**The Problem**

But before we begin, talking about the problem, used car prices, we should first understand the importance of data analysis. As you know, data is collected everywhere around us. Whether it's collected manually by scientists or collected digitally, every time you click on a website, or your mobile device. But data does not mean information. Data analysis and, in essence, data science, helps us unlock the information and insights from raw data to answer our questions. So data analysis plays an important role by helping us to discover useful information from the data, answer questions, and even predict the future or the unknown.

So let's begin with our scenario. Let's say we have a friend named Tom. And Tom wants to sell his car. But the problem is he doesn't know how much he should sell his car for. Tom wants to sell his car for as much as he can. But he also wants to set the price reasonably, so someone would want to purchase it. So the price he sets should represent the value of the car.

How can we help Tom determine the best price for his car? Let's think like data scientists and clearly define some of his problems. For example, is there data on the prices of other cars and their characteristics? What features of cars affect their prices? Color? Brand? Does horsepower also effect the selling price, or perhaps something else? As a data analyst or data scientist, these are some of the questions we can start thinking about. To answer these questions, we're going to need some data.

**Understanding the Data**

The dataset used in this course is an open dataset by Jeffrey C. Schlemmer. This dataset is in CSV format, which separates each of the values with commas, making it very easy to import in most tools or applications. Each line represents a row in the dataset. In the hands-on lab for this module, you'll be able to download and use the CSV file. Do you notice anything different about the first row? Sometimes the first row is a header, which contains a column name for each of the 26 columns. But in this example, it's just another row of data. So, here's the documentation on what each of the 26 columns represent. There are a lot of columns and I'll just go through a few of the column names, but you can also check out the link at the bottom of the slide to go through the descriptions yourself. The first attribute, symboling, corresponds to the insurance risk level of a car. Cars are initially assigned a risk factor symbol associated with their price. Then, if an automobile is more risky, this symbol is adjusted by moving it up the scale. A value of plus three indicates that the auto is risky. Minus three, that is probably pretty safe. The second attribute, normalized-losses, is the relative average loss payment per insured vehicle year. This value is normalized for all autos within a particular size classification, two door small, station wagons, sports specialty, etc., and represents the average loss per car per year. The values range from 65 to 256. The other attributes are easy to understand. If you would like to check out more details, refer to the link at the bottom of the slide. Okay, after we understand the meaning of each feature, we'll notice that the 26 attribute is price. This is our target value or label in other words. This means price is the value that we want to predict from the dataset and the predictors should be all the other variables listed like symboling, normalized-losses, make, and so on. Thus, the goal of this project is to predict price in terms of other car features. Just a quick note. This dataset is actually from 1985. So, the car prices for the models may seem a little low. But just bear in mind that the goal of this exercise is to learn how to analyze the data.

**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 you 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. 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 provid 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 as listed on this slide, as well as data visualization.

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.

With machine learning algorithms, we're able to develop a model using our data set and obtain predictions. The algorithmic libraries tackles 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 Matplotib. Statsmodels is also a Python module that allows users to explore data, estimate statistical models and perform statistical tests.

**Importing and Exporting Data in Python**

Once we have our data in Python, then we can perform all the subsequent data analysis procedures we need. Data acquisition is a process of loading and reading data into notebook from various sources. To read any data using Python's pandas package, there are two important factors to consider, format and file path. Format is the way data is encoded. We can usually tell different encoding schemes by looking at the ending of the file name. Some common encodings are: CSV, JSON, XLSX, HDF and so forth. The path tells us where the data is stored. Usually, it is stored either on the computer we are using or online on the internet. In our case, we found a dataset of used cars which was obtained from the web address shown on the slide. When Jerry entered the web address in his web browser, he saw something like this. Each row is one datapoint. A large number of properties are associated with each datapoint. Because the properties are separated from each other by commas, we can guess the data format is CSV, which stands for comma separated values. At this point, these are just numbers and don't mean much to humans, but once we read in this data we can try to make more sense out of it. In pandas, the read\_CSV method can read in files with columns separated by commas into a pandas data frame. Reading data in pandas can be done quickly in three lines. First, import pandas, then define a variable with a file path and then use the read\_ CSV method to import the data. However, read\_CSV assumes the data contains a header. Our data on used cars has no column headers. So, we need to specify read\_CSV to not assign headers by setting header to none. After reading the dataset, it is a good idea to look at the data frame to get a better intuition and to ensure that everything occurred the way you expected. Since printing the entire dataset may take up too much time and resources to save time, we can just use dataframe.head to show the first n rows of the data frame. Similarly, dataframe.tail shows the bottom end rows of data frame. Here, we printed out the first five rows of data. It seems that the dataset was read successfully. We can see that pandas automatically set the column header as a list of integers because we set header equals none when we read the data. It is difficult to work with the data frame without having meaningful column names. However, we can assign column names in pandas. In our present case, it turned out that we have the column names in a separate file online. We first put the column names in a list called headers, then we set df.columns equals headers to replace the default integer headers by the list. If we use the head method introduced in the last slide to check the dataset, we see the correct headers inserted at the top of each column. At some point in time, after you've done operations on your dataframe you may want to export your pandas dataframe to a new CSV file. You can do this using the method to\_CSV. To do this, specify the file path which includes the file name that you want to write to. For example, if you would like to save dataframe df as automobile.CSV to your own computer, you can use the syntax df.to\_CSV.

For this course, we will only read and save CSV files. However, pandas also supports importing and exporting of most data file types with different dataset formats. The code syntax for reading and saving other data formats is very similar to read or save CSV file. Each column shows a different method to read and save files into a different format.

**Getting Started Analyzing Data in Python**

At this point, we assume that the data has been loaded. It's time for us to explore the dataset. Pandas has several built-in methods that can be used to understand the datatype or features or to look at the distribution of data within the dataset. Using these methods, gives an overview of the dataset and also point out potential issues such as the wrong data type of features which may need to be resolved later on. Data has a variety of types. The main types stored in Pandas' objects are object, float, Int, and datetime. The data type names are somewhat different from those in native Python. This table shows the differences and similarities between them. Some are very similar such as the numeric data types, int and float. The object pandas type function's similar to string in Python, save for the change in name. While the datetime Pandas type, is a very useful type for handling time series data. There are two reasons to check data types in a dataset. Pandas automatically assigns types based on the encoding it detects from the original data table. For a number of reasons, this assignment may be incorrect. For example, it should be awkward if the car price column which we should expect to contain continuous numeric numbers, is assigned the data type of object. It would be more natural for it to have the float type. Jerry may need to manually change the data type to float. The second reason, is that allows an experienced data scientists to see which Python functions can be applied to a specific column. For example, some math functions can only be applied to numerical data. If these functions are applied to non-numerical data an error may result. When the dtype method is applied to the data set, the data type of each column is returned in a series. A good data scientists intuition tells us that most of the data types make sense. They make of cars for example are names. So, this information should be of type object. The last one on the list could be an issue. As bore is a dimension of an engine, we should expect a numerical data type to be used. Instead, the object type is used. In later sections, Jerry will have to correct these type mismatches. Now, we would like to check the statistical summary of each column to learn about the distribution of data in each column. The statistical metrics can tell the data scientist if there are mathematical issues that may exist such as extreme outliers and large deviations. The data scientists may have to address these issues later. 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. By default, the dataframe.describe functions skips rows and columns that do not contain numbers. It is possible to make the describe method worked for object type columns as well. To enable a summary of all the columns, we could add an argument. Include equals all inside the describe function bracket. Now, the outcome shows the summary of all the 26 columns, including object typed attributes. We see that for the object type columns, a different set of statistics is evaluated, like unique, top, and frequency. Unique is the number of distinct objects in the column. Top is most frequently occurring object, and freq is the number of times the top object appears in the column. Some values in the table are shown here as NaN which stands for not a number. This is because that particular statistical metric cannot be calculated for that specific column data type. Another method you can use to check your dataset, is the dataframe.info function. This function shows the top 30 rows and bottom 30 rows of the data frame.

**Accessing Databases with Python**

Databases are powerful tools for data scientists. After completing this module, you'll be able to explain the basic concepts related to using Python to connect to databases. 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 you 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 is Python's standard API for accessing relational databases. It is a standard that allows you 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 you 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. You use connection objects to connect to a database and manage your transactions. Cursor objects are used to run queries. You 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 your data into the application. Cursors are used to scan through the results of a database. Here are the methods used with connection objects. 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. Let's walk through a Python application that uses the DB-API to query a database. First, you import your database module by using the connect API from that module. To open a connection to the database, you 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, you 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.