Importing Datasets

**Learning Objectives**

* Analyze Python data using a dataset
* Identify three Python libraries and describe their uses
* Read data using Python's Pandas package
* Demonstrate how to import and export data in Python

Python Packages for Data Science:

A Python library is a collection of functions and methods that allow you to perform lots of actions without writing any code. The libraries usually contain built in modules providing different functionalities which we can use directly. And there are extensive libraries offering a broad range of facilities. We have divided the Python data analysis libraries into three groups. The first group is called scientific computing libraries.

1. Scientifics Computing Libraries

**Pandas** offers data structure and tools for effective data manipulation and analysis. It provides facts, access to structured data. The primary instrument of Pandas is the two-dimensional table consisting of column and row labels, which are called a data frame. It is designed to provide easy indexing functionality.

The **NumPy** library uses arrays for its inputs and outputs. It can be extended to objects for matrices and with minor coding changes, developers can perform fast array processing.

**SciPy** includes functions for some advanced math problems such as Integrals, solving differential equations, optimization, as well as data visualization.

1. Visualization Libraries

Using data visualization methods is the best way to communicate with others, showing them meaningful results of analysis. These libraries enable you to create graphs, charts and maps. The **Matplotlib** package is the most well-known library for data visualization. It is great for making graphs and plots. The graphs are also highly customizable. Another high-level visualization library is **Seaborn**. It is based on Matplotlib. It's very easy to generate various plots such as heat maps, time series and violin plots.

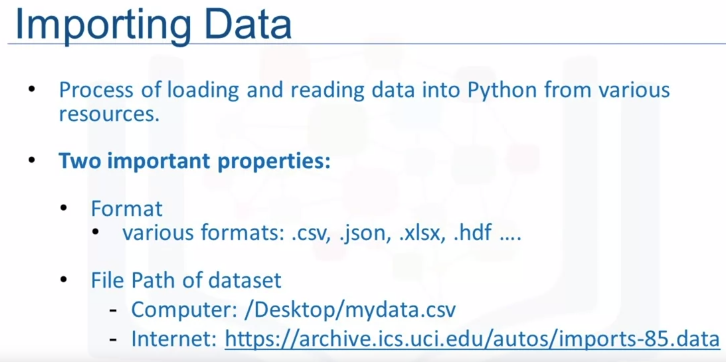
1. Algorithmic Libraries

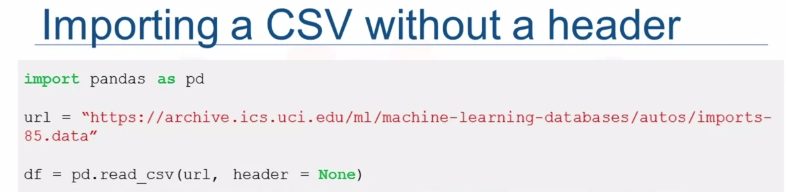
With machine learning algorithms, we're able to develop a model using our data set and obtain predictions. The algorithmic libraries tackle the machine learning tasks from basic to complex. Here we introduce two packages, the Scikit-learn library contains tools statistical modeling, including regression, classification, clustering, and so on. This library is built on NumPy, SciPy and Matplotlib. Stats models is also a Python module that allows users to explore data, estimate statistical models and perform statistical tests.

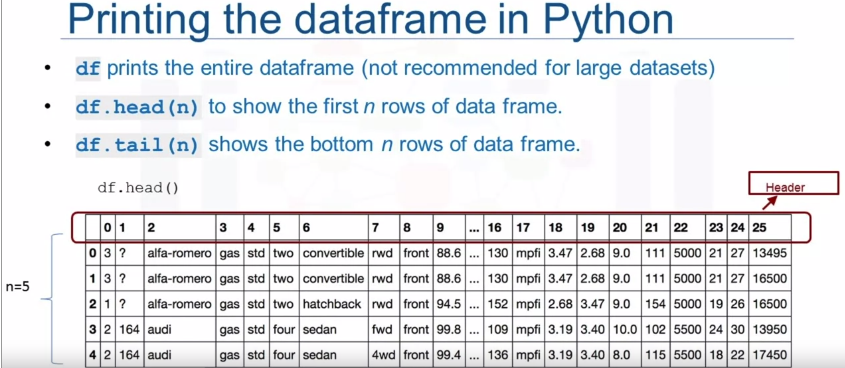
**Python Library describes:**

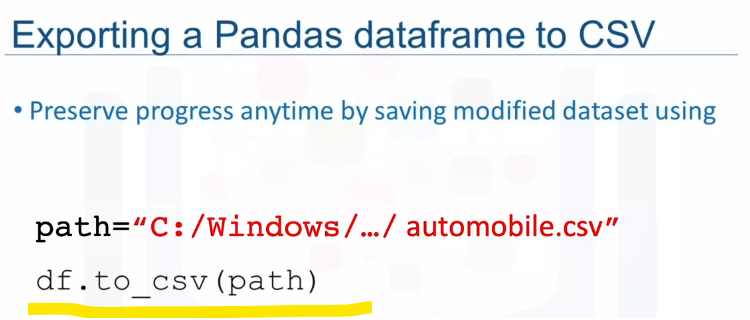
data structure and tools for effective data manipulation and analysis. It provides fast access to structured data. The primary instrument of Pandas is a two-dimensional table consisting of columns and rows labels which are called a Data Frame. It is designed to provide an easy indexing function.

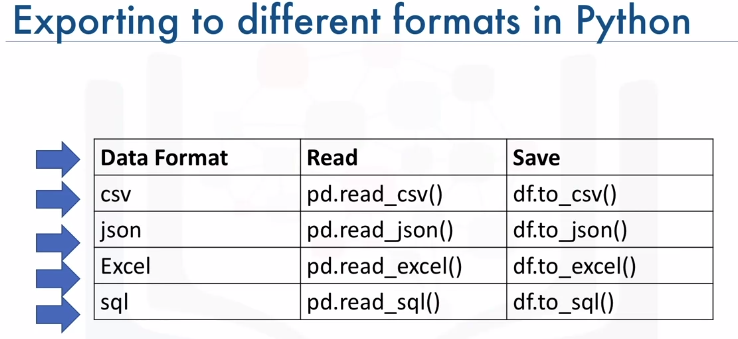
Importing and Exporting Data in Python:



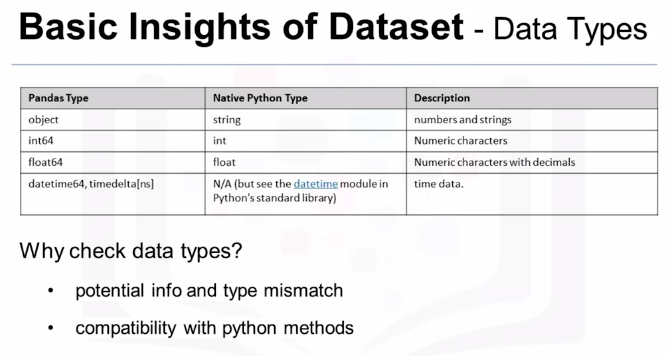


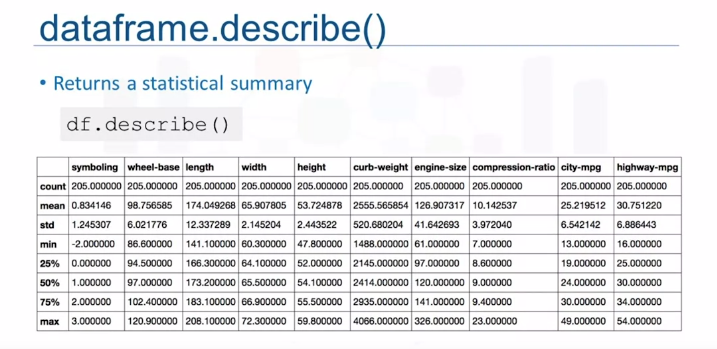






Getting Started Analyzing Data in Python:





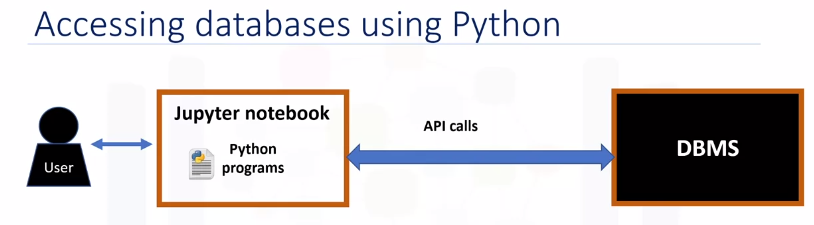
To get the quick statistics, we use the describe method. It returns the number of terms in the column as count, average column value as mean, column standard deviation as std, the maximum minimum values, as well as the boundary of each of the quartiles.



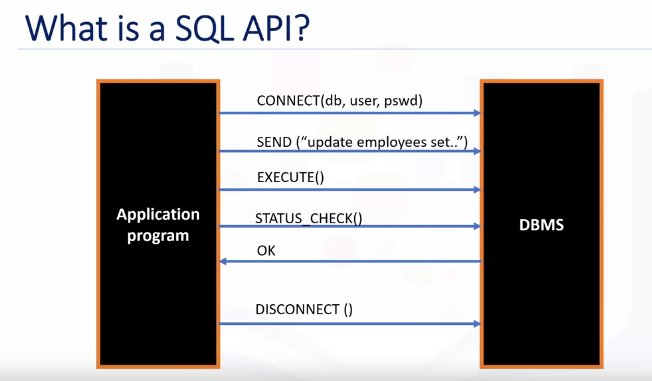
This function shows the top 30 rows and bottom 30 rows of the data frame. To view column names and data types.

Accessing Databases with Python:

Databases are powerful tools for data scientists. This is how a typical user accesses databases using Python code written on a Jupyter notebook, a web-based editor. There is a mechanism by which the Python program communicates with the DBMS.



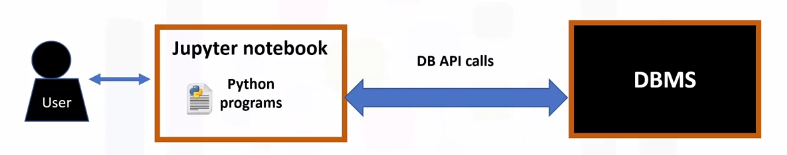
The Python code connects to the database using API calls. We will explain the basics of SQL APIs and Python DB APIs. An application programming interface is a set of functions that we can call to get access to some type of service.



The SQL API consists of library function calls as an application programming interface, API, for the DBMS. To pass SQL statements to the DBMS, an application program calls functions in the API, and it calls other functions to retrieve query results and status information from the DBMS. The basic operation of a typical SQL API is illustrated in the figure. The application program begins its database access with one or more API calls that connect the program to the DBMS. To send the SQL statement to the DBMS, the program builds the statement as a text string in a buffer and then makes an API call to pass the buffer contents to the DBMS. The application program makes API calls to check the status of its DBMS request and to handle errors. The application program ends its database access with an API call that disconnects it from the database.

**DB-API**:

DB-API is Python's standard API for accessing relational databases. It is a standard that allows us to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one. So, if you learn the DB-API functions, then we can apply that knowledge to use any database with Python.



The two main concepts in the Python DB-API are connection objects and query objects. We use connection objects to connect to a database and manage your transactions. Cursor objects are used to run queries. We open a cursor object and then run queries. The cursor works similar to a cursor in a text processing system where you scroll down in your result set and get our data into the application. Cursors are used to scan through the results of a database. Here are the methods used with connection objects.

**The connection methods:**

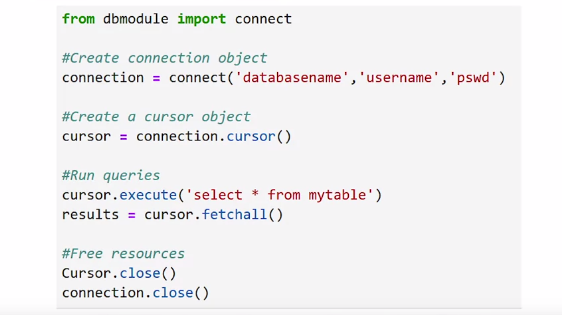
The **cursor()** method returns a new cursor object using the connection.

The **commit()** method is used to commit any pending transaction to the database.

The **rollback()** method causes the database to roll back to the start of any pending transaction.

The **close()** method is used to close a database connection.

Python application that uses the DB-API to query a database:



* First, we import our database module by using the connect API from that module.
* To open a connection to the database, we use the connection function and pass in the parameters that is, the database name, username, and password. The connect function returns connection object.
* After this, we create a cursor object on the connection object. The cursor is used to run queries and fetch results.
* After running the queries using the cursor, we also use the cursor to fetch the results of the query.
* Finally, when the system is done running the queries, it frees all resources by closing the connection. Remember that it is always important to close connections to avoid unused connections taking up resources.

Data Wrangling:

We will learn how to perform some fundamental data wrangling tasks that, together, form the pre-processing phase of data analysis. These tasks include handling missing values in data, formatting data to standardize it and make it consistent, normalizing data, grouping data values into bins, and converting categorical variables into numerical quantitative variables.

Learning Objectives

* Describe how to handle missing values
* Describe data formatting techniques
* Describe data normalization
* Demonstrate the use of binning
* Demonstrate the use of categorical variables

Pre-processing Data in Python:

Data preprocessing is a necessary step in data analysis. It is the process of converting or mapping data from one raw form into another format to make it ready for further analysis.

Data preprocessing is often called data cleaning or data wrangling.

**A missing value**: It occurs whenever a data entry is left empty.

Data format: data from different sources maybe in various formats, in different units, or in various conventions.

**Python Pandas** that can standardize the values into the same format, or unit, or convention,

**Normalization**: different columns of numerical data may have different ranges and direct comparison is often not meaningful. Normalization is away to bring all data into a similar range for more useful comparison.

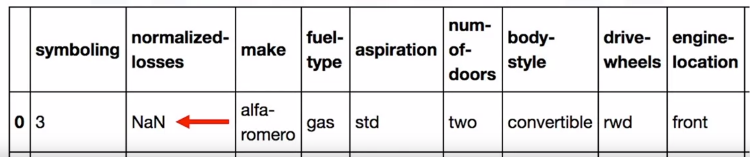
**Data binning**: binning creates bigger categories from a set of numerical values. It is particularly useful for comparison between groups of data.

**Categorical variables** converting into numeric variables to make statistical modeling easier.

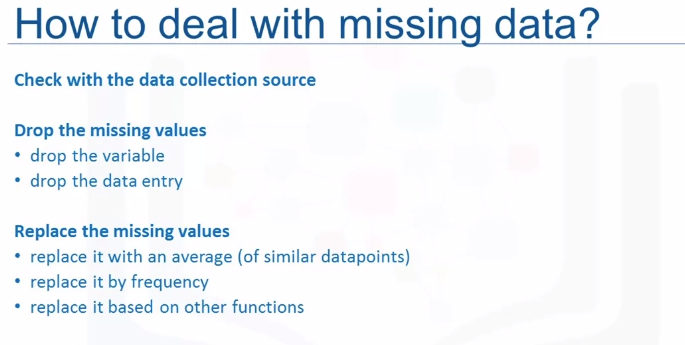
Dealing with Missing Values in Python:

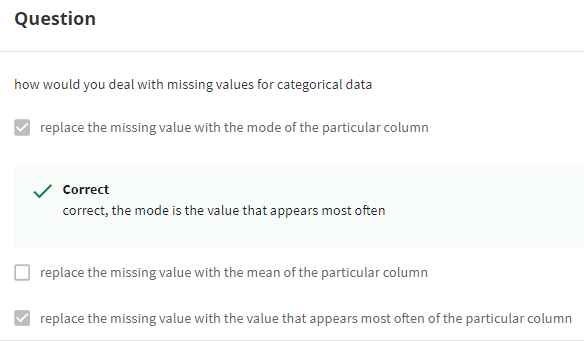
Missing value:

When no data value is stored for feature for a particular observation, we say this feature has a missing value. Usually missing value in data set appears as question mark and a zero or just a blank cell. In the example here, the normalized losses feature has a missing value which is represented with NaN.

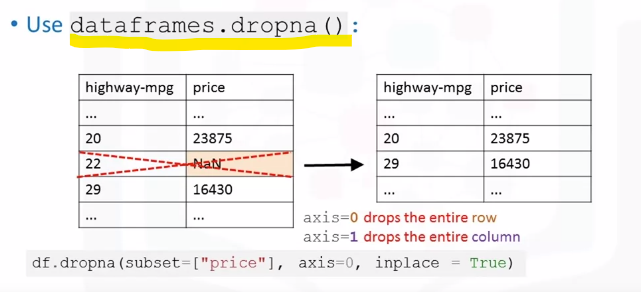


But how can we deal with missing data? There are many ways to deal with missing values and this is regardless of Python, R or whatever tool we use.





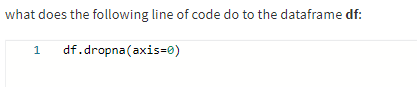
How to drop missing values in Python?

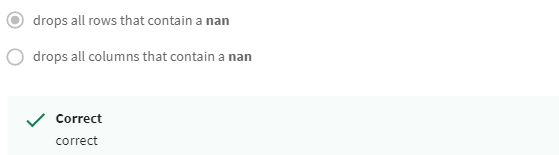


To remove data that contains missing values Panda's library has a built-in method called **dropna.**

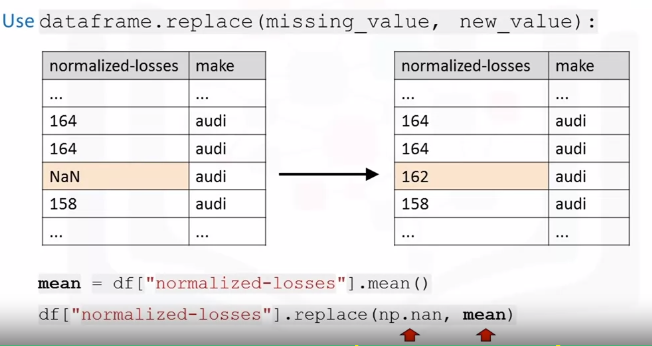
Essentially, with the dropna method, we can choose to drop rows or columns that contain missing values like NaN. So, we’ll need to specify access equal zero to drop the rows or access equals one to drop the columns that contain the missing values.

In this example, there is a missing value in the price column. Since the price of used cars is what we're trying to predict in our upcoming analysis, we have to remove the cars, the rows, that don't have a listed price. It can simply be done in one line of code using **dataframe.dropna**. Setting the argument in place to true, allows the modification to be done on the data set directly.





How to replace missing values in Python?



To replace missing values like NaNs with actual values, Pandas library has a built-in method called **replace** which can be used to fill in the missing values with the newly calculated values.

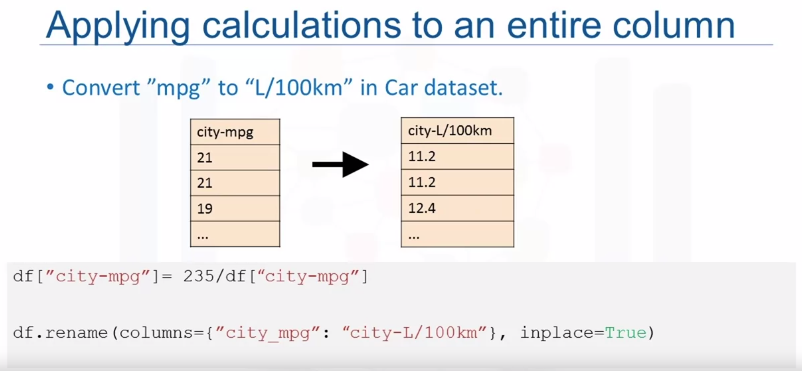
As an example, assume that we want to replace the missing values of the variable normalized losses by the mean value of the variable. Therefore, the missing value should be replaced by the **average of the entries within that column**.

In Python, first we calculate the mean of the column. Then we use the method replace to specify the value we would like to be replaced as the first parameter, in this case NaN. The second parameter is the value we would like to replace it with i.e the mean in this example. This is a fairly simplified way of replacing missing values.

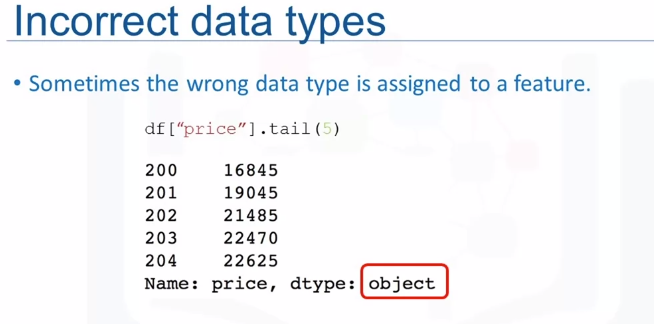
Data Formatting in Python:

Data is usually collected from different places by different people which may be stored in different formats. Data formatting means bringing data into a common standard of expression that allows users to make meaningful comparisons. As a part of dataset cleaning, data formatting ensures the data is consistent and easily understandable.

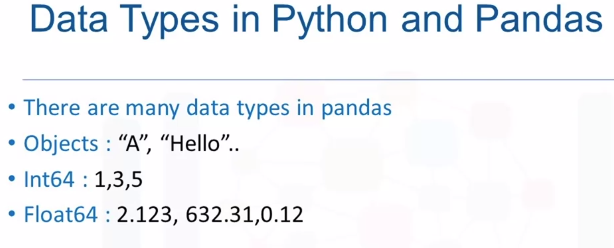


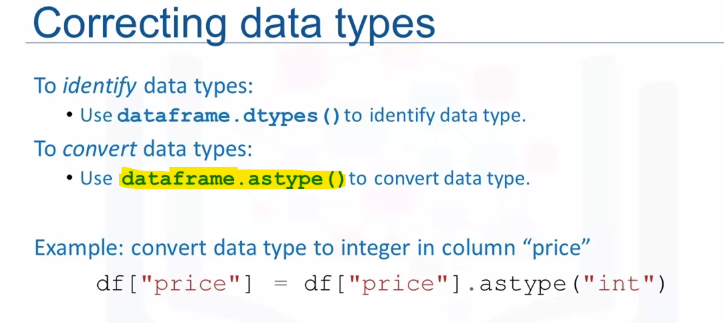


To transform miles per gallon to liters per 100 kilometers, we need to divide 235 by each value in the city-miles per gallon column. In Python, this can easily be done in one line of code. We take the column and set it to equal to 235, divide it by the entire column. In the second line of code, rename column name from city-miles per gallon to city-liters per 100 kilometers using the data frame rename method.



we noticed the assigned data type to the price feature is object. Although the expected data type should really be an integer or float type.





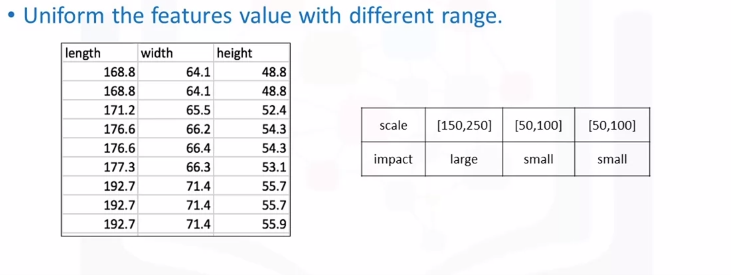
Or,



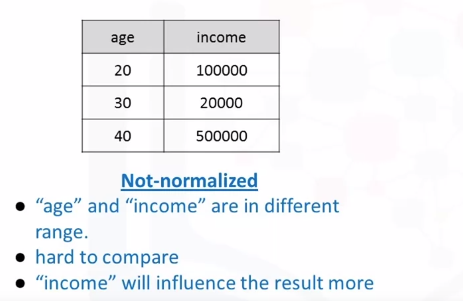


Data Normalization in Python:

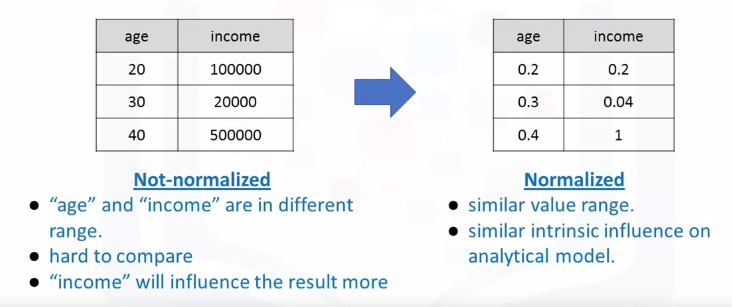
An important technique to understand in data pre-processing.



When we take a look at the used car data set, we notice in the data that the feature length ranges from 150-250, while feature width and height ranges from 50-100. We may want to normalize these variables so that the range of the values is consistent. This normalization can make some statistical analyses easier down the road. By making the ranges consistent between variables, normalization enables a fair comparison between the different features, making sure they have the same impact. It is also important for computational reasons.



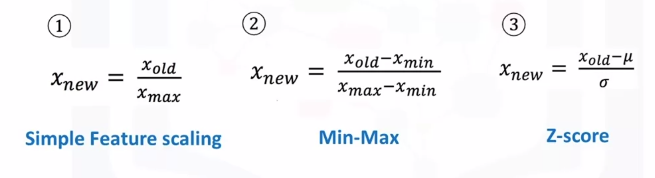
Here is another example that will help us to understand why normalization is important. Consider a data set containing two features, age and income. Where age ranges from 0-100, while income ranges from 0-20,000 and higher. Income is about 1,000 times larger than age and ranges from 20,000-500,000. So, these two features are in very different ranges. When we do further analysis, like linear regression for example, the attribute income will intrinsically influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor. So, the nature of the data biases the linear regression model to weigh income more heavily than age.



To avoid this, we can normalize these two variables into values that range from zero to one. Compare the two tables at the right. After normalization, both variables now have a similar influence on the models we will build later.

Methods of Normalizing data:

There are several techniques to normalize data but must used ways are

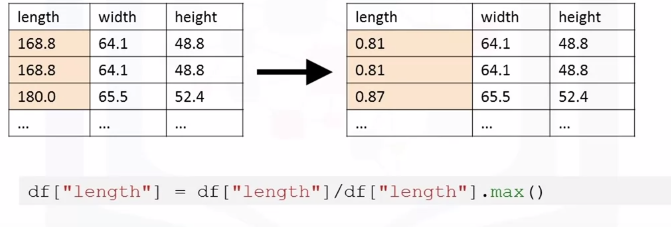


The first method called **simple feature scaling** just divides each value by the maximum value for that feature. This makes the new values range between zero and one.

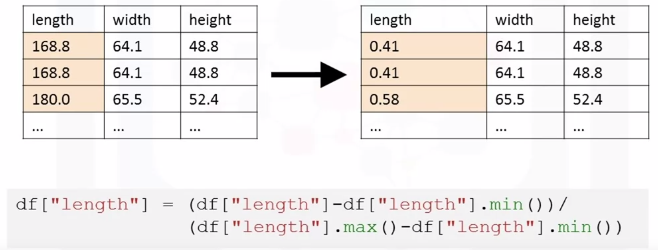
The second method called **min-max** takes each value X\_old subtract it from the minimum value of that feature, then divides by the range of that feature. Again, the resulting new values range between zero and one.

The third method is called **z-score or standard score**. In this formula for each value, we subtract the mu which is the average of the feature, and then divide by the standard deviation sigma. The resulting values hover around zero, and typically range between negative three and positive three **(-3 to 3)** but can be higher or lower.

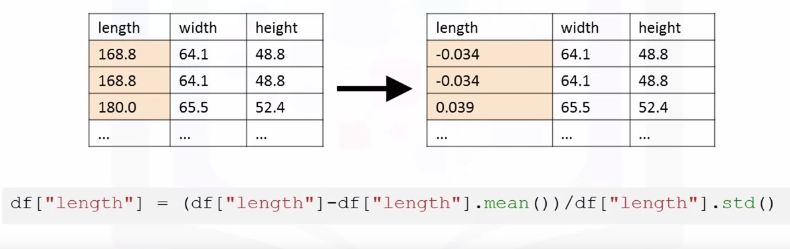
Following our earlier example, we can apply the normalization method on the length feature. First, we use the **simple feature scaling method**, where we divide it by the maximum value in the feature. Using the Pandas method max, this can be done in just one line of code.



The **min-max** method on the length feature. We subtract each value by the minimum of that column, then divide it by the range of that column. The max minus the min.



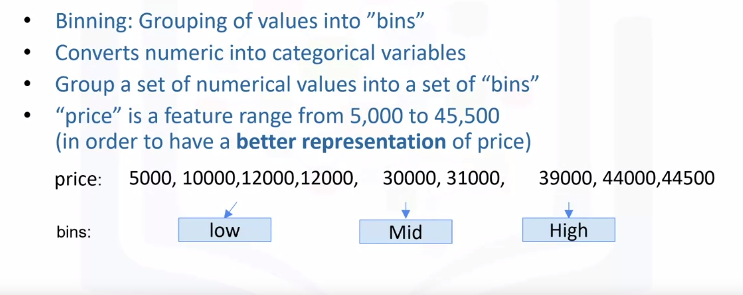
Finally, we apply the z-score method on length feature to normalize the values. Here we apply the mean and STD method on the length feature. Mean method will return the average value of the feature in the data set, and STD method will return the standard deviation of the features in the data set.



Binning in Python:

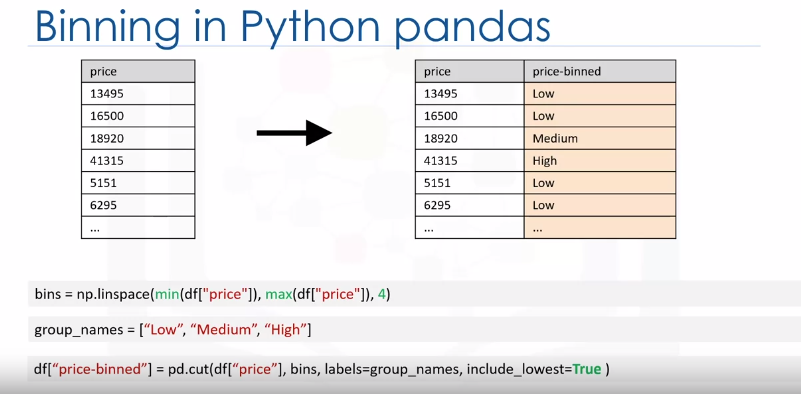
Binning is when we group values together into bins. For example, we can bin “age” into [0 to 5], [6 to 10], [11 to 15] and so on. Sometimes, binning can improve accuracy of the predictive models.

In addition, sometimes we use data binning to group a set of numerical values into a smaller number of bins to have a better understanding of the data distribution. As example, “price” here is an attribute range from 5,000 to 45,500. Using binning, we categorize the price into three bins: low price, medium price, and high prices.

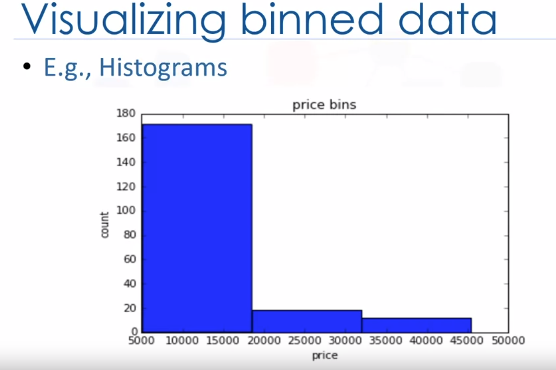


In the actual car dataset, ”price" is a numerical variable ranging from 5188 to 45400, it has 201 unique values. We can categorize them into 3 bins: low, medium, and high-priced cars. In Python we can easily implement the binning: We would like 3 bins of equal binwidth, so we need 4 numbers as dividers that are equal distance apart.

First, we use the NumPy function “**linspace**” to return the array “bins” that contains 4 equally spaced numbers over the specified interval of the price. We create a list “group\_names“ that contains the different bin names. We use the Pandas function”**cut**” to segment and sort the data values into bins.



You can then use histograms to visualize the distribution of the data after they’ve been divided into bins.



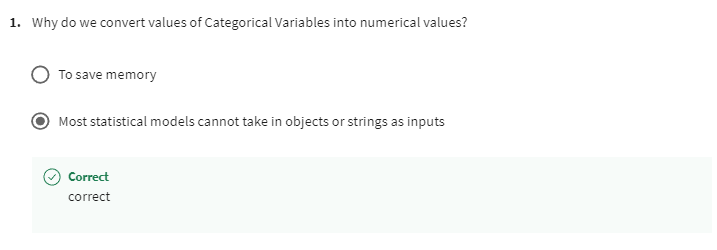
This is the histogram that we plotted based on the binning that we applied in the price feature. From the plot, it is clear that most cars have a low price, and only very few cars have high price.

Turning categorical variables into quantitative variables in Python:

In Pandas, we can use get\_dummies method to convert categorical variables to dummy variables. In Python, transforming categorical variables to dummy variables is simple. Following the example, pd.get\_dummies method gets the fuel type column and creates the data frame dummy\_variable\_1. The get\_dummies method automatically generates a list of numbers, each one corresponding to a particular category of the variable.



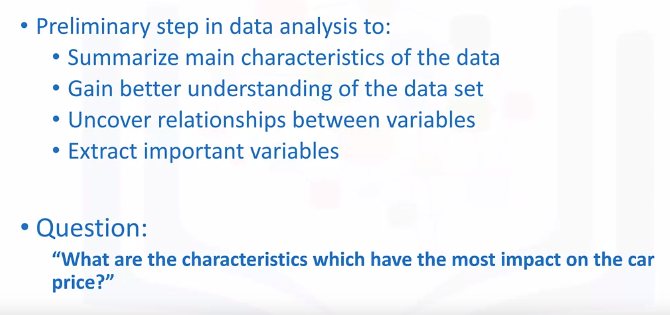
Q.



Exploratory Data Analysis:

Learning Objectives:

* Implement descriptive statistics
* Demonstrate the basics of grouping [GroupBy]
* Describe data correlation processes
* Describe why and how to apply the Chi-Squared test

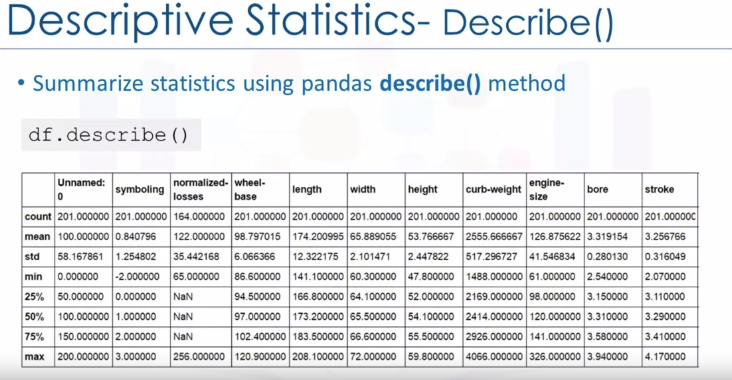


# Descriptive Statistics:

When we begin to analyze data, it's important to first explore the data before we spend time building complicated models. One easy way to do so, is to calculate some Descriptive Statistics for our data.

Descriptive statistical analysis helps to describe basic features of a data set, and obtains a short summary about the sample and measures of the data. Let's see couple different useful methods.

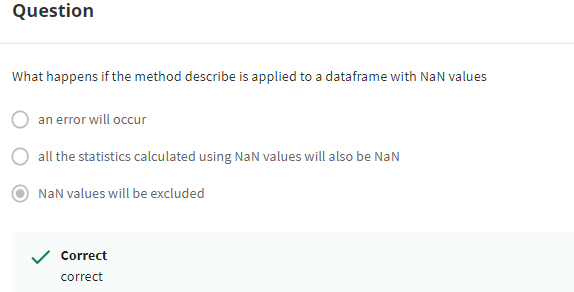
Descriptive Statistics - Describe ():



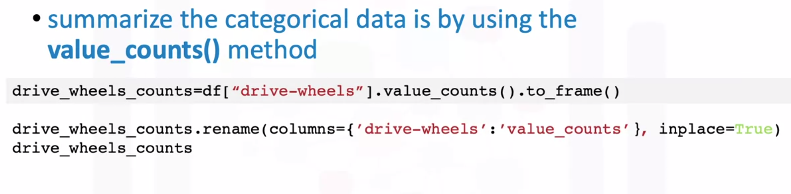
One way in which we can do this is by using the **describe** **function** in pandas.

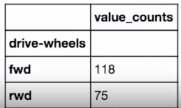
Using the describe function and applying it on our data frame, the describe function automatically computes basic statistics for all numerical variables.

It shows the mean, the total number of data points, the standard deviation, the quartiles and the extreme values. Any NAN values are automatically skipped in these statistics. This function will give us a clear idea of the distribution of different variables.



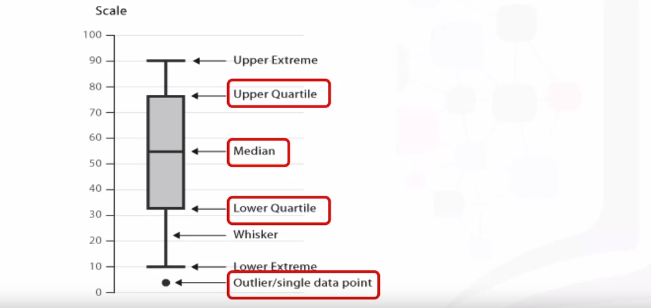
Descriptive Statistics - Value\_Counts():





These are variables that can be divided up into different categories or groups, and have discrete values. For example, in our data set we have the drive system as a categorical variable, which consists of the categories, forward wheel drive, rear wheel drive and four-wheel drive. One way we can summarize the categorical data, is by using the function value\_counts. We can change the name of the column to make it easier to read. We see that we have 118 cars in the front wheel drive category. 75 cars in the rear-wheel-drive category, and 8 cars in the four-wheel drive category.

Descriptive Statistics – Box Plots:



Box plots are a great way to visualize numeric data, since we can visualize the various distributions of the data. The main features that the box plot shows, are the median of the data, which represents where the middle data point is.

The upper quartile shows where the 75th percentile is. The lower quartile shows where the 25th percentile is.

The data between the upper and lower quartile represents the interquartile range. Next, we have the lower and upper extremes. These are calculated as 1.5 times the interquartile range, above the 75th percentile, and as 1.5 times the IQR below the 25th percentile.

Finally, box plots also display outliers as individual dots that occur outside the upper and lower extremes.

With box plots, we can easily spot outliers, and also see the distribution and skewness of the data.

**Box Plot Example**:



Box plots make it easy to compare between groups. In this example, using box plot we can see the distribution of different categories of the drive wheels feature over price feature. We can see that the distribution of price between the rear wheel drive, and the other categories are distinct. But the price for front wheel drive and four-wheel drive are almost indistinguishable.

Descriptive Statistics - Scatter Plot:

Often times we tend to see continuous variables in our data. These data points are numbers contained in some range. For example, in our data set price and engine size are continuous variables. What if we want to understand the relationship between engine size and price. Could engine size possibly predict the price of a car? One good way to visualize this is using a scatter plot. Each observation in the scatter plot is represented as a point. This plot shows the relationship between two variables. The predictor variable, is the variable that you are using to predict an outcome. In this case our predictor variable is the engine size. The target variable is the variable that you are trying to predict. In this case, our target variable is the price. Since this would be the outcome. In a scatter plot, we typically set the predictor variable on the x-axis or horizontal axis, and we set the target variable on the y-axis or vertical axis.



In this case, we will thus plot the engine size on the x-axis and the price on the y-axis.

We are using, the matplotlib functions scatter here, taking in x and y variable. Something to note is that it's always important to label your axes, and write a general plot title, so that we know what we’re looking at.

Now how is the variable engine size related to price? From the scatter plot, we see that as the engine size goes up, the price of the car also goes up. This is giving us an initial indication that there is a **positive linear relationship** between these two variables.