Experiment No.: 3

Title: Write a Python program to import data from Comma Separated Files (CSV) file, clean data, and export data in CSV file.

Objectives:

1. To learn how to Import, Clean and Export data.

Theory:

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 (file) path tells us where the data is stored. Usually it is stored either on the computer we are using, or online on the internet.
A large number of properties are associated with each data point. Because the properties are separated from each other by commas, we can guess the data format is CAV, 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 DataFrame. Reading data in pandas can be done quickly in three lines.

1. First, import pandas.
2. Then define a variable with the file path.
3. And then use the read_csv method to import the data.

However, "read_csv" assumes that the data contains a header. If data set 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 dataframe 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(n) shows the bottom n rows of data frame. Here, we printed out the first 5 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=None when we read the data. It is difficult to work with the DataFrame without having meaningful column names, however, we can assign column names in pandas. Column names can be in a separate file. We first put the Column names in a list called headers. Then, set df.columns equals headers to replace the default integer headers by the list. If we use the

head() method introduced 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 pandasdataframe to a new CSV file. You can do this using the method, ”to_csv()". To do this, specify the file path (which includes the filename) that you want to write to. For example, if you would like to save the dataframe "df" as "automobile.csv" to your own computer, you can use the syntax: df.to_csv ("automobile.csv"). This shows how to read and save csv files. However, pandas also supports importing and exporting of most data filetypes with different dataset formats. The code syntax for reading and saving other data formats is very similar to read or save csv file.

Data pre-processing 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 pre-processing is also often called “data cleaning” or “data wrangling”, and there are likely other terms.

Here are the topics that we'll be covering in this experiment: First, we'll show you how to identify and handle missing values. A “missing value” condition occurs whenever a data entry is left empty. Then, we'll cover data formats. Data from different sources may be in various formats, in different units or in various conventions. We will introduce some methods in Python pandas that can standardize the values into the same format, or unit, or convention.

After that, we'll cover data normalization. Different columns of numerical data may have very different ranges, and direct comparison is often not meaningful. Normalization is a way to bring all data into a similar range, for more useful comparison. Specifically, we'll focus on the techniques of centering and scaling. And then, we'll introduce data binning. Binning creates bigger categories from a set of numerical values. It is particularly useful for comparison between groups of data. And lastly, we'll talk about categorical variables and show you how to convert categorical values into numeric variables to make statistical modeling easier.

In Python, we usually perform operations along columns; each row of the column represents a sample, i.e., a different used car in the database. You access a column by specifying the name of the column. For example, you can access "symbolling” and “body-style”; each of these columns is a pandas series.

When nodata value is stored for a feature for a particular observation, we say this feature has a “missing value”. Usually “missing value: in dataset” appears as “?”, “N/A”, 0 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 you deal with missing data?

There are many ways to deal with missing values, and this is regardless of Python, R, or whatever tool you use. Of course, each situation is different and should be judged differently. However, these are the typical options you can consider:

• The first is to check if the person or group that collected the data can go back and find what the actual value should be. Another possibility is just to remove the data where that missing value is found.

• When you drop data, you can either drop the whole variable or just the single data

entry with the missing value. If you don't have a lot of observations with missing data, usually dropping the particular entry is the best. If you're removing data, you want to look to do something that has the least amount of impact. Replacing data is better, since no data is wasted. However, it is less accurate since we need to replace missing data with a guess of what the data should be. One standard replacement technique is to replace missing values by the average value of the entire variable. As an example, suppose we have some entries that have missing values for the 'normalized-losses' column, and the column average for entries with data is 4,500. While there is no way for us to get an accurate guess of what the missing values under the 'normalized-losses' column should have been, you can approximate their values using the average value of the column, 4,500. But what if the values cannot be averaged, as with categorical variables? For a variable like 'fuel-type', there isn't an "average" fuel type, since the variable values are not numbers. In this case, one possibility is to try using the mode –the most common. Finally, sometimes we may find another way to guess the missing data. This is usually because the data gatherer knows something additional about the missing data. For example, he may know that the missing values tend to be old cars, and the normalized losses of old cars are significantly higher than the average vehicle. And of course, finally, in some cases, you may simply want to leave the missing data
as missing data. For one reason or another, it may be useful to keep that observation, even if some features are missing. Now let's go into how to drop missing values or replace missing values in Python.

To remove data that contains missing values, pandas library has a built-in method called 'dropna'. Essentially, with the dropna method, you can choose to drop rows or columns that contain missing values, like NaN. So you'll need to specify "axis=0" to drop the rows, or "axis=1" 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'd 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 "inplace" to "true" allows the modification to be done on the dataset directly. "Inplace=True" just writes the result back into the dataframe. This is equivalent to this line of code. Don't forget that this line of code does not change the dataframe, but is a good way to make sure that you are performing the correct operation. To modify the dataframe, you have to set the parameter "inplace" equal to true.

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. There are of course other techniques, such as replacing missing values for the average of the group, instead of the entire dataset. So we've gone through two ways in Python to deal with missing data. We learned to drop problematic rows or columns

containing missing values. And then we learned how to replace missing values with other values. But don’t forget the other ways to deal with missing data: you can always check for a higher quality dataset or source. Or, in some cases, you may want to leave the missing data as missing data.

Key concepts: Data Cleaning, Data Pre-processing

Algorithm:

- Read data set in Pandas Dataframe
- Change the default header in DataFrame.
- Replace missing values with constant or values specified
- Replace missing values with mean
- Replace missing values with mode
- Replace the missing values with **imputed** values based on the other characteristics of the record
- Export pre-processed dataset in CSV or other formats