-What is Data Cleaning?

Data Cleaning means the process of identifying the incorrect, incomplete, inaccurate, irrelevant or missing part of the data and then modifying, replacing or deleting them according to the necessity. Data cleaning is considered a foundational element of the basic data science.

Data is the most valuable thing for Analytics and Machine learning. In computing or Business data is needed everywhere. When it comes to the real world data, it is not improbable that data may contain incomplete, inconsistent or missing values. If the data is corrupted then it may hinder the process or provide inaccurate results. Let’s see some examples of the importance of data cleaning.

Suppose you are a general manager of a company. Your company collects data of different customers who buy products produced by your company. Now you want to know on which products people are interested most and according to that you want to increase the production of that product. But if the data is corrupted or contains missing values then you will be misguided to make the correct decision and you will be in trouble.

At the end of all, Machine Learning is a data-driven AI. In machine learning, if the data is irrelevant or error-prone then it leads to an incorrect model building.

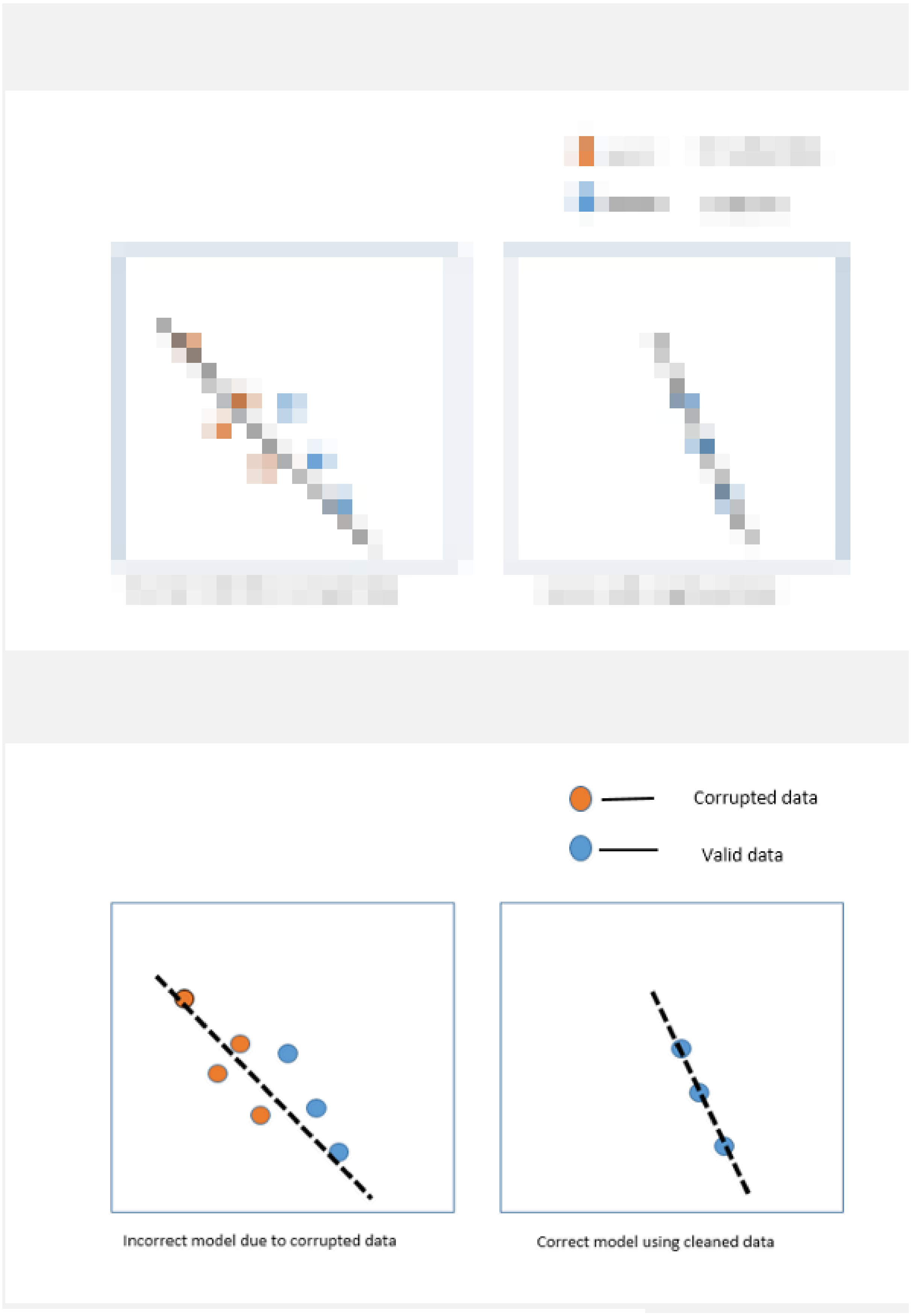


Figure 1: Impact of data on Machine Learning Modeling.

As much as you make your data clean, as much as you can make a better model. So, we need to process or clean the data before using it. Without the quality data,it would be foolish to expect anything good outcome.

Different Ways of Cleaning Data

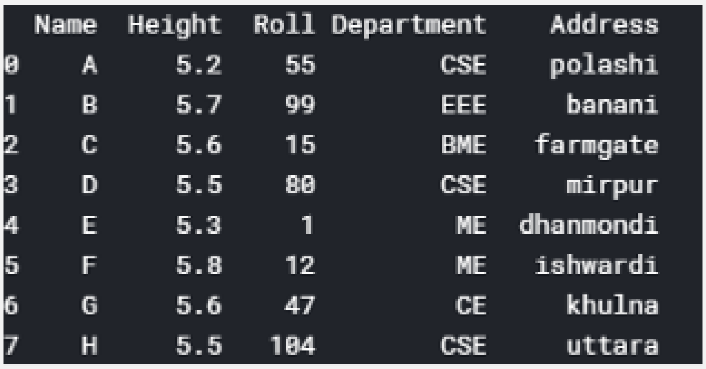
Now let's take a closer look in the different ways of cleaning data.

Inconsistent column :

If your **DataFrame (**A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns**)** contains columns that are irrelevant or you are never going to use them then you can drop them to give more focus on the columns you will work on. Let's see an example of how to deal with such data set. Let's create an example of students data set using **pandas** DataFrame.

```
import numpy as np # linear algebra



import pandas as pd # data processing, CSV file I/O



data={'Name':['A','B','C','D','E','F','G','H']



    ,'Height':[5.2,5.7,5.6,5.5,5.3,5.8,5.6,5.5],
    'Roll':[55,99,15,80,1,12,47,104],

    'Department':['CSE','EEE','BME','CSE','ME','ME','CE','CSE'],

'Address':['polashi','banani','farmgate','mirpur','dhanmondi','ishwardi','khulna','uttara']}

df=pd.DataFrame(data)

print(df)
```

[illegible]



```
  Name  Height  Roll Department    Address
0    A     5.2    55        CSE    polashi
1    B     5.7    99        EEE     banani
2    C     5.6    15        BME   farmgate
3    D     5.5    80        CSE     mirpur
4    E     5.3     1         ME  dhanmondi
5    F     5.8    12         ME   ishwardi
6    G     5.6    47         CE     khulna
7    H     5.5   104        CSE     uttara
```

Figure 2: Student data set

Here if we want to remove the “Height” column, we can use python **pandas.DataFrame.drop** to drop specified labels from rows or columns.

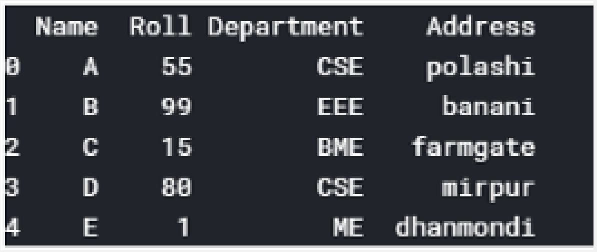
DataFrame.**drop**(*self*, *labels=None*, *axis=0*, *index=None*, *columns=None*, *level=None*, *inplace=False*, *errors='raise'*)

Let us drop the height column. For this you need to push the column name in the column keyword.

df=df.drop(columns='Height')

print(df.head())

```
[illegible]
```



```
  Name  Roll Department    Address
0    A    55        CSE    polashi
1    B    99        EEE     banani
2    C    15        BME   farmgate
3    D    80        CSE     mirpur
4    E     1         ME  dhanmondi
```

Figure 3: “Height” column dropped

Missing data:

It is rare to have a real world dataset without having any missing values. When you start to work with real world data, you will find that most of the dataset contains missing values. Handling missing values is very important because if you leave the missing values as it is, it may affect your analysis and machine learning models. So, you need to be sure that whether your dataset contains missing values or not. If you find missing values in your dataset you must handle it. If you find any missing values in the dataset you can perform any of these three task on it:

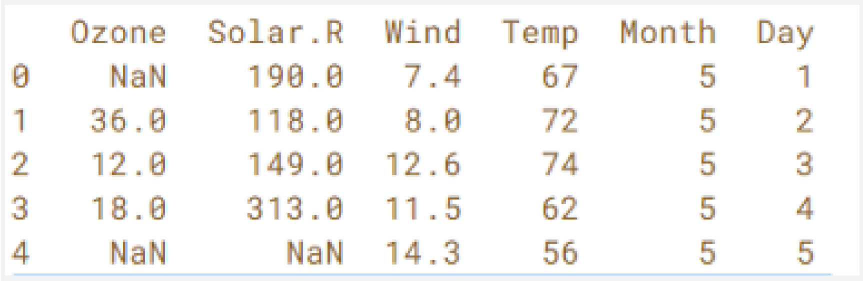
1. Leave as it is

2. Filling the missing values

3. Drop them

For filling the missing values we can perform different methods. For example, Figure 4 shows that airquality dataset has missing values.

```
airquality.head() #  return top n (5 by default) rows of a data frame
```



```
   Ozone  Solar.R  Wind  Temp  Month  Day
0    NaN    190.0   7.4    67      5    1
1   36.0    118.0   8.0    72      5    2
2   12.0    149.0  12.6    74      5    3
3   18.0    313.0  11.5    62      5    4
4    NaN      NaN  14.3    56      5    5
```

Figure 4: missing values.

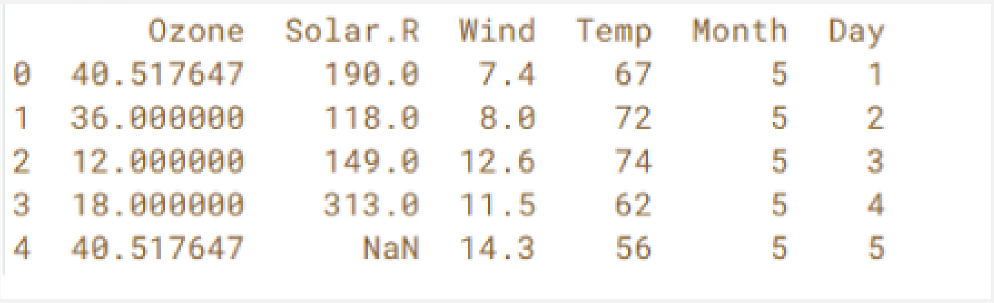
In figure 4, NaN indicates that the dataset contains missing values in that position. After finding missing values in your dataset, You can use *pandas.DataFrame.fillna* to fill the missing values.

DataFrame.**fillna**(*self*, *value=None*, *method=None*, *axis=None*, *inplace=False*, *limit=None*, *downcast=None*, ***kwargs*)

You can use different statistical methods to fill the missing values according to your needs. For example, here in figure 5, we will use the statistical mean method to fill the missing values.

```
airquality['Ozone'] = airquality['Ozone'].fillna(airquality.Ozone.mean())
```

```
airquality.head()
```



```
       Ozone  Solar.R  Wind  Temp  Month  Day
0  40.517647    190.0   7.4    67      5    1
1  36.000000    118.0   8.0    72      5    2
2  12.000000    149.0  12.6    74      5    3
3  18.000000    313.0  11.5    62      5    4
4  40.517647      NaN  14.3    56      5    5
```

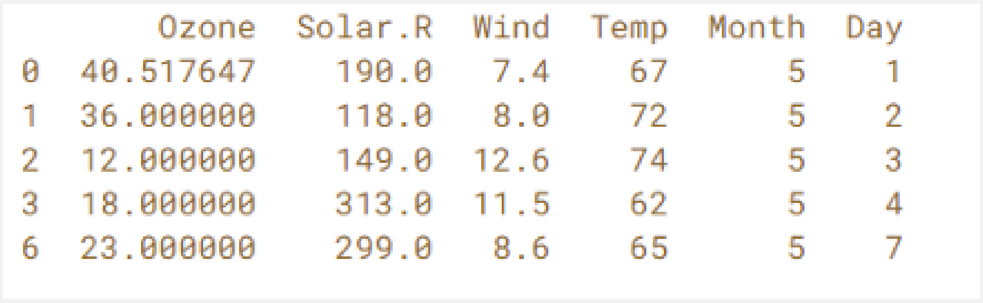
Figure 5: Filling missing values with the mean value.

You can see that the missing values in "Ozone" column is filled with the mean value of that column.

You can also drop the rows or columns where missing values are found. we drop the rows containing missing values. Here You can drop missing values with the help of **pandas.DataFrame.dropna.**

```
airquality = airquality.dropna() #drop the rows containing at least one missing value
```

```
airquality.head()
```

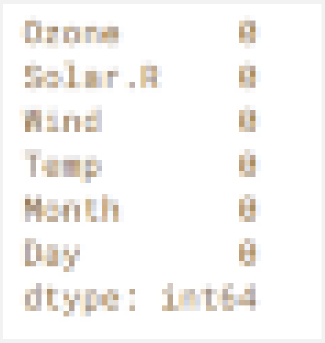


```
       Ozone  Solar.R  Wind  Temp  Month  Day
0  40.517647    190.0   7.4    67      5    1
1  36.000000    118.0   8.0    72      5    2
2  12.000000    149.0  12.6    74      5    3
3  18.000000    313.0  11.5    62      5    4
6  23.000000    299.0   8.6    65      5    7
```

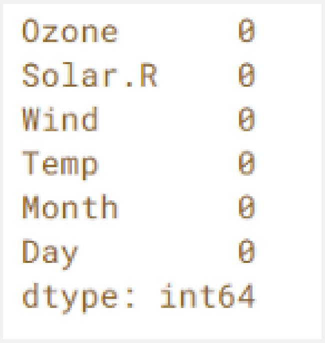
Figure 6: Rows are dropped having at least one missing value.

Here, in figure 6, you can see that rows have missing values in column Solar.R is dropped.

```
airquality.isnull().sum(axis=0)
```



```
Ozone      0
Solar.R    0
Wind       0
Temp       0
Month      0
Day        0
dtype: int64
```



```
Ozone      0
Solar.R    0
Wind       0
Temp       0
Month      0
Day        0
dtype: int64
```

Figure 7: Shows the numbers of missing values in column.

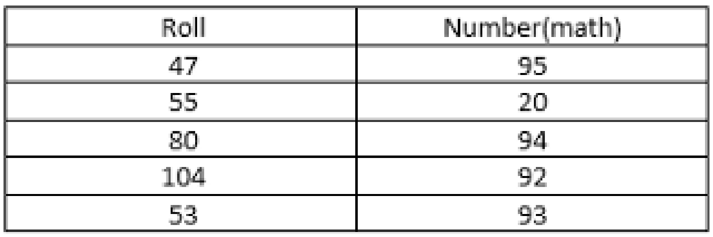
Outliers:

If you are new data Science then the first question that will arise in your head is "what does these outliers mean" ? Let's talk about the outliers first and then we will talk about the detection of these outliers in the dataset and what will we do after detecting the outliers.

According to wikipedia,

"*In statistics, an* ***outlier*** *is a data point that differs significantly from other observations.*"

That means an outlier indicates a data point that is significantly different from the other data points in the data set. Outliers can be created due to the errors in the experiments or the variability in the measurements. Let's look an example to clear the concept.



Roll	Number(math)
47	95
55	20
80	94
104	92
53	93

Figure 8: Table contains outlier.

In Figure 4 all the values in math column are in range between 90–95 except 20 which is significantly different from others. It can be an input error in the dataset. So we can call it a outliers. One thing should be added here — " ***Not all the outliers are bad data points. Some can be errors but others are the valid values.*** "

So, now the question is how can we detect the outliers in the dataset.

For detecting the outliers we can use :

1. Box Plot

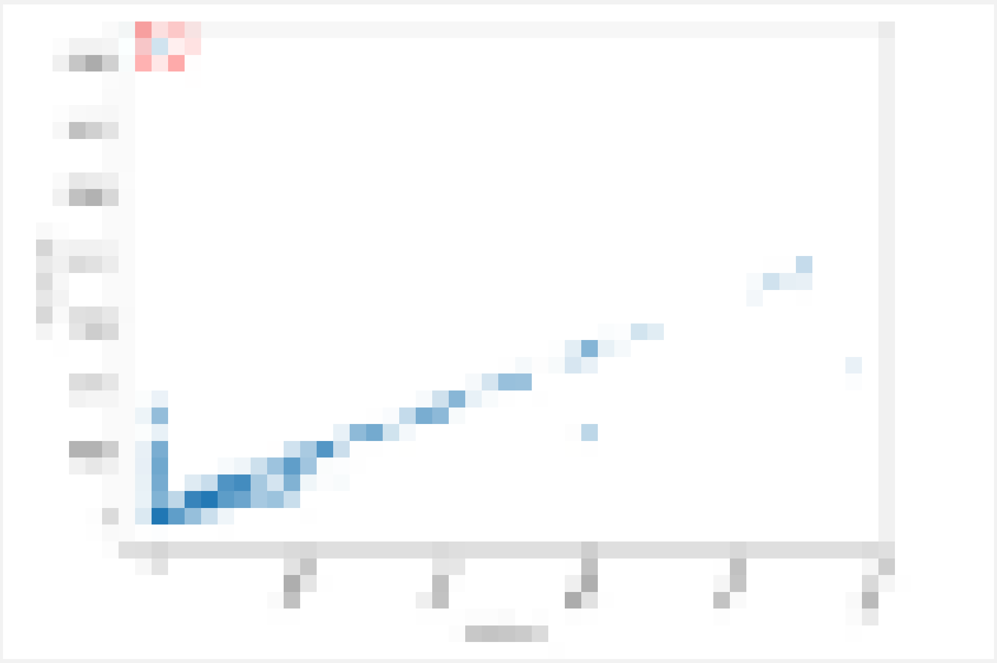
2. Scatter plot

3. Z-score etc.

We will see the Scatter Plot method here. Let's draw a scatter plot of a dataset.

```
dataset.plot(kind='scatter' , x='initial_cost' , y='total_est_fee' , rot = 70)
```

```
plt.show()
```



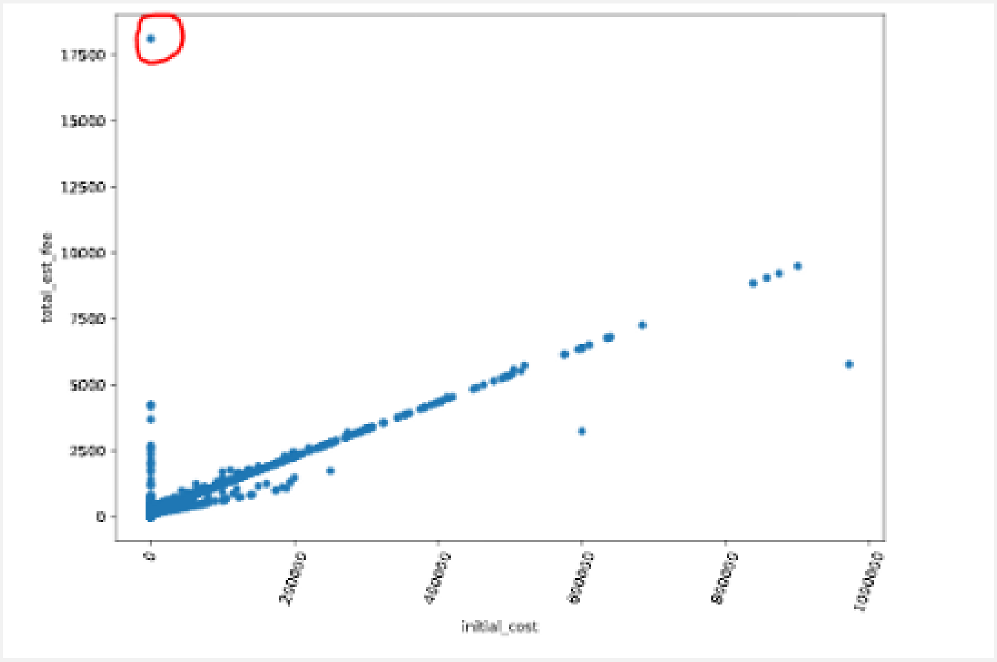


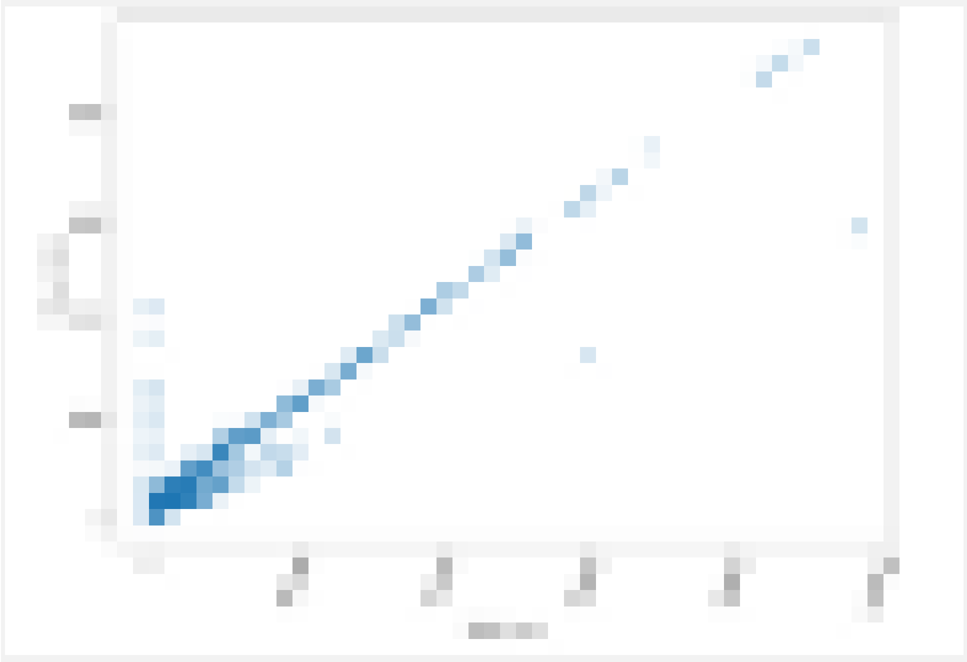
Figure 9: Scatter plotting with outlier.

Here in Figure 9 there is a outlier with red outline. After detecting this, we can remove this from the dataset.

```
df_removed_outliers = dataset[dataset.total_est_fee<17500]
```

```
df_removed_outliers.plot(kind='scatter', x='initial_cost' , y='total_est_fee' , rot =
70)
```

```
plt.show()
```



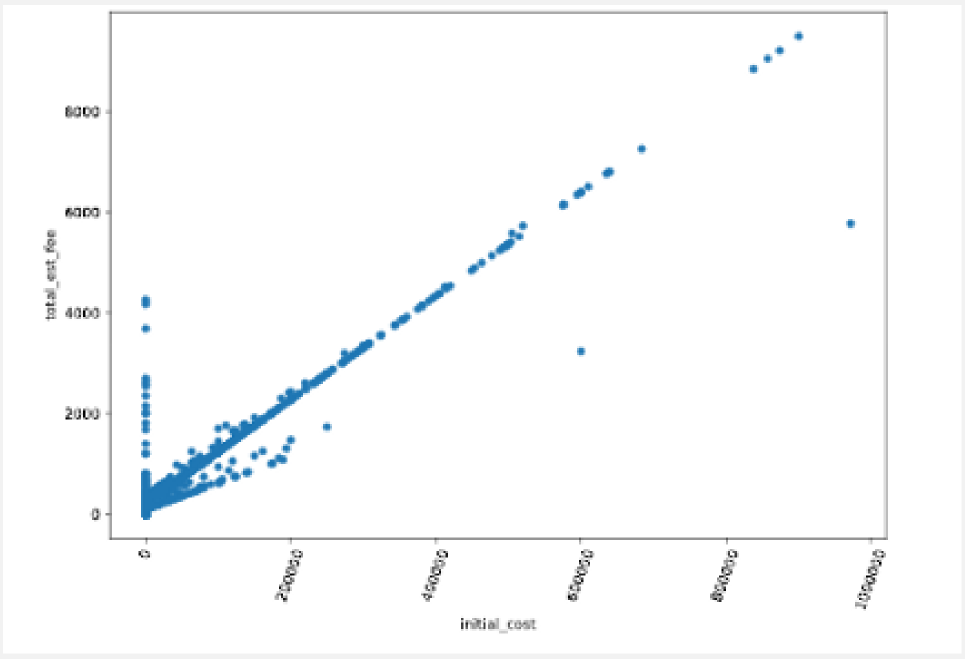
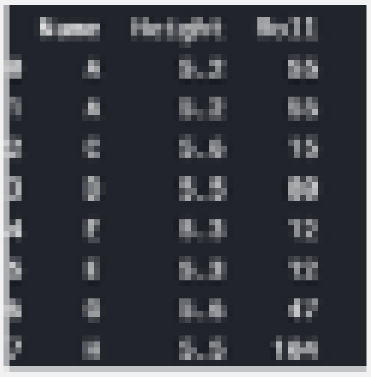


Figure 10: Scatter plotting with removed outliers.

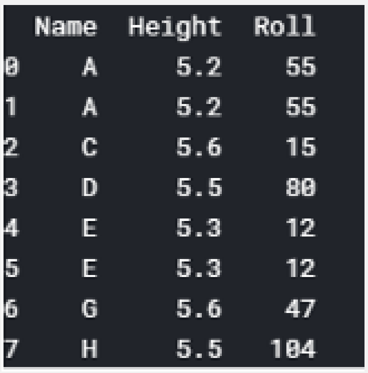
Duplicate rows:

Datasets may contain duplicate entries. It is one of the most easiest task to delete duplicate rows. To delete the duplicate rows you can use —

dataset_name.drop_duplicates(). Figure 12 shows a sample of a dataset having duplicate rows.



	Name	Height	Roll
0	A	5.2	55
1	A	5.2	55
2	C	5.6	15
3	D	5.5	80
4	E	5.3	12
5	E	5.3	12
6	G	5.6	47
7	H	5.5	104

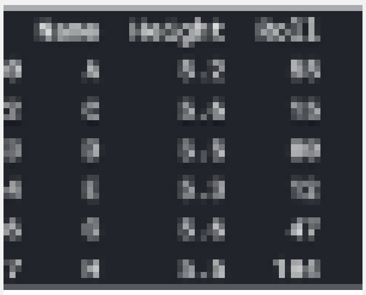


```
  Name  Height  Roll
0    A     5.2    55
1    A     5.2    55
2    C     5.6    15
3    D     5.5    80
4    E     5.3    12
5    E     5.3    12
6    G     5.6    47
7    H     5.5   104
```

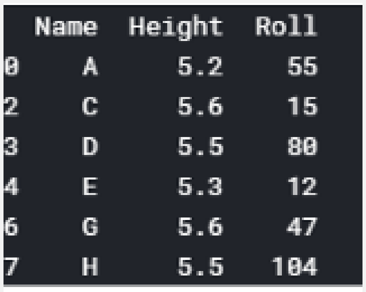
Figure 11: Data having duplicate rows.

```
dataset=dataset.drop_duplicates()#this will remove the duplicate rows.
```

```
print(dataset)
```



```
  Name  Height  Roll
0    A     5.2    55
2    C     5.6    15
3    D     5.5    80
4    E     5.3    12
6    G     5.6    47
7    H     5.5   104
```



```
  Name  Height  Roll
0    A     5.2    55
2    C     5.6    15
3    D     5.5    80
4    E     5.3    12
6    G     5.6    47
7    H     5.5   104
```

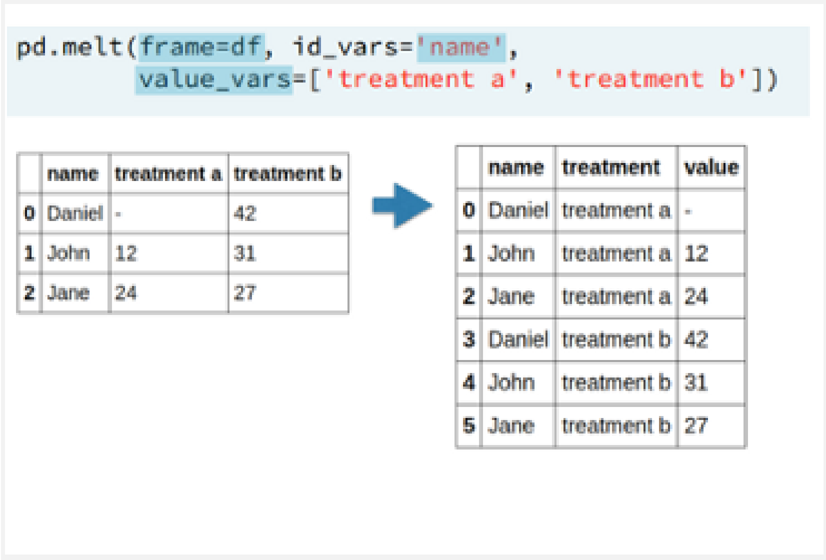
Figure 12: Data without duplicate rows.

Tidy data set:

Tidy dataset means each columns represent separate variables and each rows represent individual observations. But in untidy data each columns represent values but not the variables. Tidy data is useful to fix common data problem.You can turn the untidy data to tidy data by using ***pandas.melt*.**

```
import pandas as pd
```

```
pd.melt(frame=df,id_vars='name',value_vars=['treatment a','treatment b'])
```



```
pd.melt(frame=df, id_vars='name',
        value_vars=['treatment a', 'treatment b'])
```

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27

	name	treatment	value
0	Daniel	treatment a	-
1	John	treatment a	12
2	Jane	treatment a	24
3	Daniel	treatment b	42
4	John	treatment b	31
5	Jane	treatment b	27

Figure 13: Converting from Untidy to tidy data.

You can also see ***pandas.DataFrame.pivot*** for un-melting the tidy data.

Converting data types:

In DataFrame data can be of many types. As example :

1. Categorical data

2. Object data

3. Numeric data

4. Boolean data

Some columns data type can be changed due to some reason or have inconsistent data type. You can convert from one data type to another by using ***pandas.DataFrame.astype*****.**

DataFrame.**astype**(*self*, *dtype*, *copy=True*, *errors='raise'*, *****kwargs***)

String manipulation:

One of the most important and interesting part of data cleaning is string manipulation. In the real world most of the data are unstructured data. String manipulation means the process of changing, parsing, matching or analyzing strings. For string manipulation, you should have some knowledge about regular expressions. Sometimes you need to extract some value from a large sentence. Here string manipulation gives us a strong benefit. Let say,

"This umbrella costs $12 and he took this money from his mother."

If you want to exact the "$12" information from the sentence then you have to build a regular expression for matching that pattern.After that you can use the python libraries.There are many built in and external libraries in python for string manipulation.

```
import re
```

```
pattern = re.compile('|\$|d*')
```

```
result = pattern.match("$12312312")
```

```
print(bool(result))
```

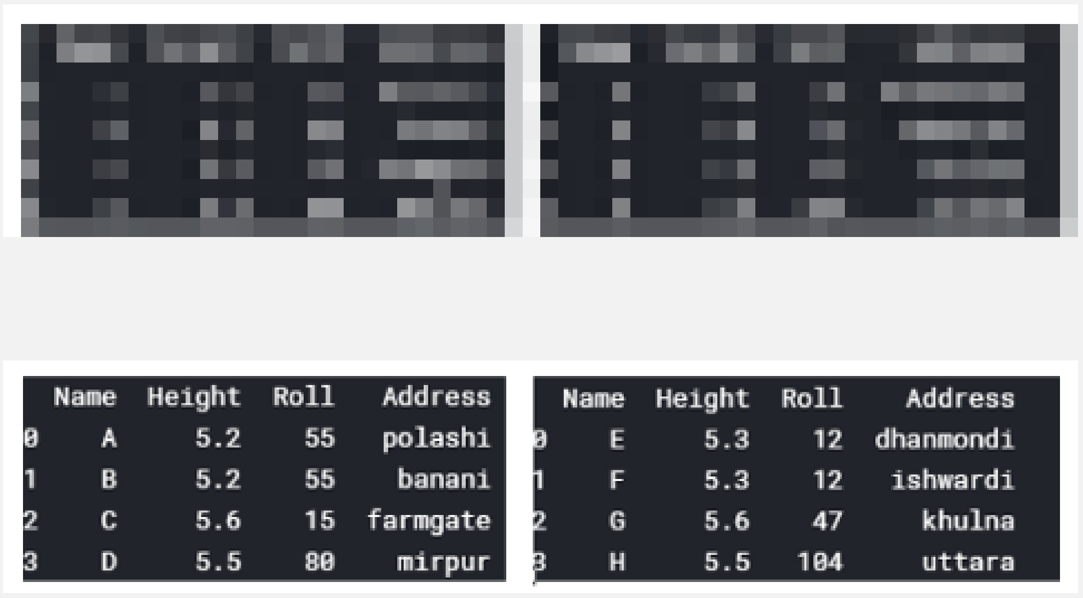
This will give you an output showing "True".

Data Concatenation:

In this modern era of data science the volume of data is increasing day by day. Due to the large number of volume of data data may stored in separated files. If you work with multiple files then you can concatenate them for simplicity. You can use the following python library for concatenate.

pandas.**concat**(*objs*, *axis=0*, *join='outer'*, *join_axes=None*, *ignore_index=False*, *keys=None*, *levels=None*, *names=None*, *verify_integrity=False*, *sort=None*, *copy=True*)

Let’s see an example how to concatenate two dataset. Figure 14 shows an example of two different datasets loaded from two different files. We will concatenate them using ***pandas.concat***.



```
  Name  Height  Roll   Address
0    A     5.2    55   polashi
1    B     5.2    55    banani
2    C     5.6    15  farmgate
3    D     5.5    80    mirpur
```

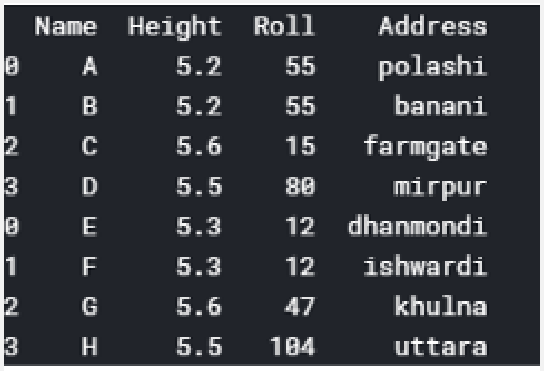
```
  Name  Height  Roll    Address
0    E     5.3    12  dhanmondi
1    F     5.3    12   ishwardi
2    G     5.6    47     khulna
3    H     5.5   104     uttara
```

Figure 14: Dataset1(left) & Dataset2(right)

```
concatenated_data=pd.concat([dataset1,dataset2])
```

```
print(concatenated_data)
```

[illegible]



```
  Name  Height  Roll    Address
0    A     5.2    55    polashi
1    B     5.2    55     banani
2    C     5.6    15   farmgate
3    D     5.5    80     mirpur
0    E     5.3    12  dhanmondi
1    F     5.3    12   ishwardi
2    G     5.6    47     khulna
3    H     5.5   104     uttara
```

Figure 15: Concatenated dataset.
https://towardsdatascience.com/what-is-data-cleaning-how-to-process-data-for-analytics-and-machine-learning-modeling-c2afcf4fbf45