Data preparation:

•Data preparation is the process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data, and combining datasets to enrich data.

•Data preparation is often a lengthy undertaking for data engineers or business users, but it is essential as a prerequisite to put data in context in order to turn it into insights and eliminate bias resulting from poor data quality.

•For example, the data preparation process usually includes standardizing data formats, enriching source data, and/or removing outliers.

•76% of data scientists say that data preparation is the worst part of their job, but efficient, accurate business decisions can only be made with clean data. Data preparation helps:

- **Fix errors quickly** — Data preparation helps catch errors before processing. After data has been removed from its original source, these errors become more difficult to understand and correct.
- **Produce top-quality data** — Cleaning and reformatting datasets ensures that all data used in analysis will be of high quality.
- **Make better business decisions** — Higher-quality data that can be processed and analyzed more quickly and efficiently leads to more timely,

Data preparation steps

The specifics of the data preparation process vary by industry, organization, and need, but the workflow remains largely the same.

1. Gather data

The data preparation process begins with finding the right data. This can come from an existing data catalog or data sources can be added ad-hoc.

2. Discover and assess data

After collecting the data, it is important to discover each dataset. This step is about getting to know the data and understanding what must be done before the data becomes useful in a particular context.

3. Cleanse and validate data

Cleaning up the data is traditionally the most time-consuming part of the data preparation process, but it’s crucial for removing faulty data and filling in gaps. Important tasks here include:

Removing extraneous data and outliers

Filling in missing values

Conforming data to a standardized pattern

Masking private or sensitive data entries

Once data has been cleansed, it must be validated by testing for errors in the data preparation process up to this point. Often, an error in the system will become apparent during this validation step and will need to be resolved before moving forward.

4. Transform and enrich data

Data transformation is the process of updating the format or value entries in order to reach a well-defined outcome, or to make the data more easily understood by a wider audience. *Enriching* data refers to adding and connecting data with other related information to provide deeper insights.

5. Store data

Once prepared, the data can be stored or channeled into a third-party application — such as a business intelligence tool — clearing the way for processing and analysis to take place.

What is data cleaning?

- Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset. But it is crucial to establish a template for your data cleaning process, so you know you are doing it the right way every time.

•**What is the difference between data cleaning and data transformation?**

•Data cleaning is the process that removes data that does not belong in your dataset. Data transformation is the process of converting data from one format or structure into another. Transformation processes can also be referred to as data wrangling, or data munging, transforming and mapping data from one "raw" data form into another format for warehousing and analyzing.

How to clean data

•Step 1: Remove duplicate or irrelevant observations

•Remove unwanted observations from your dataset, including duplicate observations or irrelevant observations. Duplicate observations will happen most often during data collection. When you combine data sets from multiple places, scrape data, or receive data from clients or multiple departments, there are opportunities to create duplicate data. De-duplication is one of the largest areas to be considered in this process. Irrelevant observations are when you notice observations that do not fit into the specific problem you are trying to analyze. For example, if you want to analyze data regarding millennial customers, but your dataset includes older generations, you might remove those irrelevant observations. This can make analysis more efficient and minimize distraction from your primary target—as well as creating a more manageable and more performant dataset.

•Step 2: Fix structural errors

•Structural errors are when you measure or transfer data and notice strange naming conventions, typos, or incorrect capitalization. These inconsistencies can cause mislabeled categories or classes. For example, you may find “N/A” and “Not Applicable” both appear, but they should be analyzed as the same category.

•Step 3: Filter unwanted outliers

•Often, there will be one-off observations where, at a glance, they do not appear to fit within the data you are analyzing. If you have a legitimate reason to remove an outlier, like improper data-entry, doing so will help the performance of the data you are working with. However, sometimes it is the appearance of an outlier that will prove a theory you are working on. Remember: just because an outlier exists, doesn’t mean it is incorrect. This step is needed to determine the validity of that number. If an outlier proves to be irrelevant for analysis or is a mistake, consider removing it.

•Step 4: Handle missing data

•You can’t ignore missing data because many algorithms will not accept missing values. There are a couple of ways to deal with missing data. Neither is optimal, but both can be considered.

1. As a first option, you can drop observations that have missing values, but doing this will drop or lose information, so be mindful of this before you remove it.
2. As a second option, you can input missing values based on other observations; again, there is an opportunity to lose integrity of the data because you may be operating from assumptions and not actual observations.
3. As a third option, you might alter the way the data is used to effectively navigate null values.

•Step 5: Validate and QA

•At the end of the data cleaning process, you should be able to answer these questions as a part of basic validation:

- Does the data make sense?
- Does the data follow the appropriate rules for its field?
- Does it prove or disprove your working theory, or bring any insight to light?
- Can you find trends in the data to help you form your next theory?
- If not, is that because of a data quality issue?

•False conclusions because of incorrect or “dirty” data can inform poor business strategy and decision-making. False conclusions can lead to an embarrassing moment in a reporting meeting when you realize your data doesn’t stand up to scrutiny. Before you get there, it is important to create a culture of quality data in your organization. To do this, you should document the tools you might use to create this culture and what data quality means to you.

•What is missing data?

•A missing value is a value which is not stored in dataset during observations. The classification of missing values was done in 1976 by D.B. Rubin. He reckoned that every data point has a possibility to be lost. The classes are as follow:

•· Missing Completely at Random (MCAR)

•· Missing at Random (MAR)

•· Missing Not at Random (MNAR)

•Classes:

•Missing Completely at Random (MCAR):

•For this case, the missing values are unrelated to the observations. If the possibility of being missed is equal for all cases, then the data is missed completely at random. MCAR data, nevertheless, are highly unusual in practice.

•For example, during the survey of a population, if any responds are lost, then they are missed completely at random.

•Missing at Random (MAR):

•This is the case when the missing variable can be defined by another variable, not from the missing values themselves.

•For instance, in the survey about the depression levels between two sexes, males are less likely to respond the questions on the depression level in comparison with females. Therefore, the missing value depends on gender only. This case is MAR.

•Missing Not at Random (MNAR):

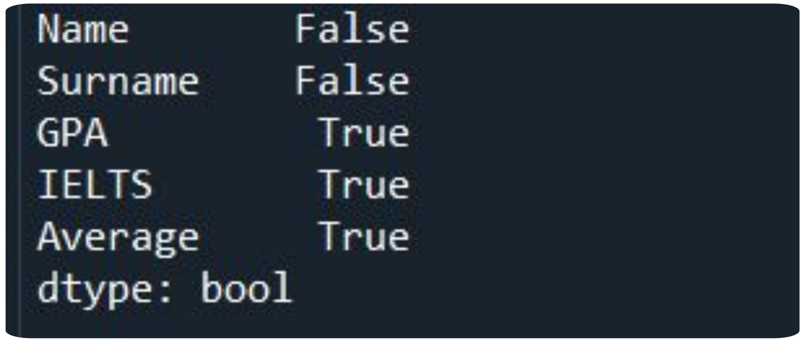
•The other calling of this class is “Not Missing at Random (NMAR)”. Here, the missing values are lost for the unknown reasons. One of the reasons for these values to be lost can be the refuse of the respondents. MNAR is a complex case, because the handling with this case is tougher than others. There is no way to drop or impute these values without introducing bias to the dataset, which can change the results and mislead us in the future.

•A good illustration of MAR could be the questionnaires in the workplace, when the employees or employers would not answer the questions regarding their salaries.

•Dealing with missing data

Continued …

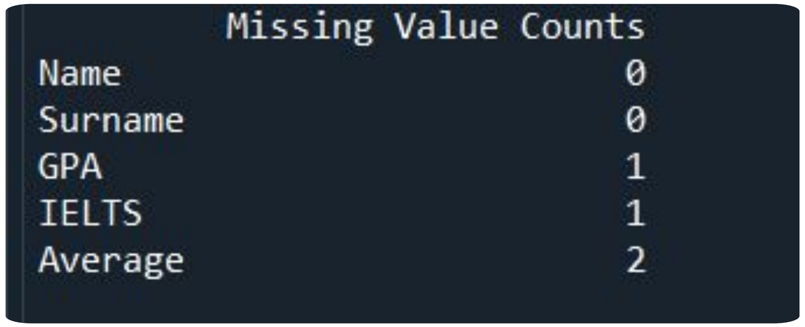
- The first method of missing data identification is:
- `df.isna().isany()`
- which returns boolean output (“True” or “False) for the columns. In case of being “True”, a column contains missing values. On the other hand, “False” shows nonexistence of missing values.



```
Name         False
Surname      False
GPA           True
IELTS         True
Average       True
dtype: bool
```

Continued ...

- The second method of missing data identification is:
- `df.isna().sum()`
- which returns the number of missing values in columns.



```
         Missing Value Counts
Name                        0
Surname                     0
GPA                         1
IELTS                       1
Average                     2
```

Continued …

•Easy way:

•Ignore tuples with missing values: This method is appropriate when the given dataset is large and several values are missed.

•Drop missing values: Only appropriate when the data is large

•Professional way:

•Despite being one method of handling with missing data, dropping has a very important disadvantage. Because of one missing value, entire data will be deleted, which is valuable during the solutions. That is why, instead of dropping, imputing (filling) the missing values is a better choice.

•Imputation:

•Imputation is the best strategy for handling missing values. Different methods for imputation have been presented, ranging from the simple to complex.

•Mean/Median/Mode:

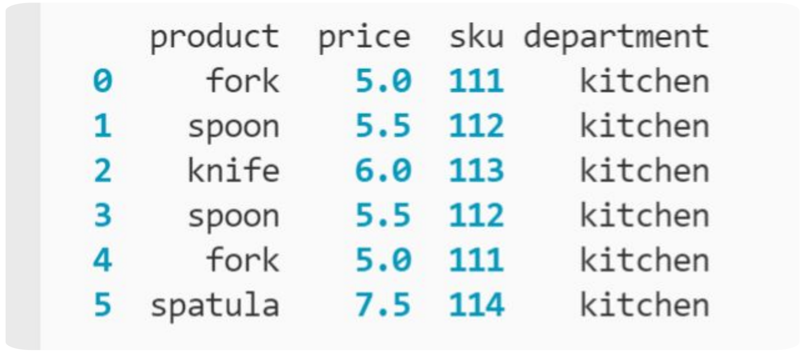
•One of the methods of imputation is using mean or median. The mean and median of the particular column should be calculated and then filled in place of missing data. For the categorical data, nonetheless, the mode function is used.

Removing Duplicates

- Removing duplicate values from your data set plays an important role in the cleansing process. Duplicate data takes up unnecessary storage space and slows down calculations at a minimum. At worst, duplicate data can skew analysis results and threaten the integrity of the data set.
- pandas is an open-source Python library that optimizes storage and manipulation of structured data. The framework also has built-in support for data cleansing operations, including removing duplicate rows and columns.
- .drop_duplicates methods are useful for the same.

How to Count the Number of Duplicated Rows in Pandas DataFrames

- Before we start removing duplicate rows, it's wise to get a better idea of where the duplicates are in our data set. Then we can decide how best to deal with them.



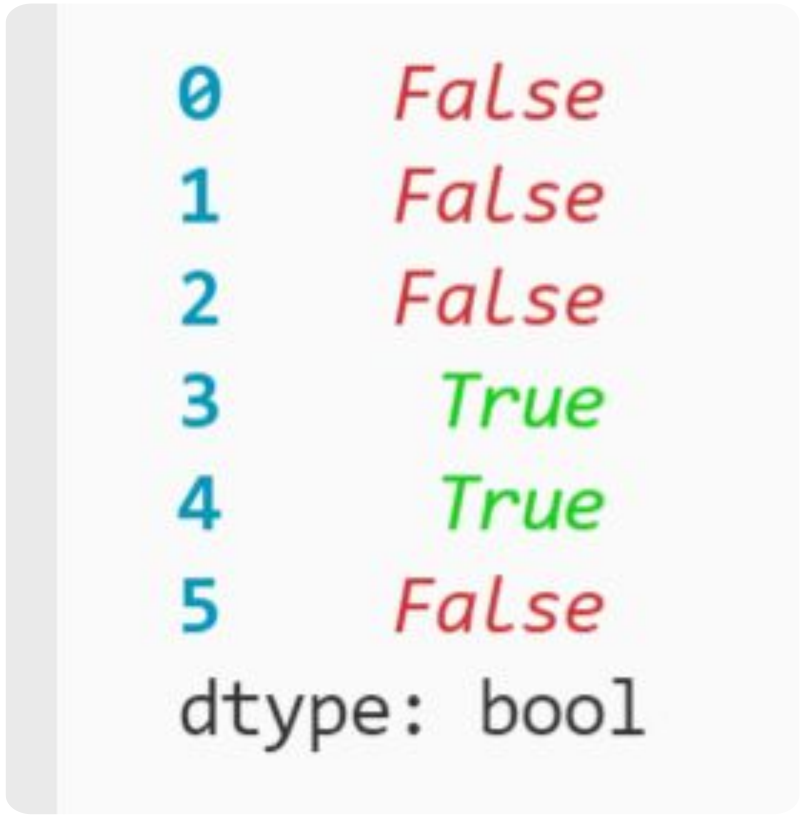
```
   product  price  sku department
0     fork    5.0  111    kitchen
1    spoon    5.5  112    kitchen
2    knife    6.0  113    kitchen
3    spoon    5.5  112    kitchen
4     fork    5.0  111    kitchen
5  spatula    7.5  114    kitchen
```

Continued ...

With the .duplicated method, we can identify which rows are duplicates:

kitch_prod_df.duplicated(). Here, we are calling

`.duplicated()` on our DataFrame **`kitch_prod_df`**. Once executed, **`.duplicated()`** returns Booleans organized into a pandas Series.



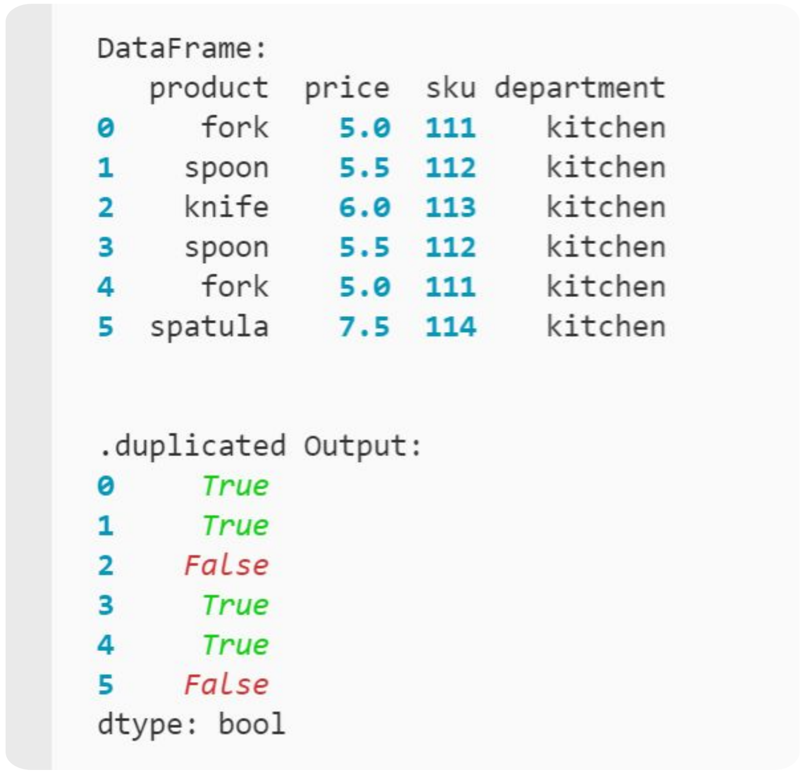
```
0    False
1    False
2    False
3     True
4     True
5    False
dtype: bool
```

Continued …

Pandas .duplicated Arguments

The Keep Argument

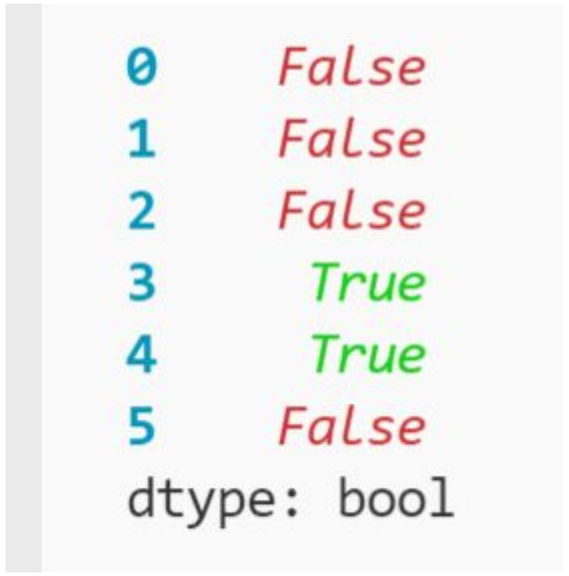
kitch_prod_df.duplicated(keep = False)



```
DataFrame:
   product  price  sku department
0     fork    5.0  111    kitchen
1    spoon    5.5  112    kitchen
2    knife    6.0  113    kitchen
3    spoon    5.5  112    kitchen
4     fork    5.0  111    kitchen
5  spatula    7.5  114    kitchen


.duplicated Output:
0     True
1     True
2    False
3     True
4     True
5    False
dtype: bool
```

Continued …

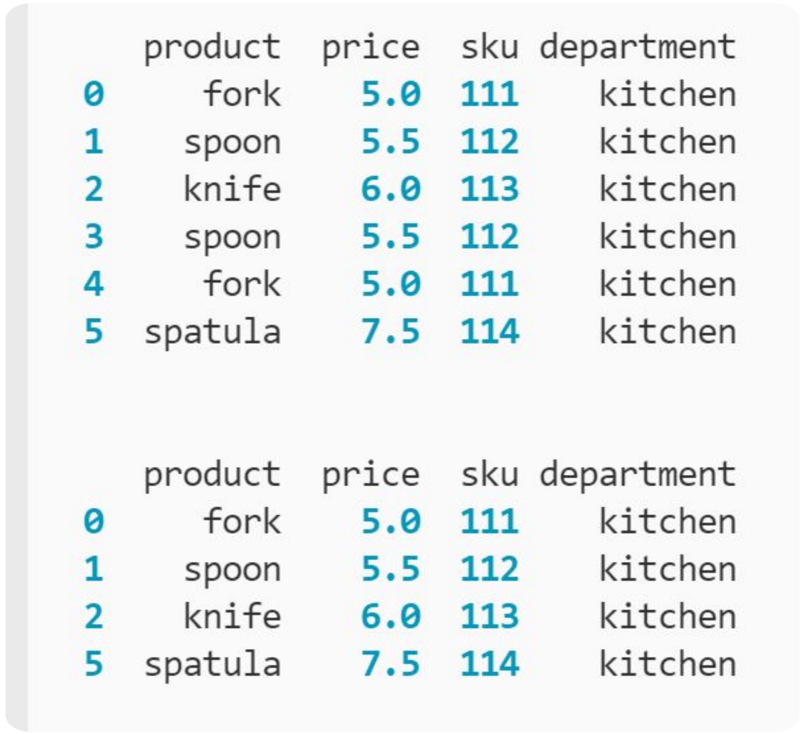
```
kitch_prod_df.duplicated(keep = 'first')
```



```
0    False
1    False
2    False
3     True
4     True
5    False
dtype: bool
```

How to Drop Duplicate Rows in Pandas DataFrames

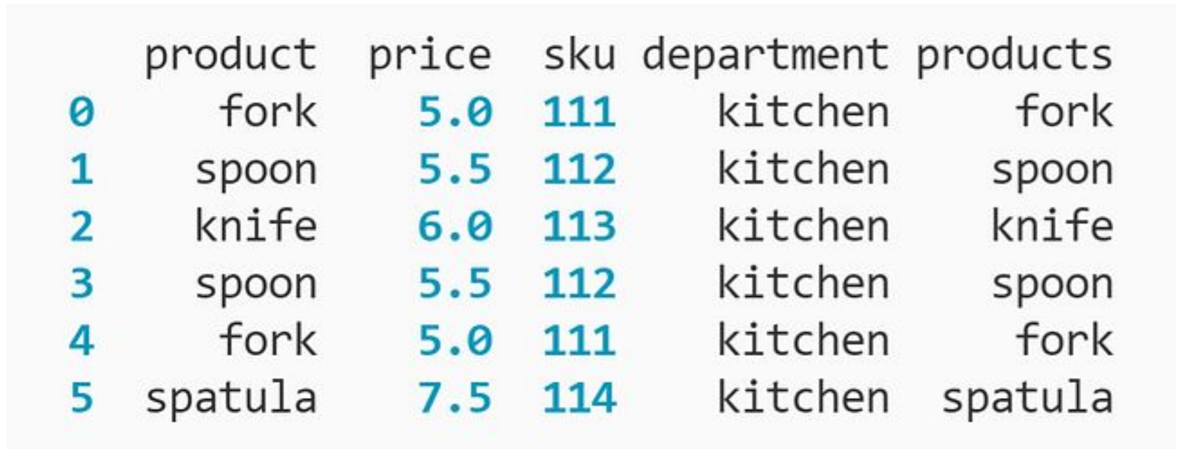
```
kitch_prod_df.drop_duplicates(inplace = True)
```



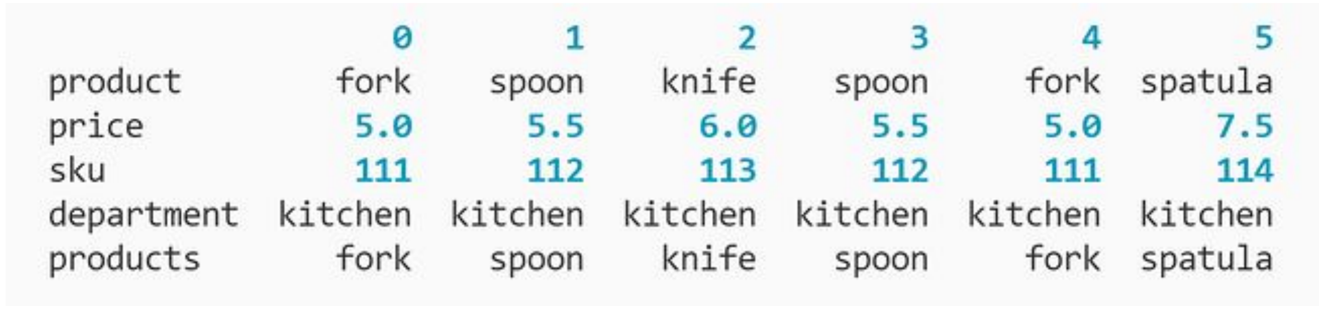
```
   product  price  sku department
0     fork    5.0  111    kitchen
1    spoon    5.5  112    kitchen
2    knife    6.0  113    kitchen
3    spoon    5.5  112    kitchen
4     fork    5.0  111    kitchen
5  spatula    7.5  114    kitchen


   product  price  sku department
0     fork    5.0  111    kitchen
1    spoon    5.5  112    kitchen
2    knife    6.0  113    kitchen
5  spatula    7.5  114    kitchen
```

How to Drop Duplicate Columns in Pandas DataFrames

- The .T property allows us to invert the axis of our DataFrame. You can see the output of printing **kitch_prod_df.T** to the terminal below.



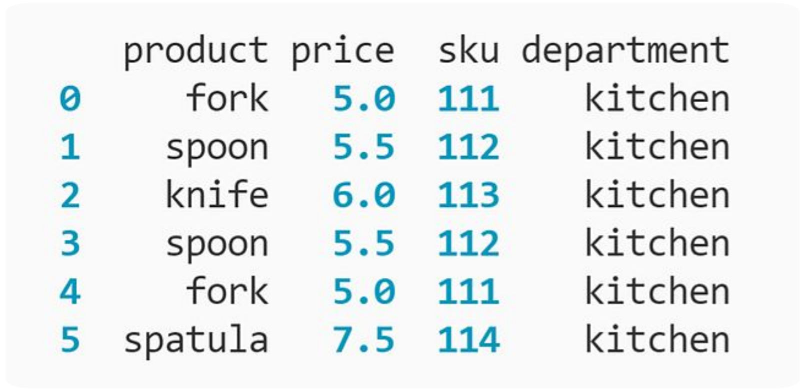
```
   product  price  sku department products
0     fork    5.0  111    kitchen     fork
1    spoon    5.5  112    kitchen    spoon
2    knife    6.0  113    kitchen    knife
3    spoon    5.5  112    kitchen    spoon
4     fork    5.0  111    kitchen     fork
5  spatula    7.5  114    kitchen  spatula
```



```
                  0        1        2        3        4        5
product        fork    spoon    knife    spoon     fork  spatula
price           5.0      5.5      6.0      5.5      5.0      7.5
sku             111      112      113      112      111      114
department  kitchen  kitchen  kitchen  kitchen  kitchen  kitchen
products       fork    spoon    knife    spoon     fork  spatula
```

Continued …

no_dup_columns = kitch_prod_df.T.drop_duplicates().T



```
   product  price  sku  department
0     fork    5.0  111     kitchen
1    spoon    5.5  112     kitchen
2    knife    6.0  113     kitchen
3    spoon    5.5  112     kitchen
4     fork    5.0  111     kitchen
5  spatula    7.5  114     kitchen
```

Slicing and Dicing

- **Slicing:**

```
# importing pandas package
import pandas as pd

# create a Pandas DataFrame
df = pd.DataFrame([['a','b'],['c','d'],['e','f'],['g','h']],
columns=['col1','col2'])

print("DataFrame:")
print(df)

# Access the elements using slicing indexer
result = df.iloc[1:3]
print("Output:")
print(result)
```

```
DataFrame:
  col1 col2
0    a    b
1    c    d
2    e    f
3    g    h

Output:
 col1 col2
1   c    d
2   e    f
```

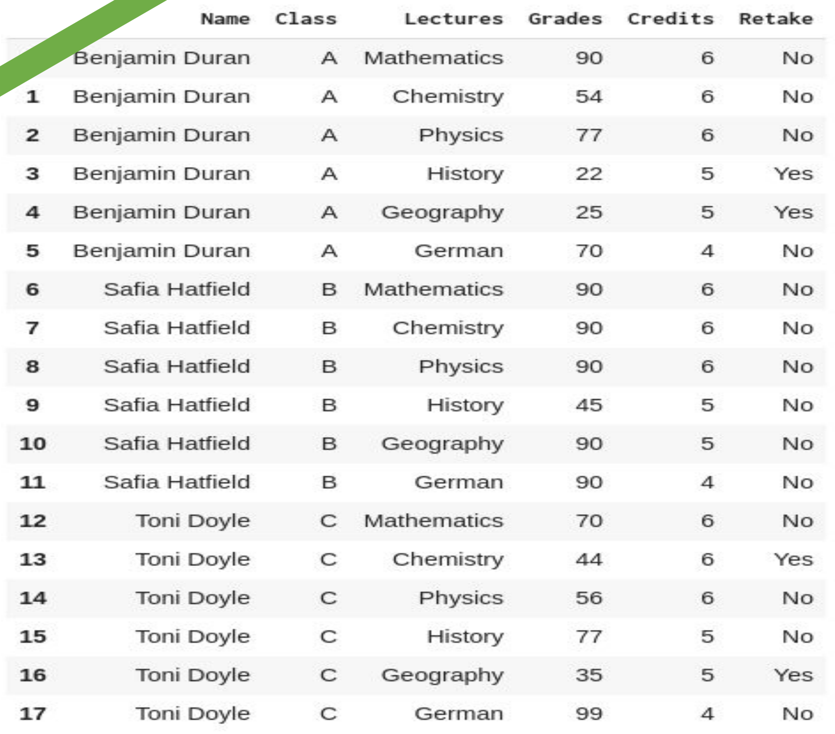
Continued …

- # importing pandas package
- import pandas as pd
- # create a Pandas DataFrame
- df = pd.DataFrame([['a','b'],['c','d'],['e','f'],['g','h’]])
- columns=['col1','col2’])
- print("DataFrame:")
- print(df)
- # Access the elements using slicing indexer
- result = df.iloc[-4:-1]
- print("Output:")
- print(result)

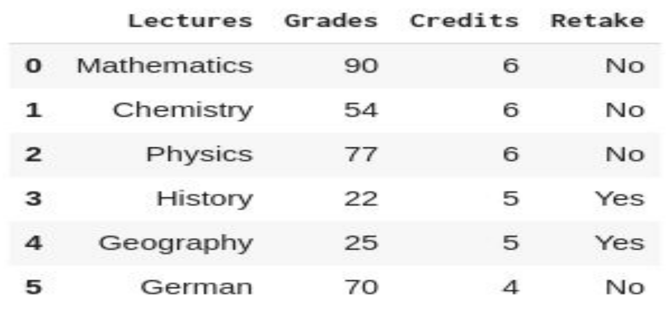
```
•DataFrame:
• col1     col2
•0   a        b
•1   c        d
•2   e        f
•3   g        h

•Output: col1     col2
•          0  a         b
•          1  c         d
•          2  e         f
```

- Grades =

- Report_Card.loc[(Report_Card["Name"] == "Benjamin Duran"), ["Lectures","Grades","Credits","Retake"]]



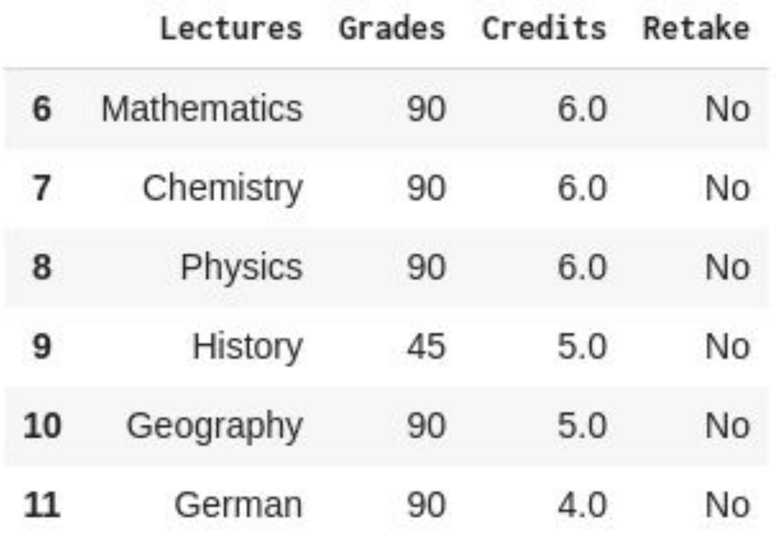
	Name	Class	Lectures	Grades	Credits	Retake
	Benjamin Duran	A	Mathematics	90	6	No
1	Benjamin Duran	A	Chemistry	54	6	No
2	Benjamin Duran	A	Physics	77	6	No
3	Benjamin Duran	A	History	22	5	Yes
4	Benjamin Duran	A	Geography	25	5	Yes
5	Benjamin Duran	A	German	70	4	No
6	Safia Hatfield	B	Mathematics	90	6	No
7	Safia Hatfield	B	Chemistry	90	6	No
8	Safia Hatfield	B	Physics	90	6	No
9	Safia Hatfield	B	History	45	5	No
10	Safia Hatfield	B	Geography	90	5	No
11	Safia Hatfield	B	German	90	4	No
12	Toni Doyle	C	Mathematics	70	6	No
13	Toni Doyle	C	Chemistry	44	6	Yes
14	Toni Doyle	C	Physics	56	6	No
15	Toni Doyle	C	History	77	5	No
16	Toni Doyle	C	Geography	35	5	Yes
17	Toni Doyle	C	German	99	4	No



	Lectures	Grades	Credits	Retake
0	Mathematics	90	6	No
1	Chemistry	54	6	No
2	Physics	77	6	No
3	History	22	5	Yes
4	Geography	25	5	Yes
5	German	70	4	No

Continued …

- Sofia_Grades = Report_Card.iloc[6:12,2:]
- or else:
- Sofia_Grades = Report_Card.iloc[[6,7,8,9,10,11],[2,3,4,5]]
- Both of the above code snippets result in the following DataFrame:



	Lectures	Grades	Credits	Retake
6	Mathematics	90	6.0	No
7	Chemistry	90	6.0	No
8	Physics	90	6.0	No
9	History	45	5.0	No
10	Geography	90	5.0	No
11	German	90	4.0	No

Filtering

- import pandas as pd

- # dictionary of lists
- d = {'Car': ['BMW', 'Lexus', 'Audi', 'Mercedes', 'Jaguar', 'Bentley'],'Date_of_Purchase': ['2021-07-10', '2021-08-12', '2021-06-17', '2021-03-16', '2021-05-19', '2021-08-22']
- }

- # creating dataframe from the above dictionary of lists
- dataFrame = pd.DataFrame(d)
- print"DataFrame...\n",dataFrame

- # fetch cars purchased after 15th July 2021
- resDF = dataFrame.loc[dataFrame["Date_of_Purchase"] > "2021-07-15"]

- # print filtered data frame
- print"\nCars purchased after 15th July 2021: \n",resD

```
DataFrame...
                   Car
Date_of_Purchase
     0     BMW      2021-07-10
     1    Lexus     2021-08-12
     2     Audi     2021-06-17
     3  Mercedes     2021-03-16
     4   Jaguar     2021-05-19
     5   Bentley     2021-08-22

   Cars purchased after 15th July
              2021:
          Car   Date_of_Purchase
     1   Lexus      2021-08-12
     5  Bentley      2021-08-22
```

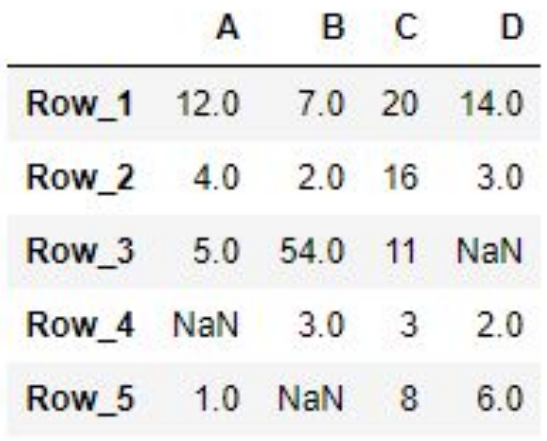
Concatenating

```
• import pandas as pd

• one = pd.DataFrame({
•     'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
•     'subject_id':['sub1','sub2','sub4','sub6','sub5'],
•   'Marks_scored':[98,90,87,69,78]},
•   index=[1,2,3,4,5])

• two = pd.DataFrame({
•     'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
•     'subject_id':['sub2','sub4','sub3','sub6','sub5'],
•   'Marks_scored':[89,80,79,97,88]},
•   index=[1,2,3,4,5])
• print (pd.concat([one,two]))
```

```
•O/P:
•
•     Name    subject_id     Marks_scored
•1    Alex      sub1             98
•2     Amy      sub2             90
•3   Allen      sub4             87
•4   Alice      sub6             69
•5  Ayoung      sub5             78
•1   Billy      sub2             89
•2   Brian      sub4             80
•3    Bran      sub3             79
•4   Bryce      sub6             97
•5   Betty      sub5             88
```

Transforming

```
import pandas as pd

# Creating the DataFrame
df = pd.DataFrame({"A":[12, 4, 5, None, 1],
                   "B":[7, 2, 54, 3, None],
                   "C":[20, 16, 11, 3, 8],
                   "D":[14, 3, None, 2, 6]})

# Create the index
index_ = ['Row_1', 'Row_2', 'Row_3', 'Row_4',
'Row_5']

# Set the index
df.index = index_

# Print the DataFrame
print(df)
```

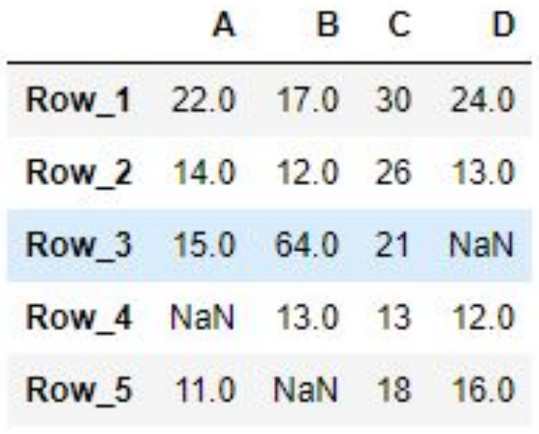


	A	B	C	D
Row_1	12.0	7.0	20	14.0
Row_2	4.0	2.0	16	3.0
Row_3	5.0	54.0	11	NaN
Row_4	NaN	3.0	3	2.0
Row_5	1.0	NaN	8	6.0

Continued …

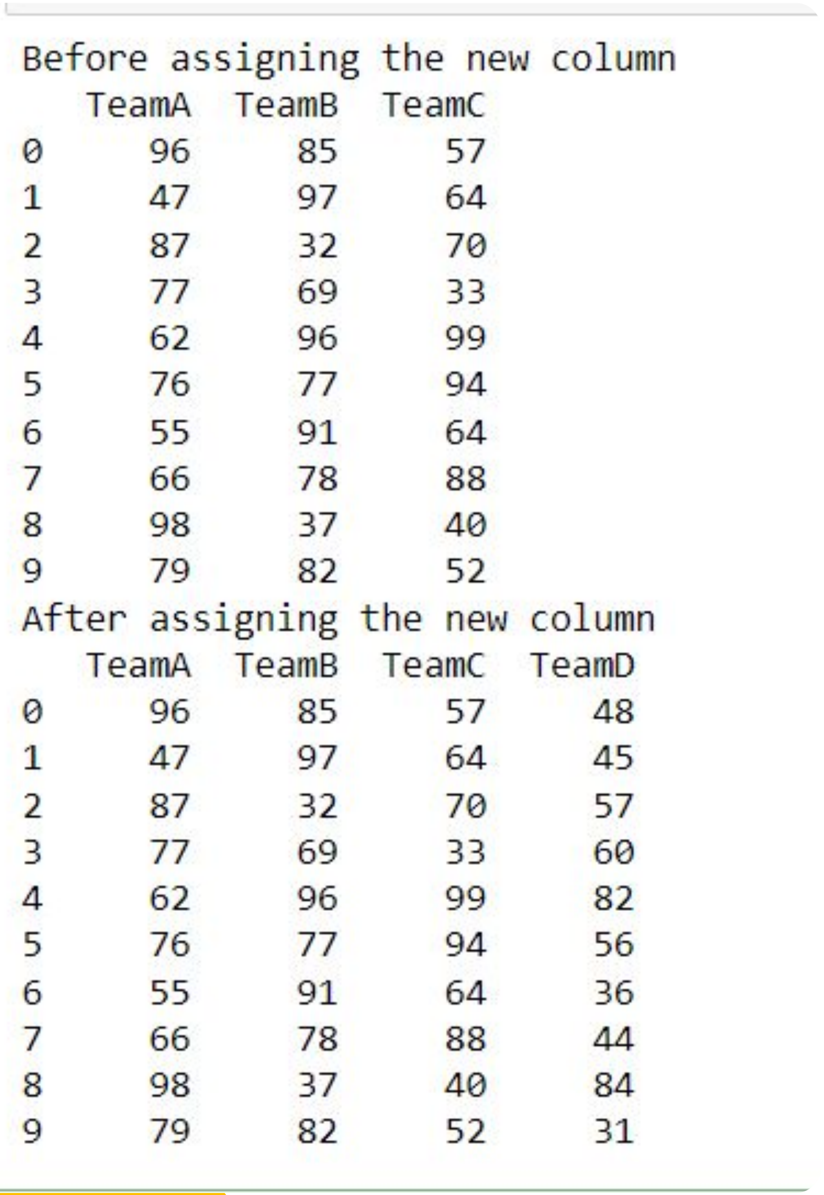
Now we will use DataFrame.transform() function to add 10 to each element of the dataframe.

```
# add 10 to each element of the dataframe
result = df.transform(func = lambda x : x + 10)

# Print the result
print(result)
```



	A	B	C	D
Row_1	22.0	17.0	30	24.0
Row_2	14.0	12.0	26	13.0
Row_3	15.0	64.0	21	NaN
Row_4	NaN	13.0	13	12.0
Row_5	11.0	NaN	18	16.0

Adding a New Variable

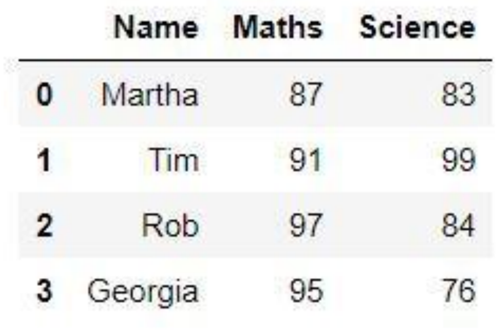


```
Before assigning the new column
   TeamA  TeamB  TeamC
0     96     85     57
1     47     97     64
2     87     32     70
3     77     69     33
4     62     96     99
5     76     77     94
6     55     91     64
7     66     78     88
8     98     37     40
9     79     82     52
After assigning the new column
   TeamA  TeamB  TeamC  TeamD
0     96     85     57     48
1     47     97     64     45
2     87     32     70     57
3     77     69     33     60
4     62     96     99     82
5     76     77     94     56
6     55     91     64     36
7     66     78     88     44
8     98     37     40     84
9     79     82     52     31
```

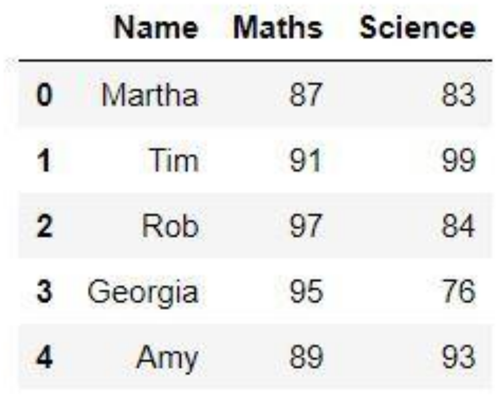
```
# import packages
import numpy as np
import pandas as pd

# setting a seed
# creating a dataframe
df = pd.DataFrame({'TeamA': np.random.randint(30, 100, 10),
                   'TeamB': np.random.randint(30, 100, 10),
                   'TeamC': np.random.randint(30, 100, 10)})
print('Before assigning the new column')

print(df)
# using assign() method to add a new column
scores = np.random.randint(30, 100, 10)

df2 = df.assign(TeamD=scores)

print('After assigning the new column')

print(df2)
```

Adding a New Cases (Row) to data frames:

```
import pandas as pd
from numpy.random import randint

dict = {'Name':['Martha', 'Tim', 'Rob', 'Georgia'],
        'Maths':[87, 91, 97, 95],
        'Science':[83, 99, 84, 76]
       }

df = pd.DataFrame(dict)

display(df)

df.loc[len(df.index)] = ['Amy', 89, 93]

display(df)
```



	Name	Maths	Science
0	Martha	87	83
1	Tim	91	99
2	Rob	97	84
3	Georgia	95	76



	Name	Maths	Science
0	Martha	87	83
1	Tim	91	99
2	Rob	97	84
3	Georgia	95	76
4	Amy	89	93

Removing data

- dataFrame = pd.DataFrame([[10, 15], [20, 25], [30, 35], [40, 45]],index=['w', 'x', 'y', 'z'],
- columns=['a', 'b'])

- print(dataFrame)

- print(dataFrame.drop('w'))
- print(dataFrame.drop(‘b’,axis=1))

```
•O/P:

•DataFrame...
•     a   b
•w  10  15
•x   20  25
•y   30  35
•z   40  45
•DataFrame after deleting a row...
•    a   b
•x  20  25
•y  30  35
•z  40  45

•DataFrame after deleting a column...

•      a
•w   10
•x    20
•y    30
•z    40
```

Sorting

- import pandas as pd

```
data = {
  "age": [50, 40, 30, 40, 20, 10, 30],
  "qualified": [True, False, False, False, False, True, True]
}
df = pd.DataFrame(data)
```

- print(df)

```
newdf = df.sort_values(by='age')
```

- O/P

```
   age  qualified
0   50       True
1   40      False
2   30      False
3   40      False
4   20      False
5   10       True
6   30       True

   age  qualified
5   10       True
4   20      False
2   30      False
6   30       True
1   40      False
3   40      False
0   50       True
```

Shuffling

- import random
-
- mylist = ["apple", "banana", "cherry"]
  O/P : ['cherry', 'banana', 'apple']
- random.shuffle(mylist)
- print(mylist)

Aggregating data

- import pandas as pd
- data = {
- "x": [50, 40, 30],
- "y": [300, 1112, 42]
- }
- df = pd.DataFrame(data)
- x = df.aggregate(["sum"])
- print(x)

```
       x     y
sum  120  1454
```

Handling outliers

1. Imports pandas and numpy libraries.
2. Creates your own data frame using pandas.
3.Outliers handling by dropping them.
4. Outliers handling using Boolean marking.
5. Outliers handling using Rescaling of features.

Continued ...

- Step 1 - Import the library
- import numpy as np
  import pandas as pd

- Step 2 - Creating DataFrame
- We have first created an empty dataframe named farm then added features and values to it. We can clearly see that in feature Rooms the value 100 is an outlier.
- farm = pd.DataFrame()
- farm['Price'] = [632541, 425618, 356471, 7412512]
- farm['Rooms'] = [2, 5, 3, 100]
- farm['Square_Feet'] = [1600, 2850, 1780, 90000]
- print(farm)

```
   Price      Rooms    quare_Feet
0  632541       2        1600
1  425618       5        2850
2  356471       3        1780
3  7412512     100       90000
```

Continued …

- Step 3: Outliers handling by dropping them.
- h = farm[farm['Rooms'] < 20] print(h)

```
     Price     Rooms    Square_Feet
0  632541      2          1600
1  425618      5          2850
2  356471      3          1780
```

Continued …

- Step 4:Marking the Outliers
- farm['Outlier'] = np.where(farm['Rooms'] < 20, 0, 1) print(farm)

```
     Price   Rooms   Square_Feet  Outlier
0   632541      2       1600            0
1   425618      5       2850            0
2   356471      3       1780            0
3  7412512    100      90000            1
```

Continued …

- Step 5: Rescaling the data
- farm['Log_Of_Square_Feet'] = [np.log(x) for x in farm['Square_Feet']] print(farm)

```
•      Price  Rooms  Square_Feet  Outlier  Log_Of_Square_Feet
• 0   632541     2        1600          0            7.377759
• 1   425618     5        2850          0            7.955074
• 2   356471     3        1780          0            7.484369
• 3  7412512   100       90000          1           11.407565
```

Data Wrangling

- Data wrangling is the process of removing errors and combining complex data sets to make them more accessible and easier to analyze. Due to the rapid expansion of the amount of data and data sources available today, storing and organizing large quantities of data for analysis is becoming increasingly necessary.
- A data wrangling process, also known as a data munging process, consists of reorganizing, transforming and mapping data from one "raw" form into another in order to make it more usable and valuable for a variety of downstream uses including analytics.
- Data wrangling can be defined as the process of cleaning, organizing, and transforming raw data into the desired format for analysts to use for prompt decision-making. Also known as data cleaning or data munging, data wrangling enables businesses to tackle more complex data in less time, produce more accurate results, and make better decisions. The exact methods vary from project to project depending upon your data and the goal you are trying to achieve. More and more organizations are increasingly relying on data wrangling tools to make data ready for downstream analytics.
- Data wrangling software has become such an indispensable part of data processing. The primary importance of using data wrangling tools can be described as:
- Making raw data usable. Accurately wrangled data guarantees that quality data is entered into the downstream analysis.
- Getting all data from various sources into a centralized location so it can be used.
- Piecing together raw data according to the required format and understanding the business context of data
- Automated data integration tools are used as data wrangling techniques that clean and convert source data into a standard format that can be used repeatedly according to end requirements. Businesses use this standardized data to perform crucial, cross-data set analytics.
- Cleansing the data from the noise or flawed, missing elements
- Data wrangling acts as a preparation stage for the data mining process, which involves gathering data and making sense of it.
- Helping business users make concrete, timely decisions
- Data wrangling software typically performs six iterative steps of Discovering, Structuring, Cleaning, Enriching, Validating, and Publishing data before it is ready for analytics.

Continued …

Benefits of Data Wrangling

- Data wrangling helps to improve data usability as it converts data into a compatible format for the end system.
- It helps to quickly build data flows within an intuitive user interface and easily schedule and automate the data-flow process.
- Integrates various types of information and their sources (like databases, web services, files, etc.)
- Help users to process very large volumes of data easily and easily share data-flow techniques.
- Some examples of basic data munging tools are:
- Spreadsheets / Excel Power Query - It is the most basic manual data wrangling tool
- OpenRefine - An automated data cleaning tool that requires programming skills
- Tabula – It is a tool suited for all data types
- Google DataPrep – It is a data service that explores, cleans, and prepares data
- Data wrangler – It is a data cleaning and transforming tool

Data Normalization

- Normalization is the process of organizing data from a database. This includes processes like creating tables and establishing relationships between those tables according to the rules designed to protect the data as well as to make the database more flexible by eliminating the redundancy and inconsistency present. Data normalization is the method of organizing data to appear similar across all records and fields. Performing so always results in getting higher quality data. This process basically includes eliminating unstructured data and duplicates in order to ensure logical data storage.
- When data normalization is performed correctly a higher value of insights are generated. In machine learning, some feature values at times differ from others multiple times. The features with higher values will always dominate the learning process. However, it does not mean that those variables are more important to predict the outcome of the model. Data normalization transforms the multiscaled data all to the same scale. After normalization, all variables have a similar weightage on the model, hence improving the stability and performance of the learning algorithm. Normalization gives equal importance to each variable so that no single variable drives the model performance. It also prevents any issues created from database modifications such as insertions, deletions, and updates.
- For businesses to achieve further heights needs to regularly perform data normalization. It is one of the most important things that can be done to get rid of errors that make an analysis of data a complicated and difficult task. With normalization, an organization can make the most of its data as well as invest in data gathering at a greater, more efficient level.

Continued …

•Methods for Data Normalization

• There are several methods for performing Data Normalization, few of the most popular techniques are as follows :

- The min-max normalization is the simplest of all methods as it converts floating-point feature values from their natural range into a standard range, usually between 0 and 1. It is a good choice when one knows the approximate upper and lower bounds on the data with few or no outliers and the data are approximately uniformly distributed across that range.
- Decimal place normalization can be done with data tables having numerical data types. By default, the algorithm places two digits after the decimal for normal comma-separated numbers. One can decide on how many decimals are required to scale this throughout the table.
- Z-score normalization is a methodology of normalizing the data and hence helps avoid the issue of outliers in the data. Here, μ is considered the mean value of the feature and σ is the standard deviation from the data points. If a value comes to be equal to the mean of all the values present, only then will it be normalized to 0. If it is below the mean value, it will be considered to be a negative number, and if it is above the mean value, it will be termed as a positive number.

Thank You