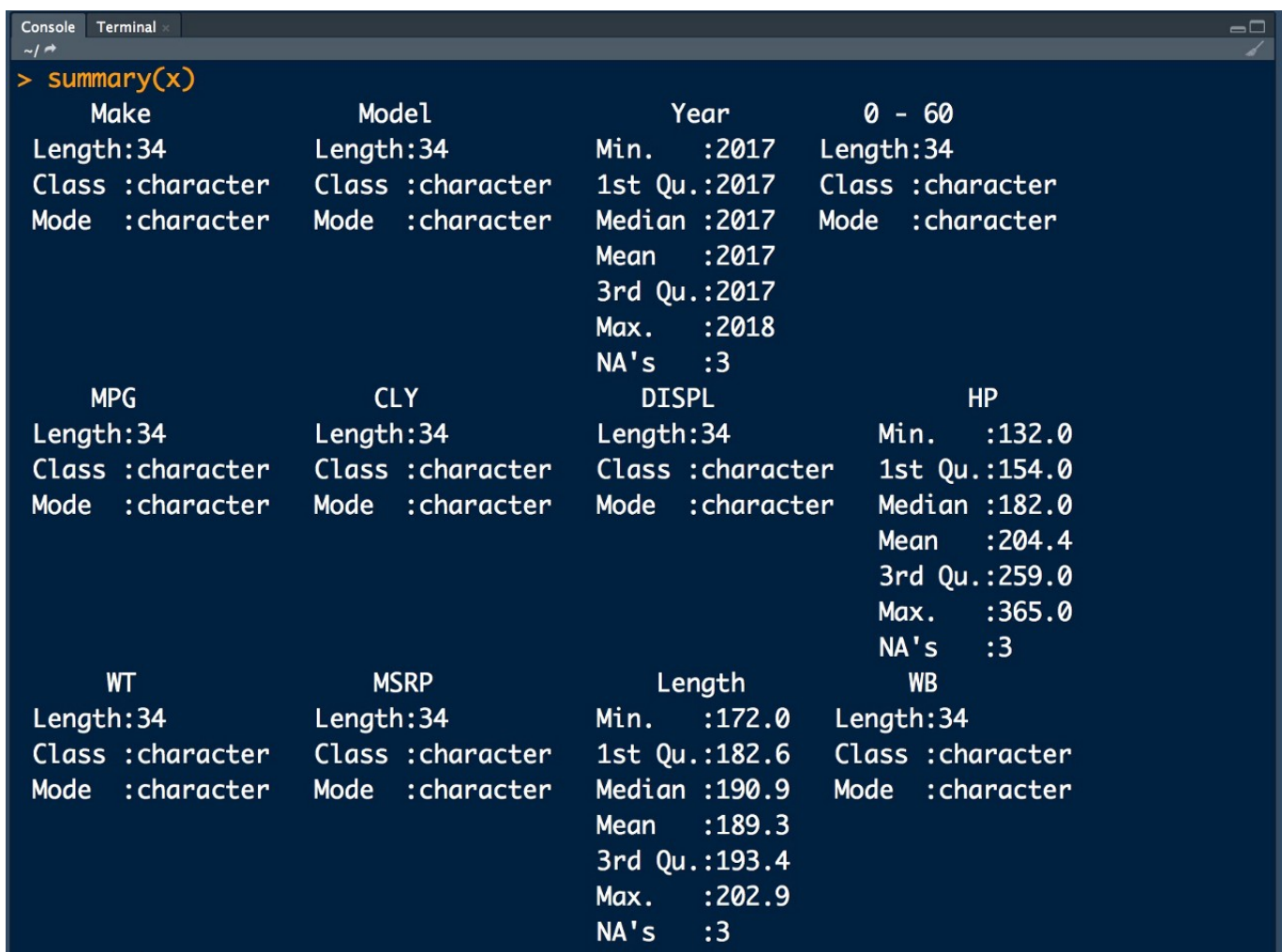
```
> is.na(x)
      customer_id revenue most_recent_visit number_of_orders
[1,]      FALSE  FALSE                FALSE             FALSE
[2,]      FALSE  FALSE                FALSE             FALSE
[3,]      FALSE  FALSE                FALSE             FALSE
```

```

[3,] FALSE FALSE FALSE FALSE
[4,] FALSE FALSE FALSE FALSE
[5,] FALSE FALSE FALSE FALSE
[6,] FALSE FALSE FALSE FALSE
[7,] FALSE FALSE FALSE FALSE
[8,] FALSE FALSE FALSE FALSE
[9,] FALSE FALSE FALSE FALSE
[10,] FALSE FALSE FALSE FALSE

```

Another function, **summary()**, gives quartiles, minimum values, and maximum values for columns that have numeric values. Columns with characters are listed with lengths.



```

> summary(x)
  Make           Model           Year           0 - 60
Length:34      Length:34      Min.   :2017      Length:34
Class :character Class :character 1st Qu.:2017      Class :character
Mode  :character Mode  :character Median :2017      Mode  :character
                                Mean  :2017
                                3rd Qu.:2017
                                Max.   :2018
                                NA's   :3

  MPG           CLY           DISPL           HP
Length:34      Length:34      Length:34      Min.   :132.0
Class :character Class :character Class :character 1st Qu.:154.0
Mode  :character Mode  :character Mode  :character Median :182.0
                                Mean  :204.4
                                3rd Qu.:259.0
                                Max.   :365.0
                                NA's   :3

  WT           MSRP           Length           WB
Length:34      Length:34      Min.   :172.0      Length:34
Class :character Class :character 1st Qu.:182.6      Class :character
Mode  :character Mode  :character Median :190.9      Mode  :character
                                Mean  :189.3
                                3rd Qu.:193.4
                                Max.   :202.9
                                NA's   :3

```

Another, called **sd()**, yields a standard deviation.

Another built-in function, called **structure**, can show the structure of the data. It is invoked by typing **str()** at the terminal.

But you will likely need libraries that can handle data in a table and add more statistical details, enough to help answer data quality questions. These libraries blend features similar to those of the built-in functions, but include new functionality that can be convenient for large datasets.

Take the library **funModeling**. It provides a function, **df_status**, which identifies bad observations in your data with more detail than the built-in functions. It provides a table that list zeroes, NA, infinite values, and the type of value.

```
> df_status(mtcars)
```

	variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
1	mpg	0	0.00	0	0	0	0	numeric	25
2	cyl	0	0.00	0	0	0	0	numeric	3
3	disp	0	0.00	0	0	0	0	numeric	27
4	hp	0	0.00	0	0	0	0	numeric	22
5	drat	0	0.00	0	0	0	0	numeric	22
6	wt	0	0.00	0	0	0	0	numeric	29
7	qsec	0	0.00	0	0	0	0	numeric	30

Another library, **stringr**, has functions that identify character strings within the observations. This is useful when there is a certain pattern you want to modify in the observations. **str_direct()** inspects the column for characters, while **str_subset()** returned the character elements that contain a given string. You can also can automate changes to the observation if a string is encountered. **str_sub()** allows for a modification of a string character.

A library you will likely use for exploratory data analysis is called **xda**. It contains functions to detect data type.

Another library, called **dplyr**, has a function, **glimpse()**, which can reveal the observations, number of rows, and what data type exists on each column, along with a mini-preview of the data.

```
Console Terminal x
~/
> glimpse(z)
Observations: 3
Variables: 14
$ `Reporting Starts` <date> 2018-09-01 2018-09-01 2018-09-01
```

```

$ `Reporting Ends`      <date> 2018-09-30, 2018-09-30, 2018-09-30
$ `Campaign Name`      <chr> NA, "Google Analytics Workshop", "Engagement"
$ Delivery              <chr> "0", "completed", "inactive"
$ Budget                <int> 0, 11, 11
$ `Budget Type`        <chr> "0", "Daily", "Daily"
$ Results               <int> NA, 39, 1
$ `Result Indicator`    <chr> NA, "actions:rsvp", "actions:rsvp"
$ Reach                 <int> 2339, 2334, 10
$ Impressions           <int> 3893, 3883, 10
$ `Cost per Results`    <dbl> NA, 2.83, 0.13
$ `Amount Spent (USD)`  <dbl> 110.50, 110.37, 0.13

```

There are also functions to merge, split, and transform rows and columns, much like you would in any database. **rbind()** and **cbind()** are built in function to combine a row and column. The dplyr library includes **slice()** and **filter()** to select rows, with the latter used to select rows according to a function or specification you provide.

There are more libraries and functions than I have listed here. But no matter what you choose to use, a key approach that should emerge from your effort is to identify the data types and observations, identifying which rows and columns may need adjustments and then using the aforementioned libraries to help you arrange the data into a useful format for your intended model.

A quick point about examining data.

First, some data types must be contained in a certain object type to be examined. R programming, for example, uses a specific object for time series data and geospatial data. The kind of analysis you conduct will dictate what kind of object is best, though you can write a line of code to change objects for a library function.

Second, every model you decide to create has a variation of what data types are accepted. In some instances it dictates what editing is needed. For example, in TensorFlow, dataset columns containing text must be either ignored or transformed by reshaping the dataset. The reshaped table will move text into the column labels and have numeric values in the observations. Those values can then be realigned as normalized values, which the algorithm can then process as a vector.

Exploring data can be time intensive projects that seems to be an expense for an organization. Yet data exploration allows a business to operate its profit drivers better.

With the rise of data analysis as a strategic advantage, business leaders can use this EDA framework to know where to explore data that is critical to a long-term strategy.

[Data Science](#)[R Programming](#)[Business Intelligence](#)[Business Strategy](#)[Analysis](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

