**😊 EDA Data Acquisition, Merging, Duplicate Values and Outlier Analysis**

1. Now we are going to work on some other data sets. So, from the folder that you downloaded in the last lab, you need to upload three files other than data.csv as you can see below.



**😄 Data Acquisition**

1. So, here we start by importing some important libraries. Now we are going to load our data.

A screenshot of a computer

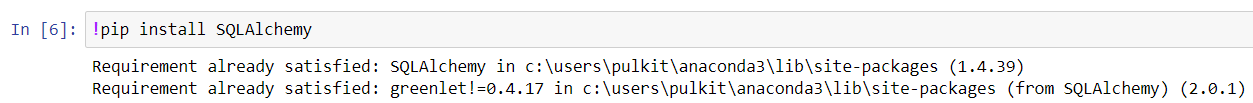
Description automatically generated

1. Below you can see that our data has been loaded for Cloud Bello customers and local Bello sales.

A screenshot of a computer

Description automatically generated

1. The third file that we have is a database file for it we need to install the SQL Alchemy library. SQL Alchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.



1. So, by running this command we were able to read the database file.

**from sqlalchemy import create\_engine**

**engine=create\_engine('sqlite:///db\_bello\_customers.db')**

A white rectangular sign with black and red text

Description automatically generated

1. To connect with this database file, we need to run this command.

**connect = engine.connect()**

**type(connect)**

A close-up of a computer screen

Description automatically generated

1. Now to know the table name inside this database we can make use of the command given below.

**from sqlalchemy import inspect**

**inspector = inspect(engine)**

**print(f"table name is {inspector.get\_table\_names()}")**

A white rectangular sign with black text

Description automatically generated

1. As we have the table name, now we can see the data inside the table using the command below.

**on\_premise\_database = pd.read\_sql\_table('Payments',connect)**

**on\_premise\_database.head()**

A screenshot of a computer

Description automatically generated

1. Now that we have the data from all three files, we are going to merge the data. So, it’ll be easier for us to work on single data instead of working on three different data sets.
2. By running these commands, we got the columns of all three tables.

**😄 Merging**

A screenshot of a computer

Description automatically generated

1. Now we will merge all these data frames by considering that they are from a single source. But if you know that the data frames are from different sources then first you need to perform EDA and then move ahead.
2. So, by running the below commands we have the concatenated data frame.

A screenshot of a computer screen

Description automatically generated

1. Run this command to get the raw data based on your preferred columns as you can see below. We’ll be using this data and move forward. So, to save this data we run this command.

A screenshot of a computer

Description automatically generated



1. And if you go to the EDA Lab folder you will see that you have a new CSV file.

A screenshot of a computer

Description automatically generated

**😄 Duplicate Entries**

1. The command below will return you the PANDAS series filled with Boolean values.

A screenshot of a computer

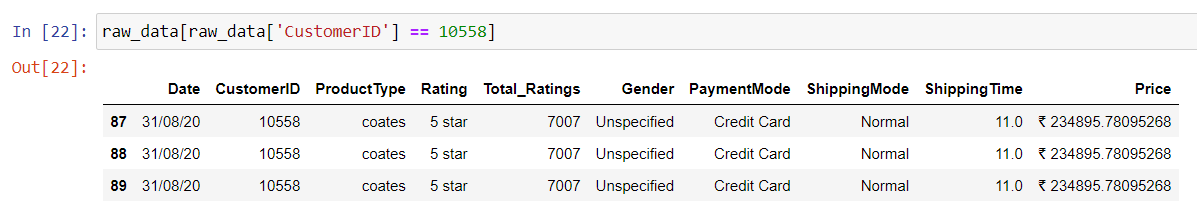
Description automatically generated

1. Now we can use these Boolean values, and mention them inside the indexing operator, and that would help us to return the rows, that have been identified as duplicates.
2. Below you can see that we have the duplicate values from the dataset.

A screenshot of a computer

Description automatically generated

1. If we filter out the by Customer ID we can see that one True value and other as a replica of it.



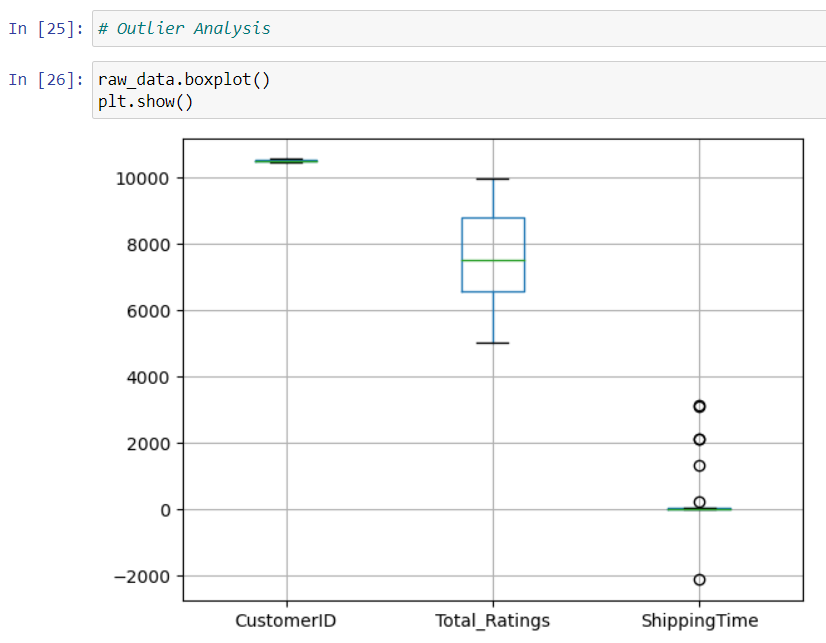
1. By running these, we have dropped the duplicated items from the dataset and we can see that there 0 duplicated items now.

A screenshot of a computer code

Description automatically generated

1. Now we will see the analysis of outliers.
2. So, the outlier is someone or some value where we would observe a slightly different value than the existing values.
3. By executing the below command, we can see that on the shipping time, we got the outliers.

**😄 Outlier Analysis**



1. Now we created a boxplot just for the outliers and you see them here in the snapshot below.
2. Well, this plot is called a box plot. This is also known as a box and a whisker plot. Now, this box and whisker plot is a graphical method to display the variation in each set of datasets, and that is the usage of this box plot.

A graph with lines and dots

Description automatically generated

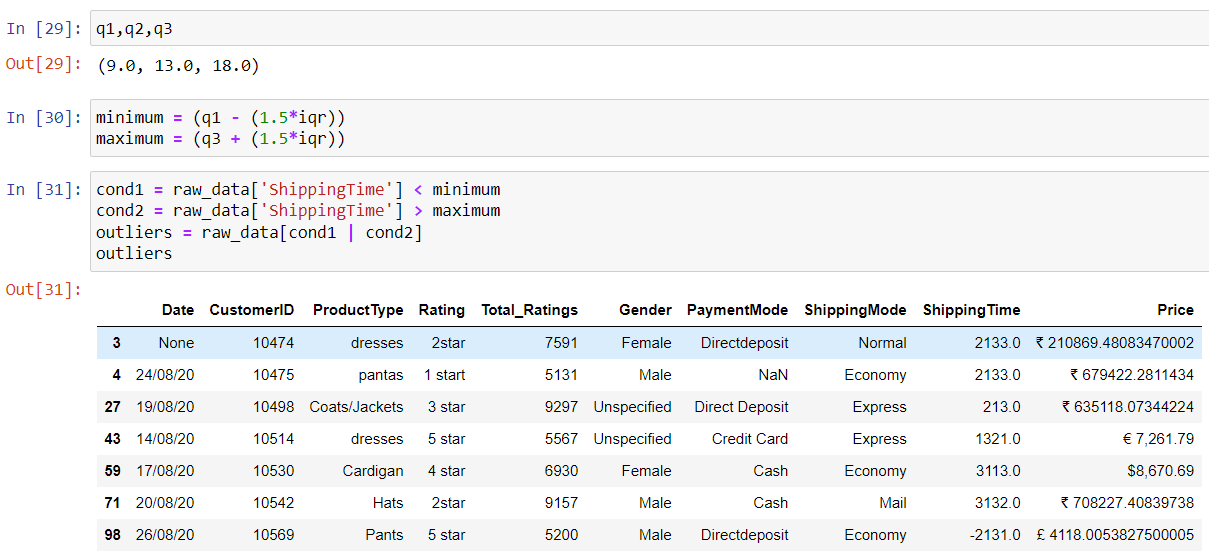
1. By using the command below, we get the interquartile value.
   * **q1: This computes the first quartile (25th percentile) of the ShippingTime column in the raw\_data DataFrame.**
   * **q2: This calculates the median (50th percentile) of the ShippingTime.**
   * **q3: This finds the third quartile (75th percentile) of the ShippingTime.**

A screen shot of a computer code

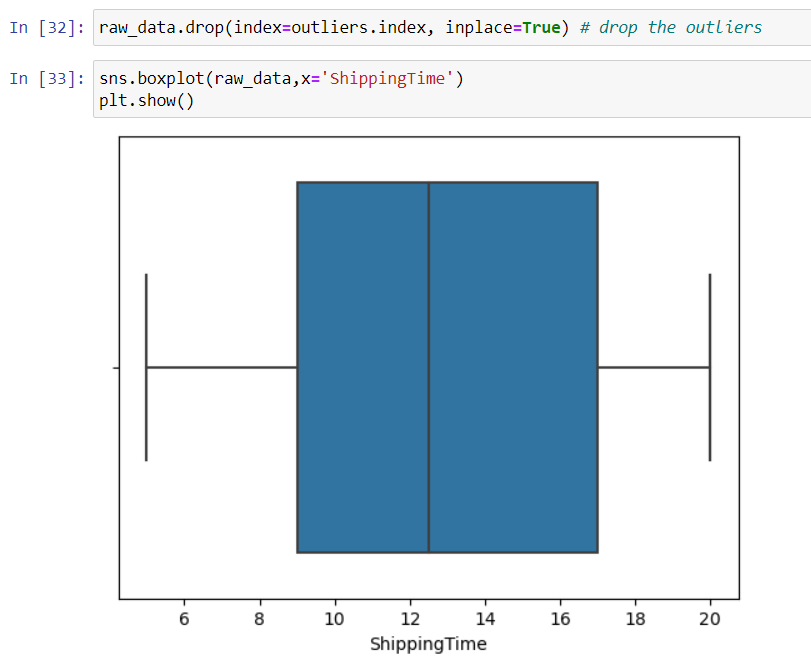
Description automatically generated

1. Now run the below commands to get the output. And below you can see that IDs which have been identified as the outliers.

* **These calculations set the thresholds for identifying outliers. Data points below minimum or above maximum are considered outliers. The factor of 1.5 is a common choice in outlier detection.**
* **cond1: This condition identifies data points in ShippingTime that are less than the minimum threshold.**
* **cond2: This condition identifies data points in ShippingTime that are greater than the maximum threshold.**
* **This line uses a logical OR (|) to combine the two conditions. It creates a new DataFrame called outliers, which includes only those rows from raw\_data where the ShippingTime is either less than minimum or greater than maximum.**

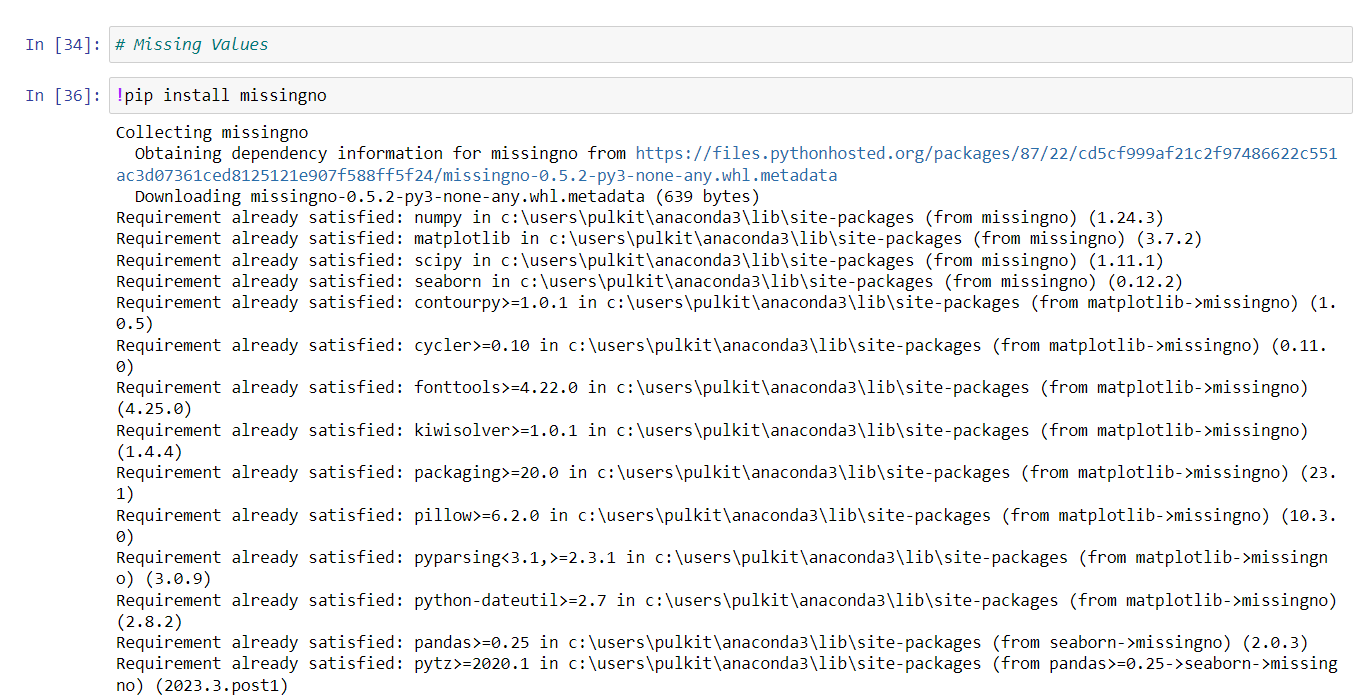


1. By using the command below first we dropped the outliers and then we ran the boxplot command to see the data and all the outliers have been cleared.



**😄 Missing Values**

1. Now we are going to work on the missing values in our dataset for that we use dropping or imputation.
2. So, first we are going to install the missing number library using pip command.

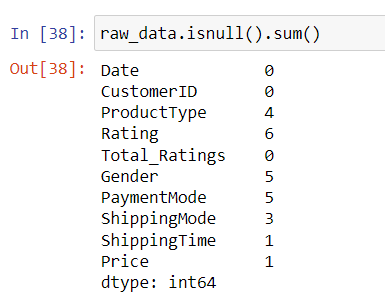


1. Then we will use the below command to import the missing number library and then have the visual representation of our data.

A screenshot of a computer screen

Description automatically generated

1. Use the below command to look at the missing numbers in the column format.



1. Now we will learn about imputation, it is about filling the missing values with a statistically computed value depending on the data type that we are dealing with.
2. So, if we say raw\_data.info, this is going to display the information about the data type of the individual column in the data set.

A screenshot of a computer

Description automatically generated

1. By using the below command, we have the unique values.

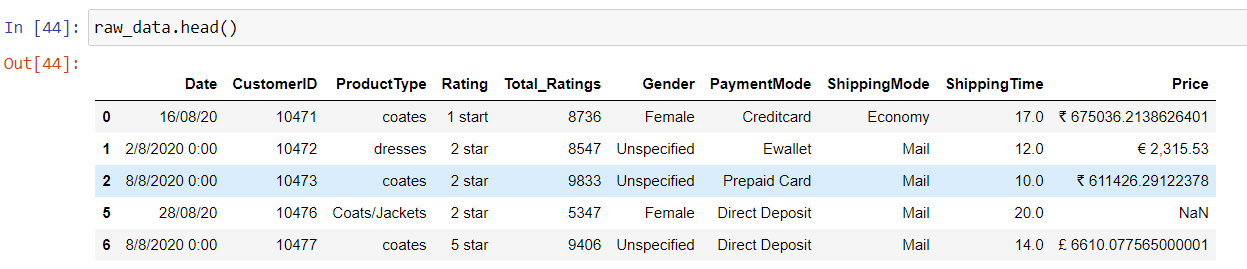
A white box with black text

Description automatically generated

1. We ran the below command to clear all the missing values but we didn’t clear the price one because it is being treated as string and we need to transform it before clearing the missing values.

A screen shot of a computer

Description automatically generated



**😄 Fixing the Errors**

1. So below we have the data in raw and if you look at the unique values in just Product type columns you will see that there are some typos and extra things in this categorical data.

A screenshot of a computer

Description automatically generated

1. To fix the errors you can use these commands and you can see that in the output 50 all the errors have been fixed.



1. Now we need to apply the same things on every column.

A screen shot of a computer program

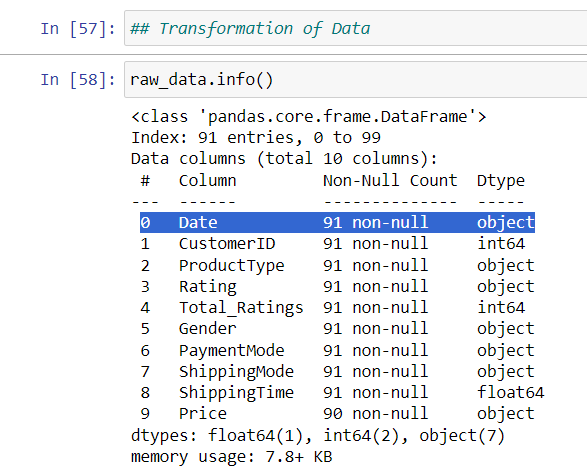
Description automatically generated

A screenshot of a computer code

Description automatically generated

**😄 Data Transformation**

1. Now we’ll be working on transforming the data. Below you can see that we ran the command to get the information of our dataset. Here you can see that the Data column is of Object data type which is not suitable because whenever we want to work with this column it would be best if it is in Data Time format.



1. Now we will convert this into Data Time format. Using the below commands we have converted it into the suitable format.

A screenshot of a computer

Description automatically generated

1. If you look at the Price column, you can see that it is also in the string format and we also need to take care of the available currencies; here we will convert the currency into INR.

A screenshot of a computer

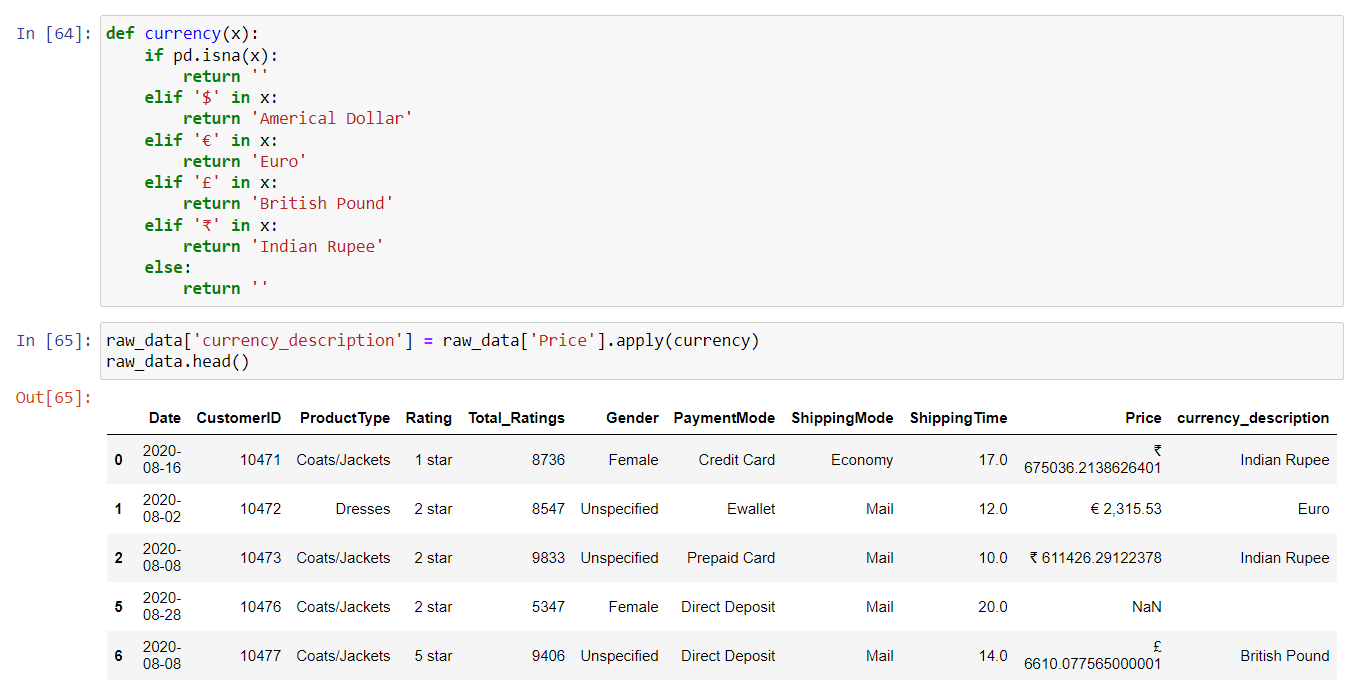
Description automatically generated

A screenshot of a computer

Description automatically generated

1. Run the code shown below and you will see that we have created a new column with the name currency description.

The provided code defines a function currency that checks the Price column in a dataset for specific currency symbols (like $, €, £, ₹) and returns the corresponding currency name (e.g., "American Dollar", "Euro", etc.). The function is applied to each value in the Price column, and the results are stored in a new column called currency\_description. Missing or unrecognized currency values are handled by returning an empty string. The updated dataset is then displayed using raw\_data.head().



A close up of a computer screen

Description automatically generated

1. Here In this code, we are cleaning the Price column by removing currency symbols and commas from the price values. Here's what happens step by step:

**Remove Dollar Sign ($)**: The line raw\_data['Price'] = raw\_data['Price'].str.replace('$', "") removes the dollar sign from all price entries in the Price column.

**Remove Pound Sign (£)**: Similarly, the next line removes the British pound symbol (£).

**Remove Euro Sign (€)**: The third line removes the euro symbol.

**Remove Rupee Sign (₹)**: This line removes the Indian rupee symbol.

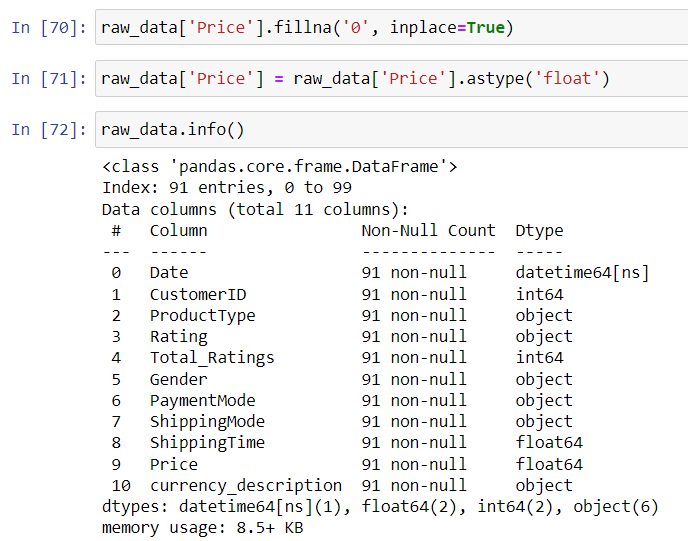
**Remove Commas (,)**: The final line removes any commas in the price values, which might be used for thousands of separators (e.g., "1,000" becomes "1000").

1. After this, the Price column will contain clean numeric values, without currency symbols or commas, making it easier to work with the data.

A screenshot of a computer

Description automatically generated

1. Below you can see that first we converted the missing values with Zero then we changed the data type into a floating number.



1. We need to check the currency for each row in the dataset, and based on the identified currency, we will append the corresponding value.
2. Here the code converts prices to Indian Rupees using the appropriate exchange rate for each row's currency, and if no currency is found, it keeps the original price.
3. Then we ran the price\_inr to check the conversion and you can see that everything has been converted to INR.

A screenshot of a computer code

Description automatically generated

1. Below you can see that we have a new column in which the prices have been converted to INR.

A screenshot of a computer

Description automatically generated

1. Now we’ll check whether any outliers are present in this data or not. Below you can see that we have some outliers in the given data.

A screen shot of a graph

Description automatically generated

1. So, to remove Outliers we will use the Interquartile range (IQR) and below you can see that we have removed the outliers from our data.

A screenshot of a computer

Description automatically generated

A screenshot of a graph

Description automatically generated

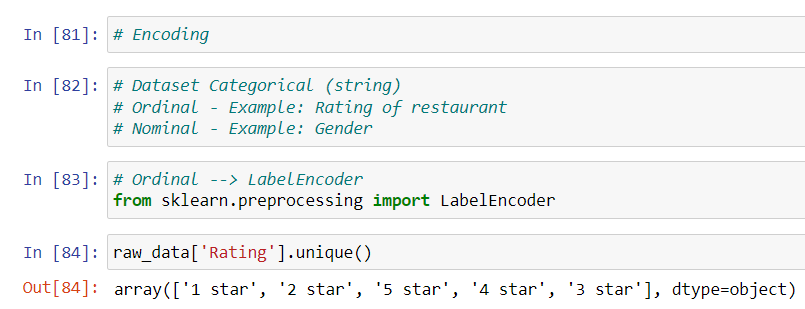
**😄 Encoding**

1. Encoding is nothing but finding a way to represent non-numerical data in a numerical format.
2. The command from sklearn.preprocessing import LabelEncoder imports the LabelEncoder class from the sklearn.preprocessing module in the scikit-learn library.

**What does LabelEncoder do?**  
LabelEncoder is used to convert categorical labels (like names or categories) into numerical values (integers). For example, it can transform categories like ['cat', 'dog', 'fish'] into [0, 1, 2], making the data easier to use in machine learning models.

In summary, this command allows you to access the LabelEncoder tool for converting categorical data into numeric form.

1. Then we get the unique values from the Rating column.



1. In this code, we use LabelEncoder to convert the values in the Rating column (which are categorical) into numerical values. The transformed ratings are stored in a new column called rating\_encoded, so we can see both the original ratings and their corresponding numeric codes side by side.
2. **Okay, so this is a methodology that we use when we have ordinal data.**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Well, when we have nominal data at that time, we make use of one hot encoder.
2. First, we start by importing one hot encoder then we apply it on the Shipping mode and get the categories.

A screenshot of a computer program

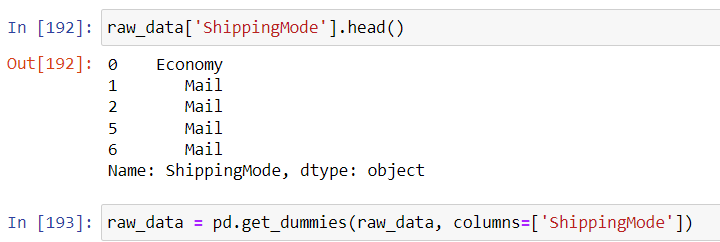
Description automatically generated

1. In this code, we use one-hot encoding to convert the categorical values in the ShippingMode column into a numerical format, where each unique category is represented as a separate column. The resulting encoded data is then stored in a new DataFrame (df1), making it easier to analyze and use in machine learning models.

A screenshot of a computer

Description automatically generated

1. The first command displays the first few entries in the ShippingMode column of the raw\_data DataFrame, allowing you to see the different shipping methods. The second command converts the ShippingMode column into multiple columns of binary values (0s and 1s) for each unique shipping method, making it easier to use in data analysis or machine learning models.



**😄 Scaling of Numerical Values**

1. The **Standard Scaler** transforms data into a format where the average (mean) is zero and the standard deviation is one, making it easier to compare different datasets. **Normalizer** adjusts the data to have a length of one, while the **Min-Max Scaler** scales the data to fit within a specific range, typically between 0 and 1, which helps in bringing all features to a similar scale.
2. In this code, we import the StandardScaler from the scikit-learn library to standardize the price\_inr data. We then create an instance of StandardScaler and fit it to the price\_inr values, which prepares the scaler to transform these prices so that they have a mean of zero and a standard deviation of one.

A screenshot of a computer

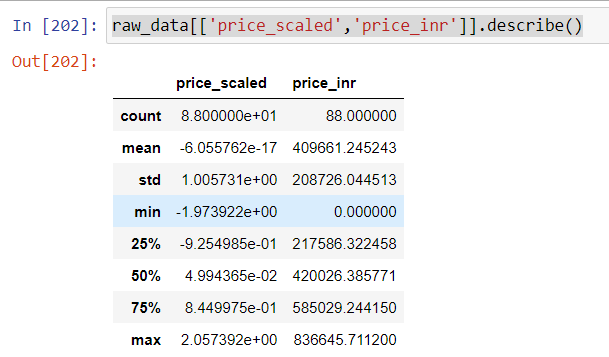
Description automatically generated

1. In this code, ss.mean\_ and ss.var\_ retrieve the mean and variance of the price\_inr data after fitting the StandardScaler. Then, we use ss.transform() to scale the price\_inr values, creating a new column called price\_scaled in the raw\_data DataFrame that contains the standardized prices, making them easier to compare across different datasets.

A screenshot of a computer

Description automatically generated

1. The command raw\_data[['price\_scaled','price\_inr']].describe() generates a summary of statistics for the price\_scaled and price\_inr columns in the raw\_data DataFrame. This summary includes key metrics such as the count, mean, standard deviation, minimum, maximum, and quartiles, helping you understand the distribution and characteristics of these two sets of price data.



1. In this code, we import the MinMaxScaler to scale the price\_inr values to a specified range between 1 and 2, ensuring that the smallest value is 1 and the largest is 2. We fit the scaler to the price\_inr data, then apply it to create a new column called price\_minmax\_scaled, which contains the scaled prices, and finally, we summarize the statistics for the price\_scaled, price\_inr, and price\_minmax\_scaled columns to understand their distributions.
2. And in the end we save data and name the file as final data.csv.

A screenshot of a computer

Description automatically generated