

Working with Data in ML

HIGHLIGHTS

- Loading and Working with Data
Data Representation (data samples, features, dimensionality, matrix representation)
 - Data visualization and EDA (histograms, time series, images)
 - Data pre-processing (normalization, standardization, dimensionality reduction techniques)
 - Summary
-

Loading a Dataset for analysis: Loading a Dataset and creating Feature and Target Matrices

Loading a dataset for machine learning applications is usually done using Python and implemented using *Jupyter Notebook*. We will demonstrate here the steps for loading a dataset and creating features and target matrices. Machine learning models are created using data. Therefore, the first step is to load the data in a format that the program understands.

One of the ways of loading data for analysis is using *scikit-learn* which is a Python library. The data is represented as a table or matrix. Most data tables for machine learning applications are two-dimensional containing rows and columns. Each row represents one observation commonly known as an *instance* and each column represents an independent variable or a *feature*.

Importing Data using Python and Seaborn in Jupyter Notebook

If you are using Anaconda, open Anaconda Navigator and follow the instructions below:

1. **Open Jupyter Notebook and type the code below to import seaborn**
: import seaborn as sns
2. **Check the version of sns, type**

```
sns.__version__  
'0.11.0'
```

3. Check how many available datasets are in Seaborn; type:

```
sns.get_dataset_names ()
```

The pre-loaded datasets in the Seaborn library list will be displayed as shown below. These are available to users. You can also upload your dataset. Uploading a data set using a .csv file is explained later.

```
['anagrams',  
'anscombe',  
'attention',  
'Brain_networks',  
'Car_crashes',  
'diamonds',  
'dots',  
'dowjones',  
'exercise',  
'flights',  
'fmri',  
'geyser',  
'glue',  
'healthexp',  
'iris',  
'mpg',  
'penguins',  
'planets',
```

```
'seaice',  
'taxi',  
'tips',  
'titanic']
```

- 4. Call a data set and check whether it has been loaded (here we are loading the ‘diamonds’ data set from the preloaded datasets and print the first five instances)**

```
df=sns.load_dataset('diamonds')
```

```
df.head ()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

- 5. Load another data set**

```
df=sns.load_dataset('iris')
```

```
df.head ()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- 6. To see the last five entries**

```
df.tail ()
```

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica

146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

7. To check the shape of the data (rows and columns)

```
df. shape
```

```
(150, 5)
```

8. Loading another data set called 'tips' and checking the shape.

```
data=sns. load_dataset('tips')
```

```
data. head (15)
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2
10	10.27	1.71	Male	No	Sun	Dinner	2
11	35.26	5.00	Female	No	Sun	Dinner	4
12	15.42	1.57	Male	No	Sun	Dinner	2
13	18.43	3.00	Male	No	Sun	Dinner	4
14	14.83	3.02	Female	No	Sun	Dinner	2

```
data. shape
```

```
(244, 7)
```

The above examples illustrate the steps for importing data for analysis. The next few steps will demonstrate how to create Features and Target matrices.

Importing Data using Seaborn into a Jupyter Notebook and creating Feature and Target Matrices.

1. Open a Jupyter Notebook.

```
import seaborn as sns
```

2. Load the 'diamonds' dataset and print the first 10 instances

```
. diamonds = sns. load_dataset ('diamonds')
```

```
diamonds. head (10)
```

	carat	cut	color	clarity	depth	table	price	x	y	z	
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	
1	0.21	Premium		E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	
3	0.29	Premium		I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	
5	0.24	Very Good		J	VVS2	62.8	57.0	336	3.94	3.96	2.48
6	0.24	Very Good		I	VVS1	62.3	57.0	336	3.95	3.98	2.47
7	0.26	Very Good		H	SI1	61.9	55.0	337	4.07	4.11	2.53
8	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49	
9	0.23	Very Good		H	VS1	59.4	61.0	338	4.00	4.05	2.39

3. Check the shape of the 'diamonds' data set (rows and columns)

```
diamonds. shape
```

```
(53940, 10)
```

The diamond data set has 53940 instances (rows) and 10 columns or features. We select ‘price’ as the target as it seems logical to predict the price using the other features.

We create a variable X to store the features and a variable Y to store the target matrix and use indexing to access only the values from the column ‘price’

- 4. Create a variable X to store the features by using drop (). The drop () function will include all of the features except the target ‘price’. Also, print the first 10 instances.**

```
X=diamonds.drop('price', axis=1)
```

```
X.head (10)
```

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
1	0.21	Premium		E	SI1	59.8	61.0	3.89	3.84
2	0.23	Good	E	VS1	56.9	65.0	4.05	4.07	2.31
3	0.29	Premium		I	VS2	62.4	58.0	4.20	4.23
4	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75
5	0.24	Very Good		J	VVS2	62.8	57.0	3.94	3.96
6	0.24	Very Good		I	VVS1	62.3	57.0	3.95	3.98
7	0.26	Very Good		H	SI1	61.9	55.0	4.07	4.11
8	0.22	Fair	E	VS2	65.1	61.0	3.87	3.78	2.49
9	0.23	Very Good		H	VS1	59.4	61.0	4.00	4.05

- 5. Create a variable Y to store the target matrix by using drop (). Print the first 10 instance.**

```
Y=diamonds['price']
```

```
Y. head (10)
```

```
0    326
```

```
1    326
```

```
2    327
```

```
3    334
```

```
4    335
```

5 336

6 336

7 337

8 337

9 338

Name: price, dtype: int64

6. Print out the shape of variable X and Y

```
X.shape
```

```
(53940, 9)
```

```
Y. shape
```

```
(53940,)
```

Reading CSV file into a Jupyter Notebook using Pandas and creating Feature and Target Matrices.

```
import pandas as pd
```

```
df=pd.read_csv("C:\\Users\\asahay\\OneDrive \\Documents\\Diamonds.csv")
```

```
df.head()
```

Column_1	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98
2.43										
1	2	0.21	Premium		E	SI1	59.8	61.0	326	3.89
3.84	2.31									
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07
2.31										
3	4	0.29	Premium		I	VS2	62.4	58.0	334	4.20
4.23	2.63									
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35
2.75										

```
df.tail()
```

	Column_1	carat	cut	color	clarity	depth	table	price	x	y	z
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

```
print(type(df))
```

```
<class 'pandas. core.frame.DataFrame'>
```

```
df.shape
```

```
(53940, 11)
```

```
df. columns
```

```
Index(['Column_1', 'carat', 'cut', 'color', 'clarity', 'depth', 'table',  
      'price', 'x', 'y', 'z'],  
      dtype='object')
```

```
df. dtypes
```

```
Column_1    int64
```

```
carat      float64
```

```
cut        object
```

```
color      object
```

```
clarity    object
```

```
depth      float64
```

```
table      float64
```

```
price      int64
```

x float64

y float64

z float64

dtype: object

```
print(df.info ())
```

```
<class 'pandas. core.frame.DataFrame'>
```

```
RangeIndex: 53940 entries, 0 to 53939
```

```
Data columns (total 11 columns):
```

```
# Column Non-Null Count Dtype
```

```
--- -----
```

```
0 Column_1 53940 non-null int64
```

```
1 carat 53940 non-null float64
```

```
2 cut 53940 non-null object
```

```
3 color 53940 non-null object
```

```
4 clarity 53940 non-null object
```

```
5 depth 53940 non-null float64
```

```
6 table 53940 non-null float64
```

```
7 price 53940 non-null int64
```

```
8 x 53940 non-null float64
```

```
9 y 53940 non-null float64
```

```
10 z 53940 non-null float64
```

```
dtypes: float64(6), int64(2), object(3)
```

```
memory usage: 4.5+ MB
```

```
None
```

```
X=df.drop('price',axis=1)
```

X.head(10)

Column_1	carat	cut	color	clarity	depth	table	x	y	z
0 1	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
1 2	0.21	Premium		E	SI1	59.8	61.0	3.89	3.84
2.31									
2 3	0.23	Good	E	VS1	56.9	65.0	4.05	4.07	2.31
3 4	0.29	Premium		I	VS2	62.4	58.0	4.20	4.23
2.63									
4 5	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75
5 6	0.24	Very Good		J	VVS2	62.8	57.0	3.94	3.96
2.48									
6 7	0.24	Very Good		I	VVS1	62.3	57.0	3.95	3.98
2.47									
7 8	0.26	Very Good		H	SI1	61.9	55.0	4.07	4.11
2.53									
8 9	0.22	Fair	E	VS2	65.1	61.0	3.87	3.78	2.49
9 10	0.23	Very Good		H	VS1	59.4	61.0	4.00	4.05
2.39									

Y=df['price']

Y.head(10)

0	326
1	326
2	327
3	334
4	335
5	336
6	336

7 337

8 337

9 338

Name: price, dtype: int64

Preprocessing the Entire Data Set

Before training the machine learning models, it is critical to check the data for missing values, outliers, noisy data and deal with them. The process of dealing with these issues is known as **Data Preprocessing**. If these issues are not resolved, it may lead to unrealistic assumptions by the models and may result in a less accurate final model and also unrealistic predictions.

There are different types of data present in a data set. The data is not always numerical but may contain nominal or ordinal data. It is a good idea to transform the data in different ways during preprocessing and test different transformations in different models. This helps to select the right transformation for the model.

The example below demonstrates how to check for missing values in each feature of the data set.

- 1. Load the data set 'diamonds' and create the features and target matrices. This set contains nine features and one target. The steps are explained here.**

```
import seaborn as sns
```

```
diamonds=sns.load_dataset('diamonds')
```

- 2. Create the feature and target matrices and print the shape (number of rows and columns)**

```
X=diamonds[['carat','cut','color',' clarity','depth','table','x','y','z']]
```

```
Y = diamonds['price']
```

```
X.shape
```

```
(53940, 9)
```

- 3. Check for missing values in all the features**

```
print ("Carat: "+ str(X['carat']. isnull (). sum ()))
```

```
print ("Cut: "+ str(X['cut']. isnull (). sum ()))  
print ("Color: "+ str(X['color']. isnull (). sum ()))  
print ("Clarity: "+ str(X['clarity']. isnull (). sum ()))  
print ("Depth: "+ str(X['depth']. isnull (). sum ()))  
print ("Table: "+ str(X['table']. isnull (). sum ()))  
print ("X: "+ str(X['x']. isnull (). sum ()))  
print ("Y: "+ str(X['y']. isnull (). sum ()))  
print ("Z: "+ str(X['z']. isnull (). sum ()))  
Carat: 0  
Cut: 0  
Color: 0  
Clarity: 0  
Depth: 0  
Table: 0  
X: 0  
Y: 0  
Z: 0
```

The results above show that there are no missing values in any of the features.

Outliers and missing values in Data Sets (Messy Data)

A data set that has missing values, or has outliers is considered as messy data set. The outliers are usually a very small or large value in a data set that are unusual information. It is important to deal with these issues using data preprocessing or transformation. It is not unusual to find all missing values from one or more instances or where some instances or features have a few values missing.

To deal with the missing data, the usual convention that is followed is to eliminate the the total row or the instance if all the features in that instance is missing values. On the other hand, if a

feature has several values missing (5 to 10%) then these values should be replaced with values using the following two methods: (1) replace the missing values using the mean or median of the values of that feature if the feature has numerical values or, (2) replace the missing values with the predicted values obtained using a regression function. Method (1) is known as **Mean imputation**, and method (2) is **Regression imputation**.

Outliers: As indicated, outliers are values that are very large or very small values. One way of detecting outliers is based on the Gaussian or Normal distribution. If the data values, follow a normal distribution. The values outside of

$[\text{mean} \pm 3 * (\text{standard deviation})]$ or, $(\mu \pm 3\sigma)$ are considered outliers.

Once the outliers are detected, there are three ways to handle them:

(1) delete the outliers. Sometimes the outliers may result from data recording errors. It is a good idea to delete them to avoid skewing the data.

(2) Assign a new value: a new value may be assigned to the outliers using either mean imputation or regression imputation. Here, we provide an example of missing values in a data set.

The data set used is Diamond, which is a CSV file.

1. Open a Jupyter notebook

2. Load the **Diamond** data set and store it in a variable called Diamond. Use the following code

```
: import pandas as pd
```

```
Diamond=pd.read_csv ("C:\\Users\\asahay\\OneDrive -  
\\Desktop\\Diamond.csv")
```

```
Diamond.head(15)
```

	Column_1	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326.0	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326.0	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327.0	4.05	4.07	2.31

3	4	0.29	Premium	I	VS2	62.4	58.0	334.0	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	NaN	58.0	335.0	4.34	4.35	2.75
5	6	0.24	Very Good	J	VVS2	62.8	57.0	336.0	3.94	3.96	2.48
6	7	0.24	Very Good	I	VVS1	62.3	57.0	336.0	3.95	3.98	2.47
7	8	0.26	Very Good	H	SI1	61.9	55.0	337.0	4.07	4.11	2.53
8	9	0.22	Fair	E	VS2	65.1	61.0	337.0	3.87	3.78	2.49
9	10	0.23	Very Good	H	VS1	59.4	61.0	NaN	4.00	4.05	2.39
10	11	0.30	Good	J	SI1	64.0	55.0	339.0	4.25	4.28	2.73
11	12	0.23	Ideal	J	VS1	62.8	56.0	340.0	3.93	3.90	2.46
12	13	0.22	Premium	F	SI1	60.4	61.0	342.0	3.88	3.84	2.33
13	14	0.31	Ideal	J	SI2	62.2	54.0	NaN	4.35	4.37	2.71
14	15	0.20	Premium	E	SI2	NaN	62.0	345.0	3.79	3.75	2.27

3. Create a variable called **price** to store the value of this feature from the dataset. Print out the top 15 values of the **price** variable.

```
price= Diamond['price']
```

```
price. Head (15)
```

```
0    326.0
1    326.0
2    327.0
3    334.0
4    335.0
5    336.0
6    336.0
7    337.0
8    337.0
9     NaN
10   339.0
11   340.0
12   342.0
13     NaN
```

14 345.0

Name: price, dtype: float64

4. Check the shape of the price variable

price. shape

(53940,)

5. The price variable has **NaN** (Not a Number) values in row 9 and 13. These are considered missing values.

6. Count the number of missing values (NaN values) in the variable **price** using the `isnull()` function

price. Isnull ().sum()

2

The result shows that there are 2 missing values in the feature **price** of the **Diamond** dataset

7. As explained above, create a variable called **depth** to store the value of this feature from the dataset. Print out the top 15 values of the **depth** variable.

depth= Diamond['depth']

depth. head(15)

0 61.5

1 59.8

2 56.9

3 62.4

4 NaN

5 62.8

6 62.3

7 61.9

8 65.1

9 59.4

10 64.0

11 62.8

12 60.4

13 62.2

14 NaN

Name: depth, dtype: float64

8. Check the shape of the feature **depth** and count the number of missing or NaN values in this feature.

```
depth. shape
(53940,)
depth. isnull().sum()
2
```

As we can see, the feature **depth** also has two missing values that are not numbers. It is also evident that the missing values in this feature are 2 out of 53940 instances. This is negligible, but these values can be replaced using mean imputation. This means that we will calculate the mean of the feature '**depth**' and replace all the missing values with the mean. Follow the steps below:

9. Calculate the mean of the feature depth and round the mean value to one decimal place

```
mean=depth. mean ( )
mean
61.74940487225996
print ("The value rounded to 1 decimal value ", round(mean,1))
```

The value rounded to 1 decimal value 61.7

10. Replace the missing values with the mean. Use fillna () function. To check that the values are replaced, print the first 15 values.

```
depth. fillna ( mean, inplace=True)
depth. head (15)
```

The result should show that the NaN values in the feature **depth** are replaced with the mean.

Outliers: As indicated, outliers are values that are very large or very small. One way of detecting outliers is based on the Gaussian or Normal distribution. If the the data values follow a normal distribution. The values outside of

$[\text{mean} \pm 3 * (\text{standard deviation})]$ or, $(\mu \pm 3\sigma)$ are considered outliers. (A)

Once the outliers are detected, there are three ways to handle them:

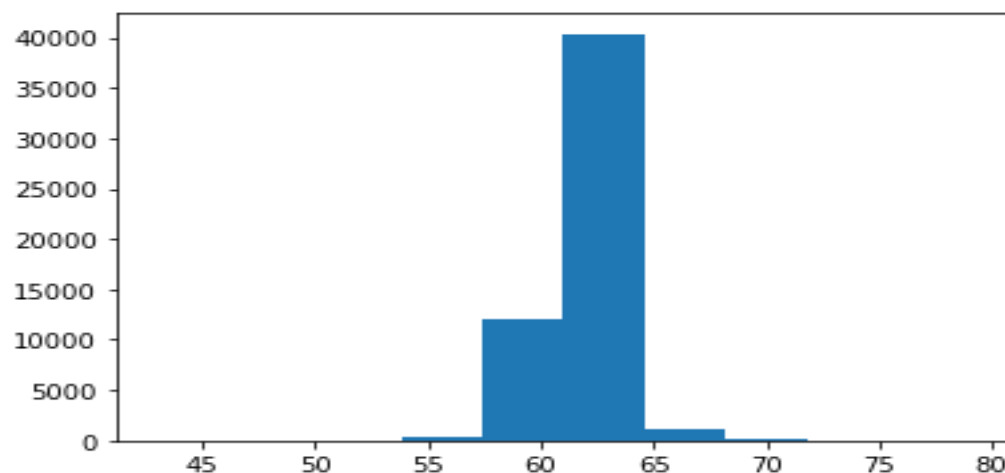
(1) delete the outliers. Sometimes the outliers may result from data recording errors. It is a good idea to delete them to avoid skewing the data.

To detect the outliers in the variable **depth**, first we will create a histogram of the feature '**depth**'. This will visually provide some ideas on the outliers. We use '**matplotlib**' to plot the **histogram** of the feature **depth**. Continue with the steps below. (Note: the steps continue from step 10 above)

11. Import Matplotlib and plot a histogram of the **depth** variable using **hist()** function. Type the following:

```
import matplotlib.pyplot as plt  
plt.hist(depth)  
plt.show()
```

The resulting histogram is shown in Figure 5.1. The shape of the histogram can be well approximated by a Gaussian or Normal distribution. Therefore, the outliers can be determined using equation (A) above.



Chapter 5 Figure 5.1: Histogram from Matplotlib

To determine the outliers in the variable **depth** we will use $[\text{mean} \pm 3 * (\text{standard deviations})]$.

12. The minimum value is found by calculating the mean of the values in the variable '**depth**' and subtracting three standard deviations from it using the following code. The minimum value will be stored in a variable, **min_val**. Use the following code.

```
min_val=depth. mean () - (3*depth.std())  
min_val  
57.45155450801473
```

The minimum value is printed above.

13. In a similar way, the maximum value will be calculated and stored in a variable, **man_val**. Use the following code.

```
max_val=depth. mean()+(3*depth.std())  
max_val  
66.0472552365052
```

The maximum value is printed above.

From the above, the minimum value is 57.452 and the maximum value is 66.047. This indicates that there are no outliers at the lower tail of the distribution (see the histogram). The maximum value in the variable depth is 66.047. This indicates that there are values outside of the maximum value.

14. We will count the number of instances that are above the maximum value. Using indexing, we calculate the values in the variable **depth** that are above the maximum value and store them in a variable named outliers. Use the code below:

```
outliers=depth[depth>max_val]  
outliers. count ()  
366
```

A partial list of outliers is shown below.

```
97    66.3
204    67.9
298    67.4
352    67.3
369    66.4
...
53495  66.9
53540  72.9
53727  66.9
53800  68.7
53863  66.8
```

Name: depth, Length: 366, dtype: float64

The total number of outliers is 366. This is less than 1% of the total number of instances 53940.

The number of outliers is significantly smaller, they can be deleted. We will delete the instances from the variable **depth**.

15. Redefine the values stored in the variable **depth** using indexing and calculate only the values below the max-val threshold and print the shape of **depth**.

```
depth=depth[depth<=max_val]
depth. shape
(53572,)
```

This shows that the outliers are deleted and the total number of instances in depth is now reduced by the number of outliers. Note that the total number of instances in the **Diamond** data set is 53940 and there are 366 outliers. The final number of instances reported is 53572 which is two more. This is because there are two feature values labeled NaN (not a number).

Dealing with Categorical Features

In a data set, it is not unusual to find categorical features. The data set with categorical or qualitative variables are not numbers, and they are classified as **nominal** or **ordinal**. Nominal data are data that are classified where there is no order involved. Examples of such data are the color of your car or marital status. Since there is no order involved, this type of data cannot be ranked, and it is difficult to say which value is bigger or better. The other type of categorical data is ordinal. These are also not numbers, but there is some order involved. An example of this type of data is student grades that are usually classified as A, B, C, ...etc. These are categorical, but there is order involved. For example, the letter grade A is better than B, B is better than C, and so on.

The data set having categorical features may be text instead of numbers. In creating machine learning models, the categorical features are transformed into numeric values. This makes the model training efficient, results in faster running time, and better model performance.

The process by which the categorical variables are converted into numerical values is known as **label encoding**. This process is referred to as **Feature engineering** that generates a label encoding that assigns a numeric value to each category. This value replaces the category in the dataset. We will demonstrate the feature engineering and label encoding using the Diamond dataset that we have used earlier.

We will use the Diamond.csv file using pandas and explore the categorical feature that we will code.

1. Import Pandas then import **Scikit Learns's Label Encoder ()**
import pandas as pd
from sklearn.preprocessing import LabelEncoder
2. Download the **Diamonds.csv file**
Diamond=pd.read_csv ("Diamonds.csv")

3. You may use the describe command to calculate the descriptive statistics of the numeric features. (The output of this command is not shown here)

```
Diamond.describe()
```

```
Diamond.head()
```

Column_1	carat	cut	color	clarity	depth	table	price	x	y	z	
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

By printing the first five instances, it is evident that the feature 'cut' is a categorical variable.

4. Use the command below to see the different categories of the feature "cut"

```
Diamond["cut"].unique()
```

```
array(['Ideal', 'Premium', 'Good', 'Very Good', 'Fair'], dtype=object)
```

```
lbl_encode=LabelEncoder()
```

```
lbl_encode.fit_transform(Diamond['cut'])
```

```
array([2, 3, 1, ..., 4, 3, 2])
```

```
Diamond['cut_label']=lbl_encode.fit_transform(Diamond["cut"])
```

```
Diamond["cut_label"].value_counts()
```

```
2 21551
```

```
3 13791
```

```
4 12082
```

1 4906

0 1610

Name: cut_label, dtype: int64

Diamond['cut'].value_counts ()

Ideal 21551

Premium 13791

Very Good 12082

Good 4906

Fair 1610

Name: cut, dtype: int64

Data pre-processing (normalization, standardization)

Normalization: Data normalization in machine learning is a part of data preprocessing and data preparation. Normalization involves rescaling the values of all the features so that they lie in the range of 0 and 1, with a maximum of 1. The equation to normalize the values of a feature is given by:

$$Z_i = \frac{\min(x)}{\max(x) - \min(x)}$$

where, Z_i is the i^{th} normalized value and x_i are the data values.

Data normalization reorganizes the data so that it can be used uniformly and is similar for further analysis. It is a process of data cleaning through eliminating redundant and unstructured data and making the data appear uniform across all features. To normalize a data set, the data needs to be scaled to a common range, such as between 0 and 1. This is done using the method of Min-Max scaling or Z-score normalization as shown above. Normalization reduces data redundancy, improves data analysis, and enhances data security. It can eliminate errors, inconsistencies, and missing values that can affect the accuracy of data and analysis.

Standardization: Data standardization is a rescaling technique. It transforms the data into a Gaussian distribution with a mean equal to zero and a standard deviation of 1. The following equation shows the standardization process of features in a data set.

$$Z_i = \frac{x_i - \text{mean}(x)}{\text{std}(x)}$$

where Z_i is the i^{th} standardized value and x_i are the data values.

Data standardization is a data pre-processing technique in many machine learning models. It is used to standardize the range of features of an input data set.

Data standardization is used when features of the input data set have large variation or differences between their ranges, or when the data are measured using different scales of measurement for example, pounds, kilograms, meters, feet, etc.

These differences in the ranges of initial features cause trouble for many machine learning models. For example, suppose a model is based on distance computation; if one of the features has a large range or variation, the distance will be governed by this particular feature. Using data standardization, the data is converted to **a standard format that is easily understood and processed by computers.**

As an example, suppose we want to predict the outcome of a complex system using neural networks (ANN's). The target or the dependent variable values range between 0 and 12,000. Also, different features, or the independent variables, have wide ranges. Suppose all the variables have an approximately Gaussian distribution. In such cases, it is important to scale the data before

training the model. One way is to use the Z-score. Note: Both the normalization and standardization techniques can be implemented using Seaborn in Jupyter notebook.

Summary

Data is one of the most important and critical parts of creating and implementing machine learning models. We demonstrated, through several examples, how to get the data for analysis. Before data can be used to create models, it must be checked and pre-processed for accuracy. We showed how to import data sets using Python and the Seaborn package in a Jupyter notebook. This chapter showed the ways to load the data for analysis, create the features and target matrix, how to perform some data visualizations, and perform EDA (Exploratory Data Analysis). Through stepwise instructions, the data manipulation process, such as checking the data for validity were shown. The last section of this chapter discussed the data pre-processing techniques, including normalization and standardization. The importance and need for data normalization and standardization were explained.