Chapter 3: Getting to know your data

Steve Elston

March 27, 2020

Introduction to Exploratory Data Analysis

Why should you learn about exploratory data analysis (EDA)? And, why discuss this topic in a book on computational statistics? Simply put, understanding the data you are working on is an essential step in any data science project. By exploring your data you gain understanding of the relationships in the data; what seems important, what might not be. A good understanding of your data set can save you quite a lot of time by focusing your efforts in the right direction.

We can define EDA as exploring complex data sets, using summary statistics and visualization, to develop understanding of the inherent relationships. Why is the focus here on visualization? Most people have excellent visual perception. Therefore, visualization is a key technique in exploratory data analysis. Not only that, a few good visualizations are a powerful way to communicate your insights to your colleagues. And, communications is a critical step in the data science process.

Virtually all data science projects require model building of some type: an inferential model, a predictive machine learning model, or a decision model. In any event, before constructing any type of model, you should develop an understanding of the relationships in the data set. Skipping this step can waste a lot of time from unexpected problems when building models. This point underscores how important EDA is.

EDA is used at every stage of good computational statistics practice. In this chapter, we will explore some basic concepts. In the next two chapters we will examine both visualization practice to improve human perception and visualization methods for large and complex data sets. In the remaining parts of this book we will use summary statistics and visualization methods not only for initial exploration of data, but also to evaluate and improve models.

The EDA Process

What can we say about the EDA process? EDA is inherently an iterative process. As the name implies, the results of any exploration are unknown before the process commences. In general, EDA will progress from the simplest of observations to generating some very specific and complex insights about the problem at hand. As you proceed you will **develop multiple views** of the relationships between the variables of complex data. Some of these views contribute significantly to your understanding. But be prepared, many views you create will not be very useful. This is an exporatory process after all!

However, it is undesirable, in fact impossible, to follow a pre-defined recipe for a given analysis. **Good EDA practice is relentlessly iterative**, and requires much trial and error. Most ideas tested will not work, but some will, and are therefore quite valuable. In summary, when performing EDA, plan on doing some significant trial and error work.

While we cannot define a specific recipe or process flow for EDA, we can define the components of the EDA process. These general components will be repeated many times, in an order determined by the state of the analysis. The state of the analysis being what you have learned and still need to learn about your data at any point in time. The key components of EDA are:

1. Summary statistics help develop an overall initial understanding of the properties of the data. As the process continues, summary statistics can be used to help guide the exploration.
2. Using appropriate chart types to create multiple views of the data. These views highlight different aspects of the inherent relationships between the variables.
3. Identify and correct problems with the pre-processing or preparation of the data. It is often the case that problems like missing data, incorrectly coded values, erroneous values and the like are discovered during the EDA process.
4. Evaluate and improve models using a combination of summary statistics and visualization. We will take up this component of the process in Part 4 of this book.
5. Continue the above steps to find important relationships between the variables. Likewise, it is common to discover that some relationships are not, in fact, important.

History of Exploratory Data Analysis

The basic ideas of using a combination of summary statistics and charts to explore the data is several centuries old. This process was often ad hoc and not studied seriously until the late 20th Century. In fact, academic statistics thought most of the 20th century was focused almost exclusively on the mathematical aspects of statistics, rather than data exploration.

19th Century Data Analysis

Several pioneers of exploratory graphics worked in the late 18th and early 19th Centuries. For example, Tufte (2001) credits **William Playfair** with being the first person to systematically use visualization to explore data. Playfair was a colorful character, known for his work in engineering and economics, as well as, being a spy for the British Government. Interest in Playfair's work continues in our present century. See for example, Spence and Waine (2005).

Graphics played a significant role in 19th century statistics. Before many of now commonly used mathematical methods were developed, graphical analysis was a primary tool. These analyses included pioneering work in medical statistics.

Florence Nightingale was a 19th Century innovator in medical data analysis (Nightingale 1859). McDonald (2020) describes her work with the noted medical statistician Dr William Farr. Nightingale and Farr documented the effect of implementing the recommendations of the Royal Sanitary Commission on reducing deaths of British soldiers during the Crimean War (1855-1856). Nightingale wrote several reports on the subject, the last of which included the area chart shown below. You can see the dramatic reduction in deaths after March 1855.

Nightingale's publications and public lectures where quite influential. She captured the imagination of the British public and triggered action by the political establishment.

Another innovator in 19th Century medical data analysis was John Snow (Snow 1854). Snow investigated the 1854 outbreak of cholera in London. He created the map shown below to identify water from a specific pump as the source of the infections. Unfortunately, there was only a minimal official response to Snow's work during his lifetime.

Tufte (2001) has documented many other examples of innovative 19th Century data visualization.

EDA in the 20th Century

For most of the 20th Century data visualization was relegated to a minor role. Mathematical statistics was considered more important and defined the mainstream. The very influential, and now controversial, statistician Ronald A Fisher (Fisher 1925a) (Fisher 1925b) was a proponent of charting data. However, the

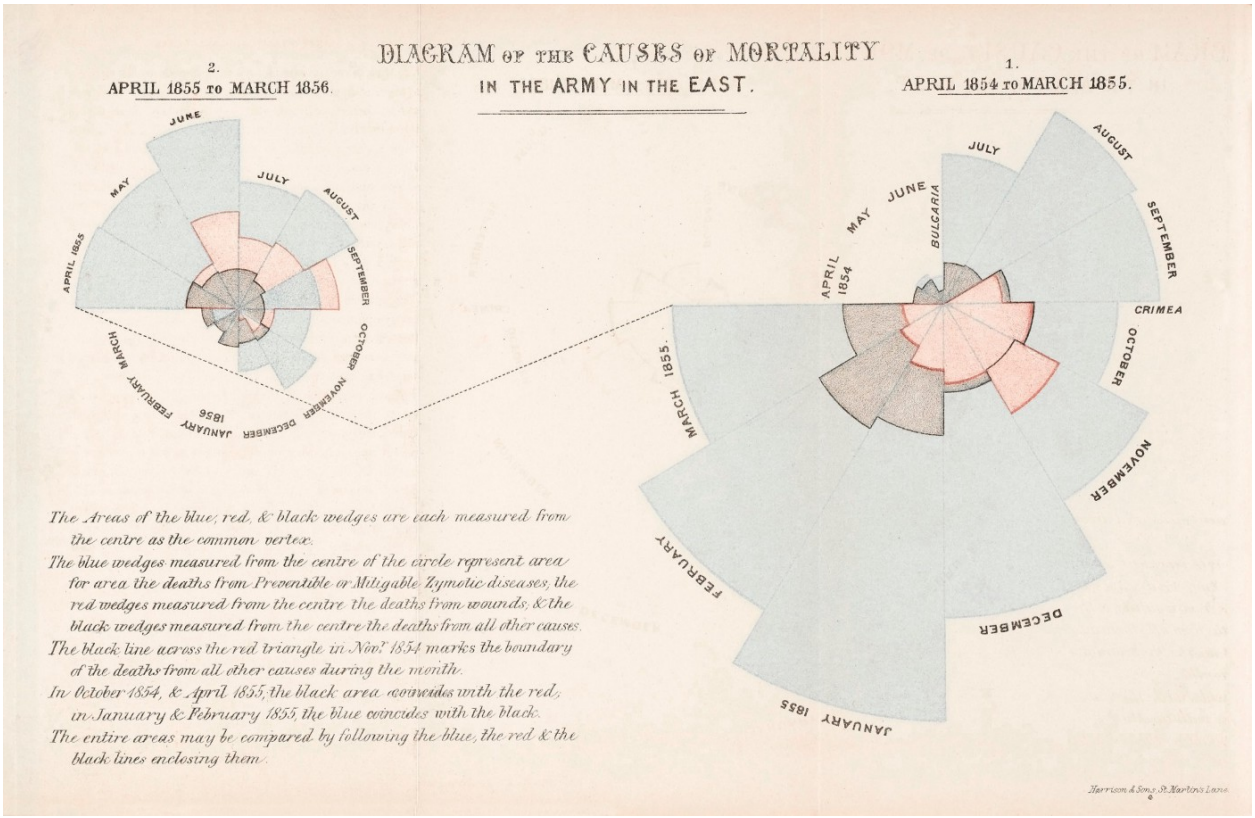


Figure 1: Rose diagram of British Army deaths by cause, published by Florance Nightinglale in 1858

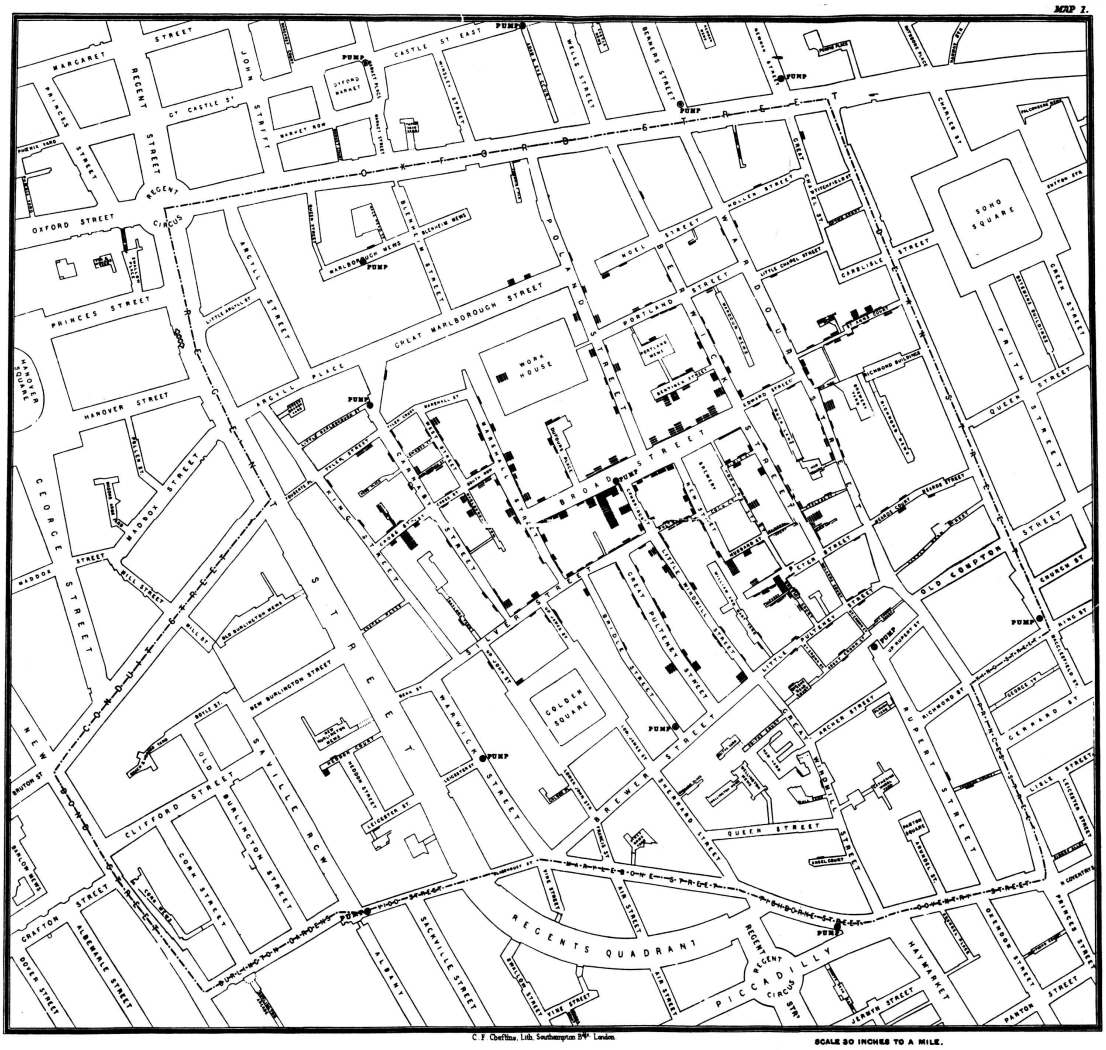


Figure 2: Map of the London Cholera outbreak compiled by John Snow, 1854

broader early 20th Century statistics community mostly focused on the mathematical aspects of Fisher's work.

The modern history exploratory data analysis begins with the American statistician, John W Tukey. In the 1960s and 1970s Tukey and his colleagues developed many techniques used today. Tukey consolidated his many ideas on data exploration into a book in the late 1970s (Tukey 1977).

William Cleveland performed considerable research on methods to visualize complex data an visualization methods which enhance human perception. We will use ideas from Cleveland's work in this part of this book. Cleveland documented his seminal work in two books (Cleveland 1993) (Cleveland 1994).

Edward Tufte's research focuses on visualization of complex data relationships. He has advanced ideas for creating plots free of clutter to enhance presentation. Tufte has published a number of books, including his first book (Tufte 2001).

Example of Data Exploration

To demonstrate the principles of EDA, we will use an automotive data set in the first parts of this chapter for the running example. The data set contains the characteristics of a number of automobiles. Our goal is to understand how the characteristics of car affect its fuel economy and price.

About the data set

The data set is from the University of California Irving Machine Learning Repository The data was compiled by Jeffrey C. Schlimmer from the following sources:

- 1985 Model Import Car and Truck Specifications, 1985 Ward's Automotive Yearbook.
- Personal Auto Manuals, Insurance Services Office, 160 Water Street, New York, NY 10038
- Insurance Collision Report, Insurance Institute for Highway Safety, Watergate 600, Washington, DC 20037

Importing and preparing the data

Now it is time to load the data set. While this is not a book about 'data munging' or Python programming we do need to perform some data preparation before moving to EDA.

The code shown below loads the data set from the UCI Machine Learning Repository. Several steps required:

1. Import the Python packages required for the rest of this chapter.
2. Load the data set from the UCI Machine Learning Repository.
3. Give the columns of the data frame useful and Python-friendly names. These names use the `_` character, and avoid characters like `.` and `-` having meaning to a Python interpreter.
4. Rows with missing values are removed. While other treatments of missing values are possible, we take the simple approach here. The missing values are coded with a `?` character. Any value of `?` is replaced with a Numpy `nan` and the Pandas `dropna` method is applied to the data frame.
5. Finally, some columns which should contain numeric values, but are coded as character strings, are coerced to the correct type.

```
import pandas as pd
import numpy as np
import io
import requests
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

## Load the .csv file
auto_price = pd.read_csv('data/Automobile price data _Raw_.csv')

## Add column names
auto_price.columns = ['symboling', 'normalized_losses', 'make', 'fuel_type', 'aspiration',
                      'num_of_doors', 'body_style', 'drive_wheels', 'engine_location',
                      'wheel_base', 'length', 'width', 'height', 'curb_weight', 'engine_type',
                      'num_of_cylinders', 'engine_size', 'fuel_system', 'bore', 'stroke',
                      'compression_ratio', 'horsepower', 'peak_rpm', 'city_mpg',
                      'highway_mpg', 'price']

## Remove rows with missing values in numeric columns, accounting for mising values coded as '?'
cols = ['price', 'bore', 'stroke', 'horsepower', 'peak_rpm']
for column in cols:
    auto_price.loc[auto_price[column] == '?', column] = np.nan
auto_price.dropna(axis = 0, inplace = True)

## Convert some columns to numeric values
for column in cols:
    auto_price[column] = pd.to_numeric(auto_price[column])
```

Basic characteristics of the data

At the start of the EDA process, it is always useful to know some basic characteristics of the data set. We start here by looking at the shape of the data frame.

```
auto_price.shape
```

```
## (195, 26)
```

Knowing the data types of each column is essential as well. The `dtypes` attribute of the Pandas data frame tells us this.

```
auto_price.dtypes
```

```
## symboling              int64
## normalized_losses     object
## make                  object
## fuel_type             object
## aspiration            object
## num_of_doors          object
## body_style            object
## drive_wheels          object
## engine_location       object
## wheel_base           float64
## length               float64
## width                float64
```

```
## height               float64
## curb_weight            int64
## engine_type           object
## num_of_cylinders      object
## engine_size            int64
## fuel_system           object
## bore                 float64
## stroke               float64
## compression_ratio    float64
## horsepower             int64
## peak_rpm               int64
## city_mpg               int64
## highway_mpg            int64
## price                  int64
## dtype: object
```

The columns of this data set have three types: 1. Integer numeric values, shown as `int`. 2. Floating point numeric values, shown as `float`. 3. Strings, shown as `object`.

Let's have a peak at the contents of the data frame using the Pandas `head` and `tail` methods. This will help use get a feel for the characteristics of the data. Further, you can often spot errors and other problems with a data set quickly with such an inspection.

```
auto_price.head()
```

```
##    symboling normalized_losses         make  ... city_mpg highway_mpg  price
## 0          3                 ?  alfa-romero  ...       21          27  13495
## 1          3                 ?  alfa-romero  ...       21          27  16500
## 2          1                 ?  alfa-romero  ...       19          26  16500
## 3          2               164         audi  ...       24          30  13950
## 4          2               164         audi  ...       18          22  17450
##
## [5 rows x 26 columns]
```

```
auto_price.tail()
```

```
##      symboling normalized_losses   make  ... city_mpg highway_mpg  price
## 200         -1                95  volvo  ...       23          28  16845
## 201         -1                95  volvo  ...       19          25  19045
## 202         -1                95  volvo  ...       18          23  21485
## 203         -1                95  volvo  ...       26          27  22470
## 204         -1                95  volvo  ...       19          25  22625
##
## [5 rows x 26 columns]
```

You can see the data types and values in the columns. Notice that there are a wide range of numeric value ranges for the different variables.

Summary statistics

As the name seems to imply, **summary statistics** are used to summarize data. In other words, summary statistics are used to characterize numeric data using just a few values.

Examining summary statistics should be done early in the EDA process. Sometimes the values of the summary statistics can be surprising. If this is the case, you should carefully consider what you do not understand about the data, or perhaps there is some undetected error in the values.

There are a great many summary statistics used in statistics, many of them highly specialized. Here, we will only concern ourselves with some of the most commonly used methods.

Maximum and minimum

The **maximum** and **minimum** are simply the extrema of the values of a variable. The range of values is simply computed as, $maximum - minimum$. It is generally useful to know what the range of values are for each numeric variable. This range can be orders of magnitude different from variable to variable.

Unexpectedly large of small values are often a flag that something is wrong with the data. Are some values coded incorrectly? Did a decimal point get misplaced?

Mean

The **mean** or average value is the most commonly used summary statistic. The formula to compute the **empirical mean** or **sample mean**, $\bar{\mathbf{x}}$, from a set of n samples $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ is:

$$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

The sample or empirical mean, is the estimate of the mean computed from the available sample. All summary statistics discussed here are sample statistics. We will have more to say about this important topic in other parts of this book.

Standard deviation and variance

The mean is a useful summary statistic but it only provides one summary about the values of a variable. We would like to know other characteristics as well. For example, we would like a measure of the **dispersion** of the values. Simply put, variables with larger dispersion have values spread over a greater range, and vice versa. Therefore, having a measure of dispersion helps you understand the range of values of a variable.

A commonly used measure of dispersion is the **standard deviation**, or square root of the sum of the squared differences between the mean and each sample value. For our sample,$\mathbf{x}$, with mean $\bar{\mathbf{x}}$, the standard deviation, σ, is computed using the following formula.

$$\sigma = sqrt\Big[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{\mathbf{x}})^2\Big]$$

Where, $sqrt$ is the square root. Notice that the denominator for a finite size sample is $n-1$,mean rather than n. This is a **bias correction** since 1 degree of freedom is used to compute the mean.

The **variance** is write as, σ^2, and is just the square of the standard deviation, $\sigma^2 = standard\ deviation^2$. As you will see, the variance is an important property of probability distributions.

Median and quartiles

The median is not the only possible measure of **central tendency**. The **median** is a commonly used alternative. The median is simply the middle value of the ordered sample, $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$. The algorithm for computing the median is quite simple. If there is no middle value, the average is taken of the two values nearest the middle.

sort($\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$)
if(n is odd): $median = x_{n/2}$
else: $median = \frac{1}{2}(x_{n/2-1/2} + x_{n/2+1/2})$

The **quartiles** are computed by dividing the data samples into 4 equal parts. The quartiles provide a measure of the dispersion of the sample distribution. The quartiles are computed in a manner identical to the median. You can interpret the quartiles as follows: - One forth of the data samples have values less than the **first (lower) quartile**. - The second quartile is the median. - One forth of the data samples have values greater than the **third (upper) quartile**.

The concepts of the median and quartiles is illustrated in the figure. The black dots on the horizontal axis represent the sample values.

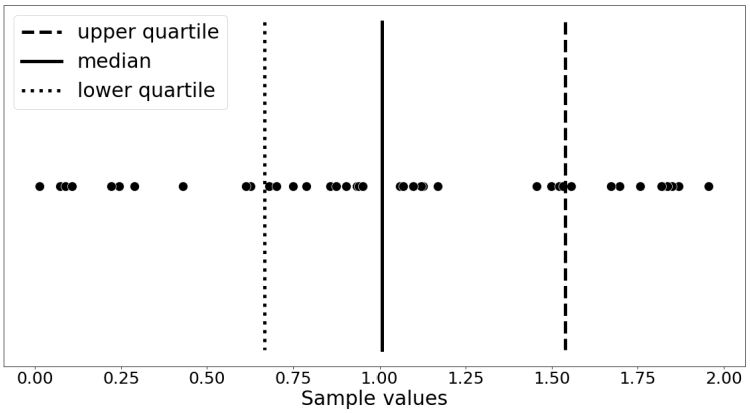


Figure 3: Quartiles and median of a sample

In the figure you can see that the average of the two values closest to the middle are used to compute the median.

Why would you need different measures of central tendency and dispersion? We will take up this question in more detail in Part 3 of this book. For now, a few points to remember are: - The median and quartiles are robust to a few **extreme values** or **outliers** in a data sample. The mean, and particularly, the variance can be influenced by a few extreme values. - The median can be quite different from the mean if the sample distribution is skewed.

Summary statistics for auto data

With the foregoing bit of theory in mind, it's time to compute and examine some summary statistics for a data sample. The code in the cell below uses the Pandas `describe` method to compute and display summary statistics for selected numeric columns. The sub-setting of the pandas data frame is performed with the `loc` method.

```
numeric_columns = ['wheel_base', 'length', 'curb_weight',
                   'engine_size', 'city_mpg',
                   'price']
```

```
pd.set_option('max_columns', None) # To print all columns of the data frame
auto_price.loc[:,numeric_columns].describe().round(decimals=1)
```

```
##        wheel_base  length  curb_weight  engine_size  city_mpg    price
## count       195.0   195.0        195.0        195.0     195.0    195.0
## mean         98.9   174.3       2559.0        127.9      25.4  13248.0
## std           6.1    12.5        524.7         41.4       6.4   8056.3
## min          86.6   141.1       1488.0         61.0      13.0   5118.0
## 25%          94.5   166.3       2145.0         98.0      19.5   7756.5
## 50%          97.0   173.2       2414.0        120.0      25.0  10245.0
## 75%         102.4   184.0       2943.5        145.5      30.0  16509.0
## max         120.9   208.1       4066.0        326.0      49.0  45400.0
```

Examine these results. There are a number of interesting characteristics of the variable you can see, in particular the following:

- The range of numeric values varies widely.

- For several variables, such as length and wheel base, the median (50% quartile) is close to the mean. Further, the median is nearly at the mid point between the minimum value and the maximum value. These facts indicating the distribution of values is close to symmetric.
- For several variables, such as price and city mpg the median (50% quartile) is much lower than the mean. Further, the median is closer to the minimum value than the maximum value. These facts indicating the distribution of values is skewed toward lower values.

Summaries For Categorical Variables

Categorical variables are quite common. Categorical variables have **discrete values**, often referred to as **levels**. These discrete values or levels represent the possible **categories**.

Clearly, numerical summary statistics are of no use with categorical variables. Instead, we must work with **counts**.

Unique values

The first thing one should know about a categorical variable is how many unique categories are there and what do they represent? The code below prints the categories of each variable of type `object` in the data frame.

```
for col in auto_price.columns:
    if(auto_price[col].dtype=='object'):
        print('\nLevels of {}'.format(col))
        print(auto_price[col].unique())
```

```
##
## Levels of normalized_losses
## ['?' '164' '158' '192' '188' '121' '98' '81' '118' '148' '110' '145' '137'
##  '101' '78' '106' '85' '107' '104' '113' '129' '115' '93' '142' '161'
##  '153' '125' '128' '122' '103' '168' '108' '194' '231' '119' '154' '74'
##  '186' '150' '83' '102' '89' '87' '77' '91' '134' '65' '197' '90' '94'
```

```
##  '256' '95']
##
## Levels of make
## ['alfa-romero' 'audi' 'bmw' 'chevrolet' 'dodge' 'honda' 'isuzu' 'jaguar'
##  'mazda' 'mercedes-benz' 'mercury' 'mitsubishi' 'nissan' 'peugot'
##  'plymouth' 'porsche' 'saab' 'subaru' 'toyota' 'volkswagen' 'volvo']
##
## Levels of fuel_type
## ['gas' 'diesel']
##
## Levels of aspiration
## ['std' 'turbo']
##
## Levels of num_of_doors
## ['two' 'four' '?']
##
## Levels of body_style
## ['convertible' 'hatchback' 'sedan' 'wagon' 'hardtop']
##
## Levels of drive_wheels
## ['rwd' 'fwd' '4wd']
##
## Levels of engine_location
## ['front' 'rear']
##
## Levels of engine_type
## ['dohc' 'ohcv' 'ohc' 'l' 'ohcf']
##
## Levels of num_of_cylinders
## ['four' 'six' 'five' 'three' 'twelve' 'eight']
##
## Levels of fuel_system
## ['mpfi' '2bbl' 'mfi' '1bbl' 'spfi' 'idi' 'spdi']
```

Notice that some of these variables have only a few levels whereas, other variables have a great many levels. There are also some variables which could be converted to numeric, such as, number of doors and number of cylinders.

Frequency tables

We need a method to summarize the frequency of categorical variables based on counts. Such a tabulation is known as a **frequency table**.

The code in the cell below computes and displays a frequency table by the following steps: 1. A count column of 1s is added to the data frame.
2. The frequency table is computed:
- The columns required are selected. - The count column is organized by level using the Pandas `groupby` method. - The sum of the counts for each group is aggregated (computed) using the Python aggregation method, `agg`.

```
auto_price['counts'] = 1
auto_price.loc[:,['counts', 'make']].groupby(['make']).agg('count')
```

```
##                counts
## make
## alfa-romero         3
## audi                6
## bmw                 8
## chevrolet           3
## dodge               9
## honda              13
## isuzu               2
## jaguar              3
## mazda              13
## mercedes-benz       8
## mercury             1
## mitsubishi         13
## nissan             18
## peugot             11
## plymouth            7
## porsche             4
## saab                6
## subaru             12
## toyota             32
## volkswagen         12
## volvo              11
```

Examine this result. The name of the `make` is in one column and the count is in the other. Notice that Toyota has the most cars in this sample. Whereas, some manufactures like Alfa-Romeo and Chevrolet have only three cars models in the sample.

Frequency tables with multiple levels

A multi-level frequency table can be constructed from two or more categorical variables. The groups are organized by the order of the columns specified. An example is shown in the code below:

```
auto_price.loc[:,['counts', 'fuel_type', 'make']].groupby(['make', 'fuel_type']).agg('count')
```

```
##                          counts
## make          fuel_type
## alfa-romero   gas             3
## audi          gas             6
## bmw           gas             8
## chevrolet     gas             3
## dodge         gas             9
## honda         gas            13
## isuzu         gas             2
## jaguar        gas             3
## mazda         diesel          2
##               gas            11
## mercedes-benz diesel          4
##               gas             4
## mercury       gas             1
## mitsubishi    gas            13
## nissan        diesel          1
```

```
##                gas            17
## peugot         diesel          5
##                gas             6
## plymouth       gas             7
## porsche        gas             4
## saab           gas             6
## subaru         gas            12
## toyota         diesel          3
##                gas            29
## volkswagen     diesel          4
##                gas             8
## volvo          diesel          1
##                gas            10
```

Examine this result. The data are grouped first by `make` and then by engine `fuel-type`. This table gives us a feel for which manufactures only make gas cars and which ones make both gas and diesel cars, and how many of each type. Notice that groups with counts of zeros do not display in the table.

This two-level frequency table can be reformatted into what statisticians call a **cross tabulation** or **contingency table**. Such a table shows the counts by the levels of one variable on an axis, by another categorical variable on the other axis.

You can continue to add levels to the frequency table. An example is created by the code below.

```
auto_price[['counts', 'fuel_type', 'aspiration', 'make']].groupby(['make', 'fuel_type', 'aspiration']).a
```

```
##                                    counts
## make          fuel_type aspiration
## alfa-romero   gas       std             3
## audi          gas       std             5
##                         turbo           1
## bmw           gas       std             8
## chevrolet     gas       std             3
## dodge         gas       std             6
##                         turbo           3
## honda         gas       std            13
## isuzu         gas       std             2
## jaguar        gas       std             3
## mazda         diesel    std             2
##               gas       std            11
## mercedes-benz diesel    turbo           4
##               gas       std             4
## mercury       gas       turbo           1
## mitsubishi    gas       std             7
##                         turbo           6
## nissan        diesel    std             1
##               gas       std            16
##                         turbo           1
## peugot        diesel    turbo           5
##               gas       std             5
##                         turbo           1
## plymouth      gas       std             5
##                         turbo           2
## porsche       gas       std             4
```

```
## saab          gas       std             4
##                         turbo           2
## subaru        gas       std            10
##                         turbo           2
## toyota        diesel    std             2
##                         turbo           1
##               gas       std            29
## volkswagen    diesel    std             2
##                         turbo           2
##               gas       std             8
## volvo         diesel    turbo           1
##               gas       std             6
##                         turbo           4
```

Notice that with three levels the table is quite a bit harder to understand. In general, as the number of grouping variables increases, frequency tables become progressively more difficult to understand, and the value of this method decreases.

> **Exercise:** Create the code to create a frequency table of body style and then number of doors. What does this table tell you about which types of cars are the most and least popular?

Introduction to Visual EDA

Now that we have examined some basic summary statistical methods we will turn our attention to the visualization aspects of EDA. As has already been mentioned, human visual perception is quite good. This makes visual methods of EDA extremely powerful.

The world of Python graphics libraries

There are a number of powerful charting packages available for the Python language. This situation can lead to confusion as to which package to use for which situation. Below is an outline to help you understand the organization of Python graphics choices:

Matplotlib

Matplotlib is a low-level scientific and technical charting package. A number of other Python charting libraries are built on top of Matplotlib. As a result, a bit of knowledge of Matplotlib will help you set the attributes of plots created with several other packages. Using these capabilities, you can create highly customized presentation quality charts. An extensive tutorial is available for Matplotlib. highly

Pandas plotting

The visualization methods for Pandas provide a simple interface for common plot methods for data frames. As with many other Python plotting libraries, Pandas visualization is built on top of Matplotlib. You can use Matplotlib functions to customize Pandas visualizations.

Seaborn

The Seaborn package provides high-level api for statistical graphics. Seaborn abstracts the details of some powerful statistical chart types. These charts make Seaborn ideal for EDA. Seaborn is built on top of Matplotlib. You can use Matplotlib to customize Seaborn plots. The Seaborn documentation is quite good, showing several examples for each plot type. An excellent tutorial with useful examples is available for Seaborn.

Interactive graphics packages

There are a number of other sophisticated and useful Python graphics packages. Detailed discussion of these packages is beyond the scope of this book.

There are two powerful open source Python interactive graphics packages which are widely used, plotly and Bokeh. Interactive graphics can be a powerful tool when exploring complex data.

Overview of basic plot types

There are a great many chart types used for data exploration. This plethora of charts arise for several reasons:
1. EDA of complex data sets requires multiple views of the data, with many views requiring different chart types.
2. There are numerous specialized charts, which have been often been developed over many years for specific domains.

Here we will only concentrate on a few of the most widely used chart types. This approach will give you a feel of the subject as well as get you started using key Seaborn chart types.

In this and subsequent chapters we will investigate ways to deal with a fundamental limitation of all computer graphics. All computer data graphics are projected onto a 2-dimensional surface. Very broadly, plots are one-dimensional (univariate) or 2-dimensional (bi-variate). In this and subsequent chapters we will examine methods to project multiple dimensions onto a 2-dimensions surface.

In the remainder of this chapter we will work with some common plot types used for EDA. The table below provides a summary we will explore:

Plot Type	Dimensions	Data types	Description
Count plot	1-d	Categorical	Counts by category
Bar plot	2-d	Numeric	Values by category
Histogram	1-d	Numeric	Count of values with bins
Box plot	2+d	Numeric	Distribution summary by categorical variables
Kernel density estimation plot	1-d or 2-d	Numeric	Estimate of probability density
Violin Plot	2+ d	Numeric	Distribution summary by categorical variables
Scatter plot	2-d	Numeric	Position by numeric value pairs
Line plots	2-d	Numeric	Position by ordered numeric value pairs

Introduction to Seaborn

Before making some plots, let's discuss the basic use of Seaborn plotting functions. The general structure of Seaborn calls is shown below:

```
sns.some_plot_type(x='x_variable', y='y_variable, optional_argument=value, data=a_data_frame)
```

This general function signature has the following arguments: 1. Specification of the x-axis and y-axis as variables or columns from the data frame. 2. Any optional arguments. There are often a great many arguments for most functions and it is always a good idea to look at the documentation and examples before trying a new plot type. 3. The data argument is a Pandas data frame.

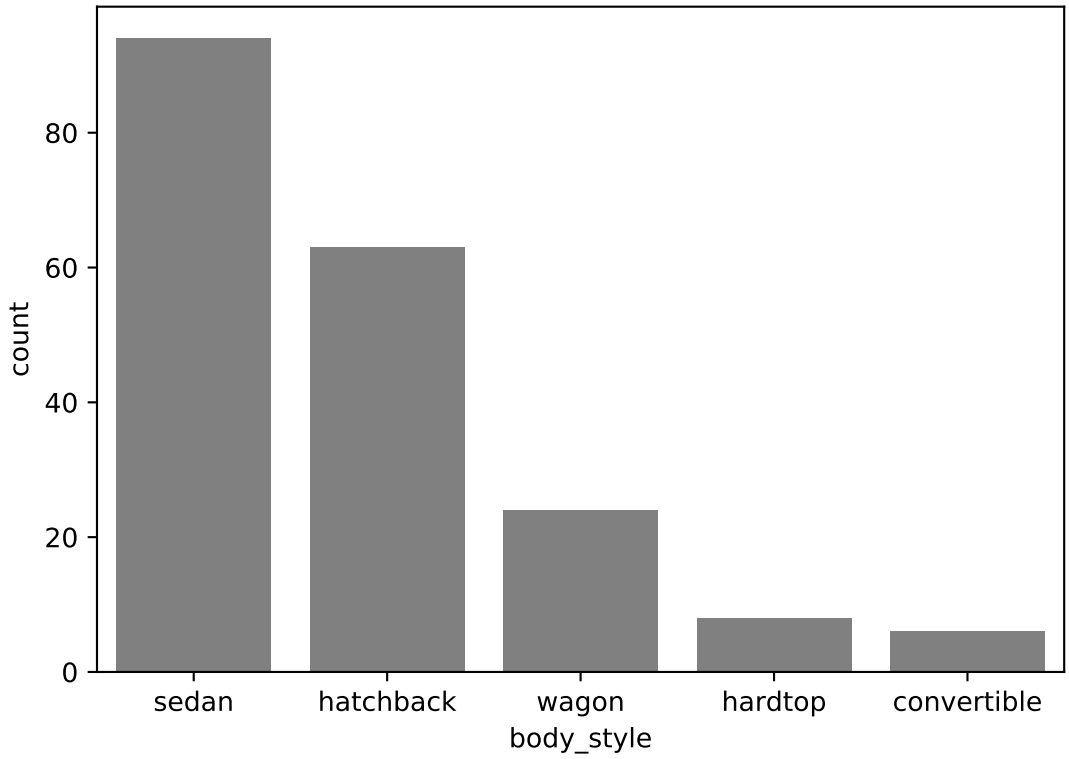
Not all Seaborn functions conform to this signature. You will see a number of variations as we proceed.

Bar plots

To start our exploration of chart types, we will work with **bar plots**. Bar plots are used to display the counts or frequency of unique values of a categorical variable. The height of the bar represents the count for each unique category of the variable.

The code in the cell below uses the Seaborn `countplot` function to make a bar plot of the frequency of body style. This function creates a bar plot for a variable in the data frame. There are a few details to discus:
1. A bar plot should always be ordered to make it clear to the viewer which category has a greater or lesser number of counts compared to another.
2. The `facecolor` of the bars is set to the same gray color. The default is to use a different color for each bar, which communicates no additional information and adds clutter to the chart.

```
sns.countplot('body_style', facecolor='gray',
              order = auto_price.body_style.value_counts().index,
              data=auto_price)
```



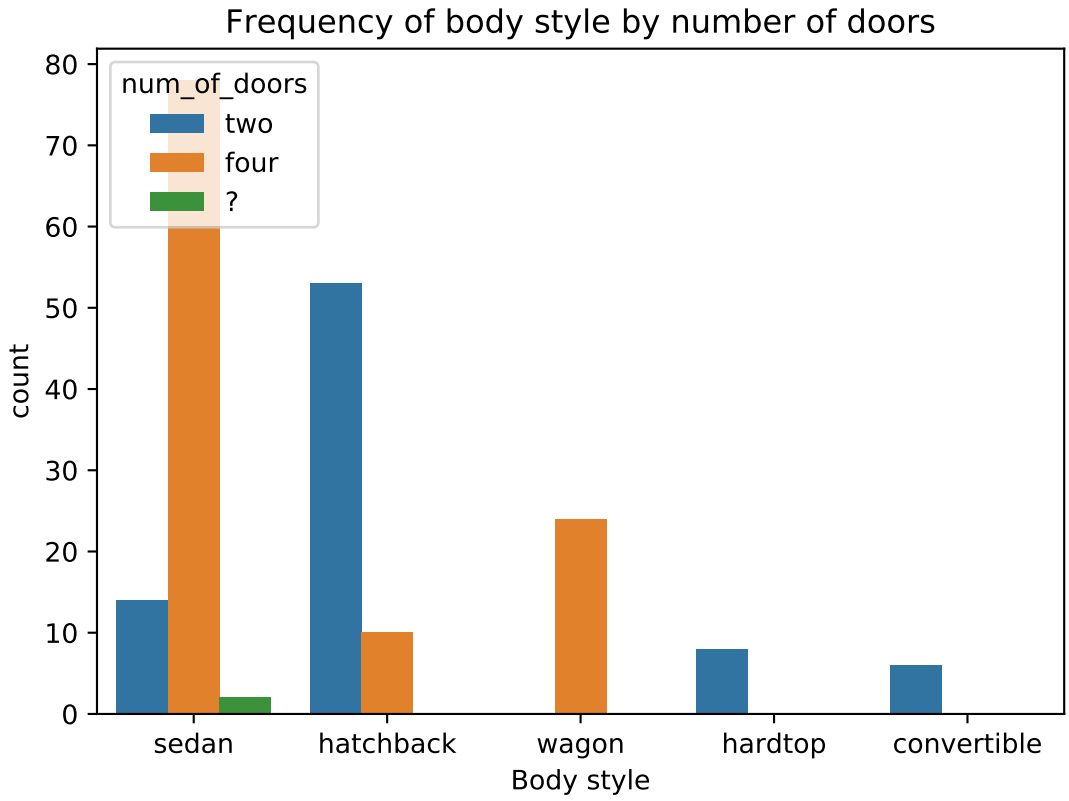
You can now see which body styles are the most and least popular. The information in the chart is easier to grasp than by staring at a table of numbers.

The display above is a bar chart for the counts of only one variable. But, even a small data set, like the automotive data, often has complexities. The code below creates a bar plot of the body style counts by number of doors.

We need to add proper annotation to the chart. A chart without annotation, such as a title and axis labels is generally useless. The viewer has no idea what they are looking at.

The Seaborn `countplot` function returns a Matplotlib axis object. Attributes of the axis object are set to desired values. In this case the x-axis label and the title.

```
ax = sns.countplot('body_style', hue='num_of_doors',
                   order = auto_price.body_style.value_counts().index,
                   data=auto_price)
ax.set_xlabel('Body style')
ax.set_title('Frequency of body style by number of doors')
```



Now you can see that the number of doors changes significantly with body style.

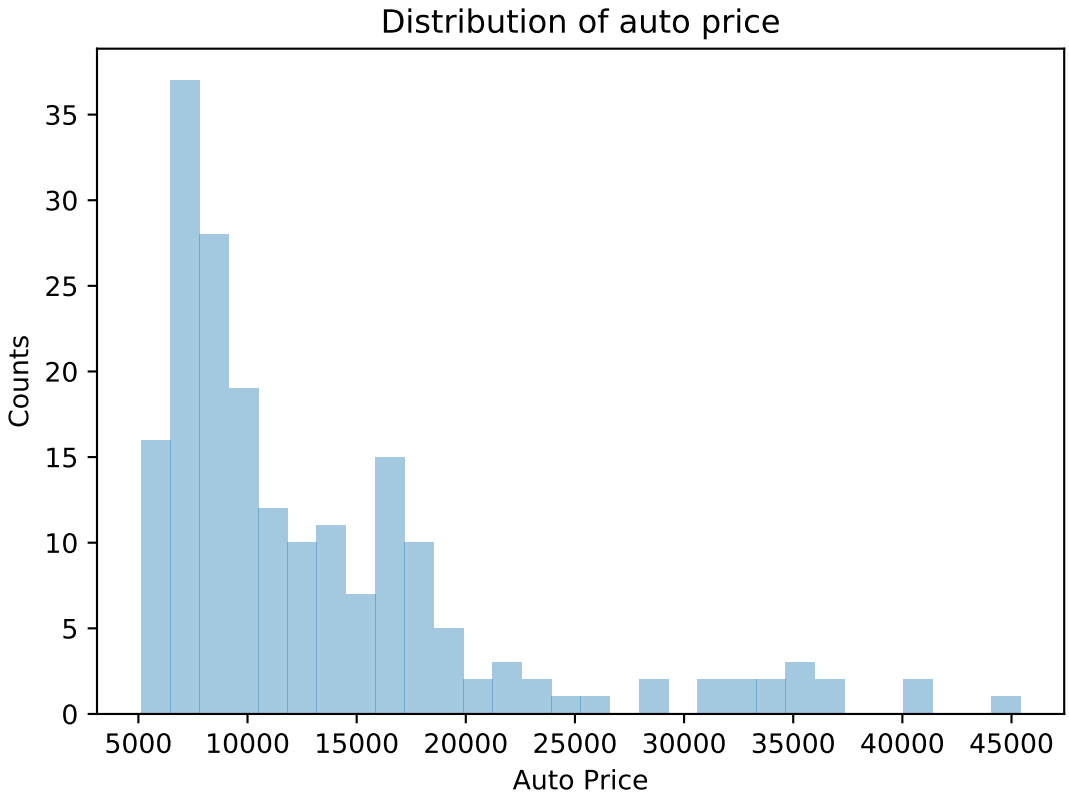
The two bar charts show different views of the relationships in the data. Neither chart contains the same information in an easy to view form. This is an important point for the EDA process. There is no one view of complex data that will tell you everything you need to know!

Histograms

Histograms are similar to bar plots. Histograms are used to display a representation of the distribution of a numeric variable. Whereas, a bar plot shows the counts of unique categories, a histogram shows the number of data values within a set of bins. The bins divide the values of the variable into equal segments. The vertical axis of the histogram shows the count of data values within each bin. The counts can be normalized to add to 1.0, so the bins can be understood as probability density estimates.

The code below creates a histogram of the price of the cars using the Seaborn `distplot` function. There are some points to notice about this code: 1. The Seaborn `distplot` function does not have a data argument. Unlike most other Seaborn chart functions, the column(s) to be plotted must be a subset of the data frame. 2. The `bins` argument sets the number of bins for the histogram. Changing the number of bins can change human perception of the distribution considerably.

```
ax = sns.distplot(auto_price.price, bins=30, kde=False)
ax.set_xlabel('Auto Price')
ax.set_ylabel('Counts')
ax.set_title('Distribution of auto price')
```



The distribution of auto price is far from symmetric. There are far more low priced cars. Further, there is a long 'tail' on this distribution, representing the few high priced cars; presumably luxury cars.

Kernel density estimation plots

Kernel density estimation plots or **KDE** plots are another way to visualize the distribution of a variable. This plot displays the results of running a truncated Gaussian kernel over the values of the variable. Regions with a large number of data values have high estimated density and vice versa.

The concept of a Gaussian weighting kernel is shown in the figure below. The density for each point in the interval is computed from a kernel weighted average of the sample values. The kernel moves over the interval to find the complete density estimate. The large dots on the horizontal axis represent the samples.

The Seaborn `distplot` function can also be used to display a KDE plot. An example is shown below. The `rug` argument is set to `True` so you can see the density of the data values for yourself.

```
ax = sns.distplot(auto_price.price, rug=True, hist=False)
ax.set_xlabel('Auto Price')
ax.set_ylabel('Density')
ax.set_title('Distribution of auto price')
```

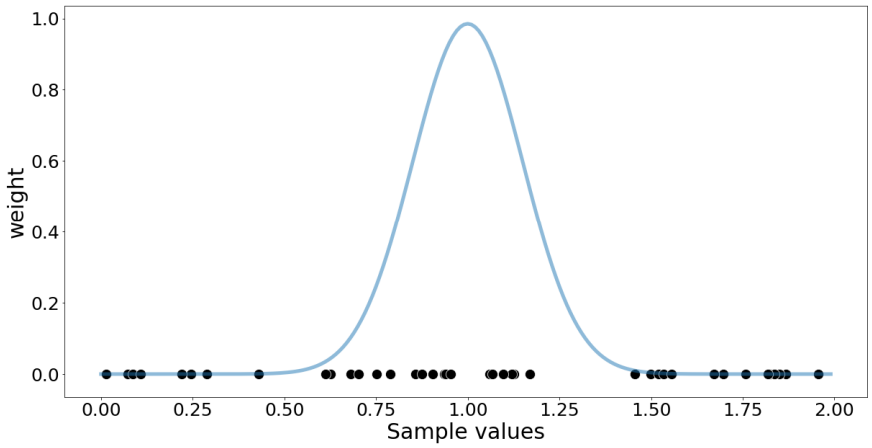
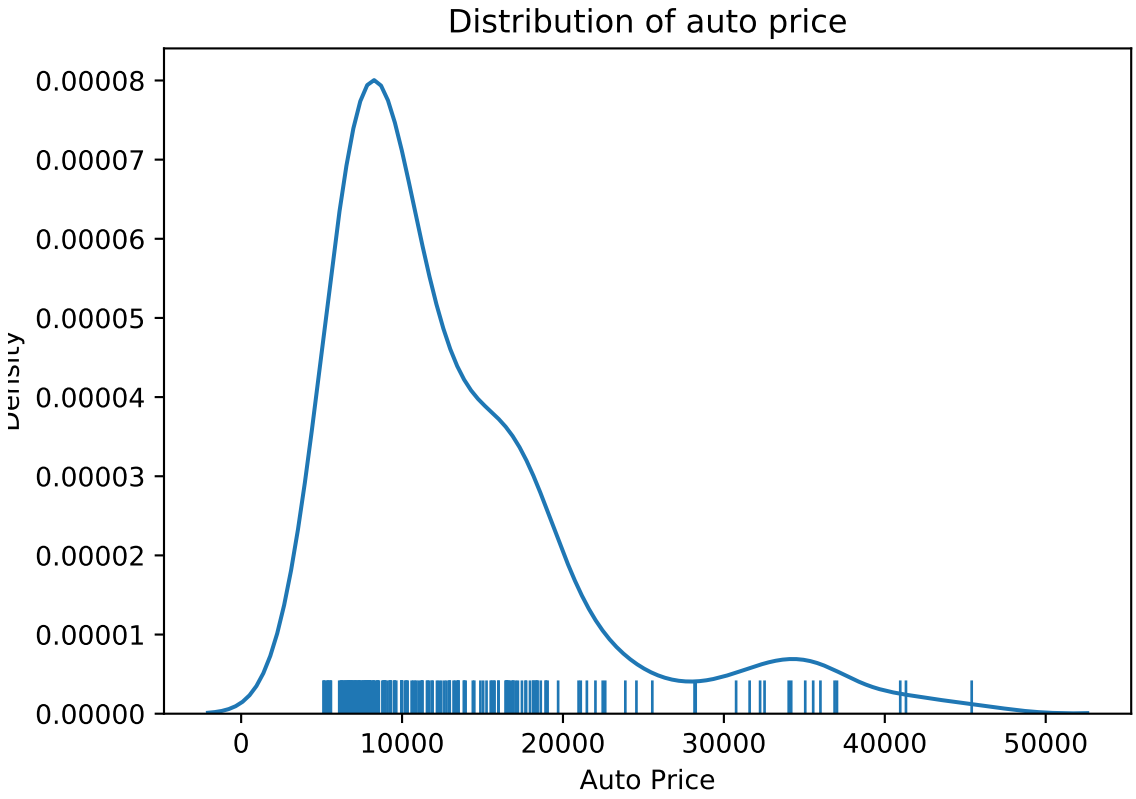


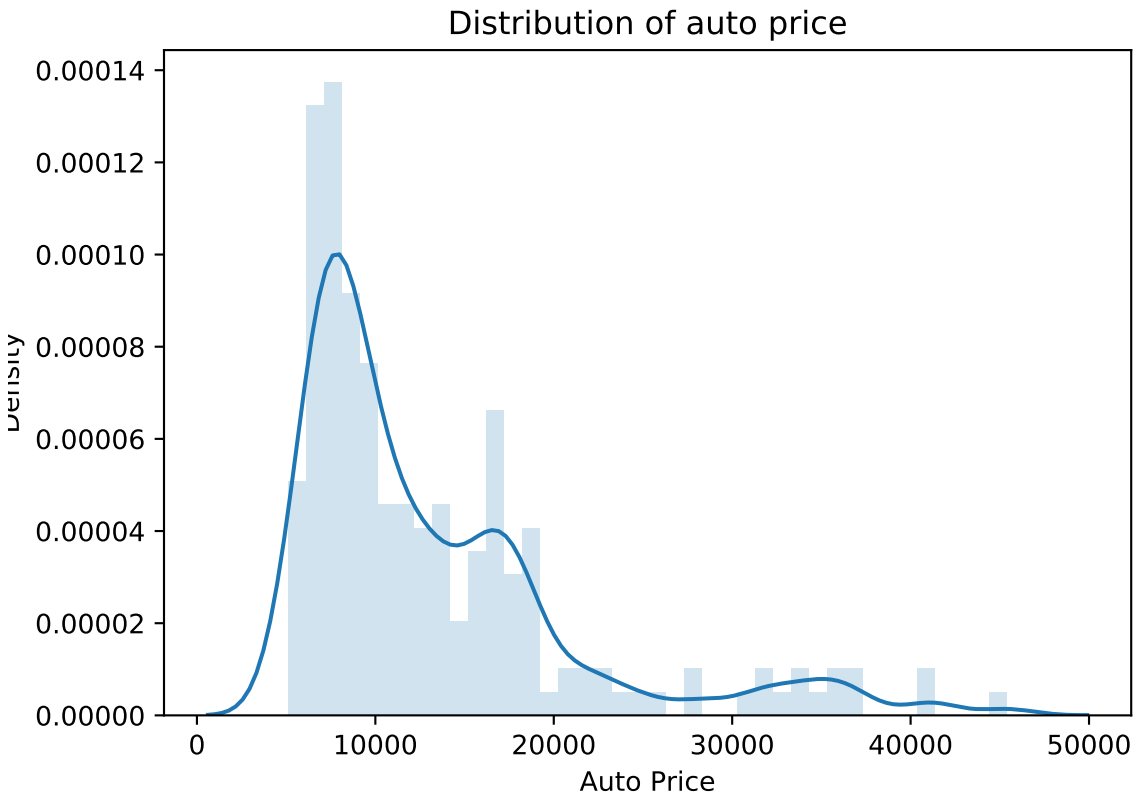
Figure 4: KDE weights on a data sample



The same general characteristics as observed with the histogram can be seen in the KDE plot. However, the smoothed character of the KDE curve can give a better impression of the overall shape of the distribution. Notice also, that the vertical scale is now in terms of density. This scaling ensure the integral under the KDE curve equals 1.

It is becoming common practice to show both the histogram and KDE curve on one plot. An example is generated below.

```
ax = sns.distplot(auto_price.price, bins=40, kde_kws={'bw':1500}, hist_kws={'alpha':0.2})
ax.set_xlabel('Auto Price')
ax.set_ylabel('Density')
ax.set_title('Distribution of auto price')
```



Having both the histogram and KDE curve on the same plot shows different levels of details; smooth and less smooth.

> **Programming Note:** The code in the cell above contains a special argument `kde_kws`. What does this argument do? To understand this, it helps to know that the KDE component displayed by `distplot` are from the `kdeplot` function. The argument `bw` sets the kernel bandwidth for the `kdeplot` function. The `kde_kws` argument passes a dictionary of arguments to `displot`. Likewise the `hist_kws` passes a dictionary of arguments to the function that generates the histogram. In this case the argument sets the density of the fill for the bars. Passing a dictionary of arguments to an underlying function is a common approach in Python.

Box plots

How do probability distributions of a variable change with respect to another variable? Specifically, what are the distributions of a numeric variable grouped by the categories of a categorical variable? Answering this question can give a great deal of insight into complex data.

A well established method for visualizing a distributions of a variable grouped by a categorical variable is the **box plot**. The box plot was introduced by John Tukey (Tukey 1977) as an EDA tool. The box plot is comprised of the following elements as illustrated in the figure below:

1. The bold line in the box shows the median of the distribution.
2. The upper and lower middle quartiles of the distribution define the upper and lower limits of the box.
3. The length of the whiskers is the lesser of +/-1.5 times the inter-quartile range (about 2.7 times the standard deviation) or the most extreme values of the data.
4. Outliers are shown by symbols, such as `+` or `*`, beyond the whiskers.

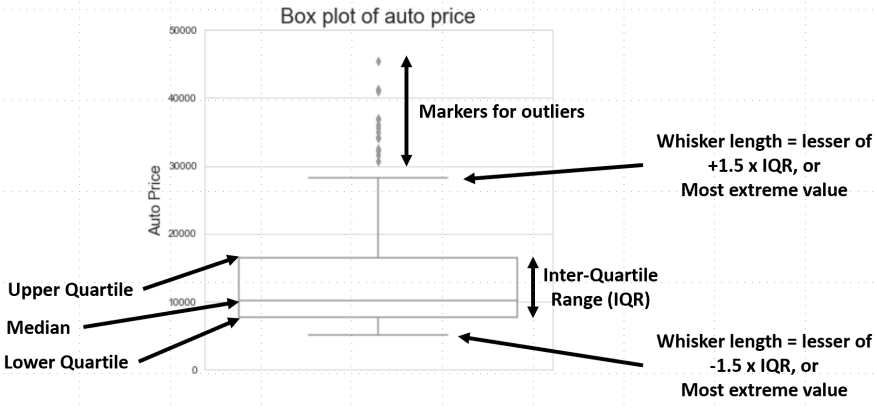
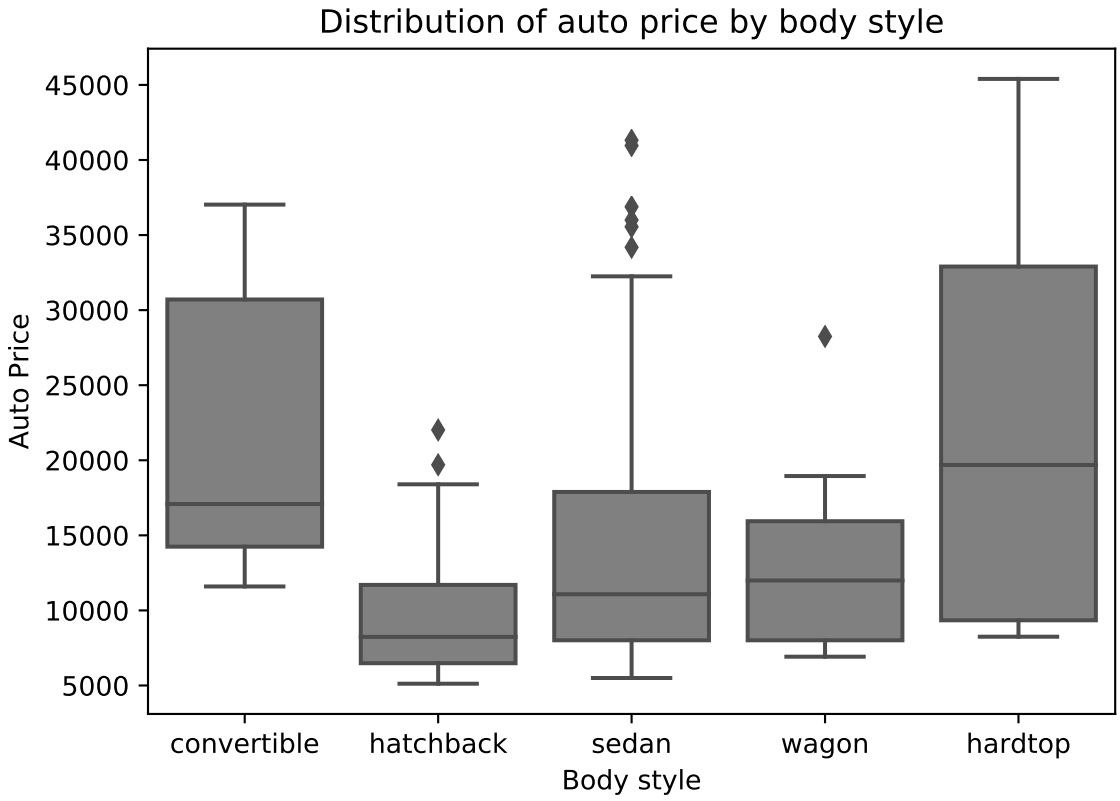


Figure 5: Interpretation of a box plot

The code below compares the the distribution of auto price, the `y` variable, grouped by the body style, the `x` variable. The fill color is set to gray to avoid distracting color paterns.

```
ax = sns.boxplot(x='body_style', y='price', color='gray', data=auto_price)
ax.set_xlabel('Body style')
ax.set_ylabel('Auto Price')
ax.set_title('Distribution of auto price by body style')
```



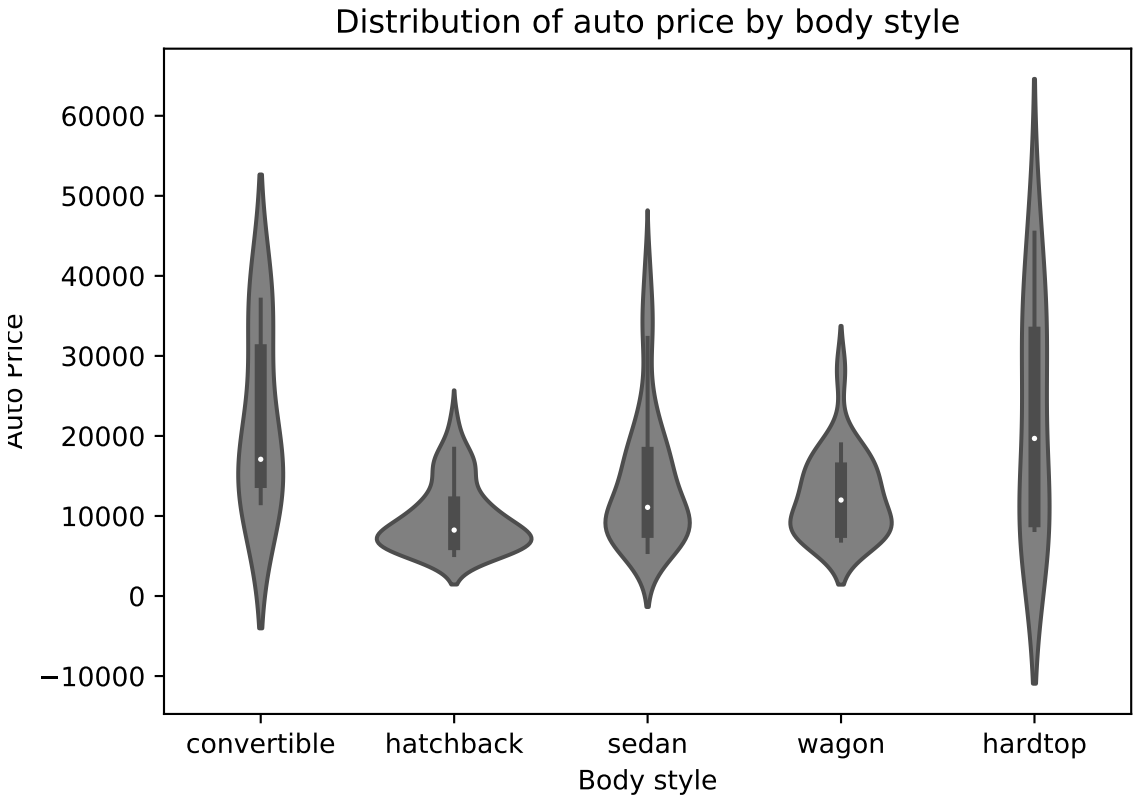
There is quite a bit about the price of autos you can learn from this chart. A few points to note: 1. Hatchbacks generally have the lowest prices and convertibles the highest. 2. Hard tops and convertibles have the widest price range. 3. The distribution of price for several body styles is skewed toward low prices. Notice the quartile below the median, the dark line, is generally narrower than the quartile above the median. 4. Price outliers are all at the upper end of prices.

Violin plots

The box plot was developed in the very early days of computer statistical graphics. Have alternatives been developed in the subsequent 50 years? Yes, several. One of the more useful alternatives is the **violin plot**. A violin plot is similar to a box plot except that it displays back-to-back KDE curves to show density. The implementation in Seaborn includes a boxplot figure between the KDE plots.

The code shown below, uses the Seaborn `violinplot` function. The fill color is set to gray, to avoid the unnecessary distraction of colors.

```
ax = sns.violinplot(x='body_style', y='price', color='gray', data=auto_price)
ax.set_xlabel('Body style')
ax.set_ylabel('Auto Price')
ax.set_title('Distribution of auto price by body style')
```



The violin plot shows more detail of the density of the subsets of the auto prices. The dot on the box plot is the median. A few details which are hard to ascertain from the box plots alone:
1. The distributions of prices for hatchbacks, wagons and sedans have highest density toward the low end of the range. 2. The distributions of prices of hardtops and convertibles is spread over a wide range.
3. One oddity, the KDE curve shows density for prices of less than zero. Using the `cut` argument to the `violinplot` function can reduce this effect.

> **Exercise:** Can we show distributions of a numeric variable grouped by more than one categorical variable? Yes, another variable can be projected by using color. With Seaborn, this task is accomplished using the `hue` argument. Create a new version of the foregoing violin plot, with the `hue` argument set to the fuel type. What additional information does this change give you about the price of cars? Have aspects of the original violin plot become harder to interpret in this new view?

If a violin plot shows more detail of a distribution, where would a box plot be useful? For situations where there is a small number of groups of categories, a violin plot is usually preferred. However, there are many situations with hundreds or thousands of groups where the simplicity of the box plot is preferred. In these situations the detail in violin plots is lost in any event.

How Where the Figure Plots Made?

You have likely noticed that there are two plots in this chapter for which code was not shown. These plots required setting many attributes. You may find seeing the code instructive.

Plot of quartiles

Let's start with the plot of the quartiles. The first step is to create some simulated data values for the plot. The code below created the values needed by the following steps:
1. The 'numpy.random.uniform' function is used to create 40 samples in the range 0.0 to 2.0.
2. A list of 40 0's is created.
3. The pandas data frame is created from a Python dictionary.

```
vals = nr.uniform(low=0.0,high=2.0,size = 40)
y = [0.0]*40
df = pd.DataFrame({'vals':vals, 'y':y})
```

Next, we create the plot by the following steps, which make use of our ability to specify almost any plot attribute using calls to Matplotlib.
1. The quartiles of the sample are computed using the NumPy `quantile` function.
2. The size of the plot area is set using the Matplotlib `figure` function.
3. A scatter plot of the samples is displayed. Two Matplotlib attributes are set, the marker size, `s`, and the marker color, `color`. This is an example of how attributes of the underling Matplotlib can be set using the `**kwargs` argument shown in the documentation. At first glance, this may seem a bit obscure, but with a little practice is not that hard to master.
4. The quantiles are plotted using the `matplotlib.pyplot.plot` function. Attributes set include the `linewidth`, the `label` for the legend, the `linestyle`, and line `color`.
5. The `legend` is displayed with the `fontsize` attribute set.
6. Tick text size is set.
7. The y-axis ticks are suppressed.
8. The label of the x-axis is set.

```
qu = np.quantile(df.vals,[0.25,0.5,0.75]) ## Find the quantiles

plt.figure(figsize=(20,10)) # Set the figure size
## Plot the sample values
ax = sns.scatterplot('vals', 'y', data=df, s=200, color='black')
## Plot the quantiles
plt.plot([qu[2],qu[2]], [0.1,-0.1], linewidth=5, label='upper quartile', linestyle='dashed', color='gra
plt.plot([qu[1],qu[1]], [0.1,-0.1], linewidth=5, label='median', color='black')
plt.plot([qu[0],qu[0]], [0.1,-0.1], linewidth=5, label='lower quartile', linestyle='dotted', color='gra
## Define some plot attributes
ax.legend(fontsize=30)
ax.tick_params(labelsize=25)
ax.get_yaxis().set_visible(False)
_=ax.set_xlabel('Sample values', fontsize=30)
```

Plot of KDE weights

The plot of the Gaussian kernel density estimation (KDE) weights was created in a similar manner to the plot of the quantiles. Again, this plot demonstrates some of the flexibility available using calls to Matplotlib.

The sample data is the same as used for the quantile example. We just need to generate some values for a Gaussian density function to plot by the following steps:
1. The Numpy `arange` function is used to create evenly spaced horizontal axis values.
2. The 'scipy.stats.norm.pdf' function is used to compute the density values for a Gaussian. Details of what this function computes will be discussed in Part 3 of this book.
3. A Pandas data frame is created.

```
x = np.arange(0.0,2.0,0.01)
gauss = ss.norm.pdf(x,1.0,0.15)/2.7
weights = pd.DataFrame({'x':x,'gauss':gauss})
```

Finally, the plot is created by the following steps. 1. The first two lines of code are identical to the first example.
2. The Seaborn `lineplot` function is used to display the Gaussian weights on the same figure axis using the `ax` argument. The Matplotlib attributes of `linewidth` and transparency, `alpha`, are set. 3. Other plot attributes are set with calls to Matplotlib functions.

```
plt.figure(figsize=(20,10))
ax = sns.scatterplot('vals', 'y', data=df, s=200, color='black')
sns.lineplot('x', 'gauss', ax=ax, data=weights, linewidth=5, alpha=0.5)
ax.tick_params(labelsize=25)
ax.set_xlabel('Sample values', fontsize=30)
_=ax.set_ylabel('weight', fontsize=30)
```

Summary

In this chapter we have discussed some basic ideas of exploratory data analysis using Python tools:

- Exploratory data analysis is an iterative process. It is impossible to say at the outset what the entire process is. Next steps are determined by the results obtained previously in the analysis.
- EDA is integrated into the modeling process.

- Failure to perform EDA can easily result in constructing models with poor performance.
- An initial impression of a data set is obtained using summary statistical methods for both numeric and categorical variables.
- Multiple graphical views are used to develop an understanding of different relationships in the data set.

It is clear there is more one can learn about the data sets discussed here. As is usually the case, the initial EDA has only scratched the surface of the important relationships.

Cleveland, William S. 1993. *Visualizing Data*. Hobart Press.

———. 1994. *The Elements of Graphing Data*. Hobart Press.

Fisher, Ronald A. 1925a. *Statistical Methods for Research Workers*. First Edition. Oliver; Boyd.

———. 1925b. *The Design of Experiments*. First Edition. Hafner Publishing Company.

McDonald, Lynn. 2020. "The Real Goods and the Oversell." *Significance* Vol. 17, No. 2: 18–21.

Nightingale, Florance. 1859. "A Contribution to the Sanitary History of the British Army." John W Parker; Son, London.

Snow, John. 1854. *On the Mode of Communication of Cholera*. C.F. Cheffins. http://matrix.msu.edu/~johnsnow/images/online_companion/chapter_images/fig12-5.jpg.

Spence, Ian, and Howard Waine. 2005. "William Playfair and His Graphical Inventions: An Excerpt from the Introduction to the Republication of His "Atlas" and "Statistical Breviary"." *The American Statistician* Vol. 59, No. 3: 224–29.

Tufte, Edward R. 2001. *The Visual Display of Quantitative Information.* Graphics Press.

Tukey, John W. 1977. *Exploratory Data Analysis.* Addison Wesley.